TARTU ÜLIKOOL

FÜÜSIKA-KEEMIATEADUSKOND

EKSPERIMENTAALFÜÜSIKA JA TEHNOLOOGIA INSTITUUT

HEIKI KASEMÄGI

MIKROPROTSESSORSÜSTEEMI JA ASSEMBLERKEELE TUNDMAÕPPIMINE KONTROLLERI LKZ180 BAASIL

Bakalaureusetöö

Juhendajad: ANDO OTS
dotsent
van.ins. TOIVO VAJAKAS

TARTU 1997

SISUKORD

1. SISSEJUHATUS

1.1. KÄESOLEVA TÖÖ EESMÄRGID

Alates 1972.aastast, mil ilmus esimene 8-bitine mikroprotsessor Intel 8008, on aastast aastasse mikroprotsessortehnika tähtsus kasvanud. Mikroprotessorid on tunginud pea kõigisse eluvaldkondadesse alates Maa sisemuse uurimisest kuni kaugete Galaktikate uurimiseni. Varsti on peaaegu kõigis kodudes mõni seade, mille üheks (ja tihti tähtsaimaks) koostisosaks on mikroprotsessor.

Muutuv aeg, maailm ja tehnoloogia sundisid ka Tartu Ülikooli kaasajastama infotehnoloogia-alast koolitust, mis hõlmab ka mikroprotsessortehnikat. Seni anti vastavat õpetust kontrolleril KIT 8080, mis baseerus protsessoril Intel 8080. Intel 8080 ilmus 1973.aastal. Kontroller KIT 8080 lasti praktikumis käiku 1986.aastal. Vahepealsete aastatega vananes kontroller nii moraalselt kui ka füüsiliselt, mistõttu tekkis vajadus uue kontrolleri järele.

Kontrollerit LKZ180 asuti Eksperimentaalfüüsika ja tehnoloogia instituudis dotsent Ando Otsa juhendamisel välja töötama 1992.aastal. Seejuures tehti koostööd Joensuu Ülikooliga. Aluseks valiti Zilogi Z80180. Valik lähtus asjaoludest, et antud ajahetkel polnud Eestis analoogsed 8-bitised universaalprotsessorid (nt. Inteli 51-seeria) kuigi laialt levinud, seevastu Zilogi protsessoreid oli aga saada ja suhteliselt odavalt. Võrreldes KIT 8080-ga lisandus kontrollerile kaasaegne suhtlemis- ja juhtimisvõimalus personaalarvuti vahendusel. Kontroller sai töökõlbulikuks 1996.aastal. Kohe aga tekkis vajadus õppeülesannete järele, et vahetada praktikumis vana kontroller uue vastu välja. See oligi käesoleva bakalaureusetöö teema põhjus.

Et Z80180 on CISC-protsessor (keerulise käsustikuga), siis on tema baasil võimalik tundma õppida klassikalise protsessori ehitust, signaale ja töötamist. Käske ja adresseerimisviise on palju, käsud on erineva mahu ja töötaktide arvuga, käskude operandid võivad olla mälus, enne eelmise käsu täitmise lõpetamist uut käsku ei loeta jne.
Loogikaanalüsaator võimaldab eelpooltoodut tundma õppida protsessori andme-, aadressi- ja juhtimissignaalide testimisega, monitorprogramm aga registrite ja mälu sisu testimise ja muutmisega.
Kuigi protsessori andmesiin ja registrid on vaid 8-bitised (paaridena 16-bitised), saab ka sellise universaalprotsessoriga teha keerulisi aritmeetikatehteid, kasutades vaid liitmist, lahutamist ja nihutamist, teostada andmevahetust ja signaalitöötlust.

Aritmeetikatehetest vaadeldakse korrutamist ja jagamist, sest protsessoril need puuduvad (v.a. **`MLT ww`**). Nende ülesannete eesmärgiks on anda juhiseid ja algoritme, kuidas on võimalik teostada tehteid, mida protsessori käsustikus pole või on vähese bittide arvuga.
Assemblerkeeles programmeerija peab olema kodus erinevates arvusüsteemides. Seepärast on vaadeldud binaarkoodi teisendamist BCD- ja Gray koodi. Liiati omavad need koodid rakenduslikku külge indikaatortabloodes, analoog- digitaalmuundurites jne.
Mälu indekseerimine on vajalik andmetöötluses. Näiteks on mingit füüsikalist suurust mõõtev andur ühendatud mikroprotsessorsüsteemiga. Süsteemis tuleb andurilt saadud andmed teatud korrapära järgi edasiseks töötluseks mällu salvestada.
Kontrolleril saab tundma õppida kolme andmevahetusmoodust: asünkroonne jadaandmevahetus pika maa taha, rõõpkoodis andmevahetus kviteerimismeetodil ja I2C-protokoll.
ASCI0-i baasil antakse võtted ja algoritmid kompaktse andmevahetusliidese initsialiseerimiseks ja andmevahetuse teostamiseks. MAX238 võimaldab pidada ühendust nt. teises ruumis asuva arvutiga. Rööpvärat PIO võimaldab õpetada kviteerimismeetodi programmilist teostamist ja I2C-protokolli põhialuseid.

Käesolevas töös on seatud järgmised eesmärgid:

1) Kontrolleri konstruktsioonist lähtudes andmebaasi koostamine kontrolleri üksikosade initsialiseerimiseks, juhtimiseks ja testimiseks. Selleks koostatakse kontrolleri osade kirjeldused, antakse juhendid ja kommentaarid koostisosade initsialiseerimiseks. Andmed esitatakse sellisel kujul,

et nende alusel oleks võimalik kontrollerit tundma õppida ja praktikumis koostisosadega suhelda ka tudengil, kel pole varem mikroprotsessrorsüsteemiga kokkupuuteid olnud.

2) Kontrolleri juhtimiseks vajaliku tarkvara kirjeldamine ning süstematiseerimine. Selleks on vajalik koostada monitorprogrammi kirjeldus koos näidetega, esitada monitorprogrammi alamprogrammide kirjeldus ja anda ülesanded nendega tutvumiseks. Ühtlasi on nende harjutuste eesmärgiks assemblerkeelega kohanemine.

3) Anda juhised kontrolleri signaalide testimiseks loogikaanalüsaatoriga ja programmide silumiseks. Signaalide testimiseks esitatakse sammhaaval põhietapid alates ajadiagrammi kirjeldamisest kuni algoritmi koostamiseni. Testimiste eesmärgiks on arendada mikroprotsessorsüsteemi töö diagnoosimisoskust. Programmide silumine on peaaegu paratamatu kõigis ülesannetes, kus nõutakse programmeerimist. Selleks antakse üldised näpunäited, et kiirendada praktikumis tööd. Osades ülesannetes on näiteprogrammid, mis tuleb kasutajal siluda.

4) Andmevahetuse ja aritmeetikatehete teostamise õpetamine. Õpetamiseks esitatakse probleem, selle kirjeldus, algoritmid ning ülesanded algoritmide kinnistamiseks ja probleemi lahendamiseks. Vahel tuleb koostada ka algoritme näiteprogrammide järgi.

1.2. ÜLEVAADE ÕPPEKONTROLLERITEST

Kontrolleri LKZ180 jaoks ülesannete koostamisel tuli alustada peaaegu nullist, sest eeskujusid on vähe ja nende kohta infot raske kätte saada. See on seotud asjaoluga, et õppekontrollereid koostatakse maailmas peamiselt kohalikeks vajadusteks piiratud koguses. Tööstuslikku tootmist käesolevat praktikumi rahuldavate kontrollerite osas pole, sellest ka info nappus. Toodetakse küll seadmeid, mis on kas puhtalt demonstratsiooniks ega võimalda aktiivselt kontrolleri töösse sekkuda, või nii spetsiifilised, keerulised ja kallid, et nende kasutamine ei tuleks kõne allagi.
Siiski on võrdluseks esitatud andmed kolme kontrolleri kohta: KIT 8080, ADSP2101 ja Nanocomputer.

Kontrolleri KIT 8080 konstrueeriti ning valmistati TÜ füüsikaosakonnas Enn Ütsi juhendamisel [1]. Süsteemse tarkvara kirjutas Toivo Vajakas.

KIT 8080 on mikroprotsessoril KR580IK80 baseeruv kontroller, mis koosneb kahel trükkplaadil koostatud lihtsast mikroarvutist ja toiteplokist.
Põhiplaadil paikneb tsentraalne osa, mis koosneb järgmistest komponenetidest:
1) mikroprotsessor KR580IK80 (Intel 8080);
2) taktigeneraator KR580GF24 (Intel 8224);
3) süsteemi kontroller KR580VK28 (Inte 8228);
4) kahest K155LP10-tüüpi integraalskeemist koosnev aadressisiini võimendi;
5) dešifraatoril K155ID3 ehitatud kiibivaliku skeem, mis jagab 32-baidise aadressivälja kahe kilobaidisteks lõikudeks.;
6) kahel K573RF2-tüüpi EPROM-il baseeruv nelja kilobaidine püsimälu koos vabade kohtadega veel kahe sama tüüpi kiibi jaoks;
7) integraalskeemidel K573RU2 koostatud muutmälu. mis mahutab 4096 baidist sõna;
8) kaks KR580VV55-tüüpi rööpväratit (Intel 8255A);
9) kaks KR580VI53-tüüpi taimerit (Intel 8253);
10) jadavärat integraalskeemil KR580VV51 (Intel 8251A);
11) katkestuste kontroller integraalskeemil KR580VI59 (Intel 8259).

Programmid sisestab kasutaja kontrolleri mällu 3×16 maatriksisse seatud klaviatuuri abil. Andmed inditseeritakse tablool, mis koosneb kolmest 4-elemendilisest rühmast - aadressiväli, koodiväli ja operandiväli. 4 keskmist elementi on 5×12-punktilised maatriksindikaatorid, ülejäänud aga 7-segmendilised.
Toiteplokk väljastab stabiliseeritud toitepingeid -5V, +5V ja +12V kontrolleri toiteks ning +15V ja -15V praktikumis kasutatavate lisaseadmete tarbeks. Muutmälul on akutoide, et info säiliks ka pärast välise toite väljalülitamist. See hoiab aega kokku pikkade programmide korduva sisestamise pealt.
Püsimälus on nelja kilobaidine monitorprogramm, mis korraldab kontrolleri tööd ja lihtsustab kasutajal oma programmide koostamist, kontrollerisse sisestamist ja silumist.
Monitorprogramm võimaldab kontrolleri kasutajal:

1) teisendada arve 10-ndsüsteemist 16-ndsüsteemi ja vastupidi;
2) arvuti mälupesade sisu testimist ja muutmälu pesade sisu muutmist;
3) programmide kirjutamist arvuti muutmällu ning nende redigeerimist protsessori käskude mnemokoodis;
4) lasta kontrolleril täita kasutaja poolt sisestatud programmi suvalist lõiku;
5) lasta arvutil täita programme käskhaaval;
6) kontrollida protsessori suvalise registri sisu ja seda muuta;
7) andmemassiive arvuti mälus ümber paigutada;
8) arvutada kontrolleri suvalise mälupiirkonna kontrollsummat.
Monitorprogrammi alamprogrammid on kättesaadavad ka kasutajale. Sellisteks on inditseerimisprogrammid, alamprogrammid sümbolite sisetamiseks klaviatuurilt, viiteaegade ja helisignaali genereerimiseks, 32-bitise positiivse täisarvu jagamiseks 16-bitise positiivse täisarvuga.

Nagu eelpool mainitud, on õppekontrolleril KIT 8080 moraalselt vananenud. Peale selle on tal õpetamise seisukohast 3 puudust:
1) Ta ei ole kohaldatud mikroprotsessorsüsteemi signaalide testimiseks, kuna puuduvad vastavad kontroll-väljundviigud.
2) Programmi sisestamine sümbolhaaval klaviatuurilt on aeganõudev ja tülikas. Viga programmis ja programmi muutmine nõuab sageli kogu programmi uuesti sisestamist.
3) Ühenduse puudumine personaalarvutiga ei võimalda õpetada assemblerprogrammide kasutamisvõtteid nagu transleerimine, laadimine mällu, linkimine, silumine jne.

Firma “Analog Devices” protsessor ADSP2101 on 16-bitine digitaalset signaalitöötlust võimaldav mikroprotsessor [2]. Protsessor põhineb Harvardi arhitektuuril. Andmete jaoks on 2 siini ja aadresside jaoks 1 siin. Kiibis on programmi jaoks muut- või püsimälu, andmete jaoks muutmälu. Integreeritud on kaks jadaväratit ja taimer. Protsessoril on 3 sõltumatut arvutusüksust: ALU, korrutaja/aku ja nihutaja. Kasutada saab fikseeritud komaga arve ja ujuvkomaarve. Protsessori taktsagedus võib olla kuni 20 MHz. Ühe tsükli jooksul on ADSP2101 suuteline tegema järgmisi operatsioone:
1) järgmise käsu aadressi genereerimine;
2) järgmise käsu koodi lugemine;
3) 1 või 2 andmeülekannet;
4) arvutamine;
5) andmete saatmine ja vastuvõtt ühe või kahe jadavärati kaudu.
Kiibis on 5 sisemist siini:
1) programmimälu aadressisiin;
2) programmimälu andmesiin;
3) andmemälu aadressisiin;
4) andmemälu andmesiin;
5) tulemuse siin.

Protsessor kasutab oma töös pipeline-meetodit. Iga käsukood on 24-bitine ja käsk täidetakse ühe taktiga. Andmete muutmälu on 1 kilobait, programmi muutmälu 2 kilobaiti. Jadaväratid on sünkroonsed, võimaldavad ka multiprotsessormoodi.

ADSP2101 on mõeldud digitaalseks signaalitöötluseks. Mikroprotsessoritest algteadmiste omandamiseks on ta liiga keerulise ehitusega ja liiga spetsiifiline. Pealegi on ta RISC-protsessor. Antud protsessor on füüsikaosakonnas olemas, kuid puudub tarkvara ja andmebaas tema rakendamiseks. Kui need puudused lähitulevikus korvatakse, leiab ta oma koha digitaalse signaalitöötluse praktikumis.

Joensuu Ülikoolis kasutatakse mikroprotsessortehnika praktikumis kontrollerit Nanocomputer [3]. Kontrolleri on valmistanud Itaalia firma SGS-ATES Componenti Elettronici Spa (SGS). Kontroller põhineb Zilogi protsessoril Z80. Protsessori taktsageduseks on valitud 2.4574 MHz. Süsteemi kuuluvad 4-kilobaidine muutmälu, 2-kilobaidine EPROM ja 2 rööpväratit Z80PIO. Muutmälu hõlmab mäluaadressivälja madalamad aadressid, püsimälu kõrgemad. Püsimälus on monitorprogramm. Üks rööpvärat on mõeldud suhtlemiseks klaviatuuri ja indikatsioonitablooga, teine on aga kasutaja käsutuses. Nihkeregistrite süsteemi abil on andmeid võimalik salvestada jadakoodis C-tüüpi helikassetile ja sealt andmeid lugeda. Sama registrite süsteemi abil saab ASCII-koodis infot väljastada jadasisendiga printerile.

Süsteem koosneb kolmest plaadist integraallülitustega. Plaatidel olevad 86 klemmi on ühendatud süsteemi koostisosade viikudega: signaalid CPU-lt, siinidelt, toitest jne. Kõik signaalid on puhverdatud ja protsessori kirjutamissignaal /WR ka hilistatud. Klaviatuur koosneb kümnest klahvist. Indikatsioonitablool on 8 valgusdioodi, mis inditseerivad ainult andmesiini bittide seisu. Rööpväratid on seatud katkestusahelasse. Toitepinged on +5V, -5V, +12V ja -12V.
Praktikumis vaadeldakse süsteemi ehitust ja töötamist, andmete inditseerimist ja sisestamist, katkestuste ja viiteaegade korraldamist, aritmeetika- ja loogikatehteid.

Antud kontrolleril on väga hea õpetada mikroprotsessorsüsteemi koostamist. Nimelt pole süsteemi osad omavahel ühendatud, vaid nende viigud on ühendatud plaatidel olevate klemmidega. Praktikumi sooritajal tuleb juhtmetega ühendada klemmid nii, et moodustuksid andme-, aadressi- ja juhtimissiin, katkestus- ja toiteahelad jne., ühesõnaga koostada mikroprotsessorsüsteem. Samas peavad süsteemi töö ja võimaluste tundmaõppimiseks antavad ülesanded olema suhteliselt lihtsad ja vähe aega nõudvad. Erinevus Nanocomputeri ja LKZ180-e vahel sisnebki selles, et Nanocomputeril saab asetada põhirõhu süsteemi koostamisele, LKZ180-l aga süsteemi koostisosade reaalsele rakendamisele mikroprotsessorsüsteemi töö tundmaõppimiseks, andmevahetuse ja signaalitöötluse teostamisele.

Turu Ülikoolis õpetavad mikroprotsessortehnikat Risto Punkkinen ja Tom Kuusela. Loengukursuse esimeses osas [4] antakse üldine ülevaade arvuti ajaloost, arvusüsteemidest, loogikalülitustest, MOP-transistoridest. Seejärel vaadeldakse mikroprotsessori üldist ehitust, põhisõlmi ja nende tööpõhimõtet, erinevaid mälusid, mälude ühendamist protsessoriga ja suhtlemist. Inteli 8085-e ja 8086-e baasil antakse ülevaade protsessori käskudest ja masintsüklitest. Vaadeldakse asünkroonset ja sünkroonset jadakoodis andmevahetust, katkestusi, taimerit ja muid süsteemi koostiosi. Kursuse teises osas [5] konkretiseeritakse esimese osa materjali Inteli 8086-e baasil. Lõpus antakse lühiülevaade protsessoritest 8087, 80286 ja 80386. Olemasolevatest loengukonspektidest võib teha järelduse, et vaadeldud kursusega praktikumi ei kaasne. See ei tähenda seda, et praktikumi üldse pole. Tõenäoliselt on praktikum omaette õppeaine. Temaatiliselt on vaadeldud loengukursus sarnane Matti Fischeri poolt loetava kursusega “Arvutite arhitektuur”.

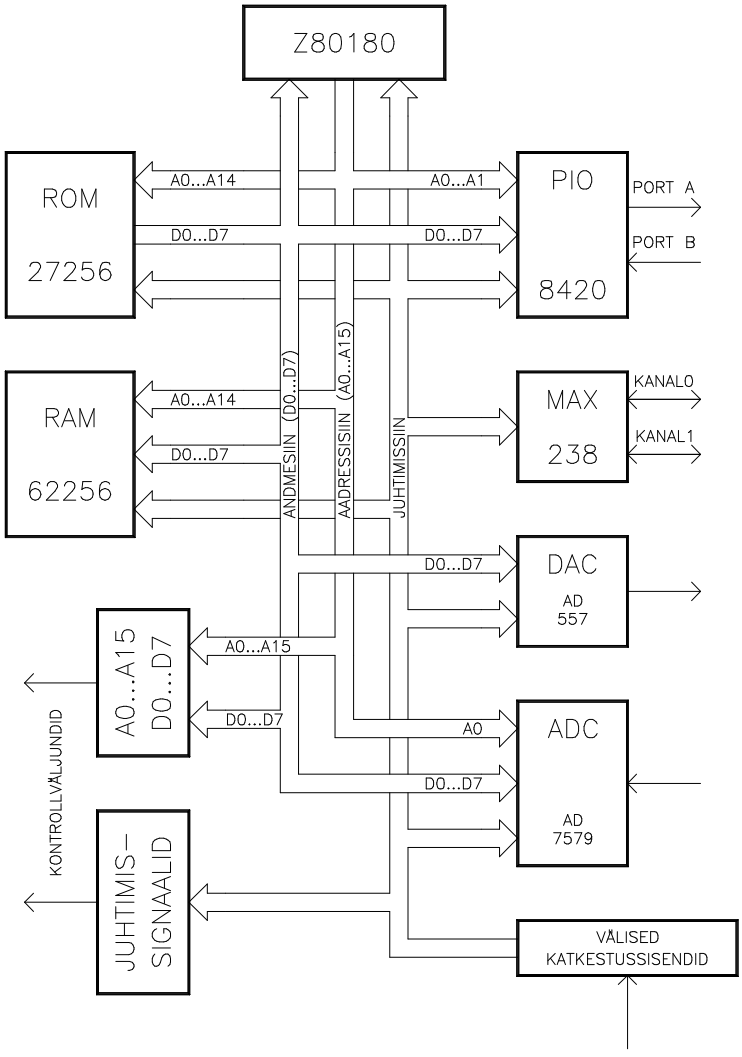
1.3. TÄHISTUSED

Signaalide, bittide ja viikude tähiste ees olev kaldkriips märgib seda, et antud signaal või bitt allub negatiivsele loogikale, s.t., et signaali aktiivne seisund on madal seisund.
Nt. /WR - kirjutamissignaal, mille aktiivne seisund on madal;
RD - lugemissignaal, mille aktiivne seisund on kõrge;
/WR/RD - signaal, mille madal seisund on aktiivne kirjutamissignaal, kõrge seisund aga aktiivne lugemissignaal;
WR//RD- signaal, mille kõrge seisund on aktiivne kirjutamissignaal, madal seisund aga aktivne lugemissignaal.

2. LKZ180 KIRJELDUS JA SUHTLEMINE SÜSTEEMI OSADEGA

2.1. ÜLDKIRJELDUS

Kontroller LKZ180 baseerub mikroprotsessoril ZILOG Z80180. Eraldi integraalskeemidel on 10-bitine analoog-digitaalmuundur AD7579 (ADC), 8-bitine digitaal-analoogmuundur AD557 (DAC) ja rööpvärat Z80PIO (PIO). Muutmälu 62256 (RAM) on 32 kilobaiti, millest 4 kilobaiti on monitorprogrammi käsutuses. Püsimälus 27256 (ROM) on personaalarvutiga sidet pidav monitorprogramm.

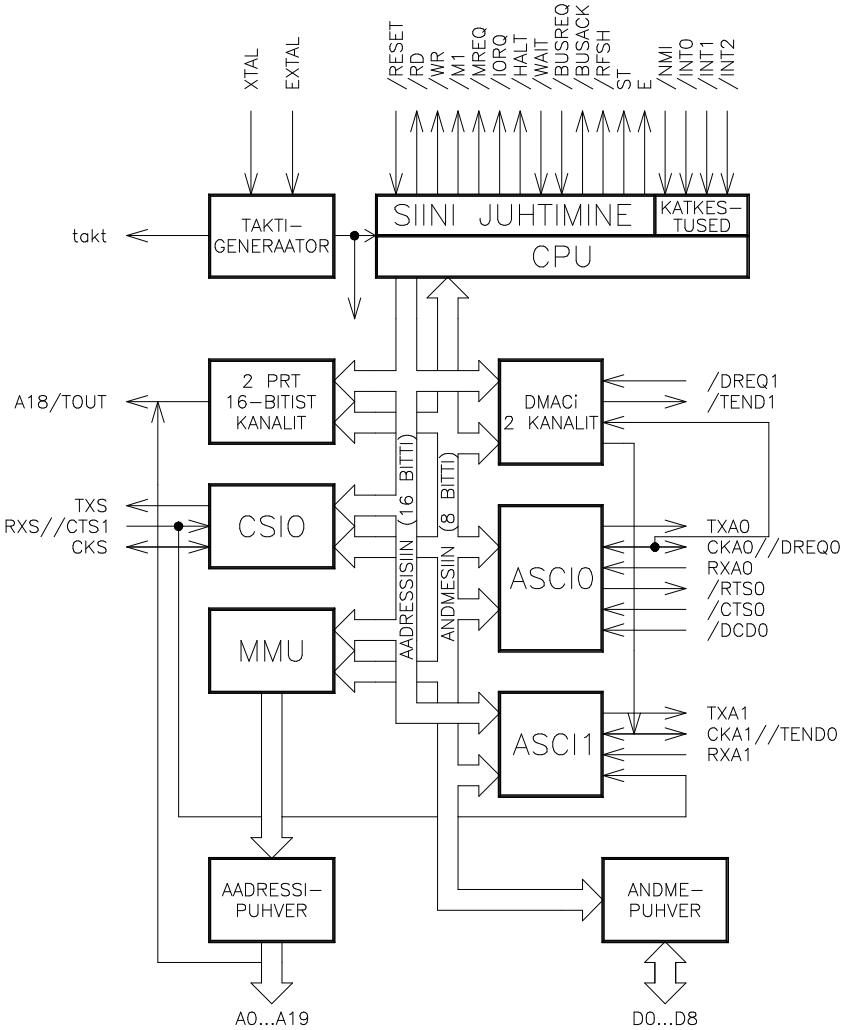


JOONIS 2.1.1: KONTROLLERI LKZ180 PLOKKSKEEM

Kuna kontrolleril pole spetsiaalseid lisaseadmeid andmete sisestamiseks protsessorisse, tulemuste inditseerimiseks ja kontrolleri töö juhtimiseks (s.t. puudub klaviatuur ja indikatsioonitabloo), siis on suhtlemine kontrolleriga teostatud personaalarvuti kaudu monitorprogrammi Z180 abil. Selle jaoks on üks ASCI kanalitest (ASCI1) ühendatud läbi MAX238 liidese personaalarvuti COM-porti.
Protsessori signaalide testimiseks on olemas spetsiaalsed kuplungid, kuhu saab ühendada loogikaanalüsaatori.
Eraldi sisend- ja väljundkuplungid on rööpvärati mõlemal pordil, ASCI kanalil 0, analoog-digitaal- ja digitaal-analoogmuunduritel. Katkestuste väliseks korraldamiseks on olemas vastavad sisendid.

Taktisignaali allikana kasutatakse kvartsostsillaatorit, mille sagedus on 12.288 MHz. Sellise sageduse korral on võimalik kasutada asünkroonsel jadaväratil maksimaalset saatmiskiirust 38400 boodi. Digitaalseadmete toitepinge on +5V, analoogseadmetel aga ±5V. Kontrollsignaalide väljundid on puhverdatud, signaalid on TTL-signaalid (0...5V). Puhvriteks on puhvervõimenditena tööle pandud registrid. Nende kasutamine väldib lühise korral kallimate seadmete riknemise. Puhvrid on ühesuunalised. Välised katkestussisendid on puhverdatud, sisendsignaal peab olema TTL-signaal. Analoog-digitaalmuunduri sisend on puhverdatud, signaal võib olenevalt reziimist olla unipolaarne standardselt 0...5V või bipolaarne standardselt -2.5V...+2.5V. Sisendpinge piirid sõltuvad tugipingest. Digitaal-analoogmuunduri väljund on puhverdatud, signaal on unipolaarne 0...2.5V. Rööpvärati port A on ühesuunalise puhvriga seatud jäigalt väljundpordiks, väljundsignaal on TTL-signaal. Port B on ühesuunalise puhvriga seatud jäigalt sisendpordiks, sisendsignaal peab olema TTL-signaal. Asünkroonse jadavärati ASCI mõlema kanali TTL-signaal muundatakse liidesega MAX238 personaalarvuti COM-pordi jaoks sobivaks signaaliks vahemikus -10V...+10V. Seejuures kasutatakse arvutiga suhtemiseks vaid ühte kanalit, teist kanalit saab kasutada andmevahetuseks teise seadmega, mis asub kaugemal kui mõni meeter ja mille sisendis-väljundis on samuti MAX-liides.

2.2. PROTSESSOR ZILOG Z80180



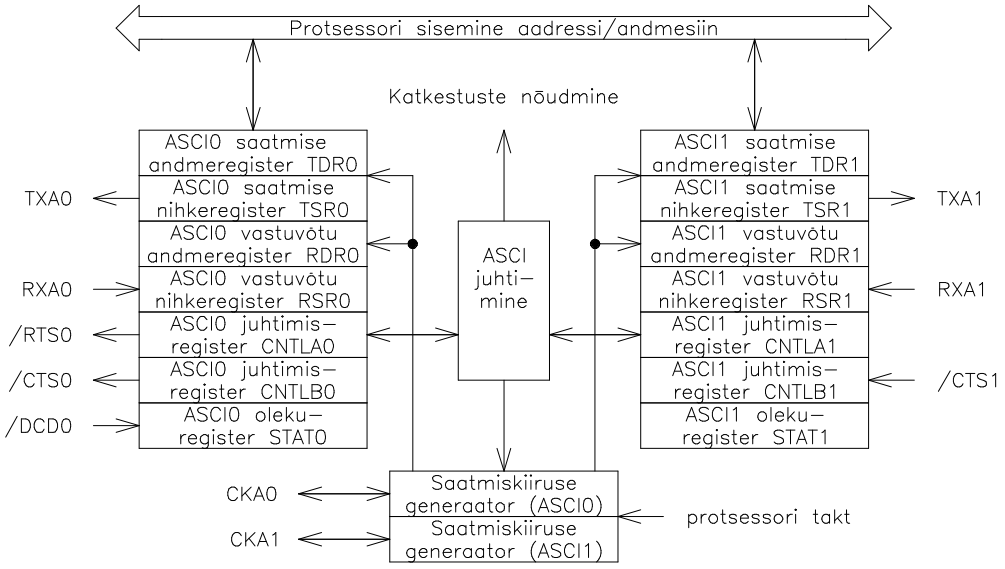
JOONIS 2.2.1: PROTSESSORI Z80180 PLOKKSKEEM

Protsessori taktsagedus on 6.144 MHz. Protsessor Z80180 [6] koosneb 5 põhiplokist: taktigeneraator, siinikontroller (hõlmab ka dünaamilise mälu värskendamist), katkestuste kontroller, mäluhaldusmoodul (MMU) ja tsentraalne töömoodul (CPU). Lisaks on integreeritud 4 sisend-väljundplokki: mällu otsepöördumine (DMA, 2 kanalit), asünkroonne jadakommunikatsiooniliides ehk asünkroonne

jadavärat (ASCI, 2 kanalit), programmeeritav ja taaslaetav taimer (PRT, 2 kanalit) ja sünkroonne jadavärat (CSIO).
Taktigeneraator genereerib süsteemi taktisageduse kas väliskristalli või -takti sisendist. Väline takt jagatakse kahega, saades sageduseks 6.144 MHz. Saadud taktisignaal antakse nii sisemistele kui välistele seadmetele.
Siinikontroller juhib siinide kasutamist CPU ja mõnede kiibisiseste välisseadmete poolt. See hõlmab muu hulgas ootetaktide lisamist, dünaamilise mälu (DRAM) värskendamist ja DMA jaoks siinide loovutamist. Kuna kontrolleril dünaamilist mälu pole, siis võib mäluvärskenduse välja lülitada.
Katkestuste kontroller jälgib, et sisemistel ja välistel katkestustel oleks õige prioriteet, et CPU saaks neile korrektselt vastata. Ühilduvuse säilitamiseks protsessoriga Z80 on olemas 3 erinevat katkestusmoodi.
Mäluhaldusmoodul MMU võimaldab kasutajal jaotada CPU poolt kasutatavat mälu 1-megabaidises aadressiruumis (loogiliselt vaid 64-kilobaidine) osadeks jagada (mappida). MMU ei toeta ühilduvust Z80CPU-ga siis, kui pakutakse juurdepääsu laiendatud mäluruumile.
Tsentraalne töötlusmoodul CPU on protsessori tuum ja ühilduv Zilogi Z80CPU-ga. Kasutatakse täiendatud Z80-e käsustikku, mis hõlmab ka 8-bitise korrutamise ja jagamise.
Otsemälupöördumise e. DMA kontroller võimaldab kiiret andmevahetust mälu ja S/V-seadmete vahel. Ülekandeid saab teha mälude vahel, mälu ja S/V-seadme vahel ning S/V-seadmete vahel. Ülekandemoode on 3. DMA ülekannet saab teostada kogu 1-megabaidise aadressiruumi ulatuses 64-kilobaidiste plokkidena.
Asünkroonsel jadaväratil ASCI on 2 täisduplekskanalit, millest igaühel on programmeeritav saatmiskiiruse generaator ja juhtsignaalid. Ka toetavad ASCI kanalid multiprotsessorformaati.
Programmeeritav taaslaetav loendur PRT koosneb kahest kanalist, millest igal on oma 16-bitine loendur (taimer) ja loendi taaslaadimise register. Loendurid kasutavad süsteemi taktsagedust, mida jagatakse 20-ga. PRT kanalil 1 on lisaks erineva olekuga signaalide genereerimist võimaldav väljund. Kontrolleril seda väljundit kasutada ei saa.
Sünkroonset jadaväratit CSIO kontrolleril kasutada ei saa, kuna selle väljund- ja sisendsignaalid pole kuplungitega ühendatud.

2.3. ASÜNKROONNE JADAVÄRAT ASCI

Protsessori Z80180 kiibisisesel asünkroonsel jadaväratil ASCI on 2 täisduplekskanalit. ASCI1 kaudu peetakse sidet personaalarvutiga, ASCI0 on aga kasutaja käsutuses. Iga ASCI kanal on eraldi programmeeritav. ASCI0-l on 7 registrit.



JOONIS 2.3.1: ASCI PLOKK-SKEEM

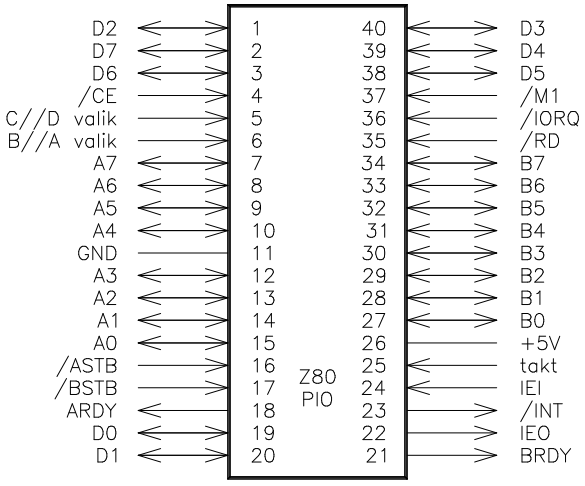
Väljastatavad andmed kirjutatakse käsuga **`OUT0 (TDR0),g`** ASCI0-i saatmise andmeregistrisse TDR0, mille aadress on 06H. TDR0 on nii kirjutatav kui ka loetav register. Sealt kantakse andmed üle

ASCI0-i saatmise nihkeregistrisse TSR0, mis on ühendatud TXA_0-väljundviiguga. Viimase kaudu saadetakse andmed bitthaaval välja. TSR0 pole sisend-väljundkäskudega loetav ega kirjutatav register.
RXA_0-sisendviigu kaudu loetakse sisestatavad andmed bitthaaval ASCI0-i vastuvõtu nihkeregistrisse RSR0, mis pole sisend-väljundkäskudega loetav ega kirjutatav register. Kui RSR0 on täis, kantakse andmed üle ASCI0-i vastuvõtu andmeregistrisse RDR0, kust neid saab käsuga **`IN0 g,(RDR0)`** lugeda. RDR0-i aadress on 08H.
ASCI0-i olekuregister STAT0 aadressiga 04H sisaldab mitut vealippu: ületäitumine, paarsusviga ja vale stopibitt. Ta võimaldab lubada ja keelata ASCI0-i poolt genereeritavaid katkestusi, peegeldab RDR0- ja TDR0-registrite ning $/DCD_0$-sisendsignaali olekut.
ASCI0-i juhtimisregister CNTLA0 aadressiga 00H võimaldab konfigureerida andmeformaati, multiprotsessormoodi, lubada-keelata saatjat ja vastuvõtjat, seada $/RTS_0$-väljundsignaali.
ASCI0-i juhtimisregister CNTLB0 võimaldab konfigureerida multiprotsessorformaati, valida saatmiskiirust ja paarsust. Registri aadress on 02H

Kuigi ASCI0-l on 3 modemi juhtimissignaali $/RTS_0$, $/CTS_0$ ja $/DCD_0$, kasutatakse neist vaid kahte esimest. Sisendsignaal $/CTS_0$ võimaldab välisseadmel juhtida ASCI0-i saatmismoodi. Kui $/CTS_0$ on kõrge ehk mitteaktiivne, hoitakse TDRE-bitt (saatmise andmeregister tühi) nullituna hoolimata TDR0-i olekust. Kui $/CTS_0$ läheb aktiivseks, peegeldab TDRE TDR0-i reaalset olekut. Seega saab selle signaaliga keelata andmete saatmise TDR0-i. Käimasolevat saatmist $/CTS_0$-i mitteaktiivne olek ei peata. $/RTS_0$ on väljundsignaal, mis ühendatuna välisseadme ASCI-kanali $/CTS_0$-sisendiga võimaldab juhtida välisseadme ASCI-kanali saatmismoodi.
$/DCD_0$-sisendsignaali ASCI0-l ei kasutata.

2.4. RÖÖPVÄRAT PIO

Kontrolleri rööpväratil Z180PIO on 2 sõltumatut 8-bitist porti, mis võivad olla nii sisendiks kui ka väljundiks. Et port A on ühesuunalise puhvri tõttu määratud väljundpordiks ja port B samal põhjusel sisendpordiks ning PIO-l ei saa kasutada juhtsignaale ARDY, BRDY, /ASTB, /BSTB ega katkestusi, kasutatakse praktikumis rööpvärati porte lihtsate paralleelsisendite ja -väljunditena.
Rööpvärati viigud on järgmised [7]:
D0...D7 - protsessori andmesiin;
/CE - kiibivalik;
B//A - A- või B-pordi valik;
C//D - andme- või juhtimisregistri valik;
/M1 - protsessori väljundsignaal /M1;



JOONIS 2.4.1: RÖÖPVÄRAT Z80PIO

/IORQ - protsessori signaal sisendi-väljundi nõudmiseks;
/RD - protsessori lugemissignaal /RD;

A0...A7 - A-pordi andmeviigud;
ARDY - A-pordi väljundsignaal, mis annab teada andmete olemasolust väljundviikudel (kontrolleril ei kasutata);
/ASTB - sisendsignaal A-pordile, et sisendviikudel on andmed (kontrolleril ei kasutata);
B0...B7 - B-pordi andmeviigud;
BRDY - B-pordi väljundsignaal, mis annab teada andmete olemasolust väljundviikudel (kontrolleril ei kasutata);
/BSTB - sisendsignaal B-pordile, et sisendviikudel on andmed (kontrolleril ei kasutata);
IEI - katkestuste lubamise sisendsignaal (kontrolleril ei kasutata);
IEO - katkestuste lubamise väljundsignaal (kontrolleril ei kasutata);
INT - katkestuse nõudmine protsessorilt (kontrolleril ei kasutata);
takt - taktisignaal;
+5V - toide;
GND - maa;

SIGNAAL			VALITUD REGISTER
/CE	B//A valik	C//D valik	
0	0	0	A-pordi andmeregister
0	0	1	A-pordi juhtimisregister
0	1	0	B-pordi andmeregister
0	1	1	B-pordi juhtimisregister
1	x	x	Seade pole valitud

TABEL 2.4.1: Z80PIO LOOGIKA VALIMINE

PIO signaalid on TTL-signaalid. PIO adresseerimiseks ja registrite valimiseks on B//A-viiguga ühendatud aadressiviik A0, C//D-viiguga - A1. /CE-viik on ühendatud dekoodriga, mille sisendis on signaalid A8, A9 ja A10. Neist formeerib dekooder kiibivalikusignaalid PIO-le, ADC-le ja DAC-le.
Rööpvärati initsialiseerimiseks tuleb portidesse kirjutada juhtimissõnad, mis määravad A-pordi väljundiks ja B-pordi sisendiks. A-pordi juhtimisregistri aadress on 302H ja sinna tuleb kirjutada käsuga **`OUT (C),g`** sõna 0FH. B-pordi juhtimisregistri aadress on 303H ja sinna tuleb kirjutada sõna 10H. A-pordi andmeregistri aadress on 300H ja sinna saab saata väljastatavaid andmeid käsuga **`OUT (C),g`** . B-pordi aadress on 301H ja sealt saab lugeda sisestatud andmeid käsuga **`IN g,(C)`**.

SIGNAAL			TÄHENDUS
/M1	/IORQ	/RD	
0	0	0	Pole tähendust
0	0	1	Katkestuse tunnustamine (ei kasutata kontrolleril)
0	1	0	Katkestuse teenindamise lõpu kontroll (ei kasutata)
0	1	1	Reset
1	0	0	Andmete lugemine PIO-st protsessorisse
1	0	1	Andmete kirjutamine protsessorist PIO-sse
1	1	0	Pole tähendust
1	1	1	Pole tähendust

TABEL 2.4.2: Z80PIO JUHTIMISSIGNAALID

2.5. ANALOOG-DIGITAALMUUNDUR ADC

Analoog-digitaalmuundur AD7579 on 10-bitise andmesõnaga [8]. Toitepinge on 5 volti. Maksimaalne diskreetimissagedus on 50 kHz ja sisendi maksimaalne sagedusriba - 25 kHz. Sisendsignaal võib olla nii bipolaarne (standardselt ± 2.5 V) kui ka unipolaarne (standardselt 0..5 V). Reziimi on võimalik lüliti abil valida.

Analoogsisend (V)	Digitaalväljund	Analoogsisend (V)	Digitaalväljund
-2.500	00 0000 0000	0	00 0000 0000
-2.49512	00 0000 0001	0.00488	00 0000 0001
0.00	10 0000 0000	2.500	10 0000 0000
+2.49512	11 1111 1111	4.99512	11 1111 1111

TABEL 2.5.1: ADC SISENDI JA VÄLJUNDI VAHELINE SEOS

Kuna ADC on protsessoriga ühendatud 8-bitise andmesiini abil, siis jaguneb tema väljundsõna kaheks: 8 madalamat + 2 kõrgemat bitti. ADC-st andmete lugemisel tuleb lugeda mõlemad baidid, muidu läheb osa infot kaduma. Seejuures pole oluline, kumb bait enne lugeda.

MADAL	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
KÕRGE	EOC*	0	0	0	0	0	DB9	DB8

*EOC - sisemine muundamise lõpetatuse lipp
**DB0..DB9 - andmebitid

TABEL 2.5.2: AD7579 VÄLJUNDANDMETE FORMAAT

ADC tuleb igaks muundamiseks eraldi käivitada ja seejärel temast andmed lugeda. Käivitamiseks tuleb temasse sisestada väljundkäsuga **`OUT (C),g`** suvaline andmesõna, sest muundur käivitub /WR-signaali langusfrondil. Madalama andmebaidi lugemiseks sisendkäsuga **`IN g,(C)`** on aadress 200H, kõrgema lugemiseks aga 201H. Mõlema käsu korral peab aadress olema BC-registris.

/CS	/WR	/RD	/HBEN	FUNKTSIOON
1	X	X	X	Pole valitud
0	1	1	X	On valitud, ootab /WR- ja /RD-signaale
0	0	1	X	Alustada muundamist /WR-i langusfrondil
0	1	0	0	Lubada lugeda madalam andmebait
0	1	0	1	Lubada lugeda kõrgem andmebait

TABEL 2.5.3: AD7579 OLEKUTABEL

/CS on kiibivaliku signaal, /WR on protsessori kirjutamissignaal, /RD on protsessori lugemissignaal, /HBEN on kõrgema andmebaidi lubamise sisendsignaal, et ADC-st saaks lugeda kõrgemat baiti.

2.6. DIGITAAL-ANALOOGMUUNDUR DAC

Kontrolleril kasutatakse 8-bitist digitaal-analoogmuundurit AD557 [9]. Toitepinge on +5 V. Väljund on unipolaarne (0..+2.56 V).
Andmete sisestamiseks DAC-i tuleb nad väljundkäsuga **`OUT (C),g`** saata aadressil 100H.

DIGITAALSISEND	ANALOOGVÄLJUND (V)	DIGITAALSISEND	ANALOOGVÄLJUND (V)
0000 0000 (00H)	0	0111 1111 (7FH)	1.270
0000 0001 (01H)	0.010	1000 0000 (80H)	1.280
0000 0010 (02H)	0.020	1100 0000 (C0H)	1.920
0000 1111 (0FH)	0.150	1111 1110 (FEH)	2.540
0001 0000 (10H)	0.160	1111 1111 (FFH)	2.550

TABEL 2.6.1: AD557 SISENDI JA VÄLJUNDI SEOS

2.7. MÄLUD

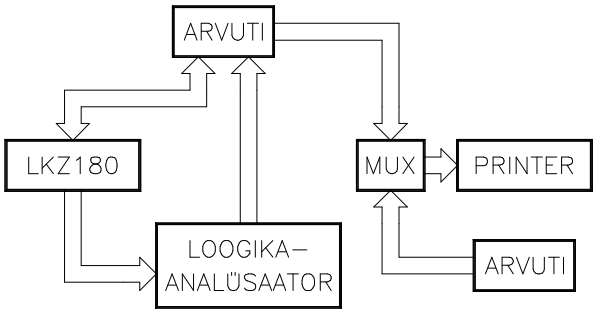
Kontrolleril kasutatav püsimälu EPROM on 32-kilobaidine [10]. Püsimälu algusaadress on 0000H ja lõppaadress - 7FFFH. Püsimällu on salvestatud personaalarvutiga sidet pidada ja kontrollerit juhtida võimaldav monitorprogramm ning mõned alamprogramid, mida saab kasutada kasutajaprogrammide koosseisus. Monitorprogramm peab personaalarvutiga sidet ASCI kanali 1 ja arvuti COM-pordi kaudu. Monitorprogramm võimaldab registrite ja mälu sisu testimist ning muutmist, programmide käivitamist tervikuna ja sammhaaval silumist.

Staatiline muutmälu RAM on 32-kilobaidine. Muutmälu algusaadress on 8000H ja lõppaadress - FFFFH. Kasutaja käsutuses on RAM-i esimesed 28 kilobaiti alates aadressist 8000H kuni aadressini EFFFH. 4 kilobaiti alates F000H kuni FFFFH on monitorprogrammi käsutuses ja **kasutaja EI TOHI sellesse mälupiirkonda midagi kirjutada**.
Iga kasutajaprogrammi alguses tuleb käsuga **`LD SP,pinuviit`** seada paika pinumälu algus. Pinumälu algusaadress peab olema väiksem kui F000H, ent tuleb jälgida, et mahukad andmemassiivid ja pikad programmid ei satuks pinumälu piirkonda.

3. KONTROLLERI SIGNAALIDE JA TÖÖ TESTIMINE

3.1. KONTROLLERI SIGNAALIDE TESTIMINE

Et kontrolleril pole autonoomset suhtlemisvõimalust, siis on see teostatud loogikaanalüsaatori ja personaalarvuti vahendusel.



JOONIS 3.1.1: SUHTLEMINE KONTROLLERIGA

MUX on multipleksor, mis võimaldab printeriga ühendada kuni 4 personaalarvutit. Kõige esimesena signaali saatnud arvuti saab kõige esimesena printerit kasutada. Järgmised peavad ootama, kuni järjekord nendeni jõuab.
Signaalide testimiseks kasutatakse loogikaanalüsaatorit BLACK STAR 3332 [11]. Testimiseks tuleb analüsaator ühendada mooduliga PM3332 ja see omakorda kontrolleriga. Ühendusteks on olemas vastavad juhtmed.
Analüsaatori toite sisselülitamise järel ilmub ekraanile peamenüü, kus on menüüd **SETUP**, **ACQUIRE**, **DISPLAY** ja **UTILITIES**. Nende menüüde alammenüüdes kasutada erinevate parameetrite vahel liikumiseks **TAB**-klahvi (valitud parameeter paistab negatiivis), parameetri muutmiseks aga **INC**- ja **DEC**-klahve. Alammenüüst väljutakse ESC-klahviga. Kõigis menüüdes saab abiinfot, kui vajutada **HELP**-klahvi.
SETUP-menüü alammenüüs **CLOCK** tuleb kasutatavaks taktiallikaks (**CLOCK USED**) valida INTRERNAL, käivitushetkeks (**CLK1**) valida langusfront NEG, trigeri positsioon (**TRIG POSITION**) seada seisu 0000, **TRIG DELAY** ja **TRIG HOLD-OFF** seisu OFF. **TRESHOLD**-iks valida TTL. Sagedus (**INT CLOCK**) tuleb valida selline, et analüsaator oleks võimeline signaalide olekuid ajaliselt eristama. Madalaima piiri seab sagedusele protsessori taktsagedus.
SETUP-menüü alammenüüdes **TRIGGER1**,**2** saab paika panna tingimused, millal analüsaator peab käivituma. Kui kasutatakse vaid ühte trigersõna, peab vastavas TRIGGER-menüüs olema valitud sõna **ON**, teises, mida ei kasutata, aga **OFF**. Trigersõna saab seada nii heksa-, okta- kui ka kahendkoodis. Seejuures on kõigis koodides olevad sõnad oma tähenduselt identsed, erinev on vaid kood. Kuna analüsaatoril kasutatakse kõiki 32-te andmekanalit, siis on kõige mõtekam trigersõna paika seada kahendkoodis, mille korral igale bitile vastab üks testitav analüsaatori sisendkanal.

KANAL	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
HI	CLK	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14

KANAL	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LOW	A15	CLK	D0	D1	D2	D3	D4	D5	D6	D7	/RD	/WR	/IORQ	/MREQ	/M1	CLK

TABEL 3.1.1: TRIGERSÕNA BITTIDE JA ANDMEKANALITE SEOS

Biti seis 1 vastab kontrolleri signaali kõrgele olekule, seis 0 - madalale olekule ja X märgib signaali suvalist olekut. Kuna signaale on palju, ei piisa 1..2 biti seadmisest, vaid tuleb valida kombinatsioon, mis kindlustaks analüsaatori käivitumise kasutaja poolt soovitud hetkel.
Seejärel valida DISPLAY-menüü alammenüü **TIMING**, misjärel ilmuvad ekraanile kursor ning 8 kanali seisundite ajadiagramm. Nüüd tuleb vajutada **RUN**-klahvi ja ekraanile ilmunud alammenüüst valida

vajalik reziim. Tavaliselt on selleks **SINGLE**, mille korral käivitub analüsaator trigersõnaga etteantud tingimuste täitumisel ja loeb andmeid mällu, kuni viimane on täidetud. Andmed ilmuvad kohe ka ekraanile. Seetõttu ongi mugav avada enne analüsaatori käivitamist TIMING-menüü. Loomulikult saab RUN-klahviga analüsaatorit ka teistest menüüdest käivitada. Seejärel käivitada kontrolleris olev programm.
Saadud andmed püsivad analüsaatori ACQ-mälus kuni järgmise käivitamiseni. Andmete mugavamaks töötlemiseks ja salvestamiseks tuleb käivitada programm LA_VIEW:

```
C:\LOGICANA\>la_view
```

Antud programmiga saab lugeda analüsaatori mälust andmed personaalarvutisse, need salvestada, graafikuna välja printida. Samuti on võimalik vaadata eelnevalt failidesse salvestatud graafikuid, neid suurendada, vähendada. Printimiseks võib kasutada programmi pakutavaid võimalusi, ent kõige mugavam on kasutada personaalarvuti klahvi **PrtSc**, millega prinditakse antud hetkel kuvaril olev pilt. Seejuures peab olema aktiveeritud kataloogis DOS620 olev programm **graphics.com**. Loomulikult peab ka printer olema arvutiga ühendatud ning sisse lülitatud.

Lisas 2 graafikul 1 on kõik 2048 lugemit, mida analüsaator võimaldab, graafikul 2 on ühe programmi ajal siinidel olnud signaalid ja graafikul 3 - sama programmi esimesed käsud.

3.2. PROTSESSORI REGISTRITE JA MÄLU SISU TESTIMINE NING MUUTMINE

Kontrolleriga suhtlemiseks tuleb personaalarvutil käivitada programm Z180.

Silumiskeskkonna käsud [12]:

```
! HELP EXIT BREAK DUMP GO PATCH XREG TRACE MOVE MATH COLDBOOT! READ WRITE
LIST VIEW CSUM
Lisainfo saamiseks käsu kohta:
HELP käsk
```

Kontrollerisse laaditud programmi kontrolleri mälus ja mälupesade sisu saab testida käsuga **DUMP**:

```
DUMP [n1 [n2]]
D [n1 [n2]]
Näitab Z180 mälu sisu ekraanil 16-ndsüsteemis ja ASCII tekstina alates
aadressist n1 ja lõpetades aadressiga n2 (kui on antud).
Näide: DUMP .256
Näide: D driver_start driver_end-1
Nt.
Z180> d 8390
ADDR  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  01234567 89ABCDEF
----  -- -- -- -- -- -- -- --  -- -- -- -- -- -- -- --  -------- --------
8390  65 73 74 20 4F 4B 0D 00  0D 52 41 4D 65 6D 6F 72  est OK.. .RAMemor
83A0  79 20 74 65 73 74 20 61  62 6F 72 74 65 64 00 20  y test a borted.
83B0  62 79 74 65 73 20 4F 4B  20 61 6E 64 20 61 76 61  bytes OK  and ava
83C0  69 6C 61 62 6C 65 00 00  00 00 00 00 00 00 00 00  ilable.. ........
83D0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
83E0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
```

Mälus olevat programmi saab assembleris vaadata käsuga **LIST**:

```
LIST [n1 [n2]]
L [n1 [n2]]
Disassembleriga näitab Z180 käske alates aadressist n1 ja lõpetades
aadressiga n2 (kui on antud).
Näide: LIST 0100
Näide: L 0100 cpu_end
Nt: Z180>l 8000
         FILL:
8000     2600     LD       H,00
8002     DD211A80 LD       IX,LOOP
8006     01FF03   LD       BC,03FF
```

```
         FILL1:
8009     DD7400    LD       (IX+00),H
800C     DD23      INC      IX
```

Mälu sisu saab baithaaval redigeerida käsuga **PATCH**:

```
PATCH [n]
P [n]
Alustab mälu sisu baithaaval redigeerimist aadressist n.
Näide: PATCH 2F60
Näide: P TABLE
Nt: Z180> p 8071
8071H (10H) = 0a
8072H (4aH) =
```

Registrite ja lippude seisude vaatamiseks ja redigeerimiseks on käsk **XREG**:

```
XREG [rr n]
X [rr n]
Kui argumente pole antud, näitab Z180 registrite jooksvaid väärtusi. Kui
argumendid on antud, siis register/lipp saab uue väärtuse n.
rr = register/lipp, mida muudetakse, üks järgnevatest:

registrid:  A    F    B    C    D    E    H    L
              AF   BC   DE   HL   IX   IY   SP   PC
      lipud:  SF   ZF   HF   P/V  NF   CF

Näide: XREG BC .63000 (16-bitine BC)
Näide: XREG B 11 (8-bitine B)
Näide: X ZF 1 ( ZERO lipp tõene)
Näide: X CF 0 ( CARRY lipp väär)
Nt.
Z180> x
A F  B C  D E  H L  IX   IY   SP   PC
006C F80C 6C39 83AF 2A88 888C B000 8294   SF=0 ZF=1 HF=0 P/V=1 NF=0 CF=0
         MEM7:
8294 CF           RST     08
Z180> x pc 8000
Z180> x
A F  B C  D E  H L  IX   IY   SP   PC
006C F80C 6C39 83AF 2A88 888C B000 8000   SF=0 ZF=1 HF=0 P/V=1 NF=0 CF=0
         INIT0:
8000 210780       LD      HL,TALK0
```

Programmi saab sammhaaval täita käsuga **TRACE**:

```
TRACE [n]
T [n]
Täidab n (vaikimisi = 1) Z180 käsku alates jooksvast Z180 PC väärtusest.
Z180 registrite ja lippude seis ilmub ekraanile iga käsu täitmise järel.
Näide: TRACE
Näide: T .10
Nt: Z180>x pc algus
Z180>x
Z180> t
A F  B C  D E  H L  IX   IY   SP   PC
006C F80C 6C39 83AF 2A88 888C B000 8233   SF=0 ZF=1 HF=0 P/V=1 NF=0 CF=0
8233 CD0080       CALL    INIT0
Z180> t
A F  B C  D E  H L  IX   IY   SP   PC
006C F80C 6C39 83AF 2A88 888C AFFE 8000   SF=0 ZF=1 HF=0 P/V=1 NF=0 CF=0
         INIT0:
8000 210780       LD      HL,TALK0
Z180> t
A F  B C  D E  H L  IX   IY   SP   PC
```

```
006C F80C 6C39 8007 2A88 888C AFFE 8003   SF=0 ZF=1 HF=0 P/V=1 NF=0 CF=0
8003 CD0810       CALL    puts
```

Kuvari seisu saab vaadata ja salvestada käsuga **VIEW**:

```
VIEW [filename]
V [filename]
```

Ilma parameetrita käsu korral saab vaadata ekraani väljundi salvestust. VIEW-reziimis saab ekraanil infot nihutada üles/alla noole- ning PgUp-, PgDown-klahvidega. ESC-klahviga saab VIEW-reziimist tagasi normaalsesse reziimi. Kui parameetriks anda faili nimi, siis kirjutatakse ekraani puhver faili.

Silumiskeskkonnast väljumiseks on käsk **EXIT (E).**
Kõigi käskude kirjeldused on juhendis [12].

Arvud on monitorprogrammis 16-ndsüsteemis. Numbrilised väärtused käskudes antakse järgneval kujul:

FFFF	16-ndsüsteemi arv
.99999	10-ndsüsteemi arv
.-99999	Negatiivne 10-ndarv
'A'	Sümboli ASCII kood
xxxxxx	Märgend xxxxxx (defineeritud .ASM lähtetekstis)

3.3. ASSEMBLERPROGRAMMI KOOSTAMINE

3.3.1. ASSEMBLERI SASM KIRJELDUS

Z180 käskude operandid võivad olla 8- või 16-bitised arvud [12]. 8-bitise operandi väärtus võib kujutada:
a) märgita arvu vahemikus 0..255;
b) märgiga arvu vahemikus -128..+127;
c) ASCII sümbolit.
16-bitise operandi väärtuste vahemik on suurem: 0..65535 märgita ja -32768..+32767 märgiga arvudel.
Arvud on assembleris 10-ndsüsteemis. 16-ndsüsteemi arvude tunnuseks on H täht lõpus. Kuna ka 16-ndsüsteemi arv peab algama numbriga, siis tuleb tähega alguse korral ette lisada 0. Assembler ei tee vahet suur- ja väiketähtedel.
Sümboli kood tuleb kirjutada apostroofide vahele.
Näited konstantidest:
1)õige on 10, 0123H, 65535, -1234, 0FFH, 'u';
2)vale on FFH (ei alga numbriga), 100000 (liiga suur arv).
Operandiks võib olla ka avaldis, mille koosseisus on konstandid ja märgendid. Avaldises on lubatud liitmine, lahutamine, korrutamine, jagamine, sulud. Avaldise väärtus peab mahtuma vastavalt 8- või 16-bitise operandi lubatud piiridesse.
Näide avaldisest:

```
        LD    HL,(RUUT+3*2)
        .......
RUUT:   DW    0,1*1,2*2,3*3,4*4
```

DW-käsuga defineeritaks ruutude tabel, iga arv tabelis võtab 2 baiti. LD-käsuga loetakse ruutude tabeli algusest 6 baidi võrra suuremalt aadressilt 16-bitine väärtus 3*3=9 HL-registrisse.

Kasutaja koostatavad programmid algavad üldjuhul aadressilt 8000H, s.t. muutmälu algusest. Programmi järel tuleb reserveerida mälu andmete salvestamiseks. Mälu reserveerimiseks on käsud **DS, DB, DW**.
DS on andmetele mälu reserveerimiseks. Mälu seis programmi töö alguses ei ole määratud. Rea üldkuju on **DS NNN** , kus NNN on konstant, mis näitab reserveeritavate baitide arvu.
Nt:

```
DS      1
DS      1000
```
Nende kahe käsuga reserveeritakse kokku 1001 baiti.
DB on mälus konstantsete andmete kirjeldamiseks. Üldkuju on **DB V1,...,VN**, kus V1,..,VN tähistab komaga eraldatud 8-bitiste väärtuste loetelu.
Nt:
```
DB      2,5,7,11,13,17,19,23
DB      0EFH,0CDH,01H,075H,0FFH
DB      'H','e','l','l','o',0
```
Lisaks on olemas eraldi variand teksti kirjeldamiseks, kus sümbolid on kokku kirjutatud ühiste apostroofide vahele. Eelmise näite viimase rea saaks kirjutada `DB 'Hello',0`
16-bitiste arvude kirjeldamiseks on eraldi korraldus **DW** (define word). Selle korralduse üldkuju on **DW W1,..,WN**, kus W1,..,WN tähistab komaga eraldatud 16-bitiste väärtuste loetelu.
Nt:
```
DW      1, 10, 100, 1000, 10000
DW      0FFFFH, 0ABCDH, 1000H
```

Iga programmis kirjeldatud masinkäsu või andmevälja ette võib panna märgendi, et edaspidi selle abil vastavale aadressile viidata. Märgend on rea algusesse paigutatud tähtedest ja numbritest koosnev nimi, mis algab tähega ja lõppu lisatud kooloniga.
EQU korralduse abil saab suvalisele arvulisele väärtusele nime anda. Korralduse üldkuju on järgmine:

nimi EQU väärtus.

Erinevalt märgendist ei ole siin nime lõpus koolonit vaja.
Nt:
```
PORTA   EQU     300H
CR      EQU     0DH
        IN      (PORTA)
        CP      CR
```

Koostatav programm algab alati reaga **include lkz180.inc.** Sellega liidetakse programmi algusesse failis lkz180.inc olevad korraldused. Põhiliselt on seal EQU-direktiivid, millega LKZ180 välisseadmetele on antud nimed. Samuti defineeritakse seal programmi algusaadressiks 8000H ja püsimälus olevate alamprogrammide aadressid.

Programmi iga rida koosneb oma ülesehituselt järgnevatest osadest:

märgend käsukood operandid.

Osad peavad olema omavahel tabulaatori või tühikutega eraldatud. Kõige mugavam on eraldajaks kasutada tabulaatorklahvi, siis on käsud ja operandid automaatselt kenasti tulpades. Rea alguses võib olla märgend. Käsukoodi ees peab olema tabulaator või tühik. Operandid on omavahel eraldatud komadega. Rea lõppu võib lisada kommentaari, mis algab semikooloniga. Programmi transleerimisel ignoreeritakse kommentaare.

Programmi lõpetamiseks on kolm võimalust:
1) Kui programm lõpeb käsuga RST 8, antakse juhtimine üle monitorprogrammile . Seejuures lõpetab kasutaja programm töö registrite seisu salvestamisega.
2) Kui programm ei lõpeta tööd, vaid kasutaja vajutab kontrolleri RESET nupule, siis antakse juhtimine monitorprogrammile. Registrite seisu ei salvestata ning neis on seis, mis oli programmi käivitamisel.
3) Programm lõpetab töö registrite seisu salvestamata, kui suunata monitorprogrammi algusesse käsuga JP 0, mis on praktiliselt identne RESET nupule vajutamisega. Erinevus on selle, et RESET viib mõned välisseadmed tagasi algolekusse, kui nad olid ümber programmeeritud. JP 0 korral seda ei tehta (v.a. ASCI, mis peab sidet kontrolleri ja arvuti vahel).
Programmi lõppu tuleb lisada rida **END**.
Selle teatatakse assemblerile programmi teksti lõpust. Mingit käsukoodi ei genereerita.

3.3.2. PROGRAMMI KIRJUTAMINE, TRANSLEERIMINE JA SISESTAMINE KONTROLLERISSE

Programmi lähtetekst tuleb valmis kirjutada ASCII-tekstiredaktoriga, soovitavalt Norton Editoriga. Kirjutamisel tuleb järgida eelnevalt antud reegleid. Lähtefaili laiendiks peab olema .ASM. Programmiga lingitavate failide laiendid võib ise valida.

Programmi transleerimiseks tuleb sisestada MS-DOS-i käsurealt **a <programmifaili nimi>**. Seejuures ei tohi laiendit .ASM lisada ja vastav fail peab olema selles kataloogis, kus a-käsk sisestati. Vastasel juhul tuleb anda käsus tee failini. Kui transleerimisel avastati vigu, tuleb need lähtetekstis parandada ja uuesti transleerida.

Programmi laadimiseks kontrollerisse tuleb sisestada MS-DOS-i käsurealt **z180 <programmifaili nimi>**. Jälle ei tohi mingeid laiendeid lisada. Silumiskeskkonda sisenemiseks ilma programmi kontrollerisse laadimata tuleb sisestada käsk **z180**.
Silumiskeskkonnas saab programmi kontrollerisse laadida käsuga **READ**. Seejuures pole vaja jälle laiendeid lisada.

```
READ [n] <filename.typ>
R [n] <filename.typ>
Loeb faili filename.typ alates aadressist n (vaikimisi = 0100h). .HEX faile
lugedes n ignoreeritakse ja arvestatakse aadresse .HEX kirjetes. See on
tavaline moodus laadida täidetav programm mällu.
Nt: Z180>r c:\heiki\kylg\kylg
Nt: Z180>r kylg
```

4 PRAKTIKUMIS TEOSTATAVAID HARJUTUSI

4.1. TUTVUMINE MONITORPROGRAMMIGA. PÜSIMÄLUS OLEVATE ALAMPROGRAMMIDE KASUTAMINE

4.1.1. ÜLESANNE MONITORPROGRAMMIGA TUTVUMISEKS

Monitorprogrammiga tutvumise eesmärgiks on harjutada monitorprogrammi käskude kasutamist, õppida tundma monitorprogrammi ja tema pakutavaid võimalusi programmide silumiseks. Kui monitorprogrammi käsud on n.ö. käe sisse kulunud, võimaldab see hilisemate harjutuste käigus kiiremini programme töödelda.

(1) Programmi laadimiseks kontrollerisse minna kataloogi, kus on transleeritud programm. Sisestada MS-DOS-i käsurealt **z180 <programmifaili nimi>**.
Silumiskeskkonnas vaadata sama programmi paiknemist mälus käsuga **dump <programmi algusaadress> <programmi lõppaadress>.** Kuvada programm käsuga **list <programmi algusaadress> <programmi lõppaadress>.** Heksakoodis aadressi asemel võib kasutada programmis defineeritud märgendeid.
Kogu eelnev ja ka järgnev monitorprogrammiga tutvumine salvestada käsuga **view <tee>faili_nimi.** Seejuures tuleb arvestada, et sama nimega fail kirjutatakse üle. Ka tuleb silmas pidada, et ekraani puhver on piiratud mahuga. Seepärast tuleb pikema silumise korral ekraani seisu korduvalt eri nimedega failidesse salvestada.
Käivitada programm käsuga **go <algusaadress>.** Algusaadress on kohustuslik, kui see pole eelnevalt käsuloendurisse salvestatud.
Testida käskudega **xreg** ja **dump**, kas tulemus vastab oodatule. Kui tulemus on oodatust erinev, s.t. programm ei tööta, siis laadida käsuloendurisse programmi algusaadress (**xreg pc algusaadress**). Käsuga **trace** või **trace n** asuda programmi siluma. Pikemate tsüklite või alamprogrammide korral, mis ei vaja silumist, seada paika katkestuspunktid (**break set aadress**) ja täita käsuga **go** programm kuni katkestuspunktideni.
Ülejäänud monitorprogrammi käske kasutada vastavalt vajadusele.

4.1.2. PÜSIMÄLUS OLEVATE ALAMPROGRAMMIDE KASUTAMINE

4.1.2.1. PÜSIMÄLUS OLEVAD ALAMPROGRAMMID

Püsimälus ROM olevate alamprogrammide tekstid on lisas 3 ja aadressid failis LKZ180.INC. Iga kasutaja võib endale teha suvalisi include-faile, ent monitorprogrammi alamprogrammide kasutamiseks peab ühes include-failis olema tekst failist LKZ180.INC

Kasutaja käsutuses on järgmised Toivo Vajakase koostatud alamprogrammid:
(a) CP_BC_HL - 16-bitiste märgita arvude võrdlemine. Arvud peavad olema registrites BC ja HL. Kui BC<HL, siis Carry:=1, muidu Carry:=0. Kui BC=HL, siis Zero:=1. muidu Zero:=0.
(b) DIV16 - 32-bitise märgita täisarvu jagamine 16-bitise täisarvuga. Jagatav on HL- ja DE-registrites, jagaja -BC-registris, jagatis - DE-registris ja jääk - HL - registris.
(c) MULT16 - 16-bitiste märgita täisarvude korrutamine. Tegurid peavad olema BC- ja DE-registrite, korrutis on registrites DE, HL.
(d) PRINTF - ASCII-koodis arvu saatmine kuvarile. HL-registris peab olema teisendatav arv, A-registris arvu alus (2,10, või 16). Saab kasutada ka alamprogramme, kus A-registrisse laetakse vastav alus: PRINTX (alus 16), PRINTD (alus 10), PRINTB (alus 2).
(e) SCANF1 - Klaviatuurilt ASCII-koodis sisestatud stringi teisendamine täisarvuks. A-registris peab olema arvu alus (2, 8, 10 või 16). Sisestamisel võib kasutada “+”- ja “-”-märke. Alamprogrammist väljumisel on väärtus HL-registris. Sisestatavad arvud võivad olla 0...65535 (0000H...FFFFH) või -32768...+32767. Nende piiride ületamist ei testita.
(f) KBHIT - Testitakse klaviatuuri olekut. Kui ühtegi sisestust pole, siis A=0, Zero=1, muidu A<>0, Zero=0.
(g) GETCH - Sümboli lugemine klaviatuurilt. Alamprogrammist väljumisel on sümbol A-registris.

(h) GETCHE - Sümboli lugemine klaviatuurilt ja saatmine “kajana” tagasi kuvarile. Loetud sümbol on A-registris.
(i) TOUPPER - A-registris oleva tähe teisendamine suurtäheks.
(j) GETCHEU - Sama, mis GETCHE, ent kuvarile saadetakse kajana suurtäht sisestatud tähest sõltumata.
(k) PUTCH - A-registris oleva baidi saatmine kuvarile.
(l) PUTS - Stringi saatmine kuvarile. Stringi algusaadress peab olema HL-registris.

4.1.2.2. ÜLESANDED

(1) Alamprogrammi MULT16 kasutades korrutada kaks 16-bitist märgita arvu. Esitada registrite sisu salvestus (käsud **`xreg`** ja **`view faili nimi`**)

(2) Võrrelda alamprogrammi CP_BC_HL kasutades eelmises harjutuses kasutatud tegureid. Esitada registrite sisu salvestus.

(3) Liita harjutuses (1) saadud korrutisele 5 ja jagada ta teguriga, mis harjutuse (2) põhjal oli väiksem. Esitada registrite sisu salvestus.

(4) Saata harjutuses (1) kasutatud tegurid ja saadud korrutis kuvarile kümnendsüsteemis.

(5) Saata harjutuses (3) kasutatud jagatav, jagaja, jagatis ja jääk kuvarile kümnendsüsteemis.

(6) Lugeda klaviatuurilt kaks arvu, korrutada need ja saata korrutis kuvarile,

(7) Koostada alamprogramm autorit ja programmi identifitseeriva logo kuvamiseks kasutajaprogrammi käivitamisel. Alamprogramm koostada nii, et seda saaks programmidega linkida nt. include-faili LKZ180.INC koosseisus.

(8) Koostada programm, mis kuvab logo, loeb klaviatuurilt kaks märgita kümnendarvu, saadab “kaja” kuvarile, võrdleb sisestatud arve ja saadab sõnalise tulemuse kuvarile, korrutab need arvud ja väljastab tulemuse kuvarile kümnend-, heksa- ja binaarkoodis.

4.2. PROTSESSORI SIGNAALIDE TUNDMAÕPPIMINE JA PROGRAMMIDE SILUMINE

4.2.1. PÕHIMÕTE

Monitorprogramm võimaldab tundma õppida protsessori registrite kasutamist ja valmiskirjutatud programme siluda, loogikaanalüsaatoriga saab uurida protsessori ja ka teiste seadmete signaale, tundma õppida masinatsükleid. Samuti võimaldab ta programme siluda.

Programmide silumisel, vigade avastamisel ja protsessori signaalide tundmaõppimisel loogikaanalüsaatori abil saadud ajadiagrammide alusel kontrollitakse signaalide vastavust koostatud programmi käskudele. Selleks on vaja õppida lugema ajadiagramme ja tundma ära masinatsükleid. Peab teadma käskude kirjeldusi ja koode.
Ajadiagrammide kaudu tundmatu programmi koostamisel ja silumisel on esmaseks ülesandeks koostada diagrammi kirjeldus järgmisel viisil:

Takt	/M1	/MREQ	/IORQ	/WR	/RD	Andmed (D)	Aadress (A)	Kirjeldus
1-4	0	0	1	1	0	00110001	8230H	Käsu **`LD SP,mn`** käsukoodi lugemine, A=käsukoodi aadress, D=käsukood
5	1	0	1	1	0	00110001	8230H	
6	1	1	1	1	1	11111111	00B0H	DRAM-i värskendus
7-8	1	0	1	1	1	11111111	00B0H	
9	1	1	1	1	1	11111111	8231H	

10-14	1	0	1	1	0	00000000	8231H	Käsu **LD SP,mn** esimese operandi n lugemine, A=operandi aadress, D=operand
15	1	1	1	1	1	11111111	00B1H	
16-17	1	0	1	1	1	11111111	00B1H	DRAM-i värskendus
18	1	1	1	1	1	11111111	8232H	
19-23	1	0	1	1	0	10110000	8232H	Käsu **LD SP,mn** teise operandi m lugemine, A=operandi aadress, D=operand
...	...	...	...	...	...	...	...	...

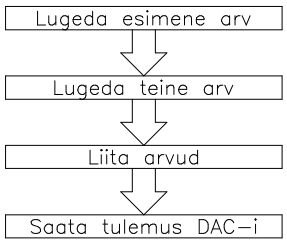
Aluseks on graafik 3 lisas 2. Ajadiagrammil võivad esineda perioodilised tsüklid, mille aadress on eelmise tsükliga võrreldes ühe võrra suurem, andmed aga samad. Kirjeldatud tsükkel on dünaamilise mälu DRAM värskendamine. Kuigi kontrolleril ei kasutata dünaamilist mälu, teostab protsessor automaatselt mäluvärskendust, kui see pole keelatud. Värskenduse saab keelata, kui seada värskenduse juhtimisregistri RCR bitt 7 (REFE) seisu 0.
Ajadiagrammi kirjeldamiseks vajalikud käskude kirjeldused ja käsukoodid on protsessori kirjelduses [6]. Nende alusel tuleb koostada programmi tekst assemblerkeeles:

```
8230H       LD    SP,B000H
            LD    IX,8246H
            LD    (IX+00),44H
            LD    A,2DH
            ADD   A,(IX+00)
            LD    BC,0100H
            OUT   (C),A
            RST   8

8246H       MEM:  DS    1
```

Sellele tuleb lisada kommentaarid:

```
8230H       LD    SP,B000H          ;Seada pinuviit
            LD    IX,8246H
            LD    (IX+00),44H       ;Lugeda üks arv
            LD    A,2DH             ;Lugeda teine arv
            ADD   A,(IX+00)         ;Liita arvud
            LD    BC,0100H
            OUT   (C),A             ;Saata tulemus DAC-ile
            RST   8

8246H       MEM:  DS    1
```

Kommenteeritud programmi järgi koostada algoritm selle kohta, mida programm teeb:



Järgnevalt tuleb programm kontrollerisse laadida, käivitada ja saada kätte ajadiagramm, mis koostatud programmi osas ei tohi esialgsest ajadiagrammist erineda. Erinevused võivad olla aga DRAM-i värskendustes.
Näiteprogrammi ajadiagrammi lugemiseks kasutati loogikaanalüsaatori sisemist takti sagedusega 33 MHz. Trigersõna:

X0000110 00100000 : kõrgemad baidid
1XXXXXXX XX0XX00X : madalamad baidid

Assemblerkeeles koostatud valmis programmi silumisel tuleks järgida järgmisi nõudeid:
1) kui on võimalik täita programmi mõtteliste osade kaupa, testida ja veenduda esialgu üksikosade õigsuses;
2) kui osa täitmisel esineb tõrkeid või vigu, täita see programmilõik sammhaaval;
3) veenduda, et programm on õigesti mällu laaditud ja mälupesad korralikud (kasutada mälu testimist);
4) tingimuslike siirete korral testida hoolikalt seatavaid lippe ja siirdekäsu vastavust algoritmile;
5) kontrollida, kas pole viga algoritmis endas;
6) kui on tulutult kaua aega viga otsitud, on praktikumi aja säästmiseks kasulik paluda kellelgi teisel programm üle vaadata, sest värske silm võib vea kiiremini leida.

4.2.2. ÜLESANDED

(1) Koostada mõnekäsuline programm, mis hõlmab mälust lugemist, mällu kirjutamist, sisend-väljundseadmetesse kirjutamist ja sealt lugemist, aritmeetika- ja loogikakäske. Käivitada koos programmiga loogikaanalüsaator. Saadud ajadiagrammi põhjal koostada käsukoodi lugemise ajadiagramm, operandide ja andmete lugemise ning kirjutamise ajadiagrammid, sisend-väljundseadmetesse kirjutamise ning sealt lugemise ajadiagrammid. Kõige lõpuks koostada käsu koodilugemise ja käsu täitmise terviklik ajadiagramm. Kontrollida koostatud diagrammide vastavust protsessori kirjelduses [6] toodule.

(2) Juhendajalt saadud ajadiagrammi järgi koostada programmi tekst . Mida programm teeb?

(3) Koostada juhendajalt saadud ajadiagrammi kirjeldus. Mida programm teeb?

(4) Koostada juhendajalt saadud ajadiagrammi kasutades programmi tekst, laadida programm kontrollerisse, siluda, käivitada programm samade algandmetega, mis olid ajadiagrammil esitatud programmil. Koos programmiga käivitada ka loogikaanalüsaator sama trigersõnaga, millega käivitati ajadiagrammil olev programm. Esitada trigersõna ja leida analüsaatori taktsagedus. Kontrollida saadud ajadiagrammi vastavust esialgsele. Erinevuste korral need esitada ja siluda programmi seni, kuni ajadiagrammid on identsed. Kui erinevusi ei õnnestu kõrvaldada, siis miks?

(5) Juhendajalt saadud ajadiagrammi põhjal koostada programmi tekst koos kommetaaridega. Koostada teksti järgi selle tegevuse algoritm, mida programm teeb.

4.3. KONTROLLERI KOOSTISOSADE TESTIMINE

4.3.1. PÕHIMÕTE

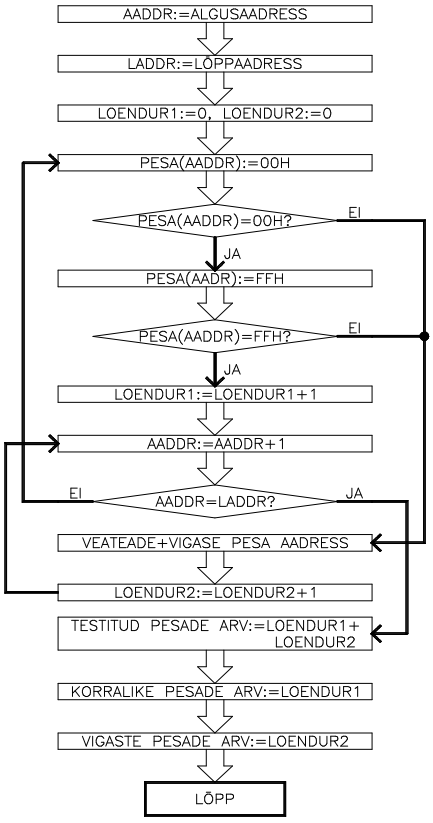
Mikroprotsessorsüsteem koosneb paljudest koostisosadest. Võib juhtuda, et mingitel süsteemisisestel või süsteemist sõltumatutel põhjustel ei tööta mõni süsteemi komponentidest normaalselt. Kui rauas või tarkvaras pole korrasoleku kontrolli ette nähtud, võivad häired põhjustada andmete kaotsiminekut või isegi kontrolleri mittekäivitumist.
Komponentide riknemist võivad põhjustada nt. tugevad voolukõikumised ja ülepinged, samuti kasutaja hooletus (viikude lühistamine jne.).
Kontrolleril on võimalik testida asünkroonset jadaväratit, rööpväratit, muutmälu teatud osa, ADC-d ja DAC-i.

4.3.2. MUUTMÄLU RAM TESTIMINE

4.3.2.1 PÕHIMÕTE

RAM-i on võimalik testida alates aadressist 8000H kuni aadressini EFFFH kaasa arvatud. Testimiseks kirjutatakse mälupessa arv 00H, loetakse pesast arv tagasi ja võrreldakse pessa kirjutatuga. Kui erinevust pole, korratakse sama arvuga FFH. Kui pesast loetud arv erines pessa kirjutatust, võib sellel

hetkel testimise katkestada ja väljastada veateate koos vigase pesa aadressiga. Võib ka lihtsalt väljastada veateate ja vigase pesa aadressi, ent jätkata testimist, kuni kogu mälu on kontrollitud.
Antud algoritmi korral joonisel 4.3.1 väljastatakse vigase pesa aadress, ent mälu testitakse lõpuni. Algandmeteks on mälu algusaadress ja lõppaadress, loendur1 loendab korralike pesi, loendur2 aga vigaseid pesi.



JOONIS 4.3.1: MUUTMÄLU TESTIMINE

4.3.2.2. ÜLESANNE

(1) Koostada algoritmi järgi joonisel 4.3.1 programm muutmälu testimiseks. Tulemuste väljastamiseks kasutada püsimälus olevaid alamprogramme. Väljastada teated testi tulemuste kohta, testitud baitide arv, korralike ja vigaste baitide arv. Kontrolleri mälu testida alates aadressist 8000H kuni EFFFH (kaasa arvatud) välja arvatud mäluosa, kus asub testprogramm ise.

4.3.3. RÖÖPVÄRATI TESTIMINE

4.3.3.1. PÕHIMÕTE

Rööpväratil on võimalik testida andmeregistreid ning sisend- ja väljundviike. Registri korrasoleku kontrollimiseks initsialiseeritakse port väljundpordiks, kirjutatakse andmeregistrisse testsõna, initsialiseeritakse port sisendpordiks, loetakse andmeregistri sisu ja võrreldakse seda porti saadetud sõnaga. Kui on erinevusi, saadetakse kuvarile veateade koos vigase biti järjenumbriga. Sisendi ja väljundi kontrollimiseks ühendatakse A-port B-pordiga, A-port initsialiseeritakse väljundpordiks, B-port - sisendpordiks ja saadetakse A-porti testsõna, mida loetakse B-pordist. Loetud sõna võrreldakse saadetud sõnaga.

4.3.3.2. ÜLESANDED

(1) Koostada programm rööpvärati portide andmeregistrite testimiseks. Tulemused väljastada kuvarile. Testida andmeregistrite korrasolekut.

(2) Koostada programm rõõpvärati sisend- ja väljundpordi kontrollimiseks. Tulemused väljastada kuvarile. Ühendada A-port B-pordiga ja testida värati korrasolekut. Kui esineb viga, siis määrata port, mis on korrast ära.

4.3.4. ANALOOG-DIGITAALMUUNDURI TESTIMINE

4.3.4.1. PÕHIMÕTE

ADC-l on olemas nii unipolaarne kui ka bipolaarne töömoodus. Mõlemas reziimis tuleb kontrollida muundamise täpsust. Selleks antakse signaaliallikast ADC sisendisse alalispinge, mis on muundamisvahemikus. Sisendpinget mõõdetakse digitaalvoltmeetriga. Seejärel käivitatakse muundur ja võrreldakse muundamisel saadud andmesõna tabelis oleva väärtusega [13, lk. 57].

4.3.4.2. ÜLESANNE

(1) Koostada programm, mis väljastab analoog-digitaalmuundi sisendis oleva pinge väärtuse kuvarile. Väärtused esitada andmesõnana. Muundist loetud andmesõna väljastada kuvarile siis, kui esineb muutus võrreldes eelneva andmesõnaga. Mõõta sisendpinge väärtused mõlema töömooduse korral vähemalt 10 erineva pingeväärtuse korral, kaasa arvatud ka minimaalne ja maksimaalne. Kas saadud tulemus vastab tabelile 2.5.1? Kui ei, siis miks? Koostada graafik, mis näitab andmesõna sõltuvust sisendpingest ja uus tabel. Kui suur sisendpinge muutus vastab ühele kvandile? Kas muundamine on lineaarne?

4.3.5. DIGITAAL-ANALOOGMUUNDURI TESTIMINE

4.3.5.1. PÕHIMÕTE

DAC on seatud unipolaarsesse töömoodusesse. Tema testimiseks tuleb kontrollida sisestatud andmesõna ja saadud väljundpinge vastavust [13, lk.57].

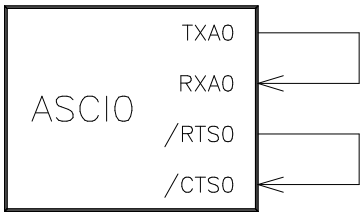
4.3.5.2. ÜLESANNE

(1) Koostada programm, mis saadaks klaviatuurilt sisestatud andmesõna DAC-i. Kontrollida digitaalvoltmeetriga muunduri väljundpinge vastavust sisestatud andmesõnale vähemalt 10 eri väärtuse korral, kaasa arvatud ka minimaalne ja maksimaalne. Koostada graafik, mis näitaks väljundpinge sõltuvust andmesõnast. Kas muundamine on lineaarne? Kas tulemus vastab tabelile 2.6.1? Kui ei, siis miks? Koostada uus tabel. Kui suur väljundpinge muutus vastab ühele kvandile?

4.3.6. ASÜNKROONSE JADAVÄRATI TESTIMINE

4.3.6.1. PÕHIMÕTE

Asünkroonse jadavärati kanalil ASCI0 saab kontrollida, kas saatmise andmeregister TDR0 on korras. Vastuvõtu andmeregistrit RDR0 ei saa kirjutamis-lugemismeetodil testida, sest RDR0 on vaid loetav register. Ühendades värati viigud joonise 4.3.2 kohaselt, saab testida värati sisendit ja väljundit. Selleks tuleb lubada saatja ja vastuvõtja, saata andmesõna käsuga **`OUT0 (TDR0),g`** TDR0 registrisse ja lugeda seejärel käsuga **`IN0 g,(RDR0)`** RDR0-registri sisu. Kui viimane on identne saadetud sõnaga, on värat korras. Saatmis- ja vastuvõtualgoritmid ning värati initsialiseerimise kirjeldus on punktis 4.7.2.



JOONIS 4.3.2: ASCI0-i TESTIMINE

4.3.6.2. ÜLESANDED

(1) Koostada programm registri TDR0 kontrollimiseks andmesõnadega FFH ja 00H. Tulemus väljastada kuvarile.

(2) Koostada programm ja algoritm ASCI0-i testimiseks, kui viigud on ühendatud joonise 4.3.2 kohaselt. Tulemused esitada kuvaril. Ühendada viigud ja testida loogikaanalüsaatori abil erinevaid saatmiskiirusi, andmeformaate ja paarsusi. Esitada testide tulemused.

4.4. ARITMEETIKATEHTED

4.4.1. ARVUDE ESITAMINE PROTSESSORIS

4.4.1.1. PÕHIMÕTE

Protsessoris Z80180 esitatakse arve otsekoodis, kui kasutatakse ainult positiivseid arve või kahendtäiendkoodis, kui kasutatakse korraga nii positiivseid kui ka negatiivseid arve. Otsekood on tavaline kahendkood, kahendtäiendkoodi korral on arvu vanim bitt märgibitt. Positiivsete arvude märgibitt on 0, negatiivsetel 1.

H	Binaar-kood	Kümnendkood, kui binaararv on	
		otsekoodis	täiendkoodis
F	1111	15	-1
E	1110	14	-2
D	1101	13	-3
C	1100	12	-4
B	1011	11	-5
A	1010	10	-6
9	1001	9	-7
8	1000	8	-8

H	Binaar-kood	Kümnendkood, kui binaararv on	
		otsekoodis	täiendkoodis
7	0111	7	+7
6	0110	6	+6
5	0101	5	+5
4	0100	4	+4
3	0011	3	+3
2	0010	2	+2
1	0001	1	+1
0	0000	0	0

H - heksakood

TABEL 4.4.1: ARVUDE ESITAMINE PROTSESSORIS

Positiivse ja negatiivse arvu teisendamiseks vastavalt negatiivseks ja positiivseks (1) inverteeritakse teisendatava arvu kõik bitid, mis on samaväärne lahutamisega arvust, mille kõik bitid on ühed, ja liidetakse saadud arvule 1 või (2) lahutatakse teisendatav arv arvust null.
Täiendkoodis oleva sama absoluutväärtusega positiivse ja negatiivse arvu summa on null.

D	(1)	D	(1)	
+ 6	0110	- 6	1010	
↓	1001	↓	0101	inverteeritud arv
	+ 1		+ 1	liita 1
- 6	1010	+ 6	0110	tulemus

D	(2)	D	(2)	
	0000		0000	arv null
+ 6	-0110	- 6	-1010	lahutada teisendatav arv
- 6	1010	+ 6	0110	tulemus

Programmi täitmise kiiruse seisukohast on meetod (2) kiirem, sest kahe tehte (lahutamine ja liitmine) asemel tuleb teha vaid 1 tehe (lahutamine).
Protsessoril on olemas käsk **CPL** 8-bitise arvu inverteerimiseks ja käsk **NEG** 8-bitise arvu teisendamiseks negatiivseks või positiivseks.

4.4.1.2. ÜLESANDED

(1) Koostada algoritm ja programm 16-bitise arvu teisendamiseks negatiivseks ja positiivseks.

4.4.2. KORRUTAMINE

4.4.2.1. PÕHIMÕTE

Protsessoril Z80180 on olemas käsk **MLT ww** 8-bitiste arvude korrutamiseks. Kui vajatakse suuremat bittide arvu, tuleb vastavad (alam)programmid ise koostada. See vajadus tekib aga kohe siis, kui soovitakse teostada signaali- ja andmetöötlust

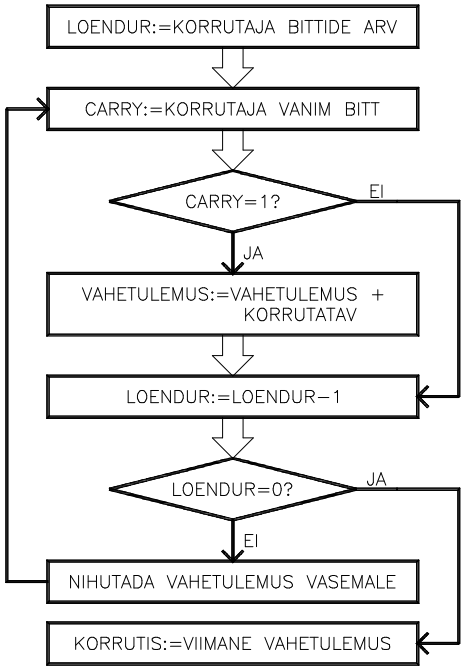
Korrutamisalgoritme on mitmeid. Siin vaadeldakse algoritmi, mille korral korrutatakse järjest korrutaja vanima kohaga läbi korrutatav, saadud vahetulemus liidetakse lõpptulemusele ja nihutatakse lõpptulemust 1 koha võrra vasemale [14, lk. 185...187]. Selliste tsüklite arv on võrdne korrutaja kohtade arvuga. Tulemuse kohtade arv on võrdne korrutaja ja korrutatava kohtade arvu summaga. Seega kahe 8-bitise arvu korrutis on maksimaalselt 16-bitine, 8- ja 16-bitise arvu korrutis on maksimaalselt 24-bitine jne.
Nt.

56D×453D=25368D	38H×1C5H=(0)6318H	01B×101B=(00)101B
99D×999D=98901D	FFH×FFFH=FEF01H	11B×111B=10101B

(D - kümnendkood; H - heksakood; B - binaarkood)

```
1011*0110 = 11D*6D=BH*6H
-------     ------------
 0110       1*0110
01100+      NIHE VASEMALE
 0000       0*0110
-------     ------------
 01100      SUMMA
011000+     NIHE VASEMALE
 0110       1*0110
-------     ------------
 011110     SUMMA
0111100+    NIHE VASEMALE
 0110       1*0110
-------     ------------
(0)1000010 = 66D=42H
```

JOONIS 4.4.1: KAHENDSÜSTEEMIS KORRUTAMINE



JOONIS 4.4.2: KORRUTAMINE VAHETULEMI NIHUTAMISEGA VASEMALE

See põhialgoritm joonisel 4.4.1 on ühesugune nii täis- kui ka murdarvude korrutamisel. Programmi kasutaja peab siis ise jälgima, et täis- ja murdosad saaksid õigesti interpreteeritud.
Nt.

```
14.5D×10.5D=(0)152.25D   E.8H×A.8H=98.4(0)H
1110.1000B×1010.1000B=10011000.01000000B
```

Algoritmis joonisel 4.4.2 esinevad tähised nagu VAHETULEMUS, LOENDUR, KORRUTAJA, KORRUTATAV, TULEMUS jne. tähistavad registreid ja/või mälupesi, kus on vastavad suurused. Korrutamisprogrammi sisendandmeteks on tegurid, väljundandmeteks korrutis.

Kui kasutatakse ka negatiivseid arve, tuleb enne korrutamist leida nende absoluutväärtused. Seejärel määrata tegurite märkide järgi korrutise märk ja see korrutamise ajaks salvestada, et pärast korrutis õigele kujule viia.
Korrutise märgi määramiseks tuleb tegurite märgibittidega teha XOR-tehe. Vastavalt selle tulemusele seada Carry-lipp (CF). Kui korrutis on positiivne, siis CF:=0, kui aga negatiivne, siis CF:=1. Carry-lipu seis tuleb korrutamise ajaks säilitada. Selleks on kõige otstarbekam viia lipuregister käsuga **PUSH AF** pinusse ja pärast korrutamist lugeda korrutise märk pinust käsuga **POP AF.** Seejuures tuleb jälgida, et pinusse salvestamiste ja pinust lugemiste arv oleks võrdne. Vastavalt Carry seisule seada korrutise märk.

4.4.2.2. ÜLESANDED

(1) Siluda alljärgnev programm MLT816 vastavalt algoritmile joonisel 4.4.2. Esitada leitud vead. Täiendada seda programmi monitorprogrammis olevate alamprogrammidega nii, et tegureid oleks võimalik sisestada klaviatuurilt ja korrutis inditseerida kuvaril.

(2) Koostada programm kahe 32-bitise märgita arvu korrutamiseks. Programm koostada nii, et seda oleks võimalik kasutada teiste programmide alamprogrammina.

```
; Programm MLT816
;Programm korrutab omavahel 2 8-bitist märgita täis- vôi murdarvu,
;kusjuures korrutis on 16-bitine. Korrutatav peab olema D-registris ja
;korrutaja E-registris. Korrutis on registris HL. Kasutatakse DE-, HL-, BC-
;ja A-registreid.

            include LKZ180.INC

MLT816:     LD    SP,STACK          ;Seada pinuviit
            LD    HL,0H             ;Vahetulemus:=0
            LD    BC,08H            ;C:=loendur
MLT8162:    LD    A,L
            SRA   E
            JP    C,MLT8163         ;Korrutaja vanim bitt = 0?
            ADD   A,D               ;Liita tulemusele korrutatav
            LD    L,A
            LD    A,H
            ADC   A,B
            LD    H,A
MLT8163:    DEC   C                 ;Vähendada loendurit
            JP    Z,MLT8161         ;Loendur = 0? Kui JA, minna lõppu
            SLA   L                 ;Ei, nihutada tulemust vasemale
            RR    H
            JP    MLT8162           ;Tagasi tsükli algusesse
MLT8161:    RST   8

            END
```

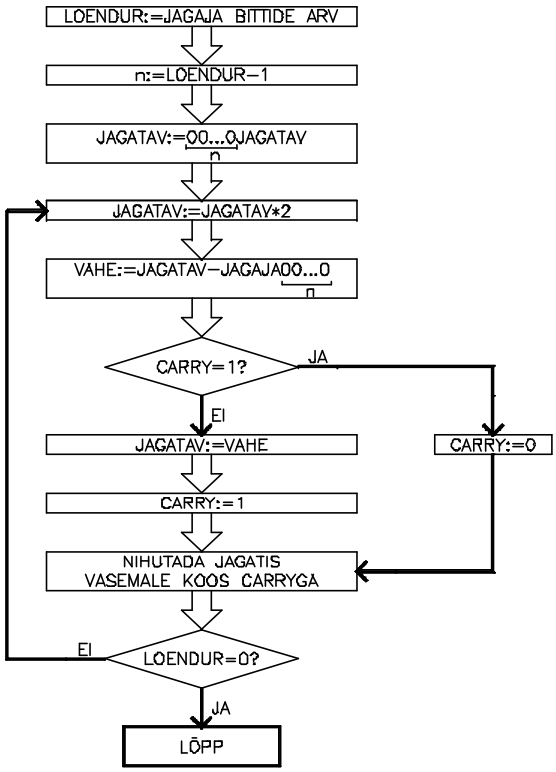
(3) Koostada algoritm märgiga arvude korrutamiseks. Koostada programm märgiga arvude korrutamiseks, kasutades silutud programmi MLT816, enda koostatud programmi 32-bitiste arvude korrutamiseks või püsimälus olevat 16-bitiste arvude korrutamise alamprogrammi MULT16.

4.4.3. TÄISARVUDE JAGAMINE

4.4.3.1. PÕHIMÕTE

Protsessoril Z80180 puuduvad käsud jagamistehete sooritamiseks. Seepärast tuleb kasutajal vajaduse korral vastavad alamprogrammid ise koostada.
Jagamine toimub binaar- ja heksakoodis sama algoritmi järgi nagu kümnendkoodiski [14, lk.187...192]:

```
0000000011010010 : 00000101 = 00101010
00000101
0000000011010010   VAHE
 00000101           NIHE
0000000011010010   VAHE
  00000101          NIHE
0000000000110010   VAHE
   00000101         NIHE
0000000000110010   VAHE
    00000101        NIHE
0000000000001010   VAHE
     00000101       NIHE
0000000000001010   VAHE
      00000101      NIHE
0000000000000000   VAHE
       00000101     NIHE
0000000000000000   JÄÄK
```



JOONIS 4.4.3: TÄISARVUDE JAGAMINE

Kui jagatav ja/või jagaja on märgiga arvud, tuleb selgitada välja jagatise ja jäägi märgid ning need jagamise ajaks salvestada. Seejärel teisendada negatiivsed arvud positiivseks, teostada jagamine ja vastavalt salvestatud märkidele teisendada jagatis ning jääk.
Märgid määratakse järgmiselt [15, lk.95]:

(+) : (+) = (+, +)
(+) : (–) = (–, +)
(–) : (+) = (–, –)
(–) : (–) = (+, –)

Sulgudes paremal pool võrdusmärki on vasemal jagatise märk ja paremal - jäägi märk. n-bitise täisarvu jagamisel m-bitise täisarvuga (n ≥ m) on jagatis maksimaalselt n-bitine (16:1=16). Alamprogrammidesse tuleb enne jagamise teostamist lisada jagaja kontroll. Kui jagaja on null, tuleb seada vealipp ja väljuda programmist. Võib lisada ka jagatava kontrolli. Kui jagatav on null, kirjutada jagatiseks kohe null ja väljuda programmist. See lisakontroll muudab nullise jagatava korral programmi kestuse lühemaks, mittenullise jagatava korral aga pisut pikemaks.

4.4.3.2. ÜLESANDED

(1) Siluda ja kommenteerida alamprogramm DIV8. Esitada avastatud vead. Millises registris on jääk?

(2) Koostada jagamisalgoritmid koos nullise jagaja kontrolliga, nullise jagatava kontrolliga ning jagatise ja jäägi märkide määramisega.

(3) Koostada programm 16-bitise märgiga täisarvu jagamiseks 16-bitise märgita täisarvuga, kontrollides seejuures kas jagatav ja jagaja on nullid või mitte.

```
; Programm DIV8
;Programm jagab 2 8-bitist märgita täisarvu. Jagatav peab olema registris
;B, jagaja aga registris C. Tulemus on registris L. Kui jagaja on 0,
```

```
;kirjutatakse L-registrisse tulemus FFH. Jääk on D-registris. Kasutatakse
;A-, BC-, HL- ja D-registreid.

            include LKZ180.INC

DIV8:       LD    SP,STACK
            LD    H,08H
            LD    D,0H
            LD    L,0H
            LD    A,0FFH
            TST   C
            JP    NZ,DIV80
            LD    L,0FFH
            JP    DIV82
DIV80:      SLA   B
            RR    D
            LD    D,A
            ADD   C
            JP    NC,DIV81
            LD    D,A
            SCF
            RL    L
            JP    DIV83
DIV81:      SCF
            CCF
            RR    L
DIV83:            DEC   H
            JP    Z,DIV82
            JP    DIV80
DIV82:            RST   8

            END
```

4.4.4. MURDARVUDE JAGAMINE

4.4.4.1. PÕHIMÕTE

Murdarvu T.M saab esitada järgmisel kujul [15, lk.12]:

$$T.M = TM \bullet A^{-n}.$$

T - murdarvu täisosa, M - murdarvu murdosa, A - arvusüsteemi alus, n - murdosa kohtade arv.
Edaspidi vaadeldakse murdarve, kus murdosa kohtade arv võrdub täisosa kohtade arvuga.
Nt:

$25.16D = 2516 \bullet 10^{-2}$	- T=25, M=16, TM=2516, A=10, n=2
$34A.D8CH = 34AD8C \bullet 16^{-3}$	- T=34A, M=D8C, TM=34AD8C, A=16, n=3
$1101.1001B = 11011001 \bullet 2^{-4}$	- T=1101, M=1001, TM=11011001, A=2, n=4

Selleks, et murdarvu T1.M1 jagamisel murdarvuga T2.M2 püsiks koma fikseerituna ja jagatise T.M täisosa kohtade arv võrduks murdosa kohtade arvuga, kasutatakse algoritmi:

$$T.M = T1.M1 / T2.M2 := [(T1M1 \bullet A^{n}) / T2.M2] \bullet A^{-n}.$$

Jagamisel tuleb arvestada, et täis- ja murdosa kohtade arv võib kahekordistuda. Nt. `FF.FFH:00.01H=FFFF.0000H, 00.01H:FF.FFH=0000.0001H`

Märgiga murdarvude jagamine toimub analoogselt märgiga täisarvude jagamisele: määrata jagatise märk, salvestada see, teisendada negatiivsed arvud positiivseteks, jagada, teisendada vastavalt märgile jagatis. Samuti tuleb kontrollida, kas jagaja on null või mitte ja võiks kontrollida ka, kas jagatav on null.

4.4.4.2. ÜLESANDED

(1) Koostada algoritm ja programm 16-bitise (8+8) märgita murdarvu jagamiseks 16-bitise (8+8) märgita murdarvuga. Seejuures kontrollida, kas jagaja on null või mitte.

(2) Koostada algoritm ja programm 16-bitise (8+8) märgiga murdarvu jagamiseks 16-bitise (8+8) märgiga murdarvuga. Seejuures kontrollida, kas jagaja on null või mitte.

4.5. KOODITEISENDUSED

4.5.1. KAHENDKOODI JA BCD-KOODI TEISENDAMINE TEINETEISEKS

4.5.1.1. PÕHIMÕTE

BCD-kood (Binary Coded Decimal) on kood, kus arvu kodeerimiseks on kasutatud korraga kahend- ja kümnendkoodi. Mitmekohaline arv kodeeritakse kümnendkoodis, kuid selle iga number esitatakse kahendkoodis. Selleks vajatakse sümboleid 0,1,2,3,4,5,6,7,8 ja 9. Kümnendarvu iga järgu esitamiseks vajatakse seega 4 bitti.

D	B	H	BCD
0	00	0	0000
1	01	1	0001
2	10	2	0010
3	11	3	0011
4	100	4	0100
5	101	5	0101
6	110	6	0110
7	111	7	0111
8	1000	8	1000
9	1001	9	1001
10	1010	A	00010000
11	1011	B	00010001
12	1100	C	00010010
13	1101	D	00010011

D	B	H	BCD
14	1110	E	00010100
15	1111	F	00010101
16	10000	10	00010110
17	10001	11	00010111
18	10010	12	00011000
19	10011	13	00011001
20	10100	14	00100000
126	1111110	7E	000100100110
127	1111111	7F	000100100111
128	10000000	80	000100101000
510	111111110	1FE	010100010000
511	111111111	1FF	010100010001
512	1000000000	200	010100010010
830	1100111110	33E	100000110000

D - kümnendkood, B - binaarkood, H - heksakood, BCD - BCD-kood

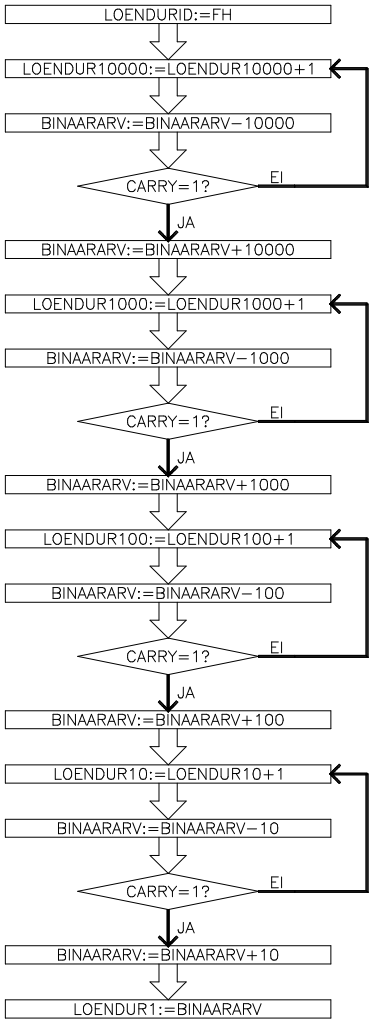
TABEL 4.5.1: KÜMNEND-, HEKSA-, BINAAR- JA BCD-KOOD

Et binaarkoodi teisendamine BCD-koodi on üsna komplitseeritud protseduur, kasutab protsessor binaarkoodi, BCD-koodi teisendatakse vajaduse järgi ainult väljundandmed. BCD-kood on sobiv binaarkoodis olevate arvude esitamiseks kümnendkujul. BCD-koodi kasutavad nt. numbertabloode dekoodrid.

4.5.1.2. ÜLESANDED

4.5.1.2.1. BINAARKOODI TEISENDAMINE BCD-KOODI [14, .lk.140...142]

Kuna protsessori registrid on 8-bitised või paaridena 16-bitised, siis on õppimiseks otstarbekas teisendada BCD-koodi 8- ja 16-bitised binaararvud. Seejuures on tulemuse esitamiseks vaja rohkem kui 8 või 16 bitti, sest binaararvule FFH vastab BCD-arv 255_{BCD} (12 bitti), binaararvule FFFFH - BCD-arv 65535_{BCD} (20 bitti). Joonisel 4.5.1 on algoritm 16-bitise binaararvu teisendamiseks, mis hõlmab ka 8-bitise arvu teisendamist. 8-bitise arvu teisendamine algab sammust **LOENDUR100:=LOENDUR100+1.** Loendurid loendavad vastavaid kümnendkohti ja nende järjestamine annab kümnendarvu.



JOONIS 4.5.1: 16-BITISE BINAARARVU TEISENDAMINE BCD-KOODI

(1) Siluda ja kommenteerida programm BINTOBCD. Esitada avastatud vead. Koostada antud programmi algoritm

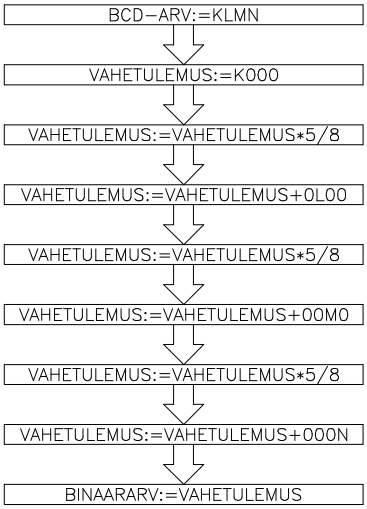
(2) Koostada programm 16-bitise binaaararvu teisendamiseks BCD-arvuks vastavalt algoritmile joonisel 4.5.1.

```
;Programm BINTOBCD
;Programm teisendab A-registris oleva binaararvu BCD-koodi. 2-baidine tulem
;on HL-registris.

            include LKZ180.INC

BINTOBCD:   LD    H,0FFH
BIN1:       DEC   H
            SUB   100
            JP    NC,BIN1
            ADD   100H
            LD    L,0FFH
BIN2:       INC   L
            ADD   10H
            JP    NC,BIN2
            SUB   10
```

```
        LD      C,A
        LD      A,L
        RLCA
        RLCA
        RRCA
        RRCA
        OR      C
        LD      L,A
        RST     8
        END
```

4.5.1.2.2. BCD-KOODI TEISENDAMINE BINAARKOODI [14, lk.142...143]

BCD-arvu teisendamisel binaarkoodi ei ületa binaararvu bittide arv BCD-arvu bittide arvu. Suurim kahekohaline e. 8-bitine BCD-arv on 99_{BCD}, millele vastab binaararv 63H. Suurim 4-kohaline e. 16-bitine BCD-arv on 9999_{BCD}, millele vastab binaararv 270FH.
Algoritm joonisel 4.5.2 on 16-bitise BCD-arvu teisendamiseks binaarkoodi, ent 8-bitise arvu teisendamine toimub analoogse algoritmi järgi. Vahe on vaid selles, et BCD-arv:=MN ja 10-ga jagamisi on vaid 1. K, L, M, ja N tähistavad BCD-arvu numbreid.



JOONIS 4.5.2: 4-KOHALISE BCD-ARVU TEISENDAMINE BINAARKOODI

(3) Siluda ja kommenteerida programm BCDTOBIN. Koostada antud programmi algoritm. Esitada avastatud vead.

(4) Koostada programm 4-kohalise BCD-arvu teisendamiseks binaarkoodi.

(5) Koostada algoritm ja programm BCD-koodis oleva n-kohalise arvu väljastamiseks kuvarile kümnendkujul. Kümnendsümbolite ASCII-koodid (antud heksakoodis):

0 - 30H	1 - 31H	2 - 32H	3 - 33H	4 - 34H	5 - 35H
6 - 36H	7 - 37H	8 - 38H	9 - 39H		

```
;Programm BCDTOBIN
;Programm teisendab 1-baidise BCD-arvu 1-baidiseks binaararvuks. BCD-arv
;peab olema A-registris ja tulemus on A-registris.

            include LKZ.INC

BCDTOBIN:   LD      B,A
            AND     0FH
            RRCA
            LD      C,A
```

```
RRCA
RRCA
SUB   C
LD    C,A
LD    A,B
XOR   0FH
ADD   C
RST   8

END
```

4.5.2. KAHENDKOODI JA GRAY KOODI TEISENDAMINE TEINETEISEKS

4.5.2.1. PÕHIMÕTE

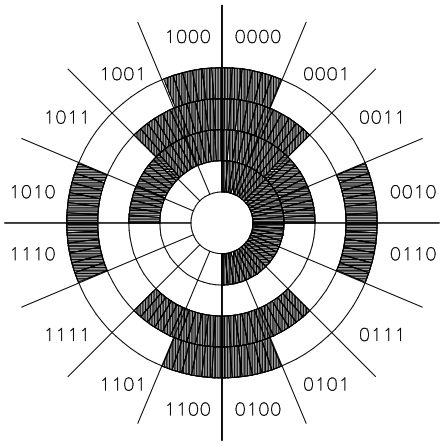
Gray kood on pisut erinev tavalistest positsioonkoodidest. Tema seos kümnend-, binaar- ja heksakoodiga on tabelis 4.5.2.

D	B	H	GRAY
0	0000	0	0000
1	0001	1	0001
2	0010	2	0011
3	0011	3	0010
4	0100	4	0110
5	0101	5	0111
6	0110	6	0101
7	0111	7	0100

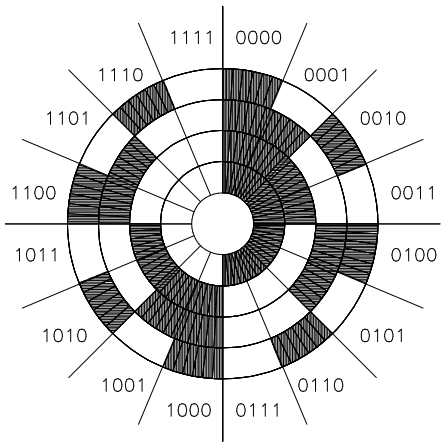
D	B	H	GRAY
8	1000	8	1100
9	1001	9	1101
10	1010	A	1111
11	1011	B	1110
12	1100	C	1010
13	1101	D	1011
14	1110	E	1001
15	1111	F	1000

TABEL 4.5.2: KÜMNEND-, BINAAR-, HEKSA- JA GRAY KOOD

Teda on sobiv kasutada nt. pöördenurga mõõtmisel. Nurk konverteeritakse otse Gray koodi, vajamata potentsiomeetrit pöördenurga määramiseks.



JOONIS 4.5.3: GRAY KOODIGA KETASANDUR



JOONIS 4.5.4: BINAARKOODIGA KETASANDUR

Gray koodi eelis tavalise binaarkoodi ees on see, et üleminekul ühe võrra suuremale või väiksemale arvule muutub Gray arvus vaid ühe biti seisund, binaarkoodis võib aga muutuda mitme biti seisund. Nt.:

```
3H = 0011B     4H=0100B      ; muutub 3 biti olek
3H = 0010G     4H=0110G      ; muutub 1 biti olek
```

Binaarkoodi teisendamine Gray koodi toimub järgmise algoritmi järgi:

$$G_n := B_n \otimes B_{n+1}\ ,\ n := 0, 1, 2, ..., N - 1.\ \text{Kui } n = N - 1, \text{ siis } B_{n+1} := 0.$$

Gray koodi teisendamine binaarkoodi toimub järgmise algoritmi järgi:

$$B_n := G_n \otimes B_n{+}1\ ,\ n := N - 1, N - 2, N - 3, ..., 0.\ \text{Kui } n = N - 1.\ \text{siis } B_{n+1} := 0.$$

G_n - n-is Gray arvu bitt, B_n - n-is binaararvu bitt, N - teisendatava arvu bittide arv, $\otimes$ - XOR-tehe

4.5.2.2. ÜLESANDED

(1) Koostada algoritm ja programm 16-bitise binaararvu teisendamiseks Gray koodi.

(2) Koostada algoritm ja programm 16-bitise Gray arvu teisendamiseks binaarkoodi.

4.6. MÄLU INDEKSEERIMINE

4.6.1. PÕHIMÕTE

Andmeid, millel on olemas teatud struktuur, on otstarbekas hoida ühe- või mitmemõõtmelistes mälutabelites, mis on analoogsed tavalise tabeliga. Kasutatavat muutmälu ei saa füüsiliselt osadeks jagada, ent saab jagada loogiliselt.

n×m-mälumaatriksit indekseeritakse järgmise algorimi järgi [14, lk.169, 173]:

ELEMENDI AADRESS := MAATRIKSI BAASAADRESS + ELEMENTIDE ARV REAS(m) × REAINDEKS × ELEMENDI SUURUS BAITIDES + VEERUINDEKS × ELEMENDI SUURUS BAITIDES

n×m-mälumaatriksitest koosnevat y-leheküljelist mälu indekseeritakse järgmiselt:

ELEMENDI AADRESS := BAASAADRESS + LEHEKÜLJE INDEKS × n × m × ELEMENDI SUURUS BAITIDES + ELEMENTIDE ARV REAS(m) × REAINDEKS × ELEMENDI SUURUS BAITIDES + VEERUINDEKS × ELEMENDI SUURUS BAITIDES

n×m-maatriksi element esitatakse kujul: E[i,j], kus i on reaindeks, j - veeruindeks. i:=0...n, j:=0...m.
n×m-maatriksitest koosneva y-leheküljelise mälu element eitatakse kujul: E[i,j,k], kus i on reaindeks, j - veeruindeks, k - leheküljeindeks. i:=0...n, j:=0...m, k:=0...y.
Indekseerimisel tuleks kontrollida, kas konkreetsed indeksid mahuvad lubatud piiridesse.

4.6.2. ÜLESANDED

(1) Koostada alamprogramm n×m-maatriksi indekseerimiseks. Alamprogramm koostada nii, et seda oleks võimalik kasutada elemendi indekseerimiseks nii mällu kirjutamisel kui ka mälust lugemisel koos võimalusega seada mõõtmeid ja sisestada konkreetse elemendi indekseid enne alamprogrammi sisenemist. Kontrollida sisestatud indeksite vastavust mõõtmetele.

(2) Koostada alamprogramm n×m-maatriksitest koosneva y-leheküljelise mälu indekseerimiseks. Alamprogramm koostada nii, et seda oleks võimalik kasutada elemendi indekseerimiseks nii mällu kirjutamisel kui ka mälust lugemisel koos võimalusega seada mõõtmeid ja sisestada konkreetse elemendi indekseid enne alamprogrammi sisenemist. Kontrollida sisestatud indeksite vastavust mõõtmetele.

4.7. ANDMEVAHETUS

4.7.1. KVITEERIMISMEETOD

4.7.1.1. PÕHIMÕTE

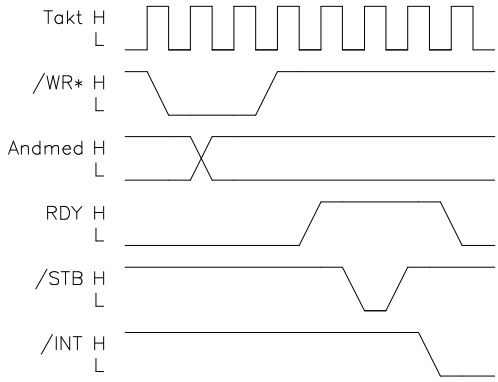
Kviteerimismeetod on andmevahetusviis, mille puhul osapooled teavitavad teineteist sellest, kas neil on andmeid saata või kas nad on suutelised andmeid vastu võtma. Andmeid võib edastada nii rööp- kui ka jadakoodis. Üks osapooltest on tavaliselt seade, mis on võimeline kahesuunaliseks andmevahetuseks (arvuti), teiseks osapooleks võib aga olla seade, mis saab ainult andmeid saata (digitaalandur), ainult vastu võtta (printer) või teostada samuti kahesuunalist andmevahetust (teine arvuti).

Kuna kontrolleril kasutatakse rööpväratit Z80PIO, siis on järgnevas protokollis aluseks võetud Z80PIO signaalid ja tööpõhimõte [7]. Konkreetselt vaadeldakse paralleelset saatmist ja vastuvõttu.

PIO-l igal pordil on 2 kviteerimissignaali ja üks katkestussignaal protsessorile. Signaalid /WR* ja /RD* on pseudo- ,s.t. kujuteldavad signaalid ning need formeeruvad järgmiselt:

/WR* := RD • /CE • •C//D • /IORQ, /RD* := /RD • /CE • C//D • /IORQ.

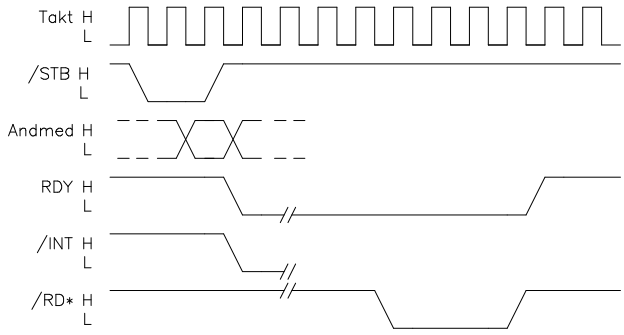
(• - AND-tehe)



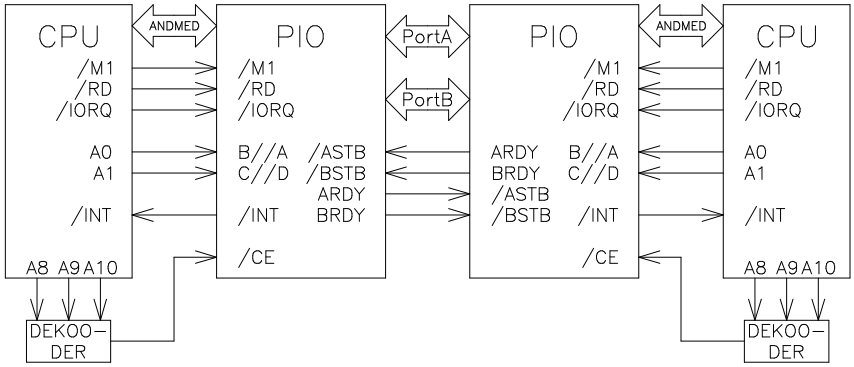
JOONIS 4.7.1: SAATMISE AJADIAGRAMM

Andmete väljastamise tsükkel initsialiseeritakse, kui protsessor täidab väljundkäsu, mis on adresseeritud rööpväratile. Andmete kirjutamiseks väljundpordi andmeregistrisse peab olema täidetud tingimus, et /WR*=0. Selle täitumisel sisestatakse andmed protsessori andmesiinilt väljundpordi andmeregistrisse. Kui /WR*-signaal läheb mitteaktiivseks, läheb taktiimpulsi järgmisel langusfrondil RDY-kviteerimissignaal aktiivseks. See annab välisseadmele märku, et andmeid tuleb lugeda. RDY jääb seni kõrgeks, kuni välisseade vastab aktiivse /STB-signaaliga. Taktiimpulsi järgmisel langusfrondil läheb RDY mitteaktiivseks. /STB tõusufront genereerib ka protsessorile katkestusnõude uute andmete saatmiseks.

Vastuvõtutsükli initsialiseerib /STB minek aktiivseks. Selle signaali järel loeb Z80PIO andmed sisendviikudelt sisendpordi andmeregistrisse. /STB-signaali tõusufront genereerib protsessorile katkestusnõude andmete lugemiseks. Taktiimpulsi langusfrondil, mis järgneb /STB mitteaktiivseks muutumisele, läheb RDY-väljundsignaal madalaks informeerimaks välisseadet, et andmed on vastu võetud, ent pole veel loetud. RDY jääb seni madalaks, kuni protsessor on andmed lugenud. Välisseadme asi on hoolitseda, et PIO-sse ei saadetaks uusi andmeid sel ajal, mil RDY on madal.



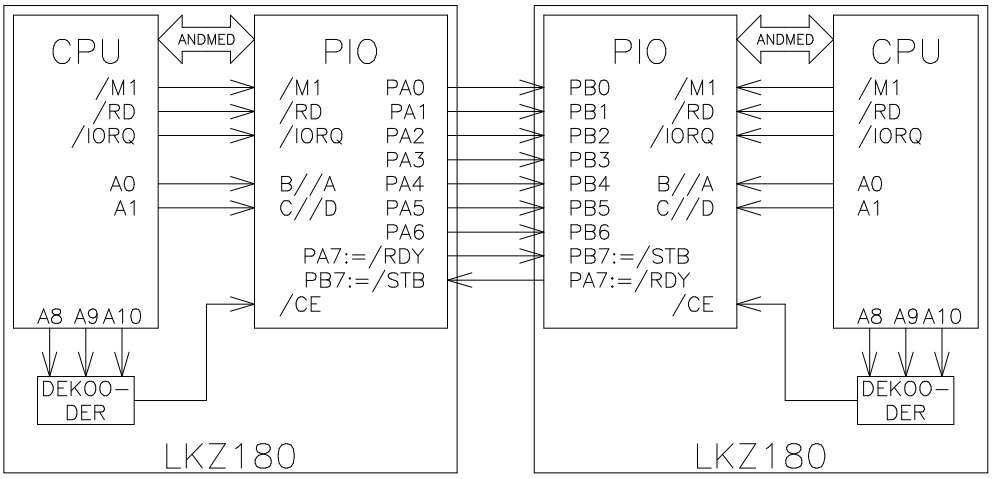
JOONIS 4.7.2: VASTUVÕTU AJADIAGRAMM



JOONIS 4.7.3: KAHE VÄRATI STANDARDÜHENDUS KVITEERIMISE KORRAL

4.7.1.2. KONTROLLERI EHITUSEST TULENEVAD ERIPÄRAD

Kontrolleril on olemas rööpvärat, kuid selle juhtimissignaale ei saa kasutada, sest need pole kuplungitesse ühendatud. Värati portide ja kuplungite vahele on lülitatud ühesuunalised puhvrid nii, et A-port töötab väljund- ja B-port sisendpordina.



JOONIS 4.7.4: KONTROLLERITE ÜHENDAMINE

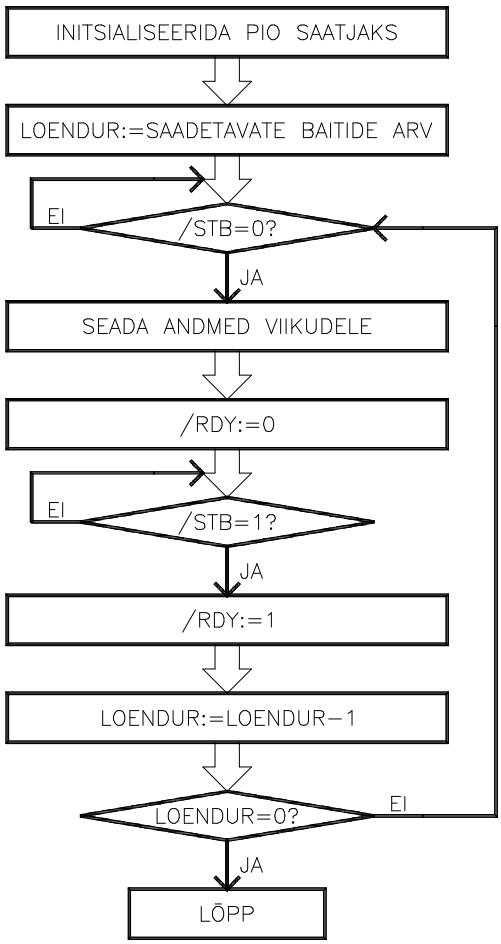
Saatjaks defineeritud kontroller kasutab A-pordi ühte viiku juhtimissignaali väljundina ja ülejäänud seitset viiku andmeväljunditena. B-pordil kasutatakse vaid ühte viiku juhtimissignaali sisendina.
Vastuvõtjaks defineeritud kontroller kasutab A-pordi ühte viiku juhtimissignaali väljundina, B-pordi ühte viiku juhtimissignaali sisendina ja ülejäänud seitset viiku andmesisenditena.
Kuna standardne andmesõna on 8-bitine siis tuleb see saata kahes osas alates vanematest bittidest. Näiteprogrammis kasutatakse formaati 4+4, aga see võib olla ka erinev. Oluline on, et saatja ja vastuvõtja kasutaksid sama formaati.

4.7.1.3. ÜLESANDED

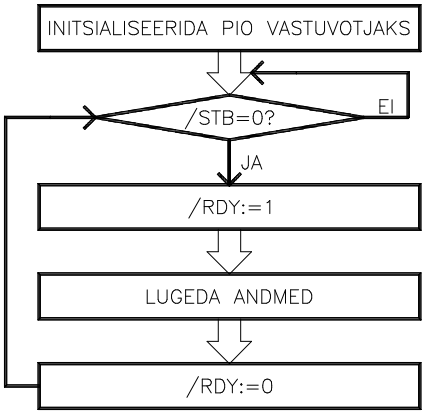
(1) Siluda alamprogramm SENDDATA ja saatmisprogramm SENDPIO. Esitada avastatud vead. Koostada ise alamprogramm INITPIO rööpvärati initsialiseerimiseks. Koostada algoritmi järgi joonisel 4.7.5 vastuvõtuprogramm, millega saaks ´vastu võtta andmeid, mis on saadetud programmiga SENDPIO. Saata koostatud programme kasutades ühest kontrollerist teise n baiti. Kontrollida programmide õigsust loogikaanalüsaatoriga siinil olevaid signaale ja monitorprogrammiga mälu sisu testides. Esitada signaalide ajadiagrammid. Kui on lahknevusi signaalide käigus võrreldes joonistel 4.7.1 ja 4.7.2 esitatuga, siis miks? Arvutada ühe baidi saatmise aeg ja saatmiskiirus boodides.

```
;Alamprogramm SENDDATA
;Alamprogramm väljastab rööpvärati kaudu paralleelse 4-bitise andmesõna.
;Andmed peavad olema H-registris. Kasutatakse H-, BC- ja A-;registreid.
;PB7=/STB, PA7=/RDY

EQU   RDY0:=00H
EQU   RDY1:=80H

SENDDATA:   LD    BC,PIOBD          ;Lugeda andmed
SENDDATA1:  IN    A,(C)
            JP    C,SENDDATA1       ;/STB=0? Kui pole, kontrollida uuesti
            LD    BC,PIOAD          ;/STB=0
            LD    A,RDY0            ;/RDY:=0
            ADD   A,H               ;Andmed ja /RDY=0
            OUT   (C),A             ;Saata välja
            LD    BC,PIOBD          ;Lugeda andmed
            NOP                     ;Viivis
            NOP
            NOP
            NOP
SENDDATA2:  OUT   (C),A
            RRA                     ;/STB=0? Kui pole, kontrollida uuesti
            JP    C,SENDDATA2
            LD    BC,PIOAD
            LD    A,RDY1
            OUT   A,(C)             ;/RDY:=1

            RET

;Programm SENDPIO
;Programmi kasutab protsessor, mis rööpväratit imiteerides saadab
;kviteeringut kasutades andmed ühest kontrollerist teise. Kasutatav formaat
;on 4+4. Andmed peavad olema mälus alates pesast SDATA. DE-registris peab
;olema ülekantavate baitide arv. Programm kasutab alamprogramme INITPIO ja
;SDL, mis on failis HANDPIO.PRC. Kasutatakse DE-, IX- ja A-registreid.

            include     LKZ.INC

SENDPIO:    LD    SP,STACK          ;Seada pinuviit
            CALL  INITPIO           ;Initsialiseerida PIO, /OBF:=1
            LD    DE,SCOUNT         ;Ülekantavate baitide arv
            LD    IX,SDATA          ;Andmeviit
SENDPIO1:   LD    L,(IX+0)
            CALL  SDL               ;Valmistada ette esimesed 4 bitti
            CALL  SENDDATA          ;Saata esimesed 4 bitti
            CALL  SDL               ;Valmistada ette viimased 4 bitti
```

```
            CALL  SENDDATA          ;Saata viimased 4 bitti
            DEC   IX                ;Suurendada andmeviita
            INC   DE                ;Vähendada loendurit
            LD    A,00H             ;On veel andmeid saata
            CP    E
            JP    NZ,SENDPIO1       ;Ja, minna SENDPIO1
            CP    D
            JP    NZ,SENDPIO1       ;Ja, minna SENDPIO1
            RST   8                 ;Ei, väljuda programmist

            include HANDPIO.PRC

SCOUNT      EQU   4                 ;Saadetavate baitide arv
SDATA:      DB    'O','K','O','K'   ;Andmed

            END
```



JOONIS 4.7.5: SAATMISALGORITM



JOONIS 4.7.6: VASTUVÕTUALGORITM

4.7.2. ASÜNKROONNE JADAKOODIS ANDMEVAHETUS

4.7.2.1. PÕHIMÕTE

Asünkroonseks jadakoodis andmevahetuseks kasutatakse ASCI0-i [6]. Andmevahetuseks tuleb ASCI0 initsialiseerida, kirjutatakse tema registritesse vajalikud juhtimissõnad.
STAT0-is on ainsateks kirjutatavateks bittideks TIE (saatmiskatkestuse lubamine) ja RIE (vastuvõtukatkestuse lubamine). Nende katkestuste keelamiseks tuleb STAT0-i kirjutada käsuga **`OUT (STAT0),g`** juhtimissõna 00H, mõlema katkestuse lubamiseks aga juhtimissõna 09H.
CNTLB0-registriga tuleb esmalt määrata kindlaks, kas multiprotsessormoodi kasutatakse või mitte. Kuna kontrolleril seda moodi ei kasutata, peab MP-bitt olema 0. Sellisel juhul ei oma MPBT-biti olek mingit tähtsust. Paarsus määratakse PEO-bitiga. Kui PEO=0, on valitud paaris paarsus, PEO=1 korral aga paaritu paarsus. Kui andmeformaat ei sisalda paarsusbitti, pole PEO olek oluline. /CTS/PS-, DR-, SS2-, SS1- ja SS0-bitid määravad saatmiskiiruse vastavalt tabelile 4.7.6.

bitt			
7	RDRF	Receive Data Register Full	R
6	OVRN	Overrun Error	R
5	PE	Parity Error	R
4	FE	Framing Error	R
3	RIE	Receive Interrupt Enable	R/W
2	DCD0	Data Carrier Detect	R
1	TDRE	Transmit Data Register Empty	R
0	TIE	Transmit Interrupt Enable	R/W

TABEL 4.7.1: ASCI0 OLEKUREGISTER STAT0 (S/V-AADRESS = 04H)

bitt	7	6	5	4	3	2	1	0	
	X	X	X	X	0	X	X	0	mõlemad katkestused keelatud
	X	X	X	X	0	X	X	1	vastuvõtukatkestus keelatud, saatmiskatkestus lubatud
	X	X	X	X	1	X	X	0	vastuvõtukatkestus lubatud, saatmiskatkestus keelatud
	X	X	X	X	1	X	X	1	mõlemad katkestuse lubatud

TABEL 4.7.2: STAT0-I JUHTIMISSÕNA

bitt			
7	MBPT	Multiprocessor Bit Transmit	R/W
6	MP	Multiprocessor Mode	R/W
5	/CTS/PS	Clear To Send/Prescale	R/W
4	PEO	Parity Even Odd	R/W
3	DR	Divide Ratio	R/W
2	SS2	Source/Speed Select 2	R/W
1	SS1	Source/Speed Select 1	R/W
0	SS0	Source/Speed Select 0	R/W

TABEL 4.7.3: ASCI0 JUHTIMISREGISTER CNTLB0 (S/V-AADRESS = 02H)

Saatmiskiirus saadakse protsessori Z80180 taktsageduse 6.144 MHz jagamisel teatud arvudega, mille määravad ära CNTLB0-registri bitid vastavalt tabelile 4.7.6:

6.144 MHz : jagaja1 : jagaja2 : jagaja3 = 6.144 MHz : üldjagaja = kiirus (boodi)

Saadud juhtimissõna tuleb käsuga **`OUT (CNTLB0),g`** kirjutada CNTLB0 registrisse.

bitt	7	6	5	4	3	2	1	0	
	X	0	PS	1	DR	SS2	SS1	SS0	paaritu paarsus
	X	0	PS	0	DR	SS2	SS1	SS0	paaris paarsus

TABEL 4.7.4: CNTLB0-I JUHTIMISSÕNA

bitt			
7	MPE	Multiprocessor Mode Enable	R/W
6	RE	Receiver Enable	R/W
5	TE	Transmitter Enable	R/W
4	/RTS0	Request To Send Channel0	R/W
3	MPBT/EFR	Multiprocessor Bit Receive/Error Flag Reset	R/W
2	MOD2	ASCI Data Format Mode 2	R/W
1	MOD1	ASCI Data Format Mode 1	R/W
0	MOD0	ASCI Data Format Mode 0	R/W

TABEL 4.7.5: ASCI0 JUHTIMISREGISTER CNTLA0 (S/V-AADRESS = 00H)

PS	JAGAJA1	DR	JAGAJA2	SS2	SS1	SS0	JAGAJA3	ÜLDJAGAJA	KIIRUS
0	10	0	16	0	0	0	1	160	38400
				0	0	1	2	320	19200
				0	1	0	4	640	9600
				0	1	1	8	1280	4800
				1	0	0	16	2560	2400
				1	0	1	32	5120	1200
				1	1	0	64	10240	600
		1	64	0	0	0	1	640	9600
				0	0	1	2	1280	4800
				0	1	0	4	2560	2400
				0	1	1	8	5120	1200
				1	0	0	16	10240	600
				1	0	1	32	20480	300
				1	1	0	64	40960	150

TABEL 4.7.6: SAATMISKIIRUSE VALIMINE

Kuna multiprotsessormoodi ei kasutata, pole CNTLA0-registri MPE-biti seisund oluline. EFR-bitt tuleb nullida, et nullida vealipud. Et partner ei hakkaks enne initsialiseerimise lõppu andmeid saatma, tuleb /RTS0-bitt seada üheks. See muudab /RTS_0-signaali mitteaktiivseks. Alles siis, kui ollakse valmis andmeid vastu võtma, nullida /RTS0. Saatja on lubatud, kui TE=1, Vastuvõtja on lubatud, kui RE=1. Nende bittide nullimine keelab vastavalt saatja ja vastuvõtja ning käimasolev saatmine ja/või vastuvõtt katkestatakse. Bittidega MOD2, MOD1 ja MOD0 määratakse kindlaks andmeformaat vastavalt tabelile 4.7.7. Saadud andmesõna tuleb käsuga **`OUT (CNTLA0), g`** kirjutada CNTLA0-registrisse.

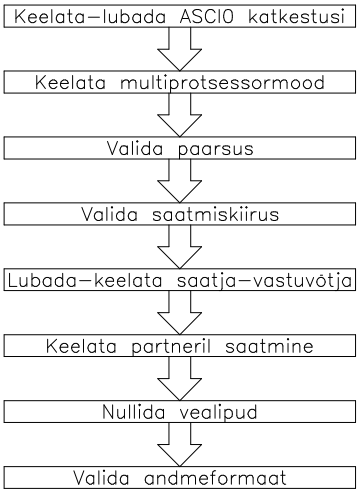
MOD2	MOD1	MOD0	ANDMEFORMAAT
0	0	0	stardibitt + 7 andmebitti + 1 stopibitt
0	0	1	stardibitt + 7 andmebitti + 2 stopibitti
0	1	0	stardibitt + 7 andmebitti + paarsusbitt + 1 stopibitt
0	1	1	stardibitt + 7 andmebitti + paarsusbitt + 2 stopibitti
1	0	0	stardibitt + 8 andmebitti + 1 stopibitt
1	0	1	stardibitt + 8 andmebitti + 2 stopibitti
1	1	0	stardibitt + 8 andmebitti + paarsusbitt + 1 stopibitt
1	1	1	stardibitt + 8 andmebitti + paarsusbitt + 2 stopibitti

TABEL 4.7.7: ANDMEFORMAADID

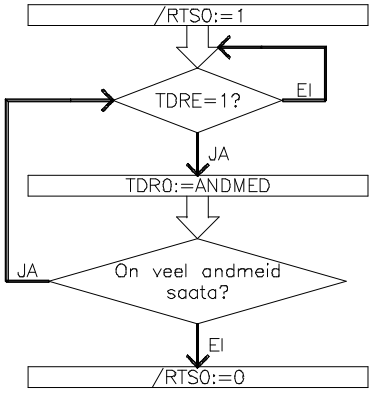
bitt	7	6	5	4	3	2	1	0	
	X	1	1	1	0	MOD2	MOD1	MOD0	vastuvõtja ja saatja lubatud, partneril mitte saata
	X	0	1	1	0	MOD2	MOD1	MOD1	vastuvõtja keelatud, saatja lubatud, partneril mitte saata
	X	1	0	1	0	MOD2	MOD1	MOD0	vastuvõtja lubatud, saatja keelatud, partneril mitte saata
	X	0	0	1	0	MOD2	MOD1	MOD0	vastuvõtja ja saatja keelatud, partneril mitte saata

TABEL 4.7.8: CNTLA0-I JUHTIMISSÕNA

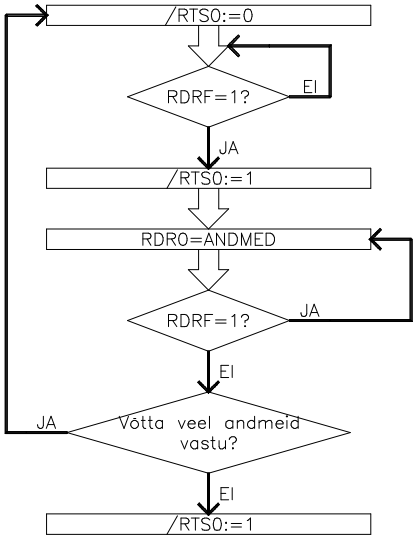
Kui partneril on saatmine keelatud, s.t. /RTS0-bitt=1, siis saatmiseks tuleb lugeda **`käsuga IN0 g,(STAT0)`** STAT0-register ja testida, kas TDRE-bitt on 0 või 1. Kui TDRE=1, siis on TDR0-register tühi ja andmeid saab saata. Saatmiseks tuleb väljastatavad andmed kirjutada TDR0-registrisse. Vastupidisel juhul tuleb STAT0-registrit lugeda ja TDRE-bitti testida seni, kuni TDRE=1.
Vastuvõtuks tuleb /RTS0-bitt nullida, lugeda STAT0-register ja testida, kas RDRF=1. Kui see on nii seada /RTS0:=1, et keelata saatmine, lugeda RDR0-registrist andmed, testida uuesti RDRF-bitti, sest saatmise keelamise ajal võis parner parajasti andmeid saata. Kui andmed on olemas, lugeda need ja seada /RTS0:=0, et lubada saatmist.
Kahepoolne andmevahetus koos saatmise ja vastuvõtuga käib algoritmi järgi joonisel 4.7.10.



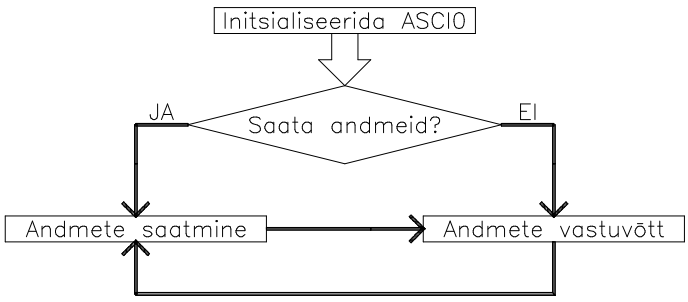
JOONIS 4.7.7: ASCI0-I INITSIALISEERIMINE



JOONIS 4.7.8: SAATMINE



JOONIS 4.7.9: VASTUVÕTT



JOONIS 4.7.10: KAHEPOOLNE ANDMEVAHETUS

4.7.2.2. ÜLESANDED

(1) Koostada programm ASCI0 initsialiseerimiseks ja andmete saatmiseks. Kasutades valmis programmi andmete vastuvõtuks, saata andmed ühest kontrollerist teise.
(2) Koostada programm ASCI0 initsialiseerimiseks ja andmete vastuvõtuks. Kasutades valmis programmi andmete saatmiseks, võtta vastu teisest kontrollerist saadetud andmed.

(3) Koostada programm andmete vastuvõtuks ja saatmiseks. Programmi kasutades teostada dupleksandmevahetus kahe kontrolleri vahel.

(4) Koostatud programme kasutades testida loogikaanalüsaatori abil saatmiskiirusi, andmeformaate, paarsusi, modemi juhtimissignaale ja andmeviike. Esitada loogikaanalüsaatori ajadiagrammid, andmesõnade diagrammid erinevate andmeformaatide ja paarsuste korral, juhtimissignaalide ja andmeviikude ajadiagrammid.

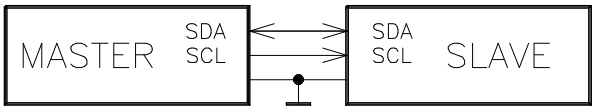
(5) Koostada programm, mis võimaldab klaviatuurilt sisestatud andmed inditseerida teise kontrolleriga ühendatud kuvaril. Andmete edastamiseks kasutada ASCI0-i. Programmi alguses peab saama valida lugemis- või kirjutamisreziimi, programmi käigus järgnegu ühel kontrolleril pärast saatmisreziimi automaatselt lugemisreziim ja vastupidi ning teisel kontrolleril vastupidi või lõpetada programm mõlemal kontrolleril. Programmi juhitagu teatud ASCII-sümbolitega kirjutamisreziimis olles.

4.7.3. ANDMEVAHETUS I2C-PROTOKOLLI JÄRGI

4.7.3.1. PÕHIMÕTE

I2C-protokolli hakkas 1980-te alguses välja töötama firma Philips Semiconductors, et lihtsamalt ühendada protsessorit kiipidega TV-süsteemis [16]. Tavalises süsteemis vajatakse laia siini aadresside ja andmete edastamiseks. Samuti vajatakse mitut aadressidekoodrit, et erinevaid kiipe adresseerida. Kuna selline struktuur on TV-, audio- ja videosüsteemides tülikas, siis oli Hollandis Eindhovenis asuva Philips Labs'i uuringute tulemuseks 2 signaalijuhtmega siin, mida hakati nimetama I2C-siiniks.
Praegu ei kasutata I2C-siini mitte ainult TV-, video- ja audiosüsteemides, vaid laialdaselt andmevahetuseks väga erinevate seadmete vahel: arvuti andmevahetus perifeerseadmetega nagu hiir, monitor, klaviatuur, printer, mälu, andmevahetus teise arvutiga jne.

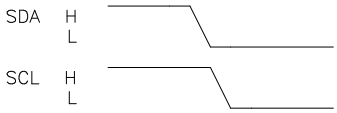
I2C-siin koosneb kahesuunalisest andmeliinist SDA, ühesuunalisest taktiliinist SCL ja maaliinist. Igal siinil oleval seadmel on oma aadress, olgu ta siis arvuti, vedelkristalltabloo, mälu jne. Olenevalt oma ehitusest saab iga siinil olev seade olla nii saatja kui ka vastuvõtja.



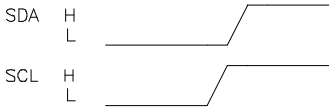
JOONIS 4.7.11: I2C-SIIN

Siini MASTER (juhtkontroller) on kiip, mis juhib siinil andmevahetust. Samal ajal on teised seadmed siinil SLAVE-id (alluvad kontrollerid).
Põhimõtteliselt võib olla siinil rohkem kui üks MASTER. Siis nimetatakse siini MULTIMASTER-siiniks. MASTER initsialiseerib siinil andmevahetuse ja tal on kontroll taktisignaali SCL üle. Taktisignaale genereerib alati MASTER. Andmeid saadetakse ja võetakse vastu jadakoodis.

Kui MASTER-iks on kasutaja poolt programmeeritav seade (arvuti) ja SLAVE-iks seade, millel on juba rauas I2C-protokoll paika pandud, pole kasutajal muud muret kui kirjutada valmis programm MASTER-ile. SLAVE tegutseb siis vastavalt oma ehitusele. Et mikroprotsessorite praktikumis tuleb teostada andmevahetust kahe kontrolleri vahel, siis tuleb koostada programm ka SLAVE-ile. Seepärast vaadeldakse järgnevalt I2C-protokolli programmilist poolt.
MASTER, kes saadab siinile START-tingimuse, viib kõigepealt SDA madalaks. Seejärel viib ta SCL-i madalaks.

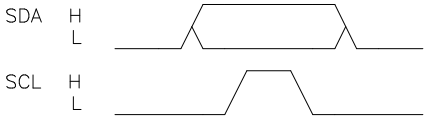


JOONIS 4.7.12: START-TINGIMUS



JOONIS 4.7.13: STOPP-TINGIMUS

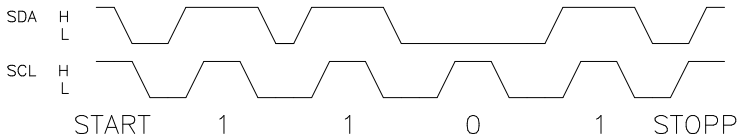
STOPP-tingimus on justkui eelneva peegelpilt. MASTER vabastab kõigepealt SCL-i, mis läheb seega kõrgeks ja seejärel SDA. Enne START-i ja peale STOPP-i on mõlemad signaalid kõrged. START on siinil olevatele seadmetele signaaliks, et hakkab toimuma andmevahetus, STOPP aga annab teada, et andmevahetus on lõppenud.



JOONIS 4.7.14: BITI SEADMINE SIINILE

MASTER seab andmebiti saatmiseks kõigepealt SDA vastavalt andmebiti olekule. Siis vabastab ta mõneks ajaks SCL-i, ootab natuke ja viib siis enne SDA oleku muutmist SCL-i jälle madalaks. Selline tegevus on vajalik seetõttu, et mitte kõik seadmed pole dünaamilise sisendiga.

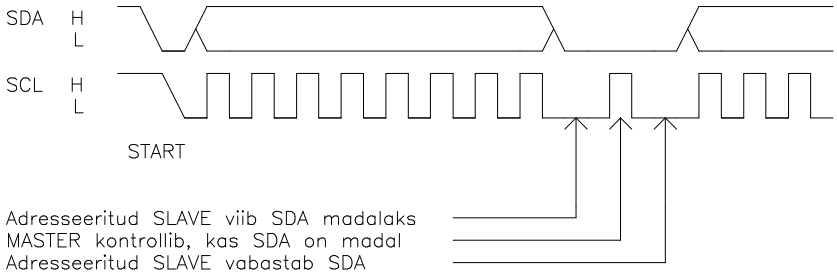
Andmed peavad olema siinil muutumatult kogu selle aja, mil taktiimpulss on kõrge. Ainus olukord, mil SDA olek võib takti kõrge seisundi ajal muutuda, on START- või STOPP-tingimuse korral. Kuna minimaalset taktsagedust pole ette antud, võib andmevahetust teostada endale meelepärase ja sobiva kiirusega. Seega näeb ülekanne välja järgmiselt:



JOONIS 4.7.15: ÜLEKANNE SIINIL

KÕIK siinile saadetud andmesõnad **PEAVAD OLEMA** muutumatu bittide arvuga. Andmesõna saadetakse alati vanimast bitist alates. Ühe andmevahetustsükli jooksul saadetud andmesõnade arv pole limiteeritud. Andmevahetustsükkel on kõik see, mis toimub siinil START- ja STOPP-tingimuse vahel. Andmevahetuse võib igal hetkel STOPP-signaaliga katkestada.

Kui SLAVE on adresseeritud või ta on andmebaidi vastu võtnud, peab ta saatma vastussignaali ACK. Selleks vabastab MASTER SDA ja seejärel ka SCL-i. Nüüd kontrollib MASTER, kas SLAVE on SDA



JOONIS 4.7.16: ANDMEVAHETUS SIINIL

madalaks seadnud. Kui on, siis on MASTER-il ACK-signaal käes. SDA peab olema ACK-i ajal stabiilselt madal enne, kui SCL läheb kõrgeks ja ta peab jääma madalaks kogu selle aja jooksul, mil SCL on kõrge. Kui SLAVE on viinud SDA madalaks, kontrollib seda MASTER SCL-i kõrge oleku ajal ja siis viib MASTER SCL-i taas madalaks. Alles nüüd võib SLAVE vabastada SDA.
Tavaliselt kasutab MASTER ACK-i ootamisel viiteaega. Kui selle jooksul pole ACK saabunud, saadab MASTER STOPP-signaali ja jätkab oma tegevust.

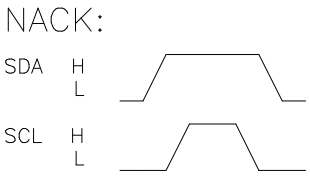
Kui SLAVE on adresseeritud ja saatnud vastuseks ACK-i, sõltub kõik edasine enne STOPP-i sellest, milline on adresseeritud kiip ja millist andmevahetust soovitakse temaga teostada. Kiipi võib saata ühe või mitu baiti, lugeda temast ühe või mitu baiti või kasutada kombineeritud andmeformaati.

Kui SLAVE on ACK-iga vastanud, tuleb lihtsalt järgmised 8 bitti siinile saata. Seejärel oodatakse ACK-i. Nüüd võib saata STOPP-i või uue andmebaidi jne.

JOONIS 4.7.17: ÜHE BAIDI SAATMINE SIINILE

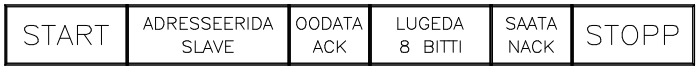
JOONIS 4.7.18: MITME BAIDI SAATMINE SIINILE

SLAVE-st lugemine toimub analoogselt saatmisega. Erinevused on vaid SDA juhtimises ja ACK-is. MASTER genereerib START-tingimuse, saadab siinile seadme aadressi ja ootab ACK-i. Edasi vabastab MASTER SDA, mida SLAVE viib vastavalt andmebittidele madalaks iga MASTERI genereeritud taktisignaali ajal. Kui kõik 8 bitti on loetud, peab MASTER saatma ACK-i SLAVE-le.
Viimase baidi lugemisel peab MASTER genereerima ACK-i asemel NACK-i, millega antakse SLAVE-le teada lugemise lõpetamisest. NACK näeb välja nagu normaalne ACK selle erandiga, et SDA jääb kõrgeks.



JOONIS 4.7.19: NACK-SIGNAAL

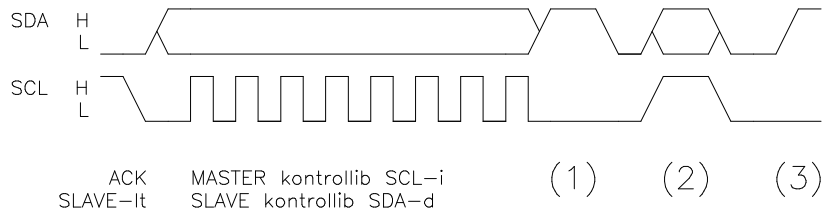
On kaks erinevat NACK-signaali:
1) MASTER saadab siinile SLAVE-i aadressi ja ootab ,et SLAVE saadaks kinnituseks ACK-i. Kui seda ei tule, on tegemist NACK-iga.
2) Kui MASTER tahab lugemist lõpetada, saadab ta ACK-i asemel enne STOPP-i NACK-i. NACK-i saatmisega öeldakse SLAVE-le, et see peab SDA vabaks andma kuni ta detekteerib kas START- või STOPP-tingimuse. See peab kindlustama, et MASTER saaks anda kas STOPP- või START-signaali.



JOONIS 4.7.20: ÜHE BAIDI LUGEMINE

(1) SLAVE vabastab SDA, mis läheb kõrgeks.

(2) a) MASTER annab ACK-i SLAVELE:
MASTER viib kõigepealt SDA madalaks, siis annab taktiimpulsi ja seejärel vabastab SDA. See on MASTER-i ACK-signaaliks SLAVE-le, et MASTER on kõik 8 bitti kätte saanud.
b) MASTER annab NACK-i SLAVE-le:
MASTER jätab SDA kõrgeks ja annab taktiimpulsi. See on MASTER-i signaal SLAVE-le, et lugemistsükkel on lõpetatud.
(3) Edasine sõltub MASTER-ist. Viimane võib saata STOPP-i. Kui (2) oli ACK, võib lugeda uue baidi või saata START-i. Kui (2) oli NACK, peaks siinile saadetama kas STOPP või uus START.



JOONIS 4.7.21: SIGNAALID SIINIL LUGEMISE AJAL

JOONIS 4.7.22: MITME BAIDI LUGEMINE

SLAVE-i aadressiga määratakse ära, kas MASTER tahab SLAVE-i kirjutada või sealt lugeda. SLAVE-i aadressi noorim bitt on 0, kui MASTER tahab SLAVE-i kirjutada ja 1, kui MASTER tahab lugeda. Seega on paarisaadressid kirjutamisaadressid ja paaritud lugemisaadressid. Ühel seadmel on kaks aadressi, mis erinevad noorima biti oleku poolest.
Nt:
kirjutamisaadress: 01000000B = 40H = 64D
lugemisaadress: 01000001B = 41H = 65D
(B - binaarkood, H - heksakood, D - kümnendkood)

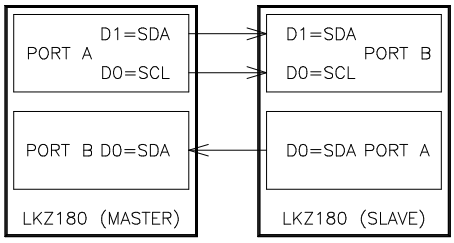
Kombineeritud formaati kasutatakse peamiselt mäludega suhtlemisel. Olgu siinil nt. 128-baidine mälu, millest soovitakse lugeda 84. baiti. Tavalise andmevahetuse puhul tuleks lugeda enne 83 eelnevat baiti, mis võtab liiga palju aega. Selle vältimiseks on 2 võimalust. Võib kirjutada SLAVE-i baidi, mis osutab mälupesale, mida soovitakse lugeda. Seejärel alustatakse tavalist lugemistsüklit. Elegantsem viis on kasutada kombineeritud formaati.

JOONIS 4.7.23: KOMBINEERITUD ANDMEFORMAAT

Alustatakse tavalist kirjutamistsüklit. Adresseeritakse SLAVE ja oodatakse ACK-i. ACK-i saabumisel saadetakse bait, mida mälu interpreteerib aadressiviidana (nii I2C-mälu töötab). Oodatake ära ACK SLAVE-lt, genereeritakse uus START ja saadetakse SLAVE-i lugemisaadress. Oodatakse ära ACK ja võetakse vastu andmebait. Seega ollakse nüüd tavalises lugemisreziimis. Edasi võib saata STOPP-i või jätkata SLAVE-ist lugemist. Mäluseadmed suurendavad automaatselt oma aadressiviita.
Võib genereerida ka uue START-i, adresseerida SLAVE-i saatmisreziimi, saata uue andmebaidi, mida SLAVE peab aadressiviidaks, saata uue START-i, siseneda lugemisreziimi jne.
Kergesti on teostatav järgmine andmevahetus: START, adresseerida SLAVE, saata aadressiviit, lugeda, lugeda, seada aadressiviit, lugeda, seada aadressiviit, kirjutada, seada aadressiviit, lugeda, STOPP.

4.7.3.2. KONTROLLERI EHITUSEST TULENEVAD ERIPÄRAD

Eelpool mainiti, et I2C-siini moodustavad vaid 2 liini, kusjuures SDA on kahesuunaline. Kontrolleril kasutatakse I2C-andmevahetuse teostamiseks rööpväratit. Kuna viimasel on ühesuunalised pordid, ei ole võimalik kasutada vaid kahte liini. A-pordi kaudu saadab MASTER takti- ja SDA-signaali, B-pordi kaudu võtab vastu SDA-signaali. SLAVE võtab B-pordi kaudu vastu SDA- ja taktisignaali, A-pordi kaudu saadab SDA-signaali.



JOONIS 4.7.24: KONTROLLERITE ÜHENDAMINE I2C-ANDMEVAHETUSEKS

4.7.3.3. ÜLESANDED

(1) Koostada algoritm ja programm n baidi saatmiseks MASTER-kontrollerist SLAVE-kontrollerisse. Andmed kirjutada SLAVE-kontrollerisse ühekordselt saadetud baasaadressist alates. SLAVE-kontrollerisse laadida olemasolev vastuvõtuprogramm. Teostada andmevahetus.

(2) Koostada algoritm ja programm n baidi lugemiseks SLAVE-kontrollerist MASTER-kontrollerisse. Andmed lugeda SLAVE-kontrollerist ühekordselt saadetud baasaadressist alates. SLAVE-kontrollerisse laadida olemasolev saatmisprogramm. Teostada andmevahetus.

(3) Koostada algoritm ja programm andmete saatmiseks SLAVE-kontrollerisse, kusjuures iga bait kirjutatakse erinevale aadressile. SLAVE-kontrollerisse laadida olemasolev vastuvõtuprogramm. Teostada andmevahetus.

(4) Koostada algoritm ja programm andmete lugemiseks SLAVE-kontrollerist, kusjuures iga bait loetakse erinevalt aadressilt. SLAVE-kontrollerisse laadida olemasolev saatmisprogramm. Teostada andmevahetus.
(5) Koostada programm andmevahetuseks SLAVE-ga, hõlmates nii andmete saatmist SLAVE-sse kui ka sealt lugemist. SLAVE-sse laadida olemasolev programm. Teostada andmevahetus.

(6) Koostada programm SLAVE-i jaoks andmete saatmiseks ja vastuvõtuks, sõltumata sellest kas kasutatakse ühte baasaadressi või kaasneb iga baidiga erinev aadress. Seejuures tuleb arvestada, et vastavad reziimid võivad vahelduda.

(7) Ülesannete (1)...(5) korral arvutada välja andmevahetuskiirus. Esitada ajadiagrammid siinil toimuva kohta. Ülesande täitmiseks kasutada loogikaanalüsaatorit.

5. KONTROLLERI JA ÜLESANNETE VÕIMALUSED MIKROPROTSESSORSÜSTEEMI JA ASSEMBLERKEELE ÕPETAMISEL

Käesolevas töös praktikumiks esitatud ülesanded on erineva raskusastmega. Kui alapunktis on mitu ülesannet, siis on tavaliselt iga järgmine ülesanne eelmisest raskem. On ka alapunkte, mis koosnevad vaid kergetest või rasketest ülesannetest.
Ülesannete diferentseerimise tingib asjaolu, et praktikumi sooritajate tase ja algteadmised on erinevad. Juhendaja vajab ülesandeid nii neile üliõpilastele, kelle tase on nõrgem ja kes alles "Mikroprotsessorite" kursusel puutuvad kokku assemblerkeel ja mikroprotsessorsüsteemiga, kui ka neile, kellele kergemad ülesanded võivad tunduda ebahuvitavate ja võimetele allajäävatena. Igal juhul peavad juhendaja antud ülesanded tagama kõigile üliõpilastele ülevaate mikroprotsessorsüsteemist ja assemblerkeelest.

Käesoleva töö põhiosas antakse kontrolleri kirjeldus ja suhtlemine süsteemi komponentidega. Vaadeldakse protsessorit, asükroonset jadaväratit, rööpväratit, DAC-i ja ADC-d. Kirjeldustes tuuakse ära vajalikud aadressid ja andmed nende seadmete initsialiseerimiseks ning nendega suhtlemiseks. Mälude kirjelduses antakse mäluaadressivälja jaotus ROM-i ja RAM-i vahel ning muutmälu jaotus kasutaja- ja monitorprogrammi vahel. Eriti oluline on jälgida muutmälu jaotusest kinnipidamist, et kontroller saaks tõrgeteta töötada.
Protsessor Z80180 on heaks aluseks mikroprotsessorite tundmaõppimisel, et edaspidi oleks lihtsam aru saada 16-, 32- jne. bitiste, RISC- ja keeruliste signaalitöötlusprotsessorite ehitusest ning tööpõhimõttest. Tema eeliseks võrreldes nt. Intel 8080-ga on see, et mitmed sõlmed nagu ASCI, taimer, CSIO paiknevad kõik ühes kiibis. Algkäivitusel seatakse nad automaatselt kindlasse olekusse ega hakka segama kasutaja tegevust. Kasutajal pruugib vaid need sõlmed vastavalt vajadusele ümber programmeerida.
Esimeseks süsteemi koostisosi hõlmavaks ülesandeks on koostisosade testimine ja nende töökõlbulikkuses veendumine. Need ülesanded on valitud seepärast, et reaalselt kontrollivad arvutid algkäivitusel vähemalt muutmälu korrasolekut. Tihti tuleb personaalarvuti kasutajal kontrollida vastavate programmide abil mõne seadme korrasolekut ja parameetrid. Testimisel on eriti olulised muunduritega seotud ülesanded.
Analoog-digitaalmuunduri testimine on oluline seetõttu, et füüsikalises eksperimendis muudetakse üha enam andurilt saadav analoogsignaal diskreetseks signaaliks. Diskreetimisel piiratakse paratamatult infot. Seepärast tuleb õppida hindama, kui suur on muundamise täpsus, kas muundur on korras ja ei tekita lisavigu ning õppida neid kõrvaldama. Muunduri testimiseks koostatud ülesanne punktis 4.3.4. nõuabki sisendpinge ja andmesõna vahelise seose ning muundamise lineaarsuse kontrollimist. Tänu kümnele bitile võimaldab ADC +5-voldise toitepinge korral eristada 4.88-millivoldist sisendpinge muutust. See on praktikumi jaoks piisav.
Digitaal-analoogmuundur võimaldab diskreetseid andmeid uuesti või esmakordselt analoogkujule viia. Andmeteks võivad olla ADC-ga diskreeditud analoogsignaalid, mida töödeldakse mikroprotsessoris digitaalsel kujul ja viiakse uuesti analoogkujule. Andmed võivad olla ka programmiliselt genereeritud, nt. siinus- ja nelinurksignaalid, mida saab kasutada sagedusanalüüsiks, välisseadmete juhtimiseks jne. Signaali täpsus sõltub peale programmi ka muunduri omadustest. Viimaste kontrolliks ongi esitatud vastav testimisülesanne punktis 4.3.5.
Süsteemi koostisosade kirjeldused ja initsialiseerimised on esitatud selliselt, et kasutaja saaks ühest kohast kätte minimaalse süsteemi koostisosadega suhtlemiseks vajaliku info.

Kontrolleri juhtimiseks vajaliku tarkvara moodustavad kontrolleri püsimälus olev monitorprogramm ja personaalarvuti mälus olev silumiskeskkonna käivitav programm Z180. Silumiskeskkonna kirjeldus koos monitorprogrammi käskudega on punktis 3.2. Punktis 3.3 antakse järgmised põhireeglid assemblerprogrammi koostamiseks:
1) kasutatavad arvud ja nende esitamine;
2) käsu operandi struktuur;
3) mälu reserveerimine assemblerkeeles kirjutatud programmis;
4) märgendite kasutamine ja konstantide deklareerimine;
5) failide linkimine assemblerprogrammiga;
6) programmi käsurea struktuur;
7) programmi lõpetamine.

Kasutaja puutub assemblerkeelega kokku kohe, kui ta asub käesolevas töös toodud ülesandeid täitma. Töö kergendamiseks antakse punktis 3.3.3. juhiseid programmi kirjutamiseks, transleerimiseks, kontrollerisse laadimiseks ja silumiskeskkonda sisenemiseks.
Monitorprogrammi käskude harjutamiseks on ülesanne (1) punktis 4.1.1. Selles antakse etapid, mida kasutaja peab silumiskeskkonnaga tutvumisel läbima, et kontrolleri programmvarustuse võimalusi täiel määral ära kasutada.
Punktis 4.1.2. antakse monitorprogrammi alamprogrammide kirjeldused, mida tudeng saab oma programmides kasutada. Nende alamprogrammidega tutvumiseks on harjutused punktis 4.1.2. Ülesanded algavad kergematega ja muutuvad järjest keerulisemaks.

Kontrolleri signaalide testimiseks on punktis 3.1. antud ülevaade loogikaanalüsaatorist ja tema käsitsemisest koos mõningate nõuannetega. Seejuures on oluline andmelugemise taktsageduse ja trigersõna seadmine. Kui analüsaatori andmelugemise taktsagedus on väiksem protsessori taktsagedusest, ei pruugi ajadiagrammid olla vastavuses reaalsete signaalidega, sest viimased muutuvad analüsaatori jaoks liiga kiiresti. Analüsaatori taktsagedus peaks olema vähemalt kahekordne protsessori taktsagedus. Trigersõna valimine on oluline seetõttu, et protsessoril on palju testitavaid siganaale. Kasutaja peab õppima leidma vajaliku kombinatsiooni, et analüsaator käivituks vajalikul hetkel.
Analüsaatori kasutamise näitlikustamiseks on lisas 2 esitatud ühe programmi ajadiagrammid. Punktis 4.2.1. on antud ajadiagrammi ühe lõigu kirjeldus. Selle järgi on koostatud programm, lisatud kommentaarid ja koostatud algoritm. Selle kõige harjutamiseks on ülesanded (2)...(5) samas punktis. Näiteprogrammis on valitud sellised käsud, et oleks esindatud võimalikult rohkem käsutüüpe. Ajadiagrammilt koostatakse käsutüüpide ajadiagrammid ja kontrollitakse nende vastavust protsessori kirjelduses [4] toodule. Selle harjutamiseks on ülesanne (1).
Punktis 4.2.1. on veel nõuded ja soovitused programmide silumiseks. Need on praktikas järele proovitud ja kehtivad reeglina ka kõrgkeeltes programmeerimisel.

Programmid MLT816, DIV8, BINTOBCD, BCDTOBIN, SENDDATA ja SENDPIO on esitatud eesmärgiga õpetada seost programmi ja algoritmi vahel. Algoritmid on õiged, ent nimetatud programmid on sihilikult muudetud algoritmile mittevastavateks. Muudetud on teostatavaid operatsioone, käske ära jäetud ja muudetud. Tudengi ülesandeks on algoritmi järgi programm siluda ja käima saada.

Aritmeetikatehetest vaadeldakse korrutamist ja jagamist, sest protsessoril vastavad käsud puuduvad, v.a. **`MLT ww`**. Esmalt aga antakse ülevaade arvude esitamisest kontrolleris, et oleks selge, millist arvkoodi kasutada. Negatiivsete ja positiivsete arvude teisendamise harjutamiseks on ülesanne (1) punktis 4.4.1.
Korrutamiseks on valitud algoritm vahetulemuse nihutamisega. Põhimõtteliselt pole programmi pikkuse ja täitmisaja suhtes olulist erinevust paremale nihutamisega võrreldes. Eeliseks on aga see, et vasemale nihutamisel formeeritakse esmalt korrutise vanemad bitid, paremale nihutamisel aga korrutise nooremad bitid. Kui signaalitöötluses on vaja korrutada murdarve, mille reaalosas on vähe kohti, murdosas aga palju ja korrutist tuleb murdosa nooremate bittide osas kärpida, siis võib lihtsalt tegurite nooremad bittid kõrvale jätta.
Korrutamise harjutamiseks tuleb siluda programm MLT816. See ülesanne on madalama raskusastmega. Kõrgema raskusastmega on harjutus (2), kus tuleb teha tehteid mitmebaidiste arvudega. See on heaks praktikaks pinumälu ja mälu kasutamisel. Ülesande (3) eesmärk on negatiivsete arvudega tehtavate tehete ja XOR-tehte harjutamine.
Jagamisülesanne (1) on kergem ja tema eesmärgiks on programmide silumisoskuse arendamine. Raskem ülesanne (2) nõuab algoritmi koostamist ja pinumälu ning lippude kasutamist teatud tingimuste salvestamiseks programmi mingi lõigu täitmise ajaks.
Murdarvude jagamine on esitatud aritmeetikatehetest keerulisim, sestap on mõlemad ülesanded kõrgema raskusastmega.

BCD- ja binaarkoodi teineteiseks teisendamise ülesanded (1) ja (3) on madalama raskusastmega, sest need nõuavad vaid programmide silumist ja algoritmi koostamist. (2),(4) ja (5) on kõrgema raskusastmega, eriti (5), kus tuleb kasutada lisaks ASCII-koodi. Samas on see praktiline ülesanne BCD-arvude inditseerimiseks kuvaril.
Gray koodi teisendamisülesanded on madalama raskusastmega, taandudes XOR-tehte algoritmi koostamisele ja kasutamisele.

Kerged on ka mälu indekseerimise ülesanded, sest kasutajale on esitatud vastavad algoritmid. Samas on mälu indekseerimisel suur praktiline tähendus eelkõige andmetöötluses aga ka mikroprotsessorsüsteemis üldse. Tihti tuleb sisendandmed paigutada mällu teatud struktuuri järgi, nt. tabelitesse, mis on sageli mitmeleheküljelised. Selliseks rakenduseks võiks olla erinevates spektraalpiirkondades ühest ja samast objektist salvestatud kujutiste digitaliseerimine ja salvestamine mällu. Üks mälulehekülg vastab ühele kindlale spektraalpiirkonnale, rea- ja veeruindeksid määravad piksli asukoha kujutises ja elemendi väärtus piksli väärtuse vastavas spektraalpiirkonnas.
Kontrolleri muutmälu puuduseks on see, et seda ei saa füüsiliselt lehekülgedeks jagada, sestap tuleb seda teha programmiliselt.

Kontrolleri eripäraks on see, et koostamisel pole PIO kviteerimis- ja katkestussignaale ei protsessori ega sisend-väljundkuplungitega ühendatud. Portide viikudel olevad puhvrid on lisaks ühesuunalised. Värati juhtimissignaalide mitteühendatus on mõneti kasulik, sest see annab põhjuse kviteerimismeetodi programmiliseks korraldamiseks. Kui kõiki PIO viike oleksid kasutatud, muudaks see värati initsialiseerimise ja temaga suhtlemise aeglasemaks ja keerulisemaks.
Punktis 4.7.1. antakse ülevaade Z80PIO kviteerimisreziimist, millesse tuleb kontrolleri ehituse tõttu teha korrektiive. Ülesannete eesmärgiks on koostada reaalselt toimuva andmevahetuse ajadiagrammid. Lihtsustamiseks on antud andmevahetuse algoritmid ja mõned programmid, ent viimased vajavad silumist, et neid reaalseks andmevahetuseks kasutada.

Asünkroonne jadavärat võimaldab tänu MAX-liidesele suhtlemist arvutiga ja andmevahetust pika liiniga mõnekümne meetri kaugusele. Punktis 4.7.2. antakse juhiseid ja nõuandeid ASCI0-i initsialiseerimiseks ja temaga suhtlemiseks. Ülesanded on kasvava raskusastmega. Eriti keeruline on ülesanne (5), sest see hõlmab ka monitorprogrammi alamprogramme andmete sisestamiseks klaviatuurilt ja inditseerimiseks kuvaril. Samas õpetab see ülesanne, kuidas lihtsate vahenditega teostada reaalajas toimuvat andmevahetust, mis on sarnane talk-programmile, kuid ei vaja Windowsi ega Unix-it. Sellist andmevahetust saab teostada ka arvuti COM-pordi kaudu, kui kirjutada programm kõrgkeeles ja kasutada selles COM-pordi otseseks juhtimiseks assmblerkeelt.

Kontrolleri puhvrite ühesuunalisus takistab I2C-protokolli täielikku rakendamist, sest SDA-liin peaks olema kahesuunaline. Reaalselt saab aga ühe liini kaudu andmeid saata, teise kaudu aga vastu võtta. I2C tähtsus seisneb selles, et seda protokolli kasutavad paljud seadmed, eriti mälud. Igaüks võib ise takti- ja andmesignaali abil kombineerida oma rakendustele sobiva protokolli. Eriti mugav on asjaolu, et I2C-d saab teostada ka arvuti COM- ja LPT-portide kaudu. Tänu sellele saab I2C-d kasutavad seadmed ühendada nimetatud portidega ja koostada programm kõrgkeeles, lisades portidega suhtlemiseks assemblerkeelsed lõigud.

6. KOKKUVÕTE

Käesolevas töös on esitatud materjal mikroprotsessorite praktikumi läbiviimiseks: süsteemi üksikosade kirjeldused, tarkvara kirjeldus ja ülesanded mikroprotsessorsüsteemi tundmaõppimiseks. Esitatud materjal on edukalt kasutatav praktikumi juhendina. Samas on ta pisut laiem, võimaldades konkretiseerida loengus saadud üldinfot mikroprotsessorsüsteemi kohta. Praktikumis saab järele kontrollida esitatud materjali vastavust tegelikkusele. Harjutused on koostatud selliselt, et õppejõul oleks ülesannete määramisel valikuvõimalus, arvestades tudengite teadmiste taset, loengukursuse mahtu 64 tundi ja praktikumi mahtu 32 tundi. Praktikumis realiseeritavate ülesannete valikul on arvestatud, et mikroprotsessorite praktikum peab samadel ülesannetele toetudes käesoleva aasta sügissemestril käima minema. Omad piirangud seadis ka bakalaureusetööle kehtestatud maht ja ettenähtud aeg.

Käesoleva töö baasil saab praktikumi edasi arendada, lisades harjutused katkestuste korraldamiseks ja muudeks ülesanneteks, s.h. signaalitöötlus. Selleks tuleb kaasata signaalitöötlusprotsessor ADSP2101, mis koos kontrolleriga LKZ180 peab andma tudengile ülevaate praktilisest signaalitöötlusest nii programmilisest kui ka raudvaralisest küljest. ADSP2101 annab võimaluse Fourier' pöörde teostamiseks. Saadud andmed inditseeritakse graafiliselt personaalarvutil, kasutades selleks Antti Raudsepa kirjutatud programmi. Selliseid vahendeid kasutades on võimalik saada kätte erinevate signaalide spektrid, hinnata raudvaralise ja programmilise filtreerimise kvaliteeti jne. Peale selle on praktikumi edasiarenduseks vajalik valmistada kontrollerile lisalülitusi nagu BCD-koodi inditseerimiseks indikatsioonilülitus, Gray koodi näitlikustamiseks analoog-digitaalmuundur, klaviatuur katkestussisendite kasutamiseks, lülitused signaalide filtreerimiseks, genereerimiseks, muundamiseks ja testimiseks loogikaanalüsaatoriga.

Lõpetuseks avaldab töö autor oma sügavaimat tänu juhendajatele dotsent Ando Otsale kõigi nende teadmiste eest, mida töö autor temalt on saanud ja loodetavasti saab ka edaspidi ning nõuannete eest käesoleva töö valmimisel, ja Toivo Vajakasele eriti programmeerimisalaste ja kontrollerisse puutuvate nõuannete eest. Autor tänab väga nõuannete eest Enn Ütsit.

7. KASUTATUD ALLIKAD

1. Matti Fischer, Toivo Vajakas, *Mikroprotsessorite praktikum I.KIT 8080*, Tartu, 1989.

2. Analog Devices*, ADSP-2100 Family DSP Microcomputers. ADSP-21xx*, USA, 1994.

3. Tuomo Kouki, *Mikroprosessorit kesäopetus*, Joensuu, 1987.

4. Risto Punkkinen, *Mikroprosessoritekniikka I*, Turku, 1993.

5. Tom Kuusela, *Mikroprosessoritekniikka II*, Turku, 1990.

6. Zilog, *Preliminary Product Specification. Z80180. Enchanced Z180 Microprocessor*, http://www.zilog.com/z180/.

7. Adam Osborne, Gerry Kane, *Osborne 4&8-Bit Microprocessor Handbook*, Berkeley, California, OSBORNE/McGraw-Hill, lk. 7-45...7-54, 1981.

8.Analog Devices, *1992 Data Converter Reference Manual Volume II*, USA, lk. 2-347...2-361, 1992.

9. Analog Devices, *1992 Data Converter Reference Manual Volume I*, USA, lk. 2-43...2-46, 1992.

10. Intel, *Memory Components Handbook*, USA, Intel Literature Distribution, lk.4-71...4-80, 1985.

11. *3332 Logic Analyser BLACK STAR Instruction Manual*, U.K., Black Star Ltd., 1996.

12. Toivo Vajakas, *Monitorprogrammi kasutusjuhend*, Tartu, 1996.

13. Ando Ots, *Digitaalelektroonika ja mikroprotsessortehnika. Seminaritööd 1*, Tartu, 1995.

14. Lance A.Leventhal, Winthrop Saville, *8080/8085 Assembley Language Subroutines*, Berkeley, California, OSBORNE/McGraw-Hill, 1983, tõlge vene keelde: Moskva, Radio i svjas, 1987.

15. A.L.Gurtovtsev, S.V.Gudõmenko, *Programmõ dlja mikroprotsessorov*, Minsk, Võšaja Škola, 1989.

16. Vincent Himpe, *I2C Bus FAQ Version 1.6*, http://www.ping.be/~ping0751/i2cfaq.zip.

8. SUMMARY

The title of the present Bsc. thesis is “Getting Acquainted with the Microprocessor System and Assembly Language with the Controller LKZ180.”

The purpose of the thesis was to create a database and compile assignments for teaching the microprocessor system with help of the controller LKZ180. LKZ180 is based on Zilog’s Z80180 and has been designed so that the user can communicate with LKZ180 via a personal computer and test the signals of the controller with the help of the logic analyser Black Star 3332. All components of the system are on a single plate. The processors clock rate is 6.144 MHz which is low enough to test signals properly with the logic analyser. At the same time this clock rate allows to achieve the maximum baud rate on the Asynchronous Serial Communication Interface (ASCI).

The opcodes, the operands and the signals of the processor’s address bus, data bus and control bus can be read from the timing diagram of the logic analyser. The user has to compile a program, to comment on the program and to elaborate the program’s algorithm based on the timing diagram. The logic analyser can be used to test the other signals of the system. It is especially important to test communication signals generated on the basis of the program and to debug programs.

The description of the controller, the instructions and exercises for testing signals, for elaborating an assembly program, for performing arithmetical and code operations as well as data communication are presented. The exercises and the aid material have been presented so that the user can obtain at least minimum necessary information from a single source. The presented assembly programs are deliberately changed with the aim to teach the debugging of programs and the using of algorithms. Some exercises require working out of an algorithm or a program based on timing diagrams.

A total of 52 exercises are presented which number exceeds the scope of practical training 32 academic hours. The exercises are divided into easier and more difficult ones. In this way the exercises can be assigned according to the user’s level, and the supervisor can choose suitable exercises for different students.