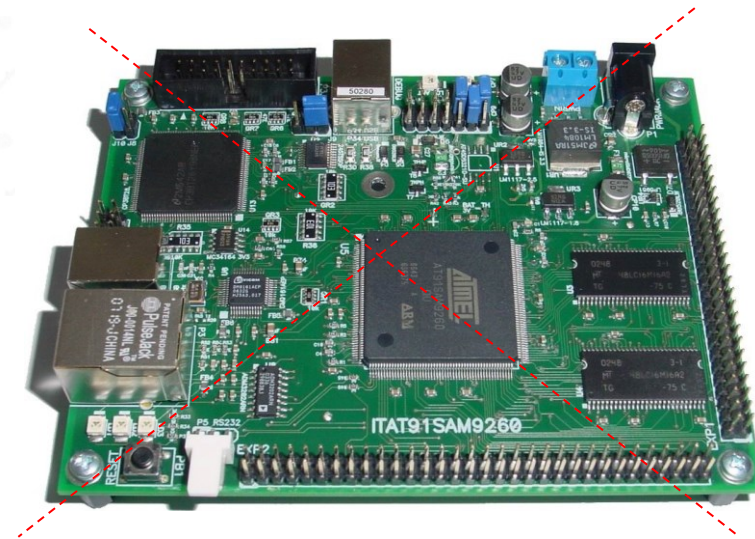


Computer architecture CA

Computer STM32H750-DK



- Računalnik FRI-SMS
 - Mikrokmlinik AT91SAM9260 iz družine mikrokmilnikov ARM9



Team CA

Instructors



Žiga Pušnik
ziga.pusnik@fri...

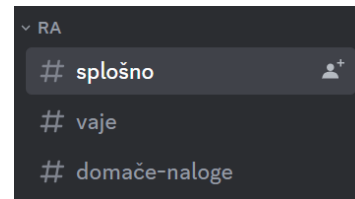


Romanela Lajić
romanela.lajic@fri...



Mira Trebar
mira.trebar@fri...

Tutors



<https://discord.gg/nmzjQU7me7>



Robert Rozman
rozman@fri.uni-lj.si

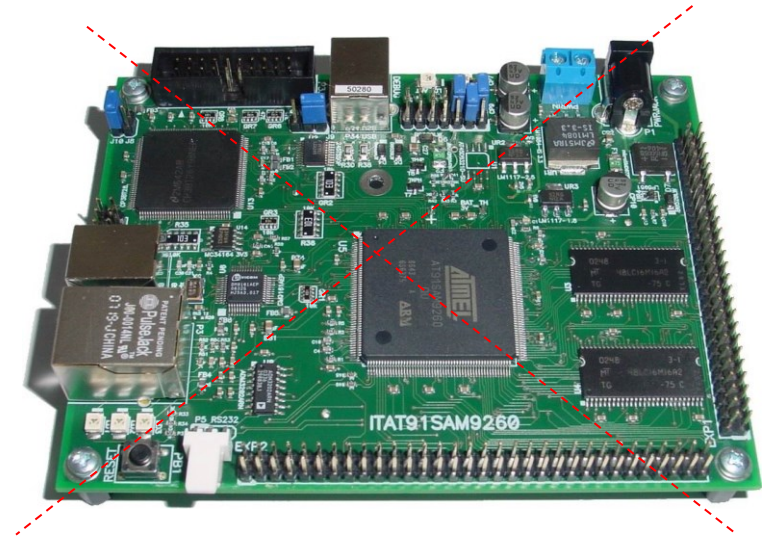


Computer architecture CA

Computer STM32H750-DK



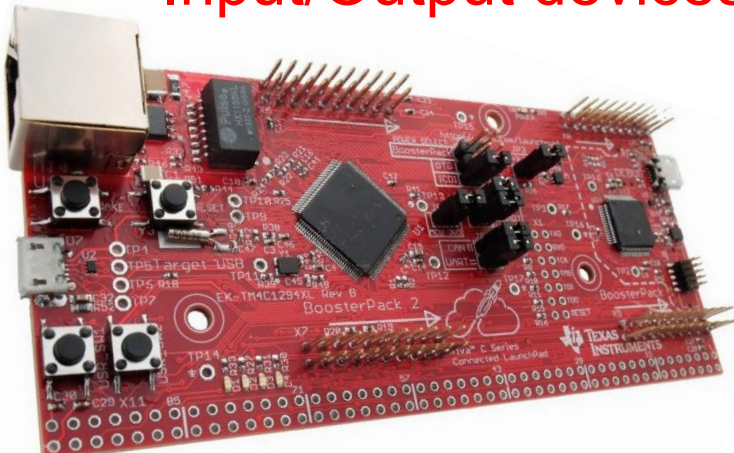
- Računalnik FRI-SMS
 - Mikrokmlinik AT91SAM9260 iz družine mikrokmilnikov ARM9



LAB 1.1 General information

Laboratory exercises

- Learning the foundations of computer architecture from a practical view
- Understanding “How the computer works” by programming in ARM assembly language
- In-depth views:
 - computer operation
 - program execution
- Content upgrades: Computer Organization, Input/Output devices and other elective courses



Content of LAB work



- Basic knowledge needed from lectures (e.g. memory address, memory words, ...)
- **Core: Programming in ARM assembly language**
- Format:
 - lab exercises (2 hands-on exercises) + 1 homework assignment
- 3 intermediate exams (quizzes during lab sessions) - (november, december, january)
- Final exam preparations and exercises

- Alternative way: course seminar for advanced students – talk to instructor

- Video tutorials :
 - [Računalniška arhitektura \(RA\) \(sharepoint.com\)](http://sharepoint.com)

Evaluation – grading

- Lab marks represents **50% of the final mark** for the course. You need to have:
 - successfully evaluated **lab work** (presence, work)
 - successfully evaluated **homework assignment**,
 - three **intermediate evaluation exams** (80 + 100 + 120 points)
 - only condition: gather **at least 150 points (50%)**
 - no additional conditions on results of evaluation exam
- Final lab grade is valid only for the current academic year. You need to repeat lab work in new school year.

Web simulator cpulator

- <https://cpulator.01xz.net/?sys=arm>
- Base project for CA course:
 - <https://cpulator.01xz.net/?sys=arm&loadasm=share/sg8LlNt.s>

The screenshot displays the cpulator web simulator interface. At the top, there are navigation buttons: "Stopped", "Step Into" (F2), "Step Over" (Ctrl-F2), "Step Out" (Shift-F2), "Continue" (F3), "Stop" (F4), "Restart" (Ctrl-R), "Reload" (Ctrl-Shift-L), "File", and "Help".

Registers: A table showing the state of various registers. The PC register is highlighted in red and contains the value 00000048.

Register	Value
r0	00000028
r1	00000040
r2	00000010
r3	00000050
r4	00000000
r5	00000000
r6	00000000
r7	00000000
r8	00000000
r9	00000000
r10	00000000
r11	00000000
r12	00000000
sp	00000000
lr	00000000
pc	00000048

Editor (Ctrl-E): Shows assembly code for "untitled.s" with language set to ARMv7. The code includes directives like .text, .org, .word, .space, .align, .global, and instructions like adr, ldr, add, and str.

```
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): A table showing memory addresses and their contents. The first few bytes contain the values 20 00 4f e2 00 30 80 e5.

Address	Memory contents and ASCII
00000040	20 00 4f e2 00 30 80 e5
00000050	aa aa aa aa aa aa aa aa
00000060	aa aa aa aa aa aa aa aa
00000070	aa aa aa aa aa aa aa aa
00000080	aa aa aa aa aa aa aa aa
00000090	aa aa aa aa aa aa aa aa
000000a0	aa aa aa aa aa aa aa aa
000000b0	aa aa aa aa aa aa aa aa
000000c0	aa aa aa aa aa aa aa aa
000000d0	aa aa aa aa aa aa aa aa
000000e0	aa aa aa aa aa aa aa aa
000000f0	aa aa aa aa aa aa aa aa
00000100	aa aa aa aa aa aa aa aa
00000110	aa aa aa aa aa aa aa aa
00000120	aa aa aa aa aa aa aa aa
00000130	aa aa aa aa aa aa aa aa
00000140	aa aa aa aa aa aa aa aa
00000150	aa aa aa aa aa aa aa aa
00000160	aa aa aa aa aa aa aa aa
00000170	aa aa aa aa aa aa aa aa
00000180	aa aa aa aa aa aa aa aa
00000190	aa aa aa aa aa aa aa aa
000001a0	aa aa aa aa aa aa aa aa
000001b0	aa aa aa aa aa aa aa aa
000001c0	aa aa aa aa aa aa aa aa
000001d0	aa aa aa aa aa aa aa aa
000001e0	aa aa aa aa aa aa aa aa

Messages: Shows the compilation process. The message "Compiling..." is followed by "Code and data loaded from ELF executable into memory. Total size is 80 bytes." and "Compile succeeded."

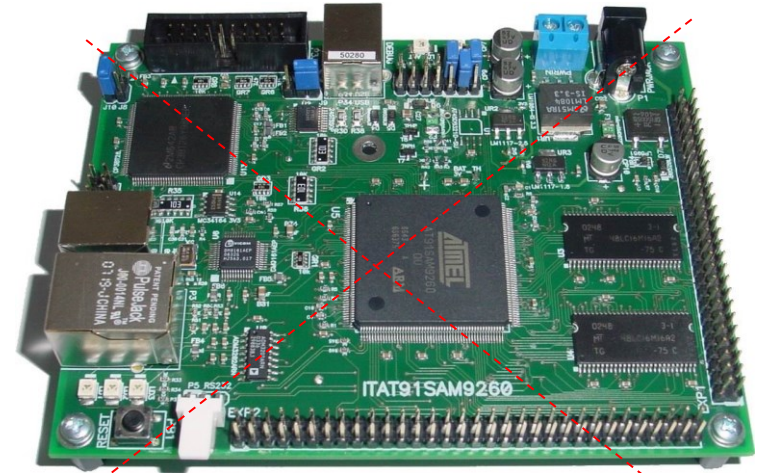
```
Compile succeeded.
```

Computer architecture CA

Computer STM32H750-DK



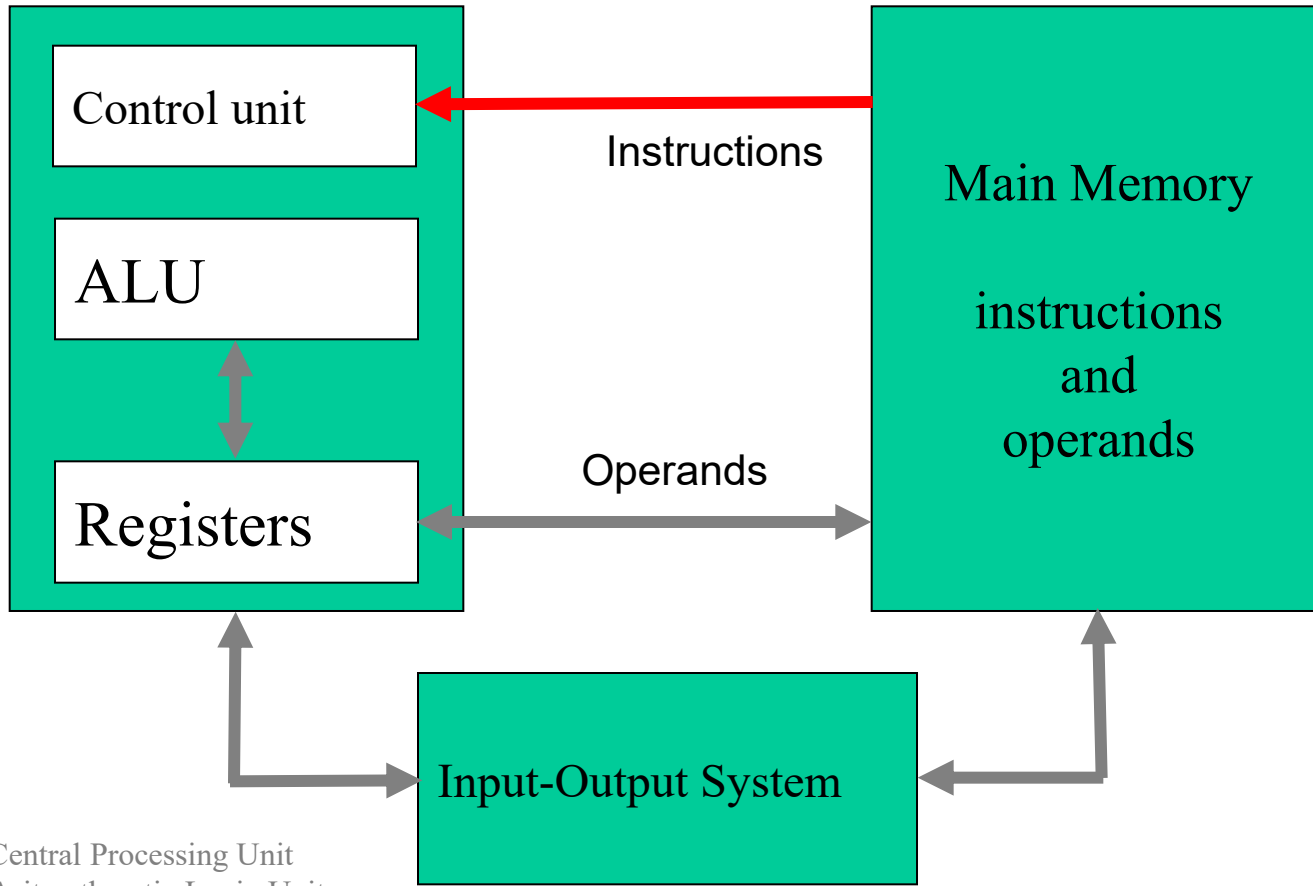
- Računalnik FRI-SMS
 - Mikrokontroler AT91SAM9260 iz družine mikrokontrolerov ARM9



LAB 1.2 Von Neumann model (VN)

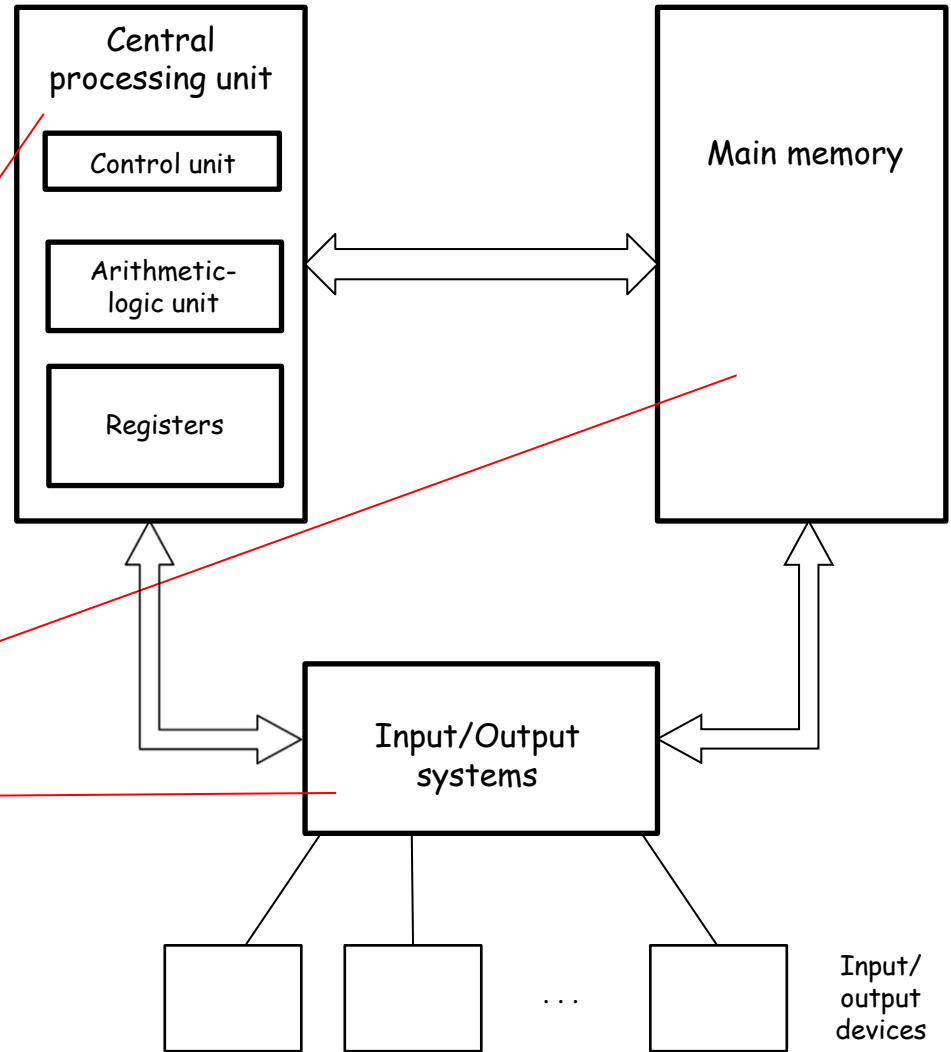
Von Neumann Computer Model

CPU



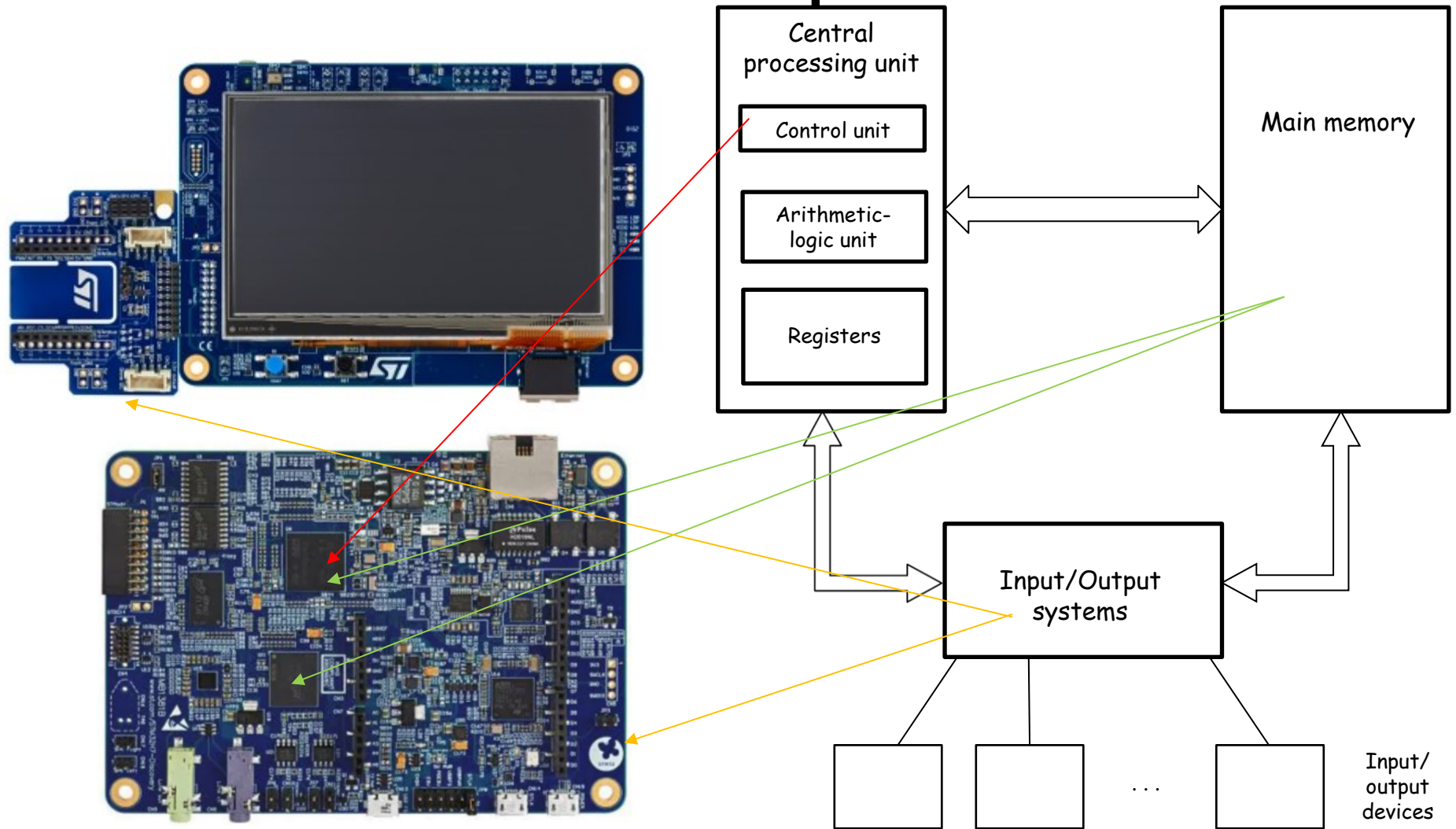
CPU - Central Processing Unit
ALU - Arithmetic Logic Unit

The basic computer model



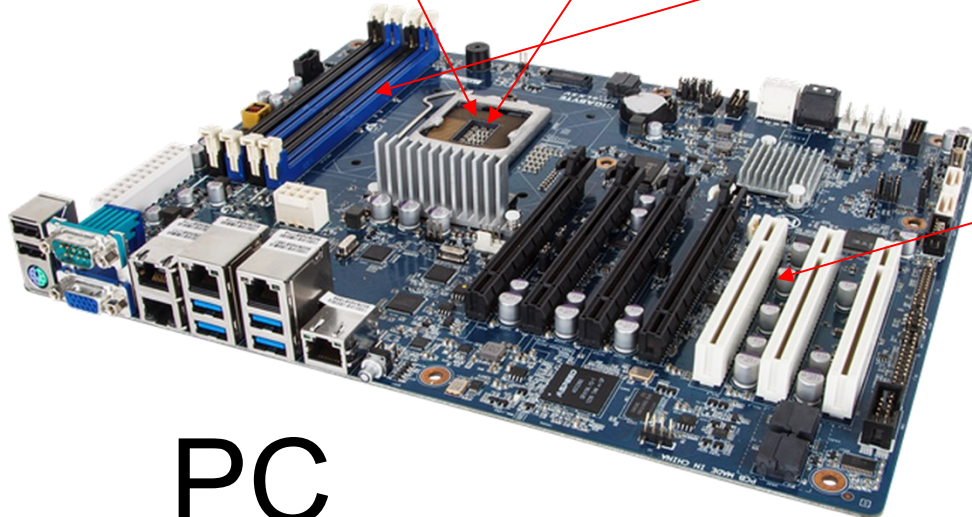
MicroControllers

The basic computer model

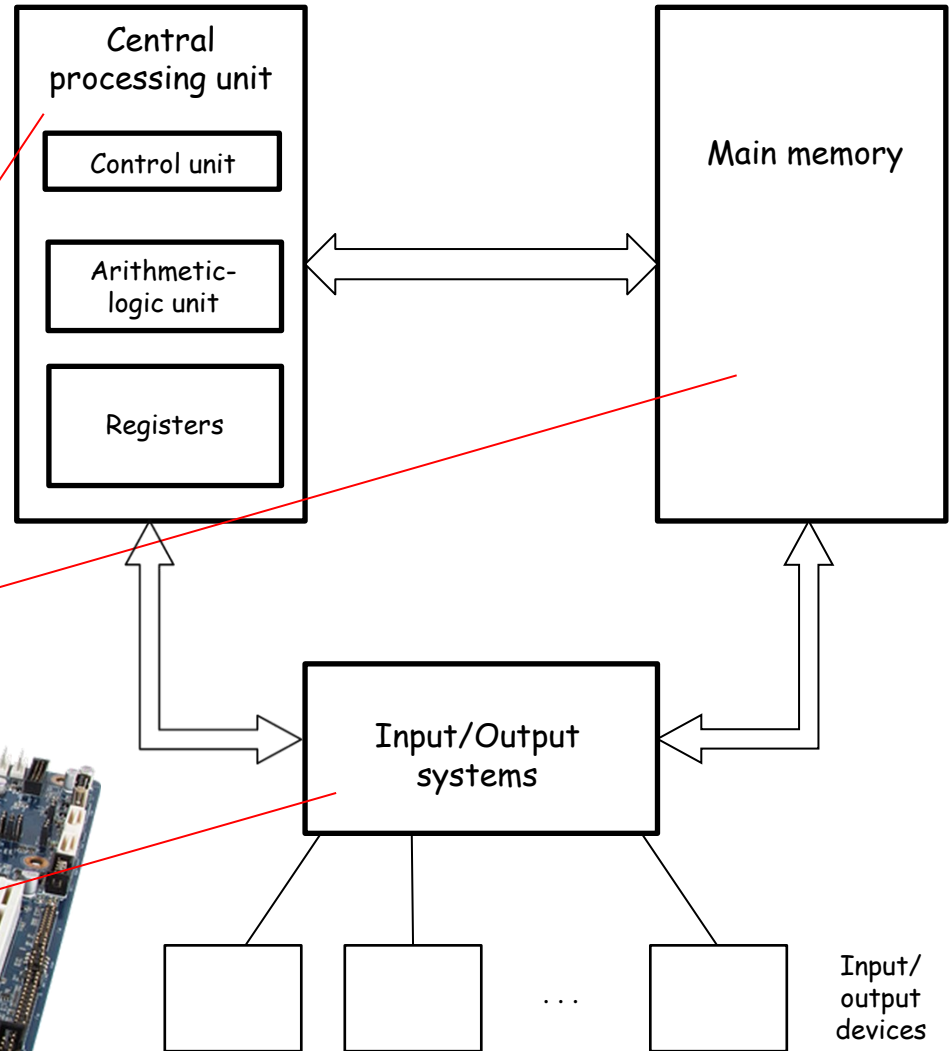


STM32H750-DK

The basic computer model



PC

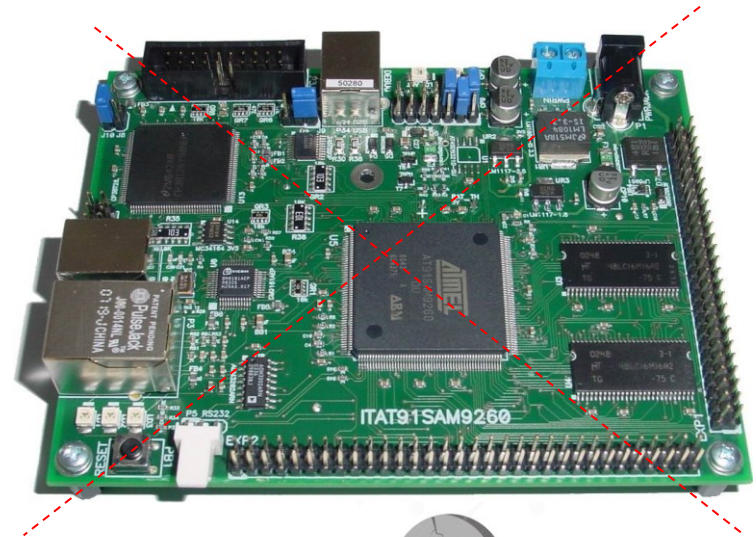


Computer architecture CA

Computer STM32H750-DK

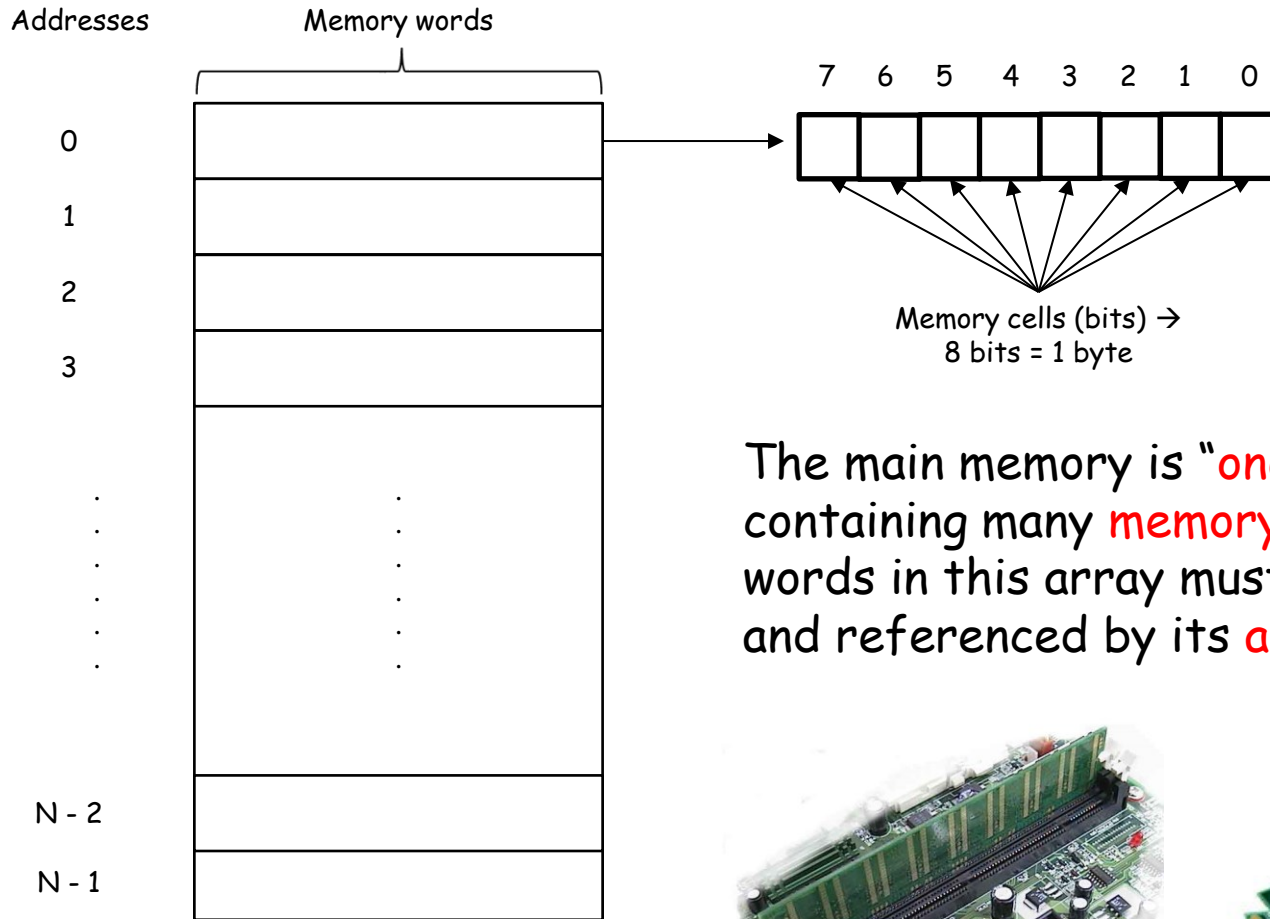


- Računalnik FRI-SMS
 - Mikrokrmilnik AT91SAM9260 iz družine mikrokrmilnikov ARM9



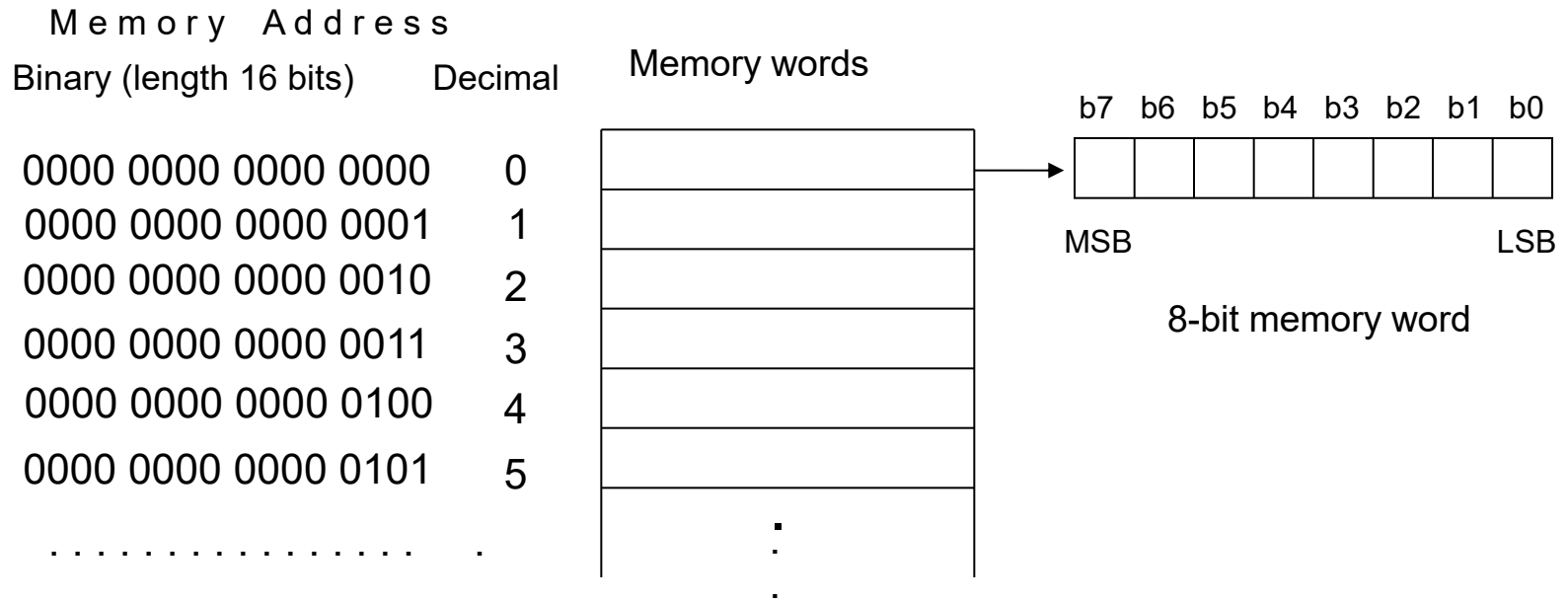
LAB 1.3 Memory

What is memory ?



The main memory is "one dimensional array" containing many **memory words**. Each memory words in this array must be **uniquely** identified and referenced by its **address**!

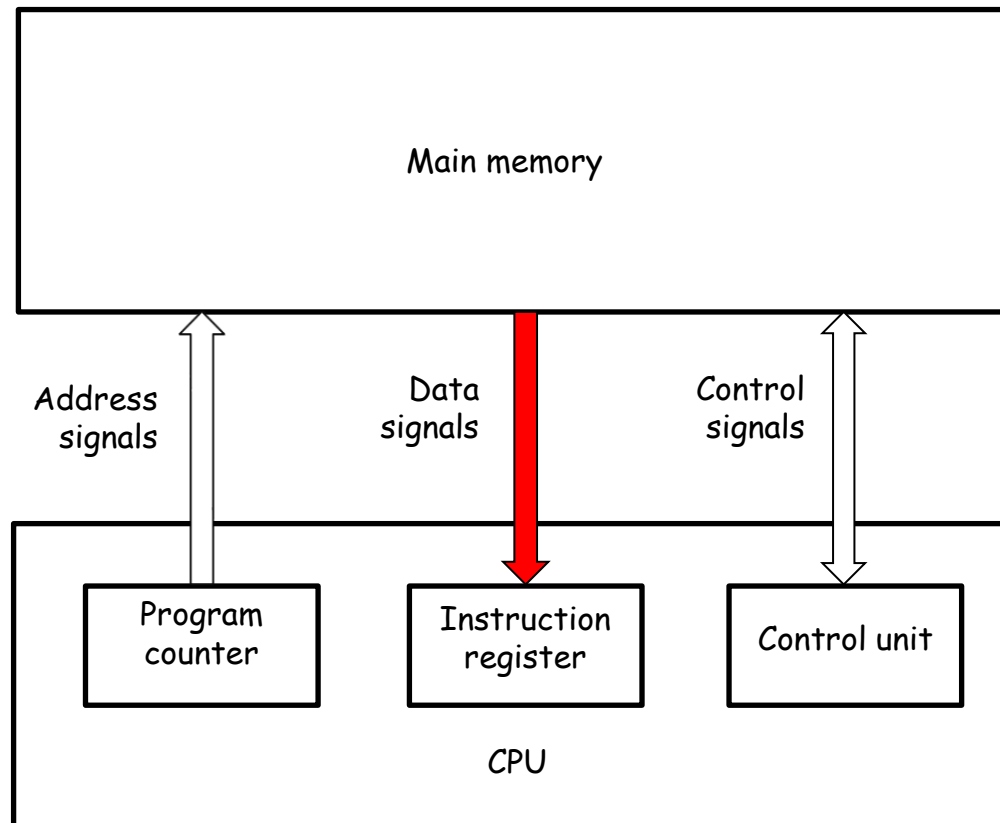




Memory Address			Memory words
Binary (length 16 bits)	Hexadecimal	Decimal	
0000 0000 0000 0000	0000	0	
0000 0000 0000 0001	0001	1	
0000 0000 0000 0010	0002	2	
0000 0000 0000 0011	0003	3	
0000 0000 0000 0100	0004	4	
0000 0000 0000 0101	0005	5	
.....	.		.
.....	.		.
1111 1111 1111 1011	FFFB	65531	
1111 1111 1111 1100	FFFC	65532	
1111 1111 1111 1101	FFFD	65533	
1111 1111 1111 1110	FFFE	65534	
1111 1111 1111 1111	FFFF	65535	

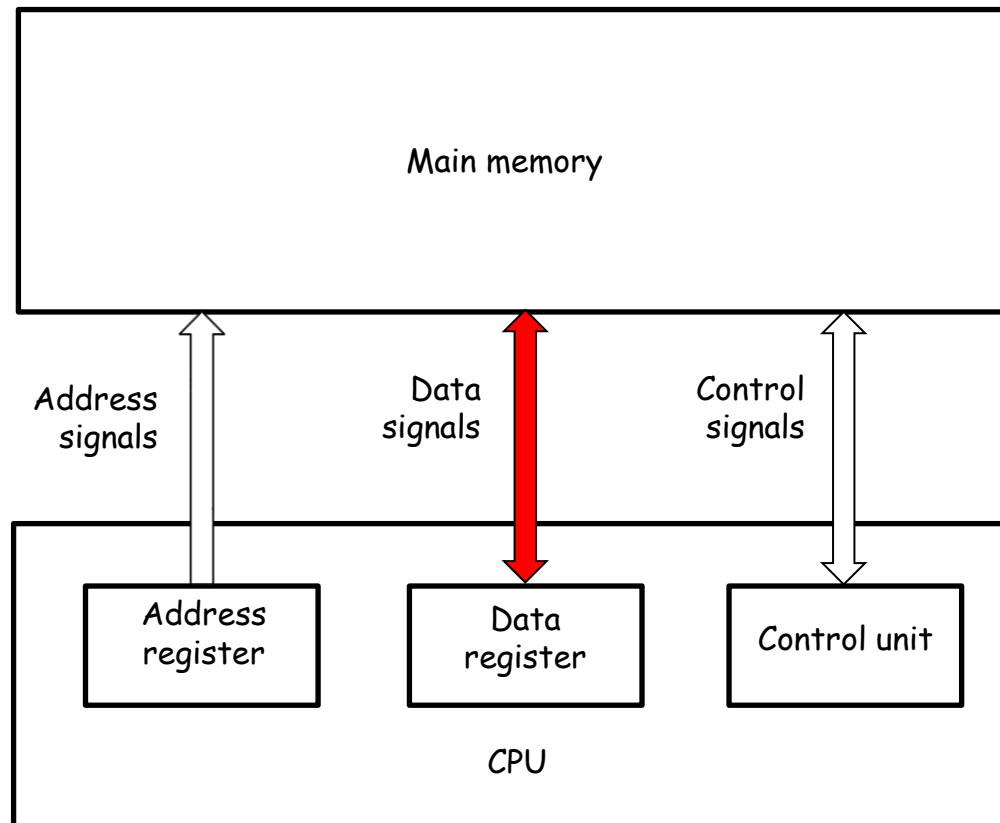
How does CPU access the main memory?

Example for accessing instructions:



How does CPU access the main memory?

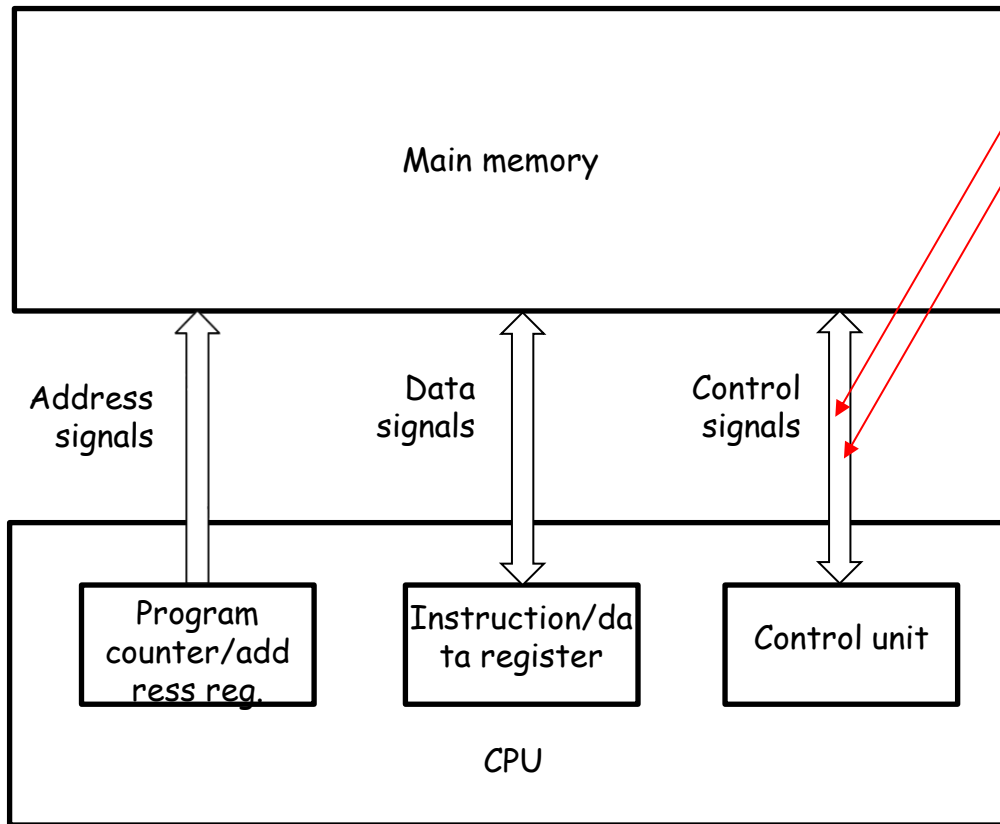
Examples for accessing operands:



Interconnection CPU <-> main memory

Bus = a group of related lines
(Address, Data, Control buses)

Line = physical connection
Signal = content transferred over the line (1bit)

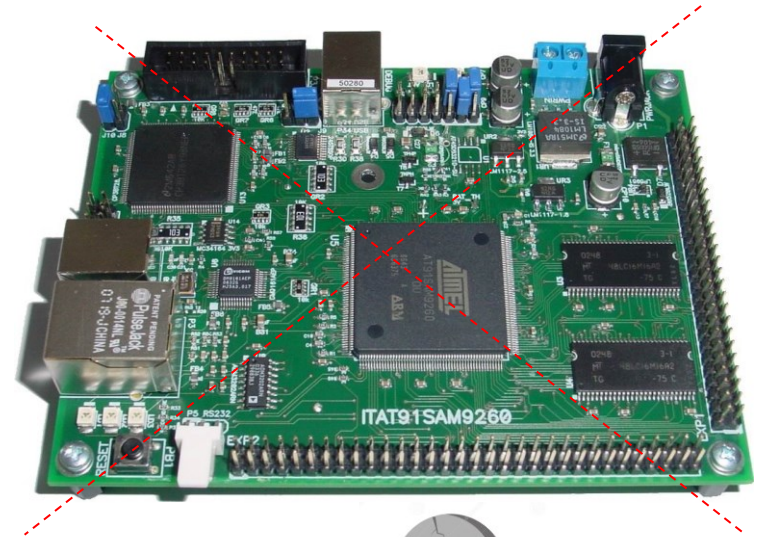


Computer architecture CA

Computer STM32H750-DK



- Računalnik FRI-SMS
 - Mikrokontroler AT91SAM9260 iz družine mikrokontrolerov ARM9



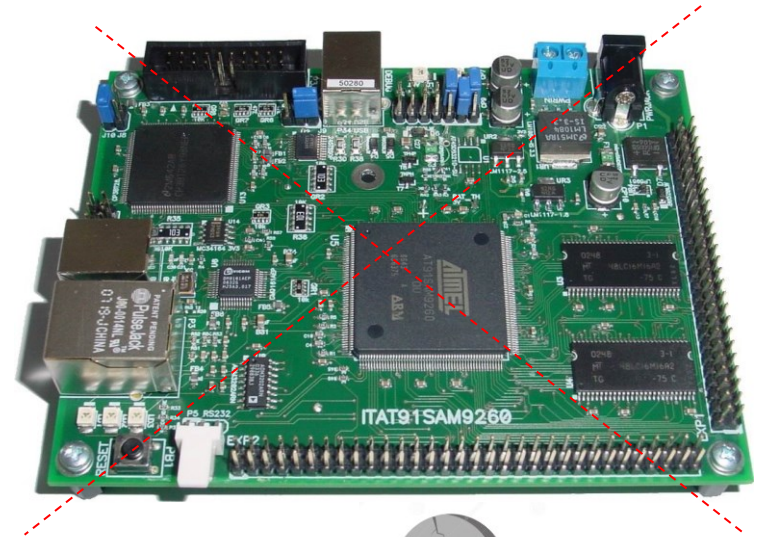
LAB 1.4 Quick intro to numeral systems

Computer architecture CA

Computer STM32H750-DK



- Računalnik FRI-SMS
 - Mikrokontroler AT91SAM9260 iz družine mikrokontrolerov ARM9



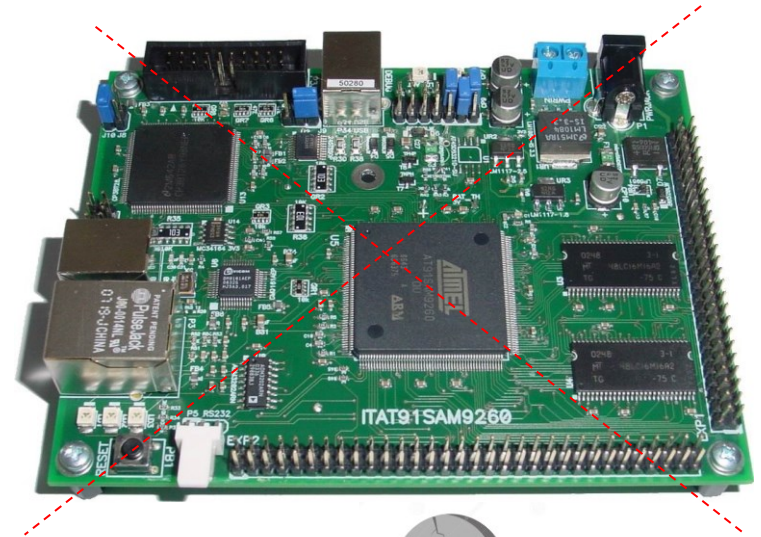
LAB 1.5 Big and Little Endian rules

Computer architecture CA

Computer STM32H750-DK

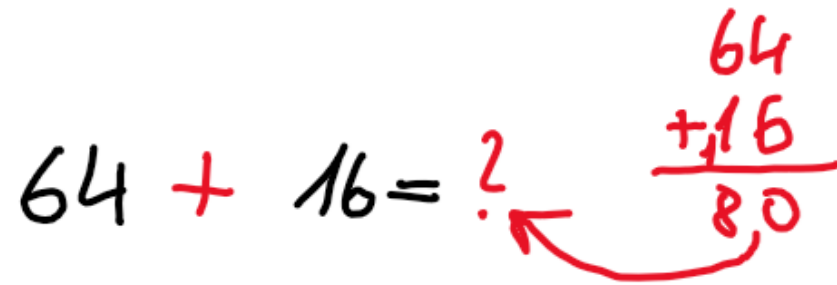


- Računalnik FRI-SMS
 - Mikrokontroler AT91SAM9260 iz družine mikrokontrolerov ARM9



LAB 1.6 Addition – human, python, assembler cases

Human (case: $64 + 16 = 80$)

$$64 + 16 = ?$$

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

Python (case: REZ = STEV1 + STEV2)

Adding two variables in Python.

<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

Objects

Print output (drag lower right corner to resize)

Global frame

STEV1	64
STEV2	16
REZ	80

```
STEV1 = 0x40
```

```
+STEV2 = 0x10
```

```
-----
```

```
REZ = 0x50
```

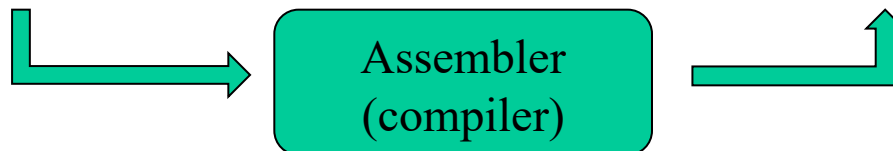

WinIDEA (case: rez = stev1 + stev2)

Evaluate the sum of two variables in ARM assembler.

Use prepared project from e-classroom)

Variables values are stored in the main memory. We perform a simple arithmetic addition with the following instructions:

Assembly language	Instruction description	Machine language
adr r0, stev1	$R0 \leftarrow \text{Addr. of stev1}$	0xE24F0014
ldr r1, [r0]	$R1 \leftarrow M[R0]$	0xE5901000
adr r0, stev2	$R0 \leftarrow \text{Addr. of stev2}$	0xE24F0018
ldr r2, [r0]	$R2 \leftarrow M[R0]$	0xE5902000
add r3, r2, r1	$R3 \leftarrow R1 + R2$	0xE0823001
adr r0, rez	$R0 \leftarrow \text{Addr. of rez}$	0xE24F0020
str r3, [r0]	$M[R0] \leftarrow R3$	0xE5803000

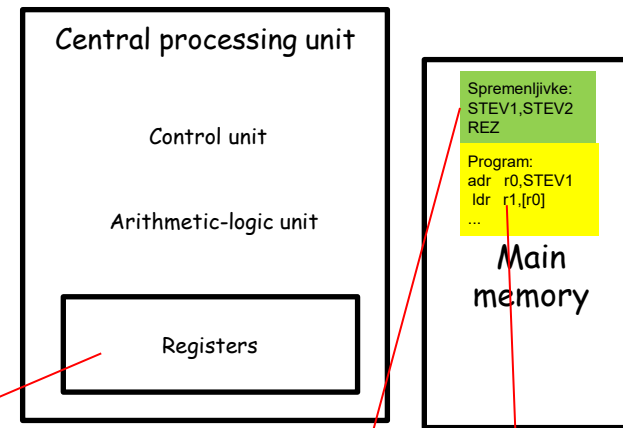


Execute instructions step-by-step and observe the register's values and the variable's values inside the main memory.

Practical example : Sum of 2 numbers

<https://cpulabz.org/?sys=arm&loadasm=share/s8zU3xx.s>

Assembly language	Instruction description	Machine language
adr r0, stev1	R0 ← Addr. of stev1	0xE24F0014
ldr r1, [r0]	R1 ← M[R0]	0xE5901000
adr r0, stev2	R0 ← Addr. of stev2	0xE24F0018
ldr r2, [r0]	R2 ← M[R0]	0xE5902000
add r3, r2, r1	R3 ← R1 + R2	0xE0823001
adr r0, rez	R0 ← Addr. of rez	0xE24F0020
str r3, [r0]	M[R0] ← R3	0xE5803000



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

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 ?

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
0000002c	e24f0014	9 .align 11 .global _start 12 _start: 14 adr r0, STEV1 _start: adr r0, 0x20 (0x20: enc
00000030	e5901000	15 ldr r1, [r0]
00000034	e24f0018	17 adr r0, STEV2 adr r0, 0x24 (0x24: enc
00000038	e5902000	18 ldr r2, [r0]
0000003c	e0813002	20 add r3, r1, r2
00000040	e24f0020	22 adr r0, REZ adr r0, 0x28 (0x28: enc
00000044	e5803000	23 str r3, [r0]
00000048	eaffffffe	26 end: b er end: b 0x48 (0x48: enc

Memory (Ctrl-M)

Go to address, label, or register:

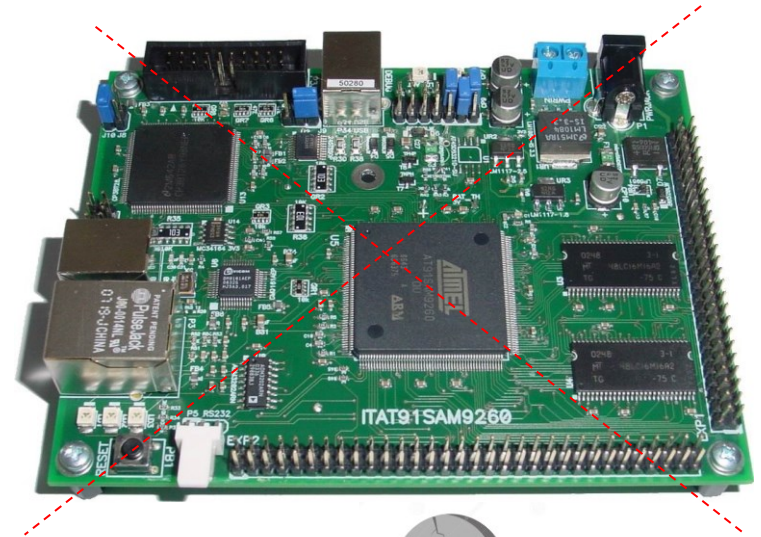
Address	Memory contents and ASCII
00000000	00000000 00000000 00000000 00000000
00000010	00000000 00000000 00000000 00000000
00000020	00000010 00000040 00000000 e24f0014
00000030	e5901000 e24f0018 e5902000 e0813002
00000040	e24f0020 e5803000 eaffffffe 00000000
00000050	aaaaaaaa aaaaaaaaaa aaaaaaaaaa aaaaaaaaaa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

Computer architecture CA

Computer STM32H750-DK



- Računalnik FRI-SMS
 - Mikrokontroler AT91SAM9260 iz družine mikrokontrolerov ARM9



LAB 1.7 Notes – empty templates

Python (case: REZ = STEV1 + STEV2)

Frames

Objects

Global frame

STEV1	64
STEV2	16
REZ	80

Python 2.7

```
1 STEV1=0x40
```

```
2 STEV2=0x10
```

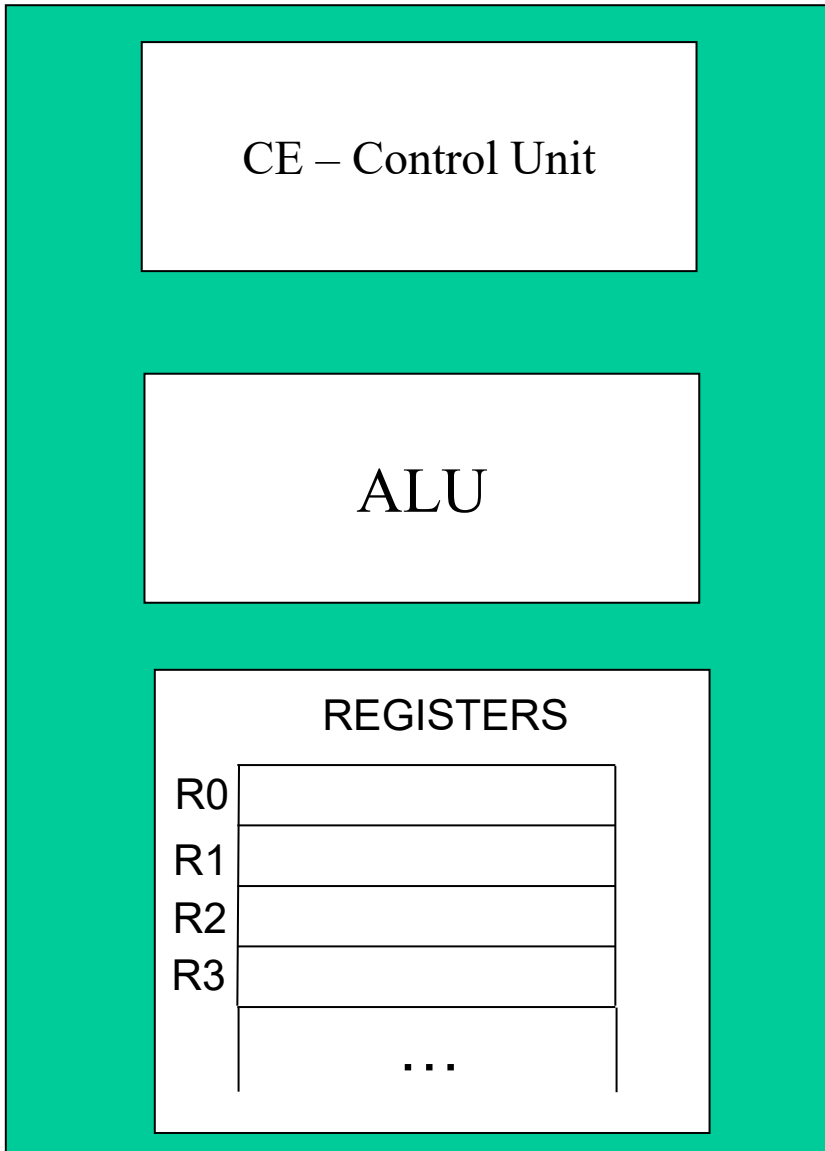
```
3 REZ = STEV1 + STEV2
```

```
→ 4 print (" STEV1 = " + hex(STEV1) + "\n+STEV2 = " + hex(STE
```

<http://goo.gl/YXQ5qN>

Case: adding two numbers

CPU



Memory

Address	Memory words (locations)	Label Content
0x00 = 0		
0x01 = 1		
0x02 = 2		
	...	
0x20 = 0		STEV1
0x24 = 4		STEV2
0x28 = 8		REZ
0x2C = 12		1. instruction
		ADR R0,STEV1

Case: adding two numbers

INSTRUCTIONS

	Machine Instr.	Assembly Instr.	Description	Comment
1.	0xE24F0014	adr r0, stev1	$R0 \leftarrow \text{Addr. of stev1}$	
2.	0xE5901000	ldr r1, [r0]	$R1 \leftarrow M[R0]$	
3.	0xE24F0018	adr r0, stev2	$R0 \leftarrow \text{Addr. of stev2}$	
4.	0xE5902000	ldr r2, [r0]	$R2 \leftarrow M[R0]$	
5.	0xE0823001	add r3, r2, r1	$R3 \leftarrow R1 + R2$	
6.	0xE24F0020	adr r0, rez	$R0 \leftarrow \text{Addr. of rez}$	
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

