ORGANIZACIJA RAČUNALNIKOV

Povzetki predavanj

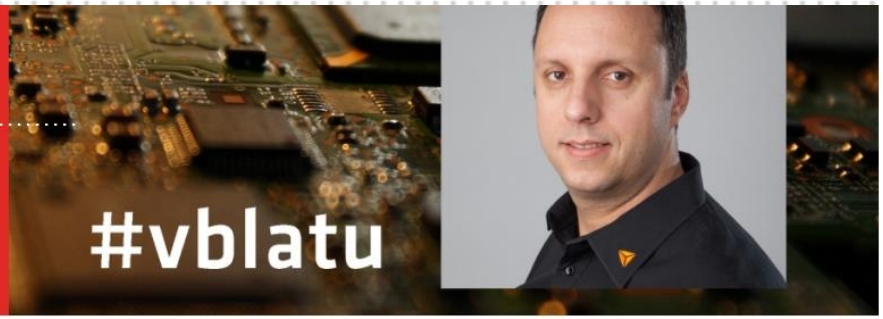
3. Mikroarhitekturni nivo računalnika

(3.2 MiMo Model Mikroprogramirane CPE)

Predavanje V blatu: Dejan Črnila - Code optimization on modern processors

NOVICA

objavljeno
25. oktober 2017



Za boljše rezultate se spustite na nižji nivo programiranja

Novice

Novice

Želite izboljšati delovanje svojih programov? Če je odgovor da, ne smete zamud optimizacije kode »Code optimization on modern processors« v okviru serije Če oktobra ob 18. uri na FRI.

Z Dejanom Črnilo, programskim inženirjem iz podjetja Dewesoft, ki se bo na predavanju skrivnosti optimiziranja kode, smo na kratko spregovorili o tem, zakaj in kdaj se lotiti op nas čaka na predavanju.

„… tudi 64-kratna pohitritev po optimizaciji kode…“

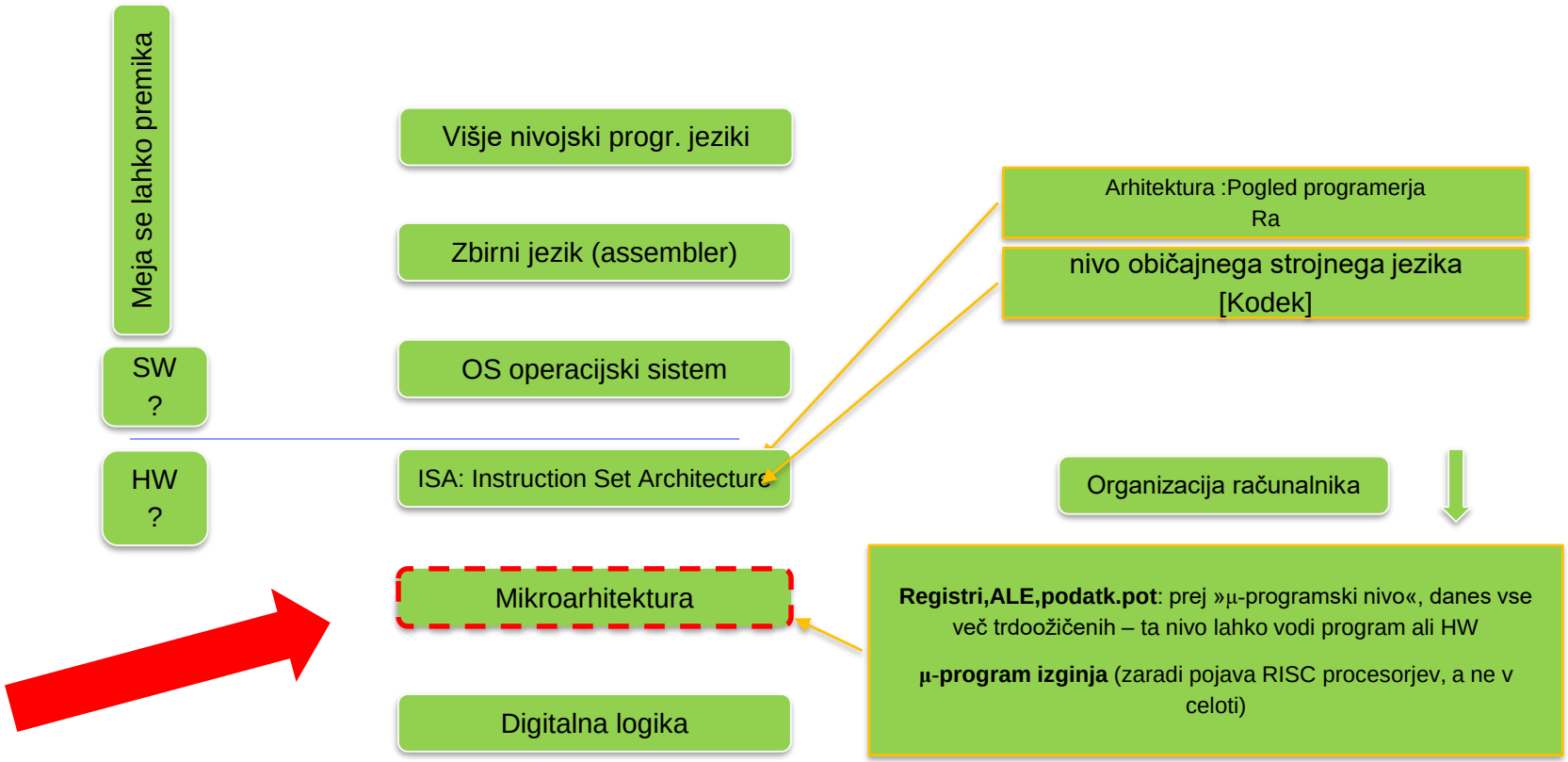
Space shuttle Atlantis launch monitoring with Dewesoft software

Namen in cilji 3. poglavja:

Razumevanje :

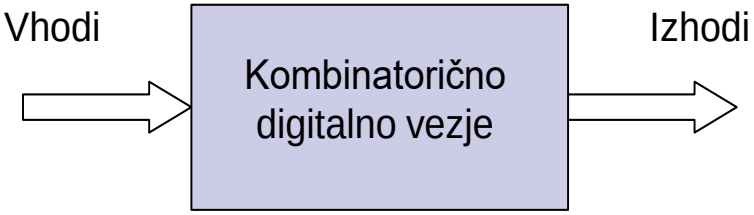
- pomembnost urinega signala, sinhronskosti v digitalnih vezjih
- delovanja CPE in njenih sestavnih delov skozi praktični primer:
  - od nivoja logičnih vrat do delujočega modela CPE
    - MiMo model mikroprogramske CPE
  - realizacija/izvedba strojnih ukazov z zaporedji mikro-ukazov
    - **razumevanje delovanja CPE** (ukazi, registri, enote, PC, …)
- ARM – pregled (mikro) arhitektur

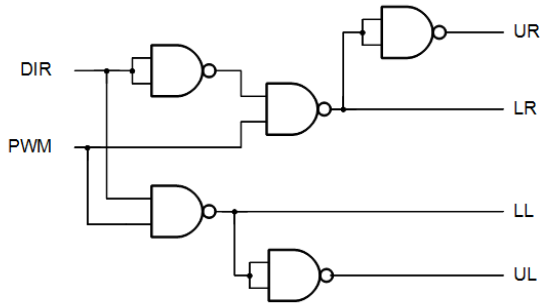
3. Mikro-arhitekturni nivo računalnika



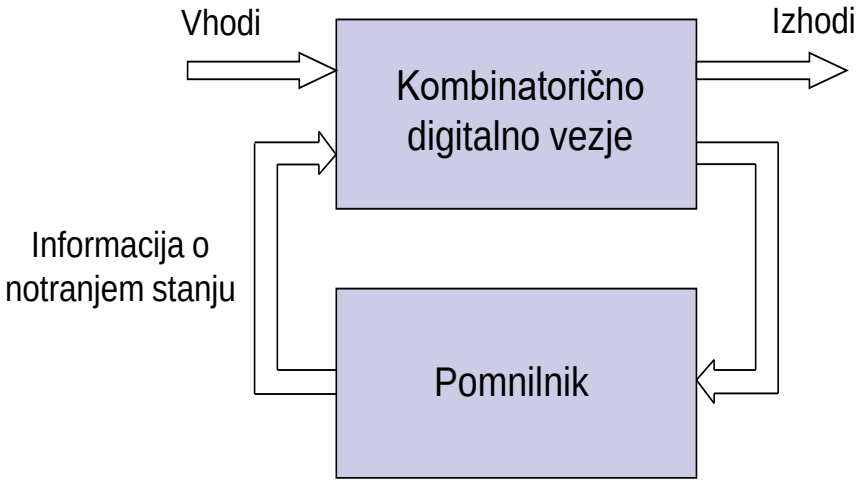
3. Mikroarhitekturni nivo računalnika

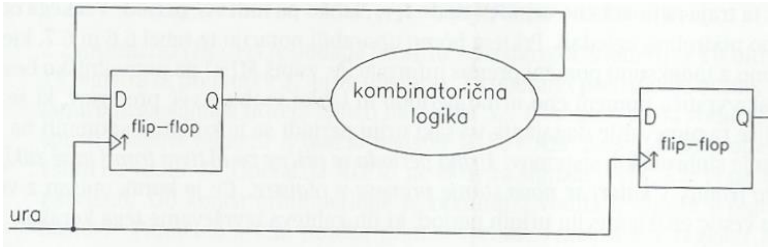
Kombinatorična logična vezja:



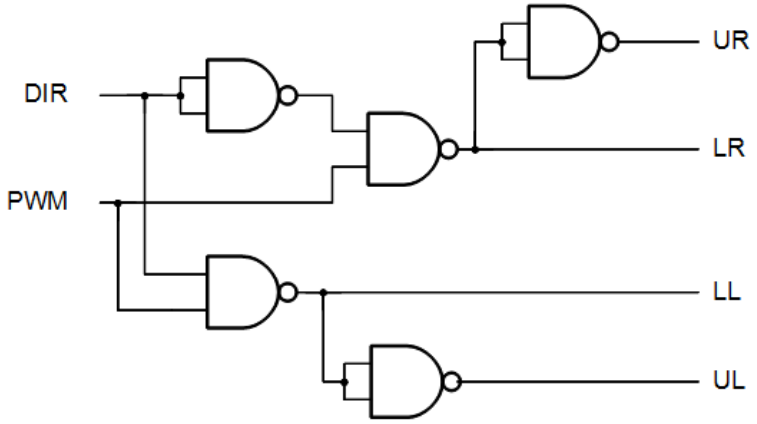


Sekvenčna logična vezja:





Asinhronska digitalna vezja :



Slabosti:

- različne zakasnitve
- zapleteno reševanje problema razl. zakasnitev

Prednosti:

- hitrost

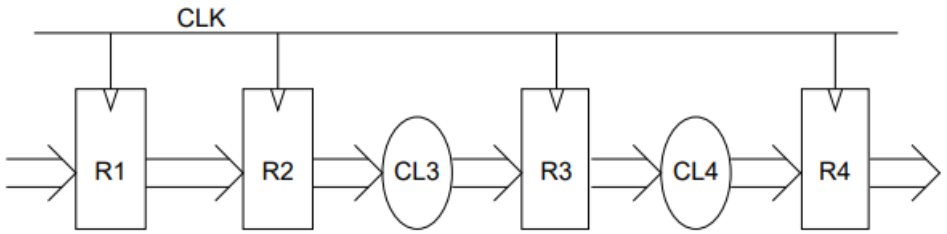
Asinhronska digitalna vezja :

Primer: MiniMIPS (asinhronska realizacija CPU MIPS R3000)

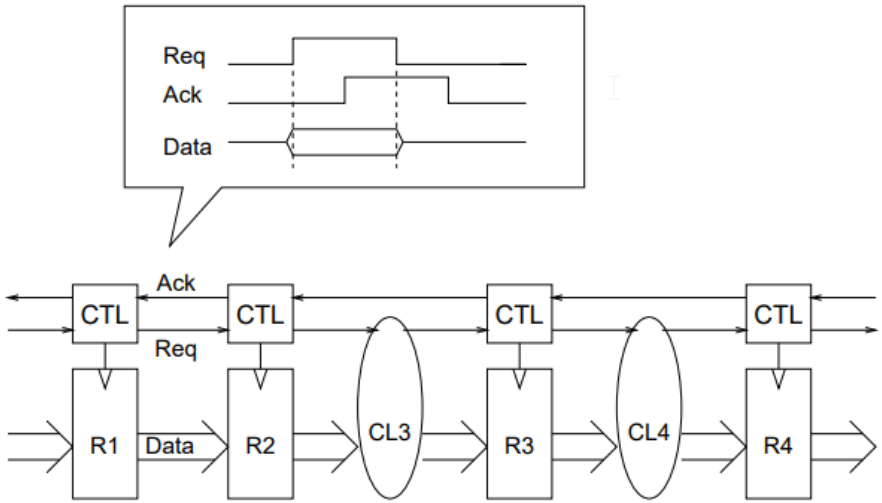
- 4-krat hitrejša od sinhronske realizacije
- bistveno bolj kompleksna (težko na kompleksnejših CPE)

Prikaz razlik med sinhronsko in asinhronsko

- Sinhronska



- Asinhronska



3. Mikroarhitekturni nivo računalnika

Asynchronous circuit design

A Tutorial

Jens Sparsø

Technical University of Denmark

Asinhronska digitalna vezja :

- Low power consumption, [102, 104, 32, 35, 73, 76]
  - . . . due to fine-grain clock gating and zero standby power consumption.
- High operating speed, [119, 120, 63]
  - . . . operating speed is determined by actual local latencies rather than global worst-case latency.
- Less emission of electro-magnetic noise, [102, 83]
  - . . . the local clocks tend to tick at random points in time.
- Robustness towards variations in supply voltage, temperature, and fabrication process parameters, [62, 72, 74]
  - . . . timing is based on matched delays (and can even be insensitive to circuit and wire delays).
- Better composability and modularity, [67, 57, 108, 97, 94]
  - . . . because of the simple handshake interfaces and the local timing.
- No clock distribution and clock skew problems,
  - . . . there is no global signal that needs to be distributed with minimal phase skew across the circuit.

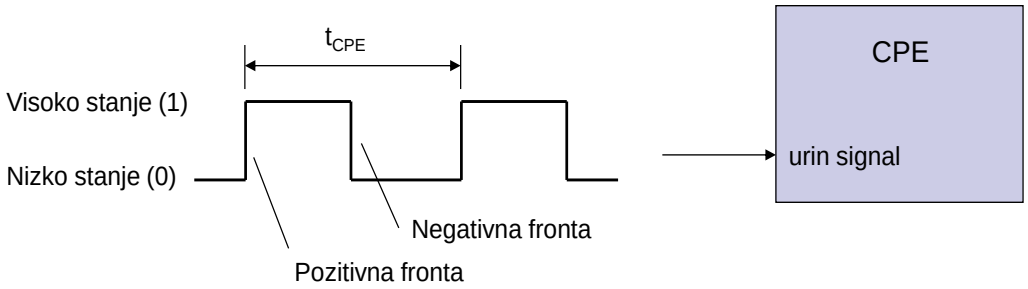
3.1 Sinhronska digitalna vezja

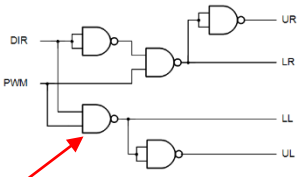
3.1.1 Urin signal:

Odvisen :

- Hitrosti vezij
- Zakasnitev v povezavah
- Števila vezij

Aktivna fronta (poz. ali neg.) sproži spremembo notr. stanja

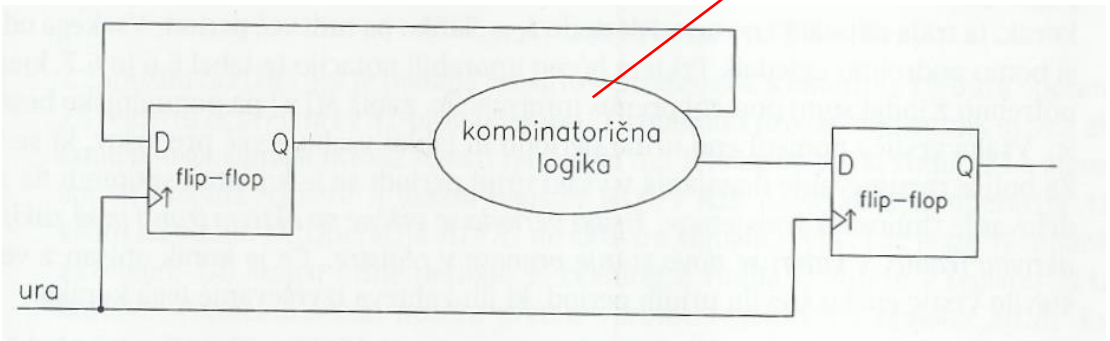


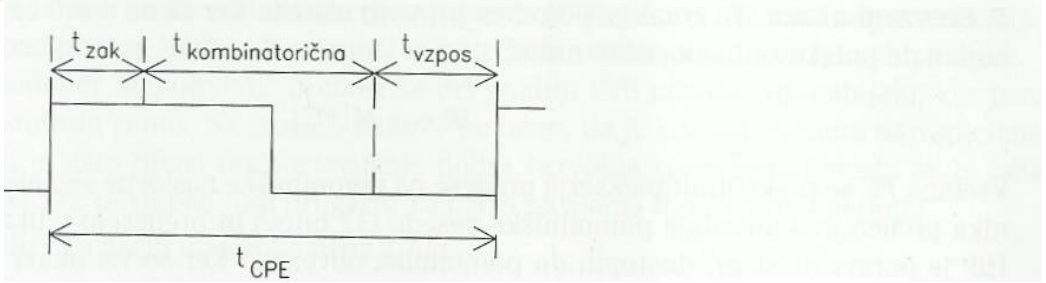


3.1.2 Sprememba stanja:

- **t_{zak}:** za spremembo FF
- **t_{komb}**: kombinacijsko vezje
- **t_{vzpos}:** stabilnosti vhodov v FF

$t_{CPE} > t_{zak} + t_{komb} + t_{vzpos}$





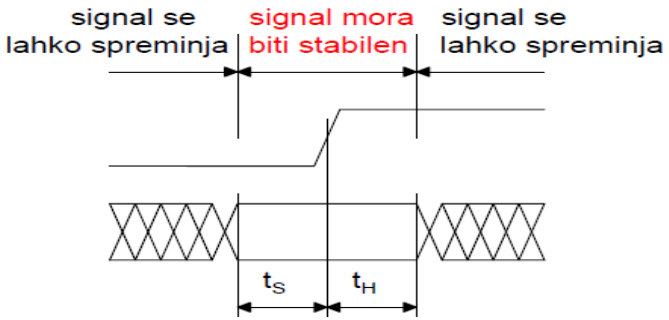
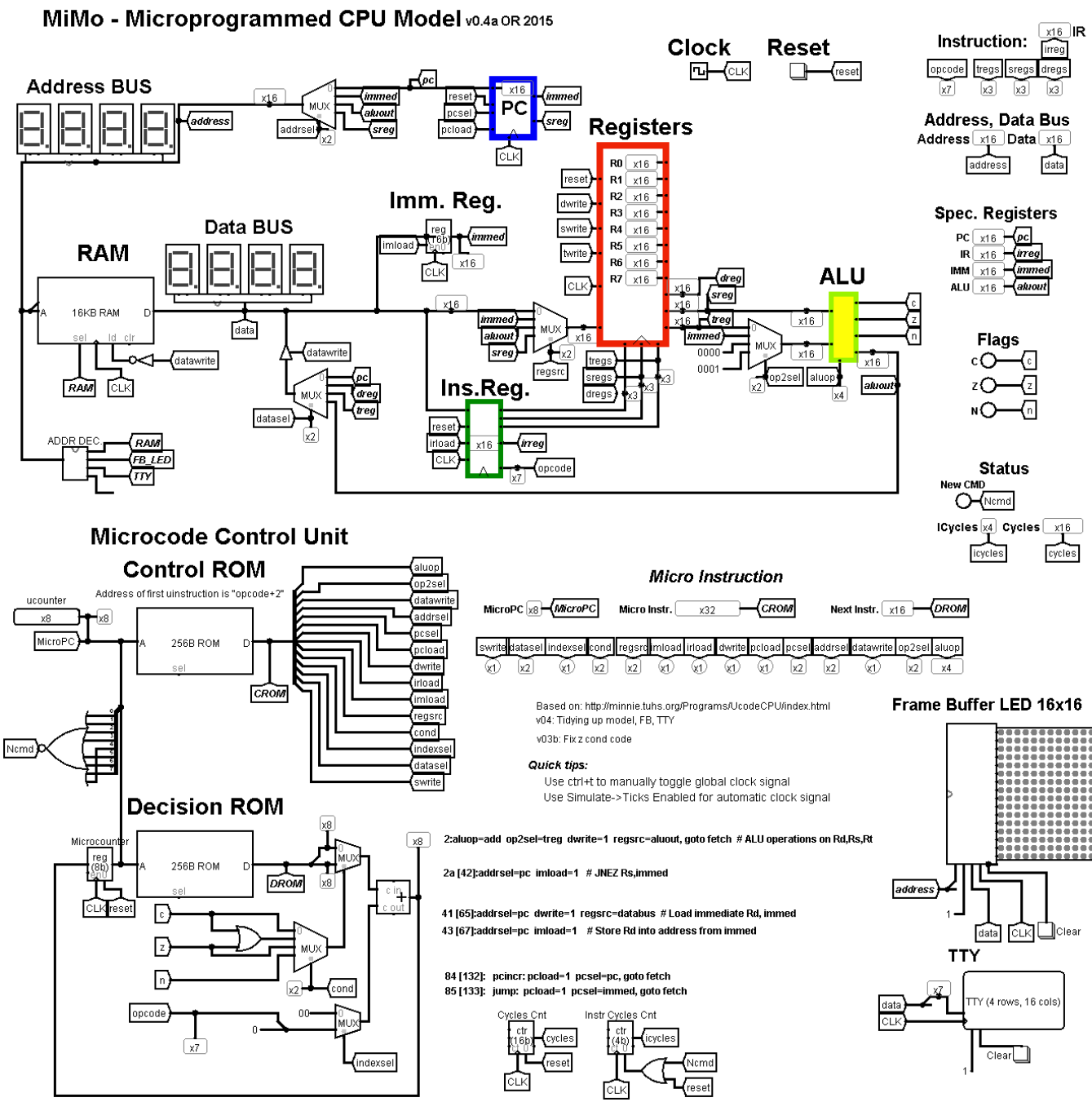


Figure 1.27: Vzpostavitveni (t_S) in držalni čas (t_H).

3.2 MiMo – Mikroprogramiran Model CPE



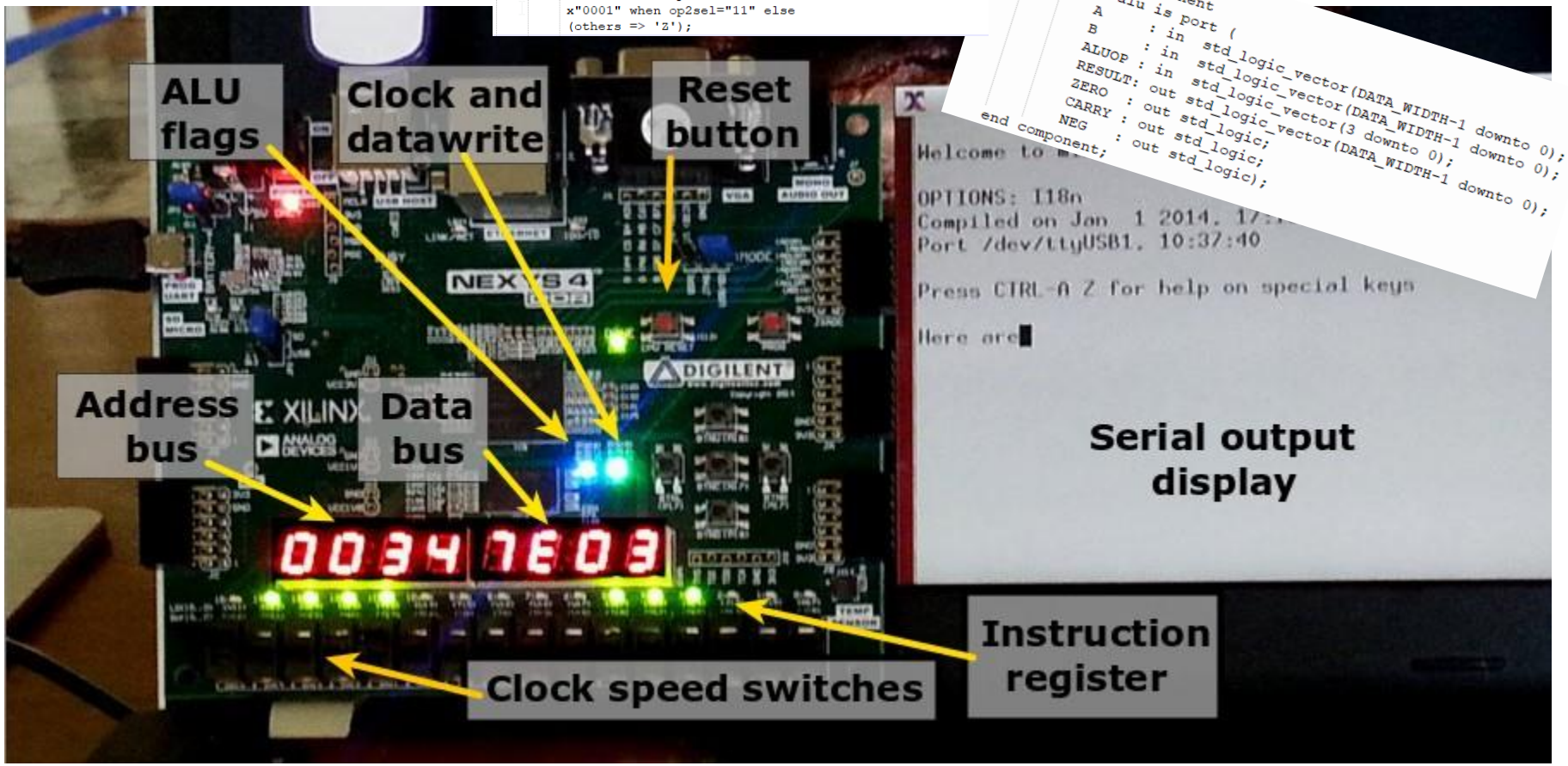
3.2 MiMo – Mikro-programiran Model CPE

FPGA realizacija

```
-- The ALU
alunit: alu port map (
    A => alu_a,
    B => alu_b,
    ALUOP => aluop,
    RESULT=> aluout,
    ZERO => zero,
    CARRY => carry,
    NEG => neg
);
```

```
-- Multiplexor into the second ALU input, and first ALU input
alu_a <= sreg;
alu_b <= treg when op2sel="00" else
         immed_out when op2sel="01" else
         x"0000" when op2sel="10" else
         x"0001" when op2sel="11" else
         (others => 'Z');
```

```
VHDL:
    -- The ALU component
    component alu is port (
        A      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
        B      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
        ALUOP  : in  std_logic_vector(3 downto 0);
        RESULT: out std_logic_vector(DATA_WIDTH-1 downto 0);
        ZERO   : out std_logic;
        CARRY  : out std_logic;
        NEG    : out std_logic);
    end component;
```



https://minnie.tuhs.org/Programs/UcodeCPU/nexys4_install.html

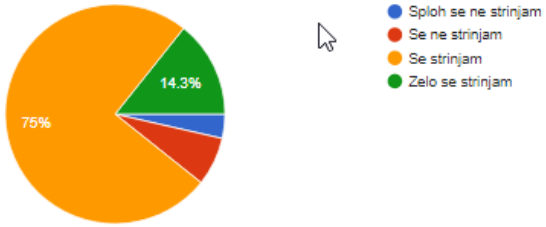
3.2 MiMo – Mikro-programiran Model CPE

Mnenja

Osebno sem potreboval kar nekaj časa, da sem razumel, kako točno deluje mikrozbirnik oz. posamezna kontrola vrstica. Ko sem enkrat povezal stvari mi pa niso več predstavljale problema …

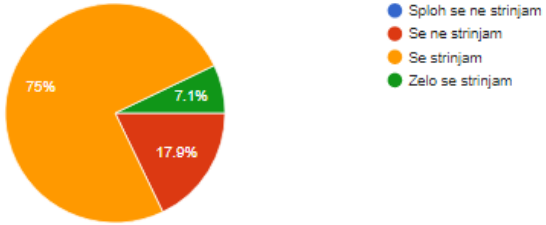
Izvedba MiMo modela na nivoju logičnih vrat v Logisimu je bila zanimiva in koristna

28 responses



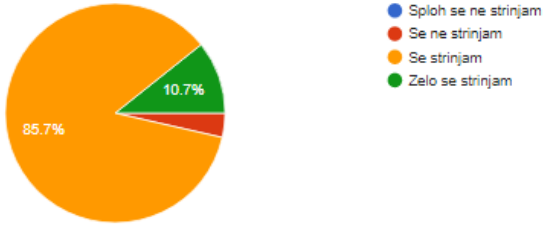
Izvedba MiMo modela na nivoju mikro zbirnika (micro-assembler) je bila zanimiva in koristna

28 responses



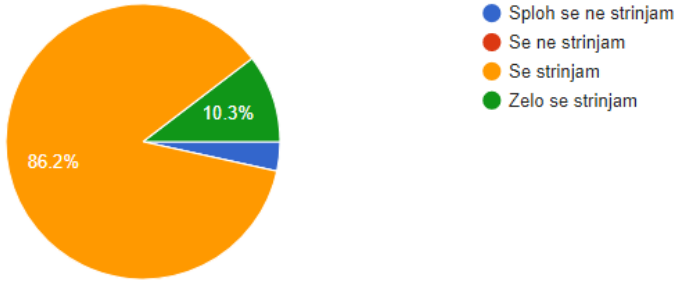
Izvedba MiMo modela na nivoju zbirnika (assembler) je bila zanimiva in koristna

28 responses



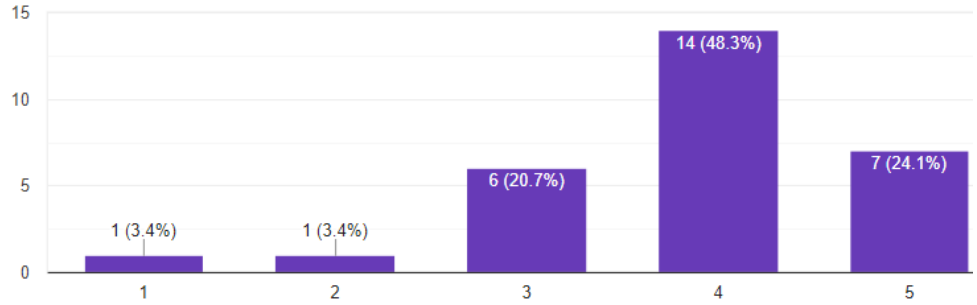
Model MiMo gledano v celoti je pripomogel k izboljšanju mojega poklicnega (strokovnega) znanja.

29 responses



Splošna ocena koristnosti MiMo modela pri predmetu - od 1 do 5 (najvišja ocena)

29 responses



3.2 MiMo – Mikroprogramiran Model CPE

Značilnosti :

- pomnilniška beseda 16 bitov
- pomnilniški naslov 16 bitov
- dolžina ukazov 16 ali 32 bitov (2 formata)
  - Format 1 : (primer **ADD R1,R2,R3** # R1<-R2+R3, R1=Dreg, R2=Sreg, R3=Treg)

Op.koda	Treg	Sreg	Dreg
7	3	3	3

  - Format 2 : (primer **LI R1, 100** # R1<-100)

Op.koda	Treg	Sreg	Dreg
7	3	3	3
16 bitni tak. operand			
16			

ADD R1,R2,R3 # R1<-R2+R3
R3: Treg: obič. 2 op. za ALE
R2: Sreg: 1.operand za ALE
R1: Dreg: ponor ALE operacije
POZOR – v formatu ukaza **obrnjen vrstni red** kot v mnemoniku**!**

- registri:
  - 8x 16bitnih splošno namenskih registrov R0-R7
- operandi (pomnilniški dostopi) so samo 16-bitni
- pomnilniško preslikan vhod/izhod

MiMo temelji na tem viru: http://minnie.tuhs.org/Programs/UcodeCPU/index.html

3.2.1 Izvrševanje ukazov – MiMo

Delovanje CPE:

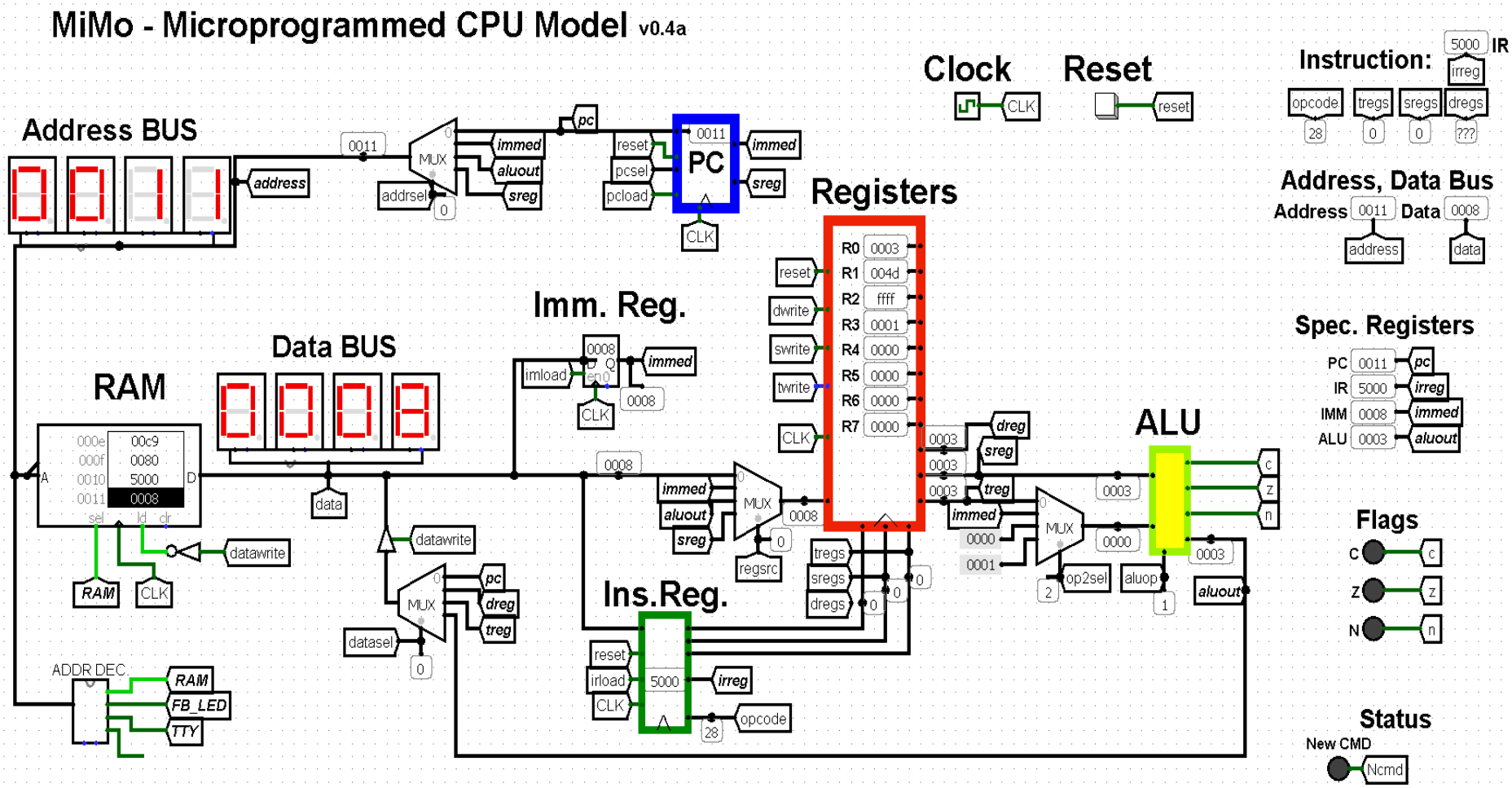
- **branje ukaza iz pomnilnika - FETCH**
  - „ukazno prevzemni cikel“
- **izvrševanje ukaza - EXECUTION**
  - „izvršilni cikel“

Elementarni koraki (večperiodna realizacija) :

- branje ukaza iz pomnilnika
- dekodiranje (analiza) ukaza
- prenos operandov v CPE (pomnilnik, takojšnji operandi)
- izvedba operacije (ALE)
- shranjevanje rezultata (registri)

- obnovitev PC (kaže na naslov nasl. ukaza)

MiMo – Podatkovna enota



MiMo – Podatkovna enota
Prikaz delovanja ob izvrševanju ukaza
jnez r1,loop #Jump to loop: if r1!=0

Primer programa v zbirniku :

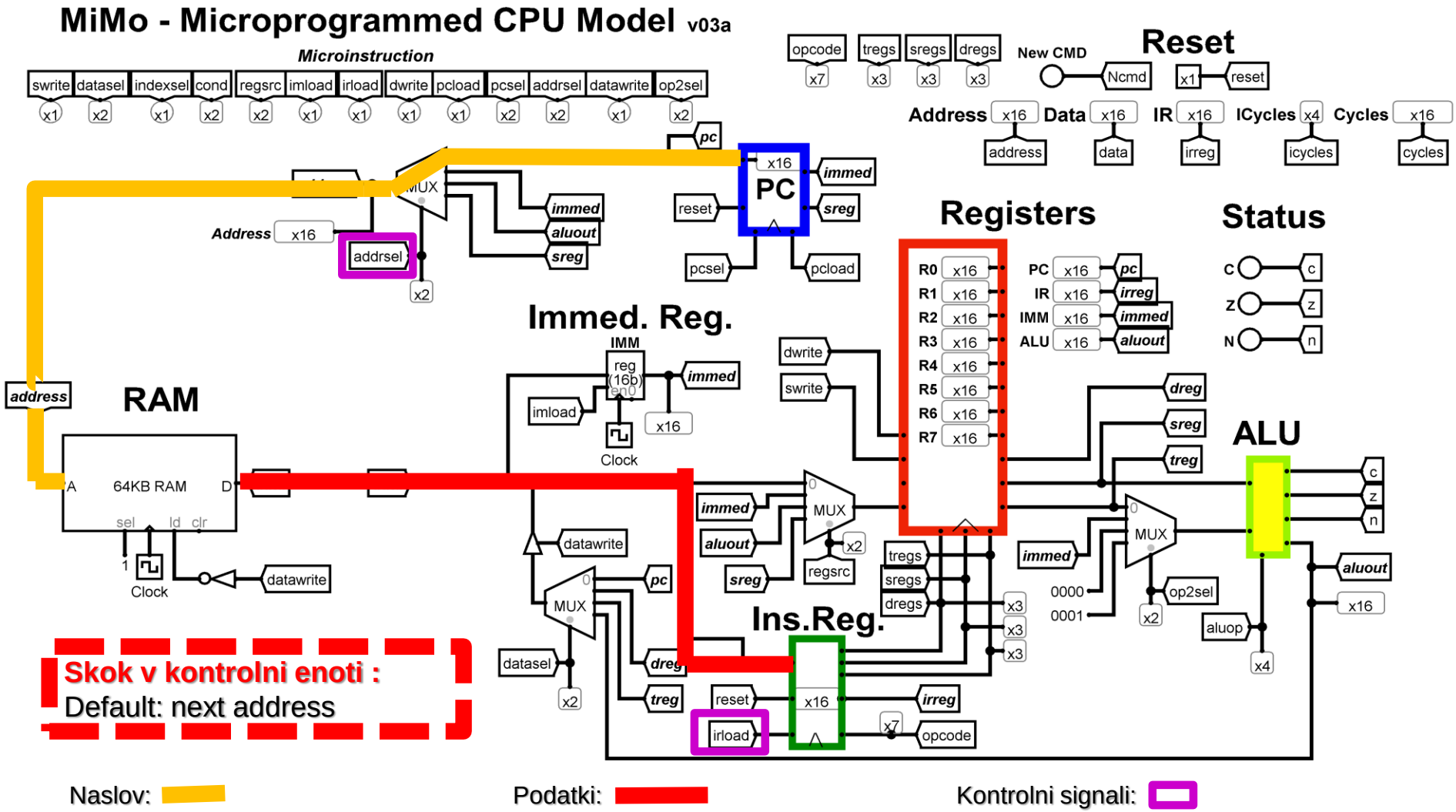
```
main: li    r0, 0          # r0 is the running sum
      li    r1, 100        # r1 is the counter
      li    r2, -1         # Used to decrement r1
loop: add   r0, r0, r1     # r0= r0 + r1
      add   r1, r1, r2     # r1--
      jnez  r1, loop       # loop if r1 != 0
      sw    r0, 256        # Save the result
```

JNEZ Rs,immed:

jnez Rs,immed (40)

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

```
fetch:    addrsel=pc irload=1                                  # Address=PC, Load IR register
          pcload=1  pcsel=pc, opcode_jump                      # PC=PC+1, jump to 2+OPC
40:       addrsel=pc  imload=1                                 # Read Immediate operand -> IMRegister
          aluop=sub  op2sel=const0, if z then pcincr else jump # ALU: Rs-0, If z then pcincr else jump
pcincr:   pcload=1  pcsel=pc, goto fetch                       # Increment PC and goto new command;
jump:     pcload=1  pcsel=immed, goto fetch                    # Set address to immed and goto new command
```

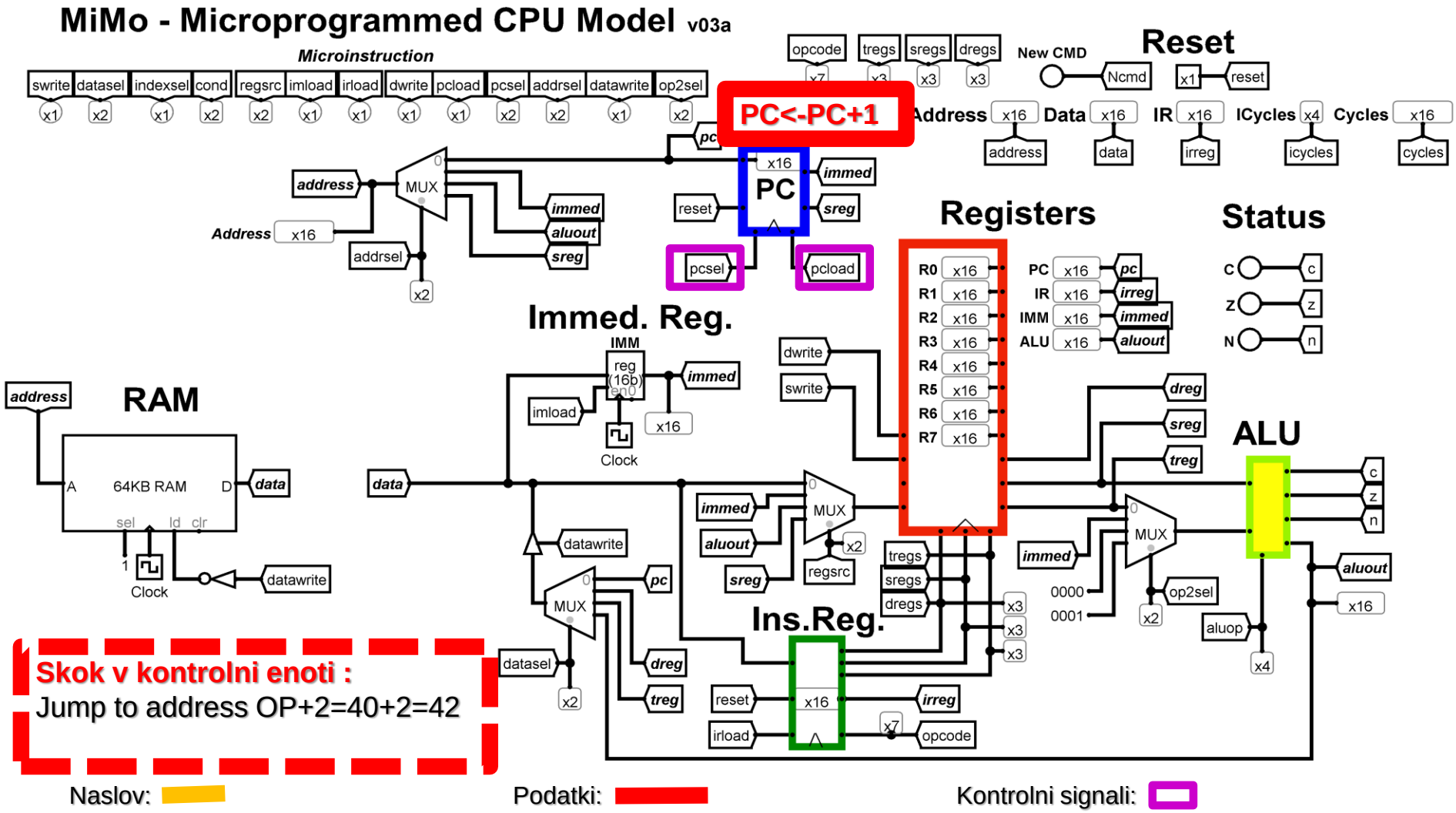


JNEZ Rs,immed:

jnez Rs,immed (40)

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

```
fetch:    addrsel=pc irload=1                                  # Address=PC, Load IR register
          pcload=1  pcsel=pc, opcode_jump                      # PC=PC+1, jump to 2+OPC
40:       addrsel=pc  imload=1                                 # Read Immediate operand -> IMRegister
          aluop=sub  op2sel=const0, if z then pcincr else jump # ALU: Rs-0, If z then pcincr else jump
pcincr:   pcload=1  pcsel=pc, goto fetch                       # Increment PC and goto new command;
jump:     pcload=1  pcsel=immed, goto fetch                    # Set address to immed and goto new command
```

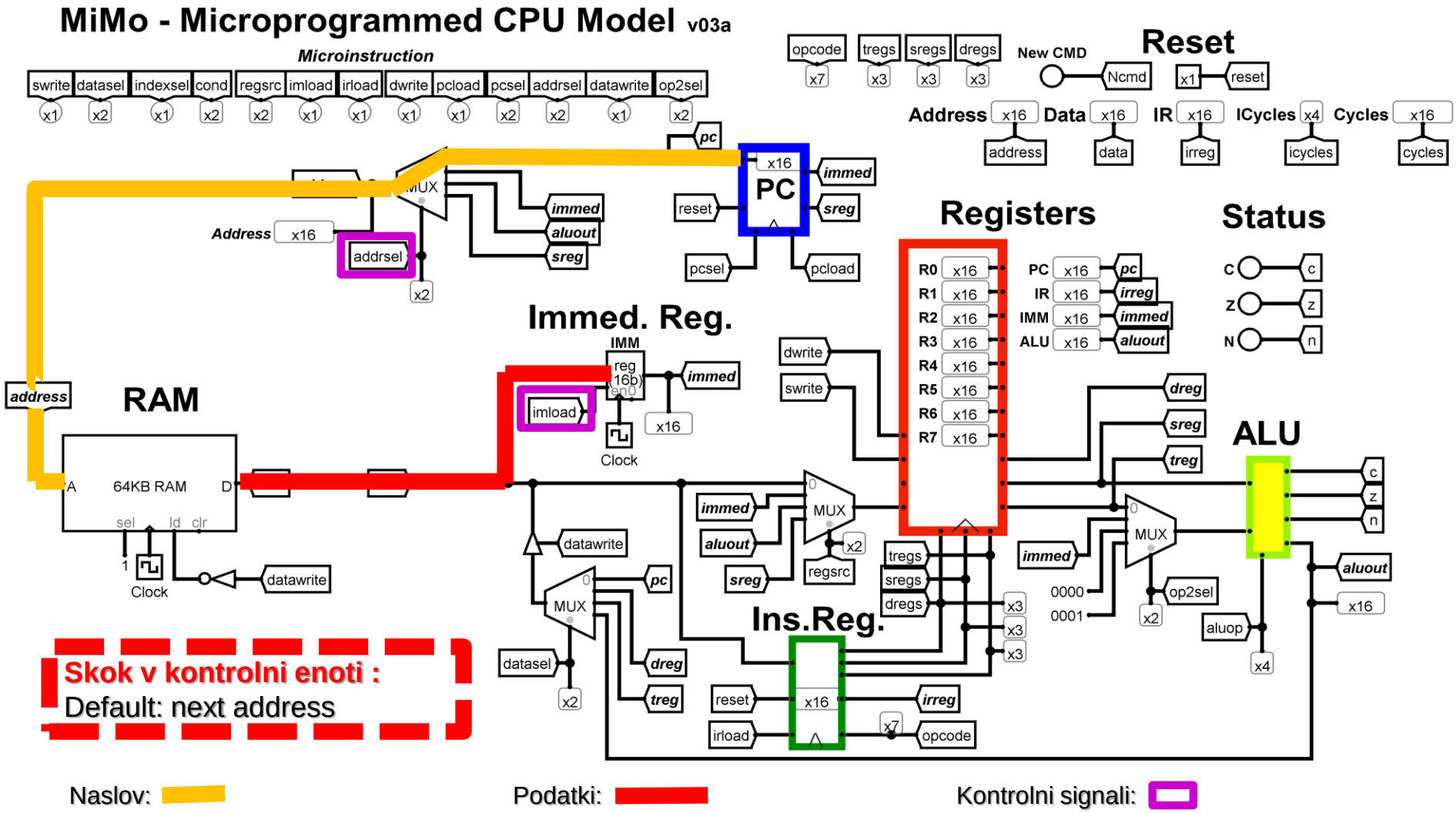


JNEZ Rs,immed:

jnez Rs,immed (40)

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

```
fetch:    addrsel=pc irload=1                                  # Address=PC, Load IR register
          pcload=1  pcsel=pc, opcode_jump                      # PC=PC+1, jump to 2+OPC
40:       addrsel=pc  imload=1                                 # Read Immediate operand -> IMRegister
          aluop=sub  op2sel=const0, if z then pcincr else jump # ALU: Rs-0, If z then pcincr else jump
pcincr:   pcload=1  pcsel=pc, goto fetch                       # Increment PC and goto new command;
jump:     pcload=1  pcsel=immed, goto fetch                    # Set address to immed and goto new command
```

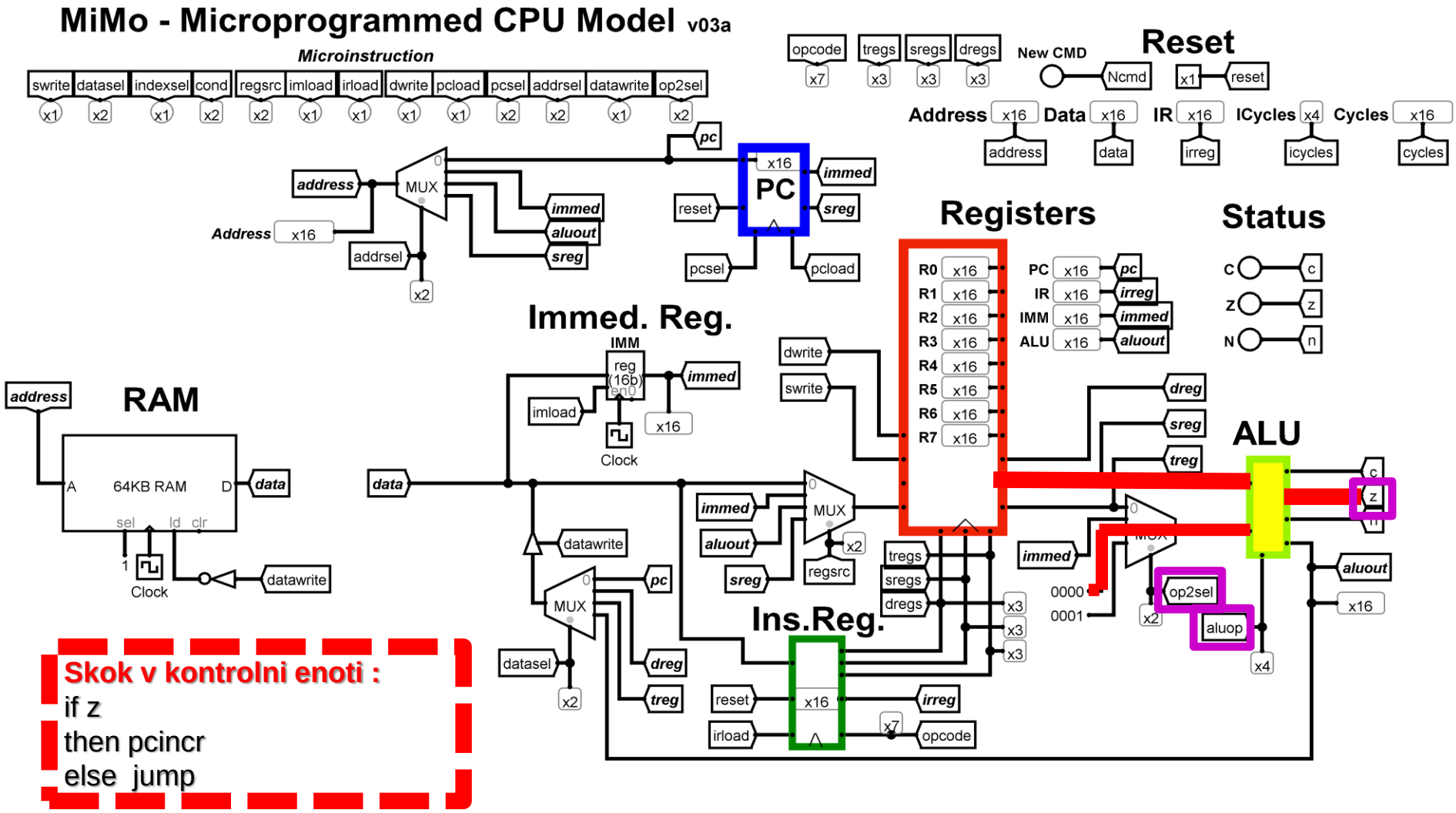


JNEZ Rs,immed:

jnez Rs,immed (40)

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

```
fetch:    addrsel=pc irload=1                                  # Address=PC, Load IR register
          pcload=1  pcsel=pc, opcode_jump                      # PC=PC+1, jump to 2+OPC
40:       addrsel=pc  imload=1                                 # Read Immediate operand -> IMRegister
          aluop=sub  op2sel=const0, if z then pcincr else jump # ALU: Rs-0, If z then pcincr else jump
pcincr:   pcload=1  pcsel=pc, goto fetch                       # Increment PC and goto new command;
jump:     pcload=1  pcsel=immed, goto fetch                    # Set address to immed and goto new command
```



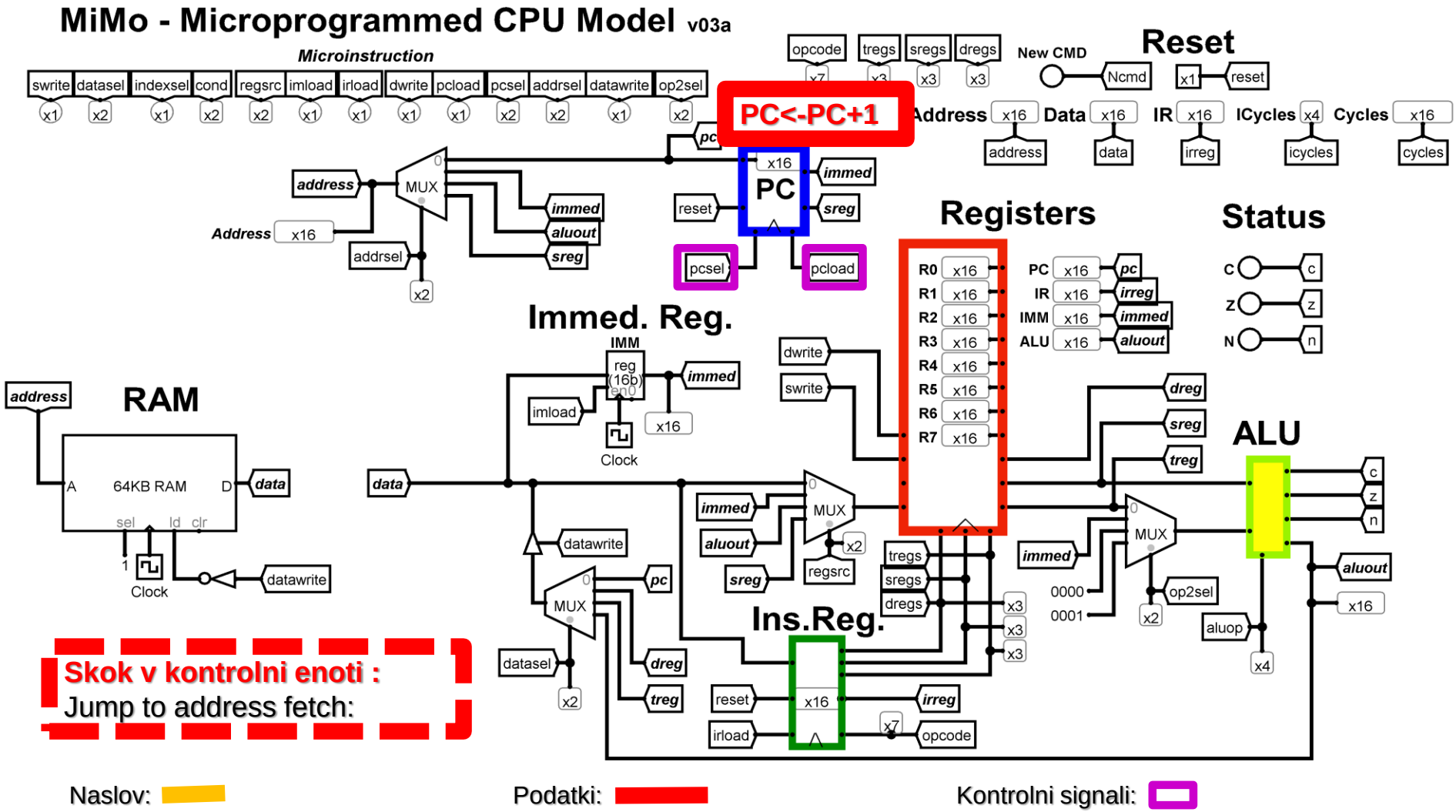
Skok v kontrolni enoti :
if z
then pcincr
else jump

JNEZ Rs,immed: velja Rs=0

jnez Rs,immed (40)

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

```
fetch:    addrsel=pc irload=1                                  # Address=PC, Load IR register
          pcload=1  pcsel=pc, opcode_jump                      # PC=PC+1, jump to 2+OPC
40:       addrsel=pc  imload=1                                 # Read Immediate operand -> IMRegister
          aluop=sub  op2sel=const0, if z then pcincr else jump # ALU: Rs-0, If z then pcincr else jump
pcincr:   pcload=1  pcsel=pc, goto fetch                       # Increment PC and goto new command;
jump:     pcload=1  pcsel=immed, goto fetch                    # Set address to immed and goto new command
```

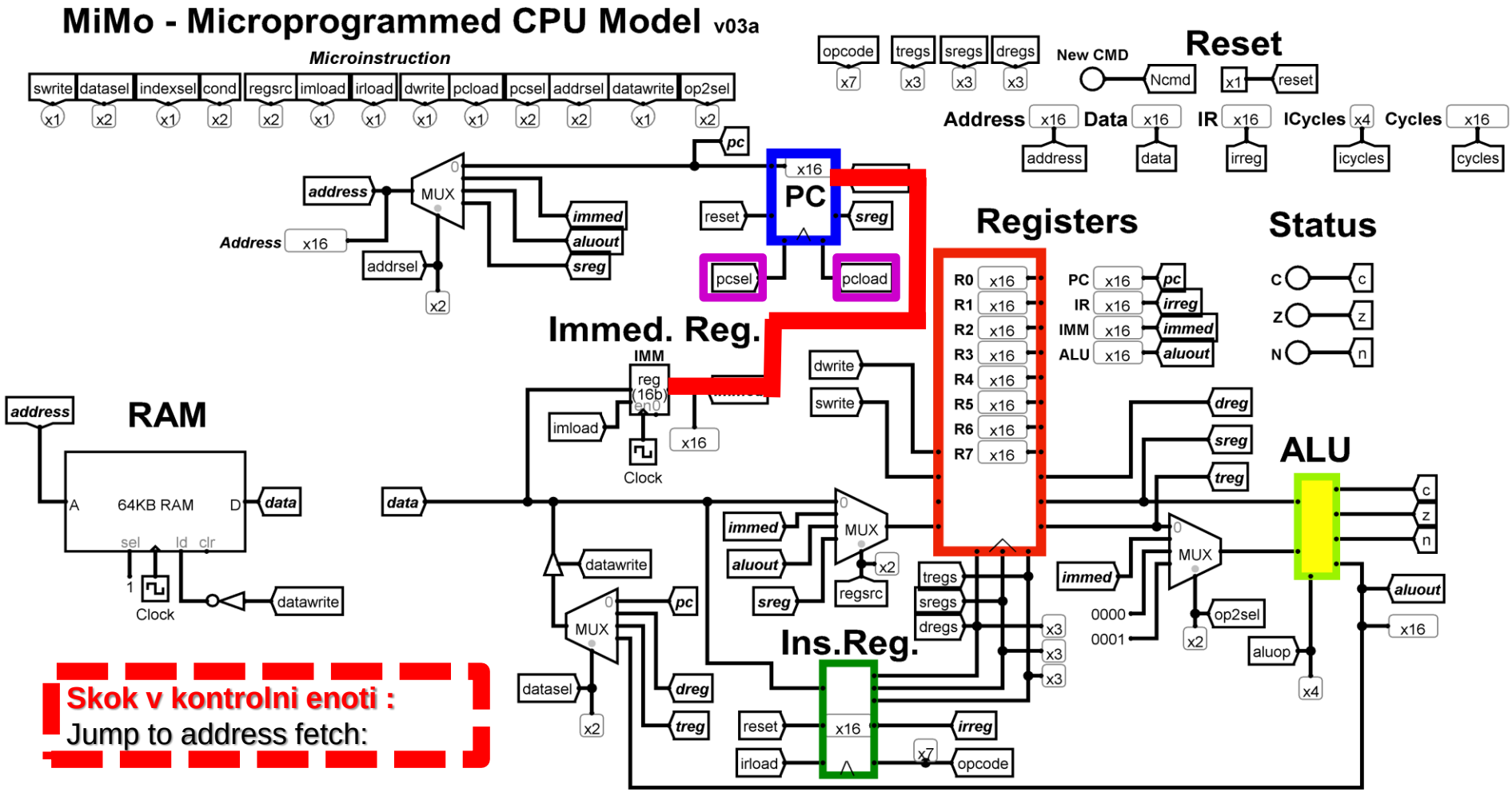


JNEZ Rs,immed: velja Rs≠0

jnez Rs,immed (40)

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

```
fetch:    addrsel=pc irload=1                                  # Address=PC, Load IR register
          pcload=1  pcsel=pc, opcode_jump                      # PC=PC+1, jump to 2+OPC
40:       addrsel=pc  imload=1                                 # Read Immediate operand -> IMRegister
          aluop=sub  op2sel=const0, if z then pcincr else jump # ALU: Rs-0, If z then pcincr else jump
pcincr:   pcload=1  pcsel=pc, goto fetch                       # Increment PC and goto new command;
jump:     pcload=1  pcsel=immed, goto fetch                    # Set address to immed and goto new command
```



3.2.2. MiMo – podatkovna enota

3.2.2.1 ALE:

Vhodi:

2x 16-bitna operanda:

- Sreg,
- izhod iz MUX-a (op2sel)

Izhodi:

- 16-bitni rezultat operacije **(»aluout«)**
- zastavice **C** (+,-), **Z** (NOR), **N** (b_{15})

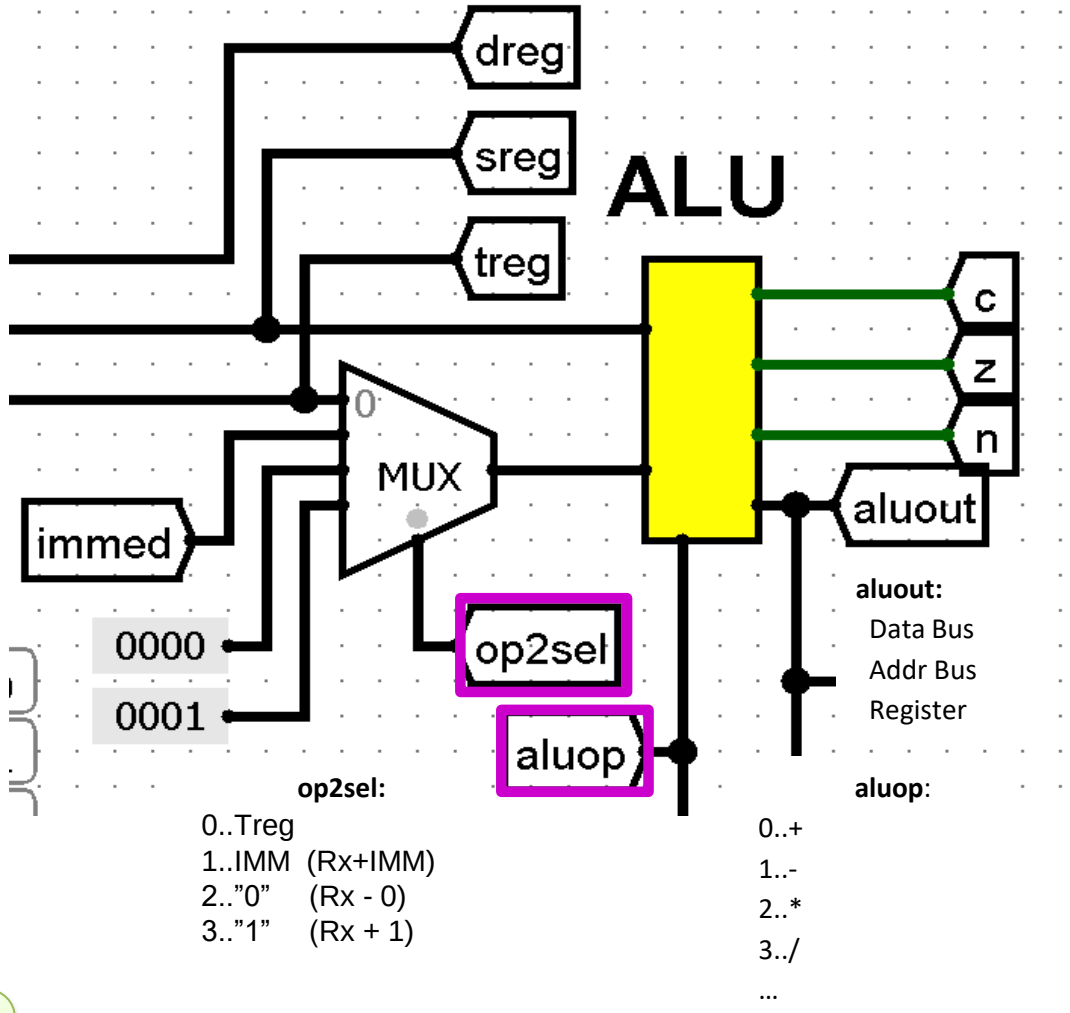
Kontrolni signali:

op2sel – določi 2. operand:

- 0..?
- 1..?
- 2..?
- 3..?

aluop – določi operacijo (iz OP kode)

- 0..?
- 1..?
- 2..?
- 3..?
- …



Označevanje :

- VHODI
- IZHODI
- KONTROLNI SIGNALI

3.2.2.1 ALE:

Vhodi:

2x 16-bitna operanda:

- Sreg,
- izhod iz MUX-a (op2sel)

Izhodi:

- 16-bitni rezultat operacije **(»aluout«)**
- zastavice **C** (+,-), **Z** (NOR), **N** (b_{15})

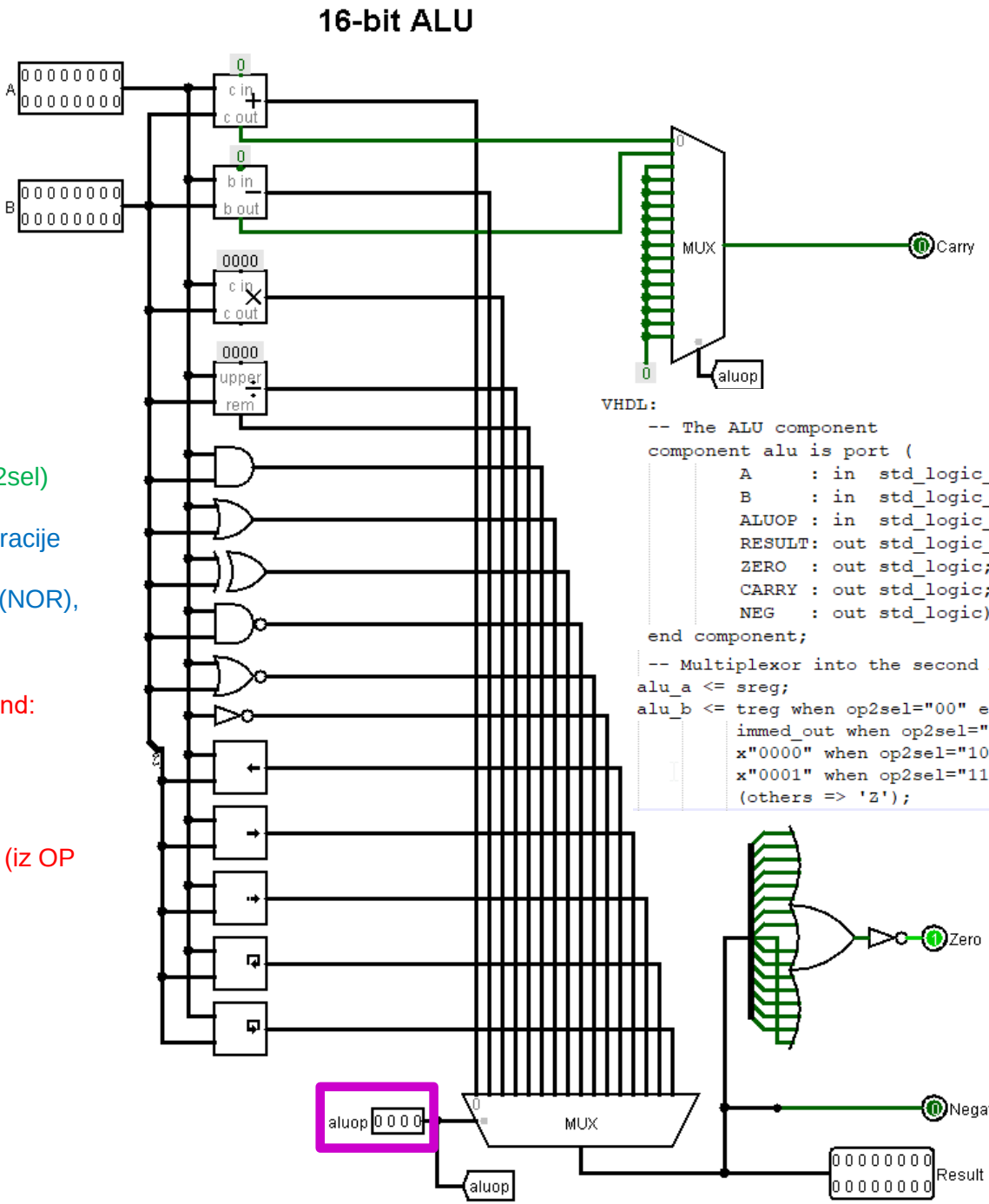
Kontrolni signali:

op2sel – določi 2. operand:

- 0..Treg
- 1..IMM (Rx+IMM)
- 2..”0” (Rx - 0)
- 3..”1” (Rx + 1)

aluop – določi operacijo (iz OP kode)

- 0..+
- 1..-
- 2..*
- 3../
- …




```
  -- The ALU
alunit: alu port map (
    A => alu_a,
    B => alu_b,
    ALUOP => aluop,
    RESULT=> aluout,
    ZERO => zero,
    CARRY => carry,
    NEG => neg
  );
```

VHDL:

```
-- The ALU component
component alu is port (
        A      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
        B      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
        ALUOP  : in  std_logic_vector(3 downto 0);
        RESULT: out std_logic_vector(DATA_WIDTH-1 downto 0);
        ZERO   : out std_logic;
        CARRY  : out std_logic;
        NEG    : out std_logic);
end component;
-- Multiplexor into the second ALU input, and first ALU input
alu_a <= sreg;
alu_b <= treg when op2sel="00" else
        immed_out when op2sel="01" else
        x"0000" when op2sel="10" else
        x"0001" when op2sel="11" else
        (others => 'Z');
```

3.2.2. MiMo – podatkovna enota

3.2.2.2 Registri

Vhodi:

16-bitni vhod (»regval«)

Izhodi:

3x 16-bitni izhodi

Dreg,Sreg,Treg

Kontrolni signali:

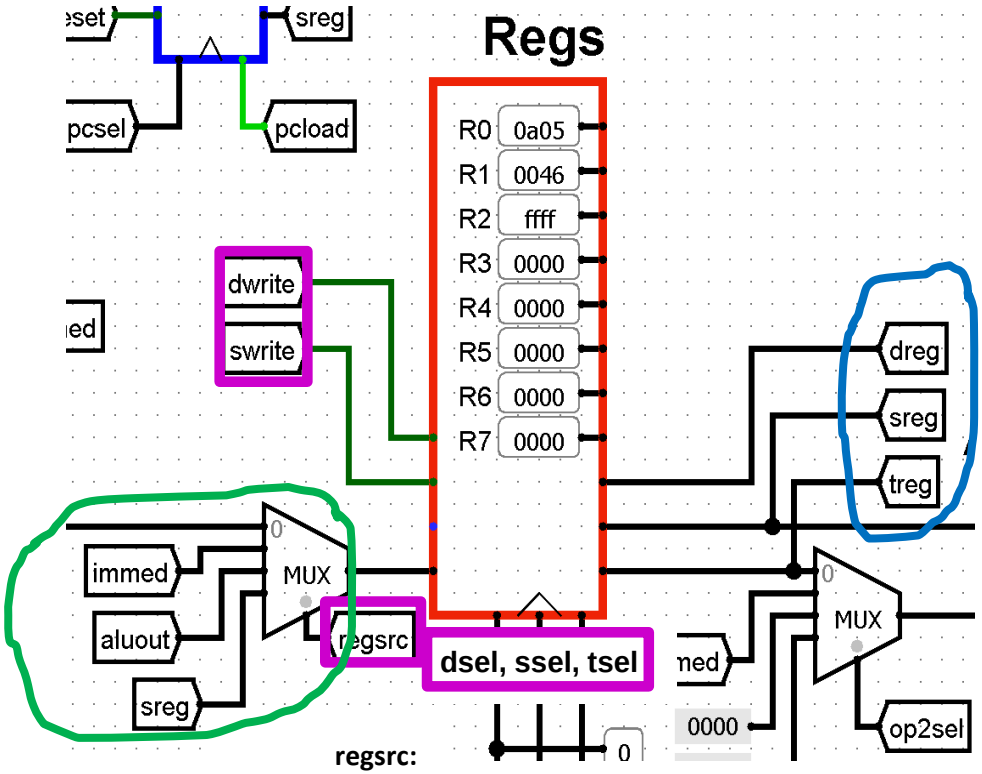
dsel, ssel, tsel – v ukazu: določajo kateri register bo na vsakem od 3 izhodov:

- Sreg ->ALU,vh.Regs,Nasl.vodilo
- Treg -> ALU,DataBus
- Dreg -> DataBus

regsrc – določa vhodni izvor:

- 0..DataBus
- 1..IMM
- 2..ALU
- 3..Sreg

dwrite, swrite, twrite (običajno samo eden) – določajo pisalno operacijo iz vhoda v izbrane registre



regsrc:

0..DBus
1..IMM
2..ALU
3..Sreg

3.2.2.2 Registri

Vhodi:

16-bitni vhod (»**regval**«)

Izhodi:

3x 16-bitni izhodi **Dreg,Sreg,Treg**

Kontrolni signali:

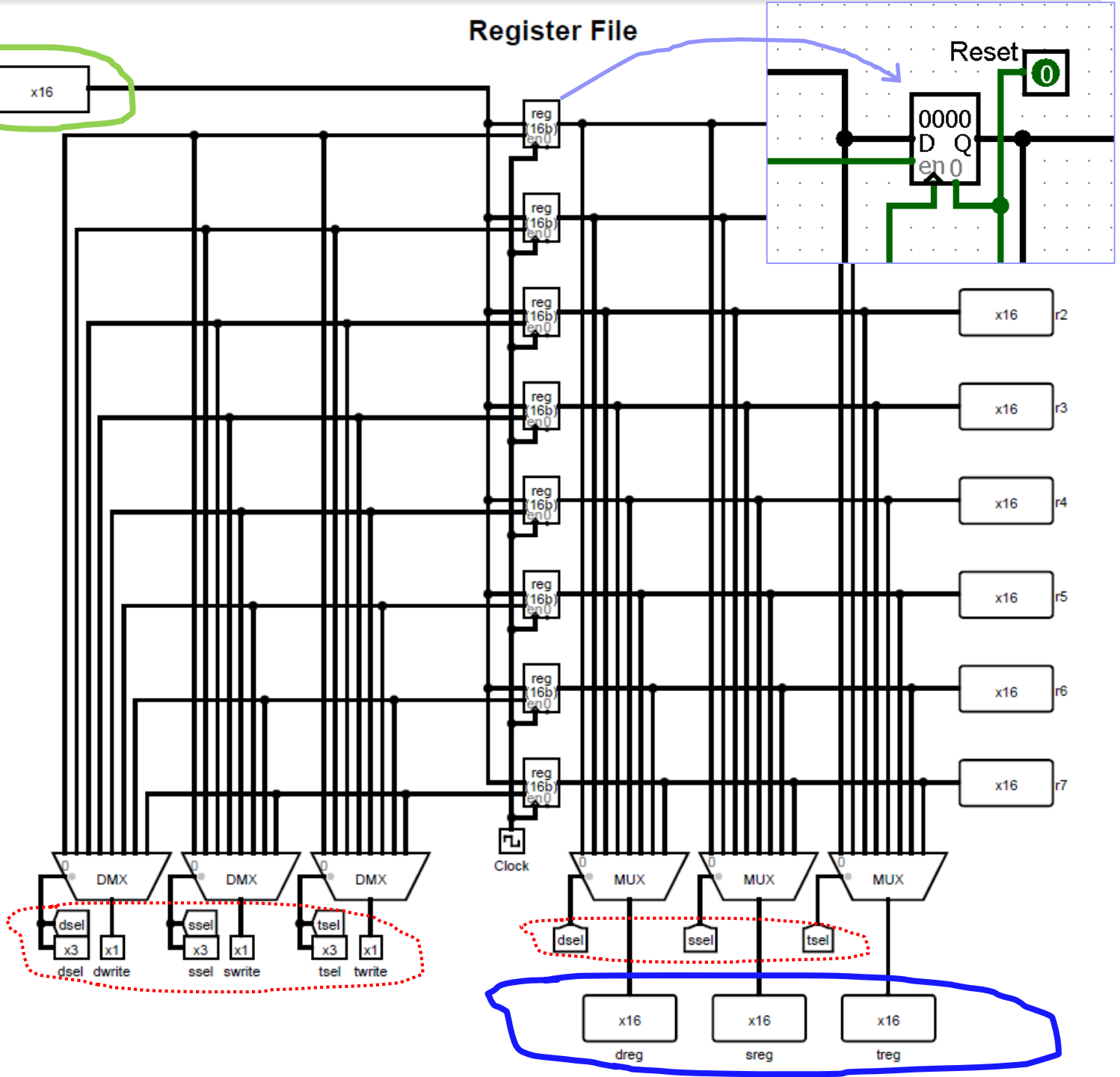
dsel, ssel, tsel – v ukazu: določajo kateri register bo na vsakem od 3 izhodov:

- Sreg ->ALU,vh.Regs,Nasl.vodilo
- Treg -> ALU,DataBus
- Dreg -> DataBus

regsrc – določa vhodni izvor:

- 0..DataBus
- 1..IMM
- 2..ALU
- 3..Sreg

dwrite, swrite, twrite (običajno samo eden) – določajo pisalno operacijo iz vhoda v izbrane registre

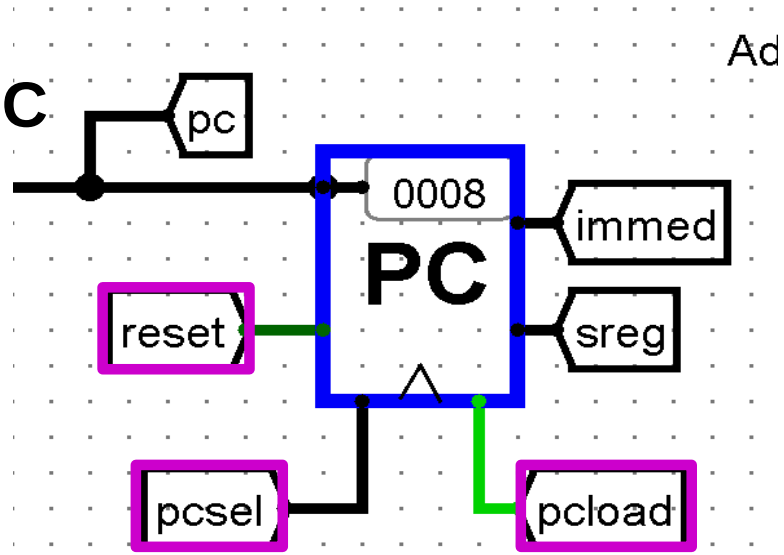


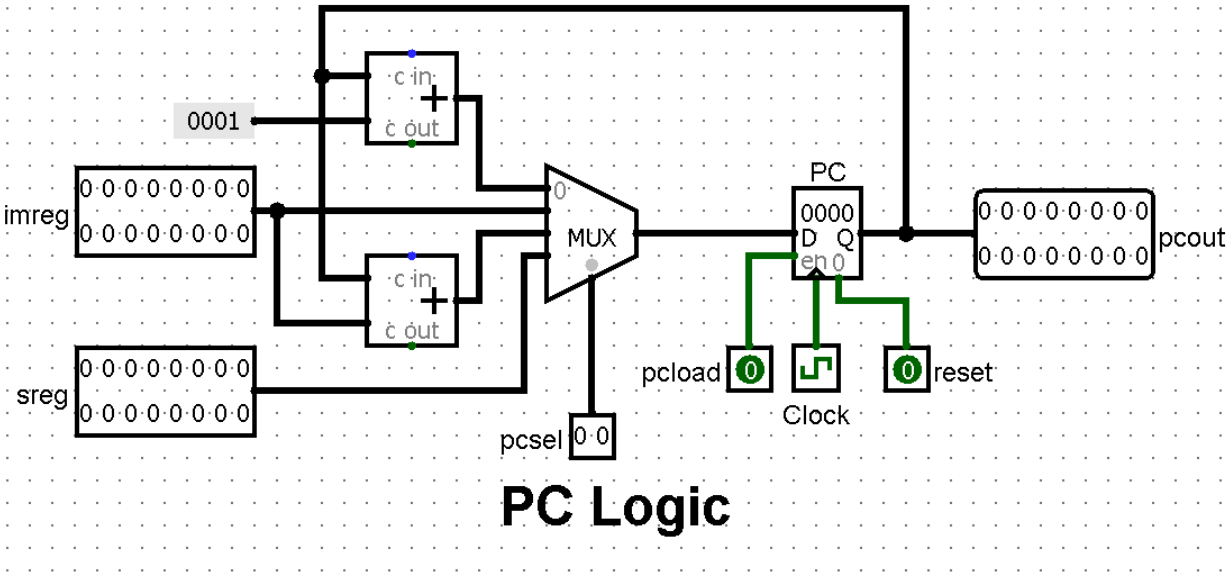
3.2.2. MiMo – podatkovna enota

3.2.2.3 Programski št.-PC

Vhodi:
- PC+1, Immreg, PC+Immreg, sreg

Izhodi:
- **pcout:** 1x 16-bitni izhod





Kontrolni signali:

pcload: določa vpis v PC

pcsel – določa vhod v PC :
- 0.. PC+1
- 1.. tak.operand (abs.skok)
- 2.. pc+tak.operand (vejitev)
- 3.. Sreg (vrnitev iz podprograma)

reset – postavi PC na 0

3.2.2. MiMo – podatkovna enota

3.2.2.4 Ukazni reg. - IR („Instruction register“)

Vhod: Podatkovno vodilo (Databus)

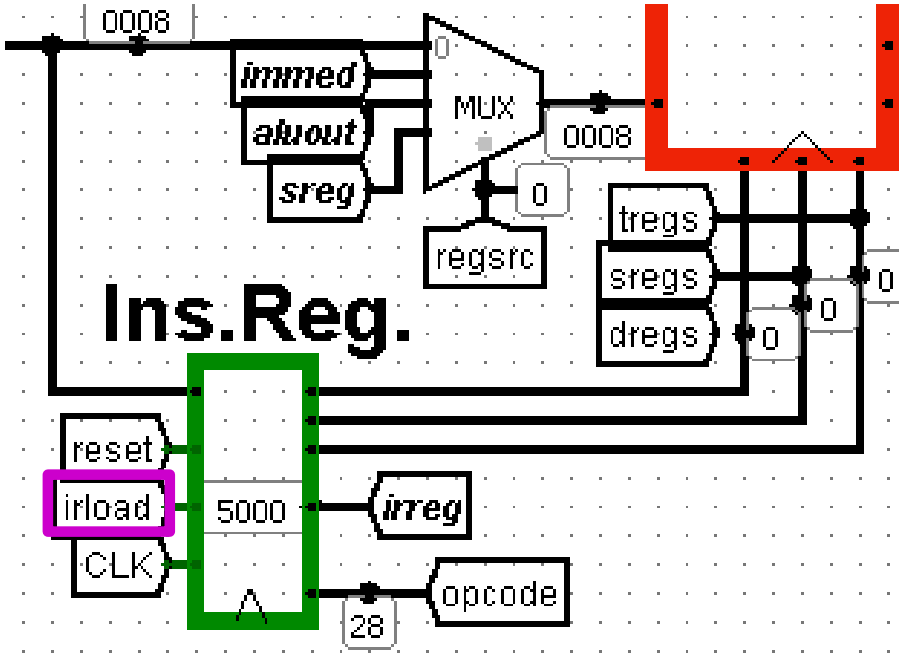
Izhodi: razdelitev ukaza na polja :

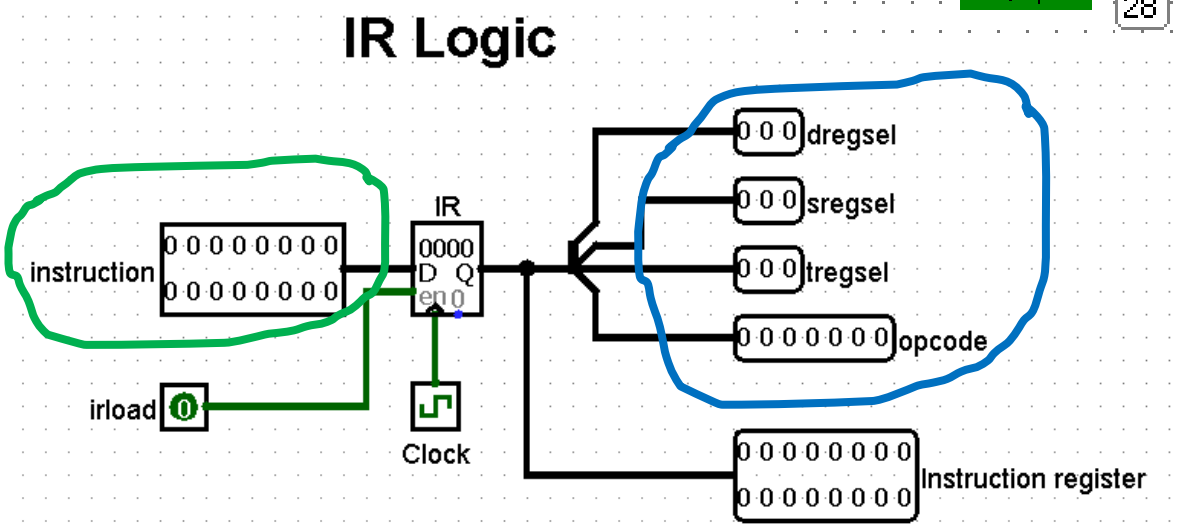
- **op.koda** (7 bitov) in
- 3x3biti za izbiro registrov **(dregs, sregs, tregs)**

skupaj 16 bitov

Kontrolni signal:

- **irload**: določa vpis v IR iz podatk. vodila





3.2.2. MiMo – podatkovna enota

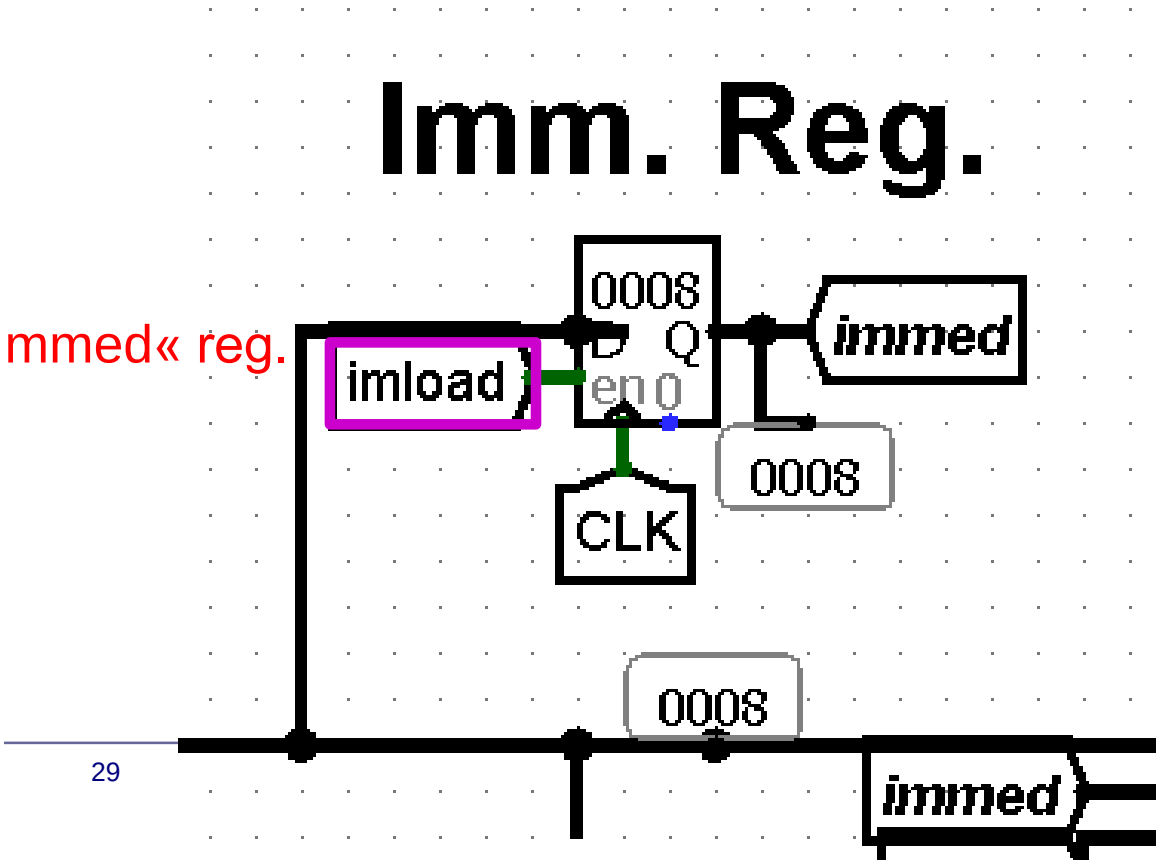
3.2.2.5 Takojšnji reg.-“immed“
(takojšnji register, „Immediate“)
shranjuje takojšnji operand

Vhod: Podatkovno vodilo

Izhod: „immed“

Kontrolni signali:

- **imload**: določa vpis v »Immed« reg. iz podatk. vodila



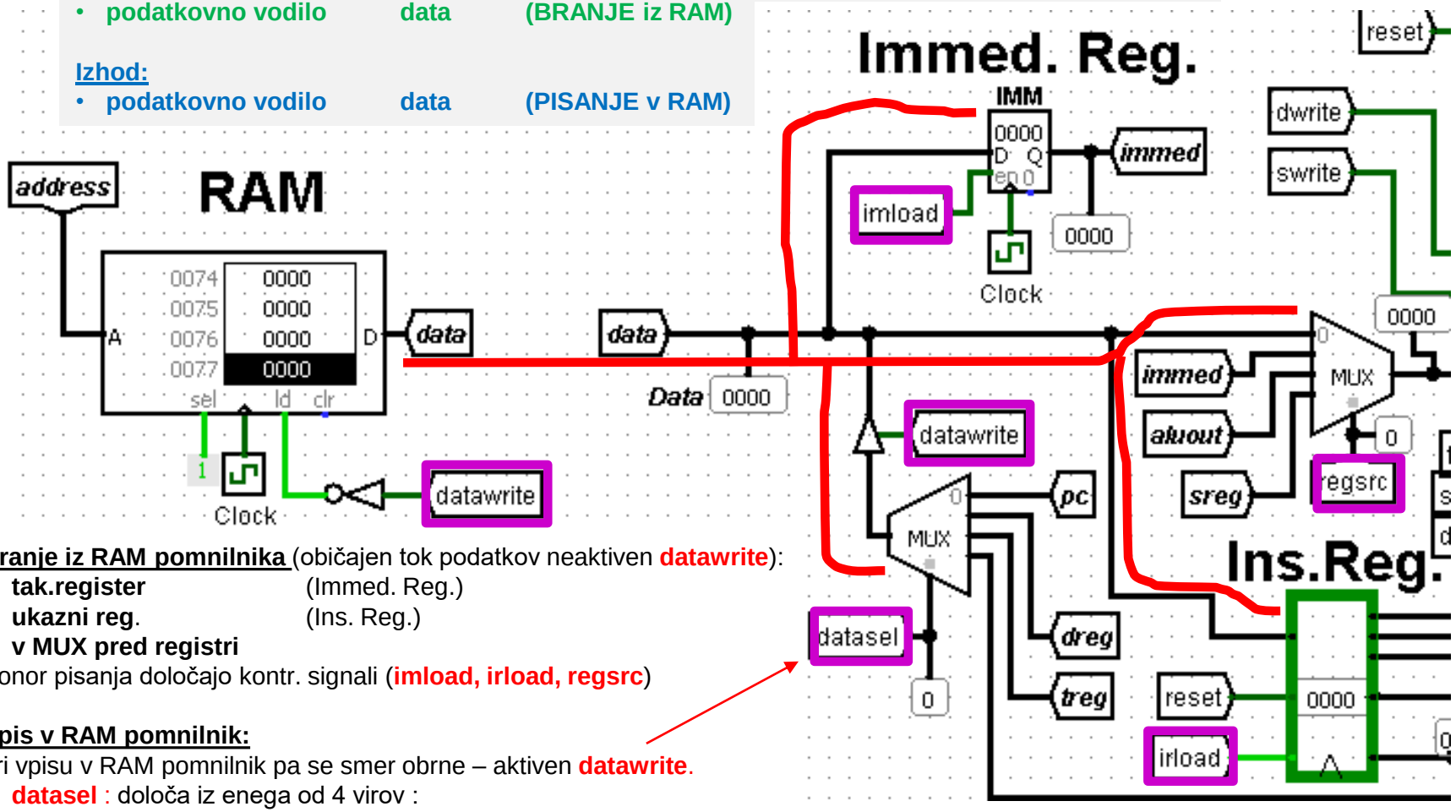
MiMo – podatkovna enota

3.2.2.6 Podatkovno vodilo

Vhod:
- podatkovno vodilo data (BRANJE iz RAM)

Izhod:
- podatkovno vodilo data (PISANJE v RAM)

Kontrolni:
- datawrite (0..BRANJE, 1..PISANJE)
- sel (vklop pomnilnika)



Branje iz RAM pomnilnika (običajen tok podatkov neaktiven **datawrite**):
- **tak.register** (Immed. Reg.)
- **ukazni reg**. (Ins. Reg.)
- **v MUX pred registri**

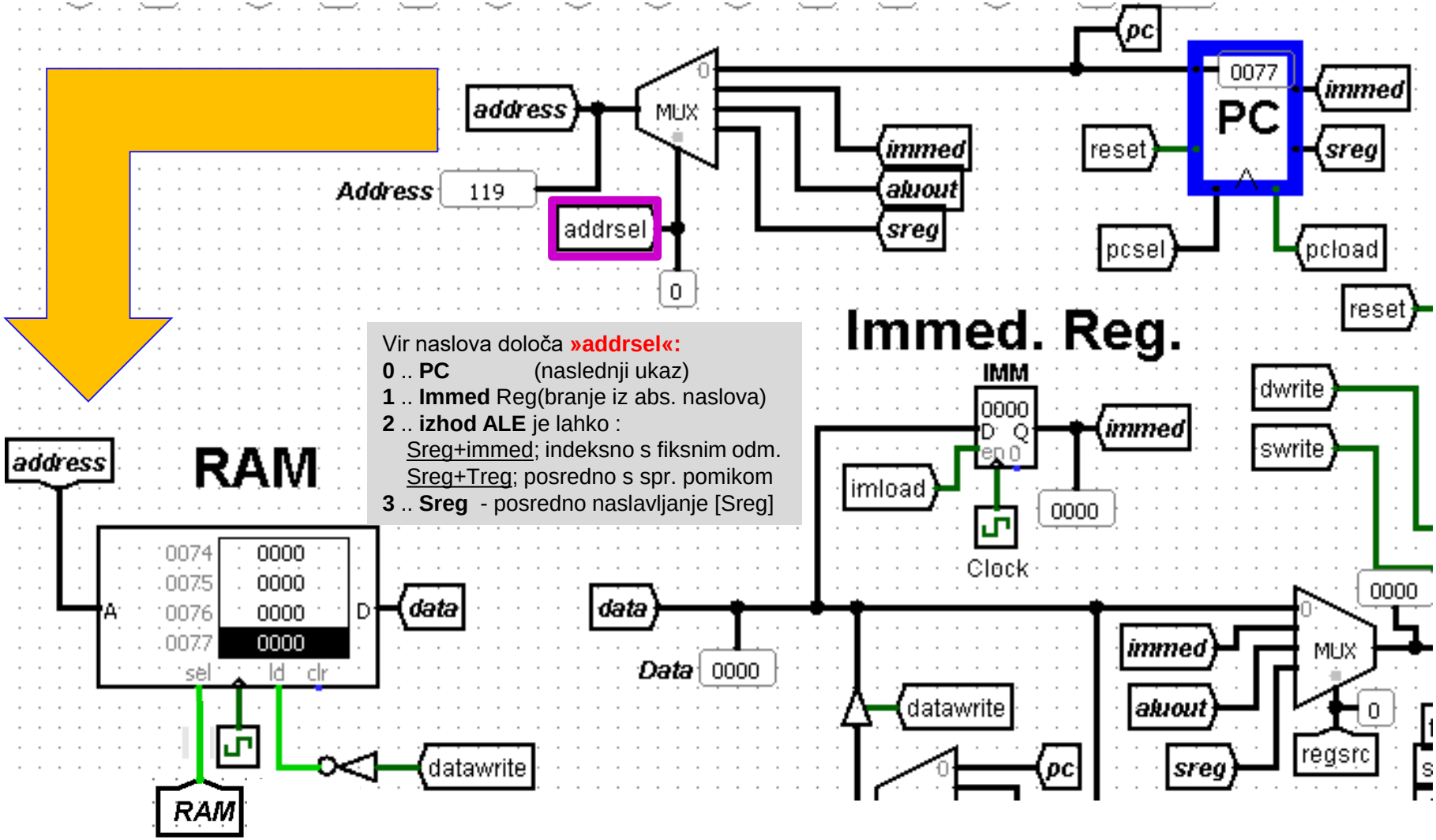
Ponor pisanja določajo kontr. signali (**imload, irload, regsrc**)

Vpis v RAM pomnilnik:

pri vpisu v RAM pomnilnik pa se smer obrne – aktiven **datawrite**.
- **datasel** : določa iz enega od 4 virov :
  - **0** .. PC (skok v podprogram,shranitev PCja)
  - **1** .. Dreg (pri ukazih STR Rx,naslov, vpis Rx->RAM)
  - **2** .. in **3** Treg in izhod ALE

MiMo – podatkovna enota

3.2.2.7 Naslovno vodilo

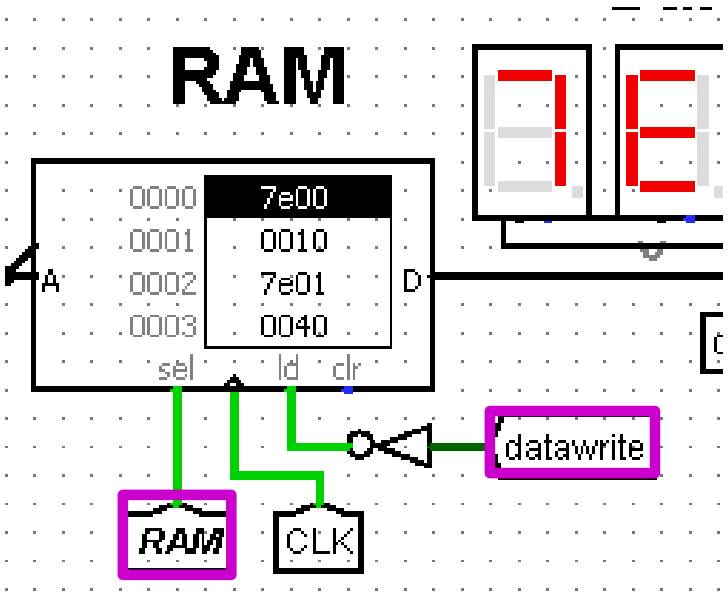


Vir naslova določa **»addrsel«:**
0 .. **PC** (naslednji ukaz)
1 .. **Immed** Reg(branje iz abs. naslova)
2 .. **izhod ALE** je lahko :
Sreg+immed; indeksno s fiksnim odm.
Sreg+Treg; posredno s spr. pomikom
3 .. **Sreg** - posredno naslavljanje [Sreg]

3.2.2.8 RAM pomnilnik

RAM pomnilnik je priključen na:

- podatkovno vodilo („**data**“)
- naslovno vodilo („**address**“)



Kontrolna signala:

- **datawrite** :
  - določa branje (0) ali pisanje (1)
- **sel(ect)** :
  - aktivira (1) RAM pomnilnik
  - deaktivira (0) RAM pomnilnik

  (uporaba: „naslovno dekodiranje“)

Vhod

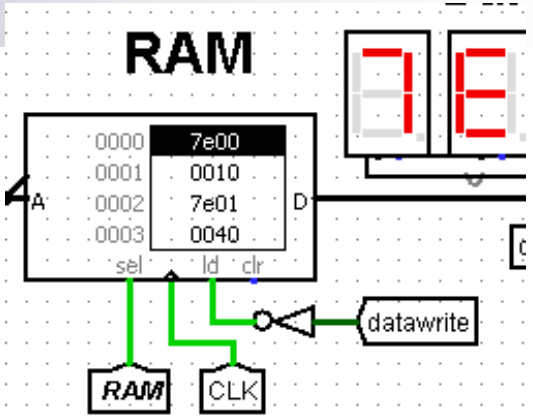
- naslov: 14 bitni, naslovni prostor 16-bitni

Izhod/Vhod:

- Podatkovno vodilo

3.2.2.8 RAM pomnilnik

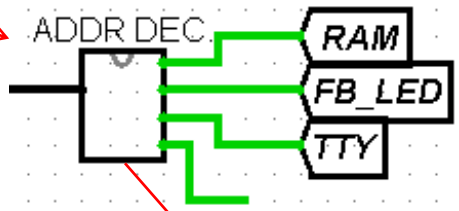
Naslov RAM 14 bitni
Naslov MiMo 16bitni ???

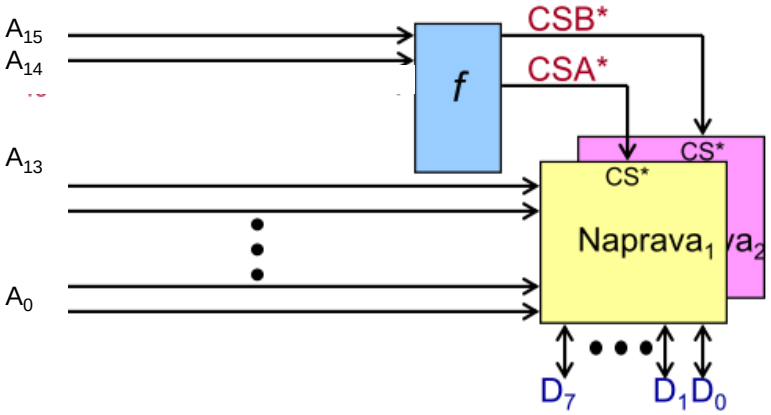


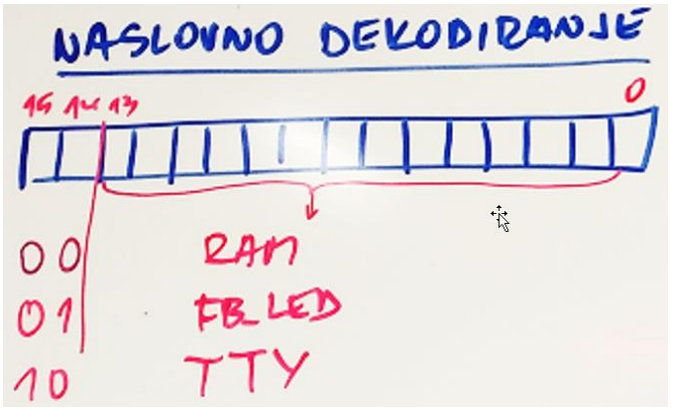
Naslovno dekodiranje

Izbira čipa (CS)

- **Kako priključimo dve (ali več) naprav na vodilo?**
  - Naenkrat mora biti izbran samo en čip (ali nobeden)
  - Za izbiro uporabimo naslednje signale:
    - R/W*, Naslov(A_0-A_{15})
- **Uporabni so biti, ki niso povezani na naslovne signale naprav** A_{15}-A_{14}
- CSA* in CSB* sta torej funkciji A_{15}-A_{14}

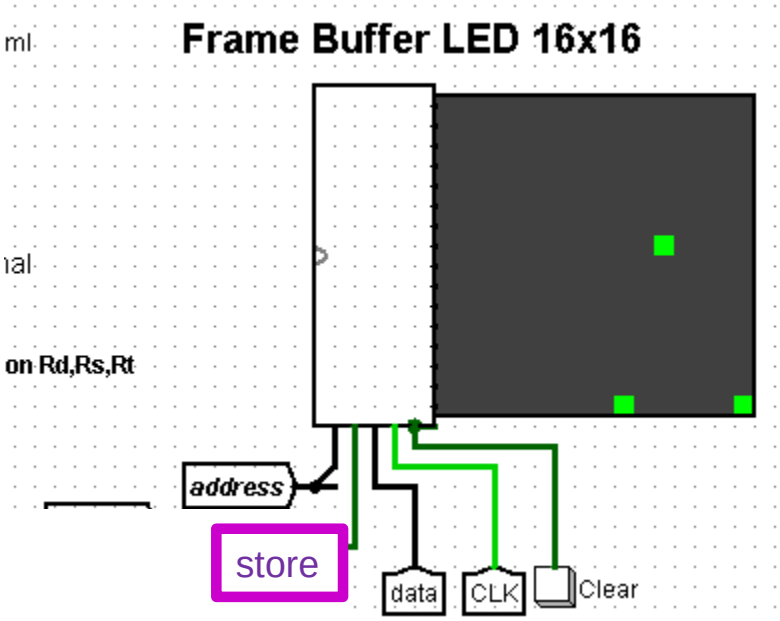






3.2.2.9 Grafični zaslon („FrameBuffer“)

Vsebina zaslona LED 16x16 je zapisana v 16 zaporednih 16-bitnih registrih („framebuffer“).



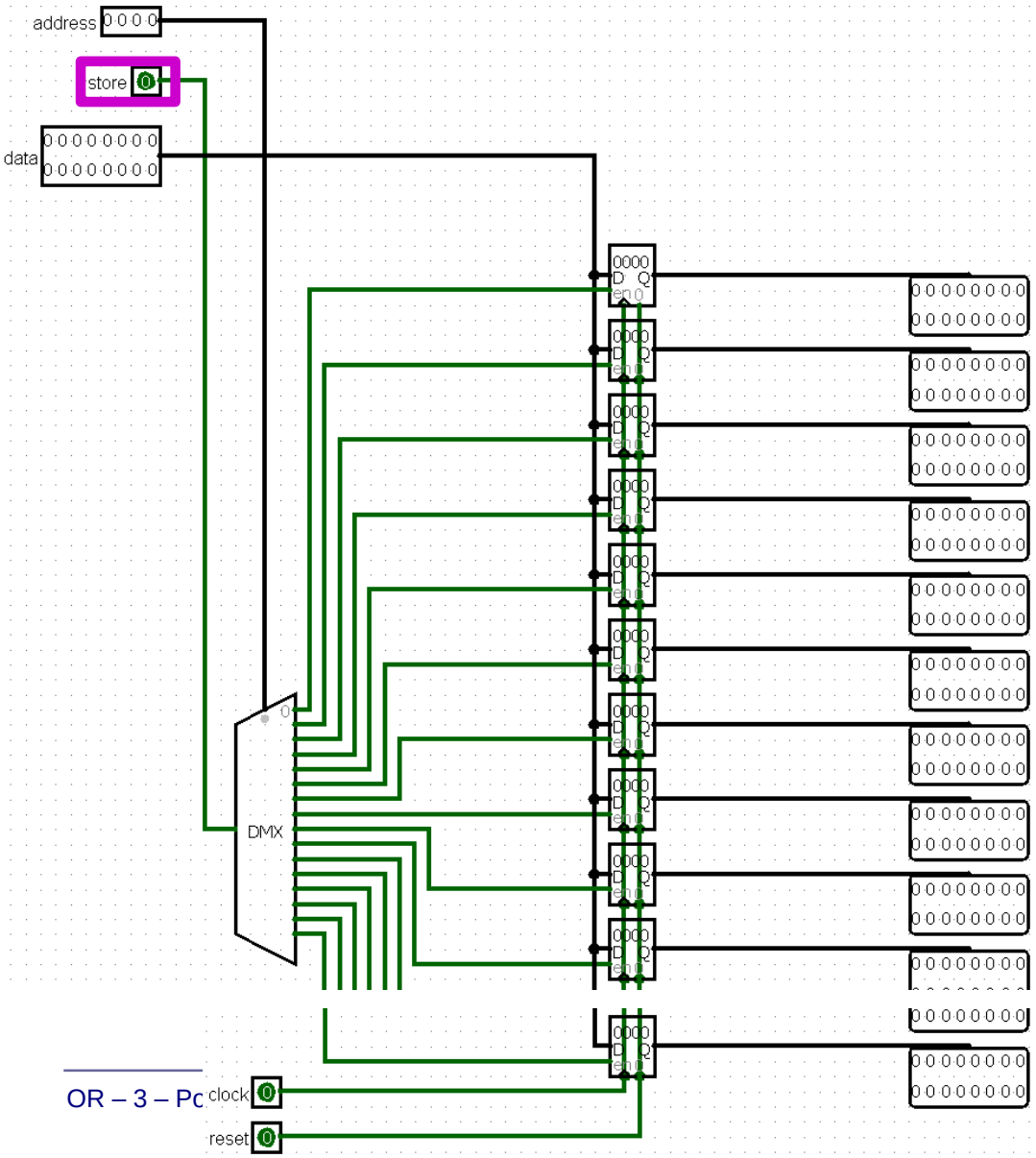
Vhodi:

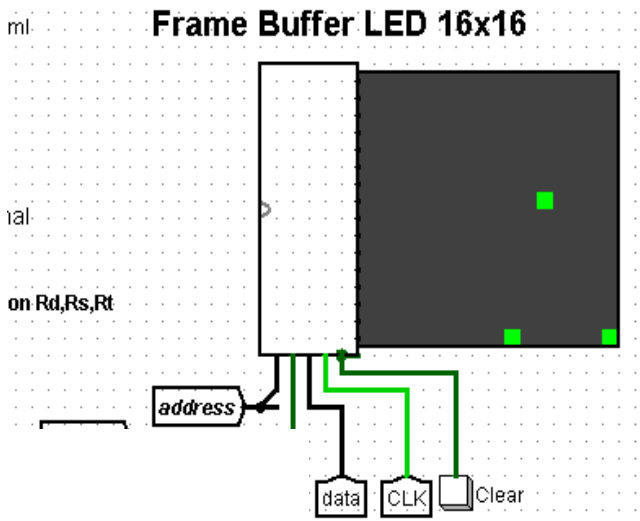
- address: naslov,
- data: podatek (vrednost),

Kontrolni signali:

- „store“ (pisanje),
- CLK,
- „Clear“

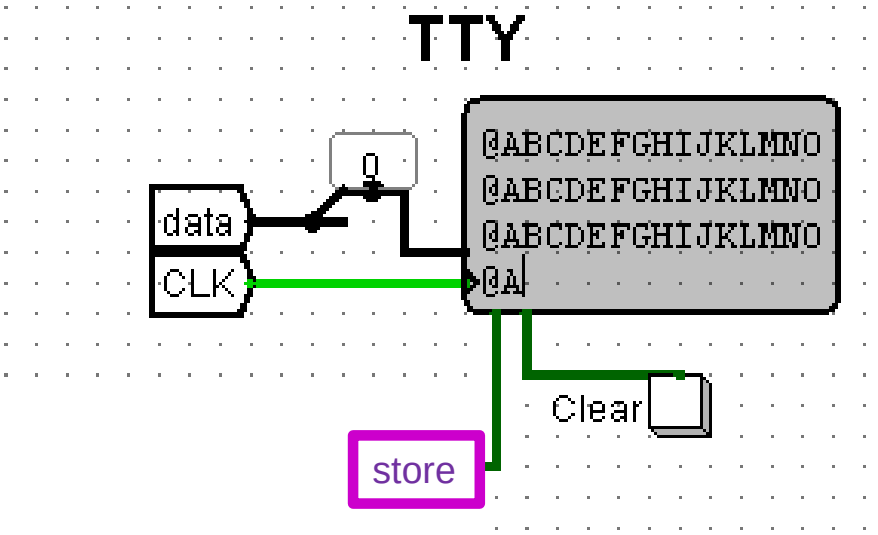
3.2.2.9 Grafični zaslon („FrameBuffer“)





3.2.2.10 Serijski terminal („TTY“)

Vsebina zaslona je prikazana v 4 vrsticah in 16 znakih.



Vhodi:

- data: podatek oz. znak (7bitna ASCII koda),

Kontrolni signali:

- „store“ (pisanje),
- CLK,
- „clear“.

3.2.3 MiMo – Kontrolna enota

Mikroukaz = elementarni korak

Vsak mikroukaz določa :

- stanje vseh kontrolnih signalov
- naslednji mikroukaz

Vhodi v KE:

opcode – operacijska koda ukaza

C, Z, N zastavice

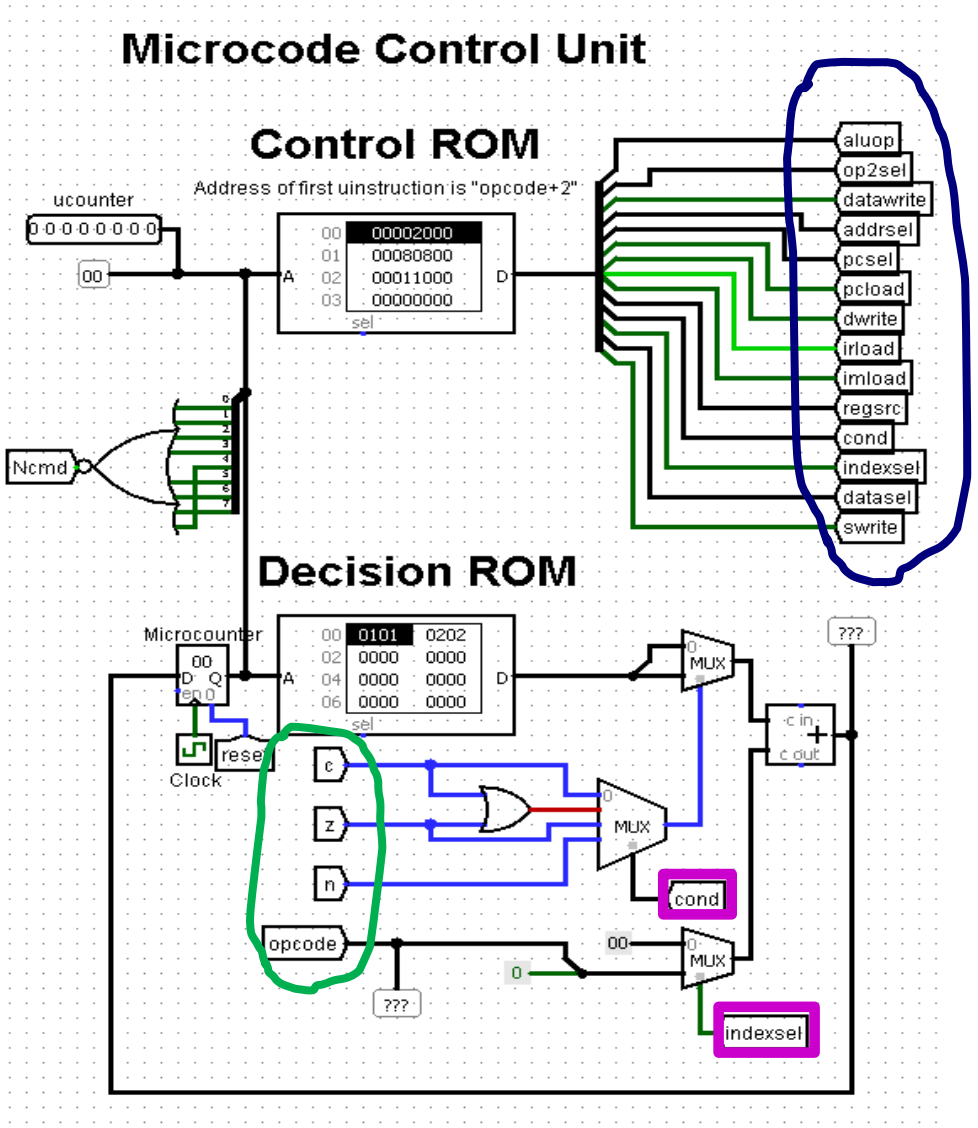
Izhodi iz KE

Vsi kontrolni signali

Kontrolni signali:

cond – izbira pogoja (C,CorZ,Z,N)

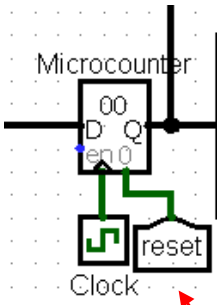
indexsel – opcode_jump
(skok na opcode+2)
ucounter(uPC)=2



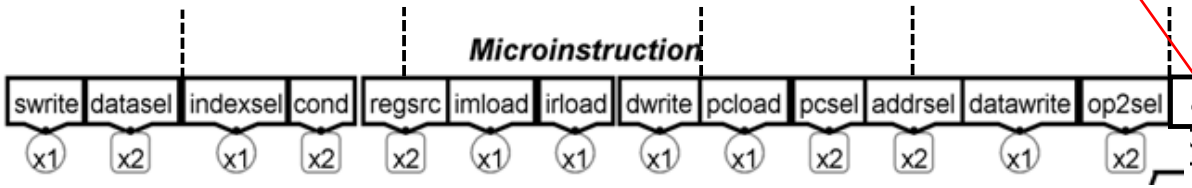
MiMo – Kontrolna enota

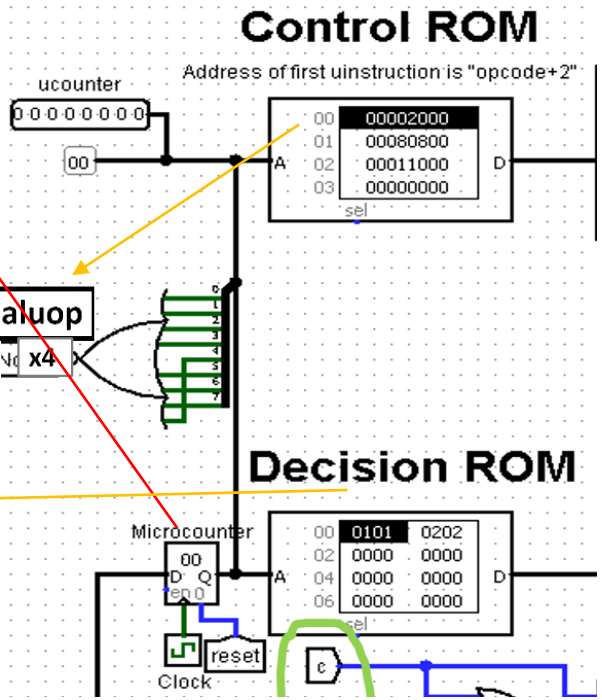
V MiMo je kontrolna enota sestavljena:

- **mikroprogramski števec „Microcounter“**



- **2 kontrolna ROM pomnilnika**
  - **32-bitni kontrolni ROM (»Control ROM«) :**





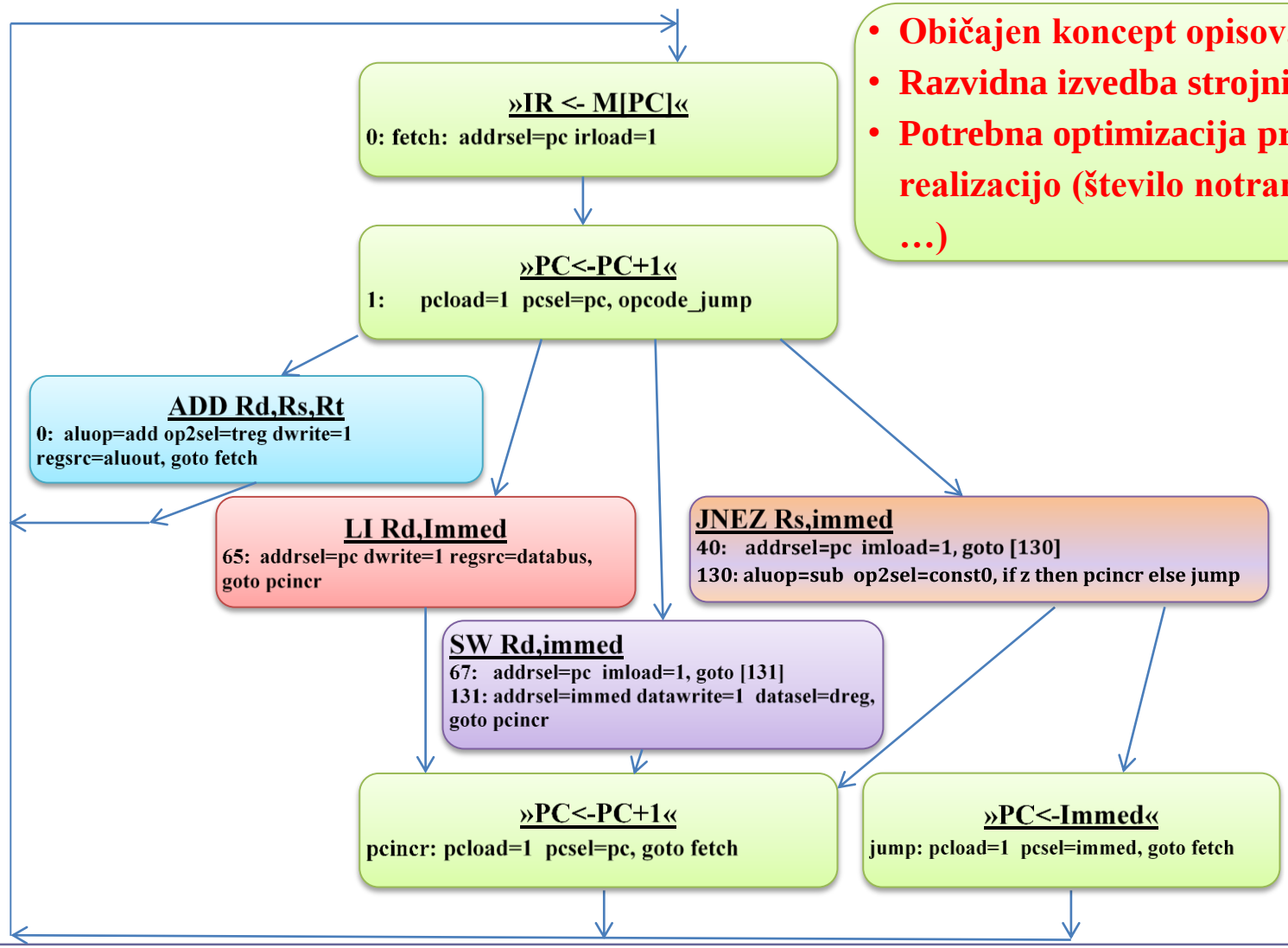
  - **16-bitni odločitveni ROM (»Decision ROM«) :**

T: 8	F: 8

    - T: naslov naslednjega mikroukaza, če je pogoj izpolnjen (0-0xff)
    - F: naslov naslednjega mikroukaza, če pogoj ni izpolnjen (0-0xff)

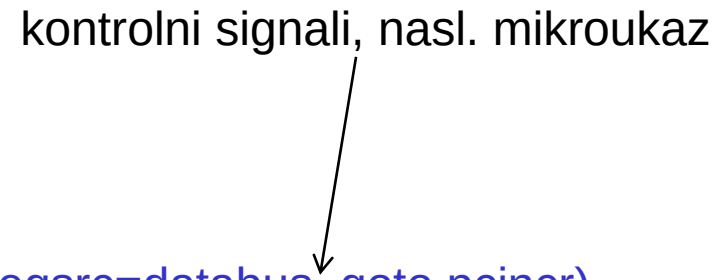
- **Ostala logika**
  - **Zastavice (C,N,Z), spremembe mikroprogramskega števca, …**

MiMo – Diagram prehajanja stanj



- Običajen koncept opisovanja KE
- Razvidna izvedba strojnih ukazov
- Potrebna optimizacija pred samo realizacijo (število notranjih stanj, …)

3.2.4 Mikro-zbirnik



- Mikroukaz : (63: addrsel=pc dwrite=1 regsrc=databus, goto pcincr)

Op.koda	kontrolni signali	naslednji mikroukaz

- Večbitni kontrolni signali:

kontr. signal	opisna vrednost	enota
aluop	add, sub, mul, div, rem, and, or, xor, nand, nor, not, lsl, lsr, asr, rol, ror	ALE
op2sel	treg, immed, const0, const1	ALE
addrsel	pc, immed, aluout, sreg	nasl. vodilo
pcsel	pc, immed, pcimmed, sreg	PC
regsrc	databus, immed, aluout, sreg	registri
cond	c, corz, z, n	kontr. enota

Mikro-zbirnik

datoteka **basic_microcode.def**

```
fetch:   addrsel=pc irload=1                    # Address=PC, Load IR register
         pcload=1  pcsel=pc, opcode_jump        # PC=PC+1, jump to 2+OPC

# ALU operation '+' on Rd,Rs,Rt
0:       aluop=add  op2sel=treg  dwrite=1  regsrc=aluout, goto fetch

# JNEZ Rs,immed
40:      addrsel=pc  imload=1
         aluop=sub  op2sel=const0, if z then pcincr else jump

# li Rd,Immed
63:      addrsel=pc  dwrite=1  regsrc=databus, goto pcincr

# Rd->M[immed]
65:      addrsel=pc  imload=1
         addrsel=immed  datawrite=1  datasel=dreg, goto pcincr

pcincr:  pcload=1  pcsel=pc, goto fetch

jump:    pcload=1  pcsel=immed, goto fetch
```

se prevede v

indexsel, pcload

```
00: 00002000 0101    # fetch: addrsel=pc irload=1
01: 00080800 0202    #          pcload=1  pcsel=pc, opcode_jump
02: 00011000 0000    # 0:     aluop=add  op2sel=treg  dwrite=1  regsrc=aluout,    goto fetch
2a: 00004000 8282    # 40:    addrsel=pc  imload=1                               (goto 82)
41: 00001000 8484    # 63:    addrsel=pc  dwrite=1  regsrc=databus,              goto pcincr
43: 00004000 8383    # 65:    addrsel=pc  imload=1                               (goto 83)
82: 00040021 8485    #          aluop=sub  op2sel=const0,                        if z then pcincr else jump
83: 001000c0 8484    #          addrsel=immed  datawrite=1  datasel=dreg,        goto pcincr
84: 00000800 0000    # pcincr:      pcload=1  pcsel=pc,                          goto fetch
85: 00000a00 0000    # jump:  pcload=1  pcsel=immed,                             goto fetch
```

Zaporedje mikroukazov za ukaz v zbirniku: JNEZ Rs,immed

Oznaka za razlago	Program v mikro-zbirniku za strojni ukaz **JNEZ Rs,immed**
	# Common start for all uinstructions : Read from M[PC] to IR, increment PC and goto »2+opcode«
a:	**fetch: addrsel=pc irload=1** # Address=PC, Load IR register, goto next line (empty 2nd part)
b:	**pcload=1 pcsel=pc, opcode_jump** # PC=PC+1, goto »2+opcode«
	# Example of assembler instruction body: JNEZ Rs,immed (opcode=40, address = 40+2=42)
c:	**40: addrsel=pc imload=1** # Read Immediate operand -> IMRegister
d:	**aluop=sub op2sel=0, if z then pcincr else jump** # If z then pcincr else jump to immed
	# Increment PC and goto new command; for all commands that use immediate operand
e:	**pcincr: pcload=1 pcsel=pc, goto fetch** #additional PC<-PC+1, read new uinstruction
	# Set address to immed and goto new command; for absolute jumps to immed address
f:	**jump: pcload=1 pcsel=immed, goto fetch** #read new uinstruction from immed address

RAZLAGA:

a: najprej se prebere ukaz iz pomn. naslova PC v ukazni register (IR): **IR<-M[PC]**
b: PC se poveča za 1, nato skoči na vrstico »op.koda+2« : **PC<-PC+1, goto »op.koda + 2«**
c: v register »immed« se prebere operand iz pomnilnika M[PC]: **immed <- M[PC]**
d: izvedi ALU operacijo SUB med Rs in konst.0, **če rez=0 goto pcincr:, sicer pojdi na jump**:
e: če velja **Rs=0**, **povečaj PC in nadaljuj z novim ukazov** (ni skoka)
f: če velja **Rs≠0, skoči na immed naslov**

Razpored mikroukazov v obeh kontrolnih pomnilnikih po naslovih:

Naslov	Vsebina
0,1	mikroukaza za **branje mikroukazov** in **povečevanje PC (a: in b:)**
2-129	prvi **mikroukazi** za vse ukaze **z op. kodo 0-127** (naslov=op.koda+2)
130+ 0x82+	vsi ostali mikroukazi za vse ukaze v zbirniku (si sledijo po vrstnem redu)

Prevajanje mikroprograma:

- **.\micro_assembler.exe basic_microcode.def**
- pripravi se **datoteki,** ki se vneseta v kontrolno enoto v MiMo model (Logisim):
  - ucontrol.rom
  - udecision.rom
- vnos obeh v Logisim: **desni klik na ROM elementa in »Load Image«**

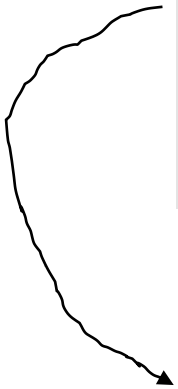
3.2.5 Zbirnik

- Prevajanje programa v zbirniku:
  - .\assembler.exe basic_program.s
- pripravi se datoteka, ki se vnese v RAM pomnilnik v MiMo model (Logisim):
  - **basic_program.ram**
- primer prevajanja v zbirniku ->

3.2.5 Zbirnik

- **Primer (testni):**

```
main:   li    r0, 0          # r0 is the running sum
        li    r1, 100        # r1 is the counter
        li    r2, -1         # Used to decrement r1
loop:   add   r0, r0, r1     # r0= r0 + r1
        add   r1, r1, r2     # r1--
        jnez  r1, loop       # loop if r1 != 0
        sw    r0, 256        # Save the result
inf:    jnez  r2, inf        # loop if r1 != 0 -> loop forever
```



```
0000: 00007e00  0111111000000000        main:   li      r0, 0
0001: 00000000  0000000000000000
0002: 00007e01  0111111000000001                li      r1, 100
0003: 00000064  0000000001100100
0004: 00007e02  0111111000000010                li      r2, -1
0005: 0000ffff  1111111111111111
0006: 00000040  0000000001000000        loop:   add     r0, r0, r1
0007: 00000089  0000000010001001                add     r1, r1, r2
0008: 00005008  0101000000001000                jnez  r1, loop
0009: 00000006  0000000000000110
000a: 00008200  1000001000000000                sw      r0, 256
000b: 00000100  0000000100000000
000c: 00005010  0101000000010000        inf:    jnez  r2, inf
000d: 0000000c  0000000000001100
```

Op.koda	Treg	Sreg	Dreg
7	3	3	3

Zbirnik – primeri ukazov

rdeče: trenutno že implementirani ukazi v modelu MiMo v04b.

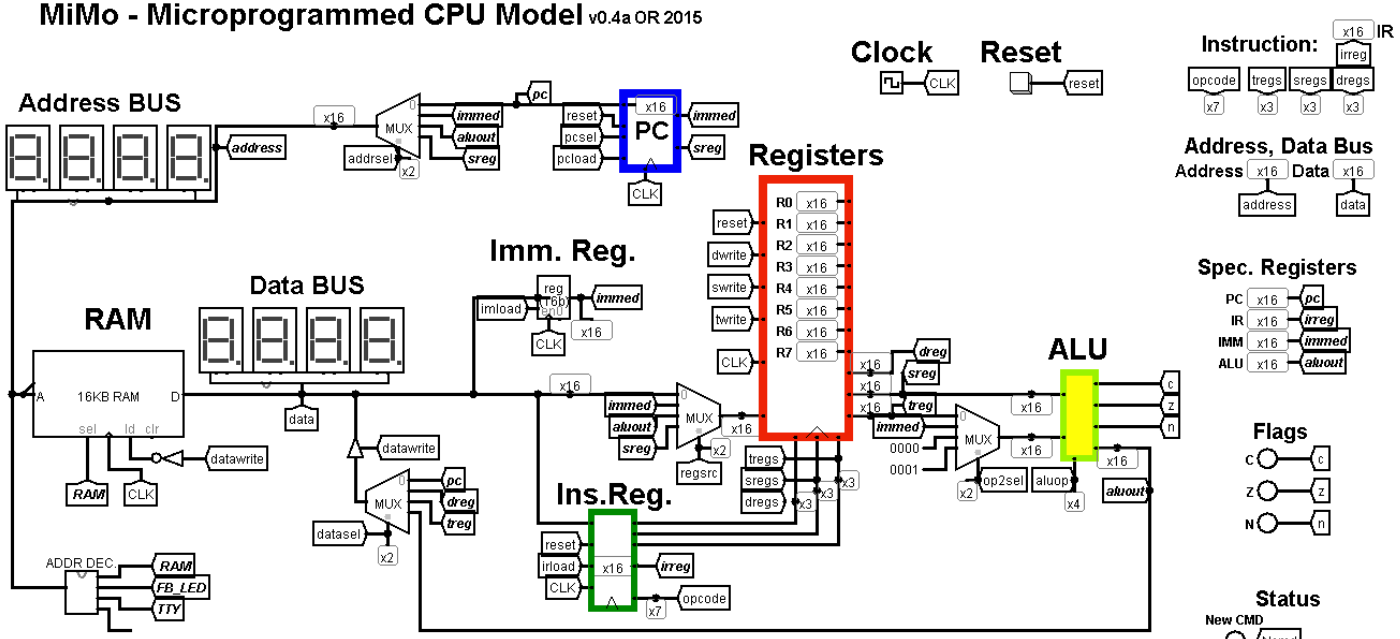
assembler.pl (zbirnik), list_of_instructions.txt (dokumentacija) :

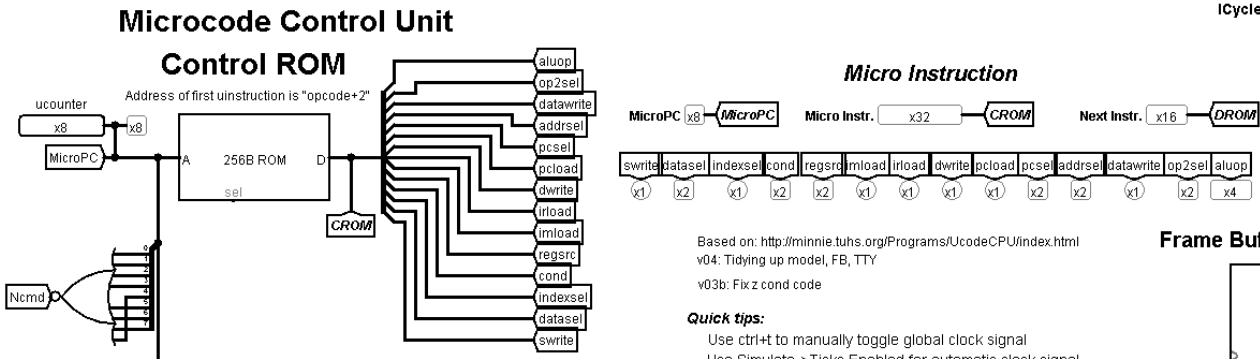
- add Rd,Rs,Rt (0) — Rd <- Rs + Rt, PC <- PC + 1
- sub Rd,Rs,Rt (1) — Rd <- Rs – Rt, PC <- PC + 1
- …
- jeqz Rs,immed (39) — if Rs == 0, PC <- immed else PC <- PC + 2
- jnez Rs,immed (40) — if Rs != 0, PC <- immed else PC <- PC + 2
- …
- beq Rs,Rt,immed (46) — if Rs == Rt, PC <- PC + immed else PC <- PC + 2
- bne Rs,Rt,immed (47) — if Rs != Rt, PC <- PC + immed else PC <- PC + 2
- …
- li Rd,immed (63) — Rd <- immed, PC <- PC + 2
- sw Rd,immed (65) — M[immed] <- Rd, PC <- PC + 2
- …
- lw Rd,immed (64) — Rd <- M[immed], PC <- PC + 2
- lwi Rd,Rs,immed (66) — Rd <- M[Rs+immed], PC <- PC + 2
- swi Rd,Rs,immed (67) — M[Rs+immed] <- Rd, PC <- PC + 2

MiMo – Model Mikroprogramirane CPE

Naslov/ signal	Kontrolni (»Control«) ROM 256x32bitov (23 izkoriščenih)														Opis vsebine mikroprograma				Odločitveni (»Decision«) ROM 256x16bitov	
	1	2	1	2	2	1	1	1	1	2	2	1	2	4	Oznaka/ op.koda:	Oznaka: strojni ukaz ali »mikroukaz«	Opis mikroukaza	Mikroukaz	true 8bit	false 8bit
	swrite	datasel	indexsel	cond	regsrc	imload	irload	dwrite	pcload	pcsel	addrsel	datawrite	op2sel	aluop						
0							1				0				fetch:	»IR<-M[PC]«	IR<-M[PC],goto [1]	addrsel=pc irload=1	1	1
1			1						1	0						»PC<-PC+1«	PC++, goto »Op+2«	pcload=1 pcsel=pc, opcode_jump	2	2
2					2			1							0:	ADD Rd,Rs,Rt	ADD op. Rd,Rs,Rt, goto fetch:	aluop=add op2sel=treg dwrite=1 regsrc=aluout, goto fetch	0	0
42 0x2a						1					0				40:	JNEZ Rs,immed	immed<-M[PC], goto [0x82]	addrsel=pc imload=1	82	82
65 0x41								1			0				63:	LI Rd,Immed	Rd<-M[PC], goto pcincr:	addrsel=pc dwrite=1 regsrc=databus, goto pcincr	84	84
67 0x43						1					0				65:	SW Rd,immed	immed<-M[PC], goto [0x83]	addrsel=pc imload=1, goto 83	83	83
130 0x82				2									2	1		JNEZ Rs,immed	SUB op. Rs-0, if Z then pcincr: else jump:	aluop=sub op2sel=const0, if z then pcincr else jump	84	85
131 0x83		1									1	1				SW Rd,immed	Rd->M[immed]; goto pcincr:	addrsel=immed datawrite=1 datasel=dreg, goto pcincr	84	84
132 0x84									1						pcincr:	PC++, goto fetch:	PC<-PC+1, goto fetch:	pcload=1 pcsel=pc, goto fetch	0	0
133 0x85									1	1					jump:	PC<-immed, goto fetch:	immed->PC, goto fetch:	pcload=1 pcsel=immed, goto fetch	0	0

MiMo - Microprogrammed CPU Model v0.4a OR 2015





Based on: http://minnie.tuhs.org/Programs/UcodeCPU/index.html
v04: Tidying up model, FB, TTY

v03b: Fix z cond code

Quick tips:

Use ctrl+t to manually toggle global clock signal

Use Simulate->Ticks Enabled for automatic clock signal

2:aluop=add op2sel=treg dwrite=1 regsrc=aluout, goto fetch # ALU operations on Rd,Rs,Rt

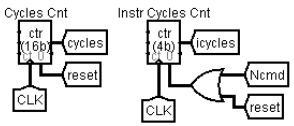
2a [42]:addrsel=pc imload=1 # JNEZ Rs,immed

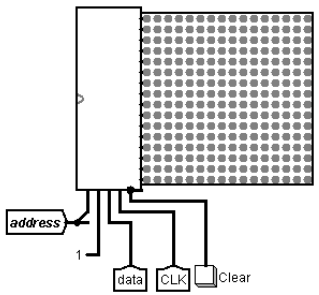
41 [65]:addrsel=pc dwrite=1 regsrc=databus # Load immediate Rd, immed

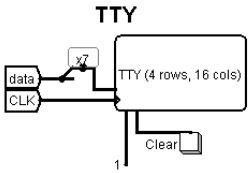
43 [67]:addrsel=pc imload=1 # Store Rd into address from immed

84 [132]: pcincr: pcload=1 pcsel=pc, goto fetch

85 [133]: jump: pcload=1 pcsel=immed, goto fetch







Primer izvedbe ukaza :

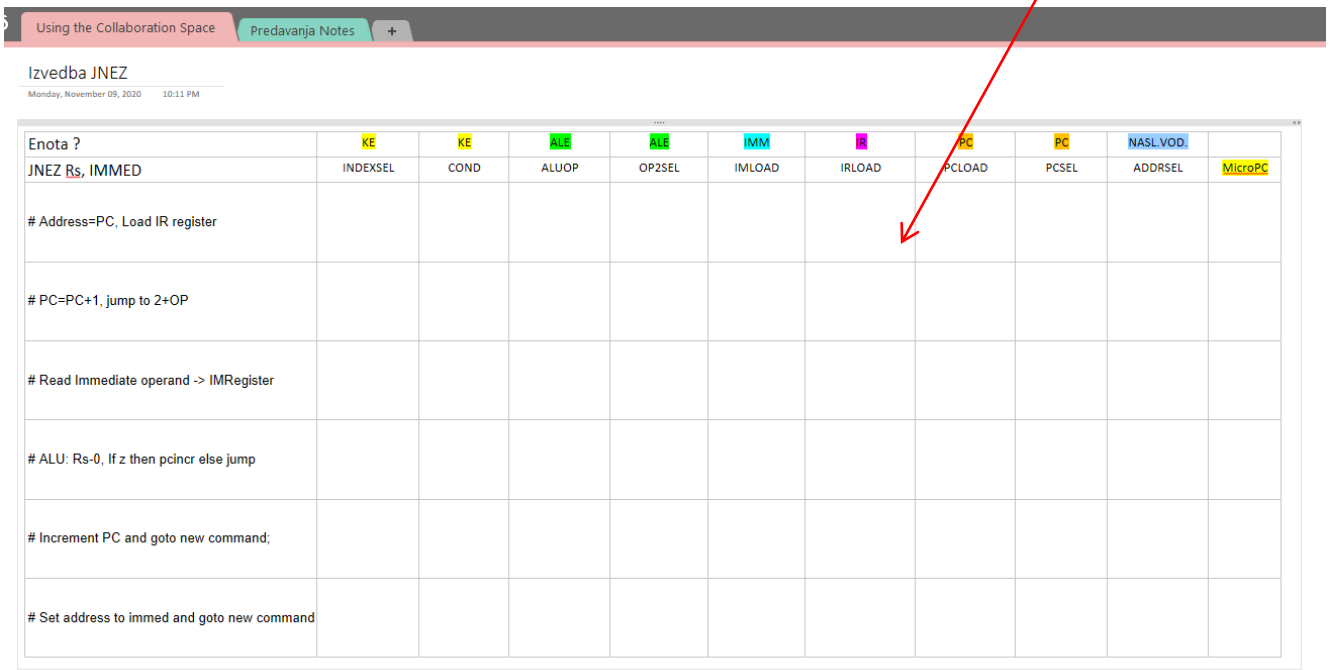
```
main: li    r0, 0          # r0 is the running sum
      li    r1, 100        # r1 is the counter
      li    r2, -1         # Used to decrement r1
loop: add   r0, r0, r1     # r0= r0 + r1
      add   r1, r1, r2     # r1--
      jnez  r1, loop       # loop if r1 != 0
      sw    r0, 256        # Save the result
```


Izvedba strojnega ukaza JNEZ R1,LOOP po skupinah kontrolnih signalov

Določite aktivna stanja kontrolnih signalov pri vsakem mikroukazu

Primer izvedbe ukaza :

```
main: li    r0, 0          # r0 is the running sum
      li    r1, 100        # r1 is the counter
      li    r2, -1         # Used to decrement r1
loop: add   r0, r0, r1     # r0= r0 + r1
      add   r1, r1, r2     # r1--
      jnez  r1, loop       # loop if r1 != 0
      sw    r0, 256        # Save the result
```



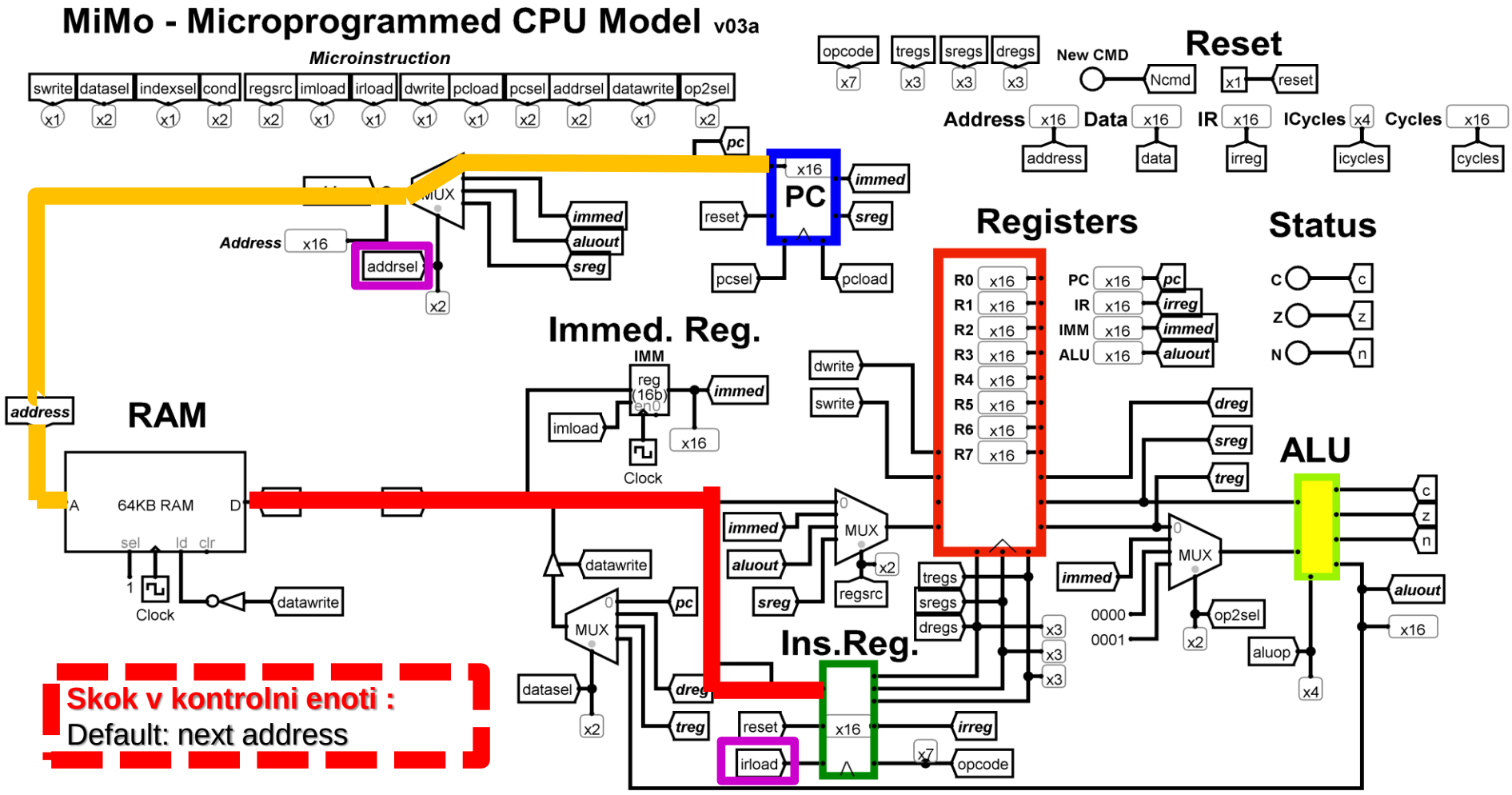
Enota ?	KE	KE	ALE	ALE	IMM	IR	PC	PC	NASL.VOD.	
JNEZ Rs, IMMED	INDEXSEL	COND	ALUOP	OP2SEL	IMLOAD	IRLOAD	PCLOAD	PCSEL	ADDRSEL	MicroPC
# Address=PC, Load IR register										
# PC=PC+1, jump to 2+OP										
# Read Immediate operand -> IMRegister										
# ALU: Rs-0, If z then pcincr else jump										
# Increment PC and goto new command;										
# Set address to immed and goto new command										

JNEZ Rs,immed:

jnez Rs,immed (40)

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

```
fetch:
        # Address=PC, Load IR register
        # PC=PC+1, jump to 2+OPC
40:     # Read Immediate operand -> IMRegister
        # ALU: Rs-0, If z then pcincr else jump
pcincr: # Increment PC and goto new command;
jump:   # Set address to immed and goto new command
```

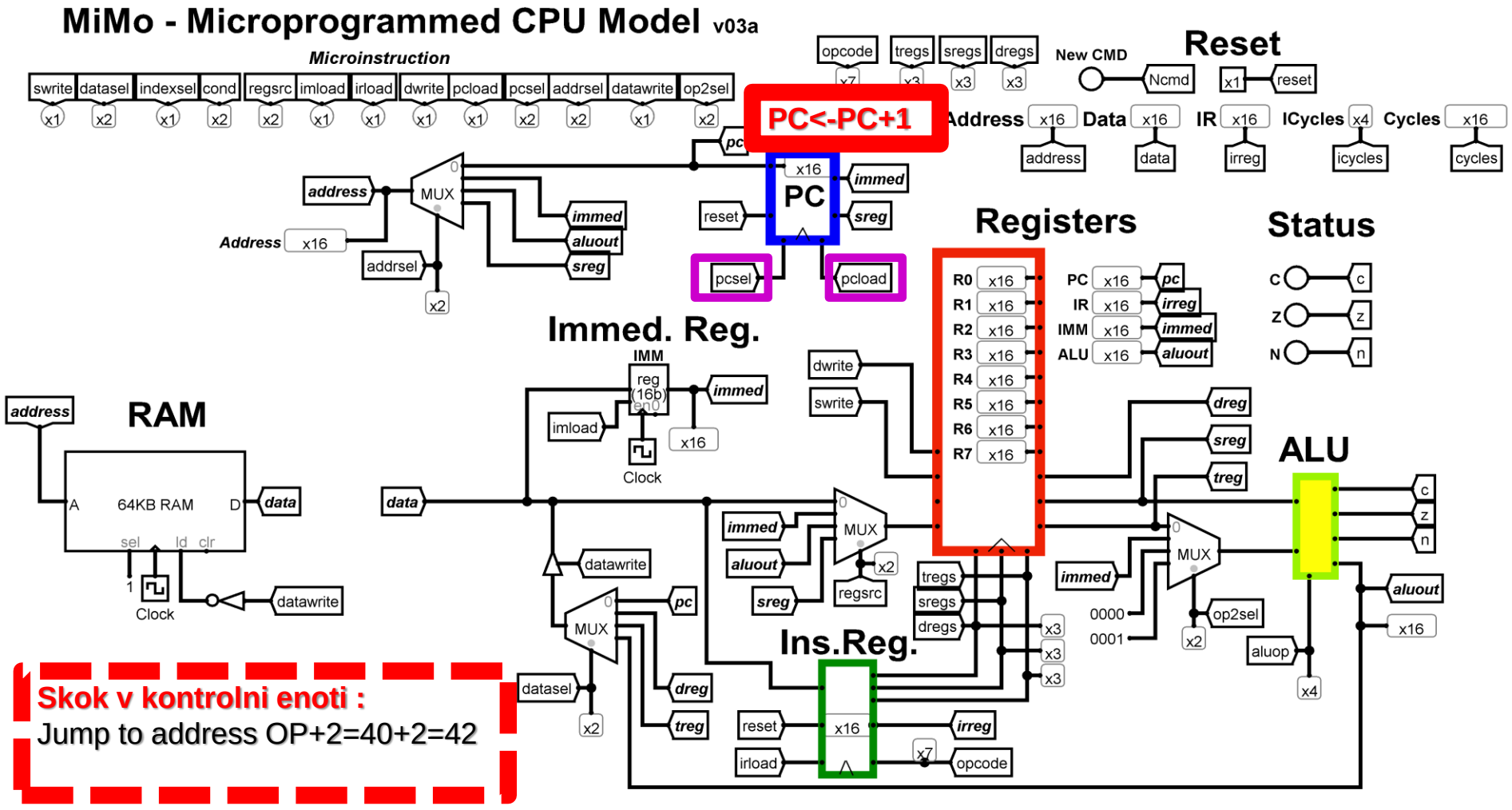


JNEZ Rs,immed:

jnez Rs,immed (40)

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

```
fetch:
                    # Address=PC, Load IR register
                    # PC=PC+1, jump to 2+OPC
40:
                    # Read Immediate operand -> IMRegister
                    # ALU: Rs-0, If z then pcincr else jump
pcincr:             # Increment PC and goto new command;
jump:               # Set address to immed and goto new command
```



Skok v kontrolni enoti :

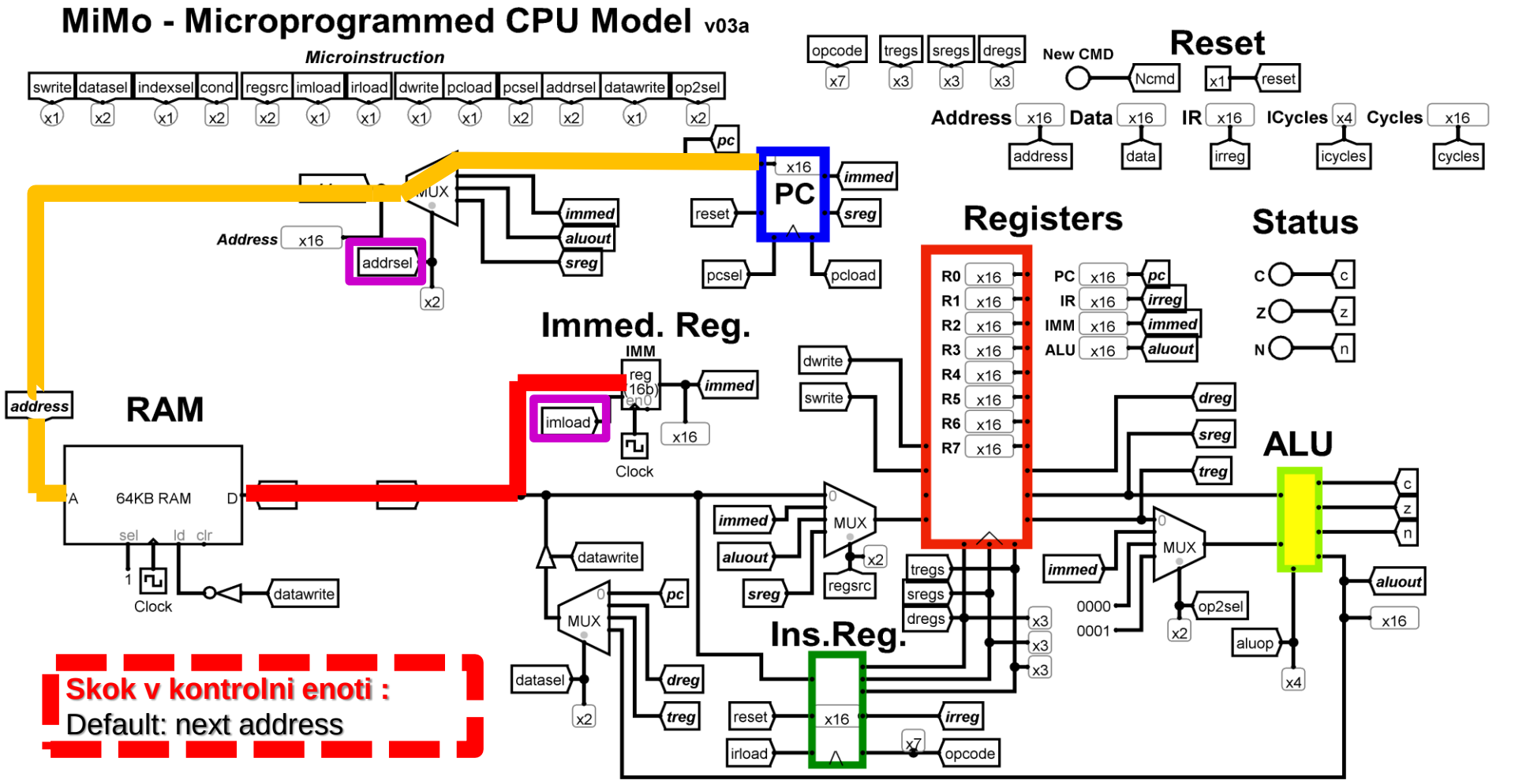
Jump to address OP+2=40+2=42

JNEZ Rs,immed:

jnez Rs,immed (40)

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

```
fetch:
                                # Address=PC, Load IR register
                                # PC=PC+1, jump to 2+OPC
40:
                                # Read Immediate operand -> IMRegister
                                # ALU: Rs-0, If z then pcincr else jump
pcincr:
                                # Increment PC and goto new command;
jump:
                                # Set address to immed and goto new command
```



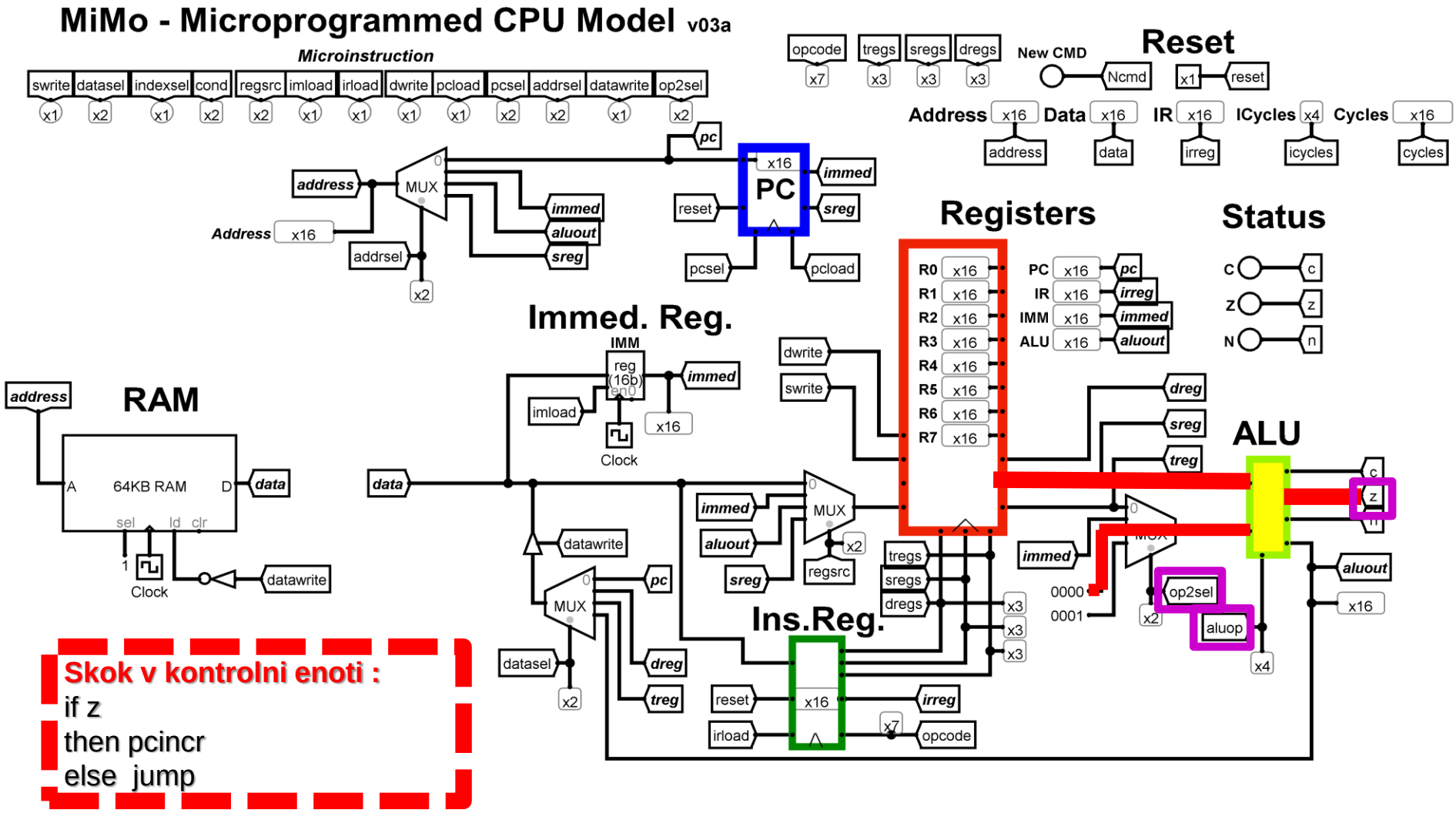
Skok v kontrolni enoti :
Default: next address

JNEZ Rs,immed:

jnez Rs,immed (40)

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

```
fetch:
        # Address=PC, Load IR register
        # PC=PC+1, jump to 2+OPC
40:
        # Read Immediate operand -> IMRegister
        # ALU: Rs-0, If z then pcincr else jump
pcincr:
        # Increment PC and goto new command;
jump:
        # Set address to immed and goto new command
```



Skok v kontrolni enoti :

if z

then pcincr

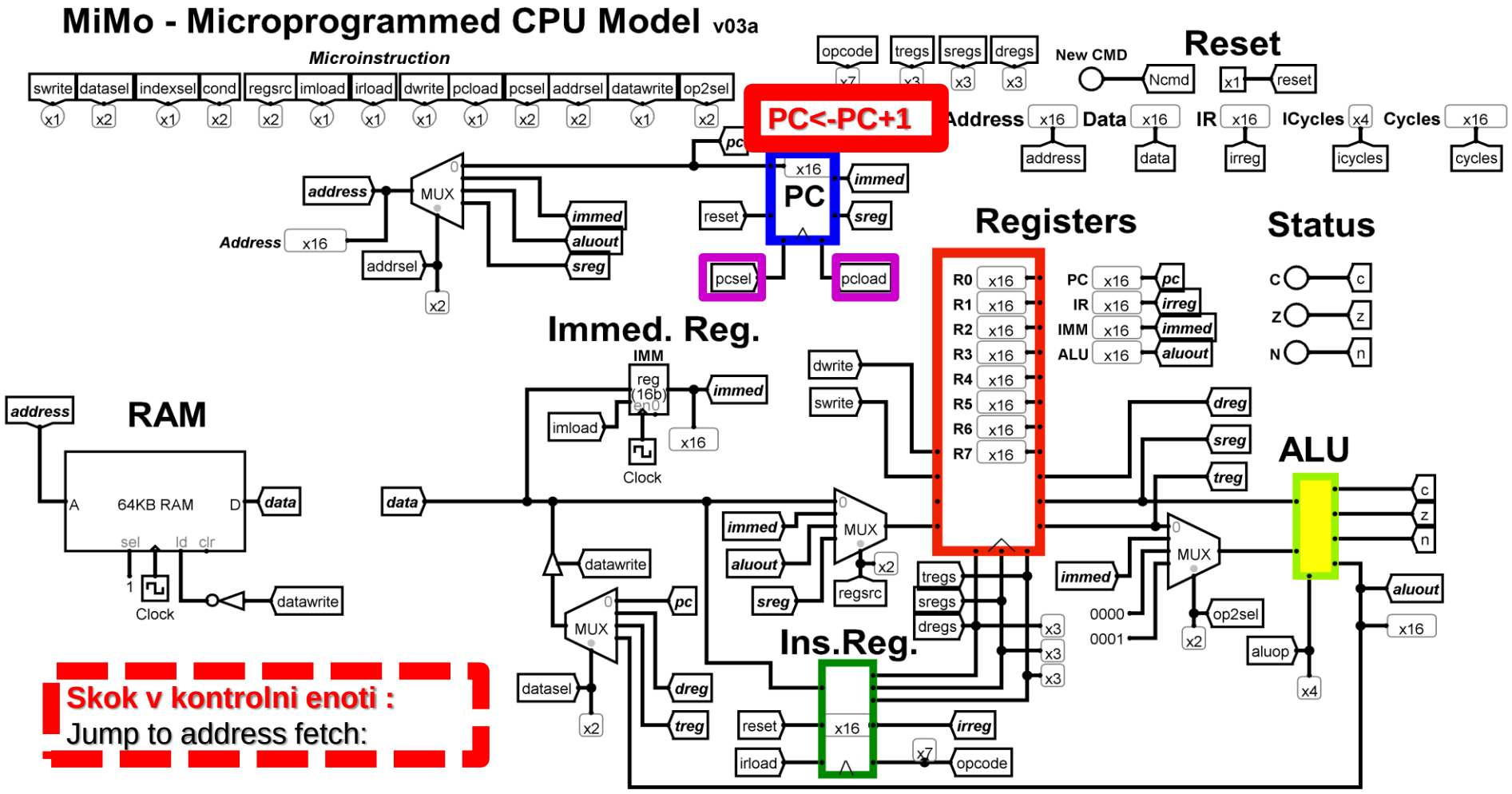
else jump

JNEZ Rs,immed: velja Rs=0

jnez Rs,immed (40)

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

```
fetch:      # Address=PC, Load IR register
            # PC=PC+1, jump to 2+OPC
40:         # Read Immediate operand -> IMRegister
            # ALU: Rs-0, If z then pcincr else jump
pcincr:     # Increment PC and goto new command;
jump:       # Set address to immed and goto new command
```

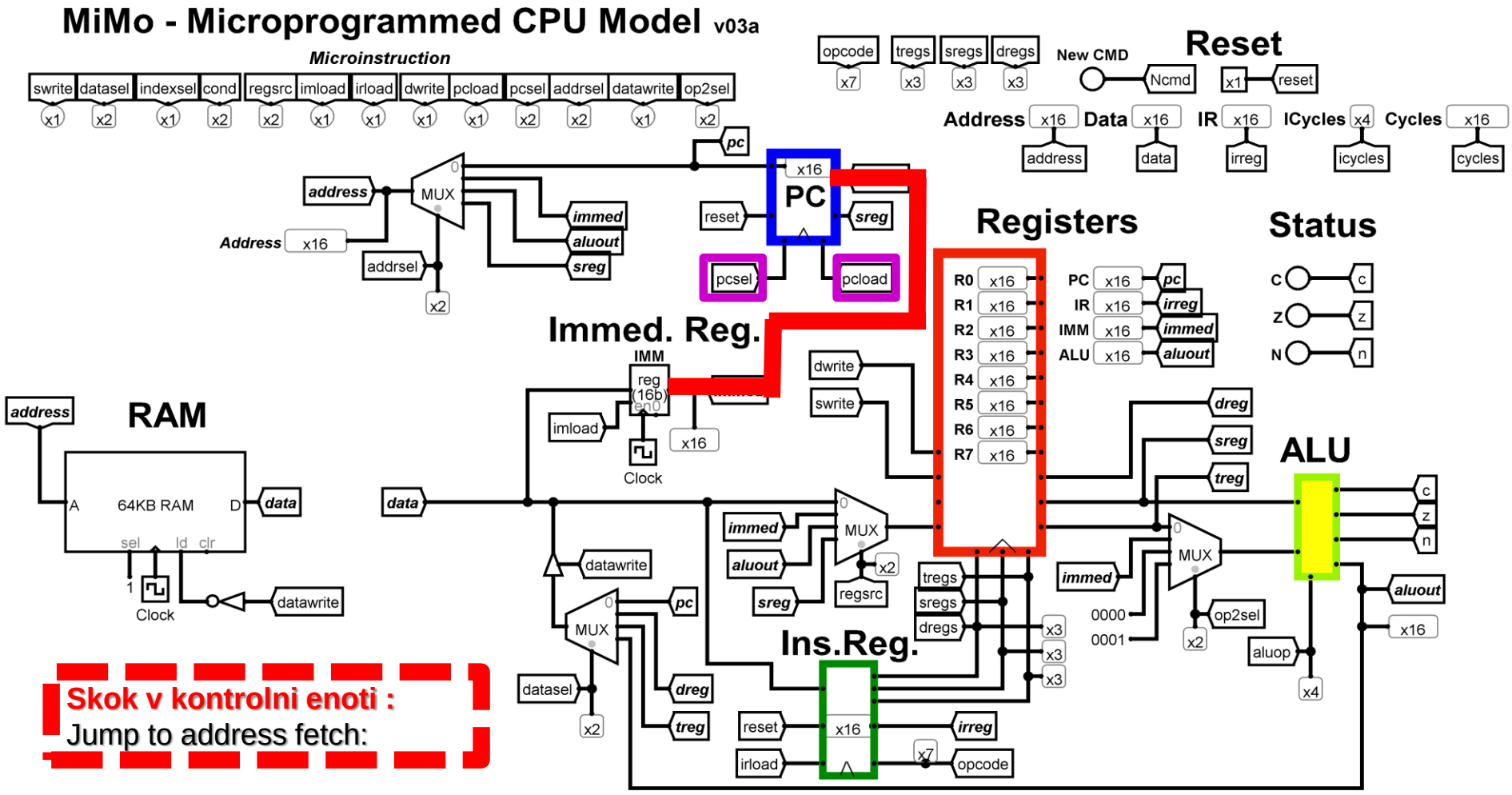


JNEZ Rs,immed: velja Rs≠0

jnez Rs,immed (40)

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

```
fetch:
                    # Address=PC, Load IR register
                    # PC=PC+1, jump to 2+OPC
40:
                    # Read Immediate operand -> IMRegister
                    # ALU: Rs-0, If z then pcincr else jump
pcincr:
                    # Increment PC and goto new command;
jump:               # Set address to immed and goto new command
```



Skok v kontrolni enoti :

Jump to address fetch:

JNEZ Rs,immed:

```
fetch:                  # Address=PC, Load IR register
                        # PC=PC+1, jump to 2+OPC
40:                     # Read Immediate operand -> IMRegister
                        # ALU: Rs-0, If z then pcincr else jump
pcincr:                 # Increment PC and goto new command;
jump:                   # Set address to immed and goto new command
```

Izvedba JNEZ

Monday, November 09, 2020 10:11 PM

Enota ?	KE	KE	ALE	ALE	IMM	IR	PC	PC	NASL	
JNEZ Rs, IMMED	INDEXSEL	COND	ALUOP	OP2SEL	IMLOAD	IRLOAD	PCLOAD	PCSEL	ADDRSEL	MicroPC
# Address=PC, Load IR register						1			PC (0)	0
# PC=PC+1, jump to 2+OP	1						1	"PC+1"		1
# Read Immediate operand -> IMRegister					1				PC(0)	42 (40+2)
# ALU: Rs-0, If z then pcincr else jump		Z	SUB (1)	CONST0						130
# Increment PC and goto new command;							1	"PC+1"		Z=1, 132
# Set address to immed and goto new command							1	"IMMED"		Z=0, 133

3.2.6 Primerjava Mikroprogramska/Trdoožičena KE

Mikroprogramska KE:

- **Počasnejša**
- **Enostavna, fleksibilna**
- **Možnost realizacije različnih arhitektur**

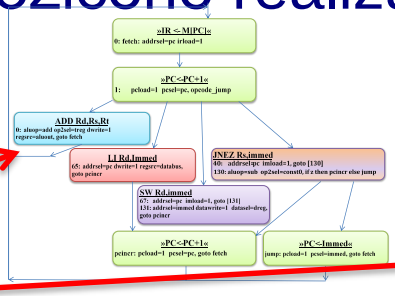
Trdoožičena KE :

- **Hitrejša**
- **Potrebnih več logičnih vezij**
- **Realizacijsko bolj zapletena**

Pristop k realizaciji trdožičene realizacije KE

Izhodišča:

- Diagram poteka
- Elementarni koraki
- Implementacija, izbrani primeri:



Elementarni koraki:

0 .. Branje ukaza -> IR

1 .. Format 2: branje operanda
Format 1: nop

2 .. Vse operacije :
- ALE, skok, reg.write, R/W from Mem

Realizacija el. korakov:

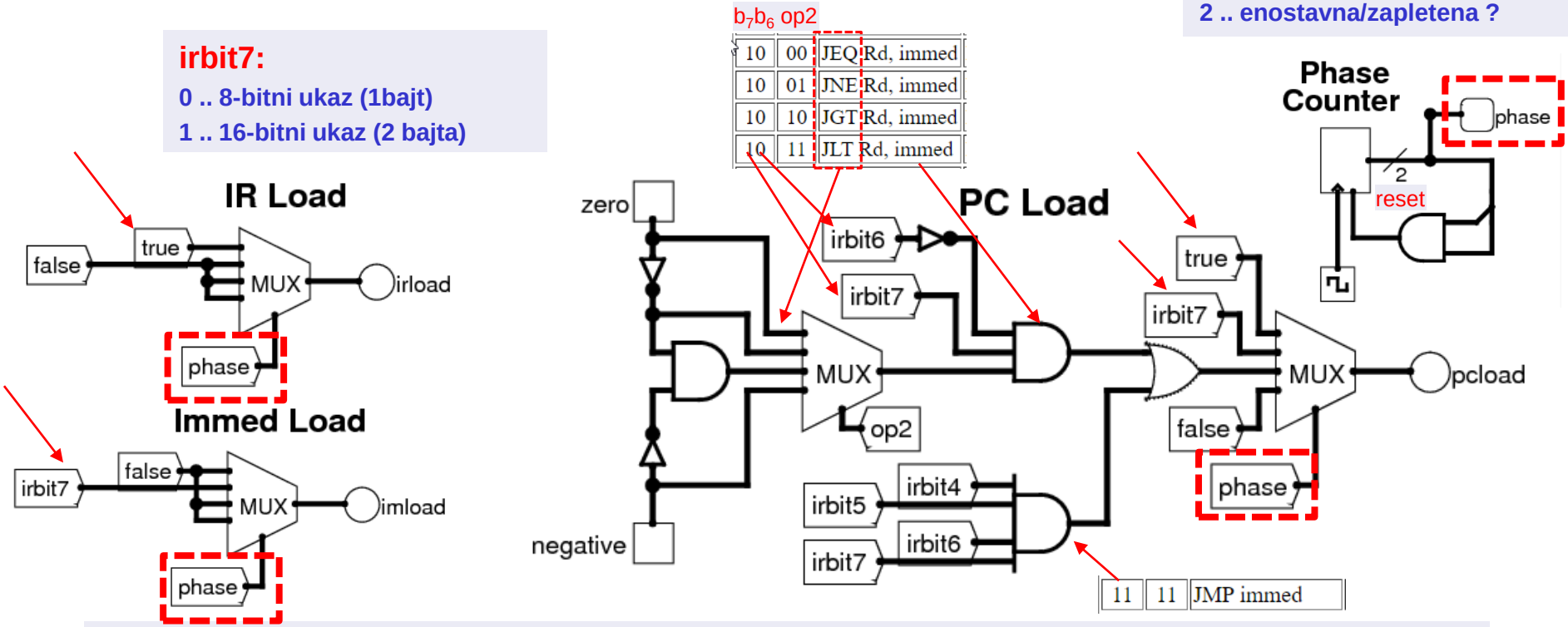
0,1 .. enostavna/zapletena ?

2 .. enostavna/zapletena ?

irbit7:

0 .. 8-bitni ukaz (1bajt)

1 .. 16-bitni ukaz (2 bajta)

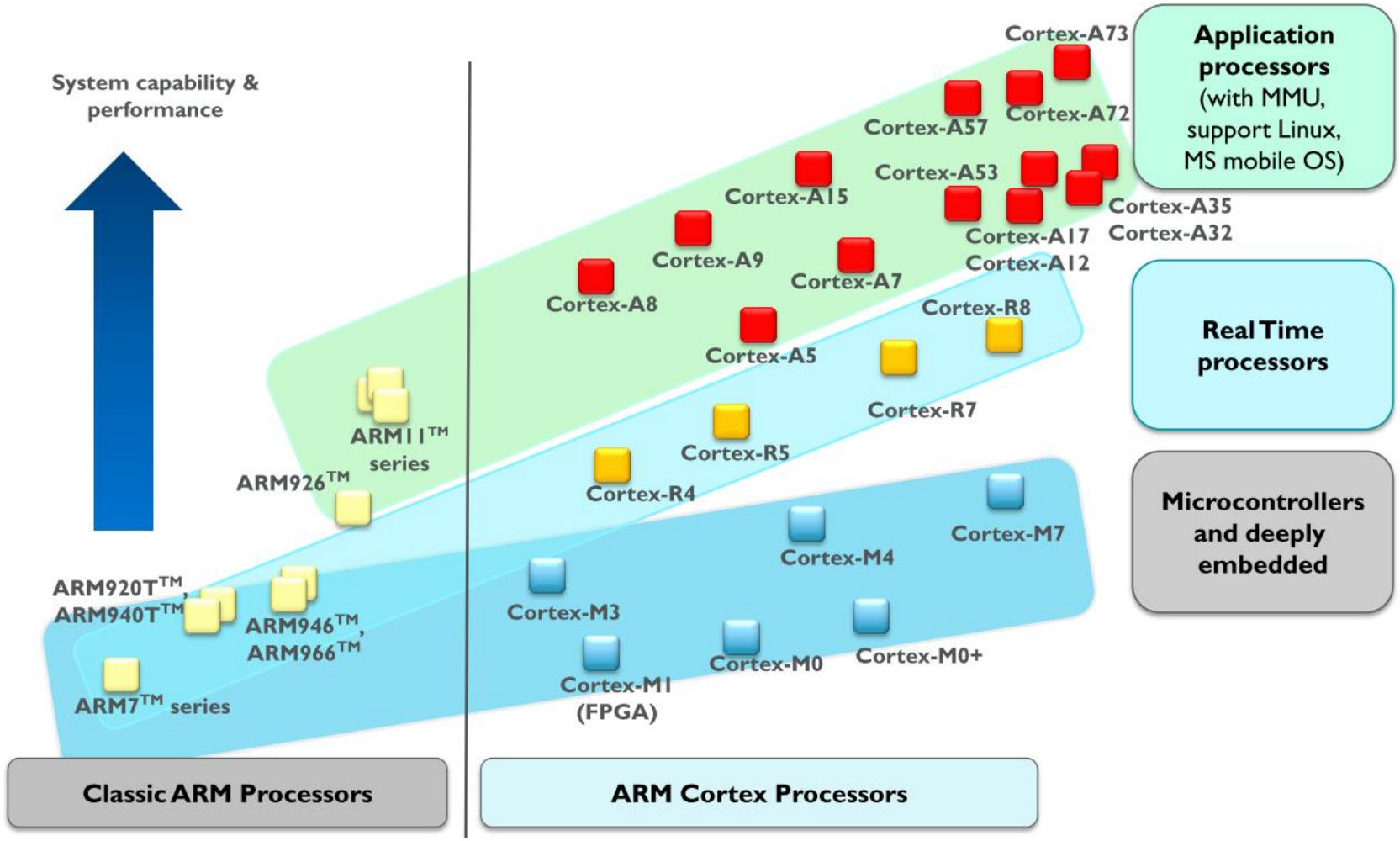


Podrobnejši opis - dodatno gradivo: http://minnie.tuhs.org/CompArch/Tutes/week03.html

3.3 Družina ARM procesorjev

3.3.1 Splošni pregled

*„**ARM**“ : **A**corn **R**ISC **M**achine, **A**dvanced **R**ISC **M**ACHINE*



3.3 Družina ARM procesorjev

3.3.1 Splošni pregled

*„**ARM**“ : **A**corn **R**ISC **M**achine, **A**dvanced **R**ISC **M**ACHINE*

Arhitektura	Oznake	Značilnosti	Opis
ARMv1			1985: nastane pod vplivom članka o RISC procesorjih (Berkeley)
ARMv2		prva komercialna	1986: 30000 tranzistorjev
ARMv3		32-bitno nasl. navidezni pomn.	
ARMv4	StrongARM, ARM7TDMI, ARM9TDMI	half-word load/store	
ARMv5	ARM7EJ, ARM9E, ARM10E, XScale	Thumb Enhanced DSP Jazelle	• Atmel 9260 (FRI-SMS) • podpora DSP operacijam • uvedba tretjega set ukazov (poleg ARM,Thumb) za pospešeno izvajanje Jave • podpora navideznem pomnilniku
ARMv6	ARM11, Cortex M	MMU Multiprocesiranje SIMD 6 novih status bitov	ARM11MP (prvi multiproc.) • Raspberry PI
ARMv7	Cortex A, M, R	NEON (MPE) VE-Virt.Ext. LPAE-Large.Ph.Addr.Ext. VFP-Vector Floating Point	Cortex A (Application Processor) : • A5, A8(MPE), A9(MPE,MP), • A15,A17 (MPE,VE,LPAE,VFPv4) Cortex M (MicroController) : • M3 (MPU, Bit Banding), M4 (M3 + DSP, FPU) • M7 (M4 + 64bit buses + 2xPower Eff. of M4,
ARMv8	Cortex A50 (A53, A57, A72, A73, A75, A76…)	64bitna	Microsoft Win8, Windows 10 Mobile, IoT Core

http://www.arm.com/products/processors/cortex-a

http://www.arm.com/products/processors/cortex-m

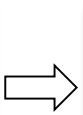
3.3.1 Splošni pregled ARM procesorjev

ARM (Advanced RISC Machine) = RISC? :

+ load/store arhitektura
+ cevovodna zgradba
+ reduciran nabor ukazov, vsi ukazi 32-bitni
+ ortogonalen registrski niz, vsi registri 32-bitni

```
a += (j << 2); se spremeni v 1 strojni ukaz:
ADD  Ra, Ra, Rj, LSL #2
```

```
while(i != j) {
if (i > j)
        i -= j;
else
        j -= i;
}
```



```
loop:   CMP  Ri, Rj
        SUBGT Ri, Ri, Rj    ; i = i-j;
        SUBLT Rj, Rj, Ri    ; j = j-i
        BNE  loop           ; if ( i != j )
```

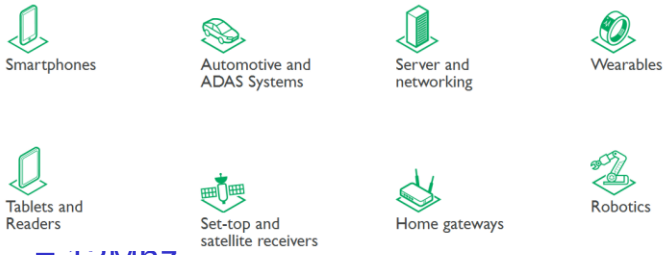
- hitri pomikalnik pred ALE
- pogojno izvajanje ukazov – ukaz se izvede le, če je stanje zastavic ustrezno.
- veliko načinov naslavljanja
- veliko formatov ukazov
- nekateri ukazi se izvajajo več kot en cikel (npr. *load/store multiple*) – obstaja nekaj kompleksnejših ukazov, kar omogoča manjšo velikost programov
- dodaten 16-bitni nabor ukazov Thumb omogoča krajše programe

3.3.2 ARM Cortex-A družina procesorjev

Namen : Kompleksnejše aplikacije, multimedija

NEON (advanced SIMD) ali MPE (»Media Processsing Engine«)

- 64 ali 128 bitni SIMD
- do 16 hkratnih (krajših) operacij
- v vseh Cortex A8, opcijsko tudi v A9
- video,audio in 3D grafika:
  - dekodira :
    - MP3 @ f_{cpe} = 10Mhz,AMR @ f_{cpe} = 13MHz,
    - MPEG4 VGA 30fr/sec @ f_{cpe} = 275 MHz
    - H.264 video @ fcpe = 350MHz



Cortex A8	Cortex A9	Cortex A15	Cortex A76
• **ARMv7 (32b)** • **NEON 64b** • **VFPv3** • **Jazelle RCT (Just in Time byte code apps) superskalarni (dual issue)** • **dual ALU pipeline**	• ARMv7 (32b) • MPU • out-of-order with speculative execution • bolj zmogljiv • večji L1, L2 PP • NEON 64b	• ARMv7 (32b) • NEON 128b • VFPv4	• ARMv8 (64b) • Napoved skokov • Out-of-order execution • 7nm !!!

http://www.arm.com/products/processors/cortex-a

https://en.wikipedia.org/wiki/ARM_Cortex-A

Announced (32-bit)

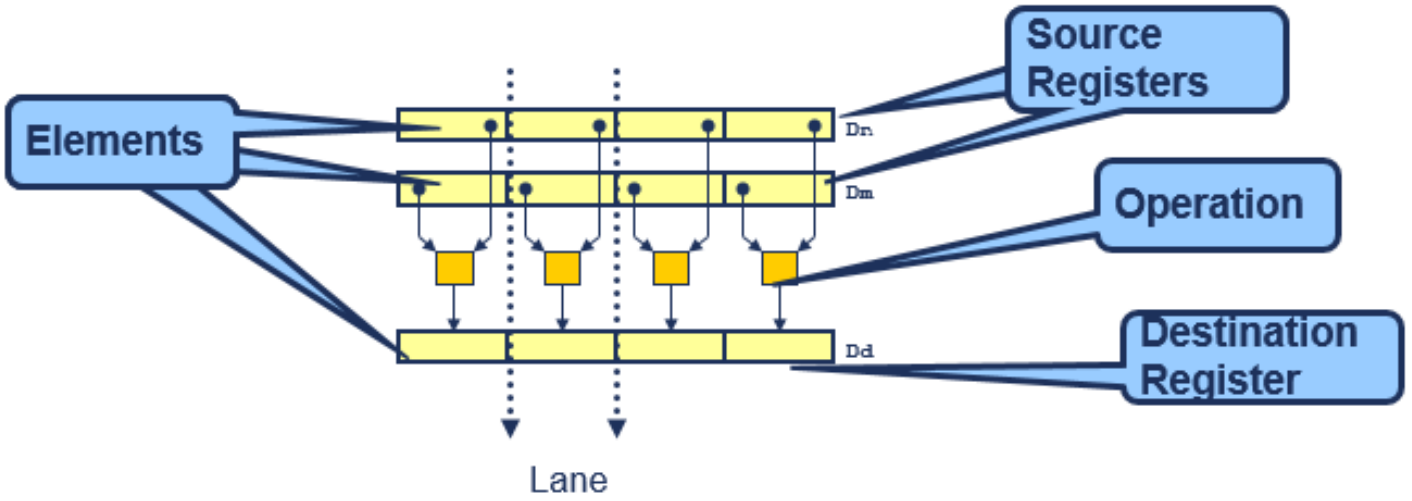
Year	Core
2005	Cortex-A8
2007	Cortex-A9
2009	Cortex-A5
2010	Cortex-A15
2011	Cortex-A7
2013	Cortex-A12
2014	Cortex-A17
2016	Cortex A32

Announced (64-bit)

Year	Core
2012	Cortex-A53
2012	Cortex-A57
2015	Cortex-A72
2015	Cortex-A35
2016	Cortex-A73
2017	Cortex-A75
2018	Cortex-A76
2019	Cortex-A77
2020	Cortex-A78
2021	Cortex-A510,710

What is NEON?

- **NEON is a wide SIMD data processing architecture**
  - Extension of the ARM instruction set (v7-A)
  - 32 x 64-bit wide registers (can also be used as 16 x 128-bit wide registers)
- **NEON instructions perform "Packed SIMD" processing**
  - Registers are considered as **vectors** of **elements** of the same data type
  - Data types available: signed/unsigned 8-bit, 16-bit, 32-bit, 64-bit, single prec. float
  - Instructions usually perform the same operation in all **lanes**



THE ARCHITECTURE FOR THE DIGITAL WORLD®

ARM

NEON vectorizing example

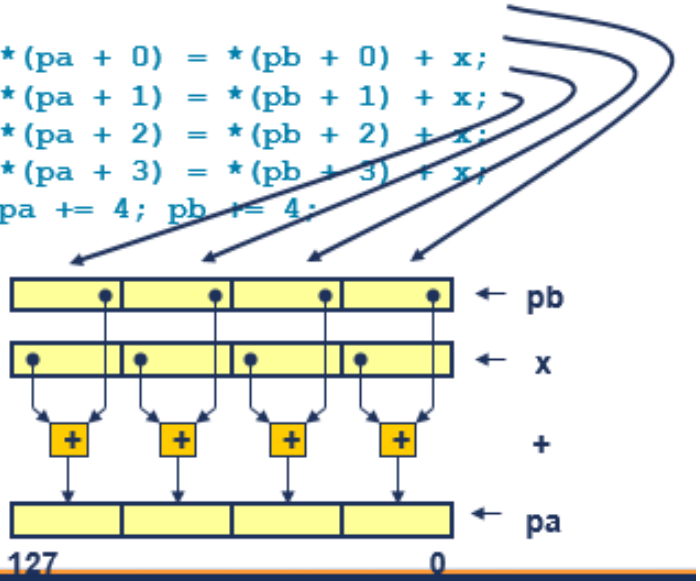
- **How does the compiler perform vectorization?**

```
void add_int(int * __restrict pa,
             int * __restrict pb,
             unsigned int n, int x)
{
  unsigned int i;
  for(i = 0; i < (n & ~3); i++)
    pa[i] = pb[i] + x;
}
```

1. Analyze each loop:
   - Are pointer accesses safe for vectorization?
   - What data types are being used? How do they map onto NEON vector registers?
   - Number of loop iterations

2. Unroll the loop to the appropriate number of iterations, and perform other transformations like pointerization

```
void add_int(int *pa, int *pb,
             unsigned n, int x)
{
  unsigned int i;
  for (i = ((n & ~3) >> 2); i; i--)
  {
    *(pa + 0) = *(pb + 0) + x;
    *(pa + 1) = *(pb + 1) + x;
    *(pa + 2) = *(pb + 2) + x;
    *(pa + 3) = *(pb + 3) + x;
    pa += 4; pb += 4;
  }
}
```

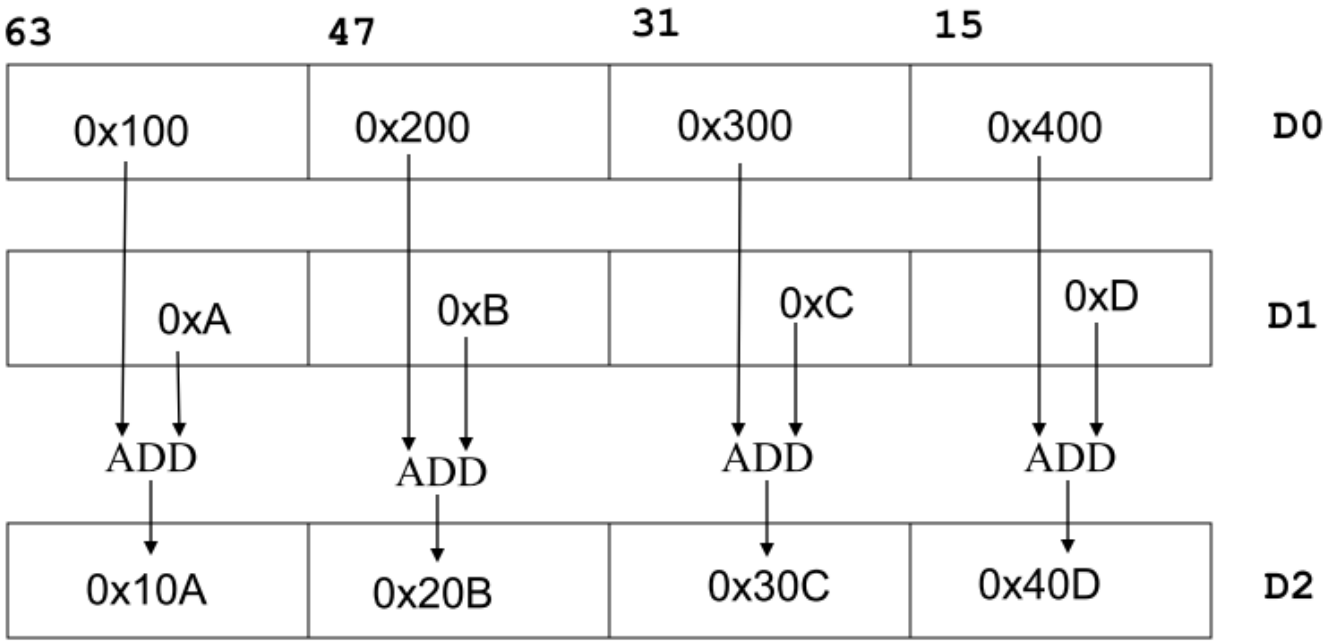


3. Map each unrolled operation onto a NEON vector lane, and generate corresponding NEON instructions

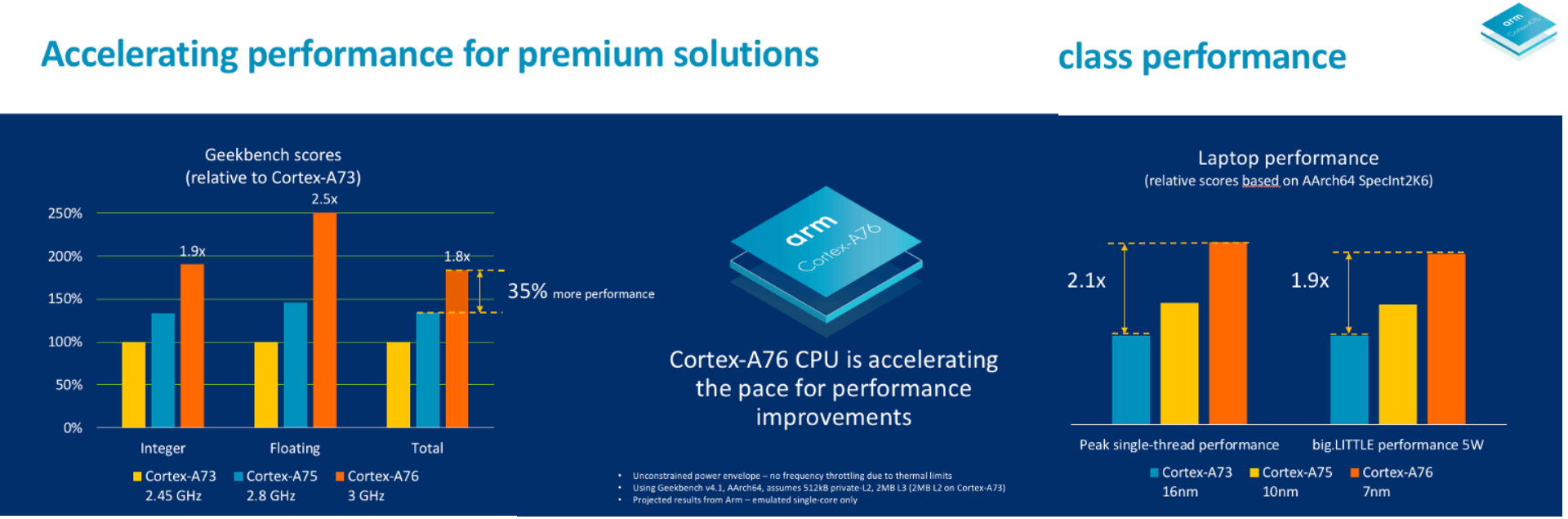
NEON – Advanced SIMD engine (ARM)

Example SIMD Instruction – Vector ADD

- Register split into equal size and type elements
- Same operation performed on each set of data
- VADD.U16 D2, D1, D0



3.3.2 ARM Cortex-A družina procesorjev



http://www.arm.com/products/processors/cortex-a

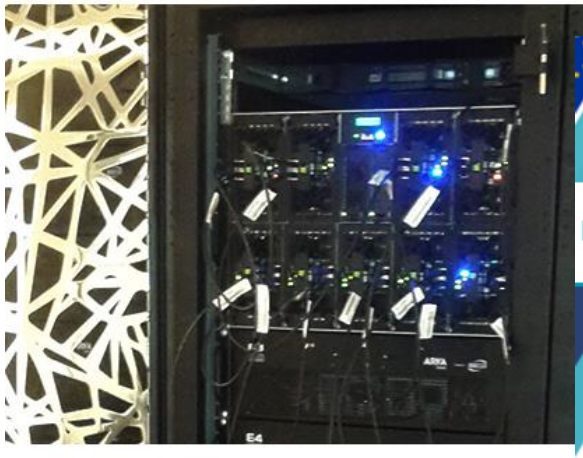
ARM in High Performance Computing (HPC)

http://www.montblanc-project.eu/home

MONT-BLANC

EUROPEAN APPROACH TOWARDS ENERGY EFFICIENT HIGH PERFORMANCE

Home Project Publications End-User Group Prototypes Applications Tools Press Corner Contact Search



MONT-BLANC

EUROPEAN MODULAR AND POWER-EFFICIENT HPC PROCESSOR

PROJECT EUROPEAN PROCESSOR PROTOTYPES APPLICATIONS SOFTWARE TOOLS

ABOUT MONT-BLANC

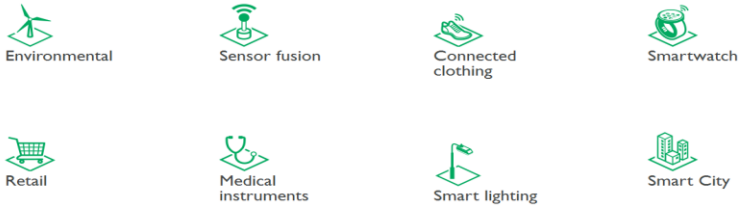
Compute efficiency and energy efficiency are more than ever major concerns for future Exascale systems.

Since October 2011, the aim of the European project called Mont-Blanc has been to design a new type of computer architecture capable of setting future global HPC standards, built from energy efficient solutions used in embedded and mobile devices. Phases 1 and 2 of the project are

Presentation	Deliverables
Project history	Discover the public reports published by the projects
Mont-Blanc 1 (2011-2015)	
Mont-Blanc 2 (2013-2016)	Mont-Blanc 2020
Mont-Blanc 3 (2015-2018)	Mont-Blanc 3 (2015-2018)
Mont-Blanc 2020 (2017-2020)	Mont-Blanc 2 (2013-2016)
Project data	Mont-Blanc (2011-2015)

3.3.3 ARM Cortex-M družina procesorjev

Namen : Mikrokrmilniški sistemi



Announced	
Year	Core
2004	Cortex-M3
2007	Cortex-M1
2009	Cortex-M0
2010	Cortex-M4
2012	Cortex-M0+
2014	Cortex-M7
2016	Cortex-M23,33
2018	Cortex-M35

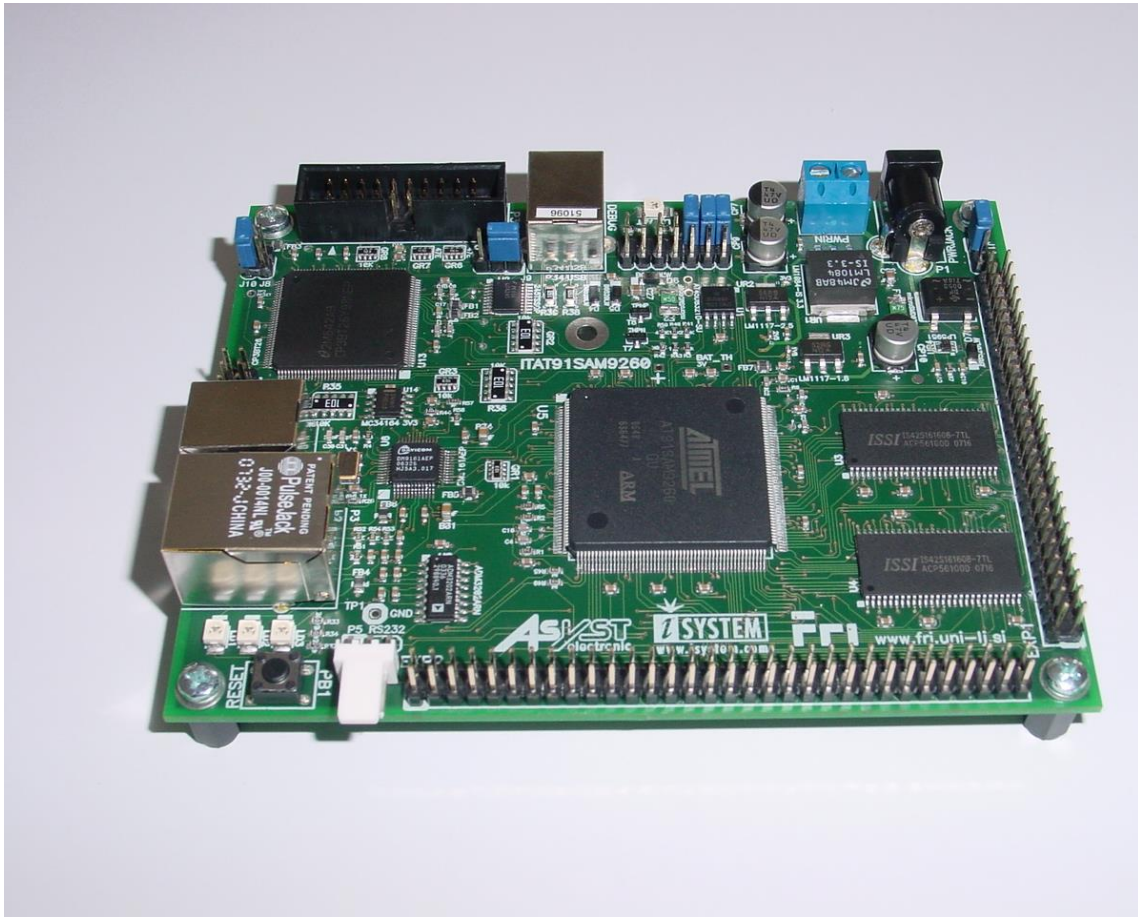
ARM Cortex-M optional components[6][7]

ARM Cortex-M	SysTick Timer	Bit-banding	Memory Protection Unit (MPU)	Tightly-Coupled Memory (TCM)	CPU cache	Memory architecture	ARM architecture
Cortex-M0[1]	Optional*	Optional[9]	No	No	No[10]	Von Neumann	ARMv6-M
Cortex-M0+[2]	Optional*	Optional[9]	Optional (8)	No	No	Von Neumann	ARMv6-M
Cortex-M1[3]	Optional	Optional	No	Optional	No	Von Neumann	ARMv6-M
Cortex-M3[4]	Yes	Optional*	Optional (8)	No	No	Harvard	ARMv7-M
Cortex-M4[5]	Yes	Optional*	Optional (8)	No	**Possible**[11]	Harvard	ARMv7E-M
Cortex-M7	Yes	No	Optional (8 or 16)	Optional	Optional	Harvard	ARMv7E-M

http://www.arm.com/products/processors/cortex-m

3.3.3 Mikroračunalniški sistem FRI-SMS

- Mikrokrmilnik Atmel SAM 9260
- Procesorska plošča z osnovnim naborom V/I naprav (Ethernet, vmesnik za SD/MMC kartico, USB, RS232)
- 64MB SDRAM delovnega pomnilnika
- 4 MB ROM NOR flash pomnilnika za OS
- ~~Razširitveno vezje z bogatim naborom dodatnih V/I naprav (še v fazi prototipa)~~



Značilnosti:

- Oznaka ARM926EJ-S
- 2 ločena predpomnilnika:
  - 8Kb operandni PP
  - 8Kb ukazni PP
- 200 MIPS at 180 MHz
- notranja pomnilnika :
  - 2x4KB internal RAM
  - 32KB internal ROM
- MMU enota
- USB,ETH,ADC,MMC,UART,SPI,TWI

3.3.4 Raspberry Pi - RPi

Raspberry Pi 1 model B+	
Release date	February 2012; 3 years ago
Introductory price	US$25 (model A, B+[1]), US$20 (model A+), US$35 (RPi 1 model B, RPi 2 model B), US$30 (CM)
Operating system	Linux (e.g. Raspbian), RISC OS, FreeBSD, NetBSD, Plan 9, Inferno, AROS
CPU	700 MHz single-core ARM1176JZF-S (model A, A+, B, B+, CM)[2]
Memory	256 MB[3] (model A, A+, B rev 1) 512 MB (model B rev 2, B+, CM)
Storage	SDHC slot (model A and B), MicroSDHC slot (model A+ and B+), 4 GB eMMC IC chip (model CM)
Graphics	Broadcom VideoCore IV[2]
Power	1.5 W (model A), 1.0 W (model A+), 3.5 W (model B) or 3.0 W (model B+)

Raspberry Pi 1 model B+	
Release date	February 2012; 3 years ago

Micro architecture improvements in ARM11 cores include:

- SIMD instructions which can double MPEG-4 and audio digital signal processing algorithm speed
- Cache is physically addressed, solving many cache aliasing problems and reducing context switch overhead
- Unaligned and mixed-endian data access is supported
- Reduced heat production and lower overheating risk
- Redesigned pipeline, supporting faster clock speeds (target up to 1 GHz)
  - Longer: 8 (vs 5) stages
  - Out-of-order completion for some operations (e.g. stores)
  - Dynamic branch prediction/folding (like XScale)
  - Cache misses don't block execution of non-dependent instructions
  - Load/store parallelism
  - ALU parallelism
- 64-bit data paths

Vir: Wikipedia.com

3.3.4 Raspberry Pi 2- RPi2

Raspberry Pi 2 model B	
Release date	February 2015; 9 months ago
Introductory price	US$35
Operating system	Same as for Raspberry Pi 1 plus Windows 10 IoT Core[4] and additional distributions of Linux such as Ubuntu
CPU	900 MHz quad-core ARM Cortex-A7
Memory	1 GB RAM
Storage	MicroSDHC slot
Graphics	Broadcom VideoCore IV
Power	4.0 W

In early February 2015, the next-generation Raspberry Pi, Raspberry Pi 2, was released.[20]

The new computer board is initially available only in one configuration (model B) and features

- a Broadcom BCM2836 SoC, with a quad-core ARM Cortex-A7 CPU and
- a VideoCore IV dual-core GPU;
- 1 GB of RAM
- with remaining specifications being similar to those of the previous generation model B+.

The Raspberry Pi 2 retains the same US$35 price point of the model B,[21] with the US$20 model A+ remaining on sale.

Vir: Wikipedia.com

3.3.4 Raspberry Pi 3

Raspberry Pi 3 model B

Release date	29 February 2016;
Introductory price	US$35
Operating system	Same as for Raspberry Pi 1 plus Windows 10 IoT Core[4] and additional distributions of Linux such as Ubuntu
CPU	Broadcom BCM2837 1.2 GHz 64/32-bit quad-core ARM Cortex-A53
Memory	1 GB LPDDR2 RAM at 900 MHz
Storage	MicroSDHC slot
Graphics	Broadcom VideoCore IV at higher clock frequencies (300 MHz & 400 MHz)
Power	800 mA (4.0 W)

The Raspberry Pi 3 is the third generation Raspberry Pi. It replaced the Raspberry Pi 2 Model B in February 2016. Compared to the Raspberry Pi 2 it has:

- A **1.2GHz 64-bit quad-core ARMv8 CPU**
- 802.11n Wireless LAN
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)

The Raspberry Pi 3 has an identical form factor to the previous Pi 2 (and Pi 1 Model B+) and has complete compatibility with Raspberry Pi 1 and 2.

Raspberry Pi 3 Model B released in February 2016 is bundled with on-

$20-

Vir: Wikipedia.com

3.3.4 Raspberry Pi 4

Raspberry Pi 4 Model B	
Release date	24 June 2019; 16 months ago (**Current**)
Introductory price	•US$35 (Pi 4 2 GiB)[1]
System on a chip	Broadcom BCM2711B0[5]
CPU	**Pi 4 B**: 1.5 GHz quad-core A72 64-bit[5]
Memory	•**Pi 4 B**: 2, 4, or 8 GiB LPDDR4-3200 SDRAM[7][3]
Storage	MicroSDHC slot
Graphics	Broadcom VideoCore VI 500 MHz[10]
Power	5 V; 3 A (for full power delivery to USB devices)[12]

Raspberry Pi 4 Model B was released in June 2019[2] with a

- 1.5 GHz 64-bit quad core ARM Cortex-A72 processor,
- on-board 802.11ac Wi-Fi, Bluetooth 5,
- full gigabit Ethernet (throughput not limited),
- two USB 2.0 ports,
- two USB 3.0 ports, and
- dual-monitor support via a pair of micro HDMI (HDMI Type D) ports for up to 4K resolution.

The Pi 4 is also powered via a USB-C port, enabling additional power to be provided to downstream peripherals, when used with an appropriate PSU.

Vir: Wikipedia.com

3.3.4 Raspberry Pi

Family	Model	Form Factor	Ethernet	Wireless	GPIO	Released	Discontinued
Raspberry Pi	B	Standard[a]	Yes	No	26-pin	2012	Yes
	A		No			2013	No
	B+		Yes		40-pin	2014	
	A+	Compact[b]	No			2014	
Raspberry Pi 2	B	Standard[a]	Yes	No		2015	
Raspberry Pi Zero	Zero	Zero[c]	No	No		2015	
	W/WH			Yes		2017	
Raspberry Pi 3	B	Standard[a]	Yes	Yes		2016	
	A+	Compact[b]	No			2018	
	B+	Standard[a]	Yes			2018	
Raspberry Pi 4	B (1 GiB)	Standard[a]	Yes (Gigabit Ethernet)	Yes		2019[32]	Yes[1]
	B (2 GiB)						
	B (4 GiB)						
	B (8 GiB)					2020	

Vir: Wikipedia.com

3.3.5 ST Discovery F4

STM Discovery F4 (Cortex M4)

•STM32F407VGT6 microcontroller featuring 32-bit Arm® Cortex®-M4 with FPU core, 1-Mbyte Flash memory and 192-Kbyte RAM in an LQFP100 package

•USB OTG FS

•ST MEMS 3-axis accelerometer

•ST-MEMS audio sensor omni-directional digital microphone

•Audio DAC with integrated class D speaker driver

•User and reset push-buttons

•Eight LEDs:

- •LD1 (red/green) for USB communication
- •LD2 (red) for 3.3 V power on
- •Four user LEDs, LD3 (orange), LD4 (green), LD5 (red) and LD6 (blue)

•Board connectors:

- •USB with Micro-AB
- •Stereo headphone output jack
- •2.54 mm pitch extension header for all LQFP100 I/Os for quick connection to prototyping board and easy probing

•External application power supply: 3 V and 5 V

•

https://www.st.com/en/evaluation-tools/stm32f4discovery.html

3.3.5 ST Discovery F7

STM Discovery F7 (Cortex M7)

•STM32F769NIH6 microcontroller featuring 2 Mbytes of Flash memory and 512+16+4 Kbytes of RAM, in BGA216 package

•On-board ST-LINK/V2-1 supporting USB reenumeration capability

•USB ST-LINK functions: virtual COM port, mass storage, debug port

•4" capacitive touch LCD display with MIPI® DSI connector (on STM32F769I-DISCO only)

•SAI audio codec

•Two audio line jacks, one for input and one for output

•Stereo speaker outputs

•Four ST MEMS microphones on DFSDM inputs

•Two SPDIF RCA input and output connectors

•Two push-buttons (user and reset)

•512-Mbit Quad-SPI Flash memory

•128-Mbit SDRAM

•Connector for microSD card

•Wi-Fi or Ext-EEP daughterboard connector

•USB OTG HS with Micro-AB connector

•Ethernet connector compliant with IEEE-802.3-2002

•Arduino™Uno V3 connectors

aws qualified device Internet of things

arm MBED Enabled

https://www.st.com/en/evaluation-tools/32f769idiscovery.html

3.3.5 ST Discovery STM32MP157C

STM Discovery MP1 (2xCortex A7 + 1xCortex M4)

•STM32MP157 Arm®-based dual Cortex®-A7 32 bits + Cortex®-M4 32 bits MPU in TFBGA361 package

•4-Gbit DDR3L, 16 bits, 533 MHz

•1-Gbps Ethernet (RGMII) compliant with IEEE-802.3ab

•USB OTG HS

•Audio codec

•4 user LEDs

•2 user and reset push-buttons, 1 wake-up button

•5 V / 3 A USB Type-CTM power supply input (not provided)

•Board connectors:Ethernet RJ454 × USB Host Type-AUSB Type-CTM DRPMIPI DSISMHDMI®Stereo headset jack including analog microphone inputmicroSDTM cardGPIO expansion connector (Raspberry Pi® shields capability)

•ARDUINO® Uno V3 expansion connectors

•STM32CubeMP1 and full mainline open-source Linux® STM32 MPU OpenSTLinux Distribution (such as STM32MP1Starter) software and examples

•4" TFT 480×800 pixels with LED backlight, MIPI DSISM interface, and capacitive touch panel

•Wi-Fi® 802.11b/g/n

•Bluetooth® Low Energy 4.1

STM32MP1

https://www.st.com/en/evaluation-tools/stm32mp157c-dk2.html

3.4 RISC-V (https://riscv.org/)

RISC-V: The Free and Open RISC Instruction Set Architecture

RISC-V is a free and open ISA enabling a new era of processor innovation through open standard collaboration. Born in academia and research, RISC-V ISA delivers a new level of free, extensible software and hardware freedom on architecture, paving the way for the next 50 years of computing design and innovation.