

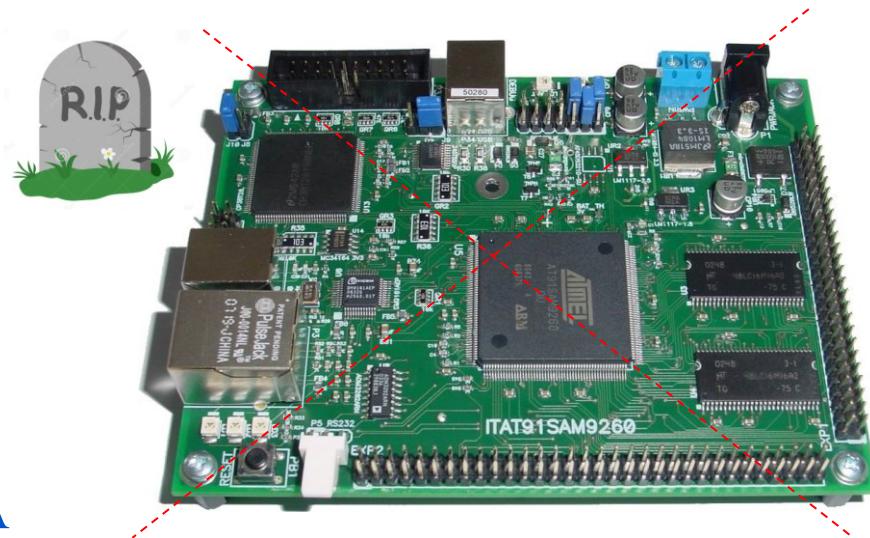
# Računalniška arhitektura RA

Računalnik STM32H750-DK



- Računalnik FRI-SMS

- Mikrokontroler AT91SAM9260 iz družine mikrokontrolnikov ARM9



## Ekipa RA

### Asistentke, asistenti



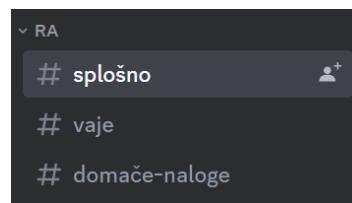
Žiga Pušnik  
[ziga.pusnik@fri....](mailto:ziga.pusnik@fri....)



Romanela Lajić  
[romanela.lajic@fri....](mailto:romanela.lajic@fri....)



Mira Trebar  
[mira.trebar@fri...](mailto:mira.trebar@fri...)



<https://discord.gg/nmjQU7me7>



Robert Rozman  
[rozman@fri.uni-lj.si](mailto:rozman@fri.uni-lj.si)

### Tutorji

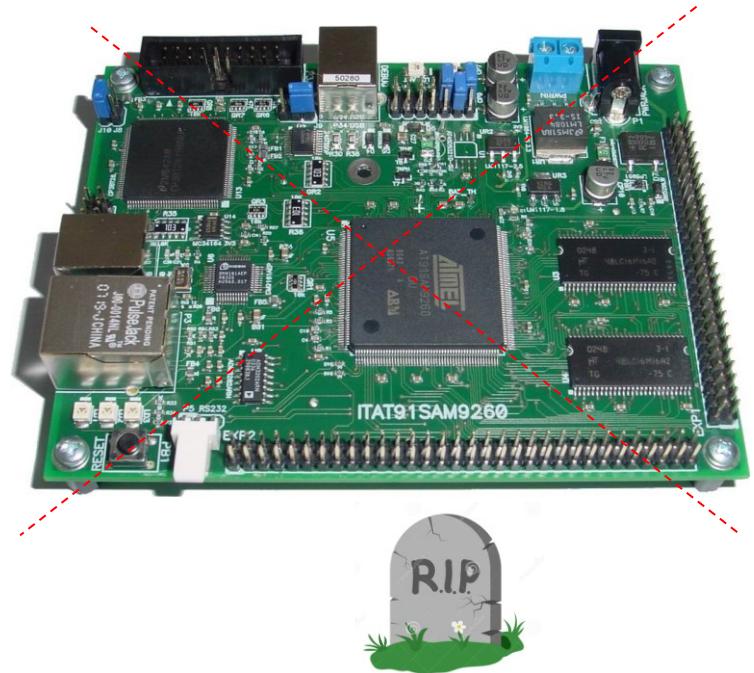
# Računalniška arhitektura RA

Računalnik STM32H750-DK



- Računalnik FRI-SMS

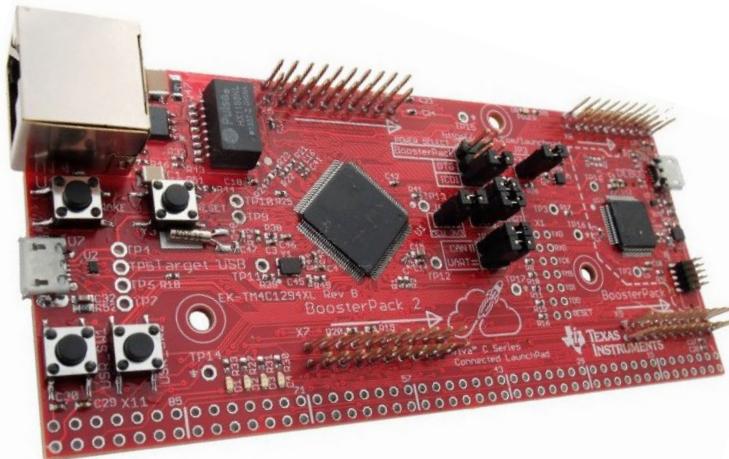
- Mikrokontroler AT91SAM9260 iz družine mikrokontrolnikov ARM9



LAB 1.1 Splošne informacije

# Laboratorijske vaje RA

- Spoznati osnove računalniške arhitekture s praktičnega vidika
- Razumeti delovanje računalnika (ARM) s programiranjem v zbirnem jeziku
- Podrobnejši vpogled:
  - v **delovanje** računalnika
  - v **izvajanje programov** na računalniku
- Vsebinske nadgradnje -> predmeti Organizacija računalnikov, Vhodno izhodne naprave in ostali



# Vsebina vaj



- Potrebne osnove s predavanj (npr. pomnilniški naslov, vsebina, ...)
- **Jedro: Programiranje v zbirnem jeziku ARM**
- Oblika:
  - Sprotne vaje (2. praktični vaji) + domača naloga
- Tri preverjanja (november, december, januar)
- Priprava na izpit (avditorne naloge)
- Predmetni seminar po dogovoru z asistentom
- Video gradiva (ponovitev, utrditev):
  - [Računalniška arhitektura \(RA\) \(sharepoint.com\)](https://sharepoint.com)

# Ocenjevanje

Vaje prispevajo **50% h končni oceni** in morajo biti opravljene naslednje obveznosti:

- Uspešno **opraviti sprotne naloge in biti prisoten** na laboratorijskih vajah
- Uspešno **oddati in zagovarjati** domačo nalogu,
- Tri preverjanja ( $80 + 100 + 120$  točk)
  - skupaj potrebno **zbrati vsaj 150 točk (50%)**
  - ni omejitev na posameznih preverjanjih
- Ocena vaj velja le v tekočem študijskem letu. Kdor v istem letu ne opravi predmeta v celoti, mora prihodnje leto ponovno opraviti vaje.

# Spletni simulator cpulator

- <https://cpulator.01xz.net/?sys=arm>
- začetni projekt RA:
  - <https://cpulator.01xz.net/?sys=arm&loadasm=share/sg8LlNt.s>

The screenshot shows the cpulator debugger interface with the following sections:

- Toolbar:** Stopped, Step Into (F2), Step Over (Ctrl-F2), Step Out (Shift-F2), Continue (F3), Stop (F4), Restart (Ctrl-R), Reload (Ctrl-Shift-L), File ▾, Help ▾.
- Registers:** Shows CPU registers r0 to pc. The pc register (00000048) is highlighted with a red box.
- Editor (Ctrl-E):** Assembly code input field. Language: ARMv7. File: untitled.s. The code is:

```
1 .text
2 .org 0x20
3 @spremenljivke
4 stev1: .word 0x40
5 stev2: .word 0x10
6 rez: .space 4
7
8 .align
9 .global _start
10 _start:
11
12 @program
13 adr r0, stev1
14 ldr r1, [r0]
15
16 adr r0, stev2
17 ldr r2, [r0]
18
19 add r3, r2, r1
20
21 adr r0, rez
22 str r3, [r0]
23
24 end: b end
```
- Memory (Ctrl-M):** Memory dump table. Address column (Address) and Content column (Memory contents and ASCII). Addresses range from 00000040 to 000001e0. Content is mostly 'aa' (hex 41).
- Messages:** Compiling... Code and data loaded from ELF executable into memory. Total size is 80 bytes.
- Build Log:** Assemble: arm-altera-eabi-as -mfloating-point=soft -march=armv7-a -mcpu=cortex-a9 -mfpu=neon-fp16 --gdwarf2 -o work/asmhSiYoH.s.o work/asmhSiYoH.s  
Link: arm-altera-eabi-ld --script build\_arm.ld -e \_start -u \_start -o work/asmhSiYoH.elf work/asmhSiYoH.s  
Compile succeeded.

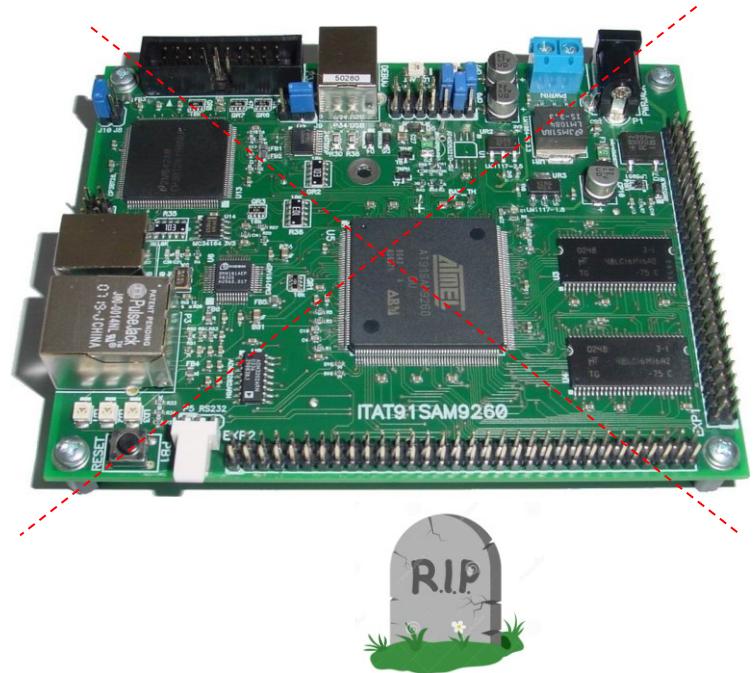
# Računalniška arhitektura RA

Računalnik STM32H750-DK



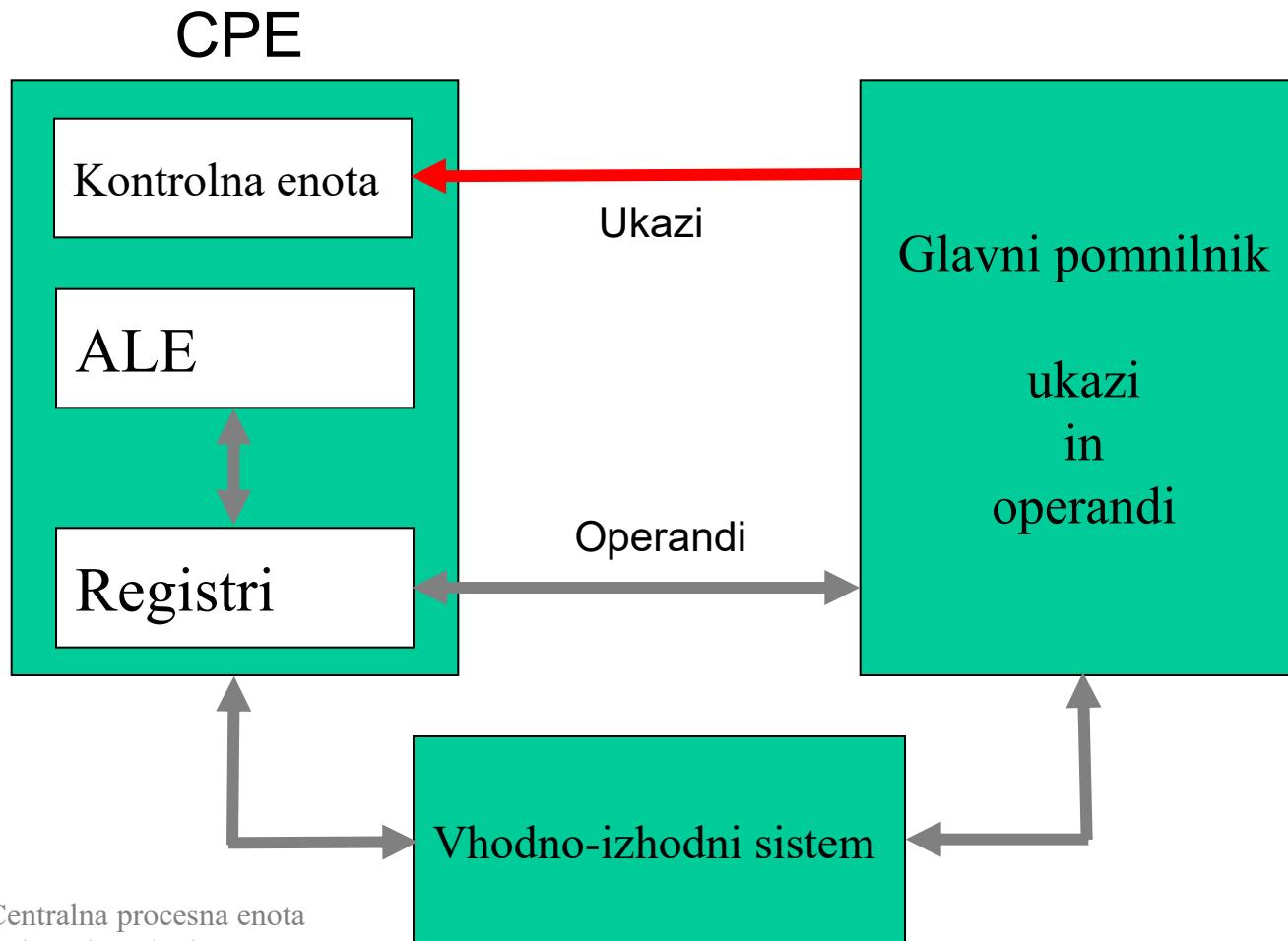
- Računalnik FRI-SMS

- Mikrokontroler AT91SAM9260 iz družine mikrokontrolnikov ARM9

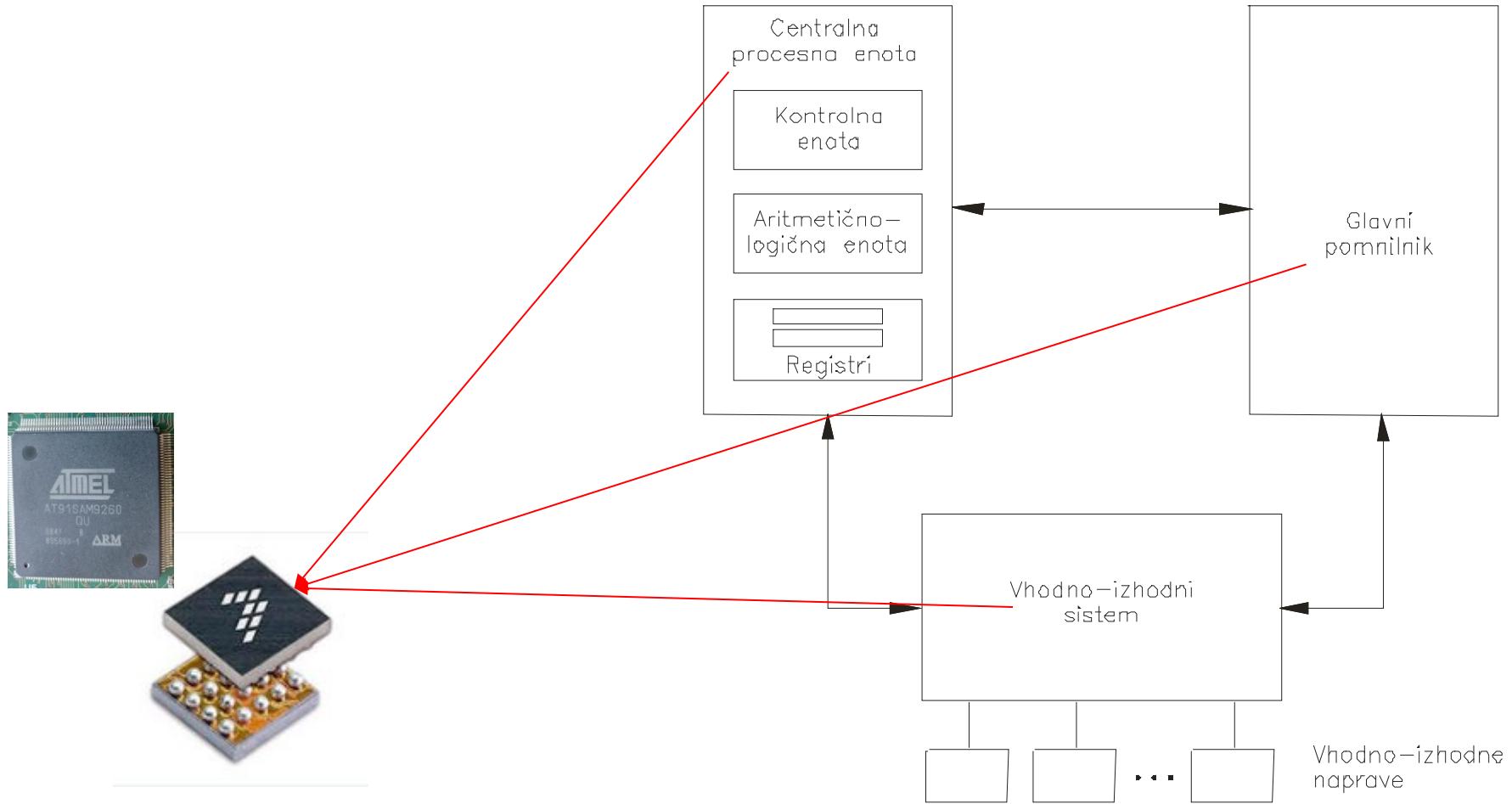


LAB 1.2 Von Neumannov model (VN)

## Von Neumannov računalniški model

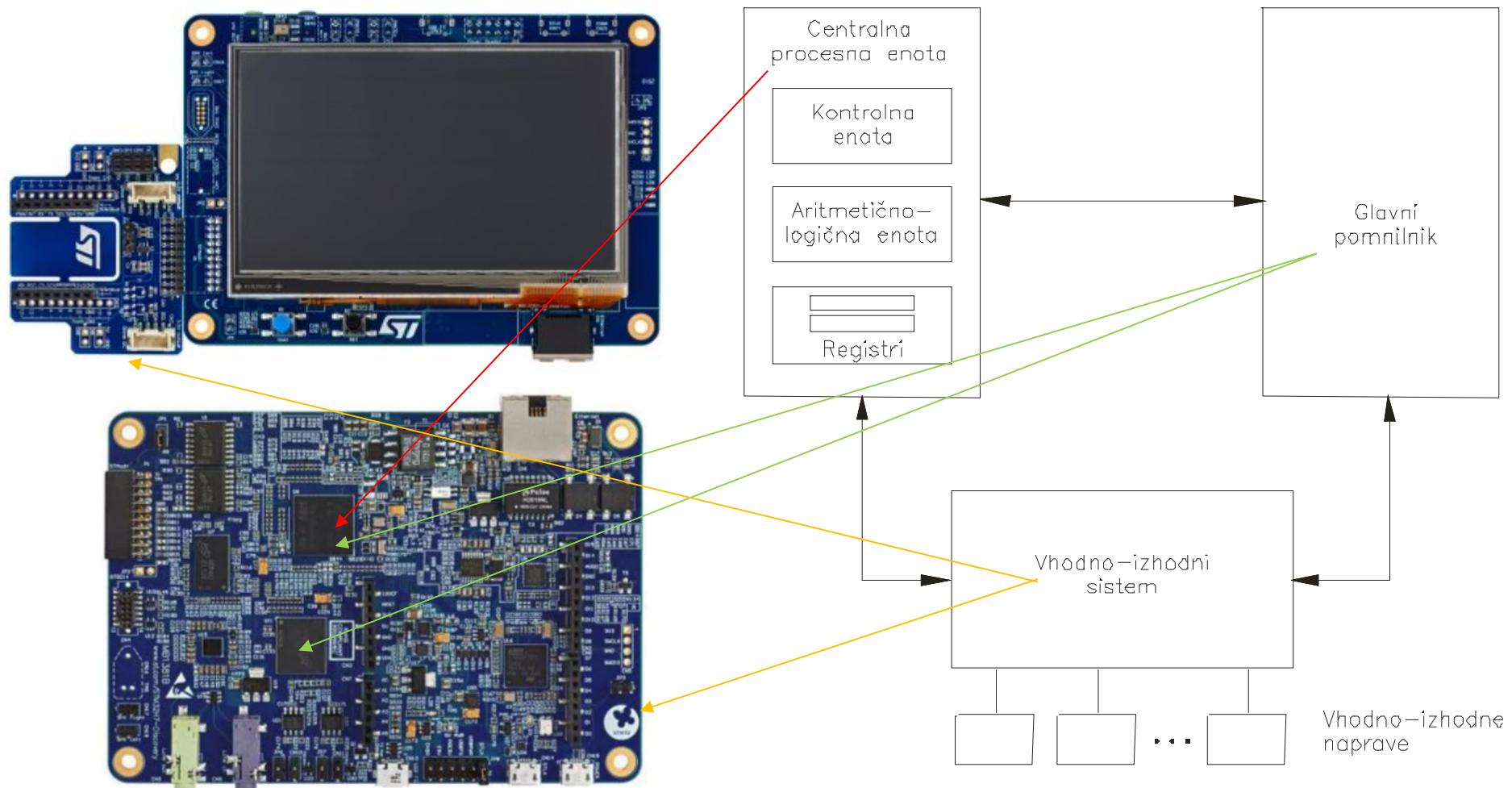


# Osnovni model računalnika



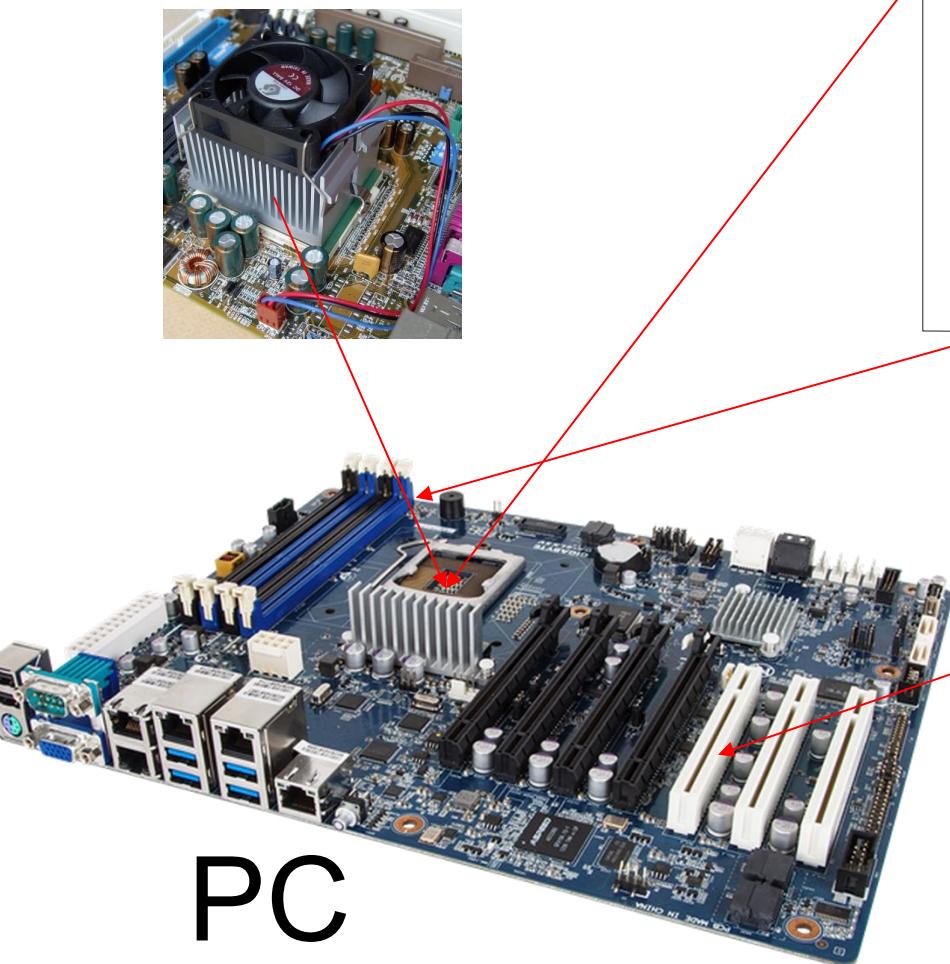
## Mikrokrmlniki

# Osnovni model računalnika

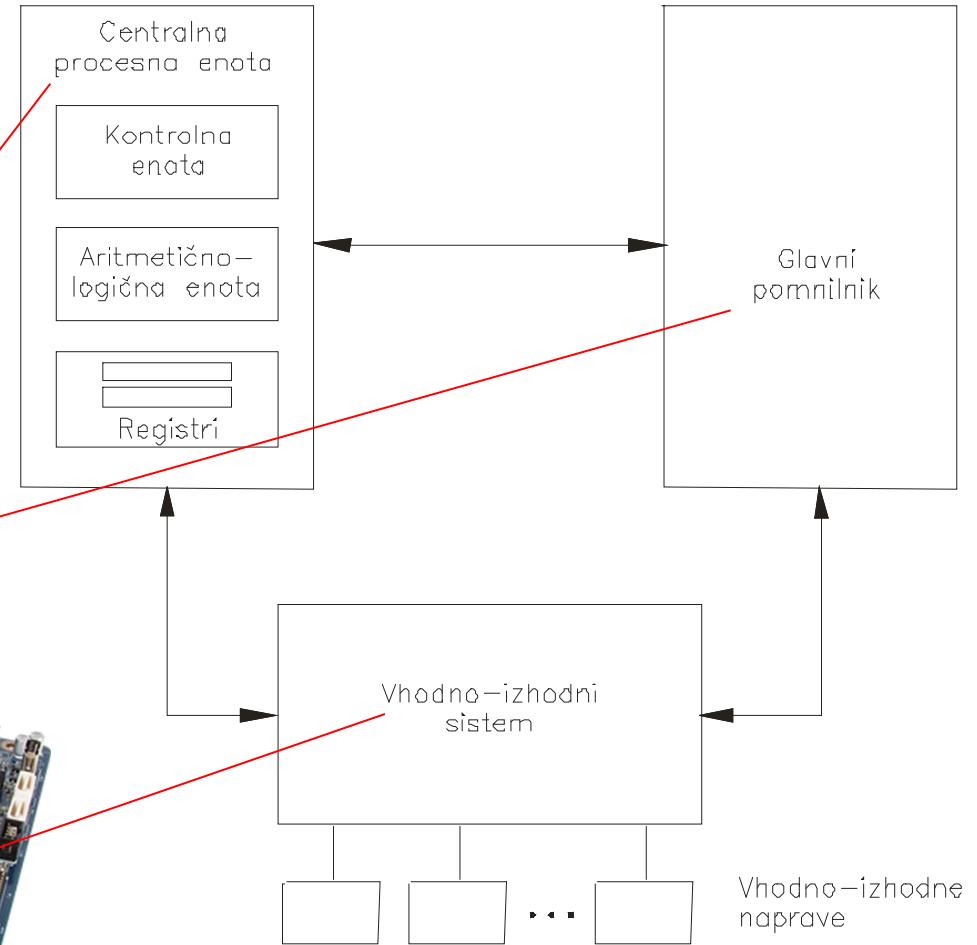


STM32H750-DK

# Osnovni model računalnika



PC



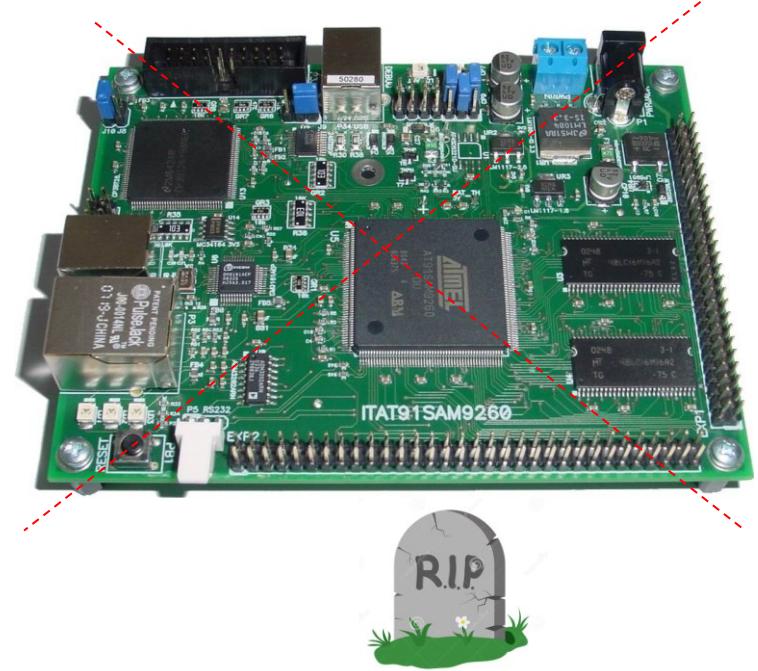
# Računalniška arhitektura RA

Računalnik STM32H750-DK



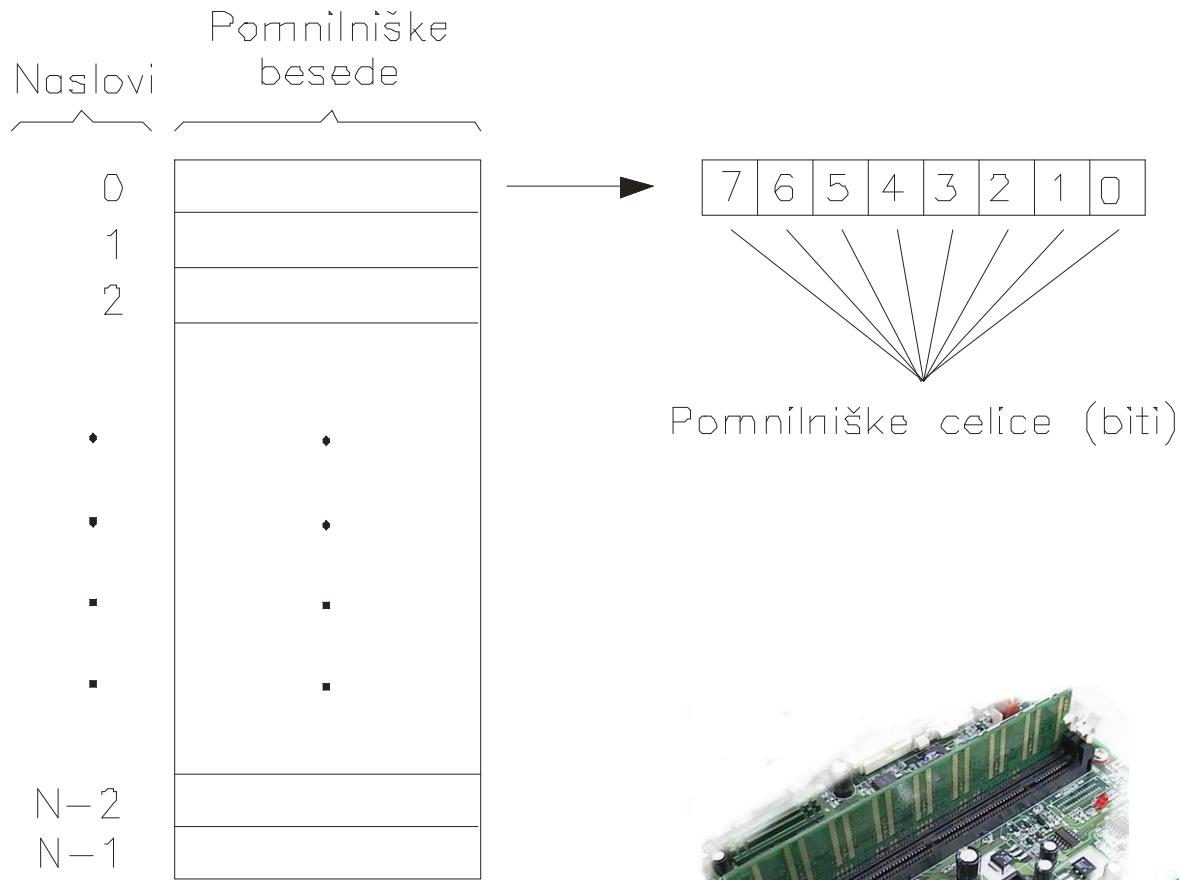
- Računalnik FRI-SMS

- Mikrokontroler AT91SAM9260 iz družine mikrokontrolnikov ARM9



LAB 1.3 Pomnilnik

# Kaj je pomnilnik ?

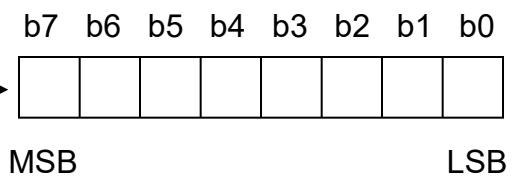


### Pomnilniški naslov

Dvojiško (dolžina 16 bitov) Desetiško

0000 0000 0000 0000	0
0000 0000 0000 0001	1
0000 0000 0000 0010	2
0000 0000 0000 0011	3
0000 0000 0000 0100	4
0000 0000 0000 0101	5
.....	.....

Pomnilniške besede



8-bitna pomnilniška beseda

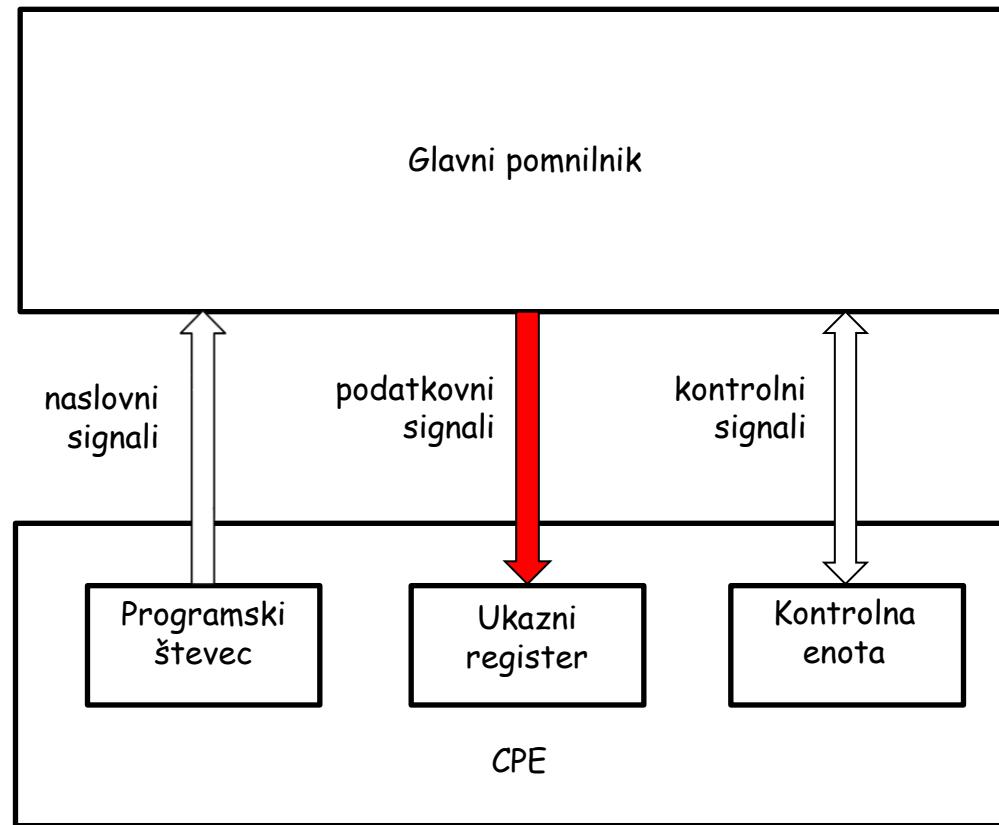
## Pomnilniški naslov

Dvojiško (dolžina 16 bitov) Šestnajst. 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	FFFFB	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

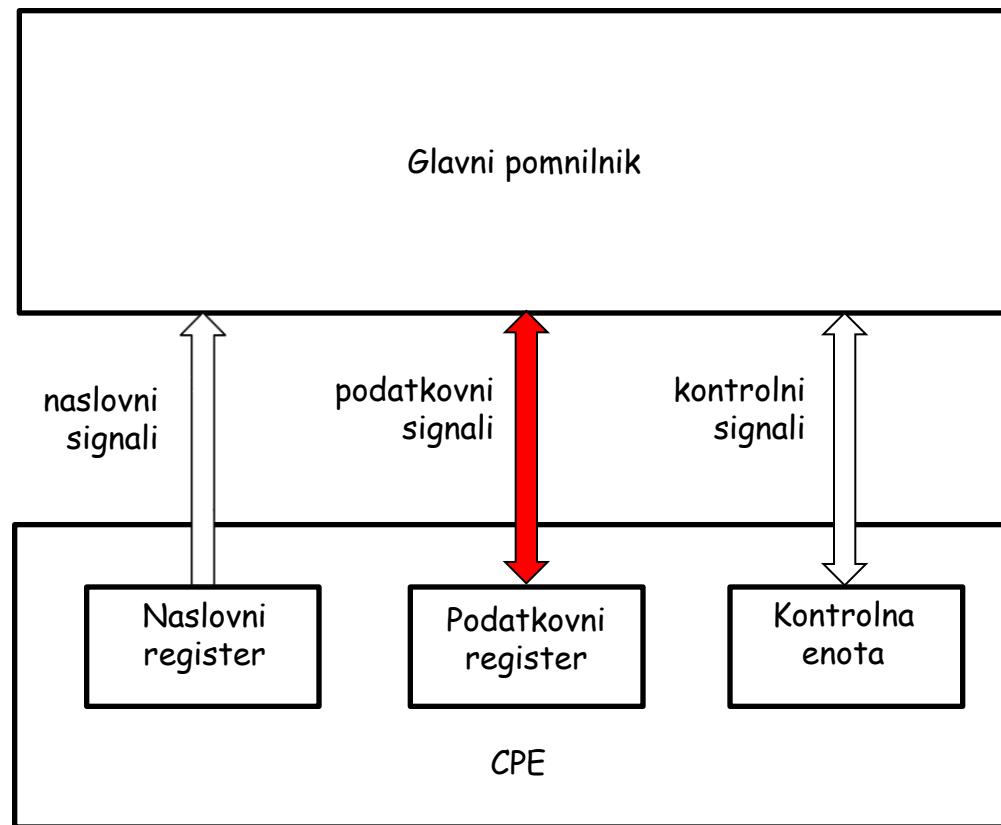
# Kako CPE dostopa do glavnega pomnilnika?

Primer za ukaze:



# Kako CPE dostopa do glavnega pomnilnika?

Primera za operand:



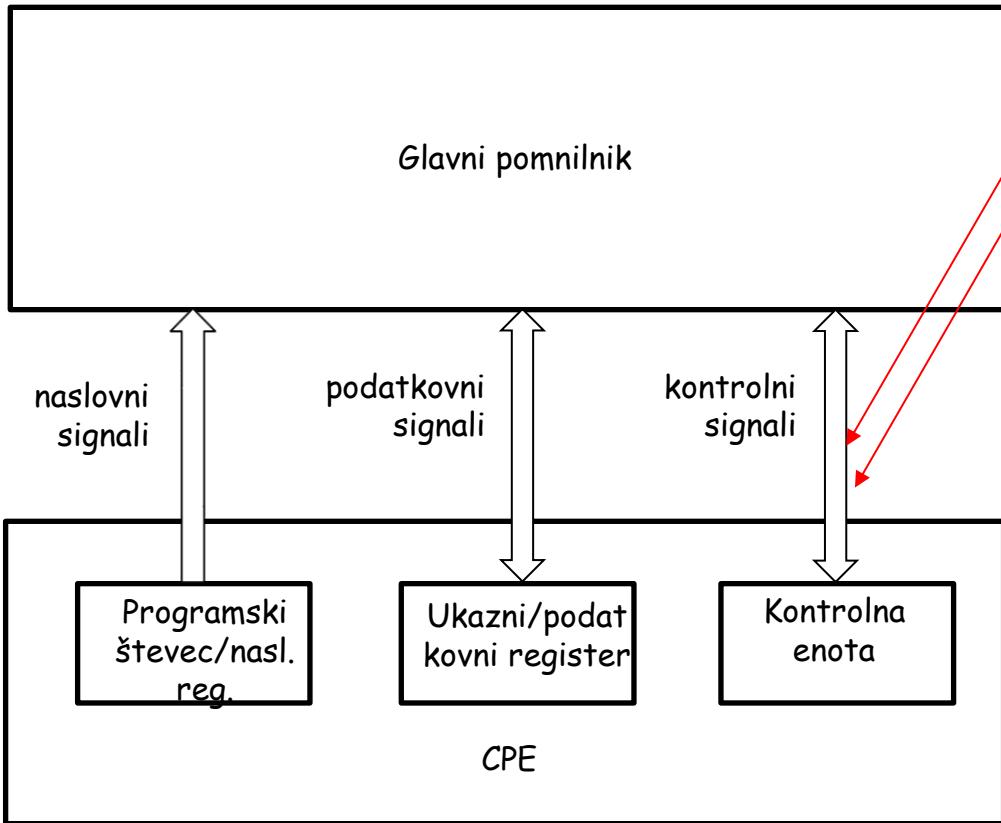
# Povezava CPE <-> glavni pomnilnik

Vodilo = skupina povezav

(naslovno, podatkovno,  
kontrolno, ...)

Linija = povezava

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



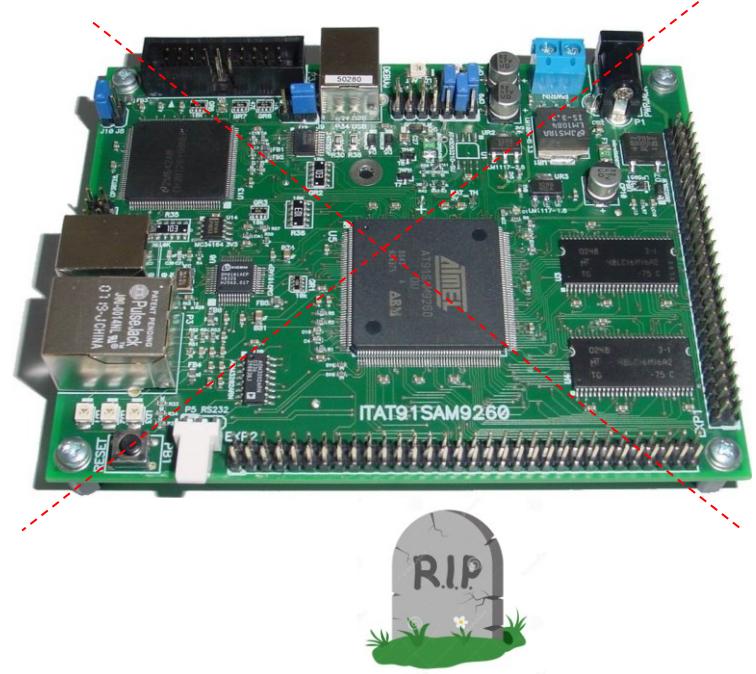
# Računalniška arhitektura RA

Računalnik STM32H750-DK



- Računalnik FRI-SMS

- Mikrokontroler AT91SAM9260 iz družine mikrokontrolnikov ARM9



LAB 1.4 Številski sistemi (BIN,HEX) na hitro

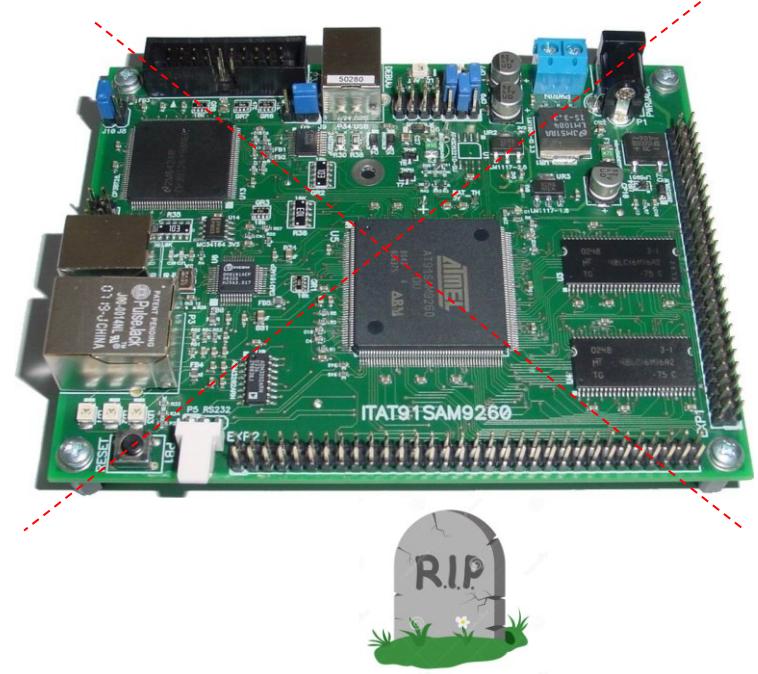
# Računalniška arhitektura RA

Računalnik STM32H750-DK



■ Računalnik FRI-SMS

□ Mikrokontroler AT91SAM9260 iz družine mikrokontrolnikov ARM9



LAB 1.5 Pravilo tankega/debelega konca na hitro

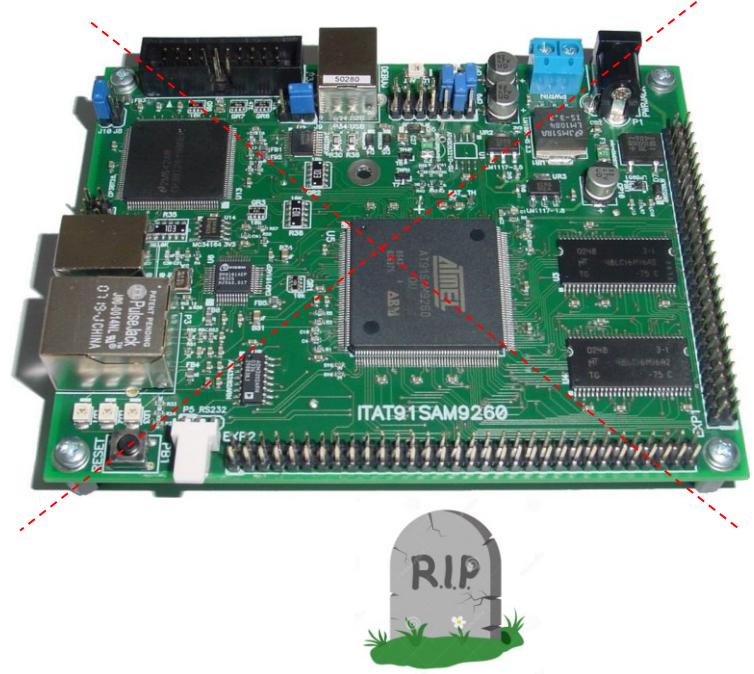
# Računalniška arhitektura RA

Računalnik STM32H750-DK



■ Računalnik FRI-SMS

□ Mikrokontroler AT91SAM9260 iz družine mikrokontrolnikov ARM9



LAB 1.6 Seštevanje – človek, python, zbirnik

# Človek (zgled: $64 + 16 = 80$ )

$$64 + 16 = ?$$

A handwritten addition problem is shown. The numbers 64 and 16 are written vertically, with a plus sign between them. A question mark is placed after the equals sign. To the right of the equation, the sum 80 is written above a horizontal line, with a plus sign and the number 16 written above the line. A red curved arrow points from the question mark to the sum 80.

$$\begin{array}{r} 64 \\ + 16 \\ \hline 80 \end{array}$$

# Python (zgled: REZ = STEV1 + STEV2)

Seštevanje spremenljivk v Pythonu.

<http://goo.gl/YXQ5qN>

Python 2.7

```
1 STEV1=0x40
2 STEV2=0x10
3 REZ = STEV1 + STEV2
→ 4 print (" STEV1 = " + hex(STEV1) + "\n+STEV2 = " + hex(STE
```

Frames

Global frame

STEV1 64

STEV2 16

REZ 80

Objects

Print output (drag lower right corner to resize)

STEV1 = 0x40

+STEV2 = 0x10

-----

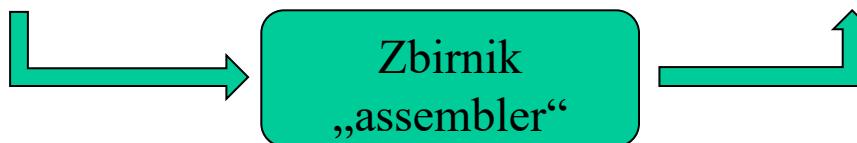
REZ = 0x50

# Zbirni jezik (zgled: rez=stev1+stev2)

**Seštevanje spremenljivk v zbirniku ARM. Prenesite pripravljen projekt na e-učilnici.**

Vrednosti spremenljivk so shranjene v pomnilniku. Operacije realiziramo s programom z naslednjimi ukazi:

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

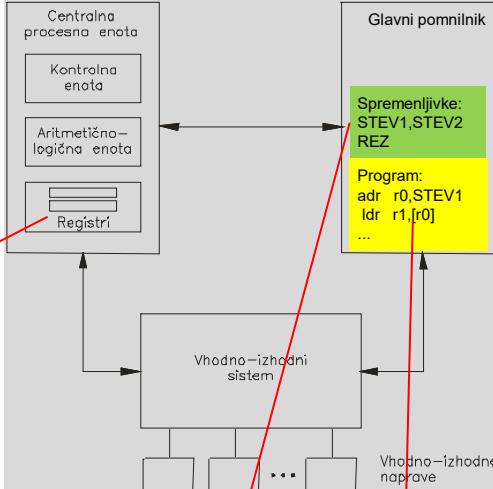


Ukaze izvajajte po korakih in opazujte vrednosti registrov in vrednosti spremenljivk v pomnilniku.

# Praktično delo: vsota dveh števil

<https://cpulator.01xz.net/?sys=arm&loadasm=share/s8zU3xx.s>

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



Registers

Register	Value
r0	00000000
r1	00000000
r2	00000000
r3	00000000
r4	00000000
r5	00000000
r6	00000000
r7	00000000
r8	00000000
r9	00000000
r10	00000000
r11	00000000
r12	00000000
sp	00000000
lr	00000000
pc	0000002c
cpsr	000001d3 NZCVI SVC
spsr	00000000 NZCVI ?
s0	00000000
s1	00000000
s2	00000000
s3	00000000
s4	00000000
s5	00000000
s6	00000000

Disassembly (Ctrl-D)

```

Go to address, label, or register: 00000000

Address      Opcode      Disassembly
00000020      00000010    STEV1:
                           andeq   r0, r0, r0,
00000024      00000040    STEV2:
                           andeq   r0, r0, r0,
00000028      00000000    REZ:
                           andeq   r0, r0, r0
9             .align
11            .global _start
12            _start:
14            adr r0,STEV1
_start:
0000002c      e24f0014    adr r0, 0x20 (e
00000030      e5901000    ldr r1, [r0]
15            00000000
17            adr r0,STEV2
                           adr r0, 0x24 (e
18            ldr r2, [r0]
20            add r3, r1, r2
22            adr r0,REZ
                           adr r0, 0x28 (e
23            str r3, [r0]
26            end: b er
end:
b            0x48 (0x48: enc

```

Memory (Ctrl-M)

Address	Memory contents and ASCII
00000000	00000000 00000000 00000000 00000000
00000010	00000000 00000000 00000000 00000000
00000020	00000010 00000040 00000000 e24f0014
00000030	e5901000 e24f0018 e5902000 e0813002
00000040	e24f0020 e5803000 eaffffff 00000000
00000050	aaaaaaaa aaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaa
00000060	aaaaaaaa aaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaa
00000070	aaaaaaaa aaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaa
00000080	aaaaaaaa aaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaa
00000090	aaaaaaaa aaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaa
000000a0	aaaaaaaa aaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaa
000000b0	aaaaaaaa aaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaa
000000c0	aaaaaaaa aaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaa
000000d0	aaaaaaaa aaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaa
000000e0	aaaaaaaa aaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaa
000000f0	aaaaaaaa aaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaa
00000100	aaaaaaaa aaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaa
00000110	aaaaaaaa aaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaa
00000120	aaaaaaaa aaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaa
00000130	aaaaaaaa aaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaa
00000140	aaaaaaaa aaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaa
00000150	aaaaaaaa aaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaa
00000160	aaaaaaaa aaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaa

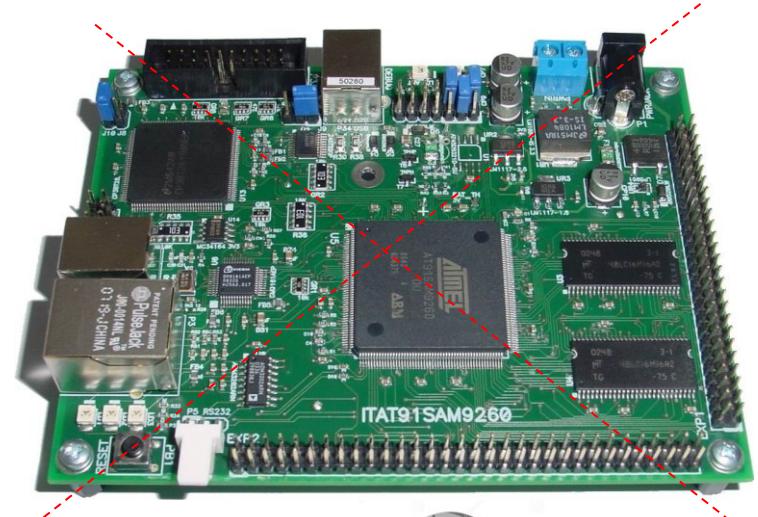
# Računalniška arhitektura RA

Računalnik STM32H750-DK



■ Računalnik FRI-SMS

□ Mikrokontroler AT91SAM9260 iz družine mikrokontrolnikov ARM9



LAB 1.7 Predloge za zapiske

# Python (zgled: REZ = STEV1 + STEV2)

Frames

Objects

Global frame

STEV1 64

STEV2 16

REZ 80

Python 2.7

```
1 STEV1=0x40
2 STEV2=0x10
3 REZ = STEV1 + STEV2
→ 4 print (" STEV1 = " + hex(STEV1) + "\n+STEV2 = " + hex(STE
```

<http://goo.gl/YXQ5qN>

## Zgled: izvedba programa za seštevanje dveh števil

CPE

KE - Kontrolna enota

ALE

REGISTRI

R0

R1

R2

R3

...

Pomnilnik

Naslov Pomnilniške besede

0x00 = 0

0x01 = 1

0x02 = 2

...

0x20 = 32

STEV1

0x24 = 36

STEV2

0x28 = 40

REZ

0x2C = 44

1. ukaz

ADR R0,STEV1

Oznaka  
Vsebina

## UKAZI

	<b>Strojni jezik</b>	<b>Zbirni jezik</b>	<b>Opis ukaza</b>	<b>Komentar</b>
1.	0xE24F0014	adr r0, stev1	$R0 \leftarrow \text{nasl. stev1}$	
2.	0xE5901000	ldr r1, [r0]	$R1 \leftarrow M[R0]$	
3.	0xE24F0018	adr r0, stev2	$R0 \leftarrow \text{nasl. stev2}$	
4.	0xE5902000	ldr r2, [r0]	$R2 \leftarrow M[R0]$	
5.	0xE0823001	add r3, r2, r1	$R3 \leftarrow R1 + R2$	
6.	0xE24F0020	adr r0, rez	$R0 \leftarrow \text{nasl. rez}$	
7.	0xE5803000	str r3, [r0]	$M[R0] \leftarrow R3$	

# Pravilo tankega in debelega konca / Big vs. Little Endian

---

MSB      LSB

0 x AA BB CC DD

Debeli konec  
Big Endian



Tanki konec  
Little Endian

