1. DOMAČA NALOGA PRI PREDMETU ORGANIZACIJA RAČUNALNIKOV

1. Zapišite zaporedje mikroukazov, ki se izvedejo pri izvedbi strojnega ukaza "LI Rd,Immed" (primer: "LI r3,150")

63: addrsel=pc dwrite=1 regsrc=databus, goto pcincr

- **addrsel=pc**, nastavi naslovno vodilo na vrednost programskega števca
- **dwrite=1**, omogoči vpis v d-register
- **regsrc=databus**, nastavi vsebino, ki bo vpisana v register na podatkovno vodilo (vsebina podatkovnega vodila je v tem primeru takojšnji operand, ki se bo v tem primeru vpisala v d-register)
- **goto pcincr**, programski števec se poveča za 1, nato pa se pokliče fetch kjer se PC najprej naloži v IR nato pa se še enkrat poveča.

2. Realizacija mikroukazov

Realiziral sem 22 mikroukazov, poizkusal sem zajeti vse skupine/vrste ukazov. Spodaj sledijo realizacije, pri bolj zahtevnejših/zanimivih ukazih pa tudi kratek komentar. Podobo kot sta že podana ukaza jump in pcincr sems am dodal še ukaz jumprel:

jumprel: pcload=1 pcsel=pcimmed, goto fetch

ki ga kasneje uporabljam pri vseh ukazih, ki potrbujejo relativni skok (PC + IMMED)

```
# sub Rd,Rs,Rt (1) Rd <- Rs * RtPC <- PC + 1
1:    aluop=sub  op2sel=treg  dwrite=1  regsrc=aluout, goto fetch
```

```
# mul Rd,Rs,Rt (2) Rd <- Rs * Rt      PC <- PC + 1
2:    aluop=mul op2sel=treg dwrite=1 regsrc=aluout, goto fetch
```

```
# lsl Rd,Rs,Rt (11) Rd <- Rs << Rt         PC <- PC + 1
11: aluop=lsl op2sel=treg dwrite=1 regsrc=aluout, goto fetch
```

```
# lsr Rd,Rs,Rt (12) Rd <- Rs >> Rt         PC <- PC + 1
12: aluop=lsr op2sel=treg dwrite=1 regsrc=aluout, goto fetch
```

```
# asr Rd,Rs,Rt (13)  Rd <- Rs >> Rt (filled bits are the sign bit)
     PC <- PC + 1
13: aluop=asr op2sel=treg dwrite=1 regsrc=aluout, goto fetch
```

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

```
# subi Rd,Rs,immed (17) Rd <- Rs - immed   PC <- PC + 2
17: addrsel=pc imload=1
    op2sel=immed aluop=sub regsrc=aluout dwrite=1, goto pcincr
```

```
# addc Rd,Rs,Rt,immed (31)
#    Rd <- Rs + Rt
#    if carry set, PC <- immed else  PC <- PC + 2
31: addrsel=pc imload=1
    aluop=add op2sel=treg regsrc=aluout dwrite=1, if c then jump
else pcincr
```

```
# 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
```

```
# beq Rs,Rt,immed (46)
#    if Rs == Rt, PC <- PC + immed else  PC <- PC + 2
46: addrsel=pc imload=1
    aluop=sub op2sel=treg, if z then jumprel else pcincr
```

Najprej se naloži immed, nato pa se v ALU enoti odštejeta Rs in Rt ob čemer se postavijo tudi zastavice, če se postavi zastavica Z se izvede jumprel sicer pa pa se izvede pcincr

```
# bne Rs,Rt,immed (47)
#    if Rs != Rt, PC <- PC + immed else  PC <- PC + 2
47: addrsel=pc imload=1
    aluop=sub op2sel=treg, if z then pcincr else jumprel
```

```
# br immed (58)
#    PC <- PC + immed
58: addrsel=pc imload=1, goto jumprel
```

```
# jsr immed (59)
#    R7--
#    M[R7] <- PC + 2, i.e. skip the current 2-word instruction
#    PC <- immed
59: addrsel=pc imload=1
    pcsel=pc pcload=1
    aluop=sub op2sel=const1 swrite=1 regsrc=aluout
    datawrite=1 addrsel=sreg datasel=pc, goto jump
```

Ukaz jsr (**J**ump to **S**ub**R**outine) najprej naloži takojšnji operand, nato poveča programski števec zatem pa za 1 zmanjša kazalec sklada (SP se nahaja v registru 7 in ga je porebno predhodnje nastaviti), nato pa se vrednost programskega števca zapiše na naslov, ki je shranjen v registru 7.

```
# rts  (60)
#     PC <- M[R7]
#     R7++
60: addrsel=sreg imload=1
    pcsel=immed pcload=1
    aluop=add op2sel=const1 swrite=1 regsrc=aluout, goto fetch
```

Ukaz rts (**R**e**T**urn from **S**ubroutine) najprej vsebino iz naslova ki je shranjen v SP (R7/Sreg) naloži v immed, nato pa najprej vsebino immed naloži v PC , ter PC poveča za 1 in skoči na naslednji ukaz.

```
#inc Rs (61)
#     Rs <- Rs + 1          PC <- PC + 1
61: aluop=add op2sel=const1 regsrc=aluout swrite=1, goto fetch
```

```
#dec Rs (62)
#     Rs <- Rs - 1          PC <- PC + 1
62: aluop=sub op2sel=const1 regsrc=aluout swrite=1, goto fetch
```

```
# lw Rd,immed (64)
#     Rd <- M[immed]        PC <- PC + 2
64: addrsel=pc  imload=1
      addrsel=immed regsrc=databus dwrite=1, goto pcincr
```

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

```
# swi Rd,Rs,immed (67)
#     M[Rs+immed] <- Rd     PC <- PC + 2
67: addrsel=pc  imload=1
    aluop=add op2sel=immed addrsel=aluout datasel=dreg datawrite=1,
goto pcincr
```

```
# push Rd (68)
#     R7--
#     M[R7] <- Rd                PC <- PC + 1
68: aluop=sub op2sel=const1 regsrc=aluout swrite=1
    addrsel=sreg datasel=dreg datawrite=1, goto fetch
```

```
# pop  Rd (69)
#     Rd <- M[R7]
#     R7++                   PC <- PC + 1
69: addrsel=sreg regsrc=dat abus dwrite=1
    aluop=add op2sel=const1 regsrc=aluout swrite=1, goto fetch
```

```
# move Rd,Rs (70)
#     Rd <- Rs               PC <- PC + 1
70: regsrc=sreg dwrite=1, goto fetch
```

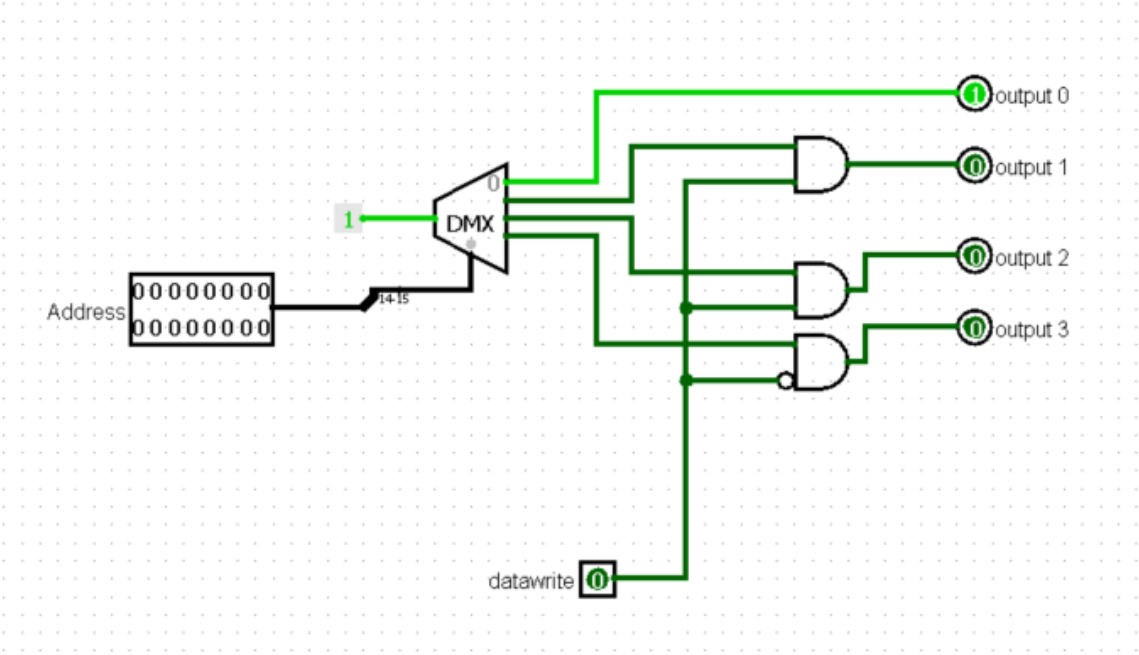
3. Testiranje ukazov

Ukaze (ne vse) sem uporabil v programu, ki je tudi del moje dodatne naloge, kjer sem v MiMo model dodal še vhodno napravo joystick in delovanje "sestavljenih " ukazov stm in ldm. Za testiranje vhodne naprave joystick pa sem napisal program, s katerim lahko premikamo piko po 16x16 matriki. Program bi lahko bil napisan bistveno bolj učinkovito (priložena je tudi bolj optimizirana oblika programa), a sem hotel vanj vključiti tudi skoke v podprograme in ukaza stm in ldm.

```
                                #     Št. Urinih period (vključno z fetch)
main:   li r7, 0x0fff                   #     4
        li r5, 0                        #     4
loop:   jsr prep_horizontal             #     7
        jsr prep_vertical               #     7
        beq r1, r5, skip                #     5
        jsr reset                       #     7
skip:   jsr write                       #     7
        br loop

reset:  sw r0, 0x4fff                   #     5
        move r5, r2                     #     3
        rts                             #     5

write:  swi r0, r1, 0x4000              #     5
        rts                             #     5



prep_horizontal:        stm {r1-r2}     #     8
                        lw r2, 0xC010   #     5
                        li r1, 1        #     4
                        lsl r0, r1, r2  #     3
                        ldm {r1-r2}     #     8
                        rts             #     5

prep_vertical:          lw r1, 0xC011   #     5
                        rts             #     5
```

4. Pravilno umestite v pomnilniški prostor izhodni napravi FB 16x16 in TTY

V address decoder sem dodal še signal datawrite, tako da sem naprave lahko razdelil na read/write, read only in write only. Katera naprava je aktivirana pa je odvisno od 14 in 15 bita naslova, ki je trenutno prisoten na naslovnem vodilu.



5. Opišite dogajanje v podatkovni enoti po vseh elementarnih korakih, ki bi bili potrebni za pravilno realizacijo klicev podprogramov in vrnitev iz podprogramov.

Klic podprogramov, je realiziran s pomočjo sklada, v tem primeru je kazalec na sklad v registru 7. V register 7 najprej inicializiramo začetni naslov sklada, ob tem se moremo zavedati, da je tu implementacija sklada padajoča (z dodajanjem na sklad se bo naslov kazalca zmanjševal). Ob klicu podprograma z ukazom **JSR *immed,*** *se najprej kazalec sklada zmanjša za 1, nato pa se na lokacijo na katero sedaj kaže SP shrani PC + 1 ter se izvede skok na takojšnji operand. Ob vrnitvi iz podprograma z klicom ukaza* ***RTS*** *se PC nastavi na vrednost* , ki je shranjena na naslovu na katerega kaže kazalec sklada, nato pa se kazalec poveča za 1.

Pri skokih v podprograme nam zelo prau pride tudi možnost shranjevanja trenutnih vrednosti registrov, tako da se po vrnitvi iz podprograma njihova vrednost ne spremeni. Za sharnejvanje registrov se uporabljata ukaza **PUSH Rd**, ki vrednost registra Rd porine na vrh sklad in **POP Rd**, ki z vrha sklada vzame vrednost in jo shrani v Rd. V sklopu dodatne naloge pa sem v prevajalnik dodal možnost uporabe ukazov **STM {Rn [- Rm]}** in **LDM {Rn [- Rm]}**, ki nam poenostavijo shranjevanje

vrednosti registrov, tako da za shranjevanje npr. štirih registrov napišemo samo 2 ukaz, prevajalnik pa ju prevede v ločene PUSH/POP ukaze.