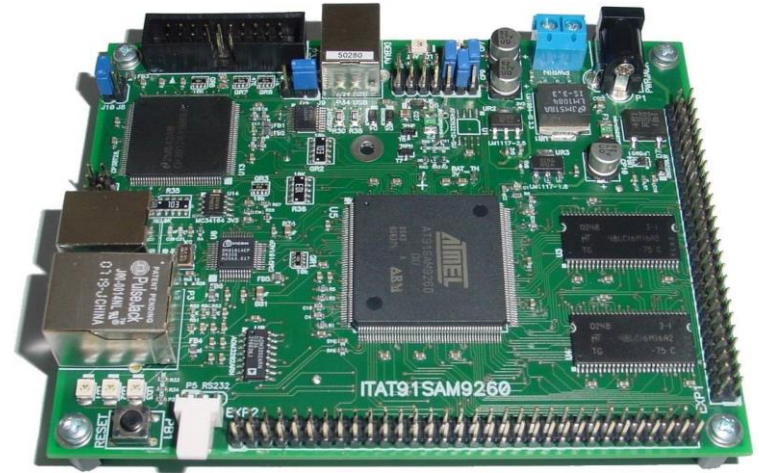


# Computer architecture CA

Computer STM32H750-DK



- Računalnik FRI-SMS
  - Mikrokontrolnik AT91SAM9260 iz družine mikrokontrolnikov ARM9

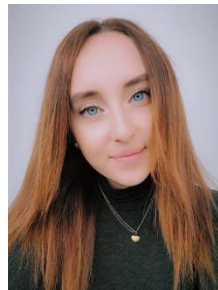


Team CA

Tutors



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



Anamari Orehar  
[ao6477@student.unilj.si](mailto:ao6477@student.unilj.si)



Kristian Šurbek  
[ks5453@student.unilj.si](mailto:ks5453@student.unilj.si)



Andrej Sušnik  
[as1767@student.uni-lj.si](mailto:as1767@student.uni-lj.si)



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

# Computer architecture CA

Computer STM32H750-DK



- Računalnik FRI-SMS
  - Mikrokontrolnik AT91SAM9260 iz družine mikrokontrolnikov 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** elective course and others (Input/Output devices, ...)



# Content of LAB work



- Basic knowledge needed from lectures (e.g. memory address, memory words, ...)
- **Core: Programming in ARM assembly language**
- Format: lab 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

*\*Due to Covid, can be changed*

# 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
    - \*in case of Covid lockdown, 1. and 2. test change to homeworks and 3. test becomes a part of written and/or oral exam
- Final lab grade is valid only for the current academic year. You need to repeat lab work in new school year.
- *\*Due to Covid, grading can be changed*



# Web simulator cpulator

- <https://cpulator.01xz.net/?sys=arm>

The screenshot displays the cpulator web simulator interface. At the top, there is a control bar with buttons for 'Stopped', 'Step Into' (F2), 'Step Over' (Ctrl-F2), 'Step Out' (Shift-F2), 'Continue' (F3), 'Stop' (F4), 'Restart' (Ctrl-R), and 'Reload' (Ctrl-Shift-L). There are also 'File' and 'Help' dropdown menus.

The main interface is divided into several panels:

- Registers:** A list of registers (r0-r13) with their current values, all shown as 00000000.
- Editor (Ctrl-E):** A code editor showing assembly code for ARMv7. The code includes:

```
1 stev1: .word 0x40
2 stev2: .word 0x10
3 rez: .space 4
4
5 .global _start
6 _start:
7
8 adr r0, stev1
9 ldr r1, [r0]
10
11 adr r0, stev2
12 ldr r2, [r0]
13
14 add r3, r2, r1
15
16 adr r0, rez
17 str r3, [r0]
18
19 loop: b loop
20
21
```
- Memory (Ctrl-M):** A memory dump showing addresses and their contents. The address range is from 00000000 to 000001b0. The contents are mostly 'aa aa aa aa', with some specific values at the beginning: 40 00 00 00, 10 00 00 00, 20 00 4f e2, 30 00 4f e2, and 80 00 4f e2.
- Messages:** A log showing the compilation process:

```
Code and data loaded from ELF-executable into memory. Total size is 48 bytes.
Assemble: arm-altera-eabi-as -mfloat-abi=soft -march=armv7-a -mcpu=cortex-a9 -mfpu=neon-fp16 --gdwarf2 -o work/asmV0FoCU.s.o work/a
Link: arm-altera-eabi-ld --script build_arm.ld -e _start -u _start -o work/asmV0FoCU.s.elf work/asmV0FoCU.s.o
Compile succeeded.
```

# Integrated Development Environment (IDE) WinIDEA



simpr - winIDEA - [C:\winIDEA\Projekti\Zgled\_2020\user.s]

File View Project Simulator Debug Test Plugins Tools Window Help

Project Workspace

Filter

sample.elf

```
user.s crt0.s sample.lcf
stev2: .word 0x10
rez: .space 4

.align
.global __start
__start:

    ldr r1, stev1
    ldr r2, stev2
    add r3, r2, r1
    str r3, rez

end: b end
```

Memory 0x0000000

Area	Virtual	Address	Symbol
		00000000	09 00 00 EA 08 00 00 EA
		00000008	07 00 00 EA 06 00 00 EA
		00000010	05 00 00 EA 04 00 00 EA
		00000018	03 00 00 EA 02 00 00 EA
		00000020	40 00 00 00 10 00 00 00
		00000028	00 00 00 00 14 10 1F E5
		00000030	14 20 1F E5 01 30 82 E0
		00000038	18 30 0F E5 FE FF FF EA
		00000040	00 00 00 00 00 00 00 00
		00000048	00 00 00 00 00 00 00 00
		00000050	00 00 00 00 00 00 00 00
		00000058	00 00 00 00 00 00 00 00
		00000060	00 00 00 00 00 00 00 00

Disassembly

Address Data Disassembly Registers

Address	Data	Disassembly	Registers
		<u>start</u>	R0 00000000
		ldr r1, stev1	R1 00000000
000(14101)		ldr r1, [pc, -0014]	R2 00000000
		ldr r2, stev2	R3 00000000
000(14201)		ldr r2, [pc, -0014]	R4 00000000
		add r3, r2, r1	R5 00000000
000(01308)		add r3, r2, r1	R6 00000000
		str r3, rez	R7 00000000
000(18300)		str r3, [pc, -0018]	R8 00000000
		__end: b __end	R9 00000000
			R10 00000000

Output

Compiling ...  
crt0.s  
user.s  
Linking  
"sample.elf (Dir:C:\winIDEA\Projekti\Zgled\_2020\Debug\)" ... was successfully generated.  
0 Error(s) 0 Warning(s)

Build Find In Files Tools Script

# Computer architecture CA

Computer STM32H750-DK



- Računalnik FRI-SMS
  - Mikrokontrolnik AT91SAM9260 iz družine mikrokontrolnikov ARM9



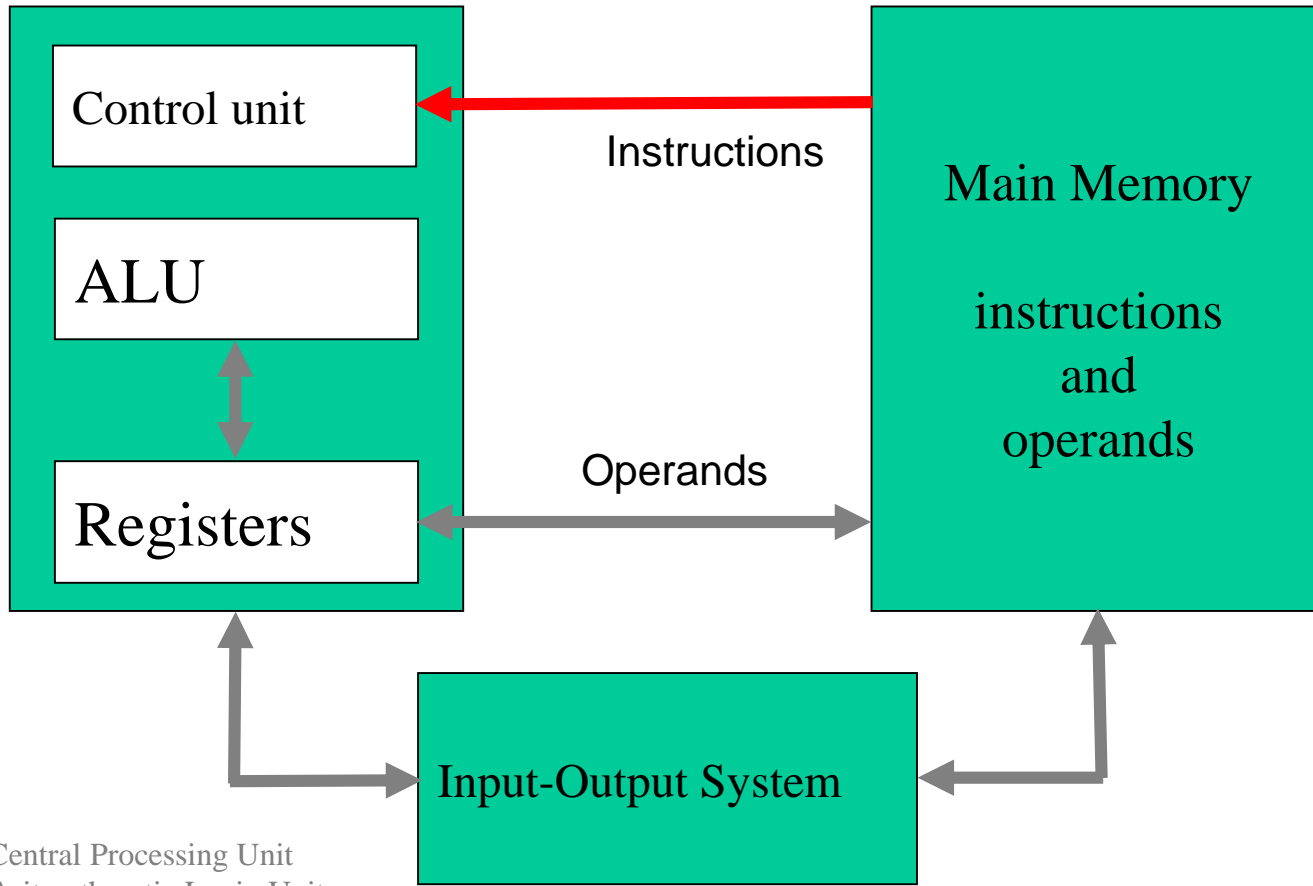
LAB 1.2 Von Neumann model (VN)



---

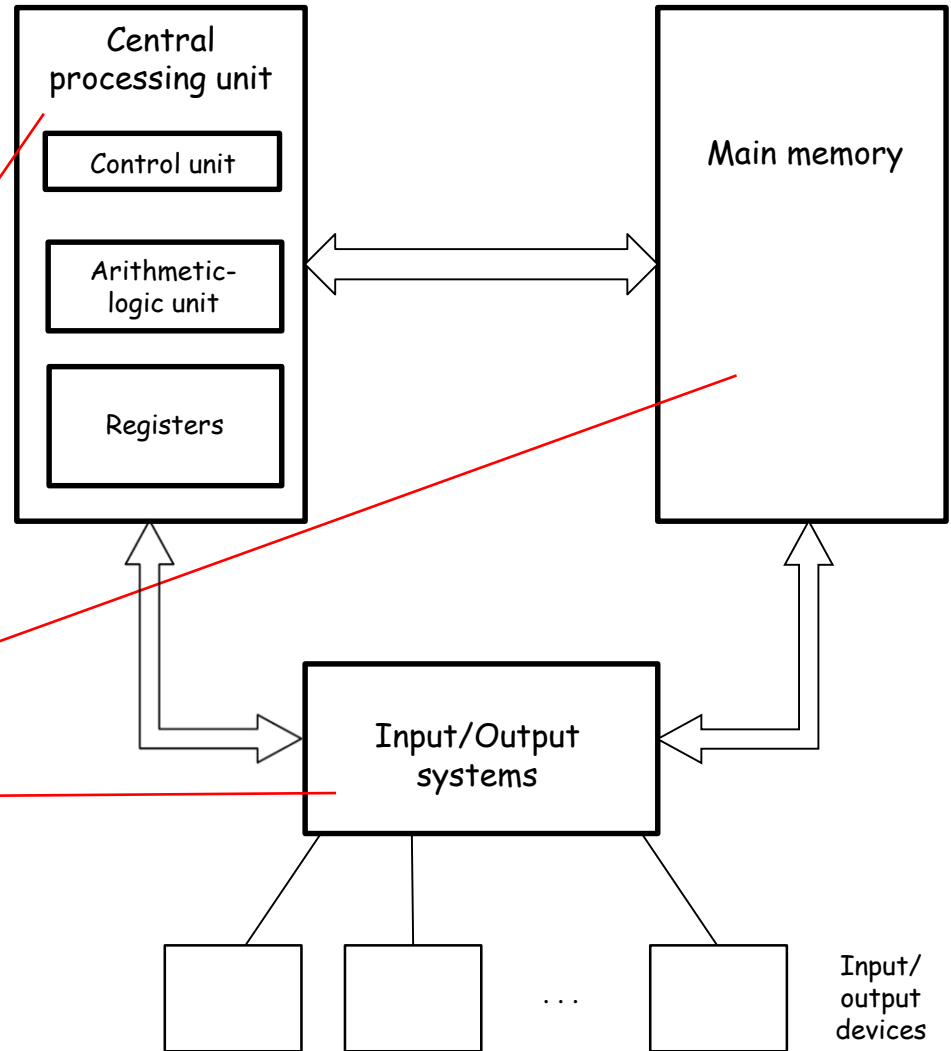
# Von Neumann Computer Model

## CPU



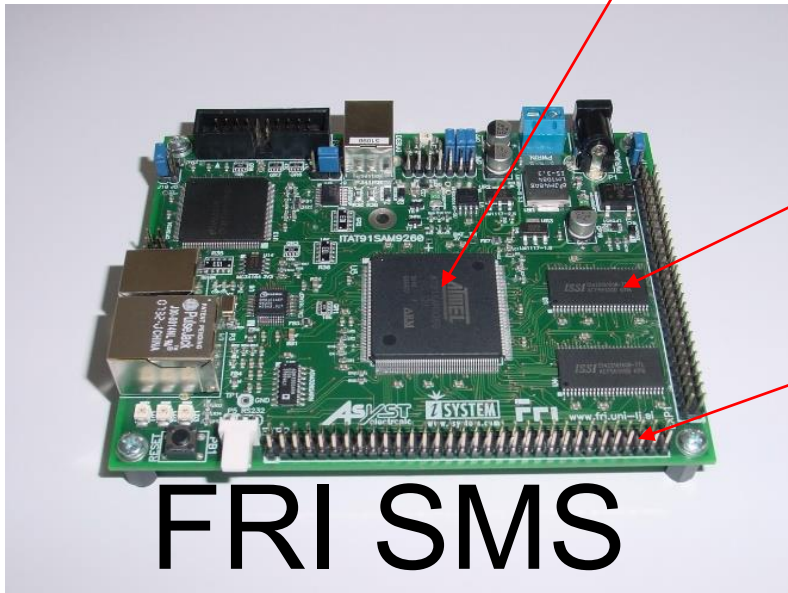
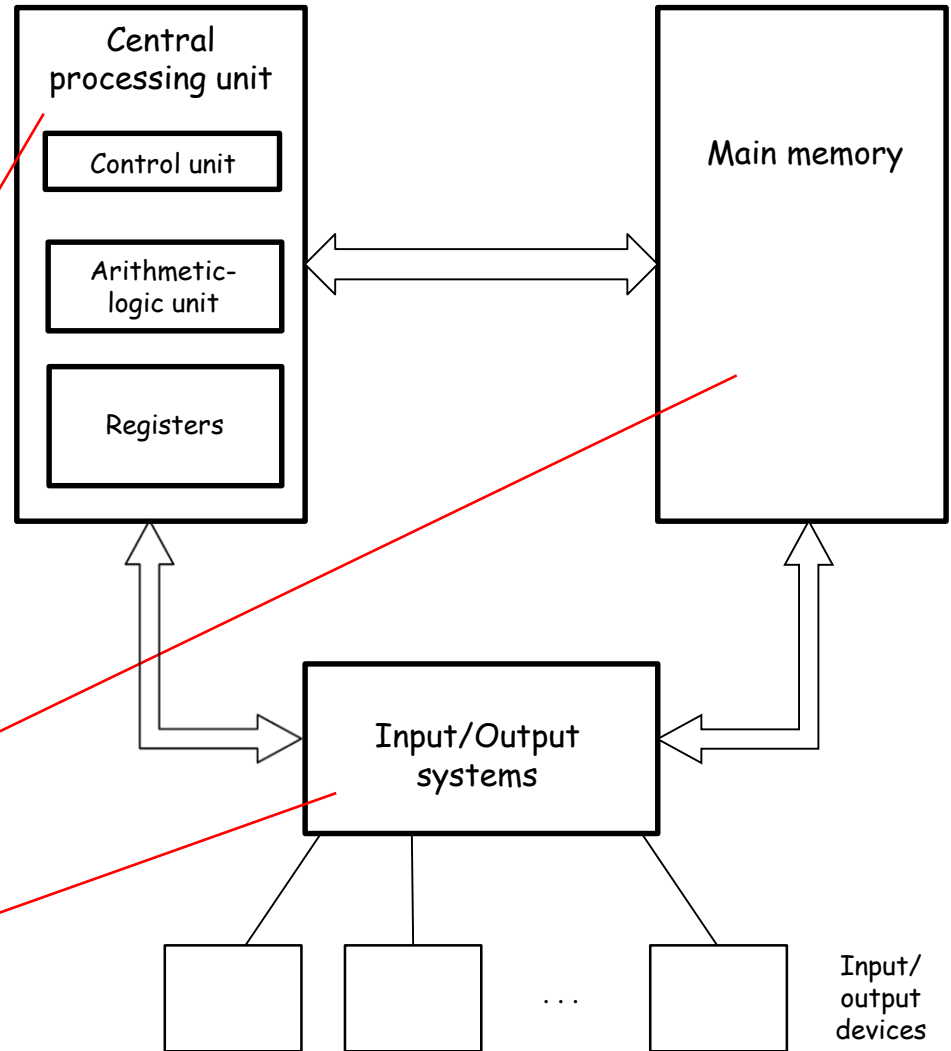
CPU – Central Processing Unit  
ALU – Arithmetic Logic Unit

# The basic computer model

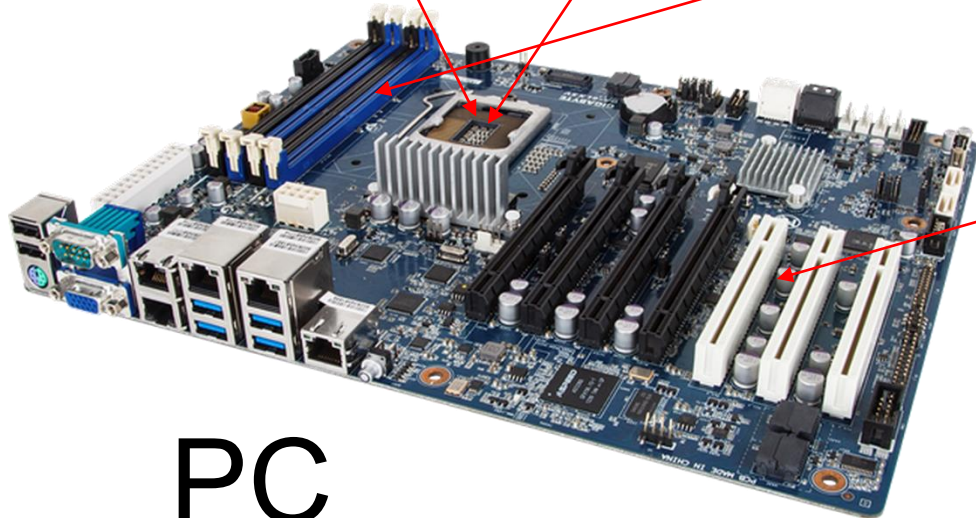


MicroControllers

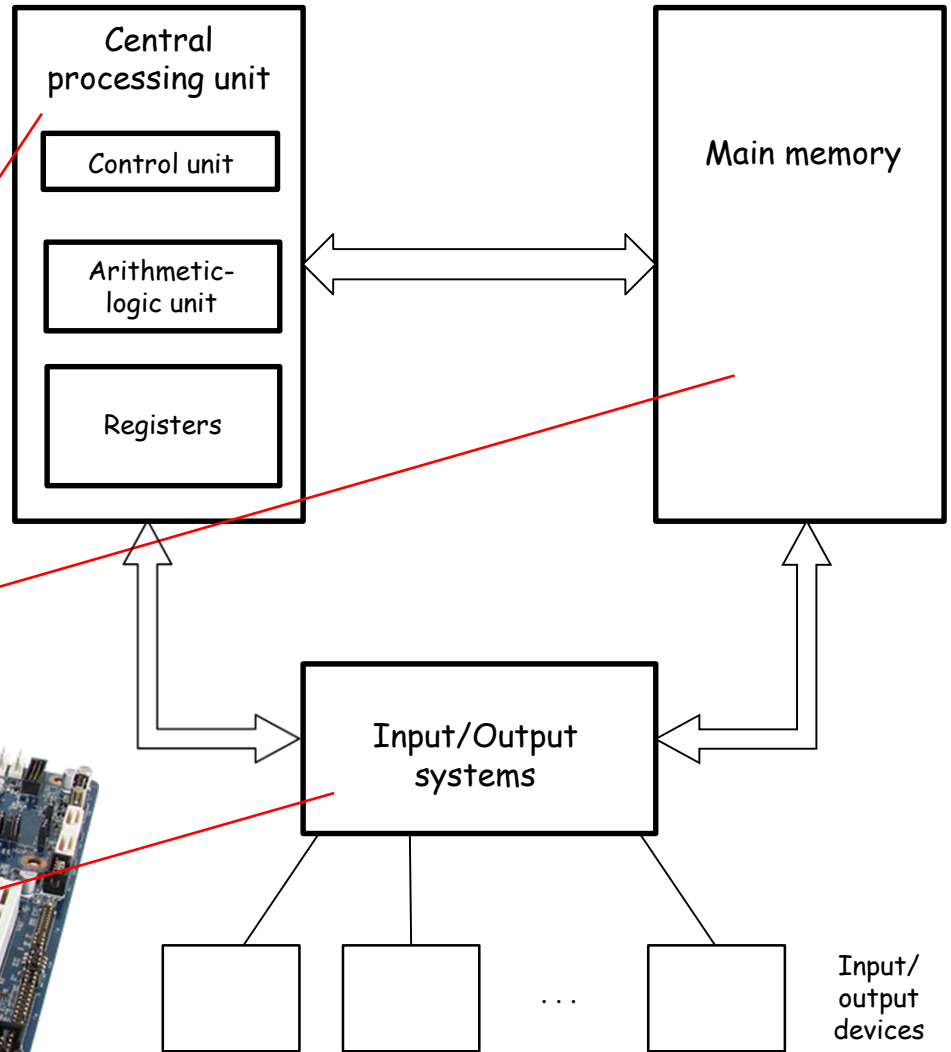
# The basic computer model



# The basic computer model



PC

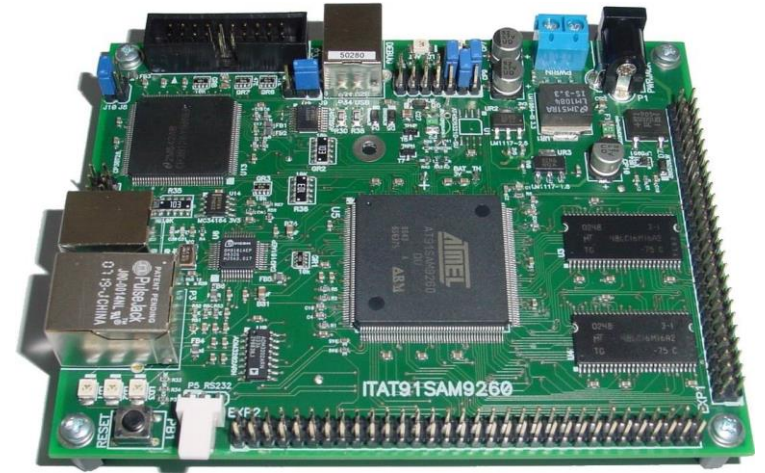


# Computer architecture CA

Computer STM32H750-DK



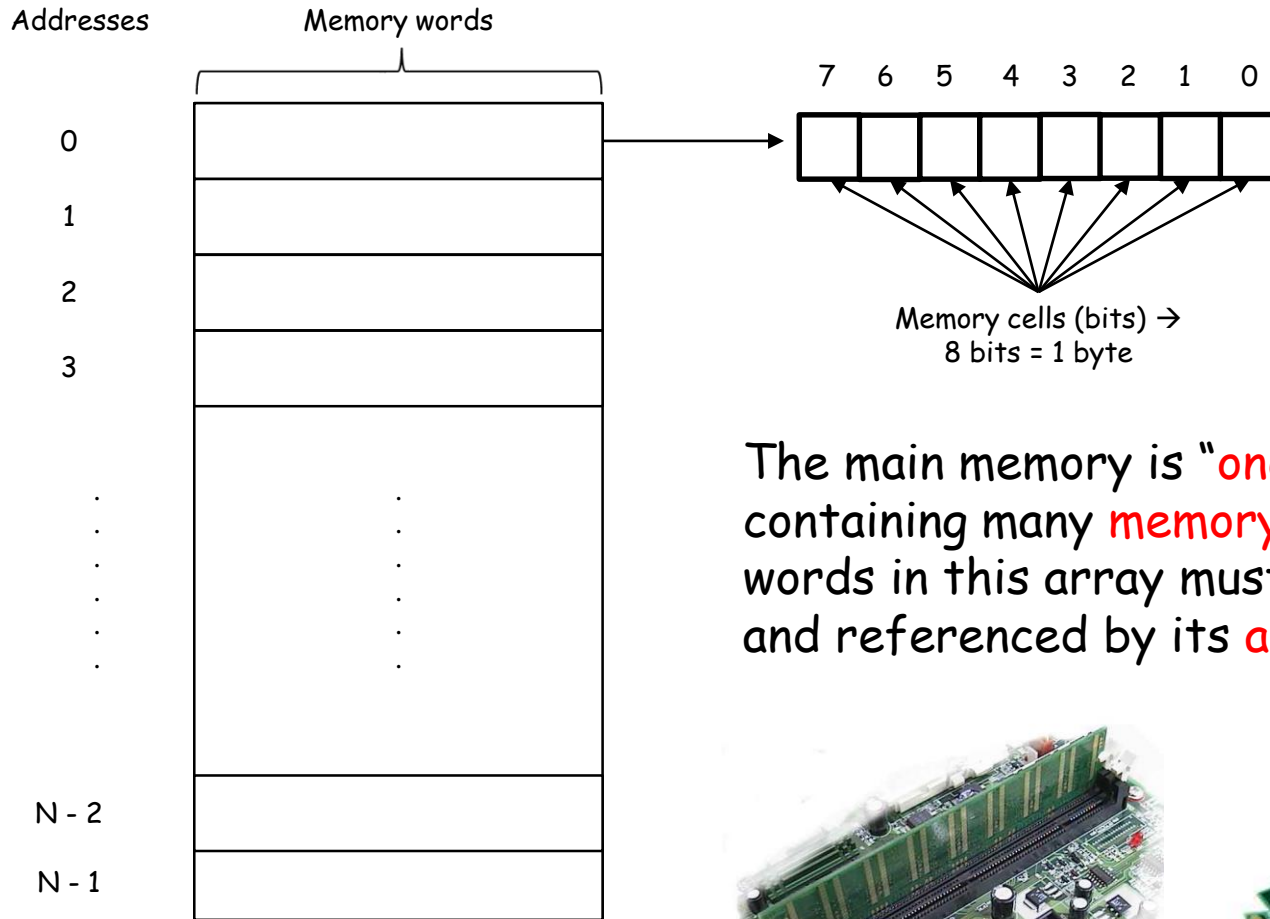
- Računalnik FRI-SMS
  - Mikrokontroler AT91SAM9260 iz družine mikrokontrolerov ARM9



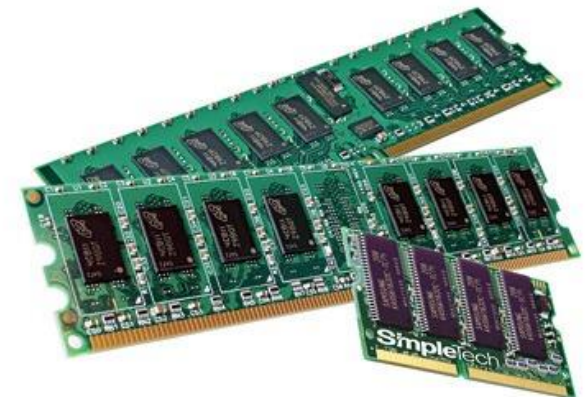
LAB 1.3 Memory



# What is memory ?

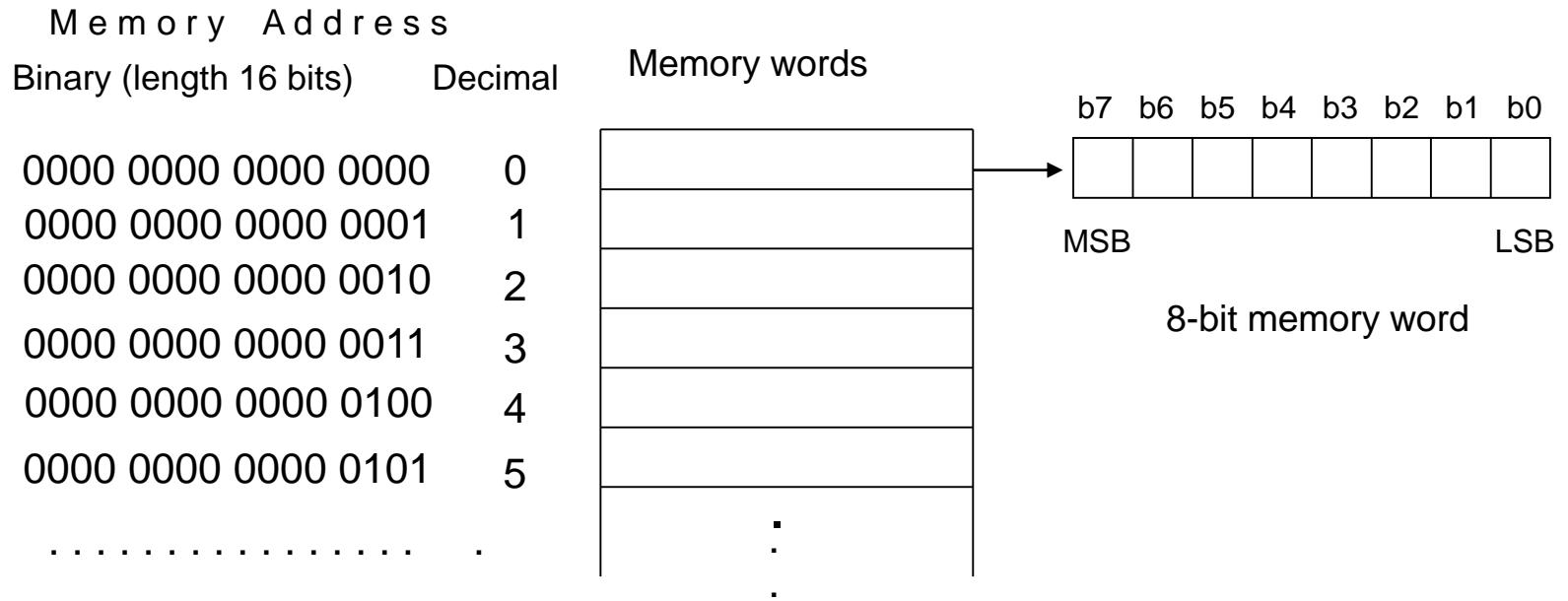


The main memory is "one dimensional array" containing many **memory words**. Each memory words in this array must be **uniquely** identified and referenced by its **address**!



# Main Memory in von Neumann Computer

---

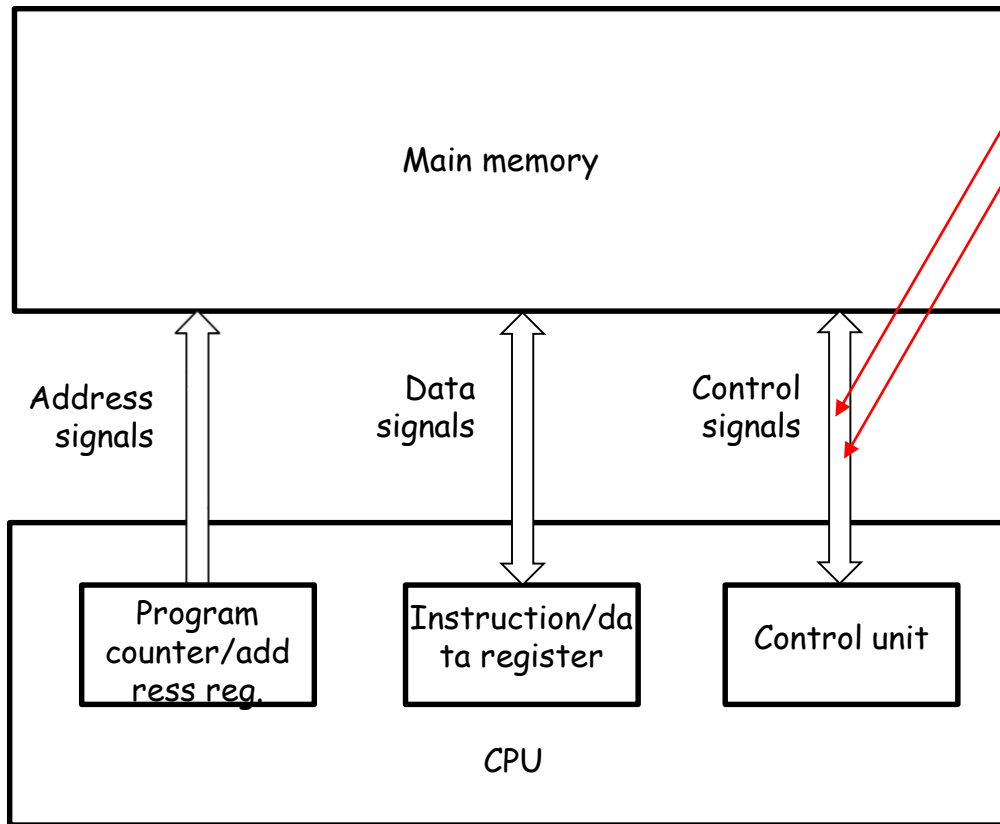


Memory Address			Memory words
Binary (length 16 bits)	Hexadecimal	Decimal	
0000 0000 0000 0000	0000	0	
0000 0000 0000 0001	0001	1	
0000 0000 0000 0010	0002	2	
0000 0000 0000 0011	0003	3	
0000 0000 0000 0100	0004	4	
0000 0000 0000 0101	0005	5	
.....	.		.
.....	.		.
1111 1111 1111 1011	FFFB	65531	
1111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

# Interconnection CPU <-> main memory

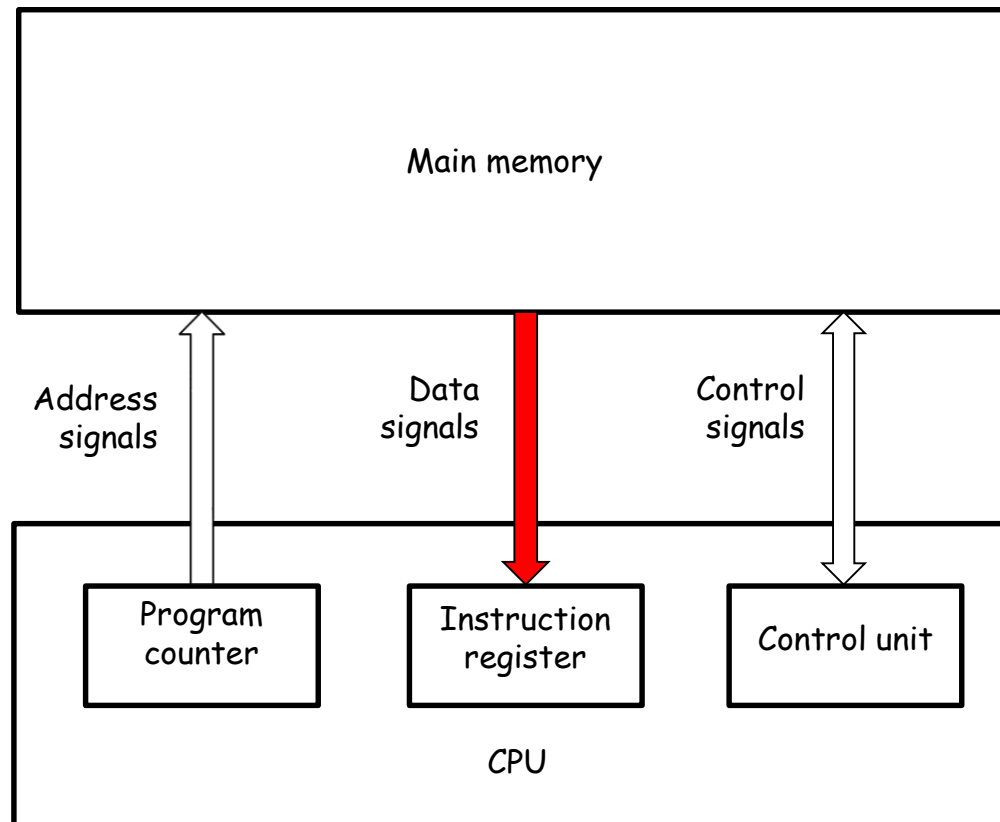
Bus = a group of related lines  
(Address, Data, Control buses)

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



# How does CPU access the main memory?

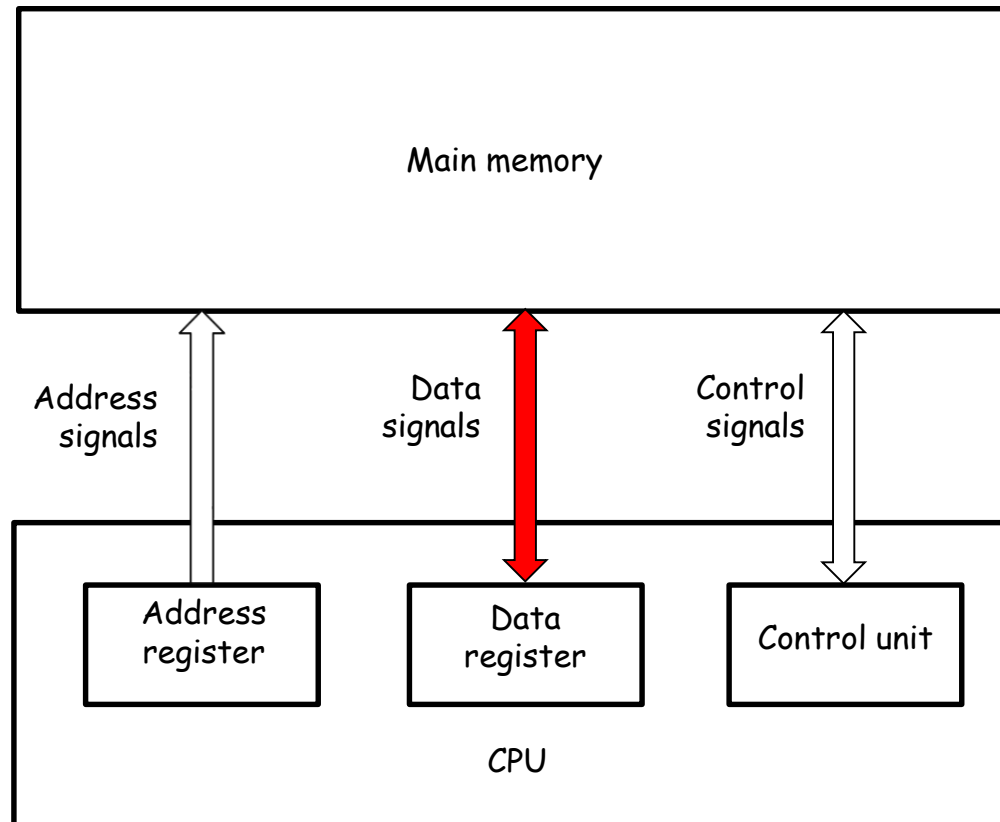
Example for accessing instructions:





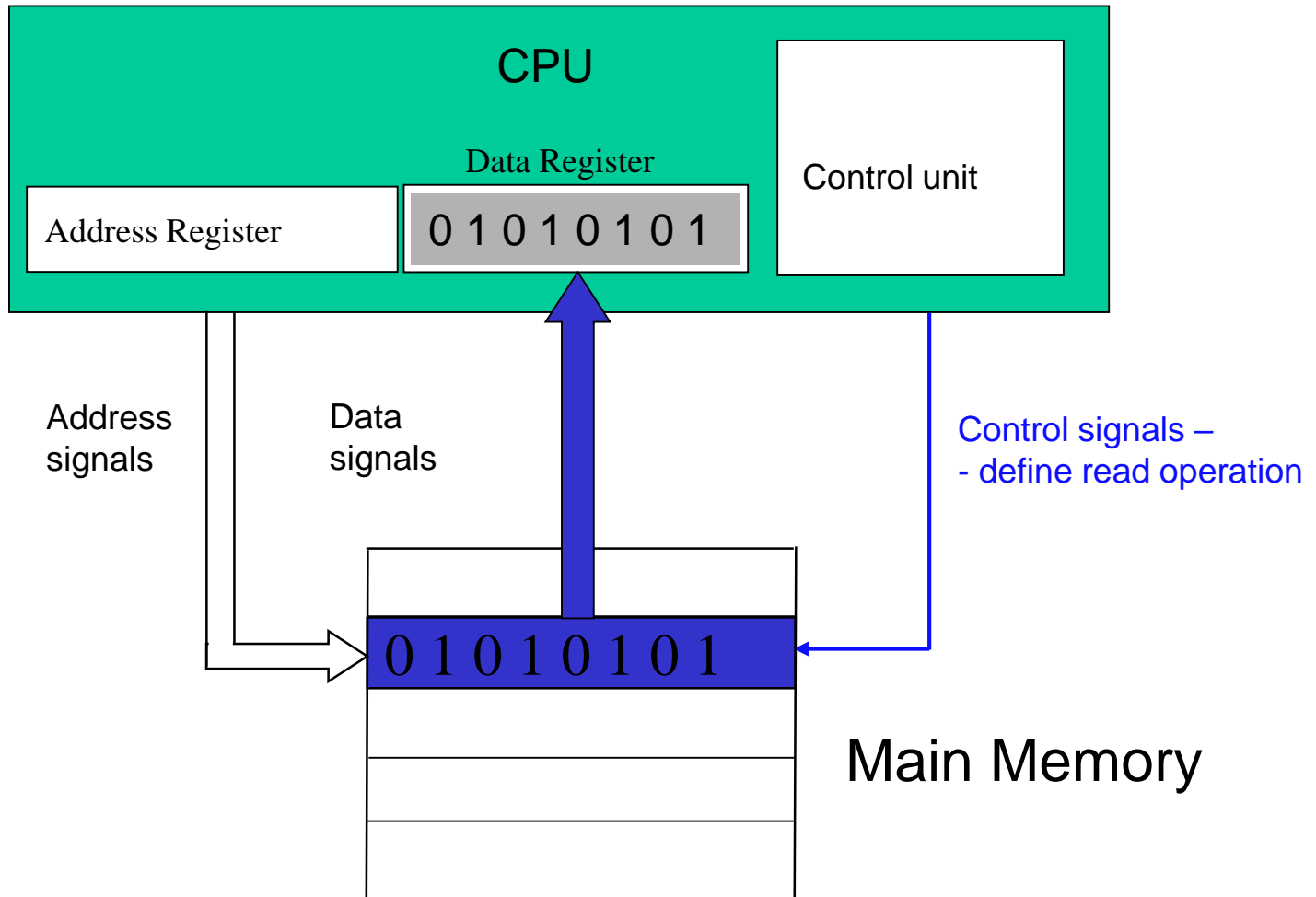
# How does CPU access the main memory?

Examples for accessing operands:



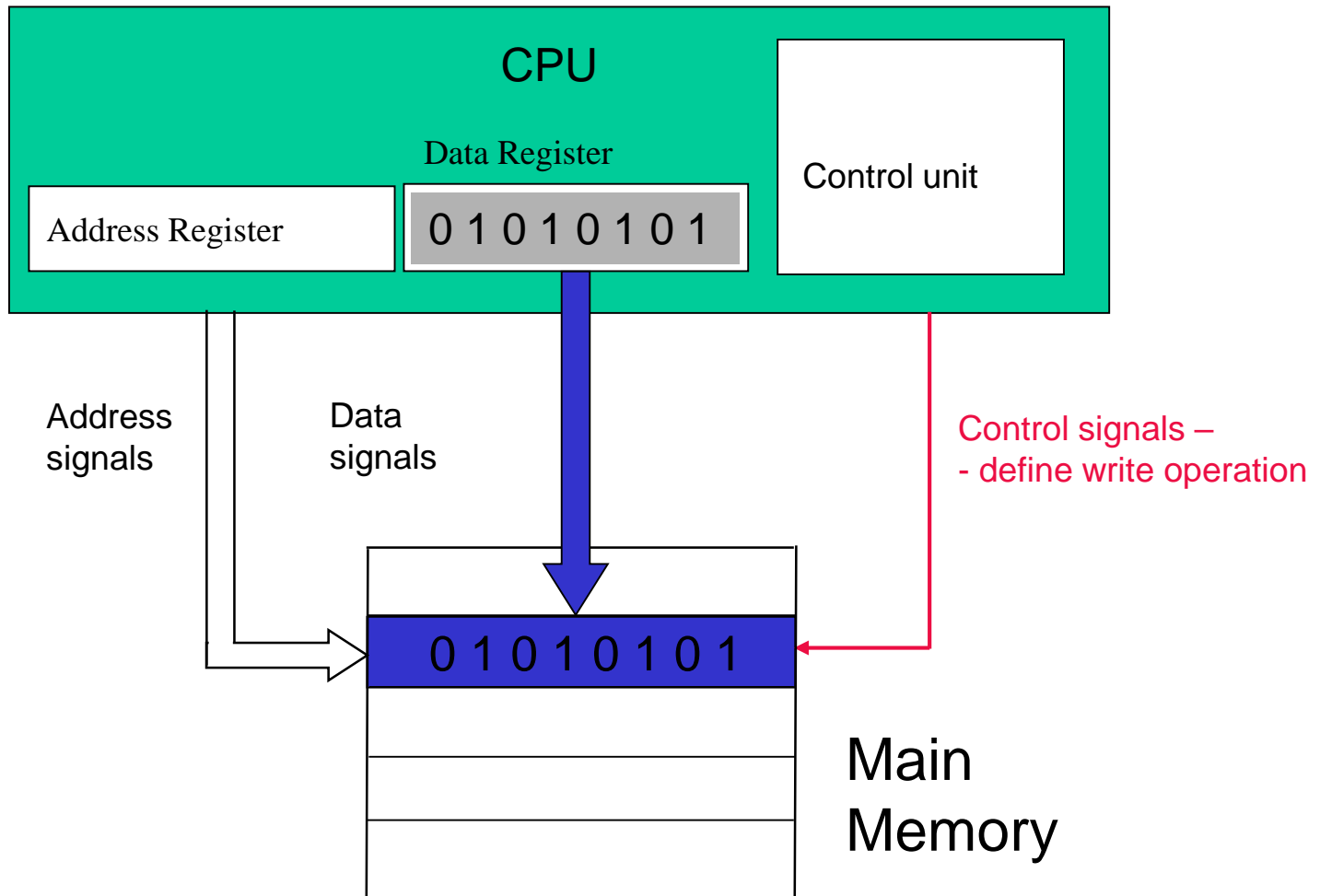
# CPU and Main Memory

## – read access



# CPU and Main Memory

## – write access



# Computer architecture CA

Computer STM32H750-DK



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

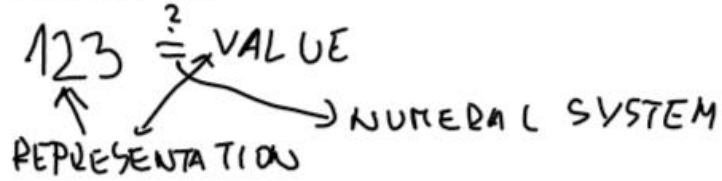


LAB 1.4 Quick intro to numeral systems

# LAB 1.4 Quick intro to numeral systems

Numeral systems - short intro

petek, 16. oktober 2020 08:50



DECIMAL: (BASE 10) 0-9

$$123 = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0 = 100 + 20 + 3 = \underline{\underline{123}}$$

BINARY: (BASE 2) BIN 0,1

$$0111 = 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 4 + 2 + 1 = \underline{\underline{7}}$$

HEXADECIMAL: (BASE 16) HEX

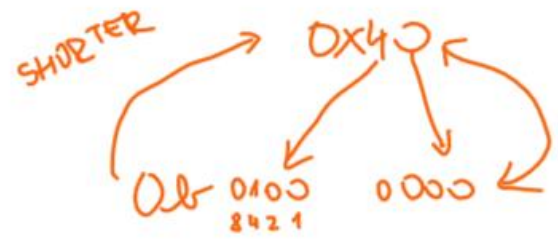
$$0x40 = 4 \cdot 16^1 + 0 \cdot 16^0 = \underline{\underline{64}}$$

DIGITS

HEX	VALUE	BIN
0	0	0000
...	...	...
9	9	
A	10	
B	...	
F	15	1111

RELATED

1 DIGIT HEX ↔ 4 DIGITS IN BIN





# Computer architecture CA

Computer STM32H750-DK



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

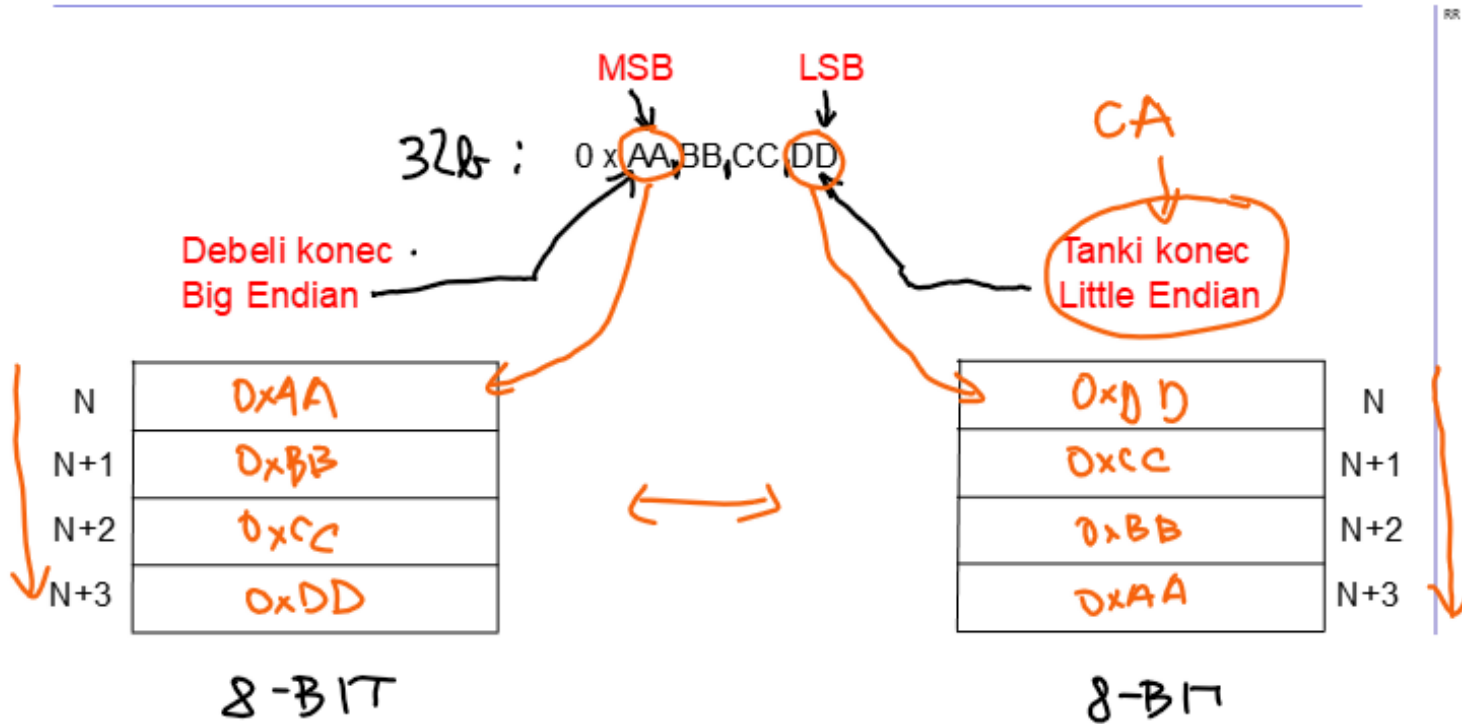


LAB 1.5 Big and Little Endian rules

# LAB 1.5 Big and Little Endian rules

Big vs. Little Endian

Monday, October 12, 2020 2:18 PM



# Computer architecture CA

Computer STM32H750-DK

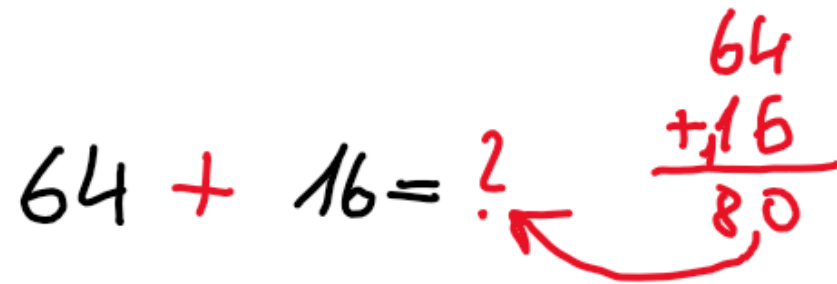


- Računalnik FRI-SMS
  - Mikrokontroler AT91SAM9260 iz družine mikrokontrolerov ARM9



LAB 1.6 Addition – human, python,  
assembler cases

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

$$64 + 16 = ?$$


A handwritten vertical addition problem is shown to the right of the equation. It consists of the number 64, followed by a plus sign and 16, a horizontal line, and the result 80. A red arrow originates from the 80 and points to the question mark in the equation  $64 + 16 = ?$ .

# 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
```

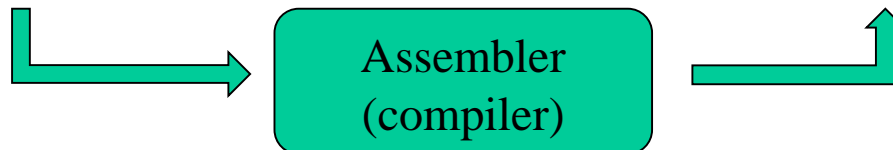
# WinIDEA (case: rez = stev1 + stev2 )

Evaluate the sum of two variables in ARM assembler.

**(WinIdea: Download prepared project from e-classroom)**

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

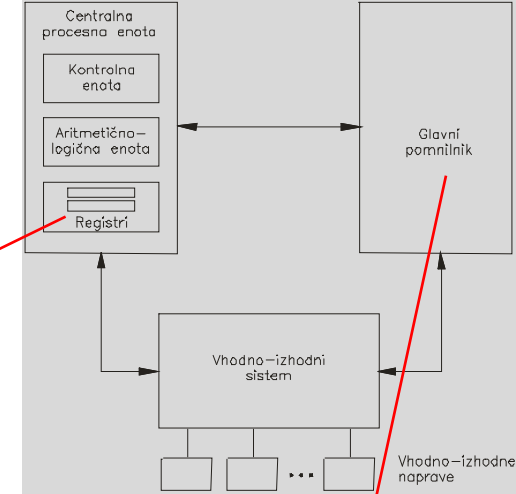
Assembly language	Instruction description	Machine language
adr r0, stev1	$R0 \leftarrow \text{Addr. of stev1}$	0xE24F0014
ldr r1, [r0]	$R1 \leftarrow M[R0]$	0xE5901000
adr r0, stev2	$R0 \leftarrow \text{Addr. of stev2}$	0xE24F0018
ldr r2, [r0]	$R2 \leftarrow M[R0]$	0xE5902000
add r3, r2, r1	$R3 \leftarrow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

<https://cpulator.01xz.net/?sys=arm>



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	00000008
r1	00000040
r2	00000010
r3	00000050
r4	00000000
r5	00000000
r6	00000000
r7	00000000
r8	00000000
r9	00000000
r10	00000000
r11	00000000
r12	00000000

Editor (Ctrl-E)

Compile and Load (F5) Language: ARMv7 untitled.s [changed since save]

```
1 stev1: .word 0x40
2 stev2: .word 0x10
3 rez: .space 4
4
5 .global _start
6 _start:
7
8 adr r0, stev1
9 ldr r1, [r0]
10
11 adr r0, stev2
12 ldr r2, [r0]
13
14 add r3, r2, r1
15
16 adr r0, rez
17 str r3, [r0]
18
19 loop: b loop
20
21
```

Memory (Ctrl-M)

Go to address, label, or register:

Address	Memory contents and ASCII
TTTTTT00	aa aa aa aa aa aa aa aa
ffffffd0	aa aa aa aa aa aa aa aa
ffffffe0	aa aa aa aa aa aa aa aa
fffffff0	aa aa aa aa aa aa aa aa
00000000	40 00 00 00 10 00 00 00
00000010	00 10 90 e5 18 00 4f e2
00000020	20 00 4f e2 00 30 80 e5
00000030	aa aa aa aa aa aa aa aa
00000040	aa aa aa aa aa aa aa aa
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

Messages

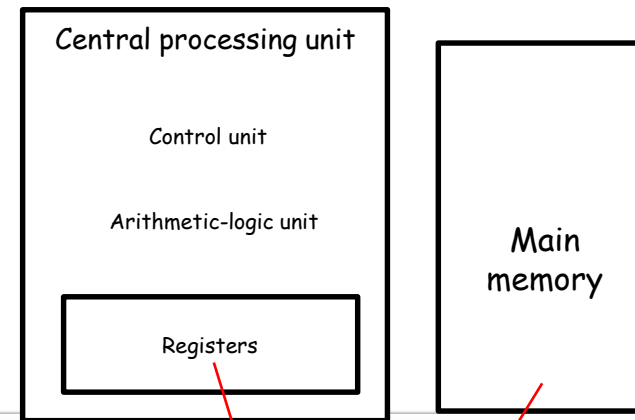
Code and data loaded from ELF executable into memory. Total size is 48 bytes.

Assemble: arm-altera-eabi-as -mfloat-abi=soft -march=armv7-a -mcpu=cortex-a9 -mfpu=neon-fp16 --gdwarf2 -o work/asmV0FoCU.s.o work/a  
Link: arm-altera-eabi-ld --script build\_arm.ld -e \_start -u \_start -o work/asmV0FoCU.s.elf work/asmV0FoCU.s.o  
Compile succeeded.



# Practical example „zglesd“

## Integrated Development Environment (IDE) WinIDEA



The screenshot shows the WinIDEA IDE interface. The main window displays assembly code for a program named 'user.s'. The code includes labels 'stev2', 'rez', and '\_\_start', with instructions like 'ldr', 'add', and 'str'. The 'Registers' window on the right shows a memory dump starting at address 0x00000000. The dump shows hexadecimal values for memory locations, with a red arrow pointing to the value 'EA 08 00 00 EA' at address 00000010.

```
stev2:  .word 0x10
rez:    .space 4

        .align
        .global __start
__start:

        ldr r1, stev1
        ldr r2, stev2
        add r3, r2, r1
        str r3, rez

end:    b     end
```

Area	Virtual	Address	Symbol	Value
		00000000		09 00 00 EA 08 00 00 EA
		00000008		07 00 00 EA 06 00 00 EA
		00000010		05 00 00 EA 04 00 00 EA
		00000018		03 00 00 EA 02 00 00 EA
		00000020		40 00 00 00 10 00 00 00
		00000028		00 00 00 00 14 10 1F E5
		00000030		14 20 1F E5 01 30 82 E0
		00000038		18 30 0E E5 FE FF FF EA
		00000040		00 00 00 00 00 00 00 00
		00000048		00 00 00 00 00 00 00 00
		00000050		00 00 00 00 00 00 00 00
		00000058		00 00 00 00 00 00 00 00
		00000060		00 00 00 00 00 00 00 00

### Razvojno okolje WinIDEA (dokumentacija, začetni projekti)

- Instalacija orodja WinIdea - Windows
  - Video: Odpiranje začetnega projekta za simulator
  - Začetni projekt za winIDEA (simulator)
  - Začetni projekt za winIDEA 2016 (ploščica)
  - [Skrto za udeležence](#)
  - Instalacija orodja WinIdea - Linux
  - Seznam ukazov zbirnika ARM
  - ARM sistem FRI-SMS (ploščica)
  - [Skrto za udeležence](#)
  - WinIdea: Software User Guide
  - Povezava na online Help za WinIdea
- [Download](#)

The screenshot shows the 'Registers' window in WinIDEA. It displays a list of registers from R0 to R15, each with its corresponding hexadecimal value. The registers are listed in a column, with R0 at the top and R15 at the bottom. The values are mostly zeros, except for R0 which is 00000000.

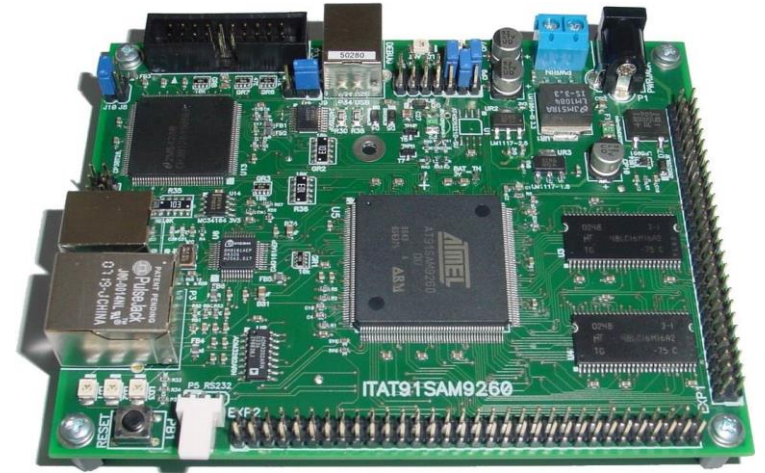
Register	Value
R0	00000000
R1	00000000
R2	00000000
R3	00000000
R4	00000000
R5	00000000
R6	00000000
R7	00000000
R8	00000000
R9	00000000
R10	00000000
R11	00000000
R12	00000000
R13	00000000
R14	00000000
R15	00000000

# Computer architecture CA

Computer STM32H750-DK



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



LAB 1.7 Notes 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
```

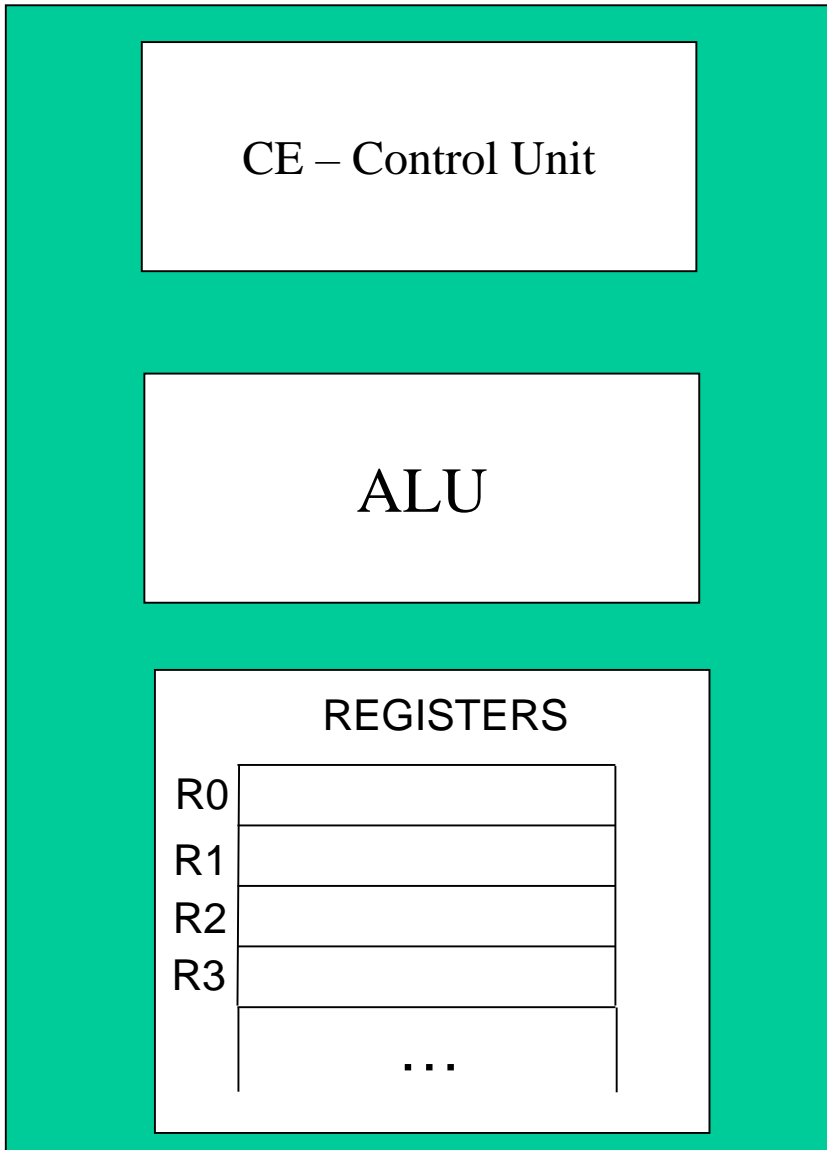
```
→ 4 print (" STEV1 = " + hex(STEV1) + "\n+STEV2 = " + hex(STE
```

---

<http://goo.gl/YXQ5qN>

# Zgled: adding two numbers

## CPU



## Memory

Address	Memory words (locations)	Label Content
0x00 = 0		STEVI
0x04 = 4		STEVI
0x08 = 8		REZ
0x0C = 12		1. instruction ADR R0,STEVI

## 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

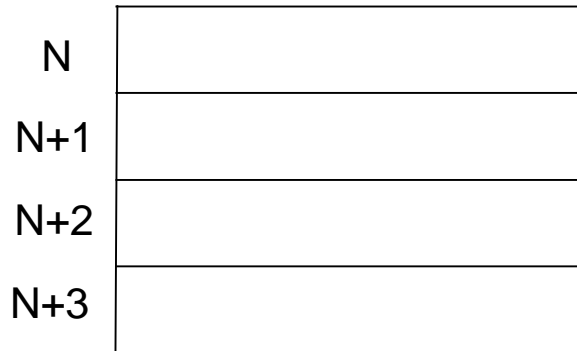
---

MSB

LSB

0 x AA BB CC DD

Debeli konec  
Big Endian



Tanki konec  
Little Endian

