Organizacija računalnikov
Študijsko leto 2023/2024

1. domača naloga - MiMo model CPE

Marcel Homšak
63210103

Ljubljana, 12. februar 2024

Kazalo

1 Kratek povzetek vsebine

V seminarski nalogi je opisana implementacija MiMo modela CPU. Glavni cilj je bil realizirati strojne ukaze na nivoju mikrozbirnika ter testirati njihovo delovanje v simulacijskem okolju.

Spoznal sem delovanje in strukturo MiMo modela CPE ter implementiral večino mikroukazov, 70 od 75. Napisal sem več testnih programov za preverjanje delovanja procesorja v simulacijskem okolju Logisim. Dodal sem RGB video zaslon 32x32 pikslov ter preveril njegovo delovanje. Najbolj zanimiva mi je bila prav ta implementacija lastne naprave.

2 Opis ukaza SW Rd,Immed

Ukaz SW Rd,Immed se izvaja 5 urinih period.

Ko prevedemo assembly (.s) datoteko, v kateri imamo preprost main: sw r3, 65535 ukaz dobimo sledečo .ram datoteko:

```
0000: 00008203  1000001000000011        main:   sw      r3, 65535
0001: 0000ffff  1111111111111111
```

Če izvedemo to kodo, ki je sedaj prevedena v strojni jezik, v Logisimu lahko opazimo sledeče izvajanje:

1. **Mikroukazi:** `addrsel=pc irload=1.`
   Ukaz se naloži v ukazni register (instruction register), s kodo 8203_{16}, kar je binarno ekvivalentno 1000001000000011_2. Prvih 7 bitov predstavlja operacijsko kodo (opcode) v našem primeru je to 1000001_2 kar je enako 65_{10}. 65_{10} predstavlja namreč kodo za ukaz SW. Ostalih 9 bitov se razdeli v tri skupine po 3 bite in sicer predstavljajo po vrsti: treg, sreg, dreg. Dreg je v našem primeru 3_{10} kar pomeni, da želimo shraniti vrednost iz registra r3.

2. **Mikroukazi:** `pcload=1 pcsel=pc, opcode_jump.`
   Programski števec (PC) se poveča za 1 in skoči se na naslednji naslov.

3. **Mikroukazi:** `addrsel=pc imload=1.`
   Na podatkovnem vodilu (data bus) se nahaja vrednost FFFF (65535) - dobi se iz trenutnega naslova v RAM pomnilniku. Ta vrednost nam pove naslov, v katerega želimo shraniti vsebino iz registra r3. V takojšni register (immediate register) se naloži ta vrednost.

4. **Mikroukazi:** `addrsel=immed datawrite=1 datasel=dreg.`
   Vrednost iz dreg (v tem primeru registra r3) se zapiše v pomnilnik na lokacijo FFFF (65535).

5. **Mikroukazi:** `goto pcincr (pcincr:  pcload=1 pcsel=pc, goto fetch).`
   Programski števec se poveča še za 1 - skupno 2, saj ta en ukaz zasede 2 pomnilniški besedi v pomnilniku. Procesor je tako pripravljen na nov ukaz.

3 Realizacija strojnih ukazov s pomočjo mikroukazov

V datoteki *list_of_instructions.txt* je naštetih in podprtih 75 ukazov. Od tega sem jih realiziral 66, 4 ukazi so že bili realizirani, torej skupno je realiziranih 70/75 ukazov.
Spodaj je naštetih in opisanih nekaj realiziranih ukazov.

addi Rd,Rs,immed (16)

Urina perioda	Mikroukazi	Opis mikroukazov
1.	`addrsel=pc irload=1`	ukaz naložimo v ukazni register
2.	`pcload=1 pcsel=pc, opcode_jump`	PC=PC+1, skok na opcode
3.	`addrsel=pc imload=1`	v takojšnji register naložimo takojšnji operand
4.	`aluop=add op2sel=immed dwrite=1 regsrc=aluout`	v Rd zapišemo vsoto Rs in takojšnega operanda
5.	`goto pcincr`	PC=PC+1

Tabela 1: Ukaz addi Rd,Rs,immed (16)

subc Rd,Rs,Rt,immed (32)

Urina perioda	Mikroukazi	Opis mikroukazov
1.	`addrsel=pc irload=1`	ukaz naložimo v ukazni register
2.	`pcload=1 pcsel=pc, opcode_jump`	PC=PC+1, skok na opcode
3.	`addrsel=pc imload=1 aluop=sub op2sel=treg dwrite=1 regsrc=aluout`	v takojšnji register naložimo takojšnji operand in hkrati lahko tudi izračunamo in zapišemo rezultat operacije Rd=Rs-Rt
4.	`if c then jump else pcincr`	če je carry enak 1 potem skočimo na immed, sicer PC=PC+1

Tabela 2: Ukaz subc Rd,Rs,Rt,immed (32)

jle Rs,Rt,immed (36)

Urina perioda	Mikroukazi	Opis mikroukazov
1.	`addrsel=pc irload=1`	ukaz naložimo v ukazni register
2.	`pcload=1 pcsel=pc, opcode_jump`	PC=PC+1, skok na opcode
3.	`addrsel=pc imload=1 aluop=sub op2sel=treg`	v takojšnji register naložimo takojšnji operand in hkrati tudi izračunamo rezultat operacije Rs-Rt
4.	`if norz then jump else pcincr`	če je rezultat negativen ali pa enak 0, potem skočimo na immed, sicer PC=PC+1

Tabela 3: Ukaz jle Rs,Rt,immed (36)

blt Rs,Rt,immed (50)

Urina perioda	Mikroukazi	Opis mikroukazov
1.	`addrsel=pc irload=1`	ukaz naložimo v ukazni register
2.	`pcload=1 pcsel=pc, opcode_jump`	PC=PC+1, skok na opcode
3.	`addrsel=pc imload=1 aluop=sub op2sel=treg`	v takojšnji register naložimo takojšnji operand in hkrati tudi izračunamo rezultat operacije Rs-Rt
4.	`if n then jumppl else pcincr`	če je rezultat negativen skočimo na PC+immed, sicer PC=PC+1

Tabela 4: Ukaz blt Rs,Rt,immed (50)

swi Rd,Rs,immed (67)

Urina perioda	Mikroukazi	Opis mikroukazov
1.	`addrsel=pc irload=1`	ukaz naložimo v ukazni register
2.	`pcload=1 pcsel=pc, opcode_jump`	PC=PC+1, skok na opcode
3.	`addrsel=pc imload=1`	v takojšnji register naložimo takojšnji operand
4.	`aluop=add op2sel=immed addrsel=aluout datawrite=1 datasel=dreg`	vsebino Rd shranimo v pomnilnik na lokacijo izračunano s seštevanjem Rs in immed; M[Rs+immed]=Rd
5.	`goto pcincr`	PC=PC+1

Tabela 5: Ukaz swi Rd,Rs,immed (67)

clr Rs (71)

Urina perioda	Mikroukazi	Opis mikroukazov
1.	`addrsel=pc irload=1`	ukaz naložimo v ukazni register
2.	`pcload=1 pcsel=pc, opcode_jump`	PC=PC+1, skok na opcode
3.	`aluop=mul op2sel=const0 regsrc=aluout swrite=1`	regiser nastavimo na 0 z operacijo množenja s konstanto 0; Rs=Rs*0

Tabela 6: Ukaz clr Rs (71)

4 Testni programi

Za vsako skupino ukazov, sem izbral vsaj en ukaz in ga tudi s pomočjo Logisima stestiral za pravilnost delovanja. Teh programov je več, vendar niso praktični.

Pri tem programu sem preveril, če se na koncu v register r1 res zapiše vrednost 0x000a.

```
1 main: li r1, 3
2       addi r1, r1, 7
```

Pri tem programu sem preveril, če se carry zastavica res prižge in tako pride do skoka na naslov 7.

```
1 main: li r1, 3
2       li r2, 5
3       subc r3, r1, r2, 7
```

Pri tem programu, sem preveril, če se vrednost registra r1 res stalno povečuje za 1.

```
1 main: li r1, 10
2 loop: addi r1, r1, 1
3       br loop
```

Teh manjših programov je kar nekaj. Zaradi preprostosti teh programov pa sem se odločil napisati še en večji program, v katerem sem poizkusil uporabiti čim več različnih implementiranih ukazov. Napisal sem program, ki računa kvadratni koren števila. Za algoritem sem uporabil bisekcijo. Psevdokoda programa:

Algorithm 1 Algoritem za računanje kvadratnega korena

1: $vnos \leftarrow 100$
2: $spodnja\_meja \leftarrow 0$
3: $zgornja\_meja \leftarrow vnos$
4: $dovoljena\_razlika \leftarrow 1$
5: $razlika \leftarrow zgornja\_meja - spodnja\_meja$
6: **while** $razlika > 1$ **do**
7: $\quad zgornja\_meja \leftarrow zgornja\_meja + 1$
8: $\quad srednja\_točka \leftarrow (zgornja\_meja + spodnja\_meja) \div 2$
9: $\quad kvadrat \leftarrow srednja\_točka^2$
10: $\quad$ **if** $kvadrat = vnos$ **then**
11: $\quad\quad rezultat \leftarrow srednja\_točka$
12: $\quad\quad$ **break**
13: $\quad$ **else if** $kvadrat > vnos$ **then**
14: $\quad\quad zgornja\_meja \leftarrow srednja\_točka$
15: $\quad$ **else if** $kvadrat < vnos$ **then**
16: $\quad\quad spodnja\_meja \leftarrow srednja\_točka$
17: $\quad$ **end if**
18: $\quad razlika \leftarrow zgornja\_meja - spodnja\_meja$
19: **end while**
20: $rezultat \leftarrow spodnja\_meja$

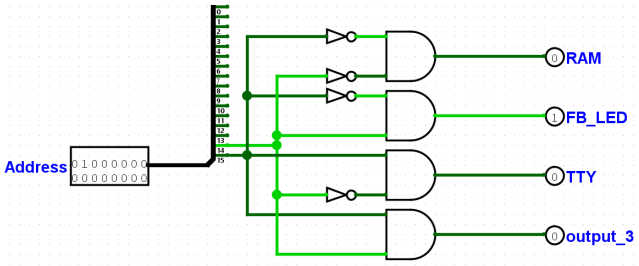
Program za računanje korena sledi na naslednji strani.

```
# r0: razlika mej
# r1: vnos
# r2: spodnja meja iskanja
# r3: zgornja meja iskanja
# r4: srednja točka
# r5: kvadrat srednje točke
# r6: rezultat (koren vnosa)
# r7: dovoljena razlika mej

start:      li r1, 361          # vnos
            li r2, 0            # spodnja meja iskanja = 0
            move r3, r1         # zgornja meja iskanja = vnos
            li r7, 1            # dovoljena razlika mej = 1

search:     addi r3, r3, 1      # zgornjo mejo povečamo za 1

            # srednja točka je (spodnja_meja + zgornja_meja)/2
            add r4, r2, r3
            lsri r4, r4, 1

            mul r5, r4, r4      # r5 = kvadrat srednje točke

            # če je kvadrat srednje točke enak vnosu
            jeq r5, r1, endperfect
            # če je kvadrat srednje točke manjši od vnosa
            jlt r5, r1, updatelow
            # če je kvadrat srednje točke večji od vnosa
            jgt r5, r1, updateup

updatelow:  move r2, r4         # spodnja meja = srednja točka
            jmp checkdiff       # poglej če sta meji blizu

updateup:   move r3, r4         # zgornja meja = srednja točka
            jmp checkdiff       # poglej če sta meji blizu

checkdiff:  sub r0, r3, r2      # r0 = zgornja_meja - spodnja_meja
            jle r0, r7, endlow  # razlika mej <= 1?
            jmp search          # sicer nadaljuj z iskanjem

endlow:     move r6, r2         # rezultat = spodnja meja
            jmp save            # shrani in končaj

endperfect: move r6, r4         # rezultat = natančen koren
            jmp save            # shrani in končaj

save:       sw r6, 48           # shrani v pomnilnik M[48] = rezultat
            jmp end

end:        jmp end             # mrtva zanka
```

5 Umestitev izhodnih naprav FB 16x16 in TTY v pomnilniški prostor

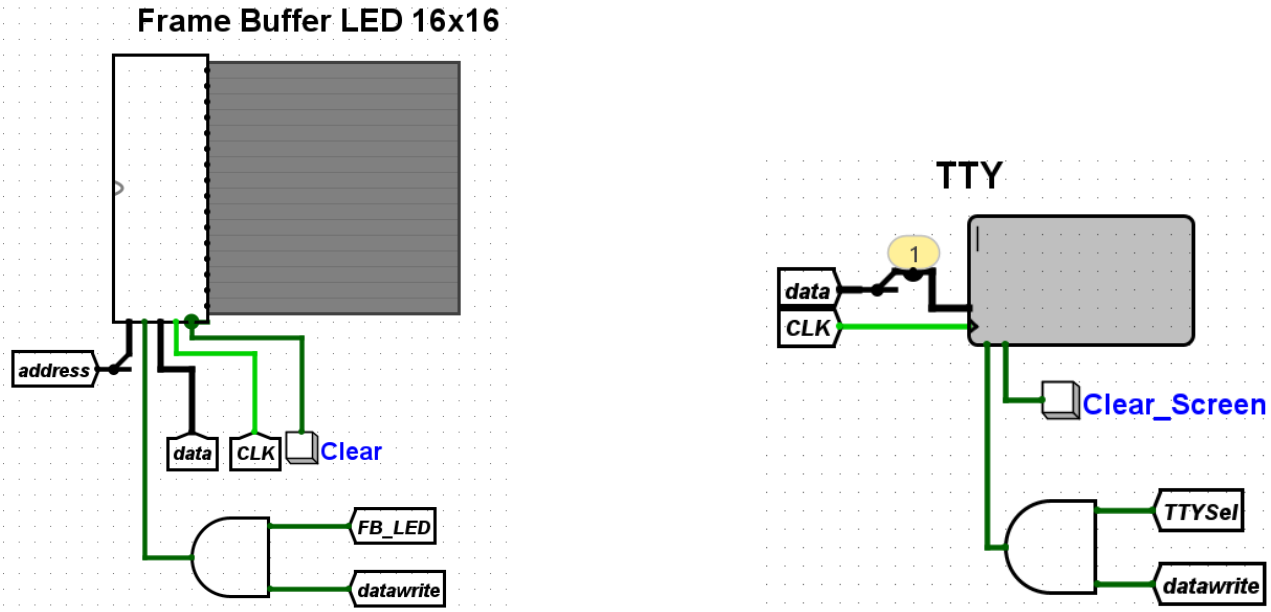
Za uspešno in pravilno umestitev izhodnih napravi FB 16x16 in TTY v pomnilniški prostor je bilo potrebno dokončati address decoder vezje. Naslov se navezuje na napravo odvisno od zadnjih dveh bitov naslova. *RAM* ima zadnja dva bita nastavljena oba na 0, *FB_LED* ima 01, *TTY* 10 in naša zadnja izbirna naprava 11. Za dopolnitev vezja sem potreboval splitter, vzel zadnja dva MSB bita ter jih čez AND vrata povezal z napravo. Da sem negiral bite sem še uporabil NOT vrata.

RAM:	$00xxxxxxxxxxxxx$
FB_LED:	$01xxxxxxxxxxxxx$
TTY:	$10xxxxxxxxxxxxx$
output_3:	$11xxxxxxxxxxxxx$



Slika 1: Address Decoder

Dodatno je bilo še potrebno nastaviti, da napravi nista vedno v stanju 1, temveč se uporabljata samo v stanju pisanja. Uporabil sem AND vrata, in na ta vrata povezal kot vhoda tunela datawrite ter izhod naslovnega dekoderja za specifično napravo.

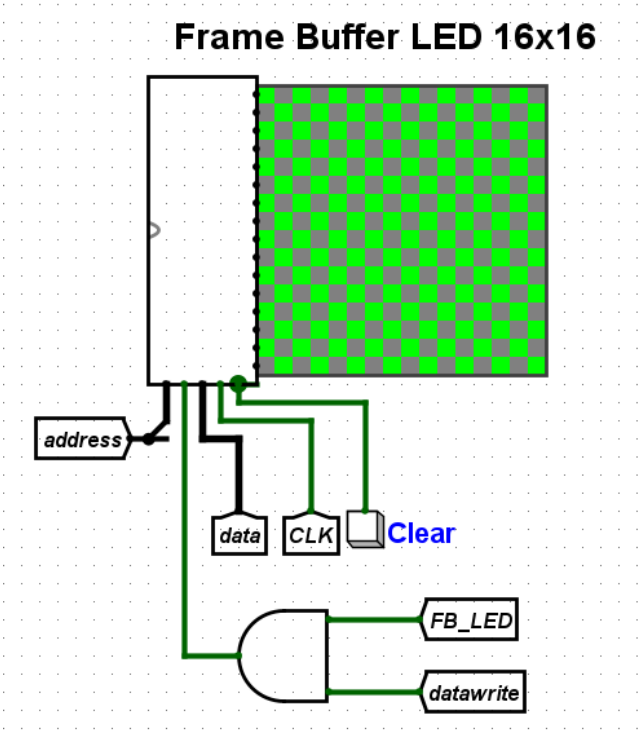


(a) Frame Buffer LED 16x16 **(b)** TTY

Slika 2: Povezave vhodno izhodnih naprav

Za izris vzorca na FB napravi sem se odločil za vzorec šahovnice. Koda:

```
main:   li r0, 21845        # 0101010101010101b
        li r1, 43690        # 1010101010101010b
        li r2, 16384        # začetni naslov
        li r3, 16399        # končni naslov
        li r4, 0            # naslovni odmik
        li r5, 0            # ostanek pri deljenju

loop:   remi r5, r4, 2      # r5 = naslovni odmik % 2
        beqz r5, store1     # pojdi na store1, če je r5%2==0
        br store2           # sicer pojdi na store2

store1: swri  r0, r2, r4    # shrani vsebino r0 na naslov r2 + r4
        br incr
store2: swri  r1, r2, r4    # shrani vsebino r1 na naslov r2 + r4
        br incr

incr:   inc r4              # inkrementiraj naslovni odmik
        blt r2, r3, loop    # ponavljaj do končnega naslova
```



Slika 3: Vzorec šahovnice

Link do posnetka generiranja šahovnice: www.youtube.com/watch?v=Pv5YD-bGgLE

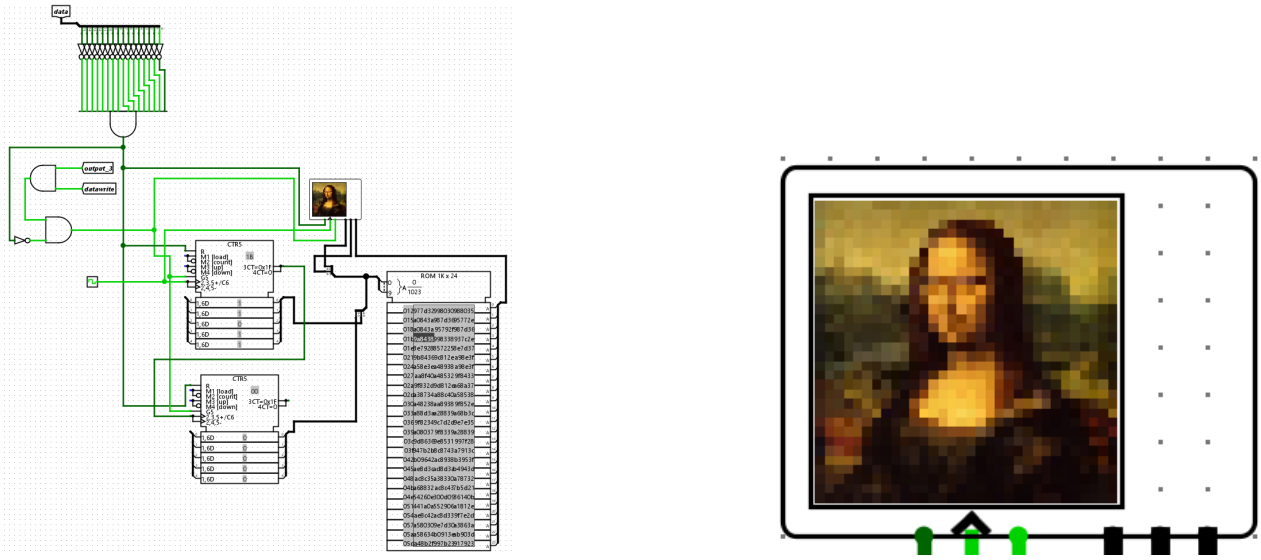
6 Opis vključitve dodatne vhodno izhodne naprave

Za vključitev naprave po lastni izbiri sem izbral RGB Video zaslon velikosti 32x32. Uporabil sem dva števca, ki štejeta od 0 do 31, določata namreč X in Y koordinate na zaslonu. Na začetku sta oba nastavljena na 0 (levi zgornji del zaslona), na koncu pa sta oba 31 (desni spodnji del zaslona). Vsakič, ko se števec poveča, se tudi naslov v ROM pomnilniku poveča, kar določa naslednji piksel slike, ki se bo izrisal na zaslonu.

Podobno kot pri napravah Frame Buffer in TTY, sem moral tudi tukaj nastaviti, da naprava ni vedno aktivna, ter omogočiti pisanje le takrat, ko je signal datawrite nastavljen na 1.

Koda, ki piše po zaslonu naprave (piksle bere iz ROM pomnilnika):

```
1  main:  li r0, 0
2         li r1, 49152      # address of my device
3         li r2, 1          # 1 == write
4         li r3, 0          # 0 == clear
5         swri r3, r1, r0   # clear display
6
7  store: swri r2, r1, r0   # write to display
8         br store
```



(a) Arhitektura naprave

(b) Zaslon

Slika 4: Moja naprava

7 Literatura

- https://www.youtube.com/watch?v=zdMhGxRWutQ

- https://www.youtube.com/watch?v=xJ8Ll2qJQR4