X36SKD

Seminar

AVR-8bit Microcontroller Instruction Set and Architecture

Miroslav Skrbek, Josef Hlaváč, Jiří Buček

Seminar 4

- AVR Microcontroller Architecture
- Machine Code
- Symbolic instruction names
- Symbolic address names
- Program development flow in assembler

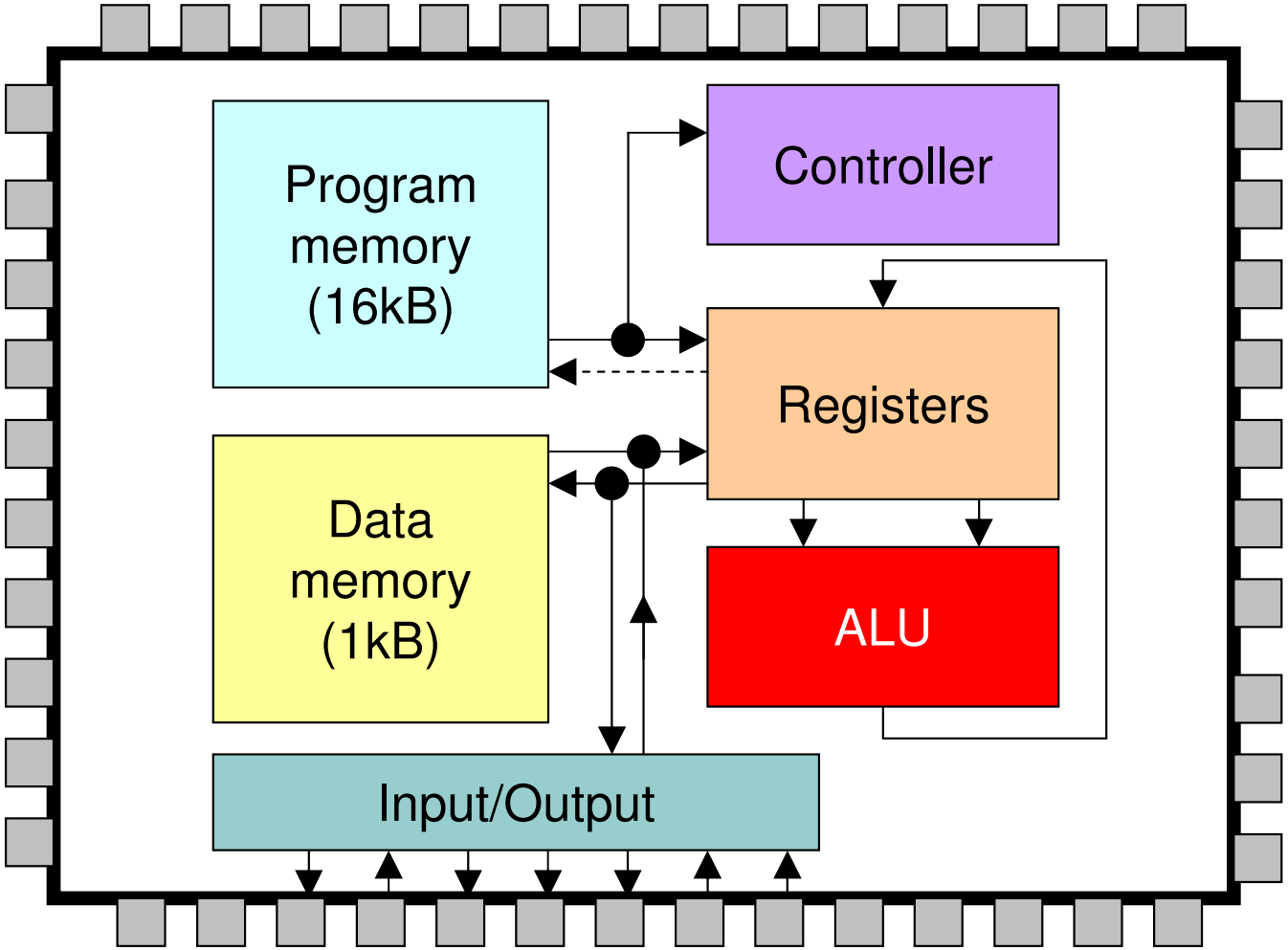
References

[1] *8-bit AVR® Instruction Set*, ATMEL Corporation, 2002. http://www.atmel.com

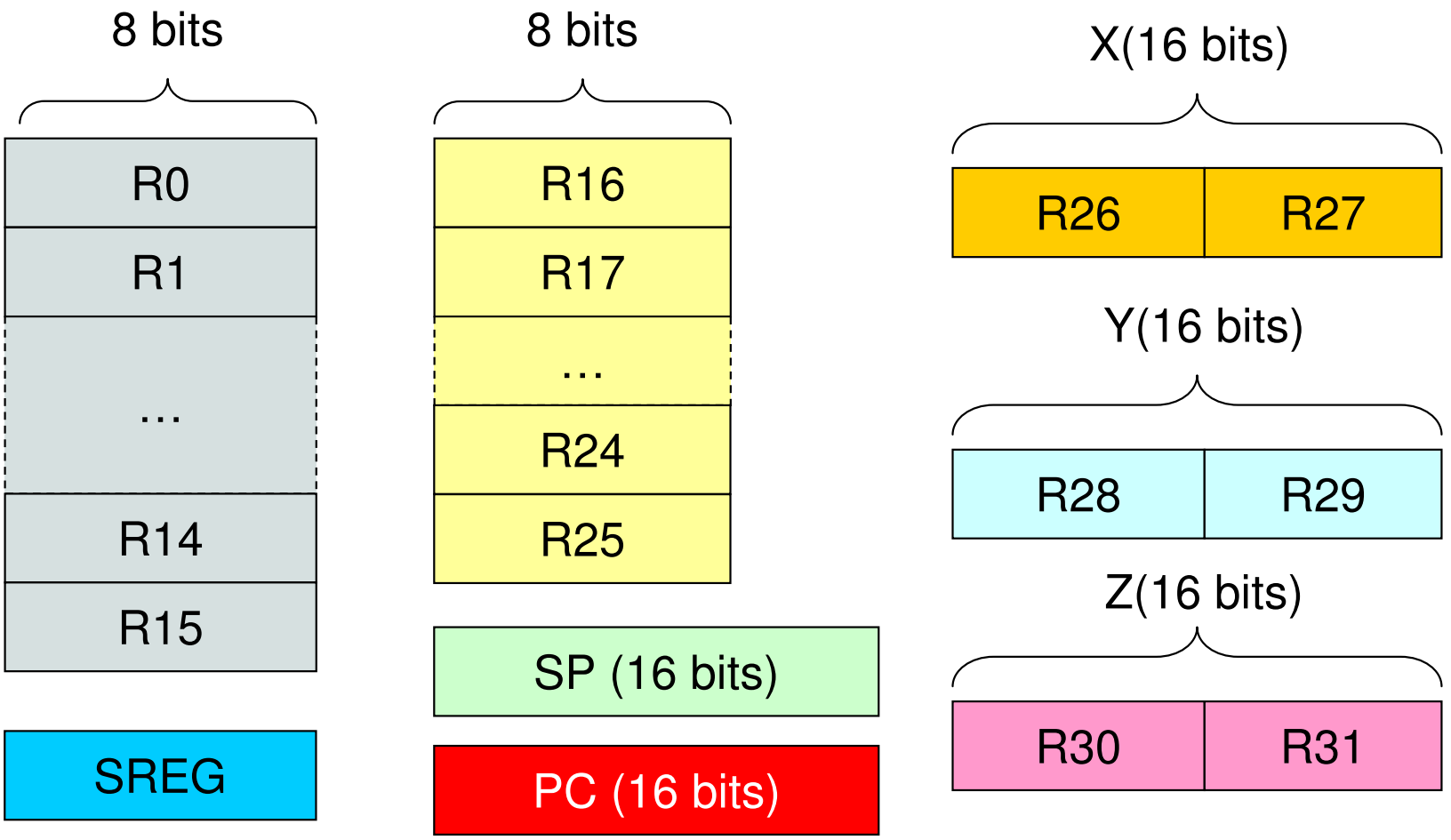
[2] *8-bit AVR® Microcontroller width 16K Bytes In-System Programmable Flash ATmega169*. Datasheet. ATMEL Corporation, 2003. http://www.atmel.com

[3] *Data-File conversion to Intel HEX-32 format*. Intel Corporation. http://www.intel.com

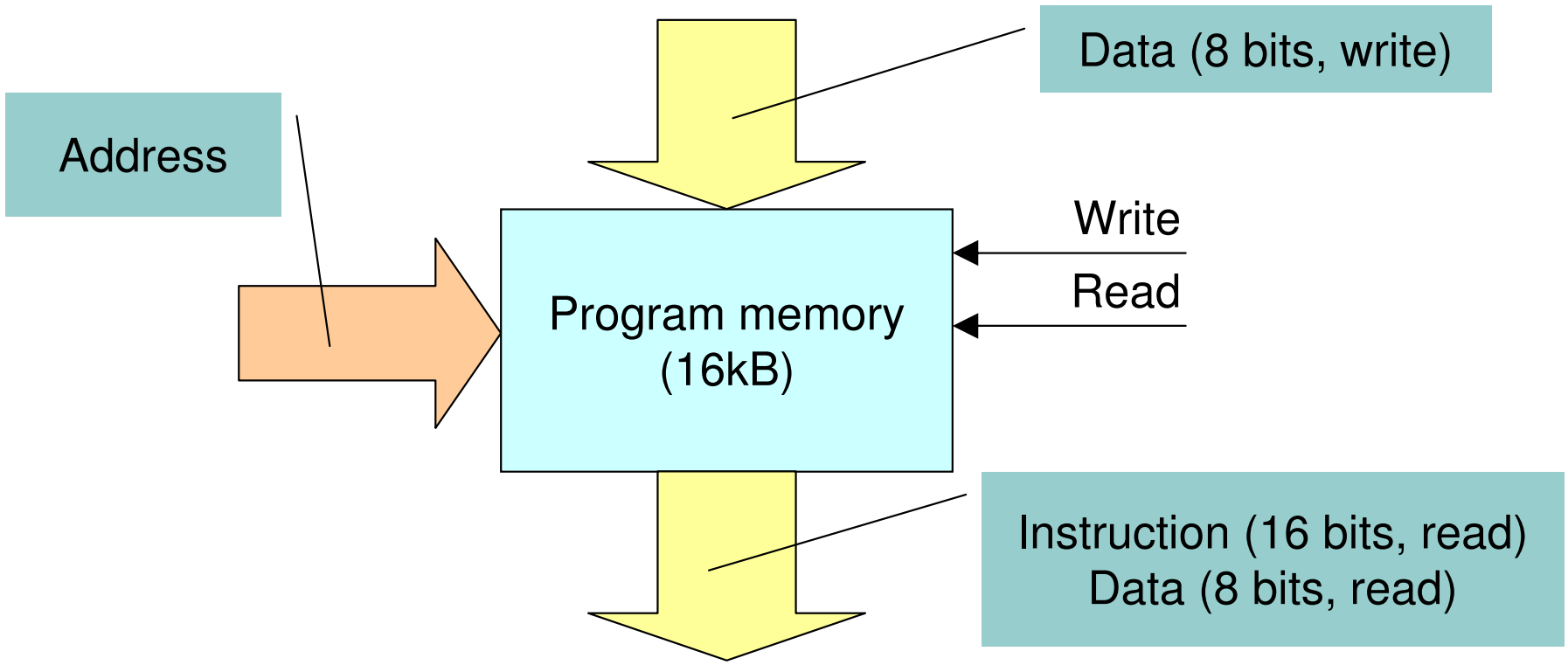
AVR microprocessor architecture



Registers

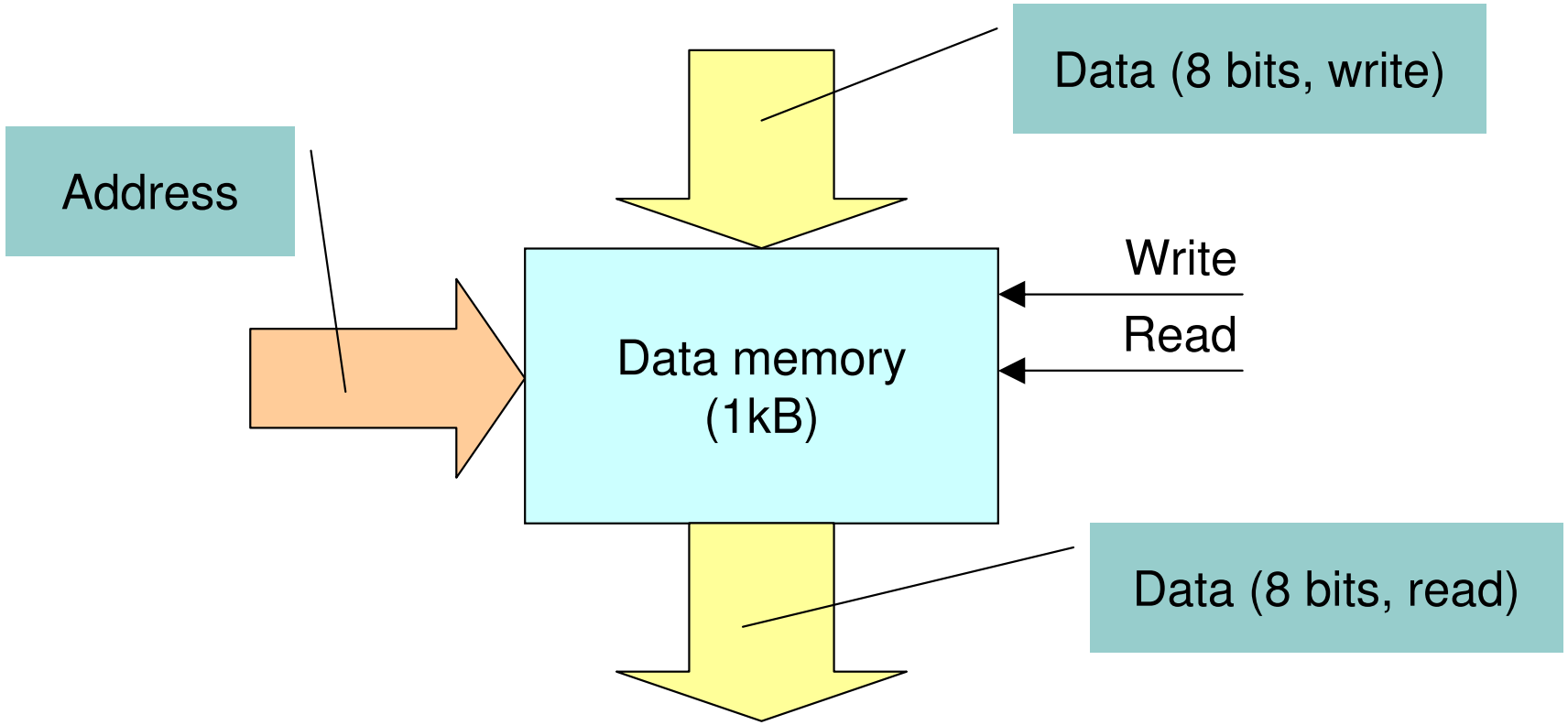

8 bits
R0
R1
...
R14
R15
8 bits
R16
R17
...
R24
R25
X(16 bits)
R26
R27
Y(16 bits)
R28
R29
Z(16 bits)
R30
R31
SP (16 bits)
SREG
PC (16 bits)

Program memory



Note: Program memory is usually FLASH type, hence writing is special

Data memory


Data (8 bits, write)
Address
Write
Read
Data memory
(1kB)
Data (8 bits, read)

Instruction cycle (controller action)

1. Fetch an instruction from the location pointed by the PC register
2. Decode the instruction
3. Get operands
4. Execute the instruction
5. Write-back the results
6. Calculate the address of the next instruction
7. Go to 1

Machine code I

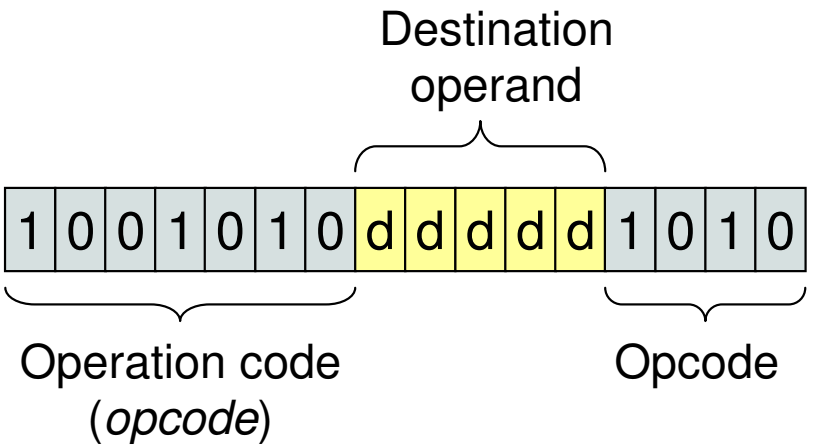
Assembly language statement
(Symbolic instruction)

```
dec r1
```

Destination operand

*Mnemonic** –
Instruction name

Machine code of the `dec` instruction



Example:

```
dec r1                    : 0x941A
```

* Mnemonic – from Greek “mnemonikos” (“μνημονικός”) – for remembrance, for memory

Machine code II

Assembly language statement
(Symbolic instruction)

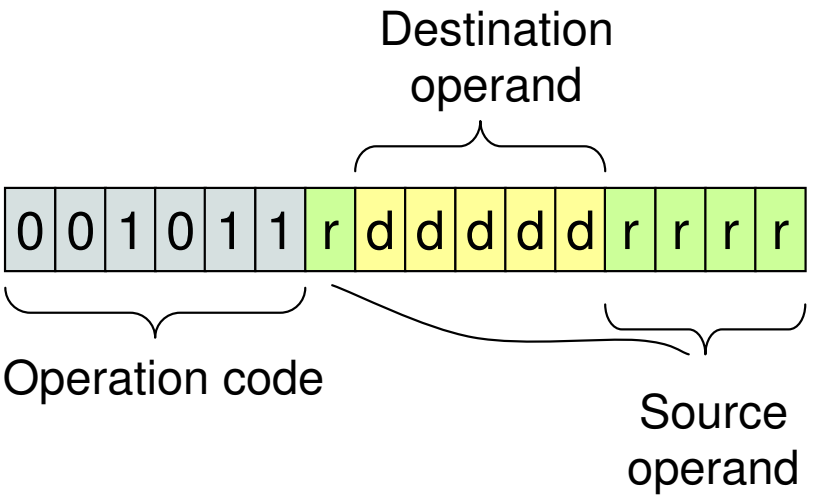
```
mov r1, r2
```

Source operand

Destination operand

Instruction name

Machine code of `mov`:



Example:

```
mov r1, r2                    : 0x2C12
```

Machine code III

Assembly language statement
(Symbolic instruction)

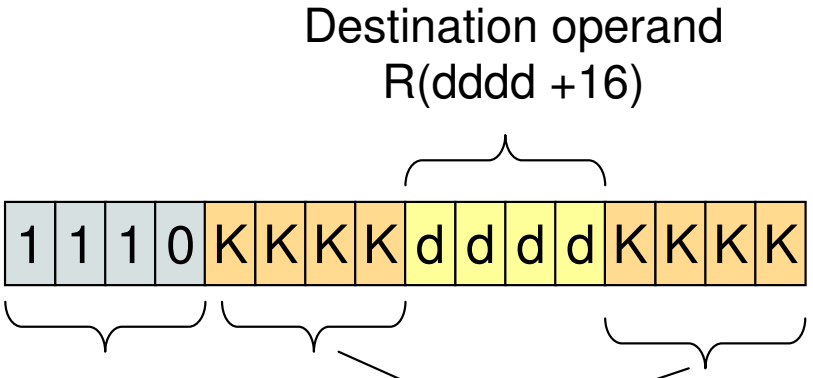
```
ldi r17, 0x54
```

Immediate constant

Destination operand

Instruction name

Machine code of `ldi`:



Example:

```
ldi r17, 0x54          : 0xE514
```

Machine code IV

Assembly language statement
(Symbolic instruction)

```
brne    PC-4
```

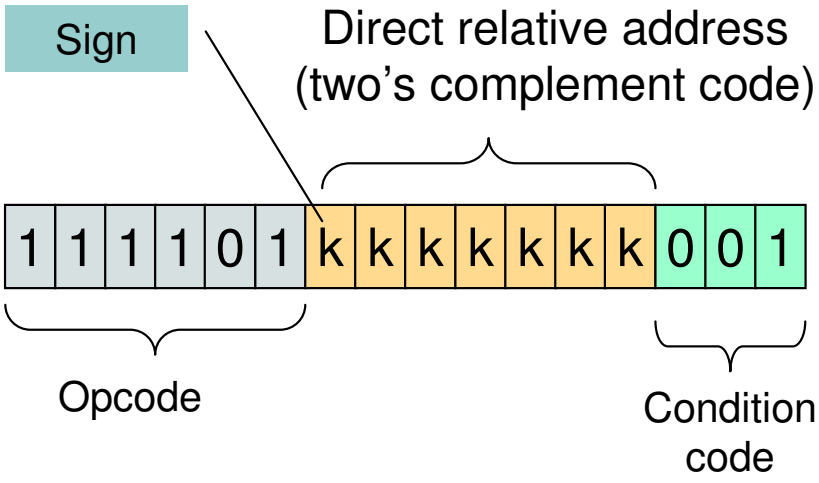
Relative jump target address

In assembly language, the address is written as absolute
(ex.: `brne 0x100, brne PC+6`*).*

While generating machine code, the compiler (assembler) computes the relative address .

Instruction name

Machine code of `brne`:



Examples:

```
brne PC-4              : 0xF7D9
brne PC+0xA            : 0xF449
```

Uwaga! Absolute target address is PC+k+1

Machine code V

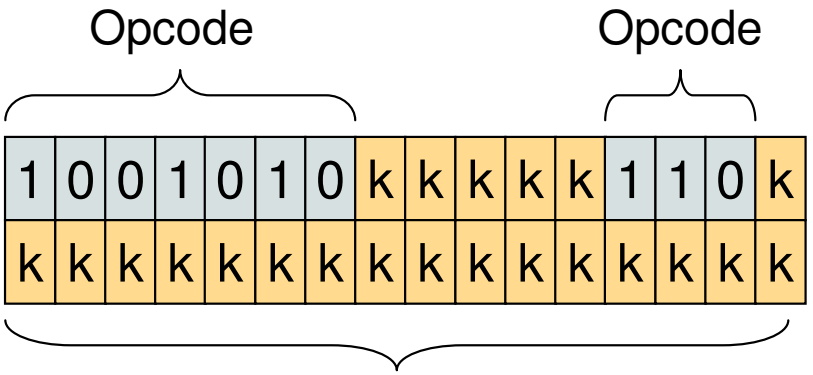
Assembly language statement
(Symbolic instruction)

```
jmp 0x125
```

Absolute jump target address

Instruction name

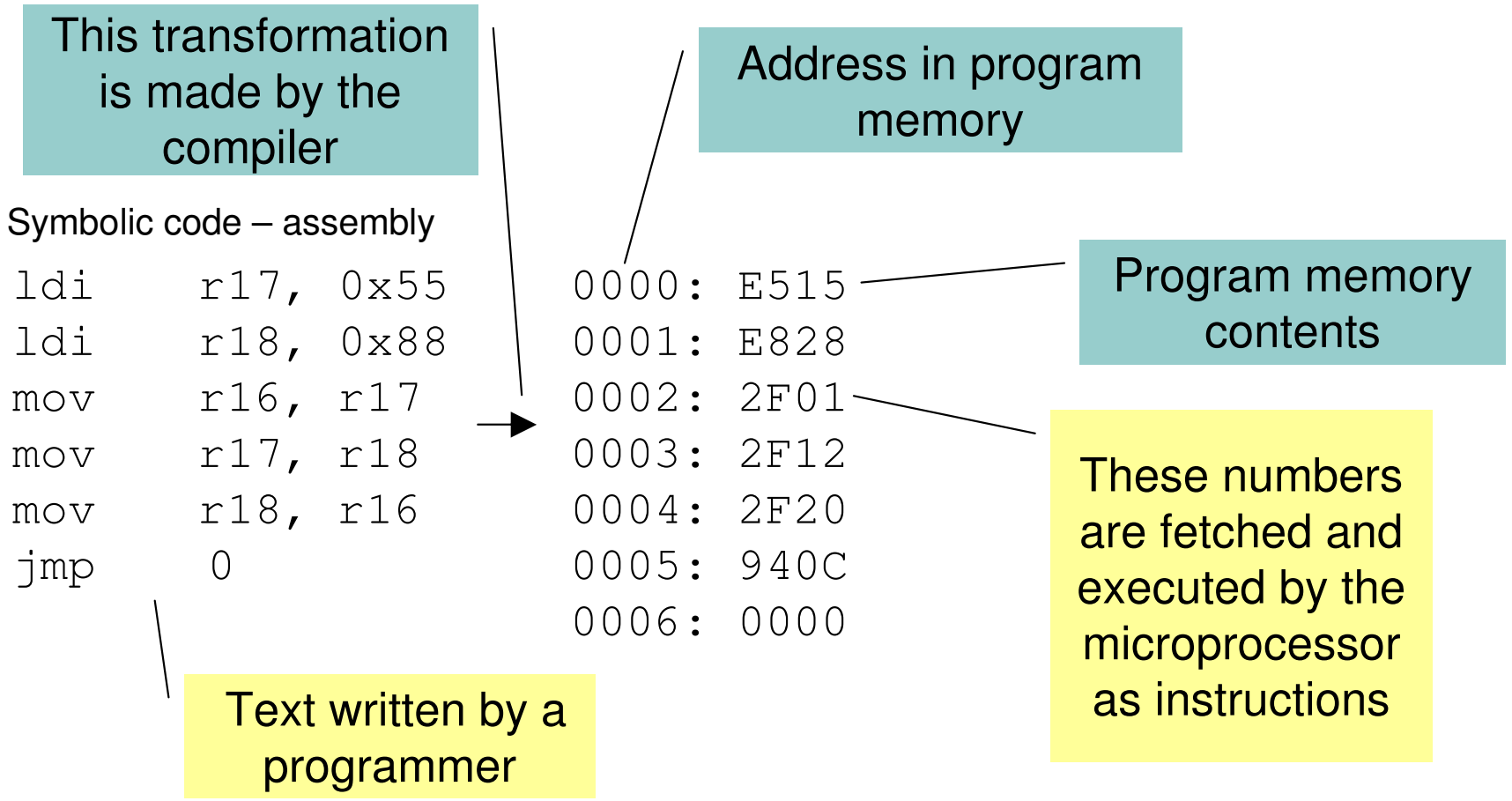
Machine code of `jmp`:



Example

```
jmp 0x125          : 0x940C, 0x0125
```

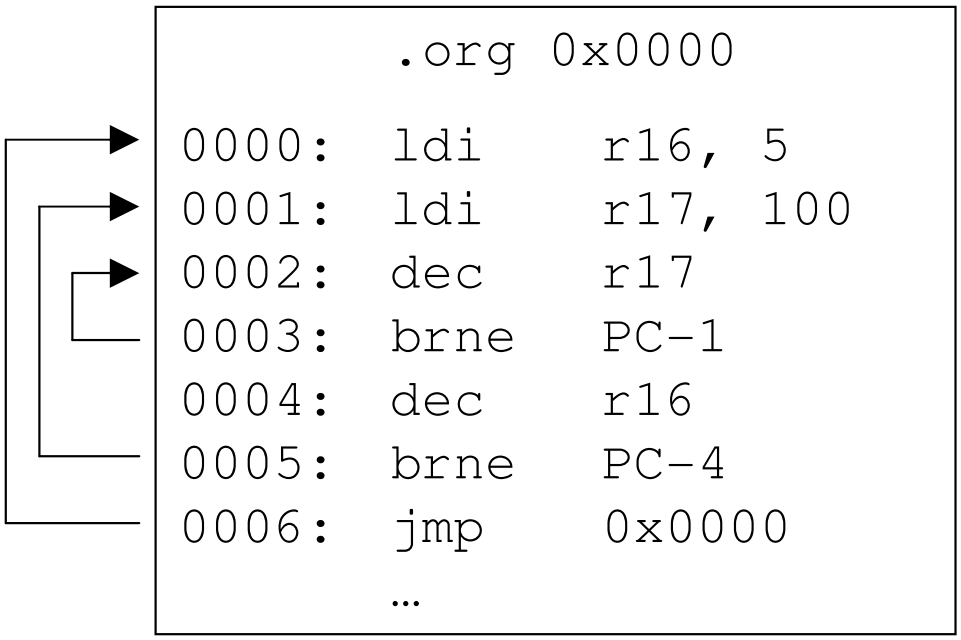
Example of program in assembly and machine code



Note: we assume the program in assembler is compiled starting from the address equal to zero.

Address modification problem when adding new instructions

Program example



```
        .org 0x0000

0000:   ldi     r16, 5
0001:   ldi     r17, 100
0002:   dec     r17
0003:   brne    PC-1
0004:   dec     r16
0005:   brne    PC-4
0006:   jmp     0x0000
        …
```

After adding two instructions

```
        .org 0x0000

0000:*  ldi     r18, 0
0001:   ldi     r16, 5
0002:   ldi     r17, 100
0003:   dec     r17
0004:*  inc     r18
0005:   brne    PC-2
0006:   dec     r16
0007:   brne    PC-5
0008:   jmp     0x0001
```

All jump instructions had to be modified

*Note: Added instructions are marked with **

Address modification problem when program is relocated

Program example

```
        .org 0x0000
0000:   ldi     r16, 5
0001:   ldi     r17, 100
0002:   dec     r17
0003:   brne    PC-1
0004:   dec     r16
0005:   brne    PC-4
0006:   jmp     0x0000
        …
```

Relocated program

```
        .org 0x0100
0100:   ldi     r16, 5
0101:   ldi     r17, 100
0102:   dec     r17
0103:   brne    PC-1
0104:   dec     r16
0105:   brne    PC-4
0106:   jmp     0x0100
```

Relative addresses of branches stay unchanged (see `brne`), addresses of absolute jumps (see `jmp`) have to be modified.

Solution – symbolic addresses – labels

Original program

```
          .org 0x0000

loop:     ldi      r16, 5
cycl1:    ldi      r17, 100
cycl2:    dec      r17
          brne     cycl2
          dec      r16
          brne     cycl1
          jmp      loop
          …
```

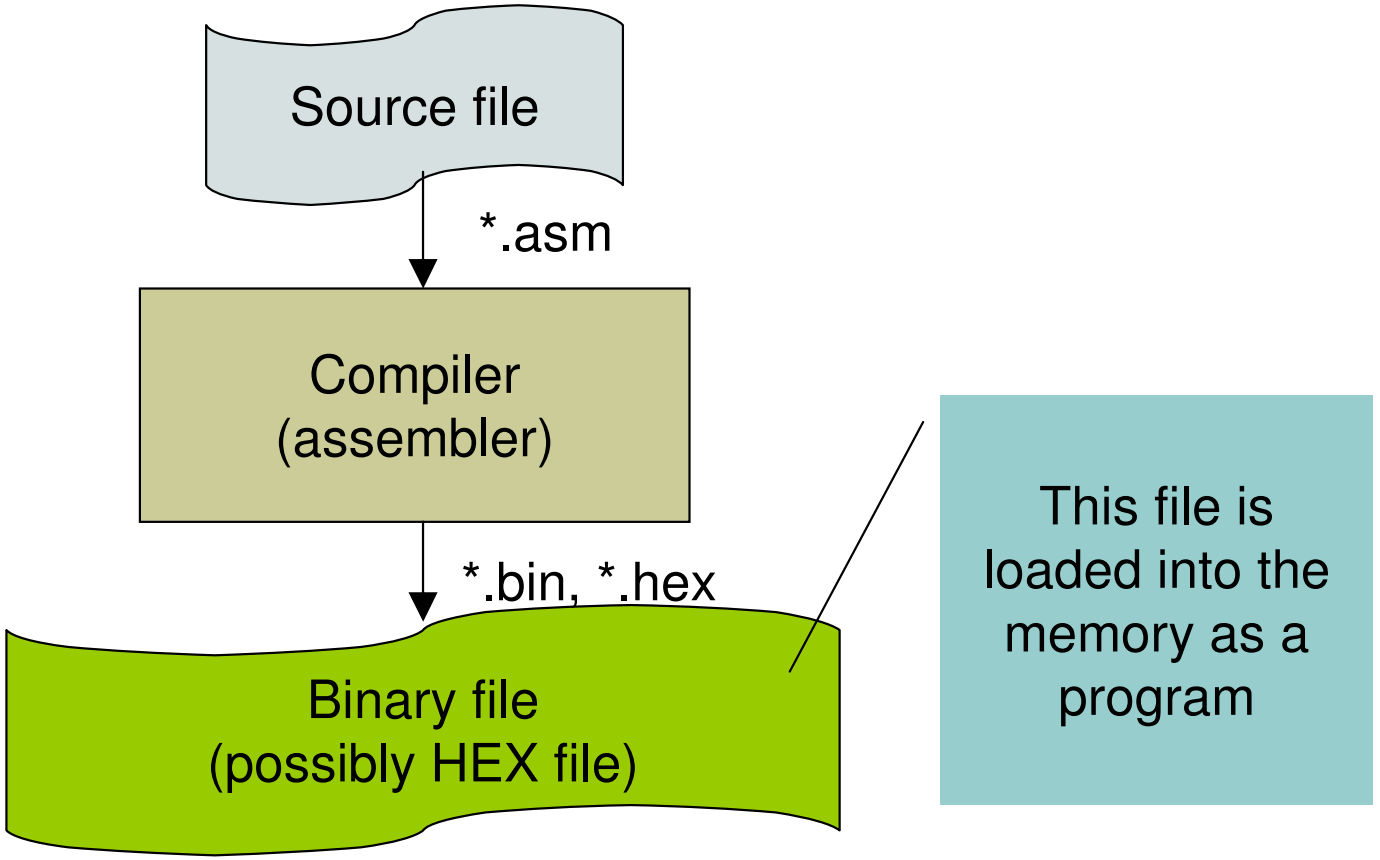
Modified and relocated program

```
          .org 0x0100

          ldi      r18, 0
loop :    ldi      r16, 5
cycl1:    ldi      r17, 100
cycl2:    dec      r17
          inc      r18
          brne     cycl2
          dec      r17
          brne     cycl1
          jmp      loop
```

Task: Translate into machine code

```
                .org 0x0200

start:          ldi   r18, 0
loop:           ldi   r16, 5
cycl1:          ldi   r17, 100
cycl2:          dec   r17
                inc   r18
                brne  cycl2
                dec   r17
                brne  cycl1
                jmp   loop
```

Technology of program development in assembler



What is a HEX file?

HEX file contains machine code prepared for loading into memory

Binary data are encoded in text form

Format: