1. domača naloga - MiMo model CPE

1. *Jasno zapišite število in zaporedje mikroukazov, ki se izvedejo pri izvedbi strojnega ukaza "SW Rd,Immed" (primer: "SW r3, 65535").*
    - *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....)*

```
# sw Rd,immed     ; Store Rd into M[immed] Rd->M[immed];
65: addrsel=pc  imload=1
    addrsel=immed  datawrite=1  datasel=dreg, goto pcincr
#   pcload=1  pcsel=pc, goto fetch
```

Število ukazov: 5 + 1 (za PC povečat, sestavljen iz 7 ukazov) -> 6

Zaporedje ukazov:

- addrsel=pc
    - izbere naslov iz PC
- imload=1
    - prižge bit za pisanje v takojšnji register
- addrsel=immed
    - izbere naslov iz takojšnega operanda
- datawrite=1
    - prižge bit za zapisovanje v pomnilnik
- datasel=dreg,
    - izbere D register za vpis
- goto pcincr
    - Poveča PC za 2

2. *Iz seznama strojnih ukazov, ki jih podpira zbirnik izberite vsaj štiri in zanje podajte ustrezna zaporedja mikroukazov za njihovo realizacijo. Izbrani ukazi naj bodo bolj zahtevni oz. netrivijalni (op.koda večja od 15) in iz različnih skupin - sestavljeni naj bodo iz vsaj 4 ali še bolje več mikroukazov (vaša uspešnost se bo merila tudi po tem kriteriju). Kratko opišite realizacijo in razložite 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 ter uporabiti v MiMo modelu.*

1. Naložimo takojšnji operand, ALU nastavimo na seštevanje, kot drugi operand izberemo takojšnjega, prižgemo bit za pisanje v D register, za vpisovanje v register pa izberemo izhod ALU. PC povečamo za 2 ker smo brali takojšnji operand, ki je na posebej naslovu. Podobno za ostale operacije ALU s takojšnjim operandom, samo zamenjamo ALU operacijo.

```
#addi Rd,Rs,immed (16)
#    Rd <- Rs + immed     PC <- PC + 2
16: addrsel=pc imload=1                  # bere immed register v immed
    aluop=add  op2sel=immed  dwrite=1  regsrc=aluout, goto pcincr
```

2. Naložimo takojšnji operand, ALU nastavimo na odštevanje, kot drugi operand nastavimo T register, odštejemo S in T register in če sta enaka se prižge bit »z« ter skoči na naslov enak takojšnjemu operandu. Če sta različna poveča PC za 2. Podobno pri ostalih skokih gledamo glede na različno prižgane bite.

```
#jeq Rs,Rt,immed (33)
#    if Rs == Rt, PC <- immed else  PC <- PC + 2
33: addrsel=pc imload=1
    aluop=sub  op2sel=treg, if z then jump else pcincr
```

3. Naložimo takojšnji operand, ALU nastavimo na seštevanje, kot drugi operand izberemo takojšnjega, prižgemo bit za pisanje v S register, za vpisovanje v register pa izberemo izhod ALU, da se shrani v S registru. Za naslov izberemo S register, ki smo ga prej izračunali, prižgemo bit za pisanje v D register, vpišemo pa vrednost iz pomnilnika. PC povečamo za 2, ker smo brali takojšnji operand. Podobno za ostala branja in shranjevanja iz pomnilnika.

```
#lwi Rd,Rs,immed (66)
#    Rd <- M[Rs+immed]    PC <- PC + 2
66: addrsel=pc  imload=1
    aluop=add  op2sel=immed swrite=1 regsrc=aluout
    addrsel=sreg dwrite=1  regsrc=databus, goto pcincr
```

4. ALU nastavimo na odštevanje, drugi operand izberemo konstanto 1 in prižgemo bit za pisanje v S register, za vpisano vrednost pa izberemo izhod ALU. To nam zmanjša vrednost na S registru za 1. ALU nastavimo na negacijo, drugi operand izberemo konstanto 0 (ni potrebno, samo za lažje sledenje programu), prižgemo bit za pisanje v S register, za vpisano vrednost pa izberemo izhod ALU. Povečamo PC za 1.

```
#neg  Rs (72)
#    Rs <-  -Rs          PC <- PC + 1
72: aluop=sub op2sel=const1 swrite=1 regsrc=aluout
    aluop=not op2sel=const0 swrite=1 regsrc=aluout, goto fetch
```

3. *Nove strojne ukaze iz 2. naloge uporabite v lastnem testnem programu (enem ali večih) v zbirniku in preizkusite njihovo delovanje. Napišite tak(e) program(e), 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 vsi skupaj (zapišite trajanja za vsak ukaz v številu urinih period v posebni tabeli).*

Ukaz	addi	jeq	lwi	neg	skupaj
St. urinih period	5	5	6	4	20

```
main:   li      r1, 0
        li      r2, 10
        addi    r1, r1, 5
        subi    r1, r1, -5
        addi    r1, r1, -5
        subi    r1, r1, 5
        addi    r1, r2, 0
        jeq     r1, r2, naprej

        addi    r1, r2, 100

naprej: addi    r1, r2, -10
        jeq     r1, r2, naprej
        neg     r1
        neg     r2

        addi    r1, r1, -5
        neg     r1
        addi    r2, r2, 5
        neg     r2

        lwi     r3, r1, -2
```

Naložimo 0 v R1 in 10 v R2

R1 prištejemo 5 in odštejemo -5 … 0 -> 10

R1 prištejemo -5 in odštejemo 5 … 10 -> 0

R1 prištejemo vrednost iz R2
Če je R1 enak R2 potem preskočimo seštevanje
R2 + 100 -> R1

R1 <- R2 – 10 … 10 -> 0
Če bi bila še vedno enaka bi skočil na »naprej«
Negiramo R1 … 0 -> ffff
Negiramo R2 … 10 -> ff

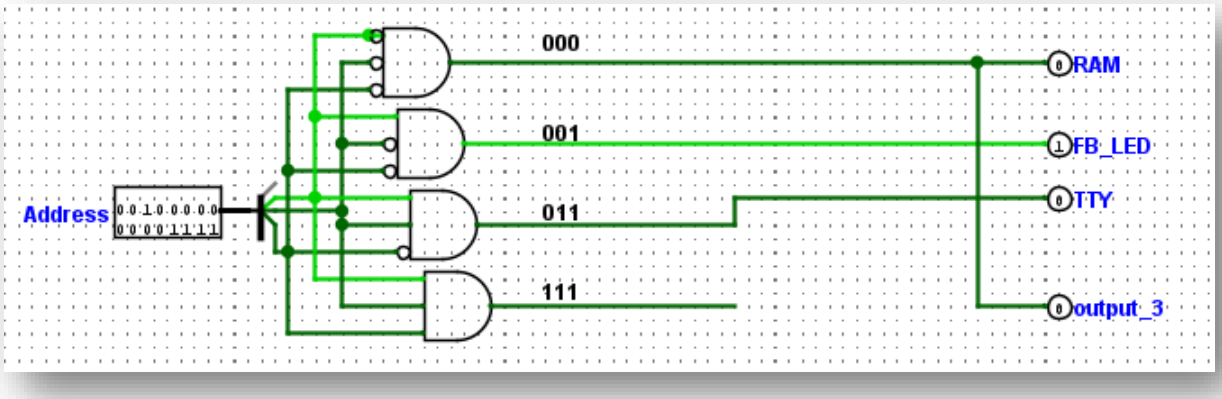
R1 odštejemo 5
Ponovno negiramo R1 … 0

R2 dodamo 5 in negiramo

V R3 naložimo vrednost iz naslova [R1 + (-2)] -> [3]

4. *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 za pisanje 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..



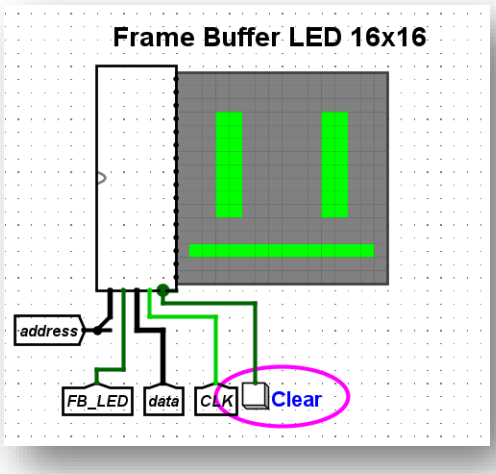
Addres decoder gleda zadnje 3 bite (bit 15, 14, 13) in na podlagi teh odloči katera naprava lahko takrat deluje (priže bit, dodeljen tej napravi).

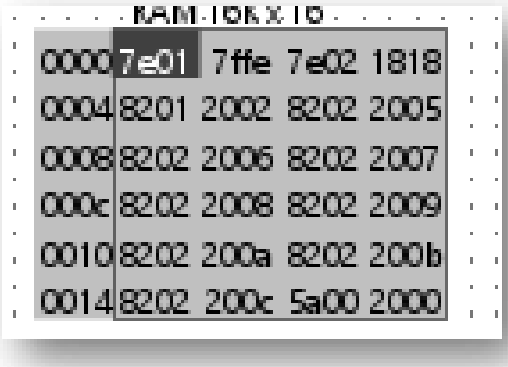
000 -> RAM (0000-2000)

001 -> FB_LED (2000-3FFF)

011 -> TTY (6000-7FFF)

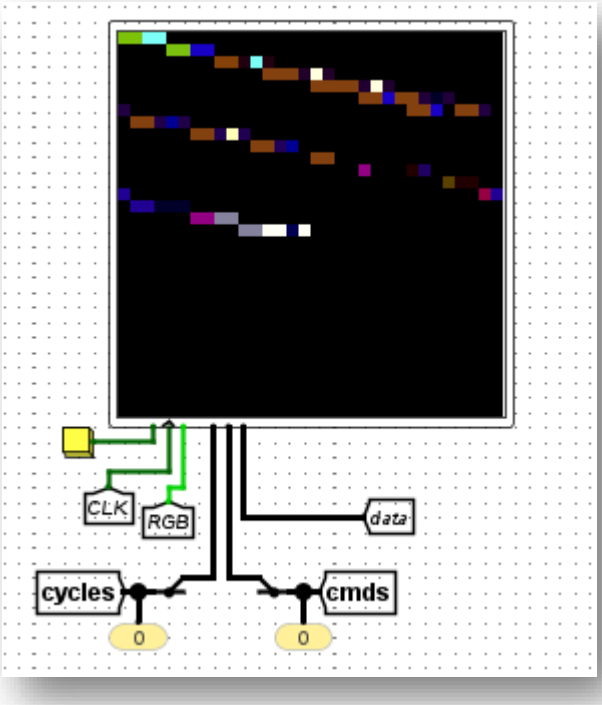
Vsi ostali naslovi se bi izvajali v prazno, oz nimajo dodeljene nobene naprave.





Na prvi sliki se vidi slika, ki nastane po izvajanju kode na desni sliki. V R1 je vrednost za usta in v R2 je vrednost za oči smeškota. Nato pa zapiše vrendosti v FB_LED.

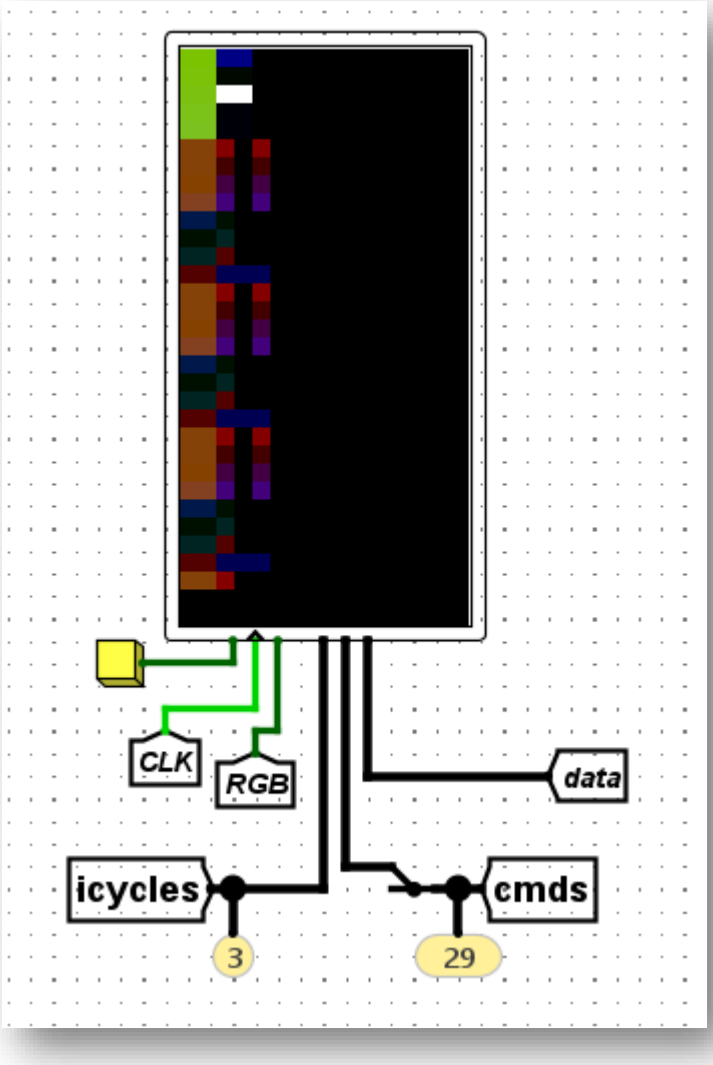
5. *V MiMo modelu je pripravljen prostor za še eno vhodno izhodno napravo. Opišite vsa potrebna opravila za pravilno vključitev dodatne vhodno izhodne naprave v MiMo model in jo tudi realizirajte (vsaj v enostavnejši obliki) ter preizkusite z ustreznim programom v zbirniku. Podajte jasen opis vseh korakov z vsemi potrebnimi podrobnostmi.*



Odločil sem se dodati RGB zaslon. Ima 6 vhodov.

1) Reset – z kliku na rumen gumb resetira zaslon v na povsem črno
2) Clock – tunel CLK (časovnik)
3) Write enable – tunel RGB (prižge ko je pravi naslov)
4) X coordinate – št. vseh ciklov
5) Y coordinate – st. izvedenih ukazov
6) Data in 565 RGB (16 bit) – tunel data (navadno vrednost registra)

Ta naprava deluje na istih naslovih kot RAM, saj grafično predstavlja izvajanje ukazov iz RAM-a. Prvi ukaz začne levo zgoraj in proti desni, drugi ukaz eno vrstico nižje itd.



Druga različica te naprave je vizualizacija dolžine ukaza v programu. V tem primeru je spremenjena vrednost vhoda X koordinate. Vhod je št. ciklov v posameznem ukazu. Za vsak ukaz v programu začne barvati od leve prosti desni, vsak ukaz v svoji vrstici.

Iz te naprave se hitro razloči zanke programa, opazujemo samo ponavljajoče vzorce na sliki. Slabost naprave je prostor zaslona. Možnost ga je še povečat vendar tudi to ima omejitev na 256 ukazov. Ko pride do dna (torej 256 ukaza) začne barvati spet na vrhu in prebava prejšne piksle. Možno je to popraviti tako, da bi še enkat prepisal vse prejšnje ukaze za eno nazaj in na koncu dodal nov ukaz, vednar bi to zelo upočasnilo fukcionalnost programa.