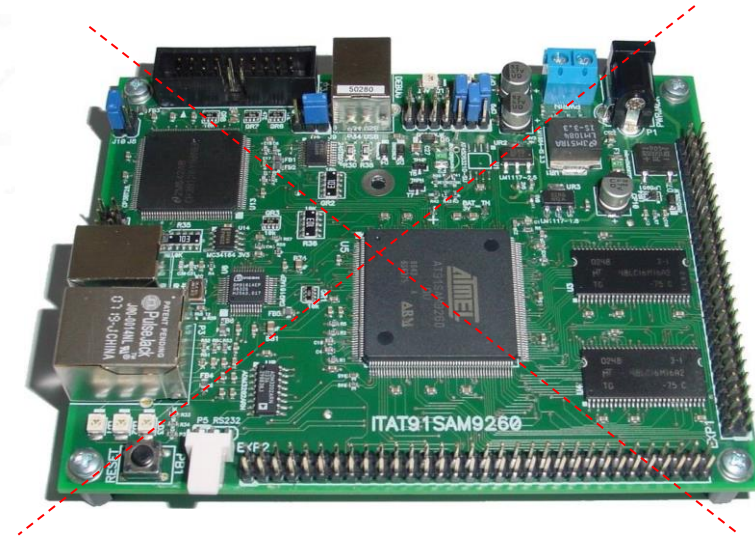


Computer architecture CA

Computer STM32H750-DK



- Računalnik FRI-SMS
 - Mikrokmlnik AT91SAM9260 iz družine mikrokmlnikov ARM9

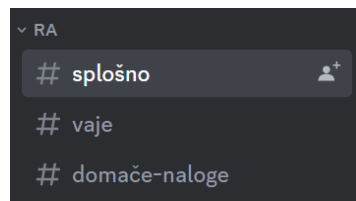
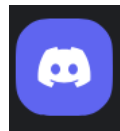


Team CA

Tutors



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



<https://discord.com/channels/110837568099713027/11156262933416902757>



Andrej Sušnik
as1767@student.uni-lj.si



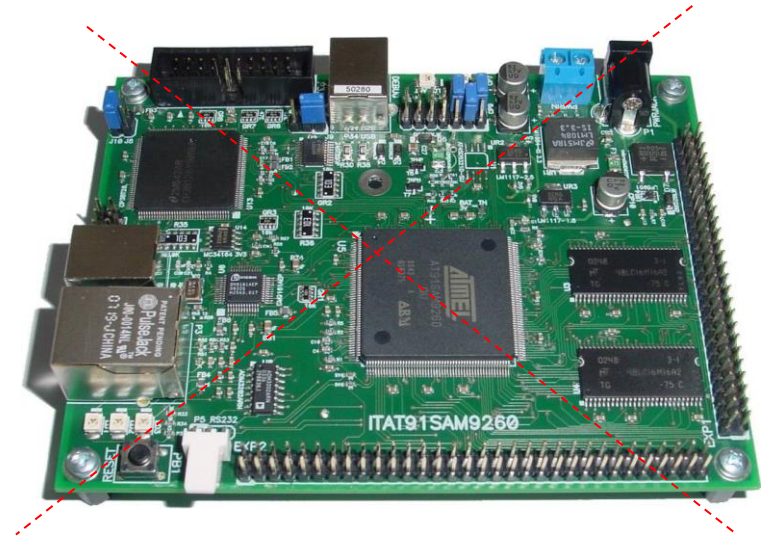
Robert Rozman
rozman@fri.uni-lj.si

Computer architecture CA

Computer STM32H750-DK



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

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>
- 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".

The interface is divided into several panels:

- Registers:** A table showing the state of various registers. The PC register is highlighted in red and contains the value 00000048.
- Editor (Ctrl-E):** A code editor showing assembly code for ARMv7. The code includes directives like `.text`, `.org 0x20`, `@spremenljivke`, `stev1: .word 0x40`, `stev2: .word 0x10`, `rez: .space 4`, `.align`, `.global _start`, `_start:`, `@program`, `adr r0, stev1`, `ldr r1, [r0]`, `adr r0, stev2`, `ldr r2, [r0]`, `add r3, r2, r1`, `adr r0, rez`, `str r3, [r0]`, and `end: b end`.
- Memory (Ctrl-M):** A table showing memory contents and ASCII values. The address 00000048 is highlighted, and the memory content is 20 00 4f e2 00 30 80 e5.
- Messages:** A log showing the compilation process. The messages are: "Compiling...", "Code and data loaded from ELF executable into memory. Total size is 80 bytes.", "Assemble: arm-altera-eabi-as -mfloat-abi=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.s.elf work/asmhSiYoH.s.o", and "Compile succeeded."

Computer architecture CA

Computer STM32H750-DK



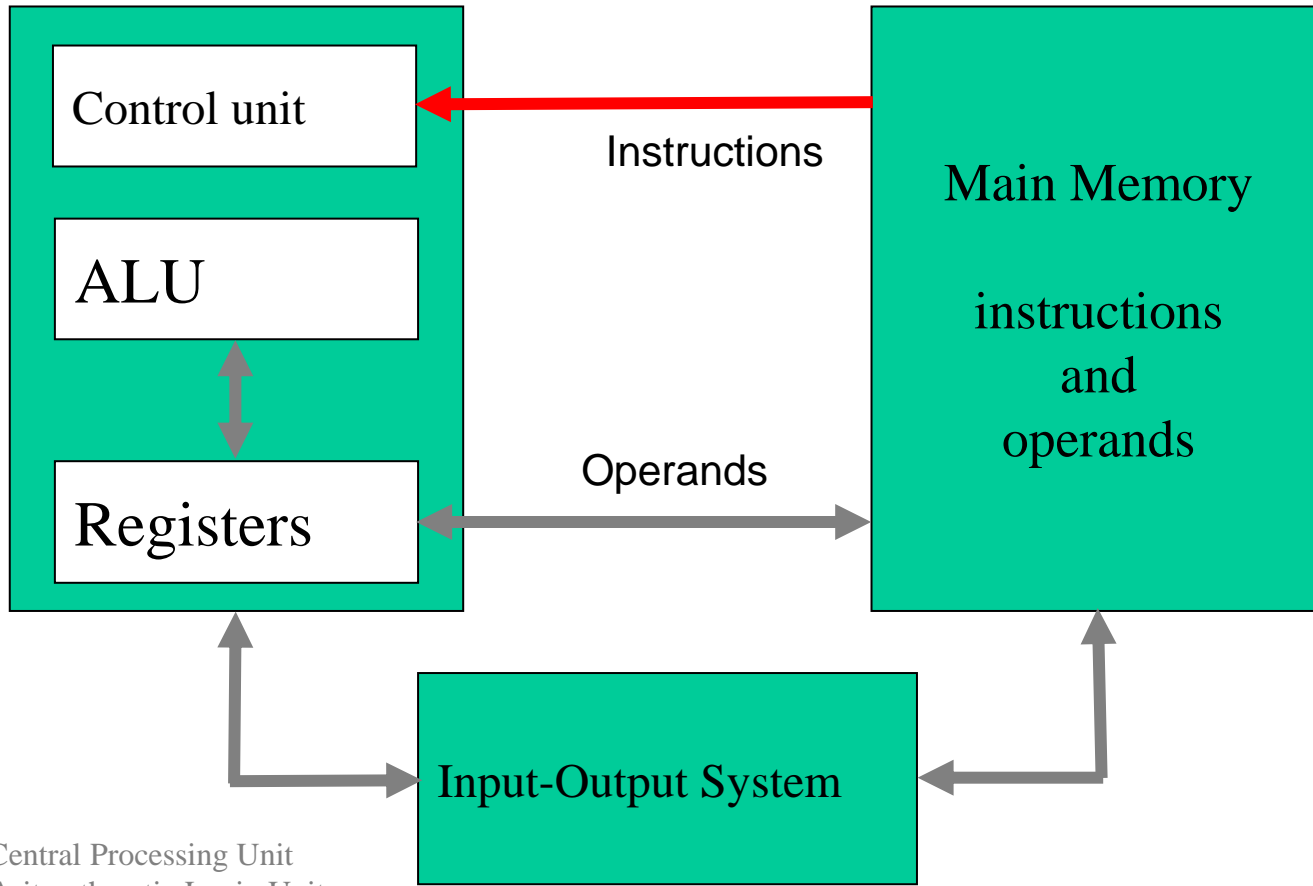
- Računalnik FRI-SMS
 - Mikrokontrolnik AT91SAM9260 iz družine mikrokontrolnikov ARM9



LAB 1.2 Von Neumann model (VN)

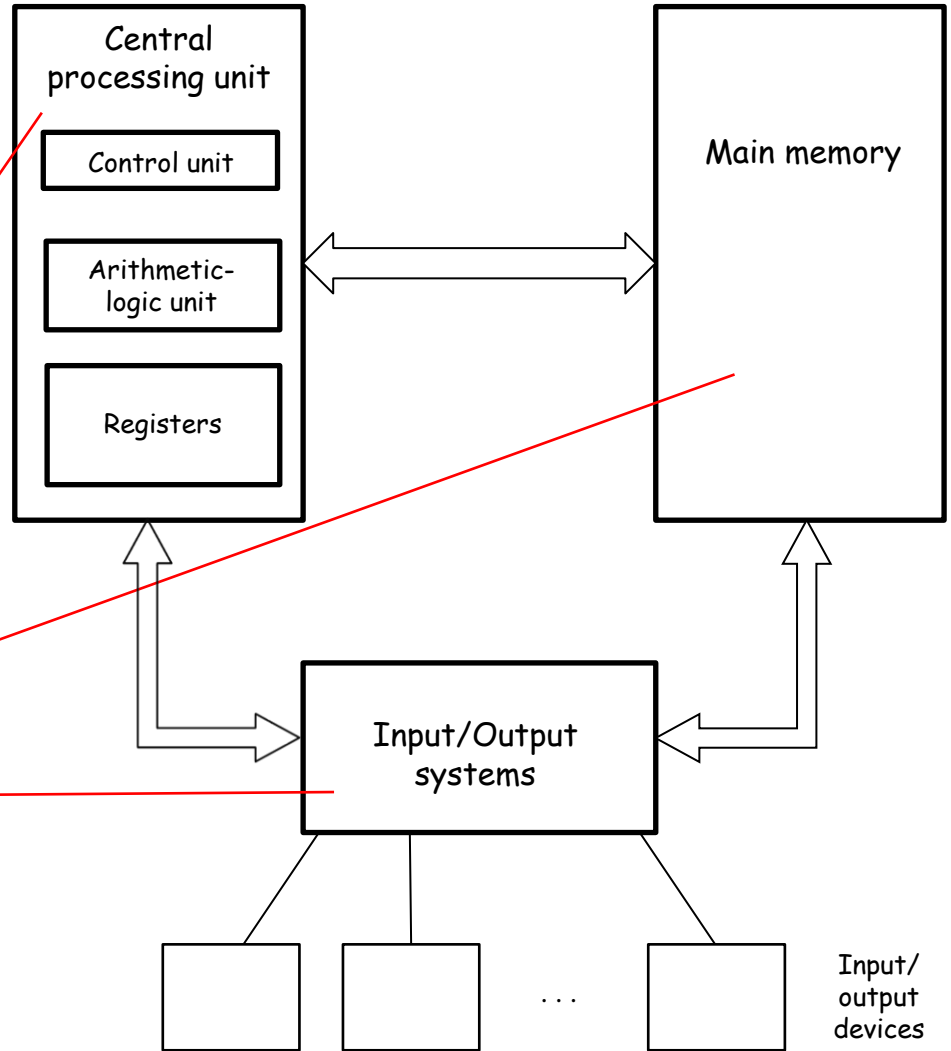
Von Neumann Computer Model

CPU



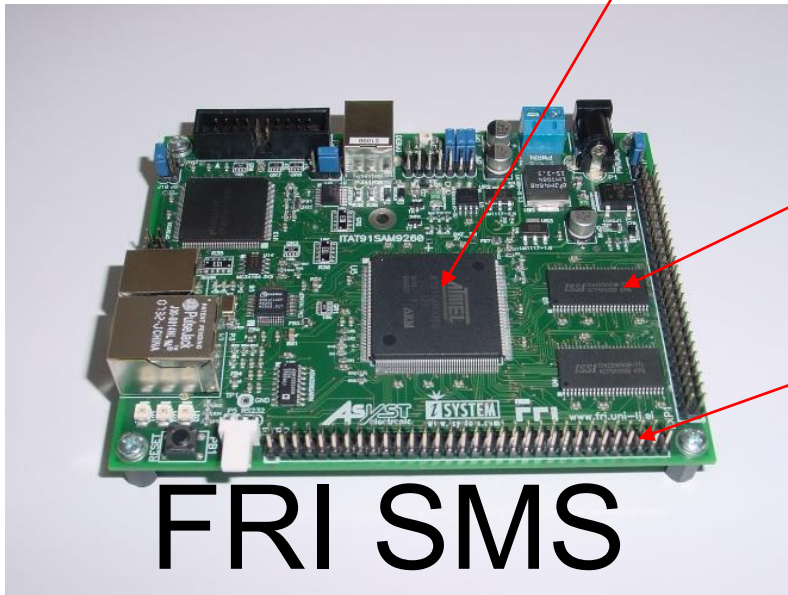
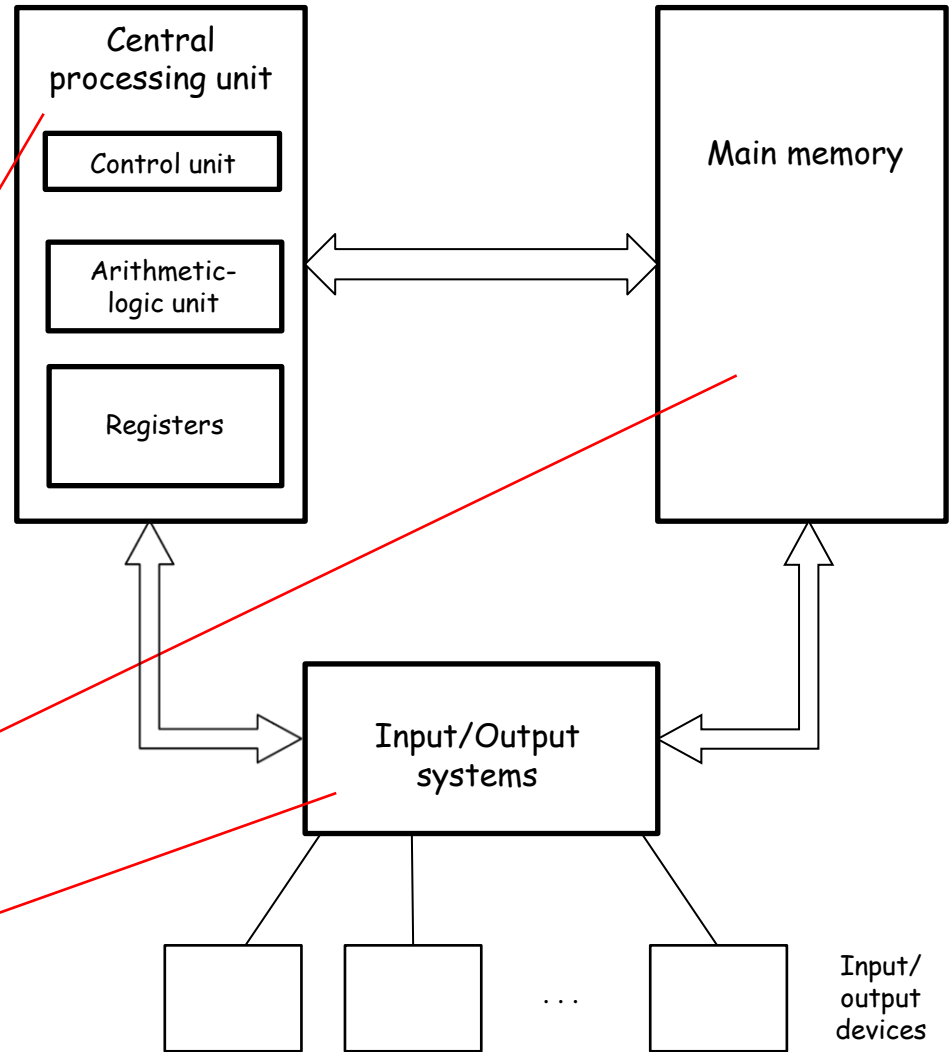
CPU – Central Processing Unit
ALU – Arithmetic Logic Unit

The basic computer model

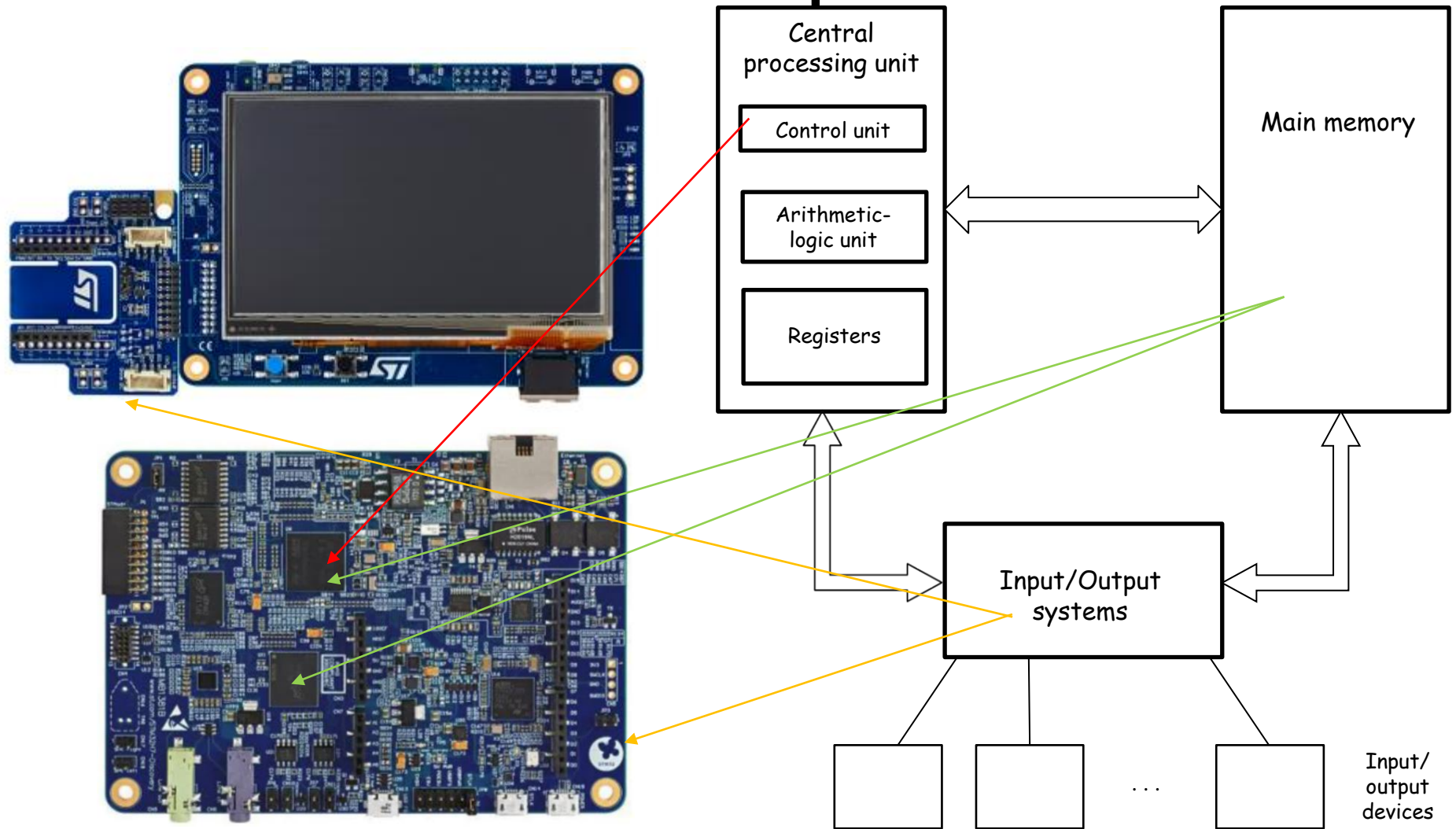


MicroControllers

The basic computer model

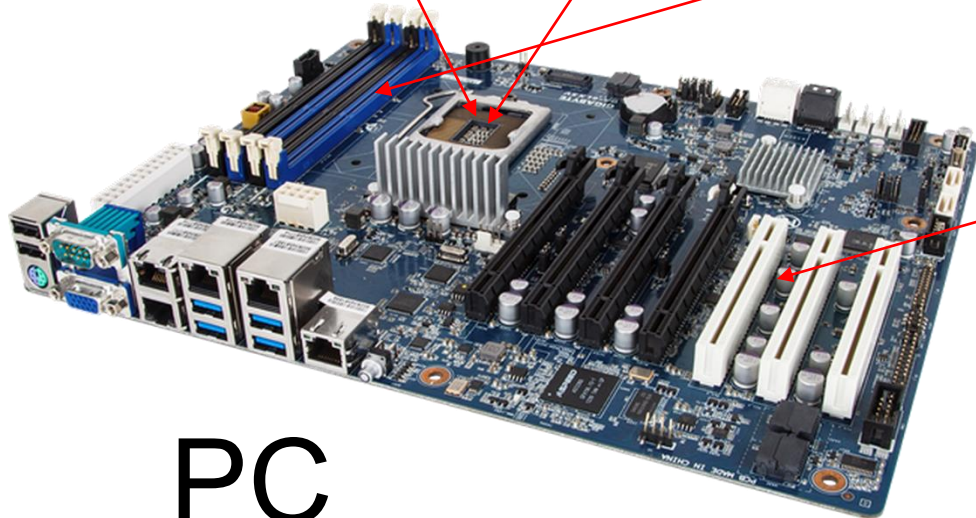


The basic computer model

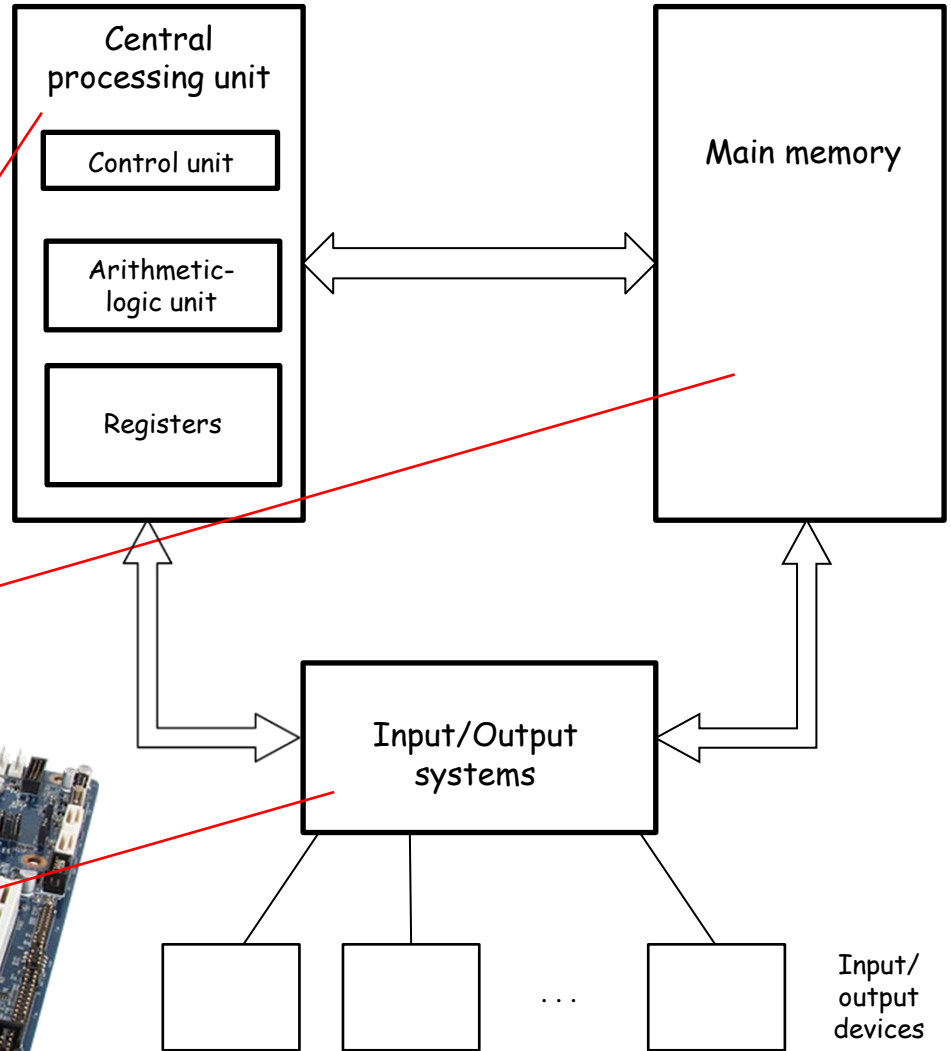


STM32H750-DK

The basic computer model



PC

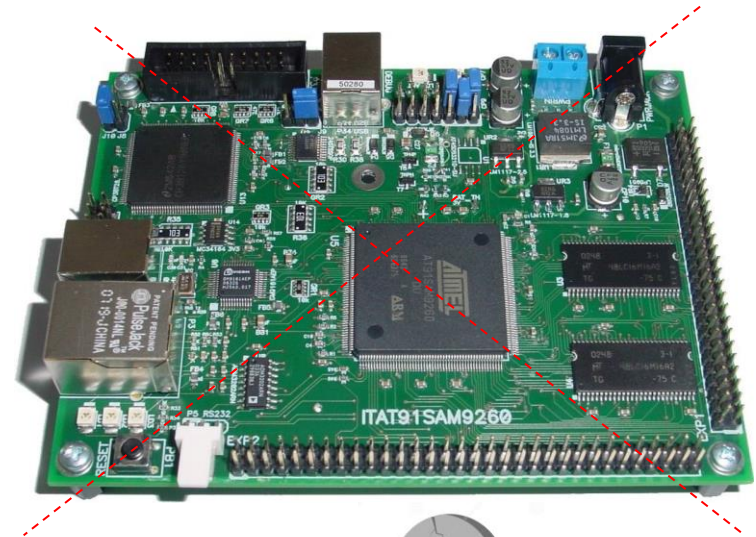


Computer architecture CA

Computer STM32H750-DK

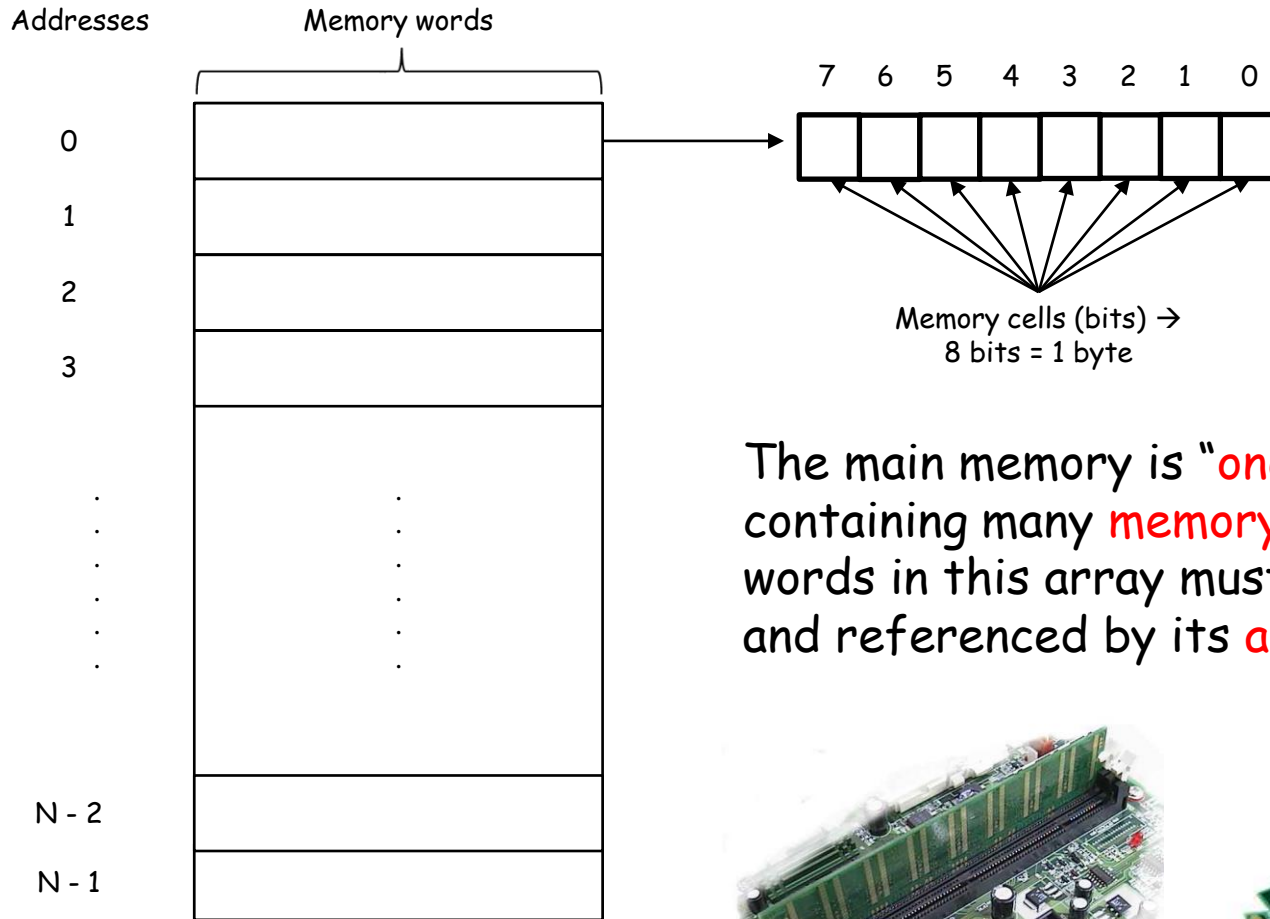


- Računalnik FRI-SMS
 - Mikrokrmilnik AT91SAM9260 iz družine mikrokrmilnikov ARM9

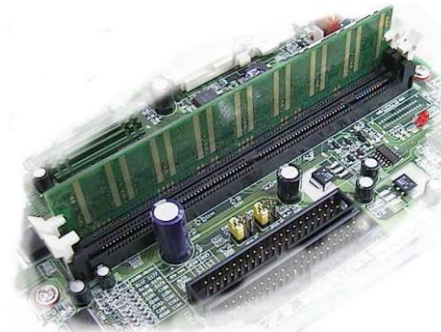


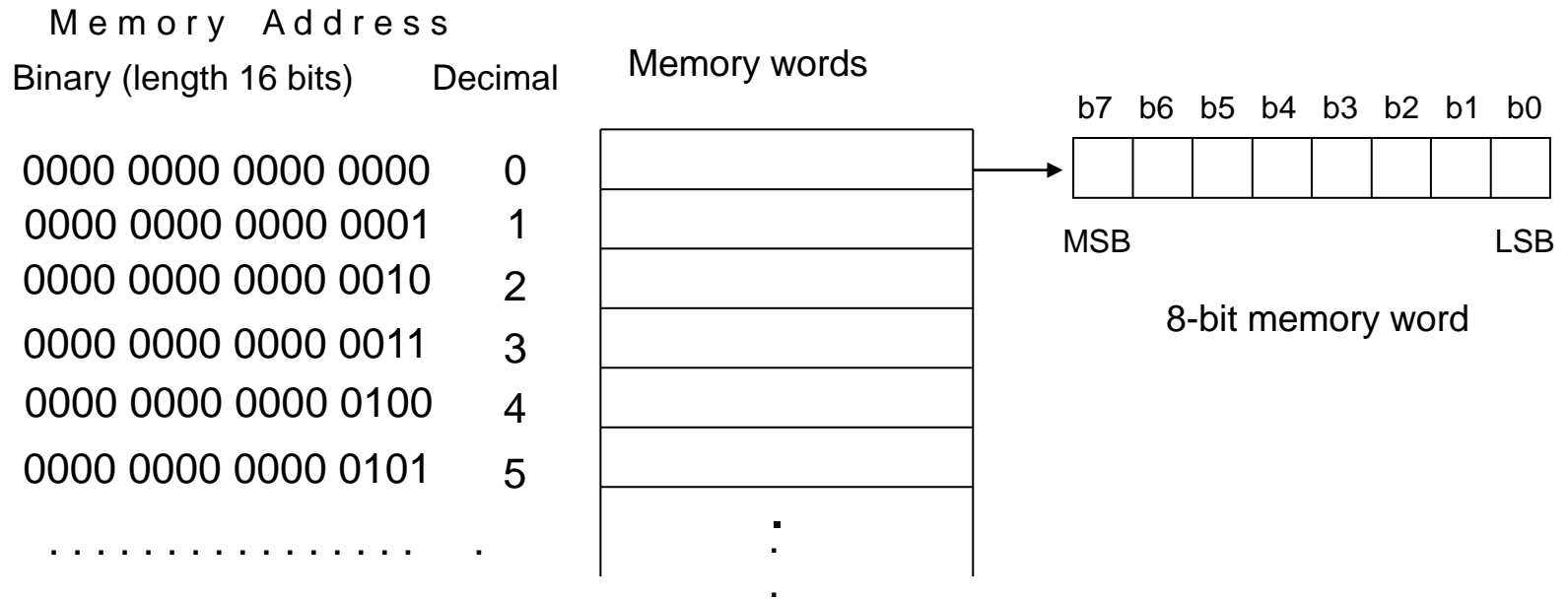
LAB 1.3 Memory

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**!



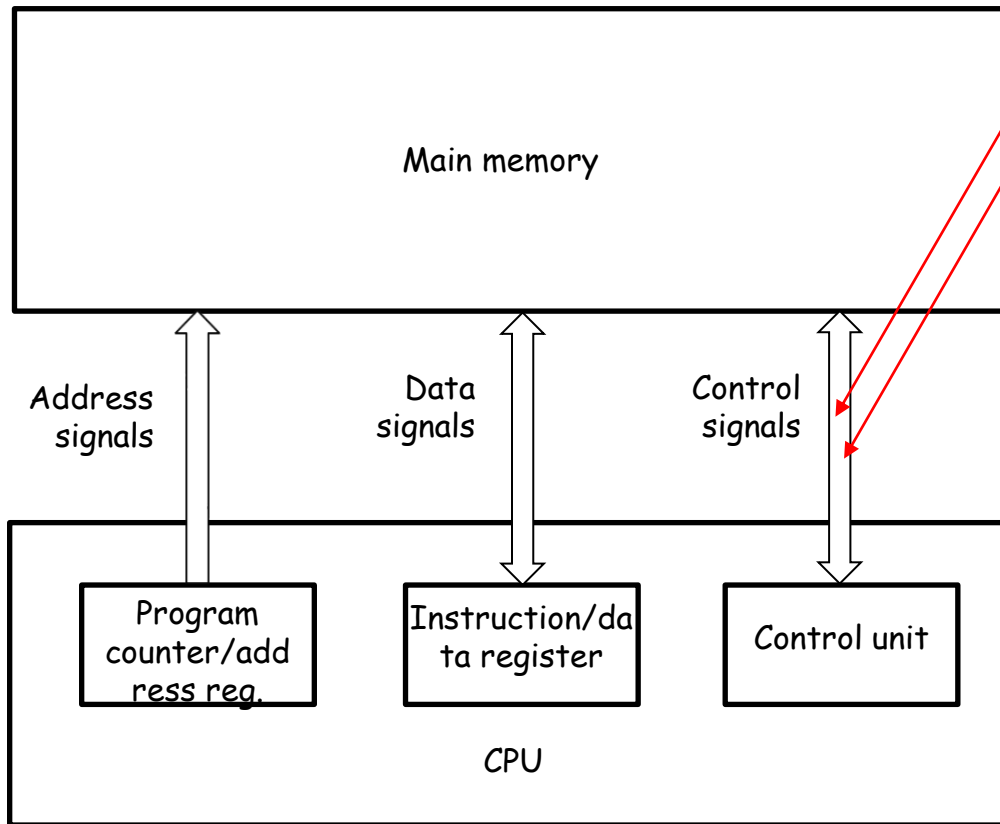


| Memory Address | | | Memory words |
|-------------------------|-------------|---------|--------------|
| Binary (length 16 bits) | Hexadecimal | Decimal | |
| 0000 0000 0000 0000 | 0000 | 0 | |
| 0000 0000 0000 0001 | 0001 | 1 | |
| 0000 0000 0000 0010 | 0002 | 2 | |
| 0000 0000 0000 0011 | 0003 | 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 | |

Interconnection CPU <-> main memory

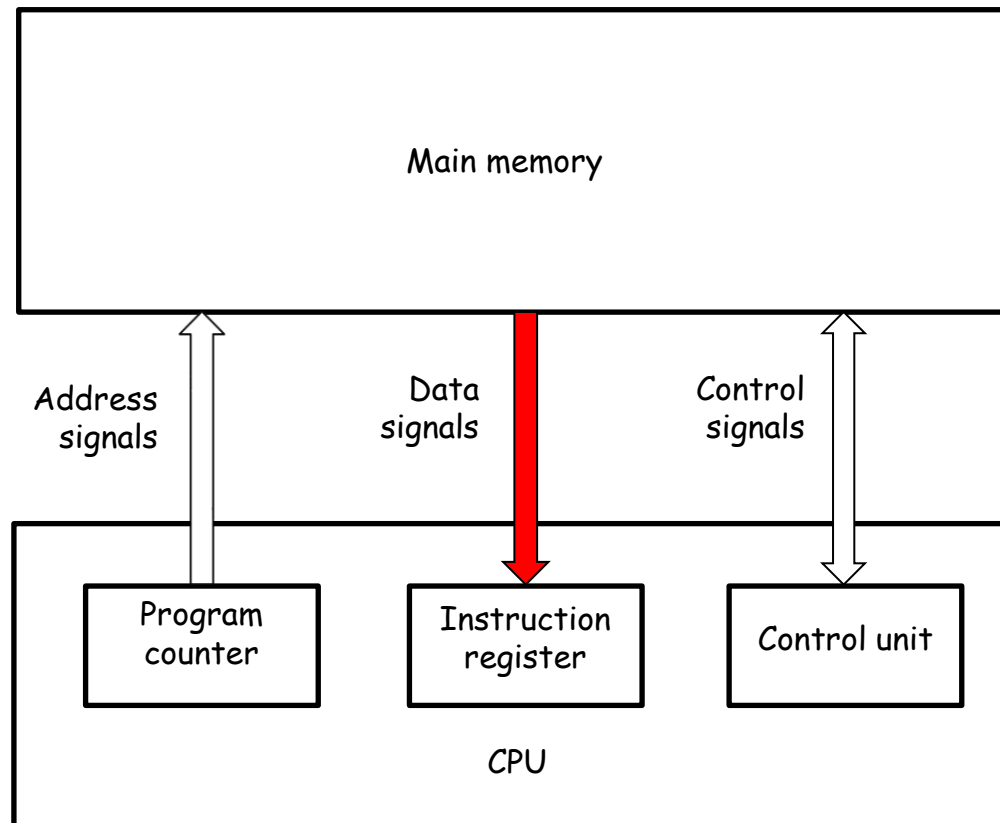
Bus = a group of related lines
(Address, Data, Control buses)

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



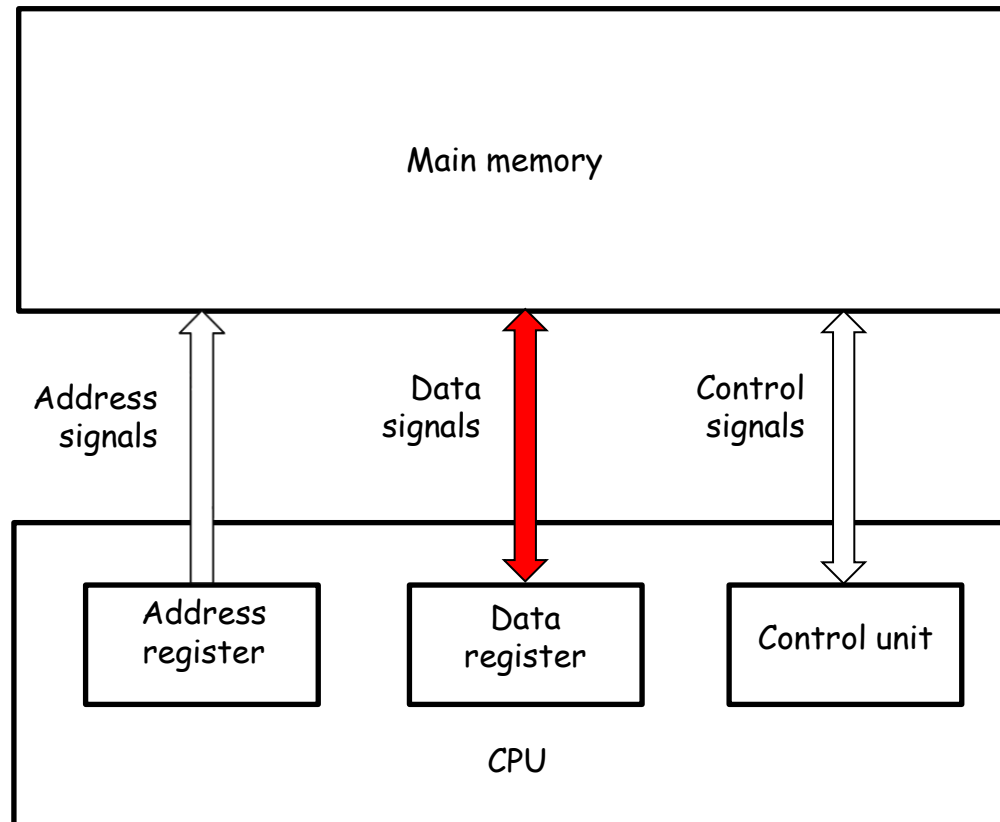
How does CPU access the main memory?

Example for accessing instructions:



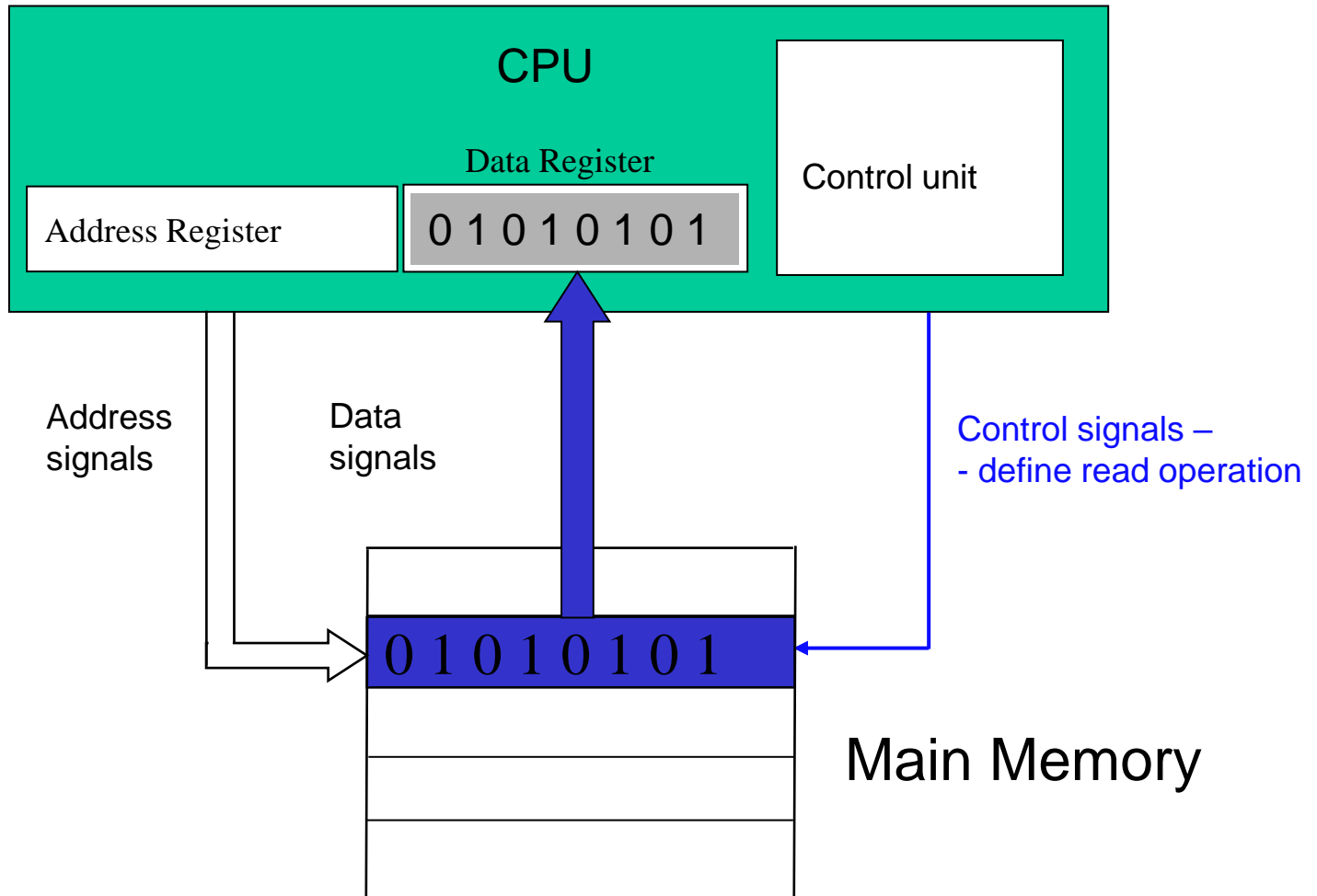
How does CPU access the main memory?

Examples for accessing operands:



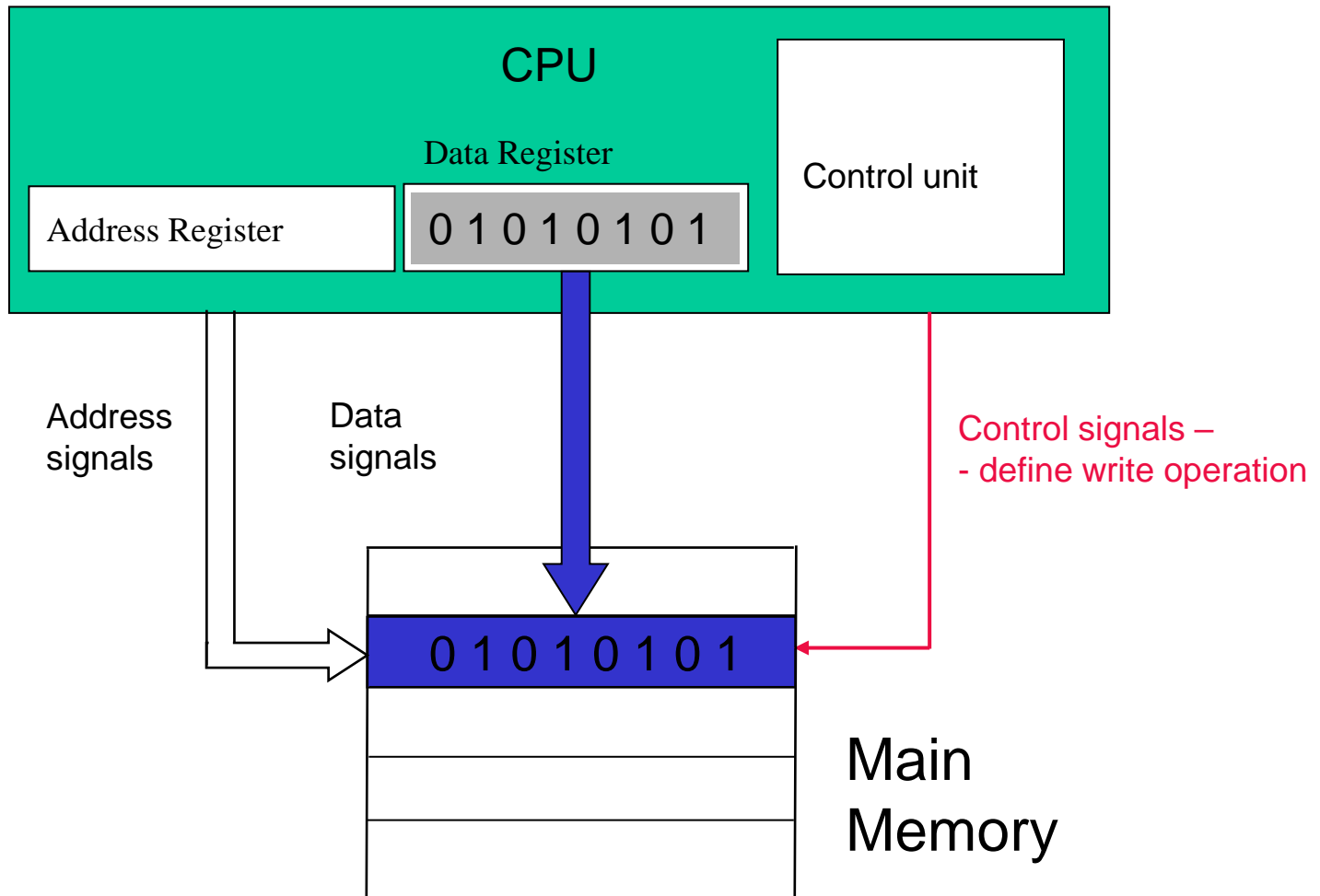
CPU and Main Memory

– read access



CPU and Main Memory

– write access

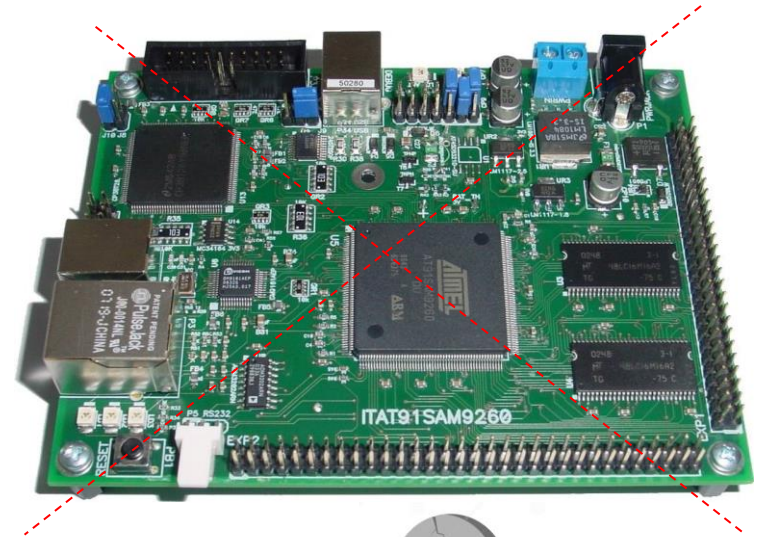


Computer architecture CA

Computer STM32H750-DK



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

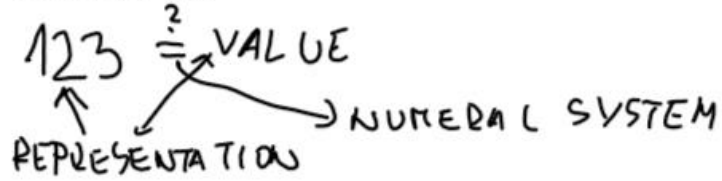


LAB 1.4 Quick intro to numeral systems

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

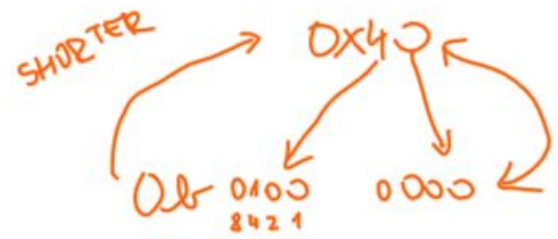
$$0x40 = 4 \cdot 16^1 + 0 \cdot 16^0 = \underline{\underline{64}}$$

DIGITS

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

RELATED

1 DIGIT HEX ↔ 4 DIGITS IN BIN

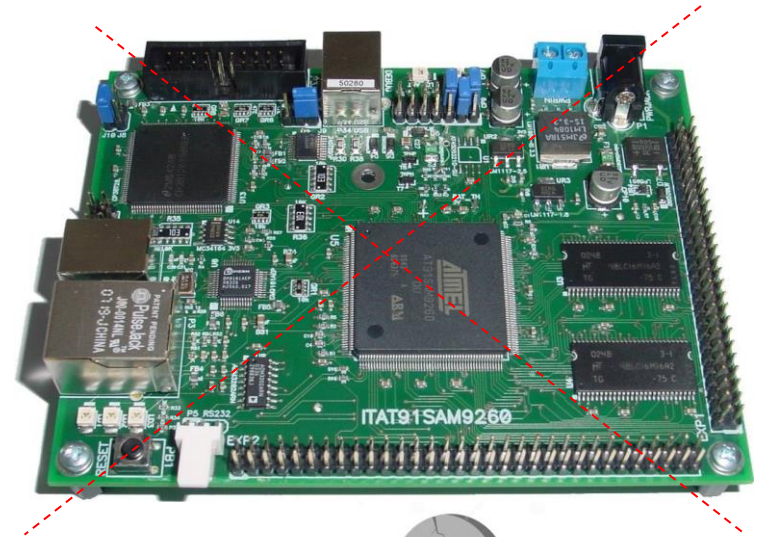


Computer architecture CA

Computer STM32H750-DK



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

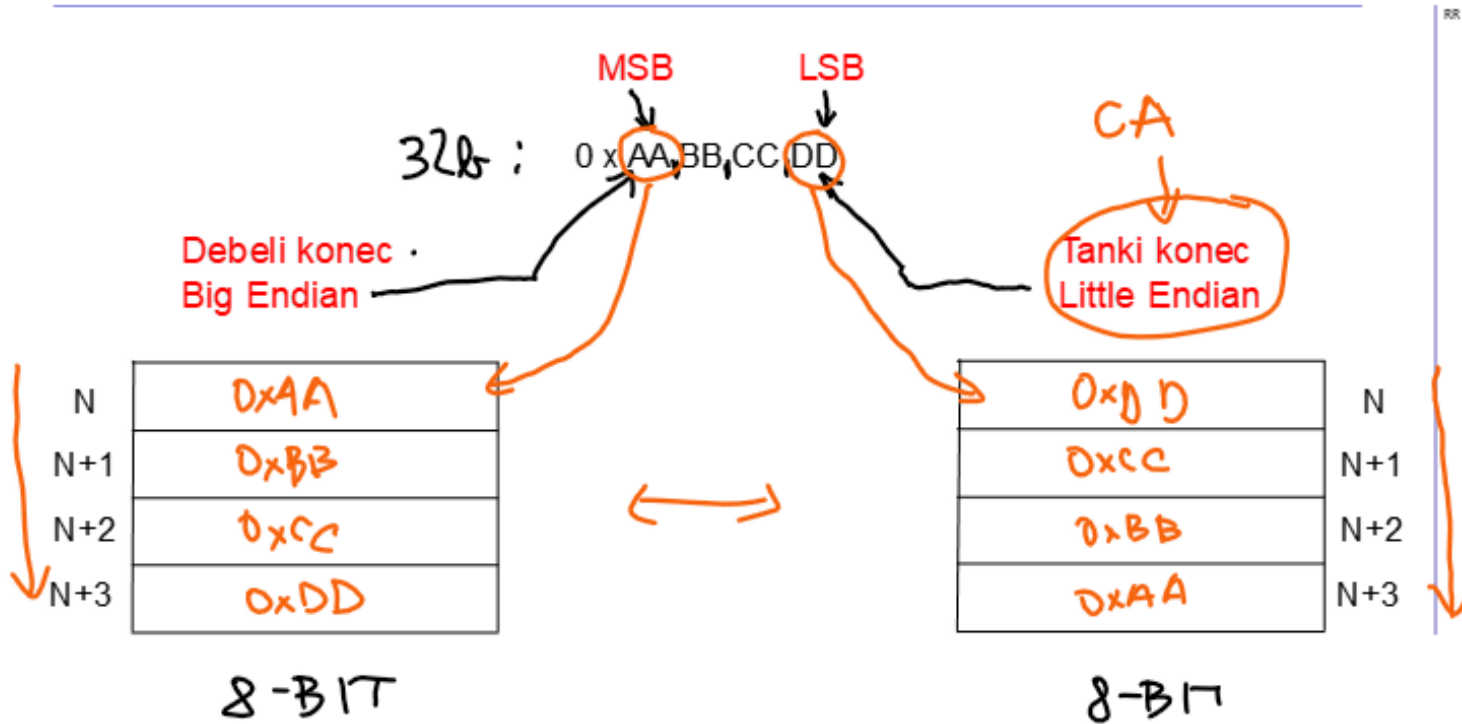


LAB 1.5 Big and Little Endian rules

LAB 1.5 Big and Little Endian rules

Big vs. Little Endian

Monday, October 12, 2020 2:18 PM

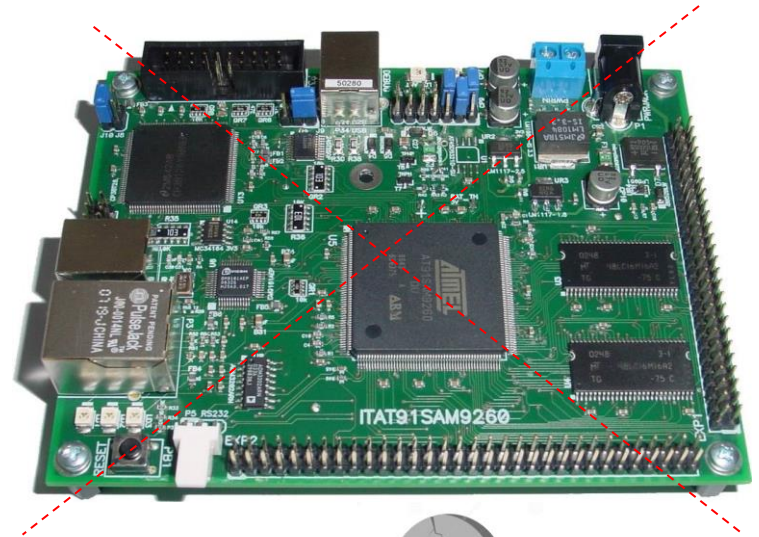


Computer architecture CA

Computer STM32H750-DK

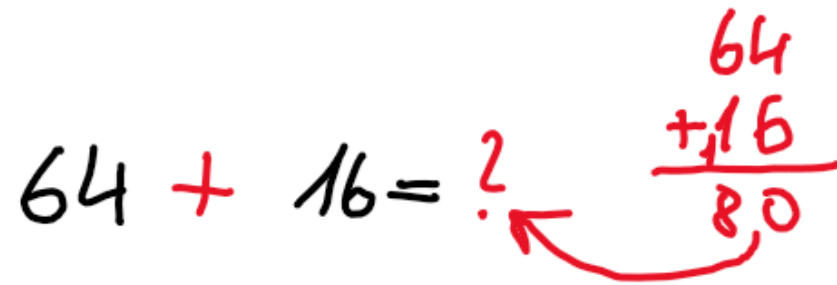


- Računalnik FRI-SMS
 - Mikrokontroler AT91SAM9260 iz družine mikrokontrolerov ARM9



LAB 1.6 Addition – human, python, assembler cases

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

$$64 + 16 = ?$$


Handwritten addition problem showing 64 plus 16 equals 80. A red arrow points from the result 80 to the question mark.

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

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
```

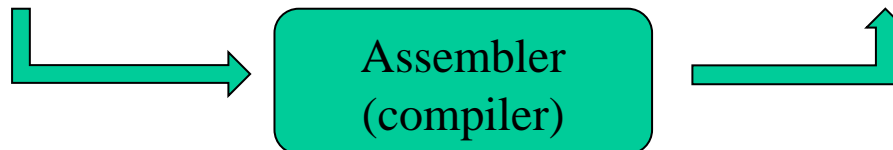
WinIDEA (case: rez = stev1 + stev2)

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 |

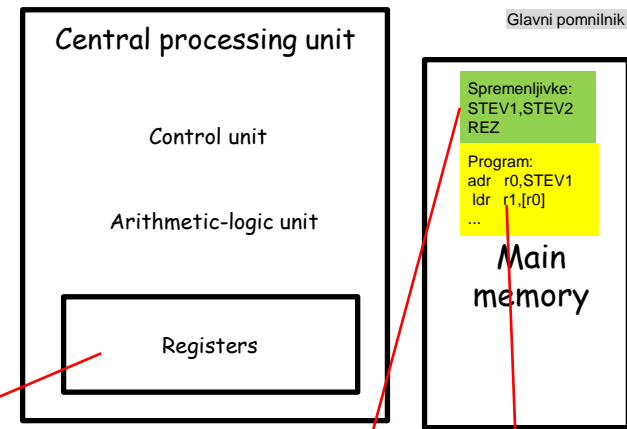


Execute instructions step-by-step and observe the register's values and the variable's values inside the main memory.

Practical example : Sum of 2 numbers

<https://cpulab01.net/?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 | 00001d3 NZCVI SVC |
| spsr | 00000000 NZCVI ? |

Disassembly (Ctrl-D)

Go to address, label, or register: 00000000

| Address | Opcode | Disassembly |
|----------|-----------|-------------------|
| | | STEV1: |
| 00000020 | 00000010 | andeq r0, r0, r0, |
| | | STEV2: |
| 00000024 | 00000040 | andeq r0, r0, r0, |
| | | REZ: |
| 00000028 | 00000000 | andeq r0, r0, r0 |
| | | 9 .align |
| | | 11 .global _start |
| | | 12 _start: |
| | | 14 adr r0, STEV1 |
| | | _start: |
| 0000002c | e24f0014 | adr r0, 0x20 (C |
| 00000030 | e5901000 | 15 ldr r1, [r0] |
| | | 17 adr r0, STEV2 |
| 00000034 | e24f0018 | adr r0, 0x24 (C |
| 00000038 | e5902000 | 18 ldr r2, [r0] |
| 0000003c | e0813002 | 20 add r3, r1, r2 |
| | | 22 adr r0, REZ |
| 00000040 | e24f0020 | adr r0, 0x28 (C |
| 00000044 | e5803000 | 23 str r3, [r0] |
| | | 26 end: b er |
| | | end: |
| 00000048 | eaffffffe | b 0x48 (0x48: enc |

Memory (Ctrl-M)

Go to address, label, or register:

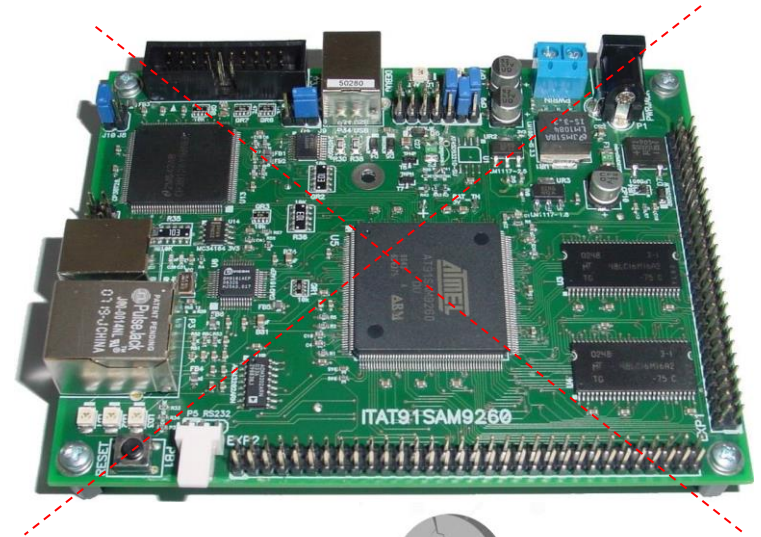
| Address | Memory contents and ASCII |
|----------|---|
| 00000000 | 00000000 00000000 00000000 00000000 |
| 00000010 | 00000000 00000000 00000000 00000000 |
| 00000020 | 00000010 00000040 00000000 e24f0014 |
| 00000030 | e5901000 e24f0018 e5902000 e0813002 |
| 00000040 | e24f0020 e5803000 eaffffffe 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
```

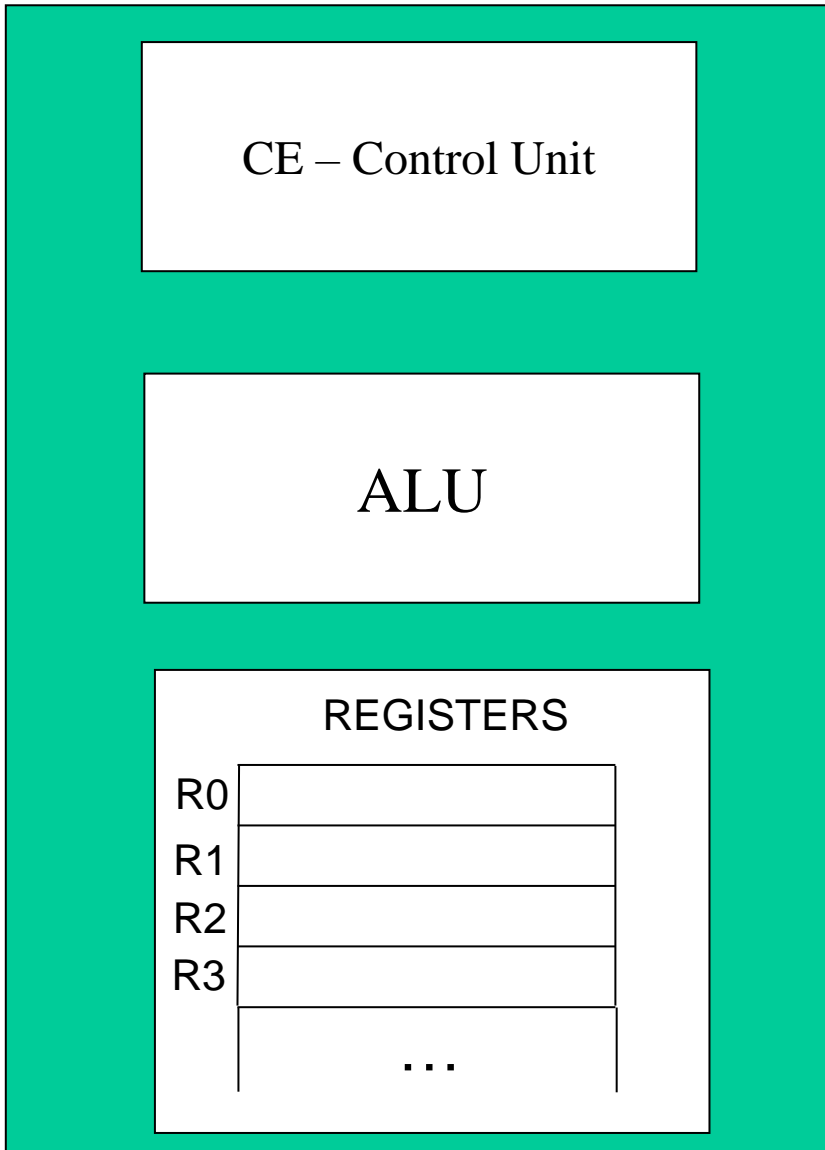
```
3 REZ = STEV1 + STEV2
```

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

<http://goo.gl/YXQ5qN>

Zgled: adding two numbers

CPU



Memory

| Address | Memory words (locations) | Label Content |
|-----------|--------------------------|--------------------------------|
| 0x20 = 0 | | STEVI |
| | | |
| | | |
| 0x24 = 4 | | STEVI |
| | | |
| | | |
| 0x28 = 8 | | REZ |
| | | |
| | | |
| 0x2C = 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

