XE36SKD

AVR Instruction Set

Subroutines

Miroslav Skrbek, Josef Hlaváč

Seminar 8

- Stack
- Subroutines
- Passing of parameters
- Recursion

Stack Initialization

```
.include "m169def.inc"
```

Inclusion of the ATmega169 I/O register definition file into the source code.

```
ldi     r16, 0x00
out     SPL, r16
ldi     r16, 0x04
out     SPH, r16
```

Storing the value of 400h into SP

Note: The m169def.inc definition file includes the following definitions of symbolic names for the lower and upper halves of the SP register:

```
.equ    SPL = 0x3d (low 8 bits of SP)
.equ    SPH = 0x3e (high 8 bits of SP)
```

The stack pointer should be initialized to an address that is high enough (but still in RAM) because the stack grows downwards. The maximum initial SP value for ATmega169 is 0x4FF and it is also defined (RAMEND) in the m169def.inc file.

Using the stack

- Storing return addresses in the stack when calling subroutines
- Storing the return address if an interrupt is invoked
- Passing parameters to subroutines
- Local variables of subroutines
- Local storage of registers to retain code transparency. Typical in interrupt handlers.

Subroutines

Add

8000h + 0100h + 0100h

```
ldi     r16, 0x00
ldi     r17, 0x80
ldi     r18, 0x00
ldi     r19, 0x01
call    add16


call    add16
```

Subroutine

```
; Sum of two 16-bit numbers

; Input:       R17:R16 addend 1
;              R19:R18 addend 2
;
; Output:      R17:R16 sum
;
; Uses:        R16, R17,
;              R18, R19, SREG

add16: add     r16, r18
       adc     r17, r19
       ret
```

Note: Parameters are passed in registers.
Remember to initialize the SP at the beginning of your program!

Task: Write a subroutine that determines the length of a string in data memory

```
; Subroutine strlen
; Input: Z (R31:R30) - beginning of the string
;
; Output: R0 - string length
;
; Uses: R31, R30, … list all used registers …
;
strlen:
      … write instructions here …
      ret
```

Note: The solution should include an example call to this subroutine.

Note: To test this task, use one of the previous tasks to copy a string from the program memory to the data memory.

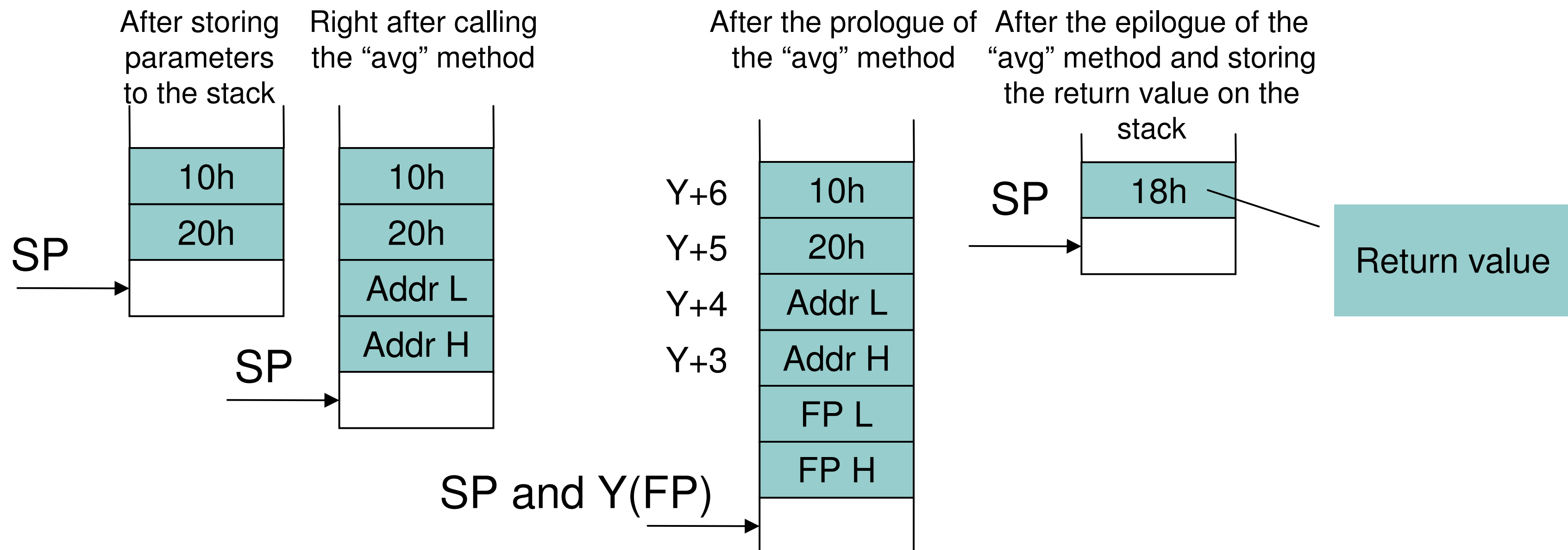
Passing parameters via stack

In Java

```
p = avg(0x10, 0x20);
```

In Java

```
int avg(byte a, byte b)
{  return (a + b) >> 1 }
```



Passing parameters to subroutines via stack

Subroutine call

```
…
ldi     r16, 0x10
push    r16
Ldi     r16, 0x20
push    r16
call    avg
pop     r16
…
```

```
avg:    ; entry code (prologue)
        push    r28
        push    r29
        in      r28, SPL
        in      r29, SPH
```

```
; body
ldd     r16, Y+6 ; parameter a
ldd     r17, Y+5 ; parameter b
add     r17, r16
lsr     r17

; exit code  (epilogue)
mov     r26, r28
mov     r27, r29
ldd     r30, Y+4
ldd     r31, Y+3
adiw    r26, 6
pop     r29
pop     r28
out     SPL, r26
out     SPH, r27
push    r17 ; push the return
               value on the stack
ijmp
;
```

Recursion

Recursive calculation of the factorial

Mathematic description

$n! = n(n-1)!$	If $n>0$
1	If $n = 0$

In Java

```
int factorial(int n) {
        if (n==0) return 1;
        else return n*factorial(n-1);
}
```

Task: Write a subroutine that calculates the factorial recursively.

- Pass parameters as well as the return values on the stack
- Reuse the “parameter passing” example (slide 9)
  - Your subroutine will have only one input parameter
  - Most changes will be to the “body”, other portions need only minor tweaks
  - Your subroutine needs to properly call itself again, if the parameter is >0
- The solution should include an example call to this subroutine
- Remember to initialize your stack