Introduction to
Computer Organization

Memory and Addressing

Tim Dosen POK

Sample Program

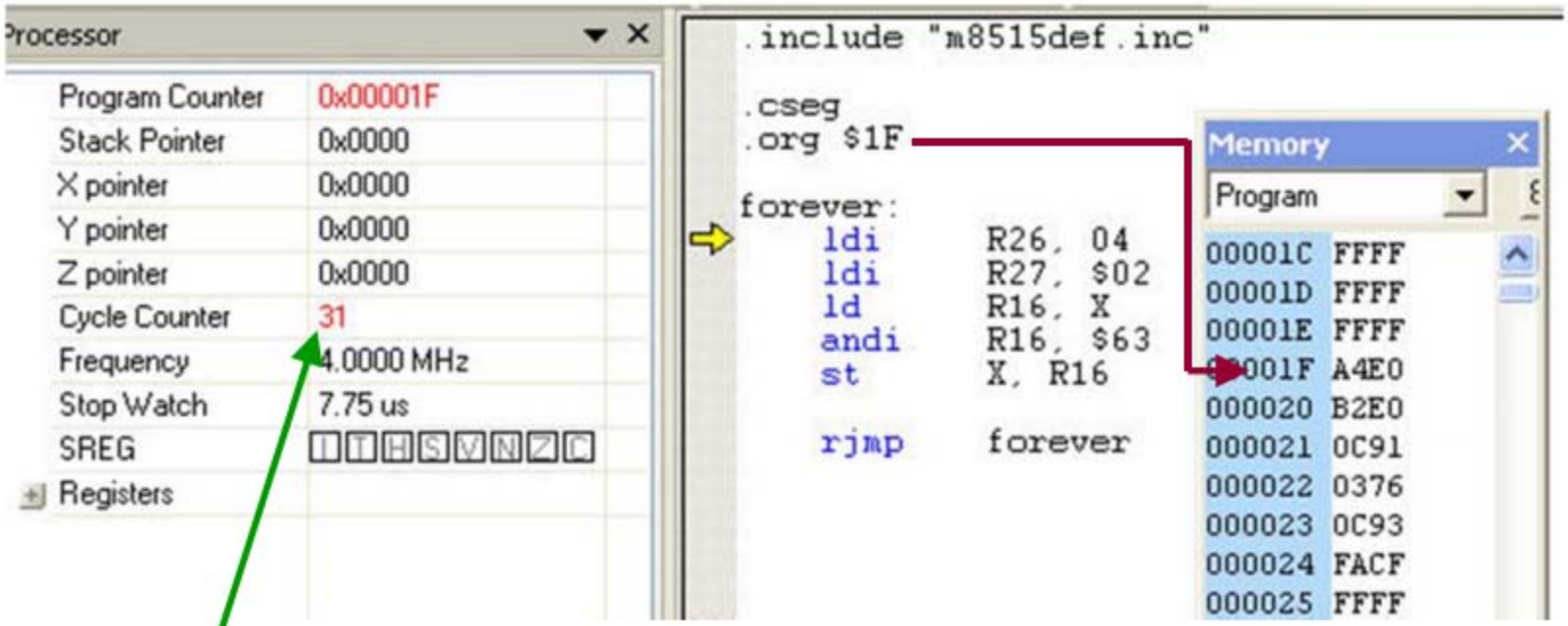
```
.include “m8515def.inc“

forever:

    ldi R26, 4 ; 4 in decimal

    ldi R27, $02 ; $02 in hexadecimal

    ld R16, X

    andi R16, 0x63

    st X, R16

    rjmp forever
```

0x02 = $02

Directives

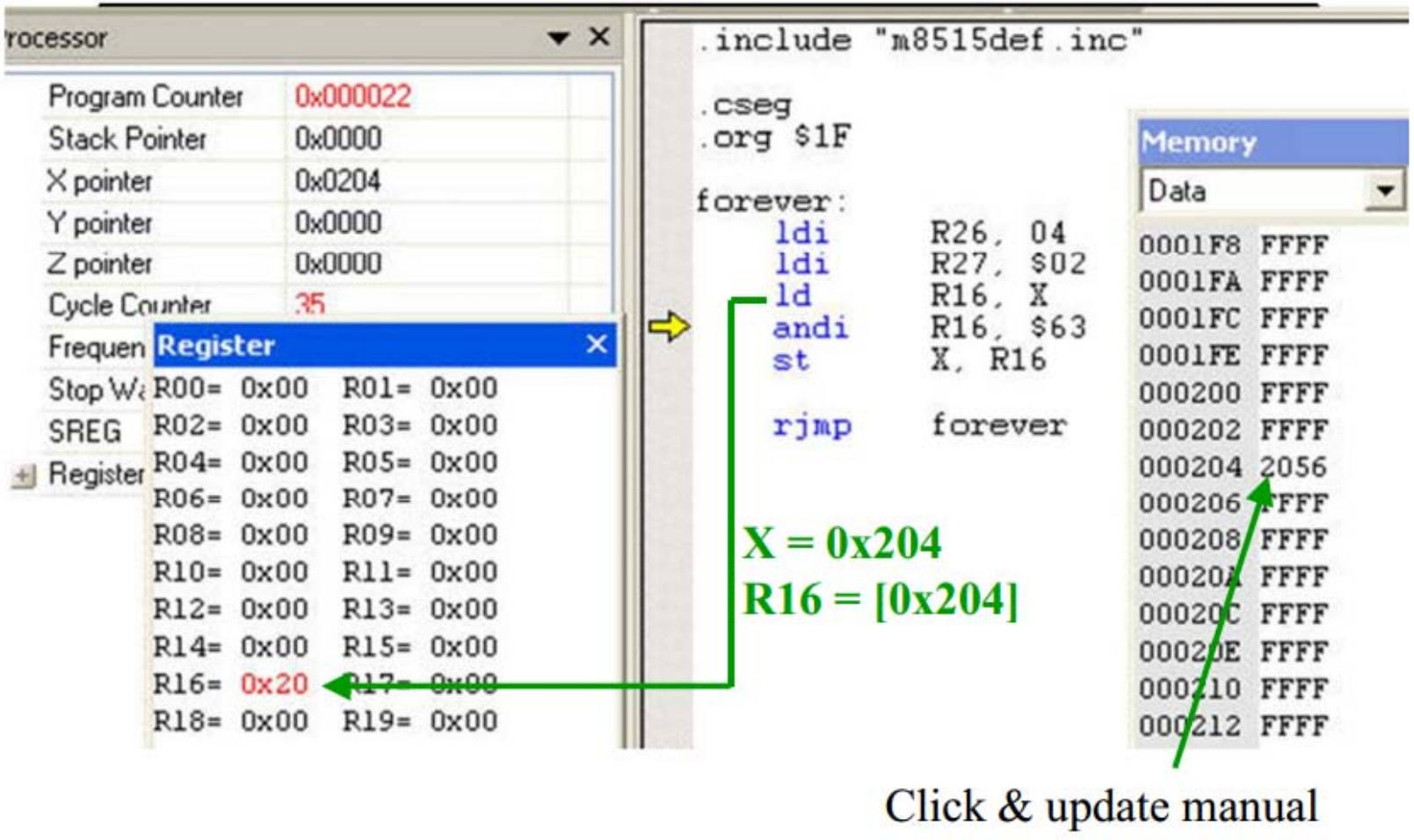
Directive	Fungsi	Contoh
.include	Memasukan definisi-definisi dari tipe prosesor yang digunakan.	.include "8515def.inc"
.def	Mendeskripsikan nama dari register	.def temp2 = r17
.equ	Mendeskripsikan sebuah nilai konstanta	. equ CONS = 123
.db	Mendefinisikan nilai yag akan disimpan pada program memory	.db "1,2"
.macro & .endmacro	Menandai dimulai & selesainya MACRO	.macro NAMAMACRO ..... .endmacro
.org	Mendefinisikan alamat penyimpanan baris kode pada alamat program memory tertentu	

Directive – Code Segment (.cseg)

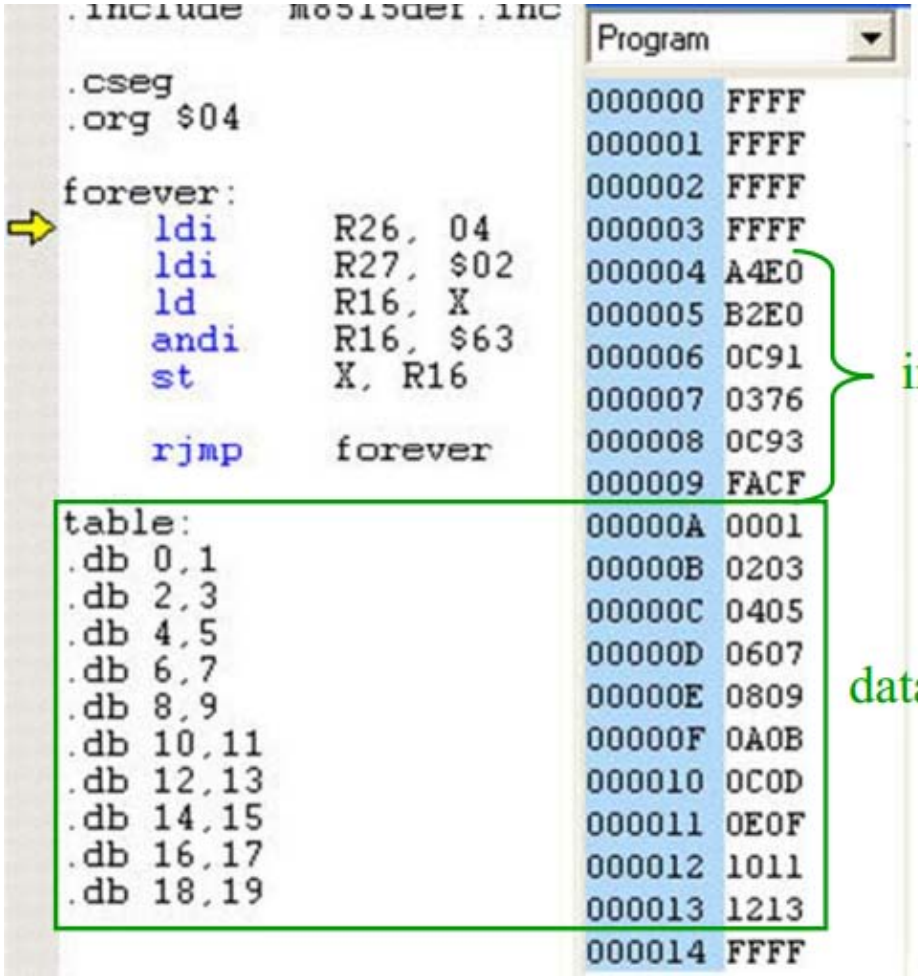


31 clock needed to reach 1st instruction

Directive – Code Segment (.cseg)(cont.)



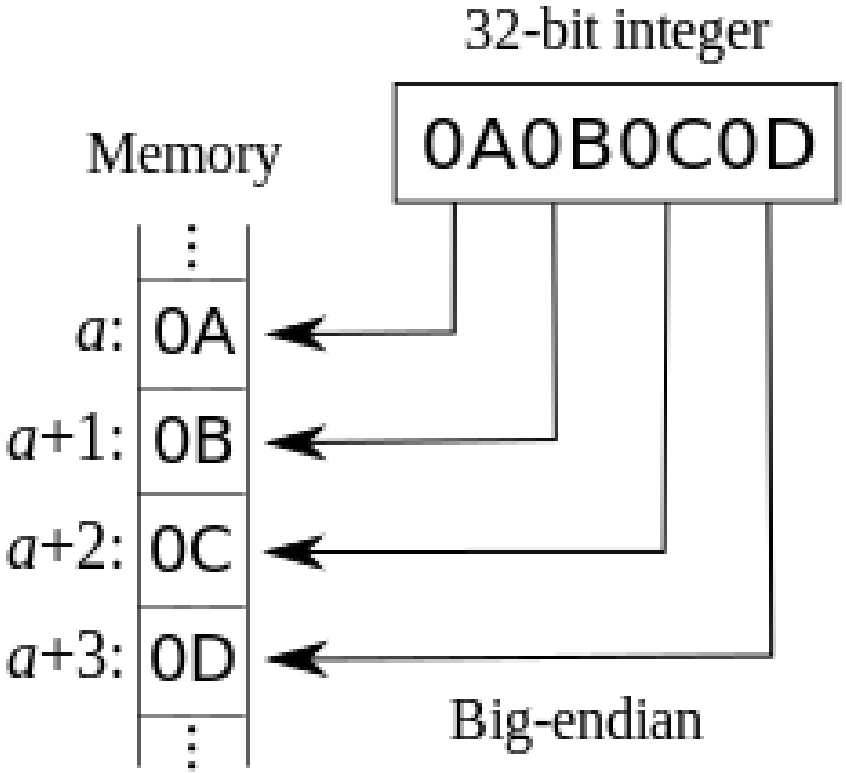
Define Constant Byte (.db)

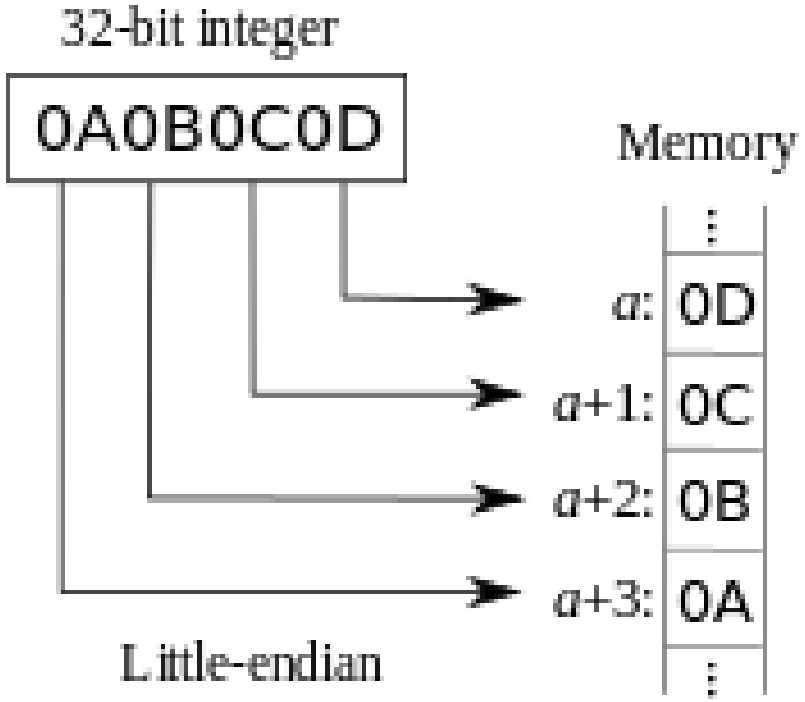


is directive for putting data into program memory (flash).

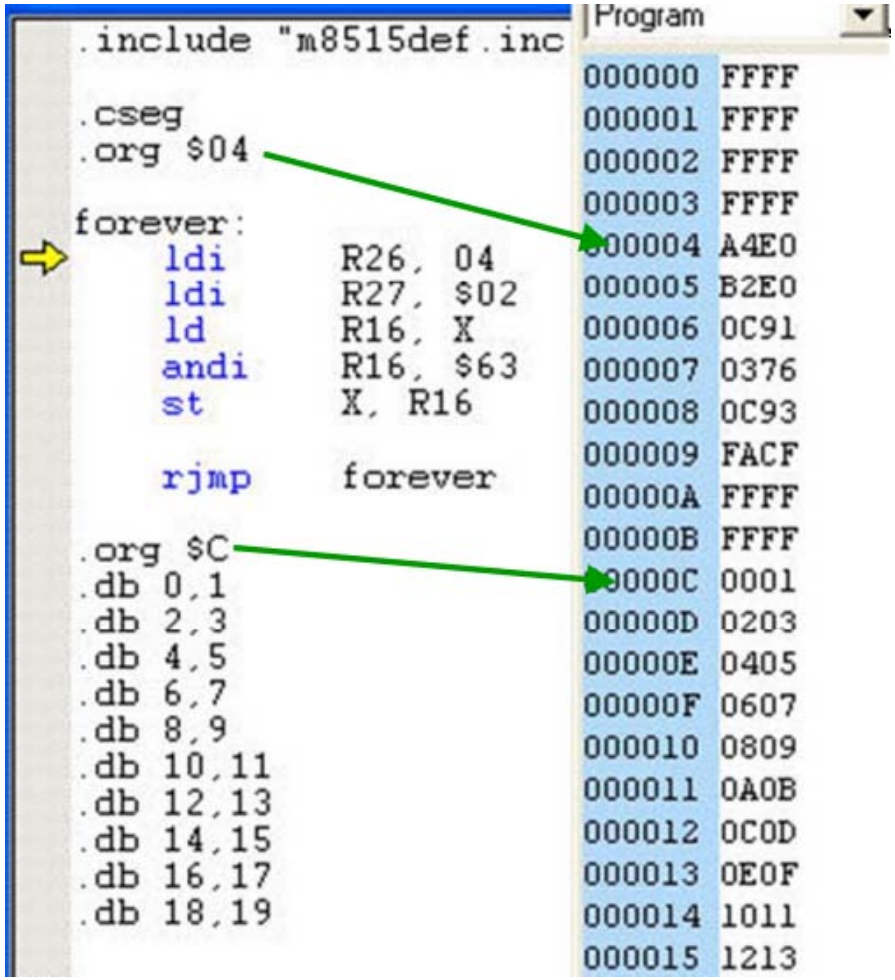
instructions

data in program memory

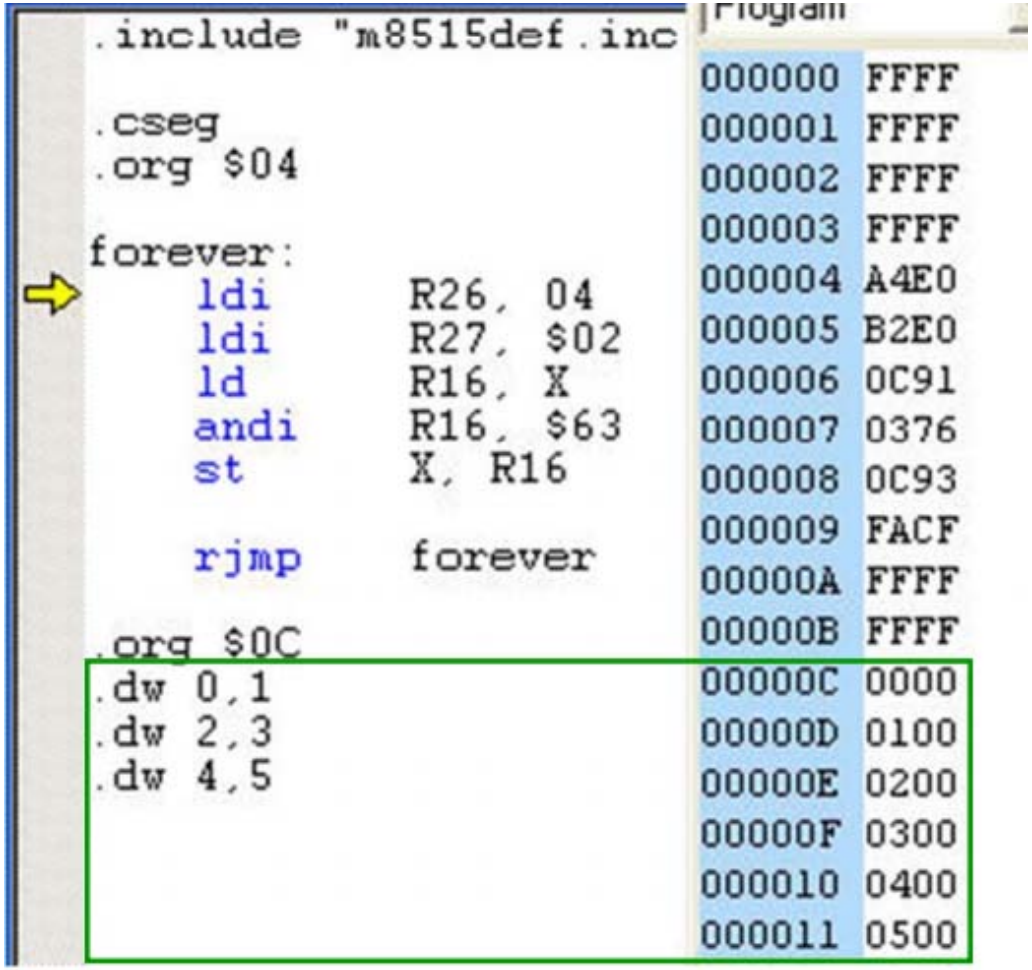

32-bit integer
0A0B0C0D
Memory
a: 0A
a+1: 0B
a+2: 0C
a+3: 0D
Big-endian


32-bit integer
0A0B0C0D
Memory
a: 0D
a+1: 0C
a+2: 0B
a+3: 0A
Little-endian

Directive – Origin (.org)



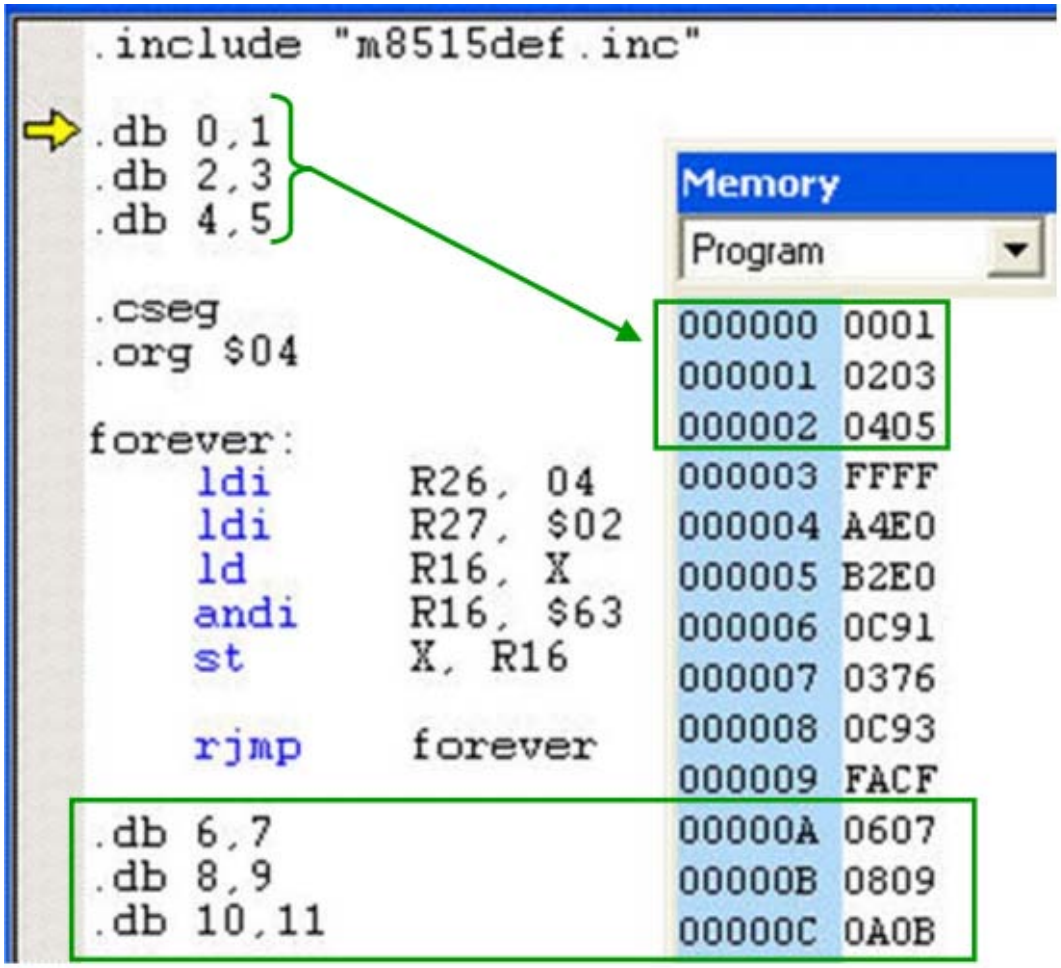
Directive – Define Word (.dw)



```
.include "m8515def.inc

.cseg
.org $04

forever:
    ldi     R26, 04
    ldi     R27, $02
    ld      R16, X
    andi    R16, $63
    st      X, R16

    rjmp    forever

.org $0C
.dw 0,1
.dw 2,3
.dw 4,5
```

```
Program
000000 FFFF
000001 FFFF
000002 FFFF
000003 FFFF
000004 A4E0
000005 B2E0
000006 0C91
000007 0376
000008 0C93
000009 FACF
00000A FFFF
00000B FFFF
00000C 0000
00000D 0100
00000E 0200
00000F 0300
000010 0400
000011 0500
```

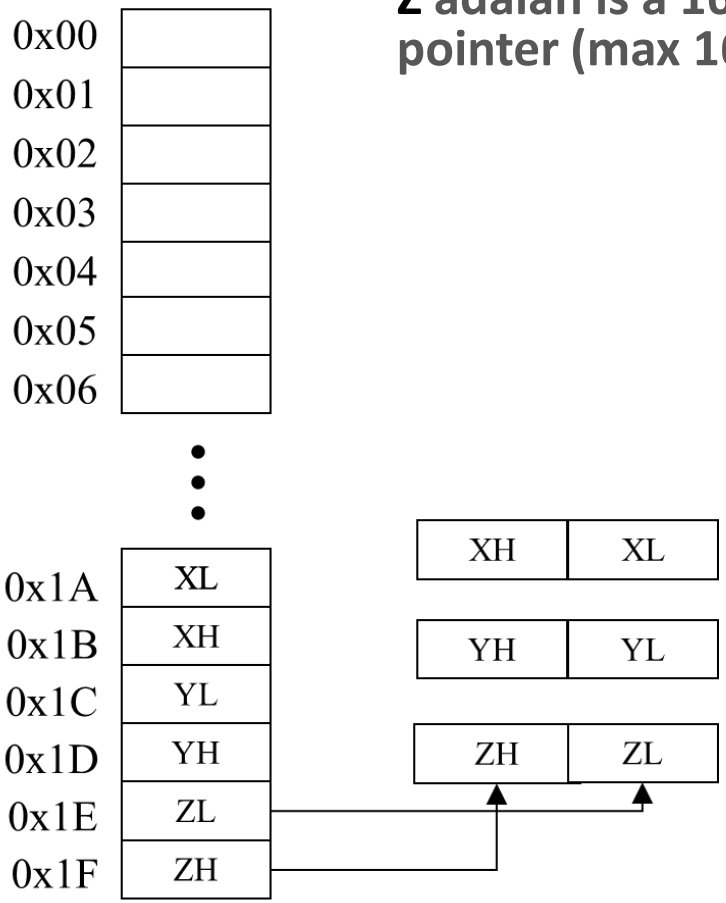
Define Constant Byte (.DB)

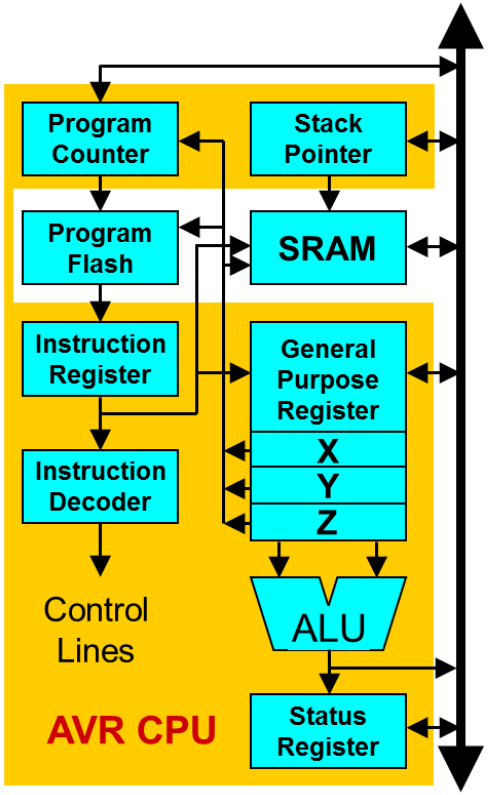


AVR Register: X, Y, Z

X, Y, Z are 16-bit pointer registers, used as address pointers (max 16 bits) to access SRAM

Z adalah is a 16-bit pointer register, used as an address pointer (max 16 bits) to access program memory

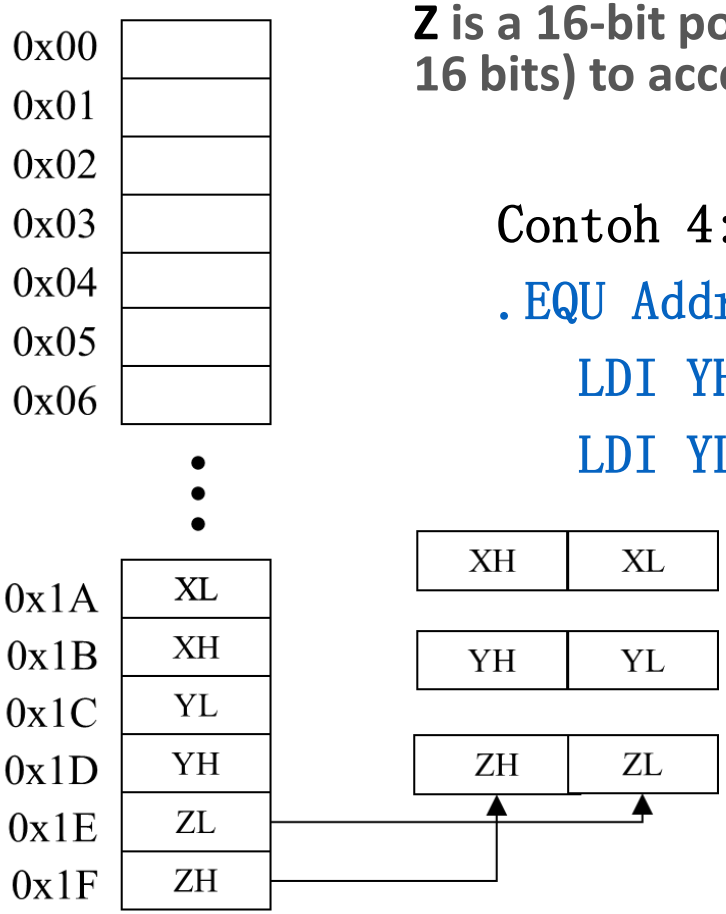




AVR Register: X, Y, Z

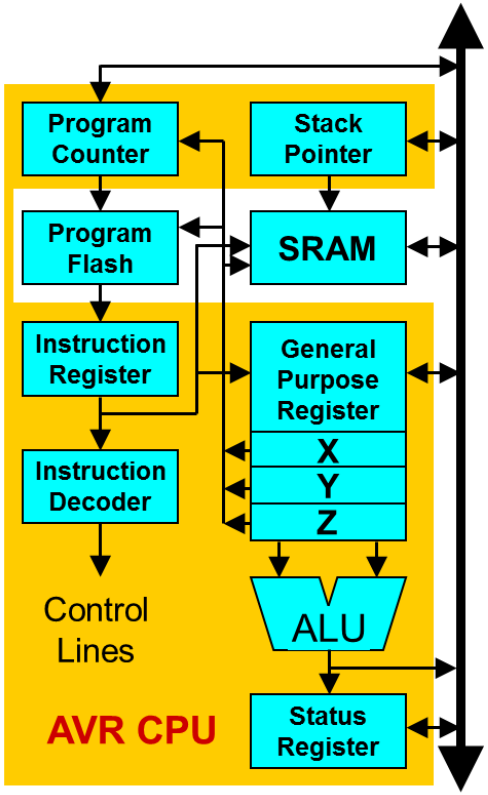
X, Y, Z are 16-bit pointer registers, used as address pointers (max 16 bits) to access SRAM

Z is a 16-bit pointer register, used as an address pointer (max 16 bits) to access program memory



Contoh 4:

```
.EQU Address = RAMEND
    LDI YH, HIGH(Address)
    LDI YL, LOW(Address)
```

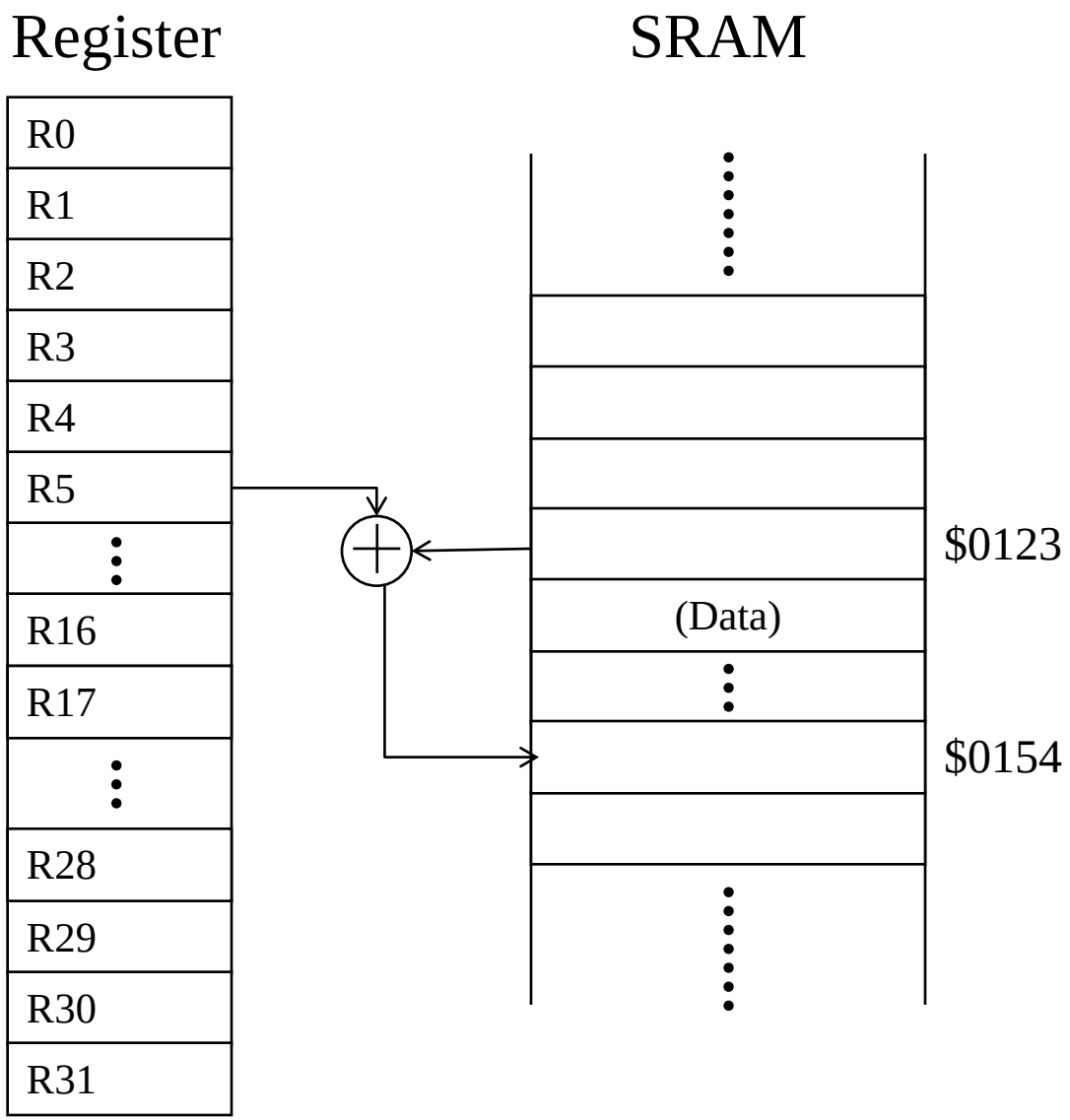


AVR Register

Pointer	Sequence	Examples
X	Read/Write from/to address X, and keep the value of pointer X	LD R1,X ST X,R1
X+	Read/Write from/to address X, and increment by “1” the value of pointer X afterwards	LD R1,X+ ST X+,R1
-X	Decrement by “1” the value of pointer X and then Read/Write from/to using the new address X	LD R1,-X ST -X,R1

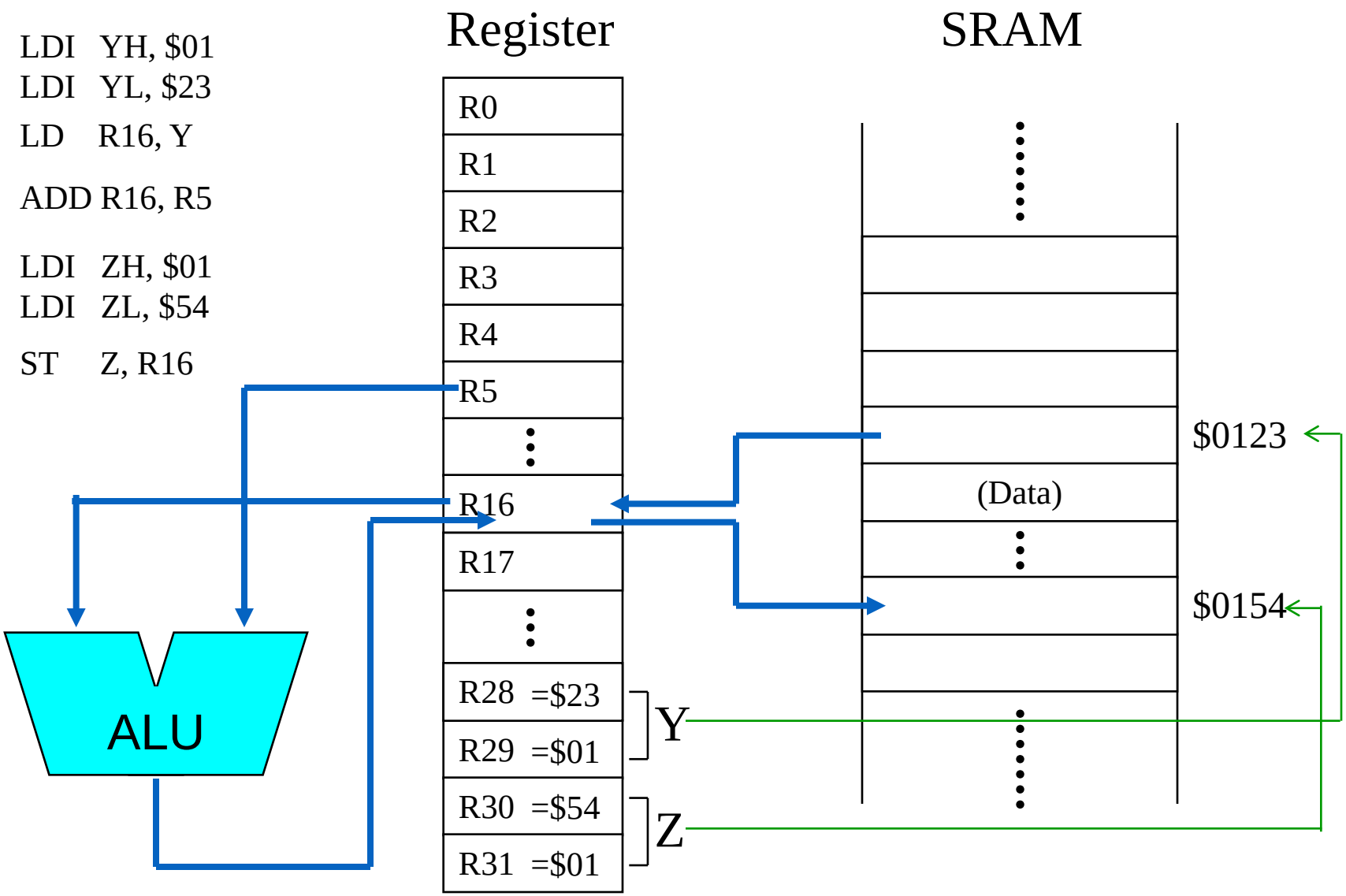
Read/Write from/to address X

Add the content of R5 with the data from memory address $0123.
Store the result in the memory address $0154.

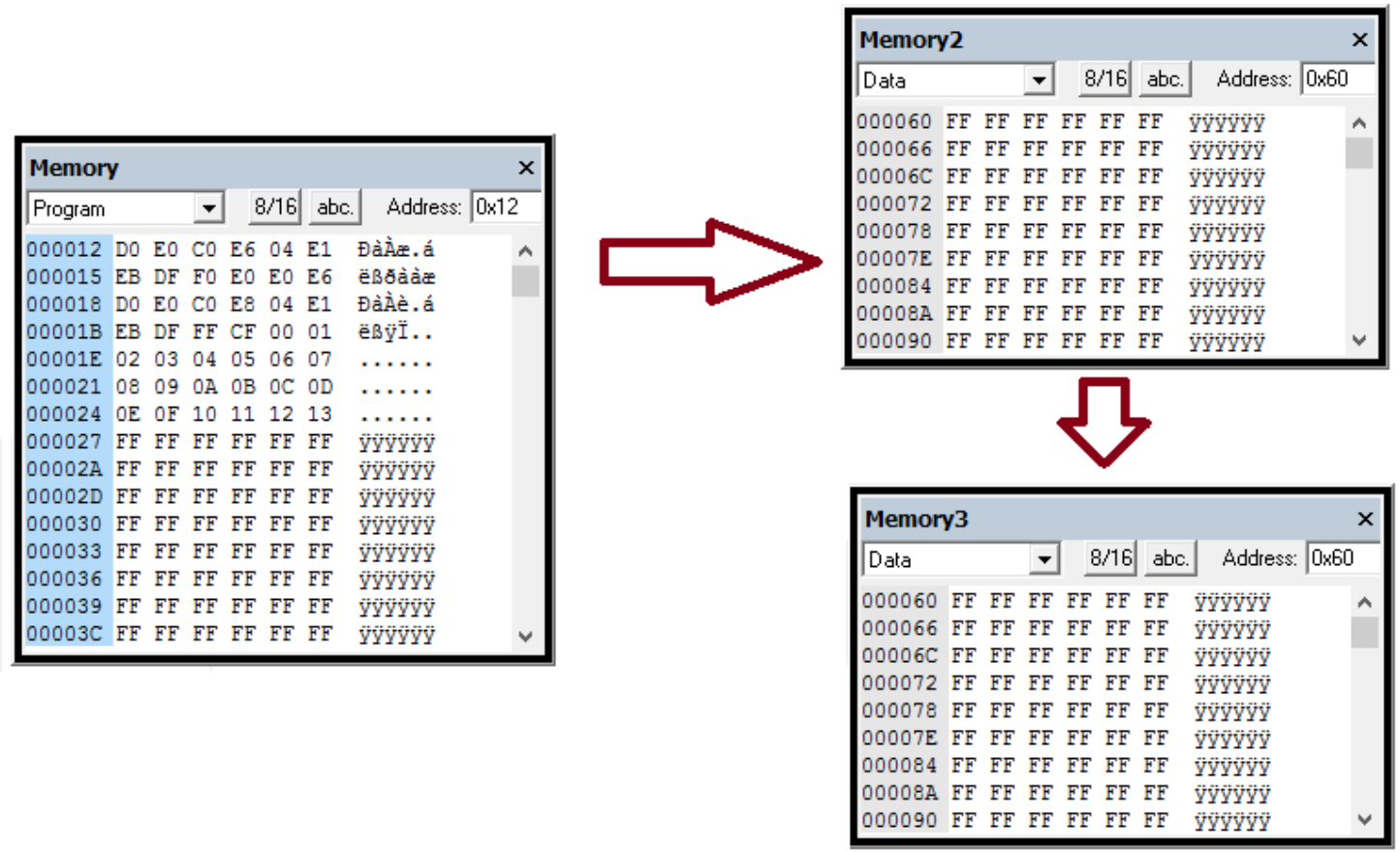


Using Data from SRAM

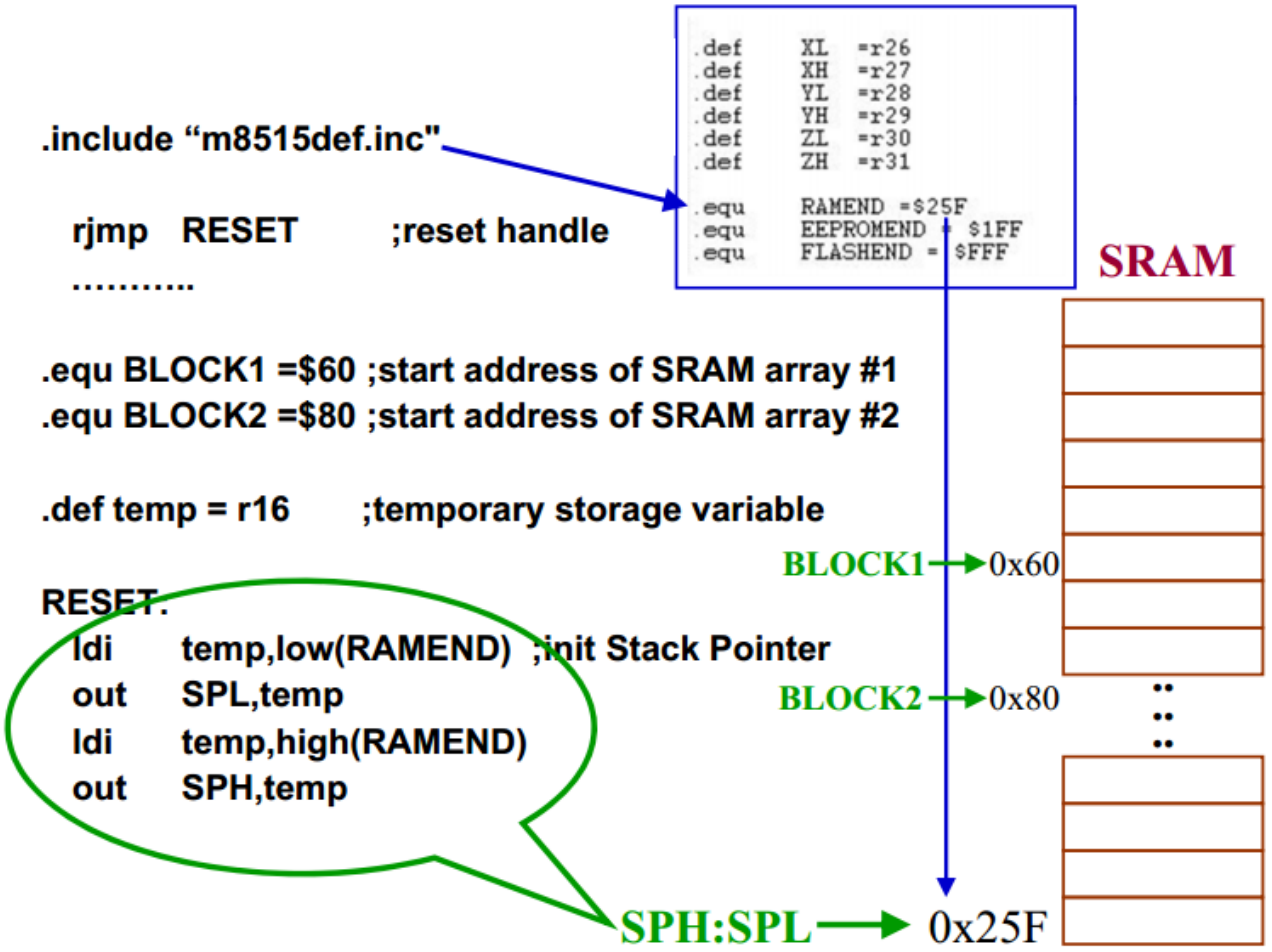
```
LDI   YH, $01
LDI   YL, $23
LD    R16, Y

ADD R16, R5

LDI   ZH, $01
LDI   ZL, $54

ST    Z, R16
```



Sample program: avr102.asm



Init Stack Pointer



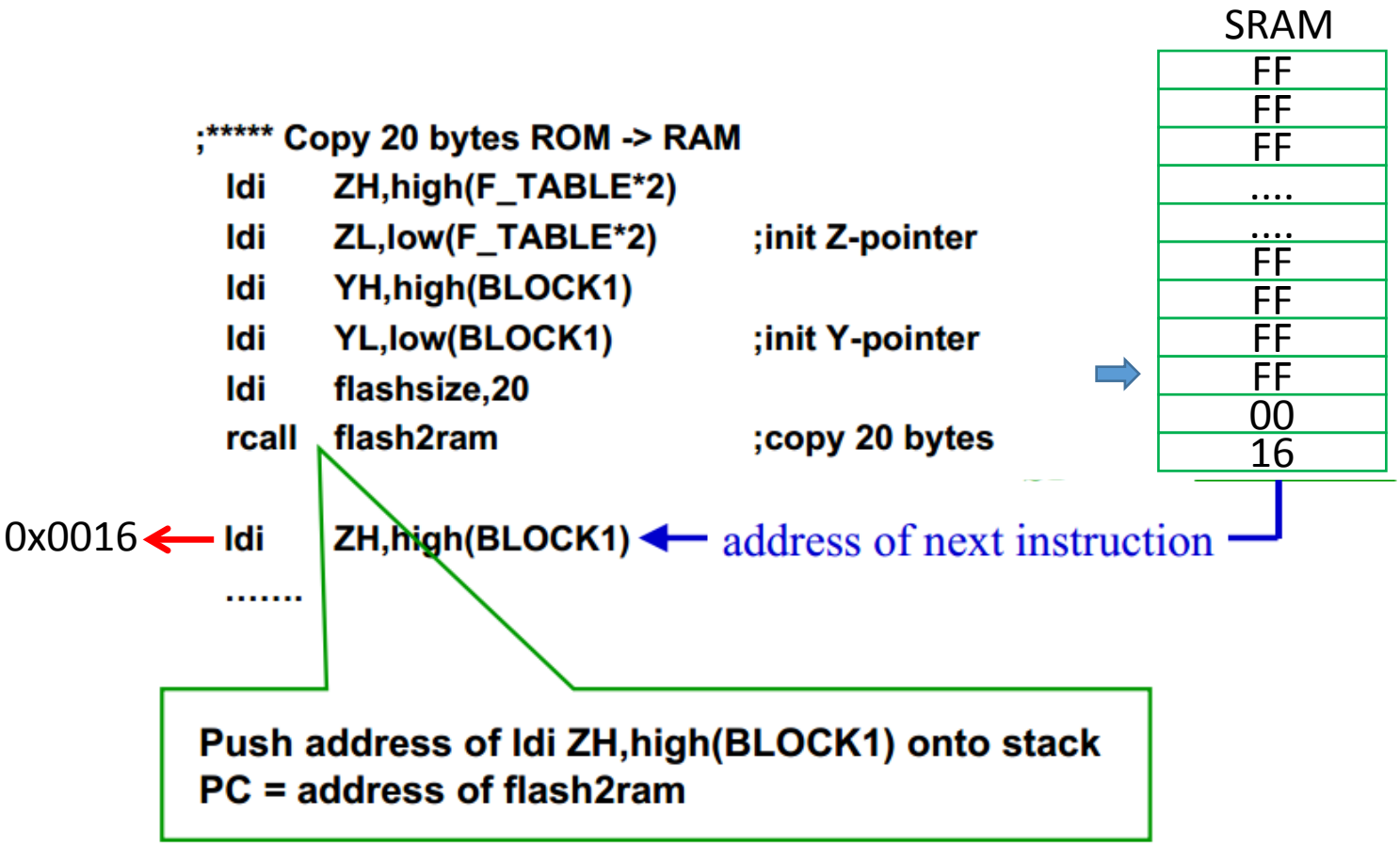
OUT instruction formation

The OUT is a 2-byte (16-bit) instruction. Of the 16 bits, the first 5 bits are set aside for the opcode, and the other 11 bits are used for the address of the source memory location and destination register. This is shown below.

```
OUT A, Rr  ;Store register Rr in I/O memory location A
```

$0 \leq d \leq 31, 0 \leq A \leq 63$

Copy Program Memory to Data Memory



Copy Program Memory to Data Memory (cont.)

```
;***** Subroutine Register variables
.def    flashsize=r16   ;size of block to be copied

flash2ram:
  lpm                   ;get constant
  st    Y+,r0           ;store in SRAM and increment Y-pointer
  adiw  ZL,1            ;increment Z-pointer
  dec   flashsize
  brne  flash2ram       ;if not end of table, loop more
  ret
```

PC = Pop(stack)
- Copy the value pointed by TOS to PC
- Increment TOS

LPM – Load Program Memory

Description:

Loads one byte pointed to by the Z-register into the destination register Rd. This instruction features a 100% space effective constant initialization or constant data fetch. The Program memory is organized in 16-bit words while the Z-pointer is a byte address. Thus, the least significant bit of the Z-pointer selects either low byte (Z_{LSB} = 0) or high byte (Z_{LSB} = 1). This instruction can address the first 64K bytes (32K words) of Program memory. The Z-pointer Register can either be left unchanged by the operation, or it can be incremented. The incrementation does not apply to the RAMPZ Register.

Devices with Self-Programming capability can use the LPM instruction to read the Fuse and Lock bit values. Refer to the device documentation for a detailed description.

Not all variants of the LPM instruction are available in all devices. Refer to the device specific instruction set summary. The LPM instruction is not implemented at all in the AT90S1200 device.

The result of these combinations is undefined:

```
LPM r30, Z+
LPM r31, Z+
```

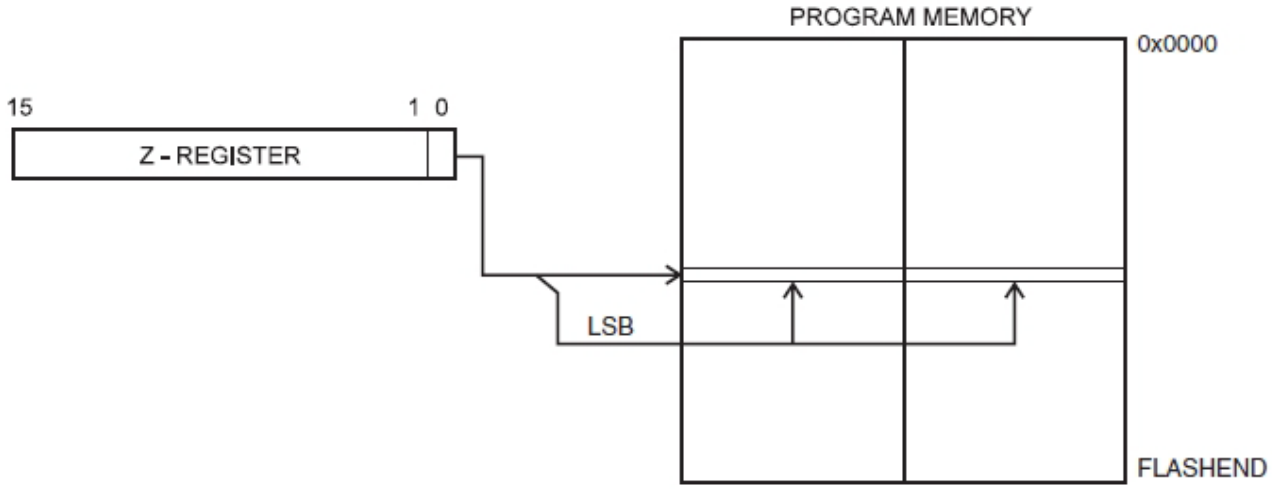
	Operation:		Comment:
(i)	R0 ← (Z)		Z: Unchanged, R0 implied destination register
(ii)	Rd ← (Z)		Z: Unchanged
(iii)	Rd ← (Z)	Z ← Z + 1	Z: Post incremented

	Syntax:	Operands:	Program Counter:
(i)	LPM	None, R0 implied	PC ← PC + 1
(ii)	LPM Rd, Z	0 ≤ d ≤ 31	PC ← PC + 1
(iii)	LPM Rd, Z+	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

(i)	1001	0101	1100	1000
(ii)	1001	000d	dddd	0100
(iii)	1001	000d	dddd	0101

Figure 9. Program Memory Constant Addressing



Constant byte address is specified by the Z-register contents. The 15 MSBs select word address. For LPM, the LSB selects low byte if cleared (LSB = 0) or high byte if set (LSB = 1). For SPM, the LSB should be cleared. If ELPM is used, the RAMPZ Register is used to extend the Z-register.

ADIW – Add Immediate to Word

Description:

Adds an immediate value (0 - 63) to a register pair and places the result in the register pair. This instruction operates on the upper four register pairs, and is well suited for operations on the pointer registers.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) Rd+1:Rd ← Rd+1:Rd + K

	Syntax:	Operands:	Program Counter:
(i)	ADIW Rd+1:Rd,K	d ∈ {24,26,28,30}, 0 ≤ K ≤ 63	PC ← PC + 1

16-bit Opcode:

1001	0110	KKdd	KKKK

BRNE – Branch if Not Equal

Description:

Conditional relative branch. Tests the Zero Flag (Z) and branches relatively to PC if Z is cleared. If the instruction is executed immediately after any of the instructions CP, CPI, SUB or SUBI, the branch will occur if and only if the unsigned or signed binary number represented in Rd was not equal to the unsigned or signed binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 1,k).

Operation:

(i) If Rd ≠ Rr (Z = 0) then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRNE k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	01kk	kkkk	k001

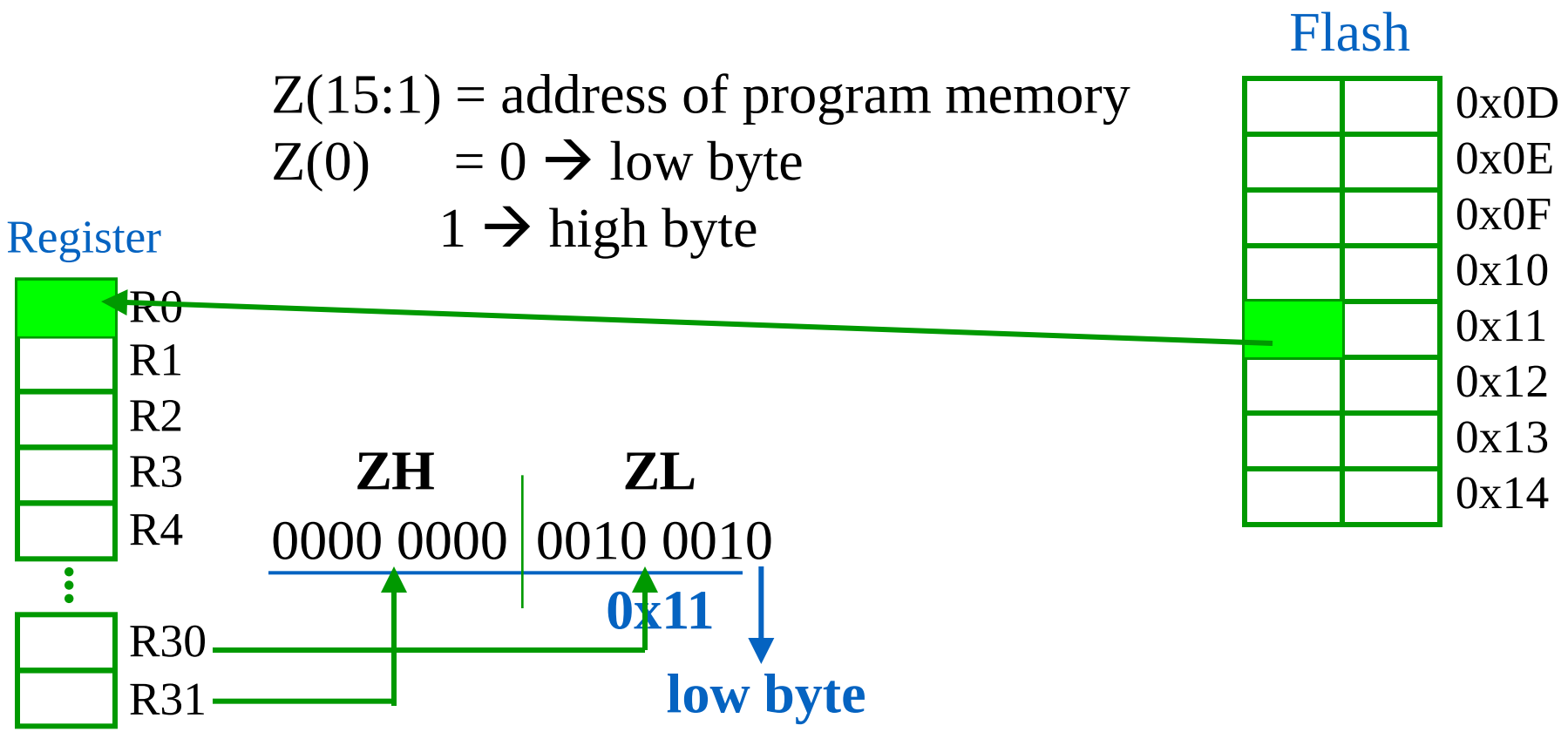
Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Copy Data Memory to Data Memory

```
;***** Copy 20 bytes RAM -> RAM
  ldi   ZH,high(BLOCK1)
  ldi   ZL,low(BLOCK1)   ;init Z-pointer
  ldi   YH,high(BLOCK2)  ;
  ldi   YL,low(BLOCK2)   ;init Y-pointer
  ldi   ramsize,20
  rcall ram2ram          ;copy 20 bytes

forever:
  rjmp forever       ;eternal loop
```

Z Pointer ke Program Memory



Low byte of program memory's content at address 0x11 is copied to R0

```
.def  ramtemp =r1    ;temporary storage register
.def  ramsize =r16     ;size of block to be copied

;***** Code

ram2ram:
  ld  ramtemp,Z+   ;get data from BLOCK1
  st  Y+,ramtemp   ;store data to BLOCK2
  dec ramsize    ;
  brne  ram2ram    ;if not done, loop more
  ret
```