**Organizacija računalnikov**

**1. domača naloga**

**MiMo model CPE**

Domen Gabrovšek

63140059

1. **Zapišite zaporedje mikroukazov, ki se izvedejo pri izvedbi strojnega ukaza »SW Rd, Immed« (primer: »SW r3, 150«).**

* *Ta ukaz je že implementiran v datoteki z definicijami mikroprogramskih realizacij*
* *Ob vsakem mikroukazu krajše opišite njegovo delovanje (stanje podatkovne enote, kaj se pravzaprav takrat zgodi, itd ...)*

**65: addrsel=pc imload=1**

**addrsel=immed datawrite=1 datasel=dreg, goto pcincr**

**addrsel=pc:** Preko multiplekserja na naslovno vodilo pošljemo vrednost (naslov) iz programskega števca. Naslov nam pove iz katere pomnilniške besede v RAM pomnilniku bomo brali.

**Imload=1:** Ko prižgemo kontrolno linijo imload, signaliziramo da bomo pisali iz podatkovnega vodila v takojšnji register.

**addrsel=immed:** Preko multiplekserja na naslovno vodilo pošljemo vrednost (naslov) iz takojšnjega registra. Naslov nam pove iz katere pomnilniške besede v RAM pomnilniku bomo brali.

**datawrite=1:** Ko prižgemo kontrolno linijo datawrite, signaliziramo da bomo pisali v RAM pomnilnik.

**datasel=dreg:** Kontrolna linija datasel kontrolira kaj bomo vpisali v RAM pomnilnik. Na voljo imamo štiri možnosti: trenutno vrednost programskega števca, vsebino d registra, vsebino t registra in vsebino izhoda aritmetično-logične enote, v našem primeru izberemo d register.

**goto pcincr:** Povečamo programski števec za 1, da dobimo naslov naslednjega ukaza.

1. **Iz seznama strojnih ukazov, ki jih podpira zbirnik izberite vsaj tri in zanje podajte ustrezna zaporedja mikroukazov za njihovo realizacijo. Izbrani ukazi naj bodo iz različnih skupin in bolj zahtevni oz. netrivijalni - sestavljeni naj bodo iz vsaj 4 ali še bolje več mikroukazov (vaša uspešnost se bo merila tudi po tem kriteriju). Kratko opišite realizacijo in predvsem mikroprogramske zapise dodanih strojnih ukazov v mikro-zbirniku:**

* *potrebno je določiti realizacijo strojnega ukaza s pomočjo mikroukazov*
* *zaporedje mikroukazov je potrebno dodati v datoteko "basic\_microcode.def" in jo vnesti in  uporabiti v MiMo modelu.*

# sub Rd,Rs,Rt

# Rd <- Rs - Rt PC <- PC + 1

1: aluop=sub op2sel=treg dwrite=1 regsrc=aluout, goto fetch

**aluop=sub:** Kontrolna linija aluop določa katero operacijo bo izvedla aritmetično-logična enota. MiMo model je implementiran na način, da izvede vseh 16 operacij, ki jih podpira potem pa s pomočjo multiplekserja izberemo katero vrednost želimo poslati na izhod. V tem primeru izberemo izhod iz odštevalnika - razliko.

**op2sel=treg:** Kontrolna linija op2sel določa kaj bomo uporabili za drugi vhod v aritmetično-logično enoto. Na voljo imamo štiri možnost: vrednost registra treg, vrednost takojšnjega registra, konstanto 0 in konstanto 1. V tem primeru i zberemo vrednost registra treg.

**dwrite=1:** Kontrolna linija dwrite določa, da bomo vrednost iz pomnilnika RAM vpisali v register d.

**regsrc=aluout:** Določa vhodni izvor za register kamor bomo vpisali vrednost, v našem primeru razliko. Register se razloči iz ukaza tako, da se ustrezno nastavi register d.

**goto fetch:** Skoči na prevzem novega ukaza.

Poleg ukaza sub Rd, Rs, Rt sem implementiral še ostalih 13 operacij, ki jih podpira aritmetično-logična enota. Skupno jih podpira 16, vendar je ukaz add že implementiran, ukaz sub je opisan zgoraj, ukaz not pa mi je delal težave in sem ga izpustil. Večina korakov je za vseh 16 operacij enaka, pri vseh aritmetično-logična enota sprejme dva vhoda, izračuna vseh 16 operacij in nato vrne rezultat glede na to kako smo določili kontrolno linijo aluop.

Implementirani ukazi za delo z aritmetično-logično enoto:

* 1: aluop=sub op2sel=treg dwrite=1 regsrc=aluout, goto fetch (odštevanje)
* 2: aluop=mul op2sel=treg dwrite=1 regsrc=aluout, goto fetch (množenje)
* 3: aluop=div op2sel=treg dwrite=1 regsrc=aluout, goto fetch (deljenje)
* 4: aluop=rem op2sel=treg dwrite=1 regsrc=aluout, goto fetch (ostanek pri deljenju)
* 5: aluop=and op2sel=treg dwrite=1 regsrc=aluout, goto fetch (in)
* 6: aluop=or op2sel=treg dwrite=1 regsrc=aluout, goto fetch (ali)
* 7: aluop=xor op2sel=treg dwrite=1 regsrc=aluout, goto fetch (ekskluzivni ali)
* 8: aluop=nand op2sel=treg dwrite=1 regsrc=aluout, goto fetch (negirani in)
* 9: aluop=nor op2sel=treg dwrite=1 regsrc=aluout, goto fetch (negirani ali)
* 11: aluop=lsl op2sel=treg dwrite=1 regsrc=aluout, goto fetch (logični pomik v levo)
* 12: aluop=lsr op2sel=treg dwrite=1 regsrc=aluout, goto fetch (logični pomik v desno)
* 13: aluop=asr op2sel=treg dwrite=1 regsrc=aluout, goto fetch (aritmetični pomik v desno)
* 14: aluop=rol op2sel=treg dwrite=1 regsrc=aluout, goto fetch (rotacija v levo)
* 15: aluop=ror op2sel=treg dwrite=1 regsrc=aluout, goto fetch (rotacija v desno)

# jeqz Rs,immed

# if Rs == 0, PC <- immed else PC <- PC + 2

39: addrsel=pc imload=1

aluop=sub op2sel=const0, if z then jump else pcincr

**addrsel=pc:** Preko multiplekserja na naslovno vodilo pošljemo vrednost (naslov) iz programskega števca. Naslov nam pove iz katere pomnilniške besede v RAM pomnilniku bomo brali.

**imload=1:** Ko prižgemo kontrolno linijo imload, signaliziramo da bomo pisali iz podatkovnega vodila v takojšnji register.

**aluop=sub:** Kontrolna linija aluop določa katero operacijo bo izvedla aritmetično-logična enota. MiMo model je implementiran na način, da izvede vseh 16 operacij, ki jih podpira potem pa s pomočjo multiplekserja izberemo katero vrednost želimo poslati na izhod. V tem primeru izberemo izhod iz odštevalnika - razliko.

**op2sel=const0:** Kontrolna linija op2sel določa kaj bomo uporabili za drugi vhod v aritmetično-logično enoto. Na voljo imamo štiri možnost: vrednost registra treg, vrednost takojšnjega registra, konstanto 0 in konstanto 1. V tem primeru i zberemo konstanto 0 s katero bomo kasneje izvedli logiko za preverjanje ali je potrebno izvesti skok ali ne.

**if z then jump else pcincr:** Pri operaciji odštevanja smo odšteli konstanto 0 od registra s in tem nastavili zastavice. Če je postavljena zastavica z potem skočimo na jump, drugače povečamo programski števec.

# swri Rd,Rs,Rt (74)

# M[Rs+Rt] <- Rd PC <- PC + 1

74: aluop=add op2sel=treg

addrsel=aluout datawrite=1 datasel=dreg, goto fetch

**aluop=add:** Kontrolna linija aluop določa katero operacijo bo izvedla aritmetično-logična enota. MiMo model je implementiran na način, da izvede vseh 16 operacij, ki jih podpira potem pa s pomočjo multiplekserja izberemo katero vrednost želimo poslati na izhod. V tem primeru izberemo izhod iz seštevalnika – vsoto.

**op2sel=treg:** Kontrolna linija op2sel določa kaj bomo uporabili za drugi vhod v aritmetično-logično enoto. Na voljo imamo štiri možnost: vrednost registra treg, vrednost takojšnjega registra, konstanto 0 in konstanto 1. V tem primeru izberemo register t.

**addrsel=aluout:** Preko multiplekserja na naslovno vodilo pošljemo vrednost (naslov) izhoda aritmetično-logične enote. Naslov nam pove iz katere pomnilniške besede v RAM pomnilniku bomo brali.

**datawrite=1:** Ko prižgemo kontrolno linijo datawrite, signaliziramo da bomo pisali v RAM pomnilnik.

**datasel=dreg:** Kontrolna linija datasel kontrolira kaj bomo vpisali v RAM pomnilnik. Na voljo imamo štiri možnosti: trenutno vrednost programskega števca, vsebino d registra, vsebino t registra in vsebino izhoda aritmetično-logične enote, v našem primeru izberemo d register.

**goto fetch:** Skoči na prevzem novega ukaza.

1. **Nove strojne ukaze iz 2. naloge uporabite v lastnem testnem programu v zbirniku in preizkusite njegovo delovanje. Napišite tak program, da se bodo dodani ukazi temeljito preizkusili. Razložite vsebino ustreznih strojnih ukazov. Opišite dogajanje ob vsakem strojnem ukazu in določite, koliko urinih period traja vsak od njih in program v celoti (zapišite trajanja za vsak ukaz in celoten program v posebni tabeli).**

Programi za testiranje posameznih ukazov se nahajajo v arhivu v mapi »testni programi«. Vsak program je v svoji datoteki in v komentarjih je opisano izvajanje.

Tabela za trajanje posameznih ukazov. Trajanje sem preveril tako, da sem v logisimu uporabljal Ctrl + T, dva pritiska štejeta za eno urino periodo. Pri ukazu li se je prvi ukaz izvajal 3 urine periode, vsi nadaljni li ukazi pa 4 urine periode.

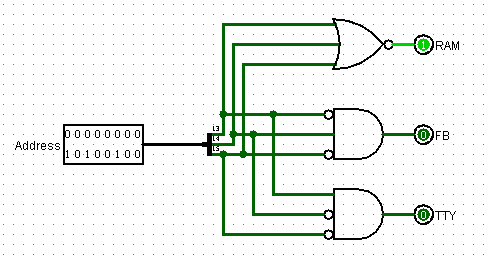
|  |  |
| --- | --- |
| Ukaz | Trajanje v urinih periodah |
| li (prvič) | 3 |
| li (vsi naslednji) | 4 |
| sw | 4 |
| jnez | 6 |
| jeqz | 6 |
| swri | 24 |
| add | 4 |
| sub | 4 |
| div | 4 |
| mul | 4 |
| rem | 4 |
| and | 4 |
| or | 4 |
| xor | 4 |
| nand | 4 |
| nor | 4 |
| lsl | 4 |
| lsr | 4 |
| asr | 4 |
| rol | 4 |
| ror | 4 |

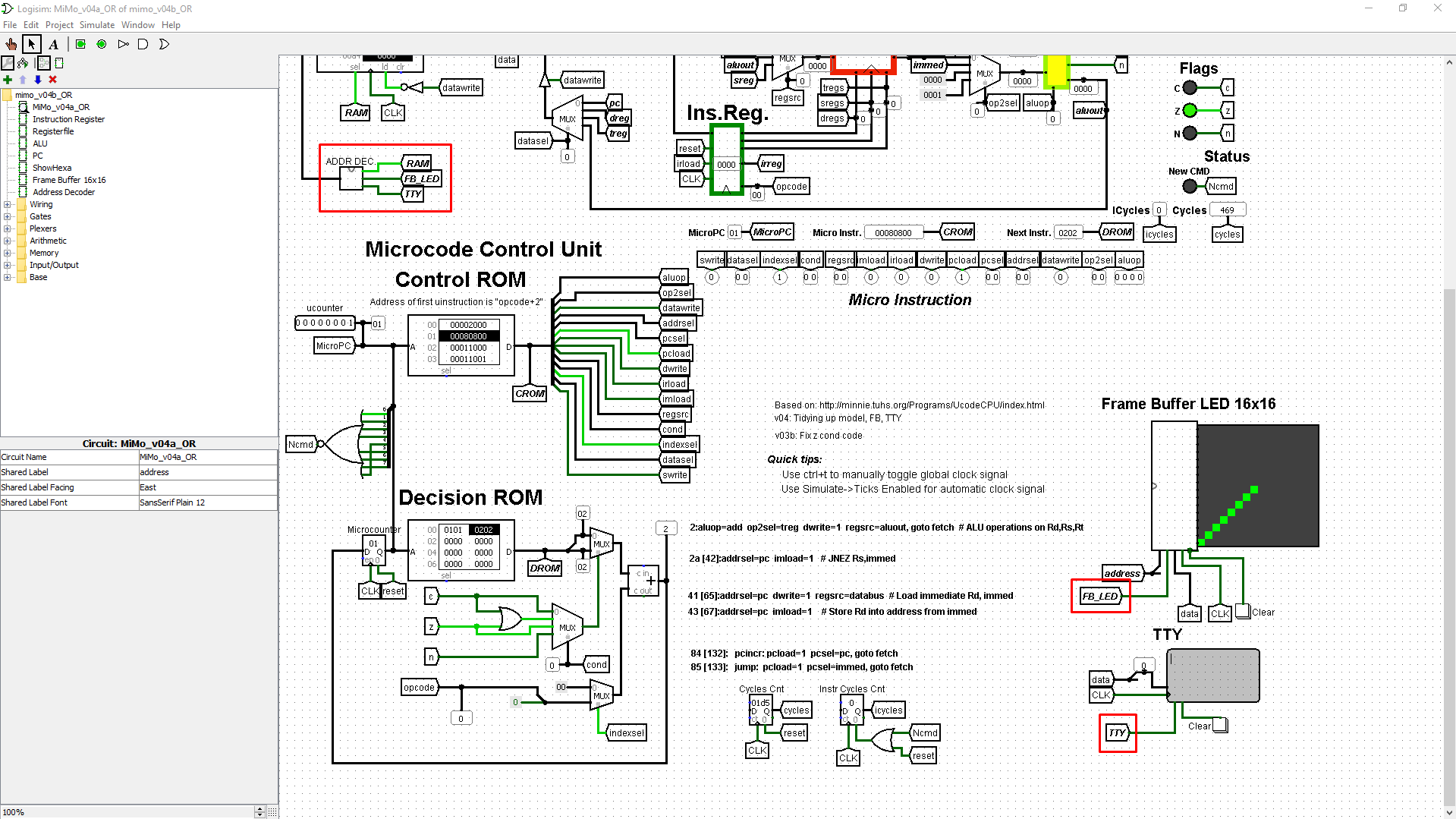
1. **Pravilno umestite v pomnilniški prostor izhodni napravi FB 16x16 in TTY po načelu "pomnilniško preslikanega vhoda/izhoda"- lahko uporabite nepopolno naslovno dekodiranje. Poskrbite, da se bo testni program za IO napravi po tej spremembi pravilno izvajal - torej se bosta izhodni napravi pojavili samo za operacijo pisanja in res samo na njima dodeljenih naslovih. Spremenite testni program, da bo izrisoval vzorec na FB napravi po vaši zamisli. Dodatno gradivo "OR\_Naslovno\_dekodiranje\_gradivo.pdf" je vključeno v distribuciji za MiMo model, smo pa to temo pojasnili tudi na predavanjih in vajah.**

Na začetku imamo v naslovnem dekodirniku vse izhode nastavljene na 1. Ker imamo več različnih naprav katere želimo naslavljati, sem uporabil nepopolno naslovno dekodiranje. Naslovi, ki se začenjo z 000 bodo šli na RAM, naslovi ki se začenjo z 010 bodo šli na »frame buffer«, naslovi ki se začnejo z 100 pa bodo šli na TTY. Zadnji (četrti) izhod sem odstranil, ker ga ne potrebujemo.

Pri realizaciji sem uporabil:

* Splitter, ki na vhodu sprejme 16 bitov in zadnje tri bite (13,14,15) pripelje na izhod. Vsak bit ima svoj izhod. Uporabimo jih kot vhod v logična vrata v nadeljavanju logike.
* RAM - NOR vrata, zato ker moramo negirati vse tri vhode da v primeru treh ničel na vhodih dobimo enico na izhod.
* FB - AND vrata, ki imajo negirani prvi in zadnji vhod, da v primeru da na vhod dobimo 010, bomo na izhod dobili enico
* TTY - AND vrata, ki imajo negirana zadnja dva vhoda, da v primeru da na vhod dobimo 100, bomo na izhod dobili enico





Po realizaciji logike za naslovni dekodirnik, je bilo potrebno še ustrezno povezati izhode dekodirnika in vhode v FB in TTY, na sliki označeno z rdečimi pravokotniki. Na sliki je tudi razviden vzorec, prikazan na FB-ju katerega sem si izmislil in implementiral v programu test-fb.s, ki se nahaja v mapi »testni programi«.

1. **Opišite dogajanje v podatkovni enoti po vseh elementarnih korakih, ki bi bili potrebni za pravilno realizacijo pisalne in bralne operacije na/iz sklada (push in pop). Elementarne korake lahko opišete vsaj tekstovno - podajte čimveč podrobnosti. Lahko dodate tudi še dejanski mikroprogram. Napišite tudi, kje vse bi lahko sklad koristno uporabili ? Podajte vse potrebne podrobnosti.**

**Viri s katerimi sem si pomagal:**

* [**https://www.rapidtables.com/convert/number/hex-to-binary.html**](https://www.rapidtables.com/convert/number/hex-to-binary.html) **(splošna pomoč pri testiranju)**
* [**https://chortle.ccsu.edu/assemblytutorial/Chapter-14/ass14\_13.html**](https://chortle.ccsu.edu/assemblytutorial/Chapter-14/ass14_13.html) **(pomoč pri testiranju asr ukaza)**
* **<https://chortle.ccsu.edu/AssemblyTutorial/Chapter-12/ass12_2.html> (pomoč pri testiranju lsl ukaza)**
* [**https://chortle.ccsu.edu/AssemblyTutorial/Chapter-12/ass12\_6.html**](https://chortle.ccsu.edu/AssemblyTutorial/Chapter-12/ass12_6.html) **(pomoč pri testiranju lsr ukaza)**
* **<https://minnie.tuhs.org/Programs/UcodeCPU/index.html> (pomoč pri splošnem razumevanju MiMo procesorja)**