eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Industria-Informatikako Oinarriak

Egilea: Patxi Alkorta Egiguren

EUSKARA ERREKTOREORDETZAREN SARE ARGITALPENA

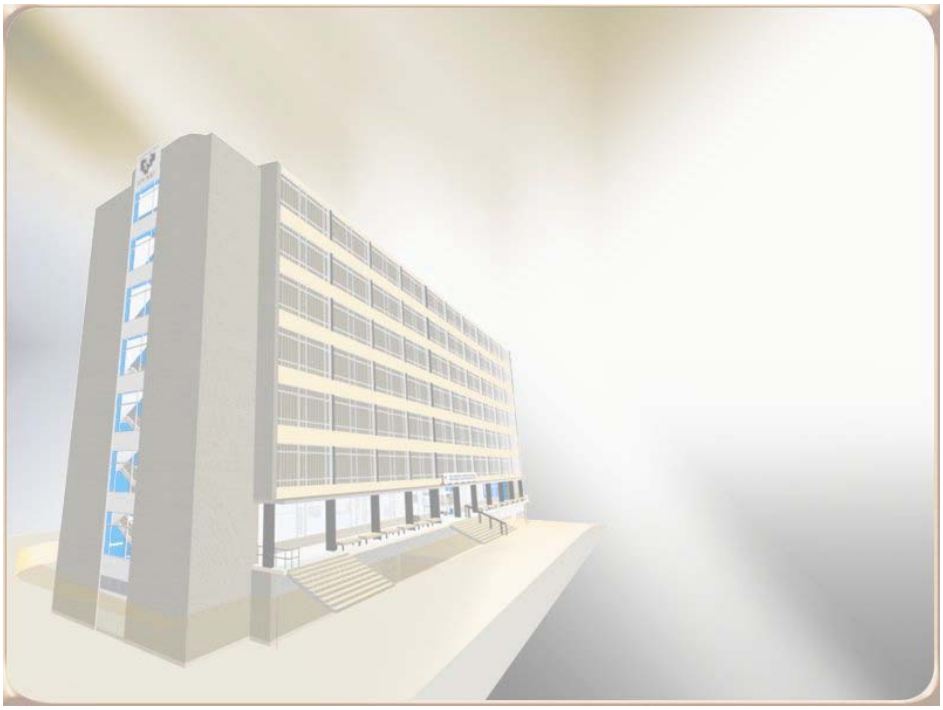
Liburu honek UPV/EHUko Euskara Errektoreordetzaren dirulaguntza jaso du

eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

SIAS/DISA
Departamento Ingeniería de Sistemas y Automática
Sistemen Ingeniaritza eta Automatika Saila

Escuela de Ingeniería de Eibar
Eibarko Ingeniaritza Eskola



Patxi Alkorta Egiguren EHUko irakaslearen
"FUNDAMENTOS DE INFORMÁTICA INDUSTRIAL"
(Ministerio de Cultura, Número de Registro de la Propiedad Intelectual BI-286-07)
lanaren euskarazko itzulpena.

ISBN: 978-84-691-5310-9
2008

Egile eta euskaratzaileak:
Patxi Alkorta Egiguren (egilea eta euskaratzailea)
Andoni Sarriegi Eskisabel (euskaratzailea)

Liburu honek UPV/EHUko Euskara Errektoreordetzaren dirulaguntza jaso du

AURKIBIDEA

1. ZIRKUITU DIGITALAK 1

1.1 ZIRKUITU KONBINAZIONALA 1
1.2 ZIRKUITU SEKUENTZIALA 1
1.2 MIKROPROZESADOREA (µP) 1
1.4 KABLEATUTAKO ETA PROGRAMATUTAKO LOGIKA-SISTEMAK 2

2. MIKROPROZESADOREA 4

2.1 MIKROPROZESADOREAREN KANPO-ARKITEKTURA 4
2.2 OINARRIZKO SISTEMA EDO GUTXIENEKO SISTEMA. 5
2.3 MEMORIA 6
2.4 S/I (I/O) UNITATEA (SARRERA-IRTEERAK). 7
2.5 PUZ/CPUA (PROZESATZEKO UNITATE ZENTRALA) 7
2.6 MIKROPROZESADOREAREN SOFTWAREA 8

3. iNTEL ETXEAREN 8051 MIKROKONTROLATZAILEA 11

3.1 PINAK (ORRATZAK) 11
3.2 MEMORIAREN ANTOLAKUNTZA 13
3.2.1 Barruko memoria (barne-memoria) 13
3.2.2 Kanpo-memoria 15
3.3 AGINDU MULTZOA 15

4. 8051 µC-AREN BARRUKO HARDWAREA 17

4.1 ETENDURAK 17
4.1.1 Etenduren gaiketa 20
4.1.2 Lehentasun-mailak 20
4.1.3 Aldiberekoak ez diren bi etendura edo gehiagoren eskaera 20
4.1.4 Aldibereko bi etendura edo gehiagoren eskaera. 22
4.1.5 Etendura-prozesua 22
4.2 TENPORIZADOREAK ETA KONTAGAILUAK 23
4.2.1 T0 tenporizadorea eta T1 tenporizadorea 23
Tenporizadore gisako funtzionamendua 24
Kontagailu gisako funtzionamendua 24
Funtzionamendu moduak 25
Tenporizadoreak erabiltzeko moduak 27
4.2 2 Zakur zaintzailea (watch-dog) 27
4.3 UART SERIEKO ATAKA (HARTZAILE-TRANSMISORE ASINKRONO UNIBERTSALA) 28
4.3.1 Komunikazio-ereduak 29
4.3.2 Serieko komunikazio asinkronoa 30
Hartzailearen laginketa-prozesua 30
4.3.3 Serieko komunikazio sinkronoa 30
4.3.4 8051 µC-aren UART Serieko Ataka 31
Funtzionamendu moduak 33
Azken ondorioak 37
4.3.5 Serieko komunikazioa paritate bikoitiarekin eta paritate bakoitiarekin 37
Paritate bikoitiaren metodoa 37
Paritate bakoitiaren metodoa 38
4.3.6 Komunikazioa multiprozesadorearen ingurunean 38
4.4 S/I (I/O) PINAK 40
4.4.1 Ia bi noranzkoko pina 40
4.4.2 Noranzko biko pina 41
4.4.3 Pinen karga-ahalmena 42
4.4.4 Soinu-sorkuntza 8051 µC-arekin 43
4.4.5 Matrize-teklatuaren konexioa 44

5. C PROGRAMAZIO-HIZKUNTZA 47

5.1 SARRERA 47
5.1.1 Jatorria 47
5.1.2 Gaur egun 47
5.1.3 8051 µC-arentzako C hizkuntza 48
5.2 OINARRIZKO ARAUAK 48
5.2.1 Aipamenak (iruzkinak) eta kode-lerroak 48
5.2.2 Aldagaien eta funtzioen izenak. Erazagupenak 48
5.2.3 Datu motak, kalifikatzaileak eta eragileak 49
5.3 C51 KONPILADOREAREN ATAL GARRANTZITSUAK 50
5.3.1 Memoria motak 50
5.3.2 Memoria-ereduak 50
5.3.3 C hizkuntzaren preprozesadorea 51
5.4 BILTEGIRATZE MOTAK 52
5.4.1 Aldagai lokalak 52
5.4.2 Aldagai orokorrak 52
5.4.3 Aldagai estatikoak 53
5.4.4 Erregistro-aldagaiak 53
5.5 ERAKUSLEAK 54
5.6 FUNTZIOAK 56
5.6.1 Funtzio motak 56
5.6.2 Funtzioen arteko komunikazioak 57
5.6.3 Parametro-pasatzea 59
5.7 EGITURAK 60
5.7.1 Egituren definizioa, erazagupena eta erabilera 60
5.7.2 Egituren erakusleak 63
5.8 PROGRAMAREN FLUXU-KONTROLA 63
5.8.1 Hautapen-egiturak 63
5.8.2 Iterazio-egiturak 65
5.9 ARRAY EDO TAULA DIMENTSIOANIZTUNAK 67
5.10 BIT MOTAKO ALDAGAIAK 68
5.11 KONSTANTEAK 69
5.12 CASTING EDO DATU MOTA BEHARKETAK C HIZKUNTZAN 70
5.13 C HIZKUNTZAKO DATU MOTEN TAMAINAK MAKINAREN ARABERA 74

6. PROGRAMAZIO MOTAK 72

6.1 PROGRAMAZIO LINEALA 72
6.2 MODULUZKO PROGRAMAZIOA 72
6.3 PROGRAMAZIO EGITURATUA 72
6.4 OBJEKTUEI ZUZENDUTAKO PROGRAMAZIOA (OZP) 73
6.5 C++ PROGRAMAZIO-HIZKUNTZARAKO SARRERA 74
6.5.1 C hizkuntzaren hedapenak objektuei zuzendutako programazioarekin (OZP) erlaziorik izan gabe 74
Aipamenak (iruzkinak) 74
Funtzioen prototipoak 75
Casting-ak edo mota-bihurketak 75
Egiturak 75
Aldagaien erazagupena 75
Balio bidezko parametro-pasatzea 76
Overloading: funtzioen gainzama 77
6.5.2 C hizkuntzaren hedapenak objektuei zuzendutako programazioarekiko (OZP) 77
Sarrera 77
Objektu-definizioa 79
Objektuei zuzendutako programa 79
Ondoretasuna (Herentzia) 79
Polimorfismoa 80
Berrerabilera 80

7. ORDENAGAILU PERTSONALAK 81

7.1 INTEL®-EN 80X86 µP FAMILIA 81

7.1.1 Intel-en 8086 µP-a ... *81*
Sarrera ... 81
Barne-erregistroak eta memoria-helbideratzea ... 82
Etendura-sistema ... 85
Hardware-etendura (edo kanpokoa) ... 86
Software-etendura ... 87
7.1.2 8088 µP-a ... *88*
Sarrera ... 88
Memoria-helbideen eta S/I (I/O) helbideen deskodetzea ... 89
8086/8088 µP-aren berrasieratzea ... 89
7.1.3 80188/80186 µC-ak ... *89*
DMA (Zuzeneko memoria-atzipena) ... 90
7.3.4 80286 µP-a ... *91*
Sarrera ... 91
Multiataza ... 92
Memoria birtuala edo alegiazko memoria ... 92
Lanerako moduak ... 92
7.1.5 80386DX µP-a ... *93*
7.1.6 80486DX µP-a ... *94*
Aurrememoria edo cache memoria ... 94
Matematika-koprozesadorea ... 96
7.1.7 Pentium µP-a (80586) ... *97*
7.2 PC-EN HEDAPEN-BUSAK ... 98
7.2.1 XT edo 8 biteko ISA® (Industry Standard Architecture;Industriako Arkitektura Estandarra) hedapen-busa ... *99*
7.2.2 AT hedapen-busa edo 16 biteko ISA ... *99*
7.2.3 MCA® eta EISA® busak, Mikrokanal Arkitektura eta ISA Hedatua ... *100*
7.2.4 PCI® (Peripheral Component Interconection; osagai periferikoen arteko konexioak) eta VESA® (Video Electronics Standards Association; Bideo-elektronikaren Estandarren Elkartea) bus lokalak ... *101*
7.2.5 PCMCIA® (PC Memory Card International Association / PCentzako Memoria Txartelen Nazioarteko Elkartea) bus eramangarria ... *101*
7.2.6 AGP® busa (Grafikoen Portu Arindua) ... *101*
7.3 BIOS-A (SARRERA ETA IRTEERAKO OINARRIZKO SISTEMA) ... 102
7.4 PC-AREN PLAKA NAGUSIA ... 103
7.4.1 Plaka motak ... *103*
PC-AT plaka ... 103
PC-ATX plaka ... 104
7.4.2 µP-aren zokaloa ... *105*
7.4.3 Cache memoria edo aurrememoria ... *106*
7.4.4 PCetako busen egitura ... *106*
7.4.5 DRAM memoria-txipak ... *107*
7.5 KOMUNIKAZIO-PORTUAK ... 108
7.5.1 RS-232® serieko ataka ... *108*
7.5.2 Ataka paraleloa ... *109*
7.5.3 Komunikazio-ataken erabilera ... *109*

8. ARIKETAK ... 112

8.1 ARIKETAK 8051 µC-AREKIN ... 112
8.2 C PROGRAMAZIO-HIZKUNTZARI BURUZKO GALDERAK ... 177
8.3 DATU-EGITUREN ARIKETAK ... 179

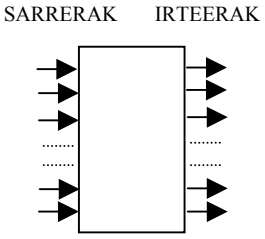
9. ERREFERENTZIA BIBLIOGRAFIKOAK ... 197

1. ZIRKUITU DIGITALAK

Mikroprozesadorea zer den ulertzeko modu erraz bat zera da, konbinaziozko eta sekuentziazko zirkuitu digitalekin erkatzea edo konparatzea.

1.1 Zirkuitu konbinazionala

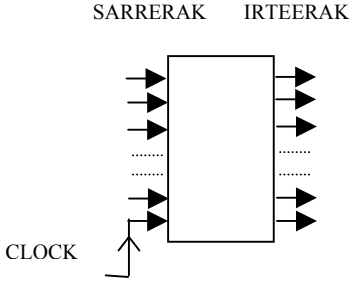
Konbinaziozko zirkuitua da ate logikoak eta elkarren artean konektagarriak dituena baina, betiere, sarrera-konbinazio berberarentzat beti irteera-konbinazio berbera ematen duena.



1-1 irudia. Konbinaziozko zirkuitua

1.2 Zirkuitu sekuentziala

Sekuentziazko zirkuitua da elkarren artean konektatutako ate logikoen multzoa baina betiere, funtzionamendu-une desberdinetan dagoen sarrera-konbinazioak irteera-konbinazio desberdinak eman ditzakeena.

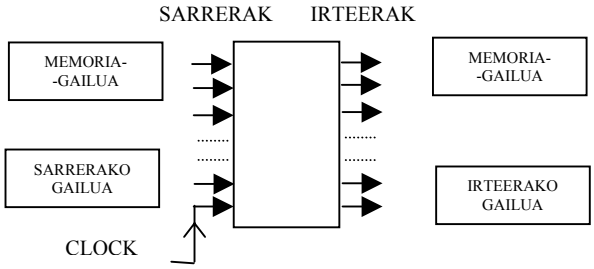


1-2 irudia. Sekuentziazko zirkuitua

Sekuentziazko zirkuituek gogoratzeko ahalmena —beraz, oroimena edo memoria— dute; konbinaziozkoek, ordea, ez.

1.2 Mikroprozesadorea (µP)

Adierazteko era hau funtzionamendua ulertzeko modu erraz eta sinplifikatua da: mikroprozesadoreak konbinazio bitarrak hartzen ditu bere sarreran, hain zuzen ere, egikaritzen edo gauzatzen ari den programaren aginduak baino ez direnak.



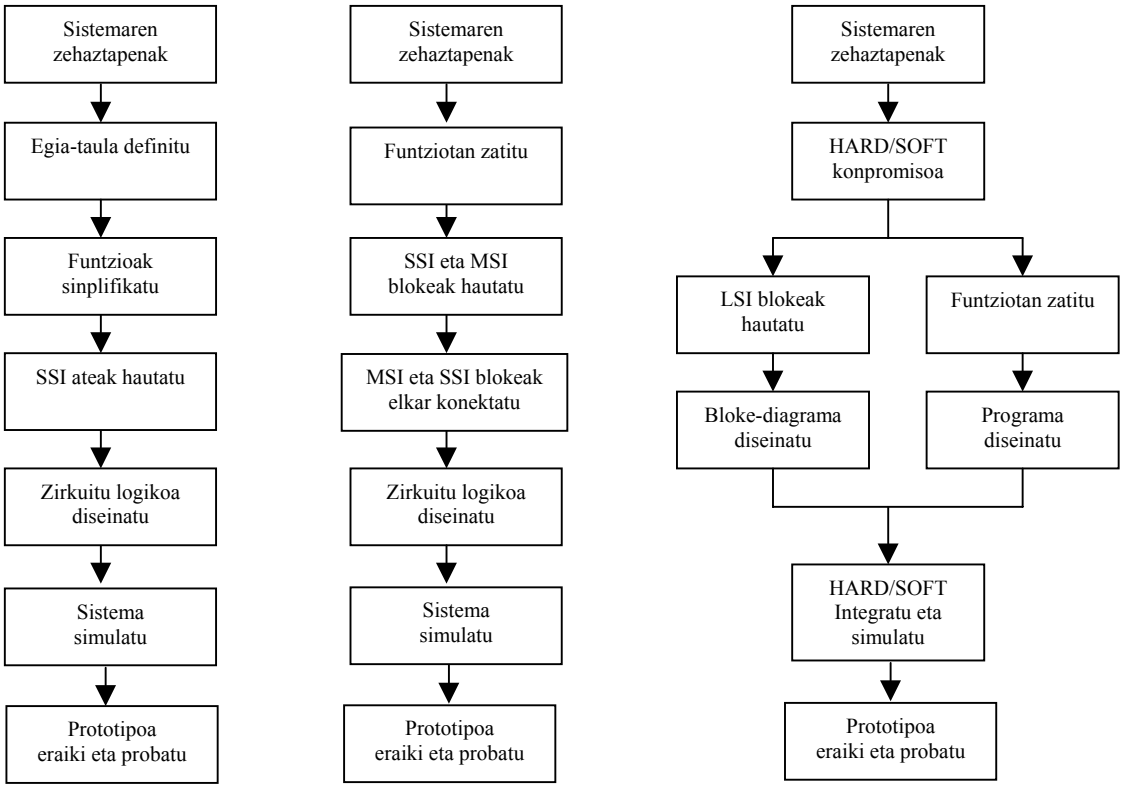
1-3 irudia. Mikroprozesadorea

Bere barnean, agindu horiek deskodetu egiten dira eta, gero, egikarituak izateko beharrezkoak diren seinaleak sortzen dira. Seinale horiek mikroprozesadoreak berarekin konektatuta dauden gailuei ematen dizkien irteerak dira.

Seinaleen sarrera eta irteera modu sinkronizatuan egiten da, gailu horrek duen erloju bidezko sarrera dela eta.

1.4 Kableatutako eta programatutako logika-sistemak

Sistema digital bat hiru integrazio-eskaletako edozein gailurekin inplementatua izan daiteke: SSI, MSI eta LSI eskaletakoekin (integrazio-eskala txiki, ertain edota handikoarekin, hurrenez hurren).



1-4 irudia. SSI, MSI eta LSI blokeekin egindako diseinu-prozesua (ezkerretik eskuinerantz)

LSI blokeekin zirkuituak eraikitzeko beharrezkoa den diseinu-prozesua, 1-4 irudian agertzen denez, SSI eta MSL blokeekin egindako diseinuaren aldean oso desberdina da.

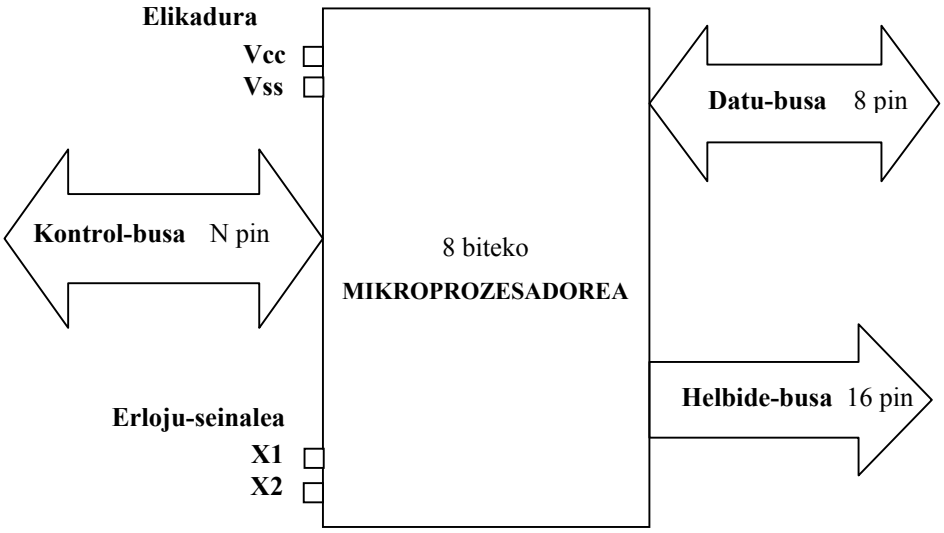
Kableatutako logika-sistema bat osagai diskretu eta digitalez osatutako zirkuitu elektroniko bat da, eta, aldi berean, hardwarea bakarrik duen sistema bat da; beraz, diseinu espezializatu eta bakarra da, eta horrek zurrun izatea dakarkio. Horregatik, kableatutako logika-sistema duen zirkuitu bat ezin da erabili prozesu baten kontrolaren automatizazioari irtenbidea emateko, baldin eta automatizazio hartarako diseinatua izan ez bada. Erabat zurruna denez, diseinatua izan den irtenbiderako bakarrik balio du.

Aldiz, *programatutako logika-sistema* batek, hardwarez nahiz softwarez osatutako zirkuitu elektronikoa denez, bare atal malguan aldaketak egiteko aukera ematen du, hau da, softwarean, eta horrekin, oinarrizko hardwarea irtenbide askotarako erabiltzea ahalbidetzen du (muga batzuen barruan), egikaritu behar duen programa aldatze hutsarekin.

2. MIKROPROZESADOREA

2.1 Mikroprozesadorearen kanpo-arkitektura

Mikroprozesadorea, funtsean, aurreko atalean ulertzera ematen den zirkuitu integratua baino askoz ere konplexuagoa da. Nola sarrerak hala irteerak ez dira agertzen era horretan desberdinduta; izan ere kanpo-arkitektura ere, orratz edo pinei dagokiena bereziki, hiru busetan banatzen baita eta horietatik zehar informazioak modu arautuan zirkulatzen du.



2-1 irudia. 8 biteko mikroprozesadorearen kanpo-arkitektura

Busa, definizioz, komunikazio-protokolo jakin bati erantzunez beren baitan datu bitarrak garraiatzeko aukera ematen duten hari eroaleen multzoa da.

Datu-busa mikroprozesadorera datuak sartzen eta irteten uzten duen busa da; beraz, noranzko biko busa da. Irteera moduan funtzionatzen duenean, µP-ak une horretan hautatutako gailuari bidaltzen dio datua, eta sarrera gisa funtzionatzen duenean, berriz, aukeratutako gailuak bidaltzen dio datua µP-ari. Datu-busa definitzen duen hari edo lerro kopuruak zehazten du datu-busaren tamaina edo busaren neurria eta baita gailuarena ere. Hala bada, 8 biteko datu-busa duen µP-a 8 biteko µP-a dela esaten da. Gauza bera gertatzen da gainerako gailu periferikoekin ere.

Helbide-busa, µP-ari gailu jakin baten kokapen desberdinetara iristeko aukera ematen dion hari multzoa da. Era horretan, gailu hori memoria bat baldin bada, bus horrek aukera emango dio µP-ari bere helbide-busean adierazi eta memoriaren helbide-busari konektatuta egongo den memoriaren kokapenera iristeko.

Helbide-busa noranzko bakarrekoa da; beraz, informazioak noranzko bakarrean bidaiatzen du: µP-arentzat irteerako busa da, eta gainerako gailuentzat, berriz, sarrerakoa.

Eta azkenik, *kontrol-busa***,** funtsean, irteeraz osatutako busa da, eta aukera ematen dio µP-ari une bakoitzean sisteman egin behar den eragiketa mota definitzeko.

Baldin eta µP-ak berarekin konektatuta dagoen memoriaren baten kokapenen baten daturen bat irakurri behar badu, bus horren memoria-irakurketa egiteko irakurtzeko baimenaren seinalea gaitu egingo da (benetan µP-ak gaituko du, berarentzat irteera bat baita) eta horrek helbide-busak adierazten dion helbidearen kokalekuan kokatuta dagoen datua, µP-ak bere datu-busaren bitartez irakur dezan aukera emango dio.

Badira µP-aren kontrol-busaren seinale gutxi batzuk, irteerak izan ordez sarrerak direnak. Haien artean, arruntenak kanpo-etendurak eta DMA/ZMArenak dira (DMA = Direct Memory Access / ZMA = Zuzeneko Memoria Atzipena).

Erlojua (clock). Hasiera-hasieran aipatua denez, CPU/PUZ (Central Processing Unit)/ (Prozesatzeko Unitate Zentrala) zirkuitu sinkrono konplexua da, eta, horregatik, erloju-seinale bat behar du bere seinale guztiak sinkronizatzeko. zirkuitu erloju-sortzaile txipean sartuta dago CPU/PUZaren alboan, eta kanpoko kuartzozko kristal bati edota RC akoplamendu (erresistentzia-kondentsadore) bati konektatu behar zaio oszilazio-maiztasuna finkatzeko. Konexio hori egiteko aukera ematen duten pinak X1 eta X2 dira.

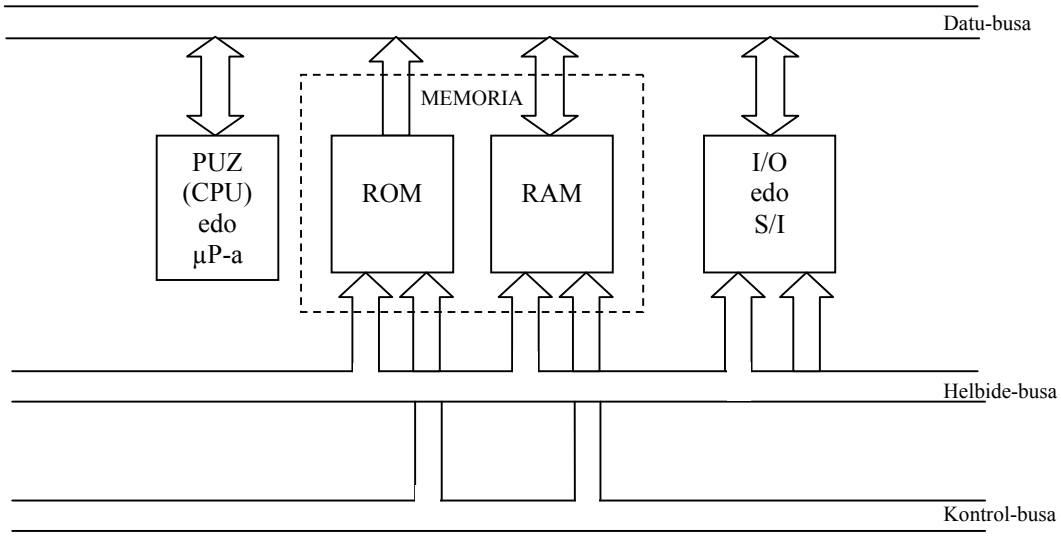
Urteetan zehar, µP eta µC asko joan dira merkatuan agertzen, eta gero eta oszilazio-maiztasun handiagoekin joan ere, integrazioaren eremuan egindako teknologia-aurrerapenei esker: 4 MHz, 8 MHz, 12 MHz, 16 MHz, 24 MHz, 30 MHz, etab.

Mikroprozesadore zahar askoren *elikadura* askotarikoa eta arautu gabea zen: +5 V, +12 V eta -5 V, edo +5 V eta -7 V, etab. Logikoa zenez, denbora asko igaro baino lehen, merkatura irten ziren µP-eredu berrietako askok +5 V-eko elikadura estandarra onartu zuten. Hala, diseinuaren sinplifikazioa lortu zuten, eta gainera, baita TTL edo CMOS familiako zirkuitu gehigarriekiko bateragarritasuna ere.

2.2 Oinarrizko sistema edo gutxieneko sistema.

Mikroprozesadore bat bere inguruan beste gailu batzuk konektatuak dituela dugunean, hala nola memoriak, I/O (S/I; Sarrera/Irteera) portuak, A/D, D/A bihurgailuak, etab., *mikroprozesadorean oinarritutako sistema*tzat edota *mikrokonputagailu*tzat hartzen da. Elementu horien arteko konexio-bideak aurreko puntuan aipatu ditugun hiru busak edo bus taldeak dira: datu-busa, helbide-busa eta kontrol-busa.

Mikroprozesadorearen ahalbideak osagai moduan aprobetxatzeko aukera ematen duten elkarren artean konektatutako elementu edo bloke funtzionalen gutxieneko multzoari *gutxieneko sistema* edo *oinarrizko sistema* deritzo. 2-2 irudiko bloke-diagramak erakusten du hiru bloke funtzionalek eratutako gutxieneko sistema petoa: CPU/PUZ Prozesu Unitate Zentrala edo µP-a, memoria-gailua eta sarrera/irteera (S/I edo I/O) unitatea dituena alegia. Bloke horien konexio-sistema lehen azaldutako *busen oinarrizko egiturena* da, eta, batez ere, malgutasuna eta hedapen-erraztasuna ditu ezaugarri.



2-2 irudia. Oinarrizko sistema baten bloke-egitura

Bloke funtzional bakoitza txip batez edo hainbat txipez osatuta egon daiteke.

Sistemaren funtzionamendua oso erraza da maila orokorrean aztertzen bada, baina konplexua xehetasunen arlora jaitsiz gero. Programatutako logika-sistema denez, zirkuituaren funtzionamendua gobernatzen duen programa bat izan behar du, eta programa hori sekuentziaz egikaritzen diren agindu multzo batez osatuta dago. Agindu horiek programaren memorian kodetuta eta biltegian metatuta egoten dira. Prozesu-unitate zentralak (PUZ/CPU) agindu horiek banaka-banaka interpretatu eta bete egiten ditu. Eragiketa, funtsean, gutxieneko sistema eratzen duten elementuen artean datuak transferitzea edo berekin aritmetika- eta logika-eragiketak egitea da.

2.3 Memoria

Mikroprozesadorean oinarritutako sistema batean, datuak eta emaitza partzialak nahiz egikaritu behar diren programak biltegian metatzeko sistema bat izateko erabiltzen da memoria.

Datuak, emaitza partzialak eta programak egikaritu behar dituen eragiketen eragigaiak metatzen dituen memoria zatiari, oro har, *datu-memoria* deritzo, eta RAM (ausazko atzipeneko memoria) izenez ere ezagutzen da (RAM memoria edo memoria bizia, alegia). Logikoki, CPU/PUZak (prozesu-unitate zentralak) datu-memoriako kokaleku guztietara sartzeko ahalbidea izan behar du, bai datu jakin bat irakurtzeko eta baita emaitza bat gordetzeko nahiz idazteko. Esandakoaren arabera, datu-memoriako txip edo RAM batek helbide-bus bat, datu-bus bat eta idazteko eta irakurtzeko baimen-seinaleak izango ditu, txipa bera gaitzeko seinaleaz gain.

Bestalde, programa osatzen duen agindu sorta metatzen duen memoria zatiari, *programa-memoria* deritzo. Agindu bakoitza CPU/PUZak jasotzen du memoria horretatik aldez aurretik ezarritako sekuentziaren arabera. Agindua deskodetu egiten da CPU/PUZaren barnean eta dagokion jarduna burutzen edo gauzatzen da.

Datu-memorian gertatzen ez den bezala, non programa egikaritu bitartean haren kokalekuetan idatz daitekeen, programa-memoriak ez du onartzen idazketa-eragiketarik; izan ere, hala eginez gero, programaren zatiren bat ezabatu edo indargabetu egin liteke. Beraz, egikaritu aurretik kargatzen da programa programa-memorian memoria-grabatzaileen bitartez. Programa-memoria, halaber, *kode-memoria* edo *irakurtzeko bakarrik den memoria* izenez ere ezagutzen da (ROM, EPROM, EEPROM, FLASH).

2.4 S/I (I/O) unitatea (sarrera-irteerak).

Unitate honek komunikazio-bidea eskaintzen du mikroprozesadore-sistemaren eta unitate periferiko edo S/I (I/O) unitate deituetan bildutako kanpoko munduaren artean. Sarrerek CPU/PUZ unitateari kanpoko unitateetatik (teklatu, kanpoko memoria, magnitude fisiko, eta abarretatik) datuak lortzeko aukera ematen diote. Irteerek, berriz, CPU/PUZ unitateak bere aritmetika- eta logika-eragiketen emaitzak (display/bistaratzea, etab.) kanpoko elementu periferikoei komunikatzeko balio dute. Irteerak, halaber, prozesu-kontroleko seinaleak bidaltzeko ere erabil daitezke, tresneriarekin gertatzen den moduan.

2.5 PUZ/CPUa (Prozesatzeko Unitate Zentrala)

Prozesatzeko unitate zentral izena ematen zaie mikroprozesadorearen bi bloke funtzionalei: kontrol-unitateari eta prozesu-unitateari (unitate aritmetiko-logikoa). Prozesatzeko-unitate zentrala txip bakarrean sartuta dagoenean (gaur egun, kasu guztietan), mikroprozesadore (µP) izeneko osagaia eratzen du.

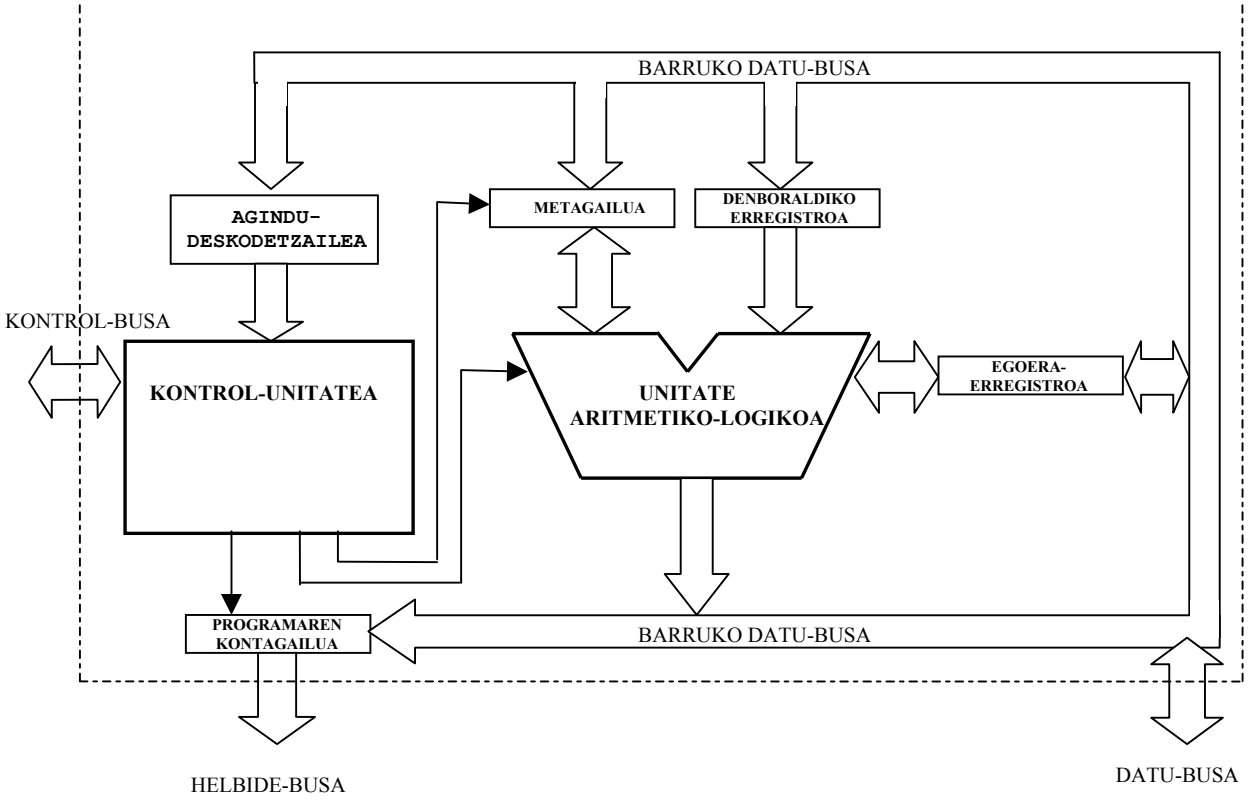
PUZ/CPUa mikrokonputagailuaren bihotza da, eta sistemak egikaritzen dituen gainerako funtzio guztiak kontrolatzen ditu. µP-ak, programaren memorian (programa-memorian edo kode-memorian) dauden aginduak irakurtzen ditu eta, gero, deskodetu eta egikaritu egiten ditu.

PUZak (CPUak), halaber, hurrengo ikasgaian azalduko diren *etendurak* deituriko hainbat kanpo-seinale ere ezagutu eta aditzen ditu.

Prozesu-unitatea, normalean, unitate aritmetiko-logiko (UAL) batez, erregistroz eta kontagailuz osatuta egoten da. UALak aritmetikako eta logikako eragiketak egiteko aukera ematen du, sistemaren beste osagai batzuek, barne-erregistroek edo metagailuek (metagailuek), hornitu behar dituzten beste bi eragigairen bitartez.

UALak halaber, egoera-erregistroa deritzon eta egindako eragiketen egoera adierazten duen erregistro horretan ere eragiten du, eta baldintzen informazioa ematen du, hala nola: zeroa, zeinu negatiboa, bururakoa, paritatea, eta abar.

Erregistroak eta kontagailuak oinarrizkoak dira mikroprozesadore guztietan. Erregistroak PUZaren (CPU) barruan datuak denboraldirako metatzeko eta PUZaren kontrolpean funtzio bereiziak egiteko erabiltzen dira. Erregistro berezien kasuan, azpimarratzekoak dira metagailuak; mikroprozesadore guzti-guztiek gutxienez metagailu bat izan ohi dute.



2-3 irudia. UPZ (CPU) edo µP-aren barruko bloke-egitura

Kontrol-unitateak, berriz, mikrokonputagailuaren unitate funtzional guztiak kontrolatzen (kontrol-busaren bitartez) eta koordinatzen ditu. Horretarako, erloju-seinale patroi bat behar du, sistemaren zirkuitu guztiak sinkronizatzeko behar diren erloju-pultsuak emateko. Kontrol-unitateak programaren memorian metatutako programatik jasotzen ditu aginduak eta, deskodetu ondoren, azpisistema funtzional egokiei ematen dizkie komando horiek egikaritzeko aginduak.

Badira beste elementu batzuk ere µP-aren zati direnak, baina ez dira aurkitzen aipatutako bi unitateetako edozeinetan. Hain zuzen ere, SP/PE (Stack Pointer/Pilako erakuslea) eta PC/PK (Program Counter/Programaren kontagailua) dira. Lehenengoa errutinetara egindako jauzietan eta etenduretan erabiltzen da; beraz, aurrerago aztertuko da. Bigarrena, berriz, egikaritu behar den hurrengo agindua duen kode-memoriaren kokapen-helbidea helbide-busean jartzeko ardura duen erregistroa da. Modu automatikoan gehitzen du kontrol-unitateak eta 00H-n kargatzen da µP-an reset egiten denean, hortxe aurkitzen baita egikaritu behar den programaren lehenengo instrukzioa.

2.6 Mikroprozesadorearen softwarea

Mikroprozesadore baten (eta µC baten) agindu multzoa agindu-deskodetzaileak (kontrol-unitateak) interpreta ditzakeen adierazpide bitarren multzo osoak osatzen du. Bistakoa da, noski, zenbat eta handiagoa eta ugariagoa den agindu multzoa hainbat eta

handiagoa eta zabalagoa izan beharko duela gailuaren eraginkortasunak eta, horren ondorioz, sartuta dagoen sistemarenak.

µP-ak, zirkuitu digital orok bezalaxe, batekoekin eta zeroekin jokatzen du soil-soilik. Alabaina, zenbaki-sistema bitarra oso nahasgarria da programazio-hizkuntza moduan erabiltzeko; izan ere, programa bat egiteko garaian, lerro bakoitzean idatzi beharko genukeen agindu bakoitzaren kode bitarraz oroitu beharko genuke.

Programen irakurketa eta idazketa hizkuntza mintzatuari hurbiltzearren, µP-en aginduak, egin behar duen eragiketa definitzen duten hitzen laburdurak edo siglak erabiliz egitea erabaki zen. Gailu horien fabrikatzaile diren konpainia horien jatorria zela eta, hitz horiek ingelesez adierazita daude. Hala bada, adibidez, MOV laburdura MOVEtik dator eta higitu, mugitu edo lekualdatu adierazten du, eta barruko edo kanpoko erregistroen artean datuak transferitzeko edo lekualdatzeko erabiltzen da.

Adierazpide horiei *oroitzeko adierazpen* (*adierazpen mnemoniko*) deritze eta oroigarri-esanahia dute. Beraz, edozein µP-ren edo µC-ren aginduak adierazpen mnemonikoen multzo mugatua dira, eta fabrikatzailearen familia bereko beste gailu batzuekin bateragarriak; kasu batzuetan, baita fabrikatzaile beraren beste familia batzuekin ere, baina, zoritxarrez, beste fabrikatzaile batzuen beste edozein gailurekin ez dira bateragarriak izaten.

Aginduen adierazpen mnemonikoak edo oroitzeko adierazpenak erabiltzen dituen hizkuntzari *mihiztadura-hizkuntza* deritzo. Baina, logikak eskatzen duenez, behar-beharrezkoa da itzultzaile bat izatea mihiztadura-hizkuntzan idatzitako programa hizkuntza bitar edo makina-kode deitura bihurtzeko, hori baita makinak uler dezakeen hizkuntza bakarra. Beraz, itzultzaile hori *mihiztatzaile* (assembler) deituriko programa baizik ez da. Jakina, lehen adierazitako arrazoiarengatik, µP edo µC jakin batentzako mihiztatzaileak ez du balio beste gailu desberdin batentzat (beste fabrikatzaile batek egindakoarentzat edota fabrikatzaile berak egindako beste familia batekoarentzat).

Badira mihiztatzaileen antzeko beste programa batzuk ere, eta horiek egiten dutena zera da, *Pascal, C* eta gisako goi-mailako hizkuntzetan idatzitako kodea makina-kodera itzuli. Programa mota horrek *konpiladore* (compiler) izena hartzen du.

Praktikan, behar-beharrezkoa da bi kasuetan ere hirugarren programa bat erabiltzea. *Estekatzaile (linker)* deritzo, eta honako hauek biltzen ditu: iturri-kode osoa eratzen duten fitxategien banakako konpilatze/mihiztatze lanen kodeak eta programa horretan erabilitako liburutegi-funtzioen kodeak, makina-kode osoa emateko.

Mihiztadura-hizkuntza behe-mailako hizkuntza da; izan ere, haren bitartez lor daiteke makinaren (µP edo µC-aren) edozein elementutara iristea eta hura manipulatzea. Hori abantaila izan liteke une jakin batean, baina desabantaila ere bada, eta, makinak eragiketa konplexuak egitea nahi denean, errutina gutxi-asko konplexu samarrak sortu behar dira zerikusia duen edozein elementu ere ahaztu gabe; hala egin ezean, µP-ak itxaron ez zen beste eragiketa bat egingo du. Adibidea: 8051 µC-arekin, zenbaki oso baten erro karratua kalkulatzea, bere agindu multzotik abiatuta.

Aldiz, C hizkuntza bezalako goi-mailako hizkuntza batek, gailuaren elementu guztietara iristeko aukera eman arren, kasu askotan konpondu egiten du hizkuntza mihiztatzaileek sortzen duten eragozpena, ia gauza guztiak egiteko moduko liburutegi-funtzioak baititu. Zehatz-mehatz neurrira datorren liburutegi-funtziorik ez dagoen kasuetan, beti manipula daiteke iturri-kodea neurrira jar dadin, edota, besterik gabe, beste berri bat sor daiteke μC-ak bere barnean dituen elementuez ia deusik ere kezkatu gabe; izan ere, horretaz konpiladorea arduratuko da.

Alde horretatik, konpiladore on batek ahal duen koderik txikiena sortuko du, baina sortutakoa eraginkorra izango da, hain zuzen ere, programatzaile aditu eta trebe batek mihiztadura-hizkuntzan egindako programa bat mihiztatuta lortuko lukeen kode bera edo antzekoa.

Agindu multzora itzulita, argi eta garbi jarri behar da μP-aren edo μC-aren agindu bat hiru elementuk zehazten dutela:

- Agindu-kodeak, mihiztadura-hizkuntzan adierazpen mnemoniko jakin bat elkartzen zaionak.
- Programa-memorian okupatzen dituen byte kopuruak (zortzikote kopuruak).
- μP-ak agindua egikaritzeko behar duen denborak, makina-ziklotan neurtuta.

Normalean, μP bakoitzak bere berariazko agindu multzoa izaten du eta agindu-kode jakin bat, agindu zehatza da μP-arentzat; kode bera, berriz, agindu desberdina izango da beste μP batentzat.

Alabaina, badaude beste fabrikatzaile batzuen beste mikro batzuen imitazioak diren μP eta μC-ak, eta, kasu horietan, jatorrizkoek duten agindu multzo bera erabiltzen dute. Orduan, mikro horiek *software-arloan jatorrizkoarekin bateragarriak (softwarez jatorrizkoarekin bateragarriak) direla* esaten da.

3. iNTEL ETXEAREN 8051 MIKROKONTROLATZAILEA

Mikroprozesadorea eta µP–etan oinarritutako sistemetan gehien erabiltzen diren elementu periferiko batzuk txip bakarrean integratuak dituen zirkuitua da mikrokontrolatzailea. Mikrokontrolatzaile-ereduak, µP edo UPZ (CPU) izateaz gainera, tenporizadore 1 edo 2, etenduren kontrolatzaile bat, serieko portu bat, A/D motako bihurgailu bat eta D/A motako beste bat, datuentzako RAM memoria (memoria bizia) eta programentzako ROM memoria (memoria hila) ditu.

DISEN-552 mikrokontrolatzailearentzako entrenatzailea erabiliko denez, Philips-en 80C552 µC-an oinarrituta, eta hori Intel-en 8051 µC-aren bertsio handitua denez, lehendabizi jatorrizko mikrokontrolatzailea aztertuko dugu.

8051 eredua Intel-en 51 familiako µC-a bat da. Merkatuan hamarkada bi badaramatzan arren, gaur egun ere oso erabilia da oraindik µC-etan oinarritutako diseinuetan, gailu honen bertsio asko baitaude Philips, Atmel, Texas Instruments eta gisako beste etxe batzuek fabrikatuta.

3.1 Pinak (orratzak)

Irudiak µC-eredu horren ondorengo pin-konfigurazioa azaltzen du DIP (Dual In Line Package) kapsulatuan sartuta; bestalde, PLCC (Plastic Leaded Chip Carrier) paketean ere eskuragarri dago.

Ondoren, 8051 µC-aren pin-identifikazio laburra egiten da:

Vcc: +5 volt-eko elikadura positiboa

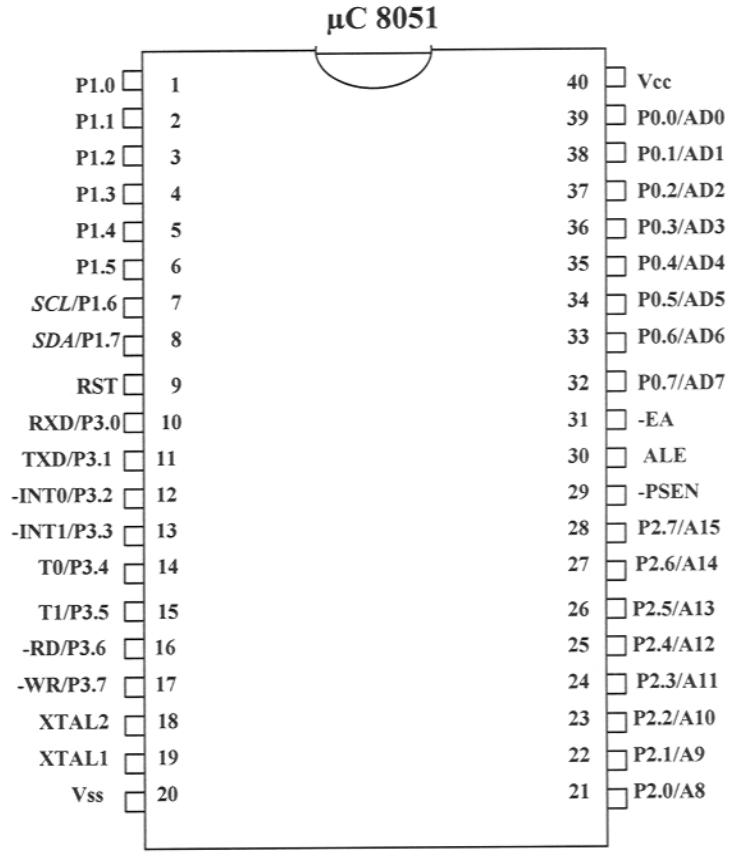
Vss: Lur-konexioa, 0 volt

XTALl eta **XTAL2:** µC-aren barruko erloju-sortzailearen sarrera eta irteera dira. Kanpotik kuartzozko kristal bat konekta daiteke edota zeramikazko durundagilea. Oszilazio-maiztasunak era zuzenean zehazten ditu programaren egikaritze-abiadura, serieko portuaren komunikazio-abiadura eta tenporizadoreen zenbaketa-abiadura.

S/I (I/0) portuak: 8051 µC-ak 4 S/I (I/O) portu ditu, bakoitza 8 pinekoa (bitekoa) eta erabiltzaileak pin bakoitza bere aldetik modu independentean programa dezake.

P0 noranzko biko portua da, eta PI, P2 eta P3 portuak ia noranzko bikoak dira; beraz, denak dira programagarriak, nola sarrera hala irteera digital moduan. P0 portuak beste hirurekiko duen beste aldeetako bat zera da, duen karga-ahalmena edo fan out deitua: P0-aren pin bakoitzak TTL-LS erako 8 sarrerari eraso diezaieke eta beste portuetakoek, aldiz, TTL-LS erako 4ri bakarrik.

Pinen bat sarrera moduan erabili nahi baldin bada, aldez aurretik “1” logiko bat idatzi behar da , pina honda ez dadin.



3-1 irudia. 8051 µC-aren pin-konfigurazioa

P1 portuak salbu, beste hirurek badituzte izendatuak, S/I (I/O) funtzioaz gain, beste funtzio gehigarri (gainerako funtzio) batzuk ere:

- **P0 portua**: Denboran zehar bere 8 harietan barrena helbide-busaren beheko aldea eta datu-busa multiplexatzen ditu, datuen nahiz kodearen kanpo-memoriaren atzipenetan.
- **P2 portua**: Helbide-busaren goiko aldea igortzen du, datuen nahiz kodearen kanpo-memoriaren atzipenetan.
- **P3 portua**: Paketean integratutako elementu periferikoen sarrerak eta irteerak agertzen ditu eta, gainera, kanpoko datu-memoriako datuak irakurtzeko eta idazteko baimena, 3-1 taulak azaltzen duen moduan.
- **ALE**: Adress Latch Enable, helbide-busaren beheko aldea LATCH OCTAL 373 kanpoko gailuan harrapatzeko µC-ak igortzen duen pultsu bat da, horrela, helbide-busaren beheko aldea eta datu-busa P0 portutik demultiplexatzeko (banatzeko).
- **-EA**: External Acces, goi-maila logikora konektatzean, barruko ROMaren programa bakarrik egikaritzen da, programaren kontagailuak OFFF (4 kilobyte) gainditzen ez badu behintzat. –EA=0 (behe-maila logikora konektatzen baldin bada), kodearen kanpo-memoriako programa egikaritzen da beti, programaren kontagailuak duen helbidearekin zerikusirik izan gabe.

- **-PSEN: K**anpo-memoriako kodea (kontrol-programa) irakurtzeko baimena ematen duen seinalea.
- **RESET:** µC-aren barruko gailuak hasieratzeko seinalea. Oszilatzailea funtzionatzen ari den bitartean eta RESET sarreran gutxienez bi ziklo-makinatan goi-maila logikoa aplikatzen denean gertatzen da hasieratzea.

Pina	*Funtzio gehigarria*
P3.0	RxD, UART serieko portuaren sarrera
P3.1	TxD, UART serieko portuaren irteera
P3.2	-INT0, 0 kanpo-etenduraren sarrera
P3.3	-INT1, 1 kanpo-etenduraren sarrera
P3.4	T0, 0 tenporizadorearen kanpoko sarrera
P3.5	T1, 1 tenporizadorearen kanpoko sarrera
P3.6	-WR, kanpoko datu-memorian idazteko baimena
P3.7	-RD, kanpoko datu-memorian irakurtzeko baimena

3-1 taula. P3 portuko pinen funtzio gehigarriak

3.2 Memoriaren antolakuntza

3.2.1 Barruko memoria (barne-memoria)

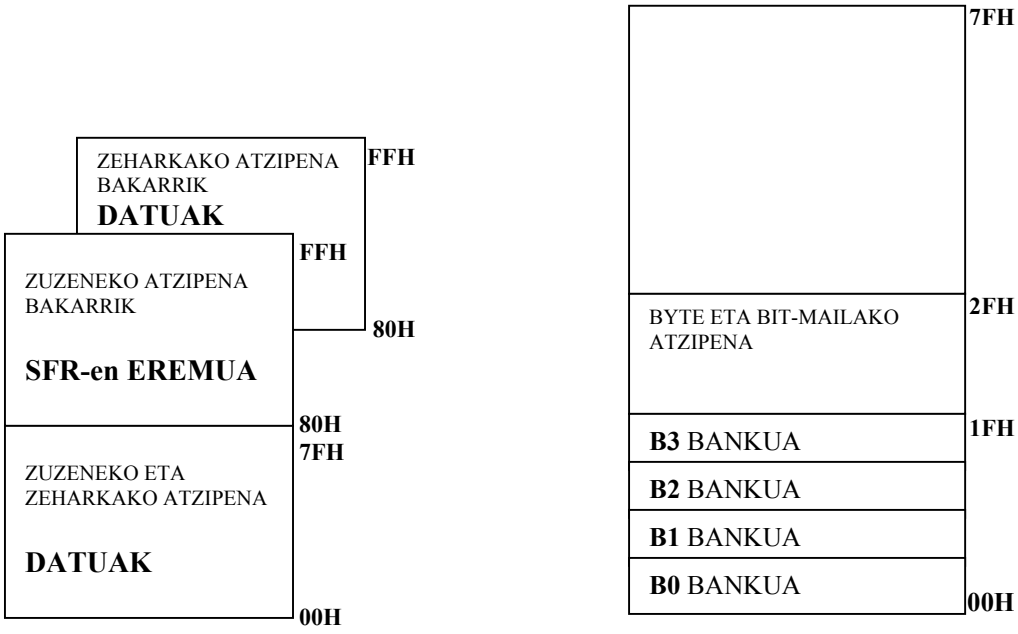
µC-ak bi memoria mota ditu bere paketean integratuta, barruko datu-memoria eta barruko kode-memoria.

Barruko kode-memoria (barruko ROMa) memoria-grabatzaile baten bitartez kargatzen da, eta kargatze-lana, bai fabrikatzaileak edota baita programatzaileak ere egin dezake. µC-ak egikarituko duen programa dauka bere barnean, –EA pina goi-maila logikoan konektatuta egon behar duelarik.

8051 µC-aren barruko kode-memoriaren tamaina, 4 kilobyte da, hots, 0000H helbidetik 0FFFH-rakoa. 80C552 µC-a ez dago memoria mota horrekin hornitua; aldiz, 83C552 µC-ak 8 kilobyteko ROM memoria du eta 87C552 µC-ak ere 8 kilobytekoa du, baina EPROM motakoa.

Barruko datu-memoriari dagokionez (barruko RAMa, alegia), µC-ak egikaritzen dituen programak prozesatutako datuak denboraldi batean gordetzeko erabiltzen da. 8051 µC-ak 128 byte (zortzikote) ditu barruko RAM memorian, datuetarako orokorki, zuzeneko eta zeharkako atzipenarekin (00H helbidetik 7FH-raino) eta beste 128 byte (zortzikote) SFRentzat (Funtzio Berezidunen Erregistroentzat) zuzeneko atzipena bakarrik dutela. Erregistro horiek okupatzen ez dituzten kokalekuak ezin litezke erabili (80H helbidetik FFraino). Ikus 3.2 irudian, ezkerreko aldea.

Datuentzat orokorki dagoen 128 zortzikoteko (byteko) eremuaren barruan, 16 memoria-posizio daude (20H helbidetik 2FHraino), eta horietara, bit-mailako nahiz byte-mailako atzipenaz hel daiteke. 3-2 irudiaren eskuinaldekoa.



3-2 irudia. Barruko datu-memoria

80C51 µC-ak: SFR bakarrik
8OC652 µC-ak: SFR eta ***SFR***
80C552 µC-ak: SFR, ***SFR*** eta **SFR**

FFH	**T3**	**PWMP**	**PWM1**	**PWM0**				**IP1***	F8H
F7H								B*	F0H
EFH	**RTE**	**STE**	**#TMH2**	**#TML2**	**CTCON**	**TM2CON**		**IEN1***	E8H
E7H								ACC*	E0H
DFH					***S1ADR***	***S1DAT***	***#S1STA***	***S1CON****	D8H
D7H								PSW*	D0H
CFH	**#CTH3**	**#CTH2**	**#CTH1**	**#CTH0**	**CMH2**	**CMH1**	**CMH0**	**TM2IR***	C8H
C7H		**#ADCH**	**ADCON**	**#P5**				**P4***	C0H
BFH								IP0*	B8H
B7H								P3*	B0H
AFH	**#CTL3**	**#CTL2**	**#CTL1**	**#CTL0**	**CML2**	**CML1**	**CML0**	IEN0*	A8H
A7H								P2*	A0H
9FH							S0BUF	S0CON*	98H
97H								P1*	90H
8FH			TH1	TH0	TL1	TL0	TMOD	TCON*	88H
87H	PCON				DPH	DPL	SP	P0*	80H

3-3 irudia. 80C51, 80C652 eta 80C552 µC-etako Funtzio Bereizien Erregistroak

Barruko datu-memoriaren behereneko 32 bytetan (00H helbidetik 1FHrainokoetan), 32 erregistro daude kokatuta, R0, Rl, R2, R3, R4, R5, R6 eta R7 izenak dituztenak, lau multzotan edo bankutan taldekatuta, eta izen berberez deitzen dira lau multzo edo banku horietan bildutako zortzi erregistroak ere (3-2 irudian, eskuinaldekoa). Lau banku edo

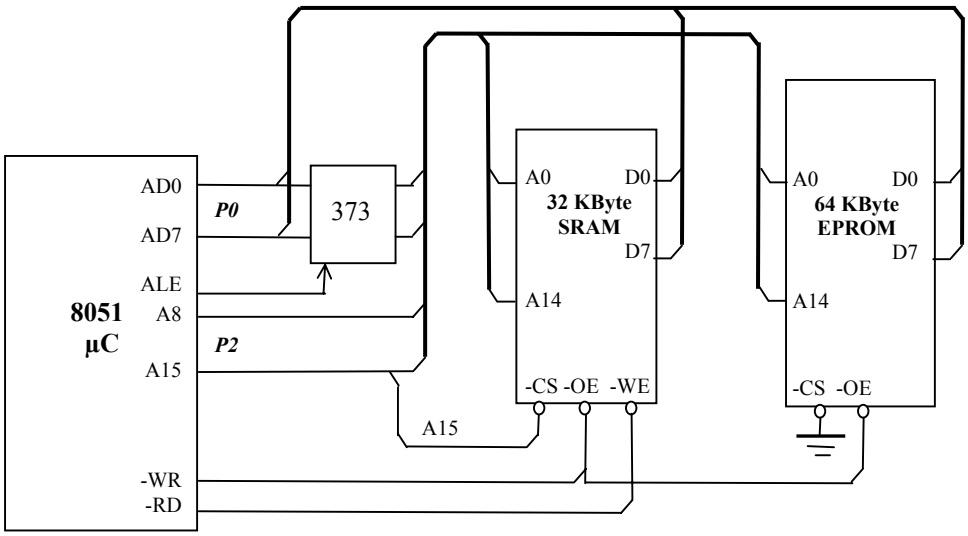
multzoetako baten hautaketa PSW Funtzio Berezidun Erregistroaren (UPZ/CPUaren egoera-erregistroaren) RS1 eta RS0 bitekin egiten da.

80C552 µC-ak 256 byteko (zortzikoteko) barruko datu-memoria du, hain zuzen ere, 8051 ereduak datu orokorrentzako jada bazeuzkanei batutako beste 128 byte. Memoria-gehikuntza hori SFR eremuaren atzeko aldean egiten da, eta, bertara iristeko, zeharkako atzipena erabiltzen da. Zuzeneko edo zeharkako atzipen modua izateak memoria horren goiko aldean eremu batera ala bestera heltzea eragiten du (80H helbidetik FFHrainokoak).

3.2.2 Kanpo-memoria

8051 µC-ak kanpoko bi memoria mota helbidera ditzake: kanpoko datu-memoria eta kanpoko kode-memoria. Kasuetako bakoitzean, 65536 byte (zortzikote) arte helbidera ditzake (64 kilobyte).

Kanpoko datu-memoria eta/edo kode-memoria erabiltzea ongi zurituta edo justifikatuta dago, diseinu jakin batentzako izen bereko barruko memoria behar adinakoa ez denean. Gogoratu beharra dago, 8051 µC-ari kanpoko memoria-txipak konektatu ahal izateko behar-beharrezkoa dela *octal latch 373* zirkuitu integratua sartzea 8051 ereduaren P0 portuan, helbide-busaren beheko aldea eta datu-busa demultiplexatzeko.



3-4 irudia. 8051 µC bat kanpo-memoriarekin: 32 kilobyte datuentzako eta 64 kilobyte programentzako

3.3 Agindu multzoa

Gaur egun mihiztadura hizkuntzan idatziriko programarik ia erabiltzen ez den arren, egungo µC-entzako konpiladoreak oso eraginkorrak baitira, izan liteke interesgarria, jakinguratsuentzat behintzat, zein agindutan bihurtuko diren µC honentzat goi-mailako hizkuntzatan idatziriko edozein programaren sententziak.

Datuak mugitzeko aginduak

MOV dir,A	F5	2	1					
MOV dir,Rn	88-8F	2	2	MOV A,Rn	E8-EF	1	1	XCH A,Rn	C8-CF	1	1
MOV dir,dir	85	2	2	MOV A,dir	E5	2	1	XCH A,dir	C5	2	1
MOV dir,@Ri	86,87	2	2	MOV A,@Ri	E6,E7	1	1	XCH A,@Ri	C6,C7	1	1
MOV dir,#dat	75	3	2	MOV A,#dat	74	2	1	XCHD A,@Ri	D6,D7	1	1
MOV Rn,A	F8-FF	1	1	MOV @Ri,A	F6-F7	1	1	MOV DPTR,#dat16	90	3	2
MOV Rn,dir	A8-AF	2	2	MOV @Ri,dir	A6-A7	2	2	PUSH dir	C0	2	2
MOV Rn,#dat	78-7F	2	1	MOV @Ri,#dat	76-77	2	1	POP dir	D0	2	2
MOVC A,@A+DPTR	93	1	2	MOVX A,@Ri	E2,E3	1	2	MOVX @Ri,A	F2,F3	1	2
MOVC A,@A+PC	83	1	2	MOVX A,@DPTR	E0	1	2	MOVX @DPTR,A	F0	1	2

Aritmetika-arloko aginduak

ADD A,Rn	28-2F	1	1	ADDC A,Rn	38-3F	1	1	SUBB A,Rn	98-9F	1	1
ADD A,dir	25	2	1	ADDC A,dir	35	2	1	SUBB A,dir	95	2	1
ADD A,@Ri	26,27	1	1	ADDC A,@Ri	36,37	1	1	SUBB A,@Ri	96,97	1	1
ADD A,#dat	24	2	1	ADDC A,#dat	34	2	1	SUBB A,#dat	94	2	1
INC A	04	1	1	DEC A	14	1	1	DA A	D4	1	1
INC Rn	08-0F	1	1	DEC Rn	18-1F	1	1				
INC dir	05	2	1	DEC dir	15	2	1	MUL AB	A4	1	4
INC @Ri	06,07	1	1	DEC @Ri	16,17	1	1	DIV AB	84	1	4
INC DPTR	A3	1	2								

Logika-arloko aginduak

ORL A,Rn	48-4F	1	1	ANL A,Rn	58-5F	1	1	XRL A,Rn	68-6F	1	1
ORL A,dir	45	2	1	ANL A,dir	55	2	1	XRL A,dir	65	2	1
ORL A,@Ri	46,47	1	1	ANL A,@Ri	56,57	1	1	XRL A,@Ri	66,67	1	1
ORL A,#dat	44	2	1	ANL A,#dat	54	2	1	XRL A,#dat	64	2	1
ORL dir,#dat	43	3	2	ANL dir,#dat	53	3	2	XRL dir,#dat	63	3	2
ORL dir,A	42	2	1	ANL dir,A	52	2	1	XRL dir,A	62	2	1
SWAP A	C4	1	1	RR A	03	1	1				
CLR A	E4	1	1	RRC A	13	1	1				
CPL A	F4	1	1	RL A	23	1	1				
				RLC A	33	1	1				

Instrukzio-kodea
Byte kopurua
Makina-zikloak

Bitak manipulatzeko aginduak (bit-arloko aginduak)

CLR C	C3	1	1	SETB C	D3	1	1	CPL C	B3	1	1
CLR bit	C2	2	1	SETB bit	D2	2	1	CPL bit	B2	2	1
ORL C,bit	72	2	2	ANL C,bit	82	2	2	MOV bit,C	92	2	2
ORL C,/bit	A0	2	2	ANL C,/bit	B0	2	2	MOV C,bit	A2	2	1

Programaren fluxu-kontrolerako aginduak

LJMP dir16	02	3	2	LCALL dir16	12	3	2	NOP	00	1	1
SJMP rel	80	2	2	RET	22	1	2	AJMP dir11	X1	2	2
JMP @A+DPTR	73	1	2	RETI	32	1	2	ACALL dir11	X1	2	2
JC rel	40	2	2	JZ rel	60	2	2	DJNZ Rn,rel	D8-DF	2	2
JNC rel	50	2	2	JNZ rel	70	2	2	DJNZ dir,rel	D5	3	2
CJNE A,dir,rel	B5	3	2	JB bit,rel	20	3	2				
CJNE A,#dat,rel	B4	3	2	JNB bit,rel	30	3	2				
CJNE Rn,#dat,rel	B8-BF	3	2	JBC bit,rel	10	3	2				
CJNE @Ri,#dat,rel	B6,B7	3	2								

Instrukzio-kodea
Byte kopurua
Makina-zikloak

Oharrak			
Rn	R0-R7 lan-erregistroa	**#dat**	**8 biteko eragigai konstantea**
@Ri	RAM barne-memoriako kokalekua zeharkako memoria-atzipenaz R0 edo R1 erabiliz helbideratua	**#dat16**	16 biteko eragigai konstantea
		rel	Hurrengo instrukzioaren 1. bytetik jauzia egiteko 8 byteko balioa (-128-tik +127-ra)
dir	RAM barne-memoria eta SFR-en 128 posizio	**dir16**	Kode-memoriako helbidea (64K)
bit	RAM barne-memoria edo SFR-en zuzeneko bit-mailako atzipeneko bita	**dir11**	Kode-memoriako helbidea (2K)

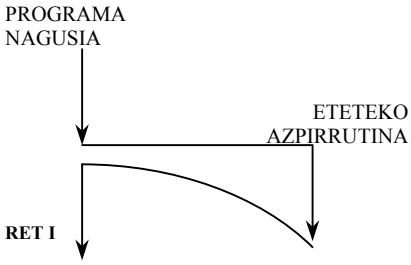
3-5 irudia. 8051 µC-aren agindu multzoa

4. 8051 µC-AREN BARRUKO HARDWAREA

4.1 Etendurak

Definizioa: *Etendura dei-seinale bat da, eta denboraldi batez eten egiten du programa nagusiaren egikaritzea, tartean ataza berri bat egiteko.*

"Ataza berri bat egiteaz" hitz egiten denean, ekintza jakin bat egiteko agindu sorta bat egitea ulertu behar da, baina denboran zehar mugatua dena eta amaieran programa nagusia egikaritzen jarraitzera behartuko duena, zehatz-mehatz eten zen unean zegoen puntutik eta eten aurreko egoeratik abiatuta.



4-1 irudia. Programa nagusiaren egikaritze-etendura

Sententzia multzo horrek *etenduraren zerbitzu-azpirrutina* izenez ezagutzen dena eratzen du. µC-aren etendura-iturri bakoitzak, sententzia multzo hori hartzeko, kode-memoriaren esparru bat gordeta dauka, hain zuzen ere 8 zortzikote (byte) jarraian 51 familiarentzat. Gordetako 8 horietatik lehenengo helbideari *etendura-bektore* deritzo. Baldin eta etendura-azpirrutinak beretzat gordetako esparrua gainezkatzen badu, orduan, sententzia bidez, leku-arazorik ez dagoen beste gune libre batera jauzi egiten du.

Etendura bat gertatzean, PUZ/CPUak pila-memorian gordetzen du programaren kontagailu-balioa (etendura amaitu ondoren programa nagusian egikaritu behar den hurrengo sententzia barne duen kode-memoriaren helbidea daukana) eta, segidan, etendura-bektorearekin kargatzen du programaren kontagailua.

Hala bada, etenduraren zerbitzu-azpirrutina egikaritzen hasiko da eta egingo duen lehen gauza zera izango da, *pila-memorian* programa nagusiak erabilitako erregistroen balioak gordetzea (pilaratzea edo pilan sartzea), azpirrutinakoek teilaka ez ditzaten, eta, gero, etendurari berari dagozkion ekintzak egingo ditu horretarako behar dituen sententziak egikarituz. Jarraian, pilatik, programa nagusian erabiltzen ari den erregistroen balioak berreskuratuko ditu (pilatik aterako ditu), baina alderantzizko ordenan, eta azkenik, etendura-azpirrutinari amaiera emango dio horretarako den etendura-azpirrutinatik itzultzeko RET I agindua egikarituz, baina, aurretik, etendura hori eskatu zuen bandera (flag-a) ezgaituko du, µC-aren etendura-sistemak ezgaitzen ez badu behintzat.

Etenduratik itzultzean, programa nagusiak eten aurretik utzi zen baldintza berberetan egikaritzen jarraituko du eta denbora-atzerapen bat jasango du bakarrik, hain zuzen ere honako denbora hauen batura dena: joateko emandako jauziarena, azpirrutina-sententzia guztiak egikaritzeko behar izandakoa eta itzulerako jauziarena.

Pila-memoria LIFO (Last In First Out, Lehena Sartu Azkena Atera) motakoa da, pilaratzeko (pilan sartzeko) sententzia PUSH (pilaratzea) da eta pilatik ateratzeko POP (pilatik ateratzea). PC/PK programaren kontagailua da 51 familiako PUZ/CPUak era automatikoan pilaratzen eta pilatik ateratzen duen erregistro bakarra, hau da, hardware bidez egiten duena. Programa nagusiak eta azpirrutinak partekatutako gainerako erregistroek (eskuz) software bidez pilaratuak edo pilatik atereak izan behar dute, goian aipatutako bi sententziak erabiliz, eta xede hori programatzaileari dagokio.

Litekeena da merkatuan 8 biteko µC-ak topatzea, hain zuzen ere, Programaren Kontagailua (PK) izateaz gainera, pilaren tratamendu automatikoa sarritan erabiltzen diren beste erregistro batzuei ere ematen dietenak, adibidez, metagailuari.

µC-arentzako programa C hizkuntza den bezalako goi-mailako hizkuntzan egina baldin bada, pilaratzeko eta pilatik ateratzeko atazak izugarri sinplifikatzen dira; izan ere, programaren aldagaiek zentzu horretan tratamendu automatikoa hartzen dute.

51 familiaren kasuan, erabiltzen den hizkuntzaren arabera, honako hau izango dugu:

- *mihiztadura-hizkuntza* erabiltzen bada etenduraren arreta-azpirrutinarentzat, azpirrutina horrek honako sententzia honekin hasi behar du: `ORG 00XYH`, non sasiagindu artezkaria laguntzen duen helbide hamaseitarra etendura-iturri horri esleitutako bektorea baita (ikus 4-1 taula). Azpirrutinaren amaiera, beti, `RET I` sententziarekin izango da.
- *C programazio-hizkuntza* erabiltzen bada, kasu honetan, funtzio bat denez, honela hasi beharko du: `void FuntzioarenIzena(void) interrupt n`, eta jarraian ireki giltza `{`, hor funtzioaren izena programatzaileari egoki iruditzen zaion edozein izan liteke, nahiz eta "interrupt" gako-izena errespetatu beharko duen eta, atzetik, etendura-iturri horri dagokion zenbaki oso bat sartu beharko du (ikus 4-1 taula). Amaiera, noski, ixteko giltzarekin: `}`.

80C552 µC-ak 15 etendura-iturri ditu (9 barrukoak eta 6 kanpokoak) RESETa kontuan hartu gabe. Etendura-eskaera iturri jakin baten etendura-bandera (flag-a) gaitzen denean gertatzen da, eta hori, noski, baldintza jakin batzuk betetzen direnean gertatzen da.

Adibidez, T0 tenporizadorearen zenbaketa-erregistroa gainezkatzen denean, haren TF0 flag-a gaitu egiten da eta, horrekin, programa nagusiaren denboraldirako etena eskatzen da tenporizadore horren arreta-azpirrutina egikaritzeko. Hori amaituta, programa nagusia egikaritzen jarraitzen da utzi zen puntu beretik eta baldintza berberetan berrabiatuta. Hala ere, horretarako, aurretik, etendura-bandera ezgaitu edo desgaitu behar da, ezen hala egin ezean, berriro ere azpirrutina egikarituko bailitzateke programa nagusiaren ordez.

Etendura-flag bakoitza iturri horrentzat zehaztutako baldintza bereiziak betetzen direnean gaitzen da. T0 eta T1 tenporizadore-flag-en, eta INT0 eta INT1 kanpoko etenduren ezgaitzea automatikoki egiten du µC-aren etendura-sistemak, hots, hardware bidez egiten da. Aldiz, gainerako etendura-banderak, software bidez (programatzaileak eskuz) deskonektatu beharra dago azpirrutinan, programa nagusira itzuli aurretik.

Azpirrutina edo etendura-funtzioa:

1. Pilaratzea (pilan sartzea)
2. Etendurak egin beharreko ekintza.
3. Flag-aren ezgaitzea
4. Pilatik ateratzea
5. Amaiera (RET I edo })

Iturria	***Bektorea***	*interrupt* ***n***	***Eskaera-baldintza***	***Flag-a***		
RESET	0000H	-	Reset pultsadorea sakatu. T3 zakur zaintzailea (watch-dog) tenporizadorea gainezkatu	***on***	***off***	***SFR***
Kanpoko 0	0003H	0	Aplikatu behe-mailakoa (IT0=0 baldin bada) edo jaitsiera-saihetsean (IT0=1 baldin bada) –INT0 (P3.2) pinean.	IE0	hard	TCON*
0 tenporizadorea	000BH	1	Zenbaketa-erregistroa gainezkatu.	TF0	hard	TCON*
Kanpoko 1	0013H	2	Aplikatu behe-mailakoa (IT1=0 baldin bada) edo jaitsiera-saihetsean (IT1=1 baldin bada) –INT1 (P3.3) pinean.	IE1	hard	TCON*
1 tenporizadorea	001BH	3	Zenbaketa-erregistroa gainezkatu.	TF1	hard	TCON*
UART serieko ataka	0023H	4	Datu bat jasotzean Datu bat bidaltzean	RI TI	soft soft	S0CON*
I^2C serieko ataka	002BH	5	I^2C busa, guztira dituen 26etatik 25 egoera posibleetako bakarren batean sartzen denean.	SI	soft	S1CON*
2 tenporizadorea 0 atzipena	0033H	6	CT0I pinean, saihets bakoitzeko, T2-ren harrapatzea CT0 erregistroan eskatzen denean.	CTI0	soft	TM2IR*
2 tenporizadorea 1 harrapaketa	003BH	7	CT1I pinean, saihets bakoitzeko, T2-ren harrapatzea CT1 erregistroan eskatzen denean.	CTI1	soft	TM2IR*
2 tenporizadorea 2 harrapaketa	0043H	8	CT2I pinean, saihets bakoitzeko, T2-ren harrapatzea CT2 erregistroan eskatzen denean.	CTI2	soft	TM2IR*
2 tenporizadorea 3 harrapaketa	004BH	9	CT3I pinean, saihets bakoitzeko, T2-ren harrapatzea CT3 erregistroan eskatzen denean.	CTI3	soft	TM2IR*
AD bihurgailua	0053H	10	AD bihurketa amaitzen denean (hurrenez hurreneko hurbilpenez, bihurtzeko denbora=50 CM).	ADCI	soft	ADCON
2 tenporizadorea 0 konparazioa	005BH	11	T2 eta CM0 konparazio-erregistroaren arteko parekotasuna (bat etortzea) gertatzean.	CMI0	soft	TM2IR*
2 tenporizadorea 1 konparazioa	0063H	12	T2 eta CM1 konparazio-erregistroaren arteko parekotasuna (bat etortzea) gertatzean.	CMI1	soft	TM2IR*
2 tenporizadorea 2 konparazioa	006BH	13	T2 eta CM1 konparazio-erregistroaren arteko parekotasuna (bat etortzea) gertatzean.	CMI2	soft	TM2IR*
2 tenporizadorea Gainezkatzea	0073H	14	8 biteko kontagailua gainezkatu. 16 biteko kontagailua gainezkatu.	T2B0 T2OV	soft soft	TM2CON TM2IR*

4-1 taula. Philips etxearen 80C552 µC-aren etendura-taula

Etendura-bandera (flag) guztiak software bidez (eskuz) gaituak eta desgaituak izan daitezke, AD bihurgailuaren bandera salbu, hori ezabatu (desgaitu) baino ezin baita egin.

4.1.1 Etenduren gaiketa

Iturri jakin baten etendura-eskaeraren baldintza bete arren, horrek ez du bermatzen etendura gauzatuko denik. Etendura bat gerta dadin, etendura horrek gaitua egon behar du, eskaera-baldintza betetzeaz gainera.

Alde horretatik, etendura-iturri bakoitza bakarka gaitua edo ezgaitua izan daiteke, IEN0 edo IEN1 *Etenduren Gaiketen Erregistroetakoren* batean bita gaituz edo ezgaituz (1ean edo 0an jarriz, hurrenez hurren). IEN0 erregistroko EA bitak osotasunezko era batean gaitzen edo ezgaitzen ditu banaka gaitutako etendura guztiak.

EA	EAD	ES1	ES0	ET1	EX1	ET0	EX0	**IEN0***
ET2	ECM2	ECM1	ECM0	ECT3	ECT2	ECT1	ECT0	**IEN1* (T2)**

4-2 irudia. 80C552 µC-aren etenduren gaiketen erregistroak

4.1.2 Lehentasun-mailak

Etendura-iturrietako bakoitzak 2 lehentasun-maila ditu. Maila horiek, IP0 edo IP1 bi *etenduren lehentasunen erregistroetakoren* batean etendura-iturriari esleitutako bita 1ean edo 0an jarrita ezartzen dira. 0ak lehentasun-maila txikia ezartzen du, eta 1ak lehentasun-maila handia.

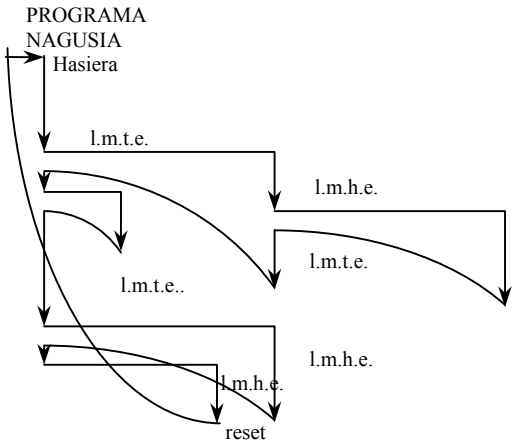
-----	PAD	PS1	PS0	PT1	PX1	PT0	PX0	**IP0***
PT2	PCM2	PCM1	PCM0	PCT3	PCT2	PCT1	PCT0	**IP1* (T2)**

4-3 irudia. 80C552 µC –aren etenduren lehentasunen erregistroak

Praktikan, litekeena da aldibereko bi etendura edo gehiago edota aldiberekoak ez direnak eskatzea. Kasu bakoitza aparte aztertuko dugu.

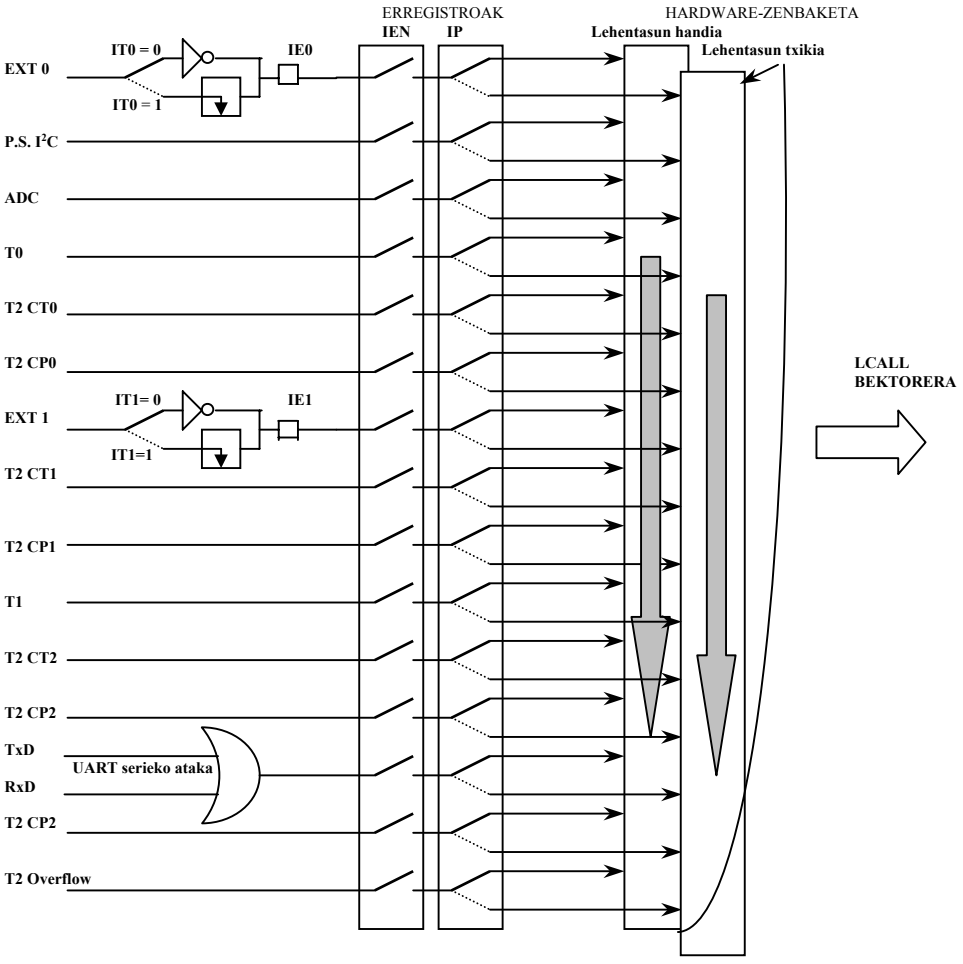
4.1.3 Aldiberekoak ez diren bi etendura edo gehiagoren eskaera

1. Lehentasun-maila txikia duen etendura bat prozesatzen denean, etendura hori lehentasun-maila handiko beste batek etena izan daiteke.
2. Lehentasun-maila txikia duen etendura bat prozesatzen denean, lehentasun-maila txikia duen beste etendura batek ezin dezake eten. Lehenengoa amaitu ondoren, bigarrena prozesatuko da dagokion flag-a gaituta baldin badago oraindik.
3. Lehentasun-maila handia duen etendura bat prozesatzen denean, ezin liteke lehentasun-maila handia duen etendura bat beste batek eten. Lehenengoa amaitu ondoren, bigarrena prozesatuko da, baldin eta berari dagokion flag-a gaituta baldin badago oraindik.



4-4 irudia. Lehentasunak direla-eta etendako hainbat etenduraren kasuak

4. Edozein etendura, edozein lehentasun-maila izan arren RESET etendurak eten dezake, arrazoia kanpokoa (reset-en pultsadorea) nahiz barrukoa bada ere (zakur zaintzailea/watch-dog).



4-5 irudia. Etendura-sistemaren irudikapena

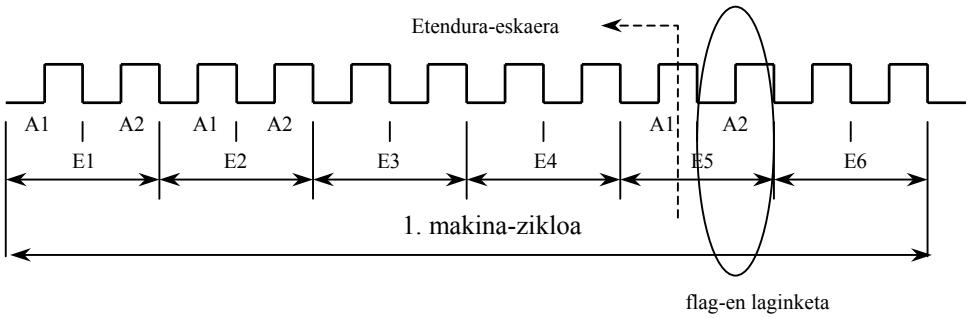
4.1.4 Aldibereko bi etendura edo gehiagoren eskaera.

1. Lehentasun-maila desberdina baldin badute, lehentasun-mailarik handiena duenari eskainiko zaio arreta lehenik, eta lehenengo etendura amaitzen denean prozesatuko da bigarrena, baldin eta haren bandera (flag-a) artean gaituta baldin badago.
2. Lehentasun-maila berbera baldin badute, etendura-sisteman lehentasun-maila beraren barruan lehentasun-mailarik handiena duenari eskainiko zaio arreta lehendabizi (4-5. irudia).

4.1.5 Etendura-prozesua

8051 µC-aren makina-zikloa 12 erloju-pultsuz osatuta dago. Bi erloju-ziklo osok egoera bat osatzen dute, eta guztira 6 egoera daude (E1etik E6ra) makina-ziklo bakoitzeko. Gainera, egoera bakoitza bi atalez (A1 eta A2) osatuta dago.

Etendura-banderei (flag-ei), makina-ziklo bakoitzaren 5. egoeraren (E5) 2. atalean (A2) laginketa egiten die etendura-sistemak, eta beren prozesatzearekin jarduten da blokeo-egoerarik ez baldin bada (4-6. irudia).



4-6 irudia. Etendura-prozesua

Blokeo-egoerarik ez baldin bada:

- Lehenengo makina-zikloan (1MZ), etendura-sistemak gaitutako flag-ak atzematen ditu.
- Bigarren makina-zikloan (2MZ), etenduren zenbaketa egin eta gaituta dauden ala ez aztertu ondoren (eta haien lehentasun-maila zein den), hala baldin badaude, aldi berean 1MZ makina-zikloan sententzia bat egikari dezake edota egikaritzen ari zena amaitu.
- 3MZ eta 4MZ makina-zikloak gauzatu bitartean, etendura-sistemak jarraian egikarituko den etendura-bektorera LCALL bat sortzen du (etendura-azpirrutinara jauzi egiten du; izan ere, PC-ak bektorea kargatu egiten baitu).

1MZ	2MZ	3MZ	4MZ	5MZ

Beraz, µC-ak, etendura eskatzen denetik bere egikaritzapena hasten deneraino, blokeo-egoerarik izan gabe duen etendura baten erantzun-denborarik txikiena 3 makina-ziklo baino handixeagoa da.

Bestalde, etendura-sistemak *blokeo-egoera* hauek aurki ditzake:

- 3. makina-zikloan (3MZ) sententzia bat egikaritzen ari da, eta hori amaitu arte ezin da sortu etendura-bektorerainoko LCALL-a (jauzia) hardware bidez.
- Lehentasun-maila bera edo handiagoa duen beste etendura bati arreta eskaini zaio eta hura egiten ari da. Orduan, gutxienez ere programa nagusiko sententzia bat gehiago bederen egikaritu egin beharko da etendura bektoretu baino lehen (PK/PCa dagokion bektorearen balioarekin kargatzen da). Guztia ere, noski, aurreko etendura amaitu ondoren egiten da.
- RET I sententzia edota IEN edo IPrako atzipen bateko sententzia bat egikaritzen ari da, eta orduan, gutxienez ere programa nagusiko sententzia bat gehiago bederen egikaritu egin beharko da etendura bektoretu baino lehen.

1MZ	2MZ	3MZ	4MZ	5MZ	6MZ	7MZ

Eskema orokor moduan, honako hau dugu:

- Hasieran, etendura eskatzeko baldintza ematen da.
- Jarraian, dagokion flag-a gaitzen da.
- Etendura-sistemak blokeo-egoeraren batekin topo egiten ez badu, etendura prozesatu egiten da (etendura gaitua dela suposatuz).
- Etendura-sistemak blokeo-egoeraren batekin topo egiten badu,
  - sistemak itxaron egiten du blokeo-egoera desagertu arte,
  - ondoren, flag-ak gaituta jarraitzen badu, µC-ak arreta eskainiko dio etendurari eta gauzatu egingo da, baina gaituta jarraitzen ez badu, ez dio arretarik eskainiko.

4.2 Tenporizadoreak eta kontagailuak

80C552 µC-ak 4 tenporizadore-kontagailu ditu: T0, T1, T2 eta T3. Lehenengo biak, T0 eta T1, Intel-en 8051 µC-ak dituen 16 biteko bi tenporizadoreak dira; beraz, baita Philips-en 80C652 µC-ak ere. T2 ere 16 biteko tenporizadorea da, baina egitura konplexuagoarekin; izan ere, lau harrapatze-erregistrora eta hiru konparazio-erregistrora konektatuta baitago. Azkenik, T3 tenporizadorea dugu, 8 biteko watch-dog edo zakur zaintzaile deitua. Atal honetan, T0, T1 eta T3 tenporizadoreak aztertuko dira.

4.2.1 T0 tenporizadorea eta T1 tenporizadorea

16 biteko bi tenporizadore-kontagailu dira. Biak berdinak dira, eta bakoitzak kontagailu-erregistro edo zenbaketa-erregistro bikoitza du. Pentsa litekeen legez, tenporizadore bakoitzaren erregistro bikoitza bi zenbaketa-erregistroz osatuta dago eta horietako bakoitza 8 bitekoa da: T0-ren kasuan, TH0 eta TL0 dira, eta T1-en kasuan, berriz, THl eta TLl (4-7. irudia).

Tenporizadore bakoitza tenporizadore moduan edo kontagailu moduan funtzionatzeko konfigura daiteke. 4-8 irudiak funtzionamenduaren eskema irudikatzen du.

4-7 irudia. T0 eta T1 tenporizadoreen zenbaketa-erregistroak

Tenporizadore gisako funtzionamendua

Zenbaketa-erregistroa, igarotako makina-ziklo (MZ) bakoitzak gehitua gertatzen da, hots, 12 erloju-pultsuko. Horregatik, zenbaketa-arrazoia hau da:

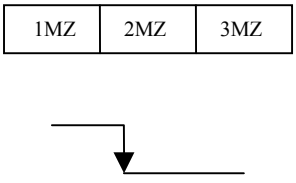
Zenbaketa-arrazoia = fosc / 12

Zenbaketa-erregistroak gainezka egiten duenean, dagokion tenporizadorearen etendura-bandera (flag-a) gaitzen da: TF0, T0 tenporizadorearen kasuan eta TF1, T1 tenporizadorearenean. Horrela, etendura-eskaera egiten da.

Kontagailu gisako funtzionamendua

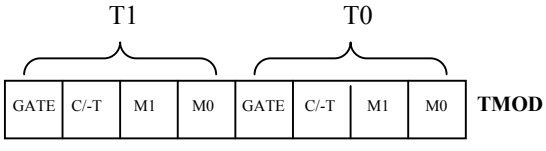
Zenbaketa-erregistroa, T0 (P3.4) sarrerako pinean atzemandako jaitsiera-saihets bakoitzak gehitua gertatzen da T0 tenporizadorearen kasuan eta T1 (P3.5) pinean T1-en kasuan. Jaitsiera-saihetsa atzemateko gutxienez bi makina-ziklo (24 erloju-pultsu) behar direnez, (lehenengoan, goiko mailako egoera logikoa atzemango da, eta hurrengoan, behe-mailakoa) zenbaketa-arrazoi handiena honako hau izango da:

Zenbaketa-arrazoi handiena = fosc / 24



Bi SFR daude T0-ri eta T1-i elkarturik: TMOD eta TCON.

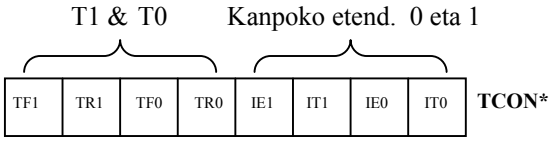
TMOD edo *Modu Erregistroa* bi tenporizadoreen moduak zehazten dituen erregistroa da: goiko nibble-a T1-i dagokio eta behekoa T0-ri.



M1	M0	modua
0	**0**	0, C/T 13 bitekoa
0	**1**	1, C/T 16 bitekoa
1	**0**	2, C/T 8 bitekoa autobirkargaz
1	**1**	3, zenbait kontagailu-tenporizadore

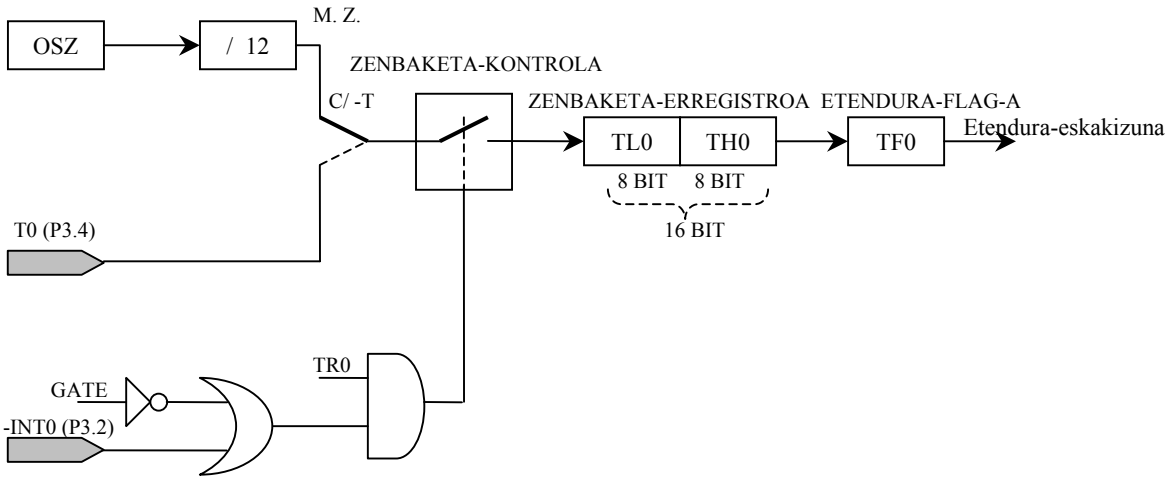
Gainera, kasu bakoitzean, C/-T bitaren egoera logikoak bere funtzionamendua zehazten du, tenporizadore (0) edota kontagailu gisa (1).

TCON edo *Kontrol-erregistroa* goiko nibble-an bi tenporizadore horien funtzionamendua (zenbaketa eta banderak) kontrolatzen dituen erregistroa da, eta beheko nibble-an, berriz, 0 eta 1 kanpoko etendurak. Erregistro hori bit-mailan helbidera daiteke (*).



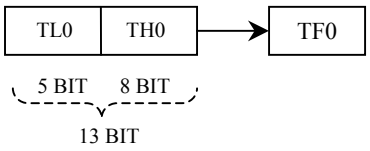
Funtzionamendu moduak

0 modua. Modu honetan, zenbaketa-erregistroak 13 bit ditu, 5 TLO-n eta 8 THO-n; horrekin, 2^{13} = 8.192 makina-ziklo edo jaitsiera-saihets zenbatzean gainezka egiten du. Gertaera horrek T0 tenporizadorearen etendura-bandera (flag-a) gaitzen du eta haren etendura eskatzen du.

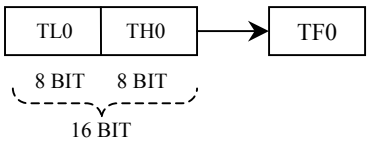


4-8 irudia. T0 Tenporizadore-kontagailuaren irudi-eskema

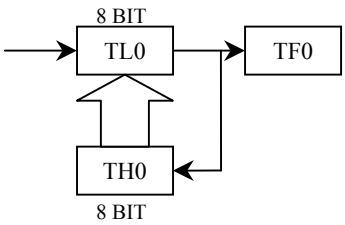
TL0-k dituen 5 bitak pisurik gutxienekoak dira. Pisurik handieneko hirurak zehaztu gabe daude; beraz, programatzaileak ez ditu kontuan hartuko.



1 modua. Modu honetan, zenbaketa-erregistroa 16 bitez osatuta dago, 8 TL0-n eta beste 8ak TH0-n; horrekin, 2^{16} = 65.536 makina-ziklo edo jaitsiera-saihets zenbatzean, gainezka egin eta etendura-flag-a gaitzen du.

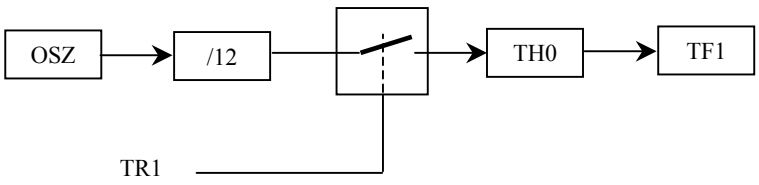


2 modua. Alabaina, modu honetan, zenbaketa-erregistroa 8 bitekoa da: TL0, eta gainezka egiten duen bakoitzean, TH0-ko balioa zenbaketa-erregistrora berriro kargatzen da. Birkargatze-prozesu hori automatikoki gertatzen da gainezka-aldi bakoitzean eta, hori dela bide, berez *autokargatzeko modu* deritzo. Kargaren balioa programatzaileak hautatzen du bere kalkuluen arabera, eta 00 hamaseitarra litzatekeen kasuan, gainezka egingo luke 2^8 = 256 makina-ziklo edo jaitsiera-saihets zenbatzean.



Beraz, laburbilduz, TL0 zenbaketa-erregistroak gainezka egiten duenean, bi gauza gertatzen dira: alde batetik, etendura-flag-a gaitzen da, eta bestetik, zenbaketa-erregistroa automatikoki kargatzen da berriro, TH0 autokargatzeko erregistroaren balioarekin, eta hori inolako makina-ziklorik edo jaitsiera-saihetsik igaro gabe egiten da.

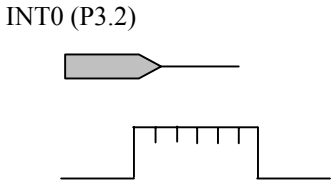
3 modua. Modu honek 8 biteko bi kontagailu independente ditu, eta TL0 eta TH0 zenbaketa-erregistroak ditu. Lehenengoak T0 tenporizadorearen egitura gordetzen du, eta bigarrenak (ikus irudia) makina-zikloak zenbatzeko aukera bakarrik ematen du, kontrol-lana zuzen-zuzenean TR1 seinalearen bidez eginez eta gainezka egiten duenean T1-ren etendura-flag-a aktibatuz.



Tenporizadoreak erabiltzeko moduak

Tenporizadore bakoitza lau modutan erabil daiteke:

1. *Tenporizadore* moduan, makina-zikloak zenbatuz: C/-T = 0 eta zenbaketa-kontrola software bidez eginez, TR0 = 1 eta GATE = 0 direla.
2. *Jaitsiera-saihetsen kontagailu* gisa, kanpoko pultsuak zenbatuz: C/- T = 1 eta software bidezko zenbaketaren kontrola TR0 = 1 eta GATE = 0 direla.
3. –INT0 pinean aplikatutako *pultsuen iraupen-kontagailu* moduan (0 kanpoko etendurarena, 1 kanpoko etenduraren P3.2 edo –INT1-ena, T1 tenporizadorearentzat). Pultsuaren iraupena makina-ziklotan zenbatzen da: C/-T=0 eta hardware bidez egindako zenbaketa-kontrola, TR0=1, GATE=1 (eta –INT0=1).



4. Baita –INT0 pinean aplikatutako *pultsuen iraupen-kontagailu* moduan ere, baina, kasu honetan, pultsuaren iraupena kanpotik datozen denbora-unitatetan zenbatzen da, hots, T0/P3.4 pinarenetan. Horregatik, modua C/-T=1 kontagailua da eta hardware bidezko zenbaketa-kontrola TR0=1, GATE=1 (eta INT0=1) dira.

4.2 2 Zakur zaintzailea (watch-dog)

Zakur zaintzailea edo watch-dog deiturikoa µC batzuek duten tenporizadore bat da. Helburu hau du: µC horiek programa egikaritzean ez dezatela programatzaileak aldez aurretik ezarritako fluxu-kontrola gal.

Alde horretatik, zakur zaintzailea (watch-dog) bere zenbaketa-erregistroak gainezka egiten duenean µC-a berrasierarazten duen (barne-berrasiera eginarazten dion) tenporizadore bat da. Horregatik, programaren puntu desberdinetan, gainezka egite hori gerta ez dadin, kargatzen aritu beharra dago. Hori egiteko, programatzaileak diseinatutako programa xehetasun osoz aztertu beharko du, hedadura osoan zehar babesteko: zati nagusia, etendura-errutinak, etab.

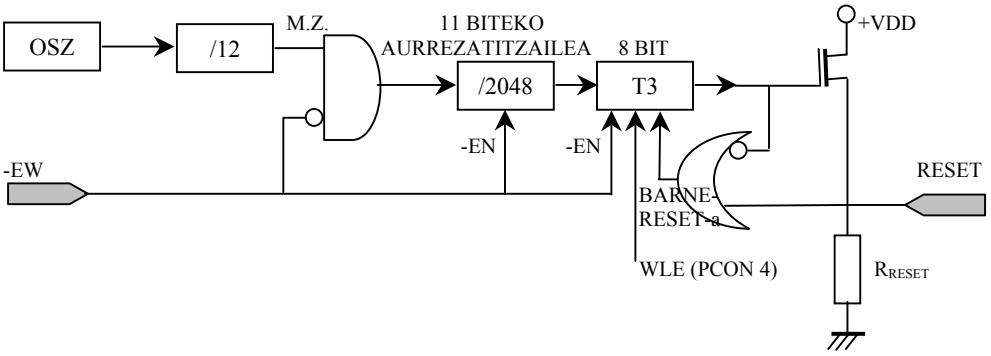
Zakur zaintzailearen (watch-dog) erabilera µC-ak egikaritutako programak dituen fluxuaren balizko kontrol-galerei zor zaie. Kontrol-galera horiek, aipatutako µC-a instalatuta egongo den lekuan izan daitezkeen irrati-maiztasunen interferentzia indartsuek edo zarata elektrikoek eragiten dituzte.

80C552 µC-aren T3 tenporizadorea 8 biteko zakur zaintzaile erako (watch-dog) tenporizadorea da, eta haren zenbaketa-arrazoia honako formula honek ematen digu:

Zenbaketa-arrazoia = fosz/ (12*2048*T3)

non T3-ren balioa 1 eta 256 (FFH eta 00H) bitartekoa den; izan ere, ertzetako balio horiekin lortuko lirateke denborarik txikiena eta handiena, hurrenez hurren.

Aztertutako T3 tenporizadorea irudikatzen duen eskema sinplifikatua honako hau da:



4-9 irudia. T3 tenporizadorearen eskema sinplifikatua

T3 tenporizadorea erabiltzeko garaian, betiere kontuan hartu behar da tenporizadore hori karga akastunak eragozteko moduan diseinatua izan dela. Beraz, arrazoi hori dela bide, T3 zenbaketa-erregistroaren karga bi ataletan edo alditan egin behar da:

1. Zakur zaintzailearen (watch-dog) karga egokitu WLE (watch-dog load enable; ZZKG zakur zaintzailearen karga gaitu) bita goi-mailako egoera logikoan jarrita. Bit hori SFR PCONaren 4. bita da, eta ez du onartzen bit-mailako atzipenik.
2. T3 zenbaketa-erregistroa kalkulatutako balioarekin kargatu. Horrela, WLE (ZZKG) behe-mailako egoeran jartzen da automatikoki T3-ren hurrengo karga ez egitea bermatzeko, baldin eta programaren sententzia bidez bit hori goi-mailako egoeran jartzen ez bada behintzat.

Ez da ahaztu behar T3 tenporizadorea kanpotik gaitu behar dela –EW pina zirkuituaren masari (lurra) konektatuta.

Zakur zaintzaile (watch-dog) tenporizadorearen karga-puntuak programan ez dira sartuko programa erabat bukatuta eta araztuta egon arte, hots, programaren azken bertsioa lortu arte.

4.3 UART serieko ataka (Hartzaile-Transmisore Asinkrono Unibertsala)

80C552 µC-a serieko bi atakekin ekipatuta dago: UART serieko ataka edo SIO0 eta I^2C serieko ataka edo SIO1. Hala bada, 51 familiako UART serieko ataka serieko ataka asinkronoa da (0 modua sinkronoa izan arren) eta I^2C serieko ataka, aldiz, ataka sinkronoa.

Modu horretan, serieko komunikazioan, serieko komunikazio sinkronoa edota serieko komunikazio asinkronoa izan daitezke.

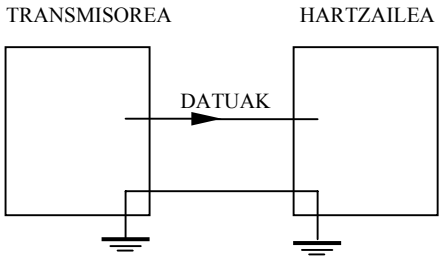
Lehenengo kasuan, hari hauek behar dira: 1 elikatzeko (Vcc), 1 lur-hargunearentzat (GND), 1 datuentzat (hari horretan zehar bidaiatzen dute eta) eta 1 erlojuarentzat edo sinkronizazio-seinalearentzat (haren pultsuek ere bidaiatu egiten baitute). Aldiz, bigarren

kasuan hauek dira beharrezkoak: hari bat Vcc-rentzat, 1 GND-rentzat eta 1 datuentzat (bidaiatu egiten dute eta).

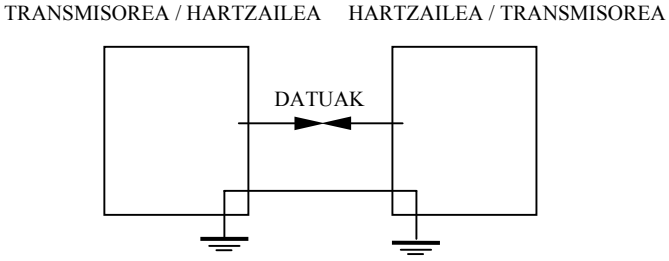
4.3.1 Komunikazio-ereduak

Bai serieko komunikazioan eta baita paralelokoan ere, hiru komunikazio-eredu erabiltzen dira:

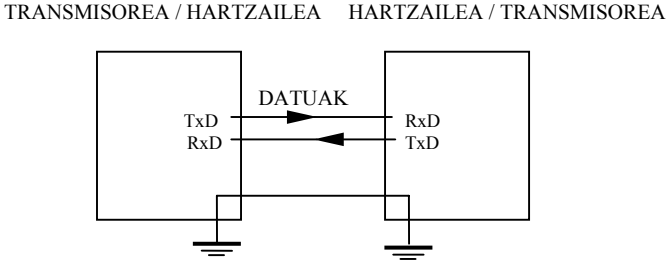
a) **Simplexa edo noranzko bakarreko komunikazioa,** non komunikazioa *noranzko bakarrekoa* den.



b) **Erdi-duplexa (half duplex) edo aldiberekoa ez den bi noranzkoko komunikazioa,** non datuen transferentzia bi noranzkoetan egiten baita, baina ez aldi berean.



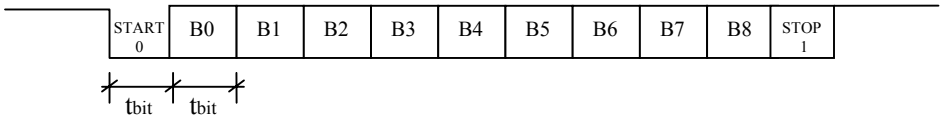
c) **Erabateko duplexa edo aldibereko bi noranzkoko komunikazioa** datuen transferentzia bi noranzkoetan aldi berean egiteko aukera ematen duen komunikazio-eredua da.



51 familiako UART serieko atakak azken komunikazio-sistema hori erabil dezake. Sistema horrekin, RxD pinetik datuak jasotzen ari den bitartean, portu bereko TxD pinetik beste datu batzuk igortzen aritu daiteke.

4.3.2 Serieko komunikazio asinkronoa

Segidan, Protokolo Asinkrono Unibertsala agertzen da:



Bai transmisorea eta baita hartzailea ere inolako arazorik gabe komunika daitezen, biek bat etorri behar dute (ados egon behar dute) puntu hauetan:

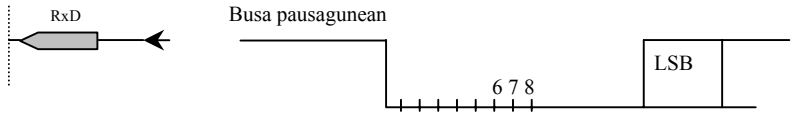
1) Komunikazio-abiaduran, VCOM, hau da, biek abiadura berbera izan behar dute baudetan (bit/s-tan) adierazita.
2) Zenbat datu-bit igorri/hartu behar diren: 5,6,7,8,9.
3) Zein datu-bit igortzen/hartzen den lehenbizi: LSB, hots, pisurik txikienekoa.
4) Komunikazioan erabiltzen den paritatea: BIKOITIA, BAKOITIA, EZ, etab.
5) Zenbat STOP-bit igorri/hartu behar diren: 1, 1'5, 2.

STOP-bita 1 bitekoa izan ordez 2 bitekoa baldin bada, denbora gehiago beharko da komunikazio-abiadura berdinarekin igortzeko/hartzeko.

Paritate-bitak akatsak atzematen ditu datu-komunikazioan, baina ez ditu zuzentzen.

Hartzailearen laginketa-prozesua

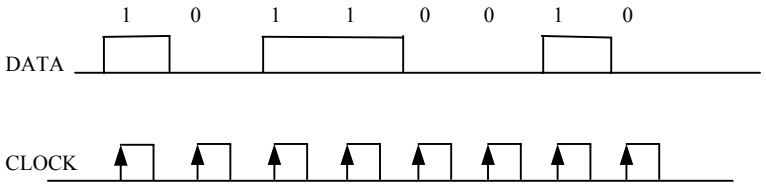
Busa pausagunean dagoenean, busa goi-mailako egoera logikoan edo "1" egoeran egoten da. Egoera aldatzean, hartzaileak START edo hasierako baldintzari zor dion zerbait dela ulertzen du, eta, hori dela bide, hori horrela den ala zarata elektrikoren bat den egiaztatzen du. Horretarako, hartzailea STARTen bitak duen denboran zehar 16 lagin hartzen hasten da: 6, 7 eta 8 laginetatik horietako 2 edo gehiago 0 badira, STARTen bita baliokoa dela ulertzen du eta, orduan, itxaron egiten du datuaren LSB bita edo pisurik gutxienekoa hartzeko, eta hala ez baldin bada, zarataren bat delakoan, ez du aintzakotzat hartzen.



4.3.3 Serieko komunikazio sinkronoa

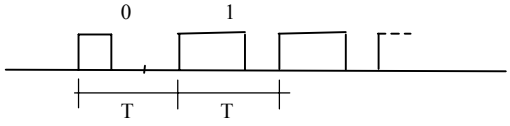
Aurretik aipatu den moduan, normalean 2 hari izaten dira (elikadurakoa eta lur-hargunekoa kontuan hartu gabe): hari batetik datuek bidaiatzen dute eta bestetik erloju-pultsuek. Erloju-pultsuak eskura izanik, transmisorearen komunikazio-abiadurak ez du hartzailearen berdina zertan izan.

Datuak jaitsiera-saihetsetan edo erlojuaren igoera-saihetsetan (gailuen, busaren, eta abarren araberakoak dira) irakurtzen dira.



4-10 irudia. Datu-biten irakurketa igoera-saihetsetan

Dena dela, serieko komunikazio sinkronoa ezar daiteke hari bakar bat erabilita hari horretan zehar datuak eta erloju-pultsuak kodetuta bidaiatzen jarrita, adibidez: T-rentzat, 1. 1/3 T-a bada, "1"egoeran badago“0” esan nahi du, eta 1. 2/3 T-a "1" egoeran izanik, ordea, "1" esan nahi du.



4-11 irudia. Serieko komunikazio sinkronoa hari bakarra erabiliz

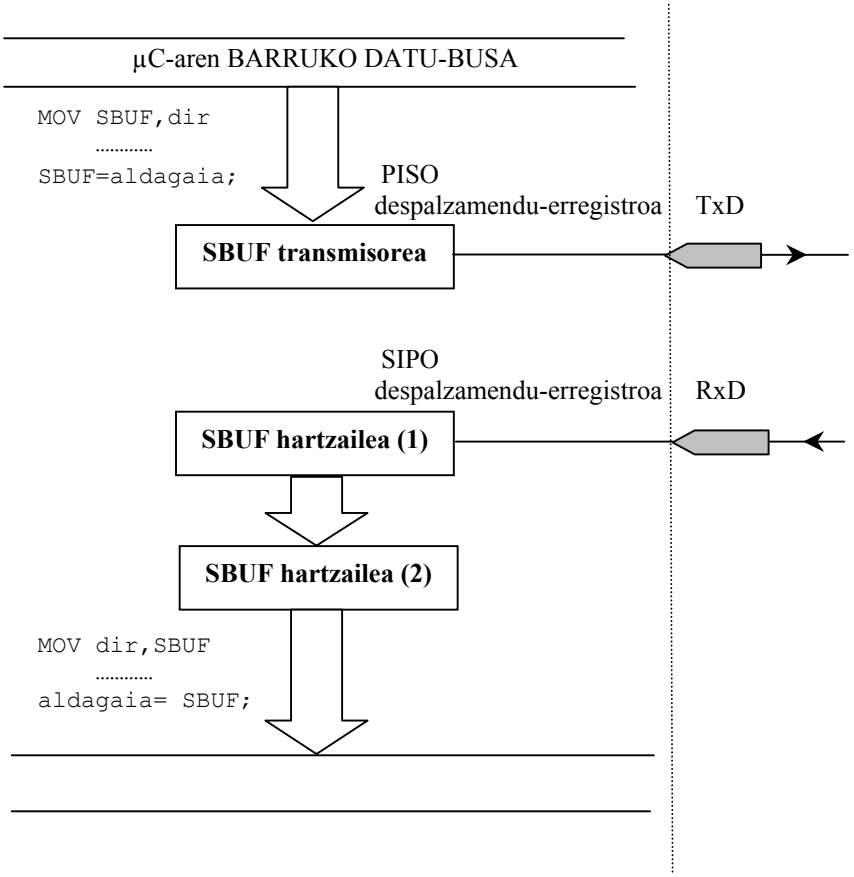
Serieko komunikazio sinkronoaren errendimendua ia % 100ekoa izan ohi da; izan ere, datuek aurretik zein atzetik ez dute eramaten START, PARITATE, STOP eta gisako bitak bezalako beste seinale laguntzailerik. Alabaina, serieko komunikazio asinkronoaren errendimendua UART 1 eran (datuetako 8 bit + start-aren bit 1 + stop-aren bit 1), % 80koa da.

4.3.4 8051 µC-aren UART Serieko Ataka

Ataka honek bi komunikazio-pin ditu, RxD eta TxD, datuak hartzeko eta transmititzeko hurrenez hurren, eta bi SFR bidez kudeatzen da: SBUF datuak hartzeko/transmititzeko, eta SCON ataka kontrolatzeko.

Atakaren barne-zirkuitua 4-10 irudian agertzen dena (bertsio sinplifikatua) baino askoz ere konplexuagoa da, baina programatzaile-mailan bere funtzionamendua ulertzeko, nahikoa da hor erakusten dena. Programatzaileak hainbat gauza jakin behar ditu horren funtzionamenduaz:

- SBUF erregistroa berbera da programatzailearentzat, baina benetan bi dira, nahiz eta baterako ala besterako atzipena erregistro berberean idatziz (MOV SBUF, dir) edo irakurriz (MOV dir, SBUF) egin. Helburu-erregistro moduan erabiltzen denean, transmisiokoa da, eta iturburu-erregistro moduan erabiltzen denean, hartzailea da.
- Aldi berean, buffer hartzailea beste bi buffer-ez osatuta dago: byte osoa 1 hartzailean RxD bidez hartzen denean, RI aktibatzen da eta hartutako edo jasotako zortzikotea 2 hartzailera igarotzen da, 1 hartzailean bigarren zortzikote bat jaso ahal izateko; baina lehenengoa µC-ak irakurri ez baldin badu bigarrena bere osotasunean hartzerako, bi byteetakoren bat galdu egingo da.



4-12 irudia. SIO0-ren barne-egituraren eskema sinplifikatua

Ataka honen kontrol-erregistroa bit-mailan helbideragarria da (*), baina datuena ez. 51 familiako µC-ak, UART portuaz gain I^2C ere badaukanean, erregistro horiek S0BUF eta S0CON moduan ezagutzen dira.

B7	B6	B5	B4	B3	B2	B1	B0	**SBUF**

SM0	SM1	SM2	REN	TB8	RB8	TI	RI	**SCON***

RI: Harrera-etenduraren flag-a (harrera-prozesua eteteko flag-a), RxD pinetik byte bakoitzaren harrera osatzean aktibatzen da eta software bidez desaktibatu behar da.

TI: Transmisio-etenduraren flag-a (transmisio-prozesua eteteko flag-a), TxD pinetik byte bakoitzaren transmisioa (bidaltzea) osatzean aktibatzen da eta software bidez desaktibatu behar da.

Bai RI flag-ak, bai TI-k, etendura-bektore berbera partekatzen dute.

RB8: Harrerako 9. datu-bita; 0 moduan ez da erabiltzen, 1 moduan stop-bita da, 2 eta 3 moduetan 9. datu-bita da eta, normalean, paritate-bit gisa erabiltzen da.

TB8: Transmisioko 9. datu-bita; 0 eta 1 moduetan ez da erabiltzen, 2 eta 3 moduetan, berriz, 9. datu-bita da eta, normalean, paritate-bit gisa erabiltzen da.

REN: Harreraren gaitze-bita da, eta, aktibatuta baldin badago, RxD bidez byteak hartu, SBUF hartzailean kargatu eta RI aktibatzeko aukera ematen du. Deskonektatuta baldin badago, harrera blokeatu egiten da, eta hartutako datu guztiak galdu egiten dira.

SM2: Aurrerago ikusiko dugun multiprozesadore modua egokitzeko bita da. Horregatik, modu hori erabiltzen ari ez bada edota bera aktibatzeak eskaintzen duen ezaugarria espresuki behar ez baldin bada, deskonektatuta egongo da (SM2=0).

SM0 eta **SM1:** Atakaren lau funtzionamendu moduetako bakarren bat zehazten dute.

SM0	SM1	modua
0	**0**	0, sinkronoa
0	**1**	1, asinkronoa, 8 biteko UART-a V_{COM} aldakorrarekin
1	**0**	2, asinkronoa, 9 biteko UART-a V_{COM} finkoarekin
1	**1**	3, asinkronoa, 9 biteko UART-a V_{COM} aldakorrarekin

Funtzionamendu moduak

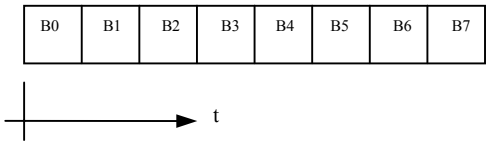
0 modua – *Desplazamendu-erregistroa komunikazio-abiadura finkoarekin*

Komunikazio modu hau seriekoa eta sinkronoa da.

Datuen sarrera eta irteera RxD pinetik egiten da. Datu bakoitza 8 biteko hitz batez osatuta dago, eta lehenbizi irteten edo sartzen (iristen) dena pisurik (esangurarik) txikieneko bita da, hots, LSB bita.

Nola transmisioan hala harreran, erloju-pultsuen irteera TxD pinetik egiten da. Harrerako datu-irakurketa erloju-seinalearen igoera-saihetsetan egiten da.

Hitzaren formatua hau da:



Komunikazio-abiadura finkoa da eta **fosz/12** balio du. fosz=11,0592 MHz denean, komunikazio-abiadura honako hau da:

$$V_{COM}= 11.059 .200/12=921.600 \text{ baud(bit/s)}$$

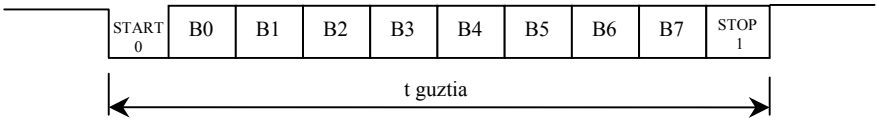
Modu hau µC-aren sarrera- eta irteera kopurua handitzeko erabil daiteke SIO0-ri konektatutako kanpoko desplazamendu-erregistroak erabilita.

1 modua – ***8 biteko UART-a komunikazio-abiadura aldakorrarekin***

Datu-sarrera RxD pinetik egiten da eta datu-irteera, berriz, TxD pinetik.

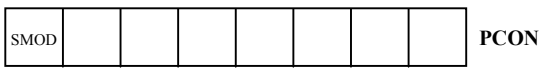
10 bit transmititzen/hartzen dira:
- START bit 1(0)
- 8 DATU-BIT (lehenik LSB)
- STOP bit 1(l), RB8 harrean

Hitzaren formatua honako hau da:



Komunikazio-abiadura aldakorra da y T1 tenporizadoreak zehazten du gainezka egitean. T1 tenporizadorea *tenporizadore* gisa edo *kontagailu* moduan konfigura daiteke bere 4 funtzionamendu moduetan. Normalean, baudak sortzeko 2 moduan erabiltzen da tenporizadore moduan (8 biteko tenporizadorea autobirkargarekin).

SFR PCONen SMOD bitaren egoera ere kontuan hartu behar da; izan ere, aktibatuta baldin badago, bikoiztu egiten du SIO0-ren komunikazio-abiadura UART (1, 2 eta 3) moduetan. Bit hori *abiadura bikoizteko bit* izenez ezagutzen da.



1 moduan (eta 3 moduan) SIO0-ren komunikazio-abiaduraren eskema argigarria 4-13 irudiak agertzen duen moduan gelditzen da.

Komunikazio-abiaduraren balioa formula honek emango digu:

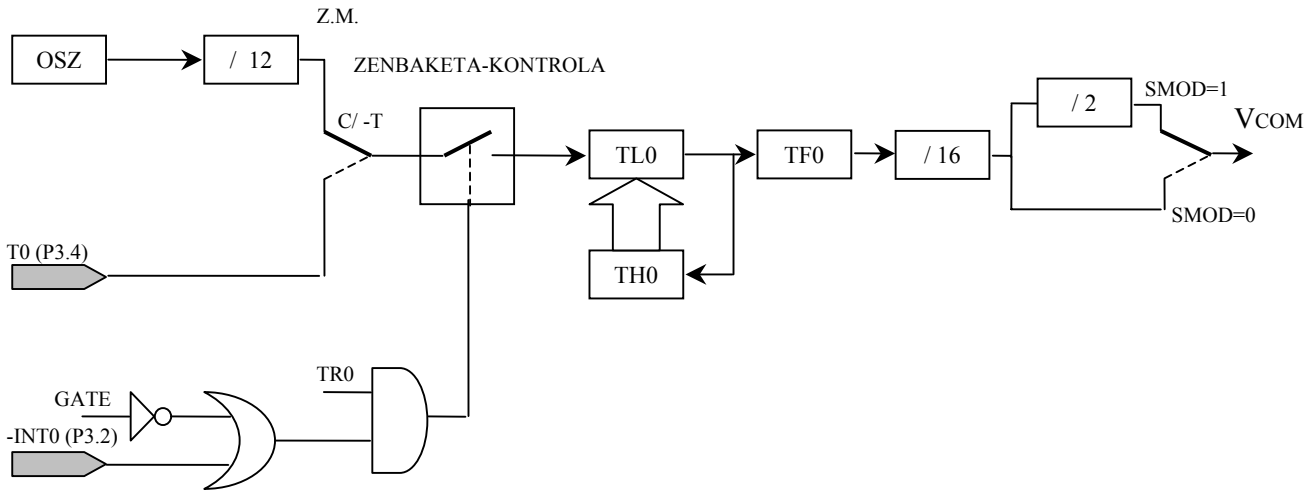
$$\mathbf{V_{COM}=fosz * 2^{SMOD} /12*(256-TH1)*32}$$

1 modu honetan, hartzen den datu bakoitzeko eragiketa hauek egiteko:

- datua SBUF hartzailean kargatu
- STOPeko bita RB8-an kargatu
- eta RI flag-a aktibatu, behar-beharrezkoa da aldez aurretik REN=1 izatea, RI=0 eta SM2=0 eta STOP-aren bita=l edo 0 izatea, hau da, STOPeko bitean akatsa onartzen da (STOP = 0 bitarekin hartutako datua bufferrean kargatzen da eta STOPeko bita RB8an).

edota beste kasua

- eta RI flag-a aktibatu, behar-beharrezkoa da aldez aurretik REN=1 izatea, RI=0 eta SM2=1 eta STOPeko bita =1 (halabeharrez). Kasu honetan ez da onartzen STOPeko bitean akatsik izatea (STOPeko bita = 0 eran hartutako datua ez da kargatzen bufferrean eta galdu egiten da). Kasu hori STOP-eko bitaren balioa egiaztatzeko erabil daiteke.



4-13 irudia. 1 eta 3 moduen komunikazio-abiaduraren sorgailuaren eskema

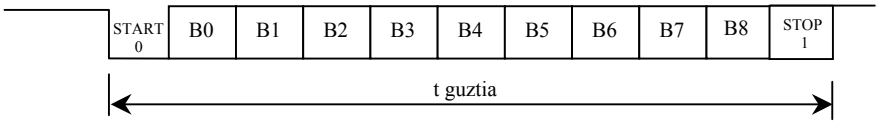
2 eta 3 moduak – ***9 biteko UARTa 2 moduarentzako komunikazio-abiadura finkoarekin eta 3 moduarentzako komunikazio-abiadura aldakorrarekin***

Datuen sarrera RxD pinetik egiten da eta datuen irteera, berriz, TxD pinetik.

11 bit transmititzen/hartzen dira:

- STARTeko bit 1 (0)
- 8 DATU-bit (lehenbizi LSB)
- DATU-bit 1, B8, 9. bita, erabiltzaileak programa dezakeena da, eta, normalean, paritate-bit moduan erabiltzen da. Harreran, RB8 bita da, eta transmisioan, TB8, biak ere SFR SCONekoak.
- STOP-bit 1 (1), harreran kontuan hartzen ez dena.

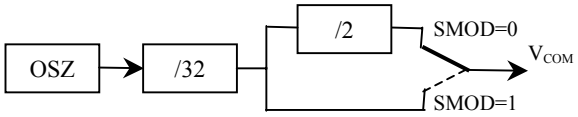
Hitzaren formatua honako hau da:



2 eta 3 moduek komunikazio-abiaduran bakar-bakarrik dute aldea. *3 moduaren komunikazio-abiadura* 1 moduarena lortzen den modu berean lortzen da, hots, T1 tenporizadorearen bidez.

$$V_{COM}=f_{OSZ} * 2^{SMOD} / 12*(256-TH1)*32$$

2 moduaren komunikazio-abiadurak 2 aukera eskaintzen ditu, PCON erregistroaren SMOD bitaren egoera bitarraren arabera:



Eta abiaduraren balioa ematen digun formula honako hau da:

$$V_{COM}=f_{OSZ} * 2^{SMOD} / 64$$

2 eta 3 moduetan, hartzen den datu bakoitzarentzat:
- datua SBUF hartzailean (8 bit) kargatua izan dadin,
- 9. datu-bita RB8-an kargatua izan dadin,
- eta RI flag-a aktibatua izan dadin,

behar-beharrezkoa da aldez aurretik:
- REN=1 izatea
- RI=0 eta SM2=0 eta 9. datu-bita=1 edo 0 izatea.
  Hau da, 9. datu-bita 1 edo 0 izateko aukera ematen da (paritate-bit bikoiti edo bakoitiarekin egindako komunikazioetarako erabiltzen da).

edota beste kasua:

- REN=1 izatea
- RI=0 eta SM2=1 eta 9. datu-bita =1 (derrigor)
  Hemen ez da onartzen 9. datu-bita 0 izatea; izan ere, modu horretan, hartutako datua ez da bufferrean kargatua izaten eta galdu egiten da (9. bita bezalaxe eta RI flag-a ere ez da aktibatzen). Ezaugarri hori multiprozesadorearen inguruneko komunikazioetarako erabiltzen da.

Azken ondorioak

1 moduaren eta 2 eta 3 moduen arteko aldea transmisioan dago. 2 eta 3 moduetan 9. datu-bita bidal daiteke (TB8, PARITATEKOA normalean) eta 1 moduan ez.

Datu-harreran bi modu motak ia berdinak dira, eta aldea hau da: 1 moduak RB8 bita erabiltzen du STOP-bita gordetzeko eta 2 eta 3 moduek, berriz, 9. datu-bita gordetzeko.

8051 µC-aren UART serieko atakaren lau komunikazio moduak taula honetan laburbil daitezke:

Modua	**Transmisio-hasiera**	**Harrera-hasiera**	**SM2 bita** (harreran bakarrik du eragina)
0	TI=0 eta idatzi SBUFen	REN=1 eta RI=0	0 (Ezer ez, hutsa)
1	TI=0 eta idatzi SBUFen	REN=1, RI=0 eta START bita	0 edo 1 (Harreran 1 bada, STOP bitak 1 izan behar du, RB8an kargatzen da)
2	TI=0 eta idatzi SBUFen	REN=1, RI=0 eta START bita	0 edo 1 (Harreran 1 bada, 9. datu-bitak 1 izan behar du, RB8an kargatzen da)
3	TI=0 eta idatzi SBUFen	REN=1, RI=0 eta START bita	0 edo 1 (Harreran 1 bada, 9. datu-bitak 1 izan behar du, RB8an kargatzen da)

Transmisio-amaiera→ TI=1 Harrera-amaiera→ RI=1

4-2 taula. UART serie atakaren funtzionamenduaren laburpena

Bai datuen harrera bai transmisioa 2 eratan kontrola daitezke: etenduraz edo bandera-kontsulta (flag-kontsulta) bidez. Bi eragiketak bi moduekin konbinatuz, 4 full duplex komunikazio modu desberdin lortzen dira. Normalean, transmisioa bandera edo flag-a kontsultatuta kontrolatzen da eta harrera-etenduraren bitartez.

RI eta TI banderek (flag-ek) etendura-bektore berbera partekatzen dute (0x0023, interrupt 4). Harrera, normalean, serieko atakaren etendura bidez kudeatzen denez (transmisioa TI bandera-kontsultaz egiten da eta), horrek UART serieko atakaren etendura gaitzea eskatzen du. Hala bada, transmisioa TI banderaren kontsulta bidez kudeatu arren, portuak transmititzen duen datu bakoitzarentzat serieko atakaren etendura-funtzioa egikarituko da, eta hor funtzio horren egikaritze-zergatiaren araberako iragazpena egin beharko da kodea egikaritzean: kode bat egikarituz RI-ren aktibatzea bada, edo bestea egikaritzea TI-en aktibatzea bada zergatia.

4.3.5 Serieko komunikazioa paritate bikoitiarekin eta paritate bakoitiarekin

Komunikazioetako akatsak detektatzeko kodeen artean, ezagunena, agian, paritate-kodea izango da. Kode hori paritate bikoitiarekin edota paritate bakoitiarekin erabil daiteke. Paritate-kodea bere azken bitaren ondoren datuari gehitzen zaion bita da, eta datuaren batekoen kopurua bikoitia edo bakoitia denaren araberako informazio kodetua darama.

Paritate bikoitiaren metodoa

Datuaren batekoen kopurua *bikoitia* bada, paritate-bita *0* da, eta datuaren batekoen kopurua bakoitia bada, paritate-bita *1* da. Beste modu batean esanda, datuaren

bateko kopuruaren eta paritate-bitaren batuketak beti zenbaki bikoitia izan behar du.
Adibidez, 03H datua baldin badugu, paritate-bita 0 da, kopuru hori sistema bitarrean 2 batekoz osatuta baitago, hots, bateko kopuru bikoitiaz.

Paritate bakoitiaren metodoa

Datuaren batekoen kopurua *bakoitia* baldin bada, paritate-bita *0* da, eta datuaren batekoen kopurua *bikoitia* baldin bada, paritate-bita *1* da. Bestela esanda, datuaren batekoen kopuruaren eta paritate-bitaren batuketak zenbaki bakoitia izan behar du beti.

Ez dago asko begiratu beharrik paritate-metodoak elkarren osagarriak direla konturatzeko; beraz, metodo batetik bestera igarotzeko, beste metodoari paritate-bita alderantzikatu besterik ez zaio egin behar.

Adibidez, 03H datua badugu, paritate-bit bikoitia 1 da, kopuru hori sistema bitarrean 2 batekoz, hots, bateko kopuru bikoitiaz osatuta dagoelako.

8051 UART serieko atakaren 2 eta 3 modu asinkronoetan, 9. datu-bita erabiltzen da parekotasun-bit moduan informazioa transmititzeko/hartzeko (bikoitia edo bakoitia, kasuaren arabera). Bai transmisioan, bai harreran, behar-beharrezkoa da datuetako bakoitzarentzako paritate-bita kalkulatzea. Mikrokontroladore horretan bi modu daude hori egiteko. Lehenengoa hardware bidezkoa da, PSW* (P, 0 bit-a) SFRan aurkitzen den paritate-bit bikoitia erabiliz. Horrek CPUaren metagailuan dagoen 8 biteko datuaren paritate bikoitia ematen digu. Beste bidea software bidezkoa da, eta datu-bitak dituen batekoak zenbatzen ditu banaka-banaka, bit-desplazamenduetan banaka zenbatuz, adibidez. Guk lehenengo modua erabiliko dugu.

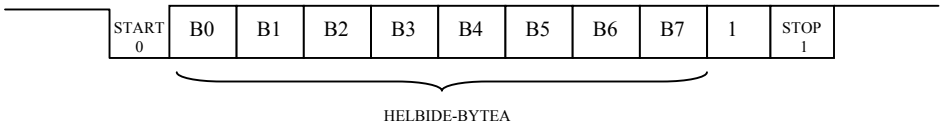
4.3.6 Komunikazioa multiprozesadorearen ingurunean

Multiprozesadorearen ingurune deitua beren artean konektatutako µC-z eratutako sarea baino ez da. 51 familiako mikrokontrolatzaileekin multiprozesadore-ingurune bat sortzeko moduetako bat µC bakoitzaren UART serieko ataken bidez egitea da. Beste aukera I^2C serieko ataken bidez egitea da, baina kasu hori ez da atal honetan aztertuko.

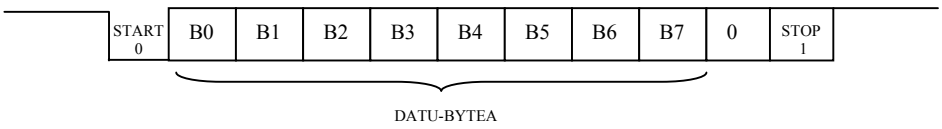
UART serieko atakaren kasuan, hori 2 eta 3 moduetan egin daiteke, non 11 bit igortzen/hartzen baitira (1 start-bit edo hasierako bit , 8 datu-bit, 1 datu programagarrien bit eta 1 stop-bit), baina bi modu horien artean, 3 moduak eskaintzen du malgutasun handiena sistemaren komunikazio-abiadura zehazteko, nahiz eta, era horretan, T1 tenporizadorea okupatuta gelditu komunikazio-abiadura sortzeko.

Morroi bakoitza eginkizun zehatz bat egiten ari da, eta morroi-helbide bat izendatuta dauka.

Morroi bakoitzaren ataka REN=1, RI=0 eta SM2=1 balioekin programatua eta UART atakaren etendura gaituta izango dira, eta horrela, morroi bakoitzak bere 9. bita 1 baliokoa duen maisuak igorritako byteak bakarrik irakurri ahal izango ditu. Byte horiek *helbide-byteak* izango dira.

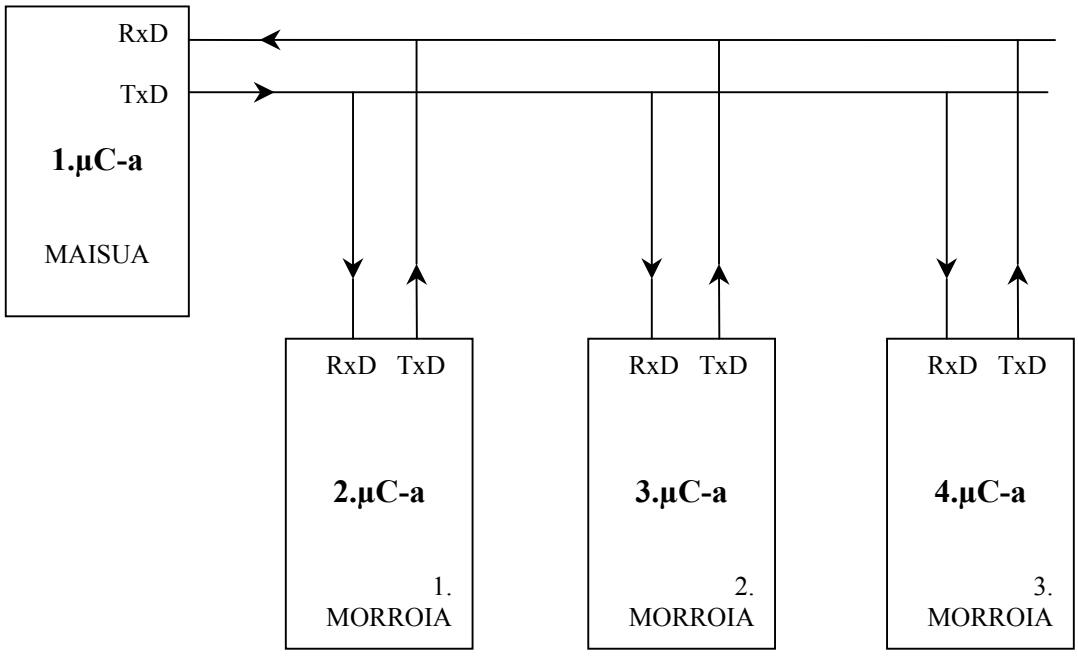


9. bita 0 baliokoa duten byteak *datu-byteak* izango dira.



Prozesadore-maisuak prozesadore-morroi bati datu-bloke bat transmititu nahi dionean:

- morroia identifikatzeko helbide-bytea bidaltzen du.
- helbide-byte bat denez, morroi guztiek hartzen dute. Kasu bakoitzean, RI bita aktibatuko da eta ataza nagusia etengo du. Etenduraren zerbitzu-azpirrutinak hartutako helbidea bere helbidearekin erkatzen du, eta biak bat ez badatoz, morroiak bere lana egiten jarraituko du. Biak bat badatoz, morroiak desaktibatu egingo du bere SM2 (SM2=0) bita, eta, modu horretan, segidan maisuak transmitituko dion datu-blokea hartu ahal izango du.
- transmisioa amaitzeko, maisuak helbide-byte bat bidaliko du. Orduan, morroi hartzaileak detektatu egingo du, eta datu-harrera amaitzeko balio izango du, SM2 bita berriro aktibatuta utzita. Helbide-byte hori transmisioak amaitzeko byte berezia izango da, eta sistemako edozein µC-ren morroi-helbidearekin ez du bat etorri behar.



4-14 irudia. µC-en SIO0-en bidez inplementatutako multiprozesadorearen ingurune baten eskema

4.4 S/I (I/O) pinak

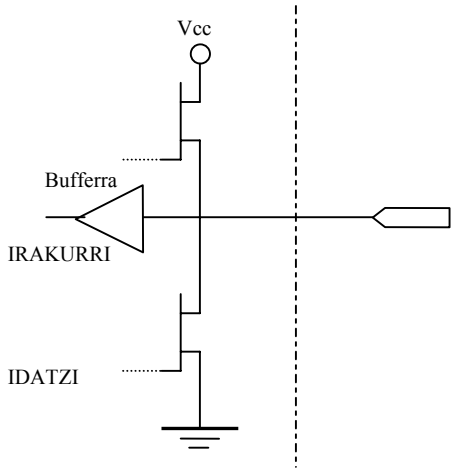
8051 µC-ak S/I-ko (I/O) 4 portu digital ditu eta bakoitza 8 pinekoa da. Sarrera gisa eta irteera gisa, eta gainera modu independentean, programagarriak dira.

Portu bakoitzeko pinen barne-zirkuituak aldatu egiten dira portu batetik bestera portu horietako bakoitzak esleituta dauzkan funtzio gehigarrien arabera, nahiz eta, funtsean, pinak bi multzotan zati ditzakegun eta multzo bakoitzarentzako gutxi gorabeherako zirkuitua erabili.

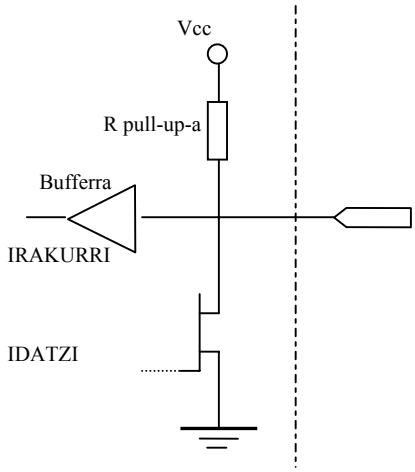
Alde horretatik, P0 portuak gainerakoekiko (P1, P2, P3) duen alde handia dela bide, P0 portua bi noranzkoko portu moduan definituta dago eta P1, P2 eta P3 portuak, berriz, ia bi noranzkoko portu moduan, idazle batzuek bi noranzkokotzat hartzen badituzte ere. Hori honengatik gertatzen da: P0 portuko edozein pinetan ere, 1eko bat idazten denean, pina inpedantzia handiko egoeran gelditzen delako.

Portuetako pinek, funtsean, elementu hauez osatutako barne-zirkuituak dituzte:

- eremu-efektuko transistorea, 0 edo 1 logikoa (idatzita) kanpora bidaltzeko kommutazio moduan funtzionatzen duena
- tarteko memoria edo bufferra, kanpotik sartutako logika-maila irakurri ahal izateko;
- pull-up erresistentzia, ia bi noranzkoko portua bada, edo beste transistore bat, bi noranzkoko portua bada.



4-15 irudia. P0 portuaren pina

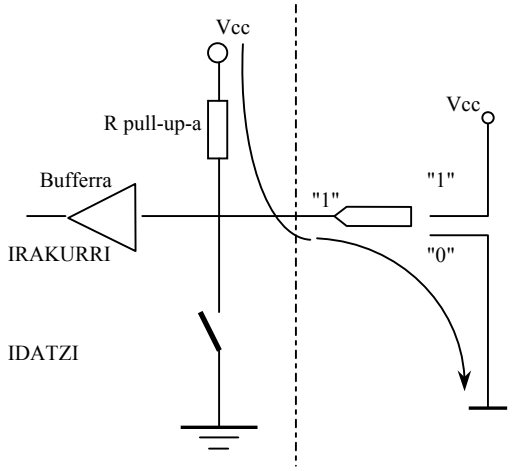


4-16 irudia. P1, P2 edo P3 portuen pina

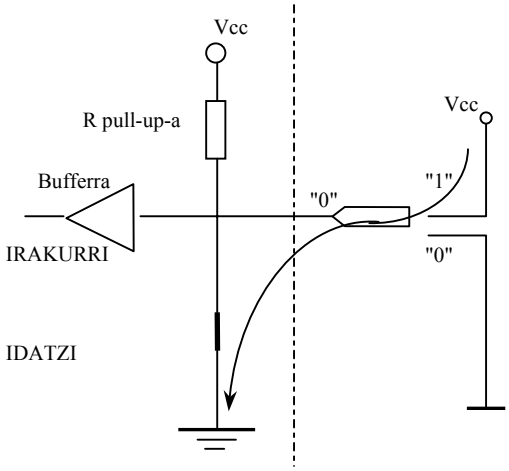
Azter dezagun orain 0 edo 1 idaztean zer gertatzen den bi pin motak aldez aurretik 0an edo 1ean daudela.

4.4.1 Ia bi noranzkoko pina

Pina goi-mailan edo sarrera moduan dagoela (aldez aurretik 1 bat idatzi da softwarez), kanpotik 1a sartzean, 1a irakurtzen da irakurketako bufferrean; baina alderantziz, kanpotik 0a sartzen bada, orduan 0a irakurtzen da. Bestela esanda, kanpotik sartzen dena irakurtzen da, eta horregatik da sarrera modua.



4-17 irudia. Sarrera modua



4-18. irudia. Irteera modua

Kanpotik behe-mailako seinalea sartzean, pinaren barruko aldetik kanpoko lur-konexioraino korronte bat sortzen da, baina barruko pull-up erresistentziaren balioak mugatua; erresistentzia horren balioa kilohm unitate-ordenakoa izan ohi da.

Bestalde, pina behe-mailan egonda, kanpotik behe-mailako seinalea sartzean, halaxe irakurriko da, baina kanpotik goi-mailako seinalea sartuz gero, zirkuitulaburreko korrontea sortzen da, eta horrek pinaren barruko transistorea hondatzen du. Horregatik, pina ez hondatzeko, korronte hori mugatuko duen kanpoko zirkuituaren kalkuluak oso zuzen egin behar dira.

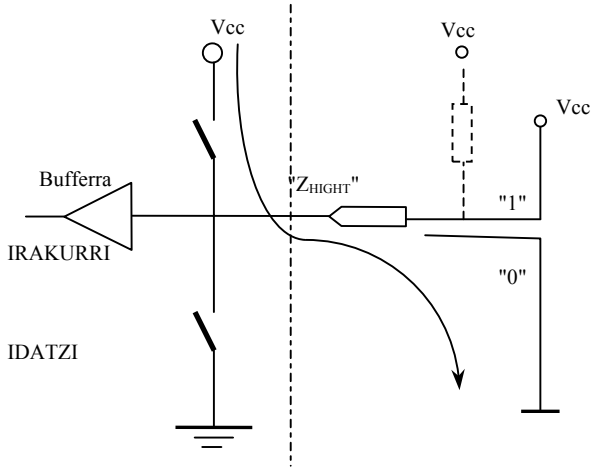
4.4.2 Noranzko biko pina

Pin mota honetan, antzeko zerbait gertatzen da gauza bera ez izan arren; izan ere, barruko pull-up erresistentziaren ordez, une oro ebakidura-egoeran dagoen transistorea dago bertan. Pina drenatzaile irekian dago.

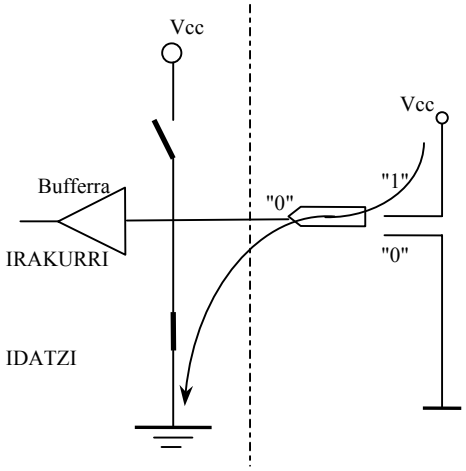
Pinean aldez aurretik 1 idazten bada, pina inpedantzia handiko egoeran gelditzen da, ez baitago tentsio-erreferentziarik edo barneko lur-konexiorik. Kasu horretan, kanpotik 1 seinalea sartzen bada, 1 irakurriko da, eta, 0 sartzen bada 0, irakurriko da.

1 balioa sartzean kanpoan ere 1 balio hori agertzea interesatzen den diseinuetarako, kanpoko pull-up erresistentzia bat konektatzen da. Erresistentzia horrek, pina 1 seinalearen mantentzeaz gainera, badu beste zeregin bat ere: pinera konektatutako zirkuituari korrontea ematea (kanpotik 0 balioa sartzen denean).

Bestalde, ia bi noranzkoko pinarekin gertatzen zen moduan, pina behe-mailan egonda, kanpotik behe-mailako seinalea sartzean, halaxe irakurriko da, baina kanpotik goi-mailako seinalea sartuz gero, zirkuitulabur-korrontea sortuko da, eta korronte horrek pinaren barruko transistorea hondatuko du. Horregatik, pina ez hondatzeko, kanpoko zirkuituaren kalkuluak oso zuzen egin behar dira.



4-19 irudia. Sarrera modua



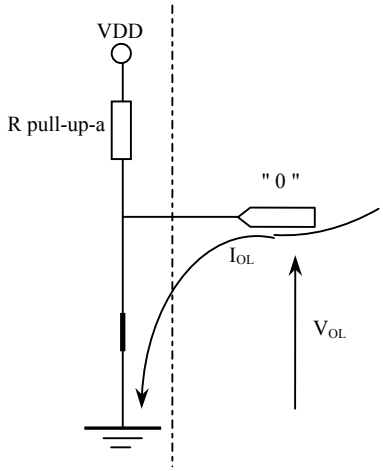
4-20 irudia. Irteera modua

4.4.3 Pinen karga-ahalmena

Irteera moduan portuetako pinen kitzikapen-ahalmena (fan-out) 8 TTL-LS sarrerakoa da P0 portuarentzat, eta 4 TTL-LS sarrerakoa P1, P2, P3 eta P4 portuentzat. P1.6 eta P1.7 pinak, I^2C busaren SCL eta SDA seinale gisa erabil daitezke (80652) eta kasu horretan beren irteerak drenatzaile ireki motakoak izango dira (kanpoko Rpull-upik gabe). P5 portua sarrerarako bakarrik erabil daiteke (80552).

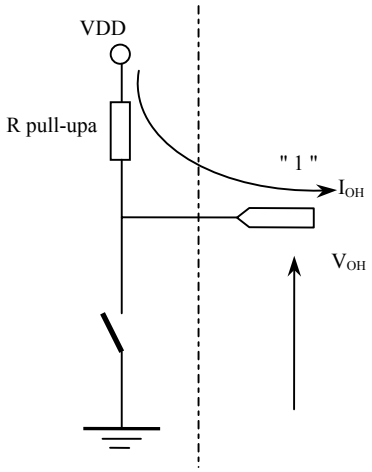
Pin baten irteeran konektatu behar dena TTL-LS zirkuitu digitala ez denean, kalkuluak egin behar dira ea konekta litekeen egiaztatzeko. Beste batzuetan, pinak emandako korrontea agian ez da behar adinakoa izango, eta orduan behar-beharrezkoa izango da transistorezko anplifikadore bat konektatzea. Edozein kasutan ere, jakin beharko da zein diren pinetako bakoitzak bidalitako eta jasotako korronte handienen balioak.

Hala bada, 80C552 µC-aren (*DC electrical characteristics*) portu bakoitzaren pin bakoitzak emandako eta jasotako korronterik handienak azaltzen dira ondoren:



4-21 irudia. Transistore asea

I_{OL}= 1,6 mA P1*,P2,P3 eta P4 portuetan
I_{OL}= 3,0 mA P1.6(SCL), P1.7(SDA) pinetan
I_{OL}= 3,2 mA P0 portuan, eta ALE, -PSEN, -PWM0 eta -PWM1 pinetan



4-22 irudia. Transistorea ebakidura-egoeran

I_{OH}= - 60 µA para V_{OH}= 2,4 V
I_{OH}= - 25 µA para V_{OH}= 0,75 V_{DD}
I_{OH}= - 10 µA para V_{OH}= 0,90 V_{DD}
} P1, P2, P3, P4 portuentzat

I_{OH}= - 400 µA para V_{OH}= 2,4 V
I_{OH}= - 150 µA para V_{OH}= 0,75 V_{DD}
I_{OH}= - 40 µA para V_{OH}= 0,90 V_{DD}
} P0 portuentzat, eta ALE, - PSEN, PWM0 eta PWM1pinentzat

4.4.4 Soinu-sorkuntza 8051 µC-arekin

Erraza da µC-arekin soinua sortzea. Horretarako erabiltzen den elementua mikrokontrolatzailearen S/I (I/O) pin bat da. Lehenbizi, ordea, jakin behar da soinua definitzen duten parametroak honako hauek direla: *maiztasuna*, *bolumena* eta *tinbrea*.

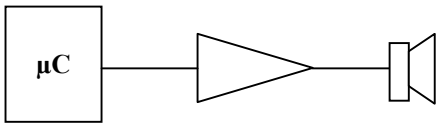
Soinua, definizioz, gizakiak entzun dezakeen seinale ziklikoa da, hau da, 20 Hz eta 20.000 Hz arteko maiztasun-eremuaren barrukoa. Seinalea ziklikoa ez denean, baina entzungarria denean, hots edo zarata deritzo. Musika klasikoan erreferentzia gisa erabilitako soinua diapasoiak sortutako *la* nota da, eta haren maiztasuna 440 Hz da.

Maiztasuna. Soinuaren maiztasuna tonu izenez ezagutzen dena da. Zenbat eta maiztasun handiagoa izan, soinua tonu altuagokoa edo zorrotza da, eta zenbat eta maiztasun txikiagoa izan, soinua tonu baxuagokoa da.

Bestalde, soinu baten *akustika-bolumena* seinalearen anplitudeak emana dator. Zenbat eta anplitude handiagoa izan, orduan eta soinu-bolumen handiagoa izaten du, eta, zenbat eta anplitude txikiagoa izan, orduan eta potentzia edo soinu-bolumen txikiagoa.

Azkenik, *soinu mota* edo *soinu-kolorea,* hots *tinbrea*, seinaleak denboran zehar duen formak definituta dago. Hala bada, maiztasun eta bolumen bereko seinaleek soinu bera ematen dute, baina bakoitzak bere soinu-ezaugarri berezia agertzen du, bere tinbrea (adibidez, bi musika-tresna desberdinek jotako musika-nota berak tinbre desberdinak ditu).

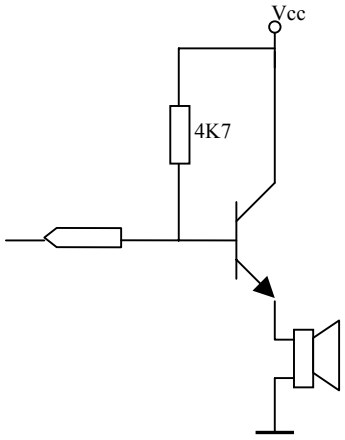
Soinua sortu edo manipulatu behar denean, benetako akustika-seinalea seinale elektriko analogiko baliokide bihurtzen da, eta alderantziz. Horrela, mikrokontrolatzailearekin soinua sortzea oso erraza da: seinale karratua ematen duen S/I (I/O) pin bat bozgorailu txiki bati konektatuta egiten da, anplifikadore soil bat erabiliz.



4-23 irudia. Bloke-diagrama

Gure kasuan, soinua sortzea interesa dakiguke alarma edo akatsak adierazteko, edota baita erabiltzaile batek tekla bat sakatzen duen bakoitzean ohartarazteko ere (telefono mugikorrarekin gertatzen den moduan). Horretarako, ez da beharrezkoa µC-aren eta bozgorailuaren artean konektatu behar den audio-anplifikadorea kalitate handikoa izatea. Beraz, irudian azaltzen denaren moduko soil-soil bat ere eraiki dezakegu.

Soinua sortzeko modu honek maiztasun-parametroa aldatzeko aukera baino ez du ematen; anplitudea eta tinbrea aldaezinak dira.



4-24 irudia. Zirkuituaren eskema

4.4.5 Matrize-teklatuaren konexioa

Teklatu bat µC-ari konektatzeko, behar-beharrezkoa da eskura dauden haren S/I (I/O) pinetako batzuek erabiltzea. Adibidez, 12 teklako telefono-teklatua µC-ari konektatu nahi bazaio µC-ak kontrola dezan, lehenbizi ikusi behar da teklatua nola antolatuta dagoen, hots, zer matrize-eraketa duen.

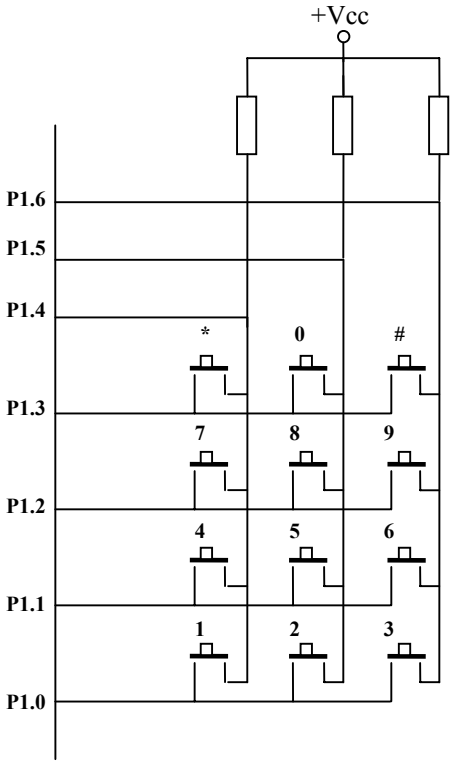
12 teklako telefonoko matrize-teklatua 4 ilara eta 3 zutabetan antolatuta dago, irudian ikusten den moduan:

1	2	3
4	5	6
7	8	9
*	0	#

4-25 irudia. Telefono-teklatua

µC-ari konektatzeko behar den S/I (I/O) pin kopurua matrizearen errenkaden (lerro, ilara) eta zutabeen batura da; kasu honetan: 4 + 3 = 7 pin. Aldi bakoitzean zein tekla sakatu den jakiteko modua hau da:

1. Errenkada-pin guztiak 1ean jartzen dira eta zutabe-pin guztiak 0an, eta teklaren bat sakatu arte itxaroten da. Portuan kargatutako balioa aldatzen ez den bitartean, ez da ezer egiten.
2. Edozein tekla sakatzean, dagokion errenkadaren egoera 0ra aldatzen da, eta horrela jakiten da tekla zein errenkadatan dagoen.
3. Gero, tekla sakatuta dagoen bitartean, portuan, errenkada-pinak 0an eta zutabe-pinak 1ean jartzen dituen balioa kargatzen da, eta behe-egoerara pasatzen den zutabe-pinak tekla zein zutabetan dagoen adierazten du.
4. Tekla zein errenkadatan eta zein zutabetan dagoen jakin ondoren, gauza jakina da zein tekla sakatu den.



4-26 irudia. Telefono-teklatua µC-ari konektatzeko modua

Pull-up erresistentziek bi helburu betetzeko balio dute: lehena, une oro konektatzen diren pinean edo elementuan, behe-mailako zaratak, etab. izan arren, bertan goi-maila logikoa izatea bermatzea da; eta bigarrena, berriz, dagokion pinak ezin eman ahal izan arren, behar duten elementuentzako behar adinako korrontea ematea da.

Tekla sakatzean, *erreboteak* sortzen dira 20 edo 30 bat ms-an, gutxi gorabehera, eta egoera horretan logika-maila handitik txikira igarotzen da irakurketa zuzena bermatu ahal izan gabe. Arazo hori konpontzeko, bi irtenbide daude. Lehenengoa zirkuitu gehigarri bat erabiltzea da, pultsazio-seinale zuzena hartzeko zarata edo hots hori iragaz dezan. Bigarrena, aldiz, software bidez inplementatutako irtenbidea da, eta haren zeregina da erreboteak dirauen denboran irakurketa-unea atzeraraztea.

Gure kasuan, bigarren irtenbidea erabiliko da pultsazioan sortzen diren erreboteak desegiteko.

5. C PROGRAMAZIO-HIZKUNTZA

5.1 Sarrera

5.1.1 Jatorria

C programazio-hizkuntza 1970eko hamarkadan garatu zuen Dennis Ritchie-k. BCPL eta B programazio-hizkuntzetan (motarik gabeetan) oinarritua da, baina hobekuntza garrantzitsuak ditu, hala nola *datu-moten* ekarpena. UNIX® sistema eragilea zuen DEC PDP-11 makinan ezarria izan zen.

Bai UNIX sistema eragilea, bai C-ren konpiladore asko eta baita hainbat aplikazio-programa ere C hizkuntzan idatziak izan dira. Iturburu-kodea makinek egikari dezaketen kodea bihurtzen duten programak dira konpiladoreak, eta *makina*, berriz, konputagailu bereizia da: PC-IBM®, PC-Mc®, 8051 µC-a, etab.

C hizkuntza ezein sistema eragileri eta ezein makinari lotu gabe dagoen hizkuntza da, eta, hori dela bide, erraza da C hizkuntza ia aldaketarik egin gabe erabiliko duten makinetan ibiltzen diren programak idaztea. Ezaugarri horrek garraiagarri edo trukagarri bihurtzen du C hizkuntza.

Gainera, C hizkuntza asmo edo xede orokorreko programazio-hizkuntza da eta abantaila hauek ditu: adierazpen-ekonomia, fluxu-kontrola, datu modernoen egiturak eta eragile multzo aberatsa. Dena dela, C hizkuntza ez da:

- *maila handiko hizkuntza*; izan ere, ez du fitxategietan atzitzeko berariazko atzipen-metodorik, ez eta karaktere-kate, array, taula eta antzeko objektu konposatuak zuzen-zuzenean tratatzeko eragiketak egiteko tresnarik ere.
- *handia*, eta horrek erraztu egiten du bere ikasketa.
- *arloren batean bereziki espezializatua*, baizik eta funtzio-liburutegiak ditu hainbat arlo behar adinako nahikotasunez tratatzeko.
- *multiprogramazioa edo gisa horretako eragiketa paraleloak egiteko aukera ematen duena,* eta fluxu-kontrol egiazkoa eta lineala (sekuentziazko egikaritzea) bakarrik eskaintzen ditu (sekuentziazko fluxu-kontrola).

Urte askotan zehar, C hizkuntzaren benetako estandarra Brian Kernighan-ek eta Dennis Ritchie-k idatzitako *C Programazio-hizkuntza (1978)* izeneko liburuan deskribatutako bertsioa izan zen, nahiz eta antzeko beste bertsio batzuk ere erabiltzen ziren. 1989ko abenduan, C hizkuntza zeharo araututa gelditu zen *ANSI araua* (American National Standards Institute) onartzean, C hizkuntzarentzako *ANSI araua* (estandarra) edo *ANSI C*, hain zuzen ere.

5.1.2 Gaur egun

C hizkuntza gaur egun erabiltzen den programazio-hizkuntzetako bat da. Duela hamarkada bat gaur egun baino askoz ere gehiago erabiltzen zen; izan ere, programazioa erabil zitekeen ia esparru guztietan erabiltzen zen (Fortran®, Pascal®, BASIC®, eta abarrekin

batera). Orain, aldiz, goi-mailako programazio-hizkuntza berriak eta oso goi-mailakoak (arlo berezietarako) erabiltzen dira, eta horiek benetako lehia egiten diote C hizkuntzari. Laburbilduz, honako hau esan daiteke: gaur egun, , mikrokontrolatzaileen hizkuntza mihiztatzaileen ordezko moduan (hardware-kontrola) gelditzen ari da C hizkuntza.

5.1.3 8051 µC-arentzako C hizkuntza

µC-en 51 familiarentzako C hizkuntzadun konpiladoreak sortzen dituen etxeetako bat Keil® etxe alemaniarra da. Ez dira berdinak PC-entzat eta 8051 µC-entzat erabiltzen diren konpiladoreak. Izango dira, noski, konpiladore batekin zein bestearekin konpila daitezkeen programak eta bietako edozeinekin ere akatsik izaten ez dutenak, baina hori ez da normalena; izan ere, konpiladore bakoitza hardware (host edo plataforma) jakin eta zehatz batek egikaritu ahal izateko kodea sortzeko doitzen (diseinatzen) da (PC bat, 80x86 µP-an oinarritutako sistema, ez da 8051 µC-an oinarritutako sistema bera).

Beste marka batzuek ere fabrikatzen dituzte C hizkuntzadun konpiladoreak 8051 µC-arentzat eta 8 biteko beste µC batzuentzat.

Zergatik C hizkuntza mihiztadura-hizkuntzaren ordez? Mihiztadura-hizkuntza erabili ordez goi-mailako hizkuntza erabiltzeak dakarren abantailarik handiena da, hasteko, fabrikatzaileak ezarritako liburutegi-funtzioak dituztela eta funtzio horiek eragiketa mota guztiak egiten dituztela datuekin; hizkuntza mihiztatzailearekin, aldiz, kasu askotan 'eskuz' egin behar dira (sortu egin behar baita).

Programatzaileak ez du arduratu behar erregistroetan eta memoria-kokalekuetan denboraldiko balioak gordetzeaz (biltegiratzeaz). Behar dituen funtzioen aldagaiak eta parametroak erabili behar ditu datu-prozesamendua erosoa eta eraginkorra izan dadin. Edozein modutan ere, beharrezkoak ez diren elementuak (aldagaiak, parametroak, array edo taulak, funtzioak, etab.) ez erazagutzeaz (deklaratzeaz) bakarrik kezkatu behar du sistemaren memoriaren erabilerarik onena egiteko eta erabilitako elementu askok (aldagaiek) memorian izango duten kokapena hautatzeraino ere iritsi, atzipen-abiadura handiagoa edo txikiagoa lortzeko.

5.2 Oinarrizko arauak

5.2.1 Aipamenak (iruzkinak) eta kode-lerroak

`/*Hau aipamena da*/`. Ezin litezke bata bestearen barruan sar.

Kode-lerroak `;` erako adierazpideekin amaitzen dira, `{` elementuarekin hasi eta `}` elementuarekin amaitzen diren blokeak salbu.

```
.                .                .
.                while(1)         while(1)
1sententzia;        2sententzia;     {
.                .                  3sententzia;
.                .                  4sententzia;
                                    }
                                 .
```

5.2.2 Aldagaien eta funtzioen izenak. Erazagupenak

Aldagaien eta funtzioen izenak (identifikatzaileak):

- Letraz edo ikur azpimarratuz hasten dira (_)
- Letrek edo zenbakiek jarraitzen diete (31 arte)
- MAIUSKULAK eta minuskulak desberdinak dira, adibidez; `InitT0()≠initt0()`
- Ezin dira erabili *Hitz Erreserbatu* hauek
  `auto,break,case,char,const,continue,default,do,double,else,enum,extern,float,for,goto,if,int,long,register,return,short signed,sizeof,static,struct,switch,typedef,union,unsigned, void,volatile,while`

Aldagaien eta funtzioen erazagupenak:

- Aldagai eta funtzio guztiak, erabiliak izan baino lehen, erazagutuak izan behar dute.
- Funtzio baten erazagupena bat etor daiteke bere definizioarekin, baldin eta definizio hori haren deia (erabilera) izan aurretik egiten bada.

5.2.3 Datu motak, kalifikatzaileak eta eragileak

Datu motak:

- `void,char,int,float,double,bit`

Kalifikatzaileak:

- `short,long,signed,unsigned,const,volatile`

Hauek dira Keil-en C51 konpiladoreak jasandako datu motak:

Datu mota	Tamaina	Balio-maila
`bit`	bit 1	0tik 1era
`unsigned char`	byte 1	0tik 255era
`signed char`	byte 1	-128tik 127ra
`enum`	2 byte	-32768tik 32767ra
`unsigned short/int`	2 byte	0tik 65535era
`signed short/int`	2 byte	-32768tik 32767ra
`unsigned long`	4 byte	0tik 4294967295era
`signed long`	4 byte	-2147483648tik 2147483647ra
`float/double`	4 byte	+ -1.175494E-38-tik + -3.402823647E+38-ra
`data,idata,pdata erakuslea`	1 byte	8 bit, 0tik a 255era
`code,xdata erakuslea`	2 byte	16 bit, 0tik 65535era
`erakusle orokorra`	3 byte	Memoria mota oro, 0tik 65535era

Eragileak:

- parentesia, `()`
- array-aren indizea, `[]`
- egituretan helbide-gabeziaren eragilea, `->`
- helbide-eragilea, `pc=&c;`
- helbide-gabeziaren eragilea, `c=*pc;`
- objektu-tamainaren eragilea, `sizeof(mota),sizeof(aldagaia)`
- gehikuntza- eta gutxitze-eragilea, `++i; i++; --i; i--;`
- mota-bihurtze behartuaren eragilea edo *cast-a*, `(mota)adierazpena;`
- eragile aritmetikoak, `*,/,%,+,-` (`%` ez da erabiltzen `float` edo `double`-rekin)

- erlazio-eragileak, `>,>=,<,<=,==,!=` (emaitza 0 edo 1)
- eragile logikoak, `!,&& ,||`
- bit-erabilerako eragileak, `,<<,>>,&,^,|`
- baldintza-eragileak, `espr1? espr2: espr3;`
- esleipen-eragileak, `=,+=,-=,*=,/=,%=,<<=,>>=,&=,^=,|=`
- koma-eragilea, `,`

5.3 C51 konpiladorearen atal garrantzitsuak

5.3.1 Memoria motak

Keil-en C51 konpiladoreak aldagai bakoitza kokatuko den memoria mota zehazteko aukera ematen du. Zehazten ez bada, erabiliko den *memoria mota memoria-ereduak* zehaztutakoa izango da, hurrengo atalean aipatzen den moduan.

8051 µC-ak *datuen barne-memoria* du, eta, hor, zuzeneko atzipenaren esparrua eta zeharkako atzipenarena bereizten dira. Gainera, bere esparru batean, `0x20`-tik `0x2F`-rako helbideetan, bit-mailako atzipena lor daiteke. Bestalde, 8051 µC-ak gehienera ere 64 kb posizio helbidera ditzake *kodeen kanpo-memorian* eta *datuen kanpo-memorian*.

`data`	zuzeneko atzipeneko barne RAMa, atzipenik arinena.Tamaina handiena=128 byte
`bdata`	zuzeneko atzipeneko barne RAMa, bit- eta byte-mailan helbideragarria. Tamaina handiena=16 byte
`idata`	zeharkako atzipeneko barne RAMa, atzipen ez hain arina. Tamaina handiena=256 byte
`pdata`	kanpoko RAM orrikagarria, atzipena MOVX @Ri-rekin. Tamaina handiena=256 byte
`xdata`	kanpoko RAMa, atzipena MOVX @DPTR-rekin. Tamaina handiena=64 kilobyte
`code`	kanpoko ROMa, atzipena MOVC @A+DPTR-rekin. Tamaina handiena=64 kilobyte

Aldagai bakarrarekin memoria mota erabiltzeko, hau da, aldagai hori gordeko duen sistemaren memoria zehazteko: lehenbizi aldagai mota idazten da, ondoren memoria mota eta, azkenik, izena:

```
unsigned char data n;
```

Gainera, C51 konpiladoreak aldagaien helbide absolutuak zehazteko aukera ematen dio programatzaileari, hau da, aldagai bakoitza sistemaren zein helbide absolututan (benetakoa) kokatuta egongo den hautatzeko aukera.

```
char xdata ald1 _at_ 0x8000;
```

Ezaugarri hori erabiliz, kanpoko memorian mapatutako I/O (S/I) gailuak zuzenean manipula daitezke.

Aldagaien kokapen absolutua egiteko aukera-murrizketak hauek dira:

- Aldagai absolutuak ezin dira hasieratu erazagupen-lerroan.
- Aldagai lokalak eta funtzioen parametroak ezin litezke koka helbide absolutuetan.
- Bit motako aldagaiak ezin litezke koka helbide absolutuetan.

5.3.2 Memoria-ereduak

Aldagai automatikoek erabiliko duten memoria-eredua, parametro-pasabidetarako eremuak eta memoria motarik gabe erazagututako aldagaiak zehazten ditu memoria-ereduak, beste agindurik ezean. C51 konpiladoreak 3 memoria-eredu ezagutzen ditu:

- `SMALL`, non aldagaiak zuzeneko atzipena duten barneko RAMean biltegiratzen baitira, `data`, gehienera ere 120 byteko. Hau da beste agindurik ezean gauzatzen den eredua. Hau da, halaber, pilarentzat ere erabiltzen den memoria. Beraz, aldagaientzat leku murriztua gelditzen da, nahiz eta erabilera gehienetan nahikoa izan; izan ere, konpiladoreak 8051 µC-aren barne RAM txikia kudeatzen du *overlay* teknikekin (parametroentzat eta funtzioen aldagai lokalentzat izendatutako eremuak gainjarri edo teilakatu egiten dira funtzioek elkarri edota modu errekurtsiboan deitzen ez diotenean).

- `COMPACT`, aldagaiak kanpoko RAMean biltzen dira eta `pdata` memoria mota erabiltzen dute; gehienera ere 256 byteko tamaina dute.

- `LARGE`, aldagaiak kanpoko RAMean biltzen dira eta `xdata` memoria mota erabiltzen dute; gehienera ere 64 kilobyteko tamaina dute.

Programa batean erabilitako memoria-eredua zehazteko, preprozesadorearen `pragma` sasiagindu artezkaria erabiliko da iturburu-fitxategiaren goiburuan, adibidez:

```
#pragma COMPACT
```

5.3.3 C hizkuntzaren preprozesadorea

Preprozesadorea da iturburu-kodea objektu-kodera itzultzeko edo bihurtzeko prozesua gauzatu baino lehen hainbat testu-manipulazio egiten dituen konpiladorearen zatia. C hizkuntzaren preprozesadoreak hainbat agindu edo komando onartzen ditu, hain zuzen ere *preprozesadorearen sasiagindu artezkari* izenez ezagutzen direnak. Hauek dira erabilienetariko batzuk: `include`, `define` eta `pragma`.

`include` sasiagindu artezkariak goiburu-fitxategien edukia bertan den fitxategian sartzen du, idatzita dagoen lerro berean. Goiburuko fitxategiaren izenak, < eta > edo “ eta ” artean idatzita egon behar du. Bi moduen arteko aldea zera da: lehenengo kasuan, fitxategia goiburuko fitxategiei esleitutako direktorioan bilatzen da eta bigarrenean, berriz, bilatze hori egungo direktorioan egiten da, eta, aurkitzen ez bada, goiburuko fitxategiei esleitutako direktorioan edota konpiladorearen ingurunean zehaztutako direktorioan jarraitzen da.

```
#include <stdio.h>
#include “fitxategia_4.c”
```

Bi kasuetan ere, `include` sasiagindu artezkariak fitxategien habiaratzea egiteko aukera ematen du, hots, fitxategiak zeharka beste fitxategi batzuen barne har daitezke, adibidez:

```
#include "fitxategia_3" /* Fitxategi honetan, fitxategia_2, fitxategia_3n */
                        /* idatzitako kode guztia barne hartuko da.       */

      **********


#include "fitxategia_2" /* Fitxategi honetan, fitxategia_1, fitxategia_2  */
                        /* eta 3fitxatea barne hartzen dira. */
```

`define` sasiagindu artezkariak ikur bat (makroaren izena) karaktere-kate batez ordezkatzen du. Asko erabiltzen da zenbakizko konstanteak, karaktere-erakoak, kate modukoak eta datu motakoak definitzeko.

```
#define GOI-MUGA  100     /* Zenbakizko konstanteak */
#define BEHE-MUGA  53
#define PI 3.1416

#define END 'e'           /* Karaktere-erako konstantea */

#define 1MEZUA "Akatsa:SIO0 ataka konektatu gabe" /* Kate-erako konstantea */

#define uchar unsigned char      /* Datu motako konstanteak */
#define stuchar static unsigned char
```

Azkenik, `pragma` sasiagindu artezkariak, beste eginkizun batzuen artean, programaren memoria-eredua zehazteko eginkizuna du.

Badaude sasiagindu artezkari gehiago ere, baina nagusiki, eta funtsean, deskribatutako hiru horiek erabiltzen dira.

5.4 Biltegiratze motak

C programazio-hizkuntzak lau aldagai mota desberdin erabil ditzake:

5.4.1 Aldagai lokalak

Barne-aldagai, aldagai automatiko edo *aldagai lokal dinamiko* izenez ere ezagutuak:

- Funtzio baten aldagai pribatuak edo lokalak dira, hau da, eragin lokala dutenak, eta horregatik, beste ezein funtziok ere ez du haietara zuzenean iristeko modurik.
- Funtzioaren gorputzaren barruan erazagutzen dira, hau da, `{` irekitzeko giltza-zeinuaren ondoren.
- Funtzioari dei egiten zaionean bakarrik hasten dira existitzen, eta funtzioa amaitzean desegin egiten dira; horregatik ezagutzen dira dinamiko izenez.
- Ez dute gordetzen azken deiko azken baliorik, zaborra baizik; beraz, sarrera bakoitzean espresuki hasieratu behar dira.

5.4.2 Aldagai orokorrak

Kanpo-aldagai izenez ere ezagutzen dira:

- Programaren funtzio guztientzat komunak edo orokorrak diren aldagaiak dira, eta bere hasieran erazagutzen dira, sasiagindu artezkarien ostean eta `main`-en aurretik.

Iturburu-fitxategi batean, funtzioren batek beste iturburu-fitxategi batean erazagututako aldagai orokorren bat erabiltzea nahi izanez gero, aipatutako aldagai hori lehenengo fitxategian erazagutu behar da, `extern` atributua erabiliz.

- Aldagai lokalek ez bezala, kanpo-aldagaiek programaren amaieraraino mantentzen dituzte beren balioak, eta baita funtzioak bukatu ondoren ere. Sarrera-parametroak eta funtzioen balio itzuliak ordezka ditzakete.
- Aldagai orokor edo kanpo-aldagaien irispena erazagupen-puntutik fitxategiaren amaierarainokoa da. Ezagunak dira `extern` atributua erabiliz erazagututako fitxategietan ere.

5.4.3 Aldagai estatikoak

Aldagai hauek lokalak edo orokorrak izan daitezke:

- *Aldagai lokal estatikoak* aldagai lokal dinamikoak bezalakoak dira, baina gelditu zaien azken balioa gorde egiten dute.

```
funtzioa(char a,int b)
{
 static int d,e=2;
    .
    .
 } /* d d=0-rekin bezala.Lehenengo deian bakarrik hasieratzen dira, */
   /* gero gelditutako azken balioak gorde egiten dituzte(d eta e). */
```

- Aldagai orokor estatikoak lehen ikusitako kanpo-aldagaiak bezalakoak dira, baina erazagutuak diren fitxategian bakarrik dira ezagunak; beste fitxategi batzuetan aurkitzen diren izen bereko beste dei batzuekiko desberdinak dira.
- Bi aldagai motak funtzio desberdinek aldagai batzuei atzipena murrizteko erabiltzen dira.

5.4.4 Erregistro-aldagaiak

- Aldagai lokalak eta funtzio-parametroak bakarrik izan daitezke.
- Konpiladoreak, ahal duen neurrian, programa egikarituko duen makinaren (azkenean, µP-a edo µC-a) barne-erregistroetan kokatuko ditu elementu hauek. Era horretan, lor litekeen atzipen-abiadurarik handiena lortzen da, eta horrela egikaritzapenarena ere, biltegiratze mota hau kasu askotan atzitzen diren elementuei egiten bazaie.

```
funtzioa(register char a,register int b)
{
 register int d;
 .
 .

}
```

- gehiago erabiltzen da µP-etan oinarritutako sistementzat µC-etan oinarritutakoentzat baino; izan ere, gure kasuan, µC-a C hizkuntzan programatzean, `data` memoria motaz oso ondo ordezka daiteke (arinena, SRF eremukoaren bestekoa).

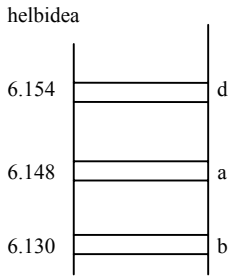
Aldagaiak erabiltzean, ahalik eta aldagai gutxien erabili behar dira, eta, gainera, komeni da aldagai bakoitza bera benetan manipulatu behar duten funtzioek bakarrik ezagutua izatea. Gainerako funtzioentzat ez da existitu behar. Beste modu batera esanda, aldagaien irispena ahal den gehiena mugatu behar da, haien balioak beste funtzio batzuek manipula ez ditzaten.

5.5 Erakusleak

Erakuslea beste aldagai baten helbidea duen aldagaia da. Erakusleak asko erabiltzen dira programazioan; izan ere, erakusleak erabiliz, beste modu batzuen bidez —adibidez array-ak erabiliz— lor daitezkeen baino kode trinko eta eraginkorragoak lortzen dira.

Ondoren datorren adibideak erreminta boteretsu hauen erabilera hobeto ulertzen lagunduko digu.

```
#pragma LARGE /* xdata-n daude memoria mota zehaztu gabeko aldagaiak       */
.
char a,*b,d; /* char motako bi aldagaien eta char datu mota duen erakusle  */
             /* (char datu motara erakusle) baten erazagupenak.           */
             /* b erakusleak oraindik ez du ezein aldagai erakusten.      */
.
.
b=&a;  /* orain b erakusleak a aldagaia erakusten du (a-ren erakuslea da), */
       /* b=6148 */

a=5;
d=*b;  /* d-ri zeharka esleitzen zaio (erakusle bidez) a-ren balioa, d=5   */
.
.
*b=10; /* a-ren hasierako balioa 10-engatik aldatzen da, zeharka.          */
b=&d;  /* Orain, b erakusleak d char motako aldagaia erakusten du, b=6154  */
```



`&` *helbide-eragilea* erakusle bati aldagai baten helbidea esleitzeko erabiltzen da, eta *helbide-gabeziaren eragilea (zehar-eragilea)* `*`, berriz, erakusle batek erakutsitako aldagai baten balioa zeharka esleitzeko, eta erakusle bat erazagutzeko ere bai.

Printzipioz, erakusle batek, objektu mota berezi bat, datu mota berezi bat (`char`,`uchar`,`int`,etab.) erakusten du, eta, ondoren, erazagupenean adierazitako aldagai mota zehatz bat erakusten du, nahiz eta bi urratsak batera ere eman litezkeen.

```
char a,*b=&a;
```

Erakusleekin egin litezkeen baliozko eragiketak honako hauek dira:

- mota bereko erakusleen esleipena
```
int *a,*b,d=4,e;
b=&d;
a=b;
e=*a;
```
- indexaketa
```
a[i] >< *(a+i) /*Baliokideak dira */
```
- erakuslearen eta zenbaki osoaren batuketa eta kenketa
```
a+2; a-3;
```
- array bereko bi erakusleren arteko kenketa edo erkaketa (konparaketa)
```
char a[4],*b,*d,e;
b=&a[0];
d=&a[3];
if(b<=d)
 e=b-d;
```
- esleipena edo erkaketa 0-rekin
```
if(b!=0)
  b=0;
```

C51 konpiladoreak bi erakusle mota onartzen edo ezagutzen ditu:

- *Erakusle bereziak* edo memoria mota berezietan kokaturiko datuen erakusleak (datu-erakusleak); oso eraginkorrak dira, eta konpiladoreak sortutako kodea murriztu egiten dute. (Memoria motaren arabera, 1 edo 2 byte hartzen dituzte).
- *Erakusle orokorrak* edo 8051 μC-aren edozein memoria motatan kokatutako datuentzako erakusleak. 3 byte hartzen dituzte eta, horrela, erakusle bereziek baino kode gehiago sortzen. Kasu batzuetan bakarrik erabiltzen dira.

Aldiz, bit-erako objektuen erakuslerik ez du ezagutzen.

Ezagututako erakusleei dagokienez, hauxe da erakusle ahalen zerrenda:

- `data erakuslea, data` memorian kokatu (`char,int,float,long`, etab.) eta byte 1eko tamaina duen datu-erakuslea
- `bdata erakuslea, bdata` memorian kokatu eta 1 byteko tamaina duen datu-erakuslea
- `idata erakuslea, idata` memorian kokatu eta 1 byteko tamaina duen datu-erakuslea
- `pdata erakuslea, pdata` memorian kokatu eta 1 byteko tamaina duen datu-erakuslea
- `xdata erakuslea, xdata` memorian kokatu eta 2 byteko tamaina duen datu-erakuslea
- `code erakuslea, code` memorian kokatu eta 2 byteko tamaina duen datu-erakuslea
- `erakusle orokorra,` 8051 μC-aren edozein memoriatan kokatu eta 3 byteko tamaina duen datu-erakuslea.

Adibide batzuk azalduko ditugu kontzeptu berri hauek argitzeko:

```
char xdata *erak1;/* erak1 xdata memorian kokatutariko char datu motarako */
                  /* erakuslea da, erakuslea memoria-ereduak zehaztutako  */
                  /* memorian biltegiratzen den bitartean. */

char xdata *idata erak1;/* erak1 xdata memorian kokatutariko char datu    */
                        /* motarako erakuslea da, erakuslea idata memoria- */
                        /* -motan biltegiratzen den bitartean. */
```

8051 μC-arentzako C hizkuntzan idatzitako programak egitean:

- erakusle orokorren ordez erakusle bereziak erabiliko dira, memoria aurrezten eta datuen atzipen lasterragoa egiten baita
- erabilitako aldagaiak erabil litezkeen txikienak izango dira, eta ahal den kopuru txikiena erabiliko da
- sarritan atzitzen diren objektuentzat, `data` memoria mota erabiltzen da, horrela, programaren azken egikaritzapena ahal denik eta lasterrena izan dadin.

OHARRA: PCentzako C hizkuntzan, erakusleak bi motatan sailkatzen dira, orokorretan eta berezietan. *Erakusle orokorrek* ez dute inolako datu motarik zehatz erakusten, eta erazagutzean `void` hitza izaten dute beren izenaren aurretik, nahiz eta une jakin batean mota jakin bat erakustera behartuak izan daitezkeen.

```
void *punt1;
```

Erakusle mota hauek ezin dute erabili `*`, helbide-gabezia eragilea.

5.6 Funtzioak

Funtzioek izugarri errazten dute programazioa; modulutan banatzeko modukoa izatea eragiten dute. Horrek esan nahi du funtzioetan idatzitako kodea modulutan bereizten dela; hala, ataza batzuk beste batzuetatik bereiztea lortzen da: kode osoa `main()` funtzio nagusiaren barruan jarri ordez, ekintza jakinen bat egiten duten taldeak bereizten joaten dira. Gainera, modu horretan programatzeak funtzioak aprobetxatzeko aukera ematen du beste programa batzuetan sortu ondoren, eta aldatu ere egin daitezke, beharrezkoa den kasuetan.

5.6.1 Funtzio motak

Bi funtzio mota daude: liburutegikoak eta erabiltzailearenak. Lehenengoak konpiladorea egin duen etxeak ematen dizkigu. Haien prototipoak, egiten den eragiketa motaren arabera bilduta daude. Adibidez, matematika-funtzioak egiten dituzten prototipoak `math.h` goiburuko fitxategian bilduta daude, sarrera-irteeretakoak `stdio.h` goiburukoan, funtzio grafikoetakoak `graphic.h` goiburukoan, etab. Goiburuko fitxategia *header* izenez ere ezagutzen da.

Erabiltzaile-funtzioak programatzaileak berak sortzen ditu, komeni zaionaren arabera. Batzuetan komenigarria izaten da erabiltzaile-funtzioen prototipoak, liburutegi-funtzioekin egin bezala, goiburuko fitxategi desberdinetan (`.h`) biltzea. Ez dago beste alderik bi funtzio-moten artean; beraz, denak ere konpiladorearentzako funtzioak dira.

Normalean, funtzioak, `main()` funtzio nagusitik deituak izan ohi dira sententzia moduan, baina beste funtzio batzuetatik ere deituak izan daitezke. Liburutegi-funtzioen kasuan, deia egiteko erabiltzen den fitxategiaren goiburukoan, funtzio hori barne duen goiburuko fitxategiaren izena sartu behar da, `#include` komandoa erabilita.

Erabiltzaile-funtzioentzako modu hori erabil daiteke, baina fitxategi batekin nahiz batzuekin lan egin behar bada eta kasua hori ez bada, orduan bi modu daude hori egiteko, deskribatutako urratsen ordenari jarraituz gero:

- Lehenengo modua:
  1. Funtzioa erazagutu (funtzioaren prototipoa).
  2. Funtzioa deitu edo dei egin (sententzia moduan).
  3. Funtzioa definitu (bere ekintzaren edukia zehaztu).
- Bigarren modua:
  1. Definitu (eta erazagutu)
  2. Deitu.

```
#include <stdio.h>         /*--LEHENENGO MODUA-- */

void IdatziMezua (void);  /* hori funtzioaren erazagupena da*/
void main(void)
{
 IdatziMezua();           /* main()-etik deitu */
}
void IdatziMezua (void)   /* erazagupenaren ondorengo definizioa */
{
 puts("Mezu hau PCaren pantailan ateratzen da");
}


#include <stdio.h>         /*--BIGARREN MODUA--*/

void IdatziMezua(void)    /* Funtzioaren definizioa eta erazagupena */
{
 puts("Mezu hau PCaren pantailan ateratzen da");
}
void main(void)
{
 IdatziMezua();           /* main()-etik deitu */
}
```

Funtzioaren izenak zehatz-mehatz berbera izan behar du erazagupenean, deian eta definizioan.

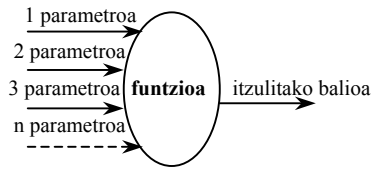
Funtzioen definizioak edozein ordenatan eta hainbat fitxategitan egin daitezke, nahiz eta fitxategiaren barruan edo hainbat fitxategiren artean ere funtzioaren definizioa ezin den zatitua izan. Era berean, funtzio bat ezin da deitu, aurreko kode-lerroren batean erazagutu gabe.

5.6.2 Funtzioen arteko komunikazioak

Funtzioen arteko komunikazioa bi eratara ezar daiteke:

- Sarrera-parametroen eta itzulitako balioen bidez
- Aldagai orokorren bidez

Lehenengo komunikazio modua aztertzean, kontuan hartu behar da funtzio bakoitzak sarrerako parametro kopuru mugagabea izan dezakeela, eta datu mota baten balio bakarra itzuli. Sarrerako parametroak funtzioaren sarrerak dira eta itzulitako balioa bere irteerakoa da.



Modu horretan, honako funtzio mota hauek izan ditzakegu:

- Sarrerako parametroak izan eta balio bat itzultzen dutenak
- Sarrerako parametroak izan eta baliorik itzultzen ez dutenak
- Sarrerako parametrorik izan ez eta balio bat itzultzen dutenak
- Sarrerako parametrorik izan ez eta baliorik itzultzen ez dutenak

Bai sarrerako parametroak eta baita itzulitako balioa ere zein motatakoak izan behar duten zehaztu beharra dago. Sarrerako ezein parametrorik ez dagoenean edota ezein baliorik itzultzen ez denean, `void` gako-hitza erabiltzen da.

Balio bat baino gehiago itzultzea interesatzen bada, trikimailu bat erabil daiteke, alegia, array baten erakusle bat itzultzea, edota datu-egitura bat itzultzea.

Funtzio baten ordez balio bat itzultzeko modua `return(adierazpena)` aurrizkia erabiltzea da, parentesi artean dei egiten dion funtzioari itzuliko dion balioa jarrita. Parentesiak hautazkoak dira, eta haren beharra duen adierazpena itzultzeko bakarrik balio dute.

```
int Batura(int a, int b) /* Batura funtzioaren definizioa eta erazagupena. */
{
 int emaitza;           /* Aldagai honen erazagupena saihestu egin daiteke */
                        /* eta kode lerro bakarra jarri funtzioaren gorpu_ */
 emaitza=a+b;           /* tzean: return(a+b);                              */
 return emaitza;
}

void main(void)
{
 int u,v,guztira;
 u=3;
 v=4;
 guztira=Batura(u,v);
}
```

Funtzioaren prototipoa (erazagupena) erabiltzen den kasuan, parametroen izenak erabili gabe utz daitezke, haien mota hutsak jarriz bakarrik, edota definizioan erabilitako izenak ez diren beste batzuk erabil daitezke.

Funtzioaren deian, parametroei *argumentu* deritze eta parametroetako izen berberez edota desberdinez deituak izan daitezke; izan ere, funtzioaren parametroen izenak funtzioarekiko lokalak dira, eta, beraz, gainerako funtzioentzako ikusezinak.

Teorian funtzio guztiek `return` sententziarekin (`main` sententzia nagusia barne) amaitu behar lukete,, sententzia horrek balio bat itzultzeko eta dei egiten dion funtzioari kontrola itzultzeko balio baitu. Alabaina, konpiladore askok `return` sententzia ez erabiltzea

onartzen dute ezein baliorik itzultzen ez denean; izan ere, funtzioaren gorputza ixteko giltzak kontrola itzultzen dio dei egiten dion funtzioari.

Horrenbestez, `main` funtzioak, `return adierazpena` sententziaz bukatzen bada, adierazpenaren balioa itzuliko du, baina ez beste ezein funtziori, baizik eta egikaritzen ari den hardwareari (PC-ari edo μC-ari). Horrela, azken sententzia horren erabilera aprobetxatu egin daiteke sistemari (hardwareari) programaren egikaritzapena arrakastatsua izan den ala ez adierazteko. `main` funtzioa `return 0`-rekin amaitzen denean, ohi denez, *amaiera normala* adierazten du, eta `return` hori beste balio batekin egiten badu, *normala ez den amaiera* edo *amaiera akastuna* egin duela adierazten du.

Programa batean honako hauek aurkitzen baditugu:

```
int main(void)
{
 .
 .
 return 0x345;
}
```

`main` funtzioaren egikaritzapena amaitzean, programak 0345 balio hamaseitarra itzuliko dio sistemari; PC-a edo 8051 μC-a izan daiteke. Azken kasu horretan, balioa R6(03H) eta R7(45H) erregistroetan biltegiratua izango da.

Funtzioen arteko komunikazioaz esandako guztia gorabehera, bada beste komunikazio modu bat sarrerako parametroen eta funtzioek itzulitako balioen ordez aldagai orokorrak erabiltzen dituena. Horrela, aldagai orokorretan behar dituzten eta sortzen dituzten balioak hartzen eta uzten joaten dira funtzioak, hurrenez hurren.

5.6.3 Parametro-pasatzea

Orain arte, berez funtzioen artean parametroak pasatzeko dauden bi moduetatik bat bakarra ikusi dugu: balio bidez egindako parametro-pasatzea alegia. Beste modua erreferentzia bidez parametroak pasatzea da.

Bi modu horien arteko aldea hau da: *balio bidezko parametro-pasatzean*, `main` funtzio nagusi deitutako funtzioaren argumentuen balioak kopiatu egiten ditu parametroetan, eta parametroren batean balio-aldaketaren bat gertatzen bada, aldaketa hori ez da islatzen dagokion argumentuan. Hori batzuetan abantailatsua da, ezen horrela babestu egiten baitira dei egiten duen funtzioaren datuak. Desabantaila bat ere badu, ordea, eta hau da: balio-pasatzea beste moduan baino astiroago egiten da eta, gainera, memoria kantitate handiagoak behar dira (kontuan hartu bai argumentuak eta baita dagokion parametroak ere mota berekoak izan behar dutela, eta horrek garrantzi handia du tamaina handiko moten kasuan).

Erreferentzia bidezko parametro-pasatzean, deitutako funtzioaren parametro bakoitza hark `main`-en duen argumentuaren erakuslea da. Horrek hau esan nahi du: desabantaila moduan ez dago deitutako funtzioaren datu-babesik, baina abantaila moduan balio-pasatzea baino askoz arinagoa da eta, aldi berean, memoria-aurrezkia lortzen da edozein datu

motarentzat ere parametroan erakusleak duen tamaina konstante mantentzean (1 edo 2 bytean).

Funtzio bati egindako deiak deitzen duen funtzioaren aldagairen bat aldatzea nahi denean, erreferentziazko parametro-pasatzea egin beste erremediorik ez dago. Adibidez, pentsa dezagun ordenamendu-errutina bat dugula eta bi elementu desordenatu trukatu behar ditugula `swap()` funtzioari dei eginda. Kasu horretan, ez da nahikoa honako hau idaztea:

```
swap(a,b);
```

`swap` funtzioa era honetan definituta:

```
void swap(int x, int y)
{
 int temp;
 temp = x;
 x = y;
 y = temp;  /* Prozedura okerra */
}
```

izan ere, balio bidezko deiaren bidez, `swap(a,b)` funtzioak ezin ditzake erasan bera deitu duen funtzioan dauden `a` eta `b` argumentuak. Aurreko funtzioak `a`-ren eta `b`-ren kopiak bakarrik trukatzen ditu. Lortu nahi diren emaitzak lortzeko modua hau da: parametroak erreferentzia bidez pasatuz, hau da, trukatu nahi diren balioen erakusleak erabiliz:

```
swap(&a,&b);
```

`swap` funtzioa era honetan definituta:

```
void swap(int *px,int *py)
{
 int temp;
 temp = *px;
 *px = *py;
 *py = temp;      /* Prozedura zuzena */
}
```

5.7 Egiturak

5.7.1 Egituren definizioa, erazagupena eta erabilera

Egitura bat bi partaide edo gehiagoz osatutako datu mota bat da, eta eremu horiek mota desberdinetakoak izan daitezke. Egitura programatzaileak berak komeni zaionaren arabera sortzen duen datu mota da. Adibidez, jarraian, datu mota berri bat definitzen da, `fitxa` deituriko egitura, lau eremuz (osagaiez) osatuta dagoena: karaktere motako hiru array edo taula eta zeinu gabeko zenbaki oso bat:

```
struct fitxa              /* fitxa egituraren datu motaren definizioa */
{
 char izena[10], deiturak[20], helbidea[30];
 unsigned kodea;
}bazkidea_1, bazkidea_2;  /* fitxa egitura motako bi aldagairen erazagupena*/
```

Egitura mota berri bat definitzeko modu bakarra dago, eta modu hori lerrokada honen aurreko adibideak azaltzen duena da: mota berriaren izena `struct` gako-hitza aurretik duela doa, eta bi giltzen artean gorputza dago, kasu honetan lau eremuz osatua. Egituraren eremuek badute irispen lokal bat egitura barnean; beraz, programan zehar beste aldagai batzuk edota beste egitura batzuetako eremuak ere erazagut daitezke izen berberak erabilita, inolako gainjartze (teilakatze) edo gatazkarik sortu gabe.

Emandako egitura jakin batentzako erazagupena egiteko, aldiz, azken definizio-lerroa aprobetxa daiteke, hala nola aurreko adibidean `bazkidea_1` eta `bazkidea_2` izenekin ageri direnena, edota erazagupena beste lerro batean honako era honetan bereizita egin daiteke:

```
struct fitxa bazkidea_3, bazkidea_4;
```

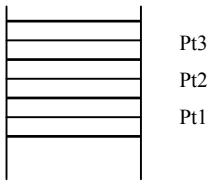
Egitura mota berri bateko aldagai guztiek definizio-lerroaren amaieran erazagutuak izan behar badute, hautazkoa denez, bere definizioan egitura mota berriaren izena jar gabe utz daiteke:

```
struct
 {
  char izena[10], deiturak[20], helbidea[30];
  unsigned kodea;
 }bazkidea_1, bazkidea_2, bazkidea_3, bazkidea_4;
```

Modu bi daude egiturak hasieratzeko: eremuz eremu edota blokean. Lehenengo kasuan, puntu bat erabiltzen da aldagaiaren izena eremuaren izenetik bereizteko, eta, bigarrenean, giltzak erabiltzen dira. Lehenengo modua egitura lokal moduko aldagaientzat nahiz aldagai orokorrentzat erabiltzen da, eta bigarrena, berriz, orokorrentzat bakarrik.

```
struct puntu
{
 int x;
 int y;
};
struct puntu Pt1={4,5}, Pt2;   /* aldagai orokorrentzat bakarrik */

Pt2.x=2;                     /* aldagai lokal nahiz orokorrentzat */
Pt2.y=3;
```

Egiturak, berez, programatzaileak sortutako konposatuzko datu motak dira, eta sistemaren memorian biltegiratzen dira unitate banaezin moduan (nahiz eta bere zatietara bereizita irits daitekeen).



Egiturekin egindako eragiketetan, honako hauek dira balio dutenak:

- Bi egitura aldagaien artean, egituraren eta funtzioaren artean, eta egituraren eta aldagaien artean, unitate gisa egindako esleipena edo kopia.
- Bere helbidea `&`-ekin hartzea, egituraren eta erakuslearen artean.
- Bere eremuetara (osagaietara) iritsi.

Baliozko eragiketen lehen kasuan adierazten den moduan, egiturak funtzioek itzulitako balio gisa erabil daitezke, baina baita sarrerako parametro moduan ere. Ikus ondorengo adibideak:

```
struct BKeragiketa
 {
  int a;
  int b;
 }emaitza;

struct BKeragiketa BatueraKenketa(char x, char y)
 {                /* Balio bidezko parametro-pasatzea */
  emaitza.a=x+y;  /* eta bi zenbaki osoko egituraren itzulera*/
  emaitza.b=x-y;
  return emaitza;
 }
#define UINT unsigned int
struct puntu
   {
    char x;
    char y;
   };

/* Funtzio honek laukizuzenaren azalera kalkulatzen du eta diagonalak */
/* mutur hauek ditu: koordenatu-jatorria eta Pt1 puntua*/
UINT LaukizuzenAzalera(struct puntu Pt1)
   {
    UINT azalera;
    azalera=(Pt1.x)*(Pt1.y);
    return azalera;
   }

/* Funtzio honek puntu-egitura beraz osatutako puntu kartesiar bat itzul_ */
/* tzen  du koordenatuak sarrerako parametro gisa bereizita pasatu ondoren */
struct puntua SortuPuntua(char a,char b)
   {
    struct puntu pun;
    pun.x=a;
    pun.y=b;
    return pun;
   }

/*Funtzio honek sarrerako parametro gisa pasatu zaizkion puntu bikotearen */
/*erdigunea itzultzen du */
struct puntua Erdigunea(struct puntu Pta,struct puntu Ptb)
   {
    struct puntu erdigunea;
    erdigunea.x=(Pta.x + Ptb.x)/2;
    erdigunea.y=(Pta.y + Ptb.y)/2;
    return erdigunea;
   }

void main(void)
   {
    struct puntu puntua_1, puntua_2, puntua_3;
    UINT laukizuzazal;
    puntua_1.x=2;
```

```
    puntua_2.y=3;
    laukizuzazal =LaukizuzenAzalera(puntua_1);
    puntua_2=SortaPuntua(6,5);
    puntua_3=Erdigunea(puntua_1, puntua_2);
   }
```

5.7.2 Egituren erakusleak

Egitura handi bat funtzio batera pasatu behar bada (baliozko parametro gisa), dakarren aurrezkiarengatik, egitura guztia kopiatzea baino askoz ere eraginkorragoa da erakusle bat pasatzea. Azken adibidean definitutako puntuaren egitura motarekin jarraituz,

```
struct puntu *erak1;  /* puntu motarako erakuslearen erazagupena */

erak1=&puntua_1;    /* erakusleari puntu_1-en helbidea esleitu */
```

`erak1`, puntu-egitura motako *puntua_1* aldagaiaren erakuslea da

Egitura baten eremuak ere atzitu daitezke erakusleen bidez.

`erak1->x` edo `(*erak1).x`, `puntu_1`-en x koordenatura
`erak1->y` edo `(*erak1).y`, `puntu_1`-en y koordenatura

5.8 Programaren fluxu-kontrola

C hizkuntzan programa bat egikaritzean prozesadoreak egiten duen bidea kontrolatzeko, kontrol-egiturak erabiltzen dira. Bi motatakoak dira C-k ezagutzen dituenak: hautapen-egiturak eta iterazio-egiturak. Ikus ditzagun banaka.

5.8.1 Hautapen-egiturak

Hautapen-egiturak erabakiak hartzeko erabiltzen dira, adierazpenen balioak egiaztatzen direlarik, eta horrela: behar den sententzia egikaritzen da (0 ez den balioa, + edo -, EGIAZKOA edo TRUE), edota, bestela, hurrengo kode-lerrora igarotzen da (`balioa==0`, GEZURRA edo FALSE), hain zuzen ere, beste hautapen-egitura bat izan litekeenera. Multzo horretan daude `if-else`, `else-if` eta `switch` sententziak.

`if-else` egituran, `else` hautazkoa da eta konpiladoreak aurreko edo azkeneko `if`-ari atxikitzen dio. `if-else` proposamenak habiaratu daitezke.

```
if(adierazpena)     /* 2 eta 3 sententziak beti egikarituko dira, eta      */
  sententzia1;      /* sententzia1, if-eko adierazpenaren balioa 0 ez den  */
sententzia2;        /* kasuan bakarrik, hots, egiazkoa denean.             */
sententzia3;

   ----------------------       ************

if(adierazpena)     /* Kasu bera baina sententzia konposatuarekin */
 {                  /* sententzia-blokea, beti giltza artean.     */
  sent0;
  sent1;
 }
sent2;
sent3;
```

```
   ----------------------        ************

if(adierazpena)  /* 'adierazpena' egiazkoa bada, sent1, sent4 eta sent5  */
 sent1;          /* egikarituko dira, eta gezurrezkoa bada, sent2, sent4 */
else             /* eta sent5. */
 sent2;
sent4;
sent5;

   ----------------------        *************

if(adieraz1)  /* 1, 2 eta 3 adierazpenak egiazkoak badira 1sent,          */
 if(adieraz2) /* 3 eta 4 egikarituko dira. 1 eta 2 adierazpenak           */
  if(adieraz3)/* egiazkoak eta 3 gezurrezkoa bada, 2, 3 eta 4 sententziak */
   sent1;     /* egikarituko dira. Aldiz, 2adieraz gezurra bada, 3sent eta*/
  else        /* 4 bakarrik egikarituko dira. */
   sent2;
sent3;
sent4;
```

Beste bi `else` gehiago ere elkar daitezke beren hurrenez hurreneko sententziekin lehenengo bi `if`-ekin, areago, bi lehenengo `if`-en sententziak bakarrik ere erantsi daitezke.

Baldin eta azkeneko adibidean, `else` bakar hori bigarren `if`-arekin lotzea interesatuko balitz, giltzak erabili behar lirateke erlazioa derrigortzeko:

```
if(adieraz1)
 if(adieraz2)
  {
   if(adieraz3)
    sent1;
  }
 else
  sent2;
sent3;
sent4;
```

Kontuan hartu behar da, halaber, egitura mota hauetan sartutako irekierak iturburu-kodearen irakurketa eta ulermena errazten dituela, baina ez duela inolako eraginik konpiladorearentzat.

`else-if` egitura erabaki anitzetan erabiltzen da, hau da, `else if` sententzietako bat egikaritzen denean (dagokion adierazpena egiazkoa izateagatik) zikloa amaitzen da. Litekeena da, halaber, `else if` sententziaren ezein adierazpen ere ez egikaritzea bertako adierazpen bat ere egiazkoa ez delako; baina amaieran hautazko *else* bat gehi dakioke, kasu horretan bakarrik egikaritzeko.

```
if(adieraz1)
  sent1;
else if(adieraz2)
  sent2;
else if(adieraz3)
  sent3;
.
.
else          /* Hautazkoa, eta else if proposameneko adierazpenetako bat */
  sentN;      /* ere betetzen ez bada bakarrik egikaritzen da. */
sentX;
sentY;
```

switch proposamena. Proposamen hau ere erabaki anitzentzako da, baina adierazpenak oso (`int`) edo karaktere (`char`) motako konstanteak izan daitezke bakarrik.

`else if` egituraren moduan erabilia izan daiteke:

```
switch(int/char motako adierazpena)
  {
   case konst1: sent1; /* Break-ak hautazkoak dira eta egituratik     */
                break; /* irtetea eragiten dute konstanteren bat      */
   case konst1: sent2; /* hasierako adierazpenarekin bat datorrenean. */
                break;
   case konst3: sent3;
                break;
   .
   .
   default: sentN;  /* Hautazkoa, adierazpen osoa ezein */
            break; /* konstanterekin bat ez badator egikaritzen da. */
  }
```

edo `if-else` proposamen baten moduan:

```
switch(oso/char motako adierazpena)
  {
   case  konst1:
   case  konst2:
   case  konst3:  sent3; /* if((adi==konst1)||(adi==konst2)||(adi==konst3))*/
                  break; /*   sent3;                                        */
   case  konst4:  sent4;
                  break;

   default: sentN;       /* else                                           */
            break;      /*   sentN;                                       */
  }
```

5.8.2 Iterazio-egiturak

Iterazio-egiturak haietan sartutako sententziak errepikatuta egikaritzeko erabiltzen dira. Egitura mota horietan, beren gorputz barruan dauden sententziak modu errepikatuan egikarituko dira, baina begiztan irauteko baldintza egiazkoa baldin bada bakarrik. Hiru iterazio-egitura daude C hizkuntzan: `while`, `for` eta `do-while`.

while proposamena: `while`-ren barruan dagoen sententzia soil edo konposatua egikaritzen da adierazpen-baldintza egiazkoa den bitartean. Baldintza hori gezur bihurtzen denean, hots, haren balioa 0 izatera igarotzen denean, `while` begiztatik irten egiten da, ondorengo sententziak egikaritzen jarraitzeko.

```
sent0;
while(adieraz1) /* adieraz1 balioztatu eta egiazkoa bada, sent1 egikaritu */
   sent1;
sent2;
sent3;


sent0;
while(adieraz1)
```

```
      {
       sent1;
       sent2;
      }
   sent4;
   sent5;
```

for egitura; hari lotutako sententziaren egikaritze errepikatua baldintzatzen duten hiru adierazpenez osatuta ere egon daiteke, eta, aldi berean, soila (sinple edo bakun) edo konposatua izan daiteke. Ezkerretik eskuinera, lehenengo adierazpena *hasieratzekoa* da, bigarrena *iraupenekoa* eta hirugarrena *bilakaera* edo *aldaketakoa*.

```
sent0;                          /* Sartzean, adieraz1 egikaritzen da         */
for(adieraz1;adieraz2;adieraz3) /* (hasieratzea), jarraian adieraz2          */
  sent1;                        /* (iraupena), eta, hori egiazkoa bada,      */
sent2;                          /* sent1 sententzia elkartua eta adieraz3    */
                                /* (bilakaera). Ondorengo zikloetan adieraz2,*/
                                /* sent1 eta adieraz3 egikaritzen dira (eta  */
                                /* ordena horretan egikaritu ere). adieraz2  */
                                /* egiazkoa ez denean, for begiztatik irten  */
                                /* egiten da. */


                                   ************

   for(adieraz1; ;adieraz3)      /* adieraz2 ez izatea EGIAZKO baldintza  */
     sent1;                      /* gisa ulertzen da, eta begizta infinitua */
                                 /* lortzen da. */

                                    ************

   for(adieraz1,adieraz1';adieraz2;adieraz3,adieraz3')
     1sent;
                                /* Hasieratze- eta  bilakaera-adierazpenak */

                                 /* hainbat izan daitezke */
                                /* eta ezkerretik eskuinerantz     */
                                /* balioztatzen (ebaluatzen) dira. */
```

do-while egitura `while`-ren berdina da, baina atxikitako sententzia gutxienez behin egikaritzen da; izan ere, lehenbizi egikaritu egiten da, eta gero gauzatzen da egituraren adierazpena egia ala gezurra den jakiteko egiaztapena.

```
sent0;
do
 sent1;
 while(adieraz1);
sent2;
sent3;
```

break sententzia ziklo bat uzteko erabiltzen da; beraz, proposamen batzuetan, beretatik irteera aurreratua egiteko erabil daiteke. Hautazko erabilera ere eman dakioke `switch`, `while`, `do-while` eta `for` proposamenetan.

```
for(adieraz1;adieraz2;adieraz3)
  for(adieraz1';adieraz2';adieraz3')
    for(adieraz1'';adieraz2´´;adieraz3´´)
       {
        .
        .
        break;                          /*hirugarren for-etik irteten da */
```

```
        }
        .
        .
```

goto sententziak aukera ematen du programaren egikaritzapenak `goto` hitza dagoen lerrotik bertan adierazitako etiketaraino jauzi egiteko.

```
for(adieraz1;adieraz2;adieraz3)
  for(adieraz1´;adieraz2´;adieraz3´)
    for(adieraz1´´;adieraz2´´;3´´adieraz2´´)
       {
        .
        .
        goto etik_1;              /* sent5 egikaritzera jauzi egiten du */
       }
.
sent3;
sent4;
etik_1:sent5;
```

5.9 Array edo taula dimentsioaniztunak

Bi dimentsioko array edo taulak:

```
char xdata arr2[2][3]={1,3,5,2,4,6};
```

idata,.. lerroak zutabeak

Keil-en softwarearen kanpo-memoriaren leihoak ondorengo hau erakutsiko liguke:

X:	00	01	02	03	04	05	06	..	0F
0000	01	03	05	02	04	06			

µC-ak memorian elementu bakoitzaren balio hamaseitarra gordetzen du.

Dimentsio bakarreko ondorengo *array* edo *taula* erabiliko bagenu, memorian izango lukeen kokapena berbera izango litzateke, baina, erabiltzaileari dagokionez, array edo taularen elementu bakoitzaren atzipena desberdina da.

```
char xdata arr1[6]={1,3,5,2,4,6};
```

arr1[n]

0..5

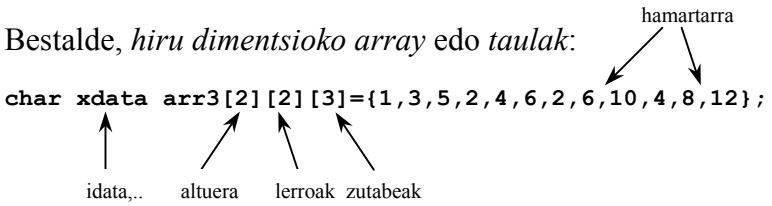
2arr[f][c]

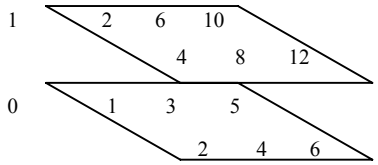
	0	**1**	**2**
0	1	3	5
1	2	4	6

Praktikan, *bi dimentsioko array* edo *taulak* bi dimentsioak balituzte bezala hasieratzen dira. Konpiladorearentzat horrek inolako eraginik ez duen arren, badu, ordea, iturburu-kodea irakurtzen duen programatzailearentzat; izan ere, array edo taularen egitura erakusten dio:

```
char xdata arr2[2][3]= {   {1,3,5},
                           {2,4,6}, };
```

Bestalde, *hiru dimentsioko array* edo *taulak*:





Keil-en softwarearen kanpoko memoriaren leihoak ondorengo hau erakutsiko liguke:

X:	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0000	01	03	05	02	04	06	02	06	0A	04	08	0C				

Bere *dimentsio bakarreko array* edo *taula* baliokidea ondorengo hau izango litzateke:

```
char xdata arr1[12]={1,3,5,2,4,2,6,10,4,8,12};
```

Array edo taularen elementuak hasieratzen direneko balioak komatxo artean baldin badaude, ASCII kodearen balioak biltegiratuko dira:

```
char xdata 1arr[4]={'0','4','F','A'};
```

X:	00	01	02	03	04	05	06	...	0F
0000	30	34	46	41					

Alderantziz, gauza bera egiten bada, baina komatxo soilak erabili gabe:

```
char xdata 1arr[4]={0,4,F,A};
```

konpiladoreak akatsa sortuko luke; izan ere, F eta A beretzat ezezagunak diren balioak dira, eta hori saihesteko balio hamaseitarrak direla adierazi beharko litzaioke:

```
char xdata 1arr[4]={0,4,0xF,0xA};
```

emaitza berbera lortzeko beste modu bat ondorengo kode-lerro hau egikaritzea izango litzateke:

```
char xdata 1arr[4]={0x0,0x4,15,10};
```

5.10 Bit motako aldagaiak

Bit motako aldagaiek bi balio posible eduki ditzakete, 0 edo 1; horietako lehenengoa gezurtzat, faltsutzat edo positibotzat har daiteke, eta bigarrena, berriz, egiazkotzat, negatibotzat, etab.

C51 konpiladoreak aukera ematen du bit motako aldagaiekin data, idata eta bdata memoria motak erabiltzeko, baina akatsa ematen du xdata, pdata eta code aldagaiak erabiltzen badira. Benetan gertatzen dena zera da, aldagai mota horiek `bdata`-n soilik kokatzen dituela, hau da, bit mailan helbideragarri den barne-RAMaren esparruan (0x20..0x2F).

```
bit DN,KV;      /*Bit motako bi aldagairen erazagupena.        */
DN=0;          /*Bit motako aldagai bati balio bat esleitu.   */
.
KV=1;
```

Lorgarria da, noski, bit motako aldagaiak byte tamainakoa edo handiagoa den aldagai bateko osagaiak izatea, eta, era horretan, banaka edo orokorki manipulatuak izan daitezke (bit mailan atzituak izan litezkeen SFRak balira bezala). Horretarako, ezinbesteko baldintza da den-denak aldagai orokor bezala erazagutzea.

```
char bdata ALD;  /* ALD izeneko aldagai orokorraren erazagupena bdata-n */
sbit AV= ALD^0;  /* ALD-en behereneko bit bietan kokatutako AV eta BV   */
sbit BV= ALD^1;  /* bit motako aldagai biren erazagupena. */
```

Bit motako aldagaiek ezin dezakete:

- bit-array edo bit-taularik eratu
- ezin daitezke ezein erakuslek erakutsiak izan

Alabaina, funtzioek onartzen dituzte bit erako parametroak eta bit bat itzul dezakete.

Hitz bateko bitak, hitz horrek duen tamainaren arabera, eskuinalderantz nahiz ezkerralderantz lekualdatuak izan daitezke.

```
int n=1; /* erazagutu eta hasieratu egiten da      n=0000000000000001 */
n<<=8;   /* 8 kokaleku ezkerrerantz desplazatua    n=0000000100000000 */
n>>=1; /* 1 kokaleku eskuinerantz desplazatua      n=0000000010000000 */
```

`n<<=8;` adierazpena, `n = n<<8;` adierazpenaren baliokidea da.

5.11 Konstanteak

Programan zehar balioa konstante mantentzen duten elementu erazagutuei konstante deitzen zaie, eta orokorrak nahiz lokalak izan daitezke. Erazagutzeko bi modu daude: lehenbizi beren mota adieraziz eta ondoren konstanteak direla esanez, edota alderantziz.

Konstante bat erazagutzen denean, erazagutze-lerro berean hasieratu behar da; izan ere, konpiladoreak ez du uzten erazagupen-lerrotik kanpo inolako baliorik ematen (ez baitira aldagaiak).

```
const char a=0xAA;
int const  b=0x2341;

char const 1arr[]={2,4,6,8};
const char 2arr[]={1,3,5,7};
```

Keil-en C51 v 4.01 konpiladorea erabiltzen denean eta konstante-arrayak edo konstante-taulak erazagutzea (eta hasieratzea) interesatzen denean, `code` memoria mota erabili beharko da, ezen `const` gako-hitzarekin akatsak sorrarazten baitira konpiladotueran:

```
char code 1arr[]={2,4,6,8};
```

5.12 Casting edo datu mota beharketak C hizkuntzan

Egitera goazen programetan eragiketa aritmetikoak nahiz esleipenak egineran, kontuan izan behar da makinak egiten dituen balio-truke edo -mozketak eta datu moten beharketa edo moldaketak. Horrelakorik balego, jakin dezagun zein izango den emaitza. Horri dagokionez, ikus ditzagun programazio-kapitulu honen hiru puntu nagusiak, hau da, honetaz jakin behar ditugun hiru gauza garrantzitsuenak:

a) C hizkuntzan, *osoko zenbakien zatiketak emaitza zatiketaren osoko baliora mozten du*, eta zatiki hamartarra guztiz galtzen da. Ikus dezagun *1.adibidea*:

```
float c;
int a=4,b=3;

c=a/b; /* c=1.000000 , 1.333333en ordez */
```

b) *Moldaketa aritmetikoei (beharketa naturalak)* dagokienez, eragiketa aritmetiko bateko eragigai guztiak datu mota komun batera pasatzen$_2$ dira (handiena eta doitasun handienekoa), eta emaitzaren datu mota berekoa izan behar du*. Beharketa horiek eragiketa aritmetikoen eragigaietan eta esleipenetan (aldagaiak, funtzioen sarrera-parametroak, funtzioek itzulitako balioak, erakusleak) gertatzen dira.

Horrela, *1.adibidean*, zatiketaren emaitza moztu gabea interesatuko balitzaigu, eragigairen bat emaitzaren datu mota berdinekoa dela erazagutzea nahikoa izango litzateke *(suposatuz eragiketaren emaitza irudikatzeko gai dela, hau programatzailearen esku geratzen da). Horrela, `1.33333`, emaitza gordeko duen datu motarengatik irudikakorra da: float. Ikus dezagun *2.adibidea*:

```
float a=4, c;
int b=3;

c=a/b;
```

Inondik ere, jarraian dugun kodearen egikaritzeak ere emaitza berbera ematen du,

```
float b=3, c;
int a=4;

c=a/b;
```

Beharpen aritmetikoetarako, hau da datu-moten zerrenda eta haien funtzionamendua, doitasun handieneko eta handien direnetatik hasita, doitasun txikieneko eta txikienetaraino:

```
long double,double,float,unsigned long int,long int,unsigned int,
int,unsigned short int,short int,unsigned char,char
```

- eragigairen bat `long double` bada, bestea ere `long double` bihurtzen da, eta eragiketaren emaitza `long double` izango da.
- bestela, eragigairen bat `double` bada, bestea `double` bihurtzen da, eta emaitza `double` izango da.
- bestela, eragigairen bat `float` bada, bestea `float` bihurtzen da, eta, orduan, emaitza `float` izango da.
- koma mugikorreko eragigairik ez badago, `char` eta `short int` bihurtzen dira.
- gero, eragigairen bat `unsigned long` bada, bestea ere `unsigned long` bihurtzen da, eta emaitza `unsigned long` izango da.
- bestela, eragigairen bat `long` bada, bestea ere `long` bihurtzen da, eta, orduan, emaitza `long` izango da.
- bestela, eragigairen bat `unsigned int` bada, bestea ere `unsigned int`, bihurtzen da, eta emaitza `unsigned int` izango da.
- Eta, bestela, orduan eragigaiek `int` izan behar dute, eta emaitza `int` izango da. Eta, bestela, eragigaiek `int` motakoak izan behar dute eta emaitza `int` izango da.

Adierazpen batean ez badago koma higikor motako eragigairik, `char` eta `short int` motakoak `int` motara bihurtzen dira automatikoki, non lehen kasuan ez baita zeinuaren hedapena (iraupena) ziurtatzen, eta bigarrenean bai.

`int->long int` mota-beharpenak bai, ziurtatzen du zeinuaren iraupena.

`int->unsigned int` mota-beharpena kontzeptuala da soilik, ez da izaten inolako aldaketarik zenbakiaren barne-adierazpenean (gordeeran), kanpo-adierazpenean (interpretazioan). Hori guztia ondorengo iturburu kodearen (programaren) bidez frogatzen da:

```
 unsigned int b;
 int a=64000;

 b=a;

 printf("Esleipenaren emaitza honako hau da: %u %d %lf
(%d)\n",b,b,a);
 Esleipenaren emaitza honako hau da: 64000 -1536 64000.000000 (-1536)
```

5-1 irudiak frogatzen du zeinudun zein zeinurik gabeko barne-adierazpena berdinak direla.

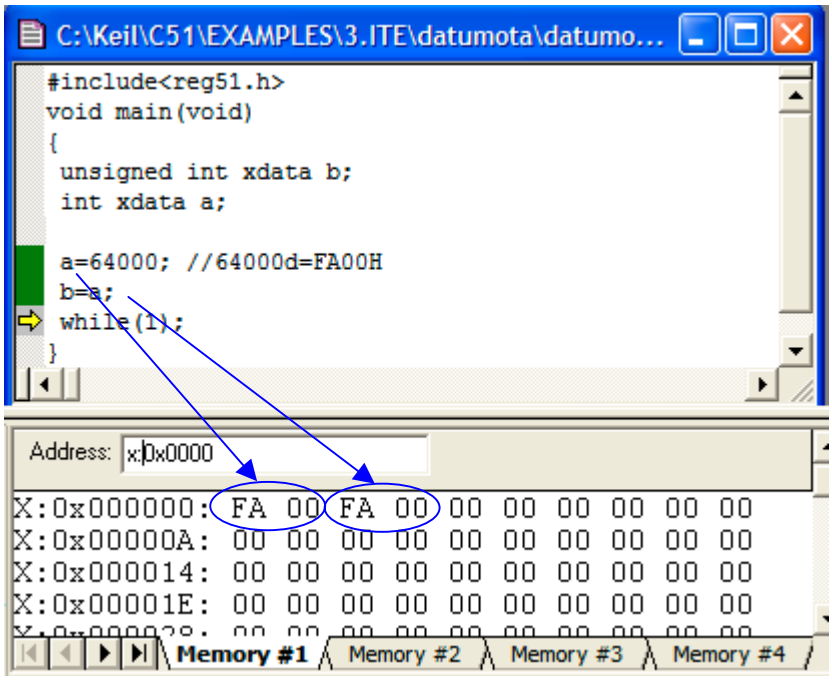
`unsigned char` eta `unsigned short` datu motak `unsigned int` datu motara bihurtzen dira beti. `c unsigned char` motakoa bada, ez du zentzurik ondorengo konparazio-adierazpenak

```
unsigned char c;

if(c==-1)
```

sekula ez baita benetakoa izango.

`char->int`, `short->int`, `int->long` bihurpenek, zeinudunak nahiz zeinurik gabeak izan, beti izango dute arrakasta (lehenak izan ezik, non ez baita zeinuaren iraupena ziurtatzen). Berdin gertatzen da `float->double` bihurpenarekin.

Txikiago diren data-motetarako bihurpenek, aldiz, biltegiratze formatu berdina erabiliz, informazio galeraren bat izan dezakete. Horrela, osokoen taldean, esleitu beharreko balioa ez bada adierazgarria helburu-aldagaiarentzako, overflow edo gainezkatze bat sortzen da. Koma mugikorreko taldean ere overflow-a izaten da osokoen kasu berberean, eta, gainera, doitasuna galtzen da beti.



5-1 irudia. Zeinudun eta zeinugabe osokoaren biltegiratzea memorian

c) *datu mota behartze eragilea edo casting-a* existitzen da, zeinak hasierako motatik programatzaileari komeni zaion motara bihurtzea eskaintzen baitio, kode-puntu horretan. Beharpen naturalak b), programatzaileak nahi duen emaitzarik ematen ez duenean bakarrik du zentzua horren erabilerak.

1.adibidean, `a` aldagaiak eta `b`-k ere osokoak izan behar badute. Orduan, soluzioa bietariko bat float izatera behartzea da eragiketan; horrela, bestea ere float bihurtuko da automatikoki, eta baita emaitza ere:

```
float c;
int a=4, b=3;

c=(float)a/b; /* a flota motara bihurtzen da, b ere bai  */
              /* c= 1.333333 */
```

Baina ez da honako hau egin behar,

```
c=(float)(a/b);/* c=1.000000 , 1.333333-ren ordez */
```

lehenengo mozturiko emaitza lortuko bailitzateke, 1, eta beharpenarekin float bihurtua izanik 1.0000000

C++-entzako OHARRAK:
1) mota beharpenetarako bada beste modu bat ere,

```
float c;
int a=4, b=3;

c=float(a)/b;
```

2) programatzaileak definituriko datu-moten bihurpenak defini daitezke, egokitzen den motaren funtzio partaideak bezala. Adibidez, `AnsiString` klaseak `c_str()` funtzio-partaidea du AnsiString C++ Builder-eko katea C hizkuntzako karaktere-kate bihurtzeko:

```
AnsiString katea1="Kaixo, zer moduz!";
char katea2[10];
...
katea2= katea1.c_str();
```

Datu mota baten edo aldagai baten byte kopurua jakiteko, `stdio.h` fitxategian definitua dagoen `sizeof()` eragilea erabiltzen da:

```
#include<stdio.h>
struct puntu
{
 char x;
 char y;
};
int main(void)
{
 int tam_doub,tam_a, puntu_tam;
 double a;


 puntu_tam=sizeof(double);
 a_tam=sizeof(a);
 puntu_tam=sizeof(struct puntu);
..
}
```

Datu mota beharpena aldagaiekin ez ezik erakusleekin ere erabiltzen da, eta funtzioetan ere bai, sarrera-parametroetan nahiz itzulitako balioetan. Ikus dezagun sarrera-parametrotzat `void` motarako erakusleak dituzten funtzioak: memoria esleipen dinamikodun kasua,

```
void *malloc(int bytetam);
void *calloc(int itemkop,int itemtam);
void free(void *erak);
```

```
int main(void)
{
 double *erak;
 erak=(double *)calloc(100,sizeof(double));
 if(erak==NULL)
  {
   puts("memoria-esleipen hutsegitea");
   return 1;
  }
 ...
 free(erak);
 return 0;
}

// -- DATU MOTA EZBERDINEN TAMAINA ETA DOITASUNA --
long double --> 10Byte(80bit),doitasun hedatua (double Visual C++en)
double -------> 8Byte(64bit), doitasun bikoitza (16 zifra edo digitu)
float --------> 4Byte(32bit), doitasun sinplea (8 digitu)
```

5.13 C hizkuntzako datu moten tamainak makinaren arabera

Mota baten tamaina bitetan ezberdina izan daiteke makina batean edo beste batean. Horrela, char mota beti da 8 bitekoa, eta int mota, berriz, µP-aren hitzaren bit kopuruko tamainakoa da beti*. Horrenbestez, long beti da short-en zabalera bikoizduna, non, makinaren arabera, `int`-ek `short`-ekin (16 biteko µP-a) edo `long`-ekin (32 biteko µP-a) bat egiten baitu:

mota	***16 biteko µP-a***	***32 biteko µP-a***
`short`	16bit	16bit
`int`	**16bit**	**32bit**
`long`	32bit	32bit

Halere, 8 biteko µC-etan, `int` mota 16 bitekoa izanik, ez du bat egiten makinaren datu-busarekin.

Moten definizioa `typedef`-ekin

```
#define uchar unsigned char // hau makro bat da: lehenengo karaktere-
                            // -katea bigarrenarekin ordezkatzen du
#define PI 3.1416

typedef unsigned char uchar;
typedef struct punto
{
 char x;
 char y;
}PUNTU;
```

edo bestea

```
typedef struct //egituraren izenik ez dago jarri beharrik
{
 char x;
 char y;
```

```
}puntu;

typedef char KATEA[100];
typedef MATRIZEA[4][4];

int main(void)
{
 KATEA cad;   // 100 karakteredun kate bat sortzen da
 MATRIZ mat1; // 4x4 doubledun 2D matrizea sortzen da
 PUNTU pto1;  // puntu kartesiarra
 puntu pto2;  // beste puntu kartesiar bat
...
}
```

6. PROGRAMAZIO MOTAK

6.1 Programazio lineala

Lehenengo programazio-hizkuntzak ataza soil samarrak egiteko garatu ziren eta programa haietako gehienak oso laburrak ziren, iturburu-kodea 100 lerro baino txikiagoa zutenak, hain zuzen ere.

Makinen potentzia handitu ahala, programen potentzia eta konplexutasuna ere handitu egin ziren. Lehenengo programazio-hizkuntzak (BASIC, COBOL eta FORTRANen lehen bertsioak) ez ziren egokiak programa horientzat, ez baitzuten erraztasunik ematen programek jada bazuten kodea berriro erabiltzeko. Horrela, geroz eta gehiago, kodea bikoiztu edo hirukoiztu egiten zen.

Gainera, programak sekuentzia logikoetan egikaritzen ziren, eta haien logika ulertzeko zaila egiten zen. Programen kontrola zaila zen eta etengabeko jauziak egiten ziren aipatutako programan zehar.

Are gehiago oraindik, aldagai guztiak orokorrak ziren; programaren edozein zatik alda zitzakeen, eta programa beraren osotasunezko emaitza bera ere aldatzera iritsi.

6.2 Moduluzko programazioa

Arazo horiei emandako lehenengo irtenbidea programa handiak osagai edo modulu txikiagoetan zatitzea izan zen. Modu horretan, programatzaileak independenteki eraiki zitezkeen, eta, azkenean, elkarrekin artean konbinatu sistema osoa eratzeko. Estrategia orokor hori *moduluzko programazio* izenez ezagutzen da.

Moduluzko programazioa egiteko euskarririk txikienari azpirrutina, prozedura edo funtzio deitu ohi zaio. Edozein modutan ere, oso garrantzitsua da programatzaileek programa bererako funtzioak egiteari ekin aurretik irizpide batzuk diziplina handiz ezartzea.

6.3 Programazio egituratua

Moduluzko programazioak, komunikazioaz kanpoko beste funtzio batzuk eragin gabe, beren artean komunika daitezkeen funtzioak dituenean (aldagai orokorrak desagertu egiten dira eta arlo edo irispen txikiagoak eta kontrolagarriagoak dituzten sarrerako parametroek eta aldagai lokalek ordezkatzen dituzte) funtzio horiek atzipen pribatua nahiz orokorrekoa dute memorian gordetako datuetara iristeko, eta hautapen-sententziak ere badituzte (adarkatzea: `if-else`,`switch case`) errepikapenezkoez gainera (errepikapena: `for`,`while`); orduan, programazio egituratua den hizkuntza dela esaten da.

Programazio egituratuarekin agertzen den kontzeptu oso garrantzitsua da *abstrakzioa*. Barruko xehetasunez kezkatu gabe zerbait aztertzeko gaitasun moduan defini daiteke abstrakzioa. Programa egituratu batean, nahikoa da jakitea funtzio jakin batek eginkizun zehatz bat egiten duela: funtzioa fidagarria den eta nola deitu behar zaion baldin badakigu, ez da garrantzitsua ataza nola egin den jakitea. Erabili egin daiteke barnea ezagutu gabe. Horri abstrakzio funtzional deitzen zaio, eta programazio egituratuaren muina da. *Adibideak:*

liburutegi-funtzioak, beste programatzaile batzuk egindako funtzioak; jakin behar diren gauza bakarrak hauek dira: zer egiten duten eta nola erabiltzen diren. barnea nola eginda dagoen kezka-iturri izan gabe.

Programazio egituratuaren desabantaila bat zera da, software baten programatzaileen arteko komunikazioak oso ona izan behar duela beren arteko edonork une jakin batean egin dezakeen edozein aldaketaren berri beste guztiek ere jakin dezaten. Aurreikus daitekeen moduan, programazio talde bateko partaideen arteko komunikazio-arazoek akats larriak eragin ditzakete. Akatsak ugaritu eta larriagotu egiten dira programaren tamaina handitzen eta konplexuago bihurtzen doan heinean.

6.4 Objektuei Zuzendutako Programazioa (OZP)

Objektuei Zuzendutako Programazioa (OZP) programen eraikuntzarako funtsezko bloke moduan objektuak erabiltzen dituen programazio-teknika edo programazio-estilo gisa defini daiteke.

Datu-abstrakzioa datuei, hizkuntzak eusten dituen datu-moten aldagaiei, aplikatutako abstrakzio-definizioa da. Datu-abstrakzioarekin, datu-moten aldagaiak erabil daitezke, haien inplementazio-xehetasunen berri jakiteko kezkarik izan gabe. Adibidez, koma higikor (`float`) motako gisa erazagututako aldagai batekin eragiketak egitean eta/edo erabiltzean, guk ez dugu zertan jakin mantisari zenbat bit dagozkion eta berretzaileari zenbat; garrantzitsuena, koma higikor (`float`) aldagai horrek ondo funtzionatzea da.

Datu-abstrakzioa ia programazio-hizkuntza guztietan egiten da. Datu-egiturak eta datu motak abstrakzio-adibide bat baino ez dira. Funtzioak eta prozedurak dira abstrakzioaren beste adibide bat.

Duela gutxi, beren berariazko datu-abstrakzio motak onartzen dituzten programazio-hizkuntzak sortu dira (Datu Mota Abstraktua edo DMA, Abstract Data Type edo ADT) Pascal, Ada, C++ moduan.

Datu mota abstraktua (ADT) programatzaileak datu horiekin egin daitezkeen eragiketa multzoekin batera definitutako datu mota bat da. Abstraktu izena ematen zaie C hizkuntzan definitutako oinarrizko edo funtsezko datuetatik —`char`,`int`,`float` eta horien antzekoetatik— bereizteko.

C hizkuntzan, `typedef` eta `struct` gako-hitzak erabiliz, datu mota abstraktuak defini daitezke eta eragiketen inplementazioa ere egin daiteke funtzio multzo batekin. C++ programazio-hizkuntzak erraztasun handia du ADTak (datu mota abstraktuak) definitzeko eta erabiltzeko.

Adibidez, `define` edota `typedef` erabiltzeak `unsigned char` datu motaren ordez `UCHAR` erabiltzen uzten du:

```
#define UCHAR unsigned char
typedef unsigned char UCHAR;
```

Horrenbestez, datu mota berri bat sortzeko, `typedef` erabil genezake, adibidez:

```
typedef struct puntu /* 'puntu' saltatu daiteke */
 {
   int x;
   int y;
 }PUNTU;
```

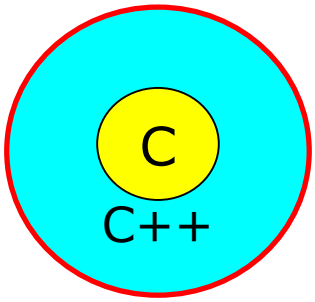
horrela, orain, puntu datu motako aldagaiak erazagut daitezke:

```
PUNTU p1, p2;
```

Objektuak, berez, datu mota abstraktuak bezalakoxeak dira. Erabiltzaileak definitutako datu-motetan bezalaxe, objektu bat datu sorta bat da, datu horietan eragiteko erabilitako funtzio elkartuekin batera. Alabaina, objektuen benetako indarra eusten dieten propietateetan dago: *herentzia, kapsularatzea* eta *polimorfismoa*, *objektua, klase, metodo* eta *mezu* kontzeptuekin batera.

6.5 C++ programazio-hizkuntzarako sarrera

C++ programazio hizkuntza Bjarne Stroustrup-ek sortu zuen 1980ko hamarkadan, Objektuei Zuzenduriko Programazioan erabiltzeko asmotan. C hizkuntzarekin gertatu zen bezala, honek ere estandarizazio-prozesu bat jasan zuen ANSI elkartearen bidez. Elkarte hartan sortu zen **ANSI C++** 1990eko hamarkadaren azken aldera . Hizkuntza berri hori C programazio-hizkuntza egituratuan oinarritua dago, eta, beraz, haren hedapen bat da.



6-1 irudia. C++ hizkuntza C-ren hedapen gisa

6.5.1 C hizkuntzaren hedapenak objektuei zuzendutako programazioarekin (OZP) erlaziorik izan gabe

Aipamenak (iruzkinak)

Iturburu-fitxategiaren barruan, hiru aipamen mota onartzen ditu:

```
1. /* C-n bezala */
2. // beste modu berri bat
3. #if 0
   testu
   handientzat
   #endif
```

Funtzioen prototipoak

C hizkuntzan aholkagarriak dira, eta C++ hizkuntzan, berriz, derrigorrezkoak. Prototipoa erabili nahi ez bada, funtzioa deitua izan aurretik definitzeko aukera dago.

Casting-ak edo mota-bihurketak

Lehendabizi esan beharra dago C++ hizkuntzan derrigorrezkoak direla mota zehatzagorantz jotzeko. Adibidez, `void`-etik `char`-era igarotzeko, casting-a egin behar da, baina ez alderantziz (`char`-etik `int`-era adibidez).

Bestalde, C++ hizkuntzak casting-a egiteko modu berri bat onartzen uzten du, bihurtu behar den motaren ordez, eragigaia edo aldagaia parentesi artean jarriz:

```
char a, b;
int d;

a=(char) a;  //modu zaharra
b=char (b);  //modu berria
```

Egiturak

Erabiltzaileak sortutako datu mota edo egitura mota berriekin, C++ hizkuntzak `struct` gako-hitza ez erabiltzeko aukera ematen du egitura mota baten aldagaiak erazagutzean, baina C hizkuntzan bezalaxe erabiltzeko aukera ere uzten du:

```
struct puntua
{
 char x;
 char y;
};
puntua Ptua1; // edo baita, struct puntua Ptua1; ere
```

Arau hori bera erabiltzen da bildurentzat ere.

Aldagaien erazagupena

1. C hizkuntzan, aldagai lokal bat beste aldagai orokor bat bezala izendatzen bada, eta, mota berdinekoa izanik, funtzioak aldagaia atzitzen duenean, lokala da nagusi, hau da, atzipena aldagai lokalera egiten du, eta ezin dezake orokorrera egin.

   C++n, aldagai orokorra atzitzeko beharra edo kasua gertatzen bada, :: aurrizkia jarri behar zaio aldagai orokorrari. Halere, beste izen bat erabiltzea litzateke hoberena aldagai horrentzat.

   ```
   ::aldagaia
   ```

2. Borland® C++ -en 4.5 bertsiotik aurrera, egitura errepikakorren barruan aldagai lokalak sortzeari uzten zaio, eta, beraz, egitura barnean bakarrik dute eragina.

   .
   .

```
for(int i=0;i<10;i++)
      sent1;
for(int i=20;i>10;i--)
      sent2;
```

Kasu honetan, lehenengo for begiztako i aldagaia eta bigarrenarena ez da bera: aldagai bakoitza begiztaren hasieran hasten da existitzen, eta bere bukaeran uzten dio existitzeari. 4.5 bertsioaren aurrekoetan, horrela erazaguturiko aldagaiek puntu horretatik fitxategiaren azkeneraino luzatzen zuten aldagaien existentzia.

Balio bidezko parametro-pasatzea

5.6.3 atalean ikusi genuen C hizkuntzan funtzioei parametroak pasatzeko bi modu zeudela: balio bidezko parametro-pasatzea eta erreferentzia bidezko parametro-pasatzea. Lehenengoa errazagoa zen arren ulertzeko eta erabiltzeko; bigarrena, erakusleak medio, eraginkorragoa zen, egikaritzapen abiadura handiagoa eta memoria aurreztea lortzen baitziren.

C++ hizkuntzak hirugarren parametro-pasatze modu bat gehiago du, C-ko biez gain. C-ko bigarren moduari *erakusleen bidezko parametro-pasatzea* edo *helbide bidezko parametro-pasatzea* deitzen dio, eta hirugarrenak *erreferentzia bidezko parametro pasatzea* izena hartzen du.

Izan ere, C++ko *erreferentzia bidezko parametro-pasatzea helbide bidezko parametro-pasatzea* bezalakoa da. Ezberdintasuna da ez duela `*` eragilea erabiltzen (abantaila da hori), eta, gainera, iturburu-kodea irakurterrazagoa egiten duela. Baina, izatez, *helbide bidezko parametro-pasatzeak* deitzen duen funtzioaren aldagaien (argumentuen) erakusleak erabiltzen ditu dei egindako funtzioan, eta horiek erakusleen bidez aldatzen dira. Horrela, azken puntu hori desabantaila bat izan daiteke, programatzaileak ideia hori argi ez badauka: nahi gabeko aldaketak sar litezke erakutsitako aldagaietan (argumentuetan).

Erakusleen edo helbide bidezko parametro-pasatzea:

```
void Datu_truke(int *a, int *b)
{
 int denb;

 denb=*a;
 *a=*b;
 *b=denb;
}
void main(void)
{
 int x=2, y=7;

 Datu_truke(&x,&y);
}
```

Erreferentzia bidezko parametro-pasatzea:

```
void Datu_truke(int& a, int& b)
{
 int denb;
```

```
 denb=a;
 a=b;
 b=denb;
}
void main(void)
{
 int x=2, y=7;

 Datu_truke(x,y);
}
```

Overloading: funtzioen gainzama

C-n debekatua dago erabiltzea. Izen berarekin funtzio bat baino gehiago izendatzea da, betiere, sarrera-parametroren bat edo itzulitako balioa datu mota ezberdina den kasuan: konpiladorea arduratzen da ezberdintasuna aurkitzeaz.

6.5.2 C hizkuntzaren hedapenak objektuei zuzendutako programazioarekiko (OZP)

Sarrera

Ohiko programazioan, programak bi osagaitan zatitzen dira: prozeduretan eta datuetan. *Prozedura* bakoitzak kaxa beltz baten moduan jokatzen du; ataza jakin bat betetzen duen osagaia da, zenbaki multzo bat bihurtu edo (Windows®) leiho bat bistaratu adibidez. Kaxa beltzak behar den bezala zatitzen badira, kode bat idatz daiteke bakoitzarentzat, beste kaxa batzuek beren barnean zer egiten duten kontuan hartu gabe. Metodo hau erabiltzearen abantaila nagusia moduluzko eta garraiagarri diren programak (moduluzko programazioa, programazio egituratua) garatzen ematen duen laguntza da. Datuei dagokienez, erabilitako datu-egiturak orokorrak, lokalak eta sarrera-parametroenak dira. Orduan, programa-kodea prozeduretan trinkotzen da, baina datuak, aldiz, prozeduretatik bereizita tratatzen dira.

Objektuei Zuzendutako Programazio-metodoak (OZP) erabiltzen direnean, prozedurak eta datuak dituzten osagaietan zatitzen da programa. *Objektu* bat datuak eta datu horietan eragiten duten funtzioak (prozedurak) dituen unitatea da. Objektu baten elementuak *partaide* gisa ezagutzen dira; objektuetan eragiten duten funtzioei *metodo partaide* edo *funtzio partaide* deritze, eta datuei *datu partaide*.

C++ hizkuntzan, programa bat objektuz osatuta dago. Programa baten objektuak beren artean *mezuak pasatuz* komunikatzen dira (objektu hartzaileak egikaritu behar ditu ekintza horiek).

Kasu berarentzat egindako ohiko programazioaren eta objektuei zuzendutako programazioaren adibidea: pantailaren leihoa izeneko objektua lau norabideetan higitu.

Ohiko programazio-metodoa: higidura bakoitzarentzako prozedura independente bat definitu ahal izango litzateke. Prozedura horiek kaxa beltz moduan jardungo lukete, eta lau eragiketak zatitzeko modurik onena izango litzateke; baina leiho objektua definituko luketen datuak prozedurekiko independenteak izango lirateke (hori eragozpena da).

Objektuei zuzendutako programazio-metodoa: metodo honekin, pakete berean konbinatzen dira datuak eta prozedurak. Lehenengo urratsa leiho izeneko objektua sortzea da, eta hark bere barnean ditu bere higidura definitzeko prozedurak. Leihoak "higitu ezkerrera" agindua jasotzen duenean, dagokion prozedura egikaritzen da.

Adibidea: helburu bera lortu, puntu kartesiar bat hasieratzea, ohiko programazio-metodoa eta objektuei zuzendutako programazio-metodoa erabiliz.

```
struc puntu
{
 int x;
 int y;
};
struct puntu EginPuntu(int a,int b)
{
  struct puntu emai;

  emai.x=a;
  emai.y=b;
  return emai;
 }
void main(void)
{
 int xp=3,yp=4;
 struct puntu P1;

 P1=EginPuntu(xp,yp);
}

class puntu
{
 int x;
 int y;
public:
 void Hasi_puntu(int,int);
};
void puntu::Init_puntu(int a,int b)
{
 x=a;
 y=b;
}
void main void(void)
{
 int xp=3,yp=4;
 puntu P1;

 P1.Hasi_puntu(xp,yp);
}
```

Objektuek beren ezaugarri bereizi batzuei eusten diete beren baitan:

- *Klaseak* izeneko talde batzuetan biltzen dira.
- Beren egungo egoera definitzen duten barne-datuak (*propietateak*) dituzte.
- *Datu-ezkutaketari* (*atzipen pribatua*) eusten diote.
- Beste objektu batzuen propietateak *herentzian* jaso ditzakete.
- Beste objektu batzuekin komunika daitezke, mezuak pasatuz edo igorriz.

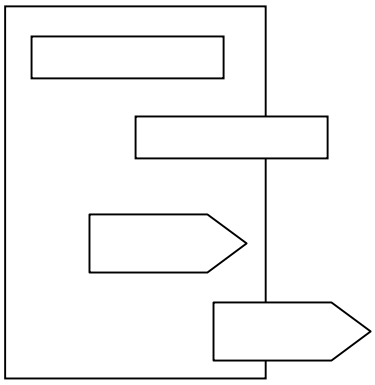
- Beren portaera definitzen duten *metodoak* (*funtzio partaideak*) dituzte.

Objektu-definizioa

Objektu bat datuak eta datu horietan eragiten duten funtzioak dituen unitatea da. Datuei datu partaide deitzen zaie, eta partaide diren funtzioei funtzio, funtzio partaide edo metodo partaide.

Datuak eta funtzioak objektu izeneko eraketa bakarrean kapsulatzen (trinkotzen) eta barneratzen dira. Datuetako batzuk objektu horren funtzioek bakarrik ikusteko moduan egon daitezke, edota, bestela, funtzio horiek edota beste objektu batzuek ere bai (beren funtzioek alegia). Gauza bera gertatzen da objektu baten funtzioekin ere: objektuaren barrutik bakarrik atzitzeko modukoak izan daitezke, edota, agian, baita beste objekturen batetik ere (kanpotik alegia). Elementu horiei publiko eta pribatu deitzen zaie, hurrenez hurren.

Laukizuzenaren perimetroak objektu baten barruko eta kanpoko aldeen arteko muga adierazten du. Objektuaren barruko aldean, aldagai lokalak (eremu partaideak) eta funtzioak daude. Erabat laukiaren barruan dauden eremu partaideak eta funtzioak ezkutukoak (pribatuak) dira kanpokoentzat; beraz, horrek esan nahi du ezaugarri horiek *kapsulatuta edo trinkotuta* daudela. Laukitik kanpora irten eta hedatzen diren eremuak eta funtzioak, kanpotik atzitzeko modukoak dira, eta *interfaze* moduan jokatzen dute. *Ezaugarri partaideak* (eremuak eta funtzioak) objektuaren interfazearen bitartez bakarrik atzitu daitezke.



6-2 irudia. Objektu baten partaideen adierazpen grafikoa

Objektuei zuzendutako programa

Objektuei zuzendutako programa bat egikaritzen denean, 3 gauza gertatzen dira:

1. Behar direnean objektuak sortzen dira.
2. Mezuak objektu batzuetatik bidaltzen dira, eta beste batzuetan hartzen, programak informazioa prozesatu edota erabiltzailearen sarrerari erantzun ahala.
3. Objektuak jada beharrezkoak ez direnean, ezabatu egiten dira, eta haiek betetako memoria berreskuratu egiten da.

Ondoretasuna (Herentzia)

Objektuen eta klaseen ezaugarri garrantzitsu bat ondoretasun edo herentzia da. Klaseak beste klase batzuk oinarritzat hartuz, antolakuntza hierarkiko baten arabera, eraikitzea ahalbidetzen duen propietateari deritzo ondoretasuna. Horrela, ondoretasuna noranzko bakarrekoa da, klase batek bere superklaseetatik jarauntsi baitezake, baina sekula ere ez bere azpiklaseetatik. Oinarrizko bi ondoretasun mota existitzen dira:

1. Ondoretasun soila (sinplea): klase batek superklase batetik bakarrik jarausten duenean (aldiz, superklase batek hainbat azpiklase izan ditzake)
2. Ondoretasun anizkoitza: klase batek hainbat superklasetatik jarausten duenean.

Objektuei zuzendutako hizkuntza askok ez du ondoretasun anizkoitza, baina bai C++ek.

Superklase batean oinarritzen diren klaseek alde batetik ezaugarri komunak dituzte, eta, bestetik, bereak. Gauza bera gertatzen da azpiklaseen eta klaseen artean.

Jatorrizko klasea: oinarrizko klase ere esaten zaio, eta haren seme-alabei, klase eratorri.

Polimorfismoa

OZParen beste ezaugarri garrantzitsu bat polimorfismoa da. Objektuek agindu bati era desberdinez erantzuteko ahalmena da polimorfismoa. Ezaugarri hori duten objektuei objektu polimorfiko deritze. Polimorfismoa funtzioen gainzama edota alegiazko funtzioak erabiliz gauzatu daiteke.

Lehenengo moduari dagokionez, pentsa dezagun sarrera-parametro batean ezberdintzen diren eta berdin deitzen diren bi funtziorekin (gainzama): bi eragiketa desberdin egiten dituzte, eta izen bera dute.

Berrerabilera

Klase bat behin sortua, araztua eta erabilia izanez gero, beste programatzaile batzuen artean zabaldua izan daiteke, haien programetan erabili ahal izan dezaten. Propietate horri berrerabilera deritzo, eta hizkuntza egituratu batean funtzio-biblioteka bat barneratzen den kasuaren antzekoa da.

OZPan, berrerabilera ondoretasunari esker existitzen da. Horrela, programatzaile batek, klase bat hartu eta adatu gabe, ezaugarri eta posibilitate berriak gehi diezazkioke. Klase berri horrek zaharraren ezaugarriak jarausten ditu, baina berriak ere baditu.

Izatez, C++ programazio-hizkuntza C hizkuntzaren hedakuntza edo zabalkuntza besterik ez da, baina Objektuei Zuzendutako Programazio gisa.

7. ORDENAGAILU PERTSONALAK

7.1 iNTEL®-en 80x86 µP familia

Kapitulu honetan, Intel®-en 80x86 µP (mikroprozesadore) familiaren ikasketa laburra egingo dugu. 80x86 familiaren ezaugarririk garrantzitsuena *software-bateragarritasun gorakorra* da, eta horrek hau esan nahi du: prozesadorerik berrienak gai dira familiako aurreko mikroprozesadoreek korritzen edo egikaritzen zuten kode osoa korritzeko edo egikaritzeko.

Ordenagailu Pertsonal gehienak (PCak), 80x86 familia horretako µP-etan edota familiako beste bertsio batzuetan (µP-en beste fabrikatzaile batzuenetan) oinarrituta daude.

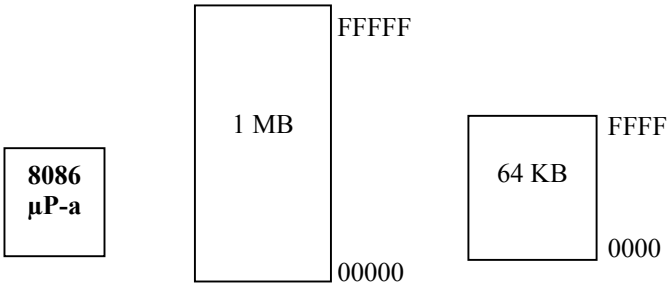
7.1.1 Intel-en 8086 µP-a

Sarrera

1978an irten zen merkatura Intelek diseinatutako 16 biteko lehenengo µP-a. Familiako lehenbizikoa izan zen, eta 29.000 transistorez osatuta zegoen. 16 biteko 14 barne-erregistro, 20 hariko (A0..A19) helbide-busa eta 16 hariko (D0..D15) datu-busa ditu.

8086 µP-a gai da ondorengo hauek helbideratzeko:

* 2^{20}=1.048.576 zortzikote (byte)=1 megabyte memoria-esparruan
* 2^{16}=65536 zortzikote (byte)=64 kilobyte S/I (Sarrera/Irteera; I/O) esparruan, non A16..A19 helbide-busak behe-mailako egoera logikoan edo "0-an" baitaude (ez dira erabiltzen)



7-1 irudia. 8086 µP-aren memoria-esparrua eta I/O esparrua

Atzipena memoria-esparrukoa edota, alderantziz, S/Ikoa (I/O) izatearen aldeko seinalea µP-aren M/-IO pina da. Pin hori "1"ean jartzen da memorian idazketa- edo irakurketa-eragiketaren bat egin behar denean. µP-ak memorian atzitzeko aginduren bat egikaritzen duenean gertatzen da hori.

Era berean, 8086 µP-ak S/I (I/O) esparruan atzitzeko aginduren bat egikaritzen duenean, M/-IO pina "0"an jartzen da.

S/I (I/O) esparruaren kokapenak edo lekutzeak S/I (I/O) portu izenez ezagutzen dira, eta ez dira nahasi behar PCen serieko eta paraleloko komunikazio-portuekin.

(RAM) datu-memorien gailuak eta (ROM, EPROM) programa-memoriaren edo kode-memoriaren gailuak bertan konektatzeko erabiltzen da memoria-esparrua.

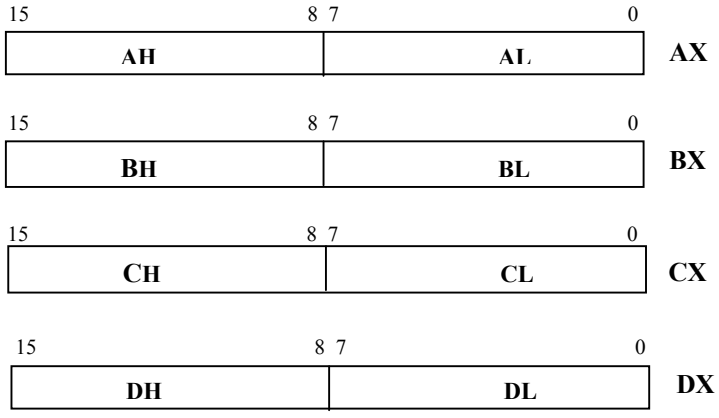
Aldiz, S/I (I/O) esparrua, S/I-ren gailu periferikoak han mapatzeko (kokatzeko edo konektatzeko) erabiltzen da, eta, horrela, µP-ak gailu periferikoren bati esleitutako portuetan idaztean eta/edo irakurtzean, aukera ematen du aldi berean aipatutako gailu periferikora edo periferikotik datuak igortzeko edo hartzeko.

Memoria-esparruan atzitzeko aginduek modu desberdinak dituzte beren helbideetan atzitzeko, eta oinarrizko aginduek baino askoz ere malgutasun handiagoa dute S/I (I/O) esparruan atzitzeko. Arrazoi hori dela eta, batzuetan memoria-esparruan S/I (I/O) gailuak mapatzera jotzen da, memoria-atzipenak egiteko agindu multzoak eskainitako malgutasuna aprobetxatzeko helburuarekin.

Barne-erregistroak eta memoria-helbideratzea

Hasieran esan den bezala, 8086 µP-ak 16 biteko 14 barne-erregistroko multzoa du, eta horiek bost taldetan sailka daiteke. Esan beharra dago 80x86 familiako µP berriago guztiek oinarri moduan erregistro multzo hori dutela, kasu bakoitzean egindako zabalkuntzekin edo hedapenekin.

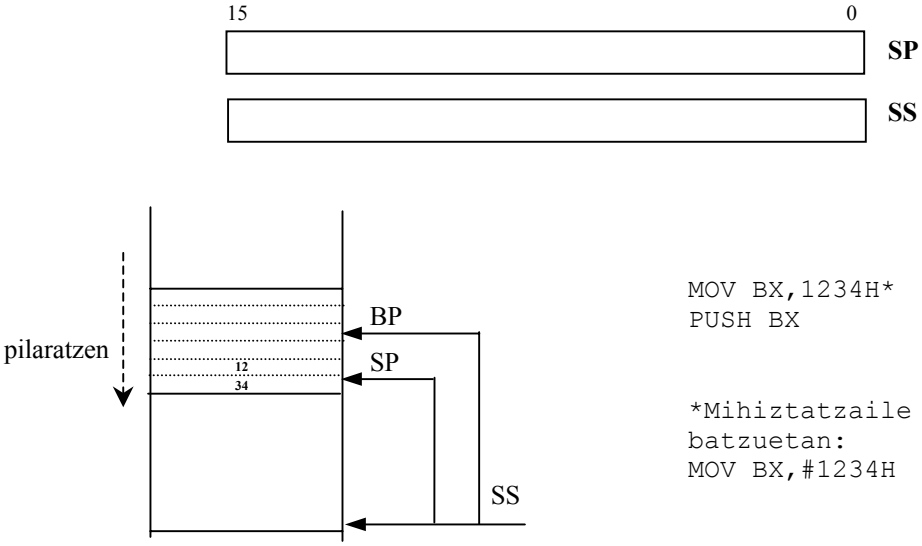
a) **Xede orokorreko erregistroak**: (AX) Metagailua, (BX) Oinarria, (CX) Kontagailua eta (DX) Datuak.
   - 16 biteko erregistro moduan erabil daitezke, edota 8 biteko moduan (goiko aldean: AH,... edo behekoan: AL,...),
   - bakoitzak egiteko eginkizun bat du izendatuta: datu-transferentziak, aritmetika-eragiketak eta logika-eragiketak, begizta iteratiboak, etab.
   - programatzaileak xede orokorreko erregistro moduan erabil ditzake.



7-2 irudia

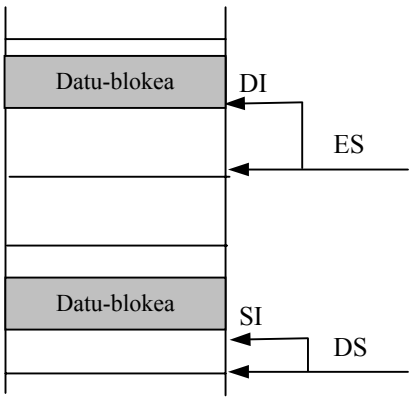
b) **Erakusle-erregistroak:** Pilako erakuslea (SP, Stack Pointer) eta Oinarri-erakuslea (BP, Base Pointer)

- Xede orokorreko erregistro gisa erabil daitezke, baina, normalean, pila-segmentuen barruan memoria-kokapenak erakusteko erabiltzen dira.
- Pila-memoria (pila), etenei arreta ematean erregistroen edukiak salbatzeko (gordetzeko) erabiltzen da. SP-k goiburukoa edo pilaren goiko (tontor) aldea nahiz gailurra erakusten du, eta BP-k aldiz, pilan gordeta dagoen edozein datu atzitzeko aukera eskaintzen du.
- Pilaren memoria LIFO (Last In First Out) erakoa da, baina, 8051 familiako µC-etan ez bezala, kasu honetan pila modu beherakorrean handitzen da, hots, beherantz.



7-3 irudia

c) **Indize-erregistroak:** (SI, Source Index) Iturburu-indizea eta (DI, Destination Index) Helburu-indizea.
   - Hauek ere xede orokorreko erregistro gisa erabil daitezke, baina, normalean, datu-segmentuen barruan memoria-kokapenak erakusteko erabiltzen dira. Horrela, datu-blokeen higidurak egin ahal izaten dituzte memoria-eremu batetik bestera.
   - SI-k higitu behar diren datu-blokeen hasierako helbidea erakusten du, eta DI-k, berriz, datu-blokea higitu behar duen esparruaren hasierako helbidea.



7-4 irudia

d) **Segmentu-erregistroak:** CS (Code Segment, Kode Segmentua), DS (Data Segment, Datu Segmentua), SS (Stack Segment, Pila Segmentua) eta ES (Extra Segment, Aparteko Segmentua)

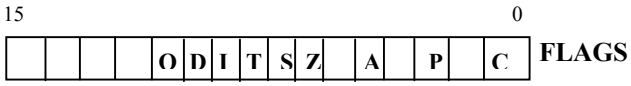
- Memoria-segmentu bereiziak definitzeko erabiltzen dira. Bakoitza 64 kilobytekoa da.
- CS-k programaren kodea bere baitan duen segmentua definitzen du, DS-k programaren datuak dituena, eta SS-k pilaren segmentua. ES-k, hala behar balitz, aparteko datuen segmentua definituko luke.

e) **Programaren Kontagailu-erregistroa edo Agindu-erakuslea:** *PC (Program Counter)* edo *IP (Instruction Pointer)*

- Erregistro garrantzitsu honek 8086 µC-ak kode-segmentuaren barruan egikaritu behar duen ondorengo aginduaren desplazamendu-helbidea du.

f) **Egoera-erregistroa (edo Flags, banderak), edo PUZaren (CPUa), Prozesatzeko Unitate Zentralaren, (µP) Egoera-erregistroa**

- Bere izenak adierazten duen moduan, programa egikaritzen ari den edozein unetan, bere barnean dituen 6 bandera (flag) biegonkorretan zehar µP-ak duen egoeraren berri ematen duen erregistroa da. Beste 3 banderek PUZaren (CPU) ezaugarri batzuk aldatzeko aukera ematen dute.



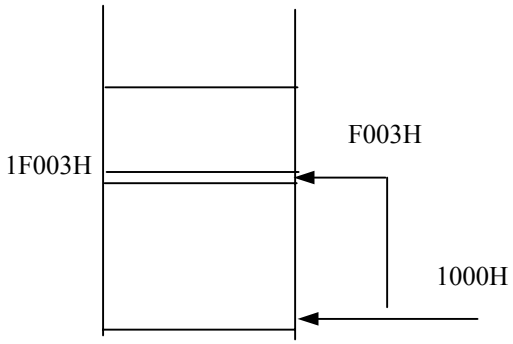
7-5 irudia

C : carry-bita
P: paritate bakoitiko bita
A: bururako flag lagungarria
Z: zero-bita; eragiketa aritmetikoaren emaitza hutsa bada, 1; bestela, 0
S: zeinu-bita
T*: tranpa (etendura), urratsez urratseko eta haustura-puntuetan oinarrituriko egikaritzea kontrolatzen du, 1 egoeran badago.
I*: hardware-etendura maskaragarria gaitzen du 1 egoeran egonez gero, edo ezgaitzen 0-n badago
D*: SI eta DI-ren gehikuntza 1-en egonez gero, edo gutxikuntza 0-n badago
O: overflow edo gainezkatze-egoera, 1 egoeran badago

Segmentu-helbidea (segmentua) eta desplazamendu -helbidea (offset) oinarritzat hartuta egiten da memoriaren helbideratzea (oinarri eta desplazamendu bidezko helbideratzea). Bestela esanda, memoriako helbide fisiko edo erreal bat definitzeko, behar-beharrezkoa da bi helbide erabiltzea: segmentuarena eta desplazamenduarena. Segmentu-helbideak 64 kilobyteko segmentua definitzen du eta desplazamenduarenak 64 kilobyte horietako benetako helbide erreala zein den definitzen du.

Hala bada, benetako helbide baten balioa zein den jakiteko, segmentuaren helbidea 10H-rekin biderkatzen da eta desplazamendu-helbidea gehitzen zaio. Eragiketak, normalean,

sistema hamaseitarrean egiten dira, eta, askotan, helbide erreala idazkera segmentatuan ematen da, hots, SEGMENT:OFFSET eran (adibidez: 1000H:F003H).



7-6 irudia

Etendura-sistema

Etendura-definizioa dokumentu honen 4.1 atalean emana dago.

80x86 μP-aren etendura-prozesua honako hiru urrats hauek adierazten dute:

1. STATUS, CS, PC erregistroetako edukiak pilan kargatzen dira, edota pilan gordetzen dira eta etendurak gaitzeko banderak ezgaitu egiten dira; kanpoko maskaragarria eta urratsez urratsekoa (desbiderapena, tranpa).
2. Etendura-bektorea PC eta CS erregistroetan kargatzen da, eta horrekin eskatutako etenduraren zerbitzu-azpirrutina egikaritzen da.
3. Pilatik berreskuratu eta PC, CS eta STATUS erregistroetan kargatzen dira erregistro horiek etendura gertatu aurretik zituzten edukiak.

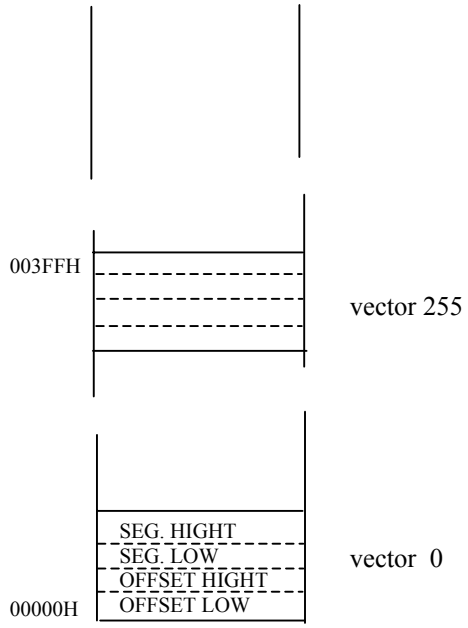
8086 mikroprozesadorearen etendura-egitura etendura-bektoreen taulan oinarritzen da. Taula horrek:

- Sistemaren memoriaren (memoria-esparruaren) lehenengo 1024 zortzikoteak (byteak) betetzen ditu.
- 256 etendura-bektore arte izan ditzake, 0tik 255era zenbatuta.
- Bektore bakoitzak 4 zortzikote (byte) betetzen ditu taulan: horietako 2 zortzikote desplazamendu-helbidearentzako eta beste 2 zortzikote segmentu-helbidearentzako.

Lehenengo 32 bektoreak iNTELek gordeta edo erreserbatuta ditu.

PCaren (Ordenagailu Pertsonalaren) kasuan:

- BIOSen (Sarrera eta Irteeren Oinarrizko Sistema) zerbitzu-errutinak PCaren memoriaren azken 64 kilobytetan grabatuta daude (goiko memoriaren barruan, goian)
- PCa etenduren erabileraren bidez kontrolatzen da: kanpoko periferikoek kanpoko etendurak eskatu eta eragiten dituzte, eta, aldi berean, han egikaritutako programek software-etendurak edo salbuespenak eragiten dituzte.



7-7 irudia

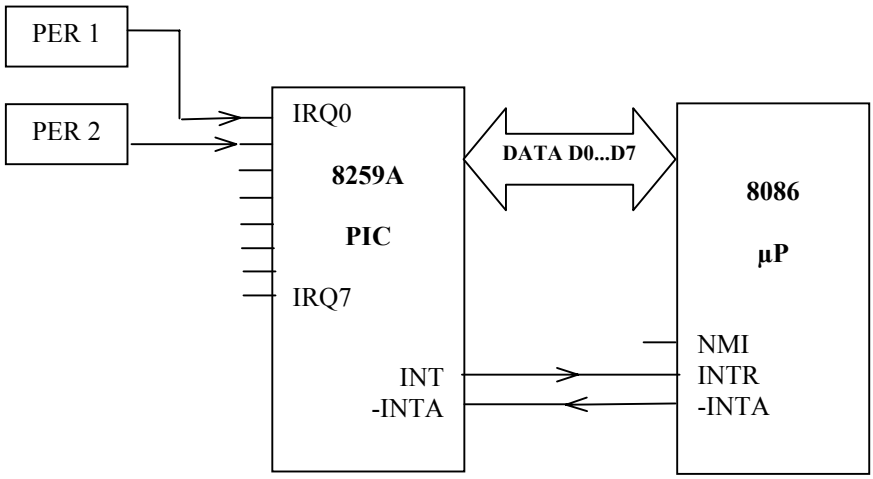
Hardware-etendura (edo kanpokoa)

Elementu periferiko batek µP-aren etendura-pin batean seinale elektriko bat egin (jarri) ondoren sortzen dena da. 8086 µP-ak hiru etendura-pin ditu: bi eskaerakoak (dituen bi hardwareen etendura motei dagozkienak) eta bat erantzutekoa (bietako baten irteerakoa).

1. *NMI: Maskaragarria ez den etendura*. NMI pinean igoera-saihets bat aplikatuz eskatzen da. Etendura hori ezin da maskaratu (ezgaitu). Eginkizun hauetarako erabiltzen da: paritate-akatsak atzemateko, larrialdietako geraldiak egiteko eta sisteman hornikuntza elektrikoaren etenak atzemateko (orduan PUZak (CPUak) pilaz babestutako memoria batean gordetzen ditu erabiltzen ari den erregistro guztiak, hots, une horretako egoera).

   Etendura honi elkartutako bektorearen zenbakia 2 da.

2. *Etendura maskaragarria (ezgaitu daitekeena).*
   INTR pinean goiko maila edota igoera-saihets bat jartzean (hori programagarria da) eskatzen da. Hori ezgaitzeko, "0"an jartzen da I flag-a (I=0). –INTA (INT Acknowledge, onartze-erantzuna) pina etendura eskatzen duen elementu periferikoari etendura jada eskatua izan dela adierazteko erabiltzen da.
   Etendura mota honetan, etendura bati baino gehiagori eman dakieke erantzuna bi pin hauek erabiliz. µP-ari hardware gehigarria konektatuta lortzen da hori. Merkatuan bada zirkuitu integratu bat funtzio hori burutzen duena, bere garaian Intel-ek diseinatua : 8259A txipa da, Etendura-kontrolatzaile Programagarria, eta horrek maskaratu daitezkeen 8 kanpo-etendura kontrolatzeko aukera ematen du. 8086 µP-ari gehienera 8 elementu periferiko konektatzeko balio izango luke horrek.



7-8 irudia

Funtzionamendua:

1. elementu periferiko batek etendura eskatzen du IRQx (x:0..7) etendura-eskaera pinean zehar. 8259A txipak 8086ari jakinaraziko dio INT→INTR bidez.
2. Etendura maskaragarria gaituta baldin badago, hots, I=1 bada, orduan 8086ak 2 beheko pultsuren bidez erantzungo dio –INTA pinean zehar: lehenak 8259A txipari 8086ak etendura onartu duela adieraziko dio, eta bigarrenak, berriz, beste hau adieraziko dio: 8086ak datu-busaren beheko aldean periferikoak eskatutako etendura-bektorearen zenbakia irakur dezala, bektore hori 8259A txipak emana baita.
3. 8086 µP-ak 4z biderkatzen du zenbaki hori, eta, horrela, etendura horren lehenengo zortzikotea (bytea) zein helbidetan dagoen kalkulatzen du. Segidan, CS:PCan bektorea kargatzen du, eta, horrela, etenduraren zerbitzu-errutina egikaritzen du.

Software-etendura

Etendura-agindu bat egikaritzean sortzen dena da. 8086 µP-aren etendura-jokoan hiru etendura-agindu daude.

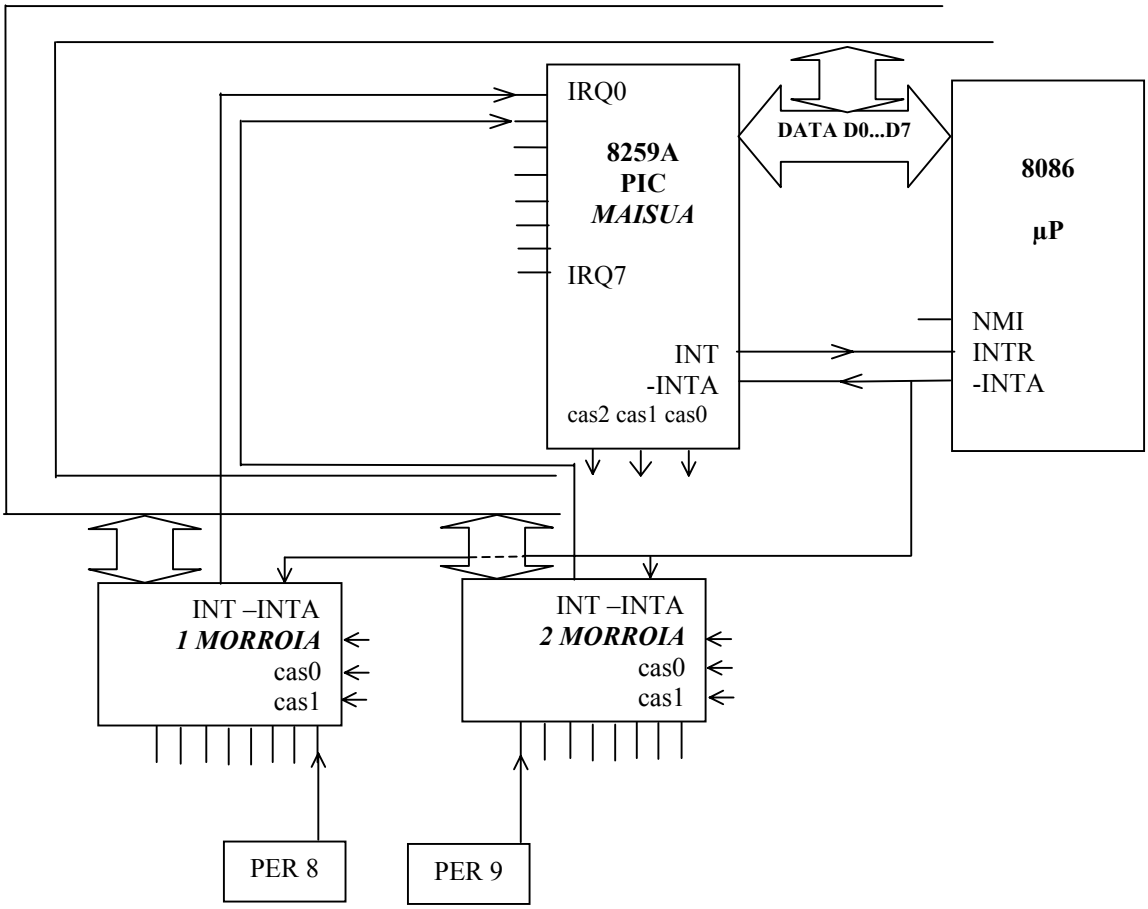
a) `INT n` agindua

Edozein etendura eskatzeko erabiltzen da, bere bektorea n-rekin izendatua dagoelarik bektore-taulan; beraz, edozein motatako etendura egikari daiteke, bai hardware motakoa bada, eta baita software motakoa bada ere.
Agindu hau 2 zortzikotekoa (bytekoa) da: lehenengoak aginduaren kodea du eta bigarrenak bektore-zenbakia.

b) `INT 3` agindua

Bektore-taulan, 3 zenbakiarekin izendatua dagoen etendura eskatzeko erabiltzen da. Programetan haustura-puntuak edo *break point* deituak sartzeko erabiltzen da, *debugger* edo programa-araztailearekin araztuak izatean.

Zortzikote (byte) bakar bateko agindua da, eta aginduaren kodea da.



7-9 irudia

c) `INTO` agindua

Bektore-taulan 4 zenbakiarekin izendatua dagoen bektorearen etendura eskatzeko erabiltzen da. Gainezkatze-flaga (overflow flag) aktibatuta baldin badago egikaritzen da, hots, O=1 baldin bada.

7.1.2 8088 µP-a

Sarrera

1979 urtean, 80x86 familiako bigarren kidea agertu zen merkatuan, hots, 8088 µP. Prozesadore hau 28.000 transistorez osatua dago, eta, egia esan, 8 biteko kanpoko datu-busa duen 8086 bat besterik ez da, barneko datu-busa eta 14 erregistroak 16 bitekoak izan arren.

8086ak duen helbideratze-ahalmen berbera dauka, I/O esparruan nahiz memoriakoan ere, eta 14 barne-erregistrodun multzoa ere aurrekoarenaren berbera du.

Beraz, 16 biteko µP batez hitz egiten ari gara, kanpo-atzipenetarako 8 bitekoa izan duen arren. Horrek esan nahi du 8088a kanpoan 16 biteko hitz edo word baten irakurketa edo idazketa egitera doan bakoitzean ziklo bitan egin beharko duela (eta ez ziklo batean, 8086aren kasuan bezala). Ezaugarri hori dela eta, kristal-maiztasun berdina izanik, 8088a 8086a baino motelagoa da.

8088a eta 8086a:

- softwarez bateragarri, agindu multzo berbera erabiltzen dute. Beraz, µP batentzako eginak izan diren programak normaltasun osoz egikaritzen dira bestean.
- Hardwarez bateraezinak izanik, µP biak 40 pineko txipak izan arren, haien banaketa ezberdina da. Beraz, bietatik batean oinarrituriko sistema batean, ezingo genuke hori bestearekin ordezkatu.

1981. urtean merkaturatu zen IBM® etxearen lehen ordenagailua, hots, PC-XT®. Ordenagailu hark 8088 µPa zeraman CPUtzat, eta ez 8086a. Izan ere, orduan 16 biteko (datu-busa) gailuak 8 bitekoak baino garestiagoak eta urriagoak ziren.

Handik gutxira, COMPAQ® etxeak bere DESKPRO® eredua merkaturatu zuen, 8086 µPan oinarritua. Beraz, PC-XTa baino bizkorragoa zen kristal-maiztasun berberarentzat.

Memoria-helbideen eta S/I (I/O) helbideen deskodetzea

Helbide-deskodetzeak aukera ematen du gailu bat hautatzeko, µP-ak gailu horri esleitu zaizkion helbideetako bat sortzen duenean.

Helbideak deskodetzearekin, gailu jakin bat mapatu, ipini edo kokatzea lortzen da, memoria-esparruaren helbide-maila jakin baten barruan edota S/I-renean (I/O), µP-an oinarritutako sistema batean.

Adibidea: 2 kilobyteko EPROM 22716 (txipa) memoriaren esleipena FFF800H-FFFFFH helbide-bitartean.

Helbideak deskodetzeko erabili ohi den elementua honako hauetakoren bat izan daiteke: ate logikoen multzo bat, merkatuko deskodetzaileren bat edo logika programagarridun gailu bat (PAL).

S/I (I/O) helbideak deskodetzeko, hots, S/I erako portuak, memoria-helbideak deskodetzeko modu berean jokatzen da, baina, kasu honetan, 8088 µP-ak IO/-M pina "1" balioan jartzen du S/I (I/O) esparrua atzitueran.

8086/8088 µP-aren berrasieratzea

RESET pina "1"ean dagoenean, µP-ak berrasieratze-egoeran irauten du inolako agindurik egikaritu gabe, eta "0"ra itzultzean, STATUS, DS, ES, SS eta PC erregistroak, 0000Han berrasieratzen dira eta CSa FFFFHean; jarraian, 8086/8088 µP-a aginduak egikaritzen hasten da FFFF:0000 (CS:PC) helbidetik abiatuta, hau da, FFFF0H benetako helbide errealetik.

7.1.3 80188/80186 µC-ak

80186 eta 80188 µC-ak nukleotzat 8086 µP-a eta 8088 µP-a dituzten bi mikrokontrolatzaile dira, hurrenez hurren. µP-arekin batera barneratzen duten periferia berbera da bientzat, eta ondorengo atalez osatuta dago:

- Erloju-sorgailu bat, 8086/88 ereduko 8284A txipa ordezkatzen duena.
- Etenduren kontrolatzaile programagarri bat, 8259A txipa ordezkatzen duena: 5 eskaera-sarrera ditu (4 maskaragarriak eta NMI 1) eta kanpoko bi 8259A txip kontrola ditzake.
- 16 biteko hiru tenporizadore programagarri: T0 eta T1, barne edo kanpoko erloju-sorgailuz erabiliak.
- DMA (Zuzeneko Memoria-Atzipena; ZMA) unitate programagarria, bi kanal independenterekin eta programetatik, periferikoetatik nahiz T2ren irteeratik egindako eskaera ahalekin.
- Egokitze-seinaleen unitate programagarri bat. Memoria-gailuentzako 6 hariko eta S/I (I/O) gailuentzako 7 hariko helbide-deskodetzailea da.

DMA (Zuzeneko memoria-atzipena)

Periferiako elementu baten (S/I gailuaren) eta memoriaren artean edota bi memoriaren artean CPU bidez datuak transferitzea oso geldia izatera irits daiteke abiadura handiak behar diren egoeretan.

DMA teknikak aukera ematen du I/O esparruen eta memorien artean nahiz memorien artean datu-transferentziak abiadura handian egiteko, CPUak edo µP-ak esku hartu gabe.

Intel-en DMA kontrolatzaile programagarriak, 8237 txipak, lau kanal ditu, eta, horien bidez, lau elementu periferiko jasan ditzake zuzeneko memoria-atzipenak egiteko. 1,6 megabyte/s arteko abiadura lor daiteke, eta kanal bakoitzak 64 kilobyte transferi ditzake.

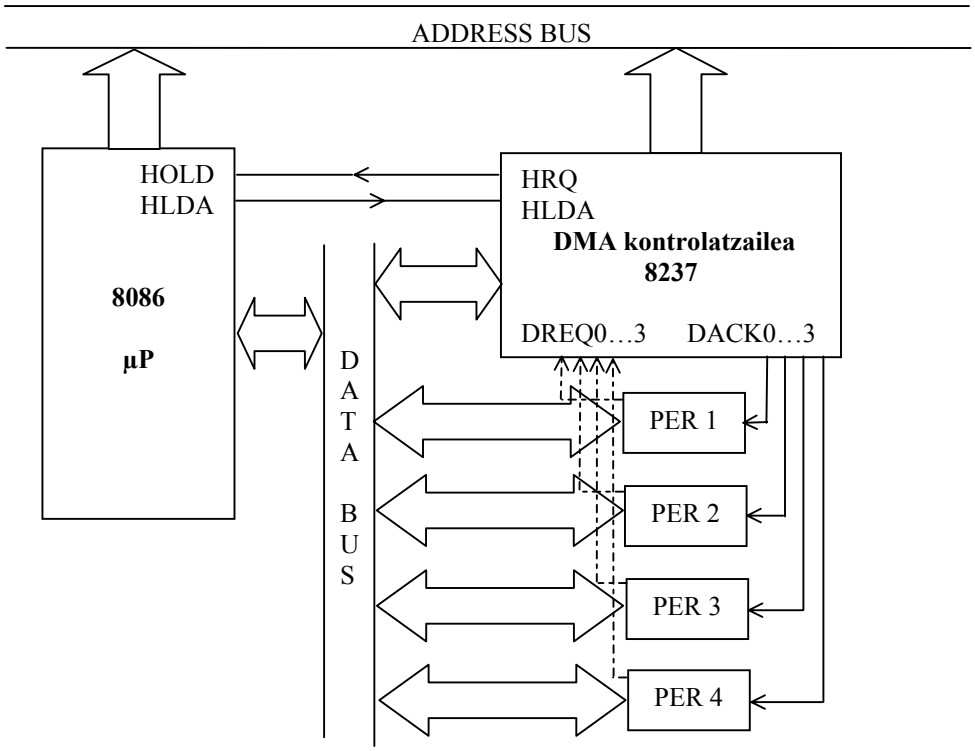
Funtzionamendua

1. Elementu periferiko batek DMA zerbitzua eskatzen dio 8237ari DREQx (x:0..3)aren bidez, eta, horren ondoren, 8237ak µP-aren HOLD pina aktibatzen du;
2. µP-ak, uneko programa utzi, bere hiru busak (datuena, helbideena eta kontrolekoa) inpedantzia handiko egoeran jarri, eta 8237ari HLDA (onespena) bidez erantzuten dio, sistemaren hiru busen kontrola laga egiten diola adieraziz;
3. 8237ak memorian zuzeneko atzipena egiteko aukera ematen dio eskatzen duen elementu periferikoari DACKx-aren bidez;
4. elementu periferikoak DMA transferentzia egiten du, eta, hori amaitzean, 8237ak desaktibatu egiten du µP-aren HOLD pina, eta µP-ak berriro berreskuratzen du sistemaren busen kontrola.

HOLD sarrerak INTR eta NMI etendura-sarrerek baino lehentasun handiagoa du. Gainera, etendurek eragina dute gauzatzen ari den aginduaren egikaritzapenaren amaieran:HOLDek egindako DMA eskaera onetsia izaten da agindua egikaritu bitartean.

PC-XTen kasuan, 8237 bat du, eta bere 0 kanala RAM dinamikoa freskatzeko erabiltzen da; 1 kanala disko gogorraren kontrolatzaileak erabiltzen du, 2 kanala disketearen kontrolatzaileak eta 3 kanala libre egon ohi da normalean.

Lau kanal baino gehiago nahi izanez gero, hainbat 8237 DMA kontrolatzaile kaskadan konektatzeko aukera dago.



7-10 irudia

7.3.4 80286 µP-a

Sarrera

1982an, 80286 µP-a irten zen merkatura. 130.000 transistorek osatzen zuten, eta 8, 10, 12 eta 16 MHz-eko erloju-maiztasunak zituen. 8086 ereduak duen 14 barne-erregistroko multzo berberaz eta 16 biteko datu-busaz osatuta dago; beraz, 16 biteko µP-a da. Haren helbide-busa aurrekoarena baino handiagoa da, 24 bitekoa, eta horrek aukera ematen dio 80286 ereduari gehienera ondorengoak helbideratzeko:

- 64 kilobyte S/I (I/O) esparruan.
- 16 megabyteko (2^{24}) memoria fisikoa (benetakoa, erreala) eta 1 gigabyteko (2^{30}) memoria birtuala edo alegiazko memoria.

1984. urtean, IBM® etxeak bere bigarren PCa jarri zuen merkatuan: PC-AT® eredua zen, eta 80286 µP-an oinarrituta zegoen.

80286 µP-a, funtsean, 8086 eredu hobetua edo aurreratua da, erabiltzaile anitzeko inguruneetarako diseinatua. Hobekuntza garrantzitsuenen artean, ondorengo hauek nabarmendu daitezke 286 ereduarentzat:

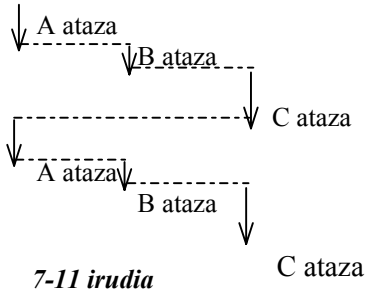
- 8086 ereduak baino abiadura handiagoan egikaritzen ditu aginduak, erloju-maiztasun berberarekin;

- MMU, Memoria Kudeatzeko Unitatea, sartua du, eta horrek aukera ematen dio multiataza egiteko eta memoria birtuala edo alegiazkoa erabiltzeko.

Multiataza

CPUak "aldi berean" anitz ataza egiteko aukera izatea da (adibidez; dokumentuak inprimatu, pantailan idatzi, etab.), kontrolatzen dituen programen artean bakoitzari eskaintzen dion arreta (denbora) oso azkar aldatuz.

Hori abiadura handian eginez, ataza guztiak aldi berean egiten diren itxura ematen du. Ataza bakoitzari oso denbora gutxi esleitzen zaio, eta ez du zertan beti berdina izan. Hori sistema eragilearen kargura egiten da.



7-11 irudia

Memoria birtuala edo alegiazko memoria

Alegiazko memoriak aukera ematen dio ordenagailuari duen memoria (memoria fisikoa) baino askoz ere memoria handiagoa balu bezala jokatzeko.

PCaren kasuan, hardwarearen eta softwarearen diseinu konplexua eginez lortzen da hori, eta horrek aukera ematen dio disko gogorraren memoria sistemaren memoria nagusiaren zati moduan erabiltzeko.

Lanerako moduak

80286 ereduak lan egiteko bi modu ditu: modu erreala edo benetakoa, eta modu babestua.

a) Modu erreala edo benetakoa

Berrasieratu ondoren µP-a abiarazteko modua da, eta MMUak ezgaituta jarraitzen du. 80286 ereduak 8086 eredua balitz bezala jarduten du, baina arinago lan eginez, eta 16 megabyte helbideragarrietatik lehenengo megabytea bakarrik du bere esku.

PC-AT ereduak MS-DOS® sistema eragileakin funtzionatzen zuenean horrela egiten zuen!!

b) Modu babestua

MMUak gaituta dago; beraz, multiataza egin eta alegiazko memoria erabil ditzake. Ataza (programa) bakoitzak besteekin partekatu gabeko memoria esparrua eta xede

orokorreko erregistro multzoa (partekatua, eta, horregatik, pilan gordeak edo salbatuak izaten dira ataza kommutatzean) erabiltzen ditu, egikaritzen denean.

Programa batek ikusitako helbidea alegiazko helbidea edo helbide birtuala da, eta MMUa da benetako helbide errealak alegiazko helbide bihurtzeaz arduratzen dena, eta baita alderantzizkoaz ere.

Benetako moduan edo modu errealean, memoriaren helbideratzea segmentu-helbidearekin eta desplazamendu-helbidearekin (offset-arekin) egiten da, eta modu babestuan, aldiz, segmentu-deskribatzailearekin eta desplazamendu-helbidearekin egiten du.

7.1.5 80386DX µP-a

1985. urtean, Intel-ek bere 32 biteko lehen µPa sartzen du merkatuan: 80386DX-a, 275.000 transistorez osatua eta 16, 20, 25 y 33 MHz-eko erloju-maiztasunekin. Barne-erregistroen multzoa 16 erregistrok osatzen dute; aurrekoekikoekin konparatuz, segmentu-erregistro bi ditu: FS eta GS. Gainera, segmentu-erregistroak izan ezik (16 bitekoak), beste guztiak 32 bitekoak dira.

Hori dela eta, 32 biteko erregistro berri hauei aurrekoekiko *E* aurrizkia gehitzen zaie: adibidez, AX→EAX. Modu horretan 32 biten atzipena lortzen da, hots EAX erabiliz, baina aurreko µPetan bezala egiten bada, hots, AX erabiliz (izan ere, horrela gertatzen da µP honek aurrekoarentzat eginiko kodea egikaritzen duenean).

Bere helbideratze-ahalmenari dagokionez, 4 gigabyteko (2^{32}, giga=10^9) memoria fisikoa (benetakoa, erreala) eta 2^{46}=64 terabyteko (tera=10^{12}) memoria birtuala edo alegiazko memoria da. S/I (I/O) esparrua, aldiz, 64 kilobytetan mugatua mantentzen da.

80286 µP-a, funtsean, 8086 eredu hobetua edo aurreratua da, erabiltzaile anitzeko inguruneetarako diseinatua. Hobekuntza garrantzitsuenen artean, ondorengo hauek nabarmendu daitezke 286 ereduarentzat:

80386 µP-a 80286 eredu hobetu eta aurreratua da: hainbat agindu erloju-ziklo bitan bakarrik egikaritzen ditu, MMU hobetua dauka, eta 32 biteko µP-a da; helbideratze-ahalmena 286a baino handiagoa da, eta beste segmentu-erregistro bi gehiago ditu.

80386 prozesadoreak hiru lan modu ditu:

- *modu erreala*, 286aren berdina
- *modu babestua*, 286arena baino zertxobait hobeagoa
- *8086 alegiazko modua*, modu erreal eta babestuaren konbinazioa. Modu hau modu errealeko programekin multiataza egiteko diseinatu zen. Memoria-helbideratzea modu errealekoa da (seg:offset), eta ataza bakoitzak memoria errealaren 4 gigabyteetatik 1 megabyte xedatzen du.

Urte batzuk geroago, 1988an, 80386DX-aren mugaturiko bertsioa merkaturatu zen: 80386SX µP-a. 32 biteko µP bat da, eta ezaugarri hauek ditu:

- 16 biteko kanpoko datu-busa du, eta horregatik DX bertsioa baino motelagoa da memoria, eta I/O esparruetako atzipenetan 32 biteko hitzekin aritzen denean.
- 24 biteko helbide-busa (80286aren helbideratze-ahalmen bera)

Prozesadore hau merkaturatzeko arrazoia izan zen garai hartako 16 biteko (datu-busa) gailuak 32 bitekoak baino merkeagoak eta lortzeko errazagoak zirela: 80386SX µPan oinarrituriko etxeetako ordenagailuetan dirua aurreztea ekarri zuen, 80386DX-ean oinarritutakoak garestiak baitziren jende gehienarentzat (pauso handiegia eman zen 286etik 386DX-era, eta geroxeago zuzendu egin nahi izan zen).

1990. urtean, Intel-ek 80386SL eskuko ordenagailuentzat diseinaturiko µPa merkaturatu zuen, non energia-kontsumoa eta pisu/bolumen murritza nagusi diren. 80386SL-a 805.000 transistorez osatua dago, hainbat periferiko integratzen baititu txip berean µParekin batera. Gainera, Kontsumorako Kudeaketa Unitate bat ere integratzen du, eta, horri esker, CPUa erabiltzen ez den denboran asko murrizten du energia-kontsumoa. 386SX-a bezala, 32 biteko µP-a izan arren, haren kanpo-busa 16 bitekoa da. Helbide-busari dagokionez, 25 bitekoa izanik, 32 megabyterainoko memoria fisikoa konektatzeko aukera ematen du.

7.1.6 80486DX µP-a

1989an, Intel etxearen 80486DX µP-a merkaturatu zen. 25, 33 eta 50 MHz-eko erloju-maiztasunak ditu, eta 1.200.000 transistorez eratuta dago. 80386 µP-aren bertsio aurreratua da:

- erloju-ziklo bakar batean agindu asko egikaritzen ditu;
- 80386DX ereduak duen helbideratze-ahalmen berbera du;
- txip berean honako hauek ditu: 80386 eredukoaren motako CPU bat, 80387 motako matematika-koprozesadorea eta 8 kilobyteko aurrememoria kode eta datuentzat.

Aurrememoria edo cache memoria

Ordenagailuaren memoria memoria-txipez osatuta dago:

- ROM memoria, abio-programa, BIOSen zerbitzu-errutinak eta abar grabatuak daudena;
- RAM memoria, sistema eragilea, aplikazio-programak (egikaritzeko, programen denboraldiko datuak, etab.) eta gisakoak kargatzen direna.

Zentzu horretan, µP-ak RAM memoria atzitzen du programen (kodea) eta denboraldiko datuen bila. Aldi berean, RAM memoria ondorengo elementu hauez osatua egon daiteke:

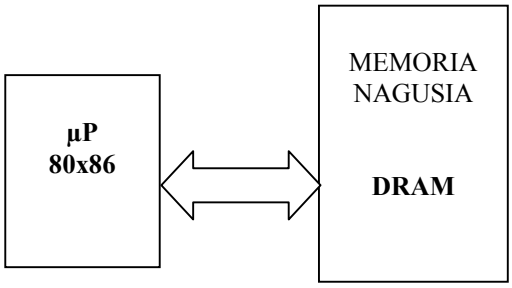
- DRAM memoria txipa (RAM Dinamikoa), kapazitate parasitoz (txikiz) eratua eta ihesak direla eta informaziorik ez galtzeko aldika-aldika freskatua edo berritua izan behar duena.
- SRAM memoria txipa (RAM eStatikoa), erregistro biegonkorrez eratua, ihesik gabea, eta, horregatik, freskatzeko zirkuiturik gabekoa (soilagoa) eta arinagoa. Baina DRAM txipa baino askoz ere garestiagoa da.

80386 µP-a baino lehenagokoetan oinarritutako sistemetan, sistema horien datu-memoria DRAM txipez osatuta zegoen; izan ere, µP-ak txip horiek baino geldiagoak ziren memoria-atzipenak egiten, eta, horrela, diru asko aurreratzen zen, nahiz eta freskatzeko sistema bat inplementatu behar zen (µP-aren memoria-zikloaren denbora DRAM txipen memoria-zikloaren denbora baino handiagoa zen eta). Idazketako edo irakurketako

memoria-zikloen denbora kokapena atzitzeko denboraren eta helburura informazioa garraiatzeko denboraren baturak ematen du.

80386 µP-tik aurrera, ordea, µP-ak DRAM memoriak baino askoz ere lasterragoak dira memoria-atzipenetan, eta horrek pentsarazi zuen arazo hau bi irtenbide posiblerekin konpon zitekeela:

1. Itxarote-egoerak sartuz (atzerapen-zikloak) µP-an, denbora-erlazioa berdindu arte. Ez da irtenbide ona; izan ere, sistemaren abiadura DRAM memoriena da.
2. Aurrememoria (cache memoria) erabili, itxarote-denborak sartzea saihesteko eta sistemaren abiadura µP-arena izan dadin lortzeko.



7-12 irudia. Cache memoriarik gabeko sistema

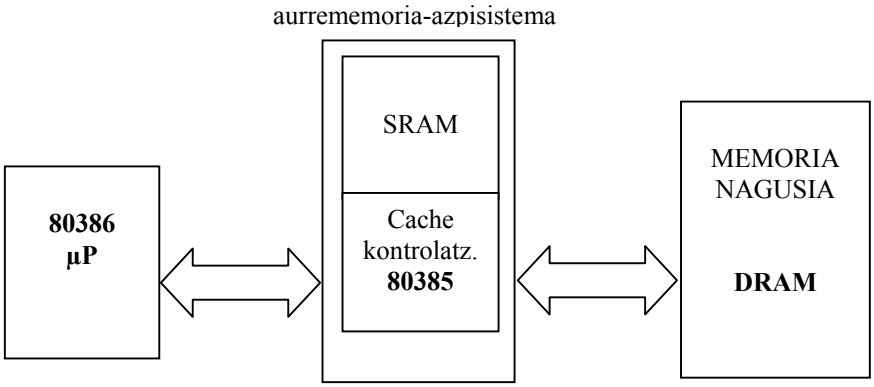
Aurrememoria edo cache memoria CPUaren eta sistemaren memoria nagusiaren (DRAMez eratua) artean kokatutako abiadura handiko memoria iragankorra da, eta SRAM memoriadun txipez osatuta dago.

Aurrememoriarekin, SRAM txipez eratutako memoria nagusidun sistemarekin lortzen direnen ia abiadura berdineko prestazioak lortzen dira, baina DRAM txipez eratutako memoria nagusidun sistemak duen kostua baino zertxobait handiagoarekin.

Aurrememoria-kontrolatzaileak (80385a 80386arentzat) DRAM txipen informazioa kopiatu egiten du SRAM txipetan (datuak eta kodeak), µP-ak behar izan baino lehen (µP-aren agindu-sekuentziari aurrea hartzen dio, iragarpen-metodoaren bidez). Aurrememoriaren kontrolatzaileak µP-ak behar duen informazioa kargatzen ez duenean, orduan, µP-ak DRAM txipak atzitu behar ditu dagozkion itxarote-egoerak sartuta.

Aurrememoriaren tamaina mugatuta dago (4 eta 256 kilobyte artean), berekin dakarren kostuagatik eta eraginkortasunagatik. Izan ere, zenbat eta aurrememoria kantitate handiagoa, orduan eta sistemaren eraginkortasun handiagoa lortzen da (sistemaren memoria nagusiaren kopia-ahalmen handiagoa); baina, aurrememoria kantitate jakin batetik aurrera, aurrememoria-azpisistemaren etekin txikia dakar aurrememoria asko gehitzeak.

Aurrememoria-azpisistema aurrememoriak eta kontrolatzaileak osatua dago. 80386 µP-aren kasuan 80385a da, eta sistema horietan kanpokoa da. 80486 eredutik aurrera, ordea, aurrememoria-azpisistema µP-aren txiparen barruan integratuta dator.



7-13 irudia. Sistema bat aurrememoriarekin.

Matematika-koprozesadorea

Eragiketa matematikoak egiteko bereziki diseinatutako prozesadorea da. µP-ak (CPUak) baino abiadura eta doitasun handiagoarekin egiten ditu eragiketak. Matematika-kalkuluak koma higikorrean egiteko agindu sorta bereizia du, eta agindu horiek barne dituzten programa guztiek aprobetxatzen dute gailu hau.

7-1 taulan, Intel-en 80x87 familiako matematika-koprozesadoreen zerrenda agertzen da, beren 80x86 familiako hurrenez hurreneko diseinatuak izan diren µP-ekin batera.

Koprozesadorea		*Mikroprozesadorea*
8087	→	8086/8088
80287	→	80286
80387SX	→	80386SX/SL
80387DX	→	80386DX
80487SX	→	80486SX

7-1 taula

Koprozesadore horiek gai dira zeinudun zenbaki osoekin aritmetikako eragiketak, trigonometriakoak eta logaritmoenak BCD (Bitar Kodetu Hamartar) kodean eta koma higikorrekin (errealekin) egiteko.

Bi urte geroago, 1991n, honako hauek merkaturatu ziren:

- 80486SX µP-a, DX bertsioa baina koprozesadorerik gabea (ezgaituta dauka)
- 80487SX kanpo-prozesadorea, 80486SX ereduan oinarritutako PCen erabiltzaileentzat; koprozesadoreen zokaloan konekta daiteke (plaka nagusian alegia) eta, horrela, haien abantailak aprobetxatu. Benetan, ordea, 80487SX koprozesadorea ez da koprozesadore soila, 80486DX prozesadore oso-osoa baizik, eta, koprozesadorearen zokaloan konektatu ondoren, sistemaren kontrola hartzen du, 80486SX prozesadorea deskonektatuz.
- 80486 Over Drive ereduak 80487SX koprozesadorearen eginkizun eta konexio modu berdin-berdinak ditu, baina µP horren barneko erloju-maiztasuna kanpoko

(plaka nagusiaren) erloju-maiztasunaren bikoitza izatearen abantailarekin; horrek, oinarrizko plaka bera mantenduz, abiadura handiagoa ematen dio sistemari.

Urtebete geroago, 1992an, honako hauek atera ziren merkatura:

- 80486DX2 µP-a, Over Drive ereduaren CPUa edo zokalo nagusiarentzako bertsioa baizik ez dena;
- eta 80486SL µP-a, 486 ereduaren bertsio bat, ordenagailu eramangarrientzat.

1994an, 80486DX3 µP-a atera zen merkatura, zokalo nagusiarentzako, eta kanpoko erloju-maiztasunaren hirukoitza du.

7.1.7 Pentium µP-a (80586)

µP familia ezagun honetako hurrengo kidea Pentium prozesadorea da. 80586 izendatu beharrekoa da, eta lege-arrazoiengatik ez zen horrela gertatu. 1993. urtean irten zen merkatura $3_1100.000$ transistorez osaturik, eta txip berean CPU 2 eta 16 kilobyteko cache memoria zuela (8 datuentzat eta beste 8 kodearentzat).

7-14 irudia. Pentium I µP-a

32 biteko helbide-busa dauka, bai I/O esparruan eta bai memoria–esparruan, eta helbideratze-ahalmena bere familiako aurreko bi ereduenen berbera da: 80386DX eta 80486-arena. Gauza bera gertatzen da barne-erregistro multzoarekin ere. Dena den, datu-busarekiko, esan beharra dago kanpokoa nahiz barnekoa 64 bitekoa dela.

Pentiuma 2 CPUz osaturik dagoelako gertatzen da hori, bata 386 motakoa, eta bestea 486 motakoa. Lehengoa osokoen unitatea da, eta bigarrena, aldiz, puntu higikor eta osokoena; beraz, lehengoa baino hobeagoa da. Unitate bakoitza 32 bitekoa da, eta unitate, bide nahiz pipeline deritze.

Pentium prozesadorea, bere supereskala-egiturari esker (bide bat baino gehiago), paraleloko prozesamenduko µP-a da, instrukzio bi egikaritzen baititu aldi berean: bide bakoitzetik bat (bide bakoitzetik doazen instrukzioen artean bata bestearekiko mendekotasunik ez denean).

Prozesadore honek bi bide erabiltzen ditu automatikoki: programa bat hartu, irakurri eta bi bidetan zatitzen du. Baina hori ezin liteke beti egin, programa batzuek barne-mendekotasunak baitituzte. Adibidez:

		U bidea		V bidea

1.	`A=3`
	`B=2`
	`C=A+B`
		`A=3`
		`C=A+B`
				`B=2`

2.	`A=1` `A=A+1`	`A=1` `A=A+1`	edo	`A=A+1;` ezinezkoa da `A=A+1`

"Pentium-arentzako diseinatutako programak" Pentium-ak bi bideak okupatuta edo beteta mantentzeko egokitutako aginduak dituzten programak dira, eta, horrekin, egikaritze-abiadura handiagoa lortzen da. Programa horiek 80386 eta 486 ereduek ere egikari ditzakete, baina abiadura txikiagoz, bide bakarra dute eta. Pentium-ak bi bideak beterik mantentzen dituenean, 64 biteko µP batek bezala jokatzen du, ziklo bakoitzean memoriatik 32 biteko instrukzio 2 irakurtzen baititu (bina-bina, 64 biteka).

Pentium µP-a 486DX2-66 eredua baino lasterragoa da, baldin eta egikaritu behar den programa bere aurrememoriaren 16 kilobytetan osorik sartzen bada eta Pentiumarentzat diseinatua izan bada. P24T Pentium-ak (Pentium SX) 64 biteko barne-datuen busa du eta 32 bitekoa kanpoan, 16 kilobyteko aurrememoria eta 2 bide, azken horrek zentzu handirik ez duen arren.

P54C Pentium-ak (Pentium I), aldiz, 64 biteko datu-busa du, bai barnean eta baita kanpoan ere. P55C Pentiuma MMX® Pentiuma da.

Pentium-ak funtzioak bikoizteko sistema inplementatzen du, hau da, bi µP Pentium bus berean konekta daitezke, eta, horrela, batak maisu gisa funtzionatzen du eta besteak morroi moduan. Biek agindu berberak egikaritzen (burutzen) dituzte aldi berean, lortutako emaitza guztiak egiaztatzen dituzte, eta emaitzak bat ez baldin badatoz, programaren egikaritzapena eteten dute.

7.2 PC-en hedapen-busak

µP-a edo CPUa PCaren elementu nagusia da, eta PC-aren plaka nagusian edo oinarrizko plakan (motherboard) kokatzen da. Plaka nagusia PCak duen zirkuitu inprimatuko txartel (zirkuitu-plaka) nagusia da, handiena alegia.

µP-a oinarrizko txartelaren gainerako elementuekin komunikatuta dago busen sistemaren bidez (datu-busaren, helbide-busaren eta kontrol-busaren bidez). Bus-sistema hori zirkuitu inprimatuko hari eroale multzoa da, eta hari horiek eskaintzen dute informazioak gailu desberdinen artean bidaiatu ahal izateko aukera.

Plaka nagusiaz gainera, µP-ak komunikatzeko erabiltzen dituen elementuak honako hauek dira: matematika-koprozesadorea, etenduren kontrolatzailea, DMAren kontrolatzailea, memoria-txipak, etab.

Baina µP-a oinarrizko plakan ez dauden beste elementu batzuekin ere komunikatzen da, eta hedapen-busei esker gertatzen da hori. Hedapen-bus bat, konektore batean, erreten edo slot batean bildutako (derrigorrez ez denak) bus-sistemaren seinale multzoa da.

Oinarrizko plakan ez dauden elementuak arteka edo erreten horietan entxufa daitezkeen zirkuitu inprimatuko txartelak izan ohi dira.

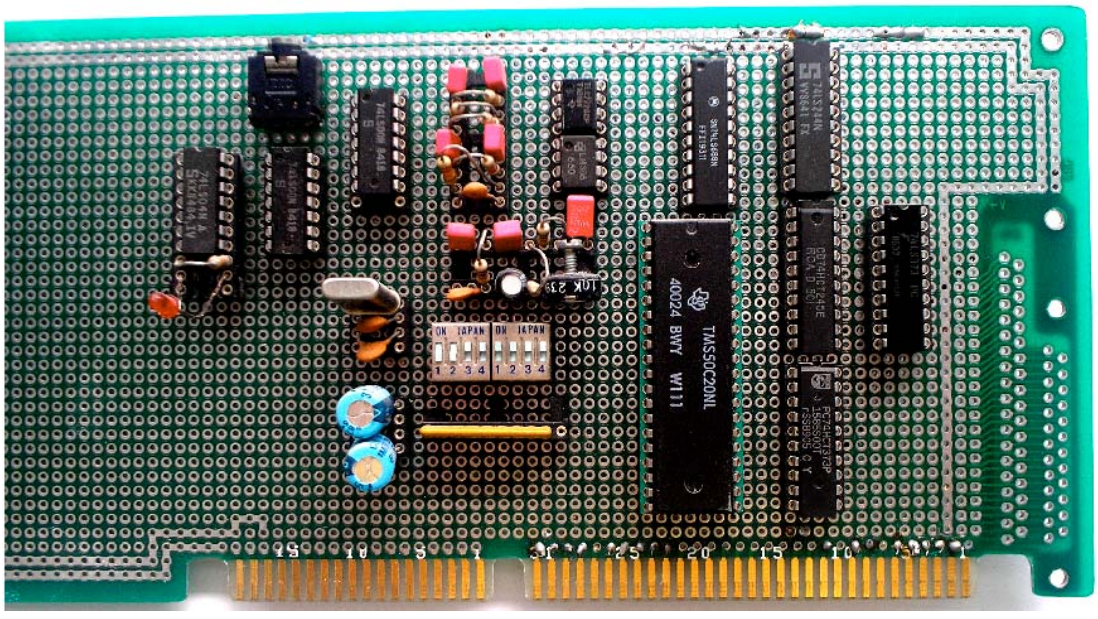
7.2.1 XT edo 8 biteko ISA® (Industry Standard Architecture;Industriako Arkitektura Estandarra) hedapen-busa

1981ean IBMren lehenengo PCa, PC-XT, merkatura irten zenean, PC hura Intel etxearen 8088 µP-an oinarrituta zegoen, kanpoko datu-busa 8 bitekoa zuen (barnekoa 16 bitekoa) eta helbide-busa 20 bitekoa.

PC hura muntatzen zuen hedapen-busa XT zen, eta erreten edo konektore (slot) bakoitzeko 62 hari eroale zituen: 8 datuenak, 20 helbideenak, 8 elikatze-harienak (GND, +5v, -5v. +12v, -12v, non batzuek errepikatu egiten diren), 26 kontrol-hari (idazketa- eta irakurketa-seinaleak memorian eta S/I-n (I/O-n), S/I eta memoriako gailu geldientzat itxarote-egoeren seinaleak, etendura-eskaeren seinaleak, DMAren zerbitzu-eskaeren sarrerak, RAM dinamikoa freskatzekoak, etab.).

7.2.2 AT hedapen-busa edo 16 biteko ISA

Merkatura irten zen IBM etxearen bigarren PCa PC-AT izenekoa izan zen, eta 80286 µP-an oinarritua zegoen. Prozesadore hark 16 biteko datu-busa zuen, eta 24 biteko helbide-busa.



7-15 irudia. AT motako txartel prototipoa (Eibarko Ingeniaritza Eskolan egina)

Hedapen-busa PC-ATren plaka nagusian muntatuta zetorren, eta 98 seinalez osatuta zegoen: 62 XT busaren hariak, eta 36 gehigarri datu-busaren, elikatze-seinaleen eta elikatze-lineen hedapenak jasateko.

7-16 irudia. AT busa (goian), eta PCI Bus lokala (behean)

Fisikoki, AT bus-konektorea XT bus-konektoreaz eta mota bereko beste konektore laburrago batez osatuta dago, biak lehenengoaren goiko aldetik batuta (estekatuta).

Horrek aukera ematen du lehengo XT hedapen-txartelak (8 biteko ISA) AT (16 biteko ISA) eta EISA® (32 bitekoa) erretenik berrienetan konektatu ahal izateko, betiere duten forma edo itxura (irudi) fisikoak horretarako aukerarik ematen badu.

7.2.3 MCA® eta EISA® busak, Mikrokanal Arkitektura eta ISA Hedatua

PCen munduko 32 biteko ezagunenetako bi hedapen-busak dira. 80386DX prozesadoreetan edo maila handiagokoetan oinarritutako PCetan muntatuta datoz (32 biteko datu-busa, 32 biteko helbide-busa eta haien aurrekoek baino erloju-maiztasun handiagoa).

MCA eta EISA busen ezaugarri nagusiak honako hauek dira:

1. 32 biteko busak dira (datu-busak).
2. ISA busaren 8 MHz-ak gainditu egiten dituzte; hedapen-busaren erloju-maiztasuna µP-aren berdina da 8 MHz baino txikiagoa edo berdina den bitartean, baina 8 MHz-era mugatuta gelditzen da µP-aren erloju-maiztasuna aipatutako kopurua baino handiagoa baldin bada, eta, hala, txartelen abiadura mugatu egiten du abiadura horretan.
3. "Bus mastering" delakoa jasaten dute. Ezaugarri horrek periferikoen CPUei (kontrolatzaileei) beren artean komunikatzeko aukera ematen diete CPU nagusiak (µP-ak) kontrolatzaileen artean datuak transferitzeko bitartekari moduan une oro jardun gabe, eta hasierarako eta amaierarako bakarrik jardunda. Horrek sistemaren errendimendua handitzen duen arren, MCA eta EISA hedapen-txartel gehienek ez dute erabiltzen.
4. Zaratarekiko eragingaitzagoak (immuneagoak) dira. Horregatik, datu-transmisioetan akats gutxiago gertatzen dira.
5. Hedapen-txartelak software bidez konfiguratzeko aukera ematen dute. Hedapen -txartelek S/I (I/O) esparruko helbideak (S/I portuak) erabiltzen dituzte PCaren CPUarekin komunikatzeko, baina memoria-helbideak, DMAren kanalak (8237) edo hardware-etendura maskaragarriak (8259A) ere erabiltzen dituzte. ISA hedapen-txartelak (8 eta 16 bitekoak) hardware bidez konfiguratzen dira DIP etengailuak fisikoki manipulatuta; MCA eta EISA hedapen-txartelak, aldiz, software bidez konfiguratzen dira dagokien konfigurazio-programaren bidez. S/I (I/O) helbideak (portuak) eta PCen beste baliabide batzuk hedapen-txartelen bati esleitzean, PCak erabilgarri dituen baliabideen zerrenda ikusi behar da, baliabide bera txartel batek baino gehiagok erabiltzeagatik gatazkarik sor ez dadin.

MCA busa 1987an sortu zuen IBM etxeak, 50 eredutik abiatutako PS/2® (Personal System) PCentzat. Bus hori bateraezina da lehengo XT eta AT busekin (erreten-tamaina eta hari-banaketa); beraz, IBM etxeari erantzunez eta ISA busekin bateragarritasuna mantentzeko asmoz, 9 fabrikatzailek (Intel®, Olivetti®, Compaq®,…) osatutako taldeak sortu zuen EISA® busa.

7.2.4 PCI® (Peripheral Component Interconection; osagai periferikoen arteko konexioak) eta VESA® (Video Electronics Standards Association; Bideo-elektronikaren Estandarren Elkartea) bus lokalak

Disko-unitateak edo inprimagailuak eta horien antzeko elementu periferiko geldiak kontrolatu ahal izateko, nahikoa da orain arte ikusitako hedapen-busak erabiltzea, baina, elementu periferiko arinak (lasterrak) kontrolatu ahal izateko —egungo txartel grafikoak edo sare lokalen kontrolatzaileak, esaterako—, behar-beharrezkoa da orain arte ikusi ditugun busak baino bus lokal lasterragoa erabiltzea.

Bus lokalak zuzen-zuzenean konektatzen du elementu periferikoa µP-aren busarekin, eta elementu periferikoa hedapen-busetik kanpo gelditzen da. Normalean, bus lokalak, PCaren arabera, ISA, MCA edo EISA erretenak bere barne izaten ditu, baina, gainera, baita bus lokalaren erretenak ere, eta, hala, aurreko txartelekiko bateragarritasuna mantentzen du.

PCI bus lokala 32 edo 64 bitekoa da (bertsioaren arabera) eta 80486 ereduaren 32 bitak edota Pentium-aren 64ak jasan ditzake. Txartelentzat eskaintzen dituen erloju-maiztasunak 50 eta 66 MHz-ekoak dira.

VESA bus lokala edo *VLB*® (VESA Local Bus), aldiz, abiadura handiagoko eta 32 biteko ISA hedapen-busaren hedapena da. 80486 µP-aren 32 bitak jasaten ditu eta 33 MHz arteko maiztasunak eskaintzen. VESA 2.0 bertsioak, ordea, Pentium-aren 64 bitak eta 50 MHz-eko maiztasuna (abiadura) ditu.

Bus hori PCI bus lokala baino merkeagoa da, bus mastering eta txartelen software bidezko konfigurazioa ere eskaintzen ez dituelako.

7.2.5 PCMCIA® (PC Memory Card International Association / PCentzako Memoria Txartelen Nazioarteko Elkartea) bus eramangarria

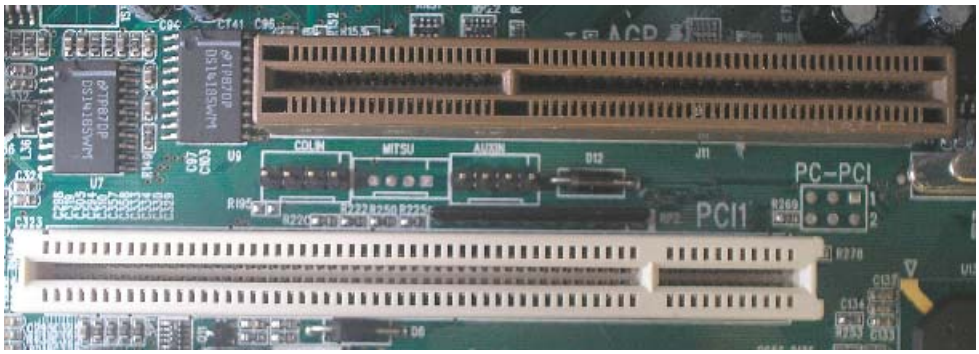
PC eramangarrientzako hedapen-busa da. Hasieran memoria-txartelentzat sortu zen, baina, gaur egun, ia elementu periferiko guztientzat erabiltzen da.

PCMCIA txartelak mahaiko PCen txartelak baino garestiagoak eta txikiagoak dira (kreditu-txartelen tamainakoak).

7.2.6 AGP® busa (Grafikoen Portu Arindua)

Bereziki txartel grafiko modernoenentzako diseinaturiko (Intel) busa da, haien prestazio bikainak eskaintzeko informazio-fluxu handia behar baitute (informazio asko abiadura handian).

Bus hau Pentium-II eta ATX plakekin agertu zen (azken AT plaketan ere existitzen den arren), eta haren helburua PCI busa arintzea da. Horretarako, txipsetari zuzenean konektatzen zaio, eta, hala, bai prozesadorerako eta bai RAM memoriarako atzipen arinagoa lortzen da.



7-17 irudia. AGP busa (goian), eta PCI Bus lokala (behean)

7.3 BIOS-a (Sarrera eta Irteerako Oinarrizko Sistema)

ROM-BIOS-a soilik irakurtzeko memoria txip multzo bat da. Multzo horretan, ordenagailuaren abio-programa eta sarrera eta irteera oinarrizko sistemaren zerbitzu-errutinak daude. Haien edukia ez da galtzen.

ROM-BIOS-a goi-memoriaren goieneko aldean kokaturik dago (1. megabytearen barnean) eta ordenagailua piztean jartzen da martxan. Orduan, lehenengo plaka nagusiaren azterketa (txekeoa) egiten du, jarraian plakari konektaturiko eta setup-ean konfiguraturiko beste gailu guztiena, eta, dena ondo joan bada ordura arte, arrankatu eta PC-aren kontrola sistema eragileari pasatzen dio.

BIOS-aren errutinek sarrerak eta irteerak kudeatzen dituzte, hau da, aplikazio-programa eta hardware arteko interfazea da: programa batek pantailan zerbait idatzi edo teklatutik zerbait irakurri behar duen bakoitzean, BIOS-aren bidez egiten du. Eragiketa horiek zuzenean egin litezke programaren eta hardwarearen artean. Dena den, arriskua dago programa horrek PC guztietan ez funtzionatzeko, hardwarearen ezaugarriak zertxobait edo oso ezberdinak izan baitaitezke.

BIOSaren erabilerak izugarri errazten du programazio-atazen eta programen bateragarritasuna. Nagusiki hiru BIOS-fabrikatzaile daude: Award®, AMI® eta Phoenix®.

BIOSa plaka nagusian dator instalaturik, 24 edo 28 pineko DIP motako txip batean, edo PLCC motako batean. Plaka erosiz gero, beraz, ez dago instalatu beharrik.

Lehenengo PCetan, ROM memoria txipetan zetorren, geroago EPROM txipetan, eta gaur egun ia BIOS guztiak FLASH-EEPROM txipetan datoz. Horietan birprogramatu egin daitezke, hau da eguneratu edo gaurkotu, plaka berean.

Plaka baten BIOS-ak data horretarainoko ia hardware multzo osoa ezagutzen du, baina ez ondorengo datadun gailuak; beraz, PCari gailu berri horietako bat konektatzeko asmorik balego, BIOSa eguneratu beharko litzateke ondorengo bertsioren batera. Izan ere, bestela ez luke ezagutuko gailu hori, edo aurreko beste eredu bat balitz bezala jardungo luke (segur aski aurrekoren bat, eskasago).

Horrenbestez, modu bi daude BIOSa eguneratzeko:

Hau da gehien erabiltzen dena: fabrikatzaileak ematen duen (ia beti Internet bidez lortua) eguneratze-programa baten bidez, BIOS zaharra berriarekin ordezkatzen da. Bestea ez da ia erabiltzen: BIOSa daukan txipa bertsio berriagoa daukan batekin ordezkatzean datza.

Lehenengo moduaz txarto eginez gero BIOSaren eguneratzea, hori hondatu edo galdu egiten da, eta plaka nagusia gauzaeztandu egiten da. Orduan, plaka aprobetxatzen jarraitu nahi izanez gero, bigarren modua erabili beharko genuke, txip hori oraindik merkaturatzen bada behintzat.

Plaka nagusi batzuek bigarren BIOS txipa ere izaten dute, plaka eguneratzekoeragiketako akatsetatik salbatzeko, eta berriro (lehen) BIOSean lehen edukia kargatzeko.

PC-AT (80286)-tik orainoko ordenagailuek beren plaka nagusian pilez elikaturiko CMOS memoria ez-hegazkorra gaineratzen dute, non sistemaren konfigurazio-datuak, eta eguneratutako ordua eta data gordetzen baitira.

SETUa BIOSaren programa berezi bat da, eta CMOSean gorderik dagoen informazioa atzitu eta alda dezake. Setup-a egikaritzeko pare bat modu daude:

- *Supr* edo *Ctrl+Alt+Esc* sakatuz, ordenagailua martxan jartzean
- *Ctrl+Alt+Ins*, *Ctrl+Alt+Return*, *Ctrl+Alt+S* sakatuz, ordenagailua martxan jarri eta gero

7.4 PC-aren plaka nagusia

Plaka ama (motherboard) edo oinarrizko plaka PC baten arkitekturaren elementu nagusia da, hardwarearen oinarrizko pieza, eta horri konektatzen zaizkio ordenagailuaren gailu eta periferiko guztiak. PCaren zirkuitu inprimatu handiena da.

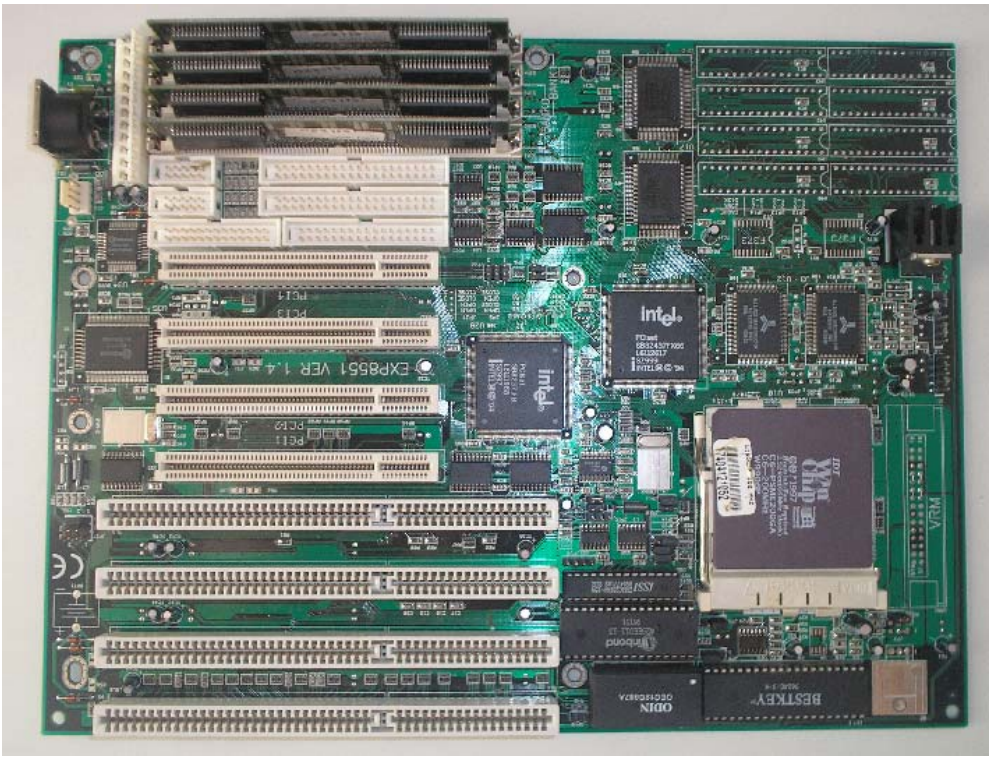
Plaka nagusiari konektatutako elementuen arteko komunikazioa hiru busen sistemaren bidez: datu-busa, helbide-busa eta kontrol-busa. Bus horiek zirkuitu inprimatuko eroale hari–multzo gisa agertzen dira plakako gailuentzat, eta hedapen-bus gisa kanpoko periferikoentzat.

7.4.1 Plaka motak

Gaur egun, plaka mota bi existitzen dira IBM-PC eta bateragarrien munduan:

PC-AT plaka

PC-AT plakak ordenagailuaren diseinua mantentzen du, eta Pentium II-an oinarrituriko ordenagailurainoko ia PC guztietan erabili da.



7-18 irudia. AT motako plaka ama

Ez du mantentzen kableak jartzeko ordena bat, eta fabrikatzaile bakoitzak bere erara egiten du. Horrela, PCa hedatzeak datuak eta elikadura-kableak gehitzea dakar, eta horrek etorkizuneko hedapenak eta barneko gailuen aireztapena zailtzen ditu.

Batzuetan, hedapen-txartel handiak ezin litezke konektatu, µPa "jotzen" baitute, kokapenaren ondorioz.

ISA eta PCI konektoreak izaten ditu, eta batzuetan AGPkoa ere bai. Baita SIMM memorientzako zokaloak ere.

512 kilobyteko edo 1 megabyteko 2. mailako (L2) aurrememoria dauka.

µP-aren zokaloa socket 7 edo socket 370 motakoa izan liteke.

PC-ATX plaka

ATX araua Pentium IIaren agerpenarekin ezarri zen, eta AT eredua puntu hauetan hobetzen du: barneko aireztapena, tamaina handiko hedapen-txartelen atzipena eta jartzea, dimentsio handiagoko eta elikatze-iturri hobeagoko kaxa (konektore berri bat eta kontsumo-kudeaketa), eta gehitu kaxaren atzealdean kokaturiko konektoreak arautzea (serieko ataka, paralelokoa, USB®, PS/2, teklatua, arratoia, etab.)

ISA, PCI, AGP konektoreak eta µP-rentzako socket 7, socket 370 edo slot 1 (Pentium II-an) zokaloa dakartza.

Elikatze-iturriak ACPI (Advanced Control Power Interface) kontsumoaren kudeaketa-sistema bat dakar, eta horrek aukera ematen du PCa sistema eragiletik edota modem bidezko telefono dei batekin (urrundik) martxan jartzeko programatzeko, bai eta itzaltzeko ere, plaka nagusia ez baita guztiz itzaltzen (standby egoeran geratzen da).

Plaka mota bien arteko alde nagusia bere neurrietan eta elikatze-iturriaren konexioan datza (konektorea): ATX kaxak handiagoak dira, eta gehienak AT plaka onartzeko prestatuta daude.

ATX plakak garestiagoak dira, baina gailuen barne-banaketa askoz ere hobea da. Azkenengo AT plakek ATX elikatze-konektoreak dakartzate.

Txipset-a µP-ren eta plakako beste gailu guztien arteko interkonexio-zubi gisa jarduten duen txip kopuruari deritzo. Txipset-ak zehazten ditu plakari konekta dakiokeen DRAMen kopurua, sistemaren busaren maiztasuna eta PCI eta AGP busen informazio-abiadura.

7.4.2 µP-aren zokaloa

Plaka nagusiko zokalo nagusia da, PC-aren µP-a kokatzen (konektatzen) den tokia.

Lehenengo Pentium eta Pentium MMX prozesadoreek erabiltzen zuten zokalo edo socket-a, AMD®-ren K5az eta K6az eta haien baliokideez (haiekiko bateragarriez) gain, socket 7-a zen.

Geroago, Pentium IIa eta Pentium IIIa (IIa 450 MHz-etik aurrera) merkaturatu zirenean, slot 1 zokaloa agertu zen. Intelek diseinatu zuen µP hauentzat; zirkuitu inprimatu luze batean konektatuak doaz beren 2. mailako (L2) aurrememoriaz, non zirkuitu inprimatua plastikozko kartutxo baten barruan baitoa. Slot 1-aren berrikuntza nagusiak bi dira: AGP eta 100 MHz-eko abiadurako sistemaren busa.

Honekin, lehiakideek slot 1-aren egile-eskubideak erostea itxaroten zuen Intelek. Baina Intelen lehiakideek (AMD®, Cyrix®, etab.) *socket 7*-a erabiltzen jarraitzea erabaki zuten beren AMD K6-II eta K6-III, Pentium II eta IIIaren baliokideentzat. Hala, *super 7 plataforma* ezaguna sortu zuten, socket 7-a AGPrekin eta 100 MHz-eko sistemaren busarekin mantentzeko.

Pentium II-a arlo profesionaletarako diseinatua izan zen, *Pentium II Xeon* ordenagailu zerbitzarietarako, *Celeron* etxeetako ordenagailuentzat. Azken horrek ez du L2 aurrememoria; hurrengoak, berriz, *Mendocino* edo *Celeron A*-k, 128 kilobyteko L2 cache memoria du.

Baina, Celeron µP bertsio berriarekin, Intel etxeak zokalo berri bat merkaturatzea erabaki zuen: *socket 370*. Zokalo hori *socket 7*-aren mota berekoa da, baina ez dira bateragarriak, ez baitira berdinak, ez seinale kopuruan eta ezta kokapenean ere. Merkaturatze honekin, Intelek onartzen du errazagoa eta merkeagoa dela beretzat socket 7 edo 370 zokaloentzako prozesadoreak fabrikatzea.

Gaur egun (2000), AT nahiz ATX plakak ekar dezakete socket 7 konektorea nahiz socket 370a. Slot 1 zokaloak ATX plaketan agertzen dira bakarrik, salbuespen bat egon litekeen arren.

7.4.3 Cache memoria edo aurrememoria

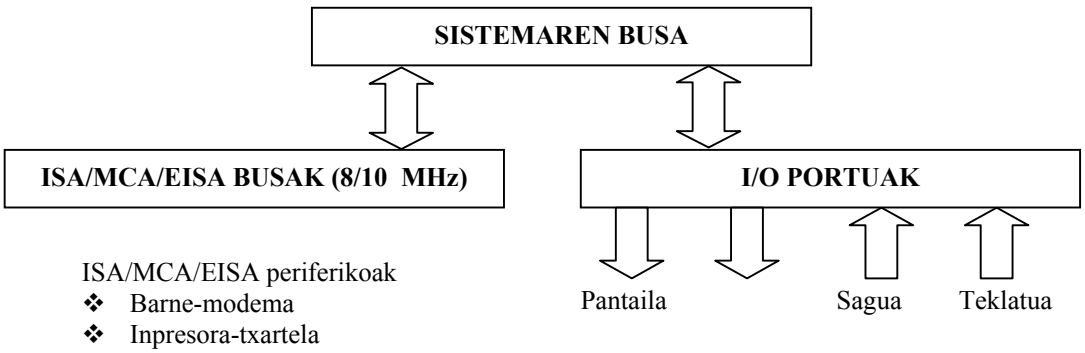
Gaur egun (2000), cache memoria edo aurrememoria maila bi daude: L1 eta L2 (1. maila eta 2. maila), AMD K6-III µPan 3. mailakoa (L3) sartu den arren. L1 µParen barneko aurrememoria da, eta, horregatik, prozesadorearen abiadura bera du; tamainaz, L2a baino txikiagoa da (32tik 64 kilobytera bitartean), eta µPak informazioa bilatzen duen lehen tokia da.

L2 µP-arekiko kanpoaldekoa da, eta sistemaren busaren erlojuaren abiadurarekin korritzen du, Pentium IIak eta IIIak µParen abiadura erdian korritzen duten arren. AMD K6-IIIren kasuan, aldiz, µParen abiadura berean korritzen du. 512 kilobytekoa edo 1 megabytekoa izaten da, eta µP informazioa bilatzen duen bigarren tokia da. L2 aurrememoriak % 15etik % 30era gehitzen ditu memoria-prestazioak, L2 gabeko sistema batekiko.

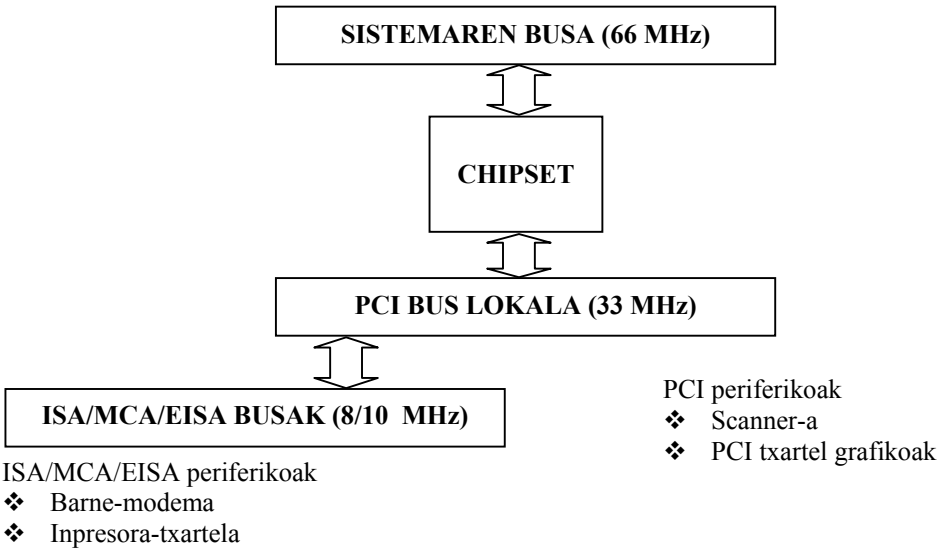
Cache memoriak *igartzeak* eta *hutsegiteak* izan litezke. Igartzetzat hartzen da µPak bilatzen ari den informazioa aurrememorian aurkitzen duenean eta hurrengo mailara atzitu behar ez duenean, eta hutsegitetzat, aurrememorian aurkitzen ez duenean. Igartze kopurua aplikazioaren eta sistema eragilearen araberakoa den arren, % 90 ibiltzen da. Aurrememoriak irakurketa- nahiz idazketa-erako atzipenetan funtzionatzen du.

7.4.4 PCetako busen egitura

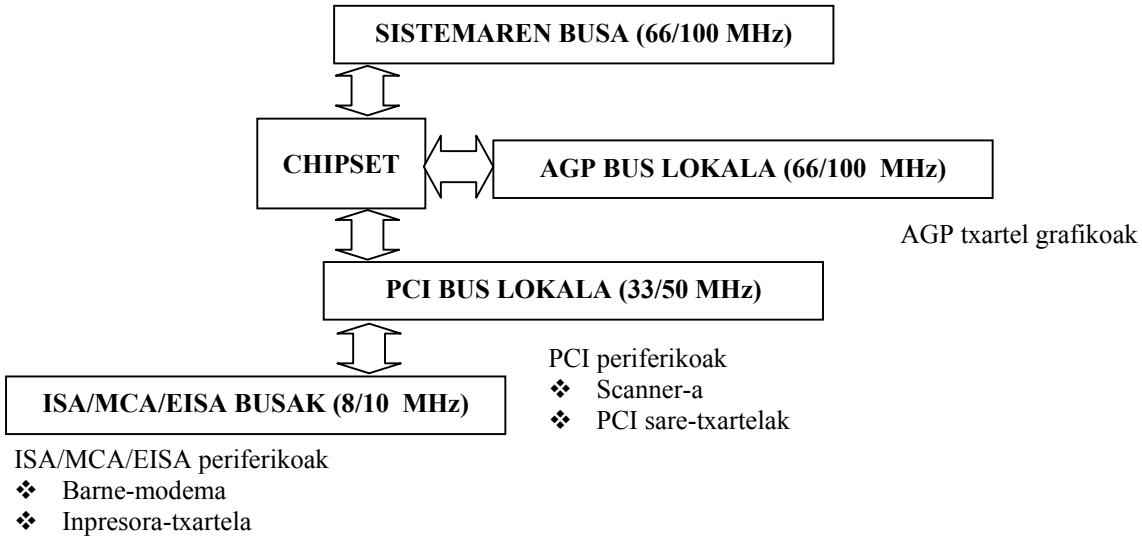
Lehenengoetatik hasi eta gaurdainokoetaraino, ordenagailu pertsonalek izan duten eboluzioa ez da izan bakarrik 80x86 µP familiak izan duen eboluzioari esker; ordenagailuen plaka nagusiaren diseinu-arkitekturaren eboluzioari esker ere izan da. Zentzu horretan, eboluziobidea honelakoa izan da: alde batetik, periferiko zaharren bateragarritasuna mantendu da, eta, bestetik, periferiko berri eta ahaltsuei zubi berriak eskaintzea. Irudi honek adierazten ditu PCen plaka nagusien lehen hiru arkitekturak.



7-19 irudia. 80386/486 µP-an oinarritutako PC-aren bus-eraikuntza



7-20 irudia. Pentium I µP-an oinarritutako PC-aren bus-eraikuntza



7-21 irudia. Pentium II µP-an oinarritutako PC-aren bus-eraikuntza

7.4.5 DRAM memoria-txipak

DRAM txip mota asko dira, eta horren arrazoia da denbora guztian eboluzionatzen ari direla. Horrela, PCen merkatukoak berriagoek eta hobeek ordezkatzen dituzte, eta, denbora gutxira, ez da bat bera ere geratzen merkatuan.

Ikus ditzagun hauetariko batzuk:

- *FPM:* 80386, 486 eta lehengo Pentium-etan oinarrituriko PCetako DRAM txipak dira. 32 eta 72 kontaktuko SIMM moduluetan muntatuak datoz (zirkuitu inprimatuko txarteletan). Onartzen duten busaren fclock handiena 66 MHz-ekoa da (60 ns-ko atzipen-denbora).

- *EDO,* Pentium, Pentium Pro eta lehenengo Pentium IIetan oinarrituriko PCetarako txipak. 72 kontaktuko SIMM moduluetan eta 168 kontaktuko DIMM moduluetan muntatuak datoz. Onartzen duten busaren fclock handiena 66 MHz-ekoa da (45, 50, 60 eta 70 ns-ko atzipen-denborak).
- *SDRAM,* Pentium IIan eta gainerako prozesadoreetan oinarrituriko PCetarako txipak. 168 kontaktuko DIMM moduluak eta busaren fclock handiena 100 MHz-ekoa.
- *PC-100 DRAM,* Intelen 100 MH-ezko BX txipsetarentzako diseinatuak, kalitatez-arau zorrotzak betetzen dituzte.
- *BEDO,* 66 MHz-eko maiztasun gehiengoa.

7.5 Komunikazio-portuak

PCaren komunikazio-portuek mikroprozesadorea kanpoko beste gailu batzuekin komunikatu ahal izateko balio dute.

Gailuen arteko komunikazioak seriean edo paraleloan egin daitezke. Ordenagailu pertsonal orok, gutxienez ere, serieko portu bat eta paraleloko beste bat izaten ditu.

Serieko atakaren eta ataka paraleloaren arteko alde nagusia hau da: ataka paraleloak datuak igortzen eta hartzen ditu, zortzikotez zortzikote (bytez byte; paraleloan) , eta serieko atakak, berriz, bitez bit (seriean). Arrazoi hori dela eta, ataka paraleloa lasterragoa da (8 aldiz lasterragoa gutxienez ere), nahiz eta komunikatzeko gutxienez ere 8 linea (hari) behar dituen. Serieko atakak, aldiz, 3 hari baino ez ditu behar: 1 transmisiorako, 1 hartzeko eta bestea masakoa (lur-konexioa).

Bi ataka motak ere PCak oinarri duen µP-aren S/I (I/O) esparruan kokatzen dira, eta honako portu hauek (S/I helbideak) dagozkio bakoitzari:

Ataka	*S/I (I/O) portu-bitartea*
serieko COM1	3F8..3FFH
serieko COM2	2F8..2FFH
serieko COM2	3E8..3EFH
serieko COM2	2E8..2EFH
Paraleloko LPT1	378..37FH
Paraleloko LPT2	278..27FH

7-2 taula

Serieko atakak elementu periferikoekin konektatzen dira —saguarekin eta kanpoko modemarekin, adibidez—, eta ataka paraleloak berriz, inprimagailu eta eskanerrekin.

7.5.1 RS-232® serieko ataka

PCaren serieko atakak EIA® amerikar erakundearen RS-232® araua (estandarra) betetzen du. Portu hori serieko komunikazio sinkronoan eta asinkronoan erabil daiteke, bere 25 pineko DB-25 konektorearen bitartez. Ia beti, ordea, modu asinkronoan erabiltzen da. Komunikazio mota honentzat oso ohikoa da 9 pineko DB-9 konektoreekin topo egitea. Gainera, komunikazio mota hau denean, ia beti 9 linea horietako 3 bakarrik erabiltzen dira: masakoa, TxD (transmisiokoa) eta RxD (harrerakoa), eta gainerako lineak ez dira erabiltzen.

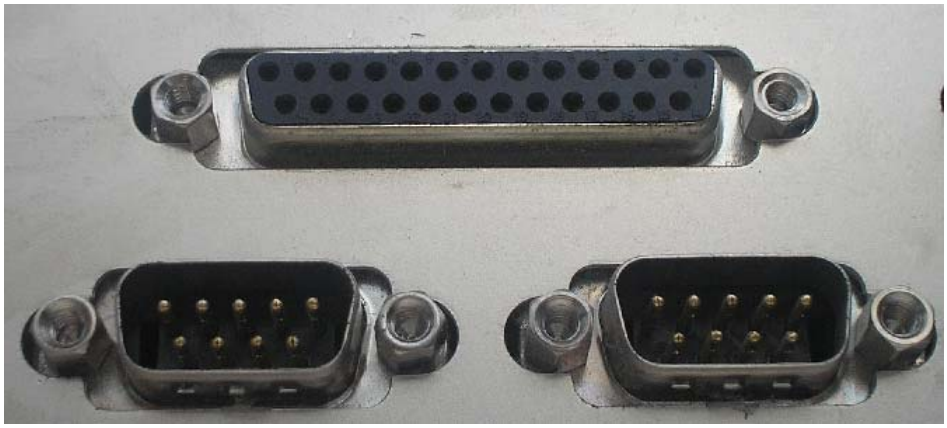
PCaren serieko ataka nagusiki UART 8250 txipean edota 16450 txipean oinarrituta dago (bigarren hori modernoagoa da).

7.5.2 Ataka paraleloa

Normalean, ordenagailu pertsonal batek bi ataka paralelo ditu (Centronics ataka gisa ere ezagunak). Horietako bat aparailuak izango duen DB-25 konektore eme bakarrera konektatua dago. Inprimagailuak eta eskanerrak dira nagusiki ataka horiek erabiltzen dituzten periferikoak. Portu horiek serieko konexioarekin baino komunikazio-abiadura handiagoak interesatzen direnean erabiltzen dira. Lau funtzionamendu modu dituzte, eta, haiekin, hainbat gehiengo komunikazio-abiadura lor daitezke:

- *SPP* (Standard Parallel Port), segundoko 500 kilobyteko transmisio-abiadurarekin.
- *EPP* (Enhaced Parallel Port), modu hedatua, noranzko biko eta segundoko 2 megabyteko komunikazio-abiadurarekin.
- *ECP* (Extended Capatibilies Port), noranzko biko komunikazioarekin, baina DMArekin erabiltzen duen moduarekin; segundoko 2,4 megabyteko abiadura du, baina periferikoak modu hau jasateko baldintzarekin.
- *ECP+EPP*

Moduaren hautapena ordenagailuaren setup-arekin egiten da, eta, arazorik izanez gero, moduaren ahalmenetan gutxitzen joatea gomendatzen da.



7-22 irudia. Ataka paraleloaren DB-25 konektorea (goian), eta serieko atakaren DB-9-ak (behean)

7.5.3 Komunikazio-ataken erabilera

Ziur aski, uneren batean, PCan egikaritzen ari den programaren baten eta serieko eta/edo paraleloko interfazeren batez hornitua den beste sistema elektronikoren baten (makinaren bat, beste PCren bat, etab.) artean komunikazioa ezartzeko, PCaren porturen bat erabili beharra izango dugu. Kasu horretan, litekeena da programa berean ere kanpoko sistemarekin komunikatzeko aukeraren bat izatea, baina litekeena da, halaber, auskalo zein arrazoirengatik (adibidez: programa geuk egina delako), komunikazio-sistema horren aukerarik ez izatea.

Borland-en C hizkuntzan egindako programa batekin komunikazioa ezarri ahal izateko (gutxienez ere *Windows*-entzako *Borland C++ 3.1* programarekin), honako `dos.h` liburutegiko funtzio hauek daude:

```
int inport(int portid)  /*16 biteko word (hitz) bat irakurtzen du portid portutik*/
unsigned char inport(int portid) /* idem baina byte bat word baten ordez*/
void outport(int portid, int value)  /*value hitza(16bit) idazten du portid-en*/
void outport(int portid, unsigned char value)  /* byte bat idazten du portid-en*/
```

Horrela, AF byte (zortzikote) hamaseitarra serieko COM1 atakatik bidali nahi baldin badugu, ondorengo sententzia hau erabil daiteke:

```
void outport(0x3F8,0xAF);
```

Alabaina, *Borland C++ Builder Standard* bertsioko erremintek, eta dirudienez, baita ondorengo bertsioetakoek ere, ez dute portu bidezko horrelako komunikazio-funtziorik; beraz, irtenbideetako bat izango litzateke sententzia baliokideak Intel etxearen 80x86 familiako hizkuntza mihiztatzaileko aginduetan idaztea.

Horretarako, nahikoa da mihiztatzaile-hizkuntzako aginduak C++eko kodea erdian sartzea. Horretarako, `asm` gako-hitza baino ez da idatzi behar, eta, ondoren, aginduak giltza artean:

```
asm
  {
    mov ax,0x0e07
    xor  bx,bx
    int  0x10
  }
```

Ikusten denez, erreminta honek Cn edo C++en erazagututako aldagaietako batzuk erabiltzeko aukera ematen du.

Pentium μP-an S/I (I/O) esparruko atzipena baimentzen duten aginduak honako hauek dira:

```
IN AL,DX
IN AX,DX
IN EAX,DX
OUT DX,AL  // AL-tik 8 biteko datua DX-en adierazitako portura
OUT DX,AX
OUT DX,EAX
```

Bestalde, behar-beharrezkoa da datuak metagailu erregistroan/erregistrotik kargatzen/deskargatzen uzten duten gutxieneko aginduak ezagutzea:

```
MOV DATA,AL
MOV DATA,AX
MOV DATA,EAX
MOV AL,DATA
MOV AX,DATA
MOV EAX,DATA
```

Mihiztatzailean kodea erabiltzeak *warning: Restarting compile using assembly* eragina sortuko balu, (`.cpp`) iturburu-fitxategian ondorengo preprozesadorearen agindu hau sartu beharko litzateke:

```
#pragma inline –w-asc
```

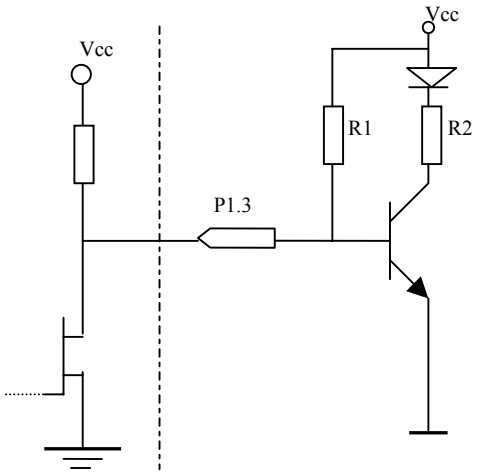
Badira, *LabVIEW*® edo *CVI LabWindows*® bezala, PCei konektatutako kontrol-softwareko eta gailu-monitorizazioko garapen-erremintak, komunikazio-portuen programazioa oso eroso eta modu grafikoan egiten zaienak.

8. ARIKETAK

8.1 Ariketak 8051 µC-arekin

1) 8051 µC-aren P1.3 pinean Led diodo bat konektatu nahi da softwaretik kontrolatua izateko, hau da, programatik piztua eta itzalia izan dadin.

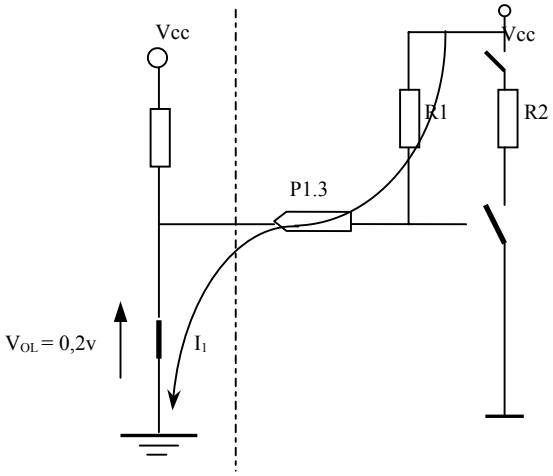
Ebazpena:
Led diodoak 20mA inguru kontsumitu behar ditu zuzenean polarizatuta dagoen bitartean (V_{AK}=0,7v) argia pitz dadin, baina korronte hori mikrokontrolatzaileko ezein pinek ere ezin dezake eman. Beraz, irtenbide bakarra aukeratutako pinak ematen duen korrontea anplifikatzea da.
Horretarako, polarizazio finkoko edo oinarrizko kommutazio-zirkuitua hautatzen da, eta, Q puntuan ezegonkortasuna agertzen duen arren, nahikoa da kommutazio-funtzionamendurako.



8-1 irudia. Zirkuituaren eskema

Azter ditzagun kommutazioaren bi kasuak:

Barrutik 0 idazten da, kanpoko transistorearen egoera ebakidurakoa da, diodoak itzalita jarraitzen du eta kanpoko Vcc-tik R1 bidez datorren korrontea pinak xurgatzen du, bere barne-transistorearen bitartez. R1 erresistentzia, pinaren transistorea ez erretzeko moduan kalkulatu behar da; izan ere, haren I_{OL} handiena=1,6mA da.

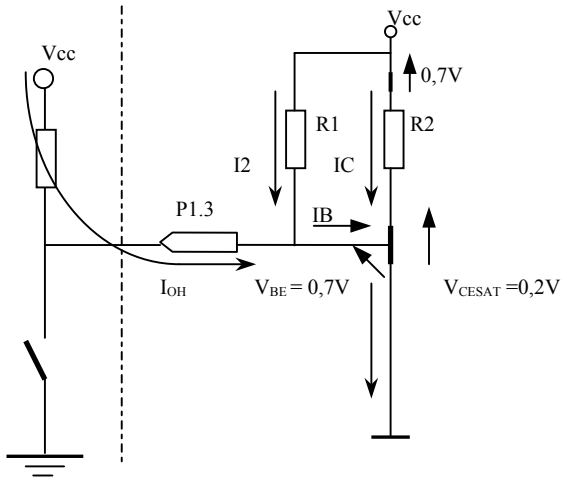


8-2 irudia. Pinean "0" idatzita

I_1=(5-0,2)/R1 <1,6 mA, I_1=1 mA suposatuz, → R_1=4,8 kohm ematen du erresistentziarentzako balio teorikoak. Merkatuko (benetako) hurbileko balio bat hartuta, I_1 berriro kalkulatuko dugu onartutako baliorik handiena ez dela gainditzen egiaztatzeko.

I_1=(5-0,2)/4,7 kohm=1,02 mA ongi dago; izan ere 1,6 mA baino txikiagoa edo berdina da.

Barruan 1 idazten baldin bada, ordea, kanpoko transistorearen egoera asetze-egoerakoa izatera pasatzen da, eta horrekin Led diodoa pizten da. Pinak emandako korrontea ezin liteke fabrikatzaileak emandako korronterik handiena baino handiagoa izan.



8-3 irudia. Pinean "1" idatzita

beraz, ekuazio hauek eratzen dira:

$I_B=I_{0H}+I2$

$I_B=I_{OH}+(5-0,7)/4,7$ ~ = 90μA+0,914=1mA

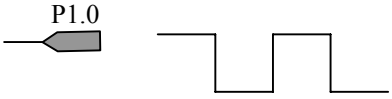
R_2=(5-0,7-0,2)/20 mA=205 ohm ematen du balio teorikoak. I_{diodoa} merkatuko 220 ohm balioarekin berriro kalkulatuta, → I_D=Icsat=18,63 mA ematen du, Led diodoa pizteko bestekoa, eta Icsat> 18,63 mA duen eta H_{FE}txik>Icsat/I_B baino =18,63/1=18,63 duen transistorea hautatuko da. Horrela, serie eta mota bereko transistore guztiak harrapatzen ditugu.

Beste aukera bat da Atmel etxeko 89 serieko μC-a (8051 bertsioak) erabiliz gauza bera egitea; izan ere, mikro horiek P1 eta P3 portuetako pin bakoitzak 20 mA-rainoko korronteak xurgatzeko aukera ematen dute (baina, betiere, fabrikatzailearen arabera, guztira 80 mA baino korronte handiagoa xurgatzen ez bada). Kasu horretan, pina zuzenean konektatzen zaio Led-ari erresistentzia korronte-mugatzaile batekin (20 mA-ko mugatzailea); orduan, pinean 0 idaztean, Led-a aktibatu egingo da, eta 1 idaztean, berriz, desaktibatu.

2) 8051 μC baten T0 tenporizadoreaz baliatuz eta P1.0 pina delarik seinale sortzailea, bozgorailu bati konektatzen zaio anplifikadore soil baten bitartez. Aipatutako bozgorailuan 1 kHz-eko seinalea sortzea da helburua. Kristalaren maiztasuna 4 MHz-ekoa da, eta bozgorailuan honela entzun behar da:

a) Etengabe
b) Lehen segundo laurdenean

Ebazpena:



8-4 irudia. Uhin karratuaren sorkuntza

a) T0 tenporizadoreak T/2 guztietan etetea eskatuko du, hots, (1/f)/2=(1/1000)/2 s = 500 µs

Hau kontuan hartuta:

1 cm=12*Tosc=12*1/fosc=12*1/4*10e6=3*10e-3 s=3 µs

orduan, 3 µs→1 mz baldin badira,
500 µs→ X mz izango dira,

X=500/3=166,66 mz ~ 167 mz izanda, hori da T0 tenporizadoreak zenbatu behar duena. Horretarako, zenbaketa-erregistroaren balioa kalkulatzen da (eta baita 2 moduarentzako birkarga egitekoa ere (8 bit)).

256-167=89

```
#include<reg51.h>
#define uchar unsigned char
sbit P10=P1^0;  /* P10 aldagaia sortzen da, P1 portuaren 0 pina bezala */
void HasiT0(void)
{
 TMOD&=0xF0;
 TMOD|=0x02;
 TL0=89;
 TH0=89;
 ET0=1;
 EA=1;
 TR0=1; /* T0 Timerra martxan hasten da */
}
void EtenT0(void) interrupt 1
{
 P10=~P10;
}

void main(void)
{
HasiT0();
while(1)
 ;
}
```

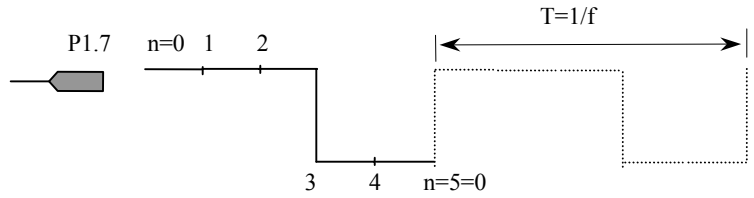
b) Bozgorailuak lehenengo segundo laurdenean (250 ms-an) bakarrik egin dezan soinua, etena zenbat aldiz eskatu behar duen kalkulatu behar da, eta azkeneko aldian tenporizadorea gelditu.

Aldi kopurua=250.000 µs/ 500 µs bakoitzean = 500 aldi

```
#include<reg51.h>
#define uchar unsigned char
sbit P10=P1^0;  /* P10 aldagaia sortzen da P1 portuaren 0 pina bezala */
void HasiT0(void)
{
 TMOD&=0xF0;
 TMOD|=0x02;
 TL0=89;
 TH0=89;
 ET0=1;
 EA=1;
 TR0=1; /* T0 Timerra martxan hasten da */
}
void EtenT0(void) interrupt 1
{
 static int aldiak;

 P10=~P10;
```

```
 aldiak++;
 if(aldiak ==500)
  {
   TR0=0; /*T0 tenporizadorea eteten du*/
   P10=0; /*transistorea beharrik gabe ez berotzeko*/
   aldiak=0; /* hurrengo aldirako hasieratzen du */
  }
}
void main(void)
{
HasiT0();
while(1)
 ;
}
```

3) 3 MHz-eko kristala duen 8051 µC-aren 1 portuko 7. pinean zehar 66,666 Hz-eko maiztasuneko uhin angeluzuzena modu konstantean sortzea nahi da, seinalearen goi-mailari seinalearen periodoaren % 60 dagokiola eta behe-mailari gainerako % 40 dagokiola kontuan hartuta. T0 tenporizadorea bakarrik erabilita, uhina marraztea, ariketa ebazteko beharrezkoak diren kalkuluak egitea eta `void InitT0(void)`, `void InterTimerT0(void) interrupt 1` eta `void main(void)` funtzioak idaztea eskatzen da.



Ebazpena:
Seinalearen periodoa honako hau da: T=1/f=1/66,666=0.015 s = 15000 µs eta T0 tenporizadoreak etenak T/5=15000/5=3000 µs-tik behin egitea eskatzen da.

Kristala 3 MHz-ekoa denez, Tmz=12*1/fxtal=12*1/3*10^6=4 µs, orduan:

4 µs→1 mz → x=750 mz
3000 µs→x mz

1 moduan, 16 bit, zenbaketa-erregistroak ondorengo balio hauekin kargatu behar dira:

65536-750=64786d=FD12H->TH0=FDH eta TL0=12H

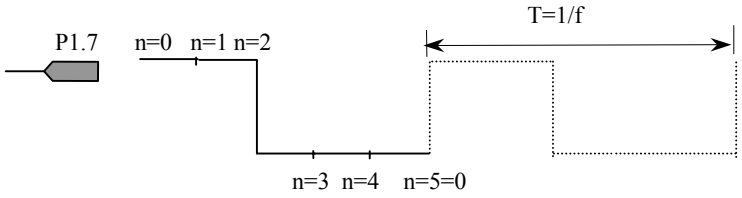
```
#include<reg51.h>
sbit P17=P1^7;
void InitT0(void)
{
 TMOD&=0xF0;
 TMOD|=0x01;
 TH0=0xFD;
 TL0=0x12;
 ET0=1;
 EA=1;
 TR0=1;
}
void InterTimerT0(void) interrupt 1
{
 static char n;

 TH0=0xFD;
 TL0=0x12;
```

```
 n++;
 if(n==3)
  P17=0;
 else if(n==5)
  {
   P17=1;
   n=0;
  }
}
void main(void)
{
 InitT0();
 while(1)
  ;
}
```

InterTimerT0( ) funtzioarentzako trinkoagoa izango litzateke soluzio hau:

```
void InterTimerT0(void) interrupt 1
{
 static char n;

 TH0=0xFD;
 TL0=0x12;
 n++;
 if((n==3)||(n==5))
  {
   P17=~P17;
   if(n==5)
    n=0;
  }
}
```

4) 12 MHz-eko kristala duen 8051 µC-aren 1 portuko 7. pinean zehar 100 Hz-eko maiztasuneko uhin angeluzuzena modu konstantean sortzea nahi da, seinalearen goi-mailari seinalearen periodoaren % 40 dagokiola eta behe-mailari gainerako % 60 dagokiola kontuan hartuta. T0 tenporizadorea bakarrik erabilita, uhina marraztea, ariketa ebazteko beharrezkoak diren kalkuluak egitea eta `void InitT0(void)`, `void InterTimerT0(void) interrupt 1` eta `void main(void)` funtzioak idaztea eskatzen da.



Ebazpena:
Seinalearen periodoa honako hau da: T=1/f=1/100=0.01 s = 10000 µs eta T0 tenporizadoreak etenak T/5=10000/5=2000 µs-tik behin egitea eskatzen da.

Kristala 12 MHz-ekoa denez, Tmz=12*1/fxtal=12*1/12*10^6=1 µs, orduan:

1 µs→1 mz → x=2000 mz
2000 µs→x mz

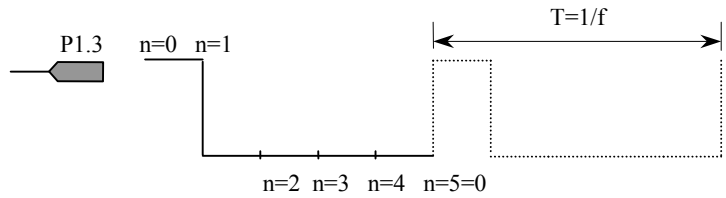
1 moduan, 16 bit, zenbaketa-erregistroak ondorengo balio hauekin kargatu behar dira:

65536-2000=63536d=F830H->TH0=F8H eta TL0=30H

```
#include<reg51.h>
sbit P17=P1^7;
void InitT0(void)
{
 TMOD&=0xF0;
 TMOD|=0x01;
 TH0=0xF8;
 TL0=0x30;
 ET0=1;
 EA=1;
 TR0=1;
}
void InterTimerT0(void) interrupt 1
{
 static char n;

 TH0=0xF8;
 TL0=0x30;
 n++;
 if(n==2)
  P17=0;
 else if(n==5)
  {
   P17=1;
   n=0;
  }
}
void main(void)
{
 InitT0();
 while(1)
  ;
}
```

5) 24 MHz-eko kristala duen 8051 µC-aren 1 portuko 3. pinean zehar 100 Hz-eko maiztasuneko uhin angeluzuzena modu konstantean sortzea nahi da, seinalearen goi-mailari seinalearen periodoaren % 20 dagokiola eta behe-mailari gainerako % 80 dagokiola kontuan hartuta. T0 tenporizadorea bakarrik erabilita, uhina marraztea, ariketa ebazteko beharrezkoak diren kalkuluak egitea eta `void InitT0(void)`, `void InterTimerT0(void) interrupt 1` eta `void main(void)` funtzioak idaztea eskatzen da.

Ebazpena:



Seinalearen periodoa honako hau da: T=1/f=1/100=0.01 s = 10000 µs eta T0 tenporizadoreak etenak T/5=10000/5=2000 µs-tik behin egitea eskatzen da.

Kristala 24 MHz-ekoa denez, Tmz=12*1/fxtal=12*1/24*10^6=0,5 µs, orduan:

0,5 µs→1 mz → x=4000 mz
2000 µs→x mz

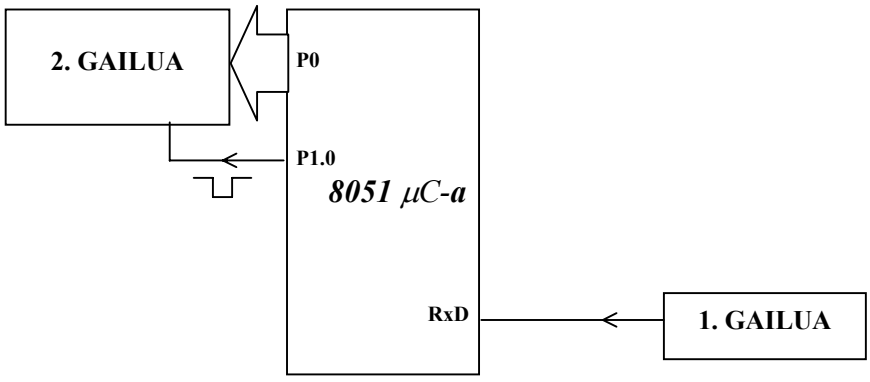
1 moduan, 16 bit, zenbaketa-erregistroak ondorengo balio hauekin kargatu behar dira:

65536-2000=63536d=F830H->TH0=F8H eta TL0=30H

```
#include<reg51.h>
sbit P13=P1^3;
void InitT0(void)
{
 TMOD&=0xF0;
 TMOD|=0x01;
 TH0=0xF0;
 TL0=0x60;
 ET0=1;
 EA=1;
 TR0=1;
}
void InterTimerT0(void) interrupt 1
{
 static char n;

 TH0=0xF0;
 TL0=0x60;
 n++;
 if(n==1)
  P13=0;
 if(n==5)
  {
   P13=1;
   n=0;
  }
}
void main(void)
{
 P13=1;  /* badaezpada P13 "1"-egaz hasteko jartzen da */
 InitT0();
 while(1)
  ;
}
```

6) 8051 µC-a dugu, eta paritate bakoitiarekin 236 bit/s abiaduraz datuak bidaltzen dizkion gailu bati seriean konektatuta dago. µC-ak hartutako datu bakoitza µC-ak bigarren gailu bati igortzen dio, paralelo bidez P0 atakaren bitartez, eta jakinarazi ere egiten dio beste sarrera baten bidez; P1 atakako (P1.0) 0 pinaren bidez, behe-mailako pultsu bat bidaliz. 8051 µC-ak duen kristalaren maiztasuna 11,0592 MHz-ekoa da.
Marraztu ariketaren eskema, SFRak beren balioekin, eta idatzi egindako kalkuluak eta `void InitUART(void)`, `void Hartudatua(void) interrupt 4` eta `void main(void)` funtzioak.

Ebazpena:



Serieko komunikazioa parekotasun bakoitiarekin eta Vcom aldakorrarekin egiten denez, 3 modua hautatu da (9 biteko UARTa Vcom aldakorrarekin), eta T1 tenporizadorearentzako zenbaketa-balioa kalkulatzen da ondorengo formula honen bitartez:

V_{COM}=fosc *2 ^{SMOD}/12*(256-TH1)*32, fxtal 11,0592 MHz denez eta SMOD=0 hartuta, →TH1=TL1=134d

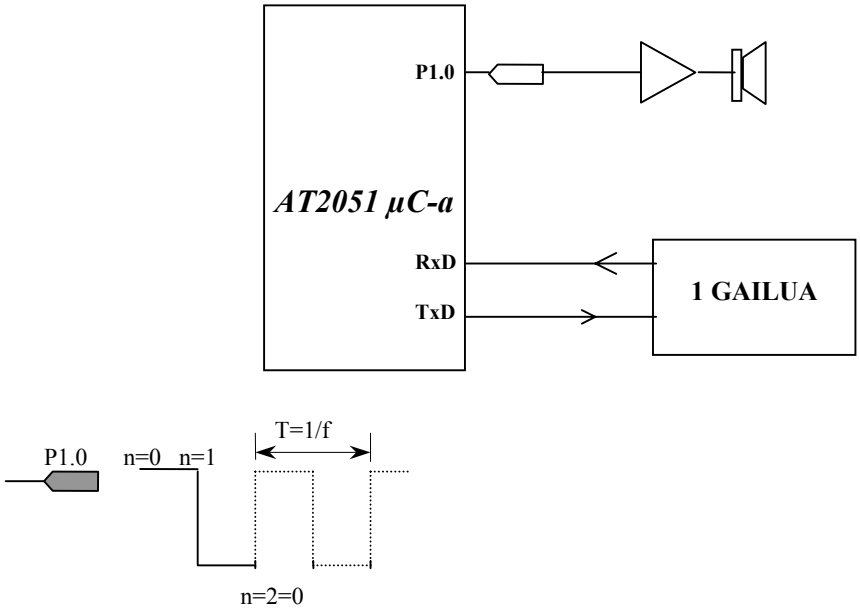
```
#include<reg51.h>
sbit P10=P1^0;
void InitUART(void)
{
 SCON=0xD0;  /*Datu-harrera gaitua eta 3 modua Vcom aldakorrarekin */
 PCON&=0x7F;
 PCON|=0x00; /*SMOD=0*/
 TMOD&=0x0F;
 TMOD|=0x20;
 TL1=134;    /* 11,0592 MHz-eko fxtal-arentzat -->Vcom=236 bit/s */
 TH1=134;
 ES=1;
 EA=1;
 TR1=1;
}
void Hartudatua(void) interrupt 4
{
 char datua;

 if(RI==1)
 {
  RI=0;
  datua=SBUF;
  ACC=datua;
  if(RB8=~P)  /* harrera-paritate bakoitiarekin */
  {
   P0=datua;  /* paraleloko transmisioa */
   P10=0;     /* 2.gailuak P0-tik datua irakur dezan behe-mailako pultsua */
   P10=1;     /* sortzen da */

  }
}
void main(void)
{
 P10=1;       /* badaezpada P10 "1"-egaz hasteko jartzen da */
 InitUART();
 while(1)
  ;
}
```

7) Atmel-en AT892051 µC-an oinarritutako sistema bat dugu (T0 eta T1 tenporizadoreak, UART serieko ataka, S/I-ko (I/O) 15 pin P1 eta P3 portuetan) 6 MHz-eko kristal-maiztasunarekin, eta UART serieko portutik zortzikoteak hartzen dira era konstantean eta modu asinkronoan paritaterik gabe, segundoko 5208 biteko abiaduran. Sistemak 33,33 Hz-eko seinalea ("beep") igortzeko gai izan behar du, eta UART bidez datu zehatza bidali, taula honek azaltzen duen moduan:

Hartu	Bidali	Seinalearen ("beep") iraupena
AAH	FFH	segundo erdia
BBH	EEH	segundo seirena
Gainerakoa	Batere ez	Batere ez

Seinalea ("beep") bozgorailu txiki baten bitartez jasoko da P1 portuko 0 pinak eskainitako seinale zehazki karratu batetik abiatuta. Marraztu ariketaren eskema, SFRak beren balioekin, eta idatzi egindako kalkuluak eta funtzio hauek: `void InitUART(void), void InitT0T1(void), void InterT0(void) interrupt 1, void Hartudatua(void) interrupt 4` eta `void main(void).`

Ebazpena:



Seinalearen periodoa: T=1/f=1/33,33=0.03 s = 30000 µs-koa da, eta T0 tenporizadoreak T/2=30000/2=15000µs-etik behin etetea erabakitzen da.
Kristala 6 MHz-ekoa denez, Tmz=12*1/fxtal=12*1/6*10^6=2 µs, eta, orduan:

2 µs→1 mz → x=7500 mz
15000 µs→x mz

1 moduan, 16 bit, zenbaketa-erregistroak honako balio hauekin kargatu behar dira:

65536-7500=58036d=E2B4H->TH0=E2H y TL0=B4H

segundo erdia 500.000 µs dira, eta, hortik, 500000/15000=33,33~33 etenaldi T0 tenporizadorearentzat.
eta segundo seirena 166.666,66 µs dira, eta, hortik, 166666,66/15000=11.11~11 etenaldi ateratzen dira T0 tenporizadorearentzat.

Serieko komunikazioa paritaterik gabe eta Vcom aldakorrarekin egiten denez, 1 modua hautatu da (8 biteko UARTa Vcom aldakorrarekin), eta T1 tenporizadorearen zenbaketa-balioa ondorengo formula honen bidez kalkulatzen da:

V_{COM}=fosc *2 ^{SMOD}/12*(256-TH1)*32, eta fxtal 6 MHz delako eta SMOD=1 hartuta.
→TH1=TL1=250d=FAH

```
#include<reg51.h>
#define uchar unsigned char
uchar hardat;
sbit P10=P1^0;
```

```
void InitT0T1(void)
{
 TMOD=0x21;
 TL0=0xB4;
 TH0=0xE2;
 TL1=0xFA;    /* 6 MHz-eko fxtal-arentzat -->Vcom=5208 bit/s, SMOD=0-kin */
 TH1=0xFA;
 ET0=1;
 ES=1;
 EA=1;
}
void InitUART(void)
{
 SCON=0x70;  /*UART 1 modua, harrera gaitua eta nahitaezkoa stopb==1 */
 PCON&=0x7F;
 PCON|=0x80; /*SMOD=1*/
 TR1=1;
}
void InterUART(void) interrupt 4
{
 if(RI==1)
 {
  RI=0;
  hardat=SBUF;
  if(hardat==0xAA)
   SBUF=0xFF;
  else if(hardat==0xBB)
   SBUF=0xEE;
 }
 else /* transmititutako datua osorik portutik irtetean: TI=1 */
 {
  TI=0;
  TR0=1;
 }
}
void InterT0(void) interrupt 1
{
 static uchar aldiak;

 P10=~P10;
 TL0=0xB4;
 TH0=0xE2;
 if((hardat==0xAA)&&(aldiak<33))
  aldiak++;
 else if((hardat==0xBB)&&(aldiak<11))
  aldiak++;
 else
  {
   TR0=0;
   aldiak=0;
  }
}
void main(void)
{
 InitT0T1();
 InitUART();
 while(1)
  ;
}

/*************************************************************************/
/* 2. Bertsioa: SMOD=0 eta beep-aren sorkuntza Hartudatua( ) funtziotik */
/*************************************************************************/

#include<reg51.h>
```

```
char aldiak;
sbit P10=P1^0;

void InitT0T1(void)
{
 TMOD=0x21;
 TL0=0xB4;
 TH0=0xE2;
 TL1=250;    /* 6 MHz-eko fxtal-arentzat -->Vcom=5208 bit/s, SMOD=0-kin */
 TH1=250;
 PT0=1;
 ET0=1;
 EA=1;
 TR1=1;
}
void InitUART(void)
{
 SCON=0x70;  /* UART 1 modua, harrera gaitua eta nahitaezkoa stopb==1 */
 PCON&=0x7F; /* SMOD=0 */
 PS=0;
 ES=1;
}
void HartuDatua(void) interrupt 4
{
 char datua;

 if(RI==1)
 {
  RI=0;
  datua=SBUF;
  if(datua==0x66)
   {
    SBUF=0xFF;
    while(TI==0)
     ;
    TI=0;
    TR0=1;
    while(aldiak<33) /* segundo erdiko beep-a sortu bitartean hemen itxaroten du */
     ;
    TR0=0;
    aldiak=0;
   }
  if(datua==0x65)
   {
    SBUF=0xEE;
    while(TI==0)
     ;
    TI=0;
    TR0=1;
    while(aldiak<11) // segundo seireneko beep-a sortu bitartean hemen itxaroten du
     ;
    TR0=0;
    aldiak=0;
   }
 }
}
void InterT0(void) interrupt 1
{
 aldiak ++;
 P10=~P10;
 TL0=0xB4;
 TH0=0xE2;
}
void main(void)
{
```

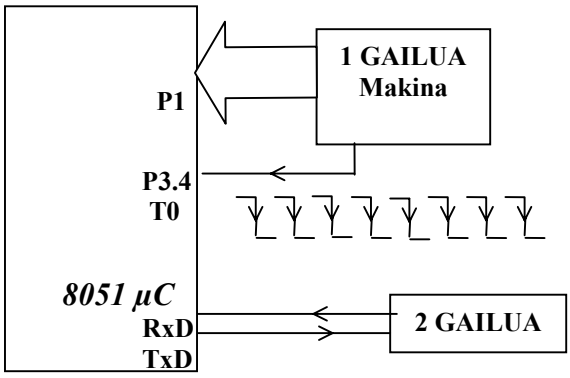
```
 InitT0T1();
 InitUART();
 while(1)
  ;
}
```

8) 6 MHz-eko kristala duen 8051 µC bat makina bati konektatua dugu P1 portuaren bidez. Horrela, makinak, 8051-ri daturen bat bidaltzera doanean, 8 jaitsiera-saihets ezartzen ditu T0 kontagailuaren pinean, eta µc-ak hartzen duen datu bakoitza, serie bidez paritate bakoitiarekin eta segundoko 220 biteko abiaduraz bidaltzen dio bigarren gailu bati.
Marraztu ariketaren eskema, SFRak beren balioekin, eta idatzi eginiko kalkuluak eta ondorengo funtzioak:

```
void InitT0UART(void)
void InterT0(void) interrupt 1
void DatuBidal(void)  (flag-a kontsultatuz)
void main(void)
```

Ebazpena:
Serieko transmisioa paritate bikoitiarekin eta Vcom aldakorrarekin egiten denez, UARTaren 3 modua sailkatzen da (9 biteko eta Vcom aldakordun UARTa) eta T1 timerrarentzako zenbaketa-balioa kalkulatzen da, honako formula honen bidez:

V_{COM}=fosc *2 ^{SMOD}/12*(256-TH1)*32, fxtal 6 MHz-ekoa denez eta SMOD=0 hartuz →TH1=TL1=114d



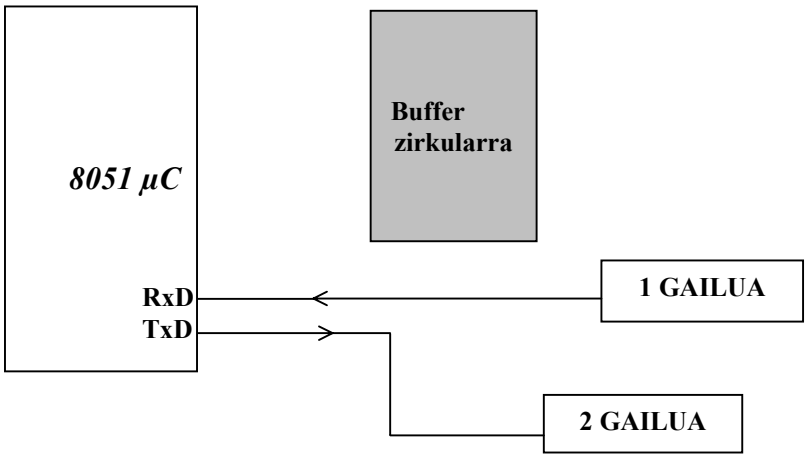

```
#include<reg51.h>

void InitT0UART(void)
{
 SCON=0xC0;  /*UART 3 modua eta transmisioa bakarrik */
 PCON&=0x7F;
 PCON|=0x80; /*SMOD=1*/
 TMOD=0x26;
 TL0=248;
 TH0=248;
 TL1=114;    /* 6 MHz-eko fxtal-a -->Vcom=220 bit/s */
 TH1=114;
 ET0=1;
 EA=1;
 TR0=1;
 TR1=1;
}
void DatuBidal(void)
{
 ACC=P1;
```

```
 TB8=~P;  /* paritate bakoitiarekin bidaltzeko */
 SBUF=P1;
 while(TI==0)
  ;
 TI=0;
}
void InterT0(void) interrupt 1
{
 DatuBidal();
}
void main(void)
{
 InitT0UART();
 while(1)
  ;
}
```

9) 11,0592 MHz-eko kristala eta 4 kilobyteko (4096 byte) datuen kanpo-memoria duen 8051 µC bat kanpoko periferiko biri konektatua dago serie atakaren bidez, paritaterik gabe eta segundoko 120 biteko abiaduraz hasieratua. 1 GAILUA da µC-ari datuak bidaltzen dizkiona, eta horrek kanpo-memorian gordetzen ditu, buffer zirkular gisa erabilian. 2 GAILUAK bufferrean gorderiko datuak hartuko ditu, baina bete den kopurua bidalitakoa baino txikiago izan bada bakarrik.
Marraztu ariketaren eskema, SFRak beren balioekin, eta idatzi eginiko kalkuluak eta ondorengo funtzioak:

```
      void InitUART(void)
      void DatuHartu(void) interrupt 4
      void DatuBidal(void) (flag-a kontsultatuz)
      void main(void)
```

Ebazpena:




```
#include<reg51.h>
char xdata *punR=0x0000;
char xdata *punT=0x0000;
char data beterik;
char data bidalia;
void InitUART(void)
{
 SCON=0x50; /* UART 1 modua, 8 bit, Vcom=120 bit/s 11,0592 MHz-ekin  */
 ES=1;
 EA=1;
 PCON&=0x7F;
 TMOD&=0x0F;
 TMOD|=0x20;
 TL1=16;
 TH1=16;
 TR1=1;
```

```
}
void DatuBidal(void)
{
 SBUF=*punT;
 while(TI==0)
  ;
 TI=0;
 punT++;
 if(punT==0x1000)
      {
       punT=0x0000;
       bidalia++;
      }
}
void DatuHartu(void) interrupt 4
{

 if(RI==1)
  {
   RI=0;
   *punR=SBUF;
   punR++;
   if(punR==0x1000)
    {
     punR=0x0000;
     beterik++;
    }
  }
 else  /* TI==1 kasuarentzako */
  ;
}
void main(void)
{
 InitUART();
 while(1)
  while(beterik>bidalia)
   DatuBidal();
}

/*****************************************************************************/
/* 2. bertsioa: paritate bikoitia erabiliz serie-komunikazioan      */
/*****************************************************************************/
#include<reg51.h>
char xdata *punR=0x0000;
char xdata *punT=0x0000;
char data beterik;
char data bidalia;
void InitUART(void)
{
 SCON=0xD0; /* UART 3 modua, 9 bit, Vcom=120 bit/s 11,0592 MHz-ekin */
 ES=1;
 EA=1;
 PCON&=0x7F;
 PCON|=0x00; /* SMOD=0 */
 TMOD&=0x0F;
 TMOD|=0x20;
 TL1=16;
 TH1=16;
 TR1=1;
}
void DatuBidal(void)
{
 ACC=*punT;
 TB8=P;
 SBUF=*punT;
```

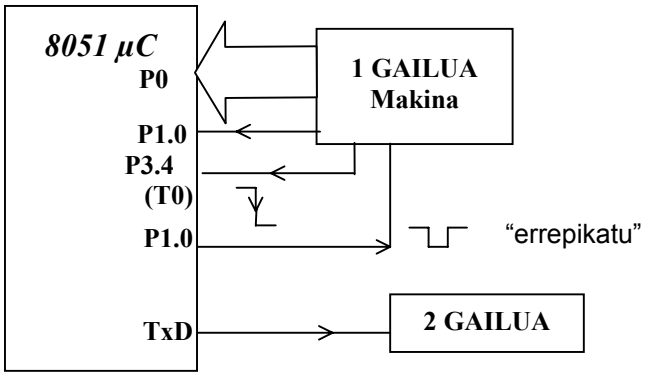
```
 while(TI==0)
  ;
 TI=0;
 punT++;
 if(punT==0x1000)
       {
        punT=0x0000;
        bidalia++;
       }
}
void DatuHartu(void) interrupt 4
{

 if(RI==1)
  {
   RI=0;
   ACC=SBUF;
   if(RB8==P) //datua ongi badator gordetzen da soilik, bestela ezer ez
    {
     *punR=SBUF;
     punR++;
     if(punR==0x1000)
      {
       punR=0x0000;
       beterik++;
      }
    }
  }
 else  /* TI==1 kasuarentzako*/
  ;
}
void main(void)
{
 InitUART();
 while(1)
  while(beterik>bidalia)
   DatuBidal();
}
```

10) 6 MHz-eko kristala duen 8051 μC bat makina bati konektaturik daukagu, P0 portuaren eta P1.0 pinaren bidez, non 8051-ak byteak paritate bakoitiarekin hartzen baititu. Bada beste pin bat ere, P3.4 edo T0 kontagailu bezala dabilenarena, non jaitsiera-saihets bat ezarriz, makinak adierazten baitio orduan emango diola datua. Datu hori ez badator paritate bitarekin bat, orduan, μC-ak behe-mailako pultsu batez erantzungo dio makinari, P1.1 pinaren bidez, azkeneko datua berriro bidaltzeko eskatuz. μC-ak hartzen duen datu zuzen bakoitza serie atakaren bidez paritate bikoitiarekin eta segundoko 220 biteko abiadurarekin loturik dagoen beste gailu bati bidaltzen dio.

Marraztu ariketaren eskema, SFRak beren balioekin, eta idatzi egindako kalkuluak. Horrez gain, `void InitUART(void)`, `void InterT0(void) interrupt 1`, `void DatuBidal(uchar byte)` (flag-a kontsultatuz) eta `void main(void)` funtzioak idatzi.

Ebazpena:
Serieko transmisioa paritate bikoitiarekin eta Vcom aldakorrarekin egiten denez, UARTaren 3 modua sailkatzen da (9 biteko eta Vcom aldakordun UARTa) eta T1 timerrarentzako zenbaketa-balioa kalkulatzen da, honako formula honen bidez:

V_{COM}=fosc *2 ^{SMOD}/12*(256-TH1)*32, fxtal 6 MHz-ekoa denez eta SMOD=0 hartuz →TH1=TL1=114



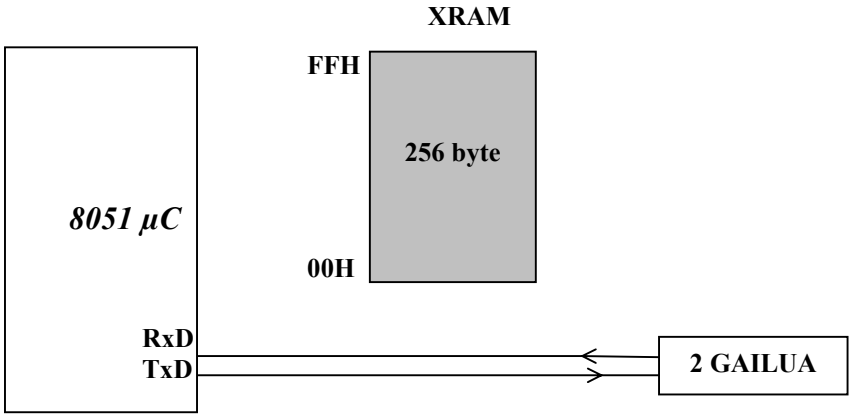

```
#include<reg51.h>
sbit P10=P1^0;
sbit P11=P1^1;
void InitT0UART(void)
{
 SCON=0xC0;  /* UART 3 moduan bidaltzen bakarrik */
 PCON&=0x7F;
 PCON|=0x80; /* SMOD=1*/
 TMOD=0x26;
 TL0=255;
 TH0=255;
 TL1=114;    /* 6 MHz-eko fxtal-a -->Vcom=220 bit/s */
 TH1=114;
 ET0=1;
 EA=1;
 TR0=1;
 TR1=1;
}
void DatuBidal(void)
{
 TB8=~P10;   /* paritate bikoitiarekin bidaltzeko */
 SBUF=P0;
 while(TI==0)
  ;
 TI=0;
}
void InterT0(void) interrupt 1
{
 ACC=P0;
 if(P!=P10)
  DatuBidal();
 else
  {
   P11=0;
   P11=1;
  }
}
void main(void)
{
 InitT0UART();
 P10=1;
 P11=1;
 while(1)
  ;
}
```

11) 11,0592 MHz-eko kristala eta 256 byte daturen kanpo-memoria dituen 8051µC bat dugu UART serie bidez segundoko 120 biteko eta paritaterik gabe kanpoko gailu bati konektaturik. Gailu horrek

banan-banan bidaltzen dizkio datuak μC-ari, eta horrek kanpo-memoriako datuekin konparatzen ditu eta 3 byte bidaliz erantzuten dio gailuari, non lehenengoak memorian horretan datu hori baino handiagoen kopurua adierazten baitu, bigarrenak balio berekoen kopurua, eta hirugarrenak, txikiagoen kopurua.

Marraztu ariketaren eskema, SFRak beren balioekin, eta idatzi egindako kalkuluak. Horrez gain, `void InitUART(void)`, `void DatuHartu(void) interrupt 4`, `void DatuBidal(uchar byte)` (flag-a kontsultatuz) eta `void main(void)` funtzioak idatzi.

Ebazpena:



Serieko transmisioa paritaterik gabe eta Vcom aldakorrarekin egiten denez, UARTaren 1 modua sailkatzen da (8 biteko eta Vcom aldakordun UARTa) eta T1 timerrarentzako zenbaketa-balioa kalkulatzen da, honako formula honen bidez:

V_{COM}=fosc *2 ^{SMOD}/12*(256-TH1)*32, fxtal 11,0592 MHz-ekoa denez eta SMOD=0 hartuz
→TH1=TL1=16

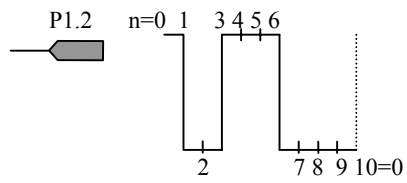
```
#include<reg51.h>
#define uchar unsigned char
void InitUART(void)
{
 TMOD&=0x0F;
 TMOD|=0x20;
 TL1=16;
 TH1=16;
 PCON&=0x7F; /* SMOD=0 */
 SCON=0x50;
 ES=1;
 EA=1;
 TR1=1;
}
void LoadXRAM(void)
{
 uchar xdata *pram,num=0xAA;

 for(pram=0x00FF;(pram<=0x00FF)&&(pram>=0x0000);pram--)
   {
    num++;
    num<<=1;
    *pram=num;
   }
}
```

```
void DatuBidal(uchar byte)
{
 SBUF=byte;
 while(TI==0)
  ;
 TI=0;
}
void DatuHartu(void) interrupt 4
{
 uchar datu,berdin=0,txiki=0,handi=0;
 uchar xdata *pdat;

 if(RI==1)
  {
   RI=0;
   datu=SBUF;
   for(pdat=0x00FF;(pdat<=0x00FF)&&(pdat>=0x0000);pdat--)
    {
     if(*pdat==datu)
      berdin++;
     else if(*pdat>datu)
      handi++;
     else
      txiki++;
    }
   DatuBidal(berdin);
   DatuBidal(handi);
   DatuBidal(txiki);
  }
}
void main(void)
{
 LoadXRAM();
 InitUART();
 while(1)
  ;
}
```

12) 3 MHz-eko kristala duen 8051 µC baten P1 portuaren 2. pinean zehar 200 Hz-eko maiztasuneko uhin angeluzuzena sortu nahi da. Lehenengo goi-mailari periodoaren hamarren bat, lehenengo behe-mailari bi hamarren, bigarren goi-mailari hurrengo 3 hamarren eta bigarren behe-mailari periodoaren azken 4 hamarrenak. Erabili T0 1 edo 2 moduan.
Marraztu uhina programaren egikaritzearen hasieratik (pare bat ziklo), eta adierazi programan erabilitako aldagaien balioak eta denboren balioak ere, µs-tan, programa osatzen duten ondorengo funtzio hauetaz gain:

```
void InitT0(void)
void InterT0(void) interrupt 1
void main(void)
```

Ebazpena:



Uhinaren periodoa T=1/f=1/200=0.005s=5ms=5000µs da. Orduan erabakitzen da, T0 timerrak T/10=5000/10=500µs-tik behin eska dezan etendura.

Kristala 3 MHz-ekoa denez, Tmz=12*1/fxtal=12*1/4*10^6=4µs, horrela:

4µs→1 mz → x=125 mz
500µs→x mz

2 modua, 8 bit, zenbaketa-erregistroak balio honekin kargatu behar dira:

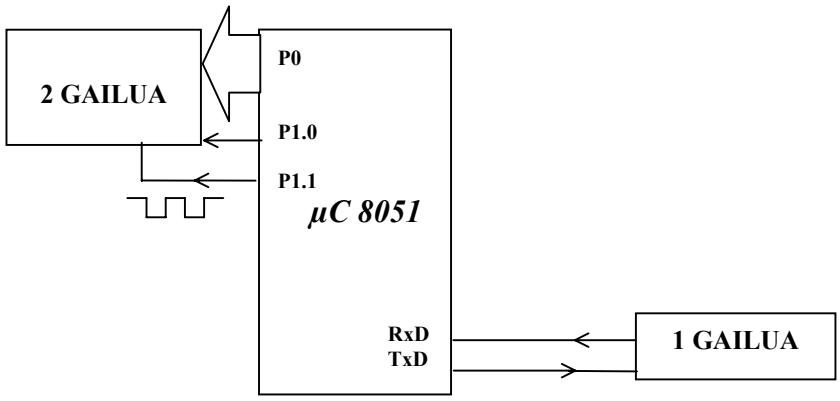
256-125=131d->TH0 y TL0

```
#include<reg51.h>
sbit P12=P1^2;
void InitT0(void)
{
 TMOD&=0xF0;
 TMOD|=0x02;
 TH0=131;
 TL0=131;
 ET0=1;
 EA=1;
 TR0=1;
}
void InterTimerT0(void) interrupt 1
{
 static char n;

 n++;
 if((n==1)||(n==6))
  P12=0;
 else if((n==3)||(n==10))
 {
  P12=1;
  if(n==10)
    n=0;
 }
}
void main(void)
{
 InitT0();
 P12=1;
 while(1)
  ;
}
```

13) 8051 µC-ari serie bidez loturik dagoen gailu (1) batek paritate bikoitiarekin eta segundoko 122 biteko abiadurarekin bidaltzen dizkio datuak. Harturiko datu bakoitzeko, µC-ak bigarren gailu (2) bati bidaltzen dio P0 paralelo portuaren bidez, eta paritate bita, oraingoan bakoitia, P1.0 pinetik bidaltzen dio. Baina hori guztia µC-ak harturiko datuak paritate bita ondo badauka bakarrik. Kasu horietan, µC-ak P1.1 pinean bi behe-mailako pultsu jarriz adierazten dio datu-bidalketa 2. gailuari: lehenak, datua jartzera doala adieraziz P1-ean, eta bigarrenak, jarri duela. Gainera, µC-ak erantzuna bidaltzen dio 1. gailuari: CCH, harturiko datuak ondo badu paritate bita, edo EEH, txarto badu.
8051-aren kristalaren maiztasuna 7,0272 MHz-ekoa da.

Marraztu ariketaren eskema, SFR-ak beren balioekin, idatzi egindako kalkuluak eta `void InitUART(void)`, `void DatuHartu(void) interrupt 4` (serieko harrera eta transmisioarentzako) eta `void main(void)` funtzioak.

Ebazpena:



Serieko transmisioa paritatearekin (bikoitia) eta Vcom aldakorrarekin egiten denez, UARTaren 3 modua sailkatzen da (9 biteko eta Vcom aldakordun UARTa) eta T1 timerrarentzako zenbaketa balioa kalkulatzen da, honako formula honen bidez:

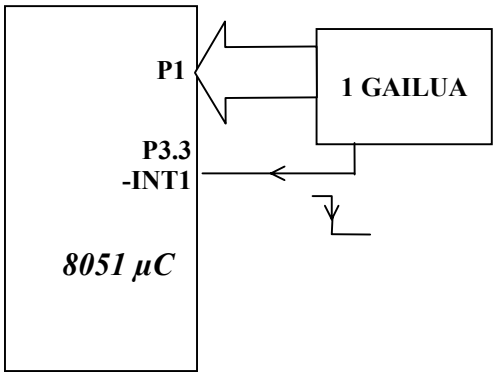
V_{COM}=fosc *2 ^{SMOD}/12*(256-TH1)*32, fxtal 7,0272 MHz-ekoa denez eta SMOD=0 hartuz
→TH1=TL1=106

```
#include<reg51.h>
sbit P10=P1^0;
sbit P11=P1^1;
void InitUART(void)
{
 SCON=0xD0;  /*Harrera gaitua eta 3 modua Vcom aldakorrarekin */
 PCON&=0x7F;
 PCON|=0x00; /*SMOD=0*/
 TMOD&=0x0F;
 TMOD|=0x20;
 TL1=106;    /* 7,0272  MHz-eko fxtal-arentzat -->Vcom=122 bit/s */
 TH1=106;
 ES=1;
 EA=1;
 TR1=1;
}
void Recibedato(void) interrupt 4
{
 char dato;
 bit pari;

 if(RI==1)
 {
  RI=0;
  dato=SBUF;
  ACC=dato;
  pari=P;
  if(RB8==pari) /* /* harrera-paritate bikoitiarekin */
  {
   TB8=0;     /* CCHk batekoen kopuru bikoitia du, ez du ACCtik pasatu behar */
   SBUF=0xCC;
   P11=0;     /* behe-mailako 1. pultsua */
   P11=1;
   P0=dato;   /* paralelo transmisioa */
   P10=~pari;
   P11=0;     /* 2. pultsua sortzen da 2 GAILUAK P0tik datua irakur dezan  */
   P11=1;
```

```
  }
  else
  {
   TB8=0;      /* EEHk batekoen kopuru bikoitia duenez */
   SBUF=0xEE;
  }
 }
 else /* TI==1 bada. Hau EEH edo CCH  transmititzen denerako da: main-era joan */
  TI=0;  /* eta berriro puntu honetara dator, TI==1 baita. */
*/
}
void main(void)
{
 P11=1;       /* badaezpada: P11-ak "1" mailaz hasten duela ziurtatzen du */
 InitUART();
 while(1)
  ;
}
```

14) 8051 µC bat dugu, eta paritaterik gabe bidaltzen dizkio datuak (gailuak µC-ari) gailu bati paraleloz konektaturik. Datu horiek P1 portuaren bidez bidaltzen dira, eta, izatez, gure egutegiko 3 urte adierazten duten 3 osoko dira (2 byte/int). Urte horiek `year` deituriko 3 elementu osoko array orokor batean gordeko dira. µC-a denbora guztian gailuaren zain egon ez dadin, 1 kanpo-etendura esleitzen zaio, jaitsiera-saihetsez eskaturik. Gailuak hurrengo bidaliko dituen urteak aurrekoen gainean idatziko dira. Sistemaren eskema marraztea eta `void InitIEX1(void ), void InterIEX1(void) interrupt 2` eta `void main(void)` funtzioak idaztea eskatzen da.

Ebazpena:




```
#include<reg51.h>
#define uchar unsigned char

int year[3];
void InitIEX1(void)
{
 IT1=1;
 EX1=1;
 EA=1;
}
void InterIEX1(void) interrupt 2
{
 static uchar n,m;

 if(m<3)
 {
```

```
  if(n%2==0)
   year[m]=P1;
  else
  {
   year[m]<<=8;
   year[m]|=P1;
   m++;
   if(m==3)
    m=0;
  }
  n++;
 }
}
void main(void)
{
InitIEX1();
while(1)
 ;
}
```

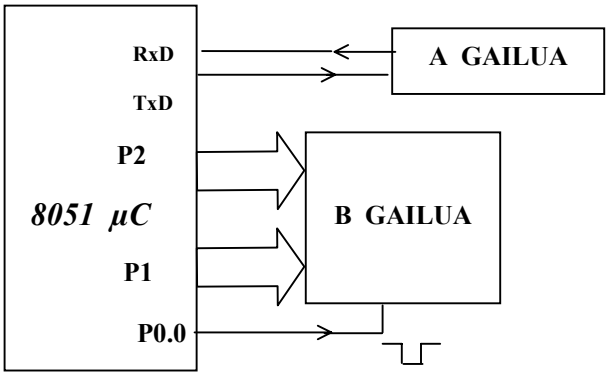
15) Idatzi byte bat pasatu eta haren paritate bita (bikoitia) itzultzen duen funtzioa (0 edo 1): `bit PariBik(uchar byte)`. Idatzi, baita ere, ondorengo iturburu-kode guztia, funtzioaren funtzionamendua ziurtatzeko

```
#include<reg51.h>
#define uchar unsigned char
sbit P10=P1^0;
bit PariBik(uchar byte)
{
 char ema,n,nbit=0;
 bit paritate;

 for(n=1;n!=0;n<<=1)
  {
   ema=byte & n;
   if(ema!=0)
    nbit++;
  }
 if(nbit%2==0)
  paritate=0;
 else
  paritate=1;
 return paritate;
}
void main(void)
{
 char datu=0x34;
 bit paritatea;

 paritatea=PariBik(dato);
 while(1)
  ;
}
```

16) 8051µC bat serie bidez gailu (A) bati loturik dugu, eta horrek µC-ari paritate bakoitiarekin eta segundoko 260,41 biteko abiadurarekin bidaltzen dizkio datuak. µC-ak hartzen duen osoko datu bakoitzeko, bigarren gailu (B) bati bidaltzen dizkio, P1 (behealdea) eta P2 (goialdea) portuen bidez, P0 portuaren 0 pinean (P0.0) behe-mailako pultsua ezarriz han. Lehen gailuak (A) osoko bat bidaltzen dio byte biko, baina byte bakoitzeko paritate bita zuzena bada bakarrik. Orduan, µC-ak CCHarekin erantzuten dio, edo, bestela, EEHarekin (harrerako paritate berberarekin). 8051 horren kristala 5 MHz-ekoa da.

Marraztu ezazu ariketaren eskema, SFRak haien balioekin, eta idatzi eman beharreko kalkuluak eta `void InitUART(void), void DatuHartu(void) interrupt 4` (serie bidezko harrera eta transmisioa) eta `void main(void)` funtzioak.

Ebazpena:



Serieko transmisioa paritatearekin (bakoitia) eta Vcom aldakorrarekin egiten denez, UARTaren 3 modua sailkatzen da (9 biteko eta Vcom aldakordun UARTa), eta T1 timerrarentzako zenbaketa-balioa kalkulatzen da, honako formula honen bidez:

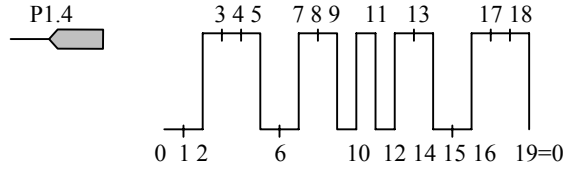
V_{COM}=fosc *2 ^{SMOD}/12*(256-TH1)*32, fxtal 5MHz-ekoa denez eta SMOD=0 hartuz →TH1=TL1=206

```
#include<reg51.h>
sbit P00=P0^0;
void InitUART(void)
{
 SCON=0xD0;  /*Harrera gaitua eta 3 modua Vcom aldakorrarekin */
 PCON&=0x7F; /*SMOD=0*/
 TMOD&=0x0F;
 TMOD|=0x20;
 TL1=206;    /* 5 MHz fxtal-arentzat -->Vcom=260,41 bit/s */
 TH1=206;
 ES=1;
 EA=1;
 TR1=1;
}
void DatuHartu(void) interrupt 4
{
 char byte;
 bit pari;
 static bit aldiak;

 if(RI==1)
 {
  RI=0;
  byte=SBUF;
  ACC=byte;
  pari=~P;
  if(RB8==pari) /* harrera-paritate bakoitiarekin */
  {
   TB8=1;    /* CCHk batekoen kopuru bikoitia  du, ez du ACCtik pasatu behar */
   SBUF=0xCC;
   if(aldiak==0)
    {
     P2=byte;
     veces=1;
    }
   else
    {
```

```
     P1= byte;
     P00=0;        /*  behe-mailako pultsua   */
     P00=1;
     aldiak =0;
    }
  }
  else
  {
   TB8=1;    /* CCHk batekoen kopuru bikoitia  du, ez du ACCtik pasatu behar */
   SBUF=0xEE;
  }
 }
 else /* TI==1 bada, EEH edo CCH transmititzen denerako: main-era doa eta  */
  TI=0;  /* berriro puntu honetara dator, TI==1 baita. */
}
void main(void)
{
 P00=1;       /* badaezpada: P10-ak "1" mailaz hasten duela ziurtatzen du */
 InitUART();
 while(1)
  ;
}
```

17) 3MHz-eko kristala duen 8051 µC baten P1 portuaren 4. pinean zehar 526,315 Hz-eko maiztasuneko irudian agertzen den pultsu multzoaren 5 periodo sortu nahi dira, 0 kanpo-etenduraren pinean jaitsiera-saihets bat ezartzen den bakoitzean. P1.4 pina goi-mailako egoeran egongo da seinalerik sortzen ez den artean. Marraztu uhina programaren egikaritzearen hasieratik (pare bat ziklo), eta adierazi programan erabilitako aldagaien balioak, programa osatzen duten ondorengo funtzio hauetaz gain: `void InitT0(void), void InterT0(void) interrupt 1, void InteIEX0 interrupt 0` eta `main()`.

Ebazpena:



Uhinaren periodoa T=1/f=1/526,315=0.019s=1,9 ms da, orduan erabakitzen da T0 timerrak T/19=0,1 ms=100 µs-tik behin eska dezan etendura.

Kristala 3MHz-ekoa denez, Tmz=12*1/fxtal=12*1/4*10^6=3µs, horrela:

4µs→1 mz → x=25 mz
100µs→x mz

2 modua, 8 bit, zenbaketa-erregistroak balio honekin kargatu behar dira:

256-25=231d->TH0 y TL0

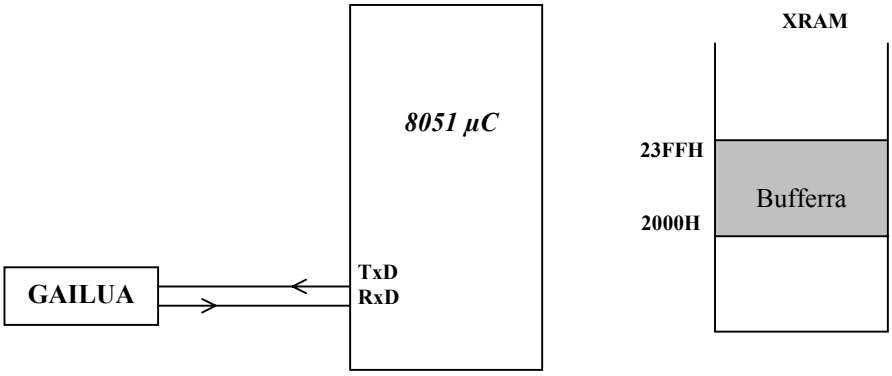
```
#include<reg51.h>
sbit P14=P1^4;
void InitT0_IEX0(void)
{
 TMOD&=0xF0;
 TMOD|=0x02;
```

```
 TH0=231;
 TL0=231;
 ET0=1;
 IT0=1;
 EX0=1;
 EA=1;
}
void InterTimerT0(void) interrupt 1
{
 static char n,veces;

 n++;
 if(n==2||n==7||n==10||n==12||n==16)
   P14=1;
 else if(n==5||n==9||n==11||n==14)
   P14=0;
 else if(n==19)
   {
    P14=0;
    n=0;
    veces++;
    if(veces==5)
     {
      TR0=0;
      veces=0;
      EX0=1; /* uhinak bukatuz gero, hurrengoak sor daitezke */
     }
    }
  }
void InterIEX0(void) interrupt 0
{
 P14=0;
 TR0=1;
 EX0=0; /* uhinak sortzen ari dela, ezin da besterik eskatu */
}
void main(void)
{
 P14=1;
 InitT0();
 while(1)
  ;
}
```

18) 6 MHz-eko kristala duen 8051 µC bat dugu, non gailu batek etengabe zortzikoteak bidaltzen baitizkio serie- atakaren bidez segundoko 220 biteko abiadurarekin eta paritaterik gabe. µC-ak datuak hartzen dituen heinean, datuen kanpo-memoriaren 2000H helbidetik aurrera gordetzen ditu, eta 1024 byte gordetzean, µC-ak BAH hitza bidaliko dio datu gehiagorik ez bidaltzeko berriro abisatu arte, hau da, DEH bidali arte. Bien bitartean, µC-ak handienetik txikienera ordenatuko ditu 1024 byteak, hasierako bufferrean utziz beste µC batek irakurri ahal ditzan. Marraztu ariketaren eskema, SFRak beren balioekin, eta idatzi kalkuluak eta funtzio hauek:

```
void InitUART(void)
void DatuHartu(void) interrupt 4
void DatuBidal(char byte)
void OrdenHanditxiki(void)
```

Ebazpena:

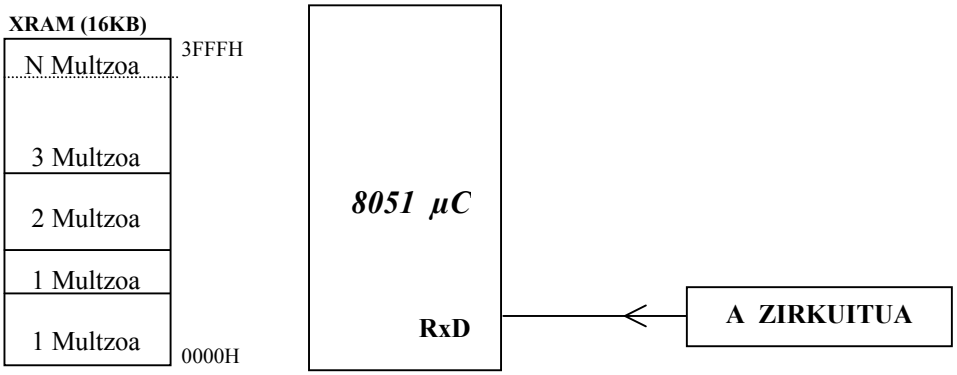



```
#include<reg51.h>
#define uchar unsigned char
uchar xdata *pnt1=0x2000;
uchar xdata *pnt2=0x2400;

void InitUART(void)
{
 SCON=0x50;  /*Harrera gaitua eta 1 modua 1 Vcom aldakorrarentzat */
 PCON&=0x7F; /*SMOD=0*/
 TMOD&=0x0F;
 TMOD|=0x20;
 TL1=185;    /* 6 MHz-eko fxtal-arentzat -->Vcom=260,41 bit/s */
 TH1=185;
 ES=1;
 EA=1;
 TR1=1;
}
void OrdenHanditxiki(void)
{
 char temp;

 for(pnt1=0x2000,pnt2=0x2001;pnt1<0x2400;pnt1++,pnt2=pnt1+1)
  for(;pnt2<0x2400;pnt2++)
   if(*pnt1<*pnt2)
    {
     temp=*pnt1;
     *pnt1=*pnt2;
     *pnt2=temp;
    }
  pnt1=0x2000; /* hurrengo betealdirako */
}
void DatuBidal(char byte)
{
 SBUF=byte;
 while(TI==0)
  ;
 TI=0;
}
void DatuHartu(void) interrupt 4
{
 if(RI==1)
 {
  RI=0;
  if(pnt1<pnt2)
   {
    *pnt1=SBUF;
    pnt1++;
```

```
    }
   else
    {
     REN=0;
     DatuHartu(0xBA);
     OrdenHanditxiki();
     DatuHartu(0xDE);
     REN=1;
    }
 }
}
void main(void)
{
 InitUART();
 while(1)
  ;
}
```

19) 6 MHz-eko kristala eta 16 kilobyteko datuen kanpo-memoria duen 51 familiako μC bat dugu. Serieko atakatik eta 62,5 bit/s abiadurarekin 8 biteko datu multzoak hartzen ditu, A zirkuitutik bidaliak. Multzo bakoitzean, lehen zortzikoteak multzoa zenbat datuk osatzen duten adierazten du (berea kontatu gabe). Multzo bakoitza datuen kanpo-memorian gordea izango da heldu ahala, eta bata bestearen jarraian gordeko dira, behetik hasita, eta memoriaren tamainarekin kezkatu gabe. Eskatzen da: sistemaren eskema marraztea, beharrezko kalkuluak egitea, SFRak dagozkien balioekin jartzea eta `void InitUART(void), void DatuHartu(void) interrupt 4` eta `void main(void)` funtzioak idaztea, sistemak zuzenki funtziona dezan.

Ebazpena:



Serieko transmisioa paritaterik gabe eta Vcom aldakorrarekin egiten denez, UARTaren 1 modua sailkatzen da (8 biteko eta Vcom aldakordun UARTa), eta T1 timerrarentzako zenbaketa-balioa kalkulatzen da, honako formula honen bidez:

V_{COM}=fosc *2 ^{SMOD}/12*(256-TH1)*32, fxtal 6MHz-ekoa denez eta SMOD=0 hartuz →TH1=TL1=6

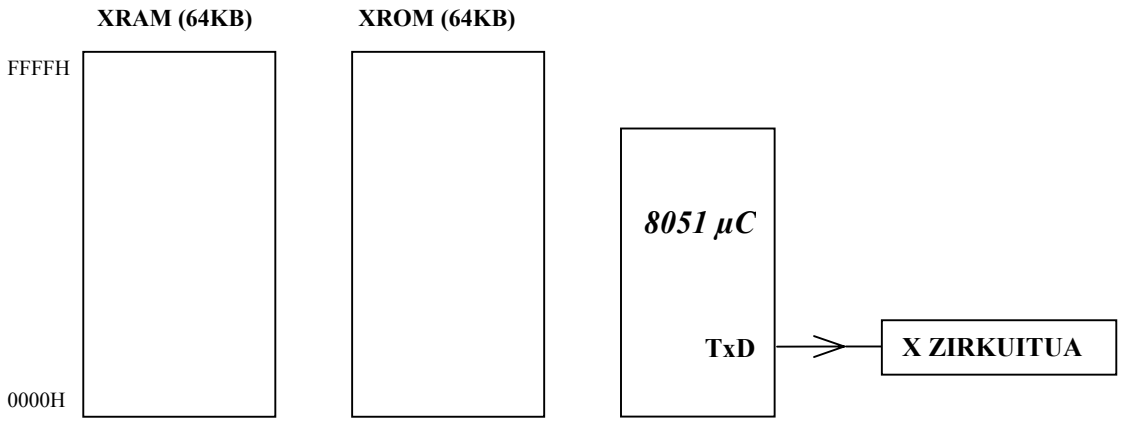
```
#include<reg51.h>
char xdata *punt1=0x0000;
void InitUART(void)
{
 TMOD&=0x0F;
 TMOD|=0x20;
 PCON&=0x7F;
 PCON|=0x00;
 TL1=6;
 TH1=6;
 SCON=0x50;
```

```
 ES=1;
 EA=1;
 TR1=1;
}
void DatuHartu(void) interrupt 4
{
 char datu;
 static char zenb;

 if(RI==1)
 {
  RI=0;
  datu=SBUF;
  if(zenb==0)
   zenb=datu;
  else
   {
    *punt1=datu;
    punt1++;
    zenb--;
    /* egin liteke bukatzean harrera ezgaitu, edo baita berriro hasi ere */
    /*if(punt1==0x3FFFF)
       REN=0; ó  punt1=0x0000; */
   }
  }
}
void main(void)
{
 InitUART();
 while(1)
    ;
}
```

20) SFRak tokatzen zaien balioekin jartzea eta ondorengo C hizkuntzako funtzioak idaztea eskatzen da, jakinik 3 MHz-eko kristaldun 8051µC-an oinarrituriko sistema bat dugula:

- `void InitUART(void)` eta `void DatuBidal(char byte)`, datuak UART serie bidez bidaltzeko paritate bakoitiarekin eta segundoko 33.96 biteko abiadurarekin.
- `void LoadXRAM(void)`, edozein baliotako datuekin betetzeko XRAM guztia.
- `void ClearXRAM(void)`, zerora hasieratzeko XRAM guztia.
- `void SendProgUART(char code *puntC)`, XROM-en kokaturiko programa UART serie bidez bidaltzeko paritate bakoitiarekin eta segundoko 33.96 biteko abiadurarekin.
- `void main(void)`, funtzio nagusia, beste guztiak deitzen dituena eta ikusteko multzo osoak zuzenki funtzionatzen duela.

Ebazpena:



Serieko transmisioa paritatearekin (bakoitia) eta Vcom aldakorrarekin egiten denez, UARTaren 3 modua sailkatzen da (9 biteko eta Vcom aldakordun UARTa) eta T1 timerrarentzako zenbaketa-balioa kalkulatzen da, honako formula honen bidez:

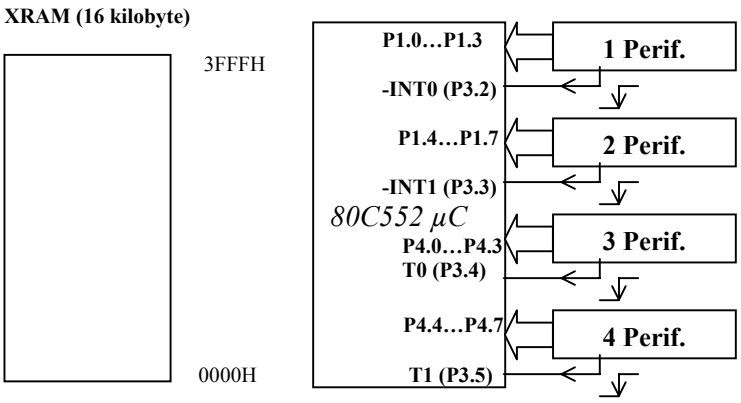
V_{COM}=fosc *2 ^{SMOD}/12*(256-TH1)*32, fxtal 3 MHz-ekoa denez eta SMOD=0 hartuz →TH1=TL1=26

Ez denez ezer hartuko, harrera ezgaitua uzten da (REN=0).

```
#include<reg51.h>

void InitUART(void)
{
 TMOD&=0x0F;
 TMOD|=0x20;
 PCON&=0x7F;
 PCON|=0x00;
 TL1=26;
 TH1=26;
 SCON=0xC0;
 TR1=1;
}
void DatuBidal(char datu)
{
 ACC=datu;
 TB8=~P;
 SBUF=datu;
 while(TI==0)
  ;
 TI=0;
}
void SendProgtoUART(char code *erak2)
{
 unsigned int n;
 for(;!((*erak2==0x00)&&(*(erak2+1)==0x00)&&(*(erak2+2)==0x00));erak++)
  DatuBidal(*erak2);
}
void LoadXRAM(void)
{
 char xdata *erak1=0x0000;
 char n;

 for(erak1=0xFFFF,n=0;erak1!=0x0000;n++,erak1--)
   *erak1=n;
 *erak1=n;
}
void ClearXRAM(void)
{
 char xdata *erak1=0x0000;


 for(erak1=0xFFFF;erak1!=0x0000;erak1--)
  *erak1=0x00;
 *erak1=0x00;
}
void main(void)
{
 char code *erak3;
 InitUART();
 CargaXRAM();
 ClearXRAM();
 erak3=0x4000;
 SendProgtoUART(erak3);/* edo erak3 erazagutu gabe eta SendProgtoUART(0x4000);*/
 while(1);
}
```

21) 6 MHz-eko kristala eta 16 kilobyteko datuen kanpo-memoria dituen 80C552 (8 pineko 6 I/O portu) µC-an oinarrituriko sistema bat dugu, zeinek bere P1 eta P4 portuetatik 4 biteko datuak jasotzen baititu horiei konektaturiko 4 periferikoetatik. Horrela, 1 periferikoa P1 portuaren behealdeari konektatua dago, 2 periferikoa P1 portuaren goialdeari, 3 periferikoa P4aren behealdeari, eta 4 periferikoa P4aren goialdeari. µC-ari jakinarazteko datua bidaliko diola periferikoren batek, honela egiten da: 0 eta 1 kanpo-etenduren bitartez 1 eta 2 periferikoen kasuan, hurrenez hurren, eta 0 eta 1 kontagailuen bitartez, 3 eta 4 periferikoen kasuan, hurrenez hurren. Guztietan, jaitsiera-saihetsa erabiliko da seinale gisa. 4 biteko datuak 8 biteko formatuan gordeko dira (datua beheko nibblean eta periferikoaren kodea, goikoan) datuen kanpo-memoria guztian, lehen helbidetik azkeneraino, eta, behin dena betetakoan, berriro ere hasieratik hasiko da betetzen.
Eskatzen da: sistemaren eskema marraztea, beharrezko kalkuluak egitea, SFRak dagozkien balioekin jartzea eta `void InitIEX01(void),void InitT01(void),void InterIEX0(void) interrupt 0, void InterIEX1(void) interrupt 2,void InterT0(void) interrupt 1,void InterT1(void) interrupt 3,void CopytoXram(char dato)` eta `void main(void)` funtzioak idaztea, sistemak zuzenki funtziona dezan.

Ebazpena:




```
#include<reg552.h>

void InitIEX01(void)
{
 IT0=1;
 IT1=1;
 EX0=1;
 EX1=1;
 EA=1;
}
void InitT01(void)
{
 TMOD=0x66;
 TL0=0xFF;
 TH0=0xFF;
 TL1=0xFF;
 TH1=0xFF;
 ET0=1;
 ET1=1;
 TR0=1;
 TR1=1;
}
void InterIEX0(void) interrupt 0
{
 char byte;

 byte=P1&0x0F;
 byte=byte|0x10;
 CopytoXram(byte);
}
void InterIEX1(void) interrupt 2
```

```
{
 char byte=0;

 byte=P1&0xF0;
 byte=byte>>4;
 byte=byte|0x20;
 CopytoXram(byte);
}
void InterT0(void) interrupt 1
{
 char byte;

 byte=P4&0x0F;
 byte=byte|0x30;
 CopytoXram(byte);
}
void InterT1(void) interrupt 4
{
 char byte;

 byte=P4&0xF0;
 byte=byte>>4;
 byte=byte|0x40;
 CopytoXram(byte);
}
void CopytoXram(char datua)
{
 static char xdata puntx=0x0000;

 if(puntx==0x4000)
  puntx=0x0000;
 *puntx=datua;
 puntx++;
}
void main(void)
{
 InitIEX01();
 InitT01();
 while(1)
  ;
}
```

22) `InitUART()` funtzioa idaztea eskatzen da, 51 familiako µC baten UART serieko ataka abiadura aldakorreko modu bietatik batean hasieratzeko, sarrera-parametroen bidez adieraziz: *modua*, *kristalaren maiztasuna (Hz-etan)*, *abiadura (bit/s-tan)*. Funtzioak behar duen guzti-guztia hasieratuko du serieko ataka pasaturiko 3 parametroek adierazten duten bezala, posible bada hori; kasu horietan 1 itzuliko du, eta, besteetan, 0. Kristalaren maiztasuna, teorian, 1 Hz-eko eta 24 MHz-eko balioen artean dago mugatua, baina portuaren baliozko abiadura heina bi balio hauek mugatzen dute: 0,25 bit/s eta 125 kilobit/s. OHARRA: ez dago sortu ezin litezkeen abiaduren baliorik.

Ebazpena:

```
#include <reg51.h>
#define fmin 24576  /*fxtal balio minimoa 0,25 bit/s-rentzat */
#define fmax 24e6
#define abmin 0.25
#define abmax 125e3

bit InitUART(char modu,long fxtal,float brate)
{
 float min,max;
 int aux;

 if((modo==1||modo==3)&&(fxtal>=fmin&&fxtal<=fmax)&&(brate<=vmax&&brate>=vmin))
```

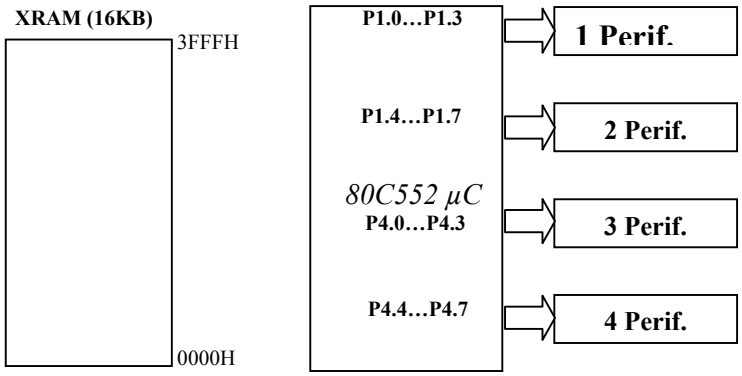
```
  {
   min=fxtal/(12*32*256); /* abiadura min eta max sorkorrak kristal. arabera */
   max=(fxtal*2)/(12*32);
   if(brate>max||brate<min) /* heinez kanpo bada, ezin du abiadura hori sortu */
     return 0;
   else
    {
     TMOD&=0x0F;
     TMOD|=0x20;
     if(brate<max/2)
      {
       PCON&=0x7F;  /* SMOD=0 */
       TH1=(256-(fxtal/(12*32*brate)));
      }
     else
      {
       PCON|=0x80;  /* SMOD=1 */
       TH1=(256-(fxtal*2/(12*32*brate)));
      }
     TL1=TH1;
     SCON=0X50;
     if(modu==3)
      SM0=1;
     ES=1;
     EA=1;
     return 1;
    }
  }
 else
  return 0;
}
void SendUART(char datu)
{
 SBUF=datu;
 while(TI==0)
  ;
 TI=0;
}
void main(void)
{
 bit emaitza;

 emaitza=InitUART(1,7e6,125);
 if(emaitza==1)
  SendUART(0xAB);
 while(1);
}
```

23) 6 MHz-eko kristala eta 16 kilobyteko datuen kanpo-memoria dituen 80C552 (8 pineko 6 I/O portu) µC-an oinarrituriko sistema bat dugu, zeinek bere P1 eta P4 portuetatik 4 biteko datuak bidatzen baitizkie 4 periferikori. Horrela, 1 periferikoa P1 portuaren behealdeari konektatua dago, 2 periferikoa P1 portuaren goialdeari, 3 periferikoa P4aren behealdeari, eta 4 periferikoa P4aren goialdeari. Transmitituriko datuak datuen kanpo-memorian gorderik daudenak dira, non, byte bakoitzeko, beheko nibbleak datu-informazioa baitu eta goikoak periferikoaren kodea. Datuen transmisioa 1 kanpo-etenduraren pinean jaitsiera-saihets bat ezartzean eskatzen da, non horrek memoriako datu guztiak 4 periferikoetara bidaltzen baititu, goiko nibblearen kodearen arabera.
Eskatzen da: sistemaren eskema marraztea, beharrezko kalkuluak egitea, SFRak dagozkien balioekin jartzea, `void InitIEX1(void)`,`void InterIEX1(void) interrupt 3` eta `void main(void)` funtzioak idaztea, sistemak zuzenki funtziona dezan.
OHARRA: datu bat idaztean periferiko baten portuan, albokoarena aldatu gabe mantendu behar da.

Ebazpena:

```
#include <reg51.h>
```

```
char xdata *erak=0x0000;
void InitIEX1(void)
{
 IT1=1;
 EX1=1;
 EA=1;
}
void InterIEX1(void) interrupt 3
{
 char perf,datu;

 while(pnt|=0x4000)
 {
  perf=0xF0&(*erak);
  dato=0x0F&(*erak);
  if(perf==0x10)
   {
    P1&=0xF0;
    P1|=dato;
   }
  else if(perf==0x20)
   {
    P1&=0x0F;
    datu<<=4;
    P1|=datu;
   }
  else if(perf==0x30)
   {
    P4&=0xF0;
    P4|=datu;
   }
  else if(perf==0x40)
   {
    P4&=0x0F;
    datu<<=4;
    P4|=datu;
   }
  erak++;
 }
erak=0x0000; /* hurrengorako */
}
void main(void)
{
 InitIEX1();
 while(1);
}
```

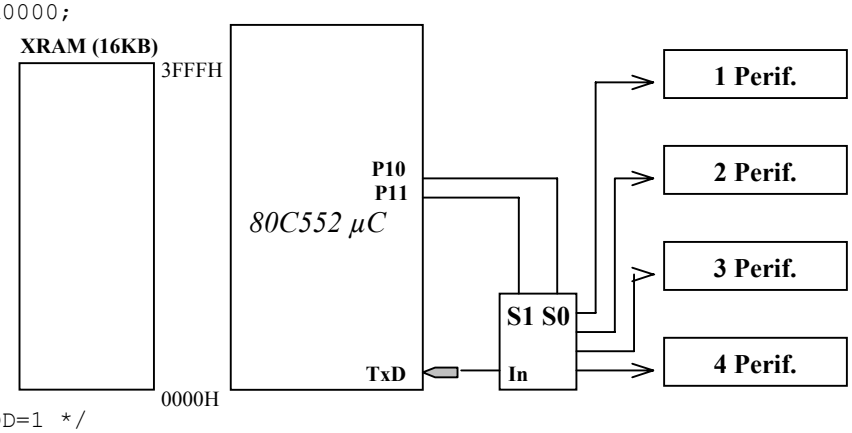


24) 6 MHz-eko kristala eta 16 kilobyteko datuen kanpo-memoria dituen 80C552 (8 pineko 6 I/O portu) µC-an oinarrituriko sistema bat dugu, zeinek bere UART serieko atakatik 8 biteko datuak transmititzen baititu (6250 bit/s eta paritaterik gabe) 4 kanpo periferikori 4 kanaldun demultiplaxore baten bitartez. Horrela, 1 periferikoa demultiplexorearen 0 kanalari konektatua dago, 2a 1 kanalari, 3a 2ari eta 4a 3 kanal edo irteerari. Transmitituriko datuak, datuen kanpo-memorian gorderik daudenak dira, non, byte bakoitzeko, beheko nibbleak datu-informazioa baitu eta goikoak periferikoaren kodea. Datuen transmisioa T0 timerraren pinean jaitsiera-saihets bat ezartzean gauzatzen da, non horrek memoriako datu guztiak 4 periferikoetara bidaltzen baititu, goiko nibblearen kodearen arabera. Erabili P1 portuaren pinak demultiplexorearean irteerak sailkatzeko.

Eskatzen da: sistemaren eskema marraztea, beharrezko kalkuluak egitea, SFRak dagozkien balioekin jartzea eta `void InitUART(void)`,`void InitT01(void)`,`void InterT0(void) interrupt 1`,`void DatuBidal(char datu)` eta `void main(void)` funtzioak idaztea, sistemak zuzenki funtziona dezan.

OHARRA: datu bat transmititzean, horrek hitzaren beheko niblea betetzen du, eta goikoan 4 zero (bitarrak) doaz.

Ebazpena:




```
#include <reg51.h>
sbit P10=P1^0; /* demux-aren 0 hautapen-bita */
sbit P11=P1^1; /* demux-aren 1 hautapen-bita */
char xdata *erak=0x0000;
void InitT01(void)
{
 TMOD=0x26;
 TL0=255;
 TH0=255;
 TL1=251;
 TH1=251;
 ET0=1;
 EA=1;
 TR0=1;
 TR1=1;
}
void InitUART(void)
{
 PCON|=0x80; /* SMOD=1 */
 SCON=0x40;
}
void DatuBidal(char datu)
{
 SBUF=datu;
 while(TI==0)
  ;
 TI=0;
}
void InterT0(void) interrupt 1
{
 char perf,datu;

 while(erak|=0x4000)
 {
  perf=0xF0&(*erak);
  datu=0x0F&(*erak);
  if(perf==0x10)
    {
     P10=0;
     P11=0;
     DatuBidal(datu);
    }
  else if(perf==0x20)
    {
     P10=1;
     P11=0;
     DatuBidal(datu);
    }
  else if(perf==0x30)
    {
     P10=0;
     P11=1;
     DatuBidal(dato);
    }
  else if(perf==0x40)
    {
     P10=1;
     P11=1;
     DatuBidal(dato);
    }
```

```
  erak++;
 }
erak=0x0000; /* hurrengorako */
}
void main(void)
{
 InitUART();
 InitT01();
 while(1);
}
```

InterT0()-entzako soluzio garatuagoa

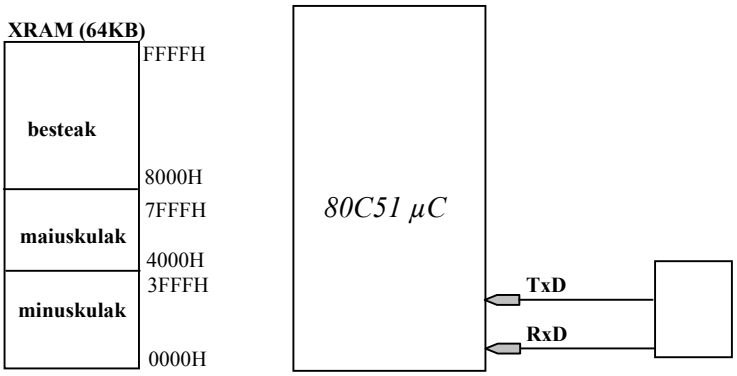
```
void InterT0_v2(void) interrupt 1
{
 char perf,datu;

 while(erak|=0x4000)
 {
  perf=0xF0&(*erak);
  datu=0x0F&(*erak);
  P1&=0xFC;              /* P12..P17 pinen egoera errespetatzen du */
  P1|=perf;              /* demux-aren irteera egokia hautatzen du */
  DatuBidal(datu);
  erak++;
 }
 erak=0x0000; /* hurrengorako */
}
```

25) 6 MHz-eko kristala eta 64 kilobyteko datuen kanpo-memoria duen 80C51 µC bat dugu, non UART serieko atakaren bidez 8 biteko datuak (6250 bit/s eta paritaterik gabe) hartzen baititugu etengabe. µC-ren eginkizuna hartzen dituen datuak hiru mota ezberdinetan sailkatzea da: alfabetoko letra minuskulak (61H eta 7AH bitarteko ascii kodea), maiuskulak (41H eta 5AH bitarteko ascii kodea), eta beste guztiak. Horrela, lehenengoak XRAMaren lehen 16 kilobyteetan gordeak izango dira, bigarrenak hurrengo 16 kilobyteetan, eta besteak XRAMaren goiko erdian. Letra mota bakoitzeko eremua betetzen denean, µC-ak datu bat bidaliko dio bidaltzaileari mota horretako datu gehiagorik bidal ez dezan: minuskulentzako `'m'`, maiuskulentzako `'M'`, eta beste guztientzako `'b'` bidaliko du. Sistemak bere XRAM guztia beterik duenean, harrera ezgaituko du. Erabili beharrezkoak diren aldagai eta erakusleak, UART harrera etenduraz eginez eta transmisioa* bakoitzak errazago uste duen bezala.

Eskatzen da: sistema osoaren bloke-diagrama marraztea, beharrezkoak diren kalkuluak egitea, SFRak beren balioekin marraztea eta `void InitUART(void),void DatuHartu(void) interrupt 4,void DatuBidal(char datu)*` eta `void main(void)` funtzioak idaztea, sistemak zuzenki funtziona dezan.

Ebazpena:

V_{COM}=fosc *2 ^{SMOD}/12*(256-TH1)*32=6250, fxtal 6 MHz-ekoa denez eta SMOD=1 hartuz→TH1=TL1=251d
(SMOD=0-rekin TH1-entzat baliogabea den osokoa ematen duenez, ez du balio.)



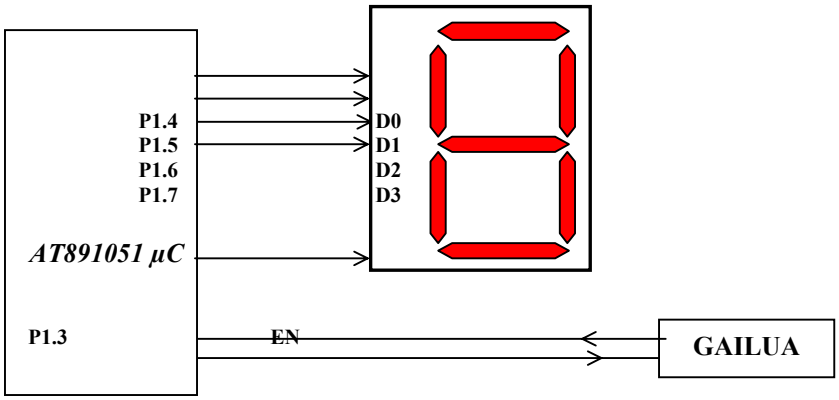

```
#include <reg51.h>
char xdata *erakmin=0x3FFF;
char xdata *erakmai=0x7FFF;
char xdata *erakbes=0xFFFF;
void InitUART(void)
{
 TMOD&=0x0F;
 TMOD|=0x20;
 TL1=251;
 TH1=251;
 PCON|=0x80; /* SMOD=1 */
 SCON=0x50; /* 1modua & REN=1*/
 ES=1;
 EA=1;
 TR1=1;
}
void DatuBidal(char datu)
{
 SBUF=datu;
 while(TI==0)
  ;
 TI=0;
}
void DatuHartu(void) interrupt 4
{
 static bit min,mai,beste;
 char byte;

 if(RI==1)
  {
   RI=0;
   byte=SBUF;
   if((0x61<=byte)&&(byte<=0x7A)&&(mins==0))
    {
     *erakmin=byte;
      erakmin--;
      if(erakmin==0xFFFF)
       {
        erakmin==0x3FFF; /* hurrengorako */
        mins=1; /* minuskulen eremua beterik */
        DatuBidal('m');
       }
    }
   else if((0x41<=byte)&&(byte<=0x5A)&&(mai==0))
    {
     *erakmai=byte;
      erakmai--;
      if(erakmai==0x3FFF)
       {
        erakmai==0x7FFF; /* hurrengorako */
        mai=1; /* maiuskulen eremua beterik */
        DatuBidal('M');
       }
    }
   else if(beste==0)
    {
     *erakbes=byte;
      erakbes--;
      if(erakbes==0x7FFF)
       {
        erakbes==0xFFFF; /* hurrengorako */
        beste=1; /* beste guztien eremua beterik */
        DatuBidal('b');
       }
    }
```

```
   if((min==1)&&(mai==1)&&(beste==1))
     REN=0; /* hiru eremuak beterik badaude, harrera ezgaitzen da */

 }/* if(RI==1)-en bukaera */
}
void main(void)
{
 InitUART();
 while(1);
}
```

26) 6 MHz-eko kristala duen AT89C2051 (51-aren arkitektura berberekoa) µC bat 7 segmentuko display bati konektatua dugu P1 portuaren bidez P1 (4, 5, 6 eta 7 pinak, 0 eta 1 ez dira erabiliko) 10 s-ko atzeranzko zenbaketa egin eta erakusteko (A-tik 0-ra). Gailuak 1 kanpo-etendura pina jaitsiera-saihets bat ezartzean hasiko da denbora-kontaketa. Horretarako, T0 timerra erabiliko da, eta display-an erakutsiko da. 0-ren kontura heltzean, µC-ak behe-mailako pultsu bat bidaliko dio gailuari P1.2 pinaren bidez; hala, display-a 0 erakusten geratuko da, beste zenbaketa berri bat eskatu arte. Display-aren egia-taula, non deskodetzaile bat integratzen baitu, eta µC-aren data sheet-a atzeko orrialdean agertzen dira.

Eskatzen da: sistemaren eskema marraztea, behar diren kalkuluak egitea, SFRak bere balioekin marraztea eta `void InitT0IEX1(void),void InterIEX1(void) interrupt 2,void InterT0(void) interrupt 1` eta `void main(void)` funtzioak idaztea, sistemak zuzenki funtziona dezan.

Ebazpena:



1 s zenbatu behar da, 10^6 µs, eta kristala 6 MHz-ekoa denez, Tmz=12*1/fxtal=12*1/6*10^6=2µs, orduan:

2µs →1 mz → x= 5*10^5 =500.000 mz
10^6µs →x mz

horrela, T0 timerraren 1 modua erabiltzea erabakitzen da, 50.000 mz zenba ditzan, eta hori 10 aldiz gertatzean, 1 s zenbatuko da:

2^{16}-50.000=15536d=3CB0H

```
#include<reg51.h>
unsigned char zenb=0xAF;
sbit P12=P1^2;  /* behe-mailako pultsuarentzat */
sbit P13=P1^3;  /* display-aren Enablearentzat  */
void InitT0IEX1(void)
{
 TMOD&=0xF0;
```

```
 TMOD|=0x01;
 TH0=0xC3;
 TL0=0x50;
 ET0=1;
 EX1=1;
 IT1=1;
 EA=1;
}
void InterIEX1(void) interrupt 2
{
 TR0=1;  /* T0 abiarazten da  atzeranzko zenbaketa sortzeko */
 P13=1;  /* erakusketa hasten da */
 P1&=num; /* A-kin */
 EX1=0;   /* aldi horretan beste etendurarik eska ez dadin */
}
void InterT0(void) interrupt 1
{
 static char aldiak,segk;

 TH0=0xC3;
 TL0=0x50;
 aldiak++;
 if(aldiak==10)
  {
   zenb-=0x10; /* 1-en gutxitzen du goiko nibblea */
   P1&=zenb;
   aldiak=0;
   segk++;
   if(segk==10)
    {
     P12=0; /* zenbaketaren bukaera */
     P12=1;
     TR0=0; /* timerra gelditzen du */
     EX1=1; /* hurrengo atzeranzko zenbaketa eskakizunerako */
     zenb=0xAF;
     segk=0;
    }
  }
}
void main(void)
{
 P13=0; /* hasieran display-a zuri */
 InitT0IEX1();
 while(1)
  ;
}
```

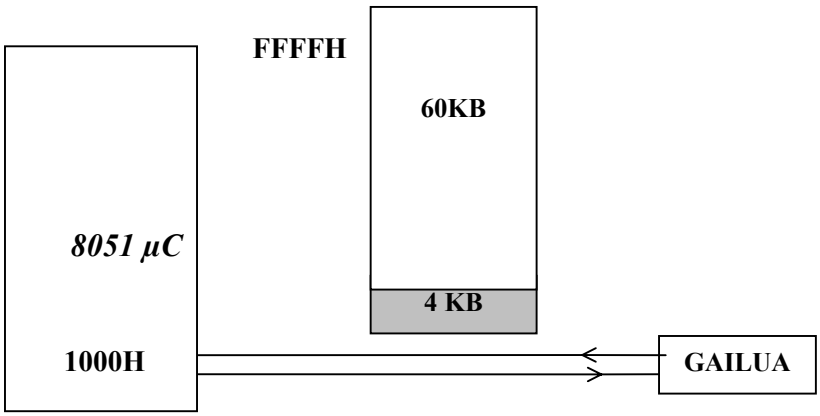
27) 4 MHz-eko kristala eta 64 kilobyteko datuen kanpo-memoria duen 8051 µC bat daukagu (kodea egikaritzeko erabiltzen den arren), non lehenengo edo beherengoko 4 kilobyteak sistemaren kontrol-programak beteak diren eta beste guztia programa-eremua den. Eremu horretan kargatuko da UART serie bidez gailu batek 3400 baudeko abiadurarekin bidaltzen dion programa (programa-eremuko lehen helbidetik azkenerantz). Baina gailuak µC-ari programa alderantziz bidaltzen dio, hau da, lehenik programaren azken agindua bidaltzen dio, eta azkenik lehena; horregatik, µC-ak programa kargatu ondoren, alderantzizko ordenan utzi behar du kargaturik, eta eremu berberean utzi, zuzenki egikaritua izan dadin.

Suposatuz gailuak ez dizkiola sekula 60 kilobyte baino handiagoak diren programak bidaliko eta programaren ondoren komando-hitz bat bidaliko diola bidalketaren bukaera adierazteko (argi, zeren hori programaren aginduren bat bezalakoa izan liteke, eta, horregatik, agiduok datu-hitzak izango dira, eta ez komando-hitzak: µC-ak hitz mota bi hauek ezberdintzeko gai izan behar du!!). µC-ak ‘O’ (Ongi) batekin erantzungo dio, eta, horren ondoren, programa ordenatuko du memorian.

Eskatzen da: sistema osoaren bloke-diagrama marraztea, beharrezkoak diren kalkuluak egitea, datu-bytea eta komando-bytea definitzea, SFRak beren balioekin jartzea eta `void InitUART(void),void HartuDatu(void) interrupt 4,void OrdenaPrograma(void)` eta `void GarbituProgramaEremua(void)` aurretik legokeen programa edo hasierako edukia atxintzeko (00H balioarekin hasieratu eremu guztia), funtzioak idaztea, eta baita `void main(void)` ere, sistemak zuzenki funtzionatu dezan.

- Ez da gure diseinuan sartzen µC XRAM-en kargaturik dauden programak egikaritzeko eskema.
- Erabili beharrezkoak iruditzen zaizkizuen aldagai orokor edo/eta lokalak, eta baita erakusleak ere, kontuan izanik guztiek beti ondo hasieratuak izan behar dutela hainbat eta hainbat programa kargatzea posible egin dezaten, dena zuzenki errepikatuz beti.

Ebazpena:



DATU bytea (bidaltzen den programaren agindu bakoitza) 9 biteko 9. bita 0 duen hitz gisa definitzen da; beste 8 bitek osatzen dute datua. KOMANDO bytea, aldiz, 9 biteko 9. bita 1 duen hitza da, eta lehen 8 bitek osatzen dute komandoa. Alderantziz har zitezkeen, baina, edonola ere, beharrezkoa da 9 bit bidaltzea/hartzea (start bita eta stop bita kontuan hartu gabe), hitz mota bat bestetik ezberdintzeko. Gainera, SM2 bitak 0 egoeran egon behar du, µC-ak mota bietako hitzak SBUF-en hartu (eta gorde) ahal izan ditzan.

```
#include <reg51.h>
xdata char *erak1=0x1000;
xdata char *erak2=0x1000;
unsigned int tam=0;

void InitUART(void)
{
 SCON=0xD0;  /* 3 modua 9 bitentzat, SM2=0 eta REN=1 */
 PCON&=0x7F; /* SMOD=0 (edo PCON|=0x80 SMOD=1 bada eta TL1=250) */
 TMOD&=0x0F;
 TMOD|=0x20;
 TL1=253;
 TH1=TL1;
 TR1=1;
 ES=1;
 EA=1;
}
void GarbituProgramaEremua(void)
{
 while(erak1!=0)
  {
   *erak1=0;
   erak1++;
```

```
  }
 erak1=0x1000;
 }
void OrdenaPrograma(void)
{
 char temp;

 for(erak1-=1,tam=tam/2;tam!=0;erak1--,erak2++,tam--)
  {
   temp=*erak1;
   *erak1=*erak2;
   *erak2=temp;
  }
 erak1=0x1000; /* hurrengo programentzako berrabiarazten dira */
 erak2=erak1;
 tam=0;
}
void HartuDatu(void) interrupt 4
{
 if(RI==1)
 {
  RI=0;
  if(RB8==0) /* DATU bytea bada, programa eremuan gordetzen da */
   {
    *erak1=SBUF;
    erak1++;
    tam++;
   }
  else   /* KOMANDO bytea bada, bukatu da harrera eta 'O' komandoarekin */
   {     /* erantzuten dio gailuari. */
    TB8=1;
    SBUF='O';
    while(TI==0)
      ;
    TI=0;
    OrdenaPrograma();
   }
  }
 }
void main(void)
{
 GarbituProgramaEremua();
 InitUART();
 while(1);
}
```

28) 3 MHz-eko kristala duen AT89C2051 (51-aren arkitektura berberekoa, ikus grafikoa) µC baten P3 portuko 5 pinean zehar, 0 irudiak agertzen duen bezalako forma duen eta 1041,6666 Hz-eko maiztasuna duen 4 uhin sortzea nahi da, kanpo-etenduraren pinean jaitsiera-saihets bat ezartzen den bakoitzean. Bakarrik T0 timerra erabiliz, zera eskatzen da: uhinak marraztea haiek sortzeko behar diren kalkuluak adieraziz, SFRak beren balioekin, eta `void InitT0IEX0(void),void InterIEX0() interrupt 0` eta `void InterT0() interrupt 1` funtzioak idaztea.

Ebazpena:

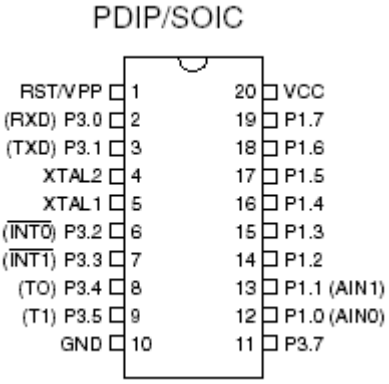
Seinalearen periodoa T=1/f=1/1041,6666= 960 µs da; orduan, T0 timerrak T/20=960 µs /20= 48 µs-tik behin eten dezan erabakitzen da.

Kristala 3 MHz-ekoa denez, Tmz=12*1/fxtal=12*1/3*10^6= 4µs, eta, orduan:

4µs→1 mz → x= 12 mz
48µs →x mz

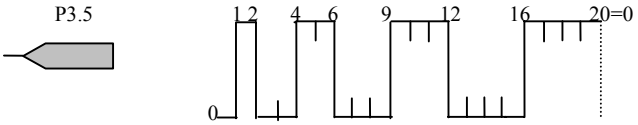
2 moduan, 8 bitekoa, zenbaketa-erregistroa eta birkargakoa balio hauekin kargatu behar dira:

2^8-12=244d



AT89C2051 µC-aren pin-konfigurazioa

```
#include<reg51.h>
sbit P35=P3^5;
void InitT0IEX0(void)
{
 TMOD&=0xF0;
 TMOD|=0x02;
 TH0=244;
 TL0=244;
 ET0=1;
 EX0=1;
 IT0=1;
 EA=1;
}
void InterIEX0(void) interrupt 0
{
 TR0=1;  /* T0 abiarazten du 4 periodoak sortzeko */
 EX0=0;  /* besterik eska ez dadin oraingoak sortzen diren bitartean */
}
void InterT0(void) interrupt 1
{
 static char etenak,uhinak;

 etenak++;

 if(etenak==1||etenak==4||etenak==9||etenak==16)
   P35=1;
 else if(etenak==2||etenak==6||etenak==12||etenak==20)
  {
   P35=0;
   if(etenak==20)
    {
     etenak=0;
     uhinak++;
     if(uhinak==4)
      {
       TR0=0; /* T0 gelditzen du 4. periodoa sortzean */
       EX0=1; /* beste 4 periodo eskatu ahal izan daitezen */
      onda=0;
      }
    }
  }
}
void main(void)
{
 InitT0();
 P35=0;
 while(1)
  ;
}
```



29) 3 MHz-eko kristala duen AT89C2051 µC-an oinarrituriko sistema bat dugu, eta tiristore batentzat ms bateko zabaleradun ate-pultsuak sortu nahi dira P3.7 pinetik, uhin-erdiko karga erresistibodun zuzentzaile monofasiko kontrolatu batentzat (ikus irudian). Suposatzen da 220 V-eko sareko maiztasuna 50 Hz-ekoa dela, eta pultsu-sorrera sareko tentsioa $\omega t=2\pi$-tik pasatzean sinkronizatua dagoela, 1 zirkuituak atzemanez. Orduan, zirkuitu horrek µC-ari eskatuko dio tiristorearen atearentzako pultsua 1 kanpo-etendura pinean jaitsiera-saihets bat ezarriz (behin), 0 eta π radianen arteko tiristorearen desarra-angelua (0 eta 10 arteko balio osoko eta positiboaren bidez, hurrenez hurren) zirkuituak emanik, P1 portuaren bidez. Zirkuituak eskakizuna behin egiten du, eta µC-a P1-en emaniko desarra-angeludun pultsua sortzen geratzen da hurrengo sareko zikloetan, zirkuituak beste angelu ezberdin bateko pultsua eskatu arte. Horrela, angelu hau π baino handiagoa bada (eskakizun baliogabea), orduan µC-ak utzi egingo dio pultsuak sortzeari, hurrengo baliodun eskakizunen bat egin arte.

Eskatzen da: bakarrik T1 timerra erabiliz, zuzentzailearen kargaren tentsioa 0, $\pi/10$ eta π desarra-angeluentzako marraztea, behar diren kalkuluak adieraziz, SFRak beren balioekin, eta `void InitT1IEX1(void),void InterIEX1(void) interrupt 2,void InterT1() interrupt 3` eta `void main(void)` funtzioak idaztea.

Ebazpena:

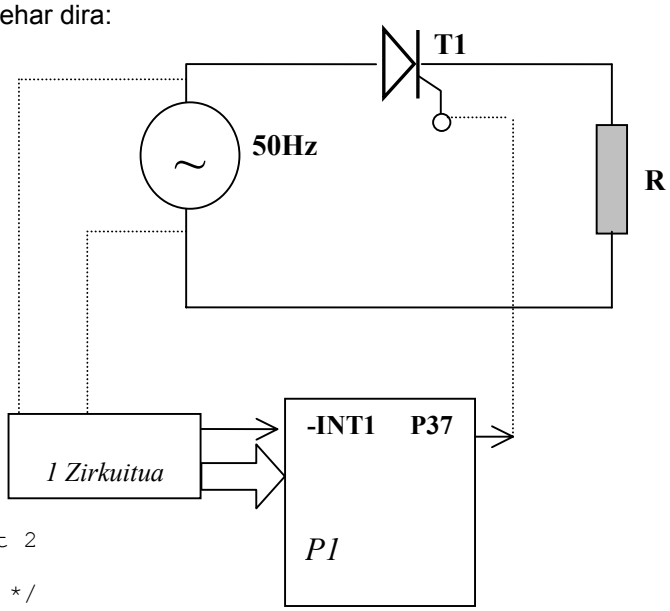
Seinalearen periodoa T=1/f=1/50= 20 ms denez, orduan T0 timerrak T/20= 20 ms /20= 1 ms =1000 µs-tik behin eten dezan erabakitzen da.

Kristala 3 MHz-ekoa denez, Tmz=12*1/fxtal=12*1/3*10^6= 4 µs, eta orduan:

4µs→1 mz → x= 250 mz
1000µs →x mz

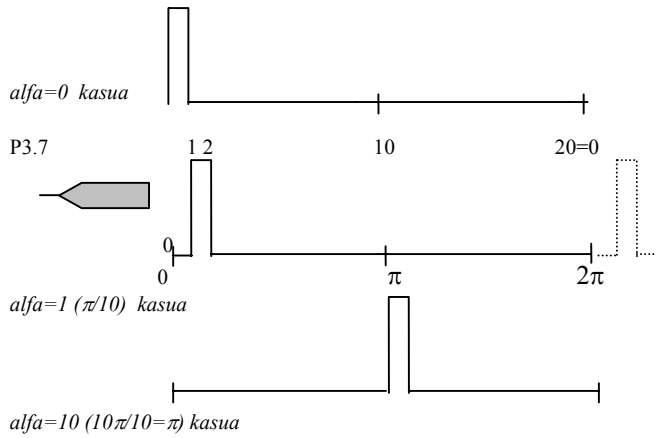
2 moduan, 8 bitekoa, zenbaketa-erregistroa eta birkargakoa balio hauekin kargatu behar dira:

2^8-250=6d




```
#include<reg51.h>
#define uchar unsigned char
uchar alfa;
sbit P37=P3^7;
void InitT1IEX1(void)
{
 TMOD&=0x0F;
 TMOD|=0x20;
 TH1=6;
 TL1=6;
 ET1=1;
 EX1=1;
 IT1=1;
 EA=1;
}
void InterIEX1(void) interrupt 2
{
 if(P1<0x0B) /* 0<=alfa<=0x0A */
  {
   alfa=P1;
   if(alfa==0)
    P37=1;
   TR1=1;   /* T1 martxan jartzen du periodoak sortzeko */
  }
 else
  TR1=0;    /* T1 gelditzen du pultsu gehiagorik ez sortzeko */
```

```
}
void InterT1(void) interrupt 3
{
 static char etenak;

 etenak++;

 if(etenak==alfa)
   P37=1;
 else if(etenak!=alfa)
  {
   P37=0;
   if(etenak==20)
    {
     etenak=0;
     if(alfa==0)
      P37=1;
    }
  }
}
void main(void)
{
 InitT0();
 P37=0;
 while(1)
  ;
}
```



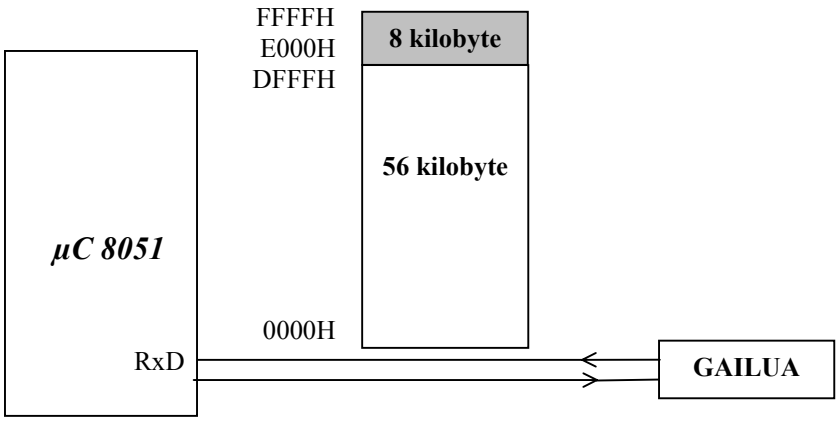
30) 7 MHz-eko kristala eta 64 kilobyteko datuen kanpo-memoria duen 8051 µC bat daukagu (kodea egikaritzeko erabiltzen den arren), non azkeneko edo gorengoko 8 kilobyteak sistemaren kontrol-programak beteak diren eta beste guztia programa-eremua den. Eremu horretan kargatuko da UART serie bidez gailu batek 1519 baudeko abiadurarekin bidaltzen dizkion programa (programa-eremuko lehen helbidetik azkenerantz). Baina gailuak µC-ari programa alderantziz bidaltzen dio, hau da, lehenik programaren azken agindua bidaltzen dio, eta azkenik lehena; horregatik, µC-ak programa kargatu ondoren, alderantzizko ordenan utzi behar du kargaturik, eta eremu berberean utzi, zuzenki egikaritua izan dadin.

Gailua programaren bat bidaltzera doanean, B1H komandoa (programa Bidali, eta, horren ondorioz, µC-ak programa-eremua garbitu behar du) bidaltzen dio 1. byte bezala, gero programa, eta azkenik, BAH (Bidalketaren Azkena), non µC-ak erantzungo dion:

- 00H (Ongi) komandoa bidaliz, horren ondoren memorian den programa ordenatuko du, baina hori programa guztia sartu ahal izan bada eremu horretan.
- EEH (Errua) komandoa bidaliz, programa ezin izan bada sartu programa-eremuan, orduan "akatsa" aldagai orokorra aktibatuko da, kontrol-programak jakin dezan ezin litekeela egikaritu jasotako azken programa.

Eskatzen da: sistema osoaren bloke-diagrama marraztea, kalkuluak egitea eta `void InitUART(void)`,`void HartuDatua(void) interrupt 4`,`void OrdenaPrograma(void)` eta `void GarbituProgramaEremu(void)` aurretik legoken programa edo hasierako edukia atxintzeko (00H balioarekin hasieratu eremu guzti hori), funtzioak idaztea, eta baita `void main(void)` ere, sistemak zuzenki funtziona dezan.

Ebazpena:



DATU bytea (bidaltzen den programaren agindu bakoitza) 9 biteko 9. bita 0 duen hitz gisa definitzen da; beste 8 bitek datua osatzen dute. KOMANDO bytea, aldiz, 9 biteko 9. bita 1 duen hitza da, eta lehen 8 bitek komandoa osatzen dute. Alderantziz har zitezkeen, baina, edonola ere, beharrezkoa da 9 bit bidaltzea/hartzea (start bita eta stop bita kontuan hartu gabe), hitz mota bat bestetik ezberdintzeko. Gainera, SM2 bitak 0 egoeran egon behar du, µC-ak mota bietako hitzak SBUF-en hartu (eta gorde) ahal izan ditzan.

```
#include <reg51.h>
xdata char *erak1=0x0000;
xdata char *erak2=0x0000;
char akatsa=0;
unsigned int tam=0;

void InitUART(void)
{
 SCON=0xD0;  /* 3 modua 9 bitentzat, SM2=0 eta REN=1 */
 PCON&=0x7F; /* SMOD=0 (edo PCON|=0x80 SMOD=1 bada eta TL1=232) */
 TMOD&=0x0F;
 TMOD|=0x20;
 TL1=244;
 TH1=TL1;
 TR1=1;
 ES=1;
 EA=1;
}
void GarbituProgramaEremua(void)
{
 while(erak1!=0xE000)
  {
   *erak1=0;
   erak1++;
  }
 erak1=0x0000;
 }
void OrdenaPrograma(void)
{
 char temp;

 for(erak1-=1,tam=(tam-1)/2;tam!=0;erak1--,erak2++,tam--)
  {
   temp=*erak1;
   *erak1=*erak2;
   *erak2=temp;
  }
 erak1=0x0000;
```

```
 erak2=erak1;
 tam=0;
}
void HartuDatua(void) interrupt 4
{
 static char datuak;
 if(RI==1)
 {
  RI=0;
  if((RB8==1)&&(SBUF==0xB1)) /* Bidali KOMANDOA hartzen badu */
   {
    datuak=1;
    GarbituProgramaEremua();
   }
  else if((RB8==1)&&(SBUF==0xBA)) /* Bidalketaren bukAera KOMANDOA hartuz */
   {
    datuak=0;
    TB8=1;
    SBUF=Ox00;  /* Programa Ongi gOrdea */
    while(TI==0)
      ;
    TI=0;
    OrdenaPrograma();
   }
  else if((RB8==0)&&(datuak==1)) /* DATU zortzikotea hartzen badu */
   {
    if((erak1!=0xE000))         /* eta tokirik badago, datua gordetzen du. */
     {
      *erak1=SBUF;
      erak1++;
      tam++;
     }
    else     /* baina tokirik ez bada, daturik gordetzen ez eta */
     {
      TB8=1;
      SBUF=0xEE;   /* "akatsa" edo Errua erantzuten*/
      while(TI==0)
       ;
      TI=0;
      akatsa=1;
     }
   }
 }
}
void main(void)
{
 InitUART();
 while(1);
}
```

31) 16 MHz-eko kristala duen 8051 µC baten 1 portuko 0 eta 1 pinetan zehar, irudiak agertzen duen bezalako forma duten 2525,2525 Hz-eko bi uhinak sortzea nahi da, 0 kanpo-etenduraren pinean jaitsiera-saihets bat ezartzen den bakoitzean (periodo bat eskakizun bakoitzeko). Bakarrik T0 timerra erabiliz, zera eskatzen da: uhinak marraztea haiek sortzeko behar diren kalkuluak adieraziz, SFRak beren balioekin, eta `void InitT0IEX0(void)`, `void InterIEX0() interrupt 0`, `void InterT0() interrupt 1` eta `void main(void)`, funtzioak idaztea.

Ebazpena:

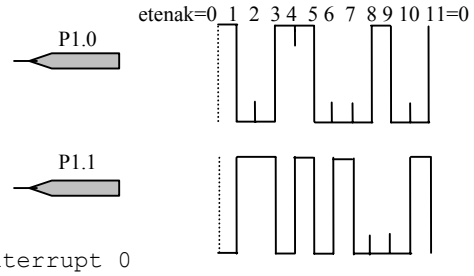
Seinalearen periodoa T=1/f=1/2525,2525=3,96e-4 s=396µs da, orduan T0 timerrak T/11= 396 µs /11= 36 µs-tik behin eten dezan erabakitzen da.

Kristala 16 MHz-ekoa denez, Tmz=12*1/fxtal=12*1/16*10^6=0.75 µs, horrela:

0.75 µs→1 cm → x= 48 cm
36 µs →x cm

2 moduan, 8 bitekoa, zenbaketa-erregistroa eta birkargakoa balio hauekin kargatu behar dira:
2^8-48=256-48=208d

```
#include<reg51.h>
sbit P10=P1^0;
sbit P11=P1^1;
void InitT0IEX0(void)
{
 TMOD&=0xF0;
 TMOD|=0x02;
 TH0=208;
 TL0=208;
 ET0=1;
 EX0=1;
 IT0=1;
 EA=1;
}
void InterIEX0(void) interrupt 0
{
 TR0=1;  /* T0-a abiarazten da periodo 1 sortzeko */
 EX0=0;  /* besterik eska ez dadin oraingoa sortzen den bitartean */
}
void InterT0(void) interrupt 1
{
 static char etenak;

 etenak++;
 if(etenak==1||etenak==3||etenak==5||etenak==11)
  {
   P10=~P10;
   P11=~P11;
   if(inters==11)
    {
     etenak=0;
     TR0=0;   /* periodo 1 sortua */
     EX0=1;   /* beste periodo bat sortu nahi bada */
    }
  }
 else if(etenak==4||etenak==6||etenak==7||etenak==10)
   P11=~P11;
 else if(etenak==8||etenak==9)
   P10=~P10;
}
void main(void)
{
 InitT0IEX0();
 P11=0;
 while(1)
  ;
}
```



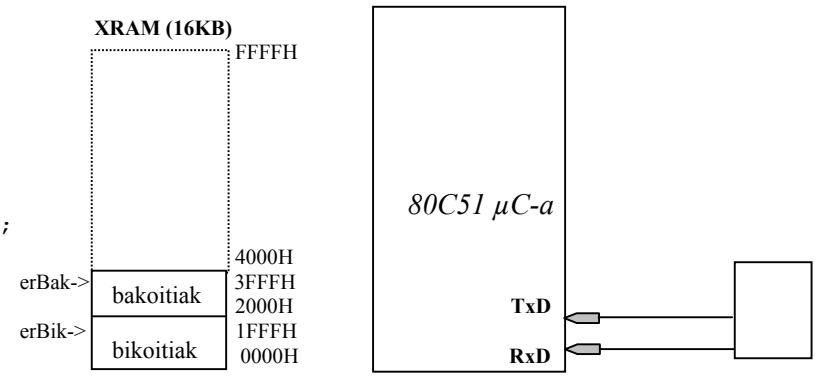
32) 6 MHz-eko kristala eta 16 kilobyteko datuen kanpo-memoria duen 8051 µC bat daukagu, non 8 kilobyte behekoetan 8 biteko datu bikoitiak gordetzen diren, eta 8 kilobyte goikoetan 8 biteko datu bikoitiak. Datu horiek kanpoan µCarekin serie bidezko konexioa duen 1 gailuak bidaltzen dizkio (Vcom=3125 bit/s), eta transmisioa START (AAH) komandoarekin hasten du. Horrela, µCak datuak hartzea espero du (bikoitiak eta bakoitiak), XRAMean tokatzen zaien eremuan gordetzeko. Eremu bakoitza betetzean, µ-ak komando bat bidaliz erantzungo dio 1 gailuari: CEH bakoitien eremua betetzean, edo AEH bikoitiena betetzean (eta horren ostean 1 gailuak ez du mota horretako datu

gehiagorik bidaliko). Eremu biak betetzen direnean, µCa, hasierako egoera berean gelditukо da: 8 kilobyte datu bakoiti eta beste 8 kilobyte bikoiti hartzeko prest.

Sistema osoaren bloke-diagrama marraztea, kalkuluak egitea `eta void InitUART(void), InterUART(void) interrupt 4`, sistemak behar den bezala funtzionatzeko beharrezkoak diren funtzioak idaztea, eta, behar izanez gero, `void TransmUART(char datu)` ere bai, `void main(void)` orokorraz gain.

Ebazpena:

9 bit ematen duen komunikazio modu bat behar da, non 9. bita DATU-byte (9. bita 0-n) KOMANDO-bytetik (9. bita 1-en) desberdintzeko erabiltzen baita. Gainera, eskatzen den Vcom-a T1 timerrari esker lortzen da bakarrik; beraz, UART 3 modua erabili behar da: kristala 6 MHz-ekoa denez, Vcom=fxtal*2^{SMOD}/12*(256-TH1)*32=3125 bit/s, non SMOD=1, TH1=246. Hasieran, µC-ak komando bat hartzea espero du, eta, horretarako, aukera bi daude:
→SM2=1 (a bertsioa), komandoak hartzeko (kargatzeko) gai da soilik.
→SM2=0 (b bertsioa), datuak nahiz komandoak hartzeko gai da, RB8 bidez atzemanez (0 edo 1, hurrenez hurren).




```
#include<reg51.h>
char xdata *erBak=0x3FFF;
char xdata *erBik=0x1FFF;
void InitUART(void)
{
 TMOD&=0x0F;
 TMOD|=0x20;
 PCON|=0x80;//SMOD=1;
 TL1=246;   //TL1==TH1=251;
 TH1=246;   //SMOD=0 bada.
 SCON=0xF0; //0xD0,a be.
 ES=1;
 EA=1;
 TR1=1;
}
void InterUART(void) interrupt 4  // a bertsioa
{
 static bit bik,bak;

 if(RI==1)
 {
  RI=0;
  if(RB8==1&&SBUF==0xAA)
    SM2=0; // datuak hartzeko prestatzen da
  else if(RB8==0&&(SBUF&1)==1)
   {
    *erBak=SBUF;  //datu bakoitia gordea
    erBak--;
    if(erBak==0x1FFF)
     {
      TB8=1;      // byte komandoa
      SBUF=0xCE; // bidali
      while(TI==0)
       ;
      TI=0;
      bak=1;      // bakoitien eremua beterik
     }
   }
  else if(RB8==0&&(SBUF&1)==0)
   {
    *erBik=SBUF; // datu bikoitia gordea
    erBik--;
```

```
    if(erBik==0xFFFF)
     {
      TB8=1;
      SBUF=0xAE;
      while(TI==0)
       ;
      TI=0;
      bik=1;     // bikoitien eremua beterik
     }
   }
  if(bak==1&&bik==1)
   {
    SM2=1;
    erBak=0x3FFF;
    erBik=0x1FFF;
   }
 }
}
void InterUART(void) interrupt 4  // b bertsioa
{
 static bit bik,bak;
 static char datuak;

 if(RI==1)
 {
  RI=0;
  if(RB8==1&&SBUF==0xAA)
    datuak==1; // datuak hartzeko prestatzen da
  else if(RB8==0&&datuak==1&&(SBUF&1)==1)
   {
    *erBak=SBUF;  //datu bakoitia gordea
    erBak--;
    if(erBak==0x1FFF)
     {
      TB8=1;     // byte komandoa
      SBUF=0xCE; // bidali
      while(TI==0)
       ;
      TI=0;
      bak=1;     // bakoitien eremua beterik
     }
   }
  else if(RB8==0&&datuak==1&&(SBUF&1)==0)
   {
    *erBik=SBUF; // datu bikoitia gordea
    erBik--;
    if(erBik==0xFFFF)
     {
      TB8=1;
      SBUF=0xAE;
      while(TI==0)
       ;
      TI=0;
      bik=1;     // bikoitien eremua beterik
     }
   }
  if(bak==1&&bik==1)
   {
    datuak=0;
    erBak=0x3FFF;
    erBik=0x1FFF;
   }
 }
}
```

```
void main(void)
{
 InitUART();
 while(1);
}
```

33) Idatz ezazu `bit InitT0(float fxtal_Hz,float time_sg)` funtzioa, T0 timerra 16 biteko tenporizadore gisa hasieratzeko, denbora *time_sg* segundotan zenba dezan eta *fxtal_Hz* Hz-etako duen kristala sarrera-parametro gisa pasatzen zaiolarik. Horrela, funtzioak itzuliko du:

- 0-a, T0-a denbora hori edo antzekoren bat zenbatzeko hasieratzea lortu bada, haren etendura baimentzeaz gain.
- 1-a, denbora heinetik (mailatik) kanpo geratzen bada.

Idatzi, gainera, main()funtzio nagusia, non InitT0()deitzen duen, adibide batekin probatzeko.

Ebazpena:

(12/fxtal)sg→1 mz → n= y*fxtal/12 mz
y sg→n mz

```
#include<reg51.h>
#define uint unsigned int
bit InitT0(float fxtal_Hz,float time_sg)
{
 uint 16bitbal;
 long nmz;  // long da, harrapatzeko heinez kanpo geratzen bada
 bit em;

 nmz=time_sg*fxtal_Hz/12);
 if(nmz>65536||nmz<=0)  // mz kopurua heinez kanpo dago
  em=1;                 // 'errorea' itzultzen du
 else
 {
  TMOD&=0xF0;
  TMOD|=0x01;
  16bitbal=65536-nmz;
  TL0=16bitbal;
  TH0=16bitbal>>8;
  ET0=1;
  EA=1;
  TR0=1;
  em=0;
 }
 return em;
}
void main(void)
{
 bit emaitza;

 emaitza=InitT0(4.2e6,3e-3);
 while(1);
}
```

34) 8 MHz-eko kristala duen 8051 µC baten P1 portuko 1 eta 3 pinetan zehar, irudiak agertzen duen bezalako forma duten 2525,2525 Hz-eko bi uhinak sortzea nahi da, 1 kanpo-etenduraren pinean jaitsiera-saihets bat ezartzen den bakoitzean (3 periodo, eskakizun bakoitzeko). T1 timerra erabiliz soilik, zera eskatzen da: uhinak marraztea haiek sortzeko behar diren kalkuluak adieraziz, SFRak beren balioekin, eta `void InitT1IEX1(void), void InterIEX1() interrupt 2, void InterT1() interrupt 3` eta `void main(void)` funtzioak idaztea.

Ebazpena:

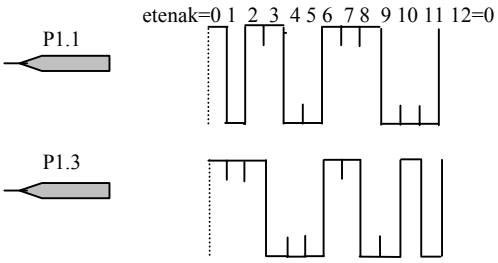
Seinalearen periodoa T=1/f=1/1111,11=900e-6 s= 900 µs da; orduan, T1 timerrak T/12= 900 µs /12= 75 µs-tik behin eten dezan erabakitzen da.

Kristala 8 MHz-ekoa denez, Tcm=12*1/fxtal=12*1/8*10^6= 1.5 µs, horrela:

1.5µs→1 mz → x= 50 mz
75µs →x mz

2 moduan, 8 bitekoa, zenbaketa-erregistroa eta birkargakoa balio hauekin kargatu behar dira:

2^8-50=256-50=206d




```
#include<reg51.h>
sbit P11=P1^1;
sbit P13=P1^3;
void InitT1IEX1(void)
{
 TMOD&=0x0F;
 TMOD|=0x20;
 TH1=206;
 TL1=206;
 ET1=1;
 EX1=1;
 IT1=1;
 EA=1;
}
void InterIEX1(void) interrupt 2
{
 TR1=1; /* T0-a abiarazten da 3 periodo sortzeko */
 EX1=0; /* besterik eska ez dadin oraingoa sortzen den bitartean */
}
void InterT1(void) interrupt 3
{
 static char etenak,periodoak;

 etenak ++;
 if(etenak==6||etenak==12)
  {
   P11=~P11;
   P13=~P13;
   if(etenak==12)
   {
    periodoak++;
    if(periodoak==3)
    {
     etenak=0;
     periodoak=0;
     TR1=0;  /* 3. periodoa sortua */
     EX1=1;  /* beste 3 periodo sortu ahal izateko */
    }
   }
  }
 else if(etenak==1|| etenak==2|| etenak==4|| etenak==9)
   P11=~P11;
 else if(inters==3||inters==8||inters==10||inters==11)
   P13=~P13;
}
void main(void)
{
 InitT1();
 while(1)
  ;
```

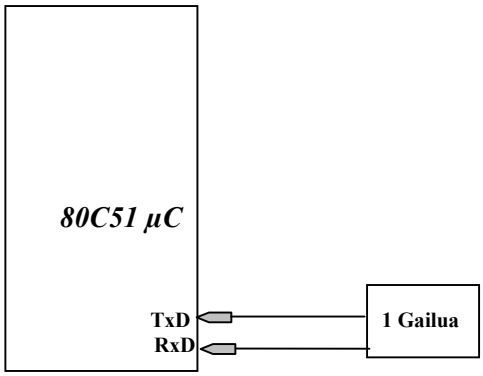
```
}
```

35) 6 MHz-eko kristala duen 8051 µC bat daukagu konektatua gailu bati UART bidez. Gailuak zeinurik gabeko 8 biteko bi eragigai bidaltzen dizkio, µC-ak batuketaren emaitzaz erantzun diezaion. Gailuak, eragigaiak bidali aurretik, BAH ("badoaz") komandoa bidali behar dio, µC-a eragigai biak jasotzeko zain egon dadin (bi eragigai soilik, gailuak espero diren baino gehiago bidaltzen badizkio ere). µCa, emaitza bidali eta gero, hasieratuta geratuko da berriro ere eragiketa berbera egiteko, beti eta beti BAH komandoa hartzen badu, noski. Vcom=3125 bit/s. Sistema osoaren bloke-diagrama marraztea, kalkuluak egitea eta `void InitUART(void),InterUART(void) interrupt 4`, eta beharrezkoa izanez gero, `void TransmUART(char datu)`, funtzioak idaztea, `void main(void)` orokorraz gain. Beharrezkoak dira sistemak zuzenki funtziona dezan.

Ebazpena: 9 bit ematean duen komunikazio modu bat behar da, non:

- Harreran (µC-arentzat), horrek DATUAK (eragigaiak) eta KOMANDOAK desberdindu ahal izan behar ditu, eta, horretarako, 9. bita erabiliko da, DATU bytea (9. bita 0an) eta KOMANDO bytea (9. bita 1ean).
- Transmisioan (µC-arentzat), µC-ak DATUAK (emaitza) bidaltzen duenez bakarrik eta 1 gailuak ez duenez beste ezer espero horretaz gain, orduan erabakitzen da 9. bita 8 biteko emaitzak gordetzeko ahalmena gainditzen dutenentzat erabilia izango dela, 9 biteko emaitzak bidali ahal izateko 1 Gailuari.

Gainera, Vcom-a T1 timerrari esker lortzen da, hori gabe ezingo bailitzake hori lortu; horretarako, UART-a 3 moduan hasieratzea behar da: kristala 6 MHz-ekoa denez, Vcom=fxtal*2^{SMOD}/12*(256-TH1)*32=3125 bit/s, non SMOD=1, TH1=246. Hasieran, µC-ak komando bat hartzea espero du, non horretarako: →SM2=1, komandoak hartzeko (kargatzeko) gai da soilik, lehenengo gauza hartzea espero duena baita.




```
#include<reg51.h>
#define uchar unsigned char
#define uint unsigned int
void InitUART(void)
{
 TMOD&=0x0F;
 TMOD|=0x20;
 PCON|=0x80;
 TL1=246;
 TH1=246;
 SCON=0xF0; //0xD0, b bertsioa
 ES=1;
 EA=1;
 TR1=1;
}
void InterUART(void) interrupt 4  // a bertsioa
{
 static uchar eragigaia,a,b,c,datuak;
 static uint emaitza;

 if(RI==1)
 {
  RI=0;
  if(RB8==1&&SBUF==0xBA)
    SM2=0; // eragigaiak hartzeko prestatzen da
  else if(RB8==0&&eragigaia<2)
   {
    if(eragigaia==0)
     {
      eragigaia++;
      a=SBUF;  // lehen eragigaia gordea
      datuak=1;
     }
```

```
    else
     {
      b=SBUF;  // bigarren eragigaia gordea
      emaitza=a+b;
      TB8=emaitza&0x100; // emaitzaren 9. bita
      SBUF=emaitza;      // emaitza bidali
      while(TI==0)
       ;
      TI=0;
      datuak=0; // hasierako baldintzak
      SM2=1;
     }
   }
 }
}
void main(void)
{
 InitUART();
 while(1);
}
```

36) Idatz ezazu `float InitT0(float fxtal_Hz,float time_sg,bit mode)` funtzioa T0 timerra 8 biteko (mode=0 bada) edo 16 biteko (mode=1 bada) tenporizadore gisa hasieratzeko, denbora `time_sg` segundotan zenba dezan, `fxtal_Hz` Hz-eko kristala eta `mode` modua sarrera-parametro gisa pasatzen zaizkiolarik. Horrela, funtzioak itzuliko du:

- T0-a hasieratzea lortu izan den denbora erreala, etendura gaitzeaz gain,
- -1, hasieratze-balioa denbora heinetik (mailatik) kanpo geratzen bada.

Idatzi, gainera, `main()` funtzio nagusia, non `InitT0()` deitzen duen, adibide batekin probatzeko.

Ebazpena:

(12/fxtal)s→1 cm → n= y*fxtal/12 mz
y s→n cm

```
#include<reg51.h>
#define uint unsigned int
float InitT0(float fxtal_Hz,float time_sg,bit mode)
{
 uint 16b_bal;
 long nmz; // long da, harrapatzeko heinez kanpo geratzen bada
 float ema;

 nmz=(uint)(time_sg*fxtal_Hz/12);

 if(mode==0&&nmz<256&&nmz>=0)
 {
  TMOD&=0xF0;
  TMOD|=0x02;
  16b_bal=256-nmz;
  TL0=16b_bal;
  TH0=16b_bal;
  ET0=1;
  EA=1;
  TR0=1;
  ema=nmz*12/fxtal;
  return ema;
 }
else if(mode==1&&nmz<65536&&nmz>=0)
 {
  TMOD&=0xF0;
  TMOD|=0x01;
  16b_bal=65536-nmz;
```

```
  TL0=16b_bal;
  TH0=16b_bal>>8;
  ET0=1;
  EA=1;
  TR0=1;
  ema=nmz*12/fxtal;
  return ema;
 }
else         // mz kopurua heinez kanpo dago (orokorrean edo hautaturiko moduan)
  return -1; // 'errorea' itzultzen du
}
void main(void)
{
 float emaitza;

 emaitza=InitT0(4.2e6,3e-3,0);
 emaitza=InitT0(4.2e6,3e-3,1);
 while(1);
}
```

37) Idatz ezazu `float InitTN (bit N,float fxtal_Hz,float time_sg,bit mode)` funtzioa, TN timerra (T0 N=0 bada, eta T1 N=1 bada) 8 biteko (mode=0 bada) edo 16 biteko (mode=1 bada) tenporizadore gisa hasieratzeko, denbora `time_sg` segundotan zenba dezan, `fxtal_Hz` Hz-eko kristala eta `mode` modua sarrera-parametro gisa pasatzen zaizkiolarik. Horrela, funtzioak itzuliko du:

- T0-a hasieratzea lortu izan den denbora erreala eskatzen zaion `time_sg` balioaren ± % 10 heinaren barnean badago, etendura gaitzeaz gain,
- -1, hasieratze-balioa denbora-heinetik (mailatik) kanpo geratzen bada.

Idatzi, gainera, `main()` funtzio nagusia, non `InitT0()` deitzen baitu, adibide batekin probatzeko.

Ebazpena:

(12/fxtal)sg→1 mz → n= y*fxtal/12 mz
y sg→n mz

```
#include<reg51.h>
#define uint unsigned int
float InitTN(bit N, float fxtal_Hz, float time_sg, bit mode)
{
 float denb,ema;
 uint 16b_bal;
 long nmz; // long da, harrapatzeko heinez kanpo geratzen bada
 bit em;

 nmz=(time_sg*fxtal_Hz/12);
 temp=12*nmz/fxtal;
 if(mode==1)
 {
  if((time_sg<temp*0.9||temp*1.1<time_sg)&&(nmz>65536||nmz<=0)  //heinez kanpo
   ema=-1;
  else if(N==0)
  {
   TMOD&=0xF0;
   TMOD|=0x01;
   16b_bal=65536-nmz;
   TL0=16b_bal;
   TH0=16b_bal>>8;
   ET0=1;
   EA=1;
   TR0=1;
   ema=denb;
 }
 else if(N==1)
```

```
 {
  TMOD&=0x0F;
  TMOD|=0x10;
  16b_bal=65536-nmz;
  TL1=16b_bal;
  TH1=16b_bal>>8;
  ET1=1;
  EA=1;
  TR1=1;
  ema=denb;
 }
}
else
{
  if((time_sg<temp*0.9||temp*1.1<time_sg)&&(nmz>256||nmz<=0)  //heinez kanpo
   res=-1;
  else if(N==0)
  {
   TMOD&=0xF0;
   TMOD|=0x02;
   16b_bal=256-nmz;
   TL0=16b_bal;
   TH0=TL0;
   ET0=1;
   EA=1;
   TR0=1;
   ema=denb;
  }
 else if(N==1)
 {
  TMOD&=0x0F;
  TMOD|=0x20;
  16b_bal=256-nmz;
  TL1=16b_bal;
  TH1=TL1;
  ET1=1;
  EA=1;
  TR1=1;
  ema=denb;
 }
}
return ema;
}
void main(void)
{
 float emaitza;

 emaitza=InitTN(0,4.2e6,3e-3,1);
 while(1);
}
```

38) Mikrokontrolatzaile-sistema batean 5 lerro bider 2 zutabeko konfigurazioa duen 10 tekladun matrize-teklatua egituratu/konektatu nahi da. P1.0 eta P1.1 pinak ezker- eta eskuin-zutabeekin konektaturik joango dira, hurrenez hurren, eta lerroak P1.2 pinetik (goiko lerroa) P1.6 pinera (beheko lerroa) konektatuak joango dira. Marraztu teklatuaren eta 80C51ren arteko konexio-eskema osoa. Idatzi Tekla10Irakur funtzioa, non sakatzen den teklaren arabera 0 eta 9 bitarteko zenbakia bidaltzen digun. Idatzi etengabe teklatuaren funtzioa deitu eta teklatuaren kodea P3 portuaren pisu gutxieneko 4 bitetan erakutsiko duen programa nagusia.

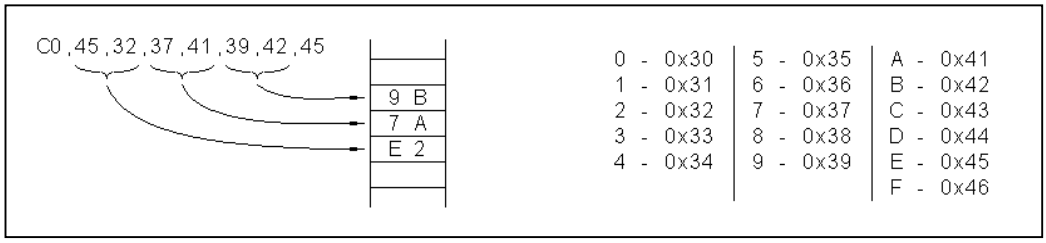
0	1
2	3
4	5
6	7
8	9

Ebazpena:

```
#include <reg51.h>
#define uchar unsigned char

void Denb(int denb)
{
 for(;denb!=0;denb--) // fxtal=4MHz eta denb=4000 badira, 20ms-ko atzerapena
   ;
}

uchar Sakatua(void)
{
 uchar bal;
 bal=~P1;
 return(bal & 0xFC); // inongo teklarik ez badago sakaturik, 0 itzultzen du.
Teklaren bat sakatuz gero, != 0
}

uchar Tekla10Irakur(void)
{
 uchar tekla[5][2]={0,1,2,3,4,5,6,7,8,9};
 uchar lerro,zutabe,ler,zut;

 P1=0xFC;
 while(Sakatua()!=0)// teklaren bat sakaturik
  ;
 Denb(4000);
 while(Sakatua()==0)// teklaren bat sakatzeko zain
  ;
 Denb(4000);
 lerro=P1;
 P1=0x83; // zutabea irakurtzeko
 zutabe=P1;
 switch(lerro)
  {
   case 0xF8: ler=0;break;
   case 0xF4: ler=1;break;
   case 0xEC: ler=2;break;
   case 0xDC: ler=3;break;
   case 0xBC: ler=4;break;
   default:ler=0xEE;break;
  }
 switch(zutabe)
  {
   case 0x82: zut=0;break;
   case 0x81: zut=1;break;
   default:zut=0xEE;break;
  }
 if(ler==0xEE||zut==0xEE)
   return 11;
 else
   return(tekla[ler][zut]);
}
void main(void)
{
 while(1)
  P3=Tekla12Irakur();
}
```

39) Idatz ezazu serie atakaren bidez datuak hartu eta ondoren datuen kanpo-memorian gordetzen dituen programa, kontuan izanik serie bidez hartzen diren datuak ASCII kodeak direla, eta, horregatik, memorian datu 1 gorde ahal izateko 2 datu beharko ditugula serie bidez. Horrela bada, irudiko datu bikote sorta hartzen badugu, irudian agertzen den bezala gordeko dira memorian. Datuen harrera hasteko, beharrezkoa izango da 0xC0 kodea hartzea, eta datuak datuen kanpo-memoriako 0x2000 helbidetik aurrera gordeko dira.

Ikus dezakezu irudian komunikazioan izango diren hamaseitar datu guztien ASCII kodea: badakigu bidaliko dizkiguten datuak 0x30-0x39 eta
0x41-0x46 bitarteetan bakarrik egongo direla (ez ditugu izango alfabetoko a-f bitartekoak) Ez dago baieztatu beharrik UART bidez datozen datuak zuzenak diren ala ez.



UART-a 2500 bit/s-ko abiaduran hasieratua izan behar da 2,4 MHz-eko kristala dugula kontuan izanik.

Ebazpena:

```
#include<reg51.h>

void HasiUART(void)
{
 SCON=0x70; //Aginduak hartzeko prest, 2500 bit/s, 1 modua
 PCON|=0x80;//SMOD=1;
 TMOD&=0x0F;
 TMOD|=0x20;
 TL1=251;
 TH1=TL1;
 TR1=1;
 ES=1;
 EA=1;
}
void HartUART(void) interrupt 4
{
 static char xdata *erak=0x2000;
 static char bak=1,datubik,datubak;
 if(RI==1)
  {
   RI=0;
   if(RB8==1&&SBUF=0xC0) // C0H agindua bada,
     SM2=0; //datu-harrerarako prestatu
   else if(RB8==0&&bak==1)
     {
      if(SBUF>=0x30&&SBUF<=0x39)
        datubak=SBUF-0x30;
      else
       datubak=SBUF-0x37;
      bak=0;
     }
   else if(RB8==0&&bak==0)
     {
      if(SBUF>=0x30&&SBUF<=0x39)
        datubik=SBUF-0x30;
      else
           datubik=SBUF-0x37;
      bak=1;
      datubak<<=4; //goiko aldea
      *erak=datubak|datubik; //goiko eta beheko aldea batu eta memorian gorde
      erak++;
      if(erak==0x0000) //memoria betetzean
           REN=0; // harrera gelditu
     }
```

```
   }
}
void main(void)
{
 HasiUART();
 while(1);
}
```

40) 8051-n oinarrituriko sistema bateko datuen kanpo-memorian eta 1000H helbidetik aurrera, gertakizunak gordetzen dira. Horietako gertakizun bakoitzak 64 byte hartzen ditu, eta lehen hiru byteetan gertakizunaren orduak (1. bytea), minutuak (2.a) eta segundoak (3.a) gordetzen dira. Orduan, hiru funtzioren bidez gertakizun hauek ordenatzea eskatzen da, non behean dagoen gertakizuna zaharrena den, eta azkena, berriena. Horretarako, `void Ordenatu (char n, char xdata *er1)` erabiliko da; `n` gertakizun kopurua izango da, eta `er1` gertakizunen esparruko hasierako helbidea, baina horrek beste bi funtzio hauek deitu beharko ditu:

`bit Konparatu(char xdata *er1, char xdata *er2)` non gertakizun biren beherengo helbideak erakusten dituzten erakusleak ematen zaizkion eta konparatu egiten dituen, esanez zein den berriena: `er1` erakusleak erakusten digun gertakizuna berriagoa (handiagoa) bada `er2`-rena baino, orduan 1 itzultzen du, eta bestela, 0.

`void Trukatu(char xdata *er1, char xdata *er2)` non gertakizun biren beherengo helbideak erakusten dituzten erakusleak ematen zaizkion eta trukatzen dituen.

Ebazpena:

```
#include <reg51.h>
bit Konparatu(char xdata *er1,char xdata *er2)
{
 bit emaitza;
 char xdata *erA=er1,*erB=er2; // jatorrizko erakusleak hobeto geldirik uztea

 if(*erA>*erB)
  emaitza=1;
 else if(*erA<*erB)
  emaitza=0;
 else // orduak berdinak balira, minutuak ikusi behar
  {
   er1++;
   er2++;
   if(*erA>*erB)
    emaitza=1;
   else if(*erA<*erB)
    emaitza=0;
   else // minutuak ere berdinak balira, orduan segundoak ikusi behar
    {
     er1++;
     er2++;
     if(*erA>*erB)
      emaitza=1;
     else
      emaitza=0;
    }
  }
 return emaitza;
}
void Trukatu(char xdata *er1,char xdata *er2)
{
 char denb,zenb;
 char xdata *erA=er1,*erB=er2;
```

```
 for(zenb=0;zenb<0x40;zenb++,erA++,erB++)
  {
   denb=*erA;
   *erA=*erB;
   *er2B=denb;
  }
}
void Ordenatu(int n,char xdata *er1)
{
 int zenb,bar;
 char xdata *er2;

 for(zenb=0,er2=er1+0x40;zenb<n;zenb++,er1=er1+0x40,er2=er2+0x40)
  for(bar=zenb;bar<n;bar++,er2=er2+0x40)
    if(Konparatu(er1,er2))
     Trukatu(er1,er2);
}
void main(void)
{
 Ordenatu(13,0x1000);
 while(1);
}
```

41) 8051 mikrokontrolatzaile-sistema batean, 2 lerro bider 3 zutabeko konfigurazioa duen 6 tekladun matrize-teklatua egituratu/konektatu nahi da. P1.0-tik P1.2 pinera zutabeak eta P1.3-tik P1.4 pinera lerroak konektatuko direla.

a) Marraztu teklatuaren eta 80C51ren arteko konexio-eskema osoa eta teklatua konfigurazio honekin egituratzeko taula eta kodeak.
b) Adierazi zer pauso emango dituzun teklatua egituratzeko (irakurtzeko) eta zein diren puntu aipagarrienak.
c) Idatzi Tekla6Irakur funtzioa, non sakatzea espero duen, eta sakatzen den teklaren arabera, bere ASCII kodea itzuliko duena.
d) Idatzi programa nagusia, non bukaezineko begizta baten barruan teklatua irakurtzen den eta sakaturiko teklen kodeak datuen barne-memoriako 40H helbidetik hasitako 32 kokalekuan gordetzen diren etengabe (buffer zirkularra).

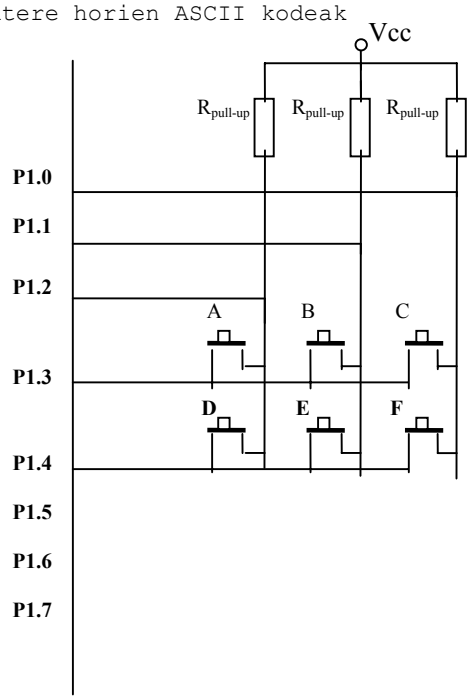
Ebazpena:

Egoera	*P1.7*	*P1.6*	*P1.5*	*P1.4*	*P1.3*	*P1.2*	*P1.1*	*P1.0*	*Hamaseitarra*
Lerroak hasieratu	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**F8**
0 Lerroa	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**F0**
1 Lerroa	**1**	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**E8**
Zutabeak hasieratu	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**E7**
0 Zutabea	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**E6**
1 Zutabea	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**E5**
2 Zutabea	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**E3**

A	**B**	**C**
D	**E**	**F**

```
#include <reg51.h>
#define uchar unsigned char

void Denb(int denb)
{
 for(;denb!=0;denb--) // fxtal=4MHz eta denb=4000 badira, 20 ms-ko atzerapena
   ;
```

```
}
uchar Sakatua(void)
{
 uchar bal;
 bal=~P1;
 return(bal & 0xF8); // inongo teklarik ez badago sakaturik, 0 itzultzen du.
Teklaren bat sakatuz gero, != 0
}
uchar Tekla6Irakur(void)
{
 uchar tekla[2][3]={'A','B','C','D','E','F'}; // karaktere horien ASCII kodeak
 uchar lerro,zutabe,ler,zut;

 P1=0xF8;
 while(Sakatua()!=0)// teklaren bat sakaturik
  ;
 Denb(4000);
 while(Sakatua()==0)// teklaren bat sakatzeko zain
  ;
 Denb(4000);
 lerro=P1;
 P1=0xE7; // zutabea irakurtzeko
 zutabe=P1;
 switch(lerro)
  {
   case 0xF0: ler=0;break;
   case 0xE8: ler=1;break;
   default:ler=0xEE;break;
  }
 switch(zutabe)
  {
   case 0xE6: zut=0;break;
   case 0xE5: zut=1;break;
   case 0xE3: zut=1;break;
   default:zut=0xEE;break;
  }
 if(ler==0xEE||zut==0xEE)
   return 0x3F; // tekla bat baino gehiago sakatuz, '?' itzultzen du
 else
   return(tekla[ler][zut]); // bestela, teklaren (karakterearen) ASCII kodea
}
void main(void)
{
 uchar n,*erak;

 while(1)
  for(erak=0x40;erak<0x40+32;erak++) // 40H helbidetik aurrerako 32 byteko
    *erak=Tekla6Irakur();            // buffer zirkularrean gordetzen
}
```



42) Sistema batek urte osoko egutegia gordetzen du 366 byteko arrai batean: pisu gutxieneko 5 bitak hileko eguna gordetzeko, eta pisu gehieneko 3 bitak asteko eguna (1 – astelehena, … 7 – igandea) gordetzeko erabiltzen dira. Urtea bisiestoa bada, `EGUTEGIA[59]` posizioak otsailaren 29a gordeko du. 30 eguneko hilak hauexek dira: apirila, ekaina, iraila eta azaroa.

Idatzi `void OsatuEgutegia(bit bisiestoa, char urtarrila_1)`. Urte hori bisiestoa dela adierazi behar diogu, aldagai hori 1 balioarekin jarriz, eta urtarrilaren 1-a asteko zer egun den ere eman behar dugu.

Ebazpena:

```
char xdata EGUTEGIA[366];
```

```
void OsatuEgutegia(bit bisiestoa, char urtarrila_1 )
{
 char hilaren_egunak[]={31,28,31,30,31,30,31,31,30,31,30,31};
 char n_hila,eguna,asteko_eguna;
 int indizea;

 if (bisiestoa)
  hilaren_egunak[1]= 29;
 for (n_hila=0,indizea=0;n_hila<12;n_hila++ )
  {
   for(eguna=1;eguna<=hilaren_egunak[n_hilea]; eguna++,indizea++ )
    {
     asteko_eguna=urtarrila_1<<5;
     EGUTEGIA[indizea]=eguna+asteko_eguna;
     urtarrila_1++;
     if(urtarrila_1==8)
      urtarrila_1=1;
    }
  }
 }
void main(void)
{
 char urtarrila_1_2007=1;
 bit bisi=1;

 OsatuEgutegia(bisi,urtarrila_1_2007);
 while(1);
}
```

43) 8051 mikrokontrolatzaile-sistema batean, 3 lerro bider 2 zutabeko konfigurazioa duen 6 tekladun matrize-teklatua egituratu/konektatu nahi da, P1.0 eta P1.1 pinetan zutabeak eta P1-etik P1.4 pinera lerroak konektatuko direla.

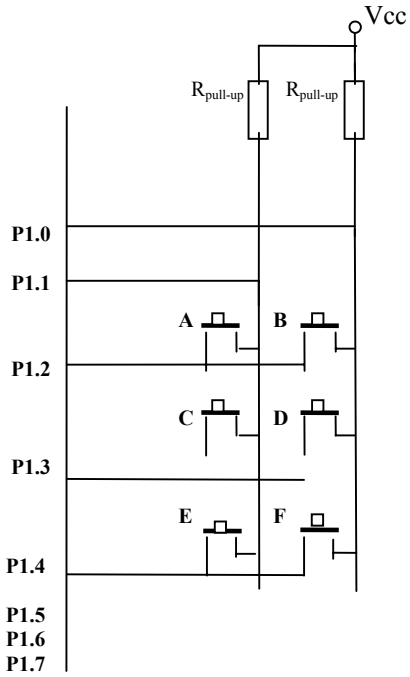
a) Marraztu teklatuaren eta 80C51ren arteko konexio-eskema osoa, eta teklatua konfigurazio honekin egituratzeko taula eta kodeak.
b) Adierazi zer pauso emango dituzun teklatua egituratzeko (irakurtzeko) eta zein diren puntu aipagarrienak.
c) Idatzi Tekla6Irakur funtzioa, i sakatzea espero duen eta sakatzen den teklaren arabera, bere balio hexamaseitarra itzuliko duena.
d) Idatzi programa nagusia, bukaezineko begizta baten barruan teklatua irakurtzen dena eta sakaturiko teklen balioa UART serie-atakatik 6250 bit/s-tan (fxtal=6MHz) eta paritate bakoitiarekin bidaliko duena, eta baita, ataka hori hasieratzeko funtzioa ere.

Egoera	*P1.7*	*P1.6*	*P1.5*	*P1.4*	*P1.3*	*P1.2*	*P1.1*	*P1.0*	*Hamaseitarra*
Lerroak hasieratu	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**FE**
0 Lerroa	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**F8**
1 Lerroa	**1**	**1**	**1**	**1**	**0**	**1**	**0**	**0**	**F4**
2 Lerroa	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**EC**
Zutabeak hasieratu	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**E3**
0 Zutabea	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**E2**
1 Zutabea	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**E1**

A	**B**
C	**D**
E	**F**

Ebazpena:

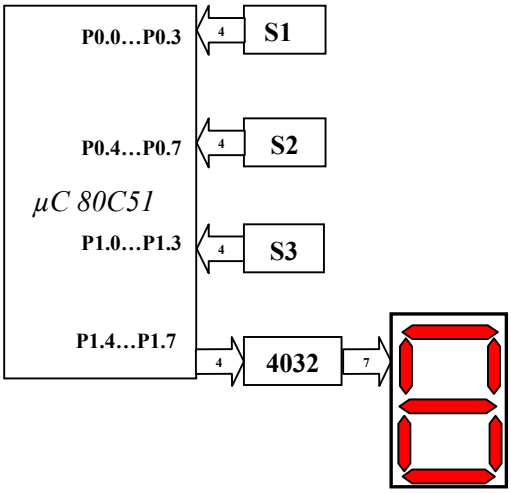
```
#include <reg51.h>
#define uchar unsigned char

void Denb(int denb)
{
 for(;denb!=0;denb--) // fxtal=4MHz eta denb=4000 badira, 20 ms-ko atzerapena
   ;
}
uchar Sakatua(void)
{
 uchar bal;
 bal=~P1;
 return(bal & 0xF8); // inongo teklarik ez badago sakaturik, 0 itzultzen du.
Teklaren bat sakatuz gero, != 0
}
uchar Tekla6Irakur(void)
{
 uchar tekla[3][2]={0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F}; // karaktere horien
balioak
 uchar lerro,zutabe,ler,zut;

 P1=0xEF;
 while(Sakatua()!=0)// teklaren bat sakaturik
  ;
 Denb(4000);
 while(Sakatua()==0)// teklaren bat sakatzeko zain
  ;
 Denb(4000);
 lerro=P1;
 P1=0xE3; // zutabea irakurtzeko
 zutabe=P1;
 switch(lerro)
  {
   case 0xF8: ler=0;break;
   case 0xF4: ler=1;break;
   case 0xEC: ler=2;break;
   default:ler=0xEE;break;
  }
 switch(zutabe)
  {
   case 0xE2: zut=0;break;
   case 0xE1: zut=1;break;
   default:zut=0xEE;break;
  }
 if(ler==0xEE||zut==0xEE)
   return 0x3F; // tekla bat baino gehiago sakatuz, '?' itzultzen du
 else
   return(tekla[ler][zut]); // bestela teklaren balioa
}
void BidalUART(uchar datua)
{
 ACC=datua;
 TB8= ~P;
 SBUF=datua;
 while(TI==0)
  ;
 TI=0;
}
void InitUART(void)
{
 TMOD&=0x0F;
 TMOD|=0x20;
 TL1=251;
```



```
 TH1=251;
 PCON|=0x80; /* SMOD=1 */
 SCON=0xD0; /* 3modua & REN=1*/
 ES=1;
 EA=1;
 TR1=1;
}
void main(void)
{
 while(1)
  BidalUART(Tekla6Irakur);
}
```

44) 5,77 MHz-eko kristala duen 8051 µC-an oinarrituriko sistema bat dugu 4 biteko 3 tenperatura-sentsoreri konektaturik. Sentsore horiek gela batean daude banandurik, eta suposatzen da 0 eta 15 º C bitarteko tenperatura izango dela. Idatz itzazu `void InitT0(void),void InterT0(void)` eta `void main(void)` funtzioak, sistemak 7 segmentuko display batean adieraz dezan sentsore bakoitzaren tenperatura segundo bateko iraupenarekin, goranzko ordenan eta '-' adieraziz hasiz (horrek ere segundo 1 irauten du). Marraztu sistema osoa, P0 eta P1 portuak eta behar diren osagaiak bakarrik erabiliz. Deskodetzailea: 40XY, sentsoreen eta displayaren konexioak: S1 (P0.0÷P0.3), S2 (P0.4÷P07), S3 (P1.0÷P01.3), DISPLAY (P1.4÷P1.7)

Ebazpena:



Segundo bat, 1000 ms, ezin duenez zuzenean zenbatu T0 tenporizadoreak, orduan 50 ms zenbatzea erabakitzen da, hau da, etenak 50 ms-tik behin eskatzea. Horrela, 20 bider hori egineran, badakigu segundo 1 pasatu dela.

Kristala 5,77 MHz-ekoa denez, Tmz=12*1/fxtal=12*1/5,77*10^6= 2,079 µs, orduan:

2,079 µs→1 mz → x=24041,66 mz
50000 µs→x mz

1 moduan, 16 bit, zenbaketa-erregistroak ondorengo balio hauekin kargatu beharko dira:

65536-24041,66 ≈ 41494 = A216H->TH0=A2H eta TL0=16H

```
#include <reg51.h>
void HasiT0(void)
{
```

```
 TMOD&=0xF0;
 TMOD|=0x01;
 TL0=0x16;
 TH0=0xA2;
 ET0=1;
 EA=1;
 TR0=1;
 P1&=0x0F;
 P1|=0x00;
}
void EtenT0(void) interrupt 1
{
 static char zenb,seg,datua;

 zenb++;
 TL0=0x16;
 TH0=0xA2;
 if(zenb==20)
 {
  zenb=0;
  seg++;
 }
 if(seg==1)
 {
  datua=P0;
  datua<<=4;
 }
 else if(seg==2)
 {
  datua=0;
  datua=P0&0xF0;
 }
 else if(seg==3)
 {
  datua=P1;
  datua<<=4;
 }
 else if(seg==4)
 {
  seg=0;
  datua=0;
 }
 P1&=0x0F;
 P1|=datua;
}
void main(void)
{
 HasiT0();
 while(1);
}
```

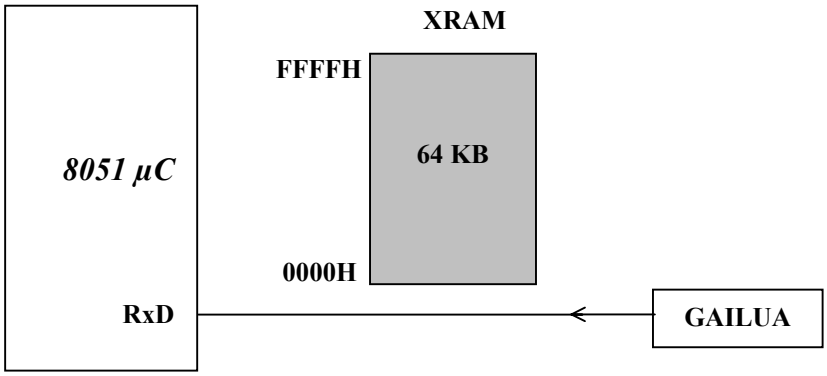
45) 11,0592 MHz-eko kristala duen 8051 µC-an oinarrituriko sistema bat dugu, non µC-ak gailu batek UART serie bidez bidalitako 16 byteko datu sortak hartu eta datuen kanpo-memorian gordetzen dituen, 1000H helbidetik aurrera. Komunikazioan errorerik ez dagoela ziurtatzeko, teknika bi erabiliko ditugu: batak datu bakoitzaren paritate bit bakoitia kontuan hartuko du, eta, besteak, 16 byteko datu sorta osoaren batuketa bitarra osokoa den datu baten eta sortaren ondoren bidaliko du. Horrela bada, datu bakoitza hartu ahala konprobatu beharko du µC-ak hori ondo datorren ala ez. Ondo datorren datu bakoitzeko, P1.0 pinean konektaturik dagoen led diodo berdea piztuko da, eta horrela iraungo du hurrengo datua aztertu arte. Aldiz, datua txarto heltzen bada, piztuko den led-a gorria izango da, P1.1 pinari konektaturikoa, alegia, eta bestea itzali egin beharko da. Kasu honetan, jada ez luke aplikatuko 2. teknika, ezta datu sorta gordeko ere kanpo-memorian. Baina, denak ondo iritsiz bakarka, ondoren

sorta gisa ondo iritsi direla konprobatu behar da, non gauza bera aplikatzen den led diodo biekin. Ondo iristen den datu sorta kanpo-memorian azken sortaren ondoren gordeko da beti. Vcom=19200 bit/s

Ebazpena:

Serieko transmisioa paritatearekin (bakoitia) eta Vcom aldakorrarekin egiten denez, UARTaren 3 modua sailkatzen da (9 biteko eta Vcom aldakordun UART-a) eta T1 timerrarentzako zenbaketa-balioa kalkulatzen da, honako formula honen bidez:

V_{COM}=fosc *2 ^{SMOD}/12*(256-TH1)*32, fxtal 11,0592 MHz-ekoa denez eta SMOD=1 hartuz (bestela 0-rekin errore handia izanen da abiaduran)→TH1=TL1= 253




```
#include<reg51.h>

char xdata *xramerak=0x1000;
sbit LedOrlegia=P1^0;
sbit LedGorria =P1^1;
void HasiUART(void)
{
 SCON=0xD0;
 PCON|=0x80;
 TMOD&=0x0F;
 TMOD|=0x20;
 TL1=253;
 TH1=253;
 ES=1;
 EA=1;
 TR1=1;
}
void HartUART(void) interrupt 4
{
 bit pbak;
 static int batuketa,osokoa;
 static char datua[18],in;
 if(RI==1)
  {
   RI=0;
   datua[in]=SBUF;
   pbak=RB8;
   ACC=datua[in];
   if(pbak==~P)
    {
     LedOrlegia=0;
     batuketa+=datua[in];
     if(in==16)
       osokoa=datua[in];
```

```
      else if(in==17)
       {
       osokoa<<=8;
       osokoa|=datua[in];
          batuketa=batuketa-datua[16]-datua[17];
       if(batuketa==osokoa)
        {
         LedOrlegia=0;
         for(in=0;in<16;xramerak++,in++)
            *xramerak=datua[in];
         in=0; // hurrengo datu sortarentzako prest
        }
       }
         in++;
        }
   else
    {
     LedOrlegia=1;
     LedGorria=0;
     REN=0;  // harrera gelditu
    }
  }
}
void main(void)
{
 HasiUART();
 while(1);
}
```

46) Software musika-erreproduzitzaile batek maiztasun ezberdinetako heinak adierazteko ikuspen-area bat erabiltzen du soinu-seinale baten frekuentzia-esparrua erakusteko. Hain zuzen ere, 16 maiztasunetan banatzen du soinu bat, eta horietako bakoitzaren anplitudea edo indarra 8 marratxoz adieraz daiteke. Marratxoak behetik gora eta elkarren jarraian marrazten dira beti. Marratxoa piztuta, "1" logikoaren bidez adierazten da µC-arentzat, eta soinuaren indarra adierazten du; itzalita badago, aldiz, indarrik eza adierazten du. Horrela bada, ikuspen-area hori 16 byteko XRAM memoriaren bidez kudeatzen da (hor du bere imajina eta horren bidez kontrolatzen) eta, horrela, 1 byte dugu maiztasun-hein bakoitzeko, non memoriako lehenengo posizioak erakuslearen lehenengo zutabea (maiztasun heina) adierazten baitu, bigarrenak bigarrena, eta horrela hurrenez hurren. Idatzi `void BeteZutabea(uchar* merak, char zutabea, uchar egoera)` hiru parametrodun funtzioa: lehena ekualizadorearen imajina gordetzen duen esparruaren lehen helbidea da, bigarrena zutabearen zenbakia (0, ezkerrerengoa, …, 15 eskuinerengoa), eta hirugarrena, zutabe horretan piztuta dauden marratxoen egoera adierazten duena (00H, denak itzalita, eta FFH, denak piztuta-ren artekoa).
Honako programa honek aurretik aipaturiko funtzioa deituz, beheko irudian agertzen den erakusle grafikoaren egoera adieraziko du.

```
#define uchar unsigned char
char Egoera[16]={5,6,7,8,0,5,2..}; // piztuta dauden marratxoen kopurua

void Ekualizadorea(uchar *merak)
{
 for(zutabea=0; zutabea<16; zutabea++)
   BeteZutabea(merak, char zutabea, Egoera[zutabea]);
}
```

Ebazpena:

```
#include<reg51.h>
#define uchar unsigned char
```

```
char Egoera[16]={5,6,7,8,0,5,2,1,5,7,8,2,1,3,4,5}; // piztuta dauden marratxoen kopurua

void BeteZutabea(uchar xdata *merak,char zutabea,uchar egoera)
{
 uchar Marratxoak[9]={0,1,3,7,0x0F,0x1F,0x3F,0x7F,0xFF};

 *(merak+zutabea)=Marratxoak[egoera];
}
void Ekualizadorea(uchar xdata *merak)
{
 char zutabea;
 for(zutabea=0; zutabea<16; zutabea++)
   BeteZutabea(merak,zutabea,Egoera[zutabea]);
}
void main(void)
{
 Ekualizadorea(0x2000);
 while(1);
}
```

8.2 C programazio-hizkuntzari buruzko galderak

1)

a) Adierazi kasu bien arteko ezberdintasuna eta idatzi kasu bakoitzaren iturburu-kode egokia.
b) Zertarako erabiltzen da `register` biltegiratze mota? Noiz erabiltzen da? Eta noiz ez?
c) Noiz eta nola hasieratzen da konstante bat? Zergatik hasieratzen da horrela?
d) Idatzi eta adierazi kasu bakoitzean, `zatitu` funtzioa (lehen sarrera parametro eta bigarrenaren arteko zatiketaren emaitza itzultzen duen funtzioa) eta horri deitzen dion `main` funtzioa, non lehen kasu batean balio bidez pasatzen zaizkion eragigaiak, eta, bigarrenean, erreferentzia bidez. Gainera, marraztu kasu bakoitzaren hiru memoria-egoera, aldagai eta parametroak adierazten direlarik: justu `zatitu` deitu aurretik, justu `zatitu` funtzioak emaitza itzuli aurretik, eta `main`-en, emaitza itzuli eta gero.

Ebazpena:

a.1

```
a=fun1(x,y);
if(a==3)
 sent1;
if(a==5)
 sent2;
else
 sent3;
```

a.2

```
a=fun1(x,y);
if(a==3)
 Sent1;
else if(a==5)
 sent2;
else
 sent3;
```

a.1 sol

```
a=fun1(x,y);
switch(a)
 {
  case 3:sent1;
  case 5:sent2;
  default:sent3;
          break;
 }
```

a.2 sol

```
a=fun1(x,y);
switch(a)
 {
  case 3:sent1;
         break;
  case 5:sent2;
         break;
  default:sent3;
          break;
 }
```

a) a.1 kasuan bi if-ak balioztatzen dira: sent1 egikaritzen bada, bigarren if-a ere balioztatzen da. Ez bada ez sent1 ezta sent2 ere egikaritzen, orduan sent3 egikaritzen da. a.2 kasuan, aldiz, balioztatzen doaz if edo else-eif baten baldintza bete arte. Aurrekoa baino azkarragoa da. Hemen ere ez bada baldintza bat ere betetzen, sent3 egikaritzen da. Kasu bakoitzaren alternatiba switch-case hautapen-sententziak erabiltzea izango litzateke, **a.1 sol** eta **a.2 sol**-en agertzen den bezala.
b) Sarritan, deituak izango diren funtzioen egikaripen-abiadura handitzeko erabiltzen da. Biltegiratze mota hau aldagai lokal eta funtzioen sarrera parametroekin erabil liteke bakarrik, eta konpiladoreak bere edukiak µP-aren barne-erregistroetan gordetzen ditu, ahal duen kasu guztietan. 8051 µC-aren kasuan, barne-erregistroak barne-memorian mapatuak direnez, data

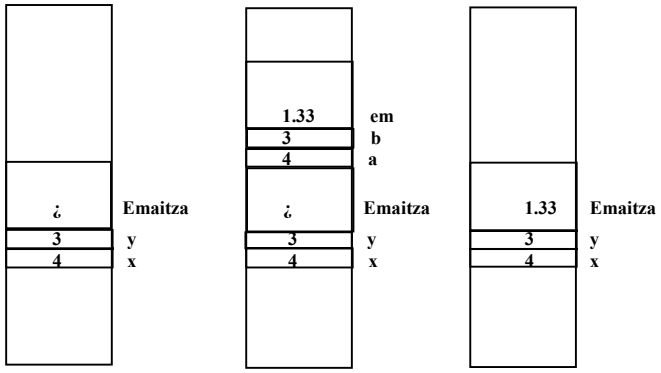
memoria mota erabiltzeko atzipen-abiadura berbera lortzen da; beraz, ez du zentzurik register erabiltzea gure kasuan, eta, haren ordez, *data* erabiliko dugu.

c) Konstante bat erazagutzen den lerro berberean hasieratzen da, zeren bestela aldagai bat izango litzateke. Hiru modu daude konstante bat erazagutzeko (lokal edo orokor gisa): const char a=2; char const a=2; eta char code a=2;

d)

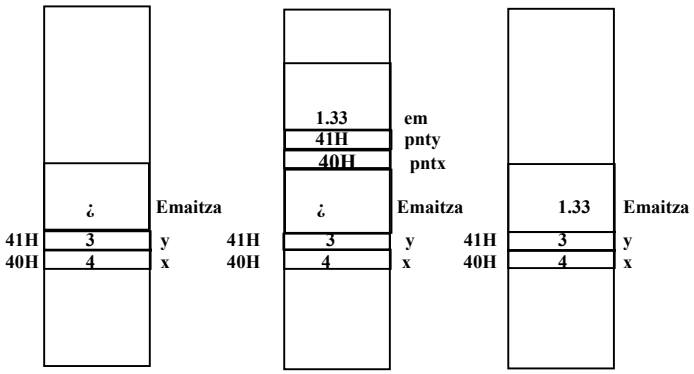
BALIO BIDEZKO PARAMETRO-PASATZEA

```
float zatitu(char a,char b)
{
 float em;

 em=(float)a/b;
 return em;
}
void main(void)
{
 char x,y;
 float Emaitza;

 x=4;
 y=3;
 Emaitza=zati(x,y);
 while(1);
}
```



ERREFERENTZIA BIDEZKO PARAMETRO-PASATZEA

```
float zatitu(char *pntx,char *pnty)
{
 float em;

 em=(float)(*pntx/*pnty);
 return em;
}
void main(void)
{
 char x,y;
 float Emaitza;

 x=4;
 y=3;
 Emaitza=zati(&x,&y);
 while(1);
}
```



2)

a) Kontuan izanda `int` 2 bytekoa eta `float` 4 bytekoa dela, tenperatura Celsius gradutan adierazteko Farenheit graduetatik abiaturik (hamartardunak), sistema batek C aldagaiaren bitartez ahal den memoria gutxien erabiliz eta doitasunik handienarekin, eta jakinda ezin duela sistemak hamartarrik adierazi, zuzendu edo utzi dagoen bezala hori egin behar duen kodea:

```
.
.
float C,F;
F=132.34;
C=(F-32)*(5/9);
.
.
```

b) Egitura bat definitua izanik, nola da posible mota honetako aldagaiak erazagutzen diren bakoitzean `struct` gako-hitza ez jartzea. Erabili puntu kartesiarraren adibidea.

Ebazpena:

a) –5/9 zatiketaren mozketa galarazi edo saihesteko, bietatik bat float bihurtzen da, eta memoriaren erabilera 2 bytetan gutxitzeko, C int motako erazagutzen da.

```
.
float F;
int C;
F=132.34;
C=(F-32)*((float)5/9);
.
```

b) Bi modu daude hau egiteko, typedef edo define erabiliz

```
typedef struct   struct point
{                {
 char x;          char x;
 char y;          char y;
}puntu;          };

puntu p1,p2;     #define puntu struct point
                  puntu p1,p2;
```

3)

a) Zer da `void` eta noiz erabiltzen da? Adierazi kasu guztiak adibide errazak erabiliz kasu bakoitzean.

b) Funtzio bati egitura motako parametroak pasatu beharko bazenizkio, nola egingo zenuke egitura horiek handiak balira? Arrazoitu erantzuna eta erabili adibideren bat.

Ebazpena:

a) void-ek adieraz lezake,
   - alde batetik, “ezer ez”; funtzioen definizio eta prototipoen erazagupenetan erabiltzen da, inongo parametrorik ez zaiola pasatzen edo/eta inongo baliorik ez duela itzultzen adierazteko.
   - bestalde, funtzioen definizio eta prototipoen erazagupenetan ere erabiltzen da, baina horiei erakusle orokorrak parametro gisa pasatzean edo/eta horiek erakusle orokorren (edozein motatakoak) bat itzultzean, non erabiltzerakoan behartuak izango baitira nahi diren motetara.

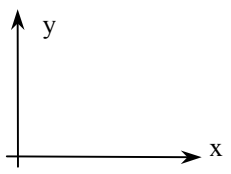
b) Egiturak handiegiak direnean balio bidez parametro gisa pasatzeko, sistemak behar den baino memoria gehiago eskatzea dakar, eta abiadura motelagoz; horregatik, soluzioa egiturak erreferentzia (erakusle) bidez pasatzea da: horrela, memoria aurrezten eta kodearen egikaripena arintzen da.

8.3 Datu-egituren ariketak

1) Idatz ezazu C programazio-hizkuntzan `ParalBol()` funtzioa, zeinek bere diagonalaren bi puntuak pasatuz parametro gisa, paralelepipedo horren bolumena kalkulatu eta itzultzen baitu. Definitu 3Dpuntu datu mota ere, hiru dimentsioko puntuaren koordenatuez osaturik, `x`, `y` eta `z` ardatzen arabera. Idatzi `ParalBol()` hori deitzen duen `main()` funtzio nagusia ere, horretarako adibide bat erabiliz.

Ebazpena:

```
#include<stdio.h>
struct 3Dpuntu
{
 float x;
```



```
 float y;
 float z;
};
float ParalBol(struct 3Dpuntu B,struct 3Dpuntu A)
{
 float bol;

 bol=(B.x-A.x)*(B.y-A.y)*(B.z-A.z);
 if(bol<0)
  bol=bol*(-1);
 return bol;
}
void main(void)
{
 struct 3Dpuntu Aptua, Bptua;
 float bolumena;

 Aptua.x=3;
 Aptua.y=1;
 Aptua.z=2;
 Bptua.x=2;
 Bptua.y=2;
 Bptua.z=5;

 bolumena=ParalBol(Bptua,Aptua);

 printf("Bolumena=%f\n",bolumena);
 getchar();
}
```

2) `x` eta `y` ardatz kartesiarrez definituriko bi dimentsioko planoan lan egiteko asmoz, `LaukizErtz()` funtzioa idaztea eskatzen da, non horrek laukizuzenaren lau ertzak itzuliko baititu, A ertza, diagonalaren angeluaren balioa (gradutan) plano horizontalarekiko, eta diagonala bera (AG) pasatzen zaizkiolarik. Gainera, funtzio hori deitzeko beharrezkoa den main(), puntu (puntu kartesiarra) egitura eta lauertz egitura ere idaztea eskatzen da, kontuan izanik egituren eremuak char datu motak osatzen dituztela.

`float sin(float x),float cos(float x)` (`x` angelua radianetan)
`struct lauertzak LaukizErtz(struct puntu A, float DiagAng, float diagonala)`

Ebazpena:

```
#include<math.h>
#include<stdio.h>
#define pi 3.1416
struct puntu
{
 int x;
 int y;
};
struct lauertzak
{
 struct puntu A;
 struct puntu B;
 struct puntu C;
 struct puntu D;
};

struct lauertzak LaukizErtz(struct puntu Ap, float DiagAng, float diagonala)
{
 struct lauertzak ema;
 int a,b;

 a=diagonala*cos(pi*DiagAng/180);
 b=diagonala*sin(pi*DiagAng/180);
```

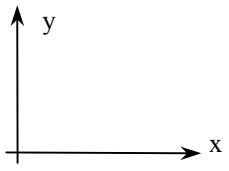
```
 printf("a=%d, y b=%d\n",a,b);
 ema.A=Ap;
 ema.B.x=ema.A.x+a;
 ema.B.y=ema.A.y;
 ema.C.x=ema.B.x;
 ema.C.y=ema.B.y+b;
 ema.D.x=ema.A.x;
 ema.D.y=ema.C.y;

 return ema;
}
void main(void)
{
 struct lauertzak emaitza;
 struct puntu Aptua;
 float ange=18.4349,diago=3.1623;

 Aptua.x=1;
 Aptua.y=1;

 emaitza=LaukizErtz(Aptua,ange,diago);
 printf("A ertza: %d,%d\n", emaitza.A.x, emaitza.A.y);
 printf("B ertza: %d,%d\n", emaitza.B.x, emaitza.B.y);
 printf("C ertza: %d,%d\n", emaitza.C.x, emaitza.C.y);
 printf("D ertza: %d,%d\n", emaitza.D.x, emaitza.D.y);
 getchar();
}
```

3) `x` eta `y` ardatz kartesiarrez definituriko bi dimentsioko planoan lan egiteko asmoz, `LaukizErtz_II()` funtzioa idaztea eskatzen da, A ertza, diagonalaren angeluaren balioa (gradutan) plano horizontalarekiko, eta diagonala bera (AG) itzultzen dituelarik, laukizuzenaren lau ertzak pasatueran egitura parametro bezala. Gainera, funtzio hori deitzeko beharrezkoa den `main()`, puntua (puntu kartesiarra), lauertz eta laukizuzen egiturak ere idaztea eskatzen da, kontuan izanik egituren eremuak char datu motak osatzen dituztela.

`float sin(float x),float cos(float x),float tan(float x),float atan(float x)` (x angelua radianetan)

Ebazpena:

```
#include<math.h>
#include<stdio.h>
#define pi 3.1416
struct puntu
{
 int x;
 int y;
};
struct lauertzak
{
 struct puntu A;
 struct puntu B;
 struct puntu C;
 struct puntu D;
};
struct laukizuzen
{
 struct puntu Aertza;
 double ange;
 double diag;
};
struct laukizuzen LaukizErtz_II(struct lauertzak ertzak)
```



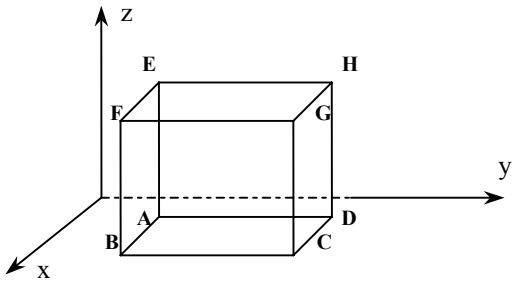
```
{
 double a,b,angelua,diagonala;
 struct laukizuzen laukiz;

 b=( ertzak.C.y-ertzak.B.y);
 a=(ertzak.B.x-ertzak.A.x);
 angelua=atan(b/a);
 diagonala=((ertzak.C.y-ertzak.B.y)/sin(angelua));
 laukiz.Aertza=ertzak.A;
 laukiz.ange=(angelua*180/pi);
 laukiz.diag=diagonala;

 return laukiz;
}
void main(void)
{
 struct lauertzak laukiz1;
 struct laukizuzen laukiz2;

 laukiz1.A.x=1;
 laukiz1.A.y=1;
 laukiz1.B.x=4;
 laukiz1.B.y=1;
 laukiz1.C.x=4;
 laukiz1.C.y=2;
 laukiz1.D.x=1;
 laukiz1.D.y=2;

 laukiz2= LaukizErtz_II(laukiz1);

 printf("A ertza:%d,%d\n angelua %lf\n diagonala %lf\n",
laukiz2.Aertza.x,laukiz2.Aertza.y,laukiz2.ange,laukiz2.diag);
}
```

4) `x,y` eta `z` ardatz kartesiarrez definituriko hiru dimentsioko planoan lan egiteko asmoarekin, `ParalHandien()` funtzioa idaztea eskatzen da, zeinek bolumen aldetik handiena den paralelepipedo baten parametroak, `ertz1paral` egituraren barnean, itzultzen baititu: A ertza, diagonalaren angeluaren balioa (gradutan) plano horizontalarekiko, eta diagonala bera (AG). Horretarako, beren zortzi ertzek definituriko bi paralelepipedo pasatuko zaizkio funtzio horri. Funtzio horretaz gain, bera deitu eta deitzeko `main( )` funtzioa eta `puntu` (puntu kartesiarra), `ertz8paral` eta `ertz1paral` egiturak idaztea ere eskatzen da.

`float sin(float x),float cos(float x),float tan(float x),float atan(float x)` (x angelua radianetan)

Ebazpena:

```
#include<math.h>
#include<stdio.h>
struct puntu
{
 int x;
 int y;
 int z;
};
struct ertz8paral
{
 struct puntu A;
 struct puntu B;
 struct puntu C;
 struct puntu D;
 struct puntu E;
```



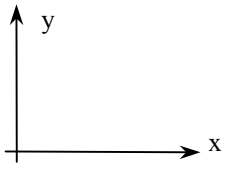
```
 struct puntu F;
 struct puntu G;
 struct puntu H;
};
struct ertz1paral
{
 struct punto Aertza;
 double ange;
 double diag;
};
struct ertz1paral ParalHandien(struct ertz8paral paral1,struct ertz1paral paral2)
{
 int ab1,ab2,bc1,bc2,cg1,cg2,ab,bc,cg;
 double bol1,bol2,ac,ag,angH,angB;
 struct ertz8paral paralH;
 struct ertz1paral ema;

 bc1=(paral1.C.y-paral1.B.y);
 ab1=(paral1.B.x-paral1.A.x);
 cg1=(paral1.G.z-paral1.C.z);
 bol1=ab1*bc1*cg1;
 bc2=(paral2.C.y-paral2.B.y);
 ab2=(paral2.B.x-paral2.A.x);
 cg2=(paral2.G.z-paral2.C.z);
 bol2=ab2*bc2*cg2;
 printf("bol1=%lf\n bol2=%lf",bol1,bol2);
 if(bol1>bol2)
  paralH=paral1;
 else
  paralH=paral2;

 bc=(paralH.C.y-paralH.B.y);
 ab=(paralH.B.x-paralH.B.x);
 cg=(paralH.G.z-paralH.C.z);

 angH=atan(bc/ab);
 ac=bc/sin(angH);

 angB=atan(cg/ac);
 ag=cg/sin(angB);

 resul.Aertza=paralH.A;
 resul.angu=(angB*180/3.1416);
 resul.diag=ag;

 return resul;
}
void main(void)
{
 struct ertz8paral parl1,parl2;
 struct ertz1paral parlH;

 parl1.A.x=3;
 parl1.A.y=1;
 parl1.A.z=1;
 parl1.B.x=3;
 parl1.B.y=3;
 parl1.B.z=1;
 parl1.C.x=0;
 parl1.C.y=3;
 parl1.C.z=1;
 parl1.D.x=0;
 parl1.D.y=3;
 parl1.D.z=1;
 parl1.E.x=3;
```

```
 parl1.E.y=1;
 parl1.E.z=3;
 parl1.F.x=3;
 parl1.F.y=3;
 parl1.F.z=3;
 parl1.G.x=0;
 parl1.G.y=3;
 parl1.G.z=3;
 parl1.H.x=0;
 parl1.H.y=3;
 parl1.H.z=3;

 parl2.A.x=4;
 parl2.A.y=1;
 parl2.A.z=1;
 parl2.B.x=4;
 parl2.B.y=3;
 parl2.B.z=1;
 parl2.C.x=0;
 parl2.C.y=3;
 parl2.C.z=1;
 parl2.D.x=0;
 parl2.D.y=3;
 parl2.D.z=1;
 parl2.E.x=4;
 parl2.E.y=1;
 parl2.E.z=3;
 parl2.F.x=4;
 parl2.F.y=3;
 parl2.F.z=3;
 parl2.G.x=0;
 parl2.G.y=3;
 parl2.G.z=3;
 parl2.H.x=0;
 parl2.H.y=3;
 parl2.H.z=3;

 parlH=ParalHandien(parl1,parl2);
 printf("Aertza:,%d,%d\n,%lf\n,%lf\n",
       parlH.Aertza.x,parlH.Aertza.y,parlH.ange,parlH.diag);
 getchar();
}
```

5) Idatzi `HiruAzalera` funtzioa, parametro moduan pasatzen zaizkion 3 elementuen, zirkulu, hiruki eta laukizuzenaren, azalerak itzultzen dituena. `main` funtzioa ere eskatzen da lehengoa probatzeko. Horretaz gain, beharrezkoa izango da honako datu-egiturak definitzea eta idaztea: `puntu` (plano kartesiarrarentzat), `zirkulu` (zentroa eta erradioa), `hiruki` (A puntua, B puntua, C puntua), `laukizuzen` (A puntua, B puntua, C puntua, D puntua) eta `hiru_azalerak` (Zazalera, Hazalera, Lazalera).

Ebazpena:

```
#include <vcl\condefs.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#pragma hdrstop

USERES("Project1.res");

#define PI 3.1416
```

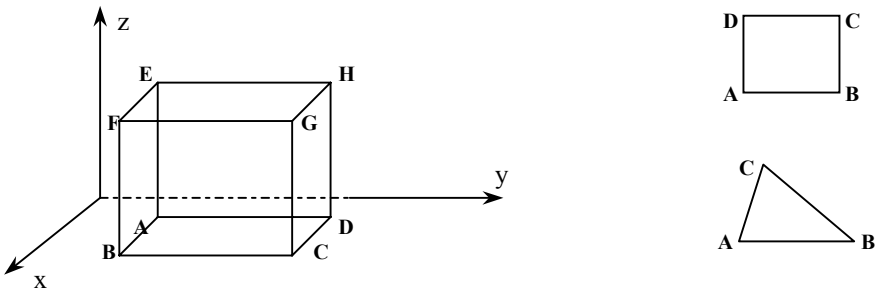


```
struct puntu
{
 char x;
 char y;
};
struct zirkulu
{
 struct puntu zentro;
 float erradio;
};
struct hirukia
{
 struct puntu A;
 struct puntu B;
 struct puntu C;
};
struct laukizuzena
{
 struct puntu A;
 struct puntu B;
 struct puntu C;
 struct puntu D;
};
struct hiru_azalerak
{
 float Zazalera;
 float Hazalera;
 float Lazalera;
};
struct hiru_azalerak HiruAzalera(struct zirkulu Z1, struct hirukia H1, struct
laukizuzena L1)
{
 struct hiru_azalerak Ema;

 Ema.Zazalera=PI*Z1.erradio*Z1.erradio;
 Ema.Hazalera=(H1.B.x-H1.A.x)*(H1.C.y-H1.A.y)/2;
 if(Ema.Hazalera<1)
  Ema.Hazalera*=(-1);
 Ema.Lazalera=(L1.B.x-L1.A.x)*(L1.C.y-L1.B.y);
 if(Ema.Lazalera<1)
  Ema.Lazalera*=(-1);
 return Ema;
}
int main(int argc, char **argv)
{
 struct zirkulu Zir;
 struct hirukia Hir;
 struct laukizuzena Lau;
 struct hiru_azalerak emaitza;

 Zir.zentro.x=0;
 Zir.zentro.y=0;
 Zir.erradio=2.34;
 Hir.A.x=1;
 Hir.A.y=1;
 Hir.B.x=3;
 Hir.B.y=1;
 Hir.C.x=2;
 Hir.C.y=3;
 Lau.A.x=1;
 Lau.A.y=1;
 Lau.B.x=3;
 Lau.B.y=1;
 Lau.C.x=3;
 Lau.C.y=3;
```

```
 Lau.D.x=1;
 Lau.D.y=3;

 rtdo=HiruAzalera(Zir,Hir,Lau);
 getchar();
 return 0;
}
```

6) Idatz itzazu `Bira90Laukiz` eta `Bira90Hiruk` funtzioak, laukizuzena eta hirukia, hurrenez hurren, erloju-noranzkoaren aurka 90 biraturik itzultzen dituztenak, eta bira puntu A ertza dutenak (kasu bietan). Horretaz gain, funtzio horiek erabili ahal izateko beharrezkoak diren egitura guztiak definitu eta idatzi, `main` funtzioaz gain.

Ebazpena:



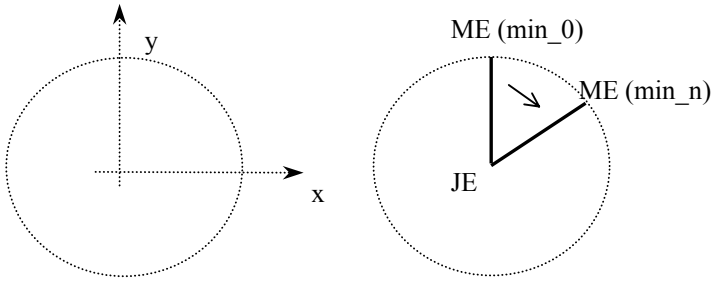

```
#include <vcl\condefs.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#pragma hdrstop
USERES("Project1.res");

#define PI 3.1416

struct puntu
{
 char x;
 char y;
};
struct hirukia
{
 struct puntu A;
 struct puntu B;
 struct puntu C;
};
struct laukizuzena
{
 struct puntu A;
 struct puntu B;
 struct puntu C;
 struct puntu D;
};
struct hirukia Bira90Hiruk(struct hirukia H1)
{
 struct hirukia Hir;
```

```
 Hir.A=H1.A;
 Hir.B.x=H1.A.x;
 Hir.B.y=H1.A.y+(H1.B.x-H1.A.x);
 Hir.C.x=Hir.A.x-(H1.C.y-H1.B.y);
 Hir.C.y=Hir.A.y+(H1.C.x-H1.A.x);

 return Hir;
}
struct laukizuzena Bira90Lauk(struct laukizuzena L1)
{
 struct laukizuzena Lau;

 Lau.A=Rec.A;
 Lau.B.x=L1.A.x;
 Lau.B.y=L1.A.y+(L1.B.x-L1.A.x);
 Lau.C.x=L1.A.x-(L1.C.y-L1.B.y);
 Lau.C.y=Lau.B.y;
 Lau.D.x=Lau.C.x;
 Lau.D.y=Lau.A.y;

 return Lau;
}
int main(int argc, char **argv)
{
 struct hirukia Hr1,HirB;
 struct laukizuzena Lk1,LauB;

 Hr1.A.x=1;
 Hr1.A.y=1;
 Hr1.B.x=3;
 Hr1.B.y=1;
 Hr1.C.x=2;
 Hr1.C.y=3;
 Lk1.A.x=1;
 Lk1.A.y=1;
 Lk1.B.x=3;
 Lk1.B.y=1;
 Lk1.C.x=3;
 Lk1.C.y=3;
 Lk1.D.x=1;
 Lk1.D.y=3;

 HirB=Bira90Hiruk(Tr1);
 LauB=Bira90Lauk(Rc1);
 getchar();
 return 0;
}
```

7) Idatzi `Minutu-orratz` funtzioa, minutu-orratza n minutu biratuta, ordulariaren zentzuan, itzultzen duena (n 0 eta 59 bitartean), JE hartuz biratzeko jatorri bezala (1 eremua), eta ME (2 eremua) biraturiko puntu bezala, non 0 minutua eta n parametroa (osokoa) pasatzen zaizkion. Horrez gain, beharrezkoa izango da eskatzen den funtzioa erabil dadin beharrezkoak diren datu-egitura guztiak definitu eta idaztea, `main` funtzioaz gain. Minutu-orratz funtzioak eskaturiko minutua itzultzen du errorea (3. eremua) 0 duelarik, edo 0 minutua 1 errorearekin, baldin eta tartez kanpoko minuturen bat eskatzen bazaio.

`float sin(float x),float cos(float x),float tan(float x),float atan(float x)` (x angelua radianetan)

Ebazpena:



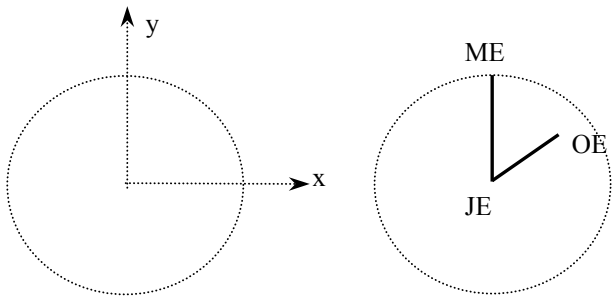

```
#include <math.h>
#define pi 3.1416
struct puntu
{
 float x;
 float y;
};
struct minutu-orratz
{
 struct punto JE;
 struct punto ME;
 int errore;
};
struct Minutu-orratz(struct minutu-orratz min0,int n)
{
 float erradio,a,b,n_rad;
 struct minutu-orratz ema;
 erradio=min0.ME.y-min0.JE.y;
 if(0<=n&&n<=59)
 {
  n_rad=((90-n*6)*pi)/180;
  ema.VM.x=erradio*cos(n_rad);
  ema.VM.y=erradio*sen(n_rad);
  ema.OM=min0.OM;
  ema.errore=0;
 }
 else
 {
  ema=min0;
  ema.errore=1;
 }
 return ema;
}
void main(void)
{
 struct minutu-orratz m0,emaitza;
 int m;

 m0.JE.x=0;
 m0.JE.y=0;
 m0.ME.x=0;
 m0.ME.y=5;
 m=4;
 emaitza=Minutu-orratz(m0,m);
}
```

8) Egunaren minutuak, 00:00etatik hasita 23:59raino (non berriro 0-tik hasten den), itzultzen dituen sistema bat dugu. Idatz ezazu orduen eta minutuen adierazleak sistemak pasaturiko minutu kopuruaren arabera biratuta itzultzen dituen `OrduMinutu` funtzioa. Horrela bada, funtzioak minutu-orratzaren

ertza, EM, 3ko balioa duen erradioa erabiliz kalkulatua, eta ordu-orratzarena, OE, 2ko erradioa erabiliz kalkulatuko du. JE ertza komuna da orratz bientzat (koordenatuen jatorria), eta laugarren eremu bat ere egongo da, PM, egunaren lehen erdia (0) eta bigarrena (1) bereizteko. Orduan, `OrduMinutu` funtzioak sistemak pasatzen dion minutu kopuruari dagozkion JE, ME, OE eta PM eremuak dituen egitura itzuliko du. Idatz itzazu behar diren egiturak eta eskaturiko `OrduMinutu` funtzioaren definizioa, `main` nagusiaz gain, adibide erraz batekin probatzen delarik dena.

`float sin(float x),float cos(float x),float tan(float x),float atan(float x), math.h` liburutegiko funtzioak erabil daitezke (x angelua radianetan)

Ebazpena:




```
#include <math.h>
#define pi 3.1416
struct puntu
{
 float x;
 float y;
};
struct ordumin
{
 struct puntu JE;
 struct puntu ME;
 struct puntu OE;
 char PM;
};
struct ordumin OrduMinutu(int n)
{
 int ordu,min;
 float ordu_rad,min_rad;
 struct ordumin ema;

 ordu=n/60;
 min=n%60;

 ordu_rad=((90-ordu*30)*pi)/180; // ordu bakoitzari 6°x5=30° dagozkio
 ema.OE.x=2*cos(ordu_rad);
 ema.OE.y=2*sen(ordu_rad);
 min_rad=((90-min*6)*pi)/180; // minutu bakoitzari 6° dagozkio
 ema.ME.x=3*cos(min_rad);
 ema.ME.y=3*sen(min_rad);
 ema.JE.x=0;
 ema.JE.y=0;
 if(ordu<12)
  ema.PM=0;
 else
  ema.PM=1;
 return ema;
 }

void main(void)
```

```
{
 struct ordumin emaitza;
 int sistema_minutuak;

 sistema_minutuak=4;
 emaitza=OrduMinutu(sistema_minutuak);
}
```

9) Idatzi `Minutu-orratz_II` funtzioa, non pasatzen zaion minutu-orratzaren posizioarekin horrek adierazten duen minutuaren n (0 eta 59 bitartean) balioa itzultzen duela. Posizioak izatez bi eremu ditu, hots minutu-orratzaren ertz biak: koordenatu jatorria, JE, eta orratzaren ertz higikorra, EM. Idatz itzazu behar diren egiturak ere, `main()` nagusiaz gain, non adibide erraz batekin probatzen den dena.

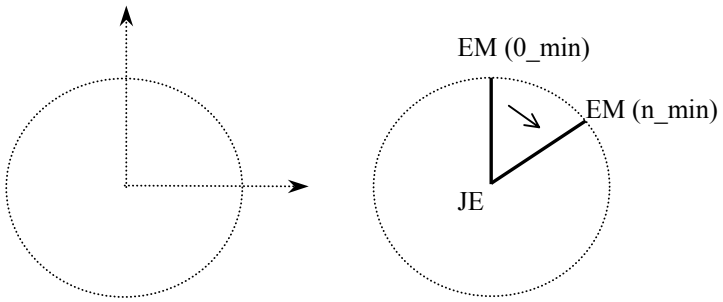
$\sqrt{x}$ → `float sqrt(float x), float sin(float x), float cos(float x), float tan(float x), float acos(float x), float asin(float x), float atan(flota x)` `math.h` liburutegiko funtzioak erabil daitezke (x angelua radianetan).

1. OHARRA: arku-funtzio trigonometriko hauek eta kalkulagailuenak eman ditzaketen bi angeluetariko soluzio bakar bat ematen dute. Ematen ez duten soluzioa parentesi artean agertzen da.

`float acos(float x)`, x>0 bada->**angelua 1.K**(angelua 4.K)
`float acos(float x)`, x<0 bada->**angelua 2.K**(angelua 3.K)
`float asin(float x)`, x>0 bada->**angelua 1.K**(angelua 2.K)
`float asin(float x)`, x<0 bada->**angelua 4.K**(angelua 3.K)
`float atan(float x)`, x>0 bada->**angelua 1.K**(angelua 3.K)
`float atan(float x)`, x<0 bada->**angelua 4.K**(angelua 2.K)

2. OHARRA: orratzaren angelua > 90º bada, angelua negatibo gisa adierazi behar da (360º kenduz), minutuak beti positiboak izan daitezen.

Ebazpena:




```
#include <vcl\condefs.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#pragma hdrstop
#define pi 3.1416
struct puntu
{
 float x;
 float y;
};
struct min
{
```

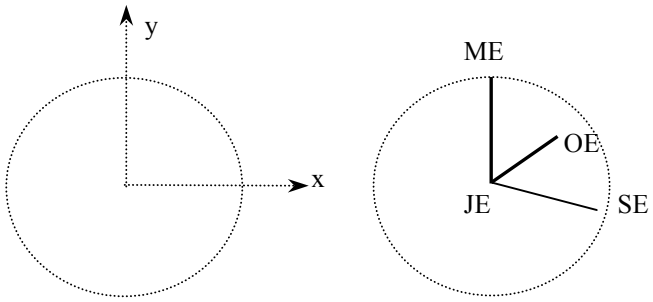
```
 struct puntu JE;
 struct puntu ME;
 int errore;
};
struct min Minutu-orratz(struct min min0,int n)
{
 float radioa,a,b,n_rad;
 struct min ema;
 radioa=min0.ME.y-min0.JE.y;
 if(0<=n&&n<=59)
 {
  n_rad=((90-n*6)*pi)/180;
  ema.ME.x=radioa*cos(n_rad);
  ema.ME.y=radioa*sin(n_rad);
  ema.JE=min0.JE;
  ema.errore=0;
 }
 else
 {
  ema=min0;
  ema.errore=1;
 }
 return ema;
}
int Minutu-orratz_II(struct min min_n)
{
// int min;
 float a,b,radioa,aco,asi,n_rad,minu;

 a=min_n.ME.x;
 b=min_n.ME.y;
 radioa=sqrt(a*a+b*b);
 aco=acos(a/radioa);
 asi=asin(b/radioa);
 if(a>0&&b>=0)//1°C
  n_rad=aco;// edo n_rad=asi;
 else if(a<=0&&b>=0)//2.K
  n_rad=aco;
 else if(a<=0&&b<0)//3.K
  n_rad=pi-asi;
 else if(a>0&&b<0)//4.K
  n_rad=asi;
 if(n_rad>pi/2) // n_rad>pi/2 bada, angelu negatibo
   n_rad=n_rad-2*pi; //moduan adierazi behar da
 minu=(90-n_rad*180/pi)/6;
 return minu;
}
struct min m0,Ema;

USERES("Project1.res");

int main(int argc, char **argv)
{
 int m;
 float mi;

 m0.JE.x=0;
 m0.JE.y=0;
 m0.ME.x=0;
 m0.ME.y=5;
 for(m=0;m<60;m+=5)
 {
  Ema=Minutu-orratz(m0,m);
  mi=Minutu-orratz_II(Ema,5);
  printf("minutua %lf da\n",mi);
```

```
 }
 getchar();
 return 0;
}
```

10) Egunaren segundoak, 00:00etatik hasita 23:59raino (non berriro 0tik hasten den) zenbatzen dituen sistema bat dugu. Idatz ezazu orduen, minutuen eta segundoen adierazleak itzultzen dituen `OrduMinSegu` funtzioa, sistemak pasaturiko segundo kopuruaren arabera biratuta. Horrela bada, funtzioak segundo-orratzaren ertza, SE, SR erradioarentzako kalkulatua, minutu-orratzaren ertza, ME, MR erradioarentzako kalkulatua, eta ordu-orratzarena, OE, OR erradioa erabiliz kalkulatuko du. JE ertza, komuna da hiru orratzentzat (koordenaden jatorria), eta bosgarren eremu bat ere egongo da, PM, egunaren lehen erdia (0) eta bigarrena (1) bereizteko. Orduan, `OrduMinSegu` funtzioak sistemak pasatzen dion segundo kopuruari dagozkion JE, SE, ME, OE eta PM eremuak dituen egitura itzuliko du. Idatz itzazu behar diren egiturak eta eskaturiko `OrduMinSegu` funtzioaren definizioa, `main` nagusiaz gain, adibide erraz batekin probatzen delarik dena.

`float sin(float x),float cos(float x),float tan(float x),float atan(float x), math.h` liburutegiko funtzioak erabil daitezke (x angelua radianetan)

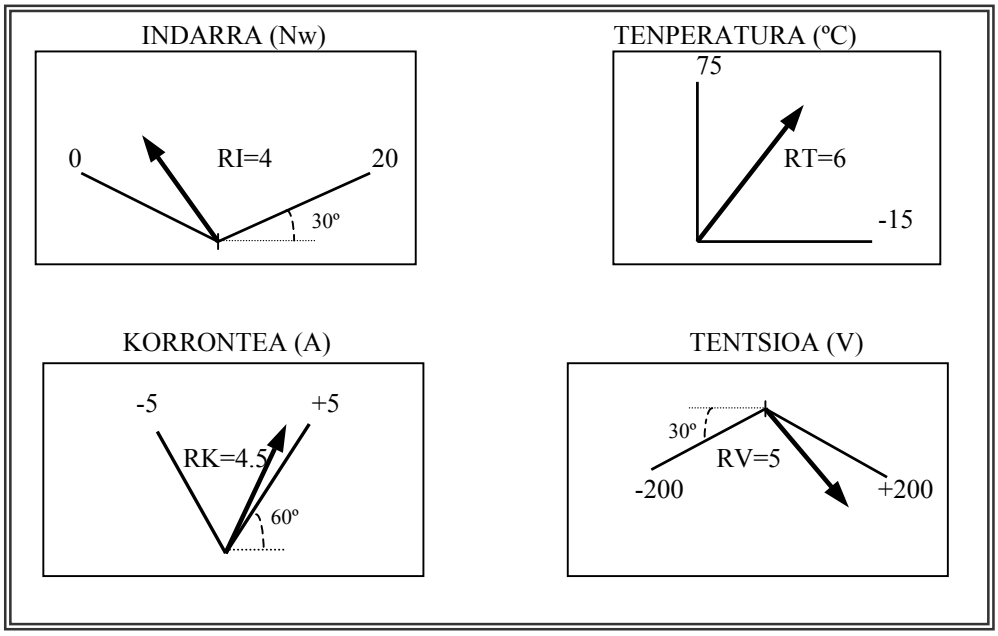
Ebazpena:




```
#include <math.h>
#define pi 3.1416
struct puntu
{
 float x;
 float y;
};
struct oms
{
 struct puntu JE;
 struct puntu SE;
 struct puntu ME;
 struct puntu OE;
 int PM;
};
struct oms OrduMinSegu(long seg, int SR, int MR, int OR)
{
 struct oms ema;
 float o_rad,m_rad,s_rad;
 int orduak,minutuak,segundoak;

 orduak=segunduok/3600;
 minutuak=segundoak/60;
 segundoak=segundoak-orduak*3600-minutuak*60;

 o_rad=((90- orduak *30)*pi)/180;
 ema.OE.x=OR*cos(o_rad);
 ema.OE.y=OR*sen(o_rad);
 o_rad=((90-minutuak*6)*pi)/180;
```

```
 ema.ME.x=ME*cos(m_rad);
 ema.ME.y=ME*sen(m_rad);
 s_rad=((90-orduak*6)*pi)/180;
 ema.SE.x=SR*cos(s_rad);
 ema.SE.y=SR*sen(s_rad);
 ema.ME.x=0;
 ema.ME.y=0;
 if(0<=orduak&&orduak<=11)
  ema.PM=0;
 else
  ema.PM=1;
 return ema;
}
void main(void)
{
 struct oms emaitza;
 int sg, sr,mr,or;

 sr=5;
 mr=4;
 or=2;
 sg=4000;
 emaitza=OrduMinSegu(sg,sr,mr,or);
}
```

11) Sare elektrikora energia itzultzeko gai den inbertsore eta korronte zuzeneko motor bikotean oinarrituriko industria-makina bat dugu. Neurriak hartzen dituen zirkuitu honetako zatiak aplikatzen den Indarra eta piezaren Tenperatura, inbertsorearen bataz besteko Korronte eta tentsioen balioen laginak hartzen ditu periodikoki (timerraren etendura bidez), guk `main`-etik behin bakarrik deituko duen `AdieraziITKV` funtzioren kasua egingo dugun arren. Idatzi beharrezkoak diren egiturak eta `AdieraziITKV` funtzioa, 4 aldagai horien balioak pasatueran, 4 adierazleen bakoitzeko ertz biak itzultzen dituena instrumentazioko panel birtualera, irudiak agertzen duen bezala, eta gainera beste hirugarren atal bat ere bai (aldagai bakoitzeko), `max` izendatua, adierazlea eskalatik kanpo geratzen den kasuetarako (+1 goitik bada, -1 behetik bada, edo bestela, 0). Idatzi main funtzio nagusia ere, kasu bat ikusteko.



Ebazpena:

```
#include <math.h>
#define pi 3.1416

struct puntu
{
 float x;
 float y;
};
struct adierazle
{
 struct puntu J;
 struct puntu E;
 int max;
};
struct Lau_adierazle
{
 struct adierazle I;
 struct adierazle T;
 struct adierazle K;
 struct adierazle V;
};
struct Lau_adierazle AdieraziITKV(float i,float t,float k,float v)
{
 Lau_adierazle ema;
 float i_rad,t_rad,k_rad,v_rad,RI=4,RT=6,RK=4.5,RV=5;
 int maxi;

 i_rad=(150-i*6)*pi/180;
 t_rad=(15+t*1)*pi/180;
 k_rad=(90-k*3)*pi/180;
 v_rad=(-90+v*0.3)*pi/180;

 res.I.J.x=0;
 res.I.J.y=0;
 res.I.E.x=RI*cos(i_rad);
 res.I.E.y=RI*sin(i_rad);
 if(i>20)
   maxi=1;
 else if(i<0)
   maxi=-1;
 else
   maxi=0;
 res.I.max=maxi;

 res.T.J.x=0;
 res.T.J.y=0;
 res.T.E.x=RT*cos(t_rad);
 res.T.E.y=RT*sin(t_rad);
 if(t>75)
   maxi=1;
 else if(t<-15)
   maxi=-1;
 else
   maxi=0;
 res.T.max=maxi;

 res.K.J.x=0;
 res.K.J.y=0;
 res.K.E.x=RI*cos(k_rad);
 res.K.E.y=RI*sin(k_rad);
 if(k>5)
   maxi=1;
 else if(k<-5)
```

```
   maxi=-1;
 else
   maxi=0;
 res.K.max=maxi;

 res.V.J.x=0;
 res.V.J.y=0;
 res.V.E.x=RV*cos(v_rad);
 res.V.E.y=RV*sin(v_rad);
 if(v>200)
   maxi=1;
 else if(v<-200)
   maxi=-1;
 else
   maxi=0;
 res.V.max=maxi;

 return ema;
}
int main(int argc, char **argv)
{
 float in=17,te=54,ko=3.232,vv=184.579;
 struct Lau_adierazle Emaitza;

 Emaitza= AdieraziITKV(in,te,ko,vv);
 return 0;
}
```

8.4 Helbide–deskodeketei buruz proposaturiko ariketak

1) 8088 µPan oinarrituriko sistema baten memoria-mapa, egia-taula eta eskema marraztea eskatzen da. SRAM motako 64 kilobyteko memoria esparrua nahi da osatu, 20000H helbidetik aurrera (gorantz), 8 kilobyteko 8 txip eta deskodetzaile komertzial edo/eta ate logikoak erabiliz.

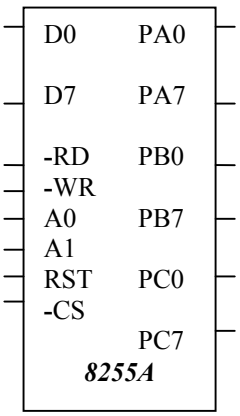
2) 8088 µPan oinarrituriko sistema baten memoria-mapa, egia-taula eta eskema marraztea eskatzen da. SRAM motako 32 kilobyteko memoria esparrua nahi da osatu, E8000H helbidetik aurrera (gorantz) 4 kilobyteko 8 txip eta deskodetzaile komertzial edo/eta ate logikoak erabiliz.

3) 8088 µPan oinarrituriko 256 kilobyteko programa-memoria duen (32 kilobyteko txipak), 2 zonaldearen azpialdean (FFFFF-E0000H), sistema baten memoria-mapa, egia-taula eta eskema marraztea eskatzen da.

4) 8088 µPan oinarrituriko 64 kilobyteko datu-memoria duen (8 kilobyteko txipak) sistema baten memoria-mapa, egia-taula eta eskema marrazteko eskatzen da. Ariketa datu-memoriari dagokion diseinua da soilik, eta goi-zonan dagoen 256 kilobyteko programa-memoriaren azpialdean kokatzen da.

5) 8088 µPan oinarrituriko sistema batean 4 I/O pin-hedatzaile (8255 txipa) kokatu nahi dira modu jarraian I/O esparruko beheko zonaldearen ondoren (gorantz). Jakinik aipaturiko beheko zonalde hori beste gailu batzuentzako hartua dela eta guztira I/O esparruko lehen 4 kilobyteak direla, zera eskatzen da: 4 hedatzaileen kokapenari dagokion I/O-mapa, egia-taula eta eskema marraztea. Ahal denik eta ate logiko gutxien erabili.

2tik 4rako deskodetzailea:
*A1, A0
*-S0, -S1, -S2, -S3
*-EN



A1	A0	Funtzioa
0	0	A Portua
0	1	B Portua
1	0	C Portua
1	1	Kontrol-erregistroa

6) 8088 µPan oinarrituriko sistema batean 4 I/O pin-hedatzaile (8255 txipa) kokatu nahi dira modu jarraian I/O esparruko goiko zonaldean. Diseinuz kanpo geratzen da gainerako zonaldea. Zera eskatzen da: 4 hedatzaileen kokapenari dagokion I/O-mapa, egia-taula eta eskema marraztea. Ahal denik eta ate logiko gutxien erabili.

7) 8088 µPan oinarrituriko sistema batean 4 txip, SRAM motakoak, nahi dira kokatu memoria-esparruan: gorengo helbidetik hasita beherantz eta denak jarraian; 8 kilobyteko txip 1, 4 kilobyteko beste 1 eta 2 kilobyteko txip 2. Zera eskatzen da: sistemaren memoria-mapa, egia-taula eta eskema marraztea, lau txip hauen kokatzeari dagokienez. Ahal denik eta ate logiko gutxien erabili.

3tik 8rako deskodetzailea:
*A0, A1, A2
*-S0, -S1, -S2, …-S7
*EN

8) 8088 µPan oinarrituriko sistema batean 3 txip nahi dira kokatu memoria-esparruan: gorengo helbidetik hasita beherantz, 16 kilobyteko EPROM 1, 4 kilobyteko SRAM 1 eta 8 kilobyteko SRAM 1. Zera eskatzen da: sistemaren memoria-mapa, egia-taula eta eskema marraztea, hiru txip hauen kokatzeari dagokionez. Ahal denik eta ate logiko gutxien erabili.

3tik 8rako deskodetzailea:
*A0, A1, A2
*-S0, -S1, -S2, …-S7
*EN

9) 8088 µPan oinarrituriko sistema batean 3 txip nahi dira kokatu memoriako gorengo 32 kilobyteko esparruan (A zonaldea): gorengo helbideetan 8 kilobyteko EPROM 1, eta lehengo helbidetik azkenerantz edo gorantz, 16 kilobyteko SRAM 1 eta 4 kilobyteko SRAM.

N-tik 2^N-rako deskodetzailea:
*A0..A(N-1)
*-S0,..-S(2^N -1)
*-EN

9. ERREFERENTZIA BIBLIOGRAFIKOAK

[1] Gil Padilla, A. J. ELECTRÓNICA GENERAL, DISPOSITIVOS Y SISTEMAS DIGITALES. Editorial McGrawHill, 1990.

[2] Angulo, J. M. MICROPROCESADORES, FUNDAMENTOS, DISEÑO Y APLICACIONES. Editorial Paraninfo, 1989.

[3] Philips Components. Integrated Ciruits Data Handbook: 8051-based-8-bit Microcontrollers. Philips, 1990.

[4] Barrón Ruiz, M. Y Martínez Pérez, J. APLICACIONES PRÁCTICAS CON EL µC8051, PROGRAMACIÓN EN LENGUAJE C. Impreso en Michelena Artes Gráficas, 1999.

[5] González Vázquez, J. A. INTRODUCCIÓN A LOS MICROCONTROLADORES. Editorial McGraw-Hill, 1993.

[6] Tavernier, C. MICROCONTROLADORES DE 4 Y 8 BITS. Editorial Paraninfo, 1997.

[7] Brian W. Kernighan, Dennis M. Ritchie. EL LENGUAJE DE PROGRAMACIÓN C. Prentice-Hall Hispanoamericana, 1989

[8] Schildt, H. C++ PARA PROGRAMADORES. Editorial McGraw-Hill, 1996.

[9] Leblanc, G. BORLAND C++ BUILDER. Ediciones Eyrolles, 1997.

[10] Carballar, J. A. SOFTWARE Y HARDWARE DE SU PC. Editorial RA-MA,1994.

[11] Brey, B. B. LOS MICROPROCESADORES INTEL, 3ª Edición. Editorial Prentice Hall, 1995.

[12] Berral Montero, I. EQUIPOS MICROINFORMÁTICOS Y TERMINALES DE TELECOMUNICACIÓN. Editorial Paraninfo, 2000.

[13] Cembranos Nistal, F. J. INFORMÁTICA INDUSTRIAL. Editorial Paraninfo, 2000.

[14] Keil Software. 8051 Demo Kit, µVision User´s Guide.