



RAČUNALNIŠKA ARHITEKTURA

3 Osnove delovanja računalnikov



3 Osnove delovanja računalnikov - vsebina

- Von Neumannov računalniški model
 - Von Neumannov računalniški model
 - Delovanje von Neumannovega računalnika
- Flynnova klasifikacija
- Glavni pomnilnik v von Neumannovem računalniku
 - Pomnilniška beseda
 - Pomnilniški naslov
 - Naslovni prostor
 - Vsebina pomnilniške besede
 - Princetonska in harvardska pomnilniška arhitektura
 - Dostop do pomnilnika
- Amdahlov zakon
- Jeziki, nivoji in navidezni računalniki
 - Računalnik kot zaporedje navideznih računalnikov
 - Prehajanje iz jezika J2 v jezik J1
 - Strojna in programska oprema računalnika
- Primer izvedbe programa v računalniku



Osnove delovanja računalnikov - vsebina:



- 3.2 Flynnova klasifikacija
- 3.3 Glavni pomnilnik v von Neumannovem računalniku
- 3.4 Amdahlov zakon
- 3.5 Jeziki, nivoji in navidezni računalniki
- 3.6 Primer izvedbe programa v računalniku

Osnove delovanja računalnikov - cilji:

- Osnovno razumevanje delovanja računalnikov
 - Von Neumannov model in razširitve (paralelnost)
- Nivoji računalniškega sistema (HW <-> SW)
- Razumevanje izvedbe programa



3.1 Von Neumannov računalniški model

- Večinoma so današnji računalniki zgrajeni na osnovi modela računanja, ki je znan pod imenom von Neumannov model (John von Neumann 1945)

- von Neumannov(-a) :

- model računanja,
- računalniški model,
- računalnik,
- arhitektura.

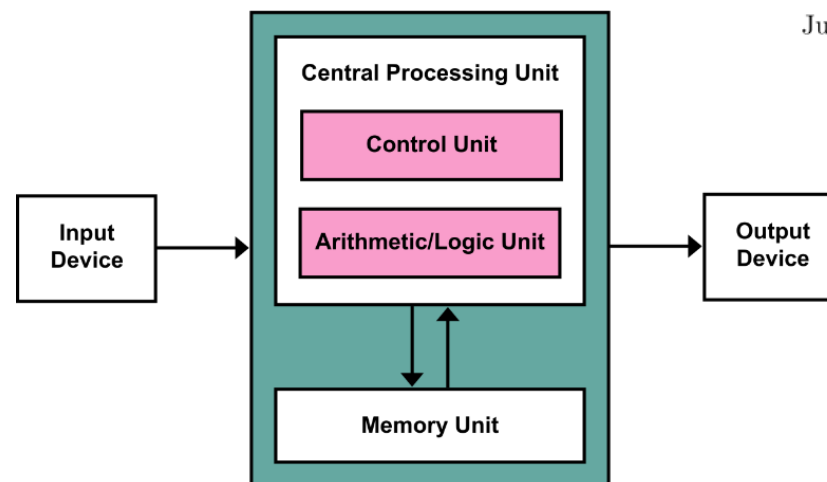
First Draft of a Report
on the EDVAC

by

John von Neumann

Moore School of Electrical Engineering
University of Pennsylvania

June 30, 1945





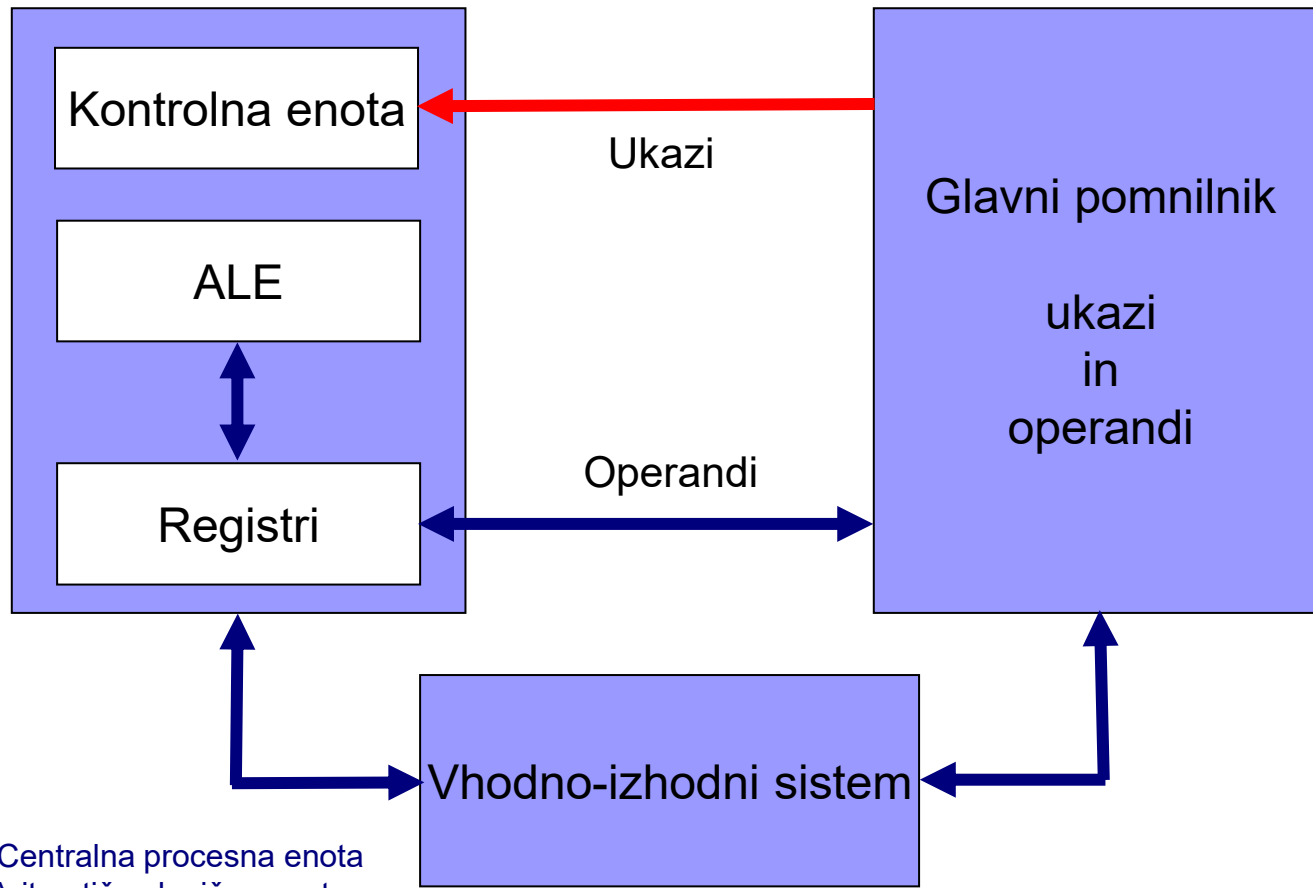
3.1 Von Neumannov računalniški model

- Sestavljajo ga trije osnovni deli:
 - CPE (centralna procesna enota)
 - Glavni pomnilnik
 - Vhodno-izhodni (V/I) sistem
- Je stroj s **shranjenim programom**, ki je shranjen v glavnem pomnilniku. Ukazi v programu določajo, kaj bo stroj delal.
- **Program vodi delovanje stroja** – program določa, kako bo stroj deloval.
- CPE jemlje ukaze iz glavnega pomnilnika in jih izvaja drugega za drugim.



Von Neumannov računalniški model

CPE



CPE – Centralna procesna enota
ALE – Aritmetično logična enota



VN Model: delovanje CPE

- **CPE** iz glavnega pomnilnika jemlje ukaze in jih izvršuje. V današnjih računalnikih je poleg CPE še več procesorjev, zato oznaka **centralna** procesna enota. Sestavljajo jo trije deli:
 - KONTROLNA ENOTA - skrbi za prevzemanje ukazov in operandov in aktiviranje operacij, ki so določene z ukazi.
 - ALE - izvaja aritmetične operacije (seštevanje . . .)
in logične operacije (AND . . .).
 - REGISTRI – več povezanih pomnilniških celic, ki služijo za shranjevanje vrednosti.
 - Programsko nedostopni registri – potrebni za delovanje CPE.
 - Programsko dostopni registri (arhitekturni registri) za shranjevanje operandov. Predstavljajo majhen in hiter pomnilnik v CPE.



VN Model: delovanje pomnilnika, V/I

- **Glavni pomnilnik** je sestavljen iz pomnilniških besed. Vsaka pomnilniška beseda ima svoj enolični naslov.
 - V njem so shranjeni ukazi in operandi.
 - Oznaka glavni zopet služi za razlikovanje od drugih pomnilnikov v današnjih računalnikih (predpomnilniki, navidezni pomnilnik).

- **V/I sistem** za prenos informacije v zunanji svet ali iz zunanjega sveta. Informacija je v CPE in glavnem pomnilniku shranjena v obliki, ki ni dostopna zunanjemu svetu.
 - Sestavni del V/I sistema so vhodno-izhodne naprave, ki pretvarjajo informacijo v neko drugo obliko, ki je primerna za uporabnika ali pa služijo kot pomožni pomnilniki.



Delovanje von Neumannovega računalnika

- Njegovo delovanje popolnoma določajo ukazi (strojni ukazi), ki jih CPE jemlje iz glavnega pomnilnika zaporedoma enega za drugim.
- Strojni ukazi so v pomnilniku shranjeni eden za drugim po naraščajočih naslovih.
- Na neki način je določeno, iz katerega naslova se vzame prvi ukaz po vklopu računalnika ali po pritisku na tipko RESET.
 - Najenostavneje: prva ali zadnja pomnilniška lokacija – najnižji ali najvišji naslov v pomnilniku.



Pri vsakem ukazu razlikujemo dva koraka

- **1. korak:** Jemanje ukaza iz pomnilnika (FETCH)
(tudi branje ali prevzem ukaza)
 - ukazno prevzemni cikel
 - angl. fetch cycle
 - V CPE je poseben register - programski števec (PC - Program Counter), ki vedno vsebuje pomnilniški naslov, na katerem je v pomnilniku shranjen naslednji ukaz.

- **2. korak:** Izvrševanje v 1. koraku prevzetega ukaza (EXECUTE)
 - izvršilni cikel
 - angl. execute cycle

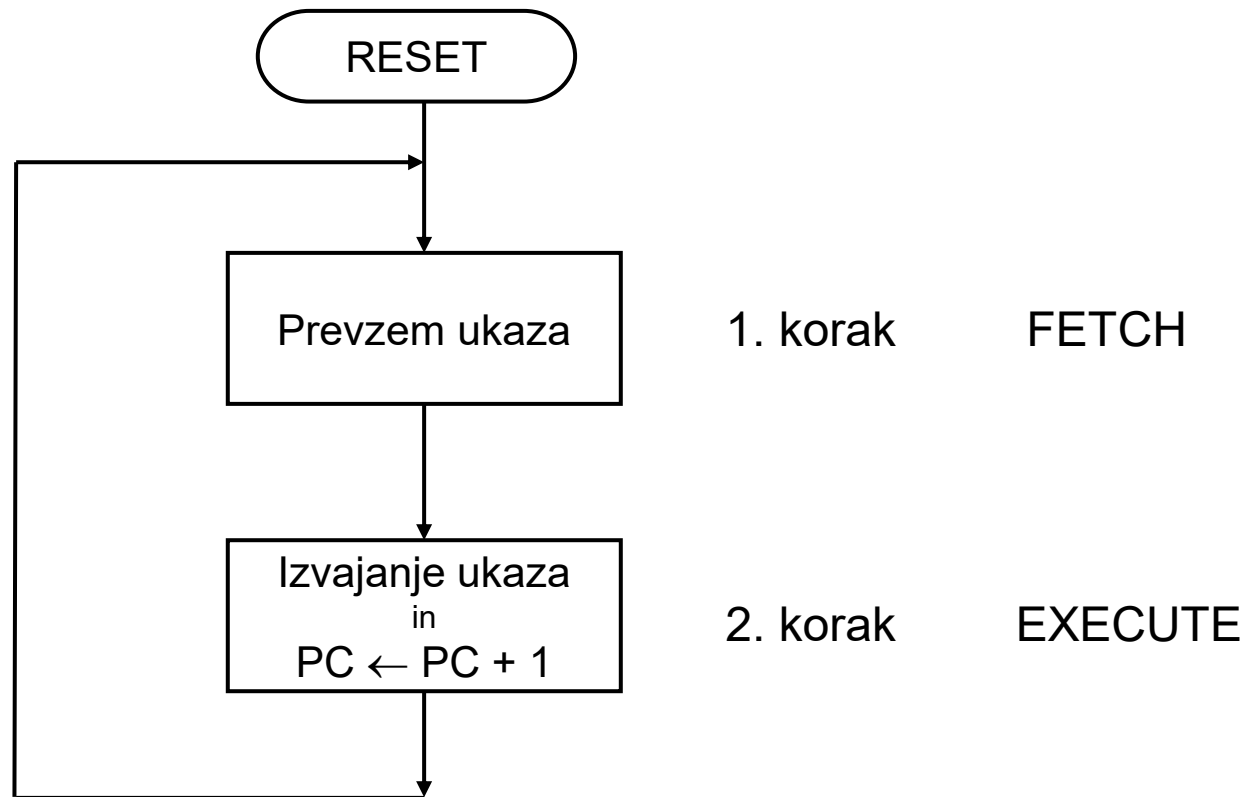


Vsebina ukaza, PC

- Vsak ukaz vsebuje dve vrsti informacij:
 - informacijo o operaciji, ki naj se izvrši,
 - informacije o operandih, nad katerimi naj se operacija izvrši.
- CPE izvrši operacijo in poskrbi, da je v PC naslov naslednjega ukaza tako, da poveča vsebino PC-ja za 1 (ali več).
- **Pravilo:** ukazi v pomnilniku so shranjeni po naraščajočih naslovih zato $PC \leftarrow PC + 1$. To pravilo je rezultat dogovora in določa vrstni red izvajanja ukazov.



2 koraka pri izvedbi ukaza





2 koraka pri izvedbi ukaza - izjemi

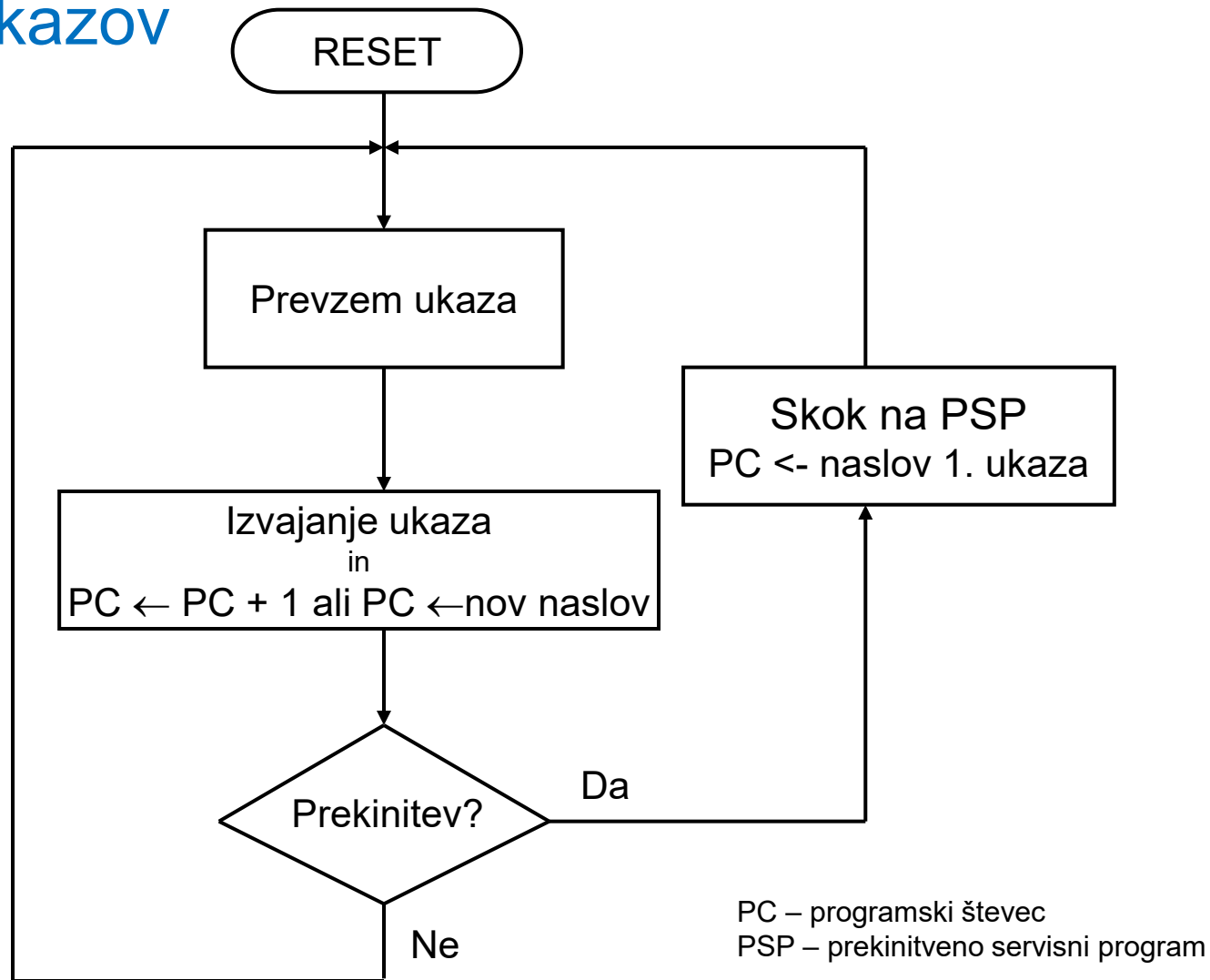
Po zaključku koraka 2 začne CPE zopet s korakom 1.
Ta dva koraka se ponavljata, dokler računalnik deluje.

- **Izjema 1:** Skočni ukazi, s katerimi lahko v PC zapišemo poljuben naslov.

- **Izjema 2:** Prekinitev ali past
CPE po koraku 2 ne prevzame ukaza po pravilu $PC \leftarrow PC + 1$, temveč začne izvajati drug program - prekinitveno servisni program (PSP). Potrebna je tudi pravilna vrnitev v prvotni program.



Izvedba ukazov





Izvedba ukazov

- Zaporedno izvajanje ukazov je počasno in predstavlja osnovno slabost von Neumannovih računalnikov.
- Razširitve osnovnega von Neumannovega modela so zajete v Flynnovi klasifikaciji iz leta 1966.



Osnove delovanja računalnikov - vsebina:

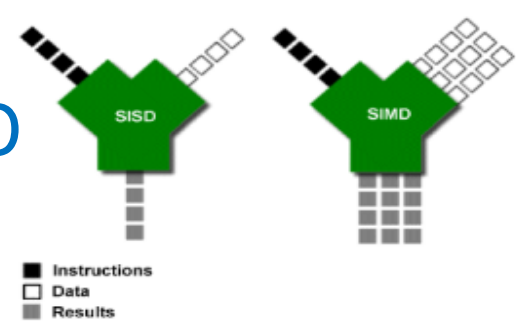
- 3.1 Von Neumannov računalniški model
- 3.3 Glavni pomnilnik v von Neumannovem računalniku
- 3.4 Amdahlov zakon
- 3.5 Jeziki, nivoji in navidezni računalniki
- 3.6 Primer izvedbe programa v računalniku



3.2 Flynnova klasifikacija

- To razvrstitev računalnikov v štiri skupine je predlagal M.J.Flynn leta 1966. Osnovna kriterija te klasifikacije za razvrstitev računalnikov sta:
 - število ukazov, ki se izvršujejo hkrati (instruction stream),
 - število operandov, ki jih en ukaz hkrati obdeluje (data stream).
- Po teh dveh kriterijih spada vsak računalnik v enega od štirih razredov:
- 1 SISD (Single Instruction Single Data)
 - klasični Von Neumannovi računalniki brez paralelizma pri ukazih in operandih
 - Intel Pentium 4

Flynnova klasifikacija: SIMD, MISD, MIMD



Source: ARS Technica

■ 2 SIMD (Single Instruction Multiple Data)

- Pravi vektorski računalniki (paralelni računalniki, grafični procesorji)
- Ukazi SSE (Streaming SIMD Extensions) pri procesorjih z arhitekturo x86, NEON enote pri ARM

■ 3 MISD ! (Multiple Instruction Single Data)

- Neobičajna arhitektura. Več ukazov nad istimi operandi – uporablja se tam, kjer se zahteva neobčutljivost na napake („redundanca“).

■ 4 MIMD (Multiple Instruction Multiple Data)

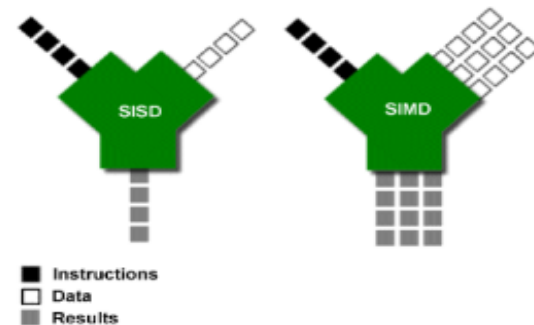
- Multiprocesorski računalniki (paralelni računalniki)
- V to skupino bil lahko (pogojno) uvrstili tudi večjedrne superskalarne računalnike, npr. Intel Core i7, čeprav jih zaradi manjšega števila jeder v primerjavi s superračunalniki, običajno še vedno uvrstimo v SISD.





Flynnova klasifikacija: SIMD, MISD, MIMD

- Pri MIMD računalnikih se naenkrat izvaja več ukazov, vsak na svojih operandih.
- MIMD računalnik tvori več povezanih navadnih von Neumannovih računalnikov – več CPE, ki so med seboj povezane.
- Večjedrne računalnike včasih štejemo tudi kar med SISD, čeprav jih lahko uvrščamo tudi v SIMD in MIMD.



Source: ARS Technica

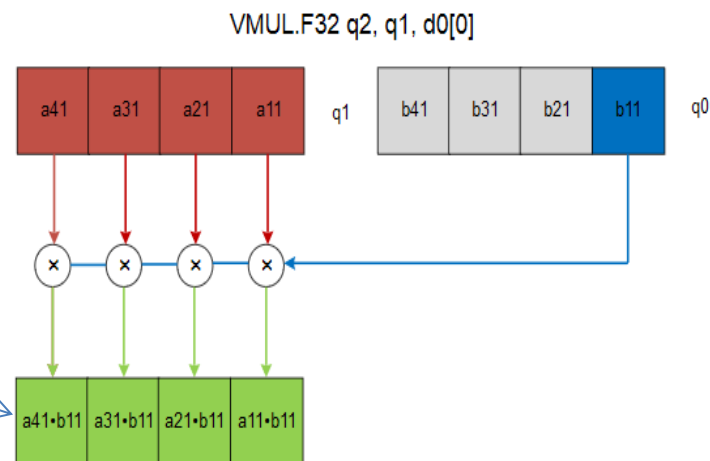
Primer: SIMD kot enota v CPE

Primer: matrično množenje: (ARM: NEON enota kot SIMD razširitev) :

Figure 4.4. Matrix multiplication showing one column of results

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} = \begin{bmatrix} a_{11} \cdot b_{11} + a_{12} \cdot b_{21} + a_{13} \cdot b_{31} + a_{14} \cdot b_{41} & \dots & \dots & \dots \\ a_{21} \cdot b_{11} + a_{22} \cdot b_{21} + a_{23} \cdot b_{31} + a_{24} \cdot b_{41} & \dots & \dots & \dots \\ a_{31} \cdot b_{11} + a_{32} \cdot b_{21} + a_{33} \cdot b_{31} + a_{34} \cdot b_{41} & \dots & \dots & \dots \\ a_{41} \cdot b_{11} + a_{42} \cdot b_{21} + a_{43} \cdot b_{31} + a_{44} \cdot b_{41} & \dots & \dots & \dots \end{bmatrix}$$

Figure 4.5. NEON vector-by-scalar multiplication



GPU : podobna filozofija, je širši koncept



2. Učinkovitejše programiranje - Primer

- Ker **poznavanje arhitekture in delovanja** računalnikov lahko vodi v **učinkovitejše programiranje** (programe).
 - Primer 2: optimizacija programa za hitrejšo delovanje ob upoštevanju vzporednosti delovanja – SIMD paralelnosti

us/Iteration	Iterations/sec
2.02500	493827.16
0.53300	1876172.61

Spodnja rešitev (**izkorišča SIMD ukaze**) je skoraj 4-krat hitrejša !

Vir: „Pomen poznavanja računalniške arhitekture“, avtor Miha Krajnc (e-učilnica).

```
double results[st];

for(int i = 0; i < st; ++i)
{
    results[i] = a[i] * b[i];
}
```

```
float results[st];

for(int i = 0; i < (st - 8); i += 8)
{
    __m256 i_a = _mm256_load_ps(&a[i]);
    __m256 i_b = _mm256_load_ps(&b[i]);
    __m256 i_c = _mm256_mul_ps(i_a, i_b);
    _mm256_store_ps(&results[i], i_c);
}

for(int i = (st - 8); i < st; ++i)
{
    results[i] = a[i] * b[i];
}
```



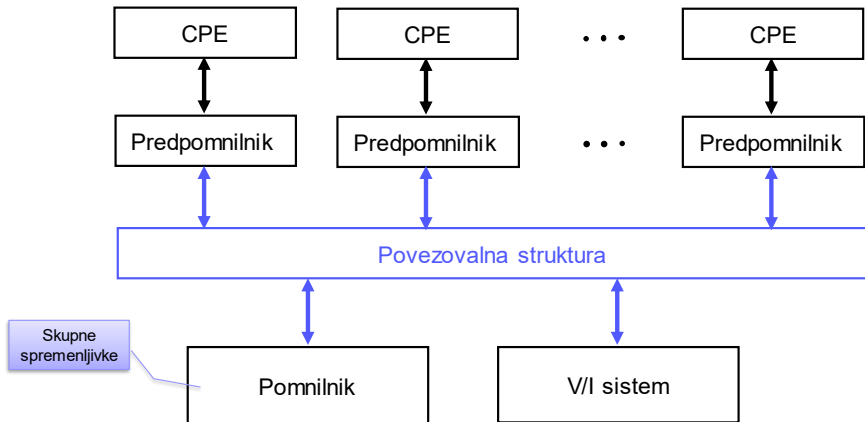
MIMD: Multiprocesor in multiračunalnik

Primeri:

- 4 MIMD (Multiple Instruction Multiple Data)

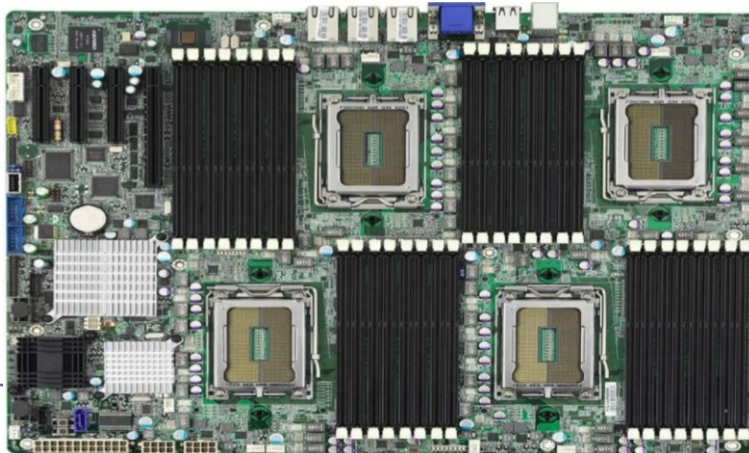
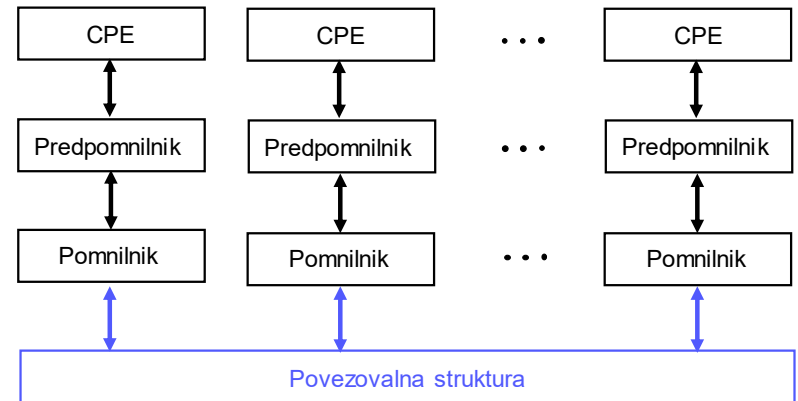
Multiprocesor

(tesneje povezani na plošči)



Multiračunalnik

(ohlapneje povezani moduli)





Osnove delovanja računalnikov - vsebina:

- 3.1 Von Neumannov računalniški model
- 3.2 Flynnova klasifikacija
- 3.3 Amdahlov zakon
- 3.4 Amdahlov zakon
- 3.5 Jeziki, nivoji in navidezni računalniki
- 3.6 Primer izvedbe programa v računalniku



3.3 Glavni pomnilnik v von Neumannovem računalniku

■ Definicija

- Glavni pomnilnik je pasivna naprava in služi za shranjevanje ukazov in operandov.
- Osnovna celica v pomnilniku je pomnilniška celica, ki lahko hrani 1 bit informacije (vsebina 0 ali 1).

■ Pomnilniška beseda (tudi pomnilniška lokacija)

- Pomnilniška beseda je definirana kot najmanjše število bitov, ki imajo svoj naslov. Pomnilniška beseda je torej najmanjša naslovljiva enota v pomnilniku.
- Pomnilnik je enodimenzionalno zaporedje pomnilniških besed.
- Pomnilniško besedo sestavlja določeno število enobitnih pomnilniških celic.
- **Dolžina pomnilniške besede** je število enobitnih pomnilniških celic, ki sestavljajo pomnilniško besedo. Danes je najpogostejša dolžina besede 1 bajt (= 8 bitov).



Pomnilniški naslov, naslovni prostor

■ Pomnilniški naslov

- Je enolična oznaka vsake pomnilniške besede
- Vsaka pomnilniška beseda ima svoj enolični pomnilniški naslov.
- Naslov pomnilniške besede je nespremenljiv.
- Število bitov, ki sestavljajo naslov, imenujemo **dolžina naslova**.
- Dolžina naslova v bitih določa naslovni prostor.

■ Naslovni prostor (tudi pomnilniški prostor)

- Je množica vseh naslovov
- In določa tudi največjo možno velikost pomnilnika.

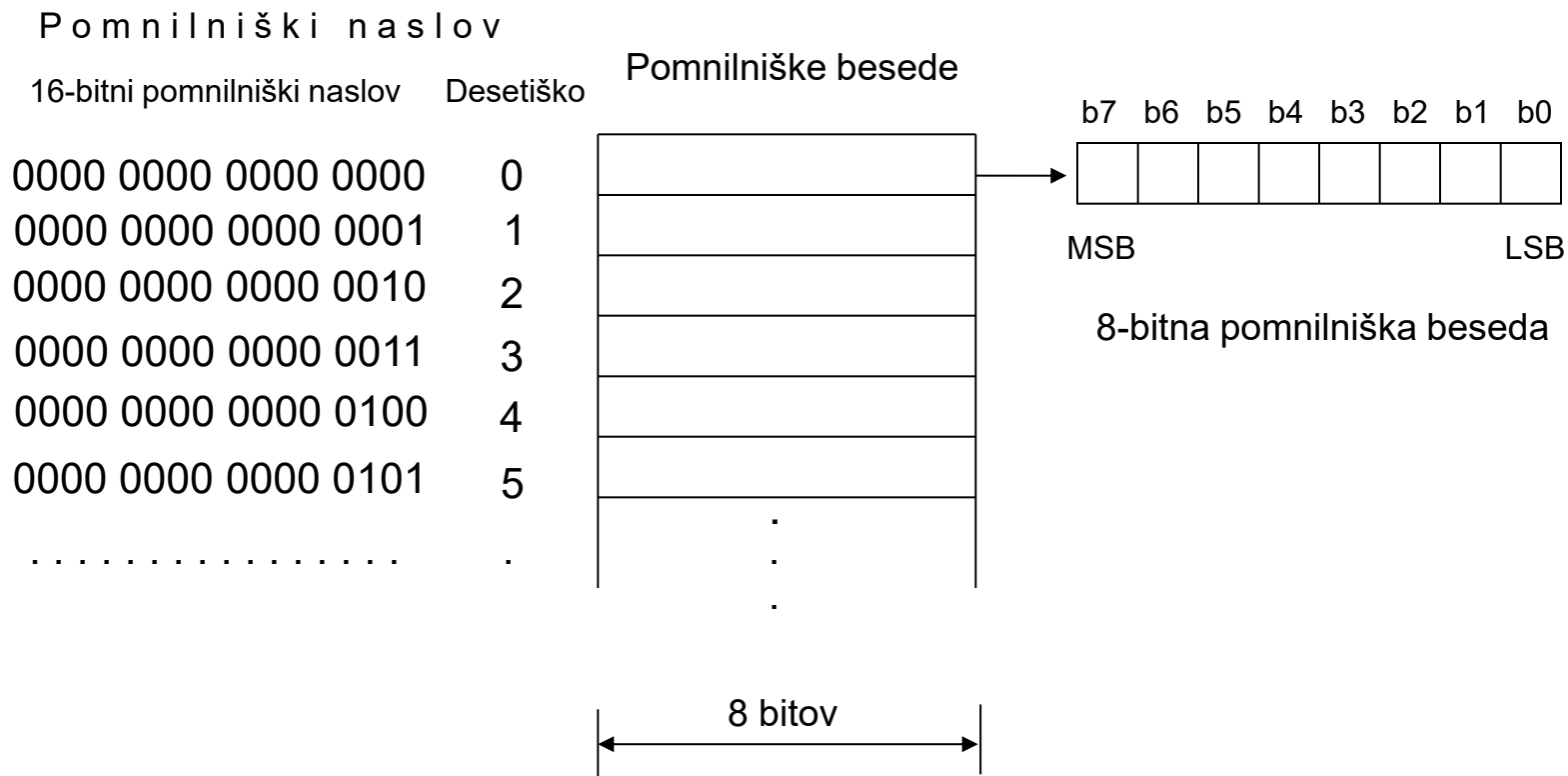


Pomnilniška beseda

- **Vsebina pomnilniške besede** se lahko spreminja. V 8-bitno pomnilniško besedo lahko shranimo $2^8 = 256$ različnih vsebin.
- Naslov pomnilniške besede pa je nespremenljiv.
- Število pomnilniških besed v glavnem pomnilniku ni nujno enako velikosti naslovnega prostora.
- Deli naslovnega prostora so lahko prazni (vsi naslovi niso uporabljeni) \Rightarrow glavni pomnilnik je običajno manjši od največje možne velikosti.



Skica zgradbe pomnilnika





Skica 16-bitnega naslovnega prostora

Pomnilniški naslov

Dvojiško (16-bitni naslov)	Šestnajstiško	Desetiško	Pomnilniške besede
0000 0000 0000 0000	0000	0	
0000 0000 0000 0001	0001	1	
0000 0000 0000 0010	0002	2	
0000 0000 0000 0011	0003	3	
0000 0000 0000 0100	0004	4	
0000 0000 0000 0101	0005	5	
.....
.....
.....
1111 1111 1111 1011	FFFB	65531	
1111 1111 1111 1100	FFFC	65532	
1111 1111 1111 1101	FFFD	65533	
1111 1111 1111 1110	FFFE	65534	
1111 1111 1111 1111	FFFF	65535	

64 K (= 2¹⁶) pomnilniških besed



Predpone kilo, mega, giga idr. so samo pri velikosti pomnilnika potence števila 2!

Pomnilniki

- 1Ki (kilo) = $2^{10} = 1024$ (1 KB = 1024 B)
- 1Mi (mega) = $2^{20} = 1\,048\,576$ (1 MB = 1 048 576 B)
- 1Gi (giga) = $2^{30} = 1\,073\,741\,824$ (1 GB = $1024 \cdot 1024 \cdot 1024 = 1\,073\,741\,824$ B)
 - Vzrok je tehnološki: npr. 10-bitni pomnilniški naslov omogoča $2^{10} = 1024$ različnih naslovov in ne 1000.
 - **Predlog IEC 1998: KiB = 2^{10} B, MiB = 2^{20} B, GiB = 2^{30} B**

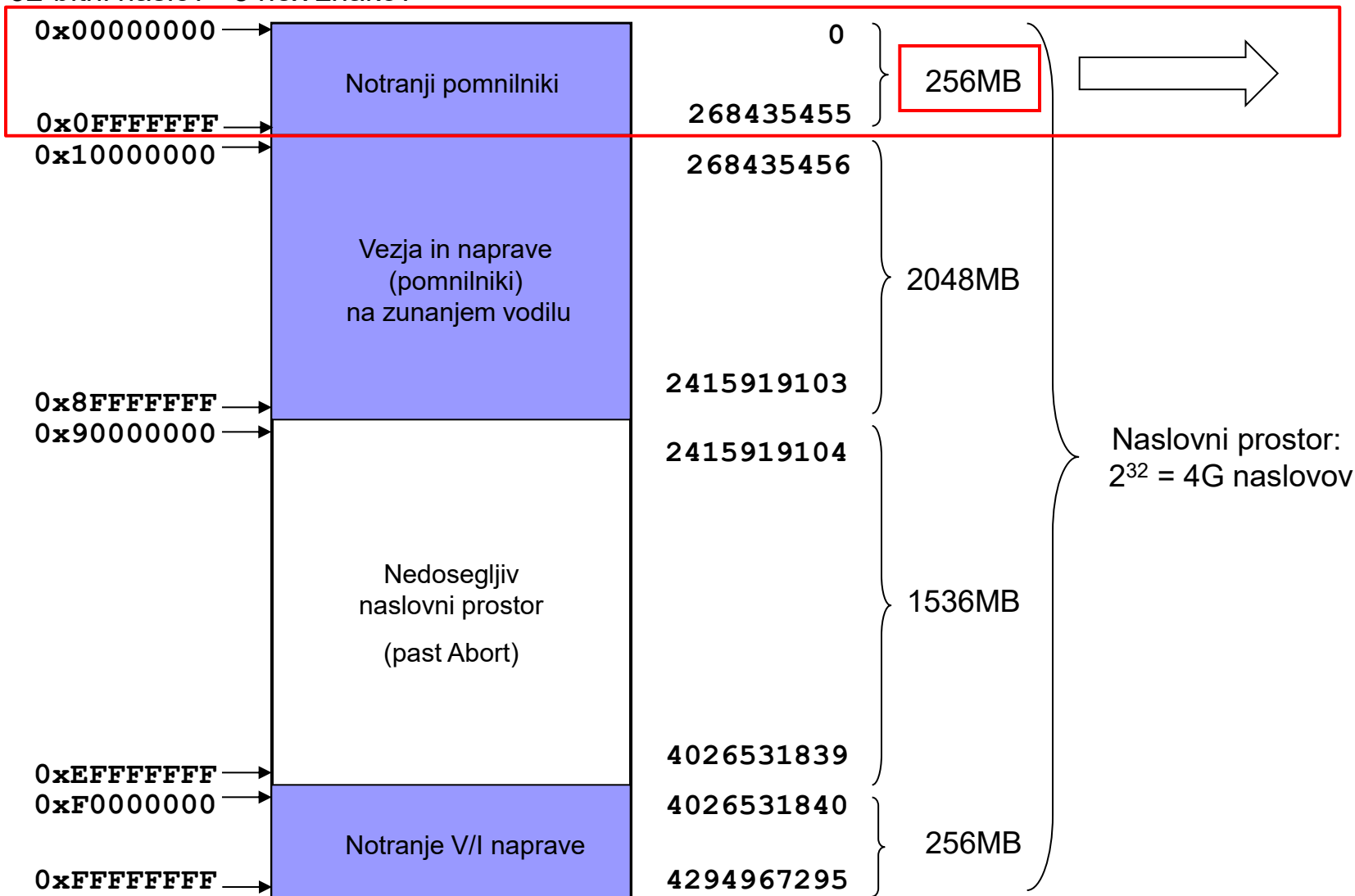
Druga področja (frekvenca, hitrost prenosa ...)

- 1k (kilo) = $10^3 = 1000$ (1 km = 1000 m)
- 1M (mega) = $10^6 = 1\,000\,000$ (100 Mb/s = 100 000 000 b/s)
- 1G (giga) = $10^9 = 1\,000\,000\,000$ (1 GHz = 1 000 000 000 Hz)



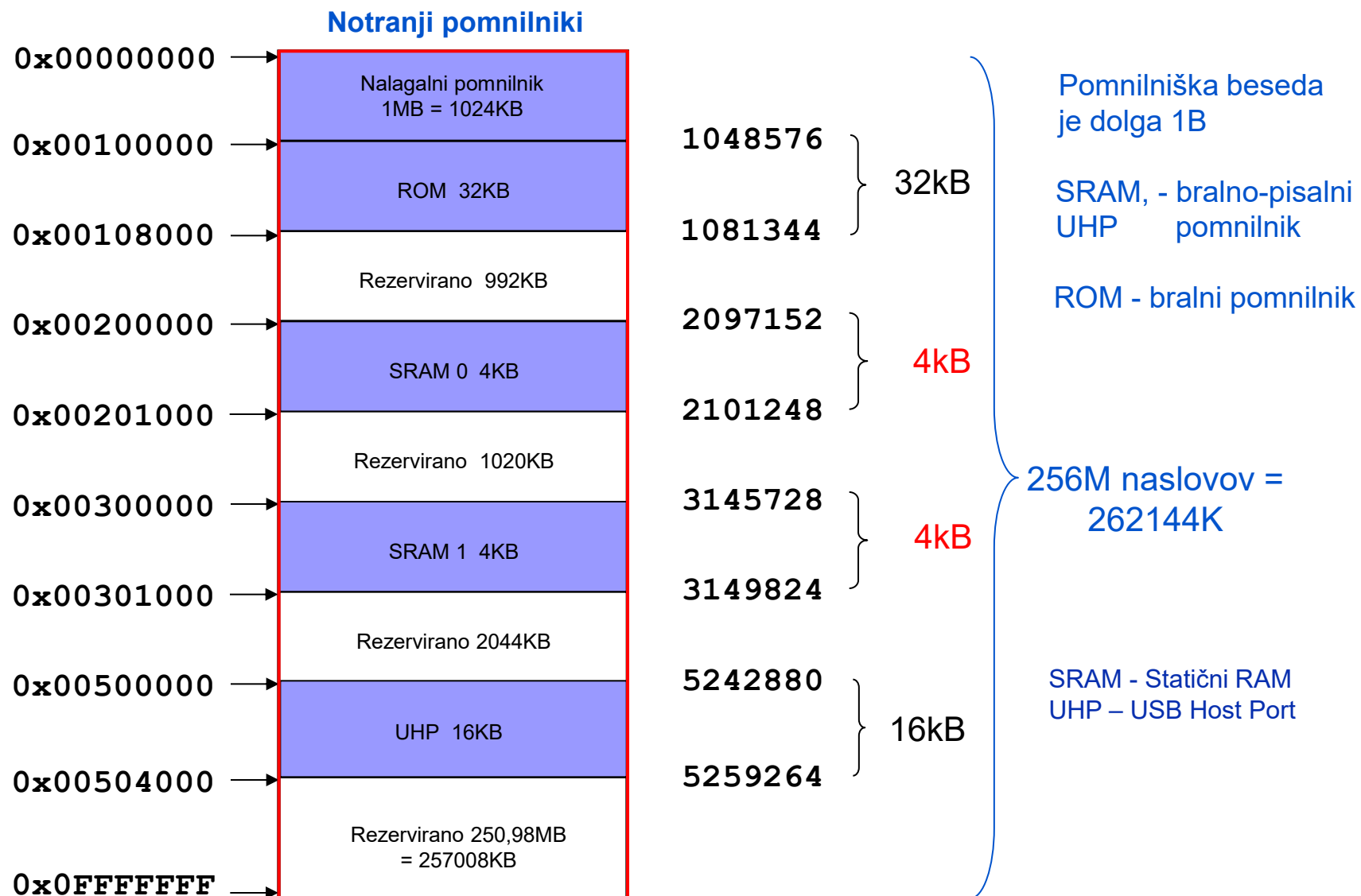
Primer slike naslovnega prostora pri procesorju AT91SAM9260 (32-bitni pomnilniški naslov)

32-bitni naslov - 8 hex znakov





Slika notranjega naslovnega prostora (prvih 256 MB) do pri AT91SAM9260

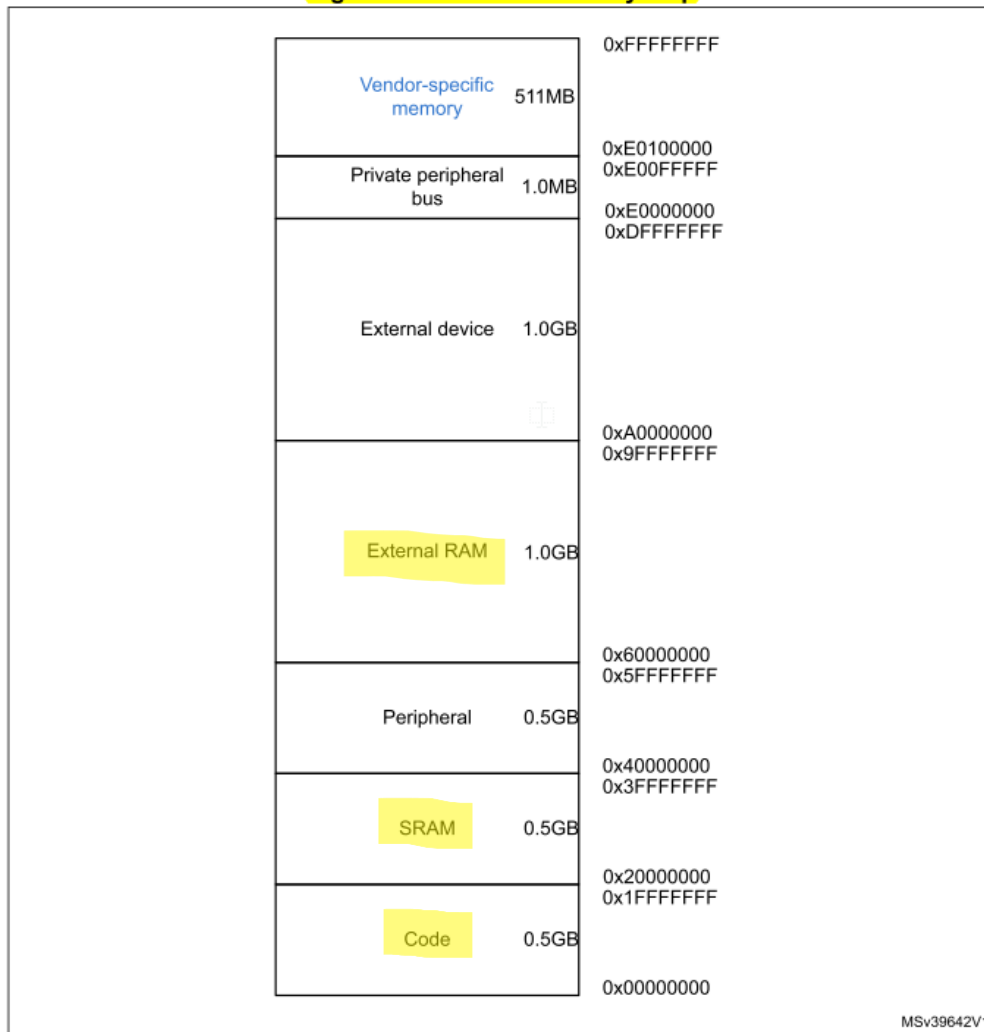




Primer slike naslovnega prostora pri mikrokrmilniku STM32H750XB

32-bitni naslov - 8 hex znakov

Figure 8. Processor memory map



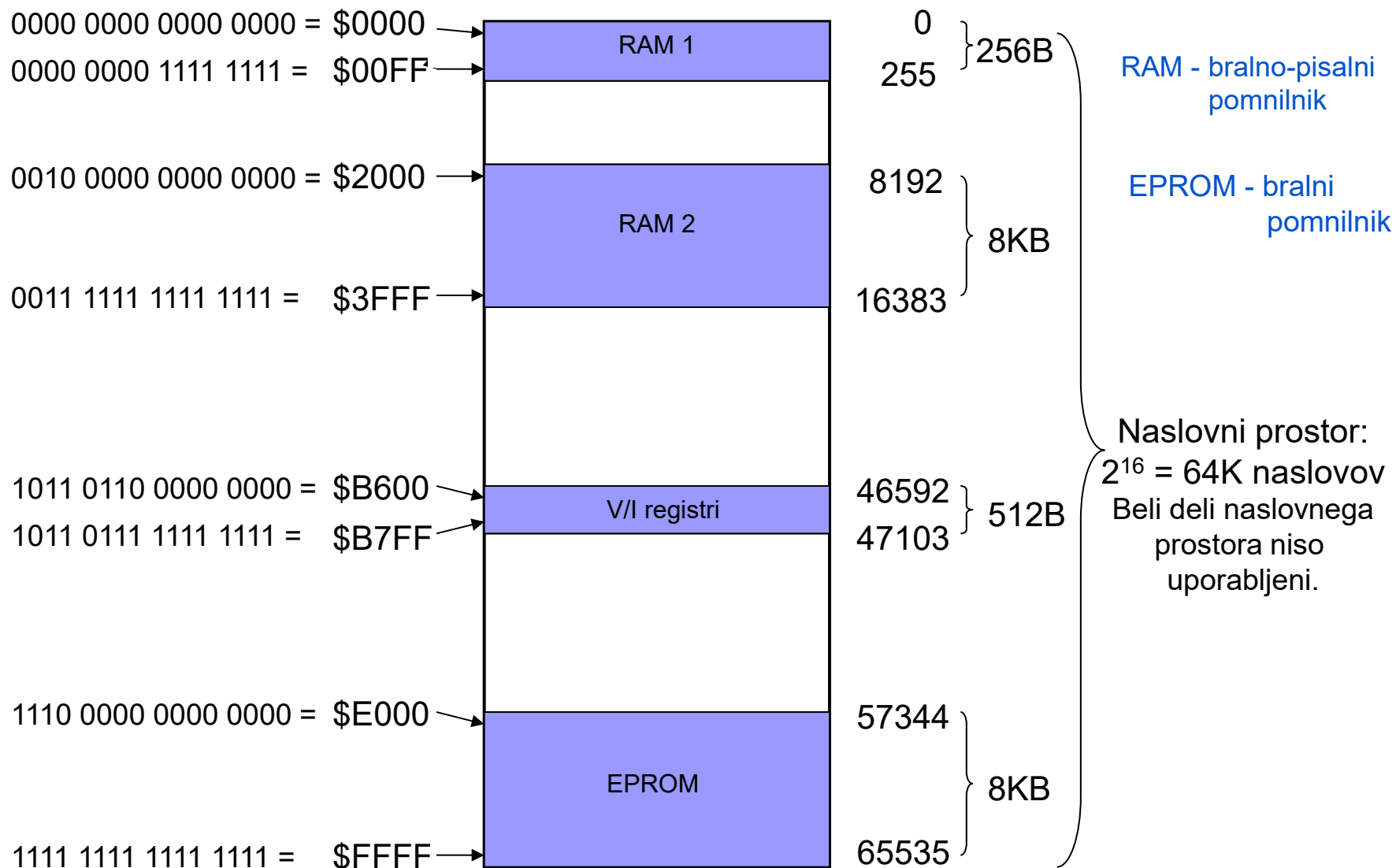
Naslovni prostor:
 $2^{32} = 4\text{G}$ naslovov

```
MEMORY
{
  FLASH (rx) : ORIGIN = 0x08000000, LENGTH = 128K
  DTCMRAM (xrw) : ORIGIN = 0x20000000, LENGTH = 128K
  RAM_D1 (xrw) : ORIGIN = 0x24000000, LENGTH = 512K
  RAM_D2 (xrw) : ORIGIN = 0x30000000, LENGTH = 288K
  RAM_D3 (xrw) : ORIGIN = 0x38000000, LENGTH = 64K
  ITCMRAM (xrw) : ORIGIN = 0x00000000, LENGTH = 64K
}
```




Primer slike pomnilnika (memory map) pri procesorju 68HC11 – procesor ima 16-bitni pomnilniški naslov

16-bitni pomnilniški naslov





Von Neumannovo ozko grlo

- Prenosi CPE \leftrightarrow gl. pomnilnik - promet
- Von Neumannovo ozko grlo - povezava med CPE in glavnim pomnilnikom. Iz pomnilnika v CPE se prenašajo vsi ukazi in operandi iz pomnilnika ali v pomnilnik.
- Eden od načinov za razširitev tega ozkega grla je razdelitev glavnega pomnilnika v dva dela.



Harvardska arhitektura

- Pomnilnik je pri harvardski arhitekturi razdeljen na dva ločena pomnilnika.
- V enem so shranjeni samo operandi – operandni pomnilnik, v drugem pa samo ukazi – ukazni pomnilnik.
- Ukazni in operandni pomnilnik lahko delujeta istočasno. Tako lahko dosežemo do dvakrat večjo hitrost.
- Harvardska arhitektura se danes uporablja pri predpomnilniku na najnižjem nivoju (operandni in ukazni predpomnilnik L1), glavni pomnilnik pa je pri večini računalnikov en sam (princetonska arhitektura).



Dostop do pomnilnika

- CPE dostopa do pomnilniške besede tako, da v pomnilnik pošlje naslov te besede in signal za smer prenosa.

- Smer prenosa - vrsta dostopa
 - CPE ← gl. pomnilnik - branje (bralni dostop)

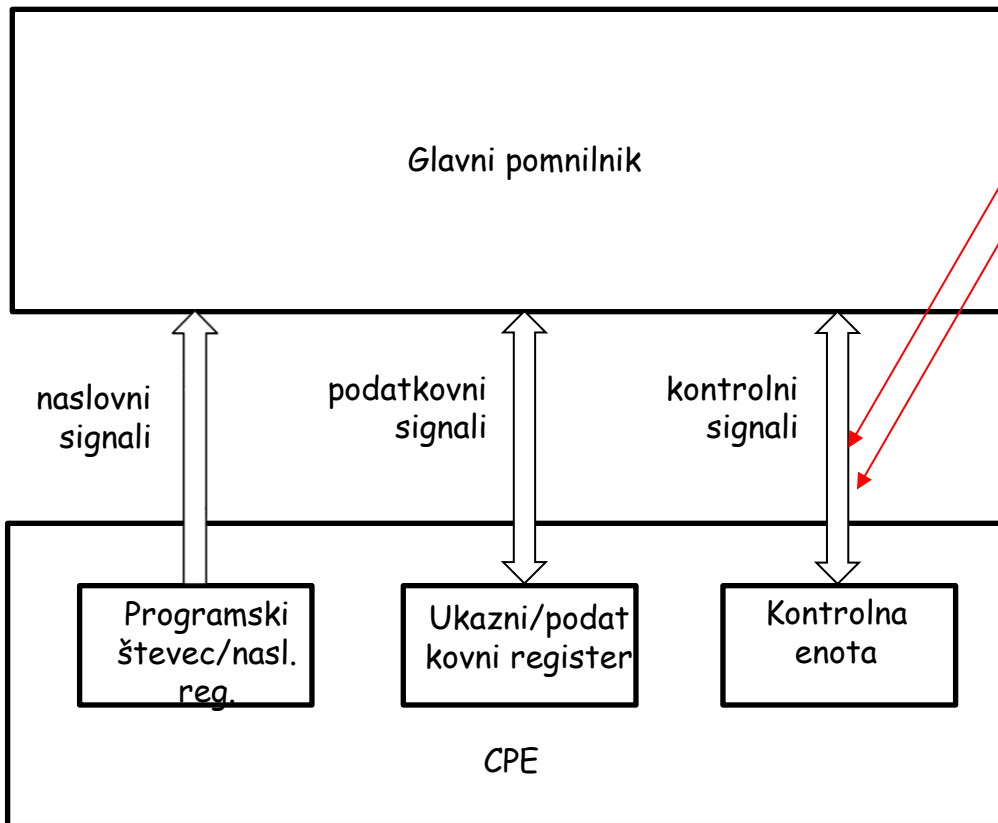
 - CPE → gl. pomnilnik - pisanje (pisalni dostop)



Povezava CPE <-> glavni pomnilnik?

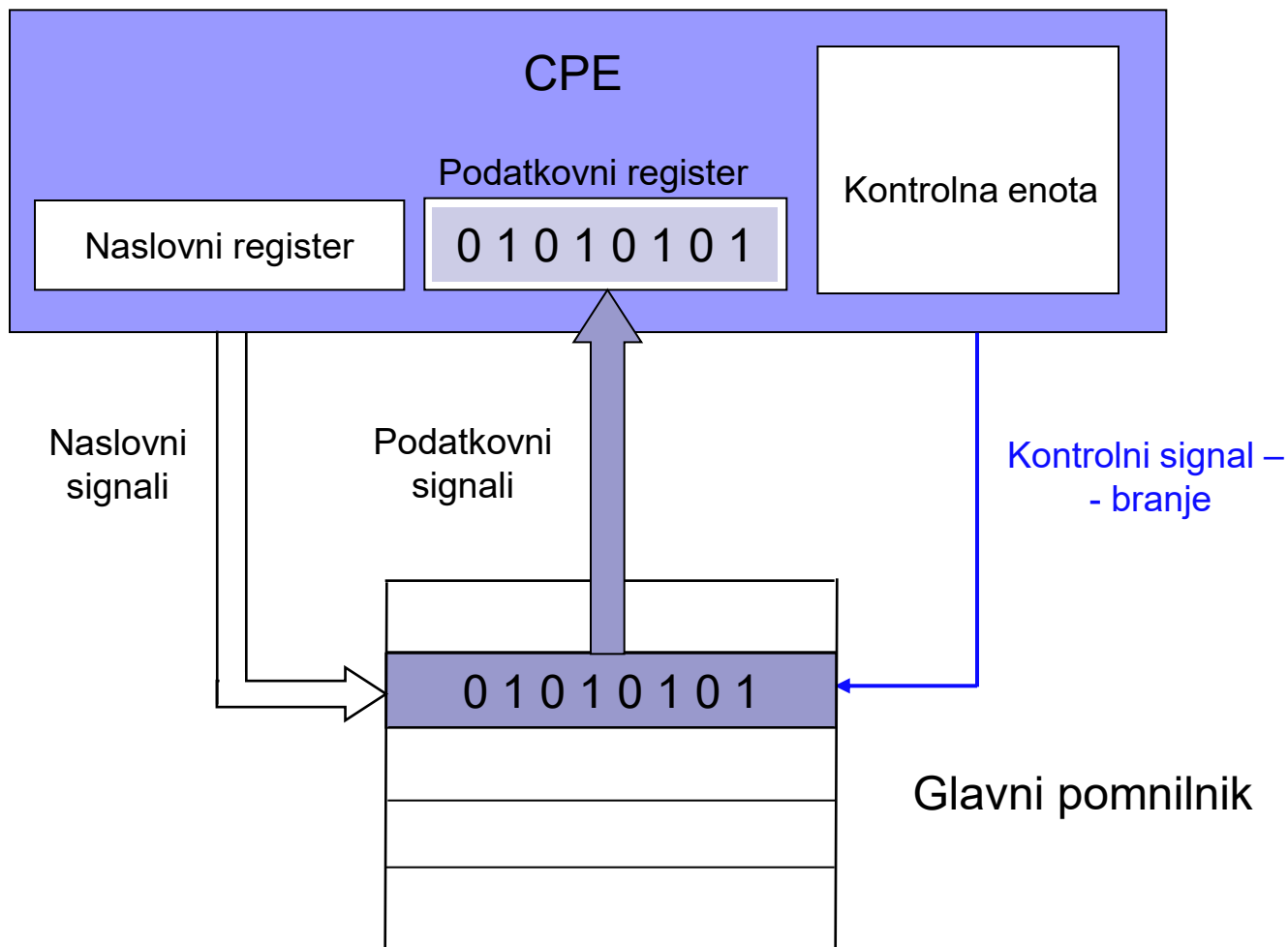
Vodilo = skupina povezav
(naslovno, podatkovno,
kontrolno, ...)

Linija = povezava
Signal = vsebina, ki se prenaša po povezavi (1bit)



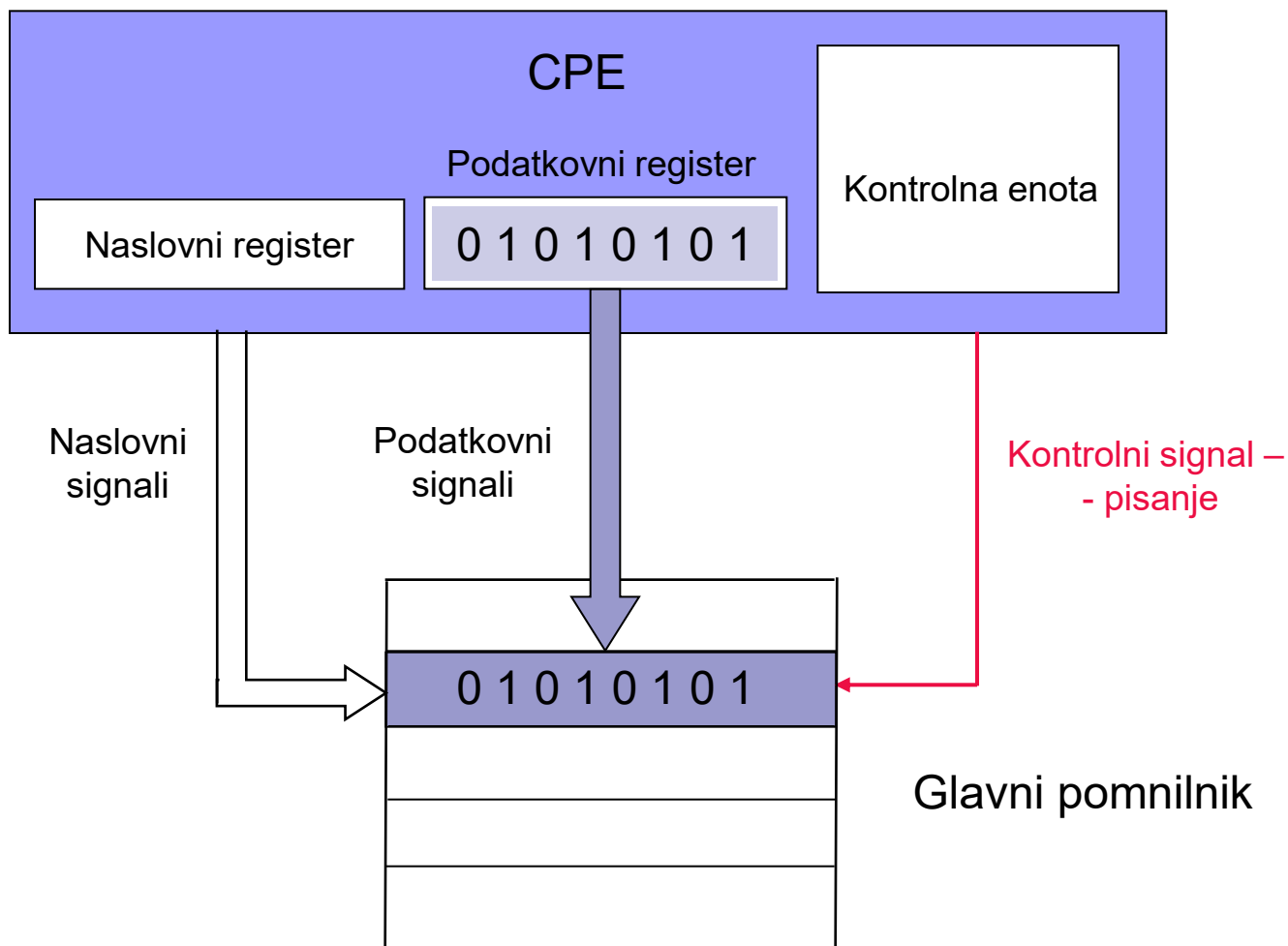


Povezava med CPE in glavnim pomnilnikom – bralni dostop

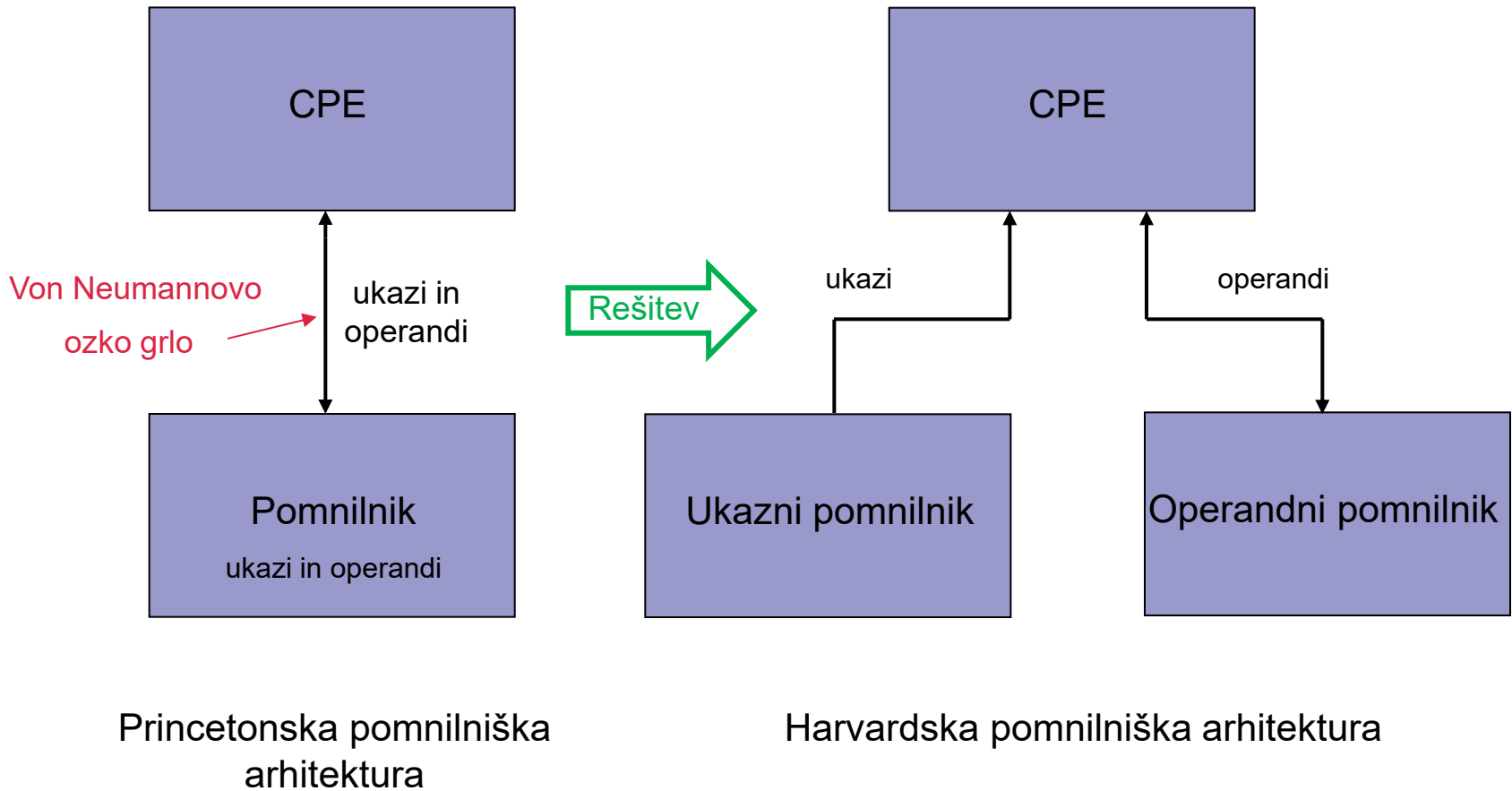




Povezava med CPE in glavnim pomnilnikom – pisalni dostop

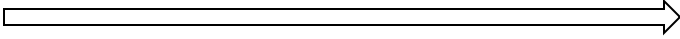


Razširitev von Neumannovega ozkega grla





Povzetek lastnosti glavnega pomnilnika v von Neumannovem računalniku

- Pomnilnik je **enodimenzionalen** in organiziran **v besede**. Vsaka beseda ima svoj, **enoličen naslov**.
- Ni razlike med **ukazi** in **operandi**.
- **Pomen** ni sestavni del operandov.
- **Veliko več bralnih kot pisalnih dostopov**,
 - Razmerje: okrog 80% branj (B), 20% pisanj (P)
 - **Zakaj?** 

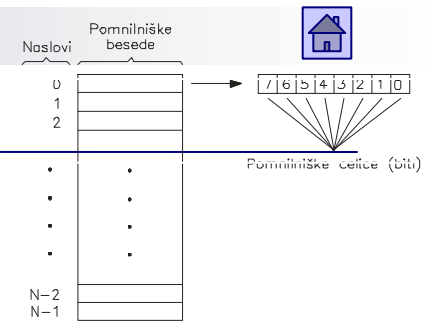
Primer

PROGRAM Zbirnik	
1B	adr r0,STEV1
2B	ldr r1,[r0]
1B	adr r0,STEV2
2B	ldr r2,[r0]
1B	add r3,r1,r2
1B	adr r0,REZ
1B1P	str r3,[r0]



Kombinacija 8 bitov v pomnilniku, npr. 1000 1011, lahko predstavlja:

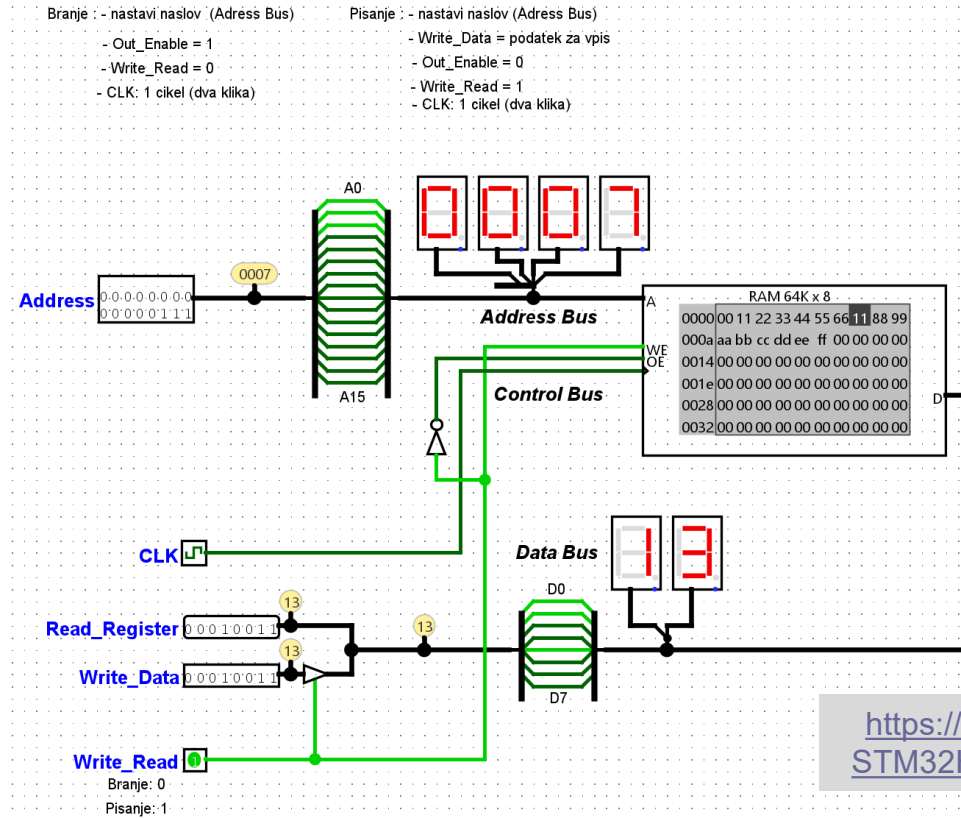
- število brez predznaka: 139 (desetiško)
ali
- število s predznakom: - 11 (desetiško)
ali
- znak v razširjeni ASCII abecedi: <
ali
- strojni ukaz: ADDA (op.koda strojnega ukaza procesorja 68HC11)
ali
- pomnilniški naslov: 139 (desetiško)
ali
- kombinacijo bitov
ali
- točka slike, vzorec zvoka, . . .



Pomnilnik

Demonstracija – Logisim EVO

Primer delovanja pomnilnika RAM



https://github.com/LAPSyLAB/RALab-STM32H7/tree/main/LogisimEVO_vezja

RAM_pomnilnik_demo_EVO.circ



Osnove delovanja računalnikov - vsebina:

- 3.1 Von Neumannov računalniški model
- 3.2 Flynnova klasifikacija
- 3.3 Glavni pomnilnik v von Neumannovem računalniku
- 3.5 Jeziki, nivoji in navidezni računalniki
- 3.6 Primer izvedbe programa v računalniku



3.4 Amdahlov zakon (1967)

- G. M. Amdahl je eden od arhitektov slavne serije računalnikov IBM 370.
- Če v računalniku za faktor N (N -krat) pohitrimo delovanje pri vseh operacijah, razen pri f -temu delu od vseh operacij, potem je povečanje hitrosti celotnega računalnika $S(N)$ enako:

$$S(N) = \frac{1}{f + \frac{1-f}{N}} = \frac{N}{1 + (N-1) * f}$$

f je delež operacij, ki niso pohitrene!

$S(N)$ = povečanje hitrosti celotnega sistema
 N = faktor povečanja hitrosti $(1 - f)$ -tega dela
 f = delež operacij, ki niso pohitrene
 $1 - f$ = delež operacij, ki so N -krat pohitrene



Amdahlov zakon – primer 1

prej

$f = 0.4$	$1-f = 0.6$
-----------	-------------

Primer 1:

potem

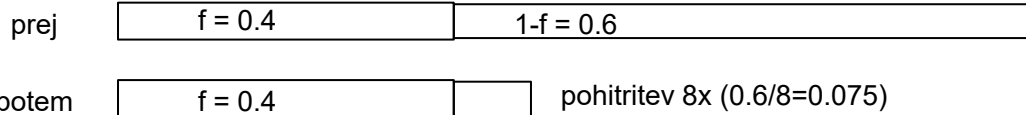
$f = 0.4$	\leftarrow
-----------	--------------

 pohitritev 8x ($0.6/8=0.075$)

- Izvajanje programov na nekem računalniku bi želeli pohitriti tako, da enojedrni procesor zamenjamo z osem-jedrnim (8 paralelno delujočih CPE).
- Kolikokrat hitreje se bodo izvajali programi, če se lahko paralelno izvaja samo 60 % programov?



Amdahlov zakon – primer 1



- $N = 8$ (del programov se lahko izvaja 8-krat hitreje)
- $1 - f = 0,6$ delež programov, ki so 8-krat pohitreni;
- $f = 0,4$ delež programov, ki niso pohitreni (40% programov se ne more izvajati paralelno)
- $S(N)$ pohitritev celote (vseh programov)

$$S(N) = \frac{8}{1 + (8 - 1) * 0.4} = \frac{8}{1 + 2.8} = 2.1$$

- Hitrost izvajanja vseh programov se poveča za faktor 2,1 (2,1 - krat).
- Če so se programi pred zamenjavo izvajali npr. 100 sekund, se bodo potem izvajali 47,6 sekunde ($100 / 2,1 = 47,6$).



Amdahlov zakon – primer 2

prej

$f = 0.1$	$1-f = 0.9$
-----------	-------------

Primer 2:

potem

$f = 0.1$	pohitritev 2x (polovični čas)
-----------	-------------------------------

- Izvajanje programa na nekem računalniku bi želeli pohitrili tako, da izvajanje 90% ukazov v programu dvakrat pohitrimo.
- Kolikokrat hitreje se bo izvajal program na takem računalniku ?

$$S(N) = ?$$



Amdahlov zakon – primer 2

prej

f = 0.1	1-f = 0.9
---------	-----------

Primer 2:

potem

f = 0.1	pohitritev 2x (polovični čas)
---------	-------------------------------

- Izvajanje programa na nekem računalniku bi želeli pohitriti tako, da izvajanje 90% ukazov dvakrat pohitrimo.
- Kolikokrat hitreje se bo izvajal program na takem računalniku ?

$$S(N) = \frac{1}{0.1 + \frac{0.9}{2}} = \frac{1}{0.1 + 0.45} = \frac{1}{0.55} = 1.818181$$

- Hitrost izvajanja programa se poveča za faktor 1.82 .



Amdahlov zakon – pomen in posledice

Amdahlov zakon:

- Paralelizacija ni idealna
- Pomembnost deleža operacij, ki se pohitrijo
- Pri večjem deležu je za enak končni učinek dovolj manjša pohitritev

Paralelizacija:

- Edina možnost vsled posebnosti razvoja elektronske tehnologije
- Ni tako enostavna glede doseganja pohitritve in programiranja
- Ima potencial večje učinkovitosti z vidika porabe energije



Osnove delovanja računalnikov - vsebina:

- 3.1 Von Neumannov računalniški model
- 3.2 Flynnova klasifikacija
- 3.3 Glavni pomnilnik v von Neumannovem računalniku
- 3.4 Amdahlov zakon
- 3.6 Primer izvedbe programa v računalniku



3.5 Jeziki, nivoji in navidezni računalniki

- Za veliko večino uporabnikov so podrobnosti o zgradbi in delovanju računalnikov nepomembne.
- Računalnik in njegove lastnosti vidijo predvsem skozi lastnosti programskega jezika, ki ga uporabljajo.
- Neki programski jezik se lahko realizira na zelo različnih računalnikih, to pa pomeni, da so različni računalniki za uporabnika, ki uporablja ta programski jezik, videti bolj ali manj enaki.



Računalnik kot zaporedje navideznih računalnikov

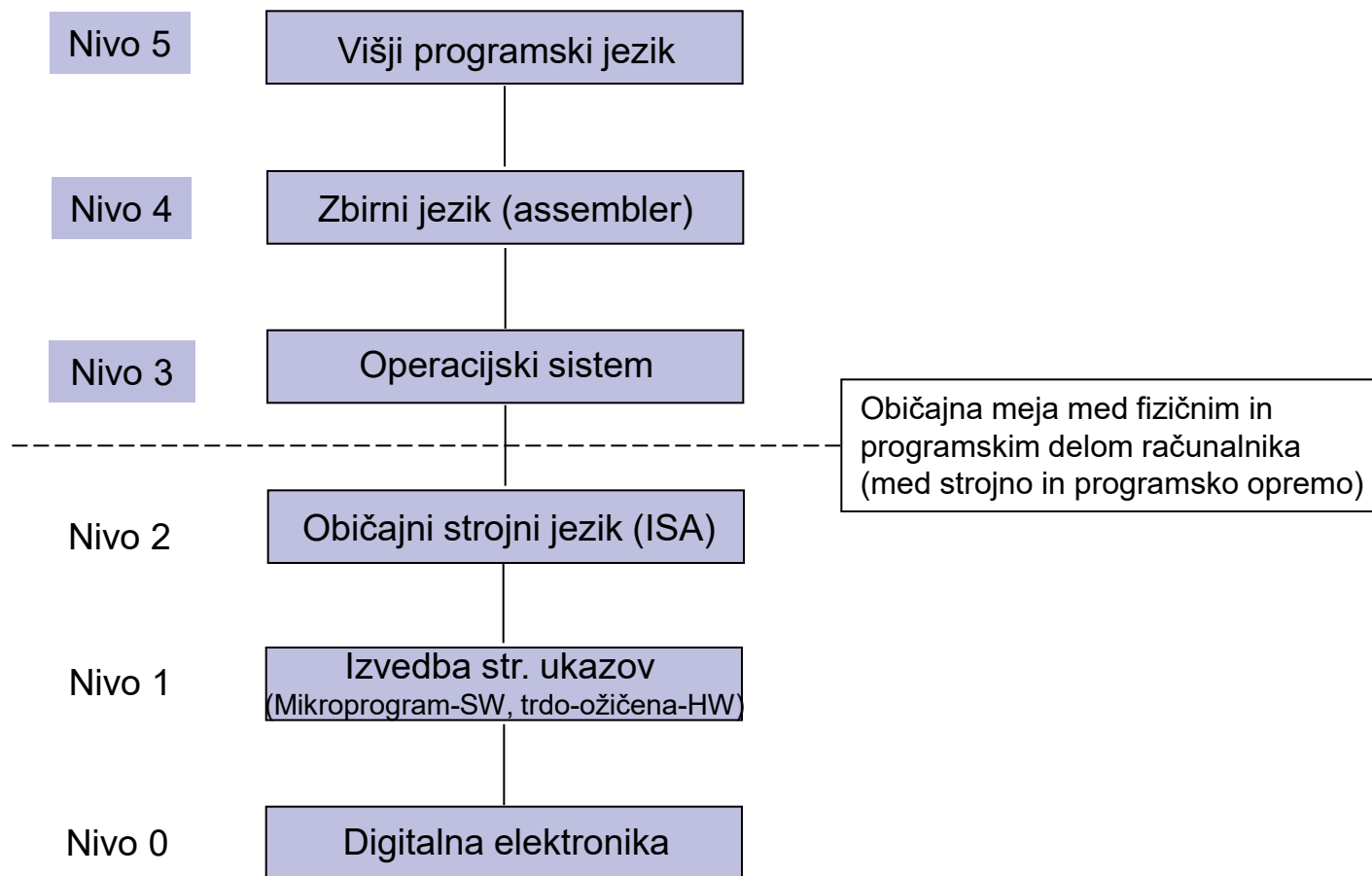
- Pri veliki večini današnjih računalnikov imamo 6 nivojev.
- Na vsakem nivoju vidimo računalnik skozi drugačen programski jezik.
- Ta programski jezik si lahko uporabnik predstavlja kot strojni jezik nekega navideznega računalnika.
- Na najnižjem nivoju (nivo 0) elektronika (logična vrata in flip-flopi) neposredno izvaja najenostavnejše ukaze.





Računalnik kot zaporedje navideznih računalnikov

Računalnik s šestimi nivoji – splošna definicija





Nivo 1

- Nivo 1 lahko zasledimo pri mnogih današnjih računalnikih. RISC računalniki nimajo nivoja 1.
 - Vsak ukaz običajnega strojnega jezika se izvrši kot zaporedje mikroukazov – računalnikom, ki tako delujejo (imajo nivo 1), rečemo, da so mikroprogramirani.
 - Pri teh računalnikih je mikroprogramski jezik dejansko pravi strojni jezik.
 - Ker v začetku računalniki tega nivoja niso imeli in je za uporabnika neviden, se pojem strojni jezik uporablja na nivoju 2.
 - Mikroprogram na nivoju 1 je napisan pri proizvajalcu in pravzaprav definira običajni strojni jezik. Uporabnik ga običajno ne more spreminjati.



Nivo 2

- Uporabnik vidi računalnik na nivoju 2 skozi uporabo običajnih strojnih ukazov, ki tvorijo običajni strojni jezik.
 - Računalniška arhitektura je določena z zgradbo in lastnostmi računalnika, kot jih vidi programer na tem nivoju.
 - Zato tudi ime ISA – Instruction Set Architecture.
 - Z običajnim strojnim jezikom ima programer popoln nadzor nad vsemi deli računalnika.
 - Pri prvih računalnikih višjih nivojev sploh ni bilo in je programiranje potekalo samo v običajnem strojnem jeziku.



Nivo 3

- Nivo 3 je nivo operacijskega sistema.
 - Jezik na tem nivoju vsebuje vse ukaze nivoja 2, ki so jim dodani novi ukazi za lažje delo z računalnikom (npr. delo z V/I napravami, paralelno izvajanje programov, diagnostični ukazi).
 - Operacijski sistem je program, ki olajša delo z računalnikom in služi kot vmesnik med uporabnikom in strojno opremo računalnika.
 - Z operacijskim sistemom želimo doseči:
 - lažje delo,
 - boljši izkoristek strojnih zmogljivosti računalnika (v določenem času opraviti kar največ dela).



Nivo 3

- Funkcije operacijskega sistema bi bilo mogoče realizirati tudi strojno na nivoju 2, vendar je programska izvedba bolj ekonomična (več operacijskih sistemov, nadgradnja . . .).
- Na tem nivoju je običajna tudi delitev uporabnikov z različno pravico uporabe ukazov.
- Nekateri ukazi nivoja 2 so običajnim uporabnikom na nivoju 3 nedostopni (dostopni samo sistemskim programerjem).
- Za večino današnjih programerjev je nivo 3 najnižji nivo, na katerem lahko delajo.



Nivoja 4, 5

- Na nivoju 4 uporabnik vidi računalnik skozi zbirni jezik.
 - Zbirni jezik je samo simbolična, človeku bližja oblika jezika nivoja 3 (in s tem tudi nivoja 2).
 - Programe v zbirnem jeziku je treba pred izvajanjem prevesti na jezik nivoja 3 (oziroma 2).

- Nivo 5 oblikujejo višji programski jeziki, ki so namenjeni večini programerjev.
 - To so npr. C, C#, C++, Java, Python, BASIC, FORTRAN, COBOL in mnogi drugi.
 - Programe, napisane v teh jezikih, je treba prevesti na jezik nivoja 4 ali nivoja 3.



Nivoji

- V računalnikih lahko ugotovimo tudi višje nivoje, kot npr. program za delo s podatkovnimi bazami, UI,
- Vsak nivo si lahko predstavljamo kot navidezni računalnik, ki ima za strojni jezik kar jezik tega nivoja, tako da običajnemu uporabniku na višjih nivojih ni potrebno poznavanje dejanskega strojnega nivoja.
- Vsekakor pa je potrebno programe, napisane v jeziku kateregakoli višjega nivoja (navideznega računalnika), pretvoriti v zaporedje ukazov strojnega jezika.
- Uporabniki se tega pretvarjanja pogosto ne zavedajo, proizvajalci računalnikov in programske opreme pa morajo poskrbeti za prehajanje iz enega jezika v drugega.



Prehajanje med nivoji

- Mehanizem prehajanja iz enega jezika v drugega je lahko realiziran na dva načina:
 - s prevajanjem,
 - z interpretiranjem.

- Po letu 1990 pa se je razširila še vmesna rešitev:
 - delno prevajanje (npr. byte code)

- Glavna razlika med prevajanjem in interpretiranjem je, da pri interpretiranju ne obstaja prevedeni (ciljni) program.

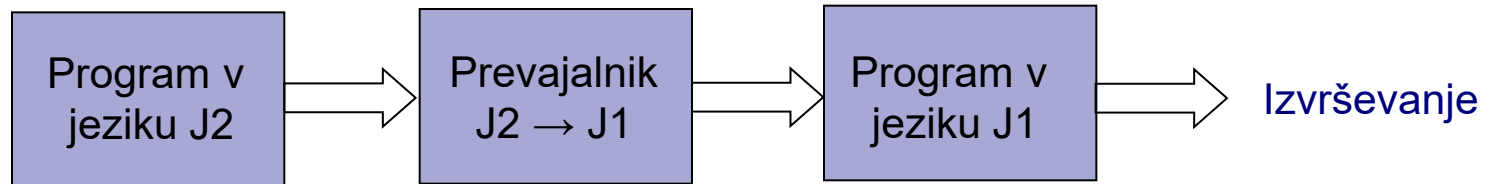


Prehajanje iz jezika J2 v jezik J1

Prevajanje

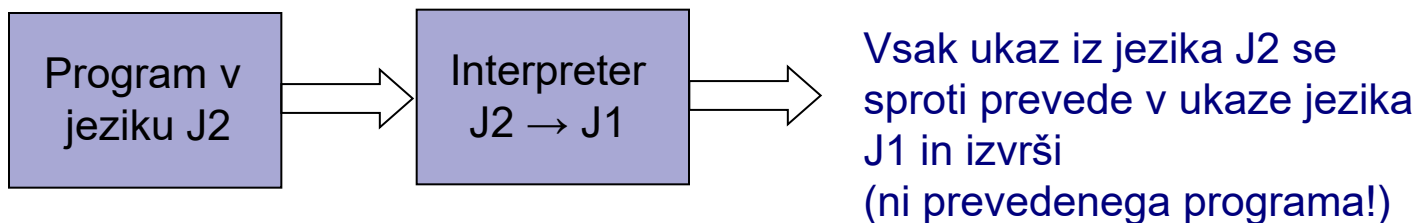
Izvorni program

Prevedeni program



Interpretiranje

Izvorni program





Prehajanje iz jezika J2 v jezik J1

- Prevedeni programi delujejo samo na računalniku s strojnim jezikom, v katerega so bili prevedeni.
- Pred prenosom na drugačen računalnik (z drugačnim strojnim jezikom J1) je treba izvorni program znova prevesti.

- Z vključevanjem velikega števila različnih računalnikov v omrežja je postala prenosljivost programov, ki jo omogoča interpretiranje, zelo pomembna.

- Delno prevajanje je neka vmesna rešitev med interpretiranjem in prevajanjem, ki omogoča hitrejša interpretiranja.



Prehajanje iz jezika J2 v jezik J1

- Delno prevajanje: Izvorni program v jeziku J2 se prevede v program v vmesnem jeziku J1, program v J1 pa se interpretira.
- Delno prevajanje v vmesni jezik J1 omogoča hitrejšo interpretiranje, ki pa je vseeno tipično 10-krat počasnejše kot izvajanje v celoti prevedenega programa.
- Tako je omogočena prenosljivost programov pri bistveno manjši izgubi hitrosti, kot če bi uporabili samo interpretiranje.



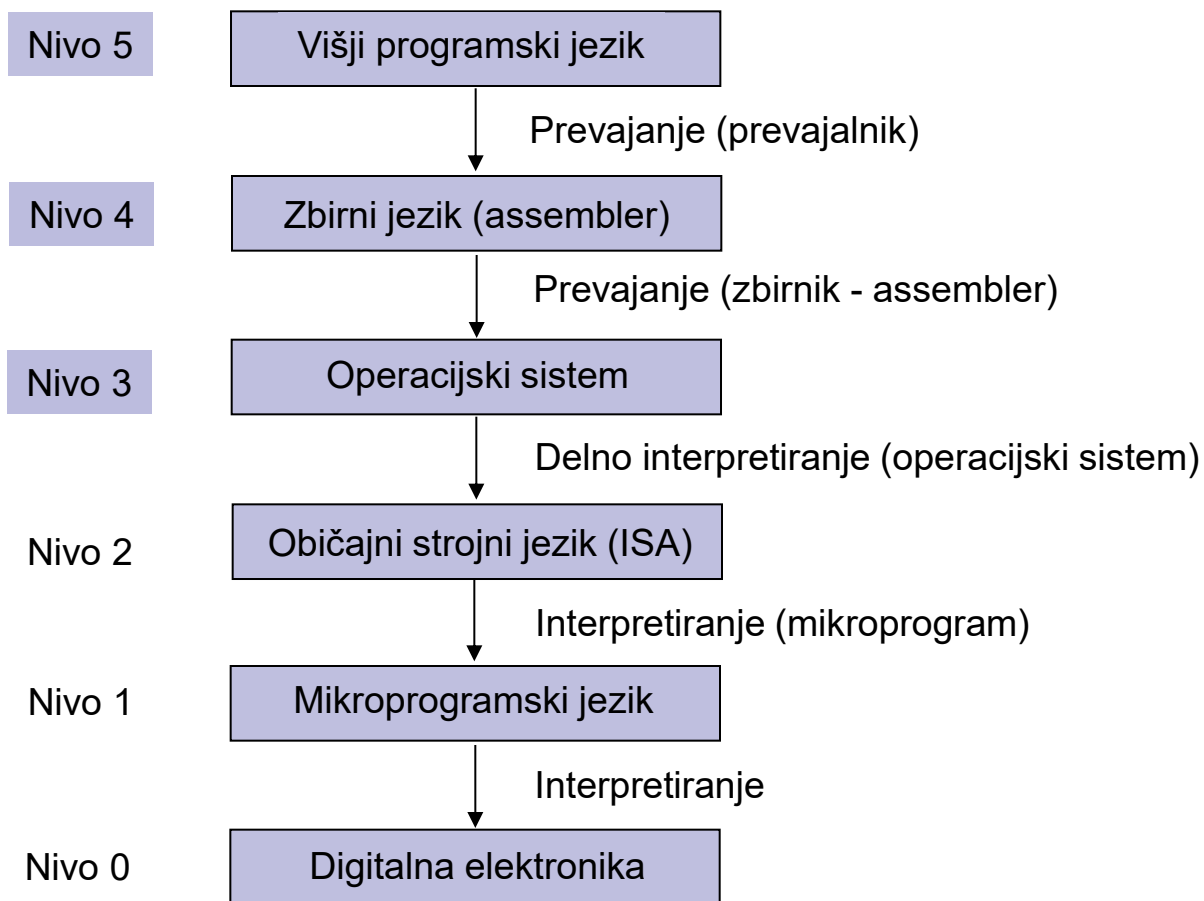
Primer navideznega računalnika

- JVM (Java Virtual Machine)
 - Virtual Machine – navidezni stroj (navidezni računalnik) je programska izvedba stroja (računalnika), ki deluje (izvaja programe) enako kot realen stroj (računalnik).
 - Javanski programi se izvajajo tako, da se najprej prevedejo (delno prevajanje) v neki vmesni jezik (Java byte code), ki se interpretira s programom JVM.



Računalnik s šestimi nivoji (mikroprogramiran)

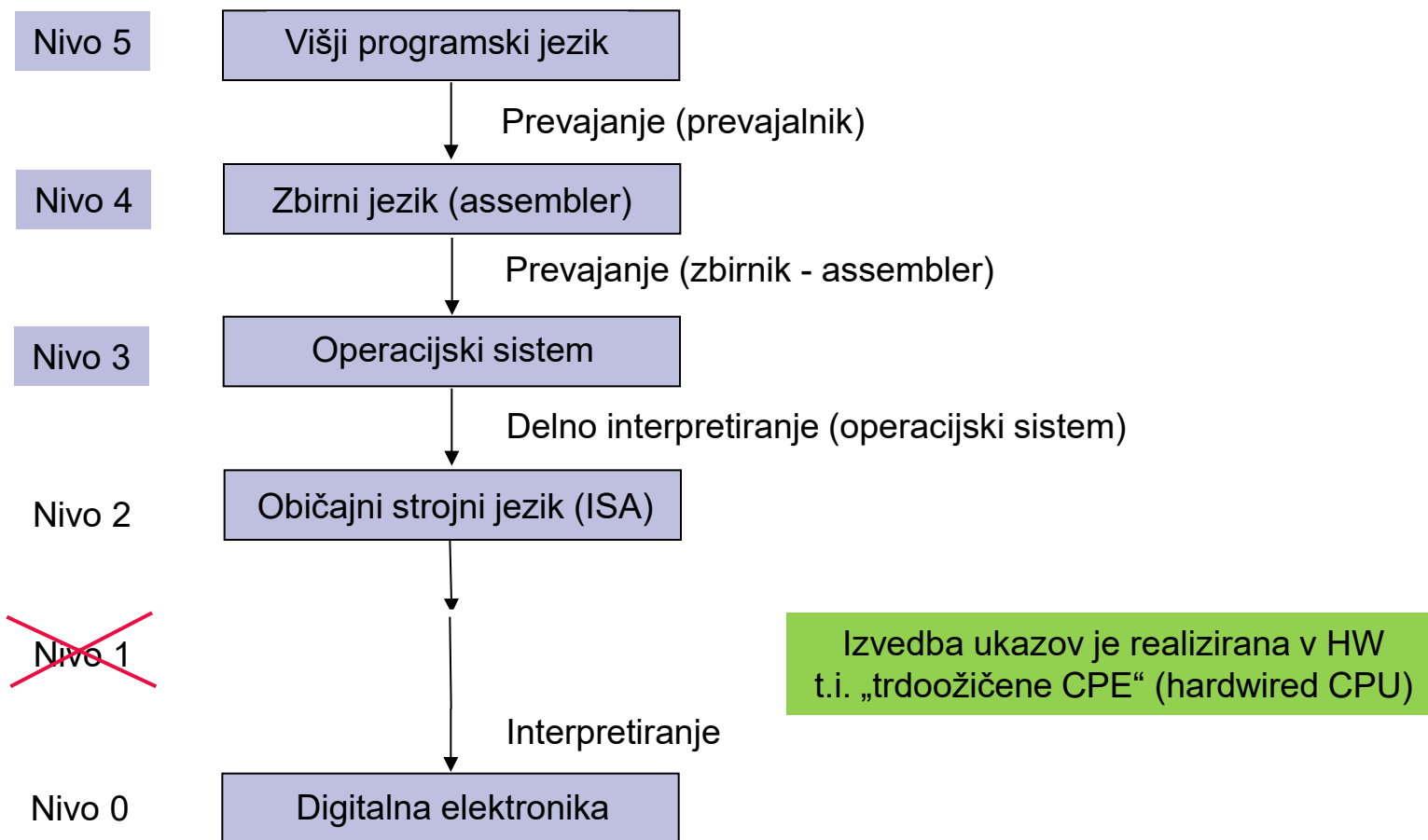
Nekoliko starejši računalniki





Računalnik s petimi nivoji

Novejši računalniki





Strojna in programska oprema računalnika

- Meja med strojnim in programskim delom računalnika ni trdna - lahko jo premikamo.
- Vsakega od nivojev lahko realiziramo tako strojno kot tudi programsko.
- Nivo 2 je npr. lahko realiziran s programom, ki teče na drugem računalniku.

Strojna in programska oprema sta logično ekvivalentni.



Strojna in programska oprema računalnika

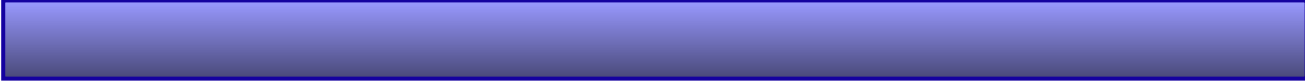
- Vsaka operacija, ki jo izvede programska oprema, se lahko realizira tudi direktno strojno (hardversko).

- Prav tako pa vsak strojni ukaz, ki ga izvaja hardver, lahko simuliramo s programom.

- Razvoj večnivojskih strojev
 - Iznajdba mikroprogramiranja (1951)
 - Iznajdba operacijskega sistema (okrog 1960)
 - Selitev funkcionalnosti v mikroprogram (okrog 1970)
 - Opuščanje mikroprogramiranja (po 1984)
 - Danes običajno kombinacija:
 - kompleksni ukazi običajnega strojnega nivoja so realizirani mikroprogramsko, enostavnejši ukazi so realizirani strojno.



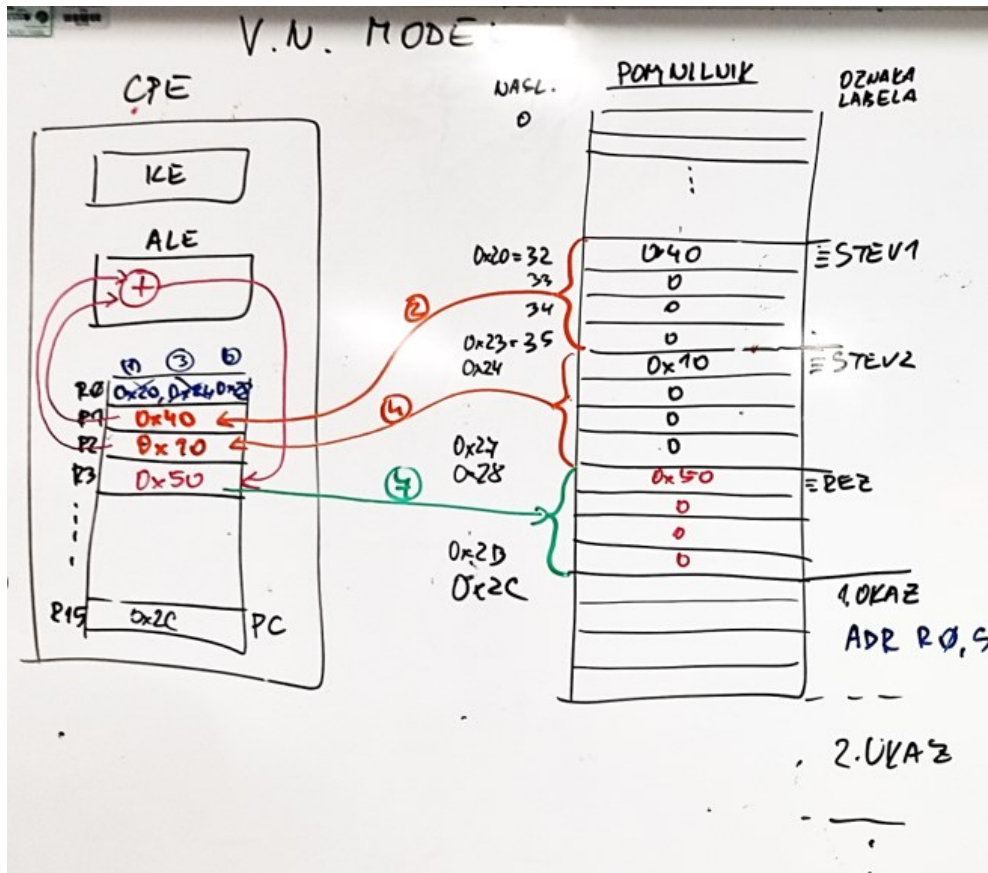
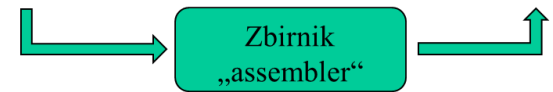
Osnove delovanja računalnikov - vsebina:

- 3.1 Von Neumannov računalniški model
 - 3.2 Flynnova klasifikacija
 - 3.3 Glavni pomnilnik v von Neumannovem računalniku
 - 3.4 Amdahlov zakon
 - 3.5 Jeziki, nivoji in navidezni računalniki
- 
- A thick horizontal bar with a blue-to-purple gradient, positioned below the list of topics.

Uvodna vaja: Programiranje v zbirniku

Zgled seštevanja dveh števil :
rez := stev1 + stev2

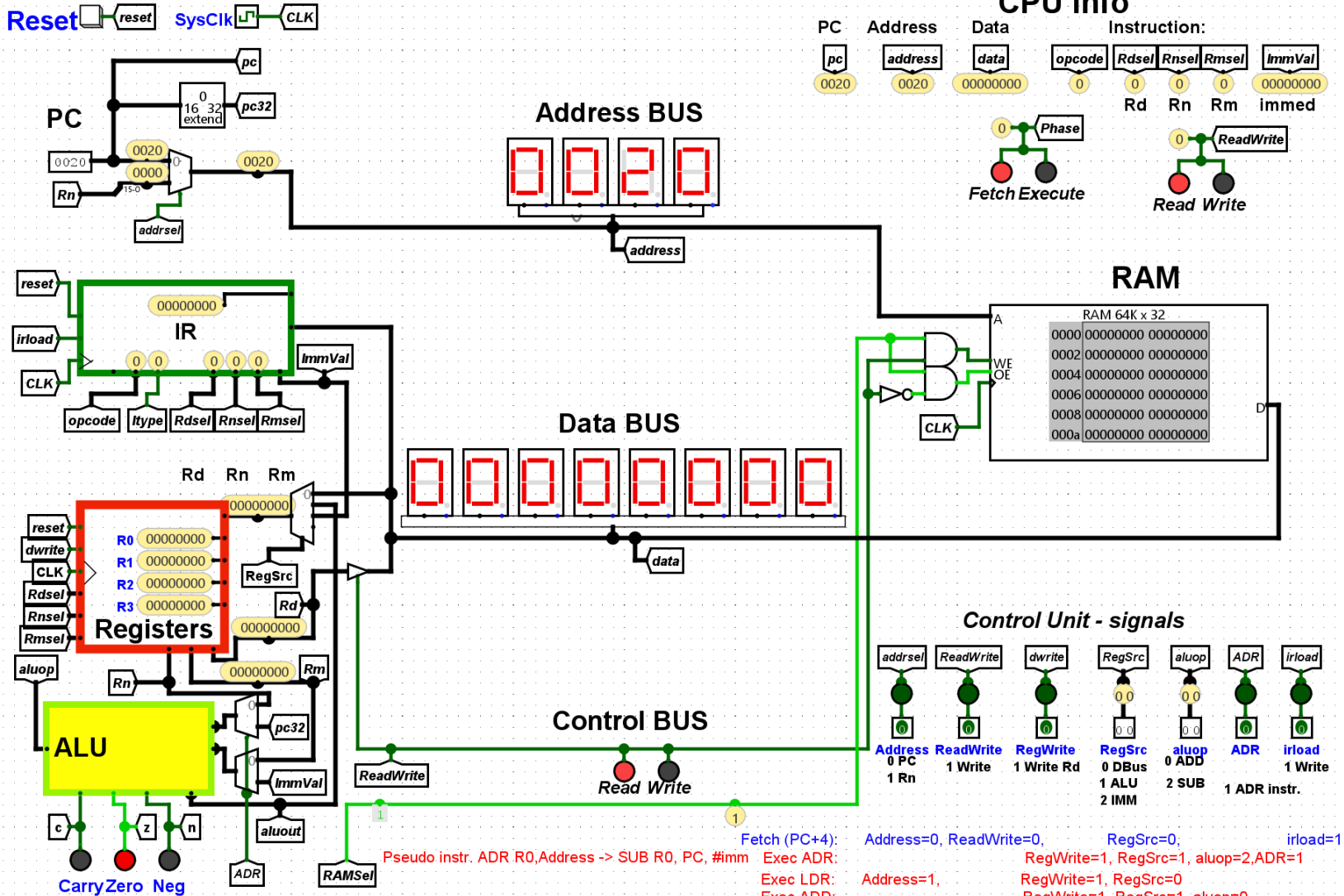
Zbirni jezik	Opis ukaza	Strojni jezik
adr r0, stev1	R0 ← nasl. stev1	0xE24F0014
ldr r1, [r0]	R1 ← M[R0]	0xE5901000
adr r0, stev2	R0 ← nasl. stev2	0xE24F0018
ldr r2, [r0]	R2 ← M[R0]	0xE5902000
add r3, r2, r1	R3 ← R1 + R2	0xE0823001
adr r0, rez	R0 ← nasl. rez	0xE24F0020
str r3, [r0]	M[R0] ← R3	0xE5803000



Mikro MiMo model CPE

Mikro MiMo - Simple CPU Model

v0 EVO

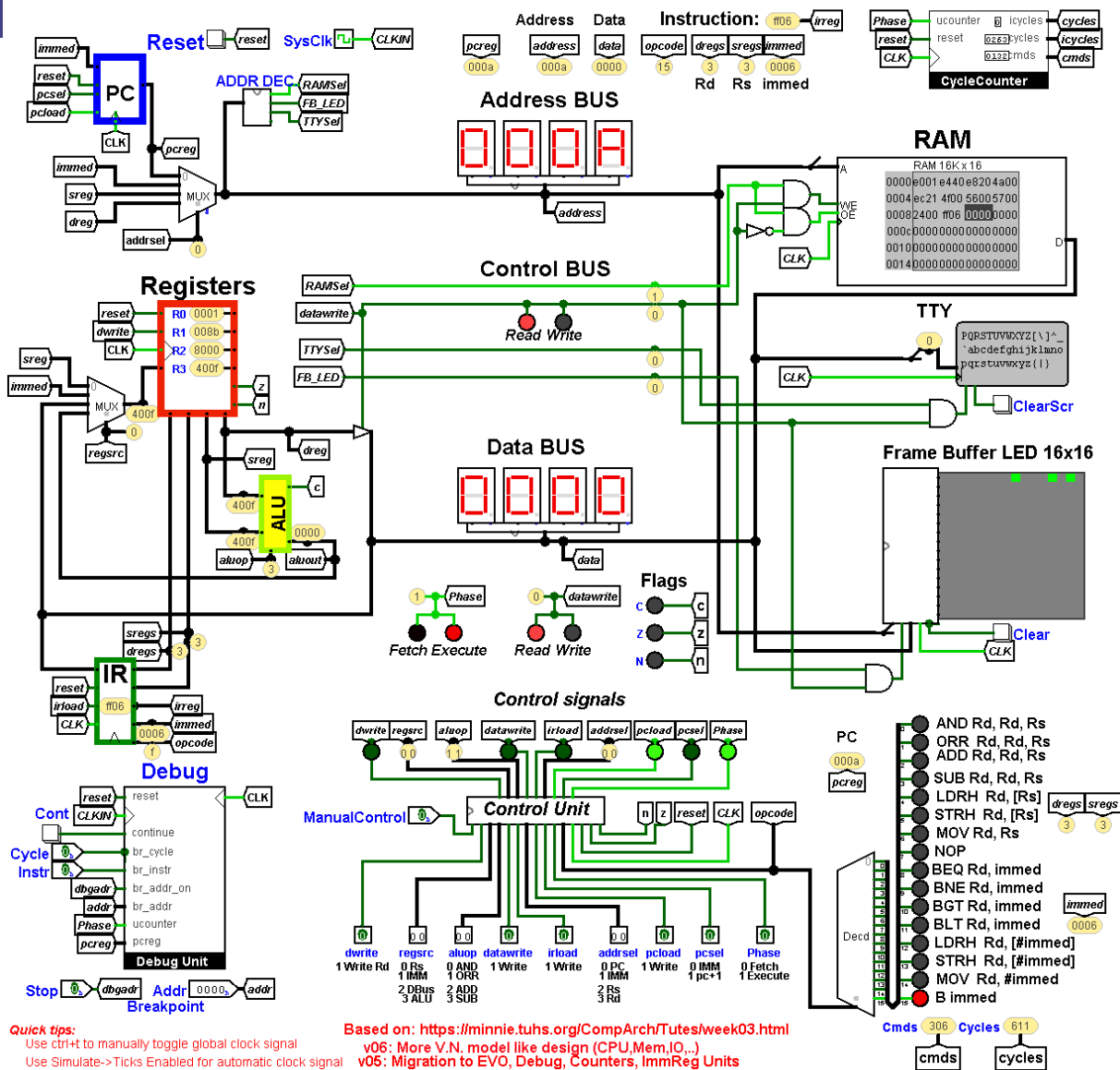


Mikro MiMo model CPE

Mikro MiMo model CPE – krmilni signali v0

PROGRAM Zbirnik	Korak	Address Vir naslova	ReadWrite Vpis/branje RAM pomn.	dwrite Vpis reg. Rd	RegSrc Vhod v reg.	aluop ALU Operac.	ADR Sub Rx,PC,#odm	lrload Vpis v uk. reg. IR	PC
Opis		0..PC, 1..Rn	0..branje, 1..pisanje	0..ne, 1..da	0..Databus, 1..ALU, 2..Immediate	0..ADD, 2..SUB	0..ne, 1..da	0..ne, 1..da	Naslov ukaza
adr r0,STEV1	(PC+4) Fetch							1	0x2C
<i>(sub r0,pc,#0x0c)</i>	Execute			1	1	2	1		
ldr r1,[r0]	(PC+4) Fetch							1	0x30
<i>(sub r0,pc,#0x10)</i>	Execute	1		1	0				
adr r0,STEV2	(PC+4) Fetch							1	0x34
	Execute			1	1	2	1		
ldr r2,[r0]	(PC+4) Fetch							1	0x38
	Execute	1		1	0				
add r3,r1,r2	(PC+4) Fetch							1	0x3C
	Execute			1	1	0			
adr r0,REZ	(PC+4) Fetch							1	0x40
<i>(sub r0,pc,#0x18)</i>	Execute			1	1	2	1		
str r3,[r0]	(PC+4) Fetch							1	0x44
	Execute	1	1						

Mini MiMo - HW Simple CPU Model RA (6. pogl)



Quick tips:

- Use ctrl+T to manually toggle global clock signal
- Use Simulate->Ticks Enabled for automatic clock signal

Based on: <https://minnie.tuhs.org/CompArch/Tutes/week03.html>

- v06: More V.N. model like design (CPU, Mem, IO...)
- v05: Migration to EVO, Debug, Counters, ImmReg Units

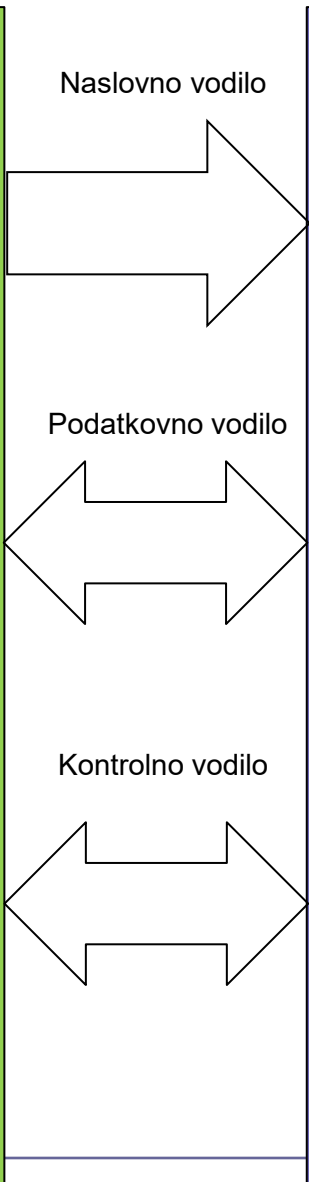
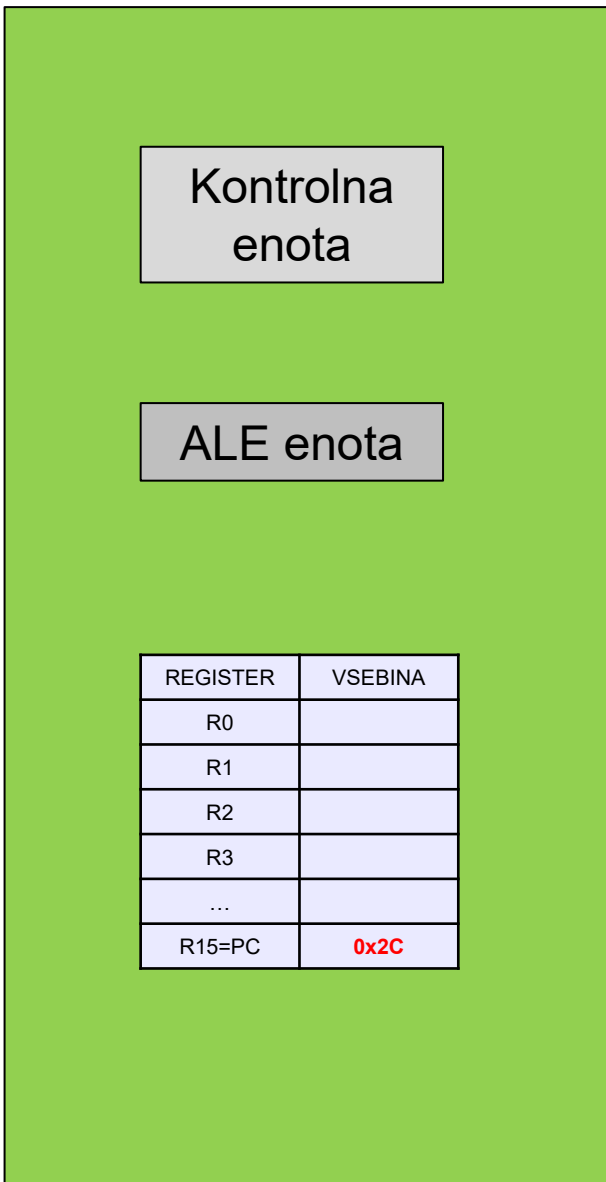
https://github.com/LAPSyLAB/RALab-STM32H7/tree/main/MiniMiMo_HW_CPE_Model

https://github.com/LAPSyLAB/RALab-STM32H7/tree/main/LogisimEVO_vezja/Prispevki

UKAZ	KORAK	Komentar
		Začetno stanje



Primer izvedbe programa



VSEBINA	NASLOV	OZNAKA
0x40	0x20	STEV1
0	0x21	
0	0x22	
0	0x23	
0x10	0x24	STEV2
0	0x25	
0	0x26	
0	0x27	
0	0x28	REZ
0	0x29	
0	0x2A	
0	0x2B	
0x14	0x2C	ADR R0,STEV1
0x10	0x2D	
0x1F	0x2E	
0xE5	0x2F	
	0x30	LDR R1,[R0]
	...	
	0x34...	ADR R0,STEV2
	0x38...	LDR R2,[R0]
	0x3C...	ADD R3,R2,R1
	0x40...	ADR R0,REZ
	0x44...	STR R3,[R0]
	...	

PROGRAM Zbirnik

```

→ ADR R0,STEV1
LDR R1,[R0]
ADR R0,STEV2
LDR R2,[R0]
ADD R3,R1,R2
ADR R0,REZ
STR R3,[R0]

```

Strojni jezik

```

→ 0xE24F0014
0xE5901000
0xE24F0018
0xE5902000
0xE0823001
0xE24F0020
0xE5803000

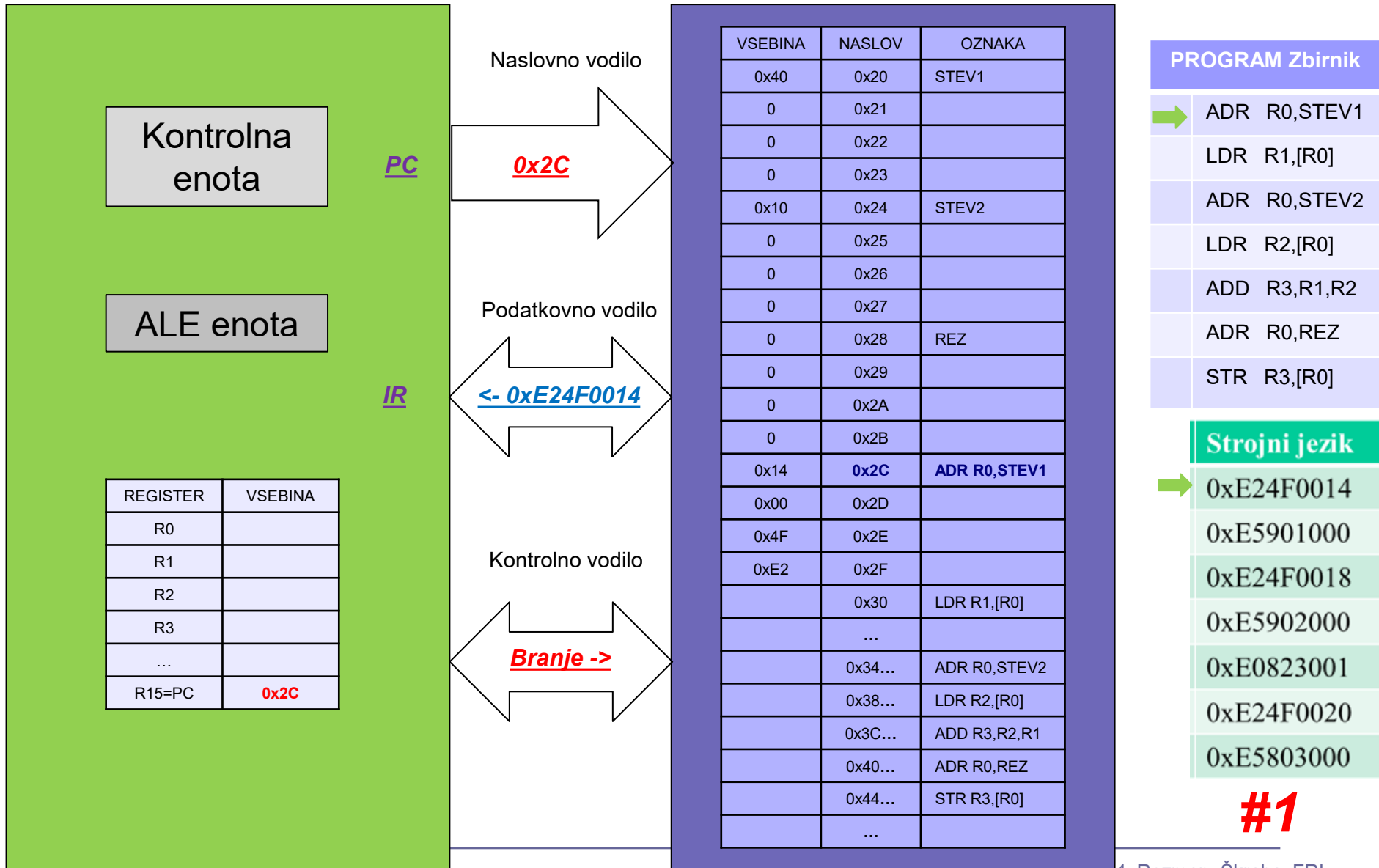
```

#0

UKAZ	KORAK	Komentar
ADR R0,STEV1	FETCH	Branje 1. ukaza



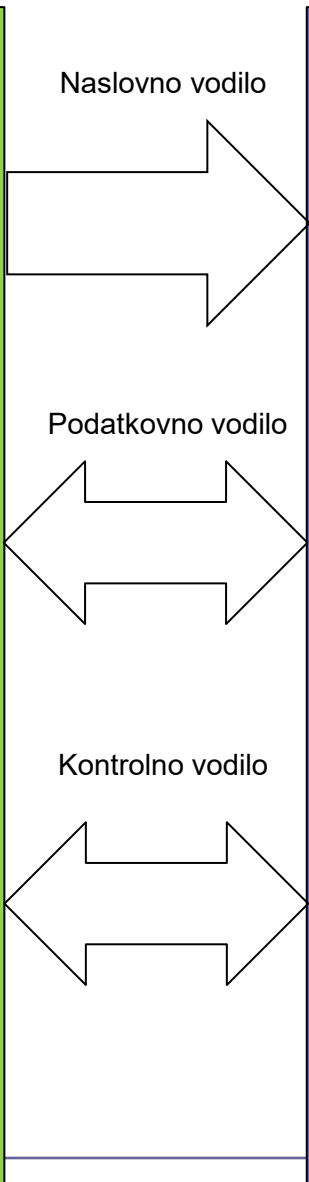
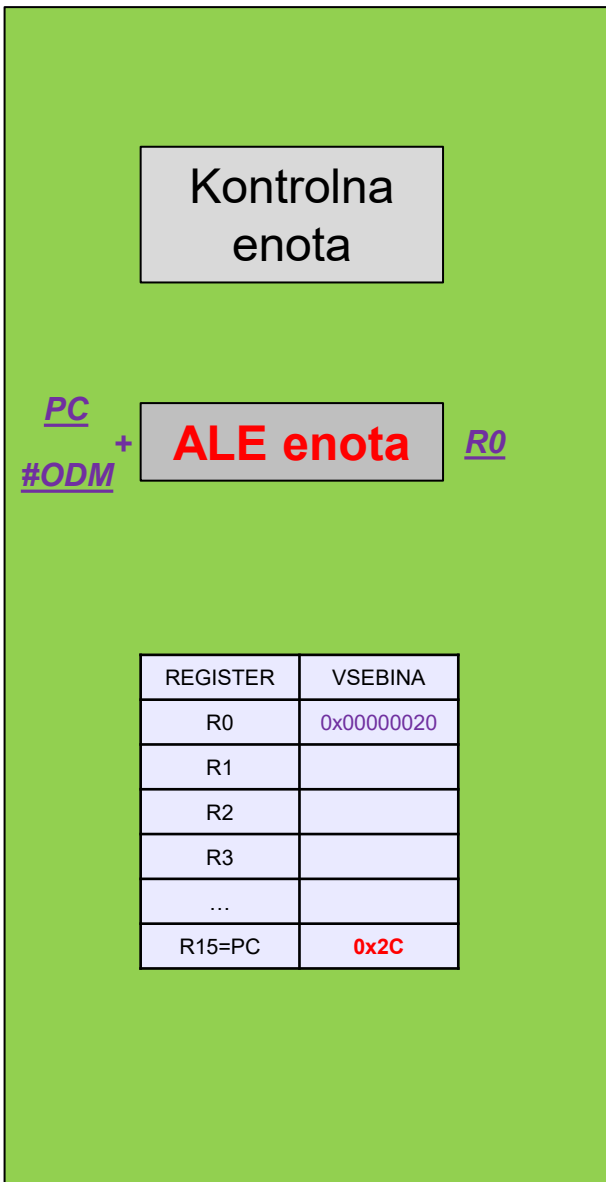
Primer izvedbe programa



UKAZ	KORAK	Komentar
ADR R0,STEV1	EXECUTE	ALE: R0 <- PC +- ODMIK



Primer izvedbe programa



VSEBINA	NASLOV	OZNAKA
0x40	0x20	STEV1
0	0x21	
0	0x22	
0	0x23	
0x10	0x24	STEV2
0	0x25	
0	0x26	
0	0x27	
0	0x28	REZ
0	0x29	
0	0x2A	
0	0x2B	
0x14	0x2C	ADR R0,STEV1
0x10	0x2D	
0x1F	0x2E	
0xE5	0x2F	
	0x30	LDR R1,[R0]
	...	
	0x34...	ADR R0,STEV2
	0x38...	LDR R2,[R0]
	0x3C...	ADD R3,R2,R1
	0x40...	ADR R0,REZ
	0x44...	STR R3,[R0]
	...	

PROGRAM Zbirnik

```

ADR R0,STEV1
LDR R1,[R0]
ADR R0,STEV2
LDR R2,[R0]
ADD R3,R1,R2
ADR R0,REZ
STR R3,[R0]

```

Strojni jezik

```

0xE24F0014
0xE5901000
0xE24F0018
0xE5902000
0xE0823001
0xE24F0020
0xE5803000

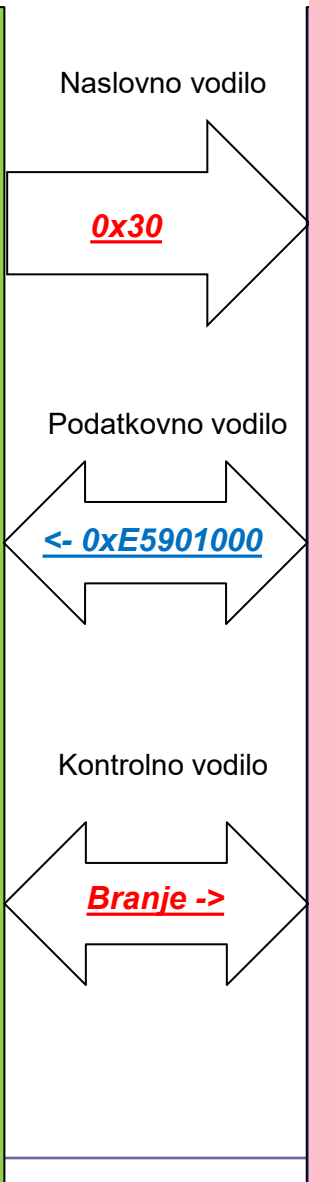
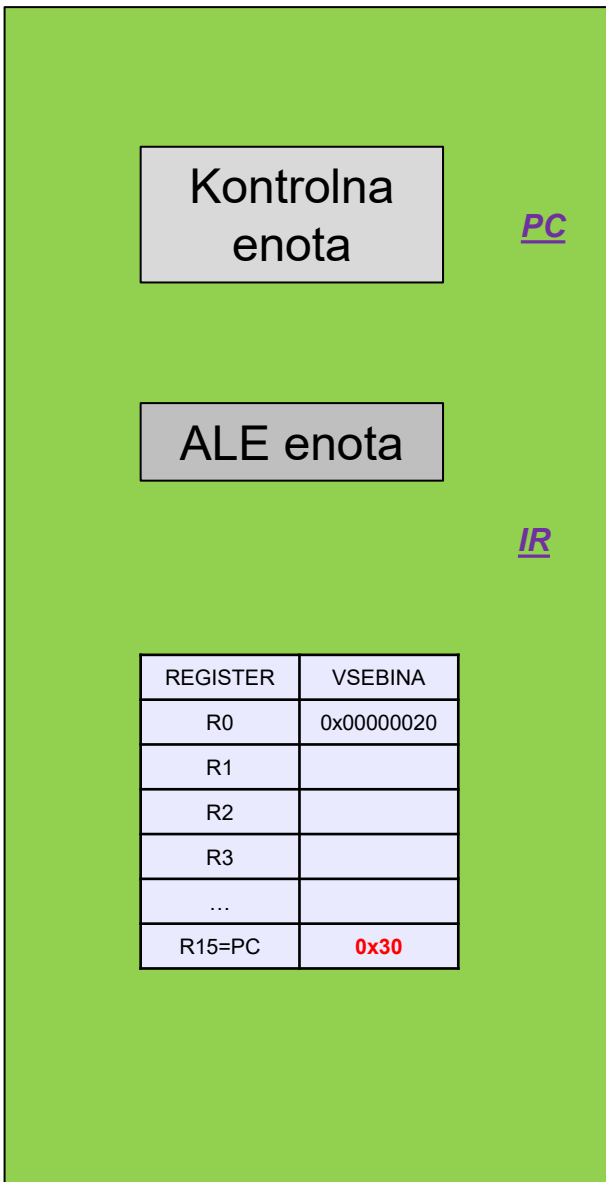
```

#2

UKAZ	KORAK	Komentar
LDR R1,[R0]	FETCH	Branje 2. ukaza



Primer izvedbe programa



VSEBINA	NASLOV	OZNAKA
0x40	0x20	STEV1
0	0x21	
0	0x22	
0	0x23	
0x10	0x24	STEV2
0	0x25	
0	0x26	
0	0x27	
0	0x28	REZ
0	0x29	
0	0x2A	
0	0x2B	
0x14	0x2C	ADR R0,STEV1
0x10	0x2D	
0x1F	0x2E	
0xE5	0x2F	
	0x30	LDR R1,[R0]
	...	
	0x34...	ADR R0,STEV2
	0x38...	LDR R2,[R0]
	0x3C...	ADD R3,R2,R1
	0x40...	ADR R0,REZ
	0x44...	STR R3,[R0]
	...	

PROGRAM Zbirnik

ADR R0,STEV1
LDR R1,[R0]
ADR R0,STEV2
LDR R2,[R0]
ADD R3,R1,R2
ADR R0,REZ
STR R3,[R0]

Strojni jezik

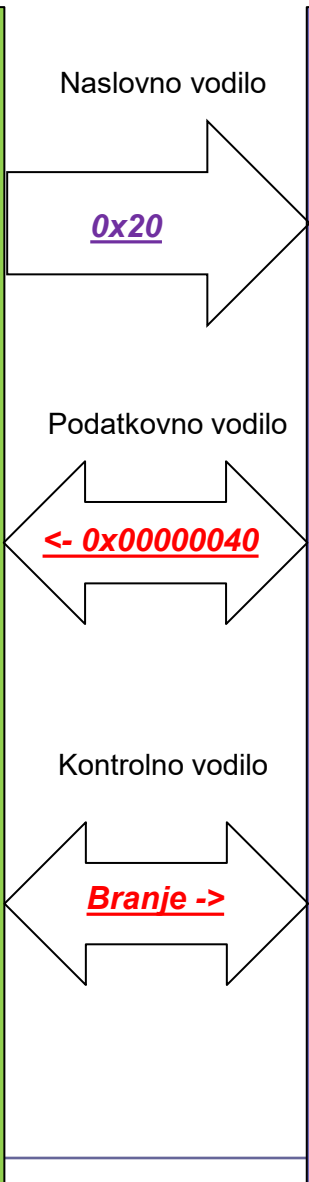
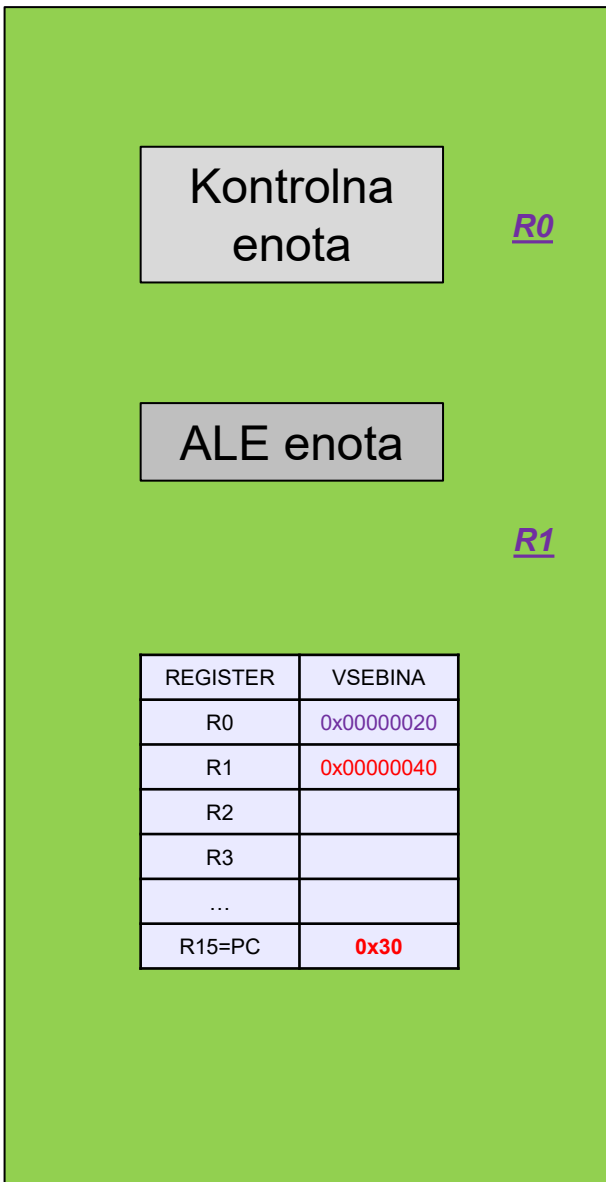
0xE24F0014
0xE5901000
0xE24F0018
0xE5902000
0xE0823001
0xE24F0020
0xE5803000

#3

UKAZ	KORAK	Komentar
LDR R1,[R0]	EXECUTE	Branje operanda M[R0] v R1



Primer izvedbe programa



VSEBINA	NASLOV	OZNAKA
0x40	0x20	STEV1
0	0x21	
0	0x22	
0	0x23	
0x10	0x24	STEV2
0	0x25	
0	0x26	
0	0x27	
0	0x28	REZ
0	0x29	
0	0x2A	
0	0x2B	
0x14	0x2C	ADR R0,STEV1
0x10	0x2D	
0x1F	0x2E	
0xE5	0x2F	
	0x30	LDR R1,[R0]
	...	
	0x34...	ADR R0,STEV2
	0x38...	LDR R2,[R0]
	0x3C...	ADD R3,R2,R1
	0x40...	ADR R0,REZ
	0x44...	STR R3,[R0]
	...	

PROGRAM Zbirnik	
ADR R0,STEV1	
LDR R1,[R0]	
ADR R0,STEV2	
LDR R2,[R0]	
ADD R3,R1,R2	
ADR R0,REZ	
STR R3,[R0]	

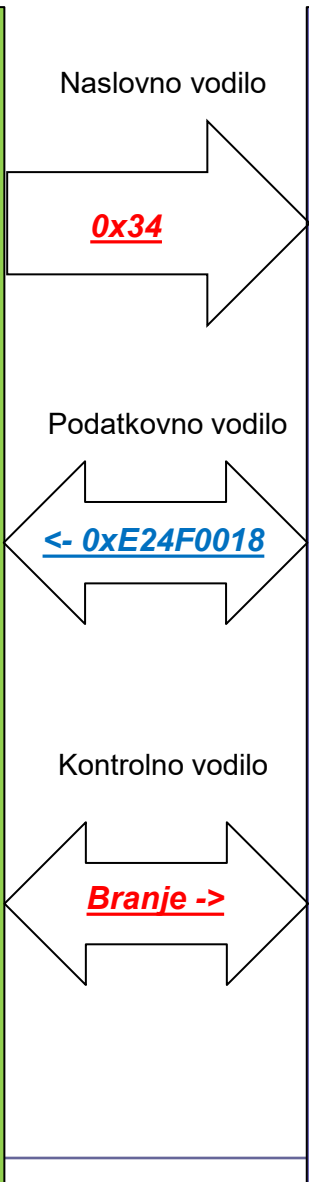
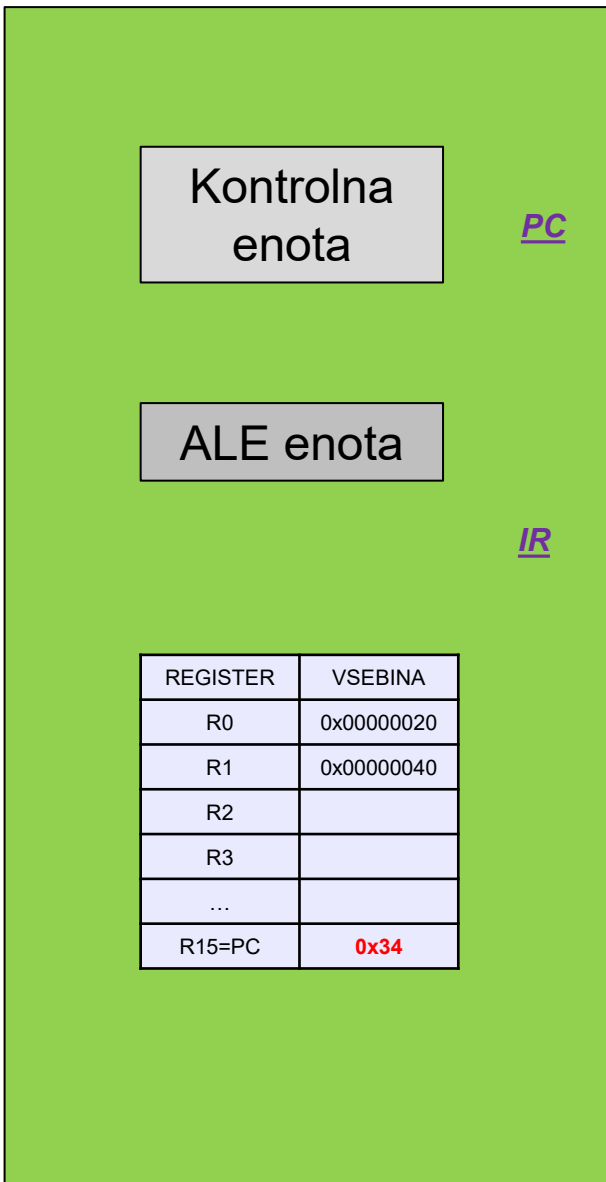
Strojni jezik	
0xE24F0014	
0xE5901000	
0xE24F0018	
0xE5902000	
0xE0823001	
0xE24F0020	
0xE5803000	

#4

UKAZ	KORAK	Komentar
ADR R0,STEV2	FETCH	Branje 3. ukaza



Primer izvedbe programa



VSEBINA	NASLOV	OZNAKA
0x40	0x20	STEV1
0	0x21	
0	0x22	
0	0x23	
0x10	0x24	STEV2
0	0x25	
0	0x26	
0	0x27	
0	0x28	REZ
0	0x29	
0	0x2A	
0	0x2B	
0x14	0x2C	ADR R0,STEV1
0x00	0x2D	
0x4F	0x2E	
0xE2	0x2F	
	0x30	LDR R1,[R0]
	...	
	0x34...	ADR R0,STEV2
	0x38...	LDR R2,[R0]
	0x3C...	ADD R3,R2,R1
	0x40...	ADR R0,REZ
	0x44...	STR R3,[R0]
	...	

PROGRAM Zbirnik

ADR R0,STEV1
LDR R1,[R0]
ADR R0,STEV2
LDR R2,[R0]
ADD R3,R1,R2
ADR R0,REZ
STR R3,[R0]

Strojni jezik

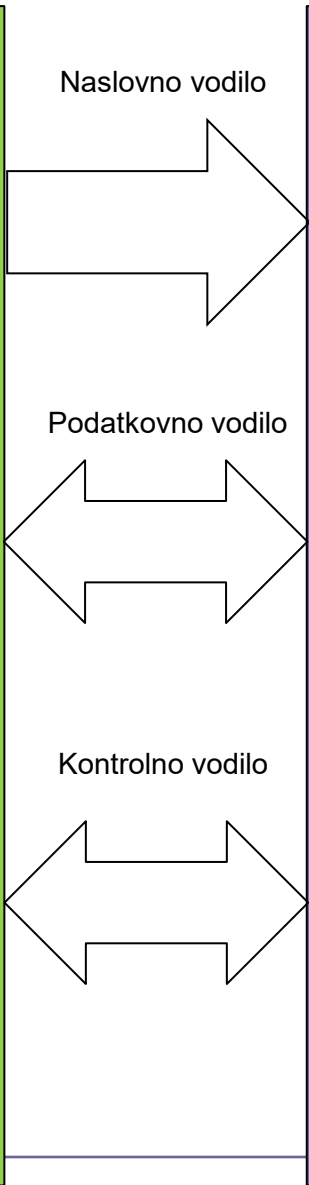
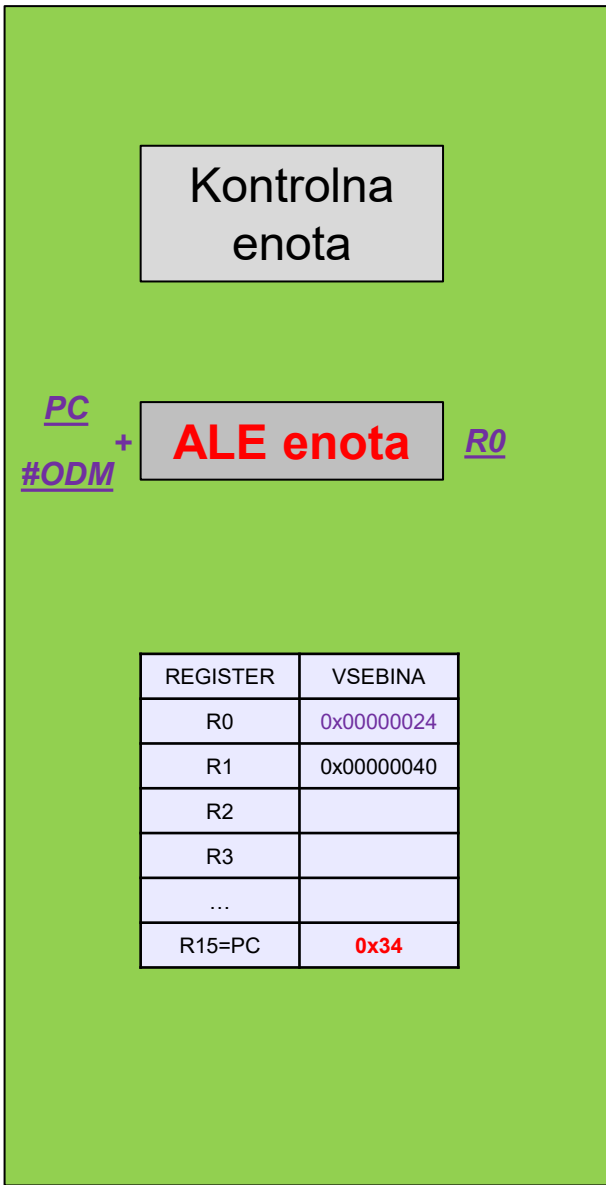
0xE24F0014
0xE5901000
0xE24F0018
0xE5902000
0xE0823001
0xE24F0020
0xE5803000

#5

UKAZ	KORAK	Komentar
ADR R0,STEV2	EXECUTE	ALE: R0 <- PC +- ODMIK



Primer izvedbe programa



VSEBINA	NASLOV	OZNAKA
0x40	0x20	STEV1
0	0x21	
0	0x22	
0	0x23	
0x10	0x24	STEV2
0	0x25	
0	0x26	
0	0x27	
0	0x28	REZ
0	0x29	
0	0x2A	
0	0x2B	
0x14	0x2C	ADR R0,STEV1
0x10	0x2D	
0x1F	0x2E	
0xE5	0x2F	
	0x30	LDR R1,[R0]
	...	
	0x34...	ADR R0,STEV2
	0x38...	LDR R2,[R0]
	0x3C...	ADD R3,R2,R1
	0x40...	ADR R0,REZ
	0x44...	STR R3,[R0]
	...	

PROGRAM Zbirnik

ADR R0,STEV1
LDR R1,[R0]
ADR R0,STEV2
LDR R2,[R0]
ADD R3,R1,R2
ADR R0,REZ
STR R3,[R0]

Strojni jezik

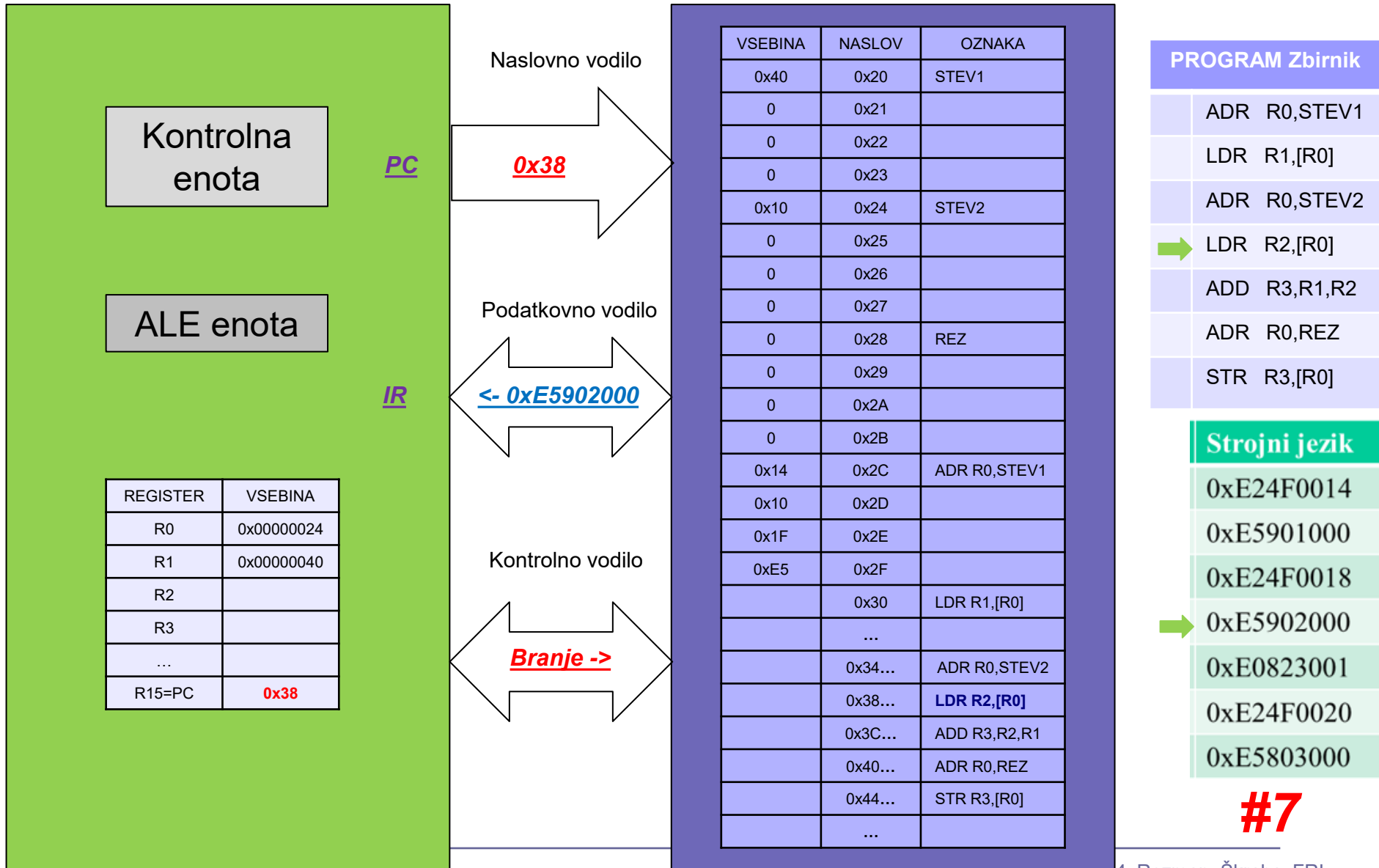
0xE24F0014
0xE5901000
0xE24F0018
0xE5902000
0xE0823001
0xE24F0020
0xE5803000

#6

UKAZ	KORAK	Komentar
LDR R2,[R0]	FETCH	Branje 4. ukaza



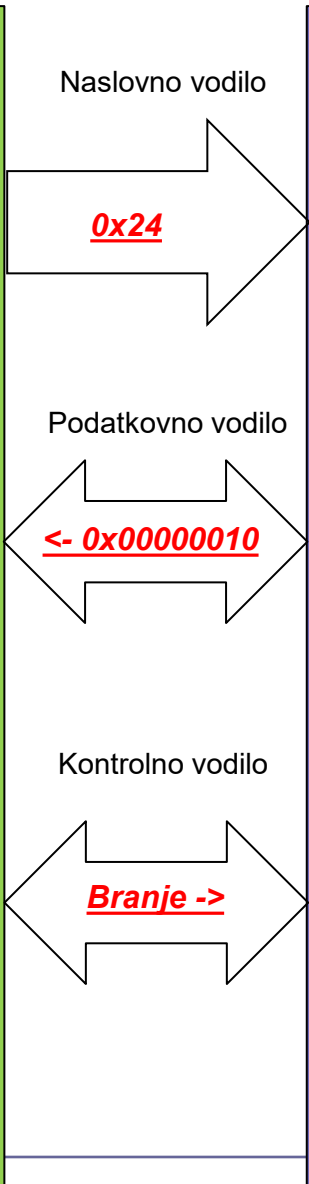
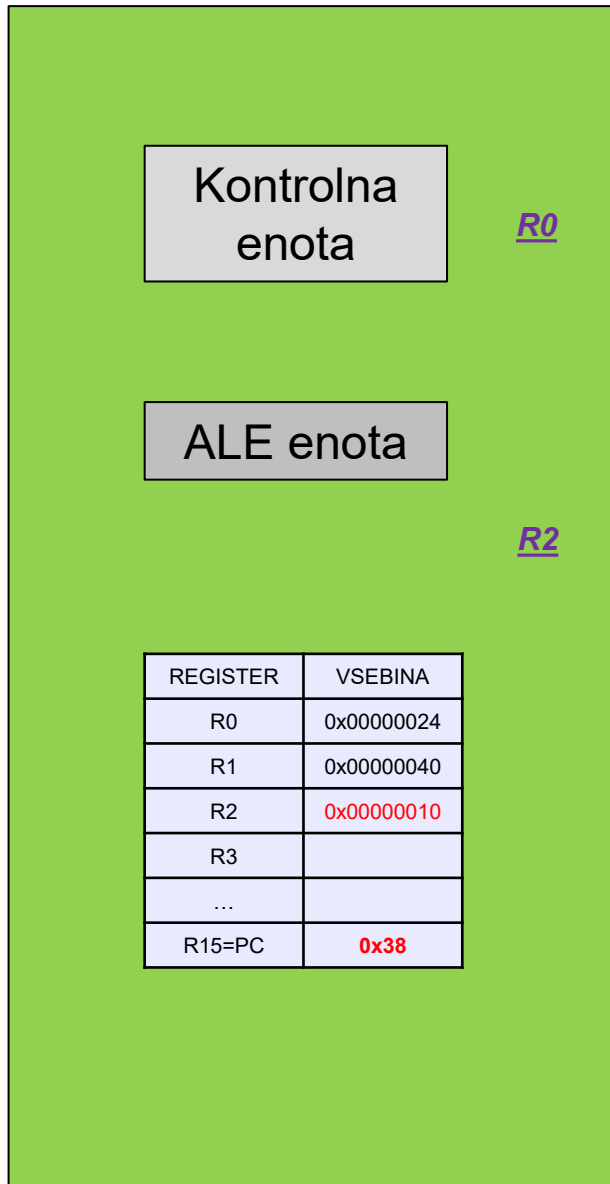
Primer izvedbe programa



UKAZ	KORAK	Komentar
LDR R2,[R0]	EXECUTE	Branje operanda M[R0] v R1



Primer izvedbe programa



VSEBINA	NASLOV	OZNAKA
0x40	0x20	STEV1
0	0x21	
0	0x22	
0	0x23	
0x10	0x24	STEV2
0	0x25	
0	0x26	
0	0x27	
0	0x28	REZ
0	0x29	
0	0x2A	
0	0x2B	
0x14	0x2C	ADR R0,STEV1
0x10	0x2D	
0x1F	0x2E	
0xE5	0x2F	
	0x30	LDR R1,[R0]
	...	
	0x34...	ADR R0,STEV2
	0x38...	LDR R2,[R0]
	0x3C...	ADD R3,R2,R1
	0x40...	ADR R0,REZ
	0x44...	STR R3,[R0]
	...	

PROGRAM Zbirnik

ADR R0,STEV1
LDR R1,[R0]
ADR R0,STEV2
LDR R2,[R0]
ADD R3,R1,R2
ADR R0,REZ
STR R3,[R0]

Strojni jezik

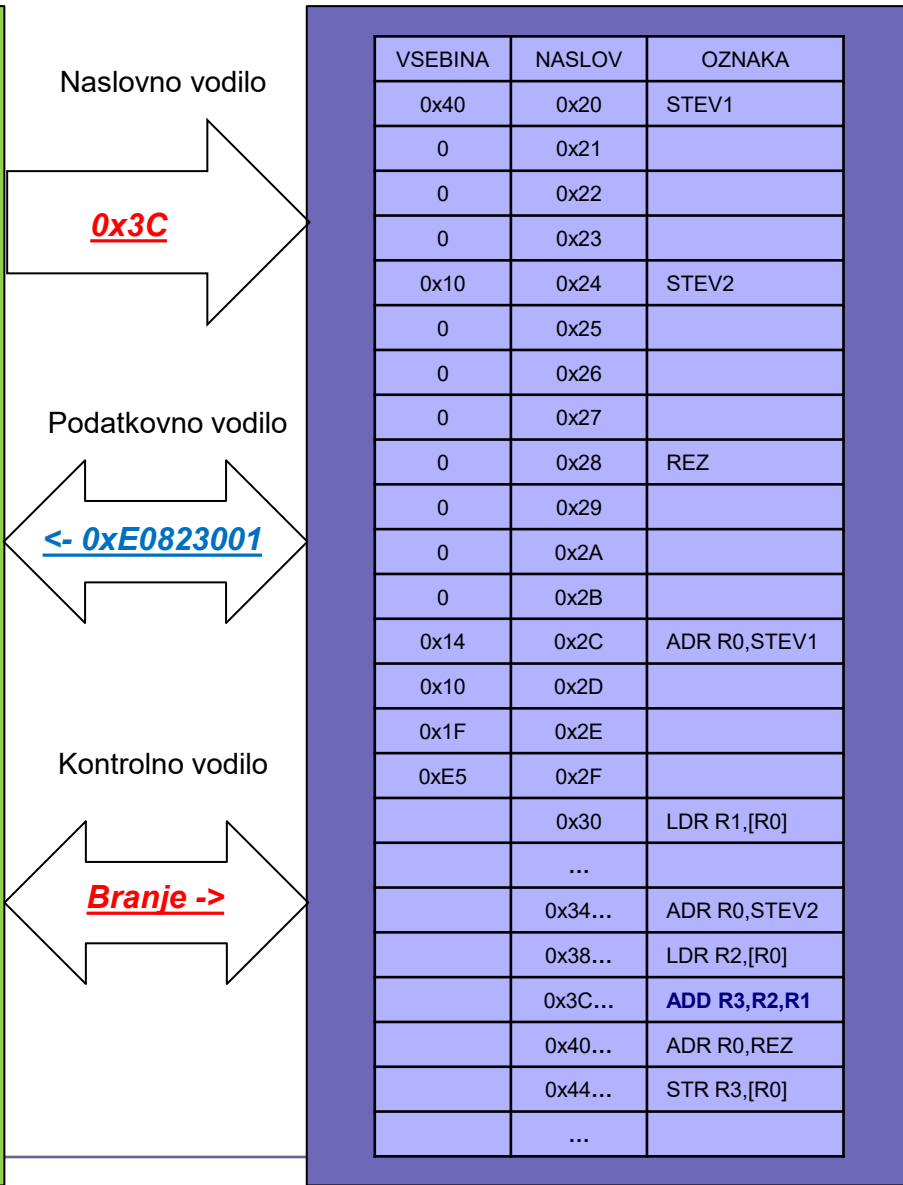
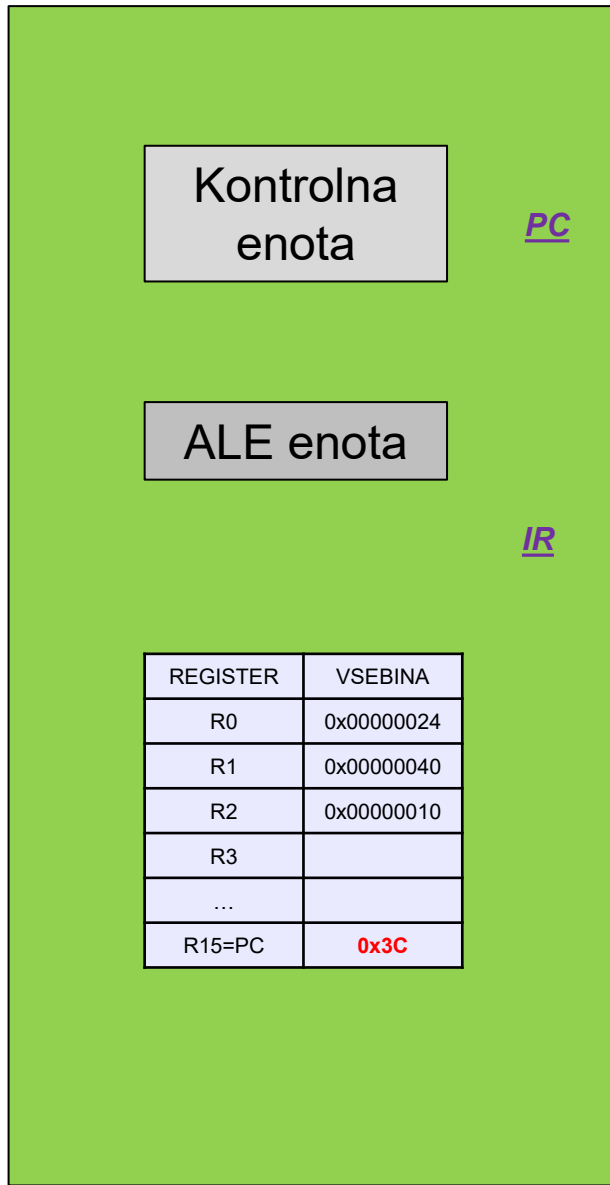
0xE24F0014
0xE5901000
0xE24F0018
0xE5902000
0xE0823001
0xE24F0020
0xE5803000

#8

UKAZ	KORAK	Komentar
ADD R3,R2,R1	FETCH	Branje 5. ukaza



Primer izvedbe programa



PROGRAM Zbirnik

ADR R0,STEV1
LDR R1,[R0]
ADR R0,STEV2
LDR R2,[R0]
ADD R3,R1,R2
ADR R0,REZ
STR R3,[R0]

Strojni jezik

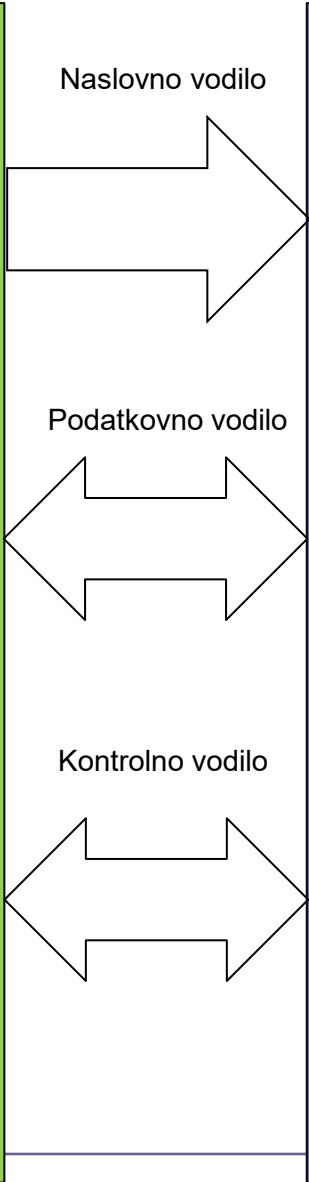
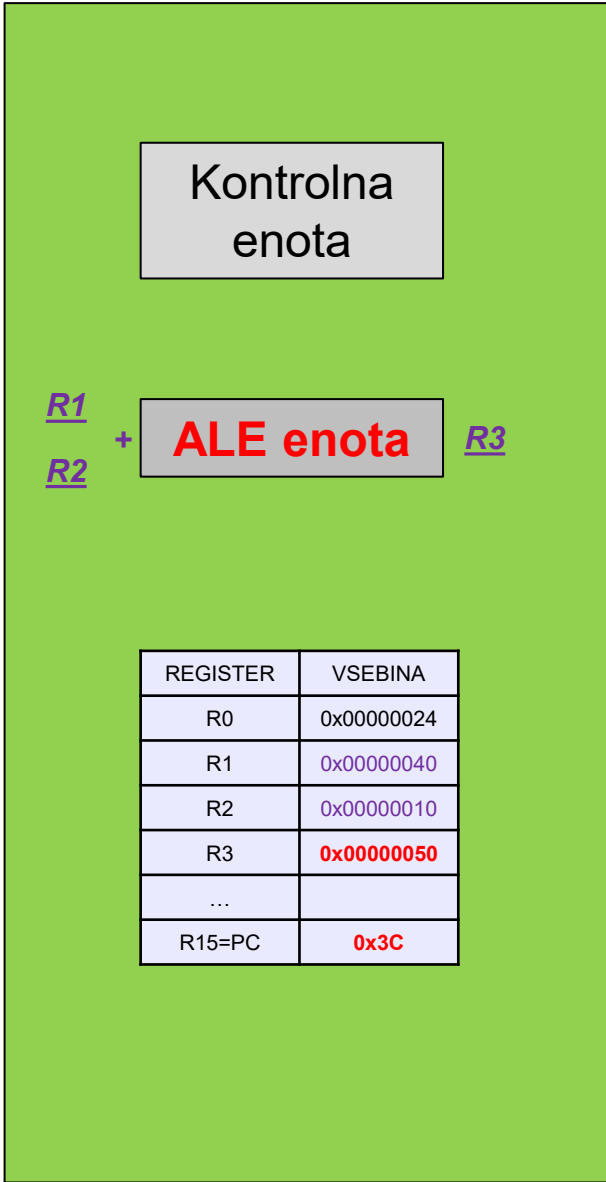
0xE24F0014
0xE5901000
0xE24F0018
0xE5902000
0xE0823001
0xE24F0020
0xE5803000

#9

UKAZ	KORAK	Komentar
ADD R3,R2,R1	EXECUTE	ALE: R3 <- R2 + R1 (vsota)



Primer izvedbe programa



VSEBINA	NASLOV	OZNAKA
0x40	0x20	STEV1
0	0x21	
0	0x22	
0	0x23	
0x10	0x24	STEV2
0	0x25	
0	0x26	
0	0x27	
0	0x28	REZ
0	0x29	
0	0x2A	
0	0x2B	
0x14	0x2C	ADR R0,STEV1
0x10	0x2D	
0x1F	0x2E	
0xE5	0x2F	
	0x30	LDR R1,[R0]
	...	
	0x34...	ADR R0,STEV2
	0x38...	LDR R2,[R0]
	0x3C...	ADD R3,R2,R1
	0x40...	ADR R0,REZ
	0x44...	STR R3,[R0]
	...	

PROGRAM Zbirnik

ADR R0,STEV1
LDR R1,[R0]
ADR R0,STEV2
LDR R2,[R0]
ADD R3,R1,R2
ADR R0,REZ
STR R3,[R0]

Strojni jezik

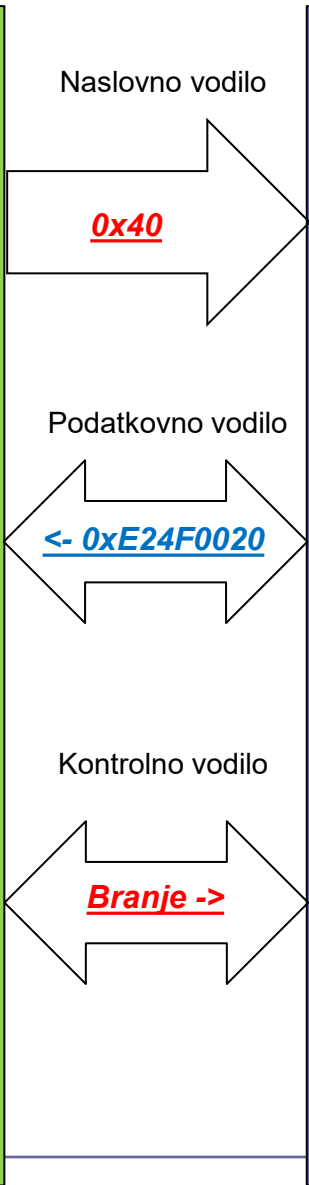
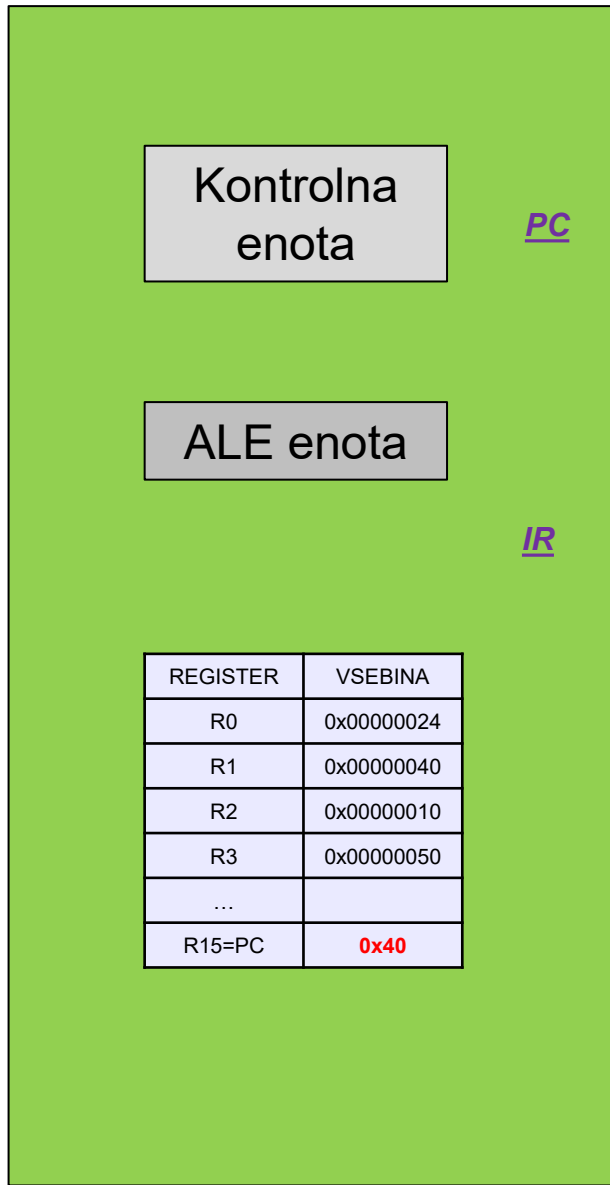
0xE24F0014
0xE5901000
0xE24F0018
0xE5902000
0xE0823001
0xE24F0020
0xE5803000

#10

UKAZ	KORAK	Komentar
ADR R0,REZ	FETCH	Branje 6. ukaza



Primer izvedbe programa



VSEBINA	NASLOV	OZNAKA
0x40	0x20	STEV1
0	0x21	
0	0x22	
0	0x23	
0x10	0x24	STEV2
0	0x25	
0	0x26	
0	0x27	
0	0x28	REZ
0	0x29	
0	0x2A	
0	0x2B	
0x14	0x2C	ADR R0,STEV1
0x00	0x2D	
0x4F	0x2E	
0xE2	0x2F	
	0x30	LDR R1,[R0]
	...	
	0x34...	ADR R0,STEV2
	0x38...	LDR R2,[R0]
	0x3C...	ADD R3,R2,R1
	0x40...	ADR R0,REZ
	0x44...	STR R3,[R0]
	...	

PROGRAM Zbirnik	
ADR R0,STEV1	
LDR R1,[R0]	
ADR R0,STEV2	
LDR R2,[R0]	
ADD R3,R1,R2	
ADR R0,REZ	
STR R3,[R0]	

Strojni jezik	
0xE24F0014	
0xE5901000	
0xE24F0018	
0xE5902000	
0xE0823001	
0xE24F0020	
0xE5803000	

#11

UKAZ	KORAK	Komentar
ADR R0,REZ	EXECUTE	ALE: R0 <- PC +- ODMIK



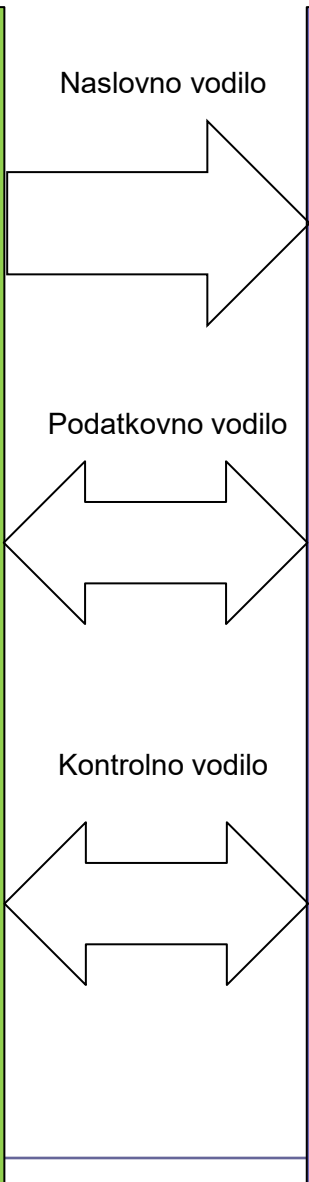
Primer izvedbe programa

Kontrolna enota

PC + **ALE enota** R0

#ODM

REGISTER	VSEBINA
R0	0x00000028
R1	0x00000040
R2	0x00000010
R3	0x00000050
...	
R15=PC	0x40



VSEBINA	NASLOV	OZNAKA
0x40	0x20	STEV1
0	0x21	
0	0x22	
0	0x23	
0x10	0x24	STEV2
0	0x25	
0	0x26	
0	0x27	
0	0x28	REZ
0	0x29	
0	0x2A	
0	0x2B	
0x14	0x2C	ADR R0,STEV1
0x10	0x2D	
0x1F	0x2E	
0xE5	0x2F	
	0x30	LDR R1,[R0]
	...	
	0x34...	ADR R0,STEV2
	0x38...	LDR R2,[R0]
	0x3C...	ADD R3,R2,R1
	0x40...	ADR R0,REZ
	0x44...	STR R3,[R0]
	...	

PROGRAM Zbirnik

ADR R0,STEV1
LDR R1,[R0]
ADR R0,STEV2
LDR R2,[R0]
ADD R3,R1,R2
➔ ADR R0,REZ
STR R3,[R0]

Strojni jezik

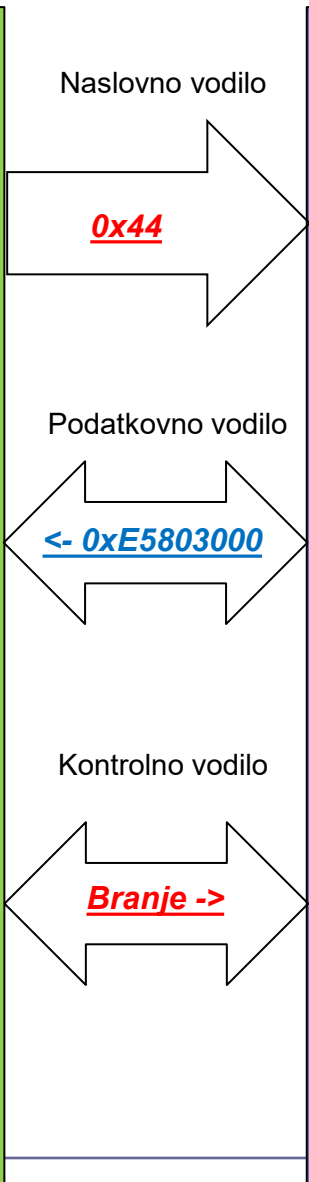
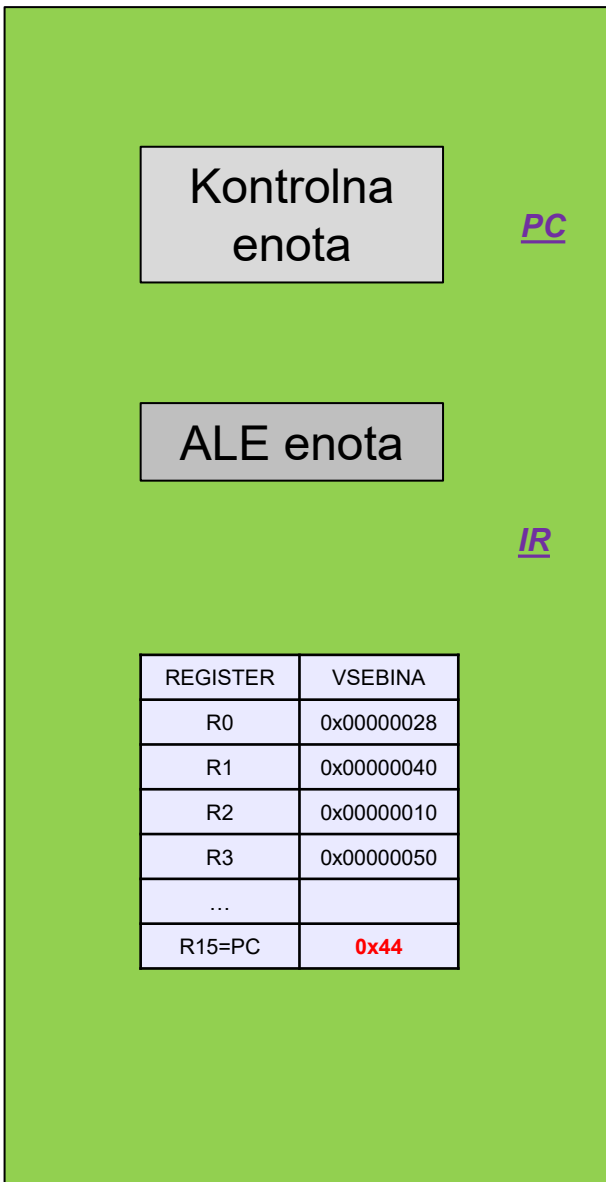
0xE24F0014
0xE5901000
0xE24F0018
0xE5902000
0xE0823001
➔ 0xE24F0020
0xE5803000

#12

UKAZ	KORAK	Komentar
STR R3,[R0]	FETCH	Branje 7. ukaza



Primer izvedbe programa



VSEBINA	NASLOV	OZNAKA
0x40	0x20	STEV1
0	0x21	
0	0x22	
0	0x23	
0x10	0x24	STEV2
0	0x25	
0	0x26	
0	0x27	
0	0x28	REZ
0	0x29	
0	0x2A	
0	0x2B	
0x14	0x2C	ADR R0,STEV1
0x10	0x2D	
0x1F	0x2E	
0xE5	0x2F	
	0x30	LDR R1,[R0]
	...	
	0x34...	ADR R0,STEV2
	0x38...	LDR R2,[R0]
	0x3C...	ADD R3,R2,R1
	0x40...	ADR R0,REZ
	0x44...	STR R3,[R0]
	...	

PROGRAM Zbirnik

ADR R0,STEV1
LDR R1,[R0]
ADR R0,STEV2
LDR R2,[R0]
ADD R3,R1,R2
ADR R0,REZ
STR R3,[R0]

Strojni jezik

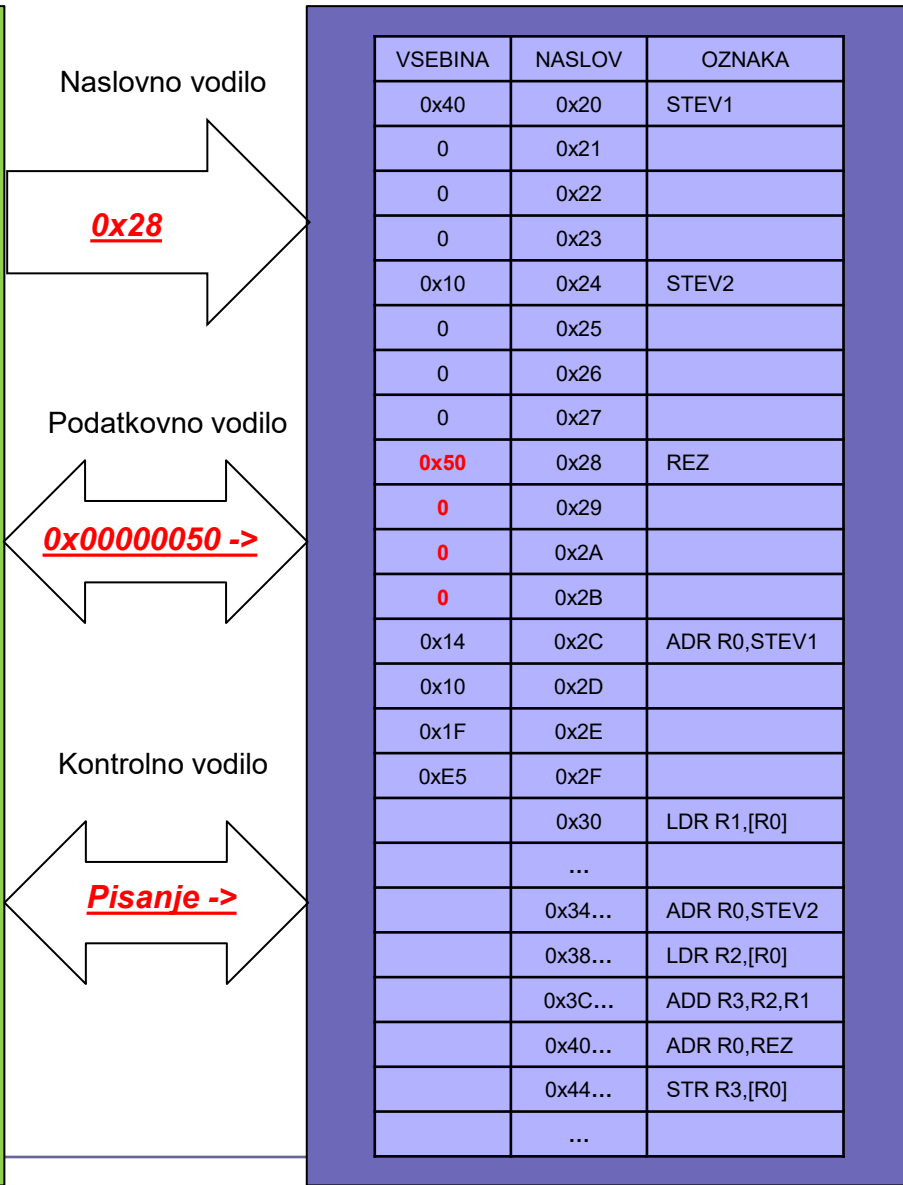
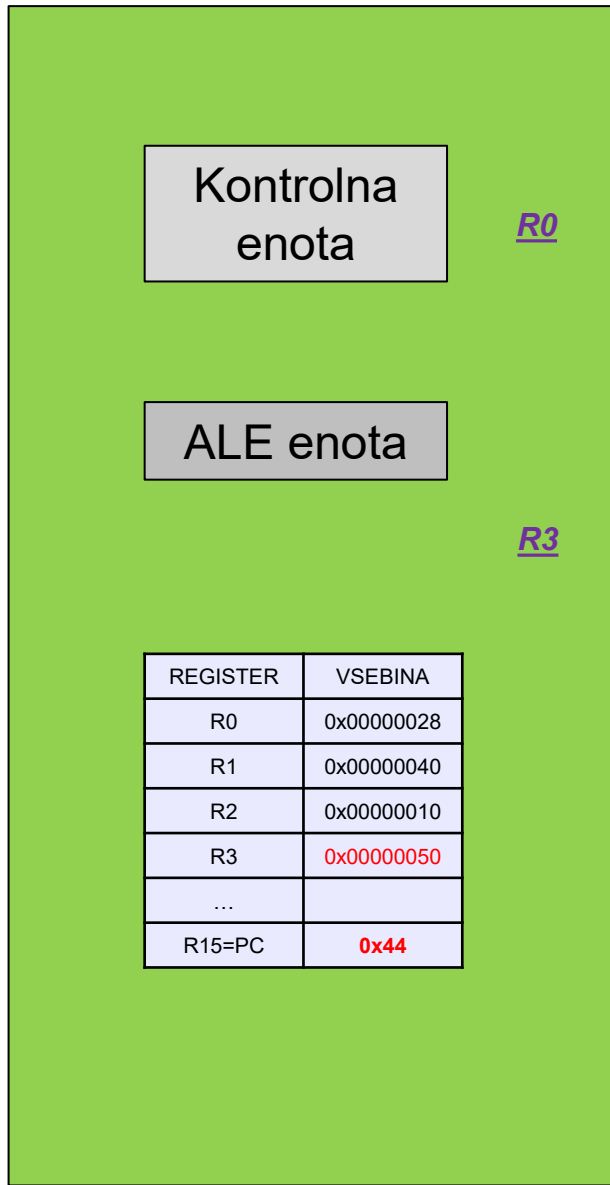
0xE24F0014
0xE5901000
0xE24F0018
0xE5902000
0xE0823001
0xE24F0020
0xE5803000

#13

UKAZ	KORAK	Komentar
STR R3,REZ	EXECUTE	Shranitev R3 v M[REZ]



Primer izvedbe programa



```

PROGRAM Zbirnik
ADR R0,STEV1
LDR R1,[R0]
ADR R0,STEV2
LDR R2,[R0]
ADD R3,R1,R2
ADR R0,REZ
STR R3,[R0]

```

```

Strojni jezik
0xE24F0014
0xE5901000
0xE24F0018
0xE5902000
0xE0823001
0xE24F0020
0xE5803000

```

#14



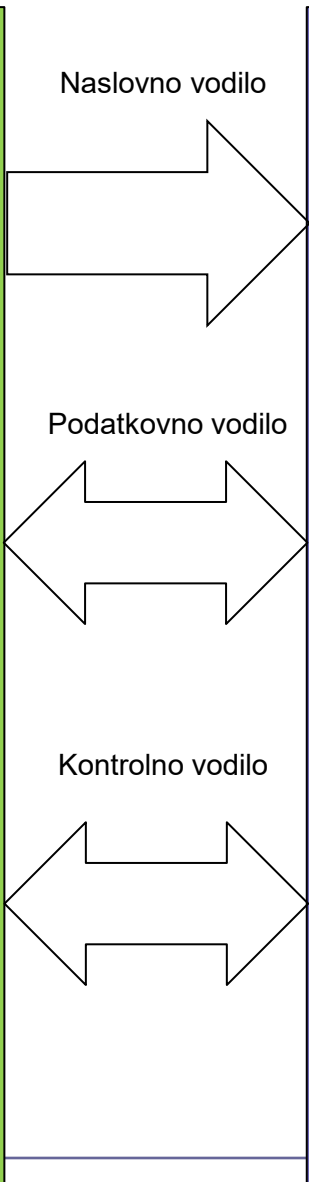
UKAZ	KORAK	Komentar
?	FETCH	Končno stanje ?

Primer izvedbe programa

Kontrolna enota

ALE enota

REGISTER	VSEBINA
R0	0x00000028
R1	0x00000040
R2	0x00000010
R3	0x00000050
...	
R15=PC	0x48



BESEDA	NASLOV	OZNAKA
0x40	0x20	STEV1
0	0x21	
0	0x22	
0	0x23	
0x10	0x24	STEV2
0	0x25	
0	0x26	
0	0x27	
0x50	0x28	REZ
0	0x29	
0	0x2A	
0	0x2B	
0x14	0x2C	ADR R0,STEV1
0x10	0x2D	
0x1F	0x2E	
0xE5	0x2F	
0x14	0x30	LDR R1,[R0]
0x20	...	
0x1F	0x34...	ADR R0,STEV2
0xE5	0x38...	LDR R2,[R0]
0x01	0x3C...	ADD R3,R2,R1
	0x40...	ADR R0,REZ
0x18	0x44...	STR R3,[R0]
	0x48...	???

PROGRAM Zbirnik

ADR R0,STEV1
LDR R1,[R0]
ADR R0,STEV2
LDR R2,[R0]
ADD R3,R1,R2
ADR R0,REZ
STR R3,[R0]

Strojni jezik

0xE24F0014
0xE5901000
0xE24F0018
0xE5902000
0xE0823001
0xE24F0020
0xE5803000

#15

Tabela za izvedbo programa

<i>CPE</i>		<i>CPE</i>	<i>VODILA - vsebina</i>			<i>Pomnilnik</i>
<i>Opis</i>	<i>CPE</i>	<i>Opis</i>	<i>Naslovno</i>	<i>Podatkovno</i>	<i>Kontrolno</i>	<i>Opis</i>
ADR R0,STEV1	FETCH					
	EXECUTE					
LDR R1,[R0]	FETCH					
	EXECUTE					
ADR R0,STEV2	FETCH					
	EXECUTE					
LDR R2,[R0]	FETCH					
	EXECUTE					
ADD R3,R1,R2	FETCH					
	EXECUTE					
ADR R0,REZ	FETCH					
	EXECUTE					
STR R3,[R0]	FETCH					
	EXECUTE					