TALLINNA TEHNIKAÜLIKOOL
ELEKTRIAJAMITE JA JÕUELEKTROONIKA INSTITUUT
ROBOTITEHNIKA ÕPPETOOL

MIKROPROTSESSORTEHNIKA

TÕNU LEHTLA LEMBIT KULMAR

Tallinn
1995

T Lehtla, L Kulmar. Mikroprotsessortehnika
TTÜ Elektriajamite ja jõuelektroonika instituut. Tallinn, 1995. 141 lk

Toimetanud Juhan Nurme
Kujundanud Ann Gornischeff

Autorid tänavad TTÜ arvutitehnika instituudi lektorit Toomas Konti ja sama instituudi dotsenti Vladimir Viiest raamatu käsikirjas tehtud paranduste ja täienduste eest.

© T Lehtla, L Kulmar, 1995
© TTÜ elektriajamite ja jõuelektroonika instituut, 1995
Kopli 82, 10412 Tallinn
Tel 620 3704, 620 3700. Faks 620 3701

ISBN 9985-69-006-0

TTÜ trükikoda. Koskla 2/9, Tallinn EE0109
Tel 552 106

Sisukord

	Saateks	5
	Digitaal- ja mikroprotsessortehnika arengut kajastavaid aastaarve	6
1.	**DIGITAALELEKTROONIKA ALUSED**	7
1.1.	**Diskreetsed ja arvsignaalid**	7
1.1.1.	Kvantimine	7
1.1.2.	Kodeerimine, dekodeerimine ja koodide liigid	8
1.1.3	Kümnendarvude teisendamine kahend-, kaheksand- ja kuueteistkümnendarvudeks	12
1.1.4.	Informatsiooni hulk ja signaali viga	13
1.2.	**Loogikafunktsioonid ja loogikalülitused ning nende esitusviisid**	14
1.2.1.	Loogikatehted	14
1.2.2.	Loogikaseadused	17
1.2.3.	Loogikalülituste süntees ja minimeerimine	21
1.3.	**Funktsionaalsed loogikalülitused**	24
1.3.1.	Trigerid	24
1.3.2.	Registrid	27
1.3.3.	Loendurid	28
1.3.4.	Summaatorid	31
1.3.5.	Kommutaatorid	34
1.3.6.	Aritmeetika-loogikaplokk	36
1.3.7.	Koodrid ja dekoodrid	37
1.4.	**Homogeensed struktuurid ja loogilised maatriksid**	40
1.4.1.	Loogilised maatriksid	40
1.4.2.	Ümberprogrammeeritavad maatriksid	43
1.5.	**Mälud**	44
1.5.1.	Muutmälud	45
1.5.2.	Püsimälud	47
1.6.	**Diskreetsed automaadid**	48
1.6.1.	Diskreetsete automaatide olemus	48
1.6.2.	Algoritmide aparatuurne realiseerimine	51
1.6.3.	Programm- ja mikroprogrammjuhtimine	57
1.6.4.	Algoritmide programmiline realiseerimine	60

2. MIKROPROTSESSORID 61

2.1. Mikroprotsessorite ja arvutite ehitus 61
2.1.1. Põhimõisted 61
2.1.2. Arvuti põhiplokkid ja siinid 63
2.1.3. Töötsüklid 65

2.2. Mikroprotsessori tööpõhimõte 67
2.2.1. Protsessori ehitus 67
2.2.2. Registrid ja nende otstarve 68
2.2.3. Ajadiagrammid 71
2.2.4. Käsu- ja andmevormingud 72
2.2.5. Protsessori käsustik 75
2.2.6. Adresseerimine 77
2.2.7. Pinumälu 79
2.2.8. Protsessori koostöö mälu ja välisseadmetega 79

2.3. Andmevahetus 82
2.3.1. Andmevahetuse meetodid 82
2.3.2. Rööpvärat 87
2.3.3. Jadavärat 90
2.3.4. Taimer 91
2.3.5. Otsemällupöördus ja DMA-kontroller 96

2.4. Tarkvara 98
2.4.1. Ülevaade mikroarvutite ja juhtraalide tarkvarast 98
2.4.2. Assembler 99
2.4.3. Intel 8080 assemblerikeel 101

2.5. Signaaliprotsessorid 105
2.5.1. Signaaliprotsessorite ehituse iseärasused 105
2.5.2. Digitaal-analoogmuundurid 106
2.5.3. Analoog-digitaalmuundurid 107
2.5.4. Transpuutrid 111

2.6. Mikroprotsessorite kasutamine 114
2.6.1. Ülevaade 114
2.6.2. Mikroprotsessorid elektriajamis 115
2.6.3. Mikroprotsessorid releekaitses 119

LISA. Programmeeritavad kontrollerid *SIMATIC* S5 127

Kirjandus 141

Saateks

Mikroprotsessortehnika on lühikese aja jooksul levinud peaaegu kõikidesse inimtegevuse valdkondadesse. Arvutid on muutunud igapäevaseks töövahendiks, mikroprotsessoreid kasutatakse juba kodumasinates, üha harvemini puutume kokku tööpinkidega, kuhu mikroprotsessorid pole veel jõudnud. Digitaal- ja mikroprotsessortehnika areng jätkub peadpööritava kiirusega. Vaevalt leidub teist tehnikaala, kus seadmete töökiirus ja efektiivsus, hind ja mõõtmed muutuksid mõne aasta jooksul 10 ja enam korda.

Mikroprotsessorist on saanud inseneride käes universaalne vahend paljude probleemide lahendamiseks. Energia- ja tööstusseadmete juhtimine, kontrolli-, mõõte-, bürootehnika jms on vaid väike loetelu aladest, kus tänapäeval mikroprotsessoriteta läbi ei saada. Kooselu arvutitega on muutnud meie arusaamu ja mõttemalle. Pidevatoimeliste protsesside juhtimiseks kasutatakse üha enam diskreetseid juhtseadmeid, sest nii paraneb süsteemi tundlikkus, täpsus ning töökindlus. Signaalide kvantimine ja kodeerimine võimaldab rakendada numbrilise infotöötluse meetodeid ja programmjuhtimise põhimõtteid ning kasutada selleks universaalseid mikroprotsessorseadmeid. Selle tulemusena vähenevad seadmete mõõtmed, mass ja hind.

Tänapäva arvutustehnika on kujunenud hiigelpüramiidiks, mille vundament on mikro-protsessorid, selle peale aga kerkivad üha uued tarkvarakorrused. Paljukihiline hierarhiline tarkvarasüsteem on lahutanud inimese protsessorist ning vaevalt suudab arvutiklahvistikul klõbistav operaator tunnetada oma tegevuse seost protsessori registrite ja siinidega ning kahendsõna bittide ja baitidega. Universaalarvutite riist- ja tarkvara arendavad tänapäeval vähesed tippspetsialistid, nende tööd kasutavad peaaegu kõik. Ja vaevalt et enamikule arvutioperaatoreist pakub lähemat huvi mikroprotsessorite ehitus

Tehniliste seadmete ja tehnoloogiaprotsesside juhtimisel on riist- ja tarkvaraprobleemid sageli spetsiifilised ning üldlahendid puuduvad. Programmeerijalt eeldatakse riistvara ehituse tundmist. Tööstuslikku juhtimissüsteemi projekteeriv insener peab aga tundma mikrokontrollerite spetsiifilisi detaile, sisend-väljundliideste omadusi ja mälu ning protsessori töö iseärasusi. See on põhjus, miks automaatikasüsteemide insener vajab algteadmisi mikroprotsessortehnikast.

Digitaal- ja mikroprotsessortehnika on kahtlemata üheks tänapäeva inseneriharidu se nurgakiviks. Digitaaltehnika aluste omandamine annab üliõpilasele võimaluse paremini mõista seda, kuidas funktsioneerib nüüdistehnika ja tehnoloogia; aitab mõista tehnika arengut ning inimese ja tehnika vahelist suhet nüüd ja tulevikus; arendab süsteemset mõtlemist ning kahandab aukartust üle mõistuse keerukana tunduvate aparaatide ja seadmete ees.

Käesolev raamat on mõeldud õppevahendiks energiatehnika õppevaldkonna üliõpilastele, kuid see võib osutuda kasulikuks ka teiste tehnikaerialade üliõpilastele.

Autorid

Digitaal- ja mikroprotsessortehnika arengut kajastavaid aastaarve:

1948	esimene bipolaarne transistor
1948	J von Neumani digitaalarvuti struktuur ja tööpõhimõte
1959	esimene integraallülitus, mille valmistasid J S Kilbi ja R Noyce
1960-64	integraalloogikaelementide väljaarendamine
1961	esimene kommertsotstarbeline integraallülitus
1962	esimene *MOS* integraallülitus
1964	integraallülituste 14 jalaga *DIP* (*dual in plane*) tüüpi kiibi väljatöötamine
1965	*MOS* integraallülituste tööstuslik tootmine
1967	256-bitine püsimälukiip
1969	firma *Intel* alustab mikroprotsessorite projekteerimist
1970	firma *IBM* ümbrikketasmäluseade
1971	esimene 4-bitine mikroprotsessor *Intel* 4004
1972	firma *Intel* 1024-bitine muutmälukiip
1972-73	8 bitiste n-*MOS* mikroprotsessorite tootmise algus
1973	firma *Intel* 8-bitine mikroprotsessor 8080
1974	firma *Motorola* 8-bitine mikroprotsessor 6800
1974	esimesed 12- ja 16-bitised mikroprotsessorid
1976	esimene ühekiibiarvuti
1976	firma *Zilog* 8-bitine mikroprotsessor Z80
1976	firma *Intel* 16 384-bitine muutmälukiip
1978	esimene 16-bitine firma *Intel* mikroprotsessor 8086
1979	firma *Motorola* 16-bitine protsessorikiip 68000
1979	firma *Intel* mikroprotsessor 8088
1980	firma *Intel* matemaatikaprotsessor 8087
1982	esimene 32-bitine firma *Hewlett-Packard*'i protsessorikiip
1982	firma *Intel* mikroprotsessor 80286 taktsagedusega 8...16 MHz
1985	firma *Inmos Ldt.* transpuuter T400
1985	firma *Intel* 32-bitine mikroprotsessor 80386 taktsagedusega 16...40 MHz
1989	firma *Intel* mikroprotsessor 80486 taktsagedusega 25...66 MHz
1990	antakse välja hilisemat poleemikat põhjustanud USA patent nr 4 942 516 mikroprotsessori (*Single Chip Integrated Circuit Computer Architecture*) leiutamise kohta Gilbert Hyattile (avaldus 1970. aastast)
1992	firma *Intel* mikroprotsessor 80486DX2
1993	firma *Intel* mikroprotsessor *Pentium* taktsagedusega vähemalt 60 MHz

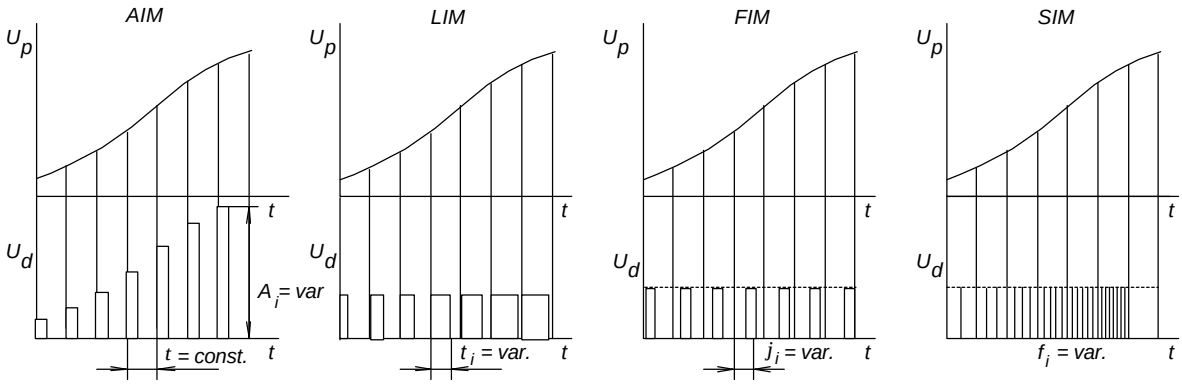
1. DIGITAALELEKTROONIKA ALUSED

1.1. Diskreetsed ja arvsignaalid

1.1.1. Kvantimine

Kvantimine tähendab klassikaliselt füüsikateoorialt kvantteooriale siirdumise menetlust. Informaatikas on **kvantimine** signaalitöötluse operatsioon, millega pidevale signaalile omistatakse kindlaks ajavahemikuks diskreetne väärtus. Kvantimine toimub nii signaali nivoo järgi kui ka ajas. Lisagem, et **signaal** on sõnumi (informatsiooni) füüsikaline kandja. Sõltuvalt füüsikalisest olemusest liigitatakse signaale pneumo-, hüdro-, elektri-, valgus- jms signaalideks. Mikroprotsessortehnikas käsitletakse peamiselt elektrisignaale, kuid erijuhtudel ka optilisi ehk valgussignaale.

Suur osa looduslikest ja tehisprotsessidest on pidevatoimelised, s. t neid iseloomustavad pidevad olekusignaalid, mida saab mõõta või hinnata suvalisel ajahetkel. Pidevatoimelisi signaale nimetatakse neid töötlevate (analoog)seadmete järgi **analoogsignaalideks**. Mikroprotsessortehnika põhineb **diskreet- ehk katkelistel signaalidel**, millele omistatakse väärtus ainult kindlail ajahetkeil. Diskreetsignaalid jagunevad impulss- ja arvsignaalideks. **Impulss-signaalides** kodeeritakse informatsiooni impulsi parameetritega. Impulsi olulisemad parameetrid on amplituud (A_i) ehk kõrgus, kestus (t_i) ehk laius, sagedus (f_i) või periood (t_i) ja faasinurk (j_i) ehk nihe taktiimpulsi suhtes. Nende nelja parameetri alusel tuntakse signaalide nelja **impulssmodulatsiooni** liiki: 1) amplituud-impulssmodulatsiooni (*AIM*), 2) laius-impulssmodulatsiooni (*LIM*), 3) sagedus-impulssmodulatsiooni (*SIM*) ja 4) faasi-impulssmodulatsioon (*FIM*), mille olemusest annab ülevaate joonis 1. Märkigem, et nende terminite asemel võib kasutada ka pulsiamplituudi-, pulsilaiuse-, pulsisageduse- ja pulsifaasimodulatsiooni mõisteid.



Joonis 1.1. Impulssmodulatsiooni liigid

Kvantimisperiood ehk diskreetimisintervall τ valitakse sõltuvalt kvanditava signaali sageduslikest omadustest (spektrist) nii, et kvantimisega ei läheks kaduma signaaliga edastatav info. Selleks peab kvantimisperiood τ olema vähemalt kaks korda lühem kui signaali spektri suurima sagedusega harmoonilise komponendi periood

$$t = \frac{1}{2 f_{\max}} \qquad \text{või} \qquad f_i = \frac{1}{t_i} = 2 f_{\max}, \tag{1.1}$$

kus $f_{\max}$ on signaali spektri harmooniliste komponentide suurim sagedus.

Juhul kui impulsi parameetrid ei ole määratletud, sisaldab üks impulss ühe biti informatsiooni, s. t impulsi olemasolu võib lugeda signaaliks 1 ning selle puudumise signaaliks 0. Kahe impulsiga opereerides saab edastada $2^2 = 4$ bitti infot jne. Mitmebitiseid impulsisignaale saab kodeerida kahendarvude koodiga ning neid nimetatakse **arvsignaalideks**. Digitaaltehnikas kasutatakse kõige enam 8-, 10-, 12- või 16-bitiseid arvsignaale, mille infosisaldus on $2^8 = 256$, $2^{10}=1024$, $2^{12} = 4096$ ja $2^{16} = 65536$ bitti.

1.1.2. Kodeerimine, dekodeerimine ja koodide liigid

Kodeerimine on informatsiooni esitusvormi muutmine sellekohase reeglistiku alusel. Numbritest koostatud koode nimetatakse **arvkoodideks**. Arvsignaale moodustatakse kodeerimisega. Eri arvusüsteemidele vastavad erinevad koodid. Arvusüsteemidest tuntakse kõige enam kümnendsüsteemi. Vähem on kasutusel nn rooma numbrite süsteem. Arvutustehnikas rakendatakse peamiselt kahendsüsteemi, kuid ka kaheksand- ja kuueteist-kümnendsüsteemi. Kõiki arvusüsteeme võib jaotada positsioonilisteks süsteemideks ning mittepositsioonilisteks süsteemideks. Viimaste hulka kuulub näiteks rooma numbrite süsteem. **Positsiooniliseks süsteemiks** nimetatakse arvusüsteemi, kus ühel ja samal arvul on erinev väärtus sõltuvalt asukohast arvujadas. Neid süsteeme iseloomustab arvude esitamise selgus ning aritmeetiliste operatsioonide lihtsus. Positsiooniliste süsteemide hulka kuuluvad nii kümnend-, kahend-, kaheksand- kui ka kuueteistkümnendsüsteem (tabel 1). Positsioonilist süsteemi kirjeldatakse üldjuhul valemiga

$$X = a_n \cdot s^n + a_{n-1} \cdot s^{n-1} + \mathrm{L} + a_1 \cdot s + a_0 \cdot s^0 + a_{-1} \cdot s^{-1} + a_{-2} \cdot s^{-2} + \mathrm{L}\,, \tag{1.2}$$

kus teguriteks a_i võivad olla suvalised süsteemis kasutatavad arvud.

Ka positsioonilisi süsteeme on erinevaid. Üldjuhul võib arvu igale numbrile anda suvalise kaalu, mida nimetatakse arvu **kohakaaluks**. Arvu mõistmiseks tuleb ette anda **koodi võti**, mis näitab arvu kõigi kohtade kaalu. Kümnendkoodi korral on kohakaaludeks arvud 10^n, kus *n* on koha järjenumber. 10 on koodi **põhiarv**, kuna selle järgi moodustatakse kõik kohakaalud. Kohti loendatakse alates paremalt, mille kohakaal kümnendkoodis on 10^0.

Kahendkoodi põhiarv on 2, kohakaaludeks aga arvud 2^n. Koodi võtmeks on harilikult 8421, s. o 2^3, 2^2, 2^1, 2^0.

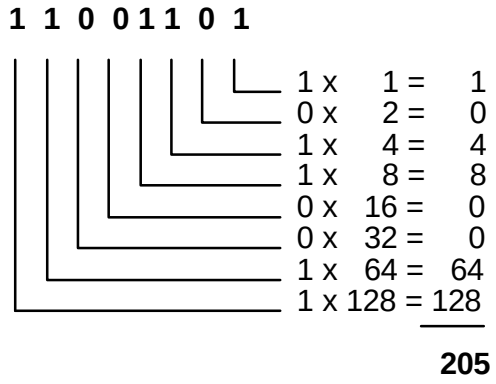
Kahendarvudel on järgmised omadused:

- kasutatakse kahte sümbolit 0 ja 1;
- põhiarvuks on 2;
- kohakaaludeks on arvud 2^n (1, 2, 4, 8, 16, 32, 64 jne), kus *n* on arvu kohanumber.

Kahendarvu väärtuse leidmiseks tuleb selle kohti tähistavad arvud korrutada kohakaaludega ning seejärel liita. Nii leitakse, et kahendarvule 11001101 vastab kümnendsüsteemis väärtus 205 (joonis 1.2). Nagu näha, on kahendarvul palju enam kohti kui vastaval kümnendarvul. Suurimale kaheksakohalisele kahendarvule vastab kolmekohaline kümnendarv 255 ning suurimale kuueteistkümnekohalisele kahendarvule viiekohaline arv 65535. Sellest tingituna on inimesel kahendarvudega opereerida tülikas.

Sümbolid: 0, 1
Põhiarv: 2
Kohakaalud: 2^n, kus *n* on kohanumber

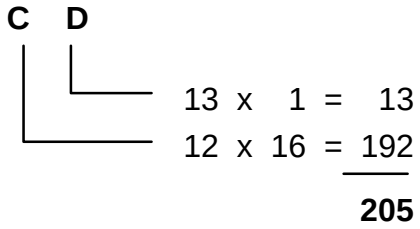
Näide:



Joonis 1.2. Kahendarvu väärtuse leidmine

Sümbolid: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
Põhiarv: 16
Kohakaalud: 16^n, kus *n* on kohanumber

Näide:



Joonis 1.3. Kuueteistkümnendarvu väärtuse leidmine

Kuueteistkümnendarvu esitamiseks kasutatakse kümnendarvu sümboleid 0...9 ning ladina kirja suurtähti *A* = 10, *B* = 11, *C* = 12, *D* = 13, *E* = 14, *F* = 15, mida antud juhul tõlgendatakse kui numbreid (joonis 1.3). Kui võrrelda kahendarvu 1100 1101 vastava kuueteistkümnendarvuga *CD*, siis on näha, et üks kuueteistkümnendarvu koht vastab neljakohalisele kahendarvule. See tähendab, et neljakohalise kahendarvu saab esitada vaid ühe kuueteistkümnendarvu sümboliga 0...*F* ehk 0...15. Kuueteistkümnendarvude väärtus leitakse samuti kui kümnend- ja kahendarvude puhul kohaväärtuste liitmisega.

Levinumatest arvkoodidest ja arvusüsteemidest annab ülevaate tabel 1.1. Tehnikas kasutatakse laialt **kahend-kümnendkoodi**, mille mõistmiseks on vaja teada, et arvu kodeerimisel on kasutatud korraga kahend- ja kümnendkoodi positsioonilisi süsteeme, s. t

mitmekohaline arv on kodeeritud kümnendkoodis, kuid iga selle number esitatakse kahendkoodis. Kahend-kümnendkoodide korral rakendatakse mitmesuguse võtmega koode, lisaks tavalisele 8421 koodile näiteks ka 4221 või 2421 võtmega koode.

Kahend-kümnendkoodiga 8421+3 saab lihtsustada kahend-kümnendarvude aritmeetika-tehteid. Koodi võti 8421+3 tähendab, et arvujada iga kümnendnumbri asemel kodeeritakse kahendkoodis 3 võrra suurem kümnendarv, näiteks arvu 5 asemel arv 8 või arvu 8 asemel 11. Saadakse neljakohalise kahendkoodi ülekanne järgmisele kohale vastavalt kümnendarvule (kümnendarvu ülekanne tekib arvu 9 järel, neljakohalise kahendarvu ülekanne aga $1111_2 = 15_{10}$ järel). Tsüklilistest ehk peegeldunud koodidest on levinud nn **Gray kood**, mida rakendatakse positsioonjuhtimisega ajamite asendiandurites. Koodi eeliseks võrreldes teiste kahendkoodidega on see, et anduri lähimatele naaberasenditele (naaberarvudele) vastavad koodid erinevad teineteisest minimaalselt, s. t ainult ühe järgu võrra. Seega muutub kood anduri modulatsiooniketta liikumisel sujuvalt ning asendisignaali töötlevate loogikaskeemide lülituste arv on minimaalne, mis suurendab seadmete töökindlust. **Ühikkoodi** kasutakse juhul, kui mingit füüsikalist suurust iseloomustab impulsside arv, näiteks kui asendit määratakse impulssanduri impulsside loendamisega.

Tähtsad on **vigu avastavad** ning **vigu korrigeerivad koodid**, mida nimetatakse **veatõrjekoodideks**. Seoses informatsioonitehnika arenguga on oodata nende koodide üha laiemat rakendust. Veatõrjekoodide abil on võimalik vähendada seadmete keerukusest ning keskkonnahäiretest põhjustatud vigu ning järelikult suurendada seadmete töökindlust. Vigu avastavate ja korregeerivate koodide puhul on lubatud ehk õigete koodide arv märksa väiksem võimalike koodide arvust. Seega on osa koode keelatud ehk vigased. Lubatud ja keelatud koodide suhet iseloomustab koodi samm *d*. Tavalise kahendsüsteemi korral $d = 1$, s. t kõik koodid on lubatud. Kui $d = 2$, siis on lubatud ja keelatud koodide arv võrdne ning niisugune kood võimaldab avastada ühe vea. Kui avastatakse keelatud kood teatatakse sellest juhtseadme või süsteemi operaatorile, kes teeb vastavad järeldused. Koodi sammu $d = 3$ korral saab avastada korraga kahte viga. Kui eeldada, et korraga esineb vaid üks viga, siis saab niisuguse koodiga ka vigu korrigeerida, s. t keelatud koodid jagunevad omakorda rühmadesse, mille koode korrigeeritakse kas üheks või teiseks lubatud koodiks. Koodi sammu suurenemisel avarduvad ka vigade avastamise ning korrigeerimise võimalused. Kui koodis on informatsiooni rohkem kui tema eristamiseks minimaalselt vaja, on tegu **liiaskoodiga**. Seega sobivad liiaskoodid veatõrjekoodideks. Tuntumad veatõrjekoodid on *Hammingi* ja *Reedi-Mulleri* koodid.

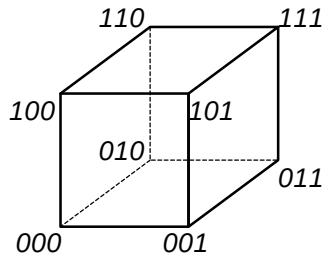
Tabel 1.1

Arvkoodid ja arvusüsteemid

Nr	Koodi nimi	Sümbolid a_i	Koodi valem või näide
1	Kahendkood	0, 1	$L\, a_3 \cdot 2^3 + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0$

2	Kaheksand-kood	0, 1, 2, 3, 4, 5, 6, 7	L $a_3 \cdot 8^3 + a_2 \cdot 8^2 + a_1 \cdot 8^1 + a_0 \cdot 8^0$
3	Kümnendkood	0, 1, 2, 3, ... , 9	L $a_3 \cdot 10^3 + a_2 \cdot 10^2 + a_1 \cdot 10^1 + a_0 \cdot 10^0$
4	Kuueteistkümnendkood	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F	L $a_3 \cdot 16^3 + a_2 \cdot 16^2 + a_1 \cdot 16^1 + a_0 \cdot 16^0$

Tabeli 1.1 järg

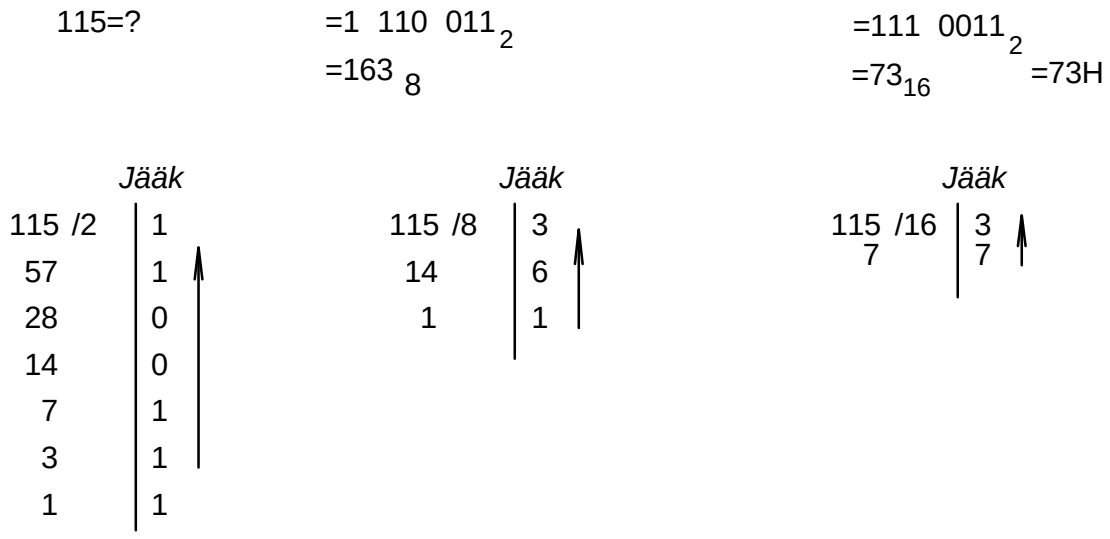
5	Kahend-kümnendkood 8421	0, 1	Näide: 852 = 1000 0101 0010
6	Kahend-kümnendkood 4221	0, 1	Näide: 852 = 1110 0111 0010
7	Kahend-kümnendkood 4421	0, 1	Näide: 852 = 1100 0101 0010
8	Kahend-kümnendkood 2421	0, 1	Näide: 852 = 1110 0101 0010
9	Kahend-kümnendkood 8421+3	0, 1	Näide: 852 = 1011 1000 0101
10	Tsükliline ehk peegeldunud kümnendkood	0, 1, 2, 3, ... , 9	00, 01, 02, 03, 04, 05, 06, 07, 08, 09, / 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, / 20, 21, 22, 23, ...
11	Tsükliline ehk peegeldunud kahendkood; Gray kood	0, 1	0 - 0000 6 - 0101 11 - 1110 1 - 0001 7 - 0100 12 - 1010 2 - 0011 8 - 1100 13 - 1011 3 - 0010 9 - 1101 14 - 1001 4 - 0110 10 - 1111 15 - 1000 5 - 0111 16 - 0000
12	Ühikkood	1	1, 11, 111, 1111, 11111
13	Vigu avastav kood *d* = 2	Lubatud Keelatud 000 001 011 010 101 100 110 111	

14	Vigu avastav ja korrigeeriv kood $d = 3$	Lubatud 000 111	Keelatud 001 010 100 110 011 101	Koodid 001, 010 ja 100 parandatakse koodiks 000 Koodid 110, 011 ja 101 parandatakse koodiks 111

1.1.3. Kümnendarvude teisendamine kahend-, kaheksand- ja kuueteistkümnendarvudeks

Kümnendarvu teisendamiseks kahendarvuks kasutatakse joonisel 1.4 näidatud skeemi. Arv *115* jagatakse tulbas arvuga *2* ning eraldatakse jagamise jääk *1*. Jagamise tulemus *57* kirjutatakse esialgse arvu alla. Seejärel korratakse kirjeldatud tegevust seni, kuni jagamise tulemuseks saadakse arv *1*. Jagamise jääkidest moodustub eraldi tulp, mis sisaldab arve 1 ja 0. Lugedes selles tulbas olevaid sümboleid alt üles leitakse lähtearvule 115 vastav kahendarv 1110011.

Nagu joonisel näidatud, saab sama toiminguga leida ka kaheksand- või kuueteistkümnendarve.



Joonis 1.4. Kümnendarvude teisendamine kahend-, kaheksand- ja kuueteistkümnendarvudeks

Märksa lihtsam on kümnendarve teisendada kaheksand- ja kuueteistkümnendarvudeks kahendkoodi abil. Selleks tuleb kümnendarv teisendada esmalt kahendarvuks ning jaotada selle kohad triaadideks või tetraadideks, olenevalt sellest kas soovitakse leida kaheksand- või kuueteistkümnendkoodi. Edasi kodeeritakse kahendarvu iga triaadi eraldi vastava kaheksandarvu või tetraadi kuueteistkümnendarvu sümboliga.

Arvude teisendamist ühest süsteemist teise näitab tabel 1.2, kuhu on koondatud mõned kümnend-, kahend-, kuuetistkümnend- ja kahend-kümnendkoodis ehk *BCD*-koodis arvud.

Tabel 1.2

Arvude teisendamine

Kümnendarv	Kahendarv	Kuueteist-kümnendarv	*BCD*-kood
0	0	0	0000
1	01	1	0001
2	10	2	0010
3	11	3	0011
4	100	4	0100
5	101	5	0101
6	110	6	0110
7	111	7	0111
8	1000	8	1000
9	1001	9	1001
10	1010	A	0001 0000
11	1011	B	0001 0001
12	1100	C	0001 0010
13	1101	D	0001 0011
14	1110	E	0001 0100
15	1111	F	0001 0101
16	1 0000	10	0001 0110
17	1 0001	11	0001 0111
18	1 0010	12	0001 1000
19	1 0011	13	0001 1001
20	1 0100	14	0010 0000
126	111 1110	7E	0001 0010 0110
127	111 1111	7F	0001 0010 0111
128	1000 0000	80	0001 0010 1000
510	1 1111 1110	1FE	0101 0001 0000
511	1 1111 1111	1FF	0101 0001 0001
512	10 0000 0000	200	0101 0001 0010

1.1.4. Informatsiooni hulk ja signaali viga

Arvsignaalis sisalduva informatsiooni hulga ja signaali vea vahel on olemas kindel sõltuvus. Arvsignaali bittide arv n ehk kahendarvu järkude arv määrab signaali diskreetsusastmete (diskreetide) arvu $N = 2^n$. Signaali kodeerimisveaks loetakse maksimaalselt ühe diskreedi väärtus. Seega on 10-bitise kahendsignaali viga 1/1024 ≅0,1 %.

1.2. Loogikafunktsioonid ja loogikalülitused ning nende esitusviisid

1.2.1. Loogikatehted

Loogikalülituste projekteerimine, talitlus ja selle analüüs põhineb loogikaalgebral (Boole'i algebra). Muutujatel saab siin olla ainult kaks väärtust 0 - väär ja 1 - tõene. Seepärast nimetatakse seda loogikat ka **binaarloogikaks**. Loogilisi muutujaid tähistatakse ladina tähestiku tähtedega.

Sõltumatuid muutujaid (sisendeid) nimetatakse argumentideks, neist sõltuvaid muutujaid aga funktsioonideks. Loogikafunktsiooni kõik argumendid on loogilised muutujad, millel on kaks väärtust 0 ja 1. Kõiki loogikafunktsioone väljendavad kolm põhitehet: loogiline korrutamine, loogiline liitmine ja loogiline eitus.

Loogiline korrutamine (NING). NING-funktsioon on võrdne ühega ainult juhul, kui kõik argumendid on võrdsed ühega. Tehte tähistamiseks kasutatakse nii harilikku korrutusmärki ($\bullet$) kui ka loogilise korrutamise eritähist - katust ($\wedge$). Loogilist korrutamist nimetatakse ka konjunktsiooniks.

Loogiline liitmine (VÕI). VÕI-funktsioon on üks siis, kui kas või üks argumentidest võrdub ühega. VÕI-tehte tähistamiseks kasutatakse kas pluss (+) märki või loogilise liitmise eritähist - V tähe kujulist märki ($\vee$). Loogilist liitmist nimetatakse ka disjunktsiooniks.

Loogiline eitus (EI). EI-funktsioonil on argumendi vastandväärtus. Kui argument on 1, siis funktsioon võrdub 0 ning vastupidi. EI-tehet tähistatakse kriipsuga sümboli peal, näiteks argumendi x eitus on $\overline{x}$. Loogilist eitust nimetatakse ka inversiooniks.

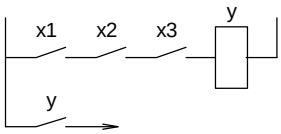
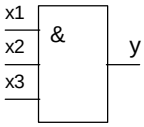
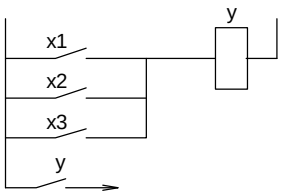
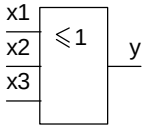
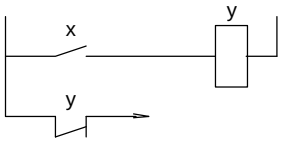
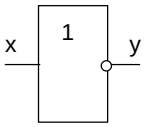
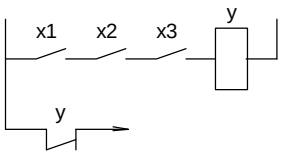
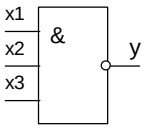
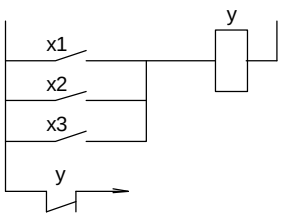
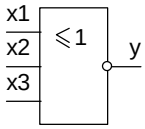
Loetletud kolm loogikatehet moodustavad **loogiliselt täieliku süsteemi**, mida rakendades saab realiseerida mis tahes loogikafunktsiooni. Kõiki kolme loogika põhifunktsiooni on loogikaalgbra reeglite alusel võimalik realiseerida ainult üht tüüpi loogikaelementide kas NING-EI või VÕI-EI abil. Järelikult võib NING-EI- ja VÕI-EI-elemente ning tehteid nendega nimetada **universaalseteks loogikaelementideks ja -teheteks**.

Lisaks põhifunktsioonidele leiavad kasutamist mitmed loogika tüüpfunktsioonid, nagu alternatiiv, ekvivalentsus, implikatsioon jt. Niisuguste funktsioonide ja elementide olemasolu lihtsustab loogikalülituste sünteesi. Loogikafunktsioonidest ja loogika tüüpelementidest annab ülevaate tabel 1.3.

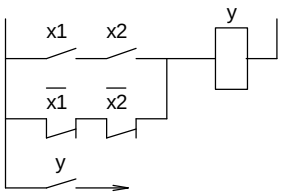
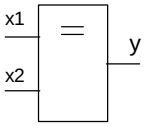
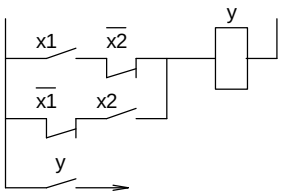
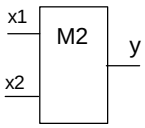
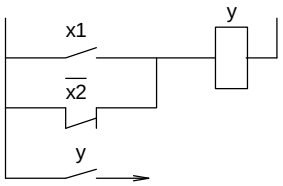
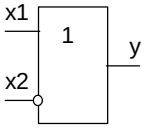
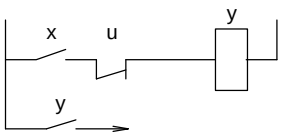
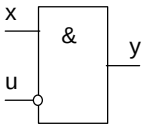
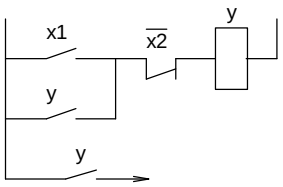
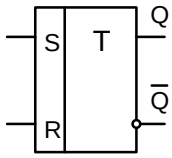
Keerukamaid loogikalülitusi, mis koosnevad paljudest loogikaelementidest ning on ette nähtud kindlate funktsioonide täitmiseks, nimetatakse **funktsionaalseteks loogikalülitusteks**, mille hulka võib lugeda ka tabelis 1.3 toodud mäluelemendi.

Tabel 1.3

Loogikafunktsioonid ja -elemendid

Nr	Loogika-funktsiooni nimetus	Loogika-funktsiooni selgitus	Loogikafunktsiooni kontaktaseskeem	Loogikafunktsiooni matemaatiline esitus	Loogika-elemendi tähis
1	2	3	4	5	6
1.	NING	Lüli väljundis on signaal 1, kui signaal 1 on tema 1. ning 2. ning 3. ning jne sisendis		$y = x1 \cdot x2 \cdot x3$ $y = x1 \wedge x2 \wedge x3$	
2.	VÕI	Lüli väljundis on signaal 1, kui signaal 1 on tema 1. või 2. või 3. või jne sisendis		$y = x1 + x2 + x3$ $y = x1 \vee x2 \vee x3$	
3.	EI	Lüli väljundis on signaal 1, kui tema sisendis on signaal 0 ja vastupidi		$y = \overline{x}$	
4.	NING-EI	Väljundis on signaal 0, kui kõigis sisendites on signaal 1		$\overline{y} = x1 \cdot x2 \cdot x3$ $\overline{y} = x1 \wedge x2 \wedge x3$ $y = \overline{x1 \cdot x2 \cdot x3}$	
5.	VÕI-EI	Väljundis on signaal 0, kui kas või ühes sisendis on signaal 1		$\overline{y} = x1 + x2 + x3$ $\overline{y} = x1 \vee x2 \vee x3$ $y = \overline{x1 + x2 + x3}$	

Tabeli 1.3 järg

1	2	3	4	5	6
6.	Ekvi-valent-sus	Väljundis on signaal 1 ainult siis, kui sisenditel on ühesugused väärtused		$y = x1 \cdot x2 + \overline{x1} \cdot \overline{x2}$	
7.	Välistav VÕI	Väljundis on signaal 1 ainult siis, kui sisendite olek on erinev		$y = \overline{x1} \cdot x2 + x1 \cdot \overline{x2}$ $y = x1 \oplus x2$	
8.	Impli-katsioon	Väljundis on signaal 0 ainult siis, kui $x2 = 1$ ja $x1 = 0$. Loogiline järeldus		$y = x1 + \overline{x2}$	
9.	Keeld	Väljund võrdub sisendiga x, kui signaal u on 0. Sinaali $u = 1$ korral on väljundis signaal 0 sõltumata sisendist x		$y = x \cdot \overline{u}$	
10.	Mälu	Sisendiga $x1$ lülitatakse väljund olekusse 1 ning see olek säilib, kuni seda ei muudeta sisendiga $x2$		$y = (x1 + y') \cdot x2$	

1.2.2. Loogikaseadused

Loogikaseadusteks nimetatakse tavaliselt binaarloogika algebra ehk Boole' i algebra seadusi. Algebraks nimetatakse üldjuhul elementide hulka, millega tehakse tehteid, kusjuures nende tehete aluseks on kindlad reeglid ehk aksioomid. Aksioomid määravad ära algebra põhitehete omadused ja seosed. Kuna nüüdismatemaatikas on palju algebra liike (universaalalgebra, hulgaalgebra, loogikaalgebra), siis kehtivad neis ka erinevad tehted ja aksioomid. Boole'i algebra elementideks on binaarloogika signaalid (argumendid) väärtustega 0 ja 1. Nende signaalidega saab sooritada kõiki loogikatehteid ning moodustada suvalisi loogikafunktsioone. Loogikatehete kohta kehtivad järgmised **binaarloogika aksioomid**:

1. Argumentide järjekorda võib tehtes muuta
   $a + b = b + a.$ (1.3)

2. Sulgusid võib avada ehk funktsiooni võib teisendada loogiliste osakorrutiste summaks
   $a(b + c) = ab + ac.$ (1.4)

3. Funktsiooni võib teisendada loogiliste osasummade korrutiseks
   $a + bc = (a + b)(a + c).$ (1.5)

4. Argumendi ja tema eituse loogiline korrutis võrdub nulliga ega muuda loogilise summa väärtust
   $a + a \cdot \bar{a} = a.$ (1.6)

5. Suvalise argumendi ja tema eituse loogiline summa võrdub alati ühega
   $a + \bar{a} = b + \bar{b} = 1.$ (1.7)

6. Suvalise argumendi ja tema eituse loogiline korrutis võrdub alati nulliga
   $a \cdot \bar{a} = b \cdot \bar{b} = 0.$ (1.8)

Loogikatehete ja aksioomide põhjal leitakse kahendarvude kohta kehtivad järgmised reeglid:

$$\bar{0} = 1;\ \bar{1} = 0;$$

$$0 \cdot 0 = 0; \quad 0 \cdot 1 = 0; \quad 1 \cdot 0 = 0; \quad 1 \cdot 1 = 1; \tag{1.9}$$

$$0+0=0;\quad 0+1=1;\quad 1+0=1;\quad 1+1=1.$$

Aksioomide põhjal tuletatakse peamised loogikaseadused:

1. **Domineerimisseadus I**. Suvalise muutujate hulga konjunktsioon on null (tühihulk), kui kas või ainult üks muutujatest võrdub nulliga

$$0 \cdot a \cdot b \cdot c \,\mathrm{L} = 0. \qquad (1.10)$$

2. **Domineerimisseadus II**. Suvalise muutujate hulga disjunktsioon on üks (universaalhulk), kui kas või ainult üks muutujatest võrdub ühega

$$1 + a + b + c + \mathrm{L} = 1. \qquad (1.11)$$

3. **Idempotentsus- ehk samaväärsusseadus** (kehtib ka kolme ja enama muutuja kohta). Argumendi loogiline korrutamine või liitmine iseendaga ei muuda tulemi väärtust

$$a \cdot a = a; \qquad a + a = a. \qquad (1.12)$$

4. **Eituse eitamise seadus**. Argumendi väärtus tema kahekordsel eitamisel ei muutu

$$\bar{\bar{a}} = a. \qquad (1.13)$$

5. **Komplementaarsus- ehk täiendiseadus**. Argumendi ja tema eituse ehk täiendi loogiline korrutis on null, loogiline summa üks

$$a \cdot \bar{a} = 0; \qquad a + \bar{a} = 1. \qquad (1.14)$$

6. **Kommutatiivsusseadus**. Argumentide järjekorda loogikatehetes võib muuta

$$a \cdot b = b \cdot a; \qquad a + b = b + a. \qquad (1.15)$$

7. **Assotsiatiivsusseadus**. Mitme argumendi loogilist korrutamist ja loogilist liitmist võib sooritada suvalises järjekorras või samaaegselt

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c = a \cdot b \cdot c;$$
$$a + (b + c) = (a + b) + c = a + b + c. \qquad (1.16)$$

8. **Distributiivsusseadus** (sulgude avamise seadus). Argumentide loogilist summat võib loogiliselt korrutada argumendiga *a* või korrutada esmalt kõiki argumente *a*-ga ning seejärel need korrutised loogiliselt liita. Argumentide loogilisele korrutisele võib liita argumendi *a* või esmalt liita loogiliselt kõikidele argumentidele *a* ning seejärel need summad loogiliselt korrutada. Kui esimene teisendus vastab sulgude avamisele
arvude algebras, siis teine on rakendatav üksnes loogikaalgebras

$$a \cdot (b + c) = a \cdot b + a \cdot c;$$

$$a + b \cdot c = (a + b) \cdot (a + c). \tag{1.17}$$

9. **Absorbtsiooni- ehk neelduvusseadused**. Kui kahe argumendi loogilist summat, kus üheks argumendiks on *a*, korrutada sama argumendiga *a*, siis teine argument neeldub ning tulemiks on samuti *a*. Sama kehtib ka siis, kui korrutatavaid summasid on rohkem ning kui kõigis neis sisaldub ühe argumendina *a*. Seadus on rakendatav nii summade korrutiste kui ka korrutiste summade kohta. Kui osasummas või osakorrutises sisaldub argumendi *a* eitus (inversioon), on tulemiks *a* ja teise argumendi korrutis *ab* või summa *a*+*b*

$$a \cdot (a + b) = a;$$
$$a \cdot (a + b) \cdot (a + c) \,\mathrm{L}\, (a + w) = a;$$

$$a + a \cdot b = a;$$
$$a + a \cdot b + a \cdot c + \mathrm{L} + a \cdot w = a; \qquad (1.18)$$

$$a \cdot (\bar{a} + b) = a \cdot b;$$
$$a + \bar{a} \cdot b = a + b.$$

10. **Kleepimisseadus**. Kui üks loogiline korrutis sisaldab argumenti *b* ja teine selle eitust, siis nende korrutiste loogilisel summeerimisel argument koondub. Kui üks loogiline summa sisaldab argumenti *b* ja teine selle eitust, siis nende summade loogilisel korrutamisel argument koondub

$$a \cdot b + a \cdot \bar{b} = a;$$
$$(a + b) \cdot (a + \bar{b}) = a; \qquad (1.19)$$

Üldised kleepimisseadused:

$$a \cdot b + \bar{a} \cdot c + b \cdot c = a \cdot b + \bar{a} \cdot c;$$
$$(a + b) \cdot (\bar{a} + c) \cdot (b + c) = (a + b) \cdot (\bar{a} + c); \qquad (1.20)$$
$$(a + b) \cdot (\bar{a} + c) = a \cdot c + \bar{a} \cdot b.$$

11. **De Morgani seadused**. Argumentide loogilise korrutise eitus võrdub nende argumentide eituste loogilise summaga. Argumentide loogilise summa eitus võrdub nende argumentide eituste loogilise korrutisega. De Morgani seadusi rakendades saab asendada loogilise liitmistehte loogilise korrutamisega ning vastupidi loogilise korrutamise tehte loogilise liitmisega

$$\overline{a \cdot b} = \bar{a} + \bar{b};$$
$$\overline{a + b} = \bar{a} \cdot \bar{b};$$
$$\overline{a \cdot b \cdot c \,\mathrm{L}\, w} = \bar{a} + \bar{b} + \bar{c} + \mathrm{L} + \bar{w}; \qquad (1.21)$$
$$\overline{a + b + c + \mathrm{L} + w} = \bar{a} \cdot \bar{b} \cdot \bar{c} \,\mathrm{L}\, \bar{w}.$$

Üldistatud De Morgani ehk Shannoni seadus

$$\overline{f(a, b, c, \mathrm{L}\, w, \cdot, +)} = f(\bar{a}, \bar{b}, \bar{c}, \mathrm{L}\,, \bar{w}, +, \cdot). \qquad (1.22)$$

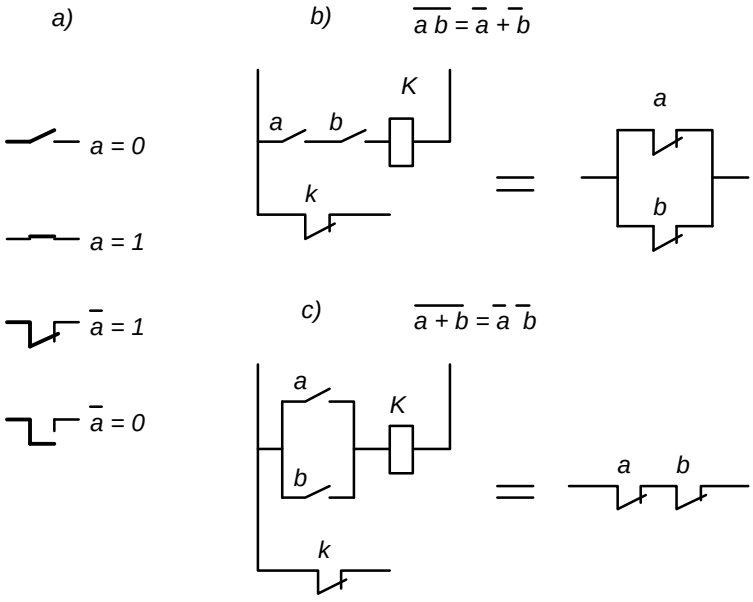
Loogikaseadusi saab tõestada loogika tõeväärtustabelitega või relee-kontaktskeemide abil. De Morgani seaduste tõestus loogika tõeväärtustabelite abil on toodud tabelis 1.4.

Tabel 1.4

a	b	ab	$\overline{ab}$	$\bar{a}$	$\bar{b}$	$\bar{a}+\bar{b}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

a	b	$a+b$	$\overline{a+b}$	$\bar{a}$	$\bar{b}$	$\bar{a}\bar{b}$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

Loogikaseaduste tõestamisel kontaktskeemide abil joonistatakse võrrandite vasak- ja parempoolsete avaldiste alusel välja kontaktskeemid ning võrreldakse neid omavahel. Avatud kontakt vastab signaalile 0 ning suletud kontakt signaalile 1. Kontaktide jadaühendus vastab loogilisele konjunktsioonile (loogiline NING-funktsioon), kontaktide rööpühendus aga disjunktsioonile (loogiline VÕI-funktsioon). Avanevale kontaktile vastab inversioon ehk loogiline EI-funktsioon. Näiteks saab De Morgani seaduste tõestuse esitada kontaktskeemide abil järgmiselt:



Joonis 1.5. De Morgani seaduse tõestus kontaktskeemide abil

Boole'i ehk loogikafunktsioonide teisendamiseks eraldatakse nende hulgast nn elementaarfunktsioonid. Nendeks on esiteks kõik mõeldavad kahe muutuja funktsioonid, sealhulgas eespool vaadeldud inversioon, disjunktsioon ja konjunktsioon; kahe muutuja funktsioone on kokku 16 (tabel 1.5). Teiseks kuuluvad elementaarfunktsioonide hulka kõik rohkem kui kahe argumendiga funktsioonid, milles argumendid on omavahel seotud kas ainult disjunktsiooni- või ainult konjunktsioonitehtega.

Tabel 1.5

a	b	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
		"0"	&	$\overline{\rightarrow}_{ab}$	a	$\overline{\rightarrow}_{ba}$	b	$\oplus$	$\vee$	$\overline{\vee}$	~	$\overline{b}$	$\rightarrow_{ba}$	$\overline{a}$	$\rightarrow_{ab}$	$\overline{\&}$	"1"
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Boole'i funktsiooni standardesituseks on tema normaalkuju. Loogikafunktsiooni normaalkuju koosneb **elementaarkonjunktsioonidest** (konjunktsioonitehte abil seotud otsestest või inverteeritud muutujatest, kus iga muutuja esineb vaid üks kord). Kui loogikafunktsioon on esitatud elementaarkonjunktsioonide disjunktsioonina, nimetatakse esitusviisi funktsiooni **disjunktiivseks normaalkujuks (DNK)**. Vähem kasutatakse loogikafunktsiooni **konjunktiivset normaalkuju (KNK)**, mil funktsioon esitatakse elementaardisjunktsioonide konjunktsioonina. Kui funktsiooni disjunktiivse normaalkuju iga elementaarkonjunktsioon sisaldab kõiki muutujaid, nimetatakse funktsiooni esitusviisi tema **täielikuks disjunktiivseks normaalkujuks (TDNK**). Täielikku disjunktiivset normaalkuju on hõlpus leida loogikafunktsiooni oleku- ehk tõeväärtustabelist.

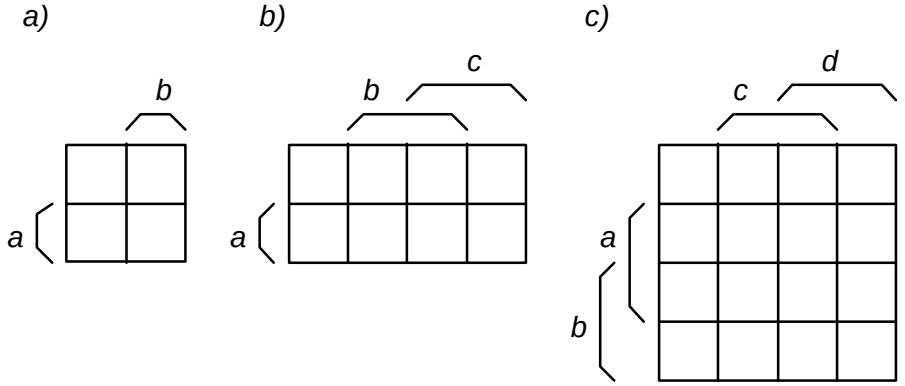
1.2.3. Loogikalülituste süntees ja minimeerimine

Loogikalülituste konstrueerimisel on oluline lülitust võimalikult lihtsustada, mis vähendab lülituse hinda ja koostamise töömahtu. Seepärast tuleb juba loogikalülituste sünteesil funktsioone kindlate kriteeriumide järgi minimeerida. Kõige enam on läbi töötatud loogikafunktsioonide täielike disjunktiivsete normaalkujude minimeerimismeetodid. Tavaliselt on eesmärgiks leida minimaalse pikkusega loogikafunktsiooni algebraline avaldis, milles on minimaalne arv sisendmuutujate tähiseid, näiteks **minimaalne disjunktiivne normaalkuju** ehk **MDNK**. Loogikafunktsioonide minimeerimiseks kasutatakse 1) vahetut

lihtsustamist, 2) lihtsustamist Karnaugh kaardi abil, 3) Quine - Mc Cluskey meetodit, 4) Blake' i meetodit jms.

Karnaugh kaart on loogikafunktsiooni tõeväärtustabeli ehk olekutabeli erikuju, mida kasutatakse funktsiooni minimeerimiseks. Karnaugh kaart on ruudu- või ristkülikukujuline lahterdatud tabel. Lahtrite arv sõltub funktsiooni sisendmuutujate (argumentide) arvust *n* ning vastab muutujate kombinatsioonide arvule 2^n. Kahe, kolme ja nelja muutuja funktsiooni Karnaugh kaardid on joonisel 1.6. Muutujad ja funktsiooni väärtsused paigutatakse tabelisse nii, et võimalik oleks esitada kõiki muutujate kombinatsioone. See eeldab muutujate erilist paigutust, nagu on näidatud joonisel. Klambriga hõivatud alas on muutujal otsene, väljaspool klambrit aga inverteeritud väärtus. Karnaugh kaarti saab koostada loogikafunktsiooni tõeväärtustabeli või algebralise võrrandi järgi.

Karnaugh kaardi iseloomulikuks omaduseks on, et funktsiooni väärtused erinevad kõrvuti asuvates lahtrites vaid ühe muutuja poolest, s. t naaberlahtrisse minekul muudab (inverteerib) oma olekut vaid üks sisendmuutuja. Seejuures loetakse naabriteks ka kaardi äärmised vasakpoolsed ja äärmised parempoolsed ning ülemised ja alumised lahtrid. Naaberlahtreid, mis erinevad vaid ühe muutuja poolest, kasutatakse loogikafunktsiooni minimeerimiseks.



Joonis 1.6. Karnaugh kaardid kahe (*a*), kolme (*b*) ja nelja muutuja (*c*) loogikafunktsiooni jaoks

On antud loogikafunktsiooni *z = f(a, b, c)* täielik disjunktiivne normaalkuju

$$z = \bar{a}\bar{b}\bar{c} + \bar{a}bc + a\bar{b}c + ab\bar{c} + abc. \qquad (1.23)$$

Avaldist saab lihtsustada, kui tuua muutujad sulgude ette, kuid see ei kindlusta soodsaima lahenduse saamist. Loogikalülituse minimeerimiseks on otstarbekas kasutada Karnaugh kaarti (joonis 1.7), millele vastab **olekutabel** 1.6, kus on toodud kõigile võimalikele sisendsignaalide kombinatsioonidele vastavad väljundsignaali(de) väärtused. Karnaugh

kaardil moodustatakse ühtedega täidetud ruutudest ristkülikukujulised lahtrid suurusega 1, 2, 4, 8, ... ruutu, taotledes et ruudud oleksid nii suured kui võimalik. Kontuurid võivad üksteisega ka kattuda.

Seejärel kirjutatakse loogikafunktsiooni avaldis disjunktiivsel normaalkujul, milles igale kontuurile vastab elementaarkonjunktsioon muutujatest, mis terve kontuuri jaoks on kas inverteerimata või inverteeritud.

Tabel 1.6

Funktsiooni 1.23 olekutabel

a	b	c	z
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

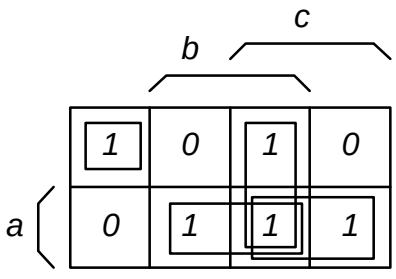
Vaadeldava kaardi tarvis saab kirjutada loogikafunktsiooni järgmisel kujul:

$$z = ab + ac + bc + \bar{a}\bar{b}\,\bar{c}. \qquad (1.24)$$

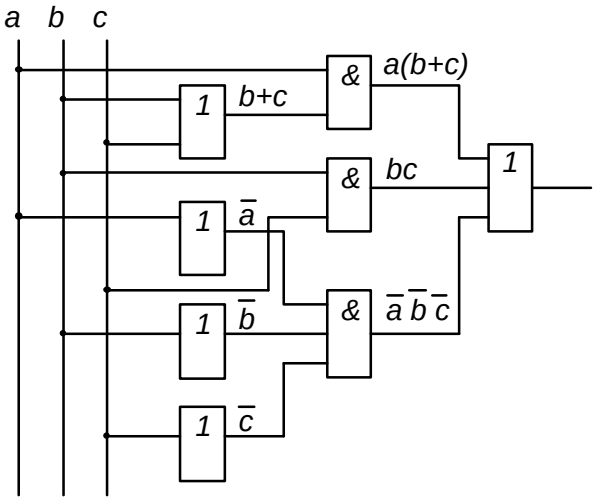
Kui tuua muutujad sulgude ette, saab avaldise tähtede arvu veelgi vähendada

$$z = a(b + c) + bc + \bar{a}\bar{b}\,\bar{c}. \qquad (1.25)$$

Sellele avaldisele vastab loogikalülitus joonisel 1.8.



Joonis 1.7. Funktsiooni Karnaugh kaart



Joonis 1.8. Funktsioonile vastav loogikalülitus

Quine - Mc Cluskey meetodi kohaselt määratakse kõigepealt esmased ehk lihtimplikandid ning seejärel eraldatakse neist olulised lihtimplikandid. Minimeerimise tulemusena saadakse funktsiooni minimaalne disjunktiivne või konjunktiivne normaalkuju.

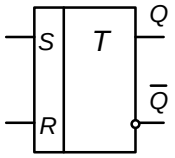
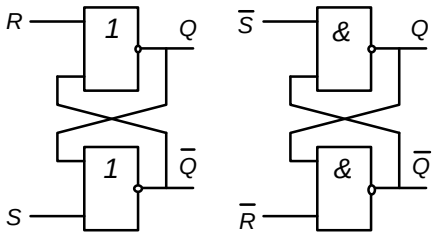
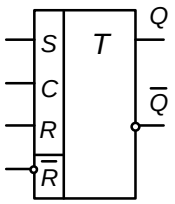
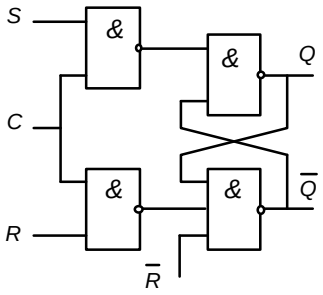
1.3. Funktsionaalsed loogikalülitused

1.3.1. Trigerid

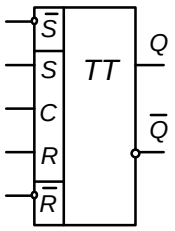
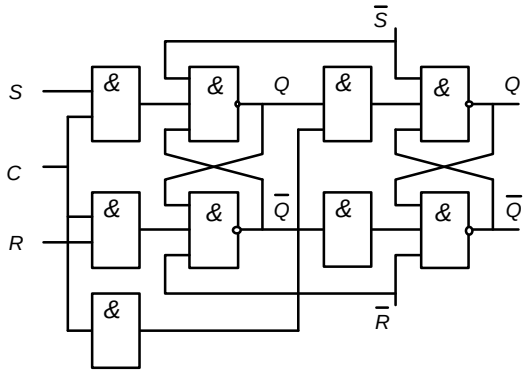
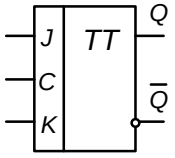
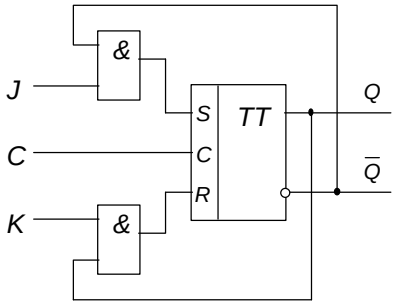
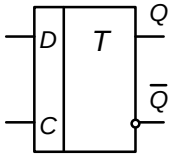
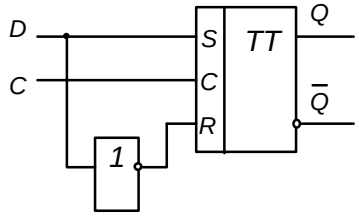
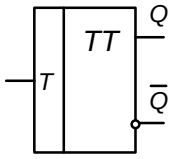
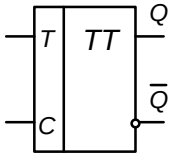
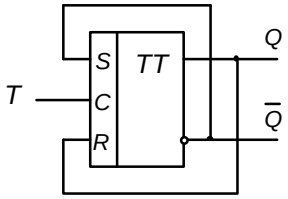
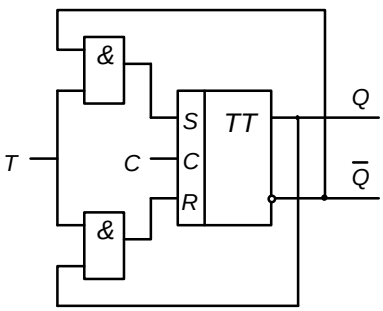
Triger (*flip-flop*) on kahe stabiilse olekuga loogikalülitus. Ühte olekutest tähistatakse numbriga 1, teist numbriga 0. Trigeri olek vastab tema väljundsignaalile. Sõltuvalt sisendsignaalist säilitab triger endise oleku või muudab seda hüppeliselt. Loogikalülituste koostamise lihtsustamiseks on trigeril tavaliselt kaks väljundit: otsene, mida tähistatakse tähega Q, ja inversne $\overline{Q}$. Tööpõhimõtte järgi jaotatakse trigereid seadesisenditega ehk *RS*-trigeriteks, loendussisenditega ehk *T*-trigeriteks, andmesisenditega ehk *D*-trigeriteks ning universaalsisenditega ehk *JK*-trigeriteks. Kui trigeri oleku muutmine toimub kas või ühe sisendi kaudu täiendava sünkroniseerimissignaali abil, nimetatakse trigerit sünkroonseks vastupidisel juhul aga asünkroonseks. Sõltuvalt tööpõhimõttest ning ehitusest liigitatakse trigereid ühe- või kahetaktilisteks ning tähistatakse vastavalt *T* või *TT*-ga. Trigerite põhilised skeemid ning tähised on tabelis 1.7

Tabel 1.7

Trigerite skeemid ja tähised

Tüüp	Olekutabel	Tingmärk	Skeem
Asünk- rooonne *RS*-triger	S R Q_t Q_{t+1} 0 0 1 1 0 0 0 0 0 1 1 0 1 0 1 1 1 0 0 1 1 1 1 x 1 1 0 x		
Sünk- roonne *RS*-triger	S R Q_t Q_{t+1} 0 0 1 1 0 0 0 0 0 1 1 0 1 0 1 1 1 0 0 1 1 1 1 x 1 1 0 x		

Tabeli 1.7 järg

Sünkroonne kahetaktiline *RS*-triger	*S* *R* Q_t Q_{t+1} 0 0 1 1 0 0 0 0 0 1 1 0 0 1 0 0 1 0 1 1 1 0 0 1 1 1 1 x 1 1 0 x		
Sünkroonne kahetaktiline *JK*-triger	*J* *K* Q_t Q_{t+1} 0 0 1 1 0 0 0 0 0 1 1 0 0 1 0 0 1 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1		
D-triger	*D* Q_t Q_{t+1} 0 0 0 0 1 0 1 0 1 1 1 1		
T-triger		*Asünkroonne*  *Sünkroonne* 	 

Asünkroonse ühetaktilise *RS*-trigeri saab koostada VÕI-EI- või NING-EI-elementidega, mis ühendatakse nii, et moodustuks positiivne tagasiside. Seepärast on trigeril võimalik vaid kaks stabiilset olekut, kus ühe elemendi väljundis on signaal 1 ja teise elemendi väljundis 0. Trigeri otsene väljund seatakse olekusse 1, kui sisendisse *S* (*set*) antakse signaal 1. Otsene väljund seatakse olekusse 0, kui sisendisse *R* (*reset*) antakse signaal 1. Juhul kui sisendite *S* ja *R* signaalid on 0-d, säilitab triger väljundis oma endise oleku. Kui mõlemasse sisendisse antakse korraga signaal 1, muutuvad nii otsene kui ka inverteeritud väljundsignaal määramatuks, mistõttu niisugune signaalikombinatsioon pole lubatud. *RS*-trigereid nimetatakse ka **seadesisenditega trigeriteks**.

Sünkroonne ühetaktiline *RS*-triger erineb asünkroonsest trigerist selle poolest, et trigeri olek muutub vaid kindlail sünkroimpulssidega määratud ajahetkeil. Lisaks infosisenditele *S* ja *R* on tal veel sünkroniseerimissisend *C* (*clock*). Trigeril võivad olla korraga nii sünkroonsed kui ka mittesünkroonsed sisendid. Sünkroniseeritud infosisend toimib hetkel, mil saabub sünkroniseerimissignaal.

Kahetaktiline *RS*-triger koosneb kahest järjestikku lülitatud ühetaktilisest trigerist. Ühetaktilise trigeri puuduseks on, et ta ei võimalda samaaegselt infot vastu võtta ja edastada, sest tema väljund muutub kohe pärast sisendsignaali saabumist. Järelikult on info sisestamise hetkel väljundi olek ebamäärane, s. t pole teada, kas sealt loetakse trigeri eelmist või järgmist olekut. Probleemi lahendab kahetaktiliste sünkroonsete trigerite kasutamine. Trigeri esimese ja teise astme sünkroniseerimissignaal on pool perioodi nihutatud. Seega kirjutatakse sünkroniseerimissignaali esimese poolperioodi jooksul info sisendist trigeri esimesse astmesse ning samal ajal on väljundist võimalik lugeda trigeri eelmisele taktile vastavat olekut. Teise poolperioodi jooksul viiakse info trigeri esimesest astmest teise, mille järel triger on valmis järgmisteks infovahetusteks. Kahetaktilise trigeri oleku muutumine toimub pärast sünkroniseerimissignaali lõppu, s. t tema tagafrondiga. Kahetaktiliste trigeritega saab koostada suvalisi loogikaskeeme, sealhulgas ühendada trigeri väljund kokku sisendiga. Peale sünkroniseeritud sisendite võivad kahetaktilisel *RS*-trigeril olla ka mittesünkroniseeritud sisendid. Seadesisenditega *RS*-trigerid on aluseks teiste trigerilülituste koostamisel.

Loendussisendiga *T*-trigeril on vaid üks infosisend *T* (*trigger, toggle*), kus iga järgnev sisendimpulss 1 muudab trigeri oleku vastupidiseks. Signaali 0 korral olek ei muutu. *T*-triger realiseerib loogikafunktsiooni

$$Q(t+1) = Q(t)\overline{T}(t) \vee \overline{Q}(t)T(t). \qquad (1.26)$$

See funktsioon vastab loogikafunktsioonile alternatiiv ehk summeerimine mooduli 2 järgi. Asünkroonse *T*-trigeri saab koostada kahetaktilisest *RS*-trigerist, kui rakendada seal täiendavaid tagasisidesid ning kasutada sisendina *T* sünkroniseerimissisendit *C*. Sünkroonse *T*-trigeri saamiseks tuleb *RS*-trigeri sisendisse lülitada loogikaelemendid NING. *T*-trigerite põhiliseks kasutusalaks on loendurid.

Andmesisendiga *D*-triger on samuti nagu *T*-triger ühe infosisendiga. Trigeri väljundsignaal kordab sisendsignaali, kuid see toimub ajaliselt sünkroniseerimisimpulsside perioodi (ühetaktilise trigeri korral poole perioodi) võrra hiljem. Seega võimaldab *D*-triger lühiajaliselt säilitada informatsiooni, mis paljude loogikaseadmete juures on väga oluline.

D-trigeri saab koostada *RS*-trigerist, kui juhtida selle *S*- ja *R*- sisendeid korraga, *S*-sisendit otse ja *R*-sisendit läbi inverteri. Sel juhul töötab triger ainult seaderež iimis, s. t tal puudub hoiderež iim. Ühist sisendit tähistatakse tähega *D* (*data, delay*). *D*-triger töötab vastavalt loogikafunktsioonile

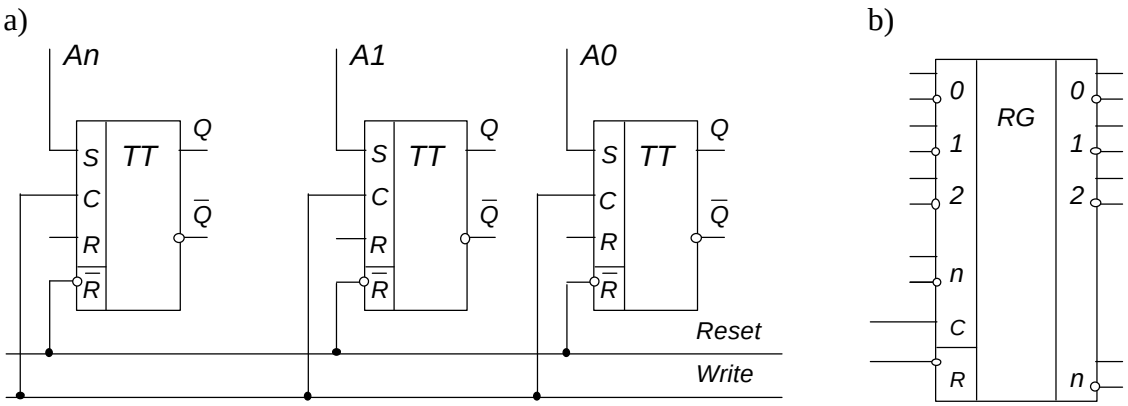
$$Q(t+1) = D(t). \tag{1.27}$$

Universaalsed *JK*-trigerid on lihtsate täiendustega muudetavad nii seade-, loendus- kui ka andmesisenditega trigeriteks. Sisendid *J* (*jump*) ja *K* (*key*) vastavad sisenditele *S* ja *R*, s. t signaal 1 sisendis *J* viib trigeri olekusse 1 ning signaal 1 sisendis *K* olekusse 0 sõltumata sellest, mis olekus triger varem oli. Erinevalt *RS*-trigerist võib *JK*-trigeri sisenditesse *J* ja *K* anda korraga signaalid 1, mis muudab trigeri oleku vastupidiseks. Seega toimib *JK*-triger niisugusel juhul nagu loendussisendiga *T*-triger. *JK*-triger toimib vastavalt loogika-funktsioonile

$$Q(t+1) = \overline{K}(t)Q(t) \vee J(t)\overline{K}(t) \vee J(t)Q(t). \tag{1.28}$$

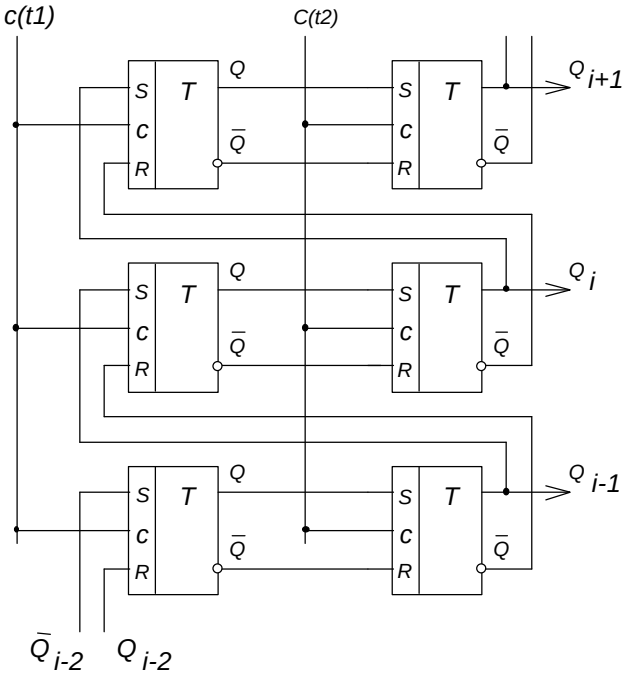
1.3.2. Registrid

Registriks nimetatakse trigeritest koosnevat seadet, mis võimaldab salvestada, säilitada ning taasesitada infot ühe sõna kaupa. Lisaks nihutatakse registri abil infosõna bitte vasakule või paremale. Sõna nihutamisega muundatakse rööpkoodis esitatud info jadakoodiks ning vastupidi jadakoodis esitatud info rööpkoodiks. Sõna pikkus sõltub registri trigerite arvust ning võib olla väga erinev. Enam on levinud 8-, 16-, 24- ja 32-bitised registrid, mis vastavad sõnapikkusele 1, 2, 3 ja 4 baiti. Registri põhimõtteskeem on joonisel 1.9. Registrit juhitakse signaalidega vastuvõtt ehk kirjutus (*write*) ja 0-seade (*reset*). Signaaliga *write* kirjutatakse sisendite *A0...An* informatsioon registrisse, signaaliga *reset* aga kustutatakse sealt. Juhul kui info kirjutatakse trigeritesse mõlema sisendi *S* ja *R* kaudu parakoodis (otse ja inverteeritult), pole eelnenud informatsiooni kustutamine (*reset*) vajalik ning registril puudub vastav juhtimissisend.



Joonis 1.9. Register: a) põhimõtteskeem, b) tähis

Nihkeregistri koostamiseks kasutatakse nii *RS*-, *D*- kui ka *JK*-trigereid. *RS*-trigeritega nihkeregistri skeem on joonisel 1.10. Trigeri otsene ja inverteeritud väljund ühendatakse järgmise trigeri seadesisenditega *S* ja *R*. Seega toimub iga taktiga infosõna nihutamine ühe biti võrra. Sõltuvalt sellest kuidas trigerid omavahel ühendatakse, nihkub infosõna kas paremale või vasakule. Iga takti keskel nihutab sünkrosignaal info trigerite esimestest astmetest teistesse.



Joonis 1.10. Nihkeregister

1.3.3. Loendurid

Loenduriks nimetatakse impulsside loendamiseks ette nähtud loogikalülitust. Loendureid kasutatakse nii automaatikaseadmetes kui ka arvutustehnikas. Energeetikas tarvitatakse loendure näiteks elektriarvestites, elektriajamite asendiandurites jm. Loendure liigitatakse summeerivateks (päripidi loendavateks), lahutavateks (tagurpidi loendavateks) ja reversiivseteks. Sõltuvalt signaali ülekande viisist loenduri trigerite vahel jaotatakse loendure jada- ja rööpülekandega loenduriteks.

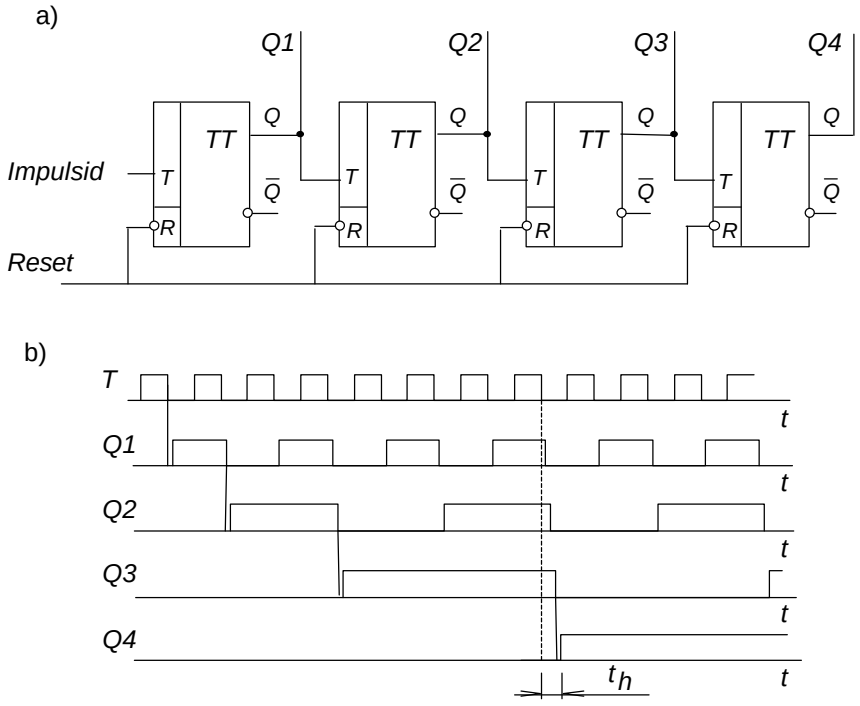
Jadaülekandega loendur koosneb järjestikku lülitatud *T*-trigeritest (joonis 1.11). Iga sisendimpulss *x* lülitab oma tagafrondiga ahela esimese trigeri ringi. Iga kahe sisendimpulsi järel lülitub trigeri väljund korraks sisse ja välja, s. t tema väljundimpulsside muutumise sagedus on kaks korda väiksem kui sisendimpulssidel. Võib öelda, et loendussisendiga

triger jagab impulsside sageduse kahega. Ahela teise trigeri väljundis on sagedus 4 korda, kolmanda trigeri väljundis 8 korda, neljanda trigeri väljundis 16 korda jne väiksem.

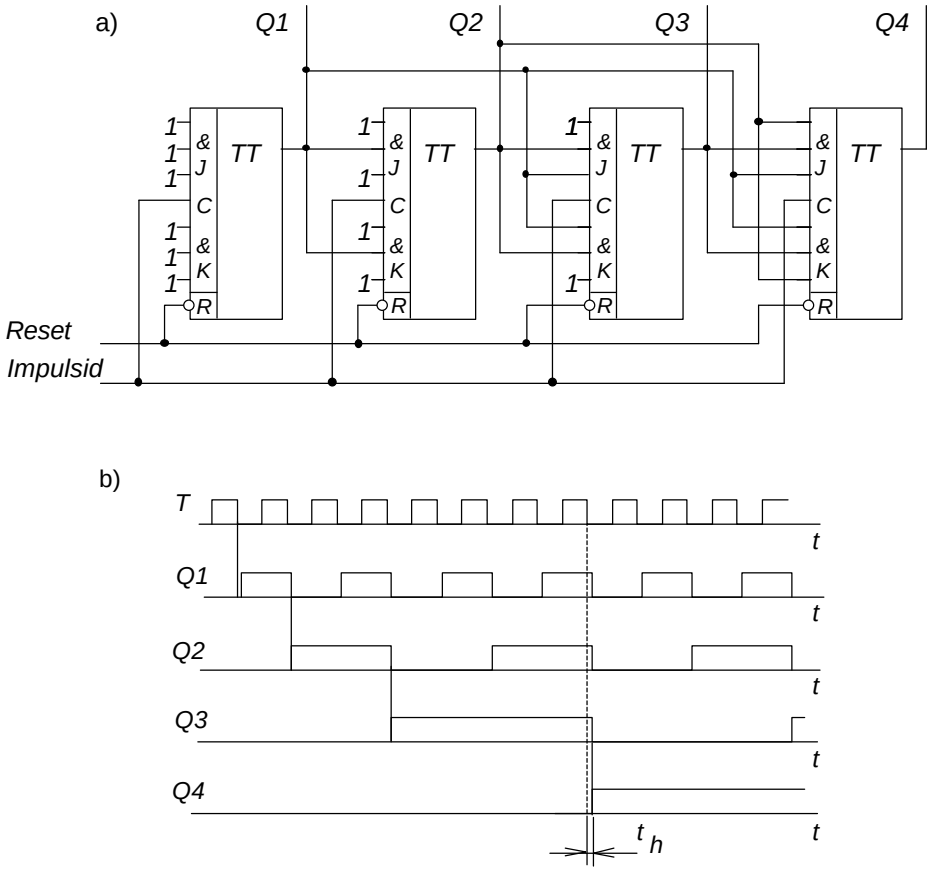
Jadaülekandega loenduri puuduseks on signaali ülekandel tekkiv hilistumine t_h, mis suureneb koos loenduri astmete arvuga. Suure loendusastmete arvu ning taktiimpulsside sageduse korral võib hilistumine ületada takti kestuse. Sel juhul ei vasta loenduri väljundsignaal enam tegelikult loendatud impulsside arvule ning süsteemis tekib viga. Vea vältimiseks tuleb vähendada taktiimpulsside sagedust, mis omakorda alandab kogu seadme töökiirust.

Rööpülekandega loendurit kasutatakse suure töökiirusega seadmetes. Võrreldes jadaülekandega loenduriga toimub trigeritevaheline signaalide ülekanne kõigi astmete jaoks korraga ning seetõttu ei sõltu hilistumine loenduri astmete arvust. Rööpülekandega loenduri skeem ja signaalidiagramm on joonisel 1.12.

Rööpülekandega loenduri iseärasuseks on, et sisendimpulsid antakse kõikidele trigeritele korraga ning eelmiste astmete väljundid lülitatakse järgmiste astmete trigerite sisenditesse. Nii valmistatakse järgmised astmed ette ümberlülitumiseks, mis toimub sisendimpulsi saabumisel sõltumatult sellest, kus ahela osas triger asub. Lülituse puuduseks on, et ahela trigerite arvu suurenemisel kasvab ka vajalik sisendite arv ning skeemi keerukus. Seepärast pole rööpülekandega loenduril tavaliselt rohkem kui 4...5 astet. Rööpülekandega loenduri eeliseid saab kasutada juhul, kui lülitada ta rühmaülekandega loenduri skeemi. Rühmaülekandega loendur koosneb mitmest 4-järgulisest rööpülekandega loendurist, mille vahel kasutatakse signaali jadaülekannet. Tänu sellele väheneb loenduri summaarne hilistumine 4 korda.

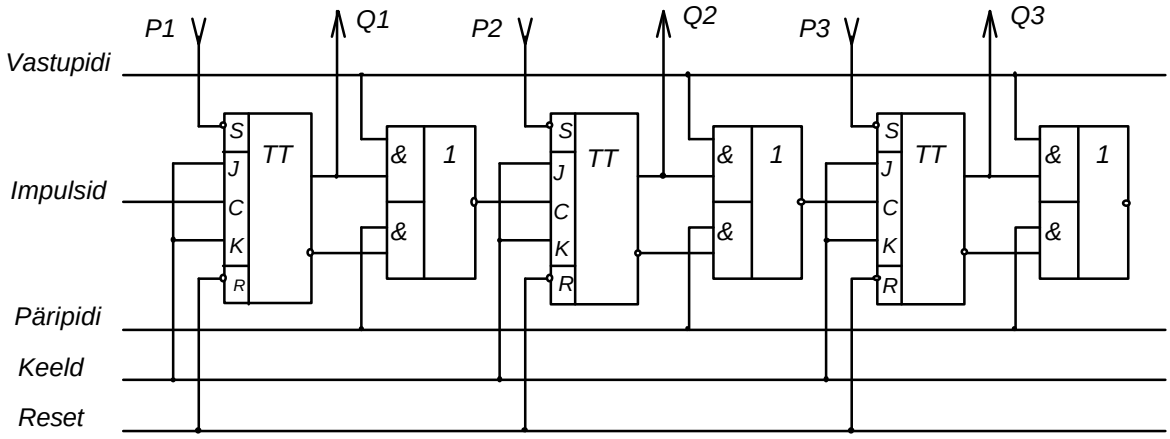


Joonis 1.11. Jadaülekandega loendur: a) skeem, b) signaalidiagramm

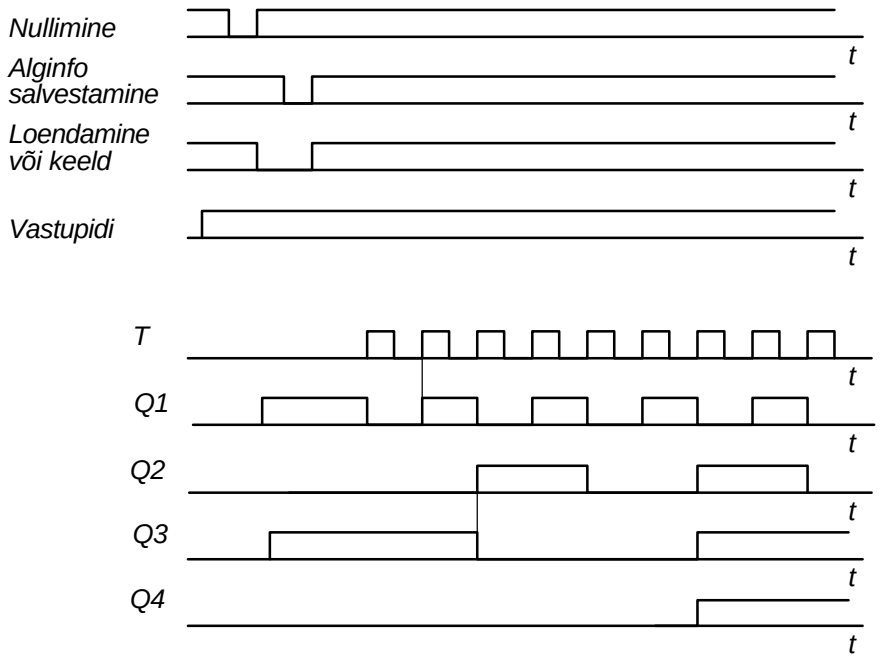


Joonis 1.12. Rööpülekandega loendur: a) skeem, b) signaalidiagramm

Reversiivloendur loendab impulsse nii päri- kui ka vastupidi. Loendussuuna muutumine toimub sõltuvalt sellest, kas ülekandeks kasutatakse trigeri otsest või inverteeritud signaali. Trigerite väljundsignaalide kommuteerimiseks rakendatakse kõigi astmete vahel täiendavat 2NING-VÕI-EI-loogikalülitust (joonis 1.13).



Joonis 1.13, a. Reversiivloenduri põhimõtteskeem



Joonis 1.13, b. Reversiivloenduri signaalidiagramm

1.3.4. Summaatorid

Summaatoriks nimetatakse arvuti loogikalülitust, mis on ette nähtud arvkoodide aritmeetiliseks summeerimiseks. Mitmejärgulise kahendarvu summaator koosneb mitmest ühejärgulisest summaatorist. Arvu summeerimisel tuleb lisaks kahe summeeritava arvu vastavatele järkudele liita nendega ka nooremate järkude summeerimisel tekkinud ülekanne. Seega on ühejärgulisel summaatoril kolm sisendit ning kaks väljundit. Summaatori loogikatabeli ning loogikafunktsiooni saab tuletada tavapärasest arvude tulba liitmise skeemist:

$$
\begin{array}{cccccc}
 & C_4 & C_3 & C_2 & C_1 & \\
 & & a_3 & a_2 & a_1 & a_0 \\
+ & & b_3 & b_2 & b_1 & b_0 \\
\hline
 & S_4 & S_3 & S_2 & S_1 & S_0 \\
 & & C_4 & C_3 & C_2 & C_1
\end{array}
$$

Vastavalt liitmise skeemile ning loogikale leitakse kõigile sisendite kombinatsioonidele väljundite väärtused ning esitatakse need tabelina. Seda tabelit 1.8 nimetatakse summaatori loogikatabeliks.

Tabel 1.8

Summaatori loogikatabel

a_i	b_i	P_i	S_i	P_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Loogikatabeli põhjal kirjutatakse väljundite jaoks loogikafunktsioonid:

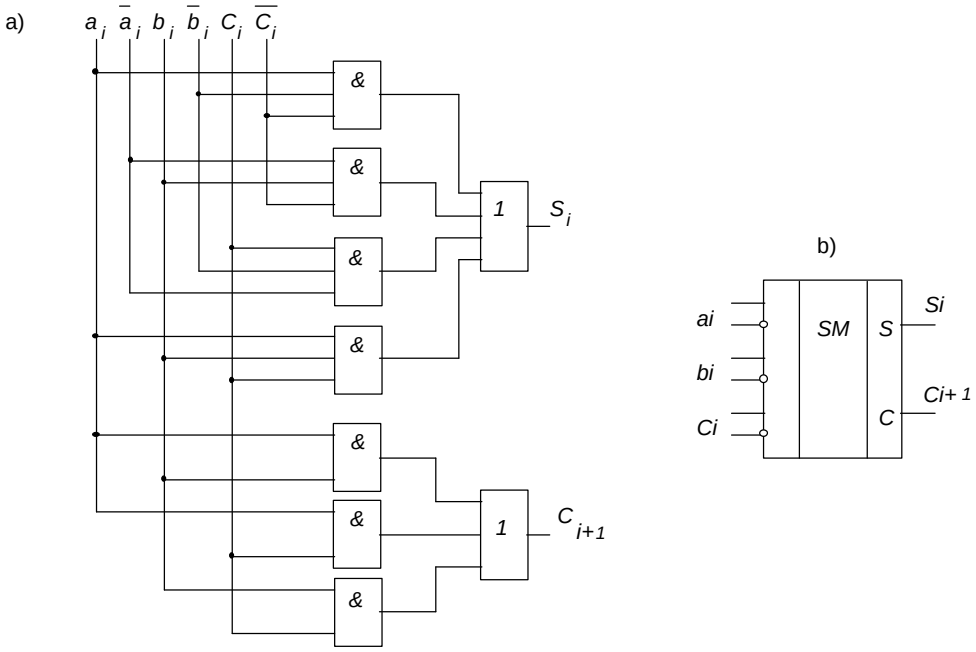
$$S_i = a_i \cdot \bar{b} \cdot_i \overline{C_i} \vee \bar{a}_i \cdot b_i \cdot \overline{C_i} \vee \bar{a}_i \cdot \bar{b_i} \cdot C_i \vee a \cdot b_i \cdot C_i; \qquad (1.29)$$

$$C_{i+1} = a_i \cdot b_i \cdot \overline{C_i} \vee a_i \cdot \bar{b_i} \cdot C_i \vee \bar{a}_i \cdot b_i \cdot C_i \vee a_i \cdot b_i \cdot C_i. \qquad (1.30)$$

Viimase avaldise saab lihtsustada kujule

$$C_{i+1} = a_i \cdot b_i \vee a_i \cdot C_i \vee b_i \cdot C_i. \qquad (1.31)$$

Vastavalt võrranditele (1.29) ja (1.31) koostatakse ühejärgulise (ühebitise) kahend-summaatori loogikaskeem (joonis 1.14). Samas on näidatud ka summaatori tingmärk.

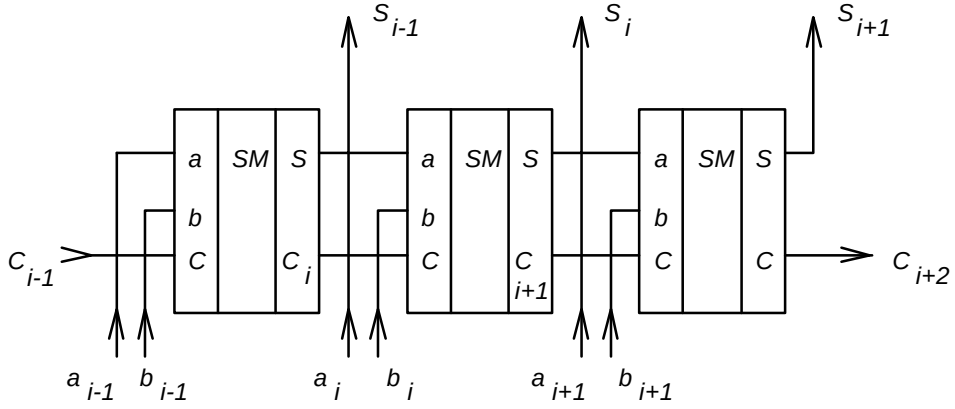


Joonis 1.14. Ühebitine täissummaator: a) loogikaskeem, b) tingmärk

Mitmejärgulised kahendsummaatorid jagunevad:

1) jadaülekandega,
2) rööpülekandega ja
3) rühmaülekandega summaatoriteks.

Jadaülekandega summaatoris moodustatakse väljundsignaal arvukohtade järjestikku summeerimisega, alates kõige nooremast (parempoolsest) kuni kõige vanema ehk vasakpoolsemani välja. Seega moodustatakse arvu summa ja ülekandesignaalid kõige nooremas kohas ning alles pärast seda summeeritakse arvude järgmised kohad. Arvukoha summeerimiseks ja ülekande moodustamiseks kulub teatud aeg, mida ülekande seisukohalt võib vaadelda hilistumisena. Kuna ülekanne toimub järjestikku, siis aeglustab see summaatori tööd. Suure kohtade arvu korral on koguhilistumine võrdne hilistumiste summaga üksikutes kohtades.



Joonis 1.15. Jadaülekandega kahendsummaatori loogikaskeem

Rööpülekandega summaatorid töötavad palju kiiremini kui jadaülekandega summaatorid. Mitmekohalise kahendarvu summeerimisel moodustatakse ülekanne korraga kõigi kohtade jaoks. Seetõttu ei kulu ülekandeks lisaaega ning summaator töötab kiiremini kui jadaülekande korral. Rööpülekandega summaatori tööpõhimõte on järgmine.

Ülekanne $i+1$ järku on avaldatav võrrandiga

$$C_{i+1} = a_i \cdot b_i \vee C_i (a_i \vee b_i). \qquad (1.32)$$

Ülekanne i järku on omakorda avaldatav võrrandiga

$$C_i = a_{i-1} \cdot b_{i-1} \vee C_{i-1} (a_{i-1} \vee b_{i-1}). \qquad (1.33)$$

Nii jätkates saab kirjutada ülekande avaldised summaatori kõigi kohtade jaoks kuni noorema kohani välja. Kui asendada seejärel ülekanded, alates kõige nooremast, vastavate avaldistega, siis

$$C_{i+1} = a_i \cdot b_i \vee a_{i-1} \cdot b_{i-1} (a_i \vee b_i) \vee a_{i-2} \cdot b_{i-2} (a_i \vee b_i)(a_{i-1} \vee b_{i-1}) \vee$$

$$\vee \text{ L } \vee a_1 \cdot b_1 (a_i \vee b_i)(a_{i-1} \vee b_{i-1}) \text{ L } (a_2 \vee b_2) \vee C_0 (a_i \vee b_i)(a_{i-1} \vee b_{i-1}) \text{ L } (a_2 \cdot b_2)(a_1 \vee b_1). \qquad (1.34)$$

Valemi (1.34) järgi võib konstrueerida skeemi, mis moodustab ülekanded summaatori kõigi kohtade jaoks korraga. Suure kohtade arvu puhul muutub skeem aga sedavõrd keeruliseks, et rööpülekandega summaatori ehitamine osutub ebaotstarbekaks. Seepärast rakendatakse rööpülekande põhimõtet kombineeritult koos jadaülekandega. Vastavaid summaatoreid nimetatakse **rühmaülekandega summaatoriteks**.

1.3.5. Kommutaatorid

Kommutaatorid jagunevad multipleksoriteks ja demultipleksoriteks. Nende tööpõhimõtte paremaks mõistmiseks on joonisel 1.16 näidatud kommutaatorite kontaktaseskeemid. **Multipleksoril** on mitu sisendit ja üks väljund. Sisendid jagunevad infosisenditeks ja juhtsisenditeks, kusjuures infosisendite arv määrab ära juhtsisendite arvu ning vastupidi. Vastavalt juhtsignaalile kommuteeritakse multipleksori väljundisse signaal ühest infosisendist. Kommuteeritavate infosisendite arv võrdub 2^n, kus n on juhtsisendite arv. Järelikult saab kahe juhtsisendiga ehk kahebitise koodiga kommuteerida 4 sisendit, kolme juhtsisendiga 8 sisendit jne.

Nelja ja kaheksa sisendiga multipleksori tööd kirjeldavad loogikavõrrandid:

$$Y_{1-4} = x_0\bar{u}_1\bar{u}_0 + x_1\bar{u}_1u_0 + x_2u_1\bar{u}_0 + x_3u_1u_0, \tag{1.35}$$

$$Y_{1-8} = x_0\bar{u}_2\bar{u}_1\bar{u}_0 + x_1\bar{u}_2\bar{u}_1u_0 + x_2\bar{u}_2u_1\bar{u}_0 + x_3\bar{u}_2u_1u_0 + \mathrm{L}$$

$$\mathrm{L} + x_4u_2\bar{u}_1\bar{u}_0 + x_5u_2\bar{u}_1u_0 + x_6u_2u_1\bar{u}_0 + x_7u_2u_1u_0. \tag{1.36}$$

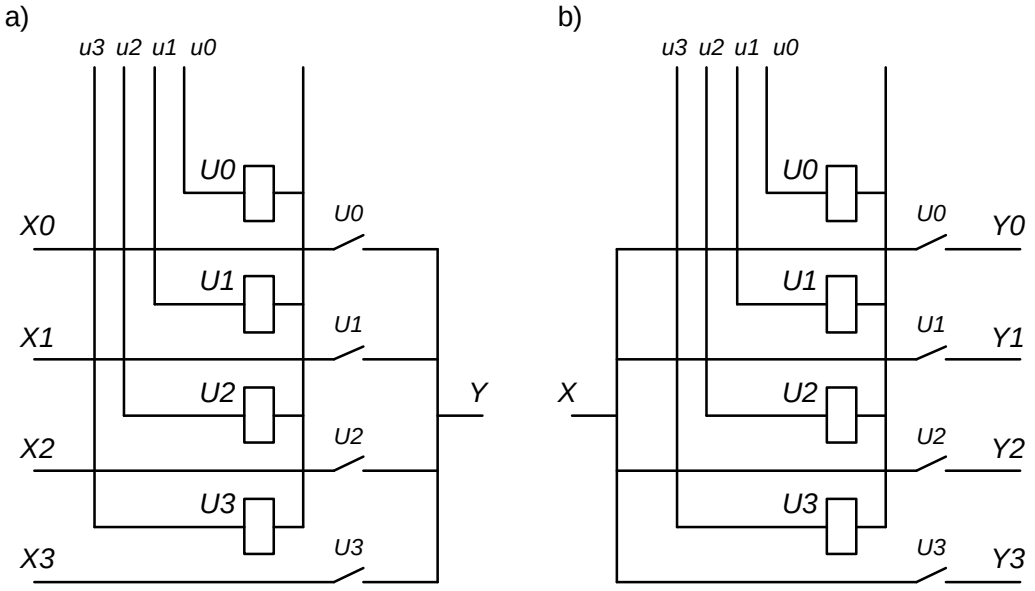
Demultipleksoril on üks infosisend ja mitu väljundit. Juhtsisendite arv sõltub väljundite arvust ja vastupidi. Vastavalt juhtsignaalile kommuteeritakse infosisendi signaal ühte väljundisse. Väljundite arv on 2^n, kus n on juhtsisendite arv. Nelja väljundiga demultipleksori tööd kirjeldavad järmised loogikavõrrandid:

$$Y_0 = x\bar{u}_1\bar{u}_0, \quad Y_1 = x\bar{u}_1u_0, \quad Y_2 = xu_1\bar{u}_0, \quad Y_3 = xu_1u_0. \tag{1.37}$$

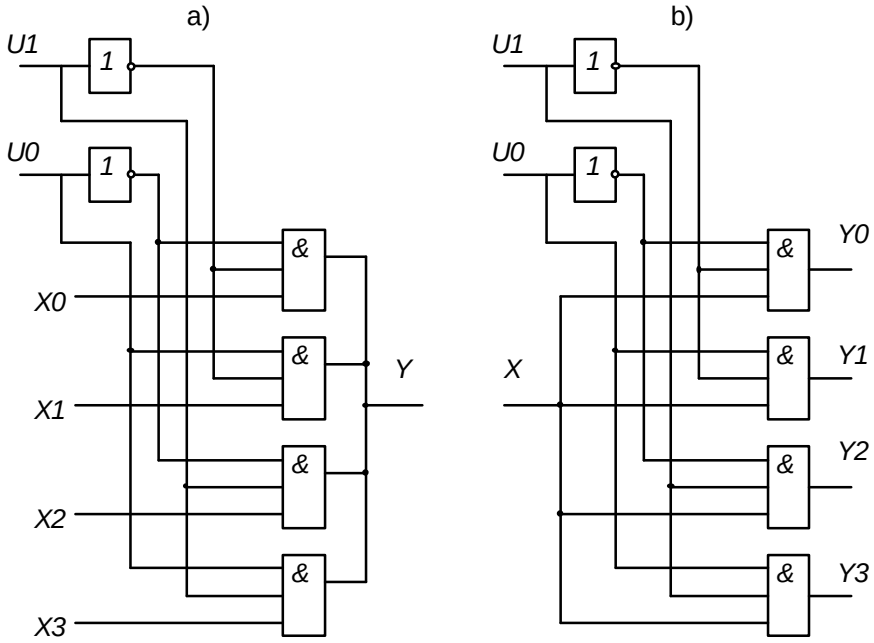
Kaheksa väljundiga demultipleksori tööd kirjeldavad võrrandid:

$$\begin{aligned} Y_0 &= x\bar{u}_2\bar{u}_1\bar{u}_0, & Y_4 &= xu_2\bar{u}_1\bar{u}_0, \\ Y_1 &= x\bar{u}_2\bar{u}_1u_0, & Y_5 &= xu_2\bar{u}_1u_0, \\ Y_2 &= x\bar{u}_2u_1\bar{u}_0, & Y_6 &= xu_2u_1\bar{u}_0, \\ Y_3 &= x\bar{u}_2u_1u_0, & Y_7 &= xu_2u_1u_0. \end{aligned} \tag{1.38}$$

Kommutaatorite loogikaskeemid on joonisel 1.17. Enam kasutatakse multipleksorit ja vähem demultipleksorit. Multipleksori väljundfunktsioon (võrrandid 1.35 ja 1.36) on esitatud loogikafunktsiooni täielikul disjunktiivsel normaalkujul. Järelikult saab piisava arvu sisenditega multipleksori abil realiseerida suvalisi loogikafunktsioone. Kolme muutuja loogikafunktsiooni realiseerimiseks tuleb kasutada multipleksorit $K_{1\text{-}4}$, nelja muutuja funktsiooni jaoks $K_{1\text{-}8}$ ja viie muutuja funktsiooni korral $K_{1\text{-}16}$.



Joonis 1.16. Kommutaatorite kontaktskeemid: a) multipleksor, b) demultipleksor



Joonis 1.17. Kommutaatorite loogikaskeemid: a) multipleksor, b) demultipleksor

1.3.6. Aritmeetika-loogikaplokk

Aritmeetika-loogikaplokk (*ALU - arithmetic logic unit*) on ette nähtud aritmeetika- ja loogikateheteks kahendarvudega (joonis 1.18) [1]. Kõik aritmeetikatehted sooritatakse arvude või nende täiendkoodide summeerimisega ja nihutamisega. Peamised loogikatehted on NING, VÕI, EI ja Mod2, mille täitmiseks on *ALU*-s vastavad loogikalülitused.

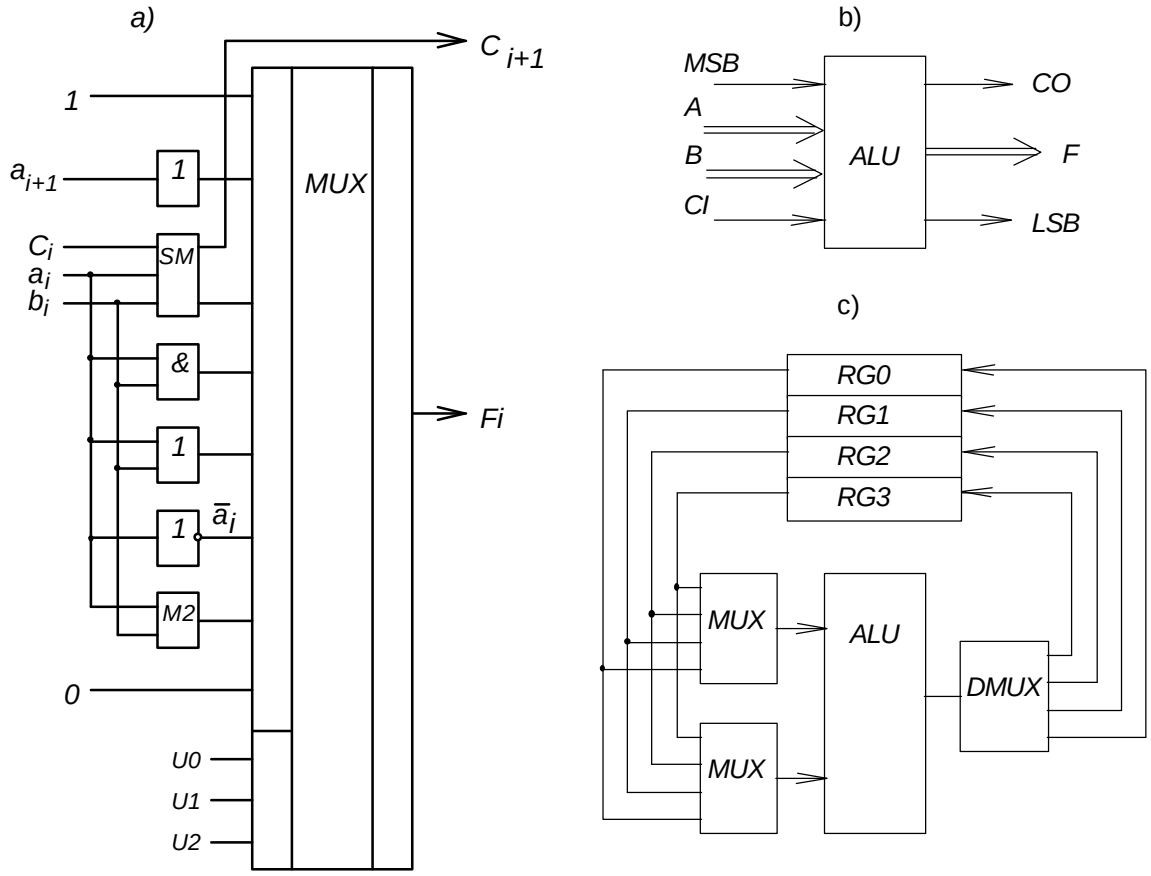
Erinevate tehete selekteerimiseks on aritmeetika-loogikaplokil kommutaator *MUX*. Mitmebitiste operandide $A = a_n, a_{n-1} \ldots a_1, a_0$, ja $B = b_n, b_{n-1} \ldots b_1, b_0$ ning bittide *MSB* (*most significant bit*) ja *CI* (*carry in*) summeerimisel kombinatsioonsummaatoriga saadakse kahendsumma $S = s_n, s_{n-1} \ldots s_1, s_0$ ning ülekandebitid *CO* (*carry out*) ja *LSB* (*least significant bit*). Ülekandebitt *CI* tähistab ülekannet kõrgemast bitist madalamasse ja *CO* vastupidi madalamast bitist kõrgemasse. Mitmebitise *ALU* madalaima ja kõrgeima biti sisendmuutujad on vastavalt *LSB* ehk madalaim bitt (viimane oluline bitt) ja *MSB* ehk kõrgeim bitt (kõige tähtsam bitt). Loogikatehetel ülekannet ei esine.

Multipleksor valib etteantud juhtkoodi $u_2\ u_1\ u_0$ järgi ühe funktsionaalsetest sisenditest ja suunab selle tulemi väljundisse F_i. Näiteks koodi 101 puhul $F_i = S_i$ (kahendliitmine ülekandega C_{i+1}), koodi 011 puhul $F_i = a_i \grave{U} b_i$ jne. Koodi 000 puhul $F_i = 0$ ja koodi 111 puhul $F_i = 1$.

Aritmeetikatehete operandide ja tulemite salvestamiseks kasutatakse registreid. Kahendsõnad suunatakse registritest *ALU* sisenditesse ja *ALU* väljundist registritesse multipleksorite ja demultipleksorite abil (joonis 1.18, c).

Otstarbekas on registreerida ka tehte tulemi teisi tunnuseid, nagu ületäitumine, nulltulem, negatiivne tulem jms. Selleks kasutatakse mikroprotsessoris olekuregistrit.

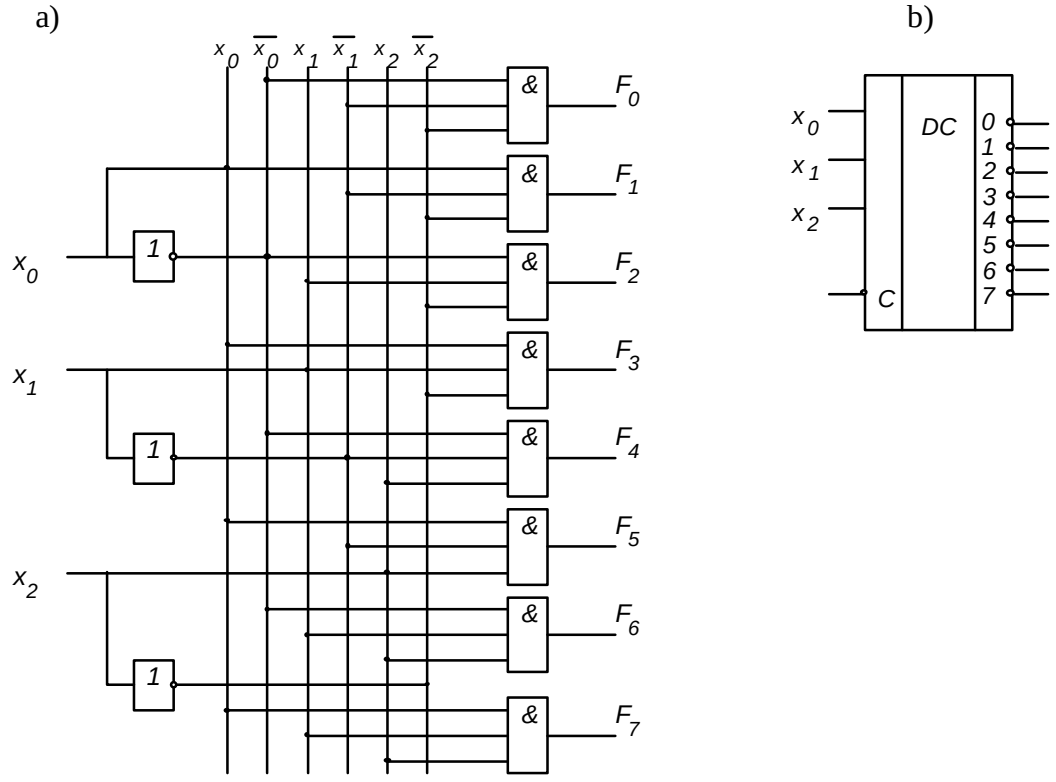
Teheteks mitmebitiste kahendarvudega kasutatakse ka vastava bittide arvuga *ALU*-sid. Mitmebitise *ALU* saab koostada ühebitistest *ALU*-dest. *ALU* loogikaskeem ja lihtsustatud tähis on joonisel 1.18.



Joonis 1.18. Aritmeetika-loogikaplokk: a) loogikaskeem, b) tingmärk, c) kasutamine koos kommutaatorite ja registritega

1.3.7. Koodrid ja dekoodrid

Dekooder on lülitus, mis on ette nähtud etteantud sisendkoodi muundamiseks soovitud väljundkoodiks. Ta tunneb ära sisestatava kahendarvu ja annab siganaali vastavasse väljundisse (joonis 1.19). Dekoodri ülesanneteks on muundada kahendkoodis arv niisuguseks koodiks, millega saab aktiveerida nõutava mälupesa, juhtida number- või tähtindikaatorit (joonis 1.20 ja 1.21), tunda ära mitmesuguseid kodeeritud signaale jne. Kuna dekoodri väljundisse ühendatavad seadmed on erinevad, siis kasutatakse nende juhtimiseks ka erinevaid dekoodreid. Näiteks on indikaatoritest levinumad 7-segmendilised vedelkristall- ja valgusdioodindikaatorid ning 10-numbrilised huumlahendusindikaatorid. Seitsmesegmendilise indikaatori dekoodril on reeglina 4 sisendit ning 7 väljundit, kümnenumbrilisel aga 4 sisendit ja 10 väljundit. Üldjuhul on dekoodril nii mitu sisendit *n*, kui mitu kohta on sisendisse antaval kahendarvul. Maksimaalne väljundite arv võrdub kombinatsioonide arvuga 2^n. Dekoodreid koostatakse peamiselt NING-elementidest.



Joonis 1.19. Kolmebitine kahendsignaali dekooder
a) loogikaskeem, b) tingmärk

Suure sisendite arvu korral kasutatakse dekodeerimiseks nn kaskaadlülitust, kus esimese astme dekooder aktiveerib ühe teise astme dekoodri ning see omakorda ühe väljundi.

Kahendkoodi dekoodri tööd kirjeldavad harilikult järgmised võrrandid:

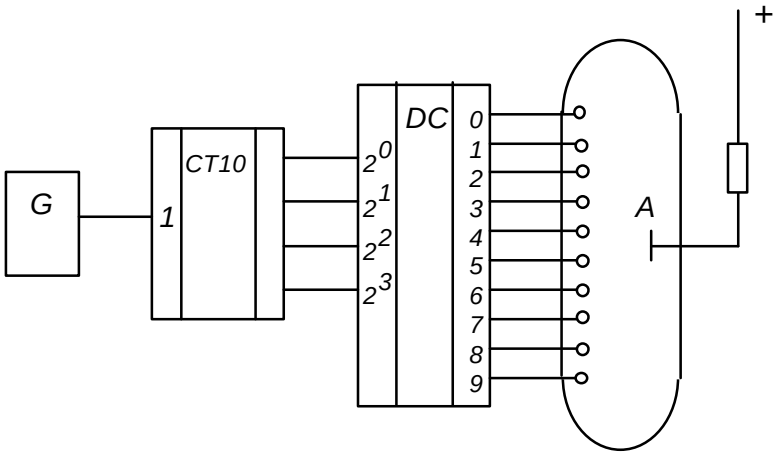
$$F_0 = \overline{x}_1\overline{x}_2 \ldots \overline{x}_{n-1}\overline{x}_n,$$

$$
\begin{aligned}
&F_1 = \overline{x}_1\overline{x}_2 \ldots \overline{x}_{n-1}x_n,\\
&F_2 = \overline{x}_1\overline{x}_2 \ldots x_{n-1}\overline{x}_n,\\
&\ldots\\
&F_{2^n-1} = x_1x_2 \ldots x_{n-1}x_n.
\end{aligned}
\qquad (1.39)
$$

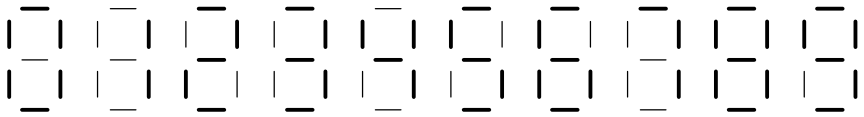
Seitsmesegmendilise indikaatori dekooder peab sisendisse antud kahendkoodi kohaselt lülitama sisse indikaatori segmendid nii, et hakkaks helendama arvule vastav kümnendnumber. Dekoodril on neli sisendit ja seitse väljundit. Kaheksandat, komasegmenti, dekoodriga ei juhita. Kuna segment *a* ei helendu numbrite 1 ja 4 korral, siis võib kirjutada, et

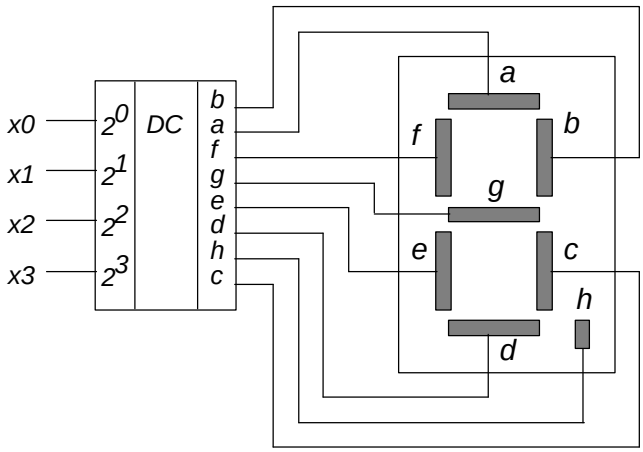
$$\overline{a} = \overline{x}_3\overline{x}_2\overline{x}_1x_0 + \overline{x}_3x_2\overline{x}_1\overline{x}_0 \qquad (1.40)$$

Analoogilised avaldised saab kirjutada ka kõigi ülejäänud segmentide kohta.



Joonis 1.20. Dekoodri kasutamine kümnenumbrilise huumlahendusindikaatori juhtimiseks





Joonis 1.21. Dekoodri kasutamine seitsmesegmendilise indikaatori juhtimiseks

1.4. Homogeensed struktuurid ja loogilised maatriksid

Loogikaseadmete koostamiseks kasutatakse peale põhielementide (NING, VÕI, EI) ka **universaalseid loogikalülitusi**. Need on loogilised polüfunktsionaalsed ehk mitmeotstarbelised lülitused, mille abil realiseeritakse suvalisi binaarloogika funktsioone. Universaalseteks loogikalülitusteks on kommutaatorid, programmeeritavad loogilised maatriksid (*PLM - programmable logic matrix*) ja mälud. Ühetaolise ja korrapärase ehituse tõttu nimetatakse neid ka homogeenseteks pooljuhtstruktuurideks. Üldjuhul nimetatakse homogeenseks polüfunktsionaalsetest elementidest koosnevat ühesuguste korduvate (iteratiivsete) sidemetega struktuuri. Polüfunktsionaalseteks nimetatakse elemente, mis realiseerivad mitmeid loogikafunktsioone. Soovitava funktsiooni valikuks peab polüfunktsionaalset elementi saama häälestada. Häälestamine toimub mitmeti: nii vajalike elementidevaheliste ühenduste lisamisega kui ka olemasolevate ühenduste katkestamisega. Näiteks olgu püsimälud ja programmeeritavad loogilised maatriksid (*PLM*). Samuti on olemas ümberprogrammeeritavad püsimälud ja loogilised maatriksid.

Struktuurielementide ühetaolisus on tehnikaseadmete oluline näitaja, see võimaldab lihtsustada seadmete valmistamist, suurendada nende töökindlust ning alandada hinda. Lihtsatest ühetaolistest elementidest valmistatud regulaarsete sidemetega süsteemid lahendavad väga keerulisi ülesandeid. Homogeense struktuuriga pooljuhtlülitusi kasutatakse arvutustehnikas laialt.

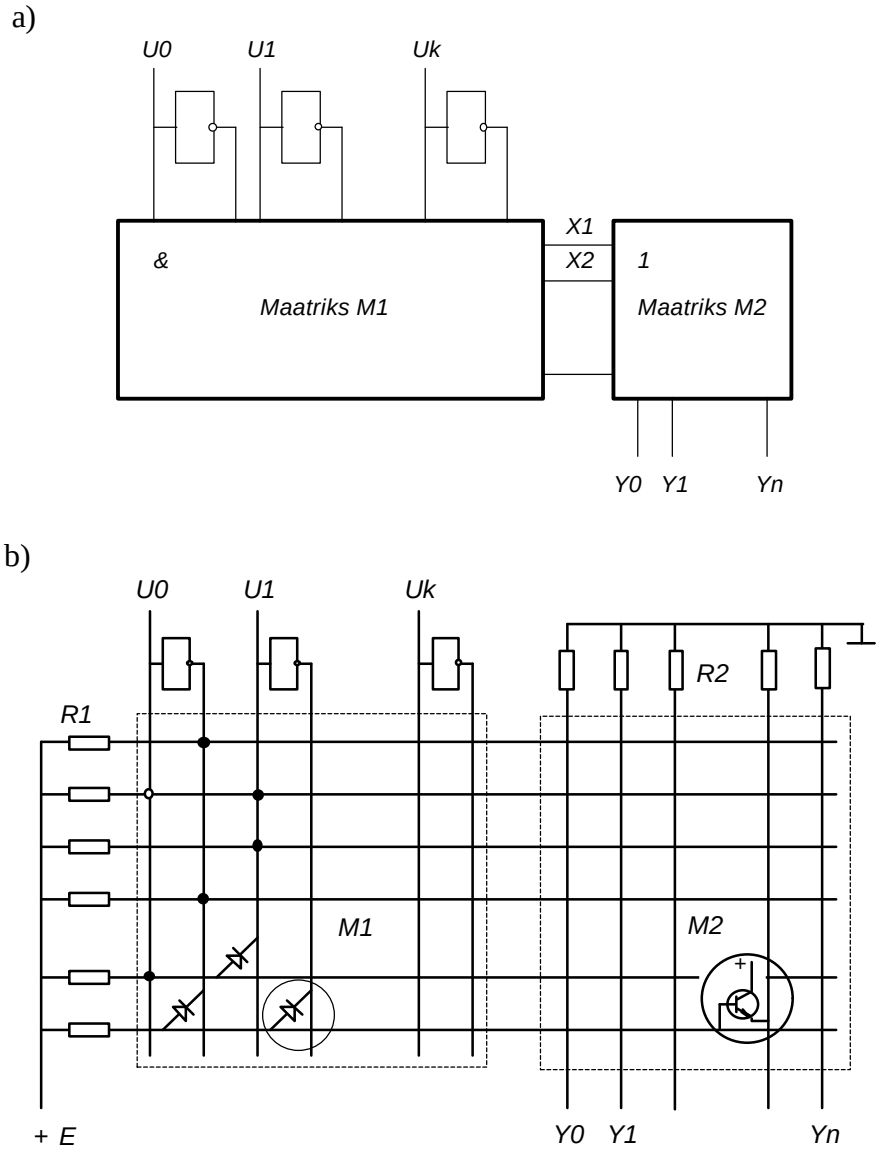
1.4.1. Loogilised maatriksid

Loogikafunktsioone esitatakse enamasti nn **disjunktiivsel normaalkujul**, s. t funktsioon avaldatakse loogiliste korrutiste loogilise summana, mis ei sisalda sulgusid. Niisuguste loogikafunktsioonide realiseerimiseks kasutatakse **loogilisi maatrikseid**, mille struktuuriskeem on joonisel 1.22, a, põhimõtteskeem joonisel 1.22, b. Antud lülituses jagunevad maatriksid omakorda NING- ja VÕI-maatriksiteks. Mõlemat liiki maatriksid kujutavad endast ristuvate siinide süsteemi, kus üksikjuhtmeid saab ristumiskohal omavahel ühendada või vastupidi olemasoleva ühenduse katkestada. Joonisel 1.22, b on rõht- ja püstjuhtmete ühenduskohad tähistatud punktiga. Tegelik ühendamine toimub aga pooljuhtelementidega, millest sagedamini kasutatakse dioode. Seepärast nimetatakse dioodidel põhinevaid maatrikseid **dioodmaatriksiteks**.

Joonisel 1.22 näidatud maatriks *M1* realiseerib NING-funktsiooni ja selle töö toimub järgmiselt. Sisendsignaalid u_0 ... u_k saabuvad maatriksi *Ml* püstjuhtmetele. Loogiliste EI-elementide abil leitakse nende signaalide inversioonid. Maatriksi *M1* rõhtjuhtmeid toidetakse takistite kaudu alalispingega *+E*. Kui sisendsignaaliga püstjuhe on dioodi kaudu ühenduses rõhtjuhtmega, nagu näidatud joonisel 1.22, b, siis kõrge sisendpotentsiaali ehk loogilise l korral jääb diood suletuks ning kõrge potentsiaal säilib ka rõhtjuhtmetes. Kui

sisendisse saabub madala potentsiaaliga signaal ehk loogiline 0, siis läbib dioodi vool, takistil tekib pingelang ning maatriksi rõhtjuhtme potentsiaal langeb samuti loogilise 0 tasemeni.

Kui ühe rõhtjuhtmega on samaaegselt ühendatud mitu sisendsignaalidega püstjuhet, siis säilib rõhtjuhtmes kõrge, s. o loogilisele l vastav potentsiaal vaid juhul, kui kõigis püstjuhtmetes on samuti kõrge potentsiaal. Vastupidisel juhul, kui kas või ühes neist on madal potentsiaal, langeb rõhtjuhtme potentsiaal samuti 0. Seega realiseerib iga rõhtjuhe loogilist NING-funktsiooni, mille sisendite arv vastab püstjuhtmetega ühendatud dioodide arvule. Maatriksi erinevate võimalike NING-funktsioonide arv vastab aga rõhtjuhtmete arvule. Nagu jooniselt näha, saab suhteliselt lihtsa maatriksiga, mil on homogeenne struktuur, asendada suurt hulka diskreetseid loogikaelemente.

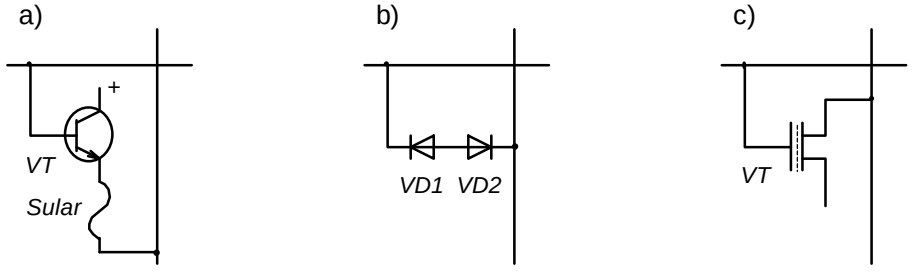


Joonis 1.22. Programmeeritav loogiline maatriks (*PLM*):
a) struktuuriskeem, b) põhimõtteskeem

Maatriksi *M1* väljundsignaalideks on konjunktsioonid, mis on omakorda disjunktiivse ehk VÕI-maatriksi *M2* sisendsignaalideks. Maatriksis *M2* kasutatakse rõht- ja püstjuhtmete ristumiskohtadel ühenduselementidena transistore, mille kollektorid on ühendatud toiteallika plussklemmiga, baasid maatriksi rõhtjuhtmetega ja emitterid püstjuhtmetega. Püstjuhtmed on takistite kaudu ühenduses ka toiteallika 0-klemmiga. Juhul kui maatriksi *Ml* väljundist saabub transistori baasile kõrge potentsiaaliga signaal 1, siis transistor avaneb ja toiteallika plussklemm ühendatakse läbi transistori maatriksi *M2* püstjuhtmega. Takistit *R2* läbib vool, mis tekitab takistil pingelangu. Pingelang takistil ongi maatriksi *M2* väljundsignaaliks. Järelikult, kui kas või üks maatriksi püstjuhtmetega ühenduses olevatest transistoridest on avatud, tekib väljundis kõrge potentsiaaliga enk loogilisele l vastav signaal. See tähendab, et maatriksi *M2* iga püstjuhe realiseerib loogilist VÕI-funktsiooni, mille maksimaalne võimalike sisendite arv vastab rõhtjuhtmete arvule. Maatriksi *M2* poolt realiseeritavate erinevate VÕI-funktsioonide arv võrdub aga püstjuhtmete arvuga. Kasutatavate sisendite arv ning funktsioonide sisu valitakse maatriksite häälestamisel, s. o vastavate ühenduste tegemisega dioodide ja transistoride ahelates. Seega saab loogiliste maatriksite abil lihtsalt realiseerida suure sisendite arvuga disjunktiivsel normaalkujul esitatud loogilisi funktsioone, kusjuures üleminek ühelt funktsioonide hulgalt teisele on suhteliselt lihtne.

Tehnoloogiliselt saab maatrikseid valmistada mitmeti. Kasutatakse maatrikseid, mille loogikafunktsioonid on ühenduselementide paigutusega fikseeritud juba maatriksi valmistamise käigus. Samuti on olemas maatriksid, kus loogikafunktsioone määravad ühendused teeb maatriksi kasutaja. Niisuguses maatriksis on potentsiaalsed võimalused suvalise loogikafunktsiooni realiseerimiseks, kui muutujate arv ei ületa maatriksi sisendjuhtmete arvu. Kasutamise lihtsustamiseks on maatriksis olemas ka kõik ühenduselemendid. See, kuidas on tehtud vajalikud ühendused, sõltub maatriksi valmistamise tehnoloogiast.

Üksikelementidest koostatud maatriksi ühendusi saab teha näiteks pistikutega kommutatsiooniväljas. Valdav osa loogilisi maatrikseid toodetakse integraallülitustena ning oma ehituselt sarnanevad need pooljuhtpüsimälule. Vajalikud ühendused tehakse maatriksi programmeerimisel, milleks kasutatakse spetsiaalseid arvutitega juhitavaid programmaatoreid. On olemas nii ühekordselt programmeeritavaid kui ka ümberprogrammeeritavaid maatrikseid. Maatriksite ja püsimälude tüüpilised elemendid on näidatud joonisel 1.23, *a*, *b*, *c*.



Joonis 1.23. Ümberprogrammeeritavates püsimäludes ja maatriksites kasutatavaid elemente:

a) sular emitteriahelas, b) elektrilise läbilöögiga lühistatav *pn*-siire,
c) laengukandjaga *MOS*-transistor

Ühenduselemendina võib kasutada transistori (joonis 1.23, a), mille emitteriahelas on sular. Loogilise maatriksi programmeerimisel põletatakse sular läbi ning ühendus katkeb. Allesjäänud ühendused tagavad maatriksi programmikohase töö. Ühenduse läbipõletamisvool on 20...30 mA.

Vajalikke elektrilisi ühendusi saab tekitada ka joonisel 1.23, b näidatud kahe vastulülituse dioodiga. Normaalses olukorras selline dioodipaar voolu ei juhi. Ühenduse tekitamiseks antakse juhtmele kõrgendatud pinge, mille tulemusena dioodi *VD2 pn*-siire lüüakse elektriliselt läbi ning rõht- ja püstjuhtmete vahel tekib dioodi *VDl* kaudu ühendus. Mõlemal juhul on tegemist ühekordselt programmeeritavate maatriksitega, sest ühenduselementide taastamine pole võimalik.

1.4.2. Ümberprogrammeeritavad maatriksid

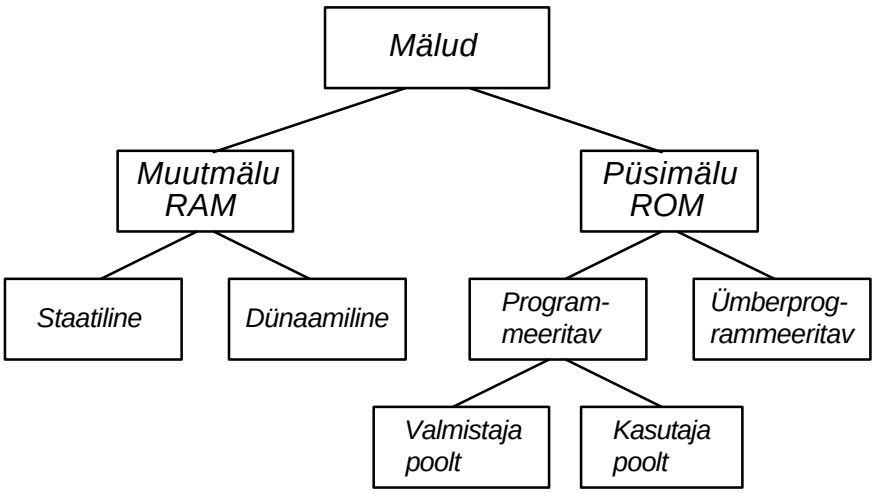
Ümberprogrammeeritavates maatriksites kasutatakse ühenduselemendina ujuvpaisuga *MOS*-transistore (joonis 1.23, c). Ujuvpaisul elektriline ühendus puudub ning see on ette nähtud laengu säilitamiseks. Transistori pais on ühendatud rõhtjuhtmega, suue püstjuhtmega ning läte toiteallika miinusklemmiga. Lähteolekus läbib paisu ergastamisel transistori vool. Programmeerimisel antakse püstjuhtmele 25...50 V pingeimpulss, mille tulemusena ujuvpais saab negatiivse laengu. Ujuvpais säilitab laengu ning transistori avamiseks tuleb paisule anda tavalisest märksa kõrgemat pinget. Hariliku juhtpinge korral jääb transistor suletuks ja vool transistori ei läbi. Transistori algolek taastub, kui tema siirdeid kiiritada 30...100 sekundit ultraviolettkiirgusega. Selleks on maatriksi või püsimälu integraallülituse keres ultraviolettkiirgust läbilaskev ava.

Programmeeritavad maatriksid võimaldavad realiseerida nii disjunktiivsel normaalkujul esitatud loogikafunktsioone kui ka keerukamaid näiteks sulgusid sisaldavaid avaldisi. Sel juhul realiseeritakse maatriksiga kõigepealt sulgudes olev funktsioon ning antakse sellele vastav väljundsignaal tagasi maatriksi vabasse sisendisse, kus edasi koos teiste sisendsignaalidega moodustatakse lõplik väljundsignaal. Põhimõtteliselt saab nii realiseerida ka mitmekordsete sisemiste sulgudega loogikafunktsioone.

1.5. Mälud

Mäluks nimetatakse informatsiooni salvestamiseks (kirjutamiseks), säilitamiseks ja lugemiseks ettenähtud seadmeid. Mälu iseloomustab mälu maht Kbaitides, Mbaitides või Ksõnades, infosõna pikkus bittides või baitides ning mälu töökiirus, s.t mälu poole pöördumise aeg mikrosekundites. Mälusid liigitatakse sõltuvalt tööpõhimõttest ning kasutusviisist. Üks võimalikke mälude liigitusi on joonisel 1.24 [1].

Muutmälu on seade informatsiooni (programmide, lähte- ja vaheandmete ning tulemite) lühiajaliseks salvestamiseks, säilitamiseks, otsinguks ning lugemiseks. Muutmälud jagunevad staatilisteks ja dünaamilisteks muutmäludeks.



Joonis 1.24. Pooljuhtmälude liigitus

Püsimälu kasutatakse programmide ning andmete pikaajaliseks säilitamiseks ja lugemiseks. Püsimälud jagunevad ühekordselt programmeeritavateks ja ümberprogrammeeritavateks püsimäludeks. Ühekordselt programmeeritavaid mälusid liigitatakse sõltuvalt sellest, kas need programmeeritakse tehases mälukiibi valmistaja poolt või programmeerib neid kiibi kasutaja. **Ümberprogrammeeritavaid püsimälusid** saab kasutaja vajaduse korral kustutada ja uuesti programmeerida.

Muut- ja püsimälude töökiirus peab olema võimalikult suur.

1.5.1. Muutmälud

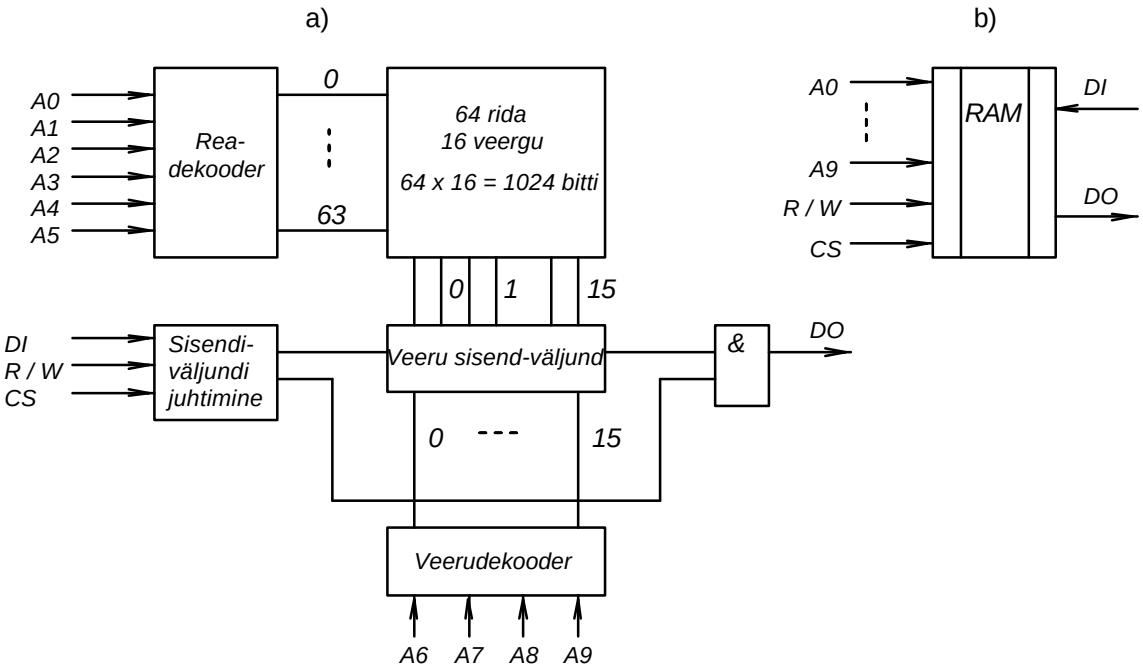
Muutmälude (*RAM - random access memory*) põhiliigiks on pooljuhtmälud, mis koosnevad trigeritest või muudest mäluelementidest. Muutmälud on toitepingest sõltuvad ning jagunevad kahte liiki, staatilisteks ja dünaamilisteks.

Staatilises muutmälus kasutatakse iga infobiti salvestamiseks ühte trigerit, mis säilitab infot seni, kuni säilib toitepinge. Kuna staatilises mälus säilib salvestatud informatsioon ka pärast mälust lugemist, püsides seal toitepinge olemasolu korral kui tahes kaua, siis nimetatakse niisugust mälu staatiliseks.

Lihtsaima staatilise muutmälu struktuur on joonisel 1.25. Mälul on 1024 aadressi ja tema kogumaht on 1024 bitti ehk 1024 pesa. Iga bitt on salvestatud trigerisse ning triger valitakse rea- ja veerudekoodri abil. Mälu juhtimiseks kasutatakse järgmisi signaale:

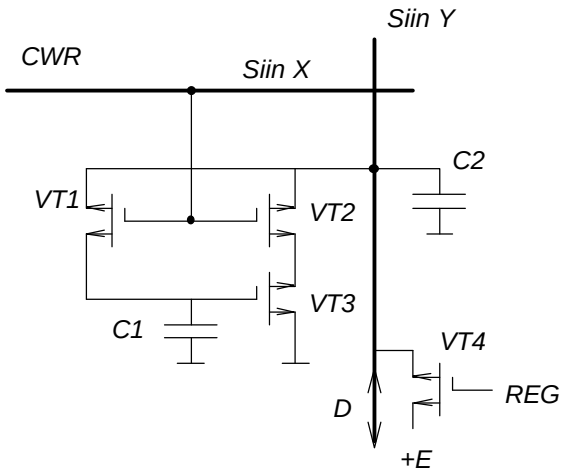
R/W = 1, (*read/write*) määrab ära lugemisrežiimi;
R/W = 0, määrab ära kirjutusrežiimi;
CS = 0, (*chip select*) lubab mälukiibist bitte lugeda (*D0*) või sellesse kirjutada (*D1*);
CS = 1, mäluelement on süsteemi tööst välja lülitatud ning ei reageeri aadressi $A_9...A_0$ koodile ega signaalile *R/W*.

Staatilise muutmälu struktuur ja kiibi tähis on joonisel 1.25. Andmesõna pikkuseks on tavaliselt 8, 16, 32 jne bitti. Vastavalt andmesõna pikkusele valitakse ka mäluelementide ühendamisviis.



Joonis 1.25. Staatiline muutmälu: a) struktuuriskeem, b) kiibi tähis

Dünaamilises muutmälus säilib info *MOSFET*-transistori paisu mahtuvuse elektrilaenguna. Tavaliselt säilib see laeng lekkevoolu tõttu väga lühikest aega. Seepärast tuleb info säilitamiseks laengut perioodiliselt näiteks iga 2 ms järel uuendada (regenereerida). Dünaamiline muutmälu on staatilise mäluga võrreldes lihtsama ehitusega (ühe biti salvestamiseks läheb vaja umbes kaks korda vähem elemente), suurema toimekiirusega ning tarvitab tööks vähem energiat. Dünaamilise muutmälu elemendi skeem on joonisel 1.26.



Joonis 1.26. Dünaamilise mäluelemendi skeem

Mäluna toimib transistori *VT2* paisuahela mahtuvus *C1*. Info kirjutatakse mällu ja loetakse sealt siini *Y* kaudu (signaal *D*). Enne info lugemist antakse signaal *REG*, mis avab transistori *VT4*, ning mahtuvus *C2* (siini *Y* parasiitmahtuvus) laetakse allikast *+E*. Seejärel antakse siinile *X* kirjutuse/lugemise sünkrosignaal *CWR*, mis avab transistori *VT3*, kuid ei saa avada transistori *VT2*.

Kui mäluelement säilitab olekut 1, siis on mahtuvus *C1* laetud ja transistor *VT2* on avatud. Sel juhul tühjeneb mahtuvus *C2* läbi avatud transistoride *VT2*, *VT3* ja signaali *D* 0-nivoo näitab, et mälus säilitati signaali 1 (inversne väljund). Kui mäluelement säilitab olekut 0, siis on mahtuvus *C1* tühjenenud, *VT2* suletud ja signaal *CWR* ei põhjusta mahtuvuse *C2* tühjenemist. Signaali *D* kõrge nivoo näitab , et mälus säilitati olekut 0 (inversne väljund).

Dünaamilisi muutmälusid regenereeritakse harilikult regenereerimissignaaliga *REG* ja koos sellega toimub mälu kõigi ridade järjestikune adresseerimine. Tavaline lugemine ega kirjutamine pole regenereerimise ajal võimalik, samuti ei saa regenereerimist alustada lugemise ega kirjutamise tsükli ajal. Regenereerimishetke kindlaksmääramine, kõigi rea-aadresside etteandmine, lugemise ja kirjutamise blokeerimine jms operatsioonid teevad dünaamiliste pooljuhtmälude kasutamise võrreldes staatiliste mäludega keeruliseks, sest nad nõuavad lisaelemente. Dünaamiliste muutmälude eeliseks on väike hind ja võimsustarve. Neid saab valmistada väga suure integratsiooniastmega, mis võimaldab toota

suure mälumahuga kiipe. Seepärast ehitatakse arvutite ja mikroprotsessorsüsteemide suuremad mäluseadmed tavaliselt dünaamilistest mälukiipidest.

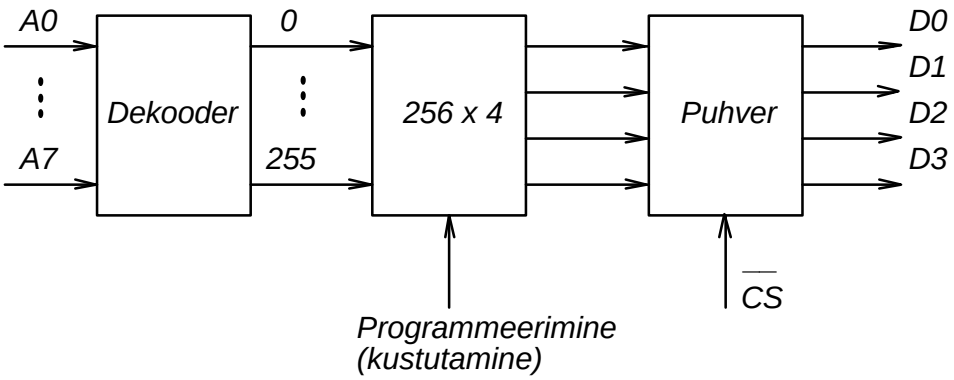
Kõigi muutmälude üheks oluliseks puuduseks on salvestise hävinemine toitepinge väljalülitumisel. Selle puuduse vältimiseks kasutatakse avariitoidet (katkematu toite allikaid) ning muid mäluseadmeid, kus informatsioon säilib teatud aja ka ilma toitepingeta.

1.5.2. Püsimälud

Püsimälu (*ROM - read only memory*) on mõeldud korduvaks informatsiooni lugemiseks. Info on salvestanud püsimällu kas pooljuhtmälukiibi valmistaja või kasutaja. Info salvestamist püsimällu nimetatakse püsimälu programmeerimiseks. Püsimälude tähtsamad alaliigid on järgmised: 1) programmeeritav püsimälu (*PROM - programmable read only memory*), 2) ümberprogrammeeritav püsimälu (*EPROM - erasable programmable read only memory*); 3) elektriliselt kustutatav ümberprogrammeeritav püsimälu (*EEPROM - electrically erasable programmable read only memory*).

Programmeeritavat püsimälu programmeeritakse kas tehases integraallülituse valmistamise käigus vastavate tehnoloogiliste maskidega või mikroprotsessorsüsteemi koostaja poolt spetsiaalsete programmaatorite abil. Esimest liiki *PROM* on tavaliselt masstoode, sest tehases salvestatakse sinna enamkasutatavad püsiandmed nagu standardkoodide teisendustabelid, keerukamate funktsioonide tabelesitused jms. Tarbija salvestab püsimällu tarbijaprogramme või juhtseadme mikroprogramme.

Ümberprogrammeeritava püsimälu (joonis 1.27) programmeerib kasutaja programmaatoriga, kuid salvestatud informatsiooni on võimalik hiljem kustutada ning püsimälu uuesti ehk ümber programmeerida. Mälu kustutatakse kas elektriliselt või ultraviolettkiirguse abil. Püsimäludes kasutatakse samu elemente kui ümberprogrammeeritavates loogilistes maatriksites. Vaatamata sellele et *EPROM*-i sisu saab hiljem muuta, on püsimälu ümberprogrammeerimine tülikas ning seepärast tuleb püsimäluprogramme hoolikalt kontrollida ja siluda ning alles siis salvestada.



Joonis 1.27. Ümberprogrammeeritava püsimälu struktuuriskeem

1.6. Diskreetsed automaadid

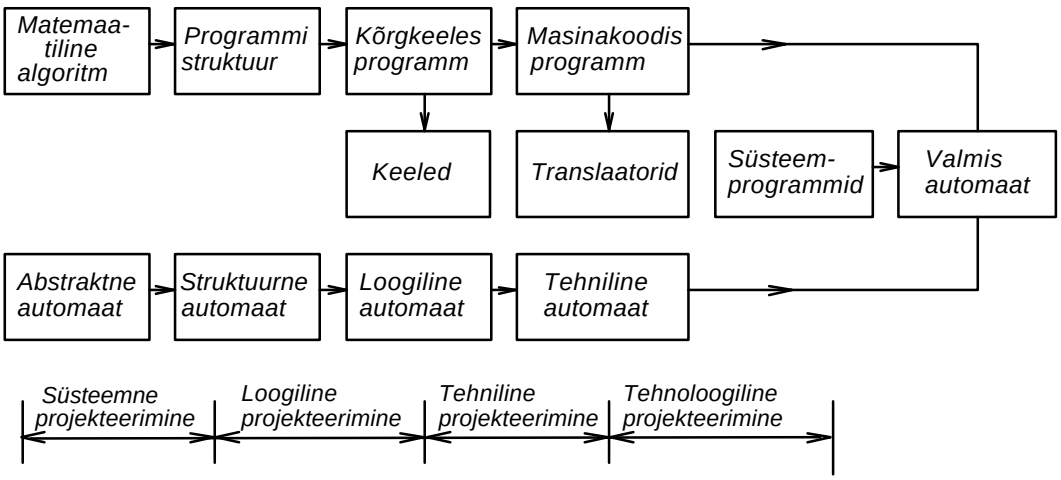
1.6.1. Diskreetsete automaatide olemus

Automaatide teooria põhineb diskreetsetel olekuvõrranditel. Enamikul juhtimisobjektidel on lõplik arv diskreetseid olekuid. Seepärast nimetatakse nende juhtseadmeid lõplikeks automaatideks. **Lõplike automaatide teooria** on aluseks erinevate diskreetsete juhtseadmete, sealhulgas arvutite ja robotite juhtseadmete väljatöötamisel. Lõplike automaatide uurimiseks pole tingimata vaja füüsilise automaadi olemasolu. Kõige enam kasutatakse nende uurimiseks mitmesuguseid matemaatilisi mudeleid, mida nimetatakse **abstraktseteks automaatideks**. Kuna abstraktseid automaate saab kirjeldada algoritmikeelte abil, siis tuleneb sellest abstraktsete automaatide ning algoritmikeelte ekvivalentsus, s. t neid keeli on võimalik asendada abstraktsete automaatidega ja vastupidi. Üheks levinumaks ja kõige üldisemaks abstraktseks automaadiks on nn **Turingi masin**. Selle esitas 1936. a inglise loogik A M Turing. Masina tähtsus põhineb Turing-Churchi teesil, mille kohaselt igasuguse algoritmi infotöötluse võib sooritada Turingi masinaga. See väide ei ole matemaatiliselt tõestatav, sest algoritmi infotöötluse mõiste pole matemaatiline, vaid intuitiivne. Katsed leida algoritmilisi protsesse kajastav formaalne eeskiri, mis oleks võimsam kui Turingi masin, on olnud edutud. Seepärast loetakse tänapäeval algoritmilisteks teisendusteks vaid teisendusi, mida saab teostada Turingi masinaga. **Abstraktne Turingi masin** on olnud aluseks arvutiteooria, programmeerimiskeelte ja keeletranslaatorite loomisel, ta on võimaldanud teoreetiliselt uurida juhtimisülesannete lahendatavust, s. t määrata nende kuulumist algoritmiliste protsesside hulka jms. Universaalseteks infotöötlusseadmeteks on füüsilised arvutid, mille näol abstraktne Turingi masin on realiseerunud tegelikkuses. Järelikult on kõiki algoritmidega esitatavaid juhtimisprotsesse võimalik teostada arvutiga.

Automaatide teoorias vaadeldakse ka mitmeid lihtsamaid väiksema üldistusastmega abstraktsete automaatide mudeleid, mis võivad olla aluseks ühtede või teiste juhtseadmete väljatöötamisel. Niisuguste automaatide näiteks on **Mealy** ja **Moore'i automaadid**.

Diskreetsete automaatide projekteerimisel on neli etappi. Automaadi loomine algab süsteemse projekteerimisega, mille käigus koostatakse automaadi struktuur, leitakse

põhilised plokid, määratakse nende otstarve ning funktsioonid. **Süsteemse projekteerimise** tulemuseks peab olema automaadi struktuuriskeem ning üksikute plokkide algoritmid. Seega minnakse abstraktselt automaadilt üle struktuursele automaadile. Järgmisel, **loogilise projekteerimise** etapil toimub automaadi loogiline süntees, tema sisemiste olekute minimeerimine ning automaadi ja selle plokkide funktsionaalskeemi koostamine. Üld- algoritmi järgi valitakse iga ploki ehitusviis ning loogikaelementide baas. Loogilisel projekteerimisel määratakse ka riist- ja tarkvara vahekord juhtseadmes. Loogiline projekteerimine peab andma automaadi kõikide plokkide funktsionaalskeemid ning nende struktuuri kirjeldused. Selleks kasutatakse automaatide loogiliseks sünteesiks ettenähtud erikeeli. Kolmandaks etapiks on automaadi **tehniline projekteerimine**, mille käigus lahendatakse kolm põhiülesannet: valitakse elemendid (integraallülitused), koostatakse nende paigutus- ja ühendusskeem trükiplaatidel ning trükiplaatide ja pistikühenduste paigutusskeem juhtseadme kapis. Erijuhul võib tehniline projekteerimine tähendada ka spetsiaalse integraallülituse väljatöötamist. Tehnilise projekteerimise tulemuseks on juhtautomaadi tehniline dokumentatsioon, mille põhjal saab valmistada juhtautomaadi. Automaadi loomise viimaseks etapiks on **tehnoloogiline projekteerimine**. Selle käigus lahendatakse trükiplaatide valmistamise tehnoloogiaga, karkasside ja kappide valmistamisega, seadmete kontrolliga jms seotud probleeme. Tehnoloogilise projekteerimise tulemusel valmib dokumentatsioon, mille põhjal võib alustada juhtautomaatide saritootmist. Juhtautomaadi riist- ning tarkvara projekteerimise ja valmistamise etappidest annab ülevaate joonis 1.28.



Joonis 1.28. Juhtautomaadi projekteerimine ja valmistamine

Juhtautomaadi kirjeldamist on otstarbekas alustada abstraktse automaadi tundmaõppimisest. Automaati vaadeldakse kui musta kasti *A* (joonis 1.29), tema sisend- ja väljundsignaale aga kui tähestiku tähti. Kuna automaat on diskreetne, siis on abstraheeritud ka aja kulg, mis võib omandada vaid diskreetseid järjestikuseid väärtusi, näiteks $t = 0, 1, 2, ...$ Üldjuhul on niisugune automaat *A* kirjeldatav

§ sisendtähestikuga ehk sisendsignaalide hulgaga

$$U = \{ u_O, u_l, ... u_i, ... u_I \}; \qquad (1.41)$$

§ väljundtähestikuga ehk väljundsignaalide hulgaga

$Y = \{ y_0, y_1, \ldots y_j, \ldots y_m \};$ (1.42)

§ olekutähestikuga ehk olekusignaalide hulgaga

$X = \{ x_0, x_1, \ldots x_k, \ldots x_n \};$ (1.43)

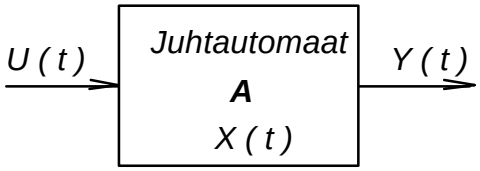
§ siirdefunktsiooniga

$S = f(x_k; u_i);$ (1.44)

§ väljundfunktsiooniga

$V = f(x_k; u_i);$ (1.45)

§ automaadi **algolekuga** x_0, mis vastab hetkele $t_0 = 0$.



Joonis 1.29. Abstraktne automaat

Siirde- ja väljundfunktsioonid määravad automaadi oleku *X(t+1)* ja väljundsignaali *Y(t+1)* hetkel *(t+1)* sõltuvalt olekust *X(t)* ja sisendsignaalist *U(t)* hetkel *t*. Kui automaadi sisendite ja väljundite arv on suurem kui üks, võib signaali *U, Y* ja *X* vaadelda kui vektoreid või mitmetähelisi sõnu. Abstraktse automaadi töötamisel toimub sisendsõnade muutumine väljundsõnadeks, kusjuures protsessis etendab olulist osa automaadi sisemine olek antud hetkel. Iga järgmine olek oleneb eelmisest. Et väljundsignaalide ja olekute vahetumine toimuks soovitud korrapärasusega, tuleb automaadi mällu salvestada programm ning ette anda algolek hetkel $t = 0$. Kuigi kõigi diskreetsete automaatide olekud ja signaalid muutuvad diskreetsetel hetkedel, liigitatakse automaadid sõltuvalt nende ajalisest käitumisest sünkroonseteks ja asünkroonseteks. **Sünkroonsetes automaatides** toimivad sisendsignaalid täpselt fikseeritud hetkedel, automaadi sisemine olek muutub aga ajal, mil sisendsignaalide toime puudub. Ajahetkede fikseerimiseks on sünkroonses automaadis olemas taktiimpulsside generaator. **Asünkroonsetes automaatides** toimub olekute muutumine suvalistel hetkedel, mis on määratud ainult sisendsignaalide muutumisega. Automaadi olek mingil hetkel sõltub sellel samal hetkel saabuvatest sisendsignaalidest ning automaadi eelnenud olekutest. Kuna olekute muutumise hetked pole fikseeritud, siis puudub asünkroonsetes automaatides taktiimpulsside generaator. Mäluga automaatide väljundsignaalid sõltuvad nii sisendsignaalidest kui ka olekutest. Niisuguste automaatide hulka kuuluvad paljud seadmed, alates lihtsatest funktsionaalsetest loogikalülitustest nagu trigerid, loendurid ja registrid kuni mikroprotsessorite ja arvutiteni välja. Juhul kui automaadil puudub mälu, on tal ainult üks sisemine olek ning automaati kirjeldab täielikult väljundfunktsioon. Mäluta diskreetseid seadmeid nimetatakse **loogilisteks kombinatsiooniskeemideks**. Praktikas on sellisteks skeemideks dekoodrid, kommutaatorid, summaatorid jms.

Võib luua ka automaadi, millel sisendid puuduvad või mille sisendsignaalid ei muutu. Niisuguseid automaate nimetatakse **autonoomseteks**, kõiki ülejäänuid aga mitteautonoomseteks. Autonoomsete automaatide töö on täielikult määratud nende sisemiste olekutega e programmiga ning ei sõltu informatsioonist väljaspool automaati.

Peale loetletud automaatide on olemas veel mitmed spetsialiseeritud diskreetsed automaadid; neist tähtsam on **mikroprogrammautomaat**.

Mikroprogrammautomaadi idee esitas M V Wilkes 1951. aastal. Ta võttis selle kasutusele universaalarvuti töö juhtimiseks. Kui tavaliselt arvuti ise juhib mingit protsessi, siis mikroprogrammautomaadi korral on tegemist automaadiga, mis juhib teist automaati, s. o arvutit ennast. Mikroprogrammjuhtimise põhimõtteid on rakendatud ka robotite ja tööpinkide juhtseadmetes.

Abstraktse automaadi matemaatilise kirjeldamise tulemuseks võivad olla: 1) algoritmide graaf- ja plokkskeemid, 2) siirde-, väljundi- ja olekutabelid, 3) loogikavõrrandid, 4) automaatide loogilise sünteesi algoritmikeelne kirjeldus või mõni teine esitusviis. Algoritmi praktiliseks teostamiseks saab kasutada kahte peamist võimalust: algoritmi realiseerimist **aparaadiks** või **programmiks**. Esimesel juhul on tulemuseks seade, teisel juhul programm. Seadme valmistamiseks kasutatakse mitmesuguseid pneumaatilisi, elektrilisi, optilisi või elektroonseid elemente. Sealjuures peab eelnevalt olema teada automaadi struktuur, mille aluseks on homogeenne või mittehomogeenne arvutuskeskkond (tehniline baas). Algoritmi programmilisel teostamisel tuleb koostada programm, mis on käskude jada, kusjuures need käsud määravad ära kõik juhtimiseks vajalikud operatsioonid, tehted lähteandmete ja vahetulemustega ning järgmise käsu aadressi. Programmi füüsiliseks kandjaks sobivad mitmesugused homogeensed struktuurid (mälud). Programmi töötlemiseks: käskude lugemiseks, dešifreerimiseks ja täitmiseks kasutatakse mingit universaalset diskreetset automaati, näiteks mikroprotsessorit. Programmi koostamist nimetatakse algoritmi **programmiliseks modelleerimiseks**, selle aparatuurset realiseerimist aga algoritmi **aparatuurseks modelleerimiseks**.

1.6.2. Algoritmide aparatuurne realiseerimine

Automaatide aparatuurne realiseerimine algab automaadi matemaatilisest kirjeldusest, näiteks siirde- ja väljunditabelitest. **Siirde-** ja **väljunditabelid** erinevad kombinatsiooniskeemide olekutabelitest selle poolest, et nad kajastavad ka automaadi olekute ajalist muutumist. Siirdetabelist 1.9 näeb, milline on automaadi olek järgmisel taktil, kui on teada automaadi senine olek x ja sisendsignaal u. Uus olek leitakse sisendsignaali u_i ja olekusignaali x_k põhjal tabeli 1.9 vastavate veergude ja tulpade ristumiskohalt. Väljunditabel 1.10 võimaldab määrata automaadi väljundsignaali y_j, kui on teada tema olek ja sisendsignaal.

Tabel 1.9

Automaadi siirdetabel

Xk / Ui	X0	X1	X2	X3	X4
U0	X1	X2		X3	X0
U1	X3	X0	X0	X4	
U2	X4		X3		X1

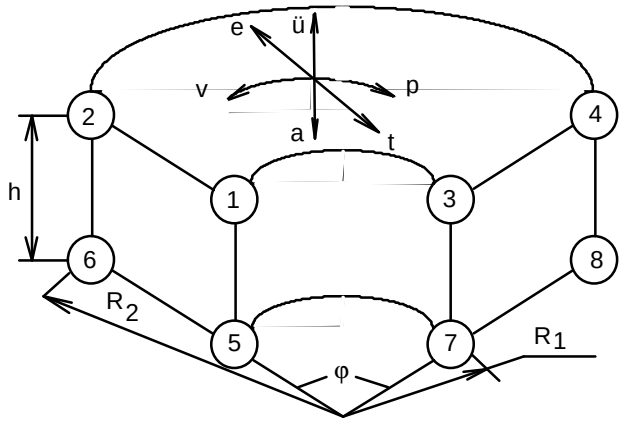
Tabel 1.10

Automaadi väljunditabel

Xk / Ui	X0	X1	X2	X3	X4
U0	Y0	Y5		Y3	Y4
U1	Y2	Y0	Y4	Y1	
U2	Y3		Y2		Y0

Struktuurne automaat erineb abstraktsest selle poolest, et tal on teatud kindel arv sisendeid ja väljundeid, milles toimivad etteantud viisil kodeeritud sisend- ja väljundsignaalid. Lisaks sellele jagab struktuurne süntees automaadi kaheks - mäluks ja kombinatsiooniskeemiks. Automaadi mälu koosneb mäluelementidest. Viimastena saab kasutada kahe sisendiga *D*-trigereid, neist koostatud registreid, etteantud mälumahuga püsi- või muutmälusid, ketasmälusid jms mäluseadmeid.

Algoritmide aparatuursest realiseerimisest parema ettekujutuse saamiseks vaadelgem struktuursete ja loogiliste automaatide sünteesi konkreetse näite varal. Selleks sobib tsüklilise positsioonjuhtimisega manipulaatori juhtautomaadi lihtsustatud mudel. Olgu tegemist silindrilises koordinaadistikus töötava manipulaatoriga, mida positsioonitakse teekonnalülitite abil. Niisuguste manipulaatorite töötsoon ja võimalikud liikumised on näidatud joonisel 1.30. Positsioonimispunktid on tähistatud ringide ja vastava numbriga.



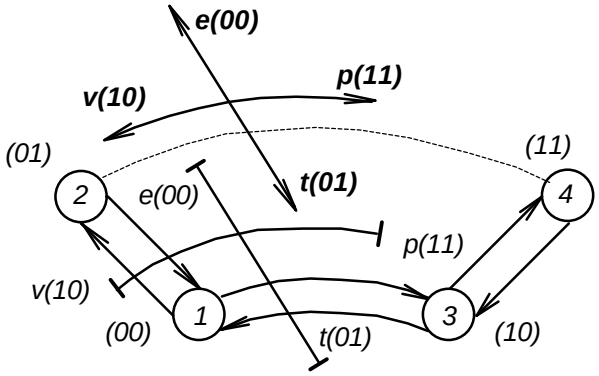
Joonis 1.30. Manipulaatori töötsoon, võimalikud liikumised ja positsioonimispunktid

Manipulaatori kolm lüli liiguvad kokku kuues suunas: edasi, tagasi, vasakule, paremale, üles ja alla, mida joonisel tähistavad nooled ja sõnade algustähed. Ajami liikumissuunda määrav käsk on juhtautomaadi väljundsignaaliks. Sisendsignaalideks on positsioonimispunktidesse paigutatud teekonnalülitite signaalid. Iga võimalikku liikumist piiravad kaks teekonnalülitit: püstsihis liikumist all ja üleval, pöördliikumist paremal ja vasakul ning radiaalsuunalist liikumumist ees ja taga asuvad lülitid. Seega on kokku kuus teekonnalülitit, mida tähistatakse tähtedega *ü* (ülemine lüliti), *a, v, p, e ja t*.

Manipulaatori positsioonimispunkte nimetatakse manipulaatori olekuteks, sest neile vastavad manipulaatori lülide eri asendid. Juhtseade peab võimaldama punktide vahel suvalist tsüklilist liikumist. Näiteks saab esemete teisaldamiseks kasutada tsüklit 1, 2, 6, 2, 1, 3, 4, 8, 4, 3 ning edasi jälle 1, 2, 6, ... jne. Lihtsuse huvides ei ole siin arvestatud manipulaatori seiskamise ja haaratsi juhtimise käske. Oletatakse, et esemete haaramine ja vabastamine toimub positsioonimispunktis liikumissuuna muutumisel ilma peatumispausita.

Reaalne juhtseade peab väljastama käske tehnoloogiaseadmetele ning võtma neilt vastu signaale. Seega on tegelikud juhtseadmed palju keerukamad kui siin vaadeldav juhtseade. Lihtsustused on vajalikud juhtautomaadi tööpõhimõtte paremaks mõistmiseks. Selguse huvides on otstarbekas näidet veelgi lihtsustada ning kirjeldada manipulaatori liikumist üksnes tasapinnal. Joonisel 1.31 on kasutatud samu tähiseid mis ruumilise liikumise mudeli juures. Lisaks on toodud väljundsignaalide, olekute ja sisendsignaalide koodid. Väljundsignaalide koodid on tähistatud rasvaselt, kusjuures vastavaid liikumisi näitavad nooled. Sisendeid tähistavad liikumise piirikud ning nende kõrval on näidatud koodid. Oletagem, et manipulaator töötab tsükliga 1, 2, 1, 3, 4, 3 ning edasi jälle 1, 2, 1, ... jne.

Manipulaatori liikumist kirjeldavat skeemi võib vaadelda kõigi eespool loetletud lihtsustuste korral ka juhtautomaadi graafskeemina, mis määrab roboti töö algoritmi. Graafskeemi järgi saab koostada juhtautomaadi siirde- ja väljunditabelid (tabelid 1.11 ja 1.12). Arusaadavuse huvides on tabelis nooltega näidatud roboti töötsükkel. Siirdetabelis esitatud automaadi olekud vastavad tema mälu olekutele. Kokku on tabelisse kantud nelja liiki signaalide koodid. Need on sisend- ja väljundsignaalide ning mälu sisendi ja väljundi ehk eelmise ja järgmise oleku koodid. Viimaseid tähistatakse tähtedega *D* ja *M* (joonis 1.32).



Joonis 1.31. Manipulaatori töötsükli graafskeem

Kõik koodid on kahekohalised. Nende nooremat ehk parempoolset kohta tähistab vastavalt u_0, y_0, D_0 või M_0; vanemat ehk vasakpoolset kohta aga u_1, y_1, D_1 või M_1. Tühjad lahtrid tabelis on tingitud sellest, et vaadeldav liikumine ei hõlma manipulaatori kõiki võimalikke liikumisi ning alati pole ka kõigile tabeli lahtritele vastavad tegevused võimalikud.

Tabel 1.11

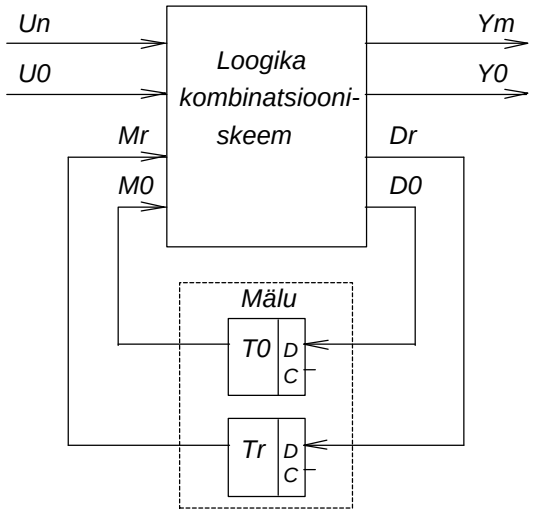
Automaadi siirdetabel

	X_{k+1}			
X_k (M) / u_i	M_1 M_0 00	01	10	11
u_1 u_0 00		00		10
01	10		00	
10	01 D_1 D_0			
11			11	

Tabel 1.12

Automaadi väljunditabel

	Y_j			
X_k (M) / u_i	M_1 M_0 00	01	10	11
u_1 u_0 00		00		10
01	10		00	
10	01 Y_1 Y_0			
11			11	



Joonis 1.32. Struktuurne automaat

Manipulaatori liikumise, s. o juhtautomaadi programmeerimiseks tuleb tema olekud salvestada mällu. Kuna olekut kirjeldab kahekohaline kahendarv, siis piisab mälust, mis koosneb kahest *D*-trigerist (joonis 1.32). Automaadi loogiline süntees seisneb tema loogika kombinatsiooniskeemi väljatöötamises. Selleks on vaja automaadi siirde- ja väljunditabelite põhjal koostada automaadi tööd kirjeldavad loogikafunktsioonid. Nende funktsioonide all mõeldakse väljundsignaale y_0, y_1, D_0 ja D_1, mis on sisendite u_0, u_1, M_0 ja M_1 funktsioonid. Püstitatud ülesande võib lahendada kahel viisil. Esiteks võib luua universaalse loogilise automaadi, mis peale ühenduste tegemist on võimeline sooritama ükskõik millist antud tabelis kirjeldatud tegevust. Ainsaks tingimuseks on, et signaalide arv ja tabelite mõõtmed ei muutuks. Sel juhul on tegemist automaadi aparatuurse liiasusega, mis võimaldab automaati vajaduse korral ümber häälestada. Teiseks võib koostada loogikaskeemi, mille korral automaat täidab ainult üht konkreetset tabelis näidatud tegevust. Sel juhul on tegemist eriotstarbelise automaadiga, mille tööd pole võimalik muuta.

Loogikafunktsioonide esitamiseks on otstarbekas siirde- ja väljunditabelist üle minna olekutabelitesse. Lihtsamal juhul võib need funktsioonid avaldada ka vahetult siirde- ja väljunditabelitest. Olekutabelisse koondatakse kõik sisendsignaalide ning neile vastavate väljundsignaalide väärtused. Tabelite 1.11 ja 1.12 põhjal koostatud automaadi olekutabelis (tabel 1.13) on näiteks neli sisendsignaali ja neli väljundsignaali. Sisendsignaalid võivad esinevad 16 kombinatsioonis, kuigi eespool toodud näite puhul oli tegemist vaid 6 kombinatsiooniga.

Tabel 1.13

Automaadi olekutabel

Sisendid				Väljundid			
U0	U1	M0	M1	D0	D1	Y0	Y1
0	0	0	0				
0	0	0	1				

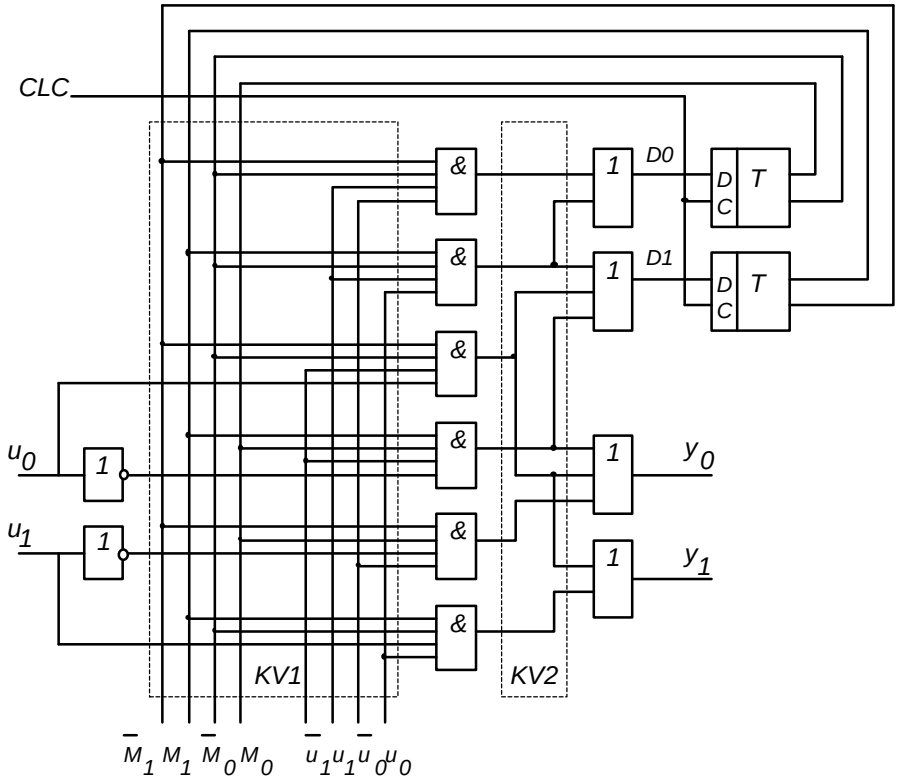
0	0	1	0	0	0	1	0
0	0	1	1	0	1	1	0
0	1	0	0	1	0	0	0
0	1	0	1				
0	1	1	0				
0	1	1	1				
1	0	0	0	0	1	1	1
1	0	0	1	0	0	0	1
1	0	1	0				
1	0	1	1				
1	1	0	0				
1	1	0	1	1	1	0	0
1	1	1	0				
1	1	1	1				

Olekutabeli 1.13 järgi kirjutatakse välja automaadi loogikavõrrandid, kusjuures arvesse võetakse ainult need sisendsignaalide kombinatsioonid, mille korral väljundsignaal võrdub ühega. Nii saadakse

$$
\begin{aligned}
D_0 &= \bar{u}_0 u_1 \overline{M}_0 \overline{M}_1 \vee u_0 u_1 \overline{M}_0 M_1, \\
D_1 &= \bar{u}_0 \bar{u}_1 M_0 M_1 \vee u_0 \bar{u}_1 \overline{M}_0 M_1 \vee u_0 u_1 \overline{M}_0 M_1, \\
y_0 &= \bar{u}_0 u_1 M_0 M_1 \vee \bar{u}_0 \bar{u}_1 M_0 M_1 \vee u_0 \bar{u}_1 \overline{M}_0 \overline{M}_1, \\
y_1 &= u_0 \bar{u}_1 \overline{M}_0 \overline{M}_1 \vee u_0 \bar{u}_1 \overline{M}_0 M_1.
\end{aligned}
\qquad (1.46)
$$

Võrrandite (1.46) põhjal saab konstrueerida juhtautomaadi loogikaskeemi. Et skeem kujuneks võimalikult lihtsaks, tuleb loogikafunktsioonid minimeerida. Võrranditest on näha, et eri funktsioonid sisaldavad ühesuguseid loogilisi osakorrutisi, mis lihtsustab oluliselt juhtautomaadi loogikat. Minimeerimisega seotud probleeme vaadeldi lähemalt punktis 1.2.3.

Nendega on võimalik tutvuda ka kirjanduse abil. Oluline on, et kui tahetakse luua roboti juhtseade, mis juhiks manipulaatori töötsooni ulatuses kõiki täiturseadme mehhanismide ja ajamite võimalikke liikumisi, siis peab juhtautomaat sisaldama vahendeid sisendsignaalide suvalistele kombinatsioonidele vastavate loogikafunktsioonide realiseerimiseks. Antud juhul peab niisugune automaat koosnema vähemalt 16 loogilisest NING-elemendist ja neljast VÕI-elemendist. Igal NING-elemendil olgu vähemalt 4 sisendit ja igal VÕI-elemendil kuni 8 sisendit. Lisaks vajatakse signaalide inverteerimiseks EI-elemente. Kui automaadiga soovitakse realiseerida ainult ühte algoritmi näiteks tabelitega 1.11 ... 1.13 esitatut, siis automaadi loogikaskeem lihtsustub ning piisab kuuest NING- ja neljast VÕI-elemendist. Sellise automaadi loogikaskeem on joonisel 1.33.



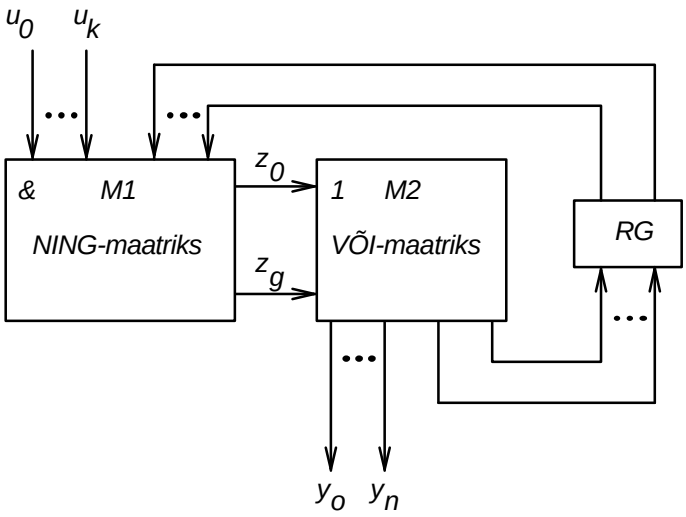
Joonis 1.33. Juhtautomaadi loogikaskeem

Automaadil on kaks sisendit ja kaks väljundit. Manipulaatori juures kasutatakse aga nelja täiturit, näiteks kontaktoreid, mis lülitavad ajameid liikuma edasi - tagasi või paremale ja vasakule. Kasutatakse ka nelja teekonnalülitit. Et ühendada juhtseade manipulaatoriga, tuleb sisendsignaalid kodeerida ning väljundsignaalid dekodeerida. Täiturite juhtimiseks on neid vaja võimendada. Kuna mäluelementideks on sünkroonsed *D*-trigerid, vajatakse automaadi tööks taktiimpulsse. Viimased saadakse taktiimpulsside generaatorist. Skeemil on kasutatud NING- ja VÕI-elemente. Kuna integraallülitustena toodetakse peamiselt NING-EI- ja VÕI-EI-elemente, siis võib automaadi tegelik skeem joonisel 1.33 esitatust veidi erineda.

Loogikaelementide sisendites olevaid kommutatsioonivälju *KV1* ja *KV2* saab valmistada ümberhäälestatavatena, kus ühendused tehakse näiteks pistikute abil. Ühendusskeemi muutmisel muutub juhtautomaadi töö algoritm. Sama otstarvet täidab ka ümberprogrammeeritav loogiline maatriks. Automaadi häälestamist võib tinglikult nimetada programmeerimiseks. Vaadeldud põhimõttel töötavad lihtsate tsükliliste positsioon-juhtimisega robotite ja muutumatu töötsükliga autooperaatorite juhtseadmed. Antud skeem on koostatud ühe konkreetse algoritmi jaoks.

Maatriksitel põhinevat loogikalülitust saab hõlpsasti muuta juhtautomaadiks, kui lisada mäluregister. Niisuguse juhtautomaadi struktuur on joonisel 1.34. Ka see, et juhtautomaate võib lülitada järjestikku, kinnitab maatriksitega juhtautomaatide universaalsust. Keeruline loogikafunktsioon realiseeritakse sel juhul mitmeastmeliselt,

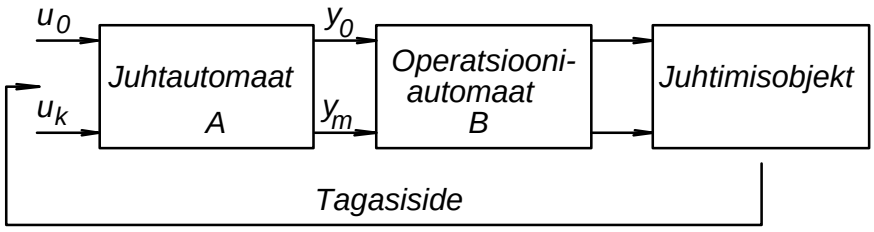
kasutades vahemuutujaid. Selliseid automaate kasutatakse sageli keerukate juhtseadmete sisemiste protsesside mikroprogrammjuhtimiseks.



Joonis 1.34. Programmeeritavate maatriksitega realiseeritud automaat

1.6.3. Programm- ja mikroprogrammjuhtimine

Süsteemide keerukuse teatud tasemel on juhtimise lihtsustamiseks otstarbekas rakendada hierarhilist struktuuri. Vastavalt sellele võib keeruka automaadi jagada **juht-** ja **operatsiooniautomaadiks**. Niisuguse automaadi struktuur koos juhtimisobjektiga on näidatud joonisel 1.35. Operatsiooniautomaat väljastab käsusignaale manipulaatorile ja tehnoloogiaseadmetele. Juhtautomaat korraldab operatsiooniautomaadi tööd. Arvutis on operatsiooniautomaadiks protsessor, juhtautomaadiks aga protsessori tööd juhtiv mikroprogrammautomaat. Operatsiooniautomaadi iseärasuseks on suur väljundsignaalide arv. Juhtautomaadil tuleb lahendada keerukaid loogikaülesandeid.



Joonis 1.35. Keerulise automaadi jagamine juht- ja operatsiooniautomaadiks

Robotsüsteem töötab kaheastmelise juhtimise seisukohalt järgmiselt. Süsteemi töötsükkel koosneb üksikutest tööoperatsioonidest, nagu manipulaatori lülide liikumine ühest positsioonimispunktist teise, detaili haaramine või vabastamine, tööpingi sisselülitamine jne. Iga taolise näiliselt lihtsa operatsiooni täitmiseks peab roboti juhtseade tegema rohkesti elementaartehteid: sisendsignaalidest sõltuvalt vastu võtma loogilisi otsuseid, leidma etteantud positsioonimispunkti ja tegeliku asendi erinevuse, moodustama juhtimis- ja indikatsioonisignaalid. Juhtseadme töös tähendab see mitmesuguste aritmeetika- ja loogikaülesannete lahendamist ning seadmete (aritmeetika-loogikaploki registrite) üksikasjalikku juhtimist. Seega tuleb roboti iga tööoperatsiooni jaoks täita juhtimisalgoritm, mis sisaldab sadu elementaartehteid. Roboti juhtimine ja programmeerimine elementaartehete kaupa on äärmiselt ebamugav ning aeganõudev. Peale selle vajatakse väga kõrge kvalifikatsiooniga spetsialiste, kes tunnevad põhjalikult seadme ehitust. Hoopis mugavam on koostada robotsüsteemi töö algoritm manipulaatori ja tehnoloogiaseadmete üksikute tööoperatsioonide kaupa. Sel juhul peavad aga juhtseadmetes olema realiseeritud kõigi võimalike töö- e väljundoperatsioonide algoritmid. Juhtimine taandub siis vajalike algoritmide valikule ning nende järjestuse määramisele. Neid algoritme on võimalik realiseerida nn jäiga loogikaga, s.o vastavate loogikaskeemide abil, või mällu salvestatud e programmeeritava loogika abil. Viimasel juhul nimetatakse igale väljundoperatsioonile vastavat alamprogrammi mikroprogrammiks ja igale sisemisele elementaaroperatsioonile vastavat tehet mikrokäsuks. Mikroprogramme realiseerivat juhtseadme osa nimetatakse aga mikroprogrammjuhtimisega automaadiks. Seega on mikroprogrammjuhtimine üldise programmjuhtimise (mida nimetatakse vahel ka makroprogrammjuhtimiseks) alumiseks tasandiks. Kui makroprogrammid koostab peamiselt robotsüsteemi kasutaja, siis mikroprogrammid koostab enamasti seadet valmistav tehas ning need kuuluvad seadme riistvara (püsivara) juurde. Analoogiliselt on jaotatud programmjuhtimise ülesanded arvutis. Kasutajaprogramme töötleb protsessor, mille tööd korraldab juhtautomaat. Kui

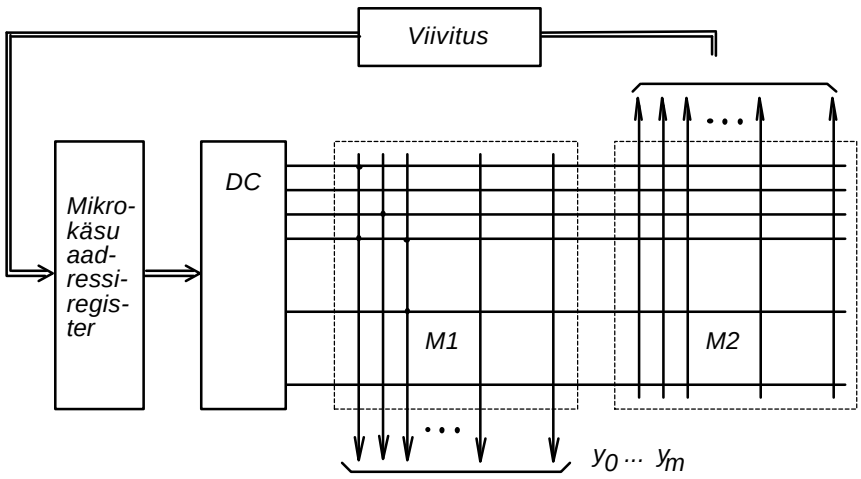
tehnoloogilist protsessi või robotsüsteemi juhib arvuti või mitmeraalijuhtseade, siis võib programmjuhtimise hierarhiaastmete arv olla 3 ja enamgi.

Mikroprogrammjuhtimise peamised eelised on järgmised.

1. Paindlikkus, s.t võime valida programmeerimiseks mitmesuguseid makrokäsustikke; võime imiteerida teisi juhtseadmeid ning kasutada nende tarkvara.
2. Projekteerimise, valmistamise ja kasutamise lihtsus, mis tuleneb struktuuri homogeensusest ning hierarhilise programmjuhtimise põhimõttest.

Loetletud eelistel on reaalselt olemas piirid, mille ulatuses neid saab kasutada. Väga lihtsate süsteemide korral võib aparatuurselt realiseeritud juhtseade osutuda efektiivsemaks kui mikroprogrammidel põhinev seade.

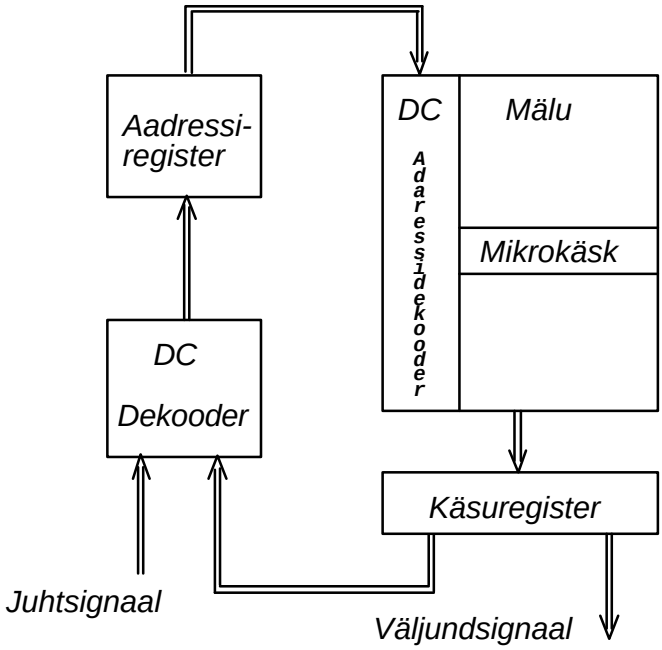
Mikroprogrammjuhtimisel põhinevaid automaate kasutati kõigepealt elektronarvuti töö juhtimiseks. Wilkesi loodud mikroprogrammjuhtimisega automaat põhines loogilistel maatriksitel, registritel ja dekoodril e dešifraatoril (joonis 1.36). Igal töötaktil peab automaat moodustama uue mikrokäsu aadressi ning väljastama täidetavale mikrokäsule vastava juhtsõna. Uue mikrokäsu aadress moodustatakse maatriksis *M2* ning see edastatakse teatud viivitusega mikrokäsu aadressiregistrisse. Sünkroonsignaali saabumisel aadress dekodeeritakse dešifraatoris *DC*, mis ergastab käsule vastava rõhtjuhtme (joonis 1.36) maatriksites *Ml* ja *M2*. Seega vastab igale rõhtjuhtmele üks mikrokäsk, nende koguarv on aga määratud mikrokäsu kahendaadressi kohtade arvuga *n* ja võrdub 2^n. Käsule vastavalt ergastatakse punktidega märgistatud kohtades maatriksi püstjuhtmed ning moodustatakse juhtsõna y_0 ... y_m. Maatriksis *M2* moodustatakse aga uue mikrokäsu aadress. See salvestatakse jällegi aadressiregistrisse. Uue mikrokäsu täitmine algab järgmisel taktil pärast seda, kui saabub uus sünkroimpulss. Maatriksite *Ml* ja *M2* programmeerimine võimaldab moodustada vajalikke mikrokäske ning väljastada neid algoritmile vastavas järjekorras, kusjuures saab korduvalt tagasi pöörduda ühe või teise käsu juurde. Niisugune mikrokäskude jada moodustabki mikroprogrammi.



Joonis 1.36. Wilkesi mikroprogrammautomaat

Küllalt suurte maatriksite korral täidab automaat mitmeid mikroprogramme, mis kõik koosnevad kindlast käsustikust valitud mikrokäskudest, kuid mille järjestus mikroprogrammides on erinev. Igale makrokäsule vastava mikroprogrammi täitmine algab sel juhul kindlast mikrokäsu aadressist ning jätkub kuni mikroprogrammi lõpuni automaatselt. Iga järgmine makrokäsk käivitab uue mikroprogrammi.

Mikroprogramme saab salvestada ka pooljuhtmällu. Mälul põhineva mikroprogramm-automaadi struktuur on näidatud joonisel 1.37. Mikrokäsu aadress salvestatakse aadressi-registrisse. Aadressi dešifreerimisel leitakse mälus sellele vastav mikrokäsk, mis sisaldab järgmise mikrokäsu aadressikoodi ja antud mikrokäsule vastava mikrooperatsiooni koodi. Mikrokäsk loetakse mälust käsuregistrisse. Sealt edasi läheb aadressikood dešifraatorisse *DC*. Aadressikoodi ja väljast saabuva juhtkäsu alusel moodustatakse dešifraatoris sünkroimpulsi saabumisel järgmise mikrokäsu aadress, mis salvestatakse aadressiregistrisse. Käsuregistrist väljastatakse seadme juhtimiseks vajalik mikrooperatsiooni kood. Kuigi mikroprogrammautomaate realiseeritakse nii mälude kui ka programmeeritavate loogiliste maatriksite baasil, on eri automaatidel veidi isesugused omadused. Mälumassiivi dešifraator on jäiga loogikaga, *PLM*- maatriks *Ml* aga paindliku loogikaga. *PLM* realiseerib loogikafunktsiooni $y_i = f(x_i)$ disjunktiivset normaalkuju, mälu aga selle funktsiooni tõeväärtustabelit. Sellest tingituna on mäludega lihtsam realiseerida lihtsa loogika ja suure väljundite arvuga funktsioone. *PLM*-i on otstarbekas kasutada seal, kus on tegemist suhteliselt keerukate loogikafunktsioonidega, ning juhul kui väljundite arv ei ole väga suur.



Joonis 1.37. Mälul põhinev mikroprogrammautomaat

1.6.4. Algoritmide programmiline realiseerimine

Juhtseadmete riistvara ja tarkvara duaalsusest (ühildatavusest ja asendatavusest) tuleneb, et juhtalgoritme saab realiseerida nii aparatuursete (riistvara) kui ka programmiliste (tarkvara) vahenditega. Esimesel juhul on realisatsiooni tulemuseks **aparatuurne automaat**, teisel juhul **programmiautomaat**.

Programmiautomaat kujutab endast mõnes algoritmikeeles esitatud programmi, mis on ette nähtud protsessi või seadme juhtimiseks. Programmi salvestamiseks ja töötlemiseks läheb aga vaja universaalset riistvaraautomaati. Tänapäeval on niisugusteks automaatideks mikroprotsessoritel põhinevad üldotstarbelised arvutid, raalid ja raalisüsteemid. Juhtraalide mälu maht, töökiirus, sisendite ja väljundite arv, väline kuju ning hind on väga erinevad. Kasutatakse nii odavaid binaarsüsteemide juhtimiseks mõeldud programmeeritavaid kontrollereid kui ka keerukaid, hierarhilise või hajusstruktuuriga mitmeraalijuhtseadmeid. Vaatamata mikroprotsessorsüsteemide mitmekesisusele on eri tüüpi protsessorite ehituses ja tööpõhimõttes palju sarnast.

2. MIKROPROTSESSORID

2.1. Mikroprotsessorite ja -arvutite ehitus

2.1.1. Põhimõisted

Arvutiks või **raaliks** nimetatakse universaalset programmjuhtimisega infotöötlusseadet, millega sooritatakse kõiki tuntud aritmeetika- ja loogikatehteid. Nimetus arvuti on õigustatud juhul, kui põhiliseks infotöötluse viisiks on arvutustehted või tekstid. Teistsuguse infotöötluse korral on seadet otstarbekas nimetada raaliks, eristamaks juhtimisülesannete lahendamiseks mõeldud eriseadmeid **bürooarvutitest**. Universaalset juhtseadet nimetatakse **juhtraaliks**.

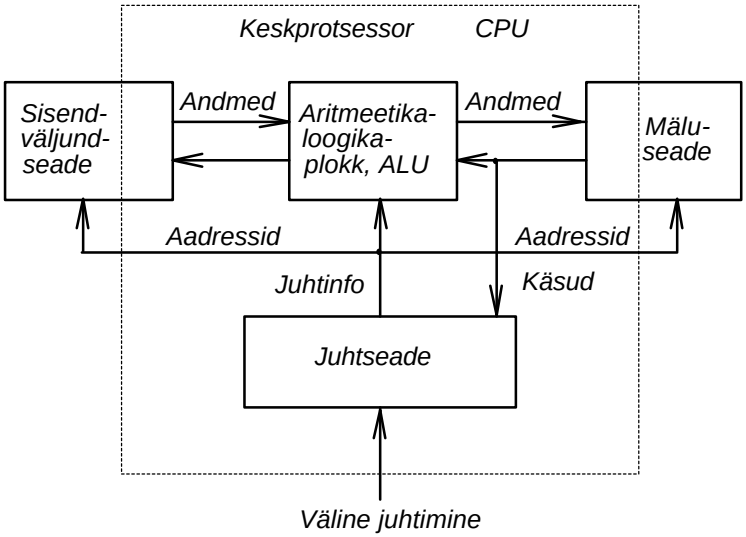
Eespool vaadeldud **mikroprogrammautomaat** võimaldab protsesside tsüklilist juhtimist, kuid seda on väga tülikas rakendada universaalse infotöötlusseadmena eriti siis, kui tuleb sooritada aritmeetika- ja loogikatehteid. Nendeks operatsioonideks on ette nähtud aritmeetika-loogikaplokk (*ALU - arithmetic-logic unit*). **Aritmeetika-loogikaploki** põhifunktsioonideks on mitmekohaliste kahendarvude summeerimine, nende nihutamine vasakule või paremale, loogiline eitus ehk inversioon, loogiline liitmine (disjunktsioon), loogiline korrutamine (konjunktsioon) ning loogiline alternatiiv ehk VÄLISTAV VÕI. Nende põhifunktsioonide kombineerimisega ning rakendamisega kindlas järjekorras sooritatakse kõiki tuntud aritmeetika- ja loogikatehteid. Näiteks toimub kahendarvude korrutamine järjestikuste summeerimis- ja nihkeoperatsioonide abil. Elementaartehete sooritamise järjekord on määratud arvutuste (näiteks korrutamise) algoritmiga, mida täidetakse vastavalt mällu salvestatud programmile. Seejuures juhitakse arvutusprotsessi ehk aritmeetika-loogikaploki, mälu ja registrite tööd mikroprogrammautomaadi abil.

Protsessoriks nimetatakse funktsionaalselt terviklikku, aritmeetika- ja loogika-operatsioonideks ette nähtud seadet, mis sooritab tehteid mälus paiknevate käskude järgi. Peale aritmeetika-loogikaploki kuulub protsessori koosseisu mitu registrit ning juhtautomaat näiteks mikroprogrammautomaat. Registrid on ette nähtud operandide ja vahetulemite ajutiseks salvestamiseks. Juhtautomaat korraldab aritmeetika-loogikaploki ja registrite tööd mällu salvestatud programmi kohaselt.

Mikroprotsessoriks nimetatakse ühel või mitmel integraallülitusel ehk kiibil (*chip*) asuvat protsessorit. Ühel kiibil asuvat mikroprotsessorit nimetatakse ka **monoliitprotsessoriks**. Juhul kui protsessor koosneb mitmest kiibist ja igal kiibil olev osaprotsessor töötleb kahendsõna teatud kohti, on tegemist **silpprotsessoriga**. Arvuti mäluseade koostatakse tavaliselt eraldi integraallülitustest. Mõnikord on aga mälu paigutatud protsessoriga ühele kiibile ning nad moodustavad koos sisendite ja väljunditega ühekiibiarvuti. **Ühekiibiarvuteid** kasutatakse peamiselt lokaalseks juhtimiseks, eriti seal kus on oluline seadme kompaktsus ja kus ei vajata suuremahulist mälu.

Arvuti või **juhtraal** (juhtarvuti) koosneb protsessorist, mälust, informatsiooni sisestamiseks ja väljastamiseks mõeldud liidestest ning sisend- väljundseadmetest. Arvuti plokkskeem ehk nn J von Neumanni klassikalise digitaalarvuti struktuur on kujutatud joonisel 2.1, kus **keskprotsessor** (*CPU - central processor unit*) on eraldatud punktiiriga. Arvuti põhiosade funktsioonid ja koostöö on korraldatud järgmiselt:

1. *ALU* sooritab aritmeetika-loogikatehteid.
2. Mäluseadmes salvestatakse tööks vajalik info (andmed, tulemused ja programmid).
3. Sisend-väljundseade on lähteandmete sisestamiseks ja tulemuste väljastamiseks. Seade sisaldab puhverregistreid ning loogika- ja muid lülitusi välisseadmetega ühendamiseks.
4. Juhtseade, mis võib olla realiseeritud mikroprogrammiautomaadina, juhib aritmeetika-loogikaploki tööd, annab juhtsignaale mälu- ja sisend-väljundseadmetele. Juhtseade korraldab juhtkäskude järgi andmete sisestamist, tehete sooritamist *ALU*-s, andme-vahetust *ALU* ja mäluseadme vahel, andmete väljastust jne.



Joonis 2.1. Digitaalarvuti klassikaline J von Neumanni struktuur

Nüüdisaegse arvutiteooria rajaja J von Neumanni loodud arvutistruktuuri peamiseks iseärasuseks on see, et programmi käsud ja arvutusteks vajalikud andmed paiknevad ühises mälus ning protsessor kasutab neid ühel ja samal viisil. Sõltuvalt programmist võib arvuti ise muuta käskude täitmise järjekorda. **Programm** on mingi tegevuse formaliseeritud eeskiri. Programm koosneb üksikutest instruktsioonidest, mida nimetatakse **käskudeks**. Programmi täidetakse arvutis üksikute käskude kaupa. Keerukamad programmid jagunevad **alamprogrammideks**, mida arvuti võib ühe programmi jooksul täita korduvalt.

2.1.2. Arvuti põhiplokid ja siinid

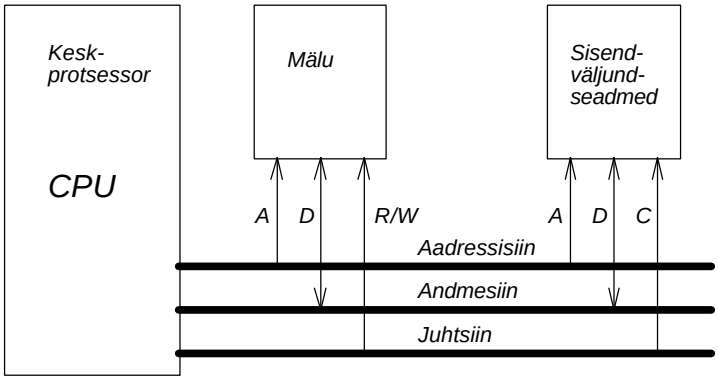
Mikroprotsessorite ja arvutite ehitus sõltub sellest, kuidas nende eri osad: *ALU*, registrid, mälu, sisend-väljundliidesed jms on ühendatud tervikuks. Juhtseadme protsessori, mälu ja sisend-väljundliideste vahel kasutatakse ühenduseks **siine** (*bus*). Sõna *bus* tähendab inglise keeles mitmejuhilist ühendust, millega saab omavahel liita palju süsteemi komponente. Juhtseadme siin koosneb mitmest paralleelsest juhist, mis ühendavad elektriliselt juhtseadme erinevaid osi. Siinid jagunevad aadressi-, andme- ja juhtsiinideks. Aadressi- ja andmesiinid on tavaliselt 8- või 16-soonelised, nende kaudu edastatakse korraga ühe- või kahebaidine sõna.

Arvutites kasutatakse nii kolmesiinilist aadressi- (*A - address*), andme- (*D - data*) ja juhtsiiniga (*C - control*) kui ka kahesiinilist, s. o ühise aadressi ja andmesiiniga ning eraldi juhtsiiniga süsteemi. Vastavad struktuurid on joonistel 2.2 ja 2.3.

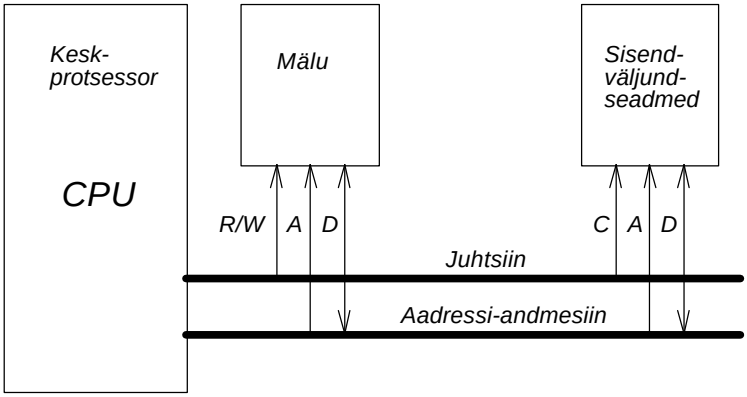
Aadressisiini 8 biti abil saab edastada aadresse 0...255, mis sobib väga väikese mälu või näiteks sisend- ja väljundliideste adresseerimiseks. Kõik sisend- ja väljundliidesed ning mälu on ühendatud siiniga, millele protsessor väljastab aadressi. Iga liides või mälu reageerib kohe, kui ta oma aadressi ära tunneb, s. t kui protsessor siini kaudu tema poole pöördub; muud liidesed sel juhul ei reageeri. Suuremate mälude adresseerimiseks on vaja 16- või enamasoonelist siini. Otseselt adresseeritavate mälupesade arv võrdub muutmälu pesade (baitide) arvuga ehk mälu mahuga. 16-bitise aadressisiini korral saab otseselt adresseerida 2^{16} = 65535 baiti = 64 Kbaiti; (2^{20} = 1 Mbait).

Kui mingi sisendseade tuvastab siinil oma aadressi, väljastab ta **andmesiinile** oma mäluregistri bittide olekud. Kui protsessor saadab andmesiinile väljastamiseks kaheksa bitti, antakse need bitid väljundliidese kaudu edasi väljundseadmetele. Samuti toimub andmesiini kaudu andmevahetus protsessori ja mälu vahel. **Juhtsiini** kaudu edastatakse signaale, mida kasutatakse arvuti töö juhtimiseks ja kontrolliks. Näiteks määravad juhtsiini kaudu edastatavad signaalid *R* (*read*) ja *WR* (*write*), kas mälu poole pöördutakse info lugemiseks või kirjutamiseks.

Aadressid ja andmed on siinil väga lühikest aega. Nii saab ühe ja sama siiniga edastada kogu nõutava info paljudelt sisenditelt protsessorisse ja vastupidi protsessorist paljudesse väljunditesse, lugeda mälust käske ning salvestada sinna vajalikku infot.

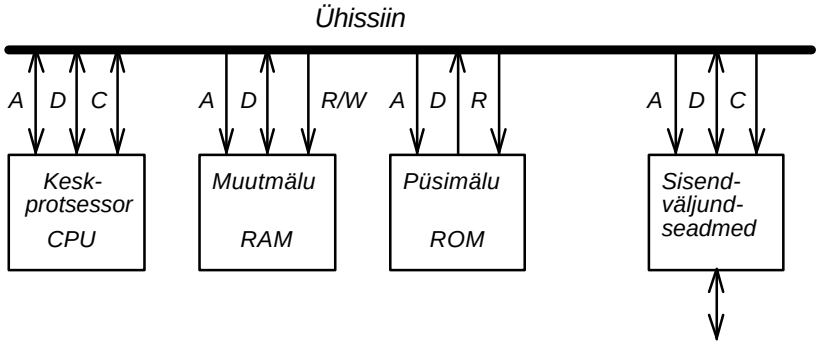


Joonis 2.2. Kolmesiinilise mikroarvuti struktuur



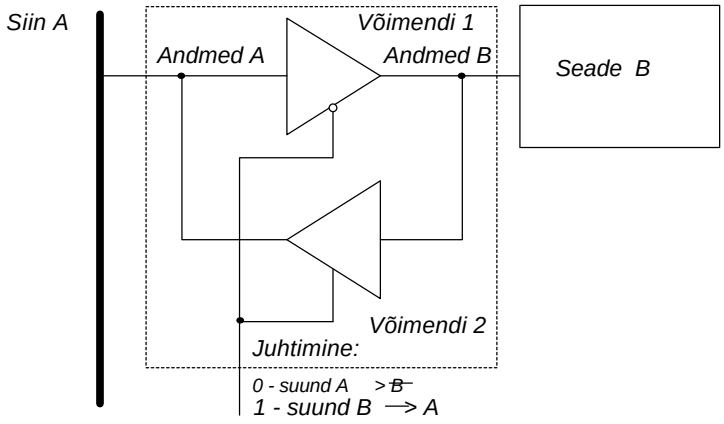
Joonis 2.3. Kahesiinilise mikroarvuti struktuur

Kaasaegsed arvutid ja juhtimissüsteemid põhinevad magistraalstruktuuril, mille universaalset **ühissiini** nimetatakse ka universaalseks kanaliks (joonis 2.4).



Joonis 2.4. Ühissiiniga arvuti struktuur

Siinis edastatakse andmeid mõlemas suunas, s. t signaalide vastuvõtt ja väljastamine toimub samu juhtmeid pidi. Selleks kasutatakse kolmeolekulisi kahesuunalisi puhvreid. Joonisel 2.5 on reversiivne andmete edastus siinilt *A* seadmele *B* kahesuunalise puhvri abil.



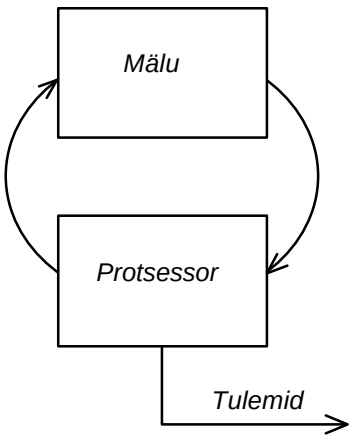
Joonis 2.5. Kahesuunaline värat

Juhtsignaali 0 korral edastab võimendi *DA1* andmeid siinilt *A* seadmele *B.* Juhtsignaali 1 korral muutub aktiivseks võimendi *DA2* ning andmevoog kulgeb seadmelt *B* siinile *A.* Samal ajal kui üks võimenditest on aktiivne, on teine võimendi suure sisetakistusega olekus. Seda olekut nimetatakse suletud ehk kolmandaks olekuks, mis tähendab, et see ei võrdu 1 ega 0-ga.

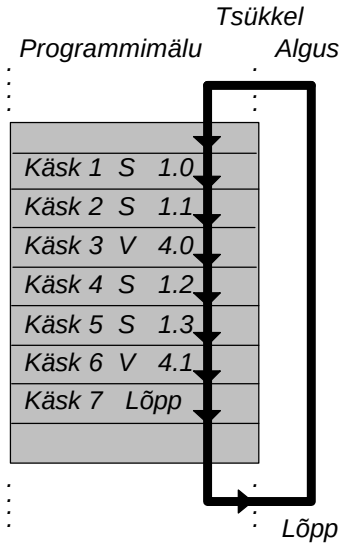
2.1.3. Töötsüklid

Arvuti töötab mällu salvestatud programmi järgi. Protsessor loeb mälust programmi käske ning täidab neid käskude sisu alusel. Programmi täitmine toimub tsükliliselt käskude kaupa. Käsu täitmiseks peab protsessor pöörduma mälu poole, lugema sealt käsukoodi, dekodeerima selle, võtma vastu käsu sisule vastavad loogilised otsused, väljastama juhtsignaalid kõigile arvuti komponentidele, leidma uue käsu aadressi ning salvestama selle aadressiregistrisse. Järgmise käsu täitmisel kordub kõik enam-vähem samas järjekorras. Erinevused käskude täitmisel on tingitud nende erinevast sisust. Ühe käsu täitmiseks kuluvat ajavahemikku nimetatakse **käsutsükliks**. Programmi täitmine ning protsessori ja mälu vaheline töö on korraldatud tsükliliselt, kusjuures töö tulemusi väljastatakse perioodiliselt (joonis 2.6). Programmi käsud on mälus järjestikku, nii nagu need on kirjutatud ka käsitsi koostatud programmilistile. Iga käsk vajab mälus ruumi, mis sõltub käsu baitide arvust.

Arvuti töö on sünkroniseeritud taktiimpulssidega. Taktiimpulsid saadakse impulsigeneraatorist. Töö stabiilsuse suurendamiseks ning reaalajas töötamiseks rakendatakse impulsigeneraatoris kvartsresonaatorit. Tänu sellele suudab arvuti täita ka kella funktsioone.

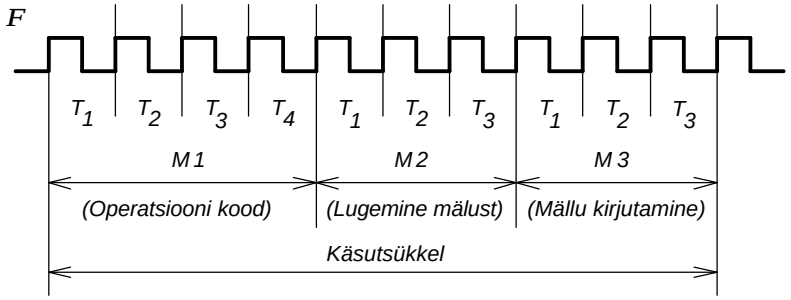


Joonis 2.6. Arvuti töötsükkel



Joonis 2.7. Käskude paiknemine mälus

Takt (*T*) on lühim ajavahemik, mille jooksul arvuti suudab sooritada elementaartehte. Kestuselt järgmiseks ajavahemikuks on arvutis masinatsükkel (*M*). **Masinatsükli** jooksul võib arvuti valida mälust käsusõna, lugeda mälust või kirjutada infosõna mällu, lugeda andmesõna sisendist või salvestada tulemisõna väljundisse, katkestada põhiprogrammi täitmine või teha mingi muu operatsioon. Eri liiki masinatsüklite arv on määratud protsessori juhtalgoritmiga. Mikroprotsessoril *Intel* 8080 on 10 erinevat masinatsüklit. Masinatsükliteks on näiteks käsu valik, mälu lugemine, mällu kirjutamine, pinumälu lugemine, pinumällu kirjutamine, sisendi lugemine, väljundisse kirjutamine, katkestuse lubamine, otsemällupöördumise lubamine jms. Ühe masinatsükli kestus on sõltuvalt sisust 3...5 takti. Käsu täitmiseks kulub 1...3 masinatsüklit. Käsutsükli ajadiagrammi näide on joonisel 2.8. Vastavalt käsule kirjutatakse operand ühest mälu piirkonnast teise. Esimese masinatsükli *M1* ajal määratakse operatsiooni- ehk tehtekood, s. t mida teha. Teise masinatsükli *M2* ajal loetakse operand mälust protsessorisse. Kolmanda masinatsükli *M3* ajal kirjutatakse operand muutmälu pessa, mille aadress on protsessori registris. Programmi käsud asuvad järjestikustes mälupesades. Seepärast tuleb järgmise käsu poole pöördumiseks suurendada aadressi vastavalt käsu pikkusele ühe, kahe või kolme võrra.



Joonis. 2.8. Keskprotsessori käsutsükli ajadiagramm

Arvutit, mille põhiülesandeks on seadmete või protsesside juhtimine, nimetatakse juhtraaliks. Juhul kui juhtraali protsessor käib korduvalt töötsükli kaupa kõik programmimällu kirjutatud käsud algusest lõpuni läbi, s. t pöördub tsükliliselt vastava aadressiga mälupesade poole, on tegemist **tsüklilise programmjuhtimisega**. Programmi lugemist algusest lõpuni nimetatakse **programmi tsükliks**. Aega, mis kulub programmitsükli läbimiseks, nimetatakse **programmi tsükliajaks**. Selle kestus sõltub juhtraali töökiirusest ning juhtkäskude olemusest. Lihtsamate tsüklilise programmjuhtimisega juhtraalide ehk programmeeritavate kontrollerite tsükliaeg koosneb kolmest üksteisele järgnevast osast:

1. Protsessor pärib kõigi sisendite olekuid ning salvestab need vahemällu. Vastav päringuprogramm on salvestatud kontrolleri juhtimissüsteemi.
2. Protsessor töötleb kasutajaprogrammi. Ta loeb käske; täidab neid selles järjekorras, nagu need on programmimälus; pärib sisendite, ajareleede jms olekuid ning moodustab lõpptulemi, millega juhitakse väljundeid.
3. Protsessor saadab väljundite uued olekud väljundikaardile.

Kokkuvõtteks võib öelda, et arvuti töö on korraldatud tsükliliselt, kusjuures tsüklid moodustavad hierarhia: programmitsükkel, käsutsükkel, masinatsükkel, takt.

2.2. Mikroprotsessori tööpõhimõte

2.2.1. Protsessori ehitus

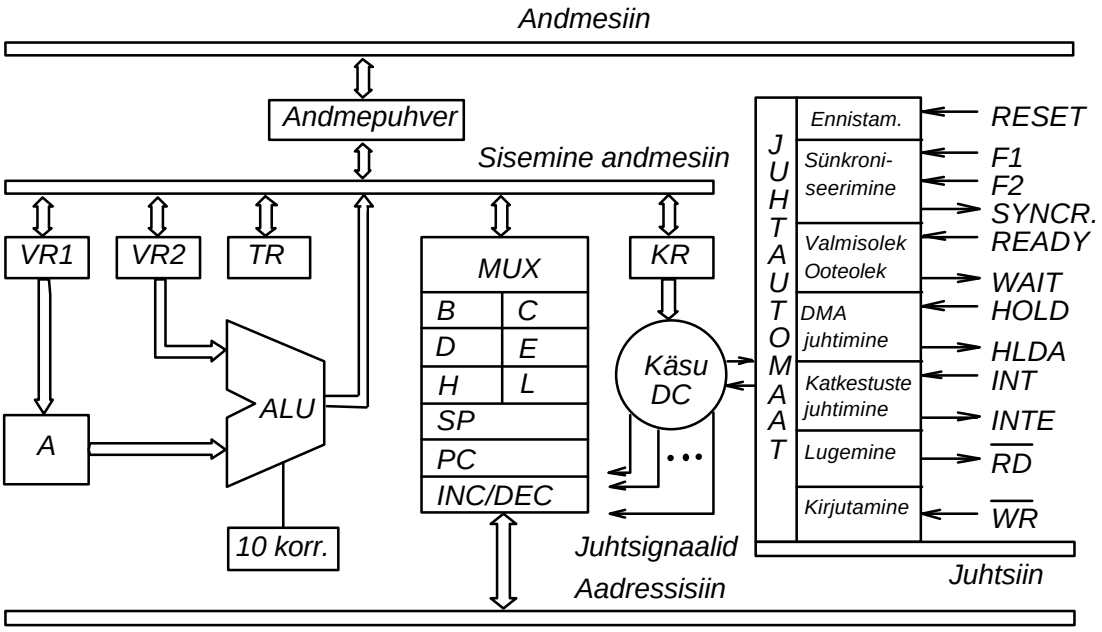
Mikroarvutite ja juhtraalide kasutaja (programmeerija) ei pea teadma riistvara üksikasjalikku ehitust, kuid tal peab olema selge ettekujutus protsessori ja arvuti tööpõhimõttest. Ta peab teadma arvuti põhilisi koostisosi ning signaale, koode või käske, millega arvuti tööd juhitakse. Kõige olulisem on tunda mikroarvuti käsustikku, sest üksikutest käskudest koostatakse kõik kasutajale vajalikud programmid.

Mikroarvuti käsustik sõltub protsessori tüübist. On olemas **fikseeritud käsustikuga protsessorid** ning ka sellised, mille käsustiku võib kasutaja oma soovi kohaselt välja mõelda. See tähendab, et mikroprotsessori seesmine juhtautomaat on kasutaja poolt programmeeritav või ümberprogrammeeritav. Teisiti öeldes, on olemas protsessorid, mille seesmise juhtautomaadi programmeerib kiibi valmistaja (tehas), ning on olemas protsessorid, mille juhtautomaadi mikroprogrammid salvestab kasutaja. Viimaste hulka kuuluvad tavaliselt silpprotsessorid.

Protsessorid eristuvad üksteisest töödeldava kahendsõna pikkuselt (8-, 16- ja 32-bitised), sisemiste registrite arvult ja tähenduselt, info adresseerimise viisi ning käskude sümboolika ja koodide poolest. Seepärast ei ole ühe mikroarvuti masinakeeles programmid otseselt ülekantavad teisele arvutile. Ometi on eri mikroarvutite ehituses ja programmeerimises palju sarnasusi, mis võimaldavad ühe arvuti juures omandatud kogemusi rakendada ka teiste arvutitega töötamisel.

Programmeerija jaoks tuleks koostada juhtraali või programmeeritava kontrolleri mudel, kus on ära näidatud kõik programmeerijale ligipääsetavad riistvara osad. Nendeks on aritmeetika-loogikaplokk, akumulaator, üldotstarbelised ehk üldregistrid ning mälu. Programmeerija peab tundma mälu aadressiruumi, millistesse pesadesse saab kirjutada programme, kus asuvad juhtseadme süsteemsed programmid ning millised aadressid on reserveeritud andmete salvestamiseks.

Joonisel 2.9 on näidatud kolme siiniga 8-bitise mikroprotsessori (*Intel* 8080) ehitus ning põhilised komponendid. Mälu aadresside ja käskude kodeerimiseks kasutatakse antud juhul kuueteistkümnendarve. Märkigem, et mõne teist tüüpi protsessori korral kasutatakse ka kaheksandarve. Programmistile on kättesaadavad 10 registrit, neist kuus 8-bitist üldregistrit, 8-bitine akumulaator ja tunnuste register ning 16-bitised registrid *SP* ja *PC*. Üldotstarbelised 8-bitised registrid on tähistatud tähtedega *B, C, D, E, H, L*. Neid võib kasutada ka paariviisi *B & C, D & E ja H & L,* s. t kolme 16-bitise registrina. Registripaare tähistatakse paari esimese registri tähise järgi, s. t *B&C = B, D&E = D ja H&L = H. A* tähistab akumulaatorit, *SP* pinumälu viita ja *PC* käsuloendurit. *INC/DEC* on kasvatamis-/kahandamislülitus (*increment/decrement*).



Joonis 2.9. Mikroprotsessori registrid ja siinid

2.2.2. Registrid ja nende otstarve

Mikroprotsessor sisaldab mitmeid registreid, mida kasutatakse tehte tulemite või tehte vahetulemite lühiajaliseks salvestamiseks, selleks et tulemid oleksid kiiresti saadaval järgmisteks teheteks.

Akumulaator on protsessori üheks kõige tähtsamaks registriks, kuhu enne tehte sooritamist viiakse üks operandidest ning kuhu salvestatakse automaatselt aritmeetika-loogikaploki tulem.

Pinumälu viit ehk **pinuviit** säilitab muutmälu selle piirkonna aadressi, mida jooksvalt kasutatakse pinumäluna.

Käsuloenduri (*PC - program counter* või *IP - instruction pointer*) ülesandeks on säilitada programmi järgmise käsu aadressi. Selle järgi võib käsuloendurit nimetada ka käsuviidaks (vrd pinuviit) või programmiloenduriks. Standardsel 8-bitisel mikroprotsessoril on 16-bitine käsuloendur, millega täidetakse programmi, mille pikkus mälus ei ületa 65 535 käsku.

Juurdekasvulülitus võimaldab suurendada või vähendada programmiloenduri sisu ühe võrra ning programmeerida selle abil loendureid ja korraldada järjestikku protsesside juhtimist.

Üldregistreid *B, C, D, E, H, L* kasutatakse programmi operandide, vahetulemite või aadresside ajutiseks säilitamiseks.

Peale nime on igal registril veel kindel kolmebitine kahendkood, mis sisaldub tehtekoodis ja võimaldab käsuga vahetult vastava registri poole pöörduda.

B 000, *C* - 001,
D 010, *E* - 011,

H 100, *L* - 101,
M 110, *A* - 111,

kus *M* tähistab mälupesa, mille aadressiks on registripaaris *H&L* olev kood.

Registrite kasutamisel tuleb arvestada, et andmesõna normaalseks pikkuseks on 8 bitti, aadressisõna pikkuseks aga 16 bitti. Vastavalt sellele salvestatakse andmeid harilikult 8-bitistesse registritesse ning üksnes erijuhul kasutatakse topeltpikkusega 16-bitist andmesõna. Registrite arv ning otstarve on eri protsessoritel erinev. Näitena on tabelis 2.1 protsessoriperekonna *Intel* 8086 ja 80386 registrite loetelu.

Tabel 2.1

Protsessoriperekonna *Intel* 8086, 80386 registrid

Üldregistrid (*General Purpose Register*)	**Segmendiregistrid** (*Segment Registers*)
AH/AL AX (EAX) Accumulator	CS Code Segment
BH/BL BX (EBX) Base	DS Data Segment
CH/CL CX (ECX) Counter	SS Stack Segment
DH/DL DX (EDX) Data	ES Extra Segment
	(FS) 386 and newer
(Exx) indicates 386+ 32 bit register	(GS) 386 and newer
Viidaregistrid (*Pointer Registers*)	**Pinumälu registrid** (*Stack Registers*)
SI (ESI) Source Index	SP (ESP) Stack Pointer
DI (EDI) Destination Index	BP (EBP) Base Pointer
IP Instruction Pointer	
Olekuregistrid (*Status Registers*)	
FLAGS Status Flags (see FLAGS)	
Eriregistrid (*Special Registers, 386+ only*)	
CR0 Control Register 0	DR0 Debug Register 0
CR2 Control Register 2	DR1 Debug Register 1
CR3 Control Register 3	DR2 Debug Register 2
	DR3 Debug Register 3
TR4 Test Register 4	DR6 Debug Register 6
TR5 Test Register 5	DR7 Debug Register 7
TR6 Test Register 6	
TR7 Test Register 7	

Registrite kasutamine

Register	Default Segment	Valid Overrides
BP	SS	DS, ES, CS
SI or DI	DS	ES, SS, CS

DI strings	ES	None
SI strings	DS	ES, SS, CS

Programmi käsk loetakse mälust **käsuregistrisse**, kus seda hoitakse seni, kuni **käsudekooder** ta ära tunneb. Käsu järgi määrab juhtautomaat protsessori masinatsükli ning realiseerib algoritmi. Algoritmi täitmiseks väljastab **juhtautomaat** signaalid protsessori funktsionaalsetele osadele: *ALU*-le, registritele jms. Juhtautomaadi algoritm määrab ära, millist operatsiooni täidetakse ning kuidas toimivad protsessori üksikud osad. Järgmise käsu täitmiseks kirjutatakse käsuloendurisse uus aadress.

Olekuregistri sisu oleneb aritmeetika-loogikaplokis sooritatava tehte tulemusest. Näiteks kui tehte tulemusena saadakse null, s. t tulemi kõik bitid on nullid, siis fikseeritakse olekuregistris nn nulli tunnus ehk **lipp** (*F - flag*). Selleks on registris eraldatud üks bitt - nulltunnuse bitt ehk nullbitt (*Z - zero*), mis tunnuse olemasolu korral viiakse olekusse $Z = 1$. Kui arvude liitmisel toimub vanemast bitist ülekanne, siis fikseeritakse ülekande tunnus (*C - carry*) ning tunnuste registri *C*-bitt viiakse olekusse $C = 1$. Seda registrit nimetatakse ka lippude registriks. Protsessorites kasutatakse erineva lippude arvuga olekuregistreid

Olekuregistri sisu järgi toimub siirdekäskude täitmine. Sõltuvalt keskprotsessori (*CPU - central processor unit*) tüübist kasutatakse täiendavalt **tingimuslike siirete plokki**, mis universaalarvutis *Pentium* on edasi arendatud **järgmist käsku prognoosivaks plokiks**. Sel viisil saavutatakse arvuti töökiiruse suurenemine.

Enamkasutatavatel lippudel on järgmine tähendus:

S (***N***) (*sign*) negatiivne tulem. Lipp seatakse $S = 1$, kui registri sisu on pärast käsu täitmist negatiivne. 8-bitises registris võib märgiga arve kujutada piirkonnas $-(2^7) = -128$ kuni $(2^7 - 1) = 127$;

Z (*zero*) nulltulem. Lipp seatakse $Z = 1$, kui registri sisu on 0; lipp pööratakse $Z = 0$, kui registri sisu ei ole 0;

AC (*auxiliary carry*) abi- ehk dekaadülekanne. Lipp seatakse $AC = 1$, kui on ülekanne registri madalamast poolbaidist kõrgemasse poolbaiti, s. t kolmandast bitist neljandasse (registri bitid nummerdatakse paremalt vasakule). Seda tunnust kasutatakse kümnendaritmeetika kahend-kümnendkoodis teheteks;

P (*parity*) paaristulem. Lipp seatakse $P = 1$, kui registri ühtede summa on paaris, ja $P = 0$, kui summa on paaritu;

CY (*carry*) ülekanne. Lipp seatakse $CY = 1$, kui oli ülekanne registri seitsmendast bitist. Kui ülekannet ei olnud, siis $CY = 0$.

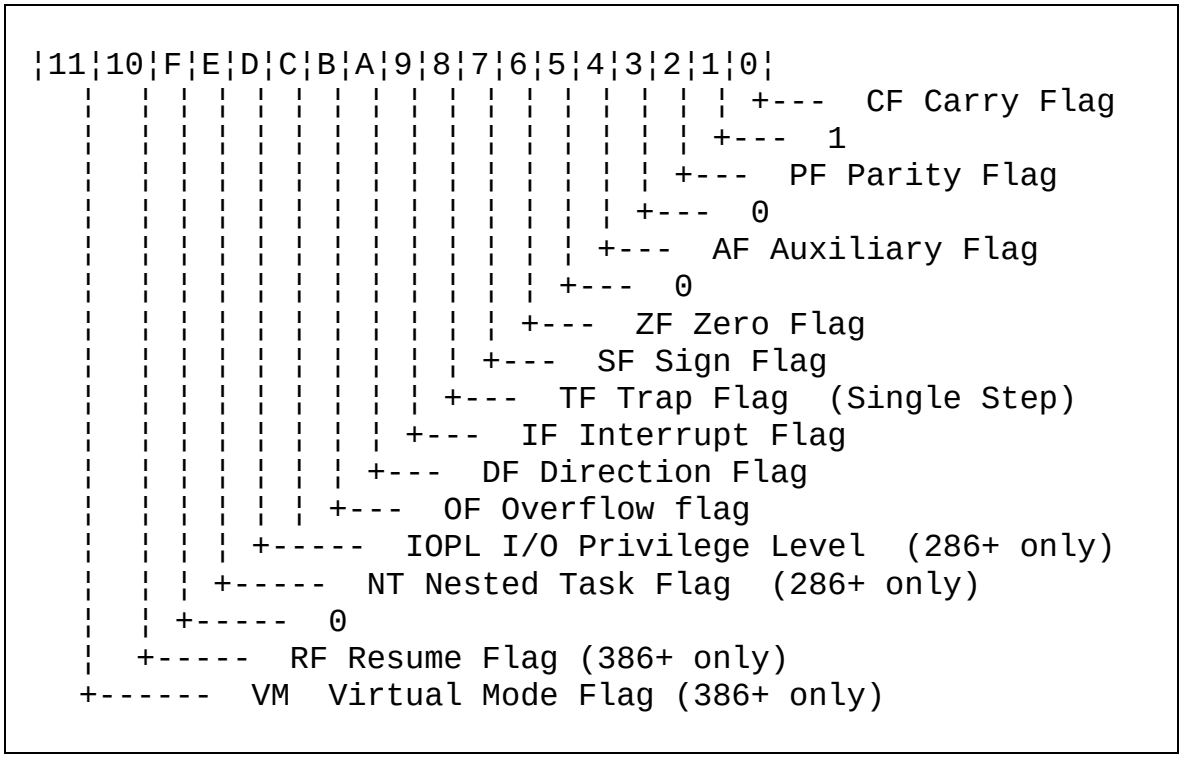
T (*trap*) püünisetunnus. Kui T = 1, s. t lipp on seatud, siis toimub programmi katkestus pärast jooksva käsu täitmist ja siire püüniseks nimetatavasse

eriprogrammi. Seda lippu kasutatakse peamiselt programmi silumisel.

Lippude tähistena võib kasutada ka mitmeid teisi tähti. Mikroprotsessori *Intel* 8086 ja 80386 lippudest annab ülevaate tabel 2.2.

Tabel 2.2

Protsessoriperekonna *Intel* 8086, 80386 olekuregister



Registreid ***A*** (akumulaator) ja ***F*** (lipud) vaadeldakse mõnikord registripaarina, **programmi olekuregistrina**. Eriline tähtsus on seejuures registril *F*, mis on ette nähtud tulemitunnuste salvestamiseks.

Protsessori olekusõna (*PSW - processor status word*). Protsessor genereerib iga masinatsükli esimese takti jooksul andmesiinile protsessori olekusõna, mis iseloomustab masinatsüklit ja sisaldab protsessori töö kohta täpsemat infot. Protsessori olekusõna salvestatakse protsessori välisesse olekuregistrisse ning seda kasutatakse liiteseadmete juhtimiseks.

2.2.3. Ajadiagrammid

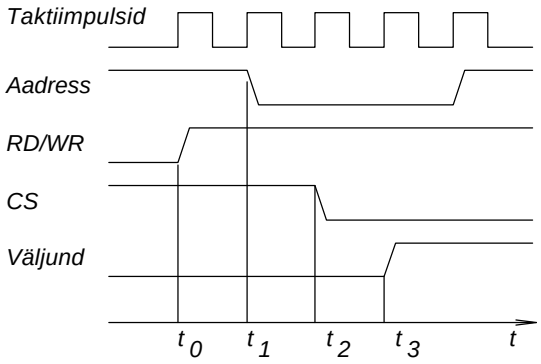
Protsessori töö ajalist kulgu iseloomustavad ajadiagrammid, mille põhjal saab hinnata ühe või teise operatsiooni elementaartehete järjekorda ning operatsiooni sooritamise kestust. Joonistel 2.11 ja 2.12 on protsessori ja mälu koostööd iseloomustavad ajadiagrammid. Mälust lugemisel sooritab protsessor järgmised elementaartehted:

1) juhtsignaal *RD/WR* viiakse olekusse *RD* (*read*), mis tähendab lugemist,
2) mälupesa aadressisõna, kust soovitakse lugeda infot, saadetakse aadressisiinile,
3) kiibivaliku siganaaliga *CS* (*chip select*) avatakse juurdepääs mällu,
4) loetakse mälupesa sisu väljundisse (andmesiinile).

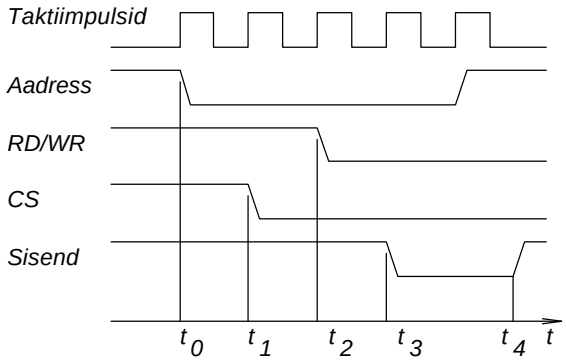
Mällu kirjutamisel sooritatakse järgmised elementaartehted:

1) mälupesa aadressisõna, kuhu soovitatakse kirjutada infot, saadetakse aadressisiinile,
2) kiibivaliku signaaliga *CS* (*chip select*) avatakse juurdepääs mällu,
3) juhtsignaal *RD/WR* viiakse olekusse *WR* (*write*), mis tähendab kirjutamist,
4) mällu kirjutatav infosõna saadetakse mälu sisendisse (andmesiinile).

Protsessori, mälu ja välisseadmete töö ajaliseks korraldamiseks kasutatakse sünkroniseerimist.



Joonis 2.10. Mälust lugemise ajadiagramm



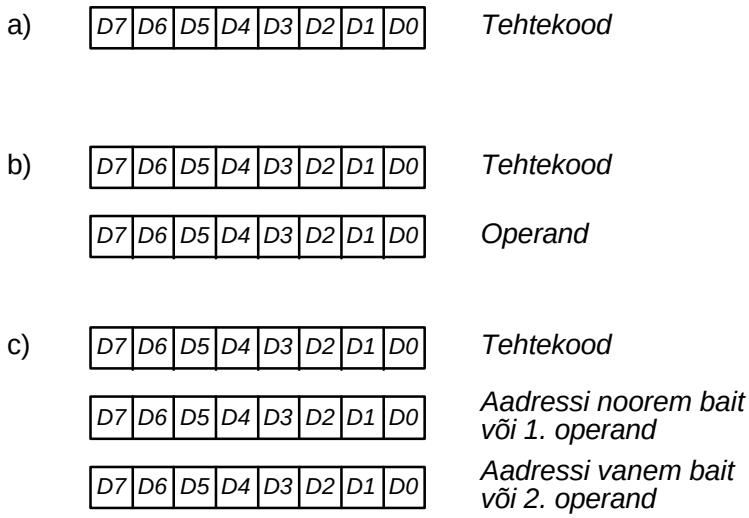
Joonis 2.11. Mällu kirjutamise ajadiagramm

2.2.4. Käsu- ja andmevormingud

Arvuti töötab **programmjuhtimise** põhimõttel. Programmeerimine toimub käskudega. **Käsk** sisaldab informatsiooni selle kohta, missugust tehet tuleb täita. Informatsiooni, mida arvutis töödeldakse, nimetatakse **andmeteks**. Andmete töötluseeskirja nimetatakse

algoritmiks. Algoritmi saab esitada plokk- ja graafskeemina või ka programmina. Käskude jadana esitatud algoritmi nimetatakse **programmiks**. Programm ja andmed salvestatakse mällu, kus need on protsessorile kergesti kättesaadavad. Arvuti töö seisneb programmi automaatses täitmises.

Käsus sisalduv informatsioon on määratud **käsuvorminguga**. Käsuvormingu peamised osad on **tehte-** ehk **operatsioonikood** ja **aadressid**. Tehtekood näitab, missugust tehet on vaja sooritada, aadressid näitavad tehete operandide asukohta mälus, s. o mälupesa asukohta või registri nime. Mälus paiknevate käskude või andmete poole pöördumist nimetatakse **adresseerimiseks**. Koos aadressiga antakse info adresseerimise viisi kohta.



Joonis 2.12. Käsuvormingu näited:
a) ühebaidine käsk, b) kahebaidine käsk, c) kolmebaidine käsk

Täidetava funktsiooni järgi võib arvuti käsud jaotada järgmistesse rühmadesse: 1) andme-edastuskäsud, 2) aritmeetika- ja loogikakäsud, 3) siirdekäsud, 4) alamprogrammikäsud, 5) nihkekäsud, 6) sisendi-väljundikäsud ja 7) muud erikäsud.

Käsu täitmine toimub etappide kaupa. **Aritmeetika-loogikakäskude** korral eristatakse järgmisi etappe: 1) käsu lugemine mälust, 2) operandide lugemine, 3) tehte sooritamine, 4) tulemi salvestamine, 5) järgmise käsu ettevalmistamine. **Siirdekäskude** puhul on olulised 1) käsu lugemine, 2) tingimuste analüüs, 3) järgmise käsu ettevalmistamine.

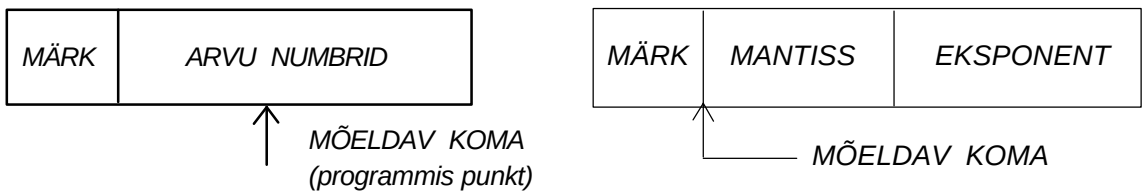
Andmed. Arvutid kasutavad järgmisi andmete tüüpe: arve, loogikaväärtusi, märke ja märgijadasid. Andmesõna pikkus avaldab olulist mõju andmete edastamisele. Mahukamate andmete korral (näiteks ujukomaarvude kasutamisel) koosneb andmeüksus e andmesõna mitmest (2 ... 4) arvutisõnast (8 ... 32 bitti). Arvulised andmed on enamasti kahendsüsteemis ning esitatakse püsikoma- ja/või ujukomaarvuna (joonis 2.13).

Püsikomaarvude puhul on koma arvu mäluvälja kindlal positsioonil. Vanematel protsessoritüüpidel paikneb koma enamasti esimese numbrikoha ees, s. t mälus saab kujutada vaid ühest väiksemaid püsikomaarve. Kaasaegsematel ei määrata koma arvuti

lülitusega ega kanta koos numbritega mällu, vaid koma asukoht määratakse kindlaks programmiga. Püsikomaarvude kasutamisel võib tekkida mäluvälja **ületäitumine**, s. o olukord, kus arvutustehte tulem ei mahu mäluvälja. Samuti võib esineda mäluvälja **alatäitumist**, s. o olukorda, kus arvutustulem on niivõrd väike, et kujutub mäluvälja piirides nullina. Töötamisel ujukomaarvudega ilmnevad nimetatud ebameeldivused tunduvalt harvemini. Püsikomaarvudel määrab arvu kõrgeim koht arvu märgi; plussmärki tähistatakse 0-ga, miinusmärki aga 1-ga. Ülejäänud bitid esitavad kas murdarve 0, 2^{-15}, ... 1-2^{-15} või täisarve 0, 1, 2 ... 2^{15} -1. **Negatiivseid arve** esitatakse üldjuhul täiendkoodis. **Murdarvulisel** kujutamisel täiendkood suurendab ehk täiendab arvu absoluutväärtust kaheni, **täisarvulisel** kujutamisel 2^n-ni, näiteks 2^{16}-ni.

Ujukomaarvude korral salvestatakse mälus lisaks märgile ja numbritele veel eksponent (arv, mis määrab koma asukoha). Eksponent näitab, mitme numbri võrra tuleb koma nihutada vasakule (kui eksponent <0) või paremale (kui eksponent >0), lähtudes algseisust, kus koma oli vahetult esimese numbri ees. Ujukomaarvud salvestatakse enamasti normaliseerituna, s.o selliselt et arvu numbrikohtade jada ehk **mantissi** esimene number erineb nullist. Kui mällu suunatakse tingimusele mittevastav arv, normaliseerib arvuti selle enne salvestamist. Ujukomaarvu kood koosneb mantissist M ja eksponendist a, mis esitavad arvu 2^aM, enamasti $0{,}5 \leq M < 1$. 16-bitistel arvutitel on ujukomaarv tavaliselt kahes arvutisõnas, kusjuures astmenäitaja ja mantiss paiknevad teineteise suhtes erinevatel arvutitel erinevalt.

Normaliseerimiseks nimetatakse mantissi numbrite nihutamist ja eksponendi muutmist nii, et arvu väärtus ei muutu. Näiteks arvu 8 900 000 ujukomaesituse moodustavad märk +, mantiss 89 (millele lisanduvad nullid) ja eksponent 7. Ujukomaarvude esitus on analoogiline tavalisele arvude esitusele 10 astme kaudu kujul $N = M \times 10^p$, kus mantissi tähistab M ja eksponenti p. Kui analoogiliselt talitada kahendarvuga, s. t esitada ta 2 astme kaudu, vastab arvu kuju ujukomakahendarvule. Arve on otstarbekas esitada ujukomaarvudena väga väikeste ja väga suurte arvude, näiteks füüsikaliste konstantide korral ja nendega opereerimisel.



Joonis 2.13. Püsi- ja ujukomaarvud

Loogikaandmeid esitavad arvutisõna või bitid eraldi. Vastavalt sellele töödeldakse loogikaandmeid kas terve sõna ulatuses või sõna üksikute bittide kaupa.

Märgijadasid kasutatakse enamasti võimsates arvutites. Väikestes juhtarvutites eraldi andmeliigina neid üldjuhul ei kasutata. Märgijadadena esitatud andmed haaravad tavaliselt suvalise arvu arvutisõnu. Märgi (tähe, numbri jne e tärgi) esitusüksus on bait, mis kõige sagedamini koosneb 8 bitist.

Programmeerija kasutab sageli ka **kompleksarve**, kuid nendega sooritatavad tehted teisendatakse programmide abil teheteks arvude **reaal- ja imaginaarosadega**.

Lisaks aritmeetilistele ehk arvandmetele salvestavad ja töötlevad arvutid suvalisi **tekstijadasid**, mis on koostatud arvutile tuntud märkidest (numbrid, suurtähed, mitmesugused erimärgid). Selliseid tekste nimetatakse märgistringideks.

Märgistringid salvestatakse arvuti mällu ASCII-koodis järjestikuste baitidena nii, et iga märk hõlmab ühe baidi. Stringide hulgas eristatakse bitistringe, mis koosnevad ainult bittidest. Bitistringidel põhineb infoloogiliste ülesannete lahendamine.

Käsu lugemisel antakse käsuloendurist ette käsu aadress. Selleks viiakse käsuloenduri sisu aadressiregistrisse ning edastatakse aadressisiini kaudu mällu. Ühtlasi antakse mälule lugemiskäsk. Tulemusena saadakse järjekordne käsk, mis viiakse andmesiini kaudu protsessori käsuregistrisse. Kui käsk näeb ette tehet mingi operandiga, siis sisaldub käsus ka operandi aadress. Kahe operandi korral antakse käsuga ette kaks aadressi.

Operandi lugemisel pöördutakse käsuga määratud mälu aadressi või registri poole ning suunatakse sealt saadud operand protsessori registrisse. Kui tehe tuleb sooritada kahe operandiga, loetakse mälust ka teine operand ning viiakse samuti protsessori registrisse. **Tehe sooritatakse** protsessoris (aritmeetika-loogikaplokis) käsu tehtekoodi kohaselt. **Tulem salvestatakse** akumulaatorisse või kaheoperandiliste tehete korral teise operandi asemele mälupessa. **Järjekordse käsu ettevalmistamine** seisneb käsuloenduri formeerimises. Kui programmi käske täidetakse järjestikku, siis suurendatakse käsuloenduri sisu ühe võrra. Siirdekäskude korral suunatakse sõltuvalt lipust käsuloendurisse siirde aadress ($Z = 1$) või suurendatakse käsuloenduri sisu ühe võrra ($Z = 0$). Seega sõltub tingimusest (lipust), kas programmi täitmist jätkatakse järgmisest käsust või pöördutakse tingimussiirde käsus näidatud aadressi poole ja jätkatakse programmi täitmist sealt. Kõiki loetletud tegevusi juhib protsessori juhtplokk.

2.2.5. Protsessori käsustik

Mikroarvuti programmeerimine masinakoodis või assemblerikeeles eeldab kõigi tema käskude põhjalikku tundmist. Mikroarvutites kasutatakse erisuguseid masinakoode ja assemblerikeeli. Seepärast tuleb lugejal täpsema teabe saamiseks pöörduda käsiraamatute või konkreetsete arvutite ja juhtseadmete juhendite poole. Siinkohal tutvustatakse firma *Intel* 8-bitise protsessori 8080 käsustikku (vt tabel 2.3), millest hiljem on välja arenenud hilisemate protsessoripõlvkondade 80286, 80386 ja 80486 käsustikud. Universaal-protsessori käsud võib jaotada rühmadesse.

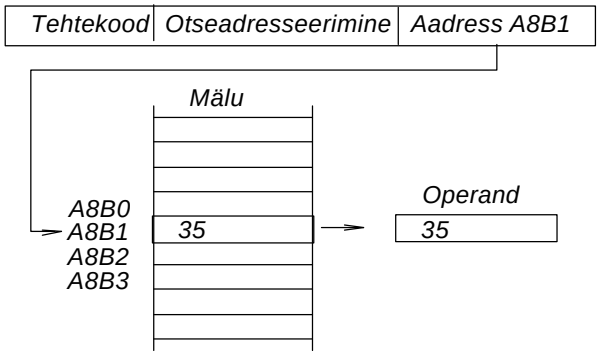
Andmeedastuskäskudega saab viia infot ühest registrist teise, mälust registrisse või vastupidi. Tüüpiliseks käsuks on näiteks ***MOV*** $\boldsymbol{r}_1, \boldsymbol{r}_2$ (*move* - liikuma), mis tähendab, et info viiakse registrist $\boldsymbol{r}_2$ registrisse $\boldsymbol{r}_1$. **Aritmeetika-loogikakäsud** võimaldavad sooritada vastavaid tehteid. Näiteks käsuga ***ADD B*** liidetakse akumulaatori sisu registri *B* sisule ning tulem salvestatakse akumulaatorisse. **Siirdekäsud** jaotatakse tingimusteta ja tingimuslikeks

siirdekäskudeks. Tingimusteta siirdekäsuks on näiteks ***JMP ad*** (*jump*), mis tõlkes tähendab hüpet. Käsk sisaldab aadressi (*ad*), kuhu tuleb käsu järgi siirduda, näiteks *JMP E5 F0*. Tingimuslikud siirdekäsud võimaldavad kontrollida tulemitunnuseid ning siirduda teatud programmiharusse. Nende käskudega organiseeritakse programmi tsüklilist täitmist. Näiteks käsk ***JNZ ad*** (*jump on no zero*) tähendab, et juhul kui tulem ei võrdu nulliga, tuleb siirduda käsus näidatud aadressile. Vastupidisel juhul jätkub programmi täitmine järgmisest mällu salvestatud käsust. Siirdekäskudega sarnased on ka **alamprogrammikäsud** näiteks ***CALL ad***, mille kohaselt pöördutakse käsus näidatud aadressi poole, kust algab mingi alamprogramm. Alamprogramm lõpeb käsuga ***RET*** (*return*), mis tähendab, et tuleb pöörduda tagasi endise aadressi poole, kus töö alamprogrammi siirdumisel katkes.

2.2.6. Adresseerimine

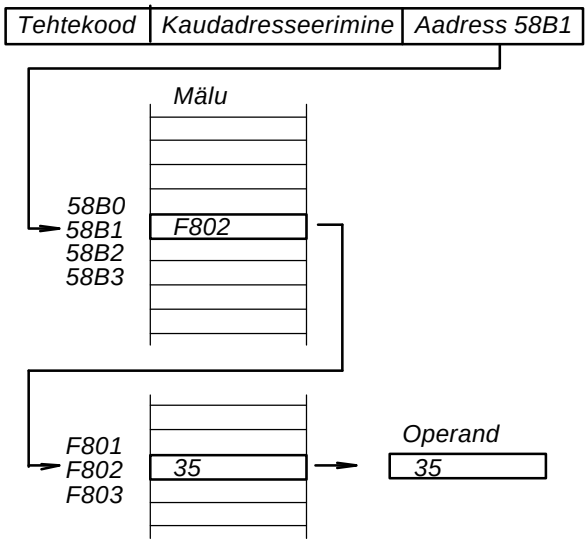
Operandide adresseerimiseks kasutatakse mitut viisi: otse- ja kaudadresseerimist, suht- ja indeksadresseerimist, vahetut adresseerimist, aga ka mitmesuguseid kombineeritud adresseerimisviise nagu kaudset indeksadresseerimist jms. Adresseerimise viise selgitavad joonistel 2.14 ...2.18 toodud skeemid. Käsus sisalduva teabe põhjal leitakse vajalik mälupesa ning loetakse sealt soovitud operand. Protsessoril võib olla 10 ja enam erinevat adresseerimisviisi.

Otseadresseerimisel antakse käsuga ette operandi aadress, mille järgi leitakse mälust operand.



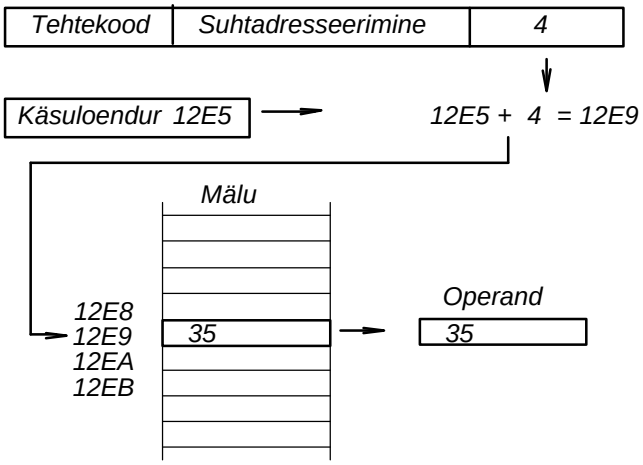
Joonis 2.14. Operandi leidmine otseadresseerimise korral

Kaudadresseerimisel leitakse kõigepealt mälust operandi aadress ning seejärel teisest mälupesast operand.



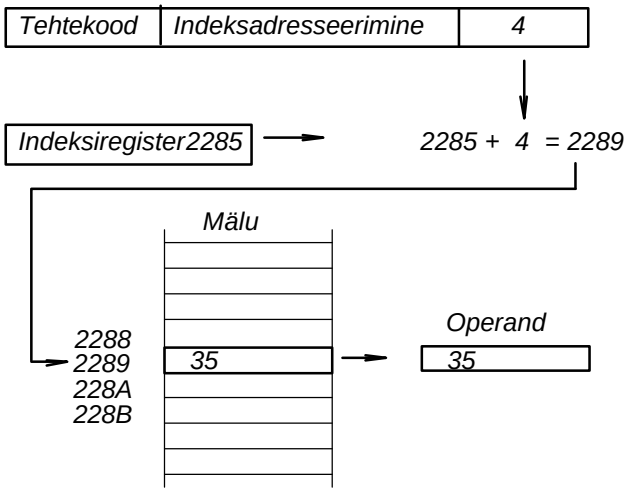
Joonis 2.15. Operandi leidmine kaudadresseerimise korral

Suhtadresseerimisel antakse operandi aadress käsuloenduri (programmi jooksva aadressi) suhtes. Operandi aadress leitakse käsuloenduri sisu ja suhtaadressi summeerimisega.



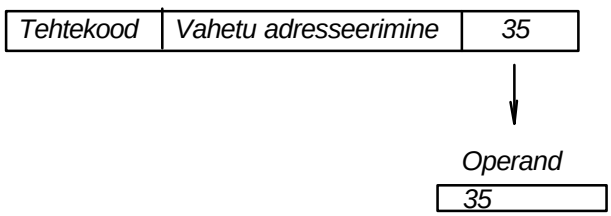
Joonis 2.16. Operandi leidmine suhtadresseerimise korral

Indeksadresseerimine sarnaneb suhtadresseerimisega, kuid käsuloenduri asemel kasutatakse baasaadressina indeksiregistris salvestatud aadressisõna.



Joonis 2.17. Operandi leidmine indeksadresseerimise korral

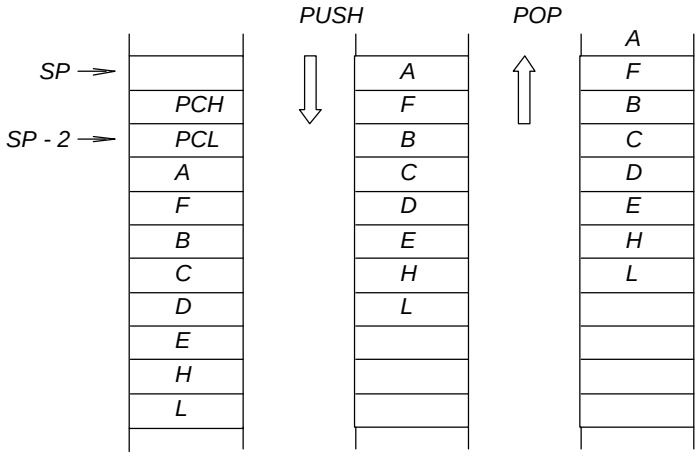
Vahetul adresseerimisel antakse operand otse käsuga.



Joonis 2.18. Operandi leidmine vahetu adresseerimise korral

2.2.7. Pinumälu

Pinumälu ehk lihtsalt pinu (*stack*) on registrite kogum, kuhu saab lühiajaliselt salvestada infot. Pinumälu on realiseeritud kas nihkeregistrina või kasutatakse selleks osa arvuti muutmälust. Tuntakse kahte liiki pinumälusid: *LIFO-* (*last in first out*) ja *FIFO-* (*first in first out*) tüüpi mälu. *LIFO*-tüüpi pinu võib võrrelda padrunisalvega, kust viimasena laaditud padruni saab kätte esimesena. Pinumälu kasutatakse koos pinumälu viidaga (*stack pointer*), mis fikseerib pinu asukoha (aadressi) üldkasutatavas muutmälus. Arvutis kasutatakse pinumälu kõige enam selleks, et sinna ajutiselt kirjutada protsessori registrite sisu juhul, kui protsessor töötleb vaheldumisi mitut algoritmi ning registrite sisu ei tohi töö katkestamisel kaduma minna. Pinumällu salvestatakse ka protsessori olekusõna *PSW* (*processor status word*). Protsessori olekusõnaks on akumulaatori ja tunnuste registri sisu ehk lihtsalt aku ja lipud. Protsessori registrite sisu kirjutatakse pinusse käsuga *PUSH* ning pinust registritesse käsuga *POP* (joonis 2.19).

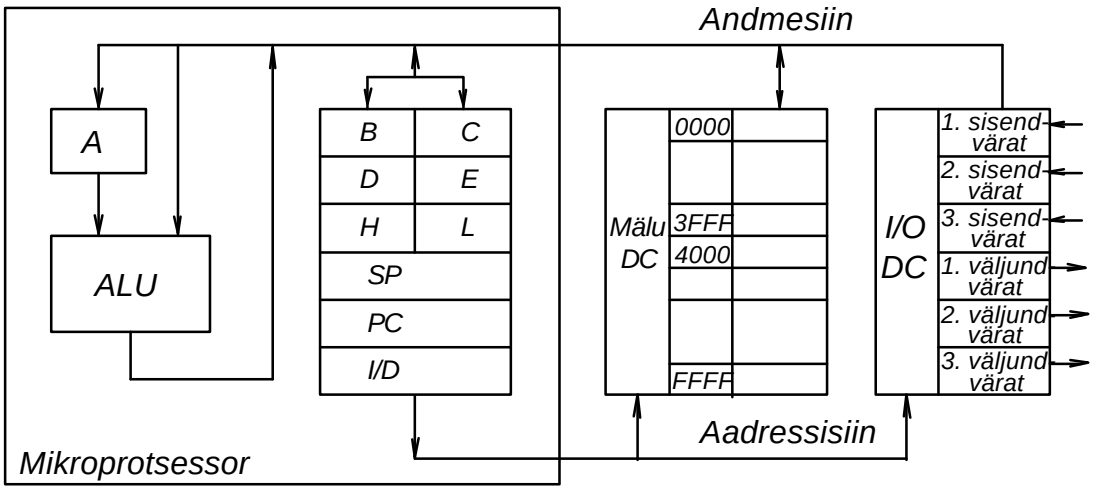


Joonis 2.19. Pinumälu ja selle kasutamine

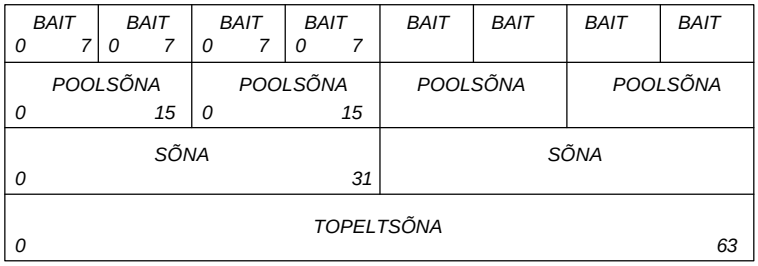
2.2.8. Protsessori koostöö mälu ja välisseadmetega

Arvuti koosneb protsessorist, mälust, sisend-väljundseadmetest ja neid kõiki ühendavast siinide süsteemist. Mikroprotsessorsüsteemi loomisel ja programmeerimisel tuleb saavutada nende põhikomponentide koostöö (joonis 2.20).

Mälu organiseerimine seisneb arvuti mäluseadmete ja mälupiirkondade valikus ning mäluväljade aadresside jaotuses. **Mäluväli** on järjestikuste baitide rühm, mille pikkuseks loetakse baitide arvu väljas ja aadressiks välja esimese baidi aadressi. Välja bitid nummerdatakse alates vasakpoolsemast järjenumbritega 0, 1, ... jne. Fikseeritud mäluväljadena (joonis 2.21) kasutatakse **poolsõna, sõna ja topeltsõna**, mille pikkusteks on kaks, neli ja kaheksa baiti, kusjuures välja aadress peab jaguma pikkusega (2, 4 või 8-ga).

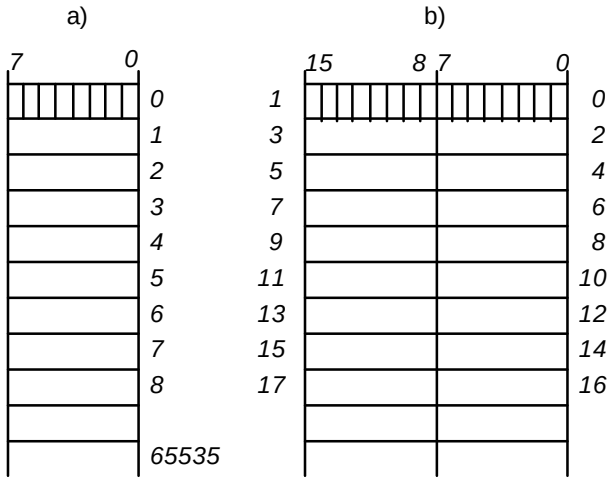


Joonis 2.20 Mikroarvuti programmisti seisukohalt



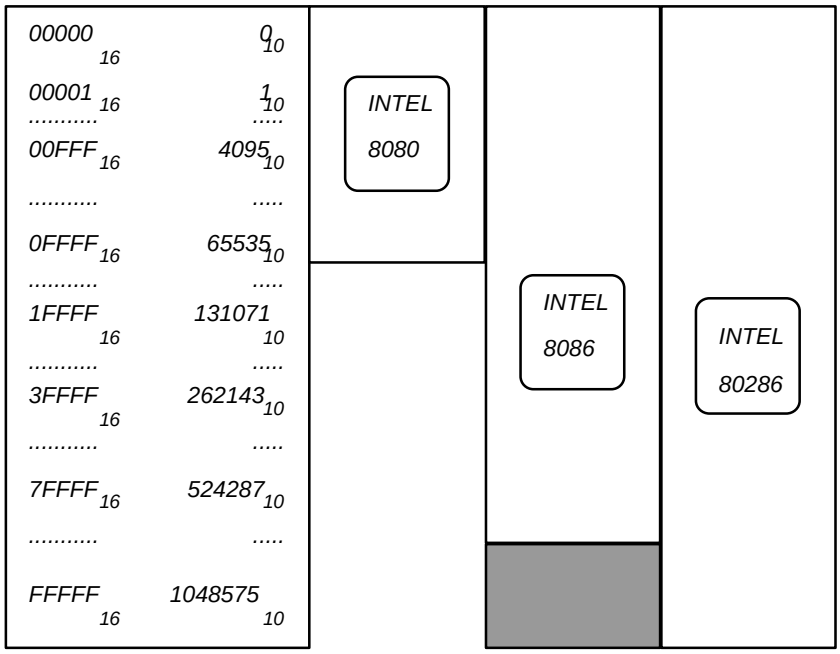
Joonis 2.21. Mäluväljad

Põhimälu struktuur. Adresseerimine võib toimuda kas **sõna-** või **baitadresseeringuga**, kus vähimaks adresseeritavaks mäluüksuseks on sõna või bait (joonis 2.22). Baitadresseering tagab mälu ökonoomse kasutamise ning võimaldab esitada käske ja andmeid eri vormingutes. Andmeid salvestatakse muutuva ja fikseeritud pikkusega mäluväljadele.



Joonis 2.22. Ühe- ja kahebaidised mäluväljad ja nende adresseerimine

Igal **põhimälu** baidil on **aadress** (s.t järjenumber). Aadresside numeratsioon algab nullist. Arvuti opereerib kahendaadressidega, aadresside üleskirjutamiseks kasutatakse harilikult 16-ndsüsteemi. Sõltuvalt arvuti mudelist ja põhimälu mahust hõlmavad aadressid järgmisi standardseid vahemikke (joonis 2.23).



Joonis 2.23. Erinevate protsessoritega adresseeritavad põhimälud

Andmevahetuse korraldamine hõlmab andmete sisestamise välisseadmetelt protsessorisse ja nende väljastamise protsessorist välisseadmetesse. Sõltuvalt sellest kas välisseade on ette nähtud andmete sisestamiseks või väljastamiseks, nimetatakse seda sisend- või väljundseadmeks. Üldjuhul kasutatakse mõistet sisend-väljundseadmed. Sisend-väljundseadmeteks on näiteks klahvistik ja kuvar, printer, magnetketasmälu, andurid, täiturid ja mõõteriistad ning mitmesugused sidekanalid. On olemas mitmeid andmevahetuse meetodeid, mida tutvustatakse lähemalt järgmises peatükis.

Tabel 2.3

Universaalprotsessori INTEL 8080 käsustik

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	NOP	LXI B,&	STAX B	INX B	INR B	DCR B	MVI B,^	RLC	-	DADB	LDAX B	DCX B	INR C	DCR C	MVI C,^	RRC	0
1		LXI D,&	STAX D	INX D	INR D	DCR D	MVI D,^	RAL	-	DADD	LDAX D	DCX D	INR E	DCR E	MVI E,^	RAR	1
2		LXI H,&	SHLD *	INX H	INR H	DCR H	MVI H,^	DAA	-	DADH	LHLD *	DCX H	INR L	DCR L	MVI L,^	CMA	2
3		LXI SP,&	STA *	INX SP	INR M	DCR M	MVI M,^	STC	-	DADSP	LDA *	DCX SP	INR A	DCR A	MVI A,^	CMC	3
4	MOV B,B	MOV B,C	MOV B,D	MOV B,E	MOV B,H	MOV B,L	MOV B,M	MOV B,A	MOV C,B	MOV C,C	MOV C,D	MOV C,E	MOV C,H	MOV C,L	MOV C,M	MOV C,A	4
5	MOV D,B	MOV D,C	MOV D,D	MOV D,E	MOV D,H	MOV D,L	MOV D,M	MOV D,A	MOV E,B	MOV E,C	MOV E,D	MOV E,E	MOV E,H	MOV E,L	MOV E,M	MOV E,A	5
6	MOV H,B	MOV H,C	MOV H,D	MOV H,E	MOV H,H	MOV H,L	MOV H,M	MOV H,A	MOV L,B	MOV L,C	MOV L,D	MOV L,E	MOV L,H	MOV L,L	MOV L,M	MOV L,A	6
7	MOV M,B	MOV M,C	MOV M,D	MOV M,E	MOV M,H	MOV M,L	HLT	MOV M,A	MOV A,B	MOV A,C	MOV A,D	MOV A,E	MOV A,H	MOV A,L	MOV A,M	MOV A,A	7
8	ADD B	ADD C	ADD D	ADD E	ADD H	ADD L	ADD M	ADD A	ADC B	ADC C	ADC D	ADC E	ADC H	ADC L	ADC M	ADC A	8
9	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB M	SUB A	SBB B	SBB C	SBB D	SBB E	SBB H	SBB L	SBB M	SBB A	9
A	ANA B	ANA C	ANA D	ANA E	ANA H	ANA L	ANA M	ANA A	XRA B	XRA C	XRA D	XRA E	XRA H	XRA L	XRA M	XRA A	A
B	ORA B	ORA C	ORA D	ORA E	ORA H	ORA L	ORA M	ORA A	CMP B	CMP C	CMP D	CMP E	CMP H	CMP L	CMP M	CMP A	B
C	RNZ	POP B	JNZ *	JMP *	CNZ *	PUSH B	ADI ^	RST 0	RZ	RET	JZ *	-	CZ *	CALL *	ACI ^	RST 1	C
D	RNC	POP D	JNC *	OUT N	CNC *	PUSH D	SUI ^	RST 2	RC	-	JC *	IN N	CC *	-	SBI ^	RST 3	D
E	RPO	POP H	JPO *	XTHL	CPO *	PUSH H	ANI ^	RST 4	RPE	PCHL	JPE *	XCHG	CPE *	-	XRI ^	RST 5	E
F	RP	POP PSW	JP *	DI	CP *	PUSH PSW	ORI ^	RST 6	RM	SPHL	JM *	EI	CM *	-	CPI ^	RST 7	F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

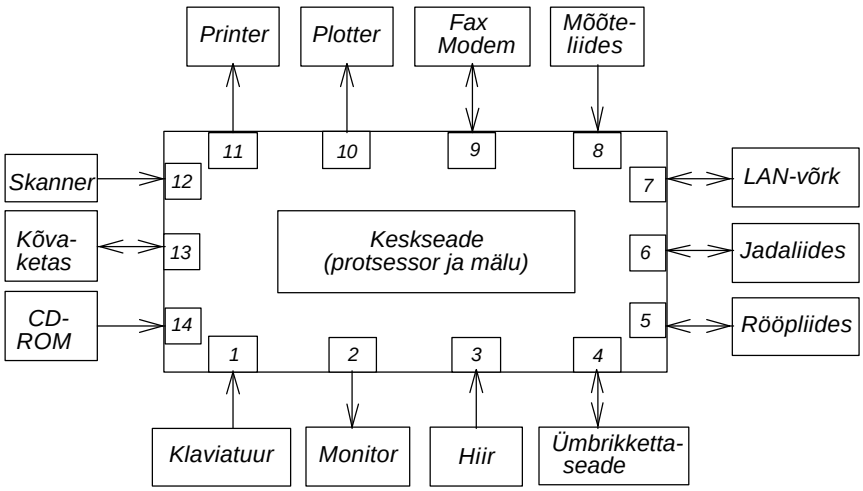
N - sisend- või väljundvärati number
& - kahebaidine operand
* - kahebaidine operand-aadress
^ - ühebaidine operand

2.3. Andmevahetus

2.3.1. Andmevahetuse meetodid

Arvutit ümbritseb väliskeskkond, milles arvuti töötab ning lahendab programmiga antud ülesandeid. Keskkonnaks on arvutioperaator, sidekanalid (arvutivõrk), automaatseadmed või -protsessid. Infovahetuseks kasutab arvuti mitmesuguseid välisseadmeid, millelt saab infotöötlusprotsessi lähteandmed ning kuhu väljastab tulemid ja juhtsignaalid. **Välisseadmed** jagunevad andmete allikateks e sisendseadmeteks ning andmete tarbijateks e väljundseadmeteks.

On võimalik, et välisseade on nii andmete allikas kui ka tarbija (magnetkettaseade), sel juhul nimetatakse teda **sisend-väljundseadmeks,** sest andmevahetus toimub mõlemas suunas. Universaalarvuti enamkasutavad välisseadmed on joonisel 2.24.



Joonis 2.24. Universaalarvuti välisseadmed.

Automaatikasüsteemide juhtraalide korral on esmatähtsad juhitavat protsessi raaliga sidestavad välisseadmed e protsessiliidesed.

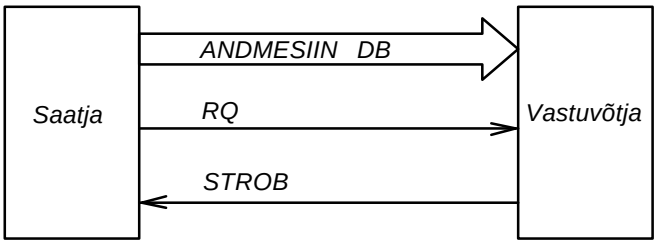
Andmevahetust iseloomustab kanali läbilaskevõime, liidese standard, andmevahetuse meetod ning andmevahetust algatav (initsieeriv) osapool. Kanali läbilaskevõimet mõõdetakse sekundis edastatud infohulgaga ehk boodidega. 1 bood = 1 bit/s. Vastavalt 1 Kbood = 1 Kbit/s ja 1 Mbood = 1 Mbit/s.

Andmevahetus protsessori ja välisseadmete vahel toimub sünkroonse või asünkroonse andmevahetuse põhimõttel. Sünkroonse andmevahetuse korral juhib andmevahetust protsessor ning välisseade töötab sünkroonselt arvuti taktigeneraatoriga. Enamik välisseadmeid töötab autonoomselt ning pole sünkroniseeritud arvuti taktigeneraatoriga.

Klassikalised asünkroonse andmevahetuse meetodid on

1) kviteerimismeetod (ingl k *handshaking*),
2) skaneerimismeetod,
3) katkestussignaalide meetod,
4) vektorkatkestuse meetod.

Asünkroonse andmevahetuse (joonis 2.25) korral edastatakse infot saatjalt vastuvõtjale, kusjuures enne andmevahetuse algust on mõlemad hõivatud oma ülesannetega. Üldjuhul on määramata hetk t_a, mil saatja soovib andmevahetust alustada; vastuvõtja valmisolek andmete vastuvõtuks; aeg, mis kulub signaaalide levimiseks liinis, ja aeg, mis vastuvõtjal kulub andmete registreerimiseks. Seepärast tuleb andmevahetuse korraldamiseks kasutada peale andmesiini *DB* ka täiendavaid juhtmeid. Küsitlussignaali *RQ* (ingl k *request*) kasutatakse selleks, et algatada andmevahetust. Signaal *STROB* on vastus saatja küsitlussignaalile *RQ*.

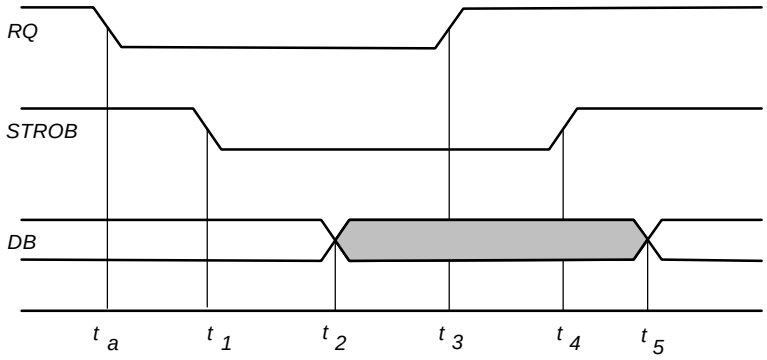


Joonis 2.25. Asünkroonne andmevahetussüsteem

Kviteerimismeetodi korral toimub andmevahetus joonisel 2.26 näidatud ajadiagrammi järgi. Jõudeolukorras on mõlema juhtsigaali nivoo kõrge. Hetkel t_a, kui saatja on valmis andmete väljastamiseks, saadab ta küsitlussignaali *RQ* nivoo madalaks muutmisega vastuvõtjale andmevahetuspäringu. Signaali *RQ* nivoo jääb madalaks seniks, kuni vastuvõtja on valmis vastuvõttu alustama, teatades sellest signaali *STROB* viimisega madalale nivoole. Kui vastuvõtja on vaba, saabub vastus kiiresti. Kui aga vastuvõtja on hõivatud, tuleb saatjal oodata kuni vastuvõtja vabanemiseni. Kui saabub vastus (t_1), väljastab saatja (t_2) andmed liinile. Hetkel t_3 teatab ta signaali *RQ* viimisega kõrgele nivoole, et andmevahetus on alanud. Selle teate järel asub vastuvõtja andmete vastuvõtule ning hetkel t_4, kui vastuvõtt on lõpetatud, saadab ta sellekohase kviitungi - signaali *STROB* kõrgele nivoole viimisega, mis teatab saatjale, et andmed on vastu võetud ning hetkel t_5 lülitab vastuvõtja andmesiini välja.

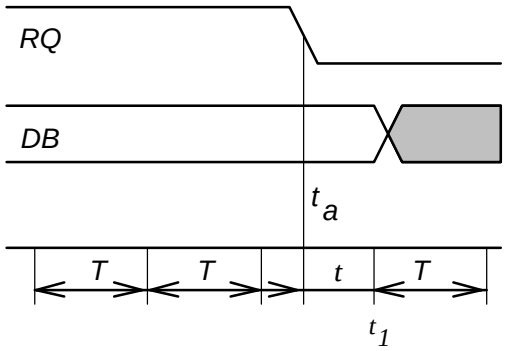
Kuna andmevahetus toimub vastastikuste küsitluste ehk kviitungi vahetamistega, siis tuleneb sellest ka meetodi nimetus. Seejuures ei ole ükski ajahetkedest jäigalt määratud, vaid nad kujunevad vabalt vastavalt sellele, kuidas protsess loomulikult kulgeb. Nii saatja kui vastuvõtja peavad reageerima signaali juhuslikule muutumisele hetkeil t_1 ja t_4 saatjas ning t_a ja t_3 vastuvõtjas. Et arvuti töötab rangelt determineeritud programmi järgi ning on

enamiku ajast hõivatud, peab signaalide avastamiseks kasutama erivõtteid, näiteks küsitlussignaali programmilist skaneerimist ja katkestussignaalide meetodit.



Joonis 2.26. Kviteerimismeetodiga andmevahetuse ajadiagramm

Skaneerimismeetodil töötades tuleb avastatav signaal (näiteks klaviatuuri küsitlussignaal *RQ*) lülitada ühele arvuti sisendkanalile. Programmis nähakse ette selle kanali perioodiline lugemine ja signaali nivoo määramine loogiliste tehetega. Skaneerimismeetodil andmevahetuse ajadiagramm on joonisel 2.27, kus T on skaneerimisperiood.



Joonis 2.27. Andmevahetuse ajadiagramm skaneerimismeetodil

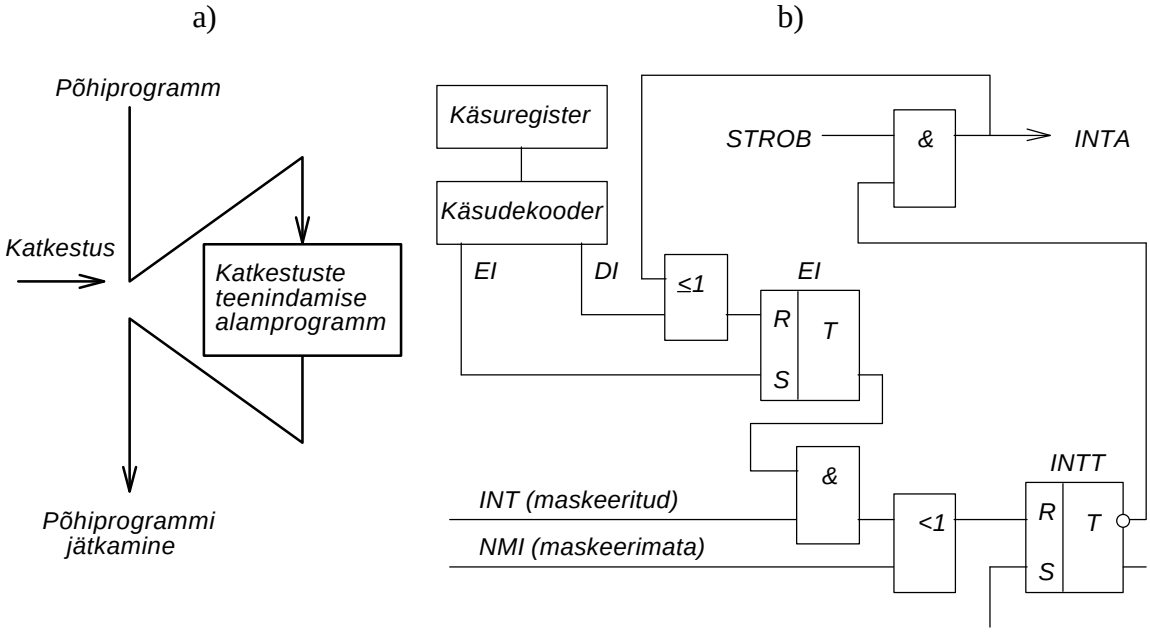
Skaneerimismeetodi puudusteks on

- kanali väike tõhusus, sest küsitlussignaali avastamine toimub harva ning süsteem on valdava aja jõudeolekus. Meetodit on otstarbekas kasutada siis, kui protsessorit vähe koormatakse;
- küsitlussignaali avastamine alles järjekordse skaneerimisprotseduuri ajal. Arvestades, et hilistumine τ < T, tuleb väikese reaktsiooniaja saavutamiseks valida lühike skaneerimisperiood, mis annab protsessorile aga märgatava lisakoormuse.

Skaneerimismeetodi eeliseks on lihtsus.

Katkestussignaalide meetod. Juhuslikul hetkel tekkiva signaali avastamise teiseks võimaluseks on katkestada protsessori töö spetsiaalse sisendsignaali, nn katkestussignaali abil. Katkestuse korral läheb protsessor põhiprogrammi täitmiselt üle katkestust ning andmevahetust teenindava alamprogrammi täitmisele (joonis 2.28). Pärast katkestust jätkab ta poolelijäänud põhiprogrammi täitmist. Meetodi realiseerimiseks varustatakse protsessor sisendiga *INT* (*interrupt*) ning väljundiga *INTA* (*interrupt asknowledged*).

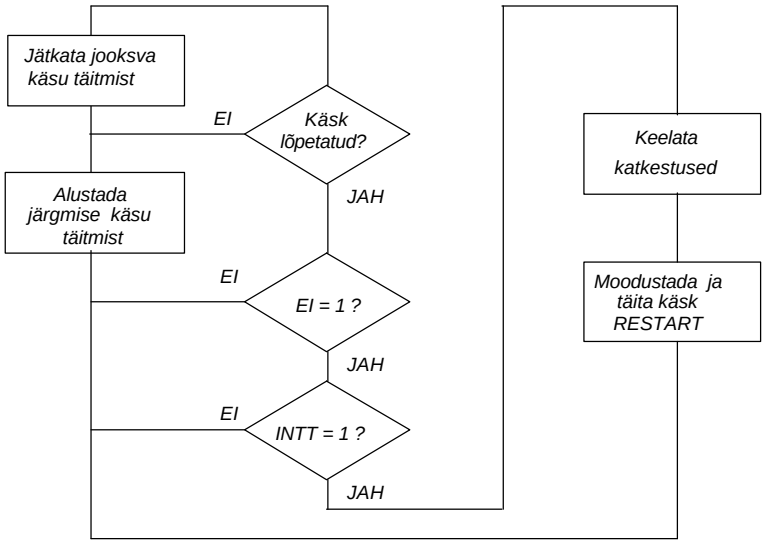
Katkestuse päringusignaal *INT* registreeritakse trigeriga *INTT*. Peale selle kuulub katkestussüsteemi veel vähemalt üks triger *EI* (*enable interrupt*), mille abil lastakse katkestussignaale sõltuvalt olukorrast läbi või tõkestatakse. Seda trigerit juhitakse nii aparatuuriga kui ka programmi erikäskude abil. Katkestuse võimalikkuse korral, mille kohta võetakse vastu loogiline otsus, annab protsessor katkestust lubava signaali *INTA* ja läheb põhiprogrammilt üle katkestuse alamprogrammi täitmisele.



Joonis 2.28. Katkestussignaalide meetod:
a) põhiprogrammi katkestamine; b) katkestuste registreerimine ja maskeerimine

Protsessori töötsüklis on ette nähtud katkestussignaalide olemasolu kontroll (joonis 2.29). Pärast käsu täitmise lõpetamist kontrollitakse esmalt, kas katkestus on lubatud. Kui jah, siis kontrollitakse trigeri *INTT* oleku järgi, kas käsu täitmise ajal on saabunud katkestussignaal. Kui ei, siis alustatakse järgmise käsu täitmist ja töö jätkub normaalselt. Kui aga katkestussignaal on saabunud (*INTT* = 1), siis keelatakse edasine katkestussignaali läbipääs trigeri *EI* viimisega olekusse 0, millega välditakse mitmekordne reageerimine samale katkestussignaalile. Moodustatakse ja täidetakse spetsiaalne taaskäivituskäsk *RESTART*, mille toimel jooksva programmi täitmine katkeb ja minnakse üle teisele, katkestavale programmile. Viimase algusaadressi määrab aparatuur või programm. Peale käsu *RESTART* täitmist jätkub protsessori töö normaalselt, aga juba katkestava programmi järgi ehk teisel programmitasandil.

Käsk *RESTART* on analoogiline alamprogrammi siirdumise käsuga, mille toimel käsuloenduri (*PC*) jooksev sisu salvestatakse näiteks mällu, käsuloendurisse aga laaditakse katkestava programmi algusaadress. Tagasipöördumine katkestatud põhiprogrammi toimub käsuga *RETURN* analoogiliselt alamprogrammist naasmisega.



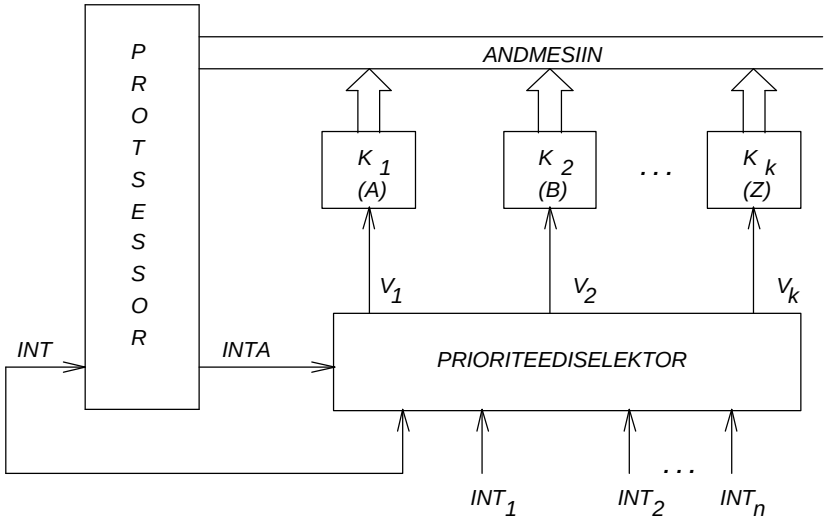
Joonis 2.29. Protsessori töötsükkel katkestussignaali olemasolu kontrollimisel

Tegelik üleminek katkestavale programmile ja tagasipöördumine sellest on komplitseeritum, sest katkestus võib toimuda igal hetkel ja mis tahes käsu täitmisel, nii et seda ei saa põhiprogrammis ette näha.

Vektorkatkestuse meetod. Taaskäivituskäsku *RESTART* üleminekuks katkestavale programmile ei moodustata fikseeritud kujul, vaid selle aadressiosa modifitseeritakse andmesiini kaudu sisestava lisainformatsiooni abil. Kui korraldada viimase üksühene vastavus katkestava välisseadmega, siis on identifitseerimisprobleem lahendatud – igale välisseadmele vastab lihtsalt erinev katkestava programmi algaadress. Lisainformatsioon antakse mitmejärgulise kahendsõna ehk -vektori kujul, millest tuleneb vektorkatkestuse meetod.

Vektorkatkestuse meetodit realiseeritakse joonisel 2.30 toodud skeemi kohaselt. Katkestussignaalid INT_1..INT_n lülitatakse prioriteediselektori sisenditele kahanevas järjekorras. Selektori väljundsignaal *INT* saadakse sisendsignaalide loogilisel liitmisel ning lülitatakse protsessori katkestussisendile. Kui protsessor on katkestussignaalile reageerinud, siis teatab ta signaaliga *INTA* (*interrupt asknowledgment*), et signaal on aktsepteeritud ja taaskäivituskäsu aadressi modifitseerimiseks on vaja lisainformatsiooni. Selle signaali toimel rakendub prioriteediselektor ja tekitab väljundsignaali ainult kõige vasakpoolsemale aktiivsele sisendsignaalile vastaval väljundil *V*.

Prioriteediselektori väljundsignaali *V* toimel tekitatakse koodris (K_1...K_k) katkestusvektor, mis väljastatakse andmesiinile. Vastavalt lülitavale koodrile saadakse andmesiinil vektori erinev väärtus (*A, B, ..., Z*), mis paigutatakse käsu *RESTART* aadressiosa vastavasse välja.



Joonis 2.30. Vektorkatkestuse meetod

Programmeeritavad andmevahetuslülitused on ette nähtud andmevahetuse korraldamiseks protsessori ja sisend-väljundseadmete vahel. Programmeeritava monoliitlülitusega sobitatakse välisseadmete andme- ja juhtsignaale arvuti süsteemisiiniga. Andmevahetus arvutiga süsteemi andmesiini kaudu toimub läbi andmesiini puhvri, mis võimaldab andmeid ajutiselt hoida ja neid ühes või teises suunas edastada.

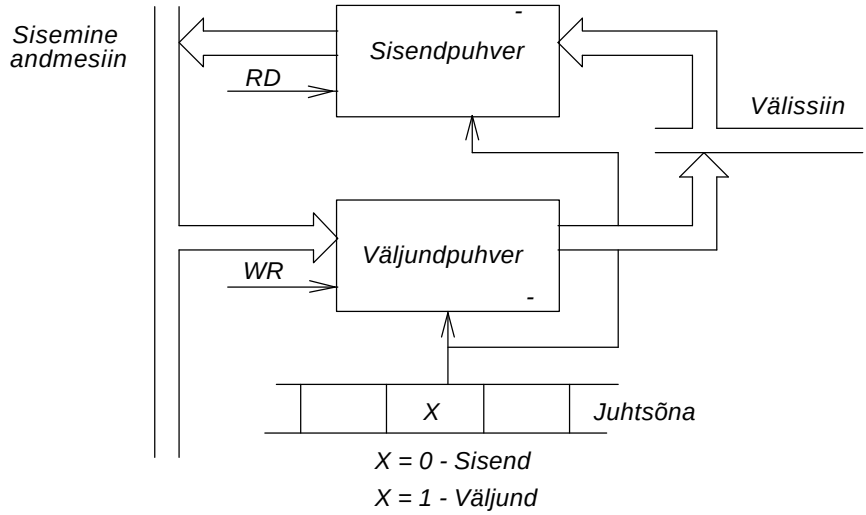
Lülitused konstrueeritakse mitmekanalilistena (l ... n), kusjuures igal kanalil on oma funktsionaalelement, milles toimub tegelik andmevahetus välisseadmetega ning mis on viimastega välissiini kaudu otseselt ühendatud. Levinumad andmevahetuse funktsionaalelemendid on järgmised: 1) rööpvärat andmevahetuseks rööpkoodis; 2) jadavärat andmevahetuseks järjestikkoodis; 3) otsemällupöördumise e *DMA*-kontroller; 4) taimer ajaintervallide genereerimiseks ja 5) klaviatuurikontroller.

2.3.2. Rööpvärat

Rööpvärati lihtsustatud struktuuriskeem on joonisel 2.31. Selle põhiosadeks on sisend- ja väljundpuhvrid (registrid). Mõlemad on sisendite poolel varustatud lukkudega (*latch*); väljunditel on kolm võimalikku olekut. Värat ühendatakse välisseadmega välissiini kaudu, milles on harilikult kaheksa juhet. Teiselt poolt on puhvrid ühendatud sisemiste andmesiinidega, mille kaudu läbi andmesiini puhvri toimub andmevahetus arvutiga. Enne töö alustamist ja pärast toitevoolu väljalülitamist tuleb värat defineerida kas sisend- või väljundseadmeks.

Signaaliga *RD* (*read*) viiakse kiibi sisendpuhver lugemisrež iimi (vastuvõtule). Signaaliga *WR* (*write*) viiakse väljundpuhver olekusse, kus toimub andmete väljastamine arvutist

välisseadmetesse. Signaaliga *RESET* viiakse lülitus algolekusse ning signaaliga *CS* toimub kiibi üldine selekteerimine. Andmete kulgemise suuna määrab ära juhtsõna. Antud juhul 1 bitt. Tegelikes seadmetes võivad väratid olla ehitatud keerukamalt (näiteks pooled välissiini juhtmed töötavad sisenditena ja pooled väljunditena). Sel juhul peab juhtsõna olema pikem kui üks bitt.



Joonis 2.31. Rööpvärati struktuuriskeem

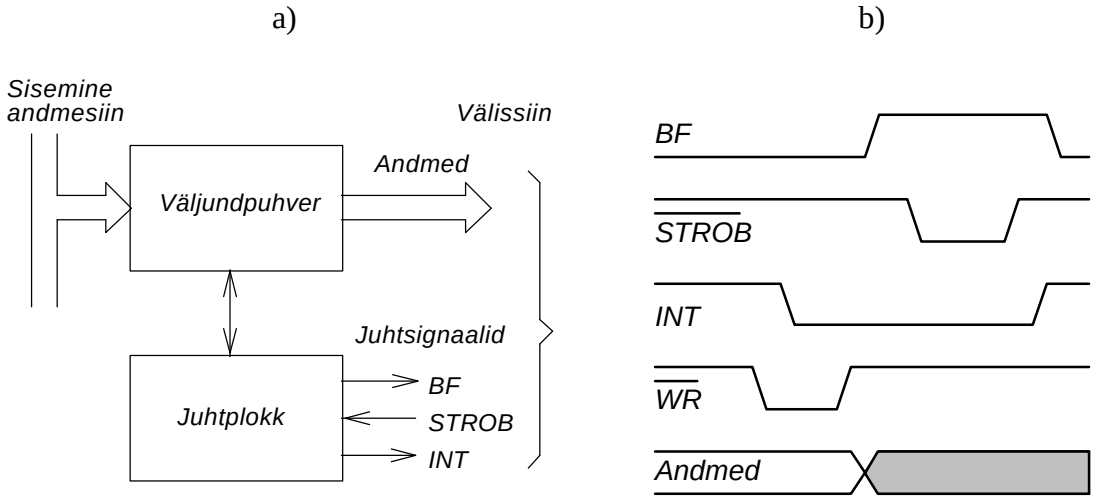
Iga juhtsõna määratlus kehtib kuni selle muutmiseni uue juhtsõna laadimisega või toitevoolu väljalülitamiseni. Et vältida seadme rikkumist värati määratlemata olukorras, mis esineb kohe pärast toitevoolu sisselülitamist, defineeritakse kõik väratid signaali *RESET* toimel automaatselt sisendseadmeteks. Lisaks andmete liikumise suunale on väratis programmeeritav ka töömoodus (põhimoodus ja strobeerimismoodus).

Põhimooduse korral on värat määratletud kas sisend- või väljundliideseks. Väratis toimub andmete lihtne ülekanne ning mingeid juhtsignaale kviteerimiseks ei moodustata. Kui värat on defineeritad sisendseadmeks, siis sisendpuhver on avatud ning registri sisu kooperib pidevalt välissiini signaalide väärtusi. Sisendregistri sisu võib suvalisel hetkel sisestada arvutisse lugemissignaaliga *RD*. Väljundpuhvri väljundid on kõrge impedantsiga olekus. Kui värat on defineeritud väljundseadmeks, siis väljundregistri väljundid on pidevalt avatud välissiinile.Väljundregistri sisu võib suvalisel hetkel muuta arvutist saadetava uue andmesõna abil kirjutussignaali *WR* toimel. Mitmes väratitüübis kopeeritakse väljundregistri sisu ka sisendregistrisse, mis võimaldab seda arvutisse tagasi lugeda.

Strobeerimismooduse korral, mis on vajalik kviteerimismeetodil töötamiseks, moodustatakse välissiinil peale andmekanali veel 3 juhtsignaaliliini. Need saadakse kas andmesiini kitsendamisel 3 juhtme võrra või moodustatakse naabervärati välissiini juhtmetest, kasutades väratite kombineeritud lülitust. Nendeks signaalideks on 1) puhver laaditud *BF* (*buffer full*), mida kasutatakse kviteerimismeetodil töötamisel

küsitlussignaalina, 2) strobeerimissignaal *STROB*, mida kasutatakse vastuvõtja valmisoleku kviteerimiseks ja katkestussignaal *INT*.

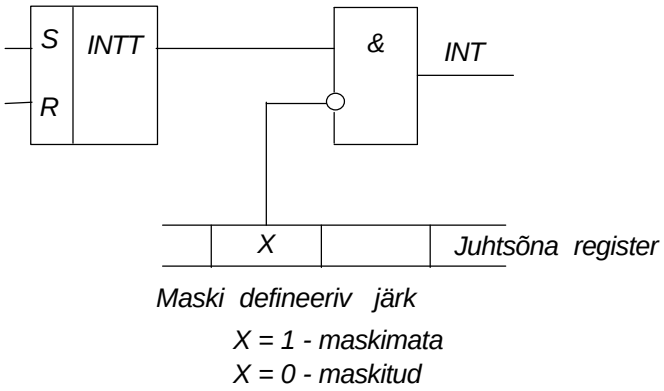
Värati tööd väljundseadmena on näidatud joonisel 2.32.



Joonis 2.32. Rööpvärat väljundseadmena: a) plokkskeem, b) ajadiagramm

Ajadiagrammilt on näha, et jõudeolukorras on puhver tühi ning signaal *BF* madal. Töötsükkel algab andmete väljastusega arvutist väratisse kirjutussignaali *WR* toimel. Viimase langeva frondiga (kirjutamise lõppedes) viiakse signaal *BF* kõrgeks. Vastuvõtja vastab sellele. Signaali lõppemisel tühjendatakse puhver ning antakse välja katkestussignaal, mille abil teatatakse arvutile, et järjekordne andmesaadetis on väljastatud ning alustada võib uut väljastustsüklit.

Maskimine. Katkestussignaali maskimiseks on juhtsõnast eraldatud üks bitt (joonis 2.33). Selle biti väärtuse järgi lubatakse või ei lubata trigeril *INTT* registreeritud katkestussignaal värati väljundisse.

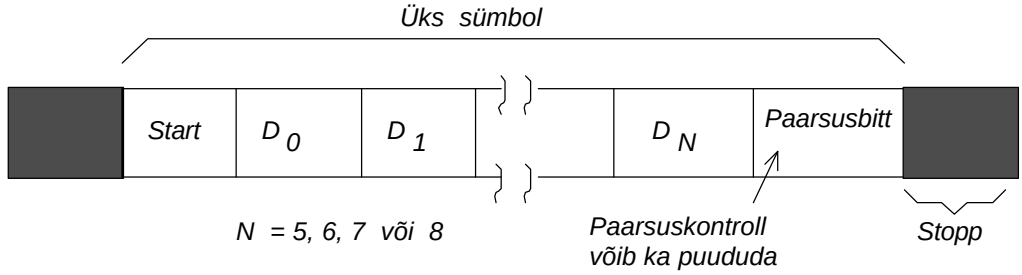


Joonis 2.33. Katkestussignaali maskimine väratis

2.3.3. Jadavärat

Jadavärati abil vahetatakse andmeid välisseadmetega järjestikkoodis. Andmeedastus toimub ühe biti kaupa kahejuhtmelises kanalis. Võimalik on asünkroonne ja sünkroonne töömoodus.

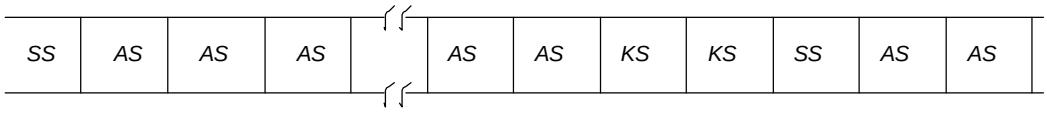
Asünkroonse töömooduse korral algab järjekordse sümboli edastamine juhuslikul hetkel. Sümboli üksikud bitid kantakse üle sünkroonselt kindlate ajavahemike *T* järel. Joonisel 2.34 on osa ühe sümboli edastamise protokollist (saadetise vorming).



Joonis 2.34. Saadetise vorming asünkroonsel järjestikedastusel

Jõudeolukorras on liinis signaali nivoo kõrge. Saadetise alustamiseks väljastab saatja kõigepealt kindla kestusega madala nivooga stardisignaali. Sellele järgnevad andmebitid D_0, D_2....D_n ja mõningates süsteemides ka paarsuskontrolli bitt. Saadetis lõpeb stoppsignaaliga, millel on alati kõrge nivoo. Pärast stoppimpulsi lõppu on süsteem valmis uue saadetise saatmiseks.

Sünkroonne töömoodus eeldab saatja ning vastuvõtja sünkroonset tööd pikema aja vältel. See esitab rangemad nõudmised taktigeneraatoritele ja nõuab erivõtete kasutamist nende sünkroniseerimiseks. Üht võimalikku sünkroonside protokolli illustreerib joonis 2.35.



Joonis 2.35. Sünkroonside protokolli graafiline esitus (saadetise vorming)

Saadetis algab spetsiaalse sünkrosümboliga *SS*, mille äratundmisel vastuvõtja häälestab end vastuvõtu algusesse ja sünkroniseerib taktigeneraatori. Ühele või mitmele eri sünkrosümbolile järgnevad andmesümbolid *AS*, mille pakett võib lõppeda veakontrolli sümbolitega *KS*. Sünkrosümboli kood peab erinema kõigist andmesümbolite koodidest, sest vastasel korral poleks teda võimalik eristada. Sünkroonse töömooduse eeliseks on

suurem andmeedastuse kiirus, sest selle puhul ei kulutata aega stardi- ja stoppsignaalidele iga sümboli alguses ja lõpus.

2.3.4. Taimer

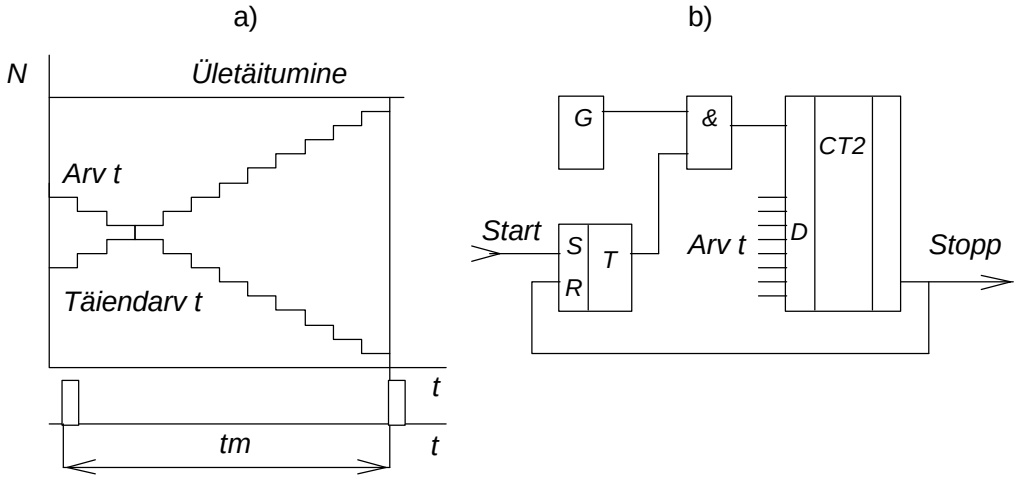
Programmeeritavat taimerit kasutatakse selleks, et genereerida täpseid ajavahemikke ja etteantud kujuga impulsspinget või loendada sündmusi. Kõiki neid tehteid saab sooritada programmi järgi, kuid alati pole see otstarbekas, sest see kulutaks palju protsessori aega ja muudaks tarkvara keerukaks. Näiteks juhul kui protsessor genereerib teatud ajavahemikku, ei saa teda kasutada muudeks teheteks. Seepärast muudab programeeritav taimer arvuti või juhtraali töö ratsionaalsemaks ning võimaldab ühe protsessoriga juhtida korraga mitut protsessi või seadet.

Pooljuhtmuundurite tüürimisel on peamine etteantud viivituse, kestuse ja sagedusega tüürimpulsside moodustamine. Vajalik lähteinfo saadakse arvutilt, mis väljastab muunduri juhtimiseks kahendkoodis juhtsõna. Türistormuundurites alustatakse viivituse moodustamist hetkel, mil võrgupinge läbib nulli. Sel momendil tekitab sünkroniseerimislülitus impulsi, mis käivitab türistoride tüürnurka moodustava taimeri. Viivituse lõppemisel moodustatakse tüürimpulss, mille toimel türistor avaneb. Mitme türistoriga, mitmefaasilistes alaldites ja pingeregulaatorites tuleb moodustada tüürimpulsid kõigi türistoride jaoks. Samal põhimõttel töötavad ka jõutransistoridel laius-impulssmuundurid.

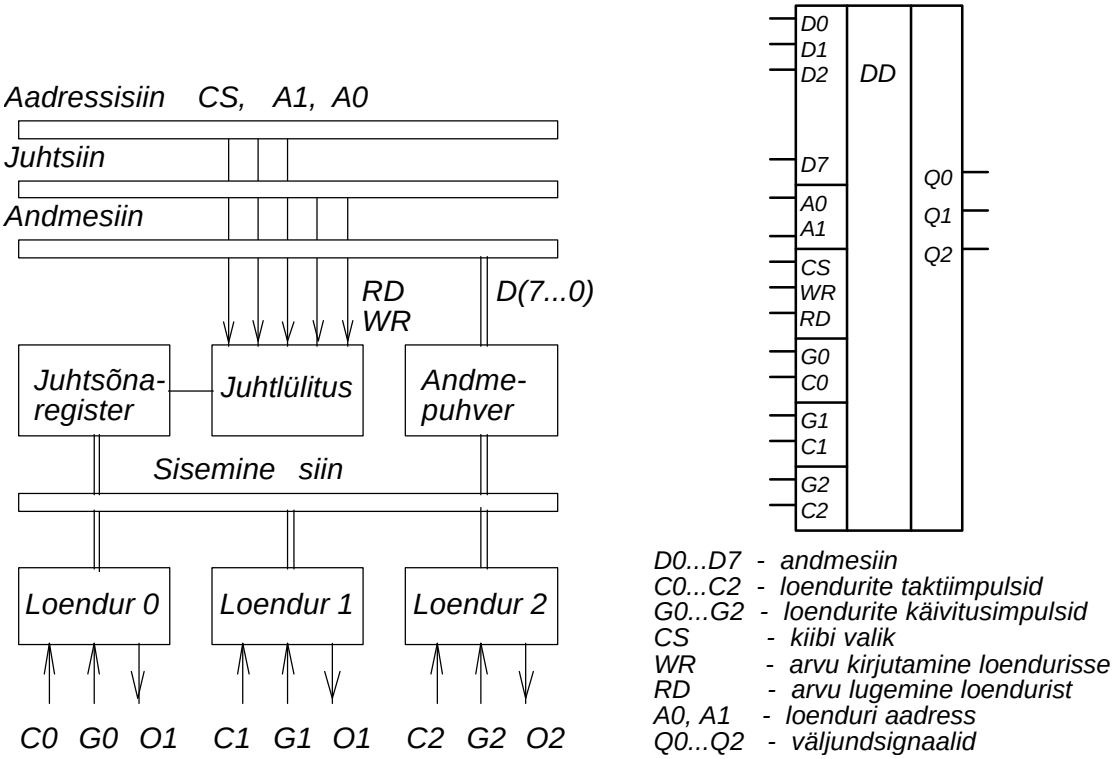
Järgnevalt vaadeldakse viivitust moodustava liidese tööpõhimõtet. Juhtsõnale vastav kahendkood muundatakse ajaintervalliks kahendloenduritega (joonis 2.36). Ajaühikuks on impulsigeneraatori impulsside periood, mis peab olema konstantne. Praktikas saavutatakse see täpsete kvartsresonaatorite, näiteks arvuti taktigeneraatori abil. Viivituse moodustamine algab käsuga *START*. Sõltuvalt pooljuhtmuunduri liigist saadakse see käsk võrgupinge nullpunkti registreerivalt sünkroniseerimislülituselt või muunduri tööperioodi algust määravalt autonoomselt taktigeneraatorilt. Käsuga *START* viiakse trigeri väljund olekusse 1. Koos sellega avab NING-element generaatori impulssidele pääsu loenduri sisendisse. Impulsside loendamiseks kasutatakse programmeeritavat kahendloendurit, mille väljundkood sõltub nii loendatud impulsside arvust kui ka eelnevalt loendurisse sisestatud lähtekoodist, mis määrab viivituse suuruse. Impulsse võib loendada nullist kuni lähtekoodiga etteantud arvuni või etteantud arvust kuni loenduri täieliku tühjenemiseni, s. o impulsside tagurpidi loendamisega. Lähtekoodi asemel võib aga kasutada ka selle täiendkoodi ning loendada impulsse, alates täiendkoodist kuni loenduri ületäitumiseni (joonis 2.36, a).

Viivituse moodustamine lõpeb loenduri tühjenemisel või ületäitmisel väljastatava *STOPP*-impulsiga, mis antakse trigeri sisendisse *R*. Trigeri väljund muutub nulliks ja generaatori impulsid ei pääse enam läbi NING-elemendi loenduri sisendisse. Seega on ajaintervall määratud kahe impulsi *START* ja *STOPP* vahelise perioodiga. *STOPP*-impulss on ühtlasi taimeri väljundsignaaliks, mis pärast võimendamist avab türistori või juhib transistorkommutaatori tööd.

Mitme ventiiliga muundurite juhtimiseks on otstarbekas kasutada mitmekanalilisi programmeeritavaid taimereid. Viimaseid toodetakse mikroprotsessorisarja integraallülitustena. Üheks tüüpiliseks programmeeritavaks taimeriks on näiteks integraallülitus 8253. Taimeril on kolm paralleelset kanalit, mis võimaldavad juhtida kolme erinevat ventiili. Kõik kanalid on üksteisest sõltumatult programmiga juhitavad. Taimeri 8253 struktuuriskeem ja selle skeemitähis on joonisel 2.37.



Joonis 2.36. Taimeri tööpõhimõte: a) loenduri töö, b) plokkskeem



Joonis 2.37. Programmeeritav taimer: a) plokkskeem, b) tingmärk

Programmeeritav taimer koosneb kolmest ühesugusest 16-bitisest loendurist, mis loendavad arve kahend- või kahend-kümnendkoodis. Infovahetus sisendi ja loendurite vahel toimub taimeri sisemise andmesiini kaudu. Loendatavad impulsid antakse sisenditesse *C*0...*C*2. Loendurid käivitatakse impulssidega *G*0...*G*2 ning impulsse loendatakse kahanevas järjekorras, s. o ülalt alla. Taimer võib töötada kuues erinevas rež iimis, mida nimetatakse töömoodusteks.

0. töömooduse korral on tegemist loendamise katkestusrež iimiga. Valitud kanali väljundis moodustub pärast loenduri tühjakslugemist signaal 1. Signaalidega *G*0...*G*2 saab loendureid käivitada, loendamist katkestada ja loendamist jätkata. Loenduri laadimine (programmeerimine) uue arvuga võib toimuda loendamise ajal. Ümberlaadimise ajaks katkestab loendur töö, alustades seda taas pärast uue arvu vanema baidi laadimist.

1. töömooduse korral moodustab loendur, mis töötab nagu ootemultivibraator, väljundis madala nivooga 0 impulsi, mille kestus võrdub loendurisse kirjutatud arvu ja taktiimpulsside perioodi korrutisega. Multivibraator käivitatakse signaalidega *G*0...*G*2. Loenduri ümberlaadimine loendamise ajal ei muuda jooksva loendamise kulgu.

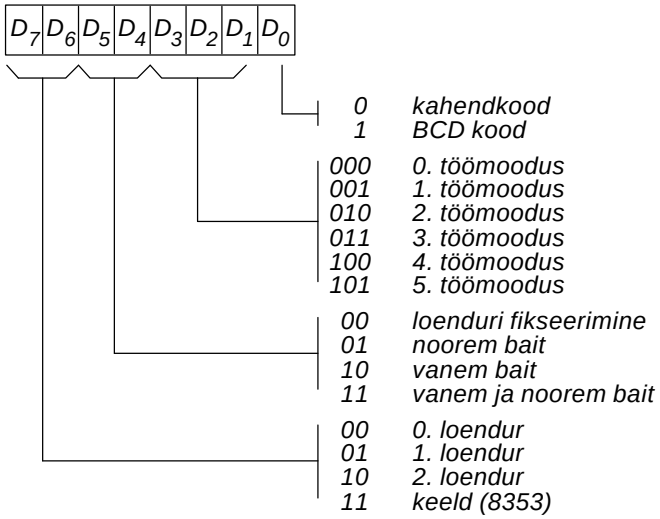
2. töömooduse korral töötab mikrolülitus nagu impulsigeneraator. Loendur jagab sisendimpulsside sageduse sinna salvestatud arvuga *n*. Kõrge nivooga signaali 1 kestus on (n-1) τ ja madala nivooga signaali 0 kestus võrdub τ, kus τ on sisendimpulsside periood.

3. töömooduse korral töötab mikrolülitus samuti nagu impulsigeneraator, mis jagab sisendsageduse arvuga *n*. Impulsside poolperioodid on paarisarvulise *n* korral võrdsed τ (n/2); paarituarvulise *n* korral on kõrge nivooga poolperioodi kestus τ (n+1)/2 ja madala nivooga poolperioodi kestus τ (n-1)/2.

4. töömooduse korral töötab mikrolülitus nagu programmiga juhitav taimer. Pärast etteantud arvu loendamist moodustatakse üksik madala nivooga impulss, mille kestus võrdub taktiimpulsside perioodiga τ. Signaale *G*0...*G*2 kasutatakse nii nagu 0. töömooduse korral. Samuti toimub loenduri ümberlaadimine.

5. töömooduse korral töötab mikrolülitus nagu aparatuuriga juhitav taimer. Pärast etteantud arvu loendamist moodustatakse madala nivooga impulss, mille kestus võrdub τ. Seni kuni loendamine pole lõppenud, käivitab signaalide *G*0...*G*2 iga positiivne impulss loenduri uuesti.

Taimerit juhitakse arvuti andmesiini kaudu edastatava käsusõnaga ja juhtsiinilt antud signaalidega *RD* ning *WR*. Loendurite aadressisisendid *A*0, *A*1 ja kiibi valiku sisend *CS* ühendatakse arvuti aadressisiiniga. Selleks võib kasutada aadressisõna noorema või vanema baidi suvalisi bitte. Käsusõna bittide tähendus on joonisel 2.38. Käsusõna kõige noorema bitiga valitakse loendatavate arvude kood. Bitid D_1, D_2, ja D_3 on ette nähtud töömooduse valikus; D_4 ja D_5 määravad loenduri baitide poole pöödrumise viisi, D_6 ja D_7 näitavad kanali kaudset aadressi, s. t kanali numbrit, kuhu tuleb salvestada käsusõna ülejäänud järgud. Näiteks võib tuua käsusõna 7*A*, mille kahendkoodiks on 0111 1010. Vastavalt sellele on programmeerimiseks valitud taimeri 1. kanal, kus toimub kahe baidi järjestikku salvestamine, töötatakse 5. töömooduse järgi ning impulsse loendatakse kahendkoodis. Taimeri juht- ja aadressisignaalidest annab ülevaate tabel 2.4.



Joonis 2.38. Programmeeritava taimeri käsusõna

Tabel 2.4

CS	RD	WR	A1	A0	Taimeri tööoperatsioon
0	1	0	0	0	0. loenduri laadimine
0	1	0	0	1	1. loenduri laadimine
0	1	0	1	0	2. loenduri laadimine
0	1	0	1	1	Käsusõna salvestamine
0	0	1	0	0	0. loenduri lugemine
0	0	1	0	1	1. loenduri lugemine
0	0	1	1	0	2. loenduri lugemine

0	0	1	1	1	Kanalid on suletud
1	x	x	x	x	Kanalid on suletud
0	1	1	x	x	Kanalid on suletud

Juhtimiseks kasutatakse inverteeritud signaale, mille korral loetakse aktiivseks madalat ehk 0 nivood. Käsusõna salvestatakse aadressi *A*0 ja *A*1. Signaali *CS* 1 korral taimer ei tööta ja tema kanalid on suletud.

Taimerit juhib arvuti. Käsusõna ja loendatavate arvude koodid viiakse taimerisse programmiga. Niisuguse programmi näide on tabelis 2.5. Taimeri adresseerimiseks kasutatakse arvuti aadressisiini vanemat baiti Loendurite aadressideks on sel juhul *E0FF*, *E1FF* ja *E2FF* ning käsusõna aadressiks *E3FF*. Kanalitesse antavad käsusõnad on oma sisult ühesugused ja erinevad üksnes kanali aadressi poolest. Käsusõnade 3*A*, 7*A* ja *BA* kohaselt salvestatakse kõikidesse loenduritesse kaks järjestikust baiti, kõik loendurid töötavad 5. töömooduse järgi ning loendavad arve kahendkoodis. Kanalitesse salvestatud arvud on toodud näites samuti ühesugused ning asuvad arvuti protsessori registripaaris *B&C*.

Muunduri erinevate kanalite töö on ajas nihutatud. Seepärast peavad ka taimeri loendureid käivitavad impulsid sisendites *G*0...*G*2 olema ajas nihutatud. Tavaliselt saavutatakse see igale kanalile vastava eraldi sünkroniseerimislülituse abil. Türistoride tüürnurga muutmiseks tuleb muuta registripaari *B&C* sisu.

Kõiki programmeeritava taimeri kanaleid saab üksteisest sõltumatult programmi abil juhtida. Vastavate programmidega saab realiseerida juhtalgoritme, mille aparatuursed lahendused on väga keerukad ning kallid. Näiteks võib tuua ilma alalisvoolu vahelülitita sagedusmuunduri juhtimise. Juhitavate ventiilide arv nendes muundurites on suur, ventiilide kommuteerimise seaduspärasus aga keerukas. Nendel põhjustel pole aparatuuriga juhitavad ilma alalisvoolu vahelülita sagedusmuundurid (tsüklokonverterid) leidnud seni laiemat kasutust. Raaljuhtimise korral osutub nende kasutuselevõtt tehniliselt ja majanduslikult põhjendatuks.

Tabel 2.5

Taimeri laadimisprogramm

Käsu aadress mälus	Käsu mnemokood	Käsu masinakood	Selgitus
1000	LXI B 03 A1	01 A1 03	Registripaari B&C salvestatakse loendatav arv 03A1
1003	LXI H E3 FF	21 FF E3	Valitakse taimeri käsusõna aadress
1006	MVI M 3A	36 3A	0. kanali käsusõna
1008	MVI M 7A	36 7A	1. kanali käsusõna
100A	MVI M BA	36 BA	2. kanali käsusõna
100C	DCR H	25	Valitakse taimeri 2. kanali aadress, mille vanem bait erineb käsusõna aadressi vanemast baidist ühe võrra

100D	MOV M, C	71	2. kanali loendurisse salvestatakse registris C sisalduv bait A1
100E	MOV M, B	70	2. kanali loendurisse salvestatakse registris B sisalduv bait 03
100F	DCR H	25	Valitakse taimeri 1. kanali aadress
1010	MOV M, C	71	1. kanali loendurisse salvestatakse registris C sisalduv bait A1
1011	MOV M, B	70	1. kanali loendurisse salvestatakse registris B sisalduv bait 03
1012	DCR H	25	Valitakse taimeri 0. kanali aadress
1013	MOV M, C	71	0. kanali loendurisse salvestatakse registris C sisalduv bait A1
1014	MOV M, B	70	0. kanali loendurisse salvestatakse registris B sisalduv bait 03

2.3.5. Otsemällupöördus ja *DMA*-kontroller

Otsemälukanali (*DMA - direct memory access)* kontroller on ette nähtud otseside loomiseks andmeallika ja tarbija vahel ning andmete plokiviisi edastamiseks maksimaalse võimaliku kiirusega. Andmeallikaks ja vastuvõtjaks võib olla nii mälu kui ka välisseade. Vastavalt sellele luuakse järgmised edastusvõimalused:

- välisseadmest mällu,
- mälust välisseadmesse,
- ühest välisseadmest teise,
- ühest mälu osast teise.

Sageli on andmeid vaja edastada sõltuvalt nende sisust, massiivi nimetusest või mingist koodist. Seepärast on enamikus *DMA*-kontrollerites otsinguvõimalus, s.t võimalus andmeid ükshaaval läbi vaadata, kuni on leitud mingi tunnus. Võimalikud töömoodused on järgmised:

- ainult andmeedastus,
- ainult tunnuse järgi otsing,
- otsing ja edastus.

Andmete liikumise suund määratakse andmete algusaadressidega, sest aadressidega on nii mäluosad kui ka välisseadmed üheselt määratud. Kontrolleri ülesandeks on peale andmeedastuse ka vajalike aadresside formeerimine. Andmeallika jooksvat aadressi nimetatakse allika-aadressiks, vastuvõtja vastavat aadressi aga sihtaadressiks.

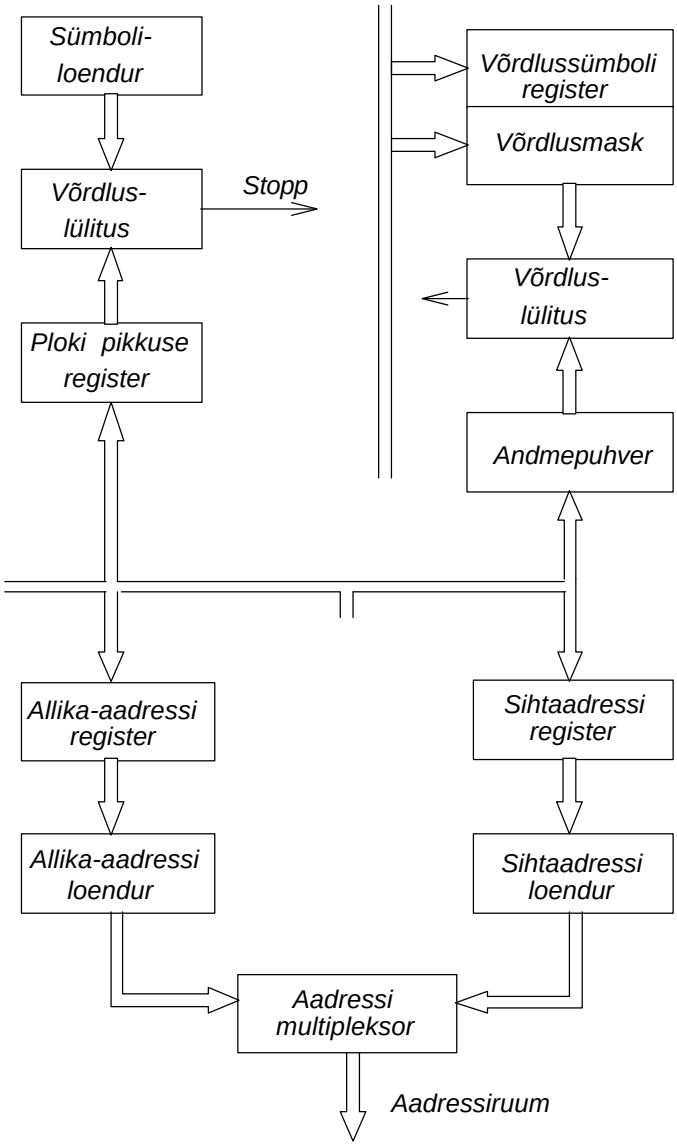
Seadme lihtsustatud struktuuriskeem on joonisel 2.39. See koosneb kõigepealt allika- ja sihtaadressi registritest, kuhu töötsükli alguses laaditakse algaadressid. Mõlema registri juurde kuuluvad loendurid, mille sisu tsükli alguses võetakse aadressiregistritest ja mis suurenevad iga sümboli edastamise järel ühe võrra.

Jooksev aadress moodustub alati loenduritel. Et seade töötab ühe aadressisiiniga, mille kaudu antakse vaheldumisi üle allika- ja sihtaadresse, on kasutusel aadressi multipleksor.

Edastatud sümbolite arvu kontrolliks on skeemis ploki pikkuse register, sümboliloendur ja võrdluslülitus. Tsükli alguses laaditakse etteantud ploki pikkus registrisse ning loendur nullitakse. Pärast sümboli edastamist suurendatakse loenduri sisu ühe võrra niikaua, kuni see võrdub ploki pikkusega. Selle momendi fikseerib võrdluslülitus, mis annab ka käsu töö lõpetamiseks.

Tööks otsingurežii mis on seadmes võrdlussümboli register koos programmeeritava maskiga, andmepuhver ja vastav võrdluslülitus. Nende abil kontrollitakse, kas loetud sümbol ühtib võrdlussümboliga, mis laaditi registrisse tsükli alguses. Maski abil, mida samuti defineeritakse programmiga, saab osa bitte võrdlusprotsessist kõrvaldada. See võimaldab otsingut koodi mingi osa järgi.

Süsteem on universaalne, selle abil luuakse otseside mitmesuguste seadmete vahel.



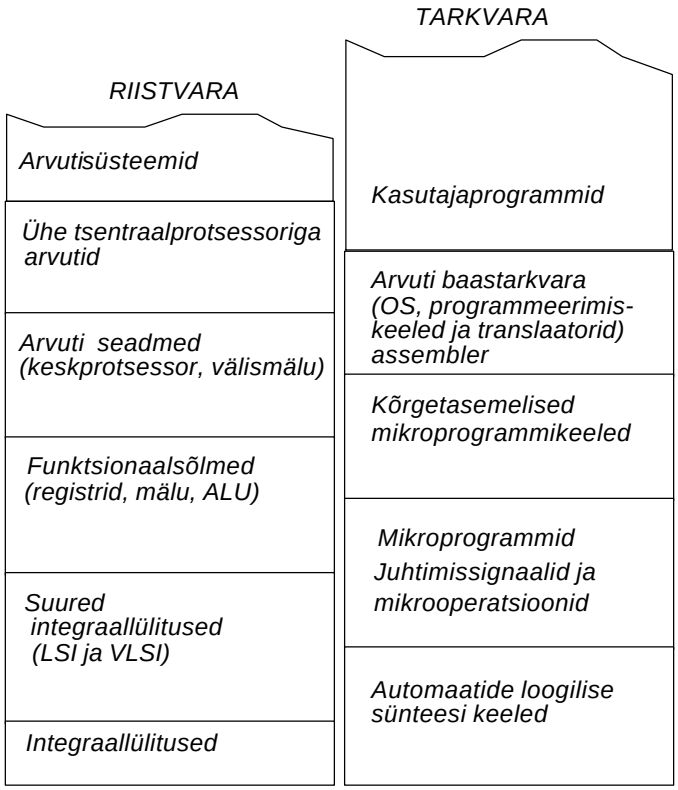
Joonis 2.39. Otsemälukanali (*DMA*) kontrolleri struktuuriskeem

2.4. Tarkvara

2.4.1. Ülevaade mikroarvutite ja juhtraalide tarkvarast

Arvuti tarkvara moodustab hierarhilise süsteemi, mille alumised tasandid toetuvad riistvarale, ülemised tasandid puutuvad aga kokku arvuti kasutajaga. Tarkvarapüramiid kasvab pidevalt, sest arvuti kasutajaid huvitab, et tarkvara arvestaks inimese tavapärast suhtlemisviisi: arendaks dialoogi, kasutaks inimesele harjumuspäraseid sümboleid, kõnet jms. Nüüdisaegsed arvutid õpivad tuvastama teksti, kujundeid, kõnet ning teevad palju muud, mida seni oskas vaid inimene. Luuakse tehisintellektiga arvuteid ja vastavat tarkvara. Tänapäeva arvutitehnika uurimisvaldkonnast moodustavad enamiku tarkvara probleemid. Nendesse süüvimine pole aga selle raamatu ülesanne. Siin käsitletakse seda osa tarkvarast, mis on vahetult seotud arvuti riistvaraga ning mikroprotsessoritega.

Arvuti riistvara ülesehitus on samuti hierarhiline, kusjuures igale riistvaratasandile vastab temaga koos töötav tarkvaratasand. Hierarhilisest riistvara-tarkvarasüsteemist annab ülevaate joonis 2.40. Märkigem, et joonisel on kujutatud vaid seda osa tarkvarast, mis puutub vahetult kokku riistvaraga.



Joonis. 2.40. Hierarhilised riistvara-tarkvaratasandid

Kõige alumiseks tarkvaratasandiks, mis võimaldab programmeerida arvutit kui tervikut, on **masinakoodide ja assembleri** tasand. Sellest allapoole jäävad tasandid nagu mikroprogrammeerimiskeeled või automaatide loogilise sünteesi keeled on ette nähtud arvuti üksikkomponentide väljaarendamiseks. Assemblerist ülespool on arvuti operatsioonisüsteem, translaatorid, kõrgprogrammeerimiskeeled ning kasutajaprogrammid.

2.4.2. Assembler

Assemblerikeelt on vaja tunda siis, kui puudub kõrgkeele translaator, mis avastaks programmis oleva vea, mida kõrgkeele tasemel ei õnnestu avastada. Mikroprotsessor-süsteemide loomisel ning juhtraalide kasutamisel tööstusseadmete ja -protsesside juhtimisel tuleb seda tihti ette.

Kõrgkeele näiteks *Basic*'u või *Pascal*'i programmi tõlgib assemblerikeelde või masinakoodi kõrgkeelde translaator. **Assemblerikeeles** programmeerimine sarnaneb masinakoodis programmeerimisega, kuid on mõnevõrra mugavam. **Masinakoodis** programmeerimisel kasutatakse absoluutaadresse, mis eeldab, et programmeerija tunneb täpselt infobittide asukohta (aadresse) mälus. Assemblerikeeles kasutatakse sümbol- ja suhtadresseerimist. **Sümboladresseerimise** korral antakse muutujale või käsule nimi, nn märgend, ja edaspidi opereeritakse selle nimega, mitte absoluutse aadressiga. Transleerimisel seatakse nimega vastavusse aadress. **Suhtadresseerimisel** ei määrata aadresse mitte mälu alguse suhtes (absoluutaadressidena), vaid mingi kokkulepitud baasi, näiteks programmi alguse suhtes. Siirdekäskude puhul on baasiks siirdekäsk ise (näiteks, siirduda 8 käsku edasi, siirduda 4 käsku tagasi). Assemblerikeel ja masinakood erinevad ka selle poolest, et transleerimise käigus on translaator võimeline avastama mitut liiki vigu.

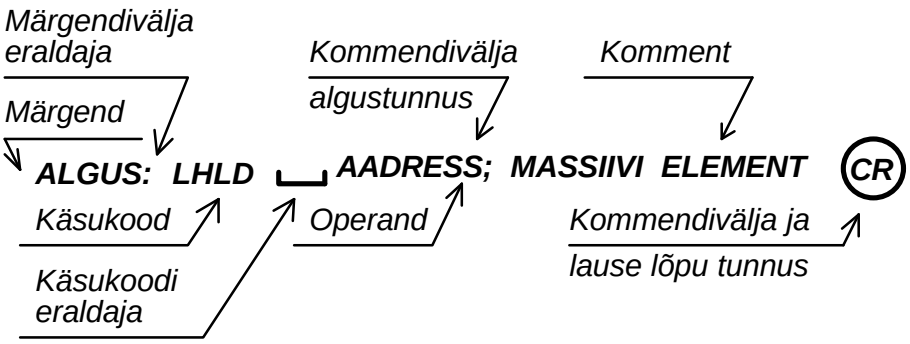
Assembleri translaator avastab 1) kirjavigu, kui nende tagajärjel tekivad keelatud sümbolid või koodid, 2) korduvalt määratud märgendeid, 3) puuduvaid märgendeid, millele on programmis viidatud, 4) adresseerimisvigu. Vigadele reageerimise võime ja viis sõltub assembleri translaatorist.

Assemblerikeeled on kindla vorminguga. Sisestatav tekst koosneb **lausetest**, iga lause on jagatud **väljadeks** [1]. Välja tüübid (joonis 2.41) on järgmised:

- **märgendiväli,**
- **käsukoodiväli,**
- **operandiväli,**
- **kommentaari- ehk kommendiväli.**

Märgendiväli võib sisaldada erineva pikkusega märgendi. Märgendiväli lõpeb eraldajaga, milleks kasutatakse mingit sümbolit. Kui esimene sümbol märgendiväljas on tühik, loetakse märgendivälja tavaliselt tühjaks.

Käsukoodiväli sisaldab masinakäsu mnemokoodi või pseudokäsu, mis iseloomustab konkreetset assemblerikeelt. Käsukoodivälja lõpu tunnuseks on tavaliselt tühik.



Joonis 2.41. Assemblerikeelse lause struktuur

Operandiväli sisaldab järgmist informatsiooni:

- **operandide nimed** (märgendiviited),
- **arvväärtused** (vahetu adresseerimine korral),
- **avaldised**.

Avaldisi defineeritakse iga konkreetse assemblerikeele puhul eraldi. Tavaliselt sisaldavad nad märgendeid, konstante ja mitmesuguseid aritmeetika- ning loogikatehteid.

Kommendiväli algab kokkulepitud sümboliga, millele võib kuni rea lõpuni järgneda suvaline tekst. Lause ja rea lõpetab tavaliselt reavahetusmärk.

Direktiivid võimaldavad lihtsustada assemblerikeeles programmeerimist võrreldes masinakoodiga. Kasutusala järgi on pseudokäsud järgmised:

1. **Nimede defineerimise ja ümberdefineerimise käsud** näiteks EQU.

```
UMRK: EQU VMRK+1AH
```

Käsu tulemusena omistatakse märgendile UMRK väärtus, mis saadakse $1A_{16}$ liitmisega märgendi VMRK väärtusele.

2. **Andmetüüpide kirjeldamise käsud** näiteks DB.

```
ARV : DB 11D
ARV1: DB ARV +1
```

Märgendile ARV vastab edaspidi ühebaidine muutuja, mille väärtus on 11_{10}. Märgendiga ARV1 tähistatud muutuja väärtus on edaspidi 12_{10}.

3. **Mälu reserveerimise käsud** näiteks DS.

```
MASS: DS ARV1 + 18D
```

Ülalkirjeldatud lause tulemusena reserveeritakse mälus koht 30-baidise massiivi jaoks.

4. **Käsud transleerimise juhtimiseks** näiteks ORG ja END.

```
ORG UMRK + 70H
```

Taolisele käsule järgnev programmiosa paigutatakse mällu, alates avaldisega UMRK + 70H määratud aadressist.

5. **Makrokäsud.** Makrokäsk on selline pseudokäsk, mille asemele translaator kirjutab terve jada tavalisi assemblerikäske. Iga makrokäsk tuleb enne kasutamist kirjeldada. Näiteks

```
        MACRO SUM A, B, C
        ...
Assemblerikeelne programm, mis arvutab  A+B = C
        ...
        ENDMACRO
```

Kirjeldus peab olema kahe vastava pseudokäsu antud juhul MACRO ja ENDMACRO vahel. Nii saab defineerida uue pseudokäsu, mille nimi on SUM ja millel on kolm parameetrit (operandi).

Assemblerikeeles kirjutatud programmi eelised võrreldes kõrgkeeles kirjutatud programmiga on järgmised:

1. Programmi kõik osad on programmeerijale kättesaadavad. Viga otsides võib minna kõige pisema detailini. Kõrgkeeles kirjutatud programmis on palju transleerimisel lisatavaid tüüpelemente, mida programmeerija ei kujuta selgesti ette.
2. Arvuti võimalused kasutatakse täielikumalt ära. Programm arvestab arvuti iseärasusi. Assemblerikeelt kasutades saadakse umbes 20 % tõhusam programm kui kõrgkeele korral.

Kahjuks on assemblerikeeles programmeerimise korral töömahukus 3...5 korda suurem kui kõrgkeeles programmeerimisel.

Assemblerikeelt ja masinakoodi peab tundma iga mikroarvuti programmeerija.

2.4.3. Intel 8080 assemblerikeel

Assemblerikeele (ASSM) tähestik koosneb järgmistest märkidest:

1) ladina tähestiku suurtähed A...Z,
2) kümnendnumbrid 0...9,
3) aritmeetikatehete märgid, eraldajad ja erimärgid + - : ( ) ' ? tühik. Peale selle võib kommentaarides kasutada suvalisi märke, sealhulgas ka vene tähestiku tähti.

Assemblerikeelne lähtemoodul koosneb **käskudest**, **direktiividest** ja **assembleri-korraldustest**. Käskude kirjutamisel kasutatakse masinakäskude mnemokoode. Direktiive kasutatakse konstantide ja identifikaatorite kirjeldamiseks, mälupiirkondade reserveerimiseks, aadressiloenduri väärtustamiseks, lähteteksti ning mitut lähtemoodulit

sisaldava faili lõpu määramiseks. Lähtetekst võib sisaldada ka kommentaare. Assemblerikorraldused ei kuulu assemblerikeele koosseisu ja nad on ette nähtud assembleerimise juhtimiseks. ASSM-keele käsuvormingu eripära on, et ühe käsu või direktiivi mitme nimega märgendamisel peavad märgendid olema eri ridades näiteks:

```
LABELA:                 ;ESIMENE NIMI
LABELB: MOV B,M         ;TEINE NIMI
```

Käsule MOV B,M võib osutada nii märgendiga LABELA kui märgendiga LABELB.

ASSM-keele **identifikaatorid** koosnevad tähtedest ja numbritest, kusjuures esikohal peab olema täht. Identifikaatori pikkus pole piiratud, kuid assembler arvestab unikaalsuse määramisel vaid esimest viit sümbolit. Transleerimisel võib assembler omistada identifikaatorile väärtuseks mingi mäluaadressi. Sellist väärtustatud identifikaatorit nimetatakse defineeritud nimeks. Identifikaatorit, mis jääb transleerimisel väärtustamata, nimetatakse mittedefineeritud nimeks. Nii käskudes kui ka direktiivides kasutatakse avaldisi. Avaldis moodustatakse termidest + ja – operaatorite abil.

Termideks on täisarvud, sümbolnimed ja ülakomadega eraldatud märgijadad. Arvude kirjutamisel võib kasutada 2nd-, 8nd-, 10nd- ja 16-ndsüsteemi. Kahendarvu tunnuseks on arvu lõpus olev täht *B*, lubatud on numbrid 0 ja 1. 8-ndsüsteemi arvude kirjutamisel kasutatakse numbreid 0,1,...,7, numbrile lisatakse tunnusena 0 või *Q* näiteks 231*Q*. Lõputunnuseta arve loetakse kümnendarvudeks. 16-ndarv kirjutatakse numbrite 0,1,...,9 ja ladina tähtede *A, B, C, D, E, F* abil, kusjuures arv peab algama numbriga (märki arvestamata) ja lõppema tunnusega *H* näiteks *1A4H* või *0B7H*. Arvule võib eelneda kas märk + või - , negatiivsete arvude kujutamisel kasutatakse täiendkoodi.

Arv võib enda alla võtta ühe või kaks baiti. Esimesel juhul on märgiga arvude kujutamise piirkonnaks -128... +127, märgita arvudel 0...255. Teisel juhul võib märgiga arv olla piirkonnas -32768... +32767, märgita arv aga piirkonnas 0...65535. Selle eest et kasutatavad arvud oleksid toodud piirides, peab hoolitsema programmist. Ülakomadega eraldatud märgijada, mis võib olla avaldise elemendiks, interpreteeritakse ASSM-keeles kui konstanti ja teisendatakse koodile ASCII vastavaks arvuks. Näiteks kui mälusõna pikkus on kaks baiti, siis konstant '*A*' teisendatakse 16-ndarvuks 0041, kus 41 on sümboli *A* kood 16-ndsüsteemis.

Direktiivid. ASSM-keele direktiivid on tabelis 2.6

Tabel 2.6

Direktiivi mnemokood	Ülesanne
ORG	Aadressiloendurile väärtuse omistamine
DS	Mälupiirkonna reserveerimine
DB	Baitide defineerimine
DW	Sõnade defineerimine
EQU	Identifikaatorile väärtuse omistamine
END	Lähtemooduli lõpp

ORG-direktiiv võimaldab omistada väärtusi aadressiloendurile. Omistatavaks väärtuseks on direktiivi operandiks oleva avaldise väärtus. Lähtemoodul algab tavaliselt direktiiviga ORG (või kommentaariga, millele järgneb direktiiv ORG). Transleerimisel saadud objektimoodul laaditakse enne täitmist operatiivmällu alates direktiiviga ORG antud aadressist. Näiteks direktiiv

ORG 300H

omistab aadressiloendurile väärtuseks 16-ndarvu 300.

DS-direktiivi kasutatakse mälupiirkonna reserveerimiseks ja selle märgendamiseks. Direktiivi operandiks on avaldis, mille väärtus määrab reserveeritava mälupiirkonna pikkuse baitides. Näiteks direktiiv

PIIRK: DS 256

reserveerib mälupiirkonna 256 baiti. Sellele piirkonnale võib osutada märgendiga PIIRK.

DB- ja DW-direktiivid. Esimene neist võimaldab salvestada konstante järjestikku baitidesse, kusjuures iga andmeelement salvestatakse ühte baiti. DB-direktiivi operandiks on list, mille elemendid on eraldatud komadega. Listi elementideks on avaldised, mis lihtsaimal juhul võivad olla arvud, identifikaatorid või ülakomade vahel oleva koodi ASCII märgijadad. Näiteks direktiiv

LABEL5: DB 220Q, 46, 'CONTROL', 2CH

salvestab kümnesse järjestikku baiti järgmised andmed:
8-ndarvu 220, 10-ndarvu 46, sõna CONTROL, (seitse tähte koodis ASCII) ja 16-ndarvu 2C. Seejuures salvestatakse arv 220 antud mälupiirkonna madalamasse ning arv 2C kõrgemasse baiti. DW-direktiiv erineb DB-direktiivist vaid selle poolest, et salvestab listi elemendid kahebaidistesse sõnadesse. Näiteks direktiiv

DATA: DW 30, TAPE, 126Q

salvestab kolme järjestikusesse sõnasse 10-ndarvu 30, programmielemendi (näiteks mingi teise lause) aadressi nimega TAPE ja 8-ndarvu 126.

EQU-direktiivi kasutatakse märgendiväljas olevale nimele väärtuse omistamiseks. Siin ei eraldata märgendiväljas olevat identifikaatorit mnemokoodist EQU kooloniga. Näiteks direktiivide

DOS EQU 1000H

RAM EQU DOS+12H

abil antakse identifikaatoritele DOS ja RAM väärtuseks 16-ndarvud 1000 ja 1012.

END-direktiiv määrab lähtemooduli lõpu.

Transleerimisel võib assembler kasutada järgmisi faile:

lähtefail (sisaldab lähteteksti),
objektifail (objektiprogrammi salvestamiseks),
listingufail (transleerimislistingu väljastamiseks),

Lähtefailid transleeritakse objektifailideks. Transleerimise tulemusi saab jälgida listingufailist. Objektifailid tuleb seejärel linkida, et saada käivitatav programm.

Näitena on toodud lihtsa ASSM-keelse programmi listing.

Näide

Järgnev assemblerikeeles kirjutatud programm lahutab ühe 16-kohalise kümnendarvu teisest, kasutades järgmisi eeldusi:

- lahutatav arv on salvestatud, noorem bait eespool, alates mälupesast MINU,
- lahutaja on salvestatud, noorem bait eespool, alates mälupesast SBTRA,
- tulemus salvestatakse, noorem bait eespool, lahutatava kohale.

Märgend	Kood	Operand	Märkused
DSUB:	LXI	D, MINU	D-s ja E-s on lahutatava aadress
	LXI	H, SBTRA	H-is ja L-is on lahutaja aadress
	MVI	C, 8	Tsükliloendur
	STC		Ülekandelipp seatakse nii, et see näitaks laenu puudumist
LOOP:	MVI	A, 99H	Akumulaatorisse viiakse arv 99H
	ACI	0	Liidetakse 0 ja ülekanne
	SUB	M	Saadakse lahutaja täiend
	XCHG		Vahetatakse D&E ja H&L registripaaride sisu
	ADD	M	Liidetakse lahutatav
	DAA		Saadakse arvu 10 kuju
	MOV	M, A	Salvestatakse tulemus mällu
	XCHG		Vahetatake tagasi D&E ja H&L registripaaride sisu
	DCR	C	Vähendatakse tsükliloenduri sisu 1 võrra
	JZ	DONE	
	INX	D	Adresseeritakse lahutatava järgmine bait
	INX	H	Adresseeritakse lahutaja järgmine bait
	JMP	LOOP	Võetakse 2 järgmist 10-ndarvu numbrit ehk järgmine bait
DONE:	NOP		

Kui käsk või direktiiv võtab enda alla mitu baiti, siis nende baitide sisu prinditakse ühte ritta (sellised on antud programmis mnemokoodidega LXI, MVI ja JZ algavad käsud, mis masinakoodis salvestatakse kahte või kolme järjestikku baiti, kusjuures listingus tuuakse ära ainult madalama baidi aadress).

2.5. Signaaliprotsessorid

2.5.1. Signaaliprotsessorite ehituse iseärasused

Tööstusprotsessid jagunevad pidevatoimelisteks ja diskreetseteks. Nende kulgu kontrollivad analoogandurid nagu termopaarid, tahhogeneraatorid, mõõtepotentsiomeetrid, tensoandurid jms on pidevatoimelised seadmed, mille väljundsignaaliks on alalispinge. Mõõtereleed, teekonnalülitid, impulsi- ja koodiandurid on diskreetsed seadmed, mille väljundsignaaliks on erinevalt kodeeritud impulsid. Juhtimissüsteemis kasutatakse diskreetse toimega kontrollereid ja juhtraale, mille väljundiks on kahendarvuna esitatud juhtsõna. Paljud täiturid nagu elektrimootorid on jällegi pidevatoimelised. Seega tuleb seadmete ja protsesside juhtimisel muundada pidevatoimelisi signaale diskreetseteks ning vastupidi. Pidevatoimelisi suurusi saab esitada alalispingena või ajaintervallina. Kõige sagedamini kasutatavaks diskreetseks suuruseks on 8421 kahendkood. Juhtimisel kasutatakse nii analoog-digitaalmuundureid ehk A/D-muundureid kui ka digitaal-analoogmuundureid ehk D/A-muundureid, kusjuures mõlemal juhul võib analoogsuuruseks olla kas alalispinge või ajaintervall. D/A-andurite ehitus on mõnevõrra lihtsam ning nad kuuluvad sageli analoog-digitaalmuundurite koosseisu.

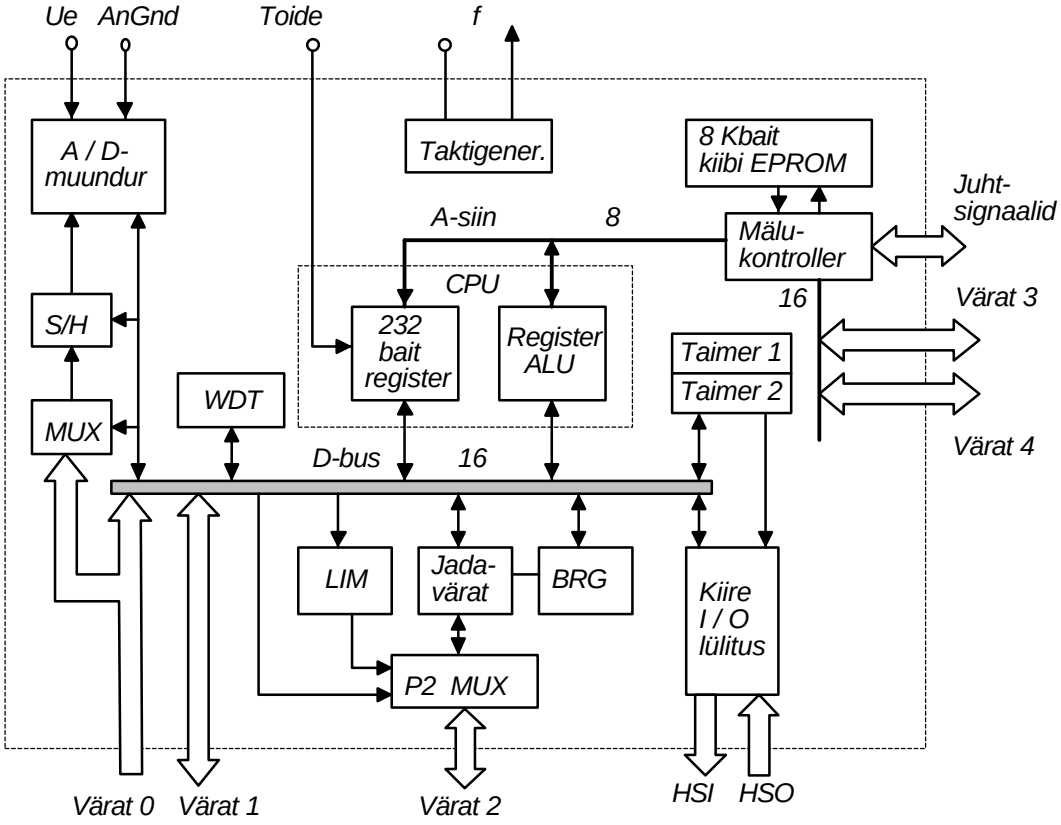
Protsessi juhtimiseks võib koostada universaalprotsessorist ning erinevatest signaali-muunduritest süsteemi. Teiseks võimaluseks on spetsiaalsete juhtimiseks ja signaalitöötluseks ettenähtud mikroprotsessorite kasutuselevõtt. Niisuguseid protsessoreid nimetatakse signaaliprotsessoriteks (*DSP - digital signal processor*). Signaaliprotsessor on varustatud mitmesuguste signaalimuunduritega nagu alalispinge analoog-digitaalmuundur, väljundpinge laiuse-impulsimodulaator, juhitava sagedusega impulsigeneraator, ajaintervalli taimer jms. Need on ette nähtud kiireks ja mahukaks infovahetuseks, kusjuures adresseeritava mälu maht on tavaliselt väiksem kui universaalprotsessoritel. Seepärast on signaaliprotsessoritel mitu sisend-väljundkanalit (mitu andme-aadressisiini), mida saab vajaduse järgi jaotada rühmadessse või eraldada. Töökiiruse suurendamiseks kasutatakse ka eraldi mälukontrollerit.

Signaaliprotsessoreid valmistatakse nii 8-, 16- kui 32-bitistena. Joonisel 2.42 on firma *Intel* 16-bitise ja 12 MHz taktisagedusega signaaliprotsessori MCS-96 üldistatud plokkskeem. Protsessori sisseehitatud 10-bitine analoog-digitaalmuundur on ühendatud välise etalonpinge allikaga *Ue* ning analoogpinge 0 klemmiga (*AnGnd*). Mõõdetavad analoogpinged sisestatakse 8-juhilise värati 0 kaudu ning kommuteeritakse multipleksori *MUX* ja diskreetimislülituse (S/H - *sample and hold*) kaudu analoog-digitaalmuunduri sisendisse.

Digitaal-analoogmuundurina kasutatakse laius-impulssmuundurit ehk pulsilaiusmuundurit (*PWM - pulse width modulation*), millel on 256 erinevat diskreetsusastet. Impulsimodulatsiooni kandevsagedus on 15,625 kHz, millele vastav impulsside periood on 64 μs. Laiuse-impulsimoduleeritud (*LIM*) ehk pulsilaiusmoduleeritud signaal väljastatakse

mitmeotstarbelise värati 2 kaudu. Erijuhul saab *LIM*-signaali väljastada kiire väljundi kanali (*HSO - high speed output*) kaudu siis, kui viimane pole hõivatud. Värati 2 kaudu toimib ka jadavärat, mille infoedastuskiirus 300...9600 boodi on programmiga valitav jadavärati kiirusjaguri (*BRG - baud rate generator*) abil Andmete sisestamiseks ja väljastamiseks saab kasutada värateid 3 ja 4 ning 8-bitist kvaasikahesuunalist väratit 1. 8-bitised väratid 3 ja 4 on programmiga ümbergrupeeritavad ühiseks 16-bitiseks aadressi-andmekanaliks.

Taimer WDT (*watchdog timer*) on ette nähtud kaitsmaks juhtraali tarkvaratõrgete eest. Sellel otstarbel saab ta lühikeste ajaintervallide järel tarkvara korrasolekut kinnitavaid signaale ning väljastab tõrke korral riistvarale signaali RESET.



Joonis 2.42. Firma *Intel* signaaliprotsessori MCS-96 plokkskeem

2.5.2. Digitaal-analoogmuundurid

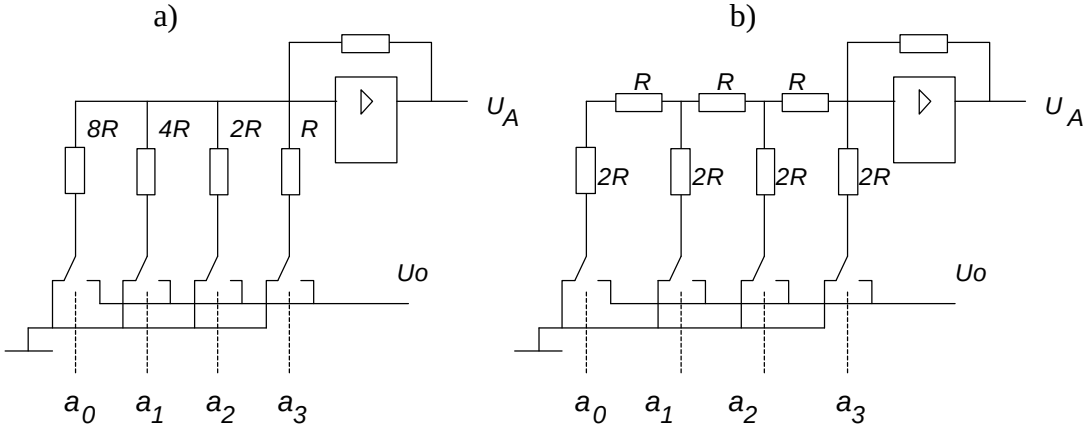
Digitaal-analoog- ehk D/A-muundurite ehitus on joonisel 2.43. Muunduri sisendsignaaliks on arvkood *Arv* U_x. Koodi muundamine alalispingeks toimub summeeriva operatsioonivõimendiga, kusjuures võimendi sisendvool on vastavalt koodile astmeliselt muudetav. Vooluastmete suurused valitakse võimendi sisendisse lülitatud takistitega nii, et koodi igale vanuselt järgmisele järgule vastab eelmisest nooremast järgust kaks korda suurem vool. Muunduri väljundpinge U_A võib arvutada valemiga

$$U_A = K_{D/A}\left(a_n \cdot 2^n + a_{n-1} \cdot 2^{n-1} + \mathrm{L} + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0\right), \qquad (2.1)$$

kus $K_{D/A}$ on muunduri ülekandetegur ja a_i kahendarvu i-nda koha väärtus, mis võrdub 0 või 1.

Signaal 0 tähendab avatud ja signaal 1 suletud kontakti. Antud juhul on kasutatud kontakte tööpõhimõtte selgitamiseks. Tegelikult kommuteeritakse takisteid transistorlülititega kontaktivabalt. Operatsioonivõimendi sisendahelaid toidetakse stabiilsest etalonpingeallikast U_0, et takisteid läbiv vool sõltuks vaid lülitite asenditest, s. t sisendkoodist.

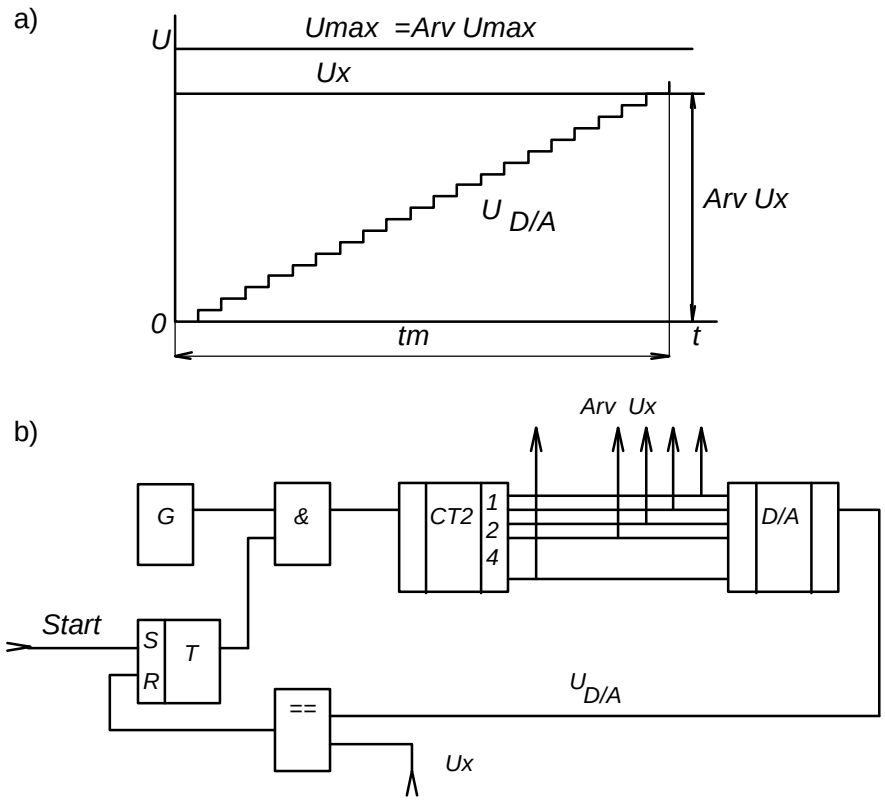
Joonisel 2.43, a toodud skeemi puuduseks on erinevate takistite kasutamise vajadus. Kuna muundureid valmistatakse tavaliselt 8...12-kohalistena, siis on koodi nooremale ja vanemale kohale vastavate takistite erinevus kuni 2^{12}= 4096 korda. Erineva takistusega täppistakistid teevad muunduri valmistamise keerukaks. Seda puudust pole joonisel 2.43, b näidatud muunduril, kus kasutatakse vaid kahe erineva takistusega *R* ja *2R* takisteid. Muunduri väljundpinge leitakse valemiga (2.1).



Joonis 2.43. Digitaal-analoogmuundurid:
a) erinevate takistitega lülitus, b) *R-2R* takistitega lülitus

2.5.3 Analoog-digitaalmuundurid

Analoog-digitaalmuundureid on tööpõhimõtte järgi kolme liiki: 1) impulsside loendamisega, 2) järgukaupa kodeerimisega ja 3) vahetu kodeerimisega. Ehituselt kõige lihtsamad on impulsside loendamisel põhinevad muundurid. Niisuguse muunduri tööd selgitab joonis 2.44. Mõõdetava alalispinge piirkond U_A viiakse vastavusse arvkoodiga *Arv U*, mis on etteantud diskreetsusastmete arv ja on valitud sõltuvalt nõutavast mõõtmistäpsusest. Ühtlasi määrab see arv muunduri poolt väljastatava kahendarvu (muunduri sõna) pikkuse, s.o bittide arvu sõnas ning sõnas sisalduva info hulga. Näiteks kümnejärguline kahendsõna määrab ära 2^{10} = 1024 diskreetsusastet. Kuna tuhande diskreetsusastme korral moodustab üks aste 0,1% kogu suurusest, siis ei ületa 10-bitise sõna muunduri viga 0,1%.

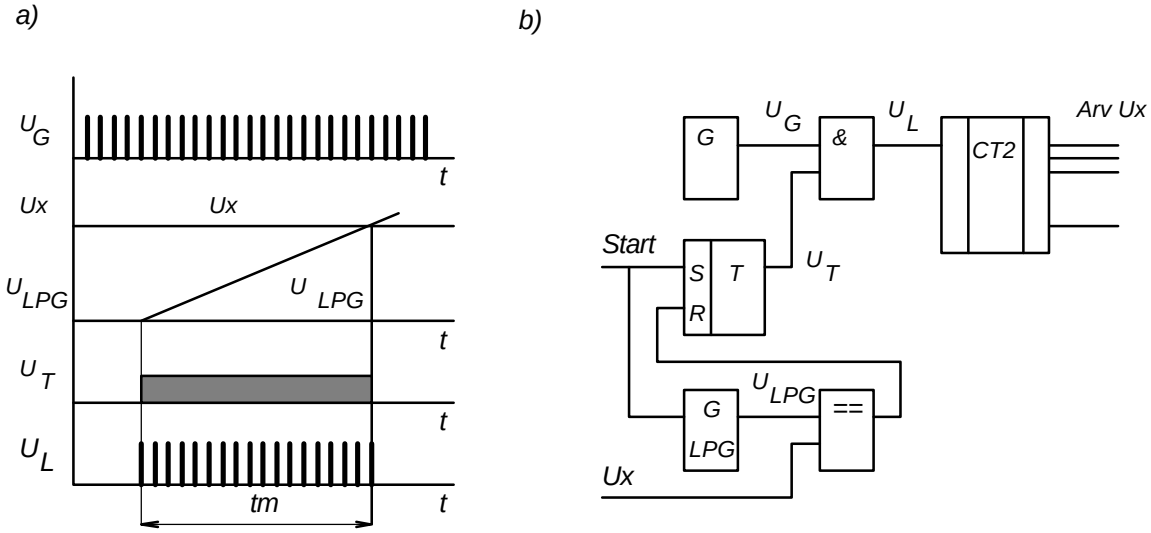


Joonis 2.44. Impulsside loendamisel põhinev analoog-digitaalmuundur

Mõõdetavale pingele U_X vastav diskreetsusastmete arv määratakse nende järjestikku loendamisega. Loendatakse seni, kuni loendatud arvule vastav diskreetne pinge saab võrdseks mõõdetava pingega. Seega on muundamiseks kuluv aeg võrdeline loendatavate diskreetsusastmete arvuga ning on maksimaalselt 2^n takti, kus *n* on muunduri kahendsõna kohtade arv. Joonisel 2.44 tähistab t_m mõõtmiseks kuluvat aega. Treppkõver iseloomustab

loenduri väljundkoodi pinget. Niisugune pinge saadakse D/A-muunduri väljundist. Trepp-pinget võrreldakse sisendpingega U_x. Pingete võrdsustamisel lülitab komparaator trigeri väljundi nulli. Loogikaelement NING suleb generaatori
impulssidele pääsu loendurisse ning impulsside loendamine katkeb. Loenduri väljundis säilib sisendpingele U_x vastav kood *Arv* U_x, mis on ühtlasi muunduri väljundsignaaliks.

Kui mõõtmistäpsus pole eriti oluline, saab A/D-muunduri skeemi lihtsustada (joonis 2.45). Selleks loobutakse D/A-muundurist ja tagasisidest pinge järgi. Kui impulsigeneraatori sagedus on konstantne, siis loendatakse impulsse alati ühesuguse sagedusega ning pinget $U_{D/A}$ võib aproksimeerida lineaarselt kasvava pingega. Järelikult võib sisendpinget U_X võrrelda lineaarpinge generaatori LPG väljundpingega. Kui ka lineaarpinge on küllalt stabiilne, siis on pingete võrdsustumiseni kuluv ajavahemik võrdeline mõõdetava pingega U_x. Konstantse sagedusega impulsside loendamise korral osutub ajavahemiku t_m jooksul loendatud impulsside arv võrdeliseks sisendpingega U_x. Lineaarpinge ja sisendpinge
võrdsustumisel loenduri töö katkestatakse ning loenduri väljundist saadakse mõõdetavale pingele U_x vastav arvkood *Arv* U_x.



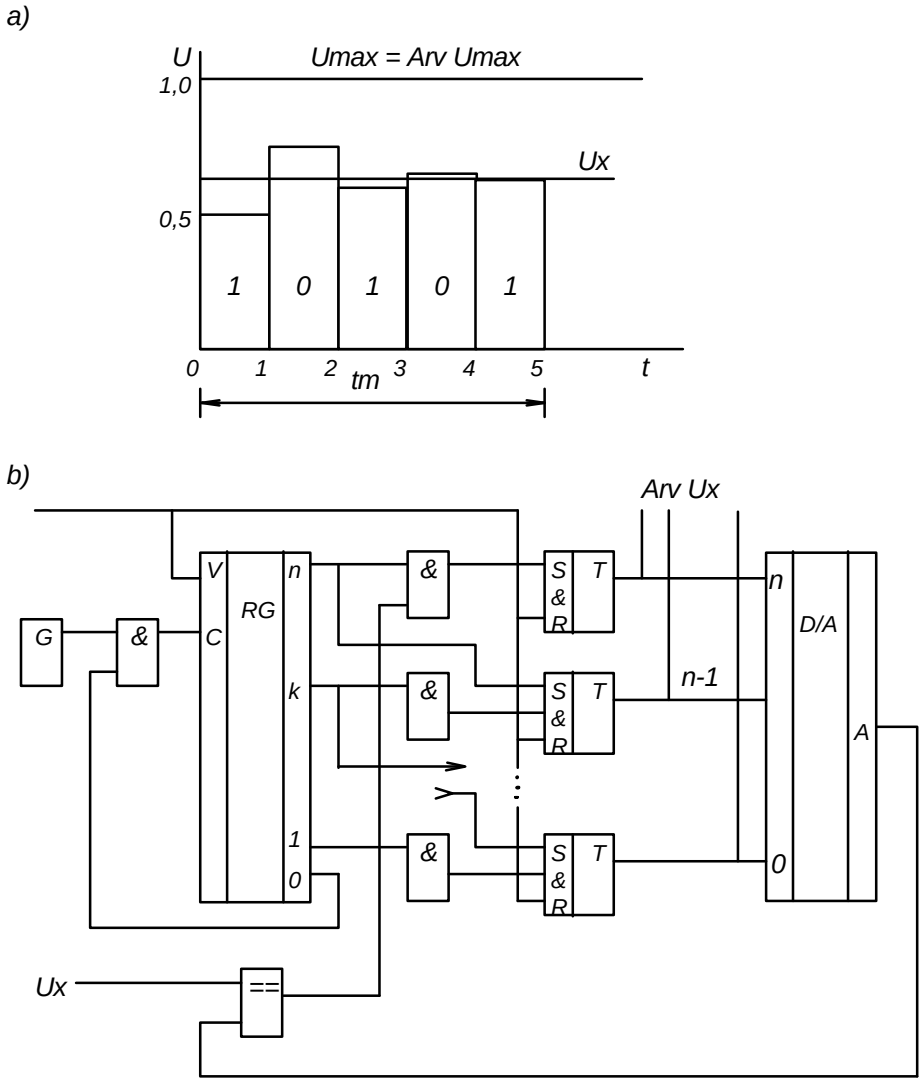
Joonis 2.45. Tagasisideta analoog-digitaalmuundur

Impulsse järjestikku loendavate muundurite peamiseks puuduseks on väike töökiirus. Seepärast kasutatakse niisuguseid A/D-muundureid suhteliselt aeglaselt muutuvate suuruste mõõtmisel ja juhtimisel.

Tunduvalt kiiremini töötavad väljundsignaali kohakaupa kodeerivad muundurid. Nende tööpõhimõtet illustreerib joonis 2.46. Analoogsignaali mõõtepiirkond U_A viiakse ka siin vastavusse teatud arvkoodiga *Arv U*. Koodi kohtade arv määrab ära nii diskreetsusastmed

kui ka muunduri täpsuse. Muundamine ei seisne siin diskreetsusastmete loendamises, vaid mõõdetavale signaalile vastava kahendarvu kohtade järjestikku kodeerimises. Alustatakse kahendarvu vanemast kohast, mis vastab ühtlasi poolele mõõtepiirkonnale U_A. Kui see koht võrdub ühega, siis $U_x > 0{,}5\ U_A$, kui aga nulliga, siis $U_x < 0{,}5\ U_A$. Edasi toimub järgmise suuruselt teise koha kodeerimine. Kui koodi vanem koht võrdus ühega ja ka teine koht võrdub ühega, siis võib väita, et $U_x > 0{,}75\ U_A$. Kui aga teine koht võrdub nulliga, siis $0{,}5\ U_A < U_x < 0{,}75\ U_A$. Järgmisena kodeeritakse koodi kolmas koht, siis neljas jne kuni kõige noorema kohani välja. Muundamise kestus on määratud kahendsõna kohtade arvuga. Kui oletada, et iga taktiga kodeeritakse üks koht, siis on kogu protsessi kestus võrdne *n* taktiga, kus *n* on kahendsõna kohtade arv.

Joonisel 2.46, b kujutatud muundur töötab järgmiselt. Käsk *Start* viib väljundsõna kõige vanemale kohale vastava trigeri olekusse 1, kõigi nooremate kohtade trigerid aga olekusse 0. 5-kohalise sõna korral (joonis 2.46, a) antakse D/A-muunduri sisendisse kood 10000. Muundur väljastab sellele koodile vastava pinge, mida võrreldakse mõõdetava sisendpingega U_x. Kui U_x osutub väiksemaks kui koodile 10000 vastav pinge, siis muutub komparaatori väljundsignaal üheks. Järgmisel taktil läbib nihkeregistri väljundist saabuv impulss NING-elemendi ja viib vanema koha trigeri olekusse 0. Ühtlasi viib sama impulss teise koha trigeri olekusse 1. D/A-muunduri sisendisse antakse sel juhul kood 01000, millele vastavat pinget võrreldakse uuesti sisendpingega U_x.

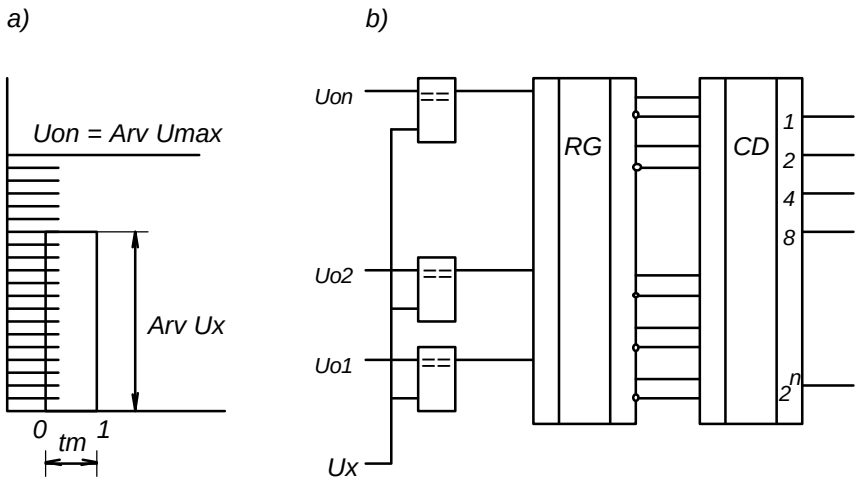


Joonis 2.46. Kohakaupa kodeerimisega analoog-digitaalmuundur

Kui esimesel võrdlemisel osutus sisendpinge U_x koodile 10000 vastavast D/A-muunduri pingest suuremaks, siis jääb komparaatori väljundpinge nulliks ning vanema järgu triger olekusse 1. Järgmisel taktil lülitatakse olekusse 1 ka teise koha triger ning komparaatoris võrreldakse sisendpinget U_x koodile 11000 vastava pingega. Kui see osutub sisendpingest suuremaks, lülitatakse teise koha trigeri väljund uuesti nulli ja kolmanda koha triger olekusse 1. Seega võrreldakse kolmandal taktil sisendpinget U_x koodile 10100 vastava pingega jne. Iga taktiga lülitab nihkeregister aktiivsesse olekusse järgmise väljundi ning sõltuvalt komparaatoris toimuva võrdluse tulemusest moodustatakse trigerite abil väljundsõna järgud. Kogu protsess toimub hetkeni, mil nihkeregister lülitab sisse viimase väljundi, mille signaal sulgeb impulsigeneraatori impulssidele pääsu registrisse ning A/D-muunduri töö peatub. Trigeritesse salvestatud kahendkood *Arv* U_x vastab sel juhul sisendpingele U_x.

Võrreldes muunduritega, mis loendavad impulsse järjestikku, töötavad kohakaupa väljundsõna kodeerivad muundurid palju kiiremini. 10-kohalise väljundsõna korral kulub muundamiseks vaid 10 takti. Impulsside loendamise korral kulub selleks kuni 1024 takti, s. o üle 100 korra rohkem.

A/D-muunduri töökiirus suureneb veelgi, kui väljundsõna vahetult kodeerida. See meetod eeldab sisendpinge U_X võrdlemist muunduri ühe töötakti kestel kõigile diskreetsusastmetele vastavate pingetega. Niisugune võrdlemine nõuab suurt hulka etalonpingeallikaid. Seega peab 10-kohalise väljundsõna korral kasutama 1024 etalonpingeallikat. Skeemi keerukuse tõttu pole niisuguse muunduri koostamine üksikelementide kaupa mõeldav. Kogu keeruka skeemi võib aga paigutada suurde integraallülitusse. Muunduri tööpõhimõte on joonisel 2.47.



Joonis 2.47. Vahetu analoog-digitaalmuundur

Sisendpinge U_X võrdlemisel i-nda etalonpingega U_{ei} saadakse komparaatorite väljundis sisendpingele vastav ühikkood, mis salvestatakse registrisse *RG*. Kooder *CD* muundab ühikkoodi kahendkoodis väljundsõnaks *Arv* U_X. Muundusprotsessiks kulub vaid üks töötakt.

2.5.4. Transpuutrid

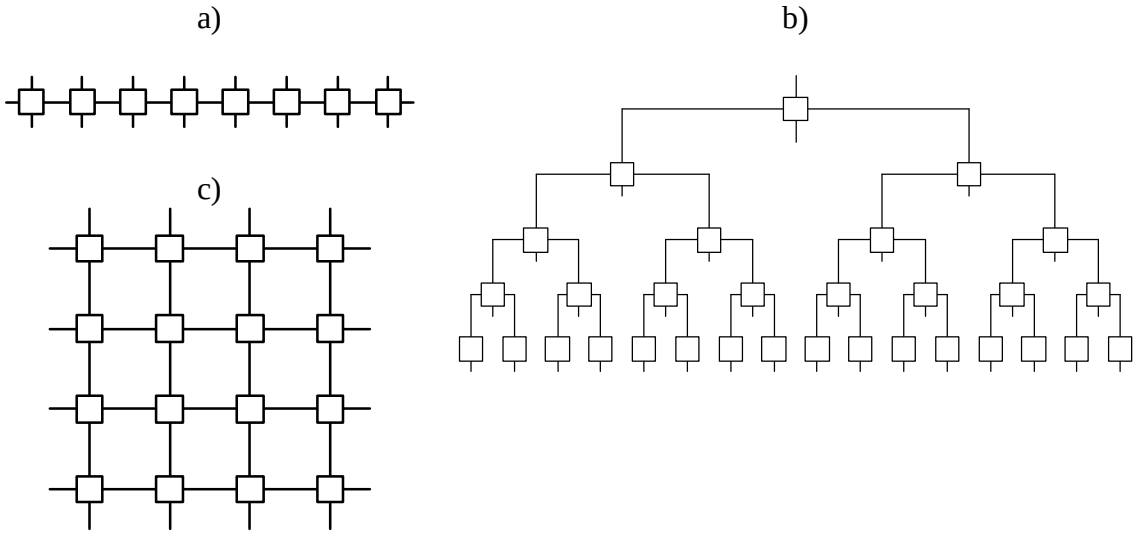
Tööstusseadmete ja protsesside juhtimiseks kasutatakse üha täpsemaid ja täiuslikumaid kuid ühtlasi ka keerukamaid algoritme. Suureneb tehisintellekti elementidega juhtimissüsteemide osatähtsus. Keerukate reaalajasüsteemide realiseerimiseks on võetud

kasutusele multiprotsessorsüsteemid ja mitmeraalijuhtseadmed. Mitme protsessori või juhtraali ühendamine ühte süsteemi toimub erinevalt. Suure süsteemi elemente võib ühendada järjestikku lineaarsesse või ringmagistraali; süsteemi võib kujundada hierarhilise või võrgustruktuurina. Süsteem võib talitleda tsentraalse või detsentraliseeritud süsteemina ehk hajusjuhtimise põhimõttel.

Mitmeraalijuhtseadme elementidena kasutatakse ka harilikke protsessoreid, kuid tänapäeval on selleks otstarbeks välja töötatud eri tüüpi mikroprotsessorid, mida nimetatakse transpuutriteks. Transpuutrite väljaarendamisel on silmas peetud, et oleks tagatud

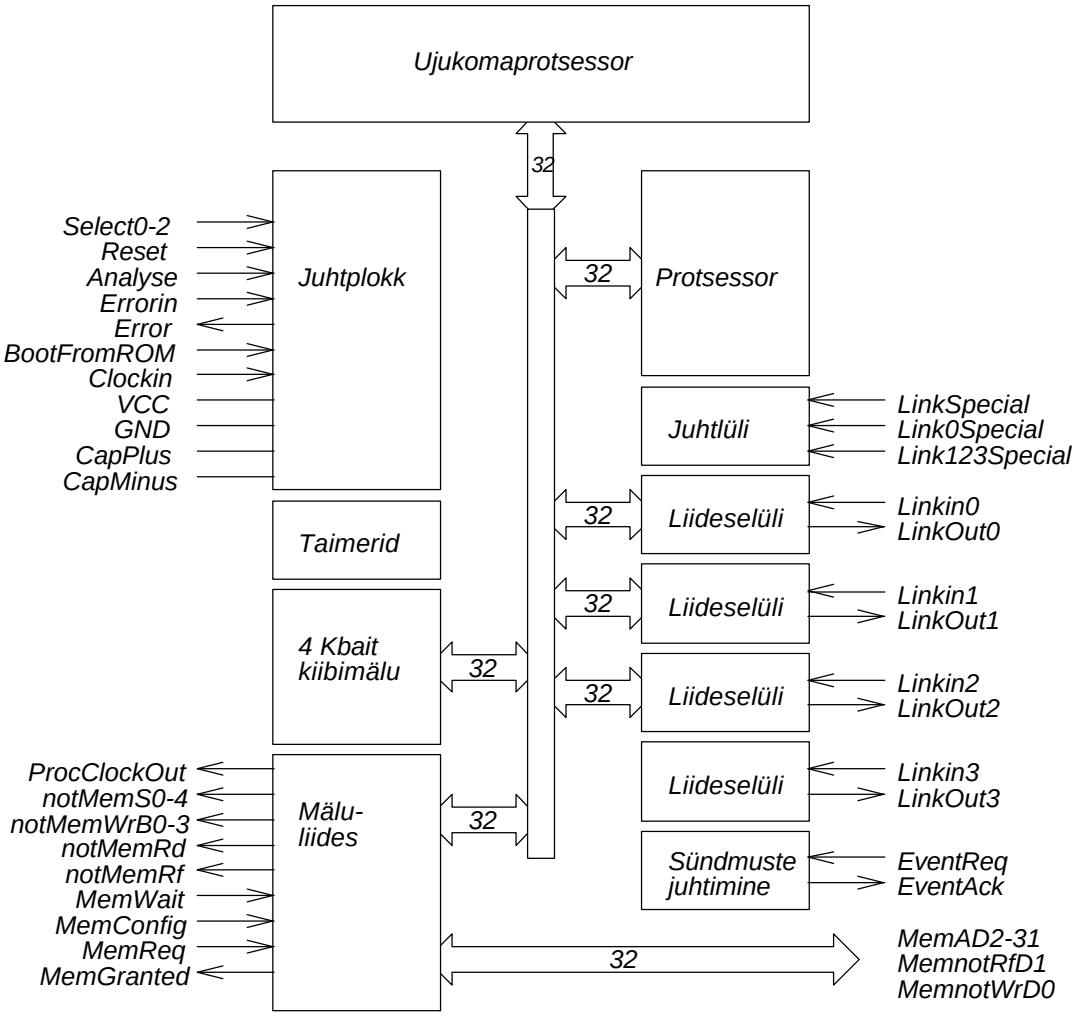
- § süsteemide edasiarendamise võimalus,
- § komponentide ühildatavus,
- § riistvara ühetaolisus ja projekteerimise lihtsus,
- § tarkvara väljatöötamise lihtsus,
- § reaalajasüsteemide juhtimine,
- § rööpprotsesside programmeerimine,
- § protsessoritevahelise kommunikatsiooni tõhusus ja lihtsus,
- § kasutajaliidese standardsus ja paindlikkus.

Transpuutrid on ehitatud nii, et neid võib korraga ühendada nelja omataolisega. Transpuutrite ühendamiseks kasutatakse jadaliidest, mis muudab ühendamise maksimaalselt lihtsaks (ainult kaks juhet). Transpuutrisüsteemide mõned tüüpilised struktuurid on näidatud joonisel 2.48. Juhtimisprotsess sünkroniseeritakse transpuutritevahelise andmevahetusega. Transpuutri andmeedastuskiirus küünib 10...20 Mboodini.



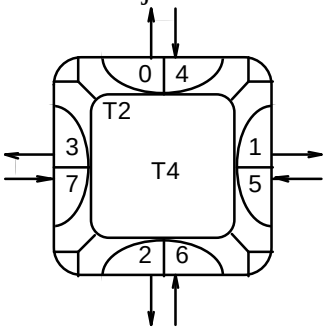
Joonis 2.48. Transpuutrisüsteemide struktuure: a) lineaarne, b) hierarhiline, c) võrgustruktuur

Transpuutri T800 plokkskeem on näidatud joonisel 2.49. Võrdluseks võib lisada, et firma *INMOS* transpuutriperekonna protsessorid T200, T400 ja T800 vastavad üksikult võttes efektiivsuselt ligikaudu firma *Intel* protsessoritele 80286, 80386 ja 80486, kuid on ette nähtud tööks suurtes süsteemides ning nende tegelik jõudlus avaldub süsteemi elementide koostöös.



Joonis 2.49. Transpuutri T800 plokkskeem

Esimene 32bitine transpuuter T-424 töötati välja 1985. aastal firma INMOS poolt. Sellest ajast alates on toimunud transpuuterite riist- ja tarkvara kiire areng.



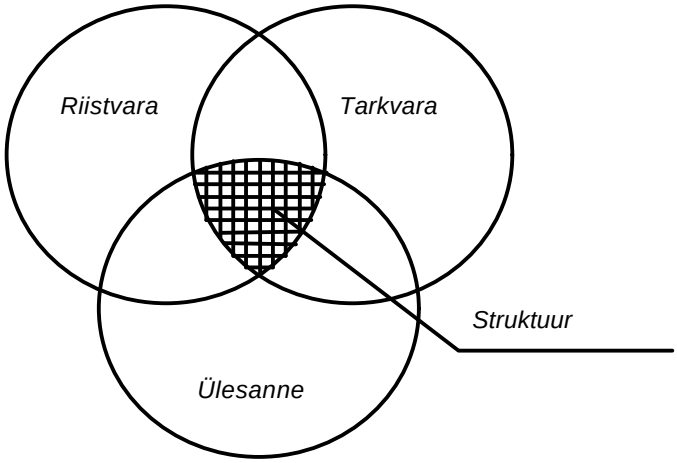
2.6. Mikroprotsessorite kasutamine

2.6.1. Ülevaade

Mikroprotsessoreid kasutatakse nii universaal- kui ka spetsiaalarvutites ehk juhtraaliseadmetes. Universaalarvutite klassi võib lugeda tasku- ja lauakalkulaatorid, mitmesugused personaalarvutid (*PC - personal computer*) (väikearvutid) alates kaasaskantavatest tasku- ja märkmikarvutitest (*notebook*) kuni bürooarvutite ja tööjaamadeni (*work station*), ning süsteemis kollektiivseks kasutamiseks mõeldud suurarvutid. Kalkulaatorid erinevad arvutitest selle poolest, et nad on mõeldud kitsamatele rakendusaladele enamasti arvutusteks ning on varustatud sageli spetsiaalsete ja odavate perifeerseadmetega.

Juhtraaliseadmete hulka kuuluvad kõige lihtsamatest alates binaarsete süsteemide kontrollerid, ühe- ja mitmeraalijuhtseadmed. Neist viimaseid võib omakorda liigitada sõltuvalt juhtseadme sisestruktuurist kas hierarhilise, ahel-, ring- või maatriksstruktuuriga seadmeteks. Juhtraalides kasutatakse sageli spetsiaalselt signaalitöötluseks ette nähtud kiiretoimelisi signaaliprotsessoreid. Mitmeraalijuhtseadmete suhteliselt uueks raalelemendiks on transpuutrid. Eriotstarbelisi signaaliprotsessoreid (*DSP - digital signal processor*) ning transpuutreid kasutatakse mitmesuguste süsteemide ja protsesside juhtimiseks paljudes tööstusharudes.

Ülesanne (algoritm, objekt), mille lahendamiseks kavatsetakse rakendada arvutit, esitab nõudeid nii tark- kui ka riistvarale. Diagrammil joonis 2.50 iseloomustab seda ringide ühisosa.

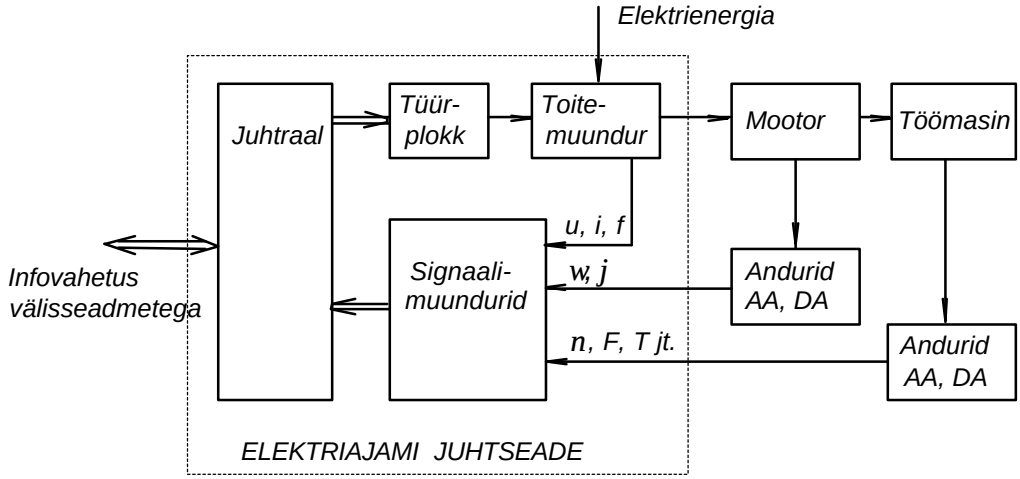


Joonis 2.50. Ülesande struktuuri seos riist- ja tarkvaraga

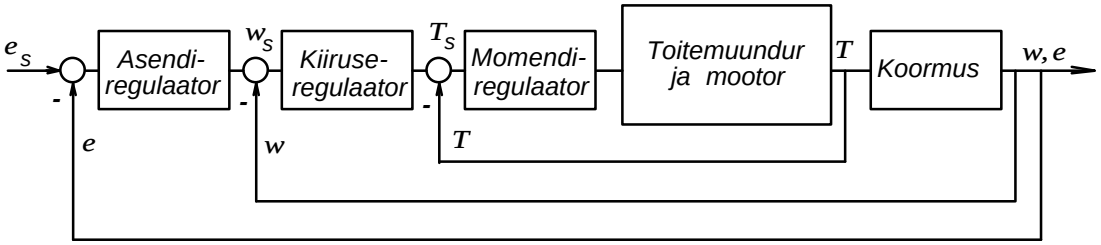
2.6.2. Mikroprotsessorid elektriajamis

Elektriajamites kasutatakse mikroprotsessoreid nii juhtimiseks kui ka kaitse otstarbel. Raaljuhtimise peamine iseärasus võrreldes hariliku mitteprogrammeeritava juhtimisega on paindlikkuses ja universaalsuses. Universaalne riistvara võimaldab luua ajameid mitmesuguste ülesannete täitmiseks, kusjuures nende funktsioone saab lihtsalt muuta. Niisuguste omaduste tõttu annab raaljuhtimisega ajamitele üleminek olulist efekti nii ajamite projekteerimisel, tootmisel kui ka kasutamisel. Riistvara unifitseerimisega kaob vajadus välja töötada kümneid või isegi sadu erinevaid ajamite juhtseadmeid. Ajamite projekteerimine taandub peamiselt tarkvarasüsteemide väljatöötamisele, mis on samuti automatiseeritav. Tootmises parandab unifitseerimine seadmete kvaliteeti ning suurendab nende töökindlust. Funktsioonide avardamine teeb niisuguste ajamite kasutamise käepärasemaks.

Raaljuhtimisega elektriajami plokkskeem on joonisel 2.51. Töömasinat käitab mootor *M*, mille võllile paigutatakse nii diskreetseid kui ka analoogandureid. Anduritega määratakse kindlaks ajami tööd iseloomustavad suurused: võlli pöördenurk j, nurkkiirus w ja kiirendus. Mootori mähiseid toidetakse pooljuhtmuundurist, mis sisaldab ajami tööks vajalikke elektriliste suuruste (pinge *u*, voolu *i*, sageduse *f*) andureid. Kõik mootori olekut iseloomustavad signaalid edastatakse sisendmuundurite kaudu kahendarvudena juhtraali. Ajami juhtfunktsioonid realiseeritakse raali salvestatud programmide abil, kusjuures nende täitmisel arvestatakse välisseadmetelt, teistelt juhtraalidelt, tehnoloogiaseadmetelt, operaatori juhtpuldilt jne saabuvaid signaale. Juhtraal väljastab juhtsõna toitemuunduri tüürplokile, mis kujutab endast samuti signaalimuundurit, muundades kahendkoodis juhtsõna nõutava sagedusega, harvendusega ja faaside arvuga impulsspingeks. Viimast kasutatakse jõumuunduri tüürimiseks. Sõltuvalt tüürpingest kujunevad välja voolud mootori mähistes ning mehaanilised suurused võllil. Märkigem, et joonisel näidatud plokkskeem ei kajasta ajami juhtimissüsteemi talitlust, mis sõltub juhtraali mällu salvestatud programmist.

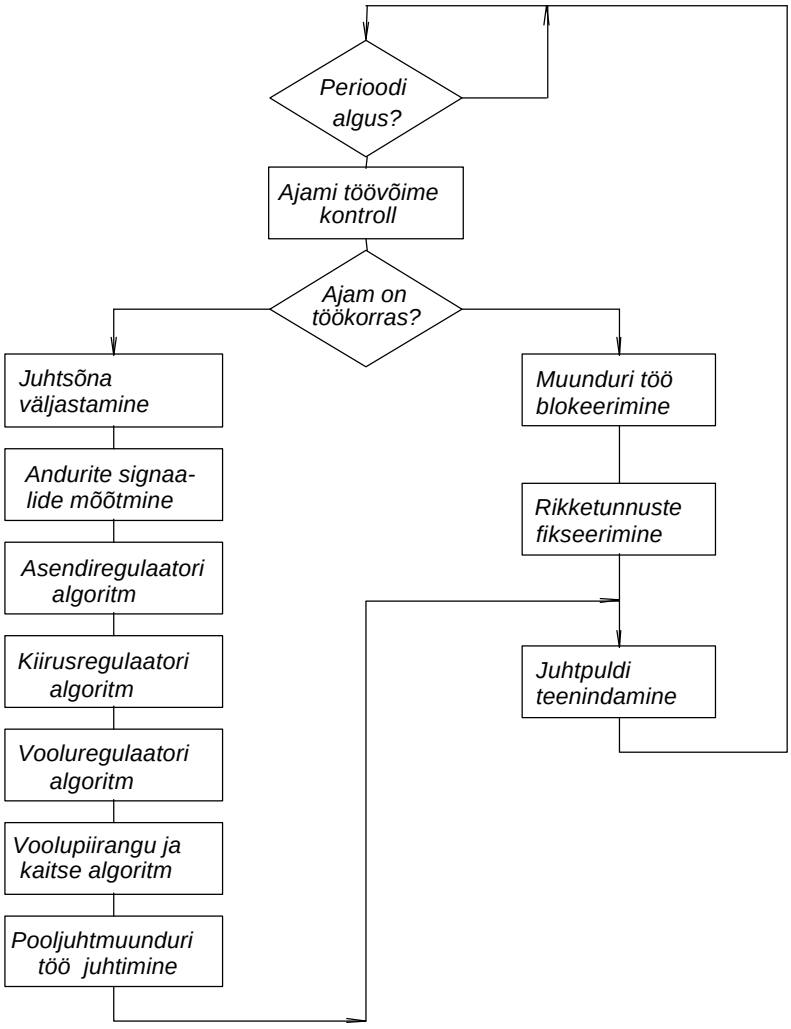


Joonis 2.51. Raaljuhtimisega elektriajami plokkskeem



Joonis 2.52. Alluvkontuuridega ajami plokkskeem

Alluvkontuuridega süsteemis võib iga sisemist kontuuri vaadelda seda haarava välimise kontuuri suhtes kui juhtimisobjekti. Kuigi alluvkontuuridega süsteemid on leidnud kasutamist peamiselt pidevatoimeliste regulaatoritega ajamites, rakendatakse sama põhimõtet ka raaljuhtimise korral. Sel juhul kujutab joonisel näidatud plokkskeem ajami näivstruktuuri, mis on aluseks ajami juhtalgoritmi koostamisel. Algoritm realiseeritakse omakorda mikroarvuti programmina. Niisuguse algoritmi plokkskeem on joonisel 2.53.



Joonis 2.53. Alluvkontuuridega ajami juhtalgoritm

Joonisel näidatud juhtalgoritmi plokkskeem kirjeldab ajami tööd tsükli jooksul, mis pidevalt kordub. Osa tsüklist täidetakse ainult eritingimusel, siis kui selgub, et ajam pole töökorras ning tuleb peatada. Ajami normaalsel töötamisel võib eristada mitmesuguseid tsükleid. Kõigepealt on ajami normaalseks tööks vaja kolme järjestikku operatsiooni: tagasisidesignaalide mõõtmine, tulemuste võrdlemine seadesuurustega ja regulaatorite väljundsuuruste arvutamine ning pooljuhtmuunduri juhtimine. Kõigi nende operatsioonide kordamise sagedus võib olla ühesugune, kuid vastavalt vajadusele kasutatakse ka erineva kestusega tsükleid. Seejuures arvestatakse kindlaid seoseid, mis piiravad nende tsüklite kestuse valikut.

Pooljuhtmuunduri juhtimise sageduse määrab konstruktsioon. Võrgust toidetavate türistoralaldite korral on juhtimisperioodi kestus sõltuvalt muunduri lülitusskeemist 3,3...20 ms. Autonoomsete vaheldite ja laius-impulssmuundurite juhtimisperioodi kestus on määratud muunduri modulatsioonisagedusega ning see võib muutuda suurtes piirides (1 ... 100 kHz), millele vastav tsükli kestus on 1 ... 0,01 ms.

Arvregulaatorite sisend- ja väljundfunktsioone esitatakse diskreetsete väärtuste jadana, kus muutujate hetkväärtused on fikseeritud ajaintervalli Δt järel. Funktsiooni tuletisteks aja järgi on vastavat järku diferentsfunktsioonid, integraalideks aga summa funktsioonid. Regulaatori väljundfunktsioon iseloomustab väljundsuuruse sõltuvust sisendsuurusest.

Näiteks pidevatoimelise proportsionaalregulaatori ehk P regulaatori tööd kirjeldab väljundfunktsioon

$$U_{reg} = K_p \cdot \Delta U(t), \qquad (2.2)$$

kus K_p on regulaatori ülekandetegur, *DU(t)* sisendsignaal. Diskreetse regulaatori korral tuleb pidev aeg *t* asendada diskreetse ajaga *n*, mis kujutab endast järjestikuste ajaintervallide järjekorranumbreid

$$U_{reg} = K_p \cdot \Delta U(n). \qquad (2.3)$$

Negatiivse tagasisidega süsteemis on regulaatori sisendsuurus *DU(n)* seadesignaali ja tagasisidesignaali vahe

$$\Delta U(n) = U_s(n) - U_{ts}(n). \qquad (2.4)$$

Diskreetse regulaatori väljundfunktsioon

$$U_{reg}(n) = K_p \cdot [U_s(n) - U_{ts}(n)]. \qquad (2.5)$$

Analoogiliselt on avaldatav ka integraal- ehk I-regulaatori väljundfunktsioon

$$U_{reg}(n) = K_i \cdot \sum_{j=0}^{n} [U_s(j) - U_{ts}(j)]. \qquad (2.6)$$

Ajamite juhtimisel leiavad kõige enam kasutamist PI-regulaatorid, millel on nii P- kui ka I-regulaatori omadused. Diskreetse PI-regulaatori väljundfunktsioon

$$U_{reg}(n) = K_p \cdot [U_s(n) - U_{ts}(n)] + K_i \cdot \sum_{j=0}^{n} [U_s(j) - U_{ts}(j)], \qquad (2.7)$$

kus võrrandi parempoolse avaldise esimene liige vastab P-regulaatorile, teine aga I-regulaatorile.

Diskreetse diferentsiaalregulaatori ehk D-regulaatori tööd kirjeldab väljundfunktsioon

$$U_{reg}(n) = K_d \cdot \left[[U_s(n) - U_{ts}(n)] - [U_s(n-1) - U_{ts}(n-1)] \right]. \qquad (2.7)$$

Kui regulaatori sisendis toimiv seadesignaal ei muutu, siis avaldub regulaatori väljundfunktsioon järgmiselt:

$$U_{reg}(n) = -K_d \cdot [U_{ts}(n) - U_{ts}(n-1)]$$

. (2.8)

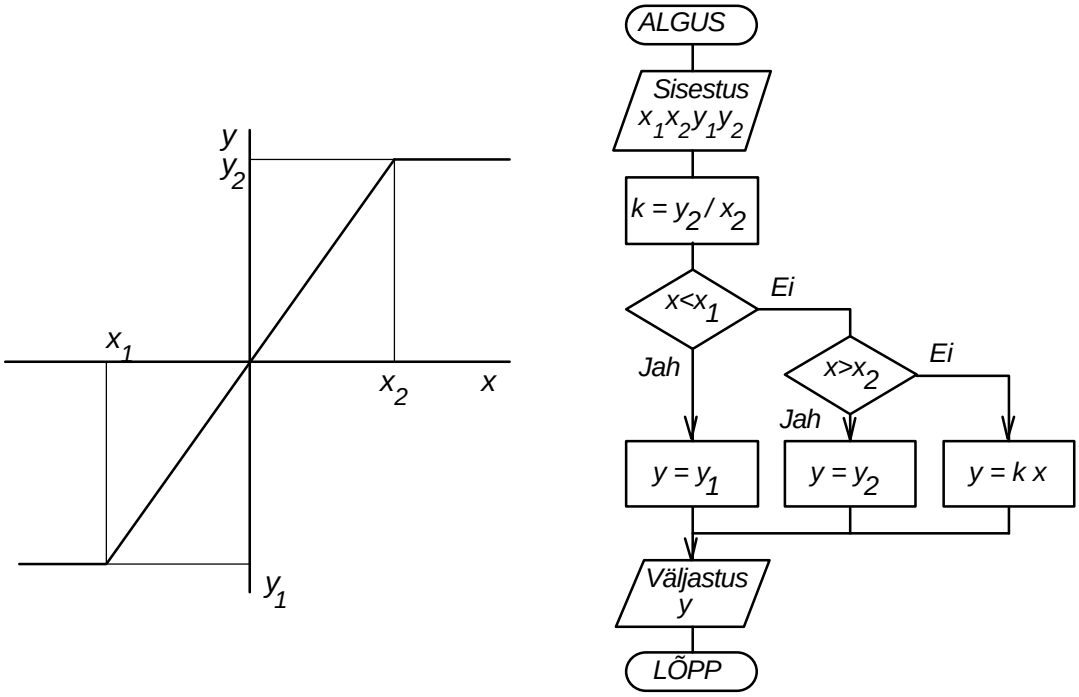
Viimasest võrrandist järeldub, et regulaatori väljundsignaal ei sõltu seadesignaalist ning on määratud ainult tagasisidesignaali muutumise kiirusega.

P-, I- ja D-regulaatorite ühendamisel saadakse PID-regulaator, mille väljundfunktsiooni võib esitada kolme liidetava summana

$$U_{reg}(n) = K_p \cdot [U_s(n) - U_{ts}(n)] + K_i \cdot \sum_{j=0}^{n} [U_s(j) - U_{ts}(j)] + \\ - K_d \cdot [U_{ts}(n) - U_{ts}(n-1)]. \quad (2.9)$$

Süsteemi muutujate piiramiseks kasutatakse mittelineaarse karakteristikuga regulaatoreid. Tüüpilisteks mittelineaarsusteks on tundetustsoon ja küllastus. Joonisel 2.54 on küllastusega funktsiooni algoritm. Kuna tegemist on tükiti lineaarse funktsiooniga, siis tuleb programmeerimisel ette anda selle üksikute lõikude võrrandid ning käänupunktide koordinaadid. Arvregulaatoris realiseeritakse piirangute algoritm alamprogrammina, mis täidetakse pärast regulaatori väljundfunktsiooni programmi. Sarnaste piirangute korral võib piirangute alamprogrammi kasutada korduvalt erinevate regulaatorite juures, muutes seejuures vaid piirangufunktsiooni parameetrite arvväärtusi.

Algoritmis sisalduvaid matemaatilisi funktsioone arvutatakse sageli ligikaudsete meetoditega. Niisuguste meetodite hulka kuuluvad näiteks mittelineaarsete funktsioonide esitamine tabelitena või nende aproksimeerimine tükiti lineaarsete või polünomiaalsete funktsioonidega. See tähendab, et suurt toimekiirust nõudvates ajamites tuleb mikroarvuti ebapiisava toimekiiruse tõttu minna arvutuste täpsuse ja nende täitmise kiiruse valikul kompromissile.



Joonis 2.54. *Küllastusega* mittelineaarne karakteristik ja sellele vastav modelleerimise algoritm

2.6.3. Mikroprotsessorid releekaitses

Energia- ja tööstussüsteemides pööratakse suurt tähelepanu süsteemide kaitsele avariide eest. Avariide ennetamiseks kasutatakse kaitsesüsteemi, mida energiasüsteemide korral nimetatakse traditsiooniliselt releekaitseks. Nimetus releekaitse on tulnud sellest, et pikka aega kasutati süsteemi kaitseks mitmesuguseid elektromehaanilisi kaitsereleesid. Hiljem, kui kaitsesüsteemid moderniseerusid, asendati elektromehaanilised releed pooljuhtreleedega. Viimastel aastatel on hakatud üle minema mikroprotsessoritel põhinevatele programmeeritavatele kaitsesüsteemidele.

Elektrimootorite kaitse seisneb ohtlike rež iimide väljaselgitamises, ohust õigeaegses signaliseerimises, mootori väljalülitamises või töörež iimi muutmises. Peamiselt ohustab mootorit ülekuumenemine, mida põhjustab elektrienergia kadu mähiste aktiivtakistusel ning pöörisvoolust tingitud kadu magnetahelas, samuti hõõrdumine laagrites ning väliskeskkonna temperatuur. Enamikul juhtudel on keskkonna temperatuur mootori omast madalam ning osa soojusenergiast hajub keskkonda. Teine osa soojusenergiast akumuleerub mootoris, põhjustades temperatuuri tõusu. Teatud temperatuurist alates muutub see ohtlikuks mähiste ja juhtmete isolatsioonile.

Teoreetiliselt oleks mootorit kõige lihtsam kaitsta, kui reageerida temperatuurile vahetult. Kahjuks on temperatuuri objektiivselt mõõta raske. Üheks põhjuseks on temperatuuri ebaühtlane jaotumine mootoris ning temperatuurivälja sõltuvus töörež iimist. Teiseks on temperatuuriandureid mootori mähistesse tülikas paigutada, nende talitlus seal on aga väikese töökindlusega. Seepärast põhineb enamiku kaitseaparaatide töö elektrimootori soojusliku oleku kaudsel hindamisel toitepinge, voolu ja sageduse ning keskkonna temperatuuri järgi. Sisuliselt peab kaitseaparaat modelleerima mootori töötamisel toimuvaid soojenemis- ja jahtumisprotsesse. Laialt levinud kaitseaparaatideks on lihtsad voolu-termo-releed, kus koormuse soojuslikke protsesse imiteerib bimetall-leht. Kasutatakse ka sulavkaitsmeid ja kaitselüliteid. Kahjuks ei kaitse need lihtsad aparaadid paljudel juhtudel mootorit piisavalt.

Alates 80-datest aastatest on paljud firmad hakanud mootoreid kaitsma mikroprotsessoritel põhinevate aparaatidega.

Programmeeritav kaitseaparaat peab tuvastama mootorit ohustava rež iimi ning vastavalt ohtlikkuse määrale reageerima ohusignaaliga või väljalülituskäsuga. Ebanormaalseteks rež iimideks loetakse kõikvõimalikke lühiseid, mootori ülekoormust, asünkroonmootori vääratust, isolatsiooni leket ning maaühendusvoolu tekkimist, toitepinge katkemist, mähiste mittesümmeetrilisi voolusid ning mõnel juhul ka koormusvoolu järsku vähenemist. Loetletud rež iimide alusel on otstarbekas analüüsida kaitse rakendumise tingimusi.

Programmeeritava kaitseaparaadi kasutajad võivad suurtes piirides muuta kaitse tehnilisi näitajaid ja tunnusjoonte kuju. Sellega tagatakse iga konkreetse mootori võimalikult kindel kaitse. Eri tüüpi kaitseaparaatidel võivad sätete reguleerimise piirid oluliselt erineda. Seepärast vaadeldagu järgnevalt esitatud sätete arvväärtusi kui võimalikke variante. Aluseks on võetud firma ABB kaitseaparaatide sätete väärtused.

Mootori kaitse programmeerimisel on vaja kõigepealt määrata pidevrež iimil maksimaalselt lubatud vool ehk nn täiskoormusvool I_u = (0,5...1,5) I_n, kus I_n on mootori nimivool.

Lühiseks loetakse reziimi, kus mootori vool $I >> In$. Lühisekaitse rakendumislävi valitakse piirides I_L = (2...12) I_n. Kaitse rakendumisaeg peab sel juhul olema väiksem kui 50 ms.

Ülekoormus fikseeritakse juhul, kui mootori vool on suurem täiskoormusvoolust, s. t I > 1,05 I_u. Seejuures arvestatakse käivitusprotsessi eraldi. Enne käivituse algust mootor seisab ning vool I < 0,12 I_u. Käivitamisel kasvab vool 60 ms jooksul mitmekordseks I > 2,5 I_u. Käivituse võib lugeda lõppenuks, kui vool 100 ms jooksul järsult väheneb I < 1,75 I_u. Käivitusprotsessi alguse ja lõpu järgi fikseeritakse käivituse toimumine ning leitakse käivitusaeg. Need andmed on aluseks ka käivitusprotsessis eraldunud soojusenergia arvutamisel.

Soojuslike protsesside modelleerimisel lähtutakse homogeense keha soojusliku tasakaalu võrrandist

$$\Delta P \cdot dt = C \cdot d J_{ü} + A J_{ü} \cdot dt, \qquad (2.10)$$

kus $\Delta P \cdot dt$ on ajavahemiku dt jooksul mootoris eralduv soojusenergia, ΔP kaovõimsus, C soojusmahtuvus, $J_{ü}$ mootori ületemperatuur võrreldes keskkonnaga ja A soojus-siirdetegur. Suurust $\tau = C / A$ nimetatakse mootori soojuslikuks ajakonstandiks. Muutumatute parameetrite korral on võrrandi lahendiks soojenemisel

$$J_{Ÿ} = J_{Ÿl} \left(1 - e^{-t/t}\right) + J_{Ÿa} e^{-t/t}, \qquad (2.11)$$

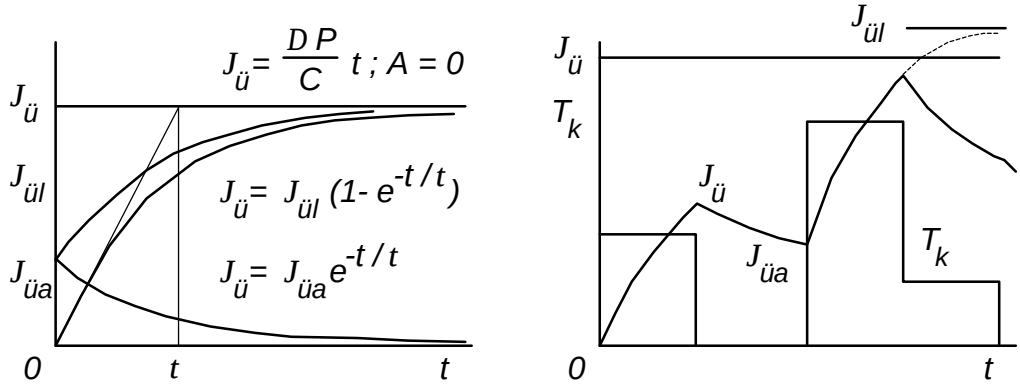
kus $J_{ül}$ on lõplik ehk väljakujunenud ületemperatuur, $J_{üa}$ algne ületemperatuur. Jahtumisel $J_{ül} = 0$ ning

$$J_{ü} = J_{üa} e^{-t/t}. \qquad (2.12)$$

Juhul kui soojusülekanne keskkonda puudub näiteks väga lühiajalise intensiivse soojenemise korral, siis on soojussiirdetegur $A = 0$ ning võrrandi (2.10) lahendiks

$$J_{ü} = \frac{P}{C} t. \qquad (2.13)$$

Võrrandiga (2.10) kirjeldatavatest protsessidest annab ülevaate joonis 2.55. Konstantse kaoenergia voo ja soojussiirdeteguri korral iseloomustavad temperatuuri ajalist muutumist eksponentsiaalsed kõverad. Aja jooksul ($t = 3\tau$) läheneb ületemperatuur väljakujunenud väärtusele.

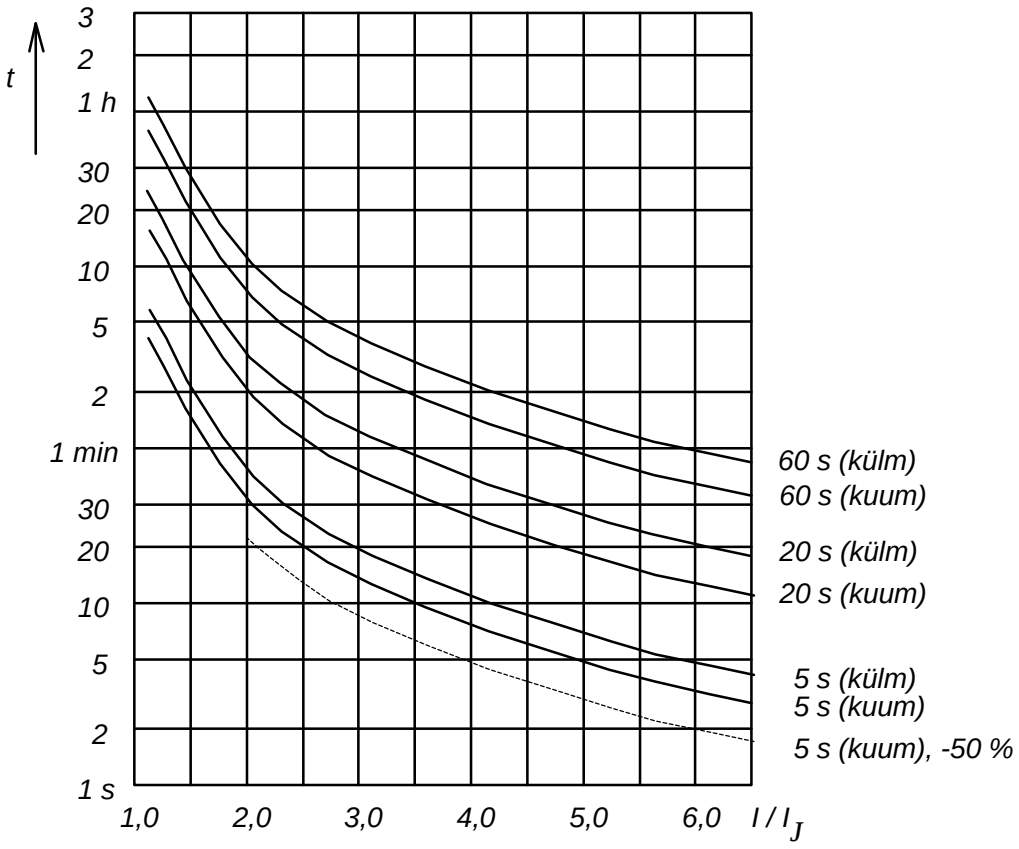


Joonis 2.55. Elektrimootori soojenemiskõveraid

Muutlik koormus põhjustab mootori vahelduvat soojenemist ning jahtumist. Pöörleva ja seisva mootori jahtumise intensiivsus on erinev. Samuti erinevad 2...3 korda vastavad soojuslikud ajakonstandid.

Kaitseaparaadi tööd ülekoormusel iseloomustavad kõige paremini joonisel 2.56 näidatud aja-voolu tunnusjooned. Need on koostatud lähtudes mootori soojusmudelist ning

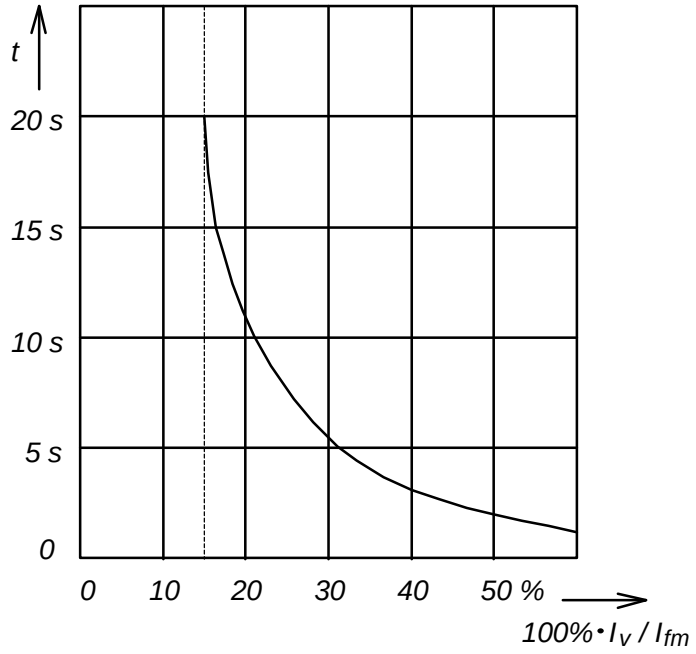
arvestavad täiskoormusvoolu, mootori soojusmahtuvust, akumuleerunud soojusenergiat, jahtumistingimusi ning nende muutumisi rootori paigalseisu ja vääratuse korral. Programmeerimisel antakse ette täiskoormusvool ja rakendumisaeg 6-kordse täiskoormusvoolu juures t_{6x}. Viimase valiku aluseks on maksimaalne oodatav käivitusaeg, s. t $t_{6x} > t_{käiv}$. Mootori soojenemisel väheneb ülekoormuse taluvusaeg. Seepärast erinevad kõverad külma ja kuuma mootori jaoks. Nn külma kõvera korral on mootori vool enne ülekoormuse algust võrdne nulliga $I = 0$, kuuma kõvera korral $I = 0{,}9\ I_u$. Vääratuse korral on väikese pöörlemissageduse tõttu jahtumistingimused tunduvalt halvemad. Seepärast vähendatakse rakendumisaega 50 % võrra. Vastav tunnusjoon on näidatud joonisel 2.56 punktiirjoonena ning lihtsuse mõttes ainult ühe sätte korral. Kui mootor töötab, arvestab kaitseaparaat pidevalt salvestunud soojusenergiat (I^2dt), mistõttu üleminek külmalt kõveralt kuumale ja vastupidi on pidev protsess, s. t tunnusjoone asukoht muutub sõltuvalt mootori eelnevast koormusest.



Joonis 2.56. Kaitseaparaadi aja-voolu tunnusjooned

Mittesümmeetrilise voolu kaitse peab rakenduma juhul, kui voolu vastujärgnevus-komponendi I_v suhteline väärtus kasvab üle lubatud piiri, näiteks siis kui $I_v / I_{fm} > 0{,}15$, kus I_{fm} on suurim faasivool. Rakendumisaja saab määrata joonisel 2.57 toodud graafiku

järgi. Väikeste voolude korral, kui $I < 0{,}25\ I_u$, pole voolude mittesümmeetria ohtlik ning kaitse ei toimi.



Joonis 2.57. Mittesümmeetrilise voolu kaitse aja-voolu tunnusjooned

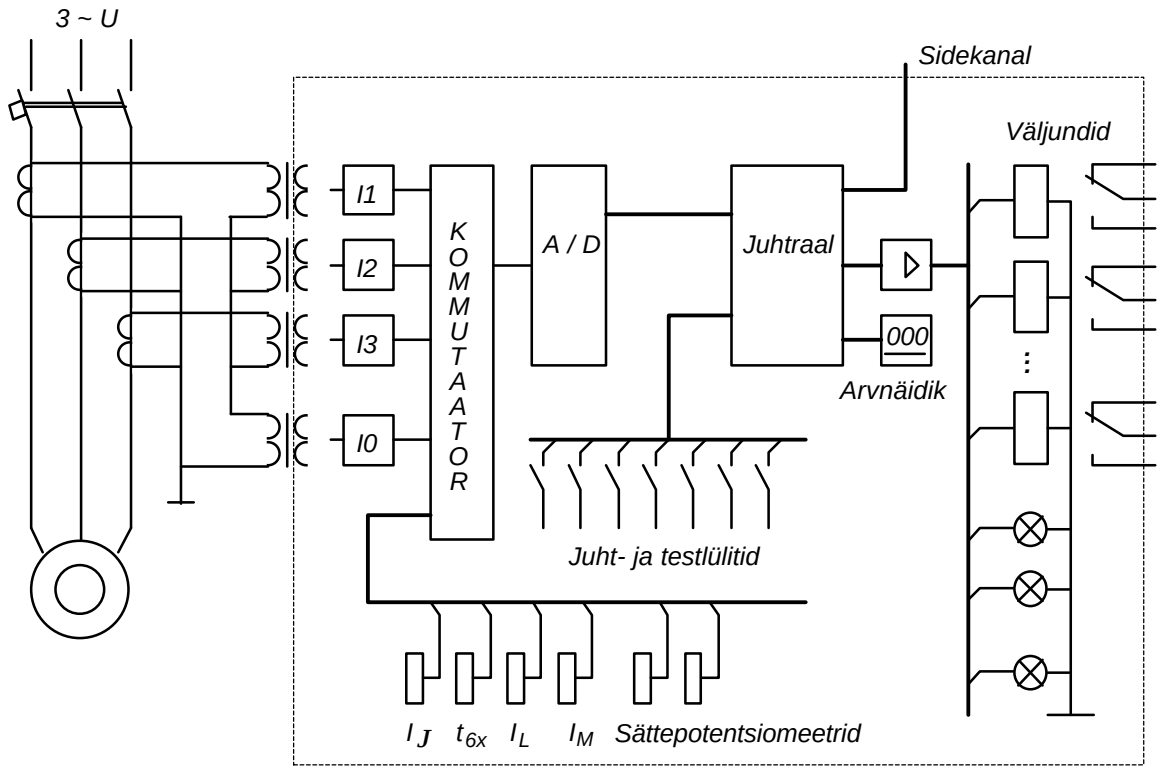
Kui isolatsioon rikneb, tekib maaühendusvool ning mootori mähised hakkavad täiendavalt soojenema. Maaühendusvoolu kaitse rakendumisläveks võib valida I_m > (0,05...0,8) I_n, rakendumisajaks aga 50 ms ... 1 s. Suurte voolude $I > 4\ I_u$ korral maaühendusvoolu kaitse töö blokeeritakse, sest sel juhul toimib lühisekaitse.

Ohtlik võib olla ka koormuse järsk vähenemine, mis viitab mehaaniliste sõlmede purunemisele (näiteks on purunenud mootorit ja töömasinat ühendav sidur). Kui koormusvool väheneb järsku alla lubatud piiri, lülitatakse mootor mõne sekundi jooksul välja.

Programmeeritavatel kaitseaparaatidel võib olla veel muidki ülesandeid. Näiteks mõõtetulemite näidustamine, salvestamine ja edastamine, andmetöötlus jms. Programmeeritava kaitseaparaadi lihtsustatud plokkskeem on joonisel 2.58.

Aparaadi sisendisse antakse mootori faasivoolud ja nulljärgnevusvool. Pärast filtreerimist lähevad need signaalid kommutaatori ja analoog-digitaalmuunduri kaudu mikroarvutisse. Samuti suunatakse sinna kõik kaitse tööks vajalikud sätted. Sätete etteandmiseks kasutatakse potentsiomeetreid või klahvistikku. Võrreldes sätete arvväärtuste numbrilise etteandmisega on potentsiomeetrite eeliseks käsitsemise lihtsus, samuti on sätted pidevalt näha potentsiomeetrite nuppude asendiga. Kõikidel programmeeritavatel kaitseaparaatidel

on olemas mõõtetulemite arvnäidik ning informatsiooni väljastamise liides teistele juhtraalidele.

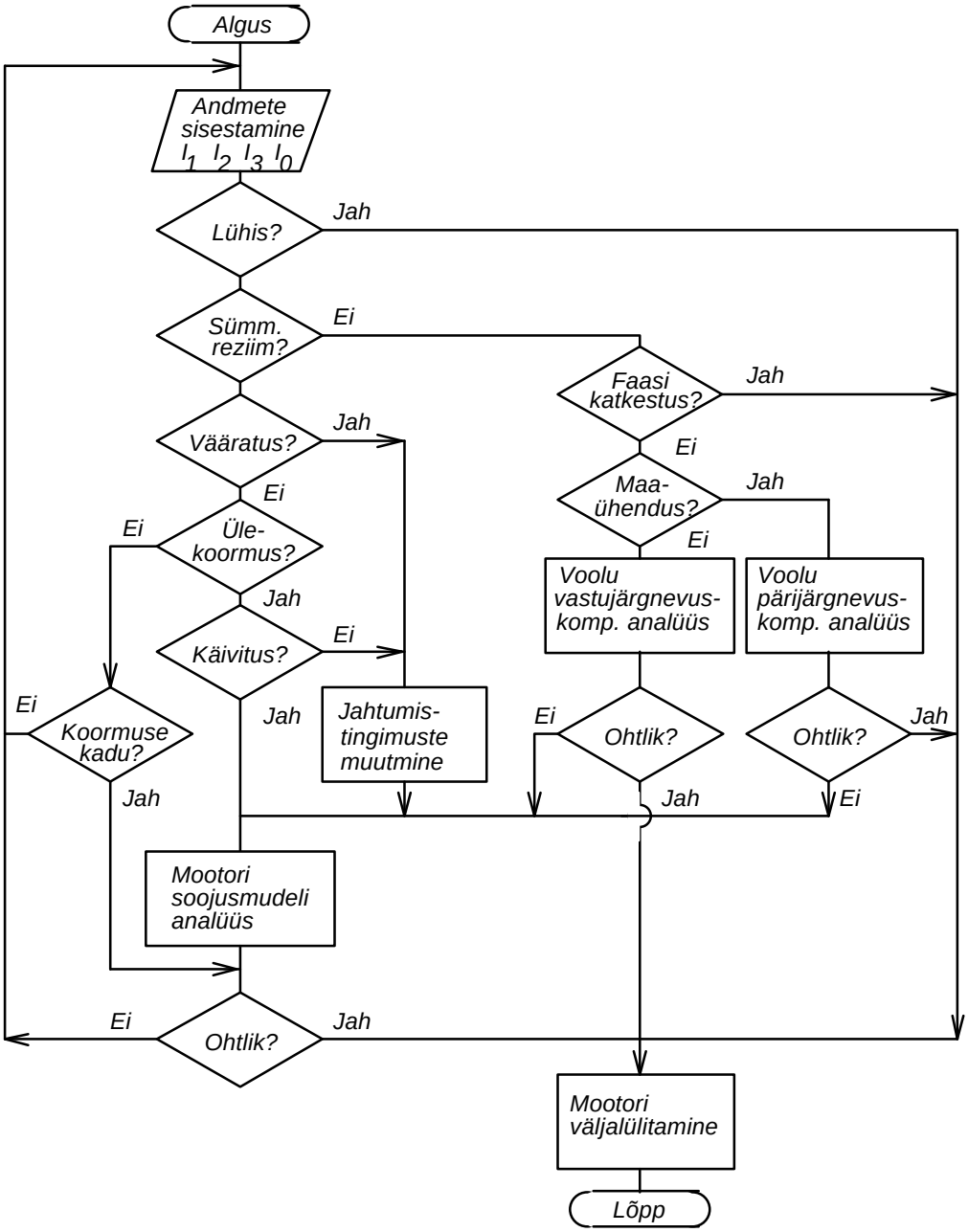


Joonis 2.58. Asünkroonmootori mikroprotsessor-kaitseaparaadi plokkskeem

Programmeeritava kaitseaparaadi tööd iseloomustab tema talitlusalgoritm. Ühe võimaliku algoritmi lihtsustatud plokkskeem on joonisel 2.59. Pärast andmete sisestamist analüüsitakse mootorivoolusid ning võrreldakse neid erinevate rež iimide sätetega. Rež iimi tuvastamise järel analüüsitakse selle ohtlikust ning otsustatakse kas tööd jätkata või mootor välja lülitada.

Teatud juhtudel tuleb analüüsi käigus lahendada küllalt keerukaid ülesandeid. Näiteks võib tuua rakendumisaja arvutamise ülekoormusel, mida iseloomustavad joonisel 2.56 näidatud tunnusjooned. Seejuures on konkreetseteks arvutusteks ja tulemi leidmiseks mitmeid võimalusi. Sageli kasutatakse raaljuhtimisega süsteemides mittelineaarsete funktsioonide arvutamiseks nn tabelimeetodit. Varem arvutatud funktsiooni väärtused on sel juhul arvuti mälupesades ning nende leidmine toimub aadressi järgi. Konkreetse meetodi valik sõltub kaitselt nõutavast toimekiirusest, lahendatava ülesande keerukusest ning kasutatava protsessori tehnilistest näitajatest.

Nüüdistehnika areng kulgeb erinevate seadmete ja masinate, sealhulgas elektrimootorite üha tihedama integreerimise kaudu terviklike tehnoloogiliste komplekside loomise suunas. Raaljuhtimisega kaitseaparaadid muutuvad nende süsteemide lahutamatuks osaks.



Joonis 2.59. Asünkroonmootori programmeeritava kaitseaparaadi talitlusalgoritmi plokkskeem