TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutisüsteemide instituut

Lembitu Valdmets 135208

8-BITISE MIKROPROTSESSORI MUDEL

Bakalaureusetöö

Juhendaja: Peeter Ellervee
professor
Kaasjuhendaja: Arvo Toomsalu
lektor

Tallinn 2016

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Lembitu Valdmets

22.05.2017

Annotatsioon

Käesolev lõputöö on valminud autori enda soovist koostada protsessori mikroarhitektuur, tuginedes bakalaureuseõppes omandatud teadmisetele, lõputöö juhendaja abile ning aine „IXX1430 Arvutisüsteemid – projekt“ käigus tekkinud meeskonnale.

Autor tahab eraldi välja tuua meeskonna liikme Martin Permani abi projekti valmimisel.

Lõputöö põhieesmärgiks oli koostada 8-bitise mikroprotsessori mudel, mida oleks võimalik kasutada õppeotstarbel, selgitamaks protsessori ehitust ja talitlust. Tulemusena valmis programmis Logisim paindlik protsessori mudel, mida on võimalik kasutada alates protsessori töö demonstreerimises kui ka masinkoodis programmeerimises.

Lõputöö on koostatud eesti keeles ja sisaldab teksti 28 leheküljel, 4 peatükki, 22 joonist, 3 tabelit.

Abstract

8-bit microprocessor model

The current thesis is about building a custom microprocessor architecture from scratch to be used for educational purposes. This model could be used in „Arvutid 1“ to improve students’ understanding of computer architectures, maschine- level programming and digital system construction.

In chapter one the main topic is introduction to thesis, furthermore there is some general information about the topic.

The second chapter is about analyzing current solutions which are being used to teach students computer architecture. Also there is a brief comparizon between diferent logic simulators.

The third chapter is mainly about designing an architecture for processor and there is also detailed description about operation automata and its diferent functions. Chapter ends with main schematic of processor.

The fourth chapter is all about instruction set. The inscruction set is dividend into 5 different types of commands: loading, saving, arithmetic and logic, register-register copying and jump commands. Each subchapter explains how these types of commands work, including all the commands with descrptions and detailed clock cycles.

The last meaningful chapter is a brief explanation about using and programming current microprocessor.

There are also intresting extras, which include 2 sample programs, simplified processor user manual and control unit’s automata table.

The thesis is in Estonian and contains 28 pages of text, 4 chapters, 22 figures, 3 tables.

Lühendite ja mõistete sõnastik

ASI	TTÜ Arvutisüsteemide instituut
MAR	*Memory address register*, mälu aadressiregister
MDR	*Memory data register*, mälu andmeregister
PC	*Program counter*, käsuloendur
IR	*Instruction register*, käsuregister
ALU	*Arithmetic and logic unit*, aritmeetika-loogikaplokk
TRD	*Targer register decoder*, tulemregistri dekooder
VHDL	*Virtual hardware description language*, virtuaalne riistvara kirjelduskeel
FR	*Flag register*, lippude register
Dc	*Decoder*, dekooder
CLKSEL	*Clock selector*, taktivalija
MCLK	*Manual clock*, manuaalne taktsignaal
ACLK	*Automatic clock*, automaatne taktsignaal
RES	*Reset*, lähtestamine
CU	*Control unit*, juhtseade

Sisukord

1 Sissejuhatus 8
2 Võimalike lahenduskäikude analüüs 9
2.1 Mikroprotsessori mudeli kasutamise olulisus 9
2.2 Võimalikud lahendused 10
2.2.1 Kasutusel olevad vahendid õppetöö läbiviimiseks 10
2.3 Loogikasimulaatori valik 10
2.4 Mikroprotsessorile esitatavad nõuded 12
3 Protsessori koostamine ja kirjeldus 13
3.1 Protsessori juhtautomaat 14
3.1.1 Protsessori juhtautomaadi algne koostamine 14
3.1.2 Protsessori juhtautomaadi riistvaraline koostamine 15
3.1.3 Protsessori juhtautomaadi mikroprogramne koostamine. 17
3.1.4 Protsessori juhtautomaadi mikrokäsud 18
3.2 Protsessori operatsioonosa 20
4 Protsessori käsustik 29
4.1 Käsustiku kirjeldus: 29
4.2 Käskude kirjeldused käsutüüpide kaudu 30
4.2.1 Laadekäsud 31
4.2.2 Salvestuskäsud 32
4.2.3 Aritmeetika- loogikakäsud 33
4.2.4 Registrite-vahelised kopeerimiskäsud 35
4.2.5 Siirdekäsud 36
5 Protsessori kasutamine ja programmeerimine 37
6 Kokkuvõte 38
Kasutatud kirjandus 39
Lisa 1 – Näiteprogrammid 40
Lisa 2 – Logisimi ja protsessorimudeli kasutamise kiirjuhend 43
Lisa 3- - Protsessori juhtautomaadi tabel. 44

Jooniste loetelu

Joonis 1. Mikroprotsessori juhtautomaadi riistvaralise realisatsiooni ekraanitõmmis. .. 16
Joonis 2. Ekraanitõmmis mikroprogramsest juhtautomaadist. 17
Joonis 3. Taktivalija sisestruktuuri loogikaskeem. 21
Joonis 4. Ekraanitõmmis käsuloendurist põhiskeemis. 22
Joonis 5. Ekraanitõmmis mälu aadressiregistrist põhiskeemis. 22
Joonis 6. Joonis mälu andmeregistrist põhiskeemis. 22
Joonis 7. ALU sisestruktuuri ekraanitõmmis. 24
Joonis 8: Ekraanitõmmis käsuregistrist põhiskeemis 24
Joonis 9. Tulemregistri dekoodri sisestruktuuri ekraanitõnmis. 25
Joonis 10. Lippude registri sisestruktuuri ekraanitõmmis, ... 26
Joonis 11. Ekraanitõmmis registrist A põhiskeemis. 26
Joonis 12. Ekraanitõmmis registrist B põhiskeemis. 27
Joonis 13. Ekraanitõmmis registrist C põhiskeemis. 27
Joonis 14. Ekraanitõmmis registrist M põhiskeemis. 27
Joonis 15. Joonis protsessori põhiskeemist Logisimis. 28
Joonis 16. Erinevate käsipikkuste kujutamine. 29
Joonis 17. Tühikäsu taktideks jagamine. 30
Joonis 18. Laadekäskude taktideks jagamine. 31
Joonis 19. Salvestuskäskude taktideks jagamine. 32
Joonis 20. Aritmeetija- ja loogikakäskude taktideks jagamine. 33
Joonis 21. Registritevaheliste kopeerimiskäskude taktideks jagamine. 35
Joonis 22. Siirdekäskude taktideks jagamine. 36

1 Sissejuhatus

Käesolev diplomitöö on valminud aine „Arvutisüsteemid – projekt“ raames, mahult ületas see projekt ka ainet ning diplomitöö mahtu saaks võrrelda lõputöö ja eeltoodud aine kogumahuga.

Põhiülesandeks oli vaja konstrueerida 8-bitise mikroprotsessori arhitektuuri ja komponente, kasutades loogikasimulaatorit Logisim. Mudel peab olema kasutatav õppetöös, demonstreerimaks protsessori struktuuri ja tööd. Mälumahu, sisend- väljundi ja katkestuste realiseerimisel tuleb lähtuda lihtsusest ja arusaadavusest.

Diplomitööna valminud protsessorit on võimalik efektiivselt ära kasutada õppetöös, kus on lihtsam õpetada protsessori struktuuri ja erinevat talitlust: käsuvõttu, käsu dekodeerimist, käsu täitmist ja andmete salvestamist.

Juba projekti alguses oli paigas, et kasutatavaks keskkonnaks protsessori valmistamiseks saab olema Logisim, mis on vabavaraline ja milles on mugav ning efektiivne sellise keerukuse astmega digitaalskeemi koostada.

Praegusteks lahendusteks protsessori töö demonstreerimiseks on kas koostatud keeles VHDL või omavad abstraktset registersiirete tasandil mudelit. Käesolev mudel aga suudab protsessori tööd simuleerida nii registersiirete tasemel, kui ka loogikatasandil, pakkudes lihtsamat arusaadavust protsessorist kui loogikaskeemist.

Peamisteks teemadeks on olemasolevate lahenduste analüüs, seejärel käsitletakse aruandes protsessori operatsioonosa ja juhtosa disainimist ning konstrueerimist. Hiljem on protsessori käsustiku kirjeldus, kus käsud on jaotatud vastavalt funktsionaalsusele gruppidesse. Aruande lisadesse on välja toodud 2 näiteprogrammi, lühike õpetus protsessori kasutamisest ja juhtautomaadi tabel.

2 Võimalike lahenduskäikude analüüs

2.1 Mikroprotsessori mudeli kasutamise olulisus

Vaadates aine „Arvutid 1“ õppeaine eesmärke, kus on kirjas: „Anda ülevaade protsessori funktsioneerimise põhimõtetest“ ja „Selgitada, kuidas on organiseeritud andmevahetus mikroarvutis erinevate komponentide vahel.“ [1], on peetud oluliseks teha üliõpilastele arusaadavaks protsessori tööpõhimõtte printsiibid. Samas ei ole otseselt kirjas, kuidas selgitatakse neid teemasid kuulajaskonnale. Sestap olemasolevaid materjale kasutades on küll võimalik selgitada digitaalloogikat ning skeemide koostamist, kuid jääb vajaka protsessori struktuuri seletamine töötava mudeli näol.

Õpikus on küll rohkelt materjale, kuid praktilisem kogemus õliõpilastel protsessori tööpõhimõtetega on siiski üsna väike võrreldes teiste digitaalloogika süsteemidega.

Olemasolevad lahendused õpiväljundi koostamisel on aegunud ning abstraktsed. Arvutid 1’s antavad praktikumid teevad küll selgeks lihtsa digitaalskeemide kasutamise, kuid ei anna üliõpilastele üldist pilti protsessorist, niisamuti ei leidu adekvaatset hierarhilist protsessorimudelit, mille kaudu oleks võimalik jälgida seadme tööd mitmel tasandil.

2.2 Võimalikud lahendused

2.2.1 Kasutusel olevad vahendid õppetöö läbiviimiseks

Praegu on 2 praktikumi juhendajat ja mõlemal on erinevad vahendid õppetöö läbiviimiseks. Ülesanded, mida antakse üliõpilastele lahendada, on siiski väga sarnased.

Hetkel on aine „Arvutid 1“ praktikumide läbiviimisel kasutusel erinevad loogikasimulaatorid, praktikume viivad läbi 2 õppejõudu: Margin Aarna ja Elmet Orasson. Margit Aarna kasutab oma statsionaarseid materjale minimaalselt ning eelistab kasutada kohapeal õpetamist. Lisaks on praktikumides kasutatava tarkvara valimine üliõpilaste enese teha, soovitatav on kasutada skeemisimulaatorit „Falstad“, kuid võib kasutada ka muid skeemisimulaatoreid õppejõuga kokkuleppel. [2].Logisim on küll kasutuses ühe võimaliku loogikasimulaatorina, kuid selle massilist kasutamist ei esine.

Elmet Orasson kasutab oma aines skeemide koostamiseks Xilinx’it: professionaalset tarkvara, mis on mõeldud keerukate digitaalsüsteemide projekteerimiseks. Praktikumide raames tuletatakse meelde diskreetset matemaatikat ning tutvutakse elektriskeemi koostamise põhisammudega: [3, p. 2] Õppetöös kasutatav tarkvara on küll pea kõikide võimalustega erinevate digitaalskeemide koostamiseks, kuid üliõpilastele nii keeruka tarkvara õpetamine on üpris keerukas. [3]

On tehtud palju korralikke protsessorimudeleid, kasutades programmeerimiseks ning süsteemide koostamiseks VHDL’i, kuid 1. kursuse üliõpilasel ei ole veel olnud kursust VHDL’i kasutamise kohta. Seetõttu oleks esimesel kursusel viibivale liiga palju informatsiooni sellest rääkida, lisaks VHDL omab abstraktsioone, mis muudab arusaamise loogikast palju hägusemaks. Suurepärase näite digitaalloogikast kõrgemal tasemel tuua Õppeaines IAY0340 „Digitaalsüsteemide Modelleerimine ja diagnoos“, kus on modelleeritud mitmeid süsteeme just keeles VHDL [4]

2.3 Loogikasimulaatori valik

Kui oli selgunud protsessori koostamise vajalikus, oli vaja välja selgitada, missuguse loogikasimulaatoriga koostada antud mudel. Nimelt lihtsasti kasutatav ja intuitiivse kasutajakeskkonnaga loogikasimulaator on väga oluline, eriti üliõpilastele, selgitamaks protsessori mudeli põhimõtteid ilma, et üliõpilased peaksid liialt sügavuti tutvuma

keskkonna kasutamisega. Suurepäraseks lisaks vajamineva loogikasimulaatori juures on ka võimalus koostada hierarhilisi skeeme, kus ka skeemis olevad elemendid võivad olla kasutaja enese koostatud.

Järgmisena on välja toodud loetelu erinevatest loogikasimulaatoritest, millele leiab interneti otsingumootorite abiga ohtralt lisainformatsiooni, koos nende parimate ja halvimate omadustega antud projekti suhtes.

Üks esimesi loogikasimulaatoreid, mida internetis leida saab, on aadressil logic.ly, niisamuti simulaatori enda nimi oleks justkui Logic.ly. Antud simulaatori on veebipõhine ning kasutab väga lihtsat keskkonda skeemide valmistamiseks. Kahjuks ilma internetiühenduseta programmi kasutada ei saa ning samuti on puudu hierarhiliste skeemide koostamise võimalus. Lisaks on elementide vahel konstrueeritavad võtmed kujutatud loogeliste juhtmetena, mis sõlmpunktides võib meenutama hakata sõlmpuntraid. Programmi täisverioon on lisaks tasuline ning tasuta versioon omab piiratud võimalusi. [5]

Järgmine on Cedar logic simulator, mis erinevalt eelmisest simulaatorist on vabavaraline ning tasuta allalaetav ja kasutatav. Skeemi koostamine on samuti intuitiivsem ning elementide vahelised ühendused ei meenuta sujuvate nurkadega juhtmeid. Kahjuks puudub võimalus koostada hierarhilisi skeeme ning värskeim versioon on aastast 2013. [6]

Kindlasti tuleb mainida programmi, nimega „Ksimus“, mida samuti saab kasutada loogikaskeemide koostamiseks, programm toetab ka hierarhilist skeemide koostamist. Kahjuks programmi kohta levib vähe informatsiooni ning kasutajatugi lõppes veelgi varem, kui Cedar Logic simulatoril. [7]

Parimaks lahenduseks projekti jaoks saab pidada loogikasimulaatorr „Logisim“. Tolle vabavaralise programmiga saab lihtsalt ja intuitiivselt koostada nii lihtsaid kui keerukaid loogikaskeeme. Lisaks toetab programm nii Windowsi kui ka Linuxi-põhiseid operatsioonisüsteeme. Võimalik on koostada ka hierarhilisi skeeme. [8]

2.4 Mikroprotsessorile esitatavad nõuded

Olemasolevate vahendite uurimise tulemusel ja Logisimi programmi valimisel skeemisimulaatoriks oli protsessori konstrueerimine selleks ajaks peaaegu pealehakkamise tulemus. Puudu oli siiski üks olulisemaid aspekte: millega peab valmis protsessor hakkama saama ning missugune oleks seadme funktsionaalsus. Järgnevalt on välja toodud loodava seadme põhiline funktsionaalsus.

Kõige tähtsam on seadmel käskude täitmise suutlikkus: seade peab olema suuteline täitma käske talle etteantud järjekorras, nagu programmist oli ette arvestanud. Lisaks tavalistele käskudele, nagu liitmine a andmete kopeerimine põhimälust mõnesse registrisse peab protsessor olema suuteline teostama siirdekäske, sest nendega on võimalik koostada programmeerimisel olulisi siirdeid, tingimusi ja tsükleid.

Järgmisena peab protsessori juhtautomaat sisaldama enamik laialdaselt kasutuses olevaid aritmeetika- ja loogikaoperatsioone, nagu loogiline AND, OR, NOR, XOR, liitmine, lahutamine, nihked, suurendamine ja vähendamine 1 võrra, sellisel juhul on võimalik teostada seadmega sarnaseid operatsioone, nagu enamik laiatarbe protsessoritel.

Enamik protsessori põhiosad on koostatud loogikaelementide tasemel, lisaks on kasutatud hierarhilist lähenemist: sellisel juhul saab vaadelda protsessori tööd nii registersiirete, kui ka loogikaelementide tasemel.

3 Protsessori koostamine ja kirjeldus

Protsessori disainimine tundub esmapilgul üpris suurejooneline töö, sest neid seadmeid peetakse üheks kõige keerulisemateks masinateks, mida inimkond on eales valmistanud. Tegelikkuses on aga võimalik valmistada väga lihtsaid ja primitiivseid seadmeid, mille funktsionaalsus suures osas on võrreldav kallite ja keerukate infotöötlusseadmetega. Käesolev protsessor pidi algusest saadik olema pigem lihtne ja hõlpsasti arusaadav, mitte keerukas paljude võimalustega seade.

Protsessori valmistamiseks oli vaja läbida edukalt mitmeid aineid, kus saadud õpiväljundid on kokkuvõttes piisavad disainimise alustamiseks.

Peamisteks aineteks, koos koodidega, olid:

- Diskreetne matemaatika, IAY0010
- Arvutid1, IAF0041
- Digitaalsüsteemid, IAY0150
- Arvutite aritmeetika ja loogika, IAY0140
- Arvutiarhitektuurid IAY0520

Diskreetne matemaatika oli väga oluline aine, mis andis põhiteadmised Boole'i algebrast, digitaalloogika põhielementidest ja funktsioonide minimeerimisest. Arvutid 1 aines sai teada, kuidas kasutades põhielemente, koostada lihtsamaid digitaalskeeme, niisamuti anti kerge ülevaade arvuti riistvaras toimuvast. Digitaalskeemid oli aineks, mis suutis kirjeldada suurepäraselt lisaks kombinatoorskeemidele ka järjestikskeeme, sisaldas ka juhtautomaatide disainimist ning sünteesi. Arvutite aritmeetikas ja loogikas oli võimalik tutvuda erinevare operatsioonidega, mida on protsessor võimeline teostama, niisamuti tutvustas eksootilisi arvsüsteeme. Arvutiarhitektuurides oli võimalik tutvuda erinevate arvutite siseehituse, nende omavahelise suhtluse ja paljude arvutite tööpõhimõtetega.

Mida aeg edasi, seda hõlpsamaks tundus mikroprotsessori koostamine minevat: oleks tegemist justkui lihtsa seadmega, mis töötleb käske. Ainult käskude töötlemisest kõige primitiivsemas mõistes ei piisa siiski multifunktsionaalse protsessori koostamiseks, sest

funktsionaalsuse lisamisega kasvab ka seadme keerukus. Juba digitaalsüsteemide aines sai praktikumis koostada eelnevalt loodud aritmeetika- ja loogikaplokile juhtseadet, mis suutis primitiivses mõistes juhtida täiturseadet - see väga primitiivses mõistes on juba protsessor. Ka taolist seadet arendades on võimalik jõuda protsessorini, eraldades juhtautomaadi ning operatsioonautomaadi. Sellest ajast saadik oli selge, et protsessori disainimisel tuleb lähtuda juhtautomaadi ja operatsioonautomaadi disainimisest ning seejärel nende integreerimisest.

3.1 Protsessori juhtautomaat

Protsessori juhtautomaat on teatavasti üks keerukamaid plokke, sest see määrab protsessori võimekuse ja ka keerukuse seadme tundmaõppimisel. Kui ka paljud operatsioonosas asuvad elemendid on standardsed, on juhtautomaat seadmetel üsna erinevad. Järgnevalt on välja toodud, kuidas projekti koostamise käigus valmis käesolev mikroprotsessori juhtautomaat.

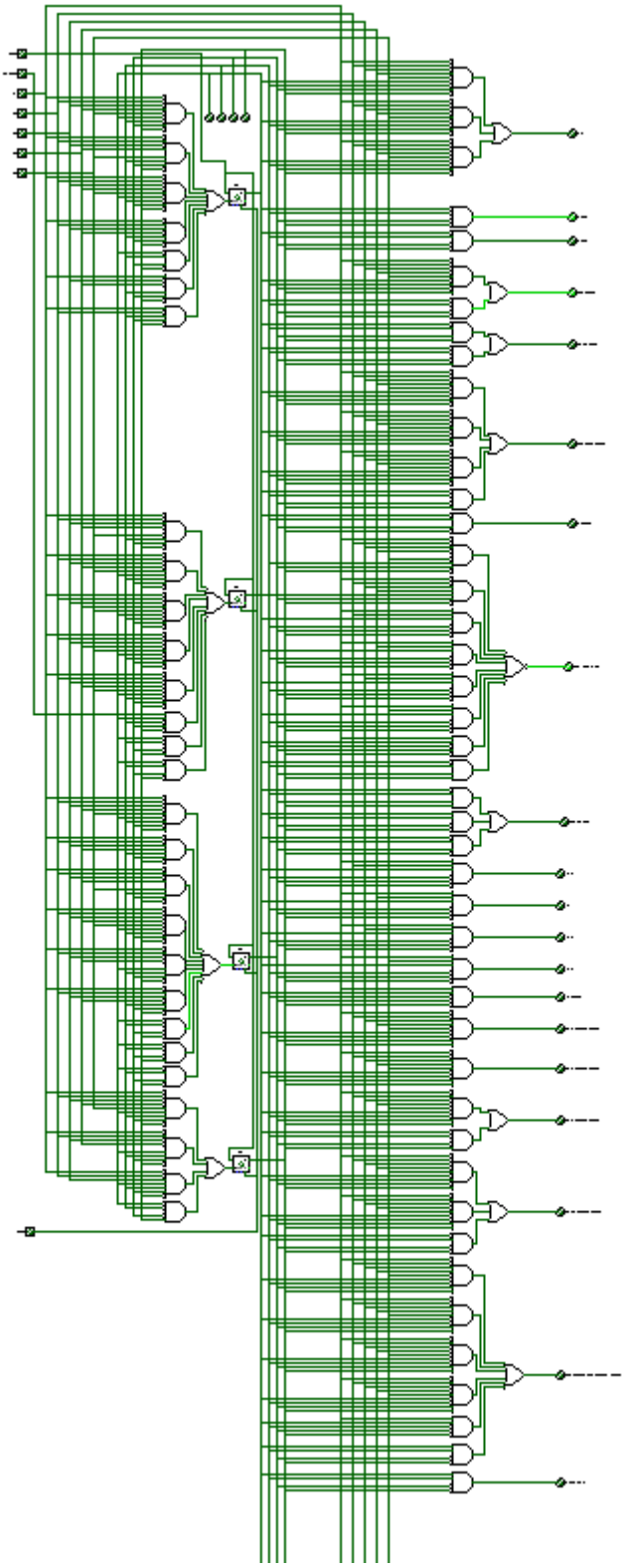
3.1.1 Protsessori juhtautomaadi algne koostamine

Ajendatuna põhjusest, et protsessor peab tulema võimalikult lihtne, tuleb ka juhtautomaat koostada võimalikult lihtsalt. Sellest ideest tekkis ka esmane plaan, kuidas konstrueerida protsessorile juhtautomaat. Järgides von Neumanni tsüklit, kus käsu täitmine toimub *fetch-decode-execute*-stiilis, on võimalik koostada rida D-trigereid, kus üks triger on korraga aktiivne ning tolle triger oleks ka käsu täitmise etapi tunnuseks. Sellisel juhul esimene triger suunaks aadressi püsimällu, teine triger kopeeriks andmed mälust protsessorisse, kolmas triger dekodeeriks väärtust ja nii edasi. Taolise disainimise juures tekkisid peamised probleemid erinevate käskude täitmisel, sest erinevad käsud vajavad eri arv takte täitmiseks. Isegi juhul, kui oleks kasutatud järjestikuseid trigereid, oleks käsu täitmise tsükkel veninud väga pikale ning käsu täitmiseks vajamine taktide arv oleks kasvanud drastiliselt.

3.1.2 Protsessori juhtautomaadi riistvaraline koostamine

Kui oli selge, et lihtsalt trigerite baasil juhtautomaati konstrueerida on väga ebatõenäoline, oli aeg pöörduda tagasi klassikaliste Moore ja Mealy automaatide juurde. Neid seadmed oli õpetatud aines „digitaalsüsteemid“ ning kuigi alguses tundus sellistes mõõtmetes juhtautomaadi genereerimine keerukas, sai kiiresti defineeritud sisendite, väljundite ja olekute arv. Sisendeid tuli kokku 10, neist 1 lippude sisend, 5 opkoodi sisendid ja 4 olekusisendit. Väljundeid, tuli aga oluliselt rohkem: 26, neid 4 esimest olid suunatud taaskord trigeritesse, mis muutsid väärtusi taktsignaali muutumisel, ja ülejäänud juhtisid protsessori operatsioonautomaati.

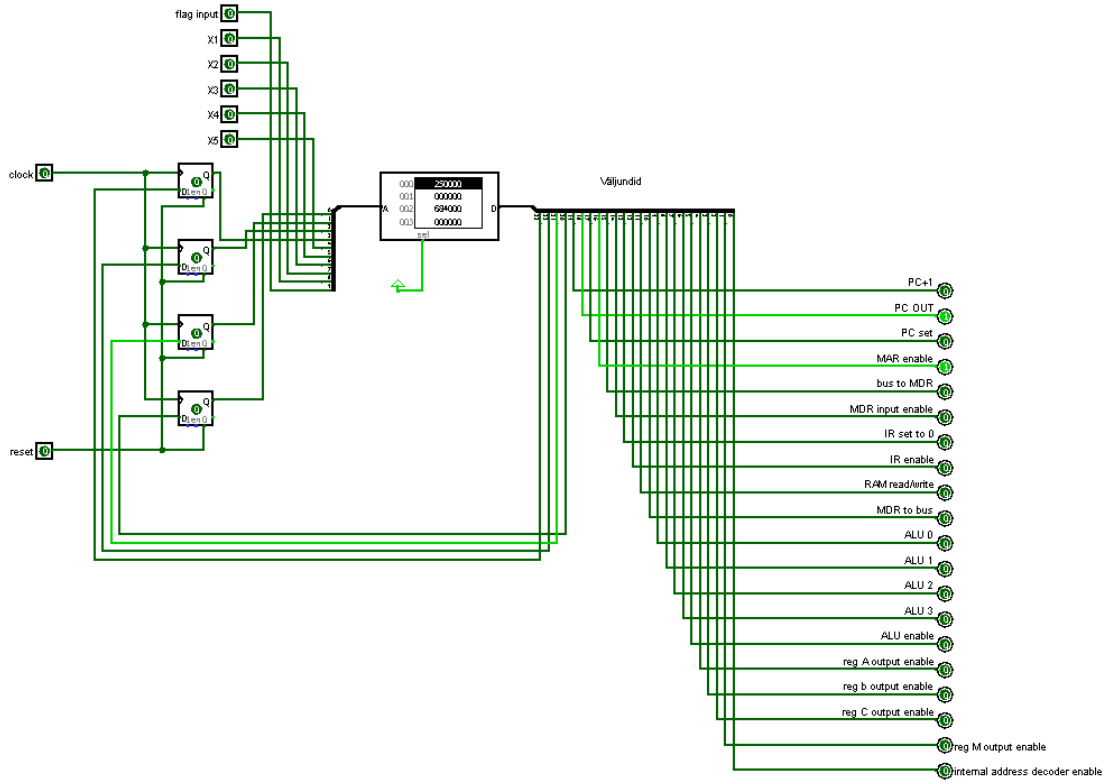
Järgenvalt sai koostada suure tabeli, kuhu sai kirja panna kõik olekud, sisendid ning väljundid. Tabelist aga loogikaskeemi oli oluliselt keerukam teha ning selle protsessi lihtsustamiseks kasutati minimeerimisprogrammi Espresso. Viimase väljundist kätte saadud lihtsustatud oli juba võimalik konstrueerida loogikaskeemiks, mis koosneb 10 sisendist, 4 trigerist ja 26 väljundist. Probleemiks oli aga tekkinud skeemi maht: ühe trigeri sisendi ette konstrueeriti lausa kaheksa 9-sisendiga AND-elemente, mis olid OR-elemendiga ühendatud trigeri sisendiga ja skeemi suurus kasvas drastiliselt.



Joonis 1. Mikroprotsessori juhtautomaadi riistvaralise realisatsiooni ekraanitõmmis.

3.1.3 Protsessori juhtautomaadi mikroprogramne koostamine.

Kui protsessori riistvaraline juhtautomaat oli valminud, oli skeemi testimine, vigade leidmine ning parandamine mahuka skeemi kujul väga keerukas ning ka esmaskursuslastele oleks liiga palju näidata nii suurt skeemi. Tekkis idee muuta suur riistvaraline skeem mikroprogramseks, mille peamiseks elemendiks on püsimälu, kus igale sisendite ja olekute kombinatsioonile vastab väljund. Kasutades metoodikat, et olekuid hoiavad 4 d-trigerit ja esimesed 4 väljundit määravad automaadi järgmise oleku, oli uut skeemi võrdlemisi lihtne koostada. [5, p. 156] Järgnevalt on välja toodud ekraanitõmmis lõplikust juhtautomaadi skeemist Logisimis.



Joonis 2. Ekraanitõmmis mikroprogramsest juhtautomaadist.

Võrreldes eelmise realisatsiooniga on viimane skeem oluliselt lihtsam ja arusaadavam, kuigi funktsionaalsus on jäänud samaks. Lisaks annab programmeeritava püsimälu kasutamine võimaluse muuta juhtautomaadi käitumist skeemi ennast muutmata. Mikroprogramse protsessori juhtautomaadi tabel on välja toodud lisas nr. 3.

3.1.4 Protsessori juhtautomaadi mikrokäsud

Protsessori juhtautomaadi disainimine põhines klassikalise automaadi koostamisele, sellega seoses tekkisid käskude täitmisel protsessoril eri olekud, mis mikroprogramse juhtimise korral muutusid mikrokäskudeks. Järgnevalt on välja toodud protsessori juhtautomaadi mikrokäskude nimed ja nende kirjeldused. Mikrokäskude enese nimed on autori välja mõeldud, illustreerimaks nende üldist tegevust töö ajal. Selgituseks tuleb lisada, et probe'i all peetakse silmas skeemil seadet, mis näitab põhisiinil asuvat väärtust.

"PC to MAR" - Mikrokäsu eesmärgiks on kopeerida käsuloenduri sisu mälu aadressiregistrisse. Selle jaoks avatakse käsuloenduri väljundventiil siinile ning mälu aadressiregistri lubamissignaal omistab väärtuse "1". Taktsignaali positiivse signaali korral on protsessori põhisiinil võimalik läbi probe'i näha käsuloenduri väärtust.

"RAM to MDR" - Mikrokäsu eesmärgiks on põhimälus olevate andmete kopeerimine mälu andmeregistrisse. Aadress, kust andmed võetakse, on eelnevalt määratud Mälu aadressiregistri määramisega mikrokäsu "PC to MAR" poolt. Paljude käskude täitmisel toimub just selle mikrokäsu ajal käsuloenduri suurendamine 1 võrra. Taktsignaali positiivse signaali korral on põhisiinil näha väärtust 0.

"MDR to IR" - Selle tegevuse juures kopeeritakse mälu aadressiregistrist andmed käsuregistrisse. Selle mikrokäsu lõpus on käsuregistrisse saadud käsk, mida hakatakse töötlema. Taktsignaali positiivse väärtuse korral on põhisiinil näha käsu väärtust.

"Decode IR" - mikrokäsk on vajalik, sest selle oleku juures juhtautomaat analüüsib sisendite ehk opkoodi järgi, millisesse järgnevasse olekusse juhtautomaat läheb. Ideaalis oleks saanud selle mikrokoodi vahele von Neumanni tsüklis jätta ning kohe peale "MDR to IR" suunduda käsu täitmise poole, kuid arusaadavuse paremaks tagamiseks ning traditsioonilise tsükli simuleerimiseks jäeti sisse. Põhisiinil mikrokäsu ajal on väärtus 0.

"MDR to reg X" - selle mikrokäsu juures kopeeritakse mälu andmeregistrist andmed registrisse, millele tulemregistri dekooder on lubava signaali väljastanud. Korraga on lubatud olekus vaid 1 register.

"ALU A & B" - Selle mikrooperatsiooni jooksul suhtleb juhtautomaat peamiselt aritmeetika- ja loogikaseadmega (ALU). ALU'le saadetakse opkoodi järgselt 4-bitine käsukood ning lubamissignaal. Tulemregistri dekooder määrab mikrokäsu ajal registri,

kuhu ALU poolt arvutatud tulemus saadetakse ning sinna registrisse saadetakse lubamissignaal. Põhisiini peal on näha ALU poolt arvutatud tulemust.

“JMP w flags” - Eesmärgiks on järgmiseks mikrokäsuks tuvastada, kas siidrekäsk läheb täitmisesse või lõpeb käsk ilma siiret sooritamata. Seda kontrollib juhtautomaat “flag input” - nimelise sisendi abil, oleku enda ajal juhtautomaadi väljundid on madala nivoo peal.. Antud mikrokäsk iseenesest ei ole hädavajalik ning siirdekäsku saaks korraldada ka ilma mikrokäsuta, sest käesolev mikroprogramne juhtautomaat on suuteline määrama järgnevat olekut nii sisendite kui ka konkreetse oleku põhjal. Parema arusaadavuse huvides on see mikrokäsk sisse jäetud riistvaralisest lahendusest, sest nii on hõlpsam aru saada protsessori töötsüklist. Taktsignaali ajal on käsuloenduril väärtus 0.

“M-reg to PC” - Antud mikrokäsu korral kopeeritakse M-registri sisu käsuloendurisse. Selle jaoks avatakse puhver M- registri väljundis ning lubav signaal käsuloenduri sisendis. Põhisiini peal on näha M-registris asuvat väärtust.

“X-reg to reg-Y” - Eesmärgiks on kopeerida opkoodi poolt antud registrist väärtus registrisse, millele viitab tulemregistri dekooder. Põhisiinil on näha kopeeritava andmebaidi väärtust.

“M-reg to MAR” - Selle mikrokäsu korral kopeeritakse M-registri väärtus mälu aadressiregistrisse. Selline tegevus on vajalik peamiselt salvestuskäskude sooritamiseks. Põhisiinil on näha kopeeritava andmebaidi väärtust.

“Reg C to RAM” - Tegemist on ainsa mikrokäsuga, mille korral toimub põhimällu andmete kopeerimine. Põhisiinil on näha kopeeritava andmebaidi väärtust.

“MDR to PC” - Antud mikrokäsk kopeerib mälu andmeregistri sisu käsuloendurisse. See tegevus on oluline eestkätt siirdekäskude realiseerimisel. Põhisiinil on näha kopeeritava andmebaidi väärtust.

“END Command” - See mikrokäsk asetseb peaaegu iga käsu täitmise lõpus, mille ainsaks eesmärgiks on kustutada hetkel täitmisel oleva mikrokäsu sisemus ning asendada see väärtusega 0. Käsu lõpetamine on vajalik, sest käimasoleva käsu järgi saab juhtautomaat aru mitmete teiste mikrooperatsioonide juures, kas tegemist on käsu laadimise või juba täitmisega. Käsu laadimise puhul käsuregistris on väärtus 0, aga käsu täitmise ajal on käsuregistris täidetava käsu opkood.

3.2 Protsessori operatsioonosa

Protsessori operatsioonosa on osa, kus toimub andmete tegelik töötlemine. Protsessori põhiskeem, kus on näha operatsioonosa, on leitav peatüki lõpust.

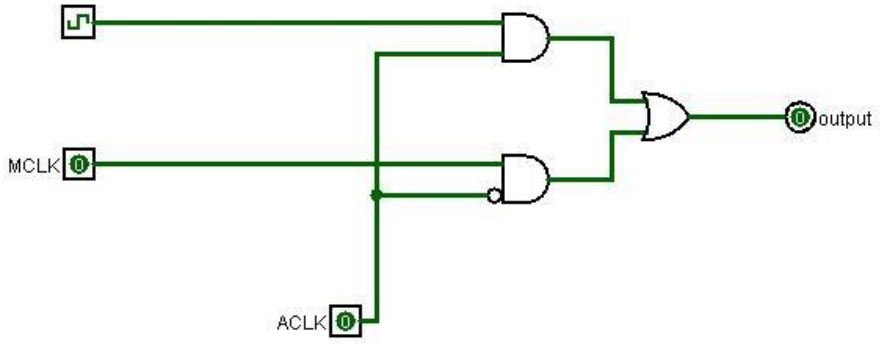
Operatsioonosa koosneb antud protsessoris järgnevatest elementidest:

- protsessori põhisiin koos sisendpuhvritega
- Taktivalija
- Käsuloendur
- Mälu aadressiregister
- Mälu andmeregister
- Aritmeetika-loogikaplokk
- Käsuregister
- Tulemregistri dekooder
- Lippude register
- Register A
- Register B
- Register C
- Register M

Protsessori põhisiin on seade, mis ühendab enesega pea kõiki komponente, mis suhtlevad omavahel 8-bitises vormingus. [5, p. 250] Käesoleva protsessori põhisiinil toimub nii käskude kui ka andmete kopeerimine. Kuna põhisiini, mis ühendab protsessori seadmeid, on ainult 1, toimub andmevahetus seadmete vahel ükshaaval, andmevahetust reguleerib juhtautomaat väljunditega. Siinil on kasutuses elementide väljundite ja siini vahel väljundpuhvrid, mis ei lase edasi seadmete poolt antavaid väärtusi põhisiinile, kui puhvrile ei ole vastavat luba antud juhtautomaadi poolt. Selline tegevus on oluline vältimaks andmete levitamist seadmetesse, kus neid ei tohi eksisteerida antud ajahetkel.

Põhisiini projekteerimine algas Logisimis siinide tundmaõppimisega, kus esialgu koostati lihtsaid skeeme, mis koosnesid peamiselt registritest, mis olid omavahel ühenduses üheainsa siiniga, selliste skeemidega eksperimenteeriti eri lahendusi sisendite ja väljundite kontrollimisel saavutamaks vigadeta suhtlust registrite vahel.

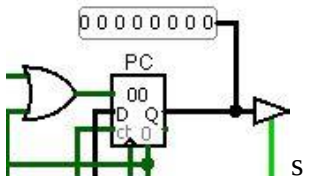
Taktivalija, joonisel on kirjas nimega „CLKSEL“, on suhteliselt lihtne multiplekseril põhinev seade, mille põhiülesandeks on lasta kasutajal valida, kas taktsignaal on Logisimi poolt genereeritud või sisestab seda kasutaja manuaalselt. Ümberlülitumine toimub sisendi „ACLK“ ümberlülitamisel. Väärtuse „0“ korral on taktivalija manuaalsel režiimil ja takte tuleb protsessori töö simuleerimiseks käsitsi sisestada, lülitades sisendit „MCLK“ korduvalt sisse ja välja. Kui sisend „ACLK“ on väärtuse „1“ peal, siis toimub automaatne taktsignaali edastamine Logismi seadetest valitud taktsignaali sagedusega.



Joonis 3. Taktivalija sisestruktuuri loogikaskeem.

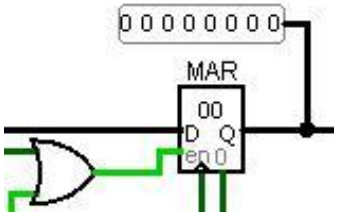
Käsuloendur (PC) lõputöö käigus valminud protsessoris on klassikalise funktsionaalsusega: see näitab järgmisena töötlusesse mineva käsu aadressi põhimälus. [5, p. 128]Taoline loendur on antud protsessori korral lähteteegis saadaval ja seda ei ole eraldi ümber konstrueeritud sellepärast, et isetehtud skeemidel ei ole võimalik Logisimis skeemiploki peale konstrueerida monitoorimisseadet, millega oleks võimalik jälgida skeemisisest tegevust kõrgemalt tasemelt. Samas taoline funktsionaalsus on Logisimi poolt antud seadmetel juba olemas.

Käsuloenduri puhul on tegemist inkrementeeritava registriga, millele saab anda suvalist lubatud pikkusega väärtust, lisaks saab registris asuvat väärtust suurendada või vähendada 1 võrra. Praeguse protsessoriga saab käsuloenduri väärtust suurendada ühe võrra või sinna sisestada kasutaja poolt soovitud väärtust.



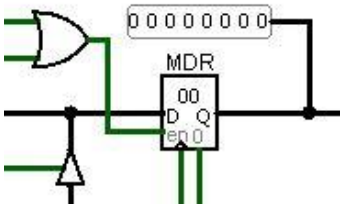
Joonis 4. Ekraanitõmmis käsuloendurist põhiskeemis.

Mälu aadressiregister ehk aadressiregister (mikrokäskude nimedes kasutatakse ka lühendit MAR) on register, mille väärtusega pöördutakse mälu poole sealt andmete lugemise ja sinna andmete kirjutamise korral. Käesolevat registrit väärtustatakse iga käsutsükli alguses, kus sinna kopeeritakse käsuloenduri väärtus, lisaks kopeeritakse sinna väärtusi lugemis- ja kirjutamiskäskude täitmiseks. Konstrueeritud on see register, kasutades Logisimis asuvast vaiketeegist elementi, nimega „register“.



Joonis 5. Ekraanitõmmis mälu aadressiregistrist põhiskeemis.

Mälu andmeregister (MDR) on register, kuhu salvestatakse mälust lugemise ajal andmebaidi väärtus enne, kui antud andmebait saadetakse edasi soovitud protsessoriossa. Salvestuskäskude korral mälu andmeregistrit ei kasutata ning andmed kopeeritakse lähteregistrist otse põhimällu. Seadme konstrueerimisel on samuti kasutatud Logisimi vaiketeegis asuvat elementi, nimega „register“.



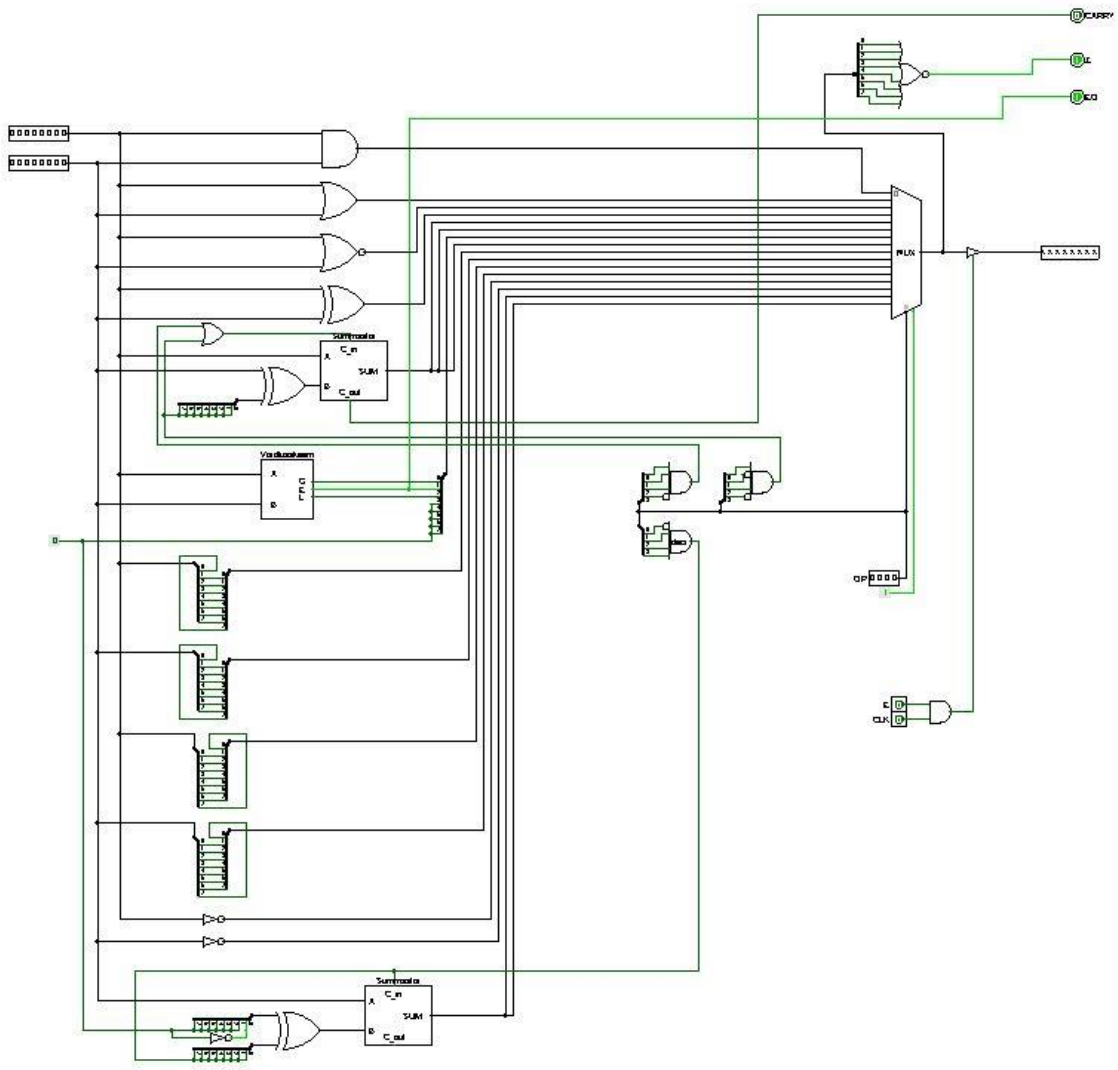
Joonis 6. Joonis mälu andmeregistrist põhiskeemis.

Aritmeetika-loogikaplokk on seade, mis tegeleb erinevate aritmeetika- ja loogikaoperatsioonidega. [5, p. 64] Kuna protsessor valmis ka aine „Arvutisüsteemid – projekt“ raames, siis käesolev element ei ole autori enese koostatud, autor määras küll seadme funktsionaalsuse.

Käesolev aritmeetika-loogikaplokk koosneb 4-bitisest operatsioonsisendist, mistõttu on võimalik sooritada kuni 16 erinevat operatsiooni. Operandide sisendid on jäigalt seotud registrite A ja B väljunditega. Järgnevalt on välja toodud loetelu ALU operatsioonidest:

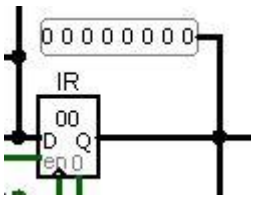
- loogiline AND
- loogiline OR
- loogiline NOR
- loogiline XOR
- liitmine ülekandeta
- liitmine ülekandega
- registrite A ja B võrdlus
- lahutamine
- registri A ringnihe vasakule
- registri B ringnihe vasakule
- registri A ringnihe paremale
- registri B ringnihe paremale
- registri B suurendamine 1 väärtuse võrra
- registri B vähendamine 1 väärtuse võrra

Käesolevat aritmeetika- ja loogikaplokki saab topeltklikiga seadmele vajutades avada ning sisemusega tutvuda, kus põhiline ehitus koosneb A ja B sisenditest, erinevatest täiturseadetest ning multiplekserist. Loogiliste operatsioonide puhul kasutati Logisimi poolt antud detaile, mis juba omasid kaht 8-bitist sisendit. Liitmist ja lahutamist on teostatud, kasutades lihtsat summaatorit, kus B-sisendi ees on XOR, mis on ühendatud liitmis- või lahutamissignaali kandva juhtmega – sellisel kujul on efektiivne demonstreerida, kuidas summaatoriga saab teostada lahutamist.. Ülekande sisseviimine on samuti realiseeritud multiplekseril.



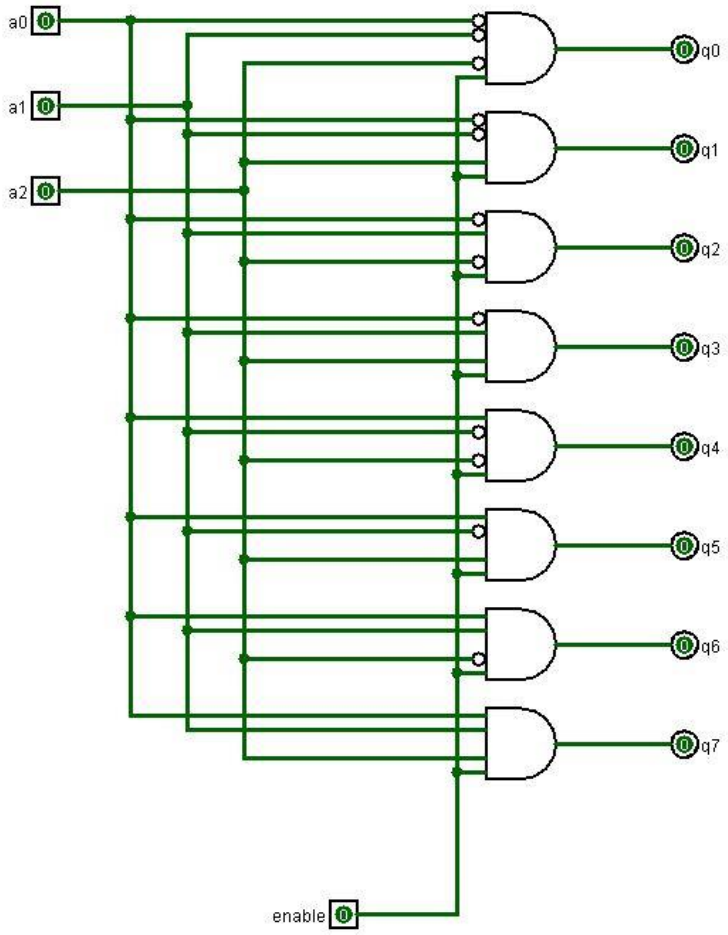
Joonis 7. ALU sisestruktuuri ekraanitõmmis.

Käsuregister on register, mis salvestab endas hetkel täitmisel olevat käsku. [5, p. 129] Andmeid hoitakse kogu käsu töötlemise ajal alates käsuregistri väärtustamise hetkest. Peale iga käsu lõppu seatakse registri väärtuseks 0, lisaks väärtustatakse selle registri väärtus nulliga asünkroonselt sisendi „Lähtesta“ aktiveerimisel. Antud protsessoril on käsuregistri opkoodi bittide väljundid juhtautomaadile sisendiks, ülejäänud kolm bitti juhitakse tulemregistri dekoodrisse.



Joonis 8: Ekraanitõmmis käsuregistrist põhiskeemis

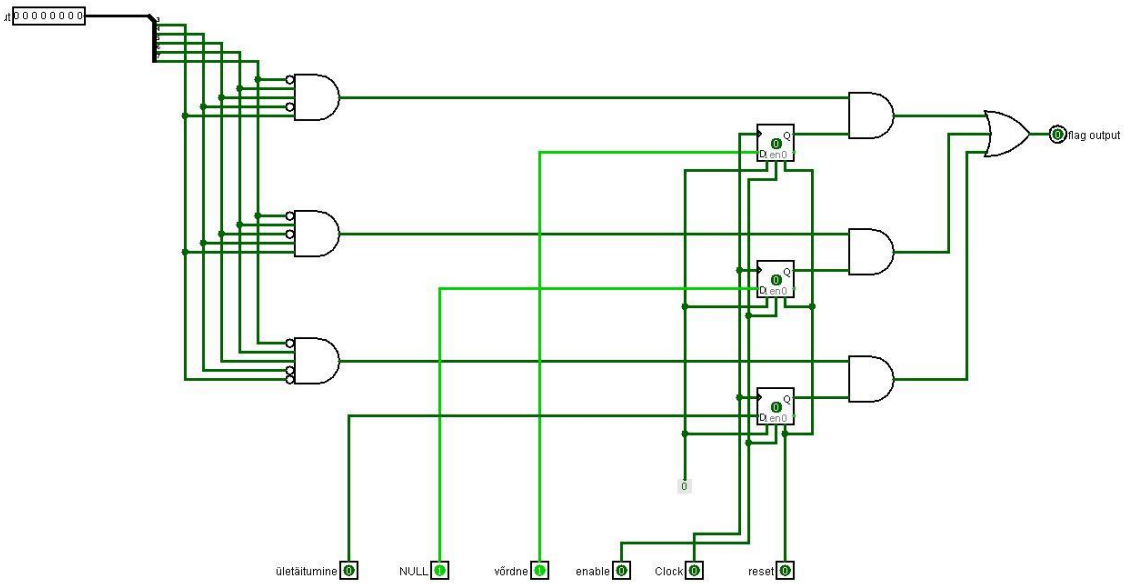
Tulemregistri dekooder on ehituselt sarnane kõikide teiste dekoodritega. Oluline on seadet mitte sassi ajada käsudekoodriga, mida antud protsessoril klassikalisel kujul ei eksisteeri. Konstrueeritud on antud seade, kasutades kaheksat AND-elementi, mille sisendid on vastavalt otseväärtustatud või inverteeritud. Käitumuslikult on seade kõrgaktiivne demultipleksor, millel on korraga ainult üks väljund väärtusega „1“. Seadmel on olemas ka lubamissisend, mille väärtus jõuab otseväärtusena igasse AND-elementi. Antud dekoodri väljundid on ühenduses protsessoris asuvate registrite sisenditega, kuhu saab paljude käskude täitmisel tulemust salvestada.



Joonis 9. Tulemregistri dekoodri sisestruktuuri ekraanitõnmis.

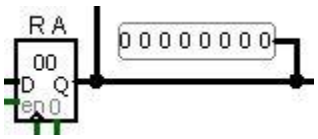
Lippude register on ainus erandlik register, mis on autori koostatud põhjusel, et juhtautomaadi sisendite kokkuhoidmisel väljastab register vaid ühe lipu väärtust korraga. Selle jaoks konstrueeriti trigerite juurde eraldi kombinatoorskeem, mis vastavate siirdekäskude opkoodide korral väljastaks vastava lipu väärtuse. Lippude enese väärtused on salvestatud D-tüüpi trigerites. Lippe on antud protsessoril kolm: ületäitumine, null ja

võrdne ning nende tulemused muutuvad aritmeetika-loogikakäske täites või protsessorit lähtestades.



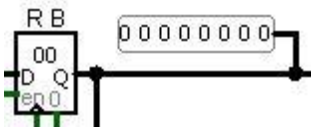
Joonis 10. Lippude registri sisestruktuuri ekraanitõmmis,

Register A on üks neljast põhiregistrist, kuhu salvestatakse nii arvutusteks vajalikke väärtusi kui ka arvutustel saadud tulemusi. Selle registri väljund on otseühenduses aritmeetika-loogikaploki A-sisendiga, seega peab aritmeetika-loogikakäksude täitmise eel veenduma, et vajalikud operandid on ka vastavates registrites. Konstrueerimisel kasututati taaskord Logisimi vaiketeegis asuvat elementi, nimega „register“.



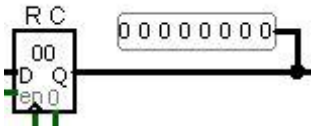
Joonis 11. Ekraanitõmmis registrist A põhiskeemis.

Register B on funktsionaalsuselt väga sarnane registriga A peamiste vahedega, et nende sisendeid aktiveerivad erinevad tulemregistri dekoodri väljundid ning registri B väljund on ühenduses aritmeetika-loogikaploki B-sisendiga. Konstrueerimisel kasutati samuti Logisimi vaiketeegis asuvat elementi, nimega „register“.Registreid A ja B eristab teistest veel andmete väärtustamine mitte tõusval vaid langeval taktifrondil. Langeva taktifrondi põhjuseks on olukorras, kus kasutatakse registreid A ja B nii sisendiks ja ühte neist ka väljundiks Samal ajahetkel on avatud nii registrite väljundid kui ka sisendid omavahel, mis tekitab olekute võidujooksu efekti. Teiste registrite väärtustamist langevalt frondile ümber tõsta ei omanud otsest vajadust ning need jäid tõusva frondi peale.



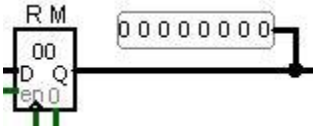
Joonis 12. Ekraanitõmmis registrist B põhiskeemis.

Register C on struktuurselt äravahetamiseni sarnane nii registri A ja B'ga, kuid omab teistsugust funktsionaalsust. Registri C väljund ega sisend ei ole otseselt seotud aritmeetika-loogikaplokiga, kuid siia registrisse on võimalik salvestada tulemusi. Lisaks on register C vaikimisi registriks andmete kopeerimisel põhimällu. Seega tuleb enne andmete valvestamist ülekantav info kopeerida just registrisse C.



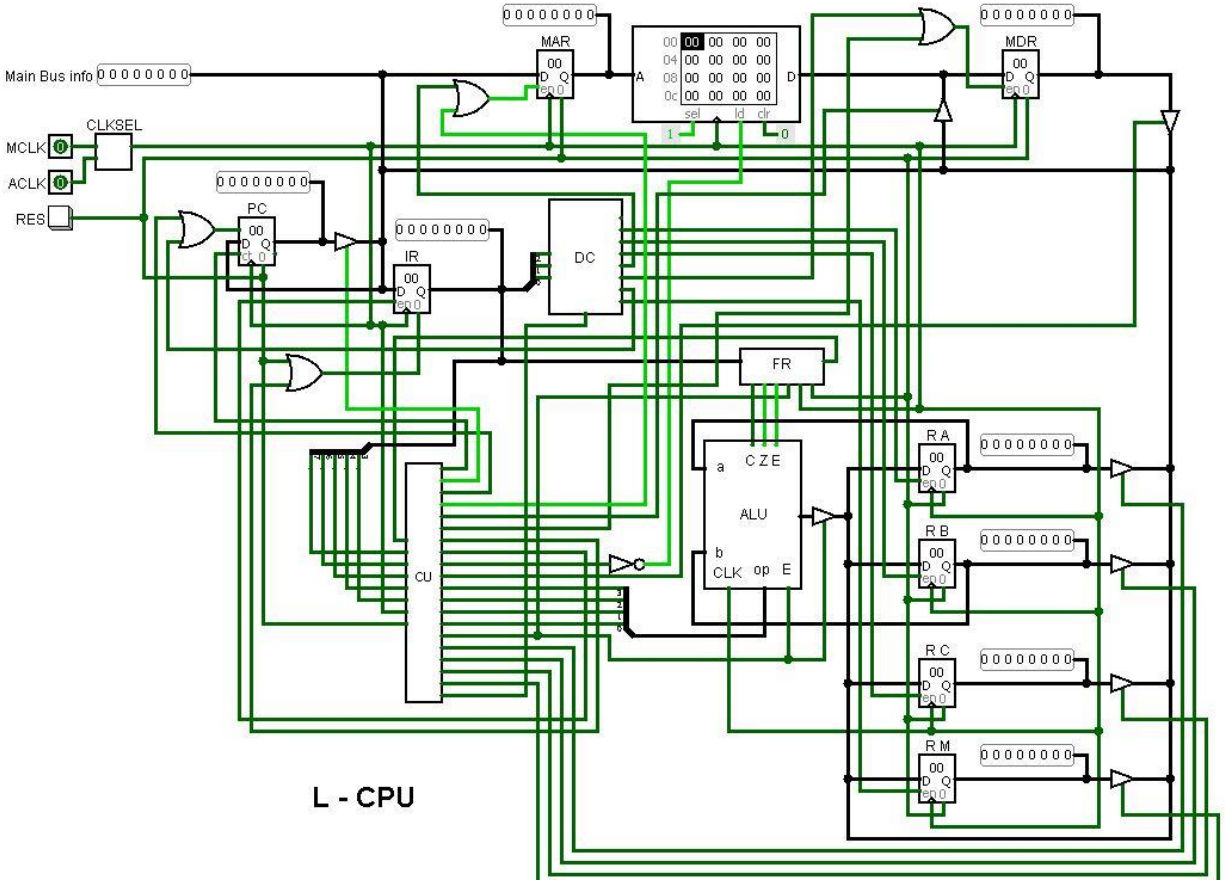
Joonis 13. Ekraanitõmmis registrist C põhiskeemis.

Register M omab samuti erifunktsionaalsust: nimelt on tegemist registriga, kus hoitakse põhimälu aadresse andmete laadimisel ja salvestamisel otsese adresseerimise puhul. Peale põhiomaduse register M ei erine kuigivõrd teistest registritest ning seda on tõhus kasutada andmete talletamiseks nii käskude kui ka aadressite osas.



Joonis 14. Ekraanitõmmis registrist M põhiskeemis.

Peatüki lõpetuseks on välja toodud ekraanitõmmis protsessori põhiskeemist sellisel kujul, nagu see näeb välja programmis Logisim.



Joonis 15. Joonis protsessori põhiskeemist Logisimis.

4 Protsessori käsustik

4.1 Käsustiku kirjeldus:

Käesoleva protsessori käsustik kuulub 2-aadressiga arvutite käsustiku alla sellepärast, et käsustikus lisaks opkoodile on võimalik vastavalt käsule kaasata kuni 2 aadressi.

Käsud jaotuvad peamiselt 5-bitiseks opkoodiks ja 3-bitiseks operandiks. Opkood defineerib täidetava käsu ning 3-bitine operandiga määratakse sihtregister, kuhu salvestatakse käsu tulemus. Käsu tulemust ei salvestata kõikide käskude puhul, näiteks salvestuskäskude puhul on operand kasutu ning väärtuseks võib sisestada 0, sellest pikemalt juba vastavates alapeatükkides.

Järgnevalt on välja toodud kolm näidet erineva pikkusega käsust:

Tühikäsk

Opkood	...

Registri kopeerimiskäsk

Opkood	Operand(sihtregistri 3-bitine kood)

Laadekäsk vahetu adresseerimisega

Opkood	Operand(sihtregistri 3-bitine kood)
Operand (8-bitine väärtus)	

Joonis 16. Erinevate käsipikkuste kujutamine.

4.2 Käskude kirjeldused käsutüüpide kaudu

Järgnevalt on välja toodud käskude kirjeldused käsutüüpide kaudu. Kõige efektiivsem on käske liigitada nende tüüpide kaudu, siis sarnaste käskude kirjeldamine on tõhusam.

Erandlikuks käsuks saab nimetada tühikäsku, mis ei tee protsessori juures midagi muud, kui hakkab pärast dekodeerimist järgmist käsku laadima. Tühikäsu tsükkel on välja toodud järgnevalt ning taktide täitumist tuleb lugeda vasakult paremale.

PC to MAR	RAM to MDR	MDR to IR	Decode IR

Joonis 17. Tühikäsu taktideks jagamine.

Opkood: 00000

Takte: 4

Operandi kasutamine tühikäsu puhul ei ole vajalik.

4.2.1 Laadekäsud

Laadekäske on kaks ja need erinevad teineteisest adresseerimise viiside poolest. Opkoodi järgne operand on oluline, sest see määrab, kuhu laetakse väärtus.

00001 - Laadida mälust, kasutades M registrit

Selle käsu puhul on tegemist otsese adresseerimisega. Andmed loetakse aadressist, mida talletab eneses M-register.

Takte: 8

00010 – Laadida mälust, kasutades aadressi, mida käsuloendur hoiab

Siin on tegemist vahetu adresseerimisega. Käsku täites laetakse justkui uut käsku, kuid käsuregistrisse kopeerimise asemel kopeeritakse andmed soovitud registrisse.

Takte: 8

Käsu 00001 taktideks jagamine

PC to MAR	RAM to MDR	MDR to IR	Decode IR	M -reg to MAR	RAM to MDR	MDR to Reg X	END Command

Käsu 00010 taktideks jagamine

PC to MAR	RAM to MDR	MDR to IR	Decode IR	PC to MAR	RAM to MDR	MDR to Reg X	END Command

Joonis 18. Laadekäskude taktideks jagamine.

4.2.2 Salvestuskäsud

Salvestuskäsud omavad samuti vahetut ja otsest adresseerimise võimalust, kuid 3-bitine operandi väärtus ei mõjuta käsu täitmist.

00100 – salvestada mällu registrist C, kasutades M-registrit aadressi jaoks

Takte: 7

00101 – salvestada põhimällu registrist C, kasutades aadressi, mida käsuloendur hoiab

Käesoleva käsu kasutamisel tuleb ettevaatlik olla, kuna käsust järgnev mälupesa kirjutatakse üle uue väärtusega, mis asetseb registris C.

Takte: 7

Käsu 00100 taktideks jagamine

PC to MAR	RAM to MDR	MDR to IR	Decode IR	M -reg to MAR	Reg C to RAM	END Command

Käsu 00101 taktideks jagamine

PC to MAR	RAM to MDR	MDR to IR	Decode IR	PC to MAR	Reg C to RAM	END Command

Joonis 19. Salvestuskäskude taktideks jagamine.

4.2.3 Aritmeetika- loogikakäsud

Aritmeetika- ja loogikakäsud on kõik samasuguste taktidega, peamine erinevus on opkoodis, mis määrab, missugust ALU funktsiooni kasutatakse. Siinsete käskude puhul on oluline kasutada 3-bitist opkoodi, sest see määrab, kuhu kirjutatakse tulemi väärtus. Kui opkoodiks sisestada 0, siis tulemuse väärtust ei kirjutata kuhugi, ainult lippude väärtused muutuvad.

Aritmeetika- ja loogikakäskude taktideks jagamine

PC to MAR	RAM to MDR	MDR to IR	Decode IR	ALU A & B	END Command

Joonis 20. Aritmeetija- ja loogikakäskude taktideks jagamine.

10000 – Registrite A ja B loogiline AND, tulemus kirjutatakse sihtregistrisse

Takte. 6

10001 – Registrite A ja B loogiline OR, tulemus kirjutatakse sihtregistrisse

Takte. 6

10010 – Registrite A ja B loogiline NOR, tulemus kirjutatakse sihtregistrisse

Takte. 6

10011 – Registrite A ja B loogiline XOR, tulemus kirjutatakse sihtregistrisse

Takte. 6

10100 – Registrite A ja B liitmine ülekandeta, tulemus kirjutatakse sihtregistrisse

Takte. 6

10101 – Registrite A ja B lahutamine, tulemus kirjutatakse sihtregistrisse

Takte. 6

10110 – Registrite A ja B võrdlus, tulemus kirjutatakse sihtregistrisse

Takte. 6

10111 – Registrite A ja B liitmine ülekandega, tulemus kirjutatakse sihtregistrisse

Takte. 6

11000 – Registri A ringnihe paremale, tulemus kirjutatakse sihtregistrisse

Takte. 6

11001 – Registri B ringnihe paremale, tulemus kirjutatakse sihtregistrisse

Takte. 6

11010 – Registri A ringnihe vasakule, tulemus kirjutatakse sihtregistrisse

Takte. 6

11011 – Registri B ringnihe vasakule, tulemus kirjutatakse sihtregistrisse

Takte. 6

11100 – Registri A inverteerimine, tulemus kirjutatakse sihtregistrisse

Registrit inverteeritakse pöördkoodi alusel, seega kõikide bittidele omistatakse vastandväärtused.

Takte. 6

11101 – Registri B inverteerimine, tulemus kirjutatakse sihtregistrisse

Registrit inverteeritakse pöördkoodi alusel, seega kõikide bittidele omistatakse vastandväärtused.

Takte. 6

11110 – Registri B dekrementeerimine, tulemus kirjutatakse sihtregistrisse

Takte. 6

11111 – Registri A inkrementeerimine, tulemus kirjutatakse sihtregistrisse

Takte. 6

4.2.4 Registrite-vahelised kopeerimiskäsud

Registrite omavahelised kopeerimiskäsud on olemuselt sarnased aritmeetika- ja loogikakäskude täitmisele taktide arvu poolest. Lühikese pikkusega sihtaadress on oluline, sest sellega defineeritakse register, kuhu andmed kopeeritakse. Tuleb märkida, et käsuloendurit sihtregistrina kasutades on võimalik teostada programmi siirdeid ehk hüppekäske.

Registritevaheliste kopeerimiskäskude taktideks jagamine

PC to MAR	RAM to MDR	MDR to IR	Decode IR	X-reg to reg-Y	END Command

Joonis 21. Registritevaheliste kopeerimiskäskude taktideks jagamine.

00110 – registri A sisu kopeerimine sihtregistrisse

Takte: 6

00111 – registri B sisu kopeerimine sihtregistrisse

Takte: 6

01000 – registri C sisu kopeerimine sihtregistrisse

Takte: 6

01001 – registri M sisu kopeerimine sihtregistrisse

Takte: 6

4.2.5 Siirdekäsud

Siirdekäsud on protsessori juures väga olulised, sest need võimaldavad teostada tingimustega siirdeid ühest programmi osast teise. Sellega on võimalik teostada nii tingimusi kui ka tsükleid programmides, andes laiad võimalused protsessori programmeerimiseks. Siirdekäskudel ei ole olulised lühikesed operandi väärtused, sestap on soovitatav need asendada väärtusega 0.

Järgnevalt on välja toodud kõik kolm tingimuskäsu opkoodi ja tingimust. Käsu täitmise pikkus taktides ei sõltu tingimusest endast, vaid tingimuse täituvusest.

01011 – Hüpata aadressile, mida hoiab käsuloendur, kui ALU tulemus on 0.

01100 – Hüpata aadressile, mida hoiab käsuloendur, kui ALU tulemus on ületäitumine.

01101 – Hüpata aadressile, mida hoiab käsuloendur, kui ALU tulemus on võrdne

Tingimuse mittetäitumise korral

PC to MAR	RAM to MDR	MDR to IR	Decode IR	JMP w flags	END Command

Tingimuse täitumise korral

PC to MAR	RAM to MDR	MDR to IR	Decode IR	JMP w flags	PC to MAR	RAM to MDR	MDR to PC	END Command

Tingimusteta siirdekäsu täitmine

PC to MAR	RAM to MDR	MDR to IR	Decode IR	PC to MAR	RAM to MDR	MDR to PC	END Command

Joonis 22. Siirdekäskude taktideks jagamine.

Lisaks tingimuslikele siirdekäskudele eksisteerib veel tingimusteta siirdekäsk:

01110 – Hüpata aadressile, mida hoiab käsuloendur

Takte: 8

5 Protsessori kasutamine ja programmeerimine

Protsessor on konstrueeritud loogikasimulaatoris Logisim ja seetõttu kasutamine toimub samas keskkonnas. Hierarhilise koostamise tõttu on alamkomponendid salvestatud omaette failidesse, kust neid saab eraldi avada ja redigeerida. Skeemi avamiseks tuleb avada peamine fail, mis on märgitud nimega „pohiskeem.circ“ Kui kõik alamskeemide failid on samas kaustas, siis avaneb skeem probleemideta. Protsessori juhtloogika põhineb mikroprogrammil, mille sisu on saadaval failis „protsessorROMsisu.txt“.

Programmeerimine toimub kas põhimällu käsitsi väärtuste sisestamist kuueteistkümnendsüsteemis või laetakse põhimällu tekstifail, kuhu on programm kirjutatud. Protsessori enese tööd saab simuleerida kas manuaalselt taktsignaali vajutades või automaatse taktigeneraatoriga.

Täpsem õpetus, kuidas protsessorit kasutada, on kirjas lisas.

6 Kokkuvõte

Käesoleva lõputöö eesmärgiks oli koostada mikroprotsessori mudel, mida saaks kasutada õppe-eesmärkidel ülikoolis. Projekti käigus valmis nii protsessor kui ka käsustik, mille hierarhiliselt koostatud skeemil on võimalik jälgida selle tööd nii registersiirete kui ka loogikaelementide tasemel. Peamisteks kitsaskohtadeks konstrueerimise ajal oli juhtautomaadi koostamine, selle testimine ning vigade parandamine ja protsessori silumine. Algse plaaniga võrreldes on peamiselt muutunud juhtautomaadi struktuur, mis algselt pidi tulema küll riistvaraline, kuid keeruka juhtautomaadi disainimise käigus selgus, et valminud skeem tulnuks liiga keeruline. Sellepärast otsustati mikroprogramse juhtimise üle, lisaks annab uus juhtimise võimaluse protsessori mikroprogrammi ümber kirjutada, sellega suureneb ka protsessori paindlikkus ning lihtsam on võimalik seadet optimeerida.

Kindlasti on võimalik antud protsessori mudelit peale projekti valmimist edasi arendada: protsessorile saab juurde lisada hulk uusi seadmeid, lisada sisend-väljundseadmeid, optimeerida protsessori tööd. Kindlasti õppeainetes kasutamisel tuleb koostada ka erinevaid ülesannete komplekte, mis baseeruksid valminud mudelil: need annakaid võimaluse tutvuda erinevate olukordadega, mis võivad tekkida masinkoodis programmeerimise juures. Üks olulisemaid täiendusi antud protsessori juures oleks veel kompilaatori valmistamine, mis oluliselt hõlbustaks programmide kirjutamist seadmele.

Kasutatud kirjandus

[1] T. Evartson, „"Arvutid 1" Kursuse ainekaart,“ TTÜ Arvutisüsteemide instituut, [Võrgumaterjal]. Available: http://www.pld.ttu.ee/~teet/a1lkaart_e.html. [Kasutatud 12 05 2017].

[2] M. Aarna, „Index of /~margit/arvutid1,“ TTÜ Arvutisüsteemide instituut, 2017. [Võrgumaterjal]. Available: http://pld.ttu.ee/~margit/arvutid1/. [Kasutatud 12 05 2017].

[3] E. Orasson, „index of /im/Elmet.Orsaaon/Arvutid_I,“ TTÜ Arvutisüsteemids instituut, 2016. [Võrgumaterjal]. Available: http://www.tud.ttu.ee/im/Elmet.Orasson/Arvutid_I/. [Kasutatud 12 05 2017].

[4] P. Ellervee, „IAY0340 - Lectures,“ TTÜ Arvutisüsteemide instituut, 09 05 2017. [Võrgumaterjal]. Available: http://ati.ttu.ee/IAY0340/schedule.html. [Kasutatud 12 05 2017].

[5] B. H. LLC, „Logicly - A logic circuit simulator for Windows and macOS - logic gates, flip-flops, computer architecture, electronics, integrated circuits,“ Bowler Hat LLC., 2008-2016. [Võrgumaterjal]. Available: https://logic.ly/. [Kasutatud 05 05 2017].

[6] „Cedar Logic Simulator download | Sourceforge.net,“ Slashdot Media, 2017. [Võrgumaterjal]. Available: https://sourceforge.net/projects/cedarlogic/. [Kasutatud 05 05 2017].

[7] S. Media, „Ksimus download | Sourceforge.net,“ Slashdot Media, 2017. [Võrgumaterjal]. Available: https://sourceforge.net/projects/ksimus.berlios/. [Kasutatud 05 05 2017].

[8] C. Burch, „About the program,“ Carl Burch, 2005. [Võrgumaterjal]. Available: http://www.cburch.com/logisim/docs/2.7/en/html/guide/about/index.html. [Kasutatud 05 05 2017].

[9] T. Evartson, Arvutitehnika riistvara : õpik kõrgkoolidele, Tallinn: Tallinna Raamatutrükikoda, 2013.

[10] „Online Cs modules: The Central Processing Unit,“ Department of Computer Science, [Võrgumaterjal]. Available: http://courses.cs.vt.edu/~csonline/index.html. [Kasutatud 2017 05 09].

[11] W. stallings, Computer organization and architecture : designing for performance / William Stallings, Pearson/Prentice Hall: Upper Saddle River (N.J.), 2006.

Lisa 1 – Näiteprogrammid

Näiteprogrammid: Lisa1 juures on välja toodud esimese näiteprogrammi ülesande kirjeldus ning lahendus lõputöö käigus valminud protsessori peal.

Ülesanne 1:

Laadida mälupesast 4d operand väärtusega 5f registrisse A

Ülesande lahendus:

Mälupesast saab laadida andmeid, kasutades kaudset adresseerimist registri M abil, kuid programmitöö käivitamise momendil on kõik protsessori siseregistrid väärtusega 0. Sestap tuleb alguses laadida registrisse M aadress, kus asub operand 5f.

Programmi kuju põhimälus on esitatud tabelina:

Käsukood 16'nd kujul	Kommentaar
17	laadida registrisse M andmebait, millele viitab käsuloendur
4d	laadimisele kuuluv andmebait
09	laadida registrisse A kasutades adresseerimiseks M-registrit

Ülesanne 2:

Korrutada mälupesades 24h ja 25h olevad väärtused 19h ja 09h ning saadud tulemus salvestada mälupessa aadressiga 26h

Ülesande lahendus:

Alguses laeme andmed mälupesadest registritesse, sellele järgnev klassikaline "liida-dekrementeeri" protsess, mille käigus lõpptulemus salvestatakse lõppregistrisse. Tingimusteta siirded on konstrueeritud süsteemil, et kuni ei ole väärtust B väärtuseni 0, liigutakse järgmise väärtuseni, mis on uuesti tagasihüppamine liitmise ja dekrementeerimiseni. Kui tulemus on lõpuks 0, hüpatakse üle tagasihüppamisest ning programm liigub töö lõpetamise suunas, milleks on andmete salvestamine põhimällu.

NB! „h" täht arvude järel rõhutab, et arvud on kirjutatud kuueteistkümnendsüsteemis, mitte tavalises kümnendsüsteemis.

Programmi kuju põhimälus:

Käsukood 16'nd kujul	Kommentaar
17	laeme M-registrisse aadressi, kust laadida
24	esimese teguri aadress
9	laeme andmed mälust aadressiga 24h
17	laeme M-registrisse aadressi, kust laadida
25	teise teguri aadress
0b	laeme andmed mälust aadressiga 25h
37	kopeerime A sisu M' registrisse
42	kopeerime C sisu B' registrisse
f3	dekrementeerime B väärtust, tulemus C'sse
32	kopeerime A sisu B'sse
a1	liidame A ja B, tulemuse kirjutame A'ga üle

42	kopeerime C väärtuse B’sse
f3	dekrementeerime B väärtust, tulemus C’sse
4a	kopeerime M väärtuse B’sse
58	hüppame ülejärgmisele käsule, kui tulemus on 0
12	aadress, kuhu hüpata
70	hüppame tagasi aadressile, kust liitsime
0a	aadress, kus oli liitmiskäsk
17	laeme aadressi, kuhu salvestada
26	aadress, kuhu tulemuse salvestamine
33	kopeerime A tulemuse registrisse C põhimällu salvestamiseks
20	kopeerime C väärtuse põhimällu aadressiga 26h

Lisa 2 – Logisimi ja protsessorimudeli kasutamise kiirjuhend

Protsessori mudel koosneb erinevatest alamdetailidest, mis asuvad ka eraldi failides. Põhiline skeem on siiski koostatud selliselt, et põhifaili avamisega avatakse ka alamskeemid ning eraldi faile avada protsessori kasutamiseks ei ole vaja.

Protsessori korrektseks ja tõrgeteta kasutamiseks on soovitatav kasutada järgnevalt:

- Käivitada programm Logisim
- Avada fail pohiskeem.circ
- Avada protsessori skeemil topeltklikiga juhtautomaadi (CU) alaskeem
- Paremklikiga ROM elemendi peal valida „Load image..:“ ning valida fail nimega protsessoriROMsisu.txt
- Protsessori kasutamise alustamiseks soovitatav vajutada RES nuppu põhiskeemil kasakul serval
- Protsessori on valmis kasutamiseks

Protsessori programmeerimiseks paremklikiga põhimälu moodulil ja valikut „Edit contents“ valides ilmub ekraanile uus aken, milles on mugav muuta põhimälus talletatavat informatsiooni. Lisaks on võimalik programme kirjutada, valides põhimälu elemendil vastava mälupesa ning kirjutades sisse käsu kuueteistkümnendsüsteemi väärtus.

Lisa 3 - Protsessori juhtautomaadi tabel.

Lipu sisend	X1	X2	32	x4	x5	x6	x7	x8	x9	d1	d2	d3	d4	PC +1	PC OUT	PC- set	MAR enable	bus to MDR	MDR input enable	IR set to 0	IR enable	RAM r/w	MDR to bus	ALU 0	ALU 1	ALU2	ALU3	ALU enable	reg A output enable	reg B output enable	reg C output enable	reg M output enable	IAD enable	dekodeeritud käsk
						PC to MAR																													
-	-	-	-	-	-	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
-	0	0	1	0	1	0	0	0	0	0	1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	salvestuskäsk
						RAM TO MDR																												
-	0	0	0	0	0	0	0	1	0	0	1	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	läheb MDR to IR'i
-	0	0	0	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	läheb MDR to Reg X'i
-	0	0	0	1	0	0	0	1	0	1	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	läheb MDR to Reg X'i
-	0	1	0	1	1	0	0	1	0	1	0	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Läheb MRD to PC
-	0	1	1	0	0	0	0	1	0	1	0	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Läheb MRD to PC
-	0	1	1	0	1	0	0	1	0	1	0	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Läheb MRD to PC
-	0	1	1	1	0	0	0	1	0	1	0	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Läheb MRD to PC
						MRD to IR																												
-	-	-	-	-	-	0	1	1	0	1	1	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	
						Decode IR																												
-	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	NOP
-	0	0	0	0	1	1	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Laadida, kasutades M-registrit
-	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Laadida, aadressiks PC-info
-	0	0	1	0	0	1	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Salvestada C'st M-reg'iga
-	0	0	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Salvestada C'st PC'ga
-	0	0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A kopeeri sihtregistrisse
-	0	0	1	1	1	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	B kopeeri sihtregistrisse
-	0	1	0	0	0	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C kopeeri sihtregistrisse
-	0	1	0	0	1	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	M kopeeri sihtregistrisse
-	1	0	0	0	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	AND A & B

Lipu sisend	X1	X2	32	x4	x5	x6	x7	x8	x9	d1	d2	d3	d4	PC +1	PC OUT	PC- set	MAR enable	bus to MDR	MDR input enable	IR set to 0	IR enable	RAM r/w	MDR to bus	ALU 0	ALU 1	ALU2	ALU3	ALU enable	reg A output enable	reg B output enable	reg C output enable	reg M output enable	IAD enable	dekodeeritud käsk
-	1	0	0	0	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	OR A & B
-	1	0	0	1	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	NOR A & B
-	1	0	0	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	XOR A & B
-	1	0	1	0	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ADD A & B
-	1	0	1	0	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	SUB A & B
-	1	0	1	1	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	võrrelda registreid A & B
-	1	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Liitmine ülekandega
-	1	1	0	0	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A ringnihe paremale
-	1	1	0	0	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	B ringnihe paremale
-	1	1	0	1	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A ringnihe vasakule
-	1	1	0	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	B ringnihe vasakule
-	1	1	1	0	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A inverteerimine
-	1	1	1	0	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	B inverteerimine
-	1	1	1	1	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	B inkrementeerimine
-	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	B dekrementeerimine
-	0	1	0	1	1	1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Hüpata, kui 0
-	0	1	1	0	0	1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Hüpata, kui on ülekanne
-	0	1	1	0	1	1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Hüpaka, kui on 0
-	0	1	1	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Tingimusteta hüpe
						MDR to reg X																												
-	-	-	-	-	-	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	
						ALU A & B																												
-	1	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	AND A & B
-	1	0	0	0	1	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	OR A & B
-	1	0	0	1	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	NOR A & B
-	1	0	0	1	1	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	1	XOR A & B
-	1	0	1	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	ADD A & B
-	1	0	1	0	1	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1	SUB A & B
-	1	0	1	1	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	1	võrrelda registreid A & B

Lipu sisend	X1	X2	32	x4	x5	x6	x7	x8	x9	d1	d2	d3	d4	PC +1	PC OUT	PC- set	MAR enable	bus to MDR	MDR input enable	IR set to 0	IR enable	RAM r/w	MDR to bus	ALU 0	ALU 1	ALU2	ALU3	ALU enable	reg A output enable	reg B output enable	reg C output enable	reg M output enable	IAD enable	dekodeeritud käsk
-	1	0	1	1	1	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	Liitmine ülekandega
-	1	1	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	A ringnihe paremale
-	1	1	0	0	1	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	1	B ringnihe paremale
-	1	1	0	1	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	1	A ringnihe vasakule
-	1	1	0	1	1	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	0	0	0	1	B ringnihe vasakule
-	1	1	1	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	1	A inverteerimine
-	1	1	1	0	1	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	0	0	1	B inverteerimine
-	1	1	1	1	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	0	0	0	0	1	B inkrementeerimine
-	1	1	1	1	1	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	1	B dekrementeerimine
						JMP w' flags																												
0	-	-	-	-	-	1	0	1	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	kui tingimus ei ole täidetud
1	-	-	-	-	-	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	kui tingimus on täidetud
						M-reg to PC																												
-	-	-	-	-	-	0	1	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	
						X-reg to reg-Y																												
-	0	0	1	1	0	0	0	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	A kopeeri sihtregistrisse
-	0	0	1	1	1	0	0	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	B kopeeri sihtregistrisse
-	0	1	0	0	0	0	0	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	C kopeeri sihtregistrisse
-	0	1	0	0	1	0	0	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	M kopeeri sihtregistrisse
						M-reg to MAR																												
-	0	0	0	0	1	0	1	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	laeme aadressi, KUST *load*
-	0	0	1	0	0	0	1	0	1	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	laeme aadressi, KUHU *load*
						MDR to PC																												
-	-	-	-	-	-	1	0	1	1	1	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	
						reg C to RAM																												
-	-	-	-	-	-	0	1	1	1	1	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	RAM'i kirjutamine
-	0	0	1	0	1	0	1	1	1	1	0	0	1	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	
						END Command																												
-	-	-	-	-	-	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	käsu lõpetamine