



ORGANIZACIJA RAČUNALNIKOV

Povzetki predavanj

3. Mikroarhitekturni nivo računalnika

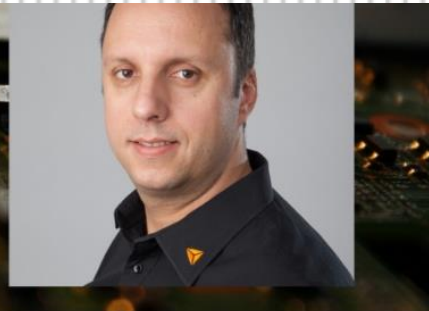
(3.2 MiMo Model Mikroprogramirane CPE)

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

NOVICA

objavljeno
25. oktober 2017

#vblatu



■ Za boljše rezultate se spustite na nižji nivo programiranja

Novice

Novice

Želite izboljšati delovanje svojih programov? Če je odgovor da, ne smete zamuditi optimizacije kode »Code optimization on modern processors« v okviru serije Če v 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 optimiziranja kode, in kdaj nas čaka na predavanju.

**„... tudi 64-kratna
pohitritev po
optimizaciji kode...“**

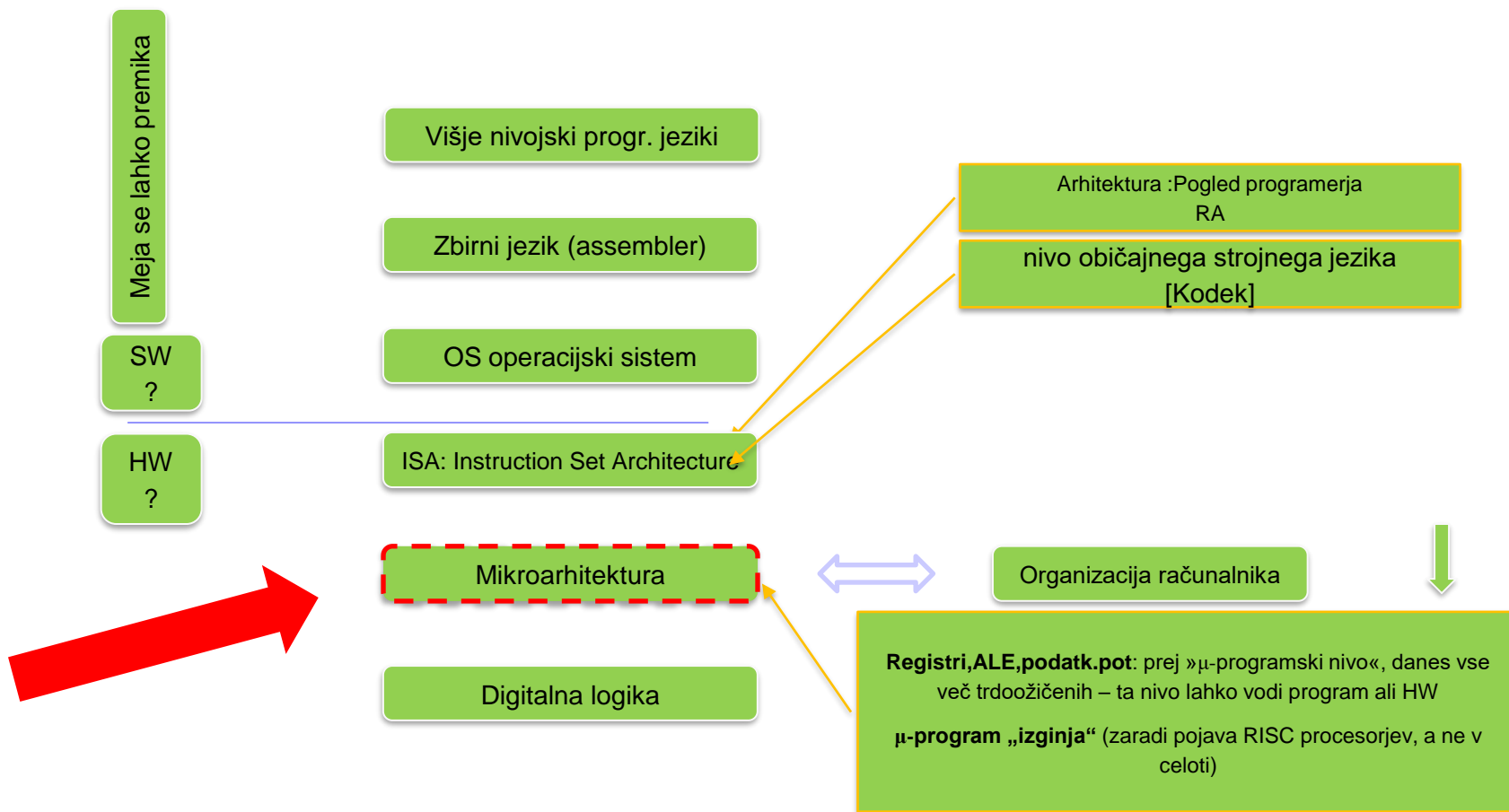


Namen in cilji 3. poglavja:

Razumevanje :

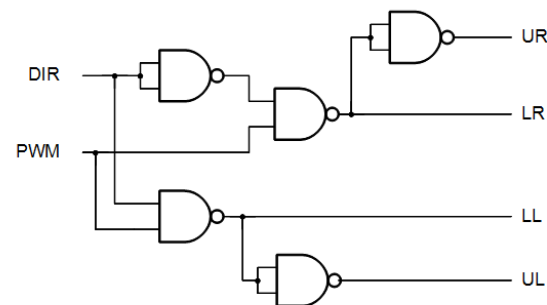
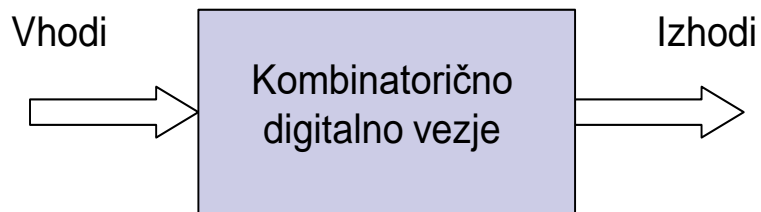
- pomembnost **urinega signala, sinhronskosti** v digitalnih vezjih
- delovanja CPE in njenih sestavnih delov skozi **praktični primer**:
 - od nivoja logičnih vrat do delujočega modela CPE
 - **MiMo model mikroprogramske CPE** (večperiodni, cevovodni)
 - realizacija/izvedba strojnih ukazov z zaporedji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

3. Mikro-arhitekturni nivo računalnika

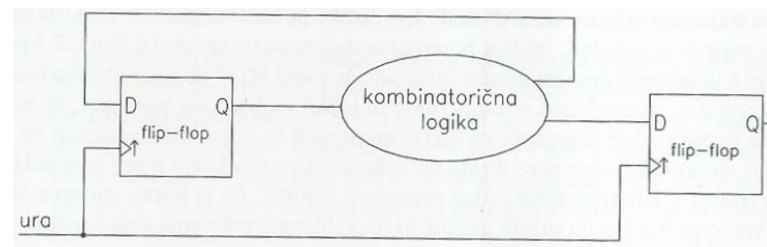
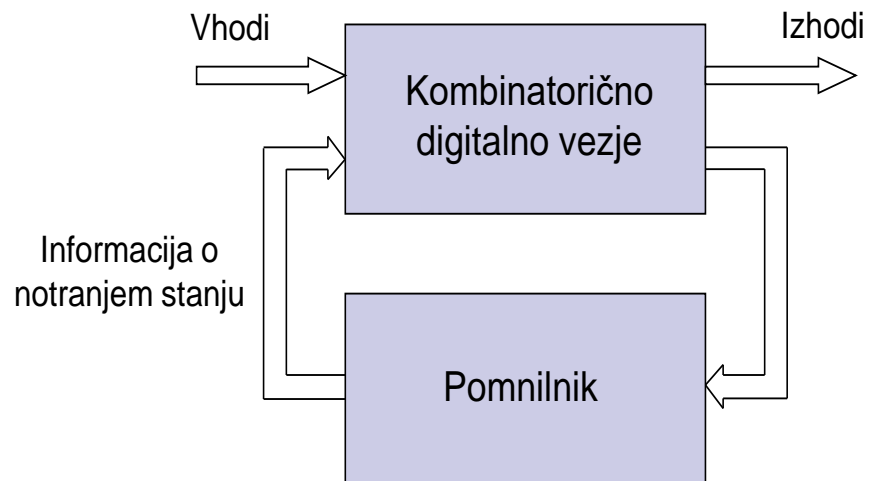


3. Mikroarhitekturni nivo računalnika

Kombinatorična logična vezja:

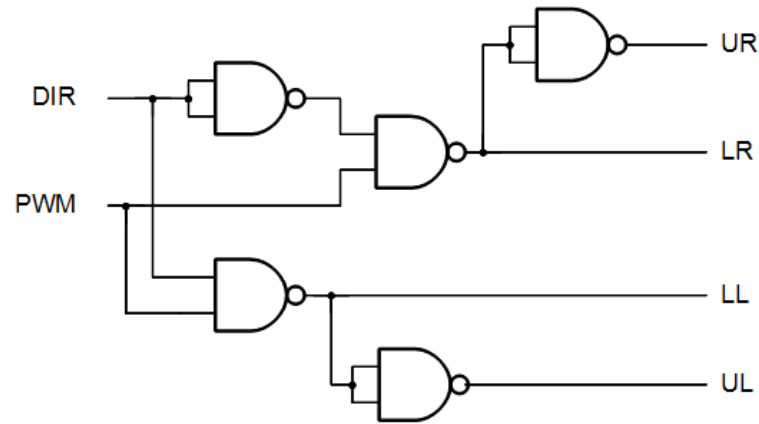


Sekvenčna logična vezja:



3. Mikroarhitekturni nivo računalnika

Asinhrona digitalna vezja :



Slabosti:

- različne zakasnitve
- zapleteno reševanje problema razl. zakasnitev

Prednosti:

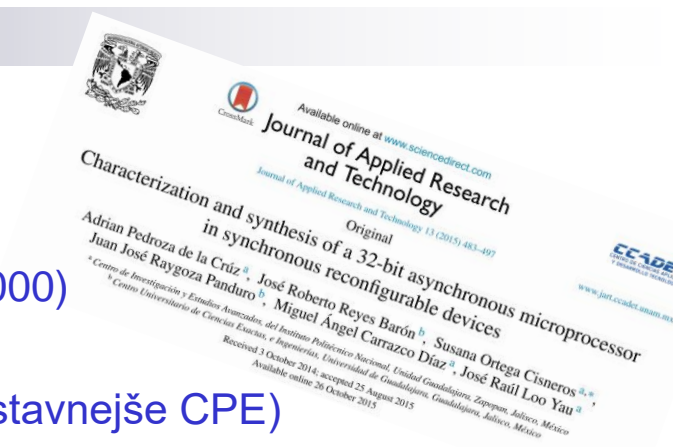
- hitrost
- poraba

3. Mikroarhitekturni nivo računalnika

Asinhronska digitalna vezja :

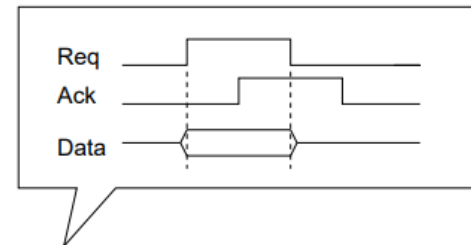
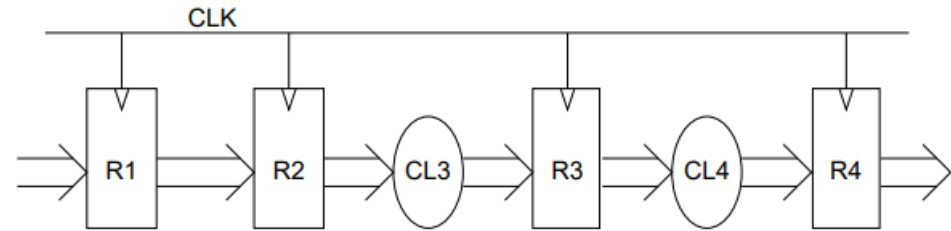
Primer: MiniMIPS (asinhronska realizacija CPU MIPS R3000)

- 4-krat hitrejša od sinhronske realizacije
- bistveno bolj kompleksna (lahko naredimo le enostavnejše CPE)

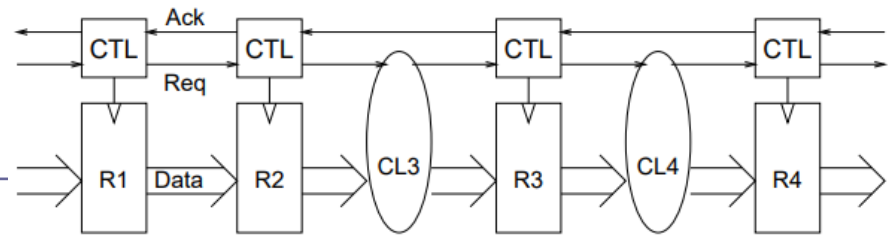


Prikaz razlik med sinhronsko in asinhronsko

■ Sinhronska



■ Asinhronska



**Computing without Clocks:
Micropipelining the ARM Processor**

Steve Furber

**The Design of an Asynchronous
MIPS R3000 Microprocessor**

Alain J. Martin, Andrew Lines, Rajit Manohar, Mika Nyström
Paul Penzes, Robert Southworth, Uri Cummings, Tak Kwan Lee
Department of Computer Science
California Institute of Technology
Pasadena CA 91125, USA

3. Mikroarhitekturni nivo računalnika

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.

Asynchronous circuits are fundamentally different; they also assume binary signals, *but there is no common and discrete time*. Instead the circuits use handshaking between their components in order to perform the necessary synchronization, communication, and sequencing of operations. Expressed in 'synchronous terms' this results in a behaviour that is similar to systematic fine-grain clock gating and local clocks that are not in phase and whose period is determined by actual circuit delays – registers are only clocked where and when needed.

Asynchronous circuit design

A Tutorial

Jens Sparsø

Technical University of Denmark

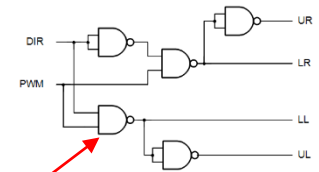
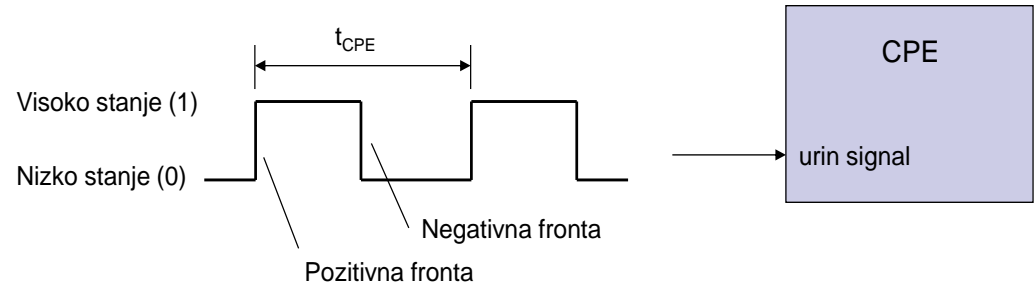
3.1 Sinhronska digitalna vezja

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

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

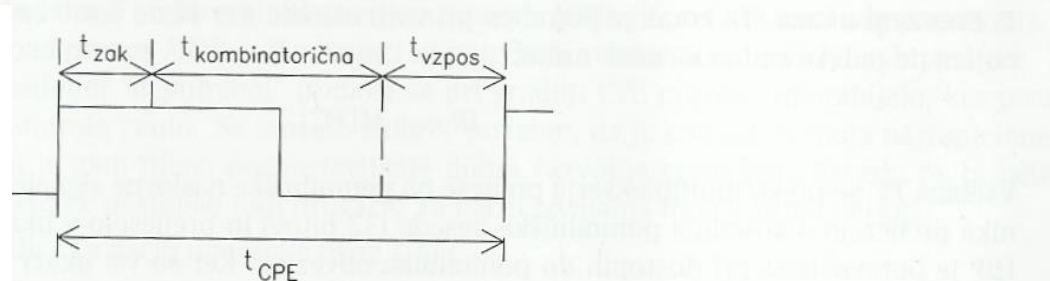
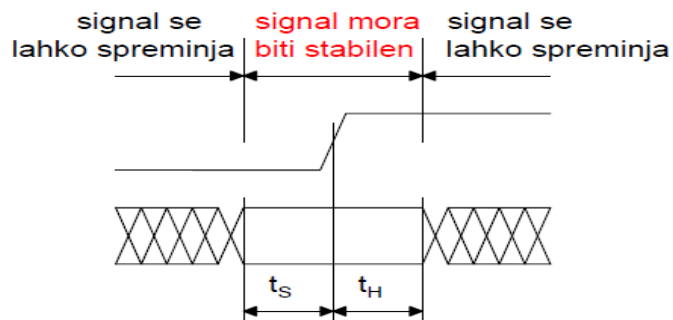
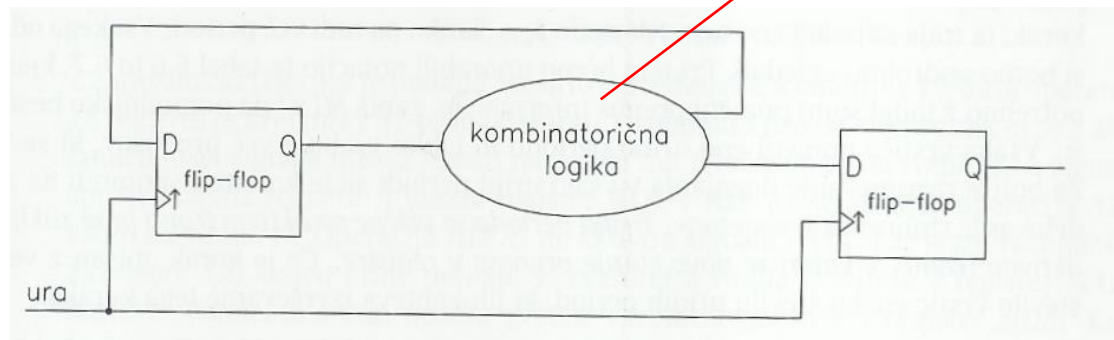
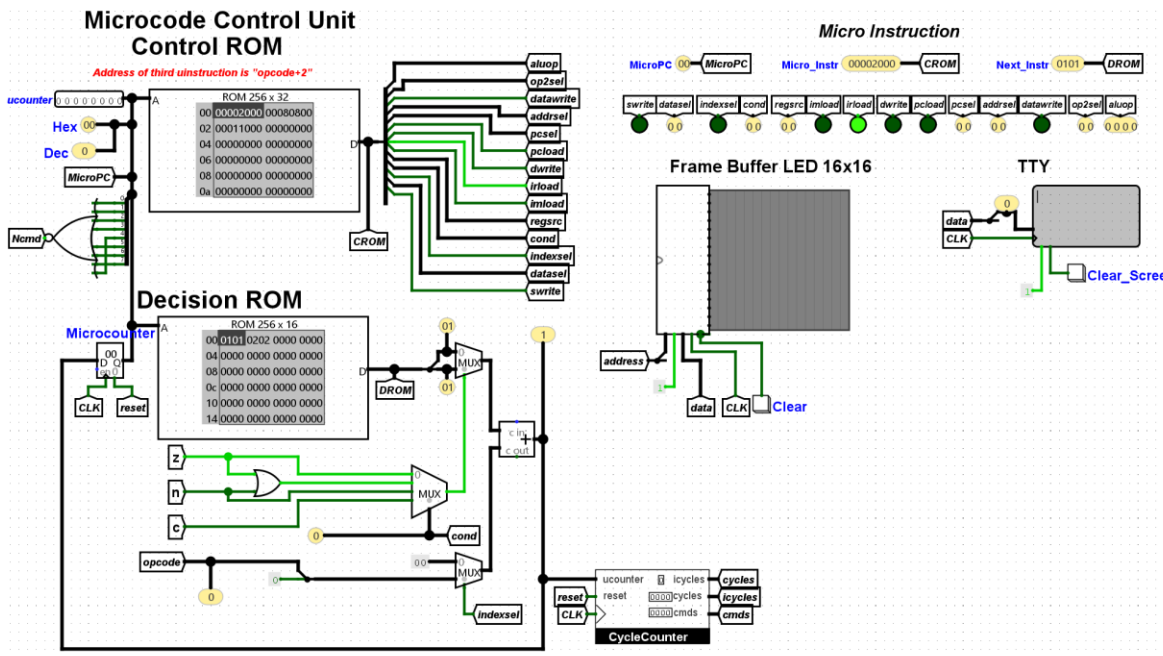
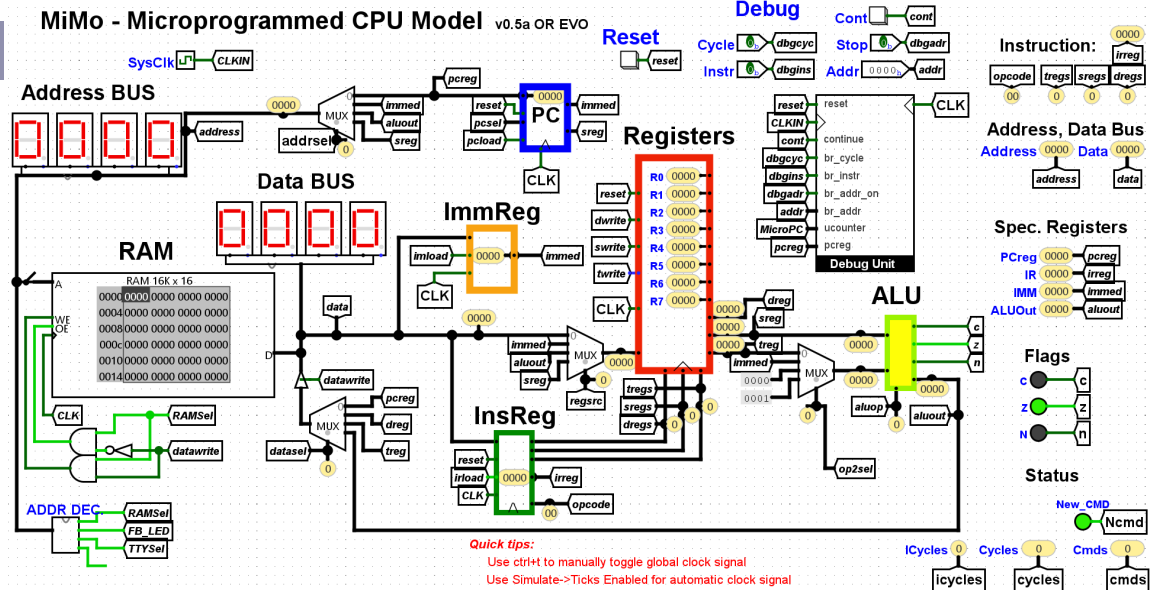


Figure 1.27: Vzpostavitveni (t_S) in držalni čas (t_H).

3.2 MiMo – Mikroprogramiran Model CPE



Based on: <http://minnie.tuhs.org/Programs/UcodeCPU/index.html>
 v05: Migration to EVO, Debug, Counters, ImmReg Units
 v05a: norz instead of corz

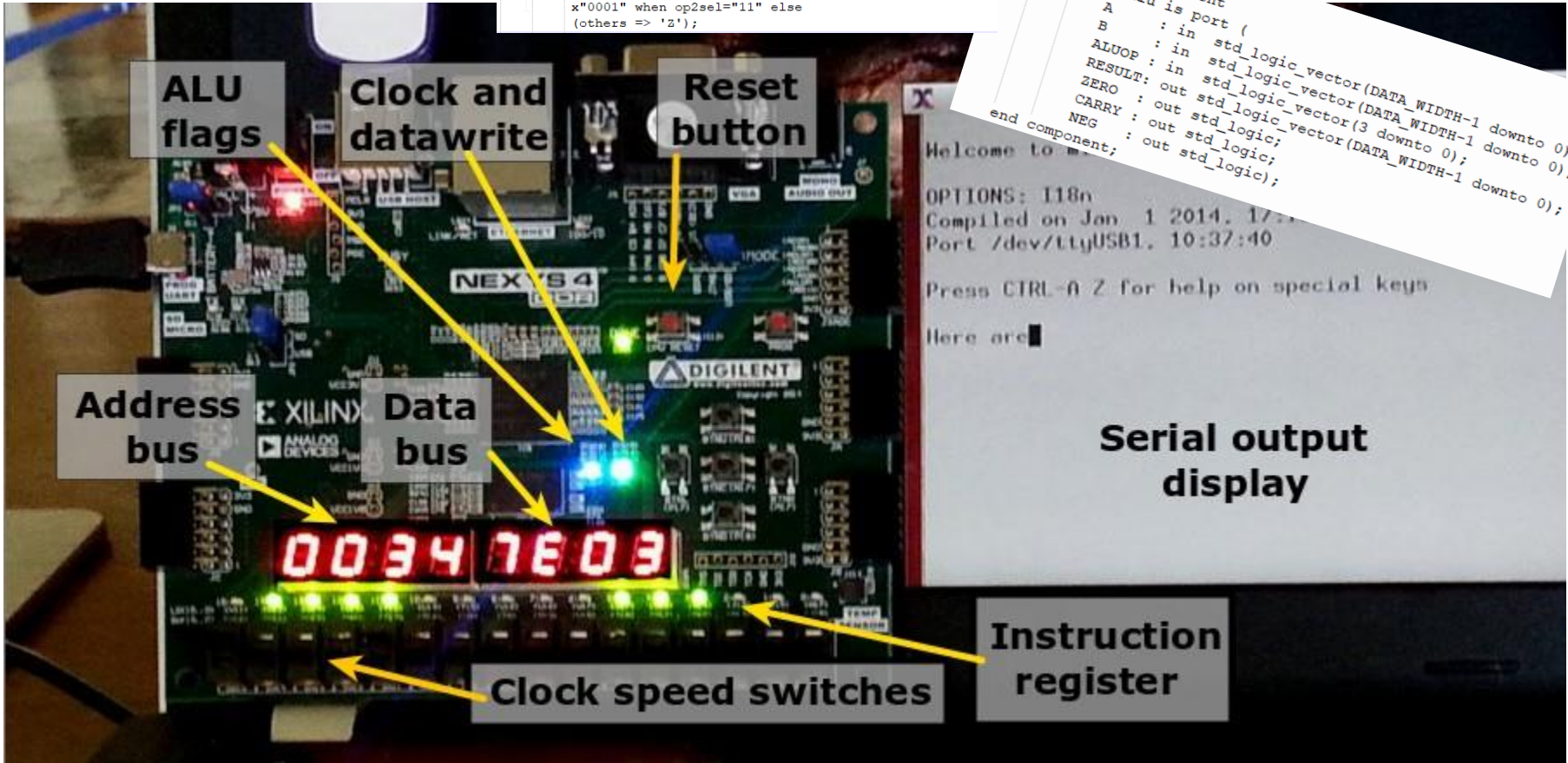
3.2 MiMO – Mikro-programiran Model CPE FPGA realizacija

<https://web.microsoftstream.com/video/590d652e-717f-4e91-9fa5-fe537f55ae7b>

```
-- 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');
```

```
-- 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 (
  B : in std_logic_vector (DATA_WIDTH-1 downto 0);
  ALUOP : in std_logic_vector (DATA_WIDTH-1 downto 0);
  RESULT: out std_logic_vector (DATA_WIDTH-1 downto 0);
  ZERO : out std_logic_vector (3 downto 0);
  CARRY : out std_logic;
  NEG : out std_logic;
end component;
  );
```



```
Welcome to
OPTIONS: 118n
Compiled on Jan 1 2014, 17:
Port /dev/ttyUSB1. 10:37:40

Press CTRL-A Z for help on special keys

Here are █
```

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

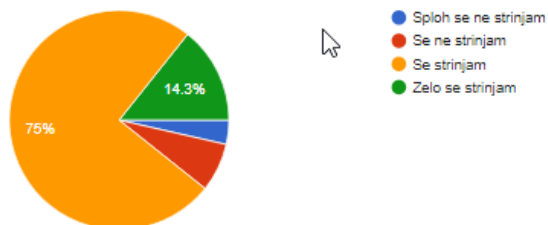
3.2 MiMo – Mikro-programiran Model CPE

Mnenja (19/20)

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 ...

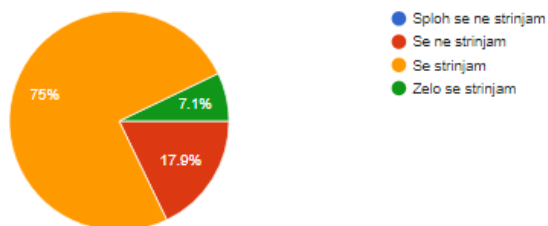
Izvedba MiMo modela na nivoju logičnih vrat v Logisimu je bila zanimiva in koristna

28 responses



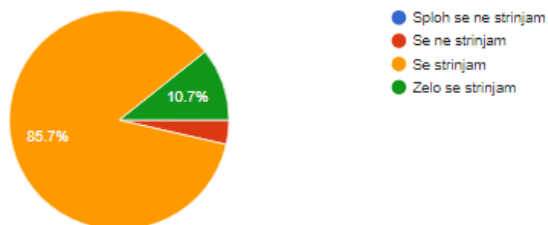
Izvedba MiMo modela na nivoju mikro zbirnika (micro-assembler) je bila zanimiva in koristna

28 responses



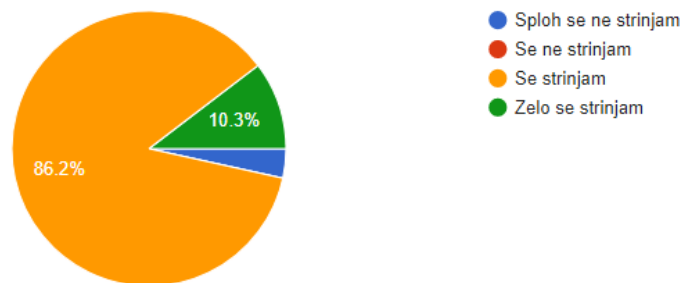
Izvedba MiMo modela na nivoju zbirnika (assembler) je bila zanimiva in koristna

28 responses



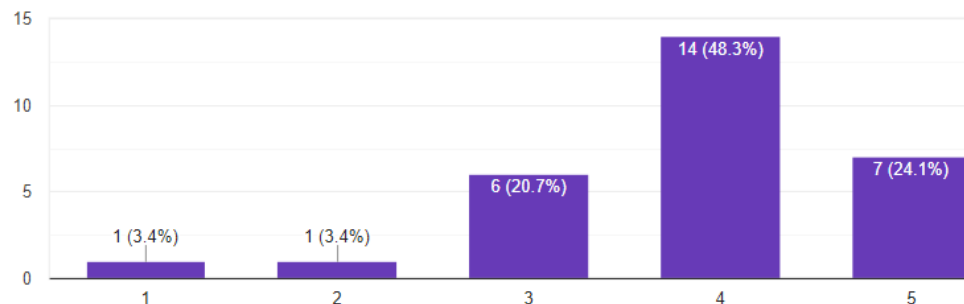
Model MiMo gledano v celoti je pripomogel k izboljšanju mojega poklicnega (strokovnega) znanja.

29 responses



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

29 responses



3.2 MiMo – Mikro-programiran Model CPE

Mnenja (21/22)

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

22 responses



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

22 responses



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

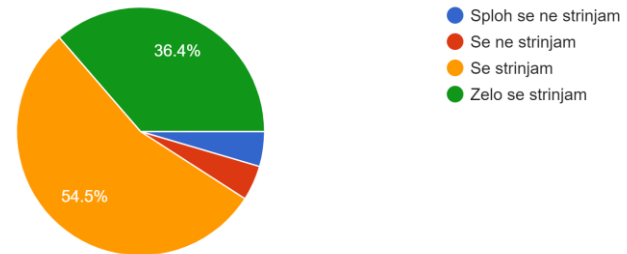
22 responses



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 ...

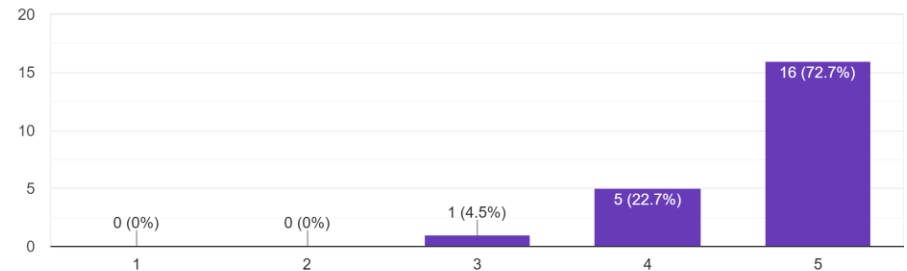
Model MiMo gledano v celoti je pripomogel k izboljšanju mojega poklicnega (strokovnega) znanja.

22 responses



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

22 responses

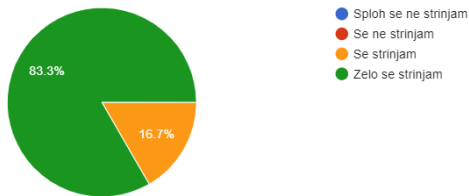


3.2 MiMo – Mikro-programiran Model CPE

Mnenja (22/23)

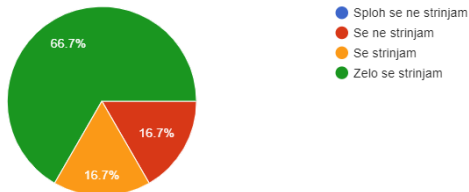
Izvedba MiMo modela na nivoju logičnih vrat v Logisimu je bila zanimiva in koristn:

6 responses



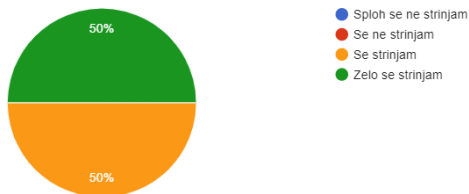
Izvedba MiMo modela na nivoju mikro zbirnika (micro-assembler) je bila zanimiva i koristna

6 responses



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

6 responses

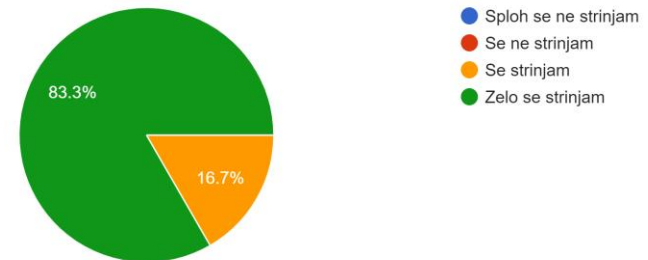


Delo v logisimu oz. simulatorju digitalnih vezij je bila zdaleč največja pomoč pri vizualizaciji same sestave računalnika od digitalnega vezja pa do organizacije.

MiMo ali kakšen lažji manj kompleksen model, bi lahko v sklopu seminarske naloge pri predmetu vsak študent sestavil sam, z tem da bi vsak imel morda specifičen problem, ki bi ga moral rešiti v svetu vgrajenih sistemov.

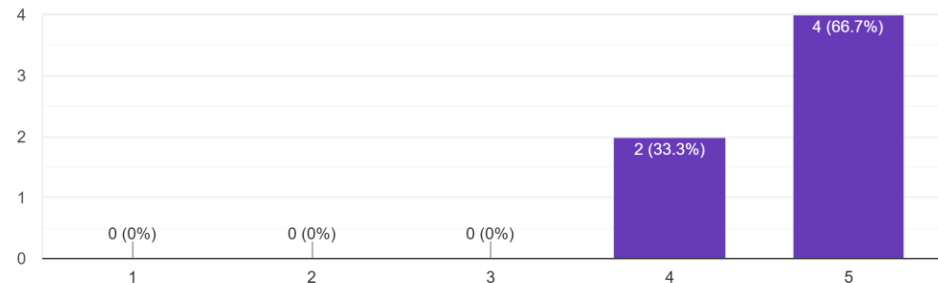
Model MiMo gledano v celoti je pripomogel k razumevanju delovanja CPE

6 responses



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

6 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 | | | |

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

ADD R1,R2,R3 # R1<-R2+R3
R3: Treg: obiĉ. 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!

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

3.2.1 Izvrševanje ukazov – MiMo

Delovanje CPE:

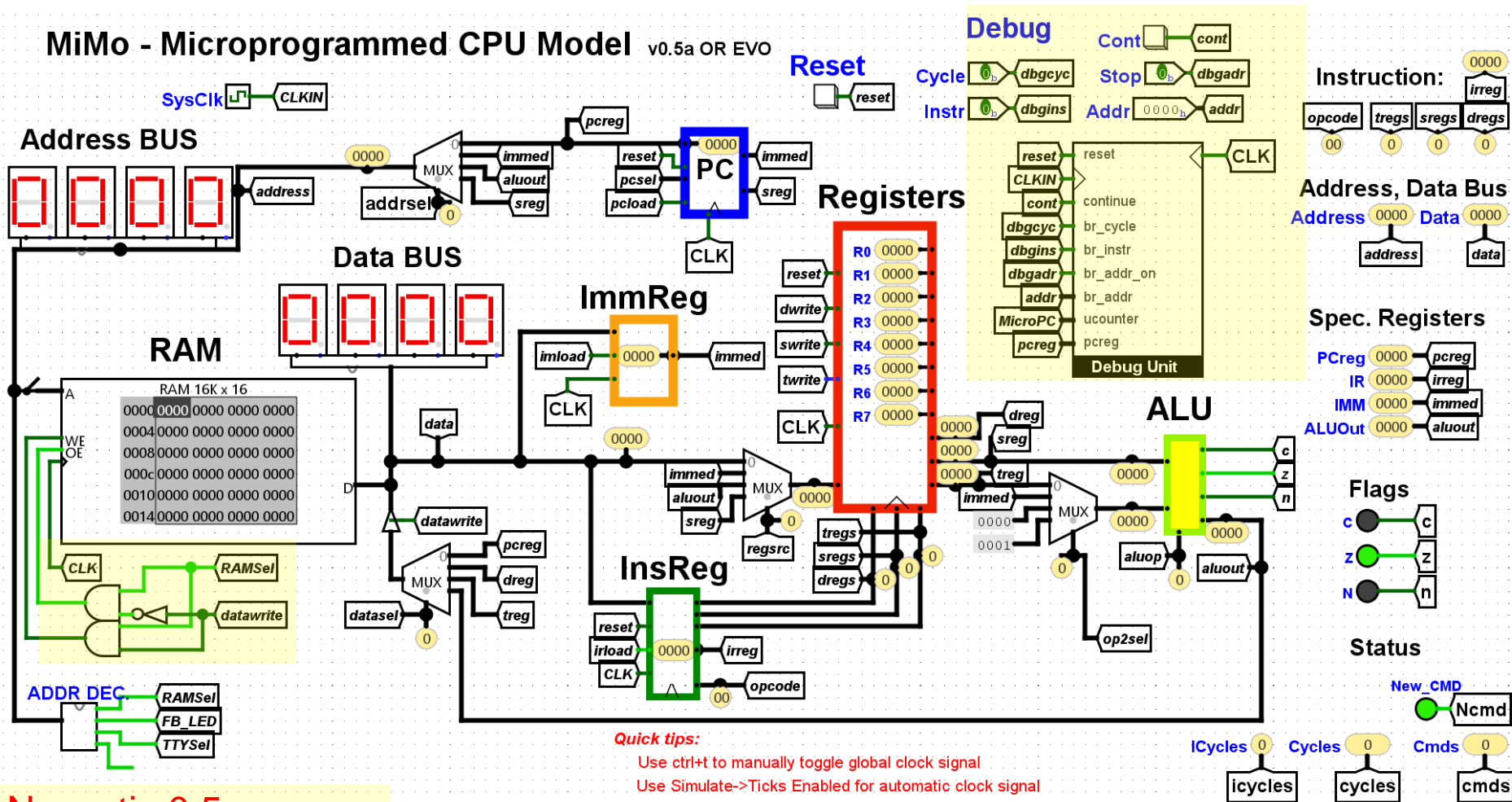
- **branje ukaza iz pomnilnika - FETCH**
 - „ukazno prevzemni cikel“
- **izvrševanje ukaza - EXECUTION**
 - „izvršilni cikel“

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 v0.5a

MiMo - Microprogrammed CPU Model v0.5a OR EVO

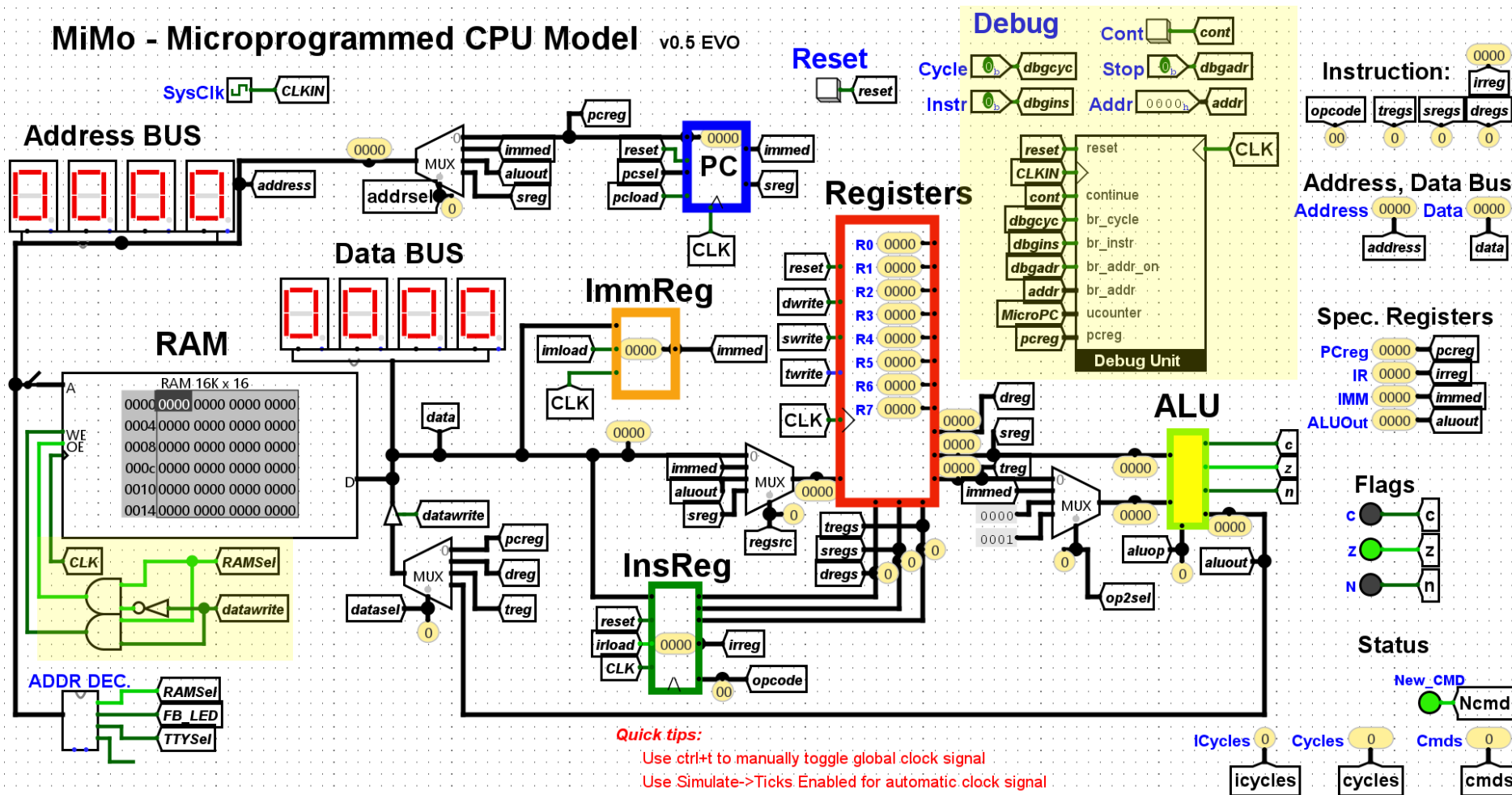


Novosti v0.5a:
pogoji v KE

<https://github.com/LAPSyLAB/MiMo> Student Release

MiMo – Podatkovna enota v0.5

MiMo - Microprogrammed CPU Model v0.5 EVO

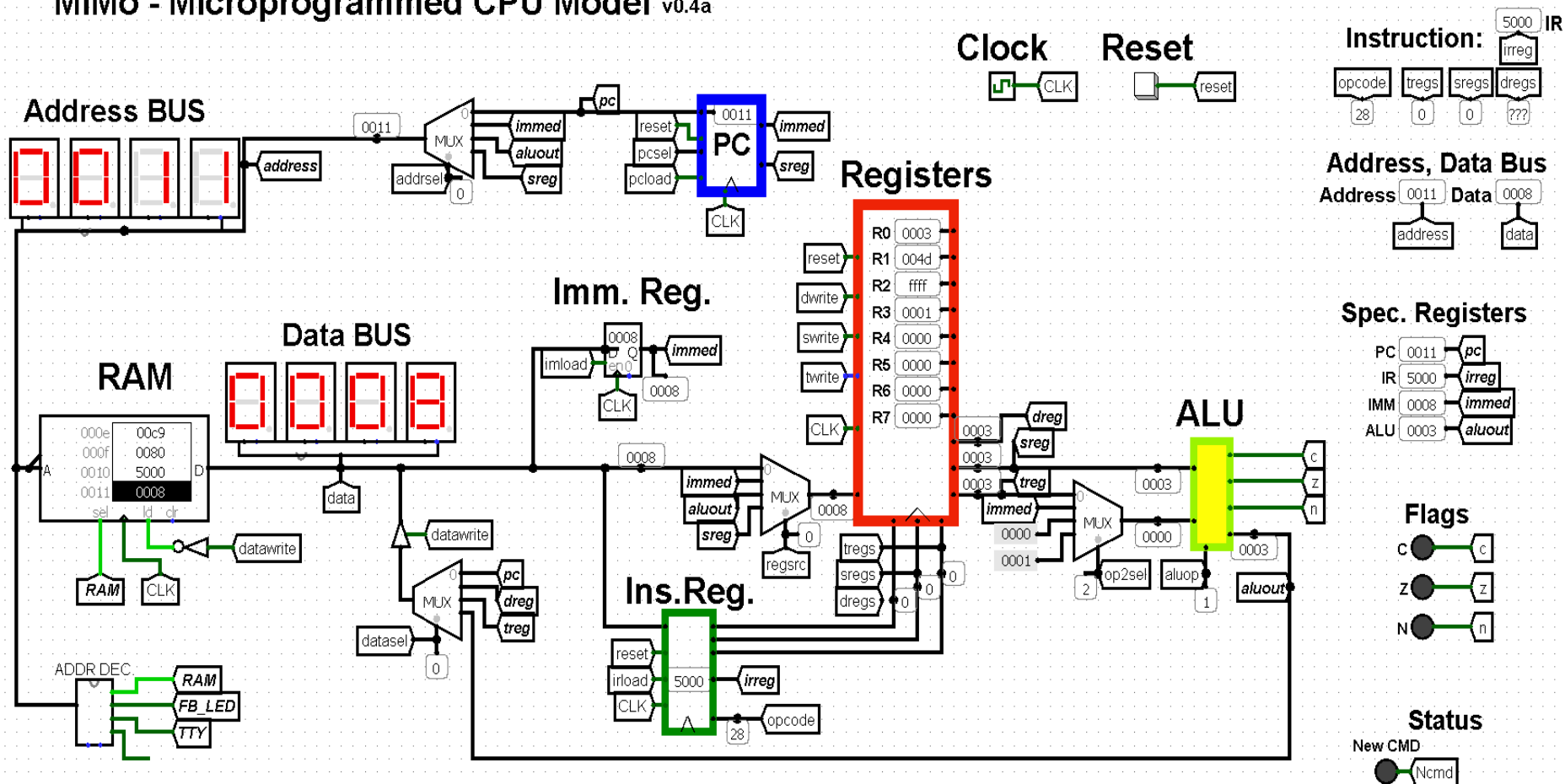


Novosti v0.5

<https://github.com/LAPSyLAB/MiMo> Student Release

MiMo – Podatkovna enota v0.4a

MiMo - Microprogrammed CPU Model v0.4a



https://github.com/LAPSYLAB/MiMo_Student_Release

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
```

Vizualni prikaz je narejen na podobni podatkovni enoti v03.a

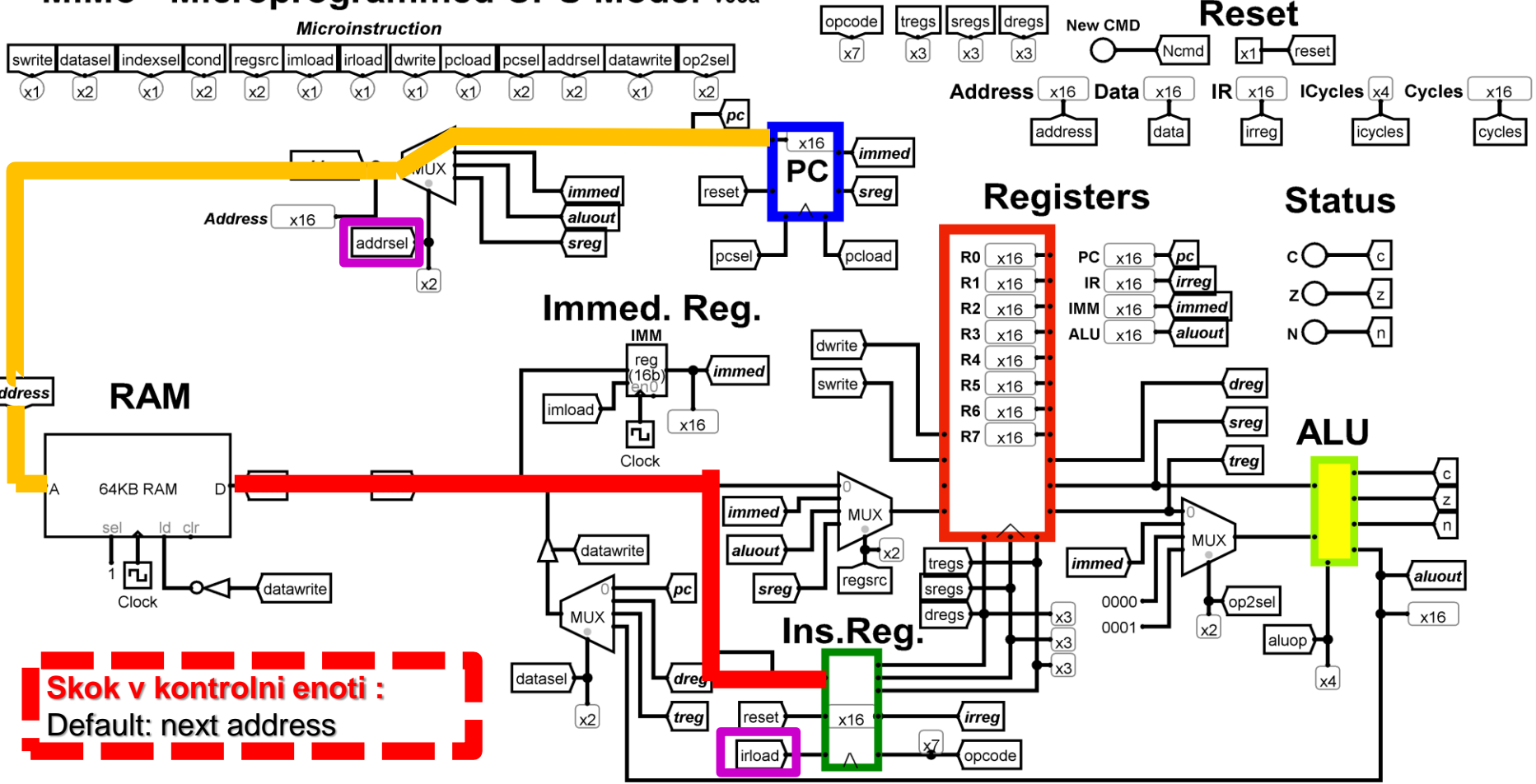
JNEZ Rs,immed:

jnez Rs,immed (40)

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

| | | |
|---------|---|---|
| fetch: | addrsel=pc irload=1 | # Address=PC, Load IR register |
| 40: | pload=1 ptsel=pc, opcode_jump | # PC=PC+1, jump to 2+OPC |
| | 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: | pload=1 ptsel=pc, goto fetch | # Increment PC and goto new command; |
| jump: | pload=1 ptsel=immed, goto fetch | # Set address to immed and goto new command |

MiMo - Microprogrammed CPU Model v03a



Skok v kontrolni enoti :
Default: next address

Naslov:

Podatki:

Kontrolni signali:

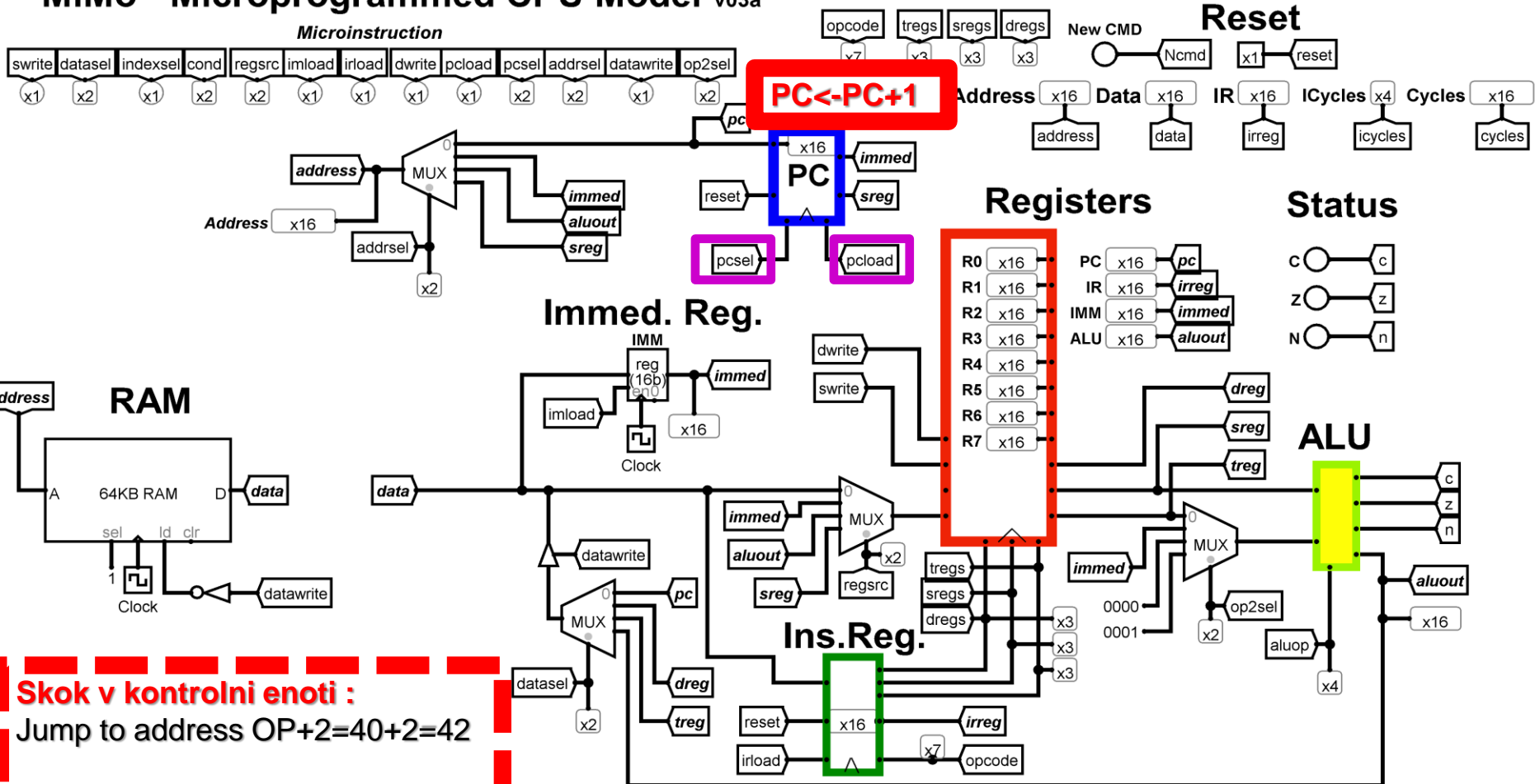
JNEZ Rs,immed:

jnez Rs,immed (40)

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

| | | |
|---------|---|---|
| fetch: | addrsel=pc irload=1 | # Address=PC, Load IR register |
| | pload=1 ptsel=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: | pload=1 ptsel=pc, goto fetch | # Increment PC and goto new command; |
| jump: | pload=1 ptsel=immed, goto fetch | # Set address to immed and goto new command |

MiMo - Microprogrammed CPU Model v03a



Skok v kontrolni enoti :

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

Naslov:

Podatki:

Kontrolni signali:

JNEZ Rs,immed:

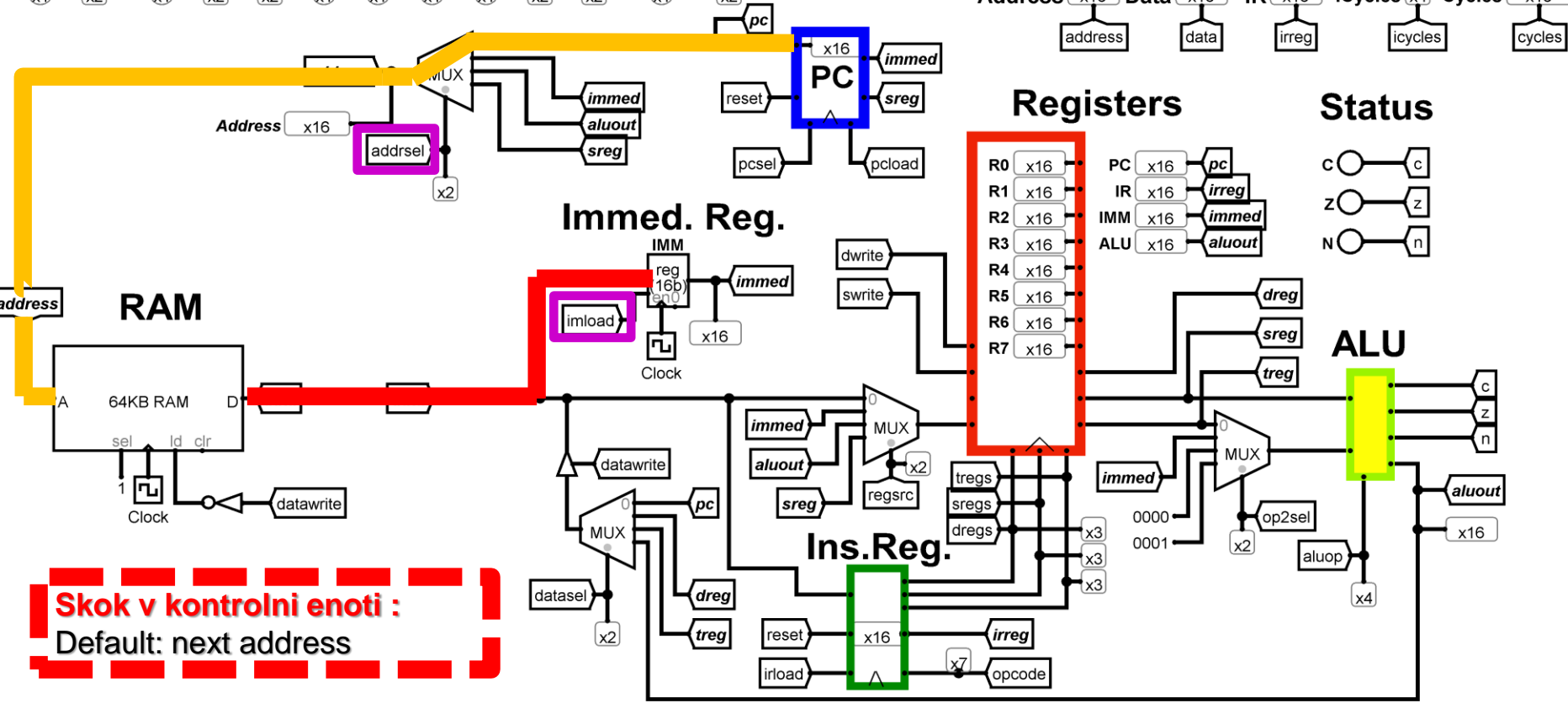
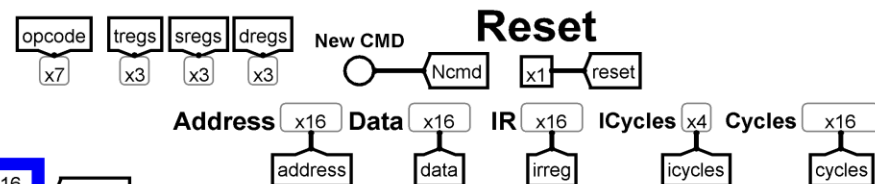
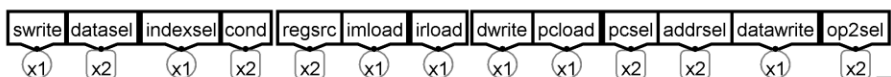
jnez Rs,immed (40)

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

| | | |
|---------|---|---|
| fetch: | addrsel=pc irload=1 | # Address=PC, Load IR register |
| | pload=1 ptsel=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: | pload=1 ptsel=pc, goto fetch | # Increment PC and goto new command; |
| jump: | pload=1 ptsel=immed, goto fetch | # Set address to immed and goto new command |

MiMo - Microprogrammed CPU Model v03a

Microinstruction



Skok v kontrolni enoti :
Default: next address

Naslov:

Podatki:

Kontrolni signali:

JNEZ Rs,immed:

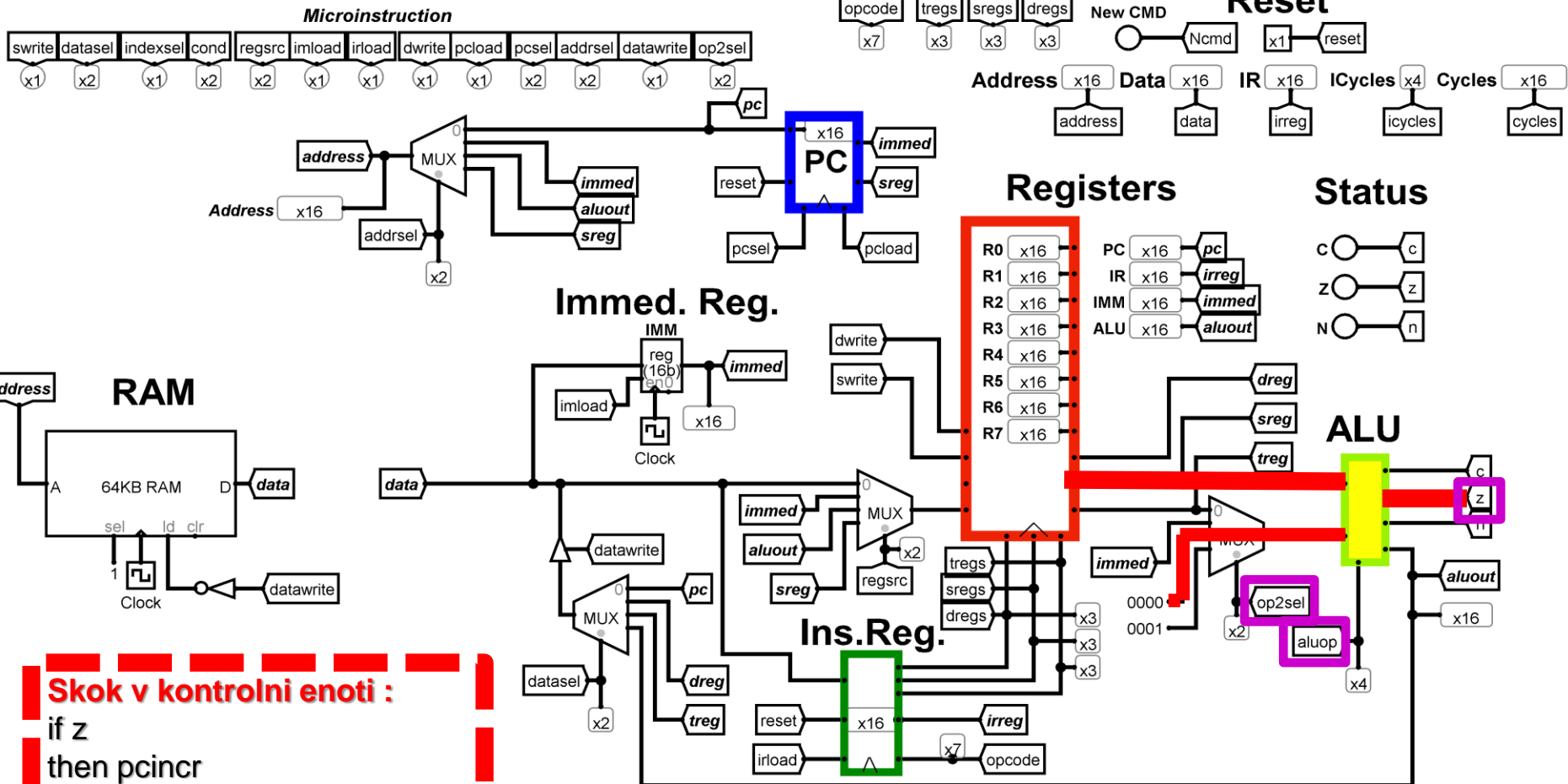
jnez Rs,immed (40)

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

fetch: addrsel=pc irload=1
 pload=1 ptsel=pc, opcode_jump
 40: addrsel=pc imload=1
 aluop=sub op2sel=const0, if z then pcincr else jump
 pcincr: pload=1 ptsel=pc, goto fetch
 jump: pload=1 ptsel=immed, goto fetch

Address=PC, Load IR register
 # PC=PC+1, jump to 2+OPC
 # 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

MiMo - Microprogrammed CPU Model v03a



Skok v kontrolni enoti :
 if z
 then pcincr
 else jump

JNEZ Rs,immed: velja Rs=0

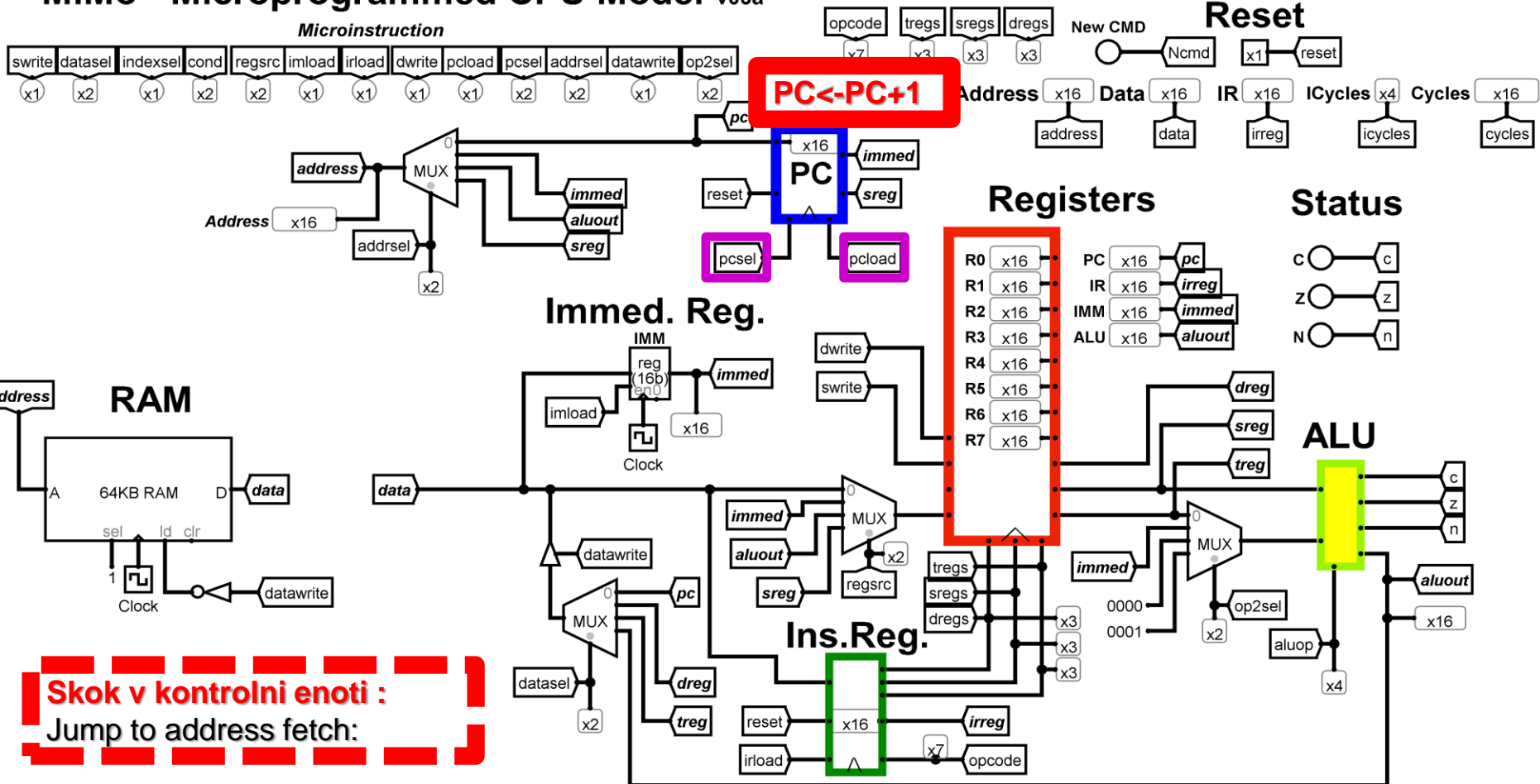
jnez Rs,immed (40)

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

fetch: addrsel=pc irload=1
 pload=1 ptsel=pc, opcode_jump
 40: addrsel=pc imload=1
 aluop=sub op2sel=const0, if z then pcincr else jump
 pcincr: pload=1 ptsel=pc, goto fetch
 jump: pload=1 ptsel=immed, goto fetch

Address=PC, Load IR register
 # PC=PC+1, jump to 2+OPC
 # 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

MiMo - Microprogrammed CPU Model v03a



Skok v kontrolni enoti:
 Jump to address fetch:

Naslov:

Podatki:

Kontrolni signali:

JNEZ Rs,immed: velja Rs≠0

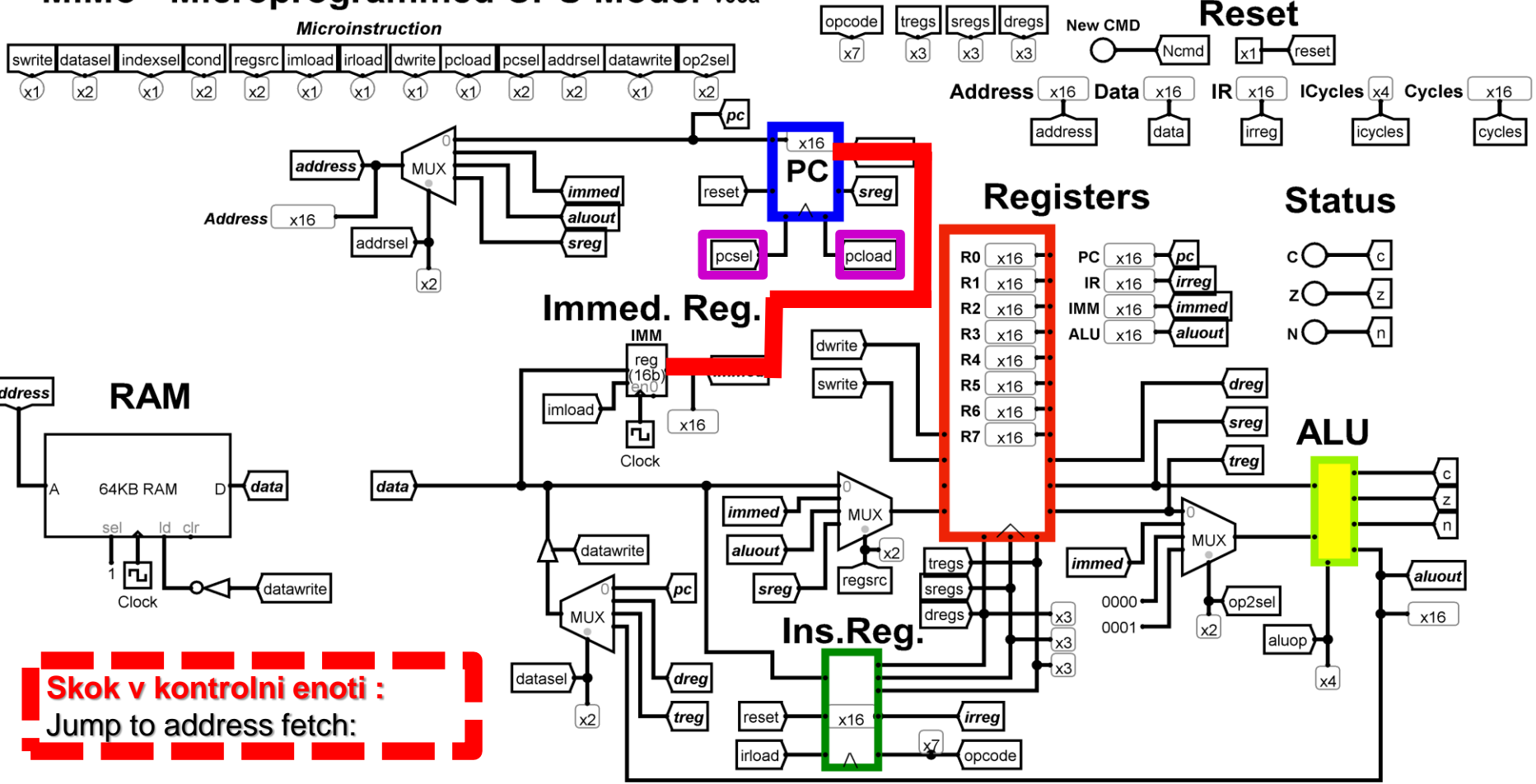
jnez Rs,immed (40)

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

fetch: addrsel=pc irload=1
 pload=1 ptsel=pc, opcode_jump
 40: addrsel=pc imload=1
 aluop=sub op2sel=const0, if z then pcincr else jump
 pcincr: pload=1 ptsel=pc, goto fetch
 jump: pload=1 ptsel=immed, goto fetch

Address=PC, Load IR register
 # PC=PC+1, jump to 2+OPC
 # 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

MiMo - Microprogrammed CPU Model v03a



Skok v kontrolni enoti :
 Jump to address fetch:

Naslov:

Podatki:

Kontrolni signali:

3.2.2. MiMo – podatkovna enota

3.2.2.1 ALE:

Označevanje :

- VHODI
- IZHODI
- KONTROLNI SIGNALI

Vhodi:

2x 16-bitna operanda:

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

Izhodi:

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

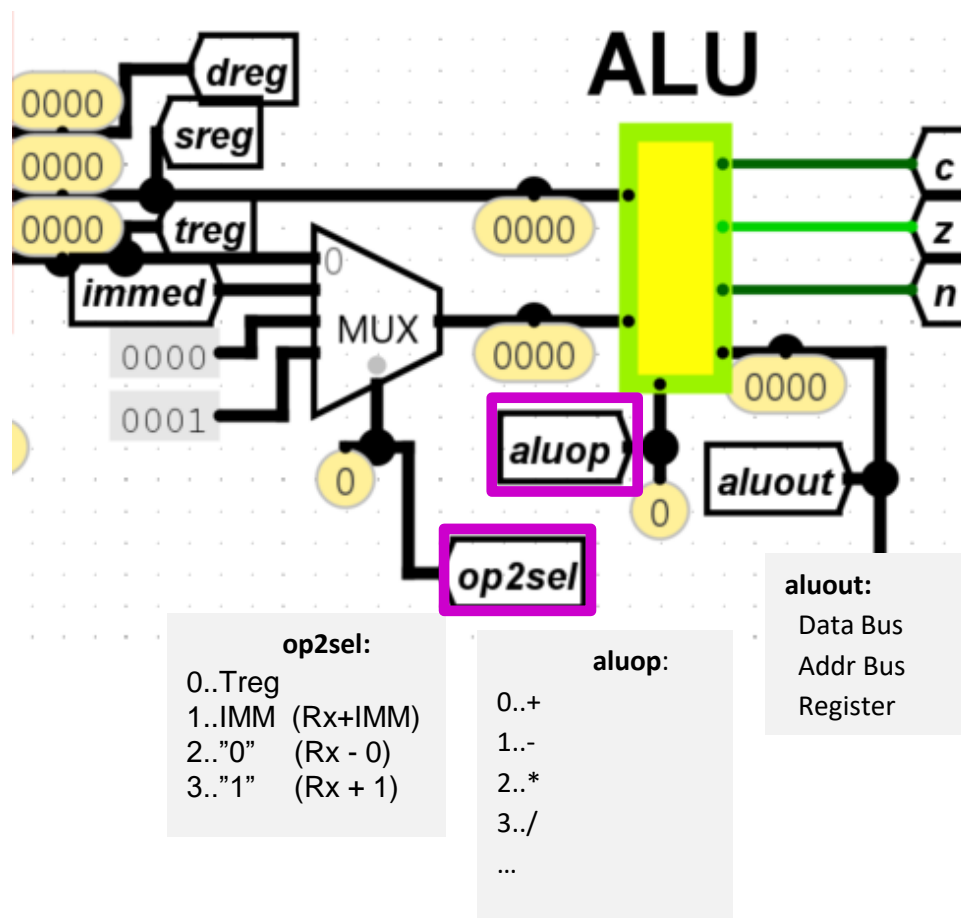
Kontrolni signali:

op2sel – določi 2. operand:

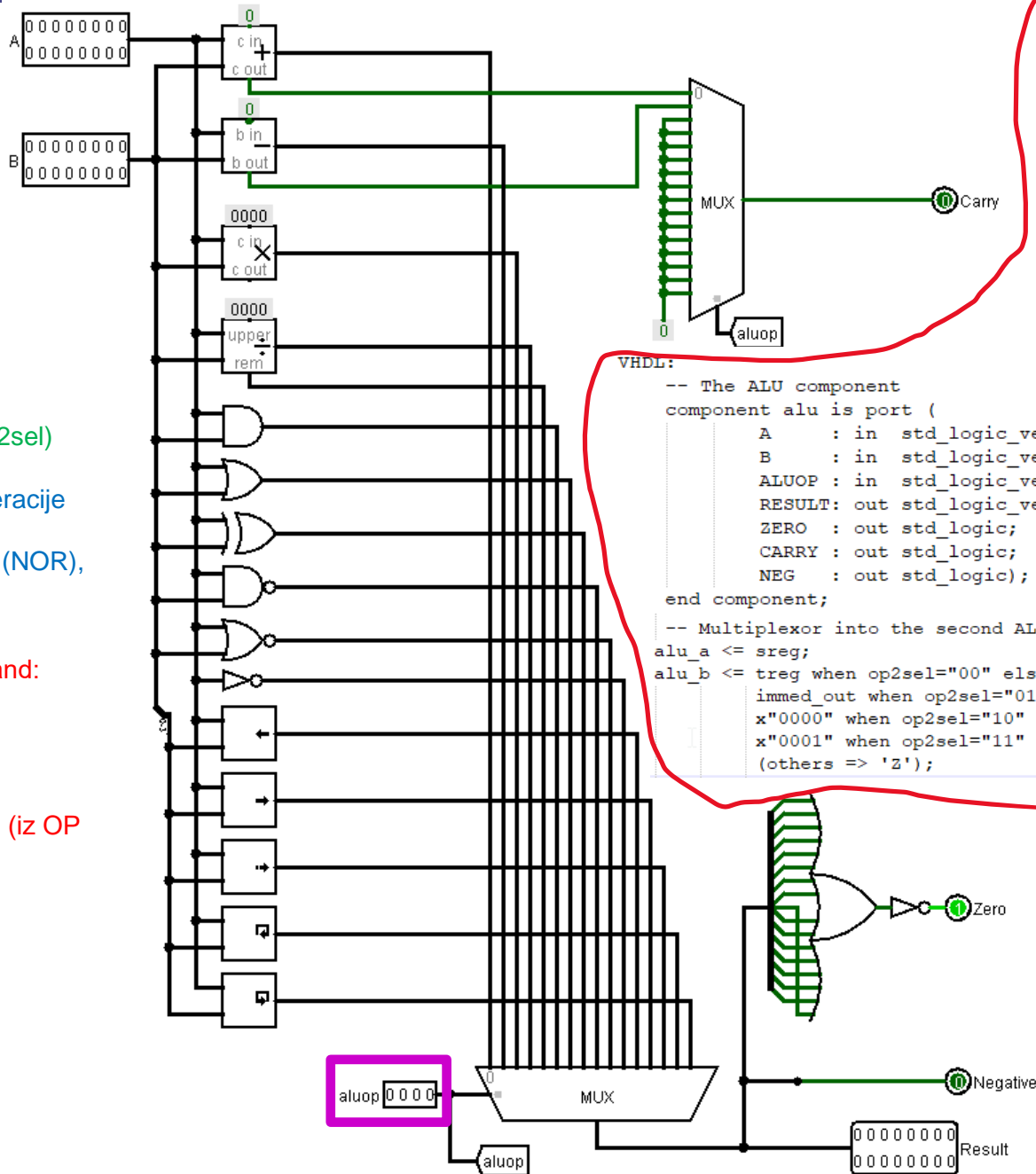
- 0..?
- 1..?
- 2..?
- 3..?

aluop – določi operacijo (iz OP kode)

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



16-bit ALU



```
-- 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');
```

3.2.2.1 ALE:

Vhodi:

2x 16-bitna operanda:

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

Izhodi:

- 16-bitni rezultat operacije («aluout»)
- zastavice C (+,-), Z (NOR), N (b₁₅)

Kontrolni signali:

op2sel – določi 2. operand:

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

aluop – določi operaciju (iz OP kode)

- 0..+
- 1..-
- 2..*
- 3../
- ...

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:

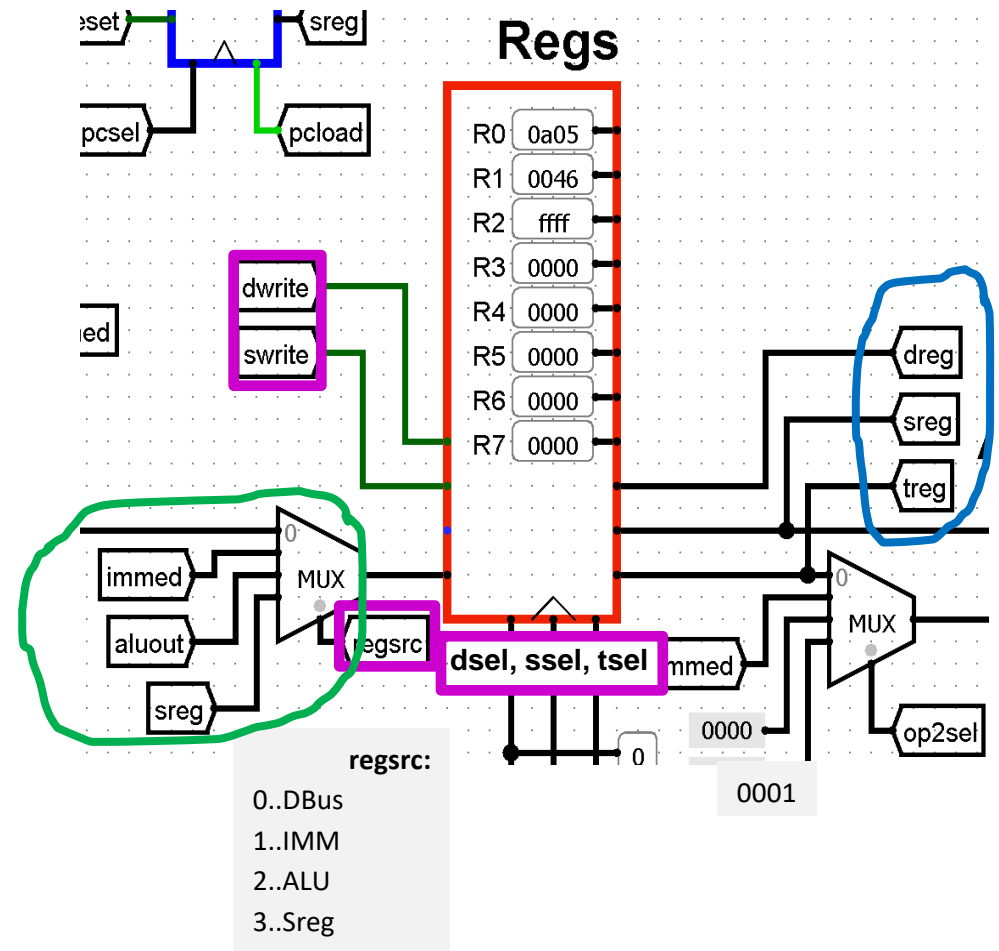
- Sreg -> ALU, vh.Regis, Nasl.vodilo
- Treg -> ALU, DataBus
- Dreg -> DataBus

regsrc – določa vhodni izvor:

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

dwrite, swrite, twrite

(običajno samo eden) – določajo pisalno operacijo iz vhoda v izbrane registre



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.Regis, Nas l.vodilo
- Treg -> ALU, DataBus
- Dreg -> DataBus

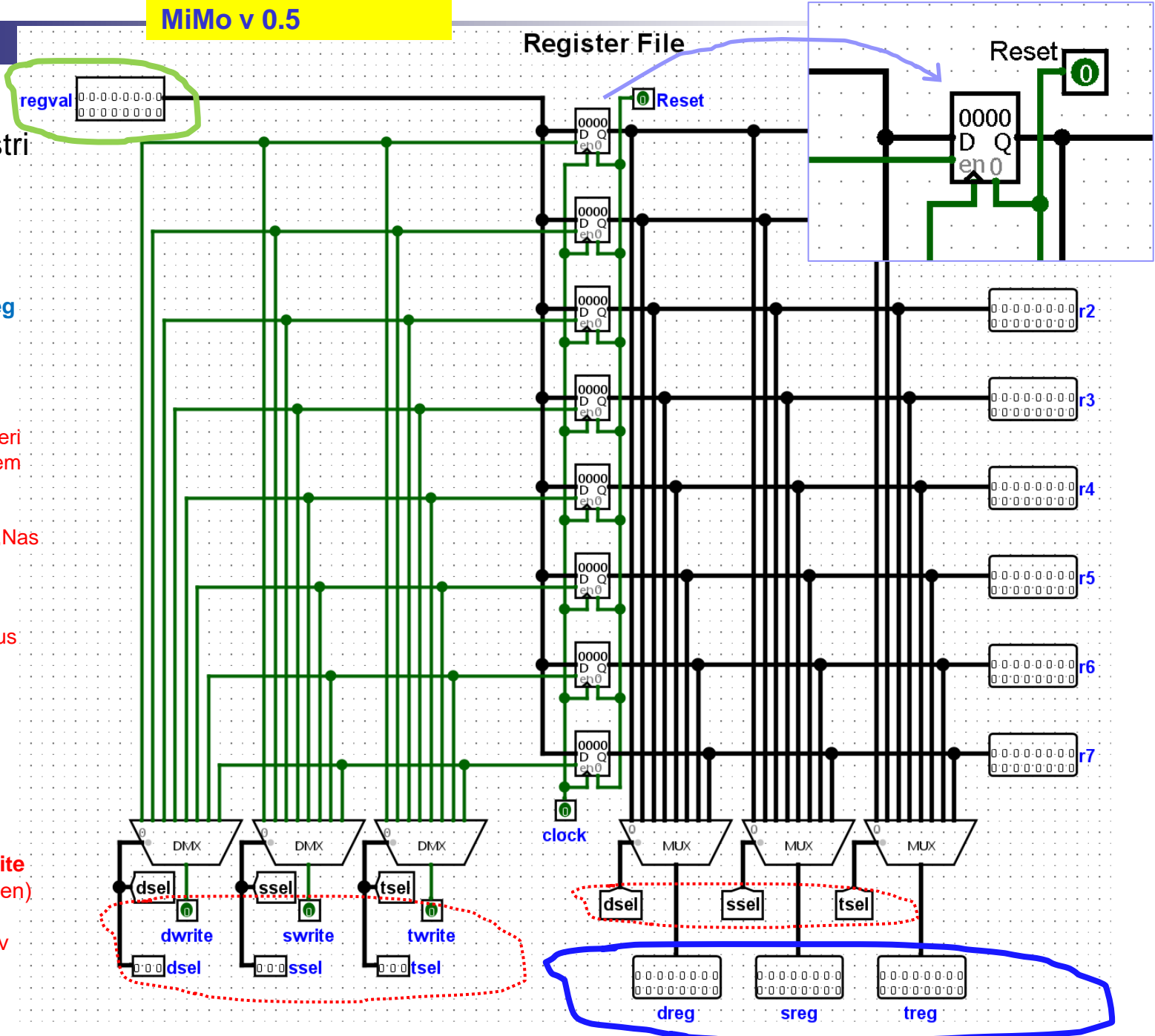
regsrc – določa

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

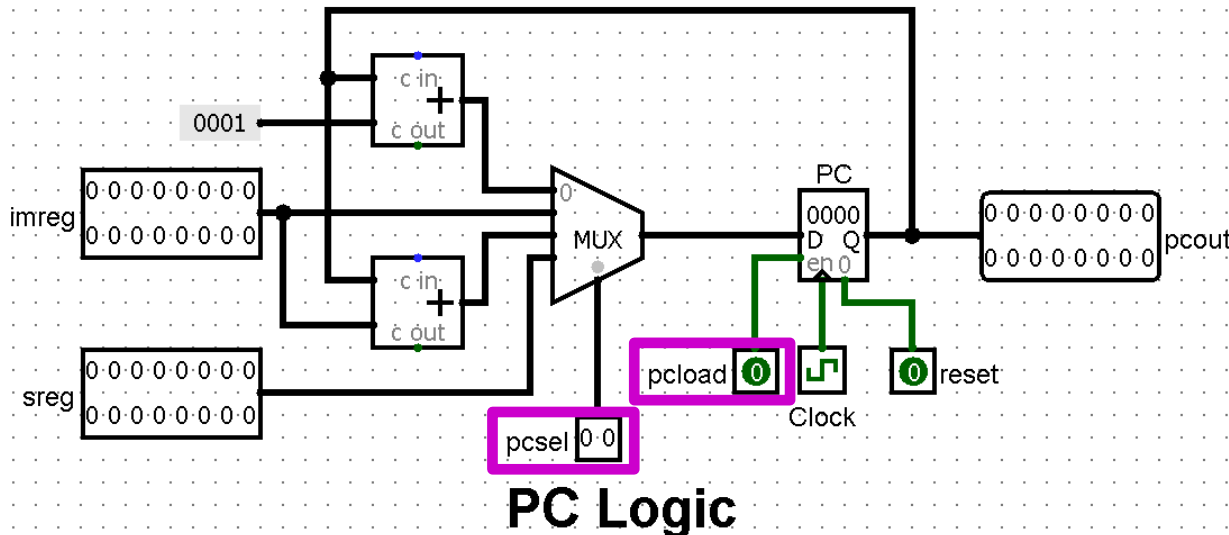
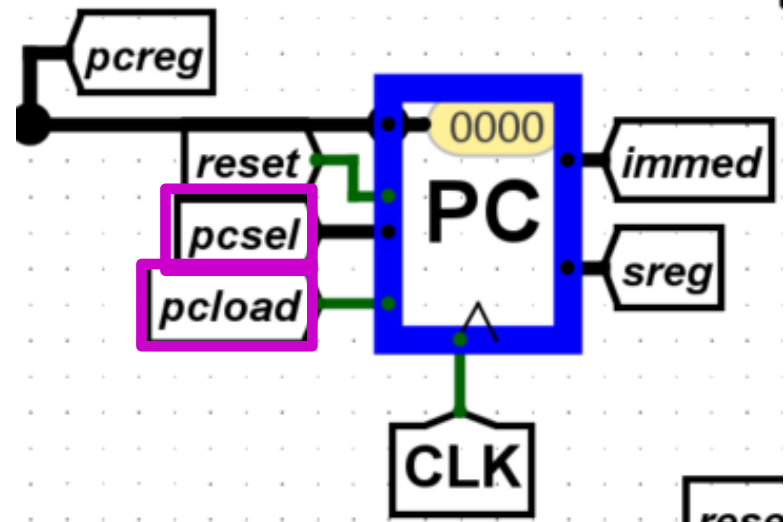
3.2.2.3 Programski št.-PC

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)

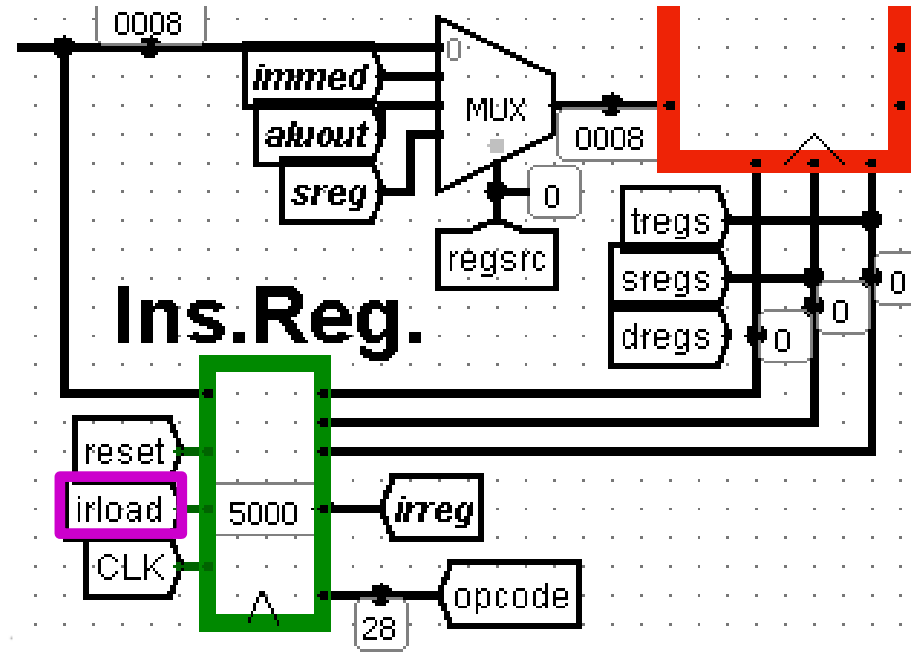
Izhodi: razdelitev ukaza na polja :

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

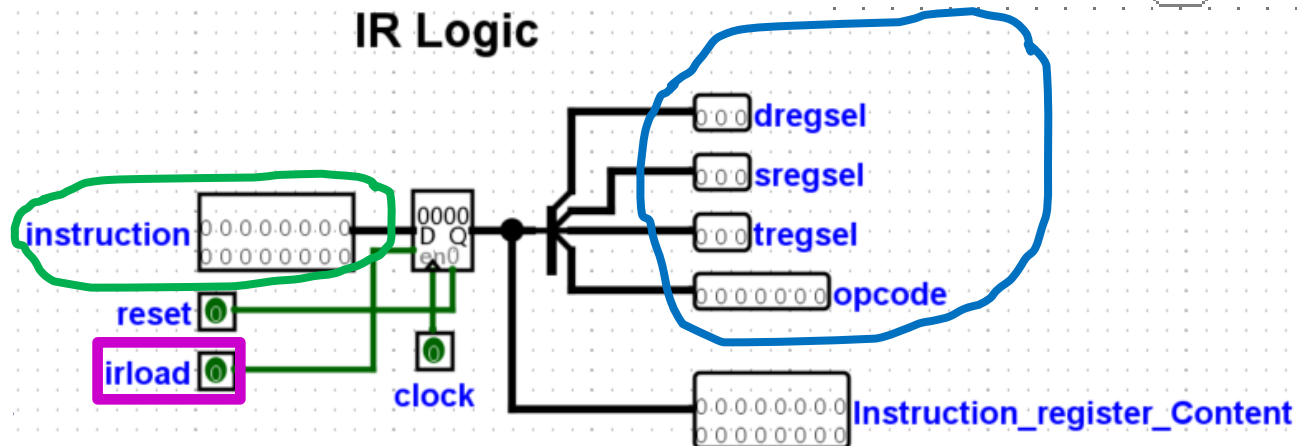
skupaj 16 bitov

Kontrolni signal:

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



IR Logic



3.2.2. MiMo – podatkovna enota

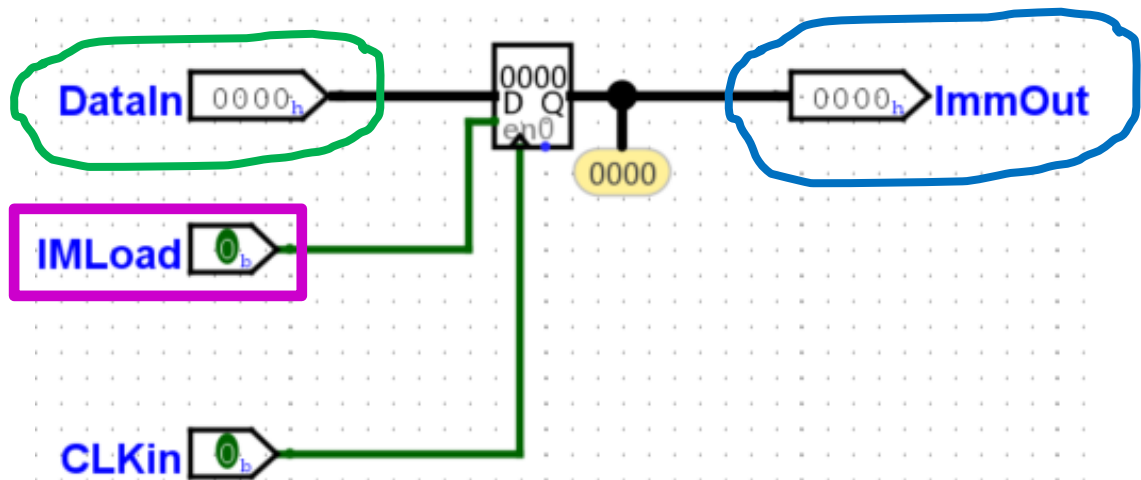
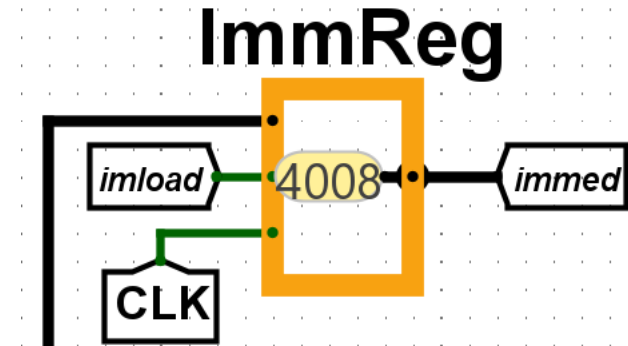
3.2.2.5 Takojšnji reg.-“immed“
(takojšnji register, „Immediate“)
shranjuje takojšnji operand

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:

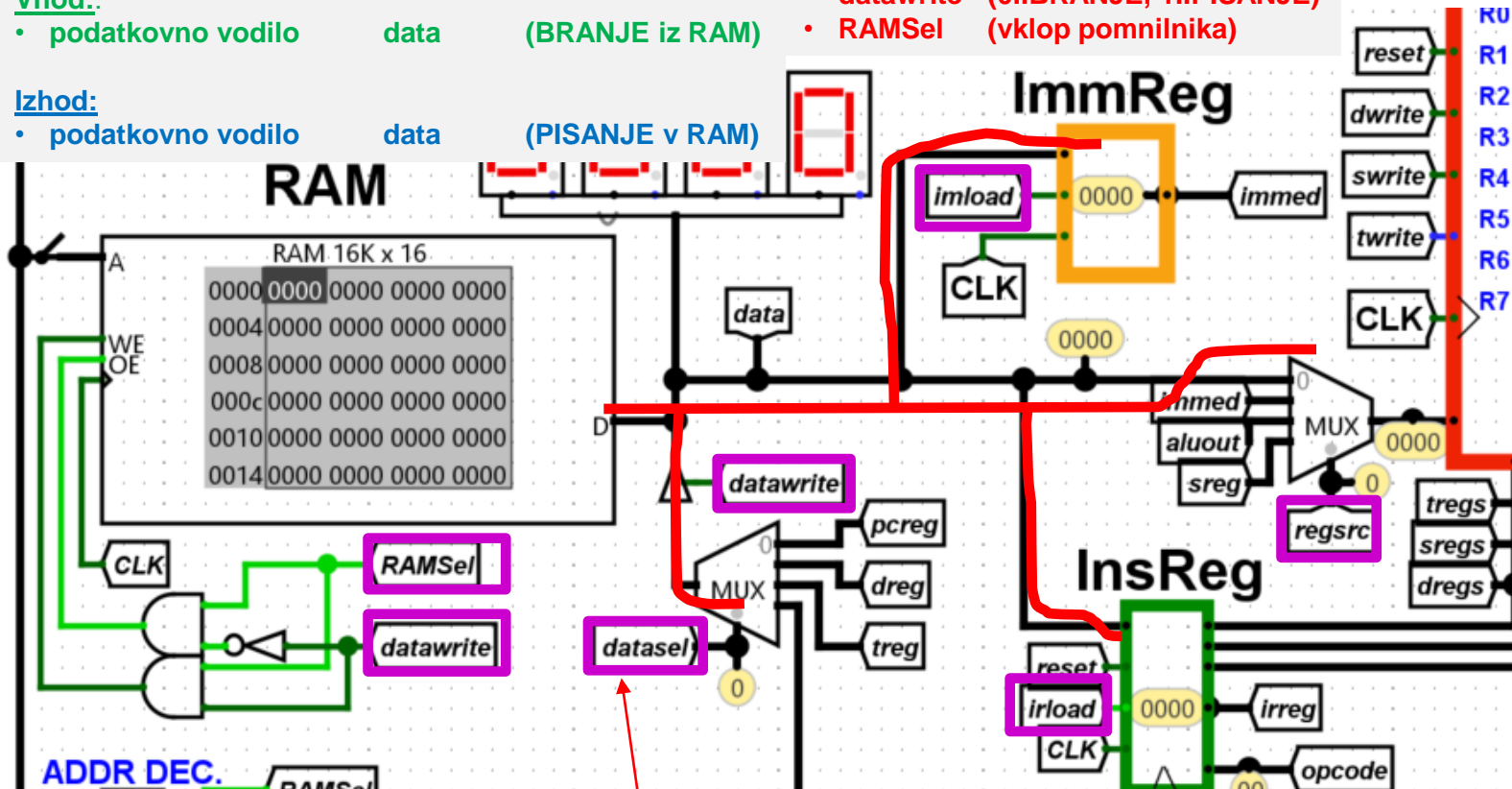
- podatkovno vodilo data (BRANJE iz RAM)

Izhod:

- podatkovno vodilo data (PISANJE v RAM)

Kontrolni:

- datawrite (0..BRANJE, 1..PISANJE)
- RAMSel (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:

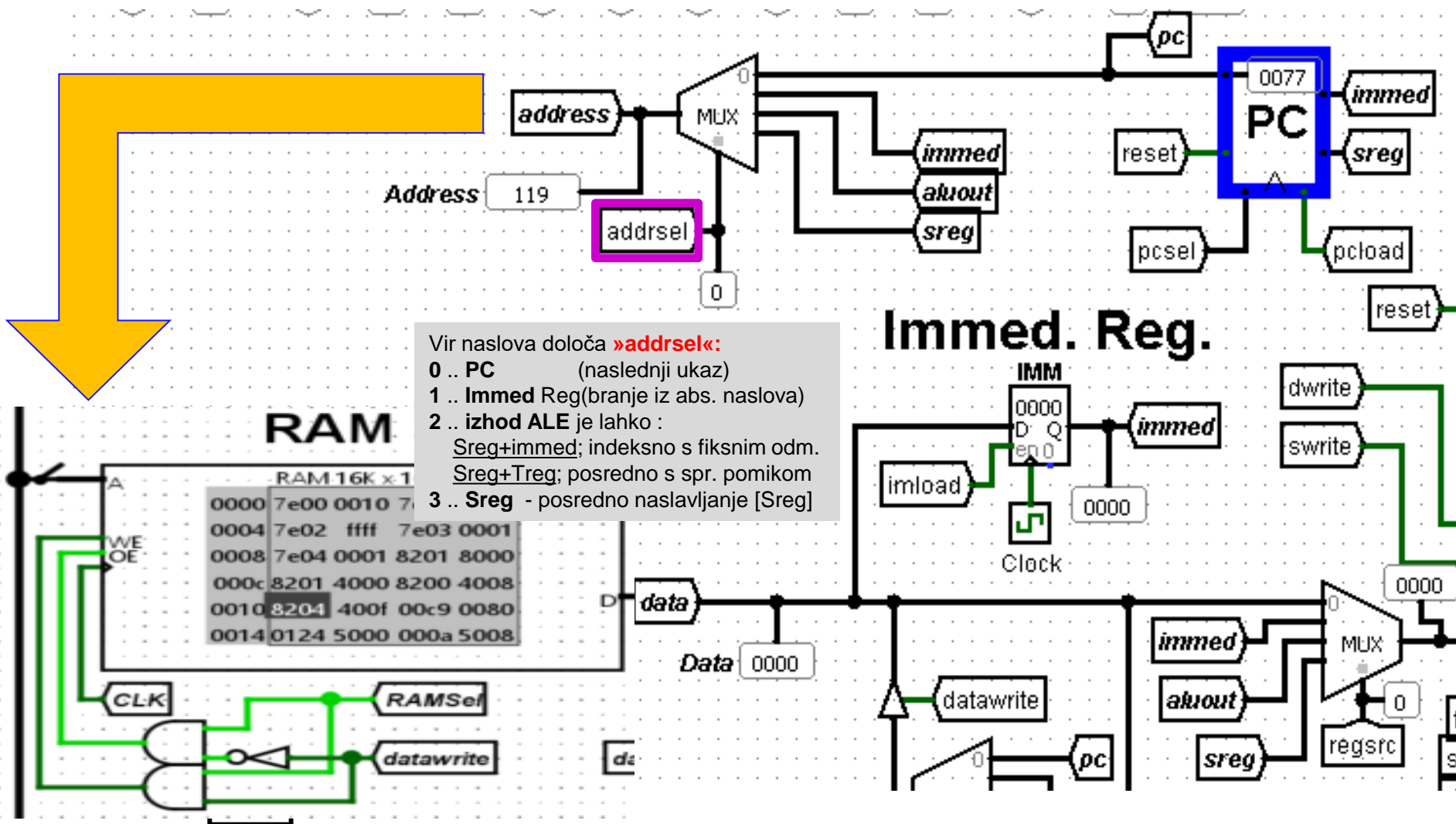
pri vpišu v RAM pomnilnik pa se smer obrne – aktiven **datawrite**.

- 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



3.2.2.8 RAM pomnilnik

RAM pomnilnik je priključen na:

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

Kontrolna signala:

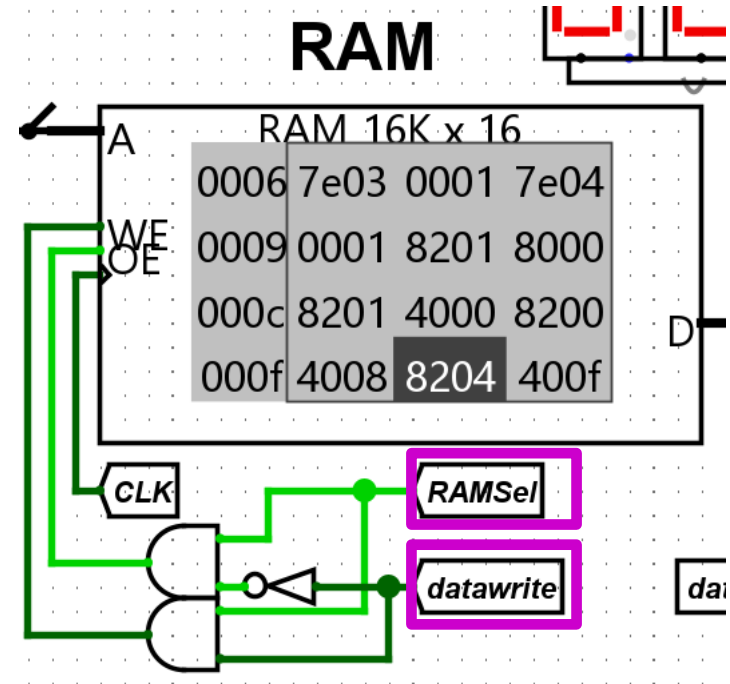
- **datawrite** :
 - določa branje (0) ali pisanje (1)
- **RAMsel(ect)** :
 - aktivira (1) RAM pomnilnik
 - deaktivira (0) RAM pomnilnik
(uporaba: „naslovno dekodiranje“)

Vhod

- **Naslov (A)**: 14 bitni, naslovni prostor 16-bitni

Izhod/Vhod :

- **Podatkovno vodilo (D)**



3.2.2.8 RAM pomnilnik

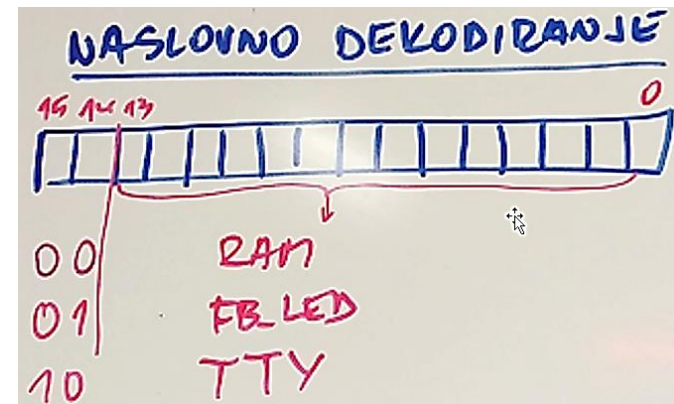
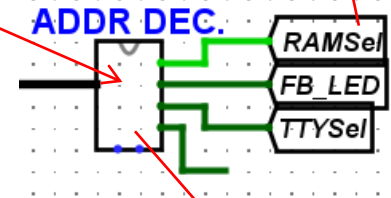
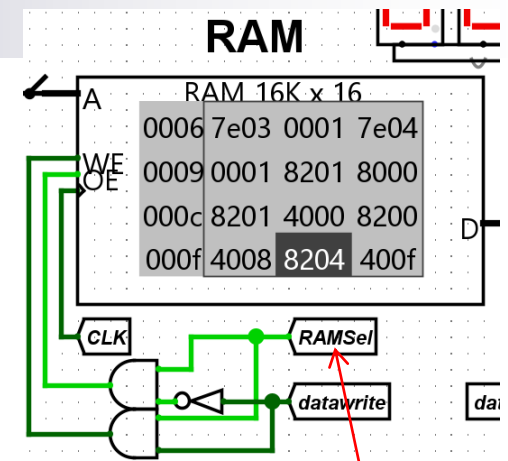
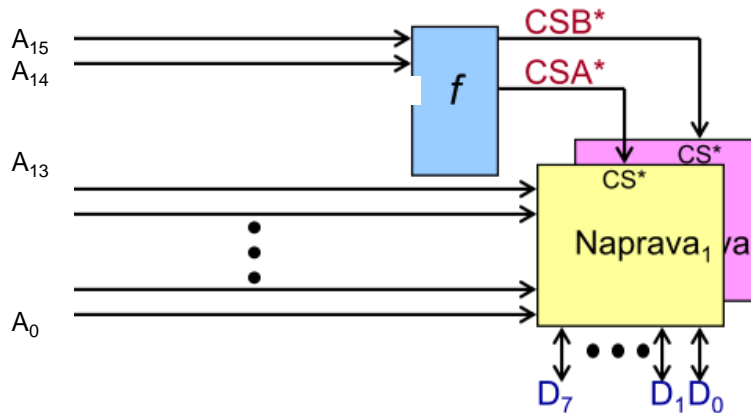
Naslov RAM 14 bitni

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“)

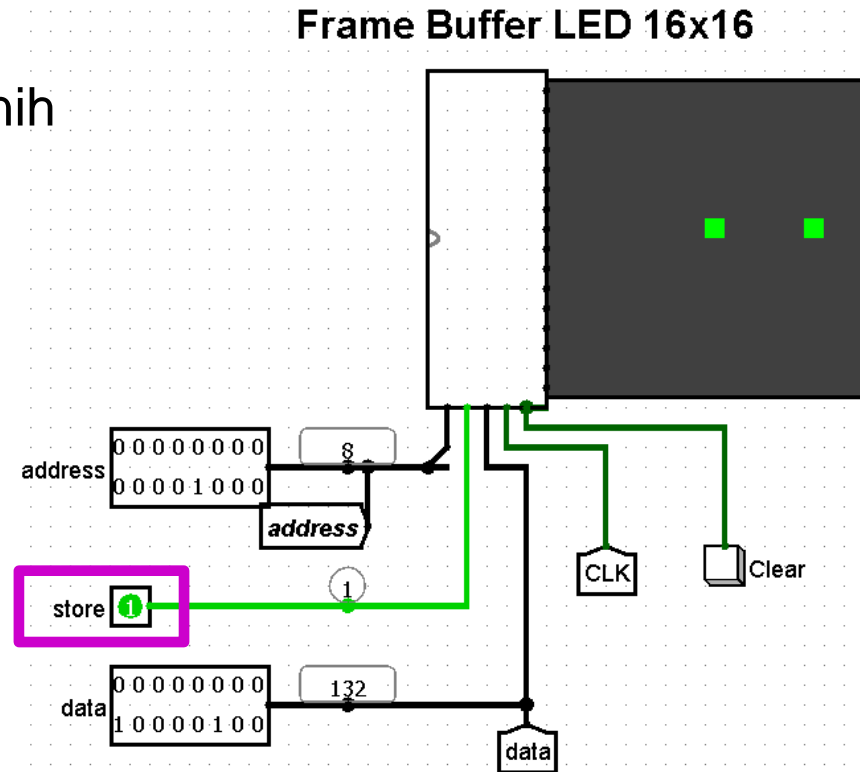
Vsebina zaslona LED 16x16 je zapisana v 16 zaporednih 16-bitnih registrih („framebuffer“).

Vhodi:

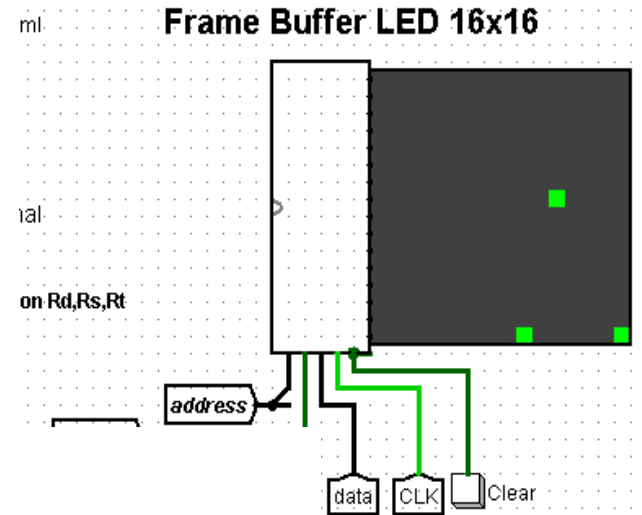
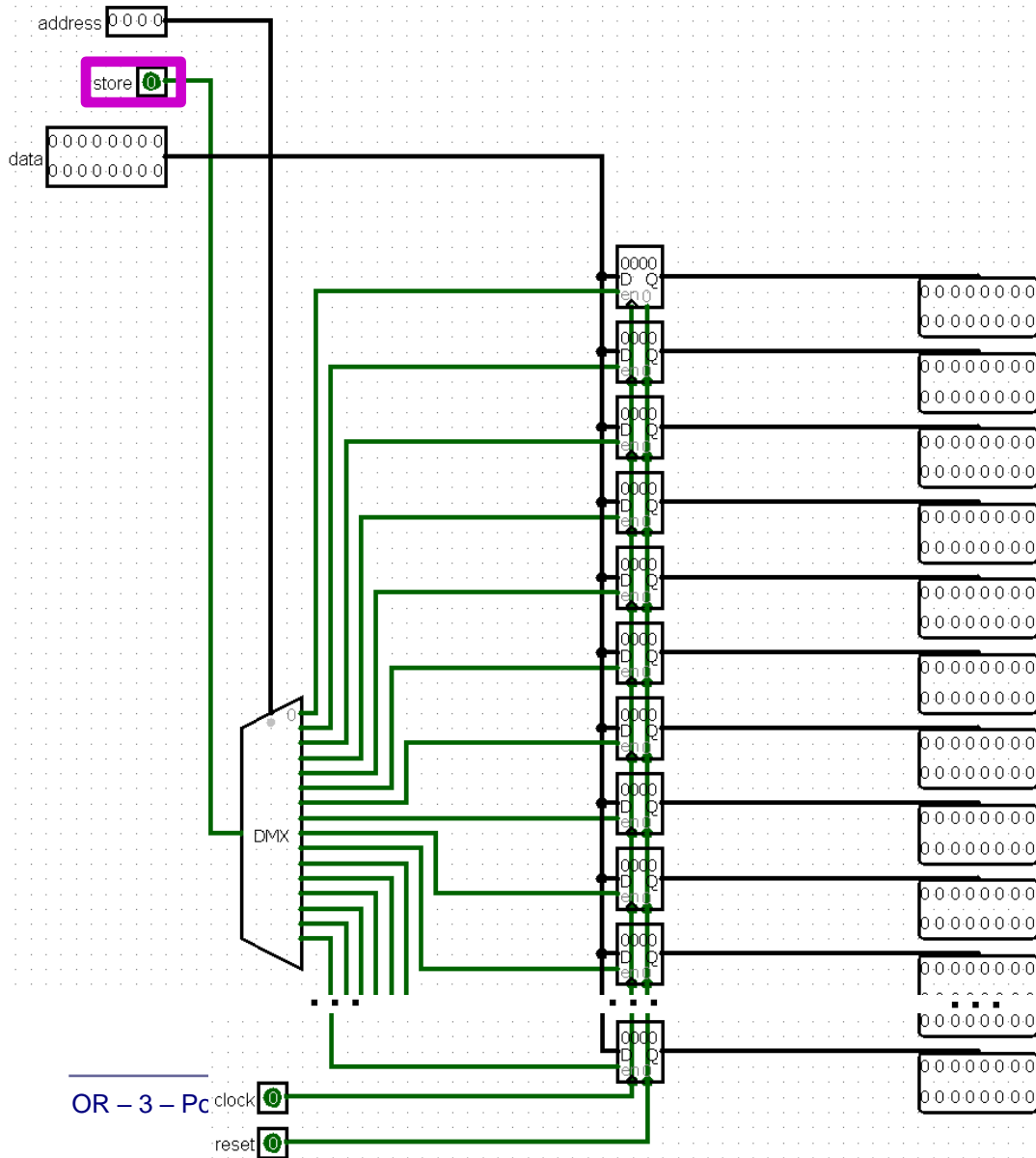
- address: naslov,
- data: podatek (vrednost),

Kontrolni signali:

- „store“ (pisanje),
- CLK,
- „Clear“



3.2.2.9 Grafični zaslon („FrameBuffer“)



3.2.2.10 Serijski terminal („TTY“)

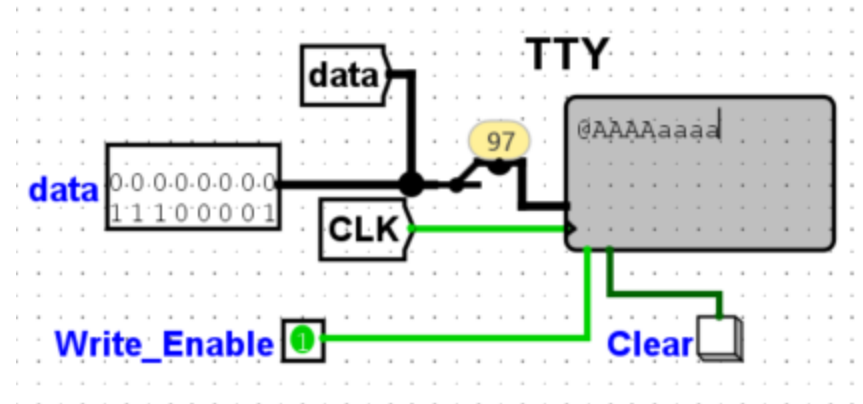
Vsebina zaslona je prikazana v 4 vrsticah in 16 znakih.

Vhodi:

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

Kontrolni signali:

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



3.2.2.11 Debug enota

Vsebina zaslona je prikazana v 4 vrsticah in 16 znakih.

Vhodi:

- clock: urin signal Logisim
- Addr: naslov ukaza ustavitve („breakpoint“)

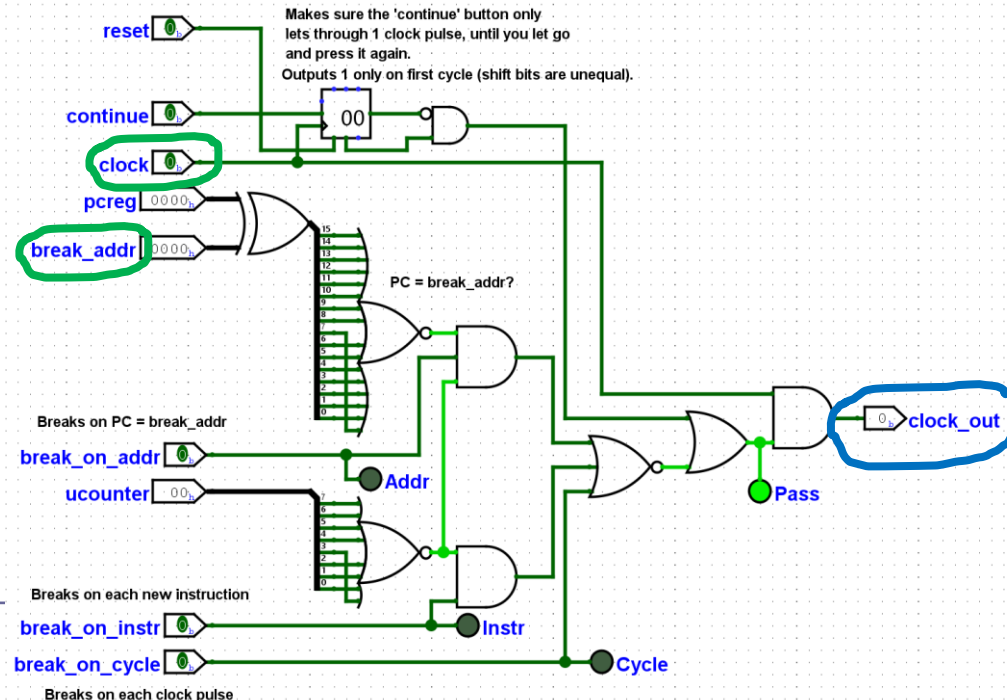
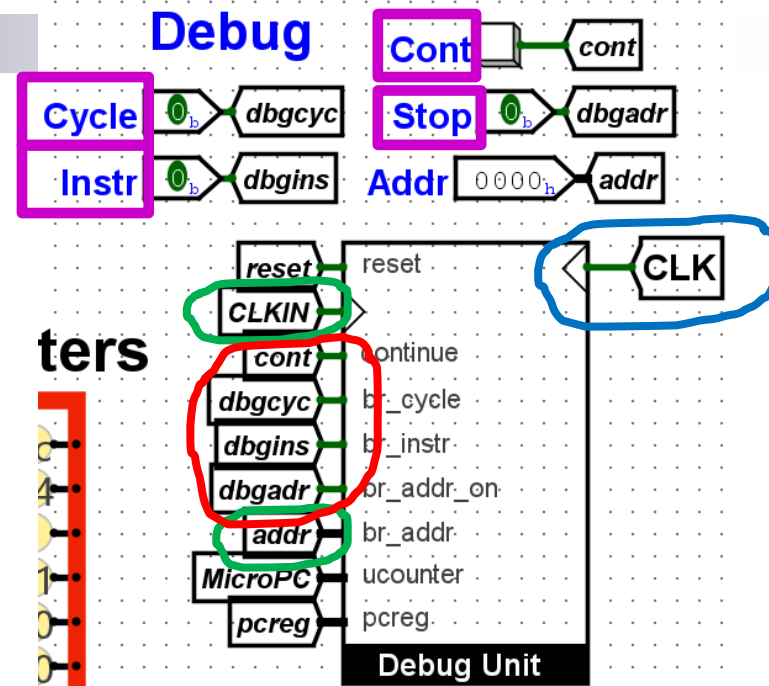
Izhod

- CLK: glavni urin signal sistema

Uporabniške kontrole:

- Cycle: ustavitev vsako periodo
- Instr: ustavitev vsak nov strojni ukaz
- Stop: ustavitev na naslovu Addr
- Cont: nadaljaj izvedbo (po ustavitvi)

Avtor: Maks Popović



3.2.3 MiMo – Kontrolna enota

Mikroukaz = elementarni korak

Vsak mikroukaz določa :

- stanje vseh ?
- naslednji ?

Vhodi v KE: 

opcode – operacijska koda ukaza
C, Z, N zastavice

Izhodi iz KE 

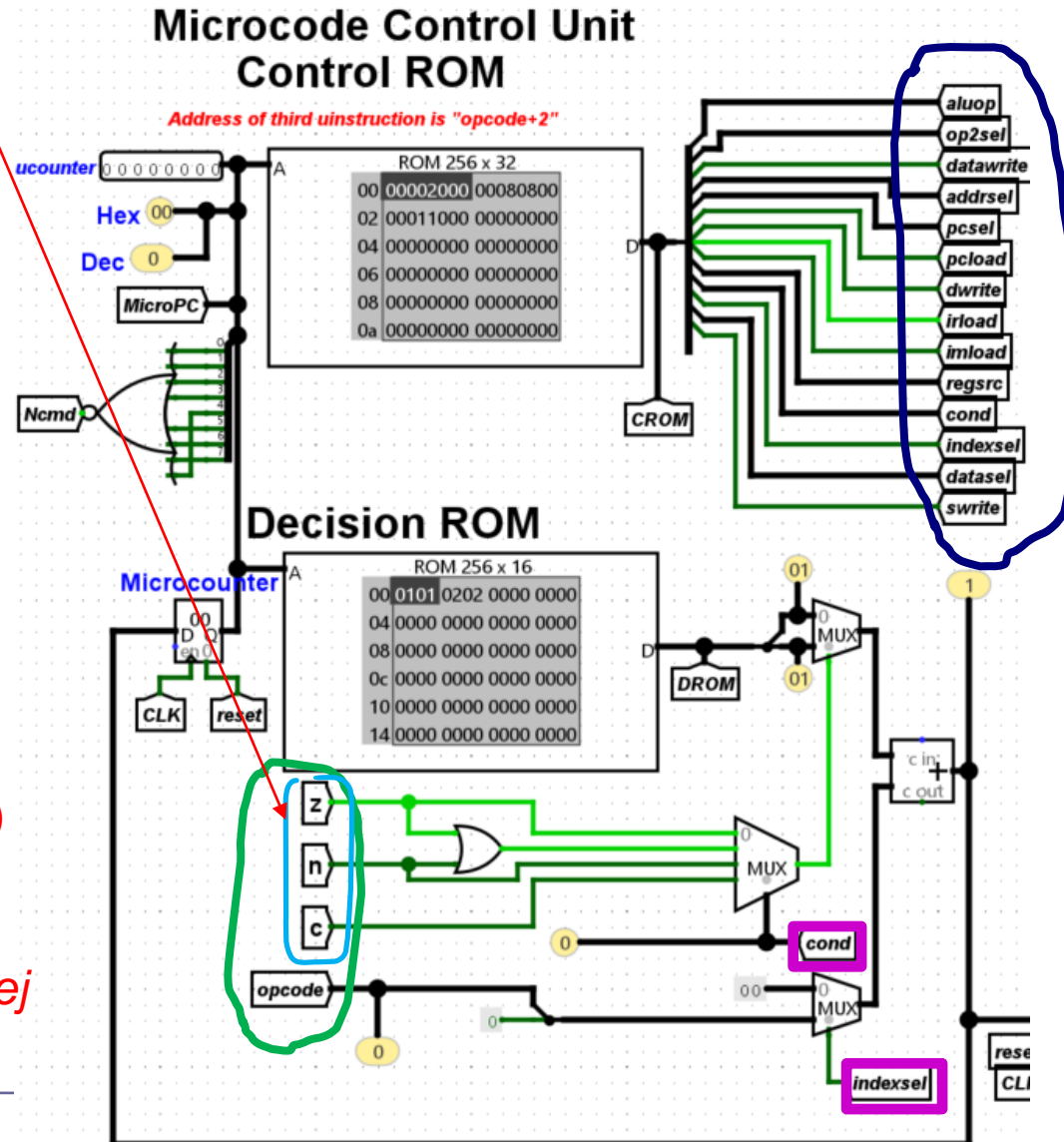
Vsi kontrolni signali

Kontrolni signali:

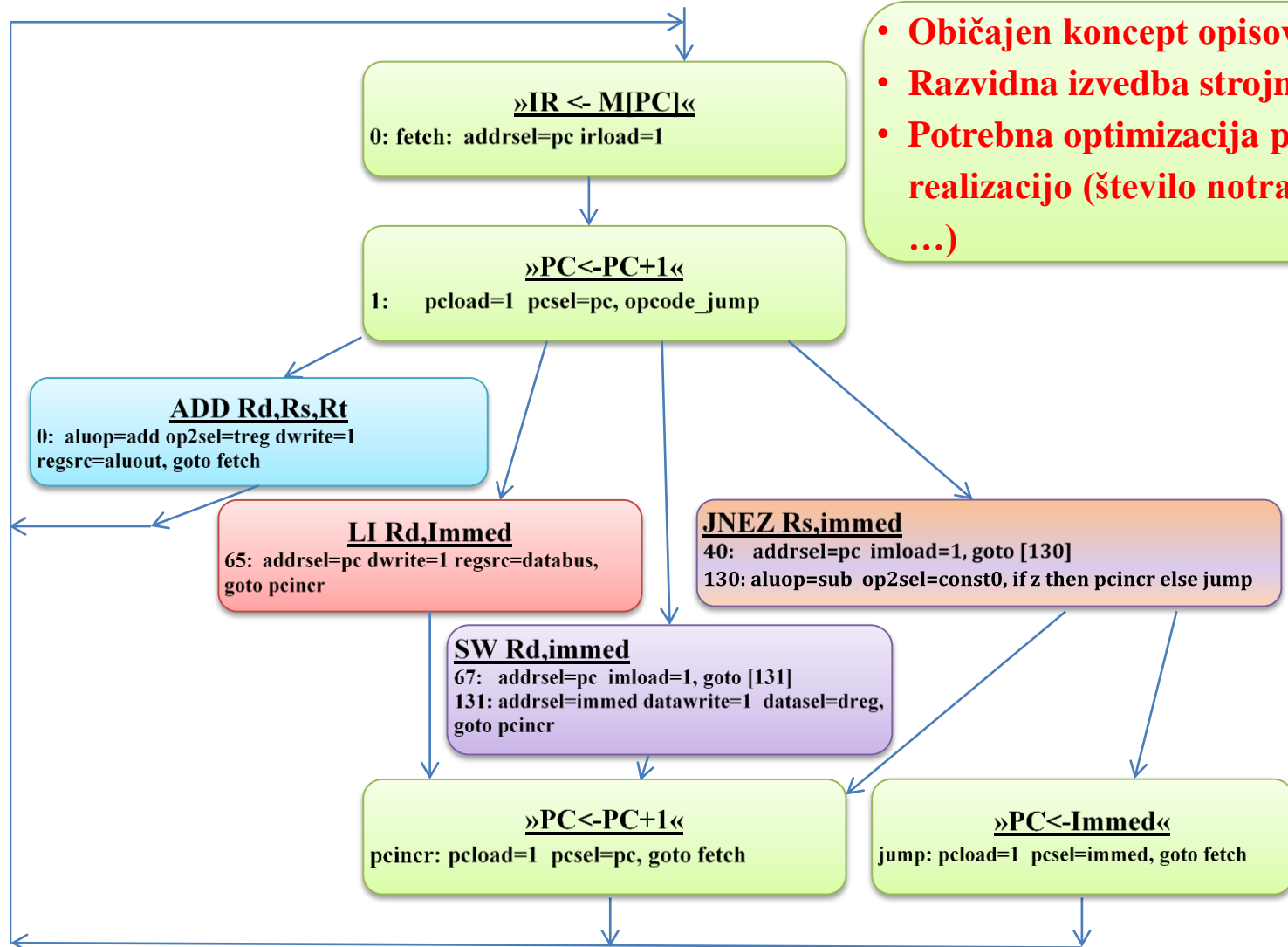
cond – izbira pogoja (Z, ZorN,N,C)

indexsel – opcode_jump

(skok na $opcode + uPC$)
 $ucounter(uPC)=2$, torej
skok na **opcode + 2**



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

kontrolni signali, nasl. mikroukaz

- 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 | z, norz, n, c | kontr. enota |

micro_assembler.pl

```
#!/usr/bin/perl
use strict;
use warnings;

# Microassembler for Warren's 16-bit microcontrolled CPU.
# (c) GPL3 Warren Toomey, 2012

die("Usage: $0 inputfile\n") if (@ARGV!=1);

# Table of control ROM values for the
# known control=value pairs
my %Cvalue= (
    'aluop=add' => 0,
    'aluop=sub' => 1,
    'aluop=mul' => 2,
    'aluop=div' => 3,
    'aluop=rem' => 4,
    'aluop=and' => 5
```

Mikro-zbirnik

datoteka `basic_microcode.def`

```
fetch:  addrsel=pc irload=1          # Address=PC, Load IR register
        pload=1 ptsel=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: pload=1 ptsel=pc, goto fetch

jump:   pload=1 ptsel=immed, goto fetch
```

se prevede v

`indexsel, pload`

```
00: 00002000 0101 # fetch: addrsel=pc irload=1
01: 00008080 0202 #          pload=1 ptsel=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: pload=1 ptsel=pc, goto fetch
85: 00000a00 0000 # jump:   pload=1 ptsel=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: | pclload=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: pclload=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: pclload=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«**
- vsebine ROM pomnilnikov se shranjujejo skupaj z modelom

Pozor na verzijo mikro-zbirnika (1.vrst. Izpisa): micro_assembler.exe v2 (11/2023)

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 ->

assembler.pl

```
#!/usr/bin/perl
use strict;
use warnings;

# Assembler for Warren's 16-bit microcontrolled CPU.
# (c) GPL3 Warren Toomey, 2012
# v0 : Original file
# v1 : Bug fixed: X processing option (11/2017)
#       few instructions' definitions changed
# v2 : corrected bug for swi,lwi ( from 'dX' to 'dsi' )

die("Usage: $0 inputfile\n") if (@ARGV!=1);

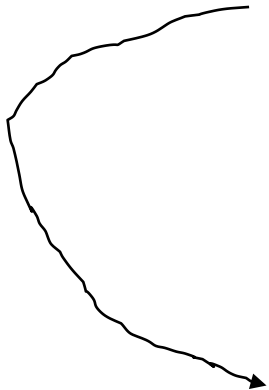
# Table of opcode names, the values
# and their arguments
# Meaning of abbreviations in %Opcode:
# D-reg           if ($atype eq 'd') {
# D-reg, S-reg is D-reg if ($atype eq 'D') {
# S-reg           if ($atype eq 's') {
# T-reg           if ($atype eq 't') {
# Absolute immediate if ($atype eq 'i') {
# Relative immediate if ($atype eq 'I') {
```


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 | | <u>jnez</u> | r1, loop |
| 0009: | 00000006 | 0000000000000010 | | | |
| 000a: | 00008200 | 1000001000000000 | | <u>sw</u> | r0, 256 |
| 000b: | 00000100 | 0000000010000000 | | | |
| 000c: | 00005010 | 0101000000001000 | inf: | <u>jnez</u> | r2, <u>inf</u> |
| 000d: | 0000000c | 0000000000001000 | | | |

| Op.koda | Treg | Sreg | Dreg |
|---------|------|------|------|
| 7 | 3 | 3 | 3 |

Zbirnik – primeri ukazov

list_of_instructions.txt (distribucija) :

rdeče: trenutno že implementirani ukazi v modelu MiMo.

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

MiMo – Model Mikroprogramirane CPE v0.5a

| 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 | pload | 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« | pload=1 pcsel=pc, opcode_jump | 2 | 2 | |
| 2 | | | | | 2 | | 1 | | | | | 0 | 0 | 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 | | | | | 0 | | 1 | | | 0 | | | | 63: | LI Rd,Immed | Rd<-immed<-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 | | | | 0 | | | | | | | | 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 | 0 | | | | pcincr: | PC++, goto fetch: | PC<-PC+1, goto fetch: | pload=1 pcsel=pc, goto fetch | 0 | 0 | |
| 133 0x85 | | | | | | | | | 1 | 1 | | | | jump: | PC<-immed, goto fetch: | immed->PC, goto fetch: | pload=1 pcsel=immed, goto fetch | 0 | 0 | |

datasel:

- 0..PC
- 1..Dreg
- 2..Treg
- 3..ALU

regsrc:

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

pcsel:

- 0..PC+1
- 1..IMM
- 2..PC+IMM
- 3..Sreg

addrsel:

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

op2sel:

- 0..Treg
- 1..IMM
- 2..”0”
- 3..”1”

cond:

- 0..z
- 1..norz
- 2..n
- 3..c

aluop:

- 0..+
- 1..-
- 2..*
- 3../
- ...

Format 1:

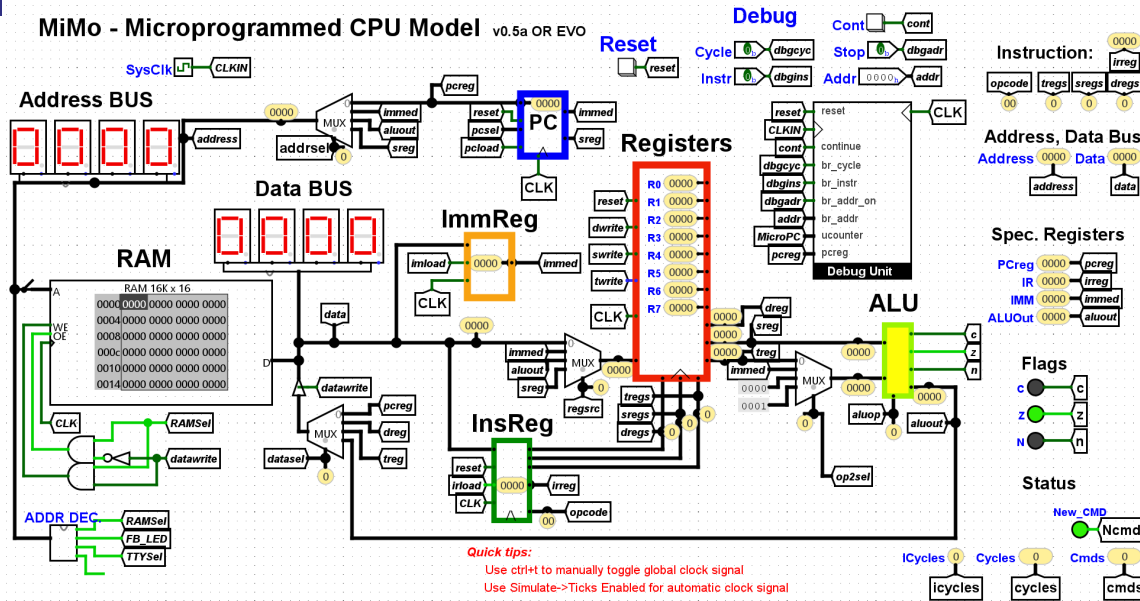
| Op.koda | Treg | Sreg | Dreg |
|---------|------|------|------|
| 7 | 3 | 3 | 3 |

Format 2:

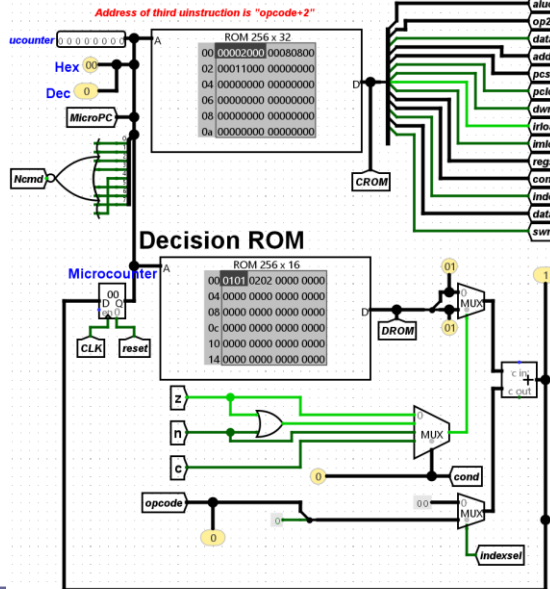
- Format 1 + 16-bitni tak. operand

v 0.5a

MiMo - Microprogrammed CPU Model v0.5a OR EVO



Microcode Control Unit Control ROM



Izvedba strojnega ukaza JNEZ R1,LOOP po skupinah kontrolnih signalov

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

```

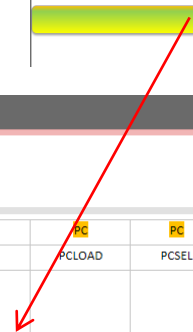
Primer izvedbe ukaza :

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
    
```

Using the Collaboration Space Predavanja Notes +

Izvedba JNEZ
Monday, November 09, 2020 10:11 PM

| Enota ? | KE | KE | ALU | ALU | 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:

40:

pcincr:

jump:

Address=PC, Load IR register

PC=PC+1, jump to 2+OPC

Read Immediate operand -> IMRegister

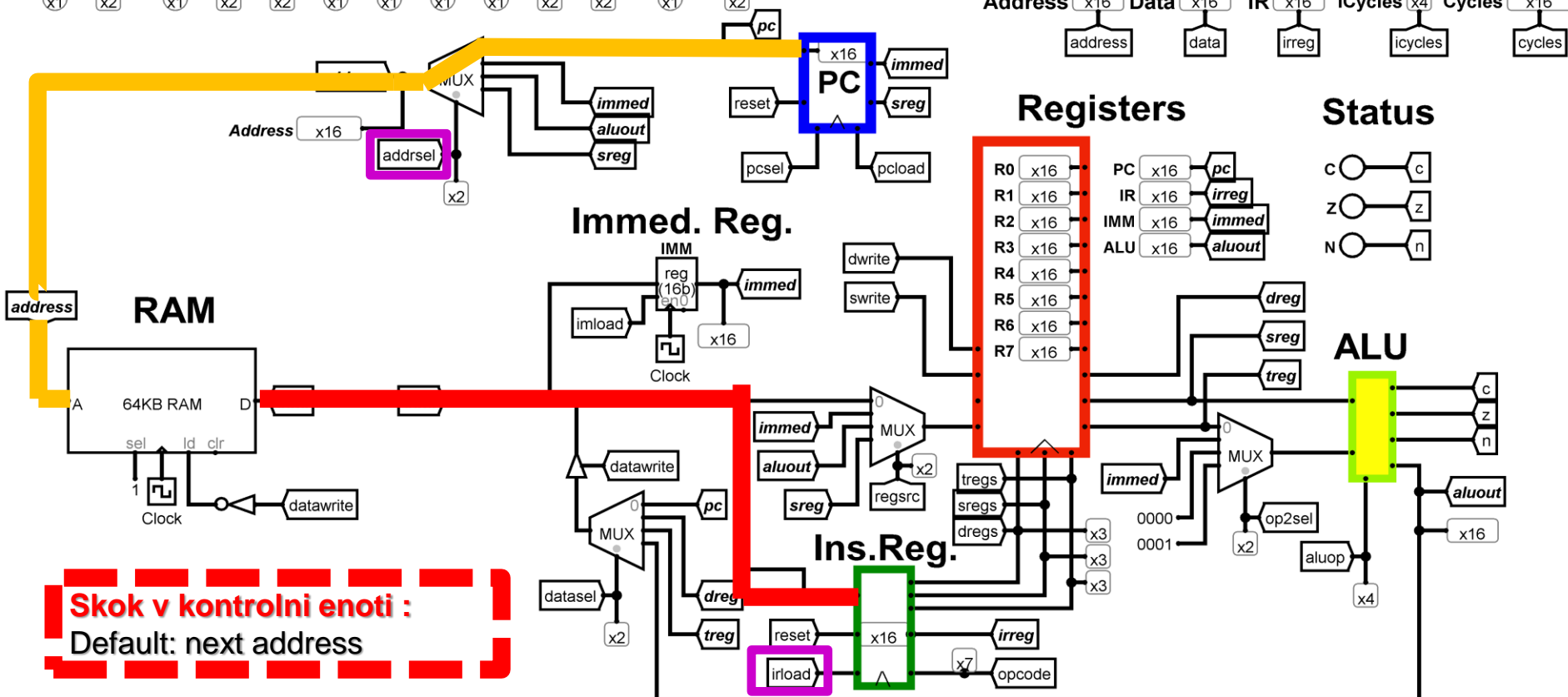
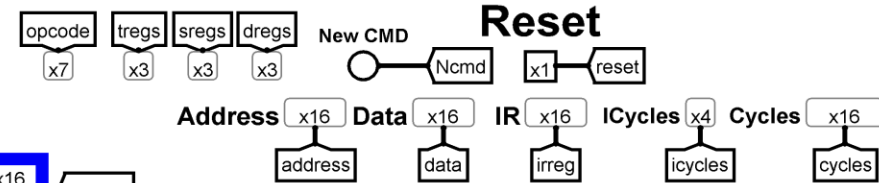
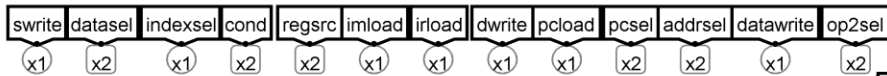
ALU: Rs-0, If z then pcincr else jump

Increment PC and goto new command;

Set address to immed and goto new command

MiMo - Microprogrammed CPU Model v03a

Microinstruction



Skok v kontrolni enoti :
 Default: next address

JNEZ Rs,immed:

jnez Rs,immed (40)

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

fetch:

40:

pcincr:

jump:

Address=PC, Load IR register

PC=PC+1, jump to 2+OPC

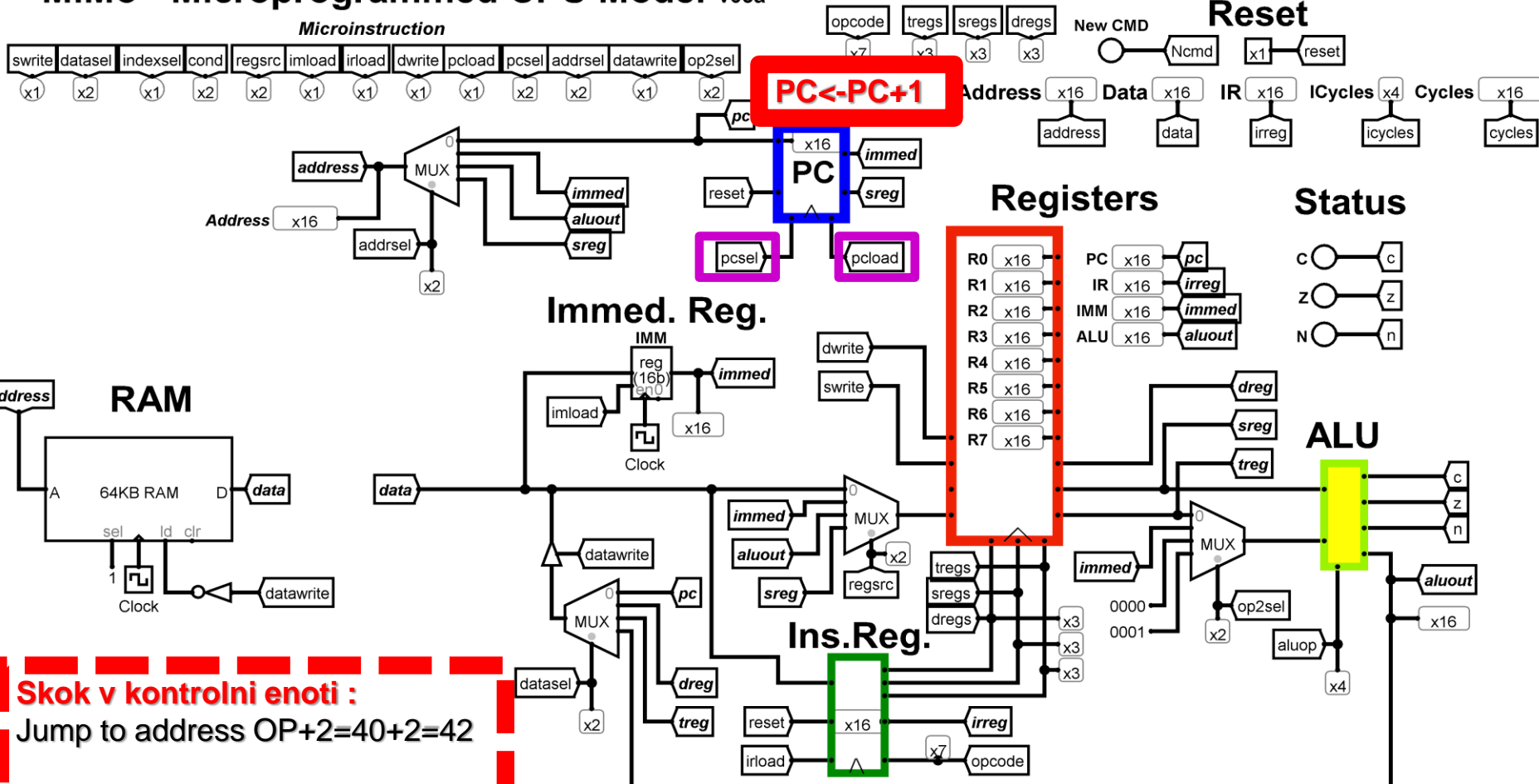
Read Immediate operand -> IMRegister

ALU: Rs-0, If z then pcincr else jump

Increment PC and goto new command;

Set address to immed and goto new command

MiMo - Microprogrammed CPU Model v03a



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:

40:

pcincr:

jump:

Address=PC, Load IR register

PC=PC+1, jump to 2+OPC

Read Immediate operand -> IMRegister

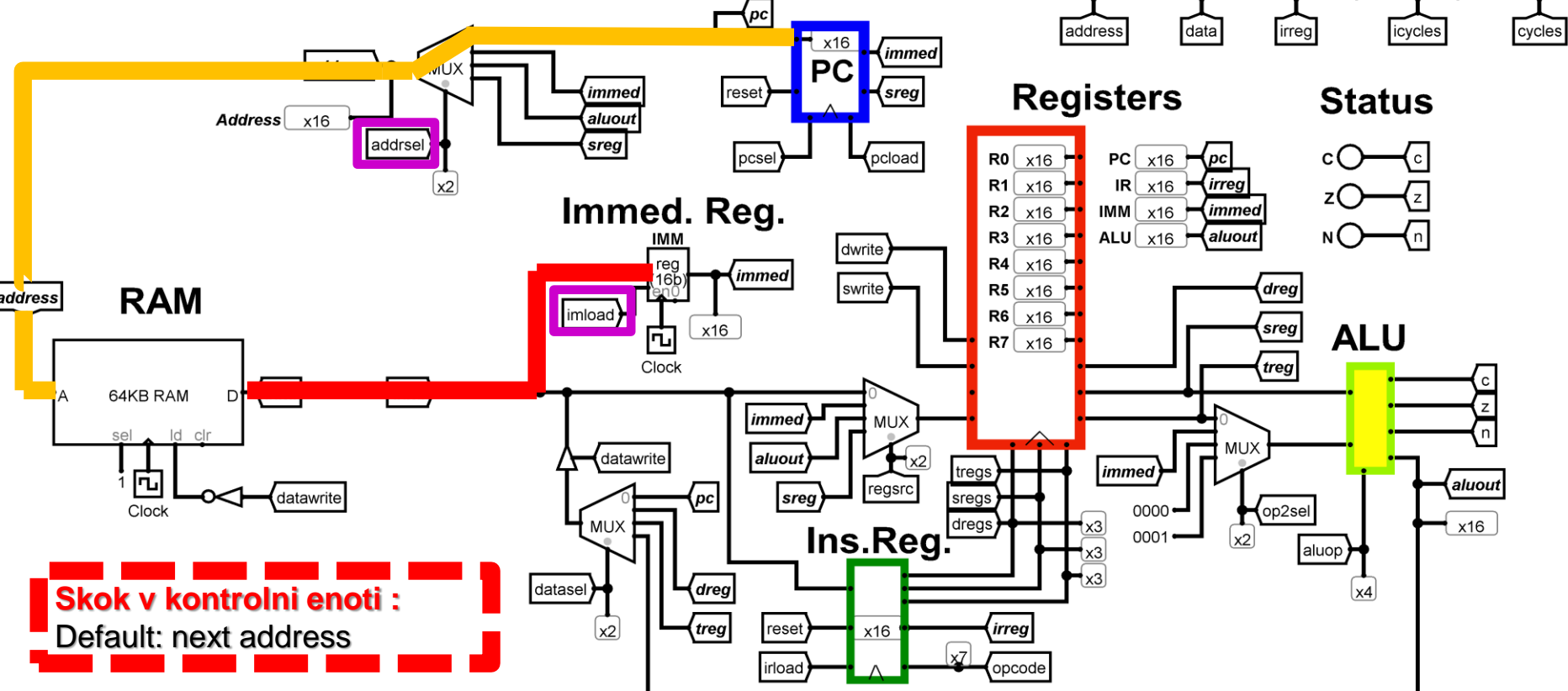
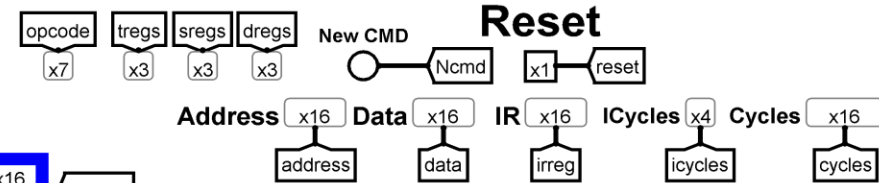
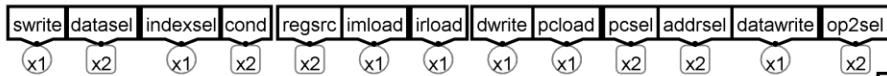
ALU: Rs-0, If z then pcincr else jump

Increment PC and goto new command;

Set address to immed and goto new command

MiMo - Microprogrammed CPU Model v03a

Microinstruction



Skok v kontrolni enoti :
Default: next address

JNEZ Rs,immed:

jnez Rs,immed (40)

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

fetch:

40:

pcincr:

jump:

Address=PC, Load IR register

PC=PC+1, jump to 2+OPC

Read Immediate operand -> IMRegister

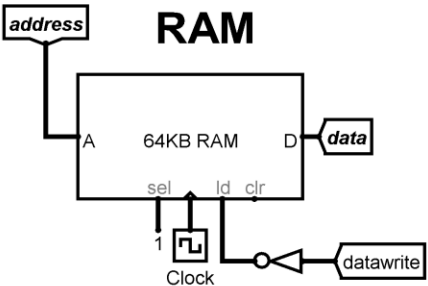
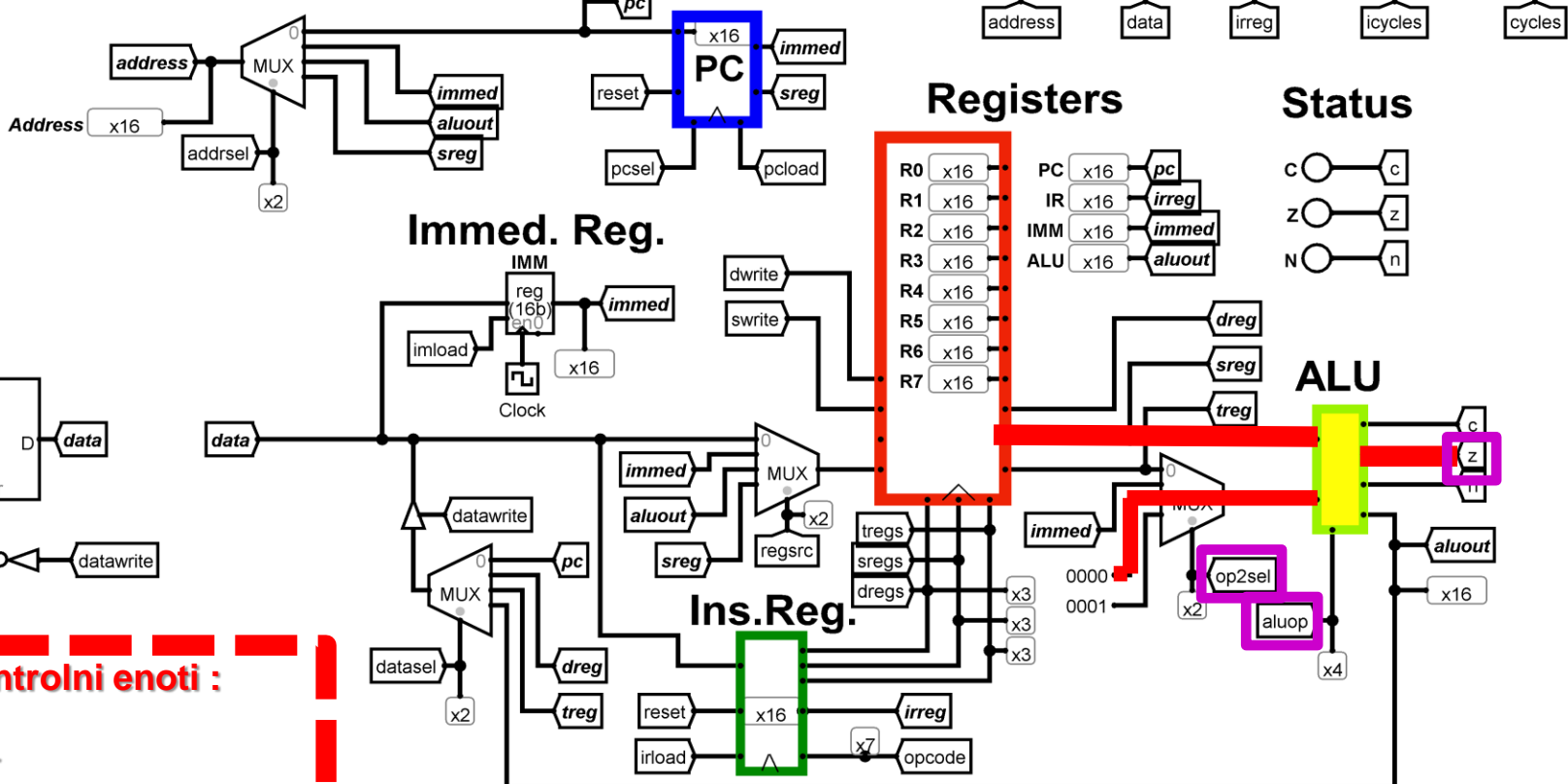
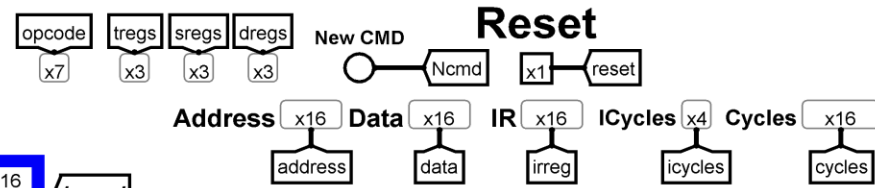
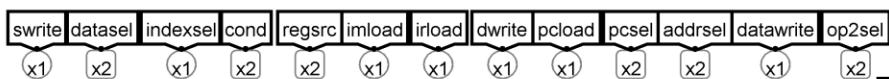
ALU: Rs-0, If z then pcincr else jump

Increment PC and goto new command;

Set address to immed and goto new command

MiMo - Microprogrammed CPU Model v03a

Microinstruction



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:

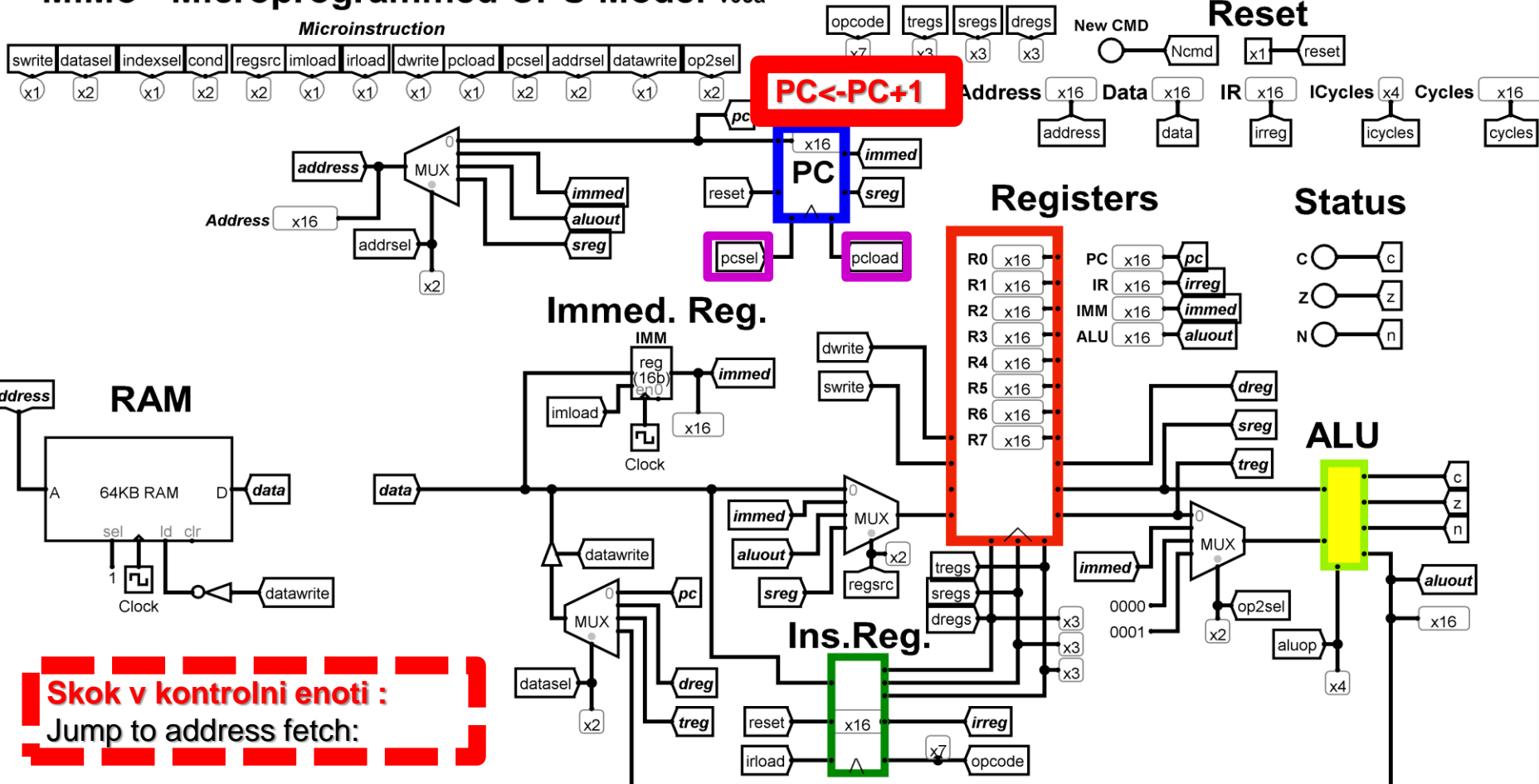
40:

pcincr:

jump:

- # Address=PC, Load IR register
- # PC=PC+1, jump to 2+OPC
- # 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

MiMo - Microprogrammed CPU Model v03a



Skok v kontrolni enoti :
Jump to address fetch:

JNEZ Rs,immed: velja Rs≠0

jnez Rs,immed (40)

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

fetch:

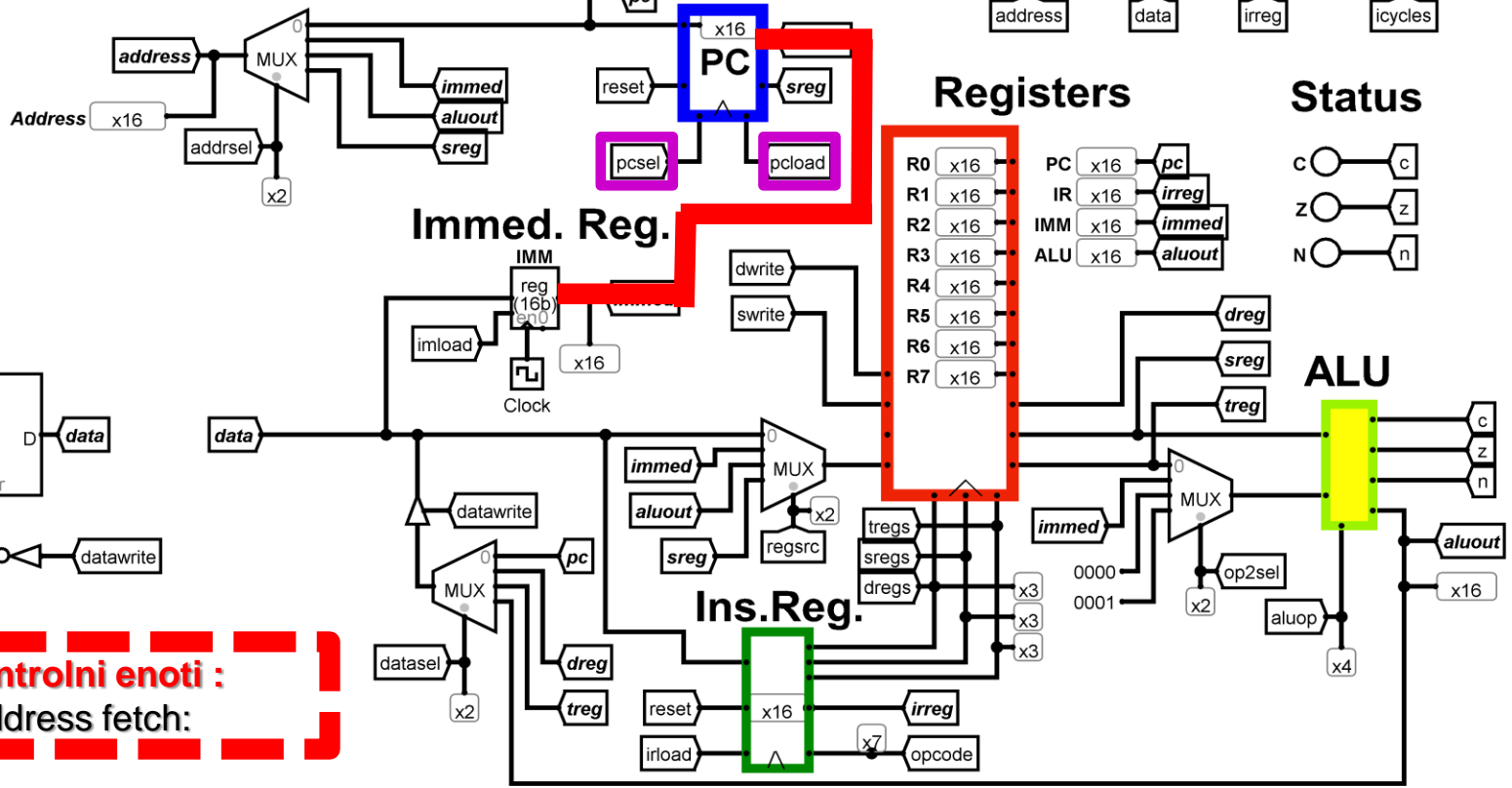
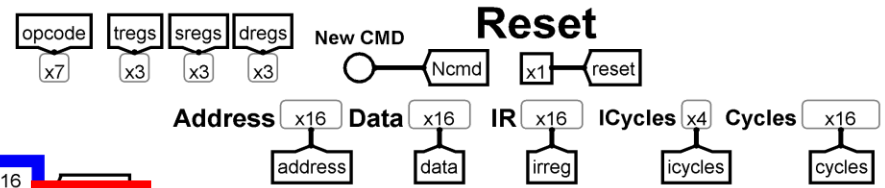
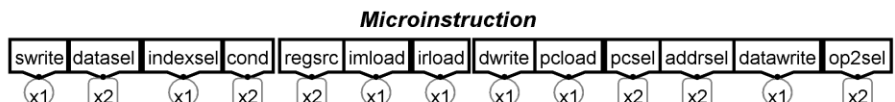
40:

pcincr:

jump:

- # Address=PC, Load IR register
- # PC=PC+1, jump to 2+OPC
- # 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

MiMo - Microprogrammed CPU Model v03a



Skok v kontrolni enoti :
Jump to address fetch:

JNEZ Rs,immed:

fetch: # Address=PC, Load IR register
 40: # PC=PC+1, jump to 2+OPC
 pcincr: # Read Immediate operand -> IMRegister
 jump: # ALU: Rs-0, If z then pcincr else jump
 # Increment PC and goto new command;
 # 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) | CONSTO | | | | | | 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
- Potrebni več logičnih vezij
- Realizacijsko bolj zapletena

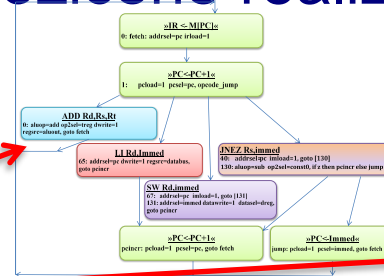
Pristop k realizaciji trdoožičene realizacije KE

Izhodišča:

- Diagram poteka
- Elementarni koraki
- Poenostavitev: CPE, ISA:

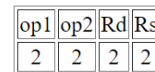
Elementarni koraki:

- 0 .. Branje ukaza -> IR
- 1 .. Format 2: branje operanda
Format 1: nop
- 2 .. Vse operacije :
 - ALE, skok, reg.write, R/W from Mem

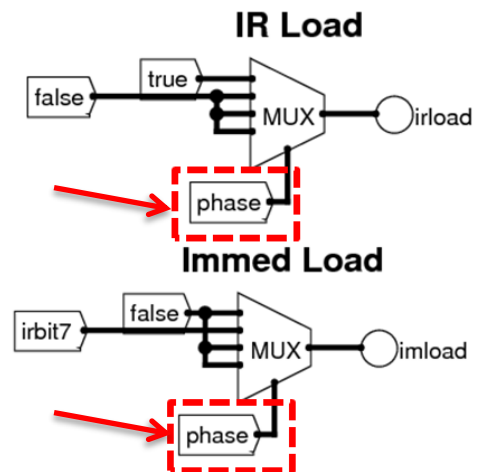
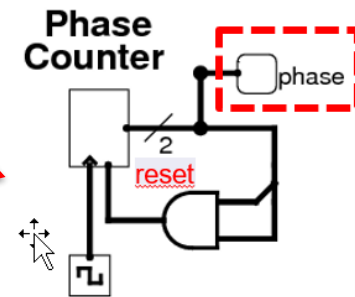
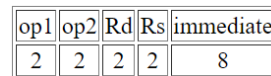


op1 op2
b7b6 b5b4

| op1 | op2 | Mnemonic | Purpose |
|-----|-----|-------------|---------------------|
| 00 | 00 | AND Rd, Rs | Rd = Rd AND Rs |
| 00 | 01 | OR Rd, Rs | Rd = Rd OR Rs |
| 00 | 10 | ADD Rd, Rs | Rd = Rd + Rs |
| 00 | 11 | SUB Rd, Rs | Rd = Rd - Rs |
| 01 | 00 | LW Rd, (Rs) | Rd = Mem[Rs] |
| 01 | 01 | SW Rd, (Rs) | Mem[Rs] = Rd |
| 01 | 10 | MOV Rd, Rs | Rd = Rs |
| 01 | 11 | NOP | Do nothing |
| 10 | 00 | JEQ Rd, imm | PC = imm if Rd == 0 |
| 10 | 01 | JNE Rd, imm | PC = imm if Rd != 0 |
| 10 | 10 | JGT Rd, imm | PC = imm if Rd > 0 |
| 10 | 11 | JLT Rd, imm | PC = imm if Rd < 0 |
| 11 | 00 | LW Rd, imm | Rd = Mem[imm] |
| 11 | 01 | SW Rd, imm | Mem[imm] = Rd |
| 11 | 10 | LI Rd, imm | Rd = imm |
| 11 | 11 | JMP imm | PC = imm |



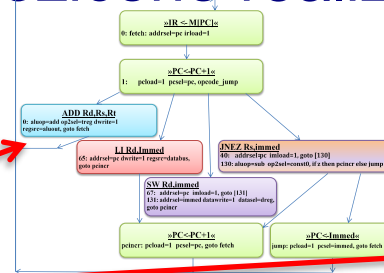
Irbit7 (b7):
0 .. 8-bitni ukaz (1bajt)
1 .. 16-bitni ukaz (2 bajta)



Pristop k realizaciji trdoožičene 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:

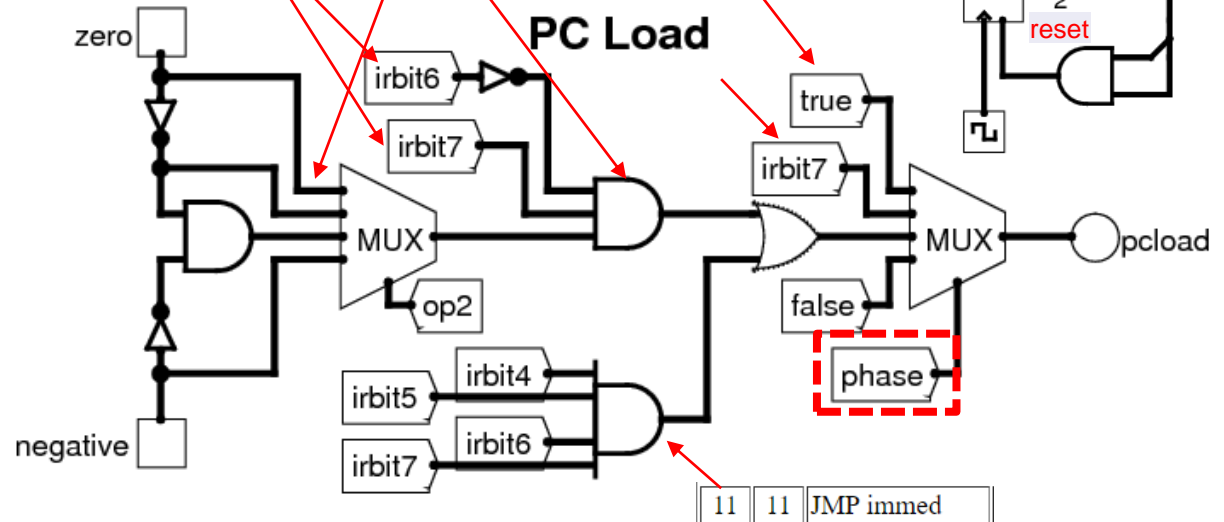
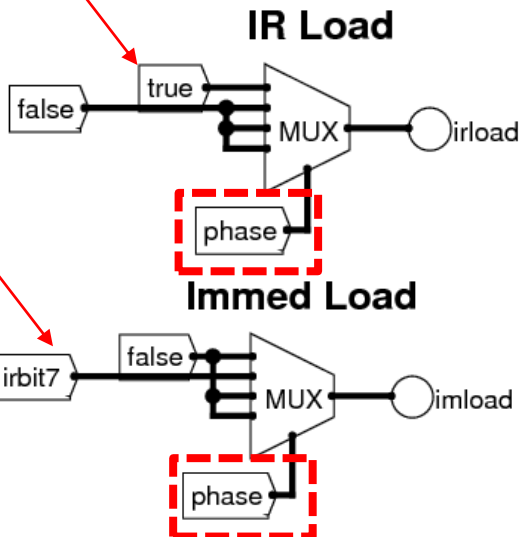
- 0,1 .. enostavna/zapletena ?
- 2 .. enostavna/zapletena ?

irbit7:

- 0 .. 8-bitni ukaz (1 bajt)
- 1 .. 16-bitni ukaz (2 bajta)

b₇b₆ op2

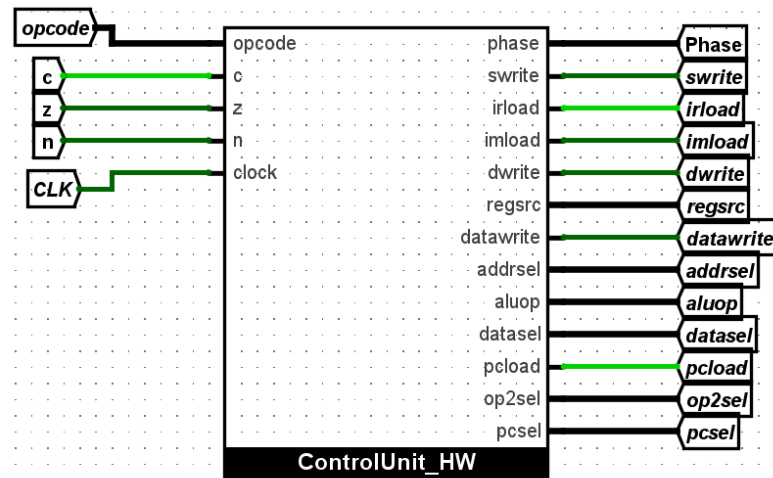
| | | |
|----|----|---------------|
| 10 | 00 | JEQ Rd, immed |
| 10 | 01 | JNE Rd, immed |
| 10 | 10 | JGT Rd, immed |
| 10 | 11 | JLT Rd, immed |



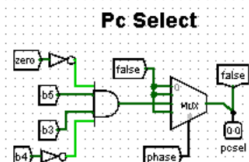
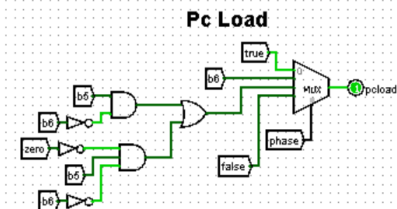
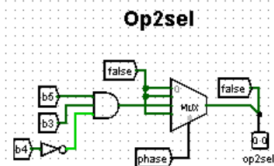
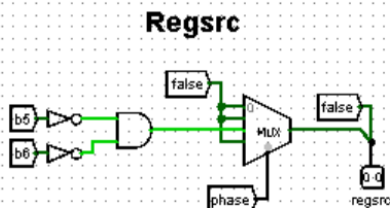
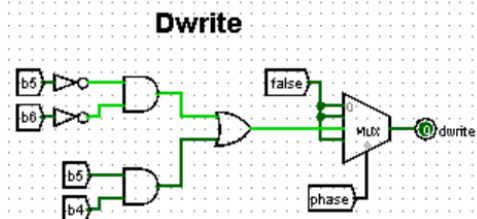
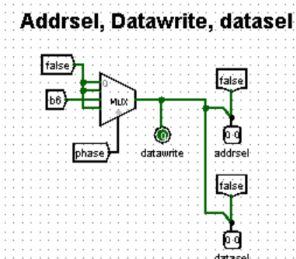
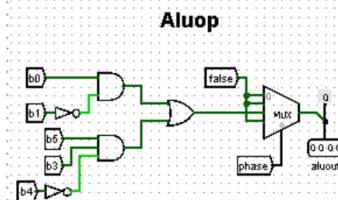
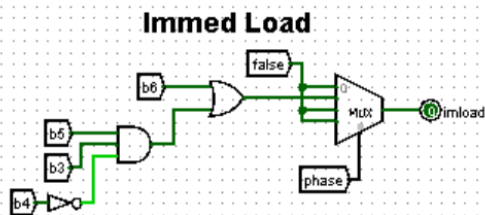
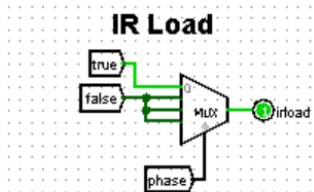
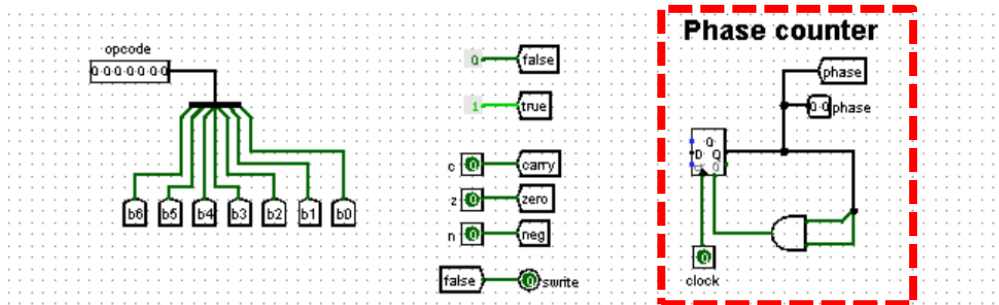
Podrobnejši opis - dodatno gradivo: <http://minnie.tuhs.org/CompArch/Tutes/week03.html>

MiMo: primer trdoožičene realizacije (osnovni ukazi)

HardWired Control Unit



Only 4 initial instructions execution supported !

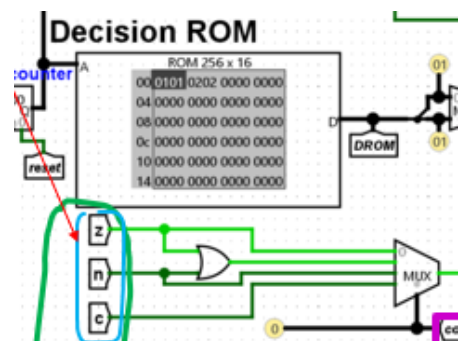


Avtor: Kiril Tofiloski

2023/24 – novosti: MiMo v1 v05a

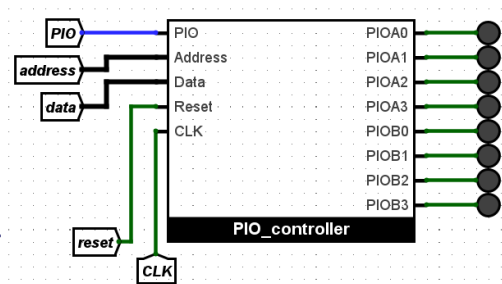
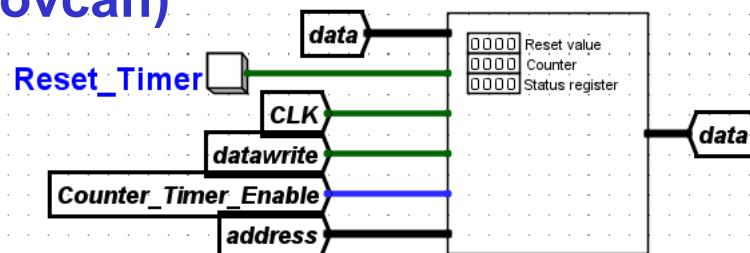
■ Sprememba MiMo v05a:

- vezave zastavic v KE
- mikrozbirnika: `micro_assembler.exe v2 (11/2023)`



■ Dodatni študentski prispevki:

- Timer/Counter : (avtor: Jakob Jelovčan)
- PIO Krmilnik (po FRI-SMS):



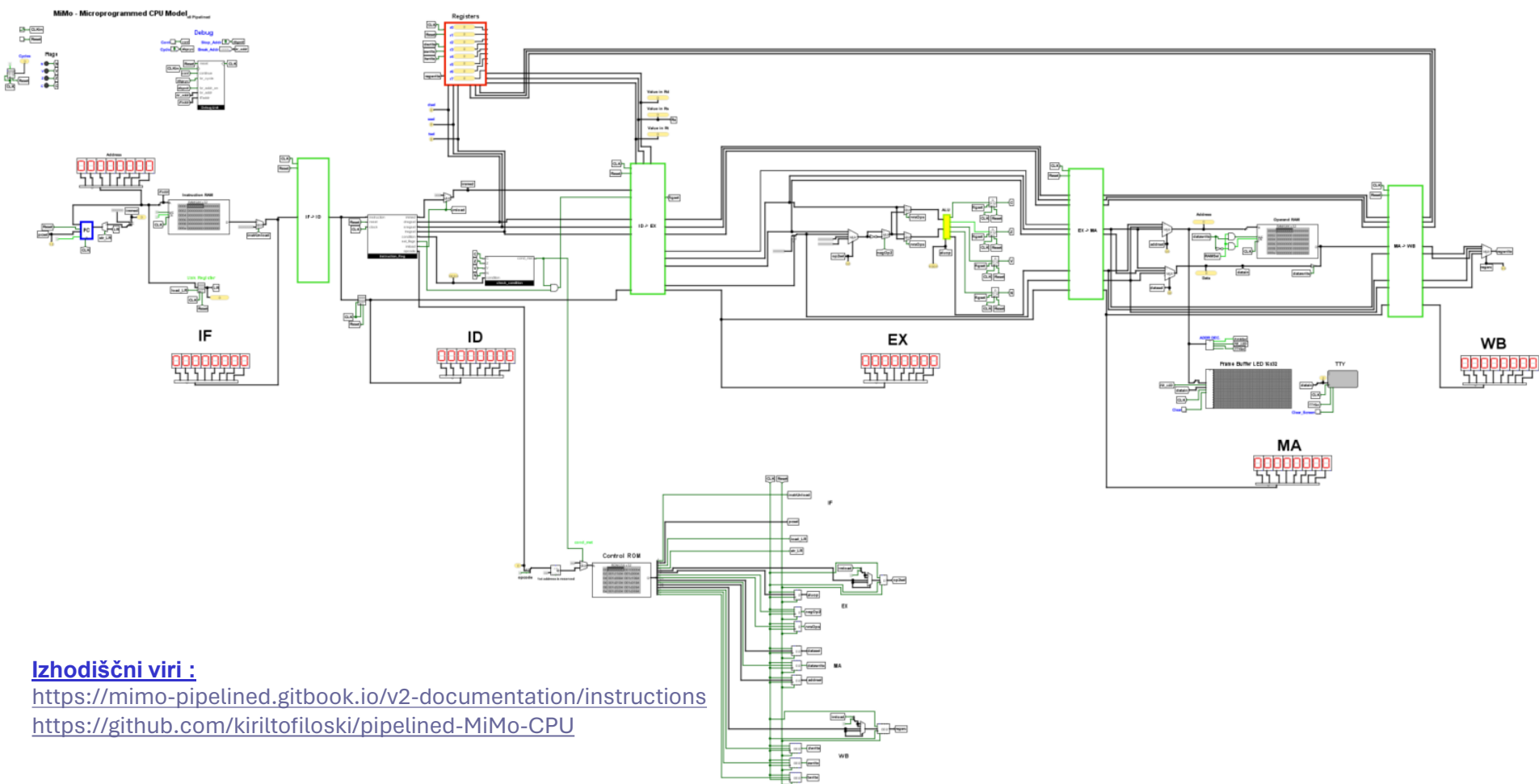
2023/24 – novosti: MiMo v2 (cevovodna različica)

Cevovodne različice MiMo v1 modela:

- mimo_32bit_v2 (osnovna cevovodna izvedba)
- mimo_32bit_v2.1 (v2+zaklenitev)
- mimo_32bit_v2.2 (v2.1+premoščanje)
- mimo_32bit_v2.3 (v2.2+predikcije)

Osnovna distribucija :

https://github.com/LAPSYLAB/MiMo_Student_Release



Izhodiščni viri :

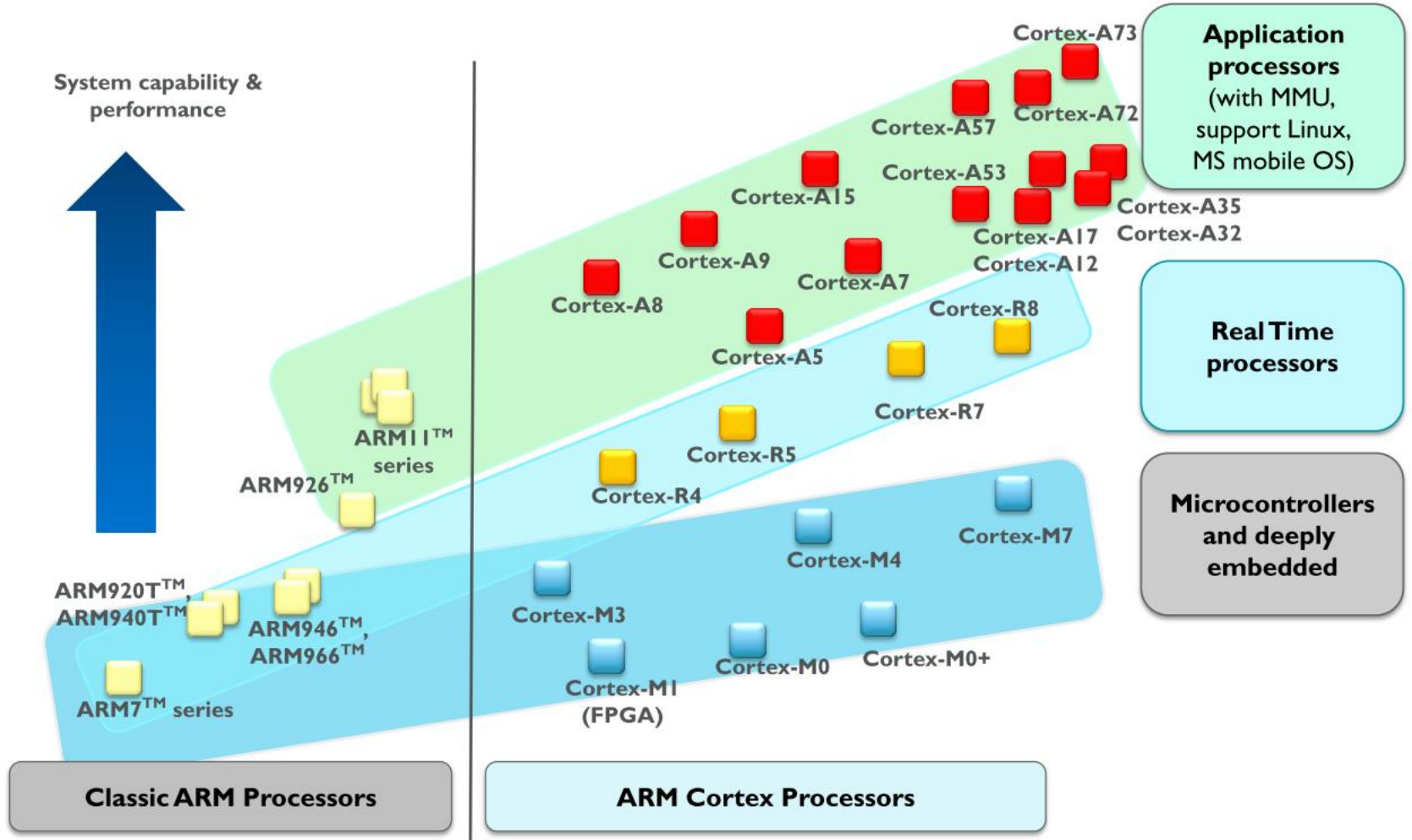
<https://mimo-pipelined.gitbook.io/v2-documentation/instructions>

<https://github.com/kiriltofiloski/pipelined-MiMo-CPU>

3.3 Družina ARM procesorjev

3.3.1 Splošni pregled

„ARM“ : Acorn RISC Machine, Advanced RISC MACHINE



3.3 Družina ARM procesorjev

3.3.1 Splošni pregled

„ARM“ : *Acorn RISC Machine, Advanced RISC MACHINE*

| 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 | <ul style="list-style-type: none"> Atmel 9260 (FRI-SMS) podpora DSP operacijam uvedba tretjega set ukazov (poleg ARM,Thumb) za pospešeno izvajanje Java podpora navideznem pomnilniku |
| ARMv6 | ARM11, Cortex M | MMU Multiprocesiranje SIMD 6 novih status bitov | ARM11MP (prvi multiproc.) <ul style="list-style-type: none"> 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) : <ul style="list-style-type: none"> A5, A8(MPE), A9(MPE,MP), A15,A17 (MPE,VE,LPAAE,VFPv4) Cortex M (MicroController) : <ul style="list-style-type: none"> 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.
- več načinov naslavljanja
- več 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 (večja gostota !)
- novejša Thumb2 ukazna arhitektura (16 ali 32 bitni ukazi)

3.3.2 ARM Cortex-A družina procesorjev

Namen : Kompleksnejše aplikacije, multimedija

NEON (advanced SIMD) ali **MPE** («Media Processing 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 @ $f_{cpe} = 10\text{MHz}$, AMR @ $f_{cpe} = 13\text{MHz}$,
 - MPEG4 VGA 30fr/sec @ $f_{cpe} = 275\text{ MHz}$
 - H.264 video @ $f_{cpe} = 350\text{MHz}$



Smartphones



Automotive and ADAS Systems



Server and networking



Wearables



Tablets and Readers



Set-top and satellite receivers



Home gateways



Robotics

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 (32/64 or 64-bit)

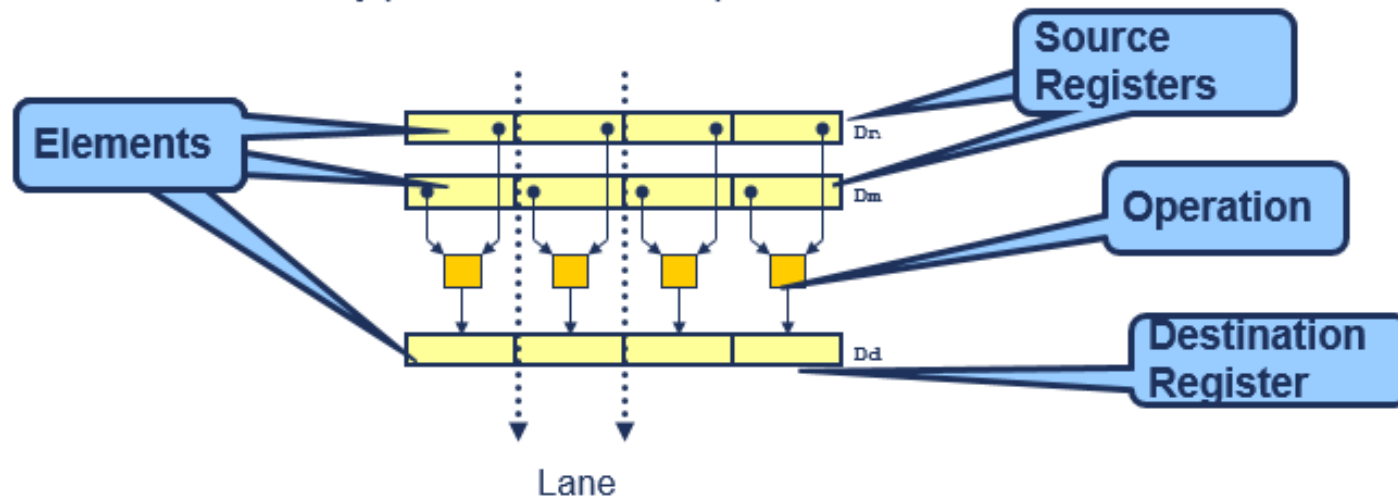
| Year | Core |
|------|-----------------|
| 2012 | Cortex-A53 |
| 2012 | Cortex-A57 |
| 2015 | Cortex-A72 |
| 2015 | Cortex-A35 |
| 2016 | Cortex-A73 |
| 2017 | Cortex-A75 |
| 2018 | Cortex-A76 |
| 2019 | Cortex-A77 |
| 2020 | Cortex-A78 |
| 2021 | Cortex-A510,710 |
| 2022 | Cortex 715 |

| Cortex A8 | Cortex A9 | Cortex A15 | Cortex A76 |
|--|---|---|--|
| <ul style="list-style-type: none"> • ARMv7 (32b) • NEON 64b • VFPv3 • Jazelle RCT (Just in Time byte code apps) superskalarni (dual issue) • dual ALU pipeline | <ul style="list-style-type: none"> • ARMv7 (32b) • MPU • out-of-order with speculative execution • bolj zmogljiv • večji L1, L2 PP • NEON 64b | <ul style="list-style-type: none"> • ARMv7 (32b) • NEON 128b • VFPv4 | <ul style="list-style-type: none"> • ARMv8 (64b) • Napoved skokov • Out-of-order execution • 7nm !!! |

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

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**



NEON vectorizing example

How does the compiler perform vectorization?

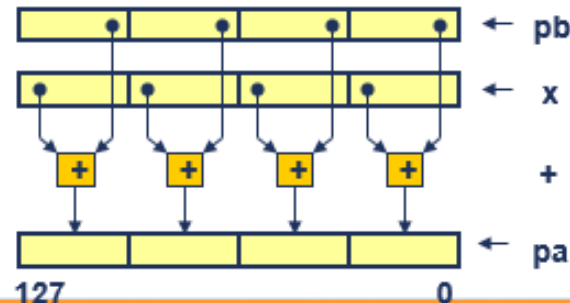
```
void add_int(int * __restrict pa,
            int * __restrict pb,
            unsigned int n, int x)
{
    unsigned int i;
    for(i = 0; i < (n & ~3); i++)
        pa[i] = pb[i] + x;
}
```

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;
    }
}
```

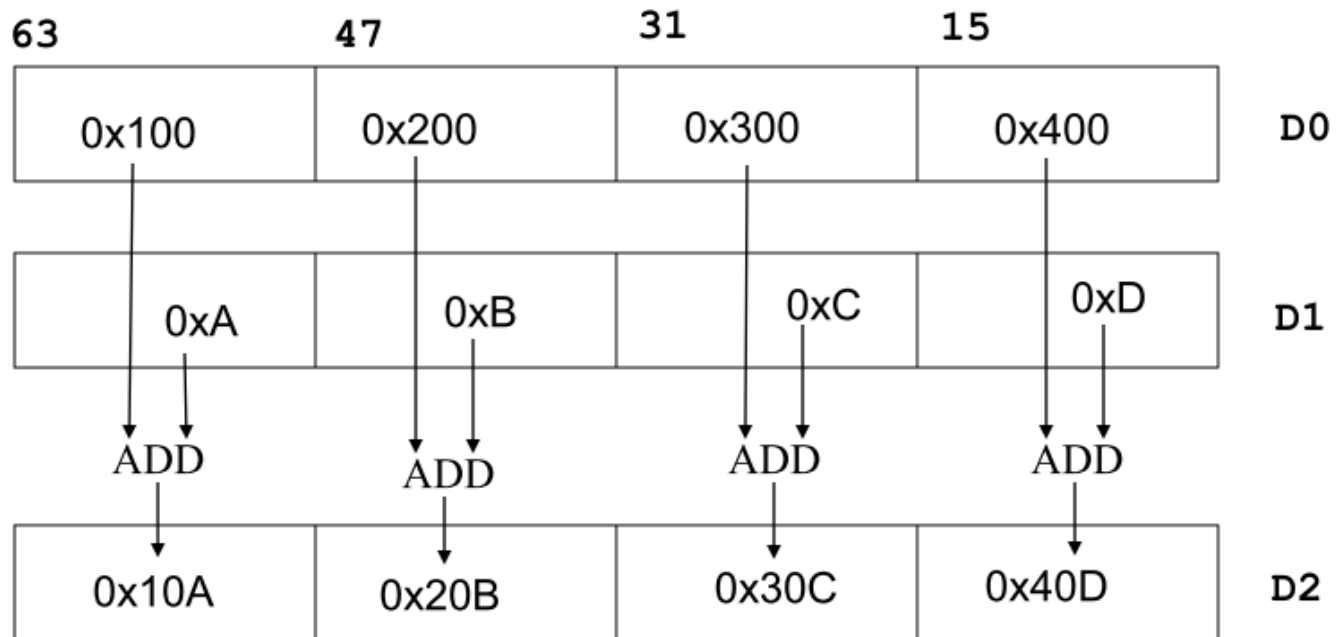


3. Map each unrolled operation onto a NEON vector lane, and generate corresponding NEON instructions

NEON – Advanced SIMD engine (ARM)

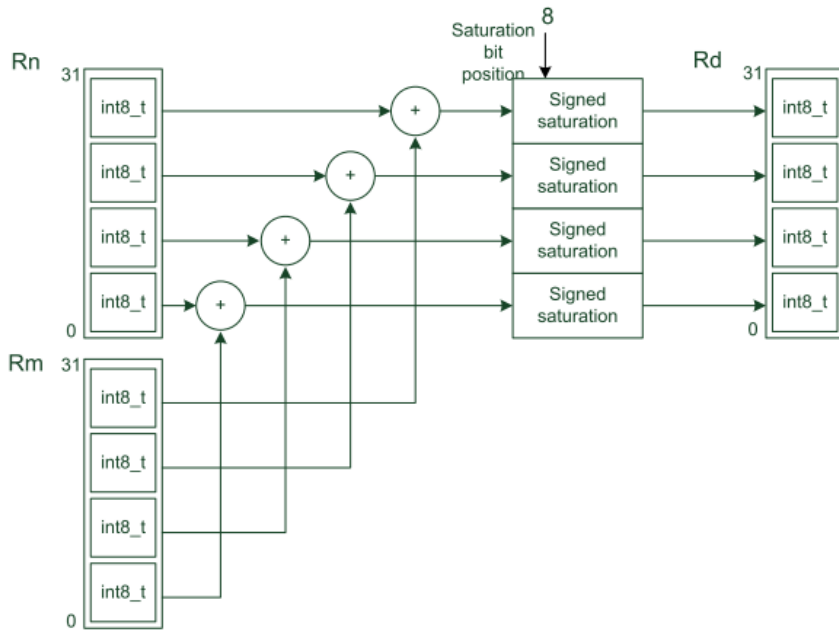
Example SIMD Instruction – Vector ADD

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



ARM Cortex M

QADD8 {<Rd>}, <Rn>, <Rm>



QADD16 {<Rd>}, <Rn>, <Rm>

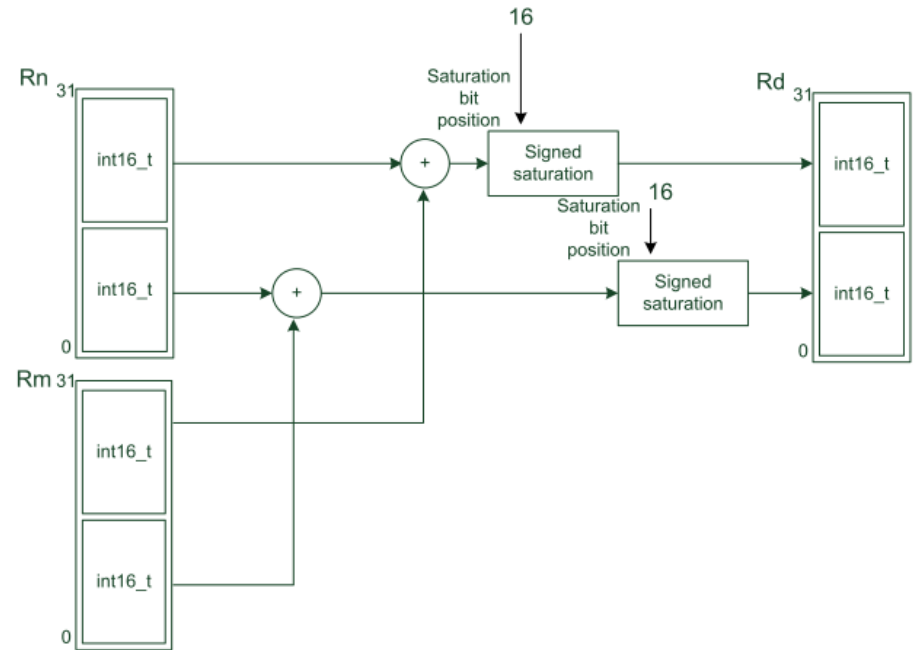


Figure 4: Example of SIMD instructions: QADD8 and QADD16

3.3.2 ARM Cortex-A družina procesorjev

Cortex-A

Cortex-A715

Second-generation Armv9 "big" CPU for best-in-class efficient performance

- The CPU cluster workhorse across "big,LITTLE" configurations.
- Targeted microarchitecture optimizations for 20% power efficiency improvements.
- Consistent performance gains to match Cortex-X1, Arm's first-generation

Cortex-A710

First-generation Armv9 "big" CPU that offers a balance of performance and efficiency

- Addition of Armv9 architecture features for enhanced performance and security.
- Optimal for mobile compute use cases such as smartphones and smart TVs.
- 30% increase in energy efficiency compared to

Cortex-A510

First-generation Armv9 high-efficiency "LITTLE" CPU

- Large performance increases for a highly efficient CPU.
- Innovative microarchitecture upgrades.
- Over 3x uplift in ML performance compared to Cortex-A55.

The technology sector's compute performance needs are ever-expanding to support new applications, experiences, and devices. The **Cortex-X Custom program enables customization and differentiation** beyond the traditional roadmap of Arm Cortex products, offering our partners a way to deliver the **ultimate performance** required for their specific use cases.

The Cortex-X1 design is based on the [ARM Cortex-A78](#), but redesigned for purely performance instead of a balance of performance, power, and area (PPA).^[1]

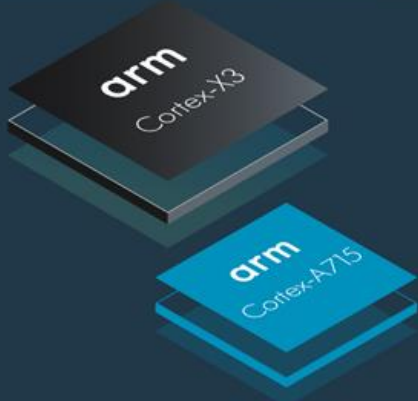
arm

CORTEX-X CUSTOM PROGRAM

3.3.2 ARM Cortex-A družina procesorjev

Premium Sustained Performance and Efficiency for Next-Generation Devices


NEW



+25% Ultimate performance
Peak performance
Compared to Cortex-X2 based smartphones

+20% Balanced for efficiency and performance
Power efficiency
Compared to Cortex-A710

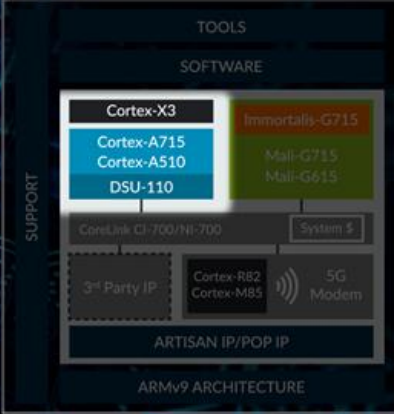
Refreshed



Power enhancements
Up to 5% power reduction
Compared to Cortex-A510 (2021)

Enhanced scalability
Up to 12 Cores

Armv9: Foundation for Total Compute Solutions

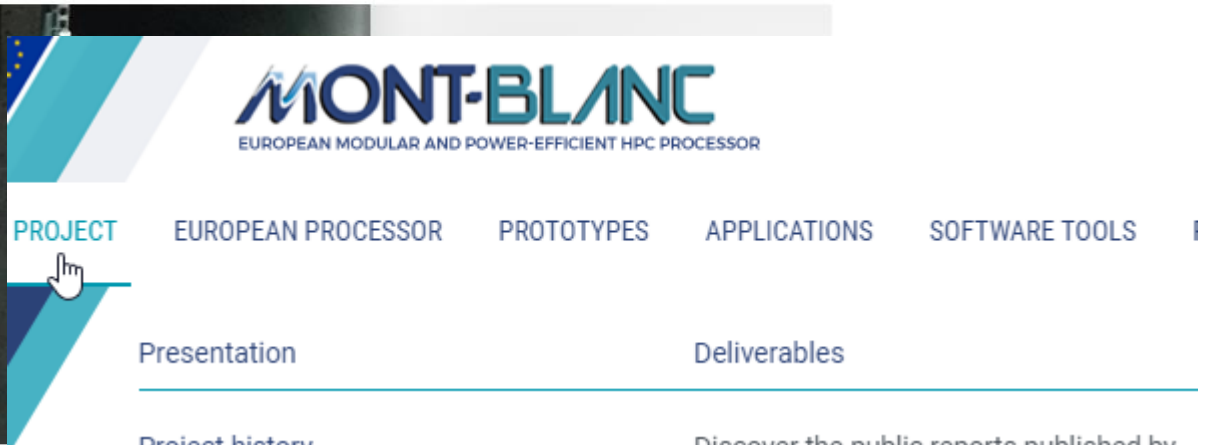


arm

<https://community.arm.com/arm-community-blogs/b/announcements/posts/compute-performance-unleashed>

ARM in High Performance Computing (HPC)

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



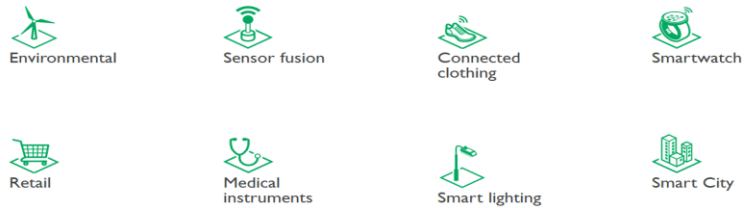
ABOUT MONT-BLANC

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

3.3.3 ARM Cortex-M družina procesorjev

Namen : Mikrokrmilniški sistemi



| 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 <u>cache</u> | Memory architecture | ARM architecture |
|---------------------------|---------------|-------------------------|------------------------------|------------------------------|---------------------------------|---------------------|------------------|
| Cortex-M0 ^[1] | Optional* | Optional ^[9] | No | No | No ^[10] | <u>Von Neumann</u> | 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 | <u>Harvard</u> | 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 |

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) |
| <u>Operating system</u> | <u>Linux</u> (e.g. <u>Raspbian</u>), <u>RISC OS</u> , <u>FreeBSD</u> , <u>NetBSD</u> , <u>Plan 9</u> , <u>Inferno</u> , <u>AROS</u> |
| <u>CPU</u> | 700 <u>MHz</u> single-core <u>ARM1176JZF-S</u> (model A, A+, B, B+, CM) ^[2] |
| Memory | 256 <u>MB</u> ^[3] (model A, A+, B rev 1) 512 MB (model B rev 2, B+, CM) |
| Storage | <u>SDHC</u> slot (model A and B), <u>MicroSDHC</u> slot (model A+ and B+), 4 <u>GB eMMC</u> IC chip (model CM) |
| Graphics | <u>Broadcom VideoCore IV</u> ^[2] |
| Power | 1.5 <u>W</u> (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



3.3.4 Raspberry Pi 2- RPi2

Raspberry Pi 2 model B

| | |
|-------------------------|--|
| Release date | February 2015; 9 months ago |
| Introductory price | US\$35 |
| <u>Operating system</u> | Same as for Raspberry Pi 1 plus Windows 10 IoT Core^[4] and additional distributions of Linux such as Ubuntu |
| <u>CPU</u> | 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.



3.3.4 Raspberry Pi 3

Raspberry Pi 3 model B

| | |
|-------------------------|--|
| Release date | 29 February 2016; |
| Introductory price | US\$35 |
| <u>Operating system</u> | Same as for Raspberry Pi 1 plus Windows 10 IoT Core^[4] and additional distributions of Linux such as Ubuntu |
| <u>CPU</u> | 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 5

•The **Raspberry Pi 5** was announced on September 28, 2023.^[26] Improvements in hardware and software reportedly make the Pi 5 more than twice as powerful as the Pi 4.^[27] It comes with a processor designed in-house, a power button, and an RTC chip, among other things. The RTC chip needs a battery, which can be purchased, but it saves a Pi user the cost of the chip. Unlike the Pi 4, it was released with either 4 or 8 GB of RAM. The 4 GB model costs \$60 USD and the 8 GB model costs \$80 USD. An important thing to note is that it lacks a 3.5 mm audio/video jack. Users must use Bluetooth, HDMI or USB audio if they want to hear sound out of the Pi 5.

The **Raspberry Pi 5** uses the Broadcom BCM2712 SoC, which is a chip designed in collaboration with Raspberry Pi. The SoC features a quad-core [ARM Cortex-A76](#) processor clocked at 2.4 GHz, alongside a VideoCore VII GPU clocked at 800 MHz. The BCM2712 SoC also features support for [cryptographic extensions](#) for the first time on a Raspberry Pi model. Alongside the new processor and graphics unit, the monolithic design of the earlier BCM2711 has been replaced with a [CPU and chipset \(southbridge\) architecture](#), as the IO functionality has been moved to the Raspberry Pi 5's custom RP1 chip



Vir: Wikipedia.com

3.3.4 Raspberry Pi

| Family | Model | SoC | Memory | Form Factor | Ethernet | Wireless | GPIO | Released | Discontinued |
|--------------------------------|--------|--------------------------------|--------|------------------------------|----------------------|--------------------|----------------------|----------|----------------------------|
| Raspberry Pi | B | BCM2835 | 256 MB | Standard ^[a] | Yes | No | 26-pin | 2012 | Yes (????) |
| | | | 512 MB | | 2012 ^[38] | | | | |
| | 256 MB | | No | | 2013 | | | | |
| | B+ | | 512 MB | Yes | 2014 | | | | |
| | A+ | | | Compact ^[b] | No | | | | |
| Raspberry Pi 2 | B | BCM2836 / 7 | 1 GB | Standard ^[a] | Yes | No | 2015 | No | |
| Raspberry Pi Zero | Zero | BCM2835 | 512 MB | Ultra-compact ^[c] | No | No | 2017 | | |
| | W / WH | | | | | | 2021 | | |
| | 2 W | BCM2710A1 ^{[d][39]} | | | | Yes | 2016 | | |
| Raspberry Pi 3 | B | BCM2837A0 / B0 | 1 GB | Standard ^[a] | Yes | Yes | 2018 | | |
| | A+ | BCM2837B0 | 512 MB | Compact ^[b] | No | Yes ^[e] | 2018 | | |
| | B+ | | 1 GB | Standard ^[a] | Yes ^[f] | | 2018 | | |
| Raspberry Pi 4 | B | BCM2711B0 / C0 ^[40] | 1 GB | Standard ^[a] | Yes ^[g] | Yes ^[e] | 2019 ^[41] | | Yes (2020) ^[42] |
| | | | 2 GB | | | | 2021 ^[43] | | |
| | | | 4 GB | | | | 2019 ^[41] | | |
| | | | 8 GB | | | | 2020 | | |
| | 400 | | 4 GB | Keyboard | | | No | | |
| Raspberry Pi Pico | Pico | RP2040 | 264 KB | Pico ^[h] | No | No | 2021 | | |
| | W | | | | | Yes ^[i] | 2022 | | |
| Raspberry Pi 5 ^[44] | | BCM2712 | 4 GB | Standard ^[a] | Yes ^[g] | Yes ^[e] | 40-pin | 2023 | |
| | | | 8 GB | | | | | | |

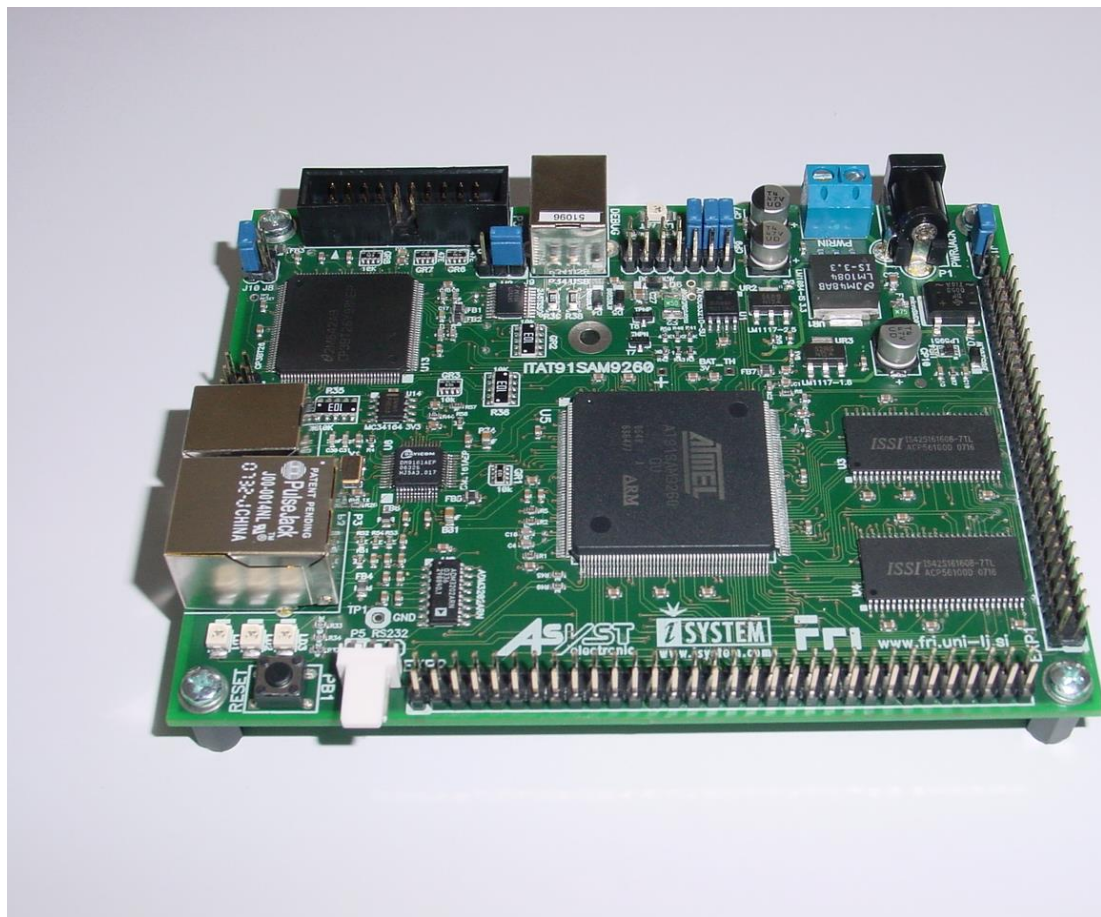
Vir: Wikipedia.com

3.3.4 Mikroračunalniški sistem FRI-SMS

- Mikrokontroler 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 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>

3.3.4 STM32H750B-DK Discovery razvojni sistem

- Arm® Cortex® core-based microcontroller with 128 Kbytes (STM32H750XBH6) of Flash memory and 1 Mbyte of RAM, in TFBGA240+25 package
- 4.3" RGB interface LCD with touch panel connector
- Ethernet compliant with IEEE-802.3-2002, and POE
- USB OTG FS with Micro-AB connector
- SAI audio codec
- One ST-MEMS digital microphone
- 2 x 512-Mbit Quad-SPI NOR Flash memory
- 128-Mbit SDRAM
- 4-Gbyte on-board eMMC
- 1 user and reset push-button
- Fanout daughterboard
- 2 x FDCANs
- Board connectors:
 - USB FS Micro-AB connectors
 - ST-LINK Micro-B USB connector
 - USB power Micro-B connector
 - Ethernet RJ45
 - Stereo headset jack including analog microphone input
 - Audio header for external speakers
 - Arduino™ Uno V3 expansion connectors
 - STMod+



<https://www.st.com/en/evaluation-tools/stm32h750b-dk.html>

3.3.4 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



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

3.3.4 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



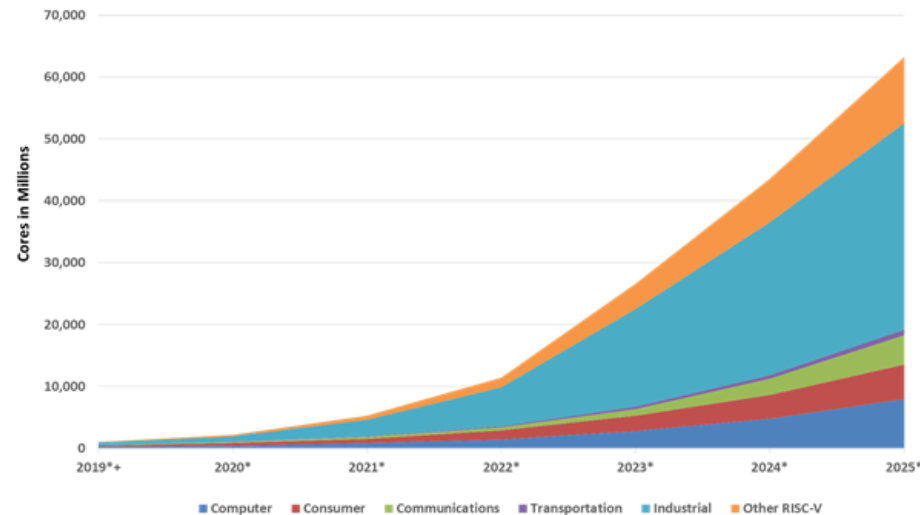
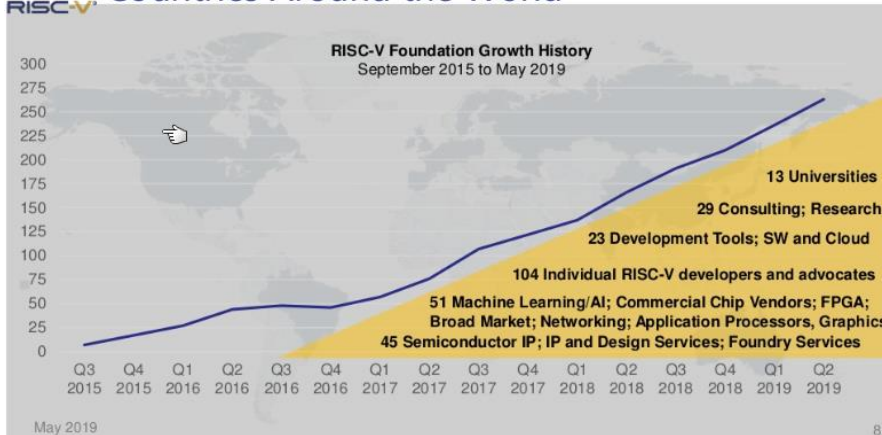
<https://www.st.com/en/evaluation-tools/stm32mp157c-dk2.html>

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.

 More than 250 RISC-V Members in 28 Countries Around the World



Source: Semico Research Corp.