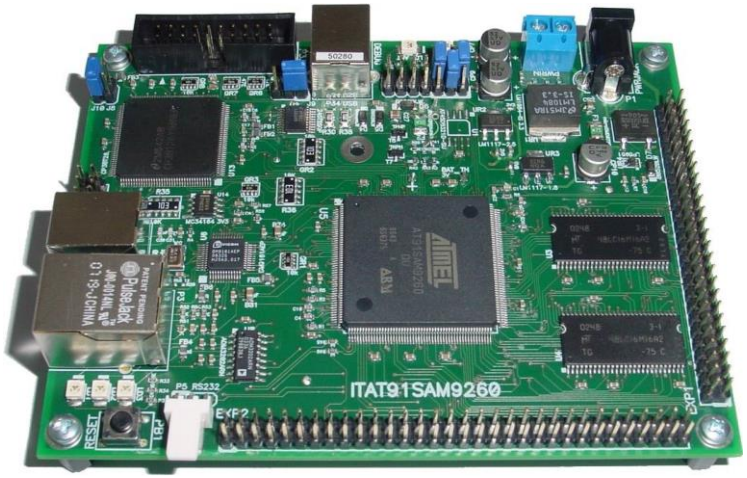
Computer architecture CA

Computer STM32H750-DK

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



Team CA

Tutors

Žiga Pušnik
ziga.pusnik@fri....

Anamari Orehar
ao6477@student.uni-lj.si

Kristian Šurbek
ks5453@student.uni-lj.si

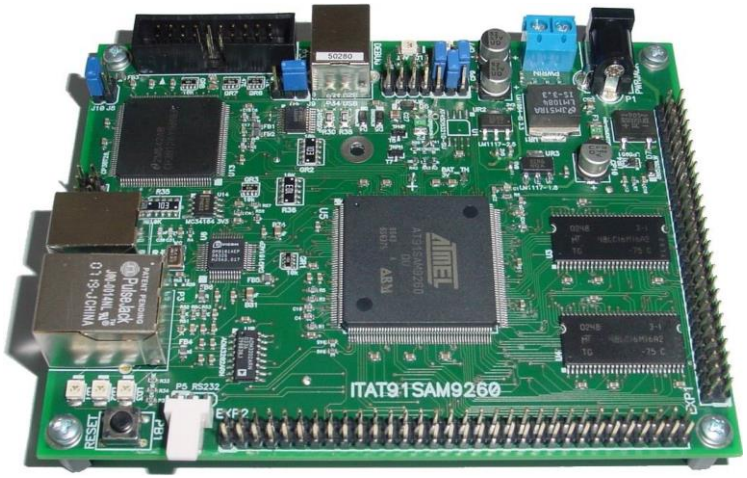
Andrej Sušnik
as1767@student.uni-lj.si

Robert Rozman
rozman@fri.uni-lj.si

Computer architecture CA

Computer STM32H750-DK

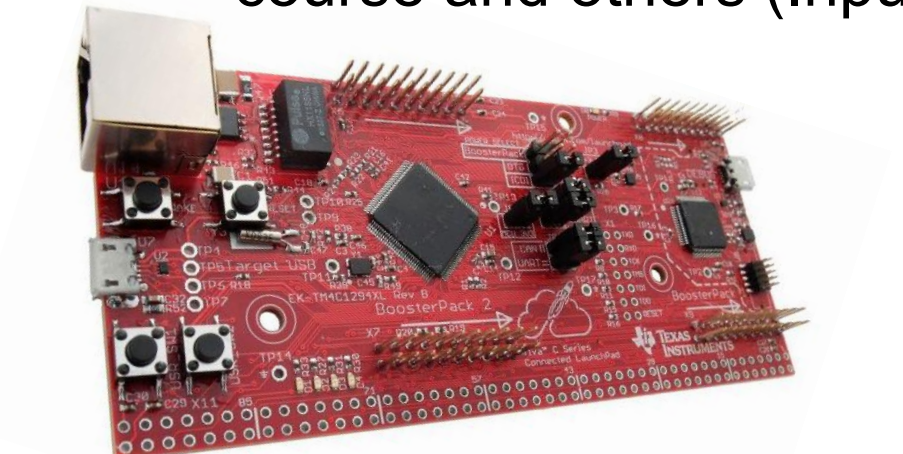
- Računalnik FRI-SMS
  - Mikrokrmilnik AT91SAM9260 iz družine mikrokrmilnikov ARM9



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

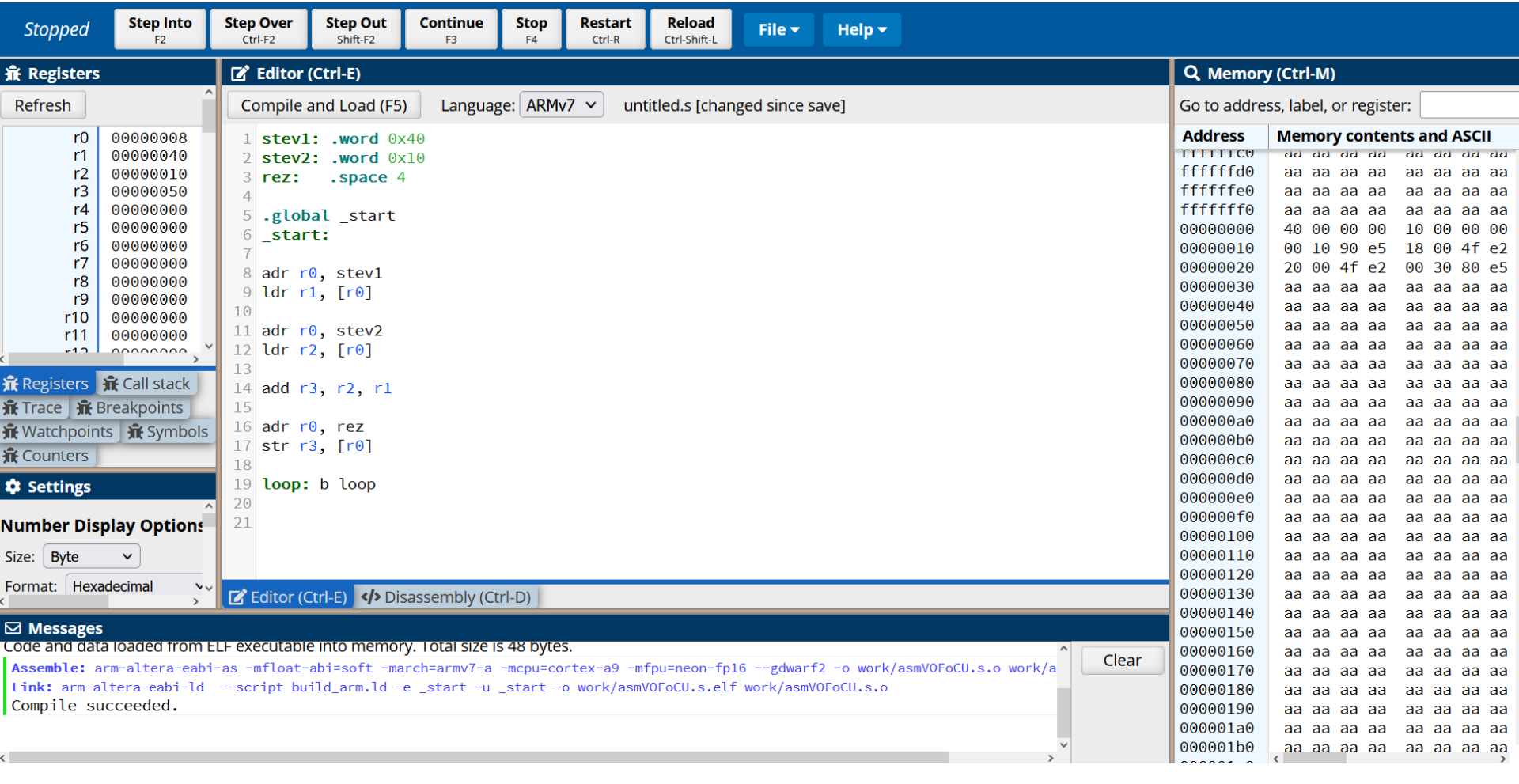
**Due to Covid, can be changed*

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



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

Symbols Project

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: 0x0000000 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

__start

Address	Data	Disassembly
		__start
		ldr r1, stev1
000(	14101	ldr r1,[pc,-0014]
		ldr r2, stev2
000(	14201	ldr r2,[pc,-0014]
		add r3, r2, r1
000(	01308	add r3,r2,r1
		str r3, rez
000(	18300	str r3,[pc,-0018]
		__end: b __end

Registers

R0 00000000
R1 00000000
R2 00000000
R3 00000000
R4 00000000
R5 00000000
R6 00000000
R7 00000000
R8 00000000
R9 00000000

Output

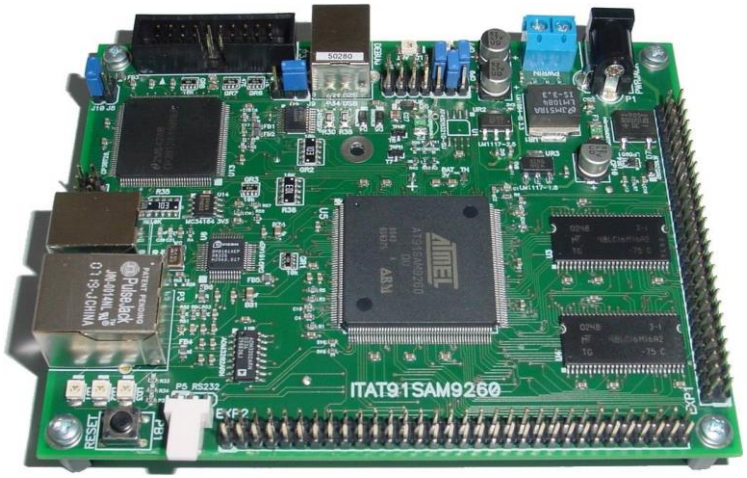
```
Compiling ...
crt0.s
user.s
Linking
"sample.elf (Dir:C:\winIDEA\Projekti\Zgled_2020\Debug\)"  ... was successfully generated.
0 Error(s)  0 Warning(s)
```

Build Find In Files Tools Script

Computer architecture CA

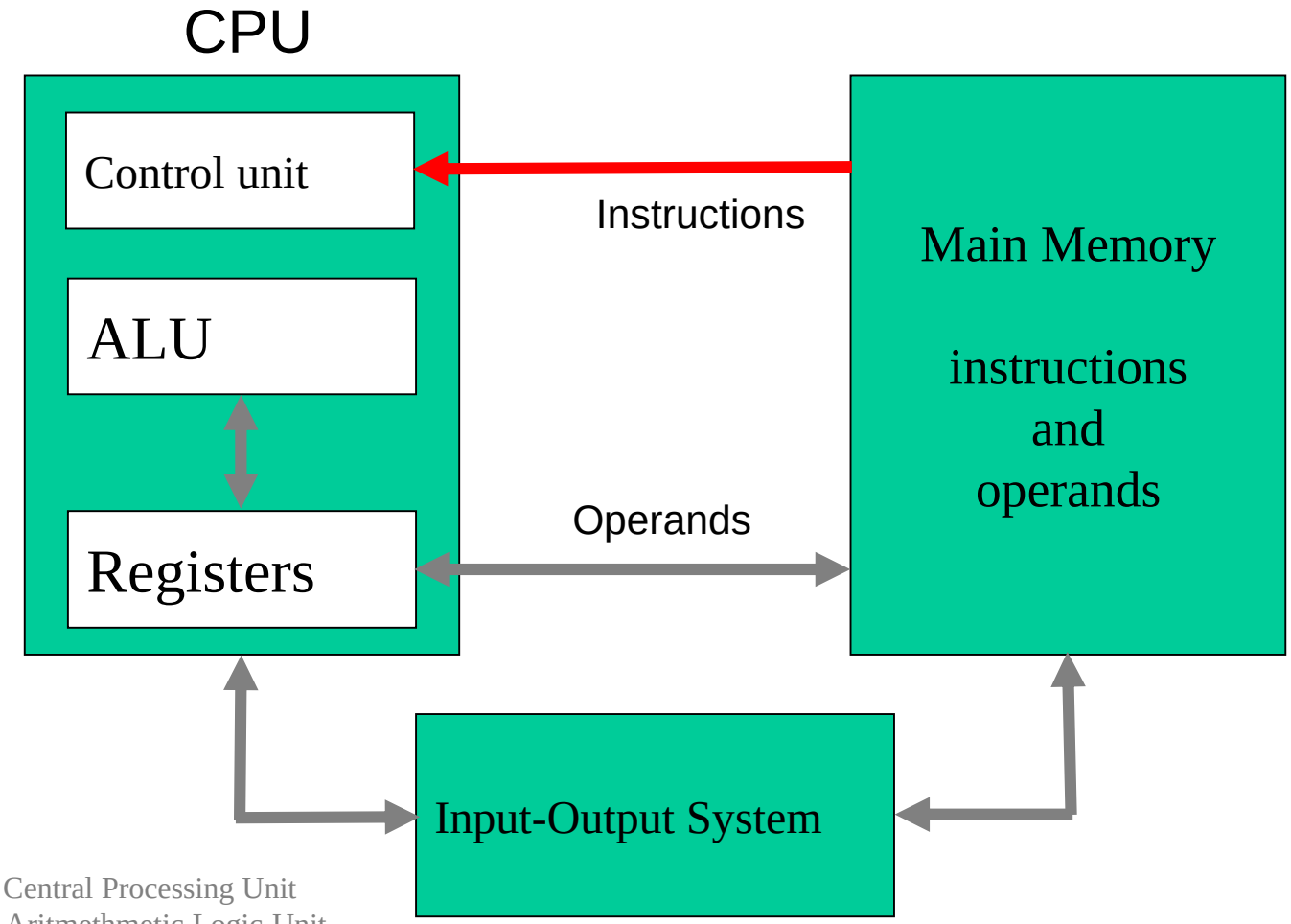
Computer STM32H750-DK

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



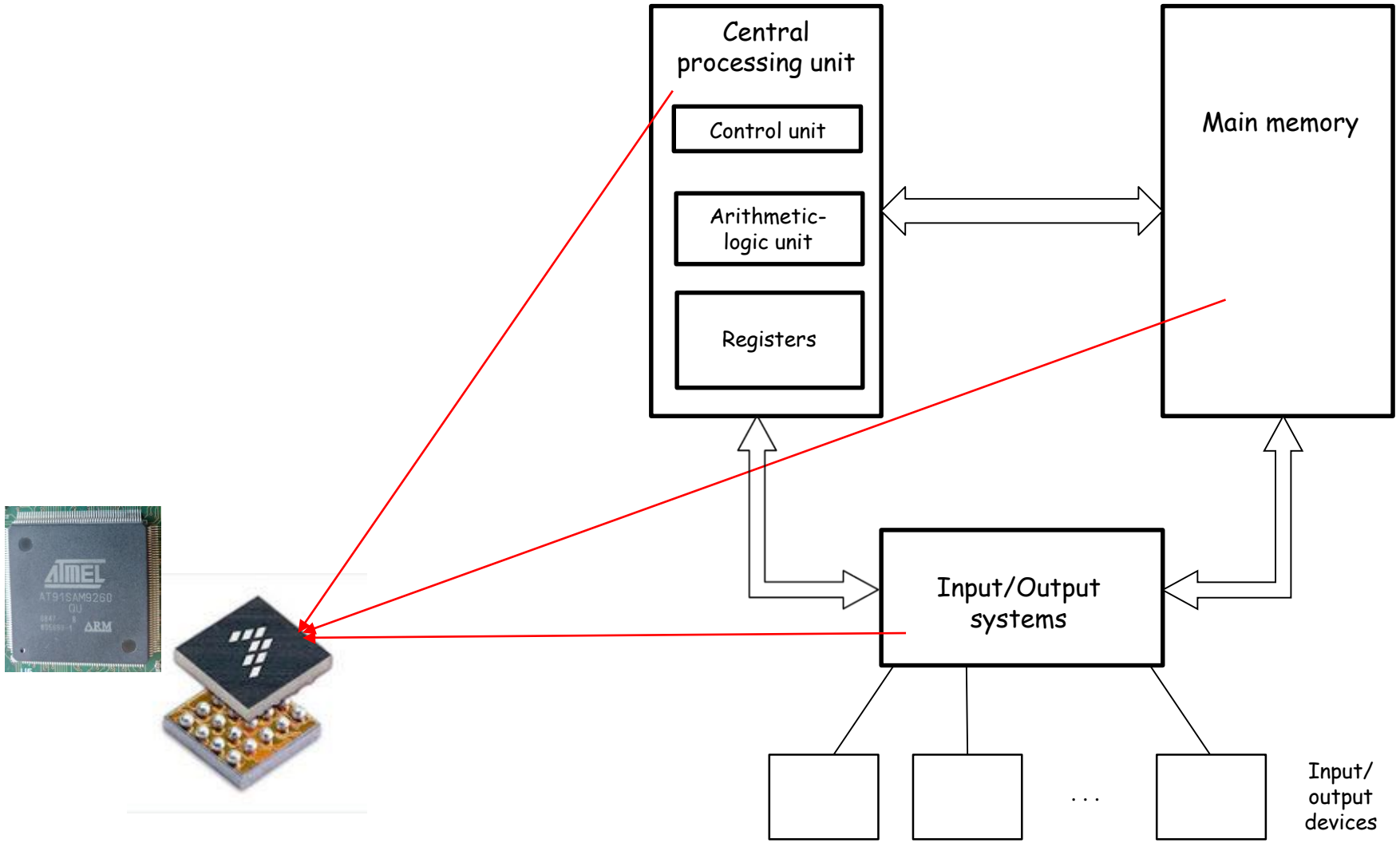
LAB 1.2 Von Neumann model (VN)

Von Neumann Computer Model



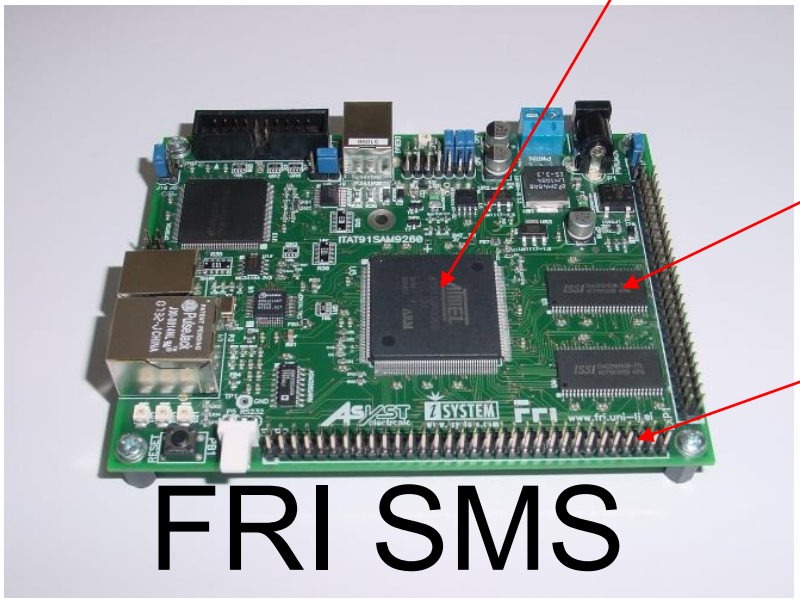
CPU – Central Processing Unit
ALU – Aritmethmetic Logic Unit

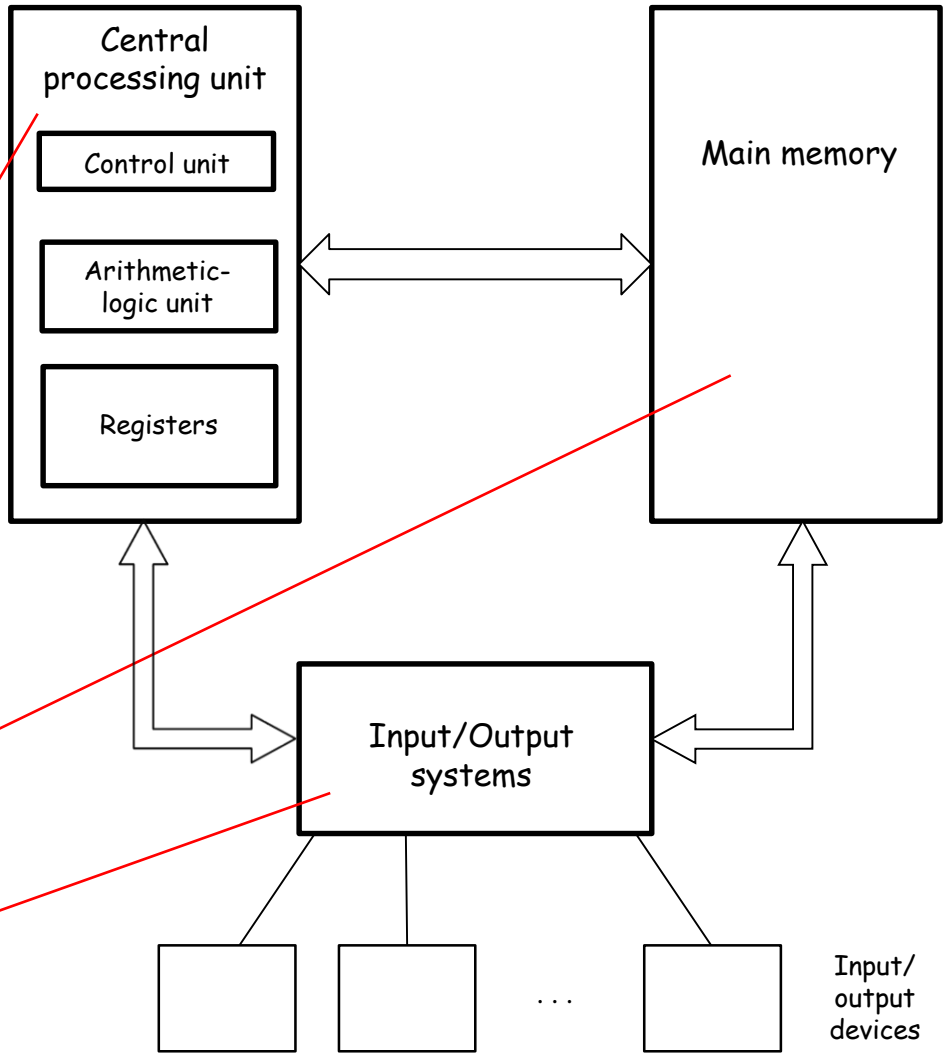
The basic computer model



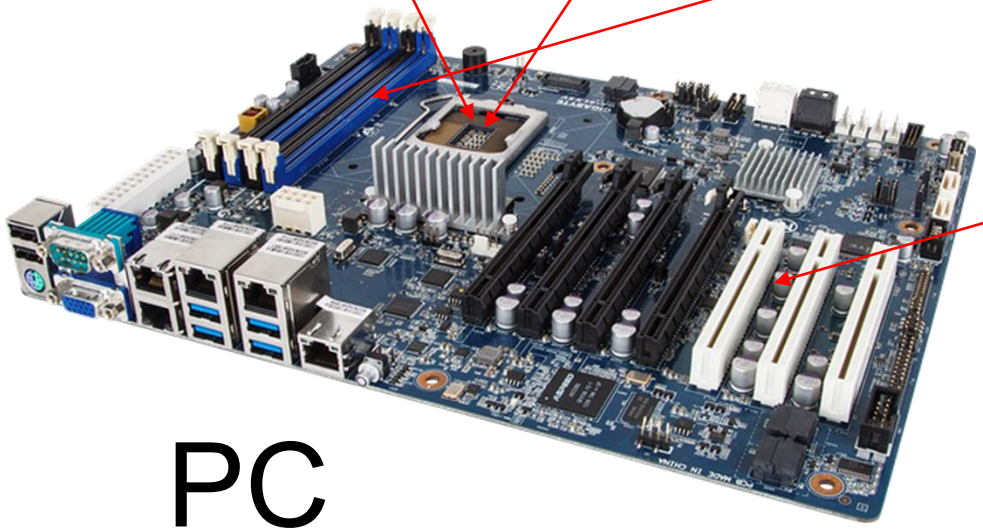
MicroControllers

The basic computer model

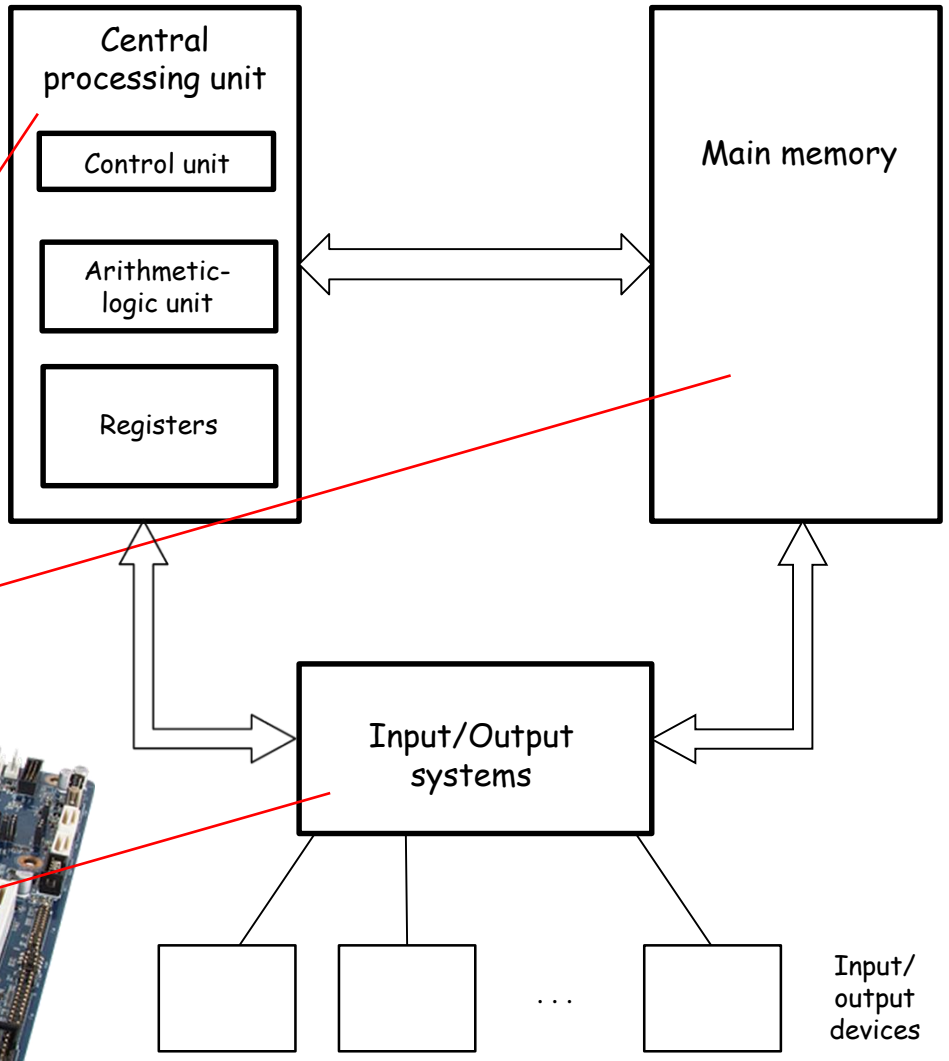




The basic computer model



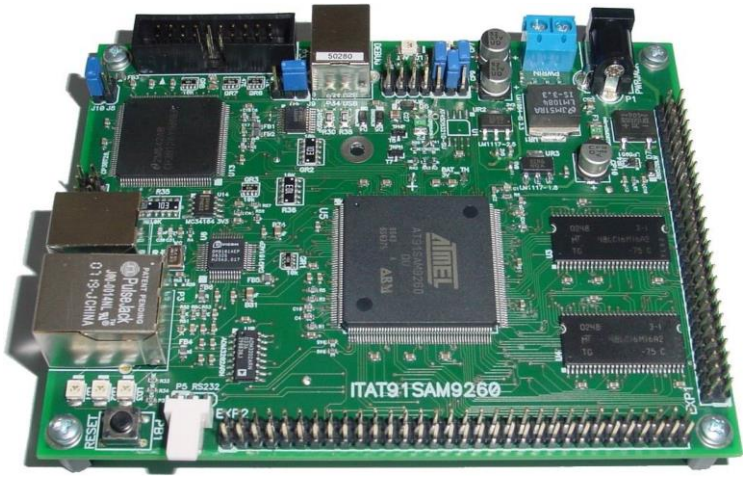
PC



Computer architecture CA

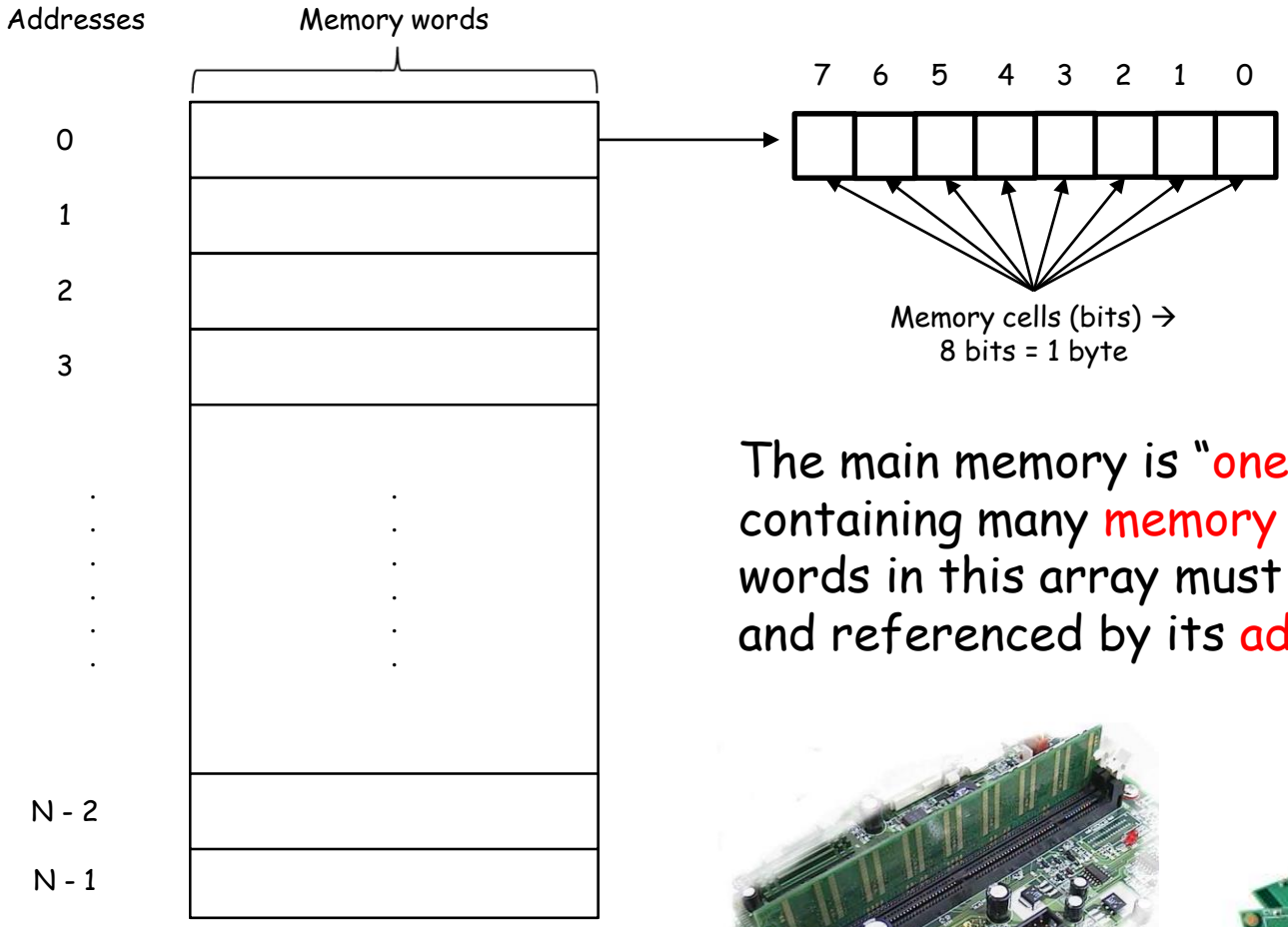
Computer STM32H750-DK

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

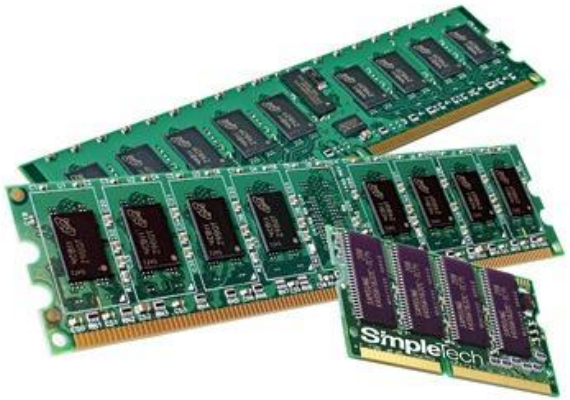


LAB 1.3 Memory

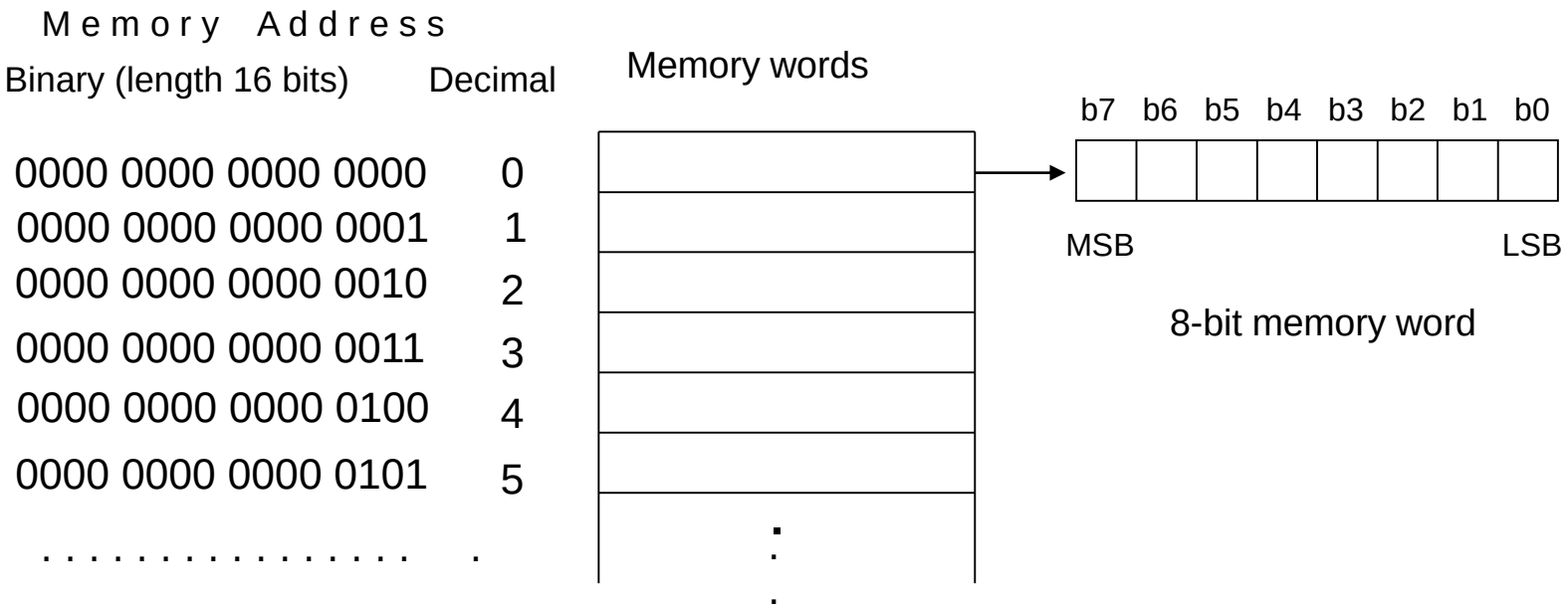
What is memory ?

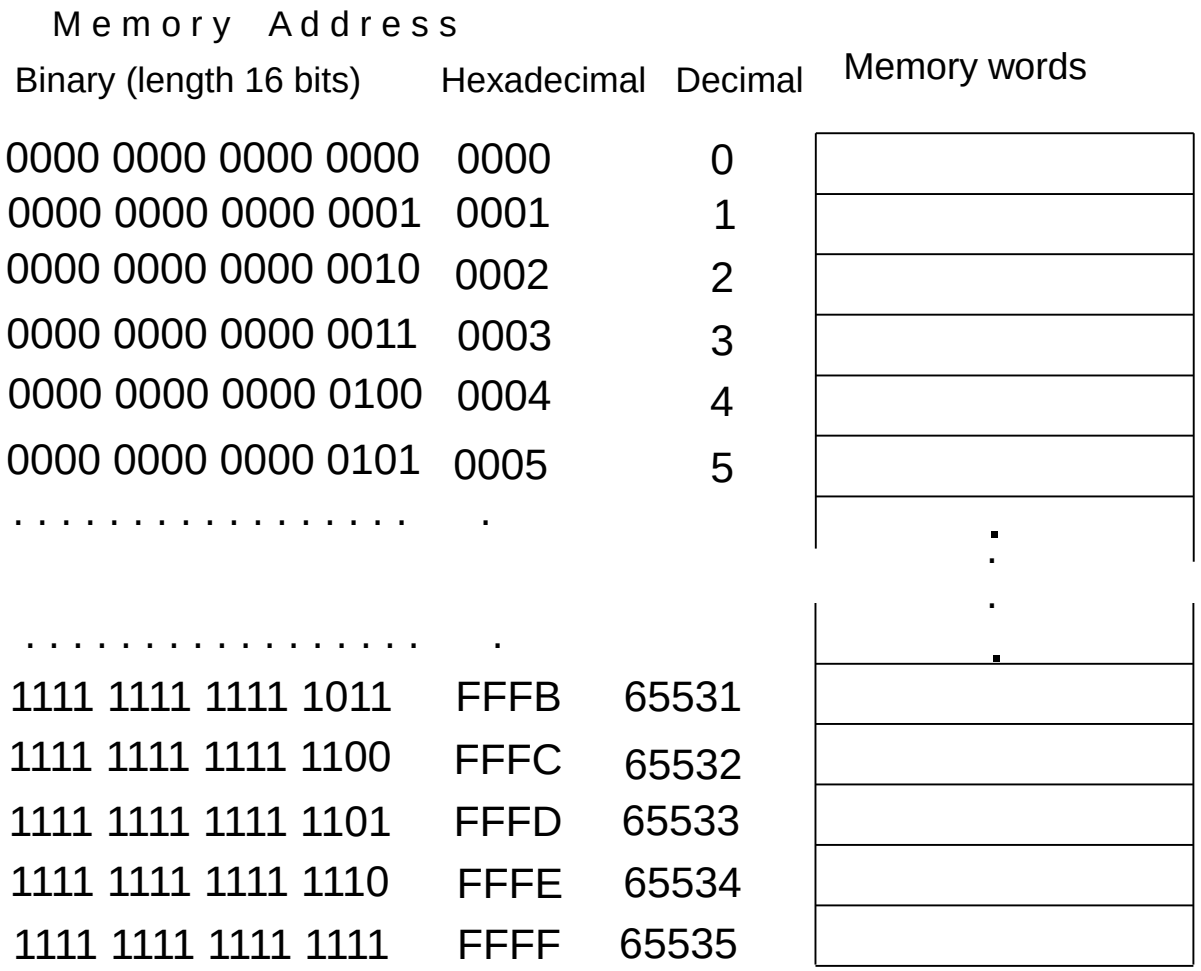


The main memory is "one dimensional array" containing many memory words. Each memory words in this array must be uniquely identified and referenced by its address!



Main Memory in von Neumann Computer

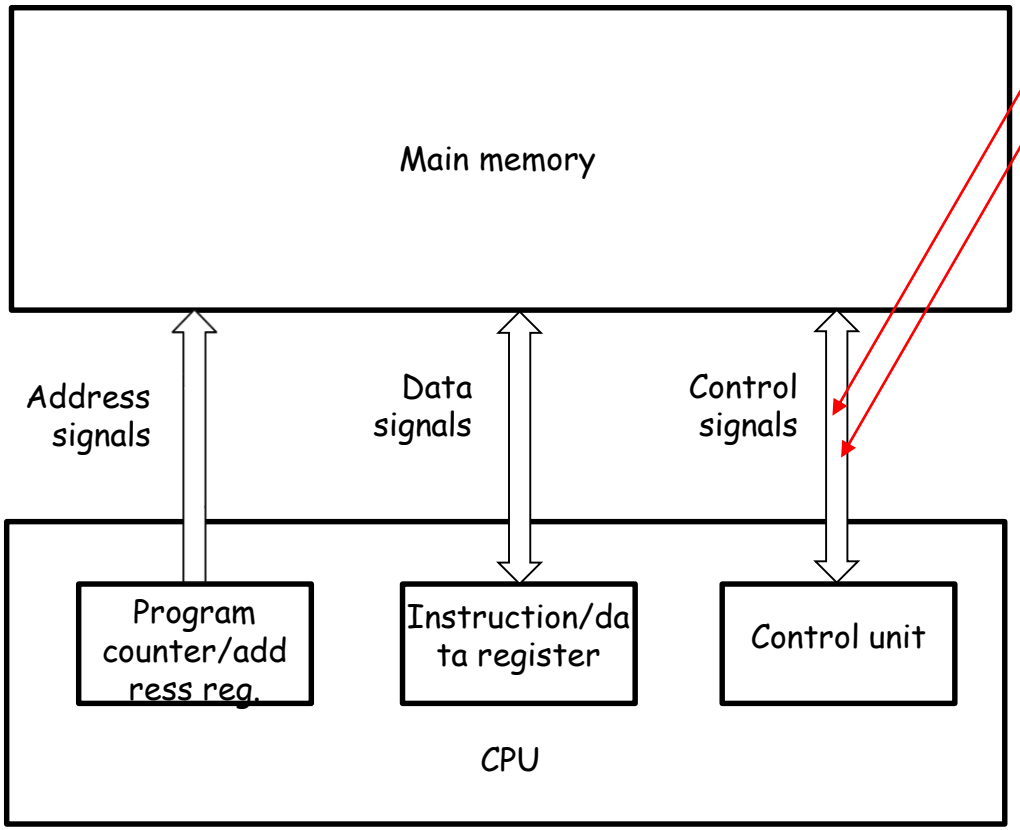



Memory Address
Binary (length 16 bits)
Hexadecimal
Decimal
Memory words
0000 0000 0000 0000 0000 0
0000 0000 0000 0001 0001 1
0000 0000 0000 0010 0002 2
0000 0000 0000 0011 0003 3
0000 0000 0000 0100 0004 4
0000 0000 0000 0101 0005 5
1111 1111 1111 1011 FFFB 65531
1111 1111 1111 1100 FFFC 65532
1111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

Interconnection CPU <-> main memory

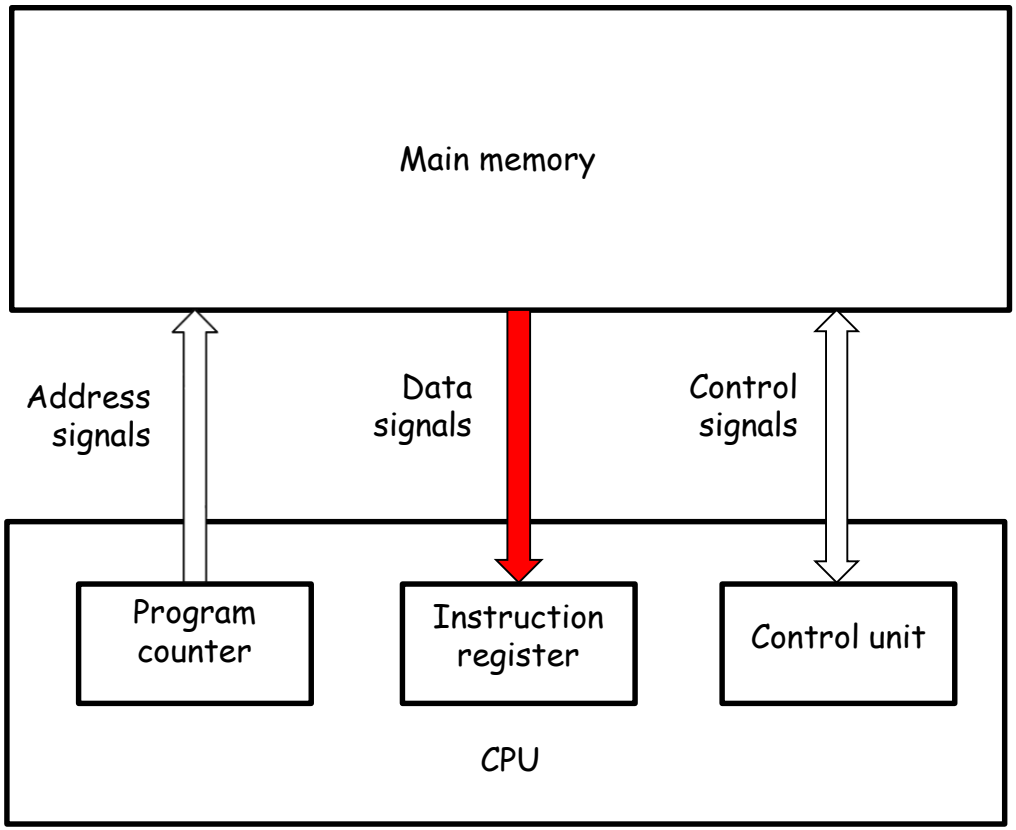
Bus = a group of related lines
(Address, Data, Control buses)

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



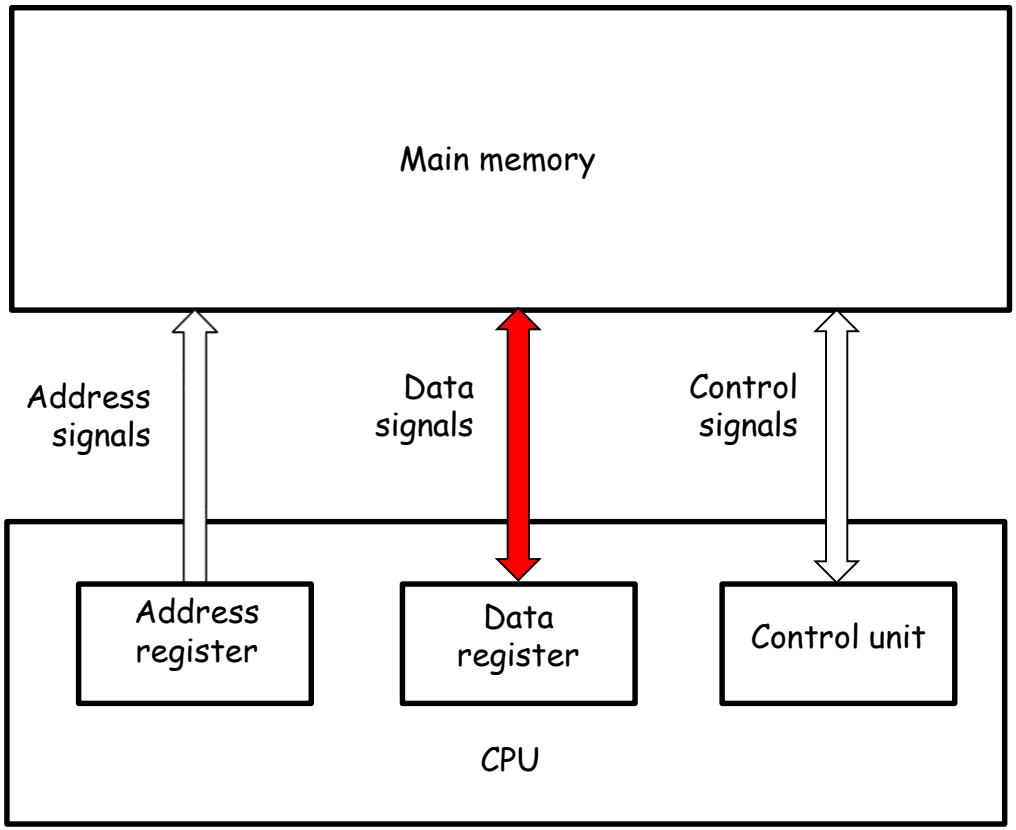
How does CPU access the main memory?

Example for accessing instructions:

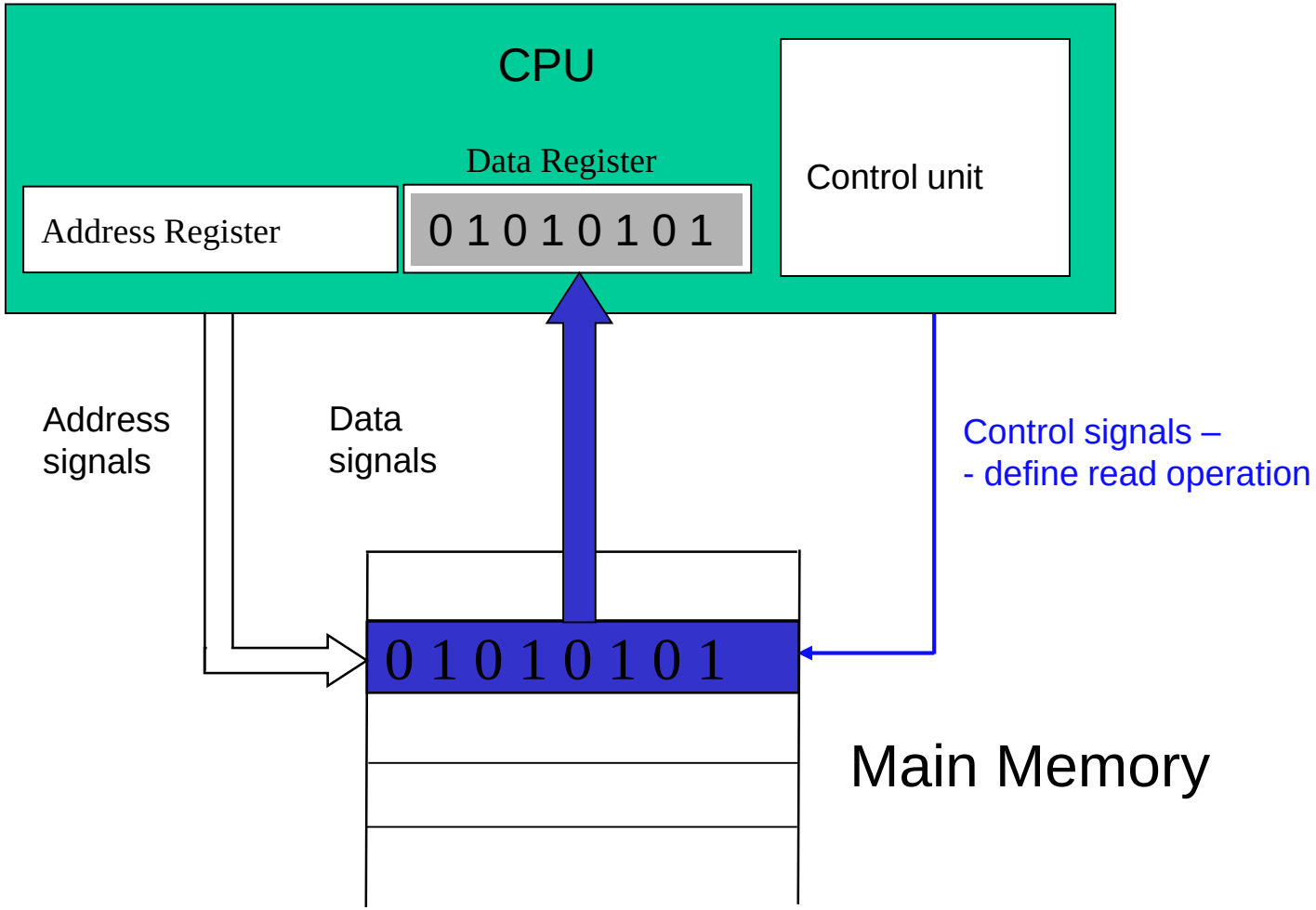


How does CPU access the main memory?

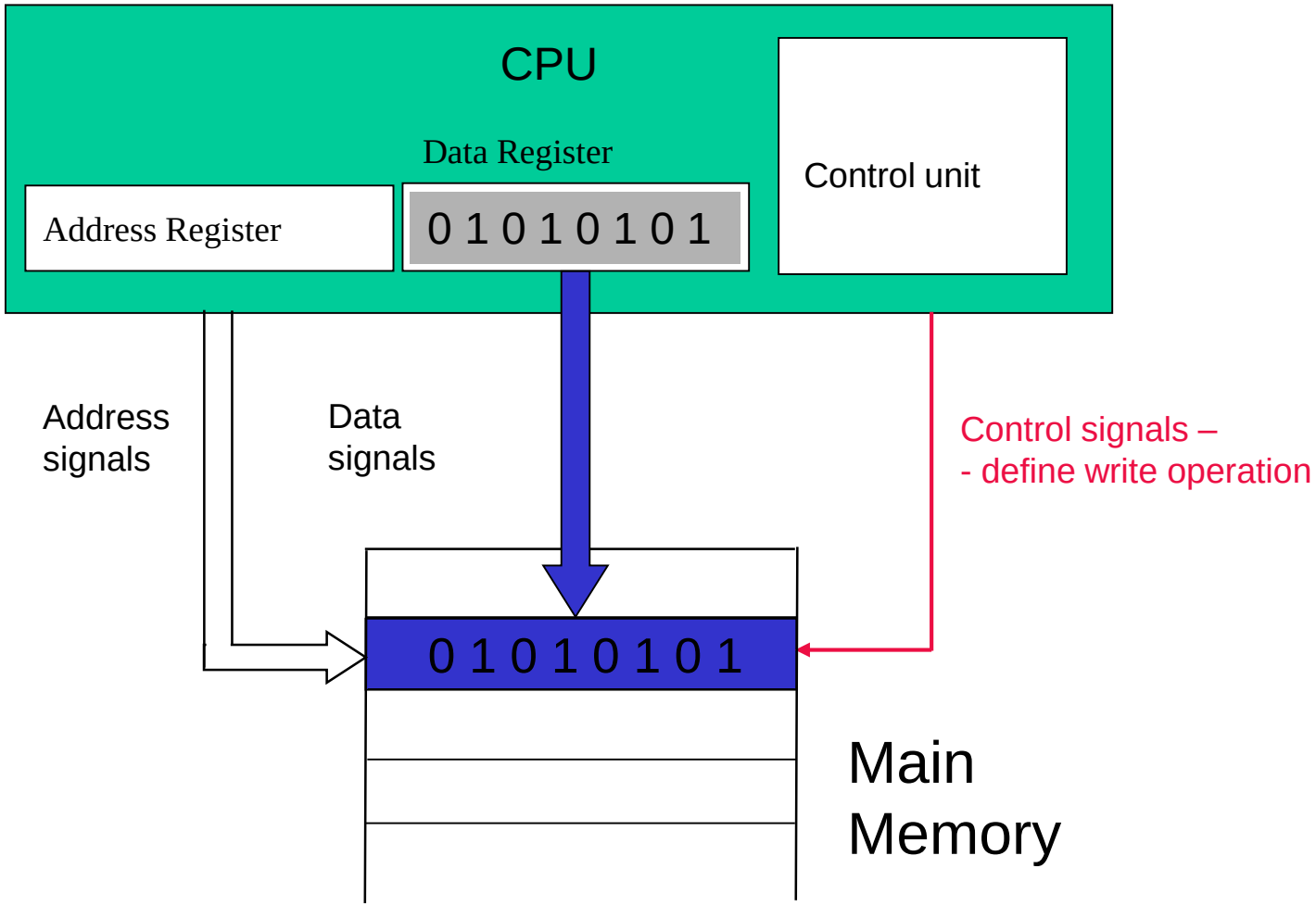
Examples for accessing operands:



CPU and Main Memory – read access



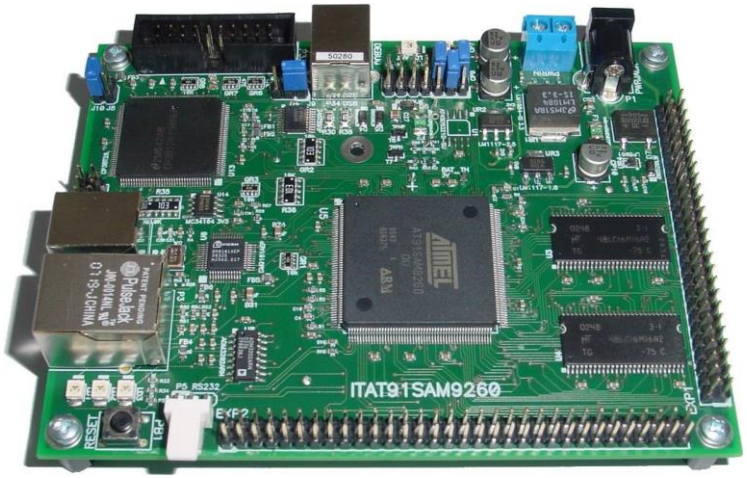
CPU and Main Memory – write access



Computer architecture CA

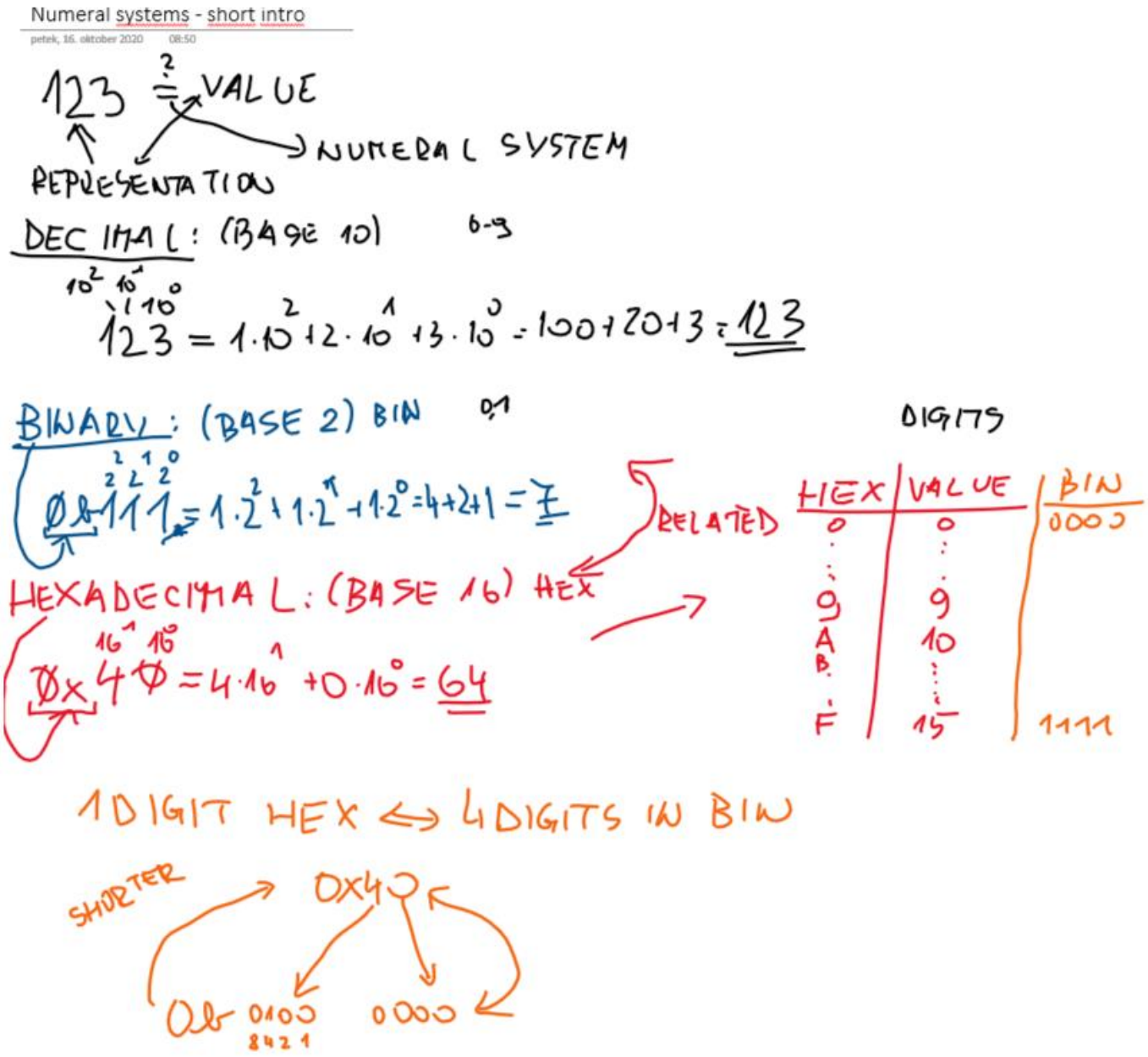
Computer STM32H750-DK

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



LAB 1.4 Quick intro to numeral systems

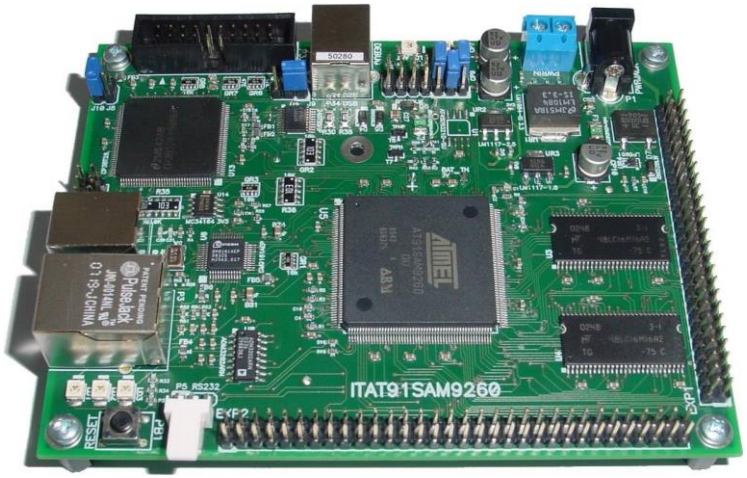
LAB 1.4 Quick intro to numeral systems



Computer architecture CA

Computer STM32H750-DK

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

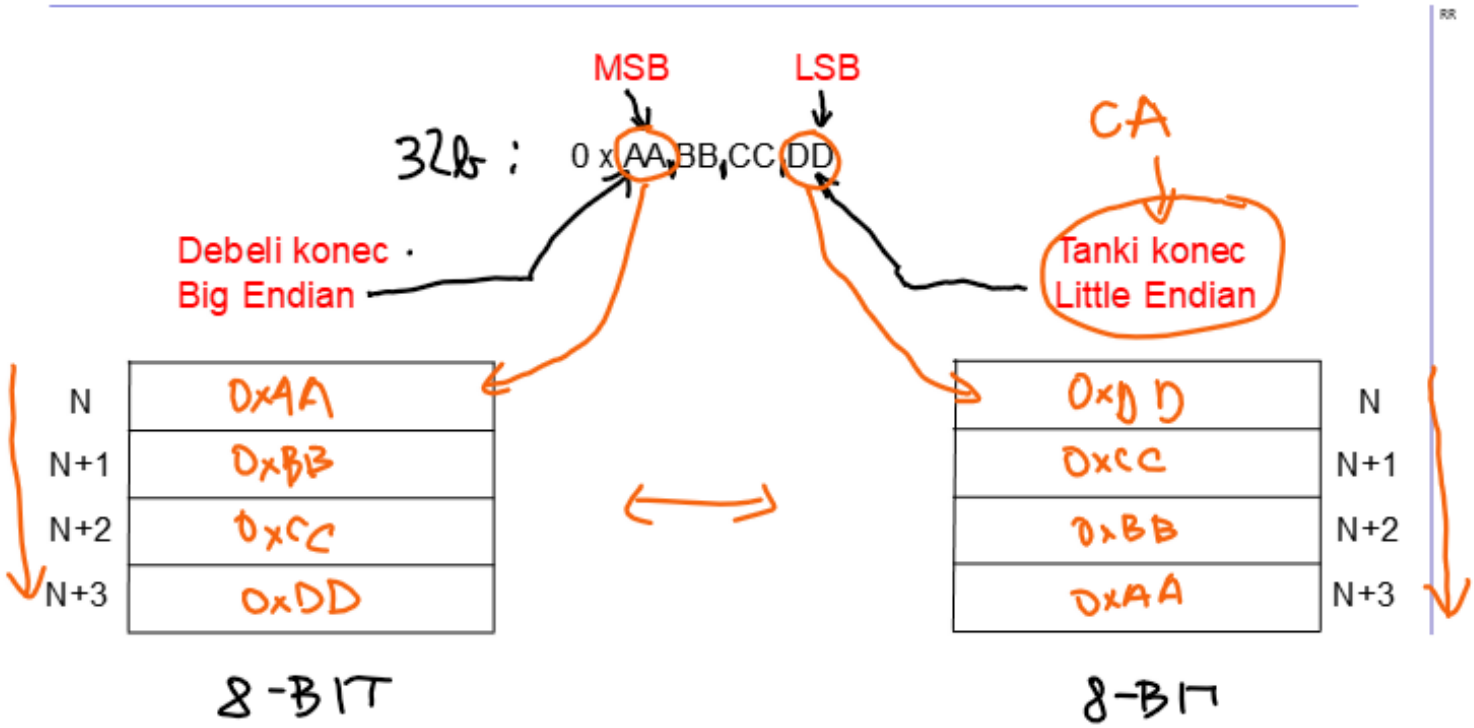


LAB 1.5 Big and Little Endian rules

LAB 1.5 Big and Little Endian rules

Big vs. Little Endian

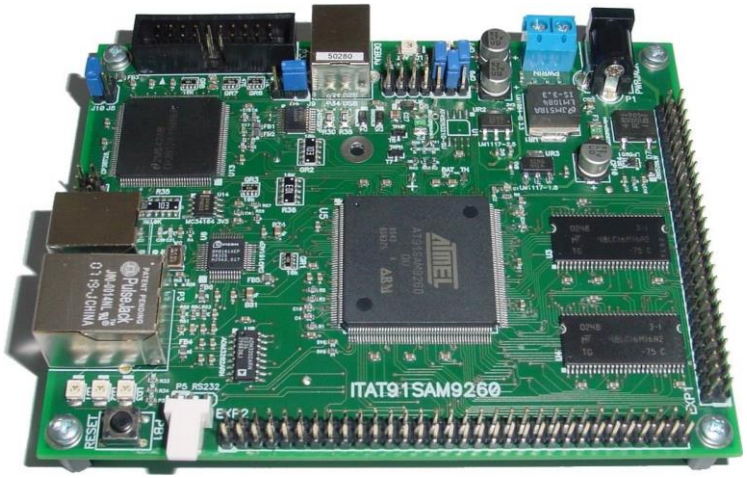
Monday, October 12, 2020 2:18 PM



Computer architecture CA

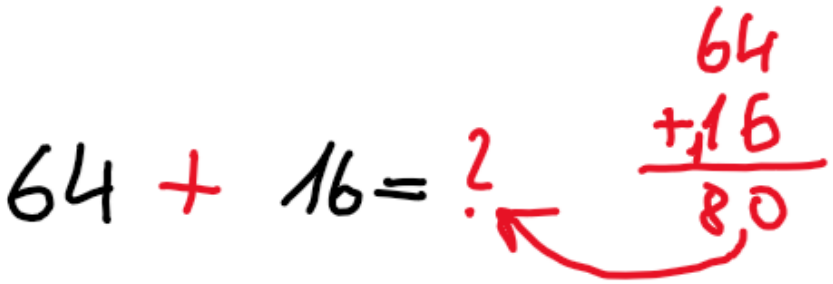
Computer STM32H750-DK

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



LAB 1.6 Addition – human, python, assembler cases

Human (case: 64 + 16 = 80)



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

Global frame

STEV1	64
STEV2	16
REZ	80

Print output (drag lower right corner to resize)

```
 STEV1 = 0x40
+STEV2 = 0x10
-------------
   REZ = 0x50
```

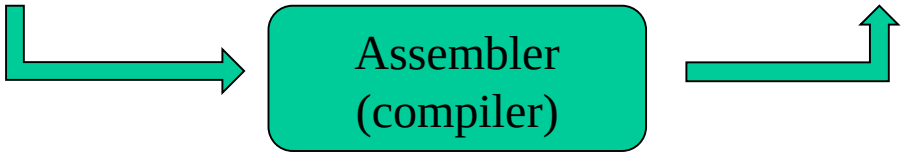
WinIDEA (case: rez = stev1 + stev2)

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 ← 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

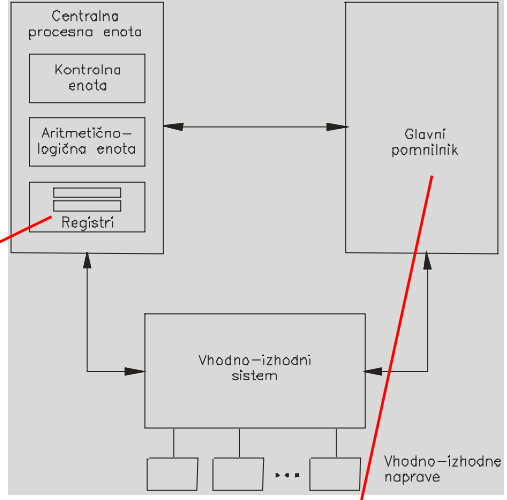


Execute instructions step-by-step and observe the register's values and the variable's values inside the main memory.

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

Refresh

r0	00000008
r1	00000040
r2	00000010
r3	00000050
r4	00000000
r5	00000000
r6	00000000
r7	00000000
r8	00000000
r9	00000000
r10	00000000
r11	00000000

Registers | Call stack | Trace | Breakpoints | Watchpoints | Symbols | Counters

Settings

Number Display Options

Size: Byte

Format: Hexadecimal

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
```

Editor (Ctrl-E) | Disassembly (Ctrl-D)

Memory (Ctrl-M)

Go to address, label, or register:

Address	Memory contents and ASCII
ffffffc0	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

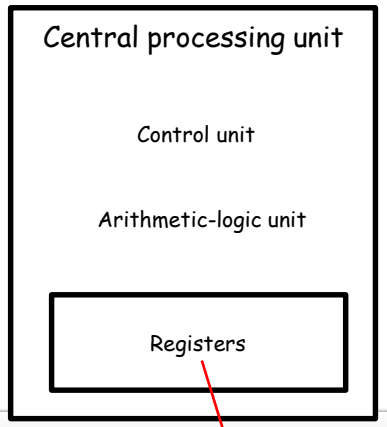
Clear

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/asmVOFoCU.s.o work/a
Link: arm-altera-eabi-ld  --script build_arm.ld -e _start -u _start -o work/asmVOFoCU.s.elf work/asmVOFoCU.s.o
Compile succeeded.
```

Practical example „zgled“

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

Symbols Project

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: 0x0000000 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

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)

Skrito za udeležence

- Instalacija orodja WinIdea - Linux
- Seznam ukazov zbirnika ARM
- ARM sistem FRI-SMS (ploščica)

Skrito za udeležence

- WinIdea: Software User Guide
- Povezava na online Help za WinIdea

Download

Registers

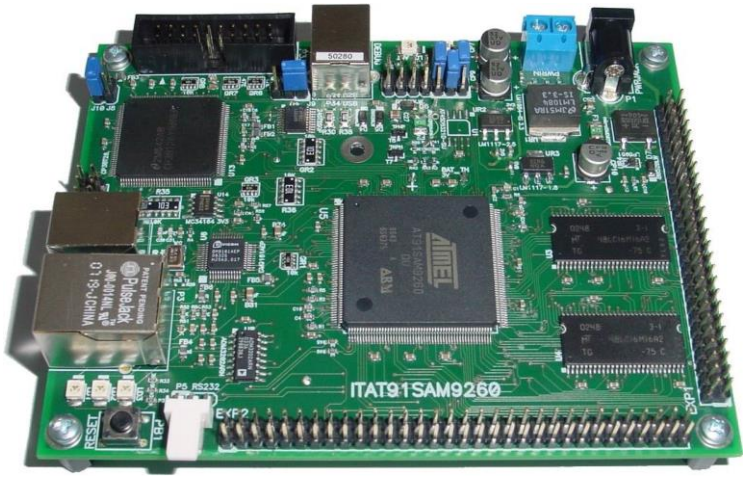
R0	00000000
R1	00000000
R2	00000000
R3	00000000
R4	00000000
R5	00000000
R6	00000000
R7	00000000
R8	00000000
R9	00000000

Build Find In Files Tools Script

Computer architecture CA

Computer STM32H750-DK

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



LAB 1.7 Notes templates

Python (case: REZ = STEV1 + STEV2)

Frames Objects

Global frame

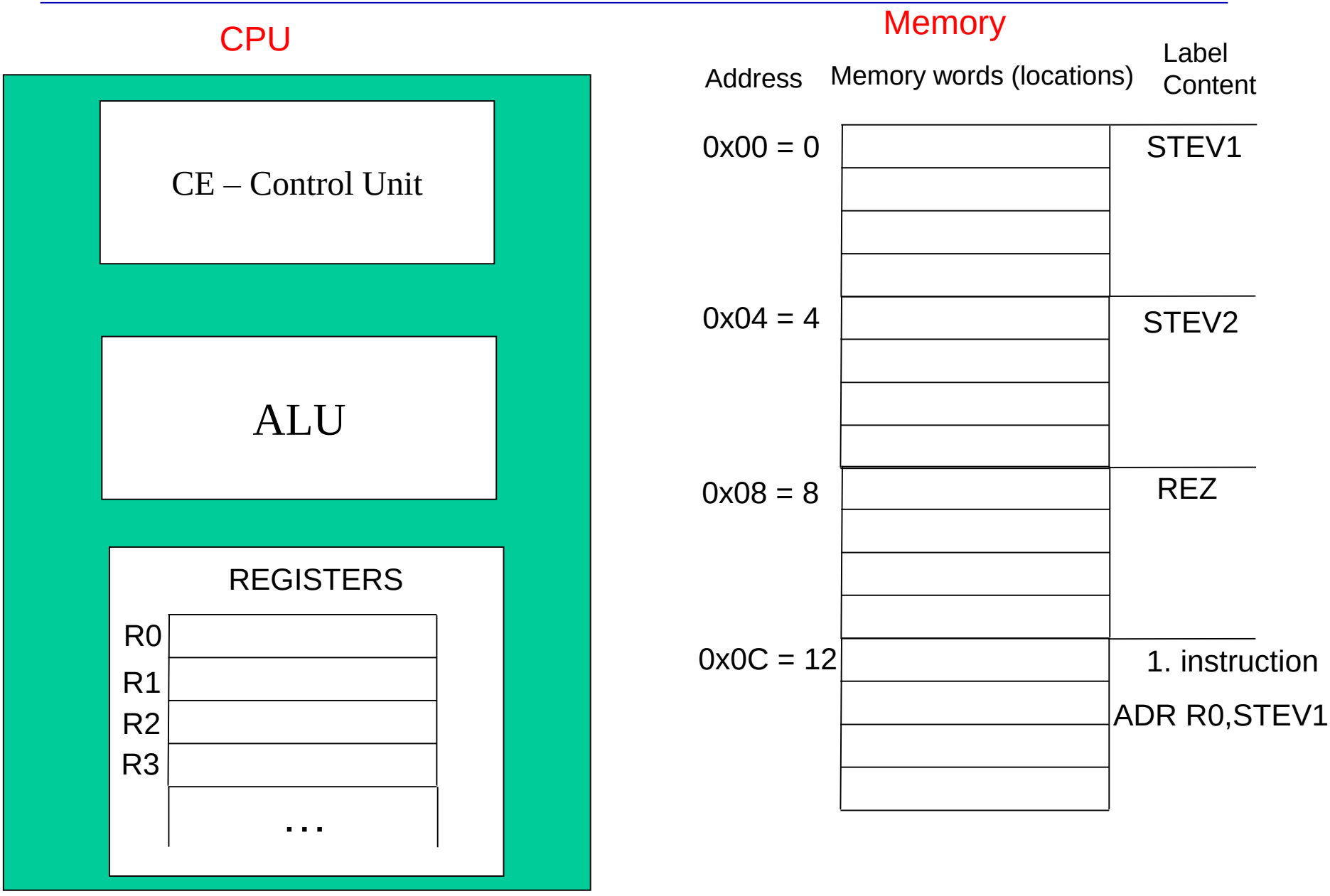
STEV1	64
STEV2	16
REZ	80

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



Zgled: adding two numbers

INSTRUCTIONS

	Machine Instr.	Assembly Instr.	Description	Comment
1.	0xE24F0014	adr r0, stev1	R0 ← Addr. of stev1	
2.	0xE5901000	ldr r1, [r0]	R1 ← M[R0]	
3.	0xE24F0018	adr r0, stev2	R0 ← Addr. of stev2	
4.	0xE5902000	ldr r2, [r0]	R2 ← M[R0]	
5.	0xE0823001	add r3, r2, r1	R3 ← R1 + R2	
6.	0xE24F0020	adr r0, rez	R0 ← Addr. of rez	
7.	0xE5803000	str r3, [r0]	M[R0] ← 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