XE36SKD

AVR Instruction Set

Working with Memory

Miroslav Skrbek, Josef Hlaváč, Jiří Buček

Seminar 7

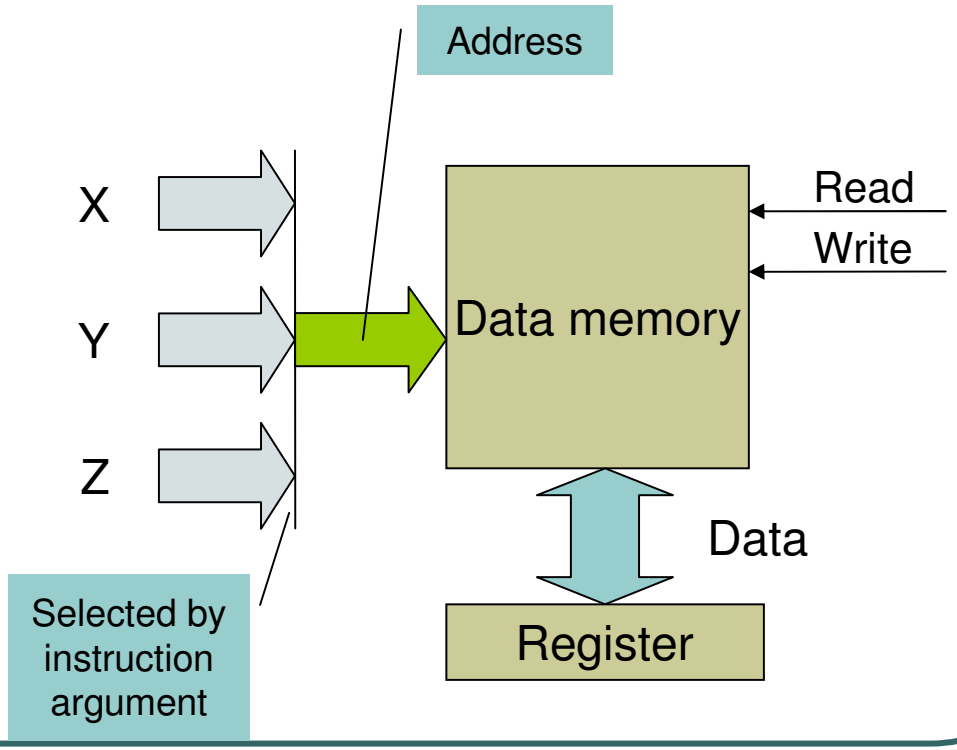
- Working with memory
- Working with arrays

Data memory

Data memory - 1kB

Addressed with 16-bit registers X, Y, Z

Index register	High orders	Low orders
X	R27	R26
Y	R29	R28
Z	R31	R30



Data memory address space

00h-1Fh	registers R0-R31
20h-5Fh	I/O registers addressable by IN and OUT instructions
60h-FFh	other I/O registers, e.g. LCD controller
100h-4FFh	memory (RAM) for free usage and for stack

Memory operations (direct addressing)

Storing of register R16 to the address 100h

```
ldi     r16, 0x55
sts     0x100, r16
```

Loading the contents of memory location at address 100h into R17

```
lds     r17, 0x100
```

Memory operations (indirect addressing)

Storing the contents of R16 to the address 100h

```
ldi     r16, 0xAB
ldi     r26, 0x00
ldi     r27, 0x01
st      X, r16
```

Loading the contents of memory cell with address 100h into R0

```
ldi     r26, 0x00
ldi     r27, 0x01
ld      r0, X
```

Storing the contents of R20 to the addresses 105h and 106h

```
ldi     r20, 0xCD
ldi     r28, 0x05
ldi     r29, 0x01
st      Y+, r20
st      Y, r20
```

Loading the contents of memory cell at 105h into R7 and at 106h to R8

```
ldi     r28, 0x06
ldi     r29, 0x01
ld      r8, y
ld      r7, -y
```

Similarly, we can use the Z register. In that case, the address is loaded into R30 and R31.

Allocating variables and arrays in the data memory

Variables and arrays in memory are declared with pseudoinstructions (.BYTE) in the data segment (.DSEG)

Declaration of two 1-byte variables `count` and `sum`

```
.DSEG

count:   .BYTE 1
sum:     .BYTE 1
```

Declaration of a 10-byte array

```
.DSEG

array:   .BYTE 10
```

Labels
(Symbolic representation of the address in the data memory. This address will be assigned at compile time. It can be influenced with the .ORG directive).

Note: If instructions follow variable or array declarations, they must be preceded by the .CSEG directive.

Example: Declare a 20-byte array and fill it with the value 55h

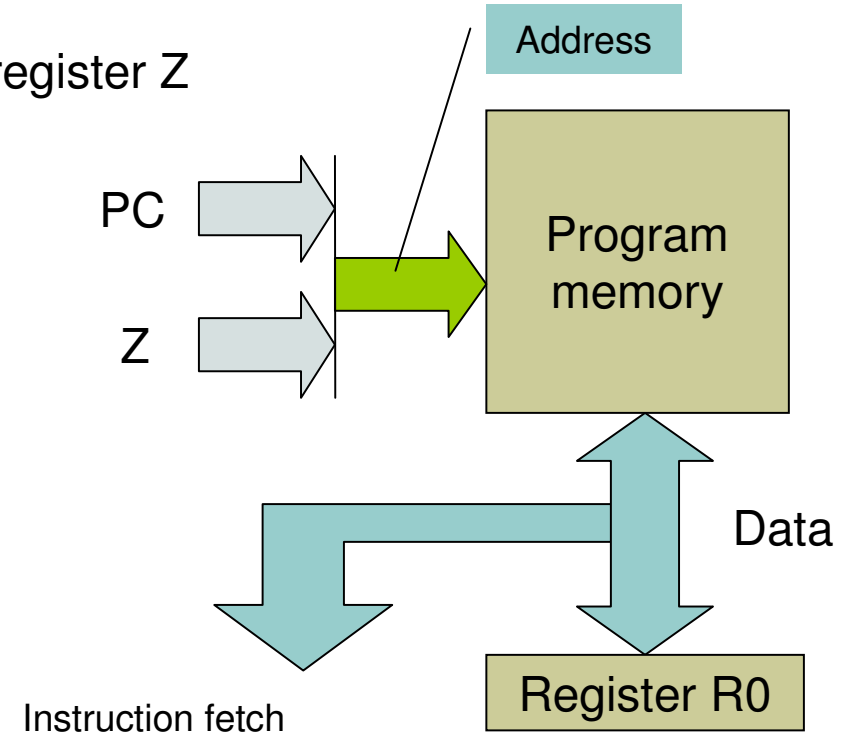
```
        .DSEG
        .ORG 0x100
array:  .BYTE 20

        .CSEG
        ldi     r30, low(array)
        ldi     r31, high(array)
        ldi     r29, 20
        ldi     r28, 0x55
cycle:  st      Z+, r28
        dec     r29
        brne    cycle
end:    rjmp    end
```

Program memory

Program memory – 16kB

Addressed with the 16-bit register Z

```
lpm
lpm r1, Z
lpm r20, Z+
```



Example: Copy a string from program memory to data mem.

```
start:              jmp     loadstr

srcstr:  .db "this is the string", 0

                    .dseg
                    .org 0x100
deststr:            .byte 256

                    .cseg
loadstr:            ldi     r30, low(srcstr<<1)
                    ldi     r31, high(srcstr<<1)
                    ldi     r28, low(deststr)
                    ldi     r29, high(deststr)
cp_cycle:           lpm
                    st      Y+, r0
                    adiw    r30, 1
                    tst     r0
                    brne    cp_cycle

                    …
```

Source string in the program memory

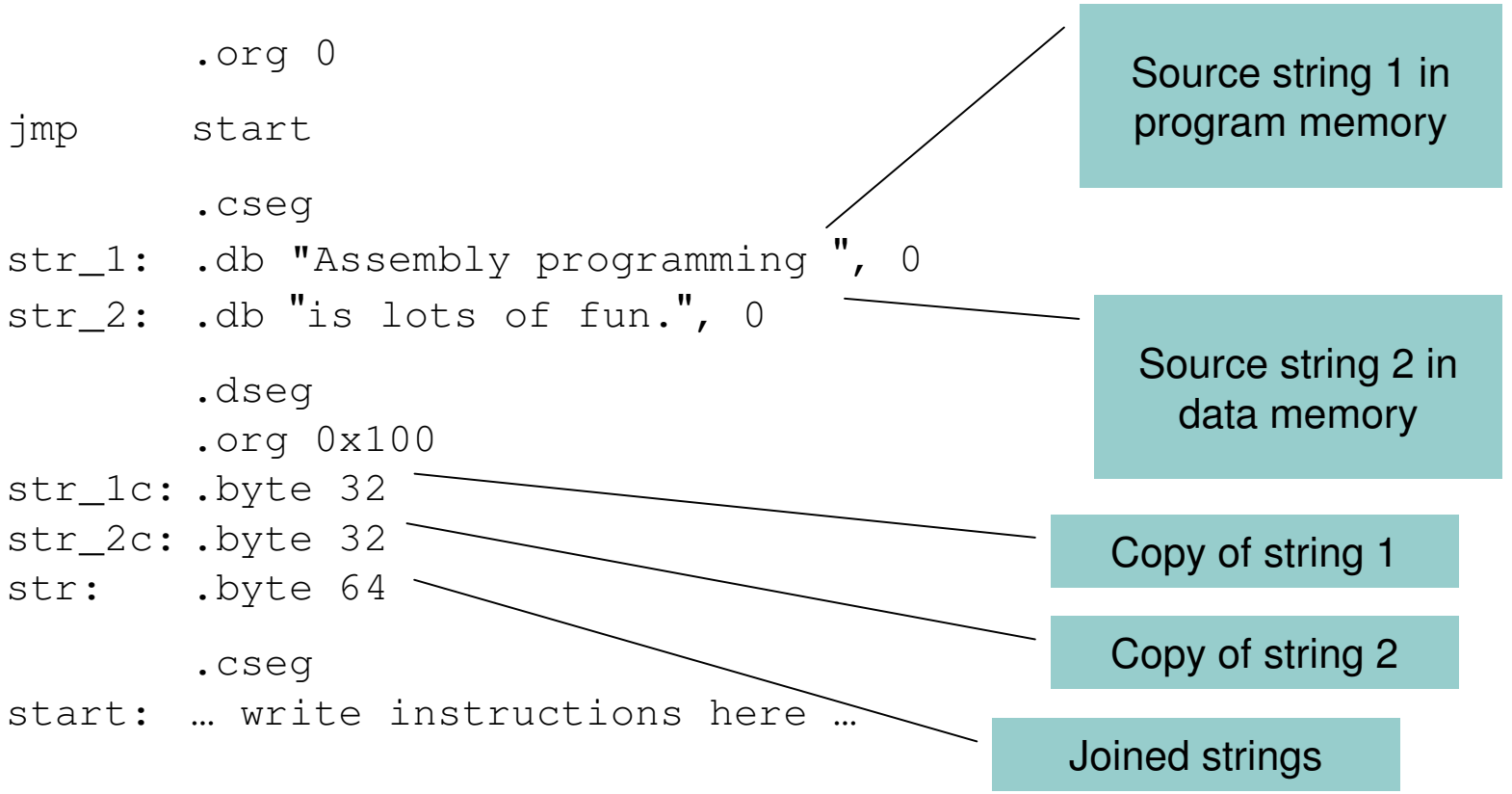
Allocated space for target string in data memory

Address multiplied by two. Program memory is addressed by words but LPM addresses by bytes.

Register Z (R31:R30) incremented

Task: Concatenate two strings into one. Load both strings from program memory, store the result in the data memory.

```
        .org 0
jmp     start

        .cseg
str_1:  .db "Assembly programming ", 0
str_2:  .db "is lots of fun.", 0

        .dseg
        .org 0x100
str_1c: .byte 32
str_2c: .byte 32
str:    .byte 64

        .cseg
start:  … write instructions here …
```



Hint

- Copy string 1 into the data memory
- Copy string 2 into the data memory
- Concatenate both strings in the data memory into one