Stack & Subroutine in AVR

Tim Dosen POK

Outline

- Stack
- Stack Pointer
- Stack operations in AVR
- Subroutine

Stack

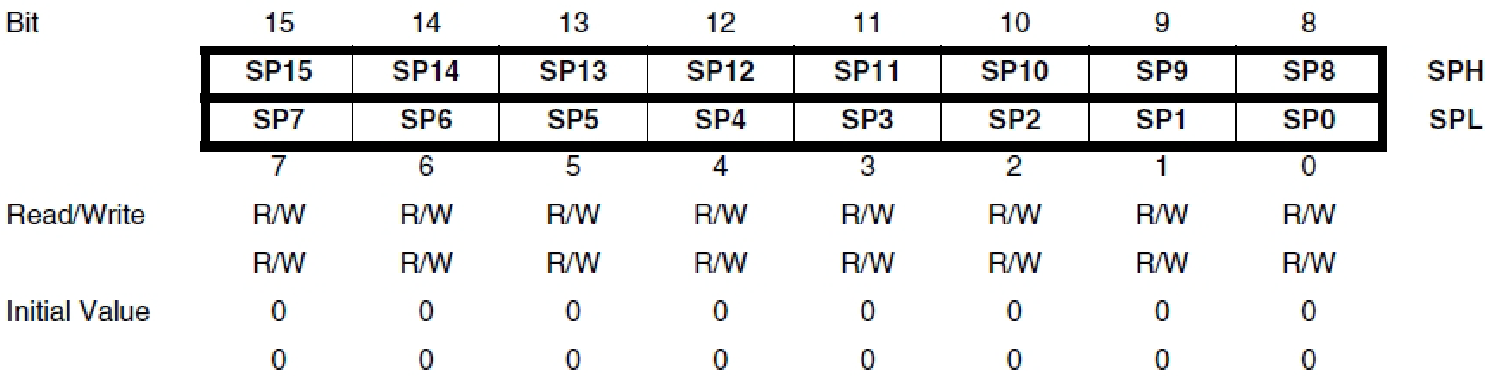
- **Stack** is mainly used for storing:
  - Temporary data
  - Local variables
  - Return Addresses after interrupts and subroutine calls
- **Stack Pointer Register** (SPH and SPL) is an I/O Register that points to the top of stack (TOS)
- **Stack space in SRAM** must be defined by program before subroutine calls are executed or interrupts are anable

Stack (cont.)

- **Last In First Out (LIFO).**
- Stack is implemented as growing from **higher memory** location to **lower memory** locations

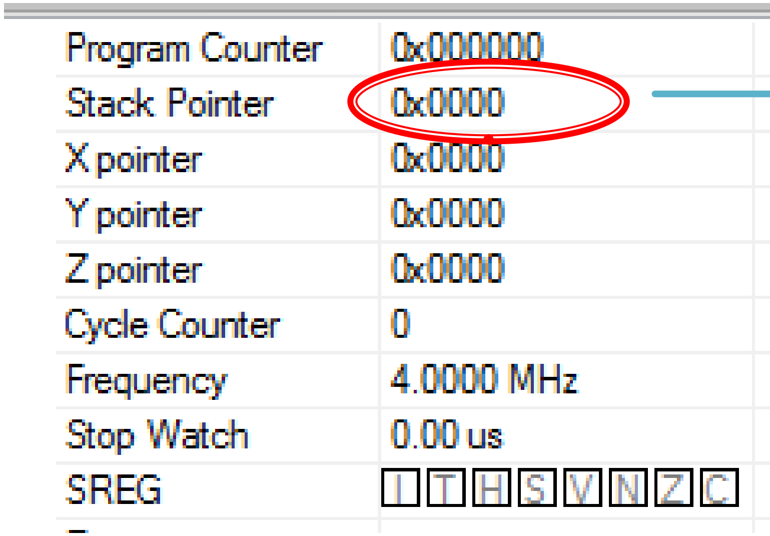
Stack Pointer

- AVR Stack Pointer is implemented as two 8-bit registers in the I/O space



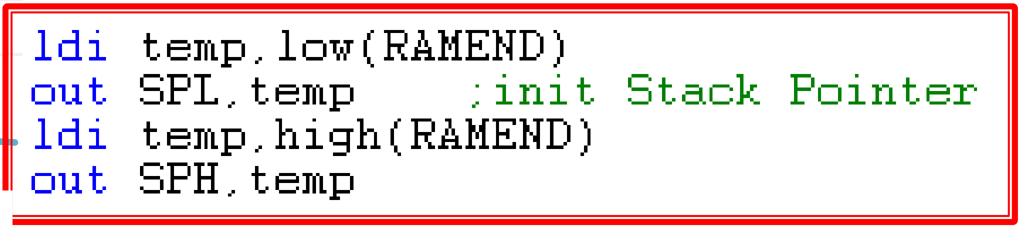
Bit	15	14	13	12	11	10	9	8	
	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Stack Pointer (cont.)



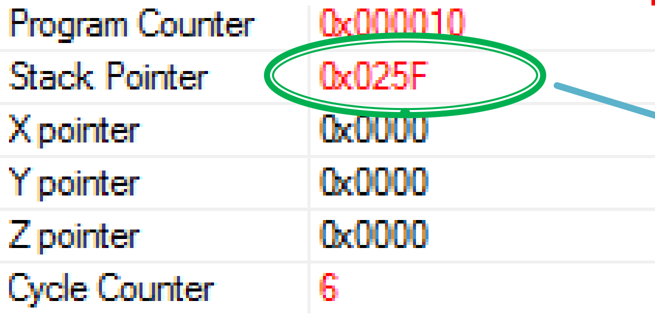
Program Counter	0x000000
Stack Pointer	0x0000
X pointer	0x0000
Y pointer	0x0000
Z pointer	0x0000
Cycle Counter	0
Frequency	4.0000 MHz
Stop Watch	0.00 us
SREG	I T H S V N Z C

Initial value = 0



```
ldi  temp,low(RAMEND)
out  SPL,temp       ;init Stack Pointer
ldi  temp,high(RAMEND)
out  SPH,temp
```

Inisialisasi nilai SPL dan SPH

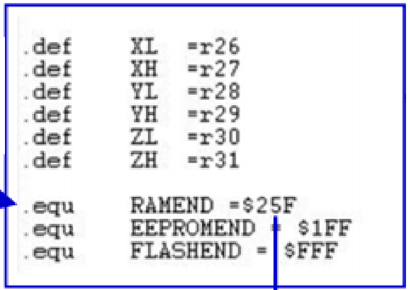


Program Counter	0x000010
Stack Pointer	0x025F
X pointer	0x0000
Y pointer	0x0000
Z pointer	0x0000
Cycle Counter	6

Points to the last byte in SRAM

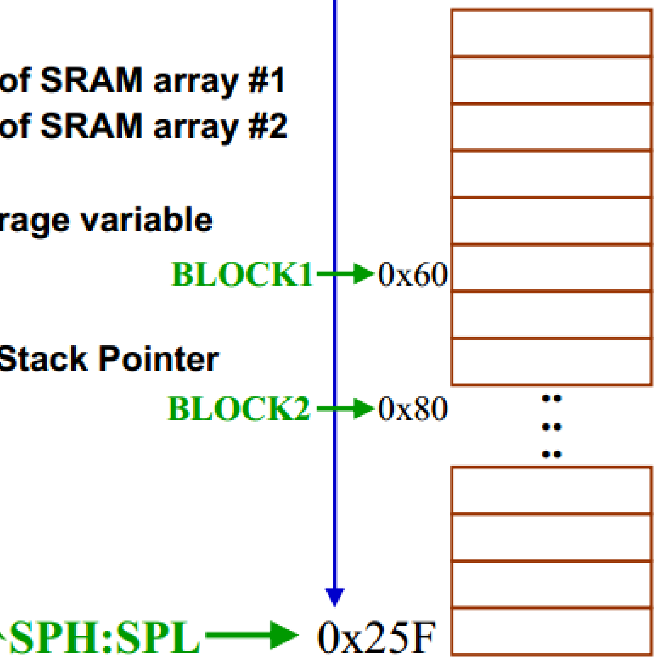
Init Stack Pointer

```
.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

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

SRAM

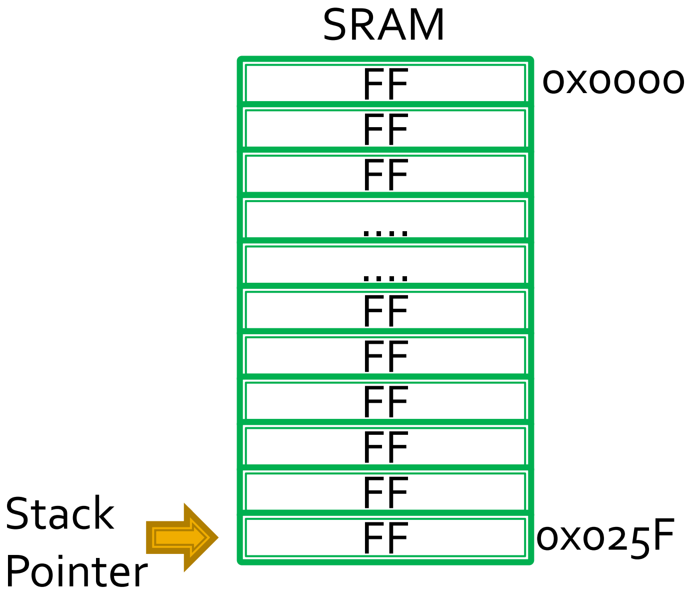


Stack operation

- PUSH [Register] instruction is used to push data onto the stack.
  - Exp: PUSH R1
  - will push the content of R1 onto the stack and then decrement SP by 1 (SP ← SP – 1)
- POP [Register] instruction is used to pop data from the stack.
  - Exp: POP R1
  - Stack Pointer is increment by 1 (SP ← SP + 1), read the content of the stack, then store it in R1.

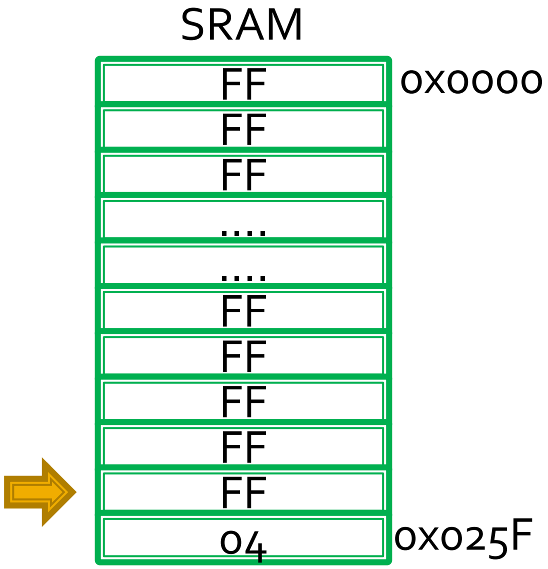
Operasi Stack (cont.)

```
  LDI   R16, 04
  MOV   R1,R16
⇨ PUSH  R1
  ....
```



Operasi Stack (cont.)

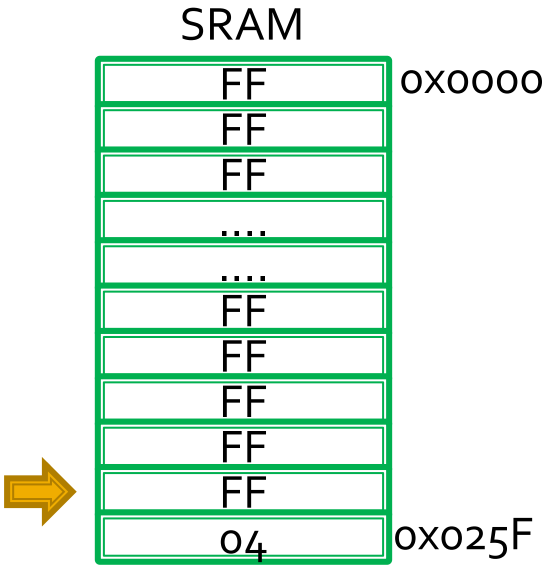
```
LDI     R16, 04
MOV     R1,R16
PUSH    R1
....
```



SRAM	
FF	0x0000
FF	
FF	
....	
....	
FF	
FF	
FF	
FF	
FF	
04	0x025F

Operasi Stack (cont.)

```
....
POP    R17
....
```

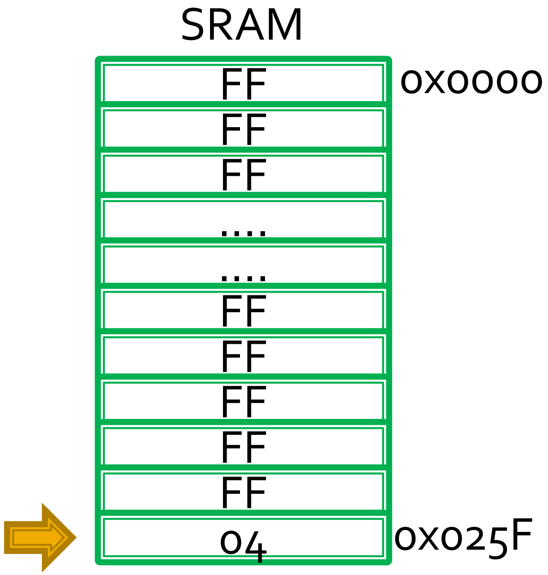


SRAM

SRAM	
FF	0x0000
FF	
FF	
....	
....	
FF	
FF	
FF	
FF	
FF	
04	0x025F

Operasi Stack (cont.)

```
....
POP    R17
....
```



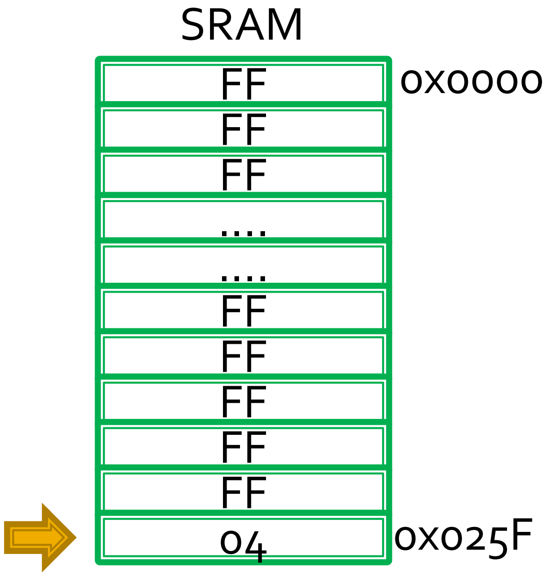
SRAM

FF	0x0000
FF	
FF	
....	
....	
FF	
FF	
FF	
FF	
FF	
04	0x025F

R17 ← 04

Operasi Stack (cont.)

```
LDI     R16, 07
PUSH    R16
....
```



SRAM

FF	0x0000
FF	
FF	
....	
....	
FF	
FF	
FF	
FF	
FF	
04	0x025F

What will happen?

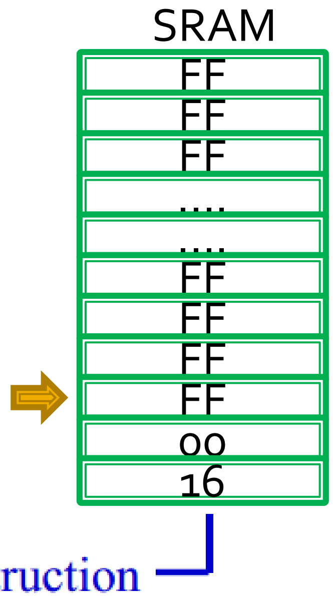
Subroutine

- Similar to the *subroutine* call in MIPS
- Use RCALL (Relative Call) instruction
  - RCALL [Label]
  - Label is the subroutine address
  - Return address will be stored in the Stack.

Subroutine call

```
;***** 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)
   .......
```

0x0016 ← ldi ZH,high(BLOCK1) ← address of next instruction



SRAM

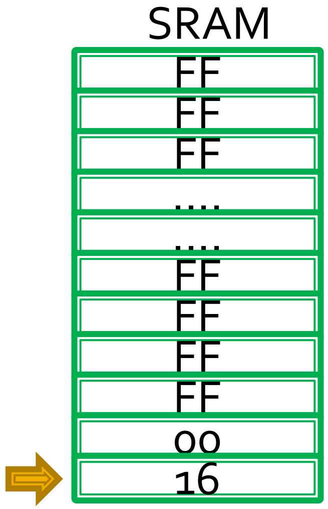
SRAM
FF
FF
FF
....
....
FF
FF
FF
FF
00
16

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

Subroutine call (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



PC ← 0x0016