Sistema digitalen diseinu-hastapenak

Oinarrizko kontzeptuak eta adibideak

Udako Euskal Unibertsitatea

SISTEMA DIGITALEN DISEINU-HASTAPENAK

Oinarrizko kontzeptuak eta adibideak

SISTEMA DIGITALEN DISEINU-HASTAPENAK

Oinarrizko kontzeptuak eta adibideak

Olatz Arbelaitz Gallego
Olatz Arregi Uriarte
Agustin Arruabarrena Frutos
Izaskun Etxeberria Uztarroz
Amaya Ibarra Lasa
Txelo Ruiz Vazquez

Udako Euskal **Unibertsitatea**
Bilbo, 2005

«Liburu hau Hezkuntza, Unibertsitate eta
Ikerketa Sailaren laguntzaz argitaratu da»

EHUko Euskara Errektoreordetzaren diru-laguntzaz prestatu
dute egileek liburu honen oinarriko testua

ISBN: 84-8438-069-6
Lege-gordailua: BI-1661-05

Inprimategia: RGM, Bilbo
Azalaren diseinua: Iñigo Ordozgoiti
Hizkuntza-zuzenketen arduraduna: Ander Altuna Gabiola

Banatzaileak: UEU. Erribera 14, 1. D BILBO telf. 946790546 Faxa. 944793039
Helbide elektronikoa: argitalpenak@ueu.org
www.ueu.org

Zabaltzen: Igerabide, 88 DONOSTIA

AURKIBIDEA

HITZAURREA

Ohikoak eta ugariak dira sistema digitalei buruzko testuliburuak auzoko hizkuntzetan, zer esanik ez ingelesez, baina baita gaztelaniaz eta frantsesez ere. Ez, ordea, euskaraz. Hala ere, badira urteak gai horiek irakasten ditugula euskaraz EHUko Informatika Fakultatean, lehen eta bigarren mailan. Hori dela eta, uste izan dugu bazela garaia metatutako eskarmentua eta materiala liburu batean biltzeko.

Sistema digitalen oinarrizko kontzeptu eta diseinu-teknikak aurkezten dira liburuan. Ez dago aukera askorik originaltasunerako gai horiek aurkeztu behar direnean, informatika zein elektronika arloetan asko landu diren gaiak direlako, baina liburua ez da beste baten itzulpena edo kopia (handik eta hemendik edan badugu ere, jakina), baizik eta gaiak irakasten lortutako esperientziaren fruitua. Liburua, beraz, originala da.

Liburuaren egitura eta edukia

Sistema digitalen oinarrizko kontzeptuak azaltzeko, ibilbide bikoitza hartu dugu liburuan. Hala, kapituluak bi partetan banatu ditugu.

Lehenengoan, gaiari dagozkion kontzeptuak, definizioak, gailuak eta abar azaldu ditugu. Bigarrenean, hainbat ariketa ebatzi ditugu, zehatz-mehatz, kontzeptu eta gailu horiek nola aplikatzen diren ahalik eta argien uzteko. Guztira, ia 60 ariketa ebazten dira xehetasun osoz liburuan zehar, sinpleenetatik konplexuenetara. Denek batera, oinarrizko sistema digitalen osagaien, diseinuaren eta funtzionamenduaren argazki zabal eta betea osatzen dute. Hala, erabiltzaileak bi modutara erabil dezake liburua: kontzeptu teorikoak aztertzeko, edo proposatzen diren ariketak egiteko eta bere ebazpenak ematen diren ebazpenekin erkatzeko. Liburu erabilgarria egitea izan da gure asmoa.

Ohikoa da bide bat baino gehiago izatea sistema digitalak sortzeko. Hori dela eta, kapitulu bakoitzeko ariketetarako proposatzen ditugunak “gure” ebazpenak dira, eta ez dute zertan bakarrak izanik. Are gehiago, litekeena da irakurleak soluzio egokiagoak aurkitzea. Dena den, saiatu gara hartu ditugun erabaki guztiak argi eta garbi azaltzen, dudarik txikiena ere uxatzeko.

Oinarrizko kontzeptuak lantzen dira liburuan, unibertsitateko lehen mailako ikasgai batean irakasten ditugunak; ezinbestean, beraz, liburuan bertan azaltzen dira erabiltzen diren kontzeptu guztiak. Honela banatu ditugu aukeratutako gaiak:

- 1. kapituluan, aljebra boolearraren kontzeptu nagusiak, funtzio logikoen adierazpenak eta haien minimizazioa azaltzen dira. Matematikako oinarrizko kontzeptuak dira kapitulu horretan ageri direnak.
- 2. kapituluan, sistema digitalen behe-mailako osagaiak azaltzen dira, ate logikoak: AND, OR, NOT, NAND, NOR, XOR... bai eta nola erabili ate horiek edozein sistema digital eraikitzeko ere.
- 3. kapituluan, zirkuitu edo bloke konbinazional erabilienak azaltzen dira: multiplexoreak, deskodegailuak, batugailuak... Maiz erabiltzen diren funtzio logikoak egiten dituzte bloke konbinazional horiek, eta edozein sistema digitalen oinarria dira.
- Ildo berean, 4. kapituluan, oinarrizko zirkuitu sekuentzial sinkrono erabilienak aztertzen dira: biegonkorrak eta erregistroak; hots, memoria duten zirkuituak.

  3. eta 4. kapituluetan azaltzen diren bloke konbinazionalek eta sekuentzialek osatzen dute (memoriekin batera) sistema digital guztien hezurdura.
- 5. kapituluan memoriak azaltzen dira: RAM, ROM... Memoria mota asko dago eta teknologia oso desberdinak erabiltzen dira memoria horiek sortzeko. Hori dela eta, oinarrizko memorien funtzionamendu logikoa azaltzen da, teknologiako xehetasunak aparte utzita.

  Ezaguna denez, datuak eta aginduak gordetzeko erabiltzen dira memoriak sistema digitaletan eta konputagailuen osagai nagusietako bat dira.
- 6. kapituluan, sistema digitalen diseinu-metodologia baten hastapenak eskaintzen dira. Hor ageri dira kontrol-unitatea, prozesu-unitatea, kontrol-seinaleak eta abar. Sistema digitalen diseinu aurreratua liburu honen helburutik at badago ere, hainbat adibide erabiltzen dira diseinuaren nondik norakoa argi eta garbi azaltzeko.
- 7. kapituluan batzen dira liburuan zehar azaldutako kontzeptu eta teknika guztiak, eta, horretarako, prozesadore sinple bat diseinatzen da: BIRD prozesadorea. Helburu didaktikoa duen prozesadorea bada ere, prozesadoreetan ohikoak diren kontzeptu asko eta asko lantzen dira.

Hiru eranskinekin osatzen da liburua. Hiruretan, sistema digitalen diseinuarekin zerikusia duen gai jakin baten laburpena egiten da.

- Lehenengoan, sistema digitaletan zenbakiak adierazteko gehien erabiltzen diren adierazpide-sistemak laburbiltzen dira: kodeketa bitarra, 2rako osagarria, zeinu/magnitudea eta koma higikorreko formatuak.
- Bigarrenean, sistema digitalen teknologiari buruzko oinarrizko kontzeptuak ageri dira; ezin dugu, baina, elektronika digitalaren teknologia azaldu liburuan, eta, horregatik, oso laburpen xumea egiten da.
- Hirugarrenean, sistema digitalak definitzeko asko erabiltzen den lengoaia bat deskribatzen da: VHDL. Berriro ere, ezin dira azaldu lengoaia horren xehetasun guztiak, baina uste dugu interesgarria izan daitekeela aztertzea nola definitzen diren liburuan zehar azaldu ditugun zirkuitu nagusiak VHDL erabiliz.

Liburuan ez dira jorratzen mikroprozesadoreak eta horien bidez egindako sistemen diseinua, ezta VLSI mailako zirkuituen sintesia, analisia eta abar ere.

Hau bezalako testuetan ohikoa den moduan, prozesadore sinple baten diseinua lantzen dugu azken kapituluan, liburuan azaldu ditugun kontzeptu, gailu eta teknika guztiak konplexua den “proiektu” batean biltzeko, eta, bidenabar, oinarrizko konputagailuen funtzionamendua azaltzeko. BIRD prozesadorea erabili dugu lantegi horretarako. Izan ere, prozesadore bera erabiltzen dute gure ikasleek beste ikasgai batean, non oinarrizko prozesadoreen funtzionamendua eta programen exekuzioa aztertzen den zehatz-mehatz. Hori guztia beste liburu batean ageri da: *Makina Hizkuntza: Oinarrizko konputagailu baten egitura, agindu-multzoa eta programazioa*. Nahi izanez gero, www.sc.ehu.es/acweb/BIRD.html helbidean eskura daiteke BIRD prozesadorearen simuladore bat (PC/windows).

Azken aipamen bat. Irudi, adierazpen aljebraiko, taula, kronograma eta abar asko ageri dira ariketetan. Hainbat aldiz errepasatu baditugu ere, litekeena da oraindik akatsen bat geratzea, hor nonbait ezkutatuta. Irakurleak, beraz, adi egon beharko du, balizko akats horietan tupust ez egiteko.

Eskerrak

Esan dugun bezala, liburua hainbat urtetako irakaskuntzaren ondorioa da, eta irakasle askoren fruitua. Ez daude guztiak egileen zerrendan. Eskerrak eman nahi dizkiegu bereziki Edurne Larraza irakasleari, eta azken urteetan, gurekin batera, ikasgai hau irakatsi duten irakasleei, haiek ere parte hartu dutelako liburu honen edukia finkatzen.

Azkenik, egileek biziki eskertzen diote UPV/EHUko Euskara Errektoreordetzari "Euskarazko ikasmaterialgintza sustatzeko diru-laguntzen 2003.eko deialdia"-ren barruan emandako diru-laguntza, zeinari esker liburu honen oinarrian dagoen materiala prestatu baitute.

Egileak

Donostia, 2005eko ekaina

P.D.: Egileek biziki eskertuko dituzte irakurleak aurkitutako akatsei buruzko informazioa, testuari buruzko iruzkinak, iradokizunak eta abar, helbide elektroniko honetan: txelo.ruiz@ehu.es.

SISTEMA DIGITALAK: SARRERA

Sistema digitalak ditugu aztergai liburu honetan. Hasi baino lehen, beraz, merezi du azaltzea zer diren sistema digitalak, zer dagoen horien oinarrian, eta zergatik erabiltzen diren gero eta aplikazio gehiagotan. Eta horixe da, hain zuzen ere, sarrera labur honetan egingo duguna.

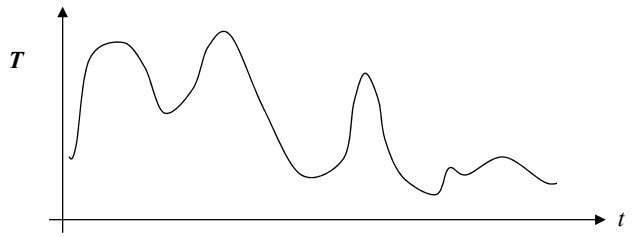
Hasteko, seinale analogikoak eta digitalak bereiziko ditugu, eta justifikatuko dugu zergatik erabiltzen diren seinale digital bitarrak sistema digitaletan. Gero, seinale digitalen ezaugarriak aipatuko ditugu. Azkenik, sistema digitalak diseinatzeko prozesuaren urrats nagusiak azalduko ditugu: analisia eta sintesia.

SEINALE ANALOGIKOAK ETA DIGITALAK

Naturan gertatzen diren fenomenoei magnitude fisikoak esleitzen zaizkie, hala nola, tenperatura, presioa, abiadura, pisua...; magnitude horien balioak denboran zehar aldatzen dira. Magnitude fisikoak neurtzeko, mota askotako neurgailuak erabiltzen ditugu; adibidez, erlojuak denbora neurtzeko, edo termometroak tenperatura neurtzeko.

Magnitudeak neurtzean, seinaleak lortzen dira. Hortaz, seinale batek adierazten du nola aldatzen den magnitude bat, beste baten arabera; esaterako, nola aldatzen den toki bateko tenperatura egunean zehar (denboraren mende, alegia). Laburrean, magnitude fisikoei buruzko informazioa ematen digute seinaleek.

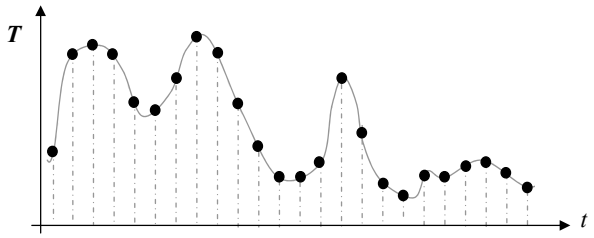
Naturako magnitude fisiko gehienak analogikoak dira, dagozkien seinaleak jarraituak baitira matematikaren ikuspuntutik, hau da, edozein balio har dezakete tarte jakin baten barruan, 1. irudian ageri den legez. Tenperaturaren kasuan, esaterako, 20 ºC-tik 30 ºC-ra igarotzeko, tarteko balio posible guztietatik pasatzen da.



1. irudia. Seinale analogiko bat.

Jarrai dezagun bi magnitude horiekin: tenperatura eta denbora; eta demagun jakin nahi dugula nola aldatzen den gela bateko tenperatura egunean zehar.

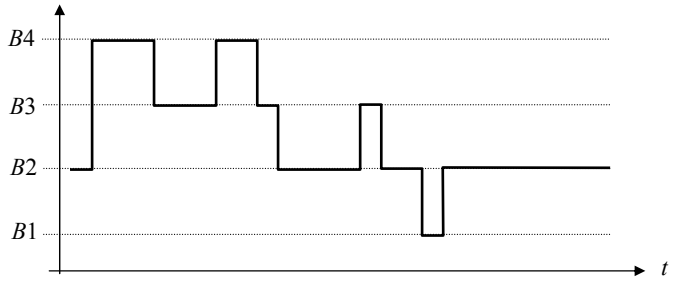
Batetik, ez dugu neurtuko tenperatura une oro, une jakin batzuetan baizik, orduan behin edo ordu erdian behin, esaterako. Hori egitean, tenperaturaren laginak hartuko ditugu —seinalea (magnitudea) "diskretizatuko" dugu—; lortutako balioetan oinarriturik, tenperaturaren aldaketari dagokion benetako kurba ondorioztatu ahal izango dugu, 2. irudian ageri den moduan. Zenbat eta puntu edo lagin gehiago hartu, hainbat eta zehatzagoa izango da lortuko dugun kurba.



2. irudia. Denbora "diskretua"; seinale analogikoaren laginak.

Bestetik, neurtzen ditugun magnitudeak analogikoak izan arren, lortuko ditugun emaitzek ezin izango dute edozein balio hartu, neurgailuen bereizmena mugatua delako. Adibidez, merkuriozko termometro arrunt bat erabiltzen badugu tenperatura neurtzeko, haren bereizmena eskuarki gradu batekoa izanik, esan ahal izango dugu tenperatura 23 ºC edo 24 ºC dela, baina ezin izango dugu neurtu tarteko baliorik, termometroak ez baitauka gradu erdiko "markarik" (23,5 ºC neurtu ahal izateko, alegia). Kasu horietan, termometroak eman dezakeen balio hurbilena neurtuko da.

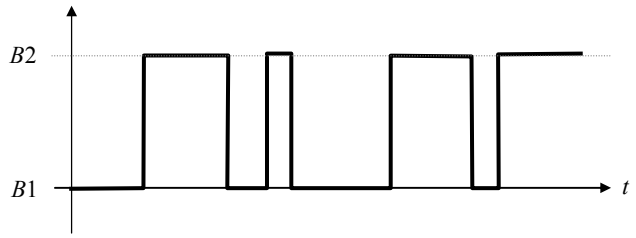
Oro har, seinale batek balio jakin batzuk bakarrik har ditzakeenean, seinalea digitala* dela esaten da. 3. irudian ageri da 4 balio har ditzakeen seinale digital bat.



3. irudia. Lau balio (B1, B2, B3 eta B4) har ditzakeen seinale digital bat.

* "Digital" hitza latinezko *digitus* (hatz) hitzetik dator, hasieran hatzak erabiltzen baitziren kontatzeko. Eskuarki, **digitu** hitza zifra adierazteko erabiltzen dugu: 1, 2, 3...

Seinale digital erabilienak seinale bitarrak dira; seinale bitarrek bi balio baino ez dituzte hartzen, 4. irudian ageri den bezala.



4. irudia. Seinale bitar bat.

SISTEMA DIGITALETAKO SEINALEAK

Sistema digitalen eginkizun nagusia informazioa prozesatzea da, horretarako seinale digitalak erabiliz.

Oro har, sistema digitaletako osagaiak transistoreen bidez egiten dira, eta, beraz, seinale elektrikoak prozesatzen dituzte (korronteak eta tentsioak). Zirkuitu digitaletan, transistoreek kommutagailu gisa funtzionatzen dute; hau da, bi egoera baino ez dituzte: irekita, ez dute korronterik eroaten; edo itxita, korrontea arazorik gabe pasatzen uzten dute. Hori dela eta, sistema digitaletan erabiltzen diren seinaleak bitarrak dira.

Hortaz, seinale bitarrek bi balio jakinetako bat bakarrik har dezakete une jakin batean. Eskuarki, seinale digitalak tentsio elektrikoak dira, eta 0 V edo 5 V balioak har ditzakete; baina, oro har, bi tentsio-balio horiek edozein izan zitezkeen (esaterako, gaur egun, 0 V eta 3 V gero eta gehiago erabiltzen dira). Bi tentsio-balioak baino gehiago, 0 eta 1 balio logikoak erabiltzen dira sistema digitalak teorikoki aztertzeko; beraz, kontuan izan testuan zehar erabiliko ditugun 0 eta 1 balioak “teorikoak” direla, eta, errealitatean, tentsio-balio jakin batzuk adierazten dituztela, erabiltzen den teknologiaren araberakoak. Zirkuitu digitalen analisi teorikoa egiteko, bi balio soilik onartzen dituen Aljebra Boolearra erabiltzen da.

Hala, seinale bitar baten balioa, une jakin batean, 0 edo 1 izango da. Seinale bitar batek ematen duen informazioari (1 edo 0) **bit** izena ematen zaio (ingelesezko ***b****inary dig****it*** terminoaren kontrakzioa da, digitu bitarra, alegia).

Bit batek informazio gutxi ematen du, eta, hori dela eta, bitak elkartuz, kode bitarrak sortzen dira, zenbakiak edo karaktereak bitarrez adierazteko. Esaterako, oso arrunta da 8 bit erabiltzea informazioa adierazteko, hain arrunta ezen izen berezia ematen baitzaio bit kopuru horri: **byte**a. Ohikoak dira, baita ere, 16, 32 eta 64 biteko elkarketak. Testuinguruaren arabera, "hitza" deritze elkarketa horiei.

n bit erabiliz informazioa bitarrez adierazteko, 2^n kode bitar desberdin lortzen dira. 1. taulan, 2^n funtzioaren ohiko balio batzuk eta balio horiek adierazteko erabiltzen diren aurrizkiak* ageri dira.

n	2^n	balioa	aurrizkia
10	2^{10}	1.024	k (kilo)
20	2^{20}	1.048.576	M (Mega)
30	2^{30}	1.063.741.824	G (Giga)

1. taula. Ohiko multiplo bitarrak.

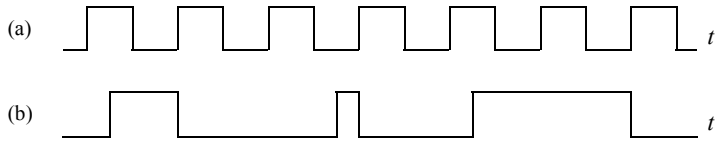
Aurrizki berezi horiek erabiltzen dira sistema digitaletan, baina bakarrik gailuen edukiera (informazio kopurua) adierazteko; adibidez, 1 Mb = 2^{20} bit, 20 GB = 20×2^{30} byte, eta abar (b = bit; B = byte).

SEINALE DIGITALEN EZAUGARRIAK

Esan bezala, seinale digital bitarrak prozesatzen dira sistema digitaletan, eta horiek, oro har, periodikoak edo ez-periodikoak izan daitezke. Seinale periodikoak behin eta berriro errepikatzen dira denboran zehar (5. irudiko (a) kasua). Errepikatzen ez diren seinaleak ez-periodikoak dira (5. irudiko (b) kasua).

---

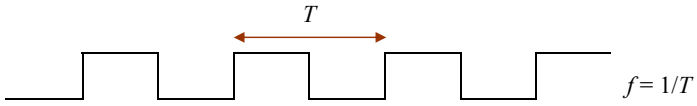
* Adi! aurrizki horiek ez dira gainerako magnitudeak adierazteko erabiltzen direnen guztiz berdinak. 1 km = 1000 m (10^3) da, eta, horregatik, 1024 (2^{10}) balioari k aurrizkia esleitu zaio, 1000tik oso gertu baitago. Beste horrenbeste gertatzen da gainerako aurrizkiekin: eskuarki 1 M = 10^6 (1.000.000) eta 1 G = 10^9 (1.000.000.000) badira ere, informazio kopurua adierazten denean, 1 M = 2^{20} eta 1 G = 2^{30} dira.
Gainerako magnitudeak ematean, aurrizkiek ohiko balioak dituzte; adibidez, gailu baten transferentzia-abiadura adierazten denean, 1 Mb/s = 10^6 bits.



5. irudia. Seinale digitalak: (a) periodikoa; (b) ez-periodikoa.

Seinaleak denboran zehar aldatzen direnez, ohikoa da haien eboluzioa grafikoki adieraztea **kronograma** izeneko irudietan, non ardatz horizontalean denbora islatzen den, aurreko irudian bezala.

Seinale periodikoen ezaugarririk behinenak **periodoa** (T) eta **maiztasuna** (f) dira. Periodoak adierazten du zenbat denbora behar duen seinaleak errepikatzeko, hots, zenbat segundo irauten duen seinalearen "ziklo" batek (ikus 6. irudia). Maiztasunak, berriz, segundo batean zenbat ziklo errepikatzen diren adierazten du, eta, beraz, periodoaren alderantzizkoa da.



6. irudia. Seinale periodiko bat: T = periodoa; f = maiztasuna.

Sistema digitaletako seinaleak gero eta maiztasun altuagokoak dira, hots, gero eta periodo laburragokoak. Hala, ohikoa da seinaleen periodoa mikrosegundotan neurtzea (1 μs = 10^{-6} s), nanosegundotan (1 ns = 10^{-9} s), edo pikosegundotan (1 ps = 10^{-12} s); eta seinaleen maiztasuna kilohertzetan[*] (1 kHz = 10^3 ziklo/s), megahertzetan (1 MHz = 10^6 ziklo/s), edo gigahertzetan (1 GHz = 10^9 ziklo/s).

Adibidez, seinale baten periodoa 125 ns baldin bada, haren maiztasuna 8 MHz da.

$$T = 125 \text{ ns} = 125 \times 10^{-9} \text{ s}$$

$$f = \frac{1}{T} = \frac{1}{125 \times 10^{-9} \text{ s}} = \frac{10^9}{125} \text{ s}^{-1} = 8 \times 10^6 \text{ Hz} = 8 \text{ MHz}$$

Eta, alderantziz, seinale baten maiztasuna 1 GHz baldin bada, irakurleak egiazta dezake haren periodoa 1 ns izango dela.

---

[*] Maiztasunaren unitatea Hertz-a (Hz) da; 1 Hz = 1 ziklo/s = 1 s^{-1} da; 1 Hz-eko seinalearen periodoa 1 s da.

SISTEMA DIGITALEN DISEINUA: ANALISIA ETA SINTESIA

Zer egin behar da sistema digital berri bat diseinatu nahi denean aplikazio jakin batean erabiltzeko? Hots, zertan datza sistema digitalak diseinatzea?

Oro har, sistema digitalen diseinua hainbat urratsetan banatzen den prozesua da. Lehendabizi, sistemaren funtzionamendua zehaztu behar da, eta hortik abiatuz, funtzionamendu hori erakutsiko duen zirkuitua lortu behar da; prozesu horri **sintesia** deritzo: osagaiak aukeratu, haien arteko loturak definitu, eta abar. Sintesiaren ondorioz, sistemaren eskema logikoa lortuko dugu. Gero, lortutako zirkuituaren **analisia** egin behar da, haren funtzionamendua egokia den egiaztatzeko. Baiezkoan, diseinua bukatutzat ematen da; ezezkoan, berriz, zerbait zuzendu beharko da, eta 1. urratsera itzuliko da. Bi urrats horiek behin eta berriro errepikatu beharko dira harik eta sistemaren funtzionamendua egokia izan arte.

Eskuarki, zirkuitu baten sintesia analisia baino konplexuagoa da. Sintesia sormen-prozesu bat da; analisia, aldiz, askoz automatikoagoa izan daiteke. Izan ere, portaera bereko sistema digital bat baino gehiago lor daiteke eta diseinatzailearen lana da horietako bat sortu eta aukeratzea.

Sistema digitalen erabilera zabalaren eraginez, gero eta konplexuagoak dira egin beharreko diseinuak; hortaz, ez da zaila akatsak sartzea diseinu-prozesuan. Akats horiek “oso garesti” ordaintzen dira gero, sistema fabrikatuta dagoenean antzematen badira. Hori dela eta, lan horretan laguntzeko asmoz, CAD (*Computer Aided Design*, konputagailuak lagunduriko diseinua) izeneko software-paketeak garatu dira.

Horien artean, sistema digitalen portaeraren analisia egiten duten simuladoreak daude. Konputagailuak simulatu behar duen sistema digitala adierazteko, bi aukera daude, oro har: eskema grafikoaren bidez edo testu-fitxategi baten bidez. Lehenengo kasuan, programa bereziak daude sistema digitalak marrazteko. Bigarren kasuan, lengoaia bereziak (HDL, *Hardware Description Languages*, hardwarea deskribatzeko lengoaiak) erabiltzen dira zirkuitua deskribatzeko. Hain ohikoak ez badira ere, zirkuituen sintesian laguntzeko programak ere badaude.

Gero eta aplikazio gehiagotan erabiltzen dira sistema digitalak: etxetresnak kontrolatzeko (domotika), ibilgailuetan, telefonoetan, aisialdiko trepetetan, eta, jakina, konputagailuetan. Kontrol-sistema analogikoak baino

fidagarriagoak dira eta, kanpoko eraginen aurrean, sendoagoak; gainera, fabrikazio-prozesuak direla eta, “merkeagoak” dira.

Liburuan zehar, sistema digitalen oinarrizko kontzeptuak analizatuko ditugu.

1. kapitulua

ALJEBRA BOOLEARRA

Aljebra boolearra da zirkuitu logikoen portaera azaltzen duen oinarrizko teoria matematikoa. Zirkuitu digitaletan prozesatzen diren seinaleek bi balio besterik ez dute hartzen —egiazkoa eta faltsua, edo 1 eta 0— eta seinale horien balio-aldaketek aljebra boolearraren axiomei eta teoremei jarraitzen diete. Horregatik, zirkuitu digitalak nola diseinatu eta erabili ikusi baino lehen, beharrezkoa da aljebra hori ezagutzea. Hori da, beraz, kapitulu honen helburua: aljebra boolearraren oinarrizko kontzeptuak azaltzea.

1.1. ALJEBRA BOOLEARRA

Giza arrazoibidearen lehen formalizazio matematikoa George Boole matematikari ingelesak egin zuen, 1854. urtean. Horretarako, zenbakizko aljebrarekin antz handia zuen beste aljebra bat definitu zuen, non "egiazkoa" (*true*, 1) eta "faltsua" (*false*, 0) balio logikoak erabili baitzituen. Aljebra horri, aljebra boolearra deitu ohi zaio.

Handik ia ehun urtetara, 1938an, Claude E. Shannon ingeniaritzako ikasleak egindako lanen bitartez, ondorioztatu zen aljebra boolearra erabil zitekeela zirkuitu digitalak analizatzeko eta diseinatzeko. Aljebra boolearrak era abstraktuan tratatzen du zirkuituen portaera, eta, horregatik, zirkuitu digitalak analizatzeko oinarrizko teoria da gaur egun ere, nahiz eta, urtetan zehar, zirkuituak eraikitzeko teknologiek asko eboluzionatu duten.

1.1.1. Aljebra boolearraren axiomak

Aljebra bat definitzeko, hiru elementu hauek behar dira: (a) osagai multzo bat; (b) eragiketa batzuk osagai horiekin; eta (c) axioma batzuk. Axiomek (edo postulatuek) aljebraren oinarrizko erregelak edo ezaugarriak definitzen dituzte.

Boolearra izateko, axioma konkretu batzuk bete behar ditu aljebra batek. Huntington matematikariak publikatu zituen, 1904. urtean, aljebra boolearraren axiomak.

Huntington-en axiomak laburbilduko ditugu hurrengo paragrafoetan. Axioma horietan, osagai multzoa ***B*** da, eta bi eragiketa erabiltzen dira, bi eragigaikoak: ***or*** —batuketa logikoa— eta ***and*** —biderketa logikoa—. Batuketa eta biderketa adierazteko, '+' eta '·' ikurrak erabili ohi dira: $a + b$ eta $a \cdot b$. Testuinguruaren arabera, ikur horiei "eragile" deritze.

Axiomak bikoitzak dira, bi eragiketetarako adierazten baitira.

- **A1 axioma.** B multzoko bi osagaien (edozeinen) batura zein biderkadura logikoa B multzokoa da. Hau da, B multzoa "itxia" da bi eragiketetarako.

$$\forall a,b \in B \rightarrow a + b \in B$$

$$\forall a,b \in B \rightarrow a \cdot b \in B$$

- **A2 axioma.** Bi eragiketek elementu neutro bana dute B multzoan. B-ko edozein elementu hartuta (a), a-ren eta elementu neutroaren arteko eragiketaren emaitza a bera da.

  Batuketaren elementu neutroa adierazteko, 0 ikurra erabiliko dugu, eta biderketarena adierazteko, 1 ikurra.

  $$\exists\, 0 \in B / \forall a \in B \rightarrow a + 0 = a$$

  $$\exists\, 1 \in B / \forall a \in B \rightarrow a \cdot 1 = a$$

- **A3 axioma.** Bi eragiketek trukatze-legea betetzen dute; hau da, eragigaien ordenak ez du eraginik emaitzan.

  $$\forall a, b \in B \rightarrow a + b = b + a$$

  $$\forall a, b \in B \rightarrow a \cdot b = b \cdot a$$

- **A4 axioma.** Bi eragiketek banatze-legea betetzen dute batak bestearekiko.

  $$\forall a, b, c \in B \rightarrow a + (b \cdot c) = (a + b) \cdot (a + c)$$

  $$\forall a, b, c \in B \rightarrow a \cdot (b + c) = (a \cdot b) + (a \cdot c)$$

- **A5 axioma.** B multzoko a elementuak (edozeinek) bere osagarria dauka, $\overline{a}$ (ez-a), baldintza hauek betetzen dituena:

  $$\forall a \in B \quad \exists\, \overline{a} \in B / a + \overline{a} = 1$$ (emaitza biderketaren elementu neutroa da)

  $$\forall a \in B \quad \exists\, \overline{a} \in B / a \cdot \overline{a} = 0$$ (emaitza batuketaren elementu neutroa da)

- **A6 axioma.** B multzoak baditu, gutxienez, bi osagai desberdin.

  $$\exists a, b \in B / a \neq b$$

Aipatu dugun bezala, boolearra deritzo axioma edo postulatu horiek betetzen dituen aljebrari. B multzoan bi osagai baino ez dagoenean —1 eta 0—, kommutazio-aljebra ere deitzen zaio. Axiomak kontsistenteak eta independenteak dira: ez dute kontraesanik sortzen eta ezin da bat ere ondorioztatu besteetatik abiatuz.

Adi! Aljebra boolearrak hainbat desberdintasun du ohiko aljebrarekin konparatuta; esaterako:

- B multzoa finitua da, eta zenbakien multzoa ez.
- Batuketa logikoak (+) banatze-legea betetzen du biderketa logikoarekiko (·); zenbakizko aljebran, aldiz, ez.

- Zenbakizko aljebran ez da $\overline{a}$ elementua (elementu osagarria) definitzen.
- Aljebra boolearrean ez dira alderantzizko elementuak definitzen bi eragiketetan ($-a$ batuketan eta $1/a$ biderketan), eta ez dira ez kenketa ez zatiketa eragiketak definitzen.

1.1.2. Aljebra boolearraren teorema erabilgarri batzuk

Aljebraren axiometatik hainbat lege edo teorema ondoriozta daitezke. Gure helburua ez da aljebra boolearra xehetasunez aztertzea, eta horregatik ez ditugu teorema horiek egiaztatuko, baina merezi du ohikoenak zerrendatzea, hainbat unetan erabiliko ditugulako. Axiomekin ikusi dugun moduan, teorema bakoitza bi eragiketetarako betetzen da.

- **T1 teorema** edo idenpotentzia-legea

  $$\forall a \in B \rightarrow a + a = a$$
  $$\forall a \in B \rightarrow a \cdot a = a$$

- **T2 teorema**

  $$\forall a \in B \rightarrow a + 1 = 1$$
  $$\forall a \in B \rightarrow a \cdot 0 = 0$$

- **T3 teorema** edo xurgapen-teorema

  $$\forall a,b \in B \rightarrow a + (a \cdot b) = a$$
  $$\forall a,b \in B \rightarrow a \cdot (a + b) = a$$

- **T4 teorema** edo inboluzio-legea

  $$\forall a \in B \rightarrow \overline{\overline{a}} = a$$

  hau da, elementu baten osagarriaren osagarria elementu bera da.

- **T5 teorema** edo elkartze-legea

  $$\forall a,b,c \in B \rightarrow (a + b) + c = a + (b + c)$$
  $$\forall a,b,c \in B \rightarrow (a \cdot b) \cdot c = a \cdot (b \cdot c)$$

- **T6 teorema** edo DeMorgan-en legeak

  $$\forall a,b \in B \rightarrow \overline{a + b} = \overline{a} \cdot \overline{b}$$
  $$\forall a,b \in B \rightarrow \overline{a \cdot b} = \overline{a} + \overline{b}$$

- **T7 teorema**

$$\forall a,b \in B \rightarrow a + \bar{a} \cdot b = a + b$$

$$\forall a,b \in B \rightarrow a \cdot (\bar{a} + b) = a \cdot b$$

Bai axiomak bai teoremak bikoteka betetzen dira. Izan ere, **dualitate-printzipioa** betetzen du aljebra boolearrak. Propietate horren arabera, axiometatik ondorioztatzen den edozein berdintza aljebraikok bere "duala" dauka, bi aldaketa hauek eginez lortzen dena:

- Bi eragiketak trukatu: biderketak batuketa bihurtu, eta alderantziz.
- Bi konstanteak trukatu: 0 konstanteak 1 bihurtu, eta alderantziz.

Esaterako, $a + 1 = 1 \rightarrow a \cdot 0 = 0$ (T2 teorema). Irakurleak erraz egiazta dezake propietate hori aurreko axioma eta teorema guztietan.

1.1.3. Oinarrizko eragiketa logikoak

1.1.3.1. Batuketa (*or*) eta biderketa (*and*)

Esan dugunez, bi eragiketa definitzen dira kommutazio-aljebran $B = \{0, 1\}$ osagai multzoarekin: batuketa logikoa —*a or b* edo $a + b$— eta biderketa logikoa —*a and b* edo $a \cdot b$ (askotan, punturik gabe ere: *a b*)—. Kontuan hartuta osagai multzoa mugatua dela —bi osagai bakarrik—, eragiketa bat definitzeko, nahikoa da eragigaien balioen konbinazio guztien emaitzak ematea. Halako informazioa biltzen duen taulari **egia-taula** deritzo.

Honela definitzen dira bi eragiketak, egia-taulak erabiliz:

a	*b*	***a or b***
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

a	*b*	***a and b***
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

1.1. taula. Batuketa (*or*) eta biderketa (*and*) eragiketa logikoen definizioak edo egia-taulak.

Beraz, batura 1 izango da eragigairen bat 1 denean; biderkadura, berriz, biak 1 direnean. Bi eragiketak horrela definituta, erraz egiazta daiteke aljebraren axiomak betetzen direla (ikus 1.1. taula):

A1 *B* itxia da bi eragiketetarako: emaitza 0 edo 1 da beti.

A2 Bi eragiketek elementu neutroa dute:

batuketaren elementu neutroa 0a da: $0 + \mathbf{0} = 0$ eta $1 + \mathbf{0} = 1$
biderketaren elementu neutroa 1a da: $0 \cdot \mathbf{1} = 0$ eta $1 \cdot \mathbf{1} = 1$

A3 Bi eragiketek trukatze-legea betetzen dute:

$$0 + 1 = 1 + 0 = 1$$
$$0 \cdot 1 = 1 \cdot 0 = 0$$

A4 Bi eragiketek banatze-legea betetzen dute bata bestearekiko:

$$\forall a, b, c \in B \rightarrow a \cdot (b + c) = (a \cdot b) + (a \cdot c)$$
$$\forall a, b, c \in B \rightarrow a + (b \cdot c) = (a + b) \cdot (a + c)$$

Hori frogatzeko, nahikoa da berdintza bakoitzaren bi aldeetako espresioen balio guztiak kalkulatzea eta konparatzea.

Froga dezagun lehenengo berdintza. 3 aldagai erabiltzen direnez, $2^3 = 8$ kasu posible analizatu behar dira. Emaitza guztiak 1.2. taulan ageri dira. Lehen hiru zutabeetan, hiru aldagaien konbinazio guztiak daude, eta, ondoan, aurreko espresioaren bi aldeetako funtzioen emaitzak.

a	b	c	$b + c$	$\mathbf{a \cdot (b + c)}$	$a \cdot b$	$a \cdot c$	$\mathbf{a \cdot b + a \cdot c}$
0	0	0	0	**0**	0	0	**0**
0	0	1	1	**0**	0	0	**0**
0	1	0	1	**0**	0	0	**0**
0	1	1	1	**0**	0	0	**0**
1	0	0	0	**0**	0	0	**0**
1	0	1	1	**1**	0	1	**1**
1	1	0	1	**1**	1	0	**1**
1	1	1	1	**1**	1	1	**1**

1.2. taula. Banatze-legearen froga.

1.2. taulan ageri denez (grisez markatutako bi zutabeak), bi espresioen balioak beti dira berdinak; hortaz, banatze-legea betetzen da.

Adierazpen duala frogatzea ariketa gisa uzten dugu.

A5 0 elementuaren osagarria 1a da: $0 + \mathbf{1} = 1$ eta $0 \cdot \mathbf{1} = 0$; beraz, $\overline{0} = 1$
1 elementuaren osagarria 0a da: $1 + \mathbf{0} = 1$ eta $1 \cdot \mathbf{0} = 0$; beraz, $\overline{1} = 0$

A6 Bi elementu daude *B* multzoan, 1a eta 0a.

1.1.3.2. Ezeztapena (*not*)

Bi eragigai erabiltzen dituzte *or* eta *and* eragiketek. *not* eragiketak, aldiz, eragigai bakarra du. Osagarria ere deitzen zaio eragiketa honi.

Hau da *not* eragiketaren egia-taula, A5 axiomatik (eta *or* eta *and* eragiketen definizioetatik) ondorioztatzen den moduan (ikus aurreko atala):

a	***not a***
0	**1**
1	**0**

1.3. taula. *not* eragiketaren egia-taula.

Eragile gisa, *not a* edo $\overline{a}$ erabiltzen da. Irakurtzean, berriz, hauetako eraren bat, denak baliokideak: *a* ezeztatua, *a*-ren osagarria, edo ez-*a*.

1.2. FUNTZIO LOGIKOAK

Funtzio logikoa aplikazio bat da, non jatorrizko multzoa $\{0, 1\}^n$ biderketa cartesiarra den, eta helburuko multzoa $\{0, 1\}$; hots: $\{0, 1\}^n \rightarrow \{0, 1\}$. Jatorrizko multzoan 2^n osagai daude, eta esaten da funtzioa *n* aldagaikoa dela; helburuko multzoan, aldiz, bi osagai besterik ez dago. Hori dela eta, *n* aldagaiko 2^{2^n} funtzio desberdin defini daitezke (aldagai bakarreko 4 funtzio; 2 aldagaiko 16 funtzio; eta abar).

Esaterako, hauek dira aldagai bakarreko 4 funtzioak:

a	f_1	f_2	f_3	f_4
0	0	0	1	1
1	0	1	0	1

1.4. taula. Aldagai bakarreko 4 funtzio logikoak; haien artean, *not* funtzioa (f_3) eta identitatea (f_2).

Modu berean, hauek dira 2 aldagaiko 16 funtzio logikoak:

a	*b*	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

1.5. taula. Bi aldagaiko 16 funtzio logikoak.

Funtzio horien artean, jadanik definitutako biderketa (*and*) eta batuketa (*or*) logikoak daude (f_2 eta f_8). Gainerakoen artean, zirkuitu digitaletan askotan erabiltzen diren beste funtzio batzuk daude:

- *xor* funtzioa (f_7), eta haren ezeztapena, *equ* funtzioa (f_{10}).

a	*b*	***a xor b***	***a equ b***
0	0	**0**	**1**
0	1	**1**	**0**
1	0	**1**	**0**
1	1	**0**	**1**

1.6. taula. *xor* eta *equ* funtzio logikoen egia-taula.

Beraz, *xor* (*exclusive or*) funtzioak 1 balioa hartzen du bi aldagaiak desberdinak direnean, eta *equ* funtzioak, aldiz, biak berdinak direnean. *xor* funtzioa adierazteko, $\oplus$ ikurra erabiltzen da, eta *equ* funtziorako, $\otimes$ ikurra. Egia-taulan ageri den moduan, honako hau betetzen da:

$$\overline{a \oplus b} = a \otimes b \qquad \text{eta} \qquad \overline{a \otimes b} = a \oplus b$$

- *nand* (f_{15}) eta *nor* (f_9) funtzioak, *and* eta *or* funtzioen ezeztapena, hurrenez hurren (ikus 1.5. taula).

Edozein funtzio logiko adieraz daiteke *not*, *and* eta *or* funtzio logikoen bidez. Ezaugarri hori dela eta, esaten da hiru funtzio horiek **sistema oso** bat osatzen dutela. Adibidez, *xor* eta *equ* funtzioak honela adieraz daitezke *and*, *or* eta *not* eragiketen bidez[1]:

$$a \oplus b = \bar{a} \cdot b + a \cdot \bar{b} \qquad \text{eta} \qquad a \otimes b = \bar{a} \cdot \bar{b} + a \cdot b$$

Berdintzak egiaztatzeko, adierazpen berrien egia-taulak egin behar dira eta 1.6. taulako definizioekin erkatu. 1.7. taulan ageri da lehenengo berdintzaren froga. Bestea irakurlearentzat uzten da.

a	*b*	$\bar{a}\,b$	$a\,\bar{b}$	$\bar{a}\,b + a\,\bar{b}$	$\boldsymbol{a \oplus b}$
0	0	0	0	**0**	**0**
0	1	1	0	**1**	**1**
1	0	0	1	**1**	**1**
1	1	0	0	**0**	**0**

1.7. taula. *xor* funtzioaren definizioa *and*, *or* eta *not* bitartez.

[1] Espresio aljebraikoetan, lehentasun hauek dituzte eragileek: *not*, *and* eta *or*.

1.2.1. Funtzio logikoen adierazpena

Bi era nagusi daude funtzio logikoak adierazteko, eta jadanik biak erabili ditugu aurreko adibideetan: egia-taulak eta adierazpen aljebraikoak. Lehendabiziko kasuan, egia-taulan, funtzioaren balioa adierazten da aldagaien konbinazio guztietarako (ez ahaztu: B multzoa finitua da, bi osagaikoa soilik, eta, beraz, konbinazio guztiak ere finituak dira eta taula batean bil daitezke). Dena den, aldagai askoko funtzioak horrela adieraztea astuna da, aldagaien balioen konbinazio guztiak asko izango baitira.

Bigarren aukera funtzioaren adierazpen aljebraikoa da, hau da, aldagaien eta eragileen bidezko adierazpena (aurreko atalean, *xor* eta *equ* funtzioekin egin dugun moduan).

Funtzio logiko baten adierazpen aljebraikotik abiatuta, erraza da haren egia-taula lortzea (nahikoa da adierazpenean adierazten diren eragiketak egitea aldagaien balio guztietarako), baina alderantzizkoa, hots, egia-taulatik abiatuta adierazpen aljebraikoa lortzea, ez da hain sinplea.

Ikus dezagun adibide bat. 1.8. taulan, f funtzioaren egia-taula ageri da. Nola adierazi funtzio hori adierazpen aljebraiko baten bitartez?

c	b	a	f
0	0	0	**0**
0	0	1	**1**
0	1	0	**1**
0	1	1	**0**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

1.8. taula. Funtzio logiko baten egia-taula.

Galdera horri erantzuteko, lehendabizi bi definizio emango ditugu.

- ***Minterm***-a (m_i): funtzio logiko baten aldagai guztien biderketa logikoa da, non aldagaiak ezeztatuta edo ezeztatu gabe ageri baitira. Egia-taulako lerro bakoitzari funtzioaren *minterm* bat dagokio.

  Minterm-a idaztean, aldagaia ezeztatuko da haren balioa 0 bada, eta ez da ezeztatuko 1 bada. Esaterako, $c = 1$, $b = 0$, eta $a = 1$ direnean, *minterm*-a $c\bar{b}a$ da.

Hala, 3 aldagaiko funtzio batek (1.8. taulakoak, esaterako) 8 *minterm* ditu. Taularen ordena berari jarraituz, hauek dira *minterm*-ak:

$$m_0 = \bar{c}\,\bar{b}\,\bar{a} \qquad m_1 = \bar{c}\,\bar{b}\,a \qquad m_2 = \bar{c}\,b\,\bar{a} \qquad m_3 = \bar{c}\,b\,a$$
$$m_4 = c\,\bar{b}\,\bar{a} \qquad m_5 = c\,\bar{b}\,a \qquad m_6 = c\,b\,\bar{a} \qquad m_7 = c\,b\,a$$

Minterm-ak izendatzeko, aldagaien balioek bitarrez adierazten duten zenbakia erabiltzen da: 0 0 0 → m_0, 0 0 1 → m_1, eta abar.

- ***Maxterm***-a (M_i): funtzio logiko baten aldagai guztien batuketa logikoa da, non aldagaiak ezeztatuta edo ezeztatu gabe ageri baitira. Egia-taulako lerro bakoitzari funtzioaren *maxterm* bat dagokio.

  Maxterm-a idaztean, aldagaia ezeztatuko da haren balioa 1 bada, eta ez da ezeztatuko 0 bada. Esaterako, c = 1, b = 0, eta a = 1 direnean, *maxterm*-a $\bar{c}+b+\bar{a}$ da.

  Esaterako, 3 aldagaiko funtzio batek 8 *maxterm* ditu. 1.8. taularen ordena berari jarraituz, hauek dira *maxterm*-ak:

$$M_0 = c+b+a \quad M_1 = c+b+\bar{a} \quad M_2 = c+\bar{b}+a \quad M_3 = c+\bar{b}+\bar{a}$$
$$M_4 = \bar{c}+b+a \quad M_5 = \bar{c}+b+\bar{a} \quad M_6 = \bar{c}+\bar{b}+a \quad M_7 = \bar{c}+\bar{b}+\bar{a}$$

Erraz egiazta daiteke, DeMorgan-en legeak erabiliz, honako erlazio hau betetzen dela *minterm*-en eta *maxterm*-en artean $m_i = \overline{M_i}$.

Minterm-ak eta *maxterm*-ak erabiliz, bi aukera ditugu funtzio logiko baten **adierazpen aljebraikoa** lortzeko, egia-taulatik abiatuta:

a. 1 balioa hartzen duten *minterm* guztien batuketa gisa. Adibidez, honela adieraz daiteke aurreko *f* funtzioa (ikus 1.8. taula):

$$f = \bar{c}\,\bar{b}\,a + \bar{c}\,b\,\bar{a} + c\,\bar{b}\,a + c\,b\,\bar{a} + c\,b\,a$$

Honako adierazpen hauek ere erabiliko ditugu funtzioak adierazteko:

$$f = \sum (m_1, m_2, m_5, m_6, m_7) = \sum (1, 2, 5, 6, 7)$$

b. 0 balioa hartzen duten *maxterm* guztien biderketa gisa. Adibidez, honela adieraz daiteke aurreko *f* funtzioa:

$$f = (c+b+a)\cdot(c+\bar{b}+\bar{a})\cdot(\bar{c}+b+a)$$

Aurrekoan bezala, honako adierazpen hauek ere erabiliko ditugu funtzioak adieratzeko:

$$f = \prod (M_0, M_3, M_4) = \prod (0, 3, 4)$$

Funtzio logikoak adierazteko aurreko bi aukerei **era kanonikoak** deitzen zaie. Zirkuitu digitaletako funtzioak adierazteko, *minterm*-en bidezko era kanonikoa erabili ohi da.

Bi adierazpen kanonikoak baliokideak dira. Hori egiaztatzeko, lehen ikusi dugun moduan, egia-taulak idatz eta konpara daitezke. Baina hori ez da aukera bakarra. Aljebra boolearraren axiomak eta teoremak aplika daitezke adierazpen batetik bestera iristeko.

Baliokidetasuna frogatzeko, $\overline{f}$ funtzioaren *minterm*-en bidezko adierazpenetik abiatuko gara, eta, horretarako, jatorrizko *f* funtzioaren 0koak hartu behar ditugu kontuan (ikus 1.8. taula):

$$f = \bar{c}\,\bar{b}\,a + \bar{c}\,b\,\bar{a} + c\,\bar{b}\,a + c\,b\,\bar{a} + c\,b\,a \qquad \text{beraz,}$$

$$\overline{f} = \bar{c}\,\bar{b}\,\bar{a} + \bar{c}\,b\,a + c\,\bar{b}\,\bar{a} \qquad (f \text{ funtzioaren osagarria})$$

Berdintzaren bi alderdiak ezeztatzen ditugu:

$$\overline{\overline{f}} = \overline{\bar{c}\,\bar{b}\,a + \bar{c}\,b\,a + c\,\bar{b}\,a}$$

Orain, T4 teorema (inboluzio-legea) aplikatzen diogu ezkerreko alderdiari eta DeMorgan-en legea (T6) eskuinekoari:

$$f = \overline{\bar{c}\,\bar{b}\,a} \cdot \overline{\bar{c}\,b\,a} \cdot \overline{c\,\bar{b}\,a}$$

Berriro DeMorgan-en legea aplikatuz funtzioaren gai bakoitzari:

$$f = (\bar{\bar{c}} + \bar{\bar{b}} + \bar{\bar{a}}) \cdot (\bar{\bar{c}} + \bar{b} + \bar{a}) \cdot (\bar{c} + \bar{\bar{b}} + \bar{\bar{a}})$$

Azkenik, berriro T4 teorema (inboluzio-legea) aplikatuz:

$$f = (c + b + a) \cdot (c + \bar{b} + \bar{a}) \cdot (\bar{c} + b + a)$$

Horixe da *f* funtzioaren bigarren adierazpen kanonikoa: 0 balioa hartzen duten *maxterm*-en biderketa. Beraz, frogatu dugu bi adierazpen kanonikoak baliokideak direla.

1.2.2. Funtzioen adierazpenaren minimizazioa

Funtzio logiko baten adierazpen aljebraikotik abiatuta, ez da zaila funtzio logiko hori gauzatzen duen zirkuitu digital bat sortzea. Hori nola egiten den, hurrengo kapituluan ikusiko dugu.

Hala ere, ikusi dugun bezala, funtzio logiko batek ez du adierazpen bakarra: bi adierazpen kanonikoak ikusi ditugu, eta gehiago ere badaude. Beraz, zein da egokiena zirkuitu fisikoa eraikitzeko?

Erantzuna kontuan hartu nahi diren diseinu-parametroen araberakoa izango da: zirkuiturik "txikiena", hots, hardware minimoa erabiltzen duena, kontsumo baxuenekoa, azkarrena, merkeena... Une honetan, zirkuitu "txikienak" sortzea interesatzen zaigu. Ez dago zuzeneko erlaziorik adierazpen aljebraikoaren eta zirkuitu digital baten osagai kopuruaren artean, baina, oro har, komeni da adierazpen aljebraiko minimoak erabiltzea zirkuituak eraikitzeko[2].

Funtzio logiko baten adierazpen minimoa lortzeko, aljebraren axiomak eta teoremak erabili behar dira, batik bat funtzioaren adierazpena sinplifikatzen dutenak. Ikus dezagun adibide bat.

Izan bedi 3 aldagaiko funtzio logikoa, honela definituta, lehenengo era kanonikoan, *minterm*-en batuketa gisa: $f = \sum (0, 1, 2, 3, 6)$.

Beraz,

$$f = \bar{c}\bar{b}\bar{a} + \bar{c}\bar{b}a + \bar{c}b\bar{a} + \bar{c}ba + cb\bar{a} = \quad \text{(banatze-legea)}$$

$$= \bar{c}\bar{b}(\bar{a} + a) + \bar{c}b(\bar{a} + a) + cb\bar{a} = \quad \text{(A5/A2 axiomak)}$$

$$= \bar{c}\bar{b} + \bar{c}b + cb\bar{a} = \quad \text{(banatze-legea)}$$

$$= \bar{c}(\bar{b} + b) + cb\bar{a} = \quad \text{(A5/A2 axiomak)}$$

$$= \bar{c} + cb\bar{a} = \quad \text{(T7 teorema)}$$

$$= \bar{c} + b\bar{a} \quad \text{adierazpen minimoa}$$

Funtzioa ezin da gehiago sinplifikatu, eta, beraz, hori da haren adierazpen minimoa. Garbi dago: "erosoagoa" da adierazpen minimo hori hasierakoa baino.

Aldagai gutxiko funtzioak horrela minimizatzea ez da zaila, baina aldagai kopurua handia bada, konplexua izan daiteke adierazpen minimoa lortzea. Izan ere, hainbat programa daude funtzio logikoak minimizatzeko konputagailuaren bidez.

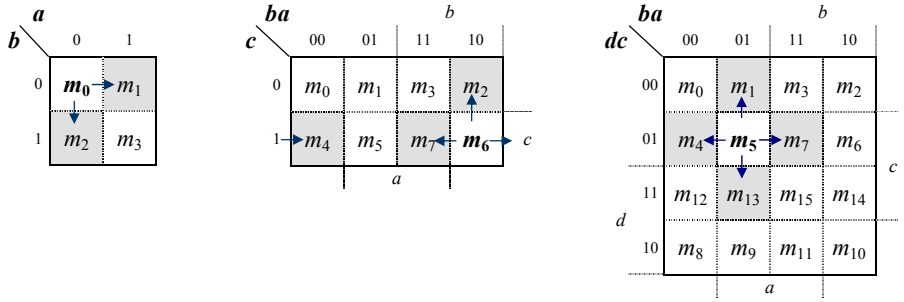
---

[2] Gaur egun, funtzio logikoen minimizazioa ez da behinola izan zen bezain garrantzitsua: gero eta transistore gehiago integratzen dira zirkuituetan eta diseinu-tresnak egokitu dira ezaugarri hori aprobetxatzeko. Hala ere, egokia eta erosoa da ahalik eta funtzio sinpleenak erabiltzea.

1.2.2.1. Karnaugh-en mapak

Prozedura aljebraikoaz gain, badago metodo grafiko nahiko sinplea funtzio logikoak minimizatzeko, aldagai kopurua baxua denean: **Karnaugh-en mapak.**

Funtzio logiko baten Karnaugh-en mapa (edo, laburrago, K-mapa) funtzioaren egia-taula baino ez da, baina bi dimentsiotan eta **ordena berezian** adierazita (gero azalduko dugu ordena horren zioa). Itxura desberdineko K-mapak daude, funtzioaren aldagai kopuruaren arabera. 1.1. irudian, 2, 3 eta 4 aldagaiko K-mapak ageri dira (5 aldagaiko mapa 4 aldagaiko bi mapa gisa adierazi ohi da; 6koa, 4 aldagaiko lau mapa moduan; eta abar; hala ere, aldagai kopurua altua denean, K-mapak ez dira oso erabilgarriak).

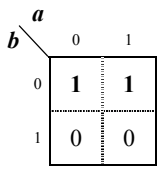


1.1. irudia. Bi, hiru eta lau aldagaiko Karnaugh-en mapak (geziek *minterm* baten "albokoak" adierazten dituzte, bit bakar batean bereizten direnak).

1.1. irudian ageri den moduan, mapetako gelaxka bakoitza funtzioaren *minterm* bat da (*minterm*-ak esleitzeko, *dcba* ordena erabili ohi da, *a* izanik beti "pisu" txikieneko aldagaia). Mapako gelaxka bakoitzean, *minterm* horretarako funtzioak hartzen duen balioa idatzi behar da. Ikus ditzagun adibide batzuk.

1. adibidea

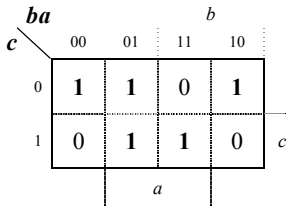
$$f = \bar{b}\,\bar{a} + \bar{b}\,a = \sum(0, 1)$$



2. adibidea

$$f = \overline{c}\,\overline{b}\,\overline{a} + \overline{c}\,\overline{b}\,a + \overline{c}\,b\,\overline{a} + c\,\overline{b}\,a + c\,b\,a$$

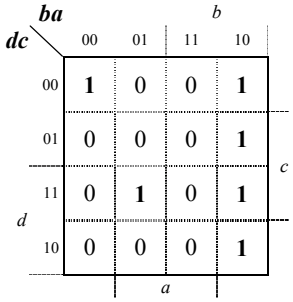
$$f = \sum (0, 1, 2, 5, 7)$$



3. adibidea

$$f = \overline{d}\,\overline{c}\,\overline{b}\,\overline{a} + \overline{d}\,\overline{c}\,b\,\overline{a} + \overline{d}\,c\,b\,\overline{a} + \\ + d\,\overline{c}\,b\,\overline{a} + d\,c\,\overline{b}\,a + d\,c\,b\,\overline{a}$$

$$f = \sum (0, 2, 6, 10, 13, 14)$$



Azter dezagun 1.1. irudiko 4 aldagaiko mapa. Ezkerraldean, *d* eta *c* aldagaien balioak ageri dira, eta goiko aldean, berriz, *b* eta *a* aldagaienak, **ordena honetan:** 00, 01, 11, 10. Ordena hori dela eta —ondoz ondoko kodeak bit bakar batean bereizten dira—, propietate berezi bat dute Karnaugh-en mapetako gelaxkek: gelaxka bati dagokion *minterm*-a eta alboko gelaxketako *minterm*-ak aldagai bakar batean bereizten dira (gauza bera gertatzen da gainerako mapetan).

Adibidez, 4 aldagaiko mapan, gelaxka bakoitzak 4 gelaxka ditu alboetan: goian, behean, ezkerrean eta eskuinean (oro har, *n* aldagaiko mapetan, *n* albo-gelaxka daude). 1.1. irudian ageri den moduan, hau gertatzen da:

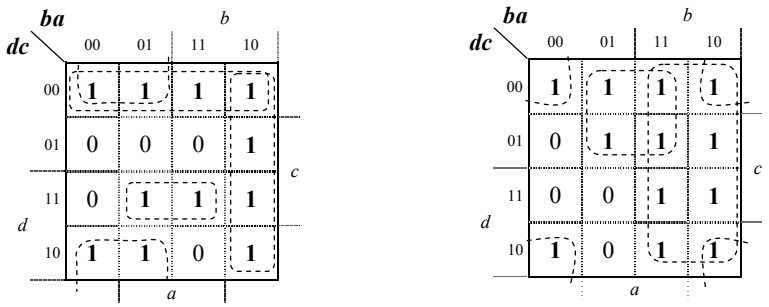
- gelaxka jakin bat (m_5)	→	$\overline{d}\,c\,\overline{b}\,a$	
- albo-gelaxkak	→	$d\,c\,\overline{b}\,a$	(m_{13}, azpian, *d* aldagaia aldatuta)
		$\overline{d}\,\overline{c}\,\overline{b}\,a$	(m_1, gainean, *c* aldagaia aldatuta)
		$\overline{d}\,c\,b\,a$	(m_7, eskuinean, *b* aldagaia aldatuta)
		$\overline{d}\,c\,\overline{b}\,\overline{a}$	(m_4, ezkerrean, *a* aldagaia aldatuta)

Gainera, mapak “zirkularrak” dira, hots, ertz batean kokatutako gelaxkaren albokoa kontrako aldean dagoen gelaxka da, 1.1. irudiko 3 aldagaiko mapan ageri den moduan (m_4 gelaxka m_6 gelaxkaren albokoa da).

K-mapen propietate hori erabiltzen da funtzioen adierazpen aljebraikoak minimizatzeko. Izan ere, baldin baditugu 1 balioko *minterm*-ak alboko bi gelaxkatan, honako sinplifikazio-erregela hau aplika dakieke, $b\,a + b\,\overline{a} = b$,

aldagai bakar batean bereiziko baitira. Hala, funtzioaren adierazpen sinpleagoa lortuko dugu: aldagai komunak baino ez dituen gai bakar batean bilduko ditugu jatorrizko bi gaiak.

Teknika hori —funtzioaren bi gai bakar batean biltzea— modu errekurtsiboan aplika daiteke tamaina bereko elkarketekin; hots, 4ko elkarketak egin daitezke 2ko elkarketekin; 8koak, 4koekin, eta abar (betiere, 2ren berreturak). 1.2. irudian, *minterm*-en elkarketa batzuk ageri dira.



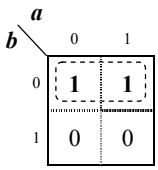
1.2. irudia. Gelaxka-elkarketa posible batzuk. Kontuan hartu mapa "zirkularra" dela.

Hau da, beraz, funtzio baten adierazpen minimoa lortzeko prozedura:

0. Kontuan hartu 1 balioa duten mapako (funtzioko) *minterm*-ak edo gelaxkak.
1. Ahal den neurrian, elkartu gelaxka horiek 2naka, 4naka, 8naka, baldin eta alboko posizioetan badaude. Aukeran, elkarketarik handiena erabili behar da.
2. Oro har, aukera bat baino gehiago izango da 1ekoak elkartzeko. Hala bada, hasi beti elkarketa-aukera bakarra (edo gutxien) duten gelaxkekin.
3. Dagoeneko beste elkarketa batean sartu diren 1ekoak berriro elkar daitezke, horrekin lortzen bada beste 1eko batzuk multzo handiagoetan elkartzea. Hain zuzen ere, $a + a = a$ da.
4. Elkartze-prozesua bukatzen da 1eko guztiak kontuan hartu eta elkarketa handienetan bildu direnean. Elkarketa kopuruak ahalik eta txikiena izan behar du.
5. Elkarketei dagozkien gai sinplifikatuen batuketa da funtzioaren adierazpen minimoa. Gai horiek lortzeko, kontuan hartu behar dira bakarrik elkarketa horietako *minterm*-etan balio bera duten aldagaiak, eta ez aldatzen direnak; adibidez: $a\,b\,c + a\,\bar{b}\,c = a\,c$.

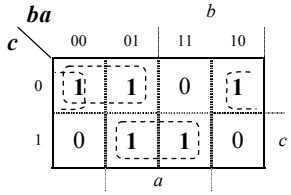
Ikus dezagun nola minimizatu aurreko adibideetako funtzioak.

1. adibidea: $f = \overline{b}\,\overline{a} + \overline{b}\,a$



Bi 1eko daude funtzioan. Bata bestearen alboan daudenez, elkartu egin daitezke. Bi gelaxka horietan komuna honako hau da: $b = 0$. Beraz, hau da funtzioaren adierazpen minimoa: $f = \overline{b}$

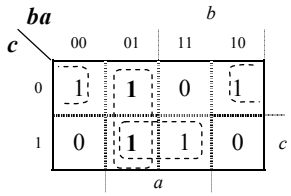
2. adibidea: $f = \overline{c}\,\overline{b}\,\overline{a} + \overline{c}\,\overline{b}\,a + \overline{c}\,b\,\overline{a} + c\,\overline{b}\,a + c\,b\,a$



1ekoak elkartzeko, *cba* = 010 *minterm*-arekin has gaitezke, eta elkartu 000 *minterm*-arekin (hori da duen aukera bakarra); gai berria $\overline{c}\,\overline{a}$ da. Gauza bera egin daiteke 101 eta 111 *minterm*-ekin, eta emaitza $c\,a$ da. Azkenik, 001 eta 000 *minterm*-ak elkar daitezke eta $\overline{c}\,\overline{b}$ gaia sortu. Ikusten den moduan, 000 *minterm*-a bi elkarketetan sartu da, horrekin funtzio sinpleagoa lortuko baita.

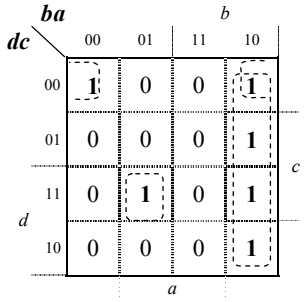
Hala, hau da funtzioaren adierazpen minimoa: $f = \overline{c}\,\overline{a} + c\,a + \overline{c}\,\overline{b}$

Hainbat kasutan, adierazpen minimo bat baino gehiago lor daiteke. Esaterako, honela ere minimiza daiteke aurreko K-mapa:



Kasu honetan, 001 *minterm*-a 101 *minterm*-arekin elkartu dugu; gai berria $\overline{b}\,a$ da, eta funtzioa $f = \overline{c}\,\overline{a} + c\,a + \overline{b}\,a$. Bi adierazpenak baliokideak dira.

3. adibidea: $f = \overline{d}\,\overline{c}\,\overline{b}\,\overline{a} + \overline{d}\,\overline{c}\,b\,\overline{a} + \overline{d}\,c\,b\,\overline{a} + d\,\overline{c}\,b\,\overline{a} + d\,c\,\overline{b}\,a + d\,c\,b\,\overline{a}$



0000 *minterm*-a elkartzeko, aukera bakarra dago: 0010 *minterm*-arekin; funtziorako gai berria $\overline{d}\,\overline{c}\,\overline{a}$ da. Bestalde, eskuineko zutabeko lau *minterm*-ak ere elkar daitezke, eta honako gai hau sortu: $b\,\overline{a}$. Azkenik, 1101 *minterm*-a ezin da beste inorekin elkartu, eta hari dagokion gaia $d\,c\,\overline{b}\,a$ izango da. Beraz: $f = b\,\overline{a} + \overline{d}\,\overline{c}\,\overline{a} + d\,c\,\overline{b}\,a$.

Laburrean. K-mapak tresna grafiko sinple bat dira funtzioen adierazpen aljebraiko minimoak lortzeko, *minterm*-ak elkartuz gai sinpleagoak erdiesteko. Mapen bidez, aljebra boolearraren axiomak eta teoremak besterik ez da aplikatzen (batik bat $b\,a + b\,\overline{a} = b$ eta $a + a = a$). Dena den, zaila da K-mapak erabiltzea aldagai kopurua altuagoa denean; kasu horietarako, bide aljebraikoa baino ez da geratzen (oro har, konputagailuaren laguntzaren bidez).

1.2.3. Zehaztu gabeko gaiak (*don't care*)

n aldagaiko funtzio logiko batean, 2^n konbinazio desberdin daude: sarrera-aldagaien konbinazio bakoitzeko bat. Sistema logikoen portaera adierazten duten funtzio logikoak interesatzen zaizkigu, eta, horietako askotan, ohikoa da sarrera-aldagaien konbinazio guztiak behar ez izatea. Bi arrazoirengatik gerta daiteke hori: (a) konbinazio horiek ez direlako inoiz gertatuko; edo (b) gertatzen badira ere, berdin zaigulako funtzioak hartuko duen balioa, eraginik izango ez duelako (ikus 1.8. ariketa).

Kasu horietan, esaten da funtzioa ez dagoela zeharo espezifikatuta, ez baita emaitza adierazten —ez 0, ez 1— hainbat sarrera-konbinaziotarako. Zehaztu gabeko balio horiek egia-tauletan eta K-mapetan adierazteko, "x" edo "–" ikurra erabiliko dugu (baita *d* letra ere —*don't care*— funtzioa *minterm*-en batukari gisa ematen denean). *Minterm* horiei **zehaztu gabeko gaiak** deritze.

Adibidez,

a	*b*	***f***
0	0	–
0	1	**1**
1	0	**1**
1	1	**0**

$f = \sum (1, 2) + d(0)$

Funtzio horrek zehaztu gabeko gai bat du: bi aldagaiak 0 direnean, "berdin zaigu" zein den funtzioaren balioa; beraz, 1 edo 0 izan daiteke.

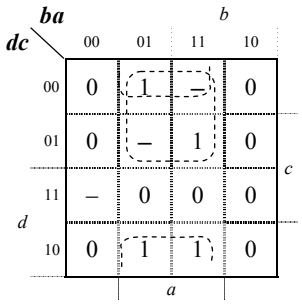
Sinplifikatu behar badugu zehaztu gabeko gaiak dituen funtzio bat, 0 gisa edo 1 gisa hartuko dugu gai horien balioa, funtzioaren adierazpena minimizatzearren. Hau da, 1 baliokotzat hartuko ditugu baldin eta *minterm*-en elkarketa handiagoak egin badaitezke haien bitartez; bestela, 0 baliokotzat hartuko ditugu. Ikus dezagun adibide bat.

$f = \sum (1, 7, 9, 11) + d(3, 5, 12)$ funtzioaren adierazpen aljebraiko minimoa lortu behar da. Funtzioaren lau *minterm* 1ekoak dira, eta badaude zehaztu gabeko hiru gai. Lehendabizi, K-mapa irudikatuko dugu.

dc \ ***ba***	00	01	11	10
00	0	1	–	0
01	0	–	1	0
11	–	0	0	0
10	0	1	1	0

b (gainean) · *c* (eskuinean) · *d* (ezkerrean) · *a* (azpian)

1 balioa duten *minterm*-ak elkartzeko, zehaztu gabeko gaiak ere erabil daitezke, horrela sortutako elkarketak handiagoak badira. Adibidean, 0001 eta 0111 *minterm*-ak elkartu ahal izango ditugu, 0011 eta 0101 zehaztu gabeko gaiak 1ekotzat hartuz. Era berean, 0011 zehaztu gabeko gaiaren bidez, elkarketa handiago batean (4koan) sar daitezke 1001 eta 1011 *minterm*-ak (0001 *minterm*-arekin batera).



Beraz, bi elkarketa horiek direla eta, hau izango da *f* funtzioaren adierazpen minimoa:

$$f = \overline{d}\, a + \overline{c}\, a$$

Hori lortzeko, zehaztu gabeko bi gai hartu ditugu elkarketetarako (1ekotzat, beraz), eta hirugarrena (1100), aldiz, ez (0kotzat hartu da).

1.3. ARIKETA EBATZIAK

Aljebra boolearraren oinarrizko kontzeptu teorikoak azaldu ditugu aurreko paragrafoetan. Atal honetan, berriz, ariketa batzuk ebatziko ditugu, kontzeptu teoriko horiek nola erabiltzen diren argi eta garbi erakusteko asmoz. Ariketa guztiak sinpleak dira, eta zehatz-mehatz azalduko ditugu; beraz, gaia ezagutzen badu, irakurleak hurrengo kapituluetara jo dezake.

» 1.1. Ariketa

Aljebra boolearraren axiomak eta teoremak erabiliz, sinplifika ezazu funtzio logiko honen adierazpen aljebraikoa:

$$f = a\,b\,c\,d + \bar{b}\,\bar{a} + a\,b\,\bar{c} + \bar{b}\,a + \bar{c}\,a$$

■

Adierazpen aljebraikoak sinplifikatzeko, modu bat baino gehiago dago. Esaterako, adibide honetan $a\,b$ faktore komuna atera daiteke 1. eta 3. gaietan, eta $\bar{b}$ 2. eta 4. gaietan. Hala,

$$f = a\,b(c\,d + \bar{c}) + \bar{b}(\bar{a} + a) + \bar{c}\,a$$

Lehenengo parentesiko adierazpenari T7 teorema aplika dakioke, eta bigarrenari A5 axioma (ikus 1.1.1. eta 1.1.2. atalak). Beraz,

$$f = a\,b(d + \bar{c}) + \bar{b}\cdot 1 + \bar{c}\,a = a\,b(d + \bar{c}) + \bar{b} + \bar{c}\,a$$

non $\bar{b}\cdot 1 = \bar{b}$ dela aplikatu baitugu (A2 axioma).

Orain, f funtzioaren lehenengo eta bigarren batugaiei aplika dakieke berriro T7 teorema, eta, hala, lehenengo batugaiko b aldagaia kendu; hau da,

$$f = a\,(d + \bar{c}) + \bar{b} + \bar{c}\,a$$

Azkenik, banatze-legea (A4 axioma) aplikatzen badugu lehenengo gaian, bi gai berdinak lortuko ditugu adierazpidean —$a\,\bar{c} = \bar{c}\,a$—, eta beraz, bat ken daiteke (T1 teorema):

$$f = a\,d + a\,\bar{c} + \bar{b} + \bar{c}\,a = a\,d + a\,\bar{c} + \bar{b} = a\,(d + \bar{c}) + \bar{b}$$

Hori da f funtzioaren adierazpen minimoa, ezin baita gehiago sinplifikatu.

«

1.2. Ariketa

Adieraz ezazu $f = d\,(\overline{a} + b\,(\overline{c} + a\,\overline{d}))$ *funtzioaren osagarria, eta minimiza ezazu haren adierazpena.*

■

Funtzio baten osagarria lortzeko, *not* eragilea aplikatu behar da. Beraz,

$$\overline{f} = \overline{d\,(\overline{a} + b\,(\overline{c} + a\,\overline{d}))}$$

Gero, adierazpena sinplifikatzeko, DeMorgan-en legeak aplika daitezke (T6 teorema), gainerako legeekin batera. Hasteko, biderketa baten ezeztapena dago; ondorioz, biderketa batuketa bihurtuko dugu eta bi biderkagaiak ezeztatu:

$$\overline{f} = \overline{d \cdot (\overline{a} + b\,(\overline{c} + a\,\overline{d}))} = \overline{d} + \overline{(\overline{a} + b\,(\overline{c} + a\,\overline{d}))}$$

Eragiketa bera aplika dakioke orain bigarren gaiari: batuketa biderketa bihurtuz eta bi eragigaiak ezeztatuz. Hau da,

$$\overline{f} = \overline{d} + \overline{\overline{a}} \cdot \overline{(b\,(\overline{c} + a\,\overline{d}))} = \overline{d} + a \cdot \overline{(b\,(\overline{c} + a\,\overline{d}))}$$

non, azkenean, $\overline{\overline{a}} = a$ dela erabili baitugu (T4 teorema).

Prozedura bera aplika daiteke behin eta berriz:

$$\begin{aligned}\overline{f} &= \overline{d} + a\,(\overline{b} + \overline{(\overline{c} + a\,\overline{d})}) = \\ &= \overline{d} + a\,(\overline{b} + (\overline{\overline{c}} \cdot \overline{(a\,\overline{d})})) = \overline{d} + a\,(\overline{b} + (c\,\overline{(a\,\overline{d})})) = \\ &= \overline{d} + a\,(\overline{b} + c\,(\overline{a} + \overline{\overline{d}})) = \overline{d} + a\,(\overline{b} + c\,(\overline{a} + d))\end{aligned}$$

Azkenik, adierazpen hori sinplifikatuko dugu:

$$\overline{f} = \overline{d} + a\,\overline{b} + a\,c\,(\overline{a} + d) = \overline{d} + a\,\overline{b} + a\,c\,\overline{a} + a\,c\,d$$

Banatze-legea aplikatu ondoren, hirugarren gaia ken daiteke $a \cdot \overline{a} = 0$ delako (A5 axioma). Era berean, 4. gaia sinplifika daiteke 1.arekin, T7 teorema erabiliz.

Beraz, hauxe da $\overline{f}$ funtzioaren adierazpen minimoa:

$$\overline{f} = \overline{d} + a\,\overline{b} + a\,c = \overline{d} + a\,(\overline{b} + c)$$

«

1.3. Ariketa

Sor ezazu funtzio honen egia-taula: $f = (a\bar{b} + c) \cdot (b + \bar{a}\,c)$

■

Funtzio baten egia-taula osatzeko, funtzioak hartzen dituen balioak kalkulatu behar ditugu, aldagaien balioen konbinazio guztietarako. Modu askotan egin badaiteke ere, egokia da taulako ezkerreko aldean funtzioaren aldagaiak idaztea, eta eskuineko aldean, berriz, funtzioak hartzen dituen balioak. Funtzioa konplexua den neurrian, komeni da funtzioaren gaien eta "azpigaien" balioak ere taulan idaztea, lagungarriak direlako funtzioaren balioa kalkulatzeko.

Beraz, adibide honetarako egingo dugun egia-taulan, *c, b* eta *a* aldagaien balioak idatziko ditugu ezkerraldean; ondoan, $a\bar{b}$, $a\bar{b}+c$, $\bar{a}c$ eta $b+\bar{a}c$ gaiak; eta azkenik, *f* funtzioa. 3 aldagaiko funtzioa denez, 8 kasu desberdin (2^3) izango ditugu aldagaien balio gisa, taulan ageri direnak.

c	***b***	***a***	$a\bar{b}$	$a\bar{b}+c$	$\bar{a}c$	$b+\bar{a}c$	f
0	0	0					
0	0	1					
0	1	0					
0	1	1					
1	0	0					
1	0	1					
1	1	0					
1	1	1					

Prest gaude egia-taula betetzeko, zutabez zutabe edo lerroz lerro, oinarrizko eragiketen edo funtzioen egia-taulak kontuan hartuz (*not*, *or*, *and*...). Esaterako, $a\bar{b}$ zutabean, 1eko bat ageriko da bakarrik $a = 1$ eta $b = 0$ direnean. Gainerako kasuetan, emaitza 0 da.

c	b	a	$a\bar{b}$	$a\bar{b}+c$	$\bar{a}c$	$b+\bar{a}c$	f
0	0	0	**0**				
0	0	1	**1**				
0	1	0	**0**				
0	1	1	**0**				
1	0	0	**0**				
1	0	1	**1**				
1	1	0	**0**				
1	1	1	**0**				

Prozedura bera errepikatu behar da gainerako zutabeak betetzeko. Hona hemen tarteko emaitzak:

c	b	a	$a\overline{b}$	$a\overline{b}+c$	$\overline{a}c$	$b+\overline{a}c$	f
0	0	0	**0**	**0**	**0**	**0**	
0	0	1	**1**	**1**	**0**	**0**	
0	1	0	**0**	**0**	**0**	**1**	
0	1	1	**0**	**0**	**0**	**1**	
1	0	0	**0**	**1**	**1**	**1**	
1	0	1	**1**	**1**	**0**	**0**	
1	1	0	**0**	**1**	**1**	**1**	
1	1	1	**0**	**1**	**0**	**1**	

Azkenik, funtzioaren balioak kalkulatuko ditugu, taulako tarteko bi emaitzak (grisez bereizita) biderkatuz. Hona, beraz, f funtzioaren egia-taula:

c	b	a	$a\overline{b}$	$a\overline{b}+c$	$\overline{a}c$	$b+\overline{a}c$	f
0	0	0	0	0	0	0	**0**
0	0	1	1	1	0	0	**0**
0	1	0	0	0	0	1	**0**
0	1	1	0	0	0	1	**0**
1	0	0	0	1	1	1	**1**
1	0	1	1	1	0	0	**0**
1	1	0	0	1	1	1	**1**
1	1	1	0	1	0	1	**1**

Aldagai askoko funtzioen egia-taulak oso luzeak dira; hainbat kasutan, hala ere, taulen idazketa sinplifika daiteke lerroak bilduz. Esaterako, aurreko taula era honetan ere idatz daiteke:

c	b	a	f
0	–	–	**0**
1	0	0	**1**
1	0	1	**0**
1	1	–	**1**

Honela irakurri behar da taulako lehen lerroa: $c = 0$ denean, $f = 0$ da, edozein direlarik b eta a. Beraz, aurreko taulako lehenengo 4 lerroak bildu ditugu lerro bakar batean. Modu berean, azkeneko lerroan adierazten da $f = 1$ izango dela $c = b = 1$ direnean, a-ren balioa kontuan hartu gabe (aurreko taulako azkeneko bi lerroak).

«

1.4. Ariketa

Hiru aldagaiko f eta g funtzioen egia-tauletan oinarrituta, idatz itzazu bi funtzioen adierazpen kanonikoak, minterm-en eta maxterm-en bitartez.

	c	*b*	*a*	***f***	***g***
0	0	0	0	1	0
1	0	0	1	1	0
2	0	1	0	1	1
3	0	1	1	0	0
4	1	0	0	1	1
5	1	0	1	1	0
6	1	1	0	0	0
7	1	1	1	1	1

■

Funtzioaren 1. era kanonikoa adierazteko, 1ekoei dagozkien *minterm*-ak batu behar dira. 2. era kanonikoa sortzeko, berriz, funtzioaren 0koei dagozkien *maxterm*-ak biderkatu behar dira.

Beraz, *minterm*-en bidez adierazi nahi badugu *f* funtzioa, 0, 1, 2, 4, 5 eta 7 *minterm*-ak kontuan hartuko ditugu:

$$f = \sum(0, 1, 2, 4, 5, 7) = m_0 + m_1 + m_2 + m_4 + m_5 + m_7 =$$
$$= \bar{c}\bar{b}\bar{a} + \bar{c}\bar{b}a + \bar{c}b\bar{a} + c\bar{b}\bar{a} + c\bar{b}a + cba$$

Eta *maxterm*-en bidez adierazteko, bakarrik 3 eta 6 *maxterm*-ak, funtzioak 0 balio duelako bi kasu horietan:

$$f = \prod(3, 6) = M_3 \cdot M_6 = (c + \bar{b} + \bar{a}) \cdot (\bar{c} + \bar{b} + a)$$

Prozedura berari jarraitu behar zaio bigarren funtzioaren adierazpen kanonikoak lortzeko. Beraz,

- *minterm*-en bidezko adierazpena:

$$g = \sum(2, 4, 7) = m_2 + m_4 + m_7 = \bar{c}b\bar{a} + c\bar{b}\bar{a} + cba$$

- eta *maxterm*-en bidezko adierazpena:

$$g = \prod(0, 1, 3, 5, 6) = M_0 \cdot M_1 \cdot M_3 \cdot M_5 \cdot M_6 =$$
$$= (c + b + a) \cdot (c + b + \bar{a}) \cdot (c + \bar{b} + \bar{a}) \cdot (\bar{c} + b + \bar{a}) \cdot (\bar{c} + \bar{b} + a)$$

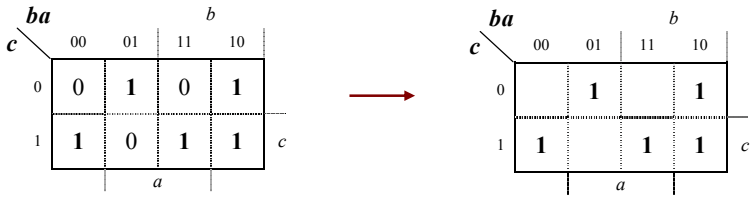
Funtzio bakoitzerako lortutako bi adierazpenak, *minterm*-en eta *maxterm*-en bidezkoak, baliokideak dira.

«

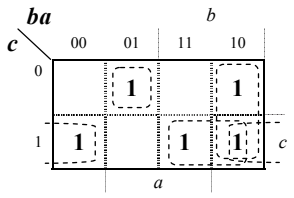
1.5. Ariketa

Lor ezazu f(c,b,a) = ∑(1, 2, 4, 6, 7) funtzioaren adierazpen aljebraiko minimoa Karnaugh-en mapak erabiliz.

Hiru aldagaiko funtzioa denez, dagokion K-mapak $2^3 = 8$ gelaxka izango ditu, *minterm* bakoitzeko bat. K-mapa betetzeko, funtzioak hartzen duen balioa jarriko dugu *minterm* bakoitzari dagokion gelaxkan. Beraz, bost 1eko (*cba* = 001, 010, 100, 110, 111 gelaxketan) eta hiru 0ko (*cba* = 000, 011, 101 gelaxketan) idatzi behar dira K-mapan. Maiz, K-mapa argiagoa izan dadin, 0koak ez dira idazten.



K-mapa horretan, 001 *minterm*-a ezin da sinplifikatu beste *minterm* batekin elkartuz; beraz, funtzioaren adierazpen minimoan agertuko da. Gainerakoak, aldiz, sinplifika daitezke: 010 *minterm*-a 110 *minterm*-arekin elkar daiteke, eta gauza bera egin daiteke 100 eta 110 *minterm*-ekin, eta 111 eta 110 *minterm*-ekin. Mapan ageri denez, hiru aldiz erabili dugu 110 *minterm*-a elkarketak egiteko. Gogoratu: aljebra boolearrean $a + a = a$ da.



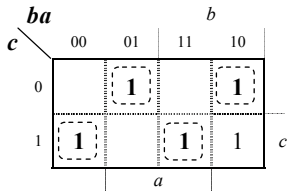
Funtzioaren adierazpen minimoa idazteko, lortu ditugun lau elkarketei dagozkien gaien batuketa egin behar da. Elkarketa bati dagokion gaia osatzeko, batu diren bi *minterm*-etan mantentzen diren aldagaiak biderkatu behar dira, eta aldatzen direnak baztertu. Esaterako, 010 *minterm*-a ($\bar{c}\,b\,\bar{a}$) eta 110 *minterm*-a ($c\,b\,\bar{a}$) elkartzean —batzean—, $b\,\bar{a}$ gaia, sinpleagoa, lortzen da. Elkarketa guztiekin gauza bera eginez gero, hau izango da funtzioaren adierazpen minimoa:

$$f = \bar{c}\,\bar{b}\,a + b\,\bar{a} + c\,\bar{a} + c\,b$$

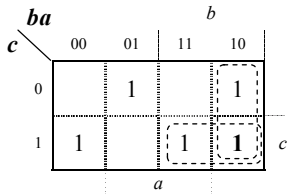
Teorian azaldu den moduan, *and*, *or* eta *not* eragileekin edozein funtzio logiko adieraz daiteke. Horrek ez du galarazten, hala ere, beste eragile logikoen erabilera funtzio bat adierazteko. Esaterako, sistema digitalak eraikitzeko asko erabiltzen den beste eragile bat *xor* funtzioa da (eta haren ezeztapena, *equ* funtzioa).

Hala, bi aldagaiko *xor* ($a\,\bar{b} + \bar{a}\,b$) funtzioari dagokion K-mapak bi 1eko dauzka, 01 eta 10 *minterm*-etan, diagonalean. Era berean, hiru aldagaiko *xor* funtzioak lau 1eko dauzka: 001, 010, 100 eta 111 *minterm*-etan. Erraz froga daiteke hori egia-taula baten bidez.

Ariketako K-mapan, esaterako, aurreko lau 1ekoak ageri dira; beraz, *minterm* horiek bil daitezke *xor* funtzioa sortzeko:



Azkenik, elkartu gabe geratu den 110 *minterm*-a minimizatzeko, hiru aukera daude: goikoarekin (010), eskuinekoarekin (100) eta ezkerrekoarekin (111). Aukeraren arabera, funtzioaren hiru adierazpen "minimo" baliokide lortuko ditugu. Esaterako,



Hau da: $f = (a \oplus b \oplus c) + b\,\bar{a}$ edo $f = (a \oplus b \oplus c) + c\,b$

Laburbilduz: ariketaren lehen partean lortu dugun adierazpena minimoa da, oinarrizko eragiketa kopurua (*and*, *or* eta *not*) kontuan hartuta. Hala ere, beste eragile batzuk erabiliz gero (adibidez, *xor* funtzioa), adierazpen "minimo" desberdinak lortuko ditugu.

«

1.6. Ariketa

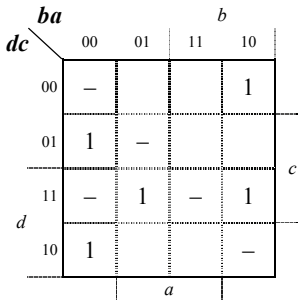
Minimiza ezazu $f = \sum(2, 4, 8, 13, 14) + d(0, 5, 10, 12, 15)$ funtzio logikoa Karnaugh-en mapa baten bidez.

Funtzioak bost 1eko ditu eta, horrez gain, badaude zehaztu gabeko bost *minterm*, hau da, 0kotzat edo 1ekotzat har daitezkeen gaiak (*d = don't care*).

Bestalde, ez da zehazten funtzioaren aldagai kopurua. Beraz, Karnaugh-en mapa marraztu ahal izateko, kalkulatu behar da kopuru hori. Ikusten denez, *minterm* handiena 15a da; ondorioz, gutxienez 4 aldagaiko funtzioa izango da ($2^4 = 16 \geq 15$). Hortaz,

$$f(d, c, b, a) = \sum(2, 4, 8, 13, 14) + d(0, 5, 10, 12, 15)$$

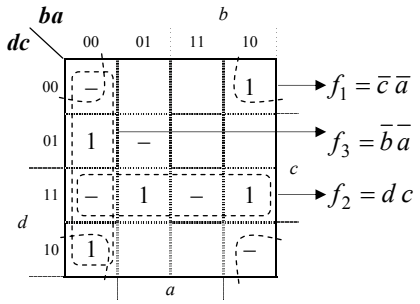
4 aldagaiko mapa erabili behar dugu, beraz, eta hor idatzi, batetik, funtzioaren 1ekoak, eta, bestetik, zehaztu gabeko gaiak ("–" edo "x" ikurra erabiliz):



Has gaitezke funtzioa minimizatzen, *minterm*-ak elkartuz. Beti bezala, elkartzeko aukera bakarra (edo, oro har, gutxien) duten *minterm*-ekin hasiko gara. Hala, bada, 0010 *minterm*-a gainerako hiru erpinetako *minterm*-ekin elkar daiteke; horretarako, 0000 eta 1010 zehaztu gabeko gaiak 1ekotzat hartu behar dira.

Era berean, hirugarren errenkadako *minterm*-ak elkar daitezke, 1100 eta 1111 *minterm*-ak 1ekotzat hartuz.

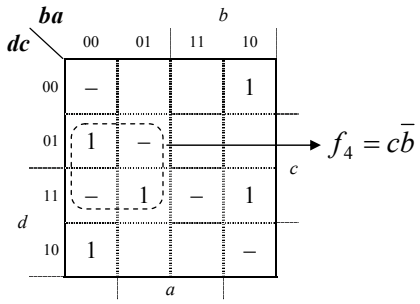
Azkenik, elkartu gabe gelditu den *minterm* bakarra (0100) lehenengo zutabekoekin elkar daiteke.



Guztira, hiru elkarketa sortu ditugu, lau *minterm*-ekoak. 4 *minterm*-eko elkarketetan, bi aldagai desagertzen dira, eta, ondorioz, funtzioak bi aldagaiko hiru gai izango ditu:

$$f(d,c,b,a) = f_1 + f_2 + f_3 = \overline{c}\,\overline{a} + d\,c + \overline{b}\,\overline{a}$$

Hori ez da, baina, 1eko guztiak elkartzeko era bakarra. Izan ere, f_3 elkarketaren ordez, beste hau egin zitekeen, 0101 eta 1100 *minterm*-ak 1ekotzat hartuz:



Ondorioz, honela ere idatz daiteke funtzioa:

$$f(d,c,b,a) = f_1 + f_2 + f_4 = \overline{c}\,\overline{a} + dc + c\,\overline{b}$$

1.7. Ariketa

$f_1 = \overline{d}\,\overline{c}\,a + \overline{d}\,c\,b\,a + d\,c\,b + d\,b\,a$ funtzioa sinplifikatzeko asmoz, hainbat zehaztu gabeko gai erabili dira. Emaitza $f_2 = ba + dc + \overline{d}a$ izan da. Zein zehaztu gabeko gai erabili dira sinplifikazioa egiteko?

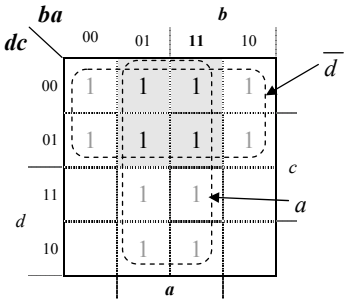
■

Bi adierazpideen arteko desberdintasunak ikusteko, haien egia-taulak edo K-mapak egin eta konpara ditzakegu. Desberdintasunek zehaztu gabeko gaiak izan behar dute.

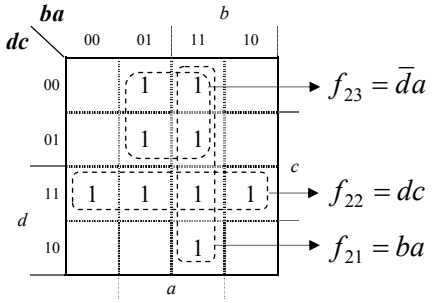
Has gaitezen adierazpen minimoarekin: $f_2 = b\,a + d\,c + \overline{d}\,a$. K-mapa betetzeko, funtzioaren 1ekoak lortu behar ditugu; batuketa bat denez, gai bakoitzari dagozkion 1ekoak.

Lehenengo gaia $b\,a$ da, bi aldagaikoa; 4 aldagaiko funtzioa denez, gai horrek lau *minterm* biltzen ditu (ageri ez diren bi aldagaien konbinazio bakoitzeko bat): $b\,a = \overline{d}\,\overline{c}\,b\,a + \overline{d}\,c\,b\,a + d\,\overline{c}\,b\,a + d\,c\,b\,a$. Gauza bera gertatzen da funtzioaren beste bi gaiekin: $d\,c$ eta $\overline{d}\,a$.

Badago beste era bat gai batek ordezkatzen dituen *minterm*-ak ezagutzeko: K-mapan bertan, gaiaren aldagaiei dagozkien errenkaden eta zutabeen arteko ebakidurako *minterm*-ak dira, hain zuzen ere. Esaterako, hauek dira $\overline{d}\,a$ gaiari dagozkion lau *minterm*-ak:



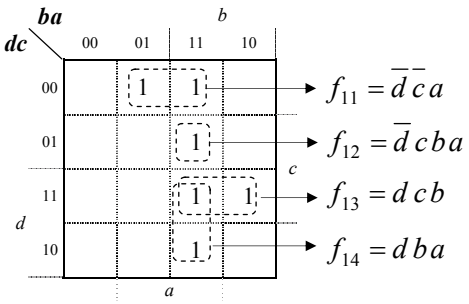
Prozesu hori errepikatzen badugu gai guztiekin, f_2 funtzioaren K-mapa hau lortuko dugu:



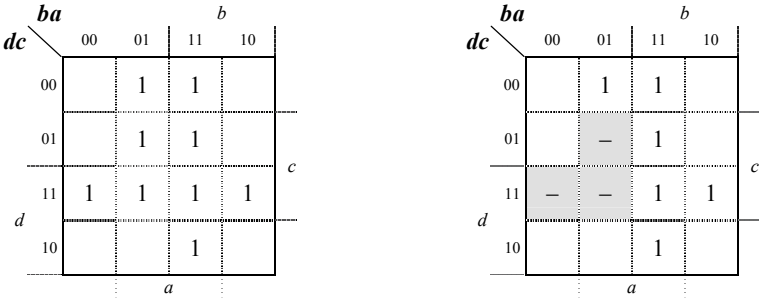
Beraz, f_2 funtzioak bederatzi 1eko ditu.

Jatorrizko funtzioarekin, $f_1 = \overline{d}\,\overline{c}\,a + \overline{d}\,c\,b\,a + d\,c\,b + d\,b\,a$, gauza bera egin behar da. Kasu honetan, gaiak 3 edo 4 aldagaikoak dira. 3 aldagaiko gaiei bina *minterm* dagozkie; esaterako, $d\,c\,b = d\,c\,b\,\overline{a} + d\,c\,b\,a$. 4 aldagaiko gaia, berriz, *minterm* bakarra da.

Hau da, beraz, f_1 funtzioaren K-mapa:



Lortutako bi mapak konparatzen baditugu, garbi ageri dira funtzioa minimizatzeko erabili diren zehaztu gabeko gaiak: bigarren mapan hutsak egonik, lehenengoan 1ekoak dituzten gelaxkak. Hau da, 1ekotzat hartu dira 0101, 1100 eta 1101 *minterm*-ak.

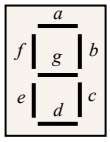


Ondorioz, honela zegoen definituta f_1 funtzioa:

$$f_1 = \overline{d}\,\overline{c}\,a + \overline{d}\,c\,b\,a + d\,c\,b + d\,b\,a \; + d(5,\, 12,\, 13)$$

«

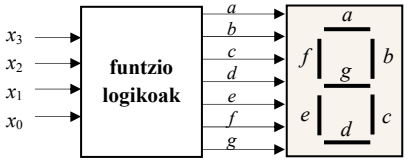
1.8. Ariketa



Sistema digital askotan, ikusgailu bereziak erabiltzen dira 0tik 9ra arteko zenbakiak erakusteko. Ikusgailu horiek "7 segmentuzko digituak" deitzen dira, argitzen diren zazpi segmentu (diodo) dituztelako, irudian ageri den moduan. Adibidez, a = 1 denean, a diodoa pizten da; 0 denean, berriz, itzali egiten da.

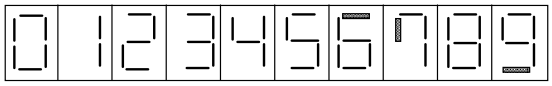
0tik 9rako zenbakiak bitarrez adierazteko, 4 bit behar dira; esate baterako, $X = x_3\ x_2\ x_1\ x_0$. Zenbakia grafikoki adierazteko, aldiz, 7 biteko kode bat behar da: a, b, c, d, e, f eta g segmentuetakoa, hain zuzen ere.

Hau egin behar da ariketa honetan: idatzi a, b... eta g funtzio logikoen adierazpen minimoak, x_3, x_2, x_1, x_0 aldagaien arabera (funtzio horiek gauzatzen dituen zirkuituari "BCD – 7 segmentu" deskodegailua deritzo).

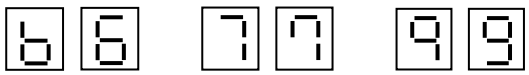


■

a, b, c, d, e, f eta *g* zazpi funtzio logikoen adierazpenak kalkulatu aurretik, zenbaki bakoitza nola irudikatzen den aztertu behar dugu. Izan ere, era bat baino gehiago dago digituak adierazteko, eta horietako bat estandarra da. Ariketa honetan, hala ere, askatasun handiz jokatuko dugu digituak irudikatzean. Esaterako, honela adieraz daitezke 10 digituak:



Irudikatze horretan, aukera bikoitza utzi dugu 6, 7 eta 9 digituak adierazteko:



Beraz, 6 zenbakiaren kasuan, berdin zaigu *a* segmentuak zein balio hartzen duen: 0 edo 1. $a = 0$ bada, aurreko irudiko ezkerreko 6a ikusiko da; $a = 1$ bada, eskuinekoa. Gauza bera gertatzen da 7 digituaren *f* segmentuarekin, eta 9 digituaren *d* segmentuarekin. Aukera bikoitz horiek erabiliko ditugu gero, funtzioak minimizatzeko orduan.

7 segmentuen egia-taulak sortu behar ditugu, hau da, zer segmentu aktibatu behar diren digitu bakoitzerako. Taulako ezkerraldean, beraz, zenbakien kode bitarrak jarriko ditugu eta eskuinaldean ikusgailuan aktibatu behar diren segmentuak. Adibidez, 0 zenbakia irudikatzeko, *a, b, c, d, e* eta *f* segmentuak aktibatu behar dira, eta *g* segmentua desaktibatu.

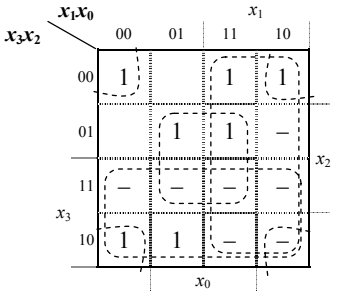
Azken ohar bat. 4 bitekin 16 zenbaki desberdin adieraz daitezke, 0tik 15era. Hala ere, 10 digitu hamartar baino ez dago; beraz, sarrerako hainbat konbinazio (10etik 15era) ez dira inoiz agertuko: zehaztu gabeko gaiak dira.

Hauek dira 7 segmentuen egia-taulak:

x_3	x_2	x_1	x_0	***a***	***b***	***c***	***d***	***e***	***f***	***g***
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	–	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	–	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	–	0	1	1
1	0	1	0	–	–	–	–	–	–	–
1	0	1	1	–	–	–	–	–	–	–
1	1	0	0	–	–	–	–	–	–	–
1	1	0	1	–	–	–	–	–	–	–
1	1	1	0	–	–	–	–	–	–	–
1	1	1	1	–	–	–	–	–	–	–

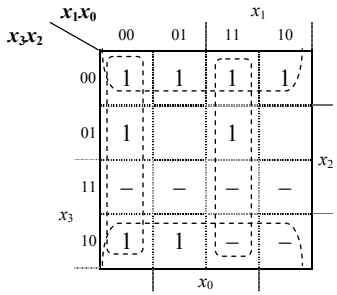
Egia-taula definituta, funtzio bakoitzaren adierazpen minimoa lortuko dugu; esaterako, K-mapak erabiliz.

Hau da *a* funtzioaren K-mapa. *a* funtzioaren 1ekoak eta zehaztu gabeko gaiak (–) idatzi ditugu mapan, eta ahal diren elkarketa handienak egin ditugu, funtzioa minimizatzeko.

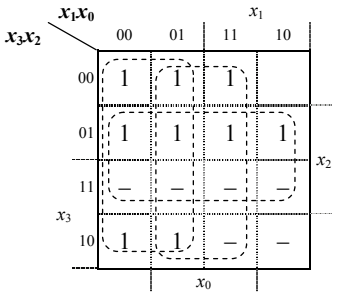


Beraz, $a = \bar{x}_2\bar{x}_0 + x_1 + x_2x_0 + x_3$ da.

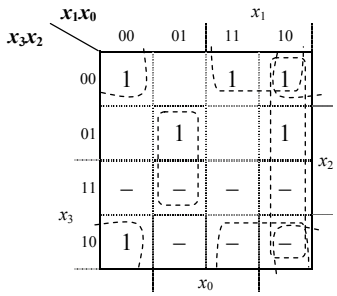
Gainerako funtzioak (*b*, *c*, *d*, *e*, *f* eta *g*) modu berean ateratzen dira. Gogoratu: zehaztu gabeko gaiak 1ekotzat zein 0kotzat har daitezke, komenigarriena dena funtzioaren adierazpen aljebraikoa minimizatzeko.



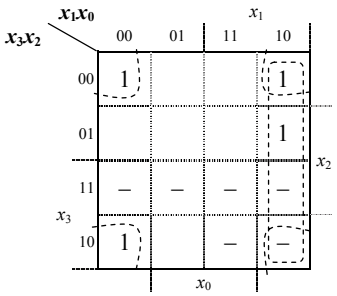
$$b = \overline{x}_1\overline{x}_0 + x_1x_0 + \overline{x}_2$$



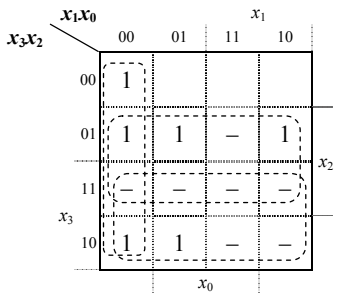
$$c = \overline{x}_1 + x_0 + x_2$$



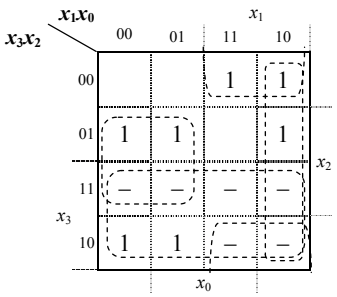
$$d = \overline{x}_2\overline{x}_0 + x_2\overline{x}_1x_0 + \overline{x}_2x_1 + x_1\overline{x}_0$$



$$e = \overline{x}_2\overline{x}_0 + x_1\overline{x}_0$$



$$f = \overline{x}_1\overline{x}_0 + x_2 + x_3$$



$$g = \overline{x}_2x_1 + x_2\overline{x}_1 + x_3 + x_1\overline{x}_0$$

7 funtzioen adierazpen minimoak lortu ditugu. Hala ere, funtzio multzo baten adierazpen minimoa ez dator bat, askotan, funtzio bakoitzaren adierazpen minimoen multzoarekin. Aukeran, funtzio batean erabiltzen diren gaiak besteetan ere erabiltzea interesgarria da, nahiz eta agian gai horiek ez izan minimoak beste funtzioetan. Hala egiten denean, funtzio guztiak osatzeko (eta gero eraikitzeko) behar den gai kopurua txikiagoa izango da.

«

1.4. ARIKETAK

1.1. Aljebra boolearraren axiomak eta teoremak erabiliz, sinplifika itzazu honako adierazpen hauek:

(1) $f = \overline{a}\,\overline{b} + \overline{a}\,b\,\overline{c} + a\,c + \overline{b}\,c + \overline{a}\,b\,c$

(2) $g = (\overline{a} + \overline{b})(a\,b + c)$

(3) $h = a\,(b + c\,(b + a))$

1.2. Froga itzazu berdintza hauek, batetik, egia-taulak erabiliz, eta, bestetik, aljebra boolearraren axiomak eta teoremak erabiliz:

(1) $a\,b\,c + \overline{a}\,b + a\,\overline{c} = b + a\,\overline{c}$

(2) $\overline{b}\,(a \oplus c) + a\,(b \oplus c) = \overline{b}\,c + \overline{c}\,a$

(3) $(a + b)(\overline{a} + c)(b + c) = (a + b)(\overline{a} + c)$

1.3. Adieraz itzazu honako funtzio hauen osagarriak, eta minimizatu adierazpenak:

(1) $f = \overline{a}(b + \overline{c}) + b + c\,\overline{d}$

(2) $g = a + a\,c + c(\overline{d} + e)$

(3) $h = a\,\overline{b} + \overline{c}\,(b + d)$

1.4. Lor itzazu honako funtzio hauen adierazpen minimoak Karnaugh-en mapak erabiliz.

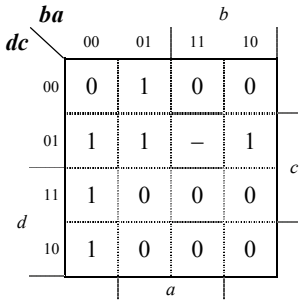
(1) $f(c, b, a) = \overline{c}\,\overline{b}\,\overline{a} + a\,\overline{b} + b\,\overline{a} + c(\overline{a} + \overline{b})$

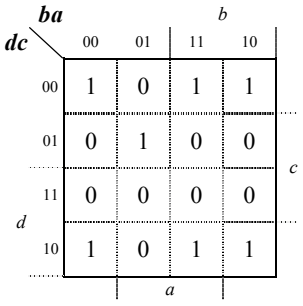
(2) $f(d, c, b, a) = a\,d + a\,\overline{b}\,c + \overline{a}\,b\,c + a\,b\,\overline{d}$

(3) $f(d, c, b, a) = \sum(0, 13, 14, 15) + d(1, 2, 3, 9, 10, 11)$

(4) $f(d, c, b, a) = \sum(1, 3, 5, 8, 9, 11, 15) + d(2, 13)$

1.5. Minimiza itzazu K-mapen bidez adierazitako bi funtzio hauek.



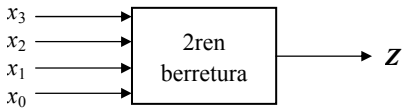


1.6. Bi funtzio logikoen egia-taulak kontuan hartuz, idatzi funtzio bakoitzaren era kanonikoa; gero, minimizatu adierazpenak K-mapak erabiliz.

d	*c*	*b*	*a*	***f***
0	0	0	–	1
0	0	1	0	0
0	0	1	1	1
0	1	–	–	–
1	0	0	0	1
1	0	0	1	1
1	–	1	0	1
1	0	1	1	0
1	1	0	0	–
1	1	0	1	0
1	1	1	1	1

c	*b*	*a*	***g***
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

1.7. Irudiko zirkuituaren sarrerek —x_3, x_2, x_1, x_0— 4 biteko zenbaki arrunt bat adierazten dute, bitar hutsez. Z irteerak, berriz, sarrerako zenbakia 2ren berretura den ala ez adierazi beharko du. Idatz ezazu funtzioaren egia-taula eta lor ezazu adierazpen minimoa Karnaugh-en mapak erabiliz.



Errepikatu aurrekoa, baina murriztapen hau kontuan hartuta: sarrerako zenbakia digitu hamartar bat da: 0tik 9ra.

1.8. 4 biteko kodeak —X (x_3, x_2, x_1, x_0)— prozesatzen dira sistema digital batean, eta, emaitza gisa, bit bateko hiru funtzio sortzen dira: BAT, ZERO eta BERDIN izenekoak. BAT funtzioa aktibatzen da sarrera-kodearen 1ekoen kopurua 0koena baino handiagoa denean; ZERO aktibatzen da 0koen kopurua 1ekoena baino handiagoa denean; eta BERDIN bi kopuruak berdinak direnean.
Idatz itzazu hiru funtzio horien egia-taulak, eta eman haien adierazpen aljebraiko minimoak.

1.9. Sistema digital batek detektatu behar du sarrerako zenbakiak (4 bitekoak) lehenak diren ala ez. Idatzi funtzio horri dagokion egia-taula eta eman funtzioaren adierazpen minimoa.

2. kapitulua

SISTEMA DIGITALETAKO OINARRIZKO GAILUAK: ATE LOGIKOAK

Aurreko kapituluan, zirkuitu digitalen oinarrizko teoria matematikoa aurkeztu dugu: aljebra boolearra, oinarrizko eragiketa logikoak, funtzioen adierazpen aljebraikoa eta adierazpenen minimizazioa. Kapitulu honetan, funtzio logikoak eraikitzeko erabiltzen diren oinarrizko zirkuitu digitalak aztertuko ditugu: ate logikoak. Sistema digitalen oinarrizko osagaiak dira ateak, logika boolearraren oinarrizko eragiketak gauzatzen dituzte, eta zirkuitu konplexuagoak eraikitzeko eta ulertzeko oinarria dira. Hori baino lehen, aztertu eta erabaki beharko dugu nola gauzatu aljebra boolearraren bi balioak, 1 eta 0.

2.1. BI BALIO LOGIKOEN GAUZATZEA

Aljebra boolearrak bi balio baino ez ditu erabiltzen: 1 (egiazkoa, aktibatuta) eta 0 (faltsua, desaktibatuta). Edozein zirkuitu eraiki baino lehen, erabaki bat hartu behar da: zer magnitude fisiko erabiliko dugu aljebraren bi balioak fisikoki adierazteko? Magnitude horren zein bi balio erabiliko ditugu 1a eta 0a adierazteko?

Teknologia asko eta oso desberdinak erabil daitezke sistema logikoak eraikitzeko. Esaterako, zirkuitu mekanikoak, hidraulikoak, optikoak eta abar eraiki daitezke logika boolearraren araberako funtzionamendua erakuts dezaten. Baina, erabilienak, dudarik gabe, zirkuitu elektronikoak dira.

Zirkuitu elektronikoen oinarrizko osagaia transistorea da. Ezin dugu azaldu liburu honetan transistoreen funtzionamendua zehatz-mehatz, baina nahiko modu sinplean irudika dezakegu haien portaera. Bi "egoera" ditu transistoreak: korronte elektrikoa eroaten du, edo ez du eroaten; hots, etengailu elektronikoa da transistorea (ikus E2 eranskina).

Transistoreak erabiltzen baditugu zirkuitu logikoak eraikitzeko, magnitude "naturalenak" balio logikoak adierazteko bi dira: korronte elektrikoa eta tentsio elektrikoa. Oro har, tentsio elektrikoa erabiltzen da. Beraz, bi tentsio-balio behar ditugu 1a eta 0a adierazteko, berdin da zein bi balio. Bi tentsio-balio horiei H (*high,* altua) eta L (*low*, baxua) deitzen zaie; edozein tentsio-balio izan daitezke, baina, eskuarki, L tentsioa 0 volt da, eta H tentsioa 5 volt (edo 3 volt, gero eta tentsio baxuagoak erabiltzen baitira).

Beraz, transistoreek osaturiko zirkuituetako L eta H tentsioak erabiliko ditugu 1 eta 0 balio logikoak adierazteko. Azken puntu bat erabaki behar dugu. Zein da 1a eta zein da 0a? Bi aukera ditugu:

H → 1	edo	H → 0
L → 0		L → 1

Sistema digital gehienetan, bi aukerak batera erabiltzen dira: zenbait seinaletan, lehenengo baliokidetza erabiltzen da, eta, beste batzuetan, bestea. Hala egiten denean, **logika mistoa** (*mixed logic*) erabiltzen dela esaten da. Lehenengo aukera erabiltzen duten seinaleak (H → 1; L → 0) logika positiboan daudela esan ohi da; bigarren aukera erabiltzen duten seinaleak, aldiz, logika negatiboan.

Zein baliokidetza erabiltzen den argi eta garbi adierazteko, honako irizpide hau erabiltzen da:

- Seinalea logika negatiboan interpretatu behar bada (H → 0; L → 1), gehitu seinalearen izenari **.L** atzizkia; bestela, erabili izen soila (edo gehitu .H atzizkia).
- Zirkuituaren sarrera edo irteera logika negatiboan interpretatu behar bada, gehitu **zirkulutxo** bat.

Esaterako,

Honela interpretatu behar dira bi seinaleak, *batu*, logika positiboan, eta *prest*, logika negatiboan:

batu = H tentsioa → 1 / aktibatuta
batu = L tentsioa → 0 / desaktibatuta

eta

prest.L = H tentsioa → 0 / desaktibatuta
prest.L = L tentsioa → 1 / aktibatuta

Laburbilduz. Zirkuitu digitalak transistoreen bidez osatzen dira eta bi tentsio-balio prozesatzen dituzte: H eta L. Tentsio horien interpretazio logikoa bikoitza izan daiteke. Besterik ez bada adierazten, H → 1 eta L → 0 izango ditugu; bestela, seinalearen izenak .L badarama edo zirkuituaren sarrerak edo irteerak zirkulutxo bat badarama, orduan H → 0 eta L → 1 izango dira.

2.2. OINARRIZKO ERAGIKETEN GAUZATZEA: *OR*, *AND* ETA *NOT* FUNTZIO LOGIKOAK

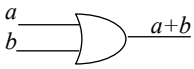
Aurreko kapituluan esan dugunez, hiru dira aljebra boolearraren oinarrizko eragiketa logikoak —*and*, *or* eta *not*— eta eragiketa horiekin edozein funtzio logiko eraiki daiteke: sistema "oso" bat osatzen dute. Beraz, hiru eragiketa horiek gauzatzen dituzten zirkuitu digitalak behar ditugu, gero horiek elkartuz edozein funtzio logiko eraiki ahal izateko. Oinarrizko zirkuitu digital horiei **ate logikoak** deritze. Hurrengo azpiataletan, ate erabilienen funtzionamendua aztertuko dugu.

2.2.1. *or* eragiketa gauzatzeko ateak

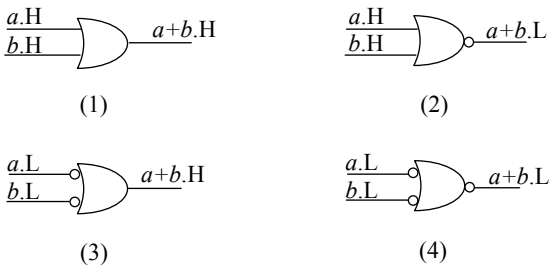
Aurreko kapituluan definitu dugun moduan, hau da *or* eragiketaren edo batuketa logikoaren egia-taula:

a	*b*	***a* + *b***
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Beraz, irteera aktibatzen da sarreraren bat aktibatzen bada. Hainbat zirkuitu daude eragiketa hori gauzatzeko, sarreren eta irteeraren logikaren arabera, sarrera kopuruaren arabera, eta abar. Dena den, guztiek forma bereko sinbolo grafiko hau dute:



Irudiko zirkuitua bi sarrerakoa eta irteera batekoa da. Lehen ikusi dugun moduan, bi aukera daude sarrerak eta irteerak interpretatzeko: logika positiboa edo logika negatiboa. Beraz, hiru seinale ditugunez, *a* eta *b* sarrerak eta *a*+*b* irteera, 8 aukera desberdin daude. Zortzi aukera horietatik, 4 hauek dira erabilienak:



2.1. irudia. *or* eragiketa logikoaren lau gauzatze ohikoak.

Azter ditzagun 2.1. irudiko zirkuituak banan-banan. Izen komertzial desberdinak dituzte, eta agian ezusteko izenak ere, baina 2.2.2. atalean argituko da zioa.

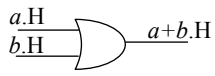
1. Bi sarrerak eta irteera logika positiboan

Eragiketaren taula logikotik abiatuta, zirkuituaren taula fisikoa lor daiteke; hots, H eta L tentsioak erakusten dituen taula. "Itzulpen" hori egiteko, seinaleen logika kontuan hartu behar da. Kasu honetan, sarreretan zein irteeran, 1 → H eta 0 → L izango dira.

2.2. irudian ageri da ate horren funtzionamendua; hots, prozesatzen diren tentsioak eta sortzen diren emaitzak.

eragiketaren taula logikoa

a	*b*	***a* + *b***
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**



OR zirkuituaren taula fisikoa

a.H	*b*.H	***a*+*b***.H
L	L	L
L	H	H
H	L	H
H	H	H

2.2. irudia. *or* eragiketa logikoaren gauzatzea: OR atea.

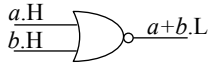
Taula fisiko hori gauzatzen duen zirkuituari **OR** atea deitzen zaio (adi! OR hori izen komertziala baino ez da). Aukera desberdin asko dago ate hori transistoreen bidez eraikitzeko, eta ez ditugu hemen ikusiko.

2. Bi sarrerak logika positiboan eta irteera logika negatiboan

Aurreko prozedura berari jarraitu behar zaio, atearen taula fisikoa lortzeko, eta emaitza 2.3. irudikoa da.

eragiketaren taula logikoa

a	*b*	***a* + *b***
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**



NOR zirkuituaren taula fisikoa

a.H	*b*.H	***a*+*b***.L
L	L	H
L	H	L
H	L	L
H	H	L

2.3. irudia. *or* eragiketa logikoaren gauzatzea: NOR atea.

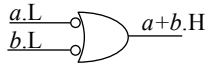
Taula fisiko hori gauzatzen duen zirkuituari **NOR** atea deitzen zaio (izen komertziala). Sinboloa *or* eragiketarena da, eta irteerak zirkulutxo bat darama, logika negatiboan dagoela adierazteko.

3. Bi sarrerak logika negatiboan eta irteera logika positiboan

Hau da 2.1. irudiko hirugarren zirkuituaren portaera adierazten duen taula fisikoa:

eragiketaren taula logikoa

a	*b*	***a*** **+** ***b***
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**



NAND zirkuituaren taula fisikoa

a.L	*b*.L	***a*+*b***.H
H	H	L
H	L	H
L	H	H
L	L	H

2.4. irudia. *or* eragiketa logikoaren gauzatzea: NAND atea.

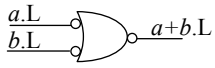
Taula fisiko hori gauzatzen duen zirkuituari **NAND** atea deitzen zaio (izen komertziala). Berriro ere, sinboloa *or* eragiketarena da; kasu honetan, hala ere, sarrerak logika negatiboan daude (bi zirkulutxoen bidez adierazita).

4. Bi sarrerak eta irteera logika negatiboan

Azkenik, hau da 2.1. irudiko 4. zirkuituaren portaera adierazten duen taula fisikoa:

eragiketaren taula logikoa

a	*b*	***a*** **+** ***b***
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**



AND zirkuituaren taula fisikoa

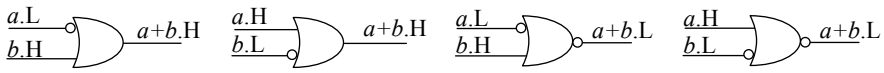
a.L	*b*.L	***a*+*b***.L
H	H	H
H	L	L
L	H	L
L	L	L

2.5. irudia. *or* eragiketa logikoaren gauzatzea: AND atea.

Taula fisiko hori gauzatzen duen zirkuituari **AND** atea deritzo (adi! izen komertziala da hori, AND deitzen bada ere, *or* eragiketa egiten du).

Aztertu ditugun lau zirkuitu horien tentsio-taulak desberdinak dira; hots **zirkuitu fisiko desberdinak** dira. Hala ere, laurek ***or*** eragiketa logikoa gauzatzen dute, sarreren eta irteeraren logika desberdina delarik.

Irakurleak nahi badu, analisi bera egin dezake aztertu ez ditugun beste lau ateekin (eskuarki, ez ditugu aukera horiek erabiliko).

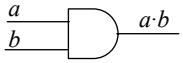


2.2.2. *and* eragiketa gauzatzeko ateak

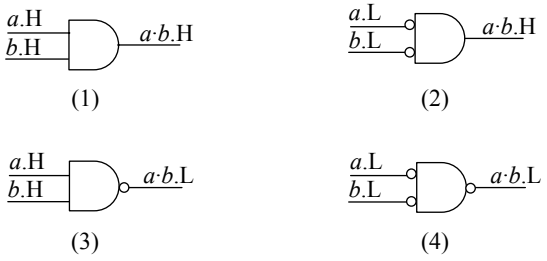
Hau da *and* eragiketaren edo biderketa logikoaren egia-taula, aurreko kapituluan ikusi dugun moduan:

a	*b*	***a* · *b***
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Beraz, irteera aktibatzen da bi sarrerak 1 direnean. *or* funtzioarekin gertatzen den moduan, *and* eragiketa gauzatzen duten hainbat zirkuitu dago, sarreren eta irteeraren logikaren arabera, sarrera kopuruaren arabera, eta abar. Berriz ere, guztiek forma bereko sinbolo grafiko hau dute:



Irudiko zirkuitua bi sarrerakoa eta irteera batekoa da. Hiru seinale ditugunez, *a* eta *b* sarrerak eta *a·b* irteera, 8 aukera desberdin daude, seinale horien logikaren arabera. Zortzi aukera horietatik, 4 hauek dira erabilienak:



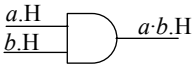
2.6. irudia. *and* eragiketa logikoaren lau gauzatze ohikoak.

Azter ditzagun 2.6. irudiko zirkuituak banan-banan. Analisia *or* ateetarako egin dugun bera da.

1. Bi sarrerak eta irteera logika positiboan

eragiketaren taula logikoa

a	*b*	***a · b***
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**



AND zirkuituaren taula fisikoa

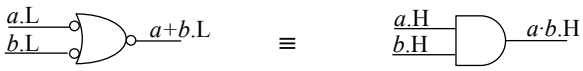
a.H	*b*.H	***a · b***.H
L	L	L
L	H	L
H	L	L
H	H	H

2.7. irudia. *and* eragiketa logikoaren gauzatzea: AND atea.

Taula fisiko hori gauzatzen duen zirkuituari **AND** atea deitzen zaio.

2.5. irudiko atea ere —sarrerak eta irteera logika negatiboan dituen *or* funtzioa— AND deitzen dela aipatu dugu. Orain uler daiteke izen horren arrazoia. Konparatzen baditugu bi **taula fisikoak** (2.5. eta 2.7. irudietakoak) berdinak direla ikusiko dugu (taulako lerroak ordena desberdinean idatzita badaude ere). Beraz, zirkuitu fisiko bera da: AND zirkuitua.

Garrantzitsua da hori. Hala izanik, **zirkuitu bakarrarekin *or* zein *and* eragiketak egin ditzakegu;** nahikoa da interpretatzea sarrerak eta irteerak logika jakin baten arabera.

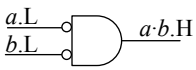


2. Bi sarrerak logika negatiboan eta irteera logika positiboan

Aurreko prozedura berari jarraitu behar zaio, atearen taula fisikoa lortzeko. Hau da emaitza:

eragiketaren taula logikoa

a	*b*	***a · b***
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**



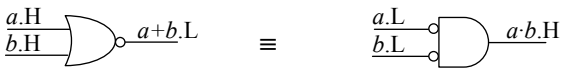
NOR zirkuituaren taula fisikoa

a.L	*b*.L	***a · b***.H
H	H	L
H	L	L
L	H	L
L	L	H

2.8. irudia. *and* eragiketa logikoaren gauzatzea: NOR atea.

Taula fisiko hori gauzatzen duen zirkuituari **NOR** atea deitzen zaio, eta erraza da egiaztatzea 2.3. irudiko zirkuitu bera dela, bi taula fisikoak berdinak direlako.

Beraz, baliokidetza hau dugu:

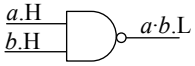


3. Bi sarrerak logika positiboan eta irteera logika negatiboan

Hau da hirugarren zirkuituaren portaera adierazten duen taula fisikoa:

eragiketaren taula logikoa

a	*b*	***a · b***
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**



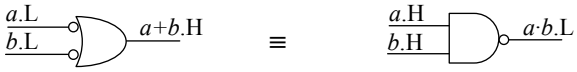
NAND zirkuituaren taula fisikoa

a.H	*b*.H	***a · b***.L
L	L	H
L	H	H
H	L	H
H	H	L

2.9. irudia. *and* eragiketa logikoaren gauzatzea: NAND atea.

Taula fisiko hori gauzatzen duen zirkuituari **NAND** atea deitzen zaio. 2.4. irudiko taula fisikoa eta 2.9. irudikoa konparatuta, kasu honetan ere ate bera dela egiaztatzen da.

Beraz,

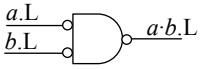


4. Bi sarrerak eta irteera logika negatiboan

Hau da 2.6. irudiko 4. ateari dagokion portaera fisikoa:

eragiketaren taula logikoa

a	*b*	***a · b***
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**



OR zirkuituaren taula fisikoa

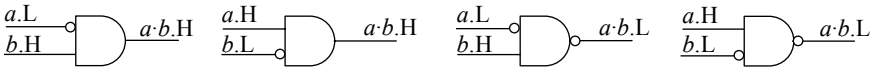
a.L	*b*.L	***a · b***.L
H	H	H
H	L	H
L	H	H
L	L	L

2.10. irudia. *and* eragiketa logikoaren gauzatzea: OR atea.

Taula fisiko hori gauzatzen duen zirkuituari **OR** atea deitzen zaio. Honezkero, badakigu zergatik: 2.2. irudiko OR atearen taula fisikoa eta aurrekoa berdinak dira; zirkuitu bera da.

Beraz,

Irakurleak nahi badu, analisi bera egin dezake aztertu ez ditugun beste lau ateekin (lehen bezala, ez ditugu aukera hauek erabiliko).



Laburbilduz. *or* eta *and* eragiketa logikoak gauzatzen dituzten zirkuituak analizatuz, hau aurkitu dugu: zirkuitu fisiko bakar batekin bi eragiketa logikoak egin daitezke, modu egokian interpretatzen badira sarreren eta irteeraren balioak. Laburpen moduan, hona hemen bikoiztasun hori kasu jakin batean: NAND atea. 2.11. irudian, NAND atearen portaeraren interpretazio biak ageri dira.

NAND zirkuituaren taula fisikoa

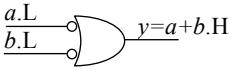
a	*b*	***y***
L	L	H
L	H	H
H	L	H
H	H	L

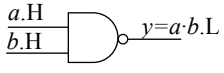
a.L	*b*.L	***y***.H
1	1	1
1	0	1
0	1	1
0	0	0

or funtzio logikoaren egia-taula

a.H	*b*.H	***y***.L
0	0	0
0	1	0
1	0	0
1	1	1

and funtzio logikoaren egia-taula





2.11. irudia. NAND atearen bi interpretazio logiko erabilienak.

NAND atearen funtzionamendua bakarra da, noski, baina funtzionamendu horren interpretazio bikoitza egin daiteke. Sarrerak logika positiboan (H → 1 eta L → 0) eta irteera logika negatiboan (H → 0 eta L → 1) interpretatzen badira, orduan *and* eragiketa logikoa egiten du; aldiz, sarrerak logika negatiboan eta irteera logika positiboan hartuta, *or* funtzio logikoa egiten du.

2.2.3. *not* eragiketa gauzatzeko ateak

Honela definitu dugu aurreko kapituluan *not* eragiketa (ezeztapena):

a	$\overline{a}$
0	**1**
1	**0**

or eta *and* eragiketekin gertatu den moduan, aukera bat baino gehiago dugu funtzio hori gauzatzeko; hain zuzen ere, 4 aukera ditugu, sarreraren eta irteeraren logikaren arabera. Azter ditzagun horietako bi; lehenengoa, sarrera logika positiboan eta irteera logika negatiboan dituena, eta bigarrena, alderantziz, sarrera logika negatiboan eta irteera logika positiboan:

a.H	$\overline{a}$.L
L	L
H	H

a.L	$\overline{a}$.H
H	H
L	L

Bi tauletan ikusten da, argi eta garbi, tentsioaren balio bera (L edo H) dagoela sarreran eta irteeran. Horrek esan nahi du ezeztapena egiteak ez dakarrela inongo aldaketa fisikorik tentsio-balioetan, baldin eta sarrera eta irteera alderantzizko logiketan interpretatzen badira. Hau da, seinale fisiko bera a aldagaia edo $\overline{a}$ aldagaia izan daiteke, erabilitako logikaren arabera.

Ondorioz, tentsio-aldaketarik ez dagoenez, ez dugu zirkuitu fisikorik behar seinale bat ezeztatzeko. Dena den, seinalearen interpretazioa aldatu dugunez gero, egokia da marka bat gehitzea, aldaketa hori egin dela zehazteko. Hala, 2.12. irudiko sinboloak erabiliko ditugu hori adierazteko.

2.12. irudia. Seinale baten bi interpretazioak alderantzizko logiketan.

Adi! 2.12. irudiko ikurrak interpretazio-aldaketa bat egin dela gogoratzeko marka soilak dira; ez dago zirkuitu fisikorik. Zirkuituen analisia errazteko, ordea, metodologia ona da marka hori gehitzea ezeztapen bat egiten den bakoitzean. Ikur horiei "inbertsore (alderanzgailu) logikoa" deituko diegu.

Seinaleak bi logiketan erabiltzen ditugunez gero, .H eta .L, ez da beharrezkoa zirkuitu bat erabiltzea ezeztapen logikoa egiteko. Baina, horrek eskatzen du seinaleak prozesatzea batzuetan .H gisa eta beste batzuetan .L gisa, ateen sarreren eta irteeren logikaren arabera. Beraz, nola lortu *a*.L seinalea *a*.H seinaletik abiatuta (edo alderantziz)?

Hauxe da aldaketa horren taula:

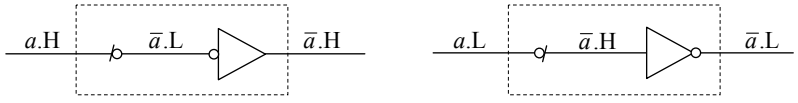
a	***a***.H	***a***.L
0	L	H
1	H	L

Balio logikoa mantentzen da, baina tentsioa aldatu behar da, eta, horretarako, zirkuitu bat beharko da: ate bat. Gauza bera gertatzen da alderantziko aldaketarekin, *a*.L seinaletik abiatuta *a*.H seinalea lortzeko.

Zirkuitu hori islatzeko (zirkuitu bera da bi kasuetan), honako sinbolo grafiko hauek erabiliko ditugu:

2.13. irudia. NOT atearen bi interpretazioak.

Zirkuitu horren izen komertziala NOT da (edo inbertsore fisikoa). NOT atearen funtzionamendua, tentsio-maila aldatzeaz gain, seinale baten ezeztapen gisa ere interpreta daiteke, baldin eta sarrera eta irteera logika berean badaude. Portaera hori islatzen da 2.14. irudian: sarrerako aldagaiaren osagarria lortzen da irteeran, sarreraren logika berean.

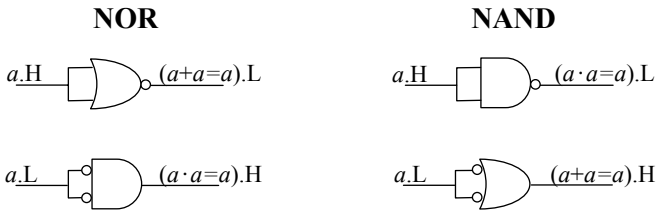


2.14. irudia. Seinale baten ezeztapena, haren logika mantenduz.

2.2.3.1. *not* eragiketaren beste gauzatze batzuk

NOT zirkuitua da ohikoena seinale baten tentsio-aldaketa betetzeko, L-tik H-ra edo alderantziz, baina ez da aukera bakarra. Izan ere, gauza bera egin daiteke lehen azaldu ditugun NAND eta NOR ateekin.

Ate horietan, bi sarrera daude; eramaten bada seinale bera bietara, $a + a = a$ edo $a \cdot a = a$ eragiketa beteko da. Irteera eta sarrerak alderantzizko logikan daudenez gero, ondorio bakarra hau da: seinale logiko bera lortu dugu, alderantzizko logikan.



2.15. irudia. NOT atearen funtzioa NOR eta NAND ateen bidez.

Lehen ikusi dugu NOR eta NAND ateekin batuketa zein biderketa logikoak egin daitezkeela; horrez gain, ikusi berri dugu ezeztapenak ere egin daitezkeela ate horiekin. Beraz, NOR ateekin soilik, edo NAND ateekin soilik, edozein funtzio logiko eraiki daiteke, ate horiekin oinarrizko hiru eragiketak —*or*, *and* eta *not*— egin baitaitezke.

2.2.4. Beste eragiketa batzuen gauzatzea: *xor* eta *equ* eragiketak

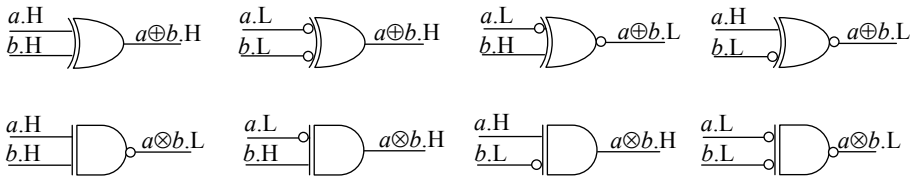
xor funtzioa eta *equ* funtzioa (*xor* funtzioaren alderantzizkoa) maiz ageri dira sistema digitaletan. Funtzio logiko guztiak bezala, oinarrizko ateekin egin daitezke, baina ohikoa da ate bereziak izatea funtzio horiek egiteko.

Horrela definitu ditugu bi funtzio horiek aurreko kapituluan:

a	b	$a \oplus b$	$a \otimes b$
0	0	**0**	**1**
0	1	**1**	**0**
1	0	**1**	**0**
1	1	**0**	**1**

Hau da, *xor* funtzioa aktibatzen da bi eragigaiak desberdinak direnean, eta *equ* funtzioa, biak berdinak direnean. Hauek dira bi ate horien sinbolo grafikoak:

or eta *and* funtzioekin ikusi dugun moduan, hainbat aukera dago *xor* eta *equ* funtzioak egiten dituzten zirkuituak eraikitzeko, sarreren eta irteeraren logikaren arabera. Horietatik guztietatik, 2.16. irudian ageri diren 8 aukerak erabiltzen dira, laster ikusiko dugun moduan, zirkuitu bakar batekin gauzatzen direlako.



2.16. irudia. *xor* eta *equ* funtzio logikoen 8 aukera.

Analiza ditzagun 2.16. irudiko zirkuituak.

1. *xor* funtzioa. Adibide gisa, 2.16. irudiko *xor* funtzioaren lehenengo bi aukeren taula fisikoak lortuko ditugu.

eragiketaren taula logikoa

a	*b*	***a* ⊕ *b***
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

1. zirkuituaren taula fisikoa

a.H	*b*.H	***a* ⊕ *b*.H**
L	L	L
L	H	H
H	L	H
H	H	L

2. zirkuituaren taula fisikoa

a.L	*b*.L	***a* ⊕ *b*.H**
H	H	L
H	L	H
L	H	H
L	L	L

2.17. irudia. *xor* funtzioaren bi gauzatzeren taula fisikoak.

Garbi dago: bi taulak berdinak dira; hots; zirkuitu bera da! Emaitza bera lortzen da *xor* funtzioaren beste bi aukerekin (2.16. irudikoak).

2. *equ* funtzioa. Adibide gisa, 2.16. irudiko *equ* funtzioaren lehenengo bi aukerak aztertuko ditugu.

eragiketaren taula logikoa

a	*b*	***a*** ⊗ ***b***
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**1**

1. zirkuituaren taula fisikoa

a.H	*b*.H	***a*** ⊗ ***b***.L
L	L	L
L	H	H
H	L	H
H	H	L

2. zirkuituaren taula fisikoa

a.L	*b*.H	***a*** ⊗ ***b***.H
H	L	H
H	H	L
L	L	L
L	H	H

2.18. irudia. *equ* funtzioaren bi gauzatzeren taula fisikoak.

Berriro ere, bi taula horiek berdinak dira (eta gauza bera gertatzen da *equ* funtzioaren beste bi aukerekin, 2.16. irudikoak).

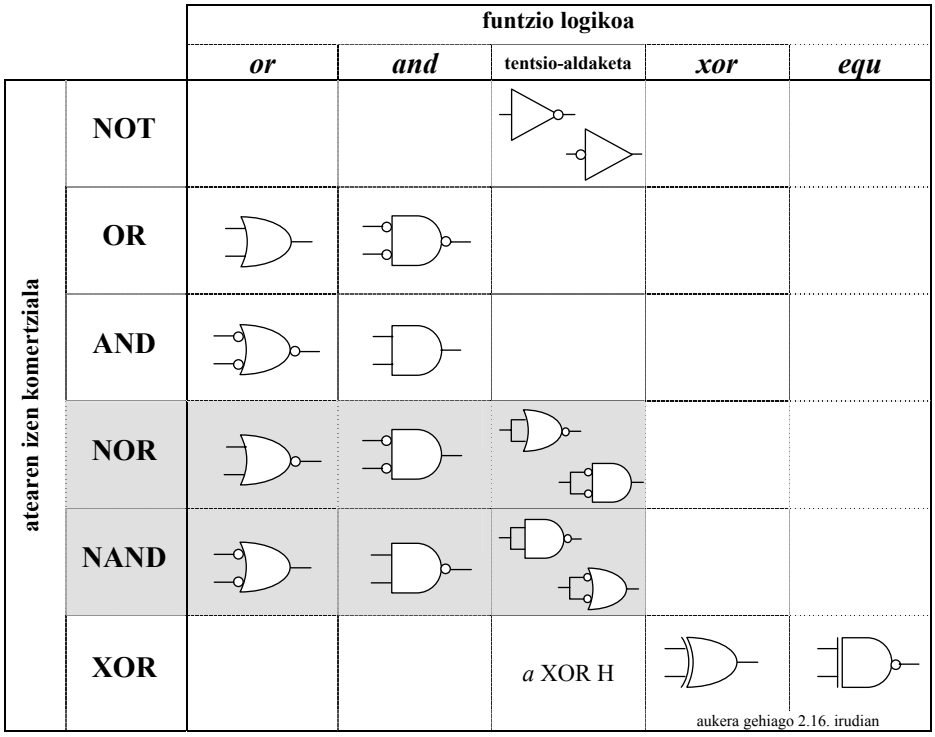
Gainera, *xor* funtzioarenak eta *equ* funtzioarenak berdinak dira (konpara itzazu 2.17. eta 2.18. irudietako taulak). Hau da: zirkuitu bera gai da bi funtzio horiek egiteko, 2.16. irudiko 8 konbinazio desberdinetan. Zirkuitu horren izen komertziala XOR da.

Bidenabar, XOR zirkuituarekin NOT atearen funtzioa (tentsio-aldaketa, alegia) ere egin daiteke. Izan ere, nahikoa da H tentsioa jartzea sarrera batean eta ezeztatu behar den aldagaia bestean (irakurlearentzat uzten da hori egiaztatzea).

2.2.5. Oinarrizko ateen laburpena

Oinarrizko zirkuitu digitalak dira ateak eta aljebra boolearraren oinarrizko funtzioak egiten dituzte. Ate baten funtzionamendua interpretatzeko aukera bat baino gehiago dago, sarreren eta irteeraren logikaren arabera (positiboa edo negatiboa.) Ondorioz, funtzio logiko desberdinak gauza daitezke, eskuarki, zirkuitu fisiko bakar batekin.

Laburpen gisa, hona hemen oinarrizko ateen eta haien interpretazio logikoen laburpena.



2.19. irudia. Oinarrizko ateen interpretazio logikoak.

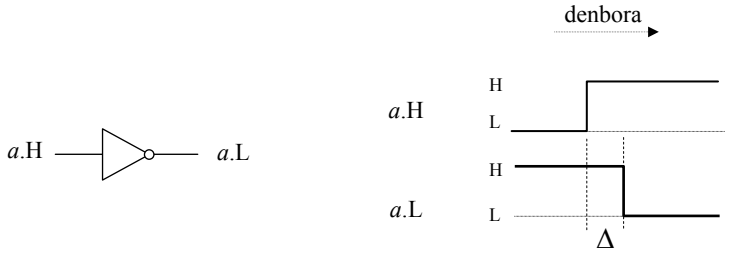
Oinarrizko ateak dira 2.19. irudikoak, baina gehiago daude. Esaterako, ohikoa da 3, 4... sarrerako ateak erabiltzea.

Gogora ezazu: 2.19. irudian ageri denez, NAND ateekin (eta NOR ateekin ere) *or* eta *and* eragiketak egin daitezke, bai eta tentsio-mailak aldatu ere. Beraz, edozein funtzio logiko eraiki daiteke NAND (edo NOR) ateak bakarrik erabiliz.

2.2.6. Ateen erantzun-denbora

Ateak zirkuitu fisikoak dira, transistoreek osatuta. Kapituluaren hasieran esan dugun moduan, transistoreek bi balio fisikoen artean kommutatzen dute: tentsio altua eta baxua. Aldaketa hori egiteko, denbora behar da, txikia bada ere; hau da, sarrerako seinalean tentsioa aldatzen bada, irteerakoa aldatuko da, baina ez bat-batean, geroxeago baizik. Denbora horri zirkuituaren **erantzun-denbora,** atzerapena edo latentzia deritzo.

Adibidez, 2.20. irudian, NOT ate baten erantzun-denbora islatzen da: Δ. Sarrera aldatu egiten da L tentsiotik H tentsiora; ondorioz, irteerak ere aldatu behar du, H-tik L-ra, baina aldaketa hori geroxeago gertatuko da.



2.20. irudia. NOT atearen erantzun-denbora.

Hurrengo kapituluetan ikusiko dugunez, erantzun-denborak garrantzitsuak dira hainbat gailuren funtzionamenduan. Oro har, gailuen erantzun-denborak gailuak egiteko erabili den teknologiaren araberakoak dira. Teknologia batzuk azkarragoak dira eta beste batzuk motelagoak, eta, eskuarki, azkarragoak diren zirkuituak kontsumo altuagokoak dira (ikus E2 eranskina).

2.3. ZIRKUITUEN SINTESIA ETA ANALISIA

Funtzio logiko baten adierazpen aljebraikoa dugularik, gai izan behar dugu funtzio hori zirkuitu digital baten bitartez eraikitzeko. Prozesu horri, zirkuituaren **sintesia** deritzo. Funtzio logiko jakin bat gauzatzen duen zirkuitua ez da bakarra, ate bat baino gehiago erabil daitezkeelako oinarrizko funtzio logikoak eraikitzeko. Adibidez, batuketa logikoak egin daitezke OR, AND, NOR edo NAND ateekin, sarreren eta irteeraren logiken arabera. Ateak aukeratzean, honako irizpide hauek erabil daitezke:

- Edozein ate erabil badaiteke, aukera ezazu ate kopuru minimoa erabiltzen duen zirkuitua; hots, NOT ate (inbertsore fisiko) gutxien duena.
- Askotan, hainbat murriztapen daude ateak erabiltzeko unean (arrazoiak oso desberdinak izan daitezke: eskura daukagun hardwarea, sistema osorako dagoeneko egin ditugun aukerak, erantzun-denborak...). Esaterako, NOR ateak soilik erabili nahi dira, edo NAND ateak soilik. Hori bada kasua, egokitu *or* eta *and* eragiketa logikoak NOR (edo

NAND) ateetara (ikus 2.19. irudia). Gogoratu: NOR edo NAND ateekin edozein funtzio logiko eraiki daiteke.

- Orain arte 2 sarrerako ateak bakarrik aztertu baditugu ere, hainbat sarreratako ateak ere badaude; esaterako, 3, 4... sarrerako OR, NAND... ateak. Beraz, horiek ere erabil daitezke, hardwarea sinplifikatzeko zein erantzun-denbora minimizatzeko.

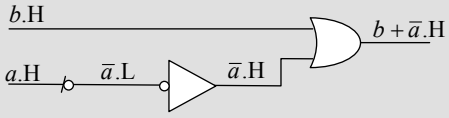
Hala ere, ez kezkatu erabiltzen dituzun zirkuituez. Izan ere, hurrengo kapituluetan ikusiko dugun moduan, funtzio logiko konplexuak ateak baino gailu konplexuagoak erabiliz eraikitzen dira.

Sistema digitalen diseinu-prozesuak konplexuak dira, eta askotan "interpretatu" egin beharko dugu zirkuitu jakin batek gauzatzen duen funtzio logikoa. Prozesu horri zirkuituaren **analisia** deritzo. Sintesia zein analisia beharrezkoak dira sistema digitalen diseinuak aurrera eramateko: f funtzio baten adierazpenetik abiatuta, sintesia eginez, zirkuitu bat lortu behar da; gero, zirkuitu hori analizatu egin behar da gauzatzen duen funtzioa lortzeko, g funtzioa; g eta f funtzioak konparatu behar dira berdinak diren egiaztatzeko. Berdinak ez badira, akatsa non dagoen bilatu beharko da, eta prozesuari ekin berriz.

2.3.1. Zirkuitu baten sintesiaren adibidea

$f = (b + \overline{a})c + \overline{d}$ funtzio logikoa eraiki behar da, non sarrerako lau seinaleak —a, b, c eta d— logika positiboan baitaude, eta irteera logika negatiboan behar den.

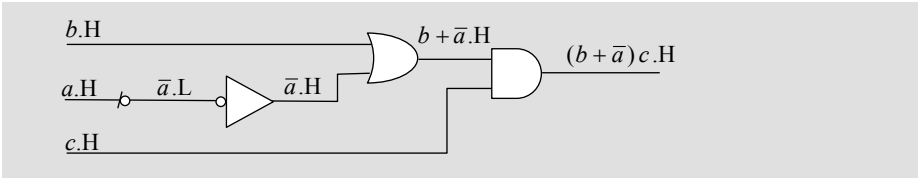
Bi batuketa logiko eta biderketa logiko bat egin behar dira, bai eta bi ezeztatze ere. Has gaitezen lehendabiziko batuketarekin: $b + \overline{a}$. Aldagai bat ezeztatu behar dugu eta bestea ez; bi sarrerak logika berean daudenez, NOT ate bat erabili beharko dugu. Esaterako,



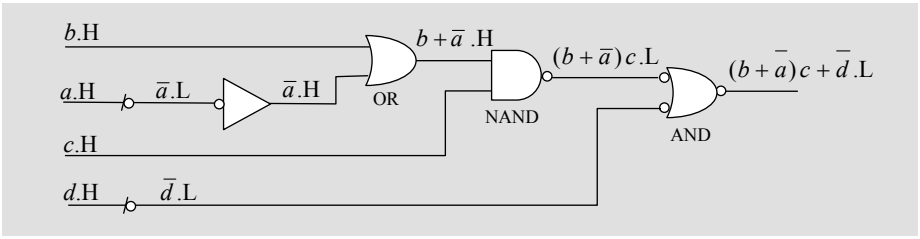
OR atearen bi sarrerak logika positiboan daude, eta, horregatik, NOT atea bigarren sarreran kokatu behar izan dugu. OR ate hori erabili beharrean, bi

sarrerak logika negatiboan dituena erabiliko bagenu, orduan NOT atea *b* aldagaiaren logika aldatzeko erabili beharko genuke.

Jarraitzeko, aurreko emaitza eta *c* aldagaia biderkatu behar ditugu. Nahikoa da AND ate bat erabiltzea:

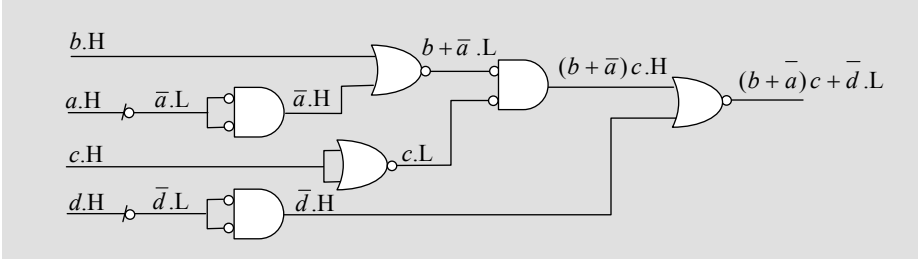


Azkenik, aurreko emaitza eta $\bar{d}$ aldagaia batu behar ditugu. *d* aldagaia ezeztatzean, logikaz ere aldatuko dugu, positibotik negatibora. Beraz, interesatzen zaigu OR atearen sarrerak logika negatiboan izatea. Horrek eskatzen digu aurreko emaitza logika negatiboan sortzea; AND ate bat erabili beharrean, NAND ate bat erabiliko dugu:



Aurreko irudiko zirkuituan, OR, NAND eta AND ate bana aukeratu ditugu biderketa eta batuketa logikoak egiteko, eta, hala, NOT ateen erabilera minimizatu dugu (ate bakar bat). Zenbait kasutan, diseinatzaileak ez du askatasun osoa izango nahi dituen ateak erabiltzeko, eta bakan batzuk erabili ahal izango ditu. Murriztapen horiek direla eta, NOT ate gehiago erabili ohi da.

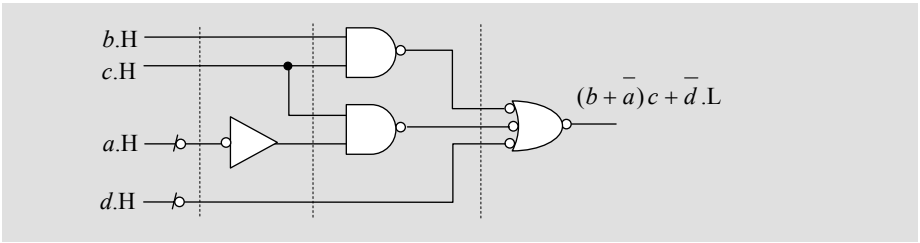
Adibide gisa, hau da funtzio bera sortzen duen beste zirkuitu bat, soilik NOR ateekin egina (gogoratu: batuketa eta biderketa logikoak zein tentsio-aldaketak egin daitezke NOR ateekin; ikus 2.19. irudia):



Lehenengo batuketa, $b + \overline{a}$, logika negatiboan dago NOR ate bat erabiltzen bada, eta hala daude biderketaren bi sarrerak ere. Horregatik, *c* aldagaiaren logika aldatu behar izan dugu, NOR ate batez (NOT atearen funtzioa eginez). Gauza bera gertatu da azken batuketarekin; bi sarrerak logika positiboan daudenez gero, $\overline{d}$ aldagaiaren logika egokitu behar izan da. Ondorioz, 2 ate gehiago ditu zirkuituak.

Ariketa gisa, funtzio bera NAND ateak soilik erabiliz egin dezake irakurleak.

Azkenik, ikus dezagun funtzioak eraikitzeko "metodo estandar bat". Aurreko kapituluan azaldu dugun bezala, edozein funtzio logiko adieraz daiteke era kanonikoan; esaterako, *minterm*-en batuketa gisa. Era kanoniko horretatik abiatuta, agian sinplifikatuta, edozein zirkuitu logiko eraiki daiteke 3 mailatan antolatuta: NOT ateak / *and* eragiketak / *or* eragiketak. Esaterako, $f = (b + \overline{a})\,c + \overline{d} = b\,c + \overline{a}\,c + \overline{d}$ funtzioa honela eraiki daiteke:

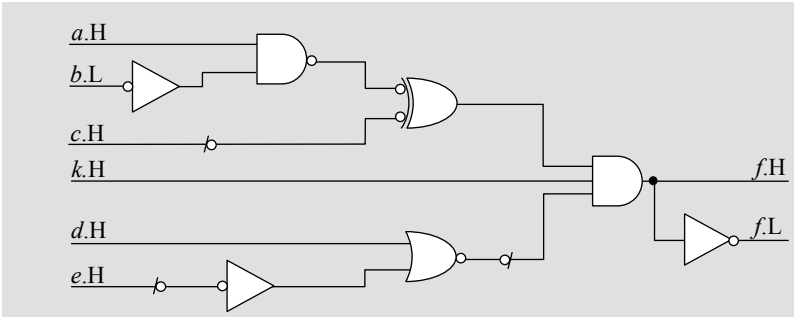


Eskuarki, egitura hori (*not*/*and*/*or*) erabiltzen denean, hardware gehiago behar da funtzioak eraikitzeko, baina, ate guztien erantzun-denbora berdina bada —Δ— (nolabait, sarrera kopuruaren independentea), zirkuituen erantzun-denbora beti bera da: 3Δ (edo 2Δ, ez badira NOT ateak erabili behar). Konparazio baterako, funtzio horren aurreko bi gauzatzeen erantzun-denbora 4Δ da.

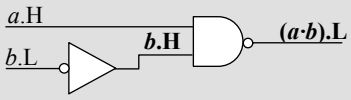
2.3.2. Zirkuitu baten analisiaren adibidea

Zirkuitu baten funtzionamendua analizatzeko, zirkuitu horrek egiten duen funtzio logikoa lortu behar da. Ikus dezagun adibide bat.

Irudiko zirkuituak 6 sarrera-aldagai prozesatzen ditu: *a*.H, *b*.L, *c*.H, *d*.H, *e*.H eta *k*.H. Emaitza bi logiketan ematen da: *f*.H eta *f*.L.

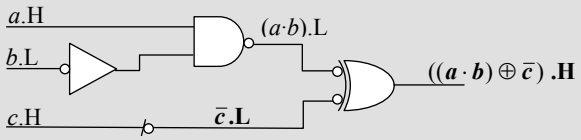


Ate bakoitzaren funtzioa lortu behar dugu, zirkuituarena eskuratzeko. Has gaitezen zati honekin:



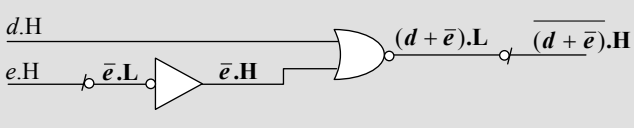
Batetik, *b* aldagaiaren logika egokitzen da (NOT ate batez), negatibotik positibora, *b*.H lortzeko, besterik gabe. Eta bestetik, biderketa logiko bat egiten da (NAND ate baten bidez), irteera logika negatiboan dagoelarik. Beraz, zati horren emaitza (*a*·*b*).L da.

Hurrengo zatia hauxe izan daiteke:



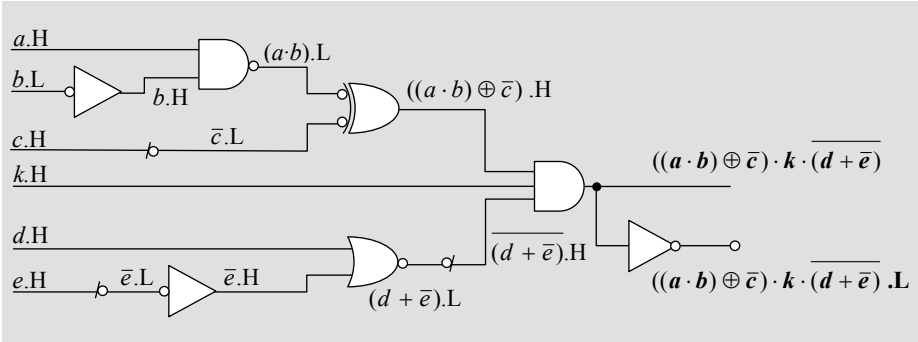
XOR ate bat dago. Lehenengo eragigaia (*a*·*b*).L da, eta bigarrena $\bar{c}$.L. Sarrerako *c* aldagaia logika positiboan dago, eta ezeztatzeko eta logikaz aldatzeko, ez da gailu fisikorik behar; horixe bera da irudiko "inbertsore logikoaren" eginkizuna.

Zirkuituaren hirugarren zati bat analizatuko dugu orain.



Bertan, *or* funtzio logikoa egiten da (kasu honetan, NOR ate baten bidez). Lehenengo eragigaia *d*.H da. Bigarren eragigairako, *e* aldagaia erabiltzen da, baina ezeztatuta. Ezeztatu ondoren, logika egokitu da, logika positiboan behar baita $\overline{e}$ aldagaia NOR atearen sarreran. NOR atearen irteera logika negatiboan dago, eta gero ezeztatzen da, logika aldatuz. Beraz, zati horren emaitza $\overline{(d+\overline{e})}$.H da.

Azkeneko atera heldu gara: 3 sarrerako biderketa.



Aurreko irudietan etiketatu ditugun gaiak biderkatzen dira. Beraz:

$$f = ((a \cdot b) \oplus \overline{c}) \cdot k \cdot (\overline{d+\overline{e}})$$

Emaitza, *f*, bi logiketan ematen da, NOT ate bat erabiliz.

Gaur egun, konputagailuaren bidezko aplikazio asko dago sistema digitalak diseinatzeko, eta halako aplikazioek automatikoki egiten dute, neurri batean, zirkuituen analisia eta sintesia. Hala ere, diseinatzaileak gai izan behar du tresna horien bidez lortutako emaitzak aztertzeko, eta, beraz, ondo kontrolatu behar ditu bi prozesuak.

2.4. ZIRKUITU INTEGRATUAK

Zirkuitu digitalen oinarrizko osagaiak dira ate logikoak, eta, aukera asko badago ere, transistoreen bidez eraikitzen dira. Funtzio logikoak betetzen dituzten transistoreak "txip" deitzen diren zirkuitu integratuetan biltzen dira. Txip batean, erdieroalea den siliziozko zati txiki batean (milimetro karratu batzuk), zirkuitu oso bat integratzen da, bere osagaien arteko konexioak jadanik egin direlarik. Silizio zatitxo hori plastikozko edo zeramikazko

kapsula batean gordetzen da, eta, kanpoalderantz, hankatxo metaliko batzuk (*pin*) jartzen dira kanpo-konexioak egin ahal izateko.

2.21. irudia. Txipen ohiko itxura.

Transistore asko integra daiteke txip batean, eta, hala, konplexutasun-maila oso desberdineko funtzio logikoak eraiki daitezke: ate logiko bat zein prozesadore osoa; erregistro bat zein memoria handi bat.

Ohikoa da zirkuitu integratuak sailkatzea erabiltzen dituzten transistore kopuruaren arabera. Hauek dira halako sailkapenetan gehien erabiltzen diren izenak[3]:

- SSI (*Small Scale Integration*) maila. Txipeko transistore kopurua txikia da, ate gutxi batzuk integratzeko adina. Gutxi gorabehera, 10 ate inguru. Oinarrizko zirkuituak dituzte barnean: ateak eta biegonkorrak.
- MSI (*Medium Scale Integration*) maila. Txip batek transistore gehiago ditu, 10 – 100 ate integratzeko adina. Eraikitzen diren funtzioak konplexuagoak dira: multiplexoreak, deskodegailuak, erregistroak, kontagailuak eta abar.
- LSI (*Large Scale Integration*) maila. Txip batean dagoen transistore kopurua handiagoa da, 100 – 10.000 ate integratzeko adina. Konputagailu baten azpisistema oso bat integra daiteke txip batean: unitate aritmetiko/logiko konplexua, erregistro-multzoa, memoria txiki bat, eta abar.
- VLSI (*Very Large Scale Integration*) maila. Transistore kopuru are handiagoa, 10.000 – 100.000 ate integratzeko behar dena. Denetarik aurki daiteke maila honetan: memoriak, zirkuitu programagarriak, prozesadore txikiak, eta abar.

[3] Sailkapen horren mugak ezin dira finkotzat hartu. Izan ere, zirkuituen integrazio-maila, hau da, transistore kopurua txip batean, gero eta handiagoa da eta oso azkar hazten da. Esaterako, 2-3 urtean behin, bikoiztu egiten da txip batean integratzen den transistore kopurua. Beraz, hartu zenbakiak erreferentzia gisa.

- ULSI (*Ultra Large Scale Integration*) maila. Txip konplexuenak dira hauek, 100.000 ate integratzeko baino transistore gehiago dutenak (esaterako, 100 milioi transistore). Mota horietakoak dira edukiera handiko memoriak edo gaur egungo prozesadoreak.

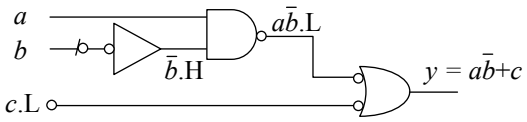
2.5. ARIKETA EBATZIAK

Zirkuitu digitalen oinarrizko hardwarea aztertu dugu kapitulu honetan: ate logikoak. Ariketa batzuk ebatziko ditugu hurrengo orrialdeetan. Helburua bikoitza da: batetik, funtzio logiko baten definiziotik abiatuta, funtzio hori exekutatzen duen zirkuitu digitala nola sortu azaltzea (sintesia); eta, bestetik, zirkuitu bat emanda, exekutatzen duen funtzio logikoa eskuratzea (analisia). Ariketa hauetan landuko ditugun funtzio logiko guztiak sinpleak dira; funtzio logiko konplexuagoak eta haiei dagokien hardwarea hurrengo kapituluetan aztertuko ditugu.

2.1. Ariketa

Irudiko zirkuituak $y = a\,\bar{b} + c$ funtzioa gauzatzen du, bi NAND ate eta NOT ate bat erabiliz. a eta b sarrerak logika positiboan datoz, eta c sarrera logika negatiboan.

Hiru ateen erantzun-denbora berdina da: Δ. Gogoratu: zirkuitu fisiko guztiek bezala, ateek ere denbora (txikia bada ere) behar dute sarrerak prozesatzeko eta emaitza sortzeko.



Une jakin batean, a = b = 1 eta c = 0 sarrera-balioak prozesatzen ari dira, eta, ondorioz, y = 0 da. Egoera egonkorra dela eta, b aldagaiaren balioa aldatzen da: 1etik 0ra. Egin ezazu kronograma bat agerian uzteko noiz aldatuko den y irteeraren balioa 0tik 1era.

■

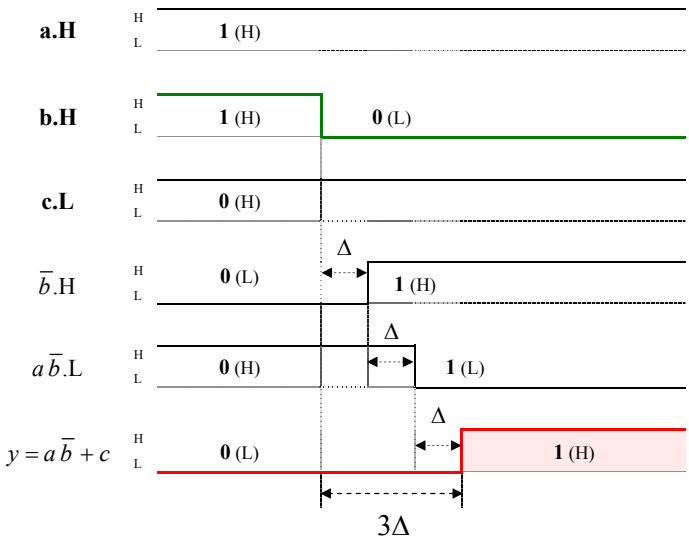
a = 1 eta *c* = 0 direnean, *y* funtzioaren balioa *b* aldagaiaren araberakoa da: *b* = 1 bada, *y* = 0 izango da; eta *b* = 0 bada, *y* = 1 izango da. Beraz, *b* aldagaiaren balioa aldatzen bada 1etik 0ra, *y* funtzioarena ere aldatuko da, 0tik 1era. Baina, horretarako, denbora behar da. Denbora hori oso txikia

bada ere (adibide gisa, Δ = 1 ns), parametro kritikoa izan daiteke hainbat aplikaziotan; esaterako, konputagailuetan, zeinetan kalkulu-abiadura ezaugarri garrantzitsuenetako bat baita.

Ikus dezagun, kronograma batean, *y* funtzioaren eboluzioa denboran zehar. Sistemaren funtzionamenduaren egia-taula islatzen da kronograman, zeinean denbora ere gehitu den: seinaleen balio fisikoak denboran zehar irudikatzen dira.

Bi balio logikoak adierazteko, bi tentsio-maila erabiltzen dira: H eta L; hain zuzen ere, aldagaien tentsio-balioak adierazi ohi dira kronogrametan.

Adibide honetan, *a* eta *c* seinaleak konstanteak dira: balio berari eusten diote denboran. *b* seinalea aldatzen denean, hainbat aldaketa sortzen dira zirkuituan. Batetik, NOT atearen erantzuna aldatuko da, Δ denbora igaro ondoren. Gero, berriro ere Δ denbora igaro ondoren, irudiko goiko NAND atearen irteera-balioa ere aldatuko da, erantzunak orain 1 izan behar baitu (L tentsioa, logika negatiboan dagoelako). Azkenik, beheko NAND atearen erantzuna aldatuko da, *y* = 1 emaitza sortzeko.



Beraz, zirkuituaren erantzun-denbora eragiketa honetarako 3Δ da: sarrerako aldaketa gertatu denetik, dagokion aldaketa irteeran ikusi arteko denbora, hain zuzen ere.

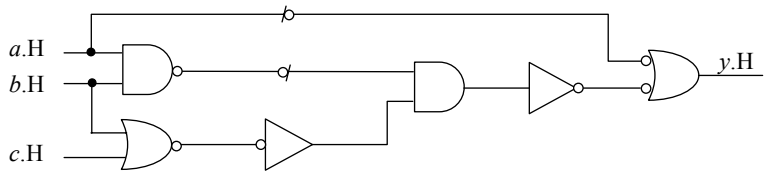
«

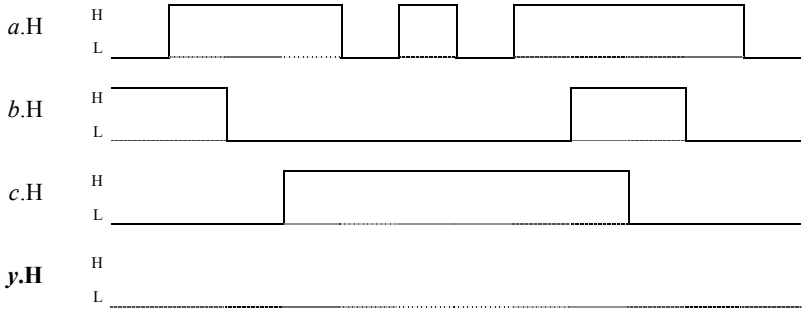
2.2. Ariketa

(a) Analizatu irudiko zirkuitua, eta adierazi gauzatzen duen funtzio logikoa.

(b) Une batean, sarrerako aldagaien balio logikoak a = 1, b = 0 eta c = 1 dira. Adierazi, zirkuituan zehar, ateen irteeretan eta sarreretan sortzen edo prozesatzen diren balio logiko zein fisikoak.

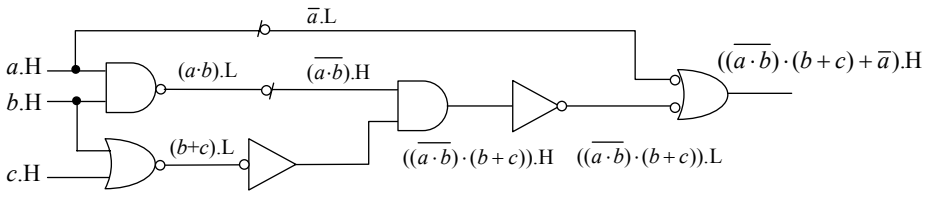
(c) Azkenik, bete ezazu irudiko kronograma.





(a) atala

Zirkuitu baten analisia egiteko, ateetan sortzen diren emaitzak kalkulatu behar ditugu. Hala, zirkuituak egingo duen funtzioa lortuko dugu. Ate bakoitzaren irteera adieraztean, zein logikatan interpretatu behar den (.L edo .H) jarriko dugu. Hona hemen ariketako zirkuituaren analisia:



Beraz, hau da zirkuituak gauzatzen duen funtzioa: $y = (\overline{a \cdot b}) \cdot (b + c) + \overline{a}$

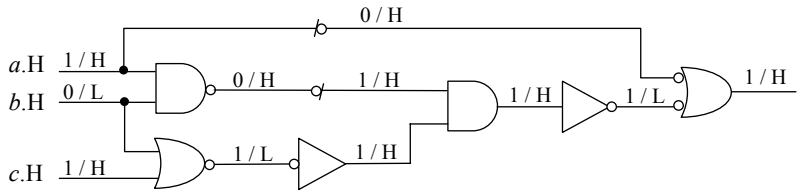
(b) atala

Aurreko irudian, seinaleen adierazpen logikoak idatzi ditugu. Sarrerako aldagaien balioen arabera, tarteko eta bukaerako funtzio horiek balio logiko konkretuak hartuko dituzte: 1 edo 0. Bi balio horiek gauzatzeko, bi tentsio-maila erabiltzen dira zirkuitu elektronikoetan: H tentsioa (altua) eta L tentsioa (baxua).

Balio logikoen eta fisikoen arteko baliokidetza, seinaleen logikaren araberakoa da. Gogoratu:

logika positiboan (.H):	0 → L	1 → H
logika negatiboan (.L):	0 → H	1 → L

Beraz, sarrerako aldagaien balio logikoak $a = 1$, $b = 0$ eta $c = 1$ badira, zirkuituan zehar sortzen diren balio logikoak zein fisikoak honako hauek dira:



(c) atala

Aurreko irudian, zirkuituaren emaitza adierazi dugu, kasu konkretu batean: $a = 1$, $b = 0$ eta $c = 1$ badira, $y = 1$ izango da. Adierazi nahi baditugu zirkuituaren erantzunak denboran zehar, sarrerako aldagaien balioak aldatzen direnean, kronograma bat egin behar dugu. Kronogrametan, seinaleen eboluzioa denboran zehar ageri da, grafikoki; eskuarki, hartzen dituzten tentsio-balioak: H edo L.

Pausoz pauso bete behar da kronograma, seinaleek hartzen dituzten balioak kontuan hartuz, eta, horretarako, erosoa da, lehenengo kronogrametan batik bat, sarrerako balioen aldaketak zein unetan gertatzen diren markatzea. Gero, tarte bakoitzean, y funtzioak hartuko duen balioa kalkulatu behar da. Prozedura hori errepikatu behar da, sarrerako seinaleen balio-konbinazio guztietarako. Lagungarria da, seinaleen balio fisikoez gain (kronogramaren "forma"), haien balio logikoak ere (0 edo 1) adieraztea.

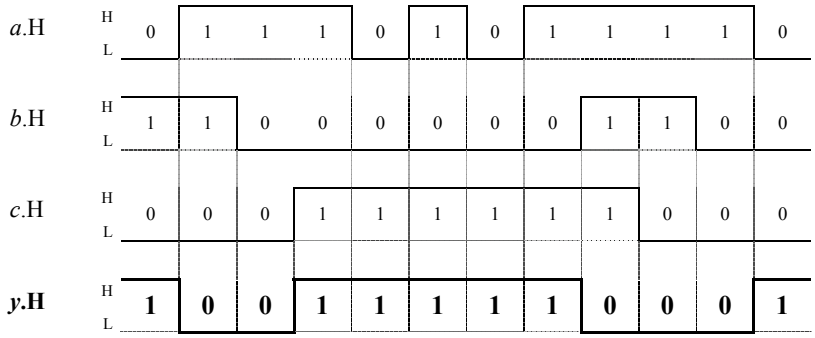
Funtzioaren balioa kalkulatzeko, komeni da funtzioaren adierazpen minimoa lortzea (aljebra boolearraren axiomak eta teoremak edo K-mapak erabiliz). Esaterako, ariketa honetan:

$$\begin{aligned} y &= (\overline{a\,b})\,(b+c)+\overline{a} = \\ &= (\overline{a}+\overline{b})\,(b+c)+\overline{a} = \\ &= \overline{a}\,b+\overline{a}\,c+\overline{b}\,b+\overline{b}\,c+\overline{a} = \\ &= \overline{a}\,(b+c+1)+0+\overline{b}\,c = \\ &= \overline{a}+\overline{b}\,c \end{aligned}$$

Azken ohar bat. Sarrera-seinaleen balioak aldatzen direnean, hala egingo dute zirkuituko ateen irteerek ere. Aurreko ariketan azaldu dugun moduan, denbora beharko da aldaketak gertatzeko. Denbora horiek garrantzitsuak dira askotan, eta merezi du islatzea kronogrametan. Beste batzuetan, aldiz, ez dute esanahi berezirik. Kronograma sinplifikatzeko asmoz, ez ditugu kontuan hartuko ateen erantzun-denborak ariketa honetan; beraz, sarreretan aldaketak gertatu ahala, irteeran islatuko dira aldaketa horiek.

Has gaitezen kronograma betetzen. Lehendabiziko emaitza $a = 0$, $b = 1$ eta $c = 0$ sarrera-balioei dagokie. Beraz, $y = \overline{a} + \overline{b}\,c = \overline{0} + \overline{1}\cdot 0 = 1$. y seinalea logika positiboan dagoenez, zirkuituaren emaitza H tentsioa izango da.

Modu berean bete behar da kronograma osoa. Hona hemen emaitza:



«

2.3. Ariketa

$y = b\bar{a} + \bar{d}(a\bar{b} + c)$ funtzio logikoa egin behar da; lau sarrerak —a, b, c eta d— logika positiboan daude, eta irteera —y— bi logiketan sortu behar da. Eraiki ezazu funtzio hori:

(a) Nahi dituzun ateak erabiliz (NOT ateak eta bi sarrerako AND, NAND, OR, eta NOR), baina ate kopurua minimoa izanik. Adieraz itzazu erabili dituzun ateen izenak.

(b) Bakarrik NAND ateak erabiliz.

(c) Bakarrik NOR ateak erabiliz.

(d) Nahi dituzun ateak erabiliz, baina zirkuituaren erantzun-denbora minimizatuz.

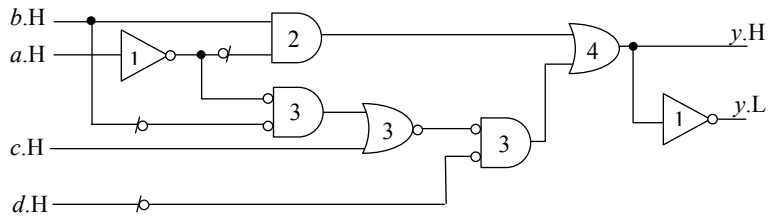
Kalkulatu aurreko zirkuituen erantzun-denbora, ateen atzerapena Δ izanik.

■

(a) atala

Ate kopurua minimizatzeko, NOT ateen kopurua da gutxitu behar dena. Horretarako, ateen sarreren eta irteeren logikaz jokatu beharko dugu, ahal den neurrian, NOT ateak ekiditeko.

Hona hemen funtzioa eraikitzeko aukera bat:

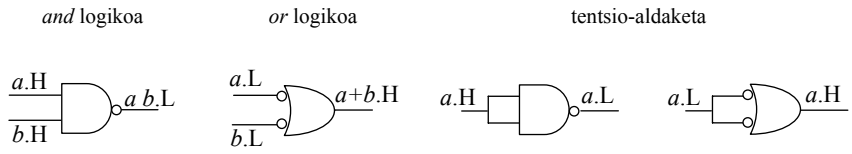


non honako 7 ate hauek erabili baitira: (1) bi NOT ate, (2) AND ate bat, (3) hiru NOR ate, eta (4) OR ate bat.

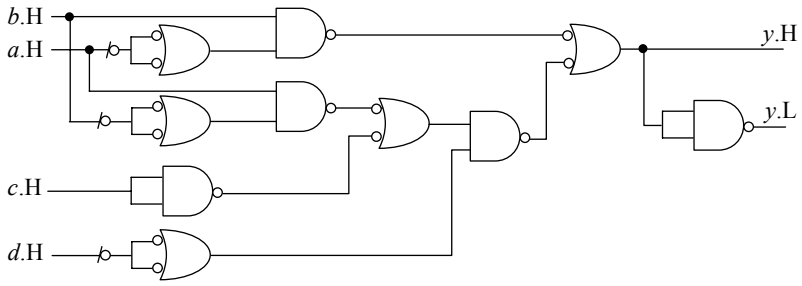
Zirkuitu horren erantzun-denbora kalkulatzeko, ateen maila kopurua kontuan hartu behar dugu. Emaitza lortzeko, gehienez 6 ate igaro behar da. Esaterako, *a* aldagaiak lehenengo NOT atetik igaro behar du; gero, hiru NOR ateetatik; gero, OR atetik; eta, azkenik, emaitza logika negatiboan ateratzeko (*y*.L), NOT atetik. Ate bakoitzaren erantzun-denbora Δ bada, zirkuituaren erantzun-denbora, guztira, 6Δ izango da (kasurik txarrenekoa).

(b) atala

Zirkuitua NAND ateekin egiten hasi aurretik, gogora ditzagun ate horren aukerak oinarrizko funtzio logikoak gauzatzeko:



Ate hori erabiliz, honela egin daiteke *y* funtzioa:

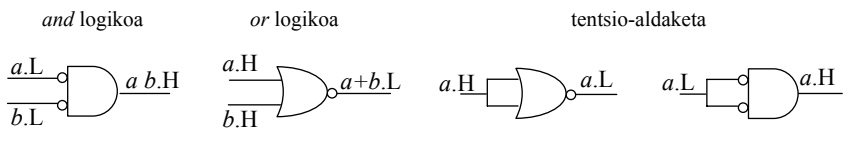


Zirkuituak 10 ate erabiltzen ditu, eta haren erantzun-denbora 6Δ da. Aurrekoarekin konparatuz, ez dugu hobetu erantzun-denbora; orduan, zer abantaila du zirkuitua eraikitzeak ate mota bakarrarekin?

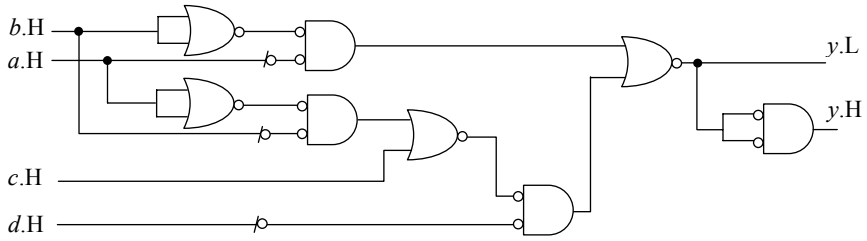
Behe-mailako (SSI) zirkuitu integratuetan, ohikoa da mota bereko hainbat ate izatea txip berean (esaterako, 4 NAND ate). Halako txipekin eraiki nahi badugu zirkuitua, egokia da ate guztiak berdinak izatea, hobeto aprobetxatuko ditugulako txipak. Zenbat eta ate mota gehiago erabili zirkuitua eraikitzeko, hainbat eta handiagoa izan ohi da txip kopurua.

(c) atala

Zirkuitua NOR ateekin egiten hasi aurretik, gogora ditzagun ate horren aukerak oinarrizko funtzio logikoak gauzatzeko:



Honela geratuko da *y* funtzioa NOR ateekin:

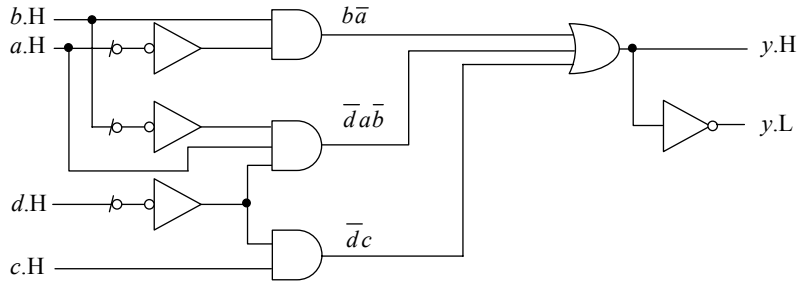


NOR ateekin sortutako zirkuituak 8 ate ditu, eta haren erantzun-denbora, berriro ere, 6Δ da.

(d) atala

Zirkuituaren erantzun-denbora minimizatzeko, ateen maila kopurua txikiagotu behar da. Horretarako, *and* eragiketa guztiak maila batean egiten dira (kasu honetan edozein ate erabiliz), eta, gero, beste maila batean, *or* eragiketak (hainbat sarreratako ateak erabiliz). Behar izanez gero, tentsio-aldaketak egingo ditugu hasierako maila batean.

Hau da, $y = b\overline{a} + \overline{d}(a\overline{b} + c)$ funtziorako, lehendabizi *a, b* eta *d* aldagaiak ezeztatu behar ditugu; gero, $b\overline{a}$, $\overline{d}a\overline{b}$ eta $\overline{d}c$ egingo ditugu; eta, azkenik, hiru gai horien *or* eragiketa.

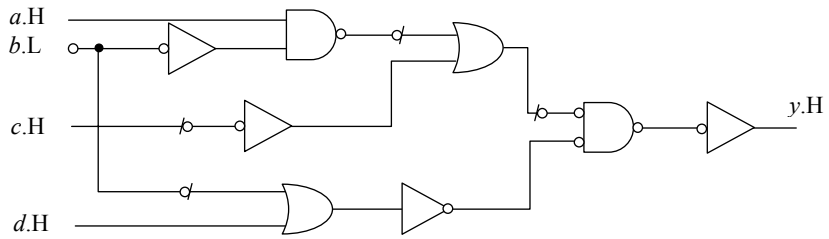


Kasu honetan, zirkuituak 8 ate ditu, 3 (edo 4) mailatan antolatuta; ateen erantzun-denbora sarrera kopuruaren independentea bada —Δ—, zirkuituaren erantzun-denbora 3Δ izango da (4Δ *y*.L sortzeko).

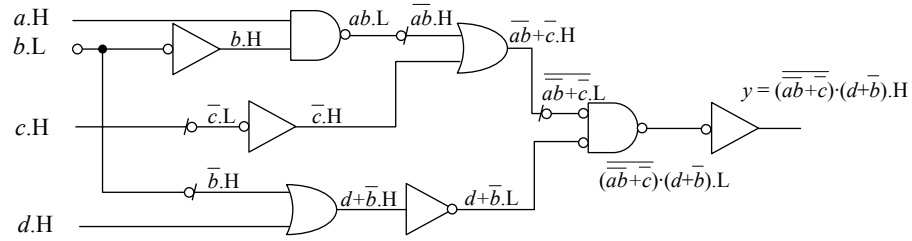
«

2.4. Ariketa

Adieraz ezazu irudiko zirkuituak egiten duen funtzioa, minimizatu lortutako adierazpena, eta egin funtzio minimoari dagokion zirkuitu berria.



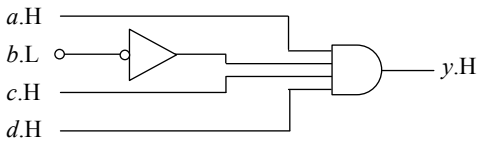
Funtzioaren adierazpena kalkulatzeko, zirkuituaren analisia egingo dugu. Horretarako, atez ate sortzen diren espresioak kalkulatuko ditugu:



Zirkuituaren analisia egin eta gero, aljebra boolearraren teoremak eta axiomak erabil daitezke adierazpen minimoa lortzeko:

$$\begin{aligned} y &= \overline{(\overline{a\,\overline{b}}+\overline{c})}\,(d+\overline{b}) = \overline{\overline{a\,\overline{b}}}\;\overline{c}\,(d+\overline{b}) = \\ &= abc(d+\overline{b}) = abcd + abc\overline{b} = abcd + 0 = \\ &= abcd \end{aligned}$$

Azkenik, hona hemen adierazpen minimoari dagokion zirkuitua, 4 sarrerako AND ate bat (eta NOT ate bat, *b* aldagaiaren logika aldatzeko) erabiliz:

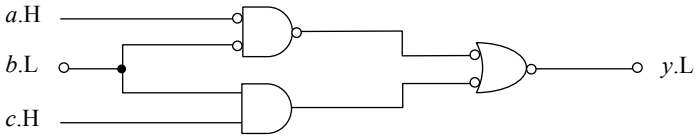


Ariketa honek garbi adierazten du funtzioak minimizatzeak duen garrantzia. Lehenengoan, zortzi ate erabili dira; bigarrenean, aldiz, bi ate bakarrik funtzio bera egiteko.

«

2.5. Ariketa

Zirkuituen logika menderatzen ez duen diseinatzaile batek irudiko zirkuitua proposatu du y = b (a+c) funtzio logikoa gauzatzeko:



Erantzun honako galdera hauei:

(a) Egokia da zirkuitua? Zer funtzio exekutatzen du benetan zirkuituak? Minimiza ezazu funtzio horren adierazpena. Zirkuitua egokia ez bada, zuzendu ezazu, baina haren "egitura" aldatu gabe.

(b) Eraiki y = b(a + c) funtzioa, baina soilik NAND ateak erabiliz.

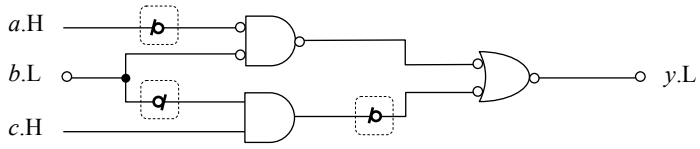
■

(a) atala

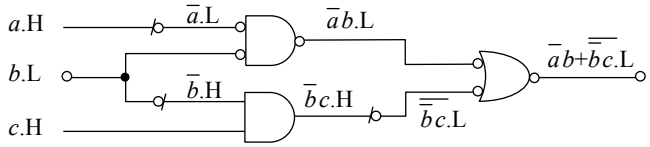
Zirkuituak betetzen duen funtzioa lortzeko, seinale guztien logika hartu behar dugu kontuan. Gogoratu: konektatuta dauden bi seinale ez badaude logika berean, tartean ezeztapen logikoa egin da.

Esaterako, irudiko *a* aldagaia logika positiboan dago, baina OR atearen (biderketa logikoaren) sarrera logika negatiboan dago; beraz, tartean interpretazio-aldaketa (ezeztapena) egon da. Gauza bera gertatzen da *b* aldagaiarekin eta AND atearen sarrerarekin; eta ate horren irteerarekin eta bukaerako beste AND atearekin (batuketa logikoa).

Beraz, argiago izateko, honela irudikatuko dugu zirkuitua:



Hau da, ondorioz, zirkuituaren analisia:



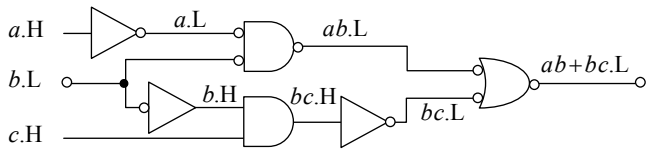
Zirkuituak gauzatzen duen funtzioa erraz sinplifika daiteke; esaterako, aljebra boolearraren axiomak eta teoremak erabiliz:

$$\begin{aligned} y &= \bar{a}\,b + \overline{\bar{b}\,c} = \bar{a}\,b + \bar{\bar{b}} + \bar{c} = \\ &= \bar{a}\,b + b + \bar{c} = b\,(\bar{a} + 1) + \bar{c} = \\ &= b + \bar{c} \end{aligned}$$

Beraz, garbi dago: emandako zirkuituak ez du gauzatzen egin nahi zen funtzio logikoa, eta aldatu egin beharko dugu.

$y = b(a + c)$ funtzioa lortu behar dugu, aurreko zirkuituaren “egitura” —bi biderketa eta batuketa bat— erabiliz. Izan ere, badirudi jatorrizko zirkuituaren helburua $y = ba + bc$ gauzatzea zela (funtzio baliokidea). Ez dago arazorik hori egiteko, baina dagokien logikan tratatu behar dira zirkuituko seinaleak.

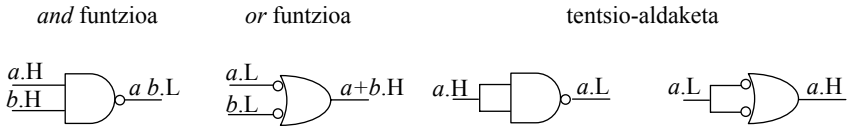
Hau da, egokitu behar ditugu a seinalearen logika (H-tik L-ra), b seinalearena bigarren biderketan (L-tik H-ra), eta biderketa horren emaitzarena (H-tik L-ra). Funtzio horietarako, NOT ateak erabili behar ditugu. Beraz, hauxe litzateke zirkuitua ondo egina:



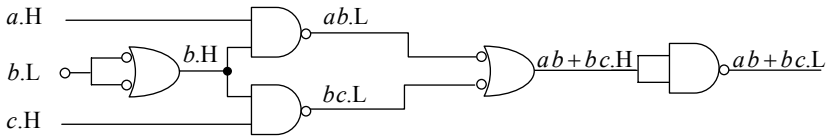
Orain bai, lortutako funtzioaren adierazpena $y = ab + bc = b(a+c)$ da, egin nahi zena, hain zuzen ere.

(b) atala

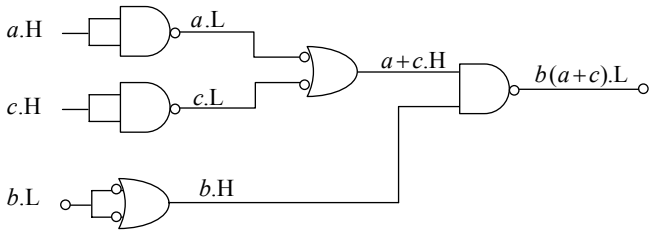
Azkenik, funtzio bera berregin behar da, baina NAND ateekin. 2.19. irudian laburbildu dugun moduan, NAND ateak hiru oinarrizko funtzioak (*or*, *and* eta *not*) egiteko balio digu. Hauek dira funtzio horiek egiteko NAND atearen aukerak:



Ate horiek bakarrik erabiliz, honela egin daiteke zirkuitua:



Era berean, eta kontuan hartuz $y = ab + bc = b(a+c)$, beste modu honetara ere eman daiteke zirkuitua:



«

2.6. Ariketa

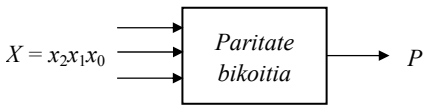
Datuak gorde edo transmititu behar direnean, ohikoa da erroreak izatea, datuen hainbat bit alda edo gal daitezkeelako. Estrategia asko dago arazo horri aurre egiteko; hau da sinpleena: datu bakoitzari bit bat gehitzea, erroreak detektatu ahal izateko.

*Bit hori **paritate-bita** deitzen da, eta kodeen 1ekoen kopurua kontrolatzeko erabiltzen da, bikoitia (paritate bikoitia) edo bakoitia (paritate bakoitia) izan dadin. Hala, detektatzen bada kode batean paritatearekin bat ez datorren 1ekoen kopurua, erroretzat hartuko da.*

*Har dezagun, adibidez, paritate bikoitiaren kasua eta hiru biteko kode hau: 010. Kode horri gehitu behar zaion paritate-bita 1 da, 1ekoen kopurua bikoitia izan dadin (bi). Beraz, paritate-bita gehituta (esaterako, ezkerrean), lau biteko kode hau izango dugu: **1**-010. Bit bat aldatzen bada kode berrian —esaterako: 1-01**0** → 1-01**1**—, erraz detektatuko da errorea, 1ekoen kopurua bakoitia (hiru) izango delako, bikoitia izan beharrean.*

Erroreak detektatzeko metodo hori sinplea da, baina ez da oso eraginkorra; esaterako, ezin dira detektatu 2 biteko erroreak (oro har, errore kopuru bikoitia). Halaber, ezin da jakin zein bitetan gertatu den errorea; hots, ezin da jatorrizko kodea berreskuratu. Hala ere, asko erabiltzen da.

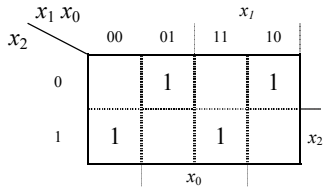
Ariketa honetan, 3 biteko zenbakien paritate-bita (bikoitia) sortzen duen zirkuitua diseinatu behar da. Zirkuituaren sarrera $X = x_2x_1x_0$ zenbakia da, eta irteera P, paritate-bita.



Zirkuitua egiteko, lehen pausoa bere portaera adierazten duen egia-taula sortzea da. Aldagaiak x_2, x_1 eta x_0 dira, eta funtzioa P.

x_2	x_1	x_0	**P**
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Gero, egia-taulatik abiatuta, funtzioaren adierazpen minimoa lortuko dugu; horretarako, K-mapa erabiliko dugu:

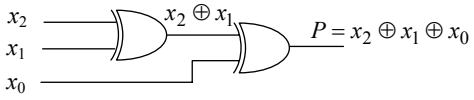


K-mapa horretan, ez dago aukerarik 1ekoak elkartzeko, funtzioa sinplifikatzeko; beraz, funtzioa lau *minterm*-en batura gisa adieraziko dugu:

$$P = x_2\,\overline{x}_1\,\overline{x}_0 + \overline{x}_2\,\overline{x}_1\,x_0 + x_2\,x_1\,x_0 + \overline{x}_2\,x_1\,\overline{x}_0$$

Halaber, funtzio hori *xor* eragiketen bidez ere eman daiteke (ikus 1.5. ariketa): $P = x_2 \oplus x_1 \oplus x_0$

Paritateari dagokion zirkuitua egiteko, bigarren adierazpen hori erabiliko dugu, irudikatzeko sinpleagoa baita. Hau izan daiteke, beraz, 3 biteko kodeetarako paritate-bita sortzen duen zirkuitua:



2.7. Ariketa

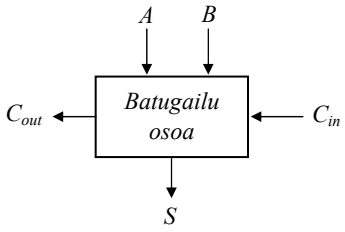
Prozesadore baten osagai nagusietako bat unitate aritmetiko/logikoa da; eta, hor, batugailua dugu elementurik garrantzitsuena: eragiketa aritmetiko gehienak batuketen bidez gauzatzen dira.

*Ariketa honetan, bit bateko bi zenbakien batuketa egiten duen zirkuitua sortu behar da (3. kapituluan ikusiko den legez, erraza da erabiltzea zirkuitu hori n biteko zenbakiak batzeko). Egin behar dugun zirkuituak 3 bit batu behar ditu: bi batugaiak gehi aurreko eragiketa batetik datorren bururakoa (*carry*). Emaitza gisa, 2 bit eman behar ditu: batura zein bururakoa. Zirkuitu horri batugailu osoa (*full adder*) deritzo.*

Ikus dezagun batuketaren adibide bat:

Batuketa hamartarra	*n biteko zenbakien batuketa bitarra*	*Bit bateko zenbakien batuketa bitarra*
1 0 1 0	1 1 1 0	0 (C_{in})
1 3 4 7	**0 1 0 1** **(5)**	**1** **(*A*)**
2 8 2 4	**0 1 1 1** **(7)**	**1** **(*B*)**
4 1 7 1	1 1 0 0 (12)	(C_{out}) 1 0 (*S*)

Beraz, zirkuituaren sarrerak honako hauek izango dira: bit bateko bi zenbaki (A eta B) eta sarrerako bururakoa (C_{in}). Eta irteerak, batura (S) eta irteerako bururakoa (C_{out}). Zirkuituaren egitura honako hau izango da:



Zirkuitua diseinatu ondoren, kalkula ezazu haren erantzun-denbora (bit bateko batuketa bat egiteko denbora), ate guztien atzerapena Δ izanik.

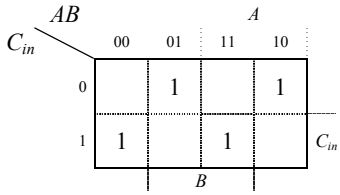
■

Aurreko ariketan bezala, zirkuitua egiten hasi aurretik, haren portaera adieraziko dugu egia-taula baten bidez. Aldagaiak *A, B* eta C_{in} dira, eta irteerako funtzioak *S* eta C_{out}. Adi! Batuketa aritmetikoa egin behar dugu, ez logikoa (*or*); beraz, 1 + 1 = 2 da; bitarrez, 10, hots, batura 0 eta bururakoa 1.

C_{in}	*A*	*B*	C_{out}	***S***
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Gero, egia-taulatik abiatuta, funtzioen adierazpen minimoak lortuko ditugu, esaterako, K-mapak erabiliz.

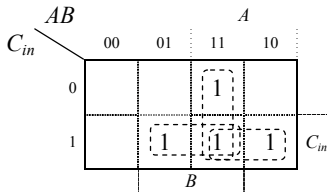
Hau da baturaren (*S*) K-mapa:



Berriz ere, hiru aldagaien *xor* funtzioaren K-mapa ageri zaigu; beraz:

$$S = C_{in} \oplus A \oplus B$$

Bururakoaren (C_{out}) K-mapa, berriz, hau da:



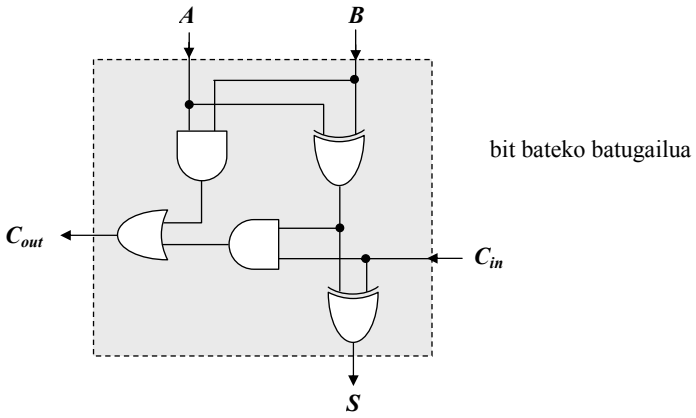
Eta adierazpen minimoa: $C_{out} = AB + C_{in}A + C_{in}B = AB + C_{in}(A + B)$

Minimoa ez bada ere, C_{out} funtzioaren beste adierazpen hau lor daiteke, mapako hirugarren zutabeko elkarketa mantenduz, eta beste bi 1ekoei dagozkien *minterm*-ak elkartu gabe utziz:

$$C_{out} = AB + C_{in}\,\overline{A}\,B + C_{in}\,A\,\overline{B} = AB + C_{in}\,(A \oplus B)$$

Bigarren adierazpen horrek abantaila bat du: gai bat komuna du baturaren adierazpenarekin ($A \oplus B$, alegia). Horrek lagundu dezake zirkuituaren ate kopurua minimizatzen, berrerabil baitaiteke, bigarren funtziorako, lehenengo funtzioaren emaitza partzial bat.

Hau izan daiteke, beraz, batugailu osoaren eskema logikoa:



Zirkuituaren atzerapen-denbora kalkulatzeko, irteera bakoitza lortzeko sarreretatik igaro behar diren ateen kopurua izango dugu kontuan. Hala, S kalkulatzeko, bi XOR baino ez ditugu; beraz, 2Δ-tan izango dugu emaitza. C_{out} lortzeko, ordea, hiru atetatik igaro behar dute seinaleek (XOR → AND → OR); beraz, 3Δ izango da atzerapena (kasurik txarrenean).

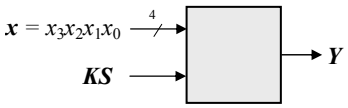
Ondorioz, baliorik handiena hartuz, zirkuituaren erantzun-denbora 3Δ da (batura lehenago lortzen bada ere).

«

» 2.8. Ariketa

Zirkuitu aritmetiko jakin batek bi funtzio egiten ditu, KS izeneko kontrol-seinalearen arabera. Zirkuituak 4 biteko BCD zenbakiak ($x = x_3x_2x_1x_0$, 0tik 9ra) prozesatzen ditu[4]*, eta honako emaitza hau sortzen du:*

```
KS = 0
  baldin (x ≥ 4) orduan Y := 1
  bestela Y := 0
KS = 1
  baldin (x = 3ren multiploa) orduan Y := 1
  bestela Y := 0
```



Zirkuitu hori diseinatu behar da, nahi diren ateak erabiliz.

Gero, kalkulatu egindako zirkuituaren erantzun-denbora, ateen atzerapena Δ izanik.

■

Zirkuituaren funtzionamenduaren logika egia-taula batean bildu behar da. Sarreran bost aldagai ditugunez, 32 konbinazio desberdin izango dira taulan. Hori dela eta, egokia da, ahal den neurrian, emaitza berbera ematen duten konbinazioak biltzea. Esaterako, $KS = 0$ denean, emaitza 0 izango da $x < 4$ diren zenbaki guztietarako: 0000, 0001, 0010 eta 0011; ondorioz, lau aukera horiek "00 – –" bezala adieraz daitezke taulako lerro bakar batean.

---

[4] Banan-banan adierazi beharrean, bitak taldekatu ohi dira sarrera edo irteera bakar batean. Hori adierazteko, marratxo bat eta bit kopurua jartzen dira; esaterako:

Bestalde, x zenbakiak 0tik 9ra doaz; hau da, ez da inoiz gertatuko $x > 9$ izatea.

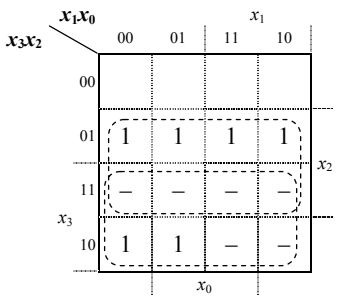
Hona hemen zirkuituaren egia-taula:

KS	x_3	x_2	x_1	x_0	***Y***
0	0	0	–	–	0
0	0	1	–	–	1
0	1	0	0	–	1
0	1	0	1	–	–
0	1	1	–	–	–
1	0	0	0	–	0
1	0	0	1	0	0
1	0	0	1	1	1
1	0	1	0	–	0
1	0	1	1	0	1
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	0	1	1
1	1	0	1	–	–
1	1	1	–	–	–

Bost aldagaiko funtzio baten adierazpen aljebraiko minimoa lortu behar dugu. K-mapen metodoa ez da oso aproposa, ez baita erraza 5 aldagaiko mapak grafikoki adieraztea. Hori egin beharrean, beste hau egingo dugu: funtzioa bi zatitan banatu, *KS* seinalearen arabera. Hala, K-mapa bat (4 aldagaikoa) egin daiteke zati bakoitzeko, eta, gero, lortutako bi azpifuntzioak bildu funtzio bakar batean.

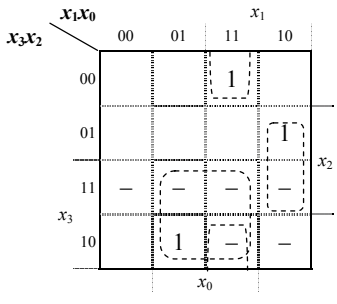
KS seinalearen balioen arabera, honako K-mapa hauek dauzkagu:

***KS* = 0**



$$f_1 = x_2 + x_3$$

***KS* = 1**

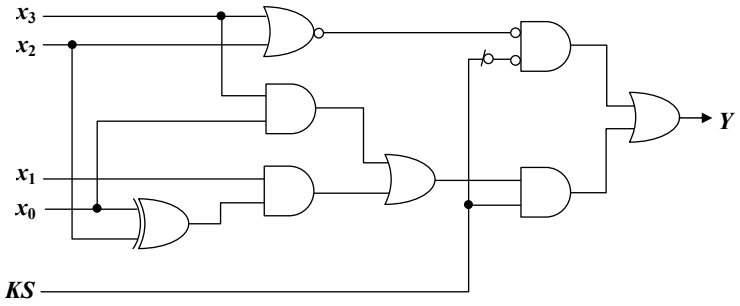


$$f_2 = x_3\, x_0 + \overline{x}_2\, x_1\, x_0 + x_2\, x_1\, \overline{x}_0$$

Lortutako bi funtzio horiek, *KS* aldagaiarekin batera, ariketan eskatzen diguten funtzioa osatuko dute. Horretarako, azpifuntzio bakoitza dagokion *KS* aldagaiaren balioaz biderkatu behar da, eta gero biak batu. Hau da:

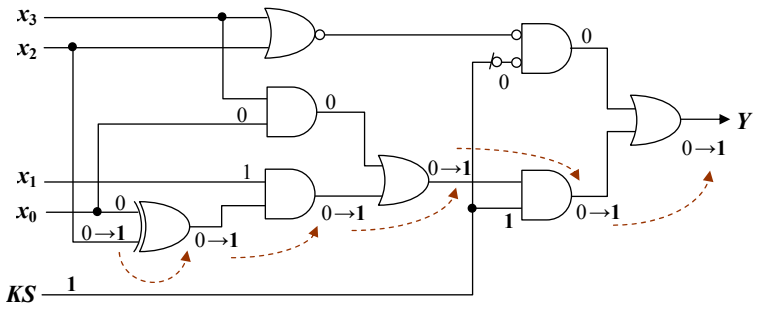
$$Y = \overline{KS}\,(x_2 + x_3) + KS\,(x_3\,x_0 + \bar{x}_2\,x_1\,x_0 + x_2\,x_1\,\bar{x}_0) =$$
$$= \overline{KS}\,(x_2 + x_3) + KS\,(x_3\,x_0 + x_1(x_2 \oplus x_0))$$

Azkenik, funtzioari dagokion zirkuitua egingo dugu:



Y funtzioa lortzeko, sarrerako seinaleek, gehienez, 5 ate-maila (XOR, AND, OR, AND, eta OR) igaro behar dituzte. Beraz, zirkuituaren erantzun-denbora 5Δ izango da (kasurik txarrena).

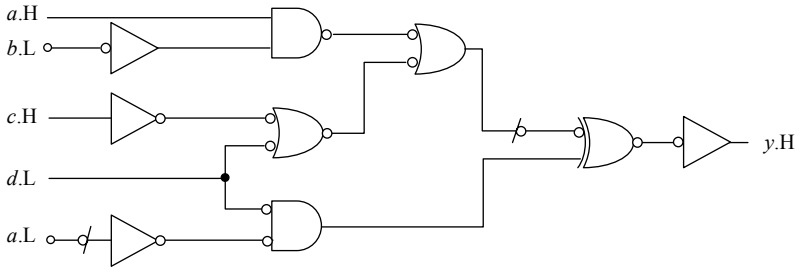
Azter dezagun adibide bat, zirkuituan zehar informazioa nola hedatzen den argi uzteko: $KS = 1$, $x_0 = 0$ eta $x_1 = 1$ dira, eta x_2 0tik 1era aldatzen da (x_3-ren balioak ez du eraginik adibide honetan). Ondorioz, *Y* 0tik 1era aldatuko da.



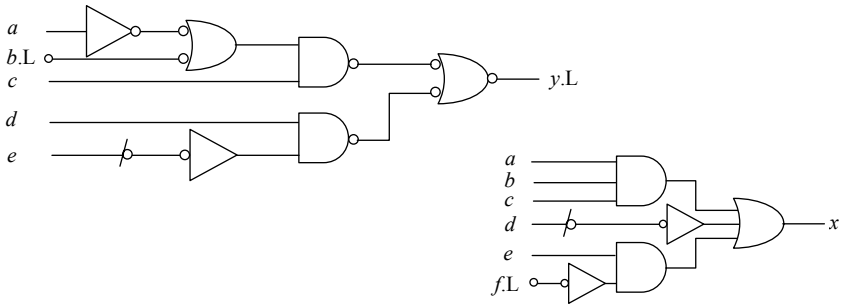
«

2.6. ARIKETAK

2.1. Analiza ezazu honako zirkuitu hau, eta adierazi gauzatzen duen funtzio logikoa. Sarrerako balio logikoak $a = 0$, $b = 1$, $c = 0$ eta $d = 1$ izanik, adierazi, zirkuituan zehar, ateen irteeretan zein sarreretan sortzen diren balio logikoak eta fisikoak.



2.2. Adierazi irudiko zirkuituek gauzatzen dituzten funtzio logikoak. Kasu bakoitzean, eman erabiltzen diren ate logikoen izenak, eta kalkulatu zirkuituaren erantzun-denbora maximoa (ateen erantzun-denbora = 2 ns). Eman erantzun-denbora maximoen adibide bana.

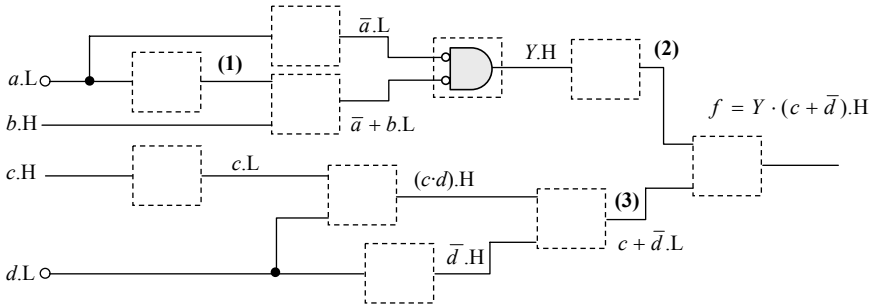


2.3. Eraiki itzazu $f = \overline{a}\,b + a\,\overline{b}\,c + \overline{a}\,c\,\overline{d}$ eta $g = d\,(a + \overline{b}\,c) + b\,\overline{a}$ funtzioak:

(a) nahi dituzun ateak erabiliz (kopuru minimoa)
(b) NAND ateak erabiliz
(c) NOR ateak erabiliz
(d) edozein ate erabiliz, baina zirkuituaren erantzun-denbora minimizatuz

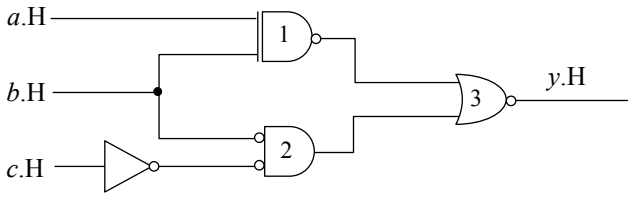
Sarrerako aldagaiak logika negatiboan daude, eta funtzioak bi logiketan eman behar dira.
Kalkulatu zirkuitu bakoitzaren erantzun-denbora, ateen erantzun-denbora 2 ns bada.

2.4. $f = Y(c + \overline{d})$ funtzioa eraiki nahi dugu, soilik NOR ateak erabiliz. Irudian, funtzioa eraikitzeko eskema dugu, tartean datu batzuk ematen direlarik. Bete itzazu falta diren ateen irudiak, adierazten den *f* funtzioa lortzeko.
Zirkuituaren tarteko emaitza gisa, *Y* funtzioa sortzen da; eman funtzio horren adierazpen sinpleena. Era berean, zer funtzio betetzen dira zirkuituko (1) eta (2) puntuetan? zuzena da (3) puntuan emandako adierazpena?



2.5. Eraiki ezazu $f = ((a \oplus b) + (a \otimes c)) \cdot d$ funtzioa XOR eta NAND ateak erabiliz. Aldagai guztiak eta funtzioa logika positiboan daude.

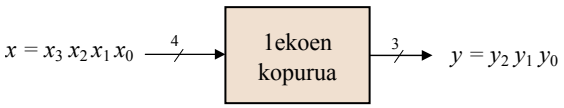
2.6. Zirkuituen logika menderatzen ez duen diseinatzaile batek irudiko zirkuitua proposatu du $y = a \oplus b + b\,c$ funtzio logikoa gauzatzeko:



Erantzun honako galdera hauei:
(a) Egokia da zirkuitua? Zer funtzio exekutatzen du zirkuituak? Minimiza ezazu haren adierazpena.
(b) Zein osagai gehitu edo kendu behar zaio zirkuituari $y = \overline{a \oplus b} + b\,\overline{c}$ funtzioa bete dezan, 1, 2 eta 3 zenbakidun ateak ukitu gabe?

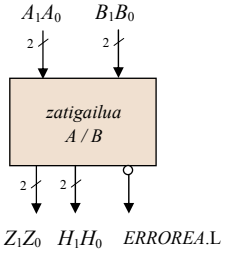
2.7. Egin itzazu lehen kapituluan proposatu diren 1.7., 1.8. eta 1.9. ariketetako funtzioei dagozkien zirkuituak. Erabil ezazu ahalik eta ate kopururik txikiena. Kalkulatu zirkuituen erantzun-denbora, ateen erantzun-denboraren funtzio gisa.

2.8. Zenbakien 1ekoen kopurua kalkulatzen duen zirkuitua diseinatu behar da. 4 biteko zenbakiak ($x = x_3\ x_2\ x_1\ x_0$) prozesatuko ditu sarrera gisa, eta, irteeran, sarrerako zenbakiaren 1ekoen kopurua adieraziko du, 3 bitetan ($y_2\ y_1\ y_0$). Esaterako, sarrerako zenbakia 1011 bada, 1ekoen kopurua 3 da; beraz, irteeran, bitarrez, 3 zenbakia lortu beharko da: $y = y_2\ y_1\ y_0 = 011$.



2.9. 4 biteko zenbakiak ($x = x_3\ x_2\ x_1\ x_0$) prozesatzen ditu zirkuitu batek. Haren irteera aktibatzen da sarrerako zenbakian gutxienez bi 1eko jarraian baldin badaude. Esaterako, $x = 0101$ bada, $y = 0$ izango da; eta $x = 1110$ bada, $y = 1$.
Egin ezazu funtzio horren egia-taula, sortu adierazpen minimoa, eta eraiki zirkuitua (nahi dituzun ateak erabiliz).

2.10. $A(A_1,\ A_0)$ eta $B(B_1,\ B_0)$ bi biteko zenbaki arrunten zatiketa egiten duen zirkuitua diseinatu behar da. Irteera gisa, zatidura eta hondarra eskainiko ditu: $Z(Z_1, Z_0)$ eta $H(H_1, H_0)$.



"Zati 0" eragiketa egiten bada, *ERROREA* seinalea aktibatuko du. Kasu horretan, zatidura eta hondarra ez daude definituta.
Egin itzazu funtzio horien egia-taulak, sortu adierazpen minimoak, eta eraiki zirkuitu osoa (nahi dituzun ateak erabiliz).
Zein da zatigailuaren erantzun-denbora?

3. kapitulua

BLOKE KONBINAZIONALAK

Aurreko kapituluan ikusi dugunez, edozein funtzio logiko eraiki daiteke oinarrizko ateak (NAND, NOR, XOR...) erabiliz. Eraiki nahi dugun sistema digitalaren tamaina edo konplexutasuna handia denean, ordea, diseinu-prozesua asko zailtzen da eta lortzen diren zirkuituak zailak dira interpretatzen. Bestalde, ohikoa da eragiketa batzuk behin eta berriz erabili behar izatea sistema digitaletan; esaterako, datu-mugimenduak, datu-hautaketak, konparaketak, kode-bihurketak, eragiketa aritmetikoak, eta abar. Hori dela eta, badaude funtzio horiek egiten dituzten modulu edo bloke digitalak. Eragiketa horietako bat erabili behar dugunean, ez dugu guk eraikiko; aitzitik, bloke horiek oinarrizko bloke edo "adreilu" gisa erabiliko ditugu gure diseinuetan.

Bloke horiek MSI (*Medium Scale of Integration*, integrazio-maila ertaina) mailakoak dira eta estandarrak diren funtzioak egiteko erabiltzen dira. Bi motatako MSI blokeak bereiz daitezke: bloke konbinazionalak eta bloke sekuentzialak.

Kapitulu honetan, bloke konbinazional erabilienak azalduko ditugu: multiplexoreak, deskodegailuak, kodegailuak, batugailuak, konparagailuak, eta unitate aritmetiko/logikoak. Bloke horien barne-egitura eta funtzionamendua aztertuko ditugu, eta nola erabili sistema konplexuagoak diseinatzeko. Hau da, maila bat gora mugituko gara diseinuan.

3.1. BLOKE KONBINAZIONALEN DEFINIZIOA

Sistema digital bat konbinazionala da baldin eta irteerak sarreretako balioen funtzioak badira soilik; hau da, ez dira kontuan hartzen zirkuituaren aurreko emaitzak. Beste era batera esanda, zirkuitu konbinazionalek ez dute "memoriarik", ez baitute "iragana" gogoratzen: unean uneko sarreren mende ematen dute irteera.

Hala, dagoeneko aztertu ditugun ate logikoak (AND, OR...) konbinazionalak dira, irteera unean uneko sarrera-balioen mendekoa baita, haien egia-tauletan agerikoa den legez.

3.1. irudian ageri da, eskematikoki, nola espezifikatu ohi diren bloke konbinazionalak.

3.1. irudia. Bloke konbinazionalen eskema orokorra (*n* sarrera eta *p* irteera).

Oro har, bloke konbinazionalek hainbat sarrera eta irteera dituzte. Bi sarrera mota bereizten dira: datu-sarrerak eta kontrol-seinaleak. Kontrol-seinaleek blokearen funtzionamendua kontrolatzen dute, eta adierazten dute nola prozesatu datu-sarrerak irteerak sortzeko.

Sarrerak aldatzen direnean bloke konbinazionalen irteerak ere aldatuko dira, dagokien moduan, baina, sistema fisiko guztiak bezala, bloke konbinazionalek ere denbora jakin bat behar dute sarreretako aldaketak irteeretan islatzeko; denbora hori blokearen erantzun-denbora edo atzerapena da. Portaera hori ariketa ebatzietan aztertuko dugu, hainbat bloke konbinazionalen kronogramak eginez.

Azter ditzagun, bada, sistema digitaletan gehien erabiltzen diren bloke konbinazional behinenak[5].

[5] Hurrengo ataletan azalduko ditugun blokeetan, logika jakin bat erabiliko dugu seinaleetarako. Baina edozein logikatan egon daitezke seinale guztiak; garrantzia duena portaera logikoa da, ez erabilitako logika.

3.2. MULTIPLEXOREAK (*multiplexers*)

Multiplexorea zirkuitu konbinazionala da, eta honako funtzio hau betetzen du: hainbat datu-sarreraren artean, bat hautatzen du irteerara pasatzeko. Beraz, datu-hautagailua da multiplexorea; hala ere, multiplexore (edo, laburdura gisa, "mux") izena erabiliko dugu, hori baita ingelesezko testu-liburuetan eta sistema digitalen arloan izen estandarra.

Oso eragiketa arrunta da datu bat hautatu behar izatea sistema digitaletan. Adibidez, demagun $A + B$ edo $A + C$ eragiketak egin behar direla sistema digital jakin batean, hau da, $A +$ (B edo C) egin behar da; beraz, bigarren eragigaia biren artean aukeratu behar da: B edo C. Aukeraketa hori egiteko, multiplexore bat erabili behar da.

Oro har, honako sarrera eta irteera hauek dituzte multiplexoreek:

- Datu-sarrerak

  S_i: Bit bateko n datu-sarrera (n = 2, 4, 8, 16...; 2ren berretura). Sarrerak zenbakituta daude —i = 0, 1, 2... (n–1)—, eta haien zenbakia edo kodea erabili behar da sarrera jakin bat aukeratzeko.

- Datu-irteera

  Y: Bit bateko irteera, non agertuko baita hautatu den sarrera.

- Kontrol-seinaleak

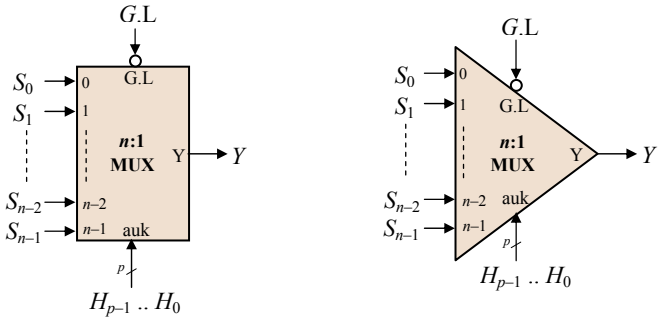
  H_j: p biteko hautatze-seinalea. Kontrol-seinale horren bidez adierazten da aukeratu nahi den sarrera. Beraz, multiplexoreak n datu-sarrera baditu, hautatze-seinalea $p = \log_2 n$ bitekoa izango da.

  Esaterako, 8 sarrerako multiplexoreak 3 bit erabiliko ditu sarreraren bat hautatzeko.

  G: Gaikuntza-seinalea (*enable*). Multiplexorearen funtzionamendu orokorra kontrolatzen du. G = 0 denean, multiplexorea desaktibatuta dago, ez du bere funtzioa betetzen eta irteeran beti 0 ematen du; G = 1 denean, ordea, multiplexorea aktibatuta dago eta, ondorioz, irteeraren balioa aukeratu den sarreraren balioa izango da.

  Eskuarki, G seinalea logika negatiboan egon ohi da, baina positiboan ere egon zitekeen.

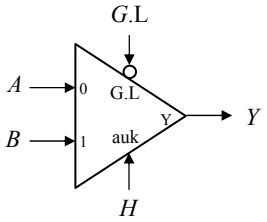
n datu-sarrerako multiplexorea n:1 tamainakoa dela esaten da. 3.2. irudian, n:1 tamainako multiplexorea adierazteko ohiko ikurrak ageri dira.



3.2. irudia. *n*:1 multiplexorea adierazteko ohiko ikurrak.

Azter ditzagun tamaina jakineko multiplexore batzuk, haien funtzionamendua eta barne-egitura argiago uztearren.

Multiplexorerik sinpleena 2:1 multiplexorea da. 2 datu-sarrera dituenez, bit bateko hautatze-seinalea behar du bi sarreren artean bat aukeratzeko. 3.3. irudian ageri dira ikurra eta funtzionamendua:




```
baldin (G = 0) orduan      Y := 0
bestela
    baldin (H = 0) orduan  Y := A
    bestela                Y := B
```

3.3. irudia. 2:1 multiplexorearen ikurra eta definizioa.

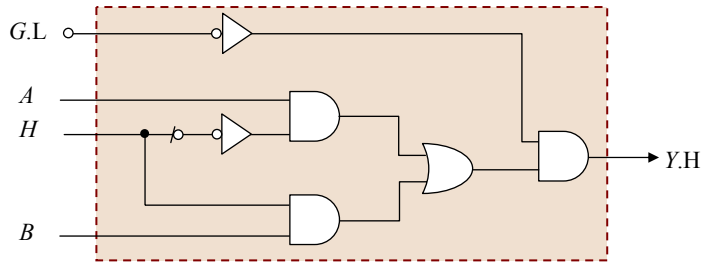
Zirkuitu guztiak bezala, multiplexoreak ere ate logikoen bidez egin daitezke. Horretarako, haien portaera adierazten duen funtzio logikoa lortu behar dugu. Esaterako, hau da 2 sarrerako multiplexorearen egia-taula:

G	*H*	*A*	*B*	*Y*
0	–	–	–	0
1	0	0	–	0
1	0	1	–	1
1	1	–	0	0
1	1	–	1	1

Bestela, honela ere defini daiteke 2 sarrerako multiplexorearen funtzioa: $G = 1$ izanda, $Y = 1$ izango da $H = 0$ eta $A = 1$ direnean, edo $H = 1$ eta $B = 1$ direnean. Egia-taulatik zein definizio horretatik, honako ekuazio logiko hau ateratzen da:

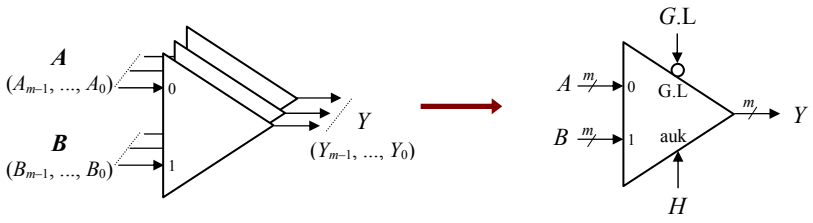
$$Y = G \cdot (\overline{H} \cdot A + H \cdot B)$$

3.4. irudian ageri da funtzio logiko hori ateen bidez egina.



3.4. irudia. 2:1 multiplexorearen barne-egitura.

Multiplexoreek bit bateko seinaleen artean aukeratzen dute. Sarrerak m bitekoak direnean, m multiplexore erabili behar dira, datu-sarreren bit bakoitzeko bat. 3.5. irudian, m biteko 2:1 multiplexorea ageri da.



3.5. irudia. m biteko 2:1 multiplexorea (ezkerreko irudiko multiplexore guztiek gaikuntza- eta hautatze-seinale bera erabiltzen dute).

Lehen aipatu dugun moduan, 2 sarrerako multiplexoreaz gain, 4koak, 8koak eta abar daude. Ateen bidez eraikiko ez dugun arren, analiza dezagun 4:1 multiplexoreari dagokion funtzio logikoa, haren barne-egituraren konplexutasuna tamainarekin batera handitzen dela agerian uzteko (ate gehiago eta sarrera gehiagokoak behar baitira).

4 sarrerako multiplexore batek 2 bit erabili behar ditu sarreraren bat hautatzeko: H (H_1H_0). H = 00 denean, S_0 sarrera aukeratzen da Y irteeran; 01 denean, S_1 sarrera; 10 denean, S_2 sarrera; eta 11 denean, S_3 sarrera. Gainera, G gaikuntza-seinalea 1 denean bakarrik funtzionatzen du multiplexoreak. Beraz,

$$Y = G\,(\overline{H}_1\,\overline{H}_0\,S_0 + \overline{H}_1\,H_0\,S_1 + H_1\,\overline{H}_0\,S_2 + H_1\,H_0\,S_3)$$

Garbi dago: ekuazio logikoa bi sarrerako multiplexorearena baino konplexuagoa da. Tamaina handiagoko multiplexoreen ekuazio logikoak era berean sor daitezke.

Laburbilduz. Datu bat, eragigai bat, seinale bat eta abar aukeratu behar denean askoren artean, multiplexore bat erabili behar da. Horrez gain, erabilera berezia bada ere, funtzio logikoak sortzeko ere erabil daitezke multiplexoreak (ikus 3.2. ariketa). Kapitulu honen (eta hurrengoen) azken ataleko ariketa ebatzietan, hainbat aldiz erabiliko ditugu multiplexoreak sistema digitalak osatzeko.

3.2.1. Hiru egoerako gailuak (*tri-state*)

Aipatu berri dugun moduan, datuak aukeratzeko erabiltzen dira multiplexoreak. Hala ere, hautagai kopurua handia denean, multiplexoreak ez dira oso egokiak, erabili beharko genituzkeen AND eta OR ateak oso handiak izango liratekeelako (aztertu 4 sarrerako multiplexorearen ekuazio logikoa). Esaterako, ez dira erabiltzen, eskuarki, 1024 sarrerako multiplexoreak.

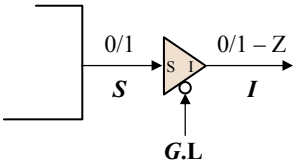
Badago beste modu bat multzo batetik datu bat aukeratzeko, baina, horretarako, ezaugarri berezi bat duten gailuak erabili behar dira: hiru egoerako gailuak.

Orain arte erabili ditugun gailu guztiek bi egoera baino ez dituzte: 1 (egiazkoa) eta 0 (faltsua). Bi egoera horiek bi tentsio desberdinez adierazten dira: H eta L. Horrez gain, lehenengo kasuan korronte elektrikoa sortzen da, eta bigarrenean xurgatzen da. Hala, bi gailuen irteerak ezin dira haien artean konektatu: biek korrontea eman edo xurgatu behar badute, “gatazka” bat gertatuko da eta gailuak matxuratuko dira.

Hiru egoerako gailuek, ordea, hiru egoera dituzte: 1, 0 eta **Z.** Hirugarrenari **inpedantzia altuko egoera** deritzo, eta “**deskonexio birtuala**” adierazten

du: ez da korronterik eroaten. Kontrol-seinale baten bidez aukeratzen da funtzionamendu normalaren (1, 0) eta deskonexio birtualaren artean.

Gailu batek ez badu aukera hori, erraza da hirugarren egoera gehitzea, **hiru egoerako bufferrak** deitzen diren gailuen bidez. Esaterako,

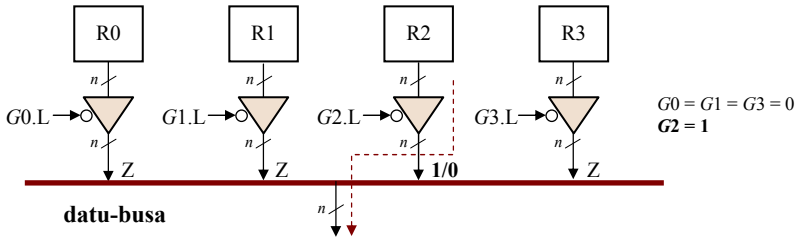


G	*S*	*I*
0	–	**Z**
1	0	**0**
1	1	**1**

3.6. irudia. Hiru egoerako bufferrak.

G seinalea (*OE*, *output enable,* izena ere erabiltzen da) desaktibatuta badago, irteera Z egoeran egongo da; bestela, sarrerako datua ematen du irteeran. Hiru egoerako buffer bat, beraz, "etengailu elektroniko" bat da: itxita dagoenean, sarrerako informazioa (1/0) irteerara igarotzen da; irekita dagoenean, ordea, ez dago konexiorik sarreraren eta irteeraren artean. Horrek bideratzen du gailu askoren irteerak puntu komun batera konektatzea, baldin eta ondo kontrolatzen badira haien egoerak (Z edo 1/0).

Hiru egoerako gailuak asko erabiltzen dira, esaterako, **busak** antolatzeko. Hainbat gailuren arteko konexio mota erabiliena da busa. Gailuak zuzenean konektatzen dira busera, baina, horretarako, ziurtatu behar da gailu bakar bat dagoela, gehienez, aktibatuta, eta gainerakoak Z egoeran daudela. Adibide bat ageri da 3.7. irudian, non *n* biteko 4 gailu konektatuta dauden bus batera datuak emateko.



3.7. irudia. Datu-aukeraketa hiru egoerako bufferren bidez: datu-busa. Nahi izanez gero, egitura bera lor daiteke 4 sarrerako *n* multiplexoreen bidez: sarrera bakoitzean gailu baten irteera konektatu, eta erabili hautatze-kodea horietako bat aukeratzeko irteeran.

3.7. irudiko kasua kontuan hartuz, R2 gailuaren irteera izango dugu datu-busean, bakarrik gailu hori baitago konektatuta datu-busera, besteak Z egoeran —deskonexio-egoeran— daudelako.

Egitura hori oso arrunta da sistema digitaletan, bai gailuak osatzeko (esaterako, memoriak) bai eta azpisistemen arteko konexioak gauzatzeko (konputagailuen osagai nagusiak, adibidez).

3.3. DESKODEGAILUAK (*decoders*)

Sistema digitalek prozesatzen duten informazioa kodetuta dago, eskuarki bitarrez. Hainbat unetan, informazio hori deskodetu behar da, edo beste kode batera pasa behar da. Horregatik, ohikoak dira funtzio horiek betetzen dituzten bloke konbinazionalak: deskodegailuak, hain zuzen ere.

Deskodegailu arruntenek kode bitar bat hartu eta irteera bakar bat aktibatzen dute, sarrerako kodearen balio hamartarrari dagokiona. Eskuarki, honako sarrera eta irteera hauek ditu *n*:*p* deskodegailu batek:

- Datu-sarrerak

  S: Deskodetu beharreko *n* biteko kode bitarra.

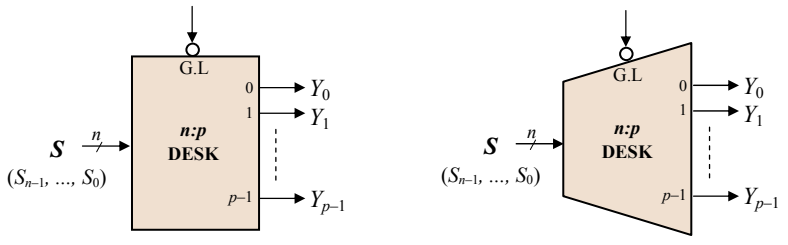
- Datu-irteerak

  Y: Bit bateko *p* irteera (Y_0, ..., Y_{p-1}). Irteera bakoitzak sarrerako kodearen balio hamartar jakin bat adierazten du. Beraz, $p = 2^n$.

- Kontrol-seinaleak

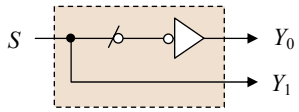
  G: Gaikuntza-seinalea, deskodegailuaren funtzionamendu orokorra kontrolatzeko: bakarrik funtzionatuko du $G = 1$ denean.

3.8. irudian, *n*:*p* biteko deskodegailua adierazteko ohiko ikurrak ageri dira.



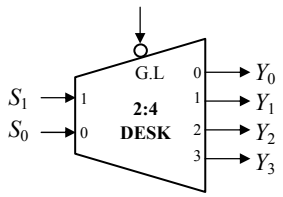
3.8. irudia. *n*:*p* deskodegailua adierazteko ohiko ikurrak.

Deskodegailu sinpleena 1:2 deskodegailua da. S = 0 denean, Y_0 irteera aktibatzen da, eta S = 1 denean, Y_1 irteera. Hau da, $Y_0 = \overline{S}$ eta Y_1 = S. Beraz, ez da gailu berezi bat erabili behar funtzio horiek gauzatzeko, nahikoa baita NOT ate bat:



Hori dela eta, ez dago 1:2 deskodegailurik bloke funtzional gisa.

Ikus dezagun, ondorioz, nola egin deskodegailu sinpleena, 2:4 deskodegailua. 3.9. irudian ageri dira deskodegailu horren ikurra eta dagokion egia-taula.



G	S_1	S_0	Y_0	Y_1	Y_2	Y_3
0	–	–	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

3.9. irudia. 2:4 deskodegailuaren ikurra eta egia-taula.

Egia-taulatik irteera bakoitzari dagokion funtzio logikoa ondorioztatzen da. Hona hemen irteeren adierazpenak:

$$Y_0 = G\,\overline{S_1}\,\overline{S_0} \qquad Y_1 = G\,\overline{S_1}\,S_0 \qquad Y_2 = G\,S_1\,\overline{S_0} \qquad Y_3 = G\,S_1\,S_0$$

Erraza da, beraz, funtzio horiek ate logikoen bidez egitea (irakurlearentzat uzten da).

Egia-taulan agerikoa da deskodegailuaren funtzionamendua: G seinalea 0 denean, ez du axola zein diren beste sarreren balioak (S_1 eta S_0), irteera guztiak 0 baitira. G = 1 denean, ordea, irteera bakarra aktibatuko da, sarrerako kodearen arabera; hots, S_1S_0 = 00 denean, Y_0 irteera aktibatzen da; S_1S_0 = 01 denean, Y_1 irteera; S_1S_0 = 10 denean, Y_2 irteera; eta, azkenik, S_1S_0 = 11 denean, Y_3 irteera.

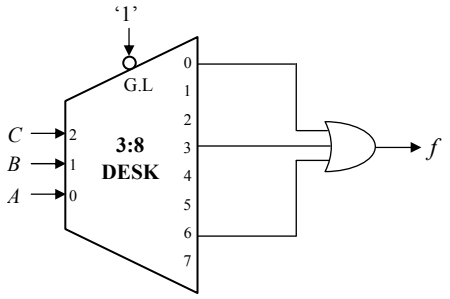
Deskodegailu bitarren artean, asko erabiltzen da 4:16 deskodegailuen bertsio berezi bat: BCD – hamartar deskodegailua. Sarrera-kodea 4 biteko BCD (*Binary Coded Decimal*) kode bat da, hau da, 0tik 9rako digitu bat bitarrez kodetuta (ikus E1 eranskina). Beraz, 10 irteera baino ez ditu

deskodegailu horrek, sarrerako kodea ez baita inoiz izango 9 baino handiagoa.

Ikusi ditugun deskodegailuez gain, badaude antzeko funtzioak egiten dituzten beste bloke konbinazional batzuk: kode-bihurgailuak. Kode-bihurgailuek bi kodeen arteko itzulpena egiten dute; esaterako, asko erabiltzen dira "BCD – bitar", "bitar – BCD" edo "BCD – 7 segmentu" izeneko deskodegailuak. Azkenak, adibidez, sarrera gisa BCD digitu bat hartzen du, eta irteera gisa 7 segmentuzko digitu bat pizteko kodea sortzen du (ikus 1.8. ariketa).

Hainbat aplikaziotan erabiltzen dira deskodegailuak sistema digitaletan. Hona hemen horietako batzuk:

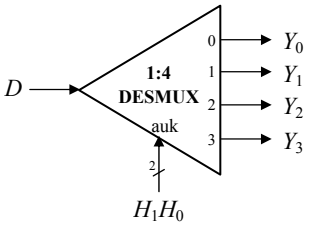
- Memoria bateko posizio bat irakurri edo idatzi ahal izateko, lehendabizi haren helbidea deskodetu behar da (ikus 5. kapitulua).
- Kontrol-automaten egoerak kodetuta izan ohi dira. Hainbat unetan, kode hori deskodetu behar da, esaterako, kontrol-seinaleak sortzeko (ikus 6. kapitulua).
- Prozesadoreetan, eta oro har sistema programagarrietan, aginduak kodetuta daude, eta, exekutatu baino lehen, aginduen eragiketa-kodea deskodetu behar da (ikus 7. kapitulua).
- Azkenik, alboko aplikazio bat bada ere, deskodegailuak funtzio logikoak sortzeko erabil daitezke. Adibidez, demagun *minterm*-en bidez adierazitako 3 aldagaiko honako funtzio logiko hau eraiki behar dugula: $f(C, B, A) = \Sigma(0, 3, 6)$. Berehalakoa da funtzio hori eraikitzea 3:8 deskodegailu bat eta OR ate bat erabiliz, 3.10. irudian ageri den legez.



3.10. irudia. Deskodegailua funtzio logikoak eraikitzeko.

3.3.1. Desmultiplexoreak

Izenak berak agerian uzten duenez, desmultiplexoreak multiplexorearen kontrako funtzioa betetzen du; hots, multiplexoreak hainbat sarreraren artean bat aukeratzen badu irteerara bideratzeko, desmultiplexoreak sarreran dagoen datua irteeretako batera bideratuko du, beti ere hautatze-kode baten arabera. 3.11. irudian ageri dira 1:4 desmultiplexorearen ikurra eta egia-taula.



D	H_1	H_0	Y_0	Y_1	Y_2	Y_3
0	–	–	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

3.11. irudia. 1:4 desmultiplexorearen ohiko ikurra eta egia-taula.

Egia-taula hori 2:4 deskodegailuari dagokionarekin alderatuz gero (3.9. irudia), guztiz berdinak direla ikusiko dugu, desberdintasun bakarra sarreren izenak izanik: deskodegailuan gaikuntza (G) dena, desmultiplexorean datu-sarrera (D) da; eta deskodegailuan kode-sarrera direnak (S_1 eta S_0), desmultiplexorean hautatze-lerroak (H_1 eta H_0) dira. Hortaz, deskodegailuak desmultiplexorearen funtzioa egin dezake, sarrerak modu egokian erabiliz gero.

3.4. KODEGAILUAK (*encoders*)

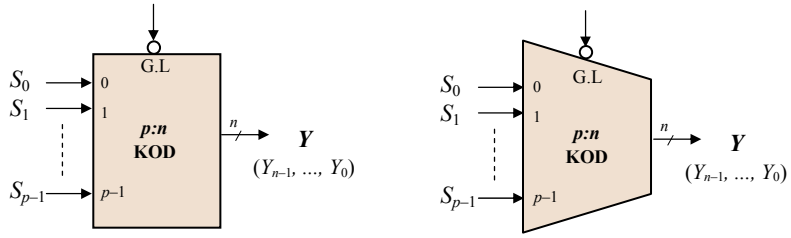
Honetan ere, izenak agerian uzten duen legez, kodegailuek deskodegailuen kontrako funtzioa betetzen dute; hots, sarreretan dagoen informazioa kodetu, eta lortutako kodea irteera gisa ematen dute. Kodegailu erabilienek sarrera hamartarrak bitarrera kodetzen dituzte.

Honako sarrera eta irteera hauek dituzte kodegailuek:

- Datu-sarrerak
  - ***S:*** Bit bateko p sarrera (S_0, ..., S_{p-1}). Sarreraren bat aktibatzen denean, dagokion kode bitarra emango du irteeran kodegailuak.

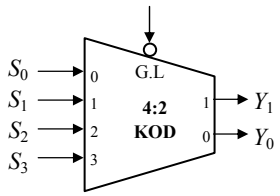
- Datu-irteerak
  - ***Y:*** *n* biteko kodea, aktibatuta dagoen sarrerari dagokion kode bitarra. Honako erlazio hau betetzen da sarrera kopuruaren eta irteerako bit kopuruaren artean: $n = \log_2 p$.
- Kontrol-seinaleak
  - ***G:*** Ohiko gaikuntza-seinalea, zirkuituaren funtzionamendua bideratzeko.

3.12. irudian, *p* sarrera eta *n* biteko irteera dituen kodegailua adierazteko ohiko ikurrak ageri dira.



3.12. irudia. *p:n* kodegailua adierazteko ohiko ikurrak.

Ikus dezagun adibide bat, 4:2 kodegailua. 3.13. irudian ageri dira kodegailu horren ikurra eta egia-taula.



G	S_3	S_2	S_1	S_0	Y_1	Y_0
0	–	–	–	–	0	0
1	0	0	0	0	0	0
1	0	0	0	1	0	0
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	1	0	0	0	1	1

3.13. irudia. 4:2 kodegailuaren ikurra eta egia-taula.

3.13. irudiko egia-taulan ez dira kasu guztiak ageri; izan ere, ez dugu aintzat hartu sarrera-seinale bat baino gehiago batera aktibatuta izatea. Hori kontuan hartu gabe, honela azal daiteke kodegailuaren funtzionamendua. *G* seinalea 0 denean, irteera guztiak 0 dira, zirkuitua ez baitago aktibatuta. Aldiz, $G = 1$ denean, aktibatutako sarrerari dagokion kode bitarra agertzen da irteeran; esate baterako, S_0 sarrera aktibatuta dagoenean, eta besteak

desaktibatuta, irteerako kodea $Y_1Y_0 = 00$ da; S_1 aktibatzen bada, $Y_1Y_0 = 01$ izango da; eta abar.

Kodegailuek bi arazo dituzte. Batetik, 3.13. irudiko egia-taulan ageri den moduan, kodegailua gaituta dagoenean, kode bera, 00 kodea, lortzen da irteeran bi kasutan: S_0 sarrera aktibatuta dagoenean, eta sarrera guztiak desaktibatuta daudenean. Beraz, irteera 00 bada, ez dago jakiterik 0 sarrera aktibatu den ala ez. Arazo hori saihesteko, kodegailu gehienek beste irteera bat izan ohi dute (A —aktibatuta— izena erabiliko dugu). Irteera horrek adierazten du sarreraren bat aktibatu den.

Bestalde, zer gertatzen da une jakin batean sarrera bat baino gehiago aktibatzen bada? Zer kode atera behar da irteeran? Arazo hori saihesteko, kodegailu gehienek "lehentasun-irizpideak" aplikatzen dituzte sarrerak kodetzeko. Oro har, une batean sarrera bat baino gehiago aktibatuta badago, pisu handienekoa kodetzen da. Kodegailu horiei **lehentasunezko kodegailuak** deritze.

Laburbilduz, hau izango da lehentasunezko 4:2 kodegailu bati dagokion egia-taula, bi arazoak konponduta:

G	S_3	S_2	S_1	S_0	Y_1	Y_0	A
0	–	–	–	–	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	–	0	1	1
1	0	1	–	–	1	0	1
1	1	–	–	–	1	1	1

3.1. taula. Lehentasunezko 4:2 kodegailuari dagokion egia-taula.

Hau da, egia-taularen azken lerroan ageri denaren arabera, S_3 sarrera aktibatzen bada, ez du axola nola dauden beste sarrerak, sarrera horrek baitauka lehentasuna: irteeran, 11 kodea sortuko du kodegailuak. S_3 sarrera desaktibatuta baldin badago, ordea, orduan S_2 aktibatzen bada, ez du axola nola dauden S_1 eta S_0 sarrerak, eta kodegailuak 10 kodea sortuko du. Hortaz, lehentasun gutxien duen sarrera S_0 da, berari dagokion irteera-kodea soilik agertuko baita beste sarrera guztiak desaktibatuta daudenean.

Taulan ageri den moduan, kodegailua aktibatuta dagoenean, bi aldiz lortzen da 00 kodea; baina ondo bereizten dira, A irteerari esker: $A = 1$ denean, S_0 sarrera kodetu da; bestela, sarrera guztiak desaktibatuta daude.

3.1. taulako egia-taulatik, honako ekuazio logiko hauek ondorioztatzen dira lehentasunezko 4:2 kodegailu baten irteeretarako:

$$Y_1 = G\,(S_3 + S_2)$$
$$Y_0 = G\,(S_3 + \overline{S_2}\;S_1)$$
$$A = G\,(S_3 + S_2 + S_1 + S_0)$$

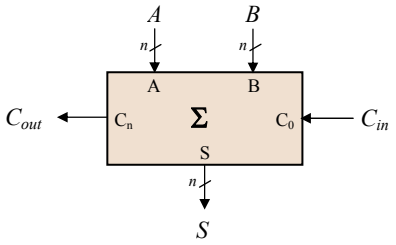
Nahi izanez gero, ate logikoen bidez eraiki daitezke funtzio horiek.

3.5. BATUGAILUAK (*adders*)

Funtsezko eragiketa aritmetikoa da batuketa edozein sistema digitaletan. Horrexegatik, badaude bi zenbaki arrunten batura kalkulatzen duten bloke konbinazionalak. Honako sarrera eta irteera hauek dituzte batugailuek:

- Datu-sarrerak
  - ***A, B:*** Batu nahi diren *n* biteko zenbaki arruntak.
  - $\boldsymbol{C_{in}}$**:** Bit bat, batuketaren sarrerako bururakoa.
- Datu-irteerak
  - ***S:*** Sarrerako datuen batura, *n* bitekoa.
  - $\boldsymbol{C_{out}}$**:** Bit bat, batuketaren irteerako bururakoa.

3.14. irudian, batugailuaren eskema orokorra ageri da.



3.14. irudia. *n* biteko bi zenbakiren batugailuaren ohiko ikurra.

Hala, batugailuak sarreretako bi datuak eta bururakoa batuko ditu, eta emaitza gisa batura ($S = A + B + C_{in}$) eta irteerako bururakoa (C_{out}) emango ditu. Azken horrek eragiketaren gainezkatzea (*overflow*) adierazten du; hau da, sarrerako zenbakiak *n* bitekoak izanik, C_{out} = 1 izango da batura ezin bada adierazi *n* bitetan (ikus E1 eranskina). Izan ere, C_{out} bita baturaren *n*+1 bita izango litzateke.

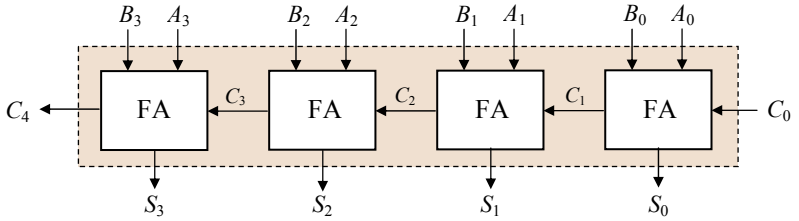
Bit bat baino gehiagoko zenbakien batugailuak egin ahal izateko, bit bateko oinarrizko batugailua diseinatzen da: "batugailu osoa" (*full adder*, FA). Batugailu horrek bit bateko bi zenbaki eta sarrerako bururakoa batzen ditu, eta, emaitza gisa, bit bateko batura eta irteerako bururakoa ematen ditu[6]. Bit bateko *n* batugailu (FA) modu egokian konektatuz, *n* biteko batugailuak lortzen dira. Estrategia hori oso arrunta da zirkuitu integratuen diseinuan: oinarrizko egitura bat behin eta berriz errepikatzea.

2. kapituluan, bit bateko bi zenbakiren batugailu osoaren barne-egitura landu dugu, 2.7. ariketan. Bertan, batugailuaren funtzio logikoei dagokien zirkuitua egina dago. Gainetik bada ere, gogora dezagun 2.7. ariketan ikusitakoa: egia-taula eta irteeren funtzio logikoak.

C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + C_{in}(A \oplus B)$$

Orain, oinarrizko bloke hori erabiliz, bit gehiagoko batugailua lortu nahi badugu, 3.15. irudian ageri den bezala lotuko ditugu bit bateko blokeak. Ageri denez, bloke bakoitzaren irteerako bururakoa sarrerako bururako gisa erabiltzen da hurrengo blokean (horrelaxe egiten ditugu batuketak hamartarrez ere). Hau da, bururakoa blokez bloke hedatzen da, eskuinetik ezkerrera. Horregatik, horrelako batugailuari CPA esaten zaio ingelesez, *Carry Propagation Adder,* bururakoa hedatzen duen batugailua, alegia. Beste izen bat ere erabili ohi da: RCA, *Ripple Carry Adder* (nolabait bururakoa blokez bloke mugitzen delako olatuen antzera, aurrera joateko).



3.15. irudia. 4 biteko bi zenbakiren batugailua.

[6] Sarrerako bururakoa kontuan hartzen ez duen bit bateko batugailuari "batugailu erdia" (*half adder*, HA) esaten zaio.

Beste horrenbeste egin daiteke *n* biteko blokeak elkarri lotuz, *n* baino bit kopuru handiagoko zenbakien batura lortzeko.

Diseinatu dugun batugailuak zenbaki arruntak batzen ditu. Ikus ditzagun bi adibide. Demagun 4 biteko batugailua daukagula eta sarrerako bururakoa 0 dela.

a. $A = 4$ (0100) eta $B = 9$ (1001) direnean, $S = 13$ (1101) da eta irteerako bururakoa $C_{out} = 0$, 13 zenbakia 4 bitetan adierazgarria baita.

b. $A = 7$ (0111) eta $B = 12$ (1100) direnean, ordea, $S = 19$ izan beharko litzateke. Zenbaki hori adierazteko, 5 bit beharko genituzke, baina 4 bit baino ez dago baturarako. Hori dela eta, irteerako bururakoa $C_{out} = 1$ izango da (gainezkatzea —*overflow*— gertatu da) eta $S = 3$ da.

bururako-bitak	0000		bururako-bitak	1000	
A	0100	(4)	A	0111	(7)
B	1001	(9)	B	1100	(12)
C_{out}/S	0 / 1101	($S = 13$)	C_{out}/S	1 / 0011	($S = 3$!)
	(a)			(b)	

3.5.1. Birako osagarrian adierazitako zenbaki osoen batuketa

Oro har, zenbaki osoak erabiltzen dira sistema digitaletan, hau da, zeinudunak, positiboak zein negatiboak. Zenbaki osoak adierazteko adierazpide-sistema erabiliena **birako osagarria** da (*two's complement*)[7]. Erabil al dezakegu diseinatu dugun batugailua 2rako osagarrian adierazitako zenbaki osoak batzeko? Ikus ditzagun adibide batzuk.

Demagun 4 biteko zenbaki osoak batu nahi ditugula. 4 bit erabiliz, [–8, +7] tarteko zenbakiak adieraz daitezke 2rako osagarrian. 3.16. irudian, bi adibide ageri dira. (a) kasuan, $A = 0010$ (+2) eta $B = 1001$ (–7) batzen dira, eta batugailuak honako emaitza hau ematen digu: $S = 1011 = -5$. Emaitza zuzena da. (b) kasuan, $A = 0111 = +7$ eta $B = 1010 = -6$ batzen dira, eta emaitza $S = 0001 = +1$ da. Kasu horretan ere, emaitza zuzena da.

Izan ere, froga daiteke diseinatu dugun batugailuak, zenbaki arruntak ez ezik, birako osagarrian dauden zenbakiak batzeko ere balio duela. Hala ere, berezitasun bat dago: C_{out} irteerak ez du adierazten eragiketaren

[7] E1 eranskinean azaltzen da birako osagarria; irakurleak ez badu ezagutzen nola kodetzen diren zenbaki osoak, eranskina irakurri beharko du atal hau ulertzeko.

gainezkatzea. Esaterako, 3.16. irudiko (b) adibidean, emaitza zuzena izan arren, C_{out} aktibatu egin da. Beraz, ezin dugu hartu C_{out} *overflow* seinale gisa.

bururako-bitak	0 0 0 0		bururako-bitak	1 1 0 0	
A	0010	(+2)	A	0111	(+7)
B	1001	(–7)	B	1010	(–6)
C_{out} / S	0 / 1011	(–5)	C_{out} / S	1 / 0001	(+1)
	(a)			(b)	

3.16. irudia. Gainezkatzea sortzen ez duten bi batuketa 2rako osagarrian.

Izan ere, alderantzizkoa ere gertatzen da: batura ezin da adierazi eta, kasu batzuetan, C_{out} = 1 da, eta, beste batzuetan, 0. 3.17. irudian, gainezkatzea sortzen duten bi batuketa ageri dira. (a) kasuan, A = 0110 (+6) eta B = 0011 (+3) batzen dira; batura, aldiz, S = 1001 = –7 da. (b) kasuan, A = 1011 = –5 eta B = 1010 = –6 batzen dira, eta emaitza S = 0101 = +5 da. Bi kasuetan, argi eta garbi, emaitza okerra da. Izan ere, bietan gainezkatzea gertatu da: baturak (+9 kasu batean, eta –11 bestean) ezin dira adierazi 4 bitetan. Hala ere, lehenengo kasuan C_{out} = 0 da, eta bigarrenean 1.

bururako-bitak	1 1 0 0		bururako-bitak	0 1 0 0	
A	0110	(+6)	A	1011	(–5)
B	0011	(+3)	B	1010	(–6)
C_{out} / S	0 / 1001	(–7 !)	C_{out} / S	1 / 0101	(+5 !)
	(a)			(b)	

3.17. irudia. Gainezkatzea sortzen duten bi batuketa 2rako osagarrian.

Beraz, laburrean, ohiko batugailuak zenbaki arruntak zein osoak (2rako osagarrian) batzeko erabil daitezke. Hala ere, irteerako bururakoak gainezkatzea adierazten du bakarrik zenbakiak arruntak direnean. Zenbaki osoak batzen direnean, beste modu batean detektatu behar da gainezkatzea: bi batugaiak positiboak izanik, batura negatiboa denean, edo alderantziz, bi batugaiak negatiboak izanik, batura positiboa denean. Gogoratu: 2rako osagarrian, pisu handieneko bitak adierazten du zenbakiaren zeinua (0, positiboa; 1, negatiboa). Beraz, honela adierazten da n biteko bi zenbaki osoen batuketaren gainezkatzea:

$$gainezkatzea = \overline{A_{n-1}} \cdot \overline{B_{n-1}} \cdot S_{n-1} + A_{n-1} \cdot B_{n-1} \cdot \overline{S_{n-1}}$$

3.5.2. Kengailuak

Batugailuak 2rako osagarrian dauden zenbaki osoak batzeko balio duen heinean, kenketak egiteko ere baliagarria izan daiteke, hau da, $S = A - B$ egiteko. Horretarako, kontuan hartu behar dugu kenketa batuketa baten bidez egin daitekeela, baldin eta kentzailearen zeinua aldatzen badugu, honelaxe: $S = A - B = A + (-B)$. Beraz, kenketa bat egiteko, kentzailearen zeinua aldatu behar dugu lehenago.

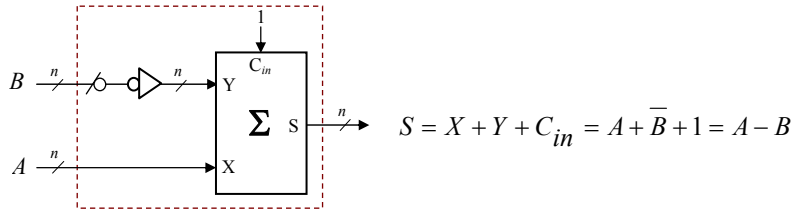
Zeinu-aldaketa honelaxe egiten da birako osagarrian (ikus E1 eranskina):

$$(-B) = \overline{B} + 1$$

Hortaz, honelaxe egin daiteke kenketa:

$$A - B = A + \overline{B} + 1$$

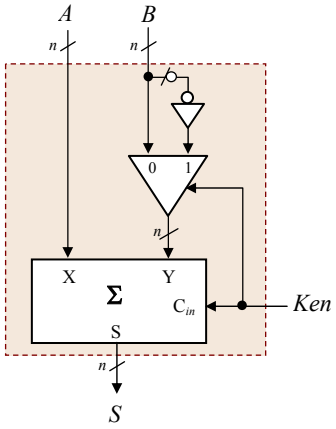
Hau da, batugailuaren sarrera batean A kenkizuna sartu behar da, beste sarreran B kentzailea ezeztatuta ($\overline{B}$), eta sarrerako bururakoan 1 balioa. Horrela, irteeran $S = A + \overline{B} + 1$ izango dugu, hau da, $S = A - B$.



3.18. irudia. $A - B$ kenketa batugailu bat erabiliz.

3.5.3. Batugailua/kengailua

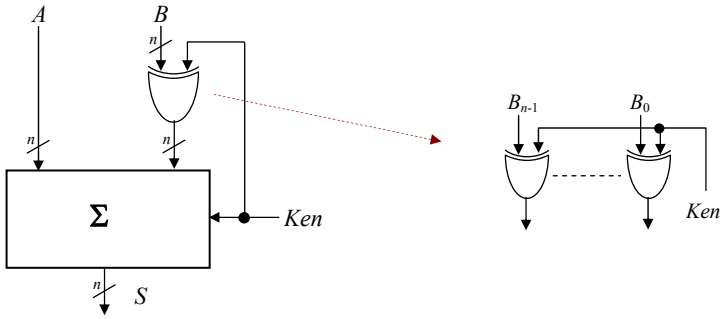
Azkenik, aurreko guztia kontuan hartuz, batugailua batuketak zein kenketak egiteko erabil dezakegu, sarrerako zenbakiak arruntak zein osoak izanik, baldin eta bigarren eragigaia (B) eta C_{in} modu egokian kontrolatzen baditugu: batuketak egiteko, $C_{in} = 0$ izan behar du; kenketak egiteko, berriz, B ezeztatu behar da eta $C_{in} = 1$ izan behar du. Hori guztia kontrol-seinale baten arabera egin daiteke, *Ken*, 3.19. irudian ageri den legez.



Ken	*S*	
0	$A + B$	$(A + B + 0)$
1	$A - B$	$(A + \text{not\_}B + 1)$

3.19. irudia. Batugailua/kengailuaren oinarrizko eskema.

Multiplexore bat eta n NOT ate erabili beharrean, badago beste modu bat B eta $\overline{B}$-ren artean bat aukeratzeko. Nahikoa litzateke n XOR ate erabiltzea "inbertsore kontrolatu" moduan, 3.20. irudian ageri den legez.



3.20. irudia. Batugailua/kengailua XOR ateak erabiliz.

Gogoratu: $B \oplus 0 = B$, eta $B \oplus 1 = \text{not\_}B$. Beraz, $Ken = 0$ denean, batuketa egingo da, eta $Ken = 1$ denean, ordea, kenketa:

$$Ken = 0 \quad \rightarrow \quad A + (B \oplus 0) + 0 = A + B$$
$$Ken = 1 \quad \rightarrow \quad A + (B \oplus 1) + 1 = A - B$$

3.6. KONPARAGAILUAK (*comparators*)

Zenbaki arrunten arteko konparaketa oso eragiketa arrunta denez gero, hori egiten duen blokea ere eskuragarri dago.

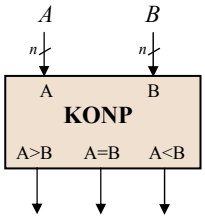
Oro har, honako sarrera eta irteera hauek dituzte konparagailuek:

- Datu-sarrerak

  A, B: Konparatu nahi diren *n* biteko zenbaki arruntak.

- Datu-irteerak

  A>B, A=B, A<B: Konparazioaren emaitzak, bit bateko hiru seinale.



3.21. irudia. *n* biteko bi zenbaki arrunten konparagailuaren ikurra.

Une jakin batean, irteera horietako bat bakarrik egongo da aktibatuta (hots, 1 balioa izango du), eta beste biak desaktibatuta. Hau da, honako konbinazio hauek bakarrik ager daitezke irteeretan:

A>B	*A=B*	*A<B*
1	0	0
0	1	0
0	0	1

n biteko konparagailuak bit bateko konparagailuen bidez eraikitzen dira. Beraz, ikus dezagun zein den bit bateko bi zenbakiren konparagailuaren barne-egitura. Horretarako, irteeren hiru funtzio logikoak lortu behar ditugu. Bit bateko zenbakien —A_0 eta B_0— konparaketa honako egia-taula honetan ageri da:

A_0	B_0	$(A>B)_{1bit}$	$(A=B)_{1bit}$	$(A<B)_{1bit}$
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

Garbi dago: A zenbakia B zenbakia baino handiagoa da $A = 1$ eta $B = 0$ direnean (eta txikiagoa kontrako kasuan). Bestela, biak berdinak dira.

Hiru irteerei dagozkien funtzio logikoak erraz ondorioztatzen dira egia-taulatik:

$$(A > B)_{1bit} = A_0 \overline{B_0}$$
$$(A = B)_{1bit} = A_0 \otimes B_0$$
$$(A < B)_{1bit} = \overline{A_0} B_0$$

Ikus dezagun orain nola egin bi biteko bi zenbaki arrunten konparaketa. Oraingo honetan, egia-taula idatzi beharrean, zuzenean aterako ditugu funtzioak. Kontuan izan $A = A_1A_0$ eta $B = B_1B_0$ direla. Hortaz, noiz izango da bi biteko A zenbakia B baino handiagoa? $A_1 = 1$ eta $B_1 = 0$ direnean, ziur gaude $A > B$ dela (berdin da nolakoak diren pisu txikieneko bi bitak). Baina, bi bit horiek (A_1 eta B_1) berdinak direnean, pisu txikieneko bitak aztertu beharko ditugu. Hortaz, $A_0 = 1$ eta $B_0 = 0$ baldin badira, orduan A zenbakia B baino handiagoa izango da. Ondorioz, honako hau da bilatzen ari garen irteera-funtzioa:

$$(A > B)_{2bit} = A_1 \overline{B_1} + (A_1 \otimes B_1) A_0 \overline{B_0}$$

Beste horrenbeste egin dezakegu gainerako bi funtzioak ondorioztatzeko. Alegia, noiz izango dira A eta B berdinak? Bada, pisu handieneko bi bitak berdinak direnean eta pisu txikienekoak ere:

$$(A = B)_{2bit} = (A_1 \otimes B_1)(A_0 \otimes B_0)$$

Eta noiz izango da $A < B$? Hona hemen erlazio horri dagokion adierazpena:

$$(A < B)_{2bit} = \overline{A_1} B_1 + (A_1 \otimes B_1) \overline{A_0} B_0$$

(Irakurlearentzat utziko dugu horri buruzko hausnarketa.)

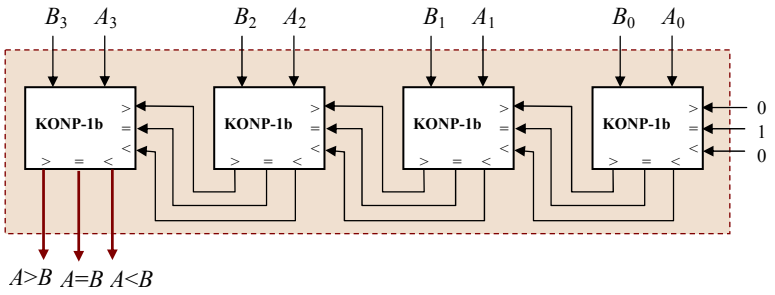
Hiru funtzio horiek sortzeko, A_1 eta B_1 bitei dagokien konparazioaz gain, A_0 eta B_0 biten konparagailuaren funtzioak erabili behar dira. Horiek ordezkatuz, honako adierazpen hauek izango ditugu:

$$(A > B)_{2bit} = A_1 \overline{B_1} + (A_1 \otimes B_1)(A > B)_{1bit}$$
$$(A = B)_{2bit} = (A_1 \otimes B_1)(A = B)_{1bit}$$
$$(A < B)_{2bit} = \overline{A_1} B_1 + (A_1 \otimes B_1)(A < B)_{1bit}$$

Beraz, n biteko konparagailuaren emaitzak n–1 biteko konparagailuaren emaitzetan oinarrituko dira, honelaxe:

$$(A > B)_{nbit} = A_{n-1}\overline{B_{n-1}} + (A_{n-1} \otimes B_{n-1})(A > B)_{(n-1)bit}$$
$$(A = B)_{nbit} = (A_{n-1} \otimes B_{n-1})(A = B)_{(n-1)bit}$$
$$(A < B)_{nbit} = \overline{A_{n-1}}B_{n-1} + (A_{n-1} \otimes B_{n-1})(A < B)_{(n-1)bit}$$

Ageri denez, azken biten konparaziorako, azkenaurrekoen konparazioaren emaitzak behar dira (eta abar). Hori dela eta, bit bateko konparagailuak, sarrerako datuen bitez gain, aurreko "urratsaren" emaitzak ere —aurreko blokearen irteerako hiru funtzioak, alegia— hartu beharko ditu sarreratzat. 3.22. irudian, 4 biteko konparagailu bat ageri da, bit bateko 4 konparagailu erabiliz.



3.22. irudia. *n* biteko konparagailua.

3.22. irudian ageri den moduan, pisu txikieneko blokearen aurrean beste konparagailurik ez dagoenez, blokeak lotzeko erabiltzen diren hiru sarreretan sartu beharreko balioek konstanteak izan behar dute (zer balio jarri behar diren jakiteko, konparaketaren ekuazioak aztertu behar dira, eta irakurlearentzat uzten dugu justifikazioa).

Diseinatu dugun konparagailuak soilik zenbaki arruntak konparatzeko balio du. Izan ere, zenbaki osoak konparatzeko, haien zeinuak kontuan hartu beharko genituzke, adierazpide-sistemaren arabera. Adibidez, konparatu nahi ditugu 2rako osagarriko bi zenbaki hauek: A = 1001 (–7) eta B = 0110 (+6). B zenbakia A zenbakia baino handiagoa bada ere, konparagailuak kontrakoa adieraziko du, A zenbakiaren pisu handieneko bita 1 delako eta B zenbakiarena 0 delako (ikus aurreko ekuazio logikoak).

Hori dela eta, zenbaki osoak konparatu behar direnean, beharrezkoa izango da konparagailu egokiak diseinatzea zenbakien adierazpidearen arabera, ariketa ebatzietan ikusiko dugun legez.

3.7. UNITATE ARITMETIKO/LOGIKOAK

Sistema digital askotan (konputagailuetan, esaterako), datuak (zenbakiak) prozesatzen dira. Datu horien gainean, hainbat eragiketa egiten dira, aritmetikoak —batuketa, kenketa, gehikuntza...— zein logikoak —*and*, *or*, *xor*...—. Eragiketa horietarako, bloke konbinazional garrantzitsu bat erabiltzen da: Unitate Aritmetiko/Logikoa, UAL (*Arithmetic and Logic Unit*, *ALU*). Orain arte aztertu ditugun zirkuituak baino konplexuagoak dira UALak, eskuarki LSI integrazio-mailakoak.

UAL bat eragiketa asko egiteko gauza denez, nolabait adierazi behar zaio egin behar duena. Horretarako, eragiketa-kode bat erabiltzen du UALak kontrol-seinale gisa.

Honako sarrera eta irteera hauek dituzte UALek:

- Datu-sarrerak

  A, B: Prozesatu nahi diren *n* biteko datuak.

  $\boldsymbol{C_{in}}$**:** Bit bat, batuketaren sarrerako bururakoa.

- Datu-irteerak

  Y: Eragiketaren emaitza, *n* bitekoa.

  Emaitzaz gain, berari buruzko informazioa ematea ohikoa da, hainbat adierazleren bidez. Esaterako,

  $\boldsymbol{C_{out}}$**:** Bit bat, batuketaren irteerako bururakoa.

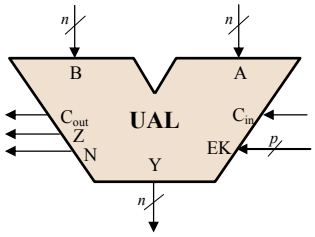
  Z: Bit bat, emaitza 0 dela adierazteko; emaitza 0 baldin bada, $Z = 1$ izango da; emaitza zero ez bada, ordea, $Z = 0$ izango da.

  N: Bit bat, emaitza negatiboa dela adierazteko; $N = 1$ izango da emaitza negatiboa denean, eta bestela $N = 0$.

- Kontrol-seinaleak

  EK: *p* biteko eragiketa-kodea, egin beharreko eragiketa adierazteko. *p* bit erabiliz, 2^p eragiketa adieraz daitezke.

Oro har, unitate aritmetiko/logikoek 2rako osagarrian adierazitako zenbakiak prozesatzen dituzte. 3.23. irudian UAL bat adierazteko ohiko ikurra ageri da.



3.23. irudia. UALak adierazteko ohiko ikurra.

Aztertzen ari garen gainerako oinarrizko blokeak bezala, konplexutasun txikiagoko blokeak erabiliz sor daitezke UALak. Ikus dezagun, adibide gisa, honako lau eragiketa hauek egiten dituen UALa.

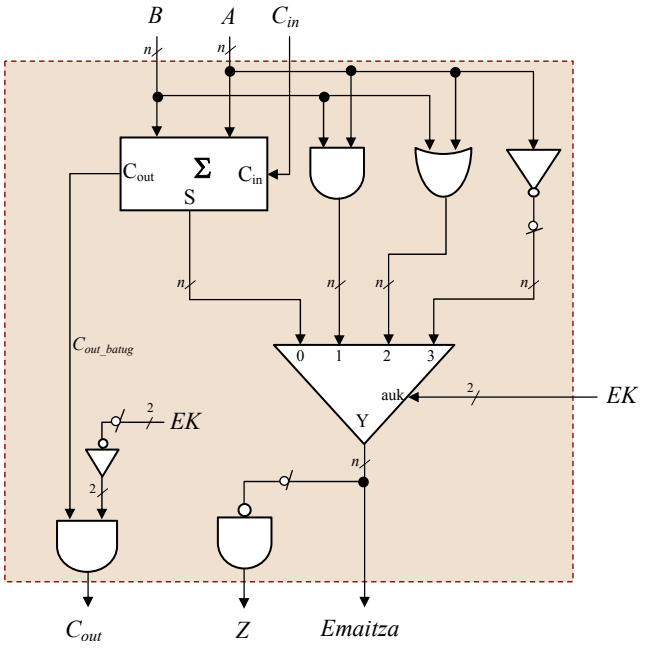
Eragiketa-kodea EK_1EK_0	Emaitza (eragiketa)
00	$A + B$
01	*A and B*
10	*A or B*
11	$\overline{A}$

Irteera gisa, emaitzaz gain, honako adierazle hauek ere sortu behar ditu: *Z*, emaitza zero denetz adierazteko, eta C_{out} irteerako bururakoa, batuketa egiten denean soilik.

UAL hori sortzeko, batetik, batugailu bat erabili behar dugu, batuketak egiteko, eta, bestetik, AND, OR eta NOT ateak. Eragiketa-kodearen arabera, aukeratuko dugu dagokion emaitza irteerarako. Beraz, lau emaitza horietatik bat aukeratzeko, 4 sarrerako multiplexore bat erabiliko dugu, non eragiketa-kodea multiplexorearen hautatze-seinalea izango den. *and*, *nor* eta *not* funtzio bakoitzerako, *n* ate logiko erabili behar dira, eragiketa logikoa bitez bit egin ahal izateko.

3.24. irudian ageri da UAL sinple horren eskema logikoa.

Emaitzaz gain, bi adierazle sortu dira. *Z* adierazleak emaitza 0 dela adierazi behar du; beraz, $Z = \overline{Y_{n-1}}\ \overline{Y_{n-2}} \ldots \overline{Y_1}\ \overline{Y_0}$ izango da. Bestalde, C_{out} sortu behar da, batuketa egiten denean (eragiketa-kodea 00), eta batugailuaren irteerako bururakoa 1 denean: $C_{out} = \overline{EK_1}\ \overline{EK_0}\ C_{out\_batug}$.



3.24. irudia. 4 eragiketako UAL baten egitura oinarrizko bloke konbinazionalak erabiliz.

Adibiderako erabili ditugun eragiketez gain, ohikoak dira UALetan, esaterako, beste hauek ere: *Clear* (*emaitza* = 0), *Preset* (*emaitza* = 1), *A*, *B*, *A* – *B*, *B* – *A*, *A* + 1, *A* – 1, *A xor B*, *A* > *B*, desplazamenduak eskuinera eta ezkerrera, eta abar.

Laburpena. Bloke konbinazional behinenak aztertu ditugu kapitulu honetan. Bloke konbinazional baten erantzuna sarreren uneko balioen araberakoa da; ez dute memoriarik. Sistema digital askotan erabili behar direnez, eskuragarriak dira bloke horiek, haiek erabiliz sistema konplexuagoak egiteko. Datu-sarrerez eta datu-irteerez gain, bloke konbinazional gehienek kontrol-seinaleak erabiltzen dituzte eragiketa jakin bat egiteko; esaterako, hautatze-kode bat multiplexoreetan sarreraren bat aukeratzeko, edo eragiketa-kodea UALetan. Hainbat bloketan, gainera, gaikuntza-seinale bat dago, blokearen funtzionamendu osoa kontrolatzeko: aktibatuta dago edo ez dago aktibatuta.

Hurrengo atalean, bloke konbinazionalen funtzionamendua landuko dugu, hainbat ariketa ebatziz.

3.8. ARIKETA EBATZIAK

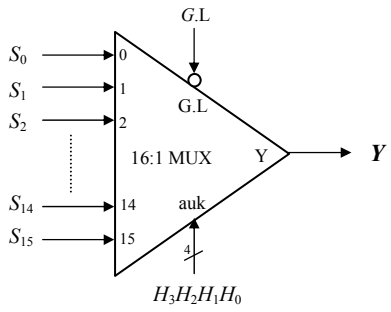
MSI mailako oinarrizko bloke konbinazionalak aztertu ditugu kapitulu honetan: multiplexoreak, deskodegailuak, batugailuak, konparagailuak… Orain, ariketa batzuk ebatziko ditugu zirkuitu sinpleak diseinatzeko asmoz. Diseinu horietan, oinarrizko bloke konbinazionalak eta aurreko kapituluan aztertutako ateak erabili beharko ditugu. Helburua bikoitza da; batetik, zirkuitu konbinazionalak erabiltzen trebetasuna hartzea, eta, bestetik, diseinu sinpleak egiten lehenengo urratsak ematea. Lehenengo lau ariketetan multiplexorea eta deskodegailua oinarrizko blokeak landuko ditugu; gero, 3.5. eta 3.6. ariketetan, batuketa-zirkuituak egingo ditugu, hainbat zenbaki-adierazpidetarako; 3.7. ariketan, zeinu/magnitudeko konparagailua diseinatuko dugu; azkenik, 3.8., 3.9. eta 3.10. ariketetan, funtzio aritmetiko batzuk egiteko zirkuitu konbinazionalak sortuko ditugu, tartean UALak erabiliz.

Zirkuitu jakin bat sortu behar den ariketetan, aukera bat baino ez da izango proposatuko dugun soluzioa; ez da, beraz, soluzio bakarra, eskuarki modu bat baino gehiago izango baita portaera jakin bat duen zirkuitua sortzeko.

3.1. Ariketa

Eraiki ezazu 16 datu-sarrerako multiplexore bat, 4 datu-sarrerako multiplexoreak erabiliz.

16 datu-sarrerako multiplexorearen eskema logikoa irudikoa da. S_0–S_{15} datu-sarreretatik bat aukeratzeko, lau hautatze-seinale behar dira (H_3–H_0). Sarrera horiez gain, ohikoa da gaikuntza-seinalea (*G*.L) izatea. Azkenik, irteera bat (*Y*) izango du hautatutako sarreran dagoen balioa emateko.



Egin nahi dugun multiplexorearen 16 datu-sarrerak 4 datu-sarrerako multiplexoretan banatu behar baditugu, gutxienez 4 multiplexore beharko ditugu. Hala, S_0–S_3 sarrerak lehenengo multiplexorean sartuko ditugu, S_4–S_7 bigarrenean, S_8–S_{11} hirugarrenean, eta, azkenik, S_{12}–S_{15} laugarrenean, 3.25. irudian ageri den moduan.

Lau sarrerako multiplexoreek irteera bana dute, eta, beraz, lau horietatik bat aukeratu beharko dugu, egin nahi dugun 16 sarrerako multiplexoreari dagokiona, hain zuzen ere. Hala, lau irteera horiek irteera bakarrera bideratzeko, 4 sarrerako beste multiplexore bat erabiliko dugu, non behar dugun irteera aukeratu ahal izango baitugu.

Datuak ondo aukeratzeko, lau hautaketa-seinaleak (H_3–H_0) era egokian banatu behar dira aipatu ditugun bost multiplexoreen artean. Beheko taulan ageri den moduan, pisu handieneko bi bitak, H_3 eta H_2, ez dira aldatzen lehenengo lau datu-sarreretarako (S_0–S_3), eta, ondorioz, sarrera horien arteko hautaketa H_1 eta H_0-ren bidez egiten da. Hori bera gertatzen da datu-sarrera multzo bakoitzean, launaka.

Beraz, lau datu-sarrerako multzo bakoitzean, H_1 eta H_0 hautatze-seinaleak erabiliko ditugu datu bat aukeratzeko; azkenik, multzo bakoitzaren emaitzetatik bat aukeratzen duen multiplexorean, H_3 eta H_2 hautatze-seinaleak erabiliko ditugu. Azter dezagun adibide bat.

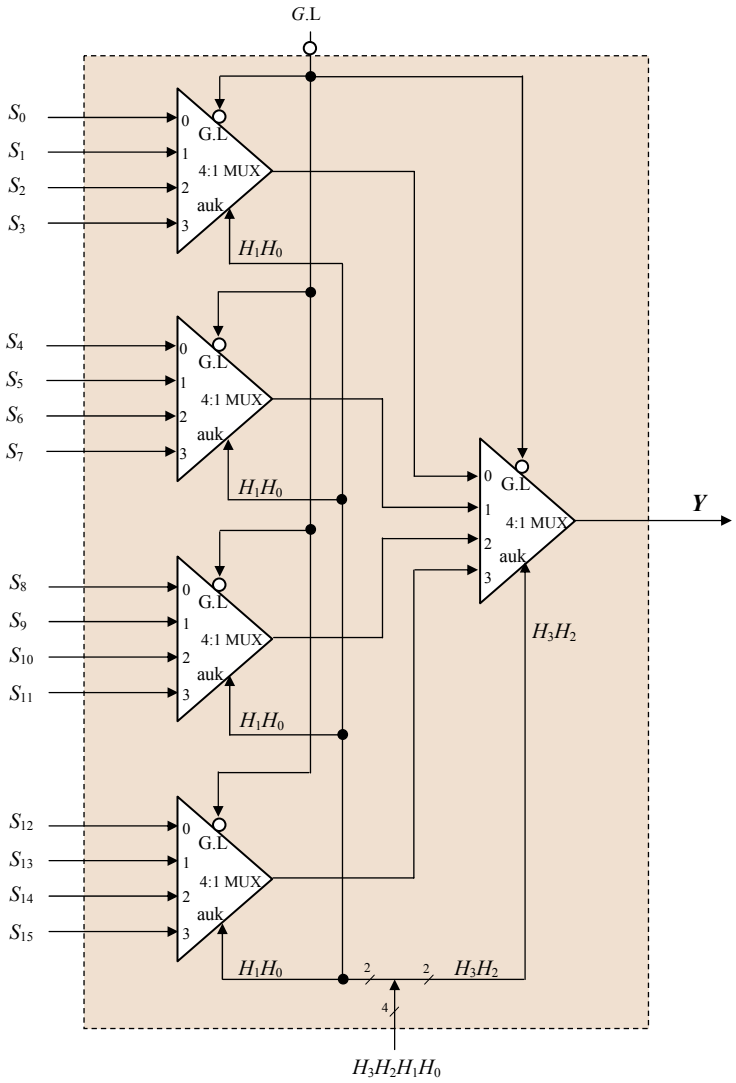
H_3	H_2	H_1	H_0	Y
0	0	0	0	S_0
0	0	0	1	S_1
0	0	1	0	S_2
0	0	1	1	S_3
0	1	0	0	S_4
0	1	0	1	S_5
0	1	1	0	S_6
0	1	1	1	S_7
1	0	0	0	S_8
1	0	0	1	S_9
1	0	1	0	S_{10}
1	0	1	1	S_{11}
1	1	0	0	S_{12}
1	1	0	1	S_{13}
1	1	1	0	S_{14}
1	1	1	1	S_{15}

Demagun S_6 datu-sarrera hautatu nahi dugula; horretarako, 0110 konbinazioa jarriko dugu H_3–H_0 hautatze-seinaleetan. H_1H_0 = 10 denez, 2 sarrera aukeratuko da lau multiplexoreetan; haien irteeretan, beraz, S_2, S_6, S_{10} eta S_{14} eskuratuko ditugu, hurrenez hurren. Azkenik, irteerako multiplexorean, H_3 eta H_2-rekin egingo da hautaketa, eta seinale horien balioak 0 eta 1 direnez, hautatuko den sarrera bigarren multiplexoreak ateratakoa izango da, S_6 alegia.

16 sarrerako multiplexorearen diseinua bukatzeko, G seinalea nola konektatu behar den aztertu behar dugu. Seinale horrek sistema osoa gaitu behar du; beraz, nahikoa da multiplexore guztien gaikuntza-sarreretara

eramatea. Hala, G aktibaturik dagoenean bakarrik erantzungo dute multiplexoreek.

3.25. irudian, 16 sarrerako multiplexorearen eskema ageri da:



3.25. irudia. 16 datu-sarrerako multiplexorea.

3.2. Ariketa

Eraiki ezazu f(d, c, b, a) = Σ (0, 3, 6, 7, 11, 14, 15) funtzioa hiru modu hauetan:

(a) 16 datu-sarrerako multiplexorea erabiliz;

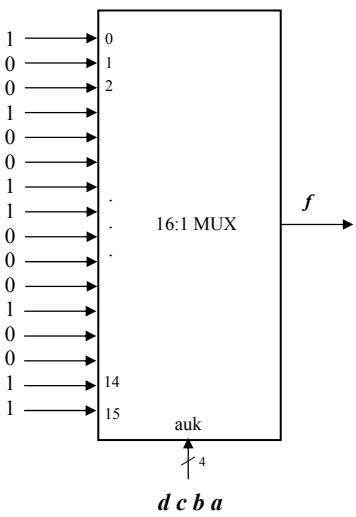
(b) 8 datu-sarrerako multiplexorea eta behar diren ateak erabiliz; eta

(c) 4 datu-sarrerako multiplexorea eta behar diren ateak erabiliz.

(a) atala

Funtzioak 1eko balioa eman behar du 0, 3, 6, 7, 11, 14 eta 15 *minterm*-etarako. Funtzio hori multiplexore batekin eraiki nahi badugu, lau aldagaikoa denez, era sinple bat izango da 16 datu-sarrerako multiplexore bat erabiltzea: hautaketa-lerroak sarrerako aldagaiak izango dira —*d*, *c*, *b* eta *a*—; eta datu-sarreretan, 0 edo 1 jarriko dugu, irteeran lortu nahi dugunaren arabera. Hau da, 0, 3, 6, 7, 11, 14 eta 15 sarreretan 1ekoak jarriko ditugu, eta gainerakoetan 0koak. 3.26. irudian, funtzioaren sarrera-konbinazio bakoitzeko, multiplexorearen sarreretan jarri beharreko balioak ageri dira. Alboan, funtzioaren gauzatzea ageri da.

d	*c*	*b*	*a*	*f*	16:1 mux
0	0	0	0	1	**1**
0	0	0	1	0	**0**
0	0	1	0	0	**0**
0	0	1	1	1	**1**
0	1	0	0	0	**0**
0	1	0	1	0	**0**
0	1	1	0	1	**1**
0	1	1	1	1	**1**
1	0	0	0	0	**0**
1	0	0	1	0	**0**
1	0	1	0	0	**0**
1	0	1	1	1	**1**
1	1	0	0	0	**0**
1	1	0	1	0	**0**
1	1	1	0	1	**1**
1	1	1	1	1	**1**



3.26. irudia. Lau aldagaiko funtzioaren egia-taula eta haren gauzatzea 16 datu-sarrerako multiplexorea erabiliz.

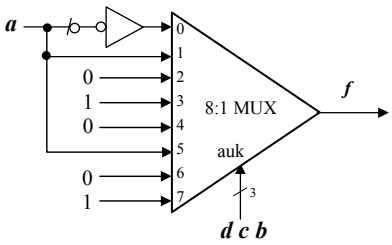
(b) atala

Multiplexore txikiagoa erabili nahi badugu, 8 datu-sarrerakoa esaterako, lau aldagaietatik hiru erabiliko ditugu hautatze-seinale moduan, eta, ondorioz, funtzioaren balioak binaka hartu beharko ditugu, multiplexorearen sarrerak definitzeko. Hainbat modutan aukera daitezke, lau aldagaien artean, behar ditugun hiru hautatze-seinaleak; adibidean, *d*, *c* eta *b* aldagaiak erabiliko ditugu. Hala, multiplexorearen 0 sarrerarako, m_0 eta m_1 *minterm*-ak kontuan hartu beharko ditugu; 1 sarrerarako, m_2 eta m_3 *minterm*-ak, eta abar, 3.27. irudiko taulan ageri den moduan. Beraz, kasu guztietan, multiplexorearen datu-sarrerak *a* aldagaiaren araberakoak izango dira. *dcb* aldagaien konbinazio bakoitzerako, lau aukera izango ditugu multiplexorearen sarrerak definitzeko:

a	*f*	**multiplexorearen sarrera**
0 1	0 0	0
0 1	0 1	*a*
0 1	1 0	$\overline{a}$
0 1	1 1	1

Lau aukera horietatik bi kasutan, *f* funtzioa (eta, ondorioz, dagokion multiplexorearen sarrera-balioa) konstantea da: 0 edo 1. Beste bietan, *f* funtzioa *a* aldagaiaren araberakoa da: $\overline{a}$ edo *a*. Esaterako, ariketa honetako m_4 eta m_5 *minterm*-etarako, $f = 0$ da; m_6 eta m_7 *minterm*-etarako, $f = 1$; m_2 eta m_3 *minterm*-etarako, $f = a$; eta m_0 eta m_1 *minterm*-etarako, $f = \overline{a}$. 3.27. irudian ageri dira ariketako funtzioaren egia-taula eta multiplexorearen sarrerak.

	d	*c*	*b*	*a*	*f*	8:1 mux
m_0	0	0	0	0	1	$\overline{a}$
m_1	0	0	0	1	0	
m_2	0	0	1	0	0	*a*
m_3	0	0	1	1	1	
m_4	0	1	0	0	0	0
m_5	0	1	0	1	0	
m_6	0	1	1	0	1	1
m_7	0	1	1	1	1	
m_8	1	0	0	0	0	0
m_9	1	0	0	1	0	
m_{10}	1	0	1	0	0	*a*
m_{11}	1	0	1	1	1	
m_{12}	1	1	0	0	0	0
m_{13}	1	1	0	1	0	
m_{14}	1	1	1	0	1	1
m_{15}	1	1	1	1	1	

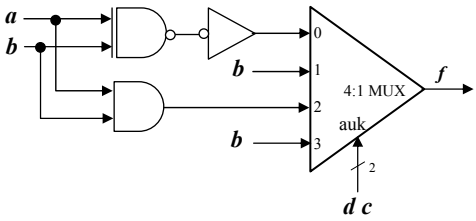


3.27. irudia. Funtzioa 8 datu-sarrerako multiplexorea erabiliz.

(c) atala

4 datu-sarrerako multiplexorea erabiltzeko, aurreko prozesuari jarraitu beharko diogu, baina funtzioaren irteerak 4naka hartuz. Kasu honetan, hautatze-seinale gisa, *d* eta *c* aldagaiak erabiliko ditugu, eta datu-sarreratzat, konstanteak edo *a* eta *b* aldagaien funtzioren bat izango dugu. Adibidez, *d* eta *c* 0 direnean (taularen lehenengo 4 errenkadetan), *f*-ren balioak 1, 0, 0, eta 1 dira, hau da, *b* eta *a* aldagaien *equ* funtzioa, hain zuzen ere. 3.28. irudian ageri dira irteerei dagozkien adierazpenak: $a \otimes b$, b, $a \cdot b$, eta b hurrenez hurren. Alboan, zirkuitua dago egina, 4:1 multiplexorea eta behar diren ateak erabiliz.

	d	*c*	*b*	*a*	*f*	4:1mux
m_0	0	0	0	0	1	$a \otimes b$
m_1	0	0	0	1	0	
m_2	0	0	1	0	0	
m_3	0	0	1	1	1	
m_4	0	1	0	0	0	b
m_5	0	1	0	1	0	
m_6	0	1	1	0	1	
m_7	0	1	1	1	1	
m_8	1	0	0	0	0	$a \cdot b$
m_9	1	0	0	1	0	
m_{10}	1	0	1	0	0	
m_{11}	1	0	1	1	1	
m_{12}	1	1	0	0	0	b
m_{13}	1	1	0	1	0	
m_{14}	1	1	1	0	1	
m_{15}	1	1	1	1	1	



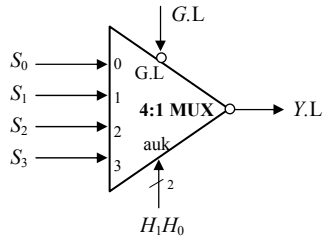
3.28. irudia. Funtzioa 4 datu-sarrerako multiplexorea erabiliz.

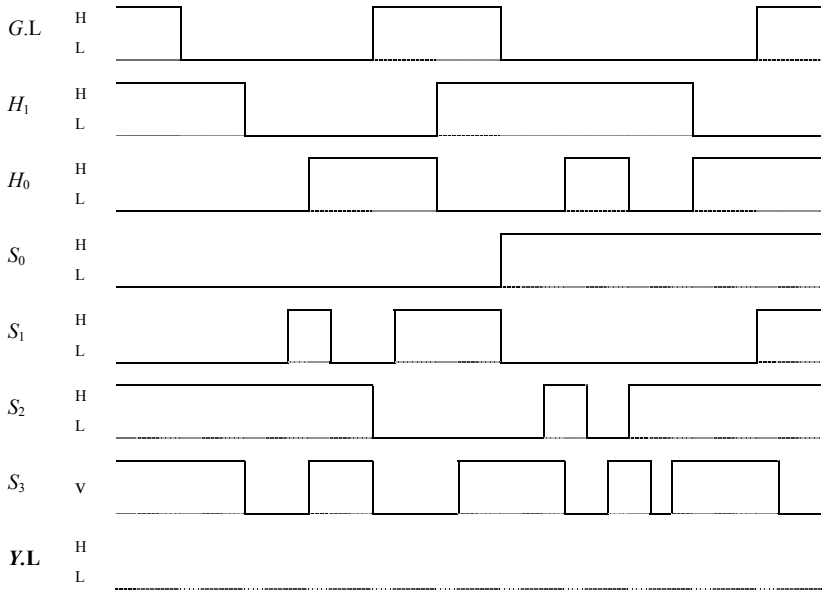
Aukera asko ditugu, beraz, funtzio bat sortzeko multiplexoreen bidez. Zein den egokiena jakin nahi badugu, erantzun zehatzik ez dagoela esan beharra dago. Hainbat irizpideren arabera aukera baitaiteke bata edo bestea. Esaterako, erabiltzen den hardwarearen arabera, (b) aukera izango litzateke, kasu horretan, soluziorik egokiena: ate logikoak erabili gabe, multiplexore txikiena erabiltzen baitu. Oro har, *n* aldagaiko funtzioa multiplexore batekin eraikitzeko, ohikoena da $n - 1$ hautatze-seinalekoa erabiltzea.

«

3.3. Ariketa

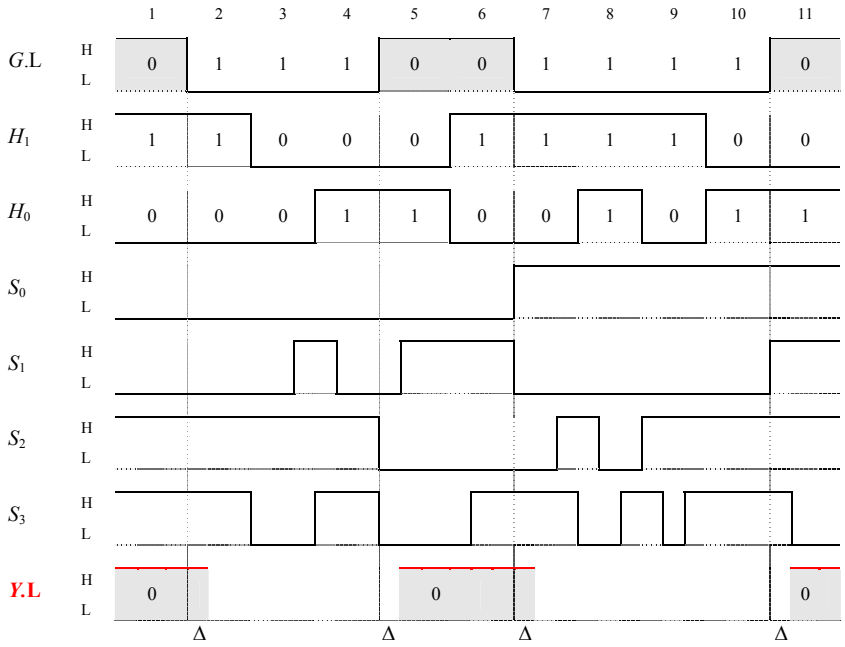
Lau sarrerako multiplexore baten datu-sarrerek eta gaikuntza-seinaleak kronograman ageri den portaera dute denboran zehar. Marraz ezazu, kronograman, multiplexoreak irteeran (Y.L) izango duen balioa. Adi! multiplexorearen irteera logika negatiboan dago.





Kronograma betetzeko, lehendabizi, komeni zaigu markatzea gaikuntza-seinalea (*G*.L) noiz dagoen desaktibatuta (grisez hurrengo irudian); izan ere, *G* desaktibatuta dagoenean (H tentsioa, logika negatiboan dagoelako), $Y = 0$ da. Adi! Multiplexorearen erantzuna ez da bat-batekoa, denbora-tarte bat, txikia bada ere, behar baitu sarrerak prozesatzeko: Δ. Kronograman bertan islatu dugu portaera hori.

Gero, $G = 1$ den tarteetan, multiplexorearen hautatze-lerroen (H_1 eta H_0) balioen aldaketak zein unetan gertatzen diren markatuko dugu, eta, bide batez, seinale horien balio logikoak idatziko ditugu.

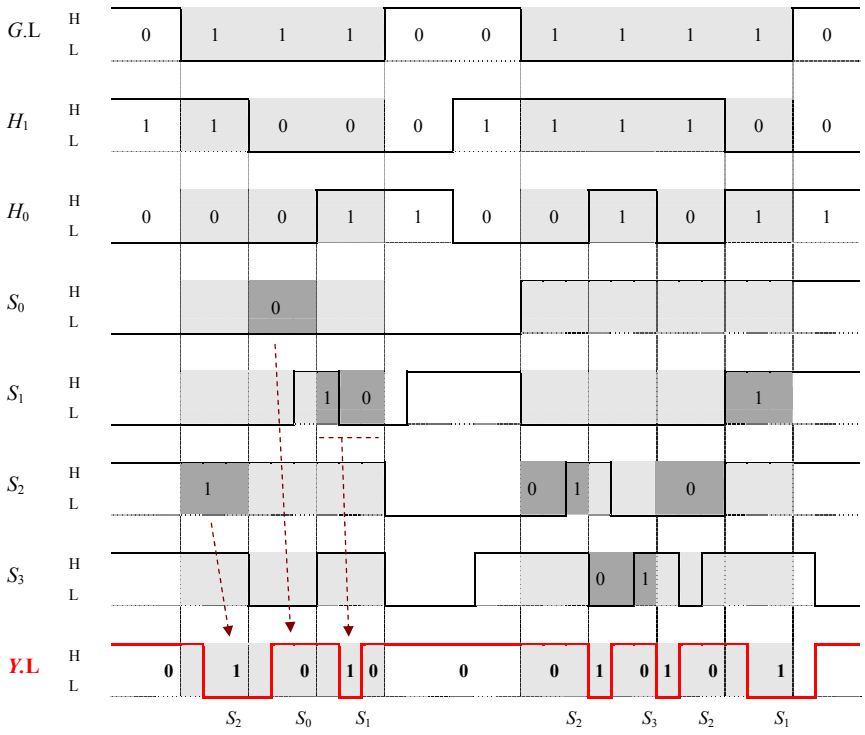


Hala, tarte bakoitzean, datu-sarrera bakarrari egingo diogu so, eta hori izango da multiplexorearen irteeran ageriko dena. Kronograman, Y funtzioak tarte horietan hartzen dituen balio logikoak (0 edo 1) zein fisikoak (grafikoki marra goian edo behean eginez) adieraziko ditugu. Irteeran, une bakoitzean hautatutako sarreraren balio logikoa mantenduko da. Irteeraren balio fisikoa, berriz, hautatutako sarreraren aurkakoa izango da, sarrerak logika positiboan baitaude, eta irteera, aldiz, logika negatiboan, Y.L.

Esate baterako, 2. zutabean, $H_1 = 1$ eta $H_0 = 0$ direnean, S_2 sarrera da hautatzen dena, eta, ondorioz, irteerara doana. Beraz, tarte horretan $S_2 = 1$ denez, $Y = 1$ izango da (L tentsioa). Bestalde, 4. zutabean, $H_1 = 0$ eta $H_0 = 1$ direnean, S_1 sarrera da irteerara doana. Tarte horretan, S_1 hasieran 1 da eta gero 0 balioa hartzen du; beraz, irteeran ere aldaketa hori islatuko da (ikusi hurrengo irudia).

Irteeraren balioa adieraztean, zirkuituaren erantzun-denbora izan dugu kontuan, eta, hori dela eta, irteeraren aldaketak desplazatuta daude markatuta dauden sarrerekiko.

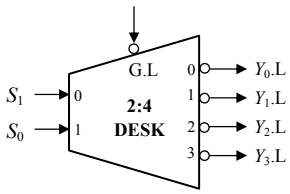
Hona hemen kronograma beteta:

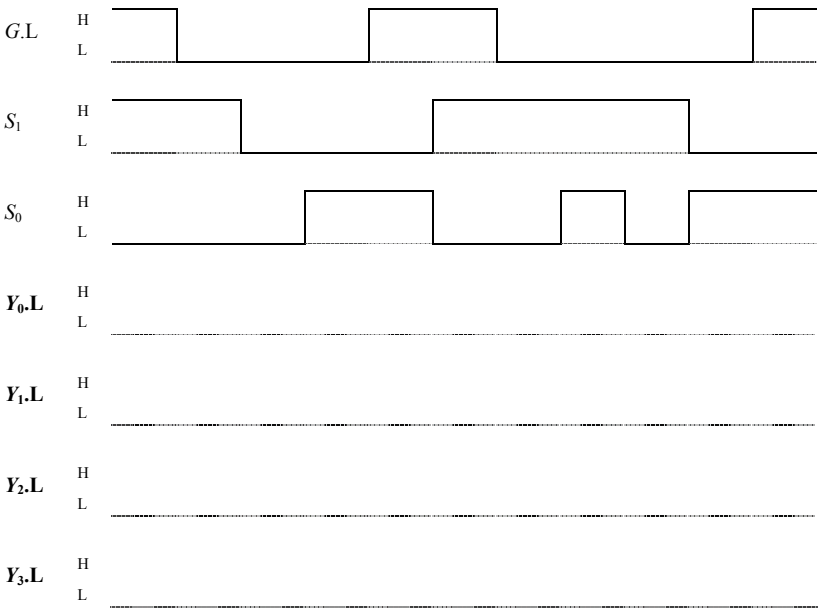


«

3.4. Ariketa

Irudian, bi sarrerako eta lau irteerako deskodegailua ageri da. Bere portaera kronograma baten bidez azaldu nahi dugu. Kronograman adierazita daude sarrerek hartzen dituzten balioak denboran zehar, eta irteeren balioak kalkulatu behar dira.



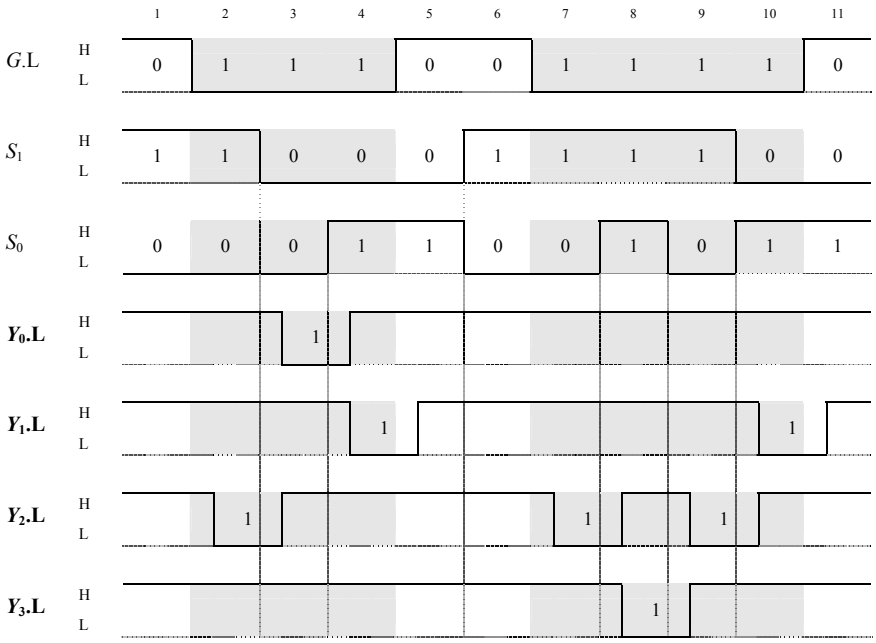


■

Aurreko ariketan bezala, lehendabizi gaikuntza-seinalea noiz dagoen aktibatuta azpimarratuko dugu. Gero, kronograma betetzeko, komeni da markatzea sarrerako balioen aldaketak zein unetan gertatzen diren, eta, tarte bakoitzean, adieraziko dugu hartzen duten balio logikoa. Azkenik, Y_i funtzioek tarte horietan hartzen dituzten balio logikoak (0 edo 1) zein fisikoak adieraziko ditugu.

Esaterako, G aktibaturik egonik, S_1 eta S_0-ren balioak 1 eta 0 badira (kronogramaren 2. zutabea), Y_2.L irteera aktibatuko da, hau da, 1 balio logikoa hartuko du, eta L tentsioa izango du, logika negatiboan baitago. 3. zutabean, berriz, S_1 eta S_0-ren balioak 0 dira eta, ondorioz, Y_0.L irteera aktibatuko da (1eko balio logikoa eta tentsio baxua, L).

Hala, tarte bakoitzeko balioak kontuan hartuta, kronograma osoa beteko dugu.

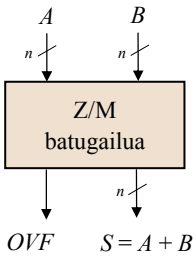


Irteeren balioak adieraztean, zirkuituaren erantzun-denbora izan dugu kontuan, eta, hori dela eta, irteerak desplazatuta daude sarrerekiko.

«

3.5. Ariketa

Zenbaki bitar arruntak zein birako osagarrian adierazitako osoak batzen dituen oinarrizko bloke konbinazionala aztertu dugu 3.5. atalean. Hainbat kasutan, hala ere, beste adierazpideak erabiliz kodetutako zenbakiak batu behar dira. Adibide gisa, zeinu/magnitudea adierazpideko zenbakien batugailua diseinatuko dugu ariketa honetan.



Sarreran, n biteko bi zenbaki hartzen ditu, eta, irteeran, haien batura eta eragiketaren balizko gainezkatzea (overflow) eskaintzen ditu.

■

Batugailua egin baino lehen, zeinu/magnitudea adierazpidearen xehetasunak azalduko ditugu (ikus E1 eranskina). Adierazpide horretan, zeinua eta magnitudea bereizten dira zenbaki-kodean. Pisu handieneko bitak zeinua adierazten du: 0 → positiboa eta 1 → negatiboa. Gainerako bitek zenbakiaren magnitudea adierazten dute, bitar arruntean.

Adibidez, 4 bitetan:

$0\,110 \rightarrow 6 \qquad 1\,011 \rightarrow -3$

Hala, n biteko $A(A_{n-1}, A_{n-2}, ..., A_0)$ zenbakian, honako banaketa hau egin dezakegu:

A_{n-1}	$\rightarrow Z_A$	A zenbakiaren zeinua (bit bat)
$A_{n-2}, ..., A_0$	$\rightarrow M_A$	A zenbakiaren magnitudea (n–1 bit)

Zeinu/magnitudea adierazpideko zenbakiak batzeko, batugailu arruntak eta beste bloke konbinazional eta ate logiko batzuk erabili behar dira.

Batuketa-algoritmoa sinplea da:

- Zenbakien zeinuak berdinak badira, hau da, biak positiboak edo biak negatiboak badira, batugaien magnitudeak batu behar dira, eta emaitzaren zeinua batugaiena izango da. Adibidez (4 bitetan):

  (+3) + (+2) = +5 → 0 011 + 0 010 = 0 101

  (–1) + (–5) = –6 → 1 001 + 1 101 = 1 110

- Zenbakien zeinuak desberdinak badira, bi magnitudeak konparatu behar dira, handienari txikiena kendu behar baitzaio. Emaitzaren zeinua magnitude handieneko zenbakienarena izango da. Adibidez (4 bitetan):

  (+3) + (–2) = +1 → 0 011 + 1 010 = 0 001

  (–7) + (+4) = –3 → 1 111 + 1 100 = 1 011

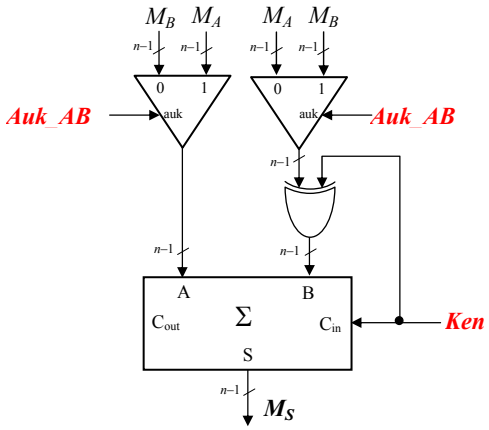
Batuketa-algoritmoa taula batean bil daiteke, honela:

$$\boldsymbol{S}(Z_S, M_S) = \boldsymbol{A}(Z_A, M_A) + \boldsymbol{B}(Z_B, M_B)$$

$Z_A = Z_B$	$M_A > M_B$	emaitza	
1	—	$Z_S = Z_A = Z_B$	$M_S = M_A + M_B$
0	1	$Z_S = Z_A$	$M_S = M_A - M_B$
0	0	$Z_S = Z_B$	$M_S = M_B - M_A$

Emaitza lortzeko, beraz, batuketa zein kenketa arruntak egin behar dira; hau da, ezagutzen dugun batugailua/kengailua erabili behar da. Hala ere, eragigaiak ez dira kasu guztietan berdinak; taulan ageri den moduan, lehen eragigaia M_A zein M_B izan daiteke, eta gauza bera gertatzen da bigarren eragigaiarekin.

Honako zirkuitu hau antola daiteke batuketa/kenketa horiek egiteko:



Batugailuan, batuketak zein kenketak egiten dira, *Ken* kontrol-seinalearen arabera (ikus 3.5.3. atala). Era berean, multiplexoreen bidez, eragigaiak aukeratzen dira: M_A eta M_B, $Auk\_AB = 1$ denean, eta M_B eta M_A $Auk\_AB = 0$ denean. Bi multiplexoreetan, hautaketa-seinalea bera da: Auk_AB.

Eragiketa zein batugaiak hautatzeko, zenbakien zeinuak eta magnitudeak kontuan hartu behar ditugu (ikus batuketa-algoritmoa). Zeinuak desberdinak direnean (*xor* funtzioa), kenketa egin behar da; hau da, $Ken = 1$ izan behar du. Batugaiak, aldiz, honela hautatu behar dira: $M_A > M_B$ denean, M_A eta M_B (hau da, $Auk\_AB = 1$); bestela, M_B eta M_A ($Auk\_AB = 0$).

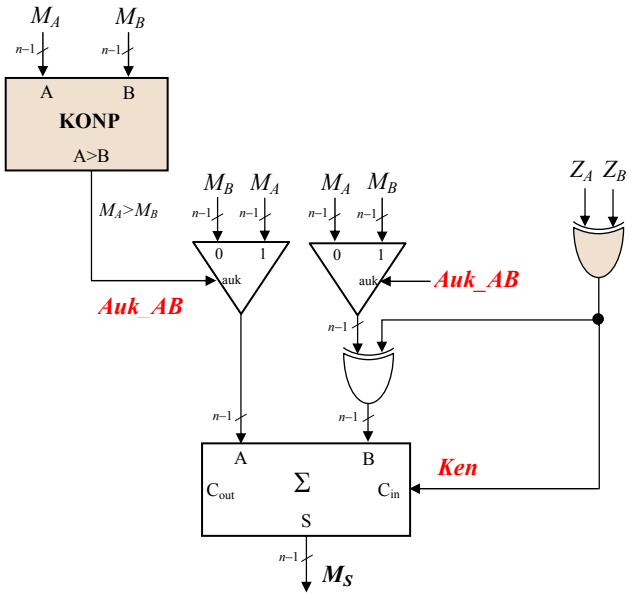
Logika hori guztia taula honetan laburbiltzen da:

$Z_A = Z_B$	$M_A > M_B$	Ken	Auk_AB
1	—	0	—
0	1	1	1
0	0	1	0

Hau da, bi funtzio logiko hauek sortu beharko ditugu (minimizatuta):

$$Ken = Z_A \oplus Z_B$$
$$Auk\_AB = M_A > M_B$$

Horretarako, XOR ate bat eta konparagailu bat erabiliko ditugu. Beraz, honako zirkuitu hau izango dugu:



Bi gauza falta zaizkigu batugailua bukatzeko: emaitzaren zeinua eta gainezkatzea.

Batuketa-algoritmoa laburbiltzen duen taulan ageri den moduan, emaitzaren zeinua Z_A edo Z_B balioen artean aukeratu behar da, batzen diren zenbakien magnitudeen arabera. Hau da zeinua definitzen duen ekuazio logikoa:

$$Z_S = Z_A\ (M_A > M_B) + Z_B\ (\overline{M_A > M_B})$$

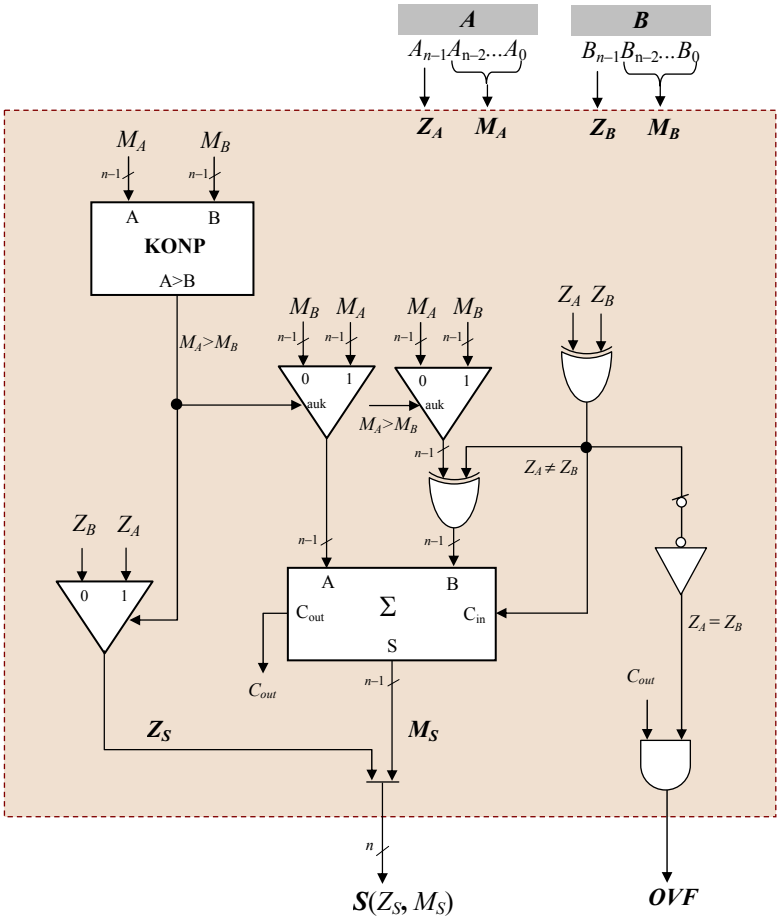
Izan ere, hori bera da multiplexore baten ekuazio logikoa: aukeratu Z_A edo Z_B seinale baten (konparagailuaren irteeraren) balioaren arabera. Beraz, nahikoa da bi sarrerako multiplexore bat funtzio hori gauzatzeko.

Azkenik, eragiketaren balizko gainezkatzea sortu behar dugu. Gainezkatzea gerta daiteke kasu bakar batean: n–1 biteko bi magnitudeak batzean (zeinuak berdinak direlako), baturak n bit behar dituenean; hau da, batugailuaren C_{out} irteera aktibatzen denean. Beste kasuetan, kenketak egiten direnean, ez da inoiz gainezkatzerik sortuko.

Beraz, honako ekuazio logiko hau izango du *OVF* funtzioak:

$$OVF = (\overline{Z_A \oplus Z_B})\ C_{out}$$

Osagai guztiak kontuan hartuta, 3.29. irudian ageri da zeinu/magnitudeko bi zenbakien batugailua.

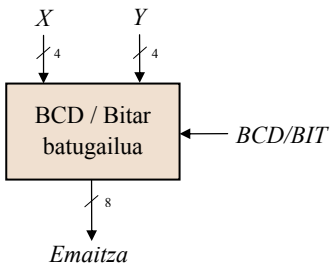


3.29. irudia. Zeinu/magnitudeko batugailua.

«

3.6. Ariketa

Batugailu bat diseinatu nahi dugu, BCD kodean zein bitar hutsez adierazitako zenbaki arruntak batzeko. Kontrol-seinale batek —BCD/BIT— datuen adierazpidea esango digu: BCD/BIT = 1 bada, batuketa BCDz egin beharko da; aldiz, BCD/BIT = 0 bada, batuketa bitarrez egingo da. Batugaiak 4 bitekoak dira eta emaitza, batura, 8 bitetan eman beharko da.



BCD/Bitar batugailua sortzeko, ohiko batugailua hartuko dugu oinarritzat. Sarrerako datuak 4 bitekoak direnez, 4 biteko batugailua erabiliko dugu.

Batugailu arrunta erabiliz, batuketa bitarrak ez du arazorik sortzen, 3.5. atalean ikusi dugun legez. Kontuan izan beharreko gauza bakarra hau da: emaitza 8 bitetan eman behar dela, baina batugailuaren irteera 4 bitekoa dela. Batura 4 bitetan eman beharko bagenu, batugailuaren C_{out} bitak —baturaren 5. bitak— gainezkatzea adieraziko luke, baina emaitza 8 bitetan eman behar denez, bit hori emaitzari erantsi beharko diogu informaziorik ez galtzeko. Gainerako hiru bitak, 0z beteko ditugu. Hau da, emaitza honela geratuko da: 0 0 0 C_{out} S_3 S_2 S_1 S_0 (ikus 3.30 irudia).

BCDz kodetutako bi digitu batzean, batura 2 BCD digitutan eman behar da (bakoitza 4 bitekoa); pisu txikienekoak batekoak adieraziko ditu, eta pisu handienekoak, berriz, hamarrekoak. Kasu honetan, zoritxarrez, batugailuaren emaitza ezin da erabili besterik gabe BCDz, eta 8 bitera —2 BCD digitutara— egokitu behar da. Azter dezagun adibide pare bat.

		C_{out} / S		BCD		
3 + 2 = 5	→	0011 + 0010 = 0 / 0101	→	0000 0101 **0 5**	→	5
8 + 1 = 9	→	1000 + 0001 = 0 / 1001	→	0000 1001 **0 9**	→	9

Bi adibide horietan, emaitza 10 baino txikiagoa da. Kasu horietan, batugailuaren emaitza egokia da: 4 irteera-bitek (*S*-k) pisu txikieneko BCD digitua adierazten dute, eta bestea 0 izango da.

Azter ditzagun beste bi adibide:

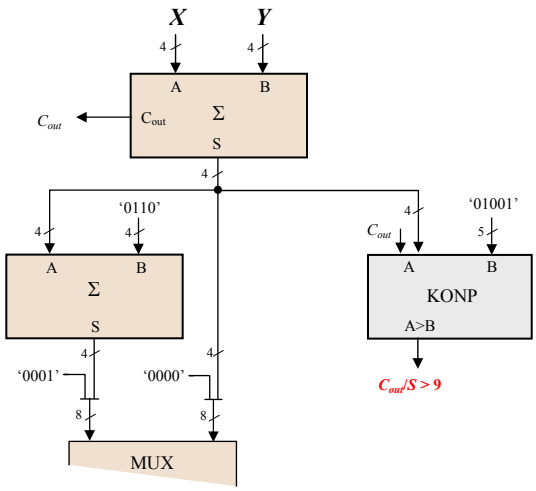
$$\begin{array}{lcll} & & C_{out}\,/\,S & \text{BCD} \\ 5+5=10 & \rightarrow & 0101+0101=0\,/\,1010 & \rightarrow \quad 0001\;0000 \quad \rightarrow \quad 16\,(10+6) \\ & & & \quad\;\mathbf{1}\quad\mathbf{0} \\ 9+9=18 & \rightarrow & 1001+1001=1\,/\,0010 & \rightarrow \quad 0001\;1000 \quad \rightarrow \quad 24\,(18+6) \\ & & & \quad\;\mathbf{1}\quad\mathbf{8} \end{array}$$

Adibide horietan, batuketaren emaitza 9 baino handiagoa da, eta ezin da interpretatu zuzenean BCD kodean. Izan ere, 6ko bat gehitu beharko diogu batugailuaren irteerari, bi BCD digituak eskuratzeko (BCD digitu bat 0tik 9ra doa, 4 bitetan; hori dela eta, 10etik 15era bitarteko 6 kodeak ez dira erabiltzen, eta horregatik batu behar dugu 6koa kode egokia eskuratzeko).

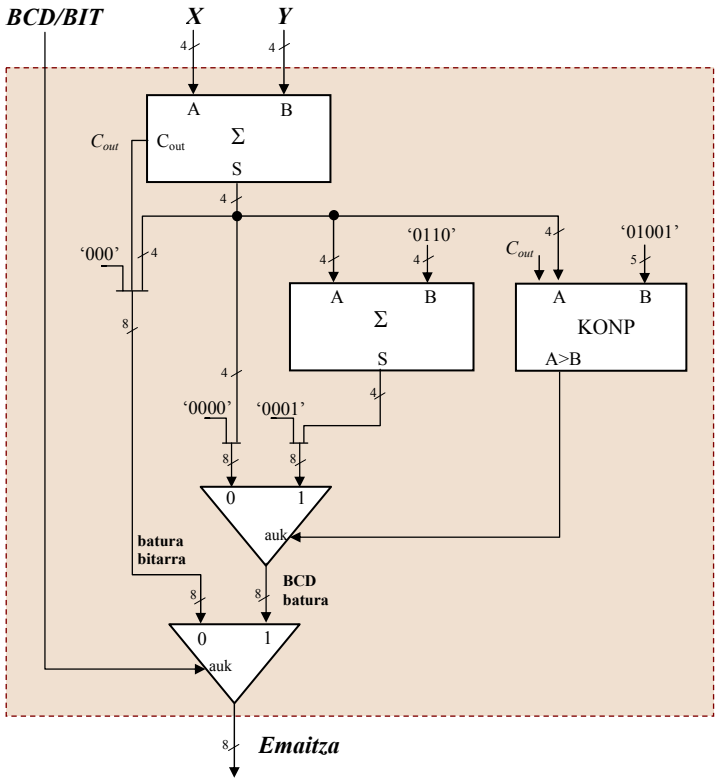
Beraz, honela labur daiteke bi BCD digituren batuketaren algoritmoa:

```
baldin (Cout/S > 9)  orduan    BCD_batura := 0001 / S+6
                     bestela   BCD_batura := 0000 / S
```

Funtzio hori betetzeko, hasierako batugailuaz gain, konparagailu bat erabili beharko dugu, 9rekiko erkaketa egiteko, eta beste batugailu bat $S + 6$ eragiketa egin ahal izateko. Ondorioz, 2 sarrerako multiplexore bat erabili beharko dugu, BCD batura egokia aukeratzeko, konparagailuaren emaitzaren arabera.



Azkenik, beste multiplexore bat erabili beharko dugu batura egokia aukeratzeko: bitarra edo BCD. Sarrerako kontrol-seinalea, *BCD/BIT*, erabiliko dugu hautatze-seinale gisa multiplexore horretan. Hona hemen zirkuitu osoaren eskema logikoa:



3.30. irudia. BCD/Bitar batugailua.

Bi ohar:

(1) ($C_{out}/S > 9$) konparazioa egiteko, 5 biteko konparagailua erabili dugu; nahi izanez gero, 4 biteko konparagailu bat eta OR ate bat ere erabil daitezke, honako erlazio hau betetzen baita: $(C_{out}/S > 9) \equiv (S > 9) + C_{out}$

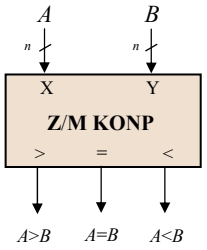
(2) 8 biteko multiplexoreak erabili ditugu aurreko irudian. Izan ere, nahikoa da 5 biteko multiplexoreak erabiltzea, pisu handieneko hiru bitak 0 baitira kasu guztietan. Beraz, nahikoa da hiru 0ko horiek bukaeran gehitzea.

«

3.7. Ariketa

Zenbaki bitar arruntak konparatzen dituen oinarrizko bloke konbinazionala aztertu dugu dagoeneko. Hainbat kasutan, hala ere, beste adierazpideak erabiliz kodetutako zenbakiak konparatu behar dira.

Ariketa honetan, zeinu/magnitudean adierazitako zenbaki osoen konparagailua diseinatuko dugu, oinarrizko bloke konbinazionalak eta behar diren ateak erabiliz. Konparatu beharreko zenbakiak n bitekoak dira.



Zeinu/magnitudea adierazpideko zenbakiak konparatu ahal izateko, adierazpidearen ezaugarriak izango ditugu kontuan, 3.5. ariketan egin dugun bezalaxe. Gogoratu, pisu handieneko bitak zenbakiaren zeinua adierazten du eta gainerakoek magnitudea.

$A = A_{n-1}\, A_{n-2} \ldots A_0 = Z_A\, M_A$ Zeinua: $Z_A = A_{n-1}$ Magnitudea: $M_A = A_{n-2} \ldots A_0$

Zirkuitua egin aurretik, lortu behar diren hiru funtzioen adierazpenak definituko ditugu. Banan-banan aztertuko ditugu:

- *A* zenbakia *B* baino txikiagoa izango da ($A < B$) hiru kasu hauetan:
  - *A* negatiboa izanda, *B* positiboa bada; esaterako, $-3 < 5$ (4 bitetan: $1011 < 0101$);
  - *A* eta *B* negatiboak izanik, *A*-ren magnitudea *B*-rena baino handiagoa bada; adibidez, $-5 < -3$ ($1101 < 1011$); eta
  - *A* eta *B* positiboak izanda, *A*-ren magnitudea *B*-rena baino txikiagoa bada; esate baterako, $3 < 5$ ($0011 < 0101$).
- *A* zenbakia *B* baino handiagoa izango da ($A > B$) hiru kasu hauetan:
  - *A* positiboa izanda, *B* negatiboa bada; esaterako, $3 > -5$ (4 bitetan: $0011 > 1101$);
  - *A* eta *B* negatiboak izanik, *A*-ren magnitudea *B*-rena baino txikiagoa bada; adibidez, $-3 > -5$ ($1011 > 1101$); eta

- *A* eta *B* positiboak izanda, *A*-ren magnitudea *B*-rena baino handiagoa bada; esate baterako, 5 > 3 (0101 > 0011).

▪ *A* zenbakia *B*-ren berdina izango da (*A* = *B*) *A* eta *B* berdinak direnean, bai zeinua eta bai magnitudea.

Analisi horretatik abiatuta, erraza da funtzio logikoen adierazpenak lortzea:

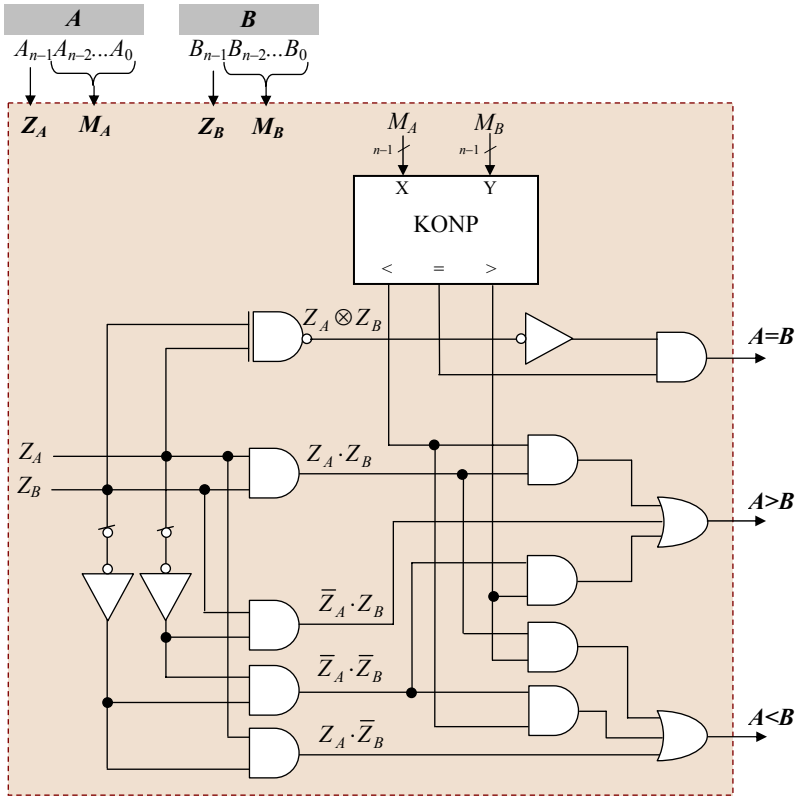
$$(A<B)=Z_A\overline{Z}_B+Z_AZ_B(M_A>M_B)+\overline{Z}_A\overline{Z}_B(M_A<M_B)$$

$$(A>B)=\overline{Z}_AZ_B+Z_AZ_B(M_A<M_B)+\overline{Z}_A\overline{Z}_B(M_A>M_B)$$

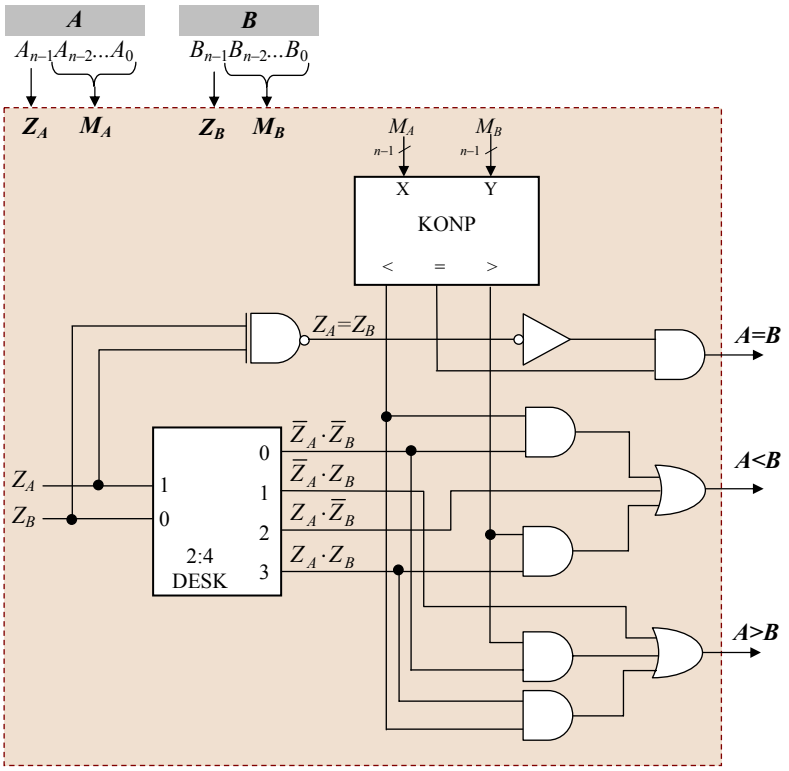
$$(A=B)=(Z_A\otimes Z_B)(M_A=M_B)$$

Funtzio horiek eraikitzeko, oinarrizko ate logikoez gain, konparagailu bat erabili beharko dugu magnitudeak konparatzeko.

Funtzio horiek gauzatuz gero, zirkuituaren diseinu-aukera bat dugu hemen:



Funtzioen adierazpenak edota zirkuitua aztertuz gero, garbi ikusten da zeinuen konbinazio guztiak behar direla: $\overline{Z}_A\overline{Z}_B$, $\overline{Z}_A Z_B$, $Z_A\overline{Z}_B$, eta $Z_A Z_B$. Hori lortzeko, lau AND eta bi NOT ate erabili ditugu aurreko zirkuituan. Badago, hala ere, zirkuitu bat hori egiten duena, deskodegailua hain zuzen ere. Zirkuituaren atal hori deskodegailuaz ordezkatuz gero, honela geratuko litzateke Z/M konparagailua:



Oharra:

Z/M adierazpidea erabiltzen denean, 0 zenbakiak adierazpide bikoitza du: **0**0...0 (+0) eta **1**0...0 (–0). Sinplifikatzearren, ez dugu aukera hori kontuan hartu ariketan. Irakurleak egiazta dezake zer-nolako emaitza emango duen diseinatu dugun zirkuituak bi 0ak konparatzen direnean.

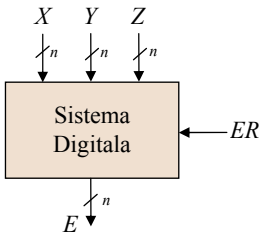
«

3.8. Ariketa

Zirkuitu konbinazional bat diseinatu nahi dugu honako funtzio hau egiteko:

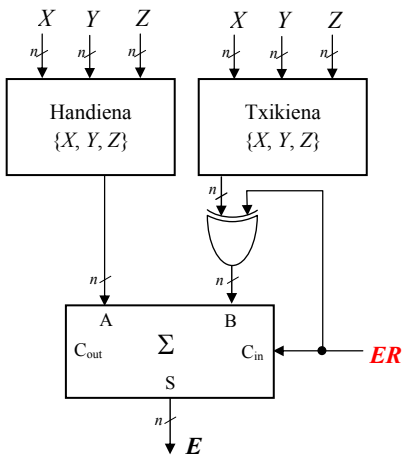
```
baldin (ER = 0) orduan  E := handiena {X,Y,Z} + txikiena {X,Y,Z}
bestela                 E := handiena {X,Y,Z} - txikiena {X,Y,Z}
```

Zirkuituak 3 datu-sarrera izango ditu, —X, Y, eta Z— eta kontrol-seinale bat —ER— egin behar den eragiketa adierazteko. Datu-sarrerak n biteko zenbaki arruntak dira. Lortu behar den emaitza ere n bitekoa izango da. Ez dugu kontuan izango gainezkatzea.



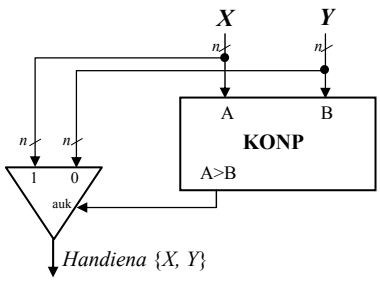
Lehendabizi, hiru zenbakien arteko handiena eta txikiena kalkulatzeko zirkuituak sortu beharko ditugu. Hori lortuta, batuketa edo kenketa bat egin beharko dugu, *ER* kontrol-seinalearen arabera: *ER* = 0 bada, batuketa; bestela, *ER* = 1 bada, kenketa.

Batugailua/kengailuaren A sarreran beti *Handiena* {*X, Y, Z*} behar da; B sarreran, aldiz, batuketaren kasuan, *Txikiena* {*X, Y, Z*}, eta, kenketaren kasuan, bere osagarria (ikus 3.5.3. atala). Hala, honelakoa izango da zirkuituaren eskema logiko orokorra:

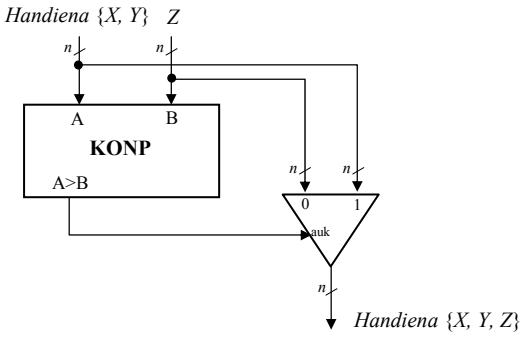


Has gaitezen hiru zenbakien arteko handiena lortzen duen zirkuitua diseinatzen. Horretarako, lehendabizi bi zenbakiren arteko handiena lortuko dugu eta, gero, hirugarren zenbakiarekin konparatuz, hiruren arteko handiena.

Bi zenbaki arrunten arteko handiena lortzeko, lehendabizi zenbaki horiek konparatuko ditugu, eta gero, konparaketaren arabera, bietatik handiena aukeratu. Horretarako, bi zirkuitu behar ditugu: konparagailua eta multiplexorea. Multiplexorean hautaketa egiteko, konparagailuaren ($A > B$) irteera erabiliko dugu. ($A > B$) = 1 denean, X aukeratuko dugu, eta, bestela, Y. Adi! konparagailuak ez digu bi zenbakien arteko handiena edo txikiena ematen, haien arteko erlazioa baizik.



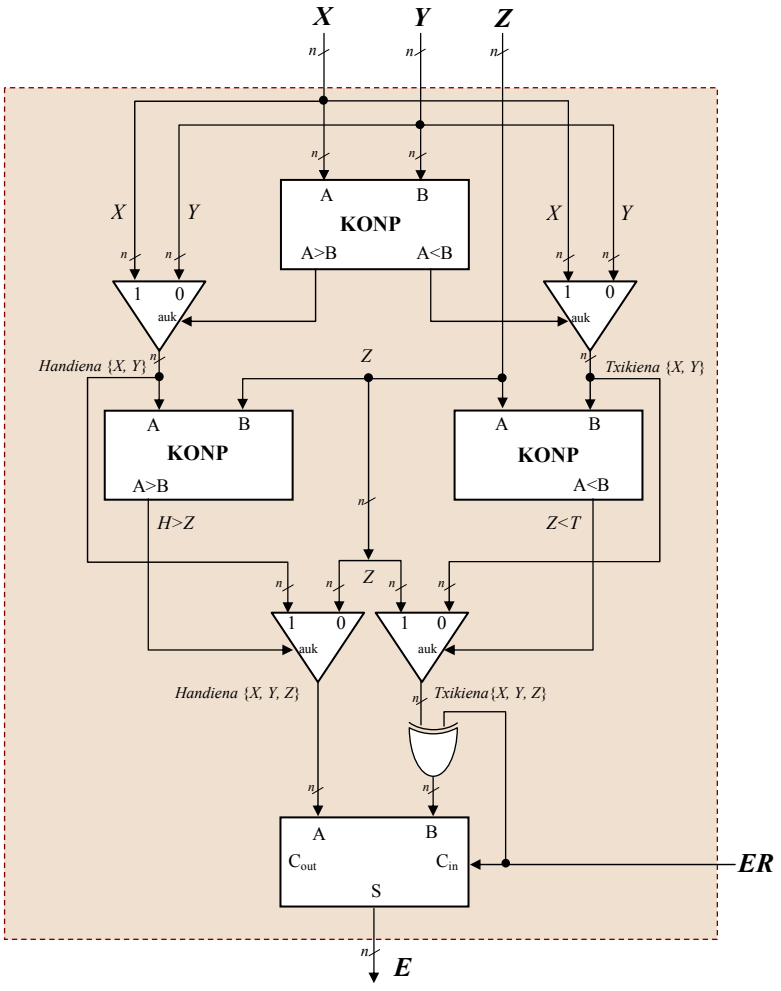
Hirugarren zenbakia konparatzeko, beste konparagailu bat eta beste multiplexore bat beharko ditugu. Haien sarrerak honako hauek izango dira: batetik, X-ren eta Y-ren arteko handiena (*Handiena* {X, Y}), eta, bestetik, Z. Multiplexorean hautaketa egiteko, bigarren konparagailuaren ($A > B$) irteera erabiliko dugu.



Hiru zenbakien arteko txikiena lortu behar dugu orain. Prozedura bera izango da. X-ren eta Y-ren arteko konparaketa egiteko, hasieran erabili dugun

konparagailua aprobetxatuko dugu, baina beste multiplexore bat sartuko dugu zenbaki txikiena aukeratzeko (*Txikiena* {*X, Y*}). Aukeraketa egiteko, konparagailuaren ($A < B$) irteera erabiliko dugu. Azkenik, *Txikiena* {*X, Y*} eta *Z*-ren arteko txikiena zein den jakiteko, hirugarren konparagailu bat beharko dugu. Beste multiplexore batek emango digu, finean, *X, Y* eta *Z*-ren arteko txikiena.

Hona hemen funtzioa gauzatzeko zirkuitu osoaren aukera bat:



3.9. Ariketa

Zirkuitu konbinazional bat eraiki nahi da honako eragiketa hau egiteko:

```
baldin (X > Y)  orduan    E := X + Z/2
                bestela   E := Y - Z×2
```

Aztertu behar da zirkuituaren diseinua hiru adierazpide hauetarako:

(a) Datuak zenbaki arruntak dira, bitarrez adierazita.

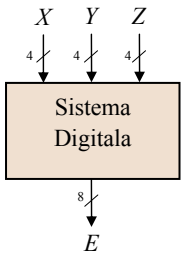
(b) Datuak zenbaki osoak dira, 2rako osagarrian adierazita.

(c) Datuak zenbaki osoak dira, zeinu/magnitudean adierazita.

Sarrerako datuak —X, Y eta Z— 4 bitekoak dira, eta emaitza 8 bitetan eman behar da.

Adierazpide bakoitzeko batugailua zein konparagailua badago:

- *zenbaki arruntetarako, ohiko batugailua eta konparagailua erabiliko ditugu (3.5. eta 3.6. ataletakoak),*
- *2rako osagarria adierazpiderako, batugailu arruntak ere balio digu (3.5. atalekoak) eta konparagailua, irakurleari egitea proposatzen zaio ariketa gisa (3.7. ariketako zeinu/magnitudekoaren ildo beretik).*
- *zeinu/magnituderako batugailua 3.5. ariketan definitutakoa izango da, eta konparagailua, 3.7. ariketan azaldu duguna.*



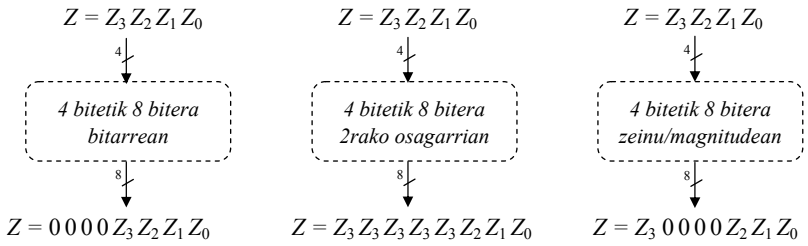
Ariketa egiten hasi aurretik, bi gauza azalduko ditugu; batetik, nola adierazi 4 biteko zenbaki bat 8 bitetan, eta, bestetik, nola egin eskatzen diren bi eragiketak —$Z \times 2$ (biderketa) eta $Z / 2$ (zatiketa)— zenbakien adierazpidea kontuan hartuz.

Bit kopurua zabaltzea

Zenbaki bat n bitetik $n + k$ bitera zabaltzeko (adib., 4 bitetik 8ra), adierazpide-sistema kontuan hartu behar da:

- Zenbaki arrunten kasuan, 4 0ko (oro har, k) gehituko behar zaizkio jatorrizko zenbakiari ezkerraldetik; lau bitetan $Z = Z_3\,Z_2\,Z_1\,Z_0$ bada, 8 bitetan $Z = 0\,0\,0\,0\,Z_3\,Z_2\,Z_1\,Z_0$ izango da.
- Birako osagarria adierazpidean, zenbakiaren bit kopurua zabaltzeko, lau (k) bitak zeinu-bitez beteko ditugu. $Z = Z_3\,Z_2\,Z_1\,Z_0$ bada, zenbaki bera 8 bitetan $Z = Z_3\,Z_3\,Z_3\,Z_3\,Z_3\,Z_2\,Z_1\,Z_0$ izango da.
- Zeinu/magnitudea adierazpidean, zeinu-bita ezkerraldeko posiziora eraman behar da, eta magnitudea zabaldu, 0z. Adibidez, $Z = Z_3\,Z_2\,Z_1\,Z_0$ bada, zenbaki bera 8 bitetan: $Z = Z_3\,0\,0\,0\,0\,Z_2\,Z_1\,Z_0$ izango da.

Beraz, zenbaki bat 4 bitetik 8 bitera pasatzeko ez da zirkuiturik behar, nahikoa da zeinu-bita kontrolatzea eta 0koak jartzea posizio jakin batzuetan:



Biderketa eta zatiketa

Liburu honetan, ez dugu zirkuiturik azalduko biderketak eta zatiketak egiteko, eragiketa horiek konplexuak baitira, eta gure helburuetatik at geratzen dira. Hala ere, biderkatzailea edo zatitzailea 2ren berretura denean, eragiketa horiek era sinplean egin daitezke desplazamenduen bidez, zirkuiturik erabili gabe.

$Z \times 2$ eragiketa egiteko, Z zenbakia bit bat ezkerrerantz desplazatu behar da; Z / 2 egiteko, berriz, bit bat eskuinerantz. Bi kasuetan, Z zenbakiaren bit bat galtzen da, ezkerrekoa edo eskuinekoa. Adi! desplazamenduak egiteko, zenbakiaren zeinua izan behar dugu kontuan, adierazpidearen arabera.

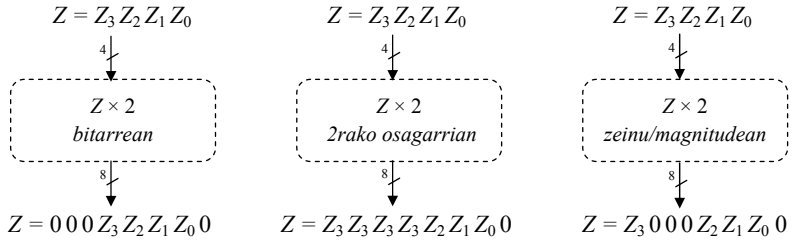
Biderketa egitean, gainezkatzea (*overflow*) gerta daiteke, emaitza ez delako sartzen daukagun bit kopuruan. Zatiketa egitean, eskuinaldetik bitak galtzen direnez, zatiduraren alde osoa lortzen da (esaterako, 5/2 egiteko, 0101 kodea bit bat desplazatu behar dugu eskuinerantz, beraz, 0010 lortuko dugu, 2 alegia, eta ez 2,5). Hala ere, ariketa honetan, gainezkatzearen arazoa ekidin daiteke, lehendabizi 8 bitera zabaltzen badugu zenbakia eta, gero, biderketa egiten badugu, eta ez alderantziz.

Zenbaki arruntekin, desplazamenduak egitean gelditzen diren hutsuneak 0z bete behar dira. $Z \times 2$ egiteko, bit bat ezkerrerantz desplazatzean, eskuinean jarriko dugu 0koa; $Z / 2$ egiteko, berriz, bit bat eskuinerantz desplazatzean, ezkerrean. Esaterako, $Z = Z_3\, Z_2\, Z_1\, Z_0$ bada, $Z \times 2 = Z_2\, Z_1\, Z_0\, 0$ izango da, eta $Z / 2 = 0\, Z_3\, Z_2\, Z_1$.

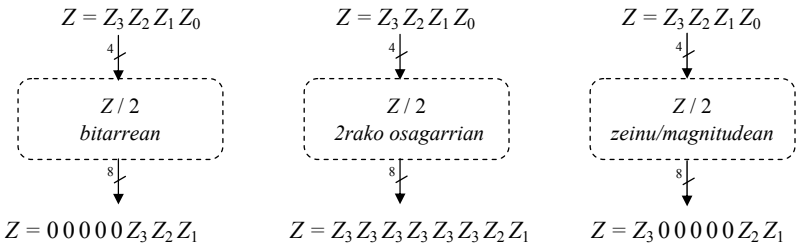
Zenbaki osoen kasuan, 2rako osagarrian adierazita badaude, modu desberdinean tratatzen dira ezkerreko eta eskuineko hutsuneak; ezkerrerantz desplazatzean, eskuinean 0koa sartuko dugu, zeinu-bita mantenduz, baina, desplazamendua eskuinerantz bada, ezkerrean pisu handieneko bita sartuko da, hau da, zeinu-bita. Adibidez, $Z = Z_3\, Z_2\, Z_1\, Z_0$ bada, $Z \times 2 = Z_2\, Z_1\, Z_0\, 0$ izango da, eta $Z / 2 = Z_3\, Z_3\, Z_2\, Z_1$.

Zeinu/magnitudea adierazpidean, zeinu-bita bere tokian mantendu behar da, eta magnitudea bakarrik desplazatu. Sortzen diren hutsuneak 0z bete beharko ditugu. Esaterako, $Z = Z_3\, Z_2\, Z_1\, Z_0$ bada, $Z \times 2 = Z_3\, Z_1\, Z_0\, 0$ izango da, eta $Z / 2 = Z_3\, 0\, Z_2\, Z_1$.

Ariketan, Z zenbakia moldatu beharko dugu biderketa eta zatiketa eginez eta emaitza 8 bitetan emanez. Hona hemen nola gelditzen den Z zenbakia 8 bitetan, biderketa egin eta gero:



Zatiketa egitean:

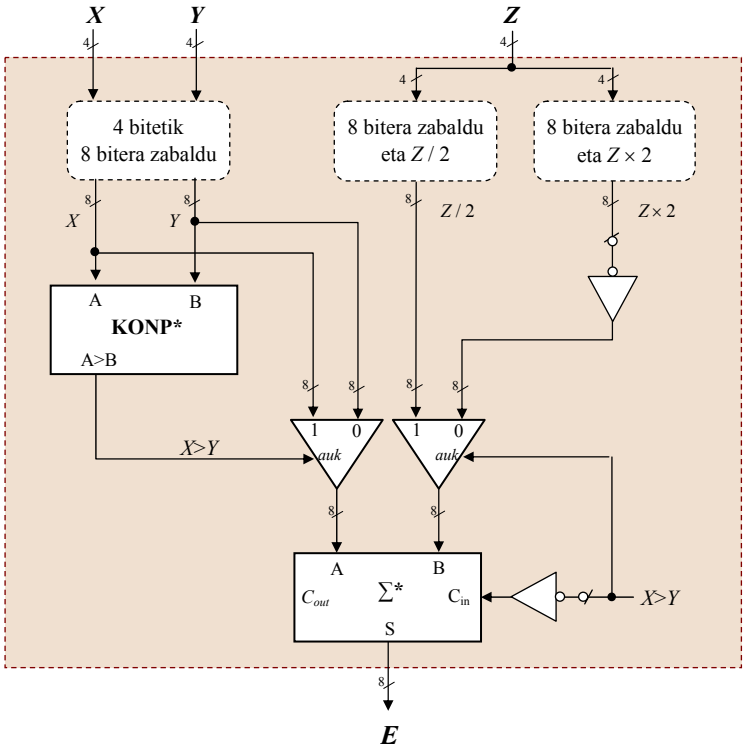


Has gaitezen zirkuitua diseinatzen. Adierazpide bakoitzerako zirkuitu bat egin beharko dugu; dena den, zirkuituaren oinarria bera izango da hiru kasuetan. Batetik bestera aldatuko dena, batugailua, konparagailua eta

zenbakien adierazpenak izango dira. Kasu bakoitzean erabili behar diren batugailuak eta konparagailuak ariketaren hasieran definitu ditugu, eta adierazpide bakoitzeko, 8 bitera zabaltzea eta biderketak zein zatiketak nola egin ere azaldu dugu.

Beraz, zirkuitu bakarrean laburbilduko ditugu hiru adierazpideetako eskemak. Lehendabizi, *X* eta *Y* sarrerako datuak, 8 bitera zabalduko ditugu eta *Z* datua, 8 bitera zabaltzeaz gain, eskuinera zein ezkerrera desplazatuko dugu. Adi! eragiketa horiek egiteko ez da zirkuiturik erabili behar. Bestalde, batugailuaren sarrera batean, *X* edo *Y* sartu beharko dugu, eta, bestean, $Z / 2$ edo $Z \times 2$. Beraz, multiplexore bana beharko dugu batugailuaren sarreretan aukera egokia egiteko. Datuen eta eragiketaren aukera *X*-ren eta *Y*-ren arteko erlazioaren arabera egingo da, eta, horretarako, konparagailua behar dugu.

Hauxe izango da zirkuituaren eskema orokorra. KONP* eta ∑* izenek adierazpide desberdinetako zirkuituak (konparagailuak zein batugailuak) adierazten dituzte.

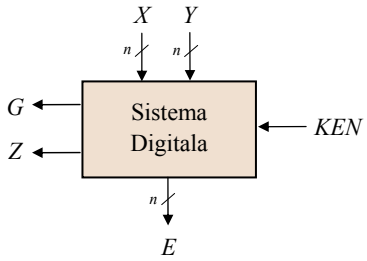


3.10. Ariketa

Zirkuitu konbinazional bat diseinatu nahi da honako eragiketa hau egiteko:

```
baldin (KEN = 0)              orduan E := X + Y
baldin (KEN = 1) eta (X > Y)  orduan E := X - Y
baldin (KEN = 1) eta (X ≤ Y)  orduan E := Y - X
```

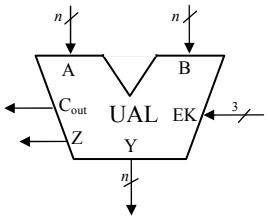
Zirkuituak sarrerako bi datu izango ditu —X eta Y—, eta kontrol-seinale bat —KEN— egin behar den eragiketa adierazteko. E emaitzaz gain, sistemak G eta Z seinaleak ere sortu beharko ditu. G aktibatuko da gainezkatzea (overflow) gertatzean, eta Z aktibatuko da emaitza 0 denean. Sarrerako datuak —X eta Y— eta emaitza —E— n biteko zenbaki arruntak dira.



Sistema bi eratara diseinatuko dugu:

(a) Bloke konbinazionalak (UALik erabili gabe) eta behar diren ateak erabiliz.

(b) UAL jakin bat, eta behar diren bestelako blokeak eta ateak erabiliz. Hona hemen UAL horren egitura eta portaera:



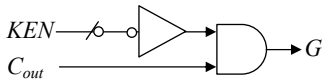
EK	***Eragiketa (Y)***
000	*Clear (0000)*
001	*A – B*
010	*B – A*
011	*A + B*
100	*A xor B*
101	*A or B*
110	*A and B*
111	*Preset (1111)*

C_{out} irteerak gainezkatzea adieraziko du eragiketa aritmetikoetan, eta Z = 1 izango da emaitza zero denean, hau da, bit guztiak 0 direnean.

■

Lehendabizi, gainezkatzearen detekzioa aztertuko dugu, berdin egiten baita (a) eta (b) kasuetan. *Overflow*-a gertatuko da bi zenbaki batzean emaitza ez bada *n* bitetan sartzen; beraz, batuketaren kasuan, nahikoa da C_{out} aztertzea. Kenketarekin, ordea, ez da inoiz gainezkatzerik gertatuko: handienari txikiena kentzen diogu, eta, ondorioz, emaitza *n* bitetan sartuko da. Hala ere, kenketak egitean, C_{out} irteera aktiba daiteke (esaterako, $8 - 6 = 2 \rightarrow C_{out}/S = 1/0010$) .

Ondorioz, *G* irteera aktibatuko da bakarrik batuketa egitean $C_{out} = 1$ bada, baina inoiz ez kenketaren kasuan; hau da, $G = \overline{KEN} \cdot C_{out}$



(a) Bloke konbinazionalak erabiliz

Zirkuitua egiteko hainbat bide eta aukera dago; edozein kasutan, egin behar den funtzioa betetzeko, batugailu/kengailu bat beharko dugu, non, kasu bakoitzean, sarrera egokiak aukeratu behar diren. Hautaketak egiteko, *KEN* kanpo-seinalea eta *X*-ren eta *Y*-ren arteko erlazioa —konparagailu batek emango diguna— izan behar ditugu kontuan.

Taula honetan, laburbilduz, batugailuaren sarreretan sartu behar diren datuak adierazten dira.

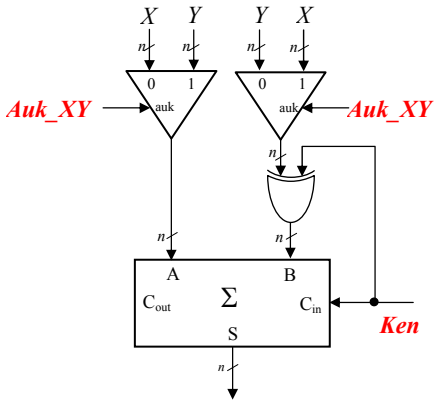
KEN	*X* > *Y*	Eragiketa	*A*	*B*	C_{in}
0	0	*X* + *Y edo* *Y* + *X*	*Y*	*X*	0
0	1	*X* + *Y edo* *Y* + *X*	*X*	*Y*	0
1	0	*Y* – *X*	*Y*	$\overline{X}$	1
1	1	*X* – *Y*	*X*	$\overline{Y}$	1

Batugailuaren sarreretan, multiplexoreak izango ditugu *X*-ren eta *Y*-ren arteko hautaketak egiteko. C_{in} sarreran, 0koa sartuko dugu batuketa egitean eta 1ekoa kenketarako.

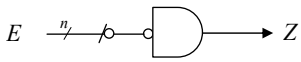
Hurrengo irudian, batuketak eta kenketak egiteko zirkuitua ageri da.

Dagoeneko ikusi dugun moduan, XOR ateak erabiltzen ditugu *X* edo $\overline{X}$ eta *Y* edo $\overline{Y}$ lortzeko.

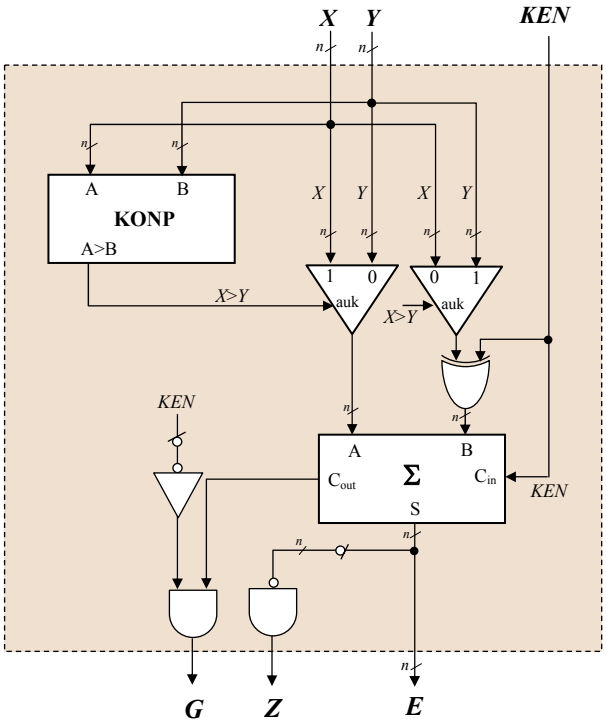
Multiplexoreen kontrola (*Auk_XY*) *X*-ren eta *Y*-ren arteko erlazioak zehaztuko du. Horretarako, sarrerako zenbakiak konparatu eta konparagailuaren $A > B$ irteera erabiliko dugu.



Bestalde, Z seinalea aktibatu behar da emaitza 0 denean, hau da, baturaren n bitak 0 direnean ($E = 0 \ldots 00$). Beraz, $Z = \overline{E_{n-1}}\ \overline{E_{n-2}} \ldots \overline{E_1}\ \overline{E_0}$



Hona hemen zirkuituaren aukera bat:



(b) UAL bat erabiliz

Proposatzen diguten UALak exekuta ditzake behar ditugun hiru eragiketak: $A + B$, $A - B$ eta $B - A$, eta, horretarako 011, 001 eta 010 kodeak erabili behar ditugu, hurrenez hurren.

Beraz, UALak behar dituen 3 biteko eragiketa-kodeak sortu behar ditugu, *KEN* seinalearen eta *X*-ren eta *Y*-ren arteko erlazioaren arabera, taula honetan ageri den moduan:

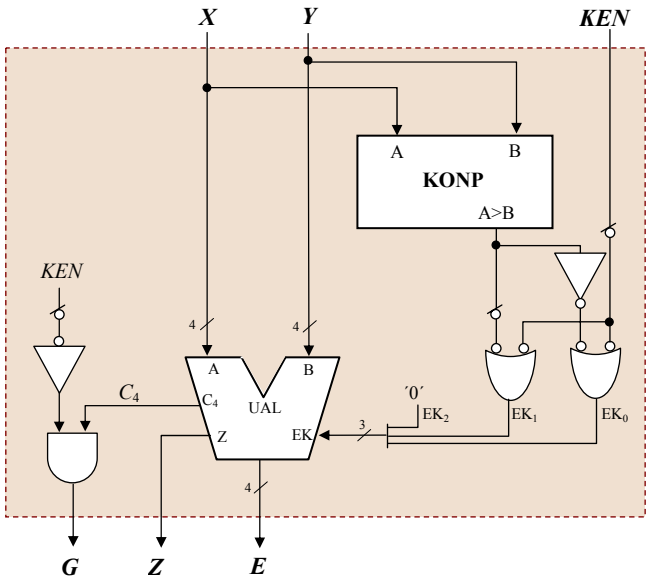
KEN	$X > Y$	Eragiketa	*EK*
0	–	$X + Y$	011
1	0	$Y - X$	010
1	1	$X - Y$	001

Taula horretatik erraz lor daitezke eragiketa-kodearen bit bakoitzaren ekuazio logikoa:

$$EK_2 = 0$$
$$EK_1 = \overline{KEN} + \overline{(X > Y)}$$
$$EK_0 = \overline{KEN} + (X > Y)$$

Bestalde, *Z* seinalea aktibatu behar da emaitza —*E*— zero denean. UALak berak ematen digu irteera gisa *Z* seinalea; beraz, zuzenean bertatik hartuko dugu.

Hona hemen zirkuituaren aukera bat:



3.9. ARIKETAK

3.1. Eraiki itzazu bi funtzio hauek 8 datu-sarrerako multiplexoreak erabiliz:

a) $f(d,c,b,a) = \sum (0,1,4,5,6,7,10,12,13,14,15)$

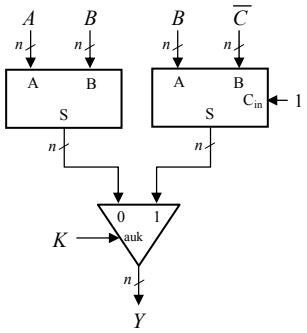
b) $f(d,c,b,a) = c + db + dba$

Egin ezazu gauza bera 4 datu-sarrerako multiplexoreak eta behar dituzun ateak erabiliz.

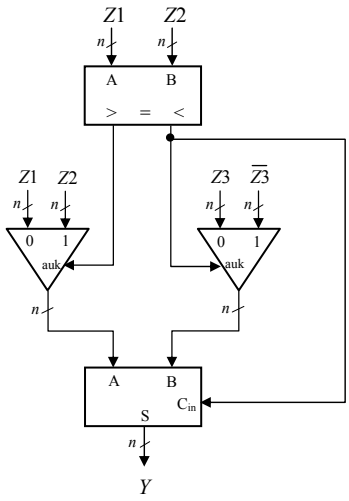
3.2. 2rako osagarrian adierazitako *n* zenbaki osoen konparagailua diseinatu behar da. Sistemak bi datu-sarrera izango ditu —*X* eta *Y*—, eta konparagailuen ohiko hiru irteerak —*X*>*Y*, *X*=*Y* eta *X*<*Y*—. Zirkuitua diseinatzeko, erabil itzazu ohiko bloke konbinazionalak eta ate logikoak.

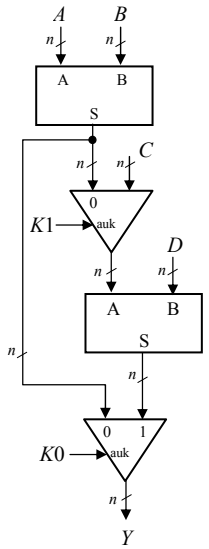
3.3. 2:4 deskodegailuak erabiliz, egin ezazu 4:16 deskodegailu bat.

3.4. Zer funtzio egiten du irudiko zirkuituak? Diseina ezazu funtzio bera egiten duen zirkuitua baina batugailu bakar bat erabiliz.



3.5. Zer funtzio egiten dute irudiko zirkuituek? Datuak *n* biteko zenbaki arruntak dira.





3.6. *A*, *B* eta *C*, *n* biteko hiru zenbaki arrunt prozesatzen ditu zirkuitu konbinazional batek, honako funtzio hau egiteko:

```
baldin (A > B) orduan Y := handiena {A, (B + C)}
baldin (A < B) orduan Y := handiena {B, (A + C)}
baldin (A = B) orduan Y := handiena {A, C}
```

Irteera, *Y*, *n*+1 bitekoa da. Diseina ezazu zirkuitu hori ohiko bloke konbinazionalak eta ate logikoak erabiliz.

3.7. Zirkuitu konbinazional batek *A* eta *B* datu-sarrerak (*n* biteko zenbaki arruntak) eta *S* kontrol-sarrera prozesatzen ditu, honako funtzio hau egiteko:

```
S = 0 denean
  baldin (A = B) orduan           Y := A
  bestela baldin (A > B) orduan   Y := A - B
  bestela                         Y := B - A

S = 1 denean
  baldin (A = B) orduan           Y := A + B
  bestela baldin (A > B) orduan   Y := not_B
  bestela                         Y := not_A
```

Emaitza, *Y*, *n* bitekoa da. Horrez gain, balizko gainezkatzea adierazi behar du zirkuituak. Diseina ezazu funtzio hori egiten duen zirkuitua, bloke konbinazionalak zein ate logikoak erabiliz.

3.8. Zirkuitu konbinazional batek zeinu/magnitudeko 4 biteko zenbaki bat, *A,* prozesatzen du, bi biteko kontrol-kode baten arabera, K_1K_0, honako funtzio hau egiteko:

Kodea (K_1 K_0)	Emaitza (E)
00	|A|
01	$2A$
10	$3A$
11	$-A$ / 2

Emaitza, *E*, 8 bitekoa da. Diseinatu zirkuitu hori bloke konbinazionalak eta ate logikoak erabiliz.

3.9. *n* biteko *A* eta *B* zenbaki osoak (2rako osagarria) prozesatu behar dira, honako funtzio konbinazionala egiteko:

```
baldin (A>0) eta (B>0) orduan          Y := A + B
bestela baldin (A<0) eta (B<0) orduan  Y := handiena{A,B}
bestela                                Y := |A| + |B|
```

Diseinatu funtzio horri dagokion zirkuitua bloke konbinazionalak erabiliz.

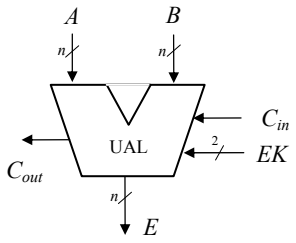
3.10. Zirkuitu konbinazional batek *n* biteko bi zenbaki arrunt prozesatzen ditu, *KS* kontrol-seinalearen arabera, honako funtzio hau egiteko:

```
baldin (KS = 1) orduan
     baldin (A ≤ B) orduan     Y := 2A + 1
     bestela                   Y := 2B + 1
bestela
     baldin (A > B) orduan     Y := A − B
     bestela                   Y := B − A
```

Emaitza, *Y*, *n* bitekoa da, eta balizko gainezkatzea sortu behar da.
Diseina ezazu zirkuitua

(a) bloke konbinazionalak erabiliz.

(b) honako UAL hau eta behar diren blokeak erabiliz:



Eragiketa-kodea EK_1 EK_0	Emaitza
0 0	A
0 1	$A - B - C_{in}$
1 0	$A + B + C_{in}$
1 1	0

3.11. Sistema digital jakin batean, 32 biteko 8 zenbaki (*Z*1 ... *Z*8) batu behar dira. Diseinatu zirkuitu konbinazional bat, ahalik eta eraginkorrena, zenbaki horiek batzeko.
Batugailu baten erantzun-denbora 2*n* ns bada (*n* = zenbakien bit kopurua), zenbat denbora behar da 8 zenbakiak batzeko?
Nola detekta daiteke eragiketaren gainezkatzea (batugailuak 32 bitekoak dira)?

4. kapitulua

BLOKE SEKUENTZIALAK

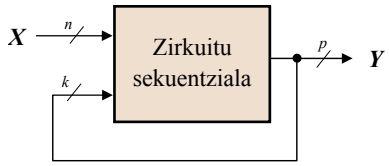
Aurreko kapituluan, bloke konbinazional behinenak aztertu ditugu: multiplexoreak, konparagailuak, deskodegailuak, batugailuak, unitate aritmetiko/logikoak, eta abar. Bloke horiek erabiliz, zirkuitu konbinazional konplexuagoak eraiki daitezke. Kapitulu honetan, beste zirkuitu mota bat aztertuko dugu: zirkuitu sekuentzialak. Lehen zatian, oinarrizko osagaiak azalduko ditugu: JK eta D biegonkorrak; bigarrenean, erregistroak: erregistro soilak, desplazamendu-erregistroak eta kontagailuak.

4.1. ZIRKUITU SEKUENTZIALAK

Aurreko kapituluetan aztertu ditugun zirkuitu guztiak —ate logikoak, multiplexoreak, batugailuak...— zirkuitu konbinazionalak dira. Esan dugun bezala, zirkuitu konbinazional baten erantzuna soilik uneko sarreren mende dago. Hau da, ez dira kontuan hartzen aurreko erantzunak: sistemak ez du memoriarik. Esaterako, batugaiak 2 eta 3 badira, batugailuaren emaitza beti 5 izango da.

Sistema digital gehienetan, ordea, erantzunak ez du izan behar bakarrik uneko sarreren funtzioa: **sistemaren egoera** ere hartu behar da kontuan erantzun egokia emateko. Hala, sarrera jakin batzuetarako sistemak emango duen erantzuna ez da beti bera izango. Adibide sinple asko eman daitezke. Esaterako, demagun semaforo bat eraiki nahi dugula. Semaforoak kolore-sekuentzia jakin bati jarraitzen dio: berdea, anbarra, gorria, berdea... Beraz, aldaketa bat gertatzen denean, semaforoaren hurrengo kolorea uneko kolorearen araberakoa da. Gauza bera gertatzen da, adibidez, kontagailu batean (geroago aztertuko ditugu kontagailuak). Kontatzeko agindua hartzen duenean, 1 gehituko dio gordeta duen balioari; hau da, 3tik 4ra igaroko da, edo 6tik 7ra. Emaitza (irteera), ondorioz, desberdina da, kontagailuaren uneko balioaren araberakoa, hain zuzen ere.

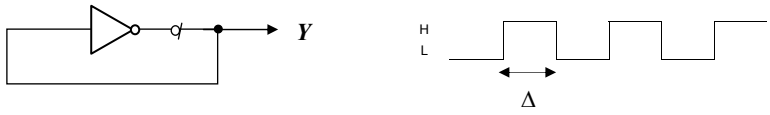
Zirkuitu sekuentzial baten erantzuna emateko, beraz, sarreren balioez gain, sistemaren egoera ere prozesatu behar da. 4.1. irudian, zirkuitu sekuentzial baten egitura orokorra ageri da.



4.1. irudia. Zirkuitu sekuentzial baten eskema orokorra.

Irudian ageri denez, zirkuituaren irteera batzuk **berrelikatu egiten** dira sarreran. Izan ere, zirkuitu sekuentzialen memoria (aurreko balioak kontuan hartzeko ahalmena) berrelikaduraren ondorioa da.

Hainbat zirkuitu diseina daitezke irteerak sarreran berrelikatuz. Askok ez dute portaera berezirik erakusten. Batzuek, aldiz, bai. Esaterako, 4.2. irudian, adibide bat ageri da.



4.2. irudia. Osziladore baten oinarrizko eskema.

4.2. irudiko NOT atearen irteera sarreran konektatuta dago. Beraz, irteera 0 denean, sarreran 0koa izango dugu, eta, ondorioz, irteera 1 izatera igaroko da. Gero, sarreran 1 izango dugunez, irteera 0 izango da eta abar: zirkuituaren irteerak oszilatu egiten du. Oszilazio horren periodoa atearen erantzun-denboraren araberakoa da, Δ. (Adibide soil bat da aurrekoa, ez baitira horrela egiten oszilatzen duten zirkuituak, funtsean antzeko egiturak erabiltzen badira ere.)

4.1.1. Zirkuitu sekuentzialen sailkapena

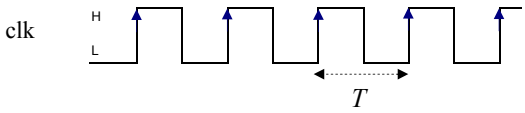
Oro har, bi motatako zirkuitu sekuentzialak ditugu:

- **Asinkronoak:** zirkuituak une oro prozesatzen ditu sarrera-seinaleak, dagokien irteera sortzeko.
- **Sinkronoak:** kontrol-seinale batek, **erlojuak,** adierazten dio zirkuituari noiz prozesatu behar diren sarrerak irteera berriak sortzeko. Bitartean, zirkuituaren erantzuna ez da aldatuko, sarreren balioak aldatu arren.

Zirkuitu sekuentzial sinkronoak diseinatzea errazagoa da, haien funtzionamendua erloju-seinalearen bidez kontrola baitaiteke. Seinaleak ez dira prozesatzen edozein unetan, baizik eta erloju-seinaleak adierazten duenean, eskuarki seinaleen balioak egonkorrak direnean; hala, sarreretan egon daitezkeen "zaratek" edo *glitch*-ek —seinaleen balio iragankorrek, zirkuituan zehar aldaketak gertatzen ari direlako— ez dute eraginik izango sistemaren erantzunean. Liburu honetan zirkuitu sekuentzial sinkronoak bakarrik aztertuko ditugu.

Aipatu dugun moduan, kontrol-seinale berezi bat erabiltzen da zirkuitu sekuentzial sinkronoetan: erlojua (ingelesez, *clock*, eta, laburtuz, clk). Izan ere, erloju-seinalea sistema sinkrono baten kontrol-seinale nagusia da, berak adierazten baitu noiz prozesatu behar diren sarrerak irteerak kalkulatzeko, eta, ondorioz, neurri batean behintzat, sistemaren "abiadura" zehazten du.

Erloju-seinalea 1 eta 0 balioen artean etenik gabe oszilatzen duen seinalea da, 4.3. irudikoa bezala.



4.3. irudia. Erloju-seinalea. Goranzko ertzak markatuta daude, gezi batez.

Oro har, erloju-seinalearen balio-aldaketek edo ertzek adierazten dute noiz prozesatu zirkuituaren sarrerak[8]. Bi aukera ditugu, beraz: erloju-seinalea 0tik 1era igarotzen den unea —L-tik H-ra, goranzko ertza— edo 1etik 0ra aldatzen denean —H-tik L-ra, beheranzko ertza—. Bi aukerak berdintsuak dira, eta, sinplifikatzeko asmoz, bakar bat erabiliko dugu: erloju-seinalearen **goranzko ertzean sinkronizatzen diren zirkuituak.**

Erloju-seinalea, definizioz, behin eta berriz errepikatzen den seinale periodiko bat da. Hori dela eta, seinale horren **periodoa** —*T*— eta **maiztasuna** —*f*— defini daitezke. Gogoratu: seinalea errepikatzeko behar den denbora-tartea adierazten du periodoak, segundotan; maiztasuna, periodoaren alderantzizkoa da, eta hertzetan (s^{-1}) neurtzen da.

Esaterako,

$T = 1$ μs (mikrosegundo, 10^{-6} s) → $f = 1/T = 10^6$ Hz = 1 MHz (megahertz)

$T = 1$ ns (nanosegundo, 10^{-9} s) → $f = 1/T = 10^9$ Hz = 1 GHz (gigahertz)

Azter ditzagun, dagoeneko, zirkuitu sekuentzial sinkronoen oinarrizko osagaiak.

4.2. BIEGONKORRAK

Biegonkorrak (ingelesez, *flip-flop*) oinarrizko osagaiak dira edozein zirkuitu sekuentzialetan. **Bit bateko memoria** duten gailuak dira biegonkorrak, eta, beraz, bi egoeretako batean egon daitezke: 1 edo 0 (hortik izena, bi egoera egonkor dituztelako).

---

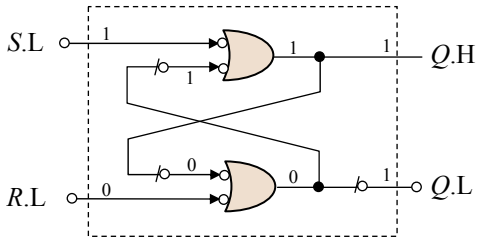
[8] Zenbait zirkuitutan, une jakin bat erabili beharrean, erloju-seinalearen tarte bat erabiltzen da aldaketak onartzeko; esaterako, erloju-seinaleak 1 balio duen tartea (edo 0koa). Liburu honetan ez ditugu zirkuitu horiek erabiliko.

Gehiago badaude ere, bi dira *flip-flop* nagusiak: JK biegonkorra eta D biegonkorra. Biak zirkuitu sinkronoak dira; hau da, erloju-seinale batek kontrolatzen du noiz gertatuko diren 1 eta 0 egoeren arteko trantsizioak.

JK eta D biegonkorren funtzionamendua azaldu baino lehen, merezi du aztertzea nola lor daitekeen memoria-propietatea ate logiko pare batetik abiatuta. Izan ere, JK eta D biegonkor sinkronoen barruan, oinarrizko egitura bera dago: SR biegonkor asinkronoa. Ikus dezagun zertan den SR biegonkorra.

4.2.1. Bit bateko memoria duen oinarrizko osagaia: SR biegonkor asinkronoa

Kapitulu honen hasieran aipatu dugun bezala, zirkuitu sekuentzialen ezaugarri nagusia berrelikadura da. 4.4. irudian, bi NAND ateen bidez egindako zirkuitu bat ageri da, SR biegonkorra.



4.4. irudia. NAND ateekin egindako SR biegonkorraren eskema (antzeko egitura bat osa daiteke NOR ateekin).

SR biegonkorrak bi sarrera —*S, set,* eta *R, reset*— eta bi irteera —*Q*.H eta *Q*.L— ditu. Izan ere, egiaztatuko dugun moduan, bi irteerek seinale bera adierazten dute murriztapen bat onartzen badugu, eta, horregatik, izen bera erabiliko dugu: *Q*. Irudian ageri denez, zirkuitua berrelikatu dugu, bi irteerak ateen sarreretara eramanez. Azter dezagun zirkuitu horren funtzionamendua.

S (*set*) sarrera aktibatzen denean —*S* = 1 eta *R* = 0—, *Q*.H irteerak 1 balioa hartzen du (*or* funtzioaren emaitza). *Q*.H irteeraren balio berria dela eta, bigarren ate logikoan 0ko bat berrelikatzen da (ezeztatuta dago); ondorioz, bigarren atearen erantzuna 0 izango da, eta, beraz, *Q*.L irteera ere 1 izango

da (4.4. irudian ageri den adibidea). Beraz, *Q* irteerak aktibatzen dira (1), haien aurreko balioa edozein izanda.

Bi irteerek balio egonkorra lortu arte, denbora-tarte bat igaro da: biegonkorraren erantzun-denbora, hain zuzen ere.

R (*reset*) sarrera aktibatzen denean —*S* = 0 eta *R* = 1—, kontrakoa gertatzen da. Beheko atearen irteera 1 izango da, eta, ondorioz, *Q*.L = 0 izango da. Era berean, lehenengo atearen irteera, *Q*.H, 0 izango da, bi sarrerak 0 direlako. Beraz, *Q* irteerak desaktibatzen dira (0).

Laburbilduz: *S* aktibatzen denean, zirkuituaren irteera (*Q*.H eta *Q*.L) 1 izango da, eta *R* aktibatzen denean, 0.

Zer gertatzen da bi sarrerak 0 direnean? Kasu horretan, zirkuituaren erantzuna uneko irteeraren balioaren araberakoa da. Biegonkorraren balioa *Q* = 0 bada, erraz egiaztatzen da bi irteeren hurrengo balioa *Q'* = 0 izango dela; era berean, uneko balioa *Q* = 1 bada, hurrengo irteera *Q'* = 1 izango da[9].

Hain zuzen ere, hori bera da zirkuitu horren propietate garrantzitsua: aurreko egoeraren memoria gordetzen du. *S* seinalearen bidez 1 balioa hartu badu, 1ean geratuko da, *S* eta *R* seinaleak 0 diren bitartean. Modu berean, *R* seinalearen bidez 0 balioa hartu badu, 0ko hori mantenduko du. Beraz, **bit bateko memoria-zirkuitua** da.

4.1. taulan, zirkuituaren portaera ageri da, orain arte analizatu ditugun hiru kasuetarako:

S	*R*	*Q'*
0	0	*Q*
0	1	0
1	0	1

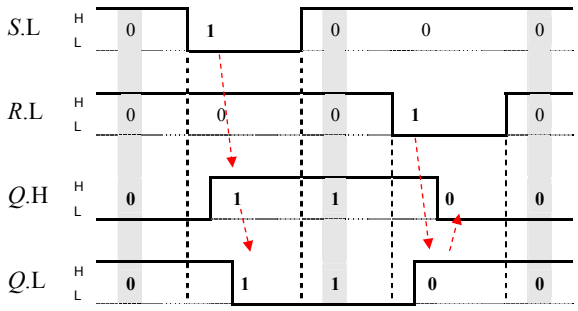
4.1. taula. SR biegonkorraren egia-taula.

Laugarren kasu bat falta da analizatzeko, *S* = *R* = 1. Zoritxarrez, kasu horretan, zirkuituaren erantzuna ezin da interpretatu aurreko kasuetan egin dugun modu berean. Bi irteerek H tentsioa hartuko dute; beraz, *Q*.H = 1 eta

[9] *Q'* izena erabiltzen da, askotan, biegonkorrak hartuko duen hurrengo balioa adierazteko.

Q.L = 0 izango dira. Ondorioz, ezin dira hartu seinale beratzat bi logiketan[10]. Hori dela eta, SR biegonkorra bit bateko memoria gisa erabil daiteke, baldin eta **inoiz ez badira aktibatzen batera bi sarrerak;** kasu hori debekatuta dago.

SR biegonkorraren portaera 4.5. irudiko kronograman bildu dugu. Hasieran, biegonkorra 0 egoeran dago, hau da, Q = 0 da (Q.H = L tentsioa eta Q.L = H tentsioa). S aktibatzen denean, biegonkorrak 1 balioa hartzen du; R = 1 denean, aldiz, 0 balioa.



4.5. irudia. SR biegonkorrari dagokion kronograma bat. NAND ateen erantzun-denbora hartu da kontuan, eta, horregatik, Q.L irteera Q.H irteera baino pixka bat geroago aktibatzen da (alderantziz 0ra igarotzeko).

SR biegonkorra asinkronoa da: sarrerak aldatzean aldatuko dira irteerak ere. Horregatik, zirkuitu oso sinplea bada ere, ez da erabiltzen, eta horren ordez, JK eta D biegonkor sinkronoak erabiltzen dira.

JK zein D biegonkorrak SR biegonkorraren eboluzioa dira. Funtsean, egitura bera dute, baina haien portaera sinkronizatu da erloju-seinale baten bidez (ez dugu ikusiko haien barne-egitura). Aldaketak, beraz, une jakin batzuetan baino ez dira gertatzen: erloju-ertzetan. Erloju-ertzaren aurreko egoera adierazteko, Q edo Q_t erabili ohi da, eta erloju-ertza pasa eta gerokoa, Q' edo Q_{t+1}.

---

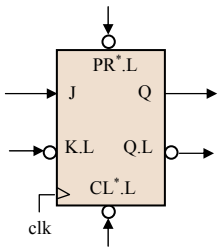
[10] Irakurleak modu sinplean egiazta dezake aurrekoa, egia-taula baten bidez. 4.4. irudiko bi irteeren ekuazio logikoak honako hauek dira:

(goikoa) $Q.\mathrm{H} = S + \overline{R}\ Q$ (behekoa) $Q.\mathrm{L} = \overline{R} \cdot (S + Q)$

Bi funtzioen egia-taulak berdinak dira, kasu batean salbu: S = R = 1 denean. Gainerako kasuetarako, interpreta daiteke bi irteerek, Q.L eta Q.H, aldagai bakar bat adierazten dutela.

4.2.2. JK biegonkorra

Bit bateko memoria duen biegonkor sinkronoa da JK biegonkorra. 4.6. irudian, ohiko JK biegonkor baten eskema logikoa eta egia-taula ageri dira[11].



clk	J	K	Q_{t+1}
↑	0	0	Q_t
	0	1	0
	1	0	1
	1	1	$\overline{Q_t}$

4.6. irudia. JK biegonkorra eta haren oinarrizko egia-taula.

Biegonkorra sinkronoa denez, erloju-seinaleak (irudian, clk) adieraziko du noiz prozesatu behar diren bi sarrera sinkronoak, *J* eta *K*, zirkuituaren erantzuna (irteera) kalkulatzeko. Egia-taulan, Q_t izenak biegonkorraren uneko balioa adierazten du, eta Q_{t+1} izenak biegonkorraren balio berria, erloju-ertza iristean hartuko duena.

Egia-taulan ageri den moduan, *J* eta *K* sarrerak 0 diren bitartean, biegonkorraren balioa —Q_t— mantentzen da. *J* aktibatuta eta *K* desaktibatuta badaude erloju-ertza gertatzen denean, biegonkorra 1 egoerara igaroko da; *K* aktibatuta eta *J* desaktibatuta badaude, berriz, 0ra igaroko da. Azkenik, bi sarrerak aktibatzen direnean, biegonkorraren balioa ezeztatuko da, hau da, *Q* 0tik 1era edo 1etik 0ra igaroko da.

Zirkuitu sinkronoa bada ere, hau da, haren aldaketak erloju-seinaleak kontrolatzen baditu ere, ohikoa da seinale asinkrono pare bat izatea biegonkorretan: *clear* (*CL**) eta *preset* (*PR**). Izartxoak adierazten du sarreren portaera asinkronoa. Edozein unetan aktibatzen direla, *CL** seinaleak 0ko bat jartzen du biegonkorrean eta *PR** seinaleak 1eko bat. Debekatuta dago biak batera aktibatzea, biegonkorraren portaera ez baitago guztiz zehaztuta kasu horretan (izan ere, *CL** eta *PR** seinaleak JK biegonkorraren oinarrian dagoen SR biegonkorraren *R* eta *S* seinaleak baino ez dira).

---

[11] Biegonkorren *J* eta *K* sarrerak edozein logikatan egon daitezke. Ohikoa da, irudian ageri den moduan, bietako bat .H izatea eta bestea .L, baina baita ere biak .H izatea. Eskuarki, irteera bi logiketan ematen da: *Q*.H eta *Q*.L. Baldin badaude, *PR** eta *CL** sarrerak logika negatiboan izan ohi dira.

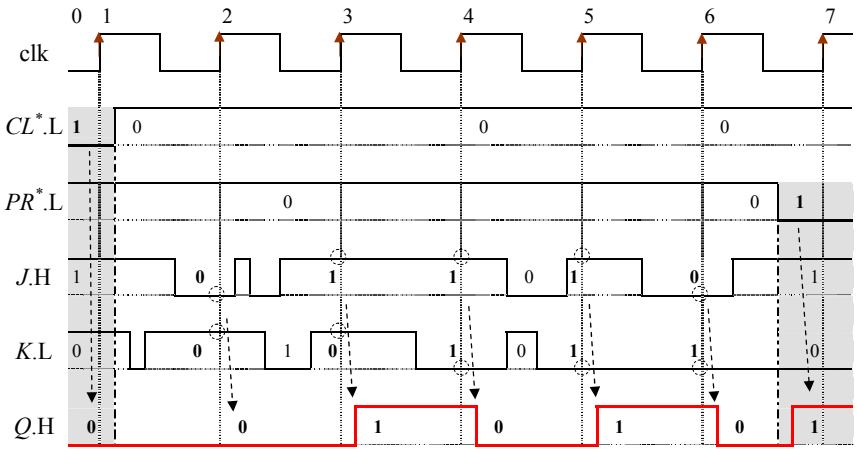
Oro har, bi seinale asinkrono horiek biegonkorra hasieratzeko erabiltzen dira bakarrik; hots, hasierako balio bat kargatzeko. Hortik aurrera, seinale sinkronoak erabiliko ditugu biegonkorra kontrolatzeko.

4.2. taulan, JK biegonkorraren egia-taula osoa ageri da.

CL^*	PR^*	clk	J	K	Q_{t+1}
1	0	–	–	–	0
0	**1**	–	–	–	1
0	0	↑	0	0	Q_t
			0	**1**	0
			1	0	1
			1	**1**	$\overline{Q_t}$

4.2. taula. JK biegonkorraren egia-taula osoa.

Azter dezagun, kronograma batean, JK biegonkorraren funtzionamendua (4.7. irudia):



4.7. irudia. JK biegonkor baten funtzionamenduaren kronograma.

0/1 Hasieran, CL^* (*clear*) seinale asinkronoa aktibaturik dagoenez, J eta K seinaleek ez dute eraginik, eta biegonkorraren irteera desaktibatuta egongo da: $Q = 0$. CL^* seinalea desaktibatzen da 1. erloju-ertzaren ondoren; hortik aurrera, eta CL^* edo PR^* seinaleak aktibatzen ez

diren bitartean, biegonkorraren portaera sinkronoa izango da, hots, haren balio-aldaketak erloju-ertzekin batera gertatuko dira.

2 2. erloju-ertza gertatzen denean, $J = K = 0$ dira: beraz, biegonkorra egoera berean mantenduko da: 0. Erloju-ertz hori baino lehen, aldaketa bat (agian, *glitch* bat) egon da K seinalean; berdin da, biegonkorrak ez ditu sarrerak prozesatzen ertza gertatu arte.

3 3. erloju-ertzean, $J = 1$ eta $K = 0$ dira. Beraz, biegonkorrak 1 balioa hartuko du. Berriro ere, 2. eta 3. erloju-ertzen artean J eta K seinaleetan izandako aldaketek ez dute eraginik izan, ez baitira prozesatu.

4/5 Erloju-ertz horietan, $J = K = 1$ dira; beraz, biegonkorraren balioa aldatzen da: 1etik 0ra eta 0tik 1era, hurrenez hurren.

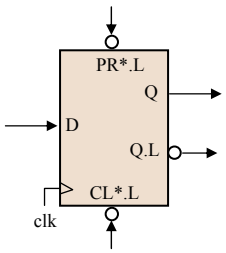
6 Une horretan, $J = 0$ eta $K = 1$ dira: ondorioz, biegonkorrak 0 balioa hartuko du.

7 Azkenik, 7. erloju-ertza baino lehen, *PR** (*preset*) seinale asinkronoa aktibatzen da. Beraz, une horretan bertan, erloju-seinalea kontuan hartu gabe, biegonkorrak 1 balioa hartuko du.

4.2.3. D biegonkorra

Esan bezala, bit bateko memoriak dira biegonkorrak, eta memoria horien gauzatze bat baino gehiago dago. Dagoeneko JK biegonkorra aztertu dugu, baina erabiliena, seguru asko, D biegonkorra da.

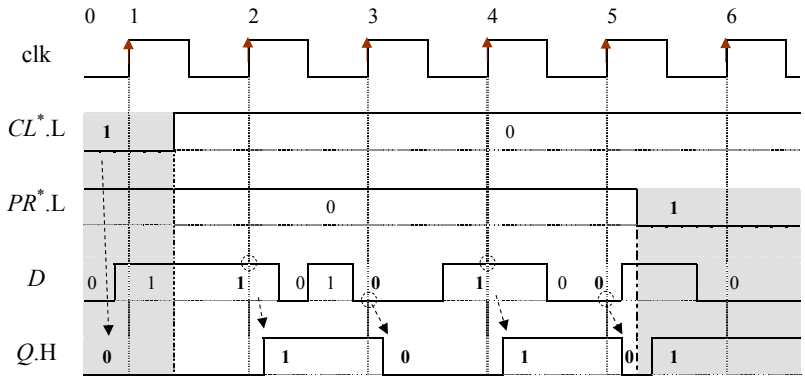
Egitura antzekoa da, baina datu-sarrera sinkrono bakarra du: D. Horrez gain, ohiko bi sarrera asinkronoak ere baditu: *clear* eta *preset*. 4.8. irudian, D biegonkorraren ohiko eskema logikoa eta egia-taula ageri dira.



*CL**	*PR**	clk	*D*	Q_{t+1}
1	0	–	–	0
0	**1**	–	–	1
0	0	↑	**0**	**0**
			1	**1**

4.8. irudia. D biegonkor baten eskema eta egia-taula.

Egia-taulan ageri den moduan, D biegonkorraren portaera oso sinplea da. Erloju-ertzetan, D sarrerako balioa irteerara igarotzen da. Ertzetik ertzera, balioa mantentzen du. Ikus dezagun portaera hori kronograma batean, 4.9. irudian.



4.9. irudia. D biegonkor baten kronograma.

0/1 Hasieran, *CL** aktibaturik dagoenez, biegonkorraren irteera 0 izango da, beste seinaleen balioak direnak direla. 1. erloju-ertzean, beraz, ez da D sarrera prozesatzen.

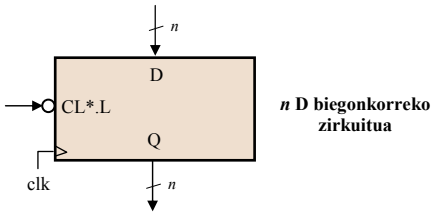
2 2. erloju-ertza baino lehen, *CL** seinalea desaktibatzen da. Beraz, biegonkorrak portaera sinkronoa izango du hortik aurrera. Erloju-ertzean $D = 1$ denez, biegonkorrak 1 balioa hartuko du.

3 2. eta 3. erloju-ertzen artean, *D* sarreraren balioa aldatzen da behin eta berriz, baina aldaketa horiek ez dute eraginik biegonkorraren balioan. 3. erloju-ertzean, berriz, $D = 0$ da; beraz, biegonkorraren irteera $Q = 0$ izango da, eta hala mantenduko da ziklo osoan zehar.

4/5 4. erloju-ertzean, *D* sarrera 1 da; beraz, irteera $Q = 1$ izango da hortik aurrera. Era berean, 5. erloju-ertzean $D = 0$ da, eta biegonkorra 0ra joango da.

6 6. erloju-zikloa baino lehen, *PR** seinale asinkronoa aktibatu da; derrigorrean, biegonkorrak 1 balioa hartuko du, erloju-seinaleari kasu egin gabe; hor mantenduko da *PR** seinalea aktibatuta dagoen bitartean.

Ohikoa da erabili behar izatea funtzio bererako D biegonkor bat baino gehiago. Kasu horietarako, *n* D biegonkor dituen zirkuitu bat erabil daiteke, eskuarki biegonkor guztietarako CL seinale komun bat erabiltzen duena:



4.2.4. JK eta D biegonkorren erabilera

Bi biegonkor erabilienak aztertu ditugu: JK eta D. Izan ere, nahikoa litzateke biegonkor bakar batekin, biek funtzio bera betetzen baitute: bit bateko memoria gordetzea. Hala ere, ohikoa da biak erabiltzea sistema digitaletan, erabilera berezitua dute eta.

JK biegonkorrak erabiltzen dira, batik bat, adierazleak sortzeko; hau da, "pizteko" eta "itzaltzeko" kontrolatu behar den bit bateko informazioa. Adibidez, eragiketa batean gainezkatzea gertatu den ala ez adierazteko; erabiltzaileari "abisuak" emateko, eta abar. *J* eta *K* sarrerak kontrol-seinale gisa erabili ohi dira, adierazlea (biegonkorra) aktibatzeko (*J*) eta desaktibatzeko (*K*). 6. kapituluan, adibide batzuk landuko ditugu.

D biegonkorrak, aldiz, informazioa gordetzeko erabiltzen dira batik bat. Izan ere, hurrengo atalean ikusiko dugun bezala, *n* D biegonkorrekin (eta multiplexoreekin) *n* biteko erregistro bat eraiki daiteke. Erregistro batean *n* biteko datu bat gorde daiteke (esaterako, 64 biteko datu bat), eta, hainbat erregistro erabiliz, erregistro-multzo bat osa daiteke, eta, azkenik, memoria bat ere. Erregistro-multzoak eta memoriak 5. kapituluan aztertuko ditugu. Bestalde, 6. kapituluan ikusiko dugun moduan, D biegonkorrak erabiltzen dira kanpo-seinaleak sinkronizatzeko ere; hots, haien aldaketak sistemako erloju-seinalearekin bat etortzeko.

Horrez gain, sistema digitalen kontrol-unitateen egoera gordetzeko erabiltzen dira D biegonkorrak (ikus 6. kapitulua). Sistema digital baten kontrol-unitatea automata bat da, hainbat egoeraren artean eboluzionatzen duena sarrera-seinale batzuen arabera; hau da, aurreikusitako sekuentzia bati jarraitzen dio. Sekuentziadore horiek nahiko modu sinplean eraiki daitezke D biegonkorren bidez. Kapitulu honetako ariketetan eta, batik bat, 6. kapituluan, sekuentziadoreak eta kontrol-unitateen diseinua landuko ditugu.

4.3. BLOKE SEKUENTZIALAK: ERREGISTROAK

Bit bateko memoria duten gailuak aztertu ditugu aurreko atalean, biegonkorrak. Jakina, bit bateko baino memoria gehiagoko gailuak behar dira, oro har, sistema digital guztietan. Gailu horiek erregistroak eta memoriak dira.

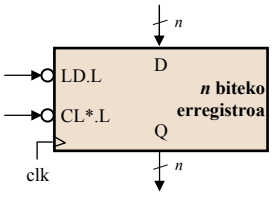
Erregistroa *n* biteko datu bat gordetzeko gauza den gailua da, eta eskuarki, D biegonkorren bidez eraikitzen da. Erregistro mota asko dago; atal honetan hiru hauek aztertuko ditugu:

- **Erregistro** soila. *n* biteko datu bat gordetzen du.
- **Desplazamendu-erregistroa.** *n* biteko datu bat gordetzeaz gain, edukiarekin eragiketa jakin bat exekuta dezake: biten desplazamendua, ezkerrera zein eskuinera.
- **Kontagailua.** *n* biteko datu bat gordetzeaz gain, eragiketa aritmetiko sinpleak exekuta ditzake: edukiari 1 gehitu ($K + 1$) edo 1 kendu ($K - 1$).

Kasu guztietan, erregistro sinkronoak analizatuko ditugu.

4.3.1. Erregistroak (*registers*)

n biteko erregistro bat gauza da *n* biteko datu bat kargatzeko eta gordetzeko, behar diren ziklo guztietan, beste datu bat kargatu nahi den arte. 4.10. irudian, erregistro orokor baten eskema logikoa eta funtzionamenduaren egia-taula ageri dira.



CL^*	clk	LD	Q_{t+1}	
1	–	–	0	edukia ezabatu
0	↑	0	Q_t	edukia mantendu
0	↑	1	D	sarrerako datua kargatu

4.10. irudia. *n* biteko erregistro baten ikurra eta egia-taula.

Aukera asko badago ere, ohikoak dira sarrera eta irteera hauek erregistroetan:

- Datu-sarrera

  D: *n* biteko datu-sarrera. Hor kokatu behar da erregistroan kargatu nahi den informazioa.

- Datu-irteera

  Q: *n* biteko datu-irteera, erregistroaren edukia hain zuzen ere.

- Kontrol-seinaleak

  ***LD*:** Kontrol-seinale nagusia, datu bat kargatu behar dela adierazteko (*load*, kargatu).

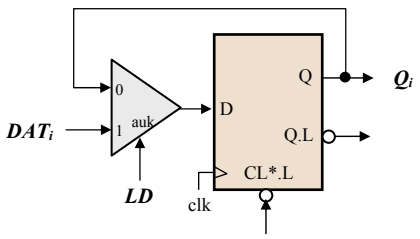
  ***CL**:** Ohiko hasieratze-seinalea (*clear*), erregistroaren edukia ezabatzeko (0ko bat sartzeko), eskuarki seinale asinkronoa.

Erregistroaren funtzionamendua 4.10. irudiko egia-taulan ageri da: datuak kargatzen dira, erloju-seinaleak adierazten duenean, baldin eta *LD* seinalea aktibatuta badago (eta *clear* desaktibatuta); bestela, erregistroaren edukia mantentzen da.

Erregistroak egiteko, D biegonkorrak erabiltzen dira. Baina D biegonkor soilak arazo bat dauka erregistroarena egiteko. Izan ere, ziklo bakar batean mantentzen du informazioa, hurrengo zikloan beste datu bat, sarreran dagoena, kargatuko baitu (ikus 4.8. irudiko egia-taula).

Nola mantendu biegonkorraren informazioa denboran zehar? Soluzioa ez da zaila: nahikoa da *D* sarrerara eramatea biegonkorraren edukia, *Q* irteera, berriro kargatzeko. Beraz, bi aukera izango ditugu D sarreran: biegonkorraren balioa edo kargatu nahi dugun berria. Hori gauzatzeko, multiplexore bat erabili beharko dugu, bata edo bestea aukeratu ahal izateko. 4.11. irudian, erregistro baten *i* bitaren eskema logikoa ageri da.

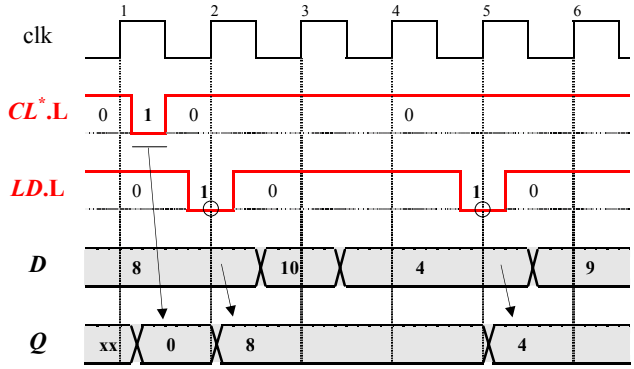
Erloju-ertz bakoitzean, bit bat kargatzen da D biegonkorrean: *LD* = 0 bada, jadanik kargatuta zegoena, hori baita multiplexorean aukeratu dena; bestela, *LD* = 1 bada, sarrerako datua kargatuko da. Laburbilduz: datu bat kargatu edo zegoena mantendu.



4.11. irudia. Erregistroen egitura (*i* bita).

Hala, 4.11. irudiko egitura *n* aldiz errepikatuz, *n* biteko erregistroa osatzen da, non *LD* eta *CL** seinaleak komunak baitira bit guztietarako.

4.12. irudian, erregistro baten portaera deskribatzen da, kronograma baten bidez. *CL** seinalea aktibatzen denean, erregistroak 0 balioa hartzen du. Gero, 2. erloju-ertzean, *LD* seinalea aktibatuta dagoenez, sarrerako datua kargatzen da erregistroan, 8ko bat. Balio hori mantentzen da erregistroan, 5. erloju-ertza arte. Une horretan *LD* seinalea berriz aktibatuta dagoenez, beste balio bat kargatuko da, 4ko bat hain zuzen ere.

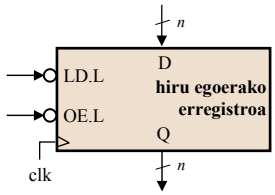


4.12. irudia. *n* biteko erregistro baten funtzionamenduaren kronograma. *n* biteko datuak adierazi behar direnean kronogrametan, ohikoa da denak biltzea lerro bakar batean, *D* eta *Q* datuekin egin dugun moduan, eta hartzen duten balioa bertan adieraztea (xx = balio ezezaguna).

4.3.1.1. Hiru egoerako erregistroak

Aurreko kapituluan azaldu dugun moduan (ikus 3.2.1. atala), badaude, bi egoera —0 eta 1— izan beharrean, hiru egoera —0, 1 eta Z— dituzten gailuak. Hirugarren egoerak irteera deskonektatuta dagoela adierazten du. Hori dela eta, hainbat irteera puntu komun batera —bus batera— konekta daitezke, baldin eta gehienez irteera bakar bat aktibatuta badago.

Erregistro arruntez gain, badaude hiru egoerako erregistroak ere, 4.13. irudikoa bezalakoak.



4.13. irudia. *n* biteko hiru egoerako erregistro baten ikurra.

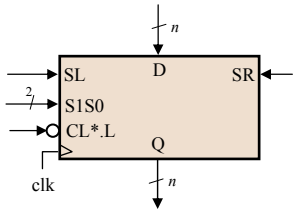
Gailu horiek, ohiko kontrol-seinaleez gain (*LD*, *CL*...), badute beste bat: *OE* (*output enable,* irteera aktibatuta). *OE* sarrera aktibatuta dagoenean, erregistroaren edukia irteeran ageri da. Desaktibatuta dagoenean, berriz, irteera Z egoeran egongo da: deskonektatuta.

OE kontrol-seinaleak erregistroaren irteeraren egoera baino ez du kontrolatzen. Beraz, irteera desaktibatuta izanda ere, erregistroaren funtzionamendua ohikoa izango da: datu bat kargatu *LD* seinalea aktibatuta badago erloju-ertzetan, edukia ezabatu eta 0 bat jarri *CL* seinalea aktibatzen denean, eta abar.

4.3.2. Desplazamendu-erregistroak (*shift registers*)

Desplazamendu-erregistroak, izenak dioen moduan, erregistroak dira, baina, datuak gordetzeaz gain, eragiketa logiko bat egin dezakete gordeta dituzten datuekin: bit bateko desplazamendua, ezkerrera zein eskuinera.

Desplazamendu-erregistro asko dago, onartzen dituzten sarreren eta eragiketen arabera: esaterako, noranzko bakarreko desplazamenduak onartzen dituztenak, datuak seriez (bitez bit) bakarrik kargatzen dituztenak, eta abar. Guk desplazamendu-erregistro orokorrena aztertuko dugu, 4.14. irudikoa.



4.14. irudia. Desplazamendu-erregistro orokor baten eskema logikoa.

Hauek dira ohiko sarrerak eta irteerak desplazamendu-erregistro batean:

- Datu-sarrerak
  - ***D:*** *n* biteko datu-sarrera, desplazamendu-erregistroan, kargatu nahi den *n* biteko datua.
  - ***SL:*** Bit bateko datu-sarrera (*left*, "ezkerrekoa"). Erregistroko pisu handieneko posizioan kargatzen den bita, edukia bit bat eskuinera desplazatzen denean.
  - ***SR:*** Bit bateko datu-sarrera (*right*, "eskuinekoa"). Erregistroko pisu txikieneko posizioan kargatzen den bita, edukia bit bat ezkerrera desplazatzen denean.

- Datu-irteera

  Q: *n* biteko datua, erregistroaren edukia hain zuzen ere.

- Kontrol-seinaleak

  S1S0: Bi biteko kontrol-seinalea, eragiketa adierazteko. Datua mantentzeaz gain, datu bat karga daiteke, edo edukia desplazatu bit bat ezkerrera edo eskuinera. Guztira, beraz, lau eragiketa.

  CL*: Hasieratze-seinalea (*clear*), erregistroaren edukia ezabatzeko (0ko bat sartzeko), eskuarki seinale asinkronoa.

Lau eragiketa egin daitezkeenez, 2 kontrol-bit behar dira —*S*1 eta *S*0—, eragiketa adierazteko. Aukera asko dago, baina, testu honetan, honela kodetuko ditugu lau eragiketak:

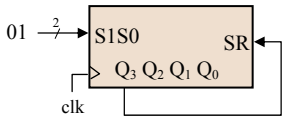
clk	*S*1	*S*0	*Q* (Q_{n-1} ... Q_0)	
↑	0	0	Q_{n-1} ... ... Q_0	Edukia mantendu
	0	1	Q_{n-2} ... Q_0 ***SR***	Edukia bit bat desplazatu ezkerrera (←); eskuinean, *SR* sarrerako bita kargatu
	1	0	***SL*** Q_{n-1} ... Q_1	Edukia bit bat desplazatu eskuinera (→); ezkerrean, *SL* sarrerako bita kargatu
	1	1	D_{n-1} ... ... D_0	*n* biteko datu bat kargatu

Adi! “Ezkerra” edo “eskuina” hitzak anbiguoak izan daitezke testuinguru honetan. Honela ulertu behar dira okerrak saihesteko: ezkerrera, pisu handiagoko bitetarantz; eskuinera, pisu txikiagoko bitetarantz.

Ikus dezagun adibide bat. 4 biteko desplazamendu-erregistro baten edukia 1110 bada, honela geratuko da bit bateko desplazamendua egin ondoren:

hasieran:	1110	
*S*1***S*0** = 0**1** (←, ezkerrera):	110-**0**	*SR* = 0 izanik
	110-**1**	*SR* = 1 izanik
S*1S*0 = **1**0 (→, eskuinera):	**0**-111	*SL* = 0 izanik
	1-111	*SL* = 1 izanik

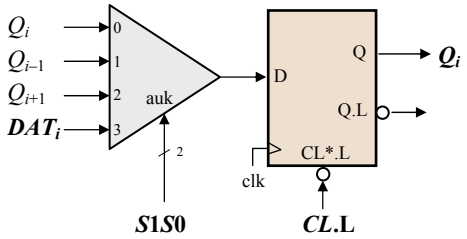
Bidenabar, desplazamendu-erregistroen bidez, edukiaren bitez biteko biraketak edo errotazioak egin daitezke. Nahikoa da, horretarako, desplazamendu bat egiten denean, berrelikatzea *SL* edo *SR* sarreretan Q_0 edo Q_{n-1} bita, hurrenez hurren, biraketaren noranzkoaren arabera. Adibidez,



edukia hasieran	1100
desplazatu bit bat ezkerrera:	100-1
desplazatu bit bat ezkerrera:	001-1
desplazatu bit bat ezkerrera:	011-0
desplazatu bit bat ezkerrera:	110-0

4.11. irudian adierazi dugu nola egin daitekeen erregistro soil bat D biegonkorrak erabiliz; ziklo bakoitzean, aukeratu egiten da, biegonkorrean kargatzeko, gordeta zegoen balioa edo balio berri bat. Desplazamendu-erregistroak egiteko ere, aurreko egitura hori erabili behar da; orain, ordea, aukera gehiago izango dugu D sarrerarako: lau, hain zuzen ere. Izan ere, bit bateko desplazamenduak honako hau esan nahi du: kargatu erregistroko *i* posizioan *i*–1 posizioko bita (desplazamendua ezkerrera) edo *i*+1 posizioko bita (desplazamendua eskuinera).

Beraz, 4 sarrerako multiplexoreak behar ditugu biegonkorren sarreretan. 4.15. irudian ageri da desplazamendu-erregistroaren bit baten egitura. Horietako *n* modulu erabiliz, *n* biteko desplazamendu-erregistro bat eraikiko dugu, non *S*1, *S*0 eta *CL* seinaleak komunak baitira.



4.15. irudia. Desplazamendu-erregistro baten bit baten eskema logikoa. *S*1*S*0 = 00, mantendu balioa; *S*1*S*0 = 01, desplazatu edukia ezkerrera (kargatu *i* posizioan *i*–1 posizioaren edukia); *S*1*S*0 = 10, desplazatu eskuinera (kargatu *i* posizioan *i*+1 posizioaren edukia); *S*1*S*0 = 11, kargatu datu berria.

Erregistroko Q_{n-1} eta Q_0 bitak (muturrekoak) bereziak dira. Desplazamendua ezkerrera egiten denean, Q_0 bitean "Q_{-1}" bita kargatu beharko genuke, baina bit hori ez da existitzen; haren ordez, *SR* sarrerako bita kargatuko dugu. Era berean, desplazamendua eskuinera denean, Q_{n-1} posizioan Q_n bita kargatu beharko genuke, baina hori ere ez da existitzen; haren ordez, *SL* sarrerako bita kargatuko dugu.

Desplazamendu-erregistroak asko erabiltzen dira sistema digitaletan. Informazioa seriean, hau da, bitez bit, prozesatu behar denean (esaterako, datuak seriean hartzeko edo transmititzeko) ohikoa da desplazamendu-erregistro bat erabiltzea. Adibidez, erregistroko Q_0 bita prozesatuko da, eta, gero, desplazamendu bat egin eskuinera, Q_1 bita Q_0 bitaren posizioan kokatzeko. Hala, begizta baten barruan, erregistroko bit guztiak prozesatuko dira.

Bestalde, bit bateko desplazamenduak ohiko eragiketa aritmetiko sinpletzat har daitezke: bider 2 eta zati 2 (ikus E1 eranskina). Adibidez (zenbaki arruntak, 6 bit):

26 = 011010	26 × 2 = 52 → 11010-0	bit bat desplazatuta ezkerrera
	26 / 2 = 13 → 0-01101	bit bat desplazatuta eskuinera

Izan ere, biderketak eta zatiketak egiten dituzten zirkuitu digitalen osagai nagusia da desplazamendu-erregistroa.

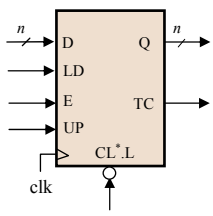
Kapitulu honetako ariketa ebatzietan (4.6. ariketan), desplazamendu-erregistro baten funtzionamenduaren kronograma bat ageri da, eta 6. kapituluan, diseinu-adibide batzuk landuko ditugu desplazamendu-erregistroak erabiliz.

4.3.3. Kontagailuak (*counters*)

Hirugarren erregistro mota kontagailua da. Kontagailua erregistro bat da (datu bat gorde dezake), baina haren ezaugarri nagusia hau da: eragiketa aritmetiko sinple bat egin dezake, $K + 1$ edo $K - 1$. Hau da, "kontatu" dezake.

Zenbait kontagailuk +1 eragiketa soilik egin dezakete (goranzko kontagailuak), baina, oro har, bi eragiketak egin ditzaketen kontagailuak erabiliko ditugu: *Up/Down* kontagailuak. Hala, kontagailu baten bidez, 0, 1, 2, 3, 2, 1, 2, 3, 4... bezalako sekuentziak sor daitezke.

4.16. irudian ageri dira ohiko kontagailu baten eskema logikoa eta haren egia-taula.



*CL**	clk	*LD*	*E*	*UP*	Q_{t+1}
1	–	–	–	–	0
0		0	0	–	Q_t
0		0	1	0	$(Q_t - 1) \bmod 2^n$
0	↑	0	1	1	$(Q_t + 1) \bmod 2^n$
0		1	–	–	D

4.16. irudia. Kontagailu baten eskema orokorra eta egia-taula (mod = modulu eragiketa, emaitza n bitekoa izan dadin; adib., 3 bitetan, 111 + 1 = 000 eta 000 – 1 = 111).

Hauek dira ohiko sarrerak eta irteerak kontagailu batean:

- Datu-sarrerak
  - ***D:*** *n* biteko datu-sarrera, kontagailuan kargatu nahi den *n* biteko datua.
- Datu-irteerak
  - ***Q:*** *n* biteko datua, kontagailuaren edukia hain zuzen ere.
  - ***TC:*** Bit bat, kontagailua mugara heldu dela adierazteko (*terminal count*; batzuetan, *RCO, ripple carry output,* deitzen zaio), kontaketaren noranzkoaren arabera: beherantz kontatzen ari bada, bit guztiak 0 direla adierazteko; gorantz kontatzen ari bada, bit guztiak 1 direla.
- Kontrol-seinaleak
  - ***LD:*** Bit bateko kontrol-seinalea, kontagailuan datuak (*D* sarrerakoak) kargatzeko.
  - ***E:*** Bit bateko kontrol-seinalea, kontatzeko agindua hain zuzen ere.
  - ***UP:*** Bit bateko kontrol-seinalea, kontaketaren noranzkoa adierazteko:

    $UP = 1 \rightarrow$ gorantz $UP = 0 \rightarrow$ beherantz
  - ***CL*:*** Hasieratze-seinalea (*clear*), erregistroaren edukia ezabatzeko (0), eskuarki seinale asinkronoa.

Lehen aipatu dugun moduan, zenbait kontagailuk ez dute *UP* seinalea, eta ezin da aukeratu kontaketaren noranzkoa; beste batzuek ez dute *CL* seinalea, eta abar. 4.7. ariketa ebatzian, kontagailu baten funtzionamenduaren kronograma bat azalduko dugu.

Kontagailuak asko erabiltzen dira sistema digitaletan, eta ohiko bloke sekuentzial gisa tratatzen dira. Hala ere, kontagailu baten barruan, oro har, erregistro bat eta batugailu bat dago, eta ez da zaila diseinatzea kontagailu bat, dagoeneko ezagutzen ditugun blokeak erabiliz. 4.17. irudian, kontagailu baten barne-egitura ageri da. Irudian ageri den moduan, *n* biteko erregistro bat eta batugailu bat erabili behar dira kontagailua egiteko. Erregistroak gordetzen du kontagailuaren balioa, eta batugailuan hurrengo balioa kalkulatzen da.

Kontagailua hasieratzeko, erregistroaren CL* sarrera erabil daiteke. Horrez gain, hiru eragiketa egin daitezke: datu bat kargatu kontagailuan ($LD = 1$), kontagailuaren balioa gehitu ($E = 1$, $UP = 1$), edo kontagailuaren balioa gutxitu ($E = 1$, $UP = 0$). Hiruretan, datu bat kargatu behar da erregistroan: kanpokoa edo batugailuaren irteera. Beraz, erregistroaren karga-seinalea aktibatu behar da $LD = 1$ edo $E = 1$ direnean (OR ate batez). +1 edo –1 eragiketa egiteko, multiplexore bat erabili dugu batugailuaren sarrera batean,

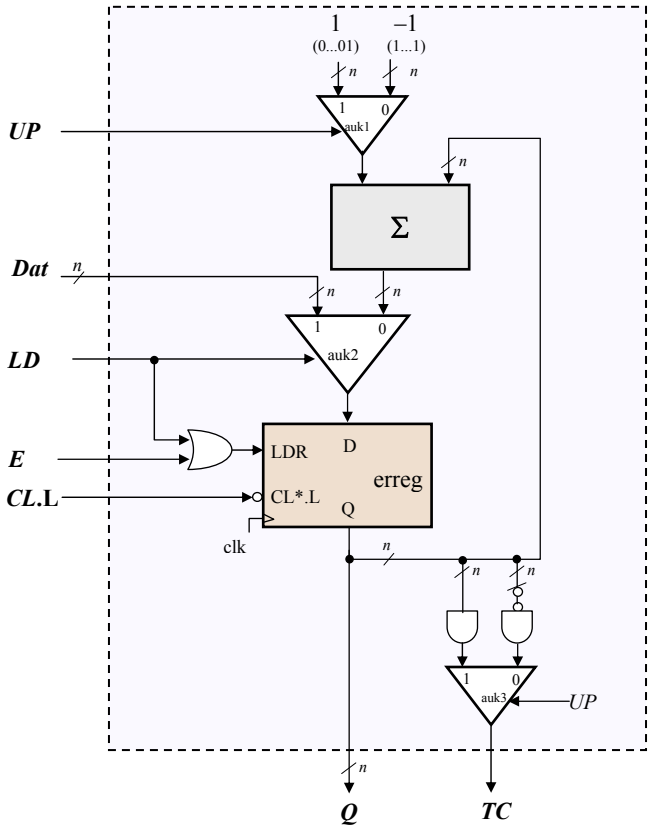
eta *UP* seinalea erabili dugu bata edo bestea aukeratzeko. Era berean, erregistroaren sarrera aukeratzeko, beste multiplexore bat erabili behar da, eta kontagailuaren *LD* seinalea erabili sarrera egokia aukeratzeko.

Azkenik, *TC* irteera emateko, ate logiko batzuk eta multiplexore bat erabili ditugu Q = 0...0 edo Q = 1...1 detektatzeko, kontaketaren noranzkoaren arabera (*UP*). Logika hori guztia honako taula honetan laburbildu dugu:

LD	*E*	*UP*	auk1	auk2	LDR	auk3
0	0	0	–	–	0	0
0	1	0	0	0	1	0
0	1	1	1	0	1	1
1	–	–	–	1	1	–

auk1 = *UP*
auk2 = *LD*
LDR = *E* + *LD*
auk3 = *UP*

Hona hemen kontagailuaren barne-egitura:



4.17. irudia. Kontagailu bat bloke funtzionalen bidez eginda.

Aipatu dugun moduan, asko erabiltzen dira kontagailuak sistema digitaletan. Adibidez, begiztetan eman beharreko iterazio kopurua kontrolatzeko, ziklo kopuru jakin bat kontatzeko, denbora-tarte jakin bat itxaroteko, eta abar. Oro har, sor daitezkeen sekuentziak oso erabilgarriak dira kontrol-unitateak eraikitzeko.

Laburpena. Kapitulu honetan zirkuitu sekuentzial sinkrono nagusiak aztertu ditugu. Zirkuitu sinkronoek memoria dute, hau da, haien irteera ez da bakarrik uneko sarreren funtzioa, sistemaren aurreko erantzunak ere kontuan hartzen baitira. Hori lortzeko, zirkuituen irteerak berrelikatzen dira sarreran.

Bit bateko memoria duten zirkuituak aztertu ditugu: JK eta D biegonkorrak. Oinarrizko gailu sinkronoak dira biegonkorrak, eta, horien bidez, asko erabiltzen diren bloke sekuentzial nagusiak sortzen dira: erregistroak.

Erregistro erabilienak hauek dira: erregistro soilak, desplazamendu-erregistroak, eta kontagailuak. Hirurak gauza dira *n* biteko hitz bat gordetzeko, baina, horrez gain, desplazamendu-erregistroekin edukiaren bit bateko desplazamenduak egin daitezke (ezkerrera zein eskuinera) eta kontagailuekin kontatze-sekuentziak egin daitezke (gorantz edo beherantz): 6, 7, 8, 9... 12, 11, 10... Erregistroak osatzeko, D biegonkorrak eta, kasuan kasu, multiplexoreak, batugailuak eta abar erabili behar dira.

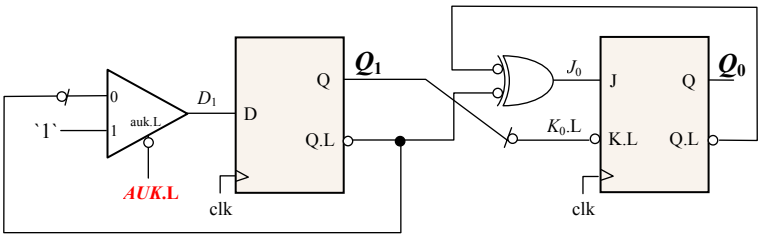
Hurrengo atalean, zirkuitu sekuentzialen portaera analizatuko dugu hainbat ariketaren bidez.

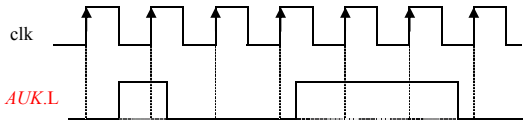
4.4. ARIKETA EBATZIAK

Kapitulua bukatzeko, ariketa batzuk ebatziko ditugu. Lehenengo bi ariketetan, zirkuitu sekuentzial sinpleenen, biegonkorren, funtzionamendua landuko dugu. Horretarako, biegonkorrez eta ateez osatutako zirkuituen kronogramak egingo ditugu. 3., 4. eta 5. ariketetan, biegonkorretan oinarritutako sekuentziadoreen eraikuntza aztertuko dugu: kanpo-seinaleen mende dauden sekuentziak zein kanpo-seinalerik gabekoak. Beste zirkuitu sekuentzial batzuen funtzionamendua aztertuko dugu 6. eta 7. ariketetan. 6.ean, desplazamendu-erregistro baten funtzionamendua aztertuko dugu, eta 7.ean, berriz, kontagailu batena. Oro har, zirkuitu konbinazionalak eta sekuentzialak (sinkronoak) batera ageri dira sistema digitaletan. Horregatik, garrantzitsua da ezagutzea elkarlanean nola diharduten. Hori da, hain zuzen ere, 8. eta 9. ariketetan landuko duguna.

4.1. Ariketa

Hartu kontuan irudiko zirkuitua eta adieraz ezazu biegonkorren D eta J, K sarrerei dagozkien funtzio logikoak. Gero, marraztu ezazu, tentsio-diagrama edo kronograma batean, sistemaren portaera: D_1 eta Q_1, eta J_0, K_0 eta Q_0 seinaleen aldaketak denboran zehar. AUK seinalearen portaera irudikoa da, eta Q_1 eta Q_0 biegonkorren hasiera-balioa 0 da.





Oro har, biegonkorrek eta multiplexoreek irudian ageri diren kontrol-seinaleak baino seinale gehiago dituzte. Zirkuitu bati dagokion seinaleren bat agertzen ez bada eskema batean, funtzionamendu egokirako behar duen balioa izango du seinale horrek. Hala gertatzen da ariketa honetan; multiplexorearen ohiko gaikuntza-seinalea aktibatuta ($G = 1$) egongo da, eta biegonkorren hasieratze-seinale asinkronoak desaktibatuta ($CL^* = PR^* = 0$).

Kronograman bi biegonkor aztertu behar direnez, gogora dezagun haien funtzionamendua. Bit bat gordetzen dute erloju-ertza aktibatzen ez den bitartean. Erloju-ertza heltzean, aldiz, biegonkorren sarrerak prozesatzen dira irteera-balioa kalkulatzeko:

clk	D	Q_{t+1}
↑	0	0
	1	1

clk	J	K	Q_{t+1}
	0	0	Q_t
↑	0	1	0
	1	0	1
	1	1	$\overline{Q_t}$

Gogoratu, biegonkor batek bit bateko “aldagai” bat gordetzen du, eta, eskuarki, bi logiketan ematen du aldagaia, Q.H eta Q.L irteeretan.

Kronograma bete ahal izateko, biegonkorren sarrerak zehazten dituzten funtzioak kalkulatu behar ditugu. Hauek dira adibide honetako biegonkorren sarrera-seinaleen ekuazioak (ikus zirkuituaren irudia):

- D biegonkorraren sarrera multiplexorearen irteera da; beraz:

  baldin ($AUK = 0$) *orduan* $D_1 = \overline{Q_1}$
  bestela $D_1 = 1$

- JK biegonkorraren sarrerak:

  $J_0 = Q_0 \oplus Q_1$ eta $K_0 = \overline{Q_1}$

Kronograma urratsez urrats betetzen hasteko prest gaude. Kasu honetan, ezinezkoa da seinale baten eboluzioa ezagutzea besteena kontuan hartu gabe eta, beraz, kronograma betetzeko prozesua bertikalean, erloju-ertzez erloju-ertz, egin beharko dugu.

Aurreko kapituluetan esan dugun moduan, edozein zirkuitu fisikok erantzun-denbora (edo atzerapen) jakin bat behar du emaitza emateko, hau da, erantzuna ez da berehalakoa. Ondorioz, zirkuitu konbinazionalen irteerak ez dira aldatuko sarrerak aldatu bezain laster, eta zirkuitu sekuentzialen irteerak ez dira erloju-ertzarekin batera aldatuko, pixka bat geroago baizik.

Kronograma sinplifikatzeko, zirkuitu sekuentzialen atzerapenak soilik irudikatuko ditugu. Has gaitezen kronograma egiten, urratsez urrats.

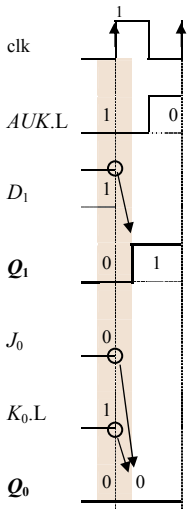
■ **Hasierako balioak**

Q_1 eta Q_0 seinaleen hasiera-balioak 0 dira, eta ez dira lehenengo erloju-ertzeraino aldatuko. Seinale horietan oinarrituz, D_1, J_0 eta K_0 sarreren balioak kalkula ditzakegu.

Multiplexorearen irteerak emango digu D_1-en balioa; *AUK* seinalea 1 denez, 1 sarrerako datua emango du irteeran multiplexoreak: 1ekoa, alegia.

J_0-ren balioa ezagutzeko, $Q_0 \oplus Q_1$ kalkulatu behar dugu: $0 \oplus 0 = 0$.

Eta azkenik, K_0, Q_1-en osagarria, 1 izango da.



■ **1. erloju-ertza**

Erloju-ertza gertatutakoan, biegonkorren irteerak (Q_1 eta Q_0) aldatu egingo dira, sarreretan dauden balioen eta funtzionamendu-taulen arabera:

$$D_1 = 1 \quad \Rightarrow Q_1 = 1$$
$$J_0 = 0,\ K_0 = 1 \quad \Rightarrow Q_0 = 0$$

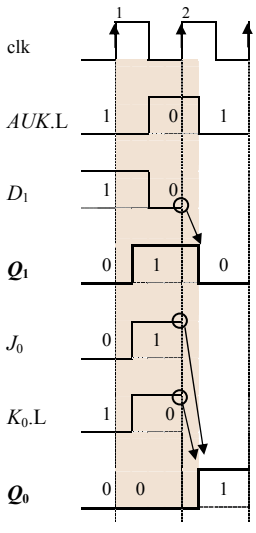
Balio horiek hurrengo erloju-ertza iritsi arte mantenduko dira.

■ **1. erloju-zikloa**

Q_1, Q_0, eta *AUK* seinaleen arabera, D_1, J_0 eta K_0 seinaleen balioak kalkula ditzakegu.

1. erloju-zikloan, *AUK* seinalearen balioa aldatu egiten da, eta, ondorioz, D_1 ere aldatuko da. Zikloaren hasieran, *AUK* = 1 da eta, beraz, multiplexorearen irteera 1 da; gero, *AUK* 0 izatera pasatuko da eta multiplexorearen 0 sarrera aukeratuko da irteeran; hau da, Q_1-en osagarria, 0koa, hain zuzen ere.

J_0-ren balioa $Q_0 \oplus Q_1 = 0 \oplus 1 = 1$ izango da, eta, azkenik, K_0 0 izango da, Q_1-en osagarria.



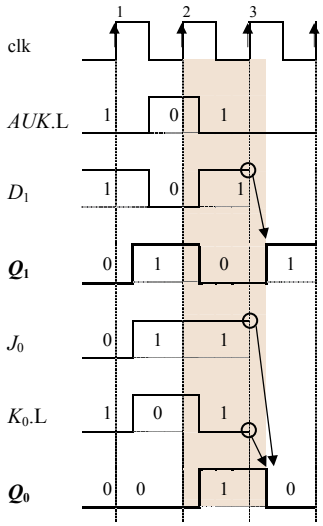
■ **2. erloju-ertza**

Erloju-ertza iristean, biegonkorren irteerak (Q_1 eta Q_0) aldatu egingo dira, sarreretan dauden balioen arabera.

$$D_1 = 0 \quad \Rightarrow Q_1 = 0$$
$$J_0 = 1,\ K_0 = 0 \quad \Rightarrow Q_0 = 1$$

■ **2. erloju-zikloa**

Berriz ere, D_1, J_0 eta K_0 seinaleek dituzten balioak kalkulatu behar ditugu.
AUK seinalea 1 denez, D_1 ere 1 izango da.
J_0-ren balioa $Q_0 \oplus Q_1 = 1 \oplus 0 = 1$ izango da, eta, azkenik, K_0, Q_1-en osagarria, 1 izango da.



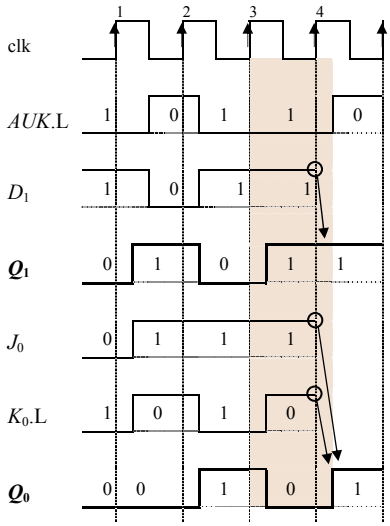
■ **3. erloju-ertza**

Erloju-ertza gertatzean, biegonkorren irteerak (Q_1 eta Q_0) aldatu egingo dira, sarreretan dauden balioen arabera.

$D_1 = 1 \quad \Rightarrow Q_1 = 1$
$J_0 = 1, K_0 = 1 \quad \Rightarrow Q_0 = 0$

■ **3. erloju-zikloa**

AUK seinaleak 1eko balioa mantentzen du 3. erloju-zikloan zehar, eta beraz, $D_1 = 1$ izango da.
Era berean, $J_0 = Q_0 \oplus Q_1 = 0 \oplus 1 = 1$ izango da, eta, azkenik, K_0, Q_1-en osagarria, 0 izango da.



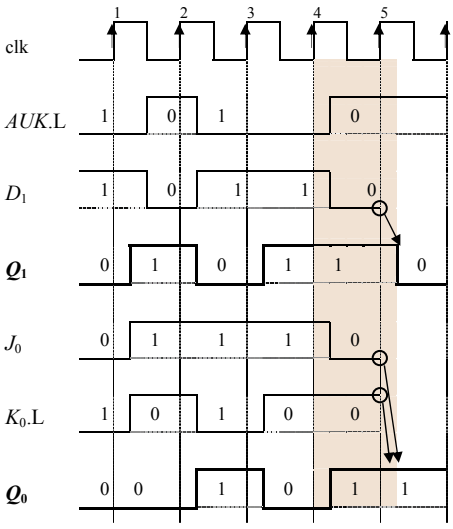
■ **4. erloju-ertza**

Erloju-ertzarekin batera, biegonkorren irteerak (Q_1 eta Q_0) aldatu egingo dira, sarreretan dauden balioen arabera.

$D_1 = 1 \quad \Rightarrow Q_1 = 1$
$J_0 = 1, K_0 = 0 \quad \Rightarrow Q_0 = 1$

■ **4. erloju-zikloa**

AUK seinaleak 0ko balioa hartzen du ziklo honetan; beraz, D_1-ek (Q_1-en osagarria) 0koa hartuko du.
$J_0 = Q_0 \oplus Q_1 = 1 \oplus 1 = 0$ da. Azkenik, K_0-k, Q_1-en osagarria denez, 0ko balioa hartuko du.



■ **5. erloju-ertza**

Erloju-ertzarekin batera, biegonkorren irteerak (Q_1 eta Q_0) aldatu egingo dira, sarreretan dauden balioen arabera.

$D_1 = 0 \quad \Rightarrow Q_1 = 0$
$J_0 = 0, K_0 = 0 \quad \Rightarrow Q_0 = 1$

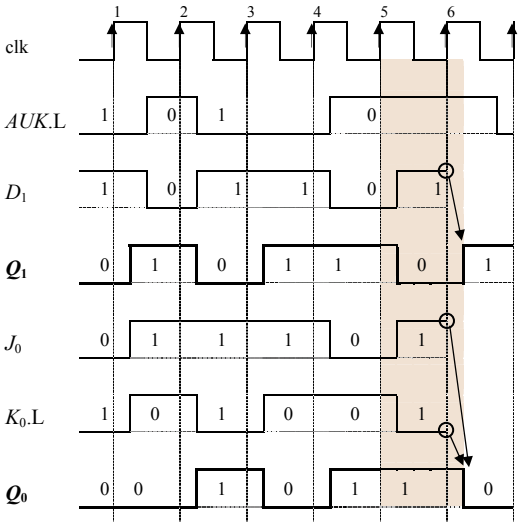
■ **5. erloju-zikloa**

AUK seinaleak 0ko balioa hartzen du ziklo honetan ere eta, beraz, $D_1 = 1$ izango da. J_0-ren balioa, $Q_0 \oplus Q_1 = 1 \oplus 0 = 1$ da eta, azkenik, K_0 ere, Q_1-en osagarria, 1 izango da.

■ **6. erloju-ertza**

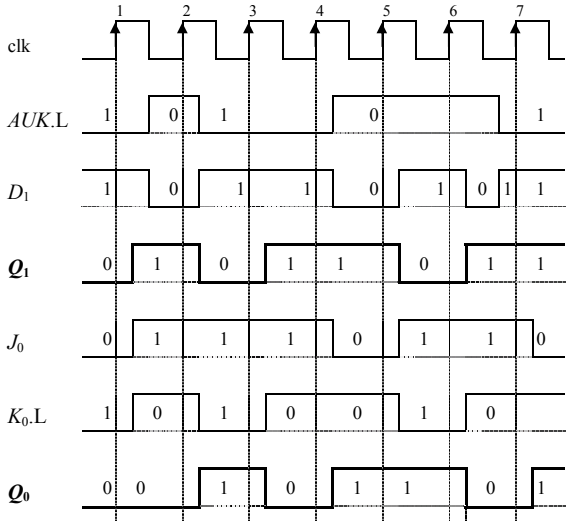
Erloju-ertzarekin batera, biegonkorren irteerak (Q_1 eta Q_0) aldatu egingo dira, sarreretan dauden balioen arabera.

$D_1 = 1 \quad \Rightarrow Q_1 = 1$
$J_0 = 1, K_0 = 1 \quad \Rightarrow Q_0 = 0$



Azken zikloa besterik ez zaigu geratzen kronograma amaitzeko. Gainerakoetan egin dugun moduan, D_1, J_0 eta K_0-ren balioak kalkulatu eta

horien arabera, ezagutu ahal izango dugu erloju-ertza eta gero Q_1-ek eta Q_0-k edukiko duten balioa. Hona hemen kronograma osoa:

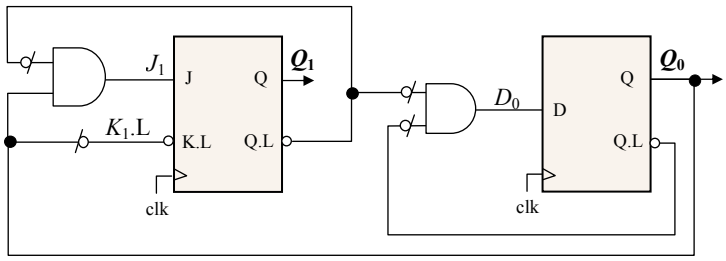


«

4.2. Ariketa

Irudiko zirkuitua bi biteko sekuentziadorea da; biegonkorren balioek sekuentzia jakin bati jarraitzen diote. Q_1-ek eta Q_0-k sekuentziadorearen egoera adierazten dute, sekuentziaren balio jakin bat, alegia.

Adieraz itzazu biegonkorren J, K eta D sarrerei dagozkien funtzio logikoak. Gero, bete ezazu egoeren arteko trantsizio-taula eta esan zein segidari jarraitzen dion zirkuituak. Azkenik, osa ezazu sistemaren funtzionamendua islatzen duen kronograma (hasieran, $Q_1Q_0 = 00$).



Zirkuituaren irudian ageri denez, bi biegonkorren sarrerak —J_1, K_1 eta D_0— sistemaren egoeraren funtzioak dira; hots, Q_1-en eta Q_0-ren funtzioak.

Zirkuitua analizatu behar dugu biegonkorren sarreren ekuazioak eskuratzeko. Analisi horren arabera, hauek dira JK biegonkorraren bi sarrerak, zirkuituaren eskeman ageri diren moduan:

$$J_1 = \overline{Q_1} \cdot Q_0$$

$$K_1 = \overline{Q_0}$$

Era berean, hau da D biegonkorraren sarreraren ekuazio logikoa:

$$D_0 = \overline{Q_1} \cdot \overline{Q_0}$$

Funtzio horiek erabili behar ditugu sistemaren portaera taula batean (egoeren arteko trantsizio-taulan) bildu ahal izateko. Egin dezagun, bada, taula hori.

Egoeren arteko trantsizio-taula betetzeko, Q_1-en eta Q_0-ren balio posible guztiak aztertu behar dira. Balio horiek sistemaren egoera —uneko egoera (UE)— adierazten dute. Kasu bakoitzerako, J_1, K_1 eta D_0 sarreren balioak kalkulatu behar ditugu, eta horien arabera, zein izango den sistemaren hurrengo egoera (HE) erloju-ertza aktibatzean. Hurrengo egoera adierazteko, Q' letrak erabiliko ditugu.

	UE					HE	
	Q_1	Q_0	J_1	K_1	D_0	$\boldsymbol{Q_1'}$	$\boldsymbol{Q_0'}$
*E*0	0	0					
*E*1	0	1					
*E*2	1	0					
*E*3	1	1					

*E*0 egoeran —Q_1Q_0 = 00—, J_1, K_1 eta D_0 seinaleek honako balio hauek hartuko dituzte:

$$J_1 = \overline{Q_1} \cdot Q_0 = \bar{0} \cdot 0 = 1 \cdot 0 = 0$$

$$K_1 = \overline{Q_0} = \bar{0} = 1$$

$$D_0 = \overline{Q_1} \cdot \overline{Q_0} = \bar{0} \cdot \bar{0} = 1 \cdot 1 = 1$$

Erloju-ertza gertatzen den unean, sarrerak prozesatuko dira. Beraz, J_1 = 0 eta K_1 = 1 direnez, JK biegonkorraren balio berria Q_1' = 0 izango da. Era berean, D_0 = 1 denez erloju-ertza aktibatzen denean, D biegonkorraren irteera, erloju-ertza eta gero, Q_0' = 1 izango da.

Analiza dezagun orain *E*1 egoera —Q_1Q_0 = 01—. Kasu honetan, hauek izango dira biegonkorren sarrera-seinaleen balioak:

$$J_1 = \overline{Q_1} \cdot Q_0 = \bar{0} \cdot 1 = 1$$

$$K_1 = \overline{Q_0} = \bar{1} = 0$$

$$D_0 = \overline{Q_1} \cdot \overline{Q_0} = \bar{0} \cdot \bar{1} = 0$$

Erloju-ertzarekin batera, balio horiek prozesatuko dira biegonkorretan. Hala, Q_1' = 1 izango da, une horretan J_1 sarrera aktibatuta dagoelako, eta, era berean, Q_0' = 0 izango da, D_0 = 0 delako.

Prozedura bera erabili behar da taulako gainerako errenkadak betetzeko. Hona hemen taula osoa beteta.

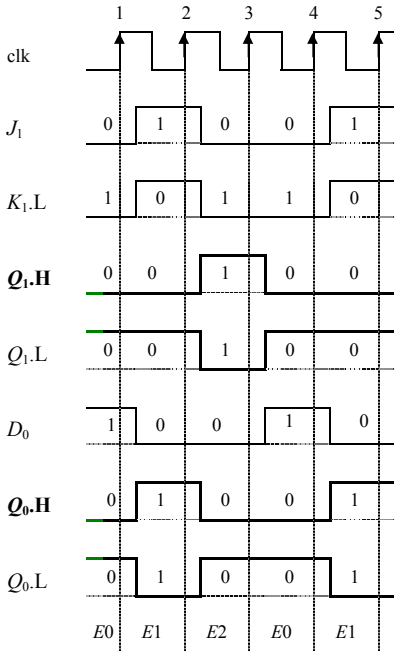
	UE					HE	
	Q_1	Q_0	J_1	K_1	D_0	$\boldsymbol{Q_1'}$	$\boldsymbol{Q_0'}$
*E*0	0	0	0	1	1	0	1
*E*1	0	1	1	0	0	1	0
*E*2	1	0	0	1	0	0	0
*E*3	1	1	0	0	0	1	0

Taula horretatik ondoriozta daiteke zirkuituak sortzen duen segida. Hasieran, $Q_1Q_0 = 00$ da; beraz, sekuentziadorearen hurrengo egoera, erloju-ertza eta gero, $Q_1Q_0 = 01$ izango da. 01 ($E1$) egoeratik, 10 ($E2$) egoerara doa; eta $E2$ egoeratik, berriro $Q_1Q_0 = 00$ egoerara doa, hau da, $E0$-ra. Une horretatik aurrera, sekuentzia osoa errepikatuko da, behin eta berriz.

Beraz, hau da sekuentziadorearen segida osoa, $E0$ egoeratik abiatuta: 00 – 01 – 10 – 00 – 01 – 10 – 00 – ...

Argi geratzen da sekuentziadorea ez dela berez 11 egoerara joango; hala ere, hasieratzen bada egoera horretan (esaterako, biegonkorren *Preset* hasieratze-seinaleak erabiliz), bere kabuz sartuko da sekuentzian: 11 – 10 – 00 – 01 – ...

Sekuentziadorearen egoera-taula kronograma batean isla daiteke.

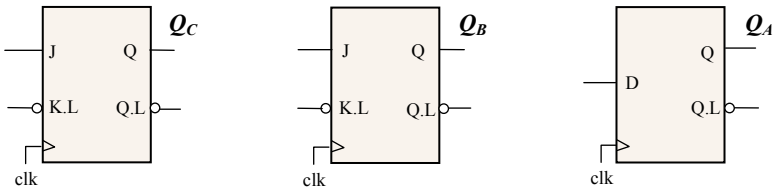


Erloju-ziklo bakoitza taulako errenkada batekin erlaziona dezakegu. Lehenengo zikloan, adibidez, $Q_1Q_0 = 00$ dira, taulako $E0$ errenkadako egoera, hain zuzen ere. Beraz, $J_1 = 0$, $K_1 = 1$ eta $D_0 = 1$ izango ditugu, eta, ondorioz, hurrengo egoera $Q_1Q_0 = 01$ izango da. Modu berean interpreta daitezke kronogramaren gainerako zikloak.

«

4.3. Ariketa

Erabil itzazu bi JK biegonkor eta D biegonkor bat 0 – 2 – 4 – 5 – 3 – 7 – 0 – 2 – 4 – ... sekuentziari jarraitzen dion zirkuitua egiteko.



Sortu behar den sekuentzia hiru bitekoa denez (4, 5 eta 7 egoerak adierazteko, 3 bit behar dira), 3 biegonkor erabili behar ditugu sekuentziadorea eraikitzeko. Adibide honetan, bi JK biegonkor pisu handieneko bitetarako, eta D biegonkor bat pisu txikieneko biterako. Sekuentziadorea eraikitzeko, biegonkorrak kontrolatzen dituzten sarreren ekuazioak lortu behar ditugu; hau da, J_C, K_C, J_B, K_B eta D_A.

Sekuentziadorearen portaera taula batean bil daiteke, non adierazten den sekuentziaren bit bakoitza —biegonkor bakoitzaren irteera— nola aldatzen den erloju-ertz bakoitzean, egoera guztietarako. Esaterako, eraiki behar dugun sekuentzian, 0aren atzetik datorren balioa 2a da. Zer adierazten du horrek? Erloju-ziklo batean $Q_CQ_BQ_A$ hirukoteak 000 balioa hartzen badu, erloju-ertza eta gero 010 balioa hartu beharko duela, hain zuzen ere. Modu berean interpretatu behar ditugu sekuentziaren gainerako balioak.

Hona hemen sekuentziari dagokion taula osoa:

Uneko Egoera	$Q_C\,Q_B\,Q_A$	Hurrengo Egoera	$Q_C'\,Q_B'\,Q_A'$
0	0 0 0	2	0 1 0
1	0 0 1	–	– – –
2	0 1 0	4	1 0 0
3	0 1 1	7	1 1 1
4	1 0 0	5	1 0 1
5	1 0 1	3	0 1 1
6	1 1 0	–	– – –
7	1 1 1	0	0 0 0

Hiru bitekin adieraz daitezkeen 8 egoeretatik 2 ez dira erabiltzen adibide honetako sekuentzian: 1a eta 6a. Beraz, egoera horiek inoiz agertuko ez direnez, haien hurrengo egoerak zehaztu gabeko gaitzat har daitezke.

Aurreko taulan adierazitako trantsizioak sortzeko, biegonkorren sarrerak kontrolatu behar dira, eta hori egiteko, biegonkorren funtzionamendua hartu beharko dugu kontuan.

D biegonkorraren kasuan, erraza da lan hori: D sarreran jarri behar dugu irteeran lortu nahi dugun balioa; hala, erloju-ertza iristean, balio hori kargatuko da biegonkorrean.

$$Q' = 0 \quad \rightarrow \quad D = 0$$
$$Q' = 1 \quad \rightarrow \quad D = 1$$

JK biegonkorren kasua, berriz, ez da hain sinplea; biegonkorrak aurretik zuen balioak badu lotura erloju-ertza eta gero edukiko duenarekin. Garrantzitsua da JK biegonkorraren taula kontuan edukitzea eragin hori nolakoa izango den jakiteko.

Ikus dezagun adibide bat: $Q = 0$ izanik $Q' = 0$ izatea nahi denekoa. Gainerako kasu guztietan gertatuko den moduan, trantsizio hori lortzeko bi aukera daude:

$J = 0$, $K = 0$ izatea: biegonkorrak aurretik zuen balioa mantenduko du.

$J = 0$, $K = 1$ izatea: biegonkorraren irteeran 0koa ezarriko da.

Bi aukerak aztertuz, hau nabarmen dezakegu: $J = 0$ izan behar da eta K-k edozein balio har dezake, 0 edo 1 (zehaztu gabeko gaia, "–" ikurra taulan).

Modu berean azter daitezke gainerako kasuak. Honako taula honek laburtzen ditu aukera guztiak:

Q	Q'	J	K	$\boldsymbol{J}$	$\boldsymbol{K}$
0	0	0 0	0 1	0	–
0	1	1 1	1 0	1	–
1	0	1 0	1 1	–	1
1	1	0 1	0 0	–	0

Prest gaude biegonkorren sarrerak definitzeko, sekuentziaren egoera-aldaketa bakoitzerako: JK biegonkorren J_CK_C eta J_BK_B sarrerak, eta D biegonkorraren D_A sarrera.

Taula hauetan ageri dira sarrera horien balioak egoera guztietarako:

UE Q_C	HE Q_C'	J_C	K_C
0	0	**0**	–
0	–	–	–
0	1	**1**	–
0	1	**1**	–
1	1	–	**0**
1	0	–	**1**
1	–	–	–
1	0	–	**1**

UE Q_B	HE Q_B'	J_B	K_B
0	1	**1**	–
0	–	–	–
1	0	–	**1**
1	1	–	**0**
0	0	**0**	–
0	1	**1**	–
1	–	–	–
1	0	–	**1**

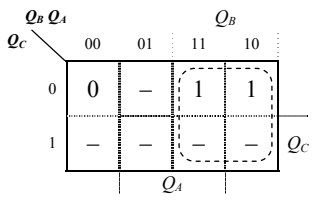
UE Q_A	HE Q_A'	D_A
0	0	**0**
1	–	–
0	0	**0**
1	1	**1**
0	1	**1**
1	1	**1**
0	–	–
1	0	**0**

Ohikoagoa da taula hori beste modu honetan ematea:

U. Egoera $Q_C\ Q_B\ Q_A$	H. Egoera $Q_C'Q_B'Q_A'$	J_C	K_C	J_B	K_B	D_A
0 0 0	0 1 0	**0**	–	**1**	–	**0**
0 0 1	– – –	–	–	–	–	–
0 1 0	1 0 0	**1**	–	–	**1**	**0**
0 1 1	1 1 1	**1**	–	–	**0**	**1**
1 0 0	1 0 1	–	**0**	**0**	–	**1**
1 0 1	0 1 1	–	**1**	**1**	–	**1**
1 1 0	– – –	–	–	–	–	–
1 1 1	0 0 0	–	**1**	–	**1**	**0**

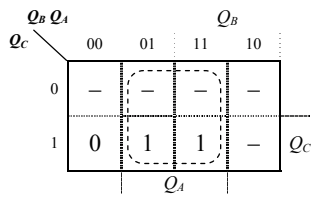
Azken pausoa, *J*, *K* eta *D* funtzio minimoak sortzea eta eraikitzea da. Funtzioak minimizatzeko, K-mapak erabiliko ditugu.

J_C



$J_C = Q_B$

K_C



$K_C = Q_A$

J_B

Q_C \ $Q_B Q_A$	00	01	11	10
0	1	–	–	–
1	0	1	–	–

$$J_B = Q_A + \overline{Q_C}$$

K_B

Q_C \ $Q_B Q_A$	00	01	11	10
0	–	–	0	1
1	–	–	1	–

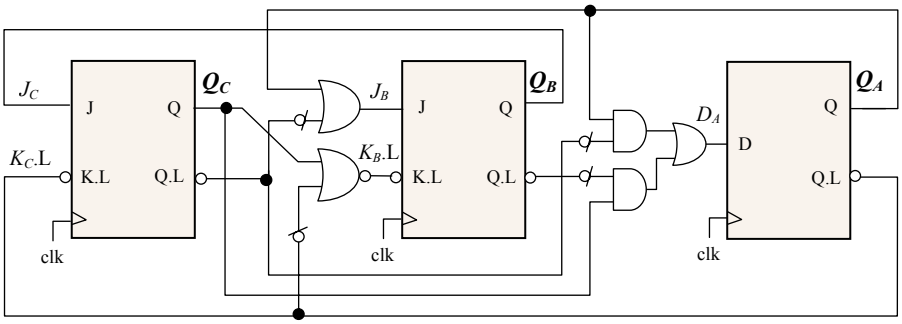
$$K_B = \overline{Q_A} + Q_C$$

D_A

Q_C \ $Q_B Q_A$	00	01	11	10
0	0	–	1	0
1	1	1	0	–

$$D_A = \overline{Q_C}\, Q_A + Q_C\, \overline{Q_B}$$

J_C, K_C, J_B, K_B eta D_A seinaleen espresio minimoak lortu ditugu. Ariketa amaitzeko, zirkuitua irudikatzea besterik ez zaigu falta. Hau izan daiteke aukera bat.



Azken ohar bat. Sekuentziadore honek ez ditu 1 eta 6 egoerak erabiltzen ohiko funtzionamenduan. Izan ere, zehaztu gabeko gaitzat hartu ditugu bi egoera horiek, eta *J*, *K* eta *D* funtzioak minimizatzeko erabili ditugu. Aztertzen baditugu hartutako erabakiak, hau aurkituko dugu: sekuentziadorea 1 edo 6 egoeretan sartzen bada (esaterako, seinale

asinkronoren bat, *clear* edo *preset*, aktibatu delako), 3 edo 4 egoeretara joango da, hurrenez hurren.

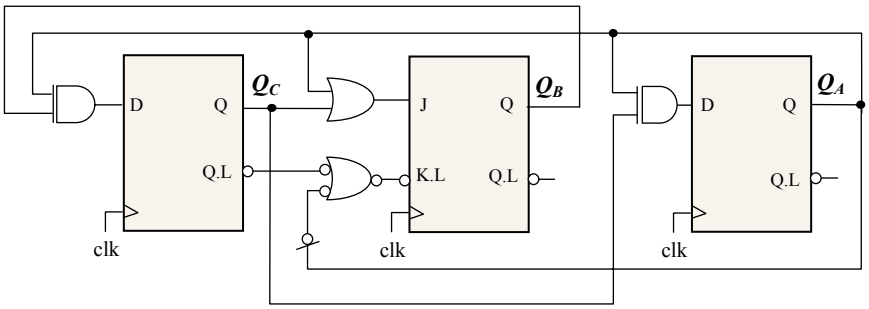
Izan ere, ezohiko funtzionamendua adierazten dute bi egoera horiek, akats bat. Segurtasuna dela eta, erabiltzen ez diren egoeretatik egoera jakin batera (hasierako egoerara, eskuarki 0 egoerara) eraman ohi dira sekuentziadoreak; hala, akats bat gertatzen bada sisteman, bere kabuz sartuko da sekuentzian, hasiera-puntuan.

Ariketa gisa, sekuentziadorea berriz diseinatzea proposatzen dugu: espero ez diren egoeretatik (1 eta 6) 0 egoerara joan behar da.

«

4.4. Ariketa

Diseinatzaile bati 0 – 5 – 3 – 6 – 0 – 5 – ... sekuentziari darraion sekuentziadorea diseinatzea eskatu zaio, eta honako eskema hau proposatu du:



Egiazta ezazu proposaturiko sekuentziadoreak eskatutako segidari jarraitzen diola; horretarako, idatz itzazu aldagaien ekuazio logikoak eta bete ezazu egoeren arteko trantsizio-taula osoa.

UE $Q_C\ Q_B\ Q_A$	D_C	J_B	K_B	D_A	**HE** $Q_C'\ Q_B'\ Q_A'$
0 0 0					
0 0 1					
0 1 0					
0 1 1					
1 0 0					
1 0 1					
1 1 0					
1 1 1					

Gero, egin ezazu zirkuituaren funtzionamendua islatzen duen kronograma. Azkenik, eraiki ezazu sekuentzia berari darraion beste sekuentziadore bat ahal den ate logiko kopururik txikiena erabiliz, hots, funtzioak minimizatuz (biegonkor berak erabiliz).

Lehenengo zeregina zirkuituaren analisia da; hots, seinale guztien ekuazioak zehaztea. Hauek dira irudiko zirkuituko seinale nagusien ekuazio logikoak:

$$D_C = Q_B \otimes Q_A$$
$$J_B = Q_C + Q_A$$
$$K_B = Q_C + \overline{Q_A}$$
$$D_A = Q_A \otimes Q_C$$

Funtzio horiek erabiliz, bete dezakegu biegonkorren trantsizio-taula: biegonkorren sarrera-seinaleen balioak eta sekuentziadorearen hurrengo egoera, uneko egoera guztietarako.

Adibidez, taulako lehenengo errenkadan, biegonkorren irteerak, $Q_C\, Q_B\, Q_A$, 0 dira; ondorioz, $D_C = 0 \otimes 0 = 1$, $J_B = 0 + 0 = 0$, $K_B = 0 + \overline{0} = 0 + 1 = 1$ eta $D_A = 0 \otimes 0 = 1$ dira. Balio horiek direla eta, erloju-ertza gertatzen denean, $Q_C' = 1$, $Q_B' = 0$ eta $Q_A' = 1$ izango dira.

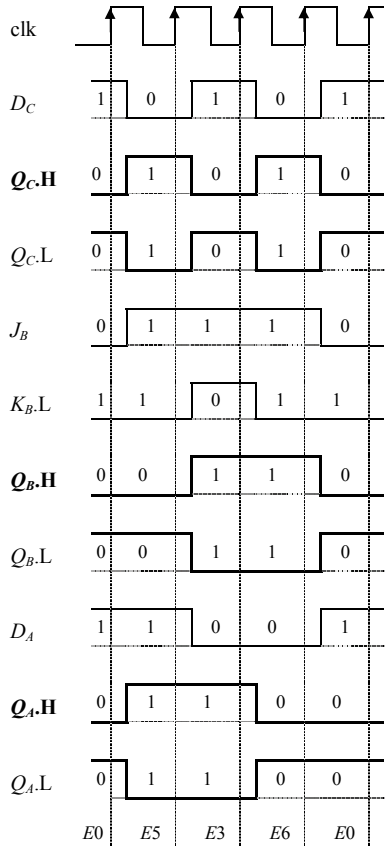
Prozesu berari jarraitu behar zaio taulako gainerako errenkadak betetzeko.

	UE $Q_C\ Q_B\ Q_A$	D_C	J_B	K_B	D_A	**HE $Q_C'\ Q_B'\ Q_A'$**
*E*0	0 0 0	1	0	1	1	**1 0 1**
*E*1	0 0 1	0	1	0	0	**0 1 0**
*E*2	0 1 0	0	0	1	1	**0 0 1**
*E*3	0 1 1	1	1	0	0	**1 1 0**
*E*4	1 0 0	1	1	1	0	**1 1 0**
*E*5	1 0 1	0	1	1	1	**0 1 1**
*E*6	1 1 0	0	1	1	0	**0 0 0**
*E*7	1 1 1	1	1	1	1	**1 0 1**

Taula hori erabiliz, egiaztatu ahal izango dugu ea zirkuituak jarraitutako sekuentzia diseinatzaileak lortu nahi zuena den ala ez; bertan topa baitezakegu, urratsez urrats, zein den zenbaki baten atzetik datorrena. Hala, lehenengo errenkada aztertuz, 0aren hurrengoa 5ekoa dela ondoriozta dezakegu; gero, 5aren atzetik, 3a dator; 3aren ondoren, 6a; eta, ondoren, berriro 0a.

Beraz, sekuentzia 0 – 5 – 3 – 6 – 0 – ... dela egiaztatu dugu, diseinatzaileak lortu nahi zuena, hain zuzen ere.

Hurrengo lana kronograma betetzea da. Aurretik betetako taula lagungarri izango dugu zeregin horretan. Izan ere, egoeren arteko trantsizio-taula baino ez da kronograma, modu grafikoan adierazita. Hau da sekuentziadoreari dagokion kronograma:



Hasieran, Q_C, Q_B eta Q_A 0 dira; ondorioz, $D_C = 1$, $J_B = 0$, $K_B = 1$ eta $D_A = 1$. Balio horiek ezagutuz, erloju-ertza gertatutakoan biegonkorren irteeretan edukiko ditugun balioak kalkula ditzakegu. $D_C = 1$ izanik, hurrengo zikloan $Q_C = 1$ izango da; $J_B = 0$ eta $K_B = 1$ direnez, erloju-ertza eta gero, $Q_B = 0$ izango da, eta D_A erloju-ertzaren aurretik 1 denez, erloju-ertza gertatu eta gero, $Q_A = 1$ izango da. Informazio hori guztia egoeren arteko trantsizio-taulako lehenengo lerroan ageri da, eta gauza bera gertatuko da gainerako zikloekin; alegia, taulako lerro jakin batekin bat etorriko direla. Beraz,

kalkulu guztiak errepikatu ordez, zuzenean taulako informazioa erabil dezakegu kronograma betetzeko.

Azkenik, sekuentzia bera egiten duen zirkuitu minimoa eraiki behar dugu. Horretarako, JK eta D biegonkorren sarreren funtzio logiko minimoak sortu behar ditugu.

Abiapuntu gisa, sekuentziadorearen egoeren arteko trantsizio guztiak taula batean bilduko ditugu. 3 biteko aukera guztiak ageri ez direnez, batzuk zehaztu gabeko gai moduan erabil daitezke:

Uneko Egoera $Q_C\,Q_B\,Q_A$		Hurrengo Egoera $Q_C'Q_B'Q_A'$	
0	0 0 0	5	1 0 1
1	0 0 1	–	– – –
2	0 1 0	–	– – –
3	0 1 1	6	1 1 0
4	1 0 0	–	– – –
5	1 0 1	3	0 1 1
6	1 1 0	0	0 0 0
7	1 1 1	–	– – –

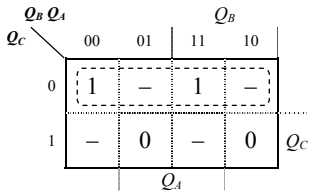
Taula hori oinarri gisa erabiliz, biegonkorren sarreretako balioak erabakiko ditugu. Gogoratu: D biegonkorren kasuan, irteeran nahi den balioa jarri behar da sarreran; JK biegonkorraren kasuan, berriz, biegonkorraren balioa hartu behar da kontuan *J* eta *K* sarreren balio egokiak erabakitzeko.

Hona hemen biegonkorren sarreren balioak kasu bakoitzean:

	U. Egoera $Q_C\,Q_B\,Q_A$	H. Egoera $\boldsymbol{Q_C'Q_B'Q_A'}$	$\boldsymbol{D_C}$	$\boldsymbol{J_B}$ $\boldsymbol{K_B}$	$\boldsymbol{D_A}$
*E*0	0 0 0	1 0 1	**1**	**0** –	**1**
*E*1	0 0 1	– – –	–	– –	–
*E*2	0 1 0	– – –	–	– –	–
*E*3	0 1 1	1 1 0	**1**	– **0**	**0**
*E*4	1 0 0	– – –	–	– –	–
*E*5	1 0 1	0 1 1	**0**	**1** –	**1**
*E*6	1 1 0	0 0 0	**0**	– **1**	**0**
*E*7	1 1 1	– – –	–	– –	–

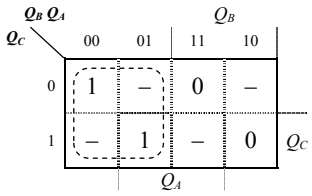
Taulako balioak erabiliz, biegonkorren sarreren adierazpen minimoak lortu behar ditugu. Horretarako, K-mapak erabiliko ditugu.

D_C



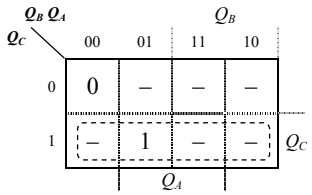
$$D_C = \overline{Q_C}$$

D_A



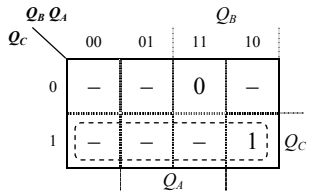
$$D_A = \overline{Q_B}$$

J_B



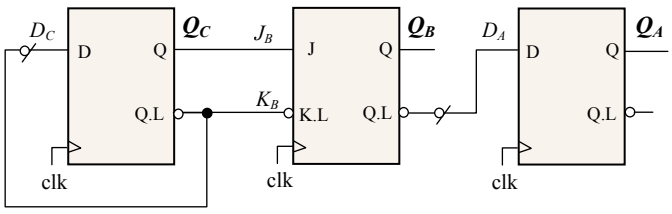
$$J_B = Q_C$$

K_B



$$K_B = Q_C$$

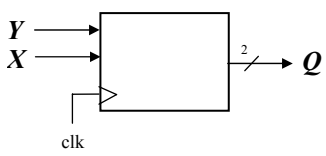
Biegonkorren sarreretarako lortu ditugun funtzioak aintzat izanik, nabarmena da aurreko soluzioa ez dela eraginkorrena, funtzio horiek erabiliz eraikitako zirkuitua aurrekoa baino askoz sinpleagoa baita.

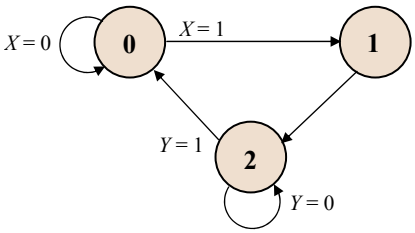


«

4.5. Ariketa

Garatu ezazu X eta Y kanpo-seinaleen mende dagoen sekuentziadore bat, irudiko grafoak adierazten duen sekuentzia sortzen duena.





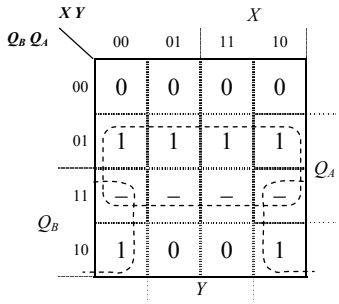
Ariketa honetan ere sekuentziadore bat eraiki behar da. Sekuentzian, 0, 1 eta 2 egoerak agertzen dira eta, beraz, nahikoak dira 2 bit (2 biegonkor) sekuentzia adierazteko. Enuntziatuan ez da zehazten biegonkorren mota; adibide honetan, esaterako, D biegonkorrak erabiliko ditugu. Adi! sekuentziadore honetan, sistemaren egoeraz gain, *X* eta *Y* kanpo-seinaleek parte hartzen dute.

Aurreko bi ariketak ebatzi ditugun modu berean ebatziko dugu hau ere. Taula bat egingo dugu, non adieraziko baitugu sistemaren egoera, erloju-ertza eta gero, sekuentziaren gainean eragina duten aldagai guztien konbinazio posible guztietarako: Q_B, Q_A, *X* eta *Y*.

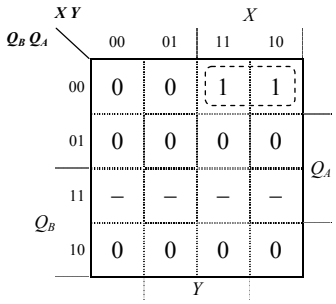
Zirkuitua eraikitzeko D biegonkorrak erabiliko ditugunez, D sarreretan kokatu beharko dira erloju-ertzaren ostean biegonkorren irteeretan izatea nahi ditugun balioak. Beraz, ez da beharrezkoa balio horiek taulan adieraztea, *Q*' balioak direlako.

UE		Kontrol-seinaleak		HE	
Q_B	Q_A	**X**	**Y**	Q_B'	Q_A'
0	0	0	–	0	0
		1	–	0	1
0	1	–	–	1	0
1	0	–	0	1	0
		–	1	0	0
1	1	–	–	–	–

Taula horretatik abiatuz, K-mapen bidez, biegonkorren D_B eta D_A sarreretarako funtzio minimoak lortuko ditugu. X eta Y kanpo-seinaleak eta Q_B eta Q_A (sistemaren egoera) dira funtzioen aldagaiak.

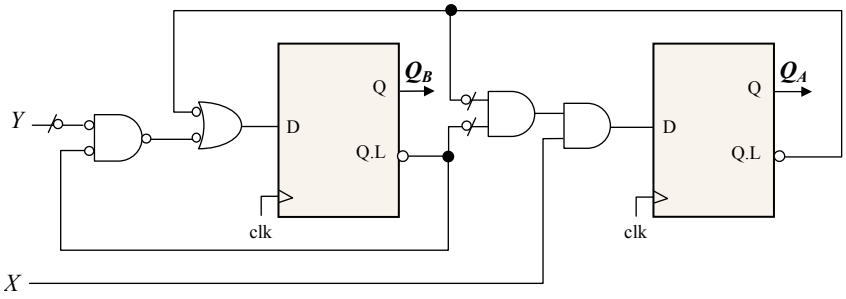


$$D_B = Q_A + Q_B \cdot \overline{Y}$$



$$D_A = \overline{Q_B} \cdot \overline{Q_A} \cdot X$$

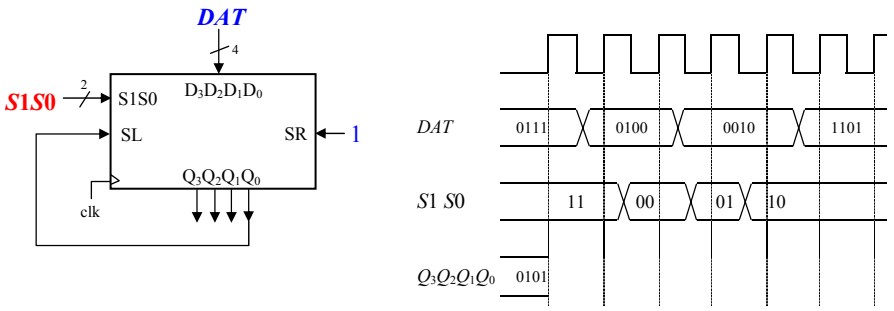
Funtzio horiek eraikitzeko aukera asko dago. Esaterako, hau izan daiteke horietako bat:



«

4.6. Ariketa

4 biteko desplazamendu-erregistroa ageri da irudian, 4.3.2. atalean azaldu duguna. Egin ezazu haren funtzionamendua islatzen duen kronograma.



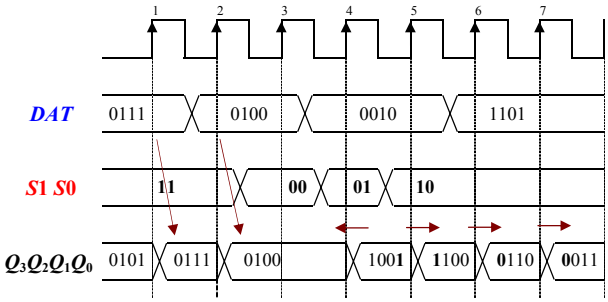
Bit bateko memoriak —biegonkorrak— landu ditugu aurreko ariketetan. Ariketa honetan, desplazamendu-erregistro baten portaera islatzen duen kronograma egin behar dugu. Desplazamendu-erregistro mota asko dago, baina erabilienak 4 eragiketa egiteko aukera ematen du: datuak kargatu eta mantendu (erregistro guztiak bezala), eta metatutako datua desplazatu, ezkerrerantz zein eskuinerantz (hortik datorkio izena). 4 eragiketa horiek kodetzeko, bi biteko eragiketa-kodea erabili ohi da —*S*1 eta *S*0—, taula honetan ageri den moduan:

***S*1**	***S*0**	Eragiketa
0	0	datua mantendu
0	**1**	← bit bateko desplazamendua ezkerrera
1	0	→ bit bateko desplazamendua eskuinera
1	**1**	*n* biteko datu bat kargatu

Desplazamendu-erregistroa zirkuitu sinkronoa da, eta beraz, edozein eragiketa egiteko, beharrezkoa da erloju-seinalea: erloju-ertza aktibatzean prozesatuko dira *S*1 eta *S*0 kontrol-seinaleak.

Bit bateko desplazamenduak egiten direnean hutsik geratuko da posizio bat, ezkerrean (desplazamendua eskuinera denean) edo eskuinean (desplazamendua ezkerrera egiten denean), eta hor kanpoko bit bat kargatuko da: SR sarrerakoa eskuinekoan, edo SL sarrerakoa ezkerrekoan.

Azalpen horiekin, nahikoa informazio badugu kronograma betetzeko.



Lehenengo bi erloju-ertzetan, $S1S0$ kontrol-seinaleek 11 balioa dute eta, ondorioz, erregistroaren sarreran dagoen datua kargatuko da: lehenengo erloju-ertzarekin batera, 0111 datua, eta, bigarrenean, 0100 datua.

Hirugarren erloju-ertzean, $S1S0$ kontrol-seinaleek 00 balioa dute. Kasu horretan, beraz, erregistroaren edukia mantendu egingo da: 0100.

Laugarren erloju-ertzean, $S1S0$ kontrol-seinaleek bit bateko desplazamendua ezkerrerantz agintzen dute: 01. Erregistroaren edukia posizio bat desplazatuko da ezkerrera; gainera, eskuineko bitean (pisu txikieneko posizioan), desplazamendu-erregistroaren SR sarreran dagoen bita kargatuko da; adibide honetan, 1 konstantea. Hala, erregistroaren edukia 0100 izatetik 1001 izatera pasatuko da.

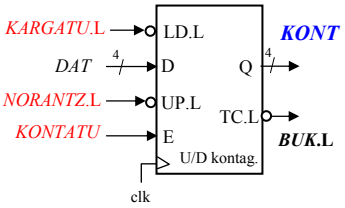
Hurrengo erloju-ertzetan, $S1S0$ kontrol-seinaleek bit bateko desplazamendua eskuinera agintzen dute: 10. Hala, erregistroaren edukia posizio bat desplazatuko da eskuinera, eta, horrekin batera, ezkerreko (pisu handieneko) posizioan, SL sarreran dagoen bita kargatuko da; adibide honetan, erregistroaren Q_0 bita eraman da SL sarrerara, hau da, edukiaren bit bateko biraketa egiten da. Horregatik, erregistroaren edukia 1001etik 1100ra pasatuko da erloju-ertz horretan; eta 1100tik 0110ra eta 0110tik 0011era hurrengoetan.

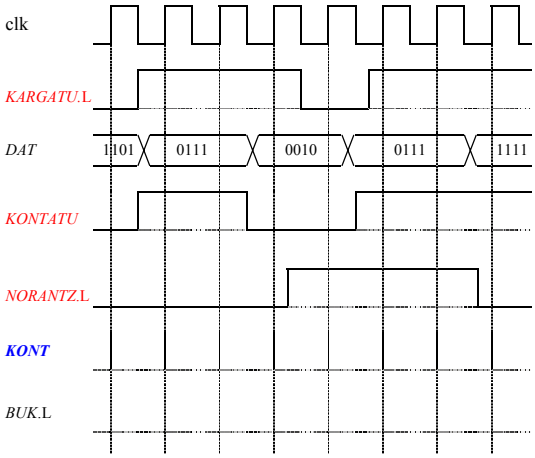
(Adi! datu-sarrera aldatzen bada ere 3. ziklotik aurrera, aldaketa horiek ez dute eraginik, desplazamenduak egin behar direlako eta ez datu berrien karga.)

«

4.7. Ariketa

4 biteko kontagailu bat ageri da irudian, gorantz eta beherantz konta dezakeena (up/down), 4.3.3. atalean azaldu duguna. Egin ezazu haren funtzionamendua islatzen duen kronograma.





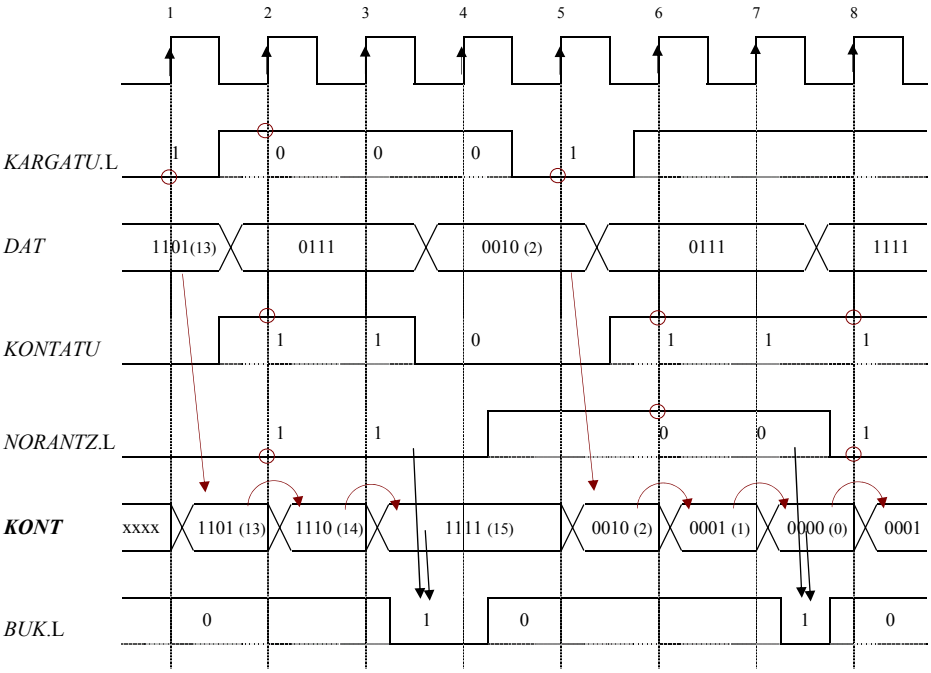
Aurreko ariketan desplazamendu-erregistroaren funtzionamendua aztertu dugu eta, honetan, kontagailuarena landuko dugu.

Ariketa honetako kontagailuak honako kontrol-seinale sinkrono hauek ditu:

- LD (*load*): datu-sarreran dagoen datua kontagailuan kargatzeko.
- E (*enable*): edukiari unitate bat gehitzeko edo kentzeko, hau da, kontatzeko.
- UP: kontaketaren noranzkoa adierazteko (E seinalearekin batera): 1 bada, +1 eragiketa beteko da, eta 0 bada, –1 eragiketa.

LD eta E seinaleak batera aktibatuta badaude, karga-eragiketa beteko da; hau da, LD seinaleak lehentasuna du.

Bi irteera ditu irudiko kontagailuak: kontagailuaren edukia (4 bit adibide honetan), eta TC seinalea. TC seinaleak kontaketaren amaiera adierazten du, kontaketaren noranzkoaren arabera, hau da: $TC = (Q = 15) \cdot UP + (Q = 0) \cdot \overline{UP}$. Beraz, adi! TC seinalea irteera baten (Q) eta sarrera baten (UP) mende dago.



1. erloju-ertza iristen denean, *KARGATU*.L seinalea aktibatuta dago eta, beraz, kargatu egingo da kontagailuan datu-sarrerako balioa: 1101, hain zuzen ere.

2. erloju-ertzean, berriz, *KARGATU*.L seinalea desaktibatuta dago eta *KONTATU* aktibatuta; *NORANTZ*.L seinalea 1 denez, kontagailuaren edukia "inkrementatu" egingo da: 1101etik 1110ra.

3. erloju-ertzean, eragiketa bera errepikatuko da, kontrol-seinaleen balioak berdinak direlako. Beraz, kontagailua 1110tik 1111 izatera pasatuko da.

4. erloju-ertzean, LD (kargatu) zein E (kontatu) kontrol-seinaleak desaktibatuta daude; ondorioz, ez da aldatuko kontagailuaren balioa.

5. erloju-ertza heltzean, berriro ere *KARGATU*.L seinalea aktibatuta dago eta, beraz, datu-sarrerako balioa kargatuko da kontagailuan: 0010, hain zuzen ere. Une honetatik aurrera, desaktibatuta mantenduko da *KARGATU*.L seinalea kronograma osoan zehar; beraz, ez da berriro datu-sarrerako balioa kontagailuan gordeko.

6. eta 7. erloju-ertzetan, eragiketa bera beteko da; *KONTATU* = 1 eta *NORANTZ*.L = 0 direnez, kontagailuaren edukia "dekrementatu" egingo da: 0010tik 0001era eta 0001etik 0000ra.

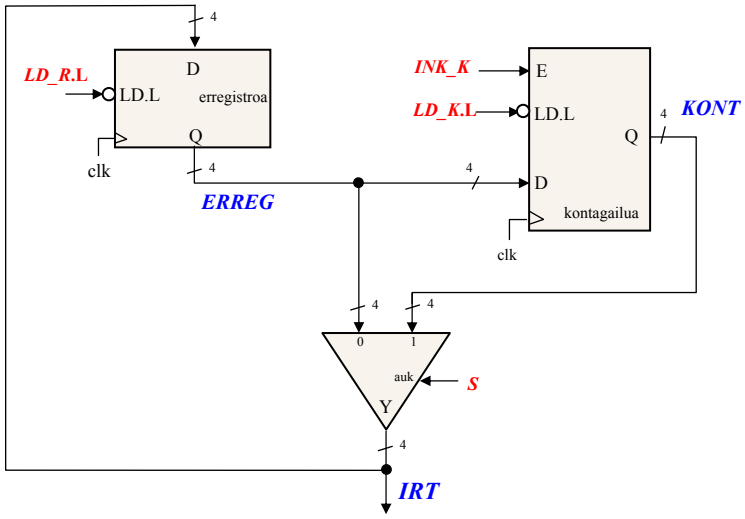
8. erloju-ertzean, kontaketarekin jarraituko da, baina orain gorantz: 0000tik 0001era.

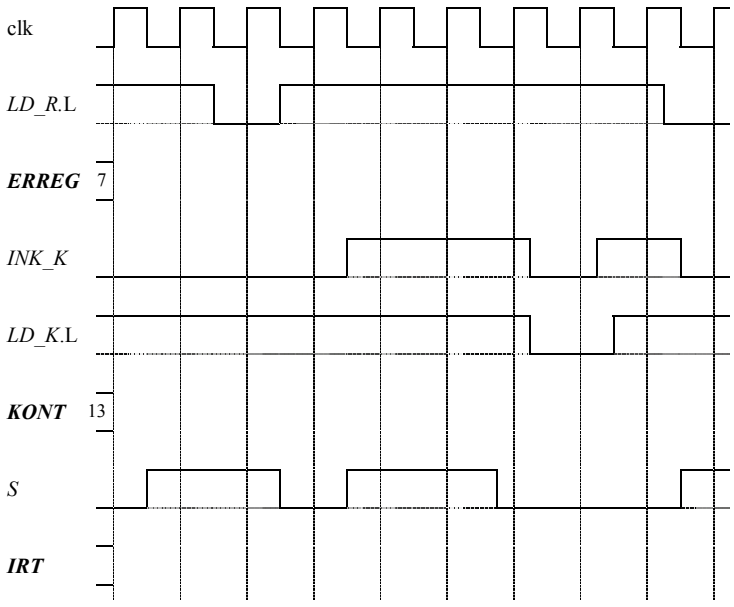
Lehen esan dugun moduan, TC (edo RCO) irteerak kontaketaren mugak adierazten ditu: 4 biteko kontagailuetan, Q = 1111, UP sarrera 1 denean, eta Q = 0000, UP sarrera 0 denean. Kronograman ageri den moduan, bi bider aktibatuko da *BUK* seinalea (TC irteera): kontagailuaren edukia 15 denean eta *NORANTZ*.L seinalea 1 denean (3. zikloa) eta edukia 0 eta *NORANTZ*.L 0 direnean (7. zikloa).

«

4.8. Ariketa

4 biteko erregistro bat, kontagailu bat eta multiplexore bat erabiltzen ditu irudiko zirkuituak. Bete ezazu haren portaera islatzen duen kronograma.



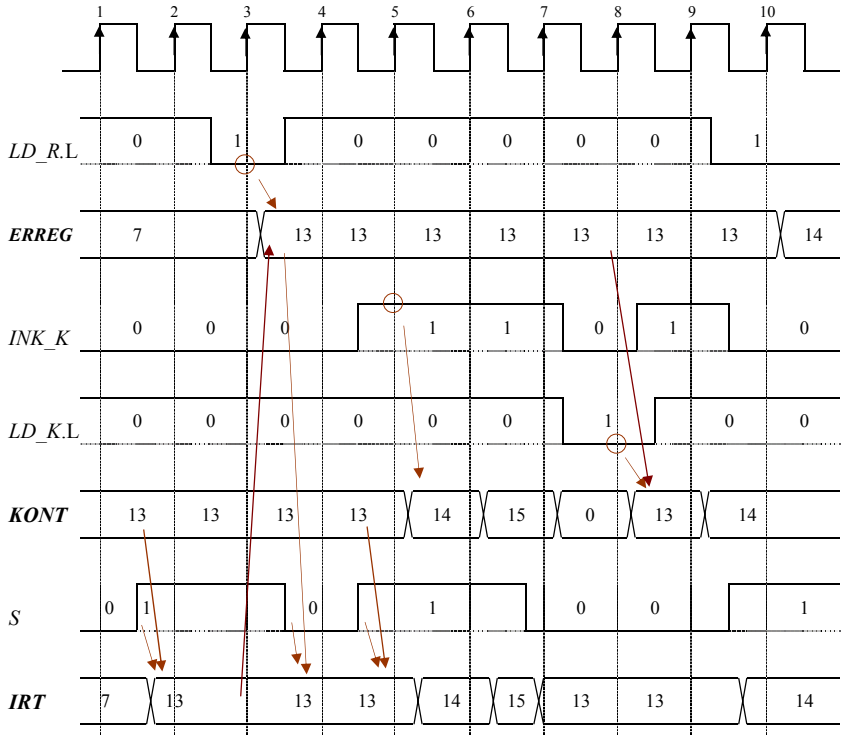


Bi motatako gailuak erabiltzen dira ariketa honetako zirkuituan: sekuentzialak —erregistroa eta kontagailua— eta konbinazionalak —multiplexorea—. Kronograma egiterakoan, zirkuitu bakoitzaren izaera kontuan eduki eta bi portaera nagusi bereizi beharko ditugu: erregistroan eta kontagailuan, zirkuitu sinkronoetan, kontrol-seinaleak erloju-ertza gertatzen denean soilik prozesatuko dira, baina, multiplexorean, zirkuitu konbinazionalean, sarrerak aldatzen diren unean gertatuko dira aldaketak irteeran. Kasu batean zein bestean, ezin dugu ahaztu zirkuituek denbora-tartetxo bat behar dutela erantzuteko, eta kronograman islatuko dugu.

Honela funtzionatzen dute zirkuitu honetako gailuek:

- Erregistroa: datu bat kargatzen du *LD_R* seinalea aktibatuta badago (erloju-ertzean).
- Kontagailua: datu bat kargatzen du (*LD_K*) edo edukia inkrementatzen du (*INK_K*) erloju-ertza heltzean.
- Multiplexorea: sarreretako datu bat aukeratu eta irteerara eramaten du, *S* seinalearen arabera.

Ikus dezagun nola egin kronograma.



Zikloz ziklo gertatzen dena aztertzen hasi aurretik, kronogramako kontrol-seinaleei begirada bat ematea lagungarria da. Erregistroaren edukia 3. eta 10. erloju-ertzak gertatu eta gero aldatuko da soilik, *LD_R* bi erloju-ertz horietan bakarrik baitago aktibatuta. Kontagailuaren edukia, berriz, 5., 6., 7., 8. eta 9. erloju-ertzak gertatu eta gero aldatuko da; 8.ean, datu-sarrerako balioa kargatuko da, eta, gainerakoetan, inkrementatu egingo da. Multiplexorearen irteera, aldiz, gehiagotan aldatuko da, *S* eta erregistroaren eta kontagailuaren edukien arabera (adi! erlojuak ez du eraginik multiplexorean).

Azter dezagun kronograma, zikloz ziklo.

1. erloju-ertzean, erregistroaren karga-seinalea eta kontagailuaren kargatzeko eta kontatzeko seinaleak 0 dira. Beraz, bien edukia mantenduko da. Multiplexoreko aukeratze-seinalea 0 denez, erregistroaren edukia izango da haren irteeran, 7ko bat, hain zuzen ere.

2. erloju-ertza gertatu baino lehen, *S* kontrol-seinalea 0tik 1era pasatzen da eta, beraz, multiplexorearen irteera ere (*IRT*) aldatuko da, orain kontagailuan dagoen balioa aukeratuko baita: 13a, kasu konkretu honetan.

3. erloju-ertza iristen denean, *LD_R* seinalea aktibatuta dago, eta, ondorioz, datu bat kargatuko da erregistroan: multiplexorearen irteeran dagoena, 13, hain zuzen ere. Hori dela eta, 4. erloju-ertza baino lehen *S* seinalea aldatu egiten den arren, multiplexorearen irteerako balioa ez da aldatuko, une horretan berdinak baitira erregistroaren eta kontagailuaren edukiak.

5. erloju-ertzean, *INK_K* seinalea aktibatuta dago, eta, ondorioz, kontagailua inkrementatu egingo da, 13tik 14ra. Aldaketa hori dela eta, multiplexorearen irteera ere aldatuko da, kontagailuaren edukia aukeratuta baitago irteeran ($S = 1$ baita).

6. erloju-ertzean, aurreko eragiketa errepikatzen da; ondorioz, kontagailua eta multiplexorearen irteera 14tik 15era aldatuko dira. Zikloaren amaieran, *S* kontrol-seinalea 1etik 0ra pasatzen da, eta, ondorioz, *IRT*, multiplexorearen emaitza, ere aldatu egingo da: erregistroan dagoen balioa (13) hartuko du.

7. erloju-ertza gertatzen denean, berriro ere, kontatzeko seinalea aktibatuta dago eta, beraz, kontagailua inkrementatu egingo da. Une horretan, kontagailuaren edukia 15 denez, 4 bitekin adieraz daitekeen zenbaki handiena, 15etik 0ra pasatuko da: $1111 + 1 = (1)\ 0000$.

8. erloju-ertzean, *LD_K* seinalea aktibatuta dago, eta, beraz, datu bat kargatuko da kontagailuan, datu-sarrerakoa: erregistroaren edukia (13).

9. erloju-ertzean, $INK\_K = 1$ da, eta, beraz, kontagailuaren edukia gehituko da (13tik 14ra). 9. zikloan zehar, *S* seinalea 0tik 1era pasatzen da eta, beraz,

multiplexorearen irteera ere aldatu egingo da, kontagailuaren irteerako balioa hartuko du, 14, hain zuzen ere.

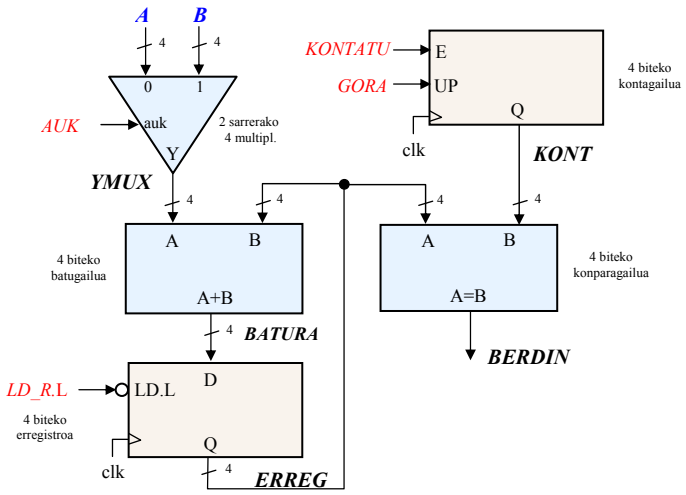
Azkenik, 10. erloju-ertzean, *LD_R* seinalea aktibatuta dago: multiplexorearen irteerako balioa kargatuko da erregistroan (14).

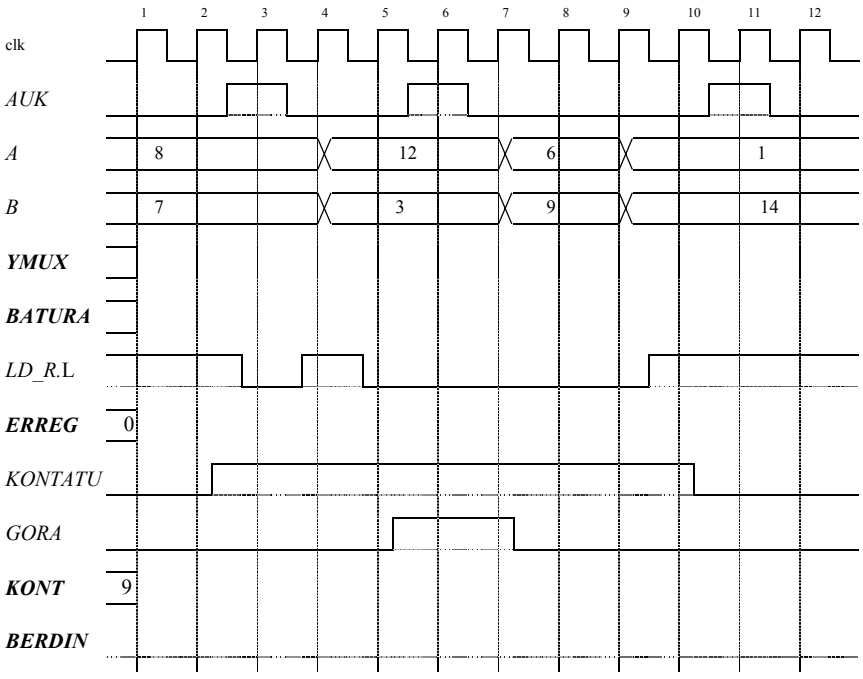
Adibide honetan, kronograma egitean, zirkuitu sinkrono zein konbinazionalen atzerapenak islatu ditugu. Aldaketak erloju-ertza gertatu baino pixka bat geroago gertatu dira zirkuitu sinkronoetan, eta sarrerak aldatu baino pixka bat geroago zirkuitu konbinazionaletan.

«

4.9. Ariketa

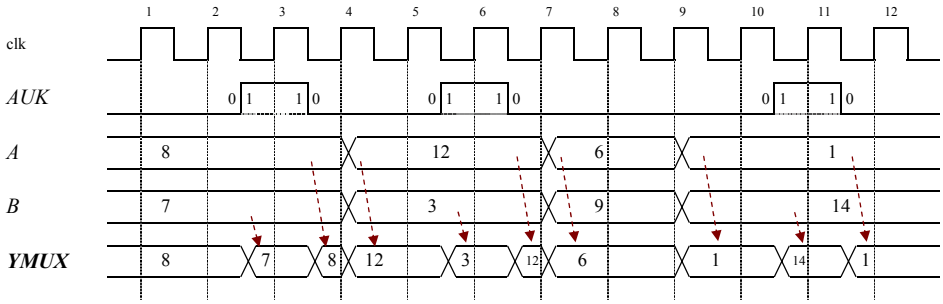
4 biteko erregistro, kontagailu, multiplexore, batugailu eta konparagailu bana erabiltzen ditu irudiko zirkuituak. Bete ezazu haren portaera islatzen duen kronograma (hasierako balioak: ERREG = 0, KONT = 9).





Kronogramari ekin baino lehen, ohartxo bat. Kronograma sinplifikatzearren, ez ditugu batuko zirkuituen erantzun-denborak, bata bestearen atzetik datozenean.

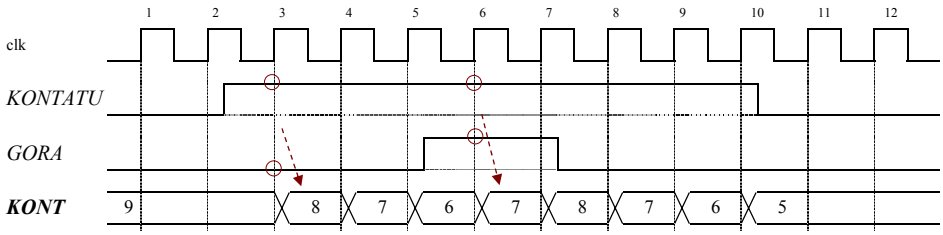
Kronograma betetzen hasi aurretik interesgarria da zirkuitua aztertzea eta atal "independenteak" identifikatzea, zirkuituaren analisia sinplifikatzeko. Adibide honetan, esaterako, multiplexorearen irteerako datua *AUK* hautatze-lerroaren eta *A* eta *B* sarrera-datuen mende dago soilik. *AUK* = 0 denean, *A* sarrerako datua izango da irteeran, eta *AUK* = 1 denean, berriz, *B* sarrerakoa. Portaera hori 4.18. irudian ageri da.



4.18. irudia. 4.9. ariketako multiplexorearen kronograma.

2. zikloan, *AUK* 0tik 1era aldatzen denean, *YMUX* 8tik 7ra pasatuko da (*B* sarrera). 3. zikloan, *AUK* 1etik 0ra aldatzen denean, *YMUX* irteeran berriro ere *A* sarrera, 8koa, izango dugu. Modu berean analiza daitezke *YMUX* irteeraren aldaketak denboran zehar.

Kontagailuaren eboluzioa ere bere kasa analiza daiteke, kontatze-seinalearen (eta noranzkoaren) mende baitago soilik. Hasieran, *KONT* = 9 da, eta kontagailuaren edukia aldatuko da *KONTATU* = 1 den ziklo guztietan: gorantz 6. eta 7. erloju-ertzetan (*GORA* = 1 delako), eta beherantz 3., 4., 5., 8., 9., eta 10. erloju-ertzetan (*GORA* = 0 delako). Portaera hori 4.19. irudian ageri da.



4.19. irudia. 4.9. ariketako kontagailuaren kronograma.

Bete ditzagun orain, zikloz ziklo, kronogramako gainerako seinaleak (ikus 4.20. irudia).

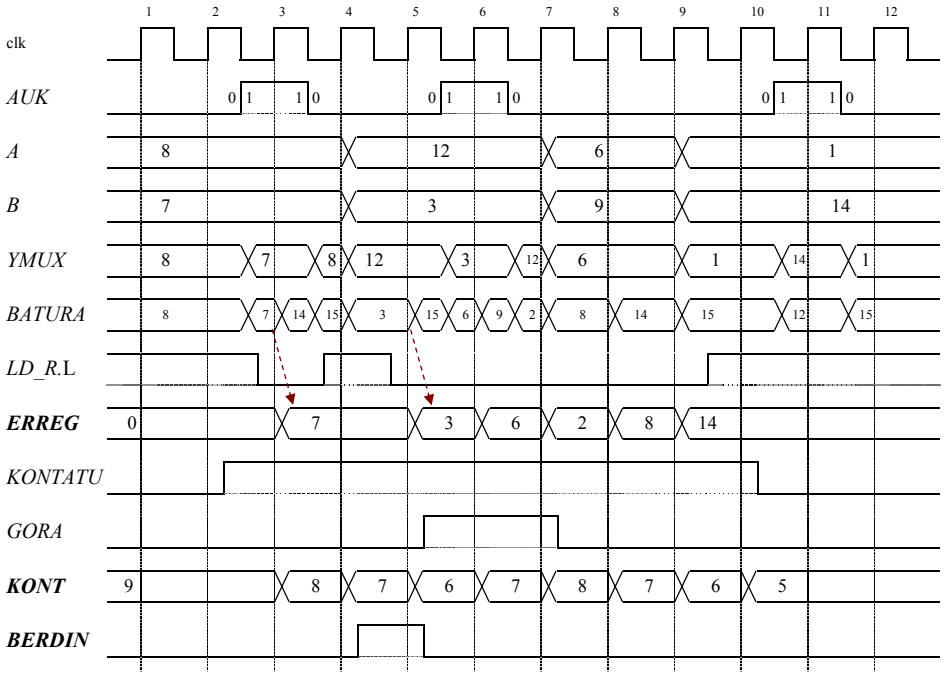
Batugailuaren eta konparagailuaren irteerak sarrerak aldatzen diren uneetan aldatuko dira; batugailuaren kasuan, multiplexorearen irteera edo erregistroaren irteera aldatzean (haien edukiak batzen baititu) eta, konparagailuaren kasuan, erregistroaren irteera edo kontagailuaren irteera aldatzen direnean. Erregistroaren irteera, berriz, erloju-ertza iritsi eta gero aldatuko da, *LD_R* seinaleak hala adierazten badu, jakina. *LD_R* seinalea aktibatuta badago erloju-ertzean, batugailuaren edukia gordeko da erregistroan.

Lehenengo bi erloju-zikloetan, erregistroaren edukia mantendu egiten da, 0 balioarekin. Hori dela eta, batugailuaren emaitza multiplexorearen irteera aldatzen denean soilik aldatuko da eta, gainera, hartuko duen balioa multiplexorearenarekin bat etorriko da.

3. erloju-ertzean, *LD_R* seinalea aktibatuta dagoenez, batugailuaren irteerako balioa erregistroan gordeko da, 7, hain zuzen ere. Erregistroaren edukia aldatzearekin batera, batugailuarena aldatuko da: 7 + 7 = 14 izatera pasatuko da. 3. erloju-zikloan zehar, multiplexorearen irteera aldatu egiten da 7tik 8ra, eta, beraz, batugailuarena ere aldatuko da, 7 + 8 = 15. Gainerako erloju-zikloetan, portaera bera errepikatuko da batugailuan zein erregistroan (adi! 4 biteko batuketak egiten dira; esaterako, 4. zikloan, 12 + 7 = 3 da, hau da, 1100 + 0111 = (1) 0011).

Azkenik, *BERDIN* irteeraren portaera aipatzea falta da. Kontagailuaren eta erregistroaren edukiak konparatzearen emaitza adieraziko digu honek. Bi balioak berdinak direnean, 1eko balioa hartuko du, eta bestela 0koa.

Kronograma honetan, 4. zikloan soilik dira berdinak erregistroaren edukia eta kontagailuarena. Tarte horretan, 7 balio dute biek, eta, beraz, *BERDIN* seinalea aktibatu egingo da. Gainerako uneetan, 0 balioko du *BERDIN* seinaleak.

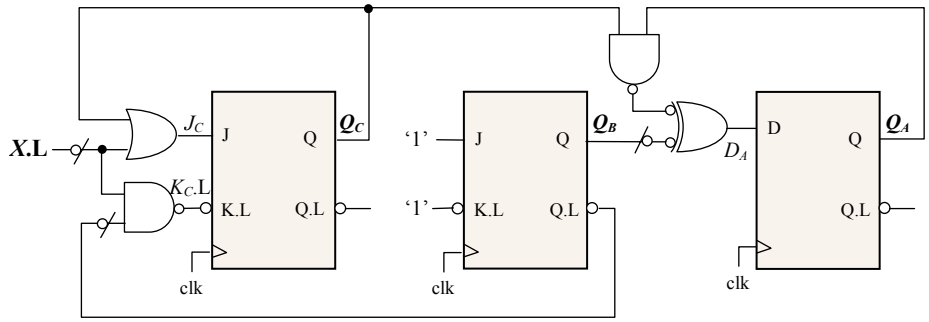


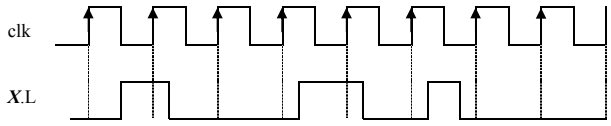
4.20. irudia. 4.9. ariketako sistemaren funtzionamenduaren kronograma osoa.

«

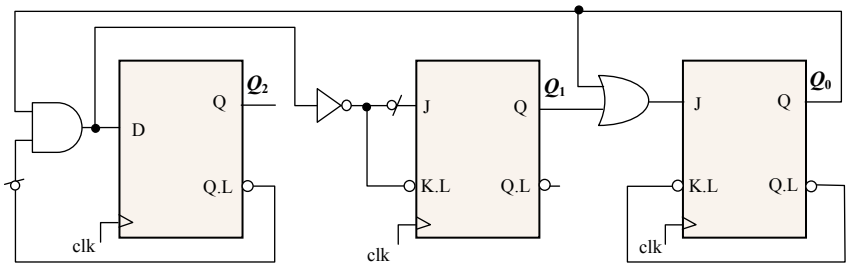
4.5. ARIKETAK

4.1. Hartu kontuan irudiko zirkuitua, eta adieraz itzazu J_C, K_C eta D_A funtzio logikoak. X seinalearen eboluzioaren arabera, marraz ezazu, kronograma batean, sistemaren portaera: J_C, K_C eta Q_C; Q_B; eta D_A eta Q_A seinaleen aldaketak denboran zehar. Biegonkorren hasiera-balioak 0 dira.

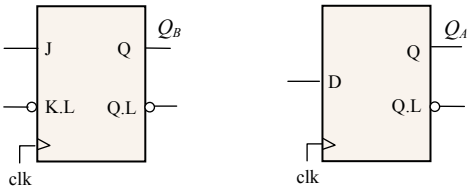




4.2. Irudiko zirkuitua 3 biteko sekuentziadorea da. Hiru biegonkorren irteerek sekuentziadorearen egoera adierazten dute.
Adieraz itzazu biegonkorren *J, K* eta *D* sarrerei dagozkien funtzio logikoak eta egin ezazu sistemaren funtzionamendua islatzen duen kronograma bat (hasieran, $Q_2Q_1Q_0 = 000$). Zein sekuentziari jarraitzen dio zirkuituak?



4.3. Erabil itzazu JK biegonkor bat eta D biegonkor bat 0 – 1 – 2 – 3 – 0 – ... sekuentziari jarraitzen dion zirkuitu bat eraikitzeko.

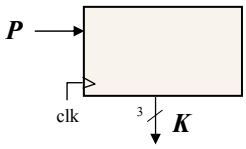


4.4. 3 biteko kontagailu batek bi modutan funtzionatzen du, *P* kontrol-seinale baten arabera. *P* = 1 denean, kontagailua hurrengo zenbaki bikoitira aldatuko da; 0 denean, aldiz, hurrengo zenbaki bakoitira. Adibidez:

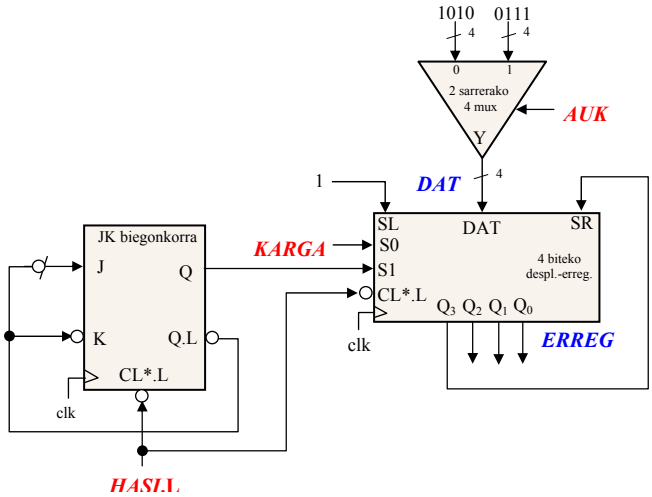
P: 0 1

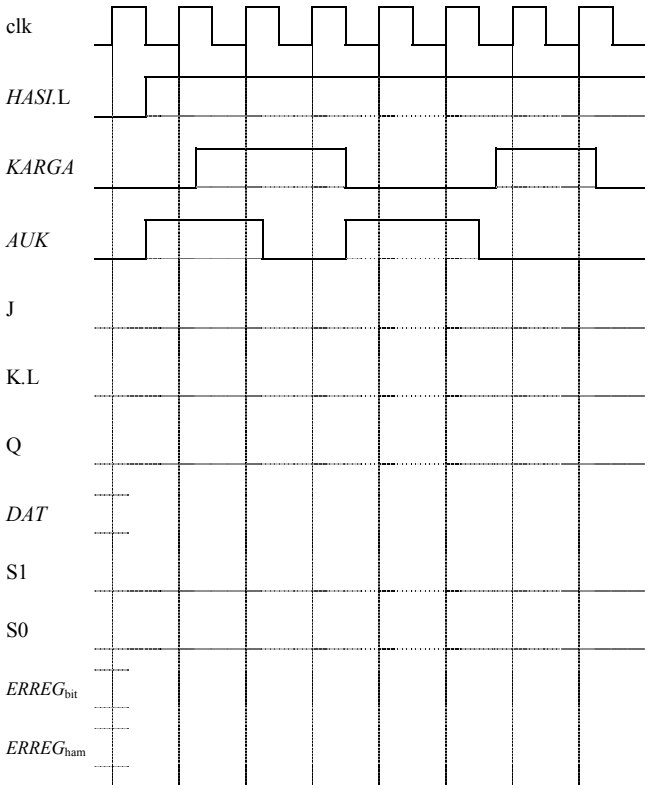
K: 1 - 3 - 5 - 7 - 1 - 2 - 4 - 6 - ...

Eraiki ezazu kontagailu hori, pisu handieneko 2 bitetarako JK biegonkorrak erabiliz, eta pisu txikieneko biterako D biegonkor bat erabiliz.



4.5. 4 biteko desplazamendu-erregistro bat, multiplexore bat eta JK biegonkor bat erabiltzen ditu irudiko zirkuituak. Bete ezazu haren portaera islatzen duen kronograma. Adierazi erregistroaren edukia bitarrez zein hamartarrez.




clk
HASI.L
KARGA
AUK
J
K.L
Q
DAT
S1
S0
ERREG$_{bit}$
ERREG$_{ham}$

5. kapitulua

MEMORIAK

Aurreko kapituluetan, oinarrizko bloke konbinazionalak eta sekuentzialak aztertu ditugu: multiplexoreak, deskodegailuak, batugailuak, biegonkorrak, kontagailuak, eta abar. Gailu horiek erabiliz, oso helburu desberdineko sistema digitalak eraiki daitezke. Aztertu ditugun funtzio logikoez gain, ohikoa da datuak edo informazioa gorde behar izatea sistema digitaletan. Gorde behar den informazio kopurua txikia bada (bit batzuk) erregistroak erabil daitezke, baina informazio kopurua handia denean memoriak erabili behar dira.

Memoria mota asko dago, informazioa gordetzeko erabiltzen den teknologiaren arabera. Kapitulu honetan, sistema digital gehienetan (eta, batik bat, konputagailuetan) erabiltzen diren memoria mota nagusiak aztertuko ditugu: RAM eta ROM memoriak. Memoria horien egitura eta funtzionamendua aztertu baino lehen, datuak gordetzeko konputagailuetan ohikoa den osagaia diseinatuko dugu: erregistro-multzoa.

5.1. SARRERA

Sistema digitalek informazioa —datuak, aginduak...— prozesatzen dute, eta, beraz, informazio hori gorde egin behar dute. Informazioa gordetzeko, erregistroak, erregistro-multzoak eta, oro har, memoriak erabiltzen dira. Erregistroetan eta erregistro-multzoetan, datu bakan batzuk gordetzen dira: eragiketa baten tarteko emaitzak, aginduen eragigaiak, eta abar. Baina informazio kopuru handiak gordetzeko, memoriak erabili behar dira. Memorian gordetzen dira, esaterako, prozesadoreetan exekutatu behar diren programak (erabiltzailearenak zein sistemarenak) eta haiek exekutatzeko behar diren datuak.

Memorian gordetzen den informazioa “hitzez hitz” antolatzen da; **hitz** bat *n* biteko datu bat da. Memoriako hitz bakoitzari (memoria-posizio bakoitzari) **helbide** bat dagokio, eta helbidea erabili behar da ohiko memoria-eragiketak egiteko: **irakurketa** eta **idazketa.** Zenbait memoria irakurri baino ezin dira egin —ROM memoriak—, baina, oro har, memoria baten edukia irakur eta idatz daiteke —RAM memoriak—.

Memorien parametro nagusien artean, bi hauek ditugu: **edukiera** eta **erantzun-denbora**. Edukierak adierazten du zenbat bit gorde dezakeen memoriak (adibidez, 16 megabyte), eta erantzun-denborak (atzipen-denbora) zehazten du zenbat denbora behar den eragiketa bat (irakurketa edo idazketa) egiteko (esaterako, 50 ns). Oro har, memoriak zenbat eta handiagoak izan, hainbat eta motelagoak dira, hau da, erantzun-denborak luzeagoak dira. Hori dela eta, erantzun-denbora garrantzitsua denean (konputagailuetan, adibidez), memoria-hierarkia bat eraikitzen da. Azkar eta maiz eskuratu behar diren datuak memoria txikietan gordetzen dira, erantzun azkarrekoak, eta informazio orokorra edo batzuetan bakarrik erabiliko dena, memoria handiagoetan (eskuarki motelagoak). Adibidez, honela antolatzen da konputagailu baten memoria-sistema:

- erregistroak: azkarrenak, baino edukiera txikikoak (hitz batzuk)
- cache memoria: tarteko abiadura eta edukiera (milioi bat hitz inguru)
- memoria nagusia: hiruretan motelena, baina edukiera handienekoa (mila milioi hitz inguru)

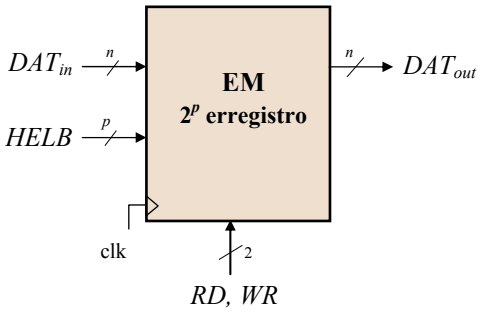
Hiru memoria mota horiek erdieroalezko memoriak dira, liburu honetan aztertzen ari garen gainerako zirkuituak bezala. Edukiera handiagoko memoriek, eskuarki, beste teknologia batzuk erabiltzen dituzte —optikoak, magnetikoak...— eta ez ditugu liburu honetan aztertuko.

5.2. ERREGISTRO-MULTZOAK

Aurreko kapituluan aztertu dugun moduan, erregistro batean *n* biteko datu bat gorde daiteke (esaterako, *n* = 64 bit). Hainbat sistema digitaletan, datu bat baino gehiagorekin lan egin behar da. Datu kopurua ez bada oso handia (esaterako, 32 datu), erregistro-multzo bat erabil daiteke; bestela, kopurua handia bada, memoria bat erabili beharko dugu.

Adibidez, erregistro-multzoak beharrezkoak dira konputagailuetan programa baten aginduak exekutatzeko prozesuan, eragiketak ahalik eta azkarren egiteko; izan ere, erregistroetan egon behar dute eragiketa baten datuek, eta erregistro batean gordeko da emaitza. Bestela, datuok memoria nagusitik eskuratu beharko lirateke behin eta berriz, eta, zoritxarrez, memoriak ohi dira zirkuitu digital motelenak.

Erregistro-multzo bat, berez, memoria txiki bat da, eta memorietan bezala egituratzen da informazioa. Erregistro-multzoa osatzeko, hainbat erregistro elkartzen dira. Bi eragiketa egin daitezke erregistro-multzo batekin: erregistro baten edukia irakurri, eta datu bat idatzi (kargatu) erregistro batean. Eskuarki, bi kontrol-seinale izango ditugu eragiketa horiek adierazteko: WR (*write*, idatzi) eta RD (*read*, irakurri). Erregistroei helbide eta izen bana esleitzen zaizkie; esaterako, 7 helbidea, R7 erregistroa. Zein erregistro irakurri edo idatzi nahi den adierazteko, haren helbidea erabili behar da. 5.1. irudian, erregistro-multzo baten eskema logiko orokorra ageri da.

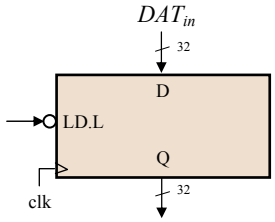


RD	*WR*	**Eragiketa**
0	0	informazioa mantendu
0	1	erregistro bat idatzi `EM[HELB] := DAT`$_{in}$
1	0	erregistro bat irakurri `DAT`$_{out}$ `:= EM[HELB]`

5.1. irudia. Erregistro-multzo baten eskema logiko orokorra. Erregistro sinkronoak erabiltzen badira, erloju-ertzetan bakarrik exekutatuko da idazketa.

Erregistro-multzo bat osatzeko, dagoeneko ezagutzen ditugun gailuak erabili behar dira: erregistroak, deskodegailu bat eta multiplexore bat. Adibide gisa, 32 biteko 4 erregistroko erregistro-multzo bat egingo dugu.

Erregistro-multzo hori osatzeko, 4 erregistro erabili behar ditugu. Bakoitzaren datu-sarrerara, erregistro-multzoan idatzi nahi den datua eramango dugu, DAT_{in} (bakar batean idatziko badugu ere).

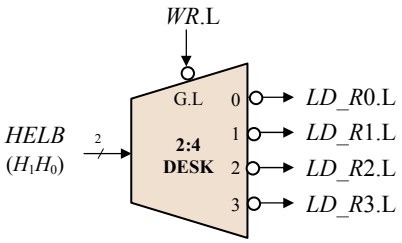


Erregistro jakin bat adierazteko (atzitzeko), haren helbidea eman behar dugu. 4 erregistro izango ditugunez, 2 biteko helbide bat beharko dugu:

$H_1H_0 = 00 \rightarrow$ R0 $\qquad$ $H_1H_0 = 10 \rightarrow$ R2

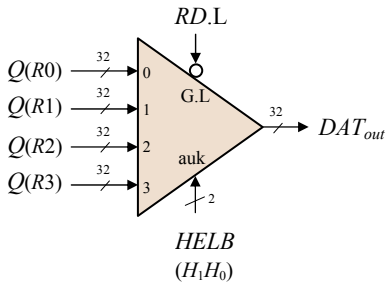
$H_1H_0 = 01 \rightarrow$ R1 $\qquad$ $H_1H_0 = 11 \rightarrow$ R3

Esaterako, R2 erregistroan idatzi nahi badugu, 10 helbidea adierazi beharko dugu eta *WR* seinalea aktibatu. Informazio horrekin gauza izan behar dugu R2 erregistroaren LD sarrera aktibatzeko. Eta horretarako oso egokia da 2:4 tamainako deskodegailu bat, modu honetan erabilita:



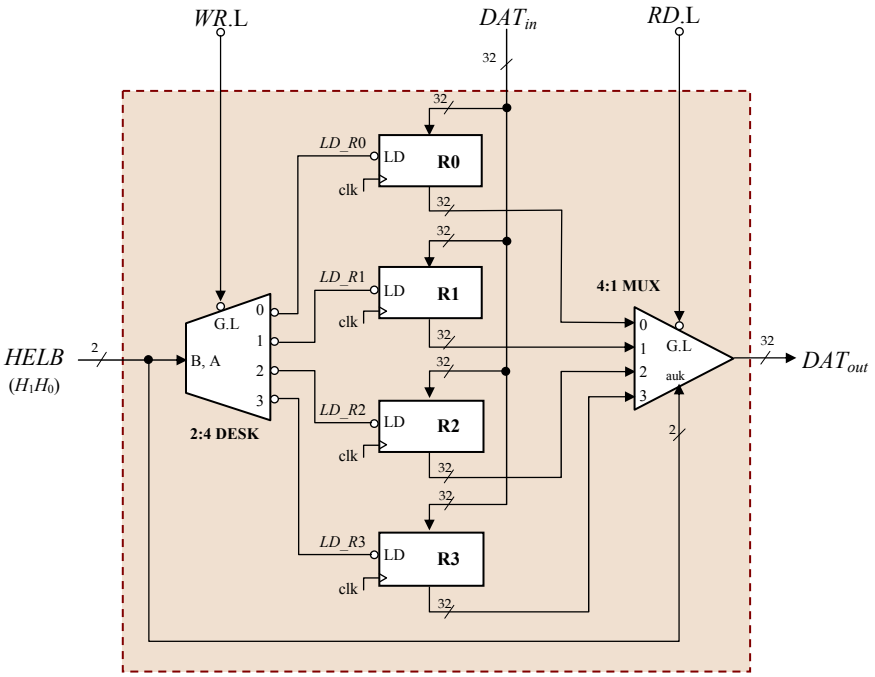
WR = 0 bada, deskodegailua desaktibatuta dago, eta, ondorioz, lau irteerak (erregistroen karga-seinaleak) 0 dira. Idazketa bat egin nahi denean, ordea, *WR* = 1 ezarri behar da; deskodegailua gaituta dago, eta irteera jakin bat aktibatuko da, sarrerako helbideari dagokiona; beraz, erregistro horretan kargatuko da datua.

Erregistro baten edukia irakurtzeko ere haren helbidea eman behar dugu. Helbide horren bidez aukeratu ahal izango dugu lau erregistroen irteeren artean interesatzen zaiguna. Beraz, multiplexore bat jarri beharko dugu, erregistroen irteera jakin bat aukeratzeko, eta helbidea erabili hautatze-seinale gisa.



RD = 0 denean, multiplexorearen irteera desaktibatuta dago; *RD* = 1 denean, ordea, erregistro jakin baten edukia ageriko da irteeran, helbideak adierazitakoa, hain zuzen ere.

5.2. irudian ageri da erregistro-multzoaren eskema osoa.

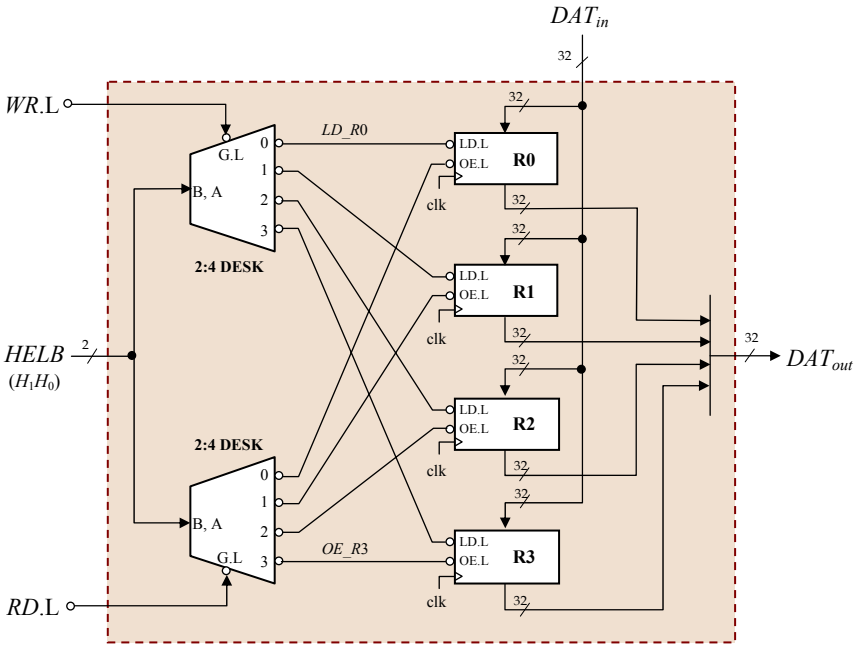


5.2. irudia. 32 biteko 4 erregistroko erregistro-multzoaren egitura.

5.2. irudian ageri den moduan, multiplexore bat erabili dugu erregistro jakin baten edukia aukeratzeko. 3.2.1. atalean azaldu dugun moduan, badago beste aukera bat gauza bera egiteko: hiru egoerako erregistroak erabiltzea.

Hiru egoerako erregistroek badute kontrol-seinale berezi bat —*OE*, *output enable*— zeinaren bidez irteera Z egoeran utz daitekeen, deskonektatuta (ez 0 ez 1). Hala, hainbat erregistroren irteerak konekta daitezke bus komun batera, baldin eta, gehienez, irteera bat aktibatuta badago. Hori lortzeko, deskodegailu bat erabil daiteke, erregistro bakoitzaren *OE* seinalea kontrolatzeko.

Egitura hori oso egokia da erregistro kopurua altua denean, eta, geroxeago azalduko dugun moduan, ohikoa da memorietan. 5.3. irudian ageri da aurreko erregistro-multzoa baina hiru egoerako erregistroak erabiliz.



5.3. irudia. 32 biteko hiru egoerako 4 erregistroko erregistro-multzoaren egitura.

Laburbilduz. Erregistro-multzo bat hainbat erregistrorekin antolatutako memoria txiki bat da. Erregistro bakoitzak helbide jakin bat du, eta erregistroaren edukia irakurtzeko edo idazteko, helbidea adierazi behar da. Idazketak egiteko, deskodegailu batek deskodetzen du helbidea eta aktibatzen du erregistro bakar baten karga-seinalea. Irakurketetarako, multiplexore bat erabil daiteke, non aukeratuko den erregistro baten edukia; aukeran, hiru egoerako erregistroak erabil daitezke, eta orduan ez da

beharrezkoa irteerako multiplexorea, baina bai beste deskodegailu bat erregistroen *OE* kontrol-seinaleak modu egokian aktibatzeko.

Erregistro-multzoak datu kopuru txikirako erabiltzen dira, eskuarki 4 eta 256 artean. Hortik aurrera, memoriak erabili behar dira.

5.3. MEMORIEN EZAUGARRI NAGUSIAK

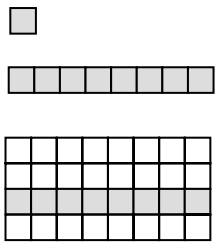
Aipatu dugun moduan, datu kopuru handia gorde behar bada sistema digital batean, erregistro-multzo bat ez da nahikoa, eta memoria bat erabili behar da. Zirkuituak eraikitzeko teknologiak direla eta, oso memoria desberdinak daude, baina hainbat ezaugarri komunak dira. Azter ditzagun, bada, ezaugarri horiek.

5.3.1. Oinarrizko kontzeptuak

Memoria batean gorde daitekeen informazio kopuru txikiena bit bat da, baina, eskuarki, memoriak ez dira erabiltzen bit solteak gordetzeko. Memoria bat bit bateko gailuen multzo handi bat da, egitura jakin batean antolatuta. Bit bat gordetzeko erabiltzen den gailu bakoitza memoria-gelaxka bat da. Azter dezagun nola egituratzen den informazioa memoria batean.

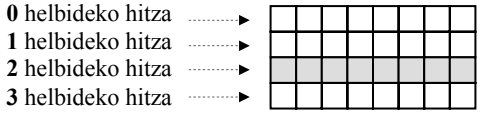
- **Hitza.** Memoriako gelaxkak —bitak— tamaina handiagoko egituretan elkartzen dira: "hitzetan". Esaterako, ohikoak dira memorietan 8 biteko hitzak (byte batekoak[12]). Beraz, memoria-hitz bat *n* biteko datu bat da; oro har, "esanahia" duen datu bat.

  Memoria bat, orduan, hitz multzo ordenatu bat da.



[12] Oro har, B letra larria erabiliko dugu byte adierazteko eta b letra xehea bit adierazteko. Esaterako 1 MB = 1 megabyte; 1 kb = 1 kilobit.

- **Helbidea.** Hitz bakoitzak posizio jakin bat du memorian, eta posizio horri helbide bat dagokio, 0tik N–1eraino, N izanik memoriak gordetzen duen hitz kopurua. Hitzaren helbidea erabili behar da hitz hori irakurtzeko edo idazteko.



  Helbideak bitarrez adierazten dira, p bit erabiliz. Hala, honako erlazio hauek betetzen dira hitz kopuruaren eta helbideen bit kopuruaren artean:

  $$p = \log_2 N \qquad \text{eta} \qquad N = 2^p$$

  Salbuespenak salbuespen, memoria baten hitz kopurua 2ren berretura da; esaterako, 512 hitz (9 biteko helbideak) edo 1024 hitz (10 biteko helbideak), baina ez 800 hitz.

- **Edukiera.** Memoriak gorde dezakeen bit kopurua adierazten du edukierak, eta bi parametroen biderketa gisa eman ohi da: hitz kopurua × hitzen tamaina.

  Hitz kopurua adierazteko, liburuaren sarreran adierazi ditugun multiploak erabiltzen dira: 1 k = 2^{10} = 1.024 hitz, edo 1 M = 2^{20} = 1.048.576 hitz. Beraz, adi! "kiloa" 1000 baino zerbait gehiago da, eta "mega" milioia baino zerbait gehiago.

  Adibidez, honako memoria hauek izan ditzakegu:

  512 × 16 → 16 biteko 512 hitz (guztira, 8 kilobit)
  256 k × 8 → 8 biteko (byte bateko) 256 k hitz (256 kilobyte = 2 megabit)
  1 M × 1 → bit bateko 1 M hitz (1 megabit)

  Edukiera bereko memoria desberdinak izan ditzakegu. Esaterako, 256 biteko memoria batek honako egitura hauetako bat izango du:

  256 hitz × 1 bit; 64 hitz × 4 bit; 32 hitz × 8 bit; eta abar.

- **Eragiketak.** Oro har, bi eragiketa egin daitezke memoria batekin: datu bat idatzi memoria-posizio batean (**idazketa**) edo memoria-posizio baten edukia irakurri (**irakurketa**). Zenbait memoriatan, ordea,

irakurketa baino ezin da egin. Bi eragiketa horiek kontrol-seinale berezien bidez egiten dira; hauek dira ohikoenak:

WR (*write*): idazketa gauzatzeko

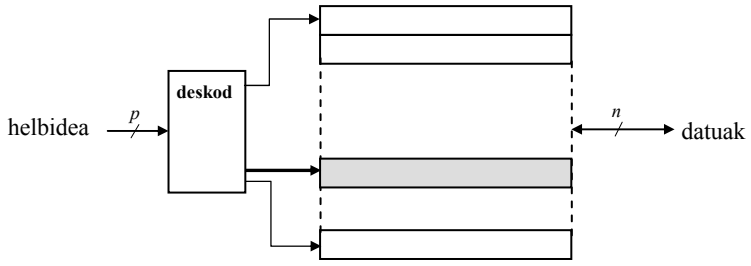
OE (*output enable*): irakurketa egiteko

CS (*chip select*): gaikuntza-seinale arrunta

Gero aztertuko ditugu seinale horiek banan-banan.

5.3.2. Memorien barne-egitura

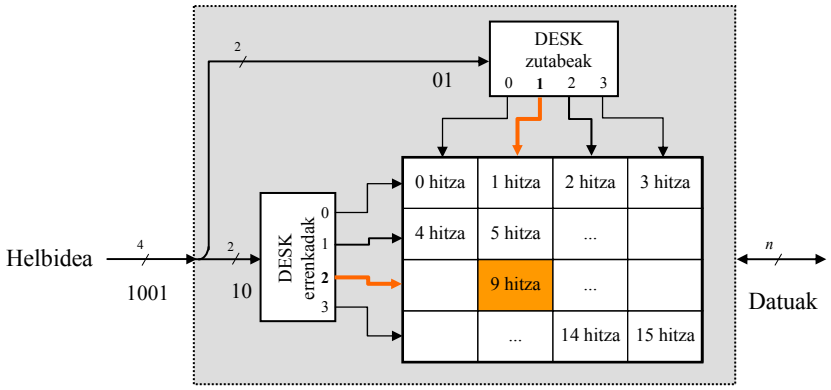
Memoria baten barne-egitura aurreko atalean aztertu dugun erregistro-multzoarena da. Memoria-posizio guztiak ordenatuta daude haien helbideen arabera. Memoria-posizio jakin bat atzitzeko, dagokion helbidea deskodetu behar da, posizio hori "aktibatzeko", irakurtzeko edo idazteko. Irakurketaren kasuan, memoria-posizio baten edukia aukeratu behar da, guztien artean, irteerara eramateko. Hori egiteko, ez da multiplexore bat erabiltzen, oso handia izango litzatekeelako. Izan ere, memoriako posizio guztiek hiru egoerako irteera dute eta, hala, irteera komun batera konekta daitezke, posizio bakar bat izango baita aktibatuta (ikus 3.2.1. atala).



Irudian ageri den moduan, ohikoa da lerro berberak erabiltzea sarrera-datuak eta irteera-datuak emateko (bi noranzkoko lerroak). Idazketa egiten ari denean, datu-lerroak sarrera gisa prozesatzen dira; irakurketa egiten denean, berriz, irteera gisa.

Helbideak deskodetu behar dituen deskodegailua oso handia izan daiteke. Esaterako, 16 k hitzeko memoria batek 14 biteko helbideak behar ditu; beraz, 14:16384 deskodegailua beharko luke, handiegia aukeran. Hori dela eta, memoriako hitzak ez dira antolatzen linealki, bi dimentsiotan baizik, errenkadak eta zutabeak osatuz, 5.4. irudian ageri den moduan (16 hitzeko

memoria baterako). Hala, hitz bat atzitzeko, bi partetan banatzen da *p* biteko helbidea memoriaren barruan: *p*/2 bit errenkadarako eta *p*/2 bit zutaberako. Bi deskodegailu erabiliz errenkada eta zutabea deskodetzeko, helbideari dagokion memoria-posizioa aukeratuko da, hitz bat idazteko edo irakurtzeko.



5.4. irudia. Memoria baten barne-egitura. Hitzek matrize bat osatzen dute, eta helbideek matrize horren errenkada eta zutabe bana adierazten dute.

Aurreko adibideari jarraituz, 14 biteko helbideak bitan banatzen dira: 7 bit errenkadetarako eta 7 bit zutabeetarako; ondorioz, 7:128 bi deskodegailu behar dira, aurrekoa (14:16384) baino askoz txikiagoak.

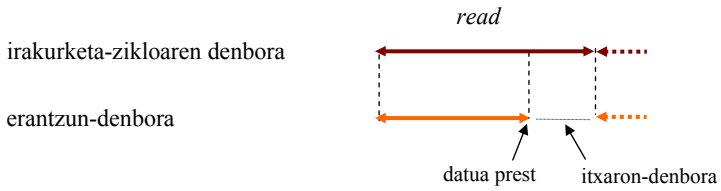
5.3.3. Denbora-parametroak

Hurrengo atalean aztertuko dugun moduan, protokolo zehatzak baliatu behar dira memoriak irakurtzeko edo idazteko. Protokolo horiek zehazten dute nola eta noiz adierazi behar diren kontrol-seinaleak, helbideak eta datuak. Liburu honetan aztertzen ari garen zirkuituetan motelenak dira memoriak, eta haien **abiadura** bi parametroren bidez definitu ohi da:

- **Atzipen-denbora (erantzun-denbora):** irakurketa edo idazketa bat hasten denetik eragiketa hori bukatu arteko denbora, jadanik datuak irakurri edo idatzi direlako.

- **Irakurketa- edo idazketa-zikloaren denbora:** eragiketa bat hasten denetik hurrengo eragiketa bat hasteko igaro behar den denbora minimoa.

Irudian, irakurketa-zikloa eta atzipen-denbora ageri dira (eskematikoki):



Eskuarki, bi denbora-parametroak berdinak dira, hau da, bigarren eragiketa has daiteke lehenengoa bukatu bezain laster, baina, zenbait memoriatan, denbora-tarte bat igaro behar da eragiketa bat bukatu denetik hurrengoari ekin ahal izateko.

Memorien funtzionamendua zehazten duten protokoloetan, denbora-parametro asko definitzen dira, baina ez ditugu hemen azalduko, fabrikatzaileen eta zirkuituen araberakoak izan ohi baitira. Nolanahi ere den, denbora-parametro horiek bi multzotan sailka daitezke: *set-up* motako denborak eta *hold* motako denborak. *Set-up* motako denbora batek adierazten du ekintza bat hasi baino zenbat denbora lehenago aktibatu behar den seinale jakin bat (esaterako, irakurketarako kontrol-seinalea aktibatu baino lehen, zenbat denbora aurretik adierazi behar den helbidea). *Hold* motako denbora batek, ostera, hau adierazten du: zenbat denbora mantendu behar den aktibatuta seinale bat ekintza bat burutu eta gero.

5.3.4. Memorien sailkapena

Aipatu dugun bezala, memoria mota asko dago. Erdieroalezko memoria nagusiak honela sailka daitezke:

- **RAM memoriak** (*Random Access Memory*)

  Bi eragiketak onartzen dituzte: irakurtzea eta idaztea. Informazioa ez da iraunkorra. Memoria-gelaxka eraikitzeko erabiltzen den teknologiaren arabera, bi motatako RAM memoriak ditugu: **RAM estatikoak** (SRAM) eta **RAM dinamikoak** (DRAM).

- **ROM memoriak** (*Read Only Memory*)

  Eskuarki, eragiketa bakarra egin daiteke: irakurketa. Zenbait kasutan, idazketak ere egin daitezke, baina prozesu desberdin batez egiten dira eta behar duten denbora askoz altuagoa da (4 edo 5 magnitude-ordena altuagoak). ROM memorien datuak iraunkorrak dira, eta, oro har, sistema digitaletik kanpo grabatzen dira. Berriro ere, teknologiaren arabera, hainbat ROM memoria daude: **ROM, PROM, EPROM, EEPROM, FLASH,** eta abar.

Memoria mota bakoitza funtzio desberdinetarako erabiltzen da sistema digitaletan (konputagailuetan) eta ezaugarri desberdinak ditu. Ezaugarriak ezaugarri, hona hemen edozein memoriari eskatuko genizkiokeen ezaugarrien zerrenda: ahalik eta **azkarrena** izatea; **bi eragiketak** onartzea; **iraunkorra** izatea, informazioa ez galtzeko; **dentsitate altukoa** izatea (bit asko integratzea zirkuitu bakar batean); **kontsumo baxukoa** izatea; eta **fidagarria** izatea (gorde den informazioan errorerik ez izatea).

Zoritxarrez, ez dago memoria bat ezaugarri horiekin guztiekin, eta, beraz, egoera bakoitzerako egokiena aukeratu beharko da. Hurrengo ataletan, memorien funtzionamendua eta ezaugarri nagusiak aztertuko ditugu.

5.4. RAM MEMORIAK

Izenak **zorizko atzipeneko memoria** esan nahi du (*random access memory*); hau da, edozein posiziotako hitza atzitzeko behar den denbora berdina da. Berez, izenak ez du adierazten memoria hauen ezaugarri nagusia, kapitulu honetan aztertuko ditugun memoria guztiak zorizko atzipenekoak baitira (badaude atzipen sekuentzialeko memoriak, zinta magnetikoak esaterako, baina ez ditugu aztertuko).

RAM memorien ezaugarri nagusia hauxe da: **idazteko** zein **irakurtzeko** memoriak dira. Bi eragiketak antzeko moduan eta denboran egiten dira. Baina RAM memorietan gordetzen den informazioa **ez da iraunkorra,** hots, memoriaren edukia ez da mantentzen sistema digitala "itzaltzen" denean[13].

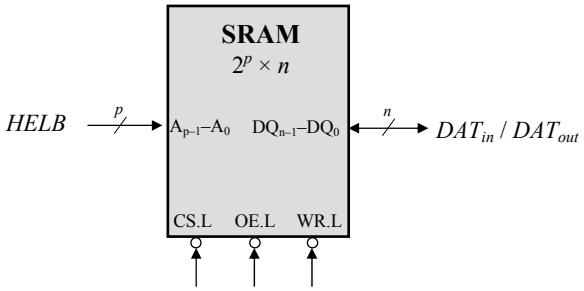
---

[13] Zirkuitu guztiek behar dute elikadura-tentsioa transistoreek funtziona dezaten, eta, horretarako, hankatxo bereziak dituzte txip guztiek. Sistema digitala ez badago "piztuta", hots, zirkuituek ez badute elikadura-tentsiorik, ez dute funtzionatzen eta, memoria duten zirkuituen kasuan (adibidez, biegonkorrak, erregistroak edo RAM memoriak), gordeta dagoen informazioa galtzen da.

RAM memoriak bi motatakoak dira, bit bateko gelaxka eraikitzeko erabiltzen den teknologiaren arabera: RAM estatikoak (SRAM) eta RAM dinamikoak (DRAM).

5.4.1. RAM estatikoak

RAM estatiko baten bit bateko oinarrizko gelaxka D biegonkor baten antzekoa da[14]. 5.5. irudian ageri da SRAM memoria baten eskema logikoa.



5.5. irudia. RAM estatiko baten eskema logikoa (A = *address*, helbidea; DQ = datu-sarrera eta -irteera).

Oro har, honako sarrera eta irteera hauek dituzte SRAM memoriek:

- Sarrerak eta irteerak

  HELB: Hitz jakin baten *p* biteko helbidea. Helbide hori adierazi behar da hitza irakurtzeko edo idazteko. *p* biteko helbideak izanik, 2^p hitz izango ditu memoriak.

  DAT_{in} / ***DAT_{out}:*** *n* biteko datuak, idatzi behar direnak edo irakurri direnak. Oro har, sarrera- eta irteera-datuek lerro berberak erabiltzen dituzte. Irakurketa egiten denean, irteera gisa funtzionatzen dute; idazketetan, aldiz, sarrera gisa.

  Hiru egoerako lerroak dira, beraz. Gogoratu: aktibatuta daudenean, 0 edo 1 egoeran egongo dira; desaktibatuta, aldiz, Z egoeran egongo dira, hots, deskonektatuta.

[14] Eskuarki, 6 transistore behar dira RAM baten gelaxka bat egiteko. Ikus E2 eranskina.

- Kontrol-seinaleak

  OE (*output enable*): Irakurketak egiteko aktibatu behar den kontrol-seinalea; datu-lerroak irteera-lerro bihurtzen ditu.

  WR (*write*): Idazketa egiteko aktibatu behar den kontrol-seinalea; datu-lerroak sarrera-lerro bihurtzen ditu.

  CS (*chip select*): Ohiko gaikuntza-seinalea (beste zirkuituetan, *G* edo *E* izenekoa).

Taula honetan laburbiltzen da memoriaren funtzionamendua:

CS	*WR*	*OE*	*funtzioa*
0	–	–	desaktibatuta
1	0	0	–
1	0	1	irakurketa
1	1	0	idazketa

CS = 0 denean, memoria desaktibatuta dago. Aitzitik, *CS* = 1 denean, memoria gaituta dago, eragiketa bat egiteko prest[15]. Desaktibatuta dagoenean edo ez bada eragiketarik egiten ari, memoriaren datu-lerroak hirugarren egoeran (Z) egongo dira.

Bestalde, datuen sarrera eta irteera lerro berberetan egiten direnez, ez dira aktibatu behar batera *OE* eta *WR* seinaleak, batak irteera gisa eta besteak sarrera gisa erabiltzen baititu datu-lerroak.

5.4.1.1. Irakurketak eta idazketak egiteko protokoloak

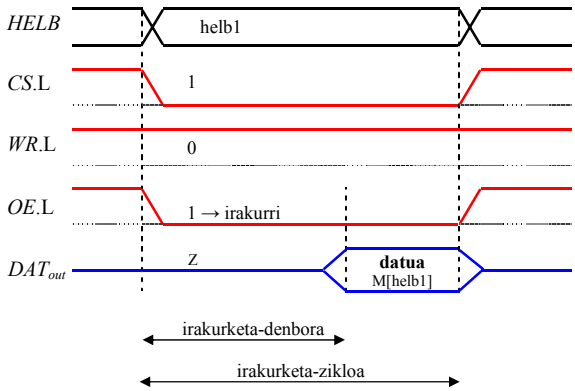
Arestian aipatu dugun moduan, protokolo jakin batzuk baliatu behar dira memoriak irakurtzeko edo idazteko, datuak eta kontrol-seinaleak ordena zehatz eta jakin batean eman behar baitira. Beraz, azter dezagun nola egin irakurketa bat edo idazketa bat SRAM memoria batean[16].

---

[15] Memoria aktibatuta dagoenean (*CS* = 1), kontsumoa altuagoa da. Hori dela eta, oso egokia da memoria desaktibatuta izatea erabili behar ez denean. Zenbait kasutan, aldiz, memoriak aktibatuta mantentzen dira, azkarrago egin ahal izateko irakurketak eta idazketak.

[16] Fabrikatzaile bakoitzak eta memoria bakoitzak bere protokoloak erabiltzen ditu, nahiz eta oso antzekoak diren haien artean; xehetasunak aparte utziko ditugu, eta protokolo horien garrantzitsuena azalduko dugu.

5.6. irudian, irakurketaren protokoloa ageri da (sinplifikatuta). Helbide bat adierazita, *CS* (*chip select*) eta *OE* (*output enable*) seinaleak aktibatu behar dira. Memoriaren datu-lerroak Z egoeran daude hasieran, baina *OE* seinalearen aktibazioaren ondorioz, irteera gisa funtzionatuko dute. Memoriari dagokion erantzun-denbora igaro ondoren, hor izango dugu irakurri den datua: helbideak adierazitako memoria-posizioan gordeta zegoen hitza. Emaitza irteeran izango da *OE* seinalea (eta *CS* seinalea eta helbidea) aktibatuta dagoen bitartean (tarte horretan aldatzen badugu helbidea, beste memoria-posizio bat irakurtzen hasiko da).

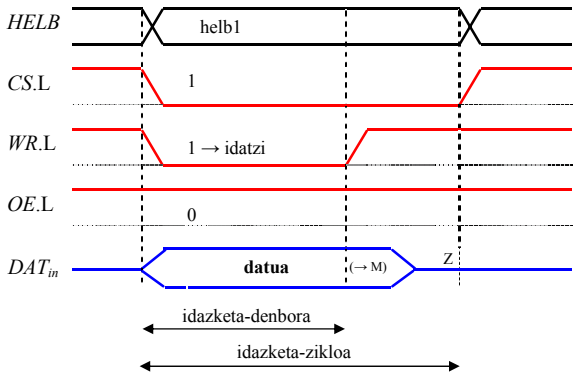


5.6. irudia. SRAM memorien irakurketa-protokoloa (sinplifikatuta).

Irakurketa-denbora —datu bat irakurtzeko behar den denbora— eta irakurketa-zikloa —irakurketa-eragiketak iraun behar duen denbora minimoa— markatu ditugu 5.6. irudian. Eskuarki, SRAM memoria baten irakurketa-denbora eta irakurketa-zikloa berdinak dira, hau da, has daiteke hurrengo irakurketa batekin lehenengoaren emaitza lortu bezain laster. Oro har, irakurketa bat egiteko, helbidea *OE* seinalea baino lehen edo aldi berean adierazi behar da. *CS* seinalea *OE* seinalearekin batera edo lehen aktiba daiteke (izan ere, *CS* seinalea beti aktibatuta egon daiteke, baina kasu horretan kontsumoa altuagoa da).

Idazketaren protokoloa antzekoa da, eta 5.7. irudian ageri da. Kasu honetan, idatzi nahi den datua datu-lerroetan jarri behar da, eta *WR* seinalea aktibatu. Gogoratu: idazketa bat denez (*OE* = 0; *WR* = 1), memoriaren datu-lerroek sarrera gisa funtzionatuko dute. Berriro ere, seinaleen arteko ohiko ordena hauxe litzateke: helbidea eta datuak, lehendabizi, eta *WR* seinalea gero (edo aldi berean).

WR seinaleak (eta datuak) denbora minimo bat egon behar du aktibatuta idazketa egiteko: idazketa-denbora hain zuzen ere.



5.7. irudia. SRAM memorien idazketa-protokoloa (sinplifikatuta).

Memoria asko dago merkatuan, tamaina eta abiadura desberdinekoak, baina, adibidez, ohikoak dira, gaur egun, 64 MB eta 10 – 50 ns-ko erantzun-denborak dituzten SRAM memoriak.

5.4.2. RAM dinamikoak

Aipatu dugun moduan, SRAM memorien oinarrizko gelaxka D biegonkorraren antzekoa da. Eraiki daiteke bit bateko memoria-gelaxka sinpleagoa? Hala balitz, memoria-edukiera handiagoak lortuko genituzke (integrazio-dentsitate handiagoko memoriak) memoriako txip batean.

Hori da DRAM (RAM dinamikoa) memorien kasua. RAM memoria dinamikoen oinarrizko gelaxka kondentsadore bat da. Bit bateko informazioa lortzeko, kondentsadorearen karga (eta, ondorioz, tentsioa) erabiltzen da: kondentsadorea kargatuta (1) edo deskargatuta (0) dago.

Teknologia horren bidez, memoria trinkoagoak edo edukiera handiagokoak lortzen dira; gutxi gorabehera, lau aldiz trinkoagoak dira DRAM memoriak SRAM memoriak baino (ikus E2 eranskina). Baina, zoritxarrez, badago arazo bat, kondentsadoreek metatzen duten karga galtzen baita denboran zehar: kondentsadoreak deskargatzen dira, memoria-txipa elikatuta badago ere. Adi! karga galtzean, gorde den informazioa galtzen da! Gainera, deskarga-denbora oso laburra da, milisegundo batzuetakoa.

Hori dela eta, memoria dinamikoak erabili ahal izateko, prozesu berezi bati ekin behar zaio periodikoki, kondentsadoreetan metatu den karga berreskuratzeko. Prozesu horri **freskaketa** (*refresh*) deitzen zaio.

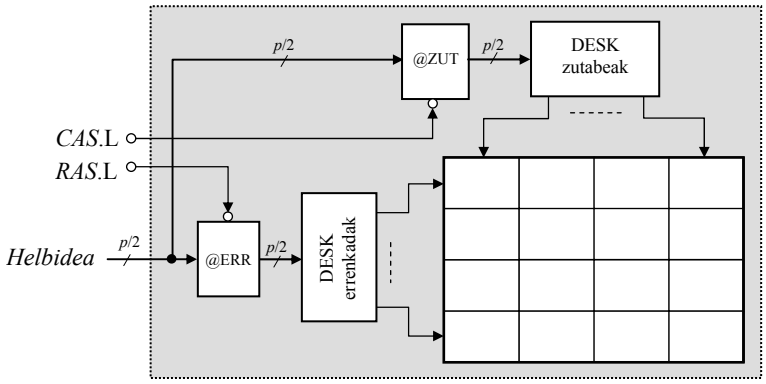
Aipatu dugun bezala, edukiera handiagoko memoriak lortzen dira RAM dinamikoekin. Horrek ere, beste ikuspuntu batetik aztertuta, arazoak sor ditzake: txipen hanka kopurua hazi egingo da. Kopuru hori mugatzeko, RAM dinamiko baten atzipen-prozesua desberdina da.

5.3.2. atalean adierazi dugun moduan, memoria baten gelaxka guztiek matrize bat osatzen dute; helbide bat emanda (p bit), errenkada eta zutabe bana zehazten dira ($p/2$ bit). Ideia hori aprobetxa daiteke memoria-txiparen hankatxoen kopurua txikiagotzeko; hitz baten helbidea dena batera eman beharrean, bi fasetan adierazten da: lehendabiziko fasean, errenkadaren $p/2$ bitak, eta, gero, zutabearen beste $p/2$ bitak. Hala, nahikoa da $p/2$ biteko helbide-sarrera bat.

Hori egin ahal izateko, bi kontrol-seinale berri gehitzen dira:

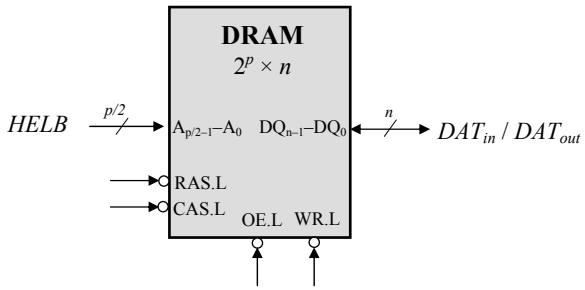
- *RAS* (*row address strobe*): hitzaren errenkada-helbidea ematen ari dela adierazteko
- *CAS* (*column address strobe*): hitzaren zutabe-helbidea ematen ari dela adierazteko.

Bi seinale horien bidez, helbidearen bi zatiak —errenkada eta zutabea— memorian bertan "gordetzen" dira, hitz jakin bat atzitu ahal izateko. 5.8. irudian, DRAM memoria baten barne-egitura ageri da, helbide-lerroak nola erabiltzen diren argi uzteko.



5.8. irudia. DRAM memoria baten helbideratze-eskema. p biteko helbideak $p/2$ biteko bi zatitan ematen dira; horregatik, $p/2$ helbide-lerro baino ez dago.

Beraz, aurreko guztia kontuan harturik, 5.9. irudian ageri da DRAM memoria baten eskema logikoa.



5.9. irudia. DRAM baten eskema logikoa.

- Sarrerak eta irteerak

  HELB: Hitz jakin baten errenkadaren edo zutabearen helbidea. Hitzen helbideak p bitekoak badira (2^p hitz daudelako), nahikoa da $p/2$ helbide-lerrorekin, helbidea bi fasetan adierazten baita.

  DAT_{in} / DAT_{out}: n biteko datuak, sarrerakoak idazketa egiten denean eta irteerakoak irakurketa egiten denean (SRAM memorietan bezala)[17].

- Kontrol-seinaleak

  RAS (*row address strobe*): Helbide-lerroetan hitz baten errenkada-helbidea dagoela adierazteko.

  CAS (*column address strobe*): Helbide-lerroetan hitz baten zutabe-helbidea dagoela adierazteko.

  WR (*write*): Idazketa egiteko aktibatu behar den kontrol-seinalea.

  OE (*output enable*): Irakurketa egiteko aktibatu behar den kontrol-seinalea; datu-lerroak irteera-lerro bihurtzen ditu.

  Aurrekoez gain, zenbait memoria-txipetan, ohiko *chip select* kontrol-seinalea ere izango dugu.

[17] Maiz, DRAM memorien hitzak bit batekoak dira (32 M x 1, adibidez). Kasu horietan, ohikoa da sarrerako eta irteerako datu-lerroak desberdinak izatea.

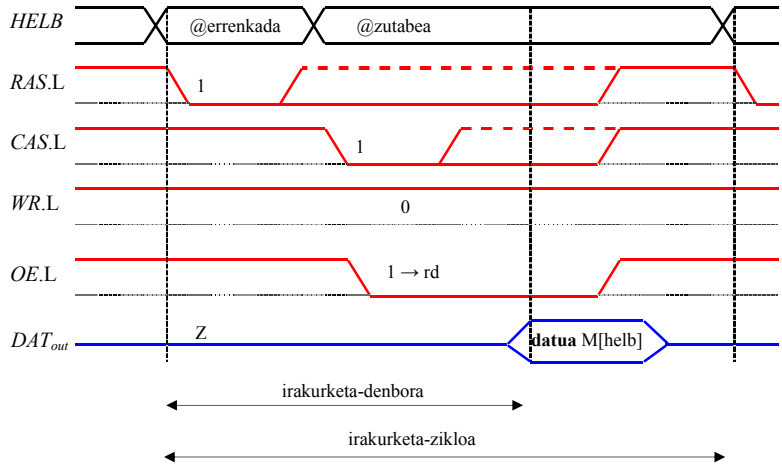
5.4.2.1. Irakurketa- eta idazketa-zikloak

Gainerako memorietan bezala, irakurketa eta idazketa dira ohiko prozesuak DRAM memorietan. 5.10. irudietan, irakurketa- eta idazketa-zikloak ageri dira, kasu orokor baterako (izan ere, desberdintasunak daude memoria komertzialen artean; beraz, zirkuitu bakoitzaren ezaugarri-orrietara jo behar da zehaztasunak argitzeko).

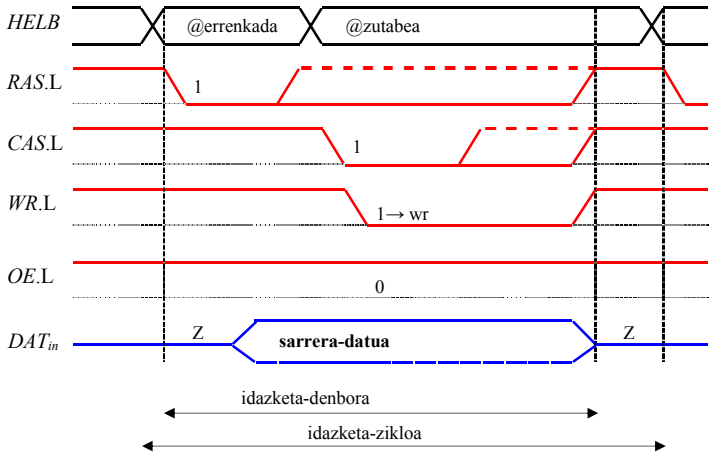
Lehen aipatu dugun moduan, atzitu nahi den hitzaren helbidea bi fasetan ematen da. Lehendabizi, errenkadaren helbidea (helbidearen pisu handieneko $p/2$ bitak) jarri behar da helbide-sarreran, eta *RAS* seinalea aktibatu. Gero, zutabearen helbidea (helbidearen pisu txikieneko $p/2$ bitak) eman behar da, eta *CAS* seinalea aktibatu (memoria batzuetan, bigarren fasean, *CAS* seinalea aktibatzeaz gain, *RAS* seinalea ere mantendu behar da aktibatuta; beste batzuetan, ordea, desaktibatu behar da).

SRAM memorien kasuan bezala, irakurketa egiteko, $WR = 0$ eta $OE = 1$ izan behar dira; eta idazketa egiteko, $WR = 1$ eta $OE = 0$.

DRAM memorietan, desberdinak dira irakurketa-denbora eta irakurketa-zikloa (gauza bera idazketaren kasuan). Izan ere, arrazoi teknologikoak direla medio, *RAS* eta *CAS* seinaleak desaktibatzen direnetik, denbora jakin bat itxaron behar da hurrengo eragiketari ekin ahal izateko (esaterako, irakurketa-denbora 60 ns izan daiteke eta irakurketa-zikloa, aldiz, 80 ns).



5.10. irudia. (a) DRAM baten irakurketa-zikloa (fabrikatzaileen arabera, RAS eta CAS seinaleak lehenago desaktiba daitezke).



5.10. irudia. (b) DRAM baten idazketa-zikloa.

Laburbilduz, eragiketa bat egiteko, honako prozedura honi jarraitu behar zaio:

- hitzaren errenkada-helbidea eman eta *RAS* seinalea aktibatu
- hitzaren zutabe-helbidea eman eta *CAS* seinalea aktibatu
- eragiketa adierazi, irakurri edo idatzi, eta, idaztean, datuak ere
- desaktibatu *RAS* eta *CAS* seinaleak

5.4.2.2. Irakurketa eta idazketa azkarrak

DRAM memoria baten ohiko eragiketa-zikloak azaldu ditugu aurreko atalean. Protokoloak "konplexuak" dira eta, hori dela eta, SRAM memoriek baino denbora gehiago behar dute. Badaude, hala ere, kasu partikular batzuk zeinetan azkarrago egin daitezkeen eragiketak. Arruntenak bi hauek dira:

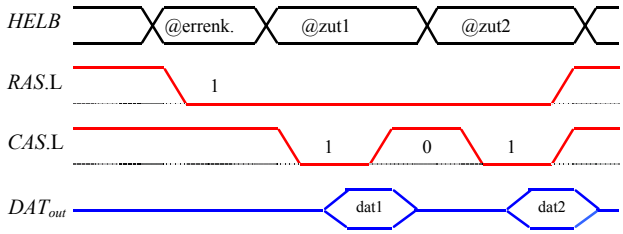
- Errenkada bereko hitz bat baino gehiago irakurri edo idatzi behar dira.

  Maiz gertatzen da egoera hori sistema digitaletan, eta ezaugarri berezi bat dauka: atzitu behar diren hitzen errenkada-helbideak berdinak dira. Hori dela eta, atzipen modu sinplifikatu bat erabil daiteke hitzak atzitzeko: FPM, ***fast page mode*** izenekoa.

  5.11. irudian ageri da *page mode* motako irakurketa bat. Errenkada bereko bi hitz irakurri behar dira. Irudian ageri den moduan, lehenengo hitzaren irakurketa ohiko prozeduraren arabera egiten da: *RAS* seinalea aktibatu behar da errenkadaren helbidearekin batera, eta, gero, *CAS* seinalea, zutabearen helbidearekin batera. Hortik aurrera, ordea, prozedura sinplifikatzen da; errenkada bereko hurrengo hitza(k)

atzitzeko, nahikoa da zutabe-helbidea(k) ematea *CAS* seinalearekin batera, eta *RAS* seinalea aktibatuta mantendu.

Page mode motako irakurketak azkarragoak dira, ez baita ohiko protokolo osoa exekutatu behar (ikus 5.6. ariketa).

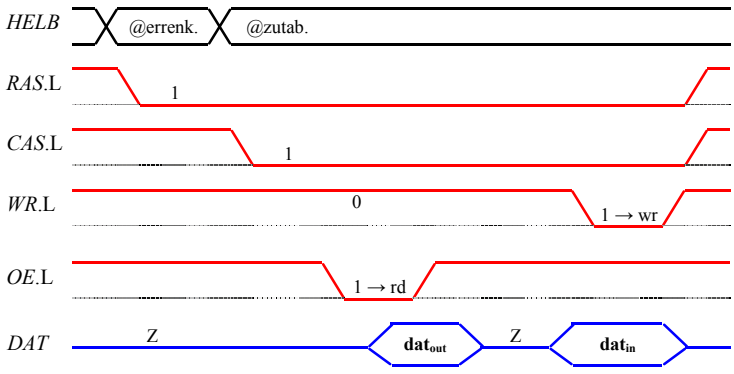


5.11. irudia. *Page mode* motako irakurketa. Bi hitzak errenkada berean daude.

- Memoria-posizio bat irakurri eta idatzi behar da, tartean beste memoria-eragiketarik egin gabe. Adibidez, M[100] := M[100] + 1 egin behar da.

  Eragiketa mota hori oso ohikoa da sistema digitaletan, eta, bi eragiketak (lehendabizi irakurketa, eta gero idazketa) oso-osorik exekutatu beharrean, RMW (***read-modify-write***) deitzen den protokolo sinplifikatua erabil daiteke.

  RMW moduan, memoria-posizio bat irakurri eta gero ez dira RAS eta CAS seinaleak desaktibatzen (ez da eragiketa bukatzen). Memoriako hitz bat aukeratuta, datu-lerroak irteera izatetik (*OE*) sarrera izatera (*WR*) igaroarazten dira eta, orduan, idazketa betetzen da. Idazketa bukatuta, eragiketa amaitutzat ematen da, *RAS* = *CAS* = 0 eginez. 5.12. irudian, RMW memoria-eragiketa ageri da.



5.12. irudia. *Read-Modify-Write* motako eragiketa. Hitz bat irakurtzen eta idazten da.

5.4.2.3. Edukien freskaketa (*refresh*)

Aipatu dugunez, DRAM memorien gelaxka kondentsadore bat da, eta kondentsadorearen bi egoerak —kargatuta eta deskargatuta— erabiltzen dira 1 eta 0 balio logikoak adierazteko. Zoritxarrez, kondentsadoreen karga galdu egiten da denboran zehar, eta, ondorioz, berreskuratu egin behar da periodikoki, informazioa ez galtzeko. Berreskuratze-prozesuari **freskaketa** (*refresh*) deritzo. Adi! Kondentsadoreen karga "azkar" galtzen da, eta, beraz, freskaketa-eragiketa oso maiz egin behar da; adibidez, memorien arabera, 10 – 100 milisegundorik behin.

Freskaketa-prozesua egiteko, memoriako bit guztiak irakurri beharko genituzke eta berridatzi: 1 diren kasuetan —hots, karga "nahikoa" badago kondentsadorean—, birkargatu, karga-maila altuena eskuratzeko. Azalduko ez ditugun teknologia kontuak direla eta, freskaketa errenkadaz errenkada egiten da (ez bitez bit): nahikoa da irakurtzea errenkada bat errenkada horren bit guztien karga berreskuratzeko.

Freskaketa-prozesua bi modutan egin daiteke:

- Modu "trinkoan". Memoriaren errenkada guztiak, lehenengotik azkeneraino, freskatzen dira, bata bestearen atzetik. Freskaketa egiten den bitartean, ezin da memoria erabili.
- Modu "banatuan". Errenkadak banan-banan (edo multzoka) freskatzen dira, eta freskaketa-prozesua ohiko eragiketekin nahasten da. Hala, memoria erabili behar ez denean egingo da errenkaden freskaketa.

Freskaketa egiteko protokoloa berezia da. Errenkadak baino ez dira helbideratu behar, eta, eskuarki, bi modutan egiten da:

- *CAS-before-RAS* izenekoa. Irakurketa eta idazketa arruntak egiteko, kontrol-seinaleen ordena *RAS* / *CAS* da. Kontrako ordenan ematen badira, orduan errenkadaren freskaketa egiten da.
- *RAS* seinalea bakarrik aktibatzen da, errenkadaren helbidea baino ez baita eman behar.

5.4.3. RAM memorien laburpena

Sistema digitaletan, informazio asko sortu, gorde, aldatu eta abar egin behar denean, RAM memoriak erabili behar ditugu. RAM memoria batean, hainbat hitz gordetzen dira, eta bakoitzak helbide bat du. Hitzaren helbidea eman behar da hitz hori irakurtzeko edo idazteko. Memoriak *N* hitz gorde

ahal baditu, helbideak $p = \log_2 N$ bitekoak izango dira, eta, alderantziz, p biteko helbideak dituen memoria bat $N = 2^p$ hitzekoa da.

Bi motatako RAM memoriak daude: SRAM eta DRAM. Lehenengoetan, memoria-gelaxka D biegonkor baten antzekoa da; bigarrenean, ordea, kondentsadore bat. DRAM memoriak estatikoak baino trinkoagoak dira (hau da, informazio gehiago gorde dezakete).

Hala ere, DRAM memoriek bi arazo dituzte. Batetik, hitzen p biteko helbideak bi fasetan adierazi behar dira: errenkada lehendabizi (p/2 bit), eta zutabea gero (beste p/2 bitak). Gainera, eragiketa bat egin ondoren, denbora-tarte bat utzi behar da hurrengo bati ekiteko. Ondorioz, DRAM memoriak SRAM memoriak baino motelagoak dira. Bestetik, kondentsadoreen karga nahiko azkar galtzen da, eta periodikoki freskatu behar da.

Mota, egitura, tamaina eta ezaugarri oso desberdineko memoriak daude merkatuan, eta ezin ditugu laburbildu haien parametro nagusiak. Hala ere, erreferentzia gisa, honako datu hauek har ditzakegu kontuan. Oro har, 1 Gbit arteko memoriak ditugu eskura, zenbait egituratan antolatuta, esaterako, hitzen tamainaren arabera; memoria estatikoak dinamikoak baino txikiagoak dira beti. Ziklo-denborak direla eta, ohikoak dira 10-50 ns-ko SRAM memoriak (azkarragoak ere, 2-4 ns-koak), eta 40-80 ns-ko DRAM memoriak.

RAM memorietan gordetzen den informazioa ez da “betiko”: elikaduratik deskonektatzen direnean, galdu egiten da; ezin dira erabili, beraz, informazioa iraunkorki gordetzeko (esaterako, CD batekin egiten dugun moduan). Beraz, kasu horietarako, beste memoria motaren bat behar da.

5.5. ROM MEMORIAK

Aipatu dugun moduan, RAM memoriak irakurtzeko eta idazteko erabiltzen dira, baina haien edukia ezin da mantendu sistema “itzaltzen” denean. Arazo hori saihesteko, beste memoria mota bat erabili behar da: ROM memoria.

ROM memoriak (*Read Only Memory*) **irakurri baino ezin dira** egin, baina haien edukia **iraunkorra** da; hau da, edukia mantentzen da elikadura-tentsioa deskonektatu eta gero.

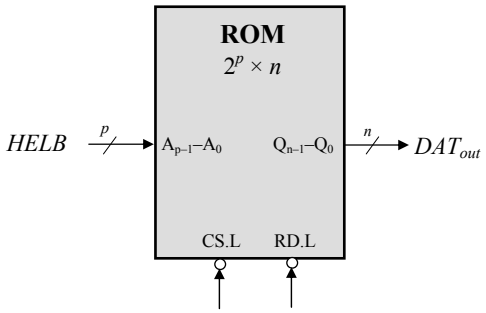
Izan ere, ROM memorien idazketa-prozesua oso bestelakoa da; zenbait kasutan, fabrikatzen direnean idazten dira, eta, beste batzuetan, erabiltzaileak idatz ditzake (grabatu edo programatu) datuak, baina gailu berezi batzuk erabiliz eta funtzionamendu normaletik aparte.

Oro har, ROM memorietan gordetzen da finkoa den informazioa edo, agian, noizbehinka bakarrik, eta modu berezian, aldatuko den informazioa. Adibidez, automata batek exekutatu behar duen programa (esaterako, argazki-kamara batena, edo auto baten kontrol-unitatearena). Funtzio logiko konplexuak sortzeko ere erabil daitezke ROM memoriak; esaterako, ASCII/KANJI kodeen arteko itzulpena, kontrol-unitateen logika, eta abar. Izan ere, nahikoa da, horretarako, funtzioaren egia-taula ROM memorian idaztea; emaitza jakin bat behar denean, memorian dagoen informazioa irakurri baino ez da egin behar (ikus 5.3. eta 5.4. ariketak).

Hainbat ROM memoria daude, erabiltzen den teknologiaren arabera; esaterako: ROM, PROM, EPROM, EEPROM edo FLASH memoriak. Azter ditzagun haien ezaugarri nagusiak.

5.5.1. ROM memorien ezaugarriak

5.13. irudian ageri da ROM memoria baten eskema logiko sinplifikatua.



5.13. irudia. ROM memorien eskema logikoa.

Oro har, honako sarrera eta irteera hauek dituzte ROM memoriek:

- Sarrerak eta irteerak

  HELB: Hitz jakin baten *p* biteko helbidea. Helbide hori adierazi behar da hitz bat irakurtzeko. *p* biteko helbideak ditugunez, 2^p hitz izango ditu memoriak.

  DAT_{out}: Irakurri den *n* biteko hitza. ROM memorien eskema logikoan ez dira ageri datu-sarrerak; gogoratu, ohiko funtzionamenduan, irakurri baino ezin dira egin.

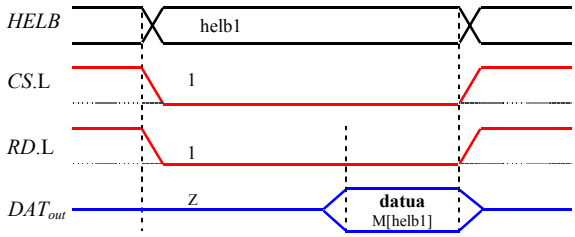
- Kontrol-seinaleak

  RD (*read*): Irakurketa egiteko aktibatu behar den kontrol-seinalea (zenbait kasutan, *OE*, *output enable*).

  CS (*chip select*): Ohiko gaikuntza-seinalea.

5.5.1.1. Irakurketa-zikloa

Kontuan hartuta irakurketa bakarrik egin daitekeela, irakurketa-zikloa da exekutatzen den bakarra, 5.14. irudian ageri den moduan.



5.14. irudia. ROM memoria baten irakurketa-zikloa.

Irakurketa-zikloa sinplea da: nahikoa da, irakurri nahi den hitzaren helbidea eman ondoren, *CS* eta *RD* seinaleak aktibatzea (*CS* seinalea aurretik aktibatuta egon daiteke).

5.5.2. ROM memoria erabilienak

ROM memorien ezaugarri nagusiak dituzten gailu mota asko dago, erabilera, teknologia, ezaugarri eta abarren arabera. Hauek dira erabilienak (laburpena baino ez dugu egingo):

- **ROM** memoriak

  Izen generikoa daramaten memoriak fabrikatze-prozesuan bertan idazten edo "programatzen" dira, behin bakarrik, maskara batzuen bidez. Fabrikatutakoan, edukia "betiko" da (ezin dira berridatzi).

  Programazio-prozesua nahiko garestia da, eta, hori dela eta, eskuarki serie handietan merkaturatzen diren gailuetan bakarrik erabiltzen dira.

- **PROM** memoriak (*programmable read only memory*)

  Kasu honetan ere behin bakarrik idatz daiteke memoria, baina idazketa-prozesua erabiltzailearen esku geratzen da.

  Memoriaren gelaxkak "fusible" bat du (transistore baten bidez gauzatua). Fusiblearen egoera erabiltzen da, gero, informazioa gordetzeko: fusiblea erreta → 0; erre gabe → 1. Memoria idazteko, gailu berezi bat erabili behar da: "grabagailua", zeinaren bidez fusible batzuk erretzen diren eta beste batzuk ez. Urtuz gero, ezin dira berreskuratu; beraz, behin bakarrik idatz daiteke memoria.

  Memoria idatzita, dagokion sistema logikoan txertatuko da, eta hor irakurri baino ezin da egin.

- **EPROM** memoriak (*erasable programmable read only memory*)

  ROM memoriak dira (sisteman irakurtzeko bakarrik), baina idazketa-prozesua behin baino gehiagotan egin daiteke.

  EPROM baten oinarrizko gelaxka MOS transistore berezi bat da, non karga gordeko den ala ez (1a eta 0a). Gelaxkaren egitura dela eta, karga hori iraunkorra da, ez da galtzen (egoera "normalean"). Baina badago prozedura berezi bat transistoreak deskargatzeko, denak batera; hots, memoria ezabatzeko. Horretarako, memoriako transistoreak argi ultramorearen pean jarri behar dira; hala, argiaren energia dela eta, transistoreen eroankortasuna hazten da eta metatzen duten karga elektrikoa galtzen da. Transistoreak argiztatu ahal izateko, memoria-txipek kuartzozko leihatilatxo garden bat dute aurrealdean, non gelaxka guztiak agerian baitaude.

  Ezabatze-prozesua, jakina, sistema logikotik aparte egiten da, eta luze xamarra da (minutuak). Aurreko kasuan bezala, memorien idazketa programagailu berezietan egiten da (milisegundoak).

  Oro har, EPROM memoriak erabiltzen dira sistema logiko baten prototipoak garatzen ari direnean, probak egiteko eta prozesuan zehar detektatzen diren erroreak zuzendu ahal izateko (gaur egun beste soluzio batzuk badaude ere).

- **EEPROM** memoriak (*electrically erasable programmable ROM*)

  EPROM memoriak ezabatzeko eta berriro programatzeko, sistematik atera behar dira. EEPROM batekin, aldiz, prozesu hori guztia sistema logikoan bertan egin daiteke, memoria ezabatzeko erabiltzen den

teknika elektrikoa baita, ez optikoa. Gainera, hitzez hitz ezaba daitezke (EPROMen kasuan, informazio guztia batera ezabatzen da).

Hori dela eta, memoria-gelaxkak EPROMenak baino konplexuagoak dira eta, gainera, memoria ezabatzeko eta berriro grabatzeko zirkuitua sistema digitalean bertan integratu behar da (beraz, garestiagoak dira).

Sisteman bertan integratzen den logika hori guztia dela eta, posible da memorien edukia “urrutitik” aldatzea (sarera konektatuta, esaterako) edo erabiltzaileari banatzen zaizkion programa berezien bidez. Beraz egokiak dira sistema konplexu askotan, hala nola, robotetan, hegazkinetan, sateliteen kontroladoreetan, eta abar, zeinak urrutitik kontrolatu eta, askotan, birprogramatu behar diren.

Memoria hauen aldaera bat EAROM (*electrically alterable ROM*) memoriak ditugu. Aplikazio berezietan, memoria-edukiera txikiak behar dituztenetan, erabiltzen dira: jokoak, irrati-sintonizadoreak eta abar.

- **FLASH** memoriak

  EPROM eta EEPROM memorien alternatiba moduan, FLASH memoriak daude. Alde batetik, FLASH memorien gelaxka EPROM memoriena bezain txikia da (transistore bat) eta, bestetik, ezabatze-prozesua elektrikoa da, EEPROM memorietan bezala. Beraz, saiatzen da bietatik onena hartzen. Ezabatzea ez da egiten, eskuarki, hitzez hitz, hori egiteko behar den zirkuiteria garestia baita transistore kopuruari begiratuz gero, baizik eta "blokeka". Idazketa ere "blokeka" egiten da: bloke batzuk berridazten dira eta beste guztiak mantentzen dira (“programazio inkrementala” deitzen zaio horri).

  Memoria mota honen erabilera asko zabaldu da, eta ohikoak dira gaur, esaterako, musika irakurgailuetan, datuak gordetzeko (diskete magnetikoak ordezkatzeko), eta abar.

5.6. BESTE MEMORIA BATZUK

Memoria mota asko dago merkatuan, eskuarki aplikazio jakin batzuetara egokituta: prototipoen garapena, konputagailuak, telefonia mugikorra, musika, eta abar. Izan ere, aurreko paragrafoetan sinplifikazio asko eta handiak egin ditugu, kontzeptu nagusiak ahalik eta argien azaltzeko asmoz. Zabala da aukera eta oso bizia merkatua.

Esaterako, konputagailuetan, SSRAM eta SDRAM memoriak erabiltzen dira (S = sinkronoa), prozesadoreen abiadura gero eta handiagoari erantzun egokia emateko. Memoria-gelaxkez gain, hainbat eta hainbat logika gehitu zaie eraginkortasunaren bila; esaterako, sarreretan eta irteeretan, erregistroak (*latch*-ak) daude, informazioa (datuak, helbideak...) gordetzeko; oro har, memoria-edukiera hainbat bankutan banatuta dago (ikus 5.2. ariketa); irakurketa eta idazketa "segmenta" daiteke, eta abar.

Beste memoria batzuk NVRAM (*non volatile RAM*) gisa ezagutzen dira. RAM memoriak dira, baina edukia iraunkorra da. Zenbait kasutan, pila bat duten RAM memoriak baino ez dira; hala, elikadura-tentsioa ez da inoiz galtzen pila hori dela medio (sistema eramangarrietan erabiltzen dira). Beste batzuetan, gailu konplexuagoak dira, bi memoria-matrize integratzen baitituzte: RAM estatiko bat eta EEPROM bat. Prozesu baten hasieran, EEPROMean dagoen informazioa RAM matrizean kargatzen da, eta hor alda daiteke, hala nahi izanez gero. Aldaketa horiek etorkizunerako gorde nahi badira (adibidez, konfigurazioko parametroak), EEPROM memorian idatziko dira. Memoria horiei "*shadow*" ere deritze.

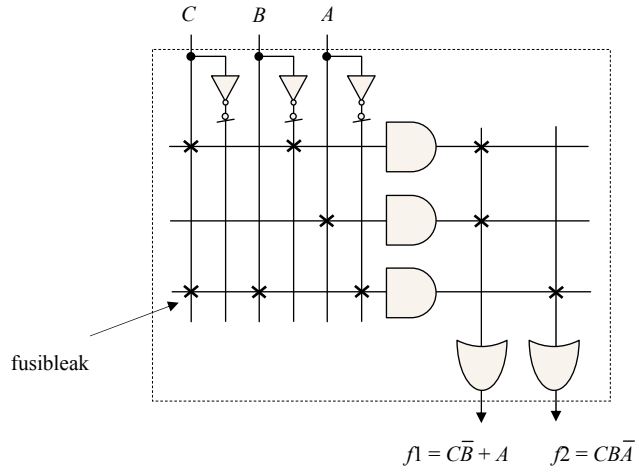
Oso bestelako teknologiak erabiliz ere lor daitezke sistema digitaletarako memoriak. Esaterako, FRAM (*ferroelectric* RAM) edo MRAM (*magnetoresistive* RAM), zeinetan, memoria-gelaxkak sortzeko, ez diren transistoreak erabiltzen baizik eta material magnetikoak.

5.7. GAILU PROGRAMAGARRIAK

Memoria mota nagusiak aztertu ditugu aurreko ataletan. Badaude, hala ere, asko erabiltzen diren beste gailu mota batzuk, memoriak ez badira ere, ROM memorien antzeko egitura dutenak: gailu programagarriak (*programmable devices*). Hainbat izen erabiltzen dira —PLD (*programmable logic device*), PLA (*programmable logic array*), PAL (*programmable array logic*), FPGA (*field programmable gate array*)...— baina antzekoak dira guztiak. Gailu horiek asko erabiltzen dira gaur egun sistema digitalak osatzeko, diseinatzaileak berak eraiki baititzake laborategian sistema digital konplexuak horietako txip bakar batean; nahikoa da, horretarako, diseinuak garatzeko softwarea (zirkuituak marraztu, portaera simulatu, eta abar) eta zirkuitu horiek programatzeko gailu berezia, logika guztia, konbinazionala zein sekuentziala, PLD batean sartzeko.

PLD zirkuituen muinean, ate logikoek eta biegonkorrek osatutako matrize (*array*) programagarriak daude. 5.15. irudian, adibide sinple bat ageri da. Eskuarki, konexio-matrize bi daude PLD batean, bat sarreran eta beste bat irteeran, eta matrize horien gurutze bakoitzean, erabiltzaileak erre dezakeen konexio-puntu bat dago, errenkadaren eta zutabearen arteko konexioa egiteko. Sarrerako matrizeak sarrerak eta hainbat AND ate konektatzen ditu, eta irteerako matrizeak AND eta OR ateak. Funtzio jakin bat sortzeko, erabiltzaileak erreko ditu behar ez dituen konexioak.

PLD zirkuituen programazioa programagailu berezien bidez egiten da, baina prozesua sinple eta azkarra da. 5.15. irudiko adibidean, 3 aldagaiko bi funtzio sortu dira.

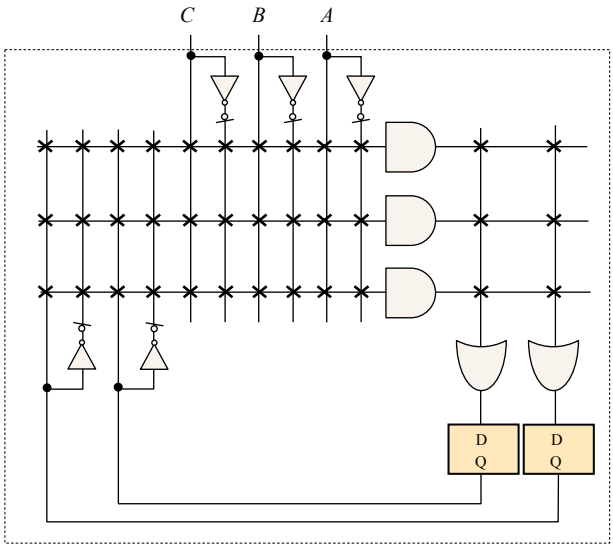


5.15. irudia. PLD baten egitura sinplifikatua, $f1$ eta $f2$ funtzioak sortzeko programatuta.

1. kapituluan azaldu dugun moduan, funtzio logikoak adierazteko modu asko dago, baina erabiliena, seguru asko, SOP (*sum of products,* biderketen batuketa) izenekoa da; hau da, AND eta OR ateen bidez gauza daitekeena. Hori da, hain zuzen ere, PLD zirkuituetan erabiltzen dena, funtzioa minimizatu ondoren (ROM bat erabiltzen denean funtzio logiko bat sortzeko, ordea, funtzioaren egia-taula osoa idatzi behar da memorian, hau da, sarrerako konbinazio guztiak hartzen dira kontuan, funtzioa minimizatu gabe).

5.15. irudiko gailua oso sinplea da, funtzio konbinazionalak baino ezin baititu gauzatu. PLD askotan, ordea, biegonkorrak ere badaude, eskuarki OR ateen irteeretan. Gainera, biegonkorren irteerak sarrera gisa prozesatzen dira,

kanpo-seinaleekin batera, eta, hala, edozein sekuentziadore eraiki daiteke. 5.16. irudian, adibide sinple bat ageri da.



5.16. irudia. Biegonkorrak dituen PLD baten egitura sinplifikatua, oraindik programatu gabe.

Aurreko irudikoak baino egitura askoz konplexuagoak erabiltzen dira gaurko gailu programagarrietan (adibidez, zenbait kasutan, prozesadore sinpleak ere badaude txiparen barruan), eta horrela sortu dira CPLD (*Complex* PLD) izeneko gailuak, baina haien ezaugarri nagusia mantentzen da, hau da, erabiltzaileak "programatzen" du zirkuitua nahi duen sistema logikoa sortzeko. Arruntak dira, era berean, behin baino gehiagotan programa daitezkeen PLD zirkuituak.

Ohikoak dira zirkuitu hauek gaurko aplikazio askotan; esaterako, seinale-prozesamendu digitala egiteko, irudiak prozesatzeko, abiadura handiko komunikazioetan eta abar.

5.8. KONPUTAGAILU BATEN MEMORIA-SISTEMA

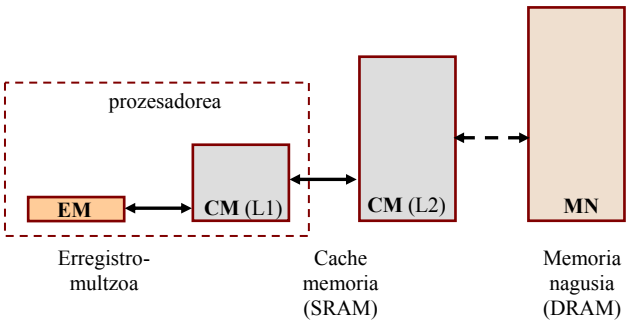
Sistema digital askotan erabiltzen dira memoriak datuak gordetzeko. Gehienetan, memorien edukiera eta datuak atzitzeko denbora ez dira parametro erabakigarriak; hau da, memoriak handiak eta azkarrak dira bete behar dituzten funtzioetarako. Datuak atzitzeko denbora eta memorien

tamaina arazoa ez direnean, kapitulu honetan azaldu ditugun memoria motak eta protokoloak ez dira garrantzitsuak; askotan, nahikoa da SRAM memoria bat (edo EPROM/EEPROM bat) erabiltzea.

Badago, hala ere, kasu "partikular" bat: konputagailua. Konputagailuen oinarrizko osagaia da memoria: ezaguna denez, memorian daude programak eta datuak. Beraz, memoria azkarrak eta handiak behar ditugu.

Zoritxarrez, prozesadoreak memoriak baino azkarragoak dira eta abiadura-diferentzia gero eta handiagoa da. Gainera, memoria handiak txikiak baino motelagoak dira. Beraz, memorien abiadurak mugatuko du konputagailuaren abiadura osoa.

Arazo hori arintzeko, konputagailu baten memoria-sistema ez da "homogeneoa": memoria-hierarkia bat osatzen da, eta hierarkiaren maila bakoitzean, teknologia desberdinak erabiltzen dira. 5.17. irudian ageri da konputagailu baten memoria-hierarkia (sinplifikatuta).



5.17. irudia. Konputagailu baten memoria-hierarkia. Hainbat kasutan, L2 mailako cachea ere prozesadorea duen txip berean integratzen da.

Hierarkiaren beheko mailan, erregistro-multzoa dugu. Ohikoa da, esaterako, 128 erregistroko multzo bat izatea gaurko prozesadoreetan. Memoria azkarrena da, baina oso txikia da: 128 hitz bakarrik, adibidez.

Bigarren mailan, cache memoria dugu. Hainbat kasutan, cache-maila bat baino gehiago dago (L1, L2...), batzuk prozesadorearen txiparen barruan eta beste batzuk kanpoan. Cache memoria osatzeko, RAM estatikoa erabiltzen da (SRAM). RAM estatikoa RAM azkarrena da, baina ez da edukiera handienekoa. Beraz, cache memoria konpromiso bat da abiaduraren eta tamainaren artean. Adibidez, ohikoak dira 32 kB-ko SRAM memoriak L1 mailako cache memorietan eta 1 MB-ko SRAM memoriak L2 mailako cacheetan.

Azkenik, memoria nagusia dugu. Memoria nagusia konputagailu baten RAM handiena da. Oro har, RAM dinamikoa da; ez da cachea bezain azkarra, baina handiagoa da. Bestalde, gogoratu: DRAM memorien edukia freskatu behar da, bestela galtzen baita. Adibidez, ohikoak dira 512 MB-ko edo 1 GB-ko DRAM memoria nagusiak gaurko konputagailuetan.

Badaude memoria-maila gehiago konputagailu batean. Datuak zein programak ez badira memoria nagusian sartzen, diskoetan gordetzen dira. Diskoak, hala ere, ez dira erdieroalezko memoriak, teknologia magnetikoak edo optikoak erabiliz egiten baitira.

Memoria-sistema baten hierarkia-maila bakoitza hurrengo mailaren azpimultzo bat da. Hala, hurbilen duen memorian bilatuko ditu prozesadoreak datuak zein aginduak: erregistroetan. Ez badaude hor, hurrengo mailan, cachean, bilatuko ditu; hutsegitean, memoria nagusira joko du, eta abar. Informazioa aurkitzen duenean, hierarkiaren beheko mailan kopiatuko du (erregistroetan edo cachean), hurrengo batean "hurbilago" izateko. Memoria-sistema osoa modu egokian kudeatzea ez da sinplea eta, horregatik, konputagailu guztiek kontroladore eta estrategia bereziak erabiltzen dituzte horretarako.

RAM memoriez gain, ROM memoriak ere erabiltzen dira konputagailuetan. Dakigun moduan, ROM memoriak irakurtzeko baino ez dira. Beraz, erabiltzaileek ezin dute aldatu haien edukia. ROM motako memorietan gordetzen dira, esaterako, konputagailua abiarazten duen programa (BIOS) edo sistemaren funtzionamendurako behar diren parametro batzuk.

5.9. ARIKETA EBATZIAK

Memorien oinarrizko kontzeptu teorikoak azaldu ondoren, egin ditzagun ariketa batzuk kontzeptu horiek argiago uztearren. Lehenengo bietan, memorien egitura fisikoa eta memoria-sistemen antolaketa lantzen da; gero, ROM memorien erabilerak aztertuko ditugu; eta, azkenik, RAM dinamikoen funtzionamendua.

5.1. Ariketa

(a) 64 k × 8-ko SRAM memoria batean, zenbat bit erabili behar dira hitzen helbideak adierazteko? Eta DRAM memoria balitz?

(b) Zer tamainatakoa da 20 biteko helbideak dituen SRAM memoria bat? Eta 27 biteko helbideak dituena?

(c) 1 Mbit edukierako SRAM memoria batek hainbat egitura izan ditzake; esaterako, 1 M × 1 bit, edo 256 k × 4 bit, edo 64 k × 16 bit. Hiru kasu horietarako, konparatu helbideetarako eta datuetarako behar den lerro kopurua (datuen sarrera- eta irteera-lerroak komunak dira).

■

(a) atala

Memoria batean N hitz badaude, helbideak $\log_2 N$ bitekoak dira, eta, alderantziz, p biteko helbideak erabiliz, 2^p hitz helbidera daitezke.

Beraz, 64 k hitzeko SRAM memoria batean, honako helbide-lerro hauek izango ditugu:

→ 64 k hitz = 64×2^{10} hitz = 2^{16} hitz

→ ondorioz, helbideak $\log_2 2^{16} = 16$ bitekoak izango dira: $A_{15} - A_0$

RAM estatikoa denez, memoria-txipak 16 lerro erabiliko ditu helbideak adierazteko.

Baina memoria dinamikoa bada, hitzen helbideak bi zatitan emango dira (ikus 5.4.2. atala): errenkada-helbidea eta zutabe-helbidea. Memoriako 64 k hitzek $2^8 \times 2^8$ matrizea osatzen dute, eta, beraz, hau izango da errenkadak (edo zutabeak) adierazteko bit kopurua: $\log_2 2^8 = 8$ bit.

DRAM memoria-txipak, beraz, 8 lerro bakarrik erabiliko ditu helbideak adierazteko.

(b) atala

Esan berri dugun moduan, p biteko helbideak erabiliz 2^p hitz helbidera daitezke. Beraz,

$p = 20$ bit $\rightarrow 2^{20}$ hitz = 1.048.576 hitz = 1 M hitz

$p = 27$ bit $\rightarrow 2^{27}$ hitz = $2^7 \times 2^{20}$ hitz = 128 M hitz

(c) atala

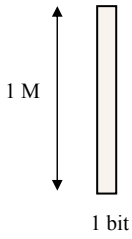
→ 1 M × 1 biteko kasua

1 M = 2^{20}

helbide-lerroak: $\log_2 2^{20} = 20$

datu-lerroak: 1

guztira: 21



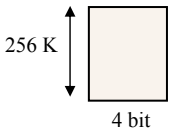
→ 256 k × 4 biteko kasua

256 k = 2^{18}

helbide-lerroak: $\log_2 2^{18} = 18$

datu-lerroak: 4

guztira: 22



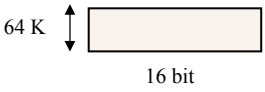
→ 64 k × 16 biteko kasua

64 k = 2^{16}

helbide-lerroak: $\log_2 2^{16} = 16$

datu-lerroak: 16

guztira: 32



Ondorioa garbia da: edukiera bereko memoriak kontuan hartuta, zenbat eta handiagoa izan hitzen tamaina, hainbat eta lerro gehiago erabili behar dira, guztira, helbide eta datuetarako. Joera hori nabarmenagoa da baldin eta datuetarako lerroak ez badira konpartitzen; esaterako, (c) ataleko hirugarren kasuan, 16 + 2 × 16 = 48 lerro erabili beharko genituzke.

«

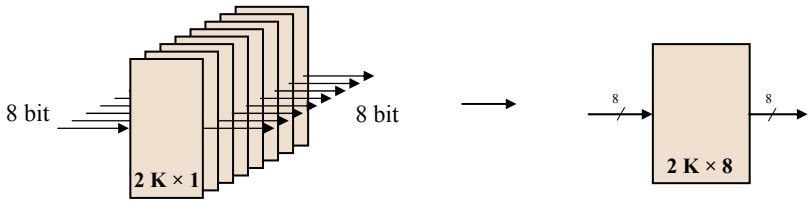
5.2. Ariketa

Tamaina jakineko hainbat memoria-txip erabiliz, tamaina handiagoko memoria-sistemak antola daitezke. Adibidez, eraiki ezazu 16 k × 8 bit (16 kilobyte) edukierako memoria-sistema bat 2 k × 1 bit (2 kilobit) edukierako txipak erabiliz.

Bi arazo konpondu behar ditugu. Alde batetik, hitzen tamaina zabaldu behar dugu, bit batetik 8 bitera. Eta, bestetik, hitz kopurua ere zabaldu behar dugu, 2 k hitzetik 16 k hitzera.

(a) hitzen tamaina zabaltzea

n biteko hitzak lortzeko bit bateko memoria-txipak erabiliz, *n* txip erabili behar ditugu, denak batera. Gure kasuan,



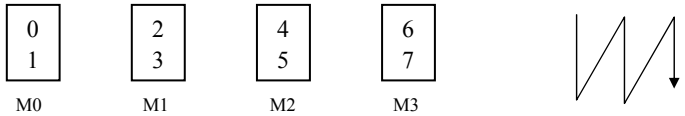
Beraz, 2 k × 8 biteko modulu bat lortzeko, 2 k × 1 biteko 8 txip erabili beharko ditugu.

(b) hitz kopurua zabaltzea

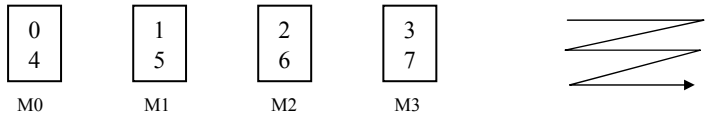
16 k hitz izan behar ditu memoria-sistemak; hori lortzeko, beraz, 2 k hitzeko 8 modulu erabili beharko ditugu. Aukera bat baino gehiago dago, ordea, 8 modulu horiek antolatzeko.

Hiru dira aukera nagusiak; azter ditzagun aukera horiek adibide sinple batez: 8 hitzeko memoria bat osatu behar da, 2 hitzeko 4 txip erabiliz.

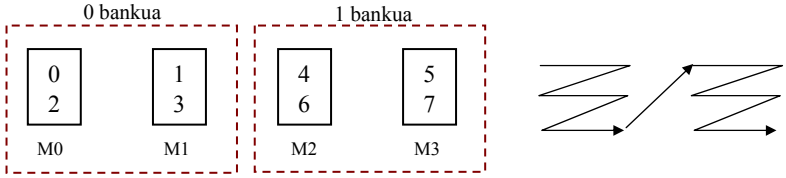
1. **Ondoz ondoko** egitura (*consecutive*): hitzak ondoz ondoko posizioetan kokatzen dira, txip bakoitza bete arte:



2. Egitura **tartekatua** (*interleaved*): hitzak moduluetan banatzen edo tartekatzen dira, banan-banan, irudian bezala:



3. Egitura **mistoa** edo **memoria-bankuak** (*memory banks*): memoria-moduluak multzotan edo **bankutan** antolatzen dira; banku bakoitzaren barruan, egitura tartekatua erabiltzen da eta, bankuen artean, ondoz ondokoa.



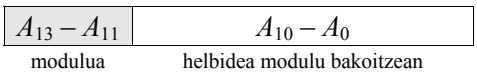
Edozein dela moduluak antolatzeko erabiltzen den egitura, memoriako atzipen bat egiten denean, irakurketa edo idazketa baterako, modulu jakin baten hitz jakin bat aukeratu behar da, hitzaren helbidearen bitartez.

Ariketako kasuan, memoria-sistema osoa 16 k hitzekoa da; beraz, hitzen helbideak 14 bitekoak dira ($\log_2$ 16 k): $A_{13} - A_0$. Modulu bakoitzak, ordea, 11 bit baino ez ditu helbideetarako ($\log_2$ 2 k). Ikus dezagun nola erabili hitzen helbideen 14 bitak aurreko hiru kasuetan.

1. **Ondoz ondoko** egitura

   Hitz baten helbidearen 14 bitak ($A_{13} - A_0$) honela erabili behar dira:

   - Pisu txikieneko 11 bitak ($A_{10} - A_0$) modulu bakoitzean posizio bat aukeratzeko, guztietan posizio bera.
   - Pisu handieneko 3 bitak ($A_{13} - A_{11}$) 8 moduluetatik bat aukeratzeko (gaitzeko).



Adibidez,

- 263 helbidea → 00 0|001 0000 0111

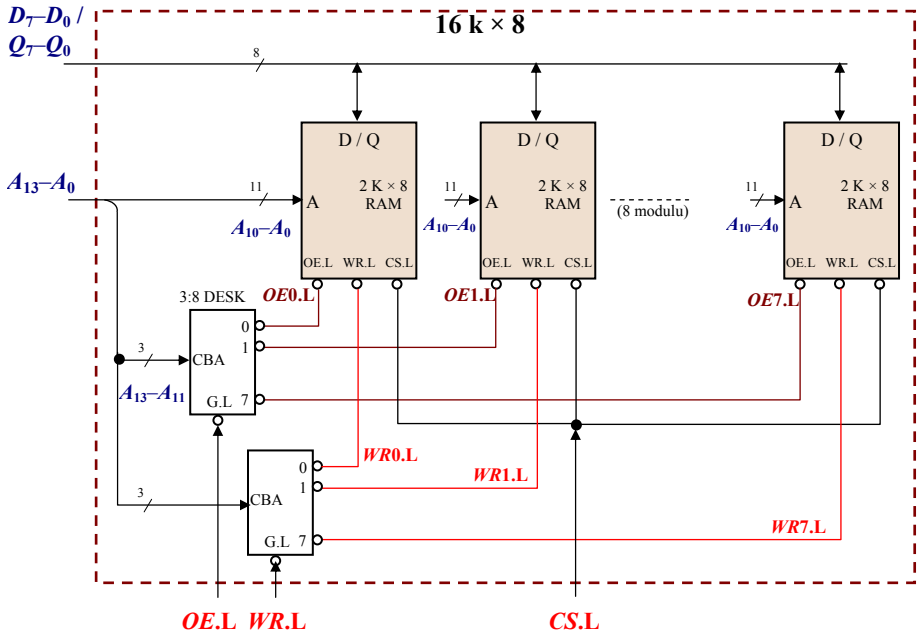
  M0 modulua — 263 helbidea moduluan

- 6160 helbidea → 01 1|000 0001 0000

  M3 modulua — 16 helbidea moduluan[18]

Modulu jakin batean idazteko edo irakurtzeko, haren kontrol-seinaleak aktibatu behar dira, eta, horretarako, modulu bakoitzaren helbidea —hitzen helbideen pisu handieneko 3 bitak— kontuan hartu behar dugu. Hala, deskodegailu bana erabiliz, erraz kontrola daitezke seinale horiek.

Irudian, memoria-sistemaren egitura ageri da. 16 k × 8 biteko memoria kontrolatzeko, ohiko seinaleak ditugu: *OE* irakurtzeko, *WR* idazteko, eta *CS* sistema aktibatzeko.



[18] Hamartarrez adierazita, nahikoa da zatitzea helbidea moduluaren tamainarekin; zatidurak (pisu handieneko bitak) memoria-modulua adierazten du, eta hondarrak (pisu txikieneko bitak) helbidea: 6160 / 2048 → zatidura (modulua) = 3; hondarra (helbidea) = 16

Ikus dezagun, adibide batez, nola funtzionatuko duen memoria:

▪ 263 helbidea → 00 0|001 0000 0111

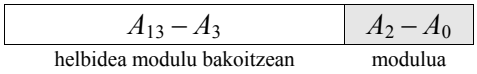
- Irakurtzean, M0 moduluko ***OE*0** seinalea bakarrik egongo da aktibatuta, eta gainerakoak desaktibatuta. Beraz, M0 moduluaren irteera agertuko da memoria-sistemaren datu-irteeran.

- Idaztean, M0 moduluko ***WR*0** seinalea bakarrik egongo da aktibatuta; datua modulu guztien sarreretan badago ere, bakar batean idatziko da: M0-n.

2. Egitura **tartekatua**

Hitz baten helbidearen 14 bitak ($A_{13} - A_0$) honela erabili behar dira:

- Pisu handieneko 11 bitak ($A_{13} - A_3$) modulu bakoitzaren posizio bat aukeratzeko, guztietan posizio bera.
- Pisu txikieneko 3 bitak ($A_2 - A_0$) 8 moduluetatik bat aukeratzeko (gaitzeko).



Adibidez,

▪ 263 helbidea → 00 0001 0000 0|111

32 helbidea — M7 modulua

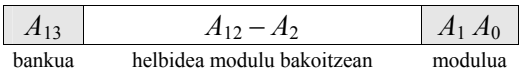
▪ 6160 helbidea → 01 1000 0001 0|000

770 helbidea — M0 modulua

Beraz, aurreko irudiko egiturarekin alderatuta dagoen diferentzia bakarra hau da: $A_{13} - A_3$ bitak erabili behar dira, helbide gisa, memoria-modulu bakoitzean; modulu jakin bat aukeratzeko, ordea, $A_2 - A_0$ bitak eramango ditugu deskodegailuetara.

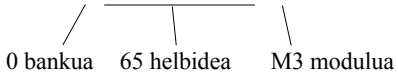
3. Memoria-**bankuak**

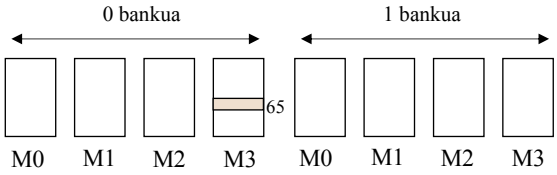
   Kasu honetarako, demagun 4 moduluko 2 banku egin nahi ditugula. Hala, banku bat aukeratzeko, bit bat behar da, helbideen pisu handienekoa, A_{13}. Gero, bankuaren barruan 4 moduluetatik bat aukeratzeko, helbideen pisu txikieneko bi bitak erabili behar dira: A_1, A_0. Gainerako 11 bitak ($A_{12} - A_2$) modulu bakoitzaren helbiderako erabiliko ditugu.



   Adibidez,

   ▪ 263 helbidea → 0|0 0001 0000 01|11





   Berriro ere, aurreko egiturekin alderatuta dagoen diferentzia bakarra helbide-biten erabileran datza. Kasu honetan, $A_{13}A_1A_0$ bitak eramango ditugu deskodegailuetara, banku eta modulu jakin bat aukeratzeko.

Edozein dela aukeratuko dugun egitura, 16 k × 8 biteko memoria-sistema osatzeko, 2 k × 1 biteko txipak erabiliz, 8 × 8 = 64 txip erabili beharko ditugu, hitz kopurua (× 8) eta hitzen tamaina (× 8) zabaltzeko.

«

5.3. Ariketa

ROM memoriak edozein funtzio logiko sortzeko erabil daitezke. Adibide gisa, eraiki itzazu honako funtzioak ROM bat erabiliz:

(a) "BCD – 7 segmentu" deskodegailua

(b) 4 biteko zenbakien karratua ($Y = X^2$)

■

Aukera asko dago funtzio logikoak sortzeko, erabiltzen den hardwarearen arabera; esaterako, funtzioaren adierazpena minimizatu eta ate logikoak erabili eraikitzeko. Nahi izanez gero, ate logikoak baino zirkuitu konplexuagoak erabil daitezke, multiplexoreak edo deskodegailuak kasu. Oro har, funtzio logiko txikietarako baino ez da egokia metodo hori.

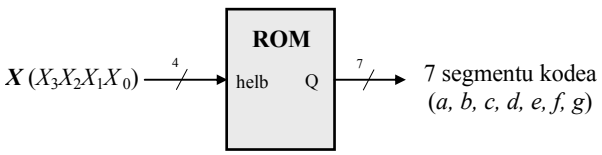
Funtzioak oso konplexuak direnean, ordea, irtenbide sinple bat ROM memoria bat erabiltzea da. Oso bestelako estrategia erabiltzen da kasu horretan funtzio logikoak sortzeko. Funtzioaren aldagaiak memoriaren helbide gisa erabiltzen dira, eta memoria-posizio bakoitzean, helbide horri (hots, funtzioaren aldagaien konbinazio jakin bati) dagokion emaitza (funtzioaren egia-taularen arabera) idazten da. Hala, funtzioaren emaitza eskuratzeko, nahikoa da memoria irakurtzea, dagokion posizioan.

Ikus dezagun adibide sinple pare bat.

(a) atala

"BCD – 7 segmentu" deskodegailua oso arrunta da sistema digital guztietan (ikus 1.8. ariketa), eta badaude zirkuitu bereziak funtzio hori egiteko; beraz, eskuarki ez da egingo memoria baten bidez.

Hala ere, erraza da ROM memoria bat erabiltzea "BCD – 7 segmentu" deskodegailua egiteko. Honako hau egin behar da: zirkuituaren egia-taula memorian gorde. Hala, BCD zenbaki bat (X) 7 segmentu kodera itzuli behar denean, nahikoa izango da memoria irakurtzea.



"BCD – 7 segmentu" deskodegailuaren egia-taula honako hau da:

X	*abcdefg*
0000	1111110
0001	0110000
0010	1101101
0011	1111001
0100	0110011
0101	1011011
0110	1011111
0111	1110000

X	*abcdefg*
1000	1111111
1001	1111011
1010	–
1011	–
1100	–
1101	–
1110	–
1111	–

Funtzioak ez dira minimizatzen eta, beraz, oro har, 16 hitz idatzi beharko ditugu ROM memorian. Kasu honetan, 10 hitz baino ez ditugu idatziko, sarrerako beste 6 konbinazioak ez baitira inoiz agertuko.

Beraz, 10 × 7 tamainako memoria bat behar dugu. 10 hitz helbideratzeko, 4 bit behar ditugu; ondorioz, memoriak 16 posizio izan beharko ditu, nahiz eta azkeneko 6ak ez ditugun erabiliko. Era berean, erabiliko dugun memoriaren hitzak 8 bitekoak badira (ohikoa da byte bateko posizioak izatea memorietan), bit bat erabili gabe utziko dugu, 7 segmentu kodea 7 bitekoa baita.

(b) atala

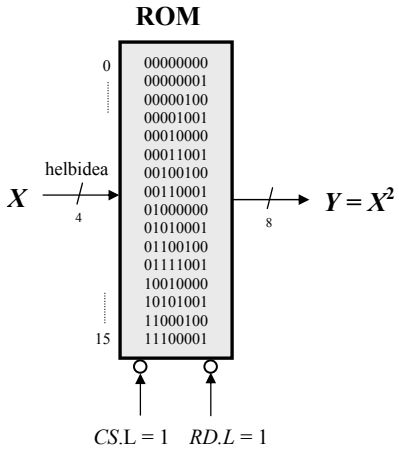
Funtzio logikoak memoria batez sortzeko prozedura beti bera da: funtzioaren egia-taula memorian idatzi behar da. Prozedura ohikoa da funtzio matematikoak sortzeko (logaritmoak, funtzio trigonometrikoak, eta abar). Egin dezagun mota horretako adibide sinple bat: 4 biteko zenbakien karratua sortzen duen zirkuitu bat.

4 biteko zenbakien karratua, oro har, 8 biteko zenbaki bat da. Beraz, 16 × 8 tamainako memoria bat erabili beharko dugu, eta hor idatzi funtzioaren egia-taula:

X	*Y*
0000	00000000
0001	00000001
0010	00000100
0011	00001001
0100	00010000
0101	00011001
0110	00100100
0111	00110001

X	*Y*
1000	01000000
1001	01010001
1010	01100100
1011	01111001
1100	10010000
1101	10101001
1110	11000100
1111	11100001

Beraz, hau da zirkuitua:



(Zirkuitu hori ate logikoen bidez eraikitzeko, 8 funtzio sortu beharko genituzke, irteera-bit bakoitzeko bat, eskuarki minimizatu ondoren).

ROM memoriaren *CS* eta *RD* kontrol-seinaleak aktibatuta utzi ditugu; hala, memoriaren irteera beti erabilgarria izango dugu. Nahi izanez gero, eragiketa egin behar denean bakarrik aktiba daitezke; hala egiten denean, sistemaren kontsumoa baxuagoa izango da, memorien kontsumoa askoz txikiagoa delako desaktibatuta daudenean.

«

5.4. Ariketa

Sekuentziadore batek honako segida honi jarraitu behar dio, kontrol-seinale baten arabera:

$N = 0 \rightarrow 0 - 1 - 3 - 7 - 6 - 4 - 0 - 1 - \ldots$

$N = 1 \rightarrow 0 - 4 - 6 - 7 - 3 - 1 - 0 - 4 - \ldots$

Eraiki ezazu sekuentziadore hori osagai hauek erabiliz: 3 JK biegonkor sekuentziadorearen egoera adierazteko, eta memoria bat egoera bakoitzean biegonkorrak kontrolatzeko behar diren J eta K seinaleak gordetzeko.

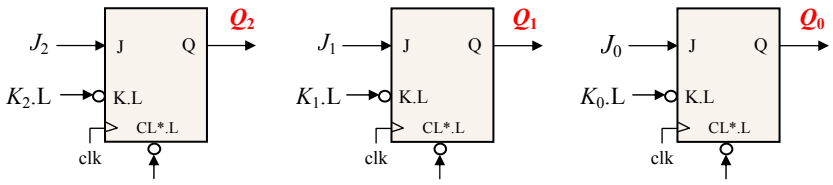
Idatz ezazu memoriaren edukia (irteerak logika positiboan dituela kontuan hartuz) eta adieraz itzazu memoria mota eta tamaina.

■

4. kapituluan aztertu dugu nola eraiki sekuentziadoreak biegonkorrak erabiliz (ikus, adibidez, 4.3. ariketa). Segidaren bit bakoitza biegonkor baten bidez eraikitzen da, eta bit guztiek, lotuta, sekuentziadorearen "egoera" adierazten dute. Gero, sekuentziaren trantsizioak taula batean biltzen dira eta, horien arabera, biegonkor bakoitzaren sarrera-funtzioak erabakitzen dira. Azkenik, funtzio horiek ate logikoen edo maila altuagoko zirkuituen bidez eraikitzen dira.

Esaterako, ROM memoria bat erabil daiteke, aurreko ariketan egin dugun moduan. Sekuentzia sortzeko, beraz, ROM memoria baten edukia irakurriko dugu, zikloz ziklo, sekuentziadorea osatzen duten biegonkorrak "egoera" bakoitzean nola kontrolatu behar diren jakiteko. Egin dezagun proposatzen diguten sekuentziadorea modu horretan.

Sekuentzia hori egiteko, hiru bit behar ditugu: Q_2, Q_1 eta Q_0. Hiru bit horiek hiru JK biegonkorren irteerak izango dira, esaterako, irudikoak. Beraz, sekuentziadorea sortzeko, biegonkor bakoitzaren J eta K sarrera-seinaleak lortu beharko ditugu.



J eta *K* sarrerak eskuratzeko, sekuentziadorearen egoeren arteko trantsizioen taula osatu behar dugu. Hona hemen taula hori, kanpoko kontrol-seinalea, *N*, kontuan hartuz:

Kontrol-seinalea *N*	**Uneko egoera** $Q_2\ Q_1\ Q_0$	**Hurrengo egoera** $Q_2'Q_1'Q_0'$
0	0 0 0	0 0 1
0	0 0 1	0 1 1
0	0 1 0	– – –
0	0 1 1	1 1 1
0	1 0 0	0 0 0
0	1 0 1	– – –
0	1 1 0	1 0 0
0	1 1 1	1 1 0

Kontrol-seinalea *N*	**Uneko egoera** $Q_2\ Q_1\ Q_0$	**Hurrengo egoera** $Q_2'Q_1'Q_0'$
1	0 0 0	1 0 0
1	0 0 1	0 0 0
1	0 1 0	– – –
1	0 1 1	0 0 1
1	1 0 0	1 1 0
1	1 0 1	– – –
1	1 1 0	1 1 1
1	1 1 1	0 1 1

3 bitekoa bada ere, sekuentziadoreak 6 egoera baino ez ditu; beraz, bi egoera —2a eta 5a— ez dira erabiltzen, eta, horregatik, kasu horien hurrengo egoera ez dago definituta (–).

Bit bakoitzari dagozkion trantsizioak egin ahal izateko, modu egokian kontrolatu behar dira biegonkorren *J* eta *K* sarrerak. Gogoratu:

$J, K = 0, 0 \rightarrow Q' = Q$ $\qquad$ $J, K = 1, 0 \rightarrow Q' = 1$

$J, K = 0, 1 \rightarrow Q' = 0$ $\qquad$ $J, K = 1, 1 \rightarrow Q' = \overline{Q}$

edo, baliokidea dena:

$0 \rightarrow 0$: $J, K = 0, –$ $\qquad$ $1 \rightarrow 0$: $J, K = –, 1$

$0 \rightarrow 1$: $J, K = 1, –$ $\qquad$ $1 \rightarrow 1$: $J, K = –, 0$

Taula honetan ageri dira *J* eta *K* sarreren balioak kasu guztietarako.

Kontrol-seinalea *N*	**Uneko egoera** $Q_2\ Q_1\ Q_0$	**Hurrengo egoera** $Q_2'Q_1'Q_0'$	**JK biegonkorren sarrerak** $J_2\ K_2$	$J_1\ K_1$	$J_0\ K_0$
0	0 0 0	0 0 1	0 –	0 –	1 –
0	0 0 1	0 1 1	0 –	1 –	– 0
0	0 1 0	– – –	– –	– –	– –
0	0 1 1	1 1 1	1 –	– 0	– 0
0	1 0 0	0 0 0	– 1	0 –	0 –
0	1 0 1	– – –	– –	– –	– –
0	1 1 0	1 0 0	– 0	– 1	0 –
0	1 1 1	1 1 0	– 0	– 0	– 1
1	0 0 0	1 0 0	1 –	0 –	0 –
1	0 0 1	0 0 0	0 –	0 –	– 1
1	0 1 0	– – –	– –	– –	– –
1	0 1 1	0 0 1	0 –	– 1	– 0
1	1 0 0	1 1 0	– 0	1 –	0 –
1	1 0 1	– – –	– –	– –	– –
1	1 1 0	1 1 1	– 0	– 0	1 –
1	1 1 1	0 1 1	– 1	– 0	– 0

Azkenik, J eta K seinaleak sortu behar ditugu, eta horretarako ROM memoria erabiliko dugu. Memoriaren helbidea $<N\ Q_2\ Q_1\ Q_0>$ 4 bitekin osatuko dugu, eta memoria-posizio bakoitzean helbide horri dagokion informazioa gordeko dugu, taulako 6 bitak: $<J_2\ K_2\ J_1\ K_1\ J_0\ K_0>$ "hitza" hain zuzen ere. Hala, ROM memoriaren irteerak emango digu, kasu bakoitzean, hiru biegonkorrak kontrolatzeko behar dugun informazioa.

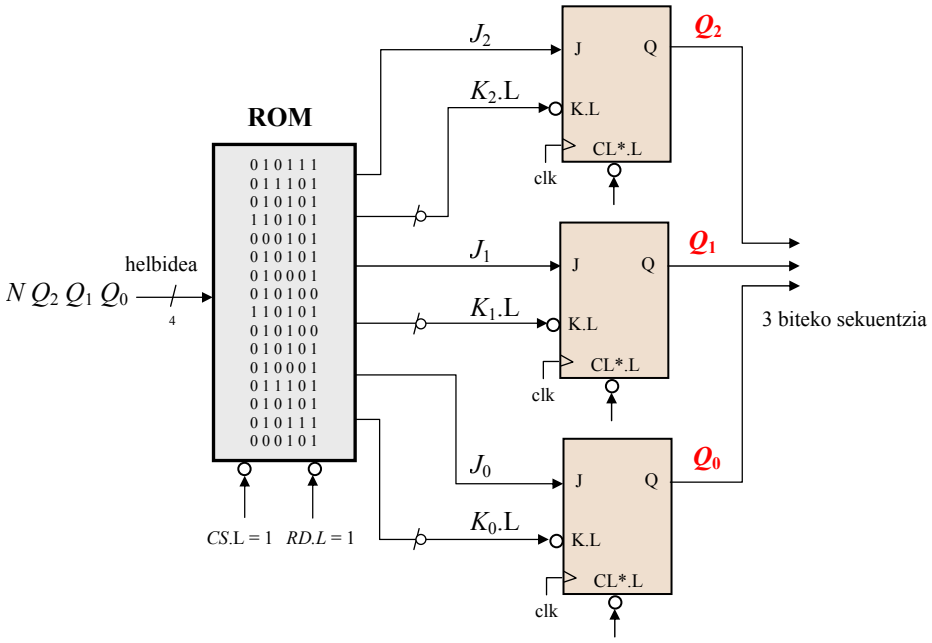
Bi ohar. Batetik, taulako hainbat balio definitu gabe daude. Memoria idatzi behar denean, ordea, balio jakin bat idatzi behar da. Berdin zaigunez, demagun 0koak izango direla.

Bestetik, kontuan hartu behar dugu eraiki nahi ditugun kontrol-seinaleen logika. Adibidez, aukeratu ditugun biegonkorretan, J seinalea logika positiboan dago eta K seinalea logika negatiboan. Memoriaren irteera guztiak, ordea, logika positiboan daude. Beraz, bietako bat: edo egokitzen ditugu K sarrerak NOT ate batez (haien logika aldatzeko), edo, egokiago kasu honetan, K seinaleak gorde beharrean, $\overline{K}$ seinaleak gorde behar ditugu memorian; gogoratu: $\overline{K}$ .H ≡ K.L. Aukera hori hartuko dugu; beraz, $K = 0$ den kasuetan, 1 idatziko dugu memorian (eta alderantziz).

Hau izango da, ondorioz, ROM memoriaren edukia:

HELBIDEA				**EDUKIA**					
N	Q_2	Q_1	Q_0	J_2	$\overline{K}_2$	J_1	$\overline{K}_1$	J_0	$\overline{K}_0$
0	0	0	0	**0**	1	**0**	1	**1**	1
0	0	0	1	**0**	1	**1**	1	0	**1**
0	0	1	0	0	1	0	1	0	1
0	0	1	1	**1**	1	0	**1**	0	**1**
0	1	0	0	0	**0**	**0**	1	**0**	1
0	1	0	1	0	1	0	1	0	1
0	1	1	0	0	**1**	0	**0**	**0**	1
0	1	1	1	0	**1**	0	**1**	0	**0**
1	0	0	0	**1**	1	**0**	1	**0**	1
1	0	0	1	**0**	1	**0**	1	0	**0**
1	0	1	0	0	1	0	1	0	1
1	0	1	1	**0**	1	0	**0**	0	**1**
1	1	0	0	0	**1**	**1**	1	**0**	1
1	1	0	1	0	1	0	1	0	1
1	1	1	0	0	**1**	0	**1**	**1**	1
1	1	1	1	0	**0**	0	**1**	0	**1**

Hona hemen, azkenik, sekuentziadorearen eskema logikoa:



CS eta *RD* kontrol-seinaleak aktibatuta daudenez gero, memoriaren irteera beti erabilgarria izango dugu.

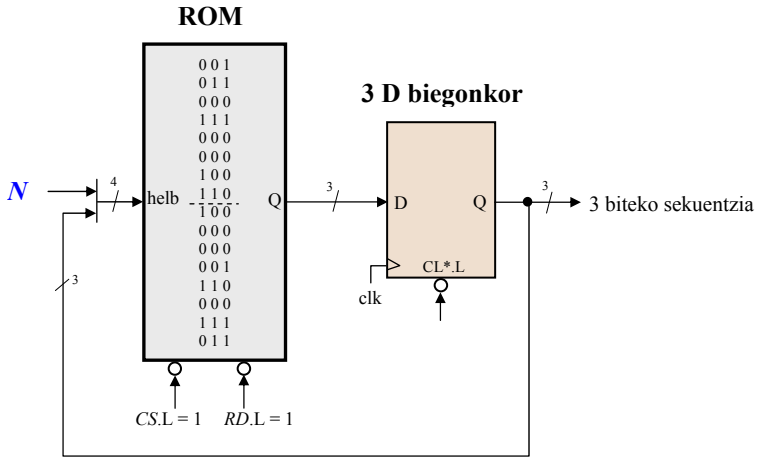
Ohar batzuk ariketa bukatu baino lehen. Batetik, ez ditugu J eta K funtzio logikoak minimizatu, aurreko kapituluko ariketetan egin dugun moduan. Izan ere, ROM memoria bat erabiltzen bada, ez du merezi funtzioak minimizatzea, funtzioaren emaitzak idazten baititugu memorian.

Bestetik, eraiki dugun sekuentziadorea 3 biteko edozein sekuentzia sortzeko erabil daiteke; nahikoa da horretarako memoriaren edukia aldatzea, hots, zirkuitutik atera eta informazio berria, sekuentzia berriari dagokiona, grabatzea.

Azkenik, irudian erabili ez badugu ere, lagungarria da konektatzea biegonkorren CL*.L sarreretan hasieratze-seinale bat, sekuentziadorea egoera jakin batean (esaterako, 0 egoeran) kokatu ahal izateko.

Memoria bat erabiltzen denean sekuentziadore bat egiteko, egokiagoa da D biegonkorrak erabiltzea JK biegonkorrak baino. Izan ere, errazagoa da sekuentziadorearen hurrengo egoera gordetzea memorian, eta ez JK biegonkorrak kontrolatzeko seinaleak.

Hori dela eta, honela egingo genuke sekuentziadore bera, baina hiru D biegonkor (edo, baliokidea dena, 3 biteko erregistro bat) erabiliz:



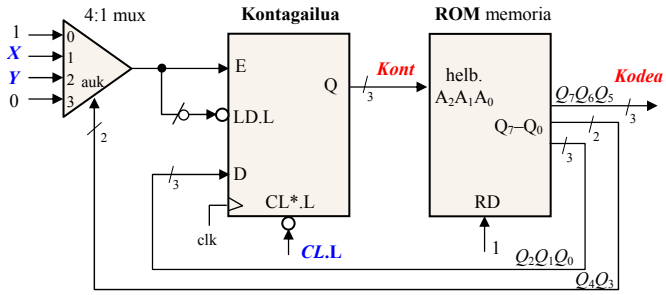
Memoriaren edukia sinplea da; egoera bakoitzerako, zein den sekuentziadorearen hurrengo egoera, *N* seinalea kontuan hartuz.

Ariketaren osagarri gisa, honako hau proposatzen diogu irakurleari: kronograma bana egitea, sortu ditugun bi zirkuituen funtzionamendua egiaztatzeko.

«

5.5. Ariketa

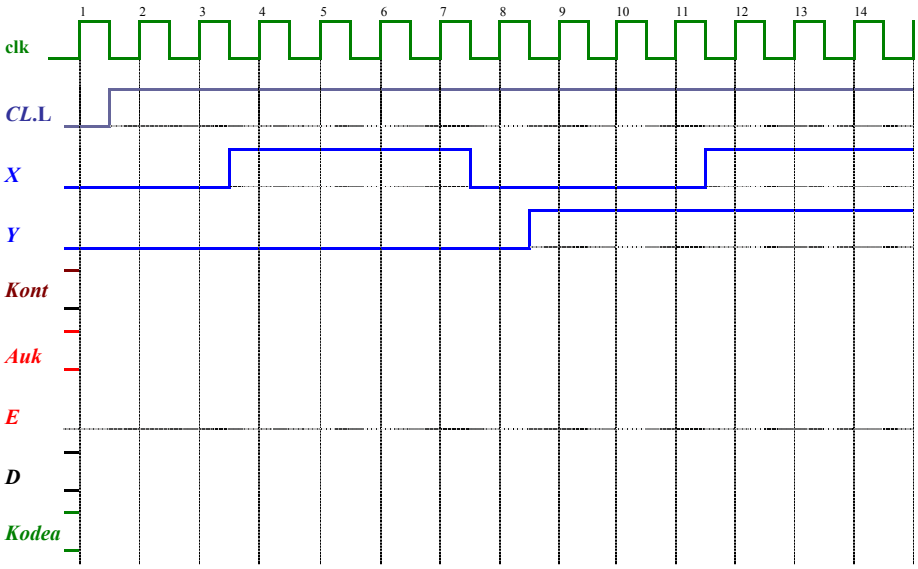
ROM memoria batez eta kontagailu batez egindako 3 biteko sekuentziadore bat ageri da irudian. Kontagailuaren edukia memoria helbideratzeko erabiltzen da; irakurritakoaren arabera, eta kanpoko bi seinaleen arabera (X eta Y), erabaki egiten da zein izango den sekuentziadorearen hurrengo "egoera": segi (E) edo jauzi (LD). Horrez gain, 3 biteko kode bat sortzen da sekuentziadorearen egoera bakoitzean.



memoriaren edukia

helb.	*edukia* (Q_7–Q_0)		
	Kodea	*Auk*	*D*
0	0 0 0	0 1	1 1 1
1	0 0 1	0 0	0 0 0
2	1 0 1	1 0	1 0 0
3	1 0 1	1 1	1 0 1
4	0 1 1	0 0	0 0 0
5	0 1 1	0 0	0 0 0
6	0 1 0	0 1	0 0 1
7	0 0 0	1 1	0 0 0

Memoriaren edukia kontuan hartuz, bete ezazu sekuentziadorearen kronograma hau.



Kronograma bete baino lehen, azter dezagun zirkuituaren funtzionamendua. Kontagailuaren edukia zikloz ziklo aldatzen da, *E* eta *LD* seinaleen balioen arabera. Bi seinaleek kontrako balioa izango dute beti; beraz, $E = 1$ ($LD = 0$) denean, kontagailuaren balioa gehituko da, eta ($E = 0$) $LD = 1$ denean, balio berri bat kargatuko da, D sarrerakoa.

Kontagailuaren edukia erabiltzen da memoria irakurtzeko (helbidea) eta memoriaren edukia erabiliko da gero kontagailua kontrolatzeko; Q_4-Q_3 bitak *E* (*LD*) seinalea zehazten duen multiplexorearen hautatze-seinaleak dira, eta Q_2-Q_0 bitak kontagailuan kargatuko den balio berria.

Auk = 00 denean, 1 konstantea aukeratzen da multiplexorean; beraz, kontagailuaren balioa gehituko da ($E = 1$ / $LD = 0$). *Auk* = 11 denean, ordea, 0 konstantea aukeratzen da ($E = 0$ / $LD = 1$), eta, ondorioz, D sarrerako datua kargatuko da kontagailuan. Beste bi kasuetan, *Auk* = 01 eta 10, *X* eta *Y* aldagaiak aukeratzen dira, eta haien balioen araberakoa izango da kontagailuaren funtzionamendua (gehitu edo kargatu).

Kontagailuaren funtzionamendua kontrolatzeaz gain, kode bat irakurtzen da memorian, ziklo bakoitzean: Q_7-Q_5 bitak.

Egin dezagun kronograma.

1. erloju-ertzean, *CL* seinalea aktibatuta dago, eta, ondorioz, kontagailuak 0 balioa hartuko du. Ziklo horretan zehar, beraz, memoriako 0 posizioa irakurriko da: *Kodea* = 000; *Auk* = 01; *D* = 111. *Auk* = 01 denez, *X* aldagaia aukeratzen da multiplexorean: 0 une horretan; hau da, $E = 0$ eta $LD = 1$ dira kontagailuan: kontagailua prest dago datu bat —*D*— kargatzeko.

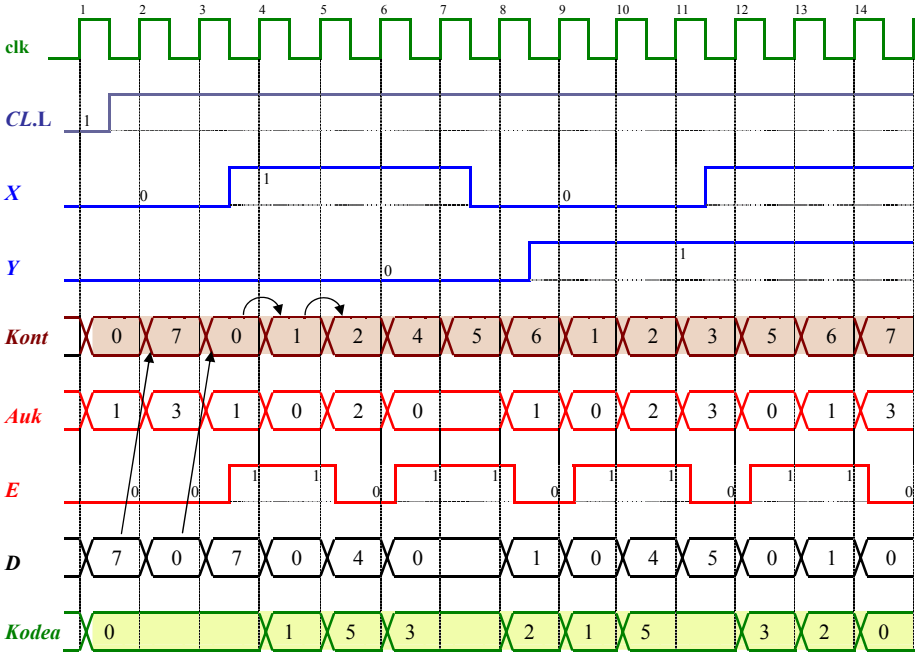
2. erloju-ertza heltzen denean, beraz, 7a kargatuko da kontagailuan, une horretan D sarreran dagoen balioa, hain zuzen ere. Memoriako 7 helbideko posizioa irakurtzen ari da orain: *Kodea* = 000; *Auk* = 11; *D* = 000. Multiplexorean 3 sarrera aukeratu da, 0koa hain zuzen ere (kargatu); beraz, erloju-ertza heltzean, $D = 000$ kargatuko da kontagailuan.

3. erloju-ertza: kontagailuaren balio berria 000 da. Zikloan zehar, memoriako 0 helbidea ari da irakurtzen: *Kodea* = 000; *Auk* = 01 (*X*); *D* = 111. Baina orain $X = 1$ izango da, eta, beraz, kontagailuaren *E* seinalea 1 izango da.

4. erloju-ertza: $E = 1$ denez, kontagailua 0tik 1era aldatzen da. Memoriako 1 helbidea irakurtzen da: *Kodea* = 001; *Auk* = 00; *D* = 000. Konstante bat aukeratu da multiplexorean: 1 balioa. Kontagailuak, beraz, kontatu egingo du (hurrengo erloju-ertza heltzean).

5. erloju-ertza: kontagailua 1etik 2ra aldatzen da. Memoriako 2 helbidea irakurtzen da: *Kodea* = 101; *Auk* = 10 (*Y*); *D* = 100. *Y* aldagaia aukeratu da multiplexorean; *Y* = 0 denez, kanpoko datua (100) kargatuko da.

Modu berean prozesatzen da informazioa gainerako erloju-zikloetan; kronograma honetan bildu ditugu kasu guztiak.



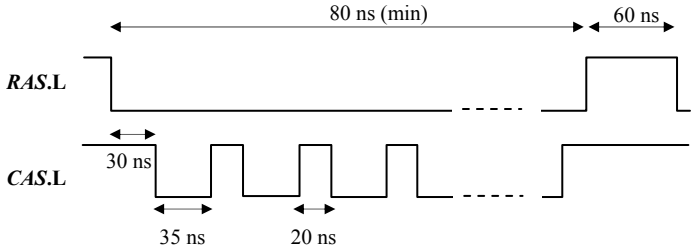
Kapitulu honetan azaldu dugun moduan, beste gailuekin alderatuta, memoriak motelak dira: hau da, haien erantzun-denbora altua da. Beraz, kronograman islatu ez badugu ere, helbidea aldatzen denetik irteera (*Kodea*, *Auk* eta *D*) lortu arte denbora-tarte bat pasatuko da. Erloju-periodoa handia bada (esaterako, 1 ms) denbora hori ez da nabarituko kronograman, baina nabarmen ageriko da periodoa txikia bada (0,5 µs, adibidez).

Hurrengo kapituluan ikusiko dugun moduan, sistema digitalen kontrol-unitateen muinean sekuentziadore bat dago, eta sekuentziadoreak sortzeko ariketa honetan ikusi dugun estrategia askotan erabiltzen da (memoria bat erabiltzen duten sekuentziadoreak “mikroprogramatuta” daudela esan ohi da).

«

5.6. Ariketa

(a) Irudian, DRAM memoria baten hitz batzuk irakurtzeko protokoloaren zati bat ageri da. Azaldu, labur, zein den hitzak irakurtzeko erabiltzen den prozedura.



(b) Memoria 1024 hitzekoa da (1 K). Zenbat denbora beharko da errenkada baten hitz guztiak irakurtzeko modu horretan? Zenbat denbora aurreztuko da, modu estandarreko irakurketarekin alderatuta?

(c) DRAM memorien edukia freskatu behar da periodikoki informazioa ez galtzeko. Freskaketa egiten den bitartean, ezin da memoria erabili. Kasu honetan, freskaketa-prozesua 10 milisegundoan behin egin behar da. Freskaketa-ziklo batek 140 ns irauten du. Zenbat denbora (%) egongo da memoria okupatuta prozesu horretan?

■

(a) atala

DRAM memoria baten hitzek matrize bat osatzen dute, eta haien helbideak bi zatitan banatzen dira: errenkada-helbidea eta zutabe-helbidea. Hala, hitz bat irakurtzeko ohiko protokoloa bi fasetan banatzen da. Lehenengoan, *RAS* seinalea aktibatzen da eta hitzaren errenkada-helbidea ematen da helbide-busean; bigarrenean, *CAS* seinalea aktibatu eta hitzaren zutabe-helbidea adierazten da. Hala, p biteko helbideak erabili behar badira, nahikoa da $p/2$ biteko helbide-busa erabiltzea.

Protokolo osoa betetzeak denbora eskatzen du; hala ere, zenbait kasutan, protokolo horren sinplifikazio batzuk erabil daitezke. Esaterako, irakurri behar badira jarraian errenkada bereko hainbat hitz, *page mode* izeneko irakurketa erabil daiteke: lehen fasean, hitzen errenkada adierazten da, berdina hitz guztietarako, *RAS* seinalearekin batera; eta, gero, hitz bakoitzaren zutabe-helbidea, *CAS* seinalea behin eta berriz aktibatuz. Azkenik, bi seinaleak, *RAS* eta *CAS* desaktibatzen dira, eragiketa bukatzeko.

Hain zuzen ere, aukera hori ageri da aurreko irudian.

(b) atala

Memoriak 1024 hitz baldin baditu, hitzek osatzen duten matrizearen tamaina 32 × 32 izango da; hau da, 32 errenkada eta 32 zutabe. 1024 hitz helbideratzeko $\log_2 1024 = 10$ bit erabili behar direnez, pisu handieneko 5 bitek errenkada adieraziko dute eta beste bostek zutabea.

Adibidez: 624 helbideko hitza
helbidea (624 bitarrean): 10011 10000
beraz, errenkada → 10011 (19); zutabea → 10000 (16);

Memoria horren errenkada bakoitzean 32 hitz daude. 32 hitz horiek irakurtzeko behar den denbora kalkulatzeko (*page mode* erabiliz), irudian ageri diren denborak hartu behar ditugu kontuan:

- *RAS* aktibatu denetik, 30 ns igaro behar dira lehendabiziko hitza irakurtzen hasteko.
- Ondoren, hitz bakoitzeko, 35 + 20 ns behar dira, *CAS* aktibatzeko eta desaktibatzeko.

Hau da:

$$30 + (35 + 20) \times 32 = 1790 \text{ ns} = 1{,}79 \ \mu\text{s}$$

hitz bakoitzeko → 1790 / 32 = 56 ns

Adibidez, hitzak 32 bitekoak badira (4 byte), memoriaren irakurketa-abiadura, kasu horretan, honako hau izan da:

$$(32 \times 4 \text{ byte}) / (1790 \times 10^{-9} \text{ s}) = 71{,}5 \text{ MB/s (megabyte segundoko)}$$

Hitzak banan-banan irakurriko bagenitu, protokolo arrunta erabiliz, askoz denbora gehiago beharko genuke:

- Hitz bat irakurtzeko denbora: 80 ns + 60 ns = 140 ns

  Izan ere, gutxienez 80 ns mantendu behar da aktibatuta *RAS* seinalea eta, gainera, bigarren hitza irakurri baino lehen tarte bat utzi behar da ezer egin gabe[19].
- Beraz, hitz guztiak irakurtzeko:

  $32 \times 140 \text{ ns} = 4480 \text{ ns}$

  irakurketa abiadura: $4 \text{ byte} / (140 \times 10^{-9} \text{ s}) = 28{,}6 \text{ MB/s}$

Konparazio baterako, *page mode* erabiliz egindako irakurketa 2,5 aldiz azkarragoa izan da (71,5 / 28,6 = 2,5).

---

[19] Zehatzak izatera, azken hitza irakurtzeko ez dira behar 140 ns, 80 baizik. *Page mode* erabiltzen denean ere, azkeneko hitza irakurtzeko nahikoak dira 35 ns

(c) atala

Freskaketa-ziklo bakoitzean, eskuarki, errenkada baten hitz guztiak freskatu ohi dira. Beraz, 32 freskaketa-ziklo egin behar dira ariketako memorian (32 errenkada × 32 zutabe), errenkada bakoitzeko bat.

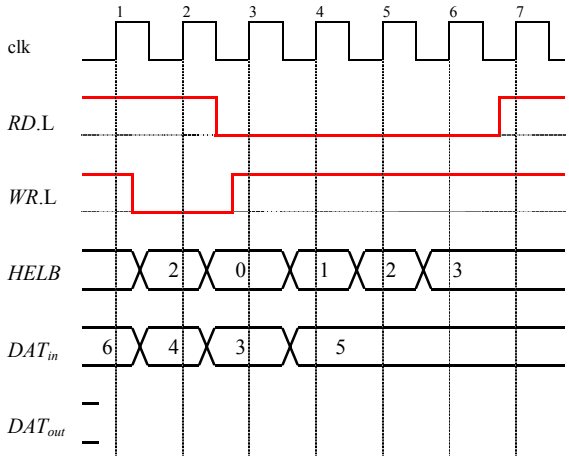
Freskaketa-denbora, beraz: 32 × 140 ns = 4480 ns = 4,48 µs

Hau da, 10 milisegundorik behin (10000 µs), 4,48 µs galduko ditugu. Ehunekoetan: (4,48 / 10000) × 100 = % 0,05.

«

5.10. ARIKETAK

5.1. Hartu kontuan 5.2. irudiko erregistro-multzoa eta bete ezazu kronograma hau. Erregistroen edukia hasieran: R0 → 10; R1 → 0; R2 → 8; R3 → –5.



5.2. Sistema digital baten memoria-sistema 4 k × 8 tamainakoa da, eta 1 k × 8 biteko 4 modulu ditu. Memoria egituratzeko, hiru aukera erabili dira: (1) helbideratze-espazioa ondoz ondokoa da; (2) helbideratze-espazioa tartekatuta dago; eta (3) bi moduluko bi memoria-banku daude.
Kasu bakoitzerako, kalkula itzazu zein modulu eta, modulu horren barruan, zein memoria-posizio dagozkien honako helbide hauei: (a) 650; (b) 1533; eta (c) 3400.

5.3. Sistema digital batek 11 biteko helbideak dituen memoria-sistema bat erabiltzen du. Bi memoria mota daude sistema horretan: 1024–2047 helbideetan, 1 k × 16 biteko SRAM memoria bat; eta 0–127 helbideetan 128 × 16 biteko ROM memoria bat.
Marraztu irudi batean nola osatu memoria-sistema hori eta nola erabili behar diren helbideen 11 bitak bi memoriak atzitzeko. Adierazi, zehatz-mehatz, nola konektatu behar diren memoria bakoitzaren helbideak, datuak eta kontrol-seinaleak. Datu-sarrera eta datu-irteera banatuta daude.

5.4. Eraiki ezazu Y = ∑ (0,5,6,7) eta Z = ∑ (0,2,3,6,7) funtzio multzoa honako zirkuitu hauek erabiliz: (a) NAND ateak; (b) 4:1 multiplexoreak; eta (c) ROM memoria bat (adierazi memoriaren tamaina).

5.5. 64 k × 8 biteko DRAM memoria batean, D = 10011100 datua dago idatzita 20367 helbidean. Prozesu batek irakurtzen du hitz hori eta, pixka bat geroago, 00000000 idazten du posizio berean.
DRAM memorien protokoloak kontuan hartuz, irudikatu kronograma batean nola egingo diren hitz horren irakurketa eta idazketa. Adierazi argi eta garbi helbideak, datuak eta kontrol-seinaleak.

5.6. Hartu kontuan 5.15. irudiko PLDa, 3×3×2 tamainakoa: 3 sarrera, 3 AND ate, eta bi OR ate, irteerak.
Erabil ezazu antzeko egiturako 4×8×4 tamainako PLD bat 2 biteko bi zenbakien biderketa egiten duen sistema konbinazional bat eraikitzeko.
Gogoratu: hasieran, fusible bat (x bat) dago matrizeen gurutze guztietan; programazio-prozesuan, erre egiten dira hainbat fusible, eta funtzioak sortzeko behar direnak bakarrik uzten dira. Hori da, hain zuen ere, marraztu behar dena: zein puntutan utzi behar den konexioa.

6. kapitulua

SISTEMA DIGITALEN DISEINU-METODOLOGIA

Sistema digitalen osagai nagusiak azaldu ditugu aurreko kapituluetan: konbinazionalak —multiplexoreak, deskodegailuak, batugailuak, unitate aritmetiko/logikoak...—, sekuentzialak —D eta JK biegonkorrak, erregistroak, kontagailuak, desplazamendu-erregistroak...— eta memoriak —RAM, ROM...—. Gailu horien funtzionamendua aztertu dugu, eta, bereziki, haiek kontrolatzeko seinaleak.

Une egokia da, beraz, sistema digital oso baten diseinuari ekiteko, eta, horretarako, hainbat gailu lotu eta kontrolatu beharko ditugu portaera jakin bat lortzeko. Aurreko kapituluetan egin den moduan, lehenengo atalean, sistema digitalen diseinu-metodologiari buruzko kontzeptu nagusiak azalduko ditugu. Bigarren atalean, hainbat ariketa ebatziko ditugu, zehatz-mehatz. Kapitulu honen helburu nagusia kontrol-unitateen diseinua lantzea bada ere, ariketa batzuk egingo ditugu sistema digital oso baten diseinuaren prozesua azaltzeko.

6.1. SISTEMA DIGITALEN KONTROL-UNITATEA ETA PROZESU-UNITATEA

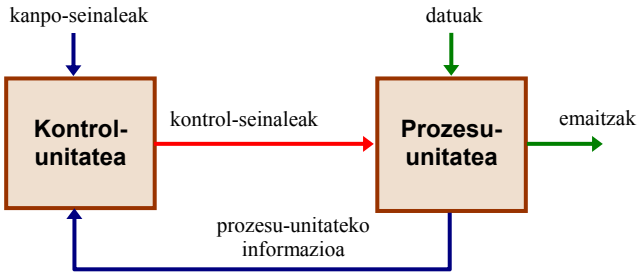
Sistema digitalak konplexuak ohi dira, eta prozedura formalak erabili behar dira diseinatzeko eta analizatzeko, sistema egokiak, egonkorrak eta seguruak sortuko badira. Liburu honen helburuetatik at dago sistema digitalen diseinu-metodologiak eta estrategiak sakonki aztertzea eta lantzea. Hastapenak baino ez ditugu azalduko.

Hala ere, diseinu-estrategia guztiak bat datoz puntu batean. Konplexutasunari aurre egiteko egokiena hau da: banatzea sistema konplexua hainbat azpisistema sinpleagotan. Hala, azpisistema sinpleagoak diseinatzen dira, gero modu egokian konektatu beharko direnak haien artean. Jakina, teknika hori errekurtsiboki aplika daiteke, gero eta azpisistema sinpleagoak definituz. Maiz, *top-down* deitzen zaio diseinu-metodologia horri, konplexuenetik sinpleenera doalako.

Ildo horretan, edozein sistema digital bi zatitan bana daiteke: **kontrol-unitatea** eta **prozesu-unitatea.** Sistemaren barruan funtzio jakin bat betetzen duten gailuek osatzen dute prozesu-unitatea; esaterako, batugailu bat biderkagailu batean, edo RAM memoria bat konputagailu baten memoria-sisteman. Diseinatzaileak modu egokian konektatu behar ditu gailu horiek, berak nahi duen funtzioa egin dezaten. Baina, dakigun moduan, zehatz-mehatz kontrolatu behar dira; adibidez, kontagailu bati adierazi behar diogu noiz kargatu datua, noiz kontatu, kontaketaren noranzkoa, eta abar. Hori bera da, hain zuzen ere, kontrol-unitatearen eginkizuna: prozesu-unitateko gailuen kontrol zehatza.

Bi azpiatal horiek, kontrol-unitateak eta prozesu-unitateak, informazioa trukatzen dute. Kontrol-unitateak kontrol-seinaleak sortzen ditu, prozesu-unitateko gailuek exekuta ditzaten; prozesu-unitateak, aldiz, egindako eragiketari buruzko informazioa itzultzen dio kontrol-unitateari, hurrengo erabakiak har ditzan. 6.1. irudiak laburbiltzen du sistema digital baten oinarrizko banaketa.

Barneko informazioa prozesatzeaz gain, kanpoko informazioa ere prozesatu behar da sistema digitaletan; esaterako, erabiltzailearen aginduak, sentsoreen informazioa, eta abar. Kontrolari dagokion informazioa kontrol-unitatean prozesatuko da; datuak, berriz, prozesu-unitatean.



6.1. irudia. Sistema digital baten oinarrizko banaketa: kontrol-unitatea eta prozesu-unitatea.

Sistema digitalen prozesu-unitateak arras desberdinak izan daitezke, desberdinak baitira sistema digitalen eginkizunak. Beraz, zaila da sistematizatzea nola osatu behar den prozesu-unitate jakin bat. Baina kontrol-unitateen eginkizuna antzekoa da sistema guztietan: prozesu-unitateko gailuen kontrol-seinaleak ordena egokian sortzea, lan jakin bat egiteko. Hori dela eta, hainbat metodo "formal" dago kontrol-unitateak sortzeko. Hurrengo paragrafoetan, horietako bat azalduko dugu.

Horri ekin baino lehen, hala ere, ohar bat. Sistema digitalak **sinkronoak** edo asinkronoak izan daitezke. 4. kapituluan aipatu dugun moduan, kontrol-seinale nagusi batek kontrolatzen du sistema sinkronoen bilakaera, **erlojuak** (*clock*). Sistema digital asinkronoetan, berriz, ez dago kontrol-seinale orokor eta nagusi bat; neurri batean, sistema "autokontrolatzen" da. Sistema asinkronoen kontrola eta, oro har, diseinua konplexuagoa da, eta ez dugu hemen aztertuko.

6.2. KONTROL-UNITATEAREN DISEINUA

Arestian aipatu dugun moduan, kontrol-unitateak erabakitzen du, une oro, zer egin behar den, eta, horretarako, prozesu-unitateko gailuen funtzionamendua kontrolatzen du, hainbat kontrol-seinaleren bidez.

Oro har, kontrol-unitatea **egoera finituko automata** edo makina bat da: aurreikusita dauden ekintzak betetzen ditu, behin eta berriz, eta haren funtzionamendua ziklikoa eta etengabea da. Une bakoitzean, kontrol-unitatea **egoera** jakin batean dago eta egoera horri dagozkion kontrol-seinaleak sortzen ditu. Egoera horretan mantenduko da automata erloju-ertza iritsi arte; une horretan, beste egoera batera aldatuko da (edo zegoenean jarraitu), prozesu-unitatetik zein kanpotik heldutako informazioaren arabera.

Laburbilduz, kontrol-unitate oro hainbat egoeraren artean "mugitzen" den automata bat da; egoera bakoitzean, **bi eginkizun** ditu: dagozkion **kontrol-seinaleak sortu** eta zein izango den **hurrengo egoera erabaki.**

Sistema digital jakin baten kontrol-unitatearen eginkizunak adierazteko modu asko dago, baina bi ditugu erabilienak: ASM (*Algorithmic State Machines*) grafoak, eta hardwarea deskribatzeko lengoaiak erabiliz idatzitako prozedurak (esaterako, VHDL; ikus E3 eranskina). Lehenengoa erabiliko dugu kapitulu honetan.

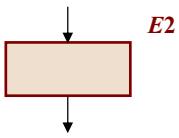
6.2.1. Kontrol-algoritmoak: ASM grafoak

ASM grafo batek kontrol-algoritmo edo -prozedura bat adierazten du; hots, une bakoitzean aktibatu behar diren kontrol-seinaleak eta hartu behar diren erabakiak zehazten ditu. Sistema digital sinkronoen kontrol-automaten eginkizunak adierazteko erabiltzen dira ASM algoritmoak, eta oso elementu gutxirekin osatzen dira.

Hauek dira ASM grafoen osagaiak:

- **Egoerak**

  ASM grafoen osagai nagusiak dira egoerak, automataren egoera adierazten baitute. Lauki batez adierazten dira grafoan. Adibidez:

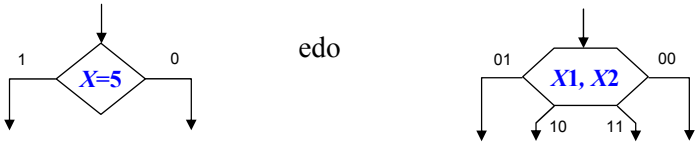
  

  Egokia da izen bat esleitzea egoerei (aurreko irudian, *E*2) identifikazioa erraza izateko.

  Gogoratu: automata egoera jakin batean dago beti, eta egoera horretan erloju-ziklo bat egongo da; erloju-ertza heltzean, beste egoera batera joango da, edo zegoen egoeran mantenduko da. Grafoak, beraz, egoeren arteko "ibilbideak" adierazten ditu.

- **Sarrerak (sarrera-aldagaiak)**

  Erabakiak hartzeko prozesatzen diren seinaleak dira, bai hurrengo egoera zein izango den erabakitzeko, bai sortu behar diren kontrol-seinaleak zehazteko. Erronbo (edo antzeko egitura) batez adieraziko ditugu. Esaterako,

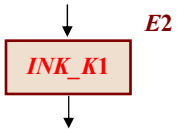


Oro har, hainbat sarrera prozesa daitezke egoera batean.

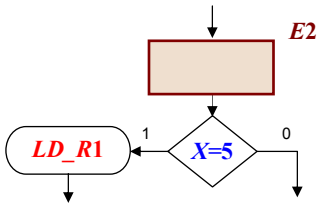
- **Kontrol-seinaleak (irteerak)**

  Edozein kontrol-automataren helburua kontrol-seinaleak sortzea da. Bi motatako kontrol-seinaleak bereizi ohi dira:

  + **Baldintza gabekoak.** Adierazten diren egoeretan, **beti** aktibatu behar dira, inongo baldintza kontuan hartu gabe. Oro har, egoera adierazten duen laukiaren barruan idatziko ditugu. Esaterako,



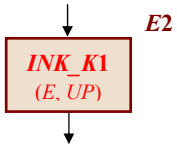
  + **Baldintzapekoak.** Adierazten diren egoeretan, ez dira beti aktibatu behar, kontuan hartu behar baitira egoera horretan prozesatzen diren sarreren balioak ere. Oro har, obalo (edo antzeko egitura) batez adieraziko ditugu.



Kontrol-seinaleak **noiz aktibatu** behar diren adierazten da kontrol-algoritmoan. Ez bada ezer adierazten, orduan seinaleak desaktibatuta (0) egongo dira.

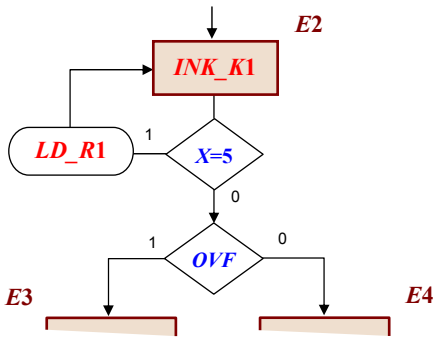
Edozein izen erabil daiteke kontrol-algoritmoan kontrol-seinaleak adierazteko; esaterako: *Kargatu_R, Irakurri_Mem, Ink_Kont, Auk_D, Desplazatu_Ezk...* Hala ere, sortu behar diren kontrol-seinaleek zirkuituetan erabiltzen direnak izan behar dute: hautatze-seinalea multiplexore batean (AUK), karga-seinalea erregistro batean (LD), gaikuntza-seinaleak (G edo CS), kontaketaren noranzkoa kontagailu batean (UP), desplazamendu-erregistroen S1 eta S0 seinaleak, eta abar.

Beraz, kontrolatu nahi diren gailuen kontrol-seinaleetara "itzuli" beharko dira, azkenean, kontrol-algoritmoko seinaleak ("esanahia" duten izenak). Lagungarria izan daiteke, hortaz, benetan sortu behar diren seinaleak ere adieraztea. Adibidez,



Azkenik, kontrol-seinale horiek eraiki behar direnean, erabiliko diren gailuen kontrol-sarreren logika (.H edo .L) kontuan hartu beharko dugu.

Osagai guztiak kontuan harturik, hau izan daiteke ASM algoritmo baten egoera "orokor" bat:



Honako eginkizun hauek ditu automatak irudiko ***E2*** egoeran dagoenean.

Batetik, kontrol-seinaleak sortzen ditu:

+ *INK_K*1 kontrol-seinalea sortzen du, beti. Baldintza gabeko kontrol-seinale bat da.
+ Baldin *X*=5 seinalea 1 bada, orduan *LD_R*1 kontrol-seinalea ere sortzen du. Baldintzapeko kontrol-seinale bat da.

Eta, bestetik, automataren hurrengo egoera erabakitzen du:

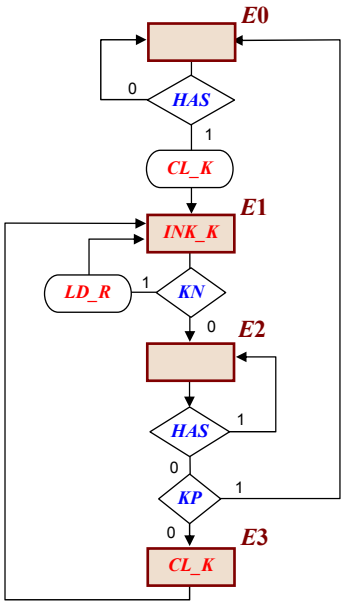
+ *X*=5 seinalea 1 bada erloju-ertza iristen denean, egoera berean mantenduko da automata hurrengo erloju-zikloan zehar.
+ Bestela, hurrengo egoera *E*3 edo *E*4 izango da, *OVF* seinalearen balioaren arabera, grafoan adierazten den moduan.

Beraz, aurreko adibideko *E*2 egoeran, prozesu-unitaterako bi kontrol-seinale sortzen dira, eta bi sarrera aztertzen dira, prozesu-unitatetik zein kanpotik etorriko direnak. ASM algoritmoen egoerak sinpleak edo konplexuak izan daitezke, baina aurreko eskemari jarraituko diote.

Laburbilduz. Sistema digitalak bi azpiatal nagusitan banatzen dira: kontrol-unitatea eta prozesu-unitatea. Kontrol-unitateak prozesu-unitatea kontrolatzen du, eta, horretarako, kontrol-seinaleak sortzen ditu. Kontrol-seinaleen sortzea kontrol-algoritmo baten bidez adierazten da, non sistemaren logika ageri baita. Kontrol-algoritmoa adierazteko aukera nagusiak bi dira; batetik, metodo grafikoak —ASM algoritmoak, esaterako—, eta, bestetik, hardwarea deskribatzeko lengoaiak —adibidez, VHDL—.

6.2.1.1. Egoeren eta kontrol-seinaleen eboluzioa denboran zehar

Esan dugun bezala, egoera jakin batean egongo da automata erloju-ziklo bakoitzean eta, tarte horretan, dagozkion kontrol-seinaleak sortuko ditu. Erloju-ertza iristean, egoerari dagozkion sarrerak prozesatuko ditu eta egoera berria erabakiko du. Portaera hori, grafoaz gain, kronograma batean ere adieraz daiteke. Ikus dezagun adibide bat.

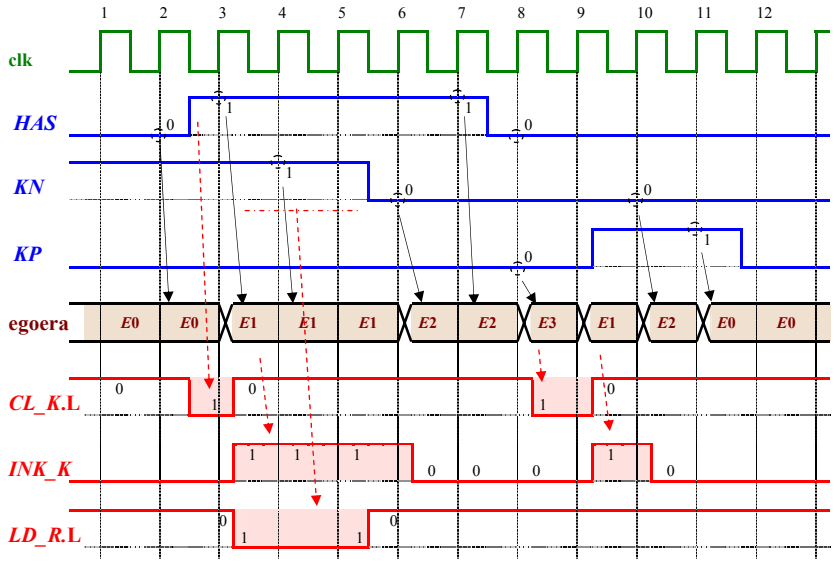


6.2. irudia. 4 egoerako ASM kontrol-algoritmo bat.

6.2. irudiko kontrol-algoritmoak lau egoerako automata bat zehazten du. Hiru sarrera prozesatzen ditu —*HAS*, *KN* eta *KP*— eta hiru kontrol-seinale sortzen ditu: *CL_K*.L —*E*0 egoeran *HAS* = 1 denean, eta *E*3 egoeran—, *INK_K* —*E*1 egoeran— eta *LD_R*.L —*E*1 egoeran *KN* = 1 denean—[20].

Kontrol-algoritmo horri dagokion kronograma, hau da, haren portaera denboran zehar, 6.3. irudian ageri da, sarrera-seinaleen balioen arabera. Automata *E*0 egoeran dago kronogramaren hasieran. Lehenengo eta bigarren erloju-ertzak heltzen direnean, automata ez da egoeraz aldatzen, *HAS* seinalea 0 delako. Baina hirugarrenean bai, *HAS* = 1 delako: automata *E*1 egoerara doa. Gainera, *E*0 egoeran dagoen bitartean, *HAS* sarrera-seinalea aktibatuta dagoela, *CL_K* baldintzapeko kontrol-seinalea aktibatzen du.

Automata *E*1 egoeran mantenduko da *KN* sarrera-seinalea 0 izan arte. 6. erloju-ertzean gertatzen da hori, eta, ondorioz, automata *E*1 egoeratik *E*2 egoerara aldatuko da. *E*1 egoeran egon den bitartean *INK_K* seinalea aktibatu du; horrez gain, eta *KN* seinalea 1 den bitartean, *LD_R* kontrol-seinalea ere aktibatu du. Adi! 5. zikloaren erdialdean edo, *KN* seinalea desaktibatzen da, eta, ondorioz, gauza bera gertatzen da *LD_R* seinalearekin, baldintzapeko kontrol-seinalea baita.



6.3. irudia. 6.2. irudiko kontrol-algoritmoaren kronograma.

[20] Gogoratu: logika negatiboan dauden seinaleak (.L) L tentsioan daude aktibatuta, eta H tentsioan desaktibatuta.

6. zikloan zehar, automata *E*2 egoeran dago; egoera horretan, *HAS* eta *KP* seinaleak prozesatzen dira, hurrengo egoera erabakitzeko. *HAS* = 1 den bitartean, automata *E*2 egoeran mantenduko da; *HAS* = 0 denean, erabakia *KP* seinalearen arabera hartzen da. 8. erloju-ertza heltzen denean, *KP* = 0 denez, automata *E*3 egoerara doa.

*E*3 egoeran, *CL_K* kontrol-seinalea sortzen du; hurrengo egoera beti bera da: *E*1. Hurrengo erloju-zikloetan, egoera hauetatik igaroko da automata: *E*2 (*KN* = 0 delako), eta, azkenik, *E*0 berriro (*HAS* = 0 eta *KP* = 1 direlako).

6.2.1.2. Hasieratze-seinalea

Aipatu dugun bezala, kontrol-unitatea automata finitu bat da; hots, etenik gabe, egoeratik egoerara aldatzen den automata bat. Beste modu batean esanda: automata egoera jakin batean izanik, eta ezagututa sarrera-seinaleen balioak, guztiz zehaztuta dago automataren hurrengo egoera.

Hala ere, ezin dugu ahaztu kontrol-automata bat sistema fisiko baten "abstrakzio logikoa" dela. Kontrol-algoritmoan adierazten den portaera logikoari jarraitzen dio sistema fisiko horrek... funtzionamenduan dagoen bitartean! Baina itzalita dagoenean? Jakina, itzalita dagoenean, sistema digitalak ez du funtzionatzen eta ez dio inongo portaera logikoari jarraitzen. Beraz, pizten dugunean, zein da sistema fisikoak hartuko duen lehenengo egoera? Kontrol-algoritmoaren ikuspuntutik, hasiera-egoera ez dago definituta, ez baitago "aurreko egoerarik"!

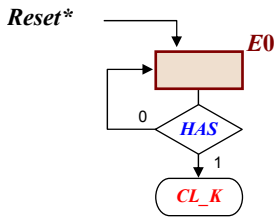
Hori dela eta, modu bat behar da sistema digital baten kontrol-unitatea egoera jakin batera hasieratzeko; hortik aurrera, kontrol-algoritmoaren logikak aginduko du. Behar hori, gainera, ez da sortzen bakarrik sistema "pizten" denean; sistema konplexua den neurrian, akatsak izan daitezke haren funtzionamendu "logikoan", eta, kasu horietan ere, egokia da aukera izatea sistema kontrolpeko bidera ekartzeko.

Hala, sistema digital guztiek seinale berezi bat dute sistema hasieratzeko; hau da, kontrol-unitatea egoera jakin batean kokatzeko. Ohikoa da seinale horri ***Reset*** deitzea. Berez, egoera guztietan aztertu beharko genuke *Reset* seinalearen balioa, aktibatuta baldin balego hasiera-egoerara joateko. Hori egin beharrean, automataren egoera bat hasiera-egoera gisa markatuko da: hasieratze-seinalea aktibatzen denean, hara joango da automata, edozein izanik uneko egoera eta eginkizunak.

Aipatu dugu arestian sistema sinkronoak landuko ditugula, non kontrol-seinale nagusi bat dagoen: erlojua. Erloju-ertza ez bada iristen, sistema ez da

egoeraz aldatuko. Hala ere, salbuespen bat egingo dugu hasieratze-seinalearekin (*Reset*). Izan ere, hurrengo atalean ikusiko dugun moduan, ohikoa da hasieratze-seinalea asinkronoa (eta logika negatibokoa) izatea, biegonkorren sarrera asinkronoak (eskuarki *clear** sarrera) erabili ohi baitira sistema hasieratzeko.

Adibidez, aurreko algoritmoaren hasiera-egoera *E*0 izatea nahi badugu, honela adieraziko dugu kontrol-algoritmoan:



Beraz, *Reset** seinalea edozein unetan aktibatzen dela, automata E0 egoerara joango da. Hortik aurrera, kontrol-algoritmoan adierazitakoa bete beharko da.

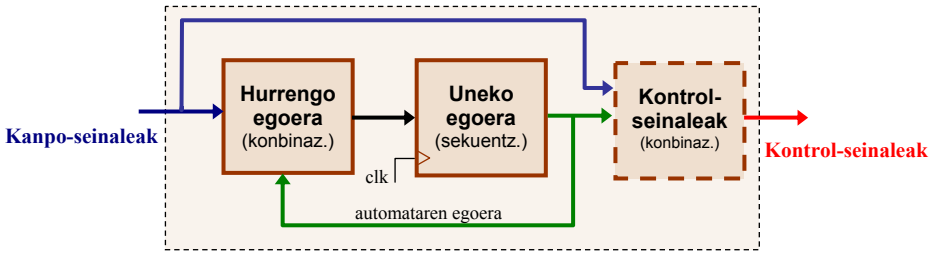
6.3. KONTROL-ALGORITMOEN GAUZATZEA

Kontrol-algoritmoetan, kontrol-unitatearen logika laburbiltzen da, eta, aipatu dugun moduan, bi eginkizunetan banatzen da logika hori: egoera bakoitzean sortu behar diren **kontrol-seinaleak** zehaztea eta automataren **hurrengo egoera** erabakitzea. Logika hori, gero, zirkuituen bidez gauzatu behar da sistema digitala osatzeko. Hainbat metodo daude kontrol-unitateak sortzeko, eta hemen horietako bat azalduko dugu: **multiplexoreen metodoa.**

Nolanahi ere den, metodo guztien oinarrian ideia bera dago: biegonkorrak (edo erregistroak) erabiltzen dira automataren egoera (**uneko egoera)** adierazteko**,** eta logika konbinazionala **hurrengo egoera** adierazten duten funtzio-logikoak sortzeko, bai eta, oro har, kontrol-seinaleak sortzeko. 6.4. irudian ageri da kontrol-unitate baten eskema logiko orokorra.

Bi funtzio horiek –uneko egoera eta hurrengo egoera— sortzeko erabiliko ditugun zirkuituen arabera, eraikitze-metodo desberdinak izango ditugu.

Azter dezagun, pausoz pauso, nola eraiki ASM kontrol-algoritmo batez adierazitako kontrol-unitate bat.



6.4. irudia. Kontrol-unitateen eskema logiko orokorra.

6.3.1. Egoeren kodeketa

Multiplexoreen metodoa erabiltzen denean, lehen pausoa egoerak kodetzea da; hots, kode bitar bat esleitzea egoera bakoitzari. Hala, esaterako, 20 egoerako kontrol-algoritmo batean 5 biteko kodeak erabiliko ditugu egoerak kodetzeko (4 bitekin ez da nahikoa, 16 egoera baino ezin baitira kodetu). Oro har, n egoera kodetzeko, $\log_2 n$ bit erabili behar dira.

Edozein kodeketa egin daiteke (diferentzia oso txikiak sortzen dira kodeketaren arabera, eta ez dira esanguratsuak). Kapitulu honetako adibideetan, Ex egoerari x kodea dagokio beti; esaterako: $E0 \rightarrow 0$ (0...00); $E1 \rightarrow 1$ (0...01), eta abar.

6.3.2. Egoera-trantsizioen taula

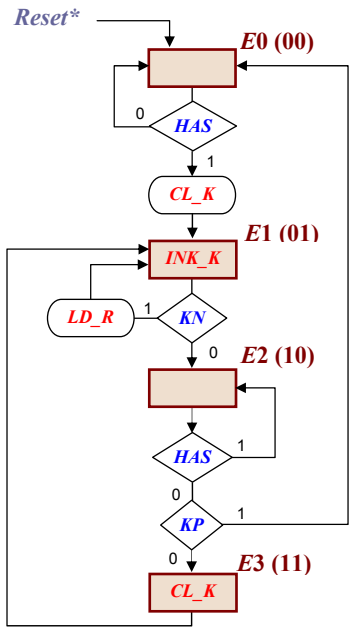
ASM algoritmoak modu grafikoan adierazten du kontrol-unitatearen eginkizuna edo logika. Logika hori taula batean bildu behar da, non, egoera bakoitzeko, hurrengo egoera posibleak eta horietara joateko baldintzak ageriko diren:

Uneko Egoera (UE) + **Baldintza** → **Hurrengo Egoera** (HE)

Har dezagun, adibide gisa, 6.2. irudiko kontrol-algoritmoa. Lau egoera dituenez gero, 2 bit behar dira egoerak kodetzeko; 6.5. irudian, kontrol-algoritmoa ageri da, egoerak kodetuta.

Algoritmoan ageri denez, automata $E0$ egoeran dagoenean, bi aukera daude hurrengo ziklorako: $E0$ egoeran jarraitzea, baldin HAS sarrera 0 bada; edo $E1$ egoerara igarotzea, HAS sarrera 1 bada. $E1$ egoeran izanda, hurrengo

zikloko egoera $E1$ edo $E2$ izango da, KN sarreraren balioaren arabera. Eta abar.



6.5. irudia. Egoerak kodetuta dituen kontrol-algoritmoa.

Egoeren arteko trantsizio horiek guztiak taula batean eman daitezke. Uneko egoeraren kodea adierazteko, A, B, C... letrak erabiliko ditugu (salbuespenik gabe, A letrak pisu txikieneko bita adierazten du). Hurrengo egoerarako, aldiz, A', B', C'...

Uneko egoera $B\ A$		**Baldintza**	**Hurrengo egoera** $B'A'$	
$E0$	0 0	$\overline{HAS}$	$E0$	0 0
		HAS	$E1$	0 1
$E1$	0 1	KN	$E1$	0 1
		$\overline{KN}$	$E2$	1 0
$E2$	1 0	HAS	$E2$	1 0
		$\overline{HAS} \cdot KP$	$E0$	0 0
		$\overline{HAS} \cdot \overline{KP}$	$E3$	1 1
$E3$	1 1	–	$E1$	0 1

6.1. taula. Egoeren arteko trantsizioen taula (6.5. irudiko kontrol-algoritmoa).

6.3.3. Kontrol-seinaleak

Kontrol-unitatea automata finitua da, eta egoeratik egoerara doa erloju-ertz bakoitzean. Baina, jakina, egoeraz aldatzea ez da, berez, sistema digital baten kontrol-unitatearen helburua, prozesu-unitatea kontrolatzea baizik, eta horretarako kontrol-seinaleak sortu behar ditu ordena egokian.

6.5. irudiko kontrol-algoritmoak 3 kontrol-seinale sortzen ditu, automataren egoeraren eta sarreren balioen arabera:

- *CL_K* seinalea bi egoeratan aktibatu behar da: *E*0 egoeran, baldin *HAS* seinalea 1 bada; eta *E*3 egoeran, beti.
- *INK_K* seinalea: *E*1 egoeran, beti.
- *LD_R* seinalea: *E*1 egoeran, baldin *KN* sarrera 1 bada.

Hauek dira, beraz, hiru kontrol-seinaleen ekuazio logikoak:

$$CL\_K = E0 \cdot HAS + E3$$
$$INK\_K = E1$$
$$LD\_R = E1 \cdot KN$$

Egoeren arteko trantsizio-taula eta kontrol-seinaleen ekuazio logikoak, batera, kontrol-algoritmoaren baliokideak dira, informazio bera ematen dutelako.

6.3.4. Kontrol-unitatearen eraikuntza

Kontrol-algoritmoaren logika gauzatzen duen kontrol-unitatea eraiki behar dugu: egoeren arteko trantsizioak betetzen dituen automata zein, nagusiki, kontrol-seinaleak.

6.3.4.1. Egoera-sekuentziadorea

Egoera-sekuentziadorea eraikitzeko, honako hardware hau erabiliko dugu:

- **D biegonkorrak** automataren egoera gordetzeko, biegonkor bat egoera-kodeetako bit bakoitzeko. Adibideko kontrol-unitaterako, beraz, 2 biegonkor behar ditugu.

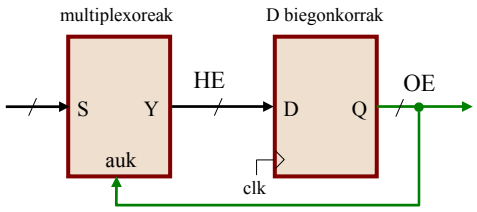
  Gogoratu D biegonkorren portaera: erloju-ertza heltzean, D sarrerako balioa metatzen dute, hurrengo erloju-ertza heldu arte.

  Beraz, sekuentziadorea eraikitzeko, biegonkorren D sarreren balioak eman behar ditugu, sekuentzia egokian, automataren uneko egoeraren eta sarrera-aldagaien balioen arabera.

- Hurrengo egoerak adierazteko, 6.1. taulako *B'* eta *A'* funtzio logikoak eraiki behar ditugu (haien *minterm*-ak kontuan hartuz). Aurreko kapituluetan ikusi dugun moduan, era asko dago funtzio logikoak eraikitzeko. Kasu honetan, multiplexoreak erabiliko ditugu. Biegonkor bakoitzaren sarreran, beraz, *n* sarrerako **multiplexore** bat jarriko dugu, *n* izanik automataren egoera kopurua.

  Multiplexore horien hautatze-seinaleak automataren egoera adierazten duten aldagaiak (*B* eta *A*, biegonkorren irteerak) izango dira. Hala, uneko egoera erabiliko dugu hurrengo egoera aukeratzeko. Erloju-ertza heltzean, multiplexoreen irteera kargatuko da biegonkorretan, automataren hurrengo egoera izateko.

Hau izango da, beraz, egoera-sekuentziadorearen eskema orokorra:



6.6. irudia. Egoera-sekuentziadorea.

Adibidean, 4 egoera dagoenez, 4 sarrerako bi multiplexore erabili beharko ditugu. Multiplexoreen sarrerak adieraztea falta zaigu oraindik, baina hori ez da zaila egoeren arteko trantsizio-taulatik abiatuta.

Multiplexore bakoitzaren *j* sarreran, *Ej* egoerari dagokion hurrengo egoeraren funtzioa kokatu behar da; funtzio hori sortzeko, *Ej* egoerari dagozkion 1ekoak (*minterm*-ak) bakarrik hartu behar ditugu kontuan.

Errepika dezagun hemen adibideko algoritmoari dagokion trantsizio-taula:

Uneko egoera *B A*		**Baldintza**	**Hurrengo egoera** *B' A'*	
*E*0	0 0	$\overline{HAS}$	*E*0	0 0
		HAS	*E*1	0 1
*E*1	0 1	KN	*E*1	0 1
		$\overline{KN}$	*E*2	1 0
*E*2	1 0	HAS	*E*2	1 0
		$\overline{HAS} \cdot KP$	*E*0	0 0
		$\overline{HAS} \cdot \overline{KP}$	*E*3	1 1
*E*3	1 1	–	*E*1	0 1

Automata *E*0 egoeran badago, bi dira aukerak hurrengo egoerarako: 00 eta 01. Beraz, hurrengo egoera adierazteko, *B'* bita 0 izango da beti, eta *A'* bita 0 izango da *HAS* = 0 bada, eta 1 *HAS* = 1 bada; hots, *A'* = *HAS*. Hau da, 0 eta *HAS* funtzio logikoak jarri behar dira multiplexoreen 0 sarreretan.

Analisi bera egin daiteke hurrengo kasuetarako. 1 sarreretan (*E*1 egoerari dagozkion sarrerak), $B' = \overline{KN}$ eta *A'* = *KN* dira. Hain zuzen ere, *KN* = 0 denean, hurrengo egoera *B'A'* = 10 izango da, eta *KN* = 1 denean, *B'A'* = 01.

*E*2 egoerari dagozkion sarreretan, hauek izango dira funtzioak (taulatik aterata): $B' = HAS + \overline{HAS} \cdot \overline{KP} = HAS + \overline{KP}$ eta $A' = \overline{HAS} \cdot \overline{KP}$.

Azkenik, multiplexoreen 3 zenbakidun sarreretan, *B'A'* = 01 jarri behar da, hori baita aukera bakarra.

Multiplexoreen sarreren balioak bildu ditugu taula honetan:

	mux_B	**mux_A**
0 sarrera	0	*HAS*
1 sarrera	$\overline{KN}$	*KN*
2 sarrera	$HAS + \overline{HAS} \cdot \overline{KP} \quad (= HAS + \overline{KP})$	$\overline{HAS} \cdot \overline{KP}$
3 sarrera	0	1

6.3.4.2. Kontrol-seinaleak

Kontrol-seinaleen ekuazioak, oro har, automataren egoeren (biegonkorren irteeren) eta sarrera-seinaleen balioen funtzioa dira (ikus 6.3.3. atala). Nahi den hardwarea erabil daiteke kontrol-seinaleen ekuazio logikoak eraikitzeko. Eskuarki, ekuazio sinpleak izan ohi dira, eta nahikoa da, kasu horretan, ate logiko bakan batzuk erabiltzea.

Askotan, lagungarria da deskodegailu bat erabiltzea automataren egoera deskodetzeko. Hala, deskodegailuaren irteera bakoitzak automataren egoera jakin bat adieraziko du (ikus 6.7. irudia).

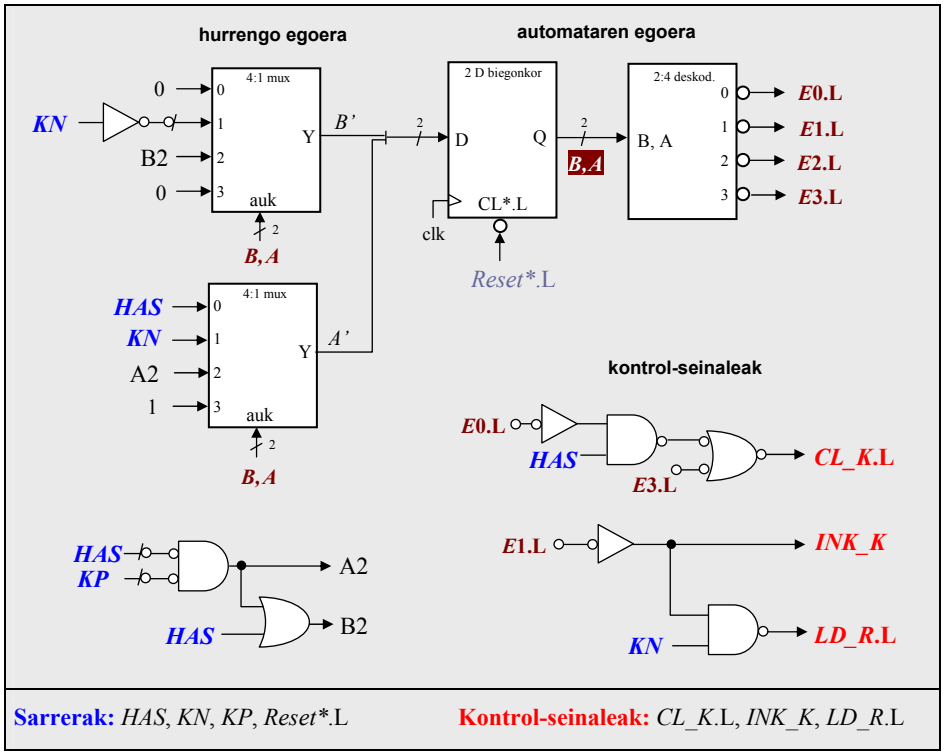
6.3.4.3. Kontrol-unitatea

Kontrol-unitatea marraztea (eta eraikitzea!) baino ez zaigu falta. Esaterako, 6.7. irudian ageri da adibideko kontrol-algoritmoa gauzatzen duen kontrol-unitatea. Aukera bat besterik ez da, modu askotan eraiki baitaiteke, erabiltzen diren zirkuituen arabera.

Irudian ageri denez, 2 D biegonkor erabili ditugu automataren egoera gauzatzeko, egoerak bi bitekoak direnez: *B* eta *A*. 2:4 deskodegailu baten irteeretan (eskuarki logika negatiboan), automataren lau egoerak ditugu.

Hurrengo egoera sortzeko, erloju-ertz bakoitzean biegonkorretan kargatuko dena, 4 sarrerako 2 multiplexoreak ditugu. Sarrera bakoitzean, egoerako trantsizio-taulatik eratorri ditugun funtzioak ipini ditugu, ate logiko sinple batzuk erabiliz. Sarreraren bat aukeratzeko, automataren egoera —hau da, bi D biegonkorren irteerak— erabili dugu hautatze-lerroetan. Automataren uneko egoeraren arabera aukeratuko da, beraz, hurrengo egoera, eta biegonkorren D sarreretan kokatuko da, hurrengo erloju-ertzean kargatzeko prest.

Azkenik, atal garrantzitsuena: kontrol-seinaleak. 3 kontrol-seinale sortu behar genituen, eta nahikoa izan da ate logiko bakan batzuk erabiltzea haien funtzio logikoak eraikitzeko (aukera asko dago funtzio horiek sortzeko; 6.7. irudikoa horietako bat baino ez da).



6.7. irudia. Kontrol-unitatea (6.5. irudiko kontrol-algoritmoari dagokiona). Zirkuituaren konexioak ez nahasteko irudietan, ohikoa da etiketak erabiltzea. Esaterako, multiplexoreen 2 sarreretan A2 eta B2 etiketak erabili ditugu, beherago ageri diren ateen emaitzak, hain zuzen ere.

6.3.4.4. Kontrol-automataren hasieratzea

Lehen aipatu dugun moduan, beharrezkoa da, sistema guztietan, hasieratze-seinale berezi bat izatea —*Reset*—, zeinaren bidez eramaten baita sistema osoa (kontrol-algoritmoa batik bat) egoera jakin batera (eskuarki, sistemaren erloju-seinalea kontuan hartu gabe, asinkronoki).

Nola lortzen da eragiketa hori? Kontrol-automataren egoera adierazten duten gailuetan eragin behar da, hau da, D biegonkorretan. Zorionez, ohikoa da izatea biegonkorretan, seinale sinkronoez gain (D, edo J eta K), seinale asinkronoak ere: *clear* (0ra eramateko) edo *preset* (1era eramateko).

Demagun hasiera-egoera *E*0 dela (ohikoena). Orduan, nahikoa da konektatzea *Reset* seinalea biegonkorren CL*.L sarreretan, 6.7. irudian ageri den moduan. *Reset* aktibatzen den unean bertan, 0ra pasako dira biegonkorrak, eta, ondorioz, automataren egoera *E*0 izango da, edozein zelarik aurreko egoera. CL sarrera logika negatiboan ohi denez, *Reset* ere logika negatiboan beharko dugu; gainera, edozein unetan aktiba daitekeenez, izartxo bat gehituko diogu beti, automataren grafoan zein zirkuituan ageri den moduan: *Reset**.

6.3.4.5. Laburpena

Atal honetan azaldu dugu nola sortu kontrol-unitate bat kontrol-algoritmo batetik abiatuta. Prozedura beti bera da, eta honela labur daiteke:

1. **Kodetu** kontrol-algoritmoko egoerak, nahi den moduan.
2. Zerrendatu taula batean —**egoera-trantsizioen taulan**— zein diren, egoera bakoitzeko, hurrengo egoerak eta egoera horietara joateko bete beharreko baldintzak.

   Egoeraz egoera, eman, egoera-bit bakoitzeko, **hurrengo egoera adierazten duen funtzio logikoa,** uneko egoeraren eta dagozkion baldintzen arabera.
3. Eman **kontrol-seinaleen ekuazio logikoak,** kontrol-algoritmoan adierazten denaren arabera.
4. Kontrol-unitatea sortzeko, hainbat **D biegonkor** (uneko egoera) eta **multiplexore** (hurrengo egoera) erabili, egoera-kodeen bit bakoitzeko bat. Multiplexoreen irteerak biegonkorren D sarreretara konektatu, eta biegonkorren irteerak (automataren uneko egoera) multiplexoreen hautatze-seinaleetara.

Eraiki (eskuarki ate logikoen bidez) **hurrengo egoeretarako** ondorioztatu diren funtzioak eta konektatu multiplexore bakoitzaren sarreretan.

5. Azkenik, **eraiki kontrol-seinaleak** haien ekuazio logikoen arabera (nahi diren zirkuituak erabiliz). Horretarako, maiz lagungarria da gehitzea **deskodegailu** bat biegonkorren irteeran, automataren egoerak deskodetzeko.

6.4. AZKEN AIPAMEN BATZUK

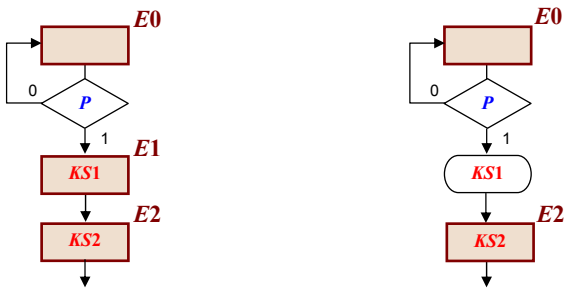
Sistema digital baten kontrol-unitatea diseinatu behar denean, hainbat aukera daude portaera digital jakin bat islatzeko. Esaterako, 12 egoerako kontrol-unitatea edo 16koa? baldintzapeko kontrol-seinalea edo baldintza gabekoa?

6.5 atalean, diseinu-ariketa batzuk landuko ditugu, baina merezi du argitzea orain zalantza horietako batzuk.

6.4.1. Kontrol-automataren egoera kopurua

Sistema digitalen diseinua proba/errore motako prozesu bat da, hainbat birfinketa-pausotatik igaro behar duena. Beraz, kontrol-algoritmoa diseinatu behar dugunean, ez gara kezkatuko, hasieran behintzat, automataren egoera kopuruaz. Hasierako diseinuetan, hainbat egoeratan banatuko dugu kontrolaren logika, sistemaren eginkizunak modu egokian betetzen direla ziur izan arte. Bigarren fase batean, kontrol-ekintzak "taldekatzen" saia gaitezke, egoera kopurua minimizatzearren.

Egoera kopurua minimiza daiteke, oro har, egoera batean sortzen diren kontrol-seinaleak aurreko egoeraren baldintzapeko irteera gisa jarrita, edo egoera desberdinetan banatu diren kontrol-seinaleak bakar batean bilduz. Esaterako,



Bigarren algoritmoan, egoera bat gutxiago erabiltzen da, *KS*1 kontrol-seinalea *E*0 egoeraren baldintzapeko kontrol-seinale gisa jarri baita. Adi! agian baliokideak dira, baina bi kontrol-algoritmo horiek ez dira berdinak (ikus 6.4.3. azpiatala).

Ez ahaztu, hala ere, **"urrezko araua":** kontrol-automatak **zuzena** izan behar du. Hau da, ondo funtzionatu behar du kasu guztietan eta baldintza guztien pean. Hori da kalitate-parametro nagusia, zuzentasuna. Beraz, "12 egoerako automata 16koa baino hobea da?" galdera ez da egokia, ez baita neurtzen kalitatea egoera kopuruaren arabera. Hobea da 16 egoerako automata 12koa baino, baldin eta diseinua zuzenagoa bada, argiagoa, eta kontrolatzeko, mantentzeko eta probak egiteko errazagoa. Adi! Sistema digitalen portaera konplexua izan daiteke, eta zaila frogatzea aukera guztietan ondo funtzionatuko dutela; beraz, behar diren egoera guztiak erabili behar dira, portaera ahalik eta kontrolatuena lortzeko.

Hala ere, askotan (ez beti, ordea) egoera gutxiagoko automatek hardware gutxiago behar dute, eta hori ere hartu behar da kontuan. Gainera, erlojuaren periodoa berdina bada, denbora gutxiago erabili behar da eragiketa batean egoera kopurua txikiagoa bada. Egoera gutxiago izateak esan nahi du, eskuarki, denbora gutxiago (ziklo bat erabiltzen du automatak egoera bakoitzean). Beraz, denbora garrantzitsua bada —adibidez, konputagailu batean—, automata diseinatu ondoren, egokia da sinplifikazio fase batera pasatzea, exekuzio-denbora minimizatzearren (baina erne! kafea saltzeko makinetan, esaterako, milisegundoek ez dute garrantzirik).

Laburbilduz, ez gara gehiegi arduratuko kontrol-algoritmoaren egoera kopuruaz; hori baino gehiago, algoritmoaren zuzentasuna eta egitura argia hartuko ditugu helburu gisa.

6.4.2. Kontrol-seinaleen izaera

Kontrol-unitateak kontrol-seinaleak sortzea du helburu, prozesu-unitatean behar direnak, eta, horretarako, hainbat informazio prozesatzen du, kanpotik zein prozesu-unitatetik datorrena.

Askotan, kontrol-seinaleen exekuzioaren ondorioak kontuan hartu behar dira, kontrol-automatan, hurrengo egoera erabakitzeko, eta oso garbi izan behar dugu haien eragina. Gogoratu: kontrol-seinaleak bi motatakoak dira: "asinkronoak" —hau da, exekutatzen dira heldu ahala, erlojua kontuan hartu gabe—, eta "sinkronoak" —exekutatu ahal izateko, erloju-ertza behar dutenak—.

Esaterako, multiplexore baten hautatze-seinaleak lehenengo motakoak dira, eragina dutelako edozein unetan (gailuari dagokion erantzun-denbora kontuan harturik). Aldiz, erregistro baten karga-seinalea (LD) ez da exekutatzen sortzen denean, hurrengo erloju-ertzarekin batera baizik. Ez ahaztu: erlojua da "kontrol-seinale" nagusia sistema digital sinkronoetan.

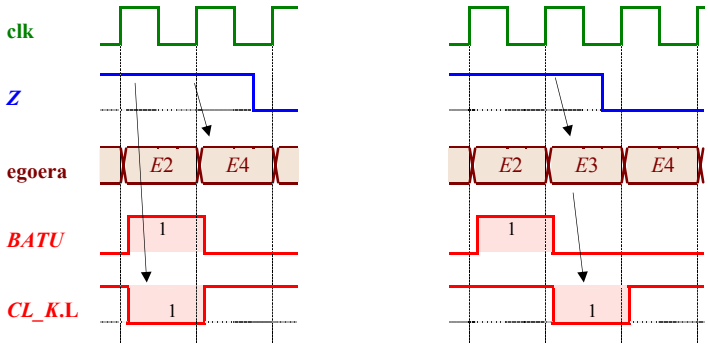
Beraz, automataren pausoak edo egoerak erabaki behar direnean, ezin da ahaztu sortzen diren kontrol-seinaleen izaera. Esaterako, ez da zuzena erregistro bat kargatzea eta kargatu den datuari buruzko informazioa eskatzea egoera (erloju-ziklo) berean (ikus 6.7. ariketa ebatzia); horretarako bi egoera behar dira kontrol-automatan: bat datua kargatzeko, eta beste bat kargatutako datua analizatzeko (edo, bestela, beste modu batean antolatu beharko da eragiketa).

6.4.3. Baldintzapeko eta baldintza gabeko kontrol-seinaleak

Aztertu bi kontrol-algoritmo zati hauek:

Berdinak dira? Baliokideak dira? Garbi dago: ez dira berdinak; baina agian bai baliokideak (egin nahi denaren arabera). Azter dezagun algoritmo horien portaera kronograma banatan (*BATU* logika positiboan eta *CL_K* negatiboan)

Lehenengo kasuan, *CL_K* seinalea automata *E*2 egoeran dagoen bitartean aktibatzen da, une horretan $Z = 1$ delako. Beraz, bi kontrol-seinaleak, *BATU* eta *CL_K*, batera (ziklo berean) sortzen dira. Bigarren kasuan aldiz, kontrol-seinale bakarra sortzen da automata *E*2 egoeran dagoen bitartean: *BATU*. Erloju-ertzarekin batera, $Z = 1$ denez, *E*3 egoerara doa automata, eta egoera horretan sortzen da *CL_K* seinalea.



Beraz, hortxe dago diferentzia, kontrol-seinaleen sortze-ordenan: biak batera edo bata bestearen ondoren. Ordena horrek eraginik ez badu gure sisteman, orduan bi aukerak baliokideak dira; bestela, ez.

6.4.4. Kanpo-seinaleen izaera

Kapituluaren hasieran esan dugun bezala, sistema sinkronoen diseinua ari gara aztertzen. Sistema mota horietan, erloju-seinaleak kontrolatzen ditu sistemaren aldaketa guztiak. Ondorioz, aldaketak gertatzen badira seinaleetan, L tentsiotik H-ra edo H-tik L-ra, une jakin batean gertatuko dira: erloju-ertza iristen denean. Hala ere, seinaleak sortzen dituzten zirkuituek denbora minimo bat behar dute erantzuna emateko, eta, beraz, aldaketak ez dira gertatzen erloju-ertza iristen den unean bertan, baizik eta denbora-tarte bat igaro eta gero.

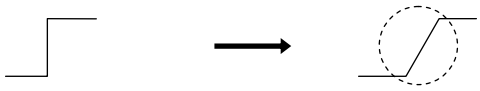
Nolanahi ere den, aldaketarako tarte hori igaro ondoren, seinale guztien balioak egonkorrak izango dira, eta balio berari eutsiko diote erloju-ziklo osoan zehar, hurrengo erloju-ertza heldu arte. Beraz, prozesatu behar direnean, seinale guztiek balio zehatza izango dute: 1 edo 0.

Portaera horrek badu, hala ere, salbuespen bat: kanpo-seinaleak. Sistemara sartzen diren seinaleak ez daude, eskuarki, erloju-seinalearen menpe, eta edozein unetan alda daitezke; sistemarentzat, **asinkronoak** dira. Esaterako, erabiltzaile batek edozein unetan saka dezake pultsadore bat, eta, jakina, seinale hori ez da bat etorriko sistemaren erlojuarekin.

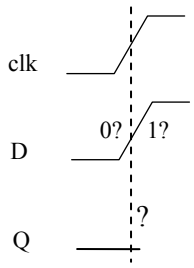
Horrek arazo bat sor dezake: zer balio prozesatuko da erloju-ertza heltzen denean, seinalea aldatzen ari bada une horretan?

Izan ere, seinaleen aldaketa fisikoa, H → L edo L → H, ez da bat-batekoa, denbora jakin bat behar baita trantsizio horietarako. Eskuarki, trantsizioak

kronogrametan bertikalki adierazten baditugu ere, anplifikatuko bagenitu, honela ikusiko genituzke:



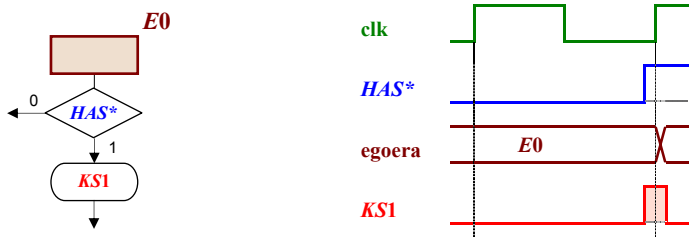
Hori dela eta, zer balio prozesatuko da, esaterako, D biegonkor baten sarreran kasu honetan?



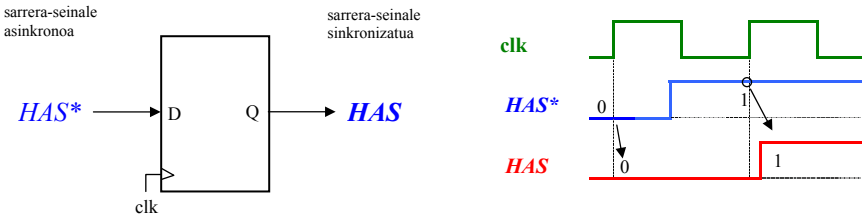
Biegonkorraren irteerako balioa ez dago definituta, eta 0 edo 1 izan daiteke. Kontrol-unitatean biegonkor bat baino gehiago badaude, litekeena da batzuek seinalea 0 gisa prozesatzea, eta besteek 1 gisa. Hala gertatuko balitz, kontrol-unitateak (gailu fisikoek) ez luke kontrol-algoritmoaren logika beteko.

Ikus dezagun adibide bat. *BA* = 00 egoeratik *BA* = 11 egoerara joan behar du automatak, *Z* = 1 bada, edo, bestela, egoera berean geratu. Erloju-ertza gertatzen denean, *Z* seinalea aldatzen ari da 0tik 1era, eta *B* biegonkorrak 1ekotzat hartzen du, baina *A* biegonkorrak 0kotzat. Beraz, *B* biegonkorra 0tik 1era aldatuko da, baina *A* ez; ondorioz, automata *BA* = 10 egoerara joango da, hau da, aurreikusita ez zegoen egoera batera.

Beste arazotxo bat gerta daiteke baldintzapeko kontrol-seinaleekin. Seinale horiek aktibatzen dira sarrera-seinaleen balioen arabera. Ikus irudiko adibidea. *KS*1 kontrol-seinalea aktibatuko da *E*0 egoeran, *HAS** = 1 bada. *HAS* seinalea asinkronoa da (kanpo-seinale bat, adibidez), eta edozein unetan alda daiteke. Beraz, kontrol-seinalearen aktibazio-denbora erloju-ziklo bat baino askoz txikiagoa izan daiteke. Esaterako, irudian ageri den moduan, *HAS** seinalea aldatu egin da erloju-zikloaren bukaera aldean; ondorioz, *KS*1 kontrol-seinalea oso denbora gutxian aktibatu da; nahikoa izango al da?



Bi arazo horietarako irtenbidea ez da zaila: kontrol-unitatean prozesatu baino lehen, kanpo-seinale guztiak **sinkronizatu** egin behar dira sistemaren erlojuarekin. Seinaleak sinkronizatzeko —haien aldaketak erloju-seinalearekin bat etorrarazteko—, D biegonkorrak erabiltzen dira. D sarreretan, kanpo-seinaleak konektatzen dira, eta Q irteeretan, seinale horiek sinkronizatuta izango ditugu. Hala, kontrol-unitateko biegonkor guztiek balio bera prozesatuko dute.



Laburbilduz. Sistema digitalen kontrolari buruzko kontzeptu nagusiak azaldu ditugu aurreko orrialdeetan. Bi ataletan banatu ohi dira sistema digitalak: kontrol-unitatea eta prozesu-unitatea. Kontrol-unitateak sistema osoa kontrolatzen du kontrol-seinaleen bidez. Prozesu-unitateak kontrol-unitateak adierazitakoa exekutatzen du, eta emaitzei buruzko informazioa itzultzen dio.

Kontrol-unitatea automata finitu bat da, egoeraz egoera igarotzen dena. Automata horren logika (sistemaren portaera) adierazteko, ASM diagramak edo kontrol-algoritmoak erabili ditugu. Kontrol-algoritmotik abiatuta, kontrol-unitatea eraiki daiteke, eta, horretarako, D biegonkorrak (automataren egoera) eta multiplexoreak (hurrengo egoera) erabili ditugu. Azkenik, kontrol-unitatearen egoeraren eta sarreren arabera, kontrol-seinaleak sortu behar dira. Izan ere, kontrol-seinaleak sortzea da edozein kontrol-unitateren helburua.

Hurrengo atalean, ariketa batzuk ebatziko ditugu, sistema digitalen diseinua —kontrol-unitateak eta prozesu-unitateak— lantzeko asmoz.

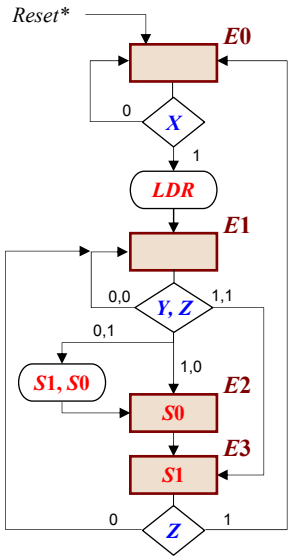
6.5. ARIKETA EBATZIAK

Atal honetan, ariketa batzuk ebatziko ditugu. Lehenengo hiru ariketetan, kontrol-unitateen gauzatzeaz arituko gara, eta, orobat, sistemaren portaera islatzen duten kronogramak sortuko ditugu. Kronograma horien bidez frogatu behar da diseinuaren zuzentasuna, edo, agian, diseinu-erroreak detektatu. Hurrengo hiru ariketetan, diseinu-prozesu osoa landuko dugu: betebehar jakin baterako sistema digital osoa nola sortu. Adibide sinpleak dira, helburua ez baita diseinu konplexuak egitea, baizik eta lantzea liburuan azaldu diren gailuen eta sistemen portaera, eta, batik bat, haien funtzionamendu zuzena ziurtatzea (logika jakin baten arabera). Azken ariketan, sistema digitalak sortzean diseinatzaile berriak egin ohi dituen errore nagusiak aztertuko ditugu. Ariketa ebatziez gain, beste hainbat ariketa proposatzen dira kapituluaren bukaeran, irakurlea saia dadin sistema digitalen diseinua lantzen.

6.1. Ariketa

Sistema digital baten kontrol-unitatearen portaera islatzen da irudiko ASM algoritmoan. Lau egoera ditu —E0, E1, E2 eta E3—, hiru sarrera prozesatzen ditu —X, Y.L eta Z—, eta hiru kontrol-seinale sortzen ditu —LDR.L, S1 eta S0—.

ASM algoritmo horri dagokion kontrol-unitatea sortu behar da, eta haren portaera irudikatu kronograma batean.



Algoritmoan grafikoki adierazten den portaera logikoa laburbilduko dugu taula batean. Hor adierazi behar da, egoera bakoitzerako, zein izango den hurrengo egoera erloju-ertz edo -ziklo bat heltzen denean. Horrez gain, kontrol-seinaleen ekuazio logikoak ere adierazi behar ditugu.

Automataren logika

Kontrol-algoritmoko egoerak kodetzea da lehenengo urratsa. Adibideko automataren lau egoerak kodetzeko, bi bit behar ditugu: *B* eta *A*. Beraz, bi D biegonkor erabiliko ditugu. Honako kode hauek erabiliko ditugu: *E*0 = 00; *E*1 = 01; *E*2 = 10; *E*3 = 11.

Hurrengo urratsa egoera-taula idaztea da. Analiza ditzagun, lehendabizi, egoeren arteko trantsizioak eta sortu beharreko kontrol-seinaleak:

- ***E*0** egoeratik abiatuta, bi aukera ditugu, *X* sarreraren balioaren arabera: *E*0n jarraitzea (*X* = 0 bada), edo *E*1 egoerara igarotzea (*X* = 1 denean).

  Horrez gain, kontrol-seinale bat sortu behar da automata *E*0 egoeran dagoen bitartean baldin eta *X* sarrera 1 bada (baldintzapeko irteera): *LDR* kontrol-seinalea.

- ***E*1** egoeraren analisia antzekoa da. Bi sarrera prozesatzen dira, *Y* eta *Z*, eta haien arabera aukeratu behar da hurrengo egoera. Biak 0 badira, *E*1 egoeran mantenduko da automata; biak 1 badira, *E*3 egoerara igaroko da; eta bata 1 eta bestea 0 badira, *E*2 egoerara joango da.

  Bi kontrol-seinale —*S*1 eta *S*0— sortu behar dira egoera horretan, kasu konkretu batean: *Y* = 0 eta *Z* = 1 direnean.

- ***E*2** egoeraren analisia oso sinplea da. Erloju-ertza heltzen denean, *E*3 egoerara pasatuko da; bitartean, *S*0 kontrol-seinalea aktibatuko du (baldintza gabeko kontrol-seinalea).

- Azkenik, ***E*3** egoeran, *Z* sarrera prozesatzen da. 0 bada, *E*1 egoerara igaroko da automata; bestela, *E*0 egoerara itzuliko da. *E*3 egoeran dagoen bitartean, *S*1 kontrol-seinalea aktibatu behar da.

Trantsizio horiez gain, *Reset** seinalea aktibatzen bada kontrol-unitatea *E*0 egoerara joango da, edozein delarik haren egoera eta, eskuarki, erloju-seinalea kontuan hartu gabe.

6.2. taulan bildu dugu kontrol-algoritmoaren logika. Bi D biegonkor (*B* eta *A*) erabiliko ditugu kontrol-unitatearen egoera eraikitzeko, eta, beraz, 4 sarrerako multiplexoreak hurrengo egoera sortzeko. Multiplexore horien sarrera bakoitza automataren egoera bati dagokio, eta hor kokatuko dugu egoera horri dagokion hurrengo egoera adierazten duen funtzio logikoa.

Uneko egoera *B A*		**Baldintza**	**Hurrengo egoera** *B' A'*	
*E*0	0 0	$\overline{X}$	*E*0	0 0
		X	*E*1	0 **1**
*E*1	0 1	$\overline{Y}\,\overline{Z}$	*E*1	0 **1**
		$\overline{Y}\,Z + Y\,\overline{Z}$	*E*2	**1** 0
		$Y\,Z$	*E*3	**1 1**
*E*2	1 0	–	*E*3	**1 1**
*E*3	1 1	$\overline{Z}$	*E*1	0 **1**
		Z	*E*0	0 0

6.2. taula. Egoeren arteko trantsizioen taula (6.1. ariketako kontrol-algoritmoa). Sistemaren egoera: *BA*; hurrengo egoera: *B'A'*.

6.2. taulan adierazten denaren arabera, hauek izango dira bi multiplexoreen lau sarrerak:

	mux_B	**mux_A**
0 sarrera	0	X
1 sarrera	$Y + Z$	$Y\,Z + \overline{Y}\,\overline{Z} = Y \otimes Z$
2 sarrera	1	1
3 sarrera	0	$\overline{Z}$

Adibidez, automata *E*0 egoeran badago, bietako bat izango da hurrengo egoera: 00 edo 01 (ikus 6.2. taula). Beraz, *B'* (pisu handieneko bita) beti 0 izango da, eta *A'* bita *X* sarreraren arabera (kasu honetan, $A = X$). Oro har, *B'* eta *A'* definitzen dituzten funtzioak lortu behar ditugu. Esaterako, *E*1 egoeran, *B'* funtzioak bi 1eko ditu; beraz, bi baldintzak batu behar ditugu funtzio logikoa sortzeko: $B' = \overline{Y}\,Z + Y\,\overline{Z} + Y\,Z = Y + Z$; eta, era berean, $A' = \overline{Y}\,\overline{Z} + Y\,Z$. Gainerako sarrerak modu bertsuan ondorioztatzen dira egoera-taulatik.

Automatak sortuko dituen kontrol-seinaleak

Aipatu dugun moduan, kontrol-seinaleak sortzea da kontrol-unitateen eginkizuna. Adibide honetako kontrol-unitateak hiru kontrol-seinale sortu behar ditu, sistemaren egoeraren eta sarreren funtzio gisa.

LDR seinalea aktibatu behar da *E*0 egoeran baldin $X = 1$ bada. *S*1 seinalea, bi kasutan: *E*3 egoeran, eta *E*1 egoeran $Y = 0$ eta $Z = 1$ badira. Azkenik, *S*0 seinalea, *E*2 egoeran, eta *E*1 egoeran $Y = 0$ eta $Z = 1$ badira.

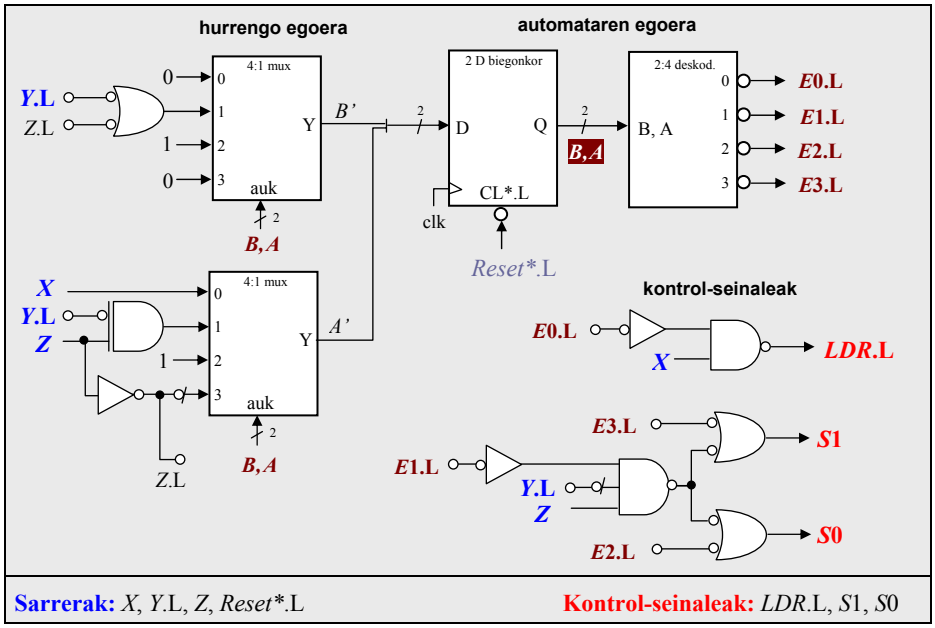
Hona hemen, beraz, hiru seinale horien ekuazio logikoak:

$$LDR = E0 \cdot X$$
$$S1 = E1 \cdot \overline{Y} \cdot Z + E3$$
$$S0 = E1 \cdot \overline{Y} \cdot Z + E2$$

Kontrol-unitatearen gauzatzea

Bi biegonkor eta 4 sarrerako bi multiplexore dira kontrol-unitate honen muina. Lagungarria da deskodegailu bat izatea ere egoerak deskodetzeko; kasu honetan, 2:4 deskodegailua. Horrez gain, ate bakan batzuk baino ez dira erabili behar, eskuarki, multiplexoreen sarreretako funtzio logikoak zein kontrol-seinaleak sortzeko.

Esaterako, hau izan daiteke adibide honetako automata gauzatzen duen kontrol-unitatea:



6.8. irudia. 6.1. ariketako kontrol-unitatea.

Sarrera- zein irteera-seinaleak eraiki behar direnean, erabaki bat hartu behar da: zein logikatan daude? positiboan ala negatiboan? Askotan, askatasun osoa daukagu bata zein bestea aukeratzeko, baina, beste batzuetan, logika jakin bat erabili behar da. Esaterako, aukera zabala bada ere, erregistroen hasieratze-seinalea (CL) edo kargakoa (LD) logika negatiboan ohi datoz. Adibide honetan, honako logika hauek erabili behar ditugu: sarrera bat logika negatiboan (*Y*.L) eta gainerakoak positiboan; irteeretan ere, bat logika negatiboan dago (*LDR*.L) eta gainerakoak positiboan (*S*1 eta *S*0). Edozein erabaki hartuta, hori bai, bat etorri behar dute gailuek behar duten logikak eta seinaleen logikak. Gogoratu: NOT ate bat nahikoa da seinale baten logika egokitzeko, hots, *A*.H → *A*.L edo *A*.L → *A*.H bihurtzeko.

Erloju-ziklo bakoitzean, 6.8. irudiko kontrol-unitatea egoera jakin batetik beste egoera jakin batera igaroko da, sarrera-seinaleen balioen arabera, kontrol-algoritmoaren logika gauzatzeko eta dagozkion kontrol-seinaleak sortzeko. Sistema hasieratzeko erabiltzen den *Reset** seinalea biegonkorren CL*.L sarreretan konektatu dugu (logika negatiboan, beraz). Hala, aktibatzen bada, automata 00 (*E*0) egoerara joango da, edozein egoeratatik eta erloju-seinalea aintzat hartu gabe.

Oharra: funtzionamendu "normalarekin" zerikusia duten seinaleak besterik ez dira ageri aurreko diagraman. Ez dira marraztu, esaterako, multiplexoreen edo deskodegailuaren gaikuntza-seinaleak, berez, beste ezertarako erabiltzen ez badira, aktibatuta mantendu behar baitira. Beste seinale batzuk, aldiz, desaktibatuta mantendu behar dira; esaterako, erabili ditugun biegonkorrek PR*.L seinalea balute, orduan seinale hori desaktibatu beharko genuke (0/H) ez baitugu ezertarako erabiliko.

Kontrol-unitatearen egiaztatzea: kronogramak

Edozein sistema digital eraiki ondoren, simulazio-prozesu bati ekin behar zaio, haren portaera logikoa esperokoa dela egiaztatzeko. Hainbat aplikazio daude simulazio horiek konputagailu batean gauzatzeko. Izan ere, simulazio-prozesuak oso konplexuak izan daitezke, aukera guztiak egiaztatu behar baitira. Sistema logikoak "sinpleak" direnean, "eskuz" analiza daiteke haien portaera, kronograma baten bidez. Kronograma horiek egitea eta ulertzea funtsezkoa da sistema logikoen funtzionamendua eta, bidenabar, simulazio-prozesu konplexuagoak ondo ulertu ahal izateko. Adibide hauek sinpleak

direnez, ez dugu arazorik izango egoera-trantsizio guztiak kronograma batean biltzeko.

Simulazio-prozesuen helburua beti bera da: sistema baten portaera logikoa egiaztatzea sistema hori eraiki baino lehen. Beraz, probatu beharko genituzke kontrol-algoritmoaren aukera guztiak ("ibilbide" guztiak). Aukera horiek, jakina, sarreren balioen araberakoak dira. Kronograma egin ahal izateko, sarrera-seinaleen balioak definitu behar ditugu; ondo pentsatu behar dira sarreren balioen sekuentziak funtzionamendu-kasu guztiak egiaztatu ahal izateko. Adibide honetan, 4 sarrera ditugu: batetik, *Reset**.L seinalea, asinkronoa, eta, bestetik, *X*, *Y*.L eta *Z* seinaleak, sinkronoak. 6.9. irudiko kronograman ageri dira seinale horietarako probatu nahi ditugun balioak.

Egin dezagun, beraz, aurreko kontrol-unitatearen kronograma. Zikloz ziklo analizatuko dugu sistemaren portaera. Prozedura beti bera da: zikloaren hasieran, multiplexoreen irteerak analizatu behar dira, une horretan aukeratuta dauden sarrerak hain zuzen ere, horiek adieraziko baitute automataren hurrengo egoera. Gero, egoera horri dagozkion kontrol-seinaleak sortu behar dira (haien ekuazio logikoak kontuan hartuz). Kronograma osoa 6.9. irudian ageri da. Analiza dezagun.

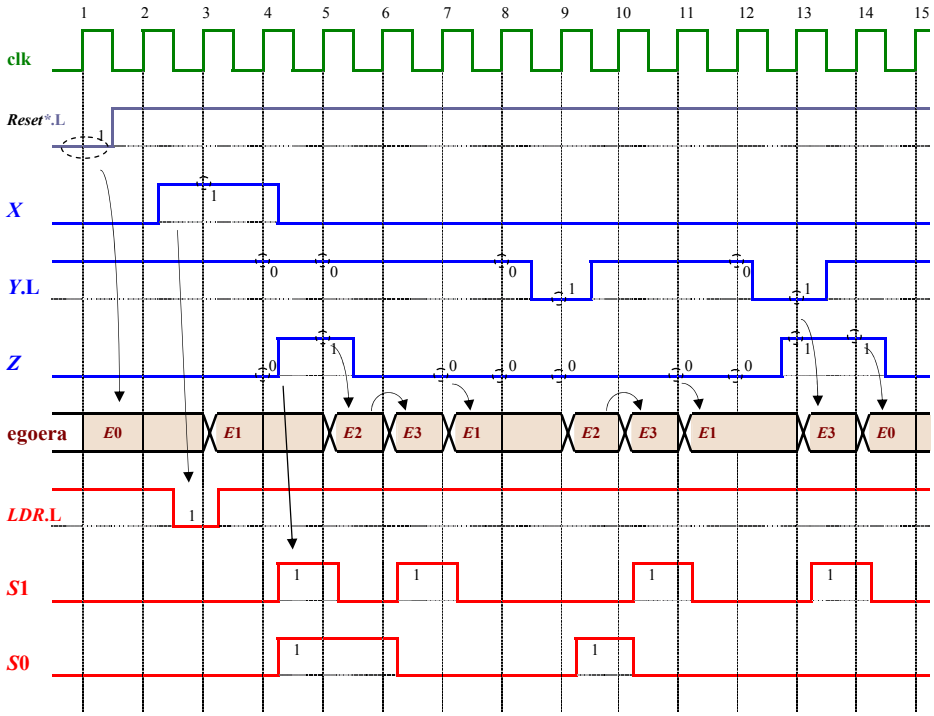
Hasieran, *Reset** seinalea aktibatuta mantentzen den bitartean (L tentsioa), sistemaren egoera *E*0 izango da, kontrol-unitatearen biegonkorren CL*.L sarrera asinkronoa aktibatuta mantentzen delako.

1. zikloa. Erloju-ertza heltzen denean, *Reset** seinalea aktibatuta dago (biegonkorren CL.L sarreran); beraz, *E*0 egoeran geratuko gara ziklo osoan. Tarte horretan $X = 0$ denez, ez da kontrol-seinalerik sortzen.

2. zikloa. *E*0 egoeran izanik, multiplexore bakoitzaren 0 sarrera dago aukeratuta (0 batean, eta *X* bestean); erloju-ertza heltzean, eta *X* seinalea 0 denez, D biegonkorretan kargatuko den balio berria *B'A'* = 00 izango da. Beraz, automata *E*0 egoeran mantenduko da. Egoera horretan kontrol-seinale bat sortu behar da, baina bakarrik $X = 1$ bada. Zikloaren erdialdean edo, *X* seinalea aktibatu egin da, eta, ondorioz, *LDR* kontrol-seinalea aktibatu beharko da ($LDR = E0 \cdot X$).

3. zikloa. Berriz ere, multiplexoreen 0 sarrerak —0 eta *X*— kargatzen dira, erloju-ertzarekin batera, biegonkorretan. Baina orain $X = 1$ denez, hurrengo egoera *B'A'* = 01 izango da. Hau da, *E*1 egoerara igaroko da automata. *LDR* seinalea, beraz, desaktibatuko da. *E*1 egoeran izanda, *S*1 eta *S*0 seinaleak aktiba daitezke, baina soilik $Y = 0$ eta $Z = 1$ direnean (hiru sarrerako

biderketa logikoa kontrol-unitatearen diagraman, 6.8. irudian). Kronograman ageri den moduan, $Y = 0$ eta $Z = 0$ dira ziklo osoan zehar; ondorioz, $S1$ eta $S0$ seinaleak ez dira aktibatuko.



6.9. irudia. 6.8. irudiko kontrol-unitatearen funtzionamenduaren kronograma.

4. zikloa. Multiplexoreen 1 sarrerak aukeratuta daude: Y eta Z sarreren *or* funtzioa batean, eta *equ* funtzioa bestean. Bi seinaleak 0 direnez, *or* funtzioa 0 izango da eta *equ* funtzioa 1. Hurrengo egoera, beraz, honako hau izango da: $B'A' = 0\ 1 = E1$.

Kontrol-seinaleei dagokienez, hau gertatuko da: zikloaren erdian Z seinalea aktibatzen denez, hortik aurrera $S1$ eta $S0$ seinaleak aktibatuko dira (hiru sarrerako *and* logikoa — $E1 \cdot \overline{Y} \cdot Z$ — aktibatu da, eta ondorioz, bi *or* funtzioen irteerak 1 izango dira; ikus 6.8. irudia).

5. zikloa. Multiplexoreen 1 sarrerak aukeratuta daude. Une horretan $Y = 0$ eta $Z = 1$ direnez, multiplexoreen irteeran $B'A' = 10$ izango dugu ($B' = Y + Z$

eta $A' = Y Z + \overline{Y}\,\overline{Z} = Y \otimes Z$). Hurrengo egoera, beraz, $E2$ izango da. Ondorioz, $S1$ seinalea desaktibatuko da; baina ez $S0$, kontrol-seinalea sortzen duen ate logikoaren beste sarrera, $E2$, aktibatu egin delako.

6. zikloa. $E2$ egoeran izanik, 2 sarreretako balioak daude multiplexoreen irteeretan; hots, $B'A' = 11$. Ez dago dudarik: hurrengo egoera $E3$ da. Orain, $S0$ seinalea desaktibatuko da, eta $S1$ aktibatu.

7. zikloa. Multiplexoreen 3 sarreretako balioak aukeratu dira hurrengo egoerarako: 0 eta $\overline{Z}$. Une horretan $Z = 0$ denez, $E1$ (01) egoerara joango gara. $S1$ seinalea desaktibatuko da. $E1$ egoeran izanik, Y-ren eta Z-ren mende daude $S1$ eta $S0$; $Y = Z = 0$ direnez, $S1 = S0 = 0$ izango ditugu.

8. zikloa. Ez dago aldaketarik; $Y = Z = 0$ direnez, hurrengo egoerarako aukeratu diren balioak 01 dira, hau da, $E1$.

9. zikloa. Erloju-ertza iristen denean, $Y = 1$ eta $Z = 0$ dira. Beraz, multiplexoreen irteeretan $B' = 1$ (*or*) eta $A' = 0$ (*equ*) izango dira. Hurrengo egoera $E2$ izango da. Horrekin batera, $S0$ aktibatuko da.

10. eta 11. zikloak. 6. eta 7. zikloen kasu bera dira. Automata $E2$tik $E3$ra eta $E3$tik $E1$era joango da. Lehenengo trantsizioarekin batera, $S0$ desaktibatuko da eta $S1$ aktibatu, eta, bigarrenarekin, $S1$ desaktibatuko da.

12. zikloa. Egoera ez da aldatzen, Y eta Z 0 direlako. $S1$ eta $S0$ desaktibatuta mantenduko dira ziklo osoan zehar.

13. zikloa. 1 sarrerak aukeratu dira multiplexoreetan; une horretan sarrerak $Y = Z = 1$ direnez, $B'A' = 11$ izango dira; hots, hurrengo egoera $E3$ izango da. Eta $S1$ bakarrik aktibatuko da.

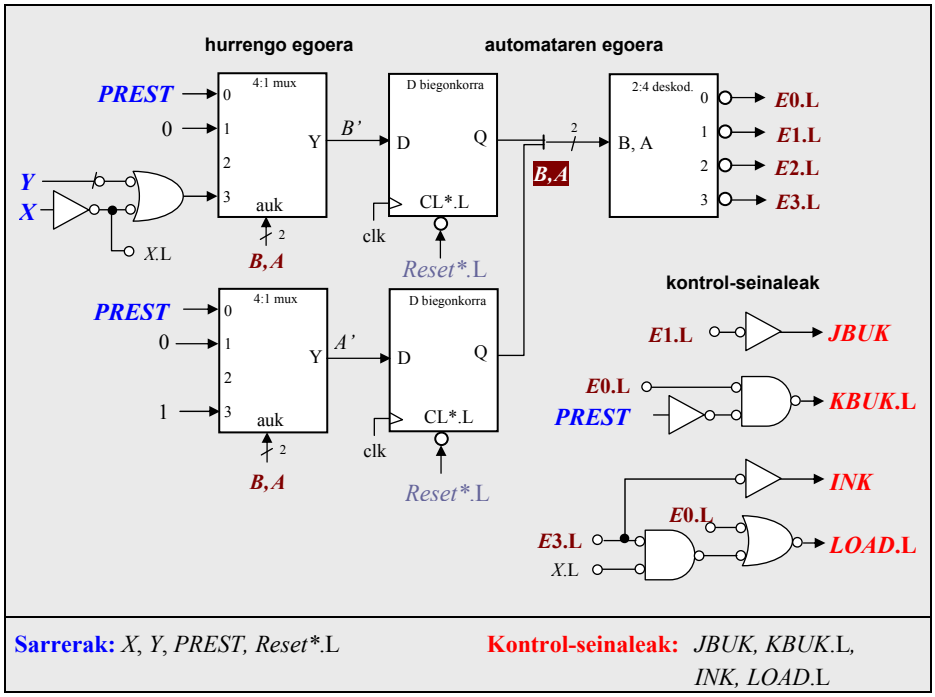
14. zikloa. 3 sarrerak aukeratu dira multiplexoreetan; une horretan $Z = 1$ denez, hurrengo egoera honako hau izango da: $B'A' = 0\overline{Z} = 00 = E0$. Hiru kontrol-seinaleak desaktibatuta daude.

Kontrol-automataren bide guztiak egiaztatu ditugu, eta, guztietan, kontrol-algoritmoarekin bat datorren portaera ikusi dugu; beraz, diseinatu dugun kontrol-unitatea eta jatorrizko kontrol-algoritmoa bat datoz.

«

6.2. Ariketa

Sistema digital baten kontrol-unitatearen eskema logikoa ageri da 6.10. irudian. Zirkuituaren portaera analizatu behar da, eta "exekutatzen" duen kontrol-algoritmoa ondorioztatu.



6.10. irudia. 6.2. ariketako kontrol-unitatea.

■

6.10. irudiko kontrol-unitatean, 2 bit erabiltzen dira egoera adierazteko: *B* eta *A*. Beraz, gehienez, 4 egoera izango ditu kontrol-algoritmoak. Bestalde, lau kontrol-seinale sortzen ditu:

$$JBUK = E1$$
$$KBUK = E0 \cdot PREST$$
$$INK = E3$$
$$LOAD = E0 + E3 \cdot X$$

Kontrol-unitateak gauzatzen duen kontrol-algoritmoa sortzeko, bi atal zehaztu behar ditugu automataren egoera bakoitzeko: (a) zein eta zeren

araberakoa izango den hurrengo egoera, eta (b) sortu behar diren kontrol-seinaleak. Analiza dezagun zirkuituaren portaera zikloz ziklo.

Analisi sinkronoa egin baino lehen, seinale asinkronoak aztertu behar ditugu. Horietako bat dugu zirkuituan, *Reset**, eta seinale horrekin D biegonkorren CL*.L sarrera asinkronoak aktibatzen dira; beraz, edozein delarik sistemaren egoera, *E*0 egoerara joango da automata *Reset** seinalea aktibatzen denean.

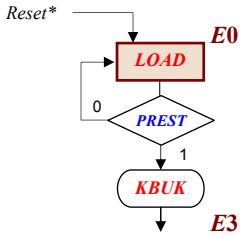
- ***E*0** egoera.

  Egoera horretan —*BA* = 00—, multiplexoreen 0 sarrerak aukeratzen dira. Sarrera horietako informazioak adieraziko du sistemaren hurrengo egoera: *B'* = *PREST* eta *A'* = *PREST* (ikus 6.10. irudia).

  Beraz, *PREST* = 0 bada, hurrengo egoera 00 = *E*0 izango da; eta *PREST* = 1 baldin bada, egoera 11 = *E*3 izango da.

  Bi kontrol-seinale aktibatu behar dira (ikus aurreko ekuazio logikoak). Batetik, *LOAD* seinalea aktibatu behar da *E*0 egoeran, beti; bestetik, *KBUK* seinalea ere aktibatu behar da, baina bakarrik *PREST* seinalea aktibatuta badago.

  Hau da, beraz, egoera horri dagokion diagrama zatia:

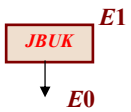


- ***E*1** egoera.

  Biegonkorren irteerek (01) multiplexoreen 1 sarrerak aukeratzen dituzte hurrengo egoera adierazteko; hau da, *B'A'* = 00. Hurrengo egoera, beraz, *E*0 izango da beti.

  Kontrol-seinale bat aktibatu behar da *E*1 egoeran, baldintzarik gabe: *JBUK*.

  Hau da, beraz, *E*1 egoerari dagokion diagrama zatia:



- ***E2*** egoera.

  6.10. irudiko kontrol-unitateak ez du $E2$ egoera erabiltzen. Izan ere, multiplexoreen 2 sarrerak ez dira erabiltzen.

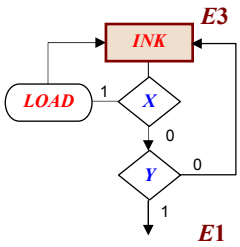
- ***E3*** egoera.

  Sistemaren egoera $E3$ (11) bada, hurrengo egoera multiplexoreen 3 zenbakidun sarreretan dago: $B' = X + \overline{Y}$ eta $A' = 1$.

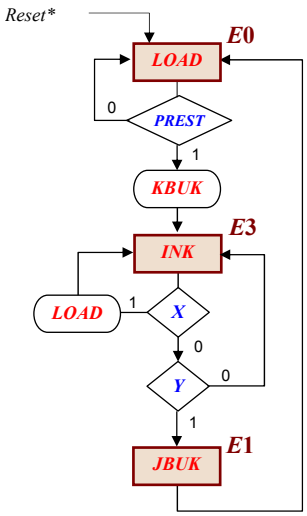
  Beraz, $X = 0,\ Y = 0 \rightarrow$ HE = 11 ($E3$)
  $X = 0,\ Y = 1 \rightarrow$ HE = 01 ($E1$)
  $X = 1 \rightarrow$ HE = 11 ($E3$)

  Bi kontrol-seinale aktibatuko dira $E3$ egoeran (ikus kontrol-seinaleen ekuazio logikoak). Batetik, *INK* (baldintzarik gabe) eta, bestetik, *LOAD*, baldin $X = 1$ bada.

  Hau da $E3$ egoerari dagokion kontrol-algoritmoaren zatia:



Aurreko egoerak batuz lortzen da, beraz, 6.10. irudiko kontrol-unitateak gauzatzen duen kontrol-algoritmo osoa:

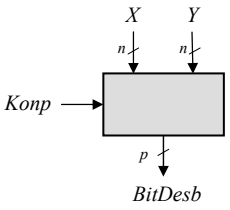


6.11. irudia. Kontrol-algoritmoa (6.10. irudiko kontrol-unitateari dagokiona).

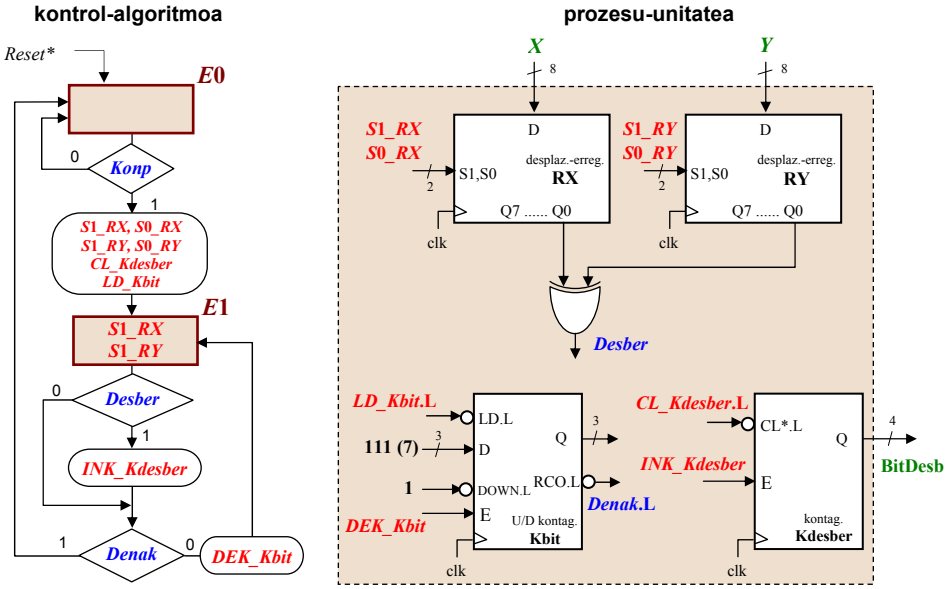
«

6.3. Ariketa

Sistema digital bat diseinatu da n biteko bi zenbaki erkatzeko; hain zuzen ere, kalkulatu behar da zenbat bitetan diren desberdinak bi zenbakiak. Esaterako, sarrerako datuak X = 1101 0110 eta Y = 1000 1111 badira, emaitza 4 izango da.



Sistema digital horren kontrol-algoritmoa eta prozesu-unitatea irudikoak dira (n = 8 bit kasurako).



Datuak bitez bit desplazatzen dira eskuinera, eta pisu txikieneko bitak konparatzen dira, XOR ate baten bidez. Konparatu den bit kopurua kontrolatzeko, Kbit kontagailua erabiltzen da, eta Kdesber kontagailua, bit desberdinen kopurua adierazteko.

ASM algoritmoari dagokion kontrol-unitatea sortu behar da (multiplexore eta biegonkorren bidez) eta, sistemaren funtzionamendua egiaztatzeko, kronograma bat egin (kronogramarako datuak: Reset, Konp, X eta Y).

■

Kontrol-unitatea sortu baino lehen, analiza dezagun sistemaren portaera. *Reset** seinaleaz gain, hiru seinale analizatzen dira kontrol-unitatean: kanpo-seinale bat —*Konp*—, eta prozesu-unitatetik heltzen diren beste bi seinale —*Denak* eta *Desber*—. *Konp* seinalea aktibatzen denean, konparatu behar diren bi zenbakiak kargatu egiten dira desplazamendu-erregistro banatan, eta bi kontagailuak hasieratzen dira: dagoeneko prozesatu den bit kopurua kontatzen duena —Kbit— eta bit desberdinen kopurua kontatzen duena —Kdesber, sistemaren emaitza—. Gero, begizta batean sartzen da automata, zenbakien bitak banan-banan konparatzeko (desplazamendu-erregistroetako Q_0 bita, XOR ate baten bidez).

Konparazio bakoitzarekin, Kbit kontagailua eguneratzen da —*DEK_Kbit,* bit bat gutxiago falta da—, Kdesber kontagailua gehitzen da bitak desberdinak direnean —*INK_Kdesber,* beste bit desberdin bat gehiago—, eta desplazamendu-erregistroen edukia desplazatzen da bit bat eskuinera —*S*1—, hurrengo bita prozesatu ahal izateko. Gogoratu: bi biteko kode bat erabili ohi da desplazamendu-erregistroak kontrolatzeko:

*S*1	*S*0	
0	0	mantendu edukia
0	**1**	desplazatu edukia bit bat ezkerrera
1	0	**desplazatu edukia bit bat eskuinera**
1	**1**	kargatu sarrerako datua

Kontrol-unitatearen gauzatzea

Beti bezala, egoera-taula batean adieraziko dugu kontrol-algoritmoa (zein egoeratatik zein egoeratara). Taula sinplea izango da, bi egoera bakarrik daudelako. Horrez gain, kontrol-seinaleei dagozkien ekuazio-logikoak ere atera behar dira.

Uneko egoera	*A*	**Baldintza**	**Hurrengo egoera**	*A'*
*E*0	0	$\overline{Konp}$	*E*0	0
		Konp	*E*1	**1**
*E*1	**1**	*Denak*	*E*0	0
		$\overline{Denak}$	*E*1	**1**

Bi egoera kodetzeko, nahikoa da bit bat; hau da, nahikoa da D biegonkor bat. Ondorioz, hurrengo egoera sortzeko, nahikoa da 2 sarrerako multiplexore bat.

Hauek izango dira multiplexorearen bi sarrerak:

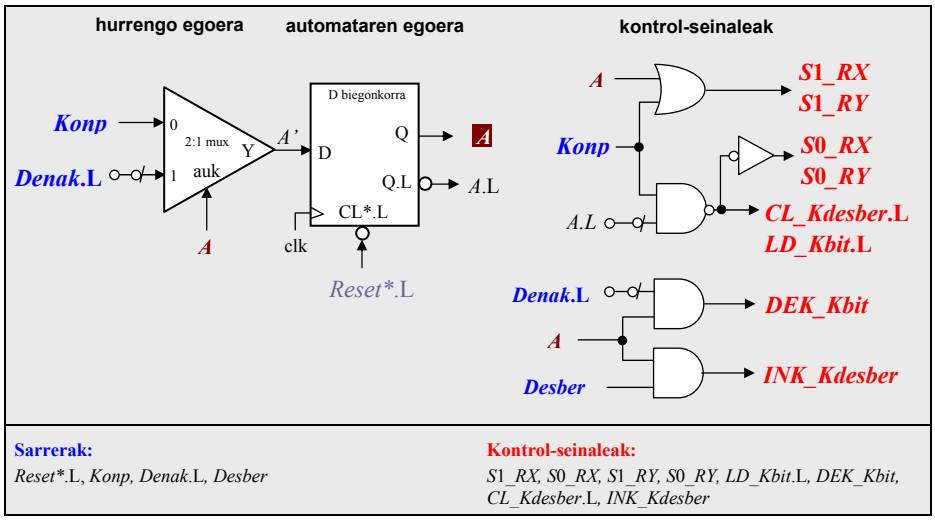
	mux_A
0 sarrera	$Konp$
1 sarrera	$\overline{Denak}$

Bi egoera baino ez dagoenez —bit bakar bat—, ez da beharrezkoa deskodegailu bat bi egoerak adierazteko. Izan ere, $E0 = \overline{A}$ eta $E1 = A$ dira.

Bestalde, hauek dira kontrol-seinaleen ekuazioak:

$$S1\_RX = S1\_RY = E0 \cdot Konp + E1 = \overline{A} \cdot Konp + A = Konp + A$$
$$S0\_RX = S0\_RY = E0 \cdot Konp = \overline{A} \cdot Konp$$
$$LD\_Kbit = CL\_Kdesber = E0 \cdot Konp = \overline{A} \cdot Konp$$
$$DEK\_Kbit = E1 \cdot \overline{Denak} = A \cdot \overline{Denak}$$
$$INK\_Kdesber = E1 \cdot Desber = A \cdot Desber$$

Beraz, honela geratuko da 6.3. ariketako prozesu-unitatea kontrolatuko duen kontrol-unitatea:



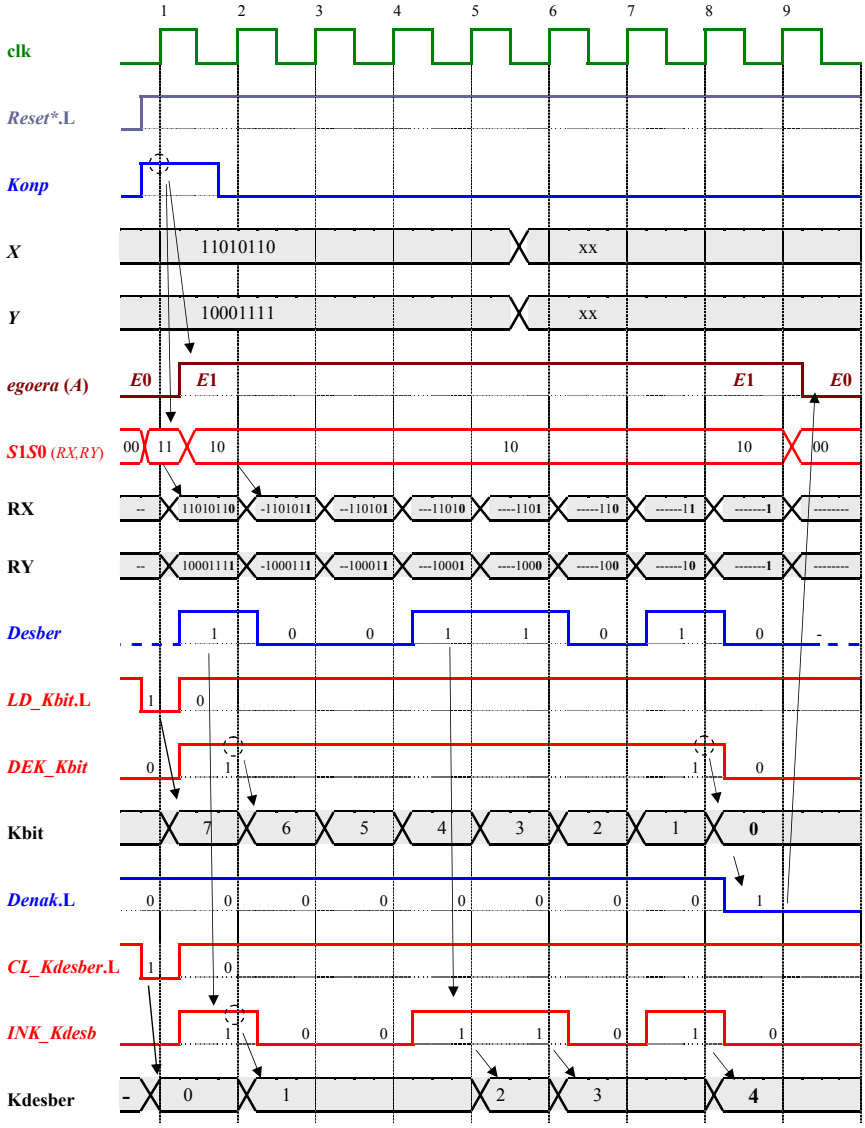
6.12. irudia. 6.3. ariketako kontrol-unitatea.

Kontrol-unitatearen egiaztatzea: funtzionamenduaren kronograma

Sistema osoa diseinatuta, simulazio-prozesu bati ekin behar zaio. Ariketetan egiten ari garen moduan, kronograma bat beteko dugu sistemaren

portaera egiaztatzeko. Kronograma horretan, kanpo-seinaleak definitu behar ditugu, *Reset** eta *Konp*, eta, orobat, konparatu behar diren bi zenbakiak: X = 1101 0110 eta Y = 1000 1111. Adibide bat baino ez da, eta emaitzak 4 izan behar du.

*Reset** seinalea aktibatuta dagoen bitartean, automata *E*0 (A = 0) egoeran egongo da, biegonkorraren CL*.L sarrera aktibatuta dagoelako.



6.13. irudia. 6.3. ariketako sistema digitalaren portaeraren kronograma.

*E*0 egoeran dagoela, *Konp* seinalea aktibatzen da, eta, ondorioz, *S*1 eta *S*0 seinaleak aktibatzen dira, desplazamendu-erregistroak kargatu ahal izateko. Era berean, Kbit eta Kdesber kontagailuak hasieratzeko seinaleak sortzen dira: *LD_Kbit*, 7ko bat kargatzeko (8 bitak prozesatu ahal izateko), eta *CL_Kdesber*, 0ko bat kargatzeko (asinkronoa).

1. erloju-ertza heltzen denean, seinale horiek prozesatzen dira: *X* eta *Y* zenbakiak kargatzen dira RX eta RY desplazamendu-erregistroetan eta 7koa Kbit kontagailuan. Adibide honetan, *X* = 11010110 eta *Y* = 10001111 zenbakiak konparatu nahi ditugu.

*E*1 egoeran gaudela, erregistroen edukia bit bat eskuinera desplazatzeko kontrol-seinaleak sortzen dira *S*1*S*0 = 10. Gainera, zenbakien bit bat —pisu txikienekoa— konparatzen ari da; desberdinak direnez, Kdesber kontagailuaren balioa gehitu behar da. Kbit kontagailuaren edukia ere eguneratuko da (bit bat gutxiago geratzen da konparatzeko). Adi! Sortu diren kontrol-seinaleak —*S*1, *INK_Kdesber* eta *DEK_Kbit*— aktibatu badira ere 1. zikloan zehar, gailu sinkronoetan —desplazamendu-erregistroetan eta kontagailuetan— prozesatu behar dira; beraz, 2. erloju-ertza heltzen denean izango dute eragina.

2. erloju-ertza heltzean, automata *E*1 egoeran mantentzen da (*Denak* = 0).

Hurrengo zikloetan, prozedura bera errepikatzen da, harik eta, 8. zikloan, Kbit kontagailua 0ra heldu arte. Zenbakien 8 bitak prozesatu dira dagoeneko. Kontagailuak berak abisatzen digu 0ra heldu dela: *Denak* = 1 izango da. Automata *E*0 egoerara joango da hurrengo erloju-ertzean.

Hala ere, *E*1 egoeran gauden bitartean, azkeneko zikloan ere, *S*1 kontrol-seinalea aktibatuta dago; beraz, *E*0 egoerara joateaz gain, beste desplazamendu bat egingo da desplazamendu-erregistroetan, beharrezkoa ez den arren. (Ez badugu nahi azken desplazamendu hori egitea, aldatu beharko genuke kontrol-algoritmoa.)

Ohartxo bat. Desplazamendu-erregistroen edukia eskuinera desplazatzen denean, ezkerreko posizioan “hutsune” bat sortzen da. Nahi den informazioarekin bete daiteke hutsune hori. Esaterako, mantendu nahi bada erregistroaren jatorrizko edukia desplazamendu guztiak egin ondoren, nahikoa da SL sarreran konektatzea erregistroaren Q_0 bita; hala, edukiaren errotazio bat osatuko da desplazamenduekin; hori bai, desplazamendu kopurua modu egokian kontrolatu behar da, bit guztiak desplazatzeko. Adibide honetan ez dugu kontuan hartu ezkerreko bitaren edukia, eta “-”

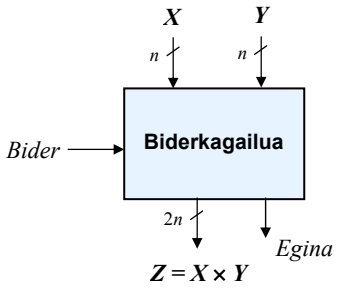
ikurra erabili dugu kronograman. Desplazamendu kopurua, ordea, egokia da (8), eta, beraz, nahikoa litzateke eramatea Q_0 bita SL sarrerara biraketa osatzeko.

«

6.4. Ariketa

Konputagailuetan gehien exekutatzen diren eragiketa aritmetikoak batuketa eta biderketa dira, eta hainbat algoritmo daude eragiketa horiek modurik eraginkorrenean egiteko. Diseinu-ariketak lantzeko asmoz, biderkagailu sinpleena diseinatuko dugu adibide honetan. X × Y egiteko algoritmorik sinpleena oso ezaguna da: gehitu X zenbakia Y aldiz: X + X + ... + X (Y aldiz). Jakina, biderketak egiteko modu hori ez da batere eraginkorra: eragiketaren erantzun-denbora (latentzia) handia da, eta, gainera, ez da konstantea, biderkatzen diren zenbakien araberakoa baita. Hala ere, aukera ematen du zirkuitu digital oso bat diseinatzeko.

Beraz, hau da proposamena: n biteko bi zenbakien biderkagailua diseinatu behar da. Irudian ageri dira zirkuituaren sarrerak eta irteerak. X eta Y biderkatu behar diren bi zenbakiak dira, n bitekoak. Emaitza Z da, 2n bitekoa. Bider *seinaleak biderketari ekiteko agintzen du, eta* Egina *adierazleak biderketa bukatu dela adierazi behar du.*



Diseinu-prozesu guztiak bezala, sistema digitalen diseinua ere proba/errore erako prozesu bat da. Aipatu dugun bezala, sistema osoa bi partetan banatuko dugu: prozesu-unitatea eta kontrol-unitatea. Lehen pausoan, prozesu-unitatearen zirriborroa egin behar da, eta, gero, saiatu prozesu-unitateko zirkuituak kontrolatzen, kontrol-unitate baten bidez. Hainbat aldaketa egin beharko da bidean kontrol- zein prozesu-unitateetan, nahi dugun portaera lortu arte. Azkenik, sistemaren portaera logikoaren eta fisikoaren simulazio zabal eta sakonak egin behar dira, sistema ondo dagoela ziur izan arte. Sistemak ondo funtzionatu beharko du beti, kasu guztietan; gaizki diseinatu den zirkuitua fabrikatu ondoren, ezin daiteke ezer egin akatsak detektatzen badira: lan guztia —eta dirua— alferrik xahutu da.

Prozesu-unitatea

Has gaitezen prozesu-unitatea diseinatzen. $Z = X \times Y$ kalkulatzen duen zirkuitua egin behar dugu. Non daude X eta Y, eta non utzi behar da Z emaitza? Zer tamainatakoak dira eragigaiak eta emaitza?

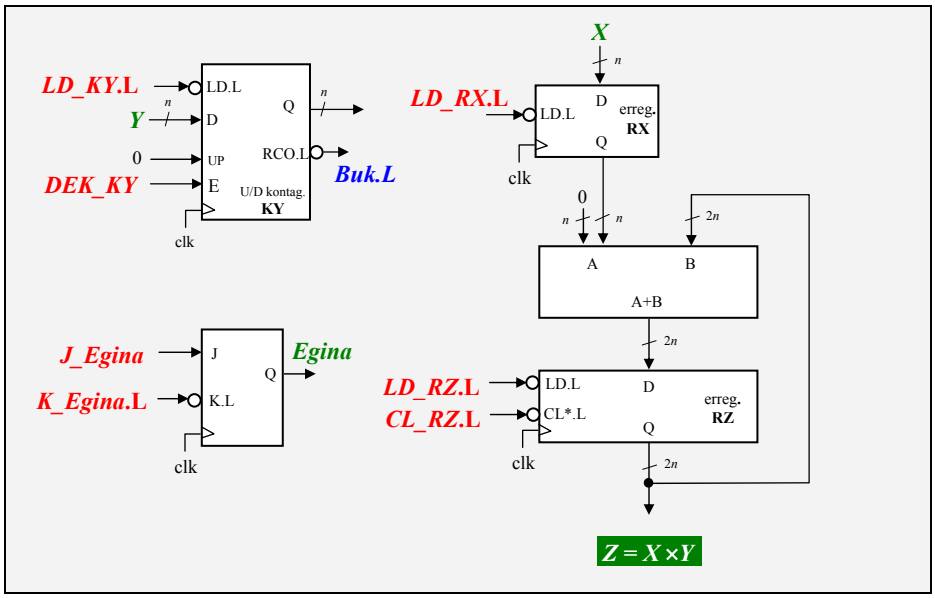
X eta Y sistemaren sarrerak dira; behin eta berriz erabili behar ditugunez gero, erregistro bitan gordeko ditugu (jakingo bagenu ez direla aldatuko eragiketa egiten den bitartean, bi erregistro horiek ez lirateke beharrezkoak izango). Bestalde, n biteko zenbakiak biderkatzen direnean, emaitza $2n$ bitekoa da. Esaterako, 10 oinarrian, digitu bateko bi zenbaki biderkatzen direnean, biderkadurak 2 digitu behar ditu oro har: $6 \times 8 = 48$.

Beraz, ***n* biteko bi erregistro** behar ditugu, X eta Y biderkagaietarako, eta **2*n* biteko erregistro bat,** emaitzarako. Bidenabar, emaitzaren erregistroan utziko ditugu batura partzial guztiak ere ($X + X + ...$). Hori dela eta, hasieran 0ko bat kargatu beharko dugu erregistro horretan. Beraz, erregistro horrek metagailuaren funtzioa beteko du: tarteko eragiketako eragigaia eta azken emaitza izango da.

Aipatu dugun bezala, hainbat batuketa egin behar ditugu biderkadura lortzeko. Horretarako, **batugailu bat** erabili behar dugu, $2n$ bitekoa. Batuketa horiek Y aldiz egin behar dira. Hori kontrolatzeko, **kontagailu bat** erabil daiteke, zeinaren bidez kontatuko baitugu zenbat batuketa egin diren. Hemen ere, aukera zabala da. Esaterako, 0tik Y-ra konta daiteke, edo Y-tik 0ra. Lehenengo kasuan, konparagailu bat erabili behar dugu kontagailuaren balioa eta Y erkatzeko. Bigarrenean, bukaera detektatzeko konparazioa 0rekin egin behar da. Konparazio hori errazagoa da: nahikoa da $\overline{Q_{n-1}} \cdot \overline{Q_{n-2}} \cdot ... \cdot \overline{Q_1} \cdot \overline{Q_0}$ funtzioa egitea; gainera, kontagailuek eman ohi dute informazio hori (kontaketaren "bukaera", 0a beherantz kontatzen denean edo balio maximoa, $2^n - 1$, gorantz kontatzen denean). Beraz, bigarren aukera hartuko dugu. Gainera, kontagailua erregistro bat denez, ez dugu erabiliko beste erregistro bat Yrako: kontagailuan bertan kargatuko dugu eragiketaren hasieran, eta, gero, banan-banan, dekrementatuko dugu.

Azkenik, eragiketa bukatu dela adierazten duen seinalea sortu behar dugu. Izan ere, lehen aipatu dugun moduan, erabiliko dugun biderketa-algoritmoaren exekuzio-denbora ez da konstantea; beraz, ez bada abisatzen eragiketa bukatu dela, kasu txarreneko denbora itxaron beharko da beti, badaezpada ere, emaitza erabili ahal izateko. Bit bateko adierazle-lana egiteko gailurik egokiena JK biegonkorra da, oso modu sinplean aktibatzen (J = 1) eta desaktibatzen (K = 1) delako.

6.14. irudian, biderkagailu sinple honen prozesu-unitatea ageri da. Ohiko osagaiak eta kontrol-seinaleak erabili ditugu. Sinplifikatzeko, sinkronotzat hartuko ditugu kontrol-seinale guztiak, kasu batean izan ezik: biegonkorren *clear* seinaleak (gogoratu: asinkronoak badira, instantean exekutatzen dira; bestela, erloju-ertza heldu arte itxaron behar da).



6.14. irudia. Biderkagailuaren prozesu-unitatea. Batuketa $2n$ bitekoa da, baina eragigai bat, X, n bitekoa da; horregatik, n 0ko gehitu dizkiogu ezkerrean.

Kontrol-algoritmoa

Prozesu-unitatearen lehen diseinua eginda, saia gaitezke kontrol-algoritmo bat garatzen.

Sarrerako n biteko bi datuez gain —X eta Y—, kanpoko bi seinale ditu biderkagailuak:

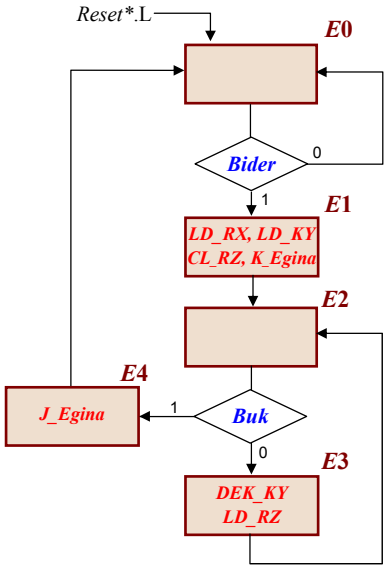
- biderketari ekiteko seinale bat: *Bider*
- sistema hasieratzeko seinale orokorra: *Reset** (asinkronoa)

Eta honako kontrol-seinale hauek sortu behar dira:

- *Egina* JK biegonkorrerako, bi: *J_Egina* eta *K_Egina*.L
- KY kontagailurako, bi: *LD_KY*.L (*Y* biderkagaia kargatzeko) eta *DEK_KY* (edukia 0ra eramateko, banan-banan)
- RX erregistrorako, bat: *LD_RX*.L (*X* biderkagaia kargatzeko)

- RZ erregistrorako, bi: *LD_RZ*.L (batura partzialak kargatzeko) eta *CL_RZ*.L (0z hasieratzeko)

Hauxe izan daiteke kontrol-algoritmoa:



6.15. irudia. Biderkagailuaren lehenengo kontrol-algoritmoa.

Algoritmo hori lehen hurbilpen bat baino ez da. *E*0 egoeran, biderketa egin behar denetz analizatzen da. Hala egin behar denean, kontrol-automata *E*1 egoerara doa, eta memoria duten gailuen edukia hasieratzeko seinaleak sortzen ditu: *X* eta *Y* kargatzeko (erregistroan eta kontagailuan); eta emaitzaren erregistroa eta aurreko biderketa egina dagoela adierazten duen adierazlea ezabatzeko: $Z = 0$ (*CL_RZ*) eta *Egina* = 0 (*K_Egina*).

Hurrengo zikloarekin batera, automata *E*2 egoerara doa. *E*2 – *E*3 – *E*2 – ... begiztan sartuko da automata, batuketa guztiak egiteko. Tartean, *E*3 egoeran, emaitza partziala kargatzeko eta KY kontagailua dekrementatzeko seinaleak sortzen ditu (*LD_RZ* eta *DEK_KY*). *E*2n dagoela, eragiketa bukatu den (batuketa guztiak egin diren) aztertzen da. Hala bada, *E*4 egoerara joango da, non *Egina* adierazlea aktibatzeko kontrol-seinalea sortuko den (*J_Egina*); eta gero, *E*0ra: *Egina* adierazlea piztuko da, eta eragiketa bukatutzat emango da.

Kontrol-algoritmo bat garatuta, merezi du haren portaera analizatzea, “bertsio hobeak” bilatzeko. Esaterako, aurreko algoritmoan, bost egoera erabili ditugu. Beharrezkoak dira bost egoera horiek? Egoera gutxiagorekin (edo gehiagorekin) hobea izango litzateke? Dagoeneko aipatu dugun

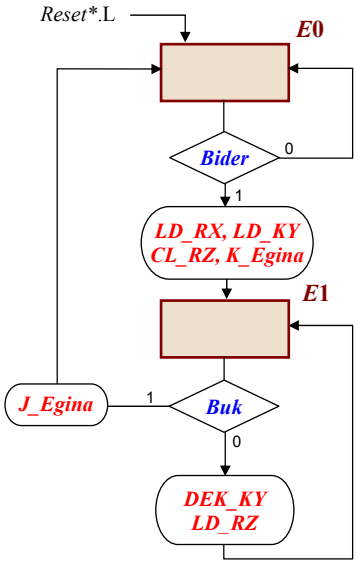
moduan, kontrol-algoritmo baten kalitatea ez da bilatu behar, inolaz ere, egoera kopuruan; bai, ordea, kontrol-seinaleen sekuentziatze zuzenean. Zuzentasuna da, eskuarki, kalitate-parametro nagusia; zuzentasuna ziurtatuta (simulazio-prozesuen bidez), sistemaren beste ezaugarri batzuk hobetu daitezke. Biderkagailuaren kasuan, esaterako, exekuzio-abiadura (biderketak egiteko denbora) parametro garrantzitsua da.

Egin ditzagun kalkulu sinple batzuk. Biderketa lortzeko egin behar den batuketa bakoitzeko, 2 erloju-ziklo behar ditugu (2 egoera, *E*2 eta *E*3, 6.15. irudiko algoritmoan). 32 biteko bi zenbaki biderkatzeko, $2^{32} - 1$ batuketa egin beharko dira kasurik txarrenean; batez bestean, erdia: 2^{31} batuketa. Beraz, 2^{32} erloju-ziklo. Erlojuaren maiztasuna, esaterako, 100 MHz bada (periodoa 10 ns), biderketa egiteko $2^{32} \times 10$ ns = 43 s beharko ditugu!

Erantzun-denbora (onartezin) hori erdira jaitsiko da batuketa bakoitzean, 2 ziklo erabili beharrean, ziklo bakar bat erabiltzen badugu; hots, *E*3 egoera *E*2 egoeraren baldintzapeko irteera bihurtzen badugu[21].

Exekuzio-denboraren gainean ia eraginik ez badu ere, antzekoa egin daiteke *E*1 eta *E*4 egoerekin.

Hona hemen kontrol-algoritmoaren bertsio laburtua:



6.16. irudia. Biderkagailuaren azken kontrol-algoritmoa.

[21] Hala ere, lehen aipatu dugun bezala, algoritmo askoz eraginkorragoak erabiltzen dira biderketak egiteko. Izan ere, adibide honetakoa ohiko diseinu-prozesua azaltzeko baino ez dugu erabili nahi.

Kontrol-unitatea

Multiplexoreen metodoa erabiliko dugu kontrol-unitatea sortzeko. Hona hemen egoeren arteko trantsizioen taula:

Uneko egoera A		Baldintza	Hurrengo egoera A'	
$E0$	0	$\overline{Bider}$	$E0$	0
		$Bider$	$E1$	1
$E1$	1	Buk	$E0$	0
		$\overline{Buk}$	$E1$	1

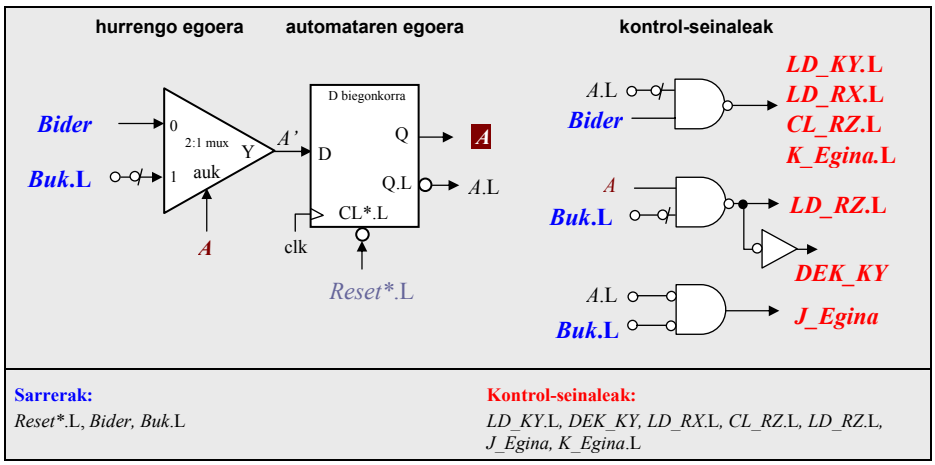
D biegonkor bat eta bi sarrerako multiplexore bat behar ditugu, eta hauek izango dira multiplexorearen bi sarrerak:

	A_mux
0 sarrera	$Bider$
1 sarrera	$\overline{Buk}$

Azkenik, sortu behar diren **kontrol-seinaleen ekuazioak:**

$$CL\_RZ = K\_Egina = LD\_RX = LD\_KY = E0 \cdot Bider = \overline{A} \cdot Bider$$
$$LD\_RZ = DEK\_KY = E1 \cdot \overline{Buk} = A \cdot \overline{Buk}$$
$$J\_Egina = E1 \cdot Buk = A \cdot Buk$$

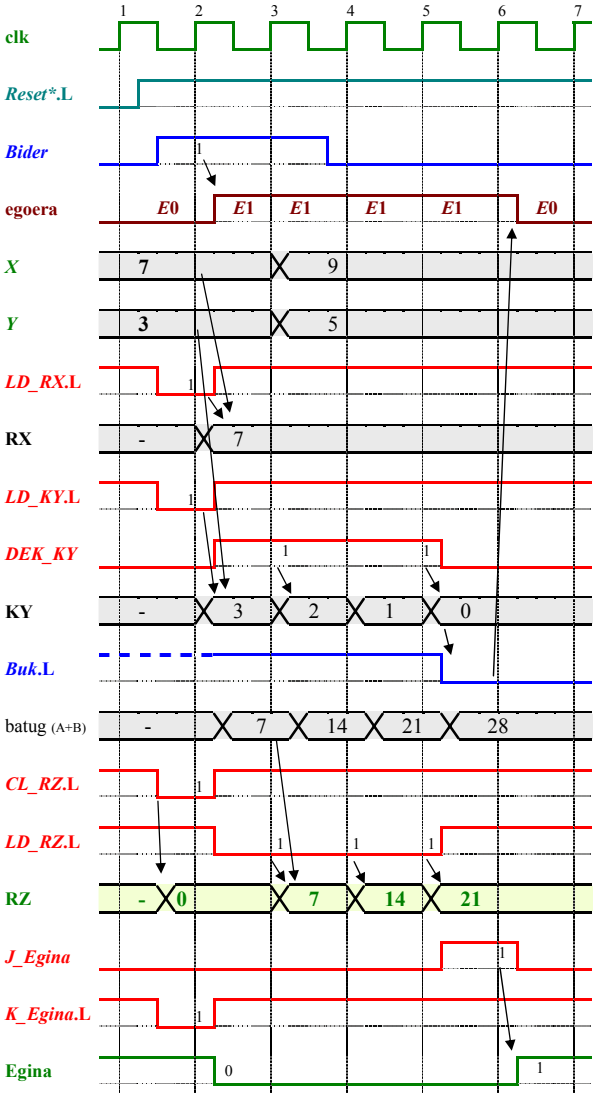
Beraz, honela geratuko da kontrol-unitatea:



6.17. irudia. Biderkagailuaren kontrol-unitatea.

Sistemaren egiaztatzea: kronogramak

Aipatu bezala, sistema logiko guztien portaera analizatu behar da konputagailu bidezko simulazioak eginez. Sistema erraza bada, eskuz ere egin daiteke simulazio hori, kronograma baten bidez. Esaterako, hau da diseinatu dugun biderkagailuaren portaeraren kronograma bat. Kasu sinple bat ageri da: $(X = 7) \times (Y = 3) = 7 + 7 + 7 = 21$.



6.18. irudia. Biderkagailuaren kronograma.

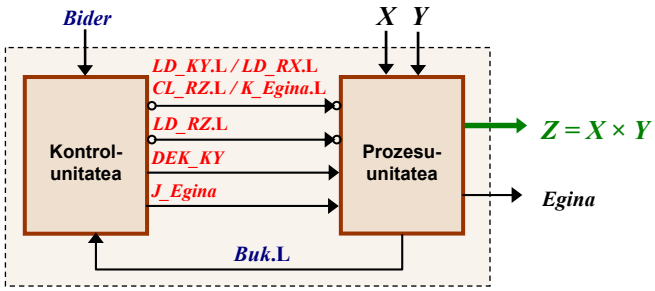
1. zikloan, automata *E*0 egoeran dago, eta *Bider* seinalea aktibatzen da; beraz, baldintzapeko kontrol-seinale hauek aktibatzen dira: *LD_RX* = *LD_KY* = *CL_RZ* = *K_Egina*. *CL_RZ* seinalea unean bertan exekutatzen da (RZ = 0), sarrera asinkrono batera doalako (CL*). Beste hirurak, ordea, hurrengo erloju-ertza heltzean exekutatuko dira. 2. zikloaren hasieran, beraz, *E*1 egoerara doa automata, eta hasieratzen dira RX erregistroa (7), KY kontagailua (3) eta *Egina* JK biegonkorra (0).

KY = 3 denez, 3 ziklo beharko ditugu biderketa egiteko. Ziklo bakoitzean *LD_RZ* seinalea aktibatuta dago, eta, beraz, erloju-ertz bakoitzarekin, emaitza partzial bat kargatuko da RZ erregistroan (7, 14 eta 21). 5. zikloan, kontagailuaren balioa 0 da, zikloz ziklo dekrementatu delako, eta, ondorioz, *Buk* seinalea aktibatzen da. Ziklo horretan, beraz, biderketa bukatutzat emateko prestatzen da automata: ez da berriz kargatuko RZ erregistroa, eta *J_Egina* kontrol-seinalea aktibatzen da *Egina* adierazlea pizteko.

6. erloju-ertza heltzen denean, automata *E*0 egoerara itzultzen da eta *Egina* adierazlea pizten da. Eragiketa bukatu da. Eragiketa betetzeko, 1 + 3 + 1 = 5 ziklo erabili dira (1 + *Y* + 1 ziklo).

Merezi du kronogramako beste puntu bat aztertzea. RZ erregistroaren edukiaz gain, batugailuaren irteera ere irudikatu dugu kronograman (6.18. irudia). Adi! Batugailua zirkuitu konbinazional bat da, hau da, ez du erabiltzen erloju-seinalea: sarrerak aldatu ahala, irteera aldatzen da (dagokion latentziarekin). Beraz, RZ aldatu bezain laster, batugailuaren emaitza ere aldatzen da. Horregatik, RZ = 21 denean (biderketaren emaitza), batugailuaren irteera ere aldatzen da: 21 + 7 = 28. Ondo, ez dago arazorik. Izan ere, kontrol-unitateak ez du kontuan hartuko emaitza partzial hori, dagoeneko erabaki baitu biderketa bukatu dela.

6.19. irudian, sistema osoaren eskema bat ageri da, unitateen arteko informazio-trukea azpimarratzeko asmoz.

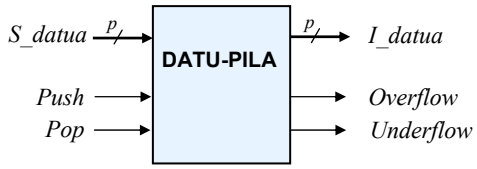


6.19. irudia. Biderkagailuaren eskema logiko orokorra.

«

6.5. Ariketa

Ariketa honetan, konputazio-sistema baten datu-pila diseinatu nahi dugu. RAM memoria bat erabili behar da pila egiteko, eta bi eragiketa onartu behar ditu: Push (datu bat pilan sartu) eta Pop (datu bat pilatik atera). Gainezkatzeak (goitik —overflow— zein behetik —underflow—) detektatu eta adierazi behar dira.



Pilaren definizioa. Prozesu-unitatea.

Datuak prozesatzea da konputagailu guztien eginkizun nagusia. Programetan prozesatzen diren datuak eta lortzen diren emaitzak memorian gordetzen dira. Konputagailu baten memoria-ahalmena modu hierarkikoan antolatu ohi da. Hierarkia horretan, hainbat maila definitzen dira, eskuarki erabilera-abiaduraren eta edukieraren arabera. Hala, beheko mailan erregistroak ditugu —azkarrenak, baina edukiera txikikoak (byte batzuk)—; ondoren, cache memoria, memoria nagusia, diskoak, eta abar.

Batzuetan, atzipen modua erabiltzen da memoriak bereizteko: edukiaren bidezko atzipena (cachea), helbidearen bidezkoa (RAM memoria), sekuentziala (zintak)... eta pila.

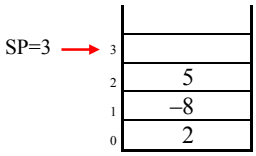
Memoria-egitura erabilienetako bat da pila. Zenbait makinatan, memoria-egitura nagusia da pila (adibidez, HP etxeko kalkulagailuetan), baina konputagailu guztietan erabiltzen da memoria nagusiaren zati bat pila gisa. Datuak gordetzeko pila bat erabiltzen denean, datuak metatu egiten dira pilan, bata bestaren gainean. Erakusle batek, SP (*stack pointer*), adierazten du, une oro, pilaren gaineko posizioa (memoria-helbidea). Datuak gordetzeko, `PUSH` agindua erabiltzen da, eta datuak eskuratzeko, `POP` agindua.

Hainbat egitura erabil daitezke pila bat osatzeko; esaterako, RAM memoria bat datuak gordetzeko eta kontagailu bat memoria helbideratzeko. Kontagailu horrek “pilaren erakuslearena” (SP) egiten du. Eragiketa bakoitzean, datuak idatzi (*push*) edo irakurtzen (*pop*) dira memorian eta

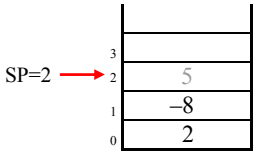
erakuslea eguneratzen da. Erabaki bat hartu behar dugu hemen; zer erakusten du erakusleak? okupatuta dagoen azken posizioa edo libre dagoen lehena? Bi aukerak zilegiak dira; adibide gisa, bigarrena erabiliko dugu; hots, libre dagoen lehenengo posizioa adierazten du kontagailuak. Hala egiten bada, honela interpretatu behar dira *Push* eta *Pop* eragiketak:

- *Push* datua
```
MEM[SP] := datua   ; idatzi datua pilan, libre dagoen lehen posizioan
SP := SP + 1       ; eguneratu erakuslea, libre dagoen hurrengo posizioa erakusteko
```
- *Pop*
```
SP := SP - 1       ; eguneratu erakuslea, azken datua eskuratu ahal izateko
datua := MEM[SP]   ; irakurri memoria
```

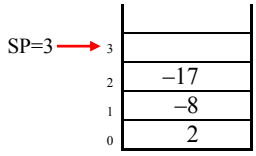
Azter dezagun adibide bat. Hau da, esaterako, pilaren egoera: hiru datu kargatu dira: 2, –8 eta 5. Erakusleak 3 helbideko posizioa erakusten du (libre dagoen lehenengo posizioa).



(a) *Pop*. Erakuslearen balioa dekrementatzen da, eta memoria (pila) irakurtzen da. Emaitza 5 da. Erakusleak markatzen du dagoeneko libre dagoen posizioa (2 helbidekoa, hurrengo idazketan erabiliko dena). Adi! pilan dagoen informazioa ez da ezabatu, baina ez balego bezala tratatuko da.



(b) *Push* –17. Datu bat kargatu (idatzi) behar da pilan. Erakusleak adierazten duen posizioan idazten da, eta erakuslea eguneratzen da.



Bestalde, zer egin behar da *Pop* eskaera prozesatu behar bada pila hutsik dagoenean? eta *Push* pila beteta dagoenean? Berriz ere, aukera asko dago erantzun egoki bat emateko. Esaterako, bi adierazle (bi JK biegonkor) erabil daitezke "pila hutsa" eta "pila beteta" adierazteko; edo SP erakuslea bera erabil daiteke funtzio hori egiteko.

Adibidez, demagun 16 posizioko memoria bat dugula pila bat egiteko. 16 helbideak emateko, nahikoa litzateke 4 biteko kontagailua; baina, hala egiten badugu, SP = 0 balioak esanahi bikoitza izango luke: batetik, pila hutsik dagoela (hasieran, esaterako), baina, era berean, pila beteta dagoela; izan ere, 15 posizioan datu bat kargatzen denean, erakuslea gehitzen da, eta, 4 bitetan, 15 + 1 = 0 da. Aukera bat baino gehiago dugu arazo hori konpontzeko. Esaterako,

(a) pilaren azken posizioa ez erabiltzea; horrela, SP = 15 balioak adieraziko du "pila beteta";

(b) biegonkor bat erabiltzea pila beteta dagoela adierazteko (eta uztea SP = 0 pila hutsa adierazteko);

(c) 5 (oro har, n+1) biteko kontagailu bat erabiltzea SP gisa. 5 bit erabiliz, 15 + 1 = 16 (10000 bitarrez) izango da eta, beraz, SP kontagailuaren pisu handieneko bitak adieraziko du pila beteta dagoela. Hori da erabiliko dugun aukera adibide honetan (ikus 6.20. irudia).

Aurreko guztia kontuan harturik, osagai hauek erabil daitezke datu-pila bat osatzeko:

- 2^n posizioko **RAM memoria bat:** pila

  Kontrol-seinaleak: *OE*, *WR* eta *CS* (hirurak logika negatiboan). Kontrol-seinale horien bidez, bi eragiketa nagusiak egin daitezke:

  irakurketa: *OE* eta *CS*
  idazketa: *WR* eta *CS*

  CS 0 denean, memoria desaktibatuta dago.

- n+1 biteko **kontagailu bat:** SP erakuslea

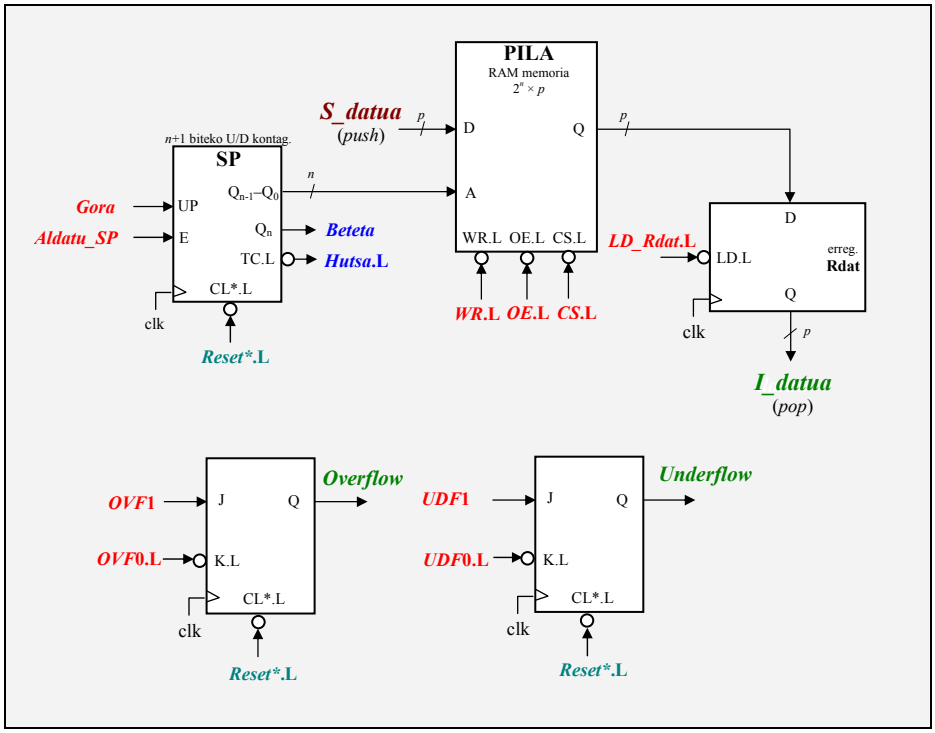
  Kontrol-seinaleak: *Gora* (UP) eta *Aldatu_SP* (E).

Aldatu_SP	*Gora*	
0	–	(ezer ez)
1	1	SP := (SP + 1) mod n
1	0	SP := (SP – 1) mod n

  Kontagailuaren TC irteerak pila hutsik dagoela adieraziko du *Gora* = 0 denean (SP = 0); Q_n bita, aldiz, pila beteta dagoela.

- **2 JK biegonkor,** *Overflow* eta *Underflow* egoerak adierazteko.
  Kontrol-seinaleak: *OVF*1 (J), *OVF*0 (K.L) eta *UDF*1 (J), *UDF*0 (K.L)
- *p* biteko **erregistro bat,** pilatik irakurri den datua gordetzeko (aukera bat da).
  Kontrol-seinalea: *LD_Rdat* (LD.L)

6.20. irudian ageri da pilaren prozesu-unitatearen eskema logikoa.



6.20. irudia. Pila: prozesu-unitatea.

Kontrol-algoritmoaren garapena

6.20. irudiko prozesu-unitatearen funtzionamendua kontrolatu behar dugu; beraz, kontrol-unitatea sortu behar dugu, ASM algoritmo batean oinarrituta. Algoritmo horrek, ohiko hasieratze-seinale asinkronoaz gain (*Reset**), bi sarrera prozesatu behar ditu: *Push* eta *Pop* eragiketa-eskaerak.

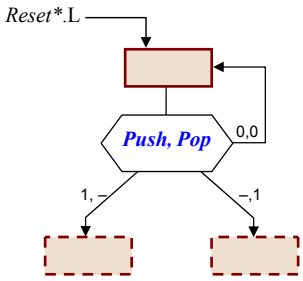
Push eragiketa-eskaera detektatzen denean, hau egin behar da: (a) egiaztatu tokia dagoela pilan; tokirik ez badago, *Overflow* adierazlea aktibatu behar da eta eragiketa bertan behera utzi; tokia badago (b) memorian idatzi behar da datua; eta (c) SP erakuslea eguneratu behar da.

Era berean, *Pop* eragiketa-eskaera detektatzen denean: (a) pila hutsik badago, *Underflow* adierazlea aktibatu behar da eta eragiketa bertan behera utzi; bestela (b) SP erakuslea dekrementatu egin behar da; eta (c) memoria irakurri behar da (eta erregistroa kargatu).

Egin dezagun, bada, pila kontrolatzen duen kontrol-algoritmoaren lehen bertsioa.

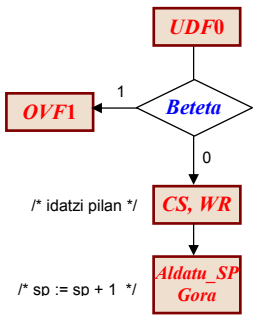
- Hasteko, lehenengo egoeran, eragiketa bat egin behar den ala ez analizatu behar dugu. Era berean, egoera hori erabil daiteke helburutzat hasieratze-seinale asinkrono bat aktibatzen denean (*Reset**).

  Suposatuko dugu ez direla inoiz aktibatuko batera *Push* eta *Pop* seinaleak (hori dela eta, 1,– edo –,1 erabili ditugu algoritmoan).



- Demagun *Push* eragiketa exekutatu behar dela (pilan idatzi). Lehenik, aztertu behar da ea tokia dagoen pilan (*beteta* seinalea). Pila beteta badago, *Overflow* seinalea aktibatu behar da (*OVF*1*)*; bestela, idazketara pasa gaitezke. Idazteko, memoriaren *CS* eta *WR* seinaleak aktibatu behar dira. Gero, pilaren erakuslearen (SP kontagailuaren) edukia gehitu behar da, pilako hurrengo posizio librea adierazteko.

  Bidenabar, *Push* eragiketa bat egiten ari denez —hots, datu bat pilan sartu nahi da, eta ez da hutsik izango— ezabatu egin daiteke balizko *Underflow* adierazlea: *UDF*0 (dagoeneko desaktibatuta bazegoen, ez du eraginik).

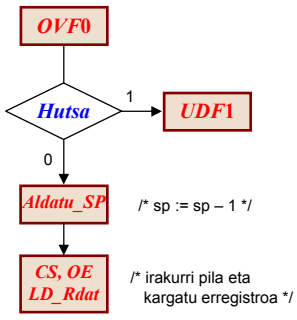


- Demagun orain *Pop* eragiketa exekutatu behar dela. Antzeko egitura izango dugu kontrol-algoritmoan, baina orain, irakurri baino lehen, SP

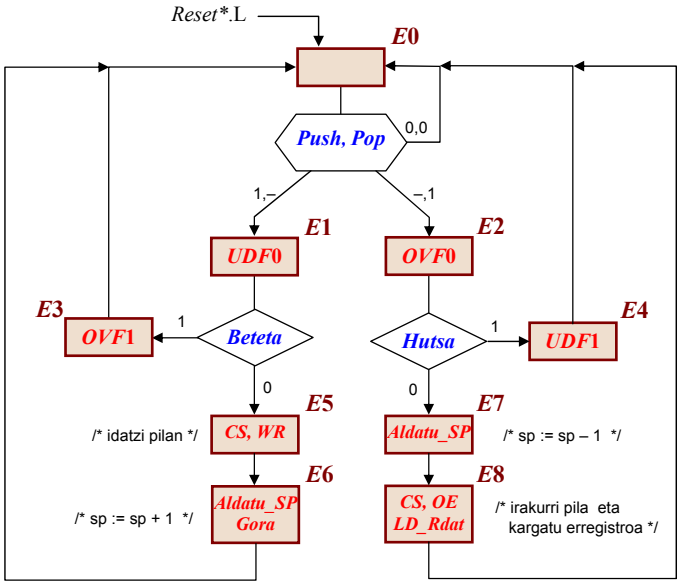
kontagailua eguneratu behar da, SP := SP – 1 eginez. Pila (memoria) irakurtzeko, bi kontrol-seinale aktibatu behar dira: *OE* eta *CS*. Bidenabar, pilatik irakurri den datua erregistroan kargatzeko seinalea sortzen da (*LD_Rdat*).

Irakurketa bat egin nahi bada hutsik dagoen pilan, orduan *Underflow* egoera adierazten duen adierazlea aktibatu beharko da (*UDF*1).

Aurreko kasuan bezala, balizko *Overflow* seinalea ezabatzen da (*OVF*0), datu bat aterako baita pilatik.



Honela geratuko litzateke algoritmo osoa (lehen bertsioa):



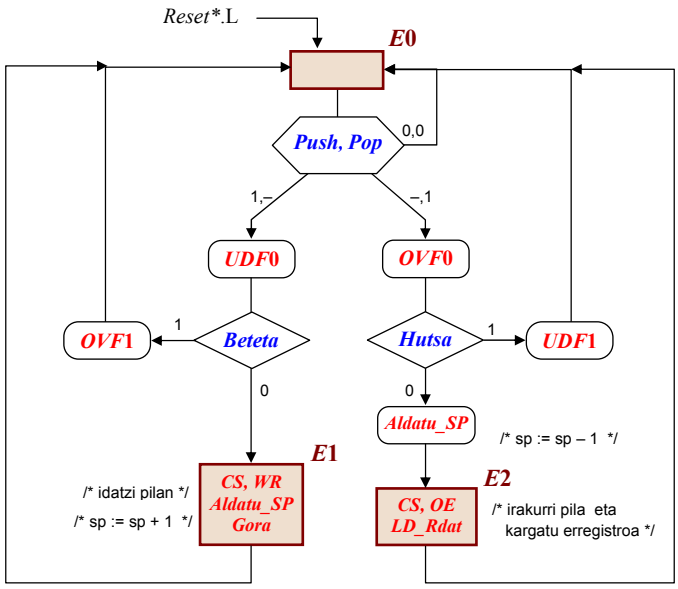
6.21. irudia. Pila: lehenengo kontrol-algoritmoa.

Bederatzi egoera ditu garatu dugun kontrol-algoritmoak. Beraz, 4 biegonkor beharko genituzke kontrol-unitatea sortzeko. Nahi izanez gero, modu sinplean labur daiteke egoera kopurua.

Adibidez, $E1$, $E2$, $E3$ eta $E4$ egoerak $E0$ egoeraren baldintzapeko irteera gisa eman daitezke, arazorik gabe. Gauza bera egin daiteke $E7$ egoerarekin. Azkenik, egoera bakar batean bil daitezke $E5$ eta $E6$ egoerak.

Adi! $E7$ eta $E8$ egoerak ezin dira bildu egoera bakar batean, bi ziklo behar direlako: bat erakuslea eguneratzeko, eta beste bat memoria irakurtzeko.

Honela geratuko da, finean, pilaren kontrol-algoritmoa:



6.22. irudia. Pila: azken kontrol-algoritmoa.

Hiru egoera ditu kontrol-automatak (2 bit, beraz, egoerak kodetzeko). Hala ere, ohartxo bat egin behar da. Sistema digital honetan, gailu gehiagorekin batera, RAM memoria bat erabiltzen da. Memoriak dira, ohiko gailu digitalen artean, motelenak. Esaterako, memoriaren erantzun-denbora 100 ns izan daiteke, gainerako gailuen erantzun-denbora baino magnitude-ordena bat edo bi gehiago. Sortu dugun kontrol-algoritmoan, memoriako irakurketak zein idazketak egoera bakar batean bete behar dira; hots, erloju-ziklo batean. Horrek muga bat ezartzen dio erloju-periodoari: memoriaren latentzia baino handiagoa izan behar du. Muga horren eragina sistema osoan nabarituko da, eta gerta liteke sistema gehiegi moteltzea. Hala bada kasua, soluzioa erraza da oso: luzatu memoria-eragiketa egoera (ziklo) gehiagotan. Adibide honetan, onartuko dugu memoriaren eragiketak ziklo batean egin daitezkeela.

Kontrol-unitatea

6.22. irudiko algoritmoa betetzen duen kontrol-unitatea sortuko dugu orain, aurreko ariketetan egin dugun modu bertsuan. Lehenik, egoera-taula eta kontrol-seinaleen ekuazioak idatziko ditugu. Eta hortik abiatuta, kontrol-unitatea marraztuko dugu.

Uneko egoera	*B A*	**Baldintza**	**Hurrengo egoera**	*B' A'*
*E*0	0 0	$\overline{Push} \cdot \overline{Pop} + Push \cdot Beteta + Pop \cdot Hutsa$	*E*0	0 0
		$Push \cdot \overline{Beteta}$	*E*1	0 1
		$Pop \cdot \overline{Hutsa}$	*E*2	1 0
*E*1	0 1	-	*E*0	0 0
*E*2	1 0	-	*E*0	0 0

Bi D biegonkor eta 4 sarrerako bi multiplexore erabiliko ditugu kontrol-unitatea sortzeko[22]. Hona hemen multiplexore horien sarrerak, aurreko taulatik eratorrita:

	mux_B	**mux_A**
0 sarrera	$Pop \cdot \overline{Hutsa}$	$Push \cdot \overline{Beteta}$
1 sarrera	0	0
2 sarrera	0	0

9 kontrol-seinale sortu behar dira. Hauek dira haien ekuazio logikoak:

- RAM memoria:

  $CS = E1 + E2$
  $OE = E2$
  $WR = E1$

- SP kontagailua

  $Aldatu\_SP = E1 + E0 \cdot Pop \cdot \overline{Hutsa}$
  $Gora = E1$

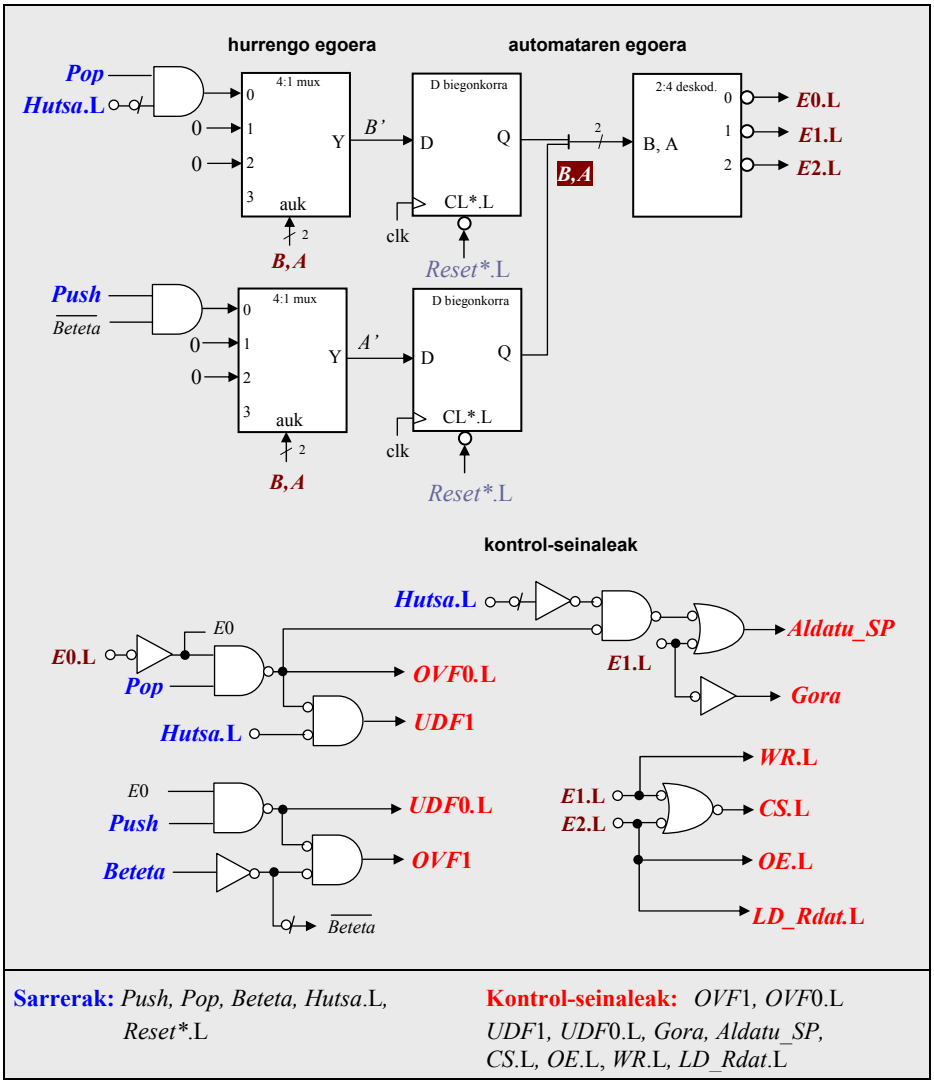
- Rdat erregistroa:

  $LD\_Rdat = E2$

- *Overflow* / *Underflow* biegonkorrak:

  $OVF1 = E0 \cdot Push \cdot Beteta$
  $OVF0 = E0 \cdot Pop$
  $UDF1 = E0 \cdot Pop \cdot Hutsa$
  $UDF0 = E0 \cdot Push$

[22] Adibide honetako *B'A'* funtzioak oso sinpleak dira, 1eko bakar bat dutelako. Beraz, nahi izanez gero, multiplexoreak erabili beharrean, dagokien funtzio logikoa sor daiteke, ateen bidez: $B' = E0 \cdot Pop \cdot Hutsa$ eta $A' = E0 \cdot Push \cdot Beteta$.



Sarrerak: *Push, Pop, Beteta, Hutsa*.L, *Reset**.L

Kontrol-seinaleak: *OVF*1, *OVF*0.L *UDF*1, *UDF*0.L, *Gora, Aldatu_SP, CS*.L, *OE*.L, *WR*.L, *LD_Rdat*.L

6.23. irudia. Pila: kontrol-unitatea.

Horrela egiten direnean, kontrol-unitate guztien egitura beti bera da: hainbat biegonkor automataren egoera gordetzeko, multiplexore batzuk hurrengo egoera sortzeko, eta hainbat ate kontrol-seinaleak sortzeko.

Pilaren diseinua bukatuta dago. Azkeneko fasean sartu beharko dugu: portaeraren simulazioa konputagailuan. Horren ordez, sistemaren portaera islatzen duen kronograma bat egingo dugu.

Pilaren funtzionamendua egiaztatzen duten kronogramak

Kasu posible guztien simulazioa kronograma bakar batean biltzea zaila da; hori egin beharrean, kasu partikularrak analizatuko ditugu, banan-banan. Adibide hauetarako, pilaren tamaina 16 hitz izango da (n = 4 biteko helbideak). Hasiera-egoera, kasu guztietan, $E0$ da.

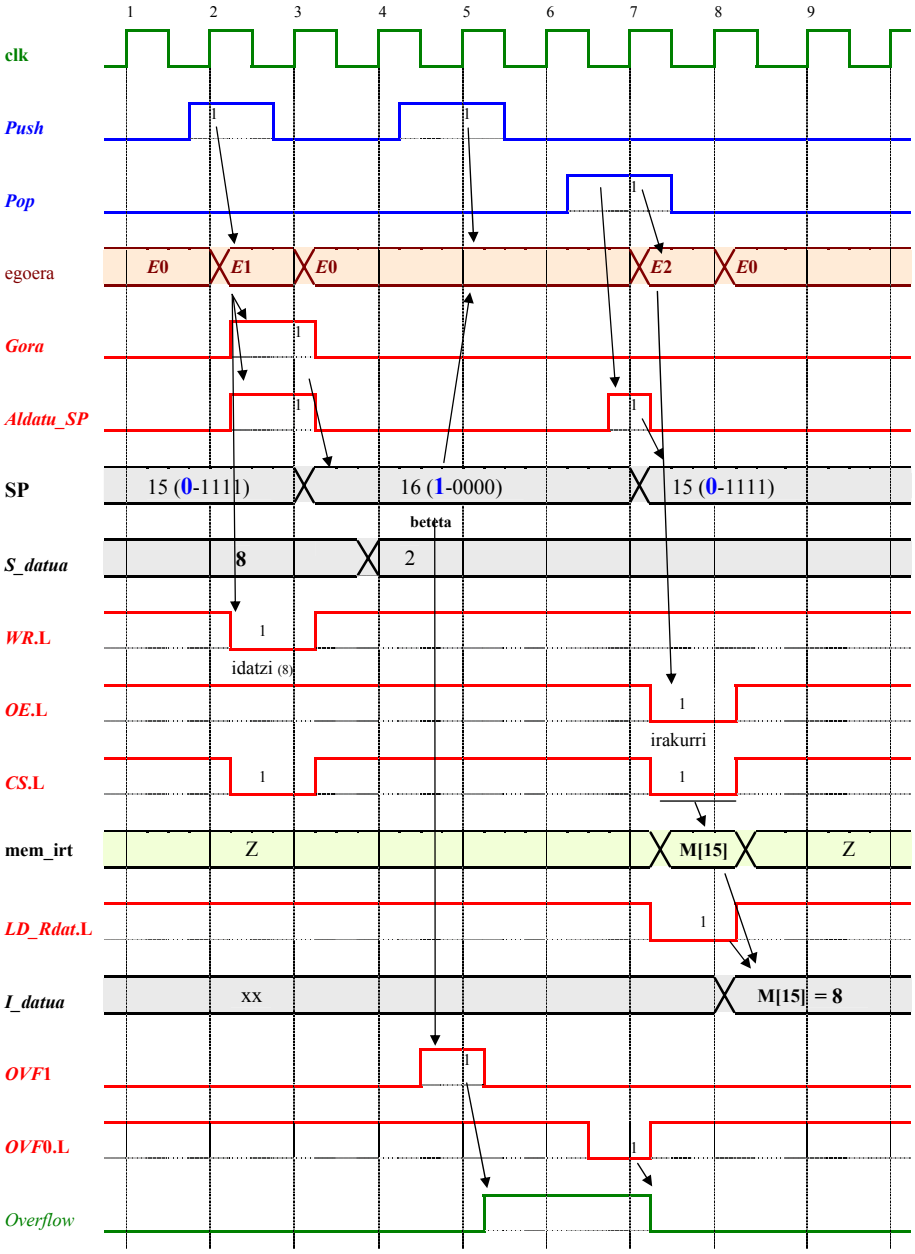
1. kasua

Pila betetzear dago, 15 datu daude pilan (SP = 15 = 01111), eta honako eragiketa hauek ikusiko ditugu: *Push* bat (datu bat pilan sartu: pila betetzen da, SP = 16 = 10000); beste *Push* bat (*Overflow*) eta, azkenik, *Pop* bat (ikus 6.24. irudia).

Hasieran, automata $E0$ egoeran dago. Geroxeago, *Push* seinalea aktibatzen da; 2. erloju-ertzarekin batera, automata $E1$ egoerara igarotzen da, *Beteta* = 0 delako (SP kontagailuaren pisu handieneko bitak adierazten du pila beteta dagoen ala ez).

$E1$ egoeran, memorian (pilan) idazten da. Horretarako, *CS* eta *WR* seinaleak aktibatu dira. Gainera, SP kontagailuaren balioa gehitzeko kontrol-seinaleak sortzen dira $E1$ egoeran (*Aldatu_SP* eta *Gora*). Seinale horien eragina erloju-ertza heltzean nabarituko da: SP = 16. Hala, pila bete egin da (SP kontagailuaren pisu handieneko bita 1 da). Horregatik, 4. zikloan zehar berriro ere *Push* seinalea aktibatzen denean, pila beteta dagoela detektatzen du automatak, eta *Overflow* adierazlea aktibatzeko kontrol-seinalea sortzen du ($OVF1$); 5. erloju-ertza heltzean, automata $E0$ egoeran mantenduko da, eta *Overflow* adierazlea piztuko da.

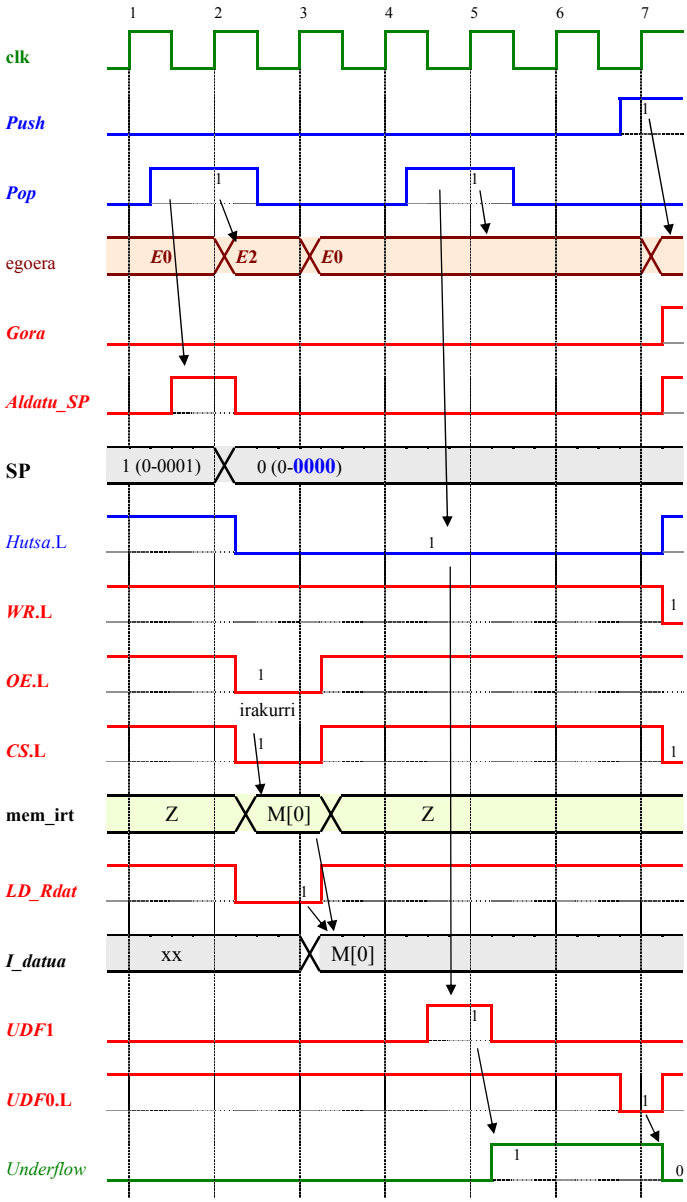
Overflow seinalea piztuta mantenduko da harik eta *Pop* eragiketa eskatzen den arte (pilatik datu bat eskuratu). Hala, 6. zikloaren erdialdean edo, *Pop* seinalea detektatzen denean, balizko *Overflow* adierazlea ezabatzeko kontrol-seinalea sortzen da ($OVF0$; dagoeneko 0an bazegoen, seinale horrek ez du eraginik izango), pila hutsik dagoen analizatzen da (seinale hori ez da kronograma honetan ageri) eta SP kontagailua gaurkotzeko seinaleak sortzen dira: *Aldatu_SP* (*Gora* = 0 denez, beherantz egingo da kontaketa). Hala, 7. zikloko erloju-ertza heltzen denean, batetik, automata $E2$ egoerara igaroko da, eta, bestetik, SP kontagailuaren balioa eguneratuko da (–1 eginez) eta *Overflow* adierazlea ezabatuko da. $E2$ egoeran dagoen bitartean, memoria irakurtzen ari da (*CS* = 1 eta *OE* = 1). Ziklo horretan zehar *LD_Rdat* seinalea ere aktibatu da. Zikloaren bukaeran, hurrengo erloju-ertzarekin, memoriatik irakurri dena (memoriaren irteeran dagoena) Rdat erregistroan kargatuko da.



6.24. irudia. Pila: kronograma (*push* eta *overflow*).

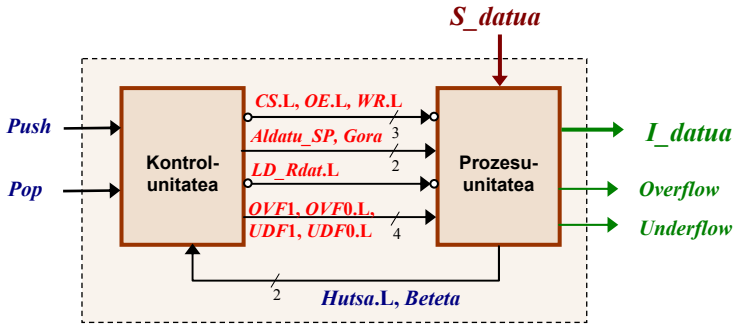
2. kasua

Kontrol-algoritmoko adar guztiak aztertu ditugu aurreko kasuan, bat izan ezik: *Pop* bat egin behar denean baina pila hutsik dagoenean (*Underflow*). Kasu hori 6.25. irudiko kronograman ageri da: bi *Pop* egiten dira, bata bestearen atzetik, pilan datu bakar bat dagoenean (SP = 1).



6.25. irudia. Pila: kronograma (*pop* eta *underflow*).

Pilaren diseinua bukatu dugu eta funtzionamendu logikoa egiaztatu, kronograma batzuen bidez. Beti bezala, bi partetan banatu dugu sistema: kontrol-unitatea eta prozesu-unitatea. Sistema osoa kontrolatzen du lehenengoak, eta behar diren kontrol-seinaleak sortu; bigarrenak, aldiz, kontrol-seinale horiek exekutatzen ditu, eta egindakoari buruzko informazioa itzultzen du.



6.26. irudia. Pilaren eskema logiko orokorra.

Egin dugun zirkuitua ez da aukera bakarra. Izan ere, hainbat modutan antola daiteke memoria-espazio bat pila gisa erabiltzeko, aplikazioen arabera. Konputagailu guztiek erabiltzen dute memoria nagusiaren zati bat pila gisa. Horretarako, erregistro berezi bat erabiltzen da memoria zati hori helbideratzeko: pilaren erakuslea, SP. Bi eragiketa egiten dira, eskuarki, pilaren gainean —`push ri` eta `pop rh`—, adibide honetan erabili dugun esanahi berarekin. Gaur egun ohikoa ez bada ere, badago beste konputagailu-arkitektura mota bat, pilazko konputagailuak, zeinetan pilarekin bakarrik lan egiten baita. Irakurle interesatuak irakur dezake bibliografian ageri den [10] liburua.

Bestalde, pilan oinarritutako konputazioa oraindik erabiltzen da hainbat kalkulagailutan. Kalkulagailu horietan, pilaren tontorrean dauden datuekin egiten dira eragiketak; beraz, eragiketa bat egin baino lehen, eragigaiak kargatu behar dira pilan. Esaterako, 3 + 4 eragiketa ordena honetan egin behar da: 3a pilan sartu, 4a pilan sartu, eta batu. Beste modu batean antolatzen da pila makina horietan: pilako gaina posizio finko bat da, eta datuak pilan desplazatzen dira, posizio batetik bestera, datuak sartzen edo ateratzen direnean. Pila egiteko, desplazamendu-erregistroak erabiltzen dira. Diseinu-ariketa gisa, proposatzen dugu pilazko kalkulagailu sinple bat egitea.

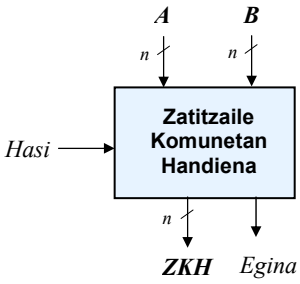
«

6.6. Ariketa

n biteko bi zenbakien Zatitzaile Komunetan Handiena (ZKH) kalkulatzen duen sistema digitala sortu behar da. Hasi *eta* Egina *seinaleek eragiketaren hasiera eta bukaera adierazten dute.*

A eta B zenbakien ZKHa kalkulatzeko, honako algoritmo hau erabil daiteke:

```
X := A; Y := B;
bitartean (X ≠ Y)
   baldin (X > Y)
      orduan    X := X - Y;
      bestela   Y := Y - X;
ambitartean
ZKH := X;
```



Diseinuarekin hasi baino lehen, ikus dezagun adibide pare bat ZKHa kalkulatzen duen algoritmoa ulertzeko. Izan bitez $A = 12$ eta $B = 5$; bi zenbakiak lehenak dira haien artean, eta, beraz, $ZKH(A, B) = 1$ da.

Izan ere, honela prozesatzen ditu zenbakiak aurreko algoritmoak ZKHa kalkulatzeko:

$X := 12$; $Y := 5$;

$12 = 5$?	ez →	$12 > 5$?	bai →	$X := 12 - 5 = 7$
$7 = 5$?	ez →	$7 > 5$?	bai →	$X := 7 - 5 = 2$
$2 = 5$?	ez →	$2 > 5$?	ez →	$Y := 5 - 2 = 3$
$2 = 3$?	ez →	$2 > 3$?	ez →	$Y := 3 - 2 = 1$
$2 = 1$?	ez →	$2 > 1$?	bai →	$X := 2 - 1 = 1$
$1 = 1$?	bai →	$\boldsymbol{ZKH = X = 1}$		

Edo, esaterako, $A = 12$ eta $B = 8$; orain, emaitzak $ZKH(A, B) = 4$ izan behar du.

$X := 12$; $Y := 8$;

$12 = 8$?	ez →	$12 > 8$?	bai →	$\boldsymbol{X} := 12 - 8 = \mathbf{4}$
$4 = 8$?	ez →	$4 > 8$?	ez →	$\boldsymbol{Y} := 8 - 4 = \mathbf{4}$
$4 = 4$?	bai →	$\boldsymbol{ZKH = X = 4}$		

Beraz, hainbat kenketa egin behar dira bi zenbakiekin, biak berdinak izan arte. Saia gaitezen prozesu-unitate bat garatzen eragiketa hori egiteko, eta, gero, kontrol-algoritmo bat sortzen prozesu-unitatea kontrolatzeko.

Prozesu-unitatea

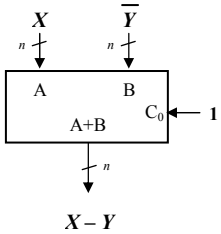
Zer zirkuitu erabili beharko ditugu sistema hori garatzeko? Algoritmoa ulertuta, ez da zaila erabakitzea beharrezko hardwarea.

Batetik, bi erregistro beharko ditugu, *A* eta *B* sarrerako datuak eta tarteko emaitzak gordetzeko: RX eta RY. Bestetik, batugailu bat erabili beharko dugu kenketak egiteko; era berean, konparagailu bat erabili behar da, RX-ren eta RY-ren arteko konparazioak egiteko (=, >).

Dakigunez, honako eragiketa hau egin behar da kenketa bat egiteko:

$$X - Y = X + (-Y) = X + \overline{Y} + 1$$

Eragiketa hori batugailu arrunt batean egin daiteke; horretarako, nahikoa da erabiltzea batugailuaren C_0 sarrera (sarrerako bururakoa) +1 egiteko, eta erabiltzea bigarren eragigaia ezeztatuta (bitez bit).



Bestalde, batzuetan $X - Y$ eta beste batzuetan $Y - X$ egin behar da. Hots, lehen eragigaia *X* edo *Y* izan daiteke eta bigarrena, ezeztatuta, *Y* edo *X*. Beraz, **multiplexoreak** erabili beharko ditugu eragigai egokiak **aukeratzeko.**

Kenketen emaitza RX edo RY erregistrora eraman behar da; beraz, erregistro horien sarrerak ere multiplexatu beharko ditugu, hasieran *A* eta *B* sarrerako datuak kargatu behar direlako, baina, eragiketan zehar, batugailuaren irteera.

Eragiketaren erantzun-denbora ez da konstantea, sarrerako datuen araberakoa baizik. Beraz, egokia da adierazle bat izatea (JK biegonkor bat) eragiketa bukatu dela abisatzeko (*Egina*).

6.27. irudian ageri da sistemaren prozesu-unitatea. Prozesu-unitateak honako kontrol-seinale hauek behar ditu:

- *LD_RX, LD_RY*: bi erregistroak, RX eta RY, kargatzeko seinaleak (LD.L)
- *h*1: erregistroetan kargatzen diren datuak aukeratzeko

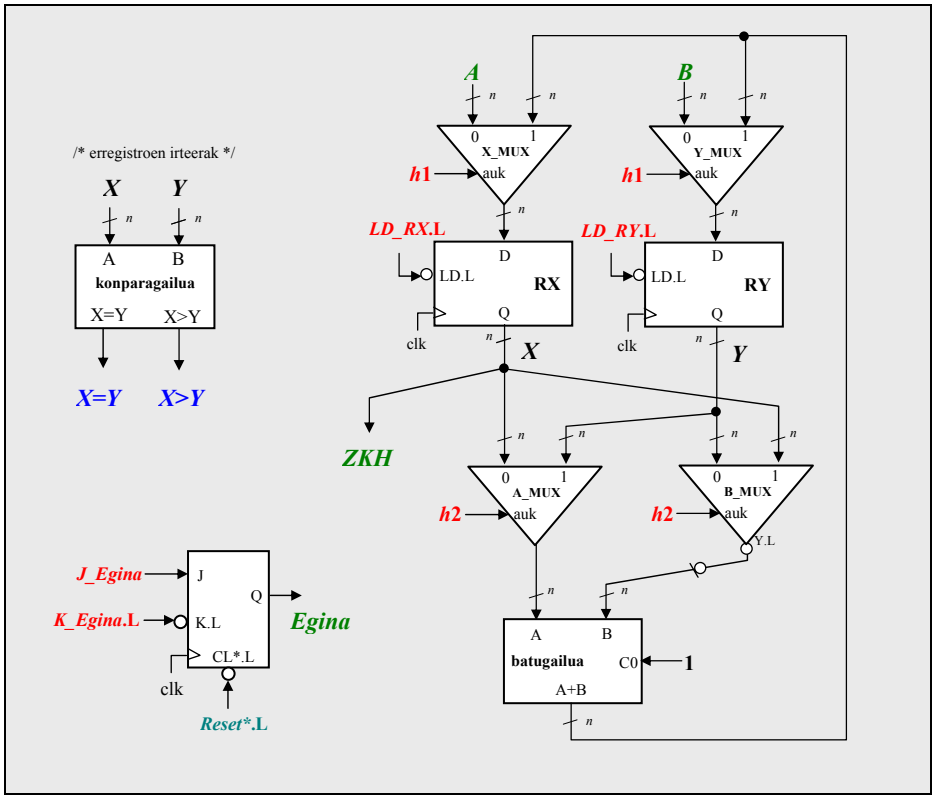
  *h*1 = 0 → sarrerako datuak
  *h*1 = 1 → kenketaren emaitza

- $h2$: kenketarako eragigaiak aukeratzeko seinalea

  $h2 = 0 \rightarrow X - Y$

  $h2 = 1 \rightarrow Y - X$

- *J_Egina*, *K_Egina*: *Egina* adierazlea kontrolatzeko (J, K.L)



6.27. irudia. Zatitzaile komunetan handiena: prozesu-unitatea.

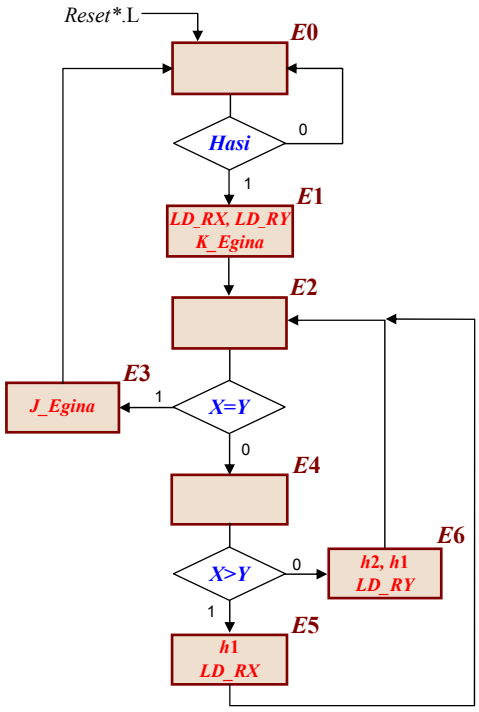
Kontrol-algoritmoa

Kontrol-algoritmoaren eginkizuna hainbat egoeratan bana daiteke. Adibide honetarako, egoera "asko" dituen algoritmo bat egingo dugu (6.28. irudia).

Lehenengo egoeran ($E0$), eragiketa egin behar den ala ez aztertzen da: *Hasi* seinalea. Eragiketari ekiteko, hasierako balioak kargatu behar dira RX eta RY erregistroetan, eta ezabatu aurreko eragiketan aktibatu den *Egina* adierazlea ($E1$ egoeran). Hortik aurrera, begizta batean sartuko gara, RX eta

RY erregistroen edukiak berdinak izan arte ($E2 - E4 - E5 - E2 - ...$ edo $E2 - E4 - E6 - E2 - ...$). Begiztan zehar, eguneratu egiten da RX edo RY erregistroen edukia kenketen emaitzekin.

RX eta RY erregistroen edukiak, X eta Y, berdinak direnean, eragiketa bukatutzat ematen da ($E2 - E3 - E0$).



6.28. irudia. ZKH: kontrol-algoritmoa.

7 egoera ditu aurreko algoritmoak; modu horretan, 3 biegonkor beharko ditugu eta 8 sarrerako multiplexoreak. Kontrol-unitatearen hardwarea laburtzeko asmoz, erraza da sinplifikatzea aurreko algoritmoa, eta, esaterako, 2 egoera bakarrik dituen kontrol-algoritmoa sortzea. Ariketa gisa uzten da. Era berean, ariketa gisa uzten dugu kontrol-unitatearen gauzatzea (multiplexoreen bidez) eta sistema osoaren funtzionamendua egiaztatzea, kronograma baten bidez, zikloz ziklo.

«

6.7. Ariketa

Sistema digitalak diseinatzen direnean, ohikoa da errore tipiko batzuk egitea, batik bat eskarmentu handia ez dagoenean. Oinarrizko errore horien laburpena egin nahi dugu ariketa honetan.

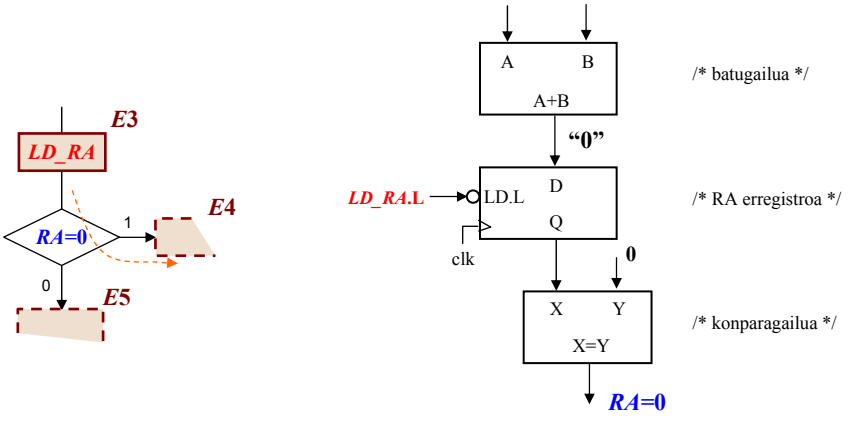
■

Behin eta berriz esan dugun moduan, erroreak detektatzeko dugun tresna nagusia konputagailuaren bidezko simulazioa da. Ondo planifikatutako simulazio zabalen emaitzak erabakigarriak dira diseinuko errore logikoak detektatzeko. Hala ere, sistema logikoa oso konplexua denean —prozesadore oso bat esaterako—, simulazioak ere oso konplexuak dira (konbinazio logikoen kopurua itzela da), eta oso zaila da balizko erroreak aurkitzea eta errorerik gabeko diseinuak egitea.

Sistema logikoak sinpleak direnean —liburu honetakoak kasu—, ez da zaila erroreak detektatzea simulazioen emaitzak analizatuz. Ariketa honetan, sistema logikoen zatitxo batzuk analizatuko ditugu, ohiko errore nagusiak azaltzeko.

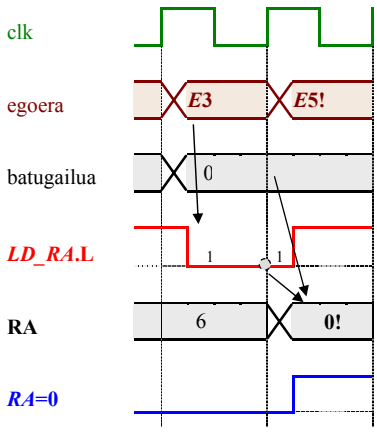
1. errorea: kontuan ez hartzea zirkuituen izaera (sinkronoa/asinkronoa) (I)

Demagun eragiketa aritmetiko baten emaitza kargatu behar dela erregistro batean, eta begiratu emaitza 0 denetz. 6.29. irudian, kontrol-algoritmoaren zati bat eta dagokion prozesu-unitatearen zatia ageri dira. Batugailuaren emaitza 0 izan da; beraz, espero beharko al genuke $E3 \rightarrow E4$ egoera-trantsizioa izatea kontrol-automatan?



6.29. irudia. 1. errorea: kontrol-algoritmoa eta prozesu-unitatea.

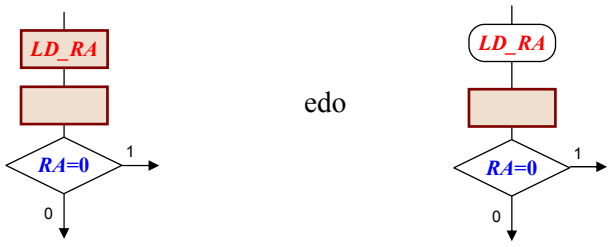
Errore arrunt bat sartu da aurreko zirkuituan, eta oso ondo detekta daiteke kronograma batean. Ikus dezagun.



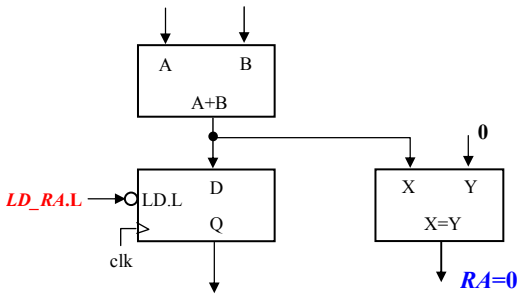
6.30. irudia. 1. errorea: kronograma.

Erloju-zikloa heltzen denean, kontrol-automata egoeraz aldatzen da, eta *E*3 egoeran sartzen da. Ondorioz, *LD_RA* kontrol-seinalea sortzen du. Adi! batugailuaren emaitza kargatzeko kontrol-seinalea sortu egin da, baina ez da exekutatuko hurrengo erloju-ertza heldu arte, erregistroaren karga sinkronoa delako. Hala, *E*3 egoeran zehar, erregistroak duen balioa, 0rekin konparatzen ari dena, **balio zaharra** da, ez batugailuaren azken emaitza. Hurrengo erloju-zikloa heltzean, bi gauza gertatuko dira. Batetik, *LD_RA* seinalea exekutatuko da eta 0a kargatuko da RA erregistroan; eta, bestetik, automata *E*5 egoerara igaroko da, erabakia hartu denean RA = 0 ez zelako oraindik!

Soluzioa erraza da, eta bat baino gehiago dago. Zuzenena, hau da: *LD_RA* seinalea ziklo batean sortu, eta haren eragina hurrengo zikloan analizatu. Esaterako,



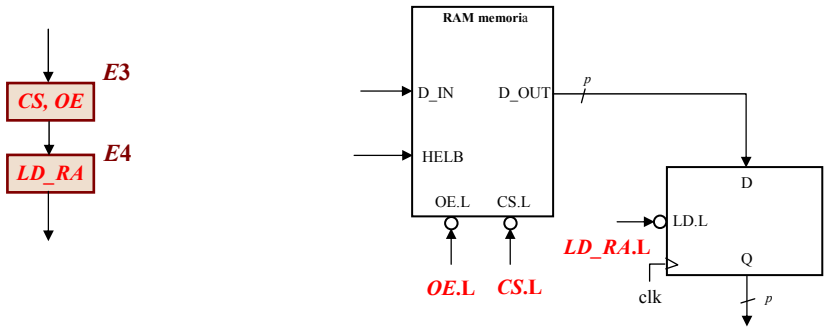
Bestela, jatorrizko kontrol-algoritmoaren zatia mantendu nahi bada, orduan prozesu-unitatea aldatu beharko genuke: konparazioa ez da egin behar erregistroaren emaitzarekin (hurrengo zikloan kargatuko baita), baizik eta batugailuaren irteerarekin, (hots, ziklo horretan erregistroaren sarreran dagoenarekin).



Lehenengo zein bigarren kasuan, irakurleak egiazta dezake kronograma batean sistemaren funtzionamendu egokia.

2. errorea: kontuan ez hartzea zirkuituen izaera (sinkronoa/asinkronoa) (II)

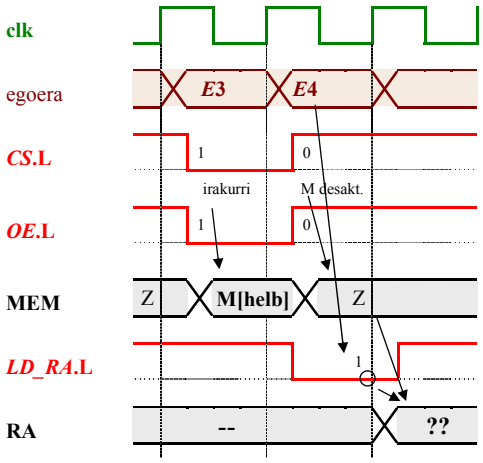
Ikusi berri dugun errorearen beste bertsio bat ageri da 6.31. irudian. Memoriako hitz bat irakurri da, eta emaitza erregistro batean kargatu behar da.



6.31. irudia. 2. errorea: kontrol-algoritmoa eta prozesu-unitatea.

*E*3 egoeran, RAM memoria irakurtzeko behar diren bi seinaleak sortuko ditu kontrol-unitateak: *CS* (gaikuntza-seinalea) eta *OE* (irakurtzeko seinalea). Memoria irakurri ondoren, egoeraz aldatuko da (*E*4) eta emaitza kargatzeko kontrol-seinalea sortuko du.

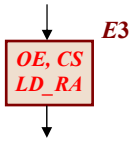
Errorea garbia da kasu honetan ere, eta erraz ikus daiteke kronograma batean.



6.32. irudia. 2. errorea: kronograma.

Automata *E*3 egoeran dagoen bitartean, RAM memoria irakurtzen ari da (*CS* eta *OE*). Baina ziklo horretan ez dago aktibatuta erregistroa kargatzeko seinalea! Erloju-ertza heltzean, automata *E*4 egoerara igaroko da, *CS* eta *OE* seinaleak desaktibatu, eta *LD_RA* aktibatuko da. Beraz, *E*4 egoeran, erregistroa prest dago datu berria kargatzeko, baina memoriaren irteeran ez daukagu aurreko zikloan irakurritakoa[23]. Erloju-ertza gertatzean, balio oker bat kargatuko da erregistroan.

Izan ere, RAM memoria (edo ROM memoria, multiplexoreak...) ez dago erlojuaren menpe, eta dagozkion kontrol-seinaleak aktibatuta daudenean bakarrik izango dugu emaitza irteeran. Beraz, memoria irakurtzen den ziklo berean aktibatu behar da karga-seinalea; hala, hurrengo zikloaren hasieran kargatuko da erregistroan memorian irakurritako balioa.



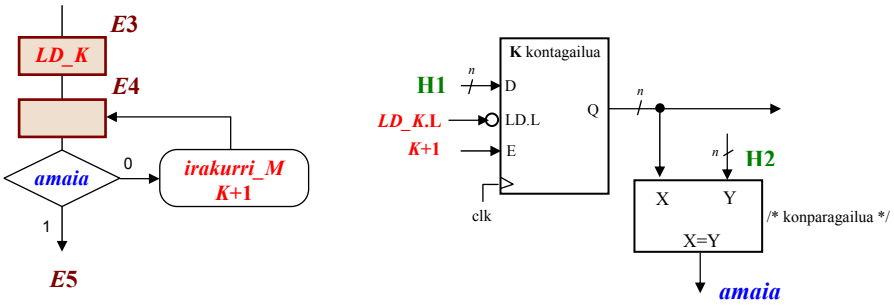
Erraza da egiaztatzea, kronograma batean, ondo funtzionatuko dutela orain memoriaren irakurketak eta erregistroaren kargak.

[23] Memoria desaktibatuta dagoenean, irteera Z egoeran dago (ez 0 edo 1, “deskonektatuta” baizik).

3. errorea: begizten kontrol zehatza ez izatea

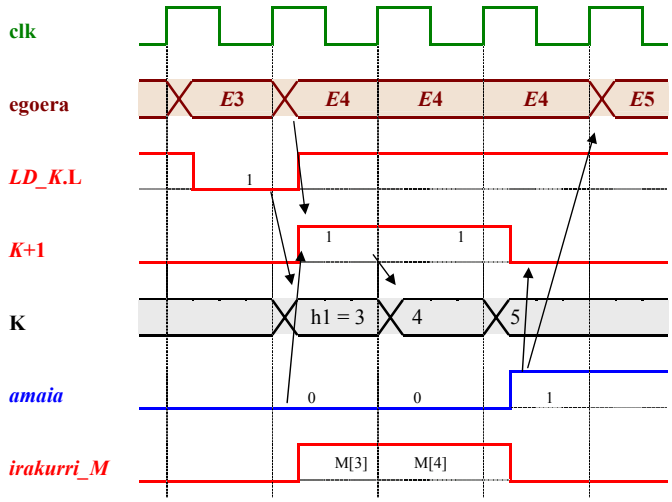
Kontrol-algoritmoetan gehien ageri den egitura begizta da: eragiketa bera errepikatu behar da hainbat aldiz. Errepikapen kopurua ondo kontrolatu behar da; bestela, erraza da begiztako iterazio bat gutxiago edo gehiago exekutatzea, eta, ondorioz, gaizki kontrolatzea sistema osoa.

Ikus ditzagun adibide batzuk. Memoria bat irakurri behar da sistema digital batean, **H1** helbidetik **H2** helbidera, biak barne. Horretarako, honako kontrol-algoritmo eta prozesu-unitate hauek garatu dira:



6.33. irudia. 3. errorea: kontrol-algoritmoa eta prozesu-unitatea.

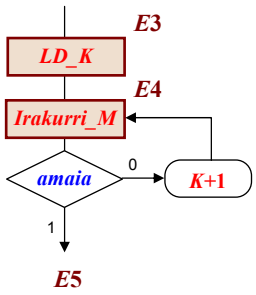
Kontrol-begizta horrek ez du ondo kontrolatzen pasatu behar den ziklo kopurua. Errorea non dagoen garbi ikusteko, hau litzateke funtzionamenduaren kronograma kasu partikular batean: H1 = 3 eta H2 = 5.



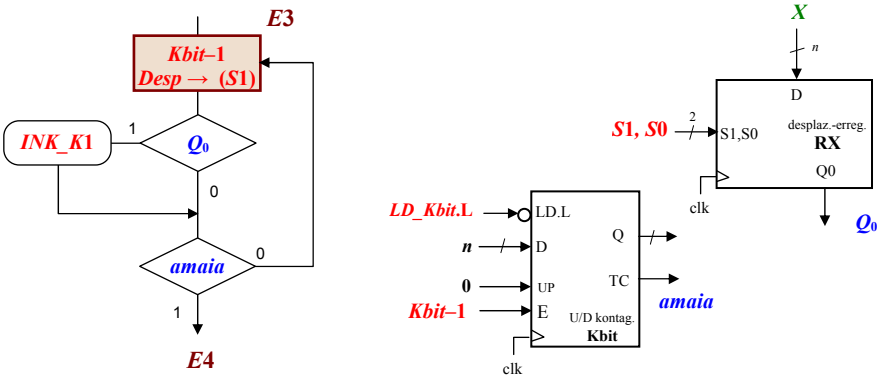
6.34. irudia. 3. errorea: kronograma.

3, 4 eta 5 helbideetako memoria-posizioak irakurtzea zen asmoa, baina gaizki kontrolatu da begiztaren bukaera, eta azken helbidekoa ez da irakurri. Adi! sistema kontrolatu den moduan, begizta bukatzeko konparazioa H2 + 1 helbidearekin egin beharko litzateke. Ziur asko, errazagoa da kontrol-algoritmoa aldatzea; izan ere, irakurtzeko seinalea begiztako ziklo guztietan eman behar da, azkenekoan ere, eta nahikoa da, horretarako, kontrol-seinale hori baldintza gabekoa izatea (egoeran bertan adieraztea).

Esaterako,

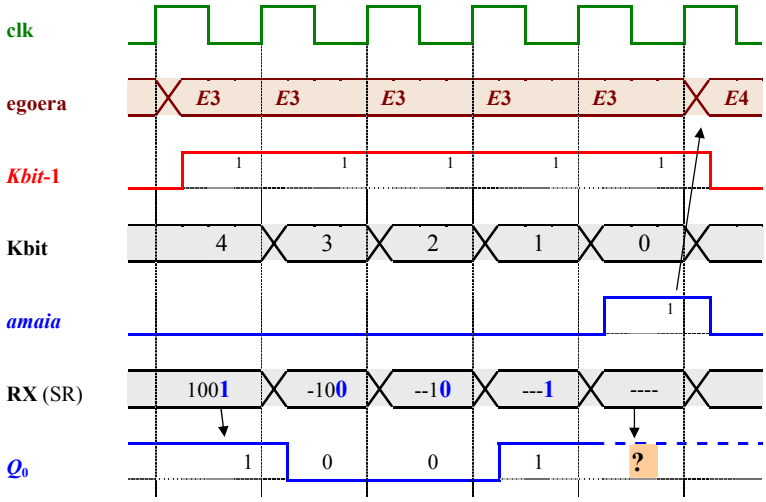


Antzeko errorea ageri da 6.35. irudiko kontrol-begiztan. Kontatu behar dugu zenbat 1eko dauden zenbaki batean. Zenbakia, n bitekoa, desplazamendu-erregistro batean kargatu da, eta bit kopurua, n, kontagailu batean. Automata $E3$ egoerara igaro da, eta hor begizta batean sartuko da, zenbakiaren bit guztiak, banan-banan, prozesatzeko. Begiztaren barruan, desplazamendu-erregistroaren Q_0 bita analizatzen da, eta, 1 bada, K1 kontagailua inkrementatu egiten da. Ziklo guztietan, Kbit kontagailua dekrementatzen da, 0ra heldu arte, eta erregistroaren edukia bit bat desplazatzen da eskuinera.



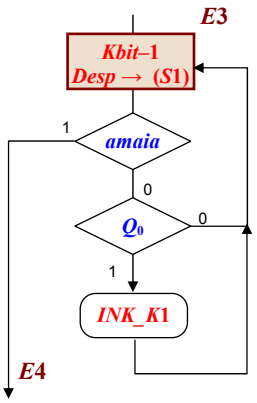
6.35. irudia. 3. errorea: bigarren adibidea (K1 kontagailua ez da ageri, ez baitu zerikusirik errorearekin).

Aurreko sisteman, n+1 aldiz errepikatzen da analisi-begizta, ez n, kronograma honetan ageri den moduan (n = 4 kasurako).



6.36. irudia. 3. errorea: 2. adibideari dagokion kronograma.

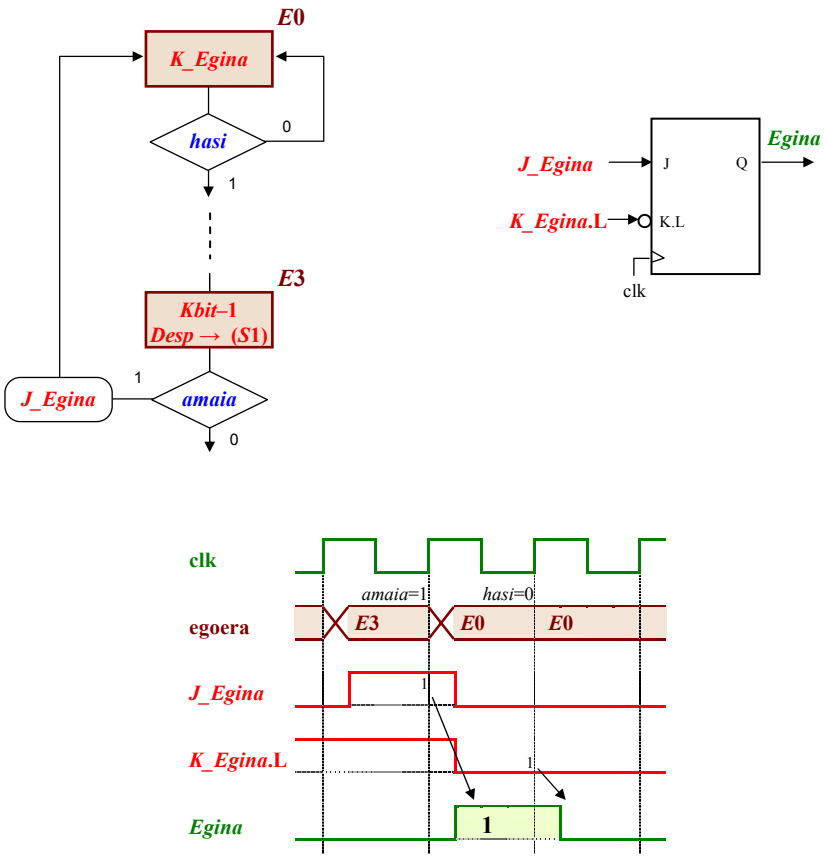
Garbi dago, 5 ziklotan mantentzen da automata $E3$ egoeran, eta, beraz, 5 aldiz analizatzen da desplazamendu-erregistroko Q_0 bita. Errorea zuzentzeko, nahikoa da egoera horretan egiten den analisiaren ordena aldatzea: lehendabizi, begizta bukatu den ala ez, eta, gero, Q_0 bita.



Bost ziklotan mantenduko da automata $E3$ egoeran, baina, azkenekoan, *amaia* = 1 denean, ez da Q_0 bita analizatuko (eta, beraz, ez da K1 kontagailuaren balioa gehituko).

4. errorea: seinaleen iraupena

Ikus dezagun azken adibidea. Eragiketa bat egin da (dena delakoa) eta, bukaeran, adierazle bat aktibatu da, eragiketa egina dagoela abisatzeko. Honela kontrolatu da adierazle hori:

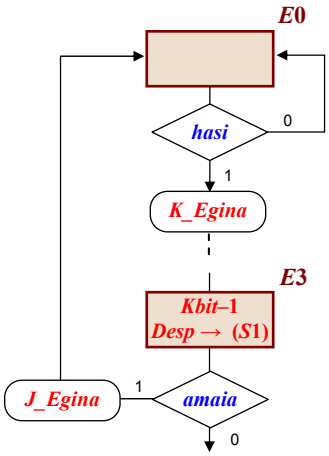


6.37. irudia. 4. errorea: kontrol-algoritmoa, prozesu-unitatea eta kronograma.

Berez, aurreko algoritmoan ez dago errorerik, baina... zertarako sortu da *Egina* seinalea? Pertsona batek ikus dezan? Makina batek trata dezan? Lehenengo kasua bada helburua, hau da, sistemaren erabiltzailearentzat sortu bada *Egina* seinalea, orduan gaizki dago. Hain zuzen ere, *Egina* adierazlea erloju-ziklo bakar batean mantenduko da aktibatuta: egoera batean sortzen da aktibatzeko seinalea, eta hurrengoan desaktibatzekoa. Eskuarki, oso

denbora txikia da ziklo bat, eta pertsona batek ez du aukerarik izango adierazlea ikusteko, eta diseinua zuzendu beharko dugu. Adierazle hori beste makina batek tratatu beharko balu, agian nahikoa izango litzateke ziklo bat, eta JK biegonkorra soberan legoke.

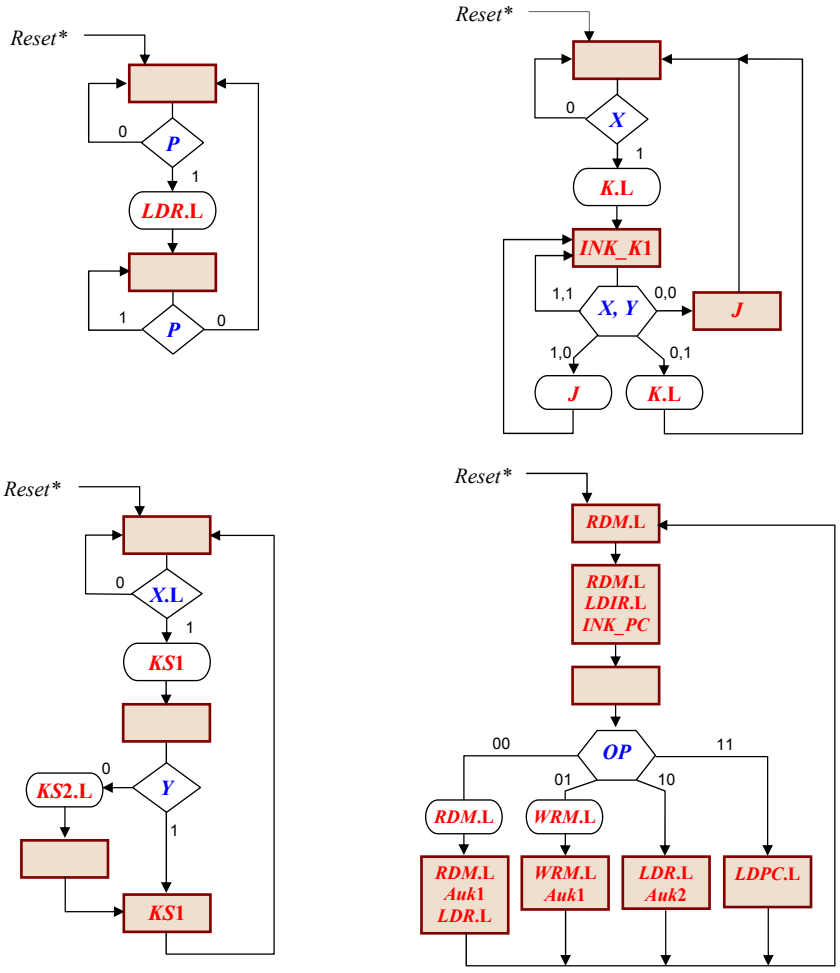
Soluzioa erraza da: ez desaktibatzea beti adierazlea *E*0 egoeran, baizik eta hurrengo eragiketa abiatzear dagoenean. Honela:



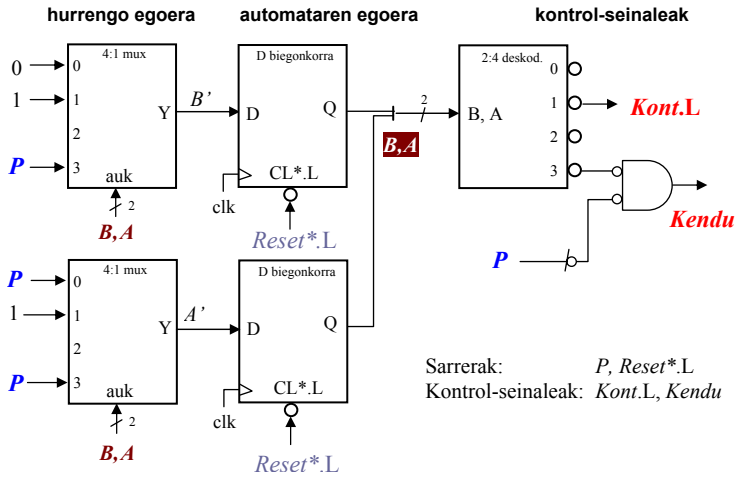
«

6.6. ARIKETAK

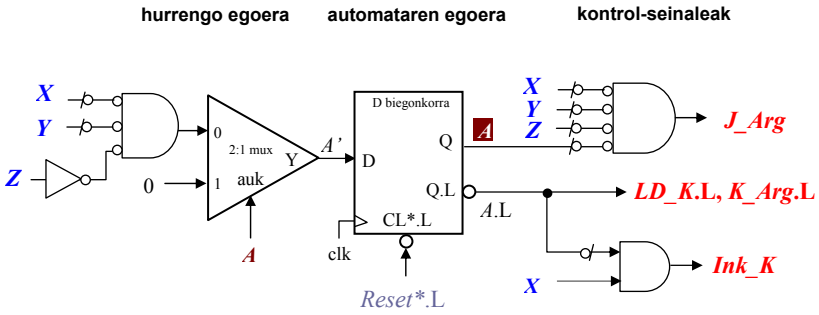
6.1. Sistema digital batzuen kontrol-algoritmoak ageri dira irudietan. Kasu bakoitzerako: (a) kodetu egoerak eta sortu egoeren arteko trantsizio-taula; (b) eraiki kontrol-unitatea multiplexoreak eta D biegonkorrak erabiliz, eta sortu kontrol-seinaleak; eta (c) egin kontrol-unitatearen funtzionamendua islatuko duen kronograma bat. Kronogrametan, batetik, sarrera-seinaleak, eta, bestetik, automataren egoerak eta kontrol-seinaleak ageri behar dira; sarrera-seinaleak definitu beharko dituzu, nahi duzun moduan, automataren egoera-trantsizio guztiak analizatu ahal izateko (hartu, erreferentzia gisa, ariketa ebatzietan erabilitakoak). Kontrol-seinaleen eta sarreren logika (.H edo .L) kontrol-algoritmoetan bertan ageri da.



6.2. Bi sistema digital sinpleren kontrol-unitateak ageri dira irudietan. Kasu bakoitzerako: (a) egin kontrol-unitatearen funtzionamendua islatzen duen kronograma bat. Beharrezkoa da kronogrametan agertzea, batetik, sarrera-seinaleak eta, bestetik, automataren egoerak eta kontrol-seinaleak; sarrera-seinaleak definitu beharko dituzu, sistemaren portaera osoa analizatu ahal izateko; eta (b) idatzi kontrol-unitateari dagokion ASM grafoa.

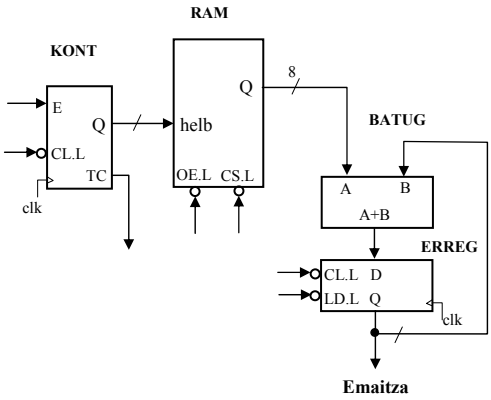


1. kontrol-unitatea



2. kontrol-unitatea

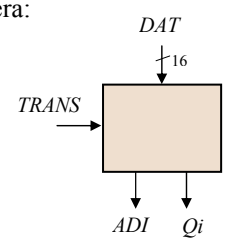
6.3. Sistema digital baten atal batean, RAM memoria bat erabiltzen da byte bateko datuak gordetzeko. *HASI* izeneko kontrol-seinalea aktibatzen denean, lehen 16 datuen batura kalkulatu behar da, eta horretarako irudiko zirkuitua antolatu da.
Egin ezazu kontrol-algoritmo bat zirkuitua kontrolatzeko, aipatutako eragiketa egin dezan. Gero, eraiki ezazu kontrol-unitatea, sortu kontrol-seinaleak eta analizatu sistema osoaren portaera kronograma baten bidez (memoriaren edukia, adibidez, honako hau da: 0, 1, 2, 3...).
Zenbat bitekoak izango dira batugailua eta erregistroa, ez badugu nahi gainezkatzea gertatzerik? eta kontagailua?



6.4. Sistema digital batek 16 biteko datu bat transmititu behar du seriean, bitez bit. Horretarako, desplazamendu-erregistro bat erabiltzen du. Hasieran, *TRANS* seinalea aktibatzen denean, transmititu behar den datua (sarrera bat) desplazamendu-erregistroan kargatzen da (bit guztiak batera), eta, gero, 16 desplazamendu egiten dira, eskuinera; hala, bit guztiak erregistroko Q_0 irteeran ageriko dira, zikloz ziklo; irteera hori da, hain zuzen ere, transmitituko den bita.
Adibide sinple batez (4 bit) ikus daiteke sistemaren portaera:

transmititu behar den datua:	0011
kargatu erregistroan (*TRANS* = 1):	$Q_3Q_2Q_1\mathbf{Q_0} = 001\mathbf{1}$
desplazatu bit bat eskuinera	$Q_3Q_2Q_1\mathbf{Q_0} = x00\mathbf{1}$
desplazatu bit bat eskuinera	$Q_3Q_2Q_1\mathbf{Q_0} = xx0\mathbf{0}$
desplazatu bit bat eskuinera	$Q_3Q_2Q_1\mathbf{Q_0} = xxx\mathbf{0}$
desplazatu bit bat eskuinera	$Q_3Q_2Q_1\mathbf{Q_0} = xxxx$

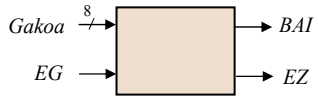
Beraz, $Q_0 \rightarrow 1, 1, 0, 0$



Bitak transmititzeaz gain, transmisioaren hasiera eta bukaera adierazi behar dira: transmisioa hasten denean, seinale batek (*ADI* esaterako) 1 balioa hartuko du, eta bukaeran 0a.

Diseinatu sistema hori honako zirkuitu hauek erabiliz: desplazamendu-erregistro bat datua hartzeko eta desplazatzeko, kontagailu bat bit guztiak transmititu direla egiaztatzeko, eta JK biegonkor bat (*ADI*) transmisioa egiten ari dela adierazteko. Antolatu prozesu-unitatea, zehaztu erabili behar diren kontrol-seinaleak eta, gero, proposa ezazu ASM kontrol-algoritmo bat eta dagokion kontrol-unitatea.
Diseinua ondo dagoela egiaztatzeko, egin ezazu kronograma bat eta aztertu, pausoz pauso, diseinatu duzun sistemaren portaera. Errore bat detektatzean, zuzendu eta errepikatu kronograma.

6.5. Sistema digital jakin batek gako-hitz batzuk erabiltzen ditu eragiketa jakin bat egiteko. Sistemaren azpiatal batek kontrolatzen ditu gakoak (8 bitekoak) eta adierazten du zuzenak direnetz. Onartzen diren gako guztiak 16 posizioko ROM memoria batean daude. Gako bat egiaztatu behar denean, *EG* seinalea aktibatuko da; une horretan, gakoa erregistro batean kargatu eta ROM memorian dauden gakoekin konparatuko da, banan-banan. Sistemak aurkitzen badu memorian berdina den gako bat, ontzat emango du sarrerako gakoa eta BAI seinalea aktibatuko du; aldiz, memoria osoa irakurrita, ez badu aurkitu gakoa, EZ seinalea sortuko du.



Diseina ezazu sistema horretarako prozesu-unitate bat zirkuitu hauek erabiliz: 16×8 biteko ROM memoria bat, 4 biteko kontagailu bat (ROM memoriaren helbideak emateko), 8 biteko erregistro bat (prozesatzen ari den gakoa gordetzeko), 8 biteko konparagailu bat (gakoa eta memoriako hitzak konparatzeko) eta bi JK biegonkor (*BAI* eta *EZ* adierazleetarako). Adierazi argi eta garbi kontrolatu behar dituzun kontrol-seinaleak. Gero, egin ezazu kontrol-algoritmoa, eta eraiki kontrol-unitatea eta kontrol-seinaleak.

7. kapitulua

OINARRIZKO PROZESADORE BATEN DISEINUA

Sistema digitalen oinarrizko osagaiak eta sistema digitalak diseinatzeko metodologia landu ditugu dagoeneko; horien bitartez, edozein sistema digital diseina daiteke. Hala, kontrol-unitateak eta helburu bereziko sistema batzuk diseinatu ditugu aurreko kapituluan.

Kapitulu honetan, ordea, helburu orokorreko sistema digital bat, hau da, prozesadore bat, diseinatuko dugu. Sistema digital oso konplexuak dira prozesadoreak; beraz, ezinbestean, hainbat eta hainbat sinplifikazio egin beharko ditugu. Izan ere, oso sinplea den oinarrizko prozesadore didaktiko bat diseinatuko dugu. Helburua bikoitza da: batetik, liburuan azaldu ditugun gaiak lantzea, beste sistema digital baten diseinuaren bidez; eta, bestetik, prozesadoreen funtzionamendua ulertzeko oinarrizko kontzeptuak azaltzea. Hori dela eta, diseinuari ekin baino lehen, konputagailuen egitura, aginduen exekuzioa eta abar laburtuko ditugu.

7.1. SARRERA

Sistema digitalak edonon aurki daitezke gaur egun: edariak saltzeko makina automatikoetan zein industria-prozesuak kontrolatzen dituzten robotetan, sakelako telefonoetan zein automobiletan. Sistema digital horien eginkizunak oso ezberdinak dira, helburu jakin bat(zuk) lortzeko diseinatu baitira. Hori dela eta, horrelako sistemei **helburu berezikoak** edo zehatzekoak deritze[24]. Mota horretako sistema batzuk diseinatu ditugu 6. kapituluan, hala nola biderkagailu bat edota datu-pila bat.

Baina, ziur aski, sistema digital garrantzitsuena **konputagailua** da. Ondo ezagutzen dugunez, konputagailuak ez dira eginkizun jakin bakar baterako erabiltzen; hain zuzen ere, haien aplikazio-eremua oso zabala eta joria da. Horregatik, **helburu orokorreko** sistema digitalak dira konputagailuak.

Erabilgarritasun zabal horren zioa diseinuan datza: konputagailua "agindu" jakin batzuk exekutatzeko diseinatzen da. Agindu horiek oinarrizko eragiketak adierazten dituzte, hala nola datu bat toki batetik bestera mugitzea konputagailuaren barruan; datu batzuekin eragiketa aritmetiko edo logiko bat egitea; exekutatu beharreko hurrengo agindua zein den erabakitzea; eta abar. **Aginduak** sekuentzietan antolatzen dira eta, datuekin batera, **programak** osatzen dituzte. Aginduak eta datuak aldatuz, eginkizun oso desberdinetarako programak idatz daitezke. Horri esker, helburu orokorreko sistema digitala da konputagailua: exekutatzen duen programa aldatuz gero, gauza ezberdinak egin ditzake.

Beraz, bi alderdi bereizten dira konputagailu batean: (a) **osagai fisikoak** (*hardwarea*): prozesadorea, memoria, periferikoak... egitura jakin baten arabera antolatuta —sistemaren arkitektura—; eta (b) **programak** (*softwarea*), konputagailuak egin behar duena zehazten dutenak. Kapitulu honetan, prozesadoreen oinarrizko hardwarea —prozesu-unitatea zein kontrol-unitatea— aztertuko dugu, bai eta nola prozesatu oinarrizko programak. Helburua ez da programak nola idatzi behar diren aztertzea, prozesadore baten diseinua eta funtzionamendua ulertzea baizik.

Hurrengo ataletan ikusiko dugun moduan, beste edozein sistema digitalen diseinua bezala tratatu behar da konputagailuen diseinua (nahiz eta, hori bai, askoz konplexuagoa izan); beraz, liburuan analizatu ditugun osagaiak eta

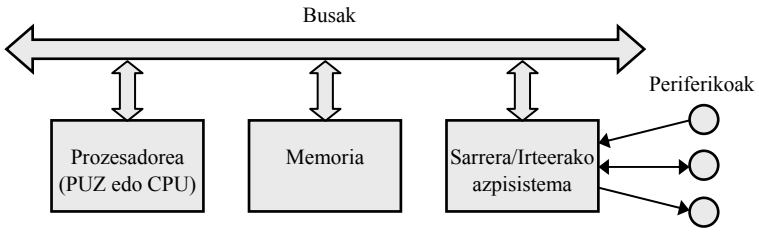
---

[24] Ohikoa da helburu bereziko sistema digitalen osagai guztiak —kontrol-unitatea eta prozesu-unitatea— zirkuitu integratu bakar batean sartzea: ASIC, *Application Specific Integrated Circuit*, izenekoan.

diseinu-metodologia erabiliko ditugu prozesadore bat diseinatzeko. Hori da, hain zuzen ere, kapitulu honen asmoa.

7.2. OINARRIZKO KONTZEPTUAK

Konputagailuak sistema digital oso konplexuak dira. Konplexutasunari aurre egiteko, ohikoa da sistema konplexua sinpleagoak diren hainbat azpisistematan banatzea, eta haien arteko konexioak definitzea. Konputagailuen kasuan, John von Neumann matematikariak 1945. urtean proposatutako "arkitektura" erabiltzen da oraindik, funtsean, oinarrizko egitura gisa. 7.1. irudian ageri da von Neumann arkitektura.



7.1. irudia. Konputagailuen von Neumann arkitektura.

Hiru azpiatal bereizten dira von Neumann arkitekturan:

- **Prozesadorea** (**CPU,** *central processing unit,* edo PUZ, prozesatzeko unitate zentrala). Programetako aginduak eta eragigaiak irakurri eta prozesatzen ditu. Gainerako sistema digitaletan bezala, bi partetan bana daiteke: kontrol-unitatea eta prozesu-unitatea. Ohikoa da, baita ere, mikroprozesadore izena erabiltzea.

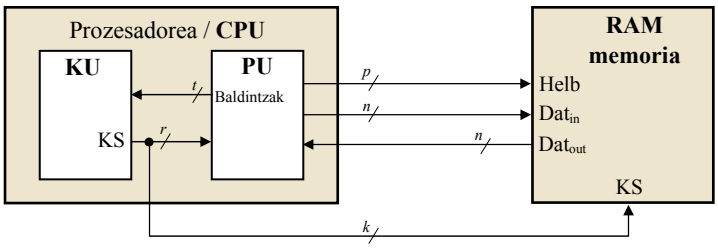
- **Memoria.** Exekutatu behar diren programak eta haien datuak gordetzen dira memorian. Hortik eskuratu behar dira programako aginduak eta eragigaiak, eta hor idatzi behar dira emaitzak. Funtsezko atala da konputagailu baten memoria; programen exekuzioa azkarra izan dadin, modu hierarkikoan antolatzen da memoria: cache memoria, memoria nagusia, diskoak, eta abar (ikus 5.8. atala).

- **Sarrera/irteerako azpisistema.** Konputagailuaren eta kanpo aldearen arteko informazio-transferentziaz arduratzen da. Informazioa kanporatzeko edo barneratzeko, sarrera/irteerako gailuak edo periferikoak erabiltzen dira: pantaila, teklatua, sagua, inprimagailuak, diskoak, eta abar. Gailu horiek kontrolatzeko, eta komunikazioa modu egokian gauza dadin, kontroladore bereziak erabiltzen dira periferiko bakoitzerako; kontroladore horien bidez transferituko dira datuak, esaterako, memoriatik pantailara edo teklatutik memoriara.

- **Busak.** Konputagailuaren osagai edo azpisistema guztiak konektatzen dituzten "bideak" dira, osagaien arteko informazio-trukea gauzatzeko. Hiru motatako busak bereizten dira usu: **helbide-busa,** memoriako posizioen zein periferikoen helbideak adierazteko; **datu-busa,** transferitu behar diren datuak garraiatzeko; eta **kontrol-busa,** egin behar den eragiketarako behar diren kontrol-seinaleak adierazteko (esaterako, memoriako *WR*, *CS*...).

7.2.1. Prozesadoreen osagai nagusiak

Esan dugun bezala, prozesadorearen funtsezko lana aginduak eskuratzea eta exekutatzea da. Prozesadorea diseinatzeko, ohiko banaketa erabiltzen da: kontrol-unitatea (KU) eta prozesu-unitatea (PU). 7.2. irudian, sistemaren eskema bat ageri da.



7.2. irudia. Prozesadorearen goi-mailako egitura, eta CPUren eta memoriaren arteko konexioak. Sinplifikatzearren, ez dugu sarrera/irteera atala kontuan hartu. (KS = kontrol-seinaleak)

Egitura eta osagai oso desberdineko prozesadoreak daude, baina, eskuarki, honako gailu hauek izango ditu prozesu-unitateak:

- **Programaren kontagailua,** PC (*program counter*): exekutatu behar den aginduaren memoria-helbidea gordetzen duen erregistro berezia. Oro har, bi eragiketa egiten dira erregistro horretan:
  - Edukia gehitu, eskuarki exekutatuko den hurrengo (ondoz ondoko) agindua erakusteko.
  - Helbide bat kargatu, ondoz ondokoa ez den agindu bat erakusteko.
- **Agindu-erregistroa,** IR (*instruction register*): exekutatzen ari den agindua gordetzen duen erregistro berezia.
- **Erregistro-multzoa,** EM (*register bank*): hainbat erregistro, exekutatu behar diren aginduen eragigaiak, tarteko emaitzak... aldi baterako gordetzeko (ikus 5.2. atala).

  Erregistro-multzoa memoria-hierarkiaren beheko maila da: edukiera txikiko memoria bat da (esaterako, 64 biteko 128 hitz), baina irakurketak eta idazketak oso azkar egin daitezke. Beraz, hor gordetzen dira behin eta berriz erabili behar diren eragigaiak, emaitzak, eta abar.
- **Unitate aritmetiko/logikoa,** UAL (ALU, *arithmetic and logic unit*): oinarrizko eragiketa aritmetikoak —batuketa, kenketa, biderketa...— zein logikoak —*and*, *or*, *xor*, *not*...— exekutatzen dituen gailua. Eskuarki, emaitzari buruzko informazioa ere eskaintzen du (emaitza 0 den, edo negatiboa, eta abar).

PC eta IR helburu bereziko erregistroak dira, aginduaren helbidea eta agindua bera gordetzeko soilik erabiltzen direlako, hurrenez hurren, aginduen exekuzio-prozesuan zehar. Erregistro-multzokoak, berriz, helburu orokorrekoak dira, programatzaileak erabil ditzakeelako haren programetan, nahi dituen datuak gordetzeko.

Osagai askoz gehiago dago prozesadore komertzialetan, baina gure helburuetatik at geratzen dira.

7.2.2. Prozesadorearen aginduak

Aipatu dugun bezala, aginduak exekutatzeko gai den sistema digitala da konputagailua. Baina, zein agindu?

Konputagailu bakoitzak **agindu-multzo** (*instruction set*) jakin eta zehatza exekuta dezake; agindu horiek konputagailuaren **makina-lengoaia** definitzen dute. Izan ere, sistemaren arkitektura makina-lengoaia jakin bat exekutatzeko diseinatzen da. Beraz, konputagailuaren arkitektura eta makina-lengoaia txanpon bereko aurki eta ifrentzua dira. Hala, agindu horiek exekutatzeko erabiltzen den hardwareari (konputagailuaren muinari) ISA (*instruction set architecture*) deitu ohi zaio.

Konputagailuek exekuta ditzaketen aginduak oso sinpleak dira: behe-mailako lengoaia bat osatzen dute. Eskuarki, erabiltzaile arruntak ez ditu programak idazten agindu horiek erabiliz; oso agindu sinpleak direnez, programa oso luze eta bihurriak egin beharko lituzke aplikazio konplexuak sortu ahal izateko. Hori dela eta, maila altuagoko lengoaiak erabiltzen dira, hala nola C, ADA, JAVA, FORTRAN... Baina konputagailuek ez dituzte "ulertzen" lengoaia horien aginduak, eta, beraz, exekutatu baino lehen, programak itzuli —konpilatu— behar dira konputagailuaren makina-lengoaiara.

Hala ere, sistema-programatzaileek konputagailuaren makina-lengoaia ere erabiltzen dute hainbat aplikazio berezitarako (sistemaren ezaugarriak ahalik eta hoberen aprobetxatzeko, sarrera/irteerako funtzio bereziak egiteko, eta abar). Zehatza izatera, konputagailuaren **mihiztadura-lengoaia** (*assembly language*) erabiltzen da; izan ere, konputagailuak, sistema digital guztiak bezala, 1ak eta 0ak baino ez ditu ulertzen, eta makina-mailako aginduak, beraz, 1ekoen eta 0koen segidak baino ez dira. Agindu horiekin lan egiteko, (ingelesezko) mnemotekniko batzuk erabiltzen dira aginduak eta haien eragigaiak zehazteko; adibidez, `add r2, r4, r5` edo `ld r1, A[r2]`. Mnemotekniko horiek konputagailuaren mihiztadura-lengoaia osatzen dute; programak memorian kargatu baino lehen, bitarrera itzuli beharko dira mihiztatzailea deitzen den programa berezi baten bidez.

Laburbilduz: konputagailu bat diseinatzeko, haren makina-lengoaia definitu behar da, eta lengoaia horren aginduak exekutatzeko behar den hardwarea modu egokian antolatu eta kontrolatu behar da.

7.2.2.1. Aginduen formatua

Makina-lengoaiako agindu batek, oro har, eragiketa sinple bat eta haren eragigaiak definitzen ditu. Informazio hori guztia emateko, **formatu** jakin bat erabiltzen da. Agindu-formatu asko dago eta ezin ditugu liburu honetan azaldu; nahi izanez gero, [ML] liburura jo dezake irakurleak aginduen formatuak aztertzeko.

Diseinatu nahi dugun prozesadorerako, luzera finkoko aginduak erabiliko ditugu: agindu guztiak bit kopuru berekoak dira. Bit horiek, eskuarki, honela antolatzen dira:

Eragiketa-kodeak definitzen du aginduak bete behar duen eragiketa: `add` (batuketa), `sub` (kenketa), `mov` (datu bat kopiatu), `ld` (memoria irakurri), `st` (memorian idatzi)... Aukera bat baino gehiago badago ere, agindu-multzoak k agindu baditu, eragiketa-kodea $\log_2 k$ bitekoa izango da. Adibidez, prozesadore batek 42 agindu baditu, 6 biteko eragiketa-kodeak beharko ditugu agindu horiek kodetzeko ($\log_2 42 = 5{,}39 \rightarrow 6$).

Eragiketa-kodeaz gain, **eragigaiak** adierazi behar dira, eta, horretarako, hainbat aukera daude; aukera bakoitza eragigaien **helbideratze-modu** bat da. Helbideratze-modu batek hau adierazten du: non dagoen eragigaia eta nola eskuratu behar den. Esan dugun bezala, modu asko dago eragigaiak adierazteko, eta hauek dira erabilienak:

- **Berehalakoa:** eragigaia aginduan bertan dago. Adibidez,

  ```
  movi r2, #8        ; r2 := 8
  ```

  Kasu horretan, eragigaia, 8 zenbakia, aginduan bertan dago eta `r2` erregistroan kargatu behar da.

- **Erregistro bidezko zuzenekoa:** eragigaia erregistro batean dago; aginduan, beraz, eragigaia duen erregistroa adierazten da. Adibidez,

  ```
  add r2, r3, r4     ; r2 := r3 + r4
  ```

  `r3` eta `r4` erregistroen edukiak batu behar dira, eta emaitza `r2` erregistroan utzi behar da. Esaterako, `r3` erregistroaren edukia 5 bada eta `r4` erregistroarena –7, `r2` erregistroan 5 + (–7) = –2 kargatuko da.

- **Absolutua:** eragigaia memorian dago, eta aginduan haren helbidea adierazten da (eskuarki aldagai baten izenaren bidez). Esaterako,

  ```
  ld r2, ALD         ; r2 := MEM[ALD]
  ```

  `ALD` helbideko memoria-posizioa irakurri behar da, eta edukia `r2` erregistroan kargatu. `ALD` = 100 bada eta memoriako 100 helbidearen edukia 20 bada, agindua exekutatu ondoren, `r2` erregistroaren edukia 20 izango da.

- **Indexatua:** eragigaia memorian dago, eta haren helbidea eskuratzeko, aginduan adierazten diren oinarri-helbidea eta indize-erregistroa batu behar dira. Esaterako,

  ```
  stx r2, ALD[r1]   ; MEM[ALD+r1] := r2
  ```

  `ALD` aldagaiaren helbideari `r1` erregistroaren edukia batu ondoren lortzen den helbideko memoria-posizioan, `r2` erregistroaren edukia idatzi behar da.

Helbideratze-modu erabilienak dira aurrekoak, baina gehiago erabiltzen dira; esaterako, erregistro bidezko zeharkakoa (eragigaiaren memoria-helbidea erregistro batean dago), memoria bidezko zeharkakoa, erlatiboa, eta abar.

Laburbilduz. Aginduak adierazteko, formatu jakin bat erabiltzen da. Zati batek, eragiketa-kodeak, oinarrizko eragiketa zehazten du; gainerakoek, eragigaiak non dauden adierazten dute, helbideratze-moduren bat erabiliz.

7.2.3. Aginduak exekutatzeko algoritmoa

Gainerako sistema digitaletan bezala, prozesadorearen kontrol-unitatearen eginkizuna zehatza da oso: sistemaren osagai guztien funtzionamendua kontrolatzea, dagozkien kontrol-seinaleen bidez, funtzio jakin bat egin dezaten. Ezagutzen dugunez, konputagailuen eginkizuna beti bera da: programetako aginduak exekutatzea. Aginduak exekutatzeko prozedura honako urrats edo fase hauetan bana daiteke:

1 Aginduaren **bilaketa** (*fetch*):

Programetako aginduak memorian daude. Beraz, exekutatu ahal izateko, memoriatik ekarri behar dira prozesadorera. Fase horri bilaketa deritzo.

Aipatu dugun bezala, PC erregistroak dauka exekutatu behar den aginduaren memoria-helbidea. Beraz, PCan dagoen helbidea erabili behar da agindua irakurtzeko. Irakurri eta gero, agindua IR erregistroan, agindu-erregistroan, utzi behar da.

Beraz, honako hau egin behar da:

```
IR := MEM[PC]   ; kargatu IR erregistroan PCak adierazten duen
                  memoria-posizioaren edukia
```

2 Aginduaren **deskodeketa** (*decode*):

Agindua IR erregistroan kargatu eta gero, eragiketa-kodea analizatu behar da, agindua zein den jakiteko. Horretarako, deskodegailu bat erabil daiteke; deskodegailuaren irteerek esango digute zein den agindua.

Lehendabiziko bi fase horiek berdinak dira agindu guztietarako.

3 Eragigaien irakurketa:

Aginduak behar dituen eragigaiak eskuratu behar dira (beharrezkoa denean). Oro har, erregistro-multzoko erregistroak irakurri behar dira eta edukia "laneko" erregistroetan utzi. Eragigaiak aginduan bertan badatoz, orduan ez da ezer egin behar, aginduaren bilaketarekin batera eskuratu direlako.

4 Aginduaren **exekuzioa:**

Eragigaiak lortu eta gero, eragiketari ekin behar zaio, eskuarki unitate aritmetiko/logikoa (batuketa bat egiteko, esaterako), memoria (datu bat eskuratzeko, adibidez) edo beste gailuren bat erabiliz.

5 Emaitzaren idazketa:

Bukaeran, eragiketaren emaitza nonbait gorde behar da (aginduaren arabera). Esaterako, erregistro-multzoko erregistro jakin batean.

Azken hiru faseak ez dira derrigorrezkoak agindu guztietarako, agindu bakoitzak behar desberdinak ditu eta.

Aurreko faseak behin eta berriz errepikatu behar dira, programa bateko aginduak banan-banan exekutatzeko. Tartean, beraz, PC erregistroaren edukia eguneratu behar da. Salbuespenak salbuespen, programen exekuzioa sekuentziala da, hau da, *i* aginduaren ondoren *i*+1 agindua exekutatzen da; hori egiteko, `PC := PC + 1` egin behar da hurrengo aginduaren bilaketari ekin baino lehen. PC erregistroaren gaurkotzea hainbat unetan egin daiteke (esaterako, agindu bakoitzaren exekuzioaren bukaeran), baina eskuarki aginduaren bilaketaren bukaeran egiten da.

Laburbilduz. Konputagailu baten kontrol-unitatearen eginkizuna da <bilaketa – deskodeketa – eragigaien irakurketa – exekuzioa – emaitzen idazketa / bilaketa – deskodeketa – ... > faseak behin eta berriz errepikatzea.

7.3. BIRD PROZESADOREAREN DISEINUA

Aurreko atalean laburbildu ditugu konputagailu baten hardwarearen eta aginduen exekuzioaren oinarrizko kontzeptuak. Hemendik aurrera, prozesadore didaktiko baten diseinuari ekingo diogu. Nahitaez, sinplifikazio asko egin beharko dugu, gure helburua sistema digitalen diseinuaren oinarrizko kontzeptuak lantzea baita.

Adibide gisa, BIRD izeneko prozesadorea diseinatuko dugu. Prozesadore horren makina-lengoaia eta arkitektura "*Makina-hizkuntza. Oinarrizko konputagailu baten egitura, agindu-multzoa eta programazioa"* izeneko liburuan ageri dira [ML]. Prozesadorea sinplea bada ere, behe-mailako programazioaren kontzeptuak azaltzeko erabiltzen baita, ezin ditugu xehetasun guztiak azaldu liburu honetan, horretarako agindu-multzo osoa analizatu beharko baikenuke. Hori dela eta, BIRD prozesadorearen azpimultzo bat diseinatuko dugu, agindu batzuk kontuan hartuz.

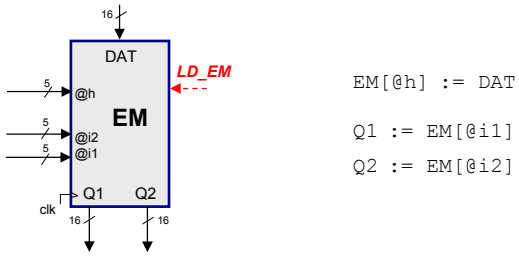
7.3.1. Hardwarearen ezaugarri nagusiak

Prozesadore bat diseinatu behar denean, hainbat erabaki hartu behar dira sistemari buruz, erabili nahi diren osagaiei buruz zein exekutatu behar diren makina-aginduei buruz.

Antzeko erabakiak hartu behar izan dira BIRD prozesadore didaktikoa garatzeko. Hala, BIRD **prozesadorea 16 bitekoa da;** hau da, agindu batean prozesatzen diren datuak 16 bitekoak dira (gaurko prozesadoreek, esaterako, 64 biteko hitzak prozesatzen dituzte). Hori dela eta, prozesu-unitateko erregistroak zein memoria-posizioak 16 bitekoak dira.

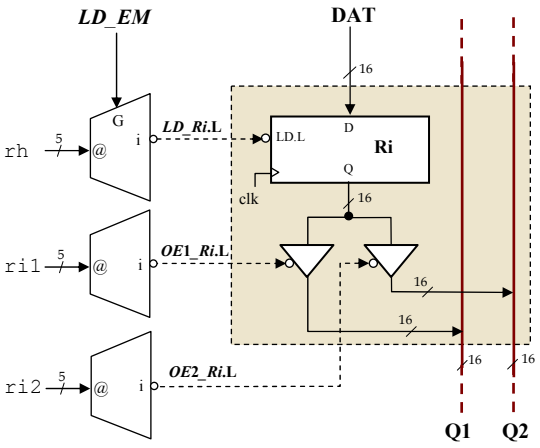
PC erregistroa ere **16 bitekoa** da. Memoria helbideratzeko erabiltzen denez, 2^{16} = 64 k hitz izango ditugu memorian. **Memoria,** beraz, 64 k × 16 bitekoa da: **helbideak zein datuak 16 bitekoak dira.** Memoriako datu-sarrera eta datu-irteera banatuta daude, eta ohiko kontrol-seinaleak ditu: *MCS* (*chip select*), *MOE* (*output enable, read*) eta *MWR* (*write*).

Erregistro-multzoa osatzeko, **16 biteko 32 erregistro** erabili dira. Hori dela eta, **erregistroen helbideak 5 bitekoak** dira (2^5 = 32). Hiru eragiketa batera onartzen du erregistro-multzoak: bi erregistroen irakurketa eta erregistro baten idazketa. Beraz, hiru helbide-sarrera ditu (@i1 eta @i2 irakurketarako, eta @h idazketarako), eta, datuetarako, bi irteera-bus (Q1 eta Q2) eta sarrera-bus bat (DAT), 7.3. irudian ageri den moduan.



7.3. irudia. BIRD prozesadorearen erregistro-multzoaren eskema.

5.2. atalean analizatu dugun erregistro-multzoaren antzekoa bada ere, BIRD makinan erabili nahi dugun erregistro-multzoak baditu desberdintasun aipagarri batzuk. Hasteko, 3 erregistrorekin batera lan egiteko, hiru deskodegailu behar ditu. Idazketa-helbidea deskodetzen duenak erregistroen LD seinaleak kontrolatuko ditu: *LD_EM* seinalea aktibatzen denean, LD bat aktibatuta eta gainerakoak desaktibatuta utziko ditu. Bestetik, erregistro-multzoak bi irteera-bus dituenez (Q1 eta Q2), erregistro bakoitzaren irteera bi busetara eraman behar da, konexioak hiru egoerako bi bufferren bidez eginez. Hala, irakurketa-helbideak deskodetzen dituzten bi deskodegailuek hiru egoerako bufferren *OE* seinaleak kontrolatuko dituzte, bus bakoitzera erregistro bakar bat konektatzeko. 7.4. irudian, erregistro-multzoko erregistro baten eskema logikoa ageri da.



7.4. irudia. BIRD prozesadorearen erregistro-multzoaren barne-egitura (erregistro bakar bat). `rh` helburu-erregistroaren helbidea da (idatzi behar den erregistroarena), eta `ri1` eta `ri2` iturburu-erregistroenak (irakurri behar direnak). Erregistro guztiak Q1 eta Q2 irteera-busetara konektatzen dira, hiru egoerako bina bufferren bidez.

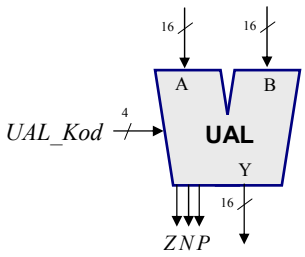
Horrelako erregistro-multzoak ohikoak dira konputagailuetan; bi irteera-bus izanik, hainbat agindutan behar diren bi eragigaiak (esaterako, batuketa bat egiteko) aldi berean eskura daitezke, eta, hala, azkarragoa izango da agindu horien exekuzioa.

Bestalde, prozesadoreetan erabiltzen diren erregistro-multzoetan, `r0` erregistroa berezia izan ohi da; izan ere, **`r0` erregistroaren edukia beti 0 da** (konstante bat da, ezin da aldatu). Erabaki bera hartuko dugu BIRD makinaren erregistro-multzorako; beraz, `r0` erregistroa ezin da idatzi (irakurri bai, jakina) eta haren edukia 0 da.

BIRD makinaren **unitate aritmetiko/logikoa**k 16 biteko bi eragigai hartu eta 16 biteko emaitza sortzen du. Unitate aritmetiko/logikoa hainbat eragiketatarako erabiltzen da konputagailuetan, eta eragiketa jakin bat aukeratzeko, kode bat eman behar zaio (ikus 3.7. atala). BIRD makinan, 4 biteko kode bat behar da, honako eragiketa hauek egin ahal izateko[25]:

Y	*UAL_Kod*	*Y*	*UAL_Kod*	*Y*	*UAL_Kod*
$A + B$	0000	A	0100	*A or B*	1000
$A - B$	0001	B	0101	*A and B*	1001
$A \times B$	0010	$A + 1$	0110	*A xor B*	1010
A / B	0011	$A - 1$	0111	Desp_ezk *A*	1011
				Desp_esk *A*	1100

16 biteko emaitzaz gain, hiru adierazle sortzen ditu: ***Z*** ($Y = 0$); ***N*** ($Y < 0$) eta ***P*** ($Y > 0$). Diseinatzen ari garen adibiderako, *Z* adierazlea baino ez dugu erabiliko: $Z = 1$ izango da emaitza 0 denean.



7.5. irudia. BIRD prozesadorearen unitate aritmetiko/logikoa.

[25] Behar izanez gero eragiketa gehiago defini zitezkeen kodearen bit kopurua aldatu gabe (eragiketen kodeketa arbitrarioa da).

7.3.2. Aginduak eta haien formatua

BIRD prozesadorearen agindu-multzoak 42 agindu ditu; gure helburuetarako asko direnez, horietatik batzuk aukeratu ditugu. Aginduok edozein agindu-multzotan ohikoak diren agindu motak islatzen dituzte: agindu aritmetiko eta logikoak, datuen mugimendua memoriaren eta erregistroen artean, eta programaren fluxuaren kontrola (jauziak).

Hain zuzen ere, hauek dira diseinu-ariketa honetarako aukeratu ditugun aginduak:

1. **Agindu aritmetikoak** eta **logikoak**

   - **`op rh, ri1, ri2`** `; rh := ri1 op ri2`
     (`op` = `add`, `sub`, `mul`, `div`, `or`, `and`, `xor` → 7 agindu guztira)

     Ohiko agindu aritmetikoak eta logikoak edozein prozesadoretan: bi erregistroren edukia (`ri1` eta `ri2`, iturburu-erregistroak) prozesatu eta emaitza erregistro batean utzi (`rh`, helburu-erregistroan).

   - **`mov rh, ri1`** `; rh := ri1`

     Datuak erregistro batetik beste batera eramateko. Eskuarki, UALa erabiltzen da eragiketa hori egiteko ($Y = A$ edo $Y = B$ eragiketa, hau da, pasatu irteerara sarrerako datua); datua aldatzen ez bada ere, UALaren adierazleak aktibatzen dira eta, zenbait kasutan, informazio hori ere erabili behar da.

   - **`opi rh, ri1, #bereh`** `; rh := ri1 op bereh`
     (`opi` = `addi`, `subi`, `muli`, `divi`, `ori`, `andi`, `xori` → 7 agindu)
   - **`movi rh, #bereh`** `; rh := bereh`

     Aurrekoen antzeko aginduak, baina bigarren eragigaia konstante bat da eta aginduan bertan adierazten da (bigarren eragigaiaren helbideratze-modua berehalakoa da).

Guztira, beraz, 16 agindu: 7 `op`, 7 `opi`, `mov` eta `movi`; desberdinak izanagatik, modu berean kontrolatzen dira, gero ikusiko dugun moduan.

2. **Memoria**rekin lan egiten duten aginduak

   Bi eragiketa egin daitezke memoriarekin: irakurketa eta idazketa; hainbat agindu daude eragiketa horiek egiteko, eta haien arteko desberdintasuna hau da: helbideratze-modua. Horietako bi kasu hartuko ditugu kontuan:

   - **`ld rh, ALD`** `; rh := MEM[ALD]`
   - **`st ri2, ALD`** `; MEM[ALD] := ri2`

     `ld` (*load*) aginduarekin, memoria-posizio bat irakurri eta irakurritako datua `rh` helburu-erregistroan kargatzen da; `st` (*store*) aginduarekin kontrakoa egiten da, hau da, `ri2` iturburu-erregistroaren edukia memorian idatzi.

     `ALD` izenak, beraz, aldagai baten memoria-helbidea adierazten du. Bi aginduetan, helbideratze absolutua erabiltzen da, hau da, irakurri edo idatzi behar den memoria-posizioaren helbidea aginduan bertan adierazten da.

   - **`ldx rh, ALD[ri1]`** `; rh := MEM[ALD+ri1]`
   - **`stx ri2, ALD[ri1]`** `; MEM[ALD+ri1] := ri2`

     Bi memoria-eragiketak, irakurri eta idatzi, baina helbideratzea indexatua izanik: eragiketa egin baino lehen, helbidea kalkulatu behar da, `ALD` oinarri-helbideari `ri1` erregistroaren edukia batuz.

3. **Agindu-fluxua**ren kontrola (jauziak, *branch*)

   - **`beq ri1, etiketa`** `; if (ri1=0) pc := pc + desp`

     Programa baten exekuzioa eskuarki sekuentziala da, hau da, *i* aginduaren ondoren *i*+1 agindua exekutatzen da; hala ere, hainbat kasutan, fluxu hori hautsi egin behar da, eta jauzi bat egin ondoz ondokoa ez den beste agindu batera: $i \rightarrow j$. Hori egin ahal izateko, hainbat jauzi-agindu daude prozesadoreetan. Horietako bat aukeratu dugu adibide honetarako: jauzi baldin 0 (*branch equal)*. **Baldintzapeko jauzi** bat da: jauzi egin behar da aipatzen den etiketa daraman agindura baldin eta `ri1` erregistroaren edukia 0 bada; bestela, ondoz ondoko agindua exekutatuko da[26].

[26] Baldintza gabeko jauzia egiteko, nahikoa da `beq r0,etiketa` agindua exekutatzea; `r0` erregistroaren edukia 0 denez, jauzia beti egingo da.

Jauziak egiteko, PC erregistroaren edukia aldatu behar da; gogoratu: PC erregistroak adierazten du exekutatu behar den agindua. Aginduen sekuentzian egiten den jauzia uneko aginduaren PCarekiko desplazamendu batez adierazten da. Desplazamendua aurrerantz (positiboa) edo atzerantz (negatiboa) izan daiteke (izan ere, 2rako osagarrian adierazten da). Gero ikusiko dugu adibide bat.

Mihiztadura-lengoaiaz idatzitako programak ulergarriagoak izateko, jauzi-aginduan desplazamendua adierazi beharrean, "etiketa" bat adierazten da, jauzia zein agindutara egin behar den adierazteko; hala, jauziaren helburuan dagoen aginduak etiketa hori eraman beharko du. Programa bitarrera itzultzean, dagokion desplazamenduaz ordezkatuko da etiketa.

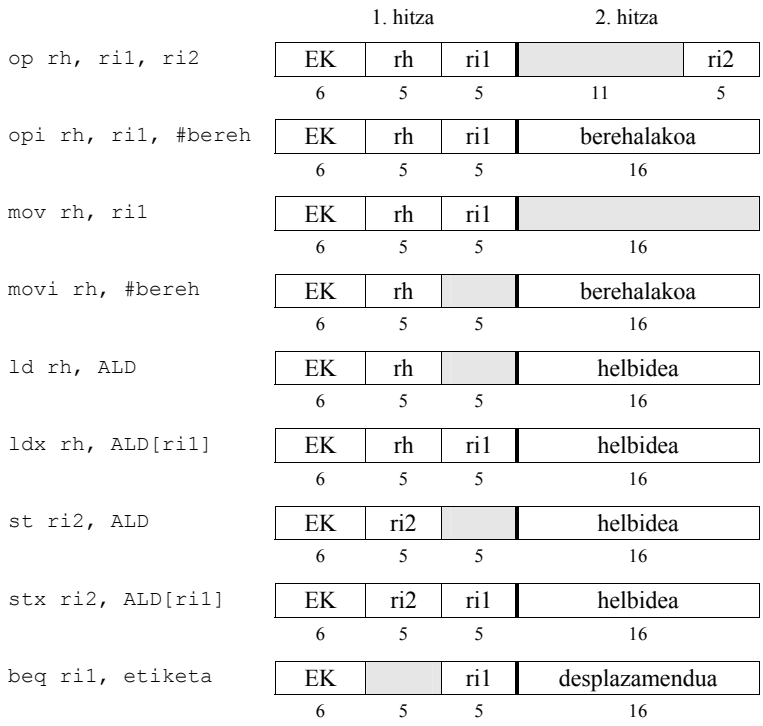
Aginduak definituta, haien formatua zehaztu behar da, eta, horretarako, eragiketa-koderako zein eragigaietarako behar diren bit kopuruak kalkulatu behar ditugu.

Adibide honetan 21 agindu baino ez ditugu erabiliko; beraz, 5 bit nahikoak lirateke eragiketa-kodea adierazteko. Hala ere, jatorrizko BIRD konputagailuan 6 bit behar direnez (42 agindu dituelako), kopuru hori mantenduko dugu, aginduen formatua ez aldatzeko. Beraz, **eragiketa-kodea 6 bitekoa** izango da.

Eragigaiak adierazteko behar den bit kopurua desberdina da agindu bakoitzean. Lehen esan dugun moduan, erregistroak adierazteko 5 bit behar dira (32 erregistro daude) eta memoria-helbideetarako, 16 bit. Era berean, berehalako datuak eta jauzien desplazamenduak adierazteko, 16 bit erabiltzen dira BIRD prozesadorean (prozesadorea diseinatzean hartu behar den erabakietako bat). Beraz, `op` bezalako aginduetan, 15 bit behar dira eragigaiak adierazteko (3 erregistro); `mov` aginduan, 10 bit (2 erregistro); `opi` motako aginduetan, 26 bit (2 erregistro eta berehalako datu bat); `movi` aginduaren kasuan, 21 bit (erregistro bat eta berehalako datu bat); `ld` eta `st` aginduetan, 21 bit (erregistro bat eta memoria-helbide bat); `ldx` eta `stx` aginduetan, 26 bit (bi erregistro eta helbide bat); eta, azkenik, `beq` aginduan, 21 bit (erregistro bat eta desplazamendua).

Kasurik luzeenean, `opi` esaterako, 32 bit behar ditugu agindua kodetzeko: 6 bit eragiketa-koderako eta 26 bit eragigaietarako; hortaz, 16 biteko 2 hitz erabili behar ditugu. Memoriarekin arazoak saihesteko, komeni da agindu guztien luzera berdina izatea eta hitzaren tamainaren multiploa; hori dela eta, BIRD prozesadoreko **aginduak 32 bitekoak dira,** hau da, bi posizio

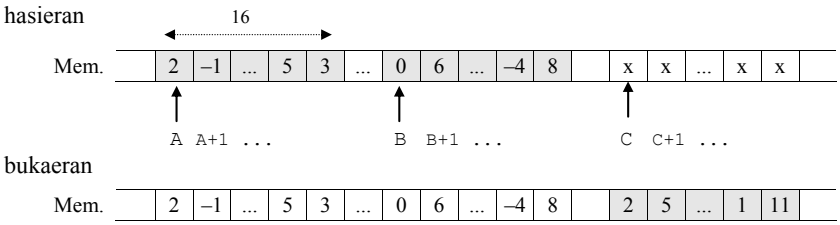
okupatuko dituzte memorian (zenbait kasutan, bit batzuk erabili gabe uzten badira ere). Hona hemen aginduen formatu zehatza.



7.6. irudia. BIRD prozesadorearen aginduen formatuak. Grisez, erabiltzen ez diren bitak. Garrantzirik ez badu ere, hauek dira eragiketa-kodeak BIRD konputagailuan: `ld`(0), `ldx`(2), `st`(3), `stx`(5), `mov`(7), `movi`(8), `add`(9), `addi`(10), `sub`(11), `subi`(12), `mul`(13), `muli`(14), `div`(15), `divi`(16), `and`(19), `andi`(20), `or`(21), `ori`(22), `xor`(23), `xori`(24), `beq`(26).

Agindu guztietan, lehen hitzaren lehen 6 bitek eragiketa-kodea (EK) adierazten dute, eta azken 5 bitek, erabiltzen badira, `ri1` iturburu-erregistroa; helburu-erregistroa erabili behar bada, tarteko 5 bitetan adierazten da. Era berean, 16 biteko datuak (helbide bat, konstante bat edo desplazamendu bat) bigarren hitzean datoz. Arazo bat dago, hala ere, `ri2` iturburu-erregistroarekin; `op` motako aginduetan, bigarren hitzean adierazten da, azken 5 bitetan; `st/stx` aginduetan, ordea, lehen hitzean adierazten da, beste aginduetan `rh` adierazteko erabiltzen den eremuan. Beraz, adi egon beharko dugu `ri2` eragigaiaren helbidea erabiltzean, ez baitator beti aginduen eremu berean.

Nahiz eta liburu honen helburua makina-mailako programazioa azaltzea ez den, ikus dezagun adibide sinple bat aurreko aginduen esanahia argitzearren. `A` eta `B` bektoreak batu behar dira eta emaitza `C` bektorean utzi. Bektoreek 16 osagai dituzte.



Honela programa daiteke eragiketa hori BIRD makinarako:

```
        ...
        movi  r1, #0          ; indize-erregistroa hasieratu
        movi  r2, #16         ; bektoreen osagai kopurua
segi:   ldx   r3, A[r1]       ; irakurri A bektorearen osagai bat
        ldx   r4, B[r1]       ; irakurri B bektorearen osagai bat
        add   r5, r4, r3      ; batu bi osagaiak (emaitza, r5-ean)
        stx   r5, C[r1]       ; gorde emaitza C bektoreko osagaian
        addi  r1, r1, #1      ; inkrementatu r1, bektoreen hurrengo osagaiaren indizea
        subi  r2, r2, #1      ; osagai bat gutxiago geratzen da
        beq   r2, buka        ; r2 = 0 bada, eragiketa bukatu da (jauzi buka-ra)
        beq   r0, segi        ; bestela, segi eragiketarekin (r0 = 0 → jauzi segi-ra)
buka:   ...
```

Begizta bat antolatu dugu eragiketa egiteko. Begiztaren gorputzean, `A` eta `B` bektoreen osagai bana irakurri, batu eta emaitza `C` bektorean gordetzen da. Helbideratze indexatua erabili dugu bektoreen osagaiak irakurtzeko eta idazteko: `A + r1`; hala, nahikoa da `r1` inkrementatzea bektoreen hurrengo osagaia eskuratu ahal izateko. `A`, `B` eta `C` izenek bektoreen lehendabiziko osagaien memoria-helbidea adierazten dute (programaren beste parte batean adierazi behar izan ditugu helbide horiek).

Begizta `r2` erregistroaren bidez kontrolatzen da. Hasi baino lehen, bektoreen luzera kargatu dugu hor (16 osagai), eta, osagai bakoitza prozesatu ondoren, 1 kendu diogu, hau da, `r2` erregistroa kontagailu gisa erabiltzen dugu. Begiztaren bukaeran, `r2` erregistroa analizatzen da. Edukia 0 bada, jauzi bat egiten da begiztatik ateratzeko, eragiketa bukatu baita; bestela,

`beq r0, segi` agindua exekutatuko da. `r0` = 0 denez, jauzia bete egingo da, `segi` etiketa duen agindua exekutatzeko, hau da, bektoreen beste osagai bat prozesatzeko.

Aginduak bi hitzekoak direnez, programa zati horrek 20 hitz betetzen ditu memorian (ikus 7.8. ariketa); programa exekutatu aurretik, lehen aginduaren memoria-helbidea kargatu behar da PC erregistroan; hortik aurrera, lehen azaldu dugun algoritmoari jarraituko zaio behin eta berriz: agindua bilatu (memoriako bi hitz irakurri, PC eta PC + 1 helbideetan), eragiketa-kodea deskodetu, eragigaiak eskuratu, eragiketa exekutatu eta emaitzak gorde.

7.3.3. BIRD prozesadorearen prozesu-unitatea

Aurreko ataletan definitu ditugu prozesu-unitatearen oinarrizko osagaiak eta exekutatu behar diren aginduak. Beraz, behar dugun informazio guztia daukagu jadanik sistema osoa diseinatzeko. Bi ataletan banatuko dugu prozesadorea: kontrol-unitatea eta prozesu-unitatea. Hala, lehendabiziko urratsa prozesu-unitateko osagaiak antolatzea da, egin beharrekoa egin ahal izateko, eta behar dituzten kontrol-seinaleak zehazteko; hori egin eta gero, kontrol-unitateko kontrol-algoritmoa garatuko dugu, ASM grafo baten bidez, kontrol-seinaleak noiz sortu behar diren zikloz ziklo adierazteko.

Lehen aipatu dugun moduan, prozesu-unitateko osagai nagusiak lau hauek dira: PC eta IR erregistro bereziak; helburu orokorreko erregistro-multzoa, EM; eta unitate aritmetiko/logikoa, UAL. Horrez gain, memoria ere hartu behar dugu kontuan, aginduak eta datuak memorian daudelako. Ikus dezagun nola antolatu osagai horiek definitu ditugun aginduak exekutatu ahal izateko, zehaztu ditugun faseen arabera.

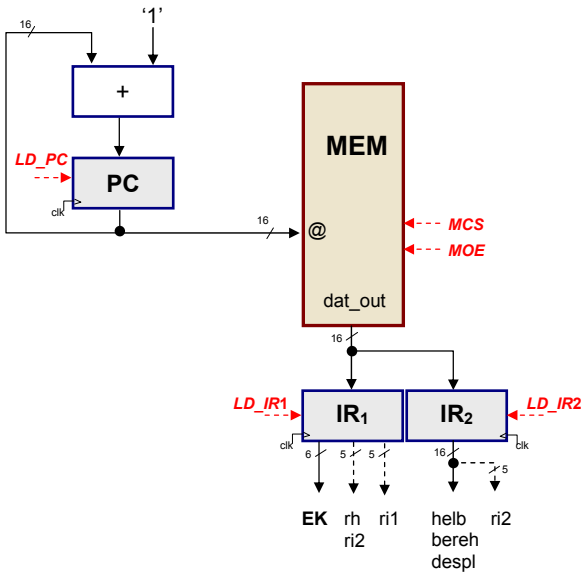
7.3.3.1. Aginduen bilaketa eta deskodeketa

1. Batetik, aginduen bilaketa dugu. Hori egiteko, PC eta IR erregistroak erabili behar ditugu. PC erregistroak exekutatu behar den aginduaren helbidea erakusten du beti; beraz, PC erregistroaren irteera memoriara eraman behar da, helbide-sarrerara. Bestalde, irakurriko den agindua IR erregistroan kargatu behar da; hau da, memoriaren irteera IR erregistrora eraman behar da.

Hala ere, gogoratu: BIRD prozesadoreko aginduek bi hitz okupatzen dituzte memorian, 32 bitekoak direlako. Beraz, bi pausotan (bi erloju-

ziklotan) irakurriko dugu agindu bat: lehen hitza lehenengo zikloan eta bigarren hitza hurrengoan. Hori dela eta, IR erregistroak ere 16 biteko bi zati independente izango ditu, IR_1 eta IR_2, aginduen lehen eta bigarren hitza gordetzeko. Pauso bakoitzean, PC erregistroaren edukia eguneratu behar da, hurrengo helbidea erakusteko. Aukera bat baino gehiago badago ere, batugailu bat erabil daiteke aginduen bilaketan behar den `PC := PC + 1` eragiketa egiteko.

Honela antola daitezke osagai horiek aginduen bilaketa egin ahal izateko:



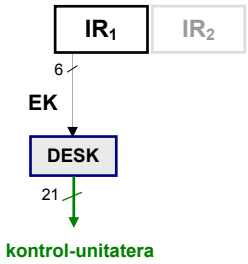
7.7. irudia. Aginduen bilaketarako behar diren osagaiak eta haien arteko konexioak.

Gailu horiek kontrolatzeko, honako kontrol-seinale hauek sortu beharko ditu kontrol-unitateak, bi ziklotan:

(1) MEM, IR, PC → *MCS*, *MOE* (irakurri), *LD_IR*1, *LD_PC*
(2) MEM, IR, PC → *MCS*, *MOE* (irakurri), *LD_IR*2, *LD_PC*

Eragiketa horiek egin eta gero, agindu bat kargatu dugu IR_1-IR_2 erregistroetan, eta PC erregistroak hurrengo aginduaren helbidea du.

2. Agindua bilatu eta IR erregistroan kargatu ondoren, eragiketa-kodea deskodetu behar da. Nahikoa da horretarako deskodegailu bat: 6 biteko eragiketa-kodea hartu eta deskodegailuaren $2^6 = 64$ irteeretako bat aktibatuko da, aginduari dagokiona. BIRD konputagailuan 42 agindu baino ez dago, eta, gure adibidean, 21; beraz, irteera horiek bakarrik eramango ditugu kontrol-unitatera, exekutatu behar duen agindua zein den jakin dezan.



7.8. irudia. Aginduen eragiketa-kodearen deskodeketa.

7.3.3.2. Aginduen exekuzioa

Aginduak deskodetu ondoren, exekutatu egin behar dira. Agindu bakoitzak behar desberdinak ditu; beraz, analiza ditzagun behar horiek aginduz agindu.

- **`op rh, ri1, ri2`** (`add`, `sub`..., `and`...)

  Zazpi agindu hauen eragigaiak erregistro-multzoan daude, eta emaitza ere erregistro-multzora eraman behar da. Eragiketa egiteko, unitate aritmetiko/logikoa erabili behar da.

  Erregistro-multzoaren irteerak zuzenean UALera eraman beharrean, ohikoa da tartean laneko erregistro (*latch*) pare bat kokatzea (R_e1 eta R_e2). Modu berean, UALaren emaitza beste laneko erregistro batean gorde ohi da (R_ual) erregistro-multzoan sartu baino lehen (ikus 7.9. irudia). Lotura horiek eginda lortzen den egiturari **datu-bidea** (*datapath*) deritzo, bertatik pasatzen baitira prozesatzen diren datuak.

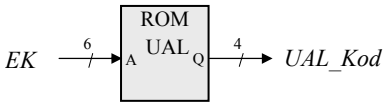
  Hiru laneko erregistro edo *latch* horiek prozesadore sinple honen funtzionamendurako beharrezkoak ez diren arren, arrazoi pedagogiko batengatik gehitu ditugu: horiei esker, aginduen exekuzioaren urratsak

—eragigaien irakurketa, eragiketa bera, eta emaitzaren idazketa— banatu egiten direlako[27].

Agindu horien guztien exekuzioa modu berean egin behar bada ere, UALean behar den kodea desberdina da, eragiketa desberdinak egin behar direlako (batuketa, biderketa, *and*...). Kode horiek, gainerako kontrol-seinaleak bezala, kontrol-algoritmoaren bidez sortu behar dira, baina, horrela egiten bada, agindu bakoitzak egoera desberdinak behar ditu kontrol-algoritmoan. Agindu desberdin asko badaude, kontrol-automataren egoera kopurua oso handia izan daiteke.

Badago, ordea, beste aukera bat: irakurtzea taula batean (ROM batean) UALak behar dituen kodeak, aginduaren eragiketa-kodea helbide gisa erabiliz (ikus, adibidez, 5.3. ariketa). Hala, kontrol-algoritmoa ez da arduratuko seinale horiez, eta bildu ahal izango dugu 7 aginduen exekuzioa bide bakar batean.

Soluzio hori agindu guztietara zabalduko dugu, eta, hortaz, agindu bakoitzak behar duen UALeko eragiketaren kodea idatzita izango dugu ROM taulan (ikus 7.9. irudia).



(A = *address*, helbidea. ROM taularen *OE* seinalea beti 1 izango da; nahi izanez gero, seinale hori ere kontrola daiteke kontrol-algoritmoan.)

- **`mov rh, ri1`**

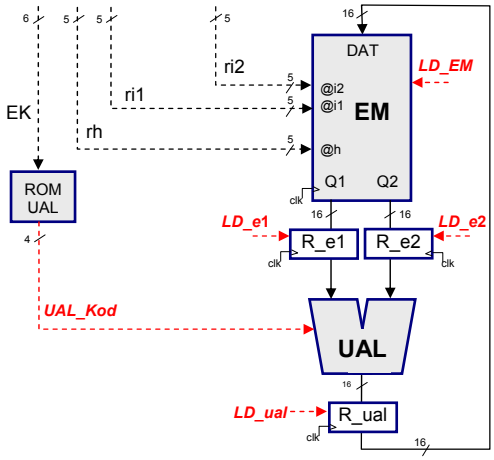
  `op` motako aginduen antzekoa da, nahiz eta eragigai bakar bat erabiltzen duen. Hainbat aukera dago agindua exekutatzeko, baina erabiliena hauxe da: irakurri `ri1` eta kargatu laneko erregistro batean (R_e1, esaterako); exekutatu UALean $Y = A$ eragiketa, hau da, igaro ezer egin gabe eta kargatu R_ual erregistroa; eta, azkenik, eraman handik helburu-erregistrora (`rh`).

  Beraz, `op` aginduak bezala exekuta daiteke; izan ere, berdin zaigu zer egiten den bigarren eragigaiarekin, ez baitugu ezertarako erabiliko.

---

[27] Halaber, liburu honen helburuetatik at geratzen den arren, hiru erregistro horiek ahalbidetzen dute datu-bidearen **segmentazioa,** prozesadoreen abiadura handiagotzeko erabiltzen den ohiko teknika.

7.9. irudian ageri da agindu horien guztien exekuzioaz arduratzen den prozesu-unitatearen zatia.



7.9. irudia. `op` eta `mov` aginduen exekuziorako erabili behar diren osagaiak.

Beraz, honela exekutatuko dira `op` motako aginduak:

(a) bi eragigaiak irakurri eta laneko erregistroetan utzi

```
R_e1 := EM[ri1];      R_e2 := EM[ri2];
```

(b) eragiketa egin eta emaitza laneko erregistroan utzi

```
R_ual := R_e1 op R_e2;      (dagokion eragiketa)
```

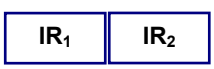
(c) emaitza EMan idatzi

```
EM[rh] := R_ual;
```

Gogoratu: agindua `mov` bada, prozesua bera da, baina eragigai bakar bat erabiliz (eta berdin zaigu zer kargatuko den beste laneko erregistroan, ez baita prozesatuko UALean).

Eragiketa horietarako, hiru erloju-ziklo behar dira eta honako kontrol-seinale hauek sortu beharko ditu kontrol-unitateak:

(1) R_e1, R_e2 → *LD_e1*, *LD_e2*
(2) UAL, R_ual → *(UAL_Kod*, ROM taulan irakurrita), *LD_ual*
(3) EM → *LD_EM*

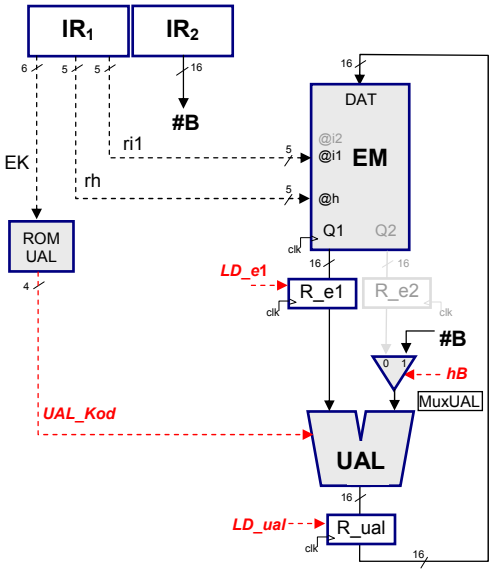


- **`opi rh, ri1, #bereh`**
- **`movi rh, #bereh`**

Aurreko aginduen antzeko funtzioa betetzen dute beste hauek; eragigai bat, ordea, ez dator EMtik, IR_2 erregistrotik baizik (aginduaren bigarren hitza). Beraz, multiplexore bat jarri beharko dugu UALaren B sarreran, R_e2 edo IR_2 aukeratu ahal izateko: MuxUAL. Gainerakoa berdina da.

`movi` aginduaren kasuan, gainera, ez da `ri1` eragigaia erabili behar; nahikoa da $Y = B$ eragiketa exekutatzea UALean; hala ere, modu berean tratatuko ditugu agindu horiek guztiak, berdin zaigulako zer gertatzen den R_e1 laneko erregistroan.

7.10. irudian ageri dira berehalako motako datuak erabiltzen dituzten agindu aritmetiko/logikoak exekutatzeko behar diren osagaiak.



7.10. irudia. `opi`, `movi` aginduen exekuziorako erabili behar diren osagaiak. (#B = berehalako datua.)

Honela exekutatuko dira `opi` motako aginduak:

(a) eragigaia irakurri eta laneko erregistroan utzi

```
R_e1 := EM[ri1];
```

(b) eragiketa egin eta emaitza laneko erregistroan utzi

```
R_ual := R_e1 op #B;
```
(aginduari dagokion eragiketa)

(c) emaitza EMan idatzi

```
EM[rh] := R_ual;
```

Eragiketa horietarako, hiru erloju-ziklo behar dira eta honako kontrol-seinale hauek sortu beharko ditu kontrol-unitateak:

(1) R_e1 → *LD_e*1
(2) MuxUAL, UAL, R_ual → *hB*, (*UAL_Kod*, ROM taulatik), *LD_ual*
(3) EM → *LD_EM*

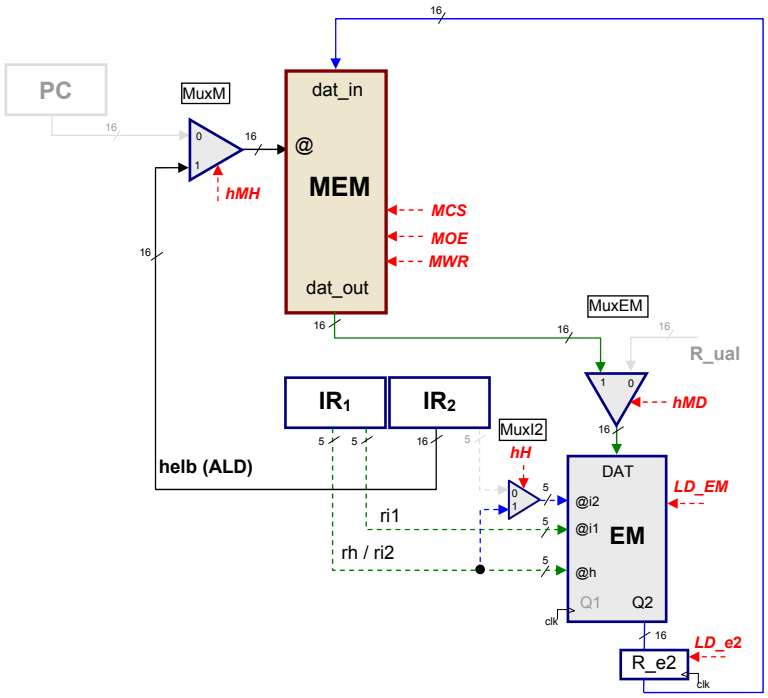
- **`ld rh, ALD`**
- **`st ri2, ALD`**

Bi aginduek memoria erabiltzen dute. Lehenengoak memoriako hitz bat irakurtzen du eta erregistro batean kargatzen du. Bigarrenak, erregistro bat irakurri eta haren edukia memorian idazten du. Memoria-posizioaren helbidea aginduan bertan adierazita dago (helbideratze absolutua); beraz, IR erregistroan dago, kasu honetan IR_2 zatian (ikus aginduen formatua 7.6. irudian).

Orain arte, PC erregistroaren edukia baino ez dugu erabili memoria helbideratzeko, aginduen bilaketa fasean. Oraingo honetan, datuen helbideak memoriara bideratu ahal izateko, multiplexore bat gehitu beharko dugu memoriaren helbide-sarreran, MuxM, PCaren eta IR_2-ren artean aukeratzeko (ikus 7.11. irudia).

`ld` aginduaren kasuan, memoriaren irteera erregistro-multzoaren datu-sarrerara eraman behar da. Ondorioz, hor ere beste multiplexore bat erabili beharko dugu, MuxEM, sarrera hori erabili dugulako dagoeneko `op` motako aginduen emaitza (R_ual) gordetzeko.

`st` aginduaren kasuan, memoriaren datu-sarrerara eraman behar da erregistro-multzoko erregistro baten edukia. Eragigaien irakurketa beti modu berean egiteko, lehendabizi erregistroa irakurri eta laneko erregistro batean utziko dugu (R_e2). Beraz, R_e2 erregistroaren irteera memoriako datu-sarrerara eramango dugu. Adi! Lehen aipatu dugun moduan, `st` aginduak behar duen eragigaia (memorian idatzi behar dena) beste aginduetan helburu-erregistroa adierazteko erabiltzen den eremuan dago; beraz, multiplexore bat jarri beharko dugu EMko @i2 sarreran, MuxI2, batzuetan IR_2-tik eta beste batzuetan IR_1-etik hartu ahal izateko erregistroaren helbidearen 5 bitak.



7.11. irudia. `ld` eta `st` aginduen exekuziorako erabili behar diren osagaiak.

Honela exekutatuko dira `ld` eta `st` aginduak:

`ld`

(a) irakurri memoria eta utzi emaitza erregistro batean

```
EM[rh] := MEM[ALD];
```
(helbidea IR_2-tik hartuta)

`st`

(a) lehendabizi, erregistro bat irakurri eta laneko erregistroan utzi

```
R_e2 := EM[ri2];
```
(dagokion helbidea aukeratuta)

(b) eta gero, memorian idatzi laneko erregistroaren edukia

```
MEM[ALD] := R_e2;
```
(helbidea IR_2-tik hartuta)

`ld` agindurako nahikoa da erloju-ziklo bat, baina `st` aginduak bi ziklo behar ditu.

Aginduak exekutatzeko, honako kontrol-seinale hauek sortu beharko ditu kontrol-unitateak:

`ld`

(1) MEM, MuxM → *MCS*, *MOE* eta *hMH*
EM, MuxEM → *LD_EM* eta *hMD*

`st`

(1) R_e2, MuxI2 → *LD_e2* eta *hH*
(2) MEM, MuxM → *MCS*, *MWR* eta *hMH*

- **`ldx rh, ALD(ri1)`**
- **`stx ri2, ALD(ri1)`**

Aurreko bi aginduekin dagoen diferentzia bakarra helbideratze-moduan dago. Izan ere, memoria-eragiketa baino lehen, helbidea kalkulatu behar da: `ri1 + ALD`. Lehenengo eragigaia erregistro-multzoan dago, eta bigarrena IR_2 erregistroan. UALean egingo dugu batuketa eta emaitza R_ual erregistroan utziko dugu. Hortik hartu behar da, beraz, memoria-helbidea, eta, horregatik, hirugarren sarrera bat behar da memoriako helbidea aukeratzen duen multiplexorean (MuxM, ikus 7.12. irudia).

Honako sarrera hauek erabiliko ditugu memoria-helbideak emateko:

hMH = '0' (00) → PC
hMH = '1' (01) → `ALD + ri1`
hMH = '2' (10) → `ALD`

Beraz, honela exekutatuko dira `ldx` eta `stx` aginduak:

`ldx`

(a) irakurri `ri1` iturburu-erregistroa eta kargatu laneko erregistroan

```
R_e1 := EM[ri1];
```

(b) batu R_e1 erregistroa eta oinarri-helbidea (IR_2-n dagoena), eta kargatu emaitza R_ual laneko erregistroan

```
R_ual := R_e1 + ALD;
```

(c) irakurri memoria, helbidea R_ual erregistrotik hartuta, eta gorde datua EMan (`rh`)

```
EM[rh] := MEM[R_ual];
```

`stx`

(a) irakurri `ri1` eta `ri2` iturburu-erregistroak (bat helbidea kalkulatzeko eta bestea memorian idazteko) eta kargatu laneko erregistroetan

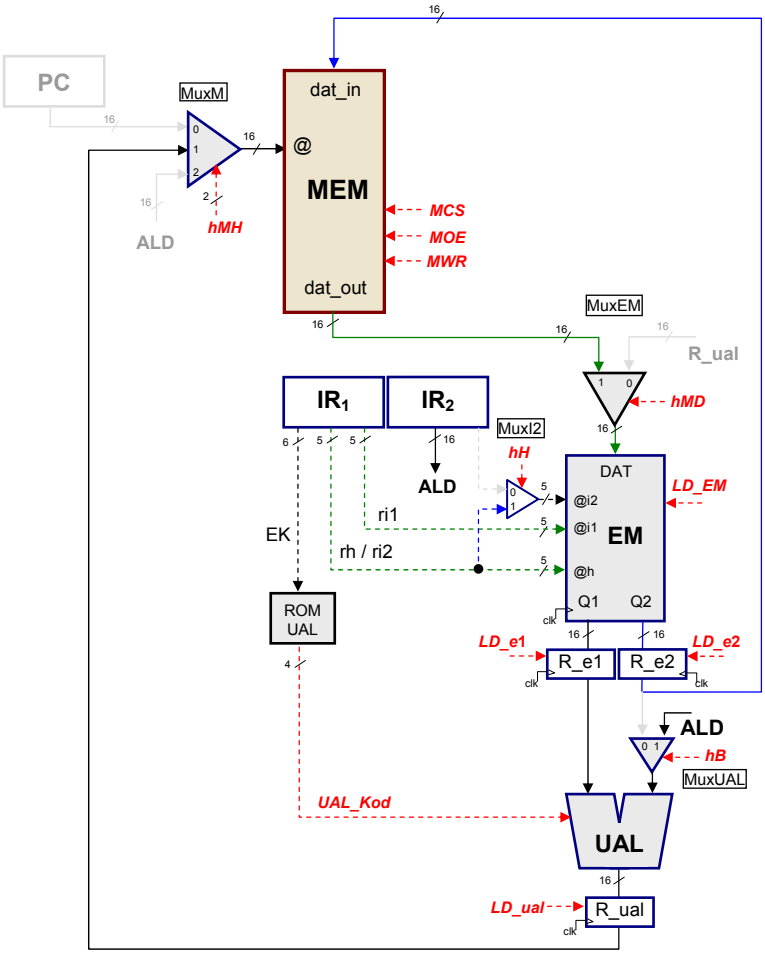
```
R_e1 := EM[ri1];      R_e2 := EM[ri2];
```

(b) batu R_e1 erregistroa eta oinarri-helbidea (IR_2-n dagoena), eta kargatu emaitza R_ual laneko erregistroan

```
R_ual := R_e1 + ALD;
```

(c) idatzi memorian, helbidea R_ual erregistrotik hartuta, R_e2 erregistroaren edukia

```
MEM[R_ual] := R_e2;
```



7.12. irudia. `ldx` eta `stx` aginduen exekuziorako erabili behar diren osagaiak.

Agindu horiek exekutatzeko, honako kontrol-seinale hauek sortu beharko ditu kontrol-unitateak:

`ldx`

(1)	R_e1	→	*LD_e*1
(2)	UAL, MuxUAL, R_ual	→	(*UAL_Kod* = +, ROMetik), *hB*, *LD_ual*
(3)	MEM, MuxM	→	*MCS*, *MOE* eta *hMH* = '1' (01)
	EM, MuxEM	→	*LD_EM* eta *hMD*

`stx`

(1)	R_e1, R_e2, MuxI2	→	*LD_e*1, *LD_e*2 eta *hH*
(2)	UAL, MuxUAL, R_ual	→	(*UAL_Kod* = +, ROMetik), *hB*, *LD_ual*
(3)	MEM, MuxM	→	*MCS*, *MWR* eta *hMH* = '1' (01)

- **`beq ri1, etiketa`** `(desplazamendua)`

Agindu bereziak dira jauziak; datuak prozesatu beharrean, aginduen fluxua kontrolatzen dute. Hau da, PC erregistroa kontrolatzen dute.

Eskuarki, programaren kontagailua banan-banan doa, baina jauzietan balio berri bat hartzen du, "aginduen gainetik jauzi" bat egiteko.

`beq` aginduaren kasuan, jauzia egin behar den ala ez jakiteko, erregistro baten balioa 0rekin konparatzen da. Konparazioa egiteko, UALaren adierazleak erabil daitezke; kasu honetan, *Z* (zero). Beraz, `ri1` erregistroa irakurri behar da, R_e1 laneko erregistroan kargatu, eta UALean prozesatu (*Y* = *A*); eragiketaren ondorioz, *Z* aktibatuko da baldin eta R_e1 laneko erregistroaren edukia 0 bada.

Konparazioaren emaitzaren arabera, kontrol-unitateak erabakiko du jauzia egin ala ez. Jauzia egin behar bada, IR_2 erregistroan (aginduan, beraz) dagoen desplazamendua erabili behar da `PC := PC + despl` egiteko, ohiko `PC := PC + 1` gehiketaren ordez. Beraz, batugailuaren sarrera bat multiplexatu behar da, 1 edo desplazamendua aukeratu ahal izateko (MuxBat, ikus 7.13. irudia).

Arazo bat daukagu jauzi-helbidearekin. Izan ere, aginduen bilaketa fasean, PC erregistroaren edukia gehitu behar izan dugu, aginduak eskuratzeko bi hitz irakurri behar direlako memorian, PC eta PC + 1 helbideetakoak. Beraz, exekutatu behar den agindua jauzia bada, hau da `PC := PC + despl` egin behar bada, une horretan PCaren edukia ez da izango dagoeneko jauzi-aginduaren helbidea, hurrengoarena baizik.

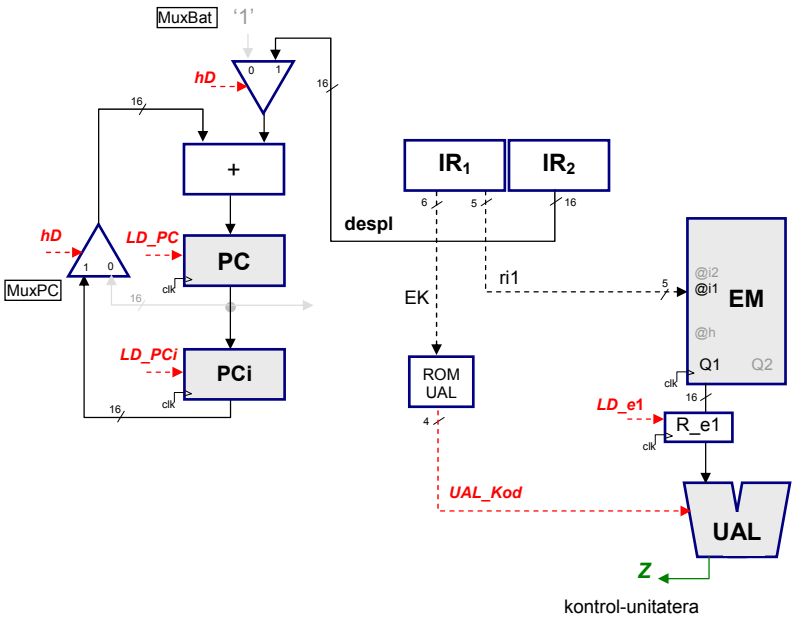
Hori dela eta, aginduaren PCaren balioa gorde egin behar da laneko erregistro batean, geroago erabili ahal izateko. Adibide honetan, PCi izeneko laneko erregistroa erabiliko dugu horretarako. Adi! Aginduaren bilaketaren lehen zikloan gorde behar da PCaren edukia PCi laneko erregistroan, eta, horretarako, *LD_PCi* kontrol-seinalea ere sortu behar da (une horretan ez badakigu ere irakurtzen ari garen agindua jauzi bat denetz).

PC edo PCi aukeratu ahal izateko, multiplexore bat —MuxPC— erabili beharko dugu batugailuaren beste sarreran, PCi + `despl` batuketa egin ahal izateko[28]. Kontrol-seinale berarekin kontrola daitezke bi multiplexoreak, bi aukera baino ez baitago:

$$hD = 0 \rightarrow \text{PC} + 1 \qquad hD = 1 \rightarrow \text{PCi} + \texttt{despl}$$

Beraz, multiplexoreen 0 sarreretan PC eta 1 konstantea kokatuko ditugu, eta 1 sarreretan PCi eta desplazamendua.

7.13. irudian ageri dira `beq` agindua exekutatzeko erabili behar ditugun osagaiak.



7.13. irudia. `beq` aginduaren exekuziorako erabili behar diren osagaiak.

[28] Hori ez da aukera bakarra; esaterako, mihiztatzaileak berak egokitu zezakeen jauzian adierazten den etiketari dagokion desplazamendua. Hala ere, hartu dugun erabakia ohikoa da, hemen azal ezin ditzakegun beste arrazoi batzuk direla medio.

Laburbilduz, honela exekutatuko da `beq` agindua:

(a) lehendabizi, `ri1` erregistroa irakurri behar da eta laneko erregistroan utzi

```
R_e1 := EM[ri1];
```

(b) gero, R_e1 erregistroaren edukia 0rekin erkatu behar da unitate aritmetiko/logikoan (horretarako, nahikoa da igarotzea A sarrera irteerara)

```
Z := (R_e1 = 0);
```

(c) aurreko konparazioaren erantzunaren arabera

```
baldin (Z) orduan PC := PCi + despl;
```

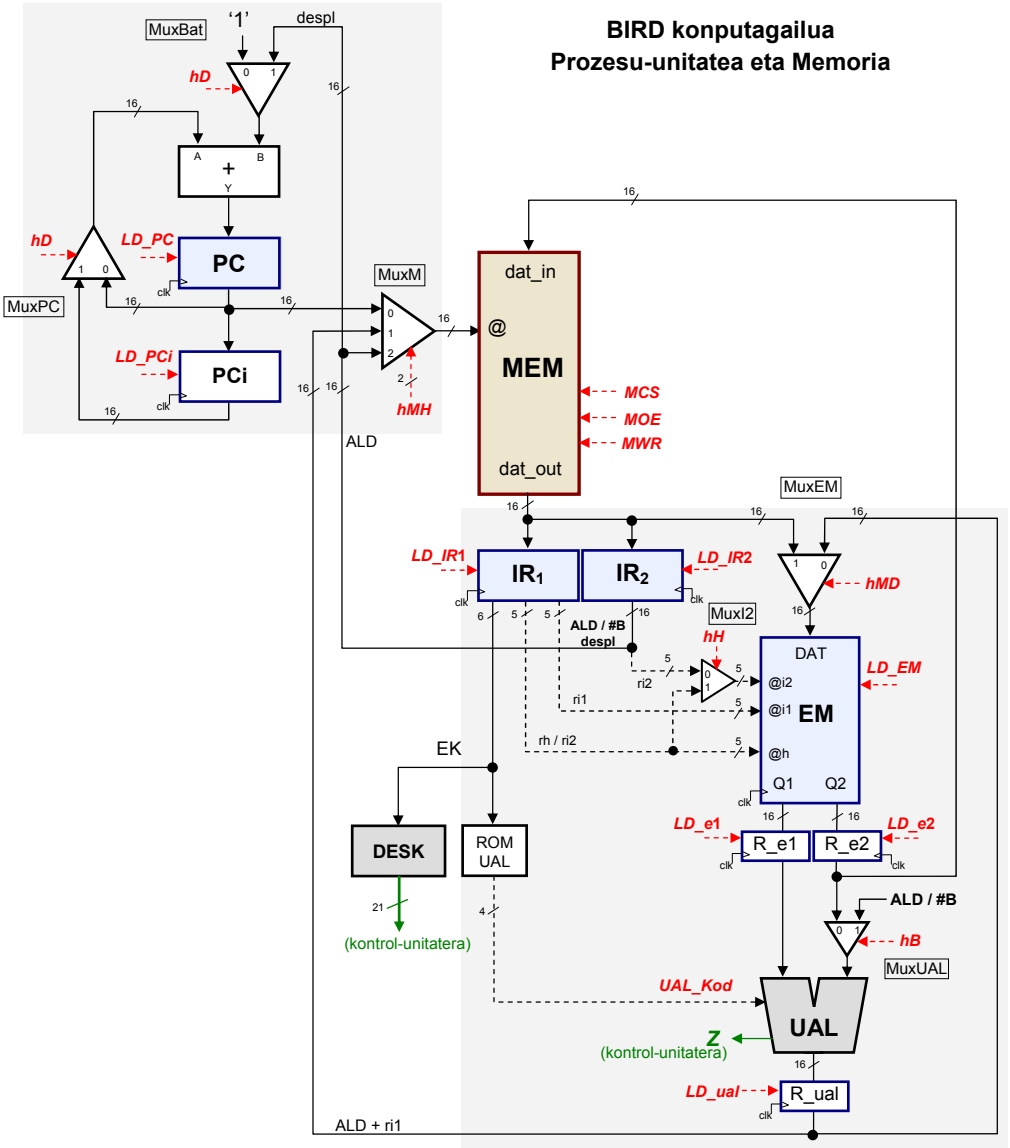
Beraz, jauzi-agindua exekutatzeko, 2 ziklo erabiliko ditugu jauzia ez bada bete behar eta 3 jauzia egin behar bada.

Eragiketa horietarako, honako kontrol-seinale hauek sortu beharko ditu kontrol-unitateak:

(1) R_e1 → *LD_e*1
(2) UAL → (*UAL_Kod* = *A*, ROMetik)
(3) PC, MuxPC, MuxBat → *LD_PC* eta *hD*

Agindu guztien exekuzioa analizatu dugu dagoeneko, behar ditugun osagaiak antolatu eta haien kontrol-seinaleak zehaztu. 7.14. irudian, aukeratu ditugun BIRD prozesadorearen 21 aginduak exekutatzeko behar den prozesu-unitatea ageri da.

Prest gaude, beraz, kontrol-algoritmoa garatzeko.



7.14. irudia. BIRD prozesadorean egitura orokorra (prozesu-unitatea eta memoria).

7.3.4. Kontrol-unitatea

Prozesu-unitatea diseinatuta, osagai guztiak kontrolatu behar dituen kontrol-unitatea garatu behar dugu[29]. Gogoratu: automata finitu bat da kontrol-unitatea, egoeraz egoera doana kontrol-seinaleak ordena egokian sortzeko. Kontrol-unitatearen logika ASM kontrol-algoritmo batez adierazten da, eta kontrol-algoritmo hori gauzatzeko biegonkorrak eta logika konbinazionala erabiltzen dira. Zorionez, oinarrizko prozesadore baten kontrol-unitatearen logika ez da oso zaila, behin eta berriz errepikatu behar baitira aginduak exekutatzeko faseak: bilaketa – deskodeketa – eragigaien irakurketa – exekuzioa – emaitzen idazketa.

Kontrol-algoritmoari ekin baino lehen, zehaztapen pare bat. Batetik, programa exekutatu ahal izateko, memorian kargatu behar da eta PC erregistroa lehen aginduaren helbidearekin hasieratu behar da. Eskuarki, sistema eragileko programa berezi batek —kargatzaileak— betetzen du prozesu hori. Ez dugu aztertuko nola egiten den lan hori; beraz, memoria eta PC erregistroa ondo kargatuta daudela suposatuko dugu.

Bestetik, kontrol-automatak zikloz ziklo aldatzen du egoera; tarte horretan, erloju-periodo bat alegia, astia izan behar du egoerari dagozkion ekintzak burutzeko. BIRD osatzeko erabili ditugun osagaien artean motelena, dudarik gabe, memoria da. Prozesadorearen diseinua ahal den sinpleena izan dadin, erabaki hau hartuko dugu: erloju-ziklo batean nahikoa denbora dago memoriako hitz bat irakurtzeko edo idazteko[30]. Ondorioz, gainerako osagaien eginkizunak ziklo batean egin ahal izango dira arazorik gabe (memoria baino azkarragoak direlako).

7.3.4.1. Kontrol-algoritmoa

Has gaitezen **kontrol-algoritmoa** garatzen, 7.14. irudiko prozesu-unitatea kontrolatzeko (automataren egoera bakoitzean egiten dena egiaztatzeko, hartu kontuan irudi hori).

---

[29] Oro har, sistema baten diseinu-prozesuan, kontrol-algoritmoaren eta prozesu-unitatearen garapena batera doaz, elkar elikatzen duten prozesuak baitira.

[30] Prozesadore baten erloju-maiztasuna kalkulu-abiadura zehazten duten parametroetako bat da (ez nagusia, hala ere). Oro har, erloju-periodoa memoriaren erantzun-denbora baino txikiagoa da, eta, ondorioz, memoria-eragiketek ziklo (egoera) bat baino gehiago behar dute.

▪ Aginduen bilaketa

Kontrol-algoritmoaren hasierako lana aginduak bilatzea da. Horretarako, 7.3.3.1. atalean ikusi dugun moduan, bi hitz irakurri behar ditugu memorian, PC eta PC + 1 helbideetakoak. Hori egiteko, beraz, automataren bi egoera erabiliko ditugu. Aginduaren lehen hitza IR_1 erregistroan kargatu behar da eta bigarrena IR_2 erregistroan.

Hauek dira, beraz, bi egoera horiek kontrol-algoritmoan:

→ Lehen zikloa: `BIL1` egoera

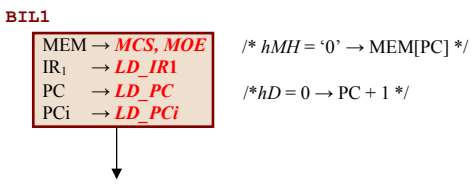
`IR1 := MEM[PC]; PC := PC + 1;`

Ziklo honetan, PC erregistroan dagoen helbidea erabili behar da memoriako helbide-sarrera gisa (horretarako, MuxM multiplexorearen *hMH* hautatze-seinaleak '0' (00) balioa izan behar du), eta memoria irakurri (*MCS*, *MOE*). Irakurritakoa IR_1 erregistroan kargatzeko, haren karga-seinalea aktibatu behar da (*LD_IR*1), hurrengo erloju-ertzean karga dadin.

Horrez gain, zikloan zehar, PC + 1 batuketa egiten ari da; zikloaren bukaeran, hurrengo erloju-ertza heltzean, PC erregistroan kargatuko da (*LD_PC*).

Eragiketa honekin erlazionatuta ez badago ere, gogoratu `beq` aginduaren exekuzioa analizatzean esan duguna: irakurtzen ari garen agindua jauzia denetz ez badakigu ere, PCaren jatorrizko balioa laneko erregistro batean gorde behar dugu, behar izatekotan, jauziaren helburu-aginduaren helbidea modu egokian kalkulatu ahal izateko. Beraz, `PCi := PC` egiteko, PCi laneko erregistroaren karga-seinalea aktibatu behar da (*LD_PCi*).

Hona hemen automataren lehen egoera[31]:



[31] Aktibatzen diren kontrol-seinaleek baino ez dute ageri behar kontrol-algoritmoan. Hala ere, kontrola nola egiten den argiago uztearren, egiten ari denarekin lotura duten baina 0 balioa hartzen duten seinaleak adierazi ditugu egoeren alboan, iruzkin gisa.

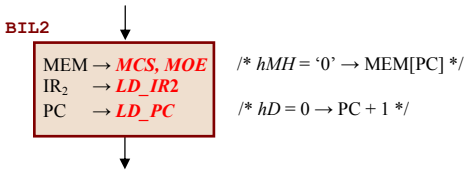
→ Bigarren zikloa: `BIL2` egoera

```
IR2 := MEM[PC]; PC := PC + 1;
```

Aurreko ziklokoa errepikatu behar da: aginduaren bigarren zatia irakurri eta IR_2 erregistroan kargatu; hau da, *MCS*, *MOE*, eta *LD_IR2* kontrol-seinaleak aktibatu behar dira.

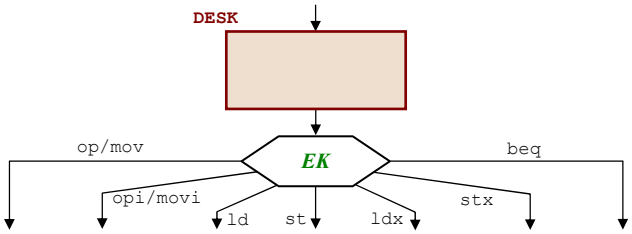
Horrez gain, PC erregistroaren balioa gaurkotuko dugu (PC + 1), programaren hurrengo aginduaren helbidea prest uzteko. Eguneratze hori edozein unetan egin daiteke, eta ohikoa da, prozesadore guztietan, bilaketaren bukaera aprobetxatzea hori egiteko. Beraz, bigarren egoera honetan ere, *LD_PC* kontrol-seinalea aktibatuko dugu.

Hau izango da, beraz, bilaketari dagokion bigarren egoera:



▪ Aginduen deskodeketa

Agindu osoa eskuratu eta IR erregistroan kargatu ondoren, deskodetu egin behar da eragiketa-kodea (EK, IR_1 erregistroko pisu handieneko 6 bitak). Hau da, deskodegailuaren irteerak analizatu behar dira. Une horretatik aurrera, kontrol-algoritmoa adarkatuko da. Adibide honetan, honako adar hauek izango ditugu `DESK` egoeran:



Adar berean bildu ditugu `op` agindu guztiak (nahikoa da agindu horiei dagozkien deskodegailuaren irteerak batzea: `add` + `sub` + ...); izan ere, desberdintasun bakarra UALeko eragiketan dago, eta, lehen azaldu dugun moduan, ROM taula batetik lortuko dugu UALak behar duen kodea. Beraz, modu berean kontrolatuko ditugu `op` motako agindu guztiak. `mov` agindua ere, adar horretan exekutatuko dugu, nahiz eta eragigai bakarra erabili behar duen. Gauza bera egin dugu `opi/movi` aginduekin.

Beraz, erloju-ertza heltzean, adar jakin batetik abiatuko da kontrol-automata, definitu dugun agindu motaren bat exekutatzeko. Ikus dezagun nola exekutatuko diren aginduak.

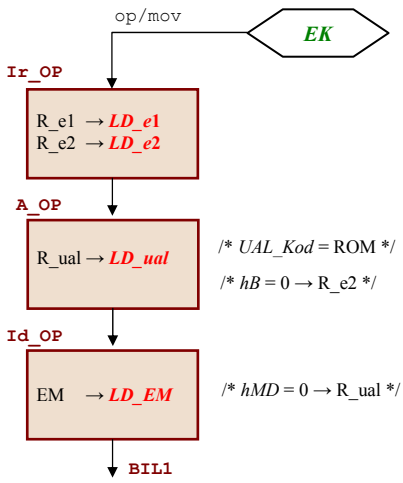
▪ `op/mov` aginduen exekuzioa

Diseinatu dugun prozesu-unitatea kontuan hartuta, hiru egoera beharko ditugu `op rh, ri1, ri2` agindua betetzeko. Lehenengoan, `ri1` eta `ri2` eragigaiak eskuratu behar ditugu EMtik, eta laneko bi erregistroetan (R_e1 eta R_e2) kargatu —*LD_e*1, *LD_e*2—. `mov rh, ri1` aginduaren kasuan, eragigai bakar bat erabiliko da; hori dela eta, berdin zaigu zer kargatuko den R_e2 erregistroan.

Bigarren egoeran, eragiketa egingo da UALean laneko erregistroetan kargatu diren bi eragigaiak erabiliz, eta emaitza R_ual laneko erregistroan kargatuko da —*LD_ual*—. UALeko eragiketa zehazteko —*UAL_Kod*— ROM taula bat erabiliko dugu. Taula horretan, agindu bakoitza exekutatzeko UALak behar duen kodea idatzita dago.

Azkenik, emaitza `rh` erregistroan kargatu behar da —*LD_EM*—. R_ual erregistroaren edukia karga dadin, 0 bidea aukeratu beharko dugu erregistro-multzoaren datu-sarreran dagoen MuxEM multiplexorean.

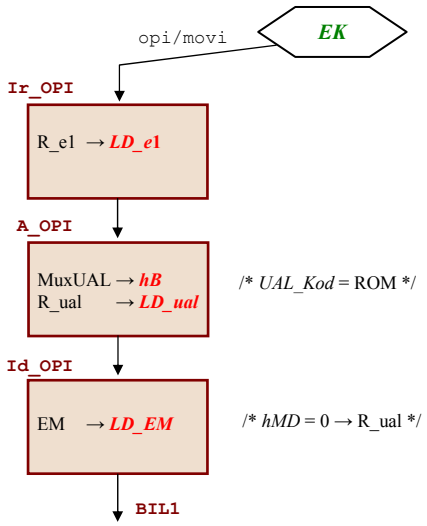
Dagoeneko amaitu da agindu horien exekuzioa; kontrol-unitatea hurrengo aginduaren bila joango da, `BIL1` egoerara. Hauek dira, beraz, `op/mov` motako aginduak exekutatzeko behar ditugun hiru egoerak (izenek egoeren egitekoa islatzen dute: eragigaien irakurketa (`Ir`), UALeko eragiketa (`A`), eta emaitzaren idazketa (`Id`)).



▪ `opi/movi` aginduen exekuzioa

Agindu hauen exekuzioa aurrekoenen antzekoa da; bigarren eragigaia, ordea, ez dator erregistro-multzotik, IR_2 erregistrotik baizik, eta, horretarako, UALeko B sarrerako multiplexorean bide egokia aukeratu behar da (*hB* = 1 MuxUAL multiplexorean, ikus 7.14. irudia).

Hauek dira, beraz, `opi/movi` motako aginduak exekutatzeko behar ditugun hiru egoerak[32].



Aurreko kasuan bezala, automata `BIL1` egoerara itzuliko da.

▪ `ld` aginduaren exekuzioa

`ld rh, ALD` agindua exekutatzeko, honako hau egin behar da: aukeratu irakurri behar den hitzaren helbidea (`ALD`, IR_2 erregistroko 16 bitak) MuxM multiplexorean —*hMH* = '2'—; irakurri memoria —*MCS*, *MOE*— eta irakurritakoa `rh` erregistroan gorde —*hMD*, *LD_EM*—.

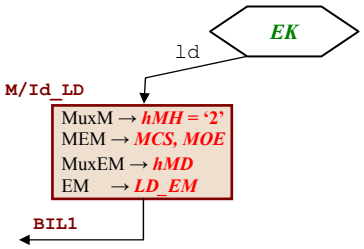
Hori guztia ziklo batean egin daiteke, baldin eta, hasieran erabaki dugun bezala, nahikoa denbora badago, ziklo batean, memoriako hitz bat irakurtzeko. Irakurketa ziklo osoan zehar egiten da, eta bukaeran, hurrengo

---

[32] `movi` aginduaren kasuan, ez da eragigairik irakurri behar; hala ere, kontrol-automata sinplifikatzearren, `opi` aginduak bezala tratatuko dugu, nahiz eta ez erabili irakurritakoa.

erloju-ertza iristean, memoriako hitza `rh` erregistroan kargatuko da (dagokion bidea irekita baitago MuxEM multiplexorean).

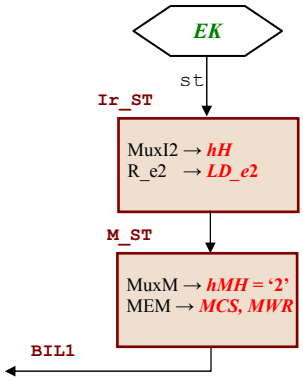
Hau da, beraz, `ld` aginduari dagokion exekuzio fasea (egoera bakar bat):



▪ `st` aginduaren exekuzioa

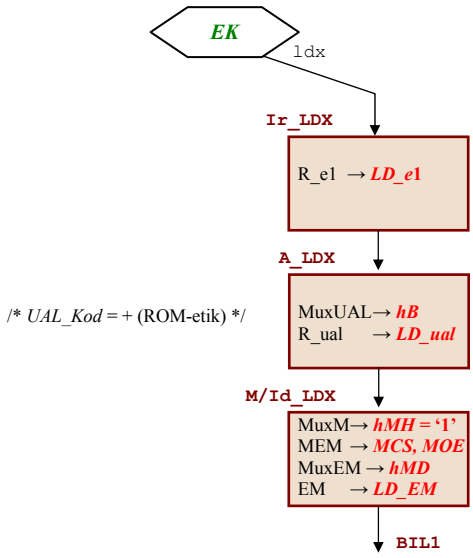
`st ri2, ALD` agindua exekutatzeko, lehendabizi `ri2` erregistroa irakurri behar da, eta R_e2 laneko erregistroan utzi —*LD_e2*—. Adi! `ri2` erregistroaren helbidea IR_1 erregistroan adierazten da (ikus 7.6. irudia), eta, beraz, bide egokia aukeratu behar da MuxI2 multiplexorean —*hH*—. Gero, R_e2 erregistroaren edukia memorian idatzi behar da (zuzeneko lotura bat dago R_e2 erregistrotik memoriako datu-sarrerara). Memorian idazteko, `ld` aginduaren kasuan bezala, `ALD` helbideari (IR_2 erregistroko 16 bitak) ireki behar zaio bidea MuxM multiplexorean —*hMH* = '2'—, eta memoriari dagozkion kontrol-seinaleak aktibatu —*MCS*, *MWR*—.

Beraz, bi ziklo behar dira `st` agindua exekutatzeko. Exekuzioa bukatuta, automata `BIL1` egoerara joango da.



▪ `ldx` aginduaren exekuzioa

`ldx` agindua exekutatzeko, lehendabizi, memoria-helbidea kalkulatu behar da; horretarako, `ri1` erregistroa irakurri behar da —*LD_e*1— eta batuketa egin UALean IR_2 erregistroan dagoen helbidearekin —*hB* eta *LD_ual*—. Azkenik, memoria irakurri behar da dagokion helbidean —*hMH* = '1', *MCS*, *MOE*— eta emaitza erregistro-multzoan utzi —*hMD*, *LD_EM*—.

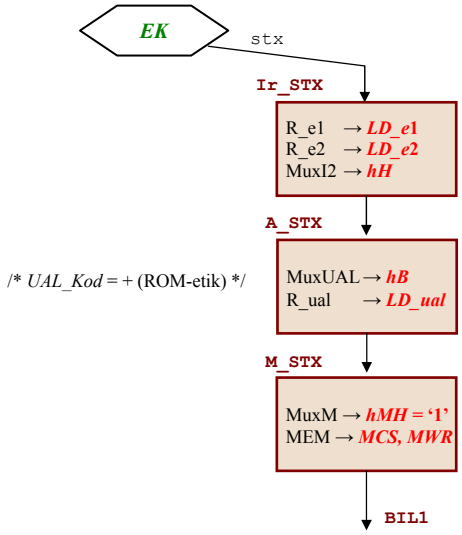


▪ `stx` aginduaren exekuzioa

`stx` aginduaren exekuzioa antzekoa da. Bi erregistro irakurri behar dira: `ri1` memoria-helbidea kalkulatzeko —*LD_e*1— eta `ri2` memorian idatzi nahi dena lortzeko. Bigarrena irakurtzeko (`st` aginduarekin gertatzen den moduan), IR_1 erregistroan adierazitako helbidea hartu behar dugu kontuan —*hH*, *LD_e*2—.

Gero, batu egin behar dira UALean IR_2 erregistroan dagoen helbidea eta R_e1-en edukia —*hB* eta *LD_ual*—. Azkenik, memoria idatzi behar da, dagokion helbidean —*hMH* = '1', *MCS*, *MWR*—.

Honela geratuko da kontrol-algoritmoa `stx` aginduaren exekuziorako:



▪ `beq` aginduaren exekuzioa

Agindu bereziak dira jauziak; datuak prozesatu beharrean, programaren fluxua kontrolatzen dute, exekutatu behar den hurrengo agindua non dagoen adieraziz.

`beq ri1, etiketa` aginduak eragigai bat lortu behar du EMtik: `ri1` erregistroaren edukia. Beraz, lehenengo egoera batean, hori egin beharko dugu: EMa irakurri eta R_e1 laneko erregistroa kargatu —*LD_e*1—.

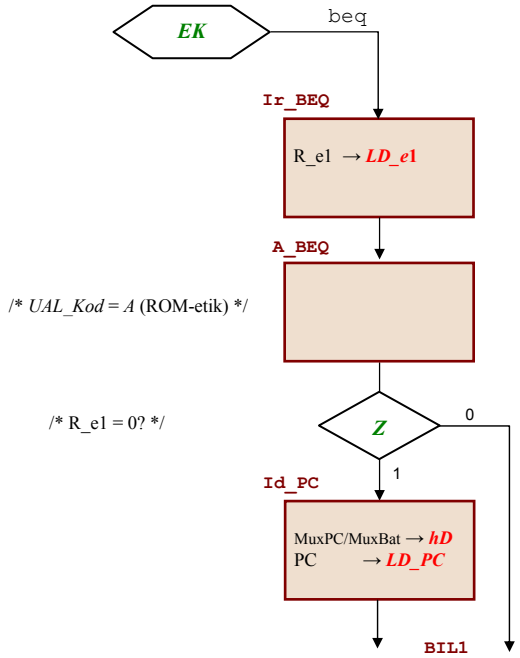
Hurrengo egoeran, 0rekiko konparazioa egingo dugu UALean (ROM taulatik datorren kodea erabiliz: $Y = A$ eragiketa). UALaren erantzuna Z adierazlean izango dugu.

UALaren informazioa eskuratuta, kontrol-unitateak erabaki bat hartuko du. $Z = 0$ bada, jauzia ez da bete behar, eta jarraitu behar da ondoz ondoko aginduarekin; izan ere, ez da beharrezkoa PCa eguneratzea, dagoeneko egin dugulako eragiketa hori aginduaren bilaketaren bigarren zikloan. Beraz, ezer egin gabe, automata `BIL1` egoerara itzuliko da, programaren exekuzioarekin jarraitzeko.

Baina $Z = 1$ bada, orduan ez da jarraitu behar ondoz ondoko agindua exekutatzen: jauziaren helburua den aginduaren helbidea PC erregistroan kargatu behar da.

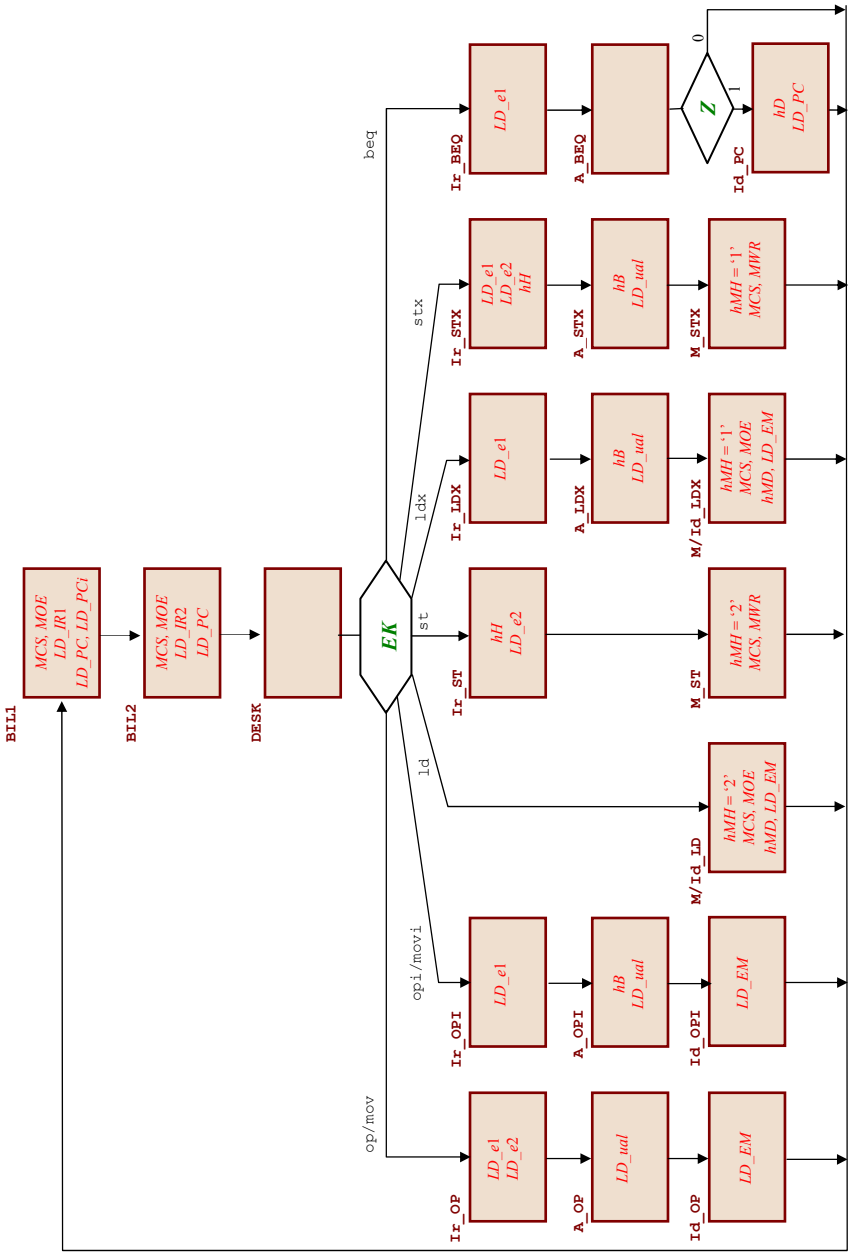
Adi! Jauziaren helburua lortzeko, desplazamendu bat (IR_2 erregistroko 16 bitak) gehitu behar zaio jauzi-aginduaren PCari; hori dela eta, PCi laneko erregistroan gorde dugu aginduaren PCa, `BIL1` egoeran. Hau da unea balio hori erabiltzeko. Beraz, MuxPC eta MuxBat multiplexoreetan, PCi eta desplazamendua aukeratu behar dira —*hD*—; eta batura, PCi + `despl`, PC erregistroan kargatu behar da —*LD_PC*—.

Hauek dira, beraz, `beq` agindua exekutatzeko behar diren egoerak:



▪ Kontrol-algoritmo osoa

Bukatu dugu BIRD prozesadoreak behar duen kontrol-algoritmoa. 7.15. irudian ageri da ASM algoritmo osoa.



7.15. irudia. BIRD prozesadorearen ASM kontrol-algoritmoa. Aginduen exekuzioan ohikoak diren faseak azpimarratu ditugu egoeren izenetan: bilaketa (BIL), deskodeketa (DESK), eragigaien irakurketa (Ir), kalkulua unitate aritmetiko/logikoan (A), memoria-eragiketak (M) eta emaitzen idazketa erregistro-multzoan (Id).

7.3.4.2. Iruzkin batzuk

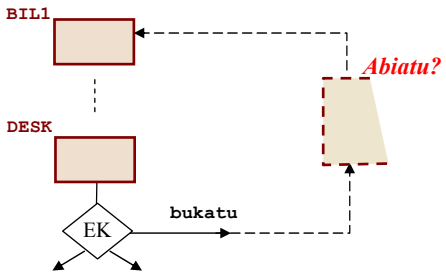
7.15. irudiko kontrol-algoritmoak islatzen du edozein prozesadoreren funtzionamendu-zikloa. Hala ere, sinplifikazio asko egin ditugu eta merezi du azalpen batzuk ematea.

- **Hasiera eta bukaera**

  Aurreko algoritmoak behin eta berriz errepikatzen du aginduen exekuzio-zikloa: bilatu, deskodetu, irakurri eragigaiak, egin kalkuluak UALean... Baina, noiz abiatzen da programaren exekuzioa eta noiz bukatzen da?

  Algoritmoan jarri ez badugu ere, argitasuna dela eta, abiatzeko seinale bat behar da. Eskuarki, seinale hori erabiltzaileak (edo sistema-eragileak) berak emango du: programaren hasiera-helbidea PC erregistroan kargatuko da eta automata `BIL1` egoerara joango da. Hortik aurrera, aginduak banan-banan exekutatuko dira, algoritmoan adierazten den moduan.

  Programaren exekuzioa bukatzeko, kode berezi bat kargatu behar da programaren azken posizioan. Kode hori detektatzen denean deskodeketa fasean, algoritmoaren exekuzioa bertan behera utziko da eta itxaron-egoera batera joango da. Handik aterako da beste programa bat exekutatzeko, lehen aipatu dugun moduan.



  Oro har, hasiera- eta bukaera-prozesuak konplexuak dira konputagailuetan (eskuarki, sistema eragilearen mende daude), eta horregatik ez ditugu sartu kontrol-algoritmoan.

- **Agindu gehiago**

  BIRD prozesadorearen 21 agindu aintzat hartu ditugu adibide gisa diseinatu dugun prozesadorean. Gehiago daude, eta modu berean kontrolatzen dira. Dudarik gabe, ariketa interesgarria da zabaltzea sortu

ditugun prozesu-unitatea eta kontrol-algoritmoa, BIRD prozesadorearen gainerako aginduak exekutatu ahal izateko.

- **Automataren egoera kopurua eta aginduen exekuzio faseak**

  6. kapituluan esan dugun bezala, kontrol-automata baten egokitasuna ez da neurtzen, inolaz ere, egoera kopuruaren arabera. Kalitate-parametro nagusiak, zalantzarik ez, zuzentasuna eta ulergarritasuna dira.

  Prozesadoreen kasuan badago kontuan hartu behar dugun beste parametro bat: **abiadura.** Programa baten exekuzio-denbora honela eman daiteke:

  $$T_{ex} = AK \times AZK \times T_{clk}$$

  non AK exekutatu diren aginduen kopurua, AZK aginduak exekutatzeko behar den ziklo kopurua (batez bestean), eta T_{clk} erloju-zikloaren periodoa baitira, hurrenez hurren.

  BIRD prozesadorean, ziklo hauek behar dira aginduak exekutatzeko:

`op / mov / opi / movi`	→	B_1	B_2	D	Ir	A	–	Id	→ 6 ziklo
`ld`	→	B_1	B_2	D	–	–		MId	→ 4 ziklo
`st`	→	B_1	B_2	D	Ir	–	M	–	→ 5 ziklo
`ldx/stx`	→	B_1	B_2	D	Ir	A		MId	→ 6 ziklo
`beq`	→	B_1	B_2	D	Ir	A	–	(Id)	→ 5 edo 6 ziklo

  Hala, 7.3.2. atalean aztertu dugun programatxoa (16 osagaiko $C = A + B$ bektore-eragiketa) exekutatzeko, honako ziklo hauek behar dira:

begiztan sartu baino lehen:	2 `movi`
	$2 \times 6 = 12$
begiztaren 15 iteraziotan:	3 `ldx`/`stx`, 3 `op`/`opi`, 2 `beq`
	$15 \times (3 \times 6 + 3 \times 6 + 5 + 6) = 705$
azken iterazioan:	3 `ldx`/`stx`, 3 `op`/`opi`, 1 `beq`
	$3 \times 6 + 3 \times 6 + 6 = 42$
guztira →	759 ziklo

  Demagun erlojua 50 MHz-ekoa dela. Orduan, $T_{clk} = 1/f = 20$ ns, eta eragiketaren exekuzio-denbora honako hau izango da: $T_{ex} = 759 \times 20$ ns $= 1{,}5$ μs.

  Aipatu dugun bezala, prozesadoreen ezaugarri garrantzitsua da exekuzio-abiadura, eta teknika asko eta oso eraginkorrak daude programen exekuzioa azkartzeko, baina, jakina, ezin ditugu testu honetan azaldu. Hala ere, eta egin dugun diseinuari begira, badugu

aukera sinple bat aginduen exekuzio-denbora laburtzeko: egoera gutxiago (ziklo gutxiago) erabili kontrol-algoritmoan agindu bakoitza prozesatzeko. Izan ere, 7.15. irudiko kontrol-algoritmoa ez dago batere optimizatuta, egoera kopurua kontuan hartuta.

Irakurlearentzat uzten ditugu optimizazio horiek, ideia batzuk emanda: irakur al daitezke eragigaiak deskodeketa egiten den bitartean, eta D eta Ir egoerak D/Ir egoera bihurtu? erabil al daitezke baldintzapeko kontrol-seinaleak egoerak aurrezteko? elimina al daiteke kontrol-seinaleren bat?

- **Memoria**

  Diseinatu dugun kontrol-algoritmoari baldintza bat jarri diogu: erloju-ziklo batean, egoera batean, nahikoa denbora dago memoria irakurtzeko edo idazteko. Hala, gutxienez, memoria-eragiketa batek behar duen denbora igaroko du kontrol-automatak egoera bakoitzean... nahiz eta memoria-eragiketa ez egin!

  Esaterako, deskodeketa egiteko behar den denbora, edo eragigaiak EMtik eskuratzekoa, edo batuketa bat egiteko behar dena, askoz txikiagoak izango dira. Ondorioa garbia da: denbora galtzen da hainbat egoeratan.

  Denbora garrantzitsua bada (hala da prozesadoreen kasuan), badago aukera bat hainbeste denbora ez galtzeko: erloju-ziklo txikiagoa erabiltzea, eta, denbora nahikorik ez badago eragiketaren bat egiteko, memoria-eragiketen kasuan adibidez, egoera bat baino gehiago erabiltzea.

  Ikus dezagun adibide bat: memoriaren erantzun-denbora 40 ns da, baina nahikoak dira 5 ns gainerako funtzioak egiteko. Erloju-zikloa 40 ns-koa bada memoria-eragiketak ziklo batean egin ahal izateko, `op` aginduaren exekuzio-denbora honako hau izango da:

  $$6 \text{ egoera} \times 40 \text{ ns} = 240 \text{ ns}$$

  5 ns-ko erloju-zikloa ere erabil daiteke, baina orduan 8 egoera (40 ns!) beharko ditugu memoria-eragiketa bat egiteko. Hala izanik, `op` aginduaren exekuzio-denbora beste hau izango litzateke:

bilaketarako:	2 hitz × 8 egoera = 16 egoera → × 5 ns = 80 ns
gainerakoa:	4 egoera × 5 ns = 20 ns
guztira	→ 100 ns

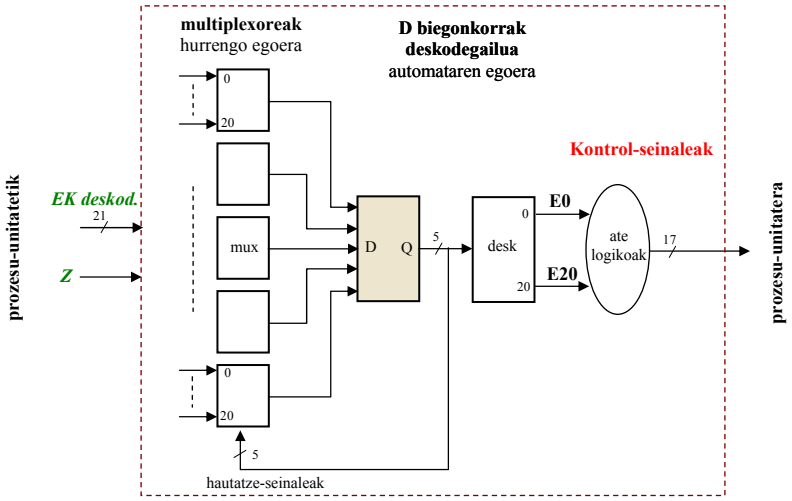
  Egoera gehiago erabiltzen badira ere, exekuzio-denbora erdira murriztu da.

7.3.4.3. Kontrol-unitatearen eraikuntza

Kontrol-algoritmoa definituta, hainbat aukera daude kontrol-unitateak sortzeko. 6. kapituluan, horietako bat aztertu dugu: multiplexoreen metodoa. Gogoratu: prozesu-unitateak behar dituen kontrol-seinaleak ordena egokian sortzea da kontrol-unitatearen helburua. Horretarako, bi atal bereizi ditugu kontrol-unitatean: uneko egoera adierazten duen atal sinkronoa, D biegonkorrez egina, eta automataren hurrengo egoera zein izango den adierazten duen atal konbinazionala, multiplexoreen bidez egina. Hortik abiatuta, erraza da kontrol-seinaleak sortzea, automataren egoera eta sarrerak kontuan hartuz eta logika konbinazional sinplea erabiliz.

7.15. irudiko kontrol-algoritmoak, optimizatu gabe, 21 egoera ditu; beraz, 5 bit behar ditugu egoerak kodetzeko. Multiplexoreen metodoa erabiltzen badugu kontrol-unitatea sortzeko, 5 D biegonkor eta 21 sarrerako 5 multiplexore erabili beharko ditugu. D biegonkorren irteerak —automataren egoera— multiplexoreen hautatze-sarreretara eraman behar dira, eta multiplexoreen irteerak —hurrengo egoera— biegonkorren D sarreretara.

Horrez gain, deskodegailu bat eta ate logiko batzuk erabili beharko ditugu kontrol-unitateak sortzen dituen 17 kontrol-seinaleak eraikitzeko. Irakurlearentzat uzten dugu multiplexoreen sarrerak zein kontrol-seinaleen ekuazioak lortzea eta kontrol-unitatea eraikitzea modu horretan.



7.16. irudia. BIRD prozesadorearen kontrol-unitatearen eskema logikoa.

Kontrol-unitate mikroprogramatua

Automataren egoera kopurua handia denean, ordea, multiplexoreen metodoa ez da oso egokia, multiplexore handiak erabili behar direlako (sarrera bat egoera bakoitzeko). Baditugu, hala ere, aukera gehiago kontrol-unitateak eraikitzeko. Horietako bat ideia sinple batean datza: D biegonkor soilak eta multiplexoreak erabili beharrean, konplexutasun-maila altuagoko osagaiak erabiltzea: **kontagailu** bat automataren egoerarako, eta **ROM** bat hurrengo egoera zein kontrol-seinaleak sortzeko. Prozesadoreen diseinuaren arloan, **kontrol mikroprogramatua**[33] deritzo kontrol-unitateak sortzeko teknika horri. Kapituluaren osagarri gisa, mota horretako kontrol-unitateen nondik norakoa azalduko dugu.

Esan dugun moduan, bi osagai hauek erabiltzen dira kontrol-unitatea eraikitzeko:

- **Kontagailu** bat sistemaren **egoera** adierazteko. Kontagailuan, bi eragiketa egiten dira: gehikuntza —E_i egoeratik E_{i+1} egoerara joateko— eta karga —E_i egoeratik E_j egoerara joateko—.

  Beraz, kontagailuaren LD eta D sarrerak kontrolatu behar ditugu (E sarrera LD sarreraren kontrakoa izango da).

- **ROM memoria** bat, automataren **hurrengo egoera** zein uneko egoerari dagozkion **kontrol-seinaleak** sortzeko.

  Egoera bakoitzeko, hitz bat izango da memorian, eta hitz horretan idatziko ditugu egoera horri dagozkion kontrol-seinaleen balioak, batetik, eta kontagailuaren LD / D sarrerak kontrolatzeko behar den informazioa, bestetik.

Edozein kontrol-algoritmo eraiki badaiteke ere, bi murriztapen jarriko dizkiogu kontrol-algoritmoaren egiturari (memorian gorde behar den informazioa mugatua izateko):

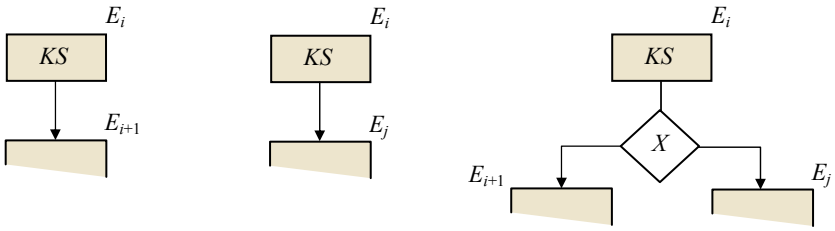
(a) Ez da onartzen baldintzapeko kontrol-seinaleak sortzea; beraz, kontrol-seinale guztiak egoeren barruan adierazi behar dira.

(b) Egoera bakoitzean, gehienez, sarrera-aldagai bat prozesa daiteke.

Bigarren murriztapena dela eta, hauek dira onartzen diren egoera-egiturak:

---

[33] Nahitaez, labur azalduko dugu nola eraiki kontrol-unitatea, eta hainbat sinplifikazio egingo dugu, ez baita kapitulu honen helburuetako bat azaltzea zehatz-mehatz kontrol-unitateak sortzeko metodo aurreratuak.



(a) kasua (b) kasua (c) kasua

(a) eta (b) kasuak antzekoak dira; sarrera-aldagairik prozesatzen ez denez, hurrengo egoera bakarra da, eta haren kodea ondoz ondokoa edo beste bat da: E_{i+1} edo E_j.

(c) kasuan, X aldagaia prozesatzen da, eta, ondorioz, hurrengo egoera bietako bat izango da, aldagaiaren balioaren arabera. Hori bai, bi horietako baten kodeak E_i egoeraren ondoz ondoko kodea izan behar du, hau da, E_{i+1}.

Aipatu ditugun bi murriztapenak direla eta, zenbait kasutan, aldaketak egin behar dira diseinatu den kontrol-algoritmoan, jarritako baldintzak bete ditzan (algoritmoaren araberakoak dira aldaketa horiek, noski, baina eskuarki errazak dira: batetik, egoera bihurtu behar dira baldintzapeko kontrol-seinaleak, eta, bestetik, sarrera-aldagai bat baino gehiago prozesatzen badira egoera berean, banatu egin behar dira, tartean egoerak gehituz), eta, horrekin batera, egoeren kodeketa egokia egin behar da[34].

Egoera bakoitzean, beraz, honako hau egin behar da: batetik, dagozkion **kontrol-seinaleak** aktibatu, eta, bestetik, **sarrera-aldagaia aukeratu** eta, haren balioaren arabera, **gehitu** kontagailuaren balioa — E_{i+1}— edo **kargatu balio berria** —E_j—. Sarrera-aldagai jakin bat aukeratu ahal izateko, aldagai guztiak multiplexore batera eramaten dira, eta multiplexore horren hautatze-seinaleak erabiltzen dira bat aukeratzeko. Sarrera-aldagaiez gain, 1 balio konstantea ere jarriko dugu multiplexorean, sarrerak prozesatzen ez dituzten egoeretan erabaki jakin bat hartu ahal izateko (ikus 7.17 irudia).

Egoera bakoitzari dagokion informazio hori guztia ROM memorian gordeko dugu; hala, nahikoa izango da egoera bakoitzean ROM memoria irakurtzea, hurrengo egoera eta aktibatu behar diren kontrol-seinaleak zein

[34] Aldaketa hori dela eta, kontrol-algoritmoak egoera gehiago izan ohi ditu. Oro har, beraz, ziklo (denbora) gehiago beharko du bere lana betetzeko. Denbora baldin bada parametro garrantzitsuena, agian multiplexoreen metodoa (edo antzekoa) erabili beharko genuke: multiplexoreak, D biegonkorrak, deskodegailua eta ate logikoak (prozesadoreen diseinuaren arloan, “kontrol kableatua” deritzo metodo horri).

diren jakiteko. Beraz, honako informazio hau gordeko dugu ROM memorian, egoera bakoitzari dagokion posizioan:

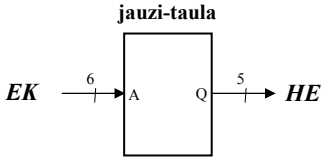
- *SA:* prozesatu behar den sarrera-aldagaia.

  Sarrera-aldagaiak multiplexore batera eraman direnez, multiplexore horren hautatze-seinaleen balioa adierazi behar dugu, egoera horretan prozesatu behar den sarrera-aldagaia aukeratzeko.
- *B:* sarrera-aldagaiak hartu behar duen balioa kontrol-algoritmoan jauzia egiteko (E_j egoerara joateko): 1 edo 0.
- *JE:* jauzia egiten bada, zein den hurrengo egoera.
- Kontrol-seinale guztien balioa (1 / 0) egoera horretan.

Egin dezagun, bada, 7.15. irudiko algoritmoa betetzen duen kontrol-unitatea. Lehenengo pausoa egoerak kodetzea da, jarri dugun murriztapena kontuan harturik: bi egoeratara joatea badago, horietako batek ondoz ondoko kodea izan behar du. Esaterako, honela kode daiteke BIRD prozesadorearen kontrol-algoritmoa:

			BIL1 → 0 BIL2 → 1 DESK → 2			
Ir_OP → 3	Ir_OPI → 6		Ir_ST → 10	Ir_LDX → 12	Ir_STX → 15	Ir_BEQ → 18
A_OP → 4	A_OPI → 7			A_LDX → 13	A_STX → 16	A_BEQ → 19
Id_OP → 5	Id_OP → 8	M/Id_LD → 9	M_ST → 11	M/Id_LDX → 14	M_STX → 17	Id_PC → 20

Kontrol-algoritmoaren egiturak eta egin dugun kodeketak lehen jarri ditugun bi murriztapenak betetzen dituzte, kasu batean izan ezik: eragiketa-kodeak analizatzen dituen egoeran. Izan ere, `DESK` egoeratik hainbat egoeratara joan daiteke, ez bitara bakarrik. Arazoa konpon liteke aginduen deskodeketa banan-banan (zikloz ziklo) egiten, baina horrek ziklo (denbora) asko gehituko lioke aginduen exekuzioari. Hori egin beharrean, eta salbuespen gisa, hau egiten da: gorde agindu bakoitzerako, taula batean (ROM), zein egoeratara joan behar duen kontrol-algoritmoak deskodeketa egoeran dagoenean.



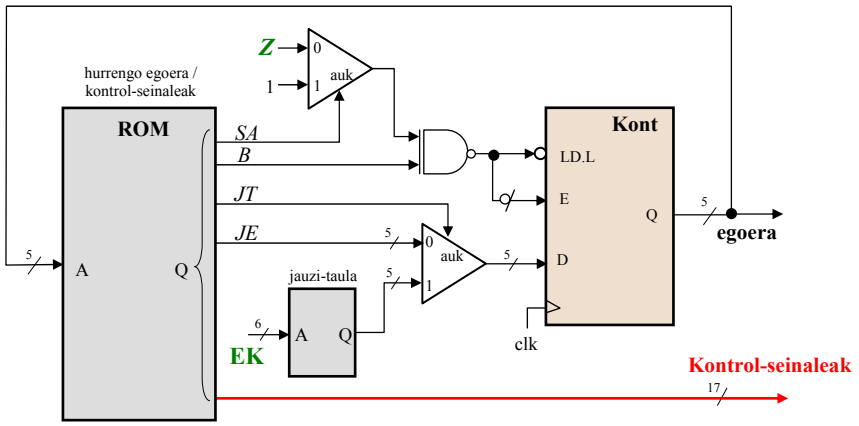
Hala, nahikoa da irakurtzea jauzi-taula hori deskodeketaren hurrengo egoera zein den jakiteko (bidenabar, eta beste zerbaitetarako erabili behar ez bada, soberan geratzen da prozesu-unitatean erabili dugun eragiketa-kodearen deskodegailua).

BIRD prozesadorearen kontrol-algoritmorako, egin dugun egoeren kodeketa kontuan izanik, honako informazio hau gorde beharko dugu jauzi-taulan:

`op/mov`[35]	→ 3	`ldx`	→ 12
`opi/movi`	→ 6	`stx`	→ 15
`ld`	→ 9	`beq`	→ 18
`st`	→ 10		

Deskodeketa fasean erabiliko den jauzi-taula dela eta, "jauzi" bat egin behar denean kontrol-algoritmoan, helburu-egoera bi iturritatik etor daiteke: ROM kontrol-memoriatik (ohikoa) edo jauzi-taula berezi batetik (eragiketa-kodea analizatzen denean bakarrik). Hortaz, multiplexore bat erabili beharko dugu, bata edo bestea aukeratzeko; multiplexore horren hautatze-seinalea ere ROM memorian idatziko dugu (*JT*). Hala *JT* = 1 izango da deskodeketa egoeran, eta 0 gainerako egoeretan.

7.17. irudian ageri da BIRD prozesadorearen kontrol-unitatearen eskema orokorra.



7.17. irudia. BIRD prozesadorearen kontrol-unitate mikroprogramatua.

[35] `op` (eta `opi`) agindu asko ditugu (`add`, `sub`, ...); beraz, jauzi-taulan, bakoitzari dagokion posizioan, hurrengo egoeraren helbidea idatzi beharko dugu: 3a (edo 6a), denak berdin exekutatzen baitira.

Prozesu-unitatetik heltzen den EKa, beraz, jauzi-taulan erabiltzen da (kontrol-automata adarkatzeko deskodeketa fasean).

Horrez gain, seinale bakar bat datorkigu prozesu-unitatetik: *Z* adierazlea. Beraz, bi sarrerako —*Z* eta 1— multiplexorea behar da kontagailuaren LD sarrera kontrolatzeko; 1 konstantea aukeratuko dugu sarrera-aldagairik prozesatzen ez den egoeretan, $E_i \rightarrow E_{i+1}$ (segi) edo $E_i \rightarrow E_j$ (jauzi) egin ahal izateko. Jauzi bat egin behar denean, hurrengo egoeraren iturburua bikoitza da: ROM memorian idatzitakoa edo jauzi-taulan idatzi dena.

Gauza bakar bat falta zaigu kontrol-unitatea bukatzeko: ROM memoriaren edukia idaztea, 7.15. irudiko kontrol-algoritmoa eta egoeren kodeketa kontuan hartuta. ROM memoriaren eduki osoa 7.18. irudian ageri da.

Azter ditzagun adibide batzuk:

- `BIL1` egoera (*E*0) (ROM memoriako 0 helbidea):

  (a) Hurrengo egoera:

  - sarrera-aldagaia? ez → *SA* = 1 (1 konstantea)
  - jauzia egiteko balioa? → *B* = 0
    (izan ere, multiplexorearen 1 sarrera 1 konstantea da; beraz, *B* = 0 balioarekin konparatuta, LD sarrera 0 izango da, eta E sarrera 1)
  - jauzi-taula erabili behar da? ez → *JT* = 0
  - jauzi-egoera? berdin da, ez baita jauzia egingo → *JE* = ----

  Informazio hori nahikoa da hurrengo egoera *E*1 (`BIL2`) izango dela ziurtatzeko, kontagailuaren E sarrera aktibatuta dago eta.

  (b) Kontrol-seinaleak:

  - *MCS*, *MOE*, *LD_IR*1, *LD_PC*, *LD_PCi*
  - Gainerakoek 0 izan behar dute: *MWR* edo *LD_EM*...
    Hala ere, zenbait kontrol-seinaleren balioa edozein izan daiteke. Esaterako, agindua bilatzen ari denean, UALean dagoen MuxUAL multiplexorearen hautatze-seinalearen balioak ez du garrantzirik, ez baita eragiketarik egingo UALean; beraz, 0 edo 1 izan daiteke.

- `DESK` egoera (*E*2) (ROM memoriako 2 helbidea):

  (a) Hurrengo egoeraren kalkulua berezia da, ez baita irakurri behar ROM memorian, jauzi-taulan baizik. Beraz,

  - sarrera-aldagaia? ez → *SA* = 1 (1 konstantea)
  - jauzia egiteko balioa? → *B* = 1, jauzia egin behar baita
  - jauzi-taula erabili behar da? bai → *JT* = 1
  - jauzi-egoera? berdin da, ez da erabiliko (jauzi-taulatik hartuko baita) → *JE* = ----

(b) Kontrol-seinaleak:
- bat ere ez; eskuarki, denak 0.

- `Id_OP` egoera (*E*5) (ROM memoriako 5 helbidea):

  (a) Hurrengo egoera ez da ondoz ondokoa, *E*0 baizik
  - sarrera-aldagaia? ez → *SA* = 1 (1 konstantea)
  - jauzia egiteko balioa? → *B* = 1, jauzia egin behar baita
  - jauzi-taula erabili behar da? ez → *JT* = 0
  - jauzi-egoera? *E*0 → *JE* = 0000

  (b) Kontrol-seinaleak:
  - *LD_EM*, emaitza `rh` erregistroan kargatzeko

Uneko egoera (@)	Hurrengo egoera				Kontrol-seinaleak															
	SA	*B*	*JT*	*JE*	*MCS*	*MOE*	*MWR*	*hMH*	*LD_IR*1	*LD_IR*2	*LD_EM*	*hMD*	*hH*	*LD_e*1	*LD_e*2	*hB*	*LD_ual*	*LD_PC*	*LD_PCi*	*hD*
0	1	0	0	0000	1	1	0	00	1	0	0	0	0	0	0	0	0	1	1	0
1	1	0	0	0000	1	1	0	00	0	1	0	0	0	0	0	0	0	1	0	0
2	1	1	1	0000	0	0	0	00	0	0	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0000	0	0	0	00	0	0	0	0	0	1	1	0	0	0	0	0
4	1	0	0	0000	0	0	0	00	0	0	0	0	0	0	0	0	1	0	0	0
5	1	1	0	0000	0	0	0	00	0	0	1	0	0	0	0	0	0	0	0	0
6	1	0	0	0000	0	0	0	00	0	0	0	0	0	1	0	0	0	0	0	0
7	1	0	0	0000	0	0	0	00	0	0	0	0	0	0	0	1	1	0	0	0
8	1	1	0	0000	0	0	0	00	0	0	1	0	0	0	0	0	0	0	0	0
9	1	1	0	0000	1	1	0	10	0	0	1	1	0	0	0	0	0	0	0	0
10	1	0	0	0000	0	0	0	00	0	0	0	0	1	0	1	0	0	0	0	0
11	1	1	0	0000	1	0	1	10	0	0	0	0	0	0	0	0	0	0	0	0
12	1	0	0	0000	0	0	0	00	0	0	0	0	0	1	0	0	0	0	0	0
13	1	0	0	0000	0	0	0	00	0	0	0	0	0	0	0	1	1	0	0	0
14	1	1	0	0000	1	1	0	01	0	0	1	1	0	0	0	0	0	0	0	0
15	1	0	0	0000	0	0	0	00	0	0	0	0	1	1	1	0	0	0	0	0
16	1	0	0	0000	0	0	0	00	0	0	0	0	0	0	0	1	1	0	0	0
17	1	1	0	0000	1	0	1	01	0	0	0	0	0	0	0	0	0	0	0	0
18	1	0	0	0000	0	0	0	00	0	0	0	0	0	1	0	0	0	0	0	0
19	0	0	0	0000	0	0	0	00	0	0	0	0	0	0	0	0	0	0	0	0
20	1	1	0	0000	0	0	0	00	0	0	0	0	0	0	0	0	0	1	0	1

7.18. irudia. Kontrol-unitateko ROM memoriaren edukia (grisez, 0ak idatzita badaude ere, edozein balio har dezaketen seinaleak). ROM kontrol-memoriaren tamaina: 21 × 7 bit.

- `A_BEQ` egoera (*E*19) (ROM memoriako 19 helbidea):

  (a) Hurrengo egoera *Z* aldagaiaren araberakoa da; beraz,

  - sarrera-aldagaia? *Z* → *SA* = 0
  - jauzia egiteko balioa? → *B* = 0, *E*0-ra joan *Z* = 0 bada
  - jauzi-taula erabili behar da? ez → *JT* = 0
  - jauzi-egoera? *E*0 → *JE* = 0000

  (b) Kontrol-seinaleak:
  - bat ere ez

- `Id_PC` egoera (*E*20):

  (a) Hurrengo egoera ez da ondoz ondokoa, *E*0 baizik

  - sarrera-aldagaia? ez → *SA* = 1 (1 konstantea)
  - jauzia egiteko balioa? 1 → *B* = 1, jauzi egin behar da
  - jauzi-taula erabili behar da? ez → *JT* = 0
  - jauzi-egoera? *E*0 → *JE* = 0000

  (b) Kontrol-seinaleak:
  - *LD_PC*, *hPCi*, *hD*

BIRD prozesadorearen diseinua bukatuta dago: prozesu-unitatea eta kontrol-unitatea definituta daude. Kontrol-unitateak 21 egoera ditu eta 17 kontrol-seinale sortzen ditu. Prozesu-unitateak aginduak exekutatzen ditu, kontrol-unitateak adierazten duen moduan, eta adierazle bat itzultzen dio kontrol-unitateari: *Z*.

Diseinu-adibidetzat hartu behar da garatu dugun prozesadorea. Jakina, prozesadore errealak askoz konplexuagoak dira eta ezin dira maila honetako liburu batean azaldu. Hala ere, sistema digitalen diseinuaz gain, prozesadoreen funtzionamenduaren eta makina-mailako aginduen kontzeptu garrantzitsuenak landu ditugu kapitulu honetan.

Hurrengo atalean, ariketa batzuk ebatziko ditugu, diseinatu dugun prozesadorearen funtzionamendua argiago uzteko asmoz.

7.4. ARIKETA EBATZIAK

Hurrengo orrialdeetan, ariketa batzuk ebatziko ditugu. Lehenengo lau ariketetan, aginduen exekuzioak aztertuko ditugu, urratsez urrats, kronogramen bidez.

Antzeko analisia egingo dugu hurrengo bi ariketetan, baina kronogrametako seinale guztiak ikusi beharrean, erregistroen edukietan zentratuko gara.

Zazpigarren ariketan, agindu bat gehituko diogu BIRD prozesadoreari, eta, azken bietan, programen kodeketa eta exekuzio-denboren kalkuluak landuko ditugu.

7.1. Ariketa

`ld r7, A` agindua exekutatu behar da BIRD konputagailuan. Egin ezazu aginduaren exekuzio osoa (bilaketatik hasita) erakusten duen kronograma bat. Azal itzazu kronograman aginduaren exekuzioarekin zerikusia duten kontrol-seinale guztiak eta gailu guztien edukia.

Hartu hasiera-datu hauek kontuan: aginduaren memoria-helbidea, 4000H; `A` aldagaiaren helbidea eta edukia, 1000H eta 6, hurrenez hurren.

■

`ld` agindua exekutatzeko, honako egoera hauetatik igaro behar du kontrol-unitateak (ikus 7.15. irudia): `BIL1`, `BIL2`, `DESK` eta `M/Id_LD`.

Bilaketa fasea azaltzeko, PCarekin, memoriarekin eta IR erregistroekin erlazioa duten seinale hauek jarriko ditugu kronograman: batetik, *hD*, PCaren batuketaren eragigaiak eta emaitza, *LD_PC*, PC, *LD_PCi* eta PCi; bestetik, *hMH*, @MEM, *MCS*, *MOE* eta MEM (irteera, dat_out); eta, azkenik, *LD_IR*1, IR_1, *LD_IR*2, eta IR_2. Gero, azken fasean EMa erabiltzen denez, hari buruzko honako informazio hau emango dugu: *hMD*, EM_DAT, *LD_EM* eta EM[r7].

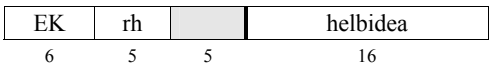
Informazio hori guztia nahikoa da aginduaren exekuzioa pausoz pauso azaltzeko.

Adi! bitarrez eman beharrean, hamaseitarrez emango dugu hainbat datu. Zenbakiak 16 oinarrian kodetzeko, <0...9, A...F> digituak erabiltzen dira:

digitu bakoitzak 4 bit ordezkatzen ditu (A = 10, B = 11, C = 12, ...; ikus E1 eranskina). H bat gehitu ohi da kodearen bukaeran, 16 oinarrian dagoela gogorarazteko.

Azter dezagun `ld r7, A` aginduaren kodeketa:

`ld rh, ALD`



EK → 000000 (ikus 7.6. irudia)
rh → 00111 (r7)
@A → 0001 0000 0000 0000 (1000H)

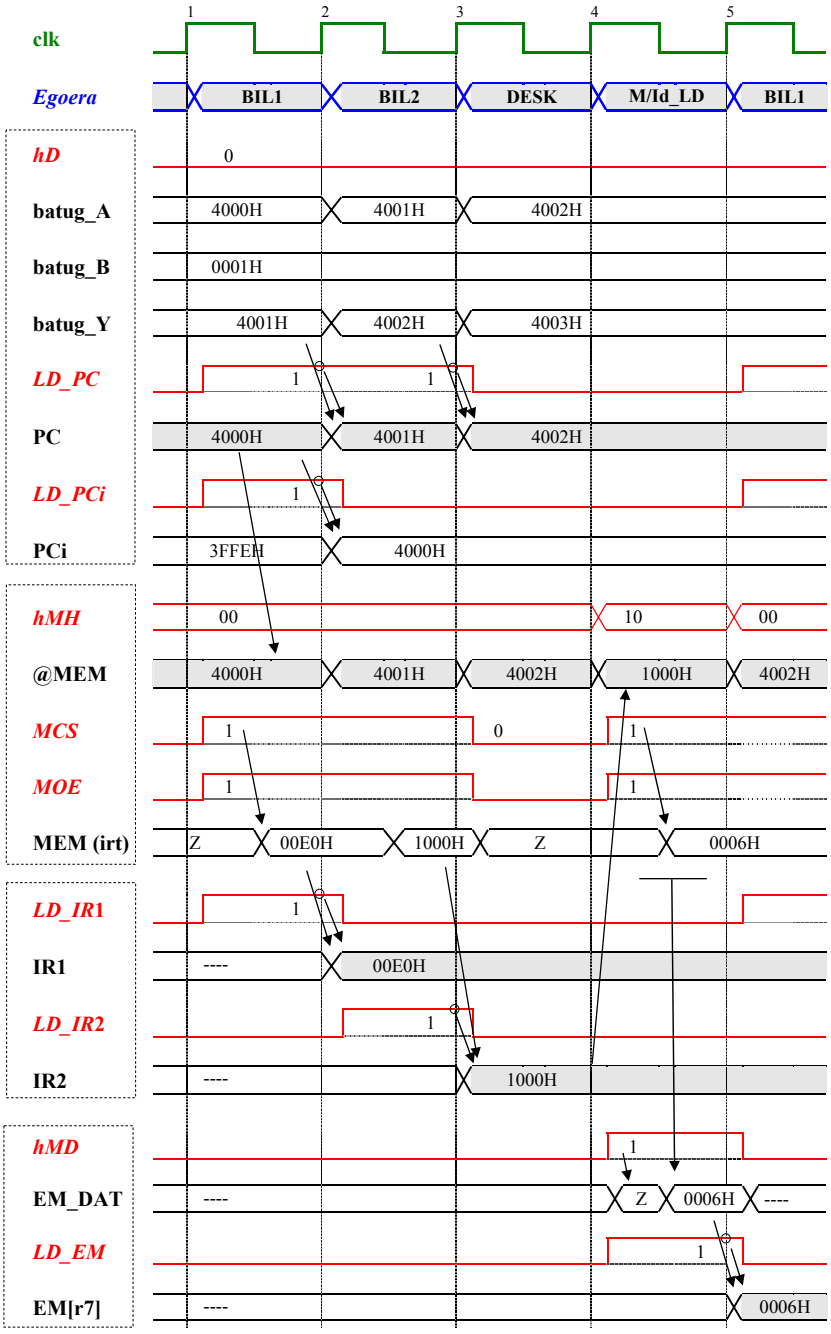
Hau da, bitarrez: 000000 00111 00000 | 0001 0000 0000 0000
edo 16 oinarrian: 00E0H 1000H

Kronograma ondo bete ahal izateko, BIRD prozesadorearen prozesu-unitatea eta kontrol-unitatea erabili behar dira (7.14. eta 7.15. irudiak). 7.19. irudiko kronograman zehaztu dugu kontrol-seinaleen eta datuen eboluzioa `ld r7, A` aginduaren exekuzioan zehar.

Lehenengo erloju-ertzarekin, automata `BIL1` egoerara doa agindua memoriatik eskuratzeko (bilaketa fasea, PC = 4000H). Egoera horretan, honako kontrol-seinale hauek aktibatzen dira. Batetik, *MCS, MOE* eta *LD_IR*1, aginduaren lehen hitza irakurtzeko eta IR_1 erregistroan kargatzeko; horretarako, MuxM multiplexorearen *hMH* hautatze-seinalea 00 izango da, PC erregistroaren edukia aukeratzeko memoria-helbide gisa. Hasieran, memoria-irteera inpedantzia altuko egoeran dago, *MCS* seinalea 0 delako; *MCS* eta *MOE* seinaleak aktibatuta, denbora-tarte bat beharko da memoriaren irteeran balio bat izateko, kasu honetan aginduaren lehen hitza: 00E0H.

Bestetik, *LD_PC* eta *LD_PCi* seinaleak aktibatzen dira, PCaren edukia gehitzeko eta PCaren balioa PCi laneko erregistroan salbatzeko. PC erregistroaren sarreran dagoen batugailuan, PC eta 1 eragigaiak aukeratu dira, multiplexoreen hautatze-seinaleak —*hD*— 0 direlako. Bidenabar, PCi laneko erregistroan, balio berria kargatu baino lehen, exekutatu den aurreko aginduaren helbidea egongo da; jauzi bat ez bada izan, 3FFEH = 4000H – 2.

Beraz, erloju-ertza heltzean, aginduaren lehen hitza IR_1 erregistroan kargatuko da, eta aginduaren bigarren hitzaren helbidea PC erregistroan, eta, bidenabar, PCaren jatorrizko balioa PCi erregistroan. Automata `BIL2` egoerara doa.



7.19. irudia. `ld r7,A` aginduaren exekuzioaren kronograma.

`BIL2` egoeran, aurreko zikloko lana errepikatzen da gutxi gorabehera: memoria irakurtzen da, PC helbide gisa erabilita (4001H) —*MCS, MOE* eta *hMH* = 00—; memoriaren irteeran, beraz, `ld` aginduaren bigarren hitza izango dugu: 1000H. *LD_IR*2 seinalea aktibatu da, eta, beraz, erloju-ertza heltzean, IR_2 erregistroan kargatuko da aginduaren bigarren zatia.

Era berean, *LD_PC* seinalea aktibatu da; ondorioz, erloju-ertza heltzean, PC + 1 = 4002H —*hD* = 0— kargatuko da PC erregistroan.

3. erloju-ziklora pasatzean, algoritmoa `DESK` egoerara aldatuko da. Egoera horretan ez da kontrol-seinalerik aktibatzen; adi! *MCS* = 0 denez, memoriaren irteera Z egoerara pasatuko da. 7.14. irudian ageri den moduan, deskodegailu batek deskodetzen du eragiketa-kodea, IR_1 erregistroko lehenengo 6 bitak: 000000 → `ld`. Beraz, hurrengo erloju-ertzarekin, automata `ld` aginduaren exekuzioari dagokion egoerara joango da: `M/Id_LD`.

4. zikloan, memoria-posizio bat irakurtzen da —*MCS* eta *MOE*—, baina orain helbidea IR_2 erregistrotik dator —`A` = 1000H—, MuxM multiplexoreko *hMH* hautatze-seinalea 10, ‘2’, delako; memoriaren irteeran, beraz, `A` aldagaiaren balioa agertuko da —0006H—. Gainera, datua erregistro-multzoan idazteko kontrol-seinalea aktibatu da —*LD_EM*— eta datu-sarrerako MuxEM multiplexoreko hautatze-seinalea —*hMD*— 1 da; hau da, prest dago memoriako datua dagokion erregistroan (`r7` kasu honetan, IR_1 erregistroan adierazten den moduan) kargatzeko.

Beraz, erloju-ertza heltzean, 6ko bat kargatuko da EMko `r7` erregistroan. Hala bukatzen da `ld` aginduaren exekuzioa; automata `BIL1` egoerara itzultzen da, hurrengo aginduaren bila, 4002H helbidekoa.

«

7.2. Ariketa

`stx r4, A[r5]` agindua exekutatu behar da BIRD konputagailuan. Egin ezazu aginduaren exekuzioa (deskodeketa fasetik hasita) erakusten duen kronograma bat. Azal itzazu kronograman aginduaren exekuzioarekin zerikusia duten kontrol-seinale guztiak eta gailu guztien edukia.

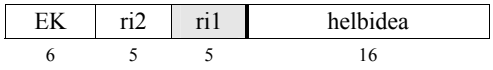
Hartu kontuan datu hauek: aginduaren memoria-helbidea, 4000H; `A` aldagaiaren helbidea, 1000H; `r4` = 20 (0014H); eta `r5` = 15 (000FH).

■

Aurreko ariketan egin dugun moduan, agindu baten exekuzioaren analisia egin behar dugu. Lehenengo bi zikloak —`BIL1` eta `BIL2`— berdinak direnez agindu guztietarako, ez ditugu berriro azalduko.

`stx r4, A[r5]` aginduaren formatua honako hau da (ikus 7.6. irudia):

`stx ri2, ALD[ri1]`



EK → 000101 (`stx`)
ri2 → 00100 (r4); ri1 → 00101 (r5);
@A → 0001 0000 0000 0000 (1000H)

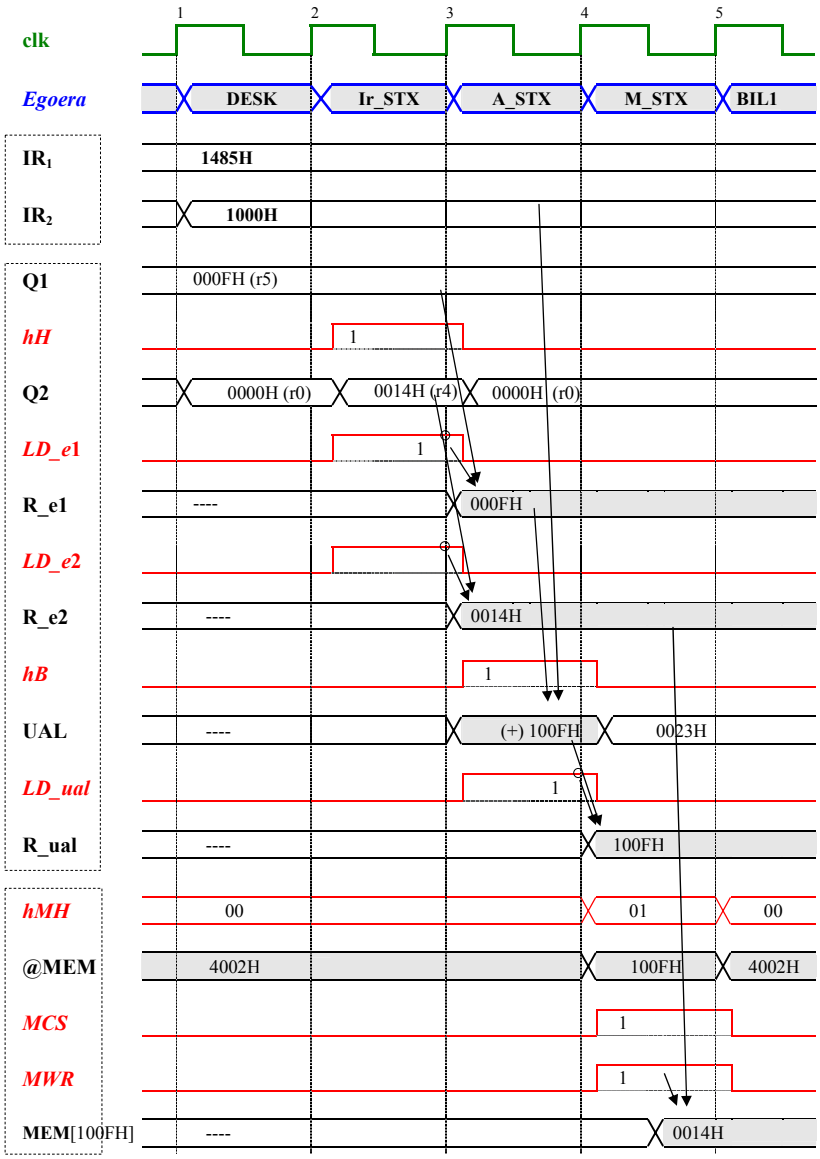
Hau da, bitarrez: 000101 00100 00101 | 0001 0000 0000 0000
edo 16 oinarrian: 1485H 1000H

Bilaketa fasea egin eta gero, egoera hauetatik igaroko da kontrol-unitatea: `DESK`, `Ir_STX`, `A_STX` eta `M_STX` (ikus kontrol-algoritmoa 7.15. irudian).

Deskodeketa fasean hasiko dugu analisia. IR_1 erregistroan, aginduaren lehen hitza kargatuta dago, eta bigarren hitza erloju-ertzarekin batera kargatu da IR_2 erregistroan, `DESK` egoerara igarotzean. Ziklo horretan, deskodegailuak `stx` agindua detektatzen du; ondorioz, automata `Ir_STX` egoerara joango da hurrengo erloju-ertzean.

Egoera horretan, dagoeneko EMko Q1 eta Q2 irteera-busetan dauden `r5` eta `r4` erregistroen edukiak laneko bi erregistroetan kargatzeko agindua ematen da —*LD_e*1 eta *LD_e*2—. Izan ere, irakurri diren erregistroen helbideak IR_1 erregistroan daude, aginduaren eragiketa-kodearekin batera. Kontuz, aurreko egoeran, `r0` erregistroaren edukia zegoen Q2 irteeran, IR_2 erregistroko azken 5 bitak erabiltzen ari direlako bigarren eragigaia irakurtzeko; baina `stx` aginduaren kasuan, IR_1 erregistroko erdiko 5 bitak

erabili behar dira, eta, horretarako, MuxI2 multiplexoreko *hH* hautatzelerroak 1 izan behar du (ikus 7.14. irudia). Erloju-ertza heltzean, R_e1 eta R_e2 erregistroak kargatuko dira (000FH eta 0014H, hurrenez hurren), eta automata `A_STX` egoerara igaroko da.



7.20. irudia. `stx r4,A[r5]` aginduaren exekuzioaren kronograma.

`A_STX` egoeran, unitate aritmetikoarekin lan egiten da. Eragigai bat R_e1 erregistrotik dator (000FH). Bestea, ordea, IR_2 erregistroan dagoen berehalako datua da (1000H), MuxUAL multiplexorearen hautatze-seinalea 1 delako —*hB*—.

UALean batuketa bat egin behar da; batuketarako behar den kodea ROM taulatik etorriko da.

LD_ual seinalea aktibatzen da, emaitza R_ual erregistroan kargatzeko. Beraz, erloju-ertzean, 1000H + 000FH = 100FH balioa kargatuko da R_ual erregistroan, hurrengo egoeran memoria-helbide gisa erabili beharko duguna. Erloju-ertza heltzean, automata `M_STX` egoerara doa.

Azken egoera horretan, memoria idazten da. MuxM multiplexorearen hautatze-seinalea *hMH* = 01 da, eta, beraz, helbidea R_ual-etik hartuko da. *MCS* eta *MWR* seinaleak aktibatu dira idazketa egiteko, eta idatzi behar den datua `Ir_STX` fasean R_e2 erregistroan kargatu dena da (0014H). Idazketarako behar den denbora igaro behar da egoera horretan; gero, erloju-ertza heltzean, automata `BIL1` egoerara itzuliko da, hurrengo agindua bilatzeko.

«

7.3. Ariketa

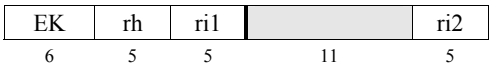
`mul r9, r6, r9` agindua exekutatu behar da BIRD konputagailuan. Egin ezazu aginduaren exekuzioa (deskodeketa fasetik hasita) erakusten duen kronograma bat. Azal itzazu kronograman aginduaren exekuzioarekin zerikusia duten kontrol-seinale guztiak eta gailu guztien edukia.

Hartu kontuan hasiera-datu hauek: `r9` = 6 (0006H) eta `r6` = –4 (FFFCH).

■

`mul r9, r6, r9` aginduaren formatua honako hau da (ikus 7.6. irudia):

`op rh, ri1, ri2`



EK → 001101 (`mul`)
rh → 01001 (r9); ri1 → 00110 (r6); ri2 → 01001 (r9);

Hau da, bitarrez: 001101 01001 00110 | 0000 0000 000 01001
edo 16 oinarrian: 3526H 0009H

Aginduaren bilaketa egin eta gero, honako egoera hauetatik igaroko da kontrol-unitatea: `DESK`, `Ir_OP`, `A_OP` eta `Id_OP` (ikus kontrol-algoritmoa 7.15. irudian). Kronograma egiteko, aginduaren exekuzioan parte hartzen duten honako seinale eta datu hauek aukeratu ditugu: IR_1 eta IR_2; Q1, Q2, *LD_EM*, *hMD* eta `r9` erregistroaren edukia; *LD_e*1, *LD_e*2, R_e1, R_e2, *hB*, UALaren irteera, *LD_ ual* eta R_ual.

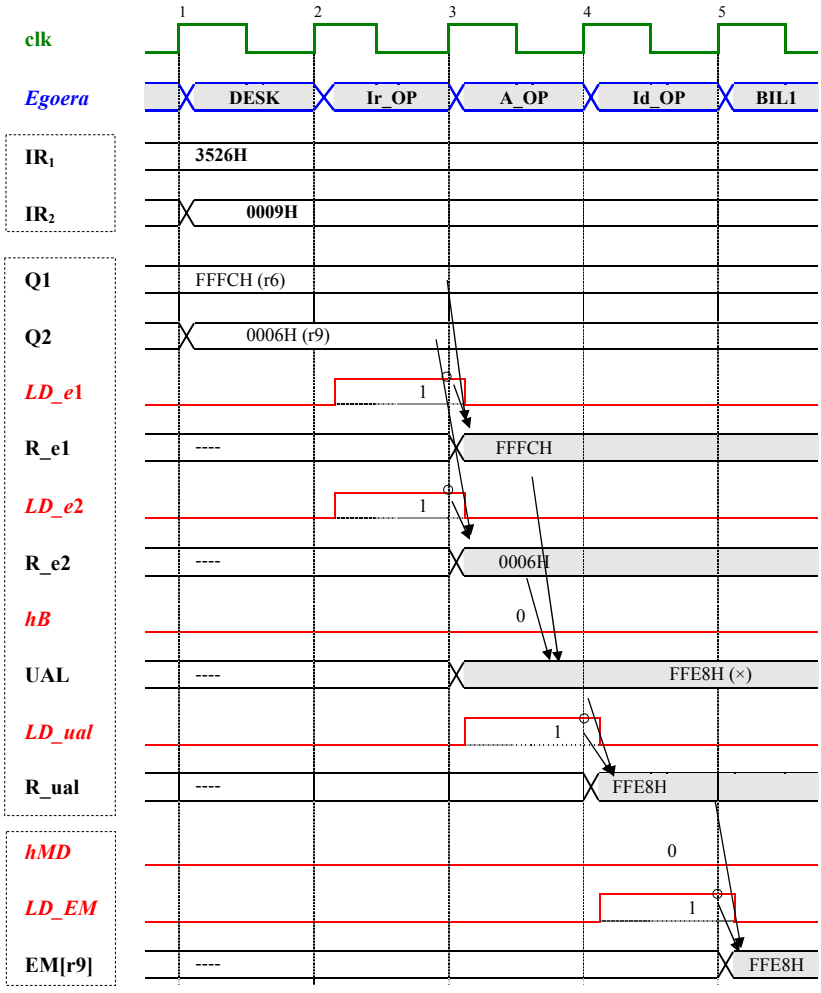
Deskodeketa fasean hasiko gara analisia egiten. IR_1 erregistroan, aginduaren lehen hitza kargatuta dago (3526H), eta bigarrena erloju-ertzarekin batera kargatu da IR_2 erregistroan (0009H); horrez gain, automata `DESK` egoerara igaro da. Ziklo horretan, deskodegailuak `mul` agindua detektatzen du; ondorioz, automata `Ir_OP` egoerara doa.

Egoera horretan, bi eragigaiak R_e1 eta R_e2 laneko erregistroetan kargatzeko seinaleak aktibatzen dira —*LD_e*1, *LD_e*2—. Hurrengo erloju-ertzarekin batera, bi erregistroak kargatuko dira eta automata `A_OP` egoerara igaroko da.

UALean biderketa egingo da (aginduari dagokion kodea ROM taulatik lortzen da, une oro). Emaitza, 6 × (–4) = –24 (FFE8H, ikus E1 eranskina), UALaren irteeran dago, R_ual erregistroan kargatzeko prest —*LD_ual*—.

Erloju-ertza heltzen denean, R_ual erregistroa kargatuko da eta automata `Id_OP` egoerara joango da, emaitza EMan idazteko.

Azken egoeran, *LD_EM* seinalea aktibatzen da, eta erregistro-multzoko datu-sarrerako multiplexorean *hMD* hautatze-seinalea 0 da; beraz, R_ual laneko erregistroaren edukia idatziko da, `r9` erregistroan (helbidea IR_1 erregistrotik hartuta).



7.21. irudia. `mul r9, r6, r9` aginduaren exekuzioaren kronograma.

«

7.4. Ariketa

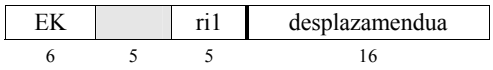
`beq r2, segi` agindua exekutatu behar da BIRD konputagailuan. Egin ezazu aginduaren exekuzioa (deskodeketa fasetik hasita) erakusten duen kronograma bat. Azal itzazu kronograman aginduaren exekuzioarekin zerikusia duten kontrol-seinale guztiak eta gailu guztien edukia.

Hartu hasiera-datu hauek kontuan: aginduaren memoria-helbidea, 2516H; `r2` = 0; `segi` etiketari dagokion desplazamendua, –8 (FFF8H).

■

`beq r2, segi` aginduaren formatua honako hau da (ikus 7.6. irudia):

`beq ri1, etiketa`



EK → 011010 (`beq`)
ri → 00010 (r2)
desplazamendua → 1111 1111 1111 1000 (–8, 2rako osagarrian)

Hau da, bitarrez: 011010 00000 00010 | 1111 1111 1111 1000
edo 16 oinarrian: 6802H FFF8H

Aginduaren bilaketa egin eta gero, honako egoera hauetatik igaroko da kontrol-unitatea: `DESK`, `Ir_BEQ`, `A_BEQ` eta `Id_PC` (ikus kontrol-algoritmoa 7.15. irudian). Kronograma egiteko, aurreko ariketetan egin dugun moduan, aginduaren exekuzioarekin lotura gehien duten seinale eta datuak aukeratu ditugu: IR_1 eta IR_2; Q1, *LD_e*1 eta R_e1; *Z* adierazlea; eta *hD*, batugailuaren sarrerak eta irteera, *LD_PC* eta PC.

Deskodeketa fasean sartu baino lehen, aginduaren lehen hitza (6802H) IR_1 erregistroan kargatu da; fase horretara igarotzean, bigarrena (FFF8H) kargatzen da, IR_2 erregistroan. Ziklo horretan, deskodegailuak `beq` agindua detektatzen du; ondorioz, automata `Ir_BEQ` egoerara doa.

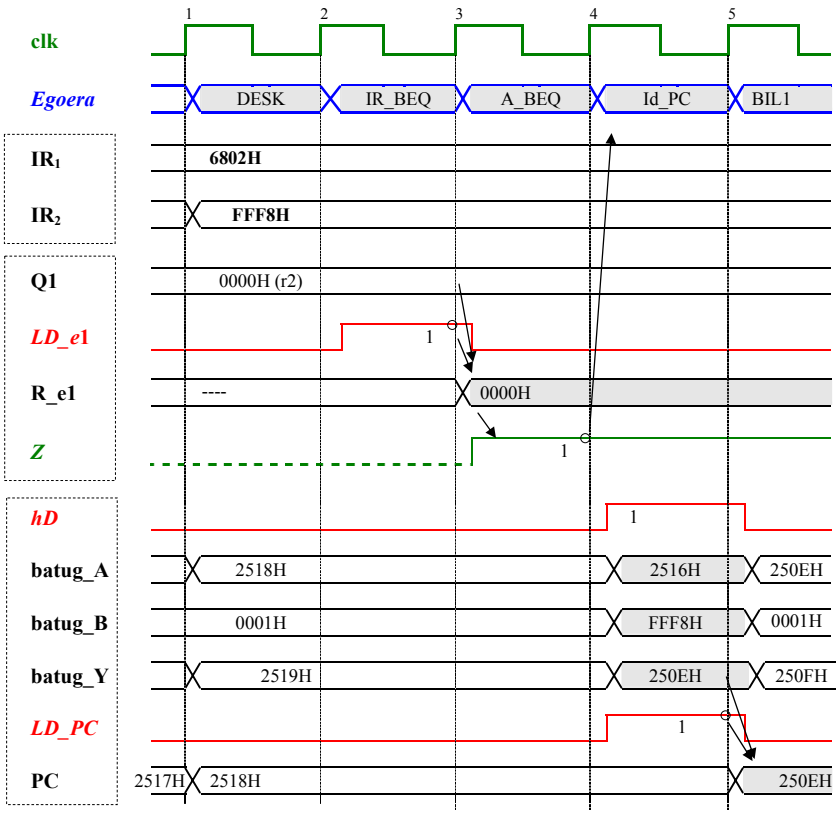
`Ir_BEQ` egoeran, *LD_e*1 seinalea aktibatzen da, erregistro-multzoko Q1 irteera-busean dagoen datua (`r2`) R_e1 laneko erregistroan kargatzeko. Erloju-ertzean, karga betetzen da —R_e1 = 0—, eta automata `A_BEQ` egoerara doa.

Egoera horretan, UALeko A sarrerako datua irteerara pasatzen da (hori baita ROM taulatik hartutako kodea), eta, garrantzitsuena kasu honetan, *Z* adierazlea aktibatzen da, emaitza 0 delako.

Z = 1 denez, automatak `Id_PC` egoerara joatea erabakitzen du, eta hala egiten du erloju-ertza heltzen denean.

Une horretara arte, PCaren batugailuan PC + 1 eragiketa ari zen egiten, baina, orain, beste bat egitera pasatuko da: PCi + `despl`, batugailuaren sarreretako MuxPC eta MuxBat multiplexoreen hautatze-seinaleak 1 direlako —*hD*—.

Horrez gain, *LD_PC* seinalea aktibatu da, agindu berri baten helbidea kargatu behar baita PCan. Beraz, erloju-ertza heltzen denean, PC erregistroaren balioa aldatuko da eta kontrol-unitatea `BIL1` egoerara joango da, jauziari dagokion agindua bilatzera; izan ere, haren helbidea PC erregistroan kargatu da.



7.22. irudia. `beq r2, segi` aginduaren exekuzioaren kronograma.

«

7.5. Ariketa

Programen exekuzioa hobeto ulertzeko, kode zati hau aztertu nahi dugu:

```
        ...
        ld    r2, ALD          ; irakurri ALD aldagaia eta utzi r2 erregistroan
        beq   r2, zero         ; r2 = 0 bada, jauzi egin "zero" etiketa duen agindura
        beq   r0, segi         ; jauzi egin "segi" etiketa duen agindura (r0 = 0)
zero:   mul   r2, r3, r3       ; bestela, egin biderketa
        st    r2, ALD          ; gorde emaitza ALD aldagaian
segi:   ...
```

Adieraz itzazu, zikloz ziklo, taula batean, BIRD prozesadorearen erregistroen eta gailu nagusien edukia programa zati hori exekutatzen denean. Hartu kontuan datu hauek: `ld` aginduaren memoria-helbidea, 2500H; `ALD` aldagaiaren helbidea eta balioa, 0100H eta 0; `r3` erregistroaren edukia, 4.

■

Programa zati horren aginduak exekutatzeko, kontrol-unitatea honako egoera hauetatik igaroko da:

`ld` → `BIL1 – BIL2 – DESK – M/Id_LD`

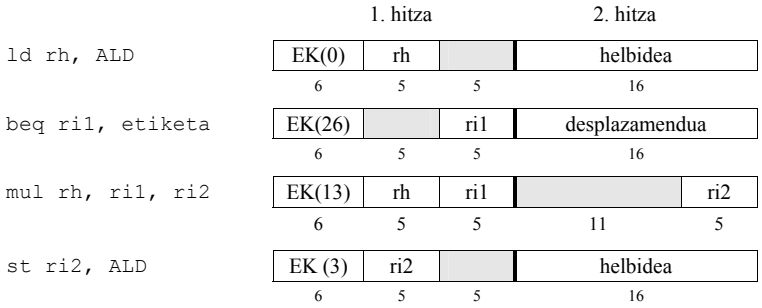
`beq` → `BIL1 – BIL2 – DESK – Ir_BEQ – A_BEQ – Id_PC`

`r2` erregistroan kargatuko den balioa 0 denez (`ALD` aldagaiaren balioa hain zuzen ere), hurrengo agindua `zero` etiketa duena izango da.

`mul` → `BIL1 – BIL2 – DESK – Ir_OP – A_OP – Id_OP`

`st` → `BIL1 – BIL2 – DESK – Ir_ST – M_ST`

Lau agindu horiek honela kodetzen dira BIRD prozesadorean (ikus 7.6. irudia):



Beraz,

```
ld   → 000000 00010 00000 | 0000 0001 0000 0000  → 0040H 0100H
beq  → 011010 00000 00010 | 0000 0000 0000 0100  → 6802H 0004H
mul  → 001101 00010 00011 | 0000 0000 0000 0011  → 3443H 0003H
st   → 000011 00010 00000 | 0000 0001 0000 0000  → 0C40H 0100H
```

Zikloz zikloko exekuzioa adierazteko, kronogramak erabili ditugu aurreko ariketetan; antzeko informazioa eman daiteke, baita ere, taula batean. Esaterako taula hau, non BIRD konputagailuaren gailu nagusien edukia ageri den: automataren egoera, PCi eta PC erregistroak, memoria-helbidea eta eragiketa, IR erregistroak, EM[r2] erregistroa eta laneko hiru erregistroak.

Programaren lau aginduen exekuzioa ageri da taulan, egoeraz egoera. Gogoratu: gailu sinkronoen gainean egiten diren eragiketak (erregistro baten karga, esaterako) erloju-ertza heltzen denean exekutatzen dira, hau da, zikloaren bukaeran (hurrengoaren hasieran); edukia, beraz, hurrengo zikloan ikusiko da.

	egoera	PCi	PC	@MEM	MEM erag	IR_1	IR_2	EM[r2]	R_e1	R_e2	R_ual
ld	BIL1	–	**2500H**	**2500H**	rd	–	–	–	–	–	–
	BIL2	2500H	2501H	**2501H**	rd	**0040H**	–	–	–	–	–
	DESK	2500H	2502H	2502H	–	0040H	**0100H**	–	–	–	–
	M/Id_LD	2500H	2502H	**0100H**	rd	0040H	0100H	–	–	–	–
beq	BIL1	2500H	**2502H**	**2502H**	rd	0040H	0100H	**0**	–	–	–
	BIL2	**2502H**	2503H	**2503H**	rd	**6802H**	0100H	0	–	–	–
	DESK	2502H	2504H	2504H	–	6802H	**0004H**	0	–	–	–
	Ir_BEQ	2502H	2504H	2504H	–	6802H	0004H	0	–	–	–
	A_BEQ	2502H	2504H	2504H	–	6802H	0004H	0	**0**	–	–
	Id_PC	2502H	2504H	2504H	–	6802H	0004H	0	0	–	–
mul	BIL1	2502H	**2506H**	**2506H**	rd	6802H	0004H	0	0	–	–
	BIL2	2506H	2507H	**2507H**	rd	**3443H**	0004H	0	0	–	–
	DESK	2506H	2508H	2508H	–	3443H	**0003H**	0	0	–	–
	Ir_OP	2506H	2508H	2508H	–	3443H	0003H	0	0	–	–
	A_OP	2506H	2508H	2508H	–	3443H	0003H	0	**4**	**4**	–
	Id_OP	2506H	2508H	2508H	–	3443H	0003H	0	4	4	**16**
st	BIL1	2506H	**2508H**	**2508H**	rd	3443H	0003H	**16**	4	4	16
	BIL2	2508H	2509H	**2509H**	rd	**0C40H**	0003H	16	4	4	16
	DESK	2508H	250AH	250AH	–	0C40H	**0100H**	16	4	4	16
	Ir_ST	2508H	250AH	250AH	–	0C40H	0100H	16	4	4	16
	M_ST	2508H	250AH	**0100H**	wr	0C40H	0100H	16	4	**16**	16

Analiza dezagun lehenengo aginduaren exekuzioa. `BIL1` egoeran, PCaren edukia 2500H da; memoria-posizio horretan dago aginduaren lehen hitza, ziklo horretan irakurtzen ari dena. IR_1 erregistroa kargatzeko seinalea aktibatuta dago; beraz, hurrengo egoerara igarotzean, aginduaren lehen hitza (0040H) IR_1 erregistroan izango da. Era berean, PC eta PCi erregistroak kargatzeko seinaleak aktibatuta daude: PCan, PC + 1 = 2501H kargatzeko; eta PCi laneko erregistroan, PCaren balioa (2500H).

`BIL2` egoeran, aginduaren bigarren hitza (PC = 2501H) bilatzen da memorian; hurrengo erloju-ertzean, IR_2 erregistroan kargatuko da (0100H). PC erregistroa eguneratzeko seinalea ere sortu da; beraz, bukaeran PC + 1 = 2502H kargatuko da PCan.

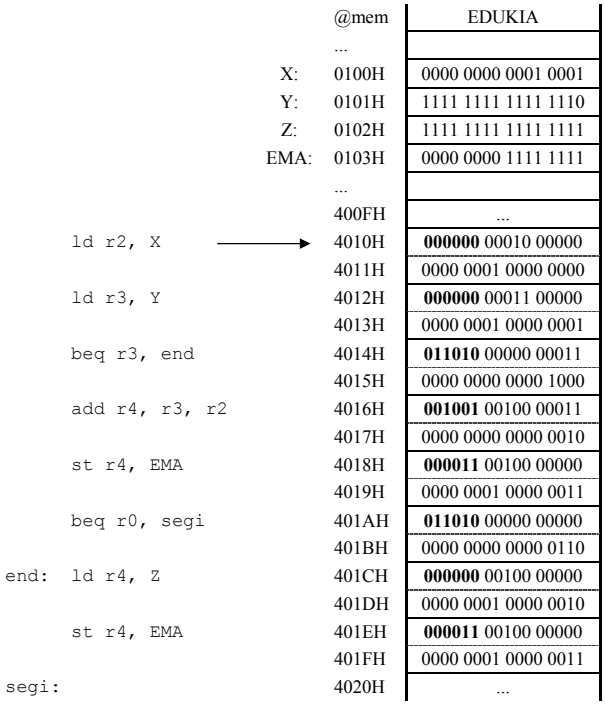
`DESK` fasean ez da seinalerik sortzen; beraz, ez da ezer aldatuko (hasieran kargatu diren IR_2 eta PC erregistroez gain). Azkenik, `M/Id_LD` egoeran, memoria irakurtzen da, baina IR2 erregistroan dagoen helbidea erabiliz; hori dela eta, memoria-helbidea 0100H da. Emaitza (0) `r2` erregistroan utzi behar da, eta horretarako *LD_EM* seinalea sortu da. Karga hori exekutatuko da hurrengo aginduaren `BIL1` egoerara igarotzean.

Gainerako aginduen exekuzioaren analisia antzekoa da, eta irakurlearentzat uzten dugu.

«

7.6. Ariketa

Irudian, programa zati bat eta dagokion memoria zatia ageri dira.



Programa hori kontuan hartuz, erantzun honako galdera hauei:

(a) Zenbat erloju-ziklo behar dira programa exekutatzeko?

(b) BIRD prozesadorearen kontrol-unitatea `Id_OP` *egoeran dago. Zer balio dute une horretan PC, R_e1, R_e2 eta R_ual erregistroek? Eta zer balio dago unitate aritmetiko/logikoaren irteeran? Eta erregistro-multzoko datu- eta helbide-sarreretan?*

(c) Kontrol-unitatea `M_ST` *egoeran dago, eta PCaren edukia* `401AH` *da. Zein da IR_1 eta IR_2 erregistroen edukia une horretan? Eta memoriako helbide-sarrera? Eta memoriako datu-sarrera?*

(d) Kontrol-unitatea `Id_PC` *egoeran dago. Zein agindu ari da exekutatzen? Zein da PCi laneko erregistroaren edukia? Zer dago PCaren datu-sarreran?*

(e) PCaren edukia `4011H` *denean, zein egoeratan dago kontrol-unitatea?*

■

(a) atala

Programaren exekuzioak iraungo duen ziklo kopurua kalkulatu ahal izateko, batetik, zein agindu exekutatuko diren jakin behar dugu, aldagaiek hartzen dituzten balioen arabera, eta, bestetik, agindu bakoitzak exekutatzeko behar duen ziklo kopurua.

Hartu kontuan programa zatia:

```
        ...
        ld  r2, X
        ld  r3, Y
        beq r3, end

        add r4, r3, r2
        st  r4, EMA
        beq r0, segi

end:    ld  r4, Z
        st  r4, EMA
segi:   ...
```

Bi jauzi daude programa horretan; beraz, exekutatuko diren aginduak jauzi horien araberakoak izango dira. Lehenengo jauzian, `r3` konparatzen da 0rekin. Aurreko aginduan `Y` aldagaia kargatu da `r3` erregistroan. Memoriaren edukiari begiratuz gero, ikusten da `Y` = –2 dela (1111 1111 1111 1110). Beraz, jauzia ez da egingo, eta ondoz ondoko aginduarekin jarraituko du programak. Bigarren jauzia, ordea, beti egingo da, `r0` erregistroaren edukia 0 delako.

Ondorioz, honako 6 agindu hauek exekutatuko dira:

```
        ld  r2, X           → 4 ziklo
        ld  r3, Y           → 4 ziklo
        beq r3, end         → 5 ziklo
        add r4, r3, r2      → 6 ziklo
        st  r4, EMA         → 5 ziklo
        beq r0, segi        → 6 ziklo

segi:   ...
```

Programa exekutatzeko behar den ziklo kopurua kalkulatzeko, agindu bakoitzak behar duen ziklo kopurua kontatu beharko dugu, kontrol-algoritmoaren arabera. 7.3.4.2. azpiatalean kalkulatu dugu ziklo kopurua agindu mota bakoitzeko (aginduekin batera ageri dira aurreko programan).

Beraz, ziklo kopurua, guztira: 4 + 4 + 5 + 6 + 5 + 6 = 30 ziklo.

(b) atala

Automata `Id_OP` egoeran badago, exekutatzen ari den agindua `op` motakoa izango da. Gure kasuan, mota horretako agindu bakarra dago programan: `add r4, r3, r2.`

Agindu horren PCa 4016H da. Baina, gogoratu: bilaketa fasean eguneratu egiten da PCaren edukia; beraz, egoera horretan PCaren edukia hau izango da: 4016H + 2 = 4018H (hurrengo aginduarena, alegia).

Aginduaren exekuzioaren azken zikloan gaude. Beraz, R_e1 eta R_e2 laneko erregistroetan, aginduaren bi eragigaiak izango dira —`r3` eta `r2`— `Ir_OP` egoeratik `A_OP` egoerara igarotzean kargatu baitira; era berean, R_ual erregistroan, UALaren emaitza izango dugu, kargatu baita `A_OP` egoeratik `Id_OP` egoerara igarotzean. `r3` erregistroan, `Y` aldagaiaren balioa izango dugu, eta `r2` erregistroan, `X` aldagaiarena. Beraz,

R_e1 = –2; R_e2 = 17; R_ual = 15;

UALa, oraindik, batuketa egiten ari da, eragiketa adierazten duen ROM taulatik batuketarako kodea heltzen delako, aginduaren exekuzio osoan zehar IR_1 erregistroko eragiketa-kodea ez baita aldatu. Beraz, 15 balioa izango dugu UALaren irteeran.

Azkenik, erregistro-multzoko datu-sarreran R_ual laneko erregistroaren edukia izango da, `Id_OP` egoeran sarrerako MuxEM multiplexoreko hautatze-seinaleak, *hMD*-k, 0 balio duelako. Helbide-sarreretan, honako hau izango dugu: @h → 00100 (idatzi behar den erregistroa, IR_1 erregistroko tarteko 5 bitak); @i1 → 00011 (IR_1 erregistroko pisu txikieneko 5 bitak); @i2 → 00010 (IR_2 erregistroko pisu txikieneko 5 bitak, MuxI2 multiplexoreko *hH* hautatze-seinalea 0 delako).

(c) atala

`M_ST` egoera `st` aginduaren exekuzioari dagokio, baina programak bi `st` agindu ditu. PC = 401AH bada, ez dago dudarik: lehendabiziko `st r4, EMA` agindua ari da exekutatzen. Agindu horren memoria-helbidea 4018H da, eta PC eguneratu egin da bilaketa fasean, hurrengo agindua erakusteko; horregatik, PC = 401AH da une horretan.

Beraz, exekutatzen ari den agindua dago IR erregistroetan:

IR_1 → 000011 00100 00000 (0C80H)

IR_2 → 0000 0001 0000 0011 (0103H)

`M_ST` egoeran, memoria idatzi behar da. `st` aginduaren kasuan, helbideratze absolutua erabiltzen denez, aginduan bertan adierazi da idatzi behar den aldagaiaren memoria-helbidea; beraz, IR_2 erregistrotik hartuko da (ikus aginduen formatua 7.6. irudian). Hau da, @MEM = 0103H (`EMA` aldagaia) da.

Era berean, datu-sarreran R_e2 laneko erregistroaren edukia izango dugu (ikus 7.14. irudia); hau da, `r4` erregistroaren edukia (`Ir_ST` fasean kargatu da R_e2 erregistroan), kasu honetan aurreko batuketaren emaitza: 15.

(d) atala

`Id_PC` egoera `beq` aginduaren exekuzioari dagokio. Izan ere, egoera horretan eguneratzen da PCaren balioa jauzia egiteko. Bi jauzi-agindu daude programan, baina lehenengoa ez da egoera horretara iristen, $Z = 0$ delako. Beraz, bigarren jauzia ari da exekutatzen: `beq r0, segi`.

PCi erregistroan, aginduaren lehen hitzaren helbidea gordetzeko kontrol-seinalea sortzen da `BIL1` fasean, eta ez da berriro aldatuko. Beraz, une horretan PCi = 401AH da.

Jauzia egin behar da, eta horrek eskatzen du jauziaren helburu-helbidea —PCi + `despl`— kalkulatzea eta PCan gordetzea. Ziklo horretan, batugailuaren sarreretan dauden bi multiplexoreetan, 1 sarrerak aukeratu dira —$hD = 1$—; beraz, batugailuaren irteera 401AH + 0006H = 4020H izango da (`segi` etiketa duen aginduaren helbidea, alegia).

(e) atala

BIRD prozesadoreko aginduak bi hitzekoak dira. Aztertzen ari garen programan, agindu guztiak helbide bikoitietan hasten dira; beraz, PCren edukia 4011H bada (zenbaki bakoitia) agindu baten bilaketaren erdian gaude: lehen hitza irakurri da eta PC erregistroa eguneratu da bigarren hitza irakurtzeko. Hortaz, kontrol-unitatea `BIL2` egoeran dago.

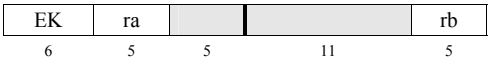
«

7.7. Ariketa

BIRD prozesadorea zabaldu nahi dugu agindu berri bat exekutatu ahal izateko:

```
swap ra,rb        →      ra := rb; rb := ra;
```

Aginduak trukatu behar ditu adierazitako bi erregistroen edukiak. Aginduaren formatua honako hau da:



Egin ezazu agindu honi dagokion adarra kontrol-algoritmoan (deskodeketa eta gero) eta adierazi prozesu-unitatean egin beharreko aldaketak agindua exekutatu ahal izateko.

■

Prozesadore bati aginduak gehitzeko, lehendabiziko pausoa da analizatzea ea prozesu-unitatean behar diren osagaiak eta haien arteko loturak daudenetz.

Modu asko dago `swap` aginduak eskatzen duen datu-trukea egiteko. Aukeran, prozesadoreak duen hardwareaz gain besterik behar ez duen irtenbidea aukeratuko dugu. `swap` agindua exekutatzeko, erregistro pare bat irakurri eta idatzi behar dira. BIRD prozesadorean hori egiteko, UALa erabili behar da, memoriatik edo UALetik datozen datuak soilik onartzen direlako EMko datu-sarreran.

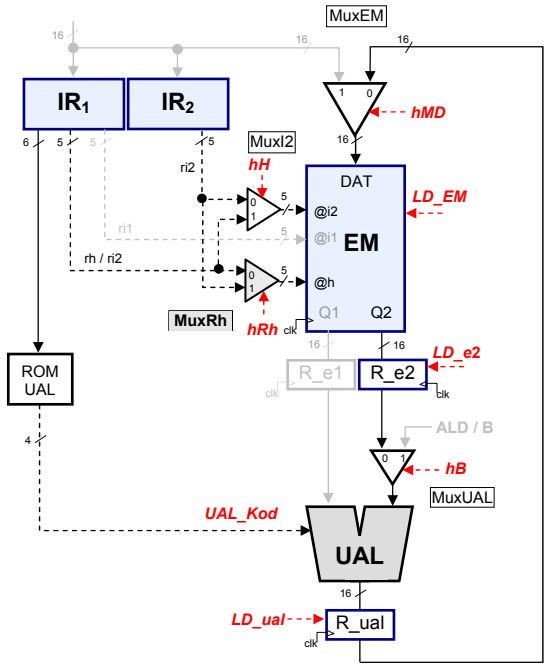
Izan ere, UALak aukera ematen du eragigai bat pasatzen uzteko, $Y = A$ edo $Y = B$ eragiketak exekutatuz. Hala ere, agindu bakoitzari eragiketa bakar bat dagokio UALean, UALeko eragiketa-kodea —*UAL_kod*— ROM taula batetik datorrelako. Demagun $Y = B$ eragiketa aukeratzen dugula erregistroen arteko datu-trukea egiteko.

Hala bada, R_e2 laneko erregistroan utzi behar dugu eragigai bat (`ra`, adibidez); gero, UALetik pasa eta R_ual laneko erregistroan utziko dugu. Azkenik, handik EMko `rb` erregistroan idatziko dugu. Baina, idazketa egin baino lehen, irakurri egin behar da `rb` erregistroa eta R_e2 erregistroan utzi; bestela, galdu egingo genuke haren edukia!

Hori dela eta, EMko @i2 helbide-sarreran, `ra` eta `rb` erregistroen helbideak eman behar dira.

Izan ere, aginduaren formatua dela eta, bi erregistro horien helbideak, IR erregistroetan daudenak, konektatuta daude @i2 sarrerara, MuxI2 multiplexorearen bidez (ikus 7.14. irudia). Horregatik aukeratu dugu R_e2 erregistroa trukea egiteko eta ez R_e1, kasu horretan multiplexore bat gehitu beharko baikenuke @i1 sarreran.

Bestalde, idazketa egiteko bi helburu-helbide erabili behar ditugu: `ra` eta `rb`. Jatorrizko prozesu-unitateak IR_1 erregistroko tarteko 5 bitak onartzen ditu bakarrik; beraz, multiplexore bat gehitu beharko dugu EMko @h helbide-sarreran, IR_1 eta IR_2 erregistroetako helbide-bitak onartu ahal izateko: MuxRh (hautatze-seinalea, *hRH*). Hona hemen nola geratuko den datu-bidea:



Honela egingo dugu datu-trukea:

- `ra` irakurri eta R_e2 erregistroan utzi:

  ```
  R_e2 := EM[ra];
  ```

  Adi! `ra` helbide gisa erabiltzeko, 1 bidea aukeratu behar da MuxI2 multiplexorean ($hH = 1$).

- R_e2 erregistroaren edukia UALetik pasa eta R_ual erregistroan utzi. Gainera, eta R_ual erregistroaren edukia EMan idatzi baino lehen, `rb` erregistroa irakurri beharko dugu eta R_e2 erregistroan kargatu. R_ual eta R_e2 erregistroen karga zikloaren bukaeran egiten denez, posiblea da bi eragiketak ziklo berean egitea, informazioa galdu gabe. Beraz, bigarren zikloan honako hau egingo dugu:

  ```
  R_ual := R_e2;        R_e2 := EM[rb];
  ```

- R_ual erregistroaren edukia EMan idatzi, `rb` erregistroan, eta, bidenabar, R_e2 erregistroaren edukia R_ual erregistrora pasa:

  ```
  EM[rb] := R_ual;      R_ual := R_e2;
  ```

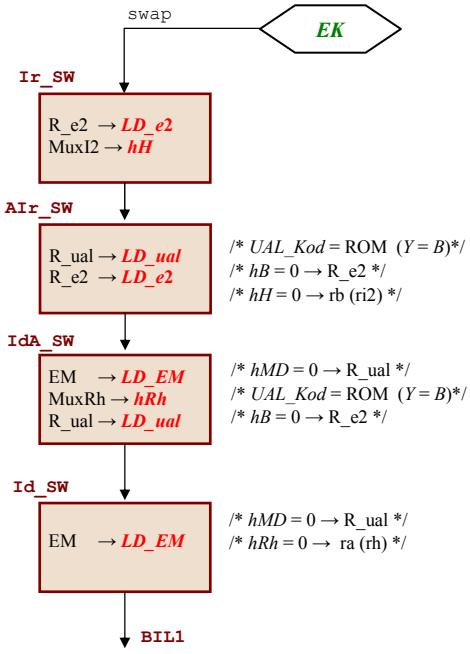
  EMko idazketa egiteko, MuxRh multiplexorean 1 bidea aukeratu behar dugu, `rb` erregistroaren helbidea aukeratzeko: $hRh = 1$.

- Azkenik, R_ual erregistroaren edukia EMan idatzi, `ra` erregistroan:

  ```
  EM[ra] := R_ual;
  ```

  Idazketa hau egiteko, ordea, $hRh = 0$ behar dugu, `ra` erregistroaren helbidea aukeratzeko.

Hona hemen kontrol-algoritmorako proposamena:



Aurreko irtenbidea aukera bat baino ez da; nahi izanez gero, irakurleak beste batzuk proposa ditzake.

Ikus dezagun aginduaren exekuzioko adibide bat, zikloz ziklo. Demagun `swap` aginduaren eragiketa-kodea 63a dela (111111). Beraz, honela kodetuko da, esaterako, `swap r3, r4` agindua:

EK → 111111
`ra` → 00011 (`r3`)
`rb` → 00100 (`r4`)

Hau da, bitarrez: 111111 00011 00000 | 0000 0000 0000 0100

edo 16 oinarrian: FC60H 0004H

Erregistroen hasierako balioak: `r3` → 5 eta `r4` → 18.

Honela aldatuko dira aginduaren exekuzioan parte hartzen duten BIRD prozesadoreko gailuetako batzuk:

	egoera	IR_1	IR_2	@i2	@h	EM[r3]	EM[r4]	R_e2	R_ual
`swap`	`Ir_SW`	**FC60H**	**0004H**	**3**	(3)	5	18	–	–
	`AIr_SW`	FC60H	0004H	**4**	(3)	5	18	**5**	–
	`IdA_SW`	FC60H	0004H	(4)	**4**	5	18	**18**	**5**
	`Id_SW`	FC60H	0004H	(4)	**3**	5	**5**	18	**18**
	`BIL1`	FC60H	0004H	(4)	(3)	**18**	5	18	18

`Ir_SW` egoeran, `r3` erregistroaren edukia irakurtzen da; hurrengo egoerara igarotzean, R_e2 laneko erregistroan utziko da.

`AIr_SW` egoeran, bi gauza egiten dira: pasatu R_e2-ren edukia R_ual erregistrora, eta irakurri `r4` erregistroaren edukia, R_e2 erregistroan gordetzeko. Bi eragiketak erloju-ertza heltzean exekutatuko dira.

`IdA_SW` egoeran, R_ual erregistroaren edukia EMan (`r4` erregistroan) idazteko kontrol-seinalea sortzen da: idazketa hurrengo zikloaren hasieran gertatuko da. Era berean, R_e2-n utzi dena R_ual-era pasatuko da.

Azkenik, `r3` erregistroan idatziko da R_ual-ean utzi den datua.

«

7.8. Ariketa

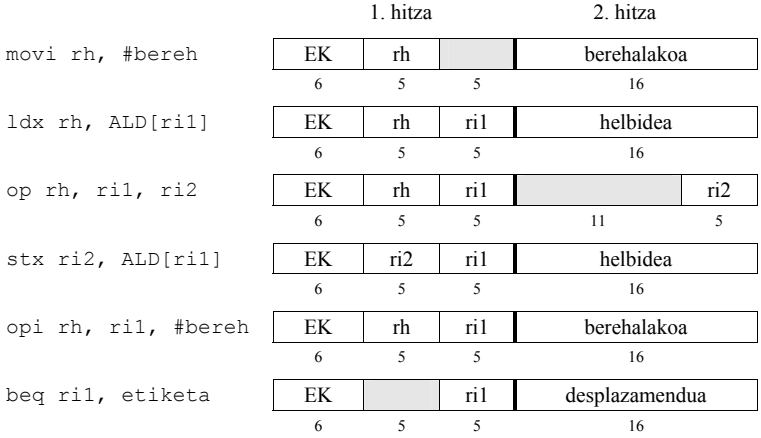
`C = B + A` bektore-eragiketa egiteko, honako programa hau idatzi dugu (ikus 7.3.2. atala):

```
        movi  r1, #0          ; indize-erregistroa hasieratu
        movi  r2, #16         ; bektoreen osagai kopurua
segi:   ldx   r3, A[r1]       ; irakurri A bektorearen osagai bat
        ldx   r4, B[r1]       ; irakurri B bektorearen osagai bat
        add   r5, r4, r3      ; batu bi osagaiak (emaitza, r5)
        stx   r5, C[r1]       ; gorde emaitza C bektorearen osagaian
        addi  r1, r1, #1      ; inkrementatu r1, bektoreen hurrengo osagaiaren indizea
        subi  r2, r2, #1      ; osagai bat gutxiago geratzen da
        beq   r2, buka        ; r2 = 0 bada, eragiketa bukatu da
        beq   r0, segi        ; bestela, segi eragiketarekin (r0 = 0)
buka:   ...
```

7.6. irudiko informazioa erabiliz, idatzi bitarrez nola geratuko den programa BIRD konputagailuaren memorian. Programaren hasiera-helbidea, 1000H; bektoreen hasiera-helbideak, `A` → 0100H; `B` → 0200H; `C` → 0300H.

■

BIRD prozesadorearen aginduak 32 bitekoak dira, hau da, bi hitz okupatzen dituzte memorian. Programan ageri diren aginduen formatua honako hau da (7.6. iruditik hartuta):



Era berean, hauek dira aginduen eragiketa-kodeak:

`movi` → 8; `ldx` → 2; `add` → 9; `stx` → 5; `addi` → 10; `subi` → 12; `beq` → 26

Aginduen formatuan ageri den moduan, bit batzuk ez dira erabiltzen zenbait kasutan; kasu horietan, 0koak idatziko ditugu memorian.

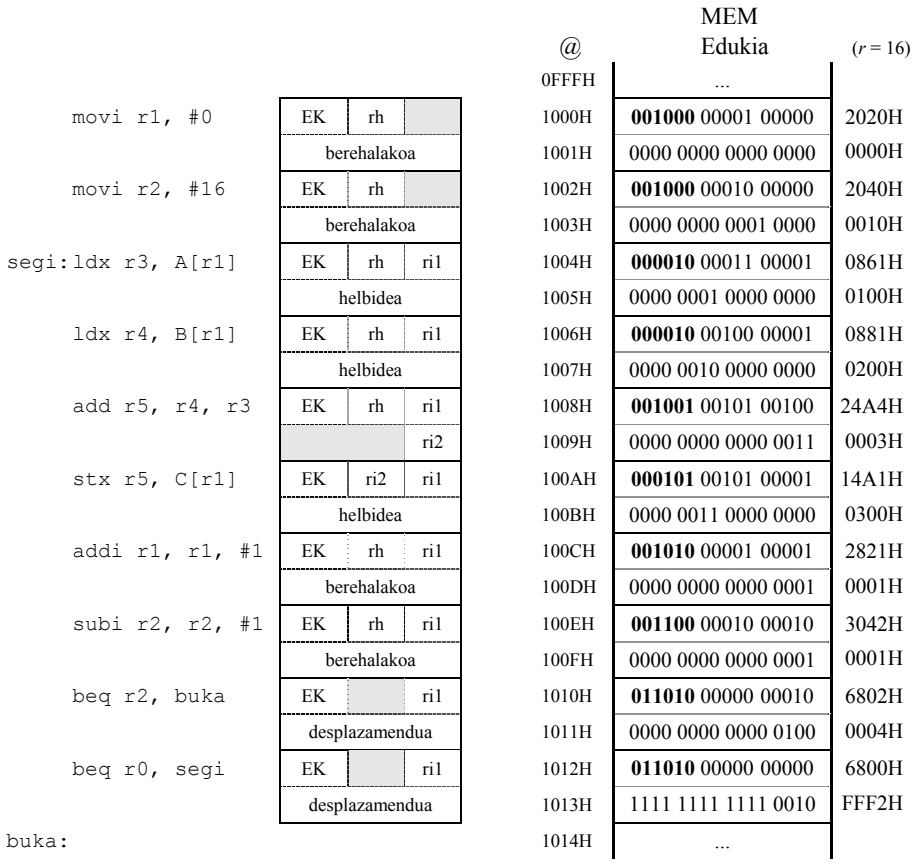
Egin dezagun lehen aginduaren kodeketa: `movi r1, #0`

Lehen hitzaren lehen 6 bitak eragiketa-kodea dira; `movi` (8) → 001000
Gero, 5 bitetan, helburu-erregistroa adierazten da, `r1` → 00001
Lehen hitzaren beste 5 bitak ez dira erabiltzen → 00000
Bigarren hitzean berehalako datua adierazten da: 0 → 0000 0000 0000 0000

Beraz, bi hitz hauek izango ditugu memorian:

1000H helbidean: 001000 00001 00000 → 2020H
1001H helbidean: 0000 0000 0000 0000 → 0000H

Aurreko prozedura berari jarraitu behar zaio gainerako aginduekin. Hona hemen programa osoari dagokion memoria-edukia:



`beq` aginduak, jauziak, kodetu behar direnean, etiketak adierazten duen agindura heltzeko desplazamendua kalkulatu behar dugu.

Lehen jauzian, `buka` etiketari dagokion desplazamendua 4 da: 1010H helbidetik 1014H helbidera.

Bigarren jauziaren desplazamendua, ordea, negatiboa da (atzera egiten da jauzia): 1012H helbidetik 1004H helbidera, hau da, –14 (adi! 12H = 18; beraz, kendura 14 da, ez 8). Birako osagarrian kodetu behar dugu desplazamendua, hau da, –14 zenbakia:

$$14 \rightarrow 0000\ 0000\ 0000\ 1110$$
$$-14 \rightarrow 1111\ 1111\ 1111\ 0001 + 1 = 1111\ 1111\ 1111\ 0010 = FFF2H$$

Programa osoa kodetzeko, beraz, 16 biteko 20 hitz (40 byte guztira) erabili behar dira BIRD prozesadorean.

«

7.9. Ariketa

100 osagaiko bi bektoreen biderkadura eskalarra —BE = $\sum (X_i \times Y_i)$— kalkulatzen du programa honek:

```
        ...
        movi  r1, #0          ; indize-erregistroa hasieratu
        movi  r2, #100        ; bektoreen osagai kopurua
        movi  r6, #0          ; batura partziala hasieratu
segi:   ldx   r3, X[r1]       ; irakurri X bektorearen osagai bat
        ldx   r4, Y[r1]       ; irakurri Y bektorearen osagai bat
        mul   r5, r3, r4      ; biderkatu bi osagaiak (r5 := r3 × r4)
        add   r6, r6, r5      ; egin batuketa partziala (r6 := r6 + r5)
        addi  r1, r1, #1      ; inkrementatu r1, bektoreen hurrengo osagaiaren indizea
        subi  r2, r2, #1      ; osagai bat gutxiago geratzen da
        beq   r2, buka        ; r2 = 0 bada, eragiketa bukatu da (jauzi buka-ra)
        beq   r0, segi        ; bestela, segi eragiketarekin (r0 = 0) (jauzi segi-ra)
buka:   st    r6, BE          ; gorde emaitza memorian
        ...
```

(a) Prozesadorearen erlojua 100 MHz-ekoa da. Kalkula ezazu zenbat denbora beharko den eragiketa exekutatzeko, kontrol-algoritmoa 7.15. irudikoa bada.

(b) Erloju-maiztasuna 250 MHz-etara igo daiteke, baina, hala egiten bada, memoria-eragiketek 3 ziklo behar dute. Zenbat aldiz azkarrago exekutatuko da programa aurreko kasuarekin alderatuta?

■

(a) atala

7.15. irudiko kontrol-algoritmotik atera daiteke agindu bakoitzak behar duen ziklo kopurua:

`op` (`mul`, `add`) → 6 ziklo

`opi` (`movi`, `addi`, `subi`) → 6 ziklo

`ldx` → 6 ziklo

`beq` → 5 ziklo jauzia ez bada egiten eta 6 ziklo egiten bada

`st` → 5 ziklo

100 aldiz exekutatzen den begizta bat da programa. Iterazio bakoitzean, bi bektoreen osagai bana irakurri, biderkatu eta aurreko emaitza partzialarekin batzen da. Emaitza, beraz, `r6` erregistroan geratuko da begiztaren bukaeran.

Iterazio bakoitzean, 1 kentzen zaio `r2` erregistroaren edukiari (`subi`). Hasieran 100 zenbakia kargatu denez erregistro horretan, bektoreen osagai guztiak prozesatu direnean, 0ko bat geratuko da.

Beraz, `beq r2, buka` jauzia ez da egingo 99 iteraziotan; azken iterazioan, ordea, bai.

Exekutatzen denean, `beq r0, segi` jauzia beti egiten da (99 aldiz adibide honetan), `r0` erregistroa 0 delako.

Programa exekutatzeko behar den ziklo kopurua, beraz, hau da:

- begiztan sartu baino lehen (3 `movi`):

  $3 \times 6 = 18$ ziklo

- begiztaren barruan (2 `ldx`, 4 `op`/`opi`, 2 `beq`):

  99 aldiz → $2 \times 6 + 4 \times 6 + 5 + 6 = 47$ ziklo → 4653 ziklo

  azken iterazioa → $2 \times 6 + 4 \times 6 + 6 = 42$ ziklo

- azken agindua:

  5 ziklo

Guztira, $18 + 4653 + 42 + 5 = 4718$ ziklo

Erloju-maiztasuna 100 MHz bada, periodoa honako hau izango da:

$$T = 1 / f = 1 / 10^8 \text{ s} = 10^{-8} \text{ s} = 10 \text{ ns}$$

Denbora osoa → 4718×10 ns = 47180 ns = **47,18 µs**

(b) atala

Erloju-maiztasuna igotzen dugu, 250 MHz-etara. Erloju-zikloa, beraz, txikiagoa izango da, 4 ns hain zuzen ere. Baina denbora hori txikiegia omen da memoria-eragiketa bat egiteko, eta, horregatik, memoria-eragiketek ziklo bat baino gehiago behar dute, 3 ziklo hain zuzen ere.

Hortaz, agindu bakoitzak behar duen ziklo kopurua aldatu da (ikus 7.15. irudia):

`op` (`mul`, `add`) → 3 (`BIL1`) + 3 (`BIL2`) + 1 + 3 = 10 ziklo

`opi` (`movi`, `addi`, `subi`) → berdin, 10 ziklo

`ldx` → 3 (`BIL1`) + 3 (`BIL2`) + 1 + 2 + 3 (`M/Id_LD`) = 12 ziklo

`beq` → 3 (`BIL1`) + 3 (`BIL2`) + 1 + 2 = 9 ziklo (edo 10 jauzia egiten bada)

`st` → 3 (`BIL1`) + 3 (`BIL2`) + 1 + 1 + 3 (`M_ST`) = 11 ziklo

Orain, beraz, hau da programa exekutatzeko behar den ziklo kopurua:

- begiztan sartu baino lehen:

  $3 \times 10 = 30$ ziklo

- begiztaren barruan:

  99 aldiz $\rightarrow 2 \times 12 + 4 \times 10 + 9 + 10 = 83$ ziklo $\rightarrow 8217$ ziklo

  azken iterazioa $\rightarrow 2 \times 12 + 4 \times 10 + 10 = 74$ ziklo

- azken agindua:

  11 ziklo

Guztira, $30 + 8217 + 74 + 11 = 8332$ ziklo

Denbora osoa 8332×4 ns = 33328 ns = **33,33 µs**

Hau da, bigarren kasuan 47,18 / 33,33 = 1,42 aldiz azkarragoa da.

«

7.5. ARIKETAK

7.1. Jauzi-agindu bakarra erabili dugu diseinatu dugun prozesadorean: `beq`, jauzi baldin 0. Antzeko beste jauzi batzuk erabili ohi dira prozesadoreetan. Esaterako, BIRD konputagailuan beste hauek ere erabiltzen dira:

```
jmp etiketa          ; pc := pc + despl
bne ri1, etiketa     ; baldin(ri1 ≠ 0) pc := pc + despl
bls ri1, etiketa     ; baldin(ri1 < 0) pc := pc + despl
```

Zabal ezazu kontrol-algoritmoa hiru jauzi horiek exekutatu ahal izateko. Aldatu behar da prozesu-unitatea jauzi horiek exekutatu ahal izateko?

7.2. 8000H helbidetik aurrera memorian gorde den programa zati hau exekutatzen da BIRD konputagailuan:

8000H → 0861H / 1000H / 3063H / 0001H / 1461H / 1000H

Zein agindu exekutatzen dira? Nola aldatuko dira erregistro-multzoko erregistroak eta memoria? Zenbat ziklo beharko dira kode zati hori exekutatzeko?

7.3. `beq` agindua exekutatzen denean BIRD konputagailuan, hiru aldiz alda daiteke PC erregistroaren balioa. Esan zein unetan eta zertarako egiten diren aldaketa horiek.

7.4. Zergatik erabili behar izan dugu multiplexore bat EMko @i2 helbide-sarreran?

7.5. R_e1, R_e2 eta R_ual laneko erregistroak erabiltzea prozesu-unitatean erosoa da, baina, prozesadore guztietan ohikoak badira ere, ez dira guztiz beharrezkoak BIRD prozesadorean. Adierazi nola exekuta zitekeen `add` agindua laneko erregistroak erabili gabe. Zenbat ziklo beharko lirateke?

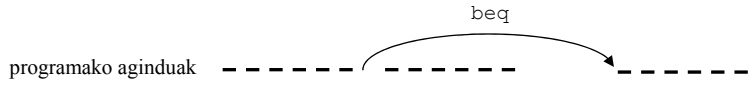
7.6. PCi erregistro laguntzailea erabili dugu exekutatzen ari den aginduaren helbidea gordetzeko, PC erregistroaren balioa aldatzen delako aginduaren bilaketa fasean, hurrengo agindua erakusteko. Hala, desplazamendu bat gehitu behar denean jauzi bat egiteko, PCi laneko erregistroaren edukia erabiliko dugu, hor gorde baitugu jauzi-aginduaren helbidea.
Proposatu beste soluzio bat problema horretarako, baina erregistro laguntzailea erabili gabe.

7.7. Programa zati hau exekutatzen du BIRD prozesadoreak:

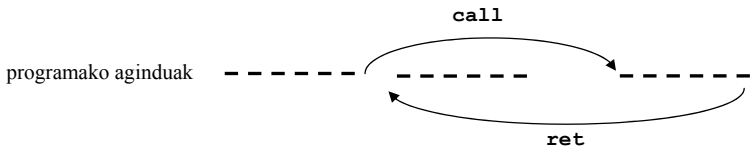
```
baldin (X = Y)                          ...
   orduan   BEKT := Y                   ld   r2, X
   bestela  BEKT := Y + 1               ld   r3, Y
                                        sub  r4, r3, r2
                                        beq  r4, etik1
                                        addi r3, r3, #1
                                 etik1: stx  r3, BEKT[r0]
                                        ...
```

(a) Lehen aginduaren memoria-helbidea 4500H da. Adierazi, zikloz ziklo, PC erregistroak hartuko dituen balio guztiak programa exekutatzean, bi kasu hauetan: (1) X = 4 eta Y = 6; eta (2) X = –2 eta Y = –2.
(b) Bete ezazu taula bat aurreko (1) kasurako (7.5. ariketan egin den moduan), programaren exekuzioaren ondorioz erregistroen eta gainerako gailuen edukiak egoeraz egoera nola aldatzen diren agerian uzteko. Hartu kontuan helbide hauek: @X = 1000H; @Y = 1001H; @BEKT = 1500H

7.8. Bi motatako jauziak bereizten dira konputagailu guztietan: itzulera gabekoak eta itzuleradunak. Lehenengoak sinpleak dira: jauzi egiten da programaren beste toki batera, hango aginduak exekutatzeko, eta hortik aurrera jarraituko da programaren exekuzioa. BIRD prozesadoreko beq jauzia mota horretako jauzi bat da.



Bigarrenak, aldiz, konplexuagoak dira: jauzia egiteaz gain, itzulera-helbidea gordetzen dute, eskuarki erregistro berezi batean. Horrela, agindu batzuk exekutatu eta gero, programa gauza da jauzia egin den puntura itzultzeko, hango aginduen exekuzioarekin jarraitzeko. Jauzi horiek erabili behar dira, esaterako, azpiprogramak edo prozedurak exekutatzeko.



Bi agindu erabili ohi dira horretarako: call, jauzia egiteko, eta ret, itzultzeko. Hartu kontuan bi agindu berrien definizio hauek:

```
call etiketa   → pc := pc + despl; EM[r31] := pc;
ret            → pc := EM[r31];
```

Definizioetan ageri denez, erregistro-multzoko `r31` erregistroan gordetzen da itzulera-helbidea; hortik hartzen du `ret` aginduak jauzia egin den puntura itzultzeko helbidea. Beraz, erregistroa erreserbatu egin behar da eragiketa horretarako.

Gehitu BIRD prozesadoreari `call` eta `ret` aginduak. Analizatu agindu horiek exekutatzeko prozesu-unitatean behar diren gailuak eta loturak, eta aldatu edo emendatu behar duzun guztia. Horrekin batera, sortu kontrol-algoritmoko adar berriak bi aginduen exekuzioa ondo kontrolatzeko.

7.9. Aginduen eragiketa-kodea deskodetzen ari den bitartean, ohikoa da, hainbat konputagailutan, eragigaiak irakurtzea eta laneko erregistroetan uztea (hori dela eta, `DESK` eta `Ir` faseak `D/Ir` izeneko fase batean biltzen dira).

Aldatu BIRD konputagailuaren kontrol-unitatea bi eragiketa horiek batera egin ahal izateko.

Zein da lortuko den abantaila nagusia eragiketak horrela eginez gero?

AZKEN HITZAK

Logikaren teoriak eta formalismo matematikoak ia mende bat badute ere, sistema digitalen historia laburragoa da. Gaur ezagutzen dugun moduan, 1960ko hamarkadan hasi zen sistema digitalen noranahiko zabaltze-prozesua, zirkuitu integratuen garapenarekin batera. Jakina, lehenago ere bazeuden zirkuitu logikoak, hasieran balbulak erabiliz, eta gero transistoreen bidez; izan ere, lehen konputagailuak 1950eko hamarkadan sortu ziren.

Zaila da azken 40-50 urteetan izandako garapena hemen laburbiltzea, teknologia elektroniko digitalak jasan duen eboluzioaren abiadura ikaragarria izan baita. Hala, transistore bakan batzuk zirkuitu batean integratzetik mila milioi integratzera igaro da; ate logiko sinple batzuk izatetik, prozesadore oso konplexuak izatera.

Teknologiaren eragina sistema digitalen diseinuan ere halakoa izan da. Esaterako, duela hogei urte garrantzitsua zen funtzio logikoen minimizazioa ez da gaur egun, inolaz ere, arazo bat, nahi adina transistore izango baitugu txip bakar batean. Diseinatzailearen esku dauden diseinu-tresnak imajinaezinak ziren duela 20 urte; konputagailua erabiliz, hainbat aplikazio daude sistemak diseinatzeko, diseinuak egiaztatzeko, zirkuitu integratuak programatzeko eta abar. Gainera, tresna horiek ez dira bakarrik goi-mailako zentro teknologikoetan erabiltzen; aitzitik, arruntak eta ohikoak dira arloko edozein enpresatan.

Edozein jakintza-arlotako ezagutzak, ordea, piramide bat osatzen du: oinarrizko gauza sinpleenetatik abiatuta, goi-mailako teknikak eta kontzeptuak lortu arte. Nahitaez, hasieratik abiatu beharko dugu piramideko erpinera helduko bagara. Hori izan da, hain zuzen ere, liburu honen helburua. Urtetan zehar, sistema digitalen diseinuaren kontzeptu nagusiak finkatu egin dira eta beste batzuk bidean geratu dira. Kontzeptu nagusi horiek azaldu ditugu liburuaren kapituluetan zehar.

Sistema digital sinkronoak baino ez ditugu tratatu liburuan, zeinetan kontrol-seinale nagusi batek, erlojuak, sistema osoa kontrolatzen baitu, erregistroen aldaketak noiz gertatu behar diren adieraziz. Sistema digital asinkronoak (erlojurik gabekoak) ere badaude, eta garrantzitsuak dira, baina ez ditugu aintzat hartu liburuan.

Sistema digitalen oinarrizko osagaien portaera logikoa azaldu dugu (kasu orokorrak bakarrik, alboan utziz kasu partikularrak), hala konbinazionalak

nola sekuentzialak. Portaera azaltzeaz gain, portaera hori kontrolatzen duten kontrol-seinaleak azaldu ditugu; kontrol-seinale horien erabilera egokia nahitaezkoa da sistema digital zuzenak eta, batik bat, fidagarriak sortzeko.

Hori lortzeko, sistema digital guztien funtsezko banaketa landu dugu: kontrol-unitatea eta prozesu-unitatea. Kontrol-unitateak adierazten dio prozesu-unitateari, kontrol-seinaleen bidez, zer egin behar duen, eta prozesu-unitateak itzuliko dio kontrol-unitateari egindakoari buruzko informazioa. Kontrol-unitatea izango da, beraz, edozein sistema digitalen muina, berari baitagokio, logika zuzen bat erabiliz, prozesu-unitateko gailuek behar dituzten kontrol-seinaleak ordena egokian sortzea.

Nolanahi ere den, sistema digitalen diseinuaren hastapenak baino ez ditugu jorratu: osagai nagusiak (prozesadoreak izan ezik) eta kontrol-unitateen garapena.

Gaur egun, konputagailuz lagunduriko tresna ahaltsuak erabiltzen dira sistema digitalak diseinatzeko, batik bat diseinatu behar den sistema konplexua denean (ikus bibliografia [13]). Zirkuitu-bibliotekak erabiliz, nahi duen sistema osatuko du diseinatzaileak, grafikoki zein VHDL bezalako lengoaiaren bat erabiliz; gero, zirkuituaren portaera logikoa eta fisikoa simulatuko ditu, behin eta berriz, behar dituen ezaugarriak lortu arte; azkenik, sistema hori guztia zirkuitu integratu batean “programatuko” du. Nolakoa sistemaren konplexutasuna, halakoak izango dira diseinurako eta simulaziorako erremintak.

Bestalde, hainbat sistema digitalek mikroprozesadore bat erabiltzen dute osagai nagusi gisa. Oro har, sistema digitaletan erabiltzen diren mikroprozesadoreak kontrol-ekintzetarako espresuki diseinatutako prozesadoreak dira (mikrokontroladoreak), sarrera/irteera funtzioetarako moldatuta, kanpoko informazioa sentsoreen bidez hartzen dutena. Kontrol-ekintza jakin bat egiteko, programa egokiak sortu behar dira, prozesadore horien mihiztadura-lengoaia zein goi-mailako lengoaia bat erabiliz.

Ezinbestean, teknika, gailu eta tresna horiek guztiak, liburu honen helburuetatik at utzi behar izan ditugu. Baina liburuan ageri dena oinarritzat hartuta, ongi hornituta abiatuko da irakurlea hurrengo pausoak ematera sistema digitalen diseinuaren munduan. Izan dadila bidaia ahalik eta kitzikagarriena.

E1 Eranskina

Zenbakien adierazpideak

Liburu honen helburua ez bada ere, merezi du laburbiltzea sistema digitaletan (bereziki konputagailuetan) zenbakiak adierazteko gehien erabiltzen diren adierazpideak. Adierazpide horiek nola ulertu eta erabili baino ez dugu azalduko. Analisi sakonagoak egiteko, ikus bibliografia [11].

E1.1. ZENBAKI ARRUNTAK (*natural*)

E1.1.1. Kodeketa bitar hutsa

Zenbaki arruntak adierazteko, kodeketa bitar hutsa da adierazpide erabiliena. Kodeketa bitarrak 2 digitu edo **bit** (***bi**nary digit*) erabiltzen ditu zenbakiak adierazteko: 0 eta 1. n bitekin [0 .. 2^n–1] tarteko zenbakiak adieraz daitezke. Alderantziz, N zenbakia adierazteko, $\log_2 N$ bit behar dira. Esaterako, 32 bitekin 0tik 4.294.967.295era arteko zenbaki guztiak adieraz daitezke. Era berean, 800 zenbakia adierazteko, 10 bit behar dira gutxienez, bit kopuruak osoa izan behar duelako: $\log_2 800 = 9{,}64 \rightarrow 10$.

Kodeketa bitar hutsa, sistema hamartarra bezala, posizio-sistema bat da: digitu edo bit bakoitzak balio desberdina dauka haren posizioaren arabera. 10 oinarrian, pisu horiek 1 (10^0), 10 (10^1), 100 (10^2), ... dira. Bitarrean, 2 oinarrian, pisu horiek 1 (2^0), 2 (2^1), 4 (2^2), 8 (2^3), ... dira.

Zenbaki hamartar arruntak bitarrean kodetzeko, oinarri-aldaketa bat egin behar da, 10 oinarritik 2 oinarrira. Horretarako, ohiko algoritmoa aplikatu behar da: 2rekin zatiketak egin, behin eta berriz. Esaterako,

***x* = 24**

24 / 2 = 12	hondarra = **0**
12 / 2 = 6	hondarra = **0**
6 / 2 = 3	hondarra = **0**
3 / 2 = **1**	hondarra = **1**

$\boldsymbol{X = 1\,1\,0\,0\,0} \rightarrow 0 \times 2^0 + 0 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 + 1 \times 2^4 = 24$

Modu berean, zenbaki bitar bat 10 oinarrira itzultzeko, bit bakoitzaren pisua hartu behar da kontuan, aurreko adibidean egin den bezala.

Zenbaki bitarrak idazteko bit asko erabili behar dira. Askotan, hainbeste bit idatzi behar ez izateko, bit horiek launaka taldekatzen dira, eta talde bakoitza r = 16 oinarriko digitu bakar batek ordezkatzen du. 16 oinarrian, 16 digitu desberdin erabiltzen dira, eta horiek adierazteko, honako ikur hauek

erabiltzen dira: 0, ..., 9, A, B, C, D, E, F. Zenbakiaren bukaeran, H bat gehitu ohi da, hamaseitarrez adierazita dagoela gogorarazteko.

Ikus dezagun 16 biteko adibide pare bat.

$$X = 33023_{10} = 1000\ 0000\ 1111\ 1111_2 \rightarrow 80FF_{16}\ (80FFH)$$
$$Y = A70BH = 1010\ 0111\ 0000\ 1011_2 \rightarrow 42763_{10}$$

E1.1.2. BCD kodeketa (*Binary Coded Decimal*)

Lehen 10 digitu hamartarrak bitarrez kodetzeko, 4 bit erabili behar dira (3 bitekin, 8 zenbaki bakarrik kode daitezke):

$0 \rightarrow 0000$; $1 \rightarrow 0001$; $2 \rightarrow 0010$; ... $8 \rightarrow 1000$; $9 \rightarrow 1001$

Zenbait aplikaziotan, zenbaki hamartarren kodeketa bitar berezi bat erabiltzen da: jatorrizko zenbakiaren digitu hamartar bakoitza, dagokion kode bitarrez adierazten da. Hau da, ez da zenbakia kodetzen, zenbakiaren digitu bakoitza baizik. Kodeketa horri BCD deritzo. Adibidez,

$$X = 1583_{10} = 0001\ 0101\ 1000\ 0011_{BCD}$$
$$Y = 0110\ 0010\ 0100_{BCD} = 624_{10}$$

Kodeketa mota horrek bit gehiago erabiltzen ditu zenbakiak kodetzeko. Izan ere, 4 bit erabiltzen ditu digitu bakoitzeko, baina 4 bitekin 16 zenbaki arte kode daitezke, eta 10 digitu hamartar baino ez dago (11-15 tarteko kodeak ez dira inoiz erabiltzen).

Ohikoa da erabiltzea BCD kodeketa zenbakiak “erakutsi” behar direnean erakusgailuetan (digituak, pantailak eta abar). Une horretan, erakutsi baino lehen, bitar hutsetik (kodeketa arrunta) BCD kodera itzuli beharko dira zenbakiak (BCD deskodegailuen bidez).

E1.2. ZENBAKI OSOAK (*integer*)

Zenbaki positiboek (arruntak) zein negatiboek osatzen dute zenbaki osoen multzoa. Beraz, zenbaki osoak erabili behar baditugu, modu bat beharko dugu zenbaki positiboak eta negatiboak —esaterako, –3 eta 3— bereizi ahal izateko.

Adierazpide bat baino gehiago dago hori egiteko. Ikus ditzagun erabilienak.

E1.2.1. Zeinu/Magnitudea (*sign-and-magnitude*)

Adierazpide honetan, zenbakiaren zeinua eta magnitudea bereizten dira. Zeinua adierazteko, bit bat erabiltzen da, pisu handieneko posizioan: 0, zenbakia positiboa bada; eta 1, negatiboa bada. Magnitudea, berriz, zenbaki arruntak bezala kodetzen da, bitar hutsez.

Esaterako, 6 bit erabiliz:

$$X = 001111_{ZM} = +01111_2 = +15_{10}$$
$$Y = 111011_{ZM} = -11011_2 = -27_{10}$$
$$Z = -18_{10} = 110010_{ZM}$$
$$K = +5_{10} = 000101_{ZM}$$

Arazotxo batzuk ditu zeinu/magnitudea adierazpideak. Esaterako, bi kode desberdin daude 0 zenbakia adierazteko: +0 (00...0) eta –0 (10...0). Horrez gain, zenbakiak batzeko algoritmoa konplexu samarra da. Bi zenbaki batzeko, lehendabizi zenbakien zeinuak konparatu behar dira. Biak berdinak badira, magnitudeak batu behar dira, eta emaitzari zenbakien zeinua esleitu. Bestela, zeinuak desberdinak badira, zenbakien magnitudeak konparatu behar dira, handienari txikiena kendu behar baitzaio; emaitzaren zeinua, magnitude handienarena da.

Ikus ditzagun adibide batzuk:

a. 10001 (–1) + 11000 (–8)

Zeinuak	→ 1 = 1 (bi zenbakiak negatiboak dira)
Magnitudeak batu	→ 0001 + 1000 = 1001
Batura	→ 11001 (–9)

b. 00011 (+3) + 11001 (–9)

Zeinuak	→ 0 ≠ 1 (positiboa bata eta negatiboa bestea)
Magnitudeak konparatu	→ 0011 (pos.) < 1001 (neg.)
Kendu handienari txikiena	→ 1001 – 0011 = 0110 (6)
Batura	→ 10110 (–6)

Bestalde, zeinu-aldaketa eragiketa oso modu sinplean egiten da zeinu/magnitudeaz adierazitako zenbakietan: nahikoa da pisu handieneko bita aldatzea 0tik 1era edo 1etik 0ra.

E1.2.2. 2rako osagarria (*two's complement*)

Zenbaki osoak adierazteko gehien erabiltzen den adierazpidea da. Zenbakia eta zeinua ez dira bereizten, eta batera kodetzen dira.

Hala ere, zenbakiak kodetu baino lehen, aldaketa bat egiten da: zenbakia negatiboa bada, 2^n konstantea gehitzen zaio, positibo bihurtzeko (n = bit kopurua). Horrela egiten bada, n bitekin sor daitezkeen kodeetatik lehenengo erdia zenbaki positiboak adierazteko erabiltzen da, eta bigarrena, berriz, zenbaki negatiboak adierazteko.

Ikus dezagun adibide bat. 4 bitekin 16 kode desberdin lortzen dira: 0000tik 1111era. 16 kode horietatik lehenengo 8ak, 0000tik 0111era, zenbaki positiboak adierazteko erabiltzen dira, 0tik 7ra, hain zuzen ere. Gainerako zortzi kodeak, 1000tik 1111era, zenbaki negatiboak adierazteko erabiltzen dira, –8tik –1era. Zenbaki horiek adierazteko $2^4 = 16$ konstantea gehitzen zaie. Beraz, –8 zenbakia $–8 + 16 = 8$ zenbakia bezala kodetzen da (bitar hutsean): 1000. Modu berean, –7 zenbakia $–7 + 16 = 9 \rightarrow 1001$; –6 zenbakia $–6 + 16 = 10 \rightarrow 1010$; ..., eta –1 zenbakia $–1 + 16 = 15 \rightarrow 1111$.

Hauek dira, adibide gisa, 4 biteko zenbaki osoak 2rako osagarrian:

hamartarra	2rako osagarria	hamartarra	2rako osagarria
0	0000	–8	1000
1	0001	–7	1001
2	0010	–6	1010
3	0011	–5	1011
4	0100	–4	1100
5	0101	–3	1101
6	0110	–2	1110
7	0111	–1	1111

Beraz, zenbaki bitar bat emanda 2rako osagarrian, honako hau egin behar da zein zenbaki den jakiteko: 0z hasten bada, zenbakia positiboa da, gainerako bitek adierazten dutena; 1ez hasten bada, zenbakia negatiboa da eta 2^n konstantea kendu behar zaio zenbakiari zenbaki negatiboa lortzeko.

Badago prozedura sinple bat azken eragiketa hori egiteko; zenbakia negatiboa bada, hau da, pisu handieneko bita 1 bada, orduan bit guztiak ezeztatu eta 1 gehitu. Adibidez, 8 biteko zenbakiak:

0001 0011 → zenbakia positiboa da: **+19**

1100 1101 → zenbakia negatiboa da:

bit guztiak ezeztatu → 0011 0010

1 gehitu → 0011 0011 = 51 → **–51**

2rako osagarrian adierazitako zenbakien biten esanahia ohikoa da, pisu handieneko bitarena izan ezik, negatiboa baita; hau da:

$$X = X_{n-1}\, X_{n-2} \ldots X_1\, X_0 = X_0 \times 2^0 + X_1 \times 2^1 + \ldots + X_{n-2} \times 2^{n-2} + \boldsymbol{X_{n-1} \times (-2^{n-1})}$$

Adibidez,

1100 1101 → zenbakia negatiboa da: 1 + 4 + 8 + 64 – 128 = –51

n bit erabiliz, $[-2^{n-1}, 2^{n-1}-1]$ tarteko zenbakiak adieraz daitezke. Esaterako, 32 bitekin, adieraz daitekeen zenbakirik txikiena –2.147.483.648 da eta zenbakirik handiena +2.147.483.647. Agerikoa denez, zenbaki positiboak negatiboak baino bat gutxiago dira, 0 zenbakia positiboen artean kodetzen baita.

Batuketak egitea 2rako osagarrian dauden zenbakiekin erraza da, berdin tratatzen direlako zenbaki positiboak eta negatiboak. Nahikoa da ohiko algoritmoak erabiltzea batuketak egiteko, eta eragiketaren bururako-bita aintzat ez hartzea. Adibidez,

+7	0111
–2	1110
5	(1) **0101**

Kenketak batuketa gisa exekutatzen dira, bigarren eragigaiaren zeinua aldatuta. Zeinu-aldaketa egiteko, lehen aipatu dugun prozedura egin behar da: zenbakiaren bit guztiak ezeztatu eta 1eko bat gehitu. Hau da: $A - B = A + \overline{B} + 1$.

Adi! Gainezkatzea gerta daiteke batuketa (zein kenketa) bat egitean: emaitza ezin da adierazi ditugun bitekin. Irakurleak egiazta dezake gainezkatzea gertatzen dela batuketan baldin eta bi eragigaiak zeinu berekoak badira eta emaitza, ordea, kontrako zeinukoa (esaterako, 4 bitekin, 6 + 4 = 10 eragiketan).

Konputagailu guztiek erabiltzen dute 2rako osagarria zenbaki osoak adierazteko. Arrazoia garbia da: batuketa zein kenketa oso modu sinplean egiten dira.

E1.3. ZENBAKI ERREALAK (*real*)

Zenbaki osoez gain, konputagailu guztiek zenbaki mota orokorrago bat erabiltzen dute: zenbaki errealak (osoak gehi frakzionarioak). Adierazpide erabiliena zenbaki errealak adierazteko, koma higikorra da (*float point*).

Koma higikorreko zenbaki batean, bi atal bereizten dira: mantisa (*m*) eta berretzailea (*b*). Esaterako, 10 oinarrian, honela adieraz daitezke zenbakiak:

$1500 = 1{,}5 \times 10^{3} \quad \rightarrow \quad m = 1{,}5 \quad b = 3$

$0{,}000032 = 3{,}2 \times 10^{-5} \quad \rightarrow \quad m = 3{,}2 \quad b = -5$

Zenbaki bitarren koma higikorreko adierazpena berdina da, baina berreturaren oinarria 2 da.

$(24)\ 011000 = 0{,}11 \times 2^{101} \rightarrow \quad m = 0{,}11 \quad b = 101$

Koma higikorreko adierazpena ez da unibokoa; hots, aukera asko dago zenbaki bera adierazteko, mantisaren komak ez baitu posizio finkoa. Esaterako,

$1500 = 1{,}5 \times 10^{3} = 15 \times 10^{2} = 0{,}15 \times 10^{4} = \ldots$

Hori dela eta, mantisa “normalizatu” bat definitu behar da.

Koma higikorreko zenbakien formatu estandar bat erabiltzen da konputagailuetan. Ez dugu azalduko formatu hori; bakarrik esan hiru atal bereizten direla:

- bit bat zenbakiaren zeinua adierazteko
- mantisa normalizatua ($0 < m < 1$)
- berretzailea (konstante bat gehituta, berretzaile negatiboak positibo bihurtzeko)

Azken ohar bat. Koma higikorreko zenbakien adierazpena ez da zehatza. Hots, erroreak egiten dira zenbaki hamartarrak adierazteko. Eskuarki arazorik ez badago ere (bit asko erabiltzen baitira zenbakiak adierazteko), kontuan hartu behar dira balizko erroreak oso zenbaki handiekin edo txikiekin lan egin behar denean.

E2 Eranskina

Zirkuitu digitalen oinarrizko teknologia

Liburu honen hasierako kapituluetan esan den moduan, transistorea da zirkuitu digitalen oinarrizko osagaia. Transistoreak erabiliz egiten dira oinarrizko funtzio logikoak exekutatzen dituzten gailuak. Horietan oinarrituta, konplexutasun handiagoko zirkuituak sortzen dira, konbinazionalak zein sekuentzialak, bai eta memoriak ere.

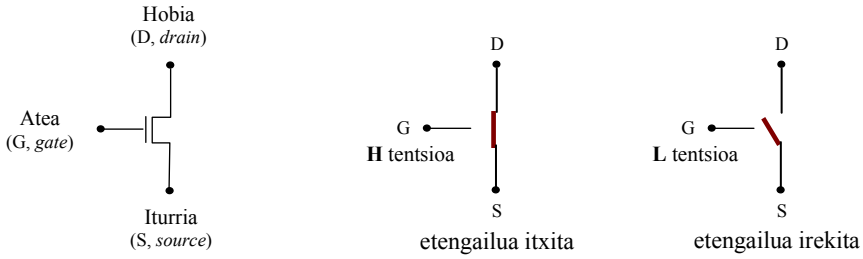
Eranskin honetan, teknologiari lotutako oinarrizko kontzeptu eta termino batzuk azalduko ditugu, gainetik bada ere. Gaian sakontzeko, ikus bibliografia [12].

E2.1. TRANSISTOREAREN FUNTZIONAMENDUA ZIRKUITU DIGITALETAN

Transistorea da zirkuitu digitalen oinarrizko osagaia, baina ez dago transistore mota bakarra. Zirkuitu digitalak eraikitzeko erabiltzen diren transistoreak bi familiakoak dira: transistore bipolarrak eta MOS (*Metal-Oxide Semiconductor*) transistoreak (NMOS eta PMOS motakoak). Bi familia horietako transistoreek ezaugarri desberdinak dituzte hainbat parametrotan: kommutazio-abiadura, kontsumoa, funtzionamendu-tentsioak, eta abar.

Zirkuitu digitalak eraikitzeko erabiltzen diren transistoreen arabera, familia edo teknologia desberdinei buruz hitz egiten da.

Dena den, portaerari dagokionez, transistoreen funtzionamendua oso sinplea da zirkuitu digitalak eraikitzeko erabiltzen direnean, etengailu gisa funtzionatzen baitute. Hiru mutur ditu transistore batek, eta horietako bat, atea (*gate*) MOS transistoreetan, kontrol-terminala da, transistorearen funtzionamendua kontrolatzen baitu.



E2.1. irudia. NMOS transistore baten eskema logikoa eta haren funtzionamendua etengailu gisa (eredu ideala). PMOS transistorearen portaera berdina da, baina atearen tentsioak kontrakoak izanik.

Atean dagoen tentsioak kontrolatzen du transistorearen funtzionamendu digitala. NMOS transistorearen kasuan (eta D eta S terminalei dagokien tentsioa jarrita), tentsio altua dagoenean atean (H), hobia eta iturria konektatzen dira; ordea, tentsio baxua badago (L), ez dago konexiorik hobiaren eta iturriaren artean (zirkuitu irekia). (PMOS transistoreen funtzionamendua bera da, baina tentsio-mailak aldatuta.)

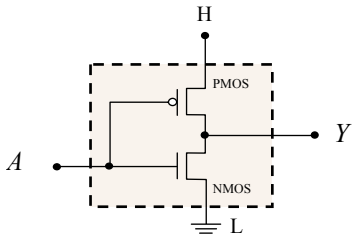
Hori dela eta, transistorea "etengailu elektroniko" gisa erabiltzen da. Bi egoera horiek —itxita eta irekita— erabiltzen dira zirkuitu logikoen bi balioak adierazteko: 1a eta 0a.

E2.2. TRANSISTOREEN BIDEZ EGINIKO ATE LOGIKOAK: BI ADIBIDE

Ikus dezagun, adibide pare baten bidez, nola eraiki daitezkeen ate logikoak transistoreak erabiliz; CMOS (*complementary metal-oxide semiconductor*) familiako adibideak aztertuko ditugu (NMOS eta PMOS transistoreak batera erabiltzen dira CMOS familian).

NOT ate bat

E2.2. irudian, NOT ate baten gauzatzea ageri da. Zirkuituak sarrera eta irteera bana du —*A* eta *Y*—, eta bi transistore erabiltzen ditu (goikoa PMOS motakoa, eta behekoa NMOS motakoa). Sarreraz eta irteeraz gain, ohiko elikadura-konexioak ere baditu zirkuituak, H eta L tentsioetara (adibidez, 5 volt (H) eta 0 volt (L, lurra)).



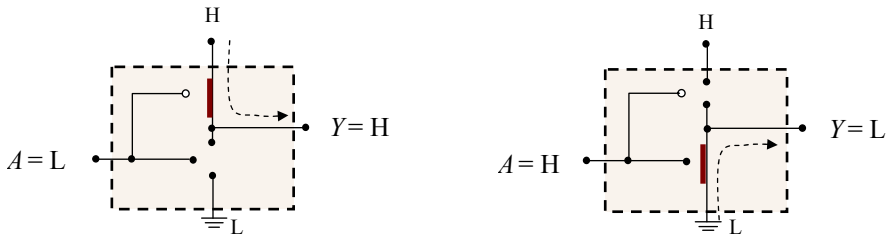
E2.2. irudia. NOT ate bat, bi transistoreen bidez egina (aukera bat).

E2.3. irudian ageri den moduan, sarrerako tentsioa baxua denean, goiko transistorea "aktibatuta" dago, eta, ondorioz, hobiaren eta iturriaren arteko bidea itxita dago; beheko transistoreak, ordea, zirkuitu ireki gisa

funtzionatuko du, “desaktibatuta” dagoelako. Hori dela eta, *Y* irteeran H tentsioa izango dugu (elikadura-tentsioa, hain zuzen ere).

Aldiz, sarrerako tentsioa altua bada, PMOS transistoreak zirkuitu ireki gisa funtzionatuko du, desaktibatuta dagoelako, eta NMOS transistoreak, berriz, zirkuitulabur gisa. Irteerako tentsioa, beraz, L izango da (beheko tentsioa, hain zuzen ere).

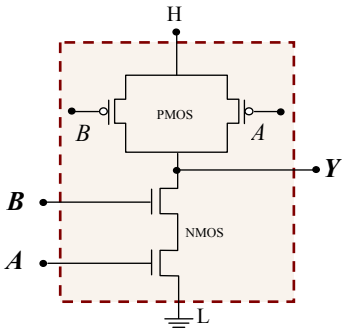
Laburbilduz, sarreran H tentsioa dagoenean, irteeran L tentsioa dago, eta sarreran L tentsioa dagoenean, irteeran H tentsioa dago. Ezagutzen dugun moduan, horixe bera da inbertsorearen, NOT atearen portaera.



E2.3. irudia. NOT atearen funtzionamendua: $A = \text{L} \rightarrow Y = \text{H}$; eta $A = \text{H} \rightarrow Y = \text{L}$.

NAND ate bat

Azter dezagun beste adibide bat. E2.4. irudian, NAND ate bat sortzeko aukera bat ageri da. Bi sarrera ditu zirkuituak, *A* eta *B*, eta irteera bat, *Y*.



E2.4. irudia. NAND ate bat, lau transistoreen bidez egina (aukera bat).

A eta *B* sarreren lau konbinazioak analizatu behar ditugu:

- A = L, B = L → Y = H

  A = B = L direnez, beheko bi transistoreak (NMOS) "irekita" (desaktibatuta) daude; goiko biak (PMOS), ordea, "itxita" (aktibatuta) daude. Ondorioz, Y irteerako tentsioa goiko H tentsioa izango da, hara konektatuta baitago goiko bi transistoreen bidez.

- A = L, B = H, edo A = H, B = L → Y = H

  Antzeko kasua da hau. Beheko bi transistoreetako bat itxita dago, eta bestea irekita dago; ondorioz, ez dago konexiorik Y-ren eta lurraren artean (bi transistoreak seriean daudelako). Aldiz, goiko transistore bat irekita badago ere, bestea itxita dago; beraz, beti dago bide bat goiko H tentsioaren eta Y irteeraren artean (transistoreak paraleloan daudelako).

- A = H, B = H → Y = L

  Orain, beheko bi transistoreak itxita daude (H tentsioa dago ateetan); beraz bide bat ireki da Y-ren eta lurraren artean; gainera PMOS transistoreak irekita daude, ateak desaktibatuta daudelako. Beraz, Y irteeran L tentsioa izango da.

Laburbilduz, honako tentsio-konbinazio hauek eskaintzen ditu zirkuituak:

A	***B***	***Y***
L	L	H
L	H	H
H	L	H
H	H	L

hau da, NAND ate baten portaera fisikoa (ikus 2. kapitulua).

Bi adibide baino ez dira izan aurrekoak. Baina horien konbinazioen bidez, zirkuitu digital konplexuagoak lortzen dira: multiplexoreak, batugailuak, biegonkorrak...

E2.3. MEMORIA-GELAXKAK

Ezagutzen dugunez, bi motatako zirkuitu digitalak bereizi ohi dira: konbinazionalak eta sekuentzialak. Lehendabizikoen erantzuna sarreren uneko balioen araberakoa da; bigarrenek, ordea, sistema digitaletan ezinbestekoa den propietate bat dute: memoria dute. Biegonkorrak aztertu

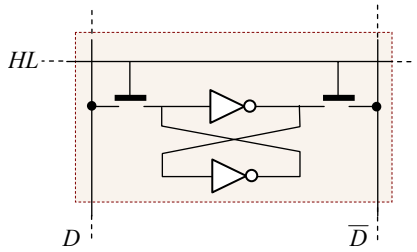
ditugunean azaldu dugun moduan, irteerak berrelikatzen direnean lortzen da zirkuituak memoria izatea.

Biegonkorrak eta erregistroak informazio kopuru txikia gordetzeko erabiltzen badira ere, informazio asko gorde behar denean, RAM eta ROM memoriak erabiltzen dira.

Dakigunez, bi motatako RAM memoriak daude: estatikoak eta dinamikoak. Egitura desberdinak erabiltzen dira, memoria mota bakoitzean, bit bateko oinarrizko gelaxka sortzeko. Ikus ditzagun gelaxka horiek, sinplifikatuta bada ere.

RAM estatikoen oinarrizko gelaxka

Biegonkorrak sortzeko erabiltzen den oinarrizko egitura 4. kapituluan ikusi dugu. Antzeko egitura erabiltzen da SRAM memorien bit bateko gelaxkak sortzeko, E2.5. irudian ageri den moduan.



E2.5. irudia. Bit bateko SRAM memoria-gelaxkaren eskema sinplifikatua. Bi etengailuak eta NOT ateak transistoreen bidez egiten dira.

Bi NOT ateen bidez, begizta bat antolatzen da, eta begizta horrek betetzen du memoriaren funtzioa, informazioa mantentzea. Bit bat kargatuta, begiztaren barruan mantenduko da.

Bit bat (adibidez, 1eko bat) kargatzeko (idazteko), hau egin behar da: jarri 1ekoa goiko NOT atearen sarreran eta kontrakoa, 0koa, NOT atearen irteeran (alderantziz, 0ko bat idazteko). Horretarako, *HL* (hautatze-lerroa)

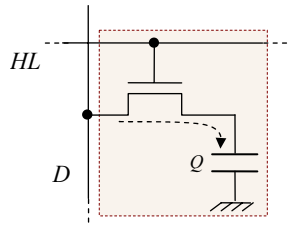
seinalea eta D eta $\overline{D}$ datu-sarrerak erabiltzen dira. HL = 1 denean, bi etengailuak (transistoreak) itxi egiten dira eta, ondorioz, D datu-sarrera NOT atearen sarreran jartzen da (eta kontrako balioa irteeran). HL sarrera deskonektatzean, etengailuak irekiko dira eta kargatutako balioa mantenduko da.

Eskuarki, lerro berberak erabiltzen dira irakurketarako eta idazketarako, lehenengo kasuan sarrera gisa eta bigarrenean irteera gisa erabiliz (hiru egoerakoa). Izan ere, irakurketa egiteko, nahikoa da HL seinalea aktibatzea, baina kasu honetan D lerroa irteera gisa erabiliz

(Adi! asko sinplifikatu ditugu aurreko prozesuak; errealitatean, konplexuagoak dira.)

RAM dinamikoen oinarrizko gelaxka

E2.6. irudian ageri da DRAM memoria baten oinarrizko gelaxka sinplifikatua. Bi balio logikoak (1/0) sortzeko, kondentsadore baten karga erabiltzen da: kondentsadorea kargatuta edo deskargatuta egongo baita.



E2.6. irudia. Bit bateko DRAM memoria-gelaxkaren eskema sinplifikatua.

HL seinalea aktibatzen denean, transistorea aktibatuko da, eta, ondorioz, etengailua itxiko da eta D sarrerako balioa (tentsioa, karga) kondentsadorera pasatuko da. HL desaktibatzen denean, kondentsadorea isolatuta geratuko da, eta metatutako karga mantenduko da, denbora-tarte batean behintzat.

Laburbilduz. Aurreko irudietan ageri den moduan, DRAM memorien gelaxkak SRAM memorienak baino sinpleagoak dira (transistore gutxiago ditu); horregatik, bit gehiago (4 aldiz edo) integra daitezke erdieroalezko material zati berean: DRAM memoriak trinkoagoak dira. Hala, 1 Gbit baino edukiera handiagoko DRAM memoriak merkaturatzen dira dagoeneko.

Ordainean, DRAM memorietan gordetzen den informazioa ez da iraunkorra, kondentsadoreak, isolatuta badaude ere, deskargatzen baitira. Horregatik, DRAM memorien edukia "freskatu" behar da aldian behin.

E2.4. ZIRKUITU INTEGRATUAK

Funtzio logikoak egiten dituzten transistoreen bidezko zirkuituak milimetro karratu batzuk baino ez dituen erdieroalezko "xafla" edo dado (*die*) batean integratzen dira, teknika fotolitikoak erabiliz; hala, zirkuitu integratuak sortzen dira. Mota, egitura, funtzio, itxura eta abar desberdineko zirkuitu integratuak daude; izan ere, konturatu gabe ere, zirkuitu integratuez inguratuta gaude: etxe-tresnetan, automobiletan, lantokietan, aisiarako erabiltzen ditugun gailuetan eta abar. Zirkuitu integratuei txip (*chip*) ere deritze.

Eskuarki, zirkuitu integratuak zeramikazko kapsula batean gordetzen dira, eta, kanpoalderantz, hankatxo metaliko batzuk (*pin*) jartzen dira kanpo konexioak egin ahal izateko. Hankatxo kopurua arazo bihur daiteke handia denean, eta, hori dela eta, kapsulatze-teknologiak garrantzi handikoak dira, txipen arteko komunikazio azkarra bideratzeko, beroa barreiatzeko eta abar.

E2.7. irudia. Txipen ohiko itxura.

Transistore asko integra daiteke txip batean, eta, hala, konplexutasun-maila oso desberdineko funtzio logikoak eraiki daitezke: ate logiko bat zein prozesadore osoa; erregistro bat zein memoria handi bat. Ohikoa da zirkuitu integratuak sailkatzea erabiltzen dituzten transistoreen kopuruaren arabera. Hauek dira halako sailkapenetan gehien erabiltzen diren izenak[36]:

[36] Sailkapen honen mugak ezin dira finkotzat hartu. Izan ere, zirkuituen integrazio-maila, hau da, transistore kopurua txip batean, gero eta handiagoa da eta oso azkar hazten da. Esaterako, 2-3 urtean behin, bikoiztu egiten da txip batean integratzen den transistore kopurua. Beraz, hartu zenbakiak erreferentzia soil gisa.

- SSI (*Small Scale Integration*) maila. Txipeko transistore kopurua txikia da, ate gutxi batzuk integratzeko adina. Gutxi gorabehera, 10 ate inguru. Oinarrizko zirkuituak dituzte barnean: ateak eta biegonkorrak.

- MSI (*Medium Scale Integration*) maila. Txip batek transistore gehiago ditu, 10 – 100 ate integratzeko adina. Eraikitzen diren funtzioak konplexuagoak dira: multiplexoreak, deskodegailuak, erregistroak, kontagailuak eta abar.

  Aurreko bi kategorietan sartu behar dira konplexutasun txikiko edo ertaineko sistema digitalak sortzeko duela gutxi arte asko erabili diren TTL (edo MOS) 74 familiako zirkuituak. Esaterako, 74LS00 izenekoa: bi sarrerako 4 NAND ate; 74LS04a: 6 NOT ate; 74LS74a: 2 D biegonkor; 74LS283a: 4 biteko batugailu bat; edo 74LS669a, 4 biteko U/D kontagailua.

  Gaur egun oso gutxitan erabiltzen dira.

- LSI (*Large Scale Integration*) maila. Txip batean dagoen transistore kopurua handiagoa da, 100 – 10.000 ate integratzeko adina. Konputagailu baten azpisistema oso bat integra daiteke txip batean: unitate aritmetiko/logiko konplexua, erregistro-multzoa, memoria txiki bat, eta abar.

- VLSI (*Very Large Scale Integration*) maila. Transistore kopuru are handiagoa, 10.000 – 100.000 ate integratzeko behar dena. Denetarik aurki daiteke maila honetan: memoriak, zirkuitu programagarriak, prozesadore txikiak, eta abar.

  Kategoria honetan sartzen dira, esaterako, sistema digital nahiko konplexuak eraikitzeko asko erabiltzen diren ASIC (*Application Specific Integrated Circuit*) zirkuituak edo CPLD (*Complex Programmable Logic Device*) zirkuitu programagarriak.

- ULSI (*Ultra Large Scale Integration*) maila. Txip konplexuenak dira hauek, 100.000 ate integratzeko baino transistore gehiago dutenak (esaterako, 100 milioi transistore). Mota horietakoak dira edukiera handiko memoriak edo gaur egungo prozesadoreak.

  Batzuk bakarrik aipatzeagatik, hemen kokatu behar ditugu gaurko prozesadoreak —Pentium, Itanium, Power...— edo edukiera oso handiko memoriak (1 Gbit baino gehiago dagoeneko).

E2.5. ZIRKUITU INTEGRATUEN PARAMETRO BATZUK

Lehen esan dugun moduan, bi motatako transistoreak dira gehien erabiltzen direnak zirkuitu integratuak sortzeko: transistore bipolarrak eta MOS transistoreak. Hainbat urtetan zehar, transistore bipolarrak MOS transistoreak baino gehiago erabili izan dira zirkuitu integratuak sortzeko; gaur egun, ordea, ia zirkuitu digital guztiak MOS zirkuituak dira.

Transistore bipolarrak erabili dira, esaterako, **TTL** (*Transistor-Transistor Logic*) eta **ECL** (*Emitter-Coupled Logic*) zirkuitu-teknologietan. Duela gutxi arte, TTL teknologiak oso erabilera zabala izan du mota guztietako zirkuitu integratuak sortzeko, gaur egun gutxi erabiltzen bada ere (prototipoak, behe-mailako zirkuituak...). ECL teknologia, aldiz, goi-mailako zirkuituetan bakarrik erabili da (azkarrena bazen ere, garestia zelako) eta dagoeneko ez da erabiltzen.

NMOS eta **CMOS** teknologietan, MOS transistoreak erabiltzen dira. Zirkuitu integratu gehienak —prozesadoreak, memoriak...— MOS motakoak dira gaur egun.

Urteetan zehar hainbat familia eta serie sortu dira teknologia bakoitzaren barruan. Esaterako, TTL zirkuituetan, 74LS (*low schottky*) familiako zirkuituak oso erabilera zabalekoak izan dira. Era berean, erabilera zabala dute CMOS familiako 74AHC (5 voltekoak), 74LVC (3,3 voltekoak)... serieetako zirkuituak.

Hainbat parametro erabiltzen dira zirkuitu integratuak analizatzeko; haien artean, honako hauek:

- **Elikadura-tentsioa**

  Zirkuitu guztiak tentsio jakin batera konektatu behar dira funtziona dezaten: “elikatu” behar dira. Tentsio horrek erlazio handia dauka zirkuituak xurgatzen duen potentziarekin (kontsumoarekin) eta, ondorioz, sortzen den beroarekin. Aukeran, tentsiorik baxuena erabili beharko genuke (tentsioa oso baxua denean beste arazo batzuk ageri dira).

  TTL teknologian, elikadura-tentsioa 5 volt izan ohi da. CMOS zirkuituen kasuan, ohikoa da, baita ere, 3,3 voltekoa (tentsio-balioak ez dira zehatz-zehatzak, tarte baten barruan daude eta). Izan ere, elikadura-tentsioa jaisteko joera dago.

- **Sarrera eta irteerako balio logikoak**

  Zirkuitu logikoen bi balioak (1/0) adierazteko, bi tentsio-maila erabili ditugu: H —tentsio altua— eta L —tentsio baxua—. Bi balio horiek ez dira zehatz-zehatzak, eta tarte baten barruan kokatzen dira.

  Adibide gisa, honako tarte hauek erabiltzen dira 5 volteko CMOS teknologian:

  Sarreran: H → [3,5 – 5] volt
  L → [0 – 1,5] volt

  Irteeran: H → [4,4 – 5] volt
  L → [0 – 0,33] volt

- **Kommutazio-abiadura,** eta, ondorioz, erantzun-denbora

  Sistema digitaletako transistoreek bi egoeren artean kommutatzen dute: irekita → itxita edo itxita → irekita.

  Aldaketa horietan oinarritzen dira funtzio logikoak, eta, beraz, funtzio jakin baten erantzun-denbora —sarrerak aldatzen direnetik irteeran emaitza lortu arte— transistoreen kommutazio-abiaduraren funtzioa da.

  Eskuarki, transistore bipolarrak MOS transistoreak baino azkarragoak izan dira, nahiz eta gaur egun diferentzia hori desagertu den.

  Bi parametro nagusi eman ohi dira fabrikatzaileen eskuliburuetan kommutazio-denbora adierazteko:

  - t_{PHL}, irteera H tentsiotik L tentsiora aldatzeko denbora
  - t_{PLH}, irteera L tentsiotik H tentsiora aldatzeko denbora

  Parametroen adibideak ematea zaila da, familia bakoitzeko aldaera asko eta oso desberdinak daudelako.

  Hala ere, erantzun-denborak, gaur egun, 2-6 ns tartean daude CMOS zirkuituetan.

- **Potentzia-xurgapena (kontsumoa)**

  Oro har, xurgatzen den potentzia kalkulatzeko, elikadura-tentsioa eta batez besteko elikadura-korrontea biderkatu behar dira.

  Eskuarki, kontsumoa eta abiadura oso erlazionatuta daude: zenbat eta azkarragoa izan zirkuitua, hainbat eta altuagoa da kontsumoa. Beraz, konpromiso bat bilatu behar da: zirkuituak ahalik eta azkarrenak izatea kontsumo ahalik eta baxuena izanik.

Berriro ere, zaila da parametro zehatzak ematea, baina, adibide gisa, honako hau esan daiteke:

CMOS ate bat → 3 – 500 μW (maiztasunaren arabera)

TTL → 1 – 5 mW (serieen arabera)

Zirkuituek xurgatzen duten potentzia biziki garrantzitsua da bateriekin funtzionatzen duten sistema eramangarrietan (ordenagailuak, musika-aparailuak, telefonoak...); kasu horietan, abiadura baino, kontsumoa optimizatu behar da (horregatik, TTL zirkuituak ez dira egokiak erabilera horietarako).

Aipatu bezala, kommutazio-denbora eta kontsumoa dira parametro nagusiak. Askotan, konparazioak egiteko, bi parametro horiek biderkatzen dira. Biderkadura txikiena duena izango da aukerarik onena.

- ***Fan-out***

  Zenbat sarreretara eraman daitekeen zirkuitu baten irteera adierazten du *fan-out* parametroak. CMOS zirkuituetan, oso altua da parametro hori (TTL zirkuituenarekin alderatuta); horrek adierazten du zirkuitu baten irteera-korrontea sarrera-korrontea baino askoz altuagoa dela (izan ere, egoera egonkorrean, sarrera-korrontea 0 da), eta, ondorioz, irteera-korrontea sarrera askotara bana daitekeela.

E3 Eranskina

VHDL, hardwarea deskribatzeko lengoaia

E3.1. SARRERA

6. kapituluan aipatu dugun moduan, hainbat metodologia daude sistema digitalak diseinatzeko. Guztien helburua da, jakina, sistema logikoen osagaiak eta konexioak zuzen definitzea, eta, azkenik, hardware egokia erabiliz, zirkuituak eraikitzea. Bi metodo nagusi daude sistema digitalak adierazteko: (a) orain arte erabili dugun metodo grafikoa —zirkuituak marraztea eta haien artean konektatzea—, eta (b) zirkuituak eta haien arteko konexioak lengoaia bereziak erabiliz deskribatzea.

Eskuarki, sistemaren diseinua CAD motako programa baten bidez egingo dugu. Programa horiek (Quartus, ALTERA etxekoa, esaterako) bi aukerak eskaintzen dituzte: tresna grafiko bat sistema digitalaren osagaiak eta konexioak marrazteko; eta hardwarea deskribatzeko lengoaiaren bat, sistemaren osagaiak, konexioak eta abar testu-fitxategi baten bidez adierazteko (oro har, bi aukerak batera erabil daitezke sistema batean, parte bat grafikoki eta beste bat testu moduan adieraziz).

Sistemaren diseinua bukatuta, erabaki bat hartu behar da: zirkuitua nola eraiki nahi den, hain zuzen ere. Garai batean, sistema osoa behe-mailako txipen bidez eraikitzen zen (esaterako, TTL 74LS serieko zirkuituak erabiliz); gaur egun, ordea, ohikoa da VLSI mailako zirkuitu programagarri bat edo ASIC bat erabiltzea, non sistema osoa integratu ahal izango dugun. Izan ere, CAD programak berak prozesatuko du diseinatu dugun sistema eta sintetizatuko du sistema osoa VLSI txip batean.

Esan dugun moduan, zirkuitu baten portaera testu moduan adieraz daiteke, eta, horretarako, hainbat lengoaia diseinatu da: HDL lengoaiak, *Hardware Description Languages*, hain zuzen ere; esaterako, Verilog edo VHDL. Ohiko programazio-lengoaietan erabiltzen diren egiturak erabiltzen dira hardwarea deskribatzeko lengoaietan ere, baina hardwareko funtzioak definitzeko. Gaur egun, **VHDL** lengoaia ia estandarra da elektronikaren industrian. Lengoaia konplexua da VHDL, hainbat sistema fisiko deskriba daitezkeelako. Eranskin honetan, lengoaia horren aurkezpen xume bat egingo dugu, eta sistema digital sinpleak nola deskribatu azalduko dugu. Gure helburua ez da VHDL lengoaiaren eskuliburua egitea, eta, hortaz, oso gainetik azalduko dugu. Edozein kasutan, suposatu dugu irakurleak programazio-lengoaiaren bat ezagutzen duela (Pascal, Ada, C, Java...).

Informazio sakonagoa behar izanez gero, bibliografian adierazitako liburuetara jo dezake irakurleak [14].

E3.2. NOLA DESKRIBATU SISTEMA DIGITAL BAT VHDL ERABILIZ

Oro har, sistema digitalak diseinatzeko, bloke funtzionalak erabiltzen dira. Ohiko bloke funtzionalak ezagunak ditugu dagoeneko, liburuan azaldu baititugu: konbinazionalak —ate logikoak, multiplexoreak, deskodegailuak, batugailuak...—; sekuentzialak —D eta JK biegonkorrak, erregistroak, desplazamendu-erregistroak, kontagailuak...—; eta memoriak. Bloke horiek zehazteko, haien **ikurra** (sarrerak eta irteerak) eta **portaera logikoa** (zer egiten duten) definitu ditugu.

Oinarrizko zirkuitu horiek erabiliz, beste bloke funtzional batzuk defini ditzake diseinatzaileak, diseinua modu hierarkikoan aurrera eramateko, eta dagoeneko diseinatu dituen azpisistemak behin eta berriz erabili ahal izateko. Azpisistema edo bloke horien definizioa bi partetan ematen da:

- **Sistemaren ikurra,** hau da, sistemaren sarrerak eta irteerak definitu behar dira.

  Ohikoa da zirkuitua lauki baten bidez adieraztea, eta bertan azaldu zirkuituaren sarrerak, irteerak eta bakoitzaren bit kopurua edo tamaina.

- **Sistemaren portaera edo funtzionamendua.** Sarrerak eta irteerak definituta, adierazi behar da nola prozesatu sarrerak eta nola lortu irteerak; hau da, zer egiten duen zirkuituak.

  Zirkuituaren funtzioa adierazteko, egia-taulak, ekuazio logikoak edo, konplexua den neurrian, behe-mailako osagaien konexio-zerrendak erabili ohi dira.

Bada, hori da, hain zuzen ere, VHDL erabiltzen denean jarraitu behar den prozedura zirkuituak definitzeko[37]. Batetik, zirkuitua edo sistema osoa bloke gisa definitu behar da, eta haren sarrerak eta irteerak identifikatu. Hori egiteko, **`entity`** izeneko egitura erabili behar da.

Bestetik, bloke horren funtzionamendua zehazteko, **`architecture`** izeneko egitura erabili behar da. Oro har, hiru eredu daude sistemaren `architecture` izeneko definizioa emateko, hau da, sistemaren portaera zehazteko:

---

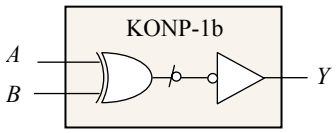
[37] Hori bera egin dugu grafikoki 6. kapituluko ariketetan (ikus, adibidez, 6.4. ariketa). Ez ahaztu, sistema konplexua den neurrian, ohikoa da sinpleagoak diren azpimoduluak eta haien arteko konexioak definitzea (*top-down* estrategia).

- Sistemaren funtzionamendu logikoa adieraziz; eredu horri **portaera** (*behavior*) izena ematen zaio. Abstrakzio-maila altuena da.
- Sistemako datuen fluxua (sarreretako datuen ibilbidea sarreretatik irteerara) adieraziz; eredu horri **datu-fluxua** (*dataflow*) izena ematen zaio.
- Sistemaren osagaien arteko konexioak adieraziz; eredu horri **egitura** (*structure*) izena ematen zaio. Hardwareari lotuen dagoen deskribapena da.

Hiru aukera horiek daude diseinatzailearen esku zirkuituak definitzeko.

Ikus dezagun nola egiten den zirkuitu baten definizioa adibide sinple baten bitartez. **Bit bateko bi zenbakien konparagailua** (berdintasuna bakarrik) definitu behar dugu VHDL erabiliz. Egia-taula egin, eta erraz lortzen da funtzio horri dagokion ekuazio logikoa (eta, hortik, nola egin zirkuitua XOR eta NOT ate baten bidez, esaterako):

$$Y = \overline{A \oplus B}$$



Aipatu dugun moduan, bi definizio eman behar ditugu VHDLn bloke hori zehazteko: blokearen ikurra (`entity`, hau da, sarrerak eta irteerak), eta blokearen funtzionamendua (`architecture`).

- "Blokearen" ikurraren definizioa (`entity`)

  Oso zirkuitu sinplea definitu behar dugu, bit bateko bi sarrera eta irteera bat baino ez baititu. Honela definitzen da VHDL erabiliz:

```
entity KONP-1b is
   port( A,B:  in BIT;
         Y:    out BIT );
end KONP-1b;
```

  Ageri denez, **`port`** hitza erabiltzen da sarrerak eta irteerak adierazteko. Horren barruan, bit bateko sarrerak "`in BIT`" gisa deklaratu behar dira, eta bit bateko irteera "`out BIT`" gisa.

- Blokearen funtzionamendua (`architecture`)

  Hainbat modutan adieraz dezakegu bit bateko konparagailuaren funtzionamendua:

  → PORTAERA (*behavior*)

  Zirkuituaren portaera logikoa adierazi behar da, egia-taula edo beste definizioren bat erabiliz. VHDL lengoaiak hainbat "agindu" eskaintzen ditu deskribapena egiteko. Aginduak bi kategoriatan sailkatu ohi dira: sekuentzialak eta konkurrenteak. Agindu konkurrenteak une oro exekutatzen dira batera, eta ez dago ordenarik haien artean. Sekuentzialak, ordea, ordena jakin batean exekutatzen dira, bata bestearen ondoren, eta `process` izeneko egituraren barruan doaz.

  Konparagailuaren kasuan, esaterako, honako definizio hau eman daiteke:

```
architecture DEF1 of KONP-1b is
begin
   process (A, B)
   begin
      if (A = B)  then Y <= '1';
                  else Y <= '0';
      end if;
   end process;
end DEF1;
```

  Hasteko, `architecture` gako-hitzaren ondoren, blokearen deskribapenari eman nahi diogun izena adierazi behar dugu (kasu honetan DEF1 izena eman diogu, baina edozein izan daiteke).

  Zirkuituaren funtzionamendua `begin` eta `end` sententzien artean adierazten da. Aurreko adibidean, esaterako, agindu sekuentzial bat (`if`) erabili dugu funtzionamendua zehazteko; horregatik, lehenengo sententzia `process (A,B)` da. Hala, `process` egituraren barneko sententziak exekutatuko dira (prozesatu, alegia) `A` edo `B` sarreren balioak aldatzean. Balio horiek aldatu ezean prozesua "lo" dago, eta "esnatuko" da baten bat aldatzean.

  Azkenik, aurreko adibidea erraz ulertzen den arren, pixka bat azalduko dugu: baldin `A` eta `B` sarreren balioak berdinak badira, orduan `Y` irteerak 1 balioa hartuko du (`Y` <= '1'; <= ikurrak esleipena adierazten du VHDLn); bestela, `Y` irteerak 0 balioa hartuko du (`Y` <= '0').

Konparagailuaren portaera beste modu honetan adieraz daiteke, agindu konkurrenteak erabiliz:

```
architecture DEF2 of KONP-1b is
begin
   Y <= '1' when (A = B) else '0';
end DEF2;
```

Kasu horretan, `Y` irteerari behin eta berriro esleitzen zaio 1 edo 0 balioa, `A` eta `B` sarreren balioen arabera.

→ DATU-FLUXUA (*dataflow*)

Datuen fluxua —nondik nora— adierazi behar da. Adibide honetan, bi sarrerak XOR ate logiko batera doaz, eta, ondoren, NOT ate batera. Beraz,

```
architecture DEF3 of KONP-1b is
begin
   Y <= not (A xor B) after 10 ns;
end DEF3;
```

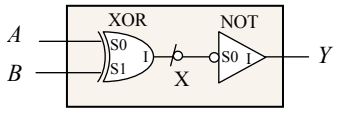
Deskribapen horretan, ez da `process` sententzia erabili behar, *not* eta *xor* funtzio logikoak sarreren aldaketak gertatzean prozesatzen baitira. Beste aldetik, aurreko kasuan ez bezala, irteera sortzeko behar den erantzun-denbora ere adierazi da, 10 ns kasu horretan (aukera bat da).

→ EGITURA (*structure*)

Sistemaren egitura adierazi behar da. Horretarako, zirkuituaren osagaiak definitu behar dira aurretik (`component`), kasu honetan XOR eta NOT ate bana, eta haien sarrerak eta irteerak etiketatu.

Gero, osagaien arteko konexioak adierazi behar dira, bai eta sarrerak eta irteerak ere. Tarteko konexioak ere espresuki adierazi behar dira, zirkuitu baten irteera beste baten sarrerara konektatu ahal izateko.

Hona hemen KONP-1b zirkuituari dagokion etiketatzea eta VHDL definizioa:



```
architecture DEF4 of KONP-1b is

component XOR                  -- XOR atearen definizioa
   port( S0,S1:in BIT;
         I:    out BIT );
end component;

component NOT                  -- NOT atearen definizioa
   port( S0:   in BIT;
         I:    out BIT );
end component;

signal X: BIT;                 -- barne-seinale baten definizioa

begin                          -- Konparagailuaren definizioa
   U0: XOR port map (S0 => A, S1 => B, I => X);
   U1: NOT port map (S0 => X,  I => Y);
end DEF4;
```

Definizio horretan, bi "unitate" edo osagai erabiltzen dira, U0 eta U1, eta haien arteko konexioak zehazten dira (`port map`). Hala, XOR atearen sarrerak eta irteerak `A`, `B` eta `X` dira, eta NOT atearenak `X` eta `Y`, hurrenez hurren. `A`, `B` eta `Y`, blokearen sarrerak eta irteerak dira, baina `X` barne-seinalea da, bloke horretatik kanpo eskuraezina; hori dela eta, bertan definitu behar izan da, `signal` moduan. Baina osagaien arteko konexioak zehazteko orduan, etiketak erabiltzen dira eta ez dira bereizten sarrerak, irteerak edo barne-seinaleak. Esaterako, NOT atearen sarrera, `S0`, `X` barne-seinalea dela adierazten da (U1 unitatean, `S0 => X`), hots, XOR atearen irteera (U0 unitatean, `I => X`).

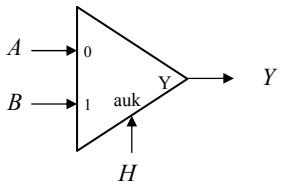
Bit bateko konparagailua definitu dugu dagoeneko: `entity` eta `architecture` (hiru aukeretatik bat) erabiliz. Hortik aurrera, nahi izanez gero, nahi dugun guztietan erabili ahal izango dugu bloke hori.

Lehen esan dugun bezala, oso zabala da VHDL lengoaia eta ez da liburu honen helburua lengoaia hori deskribatzea eta erabiltzea. Hala ere, merezi du aztertzea nola definitzen diren liburuan azaldu ditugun bloke konbinazional eta sekuentzial nagusiak. Edozein programazio-lengoaiatan ohikoak diren datu- eta kontrol-egiturak erabiltzen dira zirkuituak definitzeko, eta, horregatik, irakurleak ez du arazorik izango definizio horiek ulertzeko. Esan dugun moduan, aukera bat baino gehiago dago bloke bat definitzeko. Beraz, adibide soilak dira hurrengo ataletan azalduko ditugunak.

E3.3. OHIKO BLOKE KONBINAZIONALAK

E3.3.1. Multiplexoreak

Ikus ditzagun bi adibide. Lehenengoa, bi sarrerako multiplexorea (gaikuntza-seinalerik gabe). Hauek dira multiplexore horren sarrerak eta irteerak: datu-sarrerak, *A* eta *B*; hautatze-lerroa, *H*; eta irteera, *Y*. Seinale guztiak bit batekoak dira.



$H = 0$ denean, $Y = A$ izango da, eta $H = 1$ denean, $Y = B$ izango da. Beraz, $Y = \overline{H}\ A + H\ B$. Honela definituko dugu multiplexorea VHDL erabiliz:

1. Blokearen sarreren eta irteeren definizioa (`entity`):

```
entity 2:1-MUX is
   port( A,B,H:   in BIT;
         Y:       out BIT );
end 2:1-MUX;
```

2. Blokearen funtzionamenduaren logika (`architecture`):

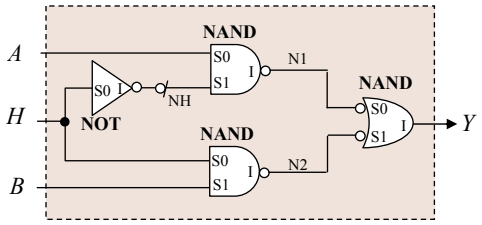
```
architecture DEF1 of 2:1-MUX is
begin
   process (A,B,H)
   begin
      if (H = '0')   then Y <= A;
                     else Y <= B;
      end if;
   end process;
end DEF1;
```

Nahi izanez gero, honela ere adieraz daiteke multiplexorearen funtzionamendua, oinarrizko funtzio logikoak erabiliz:

```
architecture DEF2 of 2:1-MUX is
begin
   Y <= ((not (H)) and A) or (H and B) after 2 ns;
end DEF2;
```

Ageri den moduan, multiplexorearen erantzun-denbora ere definitu dugu: 2 ns.

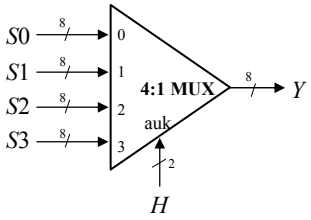
Azkenik, hau litzateke hirugarren aukera bat, multiplexorearen osagaiak eta haien arteko konexioak kontuan hartuz:




```
architecture DEF3 of 2:1-MUX is

signal NH,N1,N2: BIT;

component NOT
   port (S0: in BIT; I: out BIT);
end component;

component NAND
   port (S0,S1: in BIT; I: out BIT);
end component;


begin
   U0: NOT port map  (S0 => H,    I => NH);
   U1: NAND port map (S0 => A,  S1 => NH, I => N1);
   U2: NAND port map (S0 => H,  S1 => B,  I => N2);
   U3: NAND port map (S0 => N1, S1 => N2, I => Y);
end DEF3;
```

Nahikoa. `Entity` definizioa eta `architecture` motako definizioren bat hartuta, 2 sarrerako multiplexore bat zehaztu dugu.

Ikus dezagun bigarren adibide bat: 8 biteko 4 sarrerako multiplexorea. Sarrerak (*S*0, *S*1, *S*2 eta *S*3) eta irteera (*Y*) 8 bitekoak dira. 4 sarreretatik bat aukeratzeko, 2 biteko kode bat behar da, *H*.



Hau da bloke horren definizio bat VHDL erabiliz:

1. Blokearen sarreren eta irteeren definizioa (`entity`):

```
entity 4:1-MUX(8) is
   port( S0,S1,S2 S3:   in BIT_VECTOR(7 downto 0);
         H:     in BIT_VECTOR(1 downto 0);
         Y:     out BIT_VECTOR(7 downto 0) );
end 4:1-MUX(8);
```

   Definizio horretan, datu-mota berri bat ageri da: `BIT_VECTOR`. Bit bat baino gehiagoko seinaleak definitzeko erabiltzen da, kasu honetan 8 biteko eta 2 biteko seinaleak.

2. Blokearen funtzionamenduaren logika (`architecture`).

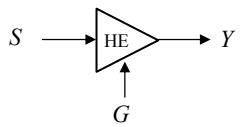
   Definizio bakar bat emango dugu, portaera motakoa. Nahi izanez gero, irakurleak ez du arazorik izango beste biak sortzeko.

```
architecture DEF1 of 4:1-MUX(8) is
begin
   process (H,S0,S1,S2,S3)
   begin
      case H is
         when "00" => Y <= S0;
         when "01" => Y <= S1;
         when "10" => Y <= S2;
         when "11" => Y <= S3;
      end case;
   end process;
end DEF1;
```

   `case` kontrol-egituraren bidez, *H* sarrerak har ditzakeen balio guztiak analizatzen dira, eta bakoitzari erantzun jakin bat ematen zaio.

E3.3.2. Hiru egoerako bufferrak

Hiru egoerako buffer batek honako funtzio hau betetzen du: aktibatuta badago, sarrerako balioa ematen du irteeran; bestela, irteera Z egoeran (inpedantzia altua edo deskonexio birtuala) uzten du.



1. Blokearen sarreren eta irteeren definizioa (`entity`):

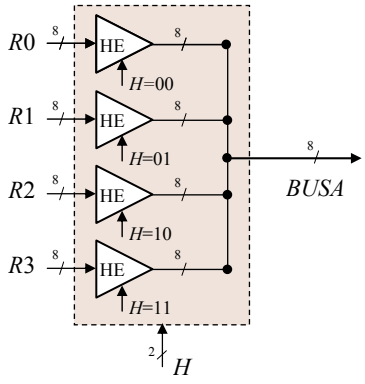
```
entity HE is
   port( G,S:  in STD_LOGIC;
         Y:    out STD_LOGIC);
end HE;
```

Kasu honetan, `G`, `S` eta `Y` seinaleak definitzean, ez dugu `BIT` datu-mota erabili, `std_logic` baizik, logika estandarra, alegia. Arrazoia honako hau da: nahiz eta seinale batek bi balio logiko baino ezin dituen hartu (0 edo 1), beste balio batzuk ere hartu behar dira kontuan sistema errealetan; esaterako, Z egoera (inpedantzia altua) hiru egoerako zirkuituetan. Z balioa eta beste batzuk —esaterako, U (*uninitialized,* hasieratu gabe) eta X (ezezaguna)— `std_logic` izeneko balio multzoan daude definituta.

2. Blokearen funtzionamenduaren logika (`architecture`):

```
architecture DEF1 of HE is
begin
   process (G, S)
   begin
      if(G = '1') then  Y <= S;
                  else  Y <= 'Z';
      end if;
   end process;
end DEF1;
```

Hiru egoerako gailuak erabiliz, hainbat gailuren irteerak konekta daitezke bus komun batera, baldin eta gehienez horietako bakar bat aktibatuta badago, gainerakoak Z egoeran izanik. Neurri batean, multiplexore baten antzeko funtzioa egiten du:



Honela definituko genuke aurreko irudiko “bus-konektorea”:

1. Blokearen sarreren eta irteeren definizioa (`entity`):

```
entity BUS-KONEKTOREA is
   port( R0,R1,R2,R3:   in STD_LOGIC_VECTOR (7 downto 0);
         H:    in STD_LOGIC_VECTOR (1 downto 0);
         BUSA: out STD_LOGIC_VECTOR (7 downto 0) );
end BUS-KONEKTOREA;
```

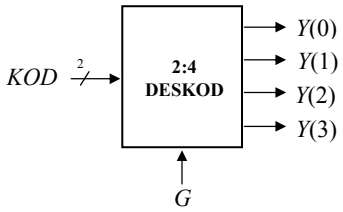
   Definizio horretan, `R0`, `R1`, `R2` eta `R3` 8 biteko datu-sarrerak dira (`in`); `BUSA`, 8 biteko datu-irteera (`out`); eta `H`, 2 biteko kontrol-sarrera (`in`).

2. Blokearen funtzionamenduaren logika (`architecture`):

```
architecture DEF1 of BUS-KONEKTOREA is
begin
   process (H)
   begin
      case (H) is
         when "00" => BUSA <= R0;
         when "01" => BUSA <= R1;     -- G barne-seinaleari
         when "10" => BUSA <= R2;     -- dagokion balioa eman
         when "11" => BUSA <= R3;
         when others => BUSA <= 'ZZZZZZZZ';
      end case;
   end process;
end DEF1;
```

E3.3.3. Deskodegailuak

Ikus dezagun nola definitu 2:4 deskodegailu bat VHDL erabiliz.



1. Blokearen sarreren eta irteeren definizioa (`entity`):

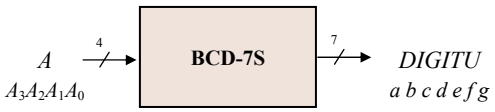
```
entity 2:4-DESKOD is
   port( G:    in BIT;
         KOD:  in INTEGER range 0 to 3;
         Y:    out BIT_VECTOR (3 downto 0) );
end 2:4-DESKOD;
```

2. Blokearen funtzionamenduaren logika (`architecture`):

```
architecture DEF1 of 2:4-DESKOD is
begin
   process (KOD,G)
   begin
      if (G = '0') then
         Y <= (others => '0');        -- irteera guztiak 0
      else
         for i in 0 to 3 loop
            if KOD = i then
                  Y(i) <= '1';        -- aktibatu i irteera
            else  Y(i) <= '0';
            end if;
         end loop;
      end if;
   end process;
end DEF1;
```

KOD eta *G* sarrerak prozesatzen dira. Gaikuntza-seinalea 0 bada, orduan irteera guztiak 0 dira; bestela, *Y* irteera bakar bat aktibatzen da, `for` begizta baten barruan, *KOD* sarreraren balioaren arabera.

Ikus dezagun beste adibide bat: "BCD – 7 segmentu" deskodegailua. 4 biteko sarrera-kode (0tik 9ra) bati dagozkion digitu baten segmentuak (*a*, ..., *g*) piztu behar dira (ikus 1.8. ariketa).



1. Blokearen sarreren eta irteeren definizioa (`entity`):

```
entity BCD-7S is
   port( A:       in BIT_VECTOR(3 downto 0);
         DIGITU:  out BIT_VECTOR(6 downto 0) );
end BCD-7S;
```

2. Blokearen funtzionamenduaren logika (`architecture`):

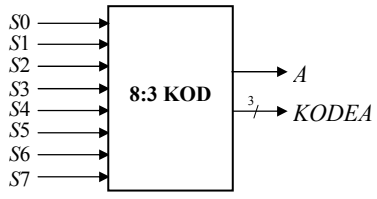
```
architecture DEF1 of BCD-7S is
begin
   process (A)
   begin
      case (A) is
         when "0000" => DIGITU <= "1111110" ; -- 0
         when "0001" => DIGITU <= "0110000" ; -- 1
         when "0010" => DIGITU <= "1101101" ; -- 2
         when "0011" => DIGITU <= "1111001" ; -- 3
         when "0100" => DIGITU <= "0110011" ; -- 4
         when "0101" => DIGITU <= "1011011" ; -- 5
         when "0110" => DIGITU <= "0011111" ; -- 6
         when "0111" => DIGITU <= "1110000" ; -- 7
         when "1000" => DIGITU <= "1111111" ; -- 8
         when "1001" => DIGITU <= "1110011" ; -- 9
         when others => DIGITU <= "0000000" ; -- (A > 10)
      end case;
end DEF1;
```

Adibidez, 1 zenbakia adierazteko zazpi segmentuzko digitu batean, *b* eta *c* segmentuak piztu behar dira. Horregatik, *DIGITU* = 0110000 izan behar du.

Sarrerako datua ez bada 4 biteko BCD zenbaki bat (>9), digituaren segmentu guztiak itzalita utzi dira.

E3.3.4. Kodegailuak

Adibide gisa, honela deskriba daiteke, VHDL erabiliz, 8:3 kodegailu sinple bat. Sarreraren bat ez badago aktibatuta, *A* irteera 0 da. Bi sarrera baino gehiago aktibatuta badaude, “altuenaren” kodea emango du (ikus 3.4. atala).



1. Blokearen sarreren eta irteeren definizioa (`entity`):

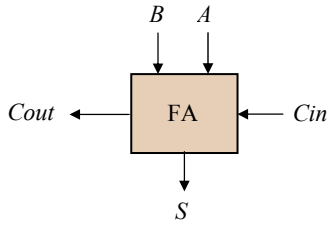
```
entity 8:3-KOD is
   port( S0,S1,S2,S3,S4,S5,S6,S7: in BIT;
         KODEA: out BIT_VECTOR(2 downto 0);
         A: out BIT);
end 8:3-KOD;
```

2. Blokearen funtzionamenduaren logika (`architecture`):

```
architecture DEF1 of 8:3-KOD is
begin
   process (S0 ,S1,S2,S3,S4,S5,S6,S7)
   begin
      if (S7 = '1')     then  KODEA <= "111"; A <= '1';
      elsif (S6 = '1')  then  KODEA <= "110"; A <= '1';
      elsif (S5 = '1')  then  KODEA <= "101"; A <= '1';
      elsif (S4 = '1')  then  KODEA <= "100"; A <= '1';
      elsif (S3 = '1')  then  KODEA <= "011"; A <= '1';
      elsif (S2 = '1')  then  KODEA <= "010"; A <= '1';
      elsif (S1 = '1')  then  KODEA <= "001"; A <= '1';
      elsif(S0 = '1')   then  KODEA <= "000"; A <= '1';
      else                    KODEA <= "000"; A <= '0';
      end if;
   end process;
end DEF1;
```

E3.3.5. Batugailuak

Bit bateko batugailuak (FA, *full adder*) hiru sarrera —*A*, *B* eta *Cin*— eta bi irteera—*S* eta *Cout*— ditu.



Beraz, honela deskribatuko dugu:

1. Blokearen sarreren eta irteeren definizioa (`entity`):

```
entity BATUG-1b is
   port( A,B,Cin: in BIT;
         S,Cout:  out BIT);
end BATUG-1b;
```

2. Blokearen funtzionamenduaren logika (`architecture`):

   Batugailuaren portaera logikoa honela defini daiteke, egia-taulatik abiatuta (ikus 3.5. atala):

```
architecture DEF1 of BATUG-1b is
begin
   process (A,B,Cin)
   begin
      if (A='0' and B='0' and Cin='0')
         then  S <= '0'; Cout <= '0';
      elsif (A='0' and B='0' and Cin='1') or
            (A='0' and B='1' and Cin='0') or
            (A='1' and B='0' and Cin='0')
         then S <= '1'; Cout <='0';
      elsif (A='0' and B='1' and Cin='1') or
            (A='1' and B='0' and Cin='1') or
            (A='1' and B='1' and Cin='0')
         then  S <= '0'; Cout <= '1';
      elsif (A='1' and B='1' and Cin='1')
         then S <= '1'; Cout <= '1';
      end if;
   end process;
end DEF1;
```

Agian errazagoa da, kasu honetan, beste deskripzio mota bat erabiltzea, batugailuaren ekuazio logikoak kontuan hartuz (*dataflow*):

$S = A \text{ xor } B \text{ xor } Cin$

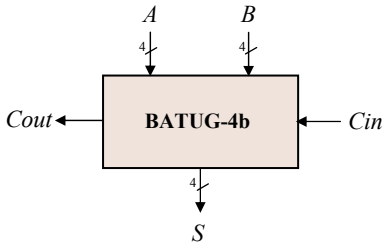
$Cout = A\ B + Cin\ (A \text{ xor } B)$

Beraz, honela eman daiteke zirkuituaren funtzionamendua:

```
architecture DEF2 of BATUG-1b is
begin
   S <= A xor B xor Cin after 10 ns;
   Cout <= (A and B) or ((A xor B) and Cin) after 10 ns;
end DEF2;
```

Aipatu dugun moduan, modulu bat definituta, nahi den guztietan erabil daiteke beste zirkuitu batzuk sortzeko. Horrela, diseinu-prozesua modu hierarkikoan egin daiteke: bloke sinpleak definitu; horiek erabiliz, modulu konplexuak sortu; eta abar.

Esaterako, bit bateko batugailuak erabiliz, 4 biteko batugailu bat defini daiteke. Batugailu horren egitura oso sinplea da: *i* biteko batugailuaren *Cin* sarrera *i*–1 biteko batugailuaren *Cout* irteera da.



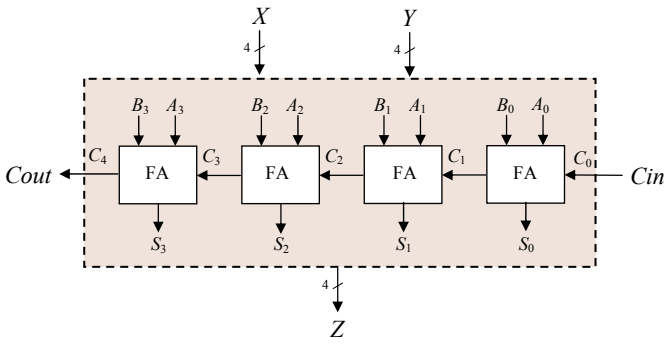
Beraz, hau da 4 biteko batugailuaren definizioa, bit bateko batugailuak erabiliz:

1. Blokearen sarreren eta irteeren definizioa (`entity`):

```
entity BATUG-4b is
   generic( N: integer := 4;          -- konstanteen definizioa
            M: time := 10 ns);
   port( A,B:  in BIT_VECTOR (N-1 downto 0);
         Cin:  in BIT;
         S:    out BIT_VECTOR (N-1 downto 0);
         Cout: out BIT);
end BATUG-4b;
```

   Orain arte erabili ez ditugun osagaiak ageri dira aurreko definizioan. Esaterako, `generic` definizioaren bidez, bi parametro definitu ditugu: `N`, prozesatzen diren datuen bit kopurua; eta `M`, erabili nahi dugun atzerapen-denbora. Hala, bi batugaiak eta batura `N` bitekoak dira, kasu honetan 4koak.

2. Blokearen funtzionamenduaren logika (`architecture`):

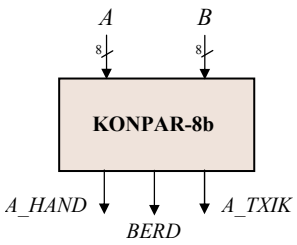


```
architecture DEF1 of BATUG-4b is

signal X,Y,Z:  BIT_VECTOR (3 downto 0);
signal Cout:   BIT;
signal TMP:    BIT_VECTOR (4 downto 0);

component BATUG-1b
   port( A,B,Cin: in BIT;
         S,Cout:  out BIT);
end component;

begin
   TMP (0) <= '0';
   G: for I in 0 to 3 generate
      FA: BATUG-1b port map (
         A => X(I), B => Y(I), Cin => TMP(I),
         S => Z(I), Cout => TMP(I+1) );
   end generate;
   Cout <= TMP(4);
end DEF1;
```

Bit batetik bestera igarotzen den bururakoa adierazteko, `TMP` bit-bektorea definitu dugu. `generate` funtzioarekin hau egiten da: bit bateko lau batugailu sortu eta haien konexioak definitu.

E3.3.6. Konparagailuak

Ikus dezagun nola definitu 8 biteko konparagailua. Sarrerak, *A* eta *B*, 8 bitekoak dira, eta hiru irteerak, *A_HAND, BERD* eta *A_TXIK*, bit batekoak.



Hau da bloke horren definizioa VHDL erabiliz:

1. Blokearen sarreren eta irteeren definizioa (`entity`):

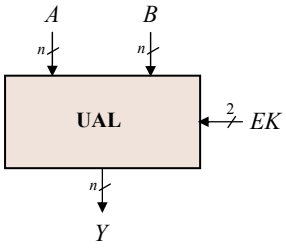
```
entity KONPAR-8b is
   port( A,B: in BIT_VECTOR(7 downto 0);
         A_HAND, BERD, A_TXIK: out BIT);
end KONPAR-8b;
```

2. Blokearen funtzionamenduaren logika (`architecture`):

```
architecture DEF1 of KONPAR-8b is
begin
   process (A, B)
   begin
      if (A>B)    then A_HAND <= '1'; BERD <= '0'; A_TXIK <= '0';
      elsif (A=B) then A_HAND <= '0'; BERD <= '1'; A_TXIK <= '0';
                  else A_HAND <= '0'; BERD <= '0'; A_TXIK <= '1';
      end if;
   end process;
end DEF1;
```

E3.3.7. Unitate aritmetiko/logikoak

Har dezagun, adibide gisa, 4 eragiketa egiten dituen unitate aritmetiko/logiko sinple bat; 2 biteko eragiketa-kode baten arabera, honako funtzio hauek egingo ditu: (00) → *A* + *B*; (01) → *A* – *B*; (10) → *A and B*; (11) → *A or B*.



Honela egin daiteke 16 biteko UALaren definizioa VHDL erabiliz:

1. Blokearen sarreren eta irteeren definizioa (`entity`):

```
entity UAL-16b is
   generic( N: integer := 16;         -- konstanteen definizioa
            M: time := 10 ns);
   port( A,B:  in BIT_VECTOR (N-1 downto 0);
         EK:   in BIT_VECTOR (1 downto 0);
         Y:    out BIT_VECTOR (N-1 downto 0));
end UAL-16b;
```

2. Blokearen funtzionamenduaren logika (`architecture`):

```
architecture DEF1 of UAL-16b is
begin
   process (A,B,EK)
   begin
      case EK is
         when "00" =>   Y <= A + B;
         when "01" =>   Y <= A - B;
         when "10" =>   Y <= A and B;
         when "11" =>   Y <= A or B;
      end case;
   end process;
end DEF1;
```

E3.4. BLOKE SEKUENTZIALAK

Ohiko zirkuitu sekuentzialen VHDL definizioak emango ditugu. Hasteko, erloju-seinalearen definizioa; gero, D eta JK biegonkorrak eta erregistro bat, desplazamendu-erregistro bat eta kontagailu bat.

E3.4.1. Erloju-seinalea

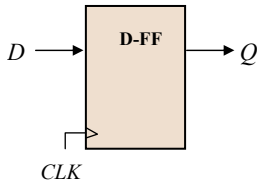
Etengabe oszilatzen duen seinalea da erlojua. Eskuarki, erloju-seinalea kanpoko seinale bat da. Hala ere, beharrezkoa bada oszilatzen duen seinale bat izatea sistema batean, honela egin daiteke:

```
process
begin
   OSZ <= not OSZ;
   wait for 10 ns
end process;
```

Hau da, 10 ns itxaron ondoren, OSZ seinalearen balioa aldatzen da (“kommutatu”), une horretan duen balioa ezeztatuz (`not OSZ`): 0tik 1era, edo 1etik 0ra. Aldaketaren ondoren, berriz ere, beste 10 ns zain geratuko da, hurrengo aldaketa egin baino lehen.

E3.4.2. D biegonkorrak

D biegonkor sinple bat definituko dugu, CLK erloju-seinaleaz gain, *D* sarrera eta *Q* irteera baino ez dituena:



1. Blokearen sarreren eta irteeren definizioa (`entity`):

```
entity D_FF is
   port( CLK:  in BIT;
         D:    in BIT;
         Q:    out BIT);
end D_FF;
```

2. Blokearen funtzionamenduaren logika (`architecture`):

```
architecture DEF1 of D_FF is
begin
   process
   begin
      wait until (CLK'event and CLK = '1');  -- goranzko ertza
      Q <= D;
   end process;
end DEF1;
```

Prozesua zain geratzen da erloju-seinalean aldaketa bat gertatu (`CLK'event`) eta balio berria 1 izan arte; hau da, 0→1 aldaketa (erlojuaren goranzko ertza) gertatu arte. Une horretan, prozesuak *D* sarrerako balioa *Q* irteerara pasatzen du.

Aipatu dugun bezala, ez dago modu bakar bat zirkuitu logiko baten portaera adierazteko. Esaterako, honela ere adieraz daiteke D biegonkor baten portaera:

```
architecture DEF2 of D_FF is
begin
   process (CLK)                        -- erlojuaren aldaketa
   begin
      if (CLK'event and CLK = '1')      -- goranzko ertza
         then Q <= D;
      end if;
   end process;
end DEF2;
```

Adierazten den prozesua soilik exekutatuko da *CLK* seinalean aldaketa bat gertatzen denean —`process (CLK)`—.

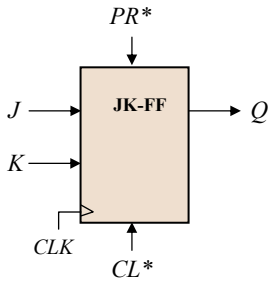
Demagun D biegonkorrak *CL* (*clear*) seinale asinkronoa ere baduela (gauzak sinplifikatzearren, suposatuko dugu logika positiboan dagoela). Honelaxe geratuko da D biegonkorraren arkitekturaren definizioa kasu horretan:

```
architecture DEF3 of D_FF_CL is
begin
   process (CLK, CL)
   begin
      if CL = '1'
         then Q <='0';                    -- clear asinkronoa
      elsif (CLK'event and CLK = '1')     -- erlojuaren ertza
         then Q <= D;
      end if;
   end process;
end DEF3;
```

Prozesua exekutatzeko, `CL` edo `CLK` seinaleen aldaketak hartzen dira kontuan. Halakoetan, lehenik `CL` seinalea aztertzen da; aktibatuta badago, biegonkorraren irteera 0ra eramango da. Bestela —`elsif`—, begiratzen da ea erlojuaren goranzko ertza gertatu den, biegonkorraren portaera sinkronoa betetzeko.

E3.4.3. JK biegonkorrak

JK biegonkor arrunt bat definituko dugu, *CLK* erloju-seinaleaz gain, *J* eta *K* sarrerak, *Q* irteera eta *CL* eta *PR* seinale asinkronoak dituena (denak logika positiboan[38]):



[38] Logika positiboa baino ez da erabiltzen VHDLn. Gauzak ez nahastearren, oraingoz logika positiboa bakarrik erabiliko dugu. Azken atalean ikusiko dugu nola tratatu logika negatiboa.

1. Blokearen sarreren eta irteeren definizioa (`entity`):

```
entity JK_FF is
   port( CLK,J,K,CL,PR: in BIT;
         Q: out BIT);
end JK_FF;
```

2. Blokearen funtzionamenduaren logika (`architecture`):

```
architecture DEF1 of JK_FF is
signal Qb: BIT;
begin
   Q <= Qb;                    -- une oro egin behar den esleipena
   process (CL, PR,CLK)
   begin
      if CL = '1'    then Qb <='0';          -- clear asinkronoa
      elsif PR = '1' then Qb <='1';          -- preset asinkronoa
      elsif (CLK'event and CLK = '1')
         then                                -- goranzko ertza
            if J&K = "00"     then Qb <= Qb;
            elsif J&K = "01"  then Qb <= '0';
            elsif J&K = "10"  then Qb <= '1';
            elsif J&K = "11"  then Qb <= (not Qb);
            else                   Qb <= Qb;
            end if;
      end if;
   end process;
end DEF1;
```

JK biegonkorraren funtzionamendua islatzeko, `Qb` barne-seinalea definitu dugu hasieran, VHDLn debekatuta baitago `port out` moduan definitutako aldagai bat (biegonkorraren `Q` irteera, kasu honetan) ekuazio baten eskuinaldean azaltzea (esaterako, `J` eta `K` 1 eta 1 direnean, `Q <= not Q` egin beharko genuke). Horregatik, prozesuaren barruan, `Qb` erabili behar da `Q`-ren ordez, baina ahaztu gabe `Q` irteerari `Qb` aldagaiaren balioa eman behar zaiola une oro: `Q <= Qb`. Hala, `Qb` aldagaiaren balioa aldatzean, `Q`-rena ere aldatuko da.

Beste aldetik, J eta K sarrerak independenteak diren arren, `J&K` adierazpena erabili dugu, modu horretan bi sarreren balioak batera adieraz daitezkeelako.

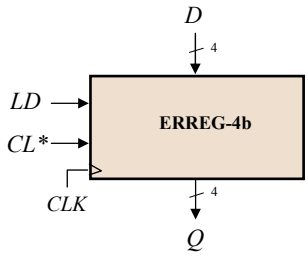
Azkenik, azpimarratu behar dugu CL sarrerak lehentasun handiagoa duela PR sarrerak baino, ordena horretan aztertzen baititu prozesuak: lehenik CL, gero PR, eta, azkena, erloju-ertza.

Funtzionamendu bera —honako hau CL eta PR sarrerarik gabe, sinplifikatzearren— honela ere isla daiteke (irakurleak modu errazean froga dezake hori):

```
architecture DEF2 of JK_FF is
signal Qb:  BIT;
begin
   Q <= Qb;
   process (CLK)
   begin
      if (CLK'event and CLK = '1') then        -- goranzko ertza
         Qb <= (J and not Qb) or (not K and Qb);
      end if;
   end process;
end DEF2;
```

E3.4.4. Erregistroak

4 biteko erregistro bat definituko dugu. Sarrerak: *D* (datuak), *LD* (karga-seinalea), *CLK* (erlojua) eta *CL* (*clear* asinkronoa). Irteerak: *Q* (datuak).



1. Blokearen sarreren eta irteeren definizioa (`entity`):

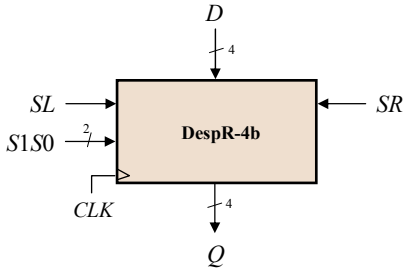
```
entity ERREG-4b is
   port( CLK:  in BIT;
         D:    in BIT_VECTOR (3 downto 0);
         LD,CL: in BIT;
         Q:    out BIT_VECTOR (3 downto 0));
end ERREG-4b;
```

2. Blokearen funtzionamenduaren logika (`architecture`):

```
architecture DEF1 of ERREG-4b is
begin
   process (CLK, CL)
   begin
      if CL = '1'    then Q <= "0000";     -- clear asinkronoa
      elsif (CLK'event and CLK = '1') then
         if LD = '1' then Q <= D;          -- karga sinkronoa
         end if;
      end if;
   end process;
end DEF1;
```

E3.4.5. Desplazamendu-erregistroak

4 biteko desplazamendu-erregistro bat definituko dugu. Sarrerak: *D* (4 biteko datua), *SL* eta *SR* (bit bateko datuak), *S*1 eta *S*0 (kontrol-seinaleak) eta *CLK* (erlojua). Irteera: *Q* (erregistroaren edukia, 4 bit).



1. Blokearen sarreren eta irteeren definizioa (`entity`):

```
entity DespR-4b is
   port( CLK:  in BIT;
         D:    in BIT_VECTOR (3 downto 0);
         S1,S0,SL,SR:   in BIT;
         Q:    out BIT_VECTOR (3 downto 0));
end DespR-4b;
```

2. Blokearen funtzionamenduaren logika (`architecture`):

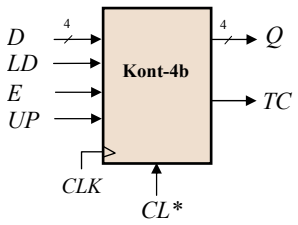
```
architecture DEF1 of DespR-4b is
signal Qb:     BIT_VECTOR (3 downto 0);
signal erag:   BIT_VECTOR (1 downto 0);
begin
   Q <= Qb;
   erag <= S1 & S0;
   process (CLK)
   begin
      if (CLK'event and CLK = '1') then
         case erag is
            when "00" =>  Qb <= Qb;
            when "01" =>  Qb <= Qb(2 downto 0) & SR;
            when "10" =>  Qb <= SL & Qb(3 downto 1);
            when "11" =>  Qb <= D;
         end case;
      end if;
   end process;
end DEF1;
```

`S1` eta `S0` seinaleak bi biteko `erag` izeneko aldagai batean “lotu” ditugu, eta, gero, `case` sententzia bat erabili aldagai horren balio guztiak aztertzeko.

Desplazamenduak egiteko, `Qb` barne-seinalearen bit batzuk eta `SR` edo `SL` sarrerak "lotu" ditugu (`Qb2Qb1Qb0&SR` edo `SL&Qb3Qb2Qb1`) eta `Qb` aldagaiari esleitu.

E3.4.6. Kontagailuak

Ikus dezagun kontagailu baten definizioa:



1. Blokearen sarreren eta irteeren definizioa (`entity`):

```
entity Kont-4b is
   port( CLK,CL,LD,E,UP: in BIT;
         D: in INTEGER range 0 to 15;
         Q: out INTEGER range 0 to 15;
         TC: out BIT);
end U/Dkont-4b;
```

2. Blokearen funtzionamenduaren logika (`architecture`):

```
architecture DEF1 of kont-4b is
signal Qb: INTEGER range 0 to 15;

begin
   Q <= Qb;
   TC <= '1' when (Qb = 15 and UP = '1') or
                  (Qb = 0 and UP = '0')
         else '0';

   process (CLK,CL)
   begin
      if CL = '1' then Qb <= 0;
      elsif (CLK'event and CLK = '1') then
         if LD = '1' then Qb <= D;
         else
            if E = '1' then
               if UP = '1'    then Qb <= Qb + 1;
                              else Qb <= Qb - 1;
               end if;
            end if;
         end if;
      end if;
   end process;
end DEF1;
```

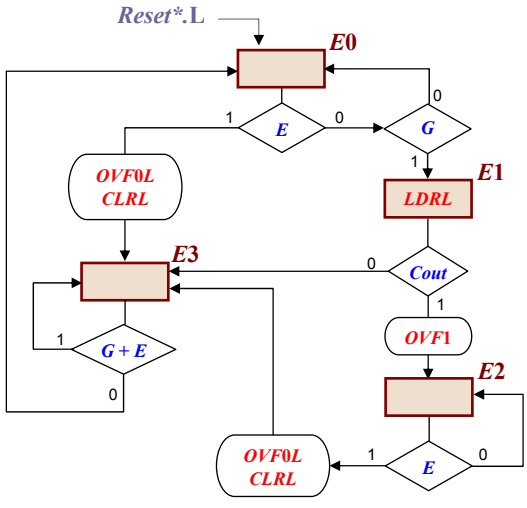
Ikusi dugunez, prozesutik kanpo bi esleipen daude, irteerari barne-seinalearen balioa esleitzen diona, eta `TC` seinalearen balioa kalkulatzen duen ekuazioa. Esleipen horiek beti exekutatzen dira, eta bertan azaltzen diren seinaleetariko baten bat aldatzean, emaitzak ere aldatuko dira.

Gero, erlojuaren eta `CL` seinalearen mende dagoen prozesua dator. Bertan, lehendabizi, `CL` asinkronoa aztertzen da. Ez badago aktibatuta, orduan, erlojuaren goranzko ertzean, `LD` seinalea aztertzen da, eta, azkenik, `E` eta `UP` seinaleak.

E3.4.7. Egoera-makinak

6. kapituluan azaldu dugun moduan, kontrol-automata batek (egoera-makina batek) kontrolatzen du sistema osoaren funtzionamendua. Automata horren logika ASM algoritmo baten bidez adieraz daiteke, eta, baita ere, VHDL lengoaia erabiliz.

Adibide gisa, ikus dezagun nola definitu honako kontrol-algoritmo hau VHDLren bidez.



Kontrol-automata beste edozein bloke bat bezala definitu behar da:

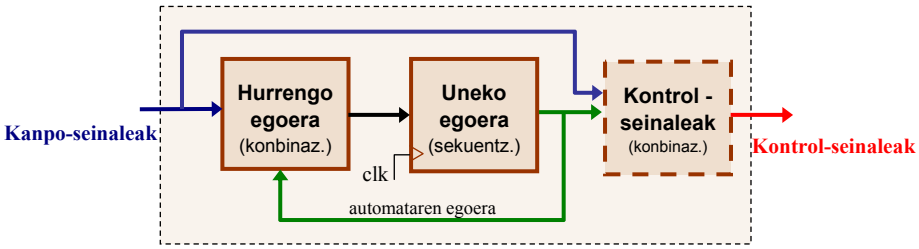
1. Blokearen sarreren eta irteeren definizioa (`entity`):

```
entity KU is
   port( CLK,RESETL: in STD_LOGIC;
         E,G,Cout:   in STD_LOGIC;
         CLRL,LDRL,OVF1,OVF0L: out STD_LOGIC;
         EGOERA:     out INTEGER range 0 to 3);
end KU;
```

Kontrol-unitate honetan, eta adibide gisa, seinale batzuk logika negatiboan erabiliko ditugu (L-z bukatzen direnak): `RESETL`, `CLRL`, `LDRL` eta `OVF0L`.

2. Blokearen funtzionamenduaren logika (`architecture`):

Dakigunez, kontrol-unitateak bi atal nagusitan banatzen dira: atal sekuentzial bat (biegonkorrak, eskuarki) automataren egoera gordetzeko, eta atal konbinazional bat hurrengo egoera adierazten duten funtzio logikoak sortzeko, bai eta, oro har, kontrol-seinaleak sortzeko.



Egitura hori VHDLz islatzeko asmoz, hiru atal bereizi ditugu `architecture` definizioan:

- prozesu bat hurrengo egoera kalkulatzeko, kanpoko seinaleen eta uneko egoeraren menpe:

  `KONB: process(UE,E,G,Cout)`

- prozesu bat uneko egoera gordetzeko, RESET seinalearen eta erloju-seinalearen menpe:

  `SEK: process(CLK,RESETL)`

- kontrol-seinaleak sortzeko atala, inongo prozesuren barruan, uneko egoeraren eta sarreren arabera, beti sortu behar direlako.

Hona hemen, beraz, algoritmo horri dagokion `architecture` definizioa.

```
architecture ALGORITMO of KU is
signal UE, HE: INTEGER range 0 to 3;

begin
   KONB:process (UE,E,G,Cout)  -- egoera kalkulatzeko prozesua
   begin
      case UE is
      when 0 =>   if E = '1'   then    HE <= 3;
                  elsif G = '1'then    HE <= 1;
                  else                 HE <= 0;
                  end if;
      when 1 =>   if Cout = '1' then   HE <= 2;
                  else                 HE <= 3;
                  end if;
      when 2 =>   if E = '0' then      HE <= 2;
                  else                 HE <= 3;
                  end if;
      when 3 =>   if (E or G) = '1' then HE <= 3;
                  else                 HE <= 0;
                  end if;
      end case;
   end process KONB;

   SEK:process (CLK, RESETL)
   begin
      if RESETL = '0' then UE <= 0;  -- RESET logika negatiboan
      elsif CLK'event and CLK = '1'  -- erlojuaren ertza
                  then  UE <= HE;
      end if;
   end process SEK;

   LDRL <= '0' when UE = 1 else '1';
   CLRL <= '0' when (UE = 0 or UE = 2) and E = '1' else '1';
   OVF1 <= '1' when UE = 1 and Cout = '1' else '0';
   OVF0L <= '0' when (UE = 0 or UE = 2) and E = '1' else '1';
   EGOERA <= UE;

end ALGORITMO;
```

Adi! Seinale batzuk logika negatiboan erabiltzeko, honako hau egin behar da: aktibatu behar denean, 0a (= L tentsioa) esleitu, eta, desaktibatu behar denean, 1a (= H tentsioa).

Honaino VHDL lengoaiaren aurkezpena. Sistema digital bat diseinatzeko erabili behar izanez gero, lengoaia hori zehatz-mehatz deskribatzen duten liburuetara jo beharko du irakurleak.

BIBLIOGRAFIA

Liburu asko erabil daitezke hemen tratatu ditugun gaiak osatzeko, sakontzeko edo beste ikuspuntu batetik analizatzeko. Liburu honetan egin dugun moduan, zenbait testutan konputagailu sinplifikatu bat erabiltzen da oinarrizko kontzeptuak argiago azaltzeko; antzekoak baina desberdinak dira konputagailu didaktiko horiek, eta egokia izan daiteke haien funtzionamendua, agindu-multzoa, kodeketa-estrategiak eta abar aztertzea.

Hona hemen aukeratu dugun liburu-zerrenda:

1. Winkel, D.; Prosser, F.: *The Art of Digital Design*, Prentice Hall, 2. ed., 1987.

2. Floyd, T.L.: *Digital Fundamentals*, Prentice Hall, 8. ed., 2003.
(Gaztelaniaz: *Fundamentos de Sistemas Digitales*, 7. ed., P.H. 2000)

3. Ercegovac, M.; Lang, T.; Moreno J.H.: *Introduction to Digital Systems*, John Wiley & Sons, 1999.

4. Roth, C.H.: *Fundamentals of Logic Design*, Thomson, 5. ed., 2004.
(Gaztelaniaz: *Fundamentos de Diseño Lógico*, Thomson, 2004)

5. Gajski, D.J.: *Principles of Digital Design*, Prentice Hall, 1996.
(Gaztelaniaz: *Principios de Diseño Digital*, P.H. 1997)

6. Morris Mano, M.; Kime, C.R.: *Logic and Computer Design Fundamentals*, Prentice Hall, 3. ed., 2005.
(Gaztelaniaz: *Fundamentos de Diseño Lógico y Computadoras*, P.H, 1. ed., 1998)

7. Comer, D.J.: *Digital Logic and State Machine Design*, Saunder College Pub., 1995.

8. Wakerly, J.F.: *Digital Design. Principles and Practices*, Prentice Hall, 3. ed., 2001.
(Gaztelaniaz: *Diseño Digital*, P.H, 3. ed., 2001)

9. Hayes, J.P.: *Introduction to Digital Logic Design*, Addison-Wesley, 1993.
(Gaztelaniaz: *Introducción al Diseño Lógico Digital.* Addison-Wesley Iberoam., 1996)

Liburuan diseinatzen den BIRD konputagailuaren funtzionamendua, agindu-multzo osoa eta programazioa honako liburu honetan ageri dira:

10. Arbelaitz, O.; et al.: *Makina-Hizkuntza. Oinarrizko konputagailu baten egitura, agindu-multzoa eta programazioa*, UPV/EHU, 2004.

Beste gai batzuei buruzko informazioa sakontzeko liburuak:

11. Ercegovac, M.; Lang, T.: *Digital Arithmetic*, Morgan Kaufmann, 2004.

12. Rabaey, J.M.; Chandrakasan, A.; Nikolic, B.: *Digital Integrated Circuits*, Prentice Hall, 2. ed., 2003.
(Gaztelaniaz: *Circuitos Integrados Digitales*, Pearson, 2004)

13. Hamblen, J.O.; Furman, M.D.: *Rapid Prototyping of Digital Systems. A Tutorial Approach*, Kluwer Academic Pub., 2. ed., 2001.

14. Pérez, S.A.; Soto, E.; Fernández, S.: *Diseño de Sistemas Digitales con VHDL*, Thomson, 2002.

AURKIBIDE ALFABETIKOA

C

D

E

F

G

H

I

J

K