

The background of the slide is a dark, stylized image of a computer circuit board. It features various traces, pads, and circular components, rendered in shades of gray and black, creating a technical and modern aesthetic.

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 yang akan disimpan pada program memory	.db "1,2"
.macro & .endmacro	Menandai dimulai & selesainya MACRO	.macro NAMAMACROendmacro
.org	Mendefinisikan alamat penyimpanan baris kode pada alamat program memory tertentu	

Directive – Code Segment (.cseg)

The screenshot displays the AVR Studio interface with three main windows:

- Processor Window:** Shows system variables. The **Cycle Counter** is highlighted with a red value of **31**. A green arrow points from the text below to this value.
- Source Code Window:** Contains the following assembly code:

```
.include "m8515def.inc"

.cseg
.org $1F

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

    rjmp   forever
```

A yellow arrow points to the **forever:** label, and a red arrow points from the **.org \$1F** directive to the memory address 0001F in the Memory window.
- Memory Window:** Shows the program memory layout. The address **0001F** is highlighted, corresponding to the value **A4E0**.

31 clock needed to reach 1st instruction

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

The screenshot shows the AVR Studio interface with the following components:

- Processor Window:** Displays system registers. The Program Counter is 0x000022, Stack Pointer is 0x0000, X pointer is 0x0204, Y pointer is 0x0000, Z pointer is 0x0000, and Cycle Counter is 35.
- Register Window:** Shows the state of all 20 registers. R16 is highlighted in red and contains the value 0x20. A green arrow points from the text 'R16 = [0x204]' to this register.
- Code Editor:** Contains the assembly code:

```
.include "m8515def.inc"

.cseg
.org $1F

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

    rjmp   forever
```

A yellow arrow points from the 'ld R16, X' instruction to the Register window, and a green arrow points from the 'rjmp forever' instruction to the Memory window.
- Memory Window:** Shows the memory layout starting at address 0001F8. Address 000204 contains the value 2056. A green arrow points from the text 'X = 0x204' to this memory location.

Handwritten green text annotations:

- X = 0x204** (with an arrow pointing to memory address 000204)
- R16 = [0x204]** (with an arrow pointing to register R16)

Click & update manual

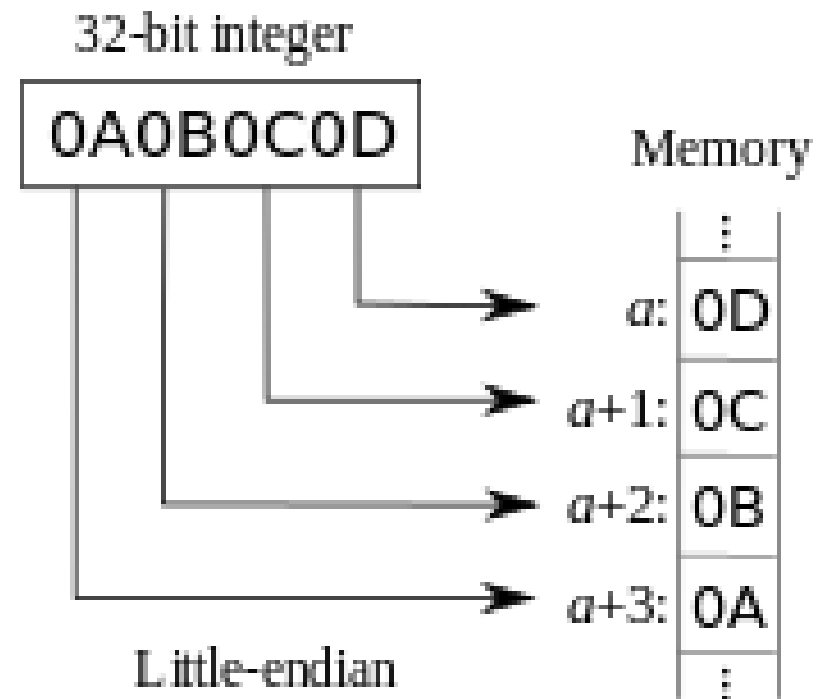
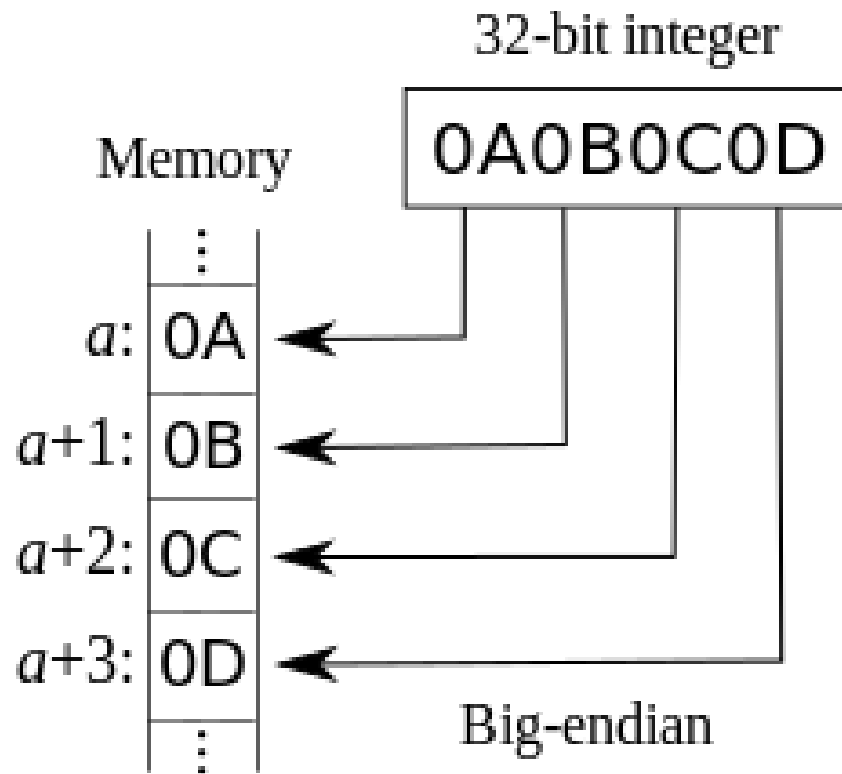
Define Constant Byte (.db)

<pre>.include "mcs51def.inc" .cseg .org \$04 forever: ldi R26, 04 ldi R27, \$02 ld R16, X andi R16, \$63 st X, R16 rjmp forever</pre>		Program
		000000 FFFF
		000001 FFFF
		000002 FFFF
		000003 FFFF
		000004 A4E0
		000005 B2E0
		000006 0C91
		000007 0376
		000008 0C93
		000009 FADF
table:		00000A 0001
.db 0,1		00000B 0203
.db 2,3		00000C 0405
.db 4,5		00000D 0607
.db 6,7		00000E 0809
.db 8,9		00000F 0A0B
.db 10,11		000010 0C0D
.db 12,13		000011 0E0F
.db 14,15		000012 1011
.db 16,17		000013 1213
.db 18,19		000014 FFFF

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

instructions

data in program memory



Directive – Origin (.org)

Program	
.include "m8515def.inc"	000000 FFFF
.cseg	000001 FFFF
.org \$04	000002 FFFF
forever:	000003 FFFF
→ ldi R26, 04	000004 A4E0
ldi R27, \$02	000005 B2E0
ld R16, X	000006 0C91
andi R16, \$63	000007 0376
st X, R16	000008 0C93
rjmp forever	000009 FACF
	00000A FFFF
.org \$C	00000B FFFF
.db 0,1	00000C 0001
.db 2,3	00000D 0203
.db 4,5	00000E 0405
.db 6,7	00000F 0607
.db 8,9	000010 0809
.db 10,11	000011 0A0B
.db 12,13	000012 0C0D
.db 14,15	000013 0E0F
.db 16,17	000014 1011
.db 18,19	000015 1213

Directive – Define Word (.dw)

Program	
.include "m8515def.inc"	000000 FFFF
.cseg	000001 FFFF
.org \$04	000002 FFFF
forever:	000003 FFFF
→ ldi R26, 04	000004 A4E0
ldi R27, \$02	000005 B2E0
ld R16, X	000006 0C91
andi R16, \$63	000007 0376
st X, R16	000008 0C93
rjmp forever	000009 FACF
	00000A FFFF
.org \$0C	00000B FFFF
.dw 0,1	00000C 0000
.dw 2,3	00000D 0100
.dw 4,5	00000E 0200
	00000F 0300
	000010 0400
	000011 0500

Define Constant Byte (.DB)

The screenshot displays an assembly editor with the following code:

```
.include "m8515def.inc"

→ .db 0,1
   .db 2,3
   .db 4,5

.cseg
.org $04

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

    rjmp   forever

.db 6,7
.db 8,9
.db 10,11
```

A green bracket groups the first three `.db` instructions, with an arrow pointing to the memory window. The memory window, titled "Memory", shows the following data:

Address	Value
000000	0001
000001	0203
000002	0405
000003	FFFF
000004	A4E0
000005	B2E0
000006	0C91
000007	0376
000008	0C93
000009	FACF
00000A	0607
00000B	0809
00000C	0A0B

The first three memory locations (000000 to 000002) are highlighted with a green box, corresponding to the first three `.db` instructions. The last three `.db` instructions and their corresponding memory locations (00000A to 00000C) are also highlighted with a green box.

AVR Register: X, Y, Z

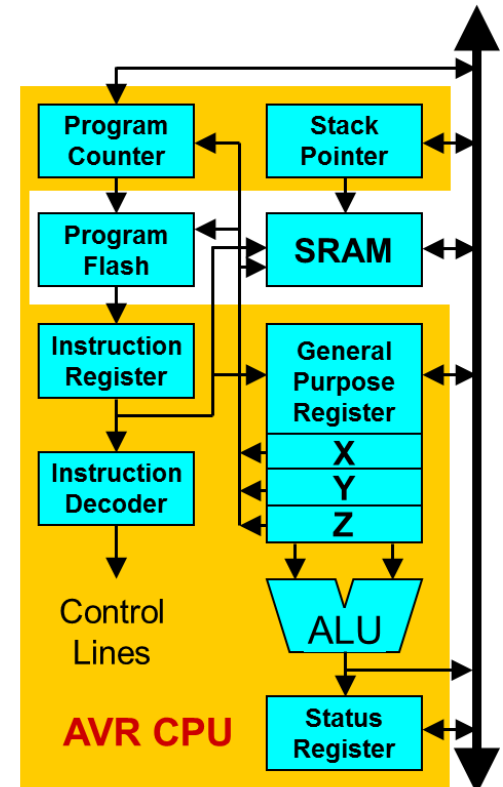
X, Y, Z are 16-bit pointer registers, used as address pointers (max 16 bits) to access SRAM

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

0x00	
0x01	
0x02	
0x03	
0x04	
0x05	
0x06	

•
•
•

0x1A	XL	XH	XL
0x1B	XH	YH	YL
0x1C	YL		
0x1D	YH	ZH	ZL
0x1E	ZL		
0x1F	ZH		



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

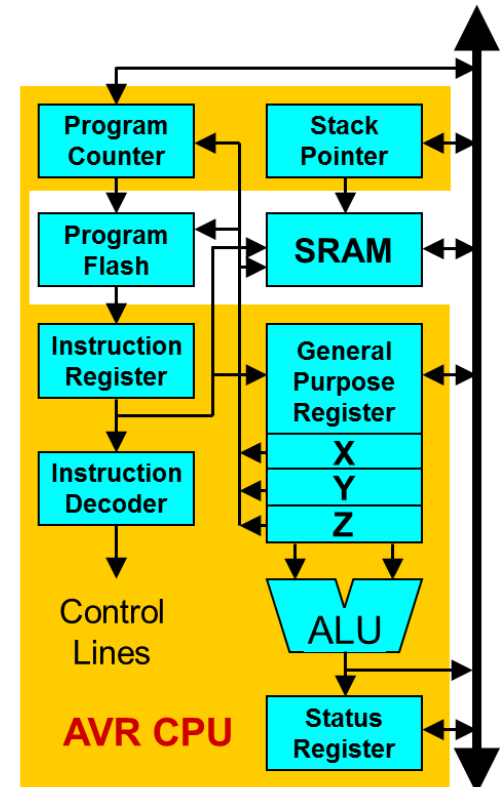
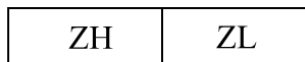
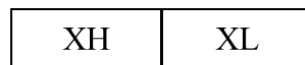
0x00	
0x01	
0x02	
0x03	
0x04	
0x05	
0x06	

⋮

0x1A	XL
0x1B	XH
0x1C	YL
0x1D	YH
0x1E	ZL
0x1F	ZH

Contoh 4:

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

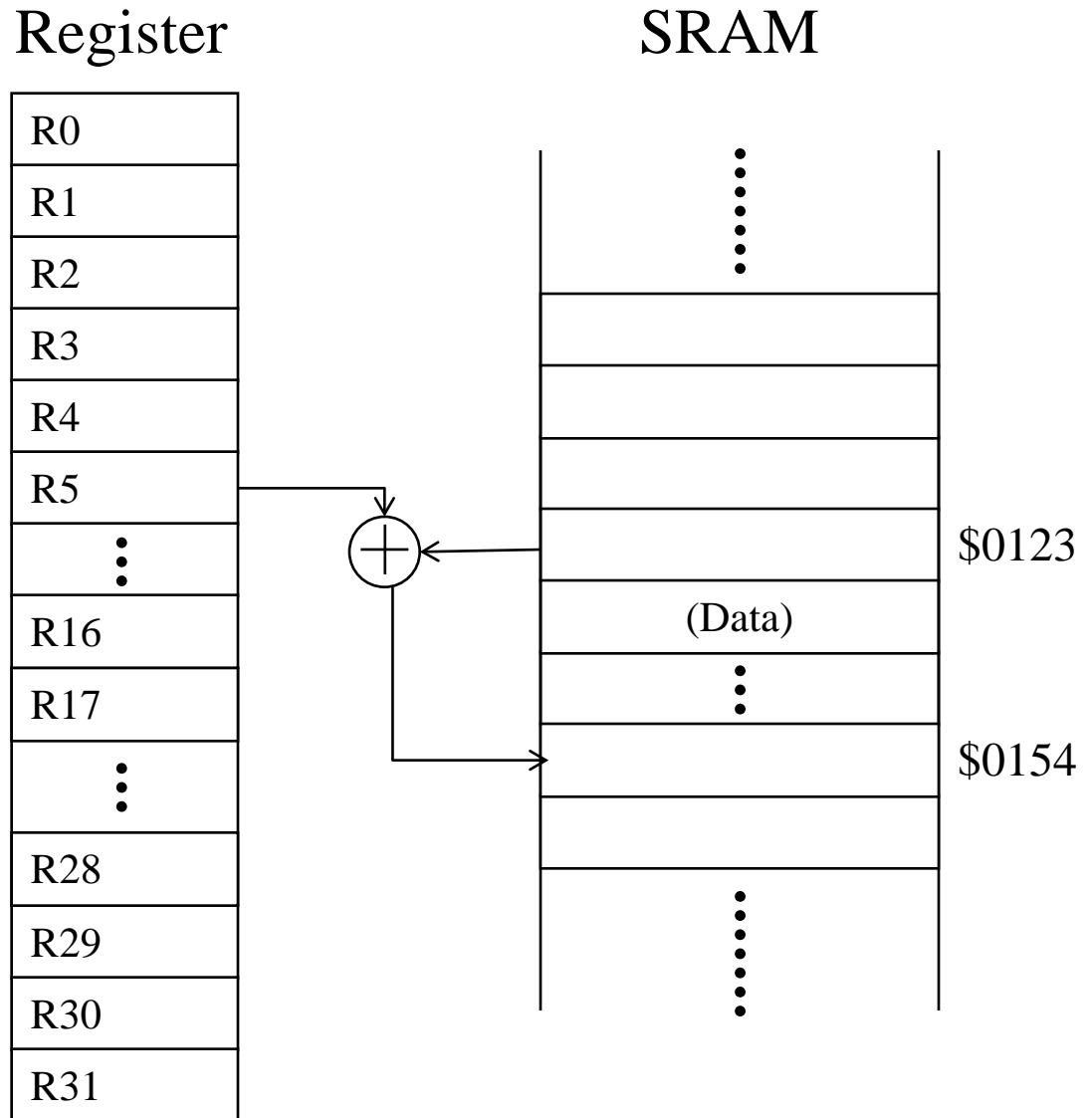


AVR Register

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

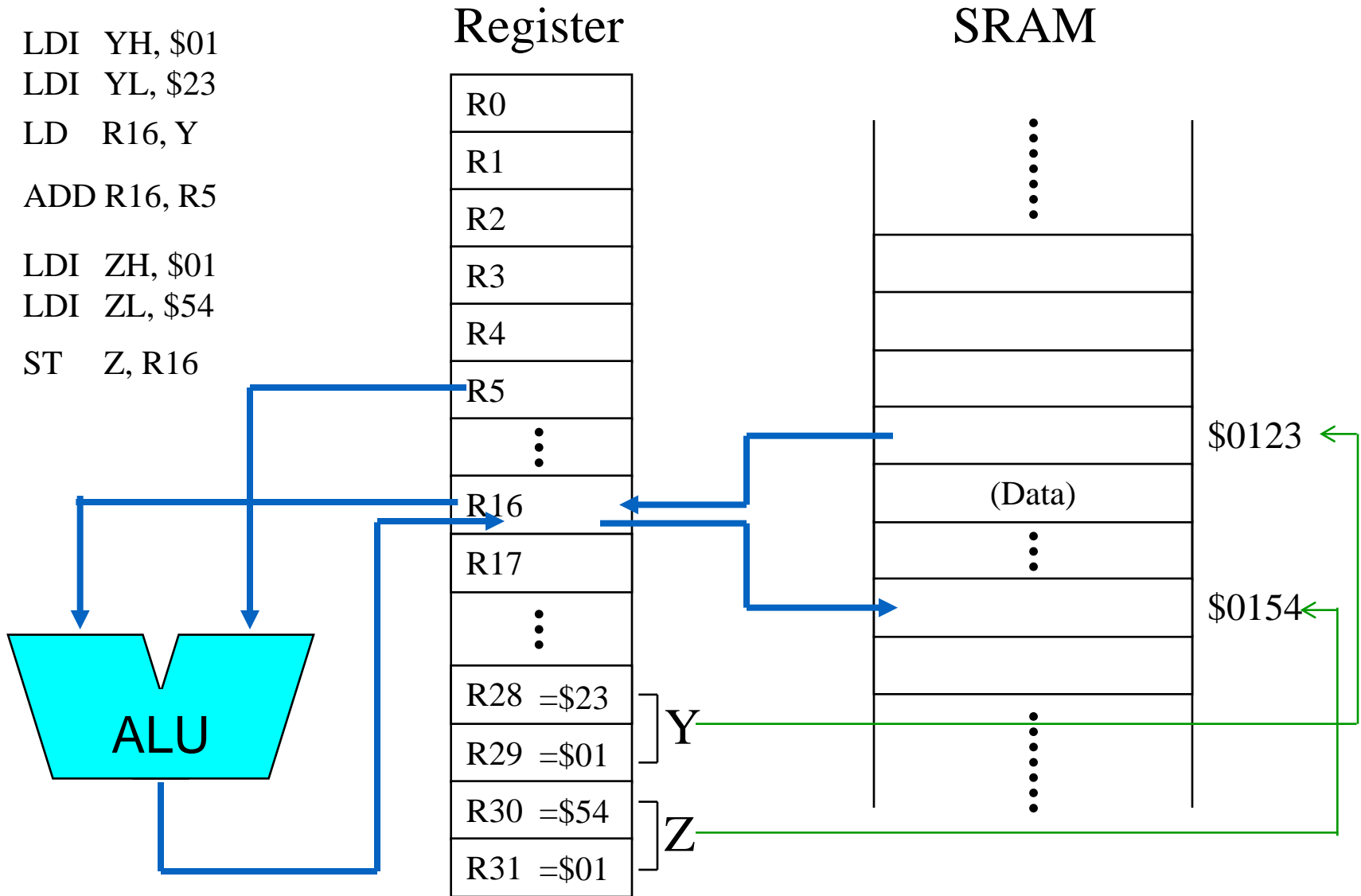
Add the content of R5 with
the data from memory
address \$0123.
Store the result in the
memory address \$0154.



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

Memory									
Program		8/16		abc.		Address: 0x12			
000012	D0 E0 C0 E6 04 E1	ÐàÀæ.á							
000015	EB DF F0 E0 E0 E6	ëßðääæ							
000018	D0 E0 C0 E8 04 E1	ÐàÀè.á							
00001B	EB DF FF CF 00 01	ëßÿĭ..							
00001E	02 03 04 05 06 07							
000021	08 09 0A 0B 0C 0D							
000024	0E 0F 10 11 12 13							
000027	FF FF FF FF FF FF	ÿÿÿÿÿÿ							
00002A	FF FF FF FF FF FF	ÿÿÿÿÿÿ							
00002D	FF FF FF FF FF FF	ÿÿÿÿÿÿ							
000030	FF FF FF FF FF FF	ÿÿÿÿÿÿ							
000033	FF FF FF FF FF FF	ÿÿÿÿÿÿ							
000036	FF FF FF FF FF FF	ÿÿÿÿÿÿ							
000039	FF FF FF FF FF FF	ÿÿÿÿÿÿ							
00003C	FF FF FF FF FF FF	ÿÿÿÿÿÿ							



Memory2									
Data		8/16		abc.		Address: 0x60			
000060	FF FF FF FF FF FF	ÿÿÿÿÿÿ							
000066	FF FF FF FF FF FF	ÿÿÿÿÿÿ							
00006C	FF FF FF FF FF FF	ÿÿÿÿÿÿ							
000072	FF FF FF FF FF FF	ÿÿÿÿÿÿ							
000078	FF FF FF FF FF FF	ÿÿÿÿÿÿ							
00007E	FF FF FF FF FF FF	ÿÿÿÿÿÿ							
000084	FF FF FF FF FF FF	ÿÿÿÿÿÿ							
00008A	FF FF FF FF FF FF	ÿÿÿÿÿÿ							
000090	FF FF FF FF FF FF	ÿÿÿÿÿÿ							



Memory3									
Data		8/16		abc.		Address: 0x60			
000060	FF FF FF FF FF FF	ÿÿÿÿÿÿ							
000066	FF FF FF FF FF FF	ÿÿÿÿÿÿ							
00006C	FF FF FF FF FF FF	ÿÿÿÿÿÿ							
000072	FF FF FF FF FF FF	ÿÿÿÿÿÿ							
000078	FF FF FF FF FF FF	ÿÿÿÿÿÿ							
00007E	FF FF FF FF FF FF	ÿÿÿÿÿÿ							
000084	FF FF FF FF FF FF	ÿÿÿÿÿÿ							
00008A	FF FF FF FF FF FF	ÿÿÿÿÿÿ							
000090	FF FF FF FF FF FF	ÿÿÿÿÿÿ							

Init Stack Pointer

```
.include "m8515def.inc"
```

```
  rjmp  RESET      ;reset handle
```

```
  .....
```

```
.equ BLOCK1 =$60 ;start address of SRAM array #1
```

```
.equ BLOCK2 =$80 ;start address of SRAM array #2
```

```
.def temp = r16      ;temporary storage variable
```

```
RESET:
```

```
  ldi   temp,low(RAMEND) ;init Stack Pointer
```

```
  out   SPL,temp
```

```
  ldi   temp,high(RAMEND)
```

```
  out   SPH,temp
```

```
.def  XL  =r26
.def  XH  =r27
.def  YL  =r28
.def  YH  =r29
.def  ZL  =r30
.def  ZH  =r31

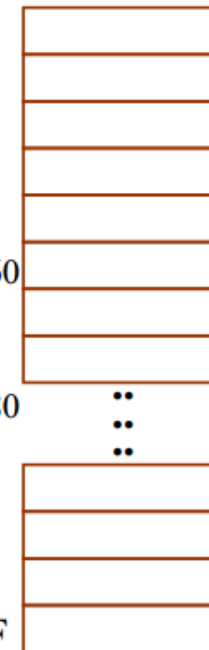
.equ  RAMEND  =$25F
.equ  EEPROMEND  = $1FF
.equ  FLASHEND = $FFF
```

SRAM

BLOCK1 → 0x60

BLOCK2 → 0x80

SPH:SPL → 0x25F



OUT instruction formation

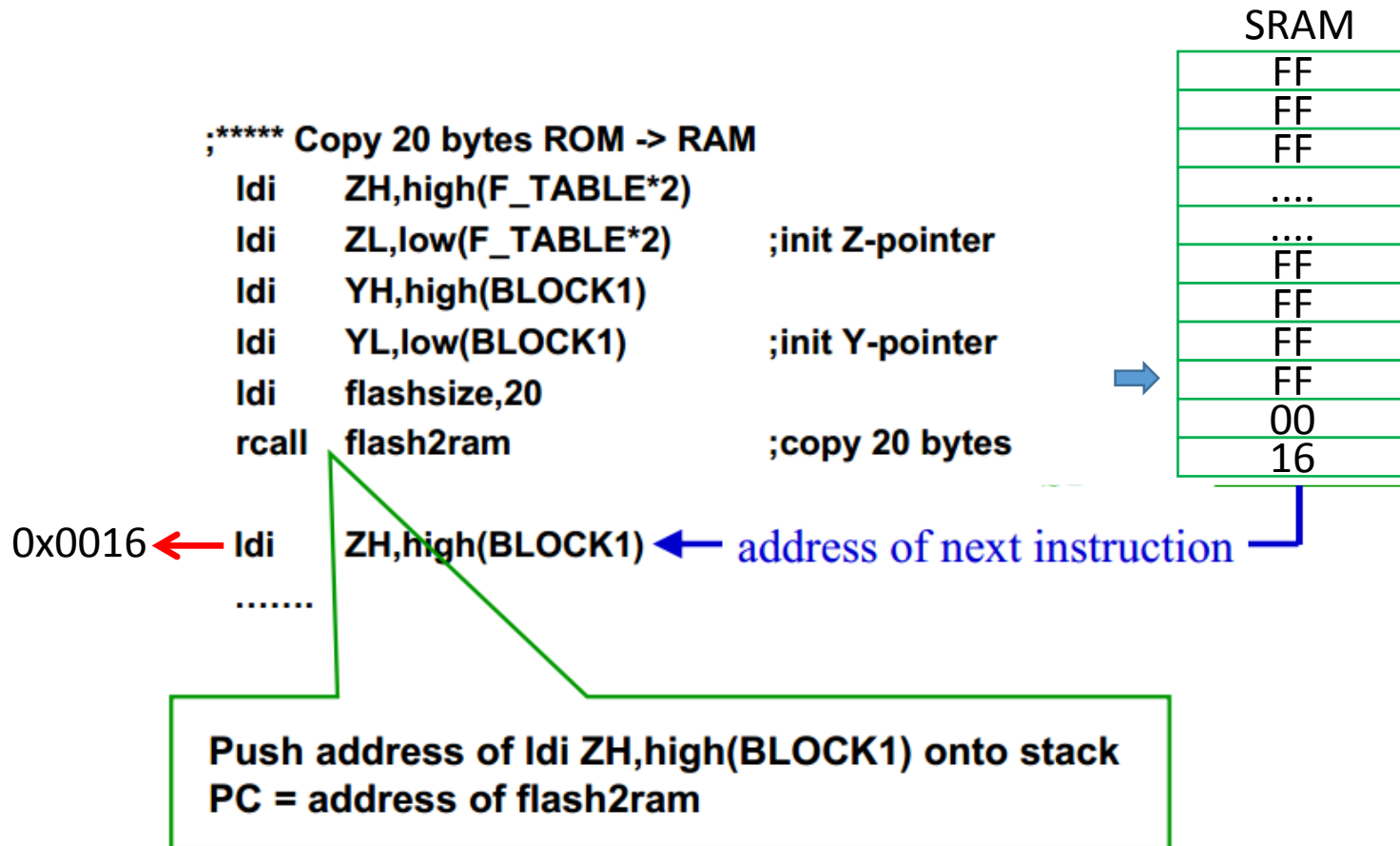
The OUT is a 2-byte (16-bit) instruction. Of the 16 bits, the first 5 bits are set aside for the opcode, and the other 11 bits are used for the address of the source memory location and destination register. This is shown below.

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:

- (i) $R0 \leftarrow (Z)$
- (ii) $Rd \leftarrow (Z)$
- (iii) $Rd \leftarrow (Z)$ $Z \leftarrow Z + 1$

Comment:

Z: Unchanged, R0 implied destination register
Z: Unchanged
Z: Post incremented

Syntax:

- (i) LPM
- (ii) LPM Rd, Z
- (iii) LPM Rd, Z+

Operands:

None, R0 implied
 $0 \leq d \leq 31$
 $0 \leq d \leq 31$

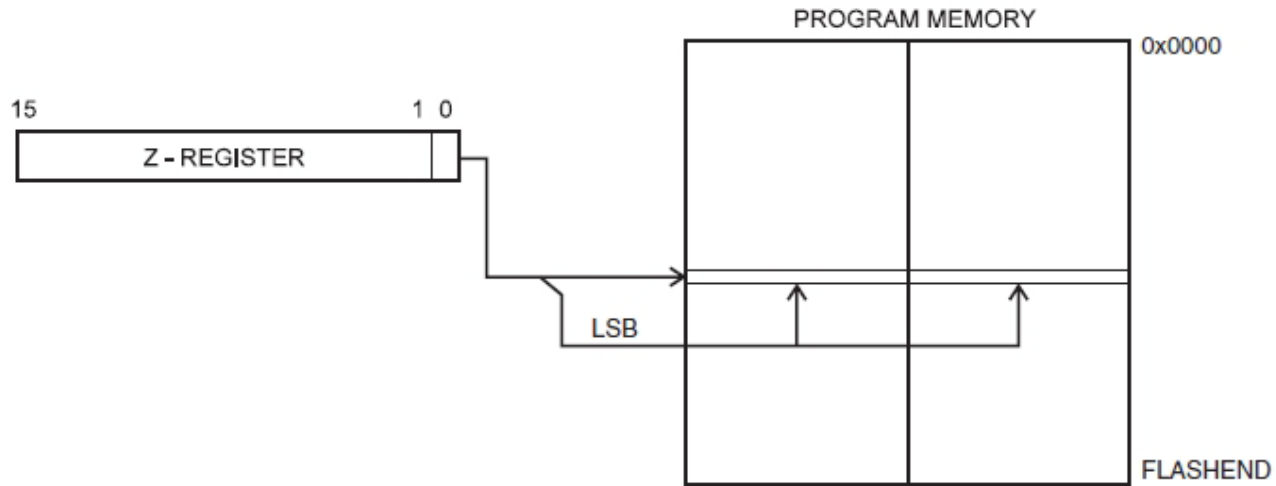
Program Counter:

$PC \leftarrow PC + 1$
 $PC \leftarrow PC + 1$
 $PC \leftarrow PC + 1$

16-bit Opcode:

(i)	1001	0101	1100	1000
(ii)	1001	000d	ddd d	0100
(iii)	1001	000d	ddd d	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) $Rd+1:Rd \leftarrow Rd+1:Rd + K$

Syntax:

- (i) ADIW Rd+1:Rd,K $d \in \{24,26,28,30\}, 0 \leq K \leq 63$

Operands:

Program Counter:

$PC \leftarrow PC + 1$

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 ($PC - 63 \leq \text{destination} \leq PC + 64$). 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 $Rd \neq Rr$ ($Z = 0$) then $PC \leftarrow PC + k + 1$, else $PC \leftarrow PC + 1$

Syntax:

- (i) BRNE k

Operands:

$$-64 \leq k \leq +63$$

Program Counter:

$$PC \leftarrow PC + k + 1$$

$$PC \leftarrow PC + 1, \text{ if condition is false}$$

16-bit Opcode:

1111	01kk	kkkk	k001
------	------	------	------

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

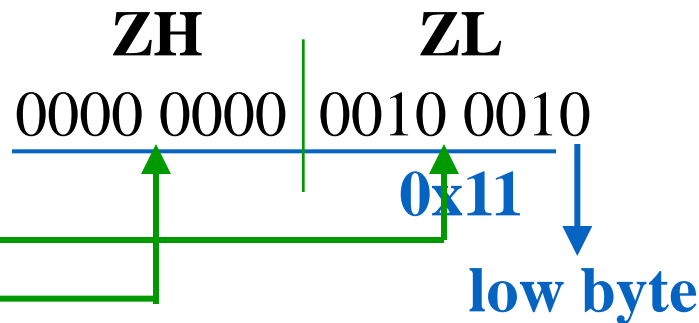
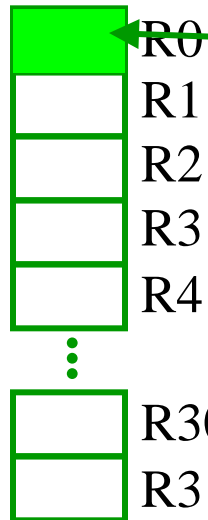
Copy Data Memory to Data Memory

```
;***** 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

$Z(15:1)$ = address of program memory
 $Z(0)$ = 0 \rightarrow low byte
1 \rightarrow high byte

Register



Flash

		0x0D
		0x0E
		0x0F
		0x10
		0x11
		0x12
		0x13
		0x14

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
```