



# AVR Memory

M. Anwar Ma'sum, Bayu Anggoroajati, Grafika Jati

# Contoh Program

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

```
forever:
```

```
    ldi R26, 04 ; 04 diartikan sebagai nilai desimal 4
```

```
    ldi R27, $02 ; $02 diartikan sebagai 02 dalam hexadecimal
```

```
    ld R16, X
```

```
    andi R16, 0x63
```

```
    st X, R16
```

```
    rjmp forever
```

0x02 = \$02

# Directive – code segment

- Beberapa Directive yang sering digunakan:

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 NAMAMACRO ..... .endmacro
.org	Mendefinisikan alamat penyimpanan baris kode pada alamat program memory tertentu	

# Directive - code segment (cont.)

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

- Processor:** Shows system variables. The **Cycle Counter** is highlighted in red and set to 31. A green arrow points from this value to the text below.
- Assembly Code:** Shows the following 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 start of the `forever` label, and a red arrow points from the `.org $1F` directive to the memory dump.
- Memory:** Shows a dump of memory addresses and values:

```
00001C FFFF
00001D FFFF
00001E FFFF
0001F A4E0
000020 B2E0
000021 0C91
000022 0376
000023 0C93
000024 FACF
000025 FFFF
```

A red arrow points to the memory address `0001F`, which corresponds to the `.org $1F` directive.

31 clock needed to reach 1<sup>st</sup> instruction

# Directive - code segment (cont.)

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

- Processor Window:** Displays system registers: Program Counter (0x000022), Stack Pointer (0x0000), X pointer (0x0204), Y pointer (0x0000), Z pointer (0x0000), Cycle Counter (35), and Frequency.
- Register Window:** Shows registers R0-R19. R16 is highlighted in red with the value 0x20. A yellow arrow points from the Register window to the code.
- Code Editor:** Contains 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
```
- Memory Window:** Shows memory addresses and values. Address 000204 contains the value 2056. A green arrow points from the memory value to the register R16.
- Annotations:** Green text indicates  $X = 0x204$  and  $R16 = [0x204]$ . A green arrow points from the memory address 000204 to the register R16.

Click & update manual

# Directive - define bytes

```
.include m0513def.inc
.cseg
.org $04
forever:
    ldi    R26, 04
    ldi    R27, $02
    ld     R16, X
    andi   R16, $63
    st     X, R16

    rjmp   forever

table:
.db 0,1
.db 2,3
.db 4,5
.db 6,7
.db 8,9
.db 10,11
.db 12,13
.db 14,15
.db 16,17
.db 18,19
```

Program	
000000	FFFF
000001	FFFF
000002	FFFF
000003	FFFF
000004	A4E0
000005	B2E0
000006	0C91
000007	0376
000008	0C93
000009	FACF
00000A	0001
00000B	0203
00000C	0405
00000D	0607
00000E	0809
00000F	0A0B
000010	0C0D
000011	0E0F
000012	1011
000013	1213
000014	FFFF

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

instructions

data in program memory

# Directive - origin

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

# Directive - define words

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



# Directive - .db dengan letak berbeda

```
.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

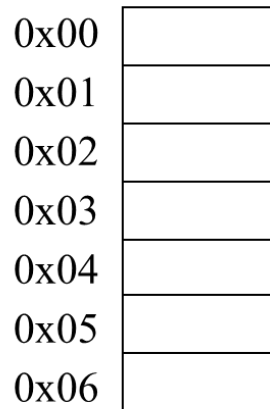
.db 6,7
.db 8,9
.db 10,11
```

Memory	
Program	
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

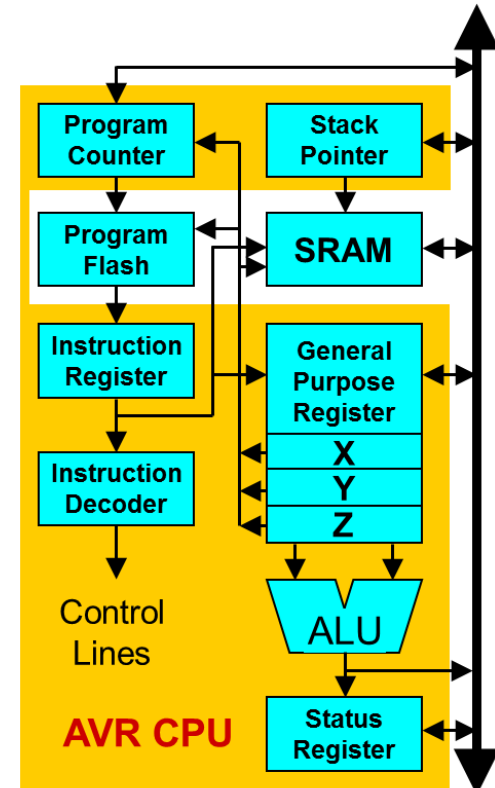
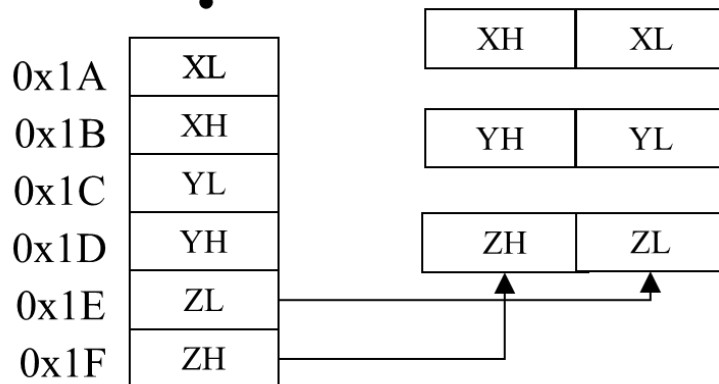
# AVR Register: X, Y, Z

X, Y, Z adalah 16-bit pointer register, digunakan untuk menunjukkan alamat dengan max 16-bit pada SRAM

Z adalah 16-bit pointer register, dapat digunakan untuk menunjukkan alamat dengan max 16-bit pada program memory



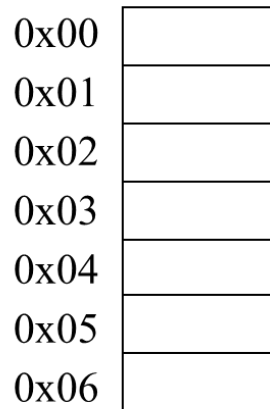
⋮



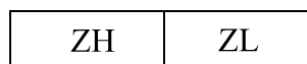
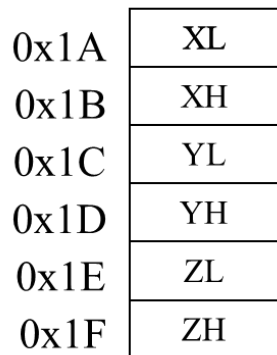
# AVR Register: X, Y, Z

X, Y, Z adalah 16-bit pointer register, digunakan untuk menunjukkan alamat dengan max 16-bit pada SRAM

Z adalah 16-bit pointer register, dapat digunakan untuk menunjukkan alamat dengan max 16-bit pada program memory



⋮

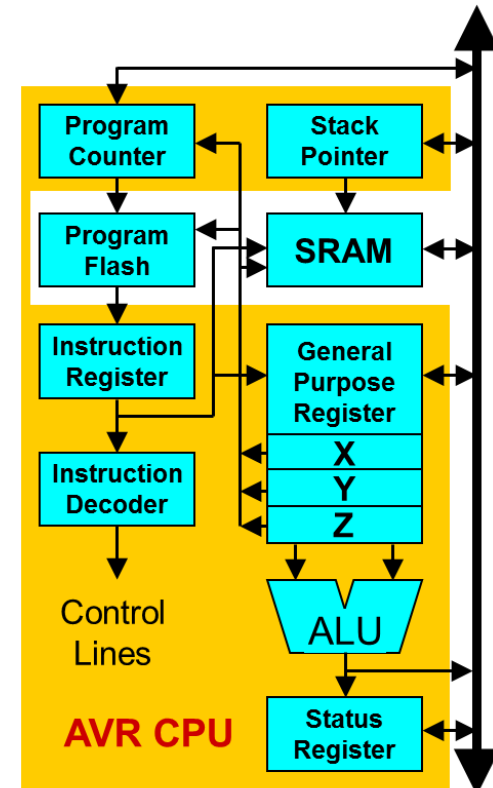


Contoh 4 :

```
.EQU Address = RAMEND
```

```
LDI YH, HIGH(Address)
```

```
LDI YL, LOW(Address)
```

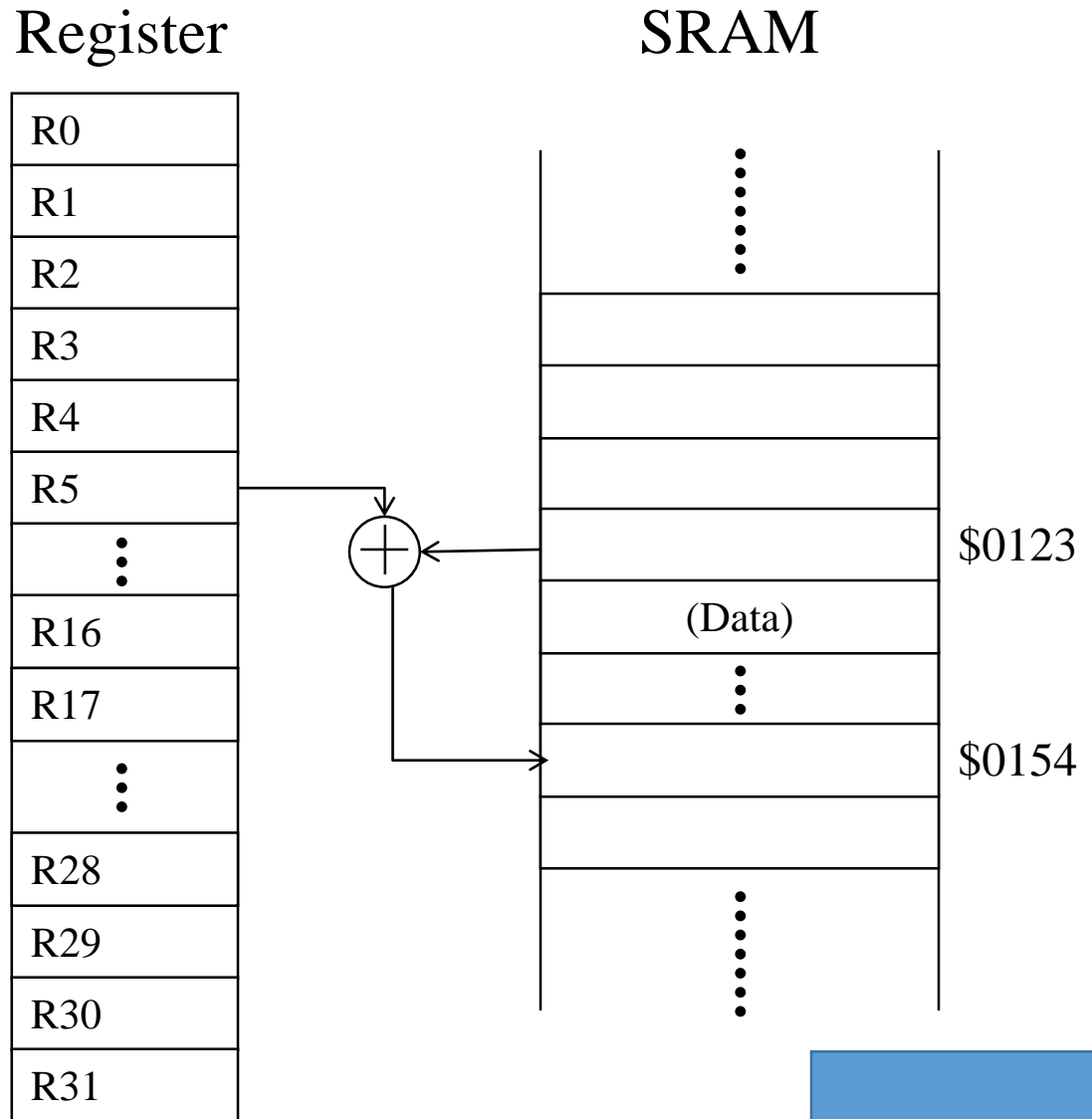


# AVR Register

Pointer	Sequence	Examples
X	Read/Write dari/ke alamat X, jangan nilai ubah pointernya	LD R1,X ST X,R1
X+	Read/Write dari/ke alamat X dan setelah itu increment satu nilai pointernya	LD R1,X+ ST X+,R1
-X	Decrement satu nilai pointer dan kemudian read/write dari/ke alamat yang baru	LD R1,-X ST -X,R1

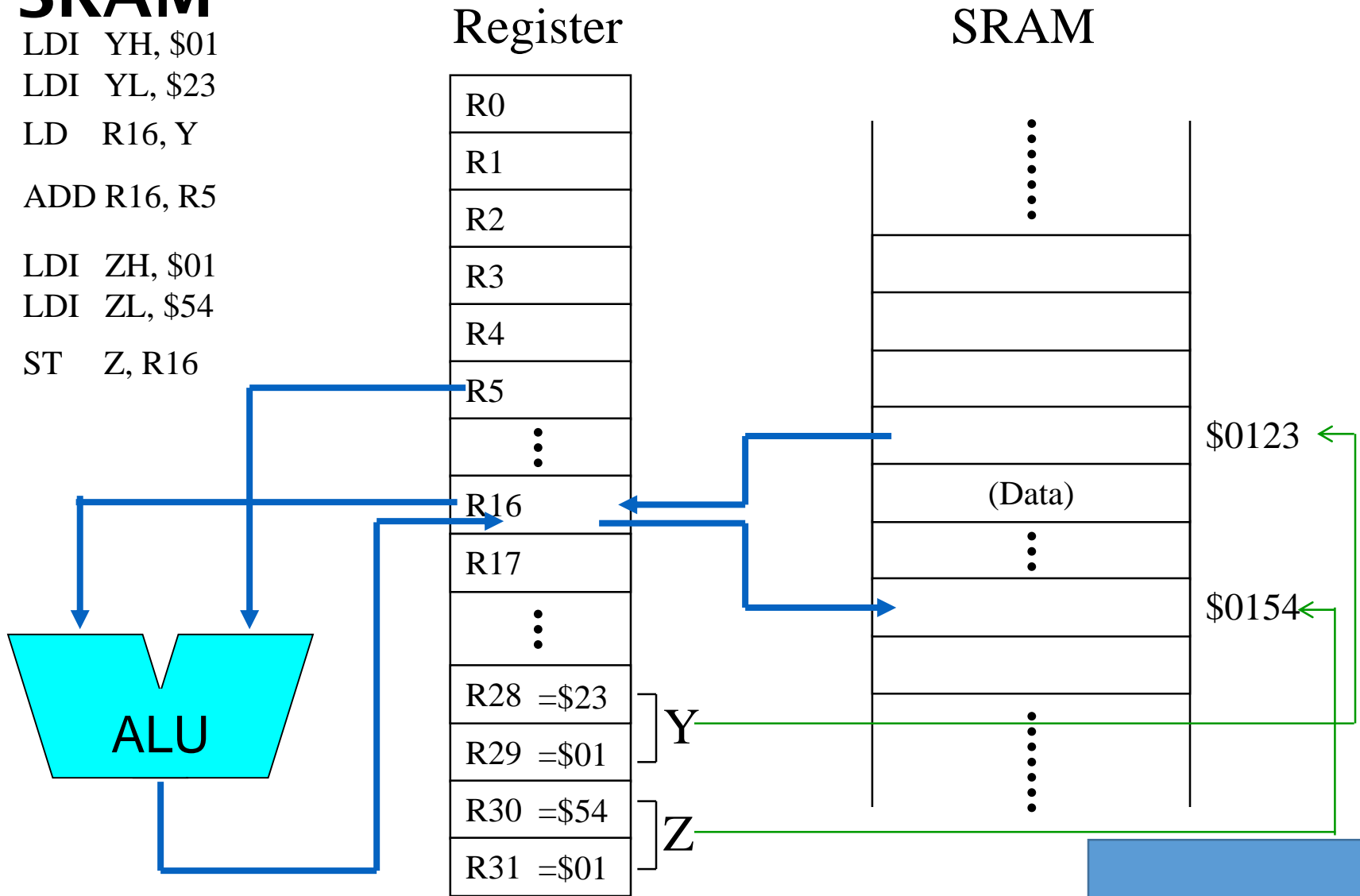
# Menggunakan Data dari SRAM

Jumlahkan isi R5 dengan data di memory address \$0123. Simpan hasilnya di memory address \$0154.



# Menggunakan Data dari SRAM

```
LDI YH, $01  
LDI YL, $23  
LD R16, Y  
  
ADD R16, R5  
  
LDI ZH, $01  
LDI ZL, $54  
ST Z, R16
```



# Contoh 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 ÝÝÝÝÝÝ
00002A FF FF FF FF FF FF ÝÝÝÝÝÝ
00002D FF FF FF FF FF FF ÝÝÝÝÝÝ
000030 FF FF FF FF FF FF ÝÝÝÝÝÝ
000033 FF FF FF FF FF FF ÝÝÝÝÝÝ
000036 FF FF FF FF FF FF ÝÝÝÝÝÝ
000039 FF FF FF FF FF FF ÝÝÝÝÝÝ
00003C FF FF FF FF FF FF ÝÝÝÝÝÝ
```



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



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

# 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
```

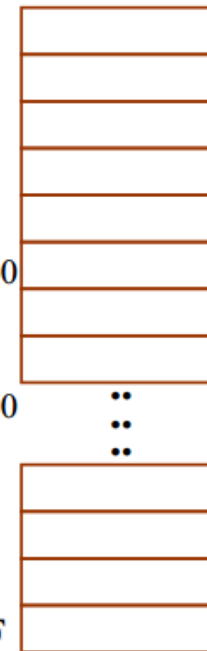
```
.def  XL  =r26  
.def  XH  =r27  
.def  YL  =r28  
.def  YH  =r29  
.def  ZL  =r30  
.def  ZH  =r31  
  
.equ  RAMEND  =$25F  
.equ  EEPROMEND  = $1FF  
.equ  FLASHEND  = $FFF
```

SRAM

BLOCK1 → 0x60

BLOCK2 → 0x80

SPH:SPL → 0x25F





# Copy Program Memory to Data Memory

```
;**** Copy 20 bytes ROM -> RAM
```

```
ldi ZH,high(F_TABLE*2)
```

```
ldi ZL,low(F_TABLE*2) ;init Z-pointer
```

```
ldi YH,high(BLOCK1)
```

```
ldi YL,low(BLOCK1) ;init Y-pointer
```

```
ldi flashsize,20
```

```
rcall flash2ram ;copy 20 bytes
```

```
ldi ZH,high(BLOCK1) ← address of next instruction
```

```
.....
```

SRAM



Push address of ldi ZH,high(BLOCK1) onto stack  
PC = address of flash2ram

# 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)**

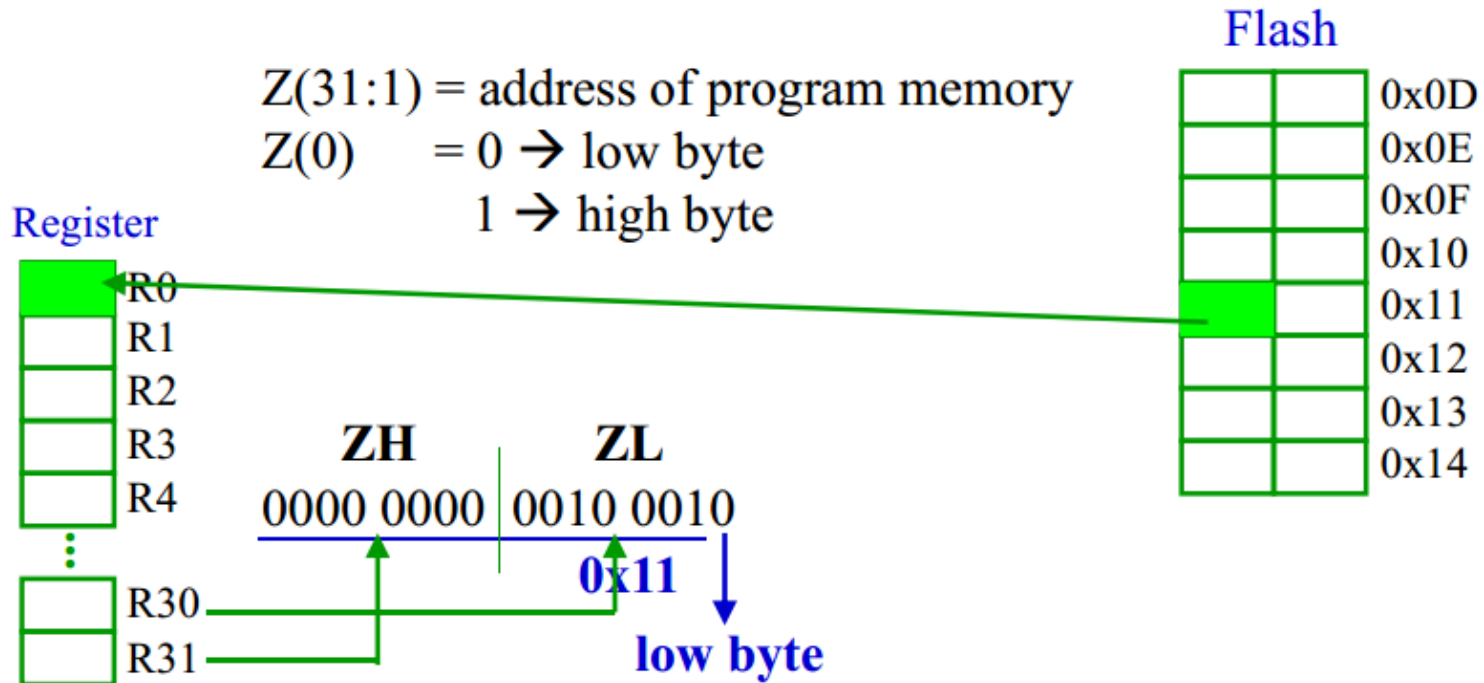
**- Copy the value pointed by TOS to PC**

**- Increment TOS**

# 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
```

# Pointer Z ke Program Memory



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