Ü. KAASIK
A. KORJUS
I. KULL

PROGRAMMEERIMINE

Ü. KAASIK, A. KORJUS, I. KULL

PROGRAMMEERIMINE

XII A-31227
-766

Ü. KAASIK
A. KORJUS
I. KULL

PROGRAMMEERIMINE

KIRJASTUS «VALGUS» TALLINN 1971

51
K 02

Kunstiliselt kujundanud P. Reeveer

Lubatud Eesti NSV Kõrgema ja Kesk-erihariduse Ministeeriumi poolt kasutada õppevahendina Eesti NSV kõrgemates koolides.

УДК 519.95
Каазик Ю., Корьюс А., Кулль И.
Программирование
1971

Книга познакомит читателя с математическими основами работы, с конструкцией и возможностями наиболее распространенных в ЭССР электронно-вычислительных машин «Минск-2», «Минск-22», «Минск-32» и «Урал-4». В ней подробно излагается система команд, методика и приемы программирования для машин «Минск-22», «Урал-4» и дается описание международного алгоритмического языка АЛГОЛ-60.

Изложение материала сопровождается многочисленными примерами применения команд и приемов программирования.

Книга предназначена для подготовки специалистов по программированию в вузах, но является полезной в качестве методического учебного пособия для всех лиц, осваивающих программирование (в том числе для студентов техникумов и учащихся старших классов средних школ).

Tartu Riikliku Ülikooli
Raamatukogu

2—2—3
15—70

SISSEJUHATUS

Elektronarvutid (ehk kasutades täpsemat terminit — programmjuhtimisega arvutid) on omandanud asendamatu koha kaasaegse ühiskonna elus. Neid rakendatakse teaduslikus uurimistöös, informatsiooniteenistuses, tehnikas, tootmise organiseerimisel ja tootmisprotsesside automatiseerimisel, majanduse planeerimisel ning paljudel teistel aladel. Arvutite rakendusvaldkond laieneb iga aastaga arvutite hulga suurenemise, arvutite võimsuse, aga ka vastavate spetsialistide — arvutusmatemaatikute-programmeerijate kaadri kiire kasvu tõttu.

Meie vabariigis alustati arvutusmatemaatikute-programmeerijate ettevalmistamist 1957. a. spetsiaalkursuste lugemisega Tartu Riiklikus Ülikoolis. Praegu moodustavad arvutusmatemaatikud juba põhiosa TRÜ Matemaatikateaduskonna üliõpilastest. Programmeerijaid valmistatakse meil ette aga ka keskkoolide ja tehnikumide baasil, vastavaid kursusi loetakse samuti Tallinna Polütehnilises Instituudis ja Eesti Põllumajanduse Akadeemias.

Meie vabariigi arvutusmatemaatikute-programmeerijate ettevalmistamine on senini olnud raskendatud, sest on puudunud eestikeelne programmeerimise õpik. Tõsi küll, on antud välja mitmeid arvuteid ja programmeerimist käsitlevaid eestikeelseid teoseid, mis on kirjutatud aga rohkem populaarteaduslikel eesmärkidel (vt. näit. [2] ja [20] kirjanduse loetelus). Õpiku nõuetele vastab küll [26], kuid selles TRÜ rotaprindil väljaantud väikesetiraažilises teoses on käsitluse aluseks võetud arvuti «Ural-1», mis meie vabariigi kõrgemates koolides enam rakendamist ei leia.

Käesolev õppevahend tugineb autorite loengutele Tartu Riiklikus Ülikoolis ja Tallinna Polütehnilises Instituudis. Arvestades Eesti NSV-s kasutatavate arvutite põhitüüpe ning autorite kogemusi, on käsitluse aluseks valitud vaid kaks arvutitüüpi — «Ural-4» ja «Minsk-22». Põhitähelepanu on sealjuures pööratud aga just arvutile «Ural-4», mille baasil toimub programmeerijate ettevalmistamine TRÜ-s.

Raamatu I peatükk on sissejuhatava iseloomuga. Selles antakse ülevaade arvutite ajaloost ja liigitusest, samuti programmjuhtimisega arvutite ehituse ning töötamise üldistest põhimõtetest.

II peatükis käsitletakse peamiselt aritmeetikaga seotud küsimusi (arvutites kasutatavad arvusüsteemid, arvutuste teostamine arvutites) ja antakse lühiülevaade matemaatilise loogika ühest lihtsamast arvutussüsteemist — lausearvutusest.

III peatükis esitatakse arvutite «Ural-4» ja «Minsk-22» käskude süsteem.

IV peatükis vaadeldakse mitmesuguseid nn. vahetu programmeerimise võtteid. Siin tutvustatakse programmi blokk-skeemi koostamist ja programmi koostamist niisuguse blokk-skeemi põhjal. Üksikasjalikult vaadeldakse tsükliliste arvutuste sooritamise võimalusi ja loogiliste skaalade kasutamist.

Kui IV peatükis vaadeldakse programmide vahetu koostamise võtteid (kus programmid koostatakse programmeerija poolt käskhaaval vastava arvuti käskude süsteemis), siis V peatükis käsitletakse juba võimalusi programmeerimise osaliseks automatiseerimiseks. Niisuguseid võimalusi annab standardprogrammide süsteemide ja koostavate programmide, aga samuti interpreteerivate programmide kasutamine. Samas peatükis kõneldakse ka programmi loogilise skeemi teisendamise metoodikast.

VI peatükis käsitletakse arvuti ja programmi kontrollimise küsimusi. Peale selle vaadeldakse siin üht arvuti kasutamise baasil suure tähtsuse omandanud arvutusmeetodit — statistiliste katsete ehk Monte-Carlo meetodit ning kõneldakse arvutusmeetodi valikuga seotud probleemidest üldse.

Suhteliselt iseseisva osa moodustab raamatu VII peatükk, kus tutvustatakse nn. automaatse programmeerimise põhimõtteid. Käsitluse aluseks on valitud algoritmiline keel ALGOL-60, millele toetub suurem osa Nõukogude Liidus kasutatavatest automaatse programmeerimise süsteemidest. Arvestades automaatse programmeerimise süsteemide pidevat täiustamist on selles peatükis vaid lühidalt tutvustatud üht Eesti NSV-s kasutusel olevat konkreetset süsteemi.

Õppevahend on eeskätt määratud Eesti NSV kõrgemate õppeasutuste nende osakondade üliõpilastele, kus programmeerimise kursus õppeplaani kuulub. Raamatut peaks aga mõningate vahelejätmistega olema võimalik kasutada ka tehnikumides ja mitmesugustel programmeerijate kursustel. Seda võimalust arvestades ongi suurem osa toodavatest näidetest valitud suhteliselt lihtsatena. Et raamat pole üldiselt mõeldud iseseisvaks õppimiseks, siis on materjal selle piiratud mahu tõttu esitatud üsna napisõnaliselt. Mis puutub raamatus kasutatavatesse terminitesse, siis on autorid püüdnud üldiselt jääda senise terminoloogia piiresse. Viimane

asjaolu ei tähenda muidugi aga olemasoleva (ja õppevahendis kasutatud) terminoloogia absolutiseerimist.

Õppevahendi üksikute osade kirjutamine jagunes autorite vahel järgmiselt. Ü. Kaasik kirjutas paragrahvid 25—32, A. Korjus paragrahvid 9—10, 12—16, 21—22 ja lisa, I. Kull paragrahvid 1—8, 20, 23—24. Peale nimetatute võtsid õpiku kirjutamisest osa veel V. Allsalu, kes kirjutas paragrahvid 17—19 ja A. Jaeger, kes kirjutas paragrahvi 11 (koos Ü. Kaasikuga). Tuleb märkida, et raamatu teksti mõningale ühtlustamisele vaatamata erinevad eri autorite poolt kirjutatud osad oma stiili ja käsitlusviisi poolest. Need erinevused ei peaks aga oluliselt raskendama raamatu kasutamist.

Lõpuks kasutavad autorid võimalust, et avaldada südamlikku tänu füüs.-mat. kandidaatidele R. Jürgensonile ja L. Kivistikule õpiku käsikirja läbivaatamise ning asjalike parandusettepanekute eest. Samuti täname sm. H. Tera, H. Arbi ja M. Vanemat käsikirja vormistamisel nähtud vaeva eest.

Autorid

I PEATÜKK

ÜLDISI TEATMEID ARVUTITEST JA PROGRAMMEERIMISEST

§ 1. ARVUTITE LIIGITUSEST. PROGRAMMJUHTIMISEGA ARVUTITE EHITUSE JA TÖÖTAMISE PÕHIMÕTETEST

1. Tänapäeval on loodud väga palju erineva konstruktsiooniga arvutustehnilisi seadmeid. Nende hulgas on oma suurte võimaluste ja laia rakendusvaldkonna tõttu kesksel kohal universaalsed programmjuhtimisega arvutid. Enne aga kui asume programmjuhtimisega arvutite üksikasjalikumale käsitlemisele, kõneleme lühidalt arvutitest ja arvutustehnilistest seadmetest üldse, nende liigitusest jms.

Kõigepealt tekib muidugi küsimus, mida nimetada arvutiks või arvutustehniliseks seadmeks. Et täpset definitsiooni on siin küllaltki raske anda, siis kõneleme ainult sellest, kuidas neid mõisteid tavaliselt käsitatakse. Üldiselt võib öelda, et *arvutite* all mõistetakse seadmeid, mis on konstrueeritud arvutuste hõlbustamiseks või kiirendamiseks, arvutusprotsesside kas osaliseks või täielikuks automatiseerimiseks.

Eelnevas lauses on esitatud ainult arvuti kõige põhilisemad omadused. Need pole aga arvuti mõiste piiritlemiseks kaugeltki ammendavad. Arvuti mõiste täpsustamisel tuleks arvesse võtta veel mitmeid teisigi aspekte, nagu näiteks vastava seadme iseloomu, kasutamise eesmärke jms. Nii näiteks ei loeta arvutiteks matemaatilisi tabeleid ja nomogramme (arvjooniseid), neid nimetatakse lihtsalt arvutusvahendeiks. Joonestusvahenditega (mõõtjoonlaud, sirkel jt.) on teatavasti samuti võimalik lahendada väga mitmesuguseid matemaatilisi ülesandeid. Et aga joonestusvahendeid kasutatakse põhiliselt siiski teistel eesmärkidel, ei loeta ka neid arvutiteks. Arvutiteks ei loeta samuti loendajaid ja arvesteid (voolumõõtjad, taksomeetrid jt.), kuigi need sooritavad teatavaid aritmeetilisi tehteid. Põhjuseks on jälle nende seadmete teistsugune kasutamisvaldkond ja eesmärk: suuruste automaatne mõõtmine ja registreerimine. Eelarvete, palgalehtede ja teiste raamatupidamisdokumentide koostamisel kasutatakse mitmeid spetsiaalseid masinaid nagu summeerimis-, raamatu-

pidamis-, faktuurmasinaid jt. Silmas pidades nende masinate küllaltki spetsiaalseid eesmärke jäetakse ka need mõnikord arvutite hulgast välja. Enamasti loetakse neid spetsiaalseteks arvutiteks.

Kaasaegsed universaalsed programmjuhtimisega arvutid lubavad lahendada ka niisuguseid ülesandeid, kus pole tegemist arvutamisega selle sõna tavalises tähenduses (näiteks informatsiooni otsimine või tõlkimine ühest keelest teise). Tõlkimise ja teiste informatsioonilis-loogiliste ülesannete lahendamiseks on loodud samuti spetsiaalseid seadmeid, mis töötavad programmjuhtimise põhimõttel, samuti nagu universaalsed arvutid. Niisuguseid seadmeid loetakse arvutiteks vastavalt sellele, kas nad on sobivad ka arvutamiseks. Jaataval juhul võib neid seadmeid (peale spetsiaalnimetuste kasutamise — tõlkimismasin, informatsioonimasin jne.) nimetada spetsialiseeritud arvutiteks. Eitaval juhul võib nende puhul kasutada aga ainult üldnimetust *arvutustehniline seade*, mis on arvuti mõistest laiem. Analoogiline on lugu ka juhtimismasinatega. Vastavalt nende sobivusele arvutamiseks kõneldakse kas juhtimis-arvutusmasinatest või lihtsalt juhtimismasinatest, kusjuures esimest liiki seadmeid võib lugeda arvutiteks, teisi tuleb lugeda aga juba arvutustehnilisteks seadmeteks.

Viimastel aastatel on hakatud veel looma paljudest arvutitest koosnevaid arvutite süsteeme, mille puhul kasutatakse enamasti üldnimetust arvutustehniline seade, mitte aga arvuti. Arvutite ja arvutustehniliste seadmete hulka ei loeta aga üldiselt roboteid ega programmjuhtimisega tööpinke, kuigi nende juhtimisseade kujutab endast informatsiooni töötlevat automaati. Viimase liigituse põhjuseks on asjaolu, et need seadmed pole ette nähtud ainuüksi informatsiooni ümbertöötamiseks, vaid teatavate *füüsiliste* tööoperatsioonide sooritamiseks.

Mis puutub arvutustehnilise seadme ja arvuti mõiste omavahelisse vahekorda, siis, nagu eespool juba nimetatud, on arvutustehnilise seadme mõiste laiem. Arvutustehniliseks seadmeks on kõik arvutid, aga ka arvutite iseseisvad osad (seadmed), nagu näiteks salvestusseadmed. Arvutustehnilisteks seadmeteks loetakse üldse kõik informatsiooni töötlevad programmjuhtimisega seadmed (kusjuures nende põhiülesandeks ei tohi olla *füüsiliste* tööoperatsioonide sooritamine)[1], olenemata sellest, kas nad sobivad arvutamiseks või mitte.

2. Parema ülevaate saamiseks kaasajal kasutatavatest arvutitest esitame nende põhimõttelise liigituse. Arvutite liigitusi (nagu üldse mistahes objektide liigitusi) võib anda mitmesuguseid, olenevalt liigituse aluseks olevatest tunnustest ja liigituse

[1] Niisuguseid seadmeid on eestikeelses kirjanduses hakatud nimetama ka *raalideks*.

eesmärkidest. Kui liigituse eesmärgiks on ainult arvutite põhirühmade eristamine, siis võib liigituse aluseks valida alljärgnevad tunnused.

1) Kas informatsiooni salvestamine toimub arvuti vastavate seadmete fikseeritud ja üksteist kindlalt eristatavate seisundite abil (nagu näiteks arvelaual või aritmomeetril), või kasutatakse selleks pidevalt muutuvaid füüsikalisi suurusi (pikkused, nurgad, pinged jms.) nagu arvutuslükatis, planimeetris jt. analoogilistes seadmetes. Esimesel juhul kõneleme *numbrilisest* ehk *diskreetse tegevusega* (*toimega*) arvutist, teisel juhul aga *pideva tegevusega* (*toimega*) arvutist Tänapäeval on ehitatud ka arvuteid, kus kasutatakse mõlemat informatsiooni salvestamise moodust. Niisuguseid arvuteid nimetatakse *hübriidseteks arvutiteks*.

2) Arvuti töö automatiseerimise aste, mis sõltub sellest: a) kas algandmete viimine arvutisse on automatiseeritud, b) kas arvutuskäiku kirjeldava programmi viimine arvutisse on automatiseeritud, c) kas arvutuste käik arvutis on automatiseeritud pärast algandmete ja programmi ettevalmistamist ja arvutisse viimist). Tähendame muuhulgas, et programmjuhtimisega arvuti korral on tingimused a), b) ja c) kõik täidetud.

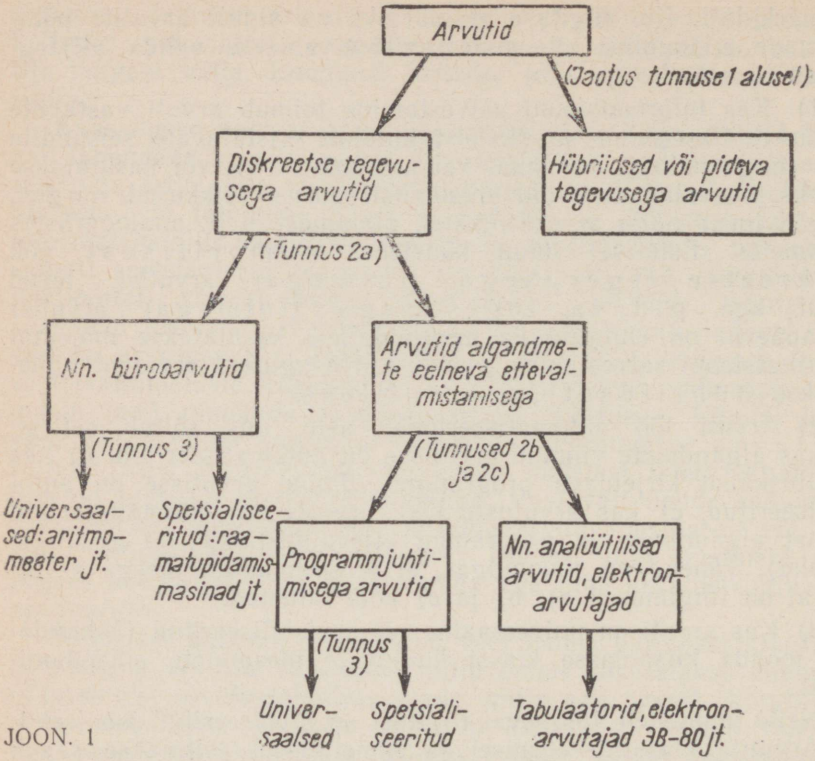
3) Kas arvuti on universaalne või spetsialiseeritud (kohandatud mõnda kitsamasse klassi kuuluvate ülesannete lahendamiseks).

Neile tunnustele tuginevat liigitust on illustreeritud joonisel 1.

Arvutite klassifitseerimisel ja nimetamisel esitatakse sageli ka füüsikalise protsessi liik või liigid (resp. vastavates protsessides kasutatavad elemendid), mis on arvuti töö aluseks. Põhimõtteliselt võib arvuti töö aluseks olla mistahes materiaalne (füüsikaline, keemiline, bioloogiline) protsess, kui see kulgeb küllaldase kiirusega ja on vajalikul määral juhitav. Nii on loodud mehhaanilisi, elektrilisi, elektro-mehhaanilisi, hüdraulilisi, pneumaatilisi, elektroonilisi (elektronarvuteid) jt. arvuteid. Uuritakse ka optiliste arvutite konstrueerimise võimalusi. Kuid kõik need lisanimetused iseloomustavad arvuti tööd vaid n.-ö. tehnilisest küljest, informatsiooni kodeerimise ja ümbertöötamise protsesside materiaalse kandja seisukohalt, ega kõnele midagi selle töö organiseerimise tasemest. Näiteks võib elektro-mehhaanilisel printsiibil konstrueerida nii aritmomeetreid kui ka programmjuhtimisega arvuteid. Joonisel 1 esitatud liigituses on aluseks võetud olulisem aspekt, nimelt informatsiooni ümbertöötamise protsessi iseloom arvutis.

3. Vaatleme nüüd programmjuhtimisega arvutite ehituse ja töötamise põhimõtteid.

Kaasaegsed programmjuhtimisega arvutid kujutavad endast keerulist elektro-mehhaanika- ja elektroonikaseadmete komplekti



JOON. 1

Teatavate arvuti juurde kuuluvate seadmete ülesandeks on lahendamisele tuleva ülesande algandmete ja lahendusprogrammi (kokkuvõtlikult, l ä h t e i n f o r m a t s i o o n i) ettevalmistamine selle informatsiooni arvutisse viimiseks. Lähteinformatsiooni ettevalmistamine seisneb tavaliselt selle andmestiku perforeerimises spetsiaalsetele kaartidele (nn. perfokaartidele) või lintidele (perfolindid, telegraafilindid). Et vastavad seadmed (perforaatorid, telegraafiaparaadid jm.) pole arvuti teiste seadmetega vahetult seotud, siis nimetatakse neid kõiki arvuti v ä l i s t e k s s e a d m e t e k s.

Arvuti teised seadmed on omavahel informatsioonikanalitega seotud ja moodustavadki õieti arvuti selle sõna otseses mõttes. Kirjeldame allpool nende seadmete ülesandeid.

S i s e n d s e a d m e t e ülesandeks on väliste seadmete abil ettevalmistatud lähteinformatsiooni automaatne arvutisse viimine.

S a l v e s t u s s e a d m e t e (ehk m ä l u) ülesandeks on arvutisse sisseviidud informatsiooni säilitamine. Salvestusseadmetele esitatakse kaks põhilist nõuet: seade peab võimaldama salvestada suurt hulka informatsiooni ja võimaldama vajaliku informat-

siooni kiiret leidmist. Neid nõudeid on kõige lihtsam realiseerida mitte ühe, vaid mitme üksteisega seotud salvestusseadme korral. Nii on kaasaegsetes programmjuhtimisega arvutites üks salvestusseade kiirelt töötav, aga väiksemahuline — nn. operatiivne salvestusseade (nimetatakse ka operatiivmäluks ehk sisemiseks mäluks) —, teised, suuremahulised, aga aeglasemad — nn. mitteoperatiivsed ehk välised salvestusseadmed (nimetatakse ka väliseks mäluks). Tähendame, et aritmeetilisi ja loogilisi tehteid sooritatakse ainult operatiivses salvestusseadmes olevate arvudega. Tehete sooritamiseks välistes (mitteoperatiivsetes) salvestusseadmetes olevate arvudega tuleb need esmalt üle kanda operatiivsesse salvestusseadmesse.

Aritmeetilises seadmes sooritatakse kõik aritmeetilised, aga samuti ka loogilised tehted. Nagu juba eespool öeldud, võetakse tehete sooritamiseks vajalikud lähteandmed operatiivsest salvestusseadmest. Tehete tulemused viiakse reeglina samuti operatiivsesse salvestusseadmesse kas automaatselt või spetsiaalse käsu toimel, olenevalt arvuti konstruktsioonist.

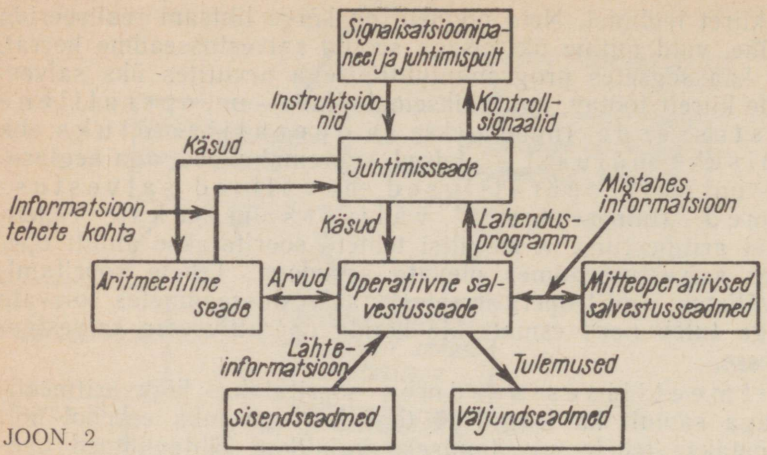
Väljundseadmete ülesandeks on lõpptulemuste väljastamine arvutist. Neid seadmeid on tavaliselt mitu, vastavalt informatsiooni väljastamise moodustele (arvude trükkimine, arvude ja teksti trükkimine, perforeerimine jne.).

On ilmne, et kõik eelnimetatud seadmed peavad ülesande lahendamisel tegutsema kooskõlastatult. Selle tagamiseks on arvutil spetsiaalne juhtimisseade, mis juhib kõikide teiste seadmete tööd vastavalt ülesande lahendamise programmile. Tänu sellele seadmele ongi võimalik seda liiki arvutite töö täielik automatiseerimine. Et arvuti töö aluseks on ülesande lahendamise programm, siis on arusaadav ka nende nimetamine programmjuhtimisega arvutiteks.

Ehk küll arvuti töö võib toimuda ja üldiselt toimubki lähteinformatsiooni sisseviimisest kuni lõpptulemuse väljastamiseni automaatselt, on siiski vaja seda tööd jälgida, teatavatel juhtudel seda katkestada, teha selles parandusi jms. Kõike seda võimaldavad signalisatsioonipaneel ja juhtimispult, millega arvutusmatemaatik-operaator kindlustab oma üldjuhtimise kogu (automatiseeritud) arvutusprotsessi üle.

Esitame äsjakirjeldatud seadmete vahekorra joonisel 2. Tähtsamad informatsioonikanalid on joonisel kujutatud ühe- või kahepoolsete nooltega, mis märgivad informatsiooni kulgemise suundi.

4. Nagu juba eespool tähendasime, on programmjuhtimisega arvuti töö aluseks ülesande lahendamise programm, s. t. ülesande lahendamise spetsiaalsel viisil kirja pandud eeskiri. Niisuguste programmide koostamise ehk programmeerimise süstemaatiline käsitlemine ongi käesoleva õpiku sisu. Selles



JOON. 2

paragrahvis pole meil aga võimalik programmide koostamisel veel pikemalt peatuda. Ütleme vaid niipalju, et programmeerimisel võetakse aluseks ülesande sõnastuses ja valitud lahendusmeetodis esinevad matemaatilised valemid ning muud vahekorrad. Üldiselt öeldes sarnaneb programmeerimine inimestele mõeldud arvutuseeskirjade koostamisega, ainult et siin peavad need eeskirjad olema rangelt formaliseeritud, pedantselt täpsed ja ammendavalt üksikasjalikud (kõik juhtumid peavad olema arvesse võetud, kõik sammud ette näidatud).

Informatsiooni ümbertöötamise eeskirju nimetatakse *algoritmideks*. Arvuti programmi ja algoritmi mõisted on seega väga lähedased, kuid neil on ka teatavaid erinevusi. Nii näiteks eeldab programmi mõiste mingi arvuti olemasolu, algoritm seda aga ei eelda. Programm peab võimaldama lõpptulemuse saamist mingisuguse praktiliselt vastuvõetava ajavahemiku jooksul, algoritmilt seda aga ei nõuta ja ta võib mõningatel juhtudel lõpptulemust üldse mitte anda. Öeldust ilmneb, et algoritmi mõiste on programmi mõistest üldisem: iga programm on ühtlasi ka algoritm, mitte aga vastupidi [2].

Järgnevalt kirjeldamegi programmi ehitust, eriti aga seda, kuidas arvuti programmi alusel töötab.

---

[2] Algoritmidega seotud probleeme käsitleb nn. *algoritmide teooria*, kus vaadeldakse mitmesuguseid algoritmide eriliike, nende omadusi, konstrueerimise võimalusi jms. Üheks olulisemaks probleemiks on siin ülesannete algoritmilise lahendatavuse või mittelahendatavuse küsimuse uurimine. Et iga algoritmiliselt lahendatav ülesanne on põhimõtteliselt lahendatav ka programmjuhtimisega arvutil ja et ükski algoritmiliselt mittelahendatav ülesanne (neid on aga tänapäeval kogunenud juba suur hulk) pole lahendatav ühegi programmjuhtimisega arvuti abil, siis võib öelda, et algoritmide teooria uurib arvutite ja programmeerimise äärmisi võimalusi.

5. Ülesande lahendamise programm koosneb käskudest ja on formaalselt võttes lihtsalt lõplik käskude jada

$$K_1, K_2, K_3, \ldots, K_n.$$

K ä s k on kõige lihtsam instruktsioon, mida saab arvutile ette kirjutada ja mida arvuti saab (automaatrežiimis töötades) täita.[3] Olenevalt arvuti ehitusest võivad käsud olla mitmesuguse struktuuriga. Näiteks võivad nad olla kujul

$T\ a_1$ (nn. ü h e a a d r e s s i l i s e arvuti korral),
$T\ a_1\ a_2$ (nn. k a h e a a d r e s s i l i s e arvuti korral),
$T\ a_1\ a_2\ a_3$ (nn. k o l m e a a d r e s s i l i s e arvuti korral).

Siin tähistab T nn. t e h t e k o o d i (ehk o p e r a t s i o o n i k o o d i), mis määrab arvutis sooritamisele tuleva tehte (liitmine, lahutamine vms.) Sümbolitega a_1, a_2 ja a_3 on aga tähistatud operatiivse salvestusseadme nende pesade[4] numbreid (nn. a a d r e s s e), kust tuleb võtta tehte sooritamiseks vajalikud arvud.

Olgu sooritamisele tulev käsk näiteks kujul Ta_1, kusjuures T võib olla näiteks mõne aritmeetilise tehte koodiks. Tehte koodi T alusel valmistatakse aritmeetiline seade ette vastava tehte sooritamiseks. Samaaegselt otsitakse operatiivse salvestusseadme pesast a_1 vajalik arv. Teine tehtest osavõttev arv on üheaadressiliste arvutite korral juba aritmeetilises seadmes. Pärast pesast a_1 leitud arvu viimist aritmeetilisesse seadmesse sooritatakse nõutud tehe. Tehte tulemus jääb aritmeetilisesse seadmesse ja seda võib kasutada vahetult järgneva tehte sooritamisel. Kui tehte tulemust on vaja säilitada, tuleb see vastava käsuga viia salvestusseadme mingisse pesasse.

Mitmeaadressiliste arvutite korral on käskude täitmises mõningaid erinevusi. Nii otsitakse operatiivsest salvestusseadmest ka teine tehtest osavõttev arv (teise aadressi põhjal) ja salvestatakse automaatselt tehte tulemus (olenevalt arvutist kas teise või kolmanda aadressi järgi).

Programmi käskude täitmine toimub üldiselt loomulikus järjekorras, s. t. pärast mingit käsku K_i sooritatakse üldiselt öeldes käsk K_{i+1}. Erandiks on ainult teatavat eri tüüpi käsud — s u u n a m i s k ä s u d. Üht tüüpi suunamiskäskude, nn. t i n g i m a t u s u u n a m i s e käskude ülesanne seisneb selles, et näidata, missugune programmi käsk tuleb täita järgmisena. Veelgi olulisemad on aga teist tüüpi suunamiskäsud, nn. t i n g i t u d s u u n a m i s e käsud, mis annavad juhtimise edasi ühele kahest etteantud

[3] Tähendame, et peale käskude saab arvutile anda mõningaid instruktsioone veel juhtimispuldilt. Neid instruktsioone ei saa kasutada programmide koostamisel.

[4] P e s a k s nimetatakse salvestusseadme kindlat osa, mis on arvuti konstruktsiooniga ette nähtud ühe arvu (või käsu) säilitamiseks.

käsust, olenevalt eelmise tehte tulemusest. Selgitame seda pisut lähemalt. Nimelt moodustatakse arvutis enamiku käskude sooritamisel veel teatavad abisignaalid (nn. signaalid ω, φ jt.). Abisignaalide tekitamise tingimused olenevad tehetest ja on igas arvutis täpselt fikseeritud. Näiteks antakse signaal ω mitmetes arvutites aritmeetiliste tehete puhul neil juhtudel, kui tehte tulemus on negatiivne, täpsemalt:

$$\omega = \begin{cases} 1, & \text{kui tehte tulemus on negatiivne,} \\ 0, & \text{kui tehte tulemus on mittenegatiivne.} \end{cases}$$

Signaali ω alusel annab tingitud suunamise käsk, olgu see näiteks kujul Ta_1a_2, juhtimise kas pesas a_1 või a_2 asuvale käsule järgmiselt: kui pärast eelmise käsu täitmist kehtib tingimus $\omega = 0$, siis antakse juhtimine pesas a_1 asuvale käsule, vastupidisel juhul (s. t. $\omega = 1$ korral) antakse juhtimine aga pesas a_2 asuvale käsule. Analoogiliselt toimivad tingitud suunamise käsud ka üheaadressilistes, kolmeaadressilistes jt. programmjuhtimisega arvutites.

Et kõikide vähegi keerulisemate ülesannete puhul sõltub lahenduskäik (lahendusmeetodi erijuhtude ehk variantide valik jne.) vahetulemustest, siis on arusaadav, et niisuguste ülesannete lahenduskäigu täielik automatiseerimine poleks ilma tingitud suunamise käskudeta üldse võimalik.

Arvutuste automatiseerimise ja programmeerimise hõlbustamise seisukohalt on väga oluline kavandada arvutuskäik võimaluse korral selliselt, et üht ja sama programmiosa kasutatakse arvutuste käigus korduvalt. Sellistel juhtudel väheneb tunduvalt programmi pikkus ja programmi koostamiseks kulutatav aeg. Niisuguseid korduvalt kasutatavate osadega programme on võimalik koostada suunamiskäskude abil, kuid selleks otstarbeks on arvuti käskude süsteemis tavaliselt ette nähtud ka spetsiaalsed käsud.

Peale mainitud käskude on arvuti käskude süsteemis veel mitmesuguseid teisi käske, näiteks loogiliste tehete (konjunktsioon ehk osa eraldamine, disjunktsioon, võrdlemine, nihutamine jt.) sooritamiseks, arvude teisendamiseks ühest arvusüsteemist teise, informatsiooni vahetamiseks üksikute salvestusseadmete vahel, informatsiooni väljastamiseks, arvuti töö kontrollimiseks ning organiseerimiseks jne.

Kõik need käsud koosnevad üldiselt samuti tehte koodist ja nn. aadressiosast, milles on kodeeritud käsu täitmiseks vajalik lisainformatsioon. Nende käskude täitmine arvutis toimub umbes samuti nagu eespool kirjeldatud juhul: tehte koodiga valmistatakse ette arvuti vastavad seadmed, aadressiosast leitakse tehte sooritamiseks vajalik lisainformatsioon, mille järel sooritatakse automaatselt käsuga ettekirjutatud protseduur.

Tähendame, et arvuti käskude süsteemi ühes mõningate programmeerimise reeglitega võib vaadelda kui teatava formalisee-

ritud keele (arvuti käskude süsteemi keele ehk lihtsalt arvuti käskude keele) süntaksit. Arvuti programmi võib seega vaadelda kui arvutuseeskirja või algoritmi, mis on esitatud selle arvuti käskude keeles. On selge, et programmide käskhaaval koostamine nõuab arvuti käskude keele põhjalikku tundmist. Kuid peale selle on kasulik teada ka programmeerimist lihtsustavaid ja kergendavaid võtteid ning meetodeid, mida käsitleme üksikasjalikumalt juba õpiku järgnevates peatükkides.

§ 2. ÜLEVAADE ARVUTITE AJALOOST

1. Käesolevas ülevaates saame arvutite ajaloost esitada ainult olulisemaid jooni ja momente. Mõnevõrra üksikasjalikumalt püüame siiski näidata programmjuhtimisega arvutite loomist ja arengut.

Arvutite ajalugu ulatub tagasi väga kaugetesse aegadesse. Täpsemaid andmeid on muistsete arvutite kohta tuua muidugi väga raske. Tõenäoliselt loodi need üheaegselt teatava elementaarse arvutusoskuse levimisega. Nii on näiteks teada, et vanas Kreekas ja Roomas, aga samuti ka idamaadel kasutati arvutamisel spetsiaalseid arvutuslaudu (nn. abakusi), mis on meie praeguste arvelaudade eelkäijateks.

On alust arvata, et vanal ajal kasutati ka tunduvalt komplitseeritumaid arvuteid. Nimelt leidsid käsnapüüdjad 1900. a. Egeuse merest mehhanismi, mille vanus on umbes kaks tuhat aastat. Spetsialistide arvates oli see mehhanism mingi vana-aja arvutusmasina koostisosaks.

Täpsemaid andmeid on võimalik esitada alates XVII sajandist. Nii leiutas šoti matemaatik John Napier (1550—1617) 1617. a. seadme (nn. Napieri arvutuspulgad), mille abil oli võimalik korrutada mitmekohalisi arve ühekohalistega. Nimetatud seadet kasutati Euroopas XVII sajandil juba võrdlemisi laialdaselt. Hiljem leiutati veel mitmeid teisi seadmeid, mis töötavad analoogiliselt Napieri arvutuspulkadega.

1642. a. konstrueeris prantsuse füüsik ja matemaatik Blaise Pascal (1623—1662) arvuti, mida sai kasutada mitmekohaliste arvude liitmiseks ja lahutamiseks. Pascali isa töötas maksukogujana ja poeg ehitas nimetatud masina just maksude kokkuarvutamise hõlbustamiseks.

1666. a. konstrueeris Samuel Morland masina, mis sarnanes Pascali omale, kuid võimaldas ka arvude korrutamist.

Arvutustööde mehhaniseerimine huvitas ka G. W. Leibnizi (1646—1716), kes 1673—1674. a. projekteeris arvuti, millega võis sooritada kõiki nelja aritmeetilist tehet. Aritmomeeter valmistati tema enda poolt 1694. a. Huvitav on märkida, et Leibniz mõistis selgesti kahendsüsteemi eeliseid kümnendsüsteemiga võrrel-

des. Ta avaldas isegi mõtte, et kahendsüsteem tõrjub tulevikus välja kümnendsüsteemi.

Mitmesuguseid suuremal või vähemal määral erinevaid aritmomeetri konstruktsioone on XVIII—XIX sajandil välja töötatud terve rida. Tootma hakati aritmomeetreid aga alles XIX saj. 20.-ndatel aastatel. Üheks kõige täiuslikumaks mehhaaniliseks aritmomeetriks oli Peterburgi inseneri V. T. Odhneri poolt 1874. a. konstrueeritud arvuti. Seda aritmomeetrit hakati tootma Venemaal 1891. aastal ja see levis laialdaselt kogu maailmas.

Üksikasjalikumalt aritmomeetrite konstrueerimise ajalool peatumata märgime siinkohal veel seda, et hiljem kasutati aritmomeetri mehhanismide käivitamiseks elektrimootoreid ja automatiseeriti täielikult üksikute tehete sooritamine. Kaasajal ehitatakse aritmomeetreid ka releede ja elektroonika-elementide baasil.

2. Möödunud sajandi 80-ndatel aastatel konstrueeris ameerika insener Herman Hollerith (1860—1926) uut tüüpi arvutusmasinad (nn. analüütilised arvutusmasinad), kus kasutatakse perfokaarte. Idee kasutada informatsiooni salvestamiseks perfokaarte polnud iseenesest uus. Niisuguseid kaarte kasutas juba prantslane Jacquard aastal 1795 oma automaat-kudumistelgedes, kus perfokaartide aukude abil kodeeriti informatsioon kangamustri kohta.

Analüütiliste arvutite komplekti kuuluvad tänapäeval mitmesugused spetsialiseeritud ülesannetega masinad: perforaatorid (nende abil perforeeritakse algandmed vastavatele kaartidele), kontrollimismasinad, sorteerimismasinad (on ette nähtud perfokaartide sorteerimiseks vastavalt kaartidele perforeeritud tunnustele), reproduktorid (kaardimassiivi dubleerimiseks), tabulaatorid (need teostavad kindlates veergudes olevate arvude liitmist ja lahutamist; töökiirus 6—9 tuhat kaarti tunnis), korrutav perforaator (liidab, lahutab ja korrutab; 6—7-kohaliste arvude puhul 700—1000 korrutamist tunnis) jne. Arvutuse tulemusi võib trükkida või ka perforeerida, et neid kasutada edaspidistes arvutustes. Arvutuste üksikasjalik käik määratakse ühendustega vastaval kommutatsioonipaneelil.

Analüütilised arvutid on sobivad niisuguste ülesannete lahendamiseks, kus tuleb ümber töötada suuri algandmete hulki lihtsate ja šabloonsete eeskirjade kohaselt. Niisuguste arvutuste näiteks on rahanduslikud arvutused (palgaarvutused), arvutused kaubanduses (laoaruanded), statistilised arvutused (rahvaloenduse kokkuvõtted) jms. Nii näiteks kasutati analüütilisi arvutumasinaid rahvaloenduse andmete läbitöötamiseks 1890. a. USA-s ja 1897. a. Venemaal. Analüütiliste arvutitega on võimalik sooritada ka komplitseeritumaid arvutusi (koostada näiteks funktsiooni väärtuste tabeleid), kuid tänapäeval pole see enam otstarbekohane, silmas pidades programmjuhtimisega arvutite olemasolu.

Analüütiliste arvutite puuduseks on nende suhteliselt väike töökiirus, samuti asjaolu, et arvutite programm ei või olla keeruline ja seda ei saa arvutuste käigus automaatselt muuta. Samuti pole arvutused täielikult automatiseeritud, sest vahetulemused tuleb arvutisse sisse viia ikkagi inimese poolt.

3. Numbriliste (ehk diskreetse tegevusega) arvutite kõrval loodi aga ka hoopis teisel põhimõttel töötavaid nn. pideva tegevusega arvuteid. Nende masinate tööpõhimõte seisneb vastavate matemaatiliste vahekordade modelleerimises mingisuguste pidevalt muutuvate füüsikaliste suuruste abil. Seda liiki arvutite näiteks võib tuua arvutuslükati, joone kõveruse mõõtja, kurvimeetri (seade kõvera kaare pikkuse mõõtmiseks), planimeetri (seade tasapinnaliste kujundite pindala mõõtmiseks), integraatori (seade integreerimiseks), integraafi (seade funktsiooni mingi algfunktsiooni joonestamiseks), integrimeetri (seade numbriliseks integreerimiseks), harmoonilise analüsaatori (seade funktsiooni Fourier' kordajate leidmiseks), diferentsiaalanalüsaatori ehk integreeriva masina (seade harilike algtingimustega diferentsiaalvõrrandi süsteemide lahendamiseks).

Sellist tüüpi arvuteid hakati konstrueerima juba XVII saj. alates. Nii konstrueeriti arvutuslükati Edmund Gunteri poolt 1624. a., mida seejärel täiustati mitmel korral ja mis sai tänapäevase kuju möödunud sajandi keskpaiku. Esimene integraaf ehitati Krakovi matemaatiku B. Abdank-Abakanovitši poolt 1879. a. Diferentsiaalanalüsaatori konstrueerimise teoreetilised alused pärinevad W. Thomsonilt (Kelvin), kes lähtus oma venna J. Thomsoni 1875. a. ehitatud integraatorist. Üks esimesi mehhaanilisi diferentsiaalintegraatoreid ehitati 1904. a. A. N. Krõlovi poolt. Esimene diferentsiaalanalüsaator USA-s ehitati 1930—1931. a. V. Bushi poolt. Üks esimesi Stieltjes-planimeetreid (seade integraali $\int_a^b f_1(x)\,df_2(x)$ arvutamiseks) ehitati A. N. Krõlovi poolt 1911. a. Tänapäeval ehitatakse peale mehhaaniliste veel mitmesuguseid elektrilisi ja elektroonikaseadmetel baseeruvaid pideva tegevusega arvuteid.

Pideva tegevusega arvutite puhul saame resultaadi 3—4 õige tüvenumbriga, kusjuures täpsuse suurendamine on seotud oluliste tehniliste komplikatsioonidega. Arvutuste suhteliselt väike täpsus ongi pideva tegevusega arvutite üks olulisi puudusi. Mainime, et arvutuste täpsust saab diskreetse tegevusega arvutite puhul suurendada ilma eriliste põhimõtteliste raskusteta.

Pideva tegevusega arvutite teiseks iseloomulikuks jooneks on see, et nad on rakendatavad ainult teatavate kindlate ülesannete tüüpide lahendamiseks, s. t. on enam või vähem spetsialiseeritud arvutid. Väiksemad pideva tegevusega arvutid (kurvimeetrid, planimeetrid) on seejuures rangelt spetsialiseeritud. Suuremad pideva tegevusega arvutid (näiteks kaasaegsed diferentsiaalanalüsaato-

rid) koosnevad üldiselt reast blokkidest, millest igaüks on ette nähtud mingisuguseks operatsiooniks (näit. korrutamiseks või integreerimiseks). Ülesande lahendamiseks tuleb need blokid ühendada vastavalt ülesande iseloomule. Lahendatavate ülesannete tüübid määratakse seega blokkide komplektiga. Tähendame, et blokkide ühendusi ei saa ülesande lahendamise käigus automaatselt muuta.

4. Nagu teame, on teaduslik-tehniliste probleemide lahendamiseks sagedasti vaja teha küllaltki suuremahulisi arvutusi, millega kaasneb veel rida nõudeid, nagu nõue arvutuste suure täpsuse ja kiiruse kohta, samuti nõue, et arvutuskäiku oleks võimalik automaatselt muuta, olenevalt vahetulemustest. Kõiki neid nõudeid rahuldab täiel määral ainult numbriline programmjuhtimisega arvuti.

Numbrilise programmjuhtimisega arvuti idee ühes vastavate üksikasjalike projektidega esitati inglise matemaatiku, Cambridge'i ülikooli professori Charles Babbage'i (1792—1871) poolt XIX saj. esimesel poolel.

Esmalt projekteeris Babbage 1822. a. arvutusmasina (nn. vahede masina), mis võimaldas automaatselt arvutada funktsiooni väärtusi $f(x_0 + kh)$ $(k = 1, 2, \ldots)$ lähtudes väärtustest

$$f(x_0),\ \Delta f(x_0),\ \Delta^2 f(x_0),\ \ldots,\ \Delta^n f(x_0),$$

eeldusel, et kõrgemat järku vahed $\Delta^n f(x)$ on konstantsed.

Esimese niisuguse masina ehitamist (kus $n = 7$ ja opereeriti 20-kohaliste arvudega) alustati 1823. a. Töö katkes aga 1833. a., kuna tekkisid tõsised raskused selle edasise finantseerimisega. Hiljem ehitati vahede-masin ($n = 4$, arvud 14-kohalised) valmis Rootsis ning seda demonstreeriti 1855. a. Londonis ja Pariisis.

1833. a. töötas Babbage välja aga uue arvutusmasina projekti. Babbage ise nimetas seda arvutit *analüütiliseks masinaks*. Samuti nagu esimene arvuti, oli ka see mehhaaniline, kuid oma konstruktsiooni põhimõtete poolest sarnanes ta kaasaegsete programmjuhtimisega arvutitega. Masin koosnes järgmistest seadmetest.

1. Aritmeetiline seade (Babbage'i terminoloogias *vabrik*). Tehete sooritamise kiirust hinnati selliselt: liitmine või lahutamine 1 sek., korrutamine või jagamine 1 min.

2. Salvestusseade (Babbage'i järgi *arvude ladu*). Seade kavandati tuhande 50-kohalise arvu salvestamiseks.

3. Seade arvuti töö juhtimiseks, kaasa arvatud arvude ülekandmine ühest *pesast* teise.

4. Seadmed arvude masinasse viimiseks ja väljastamiseks (trükkimine või perforeerimine).

Arvuti töö toimus automaatselt programmi alusel, mis salvestati perfokaartidel. Selleks oli ette nähtud kaks mehhanismi: üks perfokaartide jada määras teostamisele tulevad tehted, teine arvude

aadressid. Masin võimaldas teostada ka tingitud suunamisi ja tsüklilisi arvutusi. Programmeerimise kohta analüütilisel masinal ilmus 1843. a. artikkel.

Arvuti ise jäi aga valmis ehitamata. Üksikasjalikku kirjeldust oma masina projekti kohta Babbage ei publitseerinud, küll on aga säilinud masina seadmete detailsed joonised. Nende põhjal ehitas Ch. Babbage'i poeg aastatel 1880—1910 ühe osa aritmeetilisest seadmest ja arvutas selle abil arvu π kordsed 20 kohaga.

5. XIX saj. lõpul said laialdasema leviku osaliseks aritmomeetrid, hakati kasutama ka tabulaatoreid, Babbage'i ideed ning eksperimendid unustati. Programmjuhtimisega arvuti konstruktsiooni põhimõtted avastati teistkordselt alles sajandi võrra hiljem. Nimelt alustati 1937. a. Harvardi ülikoolis (USA) Howard H. Aikeni juhtimisel programmjuhtimisega arvuti Mark-1 (ehk *Automatic Sequence Controlled Calculator*, lühendatud nimetusega ASCC) ehitamist. Arvuti sai valmis 1944. a. Arvuti oli põhiliselt mehhaaniline, kuigi selles kasutati ka elektromagneetilisi releesid ja lüliteid. Arvuti salvestusseade mahutas 72 kahekümnekolme kohalist kümnendsüsteemi arvu, konstantide mälu (arvud, mida ei saanud arvutuste käigus automaatselt muuta) sisaldas 60 arvu. Arvuti tööd juhiti paberilindile perforeeritud programmi abil. Tehete sooritamise kiirus oli järgmine: liitmine ja lahutamine 0,3 sek., korrutamine 6 sek., jagamine 11,4 sek.

Esimene täielikult releedel töötav arvuti ehitati Belli laboratooriumis aastatel 1938—1940, kuid see oli spetsialiseeritud. Universaalne releedel töötav arvuti valmistati samas laboratooriumis 1944. a. Selles kasutati 9000 telefonireleed ja hulk teletaipe. Salvestusseade mahutas 44 seitsme tüvenumbriga arvu (arvuti töötas liikuva koma režiimis). Tehete sooritamise kiirust iseloomustavad järgmised arvud: liitmine ja lahutamine 0,3 sek., korrutamine 1 sek., jagamine 2,2 sek.

Esimestest suurtest releearvutistest tuleks nimetada veel saksalase K. Zuse arvutit ja Stokholmis ehitatud arvutit BARK.

Esimeseks arvutiks, kus kasutati elektroonikaseadmeid (elektronlampe) oli ENIAC (*Electronic Numerical Integrator and Calculator*), mis konstrueeriti J. P. Eckerti ja I. W. Mauchly poolt ning ehitati Moore'i elektriinseneride koolis Philadelphias (USA-s) 1943.—1946. a. Arvutis kasutati 18 000 elektronlampi ja 1500 releed. Elektroonikaseadmete kasutamise tõttu oli tehete sooritamise kiirus sellel arvutil tunduvalt suurem kui releearvutil. Nii võttis liitmine ja lahutamine aega 0,2 msek. (5000 tehet/sek.), korrutamine 2,8 msek. (u. 360 tehet/sek.), jagamine 6 msek. (u. 166 tehet/sek.), kusjuures tehted sooritati 10-kohaliste kümnendsüsteemi arvudega. Arvuti operatiivne salvestusseade mahutas ainult 20 kümnekohalist arvu, konstantide mälu aga 300 arvu. Tehete sooritamise suhteliselt suure kiiruse paremaks ärakasuta-

miseks ei olnud otstarbekohane arvuti tööd juhtida paberilindile või perfokaartidele perforeeritud programmi abil (nagu see oli arvuti Mark-1 puhul). Siin salvestati programm ühenduste abil umbes nii nagu seda tehakse tabulaatoritel, kusjuures peale selle kasutati veel spetsiaalseid lüliteid. Kõik see võimaldas arvutuste summaarset aega arvutis vähendada, kuid tegi programmeerimise, eriti aga programmi masinasse viimise küllaltki kohmakaks.

Arvuti ENIAC oli esialgu kavandatud ballistiliste arvutuste tegemiseks, kuid hiljem lahendati tema abil ka muid ülesandeid, näiteks osatuletistega diferentsiaalvõrrandeid.

Programmjuhtimisega arvutite ajaloost kõneldes ei saa märkimata jätta ka John v. Neumanni seisukohti arvutite konstrueerimise põhimõtete kohta, mis avaldati 1945. a. Olulisim tema ettepanekutes seisnes järgmises. Käske tuleb kodeerida ja vaadelda kui teatavaid formaalseid arve. Need tuleb salvestada operatiivses salvestusseadmes samadel alustel arvudega. See võimaldab täielikumalt ära kasutada tehete teostamise suure kiiruse ja võimaldab ka programmi automaatset muutmist arvutuste käigus ja arvuti enese poolt. Kõik see nõuab operatiivse salvestusseadme mahu suurendamist (suurusjärguni 1000 pesa). Arvutis on otstarbekas kasutada kümnendsüsteemi asemel kahendsüsteemi.

Esimene arvuti, kus rakendati J. v. Neumanni ettepanekuid, oli EDVAC (*Electronic Discrete Variable Automatic Computer*), mis ehitati Moore'i elektriinseneride koolis.

Umbes samal ajal (a. 1946—1949) ja samadel põhimõtetel ehitati esimene elektronarvuti — EDSAC (*Electronic Delay Storage Automatic Computer*) ka Euroopas. See toimus M. V. Wilkes'i juhtimisel Cambridge ülikooli matemaatikalaboratooriumis. Arvuti iseloomustamiseks toome järgmised andmed. Arvuti sisaldas umbes 3000 elektronlampi. Operatiivne salvestusseade ehitati elektroakustiliste elavhõbeda viiteahelate abil. Selle mahtuvus oli 1024 seitsmeteistkümne kohalist kahendarvu (nn. pika pesa moodustasid kaks kõrvutiasuvat naaberpesa). Üks käsk mahutati samuti ühte 17-kohalisse pessa. Käsud olid üheaadressilised. Arve vaadeldi diapasoonis $-1 < x < 1$ ja neid viidi masinasse perfolindi abil. Tehete sooritamise kiirus oli masinas selline: liitmine 1,5 msek. (670 tehet/sek.) ja korrutamine 4 msek. (250 tehet/sek.).

6. Nõukogude Liidus alustati elektronarvutite projekteerimist 1947. a. Esimene elektronarvuti МЭСМ (Малая электронная счетная машина) ehitati akadeemik S. A. Lebedevi juhtimisel Kiievis Ukraina TA Matemaatikainstituudis 1951. a. Arvutis kasutati 5300 elektronlampi. Töökiirus oli tal väike, näiteks 50 liitmistehet sekundis. Arvutil МЭСМ teostati Volžski hüdroelektrijaama kõrgepingeliinide ehitamiseks vajalikud arvutused.

1952. a. valmis (samuti S. A. Lebedevi juhtimisel) NSVL TA

Täppismehaanika ja Arvutustehnika Instituudis juba võimsam arvuti БЭСМ (Большая электронная счетная машина), mis oli kaua aega suurimaks arvutiks Euroopas. Toome ka selle arvuti kohta mõned olulisemad andmed. Arvutis kasutati 4000 elektronlampi, 5000 pooljuhtdioodi ja umbes 200 tuhat ferriitsüdamikku. Salvestusseadmeid on mitu: operatiivne salvestusseade (ferriitsüdamikel) mahutab 2048 arvu, salvestusseade magnettrumlitel (trumleid on 2) mahutab 10 240 arvu, salvestusseade magnetlintidel 120 000 arvu. Pesa pikkus on 39 kahendkohta. Masinas kujutatavad arvud kuuluvad vahemikku $10^{-9} < x < 10^{+9}$ (masin töötab liikuva koma režiimis). Käsud on kolmeaadressilised. Arvutuste sooritamise kiirus on 8—10 tuhat tehet sekundis. Arvuti on universaalne ja teda kasutati mitmesuguste teaduslik-tehniliste ning majanduslike ülesannete lahendamiseks.

1952. a. sügisel valmis NSVL TA Juhtimismasinate ja -süsteemide Laboratooriumis akadeemia korrespondentliikme I. S. Bruki juhtimisel arvuti M-2. Viimane on kolmeaadressiline masin, töökiirusega 2000 tehet sekundis. Hiljem konstrueeriti kaheaadressiline arvuti M-3, töökiirusega 30 tehet sekundis (kui operatiivsalvestusseade ehitati ümber ferriitidele, tõusis töökiirus 2000 tehteni sekundis). 1958. a. konstrueeriti kolmeaadressiline arvuti M-20, mis sooritab 20 000 tehet sekundis.

1953. a. konstrueeriti J. J. Bazilevski juhtimisel kolmeaadressiline arvuti «Strela», mis sooritab 2—3 tuhat tehet sekundis.

1954. a. konstrueeriti B. I. Ramejevi juhtimisel üheaadressiline arvuti «Ural», mis sooritab 100 tehet sekundis. See on «Ural»-tüüpi arvutite *perekonna esiklaps*, millele järgnesid peatselt «Ural-2», «Ural-4» (mõlemate töökiirus on 5—8 tuhat tehet sekundis) ja veel mitmed teised seda tüüpi arvutid.

Praeguseni on kodumaises arvutustehnikatööstuses konstrueeritud ja tootmises juba suur hulk väga mitmesuguseid programmjuhtimisega arvuteid, mida kõiki on raske isegi nimetada. Märgime siin siiski mõningad enamlevinud arvutid.

«Minsk-2», «Minsk-22» (kaheaadressilised arvutid, töökiirus 5—6 tuhat tehet sekundis), «Ural-11» (üheaadressiline, 50 000 liitmist-lahutamist ja 350 korrutamist sekundis), «Ural-14» (üheaadressiline, 45 000 liitmist-lahutamist ja 3700 korrutamist sekundis), «Setun» (üheaadressiline, töötab kolmendsüsteemis, töökiirus 4500 tehet sekundis), «Kiiev» (kolmeaadressiline, töökiirus keskmiselt 10 000 tehet sekundis), «Dnepr» (kaheaadressiline, töökiirus 15 000 tehet sekundis), «Razdan-2» (kaheaadressiline, töökiirus 5000 tehet sekundis), «Razdan-3» (15—20 tuhat tehet sekundis), «Aragats» (umbes 20 tuhat tehet sekundis), «БЭСМ-3М», «БЭСМ-4» (keskmiselt 20 000 tehet sekundis), «М-220» (keskmiselt 20 000 tehet sekundis), «КВМ-1» (koordineeriv arvuti, teeb fikseeritud koma režiimis 100 000 liitmist sekundis), «ВНИИЭМ-3»

(universaalne juhtimis-arvutusmasin, üheaadressiline, töökiirus kuni 75 000 tehet sekundis).

Kodumaiste arvutite kõige kiiremaks esindajaks on «БЭСМ-6», mis sooritab keskmiselt 1 miljon tehet sekundis.

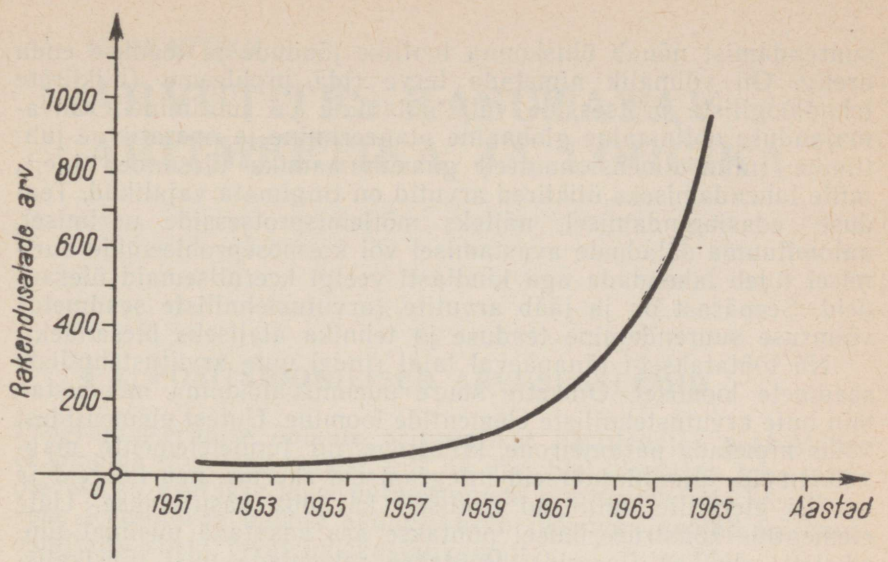
7. Pikemalt üksikute programmjuhtimisega arvutite juures peatumata kõneleme veel lühidalt tänapäeva arvutustehnika üldistest arengutendentsidest.

Kõigepealt tuleb tähendada, et arvutite tootmine ja rakendamine ning arvutustehnika areng toimub nii meil kui ka välismaal üha kiirenevas tempos. Nii näiteks oli 1952. a. kogu maailmas ainult mõnikümmend programmjuhtimisega arvutit, 60-nendate aastate algul oli selliseid arvuteid mõnekümne tuhande ümber, 1967. a. lõpul oli nende arv tõusnud aga juba 60 tuhandele. Kindlasti pakuvad huvi ka andmed arvutite hulga kohta üksikutes maades. Arvutustehniliste seadmete arvukuse seisukohalt oli 1967. a. lõpul esikohal USA, kus oli umbes 35 000 arvutit. Temale järgnesid Saksa Föderatiivne Vabariik 3500 arvutiga, Jaapan 3000 arvutiga, Inglis- ja Prantsusmaa, kus kummaski oli ligikaudu 2500 arvutit, NSVL-s 2000 arvutit, Itaalias ja Kanadas kummaski ligikaudu 1500 arvutit, Austraalias, Hollandis, Mehhikos, Rootsis, Šveitsis igaühes umbes 500 arvutit, Belgias ühes Luksemburgiga 350 arvutit, Taanis 250 arvutit, Austrias, Indias ja Norras igaühes ligikaudu 200 arvutit, Tšehhoslovakkias 150 arvutit jne.[5] Et kaasaegsed arvutid on küllaltki erinevate parameetritega, ei võimalda toodud andmed määrata veel antud maade arvutustehnilise baasi tegelikku võimsust, mida saaks palju täpsemini hinnata näiteks 1 sekundis sooritatavate liitmis- ja korrutamisoperatsioonide arvu alusel.

Praeguseks on programmjuhtimisega arvutite hulk veelgi kasvanud, kuna ülikoolid, polütehnilised instituudid, suuremad täppisteaduse- ja tehnika-alased instituudid, firmad ning tehased on enamasti loonud oma arvutuskeskused, mis on varustatud universaalsete või spetsialiseeritud arvutitega. Nii on Eesti NSV-s sedalaadi arvutuskeskusi juba kümmekond. Esimene arvutuskeskus Eesti NSV-s loodi 1959. a. Tartu Riiklikus Ülikoolis, järgmisel aastal loodi arvutuskeskus Eesti NSV TA Küberneetika Instituudis.

Ühes arvutite hulga suurenemisega on suurenenud loomulikult ka teostatavate arvutuste maht. Arvutustööde mahu suurenemine pole toimunud aga mitte ainult standardsete (ühetüübiliste) ülesannete massilisema lahendamise arvel. On toimunud midagi hoopis enamat: arvutite levik on stimuleerinud matemaatiliste ja loogiliste meetodite väljatöötamist ja rakendamist uute, aga ka klassikaliste probleemide ning ülesannete käsitlemiseks ja lahen-

[5] Andmed on võetud J. Říha artiklist «Počítače a pracovnici v automatizaci», Výběr informací z organizační a výpočetní techniky, 1968, 2, lk. 182—184. Andmed on seisuga 1967. a. detsember.



JOON. 3

damiseks, lühidalt, on aidanud kaasa teaduste *matematiseerimisele*. Matemaatika ja loogika meetodite uutest rakendusaladest võiks nimetada majandusteadust (majandusküberneetika), tõlkimist (masintõlge), automaatset programmeerimist, automaatset refereerimist, informatsiooni automaatset otsimist, mõtlemise uurimist (modelleerimist), diagnostikat, programmeeritud õpetamist jt. Mõnevõrra tinglikult kajastab seda nähtust joonis 3.

Arvutite hulga tormilisest kasvust on aga veelgi olulisem nende võimsuse kiire suurenemine ja ekspluatatsiooniomaduste paranemine. Nii kasvas perioodil 1943.—1955. a., kus arvutites kasutati peamiselt elektronlampe, arvutite töökiirus umbes 10^3 tehtelt sekundis 10^4 tehteni sekundis, operatiivsalvestusseadme mahtuvus aga 10^4 kahendkohalt 10^5 kahendkohale. Perioodil 1955.—1965. a., kus arvutites kasutati juba põhiliselt pooljuhtelemente ja ferriitsüdamikke, suurenes arvutite töökiirus 10^5 tehtelt 10^6 tehteni sekundis, kusjuures operatiivsalvestusseadme mahtuvus kasvas 10^6—10^7 kahendkohani. Märgime möödaminnes, et esimene arvuti töökiirusega suurusjärgus 1 miljon tehet sekundis — «Stretch» ehitati 60-ndate aastate algul USA-s. Üheks iseloomulikuks jooneks arvutustehnika arengus on veel arvutite gabariitide ja energiakulutuste vähenemine. Viimase asjaolu tõttu pole uuematel arvutitel enam vaja ka spetsiaalseid jahutusseadmeid.

Tähendame, et arvutite töökiiruse ja üldse võimsuse suurendamine ei toimu mitte *sportlikust huvist* asja vastu. Arvutustööde õige organiseerimise puhul kaasneb sellega ka arvutuste omahinna alanemine. Kuid veelgi olulisem on see, et arvutite võimsuse

suurendamist nõuab ühiskonna tootlike jõudude ja teaduse enda areng. On võimalik nimetada terve rida probleeme (ülikiirete tehnoloogiliste protsesside väljatöötamine ja juhtimine, rahvamajanduse optimaalne globaalne planeerimine ja operatiivne juhtimine, mitmedimensionaalsed gaasidünaamika ülesanded jne.), mille lahendamiseks ülikiired arvutid on tingimata vajalikud. Teaduse edasiarendamisel, näiteks mõtlemisprotsesside uurimisel, aatomituuma saladuste avastamisel või kosmoseprobleemide uurimisel tuleb lahendada aga kindlasti veelgi keerulisemaid ülesandeid. Seepärast on ja jääb arvutite (arvutustehniliste seadmete) võimsuse suurendamine teaduse ja tehnika alaliseks ülesandeks.

Nii töötataksegi tänapäeval laial rindel uute arvutustehniliste seadmete loomisel. Omaette suure uurimisvaldkonna moodustab siin uute arvutustehniliste elementide loomine. Uutest elementidest võiks nimetada parametrone, krüotrone, nn. tunnelelemente, magnetkelmeid, üliminiatuurseid integraalseid skeeme, neuristoreid ja optilisi elemente, milliseid osaliselt ka juba kasutatakse. Uute elementide konstrueerimisel püütakse ära kasutada uusimat füüsikalist tehnikat (laserid). Püütakse rakendada uusi füüsikaliskeemilisi protsesse, nähtusi ja omadusi, näiteks kristallide ja elementaarosakeste omadusi.

Uute arvutite loomine kulgeb muidugi väga mitmes suunas. Nii näiteks toimuvad mitmel maal uurimis- ja projekteerimistööd ülikiirete arvutite loomiseks töökiirusega mitte vähem kui 10^9 tehet sekundis. Üheks perspektiivseks suunaks ülikiirete arvutustehniliste seadmete loomisel on arvutite süsteemide konstrueerimine. Arvutuste teostamise suur kiirus saavutatakse sellistes süsteemides ülesande üksikute osade paralleelse (samaaegse) lahendamise teel. Üheks esimeseks arvutite süsteemiks Nõukogude Liidus on 16 arvutist koosnev süsteem, kus üksikarvutid on kas tüüpi «Minsk-2» või «Minsk-22». Väga paljulubavaks näib kujunevat optiliste arvutite loomine, kus peale arvutuste ülisuure kiiruse on võimalik veel tohutult suurendada salvestusseadmete mahtu. Peale selle on niisuguste arvutite puhul võimalik ehitada salvestusseadmed põhimõtteliselt täiesti uutel printsiipidel (nn. assotsiatiivsed salvestusseadmed), kus vajalik informatsioon otsitakse üles mitte pesa aadressi, vaid sisuliste tunnuste põhjal, nii nagu see toimub inimajus. Võib arvata, et seda tüüpi arvutite loomine toob kaasa olulisi muudatusi kogu ühiskonna intellektuaalses elus.

Arvutustehnika arenguga on tihedalt seotud ka programmeerimise teooria ja praktika areng. Nii on arvutite võimsuse kasv teinud paratamatuks ülemineku automaatsele programmeerimisele. Arvutite edasine areng toob kaasa aga uued võimalused, näiteks võimalused arvuti õppimiseks (või õpetamiseks) mingit ülesannet lahendama ja inimesele nii omase heuristilise mõtlemise imiteerimiseks.

II PEATÜKK

ARVUTITES KASUTATAV ARITMEETIKA JA LOOGIKA

§ 3. POSITSIOONILISED ARVUSÜSTEEMID

Programmeerimise õpikute aritmeetikat käsitlevates peatükkides ei kõnelda tavaliselt sellest, mis on arv. Seda loetakse juba teada olevaks ja asutakse kohe mitmesuguste arvusüsteemide käsitlemisele. Käesolevas õpikus püüame arvu mõistega seotud küsimusi selgitada siiski pisut üksikasjalikumalt.

Arvu mõiste käsitlemiseks on mõistagi mitmeid võimalusi. Valime siin sellise käsitlusviisi, mis on kooskõlas programmeerimise kursuse eesmärkide ja konstruktiivse iseloomuga. Selle käsitluse korral on vaja mõningaid abimõisteid, nagu tähestiku, tähe, sõna jt. sedalaadi mõisteid, mille defineerimisega alustamegi.

1. Tähestikuks nimetatakse mistahes lõplikku sümbolite hulka. Selle tähestiku sümboleid nimetatakse tähtedeks, täpsemini — vaadeldava tähestiku tähtedeks. Tähestiku näiteks võime tuua ladina tähestiku, vene tähestiku, tingmärkide tähestiku

$A = \{\triangle, \odot, \varnothing,$ $, \sigma, \varphi\}$

või tähestiku

$$B = \{O, I\}.$$

Tähestikus A on 6 tähte, tähestikus B aga 2 tähte: O ja I. Tähendame, et koma ja loogelised sulud ei kuulu tähestike A ja B tähtede hulka. Need on abisümbolid tähestiku tähtede eraldamiseks ja tähestiku enda esitamiseks.

Sõnaks mingis tähestikus nimetatakse selle tähestiku tähtede lineaarses järjekorras kirjutatud lõplikku jada (ilma komadeta või teiste vahemärkideta tähtede vahel), kusjuures selles jadas võivad esineda tähed ka korduvalt. Nii on ladina tähestiku sõnade näiteks sõnad

aritmeetika ja *tidirairairaa,*

tähestiku B sõna on aga näiteks

OIOIIIOIOOI.

Sõnadega opereerimisel tuleb kasutada veel sõnade ühenduse mõistet. Sõnade P ja Q ühenduseks nimetatakse sõna, mis saadakse siis, kui sõnale P kirjutatakse vahetult järele sõna Q. Sõnade P ja Q ühendust tähistatakse PQ abil. Analoogilist tähistusviisi kasutatakse ka rohkem kui kahe sõna ühendamisel.

Toome mõned näited sõnade ühenduse kohta. Olgu sõna P näiteks *tidirai*, sõna Q aga *rairaa*. Sõnade P ja Q ühendus PQ tuleb sel juhul *tidirairairaa*. Sõnade 25 ja 063 ühendamisel saame sõna 25063. Tähendame, et numbritest koosnevate sõnade ühendamist ei tohi segi ajada arvude liitmisega, need on täiesti erinevad tehted.

Vajalikuks osutub ka sõnade graafilise võrduse mõiste. Kaht sõna P ja Q nimetatakse graafiliselt võrdseteks, kui: 1) sõnadel P ja Q on samapalju tähti ja 2) sõnade P ja Q esimesed tähed on samad, sõnade P ja Q teised tähed on samad jne. kuni võrreldavate sõnade viimaste tähtedeni. Seega tähendab sõnade graafiline võrdus vastavate sõnade kui puhtgraafiliste objektide tähttähelist ühtimist. Sõnade P ja Q graafilist võrdust tähistatakse tavaliselt

$$P \doteq Q.$$

Mis puutub sõnade tähendusse, siis oleneb see konkreetsest interpretatsiooni- ehk tõlgenduse süsteemist. Nii näiteks vastab ladina tähestiku sõnale *king* ühes interpretatsioonisüsteemis — eesti keeles — üks tähendus, teises süsteemis — inglise keeles — aga hoopis teine tähendus (*kuningas*). Seega, kui ei ole antud mingisugust sõnade interpreteerimise süsteemi, saab neid vaadelda ainult puhtgraafiliste objektidena.

2. Arvu mõiste selgitamisel vaatleme kõigepealt naturaalarve (mittenegatiivseid täisarve). Naturaalarvu mõistet võib selgitada või defineerida muidugi mitmel viisil. Esitame siin ühe võimaluse. Lähtume tähestikust

$$B = \{O, I\}$$

ja defineerime naturaalarvud kui teatud erikujulised sõnad tähestikus B. Definitsioon on järgmine (nn. induktiivne definitsioon):

1. O on naturaalarv (nimetame seda naturaalarvuks null),
2. kui P on naturaalarv, siis P I (sõnade P ja I ühendus) on naturaalarv,
3. naturaalarvudeks loeme ainult neid sõnu, mis on konstrueeritavad punktide 1 ja 2 alusel.

Tähendame, et induktiivsel definitsioonil on ilmne konstruktiivne iseloom. Tõepoolest, definitsiooni punktid 1 ja 2 (definit-

siooni otsesed punktid) võimaldavad defineeritavaid objekte — naturaalarve — vahetult konstrueerida. Definitsiooni punkt 3 (kaudne punkt) väidab aga seda, et punktide 1 ja 2 abil saame konstrueerida kõik naturaalarvud. Viimasel ajal jäetakse induktiivsetes definitsioonides kaudne punkt tavaliselt esitamata, lugedes seda endastmõistetavaks.

Lähtudes toodud definitsioonist on naturaalarvudeks sõnad, mille esimene täht on «O» ja millele järgneb mingi lõplik jada tähtedest «I», näiteks

OI (nimetame seda naturaalarvuks üks),
OII (kaks), OIII (kolm), OIIII (neli) jne.

Naturaalarvudeks pole aga näiteks järgmised tähestiku *B* sõnad:

OO, OIOI, IIIIII.

Kahe naturaalarvu võrdsust (mille tähiseks on =) mõistame nende naturaalarvude kui tähestiku *B* sõnade graafilist võrdsust. Sõnade pikkuse võrdlemise abil on võimalik defineerida ka vahekorrad *suurem* ja *väiksem*. Samuti pole raske defineerida aritmeetilisi tehteid. Kahe naturaalarvu summa defineerime kui liidetavate naturaalarvude ühenduse, kusjuures teisel liidetaval tuleb jätta kirjutamata esimene täht «O».

Kahe naturaalarvu korrutise defineerime järgmiselt. Vastavalt esimese teguri igale tähele «I» tuleb üksteise järele välja kirjutada kõik teise teguri tähed «I». Sellisel viisil saadud sõnale tuleb lõpuks ette kirjutada täht «O», millega saamegi korrutise.

N ä i d e. Arvude OIII ja OIIII summa on OIIIIIII. Samade arvude korrutis on OIIIIIIIIIII.

Nagu näeme, on naturaalarvude liitmine ja korrutamine põhimõtteliselt väga lihtsad operatsioonid. Pole raske tõestada ka summa ja korrutise omadusi, näiteks, et arvu O liitmine annab tulemuseks teise liidetava, arvuga O korrutamine annab tulemuseks arvu O, arvuga OI korrutamine annab tulemuseks teise teguri, samuti summa ja korrutise kommutatiivsust, assotsiatiivsust jt. omadusi.

3. Selliste naturaalarvudega opereerimisel on ka omad raskused: naturaalarvud on väga vähe ülevaatlikud objektid, neid (eriti suuremaid arve) on raske üksteisest eristada, raske klassifitseerida (kas või paaris- ja paarituteks arvudeks), raskused tekivad nende (eriti suuremate arvude) suupärasel nimetamisel ja kuigi nendega opereerimine on põhimõtteliselt lihtne, osutub see ühtlasi ka küllaltki kohmakaks.

Kõige selle tõttu on otstarbekohane võtta kasutusele mingi sobivam naturaalarvude tähistamise süsteem. Selleks võib kasutada näiteks kõikvõimalikke sõnu mingis tähestikus

$$A = \{a_1, a_2, \ldots, a_p\},$$

kus $p \geqslant 2$. Konkreetsuse huvides võime tähestikuna A vaadelda näiteks tähestikku

$$A_1 = \{0, 1, 2, 3\},$$

kus $p = 4$.

Nüüd tuleb meil näidata, kuidas vastavalt etteantud naturaalarvule leitakse tema t ä h i s (sõna tähestikus A_1) ja ümberpöördult.

Esiteks lepime kokku, et neli (üldjuhul p) esimest naturaalarvu tähistame vastavalt tähestiku A_1 (A) tähtedega. Seega tähistame O tähega 0, OI tähega 1, OII tähega 2 ja OIII tähega 3.

Suuremate naturaalarvude puhul leitakse vastav tähis tähtede «I» järkjärgulise rühmitamise teel. Juhul, kui $p = 4$, tuleb rühmitada IIII-kaupa. Rühmitamisel on oluline arvestada nii *täisosa* kui ka *jääki*.

Selgitame järk-järgulist rühmitamist konkreetse näite korral. Olgu antud näiteks naturaalarv

OIIIIIIIIIIIIIIIII. (1)

Tähtede «I» järk-järguline rühmitamine IIII-kaupa annab järgmise tulemuse:

OIIIIIIIIIIIIIIIII.
I I I I
I

Kirjutades juurde sümboli «O», võime ütelda, et esimene rühmitamine andis täisosa OIIII ja jäägi OIII. Teine rühmitamine andis täisosa OI ja jäägi O. Kolmas rühmitamine andis täisosa O ja jäägi OI.

Rühmitamisel saadud jäägid OIII, O ja OI on kõik väiksemad kui OIIII ja neid saab vastavalt tähistada tähestikus A_1 tähtedega 3, 0 ja 1. Saadud jääkide tähised vastupidises järjekorras moodustavadki naturaalarvu (1) tähise

103.

Tähise ja vastava naturaalarvu vahele kirjutame lihtsalt võrdusmärgi, seega näiteks

OIIIIIIIIIIIIIIIII = 103.

Üldiselt (mingi $p \geqslant 2$ korral) toimub kogu protseduur järgmiselt. Vastavalt etteantud naturaalarvule P rühmitame tähed «I» selles sõnas p-kaupa, kusjuures saame mingi täisosa P_1 ja jäägi Q_1 ($0 \leqslant Q_1 < p$). Arvestades summa ja korrutise definitsiooni, pole raske veenduda, et kehtib võrdus

$$P = P_1 \cdot p + Q_1. \quad (2)$$

Kui $P_1 \geqslant p$, saame sama protseduuri jätkates

$$P_1 = P_2 \cdot p + Q_2 \qquad (0 \leqslant Q_2 < p)$$

. . . . . .

$$P_{n-2} = P_{n-1} \cdot p + Q_{n-1} \qquad (0 \leqslant Q_{n-1} < p).$$

Kui oletada, et P_{n-1} on esimene täisosa, mis on väiksem kui p, siis saame lõpuks

$$P_{n-1} = 0 \cdot p + Q_n \qquad (0 \leqslant Q_n < p).$$

Asendades võrduse (2) paremas pooles järk-järgult arvud P_1, P_2, . . . , P_{n-2} ja P_{n-1}, leiame

$$P = ((\ldots(Q_n \cdot p + Q_{n-1})p + \ldots + Q_3)p + Q_2)p + Q_1. \qquad (3)$$

Arvestades aritmeetiliste tehete omadusi ja kasutades võrdsete teguritega korrutise üleskirjutamiseks astendajaid, saame (3) kujul

$$P = Q_n \cdot p^{n-1} + Q_{n-1} \cdot p^{n-2} + \ldots + Q_2 \cdot p^1 + Q_1 \cdot p^0. \qquad (4)$$

Võrduse (4) alusel saame leida naturaalarvule vastava tähise ja ümberpöördult. Naturaalarvu P tähis moodustatakse kui sõna, mis koosneb jääkidele Q_n, Q_{n-1}, . . . , Q_2, Q_1 vastavatest tähestiku A tähtedest. Vastupidise teisenduse demonstreerimiseks leiame näiteks tähisele 231 vastava naturaalarvu. See toimub võrduse (4) alusel järgmiselt:

$$2\ 3\ 1 = \mathrm{OII} \cdot (\mathrm{OIIII})^{\mathrm{OII}} + \mathrm{OIII} \cdot (\mathrm{OIIII})^{\mathrm{OI}} + \mathrm{OI} \cdot (\mathrm{OIIII})^{\mathrm{O}} =$$

$$= \mathrm{OII} \cdot \mathrm{OIIIIIIIIIIIIIII} + \mathrm{OIII} \cdot \mathrm{OIIII} + \mathrm{OI} \cdot \mathrm{OI} =$$
$$= \mathrm{OIIIIIIIIIIIIIIIIIIIIIIIIIIIII} + \mathrm{OIIIIIIIIIII} + \mathrm{OI} =$$
$$= \mathrm{OIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII}. \qquad (5)$$

4. Arvude tähistamise süsteeme, mis põhinevad võrdusel (4), nimetatakse arvude tähistamise positsioonilisteks süsteemideks ehk, lühemalt, positsioonilisteks arvusüsteemideks. Et nende süsteemide olulisemaks tunnuseks on kasutusel olevate sümbolite (tähtede) arv p, tuleb täpsuse huvides kõnelda *positsioonilisest p-ndsüsteemist*. Kasutatavate sümbolite arvu p nimetatakse seejuures vastava positsioonilise arvusüsteemi aluseks. Nimetuse *positsiooniline* lisamine märgib asjaolu, et sümbolitele vastavate liikmete arvulised väärtused (vrdl. näit. liikmeid võrduse (4) või (5) paremal pool) sõltuvad selle sümboli asukohast (positsioonist) arvu tähises. Et käesolevas peatükis käsitletakse aga ainult positsioonilisi arvusüsteeme, siis jätame täiendi *positsiooniline* enamasti kirjutamata, kõneldes lihtsalt p-ndsüsteemist, q-ndsüsteemist jne.

Meile kõigile on hästi tuntud kümnendsüsteem, kus arvude tähistamiseks kasutatakse järgmisi sümboleid (tähti) ehk numbreid:

0, 1, 2, 3, 4, 5, 6, 7, 8 ja 9,

Arvutites kasutatakse aga kümnendsüsteemi asemel enamasti kahendsüsteemi. Selle põhjus on lihtne: nimelt on mingi p-ndsüsteemi kasutamisel arvutis vaja seadmeid või süsteeme, millel on p hästi eristatavat seisundit. Vastavad seadmed ja süsteemid on aga kõige lihtsamad ja töökindlamad minimaalse p väärtuse (s. t. $p = 2$) korral. Peale nende süsteemide kasutatakse arvutitega seoses veel teisigi positsioonilisi arvusüsteeme, millest vaatleme neljand, kaheksand- ja kuueteistkümnendsüsteemi.

Et kümnendsüsteemi numbritega oleme kõige rohkem harjunud, siis on otstarbekohane kasutada samu sümboleid ka kahend-, neljand- ja kaheksandsüsteemi puhul. Seega kasutame kahendsüsteemis numbreid

0 ja 1,

neljandsüsteemis numbreid

0, 1, 2 ja 3

ning kaheksandsüsteemis numbreid

0, 1, 2, 3, 4, 5, 6 ja 7.

Märgime muuhulgas, et naturaalarvude tähistamise süsteem tähestikuga A_1 ja tähistamise aluseks oleva valemiga (4) kujutabki endast neljandsüsteemi.

Kuueteistkümnendsüsteemis kümnendsüsteemi numbrimärkidest ei piisa. Selles süsteemis on tarvis veel erisümboleid (numbreid) kümnendsüsteemi tähiste

10, 11, 12, 13, 14 ja 15 (6)

jaoks. Põhimõtteliselt on vastavate erisümbolite valik suvaline. Nad peavad ainult erinema juba olemasolevatest kümnendsüsteemi numbritest ja loomulikult ka üksteisest. Võtame kümnendsüsteemi tähiste (6) jaoks kasutusele vastavalt järgmised sümbolid (numbrid):

$\overline{0}$, $\overline{1}$, $\overline{2}$, $\overline{3}$, $\overline{4}$ ja $\overline{5}$.

Tähendame, et mistahes p-ndsüsteemi alus p avaldub sellessamas arvusüsteemis alati liittähisena 10.

Alljärgnevas tabelis 1 esitame esimese kahekümne viie naturaalarvu tähised kahend-, neljand-, kaheksand-, kümnend- ja kuueteistkümnendsüsteemis.

Tabel 1

Naturaalarv	Tähis 2-nd-süsteemis	Tähis 4-nd-süsteemis	Tähis 8-nd-süsteemis	Tähis 10-nd-süsteemis	Tähis 16-nd-süsteemis
O	0	0	0	0	0
OI	1	1	1	1	1
OII	10	2	2	2	2
OIII	11	3	3	3	3
OIIII	100	10	4	4	4
OIIIII	101	11	5	5	5
OIIIIII	110	12	6	6	6
OIIIIIII	111	13	7	7	7
OIIIIIIII	1000	20	10	8	8
OIIIIIIIII	1001	21	11	9	9
OIIIIIIIIII	1010	22	12	10	$\overline{0}$
OIIIIIIIIIII	1011	23	13	11	$\overline{1}$
OIIIIIIIIIIII	1100	30	14	12	$\overline{2}$
OIIIIIIIIIIIII	1101	31	15	13	$\overline{3}$
OIIIIIIIIIIIIII	1110	32	16	14	$\overline{4}$
OIIIIIIIIIIIIIII	1111	33	17	15	$\overline{5}$
OIIIIIIIIIIIIIIII	10000	100	20	16	10
OIIIIIIIIIIIIIIIII	10001	101	21	17	11
OIIIIIIIIIIIIIIIIII	10010	102	22	18	12
OIIIIIIIIIIIIIIIIIII	10011	103	23	19	13
OIIIIIIIIIIIIIIIIIIII	10100	110	24	20	14
OIIIIIIIIIIIIIIIIIIIII	10101	111	25	21	15
OIIIIIIIIIIIIIIIIIIIIII	10110	112	26	22	16
OIIIIIIIIIIIIIIIIIIIIIII	10111	113	27	23	17
OIIIIIIIIIIIIIIIIIIIIIIII	11000	120	30	24	18

Tabelist 1 on muide näha, kui palju ülevaatlikumad on naturaalarvude tähised naturaalarvude enestega (s. o. sõnadega tabeli 1 esimeses veerus) võrreldes. Isegi kahendsüsteemi tähised on vastavatest naturaalarvudest tunduvalt lühemad. Suuremate naturaalarvude, samuti arvusüsteemi suuremate aluste puhul on tähiste ja vastavate naturaalarvude pikkuste vahekord veelgi enam tähiste kasuks.

Mitmete arvusüsteemide samaaegsel kasutamisel võib tekkida arusaamatusi. Nimelt ei saa tähise enda järgi alati ütelda, missuguse süsteemi tähisega on meil tegemist. Nii võib näiteks tähis 101 kuuluda ükskõik missugusesse ülaltoodud süsteemidest. Arusaamatuste vältimiseks tuleb seega märkida (kas kontekstis või vahetult arvu tähiste juures, näiteks indeksi kohal) vastava arvusüsteemi alus. Arvusüsteemi alust ei saa muidugi esitada sellessamas süsteemis, sest arvul p on p-ndsüsteemis ikka ühesugune kuju «10». Järelikult tuleb arvusüsteemide alused tähistada ühes

kindlas süsteemis, näiteks kümnendsüsteemis. Kümnendsüsteemi võtame aluseks samuti arvusüsteemide nimetamisel. Muide, seda kokkulepet oleme eespool tegelikult juba ka rakendanud.

5. Teatavasti kasutatakse matemaatikas peale naturaalarvude veel ratsionaalarve, irratsionaalarve, kompleksarve jne. Et konkreetsed (n.-ö. numbrilised) arvutused taanduvad aga ikka opereerimisele ratsionaalarvudega, siis tuleks siinkohal vaadelda veel ainult ratsionaalarve.

Analoogiliselt naturaalarvude definitsiooniga võib ka ratsionaalarvud defineerida induktiivselt kui teatud liiki sõnad tähestikus {O, I, /, —}, kus viimast kaht sümbolit tuleb sisuliselt mõista vastavalt kui murrujoont ja miinusmärki. Üksikasjalikku definitsiooni me siin ei esita, toome ainult mõned näited ratsionaalarvude kohta:

$$\text{OI/OIII} \qquad \left(\text{ratsionaalarv } \frac{1}{3}\right).$$

$$\text{—OIIOI/OII} \qquad \left(\text{ratsionaalarv } -2\frac{1}{2}\right).$$

Teatavate lisatingimuste korral (murru P/Q lugejal ja nimetajal pole ühiseid tegureid peale teguri 1 jne.) võib ratsionaalarvude võrdsust mõista kui nende arvude graafilist võrdsust. Liht- ja segamurdudega opereerimise reegleid arvestades pole raske defineerida ka aritmeetilisi tehteid ratsionaalarvudega. Et ratsionaalarvudega opereerimisel esinevad samasugused raskused, mis ilmnesid naturaalarvude korral, on ka antud juhul otstarbekohane kasutusele võtta ratsionaalarvude tähistamise ökonoomsemad süsteemid. Teisiti öeldes, on otstarbekohane laiendada naturaalarvude tähistamise positsioonilisi süsteeme ka ratsionaalarvude vallale.

Positsiooniline p-ndsüsteem ($p \geqslant 2$) ratsionaalarvude tähistamisel tugineb järgmisel võrdusel:

$$\begin{aligned} &P_m P_{m-1} \ldots P_1 P_0,\ Q_1 Q_2 \ldots Q_{n-1} Q_n = \\ &= P_m \cdot p^m + P_{m-1} \cdot p^{m-1} + \ldots + P_1 \cdot p + P_0 + Q_1 \cdot p^{-1} + \\ &+ Q_2 \cdot p^{-2} + \ldots + Q_{n-1} \cdot p^{-(n-1)} + Q_n \cdot p^{-n}, \end{aligned} \tag{7}$$

kus võrduse (7) vasakul poolel paikneb ratsionaalarvu tähis, paremal poolel aga vastav ratsionaalarv. Tähised P_i ja Q_j tähendavad võrduse (7) vasakul poolel p-ndsüsteemi numbreid, võrduse paremal poolel aga nendele numbritele vastavaid naturaalarve. Arvu p negatiivseid astmeid tuleb mõista tavalises mõttes: $p^{-i} = \frac{1}{p^i}$. On selge, et võrdus (7) on naturaalarvude puhul kehtiva võrduse (4) üldistus.

Et naturaalarvude tähiste leidmist on eespool juba käsitletud, siis võime ratsionaalarvude tähiste leidmisel piirduda ainult lihtmurdudega. Sel juhul on aluseks võrduse (7) erijuht

$$0,\ Q_1Q_2\ldots Q_{n-1}Q_n = Q_1\cdot p^{-1} + Q_2\cdot p^{-2} + \ldots$$
$$\ldots + Q_{n-1}\cdot p^{-(n-1)} + Q_n\cdot p^{-n}. \qquad (8)$$

Vaatleme näiteks ratsionaalarvule OIII/OIIII $\left(\text{s. t. } \frac{3}{4}\right)$ vastava kümnendsüsteemi tähise leidmist. Lihtsuse pärast kasutame ka vastavates arvutustes kümnendsüsteemi tähiseid:

$$\frac{3}{4} = \frac{3\cdot 10}{4\cdot 10} = \frac{28+2}{40} = \frac{7}{10} + \frac{1}{20} =$$
$$= \frac{7}{10} + \frac{10}{20\cdot 10} = \frac{7}{10} + \frac{5}{10^2}.$$

Seega

$$\frac{3}{4} = 0{,}75_{(10)}.$$

Arvutusi võib teha veelgi lihtsamini arvu $\frac{3}{4}$ järjestikusel korrutamisel arvuga 10 ja korrutamisel tekkivate täisosade eraldamise teel (vt. võrduse (8) paremat poolt), mis on samaväärne arvu 3 jagamisega arvuga 4.

Lihtmurdude tähiste leidmisel võivad aga esineda ka juhtumid, kus tähise sümbolite jada osutub lõpmatuks. Nii näiteks saame arvule $\frac{1}{3}$ vastava kümnendsüsteemi tähise leidmisel sümbolite jada

0,3333...,

mis on lõpmatu. Tähise lõplikkuse küsimuses annab ammendava vastuse järgmine teoreem.

Teoreem 1. *Selleks et taandumatu lihtmurru P/Q $(Q \neq 0)$ tähis positsioonilises p-ndsüsteemis oleks lõplik, on tarvilik ja piisav, et nimetaja Q kõik tegurid (tegurite kordsus pole oluline) oleksid ka vastava arvusüsteemi aluse p teguriteks.*

Tõestus. 1) Piisavus. Olgu lihtmurru P/Q nimetaja kõik tegurid arvusüsteemi aluse p teguriteks. Sel juhul leidub tegur r, nii et $Q\cdot r = p^k$ $(k > 0)$. Siis saame

$$\frac{P}{Q} = \frac{P\cdot r}{Q\cdot r} = \frac{P\cdot r}{p^k}. \qquad (9)$$

Et saadud murru lugeja $P \cdot r$ $(0 < P \cdot r < p^k)$ on täisarv, siis on tema p-ndsüsteemi tähiseks lõplik sümbolite jada $Q_1 Q_2 \ldots Q_k$, mis sisuliselt tähendab avaldist

$$Q_1 \cdot p^{k-1} + Q_2 \cdot p^{k-2} + \ldots + Q_k. \qquad (10)$$

Korrutades avaldist (10) arvuga p^{-k} ja jätkates võrdust (9), saamegi

$$\frac{P}{Q} = 0, Q_1 Q_2 \ldots Q_k,$$

millega piisavus on tõestatud.

2) T a r v i l i k k u s. Olgu nimetajal Q tegur r, mis pole arvu p teguriks. Oletame vastuväiteliselt, et murru P/Q tähis p-ndsüsteemis on lõplik, s. t. et

$$\frac{P}{Q} = 0, Q_1 Q_2 \ldots Q_n = \frac{Q_1}{p} + \frac{Q_2}{p^2} \ldots + \frac{Q_n}{p^n}.$$

Viimasest võrdusest saame omakorda

$$P \cdot p^n = Q(Q_1 \cdot p^{n-1} + Q_2 \cdot p^{n-2} + \ldots + Q_n),$$

kus parempoolne avaldis jagub arvuga r. Võrduse vasakul poolel olev avaldis aga arvuga r ei jagu, sest arvuga r ei jagu ei P ega ka p. Seega jõudsime vasturääkivuseni, millest järeldub tingimuse tarvilikkus.

Et arvutites pole võimalik opereerida sümbolite lõpmatute jadadega, tuleb lõpmatu jada korral paratamatult võtta sellest ainult osa, nimelt n sümbolit peale koma, kus n määratakse arvuti ehitusega. Asjaolust, et vaatluse alla võetakse ainult lõplik arv sümboleid, järeldub, et erinevatel ratsionaalarvudel võib olla üks ja sama tähis. Nii on näiteks arvudele $\frac{1}{3}$ ja $\frac{333}{1000}$ kümnendsüsteemi tähis (kolm kohta pärast koma) üks ja sama: 0,333.

Edasi tekib küsimus, kas võib ühel arvul olla mitu erinevat tähist. Naturaalarvude, aga samuti täisarvude puhul pole see võimalik, kui mitte arvestada sümboleid 0 tähise alguses. Pisut erinev on olukord aga lihtmurdude ja mittetäisarvuliste ratsionaalarvude korral, nagu näeme alljärgnevast näitest:

$$\frac{1}{2} = 0{,}49999\ldots,$$

$$\frac{1}{2} = 0{,}50000\ldots,$$

kus arvul $\frac{1}{2}$ on kümnendsüsteemis kaks erinevat tähist. Erinevad tulevad ka selle arvu vastavad n-kohalised tähised. Kui tähiste

leidmisel rakendada aga viimase, *n*-nda koha ümardamist, siis tulevad ühe ja sama arvu *n*-kohalised tähised ikka võrdsed.

Edasi tekib veel probleem, kuidas nimetada arvude tähiseid kümnendsüsteemist erinevates positsioonilistes arvusüsteemides. Nähtavasti on kõige otstarbekohasem nimetada tähiseid numbrimärkide järjestikuse loetlemise teel. Näiteks tähist

$$101,\ 101_{(2)}$$

tuleks seega nimetada kahendsüsteemi tähiseks *üks — null — üks — koma — üks — null — üks*, tähist

$$56{,}2_{(8)}$$

tuleks nimetada kaheksandsüsteemi tähiseks *viis — kuus — koma — kaks*» (mitte aga *viiskümmend kuus koma kaks*), tähist

$$\overline{329}_{(16)}$$

tuleks nimetada 16-süsteemi tähiseks *kolm — kaks kaetud — üheksa*.

Samuti tekib ka küsimus, kas võiks termini *tähis* asemel kasutada terminit arv, näiteks *tähise* $56{,}2_{(8)}$ asemel ütelda *arv* $56{,}2_{(8)}$. Asi seisneb nimelt selles, et need väljendid on mõlemad lubatavad, kuid neid tuleb mõista erinevalt. Tähise $56{,}2_{(8)}$ all tuleb mõista tähist, s. t. neljatähelist sõna *viis — kuus — koma — kaks*, arvu $56{,}2_{(8)}$ all aga tähisele $56{,}2_{(8)}$ vastavat ratsionaalarvu. Seega tuleb vastavaid väljendeid kasutada arvestades vastava lause mõtet.

Lõpuks mõni sõna võrdusmärgi kasutamisest. Võrdusmärki ei kasuta me mitte ainult arvu ja tema tähise vahel, vaid ka erinevate arvusüsteemide tähiste vahel, näiteks

$$1100_{(2)} = 30_{(4)}. \qquad (11)$$

Võrdus (11) tähendab (pedantse täpsusega öeldes), et tähistele $1100_{(2)}$ ja $30_{(4)}$ vastavad arvud on graafiliselt võrdsed sõnad. Edaspidi loobume aga sellisest pedantsest täpsusest ja ütleme lihtsalt, et arvud $1100_{(2)}$ ja $30_{(4)}$ on võrdsed.

§ 4. ARITMEETILISED TEHTED POSITSIOONILISTES ARVUSÜSTEEMIDES

1. Arvude tähiseid on otstarbekohane kasutada mitte ainult arvude tähistamisel, vaid ka arvutamisel. Arvutamisel tuleb lähtuda sellest, et tähis

$$P_m P_{m-1} \ldots P_0, \quad Q_1 Q_2 \ldots Q_n$$

on tegelikult arvu p astmete suhtes alanevas järjekorras järjestatud polünoom

$$P_m \cdot p^m + P_{m-1} \cdot p^{m-1} + \ldots + P_0 + Q_1 \cdot p^{-1} + Q_2 \cdot p^{-2} + \ldots + Q_n \cdot p^{-n},$$

mis on ainult kompaktsemalt üles kirjutatud.

Seda arvestades võime öelda, et tähiste

$$A' = \ldots P'_{i+1} P'_i P'_{i-1} \ldots$$
$$A'' = \ldots P''_{i+1} P''_i P''_{i-1} \ldots$$

liitmine või lahutamine taandub polünoomi

$$\ldots + (P'_{i+1} \pm P''_{i+1}) p^{i+1} + (P'_i \pm P''_i) p^i + (P'_{i-1} \pm P''_{i-1}) p^{i-1} + \ldots$$

kordajate arvutamisele, kusjuures leitavad kordajad peavad rahuldama tingimust $0 \leqslant P_i < p$. Seega tuleb tähiste liitmisel ja lahutamisel liita või vastavalt lahutada samadel kohtadel asuvad numbrid ja teostada vajalikud ülekanded — liitmisel madalamalt kohalt kõrgemale kohale, lahutamisel aga vastupidi (nn. *laenamine*).

Korrutamisel tuleb polünoomid (ehk teisiti öeldes — tähised) korrutada liikmeti. Praktiliselt sooritatakse korrutamine nii, et ühe polünoomi, näiteks teise, iga liikmega $P''_j \cdot p^j$ korrutatakse läbi kogu esimene polünoom, millega saame aritmeetikas hästi tuntud osakorrutised

$$\ldots + (P'_{i+1} \cdot P''_j) p^{i+1+j} + (P'_i \cdot P''_j) p^{i+j} + (P'_{i-1} \cdot P''_j) p^{i-1+j} + \ldots$$

Osakorrutiste arvutamine taandub põhiliselt korrutiste $P'_i \cdot P''_j$ leidmisele, s. o. vastava p-ndsüsteemi ükskordühe kasutamisele. Korrutise lõplikul leidmisel tuleb liita kõik osakorrutised.

Muuseas on kerge näha, et p-ndsüsteemi mistahes arvu korrutamine astmega p^k, kus k on positiivne täisarv, on samaväärne koma nihutamisega selles arvus k koha võrra paremale, negatiivse k korral aga koma nihutamisega k koha võrra vasakule. On kerge veenduda, et kümnendsüsteemis tuntud reegel koma asukoha määramiseks korrutises kehtib mistahes p-ndsüsteemis.

Jagamisel võib enne jagamist muuta jagaja täisarvuks, korrutades jagatavat ja jagajat arvuga p^k (mis tähendab koma nihutamist k koha võrra paremale).

2. Nagu näeme, toimub aritmeetiliste tehete sooritamine kõigis positsioonilistes arvusüsteemides meile juba kümnendsüsteemist tuntud põhimõtete alusel. Erinevad on aga igas p-ndsüsteemis ühekohaliste arvude liitmis- ja korrutamistabelid.

Alljärgnevates tabelites on tabeli äärtele esitatud liidetavad või tegurid, vastava rea ja veeru ristumiskohal aga tehte tulemus (summa või vastavalt korrutis). Kahekohalise tulemuse esimene koht kujutab endast ülekannet järgmisele kõrgemale kohale. Lahutamisel tuleb kasutada liitmistabelit, kusjuures vähendatav tuleb leida tabeli sisemisest osast, lahutatav ühelt ja vahe teiselt äärelt. Vaatlemegi nüüd neid tabeleid erinevates arvusüsteemides (2-, 8- ja 16-ndsüsteemis).

Kõige lihtsamad on vastavad tabelid ja seega ka aritmeetiliste tehete sooritamine kahendsüsteemis:

+	0	1
0	0	1
1	1	10

×	0	1
0	0	0
1	0	1

Toome mõned näited tehete kohta kahendsüsteemis:

$$\begin{array}{r} 1011101{,}101 \\ +\ \ 111011{,}111 \\ \hline 10011001{,}100 \end{array} \qquad \begin{array}{r} 1011101{,}101 \\ -\ \ 111011{,}111 \\ \hline 100001{,}110 \end{array}$$

$$\begin{array}{r} 110{,}101 \times 11{,}01 \\ \hline 110101 \\ 110101 \\ \hline 10101{,}10001 \end{array}$$

$$\begin{array}{r|l} 110111 & 101 \\ \hline 101 & 1011 \\ \hline 111 & \\ 101 & \\ \hline 101 & \\ 101 & \\ \hline \end{array}$$

Kaheksandsüsteemi liitmis- ja korrutamistabel tulevad järgmised:

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11
3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

×	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	10	12	14	16
3	0	3	6	11	14	17	22	25
4	0	4	10	14	20	24	30	34
5	0	5	12	17	24	31	36	43
6	0	6	14	22	30	36	44	52
7	0	7	16	25	34	43	52	61

Esitame mõned näited tehete kohta 8-ndsüsteemis:

$$\begin{array}{r} 143{,}271 \\ +\ 26{,}544 \\ \hline 172{,}035 \end{array} \qquad \begin{array}{r} 143{,}271 \\ -\ 26{,}544 \\ \hline 114{,}525 \end{array}$$

$$\begin{array}{r} 21{,}74 \times 15{,}3 \\ \hline 6564 \\ 13154 \\ 2174 \\ \hline 357{,}724 \end{array} \qquad \begin{array}{r} 1450{,}22 : 43 = 27{,}06 \\ \hline 106 \\ 370 \\ 365 \\ \hline 322 \\ 322 \\ \hline \end{array}$$

Kuueteistkümnendsüsteemi liitmis- ja korrutamistabel on juba ulatuslikumad (vt. tabelid 2. ja 3). Toomegi lõpuks mõned näited tehete kohta 16-ndsüsteemis:

$$\begin{array}{r} \bar{2}74{,}5\bar{0} \\ +\ 9\bar{6}3{,}1\bar{8} \\ \hline 15\bar{4}2{,}12 \end{array} \qquad \begin{array}{r} \bar{2}74{,}5\bar{0} \\ -\ 9\bar{6}3{,}1\bar{8} \\ \hline 306{,}0\bar{2} \end{array}$$

$$\begin{array}{r} 45{,}3\bar{2} \times 7{,}\bar{1} \\ \hline 36794 \\ 22\bar{0}0\bar{4} \\ \hline 261{,}1\bar{3}4 \end{array} \qquad \begin{array}{r} 9\bar{8}3\bar{0}{,}98 : \bar{3}8 = \bar{1}5{,}29 \\ \hline 948 \\ 450 \\ 438 \\ \hline 229 \\ 110 \\ \hline 798 \\ 798 \\ \hline \end{array}$$

Tabel 2

+	0	1	2	3	4	5	6	7	8	9	$\bar{0}$	$\bar{1}$	$\bar{2}$	$\bar{3}$	$\bar{4}$	$\bar{5}$
0	0	1	2	3	4	5	6	7	8	9	$\bar{0}$	$\bar{1}$	$\bar{2}$	$\bar{3}$	$\bar{4}$	$\bar{5}$
1	1	2	3	4	5	6	7	8	9	$\bar{0}$	1	$\bar{2}$	$\bar{3}$	$\bar{4}$	$\bar{5}$	10
2	2	3	4	5	6	7	8	9	$\bar{0}$	$\bar{1}$	2	$\bar{3}$	$\bar{4}$	$\bar{5}$	10	11
3	3	4	5	6	7	8	9	0	$\bar{1}$	$\bar{2}$	3	$\bar{4}$	$\bar{5}$	10	11	12
4	4	5	6	7	8	9	$\bar{0}$	$\bar{1}$	$\bar{2}$	$\bar{3}$	$\bar{4}$	$\bar{5}$	10	11	12	13
5	5	6	7	8	9	$\bar{0}$	$\bar{1}$	$\bar{2}$	$\bar{3}$	$\bar{4}$	$\bar{5}$	10	11	12	13	14
6	6	7	8	9	$\bar{0}$	$\bar{1}$	$\bar{2}$	$\bar{3}$	$\bar{4}$	$\bar{5}$	10	11	12	13	14	15
7	7	8	9	$\bar{0}$	$\bar{1}$	$\bar{2}$	$\bar{3}$	$\bar{4}$	$\bar{5}$	10	11	12	13	14	15	16
8	8	9	$\bar{0}$	$\bar{1}$	$\bar{2}$	$\bar{3}$	$\bar{4}$	$\bar{5}$	10	11	12	13	14	15	16	17
9	9	$\bar{0}$	$\bar{1}$	$\bar{2}$	$\bar{3}$	$\bar{4}$	$\bar{5}$	10	11	12	13	14	15	16	17	18
$\bar{0}$	$\bar{0}$	$\bar{1}$	$\bar{2}$	$\bar{3}$	$\bar{4}$	$\bar{5}$	10	11	12	13	14	15	16	17	18	19
$\bar{1}$	$\bar{1}$	$\bar{2}$	$\bar{3}$	$\bar{4}$	$\bar{5}$	10	11	12	13	14	15	16	17	18	19	$1\bar{0}$
$\bar{2}$	2	$\bar{3}$	$\bar{4}$	$\bar{5}$	10	11	12	13	14	15	16	17	18	19	$1\bar{0}$	$1\bar{1}$
$\bar{3}$	$\bar{3}$	$\bar{4}$	$\bar{5}$	10	11	12	13	14	15	16	17	18	19	$1\bar{0}$	$1\bar{1}$	$1\bar{2}$
$\bar{4}$	$\bar{4}$	$\bar{5}$	10	11	12	13	14	15	16	17	18	19	$1\bar{0}$	$1\bar{1}$	$1\bar{2}$	$1\bar{3}$
$\bar{5}$	$\bar{5}$	10	11	12	13	14	15	16	17	18	19	$1\bar{0}$	$1\bar{1}$	$1\bar{2}$	$1\bar{3}$	$1\bar{4}$

Tabel 3

×	0	1	2	3	4	5	6	7	8	9	$\bar{0}$	$\bar{1}$	$\bar{2}$	$\bar{3}$	$\bar{4}$	$\bar{5}$
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	$\bar{0}$	$\bar{1}$	$\bar{2}$	$\bar{3}$	$\bar{4}$	$\bar{5}$
2	0	2	4	6	8	$\bar{0}$	$\bar{2}$	$\bar{4}$	10	12	14	16	18	$1\bar{0}$	$1\bar{2}$	$1\bar{4}$
3	0	3	6	9	$\bar{2}$	$\bar{5}$	12	15	18	$1\bar{1}$	14	21	24	27	$2\bar{0}$	$2\bar{3}$
4	0	4	8	$\bar{2}$	10	14	18	$1\bar{2}$	20	24	28	$2\bar{2}$	30	34	38	$3\bar{2}$
5	0	5	$\bar{0}$	$\bar{5}$	14	19	$1\bar{4}$	23	28	$2\bar{3}$	32	37	$3\bar{2}$	41	46	$4\bar{1}$
6	0	6	$\bar{2}$	12	18	$1\bar{4}$	24	$2\bar{0}$	30	36	$3\bar{2}$	42	48	$4\bar{4}$	54	$5\bar{0}$
7	0	7	$\bar{4}$	15	$1\bar{2}$	23	$2\bar{0}$	31	38	$3\bar{5}$	46	$4\bar{3}$	54	$5\bar{1}$	62	69
8	0	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	0	9	12	$1\bar{1}$	24	$2\bar{3}$	36	$3\bar{5}$	48	51	$5\bar{0}$	63	$6\bar{2}$	75	$7\bar{4}$	87
$\bar{0}$	0	$\bar{0}$	14	$1\bar{4}$	28	32	$3\bar{2}$	46	50	$5\bar{0}$	64	$6\bar{4}$	78	82	$8\bar{2}$	96
$\bar{1}$	0	$\bar{1}$	16	21	$2\bar{2}$	37	42	$4\bar{3}$	58	63	$6\bar{4}$	79	84	$8\bar{5}$	$9\bar{0}$	$0\bar{5}$
$\bar{2}$	0	$\bar{2}$	18	24	30	$3\bar{2}$	48	54	60	$6\bar{2}$	78	84	90	$9\bar{2}$	$0\bar{8}$	$1\bar{4}$
$\bar{3}$	0	$\bar{3}$	$1\bar{0}$	27	34	41	$4\bar{4}$	51	68	75	82	$8\bar{5}$	$9\bar{2}$	$0\bar{9}$	$1\bar{6}$	$2\bar{3}$
$\bar{4}$	0	$\bar{4}$	$1\bar{2}$	$2\bar{0}$	38	46	54	62	70	74	82	$9\bar{0}$	$0\bar{8}$	$1\bar{6}$	$2\bar{4}$	$3\bar{2}$
$\bar{5}$	0	$\bar{5}$	$1\bar{4}$	$2\bar{3}$	$3\bar{2}$	$4\bar{1}$	$5\bar{0}$	69	78	87	96	$0\bar{5}$	$1\bar{4}$	$2\bar{3}$	$3\bar{2}$	$4\bar{1}$

§ 5. ARVUDE TÄHISTE LEIDMINE ERINEVATES ARVUSÜSTEEMIDES

Paragrahvis 3 käsitlesime küsimust, kuidas leida arvude, s. t. tähestiku {O, I, /, —} spetsiaalsete sõnade tähiseid mistahes positsioonilises p-ndsüsteemis ($p \geqslant 2$). Käesolevas paragrahvis vaatleme, kuidas arvu tähise järgi ühes positsioonilises arvusüsteemis leida tema tähis teises arvusüsteemis arvu ennast kasutamata. Tähiste leidmise põhimõtted on muidugi samad mis paragrahvis 3. Otstarbekohane on samuti eraldi vaadelda täisarvude ja lihtmurdude tähiste leidmist, kusjuures võib ilmselt piirduda ainult mittenegatiivsete arvude käsitlemisega.

1. Täisarvude tähiste leidmine erinevates arvusüsteemides

Olgu antud täisarvu A tähis p-ndsüsteemis

$$a_m a_{m-1} \ldots a_1 a_0, \tag{1}$$

kusjuures ülesandeks on leida selle tähis q-ndsüsteemis (tähendame, et arv q on esitatud samuti p-ndsüsteemis).

Arvu A q-ndsüsteemi tähise leidmine tähendab kordajate b_i leidmist, mis rahuldavad tingimusi $0 \leqslant b_i < q$ $(i = 0, 1, \ldots, k)$ ja võrdust

$$a_m a_{m-1} \ldots a_1 a_{0(p)} = a_m \cdot p^m + a_{m-1} \cdot p^{m-1} + \ldots + a_1 \cdot p + a_0 = \\ = b_k \cdot q^k + b_{k-1} \cdot q^{k-1} + \ldots + b_1 \cdot q + b_0. \qquad (2)$$

Võrdusest (2) nähtub, et kordajaid b_i on võimalik leida arvu A (täpsemini, arvu A tähise (1)) järk-järgulise jagamise teel arvuga q. Et tähis (1) ja arv q on esitatud p-ndsüsteemis, toimub ka jagamine p-ndsüsteemis.[6]

Esimesel jagamisel tekkiv jääk b_0 annab arvu A q-ndsüsteemi esituse viimase koha (numbri). Jagamisel saadav täisosa

$$b_k \cdot q^{k-1} + b_{k-1} \cdot q^{k-2} + \ldots + b_2 \cdot q + b_1$$

tuleb aga jälle jagada arvuga q jne. Nii leiame järk-järgult arvu A q-ndsüsteemi tähise kõik numbrikohad b_i vastupidises järjekorras. Protsessi lõpu määrab viimane täisosa b_k, mis on väiksem arvust q. Et jagamine toimub p-ndsüsteemis, saame ka jäägid b_i esialgu p-ndsüsteemis. Arvu A q-ndsüsteemi tähise lõppkuju saamiseks tuleb jäägid b_i kodeerida (tähistada) q-ndsüsteemi numbrite abil.

Toome nüüd mõningaid näiteid täisarvude tähiste leidmise kohta 2-nd-, 8-nd-, 10-nd- ja 16-ndsüsteemis. Näidete põhjal selgub ka, kuidas vastavaid arvutusi otstarbekohaselt teha ja paigutada.

Näide 1. Leiame arvu $378_{(10)}$ tähise 2-ndsüsteemis.

378	|2							
2	189	|2						
17	18	94	|2					
16	9	8	47	|2				
18	8	14	4	23	|2			
18	1	14	7	2	11	|2		
0		0	6	3	10	5	|2	
			1	2	1	4	2	|2
				1		1	2	1
							0	

Seega $378_{(10)} = 101\,111\,010_{(2)}$.

[6] Seoses sellega on arusaadav, et aritmeetiliste tehete sooritamist erinevates positsioonilistes arvusüsteemides tuleb vaadelda enne käesoleva paragrahvi käsitlemist.

Näide 2. Leiame arvu $378_{(10)}$ tähise 8-ndsüsteemis.

```
378 | 8
32  | 47 | 8
---   40   5
 58   ---
 56    7
 --
  2
```

Seega $378_{(10)} = 572_{(8)}$.

Näide 3. Leiame arvu $378_{(10)}$ tähise 16-ndsüsteemis.

```
378 | 16
32  | 23 | 16
---   16   1
 58   --
 48    7
 --
 10
```

Seega $378_{(10)} = 17\bar{0}_{(16)}$ (jäägile 10 vastab 16-ndsüsteemis number $\bar{0}$).

Näide 4. Leiame arvu $1361_{(8)}$ tähise 10-ndsüsteemis. Jagamine toimub siin 8-ndsüsteemis, jagada tuleb arvuga $12_{(8)} = 10_{(10)}$.

```
1361 | 12
12   | 113 | 12
---    106   7
 16    ---
 12      5
 --
 41
 36
 --
  3
```

Seega $1361_{(8)} = 753_{(10)}$.

Näide 5. Leiame arvu $\bar{2}39_{(16)}$ tähise 8-ndsüsteemis. Jagamine toimub siin 16-ndsüsteemis, jagada tuleb arvuga 8.

```
239 | 8
8   | 187 | 8
---   18    30  | 8
 43   --    30  | --
 40    7    --    6
 --         0
 39
 38
 --
  1
```

Seega $\bar{2}39_{(16)} = 6071_{(8)}$.

2. Lihtmurdude tähiste leidmine erinevates arvusüsteemides

Olgu lihtmurd A ($|A| < 1$) esitatud p-ndsüsteemis tähisega

$$A_{(p)} = 0, a_1 a_2 \ldots a_m. \tag{3}$$

Arvu A tähise leidmine q-ndsüsteemis tähendab niisuguste kordajate b_i leidmist, mis rahuldavad võrdust

$$A_{(p)} = b_1 \cdot q^{-1} + b_2 \cdot q^{-2} + \ldots + b_n \cdot q^{-n} + \ldots \tag{4}$$

ning võrratusi $0 \leqslant b_i < q$ ($i = 1, 2, \ldots, n, \ldots$). Kordajaid b_i on võimalik leida arvu A (täpsemini, arvu A tähise (3)) korrutamise teel arvuga q, kusjuures korrutamine toimub p-ndsüsteemis. Esimesel korrutamisel saadav täisosa b_1 annab arvu A q-ndsüsteemi esituse esimese koha (numbri). Korrutamisel saadav murdosa tuleb aga jälle korrutada arvuga q jne. Nii leiame järk-järgult arvu A q-ndsüsteemi tähise kõik numbrikohad b_i ($i = 1, 2, \ldots, n$) tavalises järjekorras. Numbrikohad b_i saame p-ndsüsteemis ja tähise lõppkuju saamiseks tuleb need jälle kodeerida q-ndsüsteemi numbrite abil.

Nagu juba teame paragrahvist 3, võib siin tähise leidmise protsess osutuda lõpmatuks. Järelikult on meil sel juhul vaja rakendada mingisuguseid lisakriteeriume, et vastavat teisendusprotsessi sobival (või õigel) kohal lõpetada. Esitame siinkohal kaks võimalikku kriteeriumi.

1) Esimese kriteeriumi puhul lähtume sellest, et arvuti opereerib ikka kindla pikkusega arvudega (resp. tähistega). Seega võib q-ndsüsteemi tähise kohtade arvutamisel leida sama palju kohti kui mitme kohaliste arvudega antud arvuti opereerib.

2) Teise kriteeriumi puhul lähtume eeldusest, et vaadeldavad arvud on ligikaudsed ja antud standardses kirjutusviisis, kus viga ei ületa poolt viimase koha ühikut. Ligikaudsete arvude tähiste leidmisel ei tohi arvu tähise täpsust kuidagi vähendada, kuid seda pole võimalik ka suurendada. Seega, kui arv A on antud p-ndsüsteemis m kohaga, siis on q-ndsüsteemis otstarbekohane esitada see n kohaga, kus n on võrratust

$$\frac{1}{2} \cdot p^{-m} \geqslant \frac{1}{2} \cdot q^{-n} \tag{5}$$

rahuldav vähim täisarv. Võrratusest (5) saame

$$n \geqslant \frac{m \log p}{\log q}. \tag{6}$$

Seega, leitav kohtade arv n on võrratust (6) rahuldav vähim täisarv. Praktiliselt arvutame mõlema kriteeriumi korral ühe koha rohkem ja siis ümardame.

Toome nüüd näiteid lihtmurdude tähiste leidmise kohta erinevates arvusüsteemides. Näidetest ilmneb ka, kuidas vastavaid arvutusi on otstarbekohane teha.

Näide 6. Leiame arvu $0{,}7250_{(10)}$ tähised 2-nd-, 8-nd- ja 16-ndsüsteemis. Kasutades teist kriteeriumit, leiame kõigepealt numbrikohtade arvu. Valemist (6) saame

2-ndsüsteemi tähise leidmisel:

$$n \geqslant \frac{4}{\log 2} \approx \frac{4}{0{,}301} \approx 13{,}3, \quad \text{seega} \quad n = 14;$$

8-ndsüsteemi tähise leidmisel:

$$n \geqslant \frac{4}{\log 8} \approx \frac{4}{0{,}903} \approx 4{,}4, \quad \text{seega} \quad n = 5;$$

16-ndsüsteemi tähise leidmisel:

$$n \geqslant \frac{4}{\log 16} \approx \frac{4}{1{,}2} \approx 3{,}3, \quad \text{seega} \quad n = 4.$$

Arvutused teeme 10-ndsüsteemis, kusjuures 2-, 8- ja 16-ga korrutatakse teatavasti ainult murdosa. Korrutamisel saadavad täisosad kirjutatakse vertikaaljoonest vasakule.

0	7 2 5 0 × 2
1	4 5 0 0
0	9 0
1	8
1	6
1	2
0	4
0	8
1	6
1	2
0	4
0	8
1	6
1	2
0	4
0	8

0	7 2 5 0 × 8
5	8 0 0 0
6	4
3	2
1	6
4	8
6	4

0	7 2 5 0 × 16
11	6
9	6
9	6
9	6
9	6

Seega saame, et

$0{,}7250_{(10)} = 0{,}101111001100110_{(2)}$,

$0{,}7250_{(10)} = 0{,}56315_{(8)}$,

$0{,}7250_{(10)} = 0{,}\overline{1990}_{(16)}$.

Näide 7. Leiame arvu $0{,}51403_{(8)}$ tähise 10-ndsüsteemis 6 kohaga peale koma. Arvutused tuleb teha 8-ndsüsteemis, kusjuures murdosi tuleb korrutada arvuga $12_{(8)} = 10_{(10)}$.

0	5	1	4	0	3 × 12
6	3	7	0	3	6
4	6	6	4	5	4
10	4	1	6	7	0
5	2	2	4	6	0
2	7	1	7	4	0
11	0	3	3	0	0
0	4	1	6	0	0

Kirjutades saadud täisosad 10-ndsüsteemi numbrite abil, saame

$$0{,}51403_{(8)} = 0{,}648529_{(10)}.$$

Esitatud võtet arvu tähise leidmiseks on võimalik kasutada ka harilike murdude korral. Toome näite ka selle kohta.

Näide 8. Leiame arvu $\frac{1}{3}_{(10)}$ tähise 8-ndsüsteemis 3 kohaga peale koma. Arvutused tulevad analoogilised eelmistega

	$\frac{1}{3} \times 8$
2	$\frac{2}{3}$ $\left(\frac{8}{3} = 2\frac{2}{3}\right)$
5	$\frac{1}{3}$ $\left(\frac{16}{3} = 5\frac{1}{3}\right)$
2	$\frac{2}{3}$
5	$\frac{1}{3}$

Seega saame, et

$$\frac{1}{3}_{(10)} = 0{,}253_{(8)}.$$

3. Kui arvu A tähis koosneb täis- ja murdosast, siis selle tähise teisendamisel mingisse teise arvusüsteemi võib teisendada eraldi täis- ja murdosa ning seejärel need teisendatud osad liita. Kuid täis- ja murdosa võib teisendada ka koos.

Näide 9. Leiame arvu $417{,}59_{(10)}$ tähise 8-ndsüsteemis, kusjuures murdosa arvutame kolme kohaga.

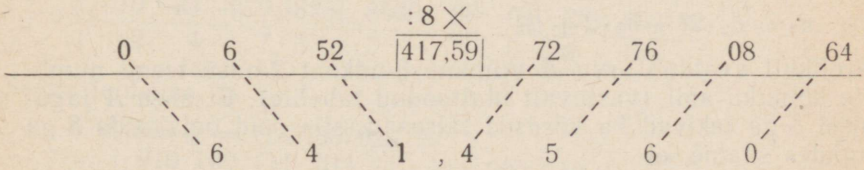
Leides eraldi täisosa, saame $417_{(10)} = 641_{(8)}$:

417	|8	
40	52	|8
17	48	6
16	4	
1		

Murdosa teisendamisel saame $0{,}59_{(10)} = 0{,}456_{(8)}$:

0	59×8
4	72
5	76
6	08
0	64

Kõiki neid arvutusi saab teha aga korraga järgmise skeemi kohaselt:



Komast vasakule kirjutame 8-ga jagamisel saadavad tulemused (joone peale täisosad, poolpõiki joone alla vastavad jäägid), komast paremale 8-ga korrutamisel saadavad tulemused (joone peale murdosad, poolpõiki joone alla vastavad täisosad). Otsitava tähise saame joone all tavalisel kujul.

Negatiivsete arvude $-C$ $(C > 0)$ tähiste teisendamisel teisendame soovitud arvusüsteemi arvu C tähise ja asetame tulemuse ette miinusmärgi «—».

4. Lõpuks vaatleme veel üht erijuhtu arvude tähiste teisendamisel, mil arvusüsteemide aluste vahel kehtib seos

$$p = q^k,$$

kus k on naturaalarv.[7]

Sel juhul on üleminek p-ndsüsteemist q-ndsüsteemi (ja samuti vastupidi) eriti lihtne. Nimelt vastab antud juhul igale p-ndsüsteemi numbrile kindel k numbrist koosnev q-ndsüsteemi numbrite rühm.

[7] Tähendame, et lähtudes seosest $p^k = q^l$ saaks käsitlust mõnevõrra üldistada.

Näitame seda juhul, kui $p = 8$ ja $q = 2$. Kõigepealt märgime, et üksikutele 8-ndsüsteemi numbritele vastavad järgmised 2-ndsüsteemi numbrite kolmikud:

$0 = 000$, $4 = 100$,
$1 = 001$, $5 = 101$,
$2 = 010$, $6 = 110$,
$3 = 011$, $7 = 111$,

kus võrduste paremal pool seisvad kolmikud $a_2a_1a_0$ tähistavad sisuliselt polünoome

$$a_2 \cdot 2^2 + a_1 \cdot 2 + a_0.$$

Olgu nüüd mingi täisarvu A tähis 8-ndsüsteemis kujul

$$A = a_m a_{m-1} \ldots a_1 a_0 \tag{8}$$

ja 2-ndsüsteemis kujul

$$A = b_n b_{n-1} \ldots b_2 b_1 b_0. \tag{9}$$

Jagades arvu A arvuga 8 ja arvestades, et $8 = 2^3$, saame esituse (8) puhul jäägi a_0, esituse (9) puhul aga jäägi $b_2b_1b_0 = b_2 \cdot 2^2 + b_1 \cdot 2 + b_0$, mis peavad olema võrdsed. Seega

$$a_0 = b_2 \cdot 2^2 + b_1 \cdot 2 + b_0.$$

Järelikult avaldub arvu A viimane 8-ndkoht 2-ndsüsteemi numbrite kolmiku abil (vastavalt ülaltoodud tabelile). Et arvu A jagamisel 8-ga tekivad ka võrdsed täisosad, siis neid omakorda 8-ga jagades saame, et

$$a_1 = b_5 \cdot 2^2 + b_4 \cdot 2 + b_3,$$

mis ütleb, et arvu A eelviimane 8-ndkoht avaldub 2-ndsüsteemi numbrite vastava kolmiku kaudu jne.

Lihtmurdude puhul saame analoogilise tulemuse, korrutades vastava murru arvuga 8.

Sama mõttekäigu alusel saame anda reegli ka arvu tähise teisendamiseks 2-ndsüsteemist 8-ndsüsteemi. Nimelt tuleb 2-ndsüsteemi tähise numbrid komast alates jaotada 3-numbrilisteks rühmadeks. Seejärel tuleb iga numbrite rühm asendada vastava 8-ndsüsteemi numbriga. Toome mõned näited arvu tähiste teisendamise kohta 8-ndsüsteemist 2-ndsüsteemi ja vastupidi.

N ä i d e 10. Leiame arvu $357{,}641_{(8)}$ tähise 2-ndsüsteemis. Asendades iga 8-ndsüsteemi numbri kolme 2-ndsüsteemi numbriga, saame

3 5 7, 6 4 1
011 101 111, 110 100 001

ehk (tarbetut nulli eest ära jättes)

$$357{,}641_{(8)} = 11\ 101\ 111,\ 110\ 100\ 001_{(2)}.$$

Näide 11. Leiame arvu 1 111 010 110, 011 100 $11_{(2)}$ tähise 8-ndsüsteemis. Rühmitades numbreid komast alates 3-kaupa, saame (fiktiivseid nulle algusesse ja lõppu juurde lisades)

(00)1′	111′	010′	110′,	011′	100′	11(0)
1	7	2	6	3	4	6

Seega:

$$1\ 111\ 010\ 110,\ 011\ 100\ 11_{(2)} = 1726{,}346_{(8)}.$$

Lõpuks teeme veel ühe praktilise iseloomuga märkuse. Nagu eespool nägime, on arvude 2-ndsüsteemi tähiste leidmine küllaltki aeganõudev protsess. Sellepärast on sageli kasulikum leida arvu tähis esialgu 8-ndsüsteemis, arvestades nimelt seda, et tähise teisendamine 8-ndsüsteemist 2-ndsüsteemi toimub täiesti triviaalselt.

Toome ka selle kohta vastava näite.

Näide 12. Leiame arvu $679{,}4825_{(10)}$ tähise 2-ndsüsteemis. Selleks leiame kõigepealt arvu $679{,}4825_{(10)}$ tähise 8-ndsüsteemis (murdosas arvutame näiteks 5 kohta). Saame

			:8×						
1	10	84	|679,4825|	8600	88	04	32	56	48
1	2	4	7, 3	6	7	0	2	(4)	

Seega leidsime, et

$$679{,}4825_{(10)} = 1247{,}36702_{(8)} =$$
$$= 1\ 010\ 100\ 111,\ 011\ 110\ 111\ 000\ 010_{(2)}.$$

§ 6. ARVUDE KUJUTAMINE ARVUTITES

1. Arvude kujutamisel (salvestamisel) arvutite seadmetes kasutatakse enamikus 2-ndsüsteemi. Ühe arvu kujutamiseks on arvutites reserveeritud kindel kahendkohtade hulk, mida nimetatakse pesaks. Nii on näiteks arvuti «Strela» pesas 43 kahendkohta, arvuti «Ural-1» pesas 36, «Ural-2» ja «Ural-4» pesas 40, «Minsk-2» ja «Minsk-22» pesas aga 37 kahendkohta.

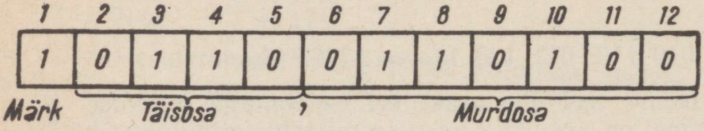
Vastavalt arvuti aritmeetilise seadme ehitusele jaotatakse arvutid kolme liiki: fikseeritud komaga, liikuva komaga ja nn. segakomaga arvutid, kus viimased võivad töötada nii fikseeritud kui ka liikuva koma režiimis.

Fikseeritud komaga arvuti pesas on arvu täis- ja murdosa jaoks reserveeritud kindlad kahendkohad. Nii näiteks võivad mingi arvuti 12-kahendkohalises pesas kahendkohad

---

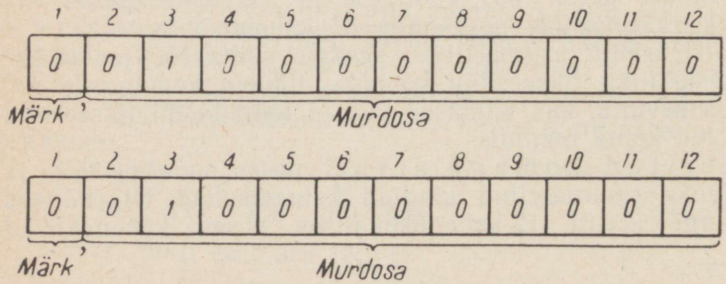
[8] Tähendame, et pesa kahendkohtade numeratsiooni alustatakse sageli ka järjekorranumbriga 0.

nr. 2—5 olla reserveeritud täisosa jaoks, kahendkohad nr. 6—12 murdosa jaoks, esimene kahendkoht aga märgi jaoks.[8] Plussmärk «+» kodeeritakse seejuures sümboliga 0, miinusmärk «—» aga sümboliga 1. Näiteks kujutatakse arv — 110,01101 sellises pesas joonisel 4 näidatud viisil.



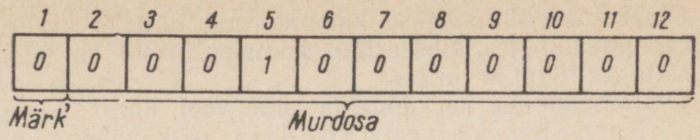
JOON. 4

Koma arvuti pesas eraldi kujutada pole vaja: aritmeetiline seade on juba selliselt konstrueeritud, et tehted pesades olevate arvudega tehakse nii, nagu asuks koma pärast viiendat kahendkohta. Põhimõtteliselt võib koma fikseerida ükskõik missuguse kahendkoha järel. Koma fikseerimine mõjustab õieti ainult korrutamist ja jagamist, kuna liitmine ja lahutamine toimuvad koma asukohast sõltumatult. Nii näiteks, kui koma on fikseeritud vahetult pärast märgikohta, annab joonisel 5 kujutatud pesades (1) ja (2) olevate arvude korrutamine joonisel 6 näidatud tulemuse. Kui koma oleks olnud fikseeritud pärast kolmandat kahendkohta, siis annaks pesades (1) ja (2) olevate arvude korrutamine joonisel 7 esitatud tulemuse. Kui aga koma oleks olnud fikseeritud pärast viiendat kahendkohta, siis annaks pesades (1) ja (2) olevate arvude korrutamine tulemuse 10 000 ($100 \times 100 = 10\,000$). Seda arvu ei saa aga sellist tüüpi pesas enam kujutada, sest seal on täisosa jaoks reserveeritud ainult neli kohta. Kui seda arvu kujutada pesas joonisel 8 näidatud viisil, siis tähendaks see hoopis arvu —0. Niisugust olukorda, kus arvutuste tulemusena saadakse arv, mis absoluutväärtuse poolest ületab maksimaalse pesas kujutatava arvu, nimetatakse *ületäitumiseks*. Et ületäitumise korral saadakse aritmeetilise tehte tulemus moonutatud kujul, tuleb seda olukorda reeglina vältida. Põhimõtteliselt võib ületäitumine tekkida kõigi nelja aritmeetilise tehte puhul.

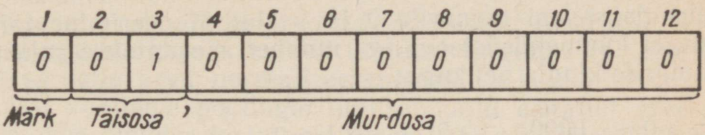


JOON. 5

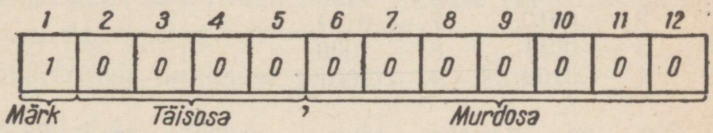
JOON. 6



JOON. 7



JOON. 8



Ületäitumise võimaluste arvestamise kergendamiseks fikseeritakse koma tavaliselt kohe pärast märgikohta. Seega rahuldavad pesas kujutatavad arvud A tingimust $|A| < 1$, millega väldi-takse ületäitumine korrutamise puhul.

Tähendame, et kahendarvude A ($|A| < 1$) kodeerimise süs-teemi, kus plussmärk kodeeritakse sümboliga 0, miinusmärk aga sümboliga 1, nimetatakse o t s e k o o d i d e süsteemiks. Arvule A vastavat otsekoodi tähistatakse sümboliga A_{ok}. Nii saame näiteks

$$[+0{,}1001011]_{ok} = 0{,}1001011,$$
$$[-0{,}1001011]_{ok} = 1{,}1001011.$$

Üldiselt saame kirjutada

$$[+0, x_1x_2 \ldots x_n]_{ok} = 0, x_1x_2 \ldots x_n,$$
$$[-0, x_1x_2 \ldots x_n]_{ok} = 1, x_1x_2 \ldots x_n,$$

kus x_i ($i = 1, 2, \ldots, n$) on arvu kahendnumbrid. Vaadeldes negatiivsete arvude otsekoodi esimesel kohal asuvat numbrit «1» formaalselt kui ühikut, võime lihtkahendmurru (kahendsüsteemi lihtmurru) otsekoodi defineerida ka järgmiselt:

$$[A]_{ok} = \begin{cases} A, & \text{kui } A \geqslant 0, \\ 1 - A, & \text{kui } A \leqslant 0. \end{cases}$$

Arvul null on selle definitsiooni kohaselt kaks erinevat otsekoodi: 0,00 ... 0 ja 1,00 ... 0.

TRÜ Raamatukogu

Otsekoodis säilitatakse kahendarvud näiteks arvuti «Ural-1» salvestusseadmetes, fikseeritud komaga arvud ka teistes «Ural» ja «Minsk» tüüpi arvutites.

Otstarbekohane on kõnelda siin veel ühest teisest koodist, nn. kümnend-kahendkoodist (ehk kahend-kümnendkoodist). Selles koodis kodeeritakse tavalised kümnendsüsteemi arvud kahendsüsteemi numbrite 0 ja 1 abil. Kodeerimine toimub selliselt, et kümnendsüsteemi iga number asendatakse kahendsüsteemi numbrite kindla nelikuga:

0 — 0000,	5 — 0101,
1 — 0001,	6 — 0110,
2 — 0010,	7 — 0111,
3 — 0011,	8 — 1000,
4 — 0100,	9 — 1001.

Nii on näiteks arvu —0,7295 kümnend-kahendkood —0,0111 0010 1001 0101 ehk, kodeerides märgi «—» sümboli 1 abil ja jättes täisosa 0 kirjutamata, 1,0111 0010 1001 0101.

Kümnend-kahendkoodi kasutatakse kümnendsüsteemi arvude arvutisse viimisel ja arvutist väljastamisel. Tuleb märkida, et kümnend-kahendkoodis esitatud arv on ikkagi kümnendsüsteemi arv ja enne arvutamist tuleb see teisendada kahendsüsteemi. Vastavat teisendust võib teha muidugi arvuti enese abil. Tähendame, et kümnend-kahendkoodi kasutatakse nii fikseeritud kui ka liikuva komaga arvutites.

2. Arvude kujutamisel liikuva komaga arvuteis (resp. liikuva koma režiimis) esitatakse nad nn. poollogaritmilisel kujul

$$A = m \cdot 10^p, \quad |m| < 1, \tag{3}$$

kus 10 tähistab mingi positsioonilise q-ndsüsteemi alust. Suurust m nimetatakse arvu A mantissiks, täisarvu p arvu A järguks. Mantiss ja järk peavad sealjuures olema esitatud q-ndsüsteemis. Arvu esitamisel kujul (3) määrab järk p sisuliselt koma asukoha arvus. Seetõttu on põhjendatud ka nimetus *liikuv koma*.

Arvu esitus poollogaritmilisel kujul (kindla q korral) pole muidugi ühene. Nii kehtib näiteks $q = 2$ (s. t. 2-ndsüsteemi) korral võrdus

$$0{,}10100 \cdot 10^{1} = 0{,}01010 \cdot 10^{10}$$

või $q = 10$ (s. t. 10-ndsüsteemi) korral võrdus

$$0{,}0592 \cdot 10^{5} = 0{,}5920 \cdot 10^{4}.$$

Arvu A, mis on esitatud poollogaritmilisel kujul, kus mantissi esimene koht peale koma on nullist erinev, s. t. kus

$$0{,}1\ldots \leqslant |m| < 1, \tag{4}$$

ehk teisiti

$$10^{-1} \leqslant |m| < 1, \tag{4'}$$

nimetatakse *normaliseeritud arvuks*. Arvu (3) teisendamist niisugusele kujule, kus on täidetud tingimus (4) ehk (4'), nimetatakse selle arvu normaliseerimiseks.

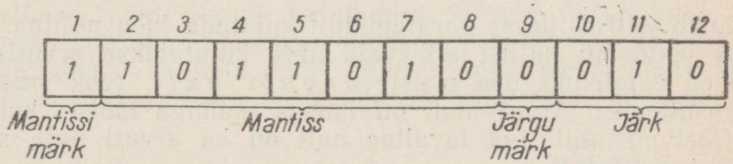
Edaspidi vaatleme poollogaritmilisel kujul esitatud arve, kus $q = 2$. Selguse huvides kirjutame aluse *kaks* aga harilikult kümnendsüsteemi kujul «2». Seega saame võrduse (3) kirjutada kujul

$$A = m \cdot 2^p, \quad |m| < 1 \tag{5}$$

ja normaliseerituse tingimuse (4) ehk (4') kujul

$$\frac{1}{2} \leqslant |m| < 1. \tag{6}$$

Liikuva komaga arvutites (resp. arvuti töötamisel liikuva koma režiimis) on reserveeritud kindel hulk pesa kahendkohti arvu mantissi ja kindel hulk kahendkohti arvu järgu kujutamiseks. Kahendkohtade jaotus pesas näeb põhimõtteliselt välja nii, nagu on kujutatud joonisel 9.



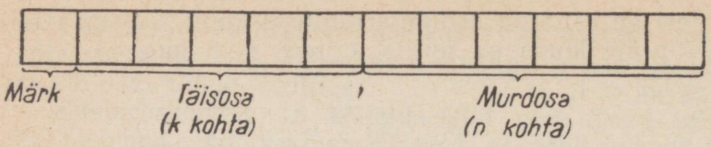
JOON. 9

Joonisel kujutatud pesas salvestatakse antud juhul arv $-0{,}1011010 \cdot 2^{10}$.

Arvutites «Ural-2» ja «Ural-4» jaotatakse pesa kahendkohad liikuva koma režiimis järgmiselt: 1. koht — mantissi (ehk arvu) märk, 2.—33. koht — mantiss, 34. koht — järgu märk, 35.—40. koht — järk. Arvutites «Minsk-2» ja «Minsk-22» on kahendkohtade jaotus järgmine: 1. koht — mantissi märk, 2.—29. koht — mantiss, 30. kohta ei arvestata (selle sisu ei võeta arvesse), 31. koht — järgu märk, 32.—37. koht — järk.

Poollogaritmilisel kujul esitatud kümnendsüsteemi arvude arvutisse viimisel tuleb need kõigepealt kodeerida kümnend-kahendkoodis. Sel korral tulevad ülaltoodud jaotuses väikesed muudatused. Nimelt ei kasutata «Ural-2» ja «Ural-4» tüüpi arvutites sel juhul 35-ndat kahendkohta. Arvutites «Minsk-2» ja «Minsk-22» ei kasutata sel juhul aga 30. ja 32. kahendkohta.

3. Edasi vaatleme arvuti pesades kujutatavate arvude diapasooni nii fikseeritud kui ka liikuva koma režiimis.



JOON. 10

Kui arvuti pesas on fikseeritud koma režiimis k kahendkohta reserveeritud täisosa ja n kahendkohta murdosa kujutamiseks (vt. joonis 10), siis selles pesas kujutatav maksimaalne arv $A_{\max}$ on

$$A_{\max} = \underbrace{11\ldots1}_{k},\quad \underbrace{11\ldots1}_{n} = 2^k - 2^{-n}.$$

Minimaalne positiivne arv $|A|_{\min}$, mida sellises pesas saab kujutada, on

$$|A|_{\min} = \underbrace{0{,}00\ldots01}_{n} = 2^{-n}.$$

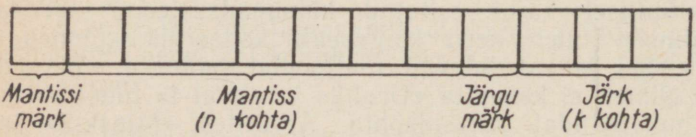
Arvu A, mis rahuldab tingimust

$$-\frac{1}{2}|A|_{\min} < A < \frac{1}{2}|A|_{\min}, \tag{7}$$

tuleb sellises pesas paratamatult kujutada juba nullina.[9] Asjaolu, et teatavaid nullist erinevaid arve kujutatakse arvutis nullina, teeb vajalikuks uue mõiste *arvuti null* (ehk *masina null*) kasutamise. Arvuti null on laiema mahuga mõiste kui tavaline (tõeline) null: iga tavaline null on ka arvuti null, mitte aga ümberpöördult.

Kui arvuti pesas on liikuva koma režiimis n kahendkohta reserveeritud mantissi ja k kahendkohta järgu jaoks, nii nagu on näidatud joonisel 11, siis selles pesas kujutatav maksimaalne arv on

$$A'_{\max} = \underbrace{0{,}11\ldots1}_{n} \cdot 2^{\overbrace{1\ldots1}^{k}} = (1 - 2^{-n})\,2^{2^k - 1}.$$



JOON. 11

[9] Tähendame, et arvu $\frac{1}{2}|A|_{\min}$ võib olenevalt ümardamisest kujutada arvuti pesas kas nullina või kujul $|A|_{\min}$.

Minimaalne positiivne normaliseeritud arv $|A'|_{\min}$ on

$$|A'|_{\min} = 0{,}10\ldots0\cdot 2^{-\overbrace{1\ldots1}^{k}} = 2^{-1}\cdot 2^{-2^k+1} = 2^{-2^k}.$$

Seega asuvad nullist erinevad normaliseeritud arvud diapasoonis

$$2^{-2^k} \leqslant |A| \leqslant (1-2^{-n})2^{2^k-1}.$$

Arvutitel «Ural-2» ja «Ural-4» on liikuva koma režiimi korral $n = 32$ ja $k = 6$. Seega asuvad normaliseeritud arvud diapasoonis

$$2^{-64} \leqslant |A| \leqslant (1-2^{-32})2^{63}$$

ehk kümnendsüsteemis kirjutatult ligikaudu

$$10^{-19} \leqslant |A| \leqslant 10^{19}.$$

Sama võrratus kehtib ka arvutite «Minsk-2» ja «Minsk-22» puhul.

Nagu näeme, on liikuva koma režiimi korral võimalik arve arvuti pesas kujutada küllaltki laias diapasoonis, mistõttu ületäitumist selles režiimis praktiliselt ei esine.

4. Vaatleme nüüd arvude kujutamise täpsust fikseeritud ja liikuva koma režiimis. Seejuures eeldame, et absoluutne viga arvude kujutamisel ei ületa poolt viimase koha (resp. mantissi viimase koha) ühikut. Seega on absoluutne viga fikseeritud koma režiimis

$$\Delta \leqslant \frac{1}{2}\cdot 2^{-n} = 2^{-n-1}$$

ja liikuva koma režiimis

$$\Delta \leqslant \frac{1}{2}\cdot 2^{-n}\cdot 2^{p} = 2^{p-n-1}.$$

Arvu kujutamise suhteline viga (ehk täpsus) on siis fikseeritud koma režiimis järgmistes piirides:

$$\delta_{\min} = \frac{\Delta}{A_{\max}} = \frac{2^{-n-1}}{2^k - 2^{-n}} \approx 2^{-k-n-1},$$

$$\delta_{\max} = \frac{\Delta}{|A|_{min}} = \frac{2^{-n-1}}{2^{-n}} = 2^{-1}.$$

Siit näeme, et arvude kujutamise täpsus varieerub fikseeritud koma režiimis väga suures vahemikus ja võib halvimal juhul ulatuda kuni 50%-ni.

Arvude kujutamise täpsus liikuva koma režiimis (normaliseeritud arvude korral) on järgmistes piirides:

$$\delta_{\min} = \frac{\Delta}{A'_{\max}} = \frac{2^{-n-1} \cdot 2^{2^k-1}}{(1-2^{-n}) \cdot 2^{2^k-1}} \approx 2^{-n-1},$$

$$\delta_{\max} = \frac{\Delta}{|A'|_{\min}} = \frac{2^{-n-1} \cdot 2^{-2^k+1}}{2^{-2^k}} = 2^{-n}.$$

Siit näeme, et normaliseeritud arvude kujutamise täpsus muutub väga vähe ja praktiliselt ei sõltu arvust. Kui opereerida liikuva koma režiimis aga ka mittenormaliseeritud arvudega, siis muutub täpsus niisama suures vahemikus nagu fikseeritud koma režiimi korral.

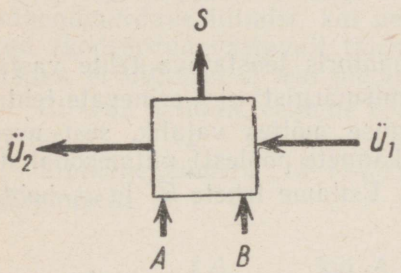
Et arvutuste täpsus määrab ka lõpptulemuse täpsuse, siis tuleb arvutamisel pöörata tähelepanu arvutuste täpsuse tõstmisele ja säilitamisele. Liikuva koma režiimis tähendab see, et tuleb opereerida normaliseeritud arvudega. Tähendame siinkohal, et liikuva koma režiimis töötamisel normaliseeritakse arvutis tehte tulemus automaatselt. Fikseeritud koma režiimis (resp. fikseeritud komaga arvutites) $k = 0$ korral tuleb aga reeglina vältida nulle koma ja esimese nullist erineva numbrikoha vahel. Seda on võimalik vältida arvudele sobivate mastaabikordajate leidmise teel.

Ülaltoodud mõttekäikude põhjal võib tekkida küsimus, kas (segakomaga arvutites) on üldse mõtet kasutada fikseeritud koma režiimi, sest liikuva koma režiimi eelised (ületäitumise praktiline puudumine, suur täpsus) on küllaltki kaalukad. Asi seisneb aga selles, et paljud ülesanded on kergesti lahendatavad fikseeritud koma režiimis ning nende lahendamiseks liikuva koma režiimis pole mingit vajadust, pealegi sooritatakse tehted fikseeritud koma režiimis kiiremini kui liikuva koma režiimis. Analoogilistest kaalutlustest lähtudes on konstrueeritud muide ka fikseeritud komaga arvuteid, mis on mõnevõrra lihtsama ehitusega kui liikuva komaga arvutid.

§ 7. SUMMAATORI TÜÜBID JA NENDELE VASTAVAD KOODID

1. Aritmeetilise seadme üheks põhiliseks osaks on s u m m a a t o r (resp. summaatorid). Summaatori põhimõttelise ehituse põhjal pole raske selgitada näiteks aritmeetiliste tehete sooritamist arvutis. Summaatori ehitusega on otseselt seotud ka mitmed arvutites kasutatavad koodid.

Summaatori ehituse käsitlemist alustame ühekohalisest kahendarvude summaatorist. See on seade kolme ühekohalise kahendarvu liitmiseks. Seadmel on 3 sisendit (arvude A, B ja ülekande $Ü_1$ jaoks) ja 2 väljundit (tulemuse madalama koha S ja kõrgema koha — ülekande $Ü_2$ jaoks). Skemaatiliselt kujutame seda seadet joonisel 12 näidatud viisil.



JOON. 12

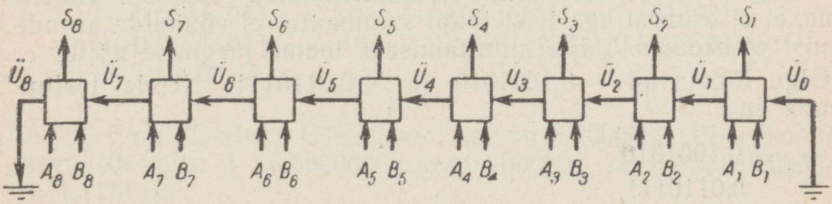
Tabel 4

A	B	$Ü_1$	S	$Ü_2$
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

Ühekohalise kahendarvude summaatori üksikasjalik ehitus pole meile antud juhul oluline. Oluline on ainult väljundsignaalide sõltuvus sisendsignaalidest, mis on esitatud tabelis 4.

Mitme ühekohalise kahendarvude summaatori ühendamisel saame mitmekohalise kahendarvude summaatori. Olenevalt ühekohaliste summaatorite ühendamise viisist saame kaht tüüpi mitmekohalisi summaatoreid.

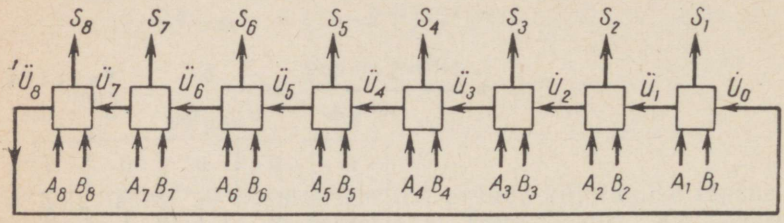
1) Mitmekohaline summaator ilma ülekandeta kõrgemast järgust (ehk 1. tüüpi summaator) on kujutatud joonisel 13.



JOON. 13

Ülekande $\dot{U}_8$ maandamine tähendab seda, et see ülekanne ei leia kasutamist, $\dot{U}_0$ maandatus tähendab, et $\dot{U}_0 = 0$ samaselt.

2) Mitmekohaline summaator tsüklilise ülekandega kõrgeimast järgust madalaimasse järku (ehk 2. tüüpi summaator) on esitatud joonisel 14.



JOON. 14

Tähistame 1. ja 2. liiki summaatoris teostatava tehte vastavalt tehtemärkidega ∅ ja #. Liitmismärgist «+» erinevate tehtemärkide kasutamine on siin selguse mõttes vajalik, sest need tehted siiski erinevad (näiteks ülekannete poolest) mitmekohaliste kahendarvude tavalisest liitmisest. Esitame tehete ∅ ja # kohta mõne näite.

Näide 1.

∅	01101110	(sisendite A_i väärtused)
	10101011	(sisendite B_i väärtused)
(1)	00011001	(väljundite S_i väärtused)

Näide 2.

#	01101110	(sisendite A_i väärtused)
	10101011	(sisendite B_i väärtused)
(1)	00011001	(väljundite S_i esialgsed väärtused)
	00011010	(väljundite S_i lõppväärtused)

Paragrahvis 6 defineerisime lihtkahendmurru otsekoodi. Tähendame, et ei esimest ega teist tüüpi summaator ei võimalda arvude liitmist otsekoodis. Väite kinnitamiseks toome järgmise näite.

Olgu liidetavad —0,1001010 ja +0,0110111. Nende tõeline summa on

+	—0,1001010
	0,0110111
	—0,0010011

Et summa on negatiivne, on selle otsekood 1,0010011. Liidame nüüd 1. ja 2. tüüpi summaatoris liidetavate otsekoodid:

$$\begin{array}{rr} & 1{,}1001010 \\ \varnothing & 0{,}0110111 \\ \hline (1) & 0{,}0000001 \end{array} \qquad \begin{array}{rr} & 1{,}1001010 \\ \# & 0{,}0110111 \\ \hline (1) & 0{,}0000001 \\ & \uparrow \\ & 0{,}0000010 \end{array}$$

Kumbki saadud tulemustest ei ole võrdne tõelise summa —0,0010011 otsekoodiga 1,0010011.

2. Esimest ja teist tüüpi summaatorid võimaldavad aga lihtkahendmurdude liitmist, kui need arvud on spetsiaalselt kodeeritud (kodeeritud vastavalt täiend- või pöördkoodis), muidugi eeldades, et liitmisel ei teki ületäitumist.

Et seda näidata, esitame kõigepealt täiend- ja pöördkoodi definitsiooni.

Lihtkahendmurru A täiendkood $[A]_{tk}$ defineeritakse järgmiselt:

$$[A]_{tk} = \begin{cases} A, & \text{kui } A \geqslant 0, \\ 10 + A, & \text{kui } A < 0. \end{cases}$$

Lihtkahendmurru A pöördkoodi $[A]_{pk}$ defineeritakse järgmiselt:

$$[A]_{pk} = \begin{cases} A, & \text{kui } A \geqslant 0 \\ 10 + A - 10^{-n}, & \text{kui } A \leqslant 0, \end{cases}$$

kus n on numbrikohtade arv pesas.

Nii on näiteks

$$[+0{,}1101011]_{tk} = [+0{,}1101011]_{pk} = 0{,}1101011,$$
$$[-0{,}0101101]_{tk} = 10 + (-0{,}0101101) = 1{,}1010011$$
$$[-0{,}0101101]_{pk} = 1{,}0 + (-0{,}0101101) - 0{,}0000001 =$$
$$= 1{,}1010010.$$

Arvul 0 on üksainus täiendkood, see on 0,0000000. Pöördkoode on arvul 0 kaks: $[+0{,}0000000]_{pk} = 0{,}0000000$ ja $[-0{,}0000000]_{pk} = 1{,}1111111$.

Positiivse arvu täiendkood ja pöördkood, aga samuti otsekood, ühtivad. Negatiivsete arvude korral on nad kõik erine-

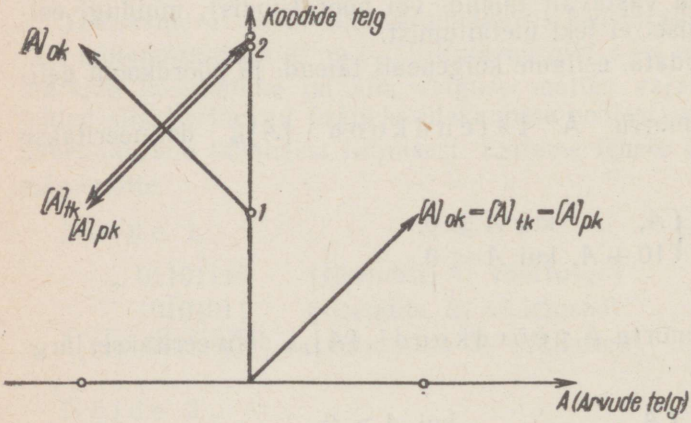
vad. Negatiivsete arvude täiendkoodi on otstarbekohane praktiliselt leida järgmiselt:

$$[-0, x_1x_2 \ldots x_n]_{tk} = 1, \bar{x}_1\bar{x}_2 \ldots \bar{x}_n + \underbrace{0,00 \ldots 01}_{n-1}, \qquad (1)$$

negatiivsete arvude pöördkoodi aga järgmiselt:

$$[-0, x_1x_2 \ldots x_n]_{pk} = 1, \bar{x}_1\bar{x}_2 \ldots \bar{x}_n, \qquad (2)$$

kus x_i $(i = 1, 2, \ldots, n)$ on kahendnumbrid ja $\bar{0} = 1$ ning $\bar{1} = 0$. Valemid (1) ja (2) on kergesti tuletatavad vastavate koodide definitsioonist. Muide, nende valemite põhjal muutuvad arusaadavamaks ka koodide nimetused *täiendkood* ja *pöördkood*.



JOON. 15

Selgitame otse-, täiend- ja pöördkoodi vahekorda joonise 15 abil. Nagu ülaltoodust nähtub, võime täiend- ja pöördkoodi leidmist vaadelda teisendusena, kus vahemiku (—1,1) arvud teisendatakse poollõigu [0,2) arvudeks. Teisendused on üksühesed, kui täiendkoodi puhul jätame vaatlusest välja koodi 1,0000000 ja pöördkoodi puhul vaatleme arve +0 ja —0 kui erinevaid arve. Asjaolu, et koode võib vaadelda kahest väiksemate mittenegatiivsete arvudena, leiab järgnevates mõttekäikudes korduvat kasutamist, näiteks teoreemide 1 ja 2 tõestamisel.

3. Kõigepealt vaatleme 1. ja 2. tüüpi summaatorit, kus koma on fikseeritud pärast esimest kohta, mida vaatleme antud juhul mitte märgikohana, vaid üheliste kohana. Sellise tõlgenduse korral saab nendes summaatorites kujutada arve A, mis rahuldavad tingimust $0 \leqslant A < 2$.

Pole raske näha, et sellise tõlgenduse puhul ning kahest väiksemate mittenegatiivsete arvude A ja B korral annab 1. tüüpi summaator järgmise tulemuse:

$$A \varnothing B = \begin{cases} A+B, & \text{kui } A+B<10, \\ A+B-10, & \text{kui } A+B \geqslant 10, \end{cases} \tag{3}$$

kus 10 tähendab siin ja ka mujal selles paragrahvis arvu kaks. Esimesel juhul saadav tulemus on muidugi ilmne. Teise juhu kohta märgime, et kahest suuremate tulemuste puhul (olenevalt liidetavatest on tulemus alati väiksem neljast) läheb kaduma ülekanne kõrgeimast kohast, s. t. 10 (kaks).

Ülaltoodud tõlgenduse ning kahest väiksemate mittenegatiivsete arvude A ja B puhul annab 2. tüüpi summaator tulemuse:

$$A \# B = \begin{cases} A+B, & \text{kui } A+B<10, \\ A+B-10+10^{-n}, & \text{kui } A+B \geqslant 10. \end{cases} \tag{4}$$

Teise juhu kohta tähendame, et kahest suuremate tulemuste korral liidetakse ülekanne kõrgeimast kohast (s. t. 10) madalaimale (n-ndale) kohale, millest tulenebki lisaliige $-10+10^{-n}$. Käesolevas paragrahvis on konkreetsete koodide korral võetud $n=7$.

4. Sõnastame ja tõestame nüüd eespool nimetatud põhilised tulemused.

Teoreem 1. *Kui* $|A|<1$, $|B|<1$ *ja* $|A+B|<1$, *siis*

$$[A]_{tk} \varnothing [B]_{tk} = [A+B]_{tk}. \tag{5}$$

Ehk teisiti: *1. tüüpi summaator võimaldab absoluutselt ühest väiksemate arvude liitmist, kui arvud on täiendkoodis ja kui liitmisel ei teki ületäitumist.*

Tõestus. Võrduse (5) kehtivus tuleb tõestada neljal erijuhul.

1) Mõlemad liidetavad A ja B on mittenegatiivsed. Seega kehtivad antud juhul võrratused $0 \leqslant A<1$, $0 \leqslant B<1$ ja $0 \leqslant A+B<1$ ning täiendkoodid ühtivad arvude enestega, s. t. $[A]_{tk}=A$, $[B]_{tk}=B$, $[A+B]_{tk}=A+B$. Kontrollime võrduse (5) kehtivust antud erijuhul

$$[A]_{tk} \varnothing [B]_{tk} = A \varnothing B. \tag{6}$$

Võrdust (6) jätkates kasutame seost (3) ja arvestame, et kehtib võrratus $A+B<10$. Seega saame

$$A \varnothing B = A+B = [A+B]_{tk}.$$

2) Üks liidetav on mittenegatiivne, teine negatiivne, summa on mittenegatiivne. Üldsust kitsendamata võime eeldada, et A on mittenegatiivne, seega $0 \leqslant A<1$, $-1<B<0$ ja $0 \leqslant A+$

$+B<1$. Täiendkoodid avalduvad järgmiselt: $[A]_{tk}=A$, $[B]_{tk}=10+B$, $[A+B]_{tk}=A+B$. Kontrollime võrduse (5) kehtivust:

$$[A]_{tk} \oslash [B]_{tk} = A \oslash (10+B). \tag{7}$$

Võrdust (7) jatkates kasutame seost (3) ja arvestame, et kehtib $A+(10+B)\geqslant 10$ (viimane järeldub võrratusest $A+B\geqslant 0$). Saame

$$A \oslash (10+B) = A+(10+B)-10 = A+B = [A+B]_{tk}.$$

3) Üks liidetav on mittenegatiivne, teine negatiivne, summa on negatiivne. Üldsust kitsendamata võime eeldada, et A on mittenegatiivne, seega $0\leqslant A<1$, $-1<B<0$ ja $-1<A+B<0$. Täiendkoodid avalduvad järgmiselt $[A]_{tk}=A$, $[B]_{tk}=10+B$, $[A+B]_{tk}=10+A+B$. Kontrollime võrduse (5) kehtivust:

$$[A]_{tk} \oslash [B]_{tk} = A \oslash (10+B). \tag{8}$$

Võrdust (8) jatkates kasutame seost (3) ja arvestame, et kehtib $A+B+10<10$ (viimane järeldub võrratusest $A+B<0$). Saame

$$A \oslash (10+B) = A+(10+B) = 10+A+B = [A+B]_{tk}.$$

4) Mõlemad liidetavad on negatiivsed. Seega kehtivad võrratused $-1<A<0$, $-1<B<0$ ja $-1<A+B<0$. Täiendkoodid avalduvad järgmiselt $[A]_{tk}=10+A$, $[B]_{tk}=10+B$ ja $[A+B]_{tk}=10+A+B$. Kontrollime võrduse (5) kehtivust:

$$[A]_{tk} \oslash [B]_{tk} = (10+A) \oslash (10+B). \tag{9}$$

Võrdust (9) jatkates kasutame seost (3) ja arvestame, et kehtib $(10+A)+(10+B)>10$ (viimane järeldub võrratusest $A+B>-1$). Saame

$$(10+A) \oslash (10+B) = (10+A)+(10+B)-10 = 10+A+B = [A+B]_{tk}.$$

Sellega on võrdus (5) kõigil neljal juhul tõestatud.

Teoreem 2. *Kui* $|A|<1$, $|B|<1$ ja $|A+B|<1$, siis

$$[A]_{pk} \# [B]_{pk} = [A+B]_{pk}. \tag{10}$$

Ehk teisiti: *2. tüüpi summaator võimaldab absoluutselt ühest väiksemate arvude liitmist, kui arvud on pöördkoodis ja kui liitmisel ei teki ületäitumist.*

Tõestus. Teoreemi 2 tõestamisel tuleks vaadelda samuti nelja juhtumit, analoogiliselt teoreemi 1 tõestusega. Käesoleval juhul tuleb arvestada aga seda, et nullil on 2 pöördkoodi 0,0000000 ja 1,1111111. Seega on otstarbekohane ülalnimetatud neljal juhul

välja jätta arvude võrdumine nulliga ja vaadelda neid juhte eraldi. Teoreemi 2 tõestus nimetatud 4 juhul:

$$\begin{array}{llll} 1) & 0<A<1, & 0<B<1, & 0<A+B<1, \\ 2) & 0<A<1, & -1<B<0, & 0<A+B<1, \\ 3) & 0<A<1, & -1<B<0, & -1<A+B<0, \\ 4) & -1<A<0, & -1<B<0, & -1<A+B<0. \end{array}$$

on analoogiline teoreemi 1 tõestuse vastavate osadega, kusjuures seose (3) asemel tuleb kasutada seost (4). Et olulisi uusi momente tõestuse selles osas ei esine, siis me seda esitama ei hakka.

5) Mõlemad liidetavad on nullid. Vahetult on kontrollitav, et

$$\begin{aligned} [+0]_{pk} \# [+0]_{pk} &= 0{,}0000000 = [+0]_{pk}, \\ [+0]_{pk} \# [-0]_{pk} &= 1{,}1111111 = [-0]_{pk}, \\ [-0]_{pk} \# [-0]_{pk} &= 1{,}1111111 = [-0]_{pk}. \end{aligned}$$

6) Mõlemad liidetavad on nullist erinevad, summa on võrdne nulliga. Olgu $A>0$, $B=-A$. Seega $[A]_{pk}=A$, $[-A]_{pk}=10+(-A)-10^{-n}$. Kontrollime võrduse (10) kehtivust:

$$[A]_{pk} \# [-A]_{pk} = A \# (10+(-A)-10^{-n}). \qquad (11)$$

Võrdust (11) jatkates kasutame seost (4) ja arvestame, et $A+(10+(-A)-10^{-n})<10$. Saame [10]

$$\begin{aligned} A \# (10+(-A)-10^{-n}) &= A+(10+(-A)-10^{-n}) = \\ &= 10-10^{-n} = 1{,}1111111 = [-0]_{pk}. \end{aligned}$$

7) Üks liidetav on nullist erinev, teine liidetav on võrdne nulliga. Olgu $A \neq 0$ ja $B=\pm 0$. Vahetult on kontrollitav, et

$$[A]_{pk} \# [+0]_{pk} = [A]_{pk}.$$

Teisel juhul (s. t. kui $B=-0$) saame

$$[A]_{pk} \# [-0]_{pk} = [A]_{pk} + 1{,}1111111. \qquad (12)$$

Arvestades seda, et eeldusest $A \neq 0$ järeldub $[A]_{pk}+1{,}1111111>10$ saame seost (4) kasutades võrdust (12) jätkata

$$\begin{aligned} [A]_{pk} \# 1{,}1111111 &= [A]_{pk} + (1{,}1111111 - 10 + 10^{-n}) = \\ &= [A]_{pk}. \end{aligned}$$

Seega on tõestatud ka teoreem 2.

Teoreemide illustreerimiseks toome mõned näited.

---

[10] Nagu juba eespool mainitud, võtame selles paragrahvis konkreetsete arvude (resp. koodide) korral $n=7$.

Näide 3. Liitmine 1. tüüpi summaatoris.

$$\begin{array}{rr} A = & 0{,}0101001 \\ + \quad B = & -0{,}1011011 \\ \hline A + B = & -0{,}0110010 \\ [A+B]_{tk} = & 1{,}1001110 \end{array} \qquad \begin{array}{rr} \varnothing & [A]_{tk} = 0{,}0101001 \\ & [B]_{tk} = 1{,}0100101 \\ \hline [A]_{tk} \varnothing [B]_{tk} = & 1{,}1001110 \end{array}$$

Näide 4. Liitmine 1. tüüpi summaatoris.

$$\begin{array}{rr} A = & -0{,}0010010 \\ + \quad B = & -0{,}1011001 \\ \hline A + B = & -0{,}1101011 \\ [A+B]_{tk} = & 1{,}0010101 \end{array} \qquad \begin{array}{rr} \varnothing & [A]_{tk} = 1{,}1101110 \\ & [B]_{tk} = 1{,}0100111 \\ \hline [A]_{tk} \varnothing [B]_{tk} = & (1)\ 1{,}0010101 \end{array}$$

Näide 5. Liitmine 2. tüüpi summaatoris.

$$\begin{array}{rr} A = & -0{,}1011001 \\ + \quad B = & 0{,}0101110 \\ \hline A + B = & -0{,}0101011 \\ [A+B]_{pk} = & 1{,}1010100 \end{array} \qquad \begin{array}{rr} \# & [A]_{pk} = 1{,}0100110 \\ & [B]_{pk} = 0{,}0101110 \\ \hline [A]_{pk} \# [B]_{pk} = & 1{,}1010100 \end{array}$$

5. Selgitame nüüd, kuidas koodide liitmisel (tehte $\varnothing$ ja $\#$ mõttes) saab kindlaks teha ületäitumist. Ületäitumine saab muidugi esineda ainult samamärgiliste liidetavate korral.

Näide 6.

$$\begin{array}{rr} A = & -0{,}0101101 \\ + \quad B = & -0{,}1101010 \\ \hline A + B = & -1{,}0010111 \\ [A+B]_{tk} \text{ pole defineeritud} & \end{array} \qquad \begin{array}{rr} \varnothing & [A]_{tk} = 1{,}1010011 \\ & [B]_{tk} = 1{,}0010110 \\ \hline [A]_{tk} \varnothing [B]_{tk} = & (1)\ 0{,}1101001 \end{array}$$

Koodide liitmisel saime positiivse tulemuse, kuigi liidetavad olid negatiivsed. See asjaolu ongi ületäitumise tunnuseks.

Näide 7.

$$\begin{array}{rr} A = & 0{,}1011001 \\ + \quad B = & 0{,}0101110 \\ \hline A + B = & 1{,}0000111 \\ [A+B]_{pk} \text{ pole defineeritud.} & \end{array} \qquad \begin{array}{rr} \# & [A]_{pk} = 0{,}1011001 \\ & [B]_{pk} = 0{,}0101110 \\ \hline [A]_{pk} \# [B]_{pk} = & 1{,}0000111 \end{array}$$

Koodide liitmisel saime negatiivse tulemuse, kuigi liidetavad olid positiivsed. Viimane asjaolu ongi ületäitumise tunnuseks.

Kasutades 1. ja 2. liiki summaatoris vastavalt täiend- ja pöördkoodi, saab ületäitumise kindlaks teha liidetavate ja tulemuse märkide võrdlemisel. Kuid ületäitumist võib põhimõtteliselt kindlaks teha ka ainuüksi lõpptulemuse põhjal. Selleks tuleb täiendkoodi ja pöördkoodi asemel kasutada vastavalt modifitseeritud täiendkoodi ja modifitseeritud pöördkoodi, kus arvu märk kodee-

ritakse kahe esimese kahendkohaga. Esitame nende koodide definitsioonid.

Lihtkahendmurru A modifitseeritud täiendkood A_{mtk} defineeritakse järgmiselt:

$$[A]_{mtk} = \begin{cases} A, & \text{kui } A \geqslant 0, \\ 100 + A, & \text{kui } A < 0. \end{cases}$$

Lihtkahendmurru A modifitseeritud pöördkood A_{mpk} defineeritakse järgmiselt:

$$[A]_{mpk} = \begin{cases} A, & \text{kui } A \geqslant 0, \\ 100 + A - 10^{-n}, & \text{kui } A \leqslant 0. \end{cases}$$

Definitsioonidest nähtub, et mittenegatiivsete arvude modifitseeritud koodid ühtivad nende arvude hariliku kujuga 2-ndsüsteemis (ainukeseks erinevuseks on, et enne koma kirjutatakse kaks nulli). Kerge on näha, et negatiivsete arvude korral saab modifitseeritud koode arvutada järgmiselt:

$$[-0, x_1x_2 \ldots x_n]_{mtk} = 11, \bar{x}_1\bar{x}_2 \ldots \bar{x}_n + 10^{-n} \tag{13}$$

ja

$$[-0, x_1x_2 \ldots x_n]_{mpk} = 11, \bar{x}_1\bar{x}_2 \ldots \bar{x}_n. \tag{14}$$

Toome mõned näited modifitseeritud koodide kohta:

$$[0{,}1101101]_{mtk} = 00{,}1101101,$$
$$[-0{,}1101101]_{mtk} = 11{,}0010011,$$
$$[-0{,}1101101]_{mpk} = 11{,}0010010.$$

Arvul 0 on üksainus modifitseeritud täiendkood ja 2 modifitseeritud pöördkoodi $[+0]_{mpk} = 00{,}0000000$, $[-0]_{mpk} = 11{,}1111111$. Tähendame, et modifitseeritud koode võib vaadelda arvust $100_{(2)}$ väiksemate mittenegatiivsete arvudena.

Kui 1. ja 2. tüüpi summaatoris on koma fikseeritud pärast teist kohta, kusjuures kaht esimest kohta me ei tõlgenda mitte märgikohtadena, vaid tavaliste numbrikohtadena (esimene koht on kaheliste, teine üheliste tähistamiseks), siis võime analoogiliselt eespool toodud tulemustega (3) ja (4) väita järgmist: arvust $100_{(2)}$ väiksemate mittenegatiivsete arvude A ja B korral annab 1. tüüpi summaator tulemuse

$$A \varnothing B = \begin{cases} A + B, & \text{kui } A + B < 100, \\ A + B - 100, & \text{kui } A + B \geqslant 100 \end{cases} \tag{15}$$

ja 2. tüüpi summaator tulemuse

$$A \# B = \begin{cases} A + B, & \text{kui } A + B < 100, \\ A + B - 100 + 10^{-n}, & \text{kui } A + B \geqslant 100. \end{cases} \tag{16}$$

Seoseid (15) ja (16) võib põhjendada analoogiliselt seoste (3) ja (4) põhjendustega.

Eeldades, et $|A| < 1$, $|B| < 1$ ja $|A + B| < 1$ saab tõestada järgmised võrdused

$$[A]_{mtk} \# [B]_{mtk} = [A + B]_{mtk} \quad (17)$$
$$[A]_{mpk} \# [B]_{mpk} = [A + B]_{mpk}. \quad (18)$$

Seega: 1. tüüpi summaator võimaldab arvusid liita modifitseeritud täiendkoodis, 2. tüüpi summaator aga modifitseeritud pöördkoodis.

Et võrduste (17) ja (18) tõestus on analoogiline vastavalt teoreemide 1 ja 2 tõestusega, siis jätame need esitamata. Märgime ainult seda, et nendes tõestustes tuleb kasutada vastavalt seoseid (15) ja (16).

Toome siiski mõned näited liitmise kohta modifitseeritud koodides. Eriti huvipakkuvad on muidugi ületäitumise juhud.

Näide 8.

$$\begin{array}{rr} & A = \ 0{,}1011110 \\ + & B = -0{,}0101001 \\ \hline A+B & = \ 0{,}0110101 \end{array} \qquad \begin{array}{rr} & [A]_{mtk} = 00{,}1011110 \\ \varnothing & [B]_{mtk} = 11{,}1010111 \\ \hline & [A]_{mtk} \varnothing [B]_{mtk} = (1)\,00{,}0110101 \end{array}$$

$$[A+B]_{mtk} = \ 00{,}0110101$$

Näide 9.

$$\begin{array}{rr} & A = -0{,}0110011 \\ + & B = -0{,}0101110 \\ \hline A+B & = -0{,}1100001 \end{array} \qquad \begin{array}{rr} & [A]_{mpk} = 11{,}1001100 \\ \# & [B]_{mpk} = 11{,}1010001 \\ \hline & (1)\,11{,}0011101 \\ & \uparrow \end{array}$$

$$[A+B]_{mpk} = \ 11{,}0011110 \qquad [A]_{mpk} \# [B]_{mpk} = 11{,}0011110$$

Näide 10. Ületäitumine, mis tekib positiivsete arvude liitmisel.

$$\begin{array}{rr} & A = 0{,}1001101 \\ & B = 0{,}1011011 \\ \hline A+B & = 1{,}0101000 \end{array} \qquad \begin{array}{rr} & [A]_{mtk} = 00{,}1001101 \\ \varnothing & [B]_{mtk} = 00{,}1011011 \\ \hline & [A]_{mtk} \varnothing [B]_{mtk} = 01{,}0101000 \end{array}$$

$[A+B]_{mtk}$ pole defineeritud.

Näide 11. Ületäitumine, mis tekib negatiivsete arvude liitmisel.

$$\begin{array}{rr} & A = -0{,}1001101 \\ + & B = -0{,}1011011 \\ \hline A+B & = -1{,}0101000 \end{array} \qquad \begin{array}{rr} & [A]_{mpk} = 11{,}0110010 \\ \# & [B]_{mpk} = 11{,}0100100 \\ \hline & (1)\,10{,}1010110 \\ & \uparrow \end{array}$$

$[A+B]_{mpk}$ pole defineeritud. $[A]_{mpk} \# [B]_{mpk} = 10{,}1010111$

Ületäitumise kriteeriumiks moditseeritud koodide korral on erinevad numbrid kahel esimesel kohal enne koma. Numbrite «01» korral kahel esimesel kohal on tegemist ületäitumisega, mis tekkis positiivsete arvude liitmisel (nn. positiivse ületäitumisega), numbrite «10» korral kahel esimesel kohal on tegemist ületäitumisega, mis tekkis negatiivsete arvude liitmisel (nn. negatiivse ületäitumisega).

6. Nagu toodud näidetest selgub, on ületäitumise kindlakstegemine modifitseeritud koodide korral palju hõlpsam kui hariliku täiend- ja pöördkoodi korral. Seetõttu kasutatakse arvutites enamasti modifitseeritud koode. Nii töötab näiteks arvutite «Ural-2» ja «Ural-4» summaator (arvude summaator) modifitseeritud pöördkoodis, kusjuures pärast tehte sooritamist teisendatakse tulemus automaatselt otsekoodi.

Lõpuks teeme veel mõned märkused teiste aritmeetiliste tehete kohta.

Arvude A ja B lahutamine taandatakse liitmisele

$$A - B = A + (-B)$$

arvu B märgi muutmise teel.

Arvude korrutamine arvutis toimub peaaegu tavalisel viisil: leitakse järjestikused osakorrutised ja liidetakse need summaatoris, kusjuures korrutamist alustatakse kõrgeimast järgust.

Jagamisel toimub järjestikune lahutamine, kusjuures jagatavast (mis asub summaatoris) lahutame jagaja, millega kaasneb jagatava (lahutamisel saadava vahe) järk-järguline nihutamine vasakule.

Aritmeetiliste tehete sooritamine liikuva koma režiimis toimub arvudega kujul

$$\begin{aligned} A_1 &= m_1 \cdot 2^{p_1}, \qquad |m_1| < 1, \\ A_2 &= m_2 \cdot 2^{p_2}, \qquad |m_2| < 1, \end{aligned}$$

järgmisel viisil.

Liitmise korral tuleb enne mantisside liitmist arvude järgud võrdsustada. Seda saab teha ainult väiksema järgu suurendamise ja vastava mantissi paremale nihutamise teel.

$$\begin{aligned} A = A_1 + A_2 &= m_1 \cdot 2^{p_1} + m_2 \cdot 2^{p_2} = \\ &= \begin{cases} (m_1 + 2^{p_2 - p_1} \cdot m_2) \cdot 2^{p_1}, & \text{kui } p_1 \geqslant p_2, \\ (2^{p_1 - p_2} \cdot m_1 + m_2) \cdot 2^{p_2}, & \text{kui } p_1 < p_2. \end{cases} \end{aligned} \tag{19}$$

Korrutamine toimub järgmiselt:

$$A = A_1 \cdot A_2 = m_1 \cdot 2^{p_1} \cdot m_1 \cdot 2^{p_2} = (m_1 \cdot m_2) 2^{p_1 + p_2}. \tag{20}$$

Jagamise korral saame

$$A = A_1 : A_2 = (m_1 \cdot 2^{p_1}) : (m_2 \cdot 2^{p_2}) = \frac{m_1}{m_2} \cdot 2^{p_1 - p_2}. \tag{21}$$

Arvude mantissid arvutatakse arvude summaatoris, järgud järgusummaatoris (näit. arvutite «Ural-2» ja «Ural-4» korral) põhimõtteliselt samuti nagu fikseeritud koma režiimis. Sellejuures tuleb märkida, et tulemuse mantiss, näiteks $m_1 + 2^{p_2-p_1} \cdot m_2$, $m_1 \cdot m_2$ või $\frac{m_1}{m_2}$ ei tarvitse esialgu osutuda normaliseerituks. Mantissi arvutamisel valemite (19), (20) ja (21) järgi võib esineda ületäitumine, kusjuures tulemus ei saa ületada arvu 2. Selline ületäitumine on likvideeritav mantissi nihutamisega ühe koha võrra paremale ühes järgu suurendamisega 1 võrra. Mantissi arvutamisel võib esineda aga ka nn. alatäitumine (mantissi esimesed kohad on nullid). Sel korral nihutatakse mantiss vasakule ja järku vähendatakse nihutatud kohtade arvu võrra. Mantissi normaliseerimine (nii üle- kui ka alatäitumise korral) ühes vastava paranduse tegemisega järgus toimub arvuti töö käigus automaatselt.

Näiteid vastavate tehete kohta võivad lugejad ilma eriliste raskusteta leida või koostada juba iseseisvalt.

§ 8. LAUSEARVUTUSE ALGMEID

1. Lausearvutus on matemaatilise loogika põhilisemaid arvutussüsteeme, mis leiab kasutamist ka programmeerimisel (loogilised tehted, programmi loogilise skeemi koostamine ja teisendamine, automaatne programmeerimine jne.).

Lausearvutust võib üles ehitada puht-loogilistest kontseptsioonidest lähtudes. Programmeerimise seisukohalt pole loogiliste aspektide üksikasjalik esitamine aga eriti oluline. Seega võib siin lausearvutuse üles ehitada analoogiliselt traditsiooniliste matemaatiliste distsipliinidega, nagu näiteks algebra või funktsiooniteooria. Lausearvutuse sellisel käsitlemisel taandubki see tegelikult kahendmuutuja funktsioonide teooriale.[11]

Lausearvutuse sellist abstraheeritud käsitlust on sobiv alustada kahendmuutuja mõistest. K a h e n d m u u t u j a on muutuja, mis võib omandada väärtusi mingist kaheelemendilisest hulgast. Selle hulga elemente võib põhimõtteliselt tähistada muidugi mitmeti, näiteks sõnadega *true* ja *false*, tähtedega *t* ja *v* või sümbolitega 1 ja 0. Valime selles paragrahvis viimase võimaluse. Sümboleid 1 ja 0 nimetatakse l o o g i l i s t e k s v ä ä r t u s t e k s ehk l o o g i l i s t e k s k o n s t a n t i d e k s (ka B o o l e ' i k o n s t a n t i d e k s ja t õ e v ä ä r t u s t e k s). Kahendmuutujaid tähistatakse tavaliselt suurte ladina tähtedega $A, B, C, \ldots$. Asjaolu, et mingi

[11] Kõne all olevat distsipliini (eriti loogikaga vähem seostatud käsitluse korral) nimetatakse mõne autori poolt ka Boole'i algebraks, mõnel juhul aga *loogika algebraks*. Tuleb aga märkida, et Boole'i algebra on lausearvutusest laiem valdkond. Termin *loogika algebra* pole aga sisuliselt vastuvõetav ja selle kasutamisest oleks parem üldse loobuda.

kahendmuutuja, näiteks A, omandab väärtuse 1 (resp. väärtuse 0) tähistame tavalisel viisil $A = 1$ (resp. $A = 0$).

Loogilisest seisukohast saab kõike seda interpreteerida järgmiselt: kahendmuutujad on väited (laused, tunnused, tingimused), mis võivad vastata tegelikkusele ($A = 1$) või ka tegelikkusele mitte vastata ($A = 0$).

2. Vaatleme nüüd kõikvõimalikke kahe argumendiga kahendmuutuja funktsioone. Pole raske näha, et neid on kokku 16. Esitame need alljärgnevas tabelis 5. Selles tabelis on ruumipuudusel funktsioonide $f_i(A, B)$ asemel kirjutatud lihtsalt f_i ($i = 1, 2, \ldots, 16$).

Tabel 5

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

Osa nendest funktsioonidest on otsese loogilise tähendusega. Nendele on antud eri nimetused ja tähised. Neid vaadeldakse kui loogilisi tehteid, analoogiliselt tehetega aritmeetikas. Esitame olulisemad loogilised tehted:

1) $f_2(A, B)$ — seda tehet nimetatakse konjunktsiooniks, tehtemärgina kasutatakse tavalist märki $\wedge$ (vahel ka märki &), mis kirjutatakse muutujate vahele (s. t. $A \wedge B$ või vastavalt $A \& B$). Tavalises keeles (s. t. sisulises tõlgenduses) vastab sellele tehtele sidesõna *ja*.

2) $f_8(A, B)$ — disjunktsioon, tehtemärgiks on $\vee$ ($A \vee B$). Tema keeleliseks (sisuliseks) vasteks on sidesõna *või* mittevälistavas mõttes.

3) $f_{14}(A, B)$ — implikatsioon, tehtemärgiks on $\supset$ ($A \supset B$), vahel ka $\rightarrow$. Tema keeleliseks vasteks on väljend *kui A, siis B*.

4) $f_{10}(A, B)$ — ekvivalents, tehtemärgiks on $\sim$ ($A \sim B$). Tema keeleliseks vasteks on väljend *A on samatõene B-ga*.

5) $f_7(A, B)$ — mitteekvivalents, tehtemärgiks on $\not\sim$ ($A \not\sim B$). Tema keeleliseks vasteks on sidesõna *või* välistavas mõttes ehk väljend *A pole samatõene B-ga*.

6) $f_{15}(A, B)$ — Shefferi tehe (ka Shefferi kriips), tehtemärgiks on $/$ (A / B). Tema keeleliseks vasteks on väljend *A ja B pole mõlemad tõesed*.

Esitatud tehted on binaarsed, s. t. rakendatavad kahele komponendile. Peale nende kasutatakse aga veel üht unaarset tehet — eitust. Eituse korral kasutame tehtemärkidena märki $\neg$ või $\bar{}$,

mille kirjutame vastavalt muutuja ette ($\neg A$) või muutuja kohale ($\bar{A}$). Eituse definitsioon antakse tabeliga 6.

Tabel 6

A	¬A
0	1
1	0

S. t. $\bar{0} = 1$ ja $\bar{1} = 0$. Muide, eituse saab tuletada ka funktsioonist $f_{13}(A, B)$, kui arvestada, et see ei sõltu üldse teisest muutujast B ja on seega tegelikult ühe muutuja funktsioon. Eituse keeleliseks vasteks on väljendid *mitte-A, A ei kehti* vms.

3. Edasi defineerime induktiivselt loogilise valemi ehk loogilise avaldise.

1. Loogilised konstandid 0 ja 1 on loogilised valemid.
2. Kõik kahendmuutujad on loogilised valemid.
3. Kui $\mathfrak{B}$ ja $\mathfrak{A}$ on loogilised valemid, siis on
   $(\mathfrak{A} \wedge \mathfrak{B})$, $(\mathfrak{A} \vee \mathfrak{B})$, $(\mathfrak{A} \supset \mathfrak{B})$, $(\mathfrak{A} \sim \mathfrak{B})$, $)\mathfrak{A} \approx \mathfrak{B})$ ja $(\mathfrak{A}/\mathfrak{B})$
   loogilised valemid.
4. Kui $\mathfrak{A}$ on loogiline valem, siis on $\neg\mathfrak{A}$ loogiline valem.

Loogilise valemi näitena olgu toodud valemid

$$(((A \wedge \neg B) \supset 0) \sim (1/C)) \tag{1}$$

ja

$$\neg((((A_1 \vee A_2) \vee A_3) \vee A_4) \supset (B \approx C)). \tag{2}$$

Et avaldised (1) ja (2) on tõepoolest loogilised valemid, pole raske kontrollida loogilise valemi definitsioonist lähtudes. Sulgude arvu vähendamiseks lepime kokku, kuidas määrata, missugune kahest naabertehtest tuleb sooritada varem. Kui sulgudega pole määratud teisiti, on tehete järjekord järgmine: 1. ¬, 2. $\wedge$, 3. $\vee$, 4. $\supset$, 5. $\sim$, $\approx$, / (viimase kolme tehte omavaheline järjekord tuleb määrata juba sulgudega). Samuti võib sulud kirjutamata jätta mitmeliikmelises konjunktsioonis $A_1 \wedge A_2 \wedge \ldots \wedge A_n$ ja disjunktsioonis $A_1 \vee A_2 \vee \ldots \vee A_n$, kus muutujad A_i võivad olla ka eitustega. Viimane kokkulepe tugineb konjunktsiooni ja disjunktsiooni assotsiatiivsusel (vt. samasused (7) ja (8)). Kui valemi ees pole eitust, siis võib kirjutamata jätta ka kõige välimise sulgudepaari.

Ülaltoodud kokkulepete põhjal võib valemid (1) ja (2) kirjutada lühemal kujul

$$A \wedge \neg B \supset 0 \sim (1/C) \tag{1'}$$

ja

$$A_1 \vee A_2 \vee A_3 \vee A_4 \supset (B \approx C). \tag{2'}$$

On kerge näha, et iga loogiline valem defineerib teatava kahendmuutuja funktsiooni. Nii näiteks defineerib valem (1) (või (1′)) kahendmuutuja funktsiooni $f(A, B, C)$ (vt. tabelit 7).

Argumentide A, B ja C fikseeritud väärtuste korral leiame funktsiooni $f(A, B, C)$ väärtuse valemist (1) või (1′), kui teeme selles valemis ettenähtud loogilised tehted (tehete järjekord on tabelis 7 märgitud numbritega (1), (2), ...), kusjuures tehete tulemused kirjutame vastavate tehtemärkide alla. Valemi (1), seega ka funktsiooni $f(A, B, C)$ väärtuse saame viimase tehtemärgi $\sim$ all.

Tabel 7

A (1)	B (2)	C (3)	A (1)	$\vee$ (5)	$\neg$ (4)	B (2)	$\supset$ (6)	0	$\sim$ (8)	(1 / C) (7)(3)	$f(A, B, C)$ (8)
0	0	0	0	0	1	0	1	0	1	1 1 0	1
0	0	1	0	0	1	0	1	0	0	1 0 1	0
0	1	0	0	0	0	1	1	0	1	1 1 0	1
0	1	1	0	0	0	1	1	0	0	1 0 1	0
1	0	0	1	1	1	0	0	0	0	1 1 0	0
1	0	1	1	1	1	0	0	0	1	1 0 1	1
1	1	0	1	0	0	1	1	0	1	1 1 0	1
1	1	1	1	0	0	0	1	0	0	1 0 1	0

4. Loogiliste valemite teisendamisel on põhiline valemite (loogilise) samaväärsuse mõiste. Loogiliste valemite $\mathfrak{A}$ ja $\mathfrak{B}$ samaväärsust mõistame kui valemitele $\mathfrak{A}$ ja $\mathfrak{B}$ vastavate kahendmuutuja funktsioonide väärtuste võrdsust argumentide (muutujate) ühe ja sama väärtuste komplekti korral. Valemite $\mathfrak{A}$ ja $\mathfrak{B}$ samaväärsust tähistame

$$\mathfrak{A} \equiv \mathfrak{B}$$

Nii on näiteks valemid $A \supset B$ ja $\neg B \supset \neg A$ samaväärsed, mida saab kindlaks teha nn. tabelimeetodi abil. Tabelis 8 esiletõstetud veergudest nähtub, et muutujate A ja B väärtuste samade komplektide korral on valemite $A \supset B$ ja $\neg B \supset \neg A$ väärtused võrdsed.

Tabel 8

A	B	A	$\supset$	B	$\neg$	B	$\supset$	$\neg$	A
0	0	0	1	0	1	0	1	1	0
0	1	0	1	1	0	1	1	1	0
1	0	1	0	0	1	0	0	0	1
1	1	1	1	1	0	1	1	0	1

Seega tõepoolest

$$A \supset B \equiv \neg B \supset \neg A.$$

Märgime, et samaväärsetel valemitel ei tarvitse tingimata olla ühed ja samad muutujad. Nii näiteks kehtib loogiline samaväärsus

$$A \equiv A \wedge (B \vee \overline{B}), \tag{3}$$

sellest hoolimata, et vasakpoolses valemis ei esine üldse muutujat B. Sisuliselt seisneb asi selles, et muutuja B parempoolses valemis mingisugust funktsionaalset rolli üldse ei mängi ja esineb seal ainult nn. fiktiivse muutujana.[12] Valemite loogilise samaväärsuse kontrollimisel pole fiktiivsete muutujate eraldamine aga tingimata vajalik. Kontrollimisel võib lihtsalt vaadelda mõlemaid valemeid (kontrollitava samaväärsuse pooli) kui mõlema poole kõikide muutujate (antud juhul muutujate A ja B) funktsioone. Loogiliste valemite teisendamiseks, muuhulgas lihtsustamiseks, on vaja teada olulisemaid loogilisi samaväärsusi. Need on järgmised:

$$\neg\neg A \equiv A \tag{4}$$

(kahekordse eituse võib ära jätta);

$$A \wedge B \equiv B \wedge A \tag{5}$$
$$A \vee B \equiv B \vee A \tag{6}$$

(konjunktsiooni ja disjunktsiooni kommutatiivsus);

$$(A \wedge B) \wedge C \equiv A \wedge (B \wedge C) \tag{7}$$
$$(A \vee B) \vee C \equiv A \vee (B \vee C) \tag{8}$$

(konjunktsiooni ja disjunktsiooni assotsiatiivsus);

$$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C) \tag{9}$$
$$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C) \tag{10}$$

(konjunktsiooni distributiivsus disjunktsiooni suhtes ja disjunktsiooni distributiivsus konjunktsiooni suhtes);

$$\neg(A \wedge B) \equiv \overline{A} \vee \overline{B} \tag{11}$$
$$\neg(A \vee B) \equiv \overline{A} \wedge \overline{B} \tag{12}$$

---

[12] Fiktiivne muutuja defineeritakse järgmiselt: loogilise valemi või kahendmuutuja funktsiooni $f(X_1, \ldots, X_i, \ldots, X_n)$ muutujat X_i nimetatakse fiktiivseks, kui võrdus $f(X_1, \ldots, X_{i-1}, 0, X_{i+1}, \ldots, X_n) = f(X_1, \ldots, X_{i-1}, 1, X_{i+1}, \ldots, X_n)$ kehtib muutujate $X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n$ kõikide väärtuste korral.

(konjunktsiooni ja disjunktsiooni duaalsus[13]);

$$(A \supset B) \equiv \overline{A} \vee B \tag{13}$$

$$A \sim B \equiv (A \supset B) \wedge (B \supset A) \equiv (\overline{A} \vee B) \wedge (A \vee \overline{B}) \equiv$$
$$\equiv (A \wedge B) \vee (\overline{A} \wedge \overline{B}) \tag{14}$$

$$A \approx B \equiv \neg(A \sim B) \equiv (A \wedge \overline{B}) \vee (\overline{A} \wedge B) \equiv (A \vee B) \wedge (\overline{A} \vee \overline{B}) \tag{15}$$

$$A/B \equiv \neg(A \wedge B) \equiv \overline{A} \vee \overline{B} \tag{16}$$

Esitame veel mõningaid loogilisi samaväärsusi, mis on vajalikud eriti valemite lihtsustamisel:

$$A \vee (A \wedge B) \equiv A \tag{17}$$
$$A \wedge (A \vee B) \equiv A \tag{18}$$
$$A \wedge A \equiv A \tag{19}$$
$$A \wedge \overline{A} \equiv 0 \tag{20}$$
$$A \wedge 1 \equiv A \tag{21}$$
$$A \wedge 0 \equiv 0 \tag{22}$$
$$A \vee A \equiv A \tag{23}$$
$$A \vee \overline{A} \equiv 1 \tag{24}$$
$$A \vee 1 \equiv 1 \tag{25}$$
$$A \vee 0 \equiv A \tag{26}$$
$$0 \supset A \equiv 1 \tag{27}$$
$$A \supset 0 \equiv \overline{A} \tag{28}$$
$$A \sim 1 \equiv A \tag{29}$$
$$A \sim 0 \equiv \overline{A} \tag{30}$$
$$A \approx 1 \equiv \overline{A} \tag{31}$$
$$A \approx 0 \equiv A \tag{32}$$
$$A/1 \equiv \overline{A} \tag{33}$$
$$A/0 \equiv 1 \tag{34}$$

Samaväärsuste (4)—(34) kehtivust võib tõestada tabelimeetodil. Paljud nendest, nagu (4), (5), (6) ja mitmed teised, on peaaegu ilmsed. Loogiliste samaväärsuste loetelu võib muidugi lõputult jätkata.

5. Valemite teisendamine põhineb järgmisel asjaolul. Kui mingis valemis $\mathfrak{A}$ esinev osavalem $\mathfrak{B}$ asendatakse valemiga $\mathfrak{C}$, mis on samaväärne valemiga $\mathfrak{B}$, siis saadakse esialgse valemiga $\mathfrak{A}(\mathfrak{B})$ loogiliselt samaväärne valem $\mathfrak{A}(\mathfrak{C})$. Ehk kokkuvõtlikult: kui $\mathfrak{B} \equiv \mathfrak{C}$, siis $\mathfrak{A}(\mathfrak{B}) \equiv \mathfrak{A}(\mathfrak{C})$.

Näitame, kuidas toimub loogiliste valemite teisendamine (lihtsustamine) konkreetsetel juhtudel.

Näide 1. Teisendada valem

$$A \wedge \neg B \supset 0 \sim (1/C) \tag{1'}$$

kujule, kus tehetest esinevad ainult konjunktsioon, disjunktsioon ja eitus ning kus eitus on vahetult üksikute muutujate ees (resp. kohal).

Samaväärsusele (28) tuginedes saame kirjutada

$$A \wedge \neg B \supset 0 \equiv \neg(A \wedge \neg B),$$

[13] Konjunktsiooni ja disjunktsiooni duaalsus on veelgi üldisema iseloomuga. Võib tõestada, et kui mingi valem $\mathfrak{A}(A_1, A_2, \ldots, A_n; \wedge, \vee, \neg)$ on moodustatud ainult konjunktsiooni, disjunktsiooni ja eituse tehte abil, siis selle eitus $\neg\mathfrak{A}(\bar{A}_1, \bar{A}_2 \ldots, A_n; \wedge, \vee, \neg)$ on loogiliselt samaväärne valemiga $\mathfrak{A}(\bar{A}_1, \bar{A}_2, \ldots, \bar{A}_n; \vee, \wedge, \neg)$, s. t. muutujad tuleb asendada nende eitustega ning konjunktsiooni- ja disjunktsioonimärgid tuleb vahetada.

millest duaalsuse seose abil leiame

$$\neg(A \wedge \neg B) \equiv \neg A \vee \neg\neg B,$$

kust kahekordset eitust ära jättes saame

$$\overline{A} \vee B.$$

Osavalemit $1/C$ saame teisendada, arvestades Shefferi tehte kommutatiivsust ja seost (33). Nii leiame

$$1/C \equiv \overline{C}.$$

Asendades leitud valemid valemisse (1′), omandab see kuju

$$A \wedge \overline{B} \supset 0 \sim (1/C) \equiv \overline{A} \vee B \sim \overline{C}.$$

Samaväärsust (14) kasutades võime nüüd kirjutada

$$\overline{A} \vee B \sim \overline{C} \equiv (\overline{A} \vee B \supset \overline{C}) \wedge (\overline{C} \supset \overline{A} \vee B) \equiv (\neg(\overline{A} \vee B) \vee \overline{C}) \wedge$$
$$\wedge (\overline{\overline{C}} \vee \overline{A} \vee B),$$

millest seoste (4), (6) ja (12) abil saame lõpuks

$$(A \wedge \overline{B} \vee \overline{C}) \wedge (\overline{A} \vee B \vee C).$$

Näide 2. Lihtsustada valem

$$(\overline{A} \wedge C \supset (A \sim 0)) \wedge \overline{B} \supset (A \sim B). \qquad (35)$$

Asendame $A \sim 0$ samasuse (30) alusel ja kasutame esimese implikatsiooni puhul seost (13) ning seejärel seost (11). Nii saame

$$(\overline{\overline{A}} \vee \overline{C} \vee \overline{A}) \wedge \overline{B} \supset (A \sim B).$$

Osavalemit $\overline{\overline{A}} \vee \overline{C} \vee \overline{A}$ tohib seoste (4), (6), (8), (24) ja (25) põhjal teisendada järgmiselt:

$$\overline{\overline{A}} \vee \overline{C} \vee \overline{A} \equiv (A \vee \overline{A}) \vee \overline{C} \equiv 1 \wedge \overline{C} \equiv 1.$$

Liikme 1 võib aga seose (21) põhjal konjunktsioonis üldse kirjutamata jätta. Seega saame

$$(\overline{\overline{A}} \vee \overline{C} \vee \overline{A}) \wedge \overline{B} \supset (A \sim B) \equiv \overline{B} \supset (A \sim B).$$

Kasutades seoseid (13), (4) ja (14) leiame nüüd

$$\overline{B} \supset (A \sim B) \equiv B \vee (A \wedge B) \vee (\overline{A} \wedge \overline{B}),$$

millest seose (17) põhjal saame

$$B \vee (\overline{A} \wedge \overline{B}),$$

mis on tunduvalt lihtsam kui lähtevalem (35).

III PEATÜKK

ARVUTI KÄSKUDE SÜSTEEM

Arvutis teostatavate operatsioonide loetelu, samuti aga ka vastavate käskude kuju on määratud arvuti konstruktsiooniga. Erinevate arvutite puhul on need märgatavalt erinevad. Seetõttu ei ole mõtet esitada mingi abstraktse arvuti käskude süsteemi kirjeldust — käskude süsteemist tuleb alati rääkida seoses teatava konkreetse arvutitüübiga. Käesolevas õpikus on nendeks konkreetseteks arvutitüüpideks valitud «Ural-4» ja «Minsk-22», kui peamised Eesti NSV-s kasutatavad arvutid.

§ 9. ÜLDISI ANDMEID ARVUTI «URAL-4» KOHTA.

1. Universaalne numbriline elektronarvuti «Ural-4» on üheaadressiline. Ta teeb tehteid nii liikuva kui ka fikseeritud komaga kahendarvudega. Fikseeritud komaga arvud esitatakse arvutis kas 20- või 40-kohalistena (märgikoht kaasa arvatud). Et koma asub vahetult märgikoha järel, siis peavad kõik arvutis «Ural-4» kasutatavad fikseeritud komaga arvud x kuuluma lõiku:

$$2^{-39} \leqslant |x| \leqslant 1 - 2^{-39}.$$

Fikseeritud komaga arvud, mis absoluutväärtuse poolest on väiksemad kui 2^{-39}, kujutavad arvutis nulle (nn. *arvuti nullid*). Kui arvutamisel saadakse tulemus, mis on absoluutväärtuselt suurem kui arv $1-2^{-39}$, tekib nn. *ületäitumine*.

Liikuva komaga arvu mantiss q (koos märgiga) on 33-kohaline, järk p (koos märgiga) 7-kohaline. Seega kõigi normaliseeritud liikuva komaga arvude mantissid peavad rahuldama tingimust

$$0{,}5 \leqslant |q| \leqslant 1 - 2^{-32},$$

järgud aga tingimust

$$0 \leqslant |p| \leqslant 63.$$

Kui arvutamisel saadakse tulemuseks liikuva komaga arv, mille järk p on suurem arvust 63 (kaheksandsüsteemis 77), siis

tekib ületäitumine. Normaliseeritud liikuva komaga arvud, mille järk p on väiksem kui kaheksandarv —77, kujutavad arvutis nulle (arvuti nullid).

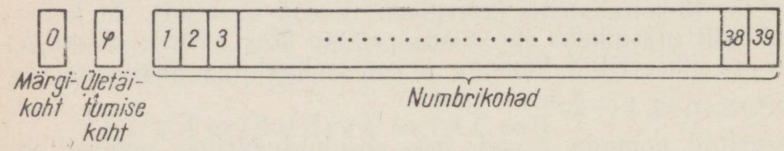
Arvutisse salvestatakse arvud kas kahend- või kümnend-kahendsüsteemis. Programmeerimisel (samuti tulemuste väljatrükkimisel arvutist) kirjutatakse kõigi kahendarvude asemel blankettidele vastavad kaheksandarvud, kümnend-kahendarvude asemel aga vastavad kümnendarvud. Üleminek kaheksandarvudelt kahendarvudele ja kümnendarvudelt kümnend-kahendsüsteemi arvudele toimub nende kandmisel perfokaartidele automaatselt (s. t. ilma lisatööta).

Arvutisse «Ural-4» on võimalik salvestada (samuti sealt väljastada) ka tähelis-numbrilist informatsiooni (teksti).

2. Informatsiooni töötlemine toimub arvuti aritmeetilises seadmes, mille põhiosa moodustavad sisendregister, summaator ja väljundregister. Operatsioonide tulemused saadakse summaatorisse (otsekoodis).

Summaator koosneb tegelikult kahest osast: summaatorist S ja nn. järgusummaatorist S_j. Summaatorites S ja S_j sooritatakse operatsioonid modifitseeritud pöördkoodis (vt. § 7).

Summaator S on ette nähtud teheteks fikseeritud komaga arvudega ja liikuva komaga arvude mantissidega. Ta sisaldab 41 kahendkohta: märgikoht, ületäitumise koht ning 39 kohta arvu kahendnumbrite salvestamiseks. Summaatori S kahendkohad on sobiv nummerdada (kümnendsüsteemis) vastavalt joonisel 16 näidatud skeemile.



JOON. 16

Summaatoris S saab sooritada tehteid arvudega x, mis rahuldavad tingimust:

$$|x| \leqslant 1 - 2^{-39},$$

ja fikseerida arve

$$|x| \leqslant 2 - 2^{-39}.$$

Järgusummaator S_j on määratud teheteks liikuva komaga arvude järkudega. Ta sisaldab 8 kahendkohta: märgikoht, ületäitumise koht ja kuus kohta liikuva komaga arvu järgu numbrite salvestamiseks. Seega järgusummaatoris S_j saab sooritada tehteid ainult selliste järkudega p, mis rahuldavad tingimust:

$$|p| \leqslant 63,$$

ja fikseerida järke

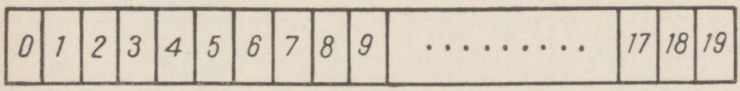
$$|p| \leqslant 127.$$

Kui mingi operatsiooni tulemusena saadakse arv, mille mantiss võrdub nulliga, siis kustutatakse (s. t. muudetakse nulliks) automaatselt ka järgusummaatoris olev arv (tulemus loetakse nulliks).

Ületäitumise korral töötatakse välja ja antakse juhtimisseadmesse signaal $\varphi = 1$. Kui aga operatsiooni sooritamisel ületäitumist ei toimunud, siis $\varphi = 0$. Olgu märgitud, et summaatorite S ja S_j ületäitumiskohtade sisu pole võimalik salvestada mäluseadmetesse.

Arvutil «Ural-4» on muutuva kestusega töötakt, mille pikkus sõltub sooritatavast operatsioonist ja sellest osa võtvatest arvudest (nn. operandidest). Näiteks operatsioonid fikseeritud koma režiimis võtavad vähem aega kui samad operatsioonid liikuva koma režiimis, liitmine ja lahutamine toimuvad kiiremini kui korrutamine ja jagamine. Arvuti sooritab sekundis keskmiselt 5000—6000 operatsiooni liikuva komaga arvudega või 9000—10 000 operatsiooni fikseeritud komaga arvudega.

3. Arvuti «Ural-4» operatiivmälus (ferriitkuup) on 4096 kahekümnest kahendkohast koosnevat pesa, mille aadressideks on neljakohalised kaheksandarvud (alates arvust 0000 kuni arvuni 7777, kusjuures pesa aadressi järele kirjutatakse veel eriline ühekohaline tunnus $\beta = 0$, näiteks 2563 0). Pesa kahendkohad on sobiv nummerdada alates 0-st kuni 19-ni vastavalt joonisel 17 toodud skeemile.



JOON. 17

Pesadesse saab salvestada fikseeritud komaga kaheksand- ja kümnendarve, käske ning tähelis-numbrilist informatsiooni. Kahendarvu (samuti käsu) kujutamisel jaotatakse pesa kahendkohad (alates esimesest) kolmekaupa gruppideks, mida nimetatakse kolmikuteks. Ühes pesas on seega 6 kolmikut, millest igaühes saab kujutada üht kaheksandnumbrit. Pesa 0-ndal kohal säilitatakse arvu (käsu) märki ja 19-ndal kohal kas kaheksandnumbrit 0 või 4 (käsu korral paikneb seal tunnus β). Pesadesse saab järelikult salvestada kaheksandarve x, mis rahuldavad tingimust:

$$2^{-23} \leqslant |x| \leqslant 0{,}7777774,$$

ja peale selle veel muidugi arvu null.

Kümnendarvud kirjutatakse operatiivmälu pesadesse kümnend-kahendsüsteemis, kusjuures pesa kahendkohad jaotatakse (alates esimesest) neljakaupa gruppideks — n e l i k u t e k s. Igas nelikus saab kujutada üht kümnendnumbrit. Et viies grupp (kohad 17—19) pole täielik, siis saab selles salvestada ainult paarisnumbreid. Arvu märk kujutatakse pesa 0-ndal kohal. Pesadesse saab seega salvestada kümnendarve x, mis rahuldavad tingimust:

$$2 \cdot 10^{-5} \leqslant |x| \leqslant 0{,}99998,$$

ja arvu null.

T ä h e l i s - n u m b r i l i n e i n f o r m a t s i o o n paigutatakse pessa järgmiselt: kohtadele 1—6 esimese, kohtadele 7—12 teise ja kohtadele 13—18 kolmanda sümboli kahendkood, kusjuures pesa kohad 0 ja 19 jäävad kasutamata. Sümbolite kahend- ning kaheksandkoodid on toodud tabelis 10. Peale selle on tabelis 9 esitatud veel sümbolite kaheksandkoodid praktiliseks kasutamiseks veidi kompaktsemal kujul: koodi esimese numbri saab rea, teise aga veeru nimest.

T a b e l 9

	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	8	9	+	—	А	Б	В	Г
2	Д	Е	Ж	З	И	Й	К	Л
3	М	Н	О	П	Р	С	Т	У
4	Ф	Х	Ц	Ч	Ш	Щ	Ы	Ь
5	Э	Ю	Я	?	.	,	:	;
6	x	/	=	<	>	∧	∨	(
7	)	[	]	↑	↓	↦	*	

Arvuti «Ural-4» tähelis-numbrilise informatsiooni sümbolite kuuekohalised koodid.

Sümbol	Kood	
	Kahendsüsteemis	Kaheksandsüsteemis
0	000 000	00
1	000 001	01
2	000 010	02
3	000 011	03
4	000 100	04
5	000 101	05
6	000 110	06
7	000 111	07
8	001 000	10
9	001 001	11
+	001 010	12
—	001 011	13
А	001 100	14
Б	001 101	15
В	001 110	16
Г	001 111	17
Д	010 000	20
Е	010 001	21
Ж	010 010	22
З	010 011	23
И	010 100	24
Й	010 101	25
К	010 110	26
Л	010 111	27
М	011 000	30
Н	011 001	31
О	011 010	32
П	011 011	33
Р	011 100	34
С	011 101	35
Т	011 110	36
У	011 111	37

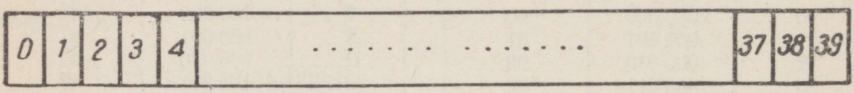
Sümbol	Kood	
	Kahendsüsteemis	Kaheksandsüsteemis
Ф	100 000	40
Х	100 001	41
Ц	100 010	42
Ч	100 011	43
Ш	100 100	44
Щ	100 101	45
Ы	100 110	46
Ь	100 111	47
Э	101 000	50
Ю	101 001	51
Я	101 010	52
?	101 011	53
.	101 100	54
,	101 101	55
:	101 110	56
;	101 111	57
×	110 000	60
/	110 001	61
=	110 010	62
<	110 011	63
>	110 100	64
∧	110 101	65
∨	110 110	66
(	110 111	67
)	111 000	70
[	111 001	71
]	111 010	72
↑	111 011	73
↓	111 100	74
→	111 101	75
*	111 110	76
tühik	111 111	77

4. Kaks järjestikuste aadressidega pesa (paarisarvulise aadressiga ja sellele järgneva paaritu aadressiga pesa) moodustavad ühe neljakümnest kahendkohast koosneva topeltpesa.[14] Topeltpesa aadressina kasutatakse temasse kuuluva paarisarvulise aadressiga pesa aadressi, kusjuures selle järele kirjutatakse tunnusena $\beta = 4$. Näiteks pesadest 2632 0 ja 2633 0 koosneva

[14] Topeltpesa asemel on kasutusel ka termin pikk pesa, kusjuures pesa nimetatakse siis lühikeseks pesaks.

topeltpesa aadress on 2632 4. Seega on topeltpesade aadressid järgmised: 0000 4, 0002 4, 0004 4, ..., 7776 4.

Topeltpesa kahendkohti on otstarbekohane nummerdada alates 0-st kuni 39-ni vastavalt joonisel 18 toodud skeemile. Topeltpesadesse saab salvestada nii liikuva ja fikseeritud komaga kaheksand- ning kümnendarve kui ka tähelis-numbrilist informatsiooni.



JOON. 18

Fikseeritud komaga kaheksandarvude kujutamisel topeltpesas jaotatakse selle kahendkohad (alates esimesest) kolmikuteks — kokku seega 13 kolmikut, kusjuures igasse kolmikusse salvestatakse üks kaheksandnumber. Arvu märk kujutatakse ikka 0-ndal kohal. Topeltpesas saab seega kujutada fikseeritud komaga kaheksandarve x, mis rahuldavad tingimust:

$$2^{-47} \leqslant |x| \leqslant 0{,}777\,777\,777\,777\,7$$

ja arvu null.

Fikseeritud komaga kümnendarvude kujutamisel topeltpesas jaotatakse kohad 1—32 neljakaupa gruppideks (kokku 8 nelikut), kusjuures igas nelikus kujutatakse ühe kümnendnumbri kümnendkahendkoodi. Arvu märk kantakse ikka 0-ndale kohale, kuid tehnilistel põhjustel ei kasutata nüüd kohti 33 kuni 39. Topeltpesasse saab salvestada fikseeritud komaga kümnendarve x, mis rahuldavad tingimust

$$10^{-8} \leqslant |x| \leqslant 999\,999\,99,$$

ja muidugi veel arvu null.

Liikuva komaga kaheksandarvu mantiss (koos märgiga) kirjutatakse topeltpesa kohtadele 0—32, järk (koos märgiga) aga sama topeltpesa kohtadele 33—39. Mantissi märk paikneb sealjuures kohal 0, järgu märk aga kohal 33. Topeltpesasse on võimalik seega salvestada liikuva komaga kaheksandarve x, mis rahuldavad tingimust

$$0{,}4 \cdot 2^{-77} \leqslant |x| \leqslant (1 - 2^{-40}) \cdot 2^{77},$$

ja arvu null.

Liikuva komaga kümnendarvud kirjutatakse topeltpesasse kümnend-kahendsüsteemis, kusjuures kahendkohad jaotatakse järgmiselt: 0-ndal kohal kujutatakse arvu mantissi märk, kohtadel 1—32 mantissi numbrid, kohal 33 arvu järgu märk ning kohtadel 35—39 arvu järgu numbrikohad. Topeltpesa 34. kohta liikuva komaga kümnendarvude kujutamisel ei kasutata. Topelt-

pesasse on võimalik salvestada liikuva komaga kümnendarve $x = q \cdot 10^p$, mis rahuldavad tingimusi:

$$0{,}1 \leqslant |q| \leqslant 1 - 10^{-8} \qquad 0 \leqslant |p| \leqslant 19$$

ja arvu null.

Ühte topeltpesasse saab salvestada kuni kuue sümboli kahendkoodid, mis paigutatakse järjest topeltpesa kohtadele 1—36. Kohad 0 ja 37—39 jäävad sealjuures kasutamata.

Olgu märgitud, et operatiivmälu pesast informatsiooni lugemiseks kulub arvutil 15 μsek, topeltpesast lugemiseks aga 30 μsek.

5. Peale operatiivmälu on arvutil «Ural-4» veel kaks automaatset salvestusseadet — salvestusseadmed magnettrumlitel ja magnetlintidel (nn. välised mälud).

Salvestusseade magnettrumlitel koosneb vastavast juhtimisseadmest ja kuni kaheksast magnettrumlist. Igale trumlile on võimalik salvestada 32 768 kahekümnekohalist kahendarvu või käsku. Pesad magnettrumlitel on nummerdatud tabelis 11 näidatud viisil.

Tabel 11

Trumli number	Pesade aadressid (kuuekohalised kaheksandarvud)
0	000 000 kuni 077 777
1	100 000 kuni 177 777
2	200 000 kuni 277 777
3	300 000 kuni 377 777
4	400 000 kuni 477 777
5	500 000 kuni 577 777
6	600 000 kuni 677 777
7	700 000 kuni 777 777

Informatsioonivahetus operatiivmälu ja magnettrumlite vahel toimub ainult pesade (20 kahendkohta) kaupa, kusjuures sekundis on võimalik magnettrumlitele kirjutada (või sealt operatiivmällu salvestada) kuni 9000 arvu.

Salvestusseade magnetlintidel koosneb juhtimisseadmest ja kuni 12-st magnetlindi kapist. Iga kapp sisaldab sealjuures neli linti, igal lindil paikneb aga kuni 32 tsooni, igaühes 4096 pesa. Seega saab magnetlintidele kirjutada kokku kuni 6 291 456 kahekümnekohalist kahendarvu.

Magnetlindi pesade aadressid on kaheksakohalised kaheksandarvud 0000 0000 kuni 2777 7777. Aadressi neli esimest kaheksandkohta moodustavad sealjuures nn. tsooninumbri *c*, neli vii-

Tabel 12

Kapi number	Lindil nr. 0 on tsoonide numbrid *c*	Lindil nr. 1 on tsoonide numbrid *c*	Lindil nr. 2 on tsoonide numbrid *c*	Lindil nr. 3 on tsoonide numbrid c
0	0000—0037	0040—0077	0100—0137	0140—0177
1	0200—0237	0240—0277	0300—0337	0340—0377
2	0400—0437	0440—0477	0500—0537	0540—0577
3	0600—0637	0640—0677	0700—0737	0740—0777
— —	— —	— —	— —	— —
13	2600—2637	2640—2677	2700—2737	2740—2777

mast kaheksandkohta aga pesa aadressi tsoonis (tähistame a_3). Igas tsoonis on pesad nummerdatud 0000-st kuni 7777-ni. Tsoonide numbrid *c* magnetlintidel on toodud tabelis 12. Magnetlindi tsooni numbri *c* arvutamiseks võib aga kasutada ka valemit (kaheksandsüsteemis)

$$c = n_1 \cdot 200 + n_2 \cdot 40 + n_3,$$

kus n_1 on kapi number, n_2 lindi number kapis ja n_3 tsooni järjekorranumber lindil (lindil on tsoonide järjekorranumbrid 0, 1, 2, . . . , 37).

Magnetlintide juhtimisseadmesse kuulub veel spetsiaalne valikublokk, mis võimaldab magnetlintidelt informatsiooni operatiivmällu kirjutada teatud tunnuste järgi valimise teel (nn. valikuoperatsioonid). Selles blokis on järgmised registrid:

1) neljakümnekohaline tunnuseregister;
2) kolmeteistkümnekohaline sammuregister;
3) neljakümnekohaline skaalaregister ja
4) kahekümne kolme kohaline võrreldavate arvude hulga loendaja.

Informatsioonivahetus magnetlintide ja operatiivmälu vahel (välja arvatud valikuoperatsioonid) toimub ainult pesade kaupa, kusjuures sekundis kirjutatakse magnetlintidele (või loetakse sealt) 4000 kahekümnekohalist kahendarvu (tsooni otsimiseks kuluvat aega pole siin arvestatud). Valikuoperatsioonid toimuvad seevastu topeltpesadesse salvestatud arvudega, kusjuures sekundis vaadatakse läbi 2000 neljakümnekohalist kahendarvu.

6. Arvuti «Ural-4» sisendkommutaator on ehitatud 64 kahekümnekohalise sisendseadme jaoks. Sisendseadmetena võib sealjuures kasutada perfokaartidelt ja telegraafilintidelt

lugemise seadmeid, sideliine, juhuslike arvude andureid jne. Arvuti normaalkomplekti kuulub kaks 80-veerulistelt perfokaartidelt lugemise seadet (kus informatsiooni lugemine toimub ridade kaupa kaardi neljakümnest veerust) ja kaks telegraafilindilt lugemise seadet. Viimaste vahendusel telegraafilindilt arvutisse salvestatav informatsioon teisendatakse automaatselt II rahvusvahelisest telegraafikoodist arvuti «Ural-4» trükkimisseadme kuuekohalisesse koodi (vt. tabel 13).

Perfokaartidelt informatsiooni lugemise kiirus on 100 kuni 300 kaarti minutis, telegraafilindilt lugemise kiirus aga 400 tähelis-numbrilise informatsiooni sümbolit sekundis.

V ä l j u n d k o m m u t a a t o r on ehitatud 64 kahekümnekohalise väljundseadme jaoks. Arvuti normaalkomplekti kuuluvad: kaks kaartide väljundperforaatorit, kaks telegraafilindi väljundperforaatorit, kiirtrükkija, millega saab väljastada nii kaheksandkui ka kümnendarve ja seade tähelis-numbrilise informatsiooni trükkimiseks.

Kaartide väljundperforaatori töötamise kiirus on 10 rida sekundis (50 kaarti minutis), telegraafilindi väljundperforaatoril aga 7—8 tähelis-numbrilise informatsiooni sümbolit sekundis.

Tähelis-numbrilise informatsiooni trükkimisseadme abil saab trükkida tulemusi tähelis-numbrilise teksti, tabelite ja graafikute kujul 512 mm laiusele paberilindile. Trükkimise kiiruseks on kuni 300 rida minutis, kusjuures ühte ritta saab paigutada kuni 128 märki. Erinevaid trükimärke on kokku 63 (vt. tabel 9 või 10).

Tähelis-numbrilise informatsiooni trükkimisseadmel on spetsiaalne register, mis sisaldab 128 kuuekohaliste kahendarvude (sümbolite koodide) salvestamiseks ette nähtud p o s i t s i o o n i. Need positsioonid kannavad järjekorranumbreid 0 kuni 177 (kaheksandarvud). Rea trükkimiseks tuleb väljastatav informatsioon kõigepealt kirjutada trükkimisseadme registrisse.

7. J u h t i m i s s e a d e tagab arvuti kõigi seadmete automaatse koostöö ülesande lahendamise käigus. Juhtimisseadmes on järgmised kolm registrit: 12-kohaline k ä s k u d e l o e n d a j a (selles registris näidatud aadressiga pesast võetav käsk täidetakse järgmisel töötaktil), 20-kohaline k ä s k u d e r e g i s t e r (sellesse registrisse kirjutatud käsk täidetakse algaval töötaktil) ja 13-kohaline t s ü k l i r e g i s t e r.

Arvuti «Ural-4» võib programmi täita nii automaatses kui ka nn. ühetaktilises režiimis. Tema tööd on tarbekorral võimalik käsitsi juhtida j u h t i m i s p u l d i l t, kus paiknevad vastavad võtmed (kahepositsioonilised lülitid) ja nupud (ühepositsioonilised lülitid). Võtmed on püsivalt kas välja lülitatud (*alumises asendis*) või sisse lülitatud (*ülemises asendis*), juhtimine nuppude kaudu toimub aga neile vajutamisega, mis annab arvutile ühekordse korralduse.

Arvuti «Ural-4» trükkimisseadme kuuekohaliste koodide ja telegraafilindi viiekohaliste koodide vastavuse tabel

Kuuekohalised koondid		Viiekohalised telegraafilindi koodid		
Sümbol	Kahendkood	Kahendkood	Register	Sümbol
0	000000	01101	arvu	0
1	000001	11101	„	1
2	000010	11001	„	2
3	000011	10000	„	3
4	000100	01010	„	4
5	000101	00001	„	5
6	000110	10101	„	6
7	000111	11100	„	7
8	001000	01100	„	8
9	001001	00011	„	9
+	001010	10001	„	+
—	001011	11000	„	—
А	001100	11000	vene	А
Б	001101	10011	„	Б
В	001110	11001	„	В
Г	001111	01011	„	Г
Д	010000	10010	„	Д
Д	010000	10010	ladina	D
Е	010001	10000	vene	Е
Ж	010010	01111	„	Ж
Ж	010010	01001	ladina	L
З	010011	10001	vene	З
З	010011	00110	ladina	N
И	010100	01100	vene	И
И	010100	01100	ladina	I
Й	010101	11010	vene	Й
Й	010101	11010	ladina	J
К	010110	11110	vene	К
Л	010111	01001	„	Л
М	011000	00111	„	М
Н	011001	00110	„	Н
О	011010	00011	„	О
П	011011	01101	„	П
Р	011100	01010	„	Р
С	011101	10100	„	С
Т	011110	00001	„	Т
У	011111	11100	„	У
У	011111	10101	ladina	У
Ф	100000	10110	vene	Ф
Ф	100000	10110	ladina	F
Х	100001	00101	vene	Х
Х	100001	10111	ladina	Х
Ц	100010	01110	vene	Ц
Ц	100010	11001	ladina	W
Ч	100011	00001	„	Т
Ч	100011	01011	„	G
Ш	100100	01011	arvu	Ш
Ш	100100	10001	ladina	Z
Щ	100101	00101	arvu	Щ
Щ	100101	01010	ladina	R
Ы	100110	10101	vene	Ы
Ы	100110	10100	ladina	S
Ь	100111	10111	vene	Ь
Э	101000	10110	arvu	Э
Э	101000	11100	ladina	U
Ю	101001	11010	arvu	Ю
Ю	101001	11101	ladina	Q
Я	101010	11101	vene	Я
?	101011	10011	arvu	?
.	101100	00111	„	.
,	101101	00110	„	,
:	101110	01110	„	:
;	101111	10100	„	;
×	110000	10010	„	X
/	110001	10111	„	/
=	110010	01111	„	=
<	110011	11000	ladina	A
>	110100	10011	„	B
∧	110101	10000	„	E
∨	110110	01111	„	V
(	110111	11110	arvu	(
)	111000	01001	„	)
[	111001	11110	ladina	K
]	111010	00111	ladina	M
↑	111011	00101	„	H
↓	111100	00011	„	O
→	111101	01101	„	P
*	111110	01110	„	C
tühik	111111	00100	tühik	

Juhtimispuldil olevate lülitite täielikku loetelu pole siinkohal mõtet esitada, sest seda peab tundma vaid arvutil töötav operaator, mitte tingimata aga programmeerija.[15] Seetõttu piirdume vaid nende lülititega, mille kasutamist programmeerimisel, eriti aga programmide nn. silumisel arvestada tuleb. Et arvuti juhtimispuldile on lülitite nimed kirjutatud vene keeles, siis anname järgnevas ka venekeelsed nimetused.

Nupule *käivitus* (*Пуск*) vajutamise järel hakkab arvuti automaatses režiimis täitma operatiivmällu juba varem kirjutatud programmi, kusjuures esimesena täidetakse käskude registris paiknev käsk. Kui nüüd vajutada nupule *peatus* (*Остановка*), siis arvuti lõpetab programmi käskude täitmise ja peatub. Pärast teistkordset nupule *käivitus* vajutamist täidab arvuti jälle programmi edasi sealt, kus see peatumisel katkestati. Niisugust töö uuestialustamist ei saa aga kasutada magnetlindi või -trumli poole pöördumise, samuti telegraafilindilt või perfokaartidelt lugemise ajal.

Nupule *ühetaktiline režiim* (*Однотактный режим*) vajutamise tulemusena arvuti täidab vaid selle käsu, mis parajasti paikneb käskude registris, kusjuures käskude registrisse kirjutatakse ühtlasi järgmine käsk. Ühetaktilises režiimis ei saa teostada magnetlindi ja -trumli poole pöördumise ning telegraafilindilt ja perfokaartidelt lugemise operatsioone.

Nupp *algkäivitus* (*Начальный пуск*) on määratud programmi viimiseks arvutisse uue ülesande lahendamise alustamisel. Sellele nupule vajutamise tulemusena salvestab arvuti sisendseadmel paikneva kaardipaki esimese kaardi sisu operatiivmälu topeltpesadesse 0004 4 kuni 0032 4 ja asub seejärel täitma pesasse 0004 0 salvestatud käsku. Märgime, et enne algkäivitust peavad olema kustutatud käskude registris, käskude loendajas ning mälu esimeses neljas pesas olevad arvud.

Võtmed *φ blokeeritud* (*Блокировка φ*) ja *peatus φ korral* (*Остановка по φ*) on määratud mitmesuguste võimalike variantide valimiseks ületäitumise korral.

Võtmete *peatus aadressi järgi* (*Остановка по адресу*) abil saab valida selle pesa aadressi, milles paikneva käsu täitmise eel soovitakse arvutit peatada. Neid võtmeid kasutatakse peaaegu eranditult vaid programmide kontrollimisel, kui on tarvis programmi teatud kohas veenduda vahetulemuste õigsuses vms.

Seitse võtit suunamiseks (*Передачи управления*) leiavad kasutamist järgmises paragrahvis kirjeldatava erilise suunamiskäsu — koodiga 23 — puhul.

---

[15] Lülitite kasutamise täielikud juhendid võib leida näiteks väljaandes: Elektronarvuti «Ural-4», TRÜ rotaprint, 1967.

Peale mitmesuguste lülitite on juhtimispuldil veel klaviatuur käskude vahetuks kandmiseks käskude registrisse ja arvude kandmiseks summaatorisse.

Juhtimispuldi juurde kuulub ka signalisatsioonipaneel, mis võimaldab operaatoril visuaalselt jälgida ja kontrollida arvuti tööd. Sellele paneelile on paigutatud signaallambid, mille järgi saab lugeda summaatorites S ja S_j, käskude registris, käskude loendajas, tsükliregistris, käskude dešifraatoris ja operatiivmälu väljundregistris olevaid arve, näha aritmeetilisest seadmest juhtimisseadmesse antud signaalide φ ja ω väärtusi, arvuti peatumise põhjust või töörežiimi. Registrites olevad arvud kantakse signalisatsioonipaneelile otsekoodis: põlev lamp esitab numbrit 1, kustunud lamp numbrit 0 (erandiks on vaid tsükliregistris olev arv, mis tsüklioperatsiooni sooritamise ajal kantakse paneelile pöördkoodis).

§ 10. ARVUTI «URAL-4» KÄSKUDE SÜSTEEM

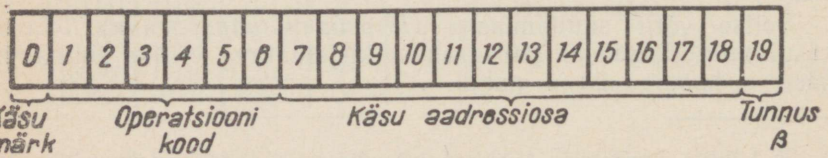
1. Arvuti «Ural-4» käskude süsteem sisaldab kokku 56 käsku. Iga käsu toimel sooritab arvuti teatud kindla, konstruktsiooniga ette määratud operatsiooni. Arvestades vastavate operatsioonide iseloomu, võib käsud jaotada järgmistesse rühmadesse:

aritmeetiliste tehete käsud;
loogiliste tehete käsud;
salvestamiskäsud;
suunamiskäsud;
käsud tulemuste väljastamiseks;
käsud väliste salvestusseadmete poole pöördumiseks.

Iga käsk salvestatakse üldiselt operatiivmälu ühes pesas — erandi selles suhtes moodustavad ainult mõned väliste salvestusseadmete poole pöördumise käsud, mis operatiivmälus võtavad enda alla 3—4 järjestikuste aadressidega pesa. Vahe tegemiseks nimetatakse viimaseid ka liitkäskudeks.

Oma tavalisel kujul on käsk (s. t. mitte liitkäsk) seega üks 20-kohaline kahendarv, mida aga tõlgendatakse koosnevana neljast osast, nii nagu on näidatud joonisel 19.

Programmeerimisel kantakse käsud blankettidele märgiga varustatud seitsmekohaliste kaheksandarvudena, mille kaks vasak-



JOON. 19

poolset kaheksandkohta esitavad operatsiooni koodi, neli järgmist kaheksandkohta annavad käsu aadressiosa, seitsmes kaheksandkoht (number 0 või 4) aga tunnuse β väärtuse. Käsu koostisosade visuaalseks eristamiseks jäetakse nende vahele kirjutamisel tavaliselt veidi suurem vahe; pluss märk aga jäetakse käsu ees enamasti kirjutamata. Seega tuleb käsud kirjutada näiteks kujul

— 07 3205 0 või 25 0216 4.

Miinusmärk käsu ees on tunnuseks, et teda täidetakse n. ö. modifitseeritud kujul. Nimelt, käskude registrisse kandmisel lahutatakse kõigist miinusmärgiga varustatud käskudest tsükliregistris olev arv (täpsemalt, käsu otsekoodile liidetakse tsükliregistris oleva arvu pöördkood). Käsu ülejäänud osade tähendused selguvad lehekülgedel 88—100 esitatud käskude tabelist (vt. tabel 14) ja sellele lisatud selgitustest. Käskudetabeli koostamisel on kasutatud järgmisi tähistusi:

$(S)_0$ — summaatoris S olev arv enne operatsiooni sooritamist;
$(S_j)_0$ — summaatoris S_j olev arv enne operatsiooni sooritamist;
(S) — pärast operatsiooni sooritamist summaatoris S olev arv;
(S_j) — pärast operatsiooni sooritamist summaatoris S_j olev arv;
q — operatsiooni tulemuse mantiss;
p — operatsiooni tulemuse järk;
$(a\beta)$ — pesas, mille aadress on $a\beta$, paiknev arv.

Vaatleme nüüd mõningaid tabelis 14 toodud käske veidi üksikasjalikumalt.

2. Aritmeetilised tehted (operatsiooni koodid 01, 41; 03, 43; 26, 44; 06, 46; 07, 47) sooritatakse kõik arvudega, millest üks paikneb summaatoris, teine aga operatiivmälu pesas, mille aadress on näidatud vastava käsu aadressiosas. Tehete tulemus saadakse alati summaatorisse ja seda ei ümardata.

Fikseeritud koma režiimis sooritatavate aritmeetiliste tehete (koodid 01, 03, 06, 07, 26) puhul interpreteeritakse summaatori S ja pesa $a\beta$ sisu fikseeritud koma arvudena (kahendmurdudena). Kui tehet realiseerivas käsus tunnus β võrdub nulliga, siis sooritatakse:

a) liitmine, lahutamine ja kontroll-liitmine (koodid 01, 03 ja 26) summaatori S kohtadel 0 kuni 19 ja pesas a 0 paiknevate arvudega (tulemus [16] jääb summaatori S kohtadele 0 kuni 19);

[16] Kontroll-liitmise tulemus jääb summaatori kohtadele 0 kuni 39.

Tabel 14

Operatsiooni kood	Operatsiooni nimetus	Käsu kuju	Käsu täitmise tulemus				Selgitused
			Summaatorisse saadakse		Tingimus, mille täidetuse korral		
			S	S_j	$\omega = 1$	$\varphi = 1$	
1	2	3	4	5	6	7	8
Aritmeetilised tehted fikseeritud koma režiimis							
01	Liitmine	$01a\beta$	$(S)_0 + (a\beta)$	0	$(S) < 0$	$\lvert(S)\rvert \geqslant 1$	Summaatoris S olevale arvule liidetakse (algebraliselt) pesas $a\beta$ olev arv
03	Lahutamine	$03a\beta$	$(S)_0 - (a\beta)$	0	$(S) < 0$	$\lvert(S)\rvert \geqslant 1$	Summaatoris S olevast arvust lahutatakse (algebraliselt) pesas $a\beta$ olev arv
06	Korrutamine	$06a\beta$	$(S_0) \cdot (a\beta)$	0	$(S) < 0$	—	Summaatoris S olev arv korrutatakse pesas $a\beta$ oleva arvuga
07	Jagamine	$07a\beta$	$(S)_0 : (a\beta)$	0	$(S) < 0$	$\lvert(S)\rvert \geqslant 1$	Summaatoris S olev arv jagatakse pesas $a\beta$ oleva arvuga
Aritmeetilised tehted liikuva koma režiimis							
41	Liitmine	$41a\beta$	q	p	$(S) < 0$	$(S_j) > 77$	Summaatoris olevale arvule liidetakse pesas $a\beta$ olev arv. Summa saadakse normaliseeritud mantissiga

43	Lahutamine	43$\alpha\beta$	q	p	$(S) < 0$	$(S_j) > 77$	Summaatoris olevast arvust lahutatakse pesas $\alpha\beta$ olev arv. Vahe saadakse normaliseeritud kujul
44	Absoluutväärväärtuste lahutamine	44$\alpha\beta$	q	p	$(S) < 0$	—	Summaatoris oleva arvu absoluutväärtusest lahutatakse pesas $\alpha\beta$ oleva arvu absoluutväärtus. Vahe saadakse normaliseeritud kujul
46	Korrutamine	46$\alpha\beta$	q	p	$(S) < 0$	$(S_j) > 77$	Summaatoris olev arv korrutatakse pesas $\alpha\beta$ oleva arvuga. Korrutis on normaliseeritud (ümardamist ei toimu)
47	Jagamine	47$\alpha\beta$	q	p	$(S) < 0$	$(S_j) > 77$	Summaatoris olev arv jagatakse pesas $\alpha\beta$ oleva arvuga. Jagatis on normaliseeritud (ümardamist ei toimu)
Operatsioonid fikseeritud komaga arvude teisendamiseks							
35	Kümnendarvu teisendamine kahendarvuks	35$\alpha\beta$	Arv ($\alpha\beta$) kahendsüsteemis	$(S_j)_0$	$(S) < 0$	—	Pesas $\alpha\beta$ olev fikseeritud koma arv teisendatakse kümnend-kahendsüsteemist kahendsüsteemi (pesas $\alpha\beta$ säilib arvu esialgne kuju)
36	Kahendarvu teisendamine kümnendarvuks	36 0000 0	Arv kümnend-kahendsüsteemis	$(S_j)_0$	Säilib	—	Summaatoris S olev arv teisendatakse kahendsüsteemist kümnend-kahendsüsteemi
Loogilised operatsioonid							
10	Märgi omistamine	10$\alpha\beta$	sign ($\alpha\beta$) · $\cdot (S)_0$	$(S_j)_0$	$(S) < 0$	—	Summaatoris S olevale arvule omistatakse pesas $\alpha\beta$ oleva arvu märk

1	2	3	4	5	6	7	8
11	Loogiline nihutamine	11n0	Nihutamise tulemus	0	$(S) < 0$	—	Summaatoris S olevat arvu (koos märgikohaga) nihutatakse n kahendkoha võrra kas vasakule või paremale. Kui $n > 0$, siis nihutatakse vasakule; kui $n < 0$, siis paremale
	Aritmeetiline nihutamine	11n4	Nihutamise tulemus	0	Säilib	$\|(S)\| \geqslant 1$	Summaatoris S olevat arvu (ilma märgikohata) nihutatakse n kahendkoha võrra kas vasakule või paremale. Kui $n > 0$, siis nihutatakse vasakule; kui $n < 0$, siis paremale
12	Osa eraldamine	12$a\beta$	Eraldatud osa	$(S_j)_0$	$(S) = 0$	—	Toimub summaatoris S oleva arvu kõigi kahendkohtade (kaasa arvatud märgikoht) loogiline korrutamine pesas $a\beta$ oleva arvu vastavate kahendkohtadega
13	Formeerimine	13$a\beta$	Formeerimise tulemus	$(S_j)_0$	$(S) = 0$	—	Toimub summaatoris S oleva arvu kõigi kahendkohtade (kaasa arvatud märgikoht) loogiline liitmine pesas $a\beta$ oleva arvu vastavate kahendkohtadega
14	Võrdlemine	14$a\beta$	Võrdlemise tulemus	$(S_j)_0$	$(S) \neq 0$	—	Toimub summaatoris S oleva arvu ja pesas $a\beta$ oleva arvu vastavate kahendkohtadega loogilise mittesamaväärsuse tehe

Operatsioonid arvude ümberkirjutamiseks							
02	Arvu saatmine summaatorisse (fikseeritud koma režiimis)	$02a\beta$	$(a\beta)$	0	$(S) < 0$	—	Kustutatakse summaatoris S ja järgusummaatoris olevad arvud ning kirjutatakse summaatorisse S pesas $a\beta$ olev arv (pesas $a\beta$ arv säilib)
16	Mälusse saatmine (fikseeritud koma režiimis)	$16a\beta$	$(S)_0$	$(S_j)_0$	$(S) < 0$	—	Summaatoris S olev arv $(S)_0$ kirjutatakse operatiivmälu pesasse $a\beta$
20	Aadressi saatmine summaatorisse	$20n\beta$	Kui $\beta = 0$, siis $n \cdot 2^{-18}$; kui $\beta = 4$, siis $-n \cdot 2^{-18}$	0	$(S) < 0$	—	Kui $\beta = 0$, siis saadetakse summaatorisse S arv $n \cdot 2^{-18}$. Kui $\beta = 4$, siis saadetakse summaatorisse arv $-n \cdot 2^{-18}$
27	Tsükliregistris oleva arvu salvestamine	$27a\beta$	$(S)_0$	$(S_j)_0$	Säilib	—	Pesa a kohtadele 0—19 formeeritakse käsk $25m\bar{\beta}$, kus sümboliga $m\bar{\beta}$ on tähistatud tsükliregistri sisu käsu 27 täitmise momendil (kui $\beta = 4$, siis kirjutatakse topeltpesa a 4 kohtadele 20—39 nullid)
40	Arvu saatmine sisendilt summaatorisse	$40m\beta$	Sisendilt loetud arv	0	$(S) < 0$	—	Sisendilt number m saadetakse arv summaatorisse S (fikseeritud komaga arvuna)
42	Arvu saatmine summaatorisse (liikuva koma režiimis)	$42a\beta$	Pesas $a\beta$ oleva arvu mantiss	Pesas $a\beta$ oleva arvu järk	$(S) < 0$	$(S_j) > -77$	Pesas $a\beta$ olev arv kirjutatakse summaatorisse, mantiss normaliseeritakse ja järgusummaatorisse saadetud järgus tehakse vastav parandus

1	2	3	4	5	6	7	8
56	Mällu saatmine (liikuva koma režiimis)	56$a\beta$	q	p	$(S) < 0$	$(S_j) > 77$	Summaatoris oleva arvu mantiss ümardatakse, summaatori S järkudesse 33—39 kirjutatakse nullid, arv normaliseeritakse ja kirjutatakse pesasse $a\beta$
67	Arvu saatmine tunnuseregistrisse	67 0000 0	$(S)_0$	$(S_j)_0$	Säilib	—	Arv $(S)_0$ saadetakse tunnuseregistrisse
70	Arvu saatmine sammuregistrisse	70 0000 0	$(S)_0$	$(S_j)_0$	Säilib	—	Summaatori S järkudes 27—39 olev arv kirjutatakse sammuregistrisse
71	Arvu saatmine skaalaregistrisse	71 0000 0	$(S)_0$	$(S_j)_0$	Säilib	—	Arv $(S)_0$ saadetakse skaalaregistrisse
72	Arvu saatmine võrreldavate arvude hulga loendajasse	72 0000 0	$(S)_0$	$(S_j)_0$	Säilib	—	Summaatori S järkudes 17—39 olev arv kirjutatakse võrreldavate arvude hulga loendajasse
73	Võrreldavate arvude hulga loendajast arvu saatmine summaatorisse	73 0000 0	Võrreldavate arvude hulga loendaja sisu	$(S_j)_0$	$(S) = 0$	—	Võrreldavate arvude hulga loendajas olev arv kirjutatakse summaatori S järkudesse 17—39

Suunamisoperatsioonid, tsüklioperatsioonid ja käsu muutmise operatsioon							
21	Suunamine signaali $\omega = 1$ järgi	21a0	$(S)_0$	$(S_j)_0$	Säilib	—	Kui $\omega = 1$, siis pesas $k+0$ paikneva käsu 21 a 0 täitmise tulemusena antakse juhtimine pesas a olevale käsule; kui aga $\omega = 0$, siis jätkub käskude täitmine loomulikus järjekorras (s. t. pesas $k+1$ oleva käsu täitmisega)
	Suunamine signaali $\omega = 0$ järgi	21a4	$(S)_0$	$(S_j)_0$	Säilib	—	Kui $\omega = 0$, siis antakse juhtimine pesas a olevale käsule; kui aga $\omega = 1$, siis jätkatakse käskude täitmist loomulikus järjekorras
22	Tingimusteta suunamine	22a0	$(S)_0$	$(S_j)_0$	Säilib	—	Juhtimine antakse pesas a olevale käsule
	Suunamine tagasipöördumisega.	22a4	$(S)_0$	$(S_j)_0$	Säilib	—	Pesas $k+0$ paikneva käsu 22 a 4 täitmise tulemusena formeeritakse pesasse a käsk 22 $k+1$ 0 ja antakse juhtimine pesas $a+1$ olevale käsule
23	Suunamine võtme järgi	23n0	$(S)_0$	$(S_j)_0$	Säilib	—	Kui võti nr. n on sisse lülitatud, siis pesas $k+0$ paikneva käsu 23 n 0 täitmisega antakse juhtimine pesas $k+2$ olevale käsule, vastupidisel juhul aga täidetakse pesas $k+1$ olev käsk

1	2	3	4	5	6	7	8
24	Tsükli lõpp	24a0	$(S)_0$	$(S_j)_0$	Säilib	—	Kui tsükliregistris on nullist erinev arv, siis antakse juhtimine pesas a olevale käsule, kusjuures tsükliregistris olevat arvu vähendatakse sammu h võrra (vastupidisel juhul jätkatakse käskude täitmist loomulikus järjekorras (tsükli registris olevat arvu ei muudeta)
25	Tsükli algus	25$m\beta$	$(S)_0$	$(S_j)_0$	Säilib	—	Arv $m\beta$ kirjutatakse tsükliregistrisse. Kui $\beta = 0$, siis käsu 24 a 0 täitmisel $h = 1$; kui $\beta = 4$, siis aga $h = 2$
30	Käsu muutmine	30a0	$(S)_0$	$(S_j)_0$	Säilib	—	Pesas $k+0$ paikneva käsu 30 a 0 täitmisel kirjutatakse pesas a olev arv pöördkoodis käskude registrisse ja liidetakse sellele (otsekoodis) pesas $k+1$ paiknev käsk (registri 1. kohal tekkiv ülekanne liidetakse 19. kohal olevale kahendkohale). Saadud käsk täidetakse pesas $k+1$ paikneva käsu asemel
		30a4	$(S)_0$	$(S_j)_0$	Säilib	—	Kui a on paarisarv, siis liidetakse (käskude registris) pesas $k+1$ paiknevale käsule pesas $a+1$ olev arv; kui aga a on paarituarv, siis pesas $a-1$ olev arv (modifitseerimine toimub täpselt samuti nagu käsu 30 a 0 korral)

Tulemuste väljastamine							
32	Kümnendarvu väljastamine	$32m\beta$	$(S)_0$	$(S_j)_0$	Säilib	—	Summaatoris S olev arv trükitakse või perforeeritakse kümnendsüsteemis (liikuva komaga arvuna) välja m-ndal väljundseadmel
	Kümnendarvu kandmine trükkimisseadme registrisse	$32\,4000 + \alpha \cdot 400 + n\beta$	$(S)_0$	$(S_j)_0$	Säilib	—	Summaatoris S olev arv kantakse (kui kahend-kümnendarv) trükkimisseadme registrisse. 1) Kui $\alpha = 1$, siis kirjutatakse registrisse ainult summaatori märgikoha sisu. 2) Kui $\alpha = 0$ ja $\beta = 0$, siis kirjutatakse registrisse summaatori kohtadel 0—16 olev arv. 3) Kui $\alpha = 0$ ja $\beta = 4$, siis kirjutatakse registrisse kogu summaatori sisu (kui liikuva komaga kümnend-kahendarv). Arv n käsu 12.—18. kohal näitab registri positsiooni numbrit, millest alates kirjutatakse väljastatavad koodid
33	Kaheksandarvu väljastamine	$33m\beta$	$(S)_0$	$(S_j)_0$	Säilib	—	Summaatoris S olev arv trükitakse või perforeeritakse kaheksandsüsteemis (fikseeritud komaga arvuna) välja m-ndal väljundseadmel

1	2	3	4	5	6	7	8
	Kaheksandarvu kandmine trükkimisseadme registrisse	33 4000 + $+\alpha \cdot$ $\cdot 400 + n\beta$	$(S)_0$	$(S_j)_0$	Säilib	—	Summaatoris S olev arv kantakse (kui kaheksandarv) trükkimisseadme registrisse 1) Kui $\alpha = 1$, siis kirjutatakse registrisse ainult summaatori märgikoha sisu. 2) Kui $\alpha = 0$ ja $\beta = 0$, siis kirjutatakse registrisse summaatori kohtadel 0—18 olev arv. 3) Kui $\alpha = 0$ ja $\beta = 4$, siis kirjutatakse registrisse kogu summaatori sisu. Arv n käsu 12.—18. kohal näitab registri positsiooninumbrit, millest alati kirjutatakse väljastatavad koodid
34	Reavahe trükkimine	34m0	$(S)_0$	$(S_j)_0$	Säilib	—	Kiirtrükkijal nihutatakse paberilinti ühe rea võrra edasi või perforeeritakse m-ndal väljundperforaatoril arv null (trükkimisseadme korral seda käsku ei kasutata)
60	Tähelis-numbrilise informatsiooni väljastamine trükkimisseadmele	60 4000 + $+\alpha \cdot$ $\cdot 400 + n\beta$	$(S)_0$	$(S_j)_0$	Säilib	—	Summaatoris S olev tähelis-numbriline informatsioon (sõna) kirjutatakse trükkimisseadme registrisse (alates registri positsioonist nr. n). 1) Kui $\alpha = 1$, siis kirjutatakse registrisse summaatori kohtadel 1—6 olev sümboli kood.

							2) Kui $\alpha = 0$ ja $\beta = 0$, siis kirjutatakse registrisse kolme esimese sümboli (s. o. summaatori kohtadelt 1—18) koodid. 3) Kui $\alpha = 0$ ja $\beta = 4$, siis kirjutatakse registrisse kõigi kuue sümboli koodid (summaatori kohad 1—36)
61	Rea trükkimine	61 4000 β	$(S)_0$	$(S_j)_0$	Säilib	—	Trükitakse trükkimisseadme registri sisu, kusjuures registri sisu kustutatakse. Kui $\beta = 4$, siis pärast trükkimist nihutatakse paberilinti ühe rea võrra edasi, $\beta = 0$ puhul aga paberit edasi ei nihutata. Kui käsk 61 4000 4 täitmise momendil registris informatsiooni ei ole, siis trükitakse reavahe
Magnettrumli poole pöördumine							
51	Magnettrumlilt lugemine	51$a_1$0 $a_3$0 00$a_2$0	Loetud arvude kontrollsumma	0	Säilib	—	Alates magnettrumli pesast a_3 kirjutatakse informatsioon magnettrumlilt järjest operatiivmälu pesadesse a_1, $a_1 + 1$, ..., ..., a_2
	Kontroll-lugemine	51$a_1$4 $a_3$0 00$a_2$0	Loetud arvude kontrollsumma	0	Säilib	—	Arvutatakse magnettrumli pesades a_3, $a_3 + 1$, ..., $a_3 + a_2 - a_1$ olevate arvude kontrollsumma (operatiivmälu pesadesse a_1, $a_1 + 1$, ..., a_2 midagi ei kirjutata)

1	2	3	4	5	6	7	8
52	Magnettrumlile kirjutamine	$52a_10$ a_30 $00a_20$	Kirjutatud arvude kontrollsumma	0	Säilib	—	Informatsioon operatiivmälu pesadest a_1 kuni a_2 kirjutatakse magnettrumlile alates pesast a_3
Magnetlindi poole pöördumine ja valikuoperatsioonid							
45	Magnetlindilt tsooni otsimine	$45c0$	—	—	—	—	Otsitakse magnetlindilt tsooni nr. c. Kuni programmis pole magnetlindi või magnettrumli poole pöördumise käsku, toimub tsooni otsimine paralleelselt programmis käsule $45c0$ järgnevate käskude täitmisega
55	Magnetlindilt lugemine	$45c0$ $55a_10$ $00a_30$ $00a_20$	Loetud arvude kontrollsumma	0	Lugemisel oli viga (operatsiooni teostamine katkestatakse)	—	Magnetlindi tsoonist nr. c (alates pesast a_3) kirjutatakse informatsioon operatiivmälu pesadesse a_1, a_1+1, ..., a_2. Kui tsoonis nr. c on vähem arve kui a_2-a_1+1, siis jätkatakse arvude kirjutamist tsoonist $c+1$
	Kontroll-lugemine magnetlindilt	$45c0$ $55a_14$ $00a_30$ $00a_20$	Loetud arvude kontrollsumma	0	Arvu lugemisel lindilt oli viga (operatsiooni teostamine katkestatakse)	—	Arvutatakse magnetlindi tsooni nr. c pesades a_3, a_3+1, ..., ..., $a_3+a_2-a_1$ olevate arvude kontrollsumma. Summeeritavaid arve operatiivmällu ei kirjutata

65	Esimene valikuoperatsioon	$45c0$ $65a_1 4$ $00a_3 0$ $00a_2 4$	Arv, mis näitab, mitu magnetlindilt läbivaadatud arvu rahuldas antud tingimust	0	Arvu lugemisel lindilt oli viga (operatsiooni teostamine katkestatakse)	—	Tunnuseregistris oleva arvu teatud kahendkohti, mis on määratud skaalaregistris oleva arvuga, võrreldakse järjest magnetlindi tsooni nr. c pesades a_3, $a_3 + h + 1$, jne. (kus h on sammuregistri kohtadel 1—12 olev arv) olevate arvude vastavate kahendkohtadega. Kui sammuregistri nullindal kohal on number 0, siis kirjutatakse arvud, mille võrreldavad kohad ühtivad tunnuseregistris oleva arvu vastavate kohtadega, järjest operatiivmälu pesadesse (alates pesast a_1), kui aga sammuregistri nullindal kohal on number üks, siis kirjutatakse iga tingimust rahuldanud arvuga koos operatiivmällu veel h temale järgnevates magnetlindi pesades olevat arvu. Operatsioon lõpetatakse, kui 1) operatiivmälu pesasse a_2 kirjutati arv; 2) magnetlindilt on läbi vaadatud $\frac{m+1}{h+1}$ arvu (m on võrreldavate arvude hulga loendajas olev konstant)

1	2	3	4	5	6	7	8
66	Teine valiku-operatsioon	45c0 66 0000 4 00$a_3$0 00n4	Arv, mis näitab, mitu magnetlindilt läbivaadatud arvu rahuldas antud tingimust	0	Arvu lugemisel lindilt oli viga (operatsiooni teostamine katkestatakse)	—	Toimub esimene valikuoperatsioon ilma nende arvude operatiivmällu kirjutamiseta, mille võrreldavad kohad ühtivad tunnuseregistris oleva arvu vastavate kohtadega. Operatsioon lõpetatakse, kui magnetlindilt on läbi vaadatud $\frac{m+1}{h+1}$ arvu
Kontroll-liitmine, peatus ja kell							
26	Kontroll-liitmine	26$a\beta$	Arvutatud kontroll-summa	0	Säilib	—	Summaatoris S olevale arvule liidetakse (otsekoodis) pesas $a\beta$ olev arv. Märgikohad liidetakse täpselt samuti nagu liidetavate arvude ülejäänud kahendkohad (märgikohal tekkinud ülekanne liidetakse summaatori 39-ndale kohale)
37	Peatus	37 0000 0	$(S)_0$	$(S_j)_0$	Säilib	—	Arvuti peatub (s. t. lõpetab programmi täitmise)
62	Kell	62 0000 0	$(S)_0$	$(S_j)_0$	Säilib	—	Selle käsu täitmise tulemusena hakkab helisema kell; arvuti aga jätkab programmi täitmist. Kellahelina katkestamiseks tuleb juhtimispuldil vajutada vastavale nupule

b) korrutamine (kood 06) summaatori kohtadel 0 kuni 20 ja pesas a 0 paiknevate arvudega (tulemus jääb summaatori S kohtadele 0 kuni 34);

c) jagamine (kood 07) summaatori kohtadel 0 kuni 39 ja pesas a 0 paiknevate arvudega (tulemus jääb summaatori S kohtadele 0 kuni 20).

Kui fikseeritud koma režiimis aritmeetilist tehet realiseerivas käsus on $\beta = 4$, siis sooritatakse:

a) liitmine, lahutamine ja kontroll-liitmine summaatori S kohtadel 0 kuni 39 ja pesas a 4 olevate arvudega (tulemus jääb summaatori S kohtadele 0 kuni 39);

b) korrutamine summaatori S kohtadel 0 kuni 34 ja pesas a 4 olevate arvudega (tulemus jääb summaatori S kohtadele 0 kuni 34);

c) jagamine summaatori S kohtadel 0 kuni 39 ja pesas a 4 olevate arvudega (tulemus jääb summaatori kohtadele 0 kuni 34).

Aritmeetilise tehte sooritamise tulemusena fikseeritud koma režiimis kirjutatakse järgusummaatorisse alati arv null.

Liikuva koma režiimis sooritatavate aritmeetiliste tehete puhul interpreteeritakse summaatori ja pesa $a\beta$ sisu liikuva komaga arvudena. Tehte tulemus saadakse alati normaliseeritud kujul, kusjuures tulemuse mantiss q jääb summaatorisse S, järk p aga järgusummaatorisse S_j.

Kui aritmeetilise tehte tulemus on negatiivne, siis töötatakse välja juhtimissignaal[17] $\omega = 1$, positiivse tulemuse korral aga $\omega = 0$. Programmeerimisel tuleb silmas pidada, et aritmeetiliste tehete tulemusena saadud null on alati positiivne. Märgime veel, et kui jagamisoperatsioonide 07 ja 47 puhul jagaja (pesas $a\beta$ olev arv) osutub nulliks, siis arvuti peatub ja signalisatsioonipaneelil süttib pealkirja *Деление на нуль* all vastav lamp.

3. Loogilised operatsioonid (koodid 10, 11, 12, 13 ja 14) sooritatakse kõik summaatoris S, kusjuures järgusummaatoris S_j säilib seal enne loogilise operatsiooni sooritamist olnud arv. Erandi selles suhtes moodustavad ainult nihutamisoperatsioonid (kood 11), mille puhul järgusummaatorisse kirjutatakse arv null. Vaatleme veidi lähemalt üksikuid loogilisi operatsioone.

Märgi omistamise operatsiooniga (kood 10) kirjutatakse pesa $a\beta$ 0-ndal kohal paiknev kahendnumber (arvu märk) summaatori S nullindale kohale, millega summaatoris S olevale arvule antaksegi pesas $a\beta$ paikneva arvu märk. Pesas $a\beta$ säilib arv, mis seal oli enne selle operatsiooni sooritamist. Kui märgi omistamise tulemusena saadakse summaatorisse S negatiivne arv, siis töötatakse välja signaal $\omega = 1$.

---

[17] Seda signaali kasutatakse tingimusliku suunamise käskudes. Kontroll-liitmise korral signaali ω välja ei töötata — säilib eelmise tehte teostamisel saadud signaali ω väärtus.

N i h u t a m i s o p e r a t s i o o n i g a nihutatakse summaatoris S olevat arvu n kahendkoha võrra. Arvu kahendkohti, mis nihutamisel lähevad summaatorist S välja, kuskil ei säilitata; nihutamisel vabanevate summaatori S kohtadele kirjutatakse aga nullid. Nihutamiskonstant n kirjutatakse käsus 13.—18. kahendkohale ja ta võib seega kaheksandsüsteemis omada väärtusi 0 kuni 77. Nihutamise suuna määrab konstandi n märk, mis kirjutatakse käsus 12-ndale kahendkohale. Kui n on positiivne (käsu 12-ndal kahendkohal on number 0), siis toimub nihutamine vasakule, kui aga n on negatiivne (käsu 12-ndal kahendkohal paikneb number 1), siis paremale. Kui nihutamiskäsus tunnus β võrdub nulliga, siis nihutatakse arvu märgikohta koos tema teiste kahendkohtadega, kusjuures summaatori S ületäitumiskoht operatsioonist osa ei võta (nn. l o o g i l i n e n i h u t a m i n e). Kui aga $\beta = 4$, siis summaatori S 0-ndal kohal paiknevat arvu märki ei nihutata, kuid operatsioonist võtab nüüd osa summaatori S ülestäitumiskoht (nn. a r i t m e e t i l i n e n i h u t a m i n e). See tähendab, et igakordsel vasakule nihutamisel nihkub summaatori esimesel kohal olev number ületäitumiskohale. Kui aritmeetilise nihutamise lõppedes jääb ületäitumiskohale number 1, siis antakse signaal $\varphi = 1$, kusjuures aga summaatorisse jääb õige tulemus.

O s a e r a l d a m i n e arvust (kood 12) toimub sel teel, et summaatoris S oleva arvu kõiki kahendkohti (kaasa arvatud ka märgikoht) korrutatakse loogiliselt pesas $a\beta$ oleva arvu vastavate kahendkohtadega. Meenutame, et loogiline korrutamine (konjunktsioon) on määratud järgmise tabeliga:

$$1 \wedge 1 = 1 \qquad 0 \wedge 1 = 0$$
$$1 \wedge 0 = 0 \qquad 0 \wedge 0 = 0.$$

Arv, mille teatud kahendkohti on tarvis eraldada, võib vaadeldava operatsiooni ajal paikneda kas summaatoris S või pesas $a\beta$. Selle operatsiooni teiseks operandiks peab siis olema konstant, milles eraldamisele kuuluvatel kahendkohtadel on numbrid 1, ülejäänud kahendkohtadel aga nullid.

F o r m e e r i m i n e (kood 13) toimub sel teel, et summaatoris S oleva arvu kõigile kahendkohtadele (kaasa arvatud märgikoht) liidetakse loogiliselt pesas $a\beta$ paikneva arvu vastavad kahendkohad.

Kahendkohtade loogiline liitmine (disjunktsioon) on teatavasti määratud tabeliga:

$$1 \vee 1 = 1 \qquad 0 \vee 1 = 1.$$
$$1 \vee 0 = 1 \qquad 0 \vee 0 = 0.$$

Arvude v õ r d l e m i s e l (kood 14) tehakse summaatoris S oleva arvu vastavate kahendkohtadega loogilise mittesamaväär-

suse tehe. Loogilise mittesamaväärsuse (mitteekvivalentsi) tehe sooritatakse järgmise tabeli järgi:

$$1 \not\equiv 1 = 0 \qquad 0 \not\equiv 1 = 1$$
$$1 \not\equiv 0 = 1 \qquad 0 \not\equiv 0 = 0.$$

Kui osa eraldamise, formeerimise või võrdlemise operatsiooni realiseerivas käsus $\beta = 4$, siis tehakse vastav operatsioon summaatori S ja topeltpesa $a4$ kohtadel 0 kuni 39 paiknevate kahendkohtadega. Kui aga käsus tunnus β võrdub nulliga, siis sooritatakse need operatsioonid summaatori kohtadel 0 kuni 39 oleva arvu ja 40-kohalise kahendarvuga, mille 20 vasakpoolset kahendkohta võrduvad pesas a 0 oleva arvu kahendkohtadega, kuid ülejäänud kahendkohad on nullid.

4. Operatsioone 02 ja 42 kasutatakse arvude toomiseks operatiivmälu pesadest summaatorisse. Sealjuures operatsiooni 02 tegemisel tõlgendatakse pesa $a\beta$ sisu fikseeritud komaga arvuna ja kirjutatakse ta sealt ilma muudatusteta summaatori S kohtadele 0 kuni 39, kustutades ühtlasi järgusummaatori S_j sisu.

Operatsiooni 42 sooritamisel vaadeldakse pesas $a\beta$ paiknevat arvu liikuva komaga arvuna ja kirjutatakse tema mantiss q summaatori S kohtadele 0 kuni 32, järk p aga järgusummaatorisse S_j. Ühtlasi mittenormaliseeritud liikuva komaga arvud normaliseeritakse.

Kui operatsioone 02 ja 42 realiseerivates käskudes $\beta = 4$, siis kirjutatakse summaatorisse 40-kohaline kahendarv topeltpesast $a4$. Kui aga $\beta = 0$, siis kirjutatakse summaatorisse 40-kohaline kahendarv, mille 20 vasakpoolset kahendkohta ühtivad pesas $a0$ oleva arvu vastavate kohtadega, ülejäänud kahendkohad on aga nullid. Operatsioone 16 ja 56 kasutatakse informatsiooni kirjutamiseks summaatorist operatiivmälu pesadesse. Operatsiooniga 16 kirjutatakse summaatori S sisu kui fikseeritud komaga arv operatiivmälu pesasse $a\beta$, kusjuures arvud summaatoris S ja järgusummaatoris S_j säilivad muutumatutena. Kui operatsiooni 16 realiseerivas käsus $\beta = 4$ (ja a on paarisarv), siis kirjutatakse topeltpesasse $a4$ summaatori S kohtadel 0 kuni 39 paiknev arv. Kui aga $\beta = 0$, siis kirjutatakse pesasse $a0$ summaatori S kohtadel 0 kuni 19 paiknev arv.

Operatsiooni 56 puhul tõlgendatakse summaatoris olevat arvu liikuva komaga arvuna, mille mantiss q on summaatoris S, järk p aga järgusummaatoris S_j. Kui operatsiooni 56 realiseerivas käsus $\beta = 4$ (ja a on paarisarv), siis sooritatakse see operatsioon järgmiselt: 1) summaatoris S paiknev arvu mantiss ümardatakse 32-kohaliseks ja summaatori S kohtadele 33 kuni 39 kirjutatakse nullid; 2) summaatoris olev arv normaliseeritakse; 3) summaatori S kohtadel 0 kuni 32 paiknev arvu mantiss kirjutatakse topeltpesa $a4$ kohtadele 0 kuni 32, järgusummaatori S_j sisu aga sama topeltpesa kohtadele 33 kuni 39.

Kui operatsiooni 56 realiseerivas käsus $\beta = 0$, siis pärast summaatoris paikneva mantissi ümardamist ja normaliseerimist kirjutatakse saadud mantissi esimesed 20 kahendkohta summaatori S kohtadelt 0 kuni 19 pesasse $a0$.

Pärast operatsiooni 56 sooritamist jääb summaatorisse S enne operatsiooni summaatoris olnud liikuva komaga arvu ümardatud ja normaliseeritud mantiss, järgusummaatorisse S_j aga (mantissi normaliseerimise käigus korrigeerituna) selle arvu järk.

5. Informatsiooni salvestamisel sisenditelt summaatorisse S kasutatakse operatsiooni 40. Arvutil «Ural-4» võib sealjuures olla kuni 64 kahekümnekohalist (või 32 neljakümnekohalist) sisendit. Igale sisendile on omistatud kindel number (kaheksandarv) ja temaga on ühendatud teatud sisendseade. Arvuti normaalkomplekti kuulub kaks perfokaartidelt lugemise seadet ja kaks telegraafilindilt lugemise seadet, kuid peale nende võib kasutada veel teisi sisendseadmeid: juhuslike arvude generaatoreid, sideliine jne.

Operatsiooniga 40 salvestatakse üks arv arvuti m-ndalt sisendilt summaatorisse S, kusjuures järgusummaatorisse S_j kirjutatakse samaaegselt arv null. Seda operatsiooni realiseerivas käsus 40 $m\beta$ paikneb sisendi number m kohtadel 13 kuni 18, kusjuures kohtadel 7 kuni 12 paiknevad numbrid operatsiooni sooritamist ei mõjuta. Kui käsus tunnus $\beta = 4$ (sisendi number m peab sel juhul olema paarisarvuline), siis informatsioon salvestatakse neljakümnekohaliselt sisendilt. Kui aga $\beta = 0$, siis kasutatakse operatsioonil kahekümnekohalist sisendit.

Informatsiooni salvestamiseks perfokaartidelt kasutatakse tavaliselt käsku 40 0002β, s. t. põhilisele perfokaartidelt lugemise seadmele omistatakse number 2. Selle käsu täitmisel loeb arvuti perfokaartidelt lugemisseadme lugemisbloki ees paiknevast kaardi reast arvu ja kirjutab selle muudatusteta summaatorisse S. Summaatorisse salvestatava arvu kõigi kahendkohtade lugemine ja kirjutamine summaatorisse S toimub üheaegselt, kusjuures kaardi 20-ndast veerust loetud kahendkoht kirjutatakse summaatori S kohale 0, 21-ndast veerust loetud kahendkoht summaatori kohale 1 jne., kaardi 59-ndast veerust loetud kahendkoht summaatori S kohale 39. Käsu 40 0002 0 toimel salvestatakse vaid arv perfokaardi veergudest 20 kuni 39 summaatori S kohtadele 0 kuni 19.

Ühel perfokaardil paikneva informatsiooni salvestamiseks arvutisse tuleb operatsiooni 40 teha 12 korda. Seejuures peab aga silmas pidama, et sisendseadme töötamise normaalse kiiruse juures (300 $\pm$ 10 kaarti minutis) tuleb kahe järjestikuse kaardirea lugemiseks vajalikud operatsioonid 40 sooritada ajavahemiku järel, mis ei ületa 6—7 millisekundit.[18] Kui see ajavahemik osutub pike-

[18] 6—7 millisekundi jooksul sooritab arvuti «Ural-4» fikseeritud koma režiimis 50—70 operatsiooni; 1,25 millisekundi jooksul 10 kuni 12 operatsiooni.

maks, siis arvuti jätab kaardi mõnedest ridadest informatsiooni lugemata.

Informatsiooni salvestamiseks telegraafilindilt kasutatakse tavaliselt kas käsku 40 0010 0 või käsku 40 0011 0 (s. t. normaalkomplekti kuuluvad telegraafisisendid varustatakse numbritega 10 ja 11). Sealjuures *n* sümbolist koosneva informatsioonimassiivi salvestamiseks tuleb *n* korda teha operatsiooni 40. Ühtlasi aga ei tohi lindil kõrvuti seisvatest ridadest informatsiooni salvestamiseks teostatavate operatsioonide 40 vahel olevad ajavahemikud ületada 1,25 millisekundit [18]: kui need ajavahemikud on pikemad, jäetakse osa informatsiooni lindilt salvestamata.

Käskude 40 0010 0 ja 40 0011 0 täitmine toimub järgmiselt: vastava numbriga telegraafilindilt lugemise seadme lugemispea ees paiknevast lindi reast loetakse sümboli kood ja seejärel teisendatakse loetud kood arvuti «Ural-4» kuuekohalisse trükkimisseadme koodi (vt. tabel 13 lk. 107) ning kirjutatakse summaatori *S* kohtadele 1 kuni 6 (summaatori ülejäänud kohtadele kirjutatakse nullid).

Lindil olevaid teenistusmärkide (registrite märgid: *vene* — kood 00000, *ladina* — kood 11111, *arvu* — kood 11011) koode summaatorisse *S* ei kirjutata. Käskudega 40 0010 0 ja 40 0011 0 saab telegraafilindilt informatsiooni summaatorisse *S* salvestada tarbekorral ka vahetult teises rahvusvahelises telegraafi koodis: tuleb ainult vastav telegraafilindilt lugemise seade lülitada sobivasse töörežiimi. Selles režiimis kirjutatakse lindilt loetud sümbolite ja teenistusmärkide koodid summaatori *S* kohtadele 1 kuni 5.

§ 11. ARVUTI «MINSK-22» KÄSKUDE SÜSTEEM.

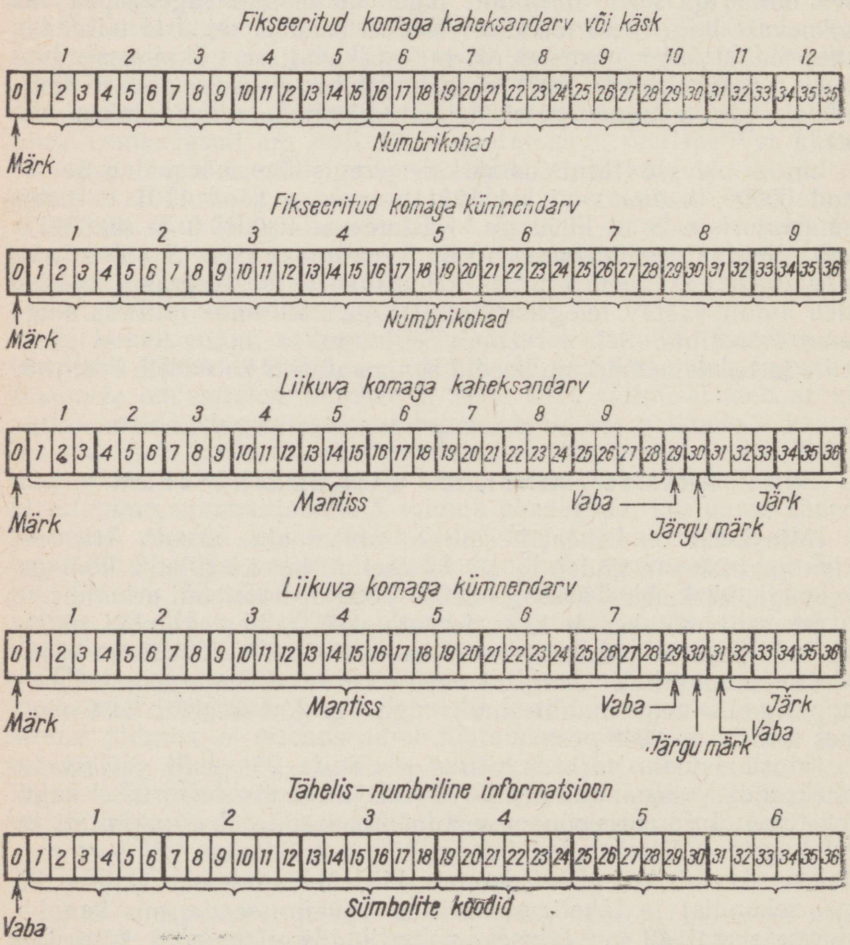
1. «Minsk-22» on kaheaadressiline universaalne arvuti. Aritmeetilisi operatsioone täidab ta nii fikseeritud kui ka liikuva komaga arvudega, keskmise kiirusega 5000—6000 operatsiooni sekundis.

Operatiivmäluks on kaks ferriitblokki, millest kumbki sisaldab 4096 pesa. Pesa pikkus on 37 kahendkohta. Välismäluna kasutatakse magnetlinti. Arvuti komplekti kuulub neli magnetlindikappi, igaüks nelja lindimehhanismiga. Iga lint sisaldab 64 tsooni, ühes tsoonis on 2048 pesa.

Informatsiooni saab arvutisse sisestada või sealt väljastada kaheksandarvudena, kümnendarvudena ja tähelis-numbrilisel kujul (tekstina). Informatsiooni sisestamiseks kasutatakse perfolinti ja perfokaarte. Väljastamisel informatsioon trükitakse või perforeeritakse. Arvuti komplekti kuulub kiirtrükkimisseade (kiirus 20 arvu sekundis) ja tähelis-numbriline trükkimisseade, mis kannab informatsiooni 42 cm laiusele paberilindile, ühte ritta kuni 128 märki (töökiirus 400 rida minutis).

Arvuti «Minsk-22» aritmeetiline seade sisaldab peale summaatori S veel kaks registrit: register R_1 ja register R_2. Summaator ja registrid sisaldavad samuti 37 kahendkohta nagu pesagi (summaatoris on küll veel lisaks ületäitumise koht). Neid kahendkohti on otstarbekohane nummerdada vasakult paremale numbritega 0 kuni 36.

Erinevate informatsiooniliikide salvestamisel on pesa kahendkohad jaotatud joonisel 20 näidatud viisil. Informatsiooni kirjutamine blanketile toimub aga joonisel 21 esitatud näite kohaselt. Märgime veel, et arvuti «Minsk-22» puhul tuleb eristada kaht liiki nulle — *normaliseeritud null* ja *täisnull*. Normaliseeritud



JOON. 20

							Leht nr.
Käsu nr.	T	±	OK	IND	A1	A2	
4621		+	10	72	6710	0121	8-ndsüsteemi arvud või käsud
10031		−	34	40	1120	7663	
3451		−	98	76	7894	5	10-ndsüsteemi arv fikseeritud komaga
6461		−	37	08	954−	04	10-ndsüsteemi arv liikuva komaga
13464			CT	A−	2M		Tekst

JOON. 21

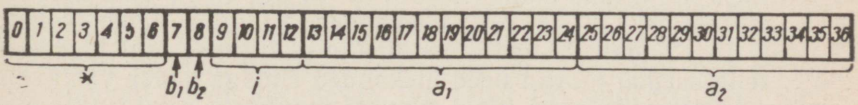
nulli mantissi kõik kahendkohad on nullid ja järk on —63. Täisnullil on aga üldse kõik kahendkohad (1—36) nullid, ainult märgikoha väärtus ei ole oluline. Täisnulli säilitatakse pidevalt arvuti ferriitmälu I bloki pesas aadressiga 0000 (sellesse pessa salvestamine on blokeeritud). Normaliseeritud null tuleb aga vajaduse korral salvestada koos konstantidega.

2. Operatiivmälu aadress tuleb arvuti «Minsk-22» puhul üles kirjutada kujul

ba,

kus *b* on ferriitbloki indeks ja *a* pesa aadress blokis. Bloki indeks *b* võib omandada vaid väärtusi 0 (I blokk) ja 1 (II blokk), aadress *a* aga väärtusi 0000—7777. Juhul $b = 0$ kirjutatakse tavaliselt ainult *a* väärtus, näiteks

7000 (I bloki pesa)
17000 (II bloki pesa).



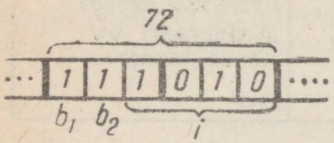
JOON. 22

Käsu üldkuju [19] on esitatud joonisel 22, kus sümboliga * on tähistatud operatsiooni kood (koos märgiga), sümboliga *i* aga nn. indekspesa aadress. Nagu jooniselt nähtub, on bloki number käsus aadressi ülejäänud osast eraldatud. Edaspidi kasutame seetõttu täiendavat tähistust:

[19] Mõnede käskude üldkuju erineb siin näidatust. Nende käskude struktuur on esitatud vastava käsu kirjelduse juures.

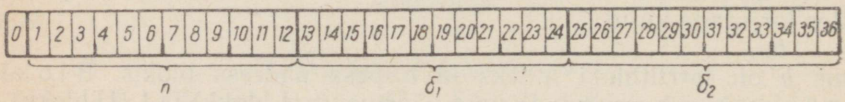
$A_1 = b_1a_1$ — käsu esimene aadress,
$A_2 = b_2a_2$ — käsu teine aadress.

Bloki numbri eraldatust tuleb eriti arvestada käskude kirjutamisel blanketile, kus aadresside A_1 ja A_2 blokinumbrid ja indekspesa aadress võetakse kokku üheks kahekohaliseks kaheksandarvuks. Näiteks on joonisel 21 toodud blanketi esimesse ritta kirjutatud käsk, kus $A_1 = 1\,6710$ ($b_1 = 1$, $a_1 = 6710$), $A_2 = 1\,0121$ ($b_2 = 1$, $a_2 = 0121$) ja $i = 0012$. Skemaatiliselt on see kujutatud joonisel 23.



JOON. 23

Käskude muutmiseks programmi täitmise ajal on arvutis «Minsk-22» kasutusel erilised indekspesad. Nendeks on reserveeritud pesad ferriitmälu I bloki algusest, aadressidega 0001—0017. Indekspesadesse salvestatakse käskude aadressiosade muutmiseks vajalik informatsioon joonisel 24 näidatud kujul, kus sümboliga δ_1 on tähistatud aadressi A_1 ümberaadresseerimissamm, sümboliga δ_2 — aadressi A_2 ümberaadresseerimissamm ja sümboliga n — nn. tsükli kordamiste arv.



JOON. 24

Programmi töö käigus, iga käsu täitmise eel kontrollib arvuti selle käsu indekspesa aadressi i. Kui $i \neq 0$, siis täidab arvuti käsu kujul, kus aadressiosale (a_1 ja a_2) on liidetud indekspesa aadressiosa δ_1 ja δ_2). Ülekanded 25-ndalt kahendkohalt 24-ndale ja 13-ndalt 12-ndale on niisuguse liitmise ajal blokeeritud[20], selle asemel toovad need ülekanded endaga kaasa vastavalt esimese ja teise aadressi bloki numbri muutmise esialgsega võrreldes vastupidiseks. Käsu koodiosa indekspesa abil muuta ei saa.

Arvuti «Minsk-22» käskude süsteemi kuuluvad käsud võib jaotada seitsmesse rühma:

1) aritmeetilisi ja loogilisi operatsioone realiseerivad käsud;
2) salvestamiskäsud;
3) suunamiskäsud;
4) spetsiaalaritmeetika käsud;

[20] Ainult magnetlindi ettetoomise käskude juures toimub siiski ülekanne 25-ndalt kahendkohalt 24-ndale.

5) teenistuslikud e. abistavad käsud;
6) magnetlindi poole pöördumise käsud;
7) sisend- ja väljundseadmete poole pöördumise käsud.

Iga aritmeetilise või loogilise operatsiooni, samuti aga salvestamis- ja spetsiaalaritmeetika käsu (välja arvatud salvestamine registrist R_1) tulemus jääb summaatorisse ja seda võib kasutada järgmise käsu juures.

Käskude kirjeldamisel kasutame edaspidi järgmisi tähistusi:
(A) — pesa A sisu;
$(S)_0$ — eelmise operatsiooni tulemus;
(S) — summaatori sisu pärast käsu täitmist;
(R_1) — registri R_1 sisu;
(R_2) — registri R_2 sisu;
$q(A)$ — arvu (A) mantiss;
$p(A)$ — arvu (A) järk;
k — täidetava käsu asukoht mälus;
$\rightarrow$ — salvestamine;
w — ületäitumise tunnus (ületäitumise korral $w = 1$, muidu $w = 0$).

3. Aritmeetilisi ja loogilisi operatsioone realiseerivatel käskudel on kõigil neli erinevat modifikatsiooni. Sümboolselt võib need üles kirjutada järgmiselt:

$(A_2) * (A_1) \rightarrow A_2$
$(A_2) * (A_1)$
$(S)_0 * (A_1) \rightarrow A_2$
$(S)_0 * (A_1)$

Need kirjutised tähendavad, et esimesel kahel juhul toimub tehe käsu aadressidega määratud pesades olevate arvudega, viimasel kahel juhul aga eelmise tehte tulemusega ja käsu esimese aadressiga antud pesas oleva arvuga. Salvestamine teise aadressiga määratud pesasse toimub vaid esimese ja kolmanda modifikatsiooni korral, kõigil neljal juhul jääb aga tulemus summaatorisse. Vastavalt nendele modifikatsioonidele on igal käsul neli erinevat operatsiooni koodi. Järgnevas (vt. tabel 15) ongi iga käsu kirjelduse juures antud neli koodi siintoodud järjekorras.

Aritmeetilisi operatsioone võib arvuti sooritada veel kahes erinevas režiimis — ümardamisega või ümardamiseta. Valitud režiim antakse ette kas arvuti puldilt või eriliste abistavate käskudega programmi töö käigus.

Liitmise või lahutamise tulemusena saadud null (nii normaliseeritud kui ka täisnull) on alati plussmärgiga. Miinusmärgiga nulli võib aga saada jagamisel (siis kui jagatav on null ning jagaja ja jagatav erimärgilised) või korrutamisel (tegurid on erimärgilised).

Liikuva komaga operatsioone on arvuti «Minsk-22» korral soovitav teha ainult normaliseeritud arvudega, sest vastasel korral võib jagamise ja korrutamise tulemus olla ebaõige. Rõhutame, et kõigi liikuva komaga operatsioonide tulemus on normaliseeritud.

Aritmeetilisi ja loogilisi operatsioone sooritavad käsud on kokkuvõtlikult esitatud tabelis 15. Lähemat selgitamist vajavad neist nähtavasti vaid nihutamistehted. Loogilise nihutamise puhul toimub kas pesas A_1 (operatsiooni koodid +60 ja +61) või summaatoris (koodid +62 ja +63) oleva arvu kõigi 37 kahendkoha nihutamine $p(A_1)$ koha võrra: $p(A_1) > 0$ korral vasakule ja $p(A_1) < 0$ korral paremale. Summaatori ületäitumisekoht nihutamisest osa ei võta. Aritmeetilise nihutamise puhul seevastu märgikoht nihutamisest osa ei võta, küll aga summaatori ületäitumisekoht. Kui nihutamise lõpul jäi ületäitumisekohale 1, siis antakse signaal $w = 1$.

4. Arvutis «Minsk-22» kasutatavad suunamis- ja salvestamiskäsud on koondatud tabelisse 16. Et suurema osa nende käskude puhul vastava operatsiooni sisu selgub juba tabelist, siis piirdume siin vaid mõne täiendava märkusega.

Aadressi salvestamisel registrist R_1 (operatsiooni kood —15) peab vastava käsu indekspesa aadress tingimata olema null. Seda käsku kasutatakse vaid pärast sisseviimist perfolindilt, mille puhul registrisse R_1 jääb aadress, mis on ühe võrra suurem selle pesa aadressist, kuhu salvestati viimane sisseviidud arv (sümbol).

Suunamist ületäitumise järgi tuleb kasutada alati siis, kui soovitakse arvutusi ületäitumise korral katkestamatult jätkata: kui signaali $w = 1$ andnud käsule ei järgne vahetult käsk koodiga —33, siis arvuti peatub.

Nulli järgi suunamise käsk (kood —34) loeb normaliseeritud nulli nulliks ainult vahetult pärast liikuva koma operatsioone. Muudel juhtudel reageerib ta vaid täisnullile.

Võtmega suunamise käsus (kood —35) kirjutatakse aadressiks a_1 (käsu kohtadele 13—24) nende võtmete numbrite summa, mille järgi soovitakse suunamist muuta. Arvuti juhtimispuldil paikneb nimelt seitse võtit, mis on varustatud numbritega 0001, 0002, 0004, 0010, 0020, 0040 ja 0100. Seega näiteks käsu

—35 00 0121 3262

täitmisel antakse juhtimine käsule (3262) siis, kui ükskõik milline võtmetest 0100, 0020 või 0001 on sisse lülitatud.

Võrdlemisega suunamise kirjeldus on tabelis 16 antud juhu jaoks, mil arvuti ei tööta perforeerimise režiimis. Perforeerimise puhul on $(S)_0$ nimelt mitte eelmise tehte tulemus, vaid väljundperforaatorist saadud informatsioon. Juhul $(S)_0 \neq (A_2)$ nüüd arvuti peatub.

5. Peale kirjeldatud põhiliste käskude on arvutis «Minsk-22» ette nähtud veel terve rida teenistuslikke käske ja käsud nn. spet-

Tabel 15

Operatsiooni koodid	Operatsiooni nimetus	Ületäituimise tunnuse $w = 1$ andmise tingimus
+04, +05, +06, +07	Võrdlemine	—
+10, +11, +12, +13	Fikseeritud koma arvude liitmine	$\|(S)\| \geqslant 1$
+14, +15, +16, +17	Liikuva koma arvude liitmine	$p(S) > 63$
+20, +21, +22, +23	Fikseeritud koma arvude lahutamine	$\|(S)\| \geqslant 1$
+24, +25, +26, +27	Liikuva koma arvude lahutamine	$p(S) > 63$
+30, +31, +32, +33	Fikseeritud koma arvude korrutamine	—
+34, +35, +36, +37	Liikuva koma arvude korrutamine	$p(S) > 63$
+40, +41, +42, +43	Fikseeritud koma arvude jagamine	$\|(S)\| \geqslant 1$ või $(A_1) = 0$
+44, +45, +46, +47	Liikuva koma arvude jagamine	$p(S) > 63$ või (A_1) pole normaliseeritud
+50, +51, +52, +53	Fikseeritud koma arvude moodulite lahutamine	—
+54, +55, +56, +57	Liikuva koma arvude moodulite lahutamine	—
+60, +61, +62, +63	Loogiline nihutamine	—
+64, +65, +66, +67	Aritmeetiline nihutamine	$\|(S)\| \geqslant 1$
+70, +71, +72, +73	Osa eraldamine (konjunktsioon)	—
+74, +75, +76, +77	Formeerimine (disjunktsioon)	—

Operatsiooni kood	Operatsiooni nimetus	Operatsiooni sisu
—10	Salvestamine	$(A_1) \rightarrow S$ ja $(A_1) \rightarrow A_2$
—11	Salvestamine märgimuuduga	$-(A_1) \rightarrow S$ ja $-(A_1) \rightarrow A_2$
—12	Mooduli salvestamine	$\lvert(A_1)\rvert \rightarrow S$ ja $\lvert(A_1)\rvert \rightarrow A_2$
—13	Salvestamine puldilt	Juhtimispuldi klaviatuuril valitud arv viiakse summaatorisse ja pesasse A_2
—14	Salvestamine märgi omistamisega	sign $(A_1) \cdot (A_2) \rightarrow S$ ja sign $(A_1) \cdot (A_2) \rightarrow A_2$
—15	Aadressi salvestamine registrist R_1	$(R_1) \rightarrow A_2$, $(S) = (S)_0$
—16	Salvestamine järgu omistamisega	$q(A_2) \cdot 2^{p(A_1)} \rightarrow S$ ja $q(A_2) \cdot 2^{p(A_1)} \rightarrow A_2$
—30	Tingimusteta suunamine	Juhtimine antakse pesas A_1 olevale käsule, $(S)_0 \rightarrow A_2$, $(S) = (S)_0$
—31	Suunamine alamprogrammi	Juhtimine antakse pesas A_1 olevale käsule, $(S) = (S)_0$ ja $-30\,00\,k+1\,0000 \rightarrow A_2$
—32	Suunamine märgi järgi	Juhul $(S)_0 \geqslant +0$ antakse juhtimine pesas A_1 olevale käsule, juhul $(S)_0 \leqslant -0$ aga pesas A_2 olevale käsule; $(S) = (S)_0$
—33	Suunamine ületäitumise järgi	Juhul $w = 0$ antakse juhtimine pesas A_1, $w = 1$ korral aga pesas A_2 olevale käsule, $(S) = (S)_0$, w kustutatakse
—34	Suunamine nulli järgi	Juhul $(S)_0 \neq 0$ antakse juhtimine pesas A_1 olevale käsule, juhul $(S)_0 = 0$ aga pesas A_2 olevale käsule, $(S) = (S)_0$
—35	Suunamine võtmega	Juhtimine antakse pesas A_2 olevale käsule siis, kui üks aadressis a_1 näidatud võtmetest on juhtimispuldil sisse lülitatud, muidu läheb juhtimine pesas $k+1$ olevale käsule; $(S) = (S)_0$

Operatsiooni kood	Operatsiooni nimetus	Operatsiooni sisu
—36	Katkestuse deblokeerimine	Juhtimine antakse pesas A_1 olevale käsule, $(A_2) \rightarrow S$, programmi katkestus deblokeeritakse (vt. käsk koodiga —06)
—37	Suunamine võrdlemisega	Juhul $(S)_0 \neq (A_2)$ antakse juhtimine pesas A_1 olevale käsule, juhul $(S)_0 = (A_2)$ aga pesas $k+1$ olevale käsule, $(S) = (S)_0$

siaalaritmeetika realiseerimiseks. Kõigi nende käskude lühike loetelu on esitatud tabelis 17, kuid mõningate operatsioonide sisu tuleb siiski veel täiendavalt selgitada.

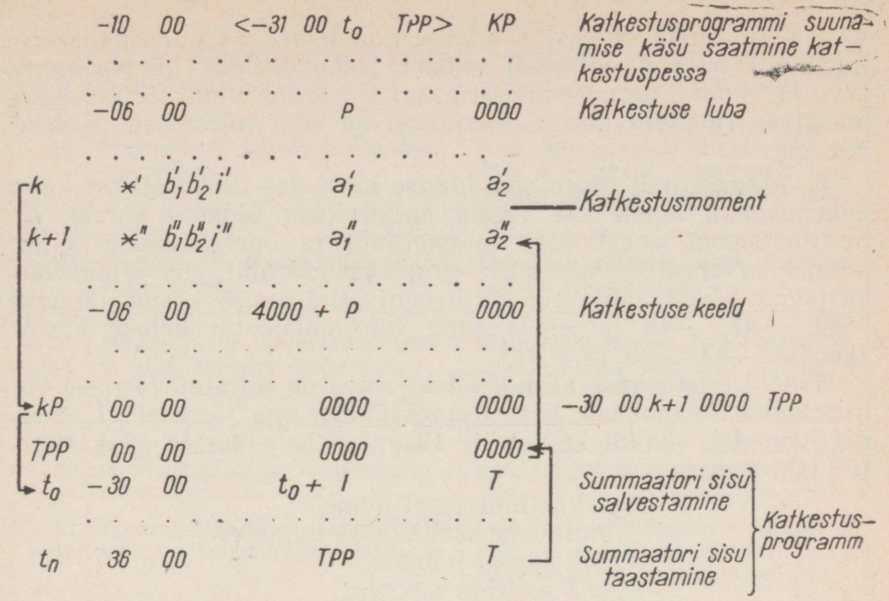
Katkestuse lubamine toimub käsuga —06 00 *P* 0000 ja keelamine käsuga —06 00 4000 + *P* 0000, kus mõlemal juhul *P* sisaldab ühe või mitu katkestuse põhjuse numbrit (vt. tabel 18) — vastavalt sellele, millistel põhjustel soovitakse rakendada programmi automaatse katkestamise süsteemi (mitme põhjuse korral kirjutatakse *P* kohale nende põhjuste numbrite summa).

Pärast käsu —06 00 *P* 0000 täitmist katkestatakse programmi töö koodiga *P* määratud põhjustel automaatselt niipea, kui vabaneb näiteks vastava väljundseadme register. Juhtimine läheb sealjuures kindlasse nn. k a t k e s t u s p e s s a. Igale katkestuse põhjusele vastab üks kindel katkestuspesa (vt. tabel 18), kuhu eelnevalt peab olema salvestatud alamprogrammi suunamise käsk (kood —31), millega juhtimine antakse erilisele katkestusprogrammile. Tavaliselt on katkestusprogrammiks mingi perforeerimis- või trükkimisprogramm. Et käskude loendaja sisu suurendamine on katkestusmomendil blokeeritud, siis tagasipöördepessa (TPP) moodustub käsk —30 00 *k* + 1 0000 (kus *k* on enne katkestamist viimasena täidetud käsu aadress), mis kindlustab juhtimise tagasipöördumise katkestusprogrammist. Pärast juhtimise üleandmist katkestusprogrammi on uus katkestamine blokeeritud. Selle blokeeringu likvideerimiseks paigutatakse katkestusprogrammi lõppu katkestuse deblokeerimise käsk (kood —36), mis ühtlasi taastab katkestamise momendil olnud summaatori seisu. Katkestamine toimub seega üldiselt joonisel 25 näidatud skeemi järgi, kus *T* on summaatori seisu salvestamiseks kasutatav tööpesa.

Tsükli lõpu käsu (kood —20) täitmisel toimub selles käsus antud aadressiga indekspesa sisu muutmine. Nimelt liidetakse seal olevatele ümberadresseerimissammudele δ_1 ja δ_2 (vt. joonis 24) pesas A_2 oleva arvu (kui käsu) aadressiosad a_1 ja a_2, kusjuures ülekanded 13-ndalt ja 25-ndalt kahendkohalt on blokeeritud (pesa A_2 kahendkohtade 0—12 sisu pole oluline). Samaaegselt toimub

Oper. kood	Operatsiooni nimetus	Operatsiooni sisu
—00	Peatus	$(S)_0 \rightarrow R_2$, $(A_1) \rightarrow R_1$, $(A_2) \rightarrow S$, arvuti peatub
—04	Ümardamise blokeerimine	$(S) = (S)_0$, järgnevad aritmeetilised operatsioonid sooritatakse ilma ümardamiseta
—05	Ümardamise deblokeerimine	$(S) = (S)_0$, järgnevad aritmeetilised operatsioonid sooritatakse ümardamisega
—06	Katkestuse luba või keeld	Juhul $a_1 < 4000$ lubatakse katkestamine aadressis a_1 näidatud põhjustel, juhul $a_1 > 4000$ ei lubata
—07	Perfolindi revers	Tööd katkestamata algab perfolindi tagasikerimine ühe massiivi võrra
—20	Tsükli lõpp	(A_2) aadressiosa liidetakse (i) aadressiosale, vähendatakse tsükli kordamiste arvu ühe võrra; kui viimane saab negatiivseks, siis antakse juhtimine pesas $k+1$ olevale käsule, muidu pesas A_1 olevale käsule
—70	Väikeste arvude korrutamine	$(A_1) \cdot (A_2) \cdot 2^{-36} \rightarrow S$, kui $\lvert(A_1)\rvert \cdot \lvert(A_2)\rvert \geqslant 2^{-36}$, siis $w = 1$
—71	Jäägiga jagamine	$\operatorname{sign}(A_2) \cdot \operatorname{res} \lvert(A_2):(A_1)\rvert \rightarrow S$
—72	Järkude liitmine	$q(A_2) \cdot 2^{p(A_1)+p(A_2)} \rightarrow S$ ja $q(A_2) \cdot 2^{p(A_1)+p(A_2)} \rightarrow A_2$
—73	Järkude lahutamine	$q(A_2) \cdot 2^{p(A_1)-p(A_2)} \rightarrow S$ ja $q(A_2) \cdot 2^{p(A_1)-p(A_2)} \rightarrow A_2$
—74	Kontrollsummeerimine	$(A_1)+(A_2) \rightarrow S$ ja $(A_1)+(A_2) \rightarrow A_2$, liitmisel toimub tsükliline ülekanne
—75	Normaliseerimine	Normaliseeritud $(A_1) \rightarrow S$ ja $\rightarrow A_2+1$, nihutuste arv $\rightarrow A_2$
—76	Ristsumma leidmine	Leitakse pesa A_1 kahendkohtades olevate ühtede arv n, $n \cdot 2^{-36} \rightarrow A_2$

indekspesas oleva tsükli kordamiste arvu n vähendamine ühe võrra. Kui vähendamise tulemus osutub mittenegatiivseks, siis salvestatakse see indekspessa endise n asemele ja antakse juhtimine pesas A_1 olevale käsule. Kui aga vähendamise tulemus osutub negatiivseks (s. t. n oli null), siis tulemust indekspessa ei salvestata ja juhtimine antakse lihtsalt programmi järgmisele käsule.



JOON. 25

Märgime, et tsükli lõpu käsk on ainus, mida täidetakse alati n. ö. loomulikul kujul — ilma indekspesa sisu abil korrigeerimiseta.

Jäägiga jagamise tehte (kood —71) tulemuseks on arvude (A_2) ja (A_1) jagamisel tekkiv jääk (tabelis res = resiid = jääk). Ümardamisrežiim selle käsu täitmisele mõju ei avalda.

Normaliseerimise tehe (kood —75) annab kaks tulemust, mis salvestatakse kahte järjestikusesse pessa A_2 ja $A_2 + 1$. Pesas

Tabel 18

Katkestuse põhjus	Põhjuse number	Katkestuspesa	Prioriteet
Väljastamine kiirtrükkimisseadmel või perforaatoril nr. 1	0001	0020	V
Väljastamine perforaatoril nr. 2	0002	0021	IV
Väljastamine laitrükkimisseadmel	0004	0022	III
Pseudokoodi (koodid —21 kuni —27) esinemine programmis	0010	0023	I
Teletaibilt sisseviimine	0020	0024	II

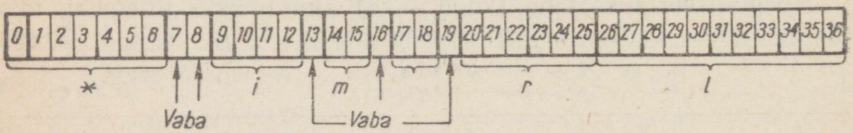
$A_2 + 1$ (ja summaatoris) saadakse nimelt arv (A_1) normaliseeritud kujul, pesa A_2 viimasel seitsmel kahendkohal (liikuva koma arvu järguna) aga normaliseerimisel teostatud nihutuste arv koos märgiga. Täisnulli normaliseerimisel on kõik tulemused võrdsed nulliga.

6. Magnetlindi poole pöördumise käskudes rühmitatakse käsu kahendkohad teisiti kui varem antud käsu üldkuju korral. Et informatsiooni vahetamine magnetlindi ja operatiivmälu vahel toimub ka arvutis «Minsk-22» grupikäskude abil, siis jagunevad vastavad käsud kahte rühma: tsooni ettetoomise käsud (koodid —40, —41, —46 ja —47) ning informatsioonivahetuse käsud (koodid —43, —44 ja —45).

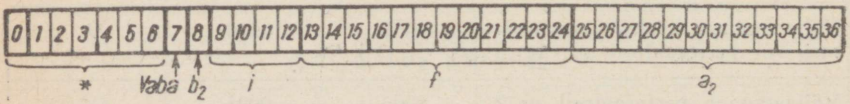
Tsooni ettetoomise käsu osadeksjaotus on esitatud joonisel 26, informatsioonivahetuse käsu osadeksjaotus aga joonisel 27. Nendel joonistel, samuti ka tabelis 19 antud kirjeldustes on kasutatud tähistust:

m — magnetlindi kapi number;
q — linditõmbemehhanismi number;
r — lindi tsooni number;
l — pesa aadress tsoonis;
$b_2a_2 = A_2$ — ferriitmälu pesa aadress;
f — ülekantava massiivi pikkus (pesade arv).

Tabelis 19 on esitatud magnetlindi poole pöördumise operatsioonid, mis peale esimese kahe realiseeritakse kõik grupikäskudena. Rõhutame veel kord, et grupikäsu koosseisu kuuluva esimese käsu struktuur on antud joonisel 26, teise käsu struktuur aga joonisel 27.



JOON. 26



JOON. 27

Täiendavaid selgitusi tabelis 19 toodud käsud nähtavasti ei vaja. Märgime vaid, et grupikäskude täitmise ajal toimub automaatselt ühtlasi nn. p a a r s u s e k o n t r o l l ning vea esinemise korral läheb juhtimine kohe grupikäskudele vahetult järgnevale käsule. Programmis peab selleks käsuks olema tingimusteta suunamine (kood —30), mis annab juhtimise uuesti grupikäsu algu-

Oper. koodid	Operatsiooni nimetus	Operatsiooni sisu
—40	Magnetlindi tsooni ettetoomine otsesuunas	Tööd katkestamata algab kapi *m* lindi *q* kerimine otsesuunas, kuni tsoon *r* on lugemispeade ette jõudnud
—41	Magnetlindi tsooni ettetoomine vastassuunas	Sama mis eelmine, ainult kerimine toimub vastassuunas
—46 —43	Magnetlindile kirjutamine tsooni ettetoomisega otsesuunas	Ferriitmälu pesast A_2 alates kirjutatakse *f* pesa sisu kapi *m* lindi *q* tsooni *r*, alates pesast *l*
—47 —43	Magnetlindile kirjutamine tsooni ettetoomisega vastassuunas	Sama mis eelmine, ainult algkerimine toimub vastassuunas
—46 —44	Kontroll-lugemine magnetlindilt tsooni ettetoomisega otsesuunas	Leitakse summaatorisse kapi *m* lindi *q* tsoonis *r* oleva *f* arvu kontrollsumma, alates pesast *l*
—47 —44	Kontroll-lugemine magnetlindilt tsooni ettetoomisega vastassuunas.	Sama mis eelmine, ainult algkerimine toimub vastassuunas
—46 —45	Magnetlindilt lugemine tsooni ettetoomisega otsesuunas	Alates kapi *m* lindi *q* tsooni *r* pesast *l* kirjutatakse *f* arvu ferriitmällu, alates pesast A_2; summaatorisse jääb kirjutatud arvude kontrollsumma
—47 —45	Magnetlindilt lugemine tsooni ettetoomisega vastassuunas	Sama mis eelmine, ainult algkerimine toimub vastassuunas

sesse. Vea teistkordsel esinemisel arvuti peatub, grupikäsu veata täitmisel jäetakse aga see järgnev käsk vahele.

7. Sisend- ja väljundseadmete poole pöördumise käske me siin eriti üksikasjalikult ei tutvusta — nende lähemaks tundmaõppimiseks tuleb kasutada vastavaid spetsiaalseid instruktsioone.[21] Piirdume vaid vastavate käskude loetelu ja lühikeste selgituste esitamisega.

Sisendseadmete poole pöördumise käskude loetelu on esitatud tabelis 20. Nende käskude puhul tuleb täiendavalt arvestada, et operatsiooni lõppedes paikneb registris R_1 arv, mis on ühe võrra

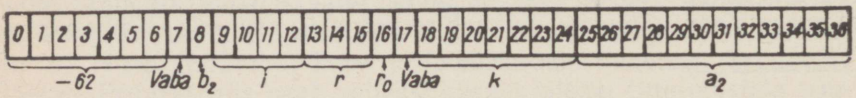
[21] Selleks võib kasutada näiteks väljaannet [33].

Oper. kood	Operatsiooni nimetus	Operatsiooni sisu
—50	Arvude sisseviimine perfolindilt	Üks massiiv arve viiakse perfolindilt ferriitmällu alates pesast A_2, ühtlasi leitakse nende arvude kontrollsumma summaatorisse
—51	Arvude kontroll-lugemine perfolindilt	Sama mis eelmine, ainult ferriitmällu kirjutamist ei toimu
—52	Teksti sisseviimine perfolindilt	Üks massiiv sümboleid loetakse perfolindilt, teisendatakse 6-kohalisse koodi ja salvestatakse ferriitmällu alates pesast A_2, summaatorisse leitakse kontrollsumma
—53	Teksti kontroll-lugemine perfolindilt	Sama mis eelmine, ainult ferriitmällu kirjutamist ei toimu
—54	Informatsiooni sisseviimine perfokaardilt	Käsu esimeses aadressis näidatud arvult perfokaartidelt loetakse informatsioon ja kirjutatakse ferriitmällu alates pesast A_2, summaatorisse leitakse kontrollsumma
—17	Sisseviimine teletaibilt	Toimub ühe teletaibil valitud sümboli viimine pessa A_2

suurem viimase sisseviidud sümboli aadressist ferriitmälus. Perfolindilt sisseviimine (kood —50) toimub nn. aadressilis-grupiviisilises režiimis, s. t. arv, mille ette on perforeeritud aadress, salvestatakse selle aadressiga määratud pessa, aadressi puudumisel aga eelmisele sisseviidud arvule järgnevasse pessa.

Informatsiooni väljastamine kiirtrükkimisseadmel või perforaatoril nr. 1 toimub käsuga, mille operatsiooni kood on —60, väljastamine perforaatori nr. 2 või teletaibi abil aga käsuga, mille operatsiooni kood on —61. Kõigil nendel juhtudel peab väljastatav arv või sümbol paiknema pesas A_2, väljastamise viis aga antakse ette aadressi a_1 kohal erilise koodiga.

Tähelis-numbrilise informatsiooni väljastamist laitrükkimisseadmel teeb käsk operatsiooni koodiga —62. Selle käsu puhul rühmitatakse tema kahendkohad joonisel 28 näidatud viisil, kus



JOON. 28

r	$r_0 = 0$	$r_0 = 1$
0	Liikuva komaga kümnendarv	(puudub)
1	Kaheksandarv	Keeratakse paberit rea võrra edasi, informatsioon trükkimisseadme mälus säilib
2	Fikseeritud komaga kümnendarv	Kustutatakse trükkimisseadme mälu ilma midagi trükkimata
3	Kümnendarv ilma märgita ja alguses olevate nullideta	(puudub)
4	Sümbol-number või vene tähestiku täht	(puudub)
5	Kuus sümbolit — numbrit või vene tähestiku tähte	Trükkimine rea vahetamisega, informatsioon säilib
6	Sümbol-number või ladina tähestiku täht	Trükkimine rea vahetamiseta, informatsioon kustutatakse
7	Kuus sümbolit — numbrit või ladina tähestiku tähte	Trükkimine rea vahetamisega, informatsioon kustutatakse

$A_2 = b_2a_2$ annab väljastatava informatsiooni asukoha ferriitmälus, k — trükkimisseadme mälu positsiooni numbri, millest alates toimub salvestamine, r — informatsiooni tõlgendamise või trükkimise viisi (vastavalt tabelile 21) ja r_0 — käsu täitmise režiimi ($r_0 = 0$ tähendab informatsiooni kandmist ferriitmälust trükkimisseadme mälusse, $r_0 = 1$ aga trükkimisseadme mälu sisu väljatrükkimist).

Nagu tabelist 21 ($r_0 = 0$ veerust) nähtub, võib sümboleid nende salvestamisel trükkimisseadme mällu tõlgendada kaht moodi. Tabelis 22 ongi toodud arvuti «Minsk-22» puhul kasutatav sümbolite koodide loetelu. Selle tabeli vasakpoolses osas toodud 32-l koodil on alati sama tähendus, parempoolsed 32 koodi on aga kahesuguse tähendusega, sõltuvalt sellest, kas neid tõlgendatakse vene tähtedena või ladina tähtedena (tabelis sulgudes).

Viimase väljastamisviisina saab arvutis «Minsk-22» kasutada veel perforeerimist perfokaartidele. Selline väljastamine toimub 80-kohaliste ridade kaupa. Seepärast tuleb ferriitmälus eraldada 36 pesa (nn. perforeerimise puhvermälu), kuhu enne iga kaardi perforeerimist teisendatakse vastav informatsioon. Väljundamine peab toimuma erilises nn. perforeerimisrežiimis, mis algab käsuga —03 00 0000 0000 ja lõpeb käsuga —03 00 4000 0000. Perforeerimis-

Kahendkood	Sümbol	Kahendkood	Sümbol
000000	0	100000	А(A)
000001	1	100001	Б(B)
000010	2	100010	В(W)
000011	3	100011	Г(G)
000100	4	100100	Д(D)
000101	5	100101	Е(E)
000110	6	100110	Ж(V)
000111	7	100111	З(Z)
001000	8	101000	И(I)
001001	9	101001	Й(J)
001010	+	101010	К(K)
001011	—	101011	Л(L)
001100	/	101100	М(M)
001101	,	101101	Н(N)
001110	.	101110	О(O)
001111	␣	101111	П(P)
010000	10	110000	Р(R)
010001	↑	110001	С(S)
010010	(	110010	Т(T)
010011	)	110011	У(U)
010100	x	110100	Ф(F)
010101	=	110101	Х(H)
010110	;	110110	Ц(C)
010111	[	110111	Ч(tühik)
011000	]	111000	Ш(tühik)
011001	*	111001	Щ(tühik)
011010	'	111010	Ы(Y)
011011	≠	111011	Ь(X)
011100	<	111100	Э(tühik)
011101	>	111101	Ю(tühik)
011110	≥	111110	Я(Q)
011111	:	111111	tühik

režiimi ajal on programmi töö katkestamine mistahes põhjusel blokeeritud, summaatori sisu säilib.

Väljastamist perfokaartidele teostab käsk operatsiooni koodiga —63, mille toimel pesa A_2 sisu saadetakse perforeerimisele. Selleks, et välja perforeerida terve rida 80 veeru ulatuses, peab programmis olema järjest 3 niisugust käsku: kaks esimest saadavad perforeerimisele kumbki 37 kahendkohta, kolmas aga 6 esimest kahendkohta koos märgikohaga.

Iga kaardi perforeerimisel saab kontrollida eelnevalt perforeeritud kaardi õigsust. Kontrollitakse võrdlemisega suunamise käsu (kood —37) abil. Selle käsu täitmisel perforeerimisreziimis toimub kontrollitavalt kaardilt loetud informatsiooni võrdlemine ferriitmälu pesades asuva informatsiooniga. Nendesse pesadesse (nn. kontrollimise puhvermälu) peab olema salvestatud kontrollitavale kaardile perforeeritud informatsioon. Ühe kaardi kontrollimiseks on tarvis kolm võrdlemisega suunamise käsku.

IV PEATÜKK

PROGRAMMEERIMISE PÕHIMÕISTED

§ 12. BLOKK-SKEEMI KOOSTAMINE

1. Ülesande lahendamine elektronarvutis toimub vastava programmi alusel. Programmiks nimetatakse ülesande lahendamise algoritmi, mis on üles kirjutatud arvuti käskude abil. Ta kujutab endast arvuti käskude teatud jada, millele on lisatud arvutusprotsessi kirjeldamiseks tarvilikud abiarvud (konstandid, skaalad jms.). Rõhutame, et ülesande algandmed programmi koosseisu ei kuulu. Näiteks arvutades ruutvõrrandi $ax^2 + bx + c = 0$ reaalarvulisi lahendeid valemi

$$x = \frac{-0{,}5b \pm \sqrt{0{,}25b^2 - ac}}{a}$$

järgi, vaadeldakse kordajaid 0,5 ja 0,25 programmi kuuluvate konstantidena, kordajaid a, b ja c aga ülesande algandmetena.

Seega jaguneb ülesande lahendamiseks vajalik informatsioon kahte suurde ossa: programmiks ja algandmeteks. Algandmeid lubatud piirides varieerides saab ühe ning sellesama programmiga lahendada vastava ülesande mitmeid üksteisest erinevaid variante.

Ülesande ettevalmistamine lahendamiseks elektronarvutil seisnebki vajalike informatsioonimassiivide koostamises, perforeerimises ja kontrollimises. Sellega seotud töö võib jaotada järgmisteks etappideks:

1) ülesande lahendusmeetodi ja -algoritmi valimine;
2) algoritmi programmeerimine;
3) programmi ja algandmete perforeerimine ja kontrollimine.

2. Lahendusmeetodi (-algoritmi) valimisel tuleb kõrvuti ülesande tüübi ja mitmesuguste iseärasustega arvestada ka arvuti tehnilisi võimalusi (operatiivmälu mahtu, töötamise kiirust jm.). Eeskätt tuleb aga silmas pidada, et valitud meetod peab võimaldama ülesande lahendamise taandada arvuti põhioperatsioonide mingi lõpliku jada sooritamisele ja tagama nõutud täpsusega tulemuste saamise võimalikult ökonoomselt (s. t. väikeste kulutustega).

Et arvutid sooritavad ainult teatud kindlaid operatsioone (aritmeetilised tehted, arvude võrdlemine ning nihutamine jt.), siis saab neil ülesandeid lahendada üksnes nn. numbriliste meetodite abil. Näiteks trigonomeetriliste funktsioonide väärtuste arvutamisel võib kasutada vastavaid ridu, määratud integraalide arvutamisel numbrilise integreerimise meetodeid jne. (Üksikasjalisemalt vaatleme meetodi valimisega seotud küsimusi VI peatükis.)

Valitud meetodi alusel koostatakse ülesande lahendamisalgoritm, mis esitatakse matemaatiliste seoste, valemite ja sõnaliste selgituste abil.

Toome näitena algoritmi suuruse $y = \sqrt{z}$, kus $z = \dfrac{bx^2 + a}{a^2 + b^2}$, väärtuste arvutamiseks (täpsusega ε), kui $1 \leqslant a \leqslant 2$; $3 \leqslant b \leqslant 4$ ja $4{,}5 \leqslant x \leqslant 20$. Ruutjuure arvutamiseks kasutatakse Newtoni iteratsioonimeetodit, kusjuures alglähendi y_0 arvutame seosest $y_0 = \dfrac{1}{2} z$. Arvutuste teostamiseks tuleb algandmetele a, b, x ja ε anda konkreetsed arvulised väärtused.

Vastav algoritm on järgmine:

1) arvutada $z = \dfrac{bx^2 + a}{a^2 + b^2}$;

2) arvutada $y_0 = \dfrac{1}{2} z$;

3) võtta $i = 0$;

4) arvutada $y_{i+1} = \dfrac{1}{2}\left(\dfrac{z}{y_i} + y_i\right)$;

5) kui $|y_{i+1} - y_i| \leqslant \varepsilon$, siis jätkata ülesande lahendamist alates punktist 8, vastasel korral aga teha seda alates punktist 6;

6) suurendada indeksi i väärtust ühe võrra;

7) jätkata ülesande lahendamist alates punktist 4;

8) võtta y väärtuseks lähendi y_{i+1} väärtus;

9) lõpetada arvutamine.

Seega kujutab ülesande lahendamisalgoritm instruktsioonide teatud lõplikku jada, mille täitmise tulemusena lahendatakse püstitatud ülesanne.

Valemite ja sõnaliste selgituste abil üleskirjutatud algoritm ei määra kirjeldatavat arvutusprotsessi tavaliselt üheselt, vaid jätab vabaks mõnede suuruste väärtuste arvutamise järjekorra. Nii pole näitena toodud algoritmis näidatud, kas muutuja z väärtuse arvutamisel tuleb leida enne lugeja (s. o. avaldise $bx^2 + a$) või nimetaja (avaldise $a^2 + b^2$) väärtus. Samuti ei kajastu sellistes algoritmides otseselt kõik operatsioonid, mida arvuti peab vaadeldava ülesande lahendamiseks tegema. Näiteks pole neis ette nähtud ülesande lahendamiseks vajaliku informatsiooni salvestamist arvutisse, algandmete teisendamist kümnendsüsteemist kahend-

süsteemi, tulemuste väljastamist; pole näidatud, millised arvutuste vahetulemused tuleb ajutiseks säilitamiseks salvestada summaatorist operatiivmällu jne.

Kõik niisugused detailid on aga programmi koostamisel õlulised. Seetõttu alustatakse ülesande programmeerimist sellega, et viiakse tema lahendamisalgoritm programmi koostamiseks sobivale kujule. Teiste sõnadega, arvestades, et ülesande lahendajaks on elektronarvuti, tehakse algoritmis vajalikud täiendused ja täpsustused. Tavaliselt vormistatakse sel teel saadud ülesande lahendamiseeskiri blokk-skeemina.

3. B l o k k - s k e e m i koostamisega seotud töö algab tavaliselt sellega, et ülesande lahendamisalgoritm jaotatakse operatsioonideks, mis on kirjeldatavad arvuti käskude abil; selgitatakse välja, missugused vahetulemused tuleb arvutuste käigus ajutiseks säilitamiseks salvestada operatiivmällu, ja määratakse operatsioonide sooritamise otstarbekohane järjekord. Kõrvuti sellega valitakse ka kõigile arvulistele suurustele esitusviis (s. t. määratakse, missuguseid neist esitatakse liikuva, millised fikseeritud komaga arvudena). Seejärel moodustatakse järjestikku tehtavatest operatsioonidest rühmad (blokid) nii, et iga blokk kirjeldab vaadeldava ülesande lahendamisprotsessist mingit loogiliselt terviklikku osa (näiteks mingi operatiivmällu salvestatava tulemuse arvutamist, teatud tingimuse kehtivuse kontrollimist vms.).

Eeskätt tuleb blokkide moodustamisel silmas pidada järgmisi nõudeid:

1) iga operatsioon võib kuuluda ainult ühte blokki;
2) bloki realiseerimisel sooritatakse alati kõik temasse kuuluvad operatsioonid;
3) operatsioonid, mis on seotud mingi tingimuse kontrollimisega, moodustavad eraldi bloki.

Peale blokkide moodustamist hinnatakse ligikaudu ülesande lahendamiseks vajaliku informatsiooni salvestamiseks tarvilike pesade arv. Seda arvu, arvuti operatiivmälu mahtu ja lahendatava ülesande iseärasusi arvestades lülitatakse algoritmi veel blokid, mis näevad ette:

1) programmi ja algandmete salvestamise arvuti operatiivmällu;
2) algandmete teisendamise kahendsüsteemi;
3) tulemuste teisendamise kümnendsüsteemi ja väljastamise;
4) arvuti töö kontrollimise;
5) informatsiooni vahetamise operatiiv- ja välismälu vahel.

Vajalike täienduste arv, iseloom ja asend algoritmis sõltub otseselt lahendatavast ülesandest. Näiteks, kui ülesande algandmed on kaheksandarvud, siis loomulikult ei nähta blokk-skeemis ette nende teisendamist kahendsüsteemi; kui ülesande lahendamiseks vajaliku materjali salvestamiseks piisab operatiivmälu pesa-

dest, ei lülitata algoritmi blokke informatsiooni vahetamiseks operatiiv- ja välismälu vahel jne. Täpsemalt, punktides 2 ja 5 nimetatud täiendused võivad teatud ülesannete korral osutuda mittevajalikeks. Punktides 1,3 ja 4 mainitud osi peab aga sisaldama iga programmi blokk-skeem. (Algoritmi täiendamisega seotud küsimusi käsitletakse VI peatükis.)

4. Blokk-skeemide koostamisel tuleb eristada kaht liiki blokke: täidesaatvaid ja kontrollivaid. Täidesaatvaks nimetatakse niisugust blokki, mille realiseerimise järel tuleb alati asuda ühes kindlas blokis paiknevate operatsioonide sooritamisele. Oma iseloomult on täidesaatvale blokile vastav operatsioonide rühm selline, et tema sooritamisega saadakse vähemalt üks tulemus, mis on kas lahendatava ülesande lõppresultaadiks või mida kasutatakse algoritmi edasises töös. Enamasti on täidesaatva bloki sisuks mingi aritmeetilise või loogilise avaldise väärtuse leidmine ja salvestamine arvuti operatiivmällu. Iseseisvate täidesaatvate blokkidena vaadeldakse tavaliselt ka järgmisi operatsioone:

1) informatsiooni salvestamine telegraafilindilt või perfokaartidelt operatiivmällu;
2) informatsioonimassiivi kirjutamine operatiivmälust välismällu ja vastupidi;
3) tulemuse väljatrükkimine või -perforeerimine;
4) reavahe trükkimine;
5) standardse programmi poole pöördumine;
6) arvuti peatamine.

Kontrollivaks nimetatakse niisugust blokki, millele vastavate operatsioonide sooritamisega kontrollitakse teatud tingimuse kehtivust. Sõltuvalt tulemusest valitakse kahest näidatud blokist välja üks, mille realiseerimisega tuleb jätkata ülesande lahendamist. Nimelt, kui tingimus osutub täidetuks, siis jätkatakse lahendamist ühele näidatud blokile, vastasel korral aga teisele blokile vastava operatsioonide rühma sooritamisega. Ühtki hiljem vajalikku tulemust seda tüüpi bloki realiseerimine ei anna. Isegi kui mingi tulemus leitakse, siis kasutatakse seda ainult sellessamas blokis.

Skeemis kujutatakse blokid tavaliselt ristkülikutena, mille sisse on märgitud neile vastavate operatsioonirühmade kirjeldused. Blokkide realiseerimise järjekord näidatakse skeemil nooltega. Igast täidesaatvast blokist väljub üksainus nool, mis suundub vahetult järgmisena teostatavasse blokki.

Kontrollivatest blokkidest väljub alati kaks noolt. Sõnaga «jah» või märgiga «+» varustatud nool viib selle bloki juurde, mis tuleb realiseerida siis, kui kontrollitav tingimus on täidetud. Teine nool, mille juurde on kirjutatud kas «ei» või «—», suundub aga blokki, millele vastavad operatsioonid sooritatakse tingimuse mittetäidetuse korral.

Kui mingisse blokki ühtki noolt ei suundu, siis peab see olema

skeemi esimene blokk (s. o. blokk, millele vastavate operatsioonide sooritamisega algab ülesande lahendamine). Igas skeemis võib olla ainult üks esimene blokk. Kui skeemi mingist blokist ühtki noolt ei välju, siis peaks see olema skeemis viimane (s. o. blokk, millele vastavate operatsioonide sooritamisega ülesande lahendamine lõpeb). Ühes skeemis võib olla mitu viimast blokki. Et oleks lihtsam avastada märkimata jäänud nooli, siis kirjutatakse tavaliselt ka skeemi viimaste blokkide juurde väljuvad nooled koos sõnadega *lõpp* ja esimesse blokki suunduv nool, mis varustatakse sõnaga *algus*. Pärast sellist täiendamist peab skeemi igasse blokki suunduma (samuti sealt väljuma) vähemalt üks nool.

Blokile vastavat operatsioonirühma kirjeldatakse skeemis aritmeetiliste või loogiliste avaldiste, matemaatiliste seoste, vahel ka sõnalise teksti abil. Seejuures kasutatakse tavaliselt tabelis 23 selgitatud sümboolikat.

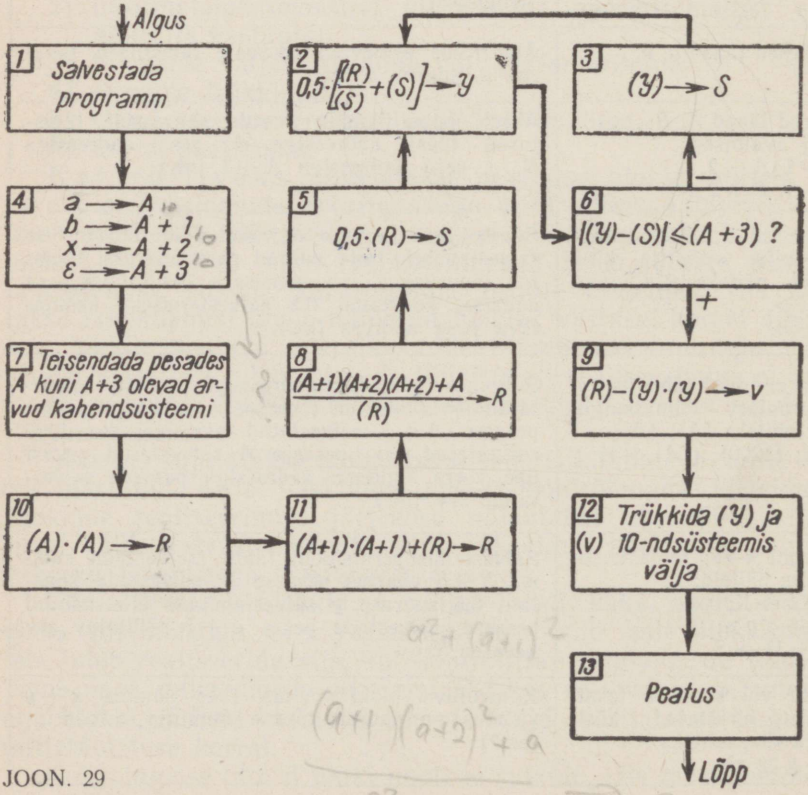
Tabel 23

Sümbol	Sümboli tähendus ja näidete selgitused (sulgudes)
Väikesed tähed *a, b, ...*	Algoritmis esinevate suuruste tähistused (suurus *a*, suurus *b* jne.)
suured tähed *A, B, ...* ning avaldised $A+1, A+2, \ldots$ $B+1, B+2, \ldots$	Arvuti operatiivmälu pesade aadresside tähistused (pesa aadressiga *A*, pesa aadressiga *B*,... pesa aadressiga $A+1$ jne.)
Väike täht või konstant sümbolite «<» ja «>» vahel, näiteks $<a>$, $<0{,}5>$, ...	Sümbolite «<» ja «>» vahel näidatud suuruse salvestamiseks ette nähtud pesa aadressi tähistus (suuruse *a* salvestamiseks kasutatava pesa aadress, konstandi 0,5 salvestamiseks kasutatava pesa aadress jne.)
Pesa aadressi tähistus või naturaalarv ümarsulgudes, näiteks (A); $(A+1)$; (1200), $((A)+1)$ jne.	Sulgudes näidatud aadressiga pesasse salvestatud arvu tähistus (pesasse *A* salvestatud arv; pesasse $A+1$ salvestatud arv; pesasse 1200 salvestatud arv; pesasse *A* salvestatud arvust ühe võrra suurema aadressiga pesasse salvestatud arv jne.)
Sümbol «→»; tema kasutamise näiteid: $a+b \rightarrow X$; $0{,}5 \rightarrow <y>$; $(A+1) \rightarrow S$	Salvestamisoperatsiooni tähis (salvestada summa $a+b$ väärtus pesasse *X*; salvestada konstant 0,5 suuruse *y* salvestamiseks ette nähtud pesasse; salvestada pesas $A+1$ säilitatav arv pesasse *S*)
Seose või sõnalise teksti järele kirjutatud küsimärk «?»; näiteks $x+y \geqslant z$?	Kontrolliva bloki tunnus (kas summa $x+y$ väärtus on suurem-võrdne muutuja *z* väärtusest?)

Skeemi blokid on otstarbekohane varustada järjekorranumbritega. Eeskätt võimaldab see väljaspool skeemi anda blokkide kohta täiendavaid selgitusi (näiteks näidata, missugused programmi käsud vastavad igale blokile). Tavaliselt kirjutatakse järjekorranumber bloki ülemisse vasakpoolsesse nurka, kusjuures nummerdamine viiakse läbi nii, et liikudes skeemis vasakult paremale ja ülalt alla blokkide järjekorranumbrid kasvavad.

5. Vaatleme nüüd näitena mõnede programmide blokk-skeeme. Arvestades, et programmi ja algandmete salvestamine, tulemuste väljastamine, andmete teisendamine kümnendsüsteemist kahendsüsteemi ja vastupidi toimub vastavate standardsete programmiosade abil, siis nende tööde sooritamiseks me toodavates näidetes blokke detailselt välja ei kirjuta (nende programmiosade koostamist ja kasutamist käsitleme hiljem eraldi). Piirdume ainult sellega, et näitame ära, kuhu tuleb skeemis vastavad programmiosad paigutada.

Näide 1. Vaatleme teises punktis toodud algoritmile vastava programmi blokk-skeemi ühte võimalikku varianti, milles on üks kontrolliv ja kaksteist täidesaatvat blokki (joonis 29).



JOON. 29

Ülesande lahendamiseks salvestatakse programm (blokk 1) ja algandmed (blokk 4) arvuti operatiivmälus selleks ette nähtud pesadesse. Salvestamine toimub kahe massiivina: esimese neist moodustab programm, teise aga algandmed. Algandmed salvestatakse pesadesse A, $A+1$, $A+2$ ja $A+3$ järjekorras: a, b, x ja ε. Selles järjestuses tuleb nende väärtused kanda ka perfokaartidele või telegraafilindile.

Bloki 7 realiseerimisega teisendatakse algandmed kahendarvudeks. Seejärel arvutatakse muutuja z väärtus (blokid 10, 11 ja 8). Muutuja z väärtuse leidmisega seotud arvutused on jaotatud kolme blokki selleks, et ära näidata operatsioonide sooritamise järjekorda ja saadud vahetulemuste salvestamiseks kasutatavaid pesi. Nimelt leitakse esitatud skeemi järgi arvutades kõigepealt a^2 väärtus ja salvestatakse see pesasse R (blokk 10). Seejärel arvutatakse b^2 ja liidetakse sellele pesas R säilitatav a^2 väärtus. Et viimast edasistes arvutustes ei kasutata, siis salvestatakse nimetaja a^2+b^2 väärtus samasse pesasse R (blokk 11). Lõpuks leitakse pesasse R muutuja z väärtus (blokk 8).

Viiendas blokis arvutatakse alglähend y_0, teises aga y_1 ja kontrollitakse, kas saadud tulemus rahuldab nõudeid täpsuse kohta (blokk 6). Kui nõutav täpsus pole veel saavutatud, võetakse y_1 uueks alglähendiks (blokk 3) ja leitakse järgmine lähend (blokk 2). Vastasel korral aga arvutatakse $v=z-y_i^2$ (blokk 9), teisendatakse v ja y_i kümnendsüsteemi ning trükitakse välja (blokk 12). Suuruse v väärtus leitakse ja väljastatakse arvuti töö kontrollimiseks. Kolmeteistkümnendas blokis peatatakse arvuti. Ülesande lahendamine on lõppenud.

Näide 2. Koostame blokk-skeemi programmile, mis järjestab arvud a_0, a_1, a_2, ..., a_n ümber nii, et nad moodustavad kasvava jada. Tähistame arvude a_0, a_1, a_2, ..., a_n salvestamiseks ette nähtud pesade aadressid vastavalt $A+0$, $A+1$, $A+2$, ..., ..., $A+n$. Ülesande lahendamisel tulevad pesadesse $A+i$ ($i=0, 1, \ldots, n$) salvestatavad arvud ümber paigutada nii, et kehtiks seos

$$(A+0)\leqslant(A+1)\leqslant(A+2)\leqslant\ldots\leqslant(A+n). \qquad (1)$$

Blokk-skeemi koostamisel võtame aluseks algoritmi, mis taandab vaadeldava ülesande lahendamise mitme üksteisega sarnase etapi (ehk sammu) järjestikusele realiseerimisele.

Esimesel sammul võrreldakse pesadesse $A+i$ ja $A+i+1$ salvestatud arve järjest i väärtuste 0, 1, 2, ..., $n-1$ korral. Kui võrdlemisel selgub, et pesas $A+i$ on suurem arv kui pesas $A+i+1$, siis vahetatakse neis arvud ära. Seejuures peetakse spetsiaalselt selleks ette nähtud pesades J ja K meeles, missuguse i väärtuse korral toimus esimene ja millise korral viimane arvude ümberpaigutamine (enne iga sammu sooritamist salvestatakse

pesasse J arv -1 ja pesasse K null). Pärast esimese sammu teostamist kehtivad järgmised seosed:

$$(A+i) \leqslant (A+(K)+1) \leqslant (A+(K)+2) \leqslant \ldots \leqslant (A+n), \tag{2}$$

kus $i = 0, 1, 2, \ldots, (K)$ ja

$$(A+0) \leqslant (A+1) \leqslant \ldots \leqslant (A+(J)-1) \tag{3}$$

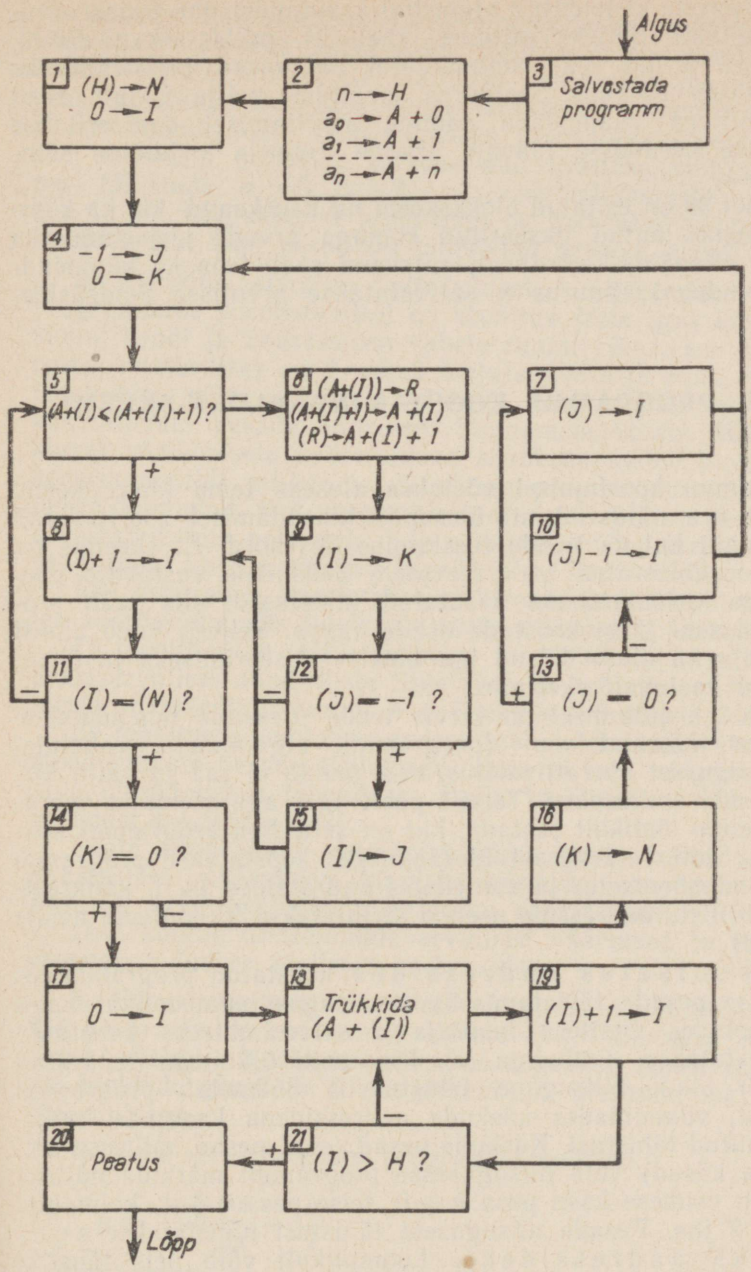
(seosel (3) on mõtet ainult siis, kui $(J) > 0$).

Kui $(K) = 0$, siis seosed (1) ja (2) langevad ühte ning järjestamine on lõppenud. Vastasel korral tuleb aga teha järjestamise teine samm. See toimub üldiselt samuti kui esimene, erinedes esimesest ainult selle poolest, et pesadesse $A+i$ ja $A+i+1$ salvestatud arvude võrdlemist teha i väärtuste $(J)_1 - 1$, $(J)_1$, $(J)_1 + 1$, ..., $(K)_1 - 1$ korral (siin $(J)_1$ ja $(K)_1$ tähistavad esimesel sammul pesadesse J ja K salvestatud arve). Ka pärast teist sammu kehtivad seosed (2) ja (3), kusjuures (K) ja (J) all tuleb mõista arve $(K)_2$ ja $(J)_2$, s. t. arve, mis pesadesse K ja J on salvestatud teise sammu sooritamisel. Peale selle ilmselt $(K)_2 \leqslant (K)_1 - 1$. Kui $(K)_2 = 0$, siis on järjestamine lõppenud. Vastasel korral tuleb teha kolmas samm. Kui $(K)_3 \neq 0$, siis neljas samm jne. Kolmas, neljas ja kõik järgnevad sammud on analoogilised teise sammuga (ainsaks erinevuseks on see, et $(J)_1$ ja $(K)_1$ asemel tuleb i väärtuste määramisel kasutada vaadeldavale sammule eelnenud sammul pesadesse J ja K salvestatud arve, ülejäänud seostes aga sellel sammul leitud suurusi (J) ja (K)).

Maksimaalselt saab $n+1$ arvu järjestamiseks vaadeldud algoritmi järgi kuluda n sammu.

Tabel 24

Samm \ Pesa	$A+0$	$A+1$	$A+2$	$A+3$	$A+4$	$A+5$	$A+6$	J	K
Algseis	1	7	2	5	4	6	3	-1	0
I samm	1	2	5	4	6	3	7	1	5
II samm	1	2	4	5	3	6	„	2	4
III samm	„	2	4	3	5	„	„	3	3
IV samm	„	„	3	4	„	„	„	2	2
V samm	„	2	3	„	„	„	„	-1	0
Lõppseis	1	2	3	4	5	6	7	-1	0


Algus
1
(H) → N
0 → I
2
n → H
a0 → A + 0
a1 → A + 1
an → A + n
3
Salvestada programm
4
−1 → J
0 → K
5
(A+(I)) ≤ (A+(I)+1)?
6
(A+(I)) → R
(A+(I)+1) → A+(I)
(R) → A+(I) + 1
7
(J) → I
8
(I)+1 → I
9
(I) → K
10
(J)−1 → I
11
(I) = (N) ?
12
(J) = −1 ?
13
(J) = 0 ?
14
(K) = 0 ?
15
(I) → J
16
(K) → N
17
0 → I
18
Trükkida (A + (I))
19
(I)+1 → I
20
Peatus
21
(I) > H ?
Lõpp

JOON. 30

Illustreerime kirjeldatud algoritmi kasutamist ühe lihtsa arvulise näite abil. Selleks esitame tabeli 24, milles on näidatud, missugused arvud iga sammu järel paiknevad järjestamiseks kasutatavates pesades. Seejuures kirjutame välja ainult need arvud, millega vaadeldava sammu sooritamisel opereeritakse (samadesse pesadesse jäävate arvude asemele kirjutame sümboli —„—).

Joonisel 30 on esitatud blokkskeem nii kaheksand- kui ka kümnendsüsteemis antud fikseeritud komaga arvude järjestamiseks (blokis 5 ettenähtud võrdlusoperatsiooni saab teha ka kümnendkahendarvudega). Suurus n salvestatakse arvutisse 8-ndsüsteemis.

§ 13. PROGRAMMI KOOSTAMINE BLOKK-SKEEMI JÄRGI

1. Programmi koostamisel võetakse aluseks tema blokk-skeem, milles on ära näidatud nii ülesande lahendamisel sooritatavad operatsioonid kui ka nende teostamise järjekord. Et skeemis on ühtlasi kooskõlastatud ka erinevatele blokkidele vastavate operatsioonide kirjeldamiseks kasutatud tähistused, siis saab programmi skeemi järgi koostada osade kaupa. Selleks tuleb ainult eraldi välja kirjutada käsud iga bloki realiseerimiseks ja ühendada need vastavalt skeemile.

Käskude kirjutamisel on tarvis teada ülesande lahendamisel kasutatava informatsiooni (programmi, algandmete, vahetulemuste) paigutust operatiivmälus, sest muidu ei saa ju välja kirjutada nende aadressiosi. Teiselt poolt pole aga võimalik operatiivmälu enne lõplikult jaotada, kui on selgunud programmi käskude arv. Sellest raskusest ülesaamiseks koostatakse programm algul nn. sümboolsetes ja suhtelistes aadressides (s. t. konkreetsete numbriliste aadresside asemel kirjutatakse käskudesse nende tähistused).

S ü m b o o l s e t e s a a d r e s s i d e s koostatud programmides kasutatakse pesade tähistamiseks samasugust sümboolikat nagu blokk-skeemides (näiteks muutuja a salvestamiseks kasutatav pesa tähistatakse sümboliga $\langle a \rangle$, konstandi 0,5 jaoks määratud pesa $\langle 0{,}5 \rangle$ jne.). Niisugune tähistusviis hõlbustab programmi koostamist, võimaldades käskude aadressidena kasutada blokkskeemis antud tähistusi. Käskude pesad (s. o. pesad, millesse salvestatakse käsud) võib niisugusesse programmi märkida näiteks järgmiselt: esimese käsu pesa $k+0$, teise pesa $k+1$, kolmanda pesa $k+2$ jne. Pesade niisuguseid tähistusi nimetatakse s u h t e l i s t e k s a a d r e s s i d e k s. Loomulikult võib neis tähelise liidetava k asemel kasutada mistahes tähte, arvulised liidetavad aga kirjutatakse suhtelistes aadressides alati kaheksandsüsteemis.

Suhtelisi aadresse saab programmeerimisel kasutada ka käskude aadressiosade märkimisel. Selleks tuleb ainult eelnevalt kokku leppida, missuguse suhtelise aadressiga me iga suuruse pesa tähistame. Reeglina moodustatakse programmi, algandmete ja arvutustulemuste salvestamiseks kasutatavate pesade suhtelised aadressid erinevate tähtede abil. Näiteks võib algandmete pesi tähistada $a+0$, $a+1, \ldots$; arvutustulemuste pesi $b+0$, $b+1$, ... Öeldakse, et programm on koostatud suhtelistes aadressides, kui kõik pesad on tähistatud eespool kirjeldatud viisil.

2. Arvutile «Ural-4» sümboolsetes aadressides koostatud programmide vormistamisel on soovitav jätta igas käsus operatsiooni koodi ja aadressiosa vahele ruumi t õ e l i s e (s. t. konkreetse numbrilise) a a d r e s s i kirjutamiseks ja enne mälujaotuse teostamist mitte märkida käskude ette nende salvestamiseks kasutatavate pesade aadresse. Suunamiskäskudes võib aga sel korral sümboolsete aadressidena kasutada nende blokkide järjekorranumbreid, millele vastavatesse programmiosadesse suunamised toimuvad. Kui programmi vormistamisel on neid soovitusi arvestatud, siis pole teda hiljem tarvis ümber kirjutada, valmis programmi on aga mugav lugeda ning kontrollida.

Kui programm koostatakse elektronarvutile «Ural-4» suhtelistes aadressides, siis tuleb tähistuste valimisel arvestada nõudeid:

1) topeltpesade tähistamiseks on lubatud kasutada ainult paarisarvuliste numbriliste liidetavatega suhtelisi aadresse, mille järele on märgitud tunnus 4 (näiteks $a+0\ 4$, $a+2\ 4, \ldots$);

2) topeltpesa suhtelise aadressiga on ära määratud ka teda moodustavate pesade tähistused, mida ei tohi programmis kasutada üheski teises tähenduses (näit. topeltpesa $a+2\ 4$ koosneb pesadest $a+2$ ja $a+3$).

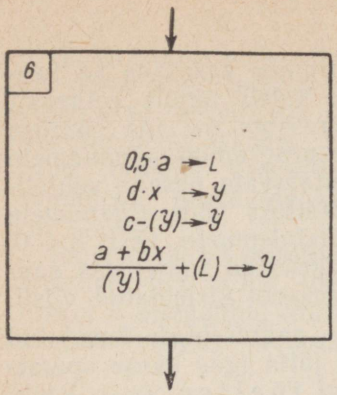
Peale programmi valmimist sümboolsetes (või suhtelistes) aadressides määratakse kindlaks käskude, konstantide, algandmete ja arvutustulemuste paigutus operatiivmälus. Selle alusel antakse kõigile aadressidele arvulised väärtused ja kirjutatakse programm ümber tõelistes aadressides.

3. Edasi selgitame arvutitele «Ural-4» ja «Minsk-22» määratud programmide koostamist näidete abil.[22] Lihtsuse mõttes piirdume esialgu näidetega, milles on tegemist ainult teatud bloki või blokkide grupi (s. t. mingi suurema ülesande osa) programmeerimisega.

N ä i d e 1. Koostame programmi suuruse

$$y = \frac{a+bx}{c-dx} - \frac{a}{2}$$

[22] Kõik arvutile «Minski-22» koostatud programmid on kasutatavad ka arvuti «Minsk-2» korral.



JOON. 31

väärtuse arvutamiseks (muutujad a, b, c, d ja x on liikuva komaga arvud). Seejuures eeldame, et vajalik informatsioon (programm, algandmed) salvestatakse operatiivmällu ning teisendatakse kahendsüsteemi toodud valemi kasutamisele eelneva töö käigus, ja et y väärtust on tarvis edasisteks arvutusteks. Seega antud juhul tuleb vaid välja kirjutada käsud üheainsa bloki (vt. joonis 31) realiseerimiseks.

Programmi koostamisel on soovitav valida operatsioonide sooritamise järjekord nii, et eelmise tehte tulemus võimaluse korral võtaks osa järgnevast tehtest. Seda arvestades tuleb näiteks alati murru nimetaja arvutada enne lugejat ja lahutatav enne vähendatavat. Ühtlasi peab aga jälgima, et ühe ning sama suuruse arvutamine (kui ta valemites esineb mitmes kohas) ei toimuks mitu korda. Neid soovitusi on silmas peetud ka joonisel 31 toodud blokis, kus on näidatud, et antud avaldise väärtuse leidmiseks tuleb käsud programmis kirja panna järgmiselt:

1) korrutise $0{,}5 \cdot a$ arvutamiseks ja salvestamiseks;
2) korrutise $d \cdot x$ leidmiseks ja salvestamiseks;
3) vahe $c - dx$ arvutamiseks ja salvestamiseks;
4) korrutise $b \cdot x$ leidmiseks (tulemust ei salvestata);
5) summa $a + bx$ arvutamiseks (tulemust ei salvestata);
6) jagatise $\frac{a+bx}{c-dx}$ arvutamiseks (tulemust ei salvestata);
7) vahe $\frac{a+bx}{c-dx} - \frac{a}{2}$ arvutamiseks ja salvestamiseks.

Samuti on blokist näha, et arvutuste käigus kasutatakse operatiivmälu kaheksat topeltpesa: $\langle a \rangle$, $\langle b \rangle$, $\langle c \rangle$, $\langle d \rangle$, $\langle x \rangle$, $\langle 0{,}5 \rangle$, L ja Y. Neid pesade tähistusi kasutame ka programmis sümboolsete aadressidena. Käskude pesad tähistame $k + 0$, $k + 1, \ldots$.

Elektronarvuti «Ural-4» jaoks sümboolsetes aadressides koostatud programm on koos selgitavate märkustega esitatud tabelis 25. Selles tabelis on käskude aadresside juurde märgitud ka topeltpesa tunnused $\beta = 4$. Need võib programmeerija soovi kor-

Pesa aadress	Käsk				Märkused
	Märk	Operatsiooni kood	Aadressiosa	β	
$k+0$		42	$<a>$	4	Arvu a tootmine summaatorisse: $S = a$
$k+1$		46	$<0{,}5>$	4	$(S) = 0{,}5 \cdot a$
$k+2$		56	L	4	Korrutise $0{,}5 \cdot a$ salvestamine pesasse L
$k+3$		42	$<d>$	4	$(S) = d$
$k+4$		46	$<x>$	4	$(S) = d \cdot x$
$k+5$		56	Y	4	Korrutise $d \cdot x$ salvestamine pesasse Y
$k+6$		42	$<c>$	4	$(S) = c$
$k+7$		43	Y	4	$(S) = c - d \cdot x$
$k+10$		56	Y	4	Vahe $c - dx$ salvestamine pesasse
$k+11$		42	$<b>$	4	$(S) = b$
$k+12$		46	$<x>$	4	$(S) = b \cdot x$
$k+13$		41	$<a>$	4	$(S) = a + b \cdot x$
$k+14$		47	Y	4	$(S) = \frac{a+bx}{c-dx}$
$k+15$		43	L	4	$(S) = \frac{a+bx}{c-dx} - 0{,}5 \cdot a$
$k+16$		56	Y	4	y väärtuse salvestamine pesasse Y

ral sümboolsetes aadressides (mitte kunagi aga tõelistes numbrilistes aadressides) märkimata jätta. Loomulikult ei tähenda see, et pole vaja teada, kas vastavat suurust salvestatakse pesas või topeltpesas.

Et saada suuruse y väärtuse arvutamiseks programmi lõplikul kujul, teeme mälujaotuse ja asendame kõikides käskudes sümboolsed aadressid tõelistega. Eraldame programmi salvestamiseks näiteks operatiivmälu pesad 0312 ÷ 0330, ülejäänud pesad valime aga järgmiselt:

$1204\,4 = <0{,}5>$ } Konstant

$1206\,4 = <a>$
$1210\,4 = <b>$
$1212\,4 = <c>$
$1214\,4 = <d>$
$1216\,4 = <x>$
} Algandmed

$3002\,4 = L$
$3004\,4 = Y$
} Tööpesad

Lõplikul kujul (s. o. nii tõelistes kui ka sümboolsetes aadressides) näeb meie programm sel juhul välja järgmiselt:

0312	42	1206	4	<a>	$0{,}5 \cdot a \to L$
0313	46	1204	4	<0,5>	
0314	56	3002	4	L	
0315	42	1214	4	<d>	$d \cdot x \to Y$
0316	46	1216	4	<x>	
0317	56	3004	4	Y	
0320	42	1212	4	<c>	$c - d \cdot x \to Y$
0321	43	3004	4	Y	
0322	56	3004	4	Y	
0323	42	1210	4	<b>	$\frac{a+bx}{c-dx} - \frac{a}{2} \to Y$
0324	46	1216	4	<x>	
0325	41	1206	4	<a>	
0326	47	3004	4	Y	
0327	43	3002	4	L	
0330	56	3004	4	Y	

Elektronarvuti «Minsk-22» jaoks on vastav programm sümboolsetes aadressides koos selgitustega esitatud tabelis 26. Selle koostamisel on eeldatud, et algandmed a, b, c, d, x ja konstant

Tabel 26

Pesa aadress	Käsk				Märkused
	Operatsiooni kood	Indeks	I aadress	II aadress	
$k+0$	35	00	<a>	<0,5>	$(S) = 0{,}5 \cdot a$
$k+1$	12	20	0000	L	Arvu $0{,}5 \cdot a$ salvestamine pesasse L
$k+2$	35	00	<d>	<x>	$(S) = d \cdot x$
$k+3$	22	20	0000	Y	Arvu $d \cdot x$ salvestamine pesasse Y
$k+4$	11	00	<c>	0000	Arvu c toomine summaatorisse: $(S) = c$
$k+5$	26	60	Y	Y	Vahe $c - d \cdot x$ arvutamine pesasse Y
$k+6$	35	00	<b>	<x>	$(S) = b \cdot x$
$k+7$	17	00	<a>	0000	$(S) = a + b \cdot x$
$k+10$	47	40	Y	0000	$(S) = \frac{a+bx}{c-dx}$
$k+11$	26	60	L	Y	Vahe $\frac{a+bx}{c-dx} - 0{,}5 \cdot a$ leidmine ja salvestamine

0,5 paigutatakse esimesse ferriitkuupi, tööpesad L ja Y aga teise feriitkuupi. Seepärast ongi pesades $k+1$ ja $k+3$ paiknevates käskudes näidatud, et nende esimene aadress märgib esimese kuubi, teine aga teise kuubi pesa (indeksina on kasutatud kahendarvu 010000, milles esimene kahendkoht on 0, teine aga 1).

Pesades $k+5$ ja $k+11$ on käsud, mille mõlemad aadressid vastavad teise ferriitkuubi pesadele (indeks 110 000); pesas $k+10$ paikneb käsk, mille esimene aadress vastab teise, teine aga esimese kuubi pesale (indeks 100 000). Ülejäänud käskude mõlemad aadressid vastavad esimese kuubi pesadele (indeksid 000 000). Indekspesi vaadeldavas programmis kasutatud pole (kõigi käskude indeksites on neli parempoolset kahendkohta nullid).

Tabelis 26 toodud programmi mõnedes käskudes (pesades $k+1$, $k+3$, $k+4$, $k+7$, $k+10$) on kasutatud tõelist aadressi 0000. Selles pesas säilitatakse alati arvu 0 — muud informatsiooni sinna viia ei saa (pesasse 0000 on arvude kirjutamine blokeeritud). Seetõttu saabki teatud käske (koodid 12 ja 22) kasutada arvude salvestamiseks summaatorist operatiivmällu, käske koodidega 11 ja 21 aga nende toomiseks pesadest summaatorisse jne.

Paigutame käsud arvuti «Minsk-22» operatiivmälu esimese ferriitkuubi pesadesse 0312 ÷ 0323, ülejäänud andmed aga pesadesse:

1204 =	<0,5>	Konstant ja algandmed (esimene ferriitkuup)
1205 =	<a>	
1206 =	<b>	
1207 =	<c>	
1210 =	<d>	
1211 =	<x>	
11206 =	L	Tööpesad (teine ferriitkuup)
11207 =	Y	

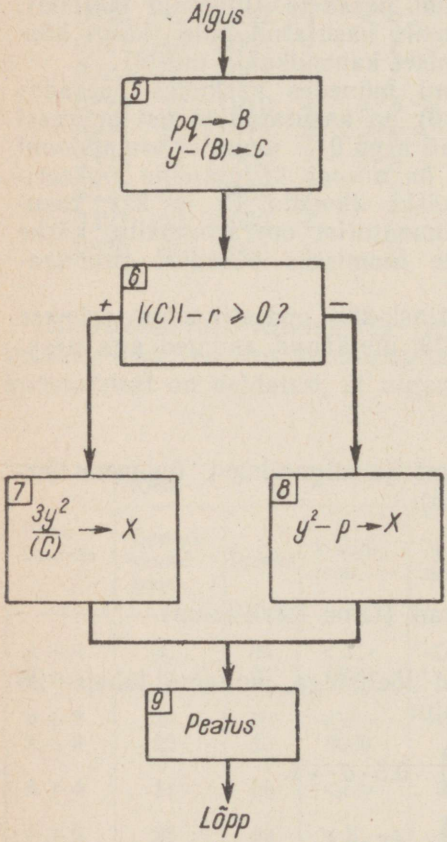
Asendades sümboolsed aadressid tõelistega, anname tabelis 26 esitatud programmile lõpliku kuju:

0312	35	00	1205	1204	$0{,}5 \cdot a \to L$
0313	12	20	0000	1206	
0314	35	00	1210	1211	$dx \to Y$
0315	22	20	0000	1207	
0316	15	00	1207	0000	$c - dx \to Y$
0317	11	60	1207	1207	
0320	35	00	1206	1211	$\frac{a+bx}{c-dx} - \frac{a}{2} \to Y$
0321	17	00	1205	0000	
0322	47	40	1207	0000	
0323	26	60	1206	1207	

Näide 2. Koostame programmi suuruse

$$x = \begin{cases} \dfrac{3y^2}{y - pq}, & \text{kui } |y - pq| \geqslant r \\ y^2 - p, & \text{kui } |y - pq| < r \end{cases}$$

väärtuse arvutamiseks, kui r, p, q, y tähistavad liikuva komaga arve ja $r > 0$. Täpsemalt, vaadeldavas näites kirjutame välja programmi joonisel 32 toodud blokkide grupi realiseerimiseks.



JOON. 32

Skeemist näeme, et ülesande lahendamisel on tarvis kolm tööpesa (B, C, X). Tegelikult võiks aga läbi saada ka üheainsa tööpesaga, sest arvu (B) kasutatakse ju ainult blokis 5 pesasse C salvestatava arvu leidmisel, hiljem teda tarvis pole. Seetõttu võiksime ka (C) kirjutada pesasse B. Analoogiline on olukord pesaga X. Kui tegemist on ülesandega, mille lahendamiseks tarviliku informatsiooni salvestamisel ilmselt ei kasutata ära operatiivmälu kõiki pesi, siis pole aga otstarbekohane töö-

pesade arvu kirjeldatud viisil vähendada: see muudab raskemaks programmi silumise (vt. VI peatükk).

Nii bloki 7 kui ka 8 realiseerimisel tuleb leida suurus y^2. Selle väärtuse arvutamise (koos operatiivmällu salvestamisega) võiks paigutada blokki 5 (siis tuleks y^2 arvutamiseks vajalikud käsud vaid üks kord välja kirjutada), kuid vaadeldaval juhul see programmi lihtsamaks ei muuda.

Koostame arvutile «Ural-4» määratud programmi algul suhtelistes aadressides. Selleks kirjutame järjest välja käsud kõikide blokkide realiseerimiseks, kusjuures algandmete ja tööpesade aadressid tähistame järgmiselt:

$<r> = a+0\ 4$, $<p> = a+2\ 4$, $<q> = a+4\ 4$,
$<y> = a+6\ 4$, $<3> = a+10\ 4$, $B = b+0\ 4$,
$C = b+2\ 4$, $X = b+4\ 4$.

Neid tähistusi kasutades saame programmi järgmisel kujul:

Blokk 5

$k+0$	42	$a+2\ 4$	$(S) = p$
$k+1$	46	$a+4\ 4$	$(S) = p \cdot q$
$k+2$	56	$b+0\ 4$	Arv $p \cdot q$ salvestatakse topeltpesasse $b+0\ 4$.
$k+3$	42	$a+6\ 4$	$(S) = y$
$k+4$	43	$b+0\ 4$	$(S) = y - p \cdot q$
$k+5$	56	$b+2\ 4$	Arv $y - p \cdot q$ salvestatakse topeltpesasse $b+2\ 4$.

Blokk 6

$k+6$	44	$a+0\ 4$	$(S) = \lvert y - p \cdot q \rvert - \lvert r \rvert$
$k+7$	21	$k+15$	Kui $(S) < 0$, siis antakse juhtimine pesas $k+15$ olevale käsule.

Blokk 7

$k+10$	42	$a+6\ 4$	$(S) = y$
$k+11$	46	$a+6\ 4$	$(S) = y^2$
$k+12$	46	$a+10\ 4$	$(S) = 3y^2$
$k+13$	47	$b+2\ 4$	$(S) = \dfrac{3y^2}{y - p \cdot q}$
$k+14$	22	$k+20$	Juhtimine antakse pesas $k+20$ olevale käsule.

Blokk 8

$k+15$	42	$a+6\ 4$	$(S) = y$
$k+16$	46	$a+6\ 4$	$(S) = y^2$
$k+17$	43	$a+2\ 4$	$(S) = y^2 - p$
$k+20$	56	$b+4\ 4$	x väärtuse salvestamine topeltpesasse $b+4\ 4$.

Blokk 9

$k+21$	37	0000 0	Arvuti peatamine.

Et suhtelistelt aadressidelt üle minna tõelistele, teeme mälu jaotuse, s. t. anname tähtedele *a*, *b* ja *k* konkreetsed arvulised väärtused. Paigutame programmi mälusse alates pesast 0300, algandmed alates pesast 0460, tööpesadeks aga eraldame mälu viimased pesad. Seega $a = 0460$, $b = 7770$ ja $k = 0300$. Asendades need väärtused programmis tähtede asemele (ja liites), saame:

0300	42	0462 4	Blokk 5
0301	46	0464 4	
0302	56	7770 4	
0303	42	0466 4	
0304	43	7770 4	
0305	56	7772 4	
0306	44	0460 4	Blokk 6
0307	21	0315 0	
0310	42	0466 4	Blokk 7
0311	46	0466 4	
0312	46	0470 4	
0313	47	7772 4	
0314	22	0320 0	
0315	42	0466 4	Blokk 8
0316	46	0466 4	
0317	43	0462 4	
0320	56	7774 4	
0321	37	0000 0	Blokk 9

Sama programmi koostamisel arvutile «Minsk-22» tähistame algandmete ning tööpesade aadressid järgmiselt: $\langle r \rangle = a + 0$, $\langle p \rangle = a + 1$, $\langle q \rangle = a + 2$, $\langle y \rangle = a + 3$, $\langle 3 \rangle = a + 4$, $B = b + 0$, $C = b + 1$, $X = b + 2$. Suuruse x väärtuse arvutamise programmi (suhtelistes aadressides) võib siis esitada kujul:

Blokk 5

$k+0$	35	00	$a+1$	$a+2$	$(S) = p \cdot q$
$k+1$	−30	00	$k+2$	$b+0$	Arvu $p \cdot q$ salvestamine pesasse $b+0$.
$k+2$	25	00	$b+0$	$a+3$	$(S) = y - p \cdot q$
$k+3$	76	00	0000	$b+1$	Arvu $y - p \cdot q$ salvestamine pesasse $b+1$.

Blokk 6

$k+4$	55	00	$a+0$	$b+1$	$(S)=\lvert y-p\cdot q\rvert-\lvert r\rvert$
$k+5$	—32	00	$k+6$	$k+12$	Kui $(S)<0$, siis antakse juhtimine pesas $k+12$ olevale käsule.

Blokk 7

$k+6$	35	00	$a+3$	$a+3$	$(S)=y^2$
$k+7$	37	00	$a+4$	0000	$(S)=3y^2$
$k+10$	46	00	$b+1$	$b+2$	Arvutatakse $\frac{3y^2}{y-pq}$ pesasse $b+2$.
$k+11$	—30	00	$k+14$	0000	Juhtimine antakse pesas $k+14$ olevale käsule.

Blokk 8

$k+12$	35	00	$a+3$	$a+3$	$(S)=y^2$
$k+13$	26	00	$a+1$	$b+2$	Arvutatakse y^2-p pesasse $b+2$.

Blokk 9

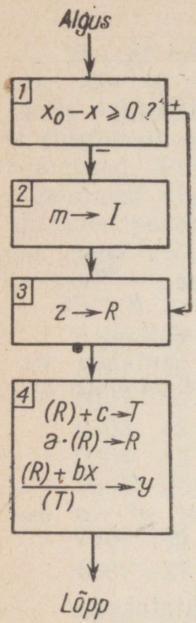
$k+14$	—00	00	0000	0000	Arvuti peatatakse.

Üleminek suhtelistelt aadressidelt tõelistele toimub samuti nagu arvutile «Ural-4» koostatud programmis. Võttes näiteks $a=0460$, $b=7770$ ja $k=0300$ (kasutame ainult esimese ferriitkuubi pesi), saame:

0300	35	00	0461	0462	Blokk 5
0301	—30	00	0302	7770	
0302	25	00	7770	0463	
0303	76	00	0000	7771	
0304	55	00	0460	7771	Blokk 6
0305	—32	00	0306	0312	
0306	35	00	0463	0463	Blokk 7
0307	37	00	0464	0000	
0310	46	00	7771	7772	
0311	—30	00	0314	0000	
0312	35	00	0463	0463	Blokk 8
0313	26	00	0461	7772	
0314	—00	00	0000	0000	Blokk 9

Näide 3. Koostame programmi muutuja $y=\frac{az+bx}{z+c}$ väärtuse arvutamiseks, kus

$$z=\begin{cases} dx^2+e, & \text{kui } x\leqslant x_0; \\ gx^2+h, & \text{kui } x>x_0. \end{cases}$$



JOON. 33

Toodud valemites esinevate muutujate a, b, c, d, e, g, h, x ja x_0 väärtused olgu liikuva komaga arvud. Programmi blokk-skeem on esitatud joonisel 33. Tema koostamisel on arvestatud, et muutuja z väärtuse arvutamiseks tuleb mõlemal juhul (nii juhul, kui $x \leqslant x_0$ kui ka $x > x_0$) sooritada ühed ning samad operatsioonid. On ju avaldiste $dx^2 + e$ ja $gx^2 + h$ väärtuste arvutamisel ainsaks erinevuseks see, et esimese avaldise korral tuleb kasutada muutujate d ja e, teise korral aga g ja h väärtusi. Seetõttu on võimalik nende avaldiste väärtusi arvutada ühe ning sama käskude rühma abil (blokk 3). Selleks tuleb ainult bloki 3 neid käske, mille täitmisel on tarvis opereerida kas muutujate d ja e või g ja h väärtustega, arvutuste käigus vastavalt ümber adresseerida. Blokis 2 salvestataksegi ümberadresseerimiskonstant $m = \langle g \rangle - \langle d \rangle = \langle h \rangle - \langle e \rangle$ pesasse I (algul viime sellesse pesasse nulli). Arvutile «Ural-4» programmi koostamisel muudame käskude aadressi käsu 30 I 0 abil, «Minsk-22» programmis aga kirjutame muudetavate aadressidega käskudesse indeksi kohale indekspesa I aadressi.

Kui tähistada[23] $\langle a \rangle = a + 0\ 4$, $\langle b \rangle = a + 2\ 4$, $\langle c \rangle = a + 4\ 4$, $\langle d \rangle = a + 6\ 4$, $\langle e \rangle = a + 10\ 4$, $\langle g \rangle = a + 12\ 4$, $\langle h \rangle = a + 14\ 4$, $\langle x_0 \rangle = a + 16\ 4$, $\langle x \rangle = a + 20\ 4$, $I = b + 0$, $R = b + 2\ 4$, $T = b + 4\ 4$ ja $Y = b + 6\ 4$, siis saame arvutile «Ural-4» suhtelistes aadressides programmi järgmisel kujul:

Blokk 1

$k+2$	42	$a+16\ 4$	$(S) = x_0$
$k+3$	43	$a+20\ 4$	$(S) = x_0 - x$
$k+4$	21	$k+7\ 4$	Kui $(S) \geqslant 0$, siis antakse juhtimine pesas $k+7$ olevale käsule.

Blokk 2

$k+5$	20	0004 0	$(S) = 4 \cdot 2^{-18}$
$k+6$	16	$b+0$ 0	Arvu $4 \cdot 2^{-18}$ salvestamine pesasse $b+0$.

[23] Nendes tähistustes esinevad tähed a ja b kahes täiesti erinevas tähenduses. Et selline praktika on üsna laialt levinud, siis pole seda ka käesolevas õpikus vältida püütud.

Blokk 3

$k+7$	42	$a+20$	4	$(S)=x$
$k+10$	46	$a+20$	4	$(S)=x^2$
$k+11$	30	$b+0$	0	Kui $(b+0)=0$, siis arvutatakse dx^2.
$k+12$	46	$a+6$	4	$(b+0)=4\cdot 2^{-18}$ korral aga gx^2.
$k+13$	30	$b+0$	0	Kui $(b+0)=0$, siis arvutatakse dx^2+e.
$k+14$	41	$a+10$	4	$(b+0)=4\cdot 2^{-18}$ korral aga gx^2+h.
$k+15$	56	$b+2$	4	z väärtuse salvestamine pesasse $b+2$ 4.

Blokk 4

$k+16$	41	$a+4$	4	$(S)=z+c$.
$k+17$	56	$b+4$	4	Arvu $z+c$ salvestamine pesasse $b+4$ 4.
$k+20$	42	$b+2$	4	$(S)=z$
$k+21$	46	$a+0$	4	$(S)=az$
$k+22$	56	$b+2$	4	Arvu az salvestamine pesasse $b+2$ 4.
$k+23$	42	$a+2$	4	$(S)=b$
$k+24$	46	$a+20$	4	$(S)=bx$
$k+25$	41	$b+2$	4	$(S)=az+bx$
$k+26$	47	$b+4$	4	$(S)=\frac{az+bx}{z+c}$
$k+27$	56	$b+6$	4	Saadud y väärtuse salvestamine pesasse $b+6$ 4.

Enne selle programmi tegelikku kasutamist peab pesasse $b+0$ olema salvestatud arv null. Selleks tuleb programmi lisada näiteks käsud:

$k+0$	20	0000	0
$k+1$	16	$b+0$	0

Arvuti «Minsk-22» korral paigutame algandmed ja vahetulemused operatiivmälu pesadesse järgmiselt:

$(a+0)=a$, $(a+1)=b$, $(a+2)=c$, $(a+3)=d$,
$(a+4)=e$, $(a+5)=g$, $(a+6)=h$, $(a+7)=x_0$,
$(a+10)=x$, $R=b+0$, $T=b+1$, $Y=b+2$

ja kasutame indekspesana I pesa 0017. Ümberadresseerimiskonstandi 00 00 0002 0000 paigutame pesasse $c+0$. Elektronarvutile «Minsk-22» suhtelistes aadressides koostatud programm tuleb siis järgmine:

Blokk 1

$k+1$	25 00 $a+10$ $a+7$	$(S) = x_0 - x$
$k+2$	−32 00 $k+4$ $k+3$	Kui $(S) \geqslant 0$, siis antakse juhtimine pesas $k+4$ olevale käsule, muidu pesas $k+3$ olevale käsule.

Blokk 2

$k+3$	−10 00 $c+0$ 0017	Pesast $c+0$ kirjutatakse arv pesasse 0017.

Blokk 3

$k+4$	35 00 $a+10$ $a+10$	$(S) = x^2$
$k+5$	37 17 $a+3$ 0000	Kui $(0017) = 0$, siis
$k+6$	16 17 $a+4$ $b+0$	$(S) = (b+0) = dx^2 + e$, $(0017) =$ 00 00 0002 0000 korral aga $(S) = (b+0) = gx^2 + h$.

Blokk 4

$k+7$	16 00 $a+2$ $b+1$	$(S) = (b+1) = z + c$
$k+10$	34 00 $a+0$ $b+0$	$(S) = (b+0) = az$
$k+11$	35 00 $a+1$ $a+10$	$(S) = bx$
$k+12$	17 00 $b+0$ 0000	$(S) = az + bx$
$k+13$	46 00 $b+1$ $b+2$	$(S) = (b+2) = \dfrac{az+bx}{z+c}$

Enne toodud programmi tegelikku kasutamist tuleb indekspesasse 0017 kirjutada arv null. Selleks võib programmile lisada käsu

$k+0$ −10 00 0000 0017.

§ 14. TSÜKLITE PROGRAMMEERIMINE

1. Arvutusprotsessi, mille käigus tuleb korduvalt rakendada üht ning sama arvutuseeskirja, nimetatakse tsükliliseks, sellise protsessi korratavat etappi aga t s ü k l i k s. Tsükliliste arvutusprotsesside osatähtsus on väga suur. Peaaegu kõik matemaatikast tuntud lahendusmeetodid on oma olemuselt tsüklilised. Veelgi tähtsamaks osutub arvutusprotsessi tsüklilisus programmeerimise seisukohalt: kui mingi ulatuslikuma ülesande lahendusalgoritmi ei õnnestu esitada tsüklilisena, siis seda algoritmi arvutil enamasti realiseerida ei saagi (vastava programmi väljakirjutamine võtaks liialt kaua aega ning see ei mahuks arvuti salvestusseadmetesse). Sellepärast ongi tsüklite programmeerimine üks põhilisemaid programmeerimisvõtteid.

Programmeerimisel eristatakse kahte liiki tsükleid. Esimesse liiki kuuluvad sellised tsüklid, mille kordamiste arv on ette teada (s. t. ei sõltu vaadeldava tsükli teostamisel saadavatest arvutustulemustest). Näiteks arvutusprotsess, millega leitakse funktsiooni

$$f(x) = \frac{bx^2 + a}{a^2 + b^2}$$

tabel argumendi x väärtustel $x_1, x_2, \ldots, x_n$ (a ja b on konstandid) koosneb tsüklist, mille kordamiste arv on n: tuleb ju tabeli arvutamisel valemit $f(x_i) = (bx_i^2 + a) : (a^2 + b^2)$ kasutada n korda ($i = 1, 2, \ldots, n$).

Teise liigi moodustavad niisugused tsüklid, kus kordamiste arv ei ole ette teada, vaid mida korratakse seni, kuni saadakse teatud tingimusi rahuldavad arvutustulemused. Näiteks suuruse $y = \sqrt{z}$ väärtuse arvutamisel Newtoni iteratsioonivalemi järgi

$$y_{i+1} = \frac{1}{2}\left(\frac{z}{y_i} + y_i\right)$$

(kus $i = 0, 1, 2 \ldots$ ja alglähend y_0 ning täpsus ε on ette antud) tuleb lähislahendeid y_i leida seni, kuni saadakse lähendid, mis rahuldavad seost

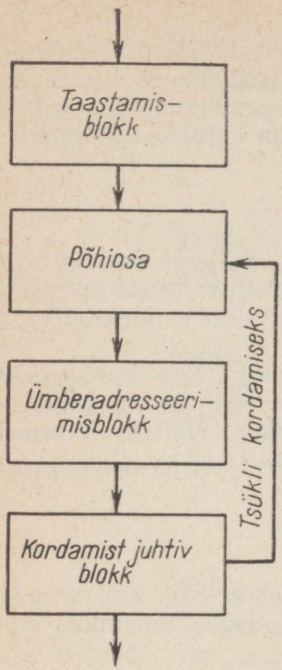
$$|y_{i+1} - y_i| < \varepsilon.$$

Muutuvat suurust, mille väärtuse alusel tsükli iga kordumise järel otsustatakse, kas temas ettenähtud arvutusi on tarvis veel kord teostada või mitte, nimetatakse selle t s ü k l i p a r a m e e t r i k s. Tingimust, mille täidetuse korral tsükkel lõpetatakse, nimetatakse selle t s ü k l i l õ p u t i n g i m u s e k s. Esimese näitena toodud arvutusprotsessis on tsükli parameetriks indeks i ja lõputingimuseks seos $i > n$; teises näites on aga nendeks vastavalt suurus $p = |y_{i+1} - y_i|$ ja seos $p < \varepsilon$.

Tsüklilise arvutusprotsessi programmeerimisel koostatakse ka programm tsüklilisena, s. t. sellisena, milles teatud käskude rühma tuleb täita korduvalt. Kui tsükli kordamiste arv n on ette teada, siis saaks muidugi programmi koostada ka nii, et kirjutatakse n korda järjest välja käskude rühm, mis realiseerib tsüklis ettenähtud arvutused. Selline programmi koostamise moodus ei ole aga otstarbekohane, sest siis võtab üksnes programmi kirjutamine peaaegu niisama palju aega kui vastavate arvutuste tegemine aritmomeetriga.

2. Kõik tsüklilised programmid on sarnase struktuuriga, koosnedes üldiselt neljast suhteliselt iseseisvast osast (vt. joonis 34).

Tsüklilise programmi p õ h i o s a moodustab blokk või blokkide grupp, millele vastavate käskudega realiseeritakse ühes tsüklis ettenähtud arvutused. Et neid arvutusi tehakse igal kordami-



JOON. 34

sel üldiselt uute lähteandmetega, siis tuleb paljudes tsüklilistes programmides enne tsükli iga kordamist muuta tema põhiosa mõnede käskude aadresse. Ümberadresseerimisbloki moodustavadki need käsud, mille täitmisega tagatakse vajalike muudatuste tegemine põhiosa käskudes.

Kui tsüklis ettenähtud arvutuste kordamisel kasutatavad lähteandmed leitakse tsükli eelmise rakendamise tulemusena (kusjuures edasiste arvutuste käigus meid huvitavad ainult tsükli *n*-kordse täitmise teel saadavad lõpptulemused), siis põhiosa käskude igakordse ümberadresseerimise asemel on otstarbekam salvestada lähteandmed kindlatesse pesadesse, kus nad töö käigus asenduvad vahepealsete arvutustulemustega. Tsüklilistes programmides, mille koostamisel on kasutatud niisugust võtet, ümberadresseerimisblokk puudub.

Taastamisbloki moodustavad käsud, mille täitmise tulemusena viiakse ettenähtud pesadesse tsüklilise arvutusprotsessi organiseerimiseks vajalikud andmed (näiteks tsükli parameetri algväärtus) ja taastatakse põhiosa käskude esialgne kuju. Taastamisblokk on vajalik eeskätt siis, kui tsüklilise programmiosa poole tuleb veel kord pöörduda.

Kordamist juhtivas blokis kontrollitakse tsükli lõputingimust ja sõltuvalt selle tulemusest kas antakse juhtimine põhiosa esimesele käsule tsükli veelkordseks rakendamiseks või lõpetatakse tsüklilise programmiosa täitmine (toimub väljumine tsüklist). Nimelt kui tsükli parameetri vaadeldav väärtus ei rahulda lõputingimust, siis korratakse tsüklis ettenähtud arvutusi. Vastupidisel juhul on aga tsüklit vajalik arv kordi täidetud ja ülesande lahendamist jätkatakse tsüklilisele programmiosale järgnevate blokkide realiseerimisega.

Tutvustame nüüd tsüklite organiseerimise ja käskude ümberadresseerimise mitmesuguseid võtteid ning võimalusi.

3. Tsüklid, mille kordamiste arv ei ole ette teada. Selgitame niisuguste tsüklite programmeerimist järgmise kahe näite abil.

Näide 1. Ülesannete lahendamiseks kasutatakse tavaliselt ligikaudseid meetodeid, mis võimaldavad tulemusi saada teatava täpsusega. Tulemustelt nõutav täpsus on sealjuures enamasti koos ülesandega ette antud, näiteks lubatava vea ülemmäärana.

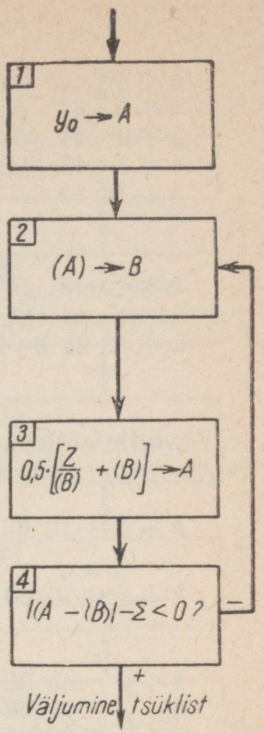
Ligikaudsete meetodite tüüpilisemateks esindajateks on mitmesugused iteratsioonimeetodid, mis seisnevad otsitavate suuruste lähisväärtuste järkjärgulises parandamises. Näiteks võrrandi $x = f(x)$ lahendamisel nn. hariliku iteratsioonimeetodiga lähtutakse otsitava lahendi mingist alglähendist x_0 ja leitakse järgmised lähendid $x_1, x_2, \ldots$ valemist

$$x_{i+1} = f(x_i) \qquad (i = 0, 1, 2, \ldots).$$

Sellist tüüpi arvutuseeskirjade realiseerimisel esinevaid tsükleid nimetatakse iteratsioonitsükliteks. Nende tsüklite kordamiste arv pole ette teada, vaid sõltub näiteks saadud lähendite jada koondumise kiirusest ja nõutavast täpsusest.

Iteratsioonitsüklite programmeerimisel kasutatakse täpsuse hinnangut p ja tsükli lõputingimusena seost $|p| < \varepsilon$, kus ε tähistab tulemustes lubatud vea ülemmäära. Koostame näitena programmi suuruse $y = \sqrt{z}$ väärtuse arvutamiseks iteratsioonivalemi

$$y_{i+1} = 0{,}5\left(\frac{z}{y_i} + y_i\right) \qquad (i = 0, 1, 2, \ldots)$$



JOON. 35

järgi, kui alglähend y_0 ning täpsus ε on ette antud. Programmi koostamisel eeldame, et $z > 0$ ja alglähend y_0 on valitud nii, et saadav lähendite jada $y_0, y_1, y_2, \ldots$ koondub. Tsükli lõputingimusena kasutame seost

$$|y_{i+1} - y_i| - \varepsilon < 0.$$

Programmi blokk-skeem on toodud joonisel 35. Selles on blokk 1 taastamisblokiks, blokid 2 ja 3 moodustavad programmi põhiosa ja blokis 4 kontrollitakse tsükli lõputingimust. Kõik uued lähendid y_{i+1} $(i = 0, 1, 2, \ldots)$ arvutame pesasse A, kusjuures eelmise lähendi y_i säilitame pesas B.

Arvutile «Ural-4» sümboolsetes aadressides koostatud programm on järgmine:

$k+0$	02	$\langle y_0 \rangle$	4	Blokk 1. Alglähend y_0 salvestatakse
$k+1$	16	A	4	tööpesasse A.
$k+2$	02	A	4	Blokk 2. Enne lähendi y_{i+1} arvutamist
$k+3$	16	B	4	salvestatakse eelmine lähend y_i pesast A pesasse B.

$k+4$	42	$\langle z\rangle$	4	} Blokk 3. Lähendi y_{i+1} arvutamine ja salvestamine pesasse A.
$k+5$	47	B	4	
$k+6$	41	B	4	
$k+7$	46	$\langle 0{,}5\rangle$	4	
$k+10$	56	A	4	
$k+11$	43	B	4	} Blokk 4. Kui $\lvert y_{i+1}-y_i\rvert-\varepsilon\geqslant 0$, siis antakse juhtimine bloki 2 esimesele käsule. Vastupidisel juhul on y väärtus nõutud täpsusega arvestatud (pesas A) ja toimub väljumine tsüklist (täidetakse pesas $k+14$ olev käsk).
$k+12$	44	$\langle\varepsilon\rangle$	4	
$k+13$	21	$k+2$	4	

Arvutile «Minsk-22» sümboolsetes aadressides kirjutatult tuleb vastav programm aga järgmine:

$k+0$	-10	00	$\langle y_0\rangle$	A } Blokk 1. Algkähend y_0 salvestatakse tööpesasse A.
$k+1$	-10	00	A	B } Blokk 2. Lähend y_i salvestatakse pesast A pesasse B.
$k+2$	45	00	B	$\langle z\rangle$ } Blokk 3. Lähendi y_{i+1} arvutamine ja salvestamine pesasse A.
$k+3$	17	00	B	0000
$k+4$	36	00	$\langle 0{,}5\rangle$	A
$k+5$	27	00	B	0000 } Blokk 4. Kui $\lvert y_{i+1}-y_i\rvert-\varepsilon\geqslant 0$, siis antakse juhtimine bloki 2 esimesele käsule. Vastupidisel juhul on y väärtus nõutud täpsusega leitud (pesas A) ja täidetakse käsk pesast $k+10$.
$k+6$	57	00	$\langle\varepsilon\rangle$	0000
$k+7$	-32	00	$k+1$	$k+10$

Näide 2. Koostame programmi integraali

$$\int_a^b f(x)\,dx$$

väärtuse arvutamiseks trapetsvalemiga

$$\int_a^b f(x)\,dx \approx h\left[\frac{y_0+y_n}{2}+\sum_{i=1}^{n-1} y_i\right],$$

kus $y_i=f(x_i)$, $x_0=a$, $x_i=x_{i-1}+h$ ja $n=\dfrac{b-a}{h}$, ning eeldatakse, et integreerimissamm $h>0$ on ette antud.

Trapetsvalemis esineva summa

$$s = \frac{y_0 + y_n}{2} + \sum_{i=1}^{n-1} y_i$$

arvutamisel võib kasutada valemeid

$$s_i = s_{i-1} + y_i,$$

kus $i = 1, 2, \ldots;\ x_i \leqslant b - h$ ning

$$s_0 = \frac{y_0 + y_n}{2}.$$

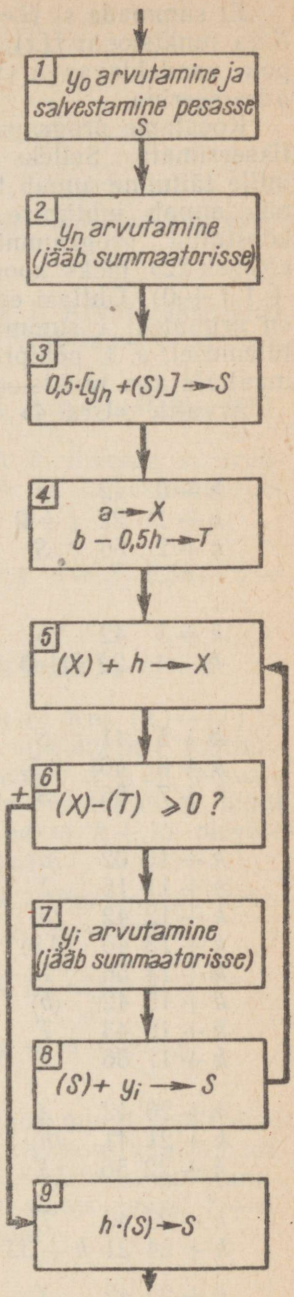
Programmi koostamisel kasutame tsükli parameetrina argumenti x (jättes jaotuspunktide arvu n üldse leidmata). Et argumendi väärtus x_{n-1}, mis leitakse seosest $x_{n-1} = x_{n-2} + h$, võib arvutamisel tekkivate ümardamisvigade kuhjumise tõttu osutuda veidi suuremaks kui arv $(b - h)$, siis ei saa me tsükli lõputingimusena kasutada seost $x_i > b - h$ (liidetav y_{n-1} võib sel korral summast $\sum_{i=1}^{n-1} y_i$ välja jääda).

Tavaliselt kasutatakse niisuguste tsüklite programmeerimisel lõputingimusena seost

$$x_i \geqslant b - \frac{h}{2} \quad \left(\text{ehk } x_i - b + \frac{h}{2} \geqslant 0\right).$$

Programmi blokk-skeem on esitatud joonisel 36. Blokkides 1—4 toimuvad ettevalmistused blokkide 5—8 kasutamiseks: pesasse S arvutatakse summa s_0, tsükli parameetri algväärtus a ja abiarv $(b - 0{,}5h)$ aga salvestatakse vastavalt pesadesse x ja T. Blokid 5, 7 ja 8 moodustavad vaadeldava tsüklilise programmi põhiosa, millele vastava käskude rühma korduva täitmise tulemusena saadaksegi summa

$$s_{n-1} = \frac{y_0 + y_n}{2} + \sum_{i=1}^{n-1} y_i.$$



JOON. 36

Et summade s_i ($i = 1, 2, \ldots$) salvestamiseks kasutatakse pesa S ja funktsiooni $f(x)$ väärtusi y_i operatiivmälus ei säilitata, siis pole vaadeldavas arvutusprotsessis tarvis põhiosa käske ümber adresseerida.

Koostame programmi nii, et funktsiooni $f(x)$ täpne kuju jääb fikseerimata. Selleks asendame programmis käskude rühma, mille täitmine annab funktsiooni $f(x)$ väärtuse, üheainsa käsuga, mis annab juhtimise pesasse $f+1$ (elektronarvutile «Ural-4» koostatud programmis kasutame selleks käsku 22 $f+0$ 4, «Minsk-22» jaoks koostatud programmis aga käsku —31 00 $f+$ $+1$ $f+0$). Ühtlasi eeldame, et juhtimise andmisel pesasse $f+1$ on argument x summaatoris; nimetatud käskude rühma täitmise tulemusel, s. t. põhiprogrammi tagasipöördumisel paiknegu summaatoris aga funktsiooni väärtus $f(x)$.

Arvutile «Ural-4» saame seega järgmise programmi:

$k+0$	42	$\langle a \rangle$	4	Blokk 1. Argumendi $x = a$ toomine summaatorisse, suunamine funktsiooni $f(x)$ väärtuste arvutamise programmi ja y_0 salvestamine pesasse S.
$k+1$	22	$f+0$	4	
$k+2$	56	S	4	
$k+3$	42	$\langle b \rangle$	4	Blokk 2. Argumendi väärtuse b toomine summaatorisse ja suunamine $f(x)$ väärtuste arvutamise programmi.
$k+4$	22	$f+0$	4	
$k+5$	41	S	4	Blokk 3. Summa s_0 arvutamine ja salvestamine pesasse S.
$k+6$	46	$\langle 0{,}5 \rangle$	4	
$k+7$	56	S	4	
$k+10$	02	$\langle a \rangle$	4	Blokk 4. $a \to X$, $b - 0{,}5h \to T$
$k+11$	16	X	4	
$k+12$	42	$\langle h \rangle$	4	
$k+13$	46	$\langle 0{,}5 \rangle$	4	
$k+14$	56	T	4	
$k+15$	42	$\langle b \rangle$	4	
$k+16$	43	T	4	
$k+17$	56	T	4	
$k+20$	42	X	4	Blokk 5. $(X)+h \to X$
$k+21$	41	$\langle h \rangle$	4	
$k+22$	56	X	4	
$k+23$	43	T	4	Blokk 6. $(X)-(T) \geqslant 0$?
$k+24$	21	$k+32$	4	
$k+25$	42	X	4	Blokk 7. Funktsiooni väärtuse y_i arvutamine.
$k+26$	22	$f+0$	4	

$k+27$	41	S	4		Blokk 8. $(S)+y_i \to S$
$k+30$	56	S	4		
$k+31$	22	$k+20$	0		
$k+32$	42	S	4		Blokk 9. $h \cdot (S) \to S$
$k+33$	46	$\langle h \rangle$	4		
$k+34$	56	S	4		

Arvuti «Minsk-22» korral tuleb sama sümboolsetes aadressides kirjutatud programm järgmine:

$k+0$	—10	00	$\langle a \rangle$	S	Blokk 1. Arvutatakse $f(a)$ ja salvestatakse pesasse S.
$k+1$	—31	00	$f+1$	$f+0$	
$k+2$	—30	00	$k+3$	S	
$k+3$	—10	00	$\langle b \rangle$	T	Blokk 2. Arvutatakse $f(b)$.
$k+4$	—31	00	$f+1$	$f+0$	
$k+5$	17	00	S	0000	Blokk 3. Summa s_0 arvutamine ja salvestamine pesasse S.
$k+6$	36	00	$\langle 0{,}5 \rangle$	S	
$k+7$	—10	00	$\langle a \rangle$	X	Blokk 4. $a \to X$, $0{,}5h - b \to T$
$k+10$	35	00	$\langle 0{,}5 \rangle$	$\langle h \rangle$	
$k+11$	26	00	$\langle b \rangle$	T	
$k+12$	14	00	$\langle h \rangle$	X	Blokk 5. $(X)+h \to X$
$k+13$	17	00	T	0000	Blokk 6. Kui $(X)-(T) \geqslant 0$, siis antakse juhtimine pesas $k+21$ olevale käsule, vastupidisel juhul aga pesas $k+15$ olevale käsule.
$k+14$	—32	000	$k+21$	$k+15$	
$k+15$	—10	00	X	0000	Blokk 7. Funktsiooni väärtuse y_i arvutamine.
$k+16$	—31	00	$f+1$	$f+0$	
$k+17$	16	00	S	S	Blokk 8. $(S)+y_i \to S$
$k+20$	—30	00	$k+12$	0000	
$k+21$	34	00	$\langle h \rangle$	S	Blokk 9. $h \cdot (S) \to S$

Kui esitatud programme soovitakse kasutada mingi konkreetse integraali arvutamisel, siis tuleb täiendavalt koostada vastav standardne alamprogramm funktsiooni $f(x)$ väärtuste arvutamiseks ja paigutada see arvuti operatiivmällu alates pesast $f+0$. Koostame näitena alamprogrammi funktsiooni

$$f(x) = \frac{3x^3+4}{x^2+7}$$

väärtuste arvutamiseks. Arvuti «Ural-4» jaoks tuleb vastav programm järgmine:

$f+0$	00	0000	0	Väljumispesa [24].
$f+1$	56	R	4	Argumendi x väärtus salvestatakse pesasse R.
$f+2$	46	R	4	Arvutatakse x^2 ja salvestatakse pesasse V.
$f+3$	56	V	4	
$f+4$	41	$\langle 7\rangle$	4	Arvutatakse nimetaja x^2+7 väärtus ja salvestatakse pesasse C.
$f+5$	56	C	4	
$f+6$	42	$\langle 3\rangle$	4	Arvutatakse lugeja $3x^3+4$ väärtus.
$f+7$	46	V	4	
$f+10$	46	R	4	
$f+11$	41	$\langle 4\rangle$	4	
$f+12$	47	C	4	Leitakse $f(x)$ väärtus.
$f+13$	22	$f+0$	0	Antakse juhtimine väljumispesasse $f+0$.

Sama funktsiooni väärtuste arvutamiseks koostatud alamprogramm omandab arvuti «Minsk-22» korral kuju:

$f+0$	00	00	0000	0000	Väljumispesa.
$f+1$	−30	00	$f+2$	R	Argumendi x väärtus salvestatakse pesasse R.
$f+2$	36	00	R	V	Arvutatakse x^2 ja salvestatakse pesasse V.
$f+3$	16	00	$\langle 7\rangle$	C	Arvutatakse x^2+7 ja salvestatakse pesasse C.
$f+4$	35	00	$\langle 3\rangle$	R	Arvutatakse lugeja $3x^3+4$ väärtus.
$f+5$	37	00	V	0000	
$f+6$	17	00	$\langle 4\rangle$	0000	
$f+7$	47	00	C	0000	Leitakse $f(x)$ väärtus.
$f+10$	−30	00	$f+0$	0000	Antakse juhtimine väljumispesasse $f+0$.

[24] Integraali arvutamise programmist antakse juhtimine pesas $f+1$ olevale käsule käsuga 22 $f+0$ 4, mille täitmise tulemusena kirjutatakse pesasse $f+0$ suunamiskäsk 22 $k+j+1$ 0 (eeldades, et käsk 22 $f+0$ 4 paikneb pesas $k+j$, antud juhul $j=1$ või 4 või 26).

4. Selliste tsüklite programmeerimist, mille kordamiste arv on ette teada, lihtsustab arvuti «Ural-4» korral spetsiaalsete tsüklioperatsioonide 25 ja 24 olemasolu. Tutvumegi mõningate programmeerimisvõtetega, mis võimaldavad operatsioonide 25 ja 24 abil organiseerida põhiosa käskude korduvat täitmist ning teha vajalikke ümberadresseerimisi.

Paiknegu tsüklilise programmi põhiosa pesades $k+1$, $k+2$, ..., $k+j$. Operatsioonide 25 ja 24 abil saavutatakse selle programmiosa n-kordne täitmine sel teel, et pesasse $k+0$ kirjutatakse käsk 25 m β, kus

$$m = \begin{cases} n-1, & \text{kui } \beta = 0; \\ 2(n-1), & \text{kui } \beta = 4; \end{cases}$$

pesasse $k+j+1$ aga käsk 24 $k+1$ 0.

Seega käsud 25 m β ja 24 $k+1$ 0 paigutatakse programmi järgmise skeemi järgi:

$$\left.\begin{array}{l} k+0 \quad 25 \quad m \quad \beta \\ k+1 \quad . \quad . \quad . \quad . \quad . \quad . \\ \quad . \quad . \quad . \quad . \quad . \quad . \quad . \\ k+j \quad . \quad . \quad . \quad . \quad . \quad . \\ k+j+1 \quad 24 \quad k+1 \quad 0 \end{array}\right\} n \text{ korda täidetav programmiosa.}$$
$$k+j+2 \quad . \quad . \quad . \quad . \quad .$$

Niisuguse struktuuriga programm täidetakse järgmiselt.

Käsu 25 m β täitmise tulemusena kirjutatakse arv m tsükliregistrisse.[25] Edasi täidetakse pesades $k+1$, $k+2$, ..., $k+j$ olevad käsud, kusjuures enne täitmist muudetakse kõigi nende käskude aadresse, mille ette on programmis kirjutatud miinusmärk. Nimelt, enne niisuguste *negatiivsete* käskude täitmist vähendab arvuti (käskude registris) nende aadresse tsükliregistris oleva arvu võrra. Seega miinusmärgiga varustatud käske täidetakse esimesel korral arvu m võrra vähendatud aadressidega.

Käsu 24 $k+1$ 0 täitmisel kontrollitakse, kas tsükliregistris on arv null. Kui tsükliregistri sisu on nullist erinev, siis vähendatakse teda sammu h võrra: kui tunnus $\beta = 0$, siis $h = 0001$; kui aga $\beta = 4$, siis $h = 0002$. Seejärel annab käsk 24 $k+1$ 0 juhtimise pesasse $k+1$ (pesades $k+1$, $k+2$, ..., $k+j+1$ paikneva programmiosa teistkordseks täitmiseks). Et tsükliregistris on nüüd arv $m-h$, siis täidetakse sel korral kõiki miinusmärgiga varustatud käske arvu $m-h$ võrra vähendatud aadressidega. Seega on kõigi ümberadresseeritavate käskude aadressid teistkordsel täitmisel arvu h võrra suuremad kui esimesel täitmisel. Analoogiliselt korratakse pesades $k+1$, ..., $k+j+1$ paik-

[25] Koos arvuga m kirjutatakse tsükliregistrisse ka tunnuse β väärtus.

nevate käskude täitmist seni, kui käsu 24 $k+1$ 0 täitmisel selgub, et tsükliregistris on arv null. Sel korral käsk 24 $k+1$ 0 annab juhtimise pesas $k+j+2$ olevale käsule (tsükliregistrisse jääb arv null).

Toome nüüd näitena mõned lihtsad tsüklilised programmid, mille koostamisel on kasutatud operatsioone 25 ja 24.

Näide 3. Koostame programmi funktsiooni $y=x^{18}$ väärtuse arvutamiseks, kui $(a+0)=x$ ja y tuleb paigutada topeltpesasse $b+0$ 4. Selle programmi võib esitada kujul

$k+0$ 25 0020 0
$k+1$ 42 $a+0$ 4
$k+2$ 46 $a+0$ 4
$k+3$ 24 $k+2$ 0
$k+4$ 56 $b+0$ 4

Pesadesse $k+2$ ja $k+3$ kirjutatud käske täidetakse 17 korda (käsu 25 aadressina on kasutatud arvu $m=0020_8=16_{(10)}$, $\beta=0$). Käskude ümberadresseerimist selle programmi täitmisel ei toimu (ühelgi käsul pole miinusmärki).

Näide 4. Koostame programmi polünoomi $p(x)=a_0x^n+a_1x^{n-1}+\ldots+a_{n-1}x+a_n$ väärtuste arvutamiseks kohal $x=x_0$. Kasutame Horneri skeemi, mille kohaselt polünoomi väärtus arvutatakse valemitest

$$p_i=p_{i-1}x_0+a_i,$$

kus $i=1, 2, \ldots, n$; $p_0=a_0$ ja $p_n=p(x_0)$.

Paigutades suurused pesadesse järgmiselt: $(c+0\ 4)=a_0$, $(c+2\ 4)=a_1$, $\ldots$, $(c+2n\ 4)=a_n$, $(d+0\ 4)=x_0$, $(d+2\ 4)=p_n$, saame programmi:

$k+0$ 42 $c+0$ 4
$k+1$ 25 $2n-2$ 4
$k+2$ 46 $d+0$ 4
$k+3$ —41 $c+2n$ 4
$k+4$ 24 $k+2$ 0
$k+5$ 56 $d+2$ 4

Vahepealseid arvutustulemusi p_i $(i=1, 2, \ldots, n-1)$ selle programmi täitmisel operatiivmällu ei salvestata, vaid säilitatakse summaatoris. Pesadesse $k+2$, $k+3$ ja $k+4$ paigutatud käske täidetakse n korda, kusjuures pesas $k+3$ olevat käsku —41 $c+2n$ 4 täidetakse iga kord uue aadressiga. Nimelt on esimesel täitmisel tema aadressiks arv $c+2$ 4, teisel arv $c+4$ 4, $\ldots$, $\ldots$, n-ndal täitmisel aga $c+2n$ 4.

Näide 5. Topeltpesadesse $a+0$ 4, $a+2$ 4, $\ldots$, $a+2n$ 4 salvestatud fikseeritud komaga arvude teisendamiseks kümnend-

kahendsüsteemist kahendsüsteemi võib kasutada järgmist programmi:

$k+0$ 25 $2n$ 4
$k+1$ —35 $a+2n$ 4
$k+2$ —16 $a+2n$ 4
$k+4$ 24 $k+1$ 0

5. Ümberadresseerimise sammuks h nimetatakse kaheksandarvu, mille võrra tsükli iga kordamise eel on tarvis muuta programmis käskude aadresse. Operatsioone 25 ja 24 eespool kirjeldatud viisil kasutades saab käske seega ümber adresseerida kas sammuga $h=1$ (käsus 25 $m\beta$ võetakse $\beta=0$) või $h=2$ (käsus 25 $m\beta$ võetakse $\beta=4$), kusjuures tsükli iga järgneva realiseerimise korral ümberadresseeritavaid käske täidetakse sammu h võrra suurema aadressiga kui eelmisel täitmisel. Operatsioone 25 ja 24 saab aga kasutada ka selliste tsükliliste programmide koostamisel, milles on tarvis käske ümber adresseerida suurema sammuga kui 2 ($h>2$).

Paiknegu tsüklilise programmi põhiosa pesades $k+1$, $k+2$, ..., $k+j$. Selle programmiosa n-kordset täitmist koos miinusmärgiga varustatud käskude ümberadresseerimisega (samm $h>2$) on võimalik organiseerida sel teel, et paigutame käsud 25 $m\beta$, kus $m=h(n-1)$ ning

$$\beta=\begin{cases}0, \text{ kui } h \text{ on paaritu arv,}\\ 4, \text{ kui } h \text{ on paarisarv,}\end{cases}$$

ja 24 a 0 programmi järgmise skeemi järgi:

$k+0$	25	m	β	
$k+1$	—	—	—	
$k+2$	—	—	—	
—	—	—	—	
$k+j$	—	—	—	n korda täidetav programmiosa.
$k+j+1$	24	$k+j+2$	0	
$k+j+2$	24	$k+j+3$	0	
—	—	—	—	
$k+j+r+1$	24	$k+j+r$	0	
$k+j+r$	24	$k+1$	0	

Selle skeemi järgi koostatud programmis täidetakse pesadesse $k+1$, $k+2$, ..., $k+j+r$ kirjutatud käske n korda, kusjuures igal täitmisel vähendatakse tsükliregistris olevat arvu sammu h võrra. Selleks sooritatakse r korda järjest operatsiooni 24, kusjuures

$$r=\begin{cases}h, & \text{kui käsus } 25m\ \beta \text{ tunnus } \beta=0,\\ \frac{h}{2}, & \text{kui käsus } 25m\ \beta \text{ tunnus } \beta=4.\end{cases}$$

Tsükliregistris olevat arvu kasutatakse käskude ümberadresseerimisel täpselt samuti kui sammu $h = 2$ (või $h = 1$) korral.

Näide 6. Olgu tarvis liita topeltpesades $c + 0$ 4, $c + 6$ 4, $c + 14$ 4, ..., $c + 6n$ 4 olevad arvud. Kasutades esitatud skeemi (antud juhul $h = 6$, $m = 6n$ ja $r = 3$), saame programmi:

$k + 0$	20	0000	0
$k + 1$	25	$6n$	4
$k + 2$	—42	$c + 6n$	4
$k + 3$	24	$k + 4$	0
$k + 4$	24	$k + 5$	0
$k + 5$	24	$k + 2$	0

Sammu h suurte väärtuste korral pole enam otstarbekohane tsükliregistri sisu muuta operatsioonide 24 abil. Sel korral võib tsüklilise programmi koostada näiteks järgmise skeemi alusel:

$k + 0$	25	m	β	
$k + 1$	—	—	—	
—	—	—	—	
$k + j$	—	—	—	
$k + j + 1$	27	$k + j + 3$	0	n korda täidetav programmiosa.
$k + j + 2$	30	$a + 0$	0	
$k + j + 3$	00	0000	0	
$k + j + 4$	24	$k + 1$	0	

Sõltumata sellest, kas ümberadresseerimise samm h on paaris- või paarituarvuline, võib vaadeldavas skeemis pesasse $k + 0$ paigutada kas käsu 25 m 0 või 25 m 4.[26] Kui kasutatakse käsku 25 m 0, siis tuleb pesasse $a + 0$ salvestada abiarv $-(h-1)\cdot 2^{-18}$, käsu 25 m 4 puhul aga abiarv $-(h-2)\cdot 2^{-18}$.

Pesadesse $k + 1$, $k + 2$, ..., $k + j$ paigutatud programmi põhiosa igakordse täitmise järel muudavad pesades $k + j + 1$, ..., $k + j + 4$ olevad käsud tsükliregistri sisu sammu h võrra. Nimelt, pesas $k + j + 1$ oleva käsuga 27 $k + j + 3$ 0 salvestatakse põhiosa esimese täitmise järel pesasse $k + j + 3$ käsk 25 m β, teise täitmise järel käsk 25 $m - h$ β jne., n-nda täitmise järel käsk 25 0000 β, sest pesades $k + j + 2$ ja $k + j + 3$ paiknevate käskude igakordse täitmise tulemusena väheneb tsükliregistri sisu kas $h - 1$ (pesas $a + 0$ on arv $-(h-1)\cdot 2^{-18}$) või $h - 2$ (pesas $a + 0$ on arv $-(h-2)\cdot 2^{-18}$) ühiku võrra, järgmise käsu 24 $k + 1$ 0 täitmisega aga veel vastavalt kas ühe või kahe ühiku võrra.

[26] Siin eeldame muidugi, et $h > 1$, sest $h = 1$ korral käsku 25 m 4 kasutada ei tohi.

Põhiosa viimase (s. t. n-nda) täitmise järel kirjutatakse pesasse $k+j+3$ käsk 25 0000 β, mis (modifitseerituna pesas $k+j+2$ oleva käsu 30 $a+0$ 0 mõjul) täidetakse tegelikult kujul 24 q β, kus

$$q = \begin{cases} 10\,000 - (h-1), \text{ kui pesas } a+0 \text{ on arv } -(h-1)\cdot 2^{-18}, \\ 10\,000 - (h-2), \text{ kui pesas } a+0 \text{ on arv } -(h-2)\cdot 2^{-18}. \end{cases}$$

Et tsükliregistris on aga null, siis see käsk 24 q β annab juhtimise pesas $k+j+4$ olevale käsule 24 $k+1$ 0, mis omakorda annab juhtimise jälle järgmises pesas (s. o. pesas $k+j+5$) olevale käsule.

Näide 7. Koostame programmi pesades $b+0$, $b+12$, $b+24$ ja $b+36$ paiknevate arvude ümbersalvestamiseks pesadesse $c+0$, $c+12$, $c+24$ ja $c+36$. Antud juhul $n=4$, $m=36$, $h=12$ ja $(a+0)=-00\ 0010\ 0$ ning programm tuleb järgmine:

$k+0$	25	0036	4
$k+1$	−02	$b+36$	0
$k+2$	−16	$c+36$	0
$k+3$	27	$k+5$	0
$k+4$	30	$a+0$	0
$k+5$	00	0000	0
$k+6$	24	$k+1$	0

Operatsioonide 25 ja 24 abil on võimalik käskude ümberadresseerimist organiseerida ka nii, et tsükli iga järgneva realiseerimise ajal täidetakse miinusmärgiga varustatud käsud sammu h võrra väiksema aadressiga kui eelmisel korral. Järgnevalt ongi antud skeem, mille alusel koostatud programmide täitmisel ümberadresseerimine toimub kas sammuga $h=-2$ või $h=-1$:

$k+0$	25	m	β	
$k+1$	−25	m	β	
$k+2$	—	—	—	
—	—	—	—	n korda täidetav programmiosa.
$k+j$	—	—	—	
$k+j+1$	−25	m	β	
$k+j+2$	24	$k+1$	0	

Sõltuvalt pesasse $k+j+1$ kirjutatud käsu −25 m β tunnusest täidetakse pesades $k+2$, $k+3$, ..., $k+j$ paiknevaid miinusmärgiga varustatud käske tsükli iga järgneva täitmise ajal kas ühe ($\beta=0$) või kahe ($\beta=4$) ühiku võrra väiksema aadressiga kui eelmisel täitmisel. (Programmi kirjutatakse ikka ümberadresseeritavate käskude maksimaalsed aadressid.) Pesadesse $k+0$ ja $k+1$ paigutatud käskude tunnustest sammu h suurus ei sõltu.

Näide 8. Koostame programmi summa $s = \sum_{i=0}^{n} a_i b_{n-i}$ arvutamiseks, kui

$(c+0\ 4) = a_0$, $(c+2\ 4) = a_1$, ..., $(c+2n\ 4) = a_n$
$(d+0\ 4) = b_0$, $(d+2\ 4) = b_1$, ..., $(d+2n\ 4) = b_n$.

Summa s salvestame topeltpesasse $e+0$ 4:

$k+0$	20	0000	0
$k+1$	16	$e+0$	4
$k+2$	25	$2n$	4
$k+3$	—42	$c+2n$	4
$k+4$	—25	$2n$	4
$k+5$	—46	$d+2n$	4
$k+6$	41	$e+0$	4
$k+7$	56	$e+0$	4
$k+10$	—25	$2n$	4
$k+11$	24	$k+3$	0

6. Tsüklilise programmi põhiosa käskude n-kordset täitmist ja ümberadresseerimist on arvutis «Ural-4» võimalik organiseerida ka ilma operatsioonideta 25 ja 24. Kui tsükli kordamiste arv n on ette teada, siis on programmi põhiosa korduvat täitmist lihtne juhtida nn. loendaja abil, kusjuures käskude ümberadresseerimiseks kasutatakse operatsiooni 30.

Kordamiste arvu loendajana kasutatakse tavaliselt üht pesa (tähistame tähega L), milles säilitatavat arvu muudetakse programmi põhiosa iga täitmise ajal kindla arvu (nn. loendaja sammu) võrra. Kui loendajas L olev arv (L) omandab teatava etteantud väärtuse m, siis antakse juhtimine tsüklilisest programmiosast välja.

Loendaja sammuks on enamasti sobiv võtta ümberadresseerimise samm h. See võimaldab arvu (L) rakendada ka käskude aadresside muutmisel. Nimelt saab siis käsku ümber adresseerida sel teel, et kirjutame tema ette käsu 30 L 0.

Tutvumegi nüüd tsükliliste programmide mõningate niisuguste skeemidega, kus põhiosa korduvat täitmist juhitakse loendajate abil.

Skeem 1.

$k+0$	20	0000	0	Nulli salvestamine loendajasse.
$k+1$	16	L	0	
$k+2$	—	—	—	Programmi põhiosa.
—	—	—	—	
$k+j$	—	—	—	
$k+j+1$	02	$\langle h \rangle$	0	Loendajas oleva arvu muutmine sammu h võrra.
$k+j+2$	01	L	0	
$k+j+3$	16	L	0	

$k+j+4$	14	$\langle m \rangle$	0	Kui $(L) \neq m$, siis juhtimine antakse pesas $k+2$ olevale käsule, $(L) = m$ korral antakse juhtimine pesas $k+j+6$ olevale käsule.
$k+j+5$	21	$k+2$	0	

Selles skeemis muutub (L) sammuga h (h võib olla kas positiivne või negatiivne) lõigus $[0, m = nh]$. Kui $h > 0$, siis tuleb (käsuga 30 L 0) ümberadresseeritavates käskudes kirjutada muutuvate aadresside minimaalsed väärtused, kusjuures põhiosa igal järgneval rakendamisel neid käske täidetakse sammu h võrra suuremate aadressidega kui eelmisel korral. Kui $h < 0$, siis käsuga 30 L 0 ümberadresseeritavad käsud kirjutatakse programmi oma maksimaalsete aadressidega (neid käske täidetakse põhiosa igal järgneval rakendamisel sammu $|h|$ võrra väiksemate aadressidega kui eelmisel korral).

Kui sammu h väärtus on teada juba programmi koostamisel (ja seda pole tarvis muuta ülesande lahendamise käigus), siis võib skeemis 1 pesasse $k+j+1$ kirjutatud käsu 02 $\langle h \rangle$ 0 asendada kas käsuga 20 h 0 (kui $h > 0$) või 20 h 4 (kui $h < 0$).

Näide 9. Olgu arvud x_i $(i = 0, 1, 2, \ldots, n)$ paigutatud järjestikuste aadressidega pesadesse $a+i$ indeksite kasvamise järjekorras. Koostame programmi selle pesa aadressi määramiseks, kus paikneb suurim arvudest x_i. Tähistades otsitava aadressi salvestamiseks kasutatava pesa aadressi tähega S, saame programmi (milles $h = 2^{-18}$ ja $m = n \cdot 2^{-18}$):

$k+0$	20	0000	0	Nulli salvestamine loendajasse L ja pesasse S.
$k+1$	16	L	0	
$k+2$	16	S	0	
$k+3$	30	S	0	Kui $x_{(S)} \geqslant x_{(L)} + 1$, siis antakse juhtimine pesas $k+12$ olevale käsule, vastupidisel juhul aga pesas $k+10$ olevale käsule.
$k+4$	02	$\langle x_0 \rangle$	0	
$k+5$	30	L	0	
$k+6$	03	$\langle x_1 \rangle$	0	
$k+7$	21	$k+12$	4	
$k+10$	02	L	0	Loendajas L olev arv kirjutatakse pesasse S.
$k+11$	16	S	0	
$k+12$	20	0001	0	Loendajas oleva arvu suurendamine ühe võrra.
$k+13$	01	L	0	
$k+14$	16	L	0	
$k+15$	14	$\langle m \rangle$	0	Kui $(L) \neq m$, siis antakse juhtimine pesas $k+3$ olevale käsule, vastupidisel juhul aga pesas $k+17$ olevale käsule.
$k+16$	21	$k+3$	0	

$k+17$	20	a	0	Arvu $x_{(S)}$ salvestamiseks kasutatud pesa aadressi arvutamine ja salvestamine pesasse S.
$k+20$	01	S	0	
$k+21$	16	S	0	

Skeem 2.

$k+0$	20	0000	0	Nulli saatmine loendajasse.
$k+1$	16	L	0	
$k+2$	14	$\langle m \rangle$	0	Kui $(L)=m$, siis antakse juhtimine pesasse $k+j+4$, vastupidisel juhul täidetakse pesas $k+4$ olev käsk.
$k+3$	21	$k+j+4$	4	
$k+4$	—	—	—	Programmi põhiosa.
—	—	—	—	
$k+j$	—	—	—	
$k+j+1$	02	$\langle h \rangle$	0	Loendajas oleva arvu muutmine sammu h võrra.
$k+j+2$	01	L	0	
$k+j+3$	22	$k+1$	0	

Skeemi 2 kasutatakse täpselt samuti nagu skeemi 1. Sisuliselt erinevad need skeemid teineteisest selle poolest, et esimeses läheb tingimuse $(L)=m$ korral juhtimine pesas $k+j+6$ olevale käsule, teises saab aga pesas $k+3$ oleva suunamiskäsuga anda juhtimise ükskõik missuguses pesas paiknevale käsule.

Skeem 3.

$k+0$	02	$\langle m \rangle$	0	Arvu m saatmine loendajasse.
$k+1$	16	L	0	
$k+2$	—	—	—	Programmi põhiosa.
—	—	—	—	
$k+j$	—	—	—	
$k+j+1$	02	$\langle h \rangle$	0	Loendajas oleva arvu muutmine sammu h võrra
$k+j+2$	01	L	0	
$k+j+3$	21	$k+1$	β	Tsükli lõputingimuse kontrollimine.

Kui $h>0$, siis tuleb võtta $m=-hn$. Tsükli lõputingimusena kasutatakse sel juhul seost $(L) \geqslant 0$, pesasse $k+j+3$ tuleb kirjutada käsk 21 $k+1$ 0 ja kanda käsuga 30 L 0 ümberadresseeritavate käskude maksimaalsed aadressid programmi sammu h võrra suurendatutena. Kui programmi põhiosa rakendatakse esimest korda, siis täidetakse neid käske minimaalsete aadressidega, igal järgneval korral aga sammu h võrra suuremate aadressidega kui eelmisel. Viimasel (s. t. n-ndal) täitmisel on nende käskude aadressid programmi kirjutatutest h võrra väiksemad.

Kui $h < 0$, siis tuleb võtta $m = -h(n-1)$. Tsükli lõputingimusena kasutatakse nüüd seost $(L) < 0$, pesasse $k+j+3$ tuleb kirjutada käsk 21 $k+1$ 4 ja kanda käsuga 30 L 0 ümberadresseeritavad käsud programmi oma minimaalsete aadressidega. Neid käske täidetakse põhiosa igal järgneval rakendamisel sammu h võrra väiksemate aadressidega kui eelmisel korral.

Näide 10. Paiknegu polünoomi

$$p(x) = a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x + a_n$$

kordajad a_i vastavalt topeltpesades $c+2i$ 4 $(i = 0, 1, \ldots, n)$. Koostame programmi polünoomi $p(x)$ tuletise

$$p'(x) = b_0x^{n-1} + b_1x^{n-2} + \ldots + b_{n-2}x + b_{n-1}$$

kordajate arvutamiseks valemitest

$$b_j = (n-j)a_j \quad (j = 0, 1, 2, \ldots, n-1),$$

kusjuures arvud b_j salvestame vastavalt topeltpesadesse $d+2j$ 4.

Selle ülesande korral tuleb valida $h = 2 \cdot 2^{-18} = 2^{-17}$, $m = -2n \cdot 2^{-18} = -n \cdot 2^{-17}$. Olgu $(e+0) = h$, $(e+1) = m$ ja $(e+3) = 0000104$, kus pesasse $e+3$ on paigutatud liikuva komaga arvu $n-j = q \cdot 2^{17}$ järk kaheksandsüsteemis. Pesa $e+2$ kasutame arvu $n-j$ mantissi q salvestamiseks.

$k+0$	02	$e+1$	0	Arvu $n \cdot 2^{-17}$ salvestamine $e+2$ (q esimene väärtus).
$k+1$	10	$k+0$	0	
$k+2$	16	$e+2$	0	
$k+3$	02	$e+1$	0	Arvu m salvestamine loendajasse.
$k+4$	16	L	0	
$k+5$	42	$e+2$	4	$b_j = (n-j)a_j$ arvutamine ja salvestamine pesasse $d+2j$ 4.
$k+6$	30	L	0	
$k+7$	46	$c+2n$	4	
$k+10$	30	L	0	
$k+11$	56	$d+2n$	4	
$k+12$	20	0001	4	q arvutamine uue j väärtuse korral.
$k+13$	01	$e+2$	0	
$k+14$	16	$e+2$	0	
$k+15$	02	$e+0$	0	Loendajas oleva arvu suurendamine $2 \cdot 2^{-18}$ võrra.
$k+16$	01	L	0	
$k+17$	21	$k+4$	0	

Skeem 4.

Kui tsükli kordamiste arv n on 2 ja käsu 30 L 0 abil pole vaja ümber adresseerida, siis võib vastava tsüklilise programmi koostada järgmise skeemi alusel:

$k+0$	—	—	—	} Programmi põhiosa.
$k+1$	—	—	—	
—	—	—	—	
$k+j$	—	—	—	
$k+j+1$	20	0000	0	} Pesas L olev arv asendatakse vastandarvuga.
$k+j+2$	03	L	0	
$k+j+3$	16	L	0	
$k+j+4$	21	$k+0$	0	} Kui $(L)<0$, siis antakse juhtimine pesas $k+0$ olevale käsule, vastupidisel juhul aga pesas $k+j+5$ olevale käsule.

Skeemi 4 kasutamisel tuleb loendajaks L valida selline pesa, millesse enne vaadeldava programmi täitmist on salvestatud mingi positiivne arv (näiteks võib selleks kasutada vaadeldavasse programmiosasse mittekuuluva käsu pesa). Pesades $k+j+1$ kuni $k+j+4$ olevate käskude esmakordse täitmise tulemusena muudetakse arv (L) negatiivseks ning juhtimine antakse pesas $k+0$ olevale käsule. Samade käskude teistkordsel täitmisel aga taastatakse loendaja L esialgne väärtus (muudetakse (L) tagasi positiivseks) ja juhtimise saab pesas $k+j+5$ olev käsk.

7. Arvuti «Minsk-22» puhul lihtsustab tsükliliste programmide koostamist spetsiaalse tsüklioperatsiooni (käsk —20 i a_1 a_2) ja indekspesade (operatiivmälu esimese ferriitkuubi pesad 0001 ÷ 0017) olemasolu. Nende abil on tsüklilistes programmides võimalik organiseerida nii põhiosa käskude n-kordset täitmist kui ka teha vajalikke ümberadresseerimisi.

Indekspesade abil muudetakse käskude aadresse järgmiselt. Käsu kandmisel operatiivmälust käskude registrisse liidetakse tema aadressiosale automaatselt arv, mis paikneb selle käsu indeksis näidatud aadressiga indekspesa kohtadel 13 ÷ 36. Seejuures ülekanded, mis tekivad kohtadel 13 ja 25, liidetakse vastavalt kohtadel 7 ja 8 olevatele kahendnumbritele (ülekandeid, mis tekivad liitmisel kohtadel 7 ja 8, summas ei arvestata). Seega, kui käsu esimene (teine) aadress vastas enne ümberadresseerimist operatiivmälu esimese ferriitkuubi pesale, siis pärast ühega võrduvat ülekannet kohalt 13 (kohalt 25), mis tekkis ümberadresseerimisel, vastab see aadress teise ferriitkuubi pesale ja vastupidi. Selliselt modifitseerituna käsk täidetaksegi, kusjuures operatiivmälus säilib tema esialgne kuju.

Illustreerime öeldut ühe konkreetse näitega. Olgu pesas 0003 arv 0047 0672 0003 ja pesas 0101 käsk —10 63 7240 1032. Selles käsus on kasutatud indeksina kahendarvu 110 011, mille neli viimast kahendkohta (arv 0011) näitavad, et tema aadressiosale liidetakse enne täitmist pesa 0003 kohtadel 13 ÷ 36 olev arv 0672 0003. Seega täidetakse vaadeldav käsk kujul: —10 23 0132 1035, pesas 0101 säilib aga käsk —10 63 7240 1032.

Tsüklilised programmid, milles põhiosa korduvat täitmist juhitakse käsu —20 i a_1a_2 abil, koostatakse järgmise skeemi alusel:

$k+0$	—10 00 $a+0$ i	Taastamisblokk.
$k+1$	— — — —	Programmi põhiosa.
—	— — — —	
$k+j$	— — — —	
$k+j+1$	—20 i $k+1$ $a+1$	Kordamist juhtiv blokk.

Selleks et pesades $k+1$, $k+2$, ..., $k+j+1$ paiknevat programmiosa täidetaks n korda, tuleb enne programmi põhiosa esmakordset täitmist viia indekspesa i kohtadele $0 \div 12$ arv $m = (n-1) \cdot 2^{-12}$, kohtadele $13 \div 36$ aga arv, millega tuleb modifitseerida pesa i abil ümberadresseeritavate käskude aadresse põhiosa esmakordsel täitmisel. Pesas $k+0$ paikneva käsu (—10 00 $a+0$ i) täitmisel salvestataksegi niisugune konstant pesast $a+0$ indekspesasse i ja alles seejärel täidetakse pesades $k+1$, $k+2$, ..., $k+j+1$ paiknev programmiosa. Kui enne käsu (—20 i $k+1$ $a+1$) täitmist on pesa i kohtadel $0 \div 12$ nullist erinev arv, siis selle käsu täitmise tulemusena vähendatakse pesa i nendel kohtadel paiknevat arvu ühe ühiku (s. o. $1 \cdot 2^{-12}$) võrra ja antakse juhtimine pesades $k+1$, $k+2$, ..., $k+j+1$ oleva programmiosa veelkordseks täitmiseks pesas $k+1$ paiknevale käsule. Samal ajal liidetakse automaatselt pesa i kohtadel $13 \div 36$ olevale arvule pesa $a+1$ samadel kohtadel säilitatav konstant (ümberadresseerimise samm). Näiteks kui enne käsu —20 i $k+1$ $a+1$ täitmist on pesas i arv 0047 0004 0016 ja pesas $a+1$ konstant 0203 0002 0007, siis pärast selle käsu täitmist jääb pesasse i arv 0046 0006 0025.

Pesades $k+1$, $k+2$, ..., $k+j+1$ paiknevat programmiosa täidetakse n korda, sest alles n-ndal korral on enne käsu —20 i $k+1$ $a+1$ täitmist pesa i kohtadel $0 \div 12$ arv null. Seetõttu käsu —20 i $k+1$ $a+1$ täitmisega antakse nüüd juhtimine pesas $k+j+2$ olevale käsule, kusjuures pesas i paiknevat arvu ei muudeta.

Illustreerime vaadeldud skeemi kasutamist mõne lihtsa näitega.

Näide 11. Koostame programmi funktsiooni $y = x^{18}$ väärtuse arvutamiseks, kui $(b+0) = x$ ja y tuleb salvestada pesasse $b+1$.

Järgnevas programmis on kasutatud indekspesa 0002, kusjuures pesas $a+0$ olgu arv 0020 0000 0000. Indekspesa 0002 abil käskude aadresse seekord ei muudeta.

$k+0$ —10 00 $a+0$ 0002
$k+1$ —10 00 $b+0$ 0000
$k+2$ 36 00 $b+0$ $b+1$
$k+3$ —20 02 $k+2$ 0000

Pesadesse $k+2$ ja $k+3$ kirjutatud käske täidetakse selles programmis 17 korda.

Näide 12. Koostame programmi polünoomi

$$p(x) = a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x + a_n$$

väärtuse arvutamiseks kohal $x = x_0$, kasutades Horneri skeemi (vt. näide 4).

Paigutades vajalikud suurused pesadesse järgmiselt: $(a+0) = m$ 0001 0000, $(a+1) =$ 0000 0001 0000, $(c+0) = a_0$, $(c+1) = a_1$, ..., $(c+n) = a_n$, $(d+0) = x_0$, $(d+1) = p(x_0)$ ja kasutades indekspesa 0010, saame programmi:

$k+0$ —10 00 $a+0$ 0010
$k+1$ —10 00 $c+0$ $d+1$
$k+2$ 35 00 $d+0$ $d+1$
$k+3$ 16 10 $c+0$ $d+1$
$k+4$ —20 10 $k+2$ $a+1$

Pesadesse $k+2$, $k+3$ ja $k+4$ paigutatud käskude rühma täidetakse selles programmis n korda ($m = (n-1)\cdot 2^{-12}$). See-juures esimesel täitmisel on pesas $k+3$ paikneva käsu esimeseks aadressiks $c+1$, teisel $c+2$, ..., n-ndal täitmisel $c+n$ (pesas $k+3$ oleva käsu teist aadressi ei muudeta).

Näide 13. Koostame programmi, mille täitmise tulemusena salvestatakse arvud pesadest $b+0$, $b+12$, $b+24$ ja $b+36$ vastavalt pesadesse $c+3$, $c+2$, $c+1$ ja $c+0$.

Paigutame pesasse $a+0$ arvu 00 03 0000 0003, pesasse $a+1$ aga arvu 00 00 0012 7777. Indekspesana kasutame pesa 0006:

$k+0$ —10 00 $a+0$ 0006
$k+1$ —10 06 $b+0$ $c+0$
$k+2$ —20 06 $k+1$ $a+1$

Pesadesse $k+1$ ja $k+2$ kirjutatud käske täidetakse siin neli korda. Pesas $k+1$ paiknevat käsku täidetakse teisel korral kujul —10 06 $b+12$ $c+2$, kolmandal korral kujul —10 06 $b+24$ $c+1$ ja neljandal korral kujul —10 06 $b+36$ $c+0$.

Näide 14. Olgu arvud x_i ($i = 0, 1, 2, \ldots, n$) paigutatud järjestikuste aadressidega pesadesse $b+i$. Koostame programmi esimese ferriitkuubi selle pesa aadressi määramiseks, milles paikneb suurim arvudest x_i.

Tähistame otsitava aadressi salvestamiseks kasutatava pesa aadressi $d+0$ ja paigutame pesadesse $a+0$, $a+1$ ja $a+2$ vastavalt arvud m $b+1$ 0000 (siin ikka $m = (n-1)\cdot 2^{-12}$),

0000 0001 0000 ja 0000 7777 0000. Järgnevas programmis on kasutatud indekspesa 0005 ja tööpesa $d+1$:

$k+0$	—10	00	$b+0$	$d+1$
$k+1$	—10	00	0000	$d+0$
$k+2$	—10	00	$a+0$	0005
$k+3$	25	05	0000	$d+1$
$k+4$	—32	00	$k+7$	$k+5$
$k+5$	—10	00	0005	$d+0$
$k+6$	—10	05	0000	$d+1$
$k+7$	—20	05	$k+3$	$a+1$
$k+10$	70	00	$a+2$	$d+0$

Vastus saadakse pesa $d+0$ järkudesse $13 \div 24$, suurim arvudest x_i jääb ühtlasi pesasse $d+1$.

§ 15. TSÜKKEL TSÜKLIS

1. Ülesannete lahendamisel tuleb sageli koostada niisuguseid programme, mille üks tsükliline programmiosa sisaldab teist. Sel korral kõneldaksegi tsüklist tsüklis. Täpsemalt, tsüklilist programmi, mis kuulub mingi teise tsüklilise programmi (nn. välimise tsükli) põhiosasse, nimetatakse tsükliks tsüklis (ehk sisemiseks tsükliks). Seejuures tsükkel, mis ühe tsüklilise programmi suhtes on sisemine, võib omakorda teise suhtes olla välimiseks. Järgnevas piirdume ainult niisuguste tsükliliste programmide koostamisega, milles on üks sisemine ja üks välimine tsükkel (kõik keerulisema struktuuriga tsüklilised programmid koostatakse analoogiliselt).

Nii välimiste kui ka sisemiste tsüklite programmeerimisel võib kasutada eelmises paragrahvis vaadeldud tsükliliste programmide üldskeeme. Seejuures peab aga silmas pidama, et näiteks arvutil «Ural-4» on ainult üks tsükliregister. Seetõttu juhul, kui mõlema (s. t. välimise ja sisemise) tsükli programmeerimisel kasutatakse operatsiooni 25 ja 24, tuleb programmis (enne sisemise tsükli alguskäsu 25 m_2 β_2 täitmist) ette näha tsükliregistri sisu salvestamine operatiivmällu ja selle tagasiviimine tsükliregistrisse pärast sisemise tsükli lõppu. Sellise tsüklilise programmi võib koostada näiteks järgmise skeemi alusel:

$k+0$	25	m_1	β_1	}
$k+1$	—	—	—	
— "	—	—	—	} Välimine tsükkel.
$k+j-2$	—	—	—	
$k+j-1$	27	$k+j+r+1$	0	}

$k+j$	25	m_2	β_2	} Sisemine tsükkel.
$k+j+1$	—	—	—	
—	—	—	—	
$k+j+r$	24	$k+j+1$	0	
$k+j+r+1$	00	0000	0	} Välimine tsükkel.
$k+j+r+2$	—	—	—	
—	—	—	—	
$k+j+r+s$	24	$k+1$	0	

Pesasse $k+j+1$ paigutatud käsu 27 $k+j+r+1$ 0 täitmisel formeeritakse pesasse $k+j+r+1$ käsk 25 t β_1 (kus t tähistab käsu 27 täitmise ajal tsükliregistris olevat arvu), millega pärast sisemise tsükli realiseerimist taastataksegi tsükliregistri sisu.

2. Koostame nüüd näitena mõned programmid, milles üks tsükkel sisaldab teist.

Näide 1. Olgu tarvis arvutada polünoomi

$$p(x) = a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x + a_n$$

väärtused punktides $x_0, x_1, x_2, \ldots, x_t$. Paigutame andmed pesadesse järgmiselt: $(c+0\ 4) = a_0$, $(c+2\ 4) = a_1$, $\ldots$, $(c+2n\ 4) = a_n$, $(d+0\ 4) = x_0$, $(d+2\ 4) = x_1, \ldots, (d+2t\ 4) = x_t$. Kui tulemuste jaoks reserveerida pesad $(e+0\ 4) = p(x_0)$, $(e+2\ 4) = p(x_1)$, $\ldots$, $(e+2t\ 4) = p(x_t)$, siis võib selle ülesande lahendamiseks arvutil «Ural-4» kasutada programmi:

$k+0$	25	$2t$	4	} Välimine tsükkel. Tööpesasse $b+0\ 4$ kirjutatakse argumendi väärtused x_i $(i = 0, 1, \ldots, t)$, pesasse $k+11$ formeeritakse käsk tsükliregistri taastamiseks ja summaatorisse tuuakse kordaja a_0.
$k+1$	—02	$d+2t$	4	
$k+2$	16	$b+0$	4	
$k+3$	27	$k+11$	0	
$k+4$	42	$c+0$	4	
$k+5$	25	$2n-2$	4	} Sisemine tsükkel. Polünoomi väärtuse arvutamine kohal $x = x_i$.
$k+6$	46	$b+0$	4	
$k+7$	—41	$c+2n$	4	
$k+10$	24	$k+6$	0	
$k+11$	00	0000	0	} Välimine tsükkel. Väärtuse $p(x_i)$ salvestamine vastavasse pesasse.
$k+12$	—56	$e+2t$	4	
$k+13$	24	$k+1$	0	

Sama ülesande lahendamiseks arvutil «Minsk-22» paigutame andmed järgmistesse pesadesse: $(a+0) = t$ 0000 0001, $(a+1) = n-1$ 0001 0000, $(c+0) = a_0$, $(c+1) = a_1$, $\ldots$, $(c+n) = a_n$, $(d+0) = x_0$, $(d+1) = x_1$, $\ldots$, $(d+t) = x_t$. Reser-

veerides tulemuste jaoks pesad $(e+0)=p(x_0)$, $(e+1)=p(x_1)$, ..., $(e+t)=p(x_t)$, saame niisuguse programmi:

$k+0$	—10 00 $a+0$ 0007	Välimine tsükkel.
$k+1$	—10 00 $c+0$ $b+0$	
$k+2$	—10 00 $a+1$ 0010	Sisemine tsükkel.
$k+3$	35 07 $b+0$ $d-1$	
$k+4$	16 10 $c+0$ $b+0$	
$k+5$	—20 10 $k+3$ $a+1$	
$k+6$	—10 07 $b+0$ $e-1$	Välimine tsükkel.
$k+7$	—20 07 $k+1$ $a+0$	

kus pesa $b+0$ kasutatakse tööpesana.

Näide 2. Paiknegu pesades $b+0$, $b+1$, $b+2$, ..., $b+j$ tekst tähelis-numbrilises koodis. Koostame programmi, mille abil on võimalik selgitada, mitu korda iga sümbol selles tekstis esineb. Teksti lõpu tunnusena olgu pesasse $b+j+1$ kirjutatud sõnavahe (tühiku) koodid, kusjuures j väärtus pole aga teada. Tulemused paigutame vastavalt pesadesse $c+0$, $c+1$, ..., $c+77$. Täpsemalt, pesasse $c+0$ paigutame selle sümboli esinemiste arvu, mille kood on 00; pesasse $c+1$ selle sümboli esinemiste arvu, mille kood on 01 jne. Arvuti «Ural-4» jaoks saame nii järgmise programmi:

$k+0$	20 0000 0	Nulli viimine pesade loendajasse $a+0$.
$k+1$	16 $a+0$ 0	
$k+2$	30 $a+0$ 0	Uuritava tekstiosa (sõna) salvestamine tööpesasse $a+2$.
$k+3$	02 $b+0$ 0	
$k+4$	16 $a+2$ 0	
$k+5$	14 $k+30$ 0	Kas kogu tekst on läbi vaadatud? Kui $(a+2)=777777$, siis on ülesanne lahendatud.
$k+6$	21 $k+32$ 4	
$k+7$	25 0002 0	
$k+10$	02 $a+2$ 0	Pesa $a+2$ kohtadel $13 \div 18$ paikneva sümboli koodi eraldamine ja salvestamine pesasse $a+1$.
$k+11$	12 $k+31$ 0	
$k+12$	16 $a+1$ 0	
$k+13$	20 0001 0	Pesasse $a+1$ salvestatud koodiga sümboli esinemiste arvu suurendamine ühe (täpsemalt 2^{-18}) võrra.
$k+14$	30 $a+1$ 0	
$k+15$	01 $c+0$ 0	
$k+16$	30 $a+1$ 0	
$k+17$	16 $c+0$ 0	
$k+20$	02 $a+2$ 0	Pesas $a+2$ oleva sõna nihutamine kuue kahendkoha võrra paremale (kohtadele $13 \div 18$ saame uue sümboli koodi).
$k+21$	11 0106 0	
$k+22$	16 $a+2$ 0	
$k+23$	24 $k+11$ 0	

$k+24$	20	0001		0	} Pesade loendaja sisu suurendamine ühe (s. t. 2^{-18}) võrra.
$k+25$	01	$a+0$		0	
$k+26$	16	$a+0$		0	
$k+27$	22	$k+2$		0	
$k+30$	77	7777		0	} Konstandid.
$k+31$	00	0077		0	
$k+32$	37	0000		0	} Peatus.

Selle programmi koostamisel on eeldatud, et pesades $c+0$, $c+1$, ..., $c+77$ paiknevad alguses (s. t. enne vaadeldava programmi töötamist) nullid. Programm koosneb kahest tsüklist: sisemine paikneb pesades $k+7$ kuni $k+23$, välimine aga pesades $k+0$ kuni $k+6$ ja $k+24$ kuni $k+27$.

Arvutile «Minsk-22» sama ülesande lahendamiseks koostatud programm on järgmine:

$k+0$	—10	00	0000	0001	} Nulli kirjutamine indekspesasse 0001.
$k+1$	—10	01	$b+0$	$a+0$	} Uuritava *sõna* salvestamine tööpesasse $a+0$.
$k+2$	07	00	$k+14$	0000	} Kui $(a+0)=444444444444$, siis on ülesanne lahendatud (täidetakse käsk pesast $k+22$).
$k+3$	—34	00	$k+4$	$k+22$	
$k+4$	—10	00	$k+16$	0002	} Sisemise tsükli kordamiste arvu kirjutamine indekspesasse 0002.
$k+5$	71	00	$a+0$	$k+15$	} Pesa $a+0$ kohtadel 31,+36 paikneva sümboli koodi eraldamine ja salvestamine indekspesasse 0003.
$k+6$	12	00	0000	0003	
$k+7$	10	03	$k+20$	$c+0$	} Pesasse 0003 salvestatud koodiga sümboli esinemiste arvu suurendamine ühe (s. t. 2^{-36}) võrra.
$k+10$	64	00	$k+21$	$a+0$	} Pesas $a+0$ oleva *sõna* nihutamine kuue koha võrra paremale.
$k+11$	—20	02	$k+5$	0000	} Sisemise tsükli lõpukäsk.
$k+12$	10	00	$k+17$	0001	} Indekspesas 0001 olevale arvule liidetakse 2^{-24}.

$k+13$	—30	00	$k+1$	0000	
$k-14$	44	44	4444	4444	Konstandid.
$k+15$	00	00	0000	0077	
$k+16$	00	05	0000	0000	
$k+17$	00	00	0001	0000	
$k+20$	00	00	0000	0001	
$k+21$	00	00	0000	0106	
$k+22$	—00	00	0000	0000	Peatus.

Selles programmis paiknevad sisemise tsükli käsud pesades $k+4$ kuni $k+11$, välimise tsükli käsud aga pesades $k+0$ kuni $k+3$ ning $k+12$ ja $k+13$.

Näide 3. Koostame programmi kahe neljarealise ruutmaatriksi A ja B korrutamiseks. Korrutismaatriksi C elemendid c_{ij} arvutatakse teatavasti valemitest

$$c_{ij} = \sum_{k=0}^{3} a_{ik} b_{kj},$$

kus $i, j = 0, 1, 2, 3$ ning a_{ik} ja b_{ik} tähistavad vastavalt maatriksite A ja B elemente.

Arvuti «Ural-4» korral olgu maatriksite A, B ja C elementide jaoks eraldatud topeltpesad $a_{ij} = a + r\ 4$, $b_{ij} = b + r\ 4$, $c_{ij} = c + r\ 4$, kus $r = 10i + 2j$ ning $i, j = 0, 1, 2, 3$. Peale selle tuleb programmis (käskude ümberadresseerimisel) tarvis veel suurusi $(d+0) = 10i$, $(d+1) = 2j$ ning $(d+2) = 2j + 10k$, kus $i, j, k = 0, 1, 2, 3$. Programm ise on järgmine:

$k+0$	25	0036	4	Välimise tsükli alguskäsk.
$k+1$	20	0000	0	Pesadesse $d+0$, $d+1$, $d+2$ ja $d+4$ kirjutatakse nullid ($i = j = k = 0$).
$k+2$	16	$d+0$	0	
$k+3$	16	$d+1$	0	
$k+4$	16	$d+4$	4	
$k+5$	16	$d+2$	0	
$k+6$	27	$k+22$	0	Tsükliregistri sisu taastamise käsu formeerimine pesasse $k+22$.
$k+7$	25	0006	4	Elemendi c_{ij} arvutamine tööpesasse $d+4$ 4.
$k+10$	30	$d+0$	0	
$k+11$	—42	$a+6$	4	
$k+12$	30	$d+2$	0	
$k+13$	46	$b+0$	4	
$k+14$	41	$d+4$	4	
$k+15$	56	$d+4$	4	
$k+16$	20	0010	0	
$k+17$	01	$d+2$	0	
$k+20$	16	$d+2$	0	
$k+21$	24	$k+10$	0	

$k+22$	00 0000 0	Tsükliregistri sisu taastamine.
$k+23$	02 $d+4$ 4	Elemendi c_{ij} salvestamine.
$k+24$	−16 $c+36$ 4	
$k+25$	20 0000 0	Nulli viimine tööpesasse $d+4$ 4.
$k+26$	16 $d+4$ 4	
$k+27$	20 0006 0	Kas maatriksi C i-nda rea kõik elemendid on arvutatud?
$k+30$	14 $d+1$ 0	
$k+31$	21 $k+35$ 4	
$k+32$	20 0002 0	Arvu $(d+1)$ suurendamine kahe võrra (j suurendamine ühe võrra).
$k+33$	01 $d+1$ 0	
$k+34$	22 $k+41$ 0	
$k+35$	20 0010 0	Arvu $(d+0)$ suurendamine $10_{(8)}$ võrra (i suurendamine ühe võrra).
$k+36$	01 $d+0$ 0	
$k+37$	16 $d+0$ 0	
$k+40$	20 0000 0	Pesasse $d+1$ kirjutatakse null ($j=0$).
$k+41$	16 $d+1$ 0	
$k+42$	24 $k+5$ 0	Välimise tsükli lõpukäsk.

Arvuti «Minsk-22» korral olgu maatriksite A, B ja C elementide jaoks eraldatud pesad $a_{ij}=a+r$, $b_{ij}=b+r$ ja $c_{ij}=c+r$, kus $r=4i+j$ ja $i, j=0, 1, 2, 3$. Peale selle kasutame veel konstante $(d+0)=0017\ 0000\ 0000$, $(d+1)=0000\ 0000\ 0001$, $(d+2)=0003\ 0000\ 0000$, $(d+3)=0000\ 0001\ 0004$, $(d+4)=0000\ 0000\ 0003$. $(d+5)=0000\ 0004\ 0000$ ja suurusi $(e+0)=j$ ning $(e+1)=4i$. Andmete sellise paigutuse korral saame järgmise programmi:

$k+0$	−10 00 $d+0$ 0001	Indekspesasse 01 salvestatakse konstant $(d+0)$.
$k+1$	−10 00 0000 $e+0$	Pesadesse $e+0$, $e+1$ ja $e+2$ nullid $(i=j=0)$.
$k+2$	−10 00 0000 $e+1$	
$k+3$	−10 00 0000 $e+2$	
$k+4$	11 00 $d+2$ $e+0$	Elemendi c_{ij} arvutamine pesasse $e+2$.
$k+5$	12 00 $e+1$ 0002	
$k+6$	35 02 $a+0$ $b+0$	
$k+7$	16 00 $e+2$ $e+2$	
$k+10$	−20 02 $k+6$ $d+3$	
$k+11$	−10 01 $e+2$ $c+0$	Elemendi c_{ij} salvestamine pesasse $c+r$ ja nulli kirjutamine pesasse $e+2$.
$k+12$	−10 00 0000 $e+2$	
$k+13$	05 00 $d+4$ $e+0$	Kas i-nda rea kõik elemendid on juba arvutatud?
$k+14$	−34 00 $k+15$ $k+17$	

$k+15$	10	00	$d+1$	$e+0$	} j suurendame ühe võrra.
$k+16$	—30	00	$k+21$	0000	
$k+17$	—10	00	0000	$e+0$	} j võetakse võrdseks nulliga ja suurendatakse arvu $(e+1)$ esimese aadressi nelja ühiku võrra.
$k+20$	10	00	$d+5$	$e+1$	
$k+21$	—20	01	$k+3$	$d+1$	} Välimise tsükli lõpukäsk.

3. Iga ülesande lahendamine algab sellega, et arvuti operatiivmällu salvestatakse vajalik informatsioon (programm, konstandid, algandmed). Arvuti «Ural-4» korral on erilise operatsiooniga *algkäivitus* võimalik kirjutada operatiivmälu topeltpesadesse 0004 4 ÷ 0030 4 ühel perfokaardil olevad arvud (11 neljakümnekohalist kahendarvu). Operatsioon *algkäivitus* lõpeb sellega, et juhtimine antakse automaatselt pesasse 0004. Tavaliselt kantakse esimesele kaardile (s. o. kaardile, millelt operatsiooniga *algkäivitus* kirjutatakse arvud topeltpesadesse 0004 4 ÷ 0030 4) teatav salvestamisprogramm, mille täitmise tulemusena viiaksegi operatiivmällu ülesande lahendamiseks vajalik informatsioon. Esitame komplitseeritud tsükli näitena ühe salvestamisprogrammi, mille täitmisel perfokaartidele kantud arvud kirjutatakse arvuti «Ural-4» operatiivmälu topeltpesadesse $a+0$ 4, $a+2$ 4, ..., $a+2n$ 4 selles järjekorras, nagu nad kaartidelt loetakse.

0004	25	$2n$	4	} Salvestamistsükli alguskäsk.
0005	40	0002	4	} Arvu lugemine kaardilt ja salvestamine vastavasse pesasse.
0006	—16	$a+2n$	4	
0007	26	0002	4	} Kaardilt loetud arvude kontrollsumma arvutamine topeltpesasse 0002 4.
0010	16	0002	4	
0011	02	0012	0	} Kaardi ridade loendamine (loendajana kasutatakse pesa 0001).
0012	01	0001	0	
0013	16	0001	0	
0014	14	0031	0	} Kas kaardilt on loetud juba kõik üksteistkümmend rida?
0015	21	0025	0	
0016	16	0001	0	} Nulli taastamine loendajas.
0017	40	0002	4	} Kontrollsumma lugemine kaardilt ja võrdlemine salvestamisel arvutatud summaga.
0020	14	0002	4	
0021	21	0024	4	
0022	37	0000	0	} Peatus. Lugemisel esines viga.
0023	02	0001	0	} Nulli taastamine kontrollsumma pesas.
0024	16	0002	4	

0025	24	0005	0	}	Salvestamistsükli lõpukäsk.
0026	21	$b+0$	4	}	Salvestamisprogrammist väljumine (juhtimine antakse pesasse $b+0$).
0027	40	0002	0	}	Nullide lugemine paki viimaselt kaardilt.
0030	22	0011	0		
0031	13	0013	0	}	Konstant.

Toodud programmi tegelikuks kasutamiseks tuleb pesadesse 0004, 0006 ja 0026 paigutatud käskudes suhtelised aadressid asendada tõelistega, enne *algkäivitust* peab aga pesadesse 0000 ÷ ÷ 0004 olema viidud nullid.

§ 16. LOOGILISED SKAALAD

1. Eelnevates paragrahvides me vaatlesime programmeerimisvõtteid, mida kasutatakse suhteliselt lihtsate tsükliliste programmide koostamisel. Nimelt sobivad need võtted eeskätt just selliste tsükliliste programmide puhul, milles üht ning sama käskude rühma täidetakse n korda järjest, kusjuures aadressid, millega iga ümberadresseeritavat käsku perioodiliselt täidetakse, moodustavad kas kasvava või kahaneva aritmeetilise progressiooni.

Esineb aga ülesandeid, mille lahendamiseks on tarvis koostada hoopis keerulisema struktuuriga tsüklilisi programme. Näiteks võib juhtuda, et tsükli teatud käsku on vaja perioodiliselt ümber adresseerida nii, et lihtsat reeglit uue aadressi arvutamiseks pole võimalik anda. Analoogiline olukord kujuneb siis, kui tsükli põhiosa moodustab harunev programm, mille korduval täitmisel on tarvis mitteregulaarselt (kuid ette teadaoleval viisil) anda juhtimine kord ühe, kord teise haru käskudele. Tsüklilistes programmides, mille täitmisel on tarvis selliselt kas muuta teatud käskude aadresse või teostada suunamisi, kasutatakse nn. loogilisi skaalasid.

L o o g i l i s e k s s k a a l a k s nimetatakse kahendarvu, milles informatsiooniühikud (kas numbrid 0 ja 1 või nende teatud kombinatsioonid) paiknevad järjekorras, mis vastab nende ühikutega vastavusse seatud objektide kasutamise järjekorrale vaadeldava tsüklilise programmi täitmisel. Skaala võib paikneda kas arvuti operatiivmälu ühes või mitmes pesas.

Selgitame skaalade koostamist ja kasutamist järgnevate näidete abil.

2. Jagunegu koostatava tsüklilise programmi põhiosa kaheks haruks. Tähistame need harud vastavalt tähtedega H_1 ja H_2. Olgu programmiga realiseeritava tsükli kordamiste arv 10, kus-

juures tsükli esimesel rakendamisel olgu tarvis täita haru H_1, teisel haru H_2, siis kolm korda järjest haru H_1, kaks korda haru H_2 ja lõpuks kolm korda haru H_1. Märgime suunamise haru H_1 esimesele käsule numbriga 1, haru H_2 esimesele käsule aga numbriga 0. Moodustame neist numbritest jada, milles 1 ja 0 paiknevad (vasakult paremale) selles järjekorras, milles programmi täitmisel tuleb teostada suunamisi harudele H_1 ja H_2 vastavate käskude rühmade täitmiseks. Nii saame skaala $s = 1011100111$, mille paigutame pesasse nii, et tema esimene kahendkoht asub pesa kohal number 1 (skaalast üle jäävatele kohtadele kirjutame nullid).

Programmis kasutatakse sellist skaalat järgmiselt: tsükli igakordsel rakendamisel nihutatakse loogilist skaalat ühe kahendkoha võrra vasakule (nihutamise tulemus salvestatakse kindlasse tööpesasse) ja sõltuvalt sellest, kas nihutamise tulemusena saadi negatiivne või positiivne arv, antakse juhtimine kas harule H_1 või harule H_2 vastava käskude rühma esimesele käsule. Vaadeldava näite korral saame pärast skaala üheteistkümnendat nihutamist summaatorisse arvu $+0$. Et kõigi eelnenud nihutamiste tulemused erinesid arvust $+0$, siis võib seda asjaolu kasutada vaadeldava tsükli lõputingimusena.

Vaatleme nüüd juhtu, kus koostatava tsüklilise programmi täitmise käigus on tarvis anda juhtimist kolmele erinevale käskude rühmale H_1, H_2 ja H_3. Märgime suunamist haru H_1 esimesele käsule koodiga 01, haru H_2 esimesele käsule koodiga 10 ja haru H_3 esimesele käsule koodiga 11. Olgu tsükli kordamiste arv antud juhul kokku 17, kusjuures harudele H_1, H_2 ja H_3 vastavaid käskude rühmi tuleb täita järjekorras:

$$H_1, H_1, H_1, H_2, H_3, H_2, H_2, H_3, H_2, H_1, H_1, H_3, H_3, H_3, H_2, H_3, H_3.$$

Koostame nendeks suunamisteks skaala, mida tsükli iga kordamise ajal nihutame kahe kahendkoha võrra paremale. Selleks kirjutame, minnes reas paremalt vasakule, üksteise kõrvale suunamiste koodid 01, 10 ja 11 järjekorras, milles tuleb täita harudele H_1, H_2, H_3 vastavaid käskude rühmi. Nii saame skaala

$$s = 11\ 11\ 10\ 11\ 11\ 11\ 01\ 01\ 10\ 11\ 10\ 10\ 11\ 10\ 01\ 01\ 01,$$

mille paigutame pesasse selliselt, et tema viimane kahendkoht paikneb arvuti «Ural-4» korral topeltpesa 37-ndas, «Minsk-22» korral aga pesa 34-ndas järgus (skaala poolt täitmata kohtadele kirjutame nullid).

Programmis kasutatakse sellist skaalat järgmiselt: tsükli igakordsel rakendamisel nihutatakse loogilist skaalat kahe kahendkoha võrra paremale (tulemus salvestatakse kindlasse tööpesasse) ning eraldatakse summaatori kahel viimasel kohal paiknev

kood. Eraldatud koodi võrreldakse kahendarvudega 000 000 ... ... 01 ja 000 000 ... 010, kusjuures ühtimine ühega neist ongi tunnuseks, et juhtimine tuleb anda vastavalt kas haru H_1 või haru H_2 esimesele käsule. Kui eraldatud kood ei ühti kummagagi nendest konstantidest, siis antakse juhtimine haru H_3 esimesele käsule. Et koodi 00 pole skaalas kasutatud, siis võib sellise tulemuse saamist eraldamisel kasutada skaala (tsükli) lõputunnusena.

Analoogiliselt saab kasutada skaalasid ka niisuguste tsükliliste programmide koostamisel, mille täitmisel on tarvis teatud järjekorras suunata rohkem kui kolmele erinevale käsule.

3. Toome nüüd paar lihtsat näidet loogiliste skaalade tegeliku kasutamise kohta.

Näide 1. Koostame programmi summa

$$t = x_1y_1 - x_2y_2 - x_3y_3 + x_4y_4 + x_5y_5 + x_6y_6 + x_7y_7 - x_8y_8 + \\ + x_9y_9 - x_{10}y_{10}$$

arvutamiseks, milles pluss- ja miinusmärgid vahelduvad mitteregulaarselt. Seades liitmisoperatsiooniga vastavusse numbri 0 ja lahutamisoperatsiooniga numbri 1, saame pluss- ja miinusmärkide vaheldumist antud avaldises kirjeldada järgmise skaala abil:

$s = 0110000101.$

Olgu arvuti «Ural-4» korral suuruste x_i ja y_i jaoks eraldatud vastavalt topeltpesad $\langle x_i \rangle = a + 2i\,4$ ja $\langle y_i \rangle = b + 2i\,4$, kui $i = 1, 2, \ldots, 12$ (i väärtused on kaheksandarvud). Paigutame skaala s pesasse $c + 0$, summa t arvutame aga topeltpesasse $d + 04$. Andmete sellise paigutuse korral võib summa t arvutada näiteks järgmise programmi abil:

$k+0$	20	0000	0	Topeltpesasse $d+0$ 4 kirjutatakse null.
$k+1$	16	$d+0$	4	
$k+2$	02	$c+0$	0	Skaala salvestatakse pesast $c+0$ ümber tööpesasse $d+2$.
$k+3$	16	$d+2$	0	
$k+4$	25	0022	4	
$k+5$	—42	$a+24$	4	Korrutis x_iy_i arvutamine ja salvestamine tööpesasse $d+4$ 4.
$k+6$	—46	$b+24$	4	
$k+7$	56	$d+4$	4	
$k+10$	02	$d+2$	0	Skaala nihutamine ühe koha võrra vasakule. Kui tulemus on negatiivne, antakse juhtimine pesasse $k+17$, vastupidisel juhul aga pesasse $k+14$.
$k+11$	11	0001	0	
$k+12$	16	$d+2$	0	
$k+13$	21	$k+17$	0	
$k+14$	42	$d+4$	4	Korrutise x_iy_i liitmine topeltpesas $d+0$ 4 olevale arvule.
$k+15$	41	$d+0$	4	
$k+16$	22	$k+21$	0	

$k+17$	42	$d+0$	4	} Korrutise $x_i y_i$ lahutamine topeltpesas $d+0$ 4 olevast arvust.
$k+20$	43	$d+4$	4	
$k+21$	56	$d+0$	4	
$k+22$	24	$k+5$	0	

Arvuti «Minsk-22» korral olgu suuruste x_i ja y_i salvestamiseks eraldatud vastavalt pesad

$$\langle x_i \rangle = a+i \quad \text{ja} \quad \langle y_i \rangle = b+i \quad (i = 1, 2, \ldots, 12).$$

Paigutame skaala s pesasse $e+0$, konstandi 0011 0001 00001 pesasse $c+1$ ja arvutame summa t pesasse $d+0$. Andmete sellise paigutuse korral saame järgmis programmi:

$k+0$	—10	00	0000	$d+0$	} Pesasse $d+0$ kirjutatakse null ja skaala salvestatakse pesast $c+0$ tööpesasse $d+1$.
$k+1$	—10	00	$c+0$	$d+1$	
$k+2$	—10	00	$c+1$	0003	} Pesas $c+1$ olev konstant kirjutatakse indekspesasse 0003.
$k+3$	35	03	$a+0$	$b+0$	} Korrutise $x_i y_i$ arvutamine pesasse $d+2$.
$k+4$	30	00	$k+5$	$d+2$	
$k+5$	60	00	$c+1$	$d+1$	} Skaala nihutamine ühe kahendkoha võrra vasakule. Kui $(s) > 0$, siis täidetakse käsk pesast $k+7$, muidu aga pesast $k+11$.
$k+6$	—32	00	$k+7$	$k+11$	
$k+7$	14	00	$d+2$	$d+0$	} Korrutise $x_i y_i$ liitmine pesas $d+0$ olevale arvule.
$k+10$	—30	00	$k+12$	0000	
$k+11$	24	00	$d+2$	$d+0$	} Korrutise $x_i y_i$ lahutamine pesas $d+0$ olevast arvust.
$k+12$	—20	03	$k+3$	$c+1$	} Tsükli lõpukäsk.

Näide 2. Koostame programmi summa

$$t = ax_1 + bx_2 + bx_3 + ax_4 + cx_5 + bx_6 + cx_7 + cx_8 + ax_9$$

arvutamiseks (kui eelmises näites vaadeldud summas vaheldusid mitteregulaarselt liitmise ja lahutamise operatsioonid, siis antud juhul tuleb mitteregulaarselt kasutada kordajaid a, b ja c).

Seame kordaja a kasutamisega vastavusse koodi 00, b kasutamisega koodi 01 ja c kasutamisega koodi 10. Koostame skaala s, mis kirjeldab kordajate a, b ning c kasutamise järjekorda, selliselt, et teda tuleb tsükli igakordsel rakendamisel nihutada pesas kahe kahendkoha võrra paremale:

$$s = 00\ 10\ 10\ 01\ 10\ 00\ 01\ 01\ 00.$$

Olgu arvuti «Ural-4» korral suuruste x_i jaoks eraldatud topeltpesad $\langle x_i \rangle = d + 2i\,4$, kus $i = 1, 2, \ldots, 11$ (i väärtused on kaheksandarvud), kordajate a, b ja c jaoks aga vastavalt topeltpesad $d + 24\,4$, $d + 26\,4$ ja $d + 30\,4$. Paigutame skaala s pesasse $k + 21$ nii, et tema esimene kahendkoht on selle pesa märgikohal (skaala viimane kahendkoht paikneb siis kohal 17). Summa t salvestamiseks eraldame topeltpesa $g + 0\,4$, pesi $g + 2$ ning $g + 3$ kasutame aga tööpesadena. Andmete sellise paigutuse korral saame t arvutamiseks järgmise programmi:

$k + 0$	20	0000	0	Nulli kirjutamine pesasse $g + 0\,4$.
$k + 1$	16	$g + 0$	4	
$k + 2$	02	$k + 21$	0	Skaala toomine summaatorisse.
$k + 3$	25	0020	4	Summa arvutamise tsükli alguskäik.
$k + 4$	16	$g + 2$	0	Skaala salvestamine tööpesasse $g + 2$.
$k + 5$	12	$k + 22$	0	Koodi 00, 01 või 10 eraldamine skaalast ja saadud tulemuse (kas kaheksandarvu 000000 0, 000002 0 või 000004 0) salvestamine pesasse $g + 3$.
$k + 6$	16	$g + 3$	0	
$k + 7$	30	$g + 3$	0	Kas korrutise ax_i, bx_i või cx_i arvutamine ja liitmine pesas $g + 0\,4$ olevale arvule.
$k + 10$	42	$d + 24$	4	
$k + 11$	—46	$d + 22$	4	
$k + 12$	41	$g + 0$	4	
$k + 13$	56	$g + 0$	4	
$k + 14$	02	$g + 2$	0	Skaala nihutamine kahe kahendkoha võrra paremale.
$k + 15$	11	0102	0	
$k + 16$	24	$k + 4$	0	Summa arvutamise tsükli lõpukäsk.
$k + 17$	37	0000	0	Pärast t väärtuse arvutamist arvuti peatub (*käivitusega* on võimalik juhtimist anda pesas $k + 23$ olevale käsule).
$k + 20$	22	$k + 23$	0	
$k + 21$	24	6050	0	Loogiline skaala.
$k + 22$	00	0006	0	Eraldamiskonstant.

Olgu arvuti «Minsk-22» korral suuruste x_i salvestamiseks eraldatud pesad $\langle x_i \rangle = d + i$, kus $i = 1, 2, \ldots, 11$ (i väärtused on kaheksandarvud), kordajate a, b ja c jaoks aga vastavalt pesad $a + 12$, $a + 13$ ja $a + 14$. Paigutame skaala s pesasse $e + 0$ nii,

et tema viimane (s. t. parempoolseim) kahendkoht on selle pesa 36-ndal kohal. Peale skaala on selle ülesande lahendamisel vaja veel konstante $(e+1) = 0010\,0001\,0000$, $(e+2) = 0000\,0000\,0003$, $(e+3) = 0000\,0000\,0102$ ja tööpesi $g+1$, 0004 ning 0005 (viimased kaks indekspesadena). Summa t paigutame pesasse $g+0$.

Andmete sellise paigutuse korral saame t arvutamiseks arvutiga «Minsk-22» järgmise programmi:

$k+0$	—10	00	0000	$g+0$	Nulli kirjutamine pesasse $g+0$.
$k+1$	—10	00	$e+0$	$g+1$	Skaala kirjutamine tööpesasse $g+1$.
$k+2$	—10	00	$e+1$	0004	Summa arvutamise tsükli algus.
$k+3$	71	00	$g+1$	$e+2$	Koodi 00, 01 või 10 eraldamine skaalast ja saadud tulemuse salvestamine pesasse 0005.
$k+4$	12	00	0000	0005	
$k+5$	11	05	0000	$a+12$	Kas korrutise ax_i, bx_i või ex_i arvutamine ja liitmine pesas $g+0$ olevale arvule.
$k+6$	37	04	$d+0$	0000	
$k+7$	16	00	$g+0$	$g+0$	
$k+10$	60	00	$e+3$	$g+1$	Skaala nihutamine kahe koha võrra paremale.
$k+11$	—20	04	$k+3$	$e+1$	Summa arvutamise tsükli lõpukäsk ja peatuskäsk.
$k+12$	—00	00	0000	0000	

Tegelikult me kasutasime mitmes erinevas pesas paiknevat skaalat juba eelmise paragrahvi näite 2 programmeerimisel. Nimelt võib selles näites skaalana vaadelda pesades $b+0$, $b+1$, ..., $b+j$ paiknevat teksti, selle skaala informatsiooniühikutena aga sümbolite kuuekohalisi kahendkoode.

Selle näite puhul arvutile «Ural-4» koostatud programmis kasutatakse loogilist skaalat pesas $k+15$, «Minsk-22» jaoks koostatud programmis aga pesas $k+7$ oleva käsu ümberadresseerimisel. Sõltuvalt uuritavast tekstist võib osutuda vajalikuks nende käskude korduv täitmine kuni 64 erineva aadressiga.

V. PEATÜKK

TÄIENDAVAID PROGRAMMEERIMISVÕTTEID

§ 17. ALAMPROGRAMMIDE SÜSTEEMID

1. Asudes koostama ülesande lahendamise programmi, huvitab programmeerijat kõigepealt küsimus, kas ta saab ära kasutada juba varem koostatud programme või programmiosi. Varem koostatud (seega ka silutud) programmi kasutamine võib programmeerija tööd tunduvalt kergendada. Eriti palju võimalusi pakuvad just sellised programmid, mis on otseselt koostatud korduvaks kasutamiseks. Neile on iseloomulik lühidus ja universaalsus, s. o. paljude parameetrite muutmise võimalus. Et programmeerijal olemasolevatest programmidest võimalikult suurem abi oleks, ongi aja jooksul välja töötatud nende koostamise ja kasutamise metoodika. Põhijoontes on see metoodika erinevate arvutitüüpide puhul ühesugune.

Sama tüüpi ülesannete lahendamise programmide omavaheline erinevus on enamasti väike. Põhiline osa programmist jääb niisuguste ülesannete lahendamisel muutmata, vahetuvad ainult mõned etteantavad suurused. Lineaarse võrrandisüsteemi lahendamise programmis näiteks võivad muutuda võrrandisüsteemi järk ja kordajad. Ühetüübiliste ülesannete massiliseks lahendamiseks ettenähtud programme nimetatakse *standardprogrammideks*. Konkreetse ülesande lahendamiseks on sel juhul vaja ainult standardprogrammide juurde koostada programmilõigud, mis organiseerivad tema kasutamist, s. t. annavad ette algandmeid, väljastavad vahetulemusi ning resultaate jms.

Erinevate ülesannete lahendamise algoritmid on üldiselt erinevad, kuid sageli esineb nendes ühesuguseid funktsionaalseid sõltuvusi, valemeid. Ka ühe ja sellesama ülesande lahendamise algoritmis võib mingi valem või selle osa korduda. Kõigi niisuguste ülesannete puhul piisab selle korduva algoritmi või valemi ühekordsest programmeerimisest. Selline programmiosa tuleb aga koostada nii, et ta oleks kasutatav mitmel pool sisemälus.

Selleks et ühes programmis mingit programmiosa ei peaks korduvalt välja kirjutama, varustatakse viimane sisendiga ja väljundiga, mis võimaldavad tema poole pöörduda mistahes kohast

programmis. Sellist programmiosa nimetatakse *alamprogrammiks*. Teda iseloomustavad eelkõige ühe kindla algoritmi realiseerimine ja kasutatavus programmi mistahes osas. Erinevalt standardprogrammist ei ole alamprogramm ülesande programmi põhiosa. Alamprogrammi näiteks on funktsioonide $\sin x$ ja $\ln x$ arvutamise programmid.

Iga arvuti juures on pidevalt kasutusel vähemalt mõnikümmend alamprogrammi. Suure arvu alamprogrammide kasutamise lihtsustamiseks ühendatakse need ühtsesse süsteemi. Eelkõige võimaldab ühtne süsteem lihtsustada alamprogrammide hoidmist ja nende sisselülitamist mistahes programmi. Süsteem esitab alamprogrammidele rea nõudeid, eeskätt nende struktuuri kohta. Süsteemi nõudeid rahuldavaid alamprogramme nimetatakse *standardseteks alamprogrammideks*. Süsteemi ennast nimetatakse *standardsete alamprogrammide süsteemiks*, vahel ka *standardprogrammide süsteemiks*. Järgnevalt vaatlemegi üht arvutite «Ural-4» puhul laialt levinud süsteemi. Kõigepealt aga selgitame veel mõningaid alamprogrammide koostamisega seotud üldisi küsimusi.

2. Alamprogrammi koostamisel ei ole ette teada millistes pesades teda edaspidi kasutatakse ja mis on sel ajal mälu teistes pesades. Koostada alamprogrammid tööks ainult mingites kindlates pesades oleks ebaratsionaalne, sest juba mälu jaotamisel satuksime enamiku ülesannete korral suurtesse raskustesse. Asukoha muutmisega aga kaasneb vajadus muuta alamprogrammi käskude aadressiosi. Seejuures kuulub muutmisele ainult osa aadresse ja neidki ei muudeta kõiki ühtemoodi. Üldiselt tuleb alamprogrammides eristada nelja liiki aadresse.

Aadresse, mis ei muutu alamprogrammi ega mingi teise programmiosa nihutamisel mälus, nimetatakse selle alamprogrammi *konstantseteks aadressideks*. Konstantseteks aadressideks on niisuguste käskude aadressiosad, kus sisuliselt ei ole tegemist mälu pesa aadressidega vaid konstantidega (nagu nihkekonstant või tsükliloendaja sisu).

Ainult alamprogrammi enda asukohast sõltuvaid aadresse nimetatakse selle alamprogrammi *sisemisteks aadressideks*. Töökorda seatud alamprogrammis asuvad sisemised aadressid alati samas aadresside vahemikus, kus töötab alamprogramm. Alamprogrammi teistesse pesadesse tööle üleviimisel tuleb muuta sisemisi aadresse. Kõigile sisemistele aadressidele tuleb juurde liita ühine konstant — alamprogrammi uue ja endise asukoha aadresside vahe. Näiteks alamprogrammi üleviimisel pesadest 1200, ... pesadesse 0400, ... peab sisemise aadressi 1431 muutuma aadressiks 0631 [$1431 + (400 - 1200) = 631$].

Neid alamprogramme, mis sisaldavad ainult konstantseid ja

sisemisi aadresse, nimetatakse kinnisteks alamprogrammideks. Kinnine alamprogramm ei kasuta mingi teise programmi või alamprogrammi käske, konstante ega tööpesi. Mõnedes süsteemides on lubatud kinnistes alamprogrammides tööpesadena kasutada veel piiratud arvu pesi sisemälu algusest või lõpust. Neid pesi nimetatakse alamprogrammide standardseteks tööpesadeks. Viimaste aadressid alamprogrammis on alati konstantsed aadressid.

Osa alamprogramme kasutavad töö ajal omakorda ühte või rohkem alamprogrammi. Kui kõik need alamprogrammid paigutada jäigalt üksteise kõrvale, saame kinnise alamprogrammi. Sellisel ühendamisviisil on aga rida puudusi. Nimelt võib mitme nii koostatud alamprogrammi üheaegsel kasutamisel mõni alamprogramm esineda korduvalt. Puuduseks on ka alamprogrammi suur pikkus: ühele pikale alamprogrammile on raskem mälus ruumi leida kui mitmele eraldi alamprogrammile. Neist puudustest vabanemiseks jäetakse alamprogrammid asetuse poolest üksteise külge ühendamata ja side alamprogrammide vahel luuakse käskude aadressidega. Alamprogrammi aadressi, mis sõltub mingi teise, temast väljaspool asuva alamprogrammi (või programmi) asukohast, nimetatakse selle alamprogrammi väliseks aadressiks. Väliseid aadresse sisaldavat alamprogrammi aga nimetatakse lahtiseks alamprogrammiks. Kõige sagedamaks väliseks aadressiks on aadress suunamiskäsus, mis annab juhtimise teisele alamprogrammile. Väliste aadresside abil on samuti võimalik ühes alamprogrammis kasutada teise konstante ja tööpesi. Standardsete tööpesade kasutamisel peab lahtise alamprogrammi koostamisel ette vaatama, et üks alamprogramm ei kustutaks teise poolt sinna salvestatud vahetulemusi. Kindlam on niisugusel juhul tööpesadena kasutada alamprogrammi juures asuvaid pesi. Tööpesade suurema arvu korral võib aga kasutada ka väljaspool alamprogrammi asuvat tööpesade rühma, mida võime käsitleda kui ainult tööpesadest koosnevat alamprogrammi.

Reas alamprogrammides tuleb kasutada veel aadresside muutmist vastavalt etteantud parameetritele. Aadressi, mis sõltub ühest või enamast parameetrist, nimetataksegi parameetrist sõltuvaks aadressiks. Näiteks võib aadress sõltuda võrrandisüsteemi järgust, etteantud täpsusest, algandmete tabeli mõõtmetest jne. Mõnel juhul võib parameetritest sõltuda ka kogu käsk (või konstant). Sõltuvus parameetritest võib olla komplitseeritud nii üleskirjutuselt kui ka programmeerimistehniliselt. Alati on aga võimalik parameetrist sõltuvaid aadresse töödelda alamprogrammis endas. Kui siinjuures piisab enne ülesande lahendamist läbiviidud töötlemisest, siis ülesande lahendamise ajaks need (tavaliselt alamprogrammi lõppu paigutatud) ümbertöötava osa pesad vabanevad.

3. Ülesande lahendamise programm koosneb tavaliselt *põhiprogrammist* ja alamprogrammidest (kui standardprogramme vaadelda suurte alamprogrammidena). Programmeerija koostab põhiprogrammi ühe kindla ülesande lahendamiseks, kuid oma töö käigus põhiprogramm kasutab alamprogramme. Pöördumisel mingi alamprogrammi poole põhiprogramm annab ette nõutavad algandmed ja suunab juhtimise tema sisendile. Alamprogrammi *sisendiks* nimetatakse pesa, milles paiknevale käsule antakse pöördumisel juhtimine. Sisendeid võib alamprogrammis sealjuures olla rohkem kui üks. Pöördumine toimub paljude arvutite puhul spetsiaalse käsuga, näiteks «Ural-4» puhul käsuga 22 *a* 4, «Minsk-22» puhul aga käsuga — 31 00 *a b*.

Algandmete etteandmine mingile alamprogrammile toimub alati ühel kindlal viisil, kuigi eri alamprogrammid võivad nõuda erinevaid pöördumisviise. Alamprogrammile antakse ette kas algandmed ise või siis informatsioon nende asukoha kohta. Selleks kasutatakse summaatorit või alamprogrammile teadaolevaid pesi (sageli spetsiaalsele pöördumiskäsule järgnevaid pesi).

Pärast töötamist annab alamprogramm juhtimise tagasi programmiosale, kus toimus alamprogrammi poole pöördumine, enamasti just pöördumiskäsule järgnevale käsule. Pesa, kus asub juhtimist tagasi andev käsk, nimetatakse alamprogrammi *väljundiks*. Enne juhtimise tagasisuunamist annab alamprogramm välja tulemused, salvestades nad fikseeritud või pöördumisel etteantud pesadesse (sageli ka summaatorisse).

Alamprogrammide tegeliku kasutamise huvides tuleb nad seostada vastavaks süsteemiks (nn. *programmeerimise väikeautomatiseerimise süsteem*). Niisuguse süsteemi põhieesmärgiks on lihtsustada põhiprogrammi ja alamprogrammide ühendamist ühtseks ülesande lahendamise programmiks. Ühendamine seisneb alamprogrammide kohalepaigutamises ja nende sisemiste, väliste ja parameetrist sõltuvate aadresside ümbertöötamises (kui selliseid esineb).

Süsteemis kasutatavad alamprogrammid koondatakse niinimetatud *alamprogrammide biblioteegiks*, mida hoitakse arvuti välises mälus. Biblioteegi programmid nummerdatakse. Nende viimiseks biblioteeki ja biblioteegi kontrolliks kasutatakse sealjuures vastavaid spetsiaalseid programme. Paljudesse biblioteekidesse viiakse koos alamprogrammidega ka mõningad andmed tema kohta (programmi üksikute osade pikkused, kontrollsumma jne.), mis võimaldavad alamprogrammi sisemällu tuua ja põhiprogrammiga ühendada minimaalse informatsiooni etteandmise teel.

4. Põhiprogrammi ja alamprogrammide ühendamiseks kasutatakse süsteemides põhiliselt kahte tüüpi ühendavaid programme — kompileerivaid ja interpreteerivaid programme.

Kompileeriv programm ühendab alamprogrammid põhiprogrammiga kokku ühtseks ülesande lahendamise programmiks. Iseloomulik on veel see, et kompileerimine toimub enne ülesande lahendamist ja saadud programm oma töö käigus kompileerivat programmi enam ei kasuta. Viimase võib pärast kompileerimist seega sisemälust kustutada.

Interpreteeriv programm asub kogu ülesande lahendamise ajal sisemälus ja ühendab alamprogrammid (tavaliselt kinnised) põhiprogrammiga ajutiselt, siis kui põhiprogramm pöördub vastava alamprogrammi poole. Pöördumine toimub seega interpreteeriva programmi kaudu. Interpreteeriva programmi kasutamisel on eeliseks, et siin ei pea kõiki põhiprogrammis kasutatavaid alamprogramme korraga sisemällu paigutama. Puuduseks on aga ülesande lahendamise aja pikenemine alamprogrammide sagedasel vahetamisel, mida põhjustab välise mälu (biblioteegi) kasutamisega seotud tunduv ajakulu.

Kompileerivat tüüpi ühendavate programmide eriliigiks on nn. koostavad programmid. Seda tüüpi ühendavate programmidega seoses kasutatakse sageli terminit *massiiv*. Massiiv on mingis mõttes terviklik programmiosa, mis paikneb järjestikustes pesades. Sõltuvalt massiivi sisust võime rääkida programmi massiivist, käskude, konstantide, tööpesade, algandmete jne. massiividest. Omaette massiividena vaadeldakse ka alamprogramme. Massiividele kantakse üle kõik alamprogrammide puhul kasutatavad nimetused ja definitsioonid. Nii kasutame nimetusi *kinnine massiiv, lahtine massiiv* ja *parameetrist sõltuv massiiv*. Kui massiivis kõik aadressid on konstantsed, siis nimetatakse massiivi invariantseks. Samuti räägime massiivide biblioteegist, nimetades biblioteeki kuuluvat massiivi biblioteegimassiiviks.

Koostava programmi kasutamisel jaotatakse kogu ülesande lahendamise programm eelnevalt massiivideks. Eeskätt eraldatakse välja alamprogrammid, kuid ka põhiprogramm jaotatakse vajaduse järgi massiivideks nii, et igaühte neist oleks lihtne eraldi programmeerida. Koostav programm ühendabki enne ülesande lahendamist massiivid terviklikuks programmiks. Seega saab ühe ülesande programmeerimisel edukalt kasutada mitut programmeerijat. Iga massiivi saab pealegi vajaduse korral lihtsalt lülitada mingi teise ülesande lahendamise programmi.

Programmeerimise väikeautomatiseerimise süsteemis võib üheaegselt olla kasutusel mitu kompileerivat (koostavat) ja interpreteerivat tüüpi ühendavat programmi.

5. Järgnevas tutvustame üht konkreetset programmeerimise väikeautomatiseerimise süsteemi, mis on kasutatav arvutite «Ural-2», «Ural-3» ja «Ural-4» korral. See E. T. Gavrilenko juhtimisel koostatud süsteem on kasutusel reas arvutuskeskustes. Süsteemi pidevalt täienev biblioteek sisaldab juba suure hulga alam-

programme, mis koos süsteemikirjeldusega on ilmunud eraldi väljaannetena (vt. näit. [13]).

Vaatlusele tuleva süsteemi põhiprogrammid — koostavad ja interpreteerivad programmid on ette nähtud massiivide ühendamiseks, kusjuures süsteem on välja töötatud eeskätt koostavate programmide kasutamisest lähtudes. Et ta on ette nähtud erinevate arvutitüüpide jaoks, siis saab iga tüübi puhul muidugi kasutada ainult temale ettenähtud põhiprogramme. Edaspidises vaatleme vaid kahte arvutil «Ural-4» kasutatavat põhiprogrammi — koostavat programmi «ССП-42» (ССП = стандартная составляющая программа) ja interpreteerivat programmi «СИП-21» (СИП = стандартная интерпретирующая программа).

Süsteemi koosseisu kuuluv alamprogrammide biblioteek jaguneb kahte ossa, kusjuures mõlemad osad on kõigi kolme arvutiliigi korral ühesugused.

Esimene osa on alamprogrammide biblioteek töötamiseks harilike arvudega. Teise osa moodustavad mõningad spetsiaalsed biblioteegid: alamprogrammide biblioteek töötamiseks kompleksarvudega, alamprogrammide biblioteek töötamiseks kahekordse pikkusega arvudega jt. Iga niisuguse biblioteegi juurde kuulub oma interpreteeriv programm, ilma milleta selle biblioteegi alamprogramme kasutada ei saa.

Süsteemi mitmesuguseid teenistuslikke programme saab kasutada kõigi ülesannete juures, nii programmi koostamisel kui ka ülesande lahendamisel. Siia kuuluvad näiteks standardsed silumisprogrammid, massiivide biblioteegi korrashoiu programmid (mis täiendavad ja kontrollivad biblioteeki), alamprogrammid informatsiooni vahetamiseks mäluseadmete vahel, alamprogrammid informatsiooni sisseviimiseks ja väljatoomiseks (trükkimiseks või perforeerimiseks) jt. Süsteemi kasutajal tuleb muidugi arvestada kogu süsteemi pideva täiustumisega kõigis tema põhiosades.

6. Vaatluseletuleva süsteemi kõige põhilisemaks mõisteks on m a s s i i v. Programmi jaotus massiivideks ei ole üheselt määratud. Ainult biblioteegi alamprogrammid moodustavad igaüks etteantud massiivi, põhiprogrammi jaotab aga massiivideks programmeerija. Kindlat jaotuse eeskirja siinjuures anda ei saa, sest lahendatavad ülesanded (ja lahendamise algoritmid) võivad olla väga erinevad. Tavaliselt osutub siiski otstarbekaks kasutada omaette programmide massiive (käskude massiive), tööpesade, konstantide, algandmete ja tulemuste massiive. Sageli saab mõne massiivi pesi aga kasutada ka mitmel otstarbel, näiteks algandmete massiivi pesi tööpesadena või tulemuste salvestamiseks.

Massiivide eristamiseks ja lühidaks ülesmärkimiseks omistatakse igale massiivile kindel number. Biblioteeki lülitatavatele

massiividele antakse sealjuures püsivad numbrid, kõigile ülejäänud massiividele aga ajutised numbrid. Numbrid valitakse vaadeldavas süsteemis kuuekohaliste kaheksandarvudena, seega massiivi M_j number N_j peab asuma vahemikus

$$000\,000 \leqslant N_j \leqslant 777\,777$$

Numbri esimesele kolmele kaheksandkohale kirjutatakse massiivi programmeerimisel kasutatud arvuti seerianumber, kui aga arvuti pole oluline, siis 777. Püsivaid numbreid hakatakse valima kasvavas järjekorras (*** 000, *** 001, ...), ajutisi numbreid aga lõpust ettepoole (*** 777, *** 776, ...).

Pärast programmi jaotamist massiivideks saab põhiprogrammi massiivide koostamise jagada mitme programmeerija vahel. Koostatavate massiivide asukohad põhiprogrammis võib sealjuures esialgu jätta fikseerimata. Alles enne ülesande tegelikku lahendamist paigutatakse massiivid koostava programmi abil sisemälu pesadesse. Fikseerimata jäävad koostamise ajal ka alamprogrammide asukohad. Need massiivid koostatakse tõelistes aadressides, algusaadress a_n valitakse seejuures vabalt mingi paarisarvulise aadressina. Ülevaatlikkuse huvides võetakse algusaadressis kolm viimast numbrikohta tavaliselt nullideks, näiteks $a_n = 4000$. Massiivile järgneva esimese paarisarvulise aadressiga pesa aadressi tähistame sümboliga a_k. Kõik vaadeldava massiivi sisemised aadressid asuvad siis a_n ja a_k vahel. Seda diapasooni nimetatakse massiivi sisemiste aadresside diapasooniks. Kuid mitte iga sellesse diapasooni kuuluv aadress ei tarvitse veel olla sisemine aadress. Nagu me allpool näeme, peab sisemine aadress asuma massiivi koostava programmiga töödeldavas osas, nn. töötlemispiirkonnas.

Analoogiliselt sisemiste aadressidega kasutatakse ka massiivi väliste aadresside kirjapanekul vastavaid diapasoone. Massiivi M_i välist aadressi nimetame massiivi M_j juurde kuuluvaks, kui ta sõltub massiivi M_j asukohast sisemälus. Sama massiivi M_j juurde kuuluvatele välistele aadressidele valib programmeerija massiivi M_i koostamisel aadresside diapasooni algusaadressiga b_n ja lõppaadressiga b_k. Massiiv M_i võib olla seotud mitme erineva massiiviga M_j. Kõigi nende massiivide juurde kuuluvate väliste aadresside jaoks tuleb massiivi M_i koostamisel valida vastavad diapasoonid. Diapasoonide valikul tuleb silmas pidada järgmisi asjaolusid.

Diapasoonide algus- ja lõppaadressid peavad olema paarisarvulised.

Sisemiste aadresside diapasoon ja väliste aadresside diapasoonid ei tohi kattuda.

Sama massiivi M_j juurde kuuluvatele välistele aadressidele võib valida ka mitu diapasooni.

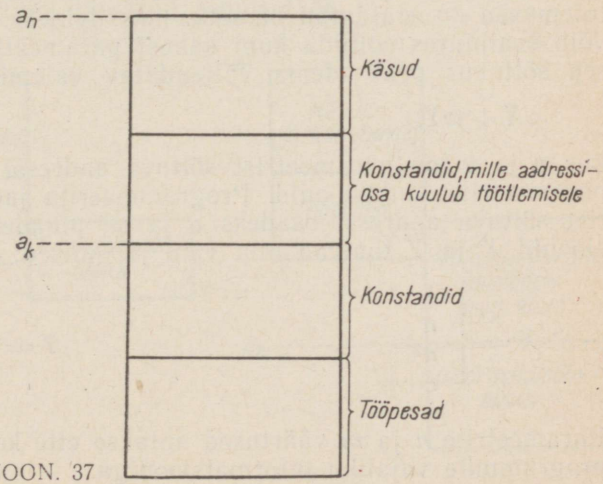
Väliste aadresside diapasoonide valik massiivi M_i programmeerimisel ei sõltu teiste massiivide diapasoonidest.

Diapasoonide algusaadressid (a_n ja b_n) peavad olema suuremad kui 0033, sest pesad 0001—0033 on arvutis «Ural-4» kasutusel algkäivituse juures. Seepärast on soovitav kinniste massiivide koostamisel pesi 0001—0033 kasutada tööpesadena. Nende pesade aadressid ei kuulu siis ühtegi diapasooni ja on seega igas massiivis konstantsed.

Massiivil tohib olla ainult üks sisemiste aadresside diapasoon. Kui sisemiste aadresside kodeerimisel osutub vajalikuks kasutada rohkem diapasoone, tuleb juurdevõetud diapasoone käsitleda kui väliste aadresside diapasoone, mis kuuluvad sellesama massiivi M_i juurde.

Koostavale (interpreteerivale) programmile tuleb ette anda massiivi see osa või osad, kus asuvad sisemiste ja välimiste aadressidega pesad. Nimetame neid osi massiivi t ö ö t l e m i s p i i r k o n d a d e k s. Väljaspool töötlemispiirkonda olevate pesade sisu koostav (interpreteeriv) programm ei muuda, sõltumata sellest, kas aadressiosa kuulub massiivi M_i mingisse aadresside diapasooni või ei kuulu.

Aadresside kodeerimine on kõige lihtsam siis, kui massiiv on joonisel 37 esitatud kujuga. Sellises massiivis on ainult üks töötlemispiirkond algusaadressiga a_n ja lõppaadressiga a_k. Üldiselt võib aga massiivis olla mistahes arv töötlemispiirkondi. Vajadus suurema arvu töötlemispiirkondade järele võib tekkida siis, kui käskude seas on modifitseerimisele mittekuuluvaid konstante. Juhul kui konstandi aadressiosa ei kuulu massiivi M_i ühtegi aad-



JOON. 37

resside diapasooni või kui tema koodiosaks on üks arvudest 11, 25, 32, 33, 34 ja 40, siis võib niisugune konstant paikneda ka töötlemispiirkonna sees (et nende koodide järel ei ole käskudes mitte aadress vaid konstant, siis koostav programm neid aadressiosi ei muuda). Ülejäänud konstandid aga tuleb kas töötlemispiirkondadest välja jätta (mis võib kaasa tuua töötlemispiirkondade arvu suurendamise) või valida aadresside diapasoonid nii, et nende konstantide aadressid jäävad väljapoole diapasoone.

Märgime veel, et töötlemispiirkonna rajad a_n ja a_k võivad olla nii paaris- kui paaritud arvud.

7. Mõningate massiivide programmeerimisel ei saa piirduda üksnes sisemiste ja väliste aadresside kasutamisega. Seetõttu näeb süsteem ette võimaluse aadresside töötlemiseks mistahes eeskirja järgi. Siin tuleb vahet teha kahe juhu vahel — kas aadresse tuleb formeerida ainult enne programmi töötamist või ka töötamise ajal.

Vajadus aadresside formeerimiseks programmi töö ajal tekib näiteks siis, kui aadresside töötlemiseks vajalikud parameetrid ei ole eelnevalt teada või massiiv peab töötama rohkema kui ühe parameetrite kombinatsiooniga. Sel juhul koostatakse massiivi juurde omaette sisendiga programmilõik — formeeriv osa, mis jääb massiivi juurde kogu ülesande lahendamise ajaks. Pöördumisel formeeriva osa poole antakse ette aadresside töötlemiseks vajalikud andmed.

Lihtsama sõltumise korral parameetrist saab aadresside formeerimiseks kasutada koostavat programmi. Keerulisemate sõltuvuste korral tuleb aga kasutada niinimetatud mittestandardset formeerivat programmi.

Koostavat programmi saab parameetritest sõltuvate aadresside formeerimiseks kasutada vaid siis, kui lähteandmed, tööpesad ja tulemused on eraldatud omaette massiivideks. Muudetav aadress võib sealjuures sõltuda kuni kahest parameetrist (n ja m) ning see sõltuvus peab olema väljendatav valemiga

$$vX + wY,$$

kus v ja w on parameetrist sõltuva aadressi osad, X ja Y aga parameetrite funktsioonid. Programmeerija jaotab iga parameetrist sõltuva aadressi osadeks v ja w, pidades silmas, et funktsioonid X ja Y tohivad olla vaid järgmised:

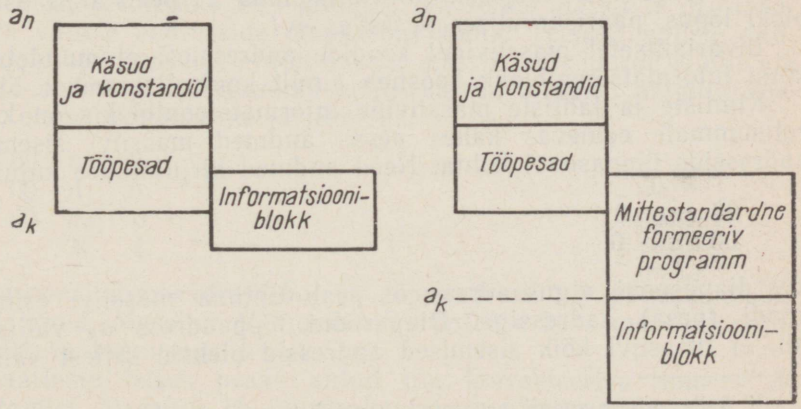
$$X = \begin{cases} 1 \\ n \\ n^2 \\ n \cdot m \end{cases} \qquad Y = \begin{cases} 1 \\ m \\ m^2 \\ n \cdot m \end{cases}$$

Parameetrite n ja m väärtused antakse ette koos muu koostavale programmile vajaliku informatsiooniga.

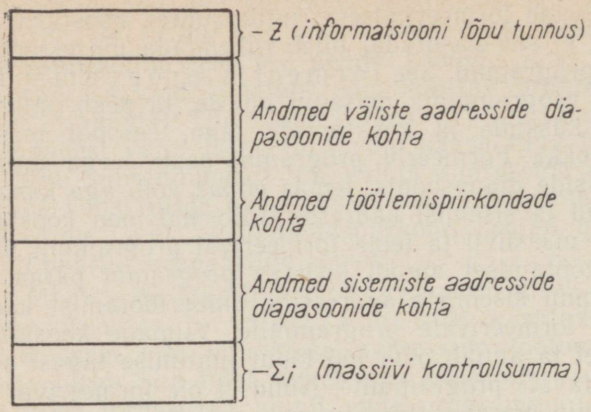
Nende aadresside formeerimiseks, mille juures koostavat programmi otseselt ei saa kasutada, tuleb rakendada mittestandardset formeerivat programmi. See formeeriv programm koostatakse omaette programmina massiivi juurde, ta peab paiknema põhiprogrammi käskude ja konstantide taga, eespool massiivi informatsiooniblokki. Formeeriv programm asub seega massiivi sisemiste aadresside diapasoonis, temas endas võib aga kasutada vaid konstantseid ja sisemisi aadresse. Informatsioon koostavale programmile on massiivil ja tema formeerival programmil ühine. Massiivi ümbertöötamisel annab koostav programm pärast formeeriva programmi sisemiste aadresside ümbertöötamist käsuga 22 *a* 4 juhtimise formeerivale programmile. Viimane koostatakse alati sellisena, et ta annab oma töö lõpul juhtimise tagasi sisendile, seega koostavale programmile. Nüüd ei ole formeerivat programmi (erinevalt massiivi formeerivast osast) enam tarvis ja tema kohale võib koostav programm paigutada järgmise massiivi.

8. Iga massiivi lõppu paigutatakse eraldi blokk andmetega massiivi aadresside kohta, seda blokki nimetame edaspidi massiivi informatsiooniblokiks. Nende andmete põhjal formeerib koostav (või interpreteeriv) programm oma töö käigus massiivis tõelised aadressid. Blokk koostatakse pärast massiivi programmeerimist ja ta võib muutuda ainult massiivi enese muutmisel (näiteks silumisel).

Informatsiooniblokk paigutatakse alati massiivi lõppu. Bloki alla võib võtta ka massiivi lõpus asuvad tööpesad, seda tingimusel, et nende sisu enne massiivi töötamist on suvaline. Formeeriva programmi kasutamisel massiivis peab see asuma massiivi lõpus, informatsioonibloki ees. Informatsioonibloki kaks võimalikku paigutust on näidatud joonisel 38.



JOON. 38



JOON. 39

Massiivi M_i informatsiooniblokki koondatakse andmed selle massiivi aadresside diapasoonide ja töötlemispiirkondade kohta joonisel 39 näidatud järjekorras.

Kõik massiivid tuleb varustada kontrollsummaga Σ_i, mis on saadud pesade a_n kuni $a_n + l - 1$ summeerimisel spetsiaalse programmi abil.[27] Programm Σ_i arvutamiseks peab olema kujul:

c		11			0050	0	$c+$	6	16	$t+$	0000 0
$c+1$		25		$l-$	0001	0	$c+$	7	01	$t+$	0000 4
$c+2$		–26	a_n+l-		0001	0	$c+$	10	16	$t+$	0001 0
$c+3$		24		$c+$	0002	0	$c+$	11	01	$t+$	0000 4
$c+4$		16		$t+$	0001	0	$c+$	12	11		0024 0,
$c+5$		14		$t+$	0001	0					

kus $t + 0000\ 4$ on tööpesa. Kontrollsumma Σ_i peab alati asuma bloki lõpus, paarisarvulises pesas $a_n + l$.

Invariantsetel massiividel koodide aadressiosi ei muudeta ja kogu informatsiooniblokk koosneb ainult kontrollsummast Σ_i.

Kinniste ja lahtiste massiivide informatsiooniblokis on kontrollsummale eelnevas kahes pesas andmed massiivi sisemiste aadresside diapasooni kohta. Need andmed kirjutatakse kujul

00 a_k 0,
00 a_n 0,

kus diapasooni algusaadress a_n peab ühtima massiivi esimese koodi (pesa) aadressiga. Diapasooni lõppaadress a_k valitakse nii, et massiivi kõik sisemised aadressid oleksid sellest väikse-

[27] Selle programmiga arvutatud kontrollsumma ei sõltu operatsiooni 26 täitmise erinevustest arvutitel «Ural-4» ja osal arvutitel «Ural-2».

mad. Tavaliselt võetakse a_k võrdseks massiivi järel asuva esimese vaba pesa aadressiga. Kui aga sisemised aadressid puuduvad, siis tuleb võtta $a_k = a_n$. Suurused a_n ja a_k peavad alati olema paarisarvulised ja täitma tingimust

$$0033 < a_n \leqslant a_k.$$

Eespool suurusi a_k ja a_n asuvad informatsiooniblokis andmed massiivi töötlemispiirkondade kohta. Need andmed (iga töötlemispiirkonna algusaadress a'_n ja lõppaadress a'_k) asuvad kahes pesas järgmiselt:

00 a'_k 0
00 a'_n 0.

Töötlemispiirkonnas jäetakse muutmata sisemised aadressid siis, kui andmed on esitatud kujul

00 a'_k 0
—00 a'_n 0.

Ainult sisemisi aadresse töödeldakse aga piirkonnas siis, kui informatsioon tema kohta on antud kujul

00 a'_k 0
40 a'_n 0.

Mõlema tunnuse, «—» ja «4», kooskasutamisel, s. t. informatsiooni esitamisel kujul

00 a'_k 0
—40 a'_n 0

aadresse üldse ei töödelda, s. t. töötlemispiirkond jääb ära. Viimast kuju kasutatakse just massiivi silumisel.

Väliste aadresside kasutamisel tuleb informatsiooniblokis anda lisaks eelnenule veel andmed iga selles massiivis M_i kasutatava väliste aadresside diapasooni kohta. Teistest väliste aadresside diapasoonidest eraldi asuva, massiivi M_j juurde kuuluvate väliste aadresside diapasooni kohta käivad andmed paigutatakse nelja pessa allnäidatud järjekorras:

00 b_{kj} 0
00 Δ_j 0
00 b_{nj} 0
N_j 4.

Diapasooni algusaadress b_{nj} ja lõppaadress b_{kj} peavad olema paarisarvud. Massiivi M_j number N_j antakse koos massiivinumbrit näitava tunnusega $\beta = 4$.

Vaatleme teises pesas antud nn. korrekteeriva suuruse Δ_j kasutamist. Massiivi M_j juurde kuuluvate väliste aadresside dia-

pasooni (seega suuruste b_{nj} ja b_{kj}) valik ei sõltu selle massiivi M_j enese alguspesast a_{nj}. Diapasooni aadresside ja massiivi M_j aadresside vastavuse loomiseks fikseerime diapasooni algusaadressile b_{nj} vastava pesa. Tavaliselt võetakse selleks pesaks a_{nj}, üldjuhul aga pesa aadressiga $a_{nj} + \Delta_j$. Seega suurus Δ_j peab näitama diapasooni algusaadressile b_{nj} vastava pesa kaugust massiivi alguspesast a_{nj}. Kui massiivi M_j juurde kuulub mitu aadresside diapasooni, siis valitakse Δ_j tavaliselt nii, et ühe diapasooni korral $\Delta_j = 0000$.

Paranduste sisseviimisel massiivi M_j on võimalik vältida juurdetoodud programmiosale järgnevate koodide käsitsi ümberadresseerimist. Siin saab suuruste Δ_i valimisega ja vastavate diapasoonide moodustamisega lasta parandustest tingitud adresseerimine ära teha koostaval programmil.

Väliste aadresside diapasoonide kohta käivat informatsiooni saab lühemalt üles kirjutada siis, kui kahe diapasooni korral $\Delta_j = \Delta_l = 0$ ja nad asuvad kõrvuti, s. o. ühe lõpp-punkt ühtib teise alguspunktiga, näiteks $b_{kj} = b_{nl}$. Andmed võib siis paigutada järgmisel viisil:

00	b_{kl}	0
00	0000	0
00	$b_{nl} = b_{kj}$	0
	N_l	4
00	b_{nj}	0
	N_j	4.

Diapasoonide niisuguse asetuseni võib jõuda eesmise diapasooni lõpp-punkti b_{kj} suurendamisega, eeldusel, et *pikendatud* diapasooni ei satu mõne töötlemispiirkonda kuuluva konstandi aadress.

Väliste aadresside diapasoonide järjekord informatsiooniblokis on vabalt valitav.

Informatsiooni üleskirjutuse kontrolliks tuleb jälgida, et massiivide numbrid N_j paikneksid paarisarvulistes pesades.

Informatsiooniblokk algab alati nn. l õ p u t u n n u s e g a Z paarituarvulises pesas (lõputunnuseks nimetatakse teda sellepärast, et koostav või interpreteeriv programm vaatab informatsiooniblokis olevad andmed läbi tagant ettepoole). Selle tunnuse võib esitada kahel kujul. Tavaliselt valitakse

$$z = -00\ 0000\ 4.$$

Mittestandardse formeeriva programmiga massiivil peab aga lõputunnuseks olema käsk

$$z = -22\ a_j\ 4,$$

mis annab juhtimise formeeriva programmi sisendpesale a_j.

§ 18. MASSIIVIDE BIBLIOTEEK

1. Massiivide biblioteegis hoitakse korduvalt kasutatavaid massiive, nn. biblioteegimassiive. Põhiprogrammi massiiv koostatakse tavaliselt ühekordseks kasutamiseks, seepärast ei ole niisuguse massiivi koostamisele praktiliselt otstarbekas erilisi kitsendusi seada. Seevastu biblioteegimassiivide koostamisel tuleb programmeerijal arvestada rea nõuete ja soovitustega: biblioteegimassiiv peab olema ökonoomne (lühike ja kiirelt töötav) ning ühtlasi lihtsalt kasutatav. Et programmeerijal oleks biblioteeki lihtsam kasutada, esitataksegi biblioteegimassiivile teatavad standardsuse nõuded.

Biblioteegimassiivi poole pöördumisel peab põhiprogramm ette andma lähteinformatsiooni (andmed) ja suunama juhtimise biblioteegimassiivi sisendile. Koos muu arvulise informatsiooniga peab massiiv saama andmed ka selle kohta, kuhu salvestada tulemused ja millisele põhiprogrammi pesale juhtimine tagasi suunata.

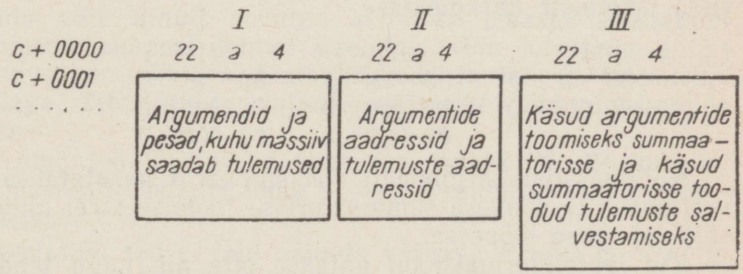
Juhtimise andmiseks biblioteegimassiivile kasutatakse vaatluseloleva süsteemi korral käsku

$$(c) = 22\ a\ 4,$$

kus a on seega väline aadress. Käsu täitmisel toimub tagasipöördumiskäsu 22 $c + 1$ 0 kirjutamine pessa a ja juhtimine läheb pesas $a + 1$ olevale käsule.

Massiivi poole pöördumise viisi valikul tuleb arvestada massiivile vajalike argumentide ja tema poolt antavate tulemuste hulgaga. Ühe argumendi ja tulemusega massiivi juures kasutatakse informatsiooni vahetamiseks summaatorit: massiivi poole pöördumisel peab argument olema summaatoris, töö lõpul, enne juhtimise andmist tagasipöördumiskäsule pesas a, tuuakse summaatorisse tulemus. Ühe argumendi ja tulemusega biblioteegimassiivideks on näiteks elementaarfunktsioonide arvutamise alamprogrammid.

Kui argumente või tulemusi on rohkem kui üks (täpsemalt, ei mahu ühte topeltpessa), siis informatsiooni vahetamiseks sum-



JOON. 40

maatorist ei piisa. Vaadeldavas süsteemis kasutatakse sel juhul pöördumiskäsule 22 *a* 4 järgnevaid pesi. Pöördumine massiivi poole võib toimuda joonisel 40 näidatud kolmel viisil. Näiteks kolme argumenti (n, A_1, A_2) nõudva ja kahte tulemust (R_1, R_2) andva massiivi poole võib pöörduda ühega järgmistest programmilõikudest:

	I	II	III
$c+0000$	22 *a* 4	22 *a* 4	22 *a* 4
$c+0001$	*n*	00 ⟨*n*⟩ 0	02 ⟨*n*⟩ 0
$c+0002$ } $c+0003$	A_2	00 ⟨A_2⟩ 0	42 ⟨A_2⟩ 4
$c+0004$ } $c+0005$	0	00 ⟨R_2⟩ 0	00 ⟨R_2⟩ 4
$c+0006$			

Argument A_1 ja tulemus R_1 antakse selliste pöördumiste korral edasi summaatori kaudu. Loomulikult peab iga konkreetse massiivi korral üks nendest pöördumisviisidest olema fikseeritud. Pealegi eeldatakse, et pöördumiskäsu järel asuvate, informatsiooni vahetamiseks ettenähtud pesade arv i on massiivi jaoks konstantne suurus. Pärast tulemuste salvestamist annab massiiv juhtimise informatsioonile järgnevale käsule (toodud näite I juhul pesas $c+0006$, II ja III juhul pesas $c+0004$ olevale käsule), mis toimub näiteks sel teel, et massiiv algab käskudega

$(a-1) = 30 \quad i \quad 0$
$(a) \quad = 00 \quad 0000 \quad 0$

ning lõpeb suunamisega pesas $a-1$ paiknevale käsule. Teiseks võimaluseks on massiivi töö lõpetamine käskudega

30 *a* 0
00 *i* 0.

Et juhtimine on antud massiivile käsuga 22 *a* 4, siis saab massiivis argumentide ja tulemustega opereerimiseks kasutada pesasse *a* kirjutatud käsku 22 $c+1$ 0. Nii võime äsjavaadeldud näite korral massiivi töö käigus tuua argumendi A_2 summaatorisse järgmiste käskudega:

I	II	III
30 *a* 0	30 *a* 0	30 *a* 0
−60 0001 0	06 0001 0	06 0001 0
	02 0000 4	00 0000 0

Kui massiivis argumenti rohkem kordi kasutatakse, siis salvestatakse ta esimese summaatorisse toomise järel muidugi massiivi mingisse tööpessa.

Kui lähteinformatsioon antakse ette nii, nagu toodud näites juhul III, siis tuleb massiivis kasutada formeerivat osa, mis pai-

gutab lähteinformatsioonina esitatavad käsud (neid nimetatakse ka p s e u d o k ä s k u d e k s) laiali vajalikele kohtadele massiivis. Massiivi võib sel juhul koostada nii, et teda saab ühe pöördumisega kasutada terve algandmete tabeli töötlemiseks, salvestades ühtlasi tulemused omaette tabelisse (s. o. järjestikustesse pesadesse). Selleks kasutatakse pseudokäskude ees märke «—» ja biblioteegimassiivis tsükliloendaja sama seisu, mis on enne pöördumist. Nii tuleb ülalvaadeldud näites argumentide A_1 ja A_2 tabeli töötlemiseks (ning tulemuste R_1 ja R_2 salvestamiseks tabelisse) ette anda informatsioon tsüklis

$$\begin{array}{rrl}
 & 25 & k \quad 4 \\
\rightarrow & -42 & \langle A_1 \rangle \quad 4 \\
 & 22 & a \quad 4 \\
 & 02 & \langle n \rangle \quad 0 \\
 & -42 & \langle A_2 \rangle \quad 4 \\
 & -56 & \langle R_2 \rangle \quad 4 \\
 & -56 & \langle R_1 \rangle \quad 4 \\
- & -24 &
\end{array}$$

kus tabeli pikkus on $(k+2):2$.

Tabelis paiknevate argumentide või tulemuste korral võib veel kasutada sellist pöördumisviisi, kus lähteinformatsiooniks on ainult tabeli alguspesa. Näiteks massiivi poole, kus argumendid paiknevad alates pesast t ja tulemuseks on üksainus arv, võib pöördumise realiseerida käskudega

$$\begin{array}{l}
c+0000\ 22\ a\ 4 \\
c+0001\ 00\ t\ 0.
\end{array}$$

Mõningate massiivide puhul võib neile informatsioonina etteantavate pesade arv olla muutuv. Neil juhtudel tuleb tagasipöördumiskäsu moodustamiseks informatsioonis ette anda informatsioonipesade arv.

2. Lisaks pöördumisviiside valiku kindlustamisele tuleb antud süsteemi biblioteegimassiivi koostamisel arvestada veel järgnevaid nõudeid.

Kasutada võib ainult arvutile «Ural-4» tehases ettenähtud käske. Arvutis kohapeal tehtud uuendusi võib kasutada ainult põhiprogrammi massiivides. Uute käskude ja muude täienduste sisseviimisega võib tekkida vajadus süsteemi põhiprogrammide muutmiseks.

Pärast oma töö lõpetamist peab iga tsüklit 25—24 kasutav biblioteegimassiiv taastama tema poole pöördumisel olnud tsükliloendaja seisu.

Ühegi massiivi töö ei tohi sõltuda võtmete *Peatus φ järgi* ja *φ blokeerimine* asenditest arvuti juhtimispuldil. Võimaluse piires tuleb vältida arvuti peatumist põhjusel *jagamine nulliga.*

Massiivi maksimaalne lubatud pikkus (koos informatsiooniblokiga) on 3776 pesa.

Ei ole soovitav kasutada massiivis võtmeid, s. o. käsku 23 *n*, sest mitme võtmeid tarvitava biblioteegimassiivi üheaegsel kasutamisel võib tekkida vajadus lahendamise ajal võtmeid ümber lülitada. Sellest kui tülikast ja aeganõudvast operatsioonist tuleb aga igati hoiduda. Kui massiivis siiski võtmeid kasutatakse, siis tuleks nende numbrid jätta pöördumisel etteantavateks: sel juhul saab programmeerija jaotada võtmeid 1 kuni 7 kogu programmi ulatuses. Ühtlustamise eesmärgil soovitatakse lahendamise kontrolli ümberlülitamiseks kasutada võtmeid tabelis 27 näidatud korras.

Tabel 27

Võtme nr.	Võti sees	Võti väljas
1	Kui kontroll avastab vea, korratakse lahendamist	Peatus, kui kontroll avastab vea
7	Kontroll on välja lülitatud	Kontroll on sisse lülitatud

Iga biblioteegimassiiv peab olema varustatud informatsiooniblokiga.

3. Kõiki biblioteegimassiive, mis kogusummas moodustavadki massiivide biblioteegi, hoitakse arvuti välises mälus — magnettrumlil või magnetlindil. Programmi formeerimisel koostava programmi abil tuleb seega perfokaartidelt operatiivmällu tuua ainult põhiprogrammi massiive. Operatsioonid biblioteegiga, nagu massiivide juurdeviimine biblioteeki ja massiivide kontroll, realiseeritakse süsteemis selleks ettenähtud erilise programmiga «Valvur-44». Sama programmiga moodustatakse biblioteegi juurde ka andmed kõigi massiivide kohta, mis paigutatakse magnettrumlil nn. püsivandmete tabelisse ehk kataloogi. Sellesse kantavad andmed moodustavad ühe osa koostavale (interpreteerivale) programmile vajalikust informatsioonist. Siia kuuluvad just andmed massiivi asukoha ja tema osade pikkuste kohta. Programmeerijal tuleb ülesande lahendamisel analoogilised andmed anda ainult põhiprogrammi massiivide kohta.

Püsivandmete kataloog koosneb kahest osast, nagu on skemaatiliselt näidatud tabelis 28.

Massiivide numbrite järjestus kataloogis on sealjuures suvaline. Andmed massiivi number $N^{(k)}$ kohta asuvad aga kolmes järjestikuses pesas, alates pesast $a_T + 3k$. Asugu massiivi M_i andmed kataloogis k-ndal kohal, s. t. $N^{(k)} = N_i$. Kataloogi II osas paiknevad siis massiivi M_i kohta joonisel 41 kujutatud andmed, kus märgikohal asuv tunnus π_i näitab, kas massiiv M_i on mag-

I	$a_T + 0000$	$-00\ a'_T\ \ 0$	Tabeli II osa algusaadress
	$a_T + 0001$	$+\ N^{(1)}$	Massiivide numbrid (tabeli I osa)
	. . . . .	. . . .	
	$a_T + n$	$+\ N^{(n)}$	
	$a_T + n + 0001$	00 0000 0	Tabeli I osa lõpu tunnus
II	$a'_T + 3 + 0000$ $+ 0001$ $+ 0002$		Massiivi number $N^{(1)}$ andmed
	$a'_T + 6 + 0000$ 0001 0002		Massiivi number $N^{(2)}$ andmed
	. . . . .	. . . . .	
	$a'_T + 3n + 0000$ 00001 0002		Massiivi number $N^{(n)}$ andmed
	$a'_T + 3(n+1)$		Tabeli kontrollsumma

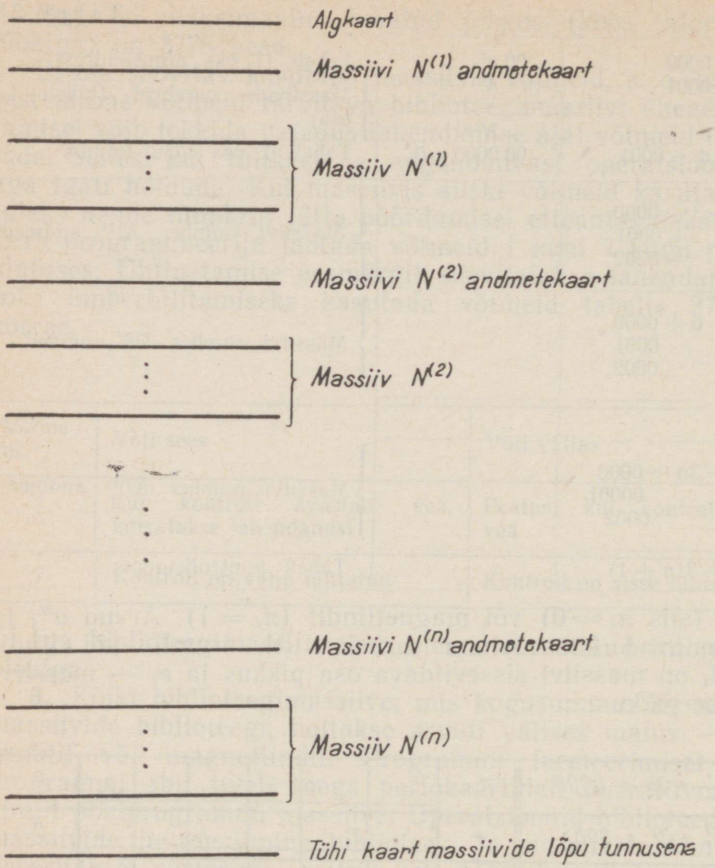
nettrumlil (siis $\pi_i = 0$) või magnetlindil ($\pi_i = 1$). Arvud a''_i ja a'_i moodustavad koosvõetuna massiivi M_i trumli- või lindiaadressi; l_i on massiivi sisseviidava osa pikkus ja e_i — massiivi töötava osa pikkus.

$a'_T + 3k + 0000$	π_i			a''_i				0
$a'_T + 3k + 0001$	+		a'_i		l_i			0
$a'_T + 3k + 0002$	+	0	0			e_i		0

JOON. 41

Nende andmete põhjal saab koostav või interpreteeriv programm tuua massiivi biblioteegist sisemälusse ja massiivi informatsiooniblokki kasutades formeerida massiivi aadressid. Rõhutame, et kataloogi pole tarvis spetsiaalselt perforeerida ja mälusse viia — selle moodustab programm «Valvur-44» biblioteegi mälusseviimise käigus.

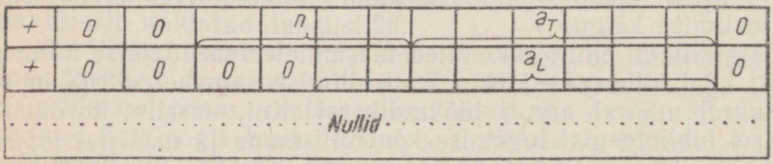
4. Massiivide biblioteeki tuleb ülesannete lahendamise käigus pidevalt kontrollida, kusjuures kontrollimise sagedus sõltub muidugi suurel määral arvuti töökindlusest. Kui massiivi korduval lugemisel biblioteegist lugemise kontrollsumma ja massiivi lõpus asuv kontrollsumma kordagi kokku ei lange, siis uuendatakse kogu biblioteek.



JOON. 42

Biblioteegi uuendamiseks viiakse massiivid arvutisse hoolikalt kontrollitud perfokaartidelt, mida edaspidi nimetame kaartide e t a l o n p a k i k s. Etalonpaki sisu on kujutatud joonisel 42.

Paki esimesel kaardil, nn. a l g k a a r d i l, on programmile «Valvur-44» vajalikud üldandmed biblioteegi suuruse ja asukoha kohta. Need andmed kantakse algkaardile joonisel 43 näidatud



JOON. 43

järjekorras, kus n tähendab püsivandmete tabelisse kantavate massiivide arvu (võib olla ka suurem tegelikult sisseviidavate massiivide arvust, aga peab olema väiksem kui $419_{(10)} = 0643_{(8)}$), a_T on magnettrumli pesa aadress, millest alates moodustatakse püsivandmete kataloog, ja a_L — magnetlindi aadress, millest alates salvestatakse sisseviidavad massiivid. Magnettrumlile viidavate massiivide jaoks algusaadressi ette ei anta. Need massiivid kantakse kataloogile vahetult järgnevatesse pesadesse, alates pesast $a + 4n + 4$.

Iga sisseviidava massiivi kaartide ees paikneb nn. andmetekaart, mis näitab, kuidas massiiv on perfokaartidele kantud, kui pikk ta on ja kuhu teda salvestada tuleb (nagu edaspidises selgub, saab sama kaarti kasutada ka koostava programmi juures). Massiivi M_i jaoks täidetud andmetekaart on näidatud joonisel 44,

−				N_i			0	0			e_i		α_i
+	0	0	0	0	0	0	0	0			l_i		γ_i
													H_i
.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....
.....	.....	.....	.....	.....	Nullid	.....	.....	.....	.....	.....	.....	.....	.....

JOON. 44

kus N_i on massiivi number; e_i — massiivi töötava osa pikkus; l_i — massiivi sisseviidava osa pikkus; H_i — tunnus, mis näitab kas massiiv tuleb salvestada magnettrumlile ($H_i = 2$) või magnetlindile ($H_i = 3$); γ_i — konstant, mis näitab, kuidas massiiv on kaartidele perfoereritud ($\gamma_i = 0$ — pesade kaupa, ilma kontrollsummadeta (12-le reale); $\gamma_i = 1$ — pesade kaupa, kontrollsummadega; $\gamma_i = 4$ — topeltpesade kaupa, ilma kontrollsummadeta (12-le reale); $\gamma_i = 5$ — topeltpesade kaupa, kontrollsummadega) ja α_i — konstant, mida kasutab ainult koostav programm (siin võib võtta $\alpha_i = 1$).

5. Programmil «Valvur-44» on kaks töörežiimi: kirjutamine ja kontrollimine. Režiim valitakse arvuti juhtimispuldil võtme nr. 2 abil. Kirjutamise korral (võti 2 sisse lülitatud) viiakse kõik etalonpakis olevad massiivid magnettrumlile, algkaardiga määratud pesadest alates. Kontrollimise korral arvutatakse vaid kontrollsummad ning võrreldakse neid etteantutega. Avastanud vea biblioteegi mingis massiivis või püsivandmete tabelis, läheb programm ise üle kirjutamise režiimi.

Kui etalonpaki algkaardil on massiivide arv n suurem kui 419, siis trükitakse märkus

ВЕЛИКА

ja arvuti peatub (vt. tabel 29).

Töö käigus trükitakse alati kõigepealt välja tabeli pea kujul

РАБОТАЕТ СТОРОЖ-44
НОМЕР ГДЕ НАЧАЛЬНЫЙ ВВОД СУММА РАБОЧ.
МАССИВА РАСП. АДРЕС ДЛИНА ВВОДА ДЛИНА

Tabel täidetakse ainult kirjutamise režiimis töötamise korral. Vasakult esimesse lahtrisse trükitakse mällukantud massiivi number. Kui massiiv kirjutatakse magnettrumlile, siis trükitakse

Tabel 29

Peatuse aadress	Peatumise põhjus	Lahendamise jätkamine
0021	Viga programmi «Valvur-44» sisseviimisel	Sisseviimise jätkamiseks arvuti käivitada; kontrollida kaarti
0202	Viga etalonpaki kaardi sisseviimisel	Kaardi sisseviimiseks panna ta uuesti sisendile ja arvuti käivitada; kontrollida kaarti
0217	Sisseviimise kontrollsumma erineb massiivi kontrollsummast	Massiivi sisseviimiseks panna massiiv koos peakaardiga uuesti sisendile ja arvuti käivitada
0372 (lukuga) [1]	Programmi töö lõpp	—
0425 (lukuga)	Massiivide arv n algkaardil on väiksem kui massiivide arv sisendil	Katkestada biblioteegi koostamine, suurendada n
0552 (lukuga)	Programm «Valvur-44» valesti sisse toodud	«Valvur-44» algkäivitusega uuesti sisse viia
0605 (lukuga)	Algkaardil $n > 419$	Katkestada biblioteegi koostamine, vähendada n

[1] Lukuga peatuseks nimetatakse peatust kujul
$(a + 0) = 37\,0000\ \ 0$
$(a + 1) = 22\ a + 0\ 0.$

teise lahtrisse МБ, kui magnetlindile, siis МЛ. Kolmandasse lahtrisse trükitakse biblioteeki viidud massiivi algusaadress (trumli või lindi pesa aadress). Andmetekaardilt võetud suurused l_i ja e_i (vt. joonis 44) trükitakse vastavalt neljandasse ja kuuendasse lahtrisse. Viiendasse trükitakse aga kas massiivi sisseviimisel arvutatud kontrollsumma, või märk «=», kui see summa ühtib massiivi lõpus antud kontrollsummaga.

Kui etalonpakist sissetoodud massiivide arv ületab algkaardil etteantud arvu n, siis trükitakse

ОШИБКА

ja arvuti peatub. Kirjutamise normaalse lõpu tunnuseks trükitakse

КОНЕЦ РАБОТЫ

Sama tunnuse trükkimisega vahetult tabeli pealdise alla lõpetab programm oma töö ka kontrolli režiimis.

Programmi «Valvur-44» töö käigus võivad esineda tabelis 29 näidatud peatused.

§ 19. KOOSTAV PROGRAMM JA INTERPRETEERIV PROGRAMM

1. Vaadeldavasse süsteemi kuulub koostav programm «ССП-42», mille abil saab ülesande lahendamise programmi kokku panna üksikutest massiividest. Järgnevas vaatlemegi, kuidas see toimub.

Kõik kasutatavad massiivid peavad olema 17. paragrahvis kirjeldatud ehitusega. Parameetrist sõltuvate väliste aadressidega massiive koostav programm ei modifitseeri, küll aga võib kasutada mittestandardset formeerivat programmi massiivi aadresside muutmiseks.

Massiivid, milledest programm koostatakse, võivad olla eelnevalt perfokaartidel, magnettrumlil või magnetlindil. Biblioteegimassiivid peavad sealjuures olema magnettrumlil või magnetlindil, püsivandmete tabel neid iseloomustavate parameetritega aga magnettrumlil.

Ülesande lahendamise programm koostatakse sisemälu pesadesse, kusjuures eeldame, et koostatav programm mahub selleks kasutada olevatesse pesadesse. Paragrahvi lõpuosas aga näeme, et programmi koostamist saab automatiseerida ka vastasel juhul.

Koostav programm kasutab sisemälu pesi järgmiselt. Pesas 0000 on koostava programmi töö ajal (ja peab olema ka biblioteegiprogrammide jaoks) arv null. Pesad 0001—0033 on kasutusel algkäivituse ajal ja koostava programmi tööpesadena. Koostav programm viiakse algkäivitusega pesadesse 0034—0776. Pesast 7776 ettepoole paigutab koostav programm informatsiooni

massiivide kohta, keskmiselt kaks topeltpesa igale massiivile (programmi koosseisu kuuluvate massiivide arv ei ole koostava programmi poolt piiratud). Mälu ülejäänud ossa, pesast 1000 kuni informatsioonipesadeni, formeeribki koostav programm ülesande lahendamise programmi. Väljapoole neid pesi võib paigutada ainult koostatava programmi tööpesade massiive. Programmi formeerimise käigus koostav programm pidevalt kontrollib, kas formeeritav programm ei ületa lubatud pikkust.

Koostavale programmile vajalikud andmed on jaotatud kahte ossa. Üks osa asub massiivide lõpus — informatsiooniblokkides. Teine osa andmeid antakse aga ette perfokaartidelt ja koostav programm viib need ise sisemällu.

Järgnevalt vaatlemegi üksikasjalikumalt just andmete teist osa, mida nimetame *muutuvandmete tabeliks*. Iga massiivi kohta peab koostav programm nimelt saama järgmise informatsiooni:

a) kuhu tuleb massiiv paigutada (massiivi alguspesa aadress),
b) massiivi number,
c) massiivi töötava osa pikkus,
d) kas massiivi tuleb töödelda,
e) massiivi sisseviidava osa pikkus,
f) massiivi sissetoomise koht ja toomisviis.

Kui kahele või enamale massiivile on sattunud sama number (mida küll harva juhtub), tuleb antavale informatsioonile lisada veel

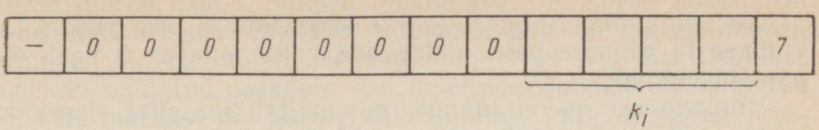
g) uus number massiivile (nn. numbriparandus).

2. Vaatleme nüüd, kuidas massiivid formeerimise käigus programmi paigutatakse ja missugune osa on seejuures massiivi alguspesa aadressil.

Programmi koostamisel jaotatakse kogu programm massiividekss, kusjuures seosed massiivide vahel luuakse enamasti väliste aadresside abil. See annab üsna vabad käed massiivide paigutamiseks, ühtlasi aga võimaluse jätta nende tegelik paigutamine koostava programmi ülesandeks. Mõnede massiivide jaoks tuleb siiski ette anda täpne asukoht — massiivi alguspesa aadress sisemälus. Kui mingi massiivi kohta tema alguspesa aadressi ette antud ei ole, siis võtab koostav programm selleks eelmise sisemällu viidud massiivi töötavale osale järgneva paarisarvulise pesa aadressi.

Massiivi M_i alguspesa aadress k_i antakse ette ühes topeltpesas, joonisel 45 näidatud viisil, kus «—» märgikohal ja «7» viimasel kaheksandkohal ongi tunnuseks, et tegemist on just alguspesa aadressiga.

Juhul kui muutuvandmete tabelis esimese massiivi andmete ees puudub alguspesa aadress, võtab koostav programm algus-

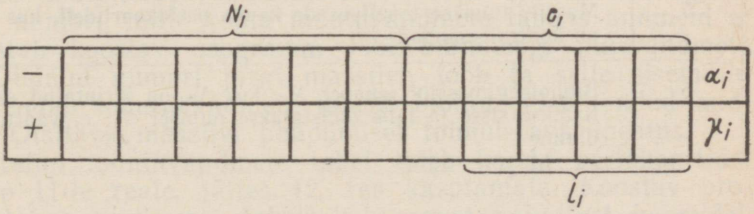


JOON. 45

aadressiks 1000 (pesades 0001—0776 asub koostav programm koos oma tööpesadega). Seega alguspesa aadresse $k_i < 1000$ võib kasutada ainult tööpesade massiivide korral. Võimalik on neid kasutada ka formeeritava programmi enda poolt sissetoodava informatsiooni salvestamisel.

Koostava programmi ehitusest on tingitud nõue, et kui viimane sisseviidav massiiv paigutatakse formeeritava programmi algusesse või keskele, siis tuleb muutuvandmete tabeli lõppu kirjutada veel algusaadress, mis näitaks massiivide tegelikku lõppu sisemälus.

Järgnevad massiivi kohta käivad suurused on sarnased massiivi biblioteeki viimisel andmetekaardile kantavate suurustega (vt. § 18). Nende massiivi M_i kohta vajalike andmete paigutus on näidatud joonisel 46, kus märk «—» on esimese pesa tunnuseks (et iga massiivi puhul alguspesa aadressi ei kasutata, siis võime pesa märgiga «—» lugeda esimeseks).



JOON. 46

Esimene arv N_i on massiivi number. Meenutame, et põhiprogrammi massiividele määrab programmeerija ajutised numbrid, biblioteegimassiividele on aga juba varem omistatud kindlad numbrid.

Teine arv e_i on massiivi töötava osa pikkus. See suurus peab olema paarisarv ja näitama mitu pesa võtab massiiv töökorras olles sisemälus enda alla. Koostavas programmis on ette nähtud võimalus omistada suurusele e_i ka negatiivseid väärtusi: sel juhul kirjutatakse esimesele kaheksandkohale arv 4, näiteks $e_i = -5300$ korral kirjutatakse 405300.

Esimese topeltpesa lõpus olev α_i on massiivi ümbertöötamise tunnus: kui $\alpha_i = 0$, siis massiivi ei töödelda (töötlemist ei vaja konstantide ja tööpesade massiivid), kui aga $\alpha_i = 1$, siis massiivi töödeldakse tema informatsiooniblokis antud andmete järgi.

Järgmine arv l_i on massiivi sisseviidava osa pikkus, s. o. viimase ja esimese pesa aadresside vahe (suurus l_i peab olema paarisarvuline).

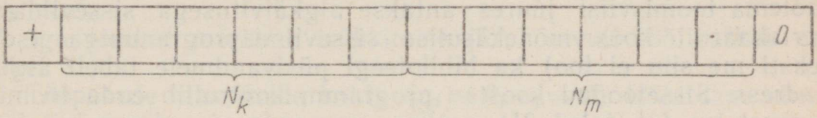
Viimane arv y_i on tunnus, mis näitab massiivi sissetoomise kohta ja viisi. Kasutatavate y_i väärtuste tähendused on toodud tabelis 30.

Tabel 30

y_i	Sissetoomise kirjeldus
0	Massiiv tuuakse pesade kaupa perfokaartide kõigilt 12-lt realt
1	Massiiv tuuakse pesade kaupa perfokaartidelt, kus 12. reas on kontrollsummad
2	Massiiv tuuakse magnettrumlilt; sel juhul peab l_i ees joonisel 46 kujutatud teise topeltpesa 3.—8. kaheksandkohal olema massiivi alguspesa trumliaadress
3	Massiiv tuuakse magnetlindilt; l_i ees, topeltpesa 1.—8. kahek sandkohal peab olema massiivi alguspesa lindiaadress
4	Massiiv tuuakse topeltpesade kaupa perfokaartide kõigilt 12-lt realt
5	Massiiv tuuakse topeltpesade kaupa perfokaartidelt, kus 12. reas on kontrollsummad
6	Biblioteegimassiiv number N_k, kus N_k on kirjutatud e_i kohale, tuuakse sisse ja talle omistatakse number N_i; e_i väärtus ei ole oluline

Harukordadel võib joonisel 46 näidatud kahele topeltpesale mõne massiivi puhul lisanduda veel üks või enam topeltpesa. Seda nimelt sel juhul, kui ülesande koostamise käigus on erinevatele massiividele ekslikult omistatud sama number. Vajaliku üksühese vastavuse taastamiseks massiivide ja numbrite vahel tuleb osa massiive sel juhul ümber nummerdada. Neile tuleb anda mingid selles ülesandes kasutamata ajutised numbrid (vt. § 17). Mingile massiivile numbriga N_k uue numbri N_m omistamisel tuleb N_k asendada numbriga N_m ka kõigi nende massiivide informatsiooniblokkides, kus esineb selle massiivi N_k (hiljem N_m) juurde kuuluvaid väliseid aadresse. Niisuguse asenduse läbiviimine programmeerija poolt siis, kui massiivid on juba koostatud, osutub üsna tülikaks, sest parandused informatsiooniblokkides muudavad ka massiivide kontrollsummasid. Seepärast ongi ette nähtud võimalus, et koostav programm ise töö käigus, enne aadresside modifitseerimist, teeb vajalikud parandused. Selleks tuleb vaid parandamisele kuuluva informatsiooniblokiga massiivi muutuvandmete

tabelisse juurde lisada andmed muudetavate numbrite kohta. Kui eelvaadeldud massiivi M_i informatsiooniblokis on mingil massiivil number N_k, mis tuleb asendada numbriga N_m, siis lisatakse joonisel 46 näidatud pesadele veel topeltpesa joonisel 47 näidatud kujul. Ühe massiivi M_i juurde võib kuuluda mitu niisugust pesa.



JOON. 47

Muutuvandmete tabelis ühe massiivi kohta käivate andmete viimase pesa tunnuseks on järgmise pesa märk «—» või tühi pesa (tabeli lõpus). Kui tabelis teine pesa suurustega l_i ja γ_i puudub, siis loeb koostav programm need võrdseks nulliga. Seega töö-pesade massiivi jaoks piisab tabelis ühest, alguspesa aadressi etteandmisel aga kahest pesast.

3. Iga biblioteegimassiivi ei ole tarvis sisse tuua eelkirjeldatud andmete etteandmisega. Piisab ka sellest, kui biblioteegimassiivi number esineb mingi massiivi informatsiooniblokis. Koostav programm vaatab nimelt läbi kõigi muutuvandmete tabelis antud massiivide informatsiooniblokid. Leidnud niisuguse massiivi numbri, mille kohta muutuvandmete tabelis andmeid ei ole, vaatab koostav programm läbi biblioteegi püsivandmete tabeli. Otsinud numbri järgi massiivi, toob ta selle sisemällu ning täiendab muutuvandmete tabelit biblioteegist saadud andmetega. Otsitava massiivi puudumisel toimub avariipeatus.

Koostatud muutuvandmete tabel tuleb kanda perfokaartidel esimesele 11-le reale, jättes 12. rea kasutamata. Koostav programm lubab sealjuures tabeli kahesugust paigutust kaartidel. Esimesel viisil perforeeritakse iga massiivi kohta käivad andmed eraldi kaardile, alates kaardi esimesest reast. Teisel viisil perforeeritakse tabel järjest kaartide kõigile 11-le reale. Lubatud on ka kasutada segamini mõlemaid viise. Esimene perforeerimisviis nõuab küll rohkem kaarte, seevastu aga võimaldab massiivi hilisemal mõnda teise programmi lülitamisel kasutada sama kaarti. Ühtlasi annab esimene viis võimaluse kasutada juba biblioteegi koostamise juures perforeeritud andmetekaarte.

Tabeli lõpu tunnusena tuleb lisada kaart, mille esimeses reas on nullid. Võib kasutada ka tühja (perforeerimata) kaarti. Koostav programm trükib alati pärast formeerimist välja saadud programmi kontrollsumma. Programmi korduva formeerimise korral on soovitatav tabeli viimase kaardi teise topeltpesa algusse (1.—7. kaheksandkohale) perforeerida eelmisel formeerimisel saadud kontrollsumma.

Formeeritud programmi väljatrükkimiseks tuleb koostava programmi töö ajaks sisse lülitada võti nr. 3.

4. Koostava programmi töö algab selle programmi enda toomisega sisemällu, pesadesse 0034—1326. Tavaliselt paikneb ta eraldi massiivina biblioteegis. Kui aga koostavat programmi biblioteeki ei ole viidud, siis tuuakse ta sisse perfokaartidelt. Mõlema toomisviisi juures antakse algkäivitusega sisseviidaval perfokaardil koos mõnekäsulise sisseviimisprogrammiga (selle teksti me siin ei too) ka biblioteegi püsivandmete tabeli algusaadress. Sissetoodud koostav programm kontrollib enda toomise õigsust ise (vt. tabel 31).

Koostavat programmi sisse viiva kaardi järel peavad asuma kaardid muutuvandmetega (ja nende järel kaart, mille esimeses reas on nullid). Järgnevad kaardisisendilt sisseviidavate massiivide (eeskätt põhiprogrammi massiivide) kaardid. Massiivide järjekord sisendil peab ühte langema massiivide järjekorraga muutuvandmete tabelis.

Töö käigus trükib koostav programm välja rea andmeid. Algul väljatrükitud pealdise

РАБОТАЕТ ССП-42

НОМЕР НАЧАЛЬНЫЙ ОТКУДА ВВОД СУММА СУММА ОШИБКА
МАССИВА АДРЕС НФ ВВЕДЕН ДЛИНА ВВОДА ОБРАБ

alla trükitakse iga massiivi kohta (vasakult paremale):

1) massiivi number,
2) massiivi algusaadress sisemälus,
3) arvuti seade, millelt massiiv on sisse toodud:
   ПК — perfokaartidelt toomisel,
   МБ — magnettrumlilt toomisel,
   МЛ — magnetlindilt toomisel,
4) massiivi sissetoodava osa pikkus,
5) massiivi sissetoomisel arvutatud kontrollsumma, kui see aga võrdub massiivi lõpus asuvaga, siis märk «=»,
6) ümbertöötatud massiivi kontrollsumma,
7) üks kolmest allpool toodud märkusest, kui massiivi kohta antud informatsioonis esines viga.

Märkus HET 7. veerus koos temale järgneva numbriga N_j ütleb, et momendil modifitseeritava massiivi (mille number trükiti 1. veerus) väline aadress kuulub massiivi M_j juurde, kuid massiivi M_j kohta puuduvad andmed nii muutuvandmete tabelis kui ka biblioteegis. Sama märkus tabeli pealdisele järgnevas reas ja ilma numbrita 1. veerus ütleb, et muutuvandmete tabelis on juhul $\gamma_i = 6$ biblioteegimassiiv number N_j, mille kohta biblioteegis puuduvad andmed.

Märkus ИВР< koos järgneva aadressiga trükitakse siis, kui sissetoodava massiivi viimane pesa (mille aadress trükiti märkuses) langeb muutuvandmete tabelisse.

Kolmas märkus ИНФОРМ trükitakse siis, kui massiivi informatsiooniblokis puudub informatsiooni lõpu tunnus või on valesti ette antud mõni töötlemispiirkond, s. t. pole täidetud nõuet $a'_k > a'_n$.

Iga toodud märkuse trükkimise järel arvuti töö katkeb (vt. tabel 31).

Tööpesade massiivide kohta ($a_i = 0$, $e_i = 0$) trükitakse ainult massiivi number ja alguspesa aadress.

Tabel 31

Peatuse aadress	Peatumise põhjus	Formeerimise jätkamine
0021	Viga «ССП-42» sissetoomisel perfokaartidelt või kaardil vale kontrollsumma	Töö jätkamiseks arvuti käivitada. Kontrollida «ССП-42» kaarte ja sissetoomiskaarti
0127 (lukuga)	Sisseviidava massiivi lõpp ulatub muutuvandmete tabelisse	Katkestada formeerimine, kontrollida muutuvandmete tabelit
0204	Massiivi M_i lugemisel magnettrumlilt või magnetlindilt saadi kolmel lugemisel valed kontrollsummad	Massiivi sissetoomise kordamiseks arvuti käivitada. Kontrollida massiivi M_i ja tema andmeid muutuvandmete tabelis
0237	Viga M_i sissetoomisel kaartidelt või kaardil vale kontrollsumma	Massiivi M_i sisseviimise kordamiseks tema kaardid uuesti sisendile asetada ja arvuti käivitada
0412 (lukuga)	Massiivil M_i on väline aadress, mis kuulub massiivi M_j juurde, kuid viimase kohta puuduvad andmed nii muutuvandmete tabelis kui ka biblioteegis	Katkestada formeerimine. Kontrollida muutuvandmete tabelit
0514 (lukuga)	Massiivi M_i informatsiooniblokis puudub informatsiooni lõpu tunnus z või pole täidetud nõuet $a'_k > a'_n$	Katkestada formeerimine. Parandada viga informatsiooniblokis
0720	Programm on formeeritud, koostav programm on töö lõpetanud	
0776 (lukuga)	Viga «ССП-42» sissetoomisel	Korrata sissetoomist: asetada sisendile kaardid, kustutada sisemälu, anda algkäivitus

Pärast massiivide sisseviimist ja modifitseerimist trükitakse tabeli lõppu КОНЕЦ РАБОТЫ ja seitsmendasse veergu formeeritud programmi kontrollsumma või märk «=», kui see summa langes kokku muutuvandmete tabeli viimasele kaardile perforeeritud kontrollsummaga.

Koostava programmi töö käigus võivad esineda tabelis 31 näidatud peatused.

5. Toome lõpuks veel ühe lihtsa näite koostava programmi tegeliku kasutamise kohta.

Olgu tarvis arvutada integraali

$$\int_a^b x^k \sin^{2m} x \, dx$$

väärtus etteantud täpsusega $\varepsilon > 0$. Selle integraali arvutamiseks kasutame biblioteegis olevat alamprogrammi number 150.200, mis realiseerib Simpsoni valemi. Pöördumine massiivi 150.200 poole mingist pesast $a + 0000$ peab toimuma käskudega

$a + 0000$ 22 b_1 4
$a + 0001$ 22 b_2 4
$a + 0002$ 42 $\langle \varrho \rangle$ 4
$a + 0003$ 42 $\langle \varepsilon \rangle$ 4
$a + 0004$ 42 $\langle a \rangle$ 4
$a + 0005$ 42 $\langle b \rangle$ 4

kus b_1 on alamprogrammi 150.200 algusaadress ferriidil; b_2 — integreeritava funktsiooni väärtusi arvutava alamprogrammi algusaadress ferriidil; ϱ — esialgne samm integreerimisel, mida alamprogramm ise töö käigus poolitab (kui vajalikku täpsust pole saavutatud); a — integreerimise alumine raja; b — integreerimise ülemine raja.

Alamprogramm 150.200 nõuab, et integreeritava funktsiooni väärtusi arvutava massiivi poole pöördumine toimuks käsuga

22 b_2 4,

kusjuures argument peab paiknema summaatoris. Selle massiivi (tema number on vabalt valitav) töö lõpul, juhtimise andmisel pesale b_2, peab summaatoris olema funktsiooni leitud väärtus. Alamprogrammi 150.200 töö lõpuks on arvutatud integraali väärtus summaatoris. Kõik arvutused toimuvad liikuva koma režiimis.

Koostame vaadeldava integraali arvutamise programmi ühe konkreetsema juhu jaoks, võttes $a = 0$; $b = \frac{\pi}{2}$ ja andes parameetritele m ning k järjest väärtused $m = 1, 2, 3$; $k = 0{,}5$; 0,6; ...; 0,9. Integreerimise täpsuseks valime $\varepsilon = 0{,}001$ ja esialgseks integreerimissammuks $\varrho = 0{,}1$.

Parameetrid k ja m ning nendele vastava integraali väärtused trükime kiirtrükkalil. Igale k ja m kombinatsioonile vastavad

arvukolmikud (k, m ja integraali väärtus) eraldame trükkimisel reavahega.

Ülesande lahendamise programmi koostame «ССП-42» abil seitsmest massiivist $M_1 \div M_7$:

M_1 — põhiprogrammi massiiv, mis juhib arvutuste käiku; omistame sellele ajutise numbri 777;

M_2 — põhiprogrammi massiiv integreeritava funktsiooni väärtuste arvutamiseks; ajutine number 776;

M_3 — alamprogramm 150.200 arvutab integraali väärtuse; selles programmis $l_3 = 0256$ ja $e_3 = 0276$;

M_4 — alamprogramm 000.330 funktsiooni $\cos x$ ($\sin x$) väärtuste arvutamiseks; selles programmis $l_4 = 0066$ ja $e_4 = 0060$;

M_5 — alamprogramm 000.360 funktsiooni $\ln x$ väärtuste arvutamiseks; selles programmis $l_5 = 0066$ ja $e_5 = 0060$;

M_6 — alamprogramm 000.364 funktsiooni e^x väärtuste arvutamiseks; selles programmis $l_6 = 0100$ ja $e_6 = 0072$;

M_7 — alamprogramm 000.314 kahendsüsteemi arvude teisendamiseks kümnendsüsteemi; selles programmis $l_7 = 0106$ ja $e_7 = 0100$.

Eeldame, et massiivid $M_3 \div M_7$ on juba biblioteeki viidud. Sealjuures on teada, et pöördumisel massiivide M_5, M_6, M_7 poole tuleb juhtimine anda massiivi esimesele pesale käsuga

22 b 4,

massiivi M_4 poole tuleb aga $\sin x$ arvutamisel pöörduda käsuga

22 $b + 0004$ 4.

Massiivide $M_4 \div M_7$ puhul kantakse argument ja tulemus üle summaatori kaudu.

Täiendavalt tuleb seega programmeerida vaid põhiprogrammi massiivid M_1 ja M_2. Massiiv M_1 peab sealjuures tsükliliselt valima parameetrite k ja m kombinatsioonid, leidma massiivide M_2 ning M_3 abil neile vastavad integraali väärtused, teisendama need suurused siis M_7 abil kümnendsüsteemi ja välja trükkima.

Kõigepealt tuleb massiivi M_1 tarbeks fikseerida aadresside diapasoonid sisemiste aadresside ja M_2, M_3 ning M_7 juurde kuuluvate väliste aadresside jaoks. Üldiselt on see jaotus üsna suurtes piirides vabalt valitav. Teeme valiku näiteks järgmiselt:

$2000 \leqslant a < a_k$ — massiivi M_1 sisemised aadressid; a_k määrame pärast M_1 programmeerimist;

$5000 \leqslant b < 6000$ — massiivi M_7 juurde kuuluvad välised aadressid;

$6000 \leqslant b < 7000$ — massiivi M_3 juurde kuuluvad välised aadressid;

$7000 \leqslant b < 7100$ — massiivi M_2 juurde kuuluvad välised aadressid.

Siinjuures tuleb märkida, et kuna massiivis M_1 esinevad välised aadressid ainult juhtimise andmisel massiivide M_2, M_3, M_7 algus-pesadele, siis piisaks tegelikult ka hoopis lühematest diapasoonidest, näiteks $5000 \leqslant b < 5002$, $5002 \leqslant b < 5004$, $5004 \leqslant b < < 5006$.

Nüüd võime juba massiivi M_1 programmeerida lõplikul kujul, näiteks järgmiselt:

```
2000   02 2050 4
   1   16 2045 4
   2   25 0002
   3   27 2031
   4   02 2052 4
   5   16 2056 4
   6   25 0004
   7   34 0004

2010   22 5000 4 ┐
   1   32 0004  │
   2   42 2054 4 ├ Parameetrite k ja m teisendamine ning
   3   22 5000 4 │ trükkimine
   4   32 0004  ┘

   5   22 6000 4 ┐
   6   22 7000 4 │
   7   42 2040 4 │
2020   42 2042 4 ├ Pöördumine M3 poole
   1   42 2046 4 │
   2   42 2044 4 ┘

   3   22 5000 4 ┐ Integraali väärtuste teisendamine ja
   4   32 0004  ┘ trükkimine

   5   42 2056 4 ┐
   6   41 2040 4 ├ k suurendamine 0,1 võrra
   7   56 2056 4 ┘

2030   24 2007
   1   00 0000
   2   42 2054 4 ┐
   3   41 2050 4 ├ m suurendamine 1 võrra
   4   56 2054 4 ┘

   5   24 2003
   6   34 0004
   7   37 0000     Lahendamise lõpp

2040   63 1463   ┐ 0,1
   1   63 1541 4 ┘

   2   40 6111 4 ┐ 0,001
   3   72 3044 4 ┘
```

4	62 2077		$\frac{\pi}{2}$		
5	—52 4200	4			
6	00 0000				
7	00 0000				
2050	40 0000		1		
1	00 0000	4			
2	40 0000		$\frac{1}{2}$		
3	00 0000				
4	00 7100		$\langle m \rangle$	Tööpesad	Informatsiooni-blokk
5	—00 0000	4			
6	00 7000		$\langle k \rangle$		
7	776	4			
2060	00 6000				
1	15 0200	4			
2	00 5000				
3	00 0314	4			
4	00 2036				
5	00 2000				
6	00 2060				
7	00 2000				
2070	00 0000				

Ühtlasi saame nüüd massiivi M_1 kohta anda järgmised andmed:

$l_1 = 0070,$
$e_1 = 0060,$
$a_k = 2060.$

Massiivis M_2 tuleb kasutada massiive M_4, M_5, M_6 ja massiivi M_1 pesi $\langle k \rangle$, $\langle m \rangle$ ja $\langle 0{,}5 \rangle$. Analoogiliselt massiivile M_1 valime ka siin aadresside diapasoonid näiteks järgmiselt:

$3000 \leqslant a < a_k$ — massiivi M_2 sisemiste aadresside diapasoon;
$7000 \leqslant b < 7010$ — massiivi M_5 juurde kuuluvad välised aadressid;
$7010 \leqslant b < 7020$ — massiivi M_6 juurde kuuluvad aadressid;
$7020 \leqslant b < 7100$ — massiivi M_4 juurde kuuluvad aadressid;
$7100 \leqslant b < 7200$ — massiivi M_1 juurde kuuluvad aadressid.

Programmeerimegi nüüd massiivi M_2, mis peab tema poole pöördumisel arvutama summaatoris etteantud muutuja x väärtusele vastava integraali väärtuse, kusjuures tulemus peab juhtimise tagasiandmisel jääma summaatorisse. Selle programmi võib koostada näiteks järgmiselt:

3000	00 0000		
1	56 3016 4	$k \ln x$ arvutamine	
2	22 7000 4		
3	46 7156 4		
4	56 3020 4		
5	02 3016 4	$2m \ln \sin x$ arvutamine	
6	22 7024 4		
7	22 7000 4		
3010	46 7154 4		
1	47 7152 4		
2	41 3020 4	$x^k \sin^{2m} x$ leidmine	
3	22 7010 4		
4	22 3000		
5	0		
6	00 7200	Tööpesad	Informatsiooni-blokk
7	—00 0000 4		
3020	00 7100		
1	777 4		
2	00 7020		
3	00 0330 4		
4	00 7010		
5	00 0364 4		
6	00 7000		
7	00 0360 4		
3030	00 3022		
1	00 3000		
2	00 3034		
3	00 3000		
4	00 0000		

Siit saime ühtlasi ka massiivi M_2 osade pikkused:

$l_2 = 0034$,
$e_2 = 0022$,
$a_k = 3022$.

Seega on kõik ülesande programmi formeerimiseks vajalikud massiivid olemas. Biblioteegimassiivide $M_3 \div M_7$ kohta leiab «ССП-42» andmed juba biblioteegi püsivandmete tabelist. Massiivide M_1 ja M_2 kohta anname koostavale programmile ette andmed (muutuvandmete tabeli) ühe perfokaardi abil. Juhul kui massiivid perforeerime kaartidele topeltpesade kaupa kontrollsummadega, tuleb vajalikud andmed kaardile kanda kujul:

— 7770000601
0000000000705
— 7760000221
0000000000345.

Kui soovime massiivid paigutada alates pesast 1000 järjekorras M_1, M_2, ... , M_7, siis piisabki nende andmete etteandmisest. Ülesande lahendamiseks asetame saadud kaardid perfokaartide sisendile järgmiselt:

1) «ССП-42» sisseviimise kaart,
2) kaart muutuvandmete tabeliga,
3) tabeli lõpu tunnuseks tühi kaart,
4) kaardid massiiviga M_1,
5) kaardid massiiviga M_2.

6. Esineb ülesandeid, mille programmeerimisel koostava programmi kasutamine on raskendatud. Et koostav programm formeerib ülesande lahendamise programmi sisemälu pesadesse, siis võib juba põhiprogrammi koostamise käigus selguda, et koostatav programm ei mahu sisemällu. Kuidas sel juhul toimida, see sõltub ülesandest ja tema lahendamisel kasutatavatest alamprogrammidest (biblioteegimassiividest).

Programmi võib näiteks jagada võimalikult iseseisvateks osadeks ja formeerida iga osa koostava programmiga nagu omaette programmi. Formeeritud osad tuleb sel juhul viia välisesse mällu või lasta välja perforeerida ja kasutada neid sobivas järjekorras. Väliseid aadresse saab nüüd kasutada ainult nende massiivide juures, mis korraga sisemälus töötavad, põhiprogrammi massiividele peab aga programmeerija lisama käsud, mis lahendamise käigus programmi osi vahetavad.

Neid raskusi on sageli võimalik ületada interpreteeriva programmi «СИП-21» kasutamisega, mille korral sisemälus hoitava programmi pikkus üldiselt väheneb. Ka nüüd jaotatakse programm osadeks, kusjuures sisemällu jäetava osa (põhiprogrammi massiivid, interpreteeriv programm) võib ikka formeerida koostava programmiga. Ülejäänud osadeks peavad olema kinnised biblioteegimassiivid ja nende vahetamine toimub automaatselt, interpreteeriva programmi abil.

Interpreteeriva programmi «СИП-21», mis paikneb alati massiivide biblioteegis kui massiiv number 777 777, põhiline ülesanne ongi massiivi sissetoomine massiivide biblioteegist ja selle modifitseerimine. Olenevalt pöördumisest võib see toimuda järgmisel kahel viisil.

1) Interpreteerivale programmile antakse ette massiivi number ja kui ei pöörduta massiivi esimese pesa poole, siis ka sisendpesa aadress (kaugusena massiivi algusest). Nende andmete järgi toob interpreteeriv programm biblioteegist massiivi, modifitseerib ta tööpesadesse (või teeb kindlaks, et modifitseeritud massiiv on tööpesades juba olemas) ning pöördub massiivi poole. Massiivilt läheb juhtimine tagasi sellele massiivile, kust toimus pöördumine.

2) Interpreteerivale programmile antakse ette massiivi number ja tema alguspesa aadress. Nende andmete põhjal tuuakse

etteantud numbriga massiiv sisemällu (näidatud pesast alates) ja modifitseeritakse ta töökorda samadesse pesadesse. Juhtimine läheb seejärel, ilma et sissetoodud massiivi poole oleks pöördutud, tagasi sellele massiivile, kust toimus pöördumine.

Interpreteeriva programmiga sissetoodavad massiivid peavad olema kinnised ja asuma biblioteegis magnettrumlil.

Mälu jaotamisel tuleb interpreteeriva programmi käskude jaoks eraldada $400_{(8)}$ pesa, näiteks mälu lõpust. Vahetult nende pesade ette tuleb veel jätta tööpesadeks vähemalt

$$2n + \max_i \max(e_i, l_i) + 2$$

pesa, kus n on interpreteeriva programmi abil sissetoodavate massiivide arv, e_i — massiivi M_i töötava osa pikkus ja l_i — sissetoodava osa pikkus. Oma tööpesade massiivi lõppu moodustab interpreteeriv programm tabeli, kuhu ta salvestab andmed antud hetkel tööpesades asuvate massiivide kohta.

Kui interpreteeriva programmi poole pöördutakse eespool kirjeldatud esimest pöördumisviisi kasutades, siis toimub kõigepealt kontroll, kas massiivi number on juba andmete tabelis. Numbri puudumisel tuuakse massiiv tööpesadesse. Sissetoodud massiivid paigutatakse üksteise järele, esimesest tööpesast alates. Kui uue massiivi sissetoomisel tuleb puudus vabadest tööpesadest, siis kõigi tööpesade sisu (koos tabeli andmetega) kustutatakse ja massiiv paigutatakse tööpesade algusesse. Suure arvu massiivide kasutamisel võivad niisugused vahetamised ülesande lahendamise

Tabel 32

Peatuse aadress	Peatumise põhjus	Lahendamise jätkamine
7431 (lukuga)	Massiiv ei mahu tööpesadesse	Suurendada tööpesade arvu
7552	Massiiv tuuakse biblioteegist sisse valesti või ei ole ta biblioteegis säilinud	Käivitamisega tuuakse massiiv veel kord sisse. Kontrollida biblioteeki programmiga *Valvur —44*
7571 (lukuga)	Massiivi ei ole	Kontrollida massiivi informatsiooniblokki. Lahtine massiiv modifitseerida koostava programmiga
7655	Massiiv on biblioteegis magnetlindi osas	Programmiga *Valvur —44* viia massiiv üle trumlile
7671	Etteantud numbriga massiivi ei ole biblioteegis	Programmiga *Valvur —44* viia massiiv biblioteeki

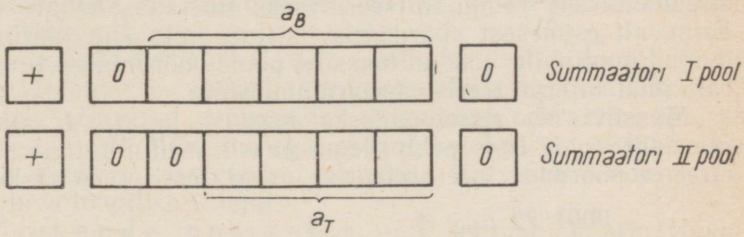
aega tunduvalt pikendada. Sellest on võimalik hoiduda kas suurema arvu tööpesade eraldamisega või mõnede sagelikasutatavamate massiivide toomisega väljaspoole tööpesi (kasutades teist pöördumisviisi).

Programmi töötamise ajal võivad esineda tabelis 32 näidatud peatused (tabelis on antud peatuste aadressid programmi asetusel mälu lõpus, pesades 7400—7777).

7. Pärast interpreteeriva programmi toomist sisemällu (alates pesast a_{int}) peab põhiprogramm ainult üks kord pöörduma tema poole käsuga

22 a_{int} + 0355 4,

andes seejuures summaatoris ette joonisel 48 näidatud andmed: kus a_B on massiivide biblioteegis püsivandmete tabeli algusaadress ja a_T — interpreteeriva programmi tööpesade algusaadress. Näidatud pöördumise järel valmistab programm ise ennast ette biblioteegi ja tööpesade kasutamiseks. Teistkordne pöördumine pessa a_{int} + 0355 pole lubatud.



JOON. 48

Massiivi poole pöördumiseks tuleb pöörduda esimesel viisil interpreteeriva programmi poole. Pöördumine otse mingi massiivi M_j poole toimub antud süsteemis käskudega

```
c + 0000  22  b_j + S_j  4
c + 0001                  } Massiivile M_i etteantav informat-
. . . .                   } sioon
```

kus b_j on massiivi M_j alguspesa aadress ja S_j — massiivi M_j sisendpesa suhteline aadress. Pöördumine sama massiivi poole interpreteeriva programmi kaudu tuleb programmeerida käskudega

```
c + 0000   22   a_int   4
c + 0001 +      N_j     4
c + 0002   00   S_j     0
c + 0003                  } Informatsioon massiivile M_j
. . . .                   }
```

Massiivi esimese pesa poole pöördumisel võib aga kasutada lühemat vormi

$$\left.\begin{array}{llll} c+0000 & 22 & a_{int} & 4 \\ c+0001 & + & N_j & 0 \\ c+0002 & & & \\ \ldots & & & \end{array}\right\} \text{ Informatsioon massiivile } M_j$$

Lühema pöördumise tunnuseks on null teise pesa 19. kahendkohal. Kogu esimese pöördumisviisi tunnuseks on märk «+» massiivi numbri N_j ees.

Pöördumisel massiivi M_j poole taastab interpreteeriv programm summaatori ja tsükliloendaja sisud, mis ta salvestas oma töö algul, pärast mingist massiivist tema poole pöördumist.

Massiivis M_j, mille poole pöördutakse, kasutatakse argumentide üleviimiseks ja juhtimise tagasiandmiseks pöördumiskäsu 22...4 poolt moodustatud suunamiskäsku. Seepärast saadab interpreteeriv programm sisendpessa käsu 22 $c+0003\,0$ või 22 $c+$ $+0002\,0$ (viimase siis, kui kasutati lühemat pöördumist).

Teist pöördumisviisi kasutatakse kinnise biblioteegimassiivi sissetoomiseks ja modifitseerimiseks mistahes kohale sisemälus. Erinevalt esimesest pöördumisviisist ei anta siin juhtimist sissetoodud massiivile: toodud massiivi poole pöördutakse tavalises korras, ilma interpreteeriva programmi abita.

Massiivi M_j sissetoomiseks mingist pesast d_j alates tuleb see aadress d_j (mis peab olema paarisarvuline) tuua summaatorisse ja pöörduda interpreteeriva programmi poole käskudega

$$\begin{array}{llll} c+0001 & 22 & a_{int} & 4 \\ c+0001 & - & N_j & 0 \end{array}$$

Pärast massiivi M_j modifitseerimist läheb juhtimine nüüd tagasi pessa $c+0002$ ja summaatorisse jääb suurus d_j+e_j, kus e_j on massiivi M_j töötava osa pikkus. Seda suurust saab ära kasutada mitme massiivi sissetoomisel üksteise järele.

Teise pöördumisviisi juures interpreteeriv programm juhtimise tagasiandmisel tsükliloendaja pöördumiseelset seisu ei taasta.

§ 20. PROGRAMMI LOOGILINE SKEEM JA SELLE TEISENDAMINE

1. Nagu me juba teame eelmistest paragrahvidest, on vähegi ulatuslikumate ülesannete programmeerimisel kõigepealt otstarbekohane koostada programmi blokk-skeem. Jättes kõrvale üksikasjad, võib öelda, et programmi blokk-skeem on programmi üksikute osade (nn. blokkide) realiseerimise järjekorra graafiline esitus. Sellise graafilise blokk-skeemi põhiliseks plussiks on tema ülevaatlikkus, mis kergendab programmi koostamist ja orientee-

rumist juba koostatud programmis. Kuid mitmete teoreetiliste küsimuste käsitlemiseks on blokk-skeemid liiga kohmakad. Kõik see teeb vajalikuks blokk-skeemi mõiste teatava avardamise ja toob meid programmi loogilise skeemi mõiste juurde. Programmi loogilise skeemi puhul on meil samuti tegemist programmi üksikute osade järjekorra fikseerimisega, kuid see ei tarvitse tingimata olla graafiline: järjekord võib olla fikseeritud näiteks spetsiaalsete sümbolite abil.

Programmi loogilise skeemi korral ei nimetata programmi üksikuid (skeemi enda seisukohalt mittejagatavaid) osi tavaliselt enam blokkideks, vaid operaatoriteks. Operaatorite kohta esitatavad nõuded on põhiliselt siiski samad mis blokkide puhul. Ka siin osutub näiteks oluliseks loogiliste ja mitteloogiliste operaatorite eristamine. Konkreetsete programmide loogilistes skeemides on aga ülevaatlikkuse huvides otstarbekohane eristada veel ka mitteloogiliste operaatorite mõningaid olulisemaid liike. Tavaliselt tõstetakse esile järgmisi operaatorite eriliike.

1) Aritmeetiline operaator, mis teostab arvutusi aritmeetiliste (aga ka loogiliste) valemite alusel ühes vastavate arvutustulemuste salvestamisega. Sellist operaatorit tähistame edaspidi tähega *A*.

2) Loogiline operaator, mis kontrollib loogiliste tingimuste täidetust ja suunab arvutuste käiku vastavalt kontrolli tulemustele (tingitud suunamine). Niisuguse operaatori tähiseks kasutame tähte *P*. Märgime aga, et loogilise operaatori tähise asemel võib skeemis soovi korral kirjutada ka vastava kontrollimisele kuuluva loogilise valemi.

3) Ülekande operaator, mis kannab informatsioonimassiive üle ühest salvestusseadmest teise või salvestusseadme ühest osast teise ossa; niisuguse operaatori tähiseks olgu *Ü*.

4) Parameetrite muutmise operaator, mis muudab arvutuste käiku juhtivaid parameetreid, näiteks teatavaid käske, pesade aadresse vms. Sellist operaatorit tähistame tähega *F*, millele vajaduse korral lisame (sulgudes operaatori taga) vastavate parameetrite tähised, näit. $F_3(i)$, $F_k(i, j)$ jne.

5) Trükkimise (üldisemalt, informatsiooni väljastamise) operaator. Selle tähisena kasutame tähte *T*.

6) Arvuti töö peatamise operaator, tähiseks — *S*.

Muidugi on võimalik esile tõsta ja eri tähtede abil tähistada veel palju teisi operaatorite liike (nagu näiteks käskude formeerimise operaator, parameetrite algseisu taastamise operaator, kontrollsummade arvutamise operaator jne.), kuid ülaltoodud liikidest piisab loogilise skeemi kõikide oluliste osade ülevaatlikuks esitamiseks. Mõningate teoreetiliste küsimuste käsitlemisel on küll vahel otstarbekohane võtta kasutusele veelgi üldisemad tähised, tähistades mitteloogilisi operaatoreid tähega *D* ja operaatoreid üldse, olenemata nende iseloomust, tähega *Q*.

2. Programmi loogilise skeemi (täpsemini, operaatorite abil esitatava skeemi ehk o p e r a a t o r s k e e m i) üleskirjutamisel juhindutakse järgmistest kokkulepetest.

1) Skeemis nummerdatakse kõik operaatorid ühtses numeratsioonis, näiteks

$$U_1\ A_2\ A_3\ T_4\ A_5.$$

2) Kui operaator sõltub parameetrist (resp. parameetritest), siis märgitakse see parameeter (resp. need parameetrid) kas sulgudesse vastava operaatori taha või ülemise indeksina, näiteks

$A_2(i)$ või A_2^i,
$A_k(l, m)$ või $A_k^{l,m}$.

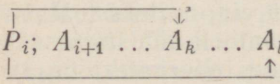
3) Kui operaatorid on skeemis vahetult üksteise kõrval ja nende vahel pole mingisuguseid arvutuste käiku suunavaid märke ega märki «;», siis täidetakse operaatorid tavalises järjekorras (vasakult paremale). Nii näiteks täidetakse skeemis $U_1 A_2 A_3$ pärast operaatorit U_1 operaator A_2 ja pärast operaatorit A_2 operaator A_3.

4) Kui mingi operaator ei anna kunagi juhtimist temast vahetult paremal asuvale operaatorile, siis eraldatakse need operaatorid semikooloniga, näiteks

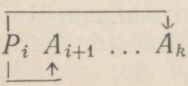
$$A_k;\ A_{k+1}.$$

Tähendame, et semikooloni kasutamine operaatorite vahel pole obligatoorne, sest olukord, kus mingi operaator ei anna juhtimist edasi vahetult järgnevale operaatorile, selgub juba suunamist tähistavate käskude põhjal.

5) Arvutuste käigu suunamist (juhtimist) märgitakse nooltega järgmisel viisil:



Selline kirjutis tähendab, et kui t i n g i m u s, mida kontrollib loogiline operaator P_i (ehk lihtsalt tingimus P_i) on täidetud, siis antakse juhtimine operaatorile A_k, kui aga tingimus P_i pole täidetud, siis antakse juhtimine operaatorile A_l. Seega antakse tingimuse P_i kehtivuse korral juhtimine edasi ülemise noole kaudu, mittekehtivuse korral aga alumise noole kaudu. Erijuhul, kui üks operaatoritest, millele loogiline operaator juhtimise edasi annab, asub vahetult selle loogilise operaatori järel, jäetakse vastav suunamist märkiv nool kirjutamata. Nii kirjutatakse



asemel lihtsalt

$$P_i\ A_{i+1}\ \dots\ A_k \qquad (1)$$

ja

$$P_i\ A_{i+1}\ \dots\ A_k$$

asemel

$$P_i\ A_{i+1}\ \dots\ A_k. \qquad (2)$$

Millisel juhul (kas tingimuse P_i kehtivuse või mittekehtivuse korral) antakse juhtimine skeemides (1) ja (2) operaatorilt P_i vahetult operaatorile A_{i+1}, see järeldub teisest, skeemis olemasolevast noolest.

Kui juhtimine antakse üle mitteloogiliselt operaatorilt, siis pole see seotud tingimuste kontrollimisega (nn. tingimatu suunamise juhtum). Sel korral võime kasutada soovi korral kas ülemist või alumist noolt. Seega võime kirjutada näiteks kas

$$A_i;\ A_{i+1}\ \dots\ A_k \quad \text{või} \quad A_i;\ A_{i+1}\ \dots\ A_k.$$

6) Kui skeemi töö ei alga mitte esimese operaatoriga, siis tähistatakse esimene tööd alustav operaator noolekesega (↑ või ↓) ja skeemi ette märgitakse «;». Näiteks

$$;\ A_1\ P_2\ A_3\ P_4\ A_5\ S_6 \quad \text{ehk} \quad ;\ A_1\ P_2\ A_3\ P_4\ A_5\ S_6,$$

tähendab, et esitatud skeemis alustab tööd operaator A_3.

Mõnedes loogilisi skeeme käsitlevates töödes kasutatakse murdjooneliste noolte asemel trükitehnilistel põhjustel ka teistsuguseid suunamistähiseid, näiteks noolekesi või nn. nurki. Sel korral kirjutatakse

$$P_1\ A_2\ \dots\ A_k \quad \text{asemel} \quad P_1\ A_2\ \dots\ A_k \quad \text{või} \quad P_1\ A_2\ \dots\ A_k$$

$$P_1\ A_2\ \dots\ A_k \quad \text{asemel} \quad P_1\ A_2\ \dots\ A_k \quad \text{või} \quad P_1\ A_2\ \dots\ A_k.$$

Noolekesed või nurgad peavad seejuures olema nummerdatud. Nummerdada võib neid muidugi mitut moodi. Näiteks nagu ülaltoodud näidetes, kus väljuvale noolele märgitakse juurde selle operaatori number, kuhu juhtimine edasi antakse ja suubuvale noolele selle operaatori number, kust juhtimine anti. Teine nummerdamise võimalus on selline, et ühe ja sama numbriga (alustades näiteks vasakult) tähistatakse teineteisele vastavate väljuvate-suubuvate noolekeste või nurkade paar.

Noolekeste ja nurkade puhul tehakse tavaliselt veel mitmeid lihtsustavaid kokkuleppeid. Näiteks võib üksteisele järgnevad suu-

namismärgid $\overset{k}{\ulcorner}\underset{l}{\llcorner}$ või $\underset{l}{\llcorner}\overset{k}{\ulcorner}$ asendada tähisega $\overset{k}{\underset{l}{\sqsubset}}$. Samuti võib suubuvad suunamismärgid (neid on kaht tüüpi: $\urcorner$ ja $\lrcorner$) asendada üht tüüpi märkidega, näiteks märgiga $\lrcorner$. Mitu järjestikust suubuvat suunamistähist võib asendada ühega, mille alla on muidugi märgitud vastavad numbrid. Nii näiteks võib järjestikused suubuvad suunamistähised $\underset{5}{\lrcorner}\overset{7}{\urcorner}\underset{1}{\lrcorner}$ mingi operaatori juures asendada tähisega $\underset{5}{\lrcorner}\underset{7}{\lrcorner}\underset{1}{\lrcorner}$, selle omakorda tähisega $\underset{5,7,1}{\underline{\qquad\lrcorner}}$ ja kuna numbrite järjekord suunamismärgi all pole oluline, siis lõpuks tähisega $\underset{1,5,7}{\underline{\qquad\lrcorner}}$. Viimane lihtsustusvõte on loomulikult kasutatav ka suubuvate noolekeste puhul.

Tähendame veel, et programmi loogilise skeemi esitamisel tuleb anda ka täiendav informatsioon skeemi iga üksiku operaatori töö iseloomustamiseks.

3. Toome nüüd kaks konkreetset näidet programmi loogiliste skeemide koostamise kohta.

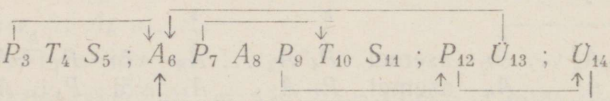
N ä i d e 1. Koostame programmi loogilise skeemi n-astme võrrandi

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0 = 0 \tag{3}$$

lahendi x^* ligikaudseks leidmiseks etteantud lõigul $[\alpha, \beta]$. Lahendusmeetodina kasutame lõigu järjestikuse poolitamise meetodit, mis eeldab, et $f(\alpha)$ ja $f(\beta)$ peavad olema erinevate märkidega. Leitav ligikaudne lahend x ei tohi erineda tõelisest lahendist x^* rohkem kui etteantud arvu ε $(\varepsilon > 0)$ võrra.

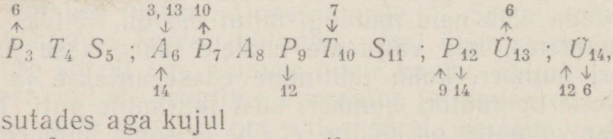
Esitatavas loogilises skeemis jätame polünoomi väärtuse arvutamise täpsustamata, kuna sellest on küllalt üksikasjalikult kõneldud juba eespool. Seda märkust arvestades tuleb programmi loogiline skeem järgmine

$$\ddot{U}_1\ A_2\ P_3\ T_4\ S_5\ ;\ A_6\ P_7\ A_8\ P_9\ T_{10}\ S_{11}\ ;\ P_{12}\ \ddot{U}_{13}\ ;\ \ddot{U}_{14},$$



Kasutades suunamistähistena noolekesi võib sama skeemi kirjutada kujul

$$\ddot{U}_1\ A_2\ \overset{6}{\overset{\uparrow}{P_3}}\ T_4\ S_5\ ;\ \overset{3,13}{\underset{14}{\overset{\downarrow}{A_6}}}\ \overset{10}{\overset{\uparrow}{P_7}}\ A_8\ \underset{12}{\underset{\downarrow}{P_9}}\ \overset{7}{\overset{\downarrow}{T_{10}}}\ S_{11}\ ;\ \underset{9}{\underset{\uparrow}{P_{12}}}\ \underset{14}{\underset{\downarrow}{\overset{6}{\overset{\uparrow}{\ddot{U}_{13}}}}}\ ;\ \underset{12}{\underset{\uparrow}{\ddot{U}_{14}}}\ \underset{6}{\underset{\downarrow}{\ }},$$



nurkasid kasutades aga kujul

$$\ddot{U}_1\ A_2\ P_3 \overset{6}{\ulcorner}\ T_4\ S_5\ ;\ \underset{3,13,14}{\lrcorner}\ A_6\ P_7 \overset{10}{\ulcorner}\ A_8\ P_9 \underset{12}{\llcorner}\ \underset{7}{\lrcorner}\ T_{10}\ S_{11}\ ;$$

$$;\ \underset{9}{\lrcorner}\ P_{12} \underset{14}{\llcorner}\ \ddot{U}_{13} \overset{6}{\ulcorner}\ ;\ \underset{12}{\lrcorner}\ \ddot{U}_{14} \underset{6}{\llcorner}.$$

Toodud skeemis on erinevatel operaatoritel järgmised tähendused:

U_1 — salvestatakse polünoomi kordajad $a_n, a_{n-1}, \ldots, a_0$, suurus ε ja lõigu otspunktid α ning β; viimaste puhul on otstarbekohane kasutada omistamise sümbolit «:=» ning kirjutada $a := \alpha$ ja $b := \beta$ (s. t. suurusele a omistatakse väärtus α ja suurusele b väärtus β);

A_2 — arvutatakse polünoomi väärtused $f(a)$ ja $f(b)$ ning salvestatakse need kindlates pesades: $d := f(a)$ ja $e := f(b)$;

P_3 — kontrollitakse, kas $f(a)$ ja $f(b)$ on erineva märgiga;

T_4 — trükitakse välja $f(a)$ ja $f(b)$ ühes märkusega, et ülesanne pole antud lähteandmete korral lahendatav;

S_5 — arvuti töö peatatakse;

A_6 — arvutatakse ja omistatakse $c := \dfrac{a+b}{2}$;

P_7 — kontrollitakse, kas $\dfrac{|a-b|}{2} = \dfrac{b-a}{2} < \varepsilon$;

A_8 — arvutatakse ja omistatakse $e := f(c)$; tähendame, et suuruse e endine väärtus pole enam vajalik;

P_9 — kontrollitakse, kas $f(c) = 0$;

T_{10} — trükitakse välja suuruse c väärtus, mis on võrrandi (3) lahendiks;

S_{11} — arvuti töö peatatakse;

P_{12} — kortrollitakse, kas $f(a)$ ja $f(c)$ on erineva märgiga;

U_{13} — tehakse ülekanne $b := c$;

U_{14} — tehakse ülekanded $a := c$ ja $d := e$.

Näide 2. Koostame tingrefleksi modelleeriva programmi loogilise skeemi.

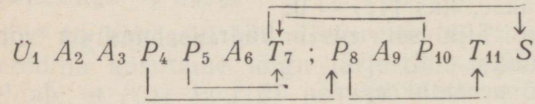
Tähistame α abil tingimatu ja β abil tingitud ärritaja, R abil aga reaktsiooni. Suurused α, β ja R võivad sealjuures omandada kaht väärtust: 1 (kui ärritaja toimib või reaktsioon esineb) ja 0 (vastupidisel juhul). Kui $\alpha = 1$, siis olenemata β väärtusest on $R = 1$. Kui $\alpha = 0$, siis $\beta = 0$ korral on ka $R = 0$. Kui $\alpha = 0$ ja $\beta = 1$, siis oleneb reaktsiooni esinemine tingrefleksi väljakujundamise tugevusest. Viimast iseloomustatakse suuruse c abil, mis muutub *eksperimendi* käigus järgmiselt. Kui $\alpha = \beta = 1$, siis liidame suurusele c teatava arvu δ ($\delta > 0$) — tingrefleks tugevneb. Kui aga $\alpha = 0$ ja $\beta = 1$, siis lahutame suurusest c arvu δ — tingrefleks nõrgeneb. Et tingrefleksi väljakujundamine on mingil määral ka stohhastiline protsess, siis liidetakse suurusele c iga üksikkatse korral veel juhuslik suurus η, mis on kas ühtlaselt või normaalselt jaotatud teatavas vahemikus $(-\gamma, \gamma)$. Problemaatilisel juhul (s. t. kui $\alpha = 0$ ja $\beta = 1$) toimime järgmiselt: kui $c > c_0$ (konstantne suurus c_0 on tingrefleksi väljakujundatuse lävivääг-

Ärritajad	Tingrefleksi väljakujundamist iseloomustava suuruse muutumine	Reaktsioon
$\alpha = 1,\ \beta = 1$	$c := c + \eta + \delta$	$R = 1$
$\alpha = 1,\ \beta = 0$	$c := c + \eta$	$R = 1$
$\alpha = 0,\ \beta = 1$	$c := c + \eta - \delta$	kui $c > c_0$, siis $R = 1$ kui $c \leqslant c_0$, siis $R = 0$
$\alpha = 0,\ \beta = 0$	$c := c + \eta$	$R = 0$

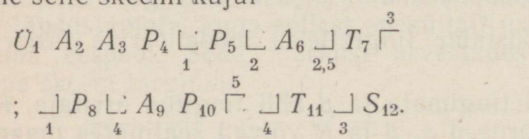
tus), siis $R = 1$, vastupidisel juhul on $R = 0$. Kirjeldatud olukorrast annab kokkuvõtliku ülevaate tabel 33.

Märkus. Kirjutist «$c := c + \eta$» tuleb mõista selliselt: suurusele c omistatakse c endine väärtus pluss η, s. t. suurusele c liidetakse lihtsalt suurus η. Analoogiliselt tuleb seda liiki kirjutisi mõista ka teistel juhtudel.

Esitame nüüd tingrefleksi modelleerimise programmi loogilise skeemi:



Kasutades suunamistähistena nurkasid ja nummerdades (alates vasakult) teineteisele vastavad nurgad ühe ja sama numbriga, saame selle skeemi kujul



Skeemis esinevate operaatorite töö kohta lisame järgmised andmed:

$Ü_1$ — salvestatakse üksikkatse andmed α ja β;
A_2 — juhusliku suuruse η genereerimine (arvutamine);
A_3 — arvutatakse ja omistatakse $c := c + \eta$;
P_4 — kontrollitakse, kas $\alpha = 1$;
P_5 — kontrollitakse, kas $\beta = 1$;
A_6 — arvutatakse ja omistatakse $c := c + \delta$;
T_7 — trükitakse $R = 1$;
P_8 — kontrollitakse, kas $\beta = 1$;
A_9 — arvutatakse ja omistatakse $c := c - \delta$;
P_{10} — kontrollitakse, kas $c > c_0$;
T_{11} — trükitakse $R = 0$;
S_{12} — arvuti töö peatatakse.

4. Loogiliste skeemide teisendamise juurde asudes tuleb märkida kõigepealt seda, et me ei vaatle mitte suvalisi skeeme. Vaatlusele tulevad skeemid peavad nimelt rahuldama järgmisi tingimusi.

1) Skeemid on muutumatud. Skeemi töö käigus ei kustutata operaatoreid ega moodustata uusi operaatoreid. Samaks jääb ka operaatorite loogiline iseloom (vt. p. 2). Samuti ei kustutata skeemi töö käigus suunamistähiseid, ei muudeta nende asendit või iseloomu ega moodustata uusi suunamisi.

2) Skeemi kõik operaatorid on jagatud kahte liiki — loogilisteks ja mitteloogilisteks. Loogilised operaatorid kontrollivad tingimusi, mis on moodustatud nn. loogilistest parameetritest loogiliste tehete abil. Loogilised operaatorid ei saa muuta loogiliste parameetrite väärtusi, seda saavad teha ainult mitteloogilised operaatorid. Eeldame samuti, et meil on mitteloogiliste operaatorite puhul üldiselt teada, missuguseid loogilisi parameetreid nad võivad muuta.

Skeemide teisendamine nõuab skeemide samaväärsuse mõiste defineerimist. Intuitiivselt on see mõiste selge: samaväärsed skeemid peavad üht ja sama lähteinformatsiooni töötama ümber põhimõtteliselt samal viisil. Et me skeemide abstraktse käsitluse korral ei saa aga vahetult kontrollida informatsiooni ümbertöötamist, tuleb toodud lähteideed täpsustada. Teatavasti töötavad informatsiooni ümber ainult mitteloogilised operaatorid (loogilised operaatorid teostavad ainult suunamisi). Seega võiksime võrrelda, millises järjekorras töötavad antud skeemis mitteloogilised operaatorid. Seejuures tuleb arvestada ka seda, et mitteloogilised operaatorid võivad muuta loogiliste parameetrite väärtusi, mis loogiliste operaatorite kaudu võib mõjustada suunamisi. Kokkuvõttes võime öelda, et kaht loogilist skeemi nimetatakse *samaväärseks*, kui nendes skeemides töötavad loogiliste parameetrite ühtede ja samade väärtuste korral ikka ühed ja samad mitteloogilised operaatorid. Kuigi antud sõnastus on küllaltki üldine, võimaldab see paljudel juhtudel siiski kindlaks teha kahe skeemi samaväärsust.

Skeemide samaväärsust tähistame märgi «≡» abil. Seega, kui skeemid $\mathfrak{A}$ ja $\mathfrak{B}$ on samaväärsed, kirjutame

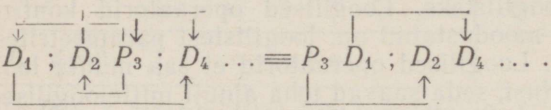
$$\mathfrak{A} \equiv \mathfrak{B}.$$

Esitame nüüd mõningaid näiteid skeemide samaväärsuse kohta. Näited on põhimõttelist laadi, sest vaadeldavatest skeemidest on esitatud ainult selle oluline fragment. Esitatud näiteid võib seega vaadelda kui skeemide teisendamise reegleid, mida rakendades saame lähteskeemi teisendada temaga samaväärseks skeemiks.

1. Loogilises skeemis võib igat operaatorit ümber paigutada tingimusel, et ükski juhtimise üleandmine operaatorilt operaato-

rile (tingitud või tingimatu suunamine) selle ümberpaigutuse juures ei muutuks. Kui eeldada, et kõik suunamised (ka naaber-operaatorite vahelised suunamised) on skeemis kirja pandud murdjooneliste noolte abil, siis võiks toodud reegli sõnastada näiteks nii: operaatoreid on võimalik skeemis ümber paigutada, kuid noolte algused ja lõpud enne ja pärast ümberpaigutust peavad ikka olema seotud samade operaatoritega.

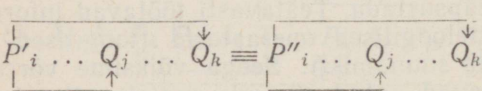
On selge, et selle reegli rakendamine muudab ainult n. ö. skeemi *kirjapilti*, ei muuda aga skeemi operaatorite funktsioneerimist. Näiteks on samaväärsed järgmised kaks skeemilõiku:



2. Teatavasti kontrollitakse loogilises operaatoris mingi loogilise parameetri või, üldiselt, mingi loogilise valemi kehtivust. Kui loogilises operaatoris asendada loogiline valem $P'(a_1, a_2, \ldots, a_n)$ samaväärse valemiga $P''(a_1, a_2, \ldots, a_n)$, siis saame esialgse skeemiga samaväärse skeemi. Seega, kui

$$P'_i(a_1, a_2, \ldots, a_n) \equiv P''_i(a_1, a_2, \ldots, a_n),$$

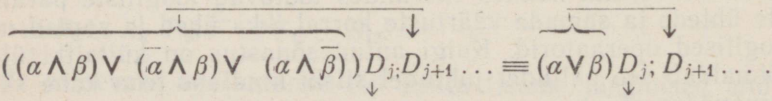
siis



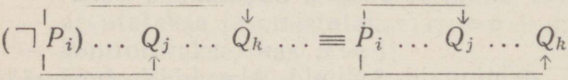
Toome ka selle kohta näite. Pole raske kontrollida, et kehtib järgmine loogiline samaväärsus

$$a \vee \beta \equiv (a \wedge \beta) \vee (\bar{a} \wedge \beta) \vee (a \wedge \bar{\beta})$$

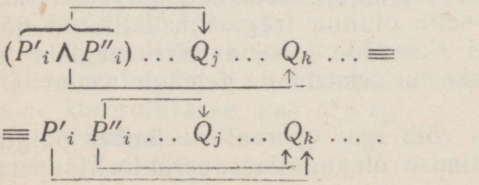
Seega saame



3. Erituse märgi võib loogilise tingimuse eest ära jätta, kui selle tingimuse juurest lähtuvad suunamistähised ümber pöörata:



4. Kahe tingimuse konjunktsiooni saab teisendada nii



(4)

P'_i	P''_i	$P'_i \wedge P''_i$	Juhtimine antakse edasi operaatorile	
			Vasakul pool	Paremal pool
0	0	0	Q_k	Q_k
0	1	0	Q_k	Q_k
1	0	0	Q_k	Q_k (P''_i kaudu)
1	1	1	Q_j	Q_j

Selle, samuti nagu mitme teise skeemi samaväärsuse kehtivust võib kontrollida näiteks tabeli 34 koostamise teel.
Seega töötavad kahes ülaltoodud skeemis kõigil neljal juhul ikka vastavalt samad operaatorid, mis tähendabki nende skeemide samaväärsust.

Arvestades konjunktsiooni kommutatiivsust, saame samaväärsuse (4) abil üles kirjutada ka järgmise samaväärsuse

$$P'_i\ P''_i \dots Q_j \dots Q_k \dots \equiv P''_i\ P'_i \dots Q_j \dots Q_k \dots ,$$

mida on muidugi võimalik leida ja kontrollida otseselt, ilma seost (4) kasutamata.

5. Kahe tingimuse disjunktsiooni saab teisendada nii

$$(P'_i \wedge P''_i) \dots Q_j \dots Q_k \dots \equiv P'_i\ P''_i \dots Q_j \dots Q_k \dots ,$$

millest disjunktsiooni kommutatiivsust kasutades võime saada uue samaväärsuse

$$P'_i\ P''_i \dots Q_j \dots Q_k \dots \equiv P''_i\ P'_i \dots Q_j \dots Q_k \dots .$$

6. Kui loogiline operaator P_i saab juhtimise ainult siis, kui $P_j = 1$, siis võib tingimuse P_i asendada tingimusega $P_i \wedge P_j$ või $P_i \vee \bar{P}_j$. Seega

$$P_j \dots ; P_i \dots Q_k \dots \equiv P_j \dots ; (P_i \wedge P_j) \dots Q_k \dots$$

$$P_j \dots ; P_i \dots Q_k \dots \equiv P_j \dots ; (P_i \vee \bar{P}_j) \dots Q_k \dots$$

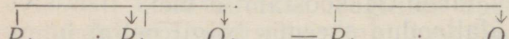
Esitatud samaväärsuste kehtivust on soovi korral võimalik kontrollida näiteks tabeliga 34 analoogiliste tabelite abil.

Märgime, et reeglit 6 on võimalik sõnastada veelgi üldisemal kujul. Nimelt võib antud eeldusel ühendada valemi P_i mistahes osavalemiga valemi P_j konjunktsiooni tehte abil (nn. konjunktiivselt) või valemi $\overline{P}_j$ disjunktiivselt.

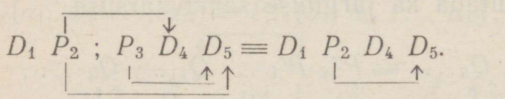
7. Samade loogiliste operaatorite järjestikusel rakendamisel on võimalik skeemi teisendada nii:



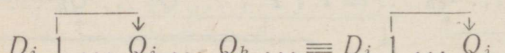
Kui teine operaator P_i ei saa juhtimist kuskilt mujalt kui esimeselt operaatorilt P_i, võib teise operaatori P_i hoopis ära jätta. Seega.



8. Kui loogilises skeemis mingile operaatorile Q_j üldse juhtimist ei anta, siis võib selle operaatori Q_j ühes temalt lähtuvate nooltega loogilisest skeemist välja jätta. Näiteks saame:

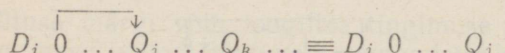


9. Kui loogilises skeemis esineb samaselt kehtiv ($P'_i \equiv 1$) või samaselt mittekehtiv ($P''_i \equiv 0$) tingimus, siis võib kustutada nende tingimuste juurest lähtuvad nooled, mis peaksid juhtimise edasi andma juhtudel $P'_i = 0$ ja $P''_i = 1$. Vastavad loogilised tingimused P'_i ja P''_i võib skeemis asendada konstantidega 1 ja 0. Näiteks saame:



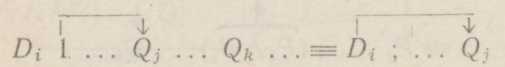
(5)

ja

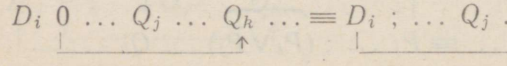


(6)

Skeemi, mis sisaldab loogilisi operaatoreid — konstante 1 ja 0, on võimalik teisendada, kui arvestada, millistelt operaatoritelt nad on juhtimise saanud. Skeemide (5) ja (6) korral on konstandid 1 ja 0 saanud juhtimise eelmiselt operaatorilt D_i. Seega saame

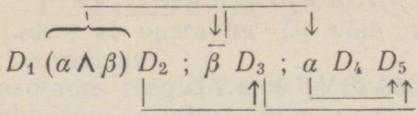


ja



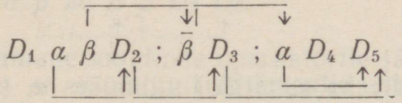
5. Toome nüüd mõningaid ulatuslikumaid näiteid loogiliste skeemide teisendamise kohta.

N ä i d e 3. Teisendame skeemi

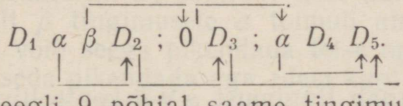


lihtsamale kujule.

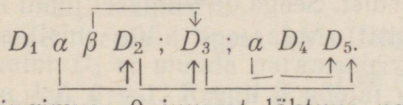
Teisendades konjunktsiooni reegli 4 alusel, saame



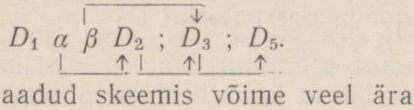
Arvestades, et tingimuse $\bar{\beta}$ kontroll toimub ainult juhul $\beta = 1$, saame reegli 6 põhjal tingimuse $\beta \wedge \bar{\beta} \equiv 0$, millega saame



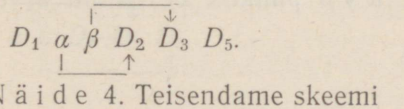
Reegli 9 põhjal saame tingimusest 0 lähtuva ülemise noole kustutada, samuti võime tingimusest β lähtuvat noolt pikendada operaatorini D_3. Seega saame



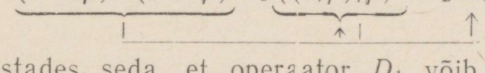
Tingimuse 0 juurest lähtuva noole kustutamisega tekkis olukord, et tingimuse α kehtivust kontrolliv operaator (operaatori D_4 ees) üldse juhtimist ei saa. Reegli 8 põhjal võib selle operaatori, ühes temast lähtuva noolega, skeemist kustutada. Et operaator D_4 saab juhtimise aga ainult sellelt operaatorilt, võib kustutada ka selle operaatori. Seega saame



Saadud skeemis võime veel ära jätta tarbetud semikoolonid ja vastavalt kaks alumist suunamistähist, millega olemegi saanud lõppkuju



N ä i d e 4. Teisendame skeemi



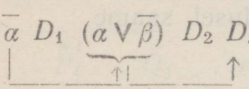
arvestades seda, et operaator D_1 võib muuta ainult loogilise parameetri β väärtust.

Kõigepealt teisendame skeemis esinevad loogilised avaldised lihtsamale kujule. Saame

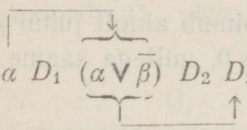
$$(\alpha \supset \overline{\beta}) \wedge (\alpha \supset \beta) \equiv (\overline{\alpha} \vee \overline{\beta}) \wedge (\overline{\alpha} \vee \beta) \equiv \overline{\alpha} \vee (\overline{\alpha} \wedge \beta) \vee$$
$$\vee (\overline{\alpha} \wedge \overline{\beta}) \equiv \overline{\alpha}$$

$$(\alpha/\beta)/\beta \equiv (\overline{\alpha} \vee \overline{\beta})/\beta \equiv (\alpha \wedge \beta) \vee \overline{\beta} \equiv \alpha \vee \overline{\beta}.$$

Asendades need avaldised, saame skeemile kuju



Kasutades reeglit 3 ja kaotades eitusmärgi tingimuses α, saame

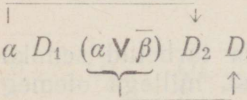


Nüüd analüüsime, millal saab tingimust $\alpha \vee \overline{\beta}$ kontrolliv operaator juhtimise. See toimub kahel juhul. Esiteks, kui $\alpha = 1$, ja teiseks, kui $\alpha = 0$ ja on töötanud operaator D_1, kus viimane võib muuta ainult parameetri β väärtust. Seega on viimasel juhul ikkagi $\alpha = 0$. Nüüd teisendame avaldist $\alpha \vee \overline{\beta}$ reegli 6 alusel. Saame

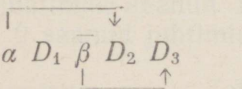
1) kui $\alpha = 1$, siis $\overline{\alpha} \vee \alpha \vee \overline{\beta} \equiv 1$,
2) kui $\alpha = 0$, siis kasutades reegli 6 juures tehtud üldistust ja arvestades, et $\overline{\alpha} = 1$, mille põhjal võime kirjutada

$$(\overline{\alpha} \wedge \alpha) \vee \overline{\beta} \equiv \overline{\beta}.$$

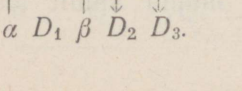
Seega saime mõlemal erijuhul erineva tulemuse. Asi seisneb aga selles, et tingimust 1 võib ära jätta, kasutades reeglit 9. Seega saame ülemist poolt pikendada operaatorini D_2



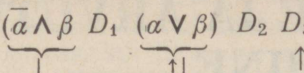
Asendades seejärel tingimuse $\alpha \vee \overline{\beta}$ punktis 2) tehtud arvutuste põhjal, saame skeemile kuju



ehk lõpuks

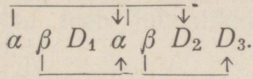


Näide 5. Teisendame skeemi

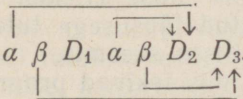


arvestades, et operaator D_1 võib muuta ainult loogilise parameetri α väärtust.

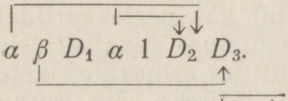
Kasutades reegleid 4, 5 ja 3, võime ära jätta konjunktsiooni, disjunktsiooni ja eituse ning saame skeemi kujul



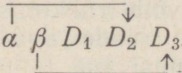
Samade loogiliste operaatorite järjestikusel rakendamisel (reegel 7) on võimalik juhtimise edasiandmist tähistavat noolt nihutada. Antud juhul saame tingimuselt α tingimusele α suunduvat noolt nihutada operaatorini D_2. Teine operaatorilt (tingimuselt) α operaatorini D_2 suunduv nool muidugi säilib. Suunamine tingimuselt β tingimusele α toimub ainult juhul $\alpha = 0$ ja vastavat noolt võib seega pikendada teise operaatorini β. Reegel 7 põhjal võib seda pikendada aga edasi kuni operaatorini D_3. Seega saame



Vaatleme nüüd operaatori (tingimuse) β teist esinemist. Et operaator D_1 ei muuda parameetri β väärtust, siis saab see juhtimise, kui $\beta = 1$. Reegli 6 põhjal võib selle tingimuse asendada tingimusega $\bar{\beta} \vee \beta \equiv 1$. Rakendades veel reeglit 9, saame kirjutada



Vaatleme skeemi osa $\alpha\, 1\, D_2$. Kui $\alpha = 1$, saab juhtimise operaator D_2; kui $\alpha = 0$, saab juhtimise samuti operaator D_2. Seega pole tingimuse α kontrollimisel antud juhul mingit vajadust ja selle võib skeemist välja jätta. Samuti jätame välja tingimuse 1. Seega saame



IV PEATÜKK

KONTROLLIMINE JA ARVUTUSMEETODI VALIMINE

§ 21. PROGRAMMI KONTROLLIMINE

1. Ülesande ettevalmistamine elektronarvutil lahendamiseks on aeganõudev ja töömahukas protsess, mille erinevatel etappidel võib tekkida väga mitmesuguseid vigu. Näiteks võib juhtuda, et ülesandele lahendamismeetodi valikul pole arvestatud kõiki tema iseärasusi, mille tõttu valitud meetod kas üldse ei sobi antud ülesande lahendamiseks või ei anna nõutud täpsusega tulemusi. Esineda võivad vead ka programmi loogilises skeemis.

Kõige tavalisemad on aga need vead, mis tekivad programmi koostamisel: valed aadressid, vahelejäänud käsud jne. Programmeerimisprotsess sisaldab endas sellisel hulgal vigade tekkimise võimalusi, et programme, mis on algusest peale koostatud ilma ühegi veata, praktiliselt peaaegu ei esine.

Pärast programmi valmimist kantakse arvutusteks vajalik materjal (programm, ülesande algandmed) vastavate seadmete (perforaatorid, telegraafiaparaadid) abil kas perfokaartidele või telegraafilindile. Perforeerija tähelepanematuse, blankettidele kantud arvude halva loetavuse ja perforaatorites esinevate rikete tõttu võib ka selle töö käigus tekkida vigu. Näiteks võib osutuda, et teatud arvude asemel on perfokaartidele kantud teised arvud, mõned sümbolid on jäänud telegraafilindile hoopiski perforeerimata jne.

Sellega pole aga kaugeltki veel kõik vigade tekkimise võimalused loetletud. Nimelt võib ülesande lahendamise ajal vigu esineda ka elektronarvuti seadmete töös. Näiteks võib trükkimisseade moonutada väljastatavaid arve, aritmeetiline seade aga anda operatsioonide realiseerimisel ebaõigeid tulemusi jne.

Elektronarvuti seadmete kõrvalekaldumised ettenähtud töörežiimist (arvuti vead) jaotatakse süstemaatilisteks ja juhuslikeks. Viimased erinevad esimestest eeskätt oma tekke põhjuste poolest.

S ü s t e m a a t i l i s e d v e a d tekivad elektronarvuti töös mitmesuguste tehniliste rikete (näiteks mingi elektronlambi või

pooljuhtseadme mittefunktsioneerimise) tõttu. Sellest on tingitud ka nende püsiv iseloom. Täpsemalt: arvuti teeb neid vigu seni, kuni vastavad rikked kõrvaldatakse. Öeldakse, et arvuti on tehniliselt korras, kui ta ei tee süstemaatilisi vigu.

Juhuslikud vead on ebapüsiva iseloomuga. Arvutuste kordamisel arvuti neid uuesti ei tee. Seetõttu polegi tavaliselt võimalik kindlaks teha ühe või teise juhusliku vea tekkimise täpset põhjust. Nende esinemine arvuti töös on üldiselt tingitud mingi juhusliku välise teguri mõjust (näiteks toitevoolu muutusest jms.). Seetõttu teeb juhuslikke vigu ka tehniliselt täiesti korrasolev arvuti.

Süstemaatiliste vigade kõrvaldamine on elektronarvutit teenindava tehnilise personali ülesandeks. Nende avastamiseks lahendatakse elektronarvutil spetsiaalseid kontrollülesandeid ehk teste. Reeglina kontrollitakse arvuti korrasolekut testidega iga uue ülesande lahendamise eel. Selle kontrolli käigus avastatud rikked kõrvaldatakse ja ülesande lahendamist alustatakse tehniliselt korrasoleva arvutiga. Paraku ei välista selline kontroll veel mitmesuguste rikete ja juhuslike vigade tekkimise võimalust ülesande lahendamise ajal.

Eelnevat kokku võttes saame öelda, et iga ülesande lahendamisel tuleb arvestada vigade tekkimise võimalust nii programmeerimise, perforeerimise kui ka ülesande lahendamise käigus ja seetõttu tarvitusele võtta abinõusid nende vigade õigeaegseks avastamiseks ning kõrvaldamiseks. Võtteid, mida kasutatakse ülesannete lahendamise ja ettevalmistamise käigus tekkivate vigade avastamiseks, nimetatakse kontrolli meetoditeks.

Nagu nägime, võib ülesande lahendamise protsessis esineda üpris erineva iseloomu ja päritoluga vigu. Mitte vähem erinevad üksteisest ka nende avastamise meetodid. Seetõttu vaatlemegi eraldi programmi kontrollimise, perforeerimisvigade avastamise ja arvuti töö kontrollimise meetodeid.

2. Tööprotsessi, mille käigus programmeerija loogiliselt põhjendatud plaani alusel kontrollib valminud programmi ja kõrvaldab sellest vead, nimetatakse silumiseks. Programmi kontrollimiseks kasutatakse mitmeid erinevaid viise. Kõige tavalisem neist seisneb selles, et programmeerija, pärast mõne päeva möödumist programmi valmimisest, vaatab selle veelkord läbi. Lugedes tähelepanelikult programmi, kontrollib ta käskude aadresse, kasutatud konstante, üleminekuid programmi ühest blokist teise jne. Seejuures on soovitav kas koostada programmist lähtudes uus blokk-skeem või arvuti tööd imiteerides välja kirjutada kõik vaadeldava programmiga realiseeritavad matemaatilised valemid. Sellise hoolika kontrolliga on võimalik avastada kõik programmis esinevad juhuslikud vead (valed aadressid, vahelejäänud käsud jne.) ning need juba enne programmi perforeerimist kõrvaldada.

Kui hoolikalt programmi *käsitsi* ka ei kontrollita, võib sellesse ikka veel jääda vigu. Eeskätt ei võimalda selline kontrollimine avastada neid vigu, mis programmeerija on teinud oma teadmiste mittetäielikkuse tõttu (vead lahendamismeetodi valikul, arvuti mõnede käskude teatud iseärasuste mitteteadmine jne.). Teiselt poolt on aga täiesti tõenäone, et mitmed vead, mida sellise kontrolliga põhimõtteliselt küll saab avastada, on ikkagi jäänud kahe silma vahele. Asjaolu, et programmeerija tunneb enda koostatud programmi peensusteni, nõrgendab sedavõrd ta tähelepanuvõimet, et ka kõige jämedam viga võib mõnikord jääda märkamatuks. Seetõttu ei saa *käsitsi* kontrollitud programmi veel lugeda täielikult korrasolevaks. Lõplikuks silumiseks tuleb teda kontrollida elektronarvuti abil.

Programmi silumine on üpris kulukas (näiteks USA-s läheb iga programmis esineva vea avastamine suurte arvutite IBM-704 või UNIVAC SCIENTIFIC abil keskmiselt maksma 20 kuni 50 dollarit). Et hoida neid kulutusi vähegi mõistlikes piirides ja mitte raisata asjatult elektronarvutite tööaega, tuleb programme siluda hästi läbimõeldud ja loogiliselt põhjendatud plaanide alusel.

Programmide silumisega seotud küsimusi vaatleme lähemalt käesoleva paragrahvi lõpul. Siin märgime ainult veel seda, et perforeerimisvigade õigeaegse parandamisega on võimalik oluliselt vähendada nii programmi silumise aega kui ka sellega seotud kulutusi. Nimelt, silumise käigus kõrvaldatakse programmist ka kõik perforeerimisvead (muidugi juhul, kui seda enne pole tehtud!). Paraku nõuab see aga hoopis suuremat töökulu kui perforeerimise vahetu kontrollimine. Arvestades seda ja asjaolu, et ülesande algandmetes olevaid vigu ei avastata silumise käigus, tuleb perforeeritud materjali eraldi kontrollida.

3. Perforeerimist võib kontrollida mitmeti. Suurem osa selleks kasutatavatest viisidest tugineb perforeeritud ja blankettidele kantud materjali võrdlemisel. Seda arvestades kirjeldame järgnevas üksikasjalisemalt neist võtetest ainult kahte.

Üks kõige tavalisemaid perforeerimisvigade avastamise viise seisneb selles, et vastavat abiprogrammi kasutades salvestatakse ülesande lahendamiseks vajalik perforeeritud materjal elektronarvuti mäluseadmeisse ja trükitakse sealt välja.

Seejuures on soovitav väljastada ka kontrollitavate informatsioonimassiivide kontrollsummad. Nimelt, elektronarvutite «Minsk-2» ja «Minsk-22» kasutamisel saadakse salvestamise käigus need summad täiesti automaatselt (salvestamisega kaasneb summeerimine), «Ural-4» puhul tuleb aga nende arvutamine ette näha salvestamist ja trükkimist teostavas abiprogrammis. Massiivide kontrollsummasid on ülesande lahendamise käigus tarvis elektronarvuti sisendseadmete töö kontrollimiseks (eriti, kui salvestamine toimub telegraafilindilt).

Väljatrükitud programmi (algandmeid) võrreldakse blankettidele kantuga. Kui võrdlemisel mingeid erinevusi ilmsiks ei tule, siis loetakse materjal õigesti perforeerituks. Erinevuste esinemise korral parandatakse avastatud vead ning korratakse kontrolli.

Perforeerimist saab kontrollida ka sel teel, et ülesande lahendamiseks vajaliku materjali kannavad erinevad perforeerijad erinevate perfoaatorite abil kahele telegraafilindile või kaartide komplektile. Saadud linte (kaardipakke) võrreldakse kas spetsiaalses seadmes või visuaalselt. Kui võrreldavate pakkide kõik vastavad kaardid (võrreldavad lindid) ühtivad, siis tuleb lugeda, et materjal on vigadeta perforeeritud. Nimelt, kahe ühtelangeva perforeerimisvea olemasolu kirjeldatud viisil valmistatud lintidel (kaardipakkides) on äärmiselt ebatõenäone.

4. Koostatud programmi õigsuse üle on võimalik lõplikult otsustada ainult tema kasutamisel saadud arvutustulemuste järgi. Selleks peavad aga teada olema lahendatava ülesande vähemalt ühe variandi vastused. Viimaste ühtimine tulemustega, mis saadakse sama variandi lahendamisega elektronarvutil, võetaksegi silumisel programmi õigsuse kriteeriumiks. Kahjuks ei taga see aga alati veel kõigi vigade kõrvaldamist programmist. Nimelt, selle kriteeriumi rakendatavus sõltub otseselt programmi kontrollimiseks kasutatava variandi valikust. Olgu näiteks tarvis kontrollida funktsiooni

$$f(x) = (x^2 - 1)\,\frac{x^2 \cos x + e^{-ax}}{(x+a)^2} + xe^{-ax}$$

väärtuste arvutamiseks koostatud programmi. Kui selleks kasutada juhtu, kus $x = 1$ (või on sellele lähedane), siis esimene liidetav praktiliselt summat ei mõjuta. Selle tõttu funktsiooni väärtus võib arvutustäpsuse piirides olla õige ka siis, kui murru

$$\frac{x^2 \cos x + e^{-ax}}{(x+a)^2}$$

väärtus on leitud valesti ja vastavas programmiosas esineb vigu. Seega, antud juhul pole nimetatud kriteerium rakendatav.

Analoogiline on olukord siis, kui näiteks funktsiooni

$$z = \begin{cases} \frac{1}{2} \ln \frac{1+x}{1-x}, & \text{kui } |x| < 1 \\ \ln(x + \sqrt{x^2 \pm 1}), & \text{kui } |x| \geqslant 1 \end{cases}$$

väärtuste arvutamiseks koostatud programmi kontrollimiseks kasutada ainult varianti, milles $x > 1$. Sel korral programmiosa, millega arvutatakse avaldise $\frac{1}{2} \ln \frac{1+x}{1-x}$ väärtusi, jääb hoopiski

kontrollimata. Antud juhul tuleb seega kasutada vähemalt kahte lahendatava ülesande varianti, millest ühe korral $|x| < 1$, teise puhul aga $|x| \geqslant 1$.

Eelnevat võib kokku võtta järgmiselt: arvestades lahendatava ülesande iseärasusi tuleb programmi silumiseks valida tema variant (või variandid), mille lahendamisega on võimalik piisava põhjalikkusega kontrollida kõiki programmi osi. Ainult sel korral näitab elektronarvutiga leitud ja käsitsilahendamisel saadud tulemuste ühtimine, et programmis pole vigu.

Reeglina on vigu igas vastvalminud programmis, mille tõttu võib juba ette öelda, et tema kasutamisega kontrollvariantide lahendamisel saadakse ebaõiged tulemused (kui arvuti neid üldse väljastabki). Et saada informatsiooni vigade asukoha ja iseloomu kohta, tuleb iga programmi siluda.

5. Ettevalmistused silumiseks algavad tavaliselt sellega, et programmi kontrollimiseks valitakse välja ülesande sobiv variant (või variandid), mis lahendatakse läbi käsitsi. Kui on võimalik, tehakse seda sama algoritmi järgi, mille alusel on koostatud silutav programm. Esiteks lihtsustab see silumist, teiseks aga näitab, kas vaadeldavat tüüpi ülesannet üldse on valitud meetodiga võimalik nõutud täpsusega lahendada. Viimast asjaolu arvestades tuleks silumiseks vajalikud kontrollvariandid läbi lahendada juba enne programmi koostamist. Seejuures peab tingimata säilitama ka kõik arvutuste vahetulemused, sest neid on hiljem tarvis programmi kontrollimisel.

Enamasti koosneb programm teatud arvust suhteliselt iseseisvatest osadest (blokkidest), mis ülesande lahendamisel töötavad mingis kindlas järjekorras. Tavaliselt alustatakse programmi silumist aritmeetiliste blokkide kontrollimisega ja alles pärast seda kontrollitakse tema loogilist struktuuri.

Programmi aritmeetilist osa kontrollitakse üksikute blokkide kaupa. Selleks lastakse arvutil teha blokis ettenähtud arvutused, kusjuures trükitakse välja nii lähteandmed kui ka arvutuste lõpp- ja vahetulemused. Et seda oleks lihtsam teha, siis tuleks niisuguse informatsiooni saamise vajadust silmas pidada juba programmi koostamisel. Nimelt, kui arvuti mälu maht seda võimaldab, siis tuleksid ühe bloki ulatuses kasutatavad tööpesad valida võimalikult kompaktse massiivina ja nii, et nendesse salvestatavad vahetulemused säiliksid kogu vaadeldava bloki täitmise ajal. Sel korral saab kontrolliks tarviliku materjali välja trükkida korraga, kohe pärast blokiga määratud arvutusprotsessi lõppemist. Väljastatud materjali ja käsitsilahendamisel saadud tulemuste kõrvutamisega tehaksegi kindlaks vigade olemasolu ja asukoht kontrollitavatele blokkidele vastavates programmiosades.

Tavaliselt kasutatakse aritmeetiliste blokkide silumisel lähteandmetena kontrollülesande algandmeid või vahetulemusi. Mõne-

del juhtudel nõuab aga kontrollvariandi lahendamine nii suurt ajakulu, et seda käsitsi pole võimalik teha. Sel korral kasutatakse aritmeetiliste blokkide kontrollimiseks *kunstlikke lähteandmeid.* Nimelt, nendeks võetakse arvud, millega blokis ettenähtud arvutuste sooritamine on mugavam.

6. Aritmeetiliste blokkide kontrollimiseks kasutatakse standardseid programme, mis silumise käigus selleks vajaliku informatsiooni väljastavadki. Arvestades, et erinevate arvutite korral (samuti arvutuskeskustes) kasutatakse selleks erinevaid silumisprogramme, me nendel üksikasjalisemalt ei peatu. Tähendame, et iga niisugune programm on varustatud kasutamisjuhendiga, millega tutvumata seda silumisel nagunii pole võimalik rakendada. Küll aga selgitame mõne sõnaga nende kasutamise üldpõhimõtteid.

Silumisprogrammi kasutamiseks tuleb vastavalt juhendile koostada ja telegraafilindile või perfokaartidele kanda silumisandmed. Aritmeetiliste blokkide kontrollimiseks ette nähtud standardne programm võib kõigi silutavate blokkide kohta nõuda näiteks järgmist informatsiooni:

1) missuguses pesas oleva käsuga tuleb alustada kontrollitavas blokis ettenähtud arvutusi ja millises pesas paikneva käsu täitmisega need arvutused lõpetatakse;

2) missugustes operatiivmälu pesades paiknevad väljastamisele kuuluvad tulemused ja kuidas neid tuleb trükkida (8-nd- või 10-ndsüsteemis, fikseeritud või liikuva koma arvudena) jne.

Selliseid andmeid ja silutavas programmis kontrollitavatele blokkidele vastavaid programmiosi kasutades väljastabki standardne programm silumise käigus meid huvitavad tulemused.

Kuigi üksikult võttes aritmeetilised blokid pärast niisugust kontrolli ja avastatud vigade parandamist töötavad õigesti, ei tarvitse nende kooskasutamine ülesande lahendamisel veel anda õigeid lõpptulemusi. Aritmeetiliste blokkide silumisega ju ei kontrollitud informatsioonivahetusi blokkide vahel ja nende realiseerimise järjekorda. Seda tehaksegi programmi loogilise struktuuri kontrollimisel.

7. Programmi loogilise struktuuri kontrollimiseks lastakse arvutil lahendada kontrollülesandeid. Seejuures väljastatakse perioodiliselt (s. t. teatud kindlate käskude sooritamise järel) informatsiooni nii blokkide töötamise järjekorra kui ka operatiivmälu teatud pesadesse lahendamise käigus salvestatavaid vahetulemusi. Nagu aritmeetiliste blokkide silumisel, nii väljastavad meile vajaliku informatsiooni vastavate silumisandmete alusel ka nüüd standardsed programmid.

Väljastatud informatsiooni järgi tehaksegi kindlaks vigade olemasolu ja asukoht programmis. Nimelt kontrollülesande korral on meil ju programmi blokkide töötamise õige järjekord teada.

Kui arvuti täitis blokid selles järjekorras ja sai õiged lõpptulemused, ongi programm silutud. Vastasel korral tuleb aga väljatrükitud informatsiooni järgi leida viga. Avastatud vead parandatakse, mille järel koostatakse uued andmed standardsete silumisprogrammide jaoks ja korratakse programmi loogilise struktuuri kontrollimist.

§ 22. ARVUTI TÖÖ KONTROLLIMINE

1. Enamikul käesoleval ajal toodetavatest elektronarvutitest puuduvad spetsiaalsed seadmed nende töö automaatseks kontrollimiseks. Selliste arvutite hulka kuuluvad ka «Ural-4», «Minsk-2» ja «Minsk-22». Et saada informatsiooni vigade esinemisest ja kõrvaldada juhuslike vigade mõju arvutustulemustele, peavad ülesannete lahendamisel kasutatavad programmid ette nägema arvutite töö süstemaatilise kontrollimise. Viimast nõuet tuleb programmeerijatel silmas pidada juba ülesannetele lahendamisalgoritmide väljatöötamisel.

Elektronarvuti töös kontrollitakse järgmisi etappe:

1) programmi ja algandmete salvestamist mäluseadmetesse;
2) operatiiv- ja välismälude vahel toimuvaid informatsioonivahetusi;
3) arvutuste õigsust ja
4) tulemuste väljastamist.

Edasises käsitluses vaatleme arvuti kontrollimiseks kasutatavaid võtteid vastavalt loetletud etappidele.

2. Ülesande lahendamiseks vajaliku materjali (programmi, algandmete) salvestamist elektronarvuti operatiivmälusse kontrollitakse k o n t r o l l s u m m a d e abil. Järgnevas kirjeldame üksikasjalisemalt kolme niisugust kontrollimisviisi. Esimene neist on kasutatav perfokaartidelt, ülejäänud aga telegraafilindilt salvestamise kontrollimisel.

Numbrilise informatsiooni perforeerimisel kantakse igale kaardile (selle esimesele üheteistkümnele reale) üksteist salvestamisele kuuluvat arvu. Kaheteistkümnendale reale lööb perforaator automaatselt eelnevatesse ridadesse perforeeritud arvude kontrollsumma. Seda arvestades kontrollitakse numbrilise informatsiooni salvestamist perfokaartidelt iga kaardi jaoks eraldi. Nimelt arvutatakse salvestamise käigus kaardi esimesest üheteistkümnest reast loetud arvude kontrollsumma ja võrreldakse seda kaardi kaheteistkümnendast reast võetud summaga. Kontrollsummade ühtimise korral loetakse informatsioon vigadeta salvestatuks ja jätkatakse ülesande lahendamist.

Kui kontrollsummad ei ühti, siis peatatakse arvuti ja märgitakse üles viimati salvestatud kaardi järjekorranumber. Erinevusi kontrollsummades võivad põhjustada nii vead arvuti töös,

defektidega perfokaardid, kui ka mõned avastamata jäänud perforeerimisvead (näiteks nihkunud perforatsioonireed). Et kindlaks teha kõiki antud pakis olevaid *kahtlasi kaarte*, jätkatakse ka kontrollsummade mitteühtimise korral salvestamist, kusjuures pärast kogu massiivi salvestamist arvuti peatatakse. Seejärel asendatakse uutega vaadeldava massiivi kaardipakis kõik väljamärgitud järjekorranumbritega kaardid ja korratakse salvestamist. Kui nüüd erinevusi kaartide kontrollsummade võrdlemisel enam ei ilmne, siis jätkatakse ülesande lahendamist. Kui aga ka vaadeldava massiivi teistkordne salvestamdine ei toimu vigadeta, tuleb ülesande lahendamine katkestada. Seda võib jätkata alles pärast töös esinevate süstemaatiliste vigade kõrvaldamist.

Kirjeldatud kontrolli võib massiivi elektronarvuti «Ural-4» operatiivmälu pesadesse a_0 kuni a_n salvestamisel teostada näiteks järgmise programmi abil (salvestamine toimub topeltpesade kaupa).

$k+0$	20	0000	0	Nullide salvestamine tööpesadesse.
$k+1$	16	t_1	0	
$k+2$	16	t_2	4	
$k+3$	25	$a_n - a_0$	4	Salvestamistsükli algus.
$k+4$	40	0002	4	Rea lugemine kaardilt ja salvestamine vastavasse topeltpesasse.
$k+5$	—16	a_n	4	
$k+6$	26	t_2	4	Kaardi kontrollsumma arvutamine.
$k+7$	16	t_2	4	
$k+10$	02	t_1	0	Kaardi ridade loendamine.
$k+11$	01	$k+34$	0	
$k+12$	16	t_1	0	
$k+13$	14	$k+35$	0	Kas on salvestatud kaardi üksteist esimest rida?
$k+14$	21	$k+24$	0	
$k+15$	16	t_1	0	Kaardi ridade loendaja taastamine.
$k+16$	40	0002	4	Kontrollsumma lugemine kaardilt ja võrdlemine arvutatud summaga.
$k+17$	14	t_2	4	
$k+20$	21	$k+23$	4	Kas kontrollsummad on võrdsed?
$k+21$	37	0000	0	Summad ei ühti. Märkida üles kaardi number, lülitada sisse võti nr. 1 ja anda *käivitus*.
$k+22$	02	t_1	0	Nulli kandmine kontrollsumma pesasse.
$k+23$	16	t_2	4	
$k+24$	24	$k+4$	0	Salvestamistsükli lõpp.

$k+25$	21	$k+30$	4	Kas viimaselt kaardilt on salvestatud kõik üksteist esimest rida?
$k+26$	40	0002	4	Viimaselt kaardilt salvestamisele mittekuuluvate ridade lugemine (see on tarvilik kontrollsummade võrdlemiseks).
$k+27$	22	$k+10$	0	
$k+30$	23	0001	0	Kas kõigi kaartide kontrollsummad ühtisid?
$k+31$	22	a	0	Salvestamisel vigu ei esinenud. Juhtimine antakse pesas a olevale käsule, millega algab programmi järgmine osa.
$k+32$	37	0000	0	Salvestamisel oli vigu. Lülitada välja võti nr. 1, asetada sisendile uuesti sama kaardipakk, anda *käivitus* salvestamise kordamiseks.
$k+33$	22	$k+0$	0	
$k+34$	00	0001	0	Konstandid.
$k+35$	00	0013	0	

Telegraafilindilt salvestamise kontrollimiseks võib näiteks iga informatsioonimassiivi arvutisse viia kaks korda, kusjuures mõlemal salvestamisel leitakse operatiivmälusse kantud materjali kontrollsumma. Saadud summasid võrreldakse. Kui nad ühtivad, siis loetakse, et lindil olev materjal on vigadeta salvestatud ja ülesande lahendamist jätkatakse. Seejuures loomulikult eeldatakse, et arvuti ei teinud salvestamise käigus süstemaatilisi vigu (neid pole sellise kontrolliga võimalik avastada).

Sellisel kontrollimisel on soovitav arvuti «Ural-4» korral kasutada telegraafilinte, millele iga massiiv on perforeeritud kahekordselt ja millel lindiosad ühe ja sama massiiviga paiknevad vahetult üksteise järel. Sellega hoiame kokku arvuti tööaega, sest niisuguse lindi korral pole tarvis enne massiivi teistkordset salvestamist arvutit peatada ja linti sisendseadmel (vaadeldava massiivi ulatuses) tagasi kerida.

Elektronarvuti «Minsk-22» operatiivmälusse võib kirjeldatud kontrolliga salvestada arvumassiivi näiteks järgmise programmilõigu abil:

$k+0$	—50	00	0000	a_1	Massiivi salvestamine operatiivmällu alates pesast a_1. Tema kontrollsumma arvutamine.
$k+1$	76	00	0000	a_2	Saadud kontrollsumma salvestamine pesasse a_2.

$k+2$	—07	00	0000	0000	Massiivi teistkordne lugemine lindilt (ilma salvestamiseta). Saadakse massiivi teine kontrollsumma.
$k+3$	—51	00	0000	0000	
$k+4$	07	00	a_2	00000	Kontrollsummade võrdlemine. Nende ühtimisel asutakse täitma pesas a olevat käsku (algab programmi järgmine osa).
$k+5$	—34	00	$k+6$	a	
$k+6$	—00	00	0000	0000	Kui kontrollsummad ei ühti, peatub arvuti pesas $k+5$ oleva käsu täitmise tulemusena.

Analoogiline programm tähelis-numbrilise informatsiooni salvestamiseks telegraafilindilt arvuti «Ural-4» operatiivmälu pesadesse a_0 kuni a_n on aga järgmine:

$k+0$	25	0002	4	Tsükkel massiivi kahekordseks salvestamiseks.
$k+1$	02	t_1	4	Kontrollsumma viimine pesast t_1 pesasse t_2.
$k+2$	16	t_2	4	
$k+3$	20	0000	0	Nulli viimine pesasse t_1.
$k+4$	16	t_1	4	
$k+5$	27	$k+24$	0	
$k+6$	25	$a_n - a_0$	0	Salvestamistsükli algus.
$k+7$	40	0010	0	Pesa esimese sümboli kuuekohalise koodi lugemine ja salvestamine.
$k+10$	—16	a_n	0	
$k+11$	40	0010	0	Pesa teise sümboli koodi lugemine ja salvestamine.
$k+12$	11	0106	0	
$k+13$	—13	a_n	0	
$k+14$	—16	a_n	0	
$k+15$	40	0010	0	Pesa kolmanda sümboli koodi lugemine ja salvestamine.
$k+16$	11	0114	0	
$k+17$	—13	a_n	0	
$k+20$	—16	a_n	0	
$k+21$	26	t_1	4	Kontrollsumma arvutamine.
$k+22$	16	t_1	4	
$k+23$	24	$k+7$	0	Salvestamistsükli lõpp.
$k+24$	00	0000	0	
$k+25$	24	$k+1$	0	Kahekordse salvestamise tsükli lõpp.

$k+26$	14	t_2	4	Kas arvutatud kontrollsummad on võrdsed? Kui on, siis täidetakse pesas *a* olev käsk, millega algab programmi järgmise osa täitmine.
$k+27$	21	*a*	4	
$k+30$	37	0000	0	Salvestamisel oli vigu. Arvuti peatub. Salvestamise kordamiseks anda *käivitus*.
$k+31$	22	$k+0$	0	

Kui soovitakse vältida materjali kahekordse salvestamisega seotud ajakulu, võib kontrollida järgmiselt. Pärast materjali perforeerimist arvutatakse kõigi lindile kantud massiivide kontrollsummad. Kui perforeerimise kontrollimisel kasutatakse elektronarvutit, siis saab seda näiteks teha selle töö käigus. Saadud summad perforeeritakse samale telegraafilindile vastava massiivi juurde. Ülesande lahendamisel arvutatakse lindilt salvestatavale massiivile teine kontrollsumma ning salvestamisel saadud ja lindilt loetud summasid võrreldakse. Nende ühtimise korral on massiiv vigadeta kirjutatud arvuti operatiivmällu ja ülesande lahendamist jätkatakse. Kui võrreldavad kontrollsummad aga osutuvad erinevateks, on massiivi salvestamisel esinenud vigu. Sellistel juhtudel arvuti tavaliselt peatatakse ja korratakse sama massiivi salvestamist.

Lõpuks tuleb märkida veel seda, et viimatikirjeldatud kontrollimisviisi on eriti soovitav kasutada siis, kui mingile telegraafilindile perforeeritud informatsioonimassiive on tarvis korduvalt viia arvuti operatiivmällu.

3. Ülesande lahendamisel saadavad vahetulemused ja muu informatsioon, mis ei mahu operatiivmällu, säilitatakse välistes mäluseadmetes (magnetlintidel ja -trumlitel). Ent arvutustel saab arvuti otseselt kasutada ainult neid informatsioonimassiive, mis on salvestatud tema operatiivmälus. Eeskätt seetõttu tulebki suuremate ülesannete lahendamise käigus salvestada teatud programmiosad, algandmete rühmad või arvutuste vahetulemused ümber arvuti operatiivmälust välistesse mäluseadmetesse ja vastupidi.

Vigade tekkimise võimalust niisuguste informatsioonivahetuste käigus on teataval määral arvestatud juba arvutite «Ural-4» ja «Minsk-22» konstruktsioonis. Näiteks on nende arvutite korral ette nähtud informatsiooni kahekordne kirjutamine ühele ja samale magnetlindile ning selle lugemise automaatne kontrollimine (nn. p a a r s u s e k o n t r o l l). Kahjuks ei garanteeri see aga veel informatsioonimassiivide vigadeta salvestamist magnetlintidelt ferriitmällu ja vastupidi. Seetõttu tuleb iga informatsioonivahetust operatiiv- ja välismälu vahel kontrollida ka ümbersalvestatavate massiivide kontrollsummade abil. Järgnevas peatumegi lühidalt selleks vajalike summade arvutamisel ja tutvustame nimetatud kontrolli põhilisi võtteid.

Ülesande lahendamiseks tarvilikku informatsiooni on võimalik magnetlintidele («Ural-4» korral aga ka magnettrumlitele) kanda ainult ferriitmälu kaudu. Seda kasutades salvestatakse kõik informatsioonimassiivid välismällu koos vastavate kontrollsummadega. Täpsemalt, igale massiivile, mida soovitakse kirjutada magnetlindile (või -trumlile), arvutatakse operatiivmälus kontrollsumma, mille edaspidises tähistame S_0-ga.

Elektronarvuti «Minsk-22» kasutamisel saadakse S_0 massiivi kuuluvate arvude summeerimisel tsüklilise liitmisoperatsiooni (kood —74) abil. Kui aga välismällu kantakse informatsiooni perfolindilt, siis pole programmis tarvis ette näha summa S_0 arvutamist. Nimelt, telegraafilindilt salvestamise operatsioonid leiavad samaaegselt ka ferriitmällu kirjutatava massiivi kontrollsumma.

«Ural-4» kasutamisel tuleb S_0 arvutamiseks programmis ette näha massiivi kuuluvate arvude kontroll-liitmine. Summeerimine peab toimuma pesade kaupa, summa S_0 saadakse aga topeltpesa arvuna. Tähistame tema kahekümnest vasakpoolsest kahendkohast moodustatud arvu S_0^1, ülejäänud numbrikohtadest saadud arvu aga S_0^2-ga. Kui ümbersalvestatav massiiv paikneb operatiivmälu pesades a_1 kuni a_2 ja aadress a_2 on paaritu, siis võib kontrollsumma S_0 leidmiseks kasutada näiteks järgmist programmilõiku:

$k+0$	20	0000	0	Summaatorisse viiakse null.
$k+1$	25	a_2-a_1	0	Summeerimine.
$k+2$	—26	a_2	0	
$k+3$	24	$k+2$	0	
$k+4$	16	a_2+1	4	Saadud summa salvestamine pesadesse a_2+1 ja a_2+2. Pesasse a_2+1 viiakse S_0^1 pesasse a_2+2 aga S_0^2.

Eespool kirjeldatud viisil saadud summa S_0 kantaksegi koos massiiviga operatiivmälust välismällu. Ümbersalvestamise käigus leiab arvuti massiivile automaatselt uue kontrollsumma, mille tähistame S_1-ga. Et S_1 arvutamisel võetakse kõrvuti massiivi arvudega summasse ka S_0, siis elektronarvuti «Minsk-22» kasutamisel peab nende suuruste vahel kehtima seos:

$$S_1 = S_0 \# S_0; \tag{1}$$

arvuti «Ural-4» korral aga

$$S_1 = S_0 \varnothing S_0^1 \varnothing S_0^2, \tag{2}$$

kus sümbol # tähistab tsüklilist liitmisoperatsiooni (kood —74) ja ∅ kontroll-liitmisoperatsiooni (kood 26).

Analoogiliselt summaga S_1 arvutatakse välismälust operatiivmällu ümbersalvestatavale massiivile kontrollsumma ka nõnda-

nimetatud *välismälust lugemise operatsioonide* teostamisel. Tähistame sel teel saadud massiivi kontrollsumma S_2-ga. Kui see informatsioonivahetus on toimunud vigadeta, siis kehtivad summade S_0 ja S_2 vahel valemitega (1) ja (2) vastavalt analoogilised seosed. Nimelt, kui kasutatakse arvutit «Minsk-22», siis

$$S_2 = S_0 \# S_0; \tag{3}$$

«Ural-4» korral aga

$$S_2 = S_0 \varnothing S_0^1 \varnothing S_0^2. \tag{4}$$

Seoseid (3) ja (4) võibki kasutada ülesande lahendamise käigus operatiiv- ja välismälu vahel toimuvate informatsioonivahetuste kontrollimiseks. Nimelt, pärast mingi massiivi ümberkirjutamist operatiivmälust välismällu arvutatakse tema jaoks vastava kontroll-lugemise operatsiooni abil summa S_2. Kui kontrollsummad S_0 ja S_2 rahuldavad kas seost (3) või (4), on informatsioonivahetus toimunud vigadeta ja ülesande lahendamist jätkatakse. Vastasel korral aga tavaliselt peatatakse arvuti ja korratakse sama massiivi ümbersalvestamist. Kui aga teistkordne informatsioonivahetus ei toimu vigadeta, tuleb ülesande lahendamine katkestada.

Seoseid (1) ja (2) pole soovitav eespoolkirjeldatud kontrollis kasutada valemite (3) ja (4) asemel. Nimelt, kui saadud summa S_1 rahuldabki üht neist seostest, võib sellest ainult järeldada, et informatsiooni on ümbersalvestamise käigus õigesti loetud operatiivmälust. Siit aga ei järeldu, et ta on ka vigadeta kirjutatud välismällu. Seega antud juhul seoste (1) ja (2) kasutamine ei taga kõigi võimalike vigade avastamist.

Järgnevas esitame näitena programmilõigu, milles seose (4) abil kontrollitakse elektronarvuti «Ural-4» operatiivmälu pesades a_1 kuni a_2 paikneva massiivi salvestamist magnettrumlile (summa S_0 on kirjutatud operatiivmälu pesadesse a_2+1 ja a_2+2). Trumlile kantakse massiiv alates pesast a_3.

$k+0$	52	a_1	0	} Massiivi ümberkirjutamine trumlile.
$k+1$	00	a_3	0	
$k+2$	00	a_2+2	0	
$k+3$	51	a_1	4	} Kontroll-lugemisega summa S_2 arvutamine.
$k+4$	00	a_3	0	
$k+5$	00	a_2+2	0	
$k+6$	16	t_1	4	} Summa S_2 salvestamine.
$k+7$	02	a_2+1	4	} Suuruse $S_0 \varnothing S_0^1 \varnothing S_0^2$ arvutamine.
$k+10$	26	a_2+1	0	
$k+11$	26	a_2+2	0	

$k+12$	14	t_1	4	Kas $S_2 = S_0 \varnothing S_0^1 \varnothing S_0^2$? Seose kehtimise korral jätkatakse ülesande lahendamist pesas a oleva käsu täitmisega.
$k+13$	21	a	4	
$k+14$	37	0000	0	Salvestamisel esines vigu. Peatus. *Käivitusega* saab korrata sama massiivi salvestamist.
$k+15$	22	$k+0$	0	

Esitatuga täiesti analoogiline on programm, mille abil saab sama massiivi salvestada arvuti «Ural-4» operatiivmälust magnetlindi tsooni C (alates pesast a_3):

$k+0$	45	C	0	Massiivi salvestamine magnetlindile.
$k+1$	57	a_1	0	
$k+2$	00	a_3	0	
$k+3$	00	a_2+2	0	
$k+4$	45	C	0	
$k+5$	02	a_2+1	4	Suuruse $S_0 \varnothing S_0^1 \varnothing S_0^2$ arvutamine ja salvestamine tööpesasse t_1.
$k+6$	26	a_2+1	0	
$k+7$	26	a_2+2	0	
$k+10$	16	t_1	4	
$k+11$	45	C	0	Summa S_2 arvutamine (kontroll-lugemine).
$k+12$	55	a_1	4	
$k+13$	00	a_3	0	
$k+14$	00	a_2+2	0	
$k+15$	21	$k+20$	0	Kas paarsuse kontrolliga avastati vigu?
$k+16$	14	t_1	4	Kas $S_2 = S_0 \otimes S_0^1 \otimes S_0^2$? Seose kehtimisel jätkatakse ülesande lahendamist pesas a oleva käsu täitmisega.
$k+17$	21	a	4	
$k+20$	37	0000	0	Salvestamisel esines vigu. Käivitusega saab sama massiivi salvestamist korrata.
$k+21$	22	$k+0$	0	

Valemid (3) ja (4) pole ainsad seosed, mille abil saab kontrollida informatsiooni ümberkirjutamist arvuti operatiivmälust välismällu. Näiteks võib vaadeldavate arvutitüüpide korral selleks kasutada ka seost

$$S_1 = S_2. \qquad (5)$$

Järgnevas programmilõigus ongi viimast seost kasutatud elektronarvuti «Minsk-22» operatiivmälu pesades a_1 kuni a_2 paikneva massiivi magnetlindile salvestamise kontrollimisel. Arv S_0 paikneb pesas a_2+1. Massiivi kirjutame kapi nr. 0 lindile nr. 0 alates pesast 00 0070. Summa S_1 salvestame pesasse a_2+2.

$k+0$	—46 00	0000	0070	Massiivi kirjutamine magnetlindile (koos summaga S_0). Kui *skeemilise* kontrolliga avastatakse vigu, korratakse kirjutamist.
$k+1$	—43 00	a_2-a_1+2	a_1	
$k+2$	—30 00	$k+0$	0000	
$k+3$	76 00	0000	a_2+2	Summa S_1 salvestamine pesasse a_2+2.
$k+4$	—47 00	0000	0070	Summa S_2 arvutamine. Kui *skeemilise* kontrolliga avastatakse vigu, korratakse kontroll-lugemist.
$k+5$	—44 00	a_2-a_1+2	0000	
$k+6$	—30 00	$k+4$	0000	
$k+7$	07 00	a_2+2	0000	Kas $S_1=S_2$? Kui see seos ei kehti, peatub arvuti. Vastupidisel juhul jätkatakse ülesande lahendamist pesas a oleva käsu täitmisega.
$k+10$	—34 00	$k+11$	a	
$k+11$	—00 00	0000	0000	

Elektronarvuti «Ural-4» kasutamise korral võib massiivid välismälus varustada summa S_0 asemel arvuga S_0^1. Valem (4) asendub siis seosega

$$S_2^1 = S_0^1 \varnothing S_0^1, \tag{6}$$

kus S_2^1 on summa S_2 kahekümnest vasakpoolsest kahendkohast moodustatud arv. Et seose (6) (samuti ka mõnede siin esitamata jäetud seoste) kasutamine kirjeldatud informatsioonivahetuse kontrollimisel on täiesti analoogiline valemite (3), (4) ja (5) rakendamisega, siis me sellel üksikasjalisemalt ei peatu.

Pärast massiivi ümberkirjutamist välismälust operatiivmällu arvutatakse tema jaoks kontrollsumma S_3. See toimub täpselt samuti nagu summa S_0 arvutamine. Seetõttu saadud S_3 peab võrduma välismälust koos massiiviga operatiivmällu salvestatud S_0-ga, summade S_2 ja S_3 vahel peab aga «Minsk-22» korral kehtima seos

$$S_2 = S_3 \# S_0 \tag{7}$$

ja «Ural-4» korral seos

$$S_2 = S_3 \varnothing S_0^1 \varnothing S_0^2. \tag{8}$$

Eeskätt nende seoste abil ongi soovitav kontrollida informatsiooni ümbersalvestamist välismälust operatiivmällu.

Järgnevas esitame näitena programmi, milles seose $S_3 = S_0$ abil kontrollitakse magnettrumlil paikneva massiivi (algab pesast

a_3) salvestamist arvuti «Ural-4» ferriitmälu pesadesse a_1 kuni a_2. Summa S_0 kirjutatakse operatiivmälu pesadesse a_2+1 ja a_2+2.

$k+0$	51	a_1	0	Massiivi lugemine trumlilt (kirjutatakse ferriitmälu pesadesse a_1 kuni a_2).
$k+1$		a_3	0	
$k+2$	00	a_2+2	0	
$k+3$	20	0000	0	Summa S_3 arvutamine.
$k+4$	25	a_2-a_1	0	
$k+5$	−26	a_2	0	
$k+6$	24	$k+5$	0	
$k+7$	14	a_2+1	4	Kas $S_3=S_0$? Kui võrdub, siis jätkatakse ülesande lahendamist pesas a oleva käsu täitmisega.
$k+10$	21	a	4	
$k+11$	37	0000	0	Salvestuses esines vigu. Peatus. Salvestamist saab korrata *käivitusega.*
$k+12$	22	$k+0$	0	

Esitatuga analoogiline on programm, milles seose (8) abil kontrollitakse magnetlindi tsoonist C massiivi (algab pesast a_3) salvestamist arvuti «Ural-4» ferriitmälu pesadesse a_1 kuni a_2 (summa S_0 kirjutatakse pesadesse a_2+1 ja a_2+2).

$k+0$	45	C	0	Magnetlindilt lugemine.
$k+1$	55	a_1	0	
$k+2$	00	a_3	0	
$k+3$	00	a_2+2	0	
$k+4$	21	$k+14$	0	Kas paarsuse kontrolliga avastati viru?
$k+5$	16	t_1	4	S_2 salvestamine.
$k+6$	20	0000	0	Suuruse $S_3 \varnothing S_0$ arvutamine.
$k+7$	25	a_2-a_1+2	0	
$k+10$	26	a_2+2	0	
$k+11$	24	$k+10$	0	
$k+12$	14	t_1	4	Kas $S_2=S_3 \varnothing S_0$?
$k+13$	21	a	4	Kui võrdub, siis jätkatakse ülesande lahendamist pesas a oleva käsu täitmisega.
$k+14$	37	0000	0	Avastati vigu. Peatus. Salvestamist saab korrata *käivitusega.*
$k+15$	22	$k+0$	0	

Elektronarvuti töös esinevate vigade tõttu võib massiiv välismälus säilitamise ajal moonduda. Seoste $S_3=S_0$, (7) ja (8) kasutamine eespool kirjeldatud kontrollis ei taga aga kõigi seda-

laadi vigade avastamist. Nimelt, kui massiiv mingil põhjusel koos kontrollsummaga S_0 kustutatakse magnetlindilt (magnettrumlilt), siis $S_0 = S_2 = S_3 = 0$ ja need seosed on rahuldatud. Seetõttu on soovitav kontrolli täiendada seose $S_2 = 0$ kontrollimisega.

4. Võtted arvutuste kontrollimiseks valitakse ja kavandatakse juba programmi koostamisel. Seejuures arvestatakse eeskätt ülesande lahendamiseks kuluvat aega ja kasutatava arvuti töökindlust. Näiteks, kui ülesande lahendamisega seotud arvutused kestavad ainult mõne minuti ja nende tegemiseks kasutatakse stabiilselt töötavat arvutit (keskmiselt üks juhuslik viga 8—10 töötunni kohta), siis programmis tavaliselt arvutuste kontrollimist ette ei nähta. Selliseid ülesandeid lahendatakse lihtsalt kaks korda. Kui arvuti on pärast lahendamist tehniliselt korras, siis võrreldakse saadud tulemusi: nende täieliku ühtimise korral loetakse ülesanne vigadeta lahendatuks. Vastasel juhul tuleb aga kogu seda protsessi korrata.

Suuremate ülesannete puhul niisuguse kontrollimisviisi kasutamine üldiselt ei taga usaldusväärsete tulemuste saamist mõistliku ajakuluga. Nimelt on üpris tõenäone, et kahekordne lahendamine annab tulemused, mis võrdlemisel ei ühti. Et arvuti tööaega otstarbekalt kasutada, tuleb selliste ülesannete lahendamiseks koostatavates programmides ette näha arvutuste perioodiline kontrollimine. Selleks jaotatakse ülesande lahendamise protsess etappideks. Täpsemalt, ülesande lahendamisalgoritm liigendatakse nii, et iga saadud osa kirjeldab mingi tervikliku arvutusprotsessi, mis ülesande lahendamisel moodustabki ühe etapi.

Kõrvuti algoritmi struktuuriga tuleb etappide määramisel tingimata arvestada ka nende ajalist kestust. Stabiilselt töötava arvuti korral on otstarbekohane selleks valida 1 kuni 3 minutit; kui aga etapi realiseerimisel kasutatakse väliseid mäluseadmeid, siis 5 kuni 20 minutit.

Iga etapi järel nähaksegi programmis ette arvutuste kontrollimine. Selleks lülitatakse tema vastavatesse kohtadesse blokid, mille realiseerimisega kontrollitakse eelnenud etapi jooksul leitud tulemusi ja arvuti korrasolekut. Kui saadud tulemused on usaldusväärsed, siis alustatakse järgmise etapi realiseerimist. Vastasel korral kas taastatakse lähteandmed ja programm ning lahendatakse uuesti sama etapp või peatatakse arvuti rikete kõrvaldamiseks.

Järgnevas tutvustame arvutuste kontrollimise põhilisi võtteid. Üks kõige lihtsamaid, aga ka kõige aeganõudvamaid võtteid seisneb selles, et kontrollitavat etappi lahendatakse kaks korda järjest ühtede ning samade lähteandmetega. Esimesel ja teisel lahendamisel saadud tulemusi või nende kontrollsummasid võrreldakse. Nende ühtimise korral loetakse saadud tulemused õigeteks ja alustatakse järgmise etapi lahendamist.

Kui esimesel ja teisel lahendamisel saadud tulemused (tähistame nad vastavalt T_1 ja T_2) ei ühti, lahendatakse etappi veel kolmas kord. Sel teel saadud tulemust (või tulemuste kontrollsummat) T_3 võrreldakse T_2 ja T_1-ga. Kui $T_3 = T_2$ või $T_3 = T_1$, siis loetakse etapi kolmandal lahendamisel leitud tulemused õigeteks ja alustatakse järgmise etapi realiseerimist. Vastasel korral töötab arvuti nii ebastabiilselt, et ülesande lahendamist on otstarbekohane jätkata alles pärast tema kordaseadmist.

Järgnevalt esitame elektronarvuti «Ural-4» jaoks koostatud standardse programmi, mille kasutamine lihtsustab kirjeldatud kontrolli. Selle kasutamiseks tuleb etapi igakordse lahendamise järel tuua saadud tulemus (tulemuste kontrollsumma) summaatorisse ja anda juhtimine esitatavasse alamprogrammi. Selleks paigutatakse etappi realiseeriva programmiosa (asugu see pesades a kuni $a + m$) järele käsud:

	$a+m+1$	22	$k+0$	4
	$a+m+2$	22	a	0

Alamprogramm säilitab saadavad tulemused, võrdleb neid omavahel ja sõltuvalt resultaadist annab juhtimise kas etapi kordamiseks pesasse $a + m + 2$ või siis järgmise etapi alustamiseks pesasse $a + m + 3$.

$k+0$	00	0000	0	Etapi viimasel lahendamisel saadud tulemuse salvestamine topeltpesasse $k+30$ 4 ja võrdlemine eelmisel ja üle-eelmisel lahendamisel saadud tulemustega. Kui $T_2 = T_1$, $T_3 = T_2$ või $T_3 = T_1$, siis antakse juhtimine pesasse $k+23$. Etapi esmakordsel lahendamisel võrreldakse T_1 juhuslike arvudega.
$k+1$	16	$k+30$	4	
$k+2$	14	$k+32$	4	
$k+3$	21	$k+23$	4	
$k+4$	02	$k+30$	4	
$k+5$	14	$k+34$	4	
$k+6$	21	$k+23$	4	
$k+7$	02	$k+32$	4	T_1 salvestamine pesasse $k+32$; pärast teistkordset lahendamist aga pesasse $k+34$. T_2 viiakse siis pesasse $k+32$.
$k+10$	16	$k+34$	4	
$k+11$	02	$k+30$	4	
$k+12$	16	$k+32$	4	
$k+13$	02	$k+36$	0	Kas etappi on juba kolm korda lahendatud? Kui pole, siis väljumine lahendamise kordamiseks.
$k+14$	01	$k+26$	0	
$k+15$	16	$k+36$	0	
$k+16$	14	$k+27$	0	
$k+17$	21	$k+0$	0	
$k+20$	37	0000	0	Peatus. $T_3 \neq T_2 \neq T_1$. Etapi lahendamise kordamiseks anda *käivitus*.
$k+21$	16	$k+36$	0	
$k+22$	22	$k+7$	0	

$k+23$	16	$k+36$	0	Saadi kokkulangevad tulemused. Loendaja viimine algseisu ja väljumine järgmise etapi lahendamiseks.
$k+24$	02	$k+30$	4	
$k+25$	30	$k+0$	0	
$k+26$	00	0001	0	
$k+27$	00	0003	0	Konstandid ja tööpesad.
$k+30$	00	0000	0	
$k+31$	00	0000	0	
$k+32$	00	0000	0	
$k+33$	00	0000	0	
$k+34$	00	0000	0	
$k+35$	00	0000	0	
$k+36$	00	0000	0	

Esitame analoogilise alamprogrammi ka elektronarvuti «Minsk-22» jaoks. Selle kasutamiseks tuuakse etapi igakordse lahendamise järel tulemuste kontrollsumma summaatorisse ning pöördutakse käskudega

$a+m+1$	-31	$k+0$	$k+14$
$a+m+2$	-30	a	0000

alamprogrammi poole.

$k+0$	76	00	0000	$k+17$	Etapi viimasel lahendamisel saadud tulemuse salvestamine ja võrdlemine eelmisel ja üle-eelmisel lahendamisel saadud tulemustega. Kui $T_2=T_1$, $T_3=T_2$ või $T_3=T_1$, siis antakse juhtimine pesasse $k+12$. Etapi esimese lahendamise järel võrreldakse T_1 juhuslike arvudega.
$k+1$	05	00	$k+17$	$k+20$	
$k+2$	-34	00	$k+3$	$k+12$	
$k+3$	05	00	$k+17$	$k+21$	
$k+4$	-34	00	$k+5$	$k+12$	
$k+5$	-10	00	$k+20$	$k+21$	T_1 ja T_2 ümbersalvestamine.
$k+6$	-10	00	$k+17$	$k+20$	
$k+7$	20	00	$k+16$	$k+15$	Kas etappi on kolm korda lahendatud?
$k+10$	-34	00	$k+14$	$k+11$	
$k+11$	-00	00	0000	0000	Peatus. $T_3 \neq T_2 \neq T_1$.
$k+12$	-10	00	$k+22$	$k+15$	Loendaja taastamine. Väljumine alamprogrammist.
$k+13$	10	00	$k+16$	$k+14$	
$k+14$	00	00	0000	0000	
$k+15$	00	00	0003	0000	Konstandid ja tööpesad.
$k+16$	00	00	0001	0000	
$k+17$	00	00	0000	0000	
$k+20$	00	00	0000	0000	
$k+21$	00	00	0000	0000	
$k+22$	00	00	0003	0000	

Elektronarvuti töös esinevad juhuslikud vead võivad arvutuste käigus muuta ka ülesande lahendamisel kasutatavat programmi, konstante ja lähteandmeid (nn. põhiinformatsiooni). Sellest tingitud arvutusvead on juba püsiva iseloomuga, mille tõttu neid etappide kahe-kolmekordse lahendamise võttega (mida me eespool käsitlesime) pole võimalik avastada.

Arvutuste ja põhiinformatsiooni säilimise samaaegne kontrollimine võib toimuda näiteks järgmiselt. Iga etapi esmakordset lahendamist alustatakse programmi ja lähteandmete ümberkirjutamisega operatiivmälust välismällu (kus see materjal säilitatakse mitme järgneva etapi jooksul). Seejärel sooritatakse kontrollitaval etapil ette nähtud arvutused ja leitakse saadud tulemuste ning põhiinformatsiooni kontrollsumma S_1. Kasutades välismälus säilitatavat materjali, taastatakse operatiivmälus programm ja lähteandmed nende esialgsel kujul, mille järel korratakse etapi lahendamist ja tulemuste ning põhiinformatsiooni kontroll-liitmist. Saadud summat S_2 võrreldakse S_1-ga. Kui $S_1 = S_2$, siis loetakse tulemused õigeteks ja alustatakse järgmise etapi lahendamist.

Kirjeldatud kontrolli kasutamisel saab ülesande lahendamist katkestada mistahes kohal, ilma et kogu lahendamisprotsessi tuleks algusest peale korrata. Nimelt, seda on võimalik jätkata iga sellise etapi lahendamisest, mille põhiinformatsioon on välismälus säilitatud. Arvestades viimast asjaolu, samal ajal aga ka kirjeldatud kontrolliga seotud suurt ajakulu, kasutatakse vaadeldavat võtet eeskätt just mitmeid tunde kestvate arvutusprotsesside kontrollimiseks. Et tagada arvuti töös esinevate süstemaatiliste vigade õigeaegset avastamist, siis teatud ajavahemike järel (tund kuni poolteist) katkestatakse ülesande lahendamine. Testidega kontrollitakse arvuti korrasolekut ja sõltuvalt selle tulemustest kas jätkatakse ülesande lahendamist alates järgmisest etapist või korratakse kahe sellise kontrolli vahelisi arvutusi.

Mõnede ülesannete puhul kehtivad saadavate arvutustulemuste või siis arvutustulemuste ja algandmete vahel seosed, mida lahendamisprotsessis ei kasutata. Arvuti tööaja ökonoomse kasutamise seisukohalt ongi kõige otstarbekam arvutusi kontrollida just taoliste seoste abil. Nimelt pärast vastavate tulemuste arvutamist asetatakse nad kontrolliks kasutatavasse seosesse ja kui see seos on nõutud täpsusega rahuldatud, siis loetakse saadud tulemused õigeteks. Näiteks võrrandite, võrrandisüsteemide, diferentsiaalvõrrandite ja diferentsiaalvõrrandisüsteemide lahendeid on võimalik kontrollida sel teel, et nad asetatakse vastavasse võrrandisse või võrrandisüsteemi ja veendutakse, kas see on nõutud täpsusega rahuldatud või mitte.

Arvutuste sellisel kontrollimisel on võimalik avastada nii arvuti töös esinevaid juhuslikke kui ka süstemaatilisi vigu. Peale selle aitab kirjeldatud kontroll hinnata ka arvutatud tule-

muste täpsust. Paraku pole aga kaugeltki kõigi ülesannete korral võimalik leida selliseks kontrolliks tarvilikke seoseid.

5. Tulemuste väljastamisel võib arvuti teha nii süstemaatilisi kui ka juhuslikke vigu. Järgnevas peatumegi nende avastamise põhilistel võtetel.

Reeglina kontrollitakse arvuti väljundseadmete tööd testidega nii enne kui ka pärast ülesande lahendamist. Sellega selgitatakse, kas tulemuste väljastamisel oli võimalik süstemaatiliste vigade esinemine. Nimelt, kui pärast ülesande lahendamist arvuti on tehniliselt korras, võis tulemuste väljastamisel esineda ainult juhuslikke vigu, vastasel korral tuleb aga saadud vastuseid pidada mitteusaldusväärseteks ja ülesanne uuesti lahendada. Ülesande lahendamiseks kulutatud tööaeg on sel juhul asjatult raisatud.

Et ökonoomsemalt kasutada arvuti tööaega, on soovitav kontrollida väljundseadmete korrasolekut ka suuremate ülesannete lahendamise käigus. Selleks tuleb koostatavas programmis ette näha mingi teksti või arvumassiivi perioodiline väljastamine. Jälgides kontrollimiseks väljastatavat materjali, otsustatakse küsimus, kas ülesande lahendamist on mõtet jätkata või tuleb see väljundseadmetes esinevate rikete kõrvaldamiseks katkestada.

Ülesande korduvat lahendamist (kui selle tingivad üksnes arvuti väljundseadmete töös esinevad vead) on võimalik vältida ka teisel viisil. Selleks tuleb ülesande lahendamise programmis ette näha kõigi väljastatavate tulemuste salvestamine magnetlindile, kust neid vajaduse korral saab vastava abiprogrammiga uuesti välja trükkida või perforeerida.

Väljundseadmete töös esinevaid juhuslikke vigu niisuguse kontrolliga ei avastata. Kui tulemused perforeeritakse telegraafilindile või perfokaartidele, varustatakse iga väljastatav informatsioonimassiiv kontrollsummaga. Ülesande lahendamise järel salvestatakse selliselt perfokaartidele (või telegraafilindile) kantud tulemusmassiiv uuesti arvutisse, kusjuures salvestamisel arvutatud kontrollsummat võrreldakse lindilt (kaartidelt) loetud summaga. Kontrollsummade ühtelangemise korral loetakse massiiv vigadeta väljastatuks. Vastasel korral tuleb väljaperforeerimist korrata.

Küsimusele, kas trükkimisseadmete töös on esinenud juhuslikke vigu, võib saada vastuse ainult siis, kui ülesande lahendamise tulemusi on välja trükitud vähemalt kaks korda. Nimelt, sel teel saadud tulemusmassiive kõrvutatakse. Kui need ühtivad, on tulemused välja trükitud vigadeta. Vastasel korral on aga väljastamisel esinenud vigu ja trükkimist tuleb korrata.

Kui arvuti komplekti kuulub mitu trükkimisseadet, on eespool kirjeldatud kontrolli soovitatav teha sel teel, et ülesande lahendamisel saadavaid tulemusi trükitakse samaaegselt välja kahel-

kolmel erineval seadmel. Trükkimise niisugune organiseerimine võimaldab ökonoomsemalt kasutada arvuti tööaega. Teiselt poolt muudab ta aga ka kontrolli efektiivsemaks: ühesuguste vigade samaaegne esinemine erinevate trükkimisseadmete töös on üpris ebatõenäone.

Tulemuste korduv väljatrükkimine suurendab ülesande lahendamiseks kuluvat arvuti tööaega (eriti, kui väljastatakse suuri tabeleid ja arvude rühmi). Seetõttu väiksemate ülesannete lahendamiseks koostatavates programmides trükkimise kontrolli ette ei nähta. Tavaliselt lahendatakse neid arvuti töö kontrollimiseks nagunii kaks korda, kusjuures vigade avastamiseks kõrvutatakse sel teel saadud tulemusmassiive.

§ 23. ARVUTUSMEETODI VALIMISE PÕHIMÕTTEID

Mingi ülesande lahendamiseks arvutil (aga samuti ilma arvutita) tuleb kõigepealt leida sobiv lahendusmeetod. Käesolevas paragrahvis vaatleme, mida on vaja silmas pidada lahendusmeetodi valimisel või väljatöötamisel, eeldades, et ülesanne lahendatakse programmjuhtimisega arvuti abil.

1. On ilmne, et valitud meetod peab kõigepealt võimaldama lahendusprotsessi hõlpsat taandamist arvuti põhitehetele (arvuti käskude süsteemi tehetele) või teistele arvutusprotseduuridele, mille jaoks on olemas standardsed alamprogrammid. Automaatse programmeerimise süsteemi olemasolu puhul peab meetod võimaldama hõlpsat üleskirjutamist vastavas algoritmilises keeles (automaatse programmeerimise keeles). Üldiselt öeldes tuleb eelistada niisuguseid meetodeid, kus arvutusprotsess toimub arvutites hästi realiseeritavate algoritmide alusel (polünoomid, elementaarfunktsioonid, tsüklilised arvutused jms.). Ebaotstarbekohasteks tuleb lugeda aga selliseid meetodeid, kus ülesande lahendamine taandub teiste ülesannete lahendamisele, mille jaoks kas efektiivsed lahendusmeetodid puuduvad, või mis toovad endaga kaasa mitmesuguseid teisi arvutuslikke komplikatsioone (näit. väga aeglane koonduvus vms.). Niisuguseid meetodeid on mõeldav kasutada ainult sel juhul, kui paremad meetodid puuduvad.

2. Valitud meetod peab tagama arvutuste nõutava täpsuse. Teatavasti saame ülesannete numbrilisel lahendamisel tulemused ikka ligikaudu, s. t. teatava veaga. Tulemuste koguviga (Δ_s) koosneb meetodi veast (Δ_m) ja nn. aritmeetilisest veast (Δ_a):

$$\Delta_s = \Delta_m + \Delta_a.$$

Meetodi viga tuleneb lahendusalgoritmist enesest, kusjuures eeldatakse, et algandmed on täpsed ja samuti sooritatakse

täpselt aritmeetilised tehted. Reeglina antakse meetodi viga koos vastava lahendusmeetodi esitamisega. Nii näiteks määratud integraali arvutamisel Simpsoni valemiga

$$\int_a^b f(x)dx \approx \frac{b-a}{6n}[(y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + \ldots + 2y_{2n-2} + 4y_{2n-1} + y_{2n}] \quad (1)$$

on meetodi viga

$$\Delta_m \leqslant \frac{(b-a)^5 M}{180(2n)^4},$$

kus

$$M = \sup_{x \in [a,b]} |f^{(4)}(x)|.$$

Monte-Carlo meetodi puhul võib meetodi viga (täpsemini, vea tõenäosust) leida Tšebõševi võrratuse põhjal. Üldiselt sõltuvad meetodid enamasti teatavast naturaalarvulisest parameetrist (jaotuspunktide arv, iteratsioonide arv, arv n Simpsoni valemi puhul, statistiliste katsete arv Monte-Carlo meetodi puhul jne.), mille suurendamisel meetodi viga monotoonselt läheneb nullile.

Aritmeetiline viga tekib aritmeetiliste tehete sooritamisest ligikaudsete arvudega. Paljude meetodite puhul, näiteks diferentsiaalvõrrandite numbrilisel lahendamisel, suureneb aritmeetiline viga (esimeses lähenduses lineaarselt) koos meetodi sammude (jaotuspunktide) arvuga n. Kuid mitmete meetodite (iteratsioonimeetodid, Monte-Carlo meetod) puhul on aritmeetilise vea ülemine tõke konstantne.

Nii näiteks integraali

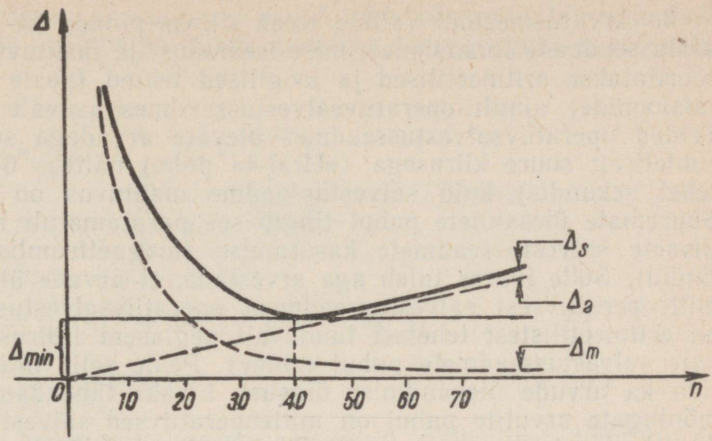
$$I = \int_0^1 \varphi(z)dz, \quad 0 \leqslant \varphi(z) \leqslant 1$$

arvutamisel Monte-Carlo meetodiga, kasutades poollõigul [0,1) ühtlaselt jaotatud juhuslikke arve ξ, saame aritmeetilise vea jaoks järgmise hinnangu

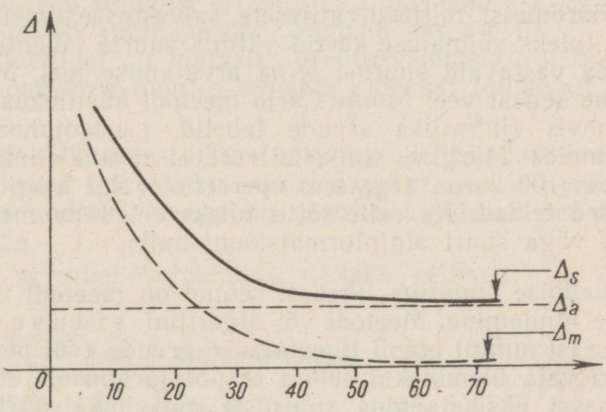
$$\Delta_a \leqslant 2^{-k}[\mathrm{var}(f) + 1],$$

kus $\mathrm{var}(f)$ on funktsiooni $f(x)$ täismuut lõigul $[a, b]$ ja k tähistab pesa kahendkohtade arvu.

Järgnevatel joonistel on kujutatud tulemuse koguvea suuruse muutumine olenevalt meetodi sammude arvust n. Joonisel 49 on kujutatud olukord, kus aritmeetiline viga suureneb lineaarselt ühes sammude arvu n kasvamisega, joonisel 50 on see aga konstantne. Joonisel 49 kujutatud juhtumil saame täpsuse mõttes parima tulemuse $n = 40$ korral. Sammude arvu edasine suurendamine ei võimalda aritmeetilise vea kasvamise tõttu enam täp-



JOON. 49



JOON. 50

sust tõsta. Seega, kui arvutustelt nõutakse suuremat täpsust, kui see on juhul $n = 40$, siis tuleb kas tõsta arvutuste täpsust üksikutel etappidel (joonisel 49 vastab sellele sirge Δ_a tõusu vähenemine), või tuleb antud meetodi asemel valida mõni teine meetod.

Kui aritmeetiline viga on konstantne, on koguvea kõver kujutatud joonisel 50. Arvutusmeetod on antud juhul rakendatav siis, kui aritmeetiline viga Δ_a on väiksem lubatud absoluutsest veast. Vastupidisel juhul tuleb kas tõsta arvutuste täpsust üksikutel etappidel või valida mõni teine meetod.

3. Arvutusmeetodi valikul tuleb silmas pidada ka arvuti salvestusseadmete funktsioneerimise iseärasusi ja mahtuvust. Nimelt sooritatakse aritmeetilised ja loogilised tehted (peale valikuoperatsioonide) ainult operatiivsalvestusseadmes asuvate sõnadega. Tehted operatiivsalvestusseadmes olevate arvudega sooritatakse suhteliselt suure kiirusega («Ural-4» puhul näiteks 6—10 tuhat tehet sekundis), kuid salvestusseadme mahtuvus on ise väike. Suuremate ülesannete puhul tingib see paratamatult mitteoperatiivsete salvestusseadmete kasutamise (magnettrumlid, magnetlindid). Selle juures tuleb aga arvestada, et arvude ülekandmine mitteoperatiivsest salvestusseadmest operatiivsalvestusseadmesse on aritmeetilistest tehetest tunduvalt aeglasem (kiirus on erinevate salvestusseadmete puhul erinev). Peale selle tuleb kontrollida ka arvude ülekandmise õigsust. Lisaks tähendame veel, et mõningate arvutite puhul on mitteoperatiivsed salvestusseadmed (eriti salvestusseade magnetlintidel) väiksema töökindlusega kui operatiivsalvestusseade.

Kõike öeldut arvestades on otstarbekohane eelistada arvutusmeetodit, mille alusel koostatavas programmis esineb võimalikult vähe pöördumisi mitteoperatiivsete salvestusseadmete poole. Nii näiteks tuleks võimaluse korral vältida suurte tabelite kasutamist ja püüda vastavaid suurusi leida arvutamise teel. Mööda minnes kõneleme sellest veel Monte-Carlo meetodi käsitlemisel järgmises paragrahvis (juhuslike arvude tabelid, pseudojuhuslike arvude programmid). Märgime siinkohal veel, et arvude viimine arvutisse on umbes 100 korda aeglasem operatsioon kui loogilised ja aritmeetilised tehted. Ka selle tõttu tuleks eelistada meetodeid, mis ei nõua väga suurt alginformatsiooni hulka.

4. Eelmise punktiga tihedalt seotud on meetodi või algoritmi sidususe hindamine. Meetodi või algoritmi s i d u s u s e k s arvutusprotsessi mingil etapil nimetatakse arvude (või pesade) hulka, mida on vaja üleminekul sellelt etapilt järgmisele etapile. Arvutusprotsessi üksiketappide sidususte maksimaalväärtust nimetatakse meetodi kogusidususeks ehk lihtsalt meetodi sidususeks. See annab ülesande lahendamiseks tingimata vajalike tööpesade arvu. Tähendame, et ülesande tegelikul lahendamisel arvuti abil läheb küll tööpesi vaja mõnevõrra rohkem, sest neid on vaja reserveerida veel näiteks loendajateks, arvutuste kontrollimiseks jne. Meetodi sidususe hindamine on oluline eriti operatiivsalvestusseadme kasutamise planeerimisel, mis on teatavasti suhteliselt väikese mahtuvusega.

Esitame nüüd mõned näited sidususe leidmise kohta.

N ä i d e 1. Olgu vaja lahendada diferentsiaalvõrrand

$$y' = f(x, y)$$

algtingimustega $x = x_0$, $y = y_0$. Antud võrrandi lahendamisel Runge-Kutta meetodiga tuleb arvutusi teha järgmiste valemite põhjal:

$$k_1^{(i)} = hf(x_i, y_i),$$

$$k_2^{(i)} = hf\left(x_i + \frac{h}{2}, y_i + \frac{k_1^{(i)}}{2}\right),$$

$$k_3^{(i)} = hf\left(x_i + \frac{h}{2}, y_i + \frac{k_2^{(i)}}{2}\right),$$

$$k_4^{(i)} = hf(x_i + h, y_i + k_3^{(i)}),$$

$$\Delta y_{i+1} = \frac{1}{6}(k_1^{(i)} + 2k_2^{(i)} + 2k_3^{(i)} + k_4^{(i)}),$$

$$y_{i+1} = y_i + \Delta y_{i+1},$$
$$x_{i+1} = x_i + h,$$

kus h on integreerimissamm. Esitatud valemitest nähtub, et $(i+1)$-se punkti (x_{i+1}, y_{i+1}) arvutamiseks on vaja teada ja salvestada suurusi x_i, y_i, $k_1^{(i)}$, $k_2^{(i)}$, $k_3^{(i)}$ ja $k_4^{(i)}$. Suurust Δy_{i+1} pole muide salvestada vaja, sest seda võib kasutada vahetult järgmisel sammul. Kui eeldada, et sidusus funktsiooni $f(x, y)$ arvutamisel on s, siis on antud meetodi kogusidusus $s + 6$.

Paljudel juhtudel on võimalik arvutusvalemeid või arvutuste järjekorda mõnevõrra muutes meetodi sidusust tunduvalt vähendada.

Näide 2. Arvutades määratud integraali Simpsoni valemiga (1), on meil vaja salvestada suurused $y_0, y_1, y_2, \ldots, y_{2n}$, kokku seega $2n + 1$ suurust. Seega on algoritmi sidusus sellel arvutuste etapil $2n + 1$.

Kui aga arvutusi korraldada näiteks selliselt

$$S_0 = 0,$$

$$\Delta S_i = \frac{b-a}{6n}[y_{2i} + 4y_{2i+1} + y_{2i+2}],$$

$$S_{i+1} = S_i + \Delta S_i \quad (i = 0, 1, 2, \ldots, n-1)$$

kus integraali väärtus tuleb

$$\int_a^b f(x)\,dx = S_n,$$

on S_{i+1} arvutamiseks vaja ainult suurusi y_{2i}, y_{2i+1}, y_{2i+2} ja S_i. Seega on algoritmi sidusus sel korral ainult 4.

Näide 3. Lineaarse võrrandisüsteemi lahendamisel Monte-Carlo meetodiga (vt. § 24, näide 4) on vaja kontrollida iga k-dimensionaalse juhusliku vektori

$$(\xi_1^i, \xi_2^i, \ldots, \xi_k^i) \tag{2}$$

puhul tingimuse

$$F(x_1, x_2, \ldots, x_k) \leqslant c \tag{3}$$

täidetust. Tingimust (3) rahuldavad vektorid (2) tuleb säilitada. Tähistades nende arvu m abil, võib öelda, et säilitada tuleb mk arvu (s. t. meetodi sidusus sellel etapil on mk). Vektorite säilitamine on vajalik vastavate vektorite koordinaatide järjestamiseks kasvavas järjekorras, et pärast järjestamist saaksime leida jada keskel asuva koordinaadi $\tilde{\xi}_j^{\frac{m+1}{2}}$ (või kaks keskmist $\tilde{\xi}_j^{\frac{m}{2}}$ ja $\tilde{\xi}_j^{\frac{m}{2}+1}$). Kui koordinaate järjestada aga paralleelselt tingimuse (3) kontrollimisega, oleks vaja ainult salvestada $\frac{m}{2}+1$ antud hetkel kõige väiksemat koordinaati. Kokku oleks vaja salvestada seega ainult $k\left(\frac{m}{2}+1\right)$ arvu, s. t. umbes poole vähem kui eelmisel juhul. Kui ellipsoidi keskpunkti arvutamisel kasutada aga valemit

$$\bar{x}_j = \frac{1}{m}\sum_{\nu=1}^{m} \xi_j^{\nu},$$

siis on vaja keskpunkti iga koordinaadi arvutamiseks kaht arvu ja sidusus tuleb sel juhul ainult $2k$.

5. Arvutusmeetod peab võimaldama arvutuste hõlpsat kontrollimist. Teatavasti esinevad juhuslikud vead ka täiesti korrasoleva arvuti töös. Seetõttu tuleb vähegi suurema ülesande lahendamisel kavandada arvutuste õigsuse kontroll arvuti enese abil. Et sellest küsimusest on üksikasjalikumalt kõneldud juba eespool, siis me sellel siin enam ei peatu.

6. Meetodi valikul tuleb silmas pidada ka tema majanduslikku ökonoomsust. Üldiselt öeldes, kui ülesande lahendamise tähtaeg lubab, tuleb lahendusmeetod valida selline, et see võimaldaks ülesande lahendada kõige väiksemate kulutustega, s. t. et suurus

$$M_1 + M_2 + M_3$$

(kus M_1 on arvutusmeetodi väljatöötamise või valimise maksumus, M_2 — programmeerimise või vastavalt lahendusmeetodi algoritmilises keeles esitamise maksumus, M_3 — arvuti aja maksumus) oleks minimaalne. Maksumuse hindamisel tuleb arvestada vastavate töötajate töötasusid ja loomulikult arvuti töötunni maksumust, mis näiteks arvuti «Ural-4» puhul on 50 rbl. Massiliselt lahendatavate spetsiaalset tüüpi ülesannete korral on otstarbekohane valida või välja töötada selline meetod, mille puhul ülesande lahendusaeg oleks minimaalne. Meetodi valimisele või väljatöötamisele ning programmeerimisele võib sel juhul kulutada ka mõnevõrra rohkem aega, sest programmi hoolikamal viimistlemisel saavutatav väike masinaaja kokkuhoid võib muutuda programmi massilisel kasutamisel juba küllaltki oluliseks.

§ 24. MONTE-CARLO MEETOD

1. Programmjuhtimisega arvutite loomine ja areng on põhjustanud arvutusmeetodite valdkonnas mitmeidki ümberhinnanguid. Nii näiteks tunnistatakse tänapäeval efektiivseteks mõned sellised meetodid, mida varem (ilma kiirelt töötavate arvutiteta) polnud üldse mõeldav kasutada. Üheks niisuguseks on *Monte-Carlo ehk statistiliste katsete meetod*, mille laialdasem uurimine algas käesoleva sajandi 50-nendatel aastatel.

Monte-Carlo meetodi põhimõte seisneb selles, et ülesannete lahendamine taandatakse teatavate lõikude pikkuste, kujundite pind- või ruumalade (täpsemini, teatavate hulkade mõõtude) ligikaudsele leidmisele suure hulga juhuslike katsete sooritamise teel. Meetodi rakendamine eeldab seega, et meil on võimalik vajalikul hulgal kasutada etteantud omadustega juhuslikke arve. Meenutamegi kõigepealt mõningaid juhuslike arvudega seotud olulisemaid mõisteid.

Juhuslikke arve (resp. juhuslike arvude jada ehk voogu) iseloomustatakse mitmete karakteristikutega. Üheks olulisemaks karakteristikuks on juhuslike arvude ξ *jaotusfunktsioon* $F_\xi(x)$, mis defineeritakse järgmiselt

$$F_\xi(x) = P\{\xi < x\}, \tag{1}$$

s. t. $F_\xi(x)$ on tõenäosus selleks, et juhuslik arv ξ osutub väiksemaks etteantud arvust x. Valemi (1) põhjal saame kergesti leida ka tõenäosuse selleks, et juhuslik arv ξ asub poollõigul $[\alpha, \beta)$:

$$P\{\alpha \leqslant \xi < \beta\} = F_\xi(\beta) - F_\xi(\alpha). \tag{2}$$

Juhuslike arvude ξ *tihedusfunktsiooniks* nimetatakse funktsiooni $f_\xi(x)$, mis iga x korral rahuldab tingimust

$$F_\xi(x) = \int_{-\infty}^{x} f_\xi(t)\,dt. \tag{3}$$

Võrduste (2) ja (3) põhjal leiame, et mistahes α ja β korral kehtib seos

$$P\{\alpha \leqslant \xi < \beta\} = \int_{\alpha}^{\beta} f_\xi(t)\,dt. \tag{4}$$

Võrdusest (3) lähtudes saab kergesti näidata, et punktides x, kus $F_\xi(x)$ on diferentseeruv, kehtib seos

$$f_\xi(x) = \frac{d}{dx} F_\xi(x). \tag{5}$$

Juhuslike arvude jada karakteristikuks on ka tõketega a ja b $(a < b)$ määratav *diapasoon*, mille vahel asub selle jada iga arv, s. t. iga ξ korral kehtib

$$a \leqslant \xi \leqslant b.$$

Jada mitmed teised karakteristikud (keskväärtus, dispersioon) on juba arvutatavad eespool toodud karakteristikute abil. Muide, nagu nägime valemist (5), pole ka toodud karakteristikud üksteisest sõltumatud.

Monte-Carlo meetodi puhul kasutatakse kõige enam mingil lõigul (või vahemikus) ühtlaselt jaotatud juhuslike arvude jada. Sellise jaotusega arvude jada on suhteliselt lihtne saada ja hõlpus ka rakendada. Sõovi korral on niisugust jada võimalik teisendada praktiliselt mistahes etteantud jaotusfunktsiooniga arvude jadaks.

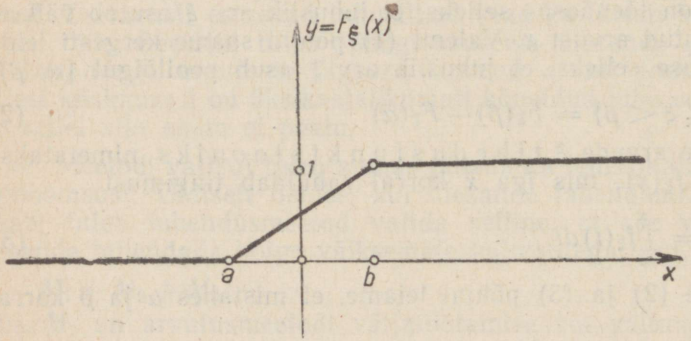
Lõigul $[a, b]$ ühtlaselt jaotatud juhuslike arvude jada korral avaldub jaotusfunktsioon kujul

$$F_\xi(x) = \begin{cases} 0, & \text{kui } x \leqslant a, \\ \dfrac{x-a}{b-a}, & \text{kui } a \leqslant x \leqslant b, \\ 1, & \text{kui } x \geqslant b, \end{cases}$$

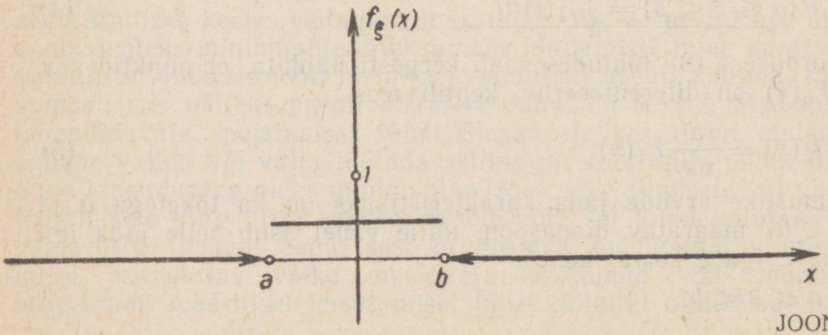
ja tihedusfunktsioon kujul

$$f_\xi(x) = \begin{cases} 0, & \text{kui } x < a \text{ või } x > b, \\ \dfrac{1}{b-a}, & \text{kui } a \leqslant x \leqslant b. \end{cases}$$

Esitame ka nende funktsioonide graafikud joonistel 51 ja 52.



JOON. 51



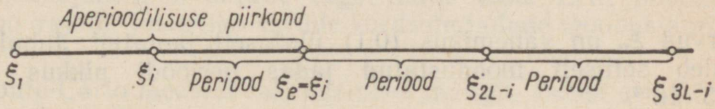
JOON. 52

2. Etteantud karakteristikutega juhuslike arvude jada saamiseks on mitmeid võimalusi.

1) On võimalik kasutada spetsiaalseid *juhuslike arvude tabeleid*. Et välistada seaduspärasuste (perioodide) esinemist tabelist leitavate arvude vahel, peab tabel olema küllalt suur. Sel juhul pole aga tabelit võimalik mahutada arvuti operatiivsesse salvestusseadmesse, mistõttu tabeli kasutamine on aegavõttev. Kõige selle tõttu ei saa juhuslike arvude tabeli kasutamist lugeda eriti otstarbekohaseks mooduseks.

2) Juhuslikke arve võib arvutis leida ka spetsiaalsete programmide abil. Sel viisil saadud arve nimetatakse *pseudojuhuslikeks* (ka kvaasijuhuslikeks) arvudeks. Jada järjekordne arv ξ_n leitakse vastava algoritmi $\mathfrak{A}$ abil k eelmisest juhuslikust arvust, s. t.

$$\xi_n = \mathfrak{A}(\xi_{n-1}, \xi_{n-2}, \ldots, \xi_{n-k}). \qquad (6)$$



JOON. 53

Kuna arvuti pesas on kindel hulk kahendkohti, siis seosest (6) järeldub, et arvud ξ_n hakkavad teatavast indeksist alates perioodiliselt korduma (olukorda iseloomustab joonis 53). Oluline on seejuures hinnata nn. *aperioodsuse piirkonna* (arvude hulk jada algusest kuni perioodi ilmnemiseni) suurust. See oleneb loomulikult algoritmist $\mathfrak{A}$, aga samuti ka lähtearvude $\xi_1, \xi_2, \ldots$ $\ldots, \xi_k$ valikust. Aperioodsuse piirkond pikeneb üldiselt veel argumendi kohtade arvu k suurendamisega, kuid suurema k korral muutub reeglina suuremaks arvutuste maht. Seepärast kasutatakse valemeid (6) vaid väikeste k väärtuste korral ($k = 1$, 2 või 3). Juhtudel, kus pseudojuhuslike arvude jada ei rahulda täielikult vastavaid nõudeid (pole näiteks kooskõlas arvude ühtlase jaotuse hüpoteesiga) või kui sellist arvude jada analüüsi pole üldse läbi viidud, on otstarbekohane perioodiliselt (näiteks iga 15 arvu järel) teha genereeritavate pseudojuhuslike arvude juhuslikke *häiritusi*. Selleks võib kasutada näiteks mõningates tööpesades olevate arvude kontroll-liitmist (liitmise asemel võib kasutada ka mitteekvivalentsi tehet). Muide, selliste *häirituste* korral pikeneb üldiselt ka aperioodsuse vahemik. Nii on võimalik näidata, et kui *häiritusi* teha iga m pseudojuhusliku arvu järel, siis pikeneb periood $\sqrt{m}$ korda. Perioodi pikenemine suurendab aga omakorda aperioodsuse vahemikku.

Esitame nüüd mõningaid konkreetseid näiteid pseudojuhuslike arvude leidmise algoritmide kohta.

A. Genereerimisalgoritm, kus põhimõtteks on korrutise *keskmise osa* eraldamine. Lähtearve on kaks: ξ_1 ja ξ_2, mis koosnevad $2n$ kahendkohast. Arvude ξ_1 ja ξ_2 korrutises on $4n$ kahendkohta, kuid sellest võetakse arvesse ainult kahendkohad järjekorranumbrist $n+1$ alates kuni järjekorranumbrini $3n$. Nendest saame uue pseudojuhusliku arvu ξ_3, mis koosneb seega samuti $2n$ kahendkohast. Protsess jätkub arvudega ξ_2 ja ξ_3 jne. Arvud ξ_n on vahemikus (0,1) ühtlaselt jaotatud.

B. Arvutuseeskiri, kus kasutatakse jada järgmise liikme saamiseks kahe eelmise arvu liitmist (nn. Fibonacci jada põhimõte). Arvutused tehakse järgmiselt:

$$a_0 = 0, \quad a_1 = 1,$$
$$a_{n+2} = a_n + a_{n+1} \pmod{M},$$
$$\xi_n = \frac{1}{M} a_{n+2}.$$

Arvud ξ_n on vahemikus (0,1) ühtlaselt jaotatud. Juhul $M = 2^{44}$ tuleb selliselt moodustatud jadas perioodi pikkus ligikaudu $2{,}5 \cdot 10^{13}$.

C. Alljärgnev programm arvuti «Ural-4» jaoks võimaldab leida vahemikus (0,1) ühtlaselt jaotatud juhuslikke arve:

$(l+0) = 00\ \ 0000\ \ 0$ $(l+1) = 02\ \ m+0000\ \ 4$	}	Pseudojuhusliku arvu $\pm\xi_n$ toomine summaatorisse.
$(l+2) = 11\ \ 0001\ \ 0$	}	Summaatoris oleva arvu loogiline nihutamine ühe koha võrra vasakule.
$(l+3) = 16\ \ m+0002\ \ 4$	}	Nihutatud arvu salvestamine.
$(l+4) = 02\ \ m+0000\ \ 4$	}	Arvu $\pm\xi_n$ toomine summaatorisse.
$(l+5) = 11\ \ 0101\ \ 0$	}	Loogiline nihutamine ühe koha võrra paremale.
$(l+6) = 14\ \ m+0002\ \ 4$	}	Nihutatud arvude mitteekvivalents.
$(l+7) = 16\ \ m+0000\ \ 4$		Arvu $\pm\xi_{n+1}$ salvestamine.
$(l+10) = 10\ \ l+1\ \ 0$		Arvu $+\xi_{n+1}$ saamine.
$(l+11) = 22\ \ l+0\ \ 0$		Programmist väljumine.

Esitatud programmi poole pöördumine toimub käsuga $(a+0) = = 22\ l + 0000\ 4$, mis pärast programmi täitmist annab juhtimise automaatselt tagasi põhiprogrammile, nimelt pesas $a+1$ asuvale käsule, kusjuures summaatoris asub järgmine pseudojuhuslik arv ξ_{n+1}.

Nihutamise asemel ühe koha võrra vasakule ja ühe koha võrra paremale on katsetatud ka teisi võimalusi, näiteks nihutamist seitsme koha võrra. Lähtearvuks ξ_1 tuleb valida arv, kus nullid ja ühed paiknevad ebakorrapäraselt. Otstarbekohane on toodud algoritmi täiendada perioodiliste *häirituste* rakendamisega, millest oli juttu eespool.

3) Monte-Carlo meetodi ulatuslikuma rakendamise korral on kõige otstarbekohasem kasutada juba spetsiaalseid seadmeid — juhuslike arvude generaatoreid ehk a n d u r e i d. Andurid võimaldavad saada kiiresti suurel hulgal kindla jaotusfunktsiooniga juhuslikke arve, kusjuures jaotusfunktsioon sõltub anduri konstruktsioonist. Anduri töö põhineb füüsikalistel juhuslikel protsessidel. Tavaliselt kasutatakse selleks otstarbeks radioaktiivsete ainete kiirgust või mürasid elektronlampides. Tuleb märkida, et andurite tööd tuleb kontrollida vastavate testide abil samuti nagu kontrollitakse pseudojuhuslike arvude algoritmide tööd. Eriti hoolikas kontroll on vajalik juhuslike arvude kordsete jadade moodustamise puhul.

3. Monte-Carlo meetodi rakendamisel mängib olulist osa T š e b õ š e v i v õ r r a t u s

$$P\{|\nu - p| < \varepsilon\} \geqslant 1 - \frac{p(1-p)}{n\varepsilon^2}, \tag{7}$$

kus p tähistab mingi sündmuse tõenäosust, ν selle sündmuse esinemissagedust n katse puhul (s. t. $\nu = \frac{m}{n}$, kus m on vaadeldava sündmuse esinemiste arv n katse puhul) ja ε on suvaline positiivne suurus.

Tšebõševi võrratuse abil on võimalik hinnata antud ülesande lahendamiseks vajalike juhuslikkude arvude hulka. Monte-Carlo meetodi rakendamisel leitakse lahendi tõelise väärtuse p ligikaudne väärtus $\nu = \frac{m}{n}$. Et hinnata viga $|\nu - p|$ või täpsemini vea tõenäosust, kasutamegi valemit (7). Valemist (7) leiame, et võrratuse $|\nu - p| < \varepsilon$ kehtivuse tõenäosus P on mitte väiksem avaldisest

$$1 - \frac{p(1-p)}{n\varepsilon^2}.$$

Tõenäosust P nimetatakse hinnangu $|\nu - p| < \varepsilon$ u s a l d a t a v u s e k s. Võrratusest (7) nähtub, et katsete arvu n suurenedes suureneb ka usaldatavus P.

Kui anda ette teatav usaldatavuse väärtus δ $(0 < \delta < 1)$ ja fikseerida ε, siis on lihtne leida katsete koguarvu n, mille puhul

$$P\{|\nu - p| < \varepsilon\} \geqslant \delta. \tag{8}$$

Kasutame selleks võrratuse (7) paremat poolt ja nõuame, et

$$1 - \frac{p(1-p)}{n\varepsilon^2} \geqslant \delta. \tag{9}$$

Pärast mõningaid teisendusi saame selle kirjutada kujul

$$n \geqslant \frac{p(1-p)}{(1-\delta)\varepsilon^2}. \tag{10}$$

Võrratused (9) ja (10) on samaväärsed. Sellest järeldub, et juhul kui katsete arv n rahuldab võrratust (10), kehtib võrratus (9) ja seega ka võrratus (8).

Siinkohal tuleb tähendada, et suurus p on meile enne ülesande lahendamist teadmata. Paremal juhul võime teada ainult p ligikaudset väärtust, halvemal juhul teame aga ainult seda, et $0 < p < 1$. Viimasel juhul saame vaid asendada võrratuse (10) lugeja $p(1-p)$ tema maksimaalse väärtusega $\frac{1}{4}$, mis saadakse $p = \frac{1}{2}$ korral. Seega leiame uue hinnangu

$$n \geqslant \frac{1}{4(1-\delta)\varepsilon^2}, \tag{11}$$

mis ei sõltu üldse p väärtusest. Mõistagi annab võrratus (11) katsete arvu n jaoks vajalikest tunduvalt suuremaid väärtusi.

Katsete arvu n hindamisel võetakse usaldatavus δ tavaliselt vahemikus $0{,}95 \div 0{,}997$, kuna ε valitakse piirides $0{,}01 \div 0{,}001$.

4. Monte-Carlo meetodit on võimalik rakendada väga mitmesuguste ülesannete lahendamisel, näiteks määratud integraalide arvutamisel, võrrandite ja võrrandisüsteemide lahendamisel, pöördfunktsioonide arvutamisel, matemaatilise füüsika võrrandite lahendamisel, stohhastilisel (tõenäosusteoreetilisel) modelleerimisel jne. Järgnevalt esitamegi mõningaid näiteid selle meetodi rakendamise kohta.

N ä i d e 1. Määratud integraali arvutamine. Olgu vaja arvutada integraal

$$I = \int_{\alpha}^{\beta} f(x)\,dx, \tag{12}$$

eeldusel, et integreerimisvahemik $[\alpha, \beta]$ on lõplik ja funktsioon $f(x)$ selles piirkonnas tõkestatud, s. t.

$$m_1 \leqslant f(x) \leqslant m_2, \qquad x \in [\alpha, \beta]. \tag{13}$$

Ülesande lahendamisel eeldame, et meil on võimalik kasutada juhuslike arvude generaatorit või programmi, mis tekitab vahemikus (0,1) ühtlaselt jaotatud juhuslike arvude jada.

Integraali (12) arvutamine on otstarbekohane taandada ühe sellise integraali arvutamisele, kus integreerimine toimub lõigul $[0, 1]$ ning integreeritav funktsioon on tõkestatud tõketega 0 ja 1. See toimub kahe teisendusega. Kõigepealt teeme muutuja vahetuse

$$x = \alpha + (\beta - \alpha) z \qquad (z \in [0, 1]),$$

millega saame

$$I = (\beta - \alpha) \int_0^1 f(\alpha + (\beta - \alpha) z)\, dz$$

ehk

$$I = (\beta - \alpha) \int_0^1 f_1(z)\, dz. \tag{14}$$

Edasi avaldame integreeritava funktsiooni kujul

$$f_1(z) = (m_2 - m_1) \frac{f_1(z) - m_1}{m_2 - m_1} + m_1, \tag{15}$$

kus võrratusest (13) järeldub, et

$$0 \leqslant \frac{f_1(z) - m_1}{m_2 - m_1} \leqslant 1, \qquad z \in [0, 1]. \tag{16}$$

Avaldise $\dfrac{f_1(z) - m_1}{m_2 - m_1}$ tähistame lühiduse huvides sümboliga $\varphi(z)$, siis avaldub võrratus (16) kujul

$$0 \leqslant \varphi(z) \leqslant 1, \qquad z \in [0, 1].$$

Asendades nüüd avaldises (14) integreeritava funktsiooni samasuse (15) alusel, saame

$$I = (\beta - \alpha)(m_2 - m_1) \int_0^1 \varphi(z)\, dz + (\beta - \alpha) m_1,$$

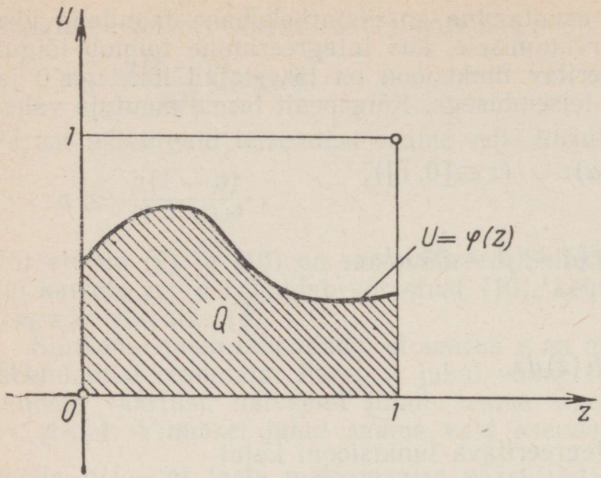
kus põhiliseks probleemiks on integraali

$$I_1 = \int_0^1 \varphi(z)\, dz \tag{17}$$

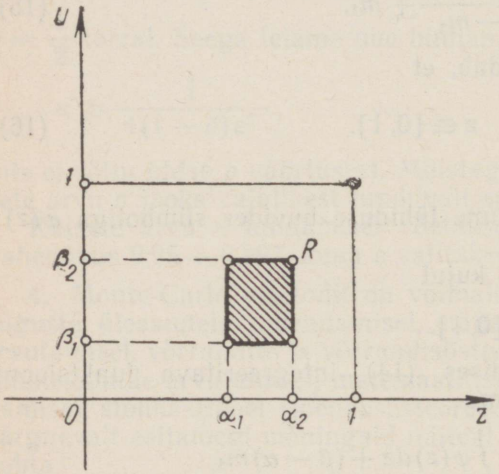
arvutamine.

Integraali (17) arvutamisel kasutame asjaolu, et integraali võib geomeetriliselt tõlgendada kui sirgetega $z = 0$, $z = 1$ ja $u = 0$ ning kõveraga $u = \varphi(z)$ piiratud kujundi Q pindala (vt. joonis 54).

Kujundi Q pindala arvutamise selgitamisel lähtume ristküliku pindala arvutamisest. Olgu ühikruudus antud ristkülik $R[\alpha_1, \alpha_2;$



JOON. 54



JOON. 55

$\beta_1, \beta_2]$, mille küljed on paralleelsed telgedega (vt. joonis 55). Selle ristküliku R pindala

$$S_R = (\alpha_2 - \alpha_1)(\beta_2 - \beta_1)$$

avaldub integraalide abil kujul

$$S_R = \int_{\alpha_1}^{\alpha_2} d\xi \cdot \int_{\beta_1}^{\beta_2} d\eta. \tag{18}$$

Valemi (18) võime kirjutada ka tõenäosuste abil, eeldades et juhuslikud arvud ξ ja η on vahemikus (0, 1) ühtlaselt jaotatud. Nii saame

$$S_R = P\{\alpha_1 \leqslant \xi \leqslant \alpha_2\} \cdot P\{\beta_1 \leqslant \eta \leqslant \beta_2\}.$$

Arvestades, et sündmused «$\alpha_1 \leqslant \xi \leqslant \alpha_2$» ja «$\beta_1 \leqslant \eta \leqslant \beta_2$» on sõltumatud, võime kirjutada

$$S_R = P\{(\alpha_1 \leqslant \xi \leqslant \alpha_2) \wedge (\beta_1 \leqslant \eta \leqslant \beta_2)\},$$

kus loogelistes sulgudes tähistatud sündmus tähendab sisuliselt seda, et punkt (ξ, η) asub ristkülikus R.
Seega

$$S_R = P\{(\xi, \eta) \in R\}. \tag{19}$$

Saadud tulemust on võimalik üldistada mitmes suunas. Nii näiteks kui kujund K avaldub lõpliku hulga ristkülikute R_i summana

$$K = \bigcup_{i=1}^{n} R_i,$$

kus ristkülikutel R_i pole ühiseid sisepunkte, siis saame

$$S_K = \sum_{i=1}^{n} S_{R_i} = \sum_{i=1}^{n} P\{(\xi, \eta) \in R_i\} =$$

$$= P\{(\xi, \eta) \in \bigcup_{i=1}^{n} R_i\} = P\{(\xi, \eta) \in K\}. \tag{20}$$

Teisendamisel arvestasime siin asjaolu, et ristkülikutel R_i pole ühiseid sisepunkte ja punkti (ξ, η) ristkülikute R_i rajajoonele sattumise tõenäosus on 0. Seega võib lugeda, et sündmused «$(\xi, \eta) \in R_i$» on üksteist välistavad ning järelikult tohib rakendada tõenäosuste liitmisteoreemi.

Teiseks on võimalik kergesti näidata, et valemid (19) ja (20) kehtivad mitte ainult tasapinnalisel juhul, vaid mistahes k-dimensionaalses eukleidilises ruumis, kus ristkülikute asemel tuleb vastavalt vaadelda k-dimensionaalseid risttahukaid.

Pöördume nüüd tagasi integraali (17) arvutamise juurde. Teatavasti defineeritakse see kui teatava lõpliku summa (nn. integraalsumma) piirväärtus. Kuid integraali saab avaldada ka ülem- või alamsumma abil. Kasutades näiteks alamsummat, saame kirjutada

$$\int_0^1 \varphi(z)\,dz = \lim_{\max \Delta z_i \to 0} \sum_i \mu_i \Delta z_i,$$

kus $\mu_i = \min_{z \in [z_i, z_{i+1}]} \varphi(z)$ ja $\Delta z_i = z_{i+1} - z_i$. Küllalt peene alajaotuse korral erineb alamsumma vastavast piirväärtusest (integraalist) küllalt vähe, s. t.

$$\int_0^1 \varphi(z)\,dz \approx \sum_i \mu_i \Delta z_i.$$

Et alamsumma kujutab geomeetriliselt ilma ühiste sisepunktideta ristkülikute lõplikku summat, on selle puhul rakendatav seos (20). Piirile üle minnes saame

$$\int_0^1 \varphi(z)\,dz = P\{(\xi, \eta) \in Q\}. \tag{21}$$

Seega taandub integraali (17) arvutamine tõenäosuse $P\{(\xi, \eta) \in Q\}$ arvutamisele. Viimase leidmisel toimime järgmiselt. Fikseerime kõigepealt positiivsed arvud δ ja ε ning leiame võrratuse (11) põhjal katsete koguarvu n. Seejärel tekitame kaks juhuslike arvude jada

$$\begin{matrix} \xi_1, \xi_2, \ldots, \xi_n; \\ \eta_1, \eta_2, \ldots, \eta_n, \end{matrix} \tag{22}$$

kus arvud ξ_i ja η_i on vahemikus (0, 1) ühtlaselt jaotatud. Praktiliselt tekitatakse arvud ξ_i ja η_i vaheldumisi:

$$\xi_1, \eta_1, \xi_2, \eta_2, \ldots, \xi_n, \eta_n.$$

Edasi kontrollitakse iga arvupaari (ξ_i, η_i) puhul tingimust $(\xi_i, \eta_i) \in Q$, mis on samaväärne võrratusega $\varphi(\xi_i) \geqslant \eta_i$. Kontrolli käigus loendatakse need juhud, kus ülalmärgitud tingimus on täidetud. Tähistades nende juhtude arvu m abil, leitakse suhe $\nu = \frac{m}{n}$, mis ongi ülesande lahendi $P\{(\xi, \eta) \in Q\}$ ligikaudseks väärtuseks.

Analoogiline tulemus kehtib ka kordsete integraalide korral. Nimelt, kui integreeritav funktsioon rahuldab tingimust

$$0 \leqslant \varphi(z_1, z_2, \ldots, z_k) \leqslant 1 \qquad (z_i \in [0, 1],\ i = 1, 2, \ldots, k),$$

kehtib valem

$$\int_0^1 \int_0^1 \ldots\ldots \int_0^1 \varphi\ (z_1, z_2, \ldots, z_k)\,dz_1\,dz_2 \ldots dz_k =$$

$$= P\{(\xi_1, \xi_2, \ldots, \xi_k, \eta) \in Q\} = P\{\varphi(\xi_1, \xi_2, \ldots, \xi_k) \geqslant \eta\},$$

kus Q tähistab võrratustega $0 \leqslant z_i \leqslant 1$ $(i = 1, 2, \ldots, k)$ ja $0 \leqslant u \leqslant \varphi\ (z_1, z_2, \ldots, z_k)$ defineeritud kujundit. Ülesande lahendamisel Monte-Carlo meetodiga on vaja $k+1$ jada vahemikus (0, 1) ühtlaselt jaotatud juhuslikest arvudest. Praktiliselt talitatakse nii, et rühmitatakse ühe jada liikmed $(k+1)$-liikmelisteks rühmadeks

$$\xi_1, \xi_2, \ldots \xi_k, \eta.$$

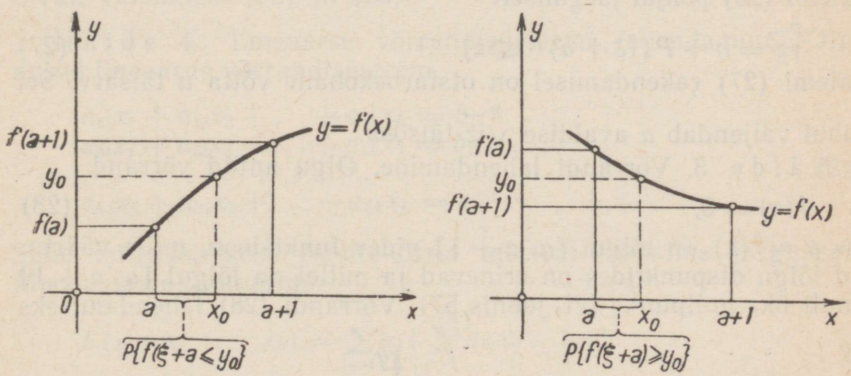
Seejärel leitakse nende rühmade arv m, mis rahuldavad tingimust

$$\varphi(\xi_1, \xi_2, \ldots \xi_k) \geqslant \eta.$$

Suhe $\nu = \frac{m}{n}$ on ka antud juhul k-kordse integraali ligikaudseks väärtuseks.

Tähendame, et määratud integraalide arvutamisel ristküliku-, trapetsivalemi või Simpsoni valemi abil on sõlmpunktide arvu N ja kordsuse k vaheline seos $N = \tau^k$, kus τ on jaotuspunktide arv lõigul $[0, 1]$. Kogu arvutustöö maht on aga võrdeline arvuga N ja kasvab seega k suurenemisel eksponentsiaalselt. Monte-Carlo meetodi puhul on k-kordse integraali arvutamisel üheks statistiliseks katseks vaja $k + 1$ juhuslikku arvu. Seega kasvab arvutustöö maht Monte-Carlo meetodi puhul integraali kordsuse suurenemisel ainult lineaarselt. Sellest nähtub, et Monte-Carlo meetodid on eriti efektiivsed just suure kordsusega integraalide arvutamisel.

N ä i d e 2. Pöördfunktsiooni väärtuste arvutamine. Olgu antud funktsioon $y = f(x)$, mis on pidev ja monotoonne mingil lõigul $[a, a + 1]$. Meie ülesandeks on leida selle funktsiooni pöördfunktsioon $x = f^{-1}(y)$. Ülesande lahendamiseks Monte-Carlo meetodiga on vajalik juhuslike arvude programm (või andur), mis võimaldab genereerida vahemikus (0, 1) ühtlaselt jaotatud juhuslike arvude jada ξ.



JOON. 56

Fikseerime funktsiooni $f(x)$ mingi väärtuse y_0, mis vastavalt funktsiooni monotoonsele kasvamisele või kahanemisele (vt. joon. 56) peab rahuldama võrratusi

$$f(a) < y_0 < f(a+1) \quad \text{või} \quad f(a+1) < y_0 < f(a).$$

Leiame nüüd funktsiooni etteantud väärtusele y_0 vastava argumendi väärtuse, s. t. väärtuse x_0 mille puhul kehtib $f(x_0) = y_0$ ehk $x_0 = f^{-1}(y_0)$.

Silmas pidades, et arvud ξ on vahemikus (0, 1) ühtlaselt jaotatud, on arvud $\xi + a$ vahemikus $(a, a+1)$ samuti ühtlaselt jaotatud. Rakendades valemit (4), saame

$$P\{\xi + a \leqslant x\} = \int_a^x d\xi = x - a, \tag{23}$$

mis kehtib iga $x \in [a, a+1]$ korral, muuhulgas ka x_0 korral. Kui eeldada funktsiooni $f(x)$ monotoonset kasvamist lõigul $[a, a+1]$, on võrratused $\xi + a \leqslant x_0$ ja $f(\xi + a) \leqslant f(x_0)$ samaväärsed. Võrduse (23) põhjal saame siis

$$P\{f(\xi + a) \leqslant f(x_0)\} = x_0 - a. \tag{24}$$

Asendades võrduses (24) avaldise $f(x_0)$ avaldisega y_0, saame

$$x_0 = f^{-1}(y_0) = a + P\{f(\xi + a) \leqslant y_0\}. \tag{25}$$

Kui eeldame funktsiooni monotoonset kahanemist, saame järgmise valemi

$$x_0 = f^{-1}(y_0) = a + P\{f(\xi + a) \geqslant y_0\}. \tag{26}$$

Kui funktsiooniks on konkreetselt $z = x^p$, siis leiame pöördfunktsiooni $x = \sqrt[p]{z}$ väärtuse punktis

$$a^p \leqslant z \leqslant (a+1)^p$$

valemi (25) põhjal järgmiselt

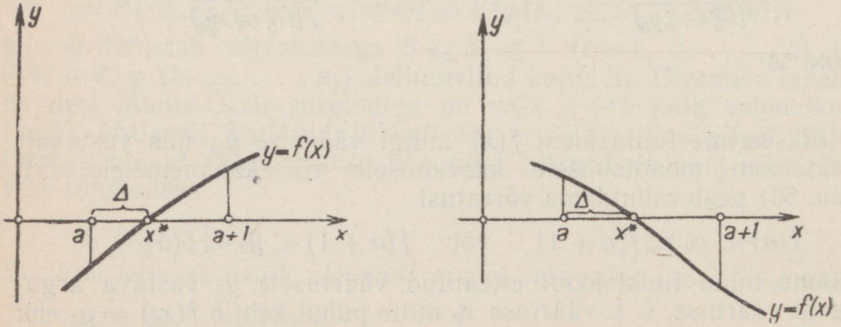
$$\sqrt[p]{z} = a + P\{(\xi + a)^p \leqslant z\}. \tag{27}$$

Valemi (27) rakendamisel on otstarbekohane võtta a täisarv. Sel puhul väljendab a avaldise $\sqrt[p]{z}$ täisosa.

Näide 3. Võrrandi lahendamine. Olgu antud võrrand

$$f(x) = 0, \tag{28}$$

kus $y = f(x)$ on lõigul $[a, a+1]$ pidev funktsioon, mille väärtused lõigu otspunktides on erinevad ja millel on lõigul $[a, a+1]$ ainult üks nullpunkt (vt. joonis 57). Võrrandi (28) lahendamiseks



JOON. 57

Monte-Carlo meetodiga kasutame jälle vahemikus (0, 1) ühtlaselt jaotatud juhuslikke arve ξ. Tähistades võrrandi lahendi x^* abil, saame kirjutada

$$x^* = a + \Delta.$$

Suuruse Δ leiame aga järgmiselt:

$$\Delta = x^* - a = \int_a^{x^*} d\xi = P\{\xi + a \leqslant x^*\}, \tag{29}$$

Vaadeldes juhtu $f(a) < 0$ leiame, et tingimused $\xi + a \leqslant x^*$ ja $f(\xi + a) \leqslant 0$ on samaväärsed, millest saame võrduse (29) põhjal

$$x^* - a = P\{f(\xi + a) \leqslant 0\}$$

ehk

$$x^* = a + P\{f(\xi + a) \leqslant 0\}. \tag{30}$$

Kui funktsiooni $f(x)$ puhul on täidetud tingimus $f(a) > 0$, avaldub lahend x^* kujul

$$x^* = a + P\{f(\xi + a) \geqslant 0\}. \tag{31}$$

Lõpuks tähendame, et kui funktsiooni $f(x)$ kohta lisaks eeldada veel monotoonsust, on võimalik valemid (30) ja (31) tuletada vastavalt valemitest (25) ja (26).

N ä i d e 4. Lineaarse võrrandisüsteemi lahendamine. Olgu antud lineaarne võrrandisüsteem

$$\begin{aligned} &a_{11}x_1 + a_{12}x_2 + \ldots + a_{1k}x_k = b_1, \\ &a_{21}x_1 + a_{22}x_2 + \ldots + a_{2k}x_k = b_2, \\ &\ldots\ldots\ldots\ldots\ldots\ldots \\ &a_{k1}x_1 + a_{k2}x_2 + \ldots + a_{kk}x_k = b_k. \end{aligned} \tag{32}$$

Selle lahendamiseks Monte-Carlo meetodil toimime järgmiselt. Moodustame ruutvormi

$$F(x_1, x_2, \ldots, x_k) = \sum_{i=1}^{k} a_i \Big(\sum_{j=1}^{k} a_{ij}x_j - b_i \Big)^2,$$

kus $a_1, a_2, \ldots a_k$ on suvalised positiivsed suurused. Edasi võtame suvalise positiivse c ja moodustame võrratuse

$$F(x_1, x_2, \ldots, x_n) \leqslant c, \tag{33}$$

mis geomeetriliselt kujutab ellipsoidi k-dimensioonaalses eukleidilises ruumis. On selge, et võrrandisüsteemi (32) lahend on ellipsoidi keskpunktiks ja samuti ka vastupidi. Ellipsoidi keskpunkti leidmisel talitame järgmiselt. Leiame esmalt risttahuka

$$A_j \leqslant x_j \leqslant B_j, \quad (j = 1, 2, \ldots, k),$$

mis sisaldab tervikuna ellipsoidi (33). Seejärel moodustame k juhuslikkude arvude jada ξ_j $(j = 1, 2, \ldots, k)$, kus arvud ξ_j on ühtlaselt jaotatud lõigul $[A_j, B_j]$. Iga k arvu rühma

$$\xi_1^i, \xi_2^i, \ldots, \xi_k^i \quad (i = 1, 2, \ldots, n) \tag{34}$$

puhul kontrollime võrratuse (33) kehtivust. Tingimust (33) rahuldavad rühmad loetleme (olgu nende arv m). Suhe $\nu = \frac{m}{n}$, kus n on kõikide vaadeldavate rühmade arv) annab meile muide ellipsoidi (33) ja vastava risttahuka ruumalade suhte.

Ellipsoidi keskpunkti koordinaate $\bar{x}_j$ $(j = 1, 2, \ldots, k)$ võib leida (resp. ligikaudu leida) mitut moodi. Nii võib tingimust (33) rahuldavates rühmades (34) järjestada arvud ξ_j^ν kasvavas järjekorras $\tilde{\xi}_j^\nu$ ja valida keskmine või kahe keskmise aritmeetiline keskmine

$$\bar{x}_j = \tilde{\xi}_j^{\frac{m+1}{2}} \quad \text{või} \quad \bar{x}_j = \frac{1}{2}\left(\tilde{\xi}_j^{\frac{m}{2}} + \tilde{\xi}_j^{\frac{m}{2}+1}\right).$$

Et see moodus nõuab aga kõikide tingimust (33) rahuldavate rühmade säilitamist kuni keskpunkti koordinaatide arvutamiseni, siis pole see nähtavasti eriti efektiivne. Otstarbekohasem on leida lihtsalt kõikide tingimust (33) rahuldavate rühmade (34) j-ndate koordinaatide aritmeetiline keskmine

$$\bar{x}_j = \frac{1}{m} \sum_{\nu=1}^{m} \xi_j^\nu, \quad (j = 1, 2, \ldots, k).$$

Ellipsoidi keskpunkti koordinaatide leidmiseks on peale esitatud võimaluste veel mitmeid teisi mooduseid, mis baseeruvad juhuslike arvude kasutamisel.

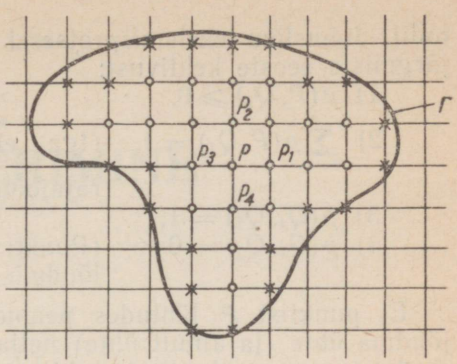
N ä i d e 5. Dirichlet' ülesande lahendamine. Teatavasti sõnastatakse Dirichlet' ülesanne järgmiselt. Olgu antud piirkond D, mille rajajoonel Γ on defineeritud mingi funktsioon $f(Q)$. Tuleb leida funktsioon $u = u(x, y) = u(P)$, mis rahuldab piirkonna D sisepunktides Laplace'i võrrandit

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \tag{35}$$

ja piirkonna rajajoonel ühtib etteantud funktsiooniga $f(Q)$, s. t.

$$u(Q) = f(Q), \quad Q \in \Gamma.$$

Nimetatud ülesande lahendamiseks kasutatakse enamasti diferentsmeetodeid. Piirkond D kaetakse võrguga (vt. joonis 58), kus tähisega «o» märgitakse piirkonna D sisepunktid, tähisega «x»



JOON. 58

rajajoone punktid. Osatuletised avalduvad järgmiste diferentside kaudu

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{1}{h^2}[u(x+h,y)-2u(x,y)+u(x-h,y)],$$
$$\frac{\partial^2 u}{\partial x^2} \approx \frac{1}{h^2}[u(x,y+h)-2u(x,y)+u(x,y-h)], \tag{36}$$

kus h on ühel sirgel asuvate naabersõlmpunktide vaheline kaugus. Asendades avaldised (36) võrrandisse (35), saame selle esitada kujul

$$u(x,y)=\frac{1}{4}[u(x+h,y)+u(x,y+h)+u(x-h,y)+$$
$$+u(x,y-h)]$$

ehk lühemalt

$$u(P)=\frac{1}{4}[u(P_1)+u(P_2)+u(P_3)+u(P_4)]. \tag{37}$$

Lähtudes funktsioonist $f(Q)$ saame leida väärtused

$$f(Q_1),\ f(Q_2),\ \ldots,\ f(Q_s) \tag{38}$$

rajajoone punktides $Q_1, Q_2, \ldots, Q_s$. Seosed (37) ja (38) annavad meile Dirichlet' ülesande uue sõnastuse diferentsmeetodite mõistete abil.

Ülesande lahendamisel kasutame juhusliku ekslemise ideed. Lähtudes mingist sisepunktist P liikugu punkt kuni rajajooneni, kusjuures punkti liikumisel valitakse iga samm stohhastiliselt ja ükski suund (neljast võimalikust) pole eelistatud. Punkti liikumine lõpeb rajapunktini jõudmisel.

Tähistame $p(P, Q_i)$ abil sündmuse tõenäosuse, mis seisneb selles, et punktist P lähtunud juhuslik ekslemine lõpeb rajajoone punktis Q_i. Tõenäolisuse $p(P, Q_i)$ leiame statistiliste katsete mee-

todil. Juhusliku ekslemisprotsessi kirjelduse põhjal võime väita järgmiste seoste kehtivust:

1) $p(P, Q_i) \geqslant 0$,

2) $\sum_{i=1}^{s} p(P, Q_i) = 1$, (Iga ekslemisprotsess lõpeb mingis rajajoone punktis.)

3) $p(Q_i, Q_i) = 1$,

4) $p(Q_i, Q_j) = 0$. (Punkti liikumine lõpeb rajapunktini jõudmisel.)

Et punktist P lähtudes peame järgmisel sammul tingimata jõudma ühte (ja ainult ühte) neljast naaberpunktist P_1, P_2, P_3 ja P_4, kusjuures nendesse punktidesse jõudmise tõenäosused on võrdsed $\frac{1}{4}$, siis peab kehtima järgmine võrdus

$$p(P, Q_i) = \frac{1}{4}[p(P_1\ Q_i) + p(P_2, Q_i) + p(P_3, Q_i) + + p(P_4, Q_i)]. \tag{39}$$

Moodustame nüüd funktsiooni

$$v(P) = \sum_{i=1}^{s} f(Q_i) p(P, Q_i) \tag{40}$$

ja näitame, et see on seoste (37) ja (38) abil sõnastatud Dirichlet' ülesande lahend.

Tõepoolest, funktsioon $v(P)$ rahuldab võrdust (37). Asendades valemis (40) avaldise $p(P, Q_i)$ võrduse (39) põhjal, saame

$$v(p) = \sum_{i=1}^{s} f(Q_i) \frac{1}{4}[p(P_1, Q_i) + p(P_2, Q_i) + p(P_3, Q_i) + + p(P_4, Q_i)] = \frac{1}{4}[\sum_{i=1}^{s} f(Q_i) p(P_1, Q_i) + \sum_{i=1}^{s} f(Q_i) p(P_2, Q_i) + + \sum_{i=1}^{s} f(Q_i) p(P_3, Q_i) + \sum_{i=1}^{s} f(Q_i) p(P_4, Q_i)] =$$

$$= \frac{1}{4}[v(P_1) + v(P_2) + v(P_3) + v(P_4)].$$

Funktsioon $v(P)$ rahuldab ka etteantud rajatingimusi. Asendades punkti P rajajoone mingi punktiga Q_j, saame

$$v(Q_j) = \sum_{i=1}^{s} f(Q_i) p(Q_j, Q_i) = f(Q_j) p(Q_j, Q_j) = f(Q_j),$$

kus arvutasime seoseid 3) ja 4). Sellega oleme näidanud, et valem (40) on tõepoolest Dirichlet' ülesande lahend. Lahendi põhimõtteliselt lihtne konstrueerimine oli võimalik tänu Monte-Carlo meetodi rakendamisele.

VII PEATÜKK

AUTOMAATNE PROGRAMMEERIMINE

§ 25. ALGORITMILISE KEELE MÕISTE

1. Nagu käesoleva õpiku eelmistes peatükkides nägime, võib ülesande lahendamise programmi koostamise käigu jaotada kaheks enam-vähem iseseisvaks etapiks. Esimesel etapil toimub sobivaima lahendusalgoritmi valimine, teisel aga selle *tõlkimine* kasutatava arvuti *käskude keelde.*

Põhilisemaks ning loomingulisemaks nendest kahest on just esimene etapp, mil toimub algoritmide konstrueerimine, võrdlemine ning täpsustamine, sobivaima kuju andmine väljavalitud algoritmile. Suurel määral sõltub selle etapi töö efektiivsus just kasutatava sümboolika otstarbekohasusest: nii algoritmide konstrueerimine, võrdlemine kui ka täpsustamine on tunduvalt lihtsam ning ülevaatlikum, kui seda teha sobivalt valitud *keeles.* Seetõttu kerkibki programmeerimise esimesel etapil teatava spetsiaalse keele kasutamise vajadus algoritmide üleskirjutamiseks.

Kuigi programmeerimise lõppeesmärgiks on arvuti käskude keeles kirjutatud algoritmi saamine, ei osutu selle keele kasutamine esimesel etapil sugugi otstarbekohaseks. Ebaotstarbekohasus avaldub siin eeskätt selles, et käskude keeles kirjutatud algoritm kaotab ülevaatlikkuse, teda on konkreetset arvutit põhjalikult tundmata võimatu lugeda, pranduste või muudatuste sisseviimine algoritmi on väga tülikas, niisugust algoritmi saab kasutada vaid ühe kindla arvuti jaoks. Pealegi tuleb algoritm käskude keele kasutamise korral tarbetult üksikasjalikult (arvuti operatsioonide kaupa) üles kirjutada, mis muudab raskeks võimalikkude algoritmide võrdlemise ning viib sageli selleni, et piirdutakse vaid ühe (enamasti esimese ettejuhtuva) algoritmi vaatlemisega. Kõike seda arvestades, ei kasuta kogenud programmeerija algoritmi koostamise ja täpsustamise käigus kunagi konkreetse arvuti käskude keelt. Veelgi enam, käskude keele kasutamine programmi koostamise esimesel etapil on just saamatu ning oma aega mõttetult raiskava programmeerija kõige iseloomulikumaks tunnuseks.

Matemaatilises kirjanduses kasutatakse algoritmide kirjeldamiseks tavaliselt nn. matemaatilist keelt, mis koosneb vale-

mitest ja nende juurde kuuluvatest sõnalistest seletustest. Niisugust keelt kasutasime ka käesoleva kursuse eelmistes peatükkides. Selle keele eelisteks on suhteline ülevaatlikkus, üldarusaadavus ja universaalsus. Viimane avaldub eeskätt selles, et matemaatilises keeles kirjutatud algoritme saab kasutada mistahes arvutite korral ja neid on lihtne ka trükis avaldada. Matemaatilisel keelel on aga omad puudused. Olulisimaks neist osutub just igasuguse seose puudumine arvuti käskude keelega — see muudab algoritmi tõlkimise käskude keelde üsna töömahukaks ja halvasti automatiseeritavaks protsessiks. Vastupidi üldiselt levinud arvamusele on aga matemaatilise keele puuduseks veel teatav ebatäpsus. Paljusid valemeid, veelgi enam aga sõnalisi seletusi saab sageli mõista mitmeti, sest puuduvad kompaktsete seletuste koostamise ranged reeglid. Väga üksikasjalike ja täpsete seletuste kasutamine aga muudaks algoritmid kohmakateks ning halvasti arusaadavateks.

Teiseks kaunis levinud keeleks, mida programmeerimise esimesel etapil laialdaselt kasutatakse, on nn. b l o k k - s k e e m i d e k e e l. Selle keele eelistest on käesoleva õpiku esimestes peatükkides juba juttu olnud. Üks eelistest — üsna suur vabadus algoritmi jaotamisel blokkideks — on aga ühtlasi selle keele üheks olulisemaks puuduseks. Niisugune vabadus muudab nimelt praktiliselt võimatuks vähegi täpsete reeglite andmise blokk-skeemi tõlkimiseks arvuti käskude keelde. Teatud määral on blokk-skeemide keele puuduseks ka tema sobimatus algoritmide publitseerimiseks. Blokk-skeemide trükkimine on nimelt suhteliselt kallis ja suuremate skeemide korral isegi tehniliselt teostamatu (kui aga skeeme trükitehnika vajadusteks liialt palju tükeldada, siis kaob nende ülevaatlikkus).

2. Eelmises punktis tehtud kriitiliste märkustega on tahetud rõhutada, et programmeerimise esimesel etapil algoritmide üleskirjutamiseks kasutatav keel peab rahuldama tervet rida tingimusi. Lühidalt kokkuvõetuna on need tingimused järgmised:

1) keel peab olema võimalikult üldarusaadav või vähemalt lihtsasti õpitav;

2) keel peab võimaldama algoritme kompaktselt ning ülevaatlikult kirja panna;

3) keel peab sisaldama algoritmide kirjutamise täpsed reeglid, mis muudavad võimatuks igasugused kaksipidimõistmised;

4) keel peab võimaldama küllalt lihtsat ning formaalset tõlkimist mistahes konkreetse arvuti käskude keelde;

5) keel peab olema sobiv algoritmide publitseerimiseks.

Et seninimetatud keeltest igaüks on vastuolus mõnega nendest tingimustest, siis tuli programmeerimise lihtsustamise huvides hakata looma uusi spetsiaalseid keeli, mis rahuldaks kõiki loetletud tingimusi. Niisuguseid keeli nimetatakse tavaliselt a l g o r i t -

milisteks keelteks ja nende loomisele on viimastel aastatel kogu maailmas väga suurt tähelepanu pööratud. Tehtud jõupingutuste tulemusel ongi praegu kasutusel juba mitusada erinevat algoritmilist keelt.

Algoritmiliste keelte suur rakenduslik tähtsus tuleneb eeskätt just sellest, et nad rahuldavad neljandat ülalsõnastatud tingimustest. Selle tingimuse täidetus annab nimelt põhimõttelise võimaluse üle minna nn. automaatsele programmeerimisele, täpsemalt programmeerimise teise etapi automatiseerimisele. See tähendab, et nimetatud tingimust rahuldavas keeles kirjutatud algoritmi saab arvuti käskude keelde tõlkida juba arvutis endas nn. programmeeriva programmi ehk translaatori abil. Automaatse programmeerimise süsteemi moodustavadki algoritmiline keel ja vastav translaator.

Senised kogemused on näidanud, et üleminek automaatsele programmeerimisele võimaldab vähendada programmide koostamiseks ja silumiseks kuluvat inimaega ca 3—5 korda. Eriti märgatavalt väheneb just silumisele kuluv aeg. Tõsi küll, võrreldes vilunud programmeerija poolt vahetult käskude keeles kirjutatud programmidega on translaatori abil saadud programmid sageli vähem ökonoomsed. Statistika näitab nimelt, et automaatse programmeerimise kasutamisel kulub ülesannete lahendamisele keskmiselt kuni 2 korda enam arvuti aega, programmide silumine nõuab aga automaatse programmeerimise korral arvuti aega tavaliselt 2—3 korda vähem.

Arvestades, et peamiseks kitsaskohaks arvutite laialdasel rakendamisel osutub just programmeerijate vähesus, on üleminek automaatsele programmeerimisele siin oluliseks sammuks edasi. Eriti tähtsaks tuleb ühtlasi lugeda veel programmeerijate ettevalmistamise lihtsustumist, sest automaatse programmeerimise süsteemi kuuluva algoritmilise keele õppimine on märgatavalt lihtsam (vt. tingimus 1) kui vahetu programmeerimise õppimine arvuti käskude keeles. Automaatse programmeerimise puhul pole programmeerijal praktiliselt üldse tarvis tunda arvuti käske ja nende kasutamist. Ta ei tarvitse näiteks isegi teada, mitmeaadressilise arvutiga või üldse mis tüüpi arvutiga tal tegemist on, sest samas keeles kirjutatud programmid sobivad igale arvutile, mille standardsete programmide süsteemi on kord lülitatud vastav translaator.

3. Seni loodud algoritmilistest keeltest on eriti laialdase leviku osaliseks saanud keel nimega ALGOL (lühend ingliskeelsetest sõnadest *algorithmic language* — algoritmiline keel). Selle keele esialgne projekt koostati 1958. a. saksa ja ameerika teadlaste ühistööna. 1960. a. valmis keele täpsustatud variant, mis sai ametlikuks nimeks ALGOL-60 (mõningaid täiendavaid täpsustusi tehti selles keeles küll veel 1964. aastal).

ALGOL-60 kui kõige õnnestunum olemasolevatest algoritmilistest keeltest on praegu kogu maailmas üsna laialdaselt kasutusel. Väga paljude arvutitüüpide jaoks on juba koostatud vastavad translaatorid. Uusi väljatöötatavaid algoritme on hakatud publitseerima just selles algoritmilises keeles. NSV Liidus on ta võetud kõikide loodavate automaatse programmeerimise süsteemide aluseks.

Käesolev peatükk ongi pühendatud algoritmilise keele ALGOL-60 kirjeldamisele, lühiduseks kasutame aga edaspidi lihtsalt nimetust ALGOL. Käsitluse aluseks on võetud keele kohta 1964. a. ilmunud täpsustatud kirjeldus.[28] Rõhutame aga kohe, et vastavate translaatorite koostamisega seotud küsimusi me käesolevas õpikus ei puuduta.

Arvestades algoritmilise keele mitmesuguseid kasutamisvõimalusi tehakse vahet ALGOL-i kolme erineva taseme vahel. Nendeks on etalonkeel, publikatsioonide keel ja ALGOLi konkreetsed esitused.

E t a l o n k e e l e kaudu defineeritakse ALGOLi struktuur, ta on kõikide konkreetsete esituste aluseks ning eeskujuks. Mingite konkreetsete arvutite või ülesannete klasside võimalusi ja vajadusi selles keeles ei arvestata. Kõik sümbolid on etalonkeeles valitud vaid arusaadavuse lihtsustamist silmas pidades.

P u b l i k a t s i o o n i d e k e e l on määratud ALGOLis kirjutatud algoritmide trükis avaldamiseks. Seetõttu on siin lubatud etalonkeele mõningane muutmine, mis on tingitud trükitehnilistest vajadustest või parema loetavuse nõudest. Näiteks indeksid, mis etalonkeeles peavad olema paigutatud nurksulgudesse, võib publitseerimisel laduda pool rida allapoole ja ilma sulgudeta (näit. $a[i, k]$ asemel võib trükkida $a_{i,k}$), astendamist võib märgi $\uparrow$ kasutamise asemel laduda tavalisel viisil (näit. $b\uparrow(c+d)$ asemel võib trükkida b^{c+d}), peale ümarsulgude võib publitseerimisel avaldistes kasutada ka kõiki teisi sulge, arvude poollogaritmilise kuju kirjutamisel võib kasutada tavalist kümne astet (näit. $3_{10} - 5$ asemel võib trükkida 3.10^{-5}) jms.

ALGOLi k o n k r e e t n e e s i t u s saadakse etalonkeele kohandamisel kasutatava arvuti võimalustele ning tema sisend- ja väljundseadmetele (kohandamine seisneb näiteks mõningate sümbolite ärajätmises või asendamises teistega, samuti ka üksikute konstruktsioonide kasutamisvõimaluste või isegi tähenduse muutmises). Iga konkreetne esitus peab olema varustatud täpsete eeskirjadega, mis võimaldavad tõlkida etalonkeelest selle esituse keelde.

Käesolevas õpikus tuleb kirjeldamisele eeskätt just ALGOLi etalonkeel. Ühe Eesti NSV-s kasutatava konkreetse esituse juu-

---

[28] On ilmunud ka selle ametliku kirjelduse venekeelne tõlge: vt. [4].

res peatume vaid lühidalt peatüki lõpul, piirdudes sealjuures üksnes selle konkreetse esituse ja etalonkeele vaheliste erinevuste esiletoomisega.

4. Algoritmilise keele kirjeldamine tähendab kõigi nende täpsete reeglite defineerimist, mis muudavad võimatuks igasugused kaksipidimõistmised algoritmide kirjutamisel ning lugemisel. Seetõttu peab keele kirjeldus koosnema loogiliselt järjekindlast definitsioonide jadast. Niisugust definitsioonide jada on aga üsna tülikas jälgida. Seda arvestades on käesolevas peatükis kohati mindud mitte päris järjekindlat teed, mis väljendub selles, et mõningaid mõisteid kasutatakse enne nende täpselt defineerimist. Sellist ebajärjekindlust on püütud kompenseerida veidi ulatuslikumate ja kohati isegi kordavate kirjeldavate selgitustega. Peamiseks esitusviisiks on üldse valitud sõnalised seletused.

Paralleelselt etalonkeele sõnalise kirjeldamisega ning vastavate näidete esitamisega on järgnevas käsitluses aga toodud ka kõigi vaatluselevõetavate konstruktsioonide ranged definitsioonid. Need definitsioonid antakse eriliste metalingvistiliste (s. t. defineeritavast keelest välja ulatuvate) valemite näol, kus peale etalonkeele enda sümbolite kasutatakse omaette sümbolitena veel vastavate konstruktsioonide nimetusi erikujulistes [29] sulgudes ⟨ ⟩ ja seosemärke ::= (tähenduses *on definitsiooni kohaselt*) ning | (tähenduses *või*).

Nii näiteks kirjutis

| ⟨tühi⟩ ::=

tähendab, et t ü h i konstruktsioon on selline, mis ei sisalda ühtki sümbolit. Definitsioon

| ⟨number⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

seevastu teatab, et *numbriks* nimetatakse ükskõik millist sümbolitest 0, 1, . . . , 9.

Oma olemuselt on sellised definitsioonid aga enamasti induktiivsed, s. t. definitsiooni sõnastus võib sisaldada ka defineeritavat mõistet ennast. Selgitame induktiivsete definitsioonide olemust järgmise kahe lihtsa näite abil. Rõhutame ühtlasi juba kohe, et niisugustes definitsioonides ei tohi kasutada ühtki kõrvalist sümbolit, ka mitte kirjavahemärke. Sellepärast ongi kõik kõrvaljoonega märgitud definitsioonid (samuti toodavad näited) kogu järgnevas käsitluses esitatud ilma igasuguste kirjavahemärkideta (kuigi grammatikareeglid nõuaksid näiteks punkti panemist definitsiooni lõppu).

---

[29] Eelmistes peatükkides kasutatud tähenduses meil neid sulgusid enam tarvis ei tule.

Näide 1. Eeldades, et meil on juba teada mõiste *number* tähendus (mis eelmise definitsiooni kohaselt tähendab üht sümbolitest 0, 1, 2, 3, 4, 5, 6, 7, 8 ja 9) võime järgmiselt defineerida mõiste *arv*:

⟨arv⟩ ::= ⟨number⟩ | ⟨number⟩ ⟨arv⟩

See tähendab, et *arv* on kas lihtsalt number või mingi teine *arv*, millele vahetult vasakule on juurde kirjutatud veel üks number. Seega *arvudeks* siin defineeritud mõttes (tavaliselt nimetatakse neid küll märgita täisarvudeks) on igasugused numbrite jadad, näiteks

7
35
000420830116

Näide 2. Definitsioon

⟨vigur⟩ ::= O|△O△|⟨vigur⟩△|⟨vigur⟩ ⟨vigur⟩

tähendab, et *viguriks* nimetatud objekt võib koosneda kas ainult sümbolist O või sümbolite kombinatsioonist △O△ või mingist teisest *vigurist*, millele vahetult paremale on juurde kirjutatud kas sümbol △ või veel üks *vigur*. *Viguriteks* on seega näiteks sümbolite kombinatsioonid

OO△O△
△O△△△OOOO△

Seevastu aga näiteks kombinatsioonid

△△O△O
△OO△

ei ole *vigurid*, sest neid ei saa toodud definitsiooni järgi moodustada.

5. Enne etalonkeele niisuguse range esituse juurde asumist teeme veel mõned üldised sissejuhatavad märkused keele iseloomu ja struktuuri ning kasutatava terminoloogia kohta.

ALGOL kui keel on oma sõnavara ja süntaksi poolest üsna lähedane tavalisele *matemaatilisele keelele*. See tähendab, et suurem osa peamisi konstruktsioone kirjutatakse mõlemas keeles enam-vähem samal kujul ja põhiliselt samu elementaarsümboleid (tavalist trükikirja) kasutades. Olulisemateks erinevusteks on järgmised.

Et ALGOLis kirjutatud algoritm tuleb juba vahetult arvutisse viia, siis arvestades tänapäeva arvutite sisendseadmete ehitust peavad kõik konstruktsioonid olema kirjutatud üksteise järele ühte ritta. See toob paratamatult kaasa mõninga välise erinevuse mitmete tuntud konstruktsioonide, nagu näiteks murdude, indeksite, astendajate jms. kirjutamisel. Siinjuures tuleb muidugi arvestada,

et ALGOLis kirjutatud algoritmide publitseerimisel (aga ka juba kirjutamisel) ei mahu kogu algoritm tavaliselt muidugi ühele reale ära. Seetõttu on üleminek uuele reale paratamatult lubatud, kuid kokkuleppe kohaselt ei lisa see algoritmi mingit täiendavat informatsiooni (s. t. ALGOLis puudub sümbol tähenduses *üleminek uuele reale*). Et niisugusel üleminekul pole sisulist tähendust, siis on ta lubatud algoritmi mistahes kohas, kuid peab toimuma igasuguste lisamärkide kasutamiseta (sõna poolitamisel ei kasutata sidekriipsu, eelmise rea lõpus olevat tehtemärki ei tule järgmise rea alguses korrata jne.). Nimetame veel, et ka tühikul (sõnavahel) pole etalonkeeles mingit sisulist tähendust. Teda võib aga kasutada algoritmi mistahes kohas, kui see vaid lihtsustab algoritmi lugemist.

ALGOLi teiseks silmatorkavamaks erinevuseks on sõnaliste seletuste andmine võimalikult lakoonilisel ja standardsel kujul. Osutub nimelt, et kõigi algoritmides hädasti vajalike sõnaliste seletuste (s. t. selliste, mida tavaliste matemaatiliste sümbolite abil esitada ei õnnestu) andmiseks piisab vaid kahekümne neljast erinevast sõnast. Vastavad ingliskeelsed sõnad ongi lisaks trükimärkidele võetud ALGOLi põhisümbolite hulka ja nende kasutamise täpsed eeskirjad antakse keele kirjelduses.

ALGOLi veel üheks märgatavaks erinevuseks on see, et muutujate tähistamiseks võib peale tähtede (mis traditsioonilises *matemaatilises keeles* moodustab praktiliselt ainsa võimaluse) kasutada ka sõnu, lauseid ja isegi tähtede ning numbrite mistahes kombinatsioone. See näiteks võimaldab raadiust sümboli R asemel tähistada ka «sümboliga» *raadius*.

Algoritmilises keeles ALGOL kirja pandud terviklikku algoritmi nimetame edaspidi A L G O L - p r o g r a m m i k s. Et algoritm võib omakorda sisaldada teisi algoritme (alamalgoritmid), siis osutub vahel otstarbekohaseks tarvitada veel nimetust b l o k k, mis võib seega tähendada kas kogu ALGOL-programmi või selle enam-vähem iseseisvalt kasutatavat osa.

Iga blokk kujutab endast kirjelduste ja operaatorite jada, kusjuures kirjeldused peavad olema koondatud bloki algusesse. K i r j e l d u s e d on määratud vaid selles blokis esinevate muutujate ja teiste objektide iseloomustamiseks ning defineerimiseks — mingisuguseid operatsioone kirjeldused vahetult ei teosta. O p e r a a t o r i d seevastu koosnevad just teostamisele tulevate operatsioonide loeteludest.

Küsimuse sellest, missugused operaatorid algoritmi kuuluma peavad, lahendab see konkreetne matemaatikaharu, kust vastav algoritm pärineb. ALGOLi ülesandeks on vaid anda lihtne ning täpne moodus nende operaatorite ning juurdekuuluvate kirjelduste üleskirjutamiseks. ALGOLi etalonkeele kirjeldamine tähendab just operaatorite ja kirjelduste koostamise ning nende omavahelise ühendamise reeglite täpsustamist.

§ 26. ALGOLi PÕHIELEMENDID

1. Kõik ALGOL-programmid on tegelikult teatavate põhisümbolite jadad ning ALGOLi kirjeldamine tähendabki niisuguste jadade koostamise ja tõlgendamise täpsete eeskirjade andmist. Selleks tuleb kõigepealt muidugi täpsustada põhisümbolite loetelu.

ALGOLi etalonkeeles on kokku 116 põhisümbolit. Oma kirjapildi poolest jagunevad p õ h i s ü m b o l i d kahte suurde klassi: tavalised trükimärgid (tähed, numbrid, kirjavahemärgid jms.) ning ingliskeelsed sõnad. Tavaliste trükimärkide kasutamine põhisümbolitena on tingitud nõudest, et ALGOL sobiks algoritmide publitseerimiseks, samuti arvutite sisendseadmete tavalisest konstruktsioonist. Sümbolite niisugusel valikul on aga ka oma puudus: võivad tekkida raskused ALGOLi põhisümbolite eristamisel m e t a k e e l e sümbolitest (mida publitseerimisel tuleb kasutada täiendavate seletuste jms. esitamiseks). Võimalike segaduste vältimiseks trükitakse metakeele ja ALGOLi sümbolid tavaliselt erinevates kirjades. Käesolevas peatükis näiteks on kõik ALGOLi põhisümbolite tähenduses esinevad tähed ja numbrid trükitud *kursiivis*, metakeeles kasutatavad tähed aga tavalises püstkirjas. Eksituste võimalusi kirjavahemärkide tõlgendamisel on püütud vähendada sel teel, et ALGOLi programmilõikude lülitamisel metakeelsesse (antud juhul eestikeelsesse) teksti jäetakse eesti keele grammatikareegleid rikkudes enamasti ära vajalikud kirjavahemärgid. ALGOLi üksiksümboli esinemisel tekstis on sõnastus püütud valida nii, et eksitust ei saaks tekkida.

Sõnade kasutamine ALGOLi põhisümbolitena on ühelt poolt tingitud sellest, et tavalisi trükimärke on vähem kui algoritmilise keele konstrueerimisel vaja minevaid põhisümboleid. Teiselt poolt saab aga sõnade kasutamisega paremini edasi anda vastava sümboli tähendust. Põhisümbolitena kasutatavad sõnad on oma kirjapildilt muidugi tähtede jadad, kuid neid tähti ei loeta põhisümbolite hulka kuuluvateks. See tähendab, et nende sõnade näol on meil tegemist omaette terviklike sümbolitega, mitte liitsümbolitega. Selle asjaolu rõhutamiseks ongi kõik põhisümbolid-sõnad käesolevas peatükis trükitud **poolpaksus kirjas**, mitte kursiivis. Olgu ühtlasi märgitud, et ALGOLi käsitlevas kirjanduses kasutatakse poolpaksu kirja asemel samas tähenduses sageli ka sõnade allakriipsutamist.

S õ n a l i s i p õ h i s ü m b o l e i d on etalonkeeles kokku 24 (sealjuures üks põhisümbol koosneb tegelikult kahest sõnast). Esitamegi kõigepealt nende sümbolite loetelu, kus igale sümbolile järgneb tema hääldamine (sulgudes) ning tähenduse tõlked eesti ja vene keelde. Venekeelne tõlge on esitatud sellepärast, et mõnedes NSV Liidus kasutatavates ALGOLi konkreetsetes esitustes ongi sõnalisteks põhisümboliteks valitud just need venekeelsed tõlked (või üks toodud tõlgetest).

array [ə′rei] — massiiv; массив
begin [bi′gin] — algus; начало
Boolean [bu:ljen] — loogiline; булевский, логический
comment [′kɔment] — kommentaar; примечание
do [du:] — teostada; выполнить, цикл
else [els] — vastasel korral; иначе
end [end] — lõpp; конец
false [fɔ:ls] — väär; ложь
for [fɔ:] — jaoks; для
go to [gou tu:] — minna; перейти к, на
if [if] — kui; если
integer [′intidʒə] — täisarv; целый
label [′leibl] — märgend; метка
own [oun] — oma; собственный
procedure [prə′si:dʒə] — protseduur; процедура
real [′ri:əl] reaalarv; вещественный
step [step] — samm; шаг
string [striŋ] — rida; строка
switch [swit∫] — lüliti; переключатель
then [ðen] — siis; то
true [tru:] — tõene; истина
until [ʌn′til] — kuni; до
value [′vælju:] — väärtus; значение
while [wail] — seni kui; пока

2. Oma funktsiooni järgi keeles on põhisümboleid otstarbekohane jaotada nelja erinevasse liiki: tähed, numbrid, tõeväärtused ja piirajad. Neist viimaseid võib tähenduse ning kasutamise järgi jaotada veel viide alaliiki: eraldajad, sulud, kirjeldajad, spetsifikaatorid ja tehtemärgid. Lõpuks aga tehtemärgid jagunevad omakorda nelja klassi: aritmeetiliste tehete märgid, loogiliste tehete märgid, võrdlusmärgid ja järgnevusmärgid. Kõik need alajaotused võib kokku võtta järgmisteks definitsioonideks:

⟨põhisümbol⟩ :: = ⟨täht⟩ | ⟨number⟩ |
⟨tõeväärtus⟩ | ⟨piiraja⟩
⟨piiraja⟩ :: = ⟨eraldaja⟩ | ⟨sulg⟩ | ⟨kirjeldaja⟩ |
⟨spetsifikaator⟩ | ⟨tehtemärk⟩
⟨tehtemärk⟩ :: = ⟨aritmeetilise tehte märk⟩ |
⟨loogilise tehte märk⟩ | ⟨võrdlusmärk⟩ | ⟨järgnevusmärk⟩

Igasse alajaotusse kuuluvad üksiksümbolid defineerime lihtsalt vastavate loeteludega. Nende loetelude puhul piirdume vaid lühikeste selgitavate märkustega, üksikasjalikumad selgitused esitame järgmistes paragrahvides.

Etalonkeele tähtedeks on valitud inglise tähestiku 26 suurt ja 26 väikest tähte:

⟨täht⟩ ::= *A* | *B* | *C* | *D* | *E* | *F* | *G* | *H* | *I* | *J* | *K* | *L* | *M* | *N* | *O* | *P* | *Q* | *R* | *S* | *T* | *U* | *V* | *W* | *X* | *Y* | *Z* | *a* | *b* | *c* | *d* | *e* | *f* | *g* | *h* | *i* | *j* | *k* | *l* | *m* | *n* | *o* | *p* | *q* | *r* | *s* | *t* | *u* | *v* | *w* | *x* | *y* | *z*

Tähtedele etalonkeeles mingit omaette tähendust ei omistata, neid kasutatakse vaid mitmesuguste konstruktsioonide (identifikaatorite, ridade) moodustamiseks. ALGOLi konkreetsetes esitustes on tähtede hulk tavaliselt siintoodust erinev, näiteks võivad puududa väikesed tähed, võivad olla lisatud veel kreeka või vene tähestiku tähed jne. Järgnevas käsitluses me eeldame, et tähtedeks on kõik eesti tähestiku suured ja väikesed tähed, s. t. lisame äsjaloetletutele tarbekorral veel tähed *Š, Ž, Õ, Ä, Ö, Ü, š, ž, õ, ä, ö* ja *ü*.

Numbriteks on tavalised kümme araabia numbrimärki. Neid kasutatakse arvude ning muude konstruktsioonide (identifikaatorite ja ridade) moodustamiseks:

⟨number⟩ ::= *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9*

Tõeväärtusi on kaks: **true** ja **false**. Nendel sümbolitel on kindel fikseeritud tähendus ja mingis teises tähenduses neid ei kasutata:

⟨tõeväärtus⟩ ::= **true** | **false**

Eraldajad on kindla tähendusega sümbolid, mida kasutatakse programmi üksikosade või konstruktsioonide eraldamiseks teineteisest. Kõik üksteist eraldajat on toodud järgmises loetelus:

⟨eraldaja⟩ ::= , | . | $_{10}$ | : | ; | := | ␣ | **step** | **until** | **while** | **comment**

Kuigi eraldajad $_{10}$ ja := on oma kirjapildilt liitsümbolid, loetakse neid siiski omaette terviklikeks põhisümboliteks.

Sulgusid on etalonkeeles neli paari (paari moodustavad üks alustav ja üks lõpetav sulg):

⟨sulg⟩ ::= (|) | [|] | ‘ | ’ | **begin** | **end**

Nagu me hiljem näeme, on sulutüübi valik ALGOLis täpselt reglementeeritud — igal sulutüübil on keeles erinev funktsioon.

Kirjeldajad on määratud mitmesuguste kirjelduste lülitamiseks programmi. Etalonkeeles on neid ette nähtud seitse:

⟨kirjeldaja⟩ ::= **Boolean** | **integer** | **real** | **own** | **array** | **procedure** | **switch**

Tegelikult on ka spetsifikaatorid eriliiki kirjeldajad, neid on etalonkeeles kolm:

⟨spetsifikaator⟩ ::= **string** | **value** | **label**

T e h t e m ä r g i d defineeritakse järgmiste loeteludega:

⟨aritmeetilise tehte märk⟩ ::= + | − | × | / | ÷ | ↑
⟨loogilise tehte märk⟩ ::= ¬ | ∧ | ∨ | ⊃ | ≡
⟨võrdlusmärk⟩ :: = < | ≤ | = | ≥ | > | ≠
⟨järgnevusmärk⟩ ::= **go to** | **if** | **then** | **else** | **for** | **do**

Järgnevusmärkide täpset tähendust selgitame hiljem, ülejäänud tehtemärkide tähendus peaks aga olema arusaadav juba kirjapildist. Vastavad tehted on järgmised. Aritmeetilised tehted: liitmine, lahutamine, korrutamine, jagamine, täisarvude jagatise täisosa leidmine ja astendamine. Loogilised tehted: eitus, konjunktsioon, disjunktsioon, implikatsioon ja ekvivalents.[30] Võrdlused: *on väiksem kui, on võrdne või väiksem kui, on võrdne, on võrdne või suurem kui, on suurem kui* ja *ei ole võrdne.*

3. Suuremat osa piirajaist kasutatakse ALGOLis vaid teatavate kindlate konstruktsioonide puhul. Seetõttu ei saa nende täpset tähendust selgitada enne vastavate konstruktsioonide defineerimist. Leidub siiski kaks eraldajat, mille kasutamine on enamvähem iseseisev. Nendeks on ␣ ja **comment**.

Sümboliga ␣ on märgitud tühik (sõnavahe), millel etalonkeeles mingit sisulist tähendust ei ole. Keele üksikkonstruktsioone eraldatakse tühiku abil ainult programmide loetavuse parandamiseks. Translaator jätab tühiku lihtsalt tähele panemata kõikjal, välja arvatud käesoleva paragrahvi lõpul defineeritav konstruktsioon *rida.*

Eraldaja **comment** on määratud selgituste lülitamiseks programmi ja sellele eraldajale järgnev tekst ei mõjusta programmi. K o m m e n t a a r i d e lülitamisel programmi selle eraldaja abil tuleb arvestada järgmist kaht reeglit.

1) Sümbolite jada
; **comment** ⟨mistahes sümbolite jada, mis ei sisalda eraldajat;⟩; on etalonkeeles täpselt sama tähendusega nagu eraldaja ; (semikoolon).

2) Sümbolite jada
begin comment ⟨mistahes sümbolite jada, mis ei sisalda eraldajat ; ⟩ ;
on samaväärne alustava suluga **begin.**

Need reeglid tähendavad, et kommentaare tuleb lülitada programmi vahetult pärast eraldajat ; või pärast sulgu **begin.**

Pärast lõpetavat sulgu **end** võib programmi lülitada kommentaare ka ilma eraldajata **comment**. Täpsemalt, sümbolite jada **end** ⟨ mistahes sümbolite jada, mis ei sisalda eraldajat ; sulgu **end** või järgnevusmärki **else**⟩

---

[30] Juhime tähelepanu sellele, et ekvivalentsi märk ALGOLis (≡) erineb vastavast märgist lausearvutuses (vt. § 8), kus sümbolit ≡ kasutatakse teises tähenduses.

on täiesti samaväärne suluga **end**. Samaväärsus tähendab kõigil nendel kolmel juhul, et translaator tõlgendab vastavaid jadasid üheainsa sümbolina alati, kui nad paiknevad väljaspool konstruktsiooni *rida*.

4. Ülaldefineeritud põhisümbolitest moodustatakse kõik ALGOL-programmide koosseisu kuuluvad elemendid. Põhilisteks nendest elementidest on identifikaatorid, arvud ja read.

Mitmesuguste algoritmis vaja minevate suuruste tähistamiseks kasutatakse ALGOL-programmides identifikaatoreid. Identifikaator on tähtedest ja numbritest moodustatud sõna, mille esimeseks sümboliks on täht. Seega

⟨identifikaator⟩ ::= ⟨täht⟩ | ⟨identifikaator⟩ ⟨täht⟩ |
⟨identifikaator⟩ ⟨number⟩

Näiteks võib identifikaatoritena kasutada järgmisi sümboliteühendeid:

A
kÕÕL1a
Mehe nimi
ümbermõõt
St3a5b
IIDAM991A13

Nagu toodud definitsioonist nähtub, pole etalonkeeles identifikaatorite moodustamisel mingeid kitsendusi. ALGOLi konkreetsetes esitustes on aga tavaliselt kitsendatud identifikaatorite pikkus — see ei tohi olla pikem kui näiteks 6 või 12 sümbolit. Samuti pole konkreetsetes esitustes enamasti lubatud jätta identifikaatorit moodustavate sümbolite vahele tühikuid. Konkreetsetes esitustes on peale selle veel terve rida nn. reserveeritud identifikaatoreid, mida tohib kasutada vaid ühes kindlas tähenduses (etalonkeeles on vaid ette nähtud niisuguste reseveeritud identifikaatorite olemasolu võimalus ja antud mõned soovitused nende valimiseks). Reserveeritud identifikaatorite hulka kuuluvad konkreetsetes esitustes tavaliselt ka kõik sõnalised põhisümbolid. Etalonkeeles on aga kõrvuti näiteks põhisümboliga **label** lubatud kasutada ka identifikaatorit *label*.

Identifikaatorid on määratud lihtmuutujate, massiivide, märgendite, lülitite ja protseduuride tähistamiseks. Tavaliselt valitakse identifikaatoreid nii, et nende kõnekeeleline tähendus või kirjapilt kergendaks vastava ALGOL-programmi lugemist, kuid etalonkeele seisukohalt ei sisalda identifikaatoriks oleva sõna sisuline tähendus mingit informatsiooni. Näiteks identifikaatoritel

alfa beeta gamma

pole translaatori jaoks midagi ühist nurkadega (või omavahel). Samuti ei tekita mingit segadust ortograafilised vead kõnekeelest pärinevate sõnade kasutamisel identifikaatoritena, kui vaid sama identifikaatorit igal pool ühtmoodi kirjutatakse. Muidugi tuleb ka silmas pidada, et sama identifikaatorit ei tohi samas programmis kasutada erinevate suuruste tähistamiseks. Erandiks on siiski juht, kui nende suuruste määramispiirkonnad on ühisosata. Lähemalt täpsustame käesolevas kärpes nimetatud uusi mõisteid aga alles järgmistes paragrahvides.

5. ALGOLi põhielementide teise rühma moodustavad arvud. Arvud jagunevad kahte põhitüüpi: täisarvud ja reaalarvud. Täisarvud kirjutatakse ALGOLis nende tavalisel kujul. Reaalarvude kirjutusviis erineb tavalisest aga selle poolest, et kümnendkoma asemel kirjutatakse punkt ning poollogaritmilise kuju kasutamisel eraldatakse mantiss järgust spetsiaalse põhisümboliga.

Arvude võimalikud kirjutusviisid defineeritakse järgmiselt:

⟨arv⟩ ::= ⟨märgita arv⟩ | + ⟨märgita arv⟩ | − ⟨märgita arv⟩
⟨märgita arv⟩ ::= ⟨kümnendarv⟩ | ⟨järk⟩ | ⟨kümnendarv⟩⟨järk⟩
⟨kümnendarv⟩ ::= ⟨märgita täisarv⟩ | ⟨lihtmurd⟩ |
⟨märgita täisarv⟩⟨lihtmurd⟩
⟨lihtmurd⟩ ::= .⟨märgita täisarv⟩
⟨järk⟩ ::= $_{10}$⟨täisarv⟩
⟨täisarv⟩ ::= ⟨märgita täisarv⟩ | + ⟨märgita täisarv⟩ |
− ⟨märgita täisarv⟩
⟨märgita täisarv⟩ ::= ⟨number⟩ | ⟨märgita täisarv⟩⟨number⟩

Olgu märgitud, et eraldajaid . ja $_{10}$ kasutatakse etalonkeeles ainult äsjadefineeritud tähenduses.

Nagu toodud definitsioonidest nähtub, võib reaalarv koosneda kas ainult mantissist või ainult järgust, mantiss aga võib sisaldada ainult murdosa või olla lihtsalt täisarv. Näiteks on reaalarvudeks kõik järgnevad kirjutised:

0	$-2.311_{10}2$
.3001	$7_{10}-1$
0563	$+.1_{10}1$
+03.03	$-_{10}-3$
−0.056	

Kõiki neid arvusid, mis siin esitatud definitsiooni mõttes pole täisarvud, nimetatakse reaalarvudeks. Toodud näidetest on vaid esimene ja kolmas täisarvud, kõik ülejäänud aga reaalarvud (kuigi näiteks eelviimase väärtus on tegelikult täisarvuline). Neid arvude kahte tüüpi arvestades tuleb meil järgnevas vahet teha näiteks kirjutiste 3 ja 3.0 vahel, kuigi need arvud on suuruselt võrdsed.

Mingi antud arvulise suuruse kirjutis pole etalonkeeles üheselt määratud. Täisarvude juhul avaldub see mitteühesus nullide (ja märgi +) lisamise võimaluses; näiteks kirjutistele +051, 51 ja 00051 omistatakse täpselt sama tähendus. Veelgi suurem vabadus on aga reaalarvude kirjutamisel. Näiteks kirjutised

0.78 .78 +.78 +7.8₁₀ — 1 .078₁₀ + 1
+000.78 780₁₀ — 3 +78.0₁₀ — 2 +.078₁₀1

tähendavad kõik üht ning sama suurust. Tuleb ainult rõhutada, et järguks võib olla vaid täisarv, seega näiteks kirjutis *1.2₁₀2.5* pole üldse arv ja etalonkeeles seda kasutada ei tohi. Samuti pole arv ka näiteks kirjutis *78.₁₀ — 1*, sest lihtmurd ei tohi koosneda ainult punktist.

Arvu moodustavate numbrite jada pikkus pole kitsendatud etalonkeeles, küll aga kõigis konkreetsetes esitustes. Mõnedes konkreetsetes esitustes pole ka lubatud kasutada näiteks reaalarve, mis koosnevad ainult järgust või kümnendarve, millel puudub täisosa.

6. Selleks, et algoritmides saaks tarbekorral opereerida ka põhisümbolite mistahes jadadega on ALGOLi üheks põhielemendiks valitud rida. R i d a on sulgude ‘ ja ’ vahele võetud sümbolite jada, mille üksikasjalik definitsioon on järgmine:

⟨rida⟩ ::= ‘⟨lahtine rida⟩’
⟨lahtine rida⟩ ::= ⟨puhas rida⟩ | ‘⟨lahtine rida⟩’ |
⟨lahtine rida⟩ ⟨lahtine rida⟩
⟨puhas rida⟩ ::= ⟨mistahes sümbolite jada, mis ei sisalda sulgusid ‘ ja ’⟩ | ⟨tühi⟩

Selle definitsiooni kohaselt on ridadeks näiteks järgmised konstruktsioonid:

‘5x = 3y’‘(a — b) korda’ kus ‘x = ␣␣’
‘Siin ␣ on ␣ meil ␣ üks ␣ rida’

Nagu me hiljem näeme, saab ridu kasutada vaid protseduuride tegelike parameetritena. Olgu märgitud, et sulud ‘ ja ’ leiavad kasutamist vaid ridade väljaeraldamiseks.

§ 27. MUUTUJAD JA FUNKTSIOONID

1. Matemaatiliste algoritmide kõige mahukama osa moodustab tavaliselt mitmesuguste avaldiste väärtuste leidmine. Seetõttu on algoritmilise keele kirjeldamisel oluline täpselt fikseerida kõikvõimalike avaldiste moodustamise ja kasutamise reeglid. Kuigi avaldise mõiste tundub intuitiivselt üldarusaadavana, pole seda põhimõistet sugugi nii lihtne rangelt defineerida — me jõuame selleni alles järgmises paragrahvis.

Avaldised koosnevad teatavasti konstantidest, muutujatest ja funktsioonidest, millega teostatakse mitmesuguseid tehteid. Avaldise mõiste rangeks defineerimiseks peamegi kõigepealt täpsustama avaldise koosseisu kuuluda võivate lihtsamate konstruktsioonide mõisted. Mõned nendest lihtsamatest konstruktsioonidest võivad aga üldjuhul omakorda sisaldada koostisosadena avaldisi. Seetõttu peame definitsioonid paratamatult esitama rekursiivsetena.

Avaldiste põhiliste koostisosade — muutujate ja funktsioonide defineerimiseks üldkujul tuleb meil juba käesolevas paragrahvis tarvis avaldise enese mõistet. Kuigi selle mõiste täpsustamine toimub alles järgmises paragrahvis, ei tohiks tema selline ettehaarav kasutamine siiski erilist segadust tekitada. Segadusi aitab loodetavasti vältida see, et oma seniste matemaatiliste kogemuste põhjal on igale lugejale kahtlemata intuitiivselt selge, mida avaldis vähemalt oma lihtsamal kujul tähendab. Seega, käesolevas paragrahvis tuleb terminit *avaldis* esialgu mõista selle sõna kogemuslikus tähenduses.

2. Muutuva suuruse ehk m u u t u j a mõiste on üks matemaatika põhimõistetest. Sealjuures nii tavalises kõnekeeles kui ka matemaatilistes tekstides enamasti samastatakse muutuja tema nimetusega. Näiteks selle asemel, et ütelda *muutuja, mille nimi on x, omandab väärtuse* ... või midagi sarnast, ütleme tavaliselt lihtsalt *x omandab väärtuse* ... . Seetõttu ongi ka ALGOLis muutuja samastatud tema nimetusena kasutatava konstruktsiooniga. Erinevusena matemaatikas välja kujunenud tavadest võib aga ALGOLis muutujate nimetusena kasutada mitte ainult üksikuid tähti, vaid üldse mistahes identifikaatoreid.

Matemaatilistes algoritmides on sageli otstarbekohane tähistada tervet rida muutujaid ühe ning sama sümboliga, mis on varustatud sobivate indeksitega. Niisuguse tähistusviisi olulisust arvestades on see võimalus ette nähtud ka ALGOLis. Vastavalt sellele jagunevadki muutujad kahte suurde klassi — lihtmuutujad ja indeksitega muutujad:

| ⟨muutuja⟩ ::= ⟨lihtmuutuja⟩ | ⟨indeksitega muutuja⟩

L i h t m u u t u j a on ALGOLis samastatud tema nimetusena kasutatava indentifikaatoriga:

| ⟨lihtmuutuja⟩ ::= ⟨identifikaator⟩

See identifikaator on järelikult teatavale väärtusele omistatud nimetus. Seda väärtust võib avaldistes kasutada uute väärtuste moodustamiseks ja tarbekorral võib teda ka muuta (nn. omistamisoperaatoritega, millest tuleb juttu 29. paragrahvis).

I n d e k s i t e g a m u u t u j a nimetus sisaldab peale identifikaatori veel vähemalt ühe indeksi. Ainsa erinevusena tavalisest

kirjaviisist kirjutatakse indeksid aga mitte pool rida põhisümbolist madalamale, vaid vahetult identifikaatori järele nurksulgudesse (olgu ühtlasi märgitud, et nurksulgusid kasutatakse etalonkeeles peale muutuja indeksite väljaeraldamise veel ainult lüliti indeksi puhul, seega võib neid nimetada ka i n d e k s s u l g u d e k s). Kui indekseid on rohkem kui üks, siis eraldatakse nad üksteisest komadega. Samuti nagu tavalises matemaatilises kirjaviisis võib ka siin indeksitena tarbekorral kasutada nii arvusid ja muutujaid (sealhulgas indeksitega muutujaid), kui ka nendest moodustatud aritmeetilisi avaldisi.

Niisugust muutujate kompleksi, millele on omistatud ühine identifikaator ning mis erinevad üksteisest vaid indeksite poolest, nimetatakse m a s s i i v i k s. Vastavalt sellele nimetatakse ka ühte massiivi kuuluvate indeksitega muutujate ühist identifikaatorit selle m a s s i i v i i d e n t i f i k a a t o r i k s.

Indeksitega muutuja definitsiooni võib seega kokku võtta järgmiselt:

⟨indeksitega muutuja⟩ ::= ⟨massiivi identifikaator⟩ [⟨indeksite loetelu⟩]
⟨massiivi identifikaator⟩ ::= ⟨identifikaator⟩
⟨indeksite loetelu⟩ ::= ⟨aritmeetiline avaldis⟩ | ⟨indeksite loetelu⟩, ⟨aritmeetiline avaldis⟩

Indeksitega muutujateks on näiteks järgmised konstruktsioonid:

kordaja [*5, j, j + 7*] *B*[*al + a2, B*[*al, a2*]]

3. Muutujad saavad ALGOL-programmides omandada vaid kolme tüüpi väärtusi. Sealjuures mingi muutuja poolt omandatavad väärtused peavad alati olema üht ning sama tüüpi. Nendeks võimalikeks tüüpideks on: reaalarv, täisarv ja tõeväärtus, mille nimetused etalonkeeles on vastavalt

real integer Boolean

Muutuja poolt omandatavate väärtuste tüüpi nimetatakse ühtlasi selle muutuja tüübiks. ALGOL-programmis tuleb iga muutuja tüüp kindlaks määrata vastava tüübikirjeldusega. Tüübikirjeldus teatab translaatorile, kas vastavat muutujat tuleb tõlgendada (ja talle mälus ruumi eraldada) kui liikuva komaga arvu, fikseeritud komaga arvu või kahendkohta. Tüübikirjelduse lülitamine ALGOL-programmi toimub lihtmuutujate ja indeksitega muutujate korral erinevalt.

Lihtmuutujate t ü ü b i k i r j e l d u s algab kirjeldatava tüübi nimega, millele järgneb sellesse tüüpi kuuluvate lihtmuutujate loetelu. Loetelusse kuuluvad identifikaatorid eraldatakse üksteisest komadega:

⟨tüübikirjeldus⟩ ::= ⟨tüübi nimi⟩ ⟨tüübiloetelu⟩
⟨tüübi nimi⟩ ::= **real** | **integer** | **Boolean**
⟨tüübiloetelu⟩ ::= ⟨lihtmuutuja⟩ | ⟨lihtmuutuja⟩, ⟨tüübiloetelu⟩

Näiteks tüübikirjeldused

real *Ai, xx, nurk, AI, viis*
integer *j, n, kordsus, a1, a2*

teatavad translaatorile, et muutujad

Ai xx nurk AI viis

saavad vastavas ALGOL-programmis omandada vaid reaalarvulisi väärtusi, muutujad

j n kordsus a1 a2

aga ainult täisarvulisi väärtusi.

Tüübikirjeldus mõjub vaid ühe bloki ulatuses (bloki täpse definitsiooni anname 29. paragrahvis, praegu võib selle all mõista lihtsalt teatavat enam-vähem iseseisvat programmiosa). See tähendab, et erinevates blokkides võib sama identifikaatorit kasutada isegi erinevat tüüpi muutujate tähistamiseks. Ühtlasi aga tuleb arvestada, et iga lihtmuutuja tüüp peab olema kirjeldatud selles blokis, kus teda kasutatakse (tavaliselt antakse kõik tüübikirjeldused just bloki alguses). Kui muutuja tüüp on mingis blokis kirjeldatud, siis teistes selle bloki koosseisu kuuluvates väiksemates blokkides teda enam uuesti kirjeldada ei tule. Veelgi enam, kui sisemises blokis siiski esineb uus tüübikirjeldus, siis loetakse vastavat muutujat uueks, selles blokis l o k a l i s e e r i t u d muutujaks, millel välises blokis sama identifikaatoriga tähistatud muutujaga midagi ühist ei ole. Märgime veel, et ühegi bloki alguses ei tohi ühtki muutujat kirjeldada enam kui üks kord.

Peale äsjanimetatud lokaliseerimisvõimaluse saab muutujaid mingis blokis veel n. ö. t ä i e l i k u l t l o k a l i s e e r i d a. Selleks tuleb vastava tüübikirjelduse ette kirjutada täiendavalt kirjeldaja **own**, näiteks

own Boolean *pp, K*

Täielikult lokaliseeritud muutuja säilitab oma viimati omandatud väärtuse kuni järgmise tagasipöördumiseni vastavasse blokki. Vastasel juhul aga pole muutujate väärtused tagasipöördumise korral määratud (mõnedes konkreetsetes esitustes omandavad kõik muutujad kirjeldamisel vastavalt tüübile kas algväärtuse null või **false**, etalonkeeles jääb aga algväärtus määramata).

Muutujate lokaliseerimine on vajalik mälu otstarbekohase kasutamise huvides: bloki täitmise lõppedes pole temas lokaliseeritud muutujate väärtusi enam tarvis säilitada ja mälu vastavat osa saab kasutada teistes blokkides vaja minevate suuruste jaoks.

4. Indeksitega muutujate tüübi kirjeldamine toimub terve massiivi jaoks korraga, seega ühte massiivi kuuluvad indeksitega muutujad on kõik sama tüüpi. Peale massiivi kuuluvate muutujate tüübi tuleb aga veel kirjeldada ka massiiv ise. Sellise nn. massiivikirjelduse ülesandeks on siduda kirjeldatava massiiviga üks kindel identifikaator, määrata indeksite arv (massiivi dimensioon) ja nende rajad ning anda translaatorile korraldus reserveerida arvuti mälus vajalik arv pesi selle massiivi muutujate väärtuste salvestamiseks.

Massiivikirjeldus algab vajaliku tüübi nimega ning kirjeldajaga **array**, millele järgneb vastavat tüüpi massiivide loetelu. Massiivide loetelu koosneb kirjeldatavate massiivide identifikaatoritest, mille järel indekssulgudes antakse indeksite muutumisrajad. Iga muutumisrajade paar koosneb kahest aritmeetilisest avaldisest, mis on omavahel eraldatud kooloniga. Esimene nendest avaldistest määrab vastava indeksi vähima, teine aga suurima lubatava väärtuse. Muutumisrajade paarid on omavahel eraldatud komadega, nende paaride arv määrab massiivi dimensiooni. Kui mitmel kirjeldataval massiivil peavad olema samad indeksite muutumisrajad ning sama dimensioon, siis võib nende muutumisrajade loetelu esitada vaid ühekordselt pärast kõikide vastavate massiivide identifikaatoreid (mis omavahel eraldatakse komadega). Massiivikirjelduse üksikasjalik definitsioon on järgmine:

⟨massiivikirjeldus⟩ ::= **array** ⟨massiivide loetelu⟩ |
⟨tüübi nimi⟩ **array** ⟨massiivide loetelu⟩
⟨massiivide loetelu⟩ ::= ⟨massiivi segment⟩ | ⟨massiivide loetelu⟩, ⟨massiivi segment⟩
⟨massiivi segment⟩ ::= ⟨massiivi identifikaator⟩ [⟨muutumisrajade loetelu⟩] | ⟨massiivi identifikaator⟩, ⟨massiivi segment⟩
⟨muutumisrajade loetelu⟩ ::= ⟨muutumisrajade paar⟩ | ⟨muutumisrajade loetelu⟩, ⟨muutumisrajade paar⟩
⟨muutumisrajade paar⟩ ::= ⟨alumine raja⟩ : ⟨ülemine raja⟩
⟨alumine raja⟩ ::= ⟨aritmeetiline avaldis⟩
⟨ülemine raja⟩ ::= ⟨aritmeetiline avaldis⟩

Näiteks võivad massiivikirjeldused seega olla kujul

array *kordaja* [*1 : 10, —5 : n, 2 : n + 7*]
integer array *B, BB* [*0 : 10, 0 : 20*], *A, number, f1*[*—2 : 2*]

Neist viimane defineerib kaks kahedimensionaalset ning kolm ühedimensionaalset massiivi, millesse kuuluvad indeksitega muutujad on kõik täisarvulised.

Kui massiivikirjelduses puudub tüübi nimi, siis loetakse vastavate massiivide muutujad reaalarvude tüüpi, seega kirjutised **real array** ja **array** on samaväärsed. Olgu märgitud, et mõnedes konkreetsetes esitustes loetakse ka kirjeldamata tüübiga lihtmuutujad automaatselt reaalarvulisteks.

Massiivikirjeldus antakse bloki alguses ning vastavad indeksitega muutujad on selles blokis lokaliseeritud. Tarbekorral võib kasutada ka täielikku lokaliseerimist, lisades massiivikirjelduse ette kirjeldaja **own**.

Muutumisrajade arvulised väärtused leitakse programmis iga kord uuesti siis, kui toimub pöördumine vastava bloki poole. Seda arvestades tohivad muutumisrajade loetelus esineda avaldised sõltuda vaid selles blokis mitte lokaliseeritud muutujatest ja funktsioonidest. Järelikult kõige välimises blokis (kogu programmi alguses) antud massiivikirjeldustest peavad kõik muutumisrajad olema esitatud konstantidena. Kui mõne indeksi puhul alumise muutumisraja väärtus osutub suuremaks ülemise muutumisraja väärtusest, siis vastav massiiv ei ole määratud.

Indeksitega muutujate kasutamisel peab jälgima, et indeksite arv võrduks vastava massiivi kirjelduses antud dimensiooniga ning et ühegi indeksi väärtus ei läheks välja bloki alguses tema jaoks arvutatud rajadest. Tuleb rõhutada, et isegi täielikult lokaliseeritud massiivi korral on indeksitega muutujate väärtus määratud vaid siis, kui indeksid kuuluvad viimasel sellesse blokki sisenemisel arvutatud muutumisrajade vahele.

5. Peale muutujate tuleb avaldiste konstrueerimisel kasutada veel funktsioone. F u n k t s i o o n kujutab endast eeskirjade kogu mingi väärtuse leidmiseks. Niisuguse eeskirjade kogu programmilülitamise, s. t. funktsiooni defineerimise mooduste selgitamise võtame lähemale vaatlusele alles 30. paragrahvis. Siinkohal piirdume vaid juba defineeritud funktsioonide tegeliku kasutamise reeglite täpsustamisega.

Avaldiste koostisosadena tuleb meil kasutada funktsioonide väärtusi argumentide fikseeritud väärtuste korral. Niisuguseid konstruktsioone nimetatakse etalonkeeles väärtustatud funktsioonideks. Iga v ä ä r t u s t a t u d f u n k t s i o o n koosneb vastavat funktsiooni defineeriva protseduuri identifikaatorist (funktsiooni nimest), millele võib ümarsulgudes järgneda tegelike parameetrite (funktsiooni argumentide fikseeritud väärtuste) loetelu. Tegelikeks parameetriteks saavad siinjuures olla kas mingite antud avaldiste väärtused, read või massiivide, lülitite ja protseduuride identifikaatorid. Tegelikud parameetrid eraldatakse üksteisest komadega, etalonkeeles on aga veel lubatud kasutada eraldami-

seks ka sõnu (tähtede ridu) koos kooloni ja sulgudega. Sellised sõnalised eraldajad on mõeldud funktsiooni argumentide sisulise tähenduse selgitamiseks.

Sõltuvalt funktsiooni definitsioonist saab tema väärtuseks olla kas arvuline väärtus või tõeväärtus. Seega iga väärtustatud funktsioon figureerib avaldises mingi arvu või tõeväärtusena.

Väärtustatud funktsiooni täpse definitsiooni võib kokku võtta järgmiselt:

⟨väärtustatud funktsioon⟩ ::= ⟨protseduuri identifikaator⟩ ⟨tegelike parameetrite osa⟩
⟨protseduuri identifikaator⟩ ::= ⟨identifikaator⟩
⟨tegelike parameetrite osa⟩ ::= ⟨tühi⟩ | (⟨tegelike parameetrite loetelu⟩)
⟨tegelike parameetrite loetelu⟩ ::= ⟨tegelik parameeter⟩ | ⟨tegelike parameetrite loetelu⟩ ⟨parameetri eraldaja⟩ ⟨tegelik parameeter⟩
⟨tegelik parameeter⟩ ::= ⟨avaldis⟩ | ⟨rida⟩ | ⟨massiivi identifikaator⟩ | ⟨lüliti identifikaator⟩ | ⟨protseduuri identifikaator⟩
⟨parameetri eraldaja⟩ ::= , |) ⟨tähtede rida⟩ : (
⟨tähtede rida⟩ ::= ⟨täht⟩ | ⟨tähtede rida⟩ ⟨täht⟩

Näiteks väärtustatud funktsiooni

kaugus (Ai, AI) koordinaadid: (viis, xx + nurk)

kasutamine mingis avaldises tähendab, et vastaval kohal tuleb leida nelja muutuja funktsiooni *kaugus* väärtus (niisuguse nimega funktsiooni arvutuseeskiri peab olema antud sama ALGOL-programmi mõnes blokis, millesse koostisosana kuulub vaadeldavat avaldist sisaldav blokk), võttes argumentideks vastavalt lihtmuutujate *Ai, AI* ja *viis* ning avaldise *xx + nurk* väärtused.

6. Etalonkeeles nõutakse, et iga funktsioon peab olema samas programmis defineeritud. Ühtlasi aga on ette nähtud võimalus, et konkreetsetes esitustes saab mõningaid väärtustatud funktsioone kasutada ka ilma neid funktsioone ALGOL-programmis vahetult defineerimata. See tähendab, et vastavad arvutuseeskirjad peab sel juhul moodustama translaator automaatselt. Niisuguste nn. s t a n d a r d s e t e f u n k t s i o o n i d e hulka soovitatakse võtta vähemalt järgmised ühe argumendi funktsioonid:

abs (*A*) — avaldise *A* absoluutväärtuse leidmine,
sqrt (*A*) — ruutjuure leidmine avaldise *A* väärtusest,
sin (*A*) — siinuse leidmine avaldise *A* väärtusest,
cos (*A*) — koosinuse leidmine avaldise *A* väärtusest,
arctan (*A*) — arkustangensi peaväärtuse leidmine avaldise *A* väärtusest,

ln (*A*) — naturaallogaritmi leidmine avaldise *A* väärtusest,

exp (*A*) — avaldise *A* väärtuse eksponendi (s. t. e^A) leidmine,

entier (*A*) — avaldise *A* väärtuse täisosa leidmine (arvu täisosaks on suurim täisarv, mis pole antud arvust suurem),

sign (*A*) — avaldise *A* väärtuse märgi leidmine (vastavalt sellele, kas *A* väärtus on negatiivne, null või positiivne, saadakse *sign* (*A*) väärtuseks kas —1, 0 või +1).

Edaspidi toodavates näidetes kasutamegi neid standardseid funktsioone ilma täiendavate selgitusteta. Sealjuures peame silmas, et funktsioonide argumendid võivad olla nii reaalarvude kui täisarvude tüüpi. Funktsioonide eneste väärtuseks saame alati reaalarvud, välja arvatud viimased kaks funktsiooni, mille väärtused on täisarvud.

Standardsete funktsioonide identifikaatorid on reserveeritud — neid ei tohi kasutada üheski teises tähenduses.

§ 28. AVALDISED

1. Eelmises paragrahvis me vaatlesime avaldiste moodustamiseks vaja minevaid konstruktsioone, nende koostisosi. Sealjuures tuli korduvalt kasutada ka juba terminit *avaldis*, kuid selle mõiste sisu jäi esialgu veel intuitiivse arusaamise pinnale. Loogiliselt järjekindla ning vasturääkivusi mitte võimaldava algoritmilise keele konstrueerimisel sellisest intuitiivsest üldarusaadavusest aga ei piisa: kõigi keeles vajalike mõistete ja konstruktsioonide, sealhulgas ka avaldiste täpne tähendus tuleb defineerida. Käesolev paragrahv ongi pühendatud avaldise mõiste täpsustamisele.

Oma konstruktsiooni, tähenduse ja kasutamise poolest keeles jagunevad avaldised kolme liiki: aritmeetilised avaldised, loogilised avaldised ning nimelised avaldised:

⟨avaldis⟩ ::= ⟨aritmeetiline avaldis⟩ |
⟨loogiline avaldis⟩ | ⟨nimeline avaldis⟩

Nende avaldiseliikide sisuline erinevus väljendub omandatavate väärtuste iseloomus: aritmeetiliste avaldiste väärtused on alati arvud, loogiliste avaldiste väärtusteks saavad olla vaid tõeväärtused ja nimeliste avaldiste väärtusteks konstruktsioonide nimed ehk märgendid.

Et aritmeetilise avaldise rangel defineerimisel tuleb kasutada loogilise avaldise mõistet ja vastupidi, siis tuleks need avaldiseliigid tegelikult defineerida üheaegselt. Definitsioonide liigse

komplitseerituse vältimiseks läheme siiski näiliselt vähem ranget teed: alustame aritmeetilise avaldise defineerimisega, lugedes loogilise avaldise mõiste juba tuntuks, ja anname alles siis loogilise avaldise täpse definitsiooni.

2. Aritmeetiline avaldis kujutab endast mingi arvulise väärtuse leidmise eeskirja. Kõige lihtsamal juhul võib see eeskiri ütelda, et leitavaks väärtuseks on teatav konstant või ühe kindla muutuja väärtus. Neid kõige lihtsamaid avaldisi aritmeetiliste tehete märkide ja ümarsulgudega sobivalt kombineerides saadaksegi keerulisemad avaldised. Põhiliseks erinevuseks tavalisest matemaatilisest kirjutusviisist on siin vaid see, et kõik avaldisse kuuluvad sümbolid peavad olema kirjutatud ühte ritta. Seetõttu aritmeetiliste avaldiste väline kuju tuleb üldiselt erinev samade avaldiste kujust tavalises kirjaviisis. Nii näiteks traditsiooniliselt kujul

$$\frac{a^2+b}{c+b^3} \quad \text{ja} \quad x^{y^2-\sqrt{y}}$$

kirjutatavad avaldised tuleb ALGOLis esitada vastavalt kujul

$$(a \uparrow 2 + b)/(c + b \uparrow 3) \quad \text{ja} \quad x \uparrow (y \uparrow 2 - sqrt(y))$$

Peale aritmeetiliste avaldiste selle sõna kõige tavalisemas tähenduses (lihtavaldised) on ALGOLis lubatud kasutada ka n.-ö. tinglikke avaldisi, mis traditsioonilises kirjaviisis esitatakse mitmerealistena, nagu näiteks võrduse

$$x = \begin{cases} -p + \sqrt{p^2 - 4q}, & \text{kui} \quad p^2 - 4q \geqslant 0, \\ -p - \sqrt{4q - p^2}, & \text{kui} \quad p^2 - 4q < 0. \end{cases}$$

parem pool. Sellise tingliku avaldise koosseisu kuuluvad avaldised võivad omakorda olla tinglikud avaldised jne.

Öeldut täpsustades võib aritmeetilise avaldise definitsiooni võtta kokku järgmiselt:

⟨aritmeetiline avaldis⟩ ::= ⟨aritmeetiline lihtavaldis⟩ | ⟨tingimus⟩ ⟨aritmeetiline lihtavaldis⟩ **else** ⟨aritmeetiline avaldis⟩
⟨aritmeetiline lihtavaldis⟩ ::= ⟨üksliige⟩ | ⟨liitmistüüpi tehtemärk⟩ ⟨üksliige⟩ | ⟨aritmeetiline lihtavaldis⟩ ⟨liitmistüüpi tehtemärk⟩ ⟨üksliige⟩
⟨tingimus⟩ ::= **if** ⟨loogiline avaldis⟩ **then**
⟨üksliige⟩ ::= ⟨tegur⟩ | ⟨üksliige⟩ ⟨korrutamistüüpi tehtemärk⟩ ⟨tegur⟩

⟨liitmistüüpi tehtemärk⟩ ::= + | −
⟨korrutamistüüpi tehtemärk⟩ ::= × | / | ÷
⟨tegur⟩ ::= ⟨primaaravaldis⟩ | ⟨tegur⟩ ↑ ⟨primaaravaldis⟩
⟨primaaravaldis⟩ ::= ⟨märgita reaalarv⟩ | ⟨muutuja⟩ | ⟨väärtustatud funktsioon⟩ | (⟨aritmeetiline avaldis⟩)

Aritmeetilisteks avaldisteks on näiteks järgmised konstruktsioonid:

$31.47_{10} + 5$
$sin(xx + Ai)$
$(viis + abs(AI - nurk)) \uparrow (a1 + kordsus - n)$
$kordaja\ [j\uparrow 2,\ j-2,\ 7] \times f1[I]\uparrow cos(xx)/(arv - 5\uparrow arv)$
$x + 1.0$
if $p\uparrow 2 - 4 \times q \geqslant 0$ **then** $-p + sqrt(p\uparrow 2 - 4 \times q)$ **else** $-p - sqrt(4 \times q - p\uparrow 2)$
if $a \neq 0$ **then** (*if* $a > 0$ **then** a **else** $a\uparrow 2$) **else if** $b = 0$ **then** *1* **else** $b\uparrow 2$

Neist esimesed kaks on primaaravaldised, esimesed kolm tegurid ja esimesed neli üksliikmed. Eelviimase avaldise tähendus on toodud eelmisel leheküljel. Viimasena kirjutatud avaldise väärtus leitakse aga järgmiselt: kui muutuja *a* väärtus erineb nullist, siis positiivse *a* korral on avaldise väärtuseks muutuja *a* väärtus, negatiivse *a* korral aga selle väärtuse ruut; kui *a* väärtus võrdub nulliga, siis muutuja *b* väärtuse võrdumisel nulliga on avaldise väärtuseks 1, nullist erineva *b* väärtuse korral aga selle väärtuse ruut.

3. Juba avaldise definitsioonist järeldub tegelikult ka tehete eelisjärjekord: kõigepealt sooritatakse astendamised, seejärel korrutamistüüpi tehted ja lõpuks liitmistüüpi tehted. Ühe tüübi piirides sooritatakse tehted aga selles järjekorras nagu nad avaldises on kirjutatud (vasakult paremale). Tehete niisuguse järjekorra muutmiseks tuleb kasutada ümarsulgusid: alustava ning vastava lõpetava sulu vahel paiknevad tehted sooritatakse eraldi ja tulemust kasutatakse edasistes tehetes ühe arvuna.

Eelisjärjekorra mõiste selgitamiseks toome veel võrdluseks mõned avaldised, esitatuna nii ALGOLis kui ka tavalises kirjutusviisis:

$a[n] \times b/c[m,i]$	tähendab	$\frac{a_n \cdot b}{c_{mi}}$
$a \times b/c \times e/f$	tähendab	$\frac{a \cdot b \cdot e}{c \cdot f}$
$a \uparrow s \uparrow t$	tähendab	$(a^s)^t$
$a \uparrow (s \uparrow t)$	tähendab	a^{s^t}

Peale teatava arvulise väärtuse leidmise eeskirja sisaldab aritmeetiline avaldis tegelikult ka informatsiooni, mis igal konkreetsel juhul määrab üheselt leitava arvulise väärtuse tüübi ehk avaldise tüübi.

Iga muutuja tüüp on määratud vastava tüübikirjeldusega, konstandi tüüp on määratud tema kirjapildiga ja nagu me hiljem näeme, on ka iga funktsiooni väärtuste tüüp määratud tema definitsioonis. Seega ühtib ainult ühest konstandist, muutujast või väärtustatud funktsioonist koosneva avaldise tüüp vastavalt selle konstandi, muutuja või väärtustatud funktsiooni tüübiga.

Keerulisema avaldise tüübi saamiseks tuleb seal esinevaid tehteid nende sooritamise järjekorras läbi vaadates leida iga tehte tulemuse tüüp. Viimasena sooritamisele kuuluva tehte tulemuse tüüp ongi vaadeldava avaldise tüüp. Konstruktsioone, mille väärtustega teostatakse mingit tehet, nimetame selle tehte operandideks. Aritmeetilise tehte tulemuse tüübi määramiseks operandide tüüpide järgi tuleb kasutada järgmisi reegleid.

Tehete $+$, $-$ ja $\times$ korral on tulemus täisarvude tüüpi vaid siis, kui mõlemad operandid on seda tüüpi. Kõigil teistel juhtudel on tulemus reaalarvude tüüpi. Tehte / tulemus on operandide igasuguste tüüpide korral ikka reaalarvude tüüpi.

Tehe $\div$ on määratud vaid täisarvuliste operandide korral, kusjuures ka tulemus on täisarvuline. Tulemuseks on nimelt jagatise täisosa, mis määratakse järgmiselt:

$a \div b$ tähendab sama mis $sign(a/b) \times entier\ (abs(a/b))$

Näiteks kui a ja b väärtusteks on vastavalt 9 ja -2, siis $a \div b$ väärtuseks saame -4.

Astendamise tehte ⟨tegur⟩ ↑ ⟨primaaravaldis⟩ tulemuse tüübi määramisel tuleb eraldi arvestada järgmisi võimalusi.

Täisarvude tüüpi kuuluva astendaja p korral on astendamise $t \uparrow p$ tulemuse astendaja mittenegatiivsete väärtuste puhul sama tüüpi nagu t (välja arvatud juht, kui t ja p väärtused võrduvad mõlemad nulliga — astendamise tulemus pole siis üldse määratud), astendaja negatiivsete väärtuste puhul aga alati reaalarvude tüüpi (välja arvatud juht, kui t väärtus võrdub nulliga — siis pole tulemus määratud).

Reaalarvude tüüpi kuuluva astendaja p korral pole $t \uparrow p$ üldse määratud siis, kui t väärtus võrdub nulliga ja p väärtus on mittepositiivne, või kui t väärtus on negatiivne. Määratud juhul on tulemus t mittenegatiivsete väärtuste korral reaalarvuline, kusjuures $t \uparrow p$ on samaväärne kirjutisega $exp(p \times ln(t))$.

Aritmeetiliste avaldiste tegelike väärtuste leidmisel tuleb muidugi arvestada veel seda, et tehted reaalarvudega toimuvad üldiselt ligikaudselt, teatava etteantud lõpliku täpsusega. Tehete täpsuse küsimust aga etalonkeeles lähemalt ei fikseerita — see tuleb teha igas konkreetses esituses. Seega ühe ning sama aritmeetilise avaldise väärtused leitakse erinevates konkreetsetes esi-

tustes üldiselt erinevalt. Selline erinevate väärtuste omandamise võimalus leiab aga aset vaid reaalarvude puhul. Nii etalonkeeles kui ka konkreetsetes esitustes peavad kõik täisarvude tüüpi kuuluvat tulemust andvad tehted toimuma täpselt.

4. Kõrvuti aritmeetiliste tehetega tuleb algoritmides sageli sooritada ka loogilisi operatsioone, leida tõeväärtusi. Loogiline avaldis ongi mingi tõeväärtuse leidmise eeskiri. Kõige lihtsamal juhul võib selline eeskiri ütelda, et otsitavaks tõeväärtuseks on teatava (loogilise) muutuja väärtus, väärtustatud funktsioon või konstant (konstante on vaid kaks: **true** ja **false**).

Loogiliste avaldiste ühe lihtsaima eriliigi moodustavad veel võrdlused. Võrdlus koosneb kahest aritmeetilisest avaldisest, mis on ühendatud võrdlusmärgiga (s. t. ühega sümbolitest $<$, $\leqslant$, $=$, $\geqslant$, $>$ või $\neq$). Võrdlus omandab väärtuse **true** siis, kui vaadeldavate aritmeetiliste avaldiste väärtuste vahel leiab aset võrdlusmärgiga nõutud suhe, vastasel juhul on võrdluse väärtuseks **false**. Näiteks võrdlus

$$x = y$$

omandab väärtuse **true** vaid siis, kui muutujate x ja y väärtused on (selle võrdluse vaatlemise momendil) võrdsed.

Keerulisemad loogilised avaldised saadaksegi niisugustest lihtsamatest avaldistest loogiliste tehete märkidega ühendamise teel, kusjuures tarbekorral kasutatakse avaldise üksikosade väljaeraldamiseks ümarsulge. Samuti nagu aritmeetiliste avaldiste korral võib ka nüüd lisaks lihtavaldistele moodustada veel tinglikke avaldisi. Seega võime loogilise avaldise definitsiooni võtta kokku järgmiselt:

⟨loogiline avaldis⟩ ::= ⟨loogiline lihtavaldis⟩ | ⟨tingimus⟩ ⟨loogiline lihtavaldis⟩ **else** ⟨loogiline avaldis⟩
⟨loogiline lihtavaldis⟩ ::= ⟨implikatsioon⟩ | ⟨loogiline lihtavaldis⟩ ≡ ⟨implikatsioon⟩
⟨implikatsioon⟩ ::= ⟨loogiline liige⟩ | ⟨implikatsioon⟩ ⊃ ⟨loogiline liige⟩
⟨loogiline liige⟩ ::= ⟨loogiline tegur⟩ | ⟨loogiline liige⟩ ∨ ⟨loogiline tegur⟩
⟨loogiline tegur⟩ ::= ⟨loogiline sekundaaravaldis⟩ | ⟨loogiline tegur⟩ ∧ ⟨loogiline sekundaaravaldis⟩
⟨loogiline sekundaaravaldis⟩ ::= ⟨loogiline primaaravaldis⟩ | ¬ ⟨loogiline primaaravaldis⟩
⟨loogiline primaaravaldis⟩ ::= ⟨tõeväärtus⟩ | ⟨muutuja⟩ | ⟨väärtustatud funktsioon⟩ | ⟨võrdlus⟩ | (⟨loogiline avaldis⟩)
⟨võrdlus⟩ ::= ⟨aritmeetiline lihtavaldis⟩ ⟨võrdlusmärk⟩ ⟨aritmeetiline lihtavaldis⟩

Loogilisteks avaldisteks on näiteks järgmised konstruktsioonid:

$pp \vee B[a1,\ a2] > A[5] + j \supset K \wedge a1 = a2 \uparrow 2$
$p \uparrow 2 - 4 \times q \geqslant 0 \equiv \neg$ (**if** *pp* **then** $x1 \neq x2$ **else** $x1 > 0$)
if if if *a* **then** *b* **else** *c* **then** *d* **else** *e* **then** *f* **else** *g*

Loogiliste primaaravaldistena kasutatavad muutujad ja väärtustatud funktsioonid peavad kuuluma tõeväärtuste tüüpi (tüübi nimi **Boolean**), vastasel juhul kaotab avaldis mõtte. Näiteks viimastes konstruktsioonides peavad tõeväärtuste tüüpi olema muutujad *pp*, *K*, *a*, *b*, *c*, *d*, *e*, *f* ja *g*.

Toodud definitsioonist järeldub juba tehete sooritamise järjekord loogilises avaldises. Analoogiliselt aritmeetiliste avaldiste juhuga tuleb ka siin ümarsulgude vahel paiknevad tehted eraldi sooritada ning tulemust kasutada edasiste tehete sooritamisel ühe tõeväärtusena. Sulgude puudumisel sooritatakse tehteid üldiselt nende esinemise järjekorras vasakult paremale, kusjuures esimeses järjekorras leitakse aritmeetiliste avaldiste väärtused ja teostatakse siis võrdlemistehted (s. t. leitakse võrdluste tõeväärtused). Edasi tulevad teostamisele loogilised tehted järgmises eelisjärjekorras: kõigepealt toimub eitamine ($\neg$), siis konjunktsioon ($\wedge$), siis disjunktsioon ($\vee$), siis implikatsioon ($\supset$) ning kõige viimases järjekorras ekvivalents ($\equiv$). Sellise eelisjärjekorra muutmiseks tuleb kasutada sulgusid, näiteks

$$\neg(a \equiv b) \vee (c \supset d \wedge e)$$

Arvestades tõeväärtuste uut tähistusviisi meenutame, et vaadeldavate loogiliste tehete tulemused leitakse tabeli 35 kohaselt (kus *A* ja *B* tähendavad näiteks loogilisi primaaravaldisi):

Tabel 35

A	true	true	false	false
B	true	false	true	false
$\neg A$	false	false	true	true
$A \wedge B$	true	false	false	false
$A \vee B$	true	true	true	false
$A \supset B$	true	false	true	true
$A \equiv B$	true	false	false	true

5. Avaldiste kolmanda liigi moodustavad nimelised avaldised, mille väärtuseks saavad olla vaid märgendid. Märgend on operaatori nimena kasutatav identifikaator või märgita täisarv, operaatori märgendamine on aga vajalik teatava koha fikseerimiseks ALGOL-programmis, et sinna tarbekorral juhtimist saaks anda (suunamine).

Nimelises avaldises võib märgend olla antud ka erikujulise ühe argumendi funktsiooni lüliti abil. Selle funktsiooni argumendiks saavad olla vaid naturaalarvud, väärtusteks aga märgendid. Väärtuseks oleva märgendi leidmise eeskiri argumendi väärtuse järgi peab vastava lülitikirjelduse näol olema defineeritud bloki alguses. Lüliti kirjeldamise mooduseid selgitame aga alles järgmises paragrahvis, siin võtame lihtsalt teatavaks, et nimelistes avaldistes võib kasutada nn. väärtustatud lüliteid, milles argument on fikseeritud teatava aritmeetilise avaldise väärtusena.

Neid mõisteid kasutades võib nimelise avaldise definitsiooni esitada järgmiselt:

⟨nimeline avaldis⟩ ::= ⟨nimeline lihtavaldis⟩ | ⟨tingimus⟩ ⟨nimeline lihtavaldis⟩ **else** ⟨nimeline avaldis⟩
⟨nimeline lihtavaldis⟩ ::= ⟨märgend⟩ | ⟨väärtustatud lüliti⟩ | (⟨nimeline avaldis⟩)
⟨märgend⟩ ::= ⟨identifikaator⟩ | ⟨märgita täisarv⟩
⟨väärtustatud lüliti⟩ ::= ⟨lüliti identifikaator⟩ [⟨aritmeetiline avaldis⟩]
⟨lüliti identifikaator⟩ ::= ⟨identifikaator⟩

Nimelisteks avaldisteks on näiteks järgmised konstruktsioonid:

95
Operaator3
pöörduda [**if** $x \neq 0$ **then** $s + 1$ **else** *1*]
if $p \uparrow 2 - 4 \times q \geqslant 0$ **then** *reaal* **else** *imaginaar*

Naturaalarvude kasutamisel märgendina tuleb arvestada, et arvu alguses seisvaid nulle ei arvestata. Seega näiteks märgendit *00051* loetakse etalonkeeles samaväärseks märgendiga *51*. Erandiks selles reeglis on siiski märgend *0*, mis ei tähenda märgendi puudumist.

Väärtustatud lüliti argumendiks oleva avaldise väärtus ümardatakse alati täisarvuks, see tähendab, et näiteks kirjutised

pöörduda [$a + b$] ja *pöörduda* [*entier* $(a + b + 0.5)$]

on täiesti samaväärsed (täpselt sama märkus käib ka indeksitega muutuja puhul kasutatavate indeksavaldiste kohta). Väärtustatud lüliti puhul tuleb veel arvestada, et tema väärtus on määratud vaid siis, kui argumendiks osutub naturaalarv, mis ei ole suurem vastavas lülitikirjelduses oleva märgendite loetelu liikmete arvust (vastavad mõisted täpsustame järgmises paragrahvis).

Mistahes liiki avaldise väärtuse leidmine on üldiselt rekursiivne protsess, sest avaldis võib omakorda sisaldada teisi avaldisi (ümarsulgudes) ja funktsioone, mis jälle sisaldavad avaldisi ning funktsioone jne. Sellega seoses kerkivate probleemide veidi üksikasjalikuma analüüsi esitame aga alles 30. paragrahvis.

§ 29. OPERAATORID

1. Operaaatoriks nimetatakse terviklikku konstruktsiooniühikut, mis sooritab mingeid algoritmis ette nähtud operatsioone. Sellise konstruktsiooniühiku lõppemise tunnuseks on etalonkeeles alati kas sümbol; (semikoolon) või **end.** Operaatorite täitmine (s. t. vastavate operatsioonide sooritamine) toimub üldiselt selles järjekorras, nagu nad ALGOL-programmi on kirjutatud, kuid üheks algoritmides teostatavaks operatsiooniks võib olla ka operaatorite täitmise järjekorra muutmine.

Selleks et operaatorite täitmise loomuliku järjekorra muutmine osutuks tehniliselt teostatavaks, tuleb mõningatele operaatoritele tarbekorral anda nimi. Niisugune nime andmine ehk märgendamine toimub sel teel, et operaatori ette kirjutatakse märgend koos sellele järgneva kooloniga. Märgendada võib sealjuures ka juba märgendatud operaatorit, s. t. ühel operaatoril võib olla mitu erinevat märgendit. Tuleb aga jälgida, et sama bloki erinevatel operaatoritel ei oleks ühesuguseid märgendeid. Erinevaid operaatoriliike on kokku kaheksa. Nelja neist nimetatakse põhioperaatoreiks, sest nad ei saa oma koosseisus sisaldada teisi operaatoreid. Sellisteks põhioperaatoriteks on tühioperaator, omistamisoperaator, protseduurioperaator ja suunamisoperaator. Keerulisema struktuuriga operaatoriliikideks on tingimuslik operaator, tsüklioperaator, liitoperaator ja blokk.

Viimased kaks operaatoriliiki on tegelikult lihtsalt operaatorite jadad, mis on nn. operaatorsulgude **begin** ja **end** abil ühendatud üheks tervikuks. Erinevus liitoperaatori ja bloki vahel seisneb vaid selles, et viimane peab peale operaatorite sisaldama ka kirjeldusi (s. t. kas tüübi-, massiivi-, lüliti- või protseduurikirjeldusi). Meenutame, et bloki mõiste on vajalik just identifikaatorite lokaliseerimiseks: kõik need objektid, mille identifikaatorid on kirjeldatud bloki alguses, on selles blokis lokaliseeritud ja neid ei saa väljaspool blokki samas tähenduses kasutada. Rõhutame, et bloki koosseisu kuuluvate operaatorite märgendamiseks kasutatavad märgendid on automaatselt (ilma mingi kirjelduseta) lokaliseeritud kõige väiksemamahulisemas blokis, milles see märgendamine toimus.

Muidugi võib iga liitoperaatori või bloki koosseisu kuuluv operaator ise olla kas liitoperaator või blokk. Olgu veel märgitud, et ALGOL-programm tervikuna peab olema ühendatud blokiks või vähemalt liitoperaatoriks.

Operaatori, liitoperaatori ja bloki ning nende juurde kuuluvate mõistete täpsed definitsioonid võib esitada järgmiselt:

⟨ALGOL-programm⟩ ::= ⟨blokk⟩ | ⟨liitoperaator⟩
⟨blokk⟩ ::= ⟨märgendamata blokk⟩ | ⟨märgend⟩ : ⟨blokk⟩
⟨märgendamata blokk⟩ ::= ⟨bloki algus⟩ ; ⟨liitoperaatori lõpp⟩

⟨bloki algus⟩ ::= **begin** ⟨kirjeldus⟩ | ⟨bloki algus⟩ ; ⟨kirjeldus⟩
⟨kirjeldus⟩ ::= ⟨tüübikirjeldus⟩ | ⟨massiivikirjeldus⟩ | ⟨lülitikirjeldus⟩ | ⟨protseduurikirjeldus⟩
⟨liitoperaaator⟩ ::= ⟨märgendamata liitoperaator⟩ | ⟨märgend⟩ : ⟨liitoperaator⟩
⟨märgendamata liitoperaator⟩ ::= **begin** ⟨liitoperaatori lõpp⟩
⟨liitoperaatori lõpp⟩ ::= ⟨operaator⟩ **end** | ⟨operaator⟩ ; ⟨liitoperaatori lõpp⟩
⟨operaator⟩ ::= ⟨tingimusteta operaator⟩ | ⟨tingimuslik operaator⟩ | ⟨tsüklioperaator⟩
⟨tingimusteta operaator⟩ ::= ⟨põhioperaator⟩ | ⟨liitoperaator⟩ | ⟨blokk⟩
⟨põhioperaator⟩ ::= ⟨märgendamata põhioperaator⟩ | ⟨märgend⟩ : ⟨põhioperaator⟩
⟨märgendamata põhioperaator⟩ ::= ⟨tühi operaator⟩ | ⟨omistamisoperaator⟩ | ⟨protseduurioperaator⟩ | ⟨suunamisoperaator⟩

Enne vastavate näidete esitamist tuleb meil täpsustada veel mõningad selles definitsioonis kasutatud mõisted: kuut liiki operaatorite ja kaht liiki kirjelduste tähendused on ju esialgu defineerimata. Asumegi nüüd nende definitsioonide juurde, lükates vaid protseduurikirjelduse lähema vaatlemise edasi järgmise paragrahvini.

2. T ü h i o p e r a a t o r on selline operaator, mis ei sisalda ühtki sümbolit ega soorita järelikult mingit operatsiooni:

⟨tühi operaator⟩ ::= ⟨tühi⟩

Tühja operaatori mõiste on vajalik selleks, et tarbekorral saaks märgendada ka n. ö. tühja kohta. Niisuguse märgendamise otstarbekohasus selgub ülejärgmises paragrahvis toodavatest näidetest.

3. O m i s t a m i s o p e r a a t o r on määratud muutujate väärtuste muutmiseks, s. t. teatava aritmeetilise või loogilise avaldise väärtuse omistamiseks kas ühele või üheaegselt mitmele muutujale. Peale selle võib toimuda ka väärtuse omistamine protseduuri identifikaatorile. Kuigi selle võimaluse lähema käsitlemise juurde asume alles järgmises paragrahvis, anname omistamisoperaatori definitsiooni siiski kohe üldkujul:

⟨omistamisoperaator⟩ ::= ⟨vasakute poolte loetelu⟩ ⟨aritmeetiline avaldis⟩ | ⟨vasakute poolte loetelu⟩ ⟨loogiline avaldis⟩
⟨vasakute poolte loetelu⟩ ::= ⟨vasak pool⟩ | ⟨vasakute poolte loetelu⟩ ⟨vasak pool⟩
⟨vasak pool⟩ ::= ⟨muutuja⟩ := | ⟨protseduuri identifikaator⟩ :=

Näiteks omistamisoperaatori

$AI := kordaja\ [3,\ j,\ A[-2]] :=$ **if** $n = a1$ **then** $xx \uparrow n$ **else** $xx \uparrow 3$

täitmisel tuleb arvutada avaldise

if $n = a1$ **then** $xx \uparrow n$ **else** $xx \uparrow 3$

väärtus ja anda see väärtus muutujale *AI* ning *kordaja* [*3, j, A*[—*2*]].

Paneme tähele, et mõni omistamisoperaatori vasakute poolte loetelus olevatest muutujatest võib sisalduda ka o m i s t a m i s e s ü m b o l i s t := paremal paiknevas avaldises. Sellist operaatorit tõlgendatakse nii, et avaldise väärtuse arvutamisel tuleb kasutada vastava muutuja senist väärtust ja alles lõpuks omistada talle saadud uus väärtus. Üsna tüüpiliseks omistamisoperaatoriks on näiteks

$$n := n + 1$$

mis tähendab, et muutuja *n* väärtust tuleb suurendada ühe võrra. Tuleb aga rõhutada, et omistamisoperaatori vasakus pooles olevate indeksitega muutujate indeksite väärtused arvutatakse (järjekorras vasakult paremale) enne paremas pooles paikneva avaldise juurde asumist. Seega näiteks omistamisoperaatoris

$$BB[j,\ j \uparrow 2 + 3] := j := kordsus \times j - j / a1$$

tuleb indeksite väärtused arvutada muutuja *j* vana, mitte uue väärtuse abil.

Omistamisoperaatorite lülitamisel programmi peab arvestama veel seda, et omistamise sümbolist paremal paikneva avaldise väärtust peab saama arvutada, s. t. kõik selle avaldise koosseisu kuuluvad muutujad peavad (mõne varasema omistamisoperaatori toimel) juba olema omandanud väärtused.

Et muutujatele saab anda vaid nende tüübiga kooskõlas olevaid väärtusi, siis tuleb omistamisoperaatorite moodustamisel silmas pidada nii avaldise kui ka uusi väärtusi omandavate muutujate tüüpe. Etalonkeeles nõutakse, et kõigile omistamisoperaatori vasakute poolte loetelusse kuuluvatele muutujatele ja protseduuride identifikaatoritele oleks vastavate kirjeldustega antud ühesugune tüüp. Kui selleks tüübiks on tõeväärtus, siis peab omistamise sümbolist paremal paiknema loogiline avaldis; kui aga uusi väärtusi omandavad muutujad on kõik kas reaalarvude või täisarvude tüüpi, siis peab omistamise sümbolist paremal olema aritmeetiline avaldis.

Kui aritmeetilise avaldise tüüp erineb vasakute poolte loetelus olevate muutujate ühisest tüübist, siis toimub omistamisel automaatselt veel saadud väärtuse tüübi muutmine. Täisarvu teisendamine reaalarvuks ei muuda selle arvu väärtust. Vastupidine tei-

sendamine aga tähendab ümardamist lähimaks täisarvuks. Näiteks omistamisoperaator

$$n := Ai + xx/viis$$

kus n on täisarvuline muutuja, aga $Ai + xx/viis$ reaalarvuliste väärtustega avaldis, annab täitmisel täpselt sama tulemuse nagu omistamisoperaator

$$n := entier(Ai + xx/viis + 0.5)$$

4. Protseduurioperaatori täitmine tähendab vastava standardse alamprogrammi — protseduuri ühekordset täitmist ALGOL-programmi sellel kohal, kus paikneb protseduurioperaator. Oma välise kuju ja seega ka süntaktilise definitsiooni poolest langeb see operaator täielikult kokku väärtustatud funktsiooniga. Sisuline erinevus nende vahel seisneb selles, et väärtustatud funktsiooni saab kasutada üksnes niisuguse protseduuri korral, mille täitmise lõpptulemustest meid huvitab vaid üksainus arv (või tõeväärtus). Lähemalt käsitleme protseduurioperaatori kasutamisega seotud küsimusi aga alles järgmises paragrahvis.

5. Suunamisoperaator katkestab operaatorite täitmise nende loomulikus järjekorras (s. t. selles järjekorras, nagu nad ALGOL-programmi kirjutatud on) ja suunab täitmise (annab juhtimise) sellisele operaatorile, millel on teatud kindel märgend. Selle märgendi määrab suunamisoperaatori koosseisu kuuluv nimeline avaldis. Suunamisoperaatori üldkuju on järgmine:

⟨suunamisoperaator⟩ ::= **go to** ⟨nimeline avaldis⟩

Et märgendid on alati lokaliseeritud, siis pole suunamisoperaatoriga võimalik väljastpoolt blokki anda juhtimist bloki sees olevale märgendatud operaatorile (küll aga saab selliselt suunata liitoperaatori koosseisu kuuluvale märgendatud operaatorile). Seda asjaolu tuleb ALGOL-programmide koostamisel pidevalt silmas pidada: mingi programmiosa võib omaette blokiks ühendada vaid siis, kui selle programmiosa täitmine peab alati algama tema esimesest operaatorist. Etalonkeeles pole lubatud niisuguse märgendi kasutamine suunamisoperaatoris, millega märgendatud operaatorit vastavas blokis ei leidu.

Suunamisoperaatori koosseisu kuuluvaks nimeliseks avaldiseks võib olla väärtustatud lüliti. Sel juhul peab vastava bloki alguses paiknema sama nimega lüliti kirjeldus. Lülitikirjeldus algab kirjeldajaga **switch**, millele järgneb lüliti identifikaator ja pärast omistamise sümbolit veel lülituste loetelu:

⟨lülitikirjeldus⟩ ::= **switch** ⟨lüliti identifikaator⟩ := ⟨lülituste loetelu⟩

⟨lülituste loetelu⟩ ::= ⟨nimeline avaldis⟩ | ⟨lülituste loetelu⟩, ⟨nimeline avaldis⟩

Lülituste loetelu määrab vastava väärtustatud lüliti väärtuse selle argumendiks oleva avaldise väärtuse järgi: väärtustatud lüliti väärtuseks on selle nimelise avaldise väärtus, mis paikneb lülituste loetelus nii mitmendal kohal, kui suur on argumendi (naturaalarvuline) väärtus. Kui argumendi väärtus on suurem avaldiste arvust vastavas lülituste loetelus (või mittepositiivne), siis niisugust väärtustatud lülitit oma nimelise avaldise kohal sisaldav suunamisoperaator on samaväärne tühja operaatoriga, s. t. ei muuda operaatorite täitmise järjekorda. Näiteks kui bloki alguses paikneb lülitikirjeldus

switch *üleminek* := *Op2, algus, Op1,*
if $x > 0$ **then** *Op3* **else** *välja, Op4*

siis suunamisoperaator

go to *üleminek* [**if** $n < 4$ **then** n **else** $n - 3$]

ei teosta (täisarvulise) muutuja n mittepositiivsete või kaheksast suuremate väärtuste korral mingit operatsiooni. Kui aga n väärtuseks on näiteks 5 või 2, siis annab ta juhtimise operaatorile, mille märgendiks on *algus*. Kui n väärtuseks on 7, siis sõltub suunamine veel muutuja x väärtusest: positiivse väärtuse korral saab juhtimise operaator märgendiga *Op3*, muutuja x mittepositiivse väärtuse korral aga märgendiga *välja* varustatud operaator.

6. Ka tingimuslik operaator on määratud operaatorite täitmise järjekorra juhtimiseks, täpsemalt, ühe (või mitme) operaatori vahelejätmiseks. Selle operaatori võimalikud erikujud defineeritakse järgmiselt:

⟨tingimuslik operaator⟩ ::= ⟨kui-operaator⟩ | ⟨kui-operaator⟩ **else** ⟨operaator⟩ | ⟨tingimus⟩⟨tsüklioperaator⟩ | ⟨märgend⟩ : ⟨tingimuslik operaator⟩
⟨kui-operaator⟩ ::= ⟨tingimus⟩⟨tingimusteta operaator⟩

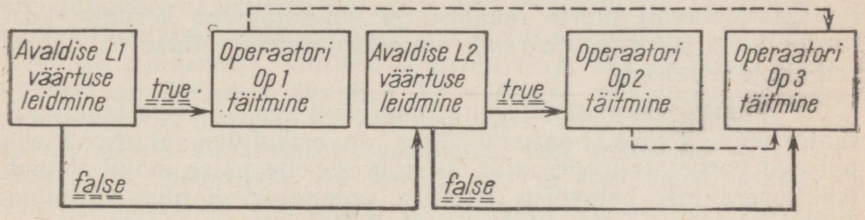
Kui-operaatori täitmisel toimub kõigepealt vastavas tingimuses oleva loogilise avaldise tõeväärtuse leidmine. Kui selleks väärtuseks osutub **true**, siis täidetakse kui-operaatori koosseisu kuuluv tingimusteta operaator, väärtuse **false** korral aga jäetakse see operaator vahele ning pöördutakse vahetult järgmisena kirjutatud operaatori juurde.

Keerulisema struktuuriga tingimuslikud operaatorid võivad olla kahte erinevat liiki. Esimese liigi korral kas täidetakse üks tingimusliku operaatori koosseisu kuuluvatest operaatoritest või asutakse kohe vahetult pärast tingimuslikku operaatorit kirjutatud operaatori täitmisele. Teise liigi korral täidetakse igal juhul üks tingimusliku operaatori koosseisu kuuluvatest operaatoritest, tingimuse täidetuse kontrollimisega tehakse vaid kindlaks, milline

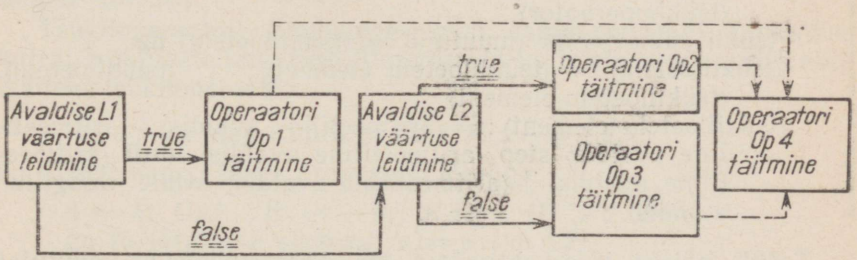
operaator tuleb just täita. Nende kahe liigi esindajateks on näiteks järgmistes operaatoripaarides olevad tingimuslikud operaatorid:

if *L1* **then** *Op1* **else if** *L2* **then** *Op2*; *Op3*
if *L1* **then** *Op1* **else if** *L2* **then** *Op2* **else** *Op3*; *Op4*

Metakeelsete sümbolitega L1 ja L2 on siin tähistatud loogilisi avaldisi, sümbolitega Op1, Op2, ... aga operaatoreid. Nende kahe operaatoritejada täitmise käiku on illustreeritud vastavalt joonistel 59 ja 60 toodud skeemidega. Punktiirjoontega on nendel skee-



JOON. 59



JOON. 60

midel tähistatud üleminekud juhul, kui eelmised operaatorid ei ole suunamisoperaatorid. Ühtlasi selgub siit järgnevusmärgi **else** tähendus: kui juhtimine jõuab selle märgini, siis suunab ta juhtimise edasi operaatorile, mis paikneb kogu tingimusliku operaatori järel. Järgnevusmärk **then** annab aga sõltuvalt tema ees paikneva avaldise väärtusest juhtimise kas vahetult tema enda taga või sellele järgneva **else** taga paiknevale operaatorile.

Tingimusliku operaatori kasutamise näitena esitame programmilõigu, mis leiab taandatud ruutvõrrandi

$$x^2 + 2px + q = 0$$

lahendid. Eeldame, et võrrandi kordajad p ja q on juba leitud ning omistatud vastavalt muutujate P ja Q väärtusteks. Samuti eeldame, et vaadeldava bloki alguses on peale P ja Q kirjelda-

tud veel reaalarvulised muutujad *D*, *X1* ning *X2* ja loogiline muutuja *reaal*. Siis programmilõik

```
D := P ↑ 2 − Q; if D ≥ 0 then
begin D := sqrt (D); X1 := −P + D;
X2 := −P − D; reaal := true end
else begin X1 := −P; X2 := sqrt(−D);
reaal := false end
```

arvutab reaalsete juurte korral need juured muutujate *X1* ja *X2* väärtustena ning annab selle juhu tunnusena muutujale *reaal* väärtuse **true**. Kompleksete juurte korral saame *X1* ja *X2* väärtusteks vastavalt juurte reaalosa ja imaginaarosa kordaja, kusjuures loogiline muutuja *reaal* omandab väärtuse **false** (s. t. juured ei ole reaalsed).

7. Viimase operaatoriliigina tuleb meil veel tutvuda tsüklioperaatoritega. Tsüklioperaator on erikujulise algusosa ehk päisega varustatud operaator, kusjuures selle päise mõjul toimub tsüklioperaatori koosseisu kuuluva operaatori korduv täitmine. Tsüklioperaatori võimalikud kujud defineeritakse järgmiselt:

⟨tsüklioperaator⟩ ::= ⟨tsükli päis⟩ ⟨operaator⟩ | ⟨märgend⟩ : ⟨tsüklioperaator⟩
⟨tsükli päis⟩ ::= **for** ⟨muutuja⟩ := ⟨tsükliloetelu⟩ **do**
⟨tsükliloetelu⟩ ::= ⟨tsükliloetelu element⟩ | ⟨tsükliloetelu⟩, ⟨tsükliloetelu element⟩
⟨tsükliloetelu element⟩ ::= ⟨aritmeetiline avaldis⟩ | ⟨aritmeetiline avaldis⟩ **step** ⟨aritmeetiline avaldis⟩ **until** ⟨aritmeetiline avaldis⟩ | ⟨aritmeetiline avaldis⟩ **while** ⟨loogiline avaldis⟩

Tsükli päises järgnevusmärgi **for** järel paiknevat muutujat nimetatakse tsükliparameetriks. Tsüklioperaatori täitmisel omistatakse tsükliparameetrile järjest uusi väärtusi ja täidetakse iga kord piiraja **do** järel paiknev operaator. Pärast igakordset uue väärtuse omistamist kontrollitakse, kas tsükliparameetrile omistatavate väärtuste varu oli juba lõppenud. Eitava vastuse korral täidetakse nimetatud operaator ja kõik kordub uuesti, jaatava vastuse korral aga antakse juhtimine kohe tsüklioperaatori järel paiknevale operaatorile.

Tsükliparameetrile omistatavate väärtuste leidmise eeskirjad on määratud tsükliloetelu elementidega nende paiknemise järjekorras (vasakult paremale). Vastavalt elementide kolmele liigile saame järgmised kolm eeskirja tsükliparameetrite väärtuste määramiseks.

Kui tsükliloetelu elemendiks on aritmeetiline avaldis, siis määrab vastav element vaid tsükliparameetri üheainsa väärtuse, milleks on selle avaldise väärtus (vahetult enne piiraja **do** järel

paikneva operaatori järjekordset täitmist). Üksnes seda tüüpi elementidest koosneva loeteluga tsüklioperaatoriks on näiteks

for $p := 1.6,\ 2.1.\ 37_{10}1,\ 34_{10} - 1$ **do** *punkt* [$p,\ 2 \times p$] := p;

mis omistab indeksitega muutujatele *punkt* [2, 3], *punkt* [2, 4], *punkt* [4, 7] ja *punkt* [3, 7] vastavalt väärtused *1.6, 2.1, 3.7* ja *3.4.*

Tsükliloetelu elementi kujul

A1 **step** *A2* **until** *A3*

(kus *A1*, *A12* ja *A3* on aritmeetilised avaldised) nimetatakse samm-elemendiks. Sel juhul esimeseks tsükliparameetrile omistatavaks väärtuseks on avaldise *A1* väärtus ja igal järgmisel korral liidetakse tsükliparameetri väärtusele avaldise *A2* väärtus (liitmise momendil). Ühtlasi kontrollitakse, kas tsükliparameetri väärtus on ületanud avaldise *A3* väärtuse, kusjuures ületamise suuna määrab avaldise *A2* väärtuse märk: kui *A2* väärtus on positiivne, siis tsükliparameetri väärtus ei tohi saada *A3* väärtusest suuremaks, negatiivse *A2* korral aga ei tohi tsükliparameetri väärtus saada *A3* väärtusest väiksemaks. Kui avaldise *A2* väärtus on tsükliparameetri muutmise momendil null, siis loetakse vaadeldav element ammendatuks (samuti nagu *A3* väärtuse ületamise korral).

Tsüklioperaatori funktsioneerimist samm-elemendi kuulumise korral tsükliloetelusse võib illustreerida ka teiste operaatorite abil. Näiteks tsüklioperaator

for $n := k$ **step** l **until** m **do** $s := s + n \uparrow 2$;

on täiesti samaväärne programmilõiguga

$n := k$; *Op1* : **if** $(n - m) \times sign\ (l) > 0$ **then**
go to *järgmine element*; $s := s + n \uparrow 2$;
$n := n + l$; **go to** *Op1*;

kus märgendiga *järgmine element* on märgendatud tsüklioperaatorile järgnev operaator (või üldjuhul tsükliloetelu järgmisele elemendile vastava programmilõigu esimene operaator).

Samm-elemendiga tsüklioperaatoreid kasutatakse eriti sageli just tehete puhul indeksitega muutujatega. Näiteks vektorite skalaarkorrutise võib arvutada järgmiselt:

korrutis := *0.0*; **for** $i := 1$ **step** *1* **until** N **do**
korrutis := *korrutis* + $A[i] \times B[i]$;

kus muutuja N väärtuseks on massiividega A ja B esitatud vektorite ühine dimensioon. Maatriksid korrutab aga programm

for $i := 1$ **step** *1* **until** m **do for** $j := 1$ **step** *1* **until** n **do**
begin *element* := *0.0*; **for** $k := 1$ **step** *1* **until** p **do**
element := *element* + $a[i,\ k] \times b[k,\ j]$;
$c[i,\ j]$:= *element* **end**

Samuti võib sellise elemendiga tsüklioperaatorit kasutada polünoomi

$$y = c_0x^n + c_1x^{n-1} + \ldots + c_{n-1}x + c_n$$

väärtuste arvutamiseks (Horneri skeemi järgi):

$y := c[0]$; **for** $i := 1$ **step** 1 **until** n **do** $y := y \times x + c[i]$;

Tsükliloetelu elementide kolmanda liigi moodustavad nn. s e n i - k u i - e l e m e n d i d, mis kirjutatakse kujul

A **while** L

kus A on mingi aritmeetiline avaldis ja L loogiline avaldis. Sellise elemendi puhul omistatakse tsüklioperaatorile iga kord avaldise A väärtus ja arvutatakse ühtlasi avaldise L väärtus. Kui selleks osutub **true**, siis täidetakse piiraja **do** järel paiknev operaator, kui L väärtuseks saadakse aga **false**, siis minnakse üle loetelu järgmise elemendi (või selle puudumisel järgmise operaatori) juurde. Näiteks tsüklioperaatori

for $n := k$ **while** $n \leqslant m$ **do begin** $A[n] := n \uparrow 2$; $k := k + 1$ **end**

võib teisi operaatoriliike kasutades kirjutada kas kujul

Opl: $n := k$; **if** $\neg\, n \leqslant m$ **then go to** *järgmine element*;
$A[n] := n \uparrow 2$; $k := k + 1$; **go to** *Opl*;

või ka kujul

for $n := k$ **step** 1 **until** m **do** $A[n] := n \uparrow 2$;

Seni-kui-elemendiga tsüklioperaatoreid kasutatakse sageli just mitmesuguste iteratiivsete algoritmide puhul. Näiteks ruutjuure leidmise algoritmi

$$\sqrt{A} = \lim_{n\to\infty} z_n; \text{ kus } z_{n+1} = \frac{1}{2}\left(z_n + \frac{A}{z_n}\right), \; z_0 = 1$$

võib programmeerida kujul

$z := 1.0$; **for** $y := .5 \times (z + A/z)$ **while**
$abs(y - z) \geqslant epsilon$ **do** $z := y$;

kus muutuja z väärtuseks saadakse ruutjuur muutuja A väärtusest täpsusega, mille määrab muutuja *epsilon* väärtus.

Selle näitega seoses tuleb veel rõhutada, et tsüklioperaatori täitmise lõppemisel ei ole tsükliparameetri väärtus määratud, seega muutuja y väärtuseks ei tarvitse olla otsitav ruutjuur. Kui aga väljumine tsüklioperaatorist toimub mõne tema koosseisu kuuluva suunamisoperaatoriga, siis säilitab tsükliparameeter oma viimati omandatud väärtuse. Märgime ühtlasi, et väljastpoolt tsük-

lioperaatorit ei saa suunata piiraja **do** järel paikneva operaatori koosseisus olevatele märgendatud operaatoritele.

Ulatuslikumaid näiteid kõigi vaadeldud operaatoriliikide kasutamise ja üldse ALGOL-programmide koostamise kohta on toodud ülejärgmises paragrahvis.

§ 30. PROTSEDUURID

1. Praktikas esineb väga sageli niisuguseid algoritme, mille erinevates kohtades tuleb rakendada üht ning sama arvutuseeskirja või valemit. Seda tüüpi algoritmi programmeerimisel on kirjutiste lühendamiseks loomulik esitada korduvalt kasutatav arvutuseeskiri üheainsa programmilõiguna, mille poole programmi teistest kohtadest võib tarbe korral pöörduda. Niisuguseid enam-vähem iseseisvaid programmilõike, mille poole mujalt programmist saab ajutiselt pöörduda ja seejärel jätkata arvutusi katkenud kohast, nimetatakse tavaliselt alamprogrammideks, ALGOLi etalonkeeles aga p r o t s e d u u r i d e k s.

Mingi arvutuseeskirja esitamine iseseisva protseduurina võib osutuda otstarbekohaseks isegi siis, kui seda vaadeldavas algoritmis kasutatakse vaid ühekordselt. Nimelt saab ulatuslikuma ülesande lahendamise peaaegu alati taandada terve rea lihtsamate ülesannete lahendamisele. Üsna sageli on nende lihtsamate ülesannete lahendusalgoritmid sealjuures aga juba varem programmeeritud ja neid säilitatakse vastava arvutuskeskuse programmoteegis. Järelikult on töö kokkuhoiu huvides tähtis, et sobivate valmis programmilõikude kui protseduuride sisselülitamine koostatavasse ALGOL-programmi oleks võimalikult lihtne. Lihtsa süsteemi olemasolu protseduuride lülitamiseks programmi on oluline veel sellelt seisukohalt, et ulatuslikuma ülesande lahendamiseks vajaliku programmi koostamisel saaks kasutada mitut programmeerijat — erinevaid protseduure võivad koostada erinevad isikud.

2. Selleks et mingit programmiosa saaks teatavas blokis kasutada kui protseduuri, peab ta vastava bloki alguses olema defineeritud protseduurina. Protseduuri defineerimine toimub erilise p r o t s e d u u r i k i r j e l d u s e g a, mis peale kirjeldaja **procedure** sisaldab veel protseduuri päise ja protseduuri sisu. P r o t s e d u u r i p ä i s määrab defineeritava protseduuri nime ja loetleb, milliseid protseduuris esinevaid identifikaatoreid tõlgendatakse formaalsete parameetritena (tarbekorral täpsustades veel eriliste spetsifikatsioonidega nende sisulist tähendust). P r o t s e d u u r i s i s u k s on vastavas alamprogrammis ette nähtud tehteid teostav operaator (tavaliselt blokk või liitoperaator). Tuleb rõhutada, et protseduuri sisuks olevat operaatorit tõlgen-

datakse alati blokina — ka siis kui ta on vormistatud näiteks liitoperaatorina või isegi põhioperaatorina. See tähendab, et protseduuri sisu koosseisu kuuluvate operaatorite märgendid on alati lokaliseeritud.

Protseduurikirjelduse üksikasjalik definitsioon on järgmine:

⟨protseduurikirjeldus⟩ ::= **procedure** ⟨protseduuri päis⟩ ⟨protseduuri sisu⟩ | ⟨tüübi nimi⟩ **procedure** ⟨protseduuri päis⟩ ⟨protseduuri sisu⟩
⟨protseduuri sisu⟩ ::= ⟨operaator⟩ | ⟨kood⟩
⟨protseduuri päis⟩ ::= ⟨protseduuri identifikaator⟩ ⟨formaalsete parameetrite osa⟩; ⟨väärtuste osa⟩ ⟨spetsifikatsioonide osa⟩
⟨formaalsete parameetrite osa⟩ ::= ⟨tühi⟩ | (⟨formaalsete parameetrite loetelu⟩)
⟨formaalsete parameetrite loetelu⟩ ::= ⟨formaalne parameeter⟩ | ⟨formaalsete parameetrite loetelu⟩ ⟨parameetri eraldaja⟩ ⟨formaalne parameeter⟩
⟨formaalne parameeter⟩ ::= ⟨identifikaator⟩
⟨väärtuste osa⟩ ::= ⟨tühi⟩ | **value** ⟨identifikaatorite loetelu⟩;
⟨identifikaatorite loetelu⟩ ::= ⟨identifikaator⟩ | ⟨identifikaatorite loetelu⟩, ⟨identifikaator⟩
⟨spetsifikatsioonide osa⟩ ::= ⟨tühi⟩ | ⟨spetsifikatsioon⟩ ⟨identifikaatorite loetelu⟩; | ⟨spetsifikatsioonide osa⟩ ⟨spetsifikatsioon⟩ ⟨identifikaatorite loetelu⟩;
⟨spetsifikatsioon⟩ ::= **string** | ⟨tüübi nimi⟩ | **array** | ⟨tüübi nimi⟩ **array** | **label** | **switch** | **procedure** | ⟨tüübi nimi⟩ **procedure**

Terminiga k o o d on selles definitsioonis nimetatud konstruktsiooni, mille koostamisel kasutatakse etalonkeelest välja ulatuvaid vahendeid, näiteks konkreetse arvuti käske. Koodi koostamise ja kasutamise reeglid tuleb täpsustada igas konkreetses esituses — etalonkeeles piirdutakse vaid vastava võimaluse reserveerimisega.

3. F o r m a a l s e d p a r a m e e t r i d on identifikaatorid, mille vahendusel protseduur tema täitmise ajaks seostatakse põhiprogrammiga. Seostamise teostab vastava nimega väärtustatud funktsioon või protseduurioperaator. Et protseduurioperaator on oma struktuurilt ja väliselt kujult ühesugune väärtustatud funktsiooniga:

⟨protseduurioperaator⟩ ::= ⟨protseduuri identifikaator⟩ ⟨tegelike parameetrite osa⟩

siis võime piirduda vaid näiteks protseduurioperaatori abil teostatava seostamise vaatlemisega.

Protseduuri seostamine põhiprogrammiga seisneb selles, et protseduuri poole pöördumise momendil kõigile protseduuri sisus esinevatele formaalsetele parameetritele kas omistatakse vastavate tegelike parameetrite väärtus või asendadakse nende identifikaatorid tegelikeks parameetriteks olevate konstruktsioonidega. Vastavalt nendele kahele võimalusele saamegi kaks sisuliselt erinevat seostamisviisi: seostamine väärtuse järgi ja seostamine nime järgi.

Etalonkeeles nõutakse, et kõik formaalsed parameetrid peavad olema juba protseduurikirjelduses nende kahe seostamisviisi järgi kahte klassi jaotatud. Nimelt tuleb vahetult pärast formaalsete parameetrite osa protseduuri päises väärtuste osa näol anda nende formaalsete parameetrite loetelu, mille seostamine peab alati toimuma väärtuste järgi (ülejäänud formaalsed parameetrid kuuluvad siis juba automaatselt nime järgi seostatavate parameetrite klassi). Näiteks kui protseduurikirjeldus algab järgmiselt:

procedure *alamprogramm1 (üks, kaks, kolm, neli, viis, viis viis, kaksüks, akolm, kolmas, välja);*
value *kaks, neli, viis, viisviis, akolm, kolmas;*

siis see tähendab, et formaalsed parameetrid

kaks, neli, viis, viisviis, akolm ja *kolmas*

tuleb selle protseduuri igakordsel kasutamisel asendada vastavateks tegelikeks parameetriteks olevate avaldiste väärtustega. Formaalsete parameetrite

üks, kolm, kaksüks ja *välja*

asemele tuleb aga protseduuri sisus panna vastavateks tegelikeks parameetriteks olevad konstruktsioonid nii, nagu nad protseduurioperaatoris on kirjutatud.

Kui protseduuri sisu on etalonkeeles kirjutatud operaator, siis protseduurioperaatori täitmine algabki sellega, et väärtuse järgi seostatavad parameetrid loetakse protseduuri sisuks olevas operaatoris (kui blokis) lokaliseeritud muutujateks ja omistatakse neile vastavate tegelike parameetritega määratud algväärtused. Seda võib tõlgendada nii, et moodustatakse protseduuri sisu hõlmav blokk, mille algusesse kirjutatakse vastavate muutujate kirjeldused ning nendele algväärtuste andmiseks vajalikud omistamisoperaatorid. Seejärel asendatakse nime järgi seostatavad parameetrid kõikjal, kus nad protseduuri sisus esinevad, vastavateks tegelikeks parameetriteks olevate konstruktsioonidega (võttes need konstruktsioonid tarbekorral sulgudesse), paigutatakse niimoodi teisendatud protseduuri sisu protseduurioperaatori asemele ja täidetakse.

4. Protseduurioperaatorite kasutamisel tuleb silmas pidada, et tegelike parameetrite arv võrduks vastavas protseduurikirjelduses antud formaalsete parameetrite arvuga. Samuti peab aga jälgima, et tegelike parameetritena kasutatavad konstruktsioonid oleks kooskõlas formaalsete parameetrite tõlgendamisega protseduuri sisus. Asi on nimelt selles, et kuigi kõik formaalsed parameetrid on protseduurikirjelduses esitatud lihtsalt identifikaatoritena, võib nende sisuline tähendus seejuures olla hoopis erinev. Formaalseks parameetriks olev identifikaator võib nimelt tähendada kas rida, muutujat, massiivi, märgendit, lülitit või protseduuri. Protseduuri päises paiknev spetsifikatsioonide osa ongi määratud formaalsete parameetrite sisulise tähenduse täpsustamiseks: iga spetsifikatsioon teatab, millist konstruktsiooniühikut tohib selle formaalse parameetri kohal tegeliku parameetrina kasutada. Näiteks kui mingis protseduurikirjelduses protseduuri päis on antud kujul

procedure *alamprogramm2 (a, b, c, d, e, f, g, h, i);*
value *a, d, e;* **real** *a, c;* **integer array** *b, g;*
integer *d;* **Boolean** *e;* **procedure** *f;* **label** *h, i;*

siis see tähendab, et vastavates protseduurioperaatorites tohib esimese ja kolmanda tegeliku parameetrina kasutada vaid reaalarvude tüüpi aritmeetilisi avaldisi, teise ja seitsmenda parameetrina vaid täisarvuliste massiivide identifikaatoreid, neljanda parameetrina vaid täisarvude tüüpi aritmeetilisi avaldisi, viienda parameetrina loogilisi avaldisi, kuuenda parameetrina mõne protseduuri identifikaatorit ning kaheksanda ja üheksanda parameetrina nimelisi avaldisi.

Iga formaalse parameetri kohta tohib anda ülimalt ühe spetsifikatsiooni. Sealjuures väärtuse järgi seostatavad parameetrid peavad tingimata esinema ka spetsifikatsioonides, nime järgi seostavate parameetrite sisulise tähenduse võib aga jätta spetsifitseerimata. Viimasel juhul tuleb vastavad tegelikud parameetrid siiski alati valida nii, et nende asendamisel protseduuri sisusse kõik konstruktsioonid jääksid süntaktiliselt lubatavateks. Seda tuleb aga mõnikord silmas pidada ka siis, kui parameetrid on spetsifitseeritud. Näiteks kui nime järgi seostatav aritmeetilise või loogilise avaldise tüüpi formaalne parameeter esineb protseduuri sisus vähemalt üks kord mõne omistamisoperaatori vasakus pooles, siis tohib vastava tegeliku parameetrina kasutada vaid muutujaid, mitte aga mistahes avaldisi. Samuti tuleb jälgida, et massiivi identifikaatorit tähendava formaalse parameetri kohal tohib tegeliku parameetrina kasutada vaid samadimensionaalse massiivi identifikaatorit ja protseduuri identifikaatorit tähendava parameetri kohal vaid niisuguse protseduuri nime, mille parameetrite arv ja sisuline tähendus sobib vaadeldava protseduuri sisus kasutatud tõlgendusega.

5. Kui tegelikuks parameetriks on rida, siis etalonkeeles kirjutatud operaatorina väljendatud protseduuri sisus tohib teda kasutada vaid tegeliku parameetrina mõne teise protseduuri poole pöördumisel, sest mingeid muid tehteid ridadega pole etalonkeeles ette nähtud. Seega mingis teises tähenduses saab rida protseduuri sisus kasutada vaid siis, kui see sisu on väljendatud koodina. Kõige sagedamini kasutatakse rida just mitmesuguste väljundseadmete poole pöördumise protseduurides, mille sisu peab loomulikult olema koostatud konkreetse arvuti käskude keeles. Selliste protseduuride kohta antavaid soovitusi vaatleme käesoleva paragrahvi lõpul.

Väärtuse järgi seostatav parameeter ei saa juba etalonkeele struktuuri kohaselt tähendada rida või lüliti identifikaatorit või protseduuri identifikaatorit, sest nendel konstruktsioonidel ei ole väärtusi (erandi võib siiski moodustada niisuguse protseduuri identifikaator, mis defineerib funktsiooni ning mille kirjelduses formaalsete parameetrite osa on tühi). Küll aga võib väärtuse järgi seostatavaks parameetriks olla massiivi identifikaator. Sel juhul protseduurioperaatori toimel moodustub lokaalne massiiv, mille indeksite rajad langevad kokku kasutatud tegeliku parameetri omadega ning mille koosseisu kuuluvad indeksitega muutujad omandavad ühtlasi vastavad algväärtused.

Samuti nagu protseduurikirjelduse formaalsete parameetrite osas nii peab ka protseduurioperaatori tegelike parameetrite osas iga kahe parameetri vahel paiknema p a r a m e e t r i e r a l d a j a, s. t. kas koma või tähtede rida koos kooloni ja sulgudega. Kõik parameetrite eraldajad loetakse aga samaväärseteks ja mingit informatsiooni nende valikuga translaatorile anda ei saa. Tavaliselt kasutatakse tähtede ridu eraldajatena just protseduurikirjeldustes, vastavates protseduurioperaatorites aga samal kohal komasid. Näiteks kui protseduurikirjeldus algab järgmiselt:

procedure *alamprogramm3 (A, B) kordajad: (c1, c2, c3) vabaliige: (c) tulemused: (F, G);*

siis võib selle protseduuri poole pöörduda ka näiteks protseduurioperaatoriga

alamprogramm3 (F)teine: (FF, 3, 5a + b, x, 13.3, A, A);

6. Protseduuride koostamisel tuleb arvestada, et peale formaalsete parameetrite tohib neis kasutada ka veel teisi identifikaatoreid. Need võivad olla selle protseduuri sisus kirjeldatud ja seega lokaliseeritud (nn. sisemised identifikaatorid), kuid põhimõtteliselt tohib samuti kasutada identifikaatoreid, mis on lokaliseeritud mõnes välimises blokis (näiteks selles, mille alguses paikneb vaadeldav protseduurikirjeldus). Niisuguseid mittelokaliseeritud identifikaatoreid sisaldavate protseduuride lülitamisel programmi peab olema väga ettevaatlik, sest protseduurioperaa-

tori kirjutisest pole nüüd näha, milliste väliste muutujate väärtusi ta võib muuta, kuhu ta võib teha suunamisi jms. Hoopis tülikas on selliseid valmis protseduure uutesse programmidesse lülitada. Seda arvestades püütakse lokaliseerimata identifikaatoritega protseduuride kasutamist tavaliselt vältida ja realiseerida kogu seos väliste blokkidega ikka parameetrite kaudu. Protseduuris mittelokaliseeritud identifikaatoreid kasutatakse praktikas peamiselt vaid siis, kui teatavad tegelikud parameetrid tuleks igas vastavas protseduurioperaatoris alati ühel ning samal kujul kirjutada.

Protseduuride koostamisel tuleb arvestada veel seda, et kui mõne formaalse parameetri identifikaator on protseduuri sisus kirjeldatud (või kasutatud mingi operaatori märgendamiseks), siis on ta sellega lokaliseeritud ja pole üldse enam formaalne parameeter vaid lihtsalt teatav uus sisemine identifikaator. See tähendab, et vastavaid tegelikke parameetreid pole protseduuri sisus (või sisu selles alamblokis, kus toimus kirjeldamine) niisugusel juhul võimalik kasutada. Järelikult formaalsete parameetrite veelkordset kirjeldamist tuleb väga ettevaatlikult kasutada, veel parem aga vältida: mitte kasutada sisemiste identifikaatoritena selliseid, mis oma kirjapildilt formaalsete parameetritega kokku langevad.

7. Kui me soovime, et protseduurikirjeldus defineeriks funktsiooni, mida saab seda protseduurikirjeldust sisaldavas blokis väärtustatud funktsioonina kasutada, siis peame protseduuri sisu nii koostama, et seal alati tuleks täitmisele (vähemalt üks) omistamisoperaator, mille vasakute poolte loetelus esineb vastava protseduuri identifikaator (olgu märgitud, et protseduuri identifikaatori igasugune teine esinemine protseduuri sisus tähendab protseduurioperaatorit või väärtustatud funktsiooni, s. t. pöördumist sellesama protseduuri poole). Peale selle peab protseduurikirjeldus nüüd algama tüübi nimega, mis seega määrab vastava funktsiooni väärtuste tüübi.

Näiteks funktsiooni, mis leiab kahe punkti vahelise kauguse tasandil, võib defineerida järgmiselt:

```
real procedure kaugus (x, y, u, v);
value x, y, u, v; real x, y, u, v;
kaugus ::= sqrt ((x — u)↑2 + (y — v)↑2);
```

Funktsiooni, mille väärtuseks on argumentideks olevate vektorite skalaarkorrutis, defineerib aga näiteks protseduurikirjeldus:

```
real procedure skalaarkorrutis (A, B) dimensioon : (N) ;
value A, B, N ; array A, B ; integer N ;
begin integer i ; real s ; s := 0.0 ;
for i := 1 step 1 until N do
s := s + A [i] × B [i] ; skalaarkorrutis := s end
```

Funktsiooni, mille väärtus mingis punktis x leitakse punktides $x_i = x_0 + ih$ $(i = 0, 1, \ldots, n)$ antud väärtuste $y_i = f(x_i)$ kaudu lineaarse interpolatsioonivalemi

$$f(x) = y_i + \frac{x - x_i}{h}(y_{i+1} - y_i)$$

(kus $x \in [x_i, x_{i+1})$ abil, võib defineerida järgmise protseduurikirjeldusega:

```
real procedure lininterp (n, y, h, x0, x);
value n, y, h, x0, x; integer n;
real h, x0, x; array y;
begin integer i; real xi;
i := entier ((x - x0)/h); xi := x0 + i × h;
lininterp := y[i] + (x - xi)/h × (y[i + 1] - y[i]);
end see protseduur annab soovitud tulemuse vaid siis, kui
x0 ⩽ x < x0 + n × h;
```

Selliselt kirjeldatud protseduuride poole pöördumine tähendab alati väärtustatud funktsiooni, mitte protseduurioperaatorit, seega peab ta paiknema mingi aritmeetilise või loogilise avaldise koosseisus, mitte omaette operaatorina. Tuleb aga rõhutada, et funktsiooni defineeriv protseduur võib üldiselt anda ka muid tulemusi peale ühe väärtuse (kuigi seda võimalust praktikas harva kasutatakse). Muuhulgas võib toimuda näiteks uute väärtuste omistamine nendele muutujatele, mis kuuluvad samasse avaldisse, kus paikneb vastav väärtustatud funktsioon. Sel juhul peab arvestama, et vastavas avaldises leitakse alati kõigepealt väärtustatud funktsioonide väärtused ja alles seejärel teostatakse ülejäänud tehted. Näiteks kui bloki alguses paikneb protseduurikirjeldus

```
integer procedure jagub (a, b, n, m); value a, n, m;
integer array a, b; integer n, m;
begin integer i, k; k := 0;
i := 1 step 1 until m do
if a[i] = a[i] ÷ n × n then
begin k := k + 1; b[k] := a[i] ÷ n end
jagub := k end on leitud n-ga jaguvate väärtuste arv ja
eraldatud need jagatised massiivi b;
```

siis selles blokis esinev operaator

```
r := N - jagub (AA, AA, 4, jagub (AA, AA, 2, N));
```

on samaväärne järgmise programmilõiguga:

```
a := 0; f := 1 step 1 until N do
if AA[f] = AA[f] ÷ 2 × 2 then
begin a := a + 1; AA[a] := AA[f] ÷ 2 end;
b := 0; f := 1 step 1 until a do
```

if $AA[f] = AA[f] \div 4 \times 4$ **then**
begin $b := b + 1$, $AA[b] := AA[f] \div 4$ **end**; $r := N - b$;
comment *8-ga mitte jaguvate väärtuste arv on leitud*;

Selles näites esines protseduuri poole pöördumine sama protseduuri poole pöördumise tegeliku parameetrina. Etalonkeeles on aga lubatud ka nn. rekursiivsed protseduurid, kus protseduuri poole pöördumine toimub sama protseduuri sisus.

Üheks lihtsaimaks selliseks protseduuriks on näiteks järgmine:

integer procedure *faktoriaal (n)*;
if $n = 1$ **then** *faktoriaal* $:= 1$
else *faktoriaal* $:= n \times$ *faktoriaal* $(n - 1)$;

Edasisi näiteid etalonkeeles kirjutatud protseduuride koostamise ja kasutamise võimalustest on toodud järgmises paragrahvis. Siin piirdume veel vaid mõne täiendava märkusega koodina esitatud sisuga protseduuride kohta.

8. Kõik operatsioonid, mis on seotud arvuti sisend- ja väljundseadmete kasutamisega, tuleb konkreetsesse esitusse lülitada koodi kujul koostatud protseduuride näol. Kuigi selliste protseduuride koostamise täpseid eeskirju etalonkeeles anda ei saa, tehakse seal siiski ettepanek muuta vastavad protseduurid standardseteks, s. t. sellisteks, mille kirjelduse translaator lülitab programmi automaatselt. Niisuguste standardsete protseduuride identifikaatorite jaoks soovitatakse reserveerida vähemalt järgmised seitse identifikaatorit, mida sel juhul ei tohi kasutada üheski teises tähenduses (sulgudes on antud identifikaatori kui ingliskeelse sõna tõlge):

insymbol (sümbol sisse),
outsymbol (sümbol välja),
inreal (reaalarv sisse),
outreal (reaalarv välja),
inarray (massiiv sisse),
outarray (massiiv välja),
length (pikkus).

Vastavate nimedega protseduuride kirjelduste kohta eeldatakse, et nad on esitatud kujul:

procedure *insymbol (kanal, rida, toodav)*;
value *kanal*; **integer** *kanal, toodav*; **string** *rida*;
⟨protseduuri sisu⟩

procedure *outsymbol (kanal, rida, viidav)*;
value *kanal, viidav*; **integer** *kanal, viidav*;
string *rida*; ⟨protseduuri sisu⟩

procedure *inreal (kanal, toodav)*; **value** *kanal*;
integer *kanal*; **real** *toodav*; ⟨protseduuri sisu⟩

procedure *outreal (kanal, viidav)*; **value** *kanal, viidav*;
integer *kanal*; **real** *viidav*; ⟨protseduuri sisu⟩

procedure *inarray (kanal, toodav)*; **value** *kanal*;
integer *kanal*; **array** *toodav*; ⟨protseduuri sisu⟩

procedur *outarray (kanal, viidav)*; **value** *kanal*;
integer *kanal*; **array** *viidav*; ⟨protseduuri sisu⟩

integer procedure *length (rida)*; **string** *rida*;
⟨protseduuri sisu⟩

Ühtlasi täpsustatakse ka nende protseduuride poole pöördumisel kasutada lubatud tegelike parameetrite kujud. Nimelt peavad vastavad protseduurioperaatorid (viimase protseduuri puhul väärtustatud funktsioon) olema järgmiste tegelike parameetritega:

insymbol (⟨aritmeetiline avaldis⟩, ⟨rida⟩, ⟨muutuja⟩)
outsymbol (⟨aritmeetiline avaldis⟩, ⟨rida⟩, ⟨aritmeetiline avaldis⟩)
inreal (⟨aritmeetiline avaldis⟩, ⟨muutuja⟩)
outreal (⟨aritmeetiline avaldis⟩, ⟨aritmeetiline avaldis⟩)
inarray (⟨aritmeetiline avaldis⟩, ⟨massiivi identifikaator⟩)
outarray (⟨aritmeetiline avaldis⟩, ⟨massiivi identifikaator⟩)
lenght (⟨rida⟩)

(komade asemel võib siin muidugi kasutada mistahes parameetrite eraldajaid).

Esimese kuue protseduuri poole pöördumisel peab esimese formaalse parameetri *kanal* väärtuseks alati andma positiivse täisarvu, mis tähendab siis vastavalt kas sisend- või väljundseadme järjekorranumbrit, mille poole toimub pöördumine. Samad protseduurid võivad teostada ka pöördumisi välise mälu poole, sel juhul parameetri *kanal* väärtus annab vastava mäluseadme või selle tsooni numbri. Järgnevas piirdume konkreetsuse huvides siiski vaid sisend- ja väljundseadmetega.

9. Vaadeldavate protseduuride sisu kohta püstitatakse järgmised eeldused.

Protseduur *insymbol* loeb esimese parameetriga määratud sisendseadmelt järjekordse põhisümboli ja võrdleb seda teiseks tegelikuks parameetriks olevas reas paiknevate põhisümbolitega. Kolmandaks tegelikuks parameetriks oleva muutuja väärtuseks omistataksegi naturaalarv, mis näitab, mitmenda põhisümboliga selles reas sisendseadmelt toodud sümbol kokku langes. Kui vastavat sümbolit tegeliku parameetrina antud reas ei ole, siis omistatakse kolmandale parameetrile väärtus null. Juhul kui sisendseadmelt saadud sümbol pole üldse ALGOLi põhisümbol, siis omistatakse kolmandale parameetrile negatiivne väärtus, mis teatava varem koostatud tabeli kohaselt vastab sellele sümbolile.

Protseduur *outsymbol* viib esimese parameetriga määratud väljundseadmesse nii mitmenda põhisümboli teise parameetri kohal olevast reast, kui suur on kolmanda parameetri väärtus. Kui kolmanda parameetri väärtus on negatiivne, siis viiakse väljundseadmesse etalonkeelde mittekuuluv lisasümbol, mis leitakse samast varem koostatud tabelist, mida kasutatakse protseduuri *insymbol* puhul.

Protseduur *inreal* loeb esimese parameetriga määratud sisendseadmelt järjekordse väärtuse ja omistab selle teiseks tegelikuks parameetriks olevale muutujale.

Protseduur *outreal* viib teise tegeliku parameetri väärtuse esimese parameetriga määratud väljundseadmesse.

Protseduur *inarray* loeb esimese parameetriga määratud sisendseadmelt teatava hulga väärtusi ja omistab need teise parameetri kohal oleva massiivi indeksitega muutujate väärtusteks. Loetavate väärtuste arvu määrab teiseks parameetriks oleva massiivi kirjeldus. Omistamise järjekord vastab indeksite väärtuste leksikograafilisele järjestusele, näiteks kirjeldusega

array *a*[*1*:*3*, *0*:*1*, *1*:*2*]

määratud massiivi muutujatele omistatakse väärtusi järjekorras

a[*1*,*0*,*1*], *a*[*1*,*0*,*2*], *a*[*1*,*1*,*1*], *a*[*1*,*1*,*2*], *a*[*2*,*0*,*1*], *a*[*2*,*0*,*2*], *a*[*2*,*1*,*1*], *a*[*2*,*1*,*2*], *a*[*3*,*0*,*1*], *a*[*3*,*0*,*2*], *a*[*3*,*1*,*1*], *a*[*3*,*1*,*2*].

Protseduur *outarray* viib esimese parameetriga määratud väljundseadmesse kõigi teise parameetriga antud massiivi indeksitega muutujate väärtused (indeksite väärtuste leksikograafilises järjestuses).

Protseduuriga *length* defineeritud funktsiooni väärtuseks on tegelikuks parameetriks kirjutatud rea pikkus, s. t. põhisümbolite arv selles reas.

10. Nende seitsme standardse protseduuri abil saab juba moodustada kõik vajalikud sisend- ja väljundprotseduurid. Toomegi lõpuks mõned niisuguste protseduurikirjelduste näited, mis ühtlasi aitavad selgitada äsjavaadeldud protseduuride kasutamist.

Tõeväärtuse viimist väljundseadmesse saab realiseerida järgmise protseduuriga:

```
procedure outboolean (kanal, tõeväärtus);
value kanal, tõeväärtus;
integer kanal; Boolean tõeväärtus;
if tõeväärtus then outsymbol (kanal, 'true', 1)
else outsymbol (kanal, 'false', 1)
```

Rea viimine väljundseadmesse (näiteks teksti trükkimine) on teostatav protseduuriga:

```
procedure outstring (kanal, rida);
value kanal; integer kanal; string rida;
begin integer i;
for i := 1 step 1 until length (rida) do
outsymbol (kanal, rida, i) end
```

Täisarvu viimiseks väljundseadmesse võib kasutada protseduuri *outreal*, kuid see on teostatav ka näiteks protseduuriga:

```
procedure outinteger (kanal, täisarv);
value kanal, täisarv; integer kanal, täisarv;
begin integer i, k;
integer array A[0 : ln(täisarv)/ln(10)]; i := 0;
if täisarv < 0 then outsymbol (kanal, '—', 1);
for k := täisarv, k ÷ 10 while k > 0 do
begin A[i] := k — k ÷ 10 × 10; i := i + 1 end
for k := i — 1 step —1 until 0 do
outsymbol (kanal, '0123456789', A[k] + 1) end
```

Veidi komplitseeritumaks osutub täisarvu sissetoomise protseduur. Eeldades, et iga täisarv on sisendseadmes esitatud numbrite jadana, mille ees võib olla märk + või — ning mille järel paikneb koma, võib vajaliku protseduuri kirjeldada järgmiselt:

```
procedure ininteger (kanal, täisarv);
value kanal; integer kanal, täisarv;
begin integer i, j; Boolean miinus;
täisarv := 0; miinus := false;
for i := 1, i + 1 while j = 0 do
insymbol (kanal, '0123456789— +', j);
if j = 11 then miinus := true;
if j > 10 then j := 1;
for i := 1, i + 1 while j ≠ 13 do
begin täisarv := 10 × täisarv + j — 1;
insymbol (kanal, '123456789 —+,', j) end
if miinus then täisarv := — täisarv end
```

§ 31. ETALONKEELES KIRJUTATUD PROGRAMMIDE NÄITEID

Järgnevates näidetes on nõutud algoritmid vormistatud kõik iseseisvate ALGOL-programmidena. Kui mõnda nendest algoritmidest on tarvis kasutada mingi ulatuslikuma programmi ühe blokina, siis tuleb muidugi ära jätta n. ö. väliste muutujate kirjeldused ja algväärtuste omistamine nendele muutujatele.

Välisseadmete poole pöörduvate standardsete protseduuride kasutamisel on järgnevates näidetes eeldatud, et kanal 1 tähendab sisendseadet (näiteks perfokaartsisendit), kanal 2 aga trükkimisseadet (mis pärast igakordset protseduuri *outreal, outarray, outboolean, outstring* või *outinteger* täitmist pöörab ette uue rea).

Näide 1. Trükkida ja salvestada esimese n algarvu tabel (salvestamine on vajalik muidugi vaid siis, kui programmi kavatsetakse jätkata nii, et seal tuleb neid arvusid kasutada). Trükkimisel jätta iga kümne arvu tagant tühi rida.

Nõutud tabeli koostab näiteks järgmine programm (mis eeldab, et algandmena vajalikku täisarvu n kandev perfokaart on paigutatud sisendseadmesse):

```
begin integer n; ininteger (1, n);
  begin comment: uus blokk tuleb moodustada
    algarvude salvestamiseks määratud massiivi A kirjeldamiseks;
    integer i, k, m; integer array A[1 : n];
i := 0; k := 1; for k := k + 1 while i < n do
begin for m := 2 step 1 until k — 1 do
if k = m × (k ÷ m) then go to kordarv;
comment: see oli kontroll, kas k jagub m-ga;
if i = 10 × (i ÷ 10) then outstring (2, '␣');
i := i + 1, A[i] := k; outinteger (2, k); kordarv:
end märgendiga 'kordarv' märgendati siin tühi operaator;
end algarvude tabeli koostamine on lõppenud;
end
```

Näide 2. Arvutada (ning trükkida)

$$u = e^{\sin \sqrt{y}} + e^{\cos \sqrt{y}},$$

kus

$$y = \begin{cases} x_1^2 + x_n^2, & \text{kui } 0{,}5 \leqslant z < 1{,}5 \\ x_1^4 + x_n^4, & \text{kui } 1{,}5 \leqslant z < 2{,}5 \\ \sum_{i=1}^{n} x_i^2, & \text{kui } 2{,}5 \leqslant z < 3{,}5 \\ \sum_{i=1}^{n} x_i^4, & \text{kui } 3{,}5 \leqslant z < 4{,}5 \\ 1, & \text{kui } z < 0{,}5 \text{ või } z \geqslant 4{,}5. \end{cases}$$

ja z on arvude x_i aritmeetiline keskmine.

Selle funktsiooni väärtuse võib arvutada näiteks järgmiselt:

```
begin integer n, i; ininteger (1, n);
begin real array x[1 : n]; real y, z;
switch alajuht := juht1, juht2, juht3, juht4;
inarray (1, x); z := 0.0;
```

```
for i := 1 step 1 until n do z := z + x[i]; z := z/n;
go to alajuht [z]; y := 1.0; go to lõpp;
juht1 : y := x[1]↑2 + x[n]↑2; go to lõpp;
juht2 : y := x[1]↑4 + x[n]↑4; go to lõpp;
juht3 : y := 0.0; for i := 1 step 1 until n do
y := y + x[i]↑2; go to lõpp;
juht4 : y := 0.0; for i := 1 step 1 until n do
y := y + x[i]↑4; lõpp:
outreal (2, exp(sin(sqrt(y))) + exp(cos(sqrt(y))))
end
end
```

Näide 3. Leida kahe etteantud (positiivse) täisarvu kahendsüsteemis liitmisel tekkivate ülekannete arv.

```
begin integer array n[1 : 2];
integer i, j, k; Boolean ü;
procedure kahendsüsteemi (A, n, k, j);
value j, k, n; integer j, k, n; Boolean array A;
begin integer i; for i := 0 step 1 until k do
begin A[j, i] := n − n ÷ 2 × 2 ≠ 0; n := n ÷ 2 end
end n on teisendatud k-kohaliseks kahendarvuks;
inarray (1, n);
k := if n[1] > n[2] then ln(n[1])/ln(2)
else ln(n[2])/ln(2);
begin Boolean array N[1 : 2, 0 : k];
for i := 1, 2 do kahendsüsteemi (N, n[i], k, i);
ü := false; j := 0; for i := 0 step 1 until k do
begin ü := ü ∧ ¬(N[1, i] ≡ N[2, i]) ∨ N[1, i] ∧ N[2, i];
if ü then j := j + 1
end ülekannete arv j leitud; outinteger (2, j);
end
end
```

Näide 4. Arvutada integraali

$$\int_a^b x^k \sin^{2m} x\, dx$$

väärtus etteantud täpsusega $\varepsilon > 0$. Integraali arvutamiseks kasutada trapetsvalemit

$$\int_a^b f(x)\,dx = h \sum_{i=1}^{n} \frac{f(a+ih)+f(a+(i-1)h)}{2},$$

kus $h = (b-a)/n$. Sõlmede arvu n kahekordistada seni, kuni saavutatakse nõutud täpsus.

Vajaliku arvutuse teostab näiteks järgmine programm (mida on muide lihtne kohandada ka mistahes määratud integraali leidmiseks):

```
begin real a, b, epsilon, y1, y2;
real procedure trapets (n, f); value n;
integer n; real procedure f;
begin integer i; real s, h; s := 0.0; h := (b — a)/n;
for i := 1 step 1 until n do
s := s + f(a + (i — 1) × h) + f(a + i × h);
trapets := s × h/2.0
end trapetsvalem, kus a ja b on lokaliseerimata;
real procedure f(x); value x; real x;
f := sin(x)↑(2 × m) × x↑k; comment: selles funktsioonis on
m ja k lokaliseerimata;
integer k ,m, n; ininteger (1, k); ininteger (1, m);
comment: mingi teise integraali arvutamisel tuleb funktsiooni
f kirjeldus, samuti täisarvude k ja m kirjeldamine ning sisse-
toomine asendada;
inreal (1, a); inreal (1, b); inreal (1, epsilon);
y1 := trapets (2, f); n := 2;
for n := 2 × n while abs(y1 — y2) > epsilon do
begin y2 := y1; y1 := trapets (n, f) end;
outreal (2, y1);
end integraali väärtus on leitud;
```

Näide 5. Trükkida funktsioonide

$$f(x)=\frac{\cos x+\sqrt{1+\sin^2 x}}{\sin^2(1-x^2)+\sqrt{1+x^2}} \text{ ja } g(x)=\frac{\sin x}{\sin^2(1-x^2)+\sqrt{1+x^2}}$$

väärtuste tabel etteantud kohtadel $x_1, \ldots, x_n$. Tabel tuleb trükkida argumentide kasvamise järjekorras, s. t. kui arvud x_i ei ole antud kasvavas järjekorras, siis peab nad kõigepealt järjestama.

Nõutud tabeli trükib arvukolmikutena $(x_i, f(x_i)$ ja $g(x_i))$ näiteks järgmine programm:

```
begin integer i, j, k, n; ininteger (1, n);
begin real array x[1 : n];
real procedure nimetaja (x); value x;
real x; nimetaja := sqrt(1.0 + x↑2) + sin(1.0 — x↑2)↑2;
real procedure f(x); value x; real x;
f := (cos(x) + sqrt(1.0 + sin(x)↑2))/nimetaja (x);
real procedure g(x); value x; real x;
g := sin(x)/nimetaja(x);
real h; inarray (1, x);
for k := 1 step 1 until n do
```

```
begin h := x[k]; j := k;
for i := k + 1 step 1 until n do
if x[i] < h then
begin h := x[i]; j := i end;
for i := j step — 1 until k + 1 do
x[i] := x[i — 1]; x[k] := h
end jada x[i] on järjestatud;
for j := 1 step 1 until n do
begin outstring (2, '␣'); outreal (2, x[j]);
outreal (2, f(x[j])); outreal (2, g(x[j]));
end tabel on trükitud;
end
end
```

Näide 6. Leida lineaarse võrrandisüsteemi

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= a_{10}\\
a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= a_{20}\\
&\ldots\\
a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n &= a_{m0}
\end{aligned}$$

(kus $m < n$) erilahend, kasutades selleks Gaussi meetodit ja võttes kõik vabad tundmatud võrdseks nulliga.

Võrrandisüsteemi võib nii lahendada näiteks järgmise programmiga (milles eeldatakse, et algandmed on paigutatud sisendseadmesse järjekorras: n, m, a_{10}, a_{11}, ..., a_{1n}, a_{20}, a_{21}, ..., $a_{m-1,n}$, a_{m0}, a_{m1}, ..., a_{mn}):

```
begin integer m n; ininteger (1, n); ininteger (1, m);
begin real array a[1 : m, 0 : n];
integer array x[1 : n]; integer i, j, k, l;
inarray (1, a);
for j := 1 step 1 until n do x[j] := 0;
for i := 1 step 1 until m do
begin for j := 1 step 1 until n do
if a[i, j] ≠ 0.0 then
begin for l := 0 step 1 until n do
a[i, l] := a[i, l]/a[i, j];
for k := 1 step 1 until m do
if k ≠ i then for l := 0 step 1 until n do
begin a[k, l] := a[k, l] —a[i, l] × a[k, j];
x[j] := i
end maatriksi veerg teisendatud ühikvektoriks;
go to järgmine rida;
end if a[i, 0] ≠ 0 then
begin outstring (2, 'pole lahendit');
go to lõpp
end; järgmine rida:
```

```
    end maatriks teisendatud 'diagonaalkujule';
    for j := 1 step 1 until n do
    if x[j] = 0 then outreal (2, 0.0)
    else outreal (2, a[x[j], 0]);
    end; lõpp:
end
```

Näide 7. Lahendada kanoonilisel kujul antud lineaarne planeerimisülesanne: leida mittenegatiivsed suurused $x_1, \ldots, x_n$ nii, et oleks rahuldatud võrratused

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &\leqslant a_{10} \\
a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &\leqslant a_{20} \\
\ldots\ldots\ldots

```
begin g[j] := a[k, j]/x; a[k, j] := 0.0 end
g[l] := 1.0/x; for i := 0 step 1 until m do
begin f[i] := a[i, l]; a[i, l] := 0.0 end
f[k] := —1.0; for i := 0 step 1 until m do
begin x := f[i]; for j := 0 step 1 until n do
a[i, j] := a[i, j] — x × g[j]
end maatriksi teisendamine lõppenud;
i := mittebaas [l]; mittebaas [l] := baas [k];
baas [k] := i; go to Op1;
trükkimine: for i := 0 step 1 until m do
begin outinteger (2, baas [i]);
outreal (2, a[i, 0]); outstring (2, '␣')
end ülesanne on lahendatud;
end; lõpp:
end
```

§ 32. ALGOLi KONKREETNE ESITUS — MALGOL

1. Tutvustame lõpuks veel üsna lühidalt üht Eesti NSV-s kasutusel olevat ALGOLi konkreetset esitust.[31] See konkreetne esitus — MALGOL — on koostatud ENSV TA Küberneetika Instituudis (M. Kotli) ning ta on määratud kasutamiseks arvuteil «Minsk-2» ja «Minsk-22».

Etalonkeelega võrreldes on MALGOL teatud määral lihtsustatud, mis väljendub eeskätt mõningate konstruktsiooniühikute täielikus väljajätmises keelest. Olulisemaiks neist on loobumine bloki mõistest ja sellega seotud lokaliseerimisest (kuigi lokaliseerumine säilib protseduurides). Samuti on ära jäetud tüübikirjeldused, nimelised avaldised ning tinglikud avaldised. Mõningal määral on muudetud protseduurikirjeldusi. Kõrvuti väljajätmistega on aga ka mõningaid täiendavaid konstruktsioone juurde toodud.

Järgnevas käsitluses pöörame peatähelepanu just neile mõistetele, mille poolest MALGOL etalonkeelest erineb. Sealjuures piirdume enamasti üksnes sõnaliste seletustega, andes täpsed definitsioonid vaid sel määral, kuivõrd nad etalonkeele vastavatest definitsioonidest oluliselt erinevad (kõik mõisted, mille tähendus ühtib etalonkeeles antuga, jätame enamasti defineerimata). Kogu käsitlus toimub enam-vähem samas järjekorras nagu etalonkeele kirjeldamisel.

2. Vaadeldava konkreetse esituse erinevused etalonkeelest algavad juba põhisümbolite loetelus. Tingituna arvuti «Minsk-2» sisendseadmete konstruktsioonist on põhisümbolite arvu tunduvalt vähendatud.

---

[31] Selle kirjeldus on ilmunud eraldi väljaandena: vt. [24].

Kokku on MALGOLis 50 p õ h i s ü m b o l i t: inglise tähestiku 26 suurt tähte, 10 numbrit ja 14 piirajat (peale selle saab mõnedes standardsetes protseduurides kasutada küll veel nelja täiendavat sümbolit, mida me aga keele kirjeldusse ei võta). Jättes juba tuntud definitsioonid esitamata võime kirjutada:

⟨põhisümbol⟩ ::= ⟨täht⟩ | ⟨number⟩ | ⟨piiraja⟩
⟨piiraja⟩ ::= , | · | ю | : | ∋ | ␣ | (|) | ' | + | — | ? | / | =

Etalonkeele kõik ülejäänud põhisümbolid MALGOLis kas puuduvad või tuleb esitada liitsümbolitena. Arvestades veel, et ka MALGOLi mõned põhisümbolid on oma kirjapildilt etalonkeele vastavatest sümbolitest erinevad, esitame tabeli 36 kujul etalonkeele kõigi mittesõnaliste piirajate vasted MALGOLis:

Tabel 36

Etalonkeeles	MALGOLis	Etalonkeeles	MALGOLis	Etalonkeeles	MALGOLis
,	,	]	);	∧	?.
.	.	'	'	∨	+.
10	ю	+	+	⊃	/.
:	:	—	—	≡	=.
;	∋	×	?	<	(:
:=	:=	/	/	≤	(=
␣	␣	÷	puudub	=	=
(	(	↑	??	≥	=)
)	)	┐	—.	>	:)
[	·(			≠	=/

Sõnalisi põhisümboleid MALGOLis ei ole, neid asendavad reserveeritud sõnad. R e s e r v e e r i t u d s õ n a tunnuseks on tema viimasele tähele järgnev apostroof. Etalonkeele põhisümboleist on MALGOLis reserveeritud sõnadena esindatud vaid järgmised seitseteist:

TRUE' *BEGIN'* *IF'*
FALSE' *END'* *THEN'*
STEP' *ARRAY'* *ELSE'*
UNTIL' *PROCEDURE'* *FOR'*
WHILE' *SWITCH'* *DO'*
COMMENT' *GOTO'*

Seega MALGOLis puuduvad etalonkeele sõnalised põhisümbolid: **Boolean, integer, real, own, string, value** ja **label.**

Lisaks nimetatutele on aga MALGOLis sisse toodud veel seitse põhisümboli funktsioonis kasutatavat reserveeritud sõna. Need on (sulgudes on antud vastava sõna tähendus):

FINISH'	(lõpp)
KEY'	(võti)
OCTAL'	(kaheksandarv)
PI'	(arv π)
START'	(lähe)
STOP'	(peatus)
SUBROUTINE'	(alamprogramm)

Et reserveeritud sõnad koosnevad tähtedest, siis tuleb programmis nende ette alati panna tühik siis, kui eelmine konstruktsioon lõpeb tähe või numbriga.

3. Identifikaator võib vaadeldavas konkreetses esituses sisaldada kuitahes palju sümboleid (nagu etalonkeeleski), kuid translaator arvestab neist vaid kuut esimest. Seega näiteks identifikaatorid *DISKREPANTS* ja *DISKREETNE* loetakse transleerimisel identseteks, nendega ei saa tähistada erinevaid muutujaid. Niisugune pikkuse kitsendus käib muide ka reserveeritud sõnade kohta: pikemates reserveeritud sõnades on olulised vaid kuus esimest tähte. Nii näiteks reserveeritud sõna *PROCEDURE'* võib kirjutada ka kujul *PROCED'* või soovi korral isegi kujul *PROCED SEEONPROTSEDUUR'*.

Arvude kirjutamisel tuleb kõigepealt silmas pidada seda, et translaator arvestab igas arvus ülimalt üheksat tüvenumbrit (kuigi kirjutada võib neid ka rohkem). Lubatud ei ole kasutada niisuguseid arvusid, mis koosnevad ainult järgust või mille mantiss koosneb üksnes lihtmurrust. Seega näiteks kirjutised *10 — 3* ja *.1 10 1* ei ole MALGOLis arvud. Arvude kasutamisel peab veel arvestama, et nad tohivad pärineda vaid vahemikust

$$[0{,}542101 \cdot 10^{-19},\ 0{,}9223372 \cdot 10^{19}].$$

Peale numbrite abil kirjutatavate arvude saab MALGOLis kasutada veel vahetult arvu $\pi = 3{,}1415927$, mida asendab reserveeritud sõna *PI'*.

Lisaks kümnendsüsteemis kirjutatud arvudele on MALGOLis ette nähtud ka kaheksandarvude kasutamise võimalus:

⟨kaheksandarv⟩ ::= *OCTAL'* ⟨täisarv⟩

Siinjuures tuleb arvestada, et enam kui 12-st numbrist koosneva kaheksandarvu korral jätab translaator esimesed numbrid arvestamata. Kaheksandarvude kirjutamisel peab muidugi silmas pidama veel seda, et numbreid 8 ja 9 kaheksandsüsteemis ei ole.

Konstruktsioon r i d a defineeritakse MALGOLis veidi lihtsustatud kujul:

⟨rida⟩ ::= '⟨puhas rida⟩'

4. M u u t u j a d on MALGOLis kõik automaatselt reaalarvude tüüpi. Seega langeb ühtlasi ära tüübikirjelduse mõiste. M a s s i i v i k i r j e l d u s e puhul tuleb aga täiendava kitsendusena arvestada seda, et massiivi segment võib sisaldada vaid üheainsa massiivi identifikaatori, s. t. igale massiivi identifikaatorile peab järgnema muutumisrajade paar (indekssulgudes).

Et MALGOLis puudub bloki ja seega ka lokaliseerituse mõiste, siis tuleb massiivikirjeldus paigutada niisugusele kohale programmis, et muutumisrajade loetelus esinevate avaldiste väärtusi oleks juba võimalik arvutada, s. t. et kõigile neis avaldistes esinevatele muutujatele oleks omistatud väärtused. Ühtlasi tuleb aga arvestada seda, et massiiv peab olema kirjeldatud enne, kui kasutatakse vastavaid indeksitega muutujaid. Kui mingit massiivi programmis enam tarvis ei ole, siis võib ta mälu kokkuhoiu huvides kustutada spetsiaalse standardse protseduuri *DELETE'* (= kustutada) abil. Seda protseduuri tuleb kasutada ka siis, kui näiteks soovitakse uue kirjelduse abil muuta massiivi indeksite muutumisrajasid. Olgu märgitud, et kustutamata massiivi uuestikirjeldamise jätab translaator täitmata.

V ä ä r t u s t a t u d f u n k t s i o o n i mõiste erineb nüüd etalonkeeles antud definitsioonist üksnes selle poolest, et parameetri eraldajana kasutatakse vaid koma. Tuleb arvestada, et kui tegelikuks parameetriks on massiivi identifikaator, siis peab selle järele tunnusena kirjutama punkti; kui aga parameetriks on protseduuri identifikaator, siis peab sellele järgnema ümarsulgude paar. Seega tegeliku parameetri definitsioon tuleb nüüd esitada kujul

⟨tegelik parameeter⟩ ::= ⟨avaldis⟩ | ⟨rida⟩ | ⟨massiivi identifikaator⟩. | ⟨lüliti identifikaator⟩ | ⟨protseduuri identifikaator⟩ ()

MALGOLi s t a n d a r d s e t e f u n k t s i o o n i d e hulka kuuluvad kõik etalonkeeles soovitatud funktsioonid. Lisaks neile on aga juurde toodud veel tangens (identifikaatoriga *TG'*), arkussiinus (*ARCSIN'*), hüperboolne siinus (*SH'*), kaheksandarvu teisendamine reaalarvuks (*FLOAT'*), reaalarvu teisendamine kaheksandarvuks (*FIX'*) jt.

5. A r i t m e e t i l i s t e a v a l d i s t e ainsa liigi moodustavad MALGOLis aritmeetilised lihtavaldised, s. t. puudub tingliku avaldise mõiste. Etalonkeelega võrreldes on teatavaks erinevuseks veel tehete eelisjärjekorra täpsustamine: jagamine teostatakse enne korrutamist ja lahutamine enne liitmist.

Tuleb muidugi meeles pidada, et aritmeetiliste tehete märgid kirjutatakse MALGOLis ebatavaliselt (vt. paragrahvi alguses toodud tabel). Näiteks avaldis $\sqrt{p^2-4q}$ tuleb nüüd esitada kujul

SQRT'(P??2 — 4?Q)

Et tingliku avaldise mõiste MALGOLis puudub, siis ka loogilise avaldisena saab siin kasutada vaid loogilisi lihtavaldisi. Täiendusena etalonkeeles antud definitsioonile võib nüüd võrdlus koosneda ka ainult ühest aritmeetilisest (liht-) avaldisest. Sellise *võrdluse* (mida tohib kasutada vaid tingimuslikus operaatoris piiraja *IF'* järel) väärtuseks on *TRUE'* siis, kui avaldise väärtus osutub negatiivseks. Seega kirjutised

IF' ⟨aritmeetiline avaldis⟩ *THEN'*

ja

IF' ⟨aritmeetiline avaldis⟩ *THEN'*

on täpselt sama tähendusega. Rõhutame siinjuures veel kord, et vastavalt paragrahvi alguses toodud tabelile on nii loogiliste tehete märkide kui ka võrdlusmärkide kirjutusviis MALGOLis tavalisest erinev.

Arvuti «Minsk-2» võimaluste täielikuma ärakasutamise huvides on MALGOLis sisse toodud veel eriliiki loogiline avaldis, mille väärtus sõltub vastava võtme asendist juhtimispuldil. Selline avaldis koosneb lihtsalt reserveeritud sõnast *KEY'* ja sellele järgnevast kaheksandarvust (võtme järjekorranumber).

Nimeliste avaldiste ainsateks liikideks on MALGOLis märgend ja väärtustatud lüliti.

6. Operaatorite kirjutamisel peab kõigepealt arvestama, et semikoolonit asendab MALGOLis sümbol ∋. Operaatoriliikidest puudub blokk, lisaks aga on võetud ühest reserveeritud sõnast koosnev operaator *STOP'*, mis peatab arvuti. Et bloki mõiste puudub, siis võib kirjeldusi ja operaatoreid paigutada programmi mistahes järjekorras. Programmis esimesena täitmisele tulevaks operaatoriks on alati

START' ⟨märgend⟩ ∋

mis annab juhtimise vastava märgendiga varustatud operaatorile. Iga programm peab sisaldama täpselt ühe sellise operaatori, lõppema peab programm aga operaatoriga *FINISH'*

Omistamisoperaatori vasakute poolte loetelu tohib MALGOLis sisaldada vaid ühe vasaku poole. Seevastu aga pole omistamisoperaatorite moodustamisel nüüd enam tarvis silmas pidada avaldise ja uut väärtust omandava muutuja tüüpe.

Suunamisoperaator ei erine etalonkeeles defineeritust. Tuleb ainult arvestada, et piirajale *GOTO'* järgnevaks nimeliseks

avaldiseks tohib olla vaid märgend või väärtustatud lüliti. Samuti pole tarvis täpsustada ei tingimusliku operaatori ega ka tsüklioperaatori definitsiooni.

7. Protseduuride kirjeldamine toimub MALGOLis veidi teisiti kui etalonkeeles. Kõigepealt on omaette klassiks — alamprogrammideks eraldatud need protseduurid, kus puuduvad formaalsed parameetrid:

⟨alamprogrammikirjeldus⟩ ::= *SUBROUTINE'* ⟨alamprogrammi identifikaator⟩ ∋ ⟨operaator⟩ ∋

Kui alamprogrammi sisuks olev operaator on liitoperaator, siis peab ta olema koostatud nii, et tema täitmine lõpeb alati viimasena kirjutatud operaatoriga (mis võib tarbe korral olla ka tühi operaator). Seega alamprogrammi sisust ei saa juhtimist välja suunata.

Pöördumine alamprogrammi poole toimub ikka protseduurioperaatoriga, mis parameetrite puudumise tõttu koosneb ainult alamprogrammi identifikaatorist.

Üldine protseduurikirjeldus erineb etalonkeele puhul defineeritust eeskätt selle poolest, et puudub nii väärtuste osa kui ka spetsifikatsioonide osa. Väärtuste osa puudumisest tuleneb, et parameetreid ei jaotata nende seostamisviisi järgi. Üldiselt toimub seostamine alati nime järgi, kui aga tegelikuks parameetriks on (ühest muutujast erinev) avaldis, siis väärtuse järgi.

Kuigi spetsifikatsioonide osa nüüd puudub, on formaalsete parameetrite sisuline tähendus siiski täpsustatud nende kirjapildiga. Nimelt kui mingi formaalne parameeter tähendab massiivi identifikaatorit, siis tuleb tema järele kirjutada (formaalsete parameetrite loetelus) punkt, kui aga parameeter tähendab protseduuri identifikaatorit, siis peab talle järgnema ümarsulgude paar.

Rõhutame, et MALGOLis ei tohi formaalne parameeter tähendada märgendit või lüliti identifikaatorit. Seega ka mistahes protseduuri sisust juhtimist välja suunata ei saa — protseduuri täitmine peab lõppema tema viimase operaatoriga.

Protseduuri sisuks tohib MALGOLi puhul olla nii liht- kui ka liitoperaator, kusjuures selle operaatori järele tuleb kirjutada veel piiraja ∋. Koodi on protseduuri sisus kasutatud vaid standardsetes protseduurides.

Kõik protseduuri sisus kasutatud identifikaatorid (peale standardsete funktsioonide ja protseduuride) on nüüd lokaliseeritud. Sealjuures massiivid, mis pole formaalseteks parameetriteks, tuleb protseduuri sisus tingimata kirjeldada. Protseduuri sisus võib anda alamprogrammikirjeldusi, kuid mitte protseduurikirjeldusi. Protseduuri rekurrentne kasutamine on keelatud.

Erinevalt etalonkeelest defineerib nüüd iga protseduurikirjeldus ühtlasi ka funktsiooni. Selle funktsiooni väärtuseks on protseduuri

sisus viimasena täidetud omistamisoperaatoris omistatud väärtus (sealjuures ei tarvitse, isegi ei tohi, selle omistamisoperaatori vasakus pooles olla vastava protseduuri identifikaator). Seega iga pöördumist protseduuri poole tõlgendab translaator kas väärtustatud funktsioonina või protseduurioperaatorina sõltuvalt sellest, kas vastav kirjutis paikneb mõnes avaldises (ehk omaette avaldisena) või iseseisva operaatorina.

Standardsete protseduuride loetelu on MALGOLis üsna ulatuslik ja täieneb pidevalt. Seetõttu piirdume siin vaid mõne olulisema sisend-väljundprotseduuri nimetamisega (üksikasjalikuks tutvumiseks kõigi olemasolevate standardsete protseduuridega tuleb kasutada vastavaid juhendeid).

Mälu vabastamine tarbetuks muutunud massiivide alt toimub näiteks protseduuriga

DELETE′ (A1., A2., B., C.) ⋑

mis antud juhul kustutab neli massiivi. Kustutatavate massiivide arv ei ole piiratud, rõhutame vaid, et iga tegeliku parameetri kohal oleva massiivi identifikaatori järele tuleb kirjutada punkt. Samad märkused käivad ka kõigi järgnevate protseduuride kohta.

Algandmete arvutisse viimiseks võib kasutada protseduure nimetustega *READ1, READARRAY′* ja *READAO′*. Esimene nendest protseduuridest on määratud lihtmuutujate algväärtuste toomiseks perfolindilt: loetakse nii mitu arvu, kui mitme muutuja identifikaatorid on kirjutatud parameetrite kohale. Teine protseduur toob perfolindilt massiivide algväärtused (leksikograafilises järjestuses). Mõlema nende protseduuri puhul tõlgendatakse andmeid kümnendarvudena. Kolmas sisendprotseduur *READAO′* on seevastu määratud kaheksandsüsteemis antud andmetega massiivide algväärtuste toomiseks perfolindilt (samuti leksikograafilises järjestuses).

Väljundpotseduuridest on olulisemad *PRINT1′. PRINTARRAY′* ja *TEXT10′*. Esimene neist trükib parameetrite kohale kirjutatud lihtmuutujate väärtused, teine vastavatesse massiividesse kuuluvate muutujate väärtused, kolmas aga üldiselt mistahes teksti. Kaks esimest protseduuri lähemat selgitust ei vaja, kolmanda võimalustega tuleb aga tutvuda vastava spetsiaalse juhendi abil.

9. Toome lõpuks veel ühe näite MALGOLis kirjutatud programmi kohta. Selleks et võrdlemine etalonkeeles kirjutatud programmiga lihtsamaks osutuks, võtame uuesti vaatlusele eelmises paragrahvis toodud näite 6.

Leida lineaarse võrrandisüsteemi

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = a_{10}$$
$$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = a_{20}$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = a_{m0}$$

(kus $m < n$) erilahend, kasutades selleks Gaussi meetodit ja võttes kõik vabad tundmatud võrdseks nulliga.

Püstitatud ülesande lahendab näiteks järgmine programm:

```
ALGUS: READ1' (M, N) ∋
ARRAY' A.(1 : M, 0 : N)., x.(1 : N). ∋
READARRAY'(A.) ∋
FOR'J := 1 ␣ STEP'1 ␣ UNTIL'N ␣ DO'X.(J). := 0 ∋
FOR'I := 1 ␣ STEP'1 ␣ UNTIL'M ␣ DO'
BEGIN'FOR'J := 1 ␣ STEP'1 ␣ UNTIL'N ␣ DO'
IF'A.(I, J). = /0 ␣ THEN'
BEGIN'FOR'L := 0 ␣ STEP'1 ␣ UNTIL'N ␣ DO'
A.(I, L). := A.(I, L)./A.(I, J). ∋
FOR'K := 1 ␣ STEP'1 ␣ UNTIL'M ␣ DO'
IF'K = /I ␣ THEN'FOR'L := 0 ␣ STEP'1 ␣ UNTIL'N ␣
  ␣ DO'
BEGIN'A.(K, L). := A.(K, L). — A.(I, L).?
A.(K, J). ∋ X.(J). := I ␣
END' ∋ GOTO'JARGMI ∋
END' ∋ IF'A.(I, 0). = /0 ␣ THEN'
BEGIN'TEXT10'('POLE ␣ LAHENDIT',' ␣ < ≡') ∋
  ∋ GOTO'LOPP ␣
END' ∋ JARGMI :
END' ∋ FOR'J := 1 ␣ STEP'1 ␣ UNTIL'N ␣ DO'
IF'X.(J). = 0 ␣ THEN'PRINT1'(X.(J).) ␣
ELSE'PRINT1'(A.(X.(J)., 0).) ∋ LOPP : STOP' ∋
START'ALGUS ∋ FINISH' ∋
```

Lisa

ARVUTI «MINSK-32».

1. Universaalne elektronarvuti «Minsk-32» (konstrueeritud 1968. aastal) sobib nii teaduslik-tehniliste, majanduslike kui ka statistiliste ülesannete lahendamiseks.[32] Tema seadmete põhikomplekti kuuluvad: keskne juhtimisseade (KJS), aritmeetiline seade (AS), informatsiooni vahetamise seade (VS), operatiivmälu (OM), keskne juhtimispult (KJ) koos kirjutusmasinaga, kuus magnetlindiseadet (MLS), seadmed informatsiooni salvestamiseks perfokaartidelt (SK) ja -lintidelt (SL), seadmed informatsiooni väljastamiseks perfokaartidele (VK) ja -lintidele (VL), alfabeetiline trükkimisseade (TR), elektrooniline aja-andur (EA), perforaatorid programmi ja andmete kandmiseks perfokaartidele ja -lintidele.

Arvuti seadmete põhikomplekti on võimalik veel täiendada sobivalt valitud lisaseadmetega (vt. arvuti põhimõtteline skeem joonisel 61).

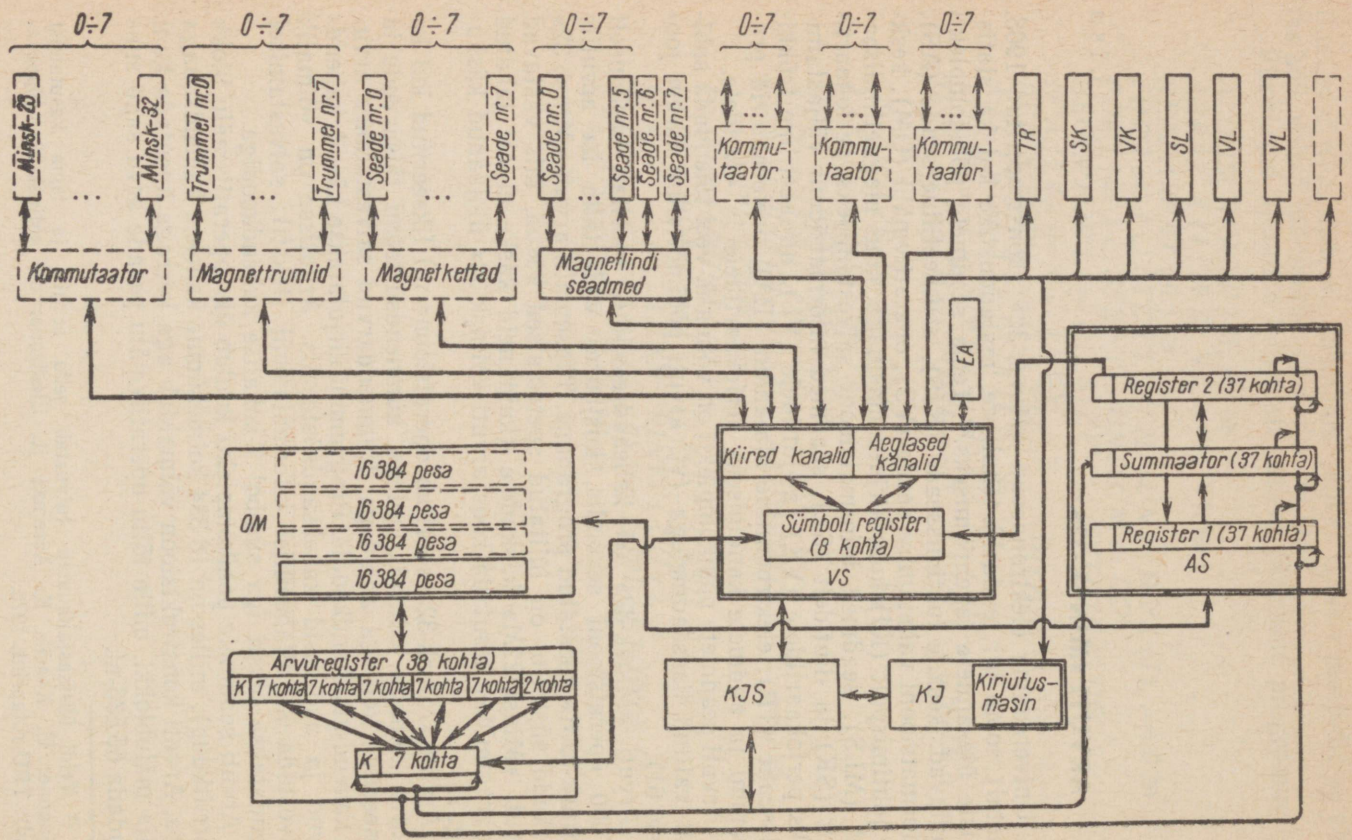
Arvuti «Minsk-32» on kaheaadressiline, sooritab keskmiselt 25 000 operatsiooni sekundis, kusjuures võib täita ka arvutile «Minsk-22M» koostatud programme. Täpsemalt, arvuti «Minsk-32» käskude süsteemi on lülitatud spetsiaalsed käsud, mis vastavad arvuti «Minsk-22M» käskudele (kuid neid käske ei ole lubatud programmides kasutada koos arvuti «Minsk-32» ülejäänud käskudega).

Arvuti «Minsk-32» sooritab operatsioone nii fikseeritud kui ka liikuva komaga kahendarvudega, kümnendsüsteemi täisarvude ja fikseeritud komaga arvudega. Kümnendarvud salvestatakse arvutis kahend-kümnendkoodis. Programmi kirjutatakse käsud, kahendarvud ja aadressid kaheksandsüsteemis. Arvutisse on võimalik salvestada ka alfabeetilist informatsiooni (teksti): salvestamisel asendatakse teksti iga sümbol 7-kohalise kahendkoodiga.

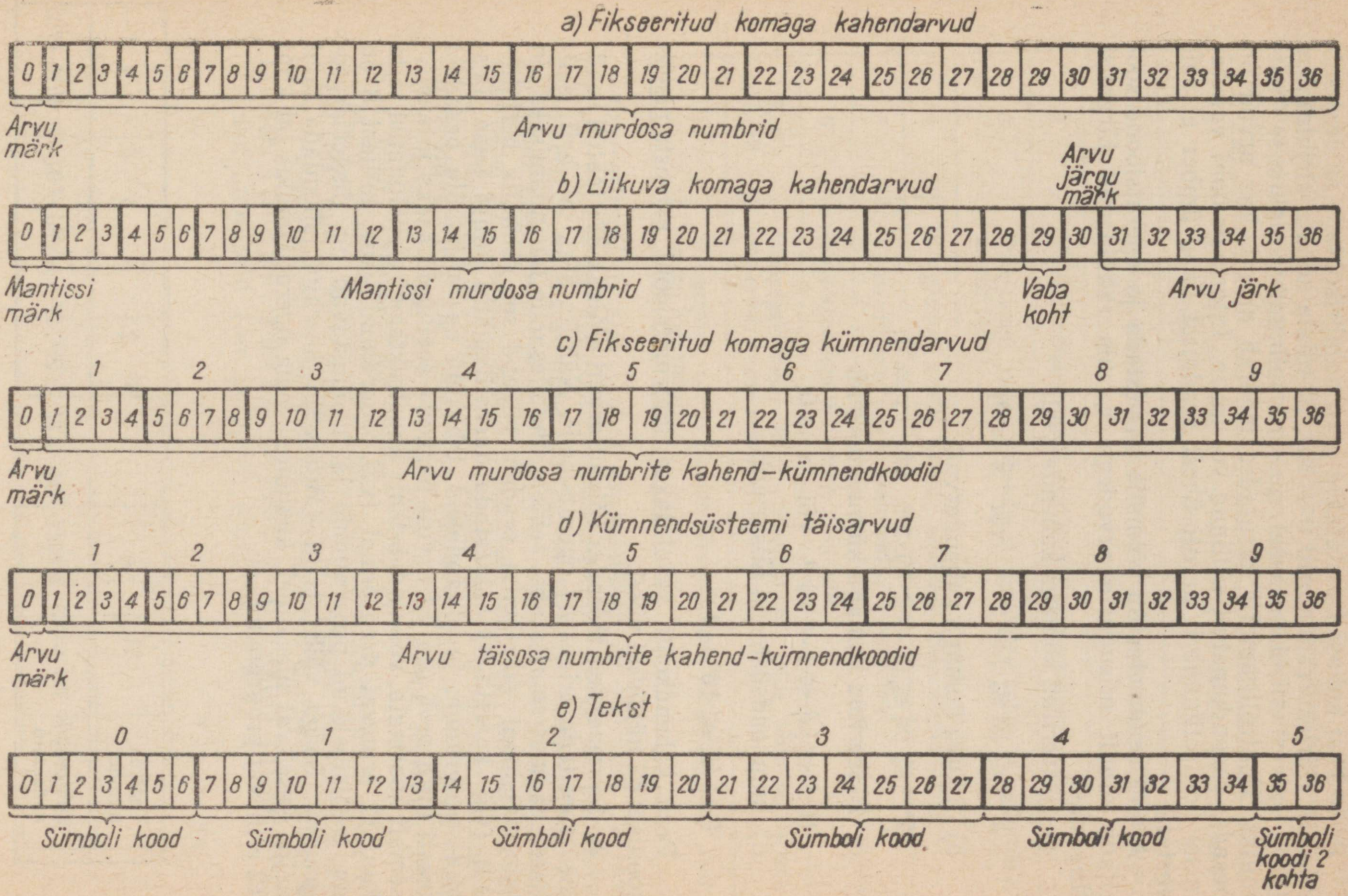
Arvuti seadmete põhikomplekti kuulub üks operatiivmälu blokk (ferriitkuup), milles on 16 384 kolmekümne kaheksa kahendkohaga pesa. Arvuti konstruktsioon võimaldab aga korraga kasutada kuni nelja mälublokki, mille tõttu operatiivmälu pesade arv saab suurendada 65 536-ni.

---

[32] Veidi üksikasjalikumalt tutvustab seda arvutit ja tema kasutamist väljaanne: Ü. Kaasik, K. Ääremaa jt. Elektronarvuti «Minsk-32» (õppevahend). TRÜ rotaprint, 1970.


0÷7
0÷7
0÷7
0÷7
0÷7
0÷7
0÷7
Minsk-20
Minsk-32
Trummel nr.0
Trummel nr.7
Seade nr.0
Seade nr.7
Seade nr.0
Seade nr.5
Seade nr.6
Seade nr.7
Kommu-taator
Kommu-taator
Kommu-taator
TR
SK
VK
SL
VL
VL
Kommutaator
Magnettrumlid
Magnetkettad
Magnetlindi seadmed
EA
Register 2 (37 kohta)
Summaator (37 kohta)
Register 1 (37 kohta)
AS
Kiired kanalid
Aeglased kanalid
Sümboli register (8 kohta)
VS
OM
16 384 pesa
16 384 pesa
16 384 pesa
16 384 pesa
Arvutiregister (38 kohta)
K
7 kohta
7 kohta
7 kohta
7 kohta
7 kohta
2 kohta
K
7 kohta
KJS
KJ
Kirjutus-masin

JOON. 61


a) Fikseeritud komaga kahendarvud
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
Arvu märk
Arvu murdosa numbrid
b) Liikuva komaga kahendarvud
Arvu järgu märk
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
Mantissi märk
Mantissi murdosa numbrid
Vaba koht
Arvu järk
c) Fikseeritud komaga kümnendarvud
1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
Arvu märk
Arvu murdosa numbrite kahend–kümnendkoodid
d) Kümnendsüsteemi täisarvud
1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
Arvu märk
Arvu täisosa numbrite kahend-kümnendkoodid
e) Tekst
0 1 2 3 4 5
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
Sümboli kood
Sümboli kood
Sümboli kood
Sümboli kood
Sümboli kood
Sümboli koodi 2 kohta

JOON. 62

Pesa 38-st kohast kasutatakse arvu numbrikohtade salvestamiseks 37 (järjekorranumbrid $0 \div 36$), 38-ndale kohale kirjutatakse (arvu salvestamisel pesasse) nn. kontrollnumber paarsuse kontrolliks. Kontrollnumbrit kasutades määrab arvuti, kas arvud on pesades moondumatult säilinud või mitte. Informatsiooni vahetamisel kontrollnumbrit arvuti ühest seadmest teise ümber ei salvestata.

Arvuti seadmetes on võimalik salvestada ja operatsioone sooritada ainult niisuguste arvudega x, mis rahuldavad järgmisi tingimusi:

a) fikseeritud komaga kahendarvud:

$$2^{-36} \leqslant |x| \leqslant 1 - 2^{-36} \quad \text{või} \quad x = 0;$$

b) liikuva komaga kahendarvud:

$$2^{-91} \leqslant |x| \leqslant (1 - 2^{-28}) \cdot 2^{63} \quad \text{või} \quad x = 0;$$

c) fikseeritud komaga kümnendarvud:

$$10^{-9} \leqslant |x| \leqslant 1 - 10^{-9} \quad \text{või} \quad x = 0;$$

d) kümnendsüsteemi täisarvud:

$$0 \leqslant |x| \leqslant 10^{9} - 1.$$

Pesa kahendkohti kasutatakse informatsiooni salvestamiseks joonisel 62 näidatud viisil.

2. Samaaegselt töötades võivad arvuti «Minsk-32» teatud seadmed sooritada erinevaid operatsioone. Näiteks ajal, mil aritmeetilises seadmes sooritatakse tehteid ühe ülesande lahendamiseks, on informatsiooni vahetamise seadme vahendusel võimalik perfolindilt (või -kaartidelt) salvestada operatiivmällu mingi teise ülesande programmi ja algandmeid, trükkida operatiivmälu pesadest teksti ning arve jne. Et arvuti üksikuid seadmeid saaks intensiivsemalt koormata, võimaldab tema konstruktsioon vaheldumisi täita kuni 12 erinevat programmi (või programmilõiku), millest 8 on nn. teenistuslikud programmid ja moodustavad nn. dispetšerprogrammi. Seega võib arvuti «Minsk-32» korraga lahendada kuni nelja iseseisvat ülesannet, kusjuures tema operatiivmälus on kuni 12 erinevat programmi.

Taseme number	0	1	2	3	4
Juhtpesade aadressid	$0 \div 7$	$10 \div 17$	$20 \div 27$	$30 \div 37$	$40 \div 47$

Tabel 37

Taseme number	Tasemesse kuuluva programmi nimetus
0	Aritmeetilise seadme, operatiivmälu ja keskse juhtimisseadme vigade analüüsimise programm
1	Informatsiooni vahetamise seadme vigade analüüsimise programm
2	Välisseadmete vigade analüüsimise programm
3	Programm, mille vahendusel operaator juhib ja kontrollib arvuti tööd
4	Programm arvuti vabade seadmete väljaselgitamiseks
5	Välisseadmete poole pöördumise programm
6	Ekstrakoodide töötlemise programm
7, 10, 11, 12	Tööprogrammid (s. t. programmid, mille täitmisel arvuti lahendab mingi iseseisva ülesande)
13	Programm arvuti koormamiseks, kui lahendamiseks pole antud ülesandeid

Operatiivmällu salvestatud programmid on jaotatud 12 tasemesse. Tasemetes kaheksandnumbritega $0 \div 6$ ja 13 paiknevad dispetšerisse kuuluvad programmid, tasemetesse numbritega $7 \div 12$ paigutab dispetšerprogramm täitmisele kuuluvad tööprogrammid (vt. tabel 37). Arvuti operatiivmälu on jaotatud kaheks piirkonnaks. Esimese piirkonna (nn. juhtpiirkond) moodustavad pesad aadressidega $0 \div 377$, teise (nn. tööpiirkond) aga operatiivmälu põhibloki pesad $400 \div 37777$ ja kõik mälu kolme lisabloki pesad (s. t. pesad aadressidega $40000 \div 177777$).

Mälu juhtpiirkonnast on igale tasemele eraldatud 8 pesa (vt. tabel 38), milles säilitatakse vastava programmi täitmiseks vajalikku informatsiooni. Tasemele juhtpiirkonnast eraldatud pesi nimetatakse juhtpesadeks ja neile on omistatud tasemesisesed järjekorranumbrid (nn. juhtpesa numbrid) $0 \div 7$.

Tabel 38

5	6	7	10	11	12	13
50 ÷ 57	60 ÷ 67	70 ÷ 77	100 ÷ 107	110 ÷ 117	120 ÷ 127	130 ÷ 137

Programmi koostamisel jaotatakse ülesande lahendamiseks vajalik informatsioon (programmi käsud, ülesande algandmed, arvutustulemused) neljaks massiiviks, milledele antakse järjekorranumbrid alates nullist kuni kolmeni. (Massiivide järjekorranumbreid tähistame edaspidi sümbolitega δ_1 ja δ_2.)

Ühte massiivi kuuluvad arvud paigutatakse operatiivmälus järjestikuste aadressidega pesadesse, milledele programmeerimisel omistatakse kaheksandsüsteemis fiktiivsed järjekorranumbrid alates nullist (nn. suhtelised aadressid). Käskude kirjutamisel märgitaksegi nendes aadresside kohale just pesade suhtelised aadressid, kusjuures iga aadressi ees antakse selle massiivi järjekorranumber, mille element salvestatakse vastavasse pesasse. (Edaspidi tähistame pesade suhtelisi aadresse sümbolitega a_1 ja a_2.)

Taseme juhtpesade 0 ja 1 kohtadele $5 \div 20$ ja $21 \div 36$ kirjutatakse sellele tasemele vastava programmi nelja informatsioonimassiivi esimeste elementide salvestamiseks kasutatud pesade tõelised aadressid nn. baasaadressid). Täpsemalt, juhtpesa 0 kohtadele $5 \div 20$ salvestatakse massiivi nr. 1 baasaadress, sama juhtpesa kohtadele $21 \div 36$ massiivi nr. 0 baasaadress, juhtpesa 1 kohtadele $5 \div 20$ massiivi nr. 3 baasaadress ja selle pesa kohtadele $21 \div 36$ massiivi nr. 2 baasaadress. Juhtpesade 0 ja 1 kohtadel $0 \div 4$ peavad olema nullid.

Kasutades juhtpesadesse paigutatud baasaadresse, annab arvuti automaatselt kõikidele käskudele tõelised aadressid. Nimelt algab iga käsu täitmine sellega, et massiivinumbrite δ_1 ja δ_2 järgi võetakse juhtpesadest baasaadressid ja liidetakse need vastavalt täidetava käsu aadressidele.

Kõigile kaheteistkümnele tasemele vastavate programmide informatsioonimassiivid paigutatakse operatiivmälus tööpiirkonna pesadesse, kusjuures igale programmile eraldatakse sellest piirkonnast teatud kindel osa. Programmile tööpiirkonnast eraldatud mäluosa esimese ja viimase pesa tõelised aadressid kirjutatakse vastavalt taseme juhtpesa 3 kohtadele $21 \div 36$ ja $5 \div 20$. Käskudele, mis ei paikne juhtpesasse 3 paigutatud aadressidega määratud mäluosas, on arvude lugemine ning kirjutamine (samuti suunamine) selle mäluosa pesadesse blokeeritud.

Arvuti «Minsk-32» käske on võimalik täitmisel modifitseerida (täpselt samuti, nagu arvuti «Minsk-22» käske) indekspesadesse paigutatud arvude abil. Indekspesadena saab kasutada kõiki operatiivmälu tööpiirkonna pesi.

Programmeerimisel omistatakse kasutatud indekspesadele kaheksandsüsteemis järjekorranumbrid $1 \div 17$, mida nimetatakse indeksiteks ja tähistatakse tähega i. Taseme juhtpesa number 4 kohtadele $21 \div 36$ salvestatakse indekspesade baasaadress, s. o. arv, mis indekspesa tõelise aadressi saamiseks tuleb liita indeksile. Kui käsus indeksi kohale on kirjutatud null, siis seda käsku

täitmisel ei modifitseerita. Juhtpesa 4 kohtadel $0 \div 20$ peavad olema nullid.

Juhtpesa number 5 kohtadele $21 \div 36$ salvestatakse selle pesa aadress, milles paikneva käsu täitmisega alustatakse või jätkatakse (pärast ajutist katkestust) vaadeldava taseme programmi täitmist. Sama juhtpesa kohtadel $5 \div 17$ salvestatakse programmi täitmise ajutisel katkestamisel juhtimissignaalide jooksvad väärtused: enne antud taseme programmi täitmise jätkamist taastatakse nendel kohtadel oleva arvu järgi juhtimissignaalidel katkestamismomendil olnud väärtused. Juhtpesa 5 kohtadel $0 \div 4$, 12, 19, 20 peavad olema nullid.

Antud tasemele vastava programmi täitmise katkestamisel salvestatakse juhtpesasse 6 arvuti summaatoris olev arv; juhtpesa 7 kohtadel $14 \div 17$ aga säilitatakse selle taseme number, mille programmi täitmine katkestati siis, kui hakati täitma vaadeldava taseme programmi. Juhtpesa 2 ei kasutata.

Üleminek ühe taseme programmi täitmiselt teise taseme programmi täitmisele toimub kas vastava käsu või spetsiaalsete juhtimissignaalide toimel. Näiteks, kui mingis 6. $\div$ 13. taseme programmis täidetakse käsk koodiga «—67», siis selle tulemusena katkestatakse antud taseme programmi täitmine ja arvuti asub täitma 5. taseme programmi (s. o. välisseadmete poole pöördumise alamprogrammi).

Programmi täitmine katkestatakse automaatselt näiteks juhul, kui tema mingi käsu täitmisel töötati välja ületäitumissignaal $\varphi = 1$ ja ületäitumist esilekutsunud käsule ei järgne programmis vahetult käsk, mis annab signaalile φ väärtuse 0. Analoogiliselt signaaliga φ kutsuvad programmi täitmise automaatse katkestamise esile juhtimissignaalid *alamprogramm* ja *tundmatu kood*.

Arvuti «Minsk-32» kontrollib automaatselt oma mitmete seadmete tööd (näiteks arvude säilitamist operatiivmälu pesades ja magnetlintidel). Seadmete töös avastatud vigade (samuti mõnede programmeerimisvigade) korral väljatöötatud signaalid (nagu *mittenumber*, *vale kontrollnumber*, *pöördumine blokeeritud mäluosasse* jne.) katkestavad samuti programmi täitmise, kusjuures arvuti kas asub täitma dispetšerprogrammi mingit osa või peatub.

Arvuti «Minsk-32» igale välisseadmele on mälu juhtpiirkonnast eraldatud üks kindel pesa (nimetatakse juhtsõna registriks), kuhu selle seadme poole pöördumisel salvestatakse tema töötamiseks vajalik informatsioon. Mälu juhtpiirkonna pesadesse on võimalik informatsiooni kirjutada ainult spetsiaalsete käskude abil.

Informatsioonivahetus operatiivmälu ja välisseadmete vahel toimub sümbolite (s. t. nende 7-kohaliste kahendkoodide) kaupa; operatiivmälu ja aritmeetilise seadme vahel aga sõltuvalt sooritatavast operatsioonist kas üksikute sümbolite või 37-kohaliste kahendarvude kaupa.

Arvuti põhikomplekti kuulub kuus magnetlindi seadet, mida kasutatakse väliste mäluseadmetena. Magnetlintidele on võimalik salvestada kuni 33 miljonit sümbolit. Magnetlintidele paigutatud informatsioon on jaotatud tsoonidesse, kusjuures tsoonide pikkused (sümbolite arvu tsoonides) määrab programmeerija. Näiteks võib programmeerija kasutada tsoone, milles on üksainus sümbol, 5000 sümbolit jne. Informatsioon kirjutatakse magnetlindile (samuti loetakse sealt) tsoonide kaupa. Ühes sekundis kirjutab arvuti operatiivmälust magnetlindile (või vastupidi) kuni 30000 sümbolit.

3. Arvuti «Minsk-32» käsud on 37-kohalised kahendarvud, mis koosnevad järgmistest osadest:

1) operatsiooni kood α (7 kahendkohta, salvestatakse pesas kohtadele $0 \div 6$);
2) indeks i (4 kahendkohta, salvestatakse pesas kohtadele $7 \div 10$);
3) massiivinumber δ_1 (2 kahendkohta, salvestatakse pesas kohtadele 11 ja 12);
4) esimene aadress a_1 (11 kahendkohta, salvestatakse pesas kohtadele $13 \div 23$);
5) masiivinumber δ_2 (2 kahendkohta, salvestatakse pesas kohtadele 24 ja 25);
6) teine aadress a_2 (11 kahendkohta, salvestatakse pesas kohtadele $26 \div 36$).

Seega on arvuti «Minsk-32» käsud üldiselt järgmise kujuga:

$$\alpha i \delta_1 a_1 \delta_2 a_2.$$

Programmi kirjutatakse käsud 12-kohaliste kaheksandarvudena, kusjuures käsu vasakult esimest kahendkohta tõlgendatakse selle arvu märgina. Asjaolu, et käsu kõigile erinevatele osadele ei vasta tema erinevad kaheksandkohad, komplitseerib käskude kirjutamist. Et üle saada sellest raskusest, siis käskude kirjeldamisel kirjutatakse eraldi kaheksandarvudena nende kõik need osad, mis juba käsu kirjelduses tuleb konkreetselt (arvuliselt) anda. Kui käsu mingi osa (välja arvatud operatsioonikood) kõik kahenkohad peavad olema nullid, siis märgitakse käsu üldkujus selle osa asemele lihtsalt number 0.

Rakendusvõimaluste alusel jaotatakse arvuti «Minsk-32» käsud kolme klassi: tööprogrammides kasutatavateks, ainult teenistuslikes programmides kasutatavateks ja arvuti «Minsk-22M» käskudele vastavateks käskudeks.

Tabelis 39 on esitatud ainult esimesse klassi kuuluvate käskude kirjeldused, kusjuures kasutatakse järgmisi tähistusi ja sümboleid:

t — taseme number;
k — taseme juhtpesa number;
(δ_1) — massiivi δ_1 baasaadress;

(δ_2) — massiivi δ_2 baasaadress;
(δ) — indekspesade baasaadress;
I — käsus kasutatud indekspesa tõeline aadress, mis käsu täitmisel arvutatakse seosest $I = (\delta) + i$;
I_1 — indekspesa I kahendkohad $5 \div 20$;
I_2 — indekspesa I kahendkohad $21 \div 36$;
(I) — arv, mis on indekspesas I momendil, mil arvuti asub täitma vaadeldavat käsku:
(I_1) — indekspesa I kohtadel $5 \div 20$ paiknev konstant (kasutatakse käsu esimese aadressi modifitseerimisel);
(I_2) — indekspesa I kohtadel $21 \div 36$ paiknev konstant (kasutatakse käsu teise aadressi modifitseerimisel);
A_1 — pesa tõeline aadress, mis käsu täitmisel arvutatakse seosest:

$$A_1 = (\delta_1) + a_1 + (I_1);$$

A_2 — pesa tõeline aadress, mis käsu täitmisel arvutatakse seosest:

$$A_2 = (\delta_2) + a_2 + (I_2);$$

(A_1) — arv, mis on pesas A_1 momendil, mil arvuti asub täitma vaadeldavat käsku;
(A_2) — arv, mis on pesas aadressiga A_2 momendil, mil arvuti asub täitma vaadeldavat käsku;
(A_1^1) — pesa A_1 kohtadel $5 \div 20$ paiknev arv;
(A_1^2) — pesa A_1 kohtadel $21 \div 36$ paiknev arv;
(A_2^1) — pesa A_2 kohtadel $5 \div 20$ paiknev arv;
(A_2^2) — pesa A_2 kohtadel $21 \div 36$ paiknev arv;
N — taseme juhtpesa aadress, mis käsu täitmisel arvutatakse seosest

$$N = t \cdot 10 + K + (I_1);$$

(N) — arv, mis on juhtpesas N momendil, mil arvuti asub täitma vaadeldavat käsku;
(S_0) — arv, mis on saadud summaatorisse eelmise käsu täitmise tulemusena;
(S) — arv, mis saadakse summaatorisse vaadeldava käsu täitmise tulemusena;
r — operatsiooni tulemus;
m — operatsiooni tulemuse mantiss;
p — operatsiooni tulemuse järk.

Käsu nimetus	Käsu kuju	Tingimus, mil antakse signaal			Operatsiooni kirjeldus	Käsu täitmiseks kuluv aeg (μ sek)
		$\omega = 1$	$\varphi = 1$	$\psi = 1$		
1	2	3	4	5	6	7
Operatsioonid fikseeritud komaga kahendarvudega						
Liitmine	$+10i\delta_1a_1\delta_2a_2$ $+11i\delta_1a_1\delta_2a_2$ $+12i\delta_1a_1\delta_2a_2$ $+13i\delta_1a_100$	$(S) < 0$	$\lvert(A_2)+(A_1)\rvert \geqslant 1$	$(S) = 0$	$(S) = (A_2)+(A_1)$; $(S) \to A_2$ $(S) = (A_2)+(A_1)$ $(S) = (S_0)+(A_1)$; $(S) \to A_2$ $(S) = (S_0)+(A_1)$	$30 \div 35$ $25 \div 30$ $25 \div 30$ $15 \div 20$
Lahutamine	$+20i\delta_1a_1\delta_2a_2$ $+21i\delta_1a_1\delta_2a_2$ $+22i\delta_1a_1\delta_2a_2$ $+23i\delta_1a_100$	$(S) < 0$	$\lvert(A_2)-(A_1)\rvert \geqslant 1$	$(S) = 0$	$(S) = (A_2)-(A_1)$; $(S) \to A_2$ $(S) = (A_2)-(A_1)$ $(S) = (S_0)-(A_1)$; $(S) \to A_2$ $(S) = (S_0)-(A_1)$	$30 \div 35$ $25 \div 30$ $25 \div 30$ $15 \div 20$
Korrutamine	$+30i\delta_1a_1\delta_2a_2$ $+31i\delta_1a_1\delta_2a_2$ $+32i\delta_1a_1\delta_2a_2$ $+33i\delta_1a_100$	$(S) < 0$ või $(S) = -0$	—	$(S) = \pm 0$	$(S) = (A_2) \cdot (A_1)$; $(S) \to A_2$ $(S) = (A_2) \cdot (A_1)$ $(S) = (S_0) \cdot (A_1)$; $(S) \to A_2$ $(S) = (S_0) \cdot (A_1)$	$30 \div 125$ $25 \div 120$ $25 \div 120$ $15 \div 110$
Jagamine	$+40i\delta_1a_1\delta_2a_2$ $+41i\delta_1a_1\delta_2a_2$ $+42i\delta_1a_1\delta_2a_2$ $+43i\delta_1a_100$	$(S) < 0$ või $(S) = -0$	$\lvert(A_2)\rvert \geqslant \lvert(A_1)\rvert$ või $(A_1) = \pm 0$	$(S) = \pm 0$	$(S) = (A_2) : (A_1)$; $(S) \to A_2$ $(S) = (A_2) : (A_1)$ $(S) = (S_0) : (A_1)$; $(S) \to A_2$ $(S) = (S_0) : (A_1)$	$30 \div 280$ $25 \div 275$ $25 \div 275$ $15 \div 265$
Absoluutväärtuste lahutamine	$+50i\delta_1a_1\delta_2a_2$ $+51i\delta_1a_1\delta_2a_2$ $+52i\delta_1a_1\delta_2a_2$ $+53i\delta_1a_100$	$(S) < 0$	—	$(S) = 0$	$(S) = \lvert(A_2)\rvert - \lvert(A_1)\rvert$; $(S) \to A_2$ $(S) = \lvert(A_2)\rvert - \lvert(A_1)\rvert$ $(S) = \lvert(S_0)\rvert - \lvert(A_1)\rvert$; $(S) \to A_2$ $(S) = \lvert(S_0)\rvert - \lvert(A_1)\rvert$	$30 \div 35$ $25 \div 30$ $25 \div 30$ $15 \div 20$

Aritmeetiline nihutamine	$+64i\delta_1a_1\delta_2a_2$ $+65i\delta_1a_1\delta_2a_2$ $+66i\delta_1a_1\delta_2a_2$ $+67i\delta_1a_100$	$(S) < 0$ või $(S) = -0$	—	$(S) = \pm 0$	Pesa A_1 kohtadel $30 \div 36$ on arv n, mille märk paikneb kohal 30. Kui $n > 0$, siis nihutatakse kas summaatoris (käsud koodidega +66 ja +67) või pesas A_2 (käsud koodidega +64 ja +65) olevat arvu (ilma märgi kohata) n kahendkoha võrra vasakule, vastupidisel juhul aga paremale. Nihutamise tulemus jääb summaatorisse, käskude +64 ja +66 korral aga salvestatakse see ka pesasse A_2	$30 \div 95$ $25 \div 90$ $25 \div 90$ $15 \div 80$
Täisarvude korrutamine	$-70i\delta_1a_1\delta_2a_2$	$(S) < 0$	$\lvert(A_2)\cdot(A_1)\rvert > \lvert(S)\rvert \cdot 2^{-36}$	$(S) = \pm 0$	Arvutatakse summaatorisse korrutise $(A_2) \cdot (A_1)$ 36 parempoolset kahendkohta (tulemuse märk ühtib korrutise $(A_2) \cdot (A_1)$ märgiga).	$25 \div 120$
Jäägi eraldamine	$-71i\delta_1a_1\delta_2a_2$	$(S) < 0$	$(A_1) = \pm 0$ või $\lvert(A_2)\rvert \geqslant \lvert(A_1)\rvert$	$(S) = \pm 0$	Arvutatakse summaatorisse arvu (A_2) jagamisel arvuga (A_1) saadav jääk (jäägil on jagatava märk)	$25 \div 275$
Tsükliline liitmine	$-74i\delta_1a_1\delta_2a_2$	$(S) < 0$	—	$(S) = 0$	Pesades A_2 ja A_1 paiknevad kahendkoodid (ka märgikohtadel olevad numbrid) liidetakse ja summa salvestatakse pesasse A_2. Liitmisel märgi kohal tekkiv ülekanne liidetakse 36 kohale	$30 \div 35$
Normaliseerimine	$-75i\delta_1a_1\delta_2a_2$	$(S) < 0$ või $(S) = -0$	—	$(A_1) = \pm 0$	Arvu (A_1) (ilma märgikohata) nihutatakse vasakule seni, kui tema esimene nullist erinev kahendnumber jõuab kohale 1. Nihutatud kahendkohtade arv (koos märgiga miinus) kirjutatakse pesa A_2 kohtadele $30 \div 36$, nihutamise tulemus aga pesasse $A_2 + 1$	$30 \div 90$

1	2	3	4	5	6	7
Ühtede loendamine	$-76i\delta_1a_1\delta_2a_2$	—	—	$r=0$	Määratakse ühtede arv r arvu (A_1) kirjapildis; $r\cdot 2^{-36}\to A_2$	$25\div 90$
Operatsioonid liikuva komaga kahendarvudega						
Liitmine	$+14i\delta_1a_1\delta_2a_2$ $+15i\delta_1a_1\delta_2a_2$ $+16i\delta_1a_1\delta_2a_2$ $+17i\delta_1a_100$	$m<0$	$p>+63$	$m=0$ või $p<-63$	$(S)=(A_2)+(A_1);\ (S)\to A_2$ $(S)=(A_2)+(A_1)$ $(S)=(S_0)+(A_1);\ (S)\to A_2$ $(S)=(S_0)+(A_1)$	$30\div 150$ $25\div 145$ $25\div 145$ $15\div 135$
Lahutamine	$+24i\delta_1a_1\delta_2a_2$ $+25i\delta_1a_1\delta_2a_2$ $+26i\delta_1a_1\delta_2a_2$ $+27i\delta_1a_100$	$m<0$	$p>+63$	$m=0$ või $p<-63$	$(S)=(A_2)-(A_1);\ (S)\to A_2$ $(S)=(A_2)-(A_1)$ $(S)=(S_0)-(A_1);\ (S)\to A_2$ $(S)=(S_0)-(A_1)$	$30\div 150$ $25\div 145$ $25\div 145$ $15\div 135$
Korrutamine	$+34i\delta_1a_1\delta_2a_2$ $+35i\delta_1a_1\delta_2a_2$ $+36i\delta_1a_1\delta_2a_2$ $+37i\delta_1a_100$	$m<0$ või $m=-0$	$p>+63$	$m=\pm 0$ või $p<-63$	$(S)=(A_2)\cdot(A_1);\ (S)\to A_2$ $(S)=(A_2)\cdot(A_1)$ $(S)=(S_0)\cdot(A_1);\ (S)\to A_2$ $(S)=(S_0)\cdot(A_1)$	$30\div 175$ $25\div 170$ $25\div 170$ $15\div 160$
Jagamine	$+44i\delta_1a_1\delta_2a_2$ $+45i\delta_1a_1\delta_2a_2$ $+46i\delta_1a_1\delta_2a_2$ $+47i\delta_1a_100$	$m<0$ või $m=-0$	$p>+63$ või jagaja mantiss on võrdne nulliga	$m=\pm 0$ või $p<-63$	$(S)=(A_2):(A_1);\ (S)\to A_2$ $(S)=(A_2):(A_1)$ $(S)=(S_0):(A_1);\ (S)\to A_2$ $(S)=(S_0):(A_1)$	$30\div 235$ $25\div 230$ $25\div 230$ $15\div 220$
Absoluutväärtuste lahutamine	$+54i\delta_1a_1\delta_2a_2$ $+55i\delta_1a_1\delta_2a_2$ $+56i\delta_1a_1\delta_2a_2$ $+57i\delta_1a_100$	$m<0$	—	$m=0$ või $p<-63$	$(S)=\lvert(A_2)\rvert-\lvert(A_1)\rvert;\ (S)\to A_2$ $(S)=\lvert(A_2)\rvert-\lvert(A_1)\rvert$ $(S)=\lvert(S_0)\rvert-\lvert(A_1)\rvert;\ (S)\to A_2$ $(S)=\lvert(S_0)\rvert-\lvert(A_1)\rvert$	$30\div 150$ $25\div 145$ $25\div 145$ $15\div 135$
Järkude liitmine	$-72i\delta_1a_1\delta_2a_2$	$m<0$ või $m=-0$	$\lvert p\rvert>63$	$(S)=\pm 0$	Moodustatakse summaatorisse arv, mille mantiss võrdub arvu (A_2) mantissiga ja järk arvude (A_2) ja (A_1) järkude summaga, tulemus salvestatakse ka pesasse A_2	35

Järkude lahutamine	$-73i\delta_1 a_1 \delta_2 a_2$	$m < 0$ või $m = -0$	$\|p\| > 63$	$(S) = \pm 0$	Moodustatakse summaatorisse arv, mille mantiss võrdub arvu (A_2) mantissiga ja järk arvude (A_2) ja (A_1) järkude vahega, tulemus kirjutatakse ka pesasse A_2	35
Operatsioonid kümnendarvudega [1]						
Liitmine	$+01i\delta_1 a_1 00$	$(S) < 0$	$\|(S)\| \geqslant 1$	$(S) = 0$	$(S) = (S_0) + (A_1)$; tulemus saadakse kahend-kümnendkoodis	$20 \div 30$
Lahutamine	$+02i\delta_1 a_1 00$	$(S) < 0$	$\|(S)\| \geqslant 1$	$(S) = 0$	$(S) = (S_0) - (A_1)$; tulemus saadakse kahend-kümnendkoodis	$20 \div 30$
Korrutamine	$+03i\delta_1 a_1 00$	$(S) < 0$	—	$(S) = \pm 0$	$(S) = (S_0) \cdot (A_1)$; tulemuseks saadakse korrutise 9 vasakpoolset kümnendkohta	$20 \div 615$
Täisarvude korrutamine	$-77i\delta_1 a_1 00$	$(S) < 0$	$\|(S_0) \cdot (A_1)\| > \|(S)\|$	$(S) = \pm 0$	Summaatorisse arvutatakse kahend-kümnendkoodis korrutise $(S_0) \cdot (A_1)$ üheksa parempoolset kümnendkohta. Tulemusel on korrutise $(S_0) \cdot (A_1)$ märk	$20 \div 615$
Loogilised operatsioonid [2]						
Arvude võrdlemine	$+04i\delta_1 a_1 \delta_2 a_2$ $+05i\delta_1 a_1 \delta_2 a_2$ $+06i\delta_1 a_1 \delta_2 a_2$ $+07i\delta_1 a_1 00$	$(S) < 0$ või $(S) = -0$		$(S) = \pm 0$	$(S) = (A_2) \simeq (A_1)$; $(S) \to A_2$ $(S) = (A_2) \simeq (A_1)$ $(S) = (S_0) \simeq (A_1)$; $(S) \to A_2$ $(S) = (S_0) \simeq (A_1)$	30 25 25 15

[1] Kui kas või ühe operandi kirjapilt ei ole arvu kahend-kümnendkood, siis on käskude «+01», «+02», «+03» ja «—77» täitmise ainsaks tulemuseks signaali *mittenumber* väljatöötamine.

[2] Loogiliste ja ümbersalvestamise operatsioonide sooritamisel signaali $\varphi = 1$ välja ei töötata, mille tõttu tabelist on neljas veerg ära jäetud.

1	2	3	4	5	6	7
Formeerimine	$+74i\delta_1a_1\delta_2a_2$ $+75i\delta_1a_1\delta_2a_2$ $+76i\delta_1a_1\delta_2a_2$ $+77i\delta_1a_100$	$(S)<0$ või $(S)=-0$		$(S)=\pm0$	$(S)=(A_2)\vee(A_1)$; $(S)\to A_2$ $(S)=(A_2)\vee(A_1)$ $(S)=(S_0)\vee(A_1)$; $(S)\to A_2$ $(S)=(S_0)\vee(A_1)$	30 25 25 15
Osa eraldamine	$+70i\delta_1a_1\delta_2a_2$ $+71i\delta_1a_1\delta_2a_2$ $+72i\delta_1a_1\delta_2a_2$ $+73i\delta_1a_100$	$(S)<0$ või $(S)=-0$		$(S)=\pm0$	$(S)=(A_2)\wedge(A_1)$; $(S)\to A_2$ $(S)=(A_2)\wedge(A_1)$ $(S)=(S_0)\wedge(A_1)$; $(S)\to A_2$ $(S)=(S_0)\wedge(A_1)$	30 25 25 15
Loogiline nihutamine	$+60i\delta_1a_1\delta_2a_2$ $+61i\delta_1a_1\delta_2a_2$ $+62i\delta_1a_1\delta_2a_2$ $+63i\delta_1a_100$	$(S)<0$ või $(S)=-0$		$(S)=\pm0$	Pesa A_1 kohtadel $30\div36$ on arv n, mille märk on kohal 30. Kui $n>0$, siis nihutatakse kas summaatoris (käsud koodidega +62 ja +63) või pesas A_2 (käsud koodidega +60 ja +61) olevat arvu (koos märgikohaga) n kahendkoha võrra vasakule, vastupidisel juhul aga paremale. Nihutamise tulemus jääb summaatorisse, käskude +60 ja +62 korral kirjutatakse see aga ka pesasse A_2	$30\div95$ $25\div90$ $25\div90$ $15\div80$
Ümbersalvestamise operatsioonid						
Salvestamine	$-10i\delta_1a_1\delta_2a_2$	$(S)<0$ või $(S)=-0$		$(S)=\pm0$	$(S)=(A_1)$; $(S)\to A_2$	25
Vastandarvu salvestamine	$-11i\delta_1a_1\delta_2a_2$	$(S)<0$ või $(S)=-0$		$(S)=\pm0$	$(S)=-(A_1)$; $(S)\to A_2$	25
Mooduli salvestamine	$-12i\delta_1a_1\delta_2a_2$	—		$(S)=0$	$(S)=\lvert(A_1)\rvert$; $(S)\to A_2$	25

Märgi omistamine	$-14i\delta_1 a_1 \delta_2 a_2$	$(S) < 0$ või $(S) = -0$		$(S) = \pm 0$	$(S) = \text{sign}\,[(A_1)] \cdot (A_2)$; $(S) \to A_2$	30
Vahetamine	$-15i\delta_1 a_1 \delta_2 a_2$	$(S) < 0$ või $(S) = -0$		$(S) = \pm 0$	$(A_2) \to A_1$; $(A_1) \to A_2$; $(S) = (A_1)$	35
Järgu omistamine	$-16i\delta_1 a_1 \delta_2 a_2$	$m < 0$ või $(S) = -0$		$(S) = \pm 0$	Moodustatakse liikuva komaga arv, mille mantissiks on arvu (A_2) mantiss ja järguks arvu (A_1) järk. Tulemus saadakse summaatorisse ja pesasse A_2	30
Kordamiste arvu salvestamine [1]	$-17i00\delta_2 a_2$	—		—	Pesa A_2 kohtadel $24 \div 36$ paiknev arv saadetakse juhtimisseadmes paiknevasse kordamiste loendajasse. Kui soovitakse, et arvuti täidab vaadeldavale käsule järgnevat käsku n korda, tuleb kordamiste loendajasse saata arv $(n-1) \cdot 2^{-36}$	15
Suunamisoperatsioonid						
Suunamine koos salvestamisega	$-30i\delta_1 a_1 \delta_2 a_2$	Säilib	—	Säilib	$(S_0) \to A_2$; juhtimine antakse pesas A_1 paiknevale käsule; $(S) = (S_0)$	20
Suunamine koos lugemisega	$-36i\delta_1 a_1 \delta_2 a_2$	$(S) < 0$	—	$(S) = \pm 0$	$(S) = (A_2)$; juhtimine antakse pesas A_1 paiknevale käsule	20

[1] Kordamiste loendajat kasutades on võimalik organiseerida ainult teatud kindlate käskude tsüklilist (nn. grupirežiimis) täitmist.

1	2	3	4	5	6	7
Suunamine ω järgi	$-32i\delta_1 a_1 \delta_2 a_2$	Säilib	—	Säilib	Kui $\omega = 1$, siis antakse juhtimine pesas A_2 paiknevale käsule, vastupidisel juhul aga käsule pesast A_1; $(S) = \overline{+}(S_0)$	10
Suunamine ψ järgi	$-34i\delta_1 a_1 \delta_2 a_2$	Säilib	—	Säilib	Kui $\psi = 1$, siis antakse juhtimine pesas A_2 paiknevale käsule, vastupidisel juhul aga käsule pesast A_1; $(S) = (S_0)$	10
Suunamine võrdlemisega	$-37i\delta_1 a_1 \delta_2 a_2$	$(S) < 0$ või $(S) = -0$	—	$(S) = \pm 0$	$(S) = (S_0) \simeq (A_2)$. Kui $(S) = 0$, siis antakse juhtimine järgmises pesas paiknevale käsule, vastupidisel juhul aga täidetakse käsk pesast A_1	$20 \div 25$
Suunamine loendaja järgi	$-22i\delta_1 a_1 \delta_2 a_2$	—	—	$(S) = 0$	Positiivsest arvust (A_2) lahutatakse 2^{-36}. Kui tulemus on negatiivne, antakse juhtimine järgmises pesas paiknevale käsule ja $(S) = +0$. Kui $(A_2) - 2^{-36} \geqslant 0$, siis antakse juhtimine käsule pesast A_1; $(S) = (A_2) - 2^{-36}$ ja $(S) \to A_2$	$20 \div 30$
Suunamine φ järgi [1]	$-33i\ 0\ 2000\ \delta_2 a_2$	—	Kustutatakse	—	Kui $\varphi = 0$, antakse juhtimine käsule pesast A_2, vastupidisel juhul täidetakse järgmine käsk	10
	$-33i\ 2\ 2000\ \delta_2 a_2$	—	Kustutatakse	—	Kui $\varphi = 1$, antakse juhtimine käsule pesast A_2, vastupidisel juhul täidetakse järgmine käsk	

[1] Kui käsule, mille täitmisel kas antakse signaal $\varphi = 1$ või omistatakse tunnustele *alamprogramm* või *tundmatu kood* väärtus 1, ei järgne vastavat suunamiskäsku, siis katkestatakse selle taseme programmi täitmine (toimub automaatne üleminek uuele tasemele).

Suunamine alamprogrammi	$-31i\delta_1 a_1 0\ l$	Säilib	—	Säilib	Formeeritakse aadress $A = (A_1^2)$. Kui $(A) \neq +0$, siis väljastatakse signaal *alamprogramm* ning toimub üleminek taseme 0 programmi täitmisele. Kui $(A) = +0$, siis kirjutatakse juhtimissignaalide väärtused vastavalt pesa A kohtadele $13 \div 17$; arv $k+1$ (k tähistab vaadeldava käsu pesa aadressi) pesa A kohtadele $21 \div 36$; arvud juhtpesadest 0 ja 1 pesadesse $A+1$ ja $A+2$; arvud pesadest A_1 ja A_1+1 juhtpesadesse 0 ja 1, ning arvud pesadest $k+1 \div k+l$ vastavalt pesadesse $A+3 \div A+l+2$. Juhtimine antakse pesas $A+l+3$ paiknevale käsule. $(S) = (S_0)$	$65+10\,l$
Väljumine alamprogrammist	$-21i\delta_1 a_1 0\ l$	—	—	—	Arvu (A_1^1) järgi taastatakse juhtimissignaalide väärtused, mis neil olid sisenemisel alamprogrammi. Arvud (A_1+1) ja (A_1+2) kirjutatakse vastavalt juhtpesadesse 0 ja 1. Juhtimine antakse pesas aadressiga $(A_1^2)+l$ paiknevale käsule, kusjuures pesasse A_1 kirjutatakse arv $+0$. $(S) = (S_0)$	40
Ülejäänud operatsioonid.						
Tehted indekspesades paiknevate arvudega [1]	$-20i\ 0\ 400\delta_2 a_2$	—	—	$r = +0$	$(I_1) + (A_2^1) \to I_1$; $(I_2) + (A_2^2) \to I_2$. Ülekanded kohtadelt 5 ja 21 on blokeeritud	30

[1] Käsu «—20» täitmisel arvutatakse aadress A_2 seosest: $A_2 = a_2 + (\delta_2)$.

1	2	3	4	5	6	7
Tehted indekspesades paiknevate arvudega	$-20i\ 0\ 1000\delta_2 a_2$	—	—	$r = +0$	$(I_1) - (A_2^1) \to I_1$; $(I_2) - (A_2^2) \to I_2$. Kui lahutatav on vähendatavast suurem, saadakse tulemus täiendkoodis; vastupidisel juhul aga otsekoodis	30
	$-20i\ 0\ 1400\delta_2 a_2$	—	—		(I_1) x $(A_2^2) \to I_1$; (I_2) x $(A_2^2) \to I_2$	$25 \div 120$
	$-20i\ 0\ 2400\delta_2 a_2$	—	—		$(I) \vee (A_2) \to I$	25
	$-20i\ 0\ 3000\delta_2 a_2$	—	—		$(I) \simeq (A_2) \to I$	25
	$-20i\ 0\ 3400\delta_2 a_2$	—	—		$(I) \wedge (A_2) \to I$	25
	$-20i\ 1\ 0000\delta_2 a_2$	—	—		$(S_0) \to I$	15
	$-20i\ 1\ 0400\delta_2 a_2$	—	—		$(A_2) \to I$	20
	$-20i\ 1\ 1000\delta_2 a_2$	—	—		$(I) \to A_2$	20
	$-20i\ 1\ 1400\delta_2 a_2$	—	—		$(I) \to S$	15
	$-20i\ 1\ 2000\delta_2 a_2$	—	—		$(A_2^1) \to I_1$; pesa I ülejäänud kohtadel säilivad seal varem olnud numbrid	25
	$-20i\ 1\ 2400\delta_2 a_2$	—	—		$(A_2^2) \to I_1$; pesa I ülejäänud kohtadel säilivad seal varem olnud numbrid	25
	$-20i\ 1\ 3000\delta_2 a_2$	—	—		$(A_2^2) \to I_2$; pesa I ülejäänud kohtadel säilivad seal varem olnud numbrid	25
	$-20i\ 1\ 3400\delta_2 a_2$	—	—		$(A_2^1) \to I_2$; pesa I ülejäänud kohtadel säilivad seal varem olnud numbrid	25
Sümboli toomine summaatorisse	$-01i\ 0\ n\delta_2 a_2$	$(S) < 0$	—	$(S) = \pm 0$	Sümbol number $J = n + (I_1)$ kirjutatakse pesast A_2 summaatori kohtadele $0 \div 6$. $J = 0, 1, 2, 3$ või 4	20

Sümboli salvestamine pesasse	$-02i\,0\,n\delta_2 a_2$	$(A_2) < 0$	—	$(A_2) = \pm 0$	Summaatori kohtadel $0 \div 6$ paikneva sümboli kood kirjutatakse pesasse A_2 J-nda sümboli kohale ($J = 0, 1, 2, 3, 4$), pesa A_2 ülejäänud kohtadel säilivad seal varem olnud numbrid	25
Sümboli numbri modifitseerimine	$-20i\,0\,0\delta_2 a_2$	—	—	$r = +0$	$(I_1) + (A_2^1) - 5 \rightarrow I_1$ ja $(I_2) + (A_2^2) \rightarrow I_2$. Liitmisel 18. kohal tekkiv ülekanne liidetakse summale 36. kohal, ülekanne 21. kohalt on blokeeritud. $(I_1) \leqslant 4$ ja $(A_2^1) \leqslant 4$	30
Kirjutada pesasse	$-06i\delta_1 a_1 0 a_2$	Säilib	—	Säilib	$(S_0) \rightarrow A$, kus $A = a_2 + (I_2) + (A_1^2)$; $(S) = (S_0)$	25
Lugeda pesast	$-05i\delta_1 a_1 0 a_2$	$(S) < 0$	—	$(S) = \pm 0$	$(A) \rightarrow S$, kus $A = a_2 + (I_2) + (A_1^2)$	25
Täita käsk	$-25i\delta_1 a_1 0 a_2$	—	—	—	Arvutatakse aadress $A = a_2 + (I_2) + (A_1^2)$ ja täidetakse pesas A paiknev käsk. Programmi täitmist jätkatakse nii, nagu pesas A olev käsk oleks paiknenud programmis käsu *täita käsk* kohal	$25 + t_1$ [1]
Suunamine koos baasaadressi muutmisega	$-26i\delta_1 a_1 \delta_2 a_2$	Säilib	—	Säilib	Arv (A_1^2) kirjutatakse juhtpesasse massiivi δ_2 baasaadressi kohale ja juhtimine antakse pesasse A, kus $A = a_2 + (A_1^2) + (I_2)$. $(S) = (S_0)$	25

[1] t_1 tähistab pesas A oleva käsu täitmiseks kuluvat aega.

1	2	3	4	5	6	7
Operatsioonid juhtpesades paiknevate arvudega	$-17i\,0\,400 + k\delta_2 a_2$	$r < 0$ või $r = -0$	$\lvert r\rvert \geqslant 1$	$r = \pm 0$	$(N) + (A_2) \to N$	25 ÷ 30
	$-17i\,0\,1000 + k\,\delta_2 a_2$				$(N) - (A_2) \to N$	25 ÷ 30
	$-17i\,0\,2400 + k\delta_2 a_2$				$(N) \vee (A_2) \to N$	25
	$-17i\,0\,3000 + k\delta_2 a_2$				$(N) \simeq (A_2) \to N$	25
	$-17i\,0\,3400 + k\delta_2 a_2$				$(N) \wedge (A_2) \to N$	25
	$-17i\,1\,k\delta_2 a_2$				$(S_0) \to N$	15
	$-17i\,1\,0400 + k\delta_2 a_2$				$(A_2) \to N$	20
	$-17i\,1\,1000 + k\delta_2 a_2$				$(N) \to A_2$	20
	$-17i\,1\,1400 + k\delta_2 a_2$				$(N) \to S$	15
Operatsiooni puudumine	$+00i\delta_1 a_1 \delta_2 a_2$	Säilib	—	Säilib	$(S) = (S_0)$. Selle käsu toimel mingeid arvutusi ei teostata	10

Märkus. Tööprogrammides on lubatud kasutada veel käske, milles operatsioonikoodideks on nn. ekstrakoodid «—00»; «—03»; «—24»; «—40»; «—64». Nende käskude toimel arvuti katkestab vaadeldava taseme programmi täitmise ja asub täitma tasemele 6 vastavat programmi. Käsu, mille operatsioonikoodiks on kas «—65» või «—67», täitmise tulemusena arvuti katkestab vaadeldava taseme programmi täitmise ja asub täitma tasemele 5 vastavat programmi.

KIRJANDUS

1. Автоматизация программирования (Сборник переводов под ред. А. П. Ершова). Гос. изд. физ.-матем. литературы. Москва, 1961.
2. Agur, U. Mõtlevad masinad. Eesti Riiklik Kirjastus. Tallinn, 1961.
3. Айнберг В. Д. и др. Программирование для электронных вычислительных машин типа «Урал». Изд. «Наука». Москва, 1966.
4. Алгоритмический язык АЛГОЛ-60 (пересмотренное сообщение). Изд. «Мир». Москва, 1965.
5. Альбом средств вычислительной техники. Казань, 1966.
6. Балуев А. Н. и др. Сборник упражнений по АЛГОЛ-60. Изд. Ленинградского университета, 1967.
7. Богомолов А. М., Коробов Б. В. Программирование для ЭВМ «Урал-2» и «Урал-4». Изд. Саратовского университета, 1965.
8. Боттенбрух Г. Структура АЛГОЛ-60 и его использование. Изд. иностранной литературы. Москва, 1963.
9. Бусленко Н. П., Шрейдер Ю. А. Метод статистических испытаний (Монте-Карло) и его реализация на цифровых вычислительных машинах. Гос. изд. физ.-матем. литературы. Москва, 1961.
10. Бусленко Н. П. и др. Метод статистических испытаний (метод Монте-Карло). Гос. изд. физ.-матем. литературы. Москва, 1962.
11. Бут Э., Бут К. Автоматические цифровые машины. Гос. изд. физ.-матем. литературы. Москва, 1959.
12. Бухтияров А. М. и др. Сборник задач по программированию. Изд. «Наука». Москва, 1965.
13. Гавриленко Е. Т. Автоматизация программирования и стандартные программы для вычислительных машин «Урал-2», «Урал-3», «Урал-4». Изд. «Машиностроение». Москва, 1966.
14. Гнеденко Б. В. и др. Элементы программирования. Гос. изд. физ.-матем. литературы. Москва, 1961.
15. Голенко Д. И. Моделирование и статистический анализ псевдослучайных чисел на электронных вычислительных машинах. Изд. «Наука». Москва, 1965.
16. Гутер Р. С. и др. Программирование и вычислительная математика. Изд. «Наука». Москва, 1965.
17. Еврейнов Э. В., Косарев Ю. Г. Однородные универсальные вычислительные системы высокой производительности. Изд. «Наука». Новосибирск, 1966.
18. Жоголев Е. А., Трифонов Н. П. Курс программирования. Изд. «Наука», Москва, 1964.
19. Илзиня И. Программирование для двухадресных цифровых вычислительных машин. Изд. АН Латв. ССР. Рига, 1962.
20. Kaasik, Ü., Salum, H., Sinisoo, M. Elektronarvutusmasinad. Eesti Riiklik Kirjastus. Tallinn, 1960.

21. Карр Дж. Лекции по программированию. Изд. иностранной литературы. Москва, 1963.
22. Китов А. И., Криницкий Н. А. Электронные цифровые машины и програмирование. Гос. изд. физ.-матем. литературы. Москва, 1963.
23. Китов А. И. Программирование информационно-логических задач. Изд. «Советское радио». Москва, 1967.
24. Котли М., Ханко П. Руководство по алгоритмическому языку MALGOL. Изд. АН ЭССР. Таллин, 1966.
25. Котли, М., Вийль, А., Рахенди, М. Система автоматического программирования для ЭВМ «Минск-22». ЭК «Бит», Таллин, 1969.
26. Криницкий Н. А. и др. Программирование. Изд. «Наука». Москва, 1966.
27. Kull, I. Arvutid ja programmeerimine. TRÜ. Tartu, 1965.
28. Лавров С. С. Универсальный язык программирования. Изд. «Наука». Москва, 1964.
29. Мак-Кракен Д. Д. Программирование для цифровых вычислительных машин. Изд. иностранной литературы. Москва, 1960.
30. Мак-Кракен Дж. Программирование на АЛГОЛе. Изд. «Мир». Москва, 1964.
31. Никитюк Н. М. Свет, кванты и вычислительная техника. Изд. «Знание». Москва, 1967.
32. Паперное А. А. Логические основы цифровых машин и программирования. Изд. «Наука». Москва, 1965.
33. Ренделл Б., Рассел Л. Реализация АЛГОЛа-60. Изд. «Мир». Москва, 1967.
34. Савинков В. М. Программирование для ЭЦВМ «Минск-22». Изд. «Статистика». Москва, 1969.
35. Шураков, В. В., Каплинский, В. В. Программирование учётно-статистических задач для ЭЦВМ «Минск-22». «Изд. «Наука». Москва 1967.
36. Уилкс М. В. Автоматические цифровые вычислительные машины. Гос. союзное изд. судостроительной промышленности. Ленинград, 1960.

REGISTER

Aadress 15, 77
Abdank-Abakanovitš 19
abisignaal 16, 335
Aiken 21
alamprogramm 177, 326
alamprogrammide biblioteek 179
alamprogrammi kirjeldus 326
algandmed 121
algkaart 194
ALGOL-60 271
ALGOL-programm 275, 296
algoritm 14
algoritmide teooria 14
algoritmiline keel 270
alumine raja 286
analüütiline arvuti 18
aperioodsuse piirkond 255
aritmeetiline avaldis 290, 324
— lihtavaldis 290
— nihutamine 102, 110
— operaator 213
— seade 13, 106
— viga 248
aritmeetilise tehte märk 279
aritmeetilised tehted 87
arv 281, 323
arvude summaator 56
arvusüsteemi alus 31
arvuti 9
— null 54, 75
arvutustehniline seade 10
automaatne programmeerimine 271
avaldis 289
avaldise tüüp 292

Baasaadress 334
Babbage 20
Bazilevski 23
biblioteegimassiiv 180
bloki algus 297
— indeks 107
blokk 275, 296
— -skeem 123, 212
— -skeemide keel 270
Boole'i konstant 68
Bruk 23
Bush 19

(massiivi) dimensioon 286
disjunktsioon 69
diskreetse tegevusega arvuti 11
dispetšerprogramm 332, 333, 335
duaalsus 72

Eckert 21
eitus 69
ekvivalents 69
eraldaja 278
etalonkeel 272
etalonpakk 194

Fibonacci jada 256
fikseeritud komaga arvuti 49
fiktiivne muutuja 72
formaalne parameeter 306
formaalsete parameetrite loetelu 306
— — osa 306
formeerimine 102
formeeriv programm 185
funktsioon 287, 310

Graafiliselt võrdsed sõnad 28
Gunter 19

Harilik iteratsioonimeetod 145
Hollerith 18
Horneri skeem 152
hübriidne arvuti 11

Identifikaator 280, 323
implikatsioon 69, 293
indeks 334
ideksavaldis 295
indeksite loetelu 284
indeksitega muutuja 283
indekspesa 108, 160, 334
indekspesade baasaadress 334
induktiivne definitsioon 28

informatsiooni vahetamise seade 332
interpreteeriv programm 180, 209
invariantne massiiv 180
iteratsioonitsükkel 145

Jacquard 18
jaotusfunktsioon 253
juhtimispult 13, 83
juhtimisseade 13, 83
juhtpesa 333
juhtsõna register 335
juhuslike arvude andur 257
— — tabel 255
juhuslikud arvud 253
— vead 227
järgnevusmärk 279
järgusummaator 76
järk 52, 281

Kaheaadressiline arvuti 15
kaheksandarv 323
kaheksandsüsteem 32, 39
kahend-kümnendkood 52
kahendmuutuja 68
kahendsüsteem 32, 39
katkestus 335
katkestuspesa 113
kinnine alamprogramm 178
kirjeldaja 278
kirjeldus 275, 297
kolmeaadressiline arvuti 15
kolmik 77
kommentaar 279
kompileeriv programm 180
konjunktsioon 69
konkreetne esitus 272
konstantne aadress 177
kontrolli meetodid 227
kontrollimise puhvermälu 120
kontrolliv blokk 124
kontrollsumma 186, 232
koostav programm 180, 197
kordamist juhtiv blokk 144
korrutamistüüpi tehtemärk 291
Krõlov 19
kui-operaator 300
kuueteistkümmendsüsteem 32, 40
käsk 15, 107, 336
käskude keel 269
käskude loendaja 83
— register 83
kümnend-kahendkood 52
kümnendarv 281
kümnendsüsteem 32

Lahtine alamprogramm 178
— rida 282
Lebedev 22
Leibniz 17
lihtmurd 281
lihtmuutuja 283
liikuva komaga arvuti 52
liitkäsk 86
liitmistüüpi tehtemärk 291
liitoperaator 296
liitoperaatori lõpp 297
loendaja 156
— samm 156
lokaliseeritud muutuja 285
loogiline avaldis 70, 293, 325
— konstant 68
— lihtavaldis 293
— liige 293
— nihutamine 102, 110
— operaator 213
— primaaravaldis 293
— sekundaaravaldis 293
— skaala 170
— tegur 293
— valem 70
— väärtus 68
loogilise tehte märk 279
loogilised operatsioonid 101
lõputunnus 188
lähteinformatsioon 12
lühike pesa 79
lüliti 295
— identifikaator 295
lülitikirjeldus 299
lülituste loetelu 299

Magnetlint 81, 336
magnettrummel 81
mantiss 52
MALGOL 321
massiiv 180, 181, 284
massiivi identifikaator 284
— informatsiooniblokk 185
— segment 286
massiivide biblioteek 189
— loetelu 286
massiivikirjeldus 286, 324
matemaatiline keel 269
Mauchly 21
meetodi viga 247
mitmekohaline kahendarvude summaator 57
mitteekvivalents 69
modifitseeritud pöördkood 64
— täiendkood 64
Monte-Carlo meetod 253
Morland 17
muutuja 283, 324
muutumisrajade loetelu 286
— — paar 286
muutuvandmete tabel 198

mälu 12
mälu juhtpiirkond 333
mälu tööpiirkond 333
märgend 294
märgendamata blokk 296
— liitoperaator 297
— põhioperaator 297
märgendamine 296
märgi omistamine 101
märgita arv 281
— täisarv 281

Napier 17
naturaalarv 28
naturaalarvu tähis 30
nelik 78
neljandsüsteem 32
Neumann 22
nihutamisoperatsioon 102
nimeline avaldis 294, 325
— lihtavaldis 295
normaliseeritud arv 53
— null 106
number 278
numbriline arvuti 11

Odhner 18
omistamise sümbol 217, 298
omistamisoperaator 297, 325
operaator 213, 275, 296, 297, 325
operaatorskeem 214
— sulud 296
operand 77, 292
operatiivmälu 13, 105, 329
operatiivne salvestusseade 13
operatsiooni kood 15
osa eraldamine 102
otsekood 51

Paarsuse kontroll 116, 236, 332
parameetri eraldaja 288, 309
parameetrist sõltuv aadress 178
parameetrite muutmise operaator 213
Pascal 17
peatamise operaator 213
pesa 15, 49, 77, 105, 332
pideva tegevusega arvuti 11, 19
piiraja 277, 322
pikk pesa 79
(arvu) poollogaritmiline kuju 52
positsioon 83
positsiooniline arvusüsteem 31
primaaravaldis 291
programm 13, 121
programmeerimine 13
programmeerimise väikeautomatiseerimise süsteem 179
programmi loogiline skeem 213
programmjuhtimisega arvuti 13
protseduur 305
protseduuri identifikaator 288
— päis 306
— sisu 306
— kirjeldus 305, 326
— operaator 299, 306
pseudojuhuslikud arvud 255
— käsud 191
publikatsioonide keel 272
puhas rida 282
põhioperaator 296, 297
põhiprogramm 179
põhisümbol 276, 277, 322
pöördkood 59

raal 10
Ramejev 23
ratsionaalarv 34
rekursiivsed protseduurid 312
reserveeritud identifikaator 280
— sõna 322
rida 282

Salvestamiskäsk 110
salvestusseade 12
samaväärsed skeemid 219
samaväärsus 71
samm-element 303
seni-kui-element 304
seostamine nime järgi 307
— väärtuse järgi 307
Shefferi tehe 69
(algoritmi) sidusus 250
signalisatsioonipaneel 13, 86
silumine 227
sisemine aadress 177
— mälu 13
— tsükkel 163
sisemiste aadresside diapasoon 182
(alamprogrammi) sisend 179
sisendkommutaator 82
— seade 12
spetsifikaator 278
spetsifikatsioon 306
spetsifikatsioonide osa 306, 308
standardne tööpesa 178
— programm 176
— programmide süsteem 177
standardsed funktsioonid 288, 324
— protseduurid 312, 327
statistiliste katsete meetod 253
suhtelised aadressid 130, 334
sulg 278
summaator 56, 76
suunamiskäsk 15, 110
— operaator 299, 325
sõna 27

sõnalised põhisümbolid 276
sümboolsed aadressid 130
süstemaatilised vead 226

Taastamisblokk 144
tase 333
tegelik parameeter 288, 324
tegelike parameetrite loetelu 288
— parameetrite osa 288
tegur 290
tehte kood 15
tehtemärk 277, 279
test 227
Thomson 19
tihedusfunktsioon 253
tingimatu suunamine 15
tingimus 214, 290
tingimuslik operaator 300, 326
tingimusteta operaator 297
tingitud suunamine 15
topeltpesa 79
translaator 271
trükkimise operaator 213
Tšebõševi võrratus 257
tsooninumber 81
tsükkel 142
— tsüklis 163
tsükli kordamiste arv 108, 143
— lõputingimus 143
— parameeter 143, 302
— päis 302
tsükliloetelu 302
— element 302
tsüklioperaator 302, 326
tsükliregister 83
tõeline aadress 131, 334
tõeväärtus 68, 278
tähelis-numbriline informatsioon 78
tähestik 27
täht 27, 278
tähtede rida 288
täidesaatev blokk 124
täielik lokaliseerimine 285
täiendkood 59
täisarv 281
täisnull 107
tööpesa 136
töötlemispiirkond 183
tühi 273
— operaator 297
tüübi nimi 285
tüübikirjeldus 284
tüübiloetelu 285

Usaldatavus 257

Valikublokk 82
vasak pool 297
vasakute poolte loetelu 297
võrdlemine 102
võrdlus 293, 325
võrdlusmärk 279
välimine tsükkel 163
väline aadress 178
— mälu 13, 105
— salvestusseade 13
välised seadmed 12
(alamprogrammi) väljund 179
väljundkommutaator 83
väljundseade 13
väärtustatud funktsioon 287, 324
— lüliti 295
väärtuste osa 306

Wilkes 22

Üheaadressiline arvuti 15
(sõnade) ühendus 28
ühtlane jaotus 254
üksliige 290
ülekande operaator 213
ülemine raja 286
ületäitumine 50, 75
ümberadresseerimine 140
ümberadresseerimisblokk 144
ümberadresseerimissamm 108, 153

SISUKORD

Sissejuhatus 5

I Üldisi teatmeid arvutitest ja programmeerimisest 9

§ 1. Arvutite liigitusest. Programmjuhtimisega arvutite ehituse ja töötamise põhimõtetest 9
§ 2. Ülevaade arvutite ajaloost 17

II Arvutites kasutatav aritmeetika ja loogika 27

§ 3. Positsioonilised arvusüsteemid 27
§ 4. Aritmeetilised tehted positsioonilistes arvusüsteemides 38
§ 5. Arvude tähiste leidmine erinevates arvusüsteemides 41
§ 6. Arvude kujutamine arvutites 49
§ 7. Summaatori tüübid ja nendele vastavad koodid 56
§ 8. Lausearvutuse algmeid 68

III Arvuti käskude süsteem 75

§ 9. Üldisi andmeid arvuti «Ural-4» kohta 75
§ 10. Arvuti «Ural-4» käskude süsteem 86
§ 11. Arvuti «Minsk-22» käskude süsteem 105

IV Programmeerimise põhimõisted 121

§ 12. Blokk-skeemi koostamine 121
§ 13. Programmi koostamine blokk-skeemi järgi 130
§ 14. Tsüklite programmeerimine 142
§ 15. Tsükkel tsüklis 163
§ 16. Loogilised skaalad 170

V Täiendavaid programmeerimisvõtteid 176

§ 17. Alamprogrammide süsteemid 176
§ 18. Massiivide biblioteek 189
§ 19. Koostav programm ja interpreteeriv programm 197
§ 20. Programmi loogiline skeem ja selle teisendamine 212

VI Kontrollimine ja arvutusmeetodi valimine 226

§ 21. Programmi kontrollimine 226
§ 22. Arvuti töö kontrollimine 232
§ 23. Arvutusmeetodi valimise põhimõtteid 247
§ 24. Monte-Carlo meetod 253

VII Automaatne programmeerimine 269

§ 25. Algoritmilise keele mõiste 269
§ 26. ALGOLi põhielemendid 276
§ 27. Muutujad ja funktsioonid 282
§ 28. Avaldised 289
§ 29. Operaatorid 296
§ 30. Protseduurid 305
§ 31. Etalonkeeles kirjutatud programmide näiteid 315
§ 32. ALGOLi konkreetne esitus — MALGOL 321

Lisa. Arvuti «Minsk-32» 329
Kirjandus 349
Register 351

Юло Каазик, Айн Корьюс, Ивар Кулль. ПРОГРАММИРОВАНИЕ. На эстонском языке. Художественное оформление П. Рээвээр. Издательство «Валгус». Таллин, Пярнуское шоссе, 10.

Toimetaja K. Kallaste. Kunstiline toimetaja H. Keigo. Tehniline toimetaja T. Linkvist. Korrektor J. Nurme. Laduda antud 6. X 1969. Trükkida antud 18. XII 1970. Kohila Paberivabriku trükipaber nr. 2, 60 × 90/16. Trükipoognaid 22,25. Arvestuspoognaid 24,10. Trükiarv 2500. MB-10237. Tellimuse nr. 5897. H. Heidemanni nim. trükikoda, Tartu, Ülikooli 17/19. II.

Hind rbl. 1.25.

A
31227

TARTU ÜLIKOOLI RAAMATUKOGU
1 0300 00102451 4