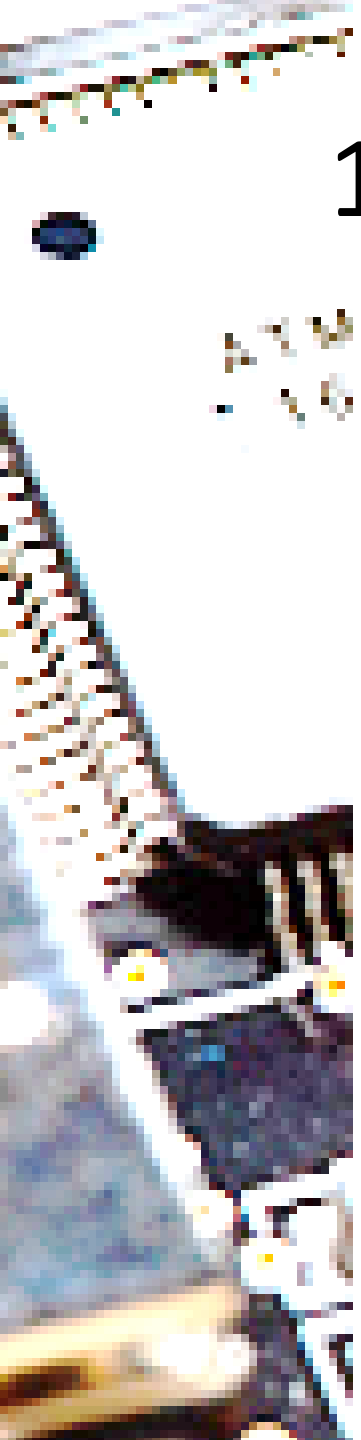
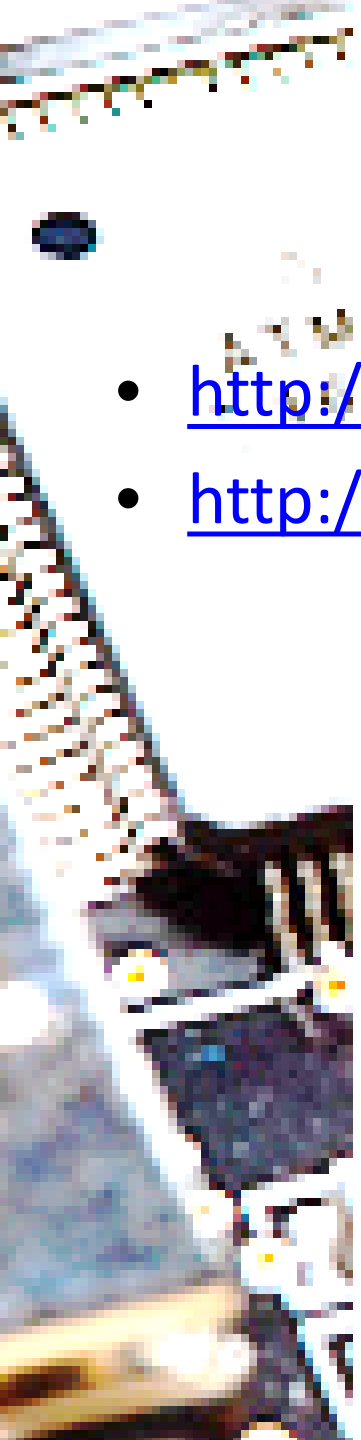


1DT301, Computer Technology I

Lecture #4,

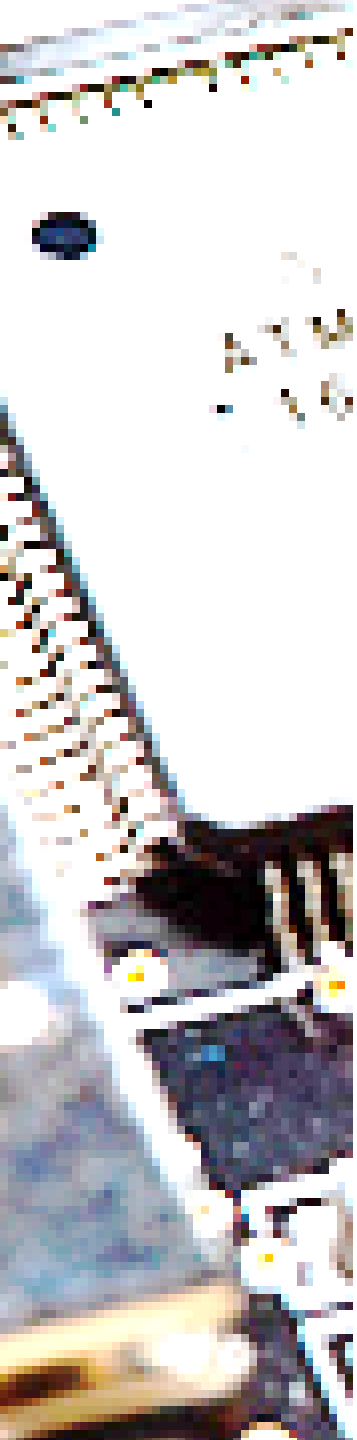
- Preview of lab 2
- Switching between programs
- Random number generator
- Calling subroutines with arguments
- Using Register Pairs
- Masking bits

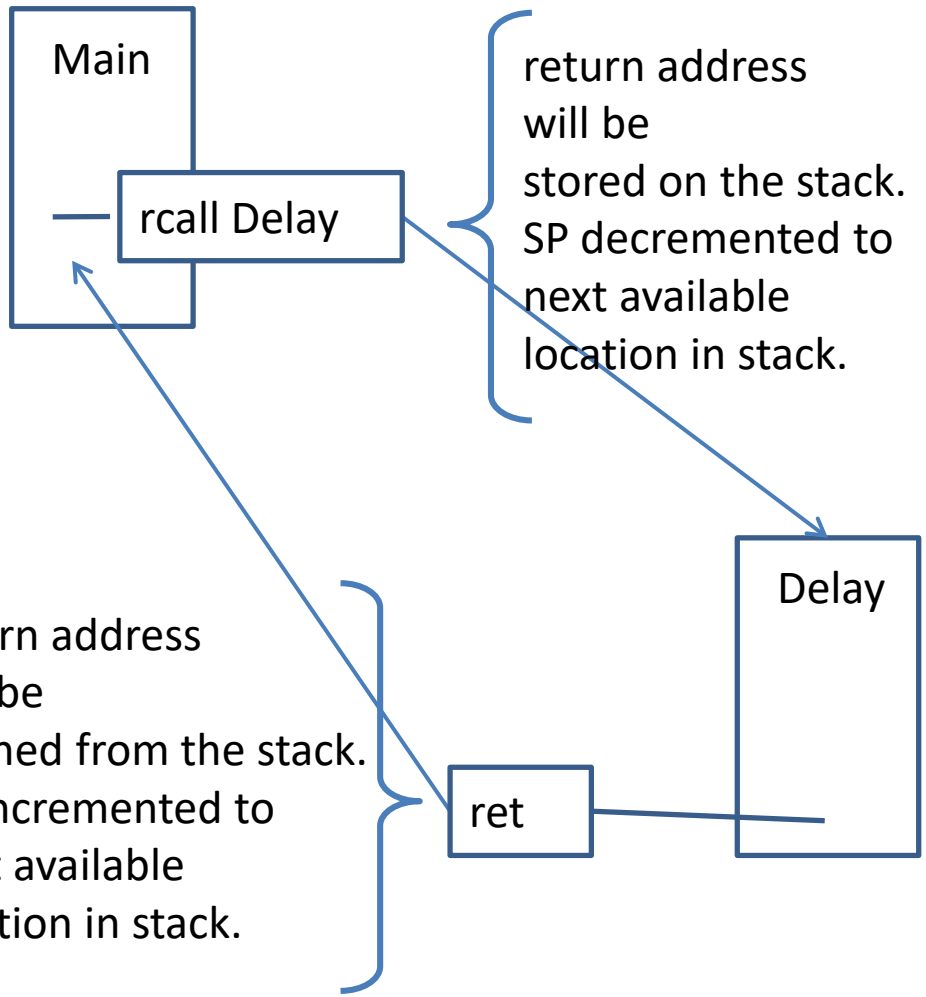


Reading:

- http://www.avr-asm-tutorial.net/avr_en/
- http://www.avrbeginners.net/

Subroutine call, example.



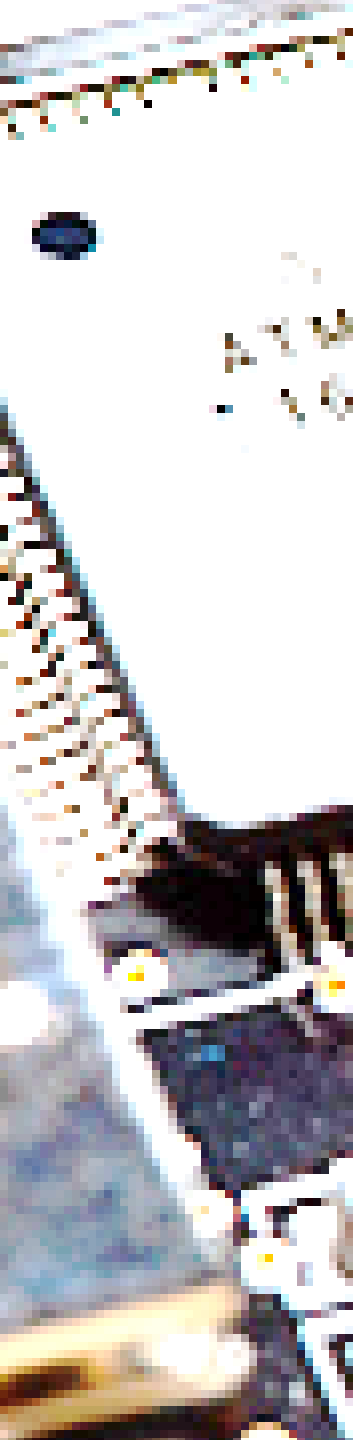


Stack

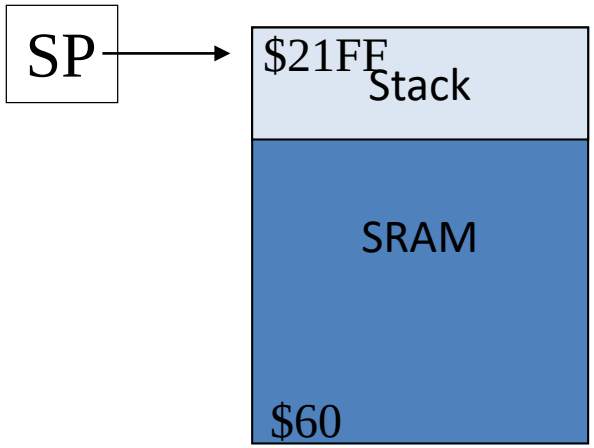
Stack

- The stack is an area of SRAM that grows downward from a fixed reference point
  - Special instructions and I/O register support stack access
    - RET, CALL, PUSH, POP, …
    - SP (Stack Pointer)
  - The stack requires initialization
    - Set SP to an appropriate address (empty stack)
  - The stack is managed by application and is used by the processor interrupt system

Initiate the Stack Pointer, SP

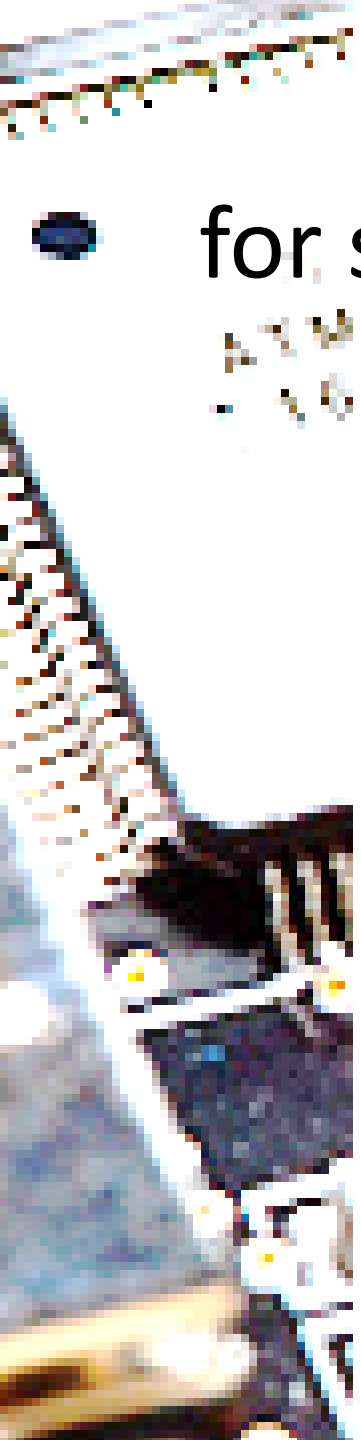


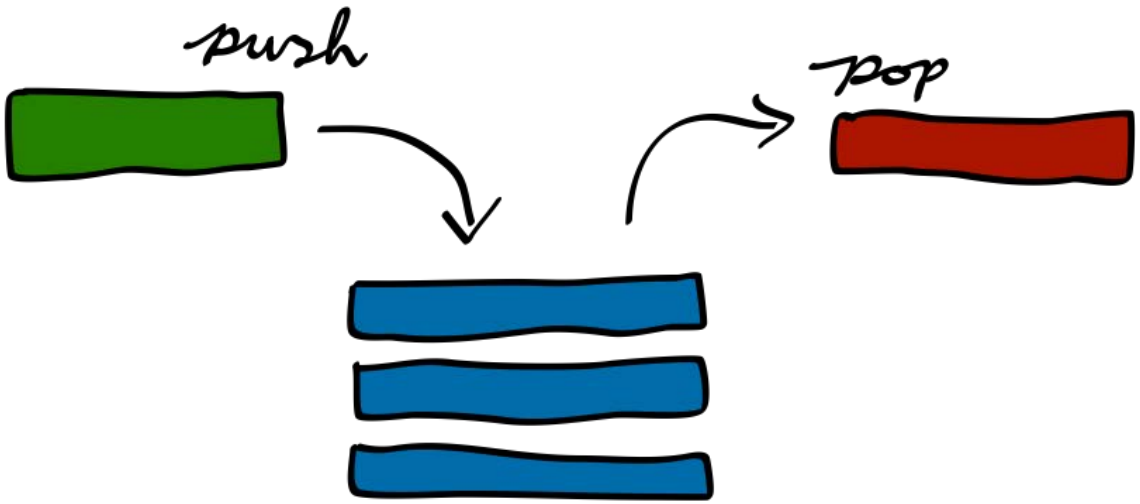
- Typically, the stack starts at the highest SRAM address and grows downward through memory
- SP points to the next available byte of stack storage (top of stack after a push)
- Initialization:
  - ldi R16, high(RAMEND)
  - out SPH, R16
  - ldi R16, low(RAMEND)
  - out SPL, R16
- RAMEND = highest address in RAM

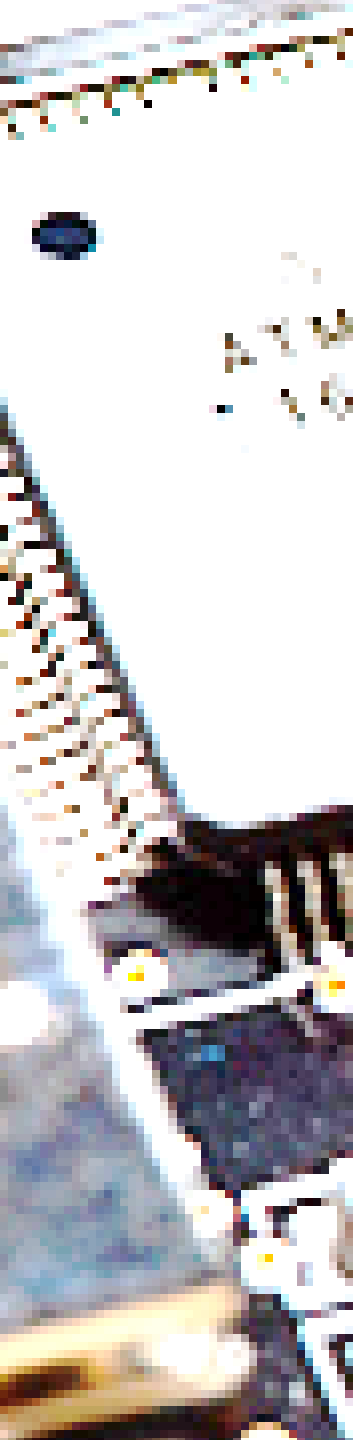


push and pop,

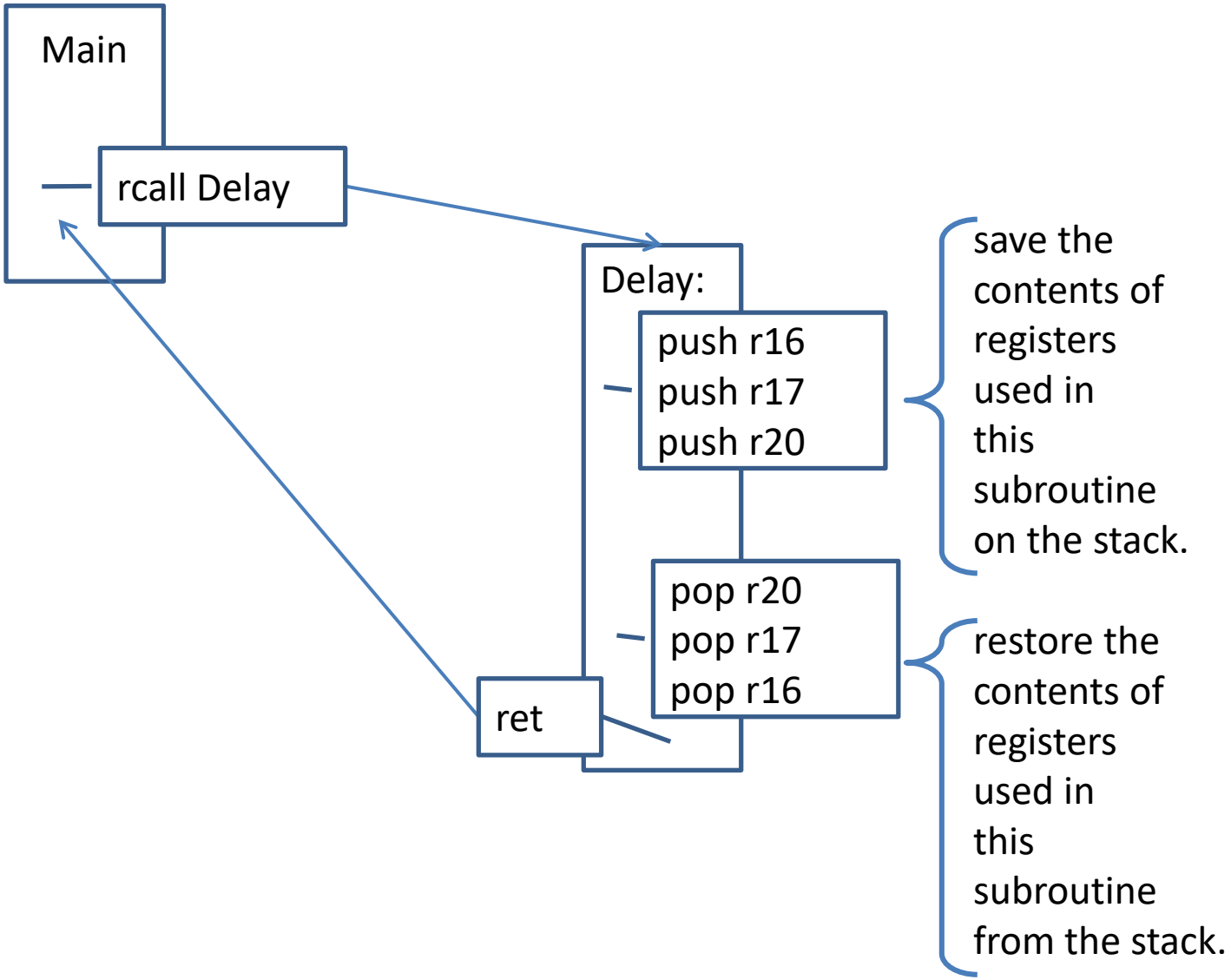
- for store and restore register contents







Subroutine call, example.



PUSH and POP

- push Rr
- Register data copied to byte at address in SP
- SP is decremented

- pop Rd
- SP is incremented
- Byte at address in SP is copied to Rd

PUSH

Addr	Content
45A	2C
45B	34
45C	36
45D	11
45E	F2
45F	5C

SP : 45D

Top of stack

push R5

Register	Content
R3	7F
R4	00
R5	A2
R6	77

POP

Addr	Content
45A	2C
45B	34
45C	36
45D	A2
45E	F2
45F	5C

SP : 45C

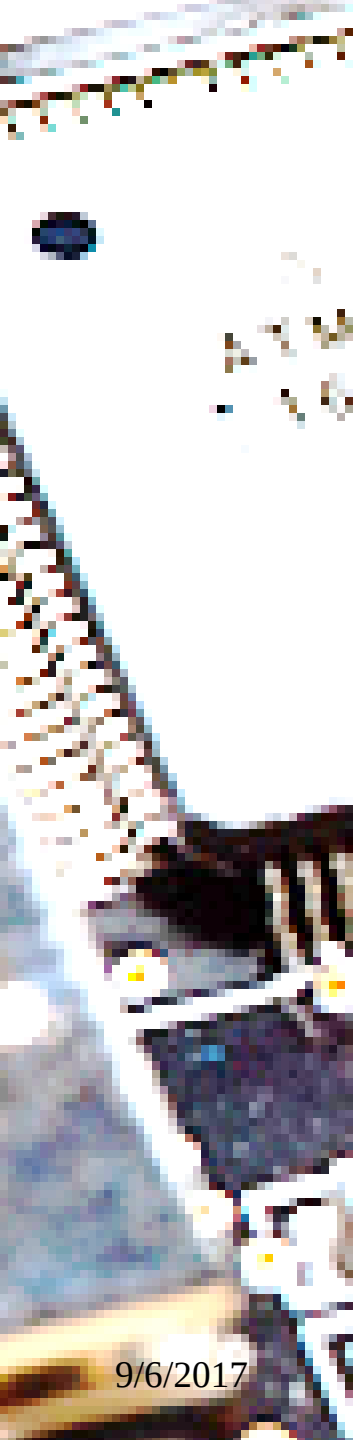
Top of stack

pop R4

Register	Content
R3	7F
R4	00
R5	A2
R6	77

Stack Usage

- Temporary storage of information
- Storage of return addresses for procedure calls and interrupts
- Storage for local variables during execution of a procedure



Temporary Storage

```
push R16                ;save register
ldi R16, $FF            ;use register
out PORTB, R16
pop R16                 ;restore register
```

RCALL – Relative Call to Subroutine

Description:

Relative call to an address within PC - 2K + 1 and PC + 2K (words). The return address (the instruction after the RCALL) is stored onto the Stack. (See also CALL). In the assembler, labels are used instead of relative operands. For AVR microcontrollers with Program memory not exceeding 4K words (8K bytes) this instruction can address the entire memory from every address location. The Stack Pointer uses a post-decrement scheme during RCALL.

Operation:

(i) PC ← PC + k + 1 Devices with 16 bits PC, 128K bytes Program memory maximum.
(ii) PC ← PC + k + 1 Devices with 22 bits PC, 8M bytes Program memory maximum.

	Syntax:	Operands:	Program Counter:	Stack:
(i)	RCALL k	-2K ≤ k < 2K	PC ← PC + k + 1	STACK ← PC + 1 SP ← SP - 2 (2 bytes, 16 bits)
(ii)	RCALL k	-2K ≤ k < 2K	PC ← PC + k + 1	STACK ← PC + 1 SP ← SP - 3 (3 bytes, 22 bits)

16-bit Opcode:

1101	kkkk	kkkk	kkkk

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
            rcall   routine     ; Call subroutine
            ...
routine:    push    r14         ; Save r14 on the Stack
            ...
```

RET – Return from Subroutine

Description:

Returns from subroutine. The return address is loaded from the STACK. The Stack Pointer uses a pre-increment schem during RET.

Operation:

(i) PC(15:0) ← STACKDevices with 16 bits PC, 128K bytes Program memory maximum.
(ii) PC(21:0) ← STACKDevices with 22 bits PC, 8M bytes Program memory maximum.

	Syntax:	Operands:	Program Counter:	Stack:
(i)	RET	None	See Operation	SP←SP + 2, (2bytes,16 bits)
(ii)	RET	None	See Operation	SP←SP + 3, (3bytes,22 bits)

16-bit Opcode:

1001	0101	0000	1000

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
                call    routine     ; Call subroutine
                ...
routine:        push    r14         ; Save r14 on the Stack
                ...
                pop     r14         ; Restore r14
                ret                 ; Return from subroutine
```

PUSH – Push Register on Stack

Description:

This instruction stores the contents of register Rr on the STACK. The Stack Pointer is post-decremented by 1 after the PUSH.

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

Operation:

(i) STACK ← Rr

	Syntax:	Operands:
(i)	PUSH Rr	0 ≤ r ≤ 31

Program Counter: PC ← PC + 1

Stack: SP ← SP - 1

16-bit Opcode:

1001	001d	dddd	1111

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
            call    routine ; Call subroutine
            ...
routine:    push    r14     ; Save r14 on the Stack
            push    r13     ; Save r13 on the Stack
            ...
            pop     r13     ; Restore r13
            pop     r14     ; Restore r14
            ret             ; Return from subroutine
```

Words: 1 (2 bytes)

POP – Pop Register from Stack

Description:

This instruction loads register Rd with a byte from the STACK. The Stack Pointer is pre-incremented by 1 before the POP.

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

Operation:

(i) Rd ← STACK

	Syntax:	Operands:	Program Counter:	Stack:
(i)	POP Rd	0 ≤ d ≤ 31	PC ← PC + 1	SP ← SP + 1

16-bit Opcode:

1001	000d	dddd	1111

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
                call    routine   ; Call subroutine
                ...
routine:        push    r14       ; Save r14 on the Stack
                push    r13       ; Save r13 on the Stack
                ...
                pop     r13       ; Restore r13
                pop     r14       ; Restore r14
                ret               ; Return from subroutine
```

Words: 1 (2 bytes)

Instruction Set Nomenclature

Status Register (SREG)

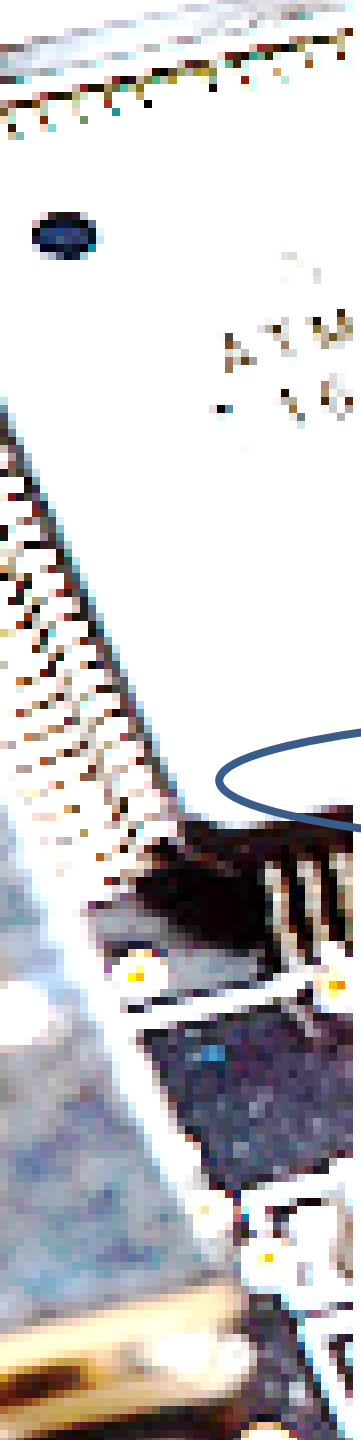
SREG:	Status Register
C:	Carry Flag
Z:	Zero Flag
N:	Negative Flag
V:	Two's complement overflow indicator
S:	N ⊕ V, For signed tests
H:	Half Carry Flag
T:	Transfer bit used by BLD and BST instructions
I:	Global Interrupt Enable/Disable Flag

Registers and Operands

Rd:	Destination (and source) register in the Register File
Rr:	Source register in the Register File
R:	Result after instruction is executed
K:	Constant data
k:	Constant address
b:	Bit in the Register File or I/O Register (3-bit)
s:	Bit in the Status Register (3-bit)
X,Y,Z:	Indirect Address Register
	(X=R27:R26, Y=R29:R28 and Z=R31:R30)
A:	I/O location address
q:	Displacement for direct addressing (6-bit)

ATMEL®

8-bit AVR® Instruction Set



LDI – Load Immediate

Description:

Loads an 8 bit constant directly to register 16 to 31.

Operation:

(i) Rd ← K

	Syntax:	Operands:	Program Counter:
(i)	LDI Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

16-bit Opcode:

1110	KKKK	dddd	KKKK

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
clr   r31       ; Clear Z high byte
ldi   r30,$F0   ; Set Z low byte to $F0
lpm             ; Load constant from Program
                ; memory pointed to by Z
```

Words: 1 (2 bytes)
Cycles: 1

ADD – Add without Carry

Description:

Adds two registers without the C Flag and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd + Rr

	Syntax:	Operands:	Program Counter:
(i)	ADD Rd,Rr	$0 \le d \le 31$, $0 \le r \le 31$	PC ← PC + 1

16-bit Opcode:

0000	11rd	dddd	rrrr

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H: $Rd3 \bullet Rr3 + Rr3 \bullet \overline{R3} + \overline{R3} \bullet Rd3$
Set if there was a carry from bit 3; cleared otherwise

S: N ⊕ V, For signed tests.

V: $Rd7 \bullet Rr7 \bullet \overline{R7} + \overline{Rd7} \bullet \overline{Rr7} \bullet R7$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: $Rd7 \bullet Rr7 + Rr7 \bullet \overline{R7} + \overline{R7} \bullet Rd7$
Set if there was carry from the MSB of the result; cleared otherwise.

R (Result) equals Rd after the operation.

ADC – Add with Carry

Description:

Adds two registers and the contents of the C Flag and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd + Rr + C

	Syntax:	Operands:	Program Counter:
(i)	ADC Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0001	11rd	dddd	rrrr

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H: $Rd3 \bullet Rr3 + Rr3 \bullet \overline{R3} + \overline{R3} \bullet Rd3$
Set if there was a carry from bit 3; cleared otherwise

S: $N \oplus V$, For signed tests.

V: $Rd7 \bullet Rr7 \bullet \overline{R7} + \overline{Rd7} \bullet \overline{Rr7} \bullet R7$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: $Rd7 \bullet Rr7 + Rr7 \bullet \overline{R7} + \overline{R7} \bullet Rd7$
Set if there was carry from the MSB of the result; cleared otherwise.

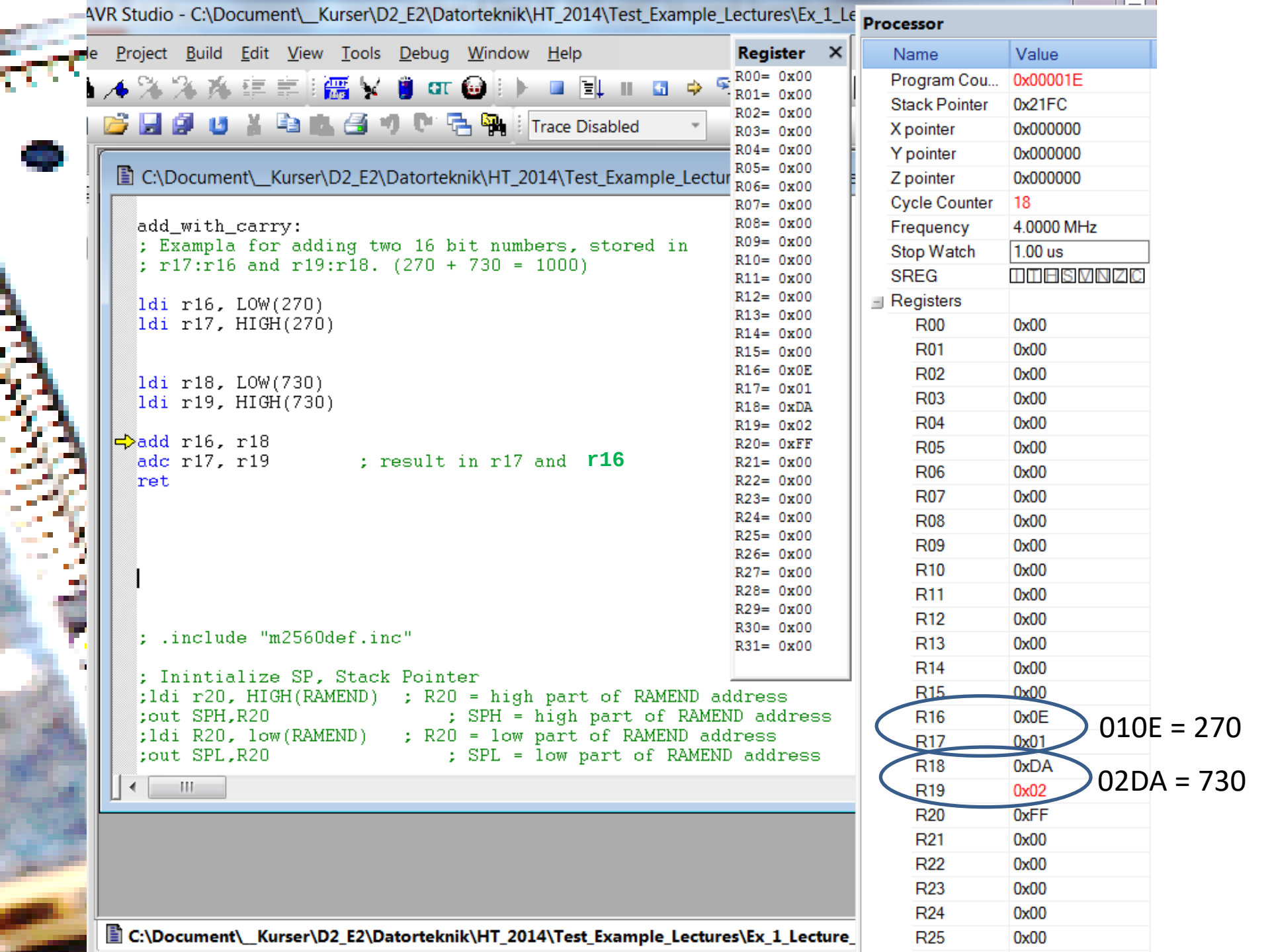
Addition, register pair

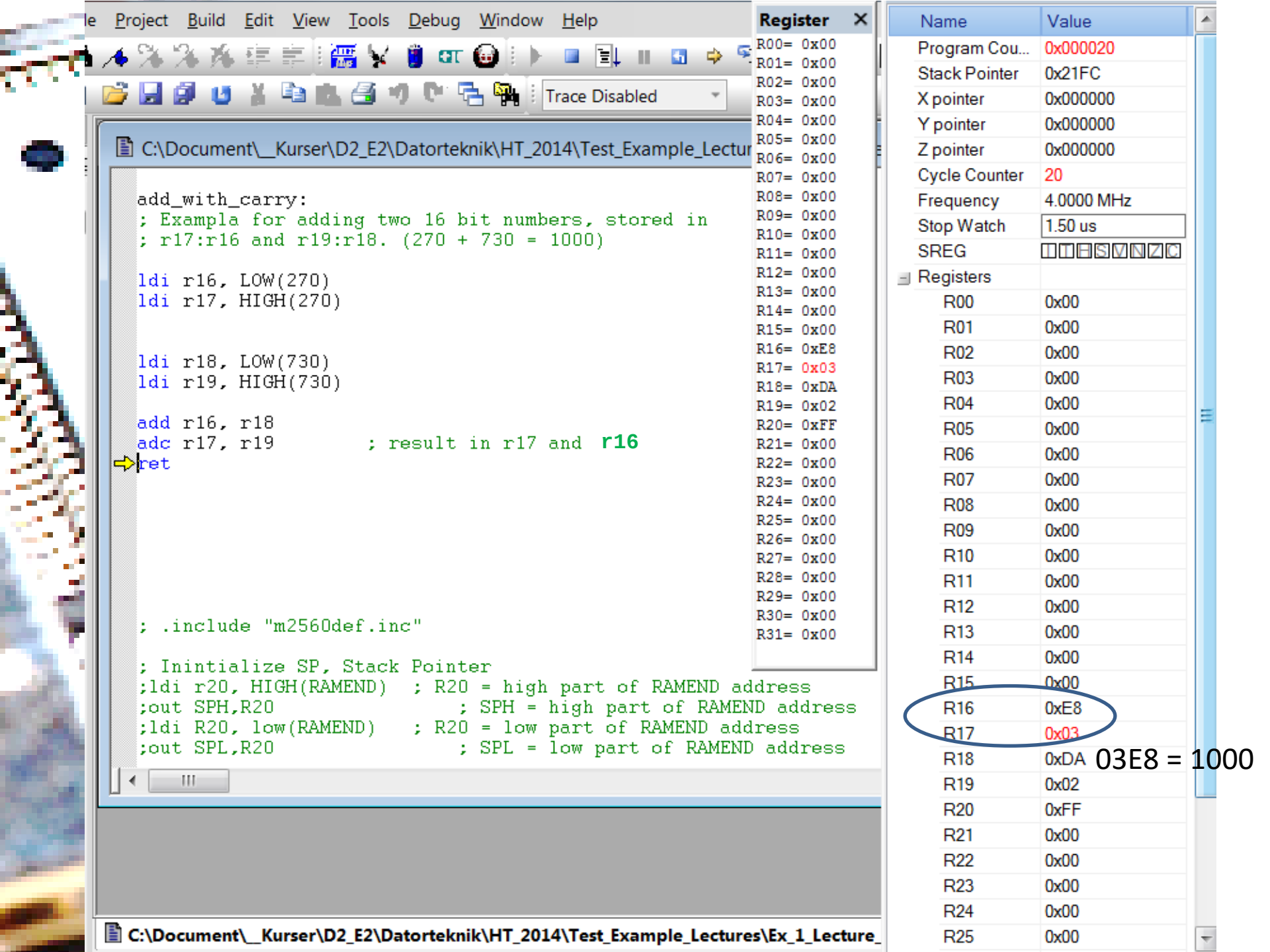
Example:

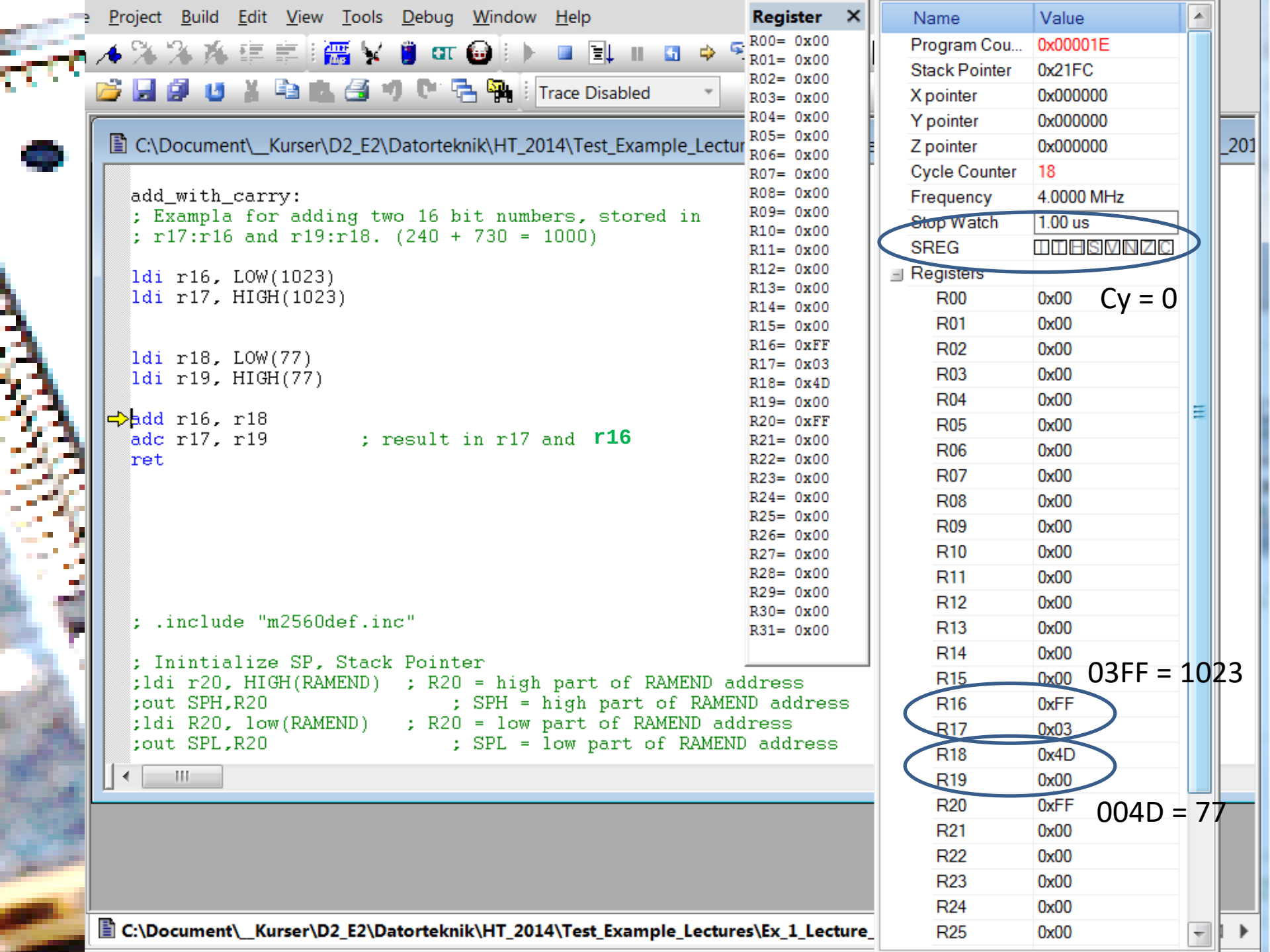
```
                        ; Add R1:R0 to R3:R2
    add   r2,r0         ; Add low byte
    adc   r3,r1         ; Add with carry high byte
```

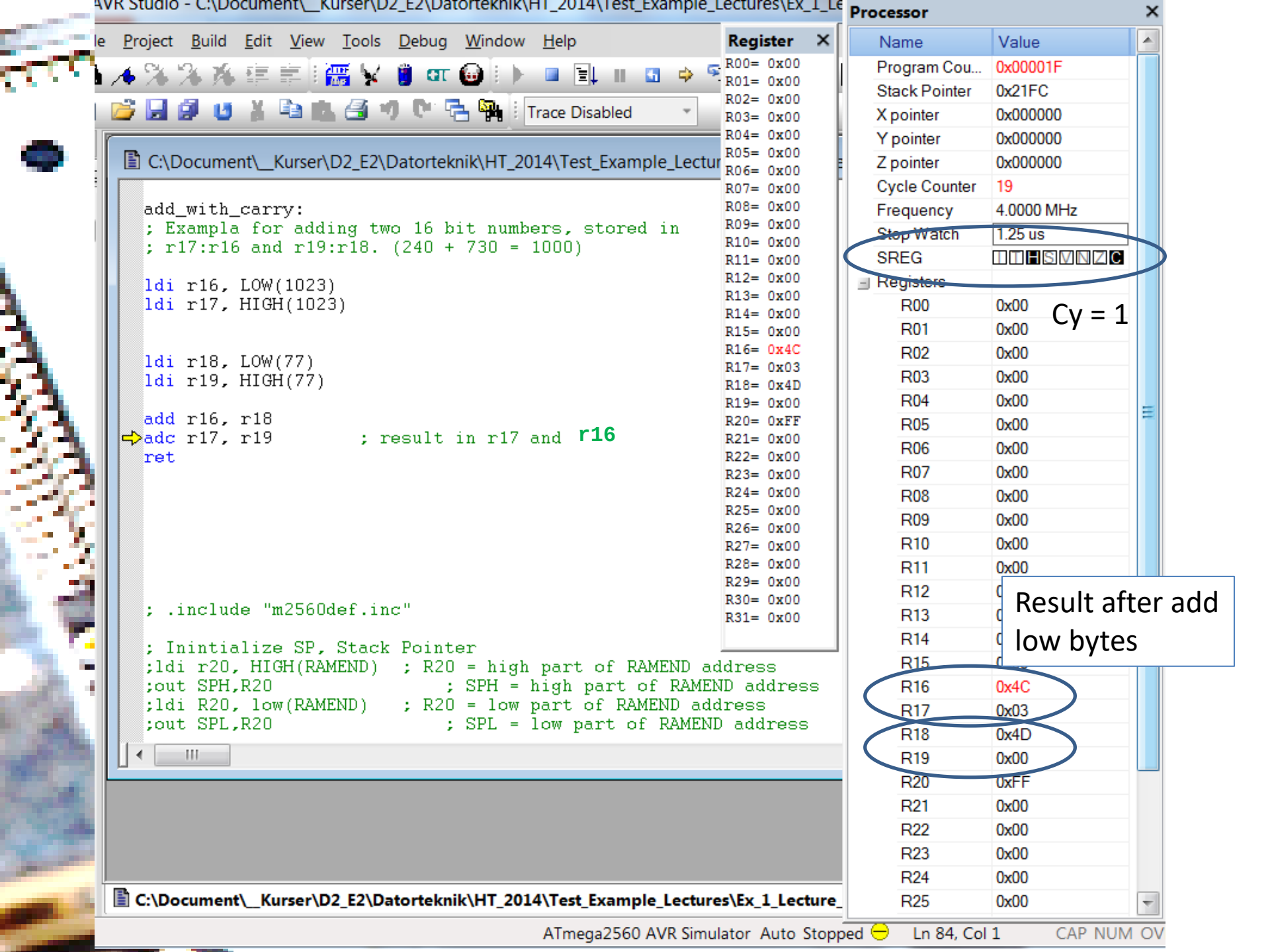
Words: 1 (2 bytes)

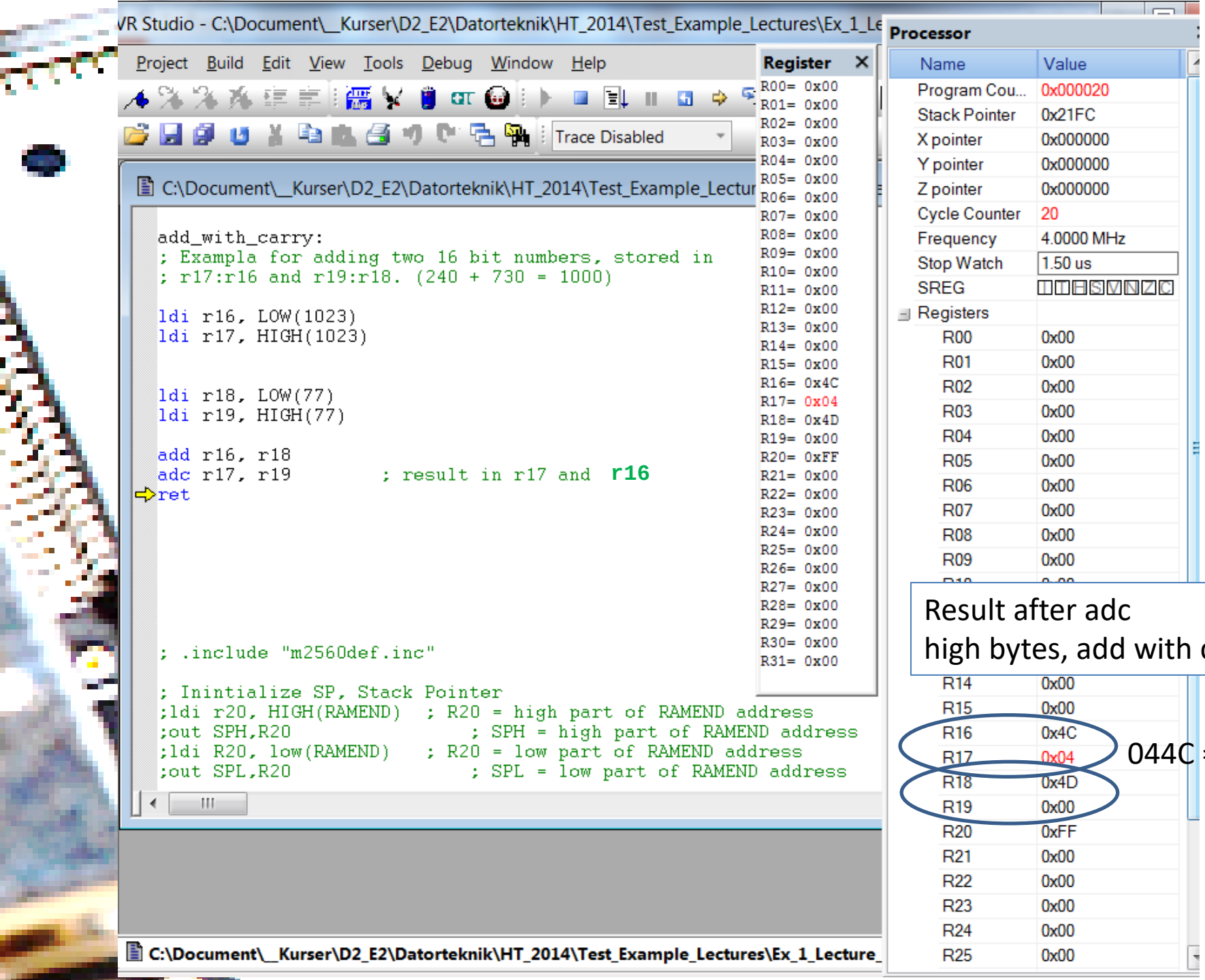
Cycles: 1


AVR Studio - C:\Document__Kurser\D2_E2\Datorteknik\HT_2014\Test_Example_Lectures\Ex_1_Le
Project Build Edit View Tools Debug Window Help
Trace Disabled
C:\Document__Kurser\D2_E2\Datorteknik\HT_2014\Test_Example_Lectur
add_with_carry:
; Exampla for adding two 16 bit numbers, stored in
; r17:r16 and r19:r18. (270 + 730 = 1000)
ldi r16, LOW(270)
ldi r17, HIGH(270)
ldi r18, LOW(730)
ldi r19, HIGH(730)
add r16, r18
adc r17, r19 ; result in r17 and r16
ret
; .include "m2560def.inc"
; Inintialize SP, Stack Pointer
;ldi r20, HIGH(RAMEND) ; R20 = high part of RAMEND address
;out SPH,R20 ; SPH = high part of RAMEND address
;ldi R20, low(RAMEND) ; R20 = low part of RAMEND address
;out SPL,R20 ; SPL = low part of RAMEND address
C:\Document__Kurser\D2_E2\Datorteknik\HT_2014\Test_Example_Lectures\Ex_1_Lecture_
Register
R00= 0x00
R01= 0x00
R02= 0x00
R03= 0x00
R04= 0x00
R05= 0x00
R06= 0x00
R07= 0x00
R08= 0x00
R09= 0x00
R10= 0x00
R11= 0x00
R12= 0x00
R13= 0x00
R14= 0x00
R15= 0x00
R16= 0x0E
R17= 0x01
R18= 0xDA
R19= 0x02
R20= 0xFF
R21= 0x00
R22= 0x00
R23= 0x00
R24= 0x00
R25= 0x00
R26= 0x00
R27= 0x00
R28= 0x00
R29= 0x00
R30= 0x00
R31= 0x00
Processor
Name Value
Program Cou... 0x00001E
Stack Pointer 0x21FC
X pointer 0x000000
Y pointer 0x000000
Z pointer 0x000000
Cycle Counter 18
Frequency 4.0000 MHz
Stop Watch 1.00 us
SREG
Registers
R00 0x00
R01 0x00
R02 0x00
R03 0x00
R04 0x00
R05 0x00
R06 0x00
R07 0x00
R08 0x00
R09 0x00
R10 0x00
R11 0x00
R12 0x00
R13 0x00
R14 0x00
R15 0x00
R16 0x0E
R17 0x01
R18 0xDA
R19 0x02
R20 0xFF
R21 0x00
R22 0x00
R23 0x00
R24 0x00
R25 0x00
010E = 270
02DA = 730


Project Build Edit View Tools Debug Window Help
Trace Disabled
C:\Document__Kurser\D2_E2\Datorteknik\HT_2014\Test_Example_Lectur
add_with_carry:
; Exampla for adding two 16 bit numbers, stored in
; r17:r16 and r19:r18. (270 + 730 = 1000)
ldi r16, LOW(270)
ldi r17, HIGH(270)
ldi r18, LOW(730)
ldi r19, HIGH(730)
add r16, r18
adc r17, r19 ; result in r17 and r16
ret
; .include "m2560def.inc"
; Inintialize SP, Stack Pointer
;ldi r20, HIGH(RAMEND) ; R20 = high part of RAMEND address
;out SPH,R20 ; SPH = high part of RAMEND address
;ldi R20, low(RAMEND) ; R20 = low part of RAMEND address
;out SPL,R20 ; SPL = low part of RAMEND address
Register
R00= 0x00
R01= 0x00
R02= 0x00
R03= 0x00
R04= 0x00
R05= 0x00
R06= 0x00
R07= 0x00
R08= 0x00
R09= 0x00
R10= 0x00
R11= 0x00
R12= 0x00
R13= 0x00
R14= 0x00
R15= 0x00
R16= 0xE8
R17= 0x03
R18= 0xDA
R19= 0x02
R20= 0xFF
R21= 0x00
R22= 0x00
R23= 0x00
R24= 0x00
R25= 0x00
R26= 0x00
R27= 0x00
R28= 0x00
R29= 0x00
R30= 0x00
R31= 0x00
Name Value
Program Cou... 0x000020
Stack Pointer 0x21FC
X pointer 0x000000
Y pointer 0x000000
Z pointer 0x000000
Cycle Counter 20
Frequency 4.0000 MHz
Stop Watch 1.50 us
SREG
Registers
R00 0x00
R01 0x00
R02 0x00
R03 0x00
R04 0x00
R05 0x00
R06 0x00
R07 0x00
R08 0x00
R09 0x00
R10 0x00
R11 0x00
R12 0x00
R13 0x00
R14 0x00
R15 0x00
R16 0xE8
R17 0x03
R18 0xDA
R19 0x02
R20 0xFF
R21 0x00
R22 0x00
R23 0x00
R24 0x00
R25 0x00
03E8 = 1000
C:\Document__Kurser\D2_E2\Datorteknik\HT_2014\Test_Example_Lectures\Ex_1_Lecture_


Project Build Edit View Tools Debug Window Help
Trace Disabled
C:\Document__Kurser\D2_E2\Datorteknik\HT_2014\Test_Example_Lectur
add_with_carry:
; Exampla for adding two 16 bit numbers, stored in
; r17:r16 and r19:r18. (240 + 730 = 1000)
ldi r16, LOW(1023)
ldi r17, HIGH(1023)
ldi r18, LOW(77)
ldi r19, HIGH(77)
add r16, r18
adc r17, r19 ; result in r17 and r16
ret
; .include "m2560def.inc"
; Inintialize SP, Stack Pointer
;ldi r20, HIGH(RAMEND) ; R20 = high part of RAMEND address
;out SPH,R20 ; SPH = high part of RAMEND address
;ldi R20, low(RAMEND) ; R20 = low part of RAMEND address
;out SPL,R20 ; SPL = low part of RAMEND address
Register
R00= 0x00
R01= 0x00
R02= 0x00
R03= 0x00
R04= 0x00
R05= 0x00
R06= 0x00
R07= 0x00
R08= 0x00
R09= 0x00
R10= 0x00
R11= 0x00
R12= 0x00
R13= 0x00
R14= 0x00
R15= 0x00
R16= 0xFF
R17= 0x03
R18= 0x4D
R19= 0x00
R20= 0xFF
R21= 0x00
R22= 0x00
R23= 0x00
R24= 0x00
R25= 0x00
R26= 0x00
R27= 0x00
R28= 0x00
R29= 0x00
R30= 0x00
R31= 0x00
Name Value
Program Cou... 0x00001E
Stack Pointer 0x21FC
X pointer 0x000000
Y pointer 0x000000
Z pointer 0x000000
Cycle Counter 18
Frequency 4.0000 MHz
Stop Watch 1.00 us
SREG
Registers
R00 0x00
R01 0x00
R02 0x00
R03 0x00
R04 0x00
R05 0x00
R06 0x00
R07 0x00
R08 0x00
R09 0x00
R10 0x00
R11 0x00
R12 0x00
R13 0x00
R14 0x00
R15 0x00
R16 0xFF
R17 0x03
R18 0x4D
R19 0x00
R20 0xFF
R21 0x00
R22 0x00
R23 0x00
R24 0x00
R25 0x00
Cy = 0
03FF = 1023
004D = 77
C:\Document__Kurser\D2_E2\Datorteknik\HT_2014\Test_Example_Lectures\Ex_1_Lecture_


AVR Studio - C:\Document__Kurser\D2_E2\Datorteknik\HT_2014\Test_Example_Lectures\Ex_1_Le
Project Build Edit View Tools Debug Window Help
Trace Disabled
C:\Document__Kurser\D2_E2\Datorteknik\HT_2014\Test_Example_Lectur
add_with_carry:
; Exampla for adding two 16 bit numbers, stored in
; r17:r16 and r19:r18. (240 + 730 = 1000)
ldi r16, LOW(1023)
ldi r17, HIGH(1023)
ldi r18, LOW(77)
ldi r19, HIGH(77)
add r16, r18
adc r17, r19 ; result in r17 and r16
ret
; .include "m2560def.inc"
; Inintialize SP, Stack Pointer
;ldi r20, HIGH(RAMEND) ; R20 = high part of RAMEND address
;out SPH,R20 ; SPH = high part of RAMEND address
;ldi R20, low(RAMEND) ; R20 = low part of RAMEND address
;out SPL,R20 ; SPL = low part of RAMEND address
Register
R00= 0x00
R01= 0x00
R02= 0x00
R03= 0x00
R04= 0x00
R05= 0x00
R06= 0x00
R07= 0x00
R08= 0x00
R09= 0x00
R10= 0x00
R11= 0x00
R12= 0x00
R13= 0x00
R14= 0x00
R15= 0x00
R16= 0x4C
R17= 0x03
R18= 0x4D
R19= 0x00
R20= 0xFF
R21= 0x00
R22= 0x00
R23= 0x00
R24= 0x00
R25= 0x00
R26= 0x00
R27= 0x00
R28= 0x00
R29= 0x00
R30= 0x00
R31= 0x00
Processor
Name Value
Program Cou... 0x00001F
Stack Pointer 0x21FC
X pointer 0x000000
Y pointer 0x000000
Z pointer 0x000000
Cycle Counter 19
Frequency 4.0000 MHz
Stop Watch 1.25 us
SREG
Registers
R00 0x00
R01 0x00
R02 0x00
R03 0x00
R04 0x00
R05 0x00
R06 0x00
R07 0x00
R08 0x00
R09 0x00
R10 0x00
R11 0x00
R12
R13
R14
R15
R16 0x4C
R17 0x03
R18 0x4D
R19 0x00
R20 0xFF
R21 0x00
R22 0x00
R23 0x00
R24 0x00
R25 0x00
Cy = 1
Result after add low bytes
C:\Document__Kurser\D2_E2\Datorteknik\HT_2014\Test_Example_Lectures\Ex_1_Lecture_
ATmega2560 AVR Simulator Auto Stopped
Ln 84, Col 1
CAP NUM OV


VR Studio - C:\Document__Kurser\D2_E2\Datorteknik\HT_2014\Test_Example_Lectures\Ex_1_Le
Project Build Edit View Tools Debug Window Help
Trace Disabled
C:\Document__Kurser\D2_E2\Datorteknik\HT_2014\Test_Example_Lectur
add_with_carry:
; Exampla for adding two 16 bit numbers, stored in
; r17:r16 and r19:r18. (240 + 730 = 1000)
ldi r16, LOW(1023)
ldi r17, HIGH(1023)
ldi r18, LOW(77)
ldi r19, HIGH(77)
add r16, r18
adc r17, r19 ; result in r17 and r16
ret
; .include "m2560def.inc"
; Inintialize SP, Stack Pointer
;ldi r20, HIGH(RAMEND) ; R20 = high part of RAMEND address
;out SPH,R20 ; SPH = high part of RAMEND address
;ldi R20, low(RAMEND) ; R20 = low part of RAMEND address
;out SPL,R20 ; SPL = low part of RAMEND address
Register
R00= 0x00
R01= 0x00
R02= 0x00
R03= 0x00
R04= 0x00
R05= 0x00
R06= 0x00
R07= 0x00
R08= 0x00
R09= 0x00
R10= 0x00
R11= 0x00
R12= 0x00
R13= 0x00
R14= 0x00
R15= 0x00
R16= 0x4C
R17= 0x04
R18= 0x4D
R19= 0x00
R20= 0xFF
R21= 0x00
R22= 0x00
R23= 0x00
R24= 0x00
R25= 0x00
R26= 0x00
R27= 0x00
R28= 0x00
R29= 0x00
R30= 0x00
R31= 0x00
Processor
Name Value
Program Cou... 0x000020
Stack Pointer 0x21FC
X pointer 0x000000
Y pointer 0x000000
Z pointer 0x000000
Cycle Counter 20
Frequency 4.0000 MHz
Stop Watch 1.50 us
SREG
Registers
R00 0x00
R01 0x00
R02 0x00
R03 0x00
R04 0x00
R05 0x00
R06 0x00
R07 0x00
R08 0x00
R09 0x00
R14 0x00
R15 0x00
R16 0x4C
R17 0x04
R18 0x4D
R19 0x00
R20 0xFF
R21 0x00
R22 0x00
R23 0x00
R24 0x00
R25 0x00
Result after adc
high bytes, add with carry
044C = 1100
C:\Document__Kurser\D2_E2\Datorteknik\HT_2014\Test_Example_Lectures\Ex_1_Lecture_

AND – Logical AND

Description:

Performs the logical AND between the contents of register Rd and register Rr and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd • Rr

	Syntax:	Operands:	Program Counter:
(i)	AND Rd,Rr	$0 \le d \le 31$, $0 \le r \le 31$	PC ← PC + 1

16-bit Opcode:

0010	00rd	dddd	rrrr

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	–

S: N ⊕ V, For signed tests.

V: 0
Cleared

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

ANDI – Logical AND with Immediate

Description:

Performs the logical AND between the contents of register Rd and a constant and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd • K

	Syntax:	Operands:	Program Counter:
(i)	ANDI Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

16-bit Opcode:

0111	KKKK	dddd	KKKK

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	–

S: N ⊕ V, For signed tests.

V: 0
Cleared

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7}$ •$\overline{R6}$• $\overline{R5}$•$\overline{R4}$ •$\overline{R3}$• $\overline{R2}$• $\overline{R1}$• $\overline{R0}$
Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

S: $N \oplus V$, For signed tests.

V: 0
Cleared

N: R7
Set if MSB of the result is set; cleared otherwise.

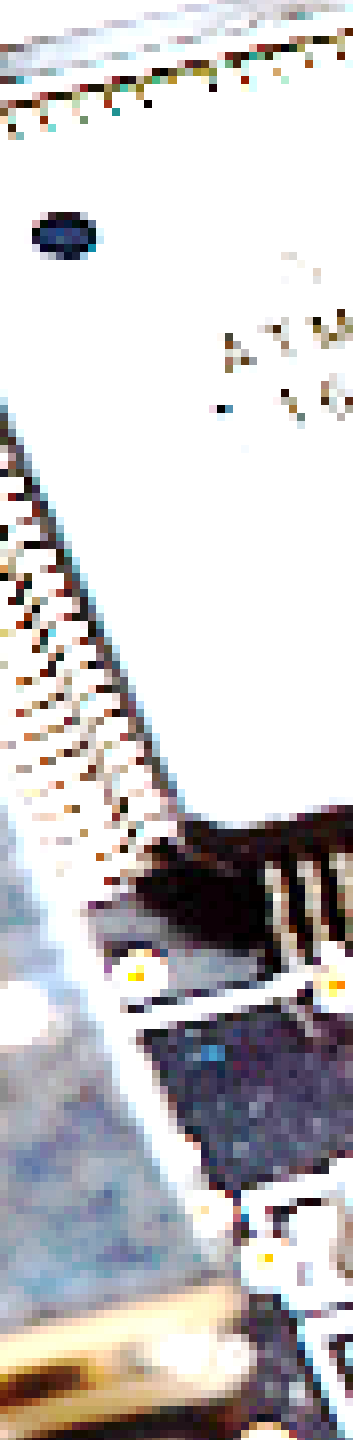
Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
andi  r17,$0F   ; Clear upper nibble of r17
andi  r18,$10   ; Isolate bit 4 in r18
andi  r19,$AA   ; Clear odd bits of r19
```

Words: 1 (2 bytes)
Cycles: 1



BLD – Bit Load from the T Flag in SREG to a Bit in Register

Description:

Copies the T Flag in the SREG (Status Register) to bit b in register Rd.

Operation:

(i) Rd(b) ← T

	Syntax:	Operands:	Program Counter:
(i)	BLD Rd,b	0 ≤ d ≤ 31, 0 ≤ b ≤ 7	PC ← PC + 1

16 bit Opcode:

1111	100d	dddd	0bbb

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
                ; Copy bit
    bst   r1,2  ; Store bit 2 of r1 in T Flag
    bld   r0,4  ; Load T Flag into bit 4 of r0
```

Words: 1 (2 bytes)
Cycles: 1

BRCC – Branch if Carry Cleared

Description:

Conditional relative branch. Tests the Carry Flag (C) and branches relatively to PC if C is cleared. 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 0,k).

Operation:

(i) If C = 0 then PC ← PC + k + 1, else PC ← PC + 1

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

16-bit Opcode:

1111	01kk	kkkk	k000

Status Register (SREG) and Boolean Formula:

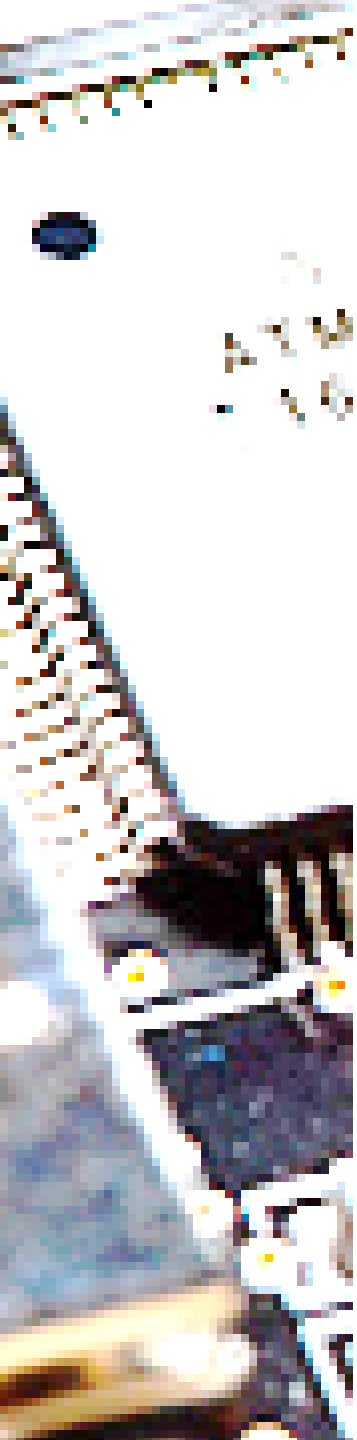
I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
            add   r22,r23     ; Add r23 to r22
            brcc  nocarry     ; Branch if carry cleared
            ...
nocarry:    nop               ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true



BRCS – Branch if Carry Set

Description:

Conditional relative branch. Tests the Carry Flag (C) and branches relatively to PC if C is set. This instr atively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC in two's complement form. (Equivalent to instruction BRBS 0,k).

Operation:

(i) If C = 1 then PC ← PC + k + 1, else PC ← PC + 1

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

16-bit Opcode:

1111	00kk	kkkk	k000

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
          cpi   r26,$56   ; Compare r26 with $56
          brcs  carry     ; Branch if carry set
          ...
carry:    nop             ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1 if condition is false
2 if condition is true

BREQ – Branch if Equal

Description:

Conditional relative branch. Tests the Zero Flag (Z) and branches relatively to PC if Z is set. 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 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 BRBS 1,k).

Operation:

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

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

16-bit Opcode:

1111	00kk	kkkk	k001

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
        cp    r1,r0     ; Compare registers r1 and r0
        breq  equal     ; Branch if registers equal
        ...
equal:  nop             ; Branch destination (do nothing)
```

BRGE – Branch if Greater or Equal (Signed)

Description:

Conditional relative branch. Tests the Signed Flag (S) and branches relatively to PC if S 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 signed binary number represented in Rd was greater than or equal to the 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 4,k).

Operation:

(i) If Rd ≥ Rr (N ⊕ V = 0) then PC ← PC + k + 1, else PC ← PC + 1

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

16-bit Opcode:

1111	01kk	kkkk	k100

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
            cp    r11,r12    ; Compare registers r11 and r12
            brge  greateq    ; Branch if r11 ≥ r12 (signed)
            ...
greateq:    nop              ; Branch destination (do nothing)
```

SBIW – Subtract Immediate from Word

Description:

Subtracts an immediate value (0-63) from a register pair and places the result in the register pair. This instruction operate 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)	SBIW Rd+1:Rd,K	d ∈ {24,26,28,30}, 0 ≤ K ≤ 63	PC ← PC + 1

16-bit Opcode:

1001	0111	KKdd	KKKK

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

S: N ⊕ V, For signed tests.

V: Rdh7 •$\overline{R15}$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R15
Set if MSB of the result is set; cleared otherwise.

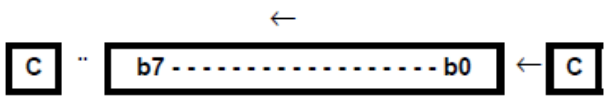
Z: $\overline{R15}$• $\overline{R14}$ •$\overline{R13}$ •$\overline{R12}$ •$\overline{R11}$• $\overline{R10}$• $\overline{R9}$• $\overline{R8}$• $\overline{R7}$• $\overline{R6}$ •$\overline{R5}$• $\overline{R4}$• $\overline{R3}$ •$\overline{R2}$• $\overline{R1}$• $\overline{R0}$
Set if the result is $0000; cleared otherwise.

ROL – Rotate Left trough Carry

Description:

Shifts all bits in Rd one place to the left. The C Flag is shifted into bit 0 of Rd. Bit 7 is shifted into the C Flag. This operation, combined with LSL, effectively multiplies multi-byte signed and unsigned values by two.

Operation:



	Syntax:	Operands:	Program Counter:
(i)	ROL Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode: (see ADC Rd,Rd)

0001	11dd	dddd	dddd

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H: Rd3

S: N ⊕ V, For signed tests.

V: N ⊕ C (For N and C after the shift)

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: Rd7
Set if, before the shift, the MSB of Rd was set; cleared otherwise.

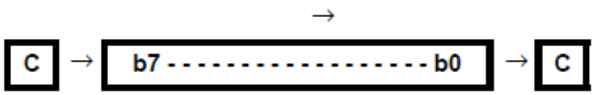
R (Result) equals Rd after the operation

ROR – Rotate Right through Carry

Description:

Shifts all bits in Rd one place to the right. The C Flag is shifted into bit 7 of Rd. Bit 0 is shifted into the C Flag. This operation, combined with ASR, effectively divides multi-byte signed values by two. Combined with LSR it effectively divides multi-byte unsigned values by two. The Carry Flag can be used to round the result.

Operation:



	Syntax:	Operands:	Program Counter:
(i)	ROR Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0111

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

S: N ⊕ V, For signed tests.

V: N ⊕ C (For N and C after the shift)

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: Rd0
Set if, before the shift, the LSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

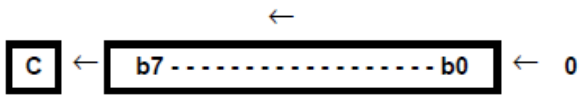
LSL – Logical Shift Left

Description:

Shifts all bits in Rd one place to the left. Bit 0 is cleared. Bit 7 is loaded into the C Flag of the SREG. This operation effectively multiplies signed and unsigned values by two.

Operation:

(i)



	Syntax:	Operands:	Program Counter:
(i)	LSL Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode: (see ADD Rd,Rd)

0000	11dd	dddd	dddd

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H: Rd3

S: N ⊕ V, For signed tests.

V: N ⊕ C (For N and C after the shift)

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

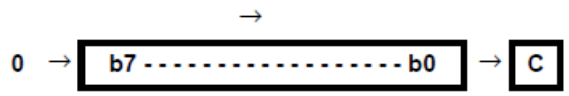
C: Rd7
Set if, before the shift, the MSB of Rd was set; cleared otherwise.

LSR – Logical Shift Right

Description:

Shifts all bits in Rd one place to the right. Bit 7 is cleared. Bit 0 is loaded into the C Flag of the SREG. This operation effectively divides an unsigned value by two. The C Flag can be used to round the result.

Operation:



	Syntax:	Operands:	Program Counter:
(i)	LSR Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0110

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	0	⇔	⇔

S: N ⊕ V, For signed tests.

V: N ⊕ C (For N and C after the shift)

N: 0

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: Rd0
Set if, before the shift, the LSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

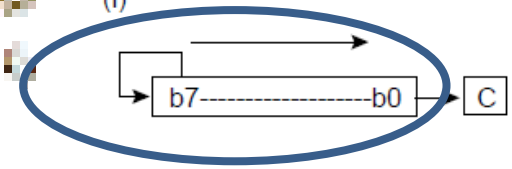
ASR – Arithmetic Shift Right

Description:

Shifts all bits in Rd one place to the right. Bit 7 is held constant. Bit 0 is loaded into the C Flag of the SREG. This operation effectively divides a signed value by two without changing its sign. The Carry Flag can be used to round the result.

Operation:

(i)



	Syntax:	Operands:	Program Counter:
(i)	ASR Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0101

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

S: N ⊕ V, For signed tests.

V: N ⊕ C (For N and C after the shift)

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: Rd0
Set if, before the shift, the LSB of Rd was set; cleared otherwise.

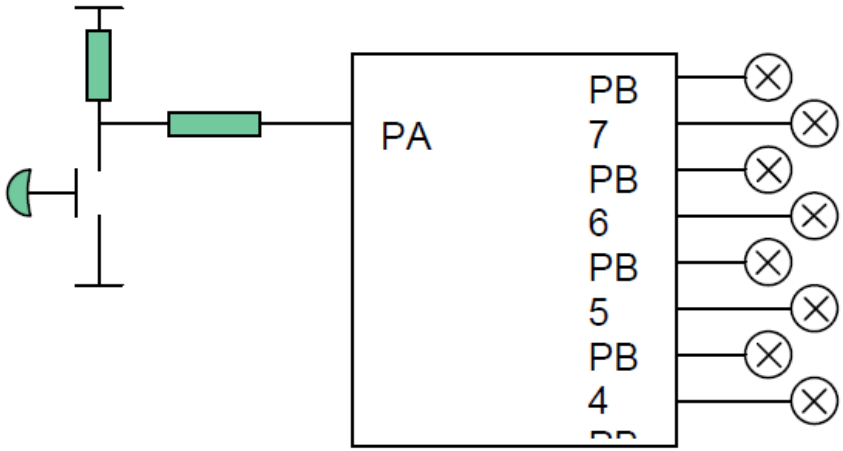
R (Result) equals Rd after the operation.

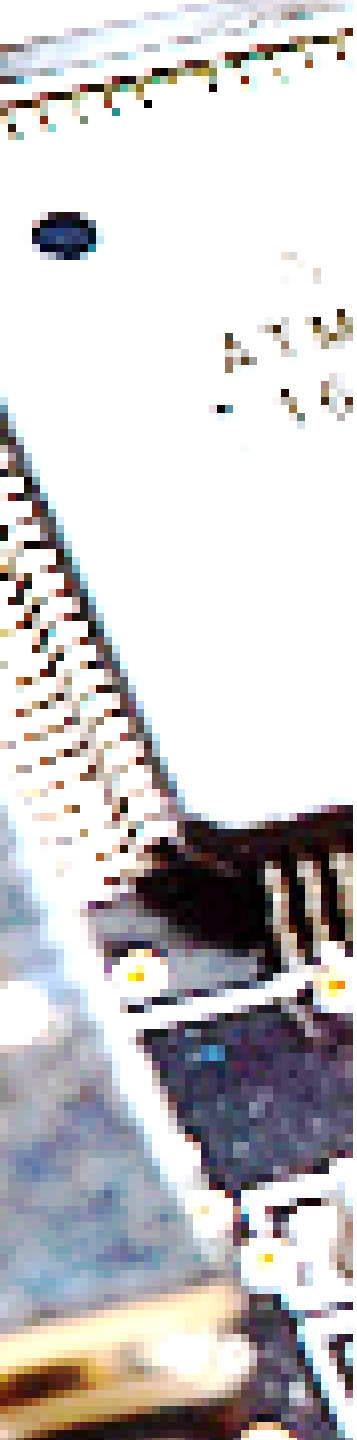
Lab 2, task 1..

Task 1:

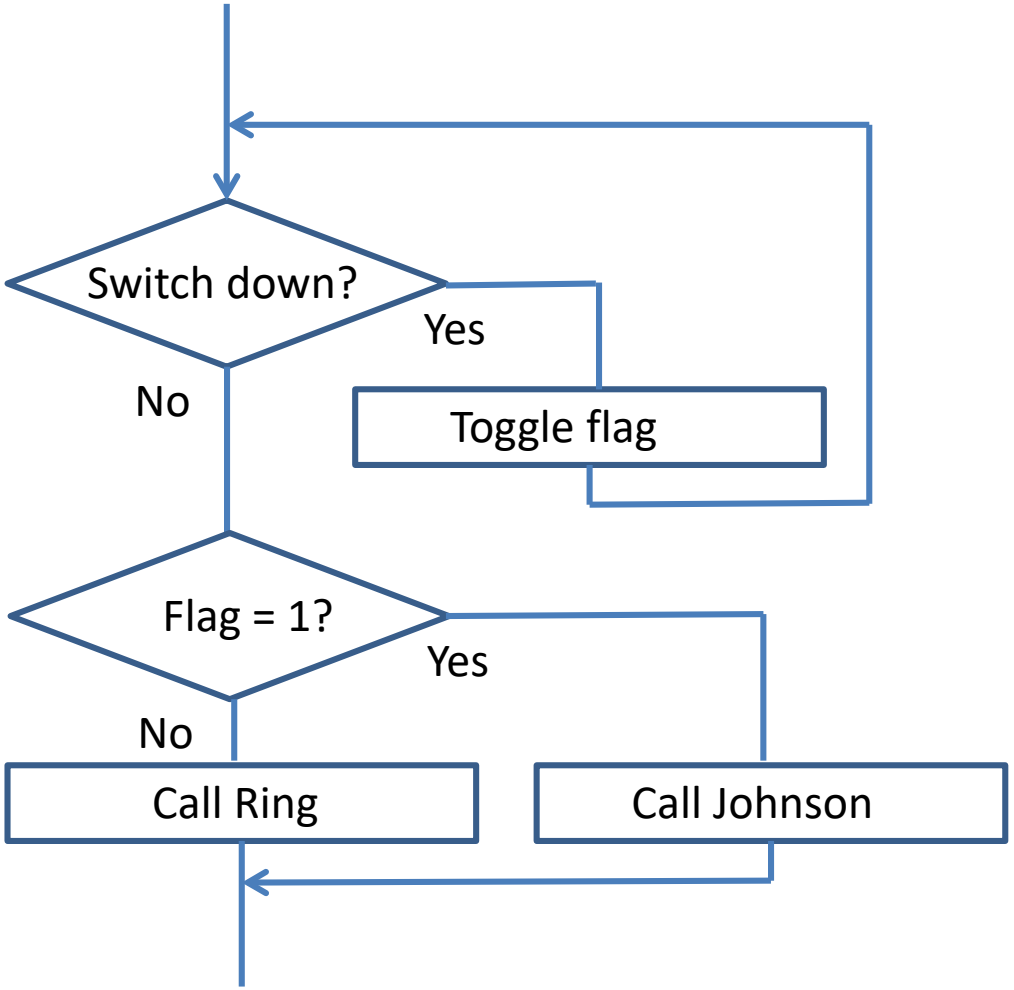
Switch – Ring counter / Johnson counter

Write a program which switch between Ring counter and Johnson counter. You should not use Interrupt in this lab. The pushbutton must be checked frequently, so there is no delay between the button is pressed and the change between Ring/Johnson. Use SW0 (PA0) for the button. Each time you press the button, the program should change counter.

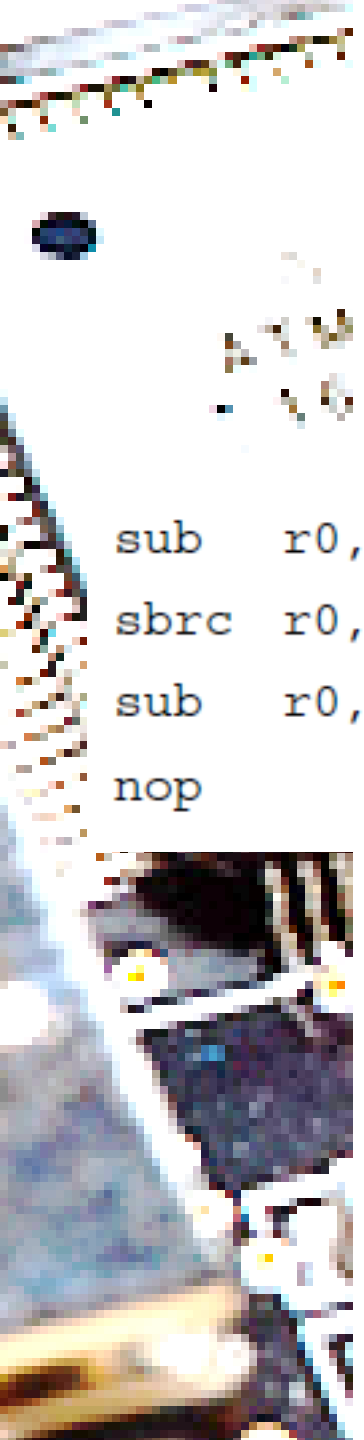




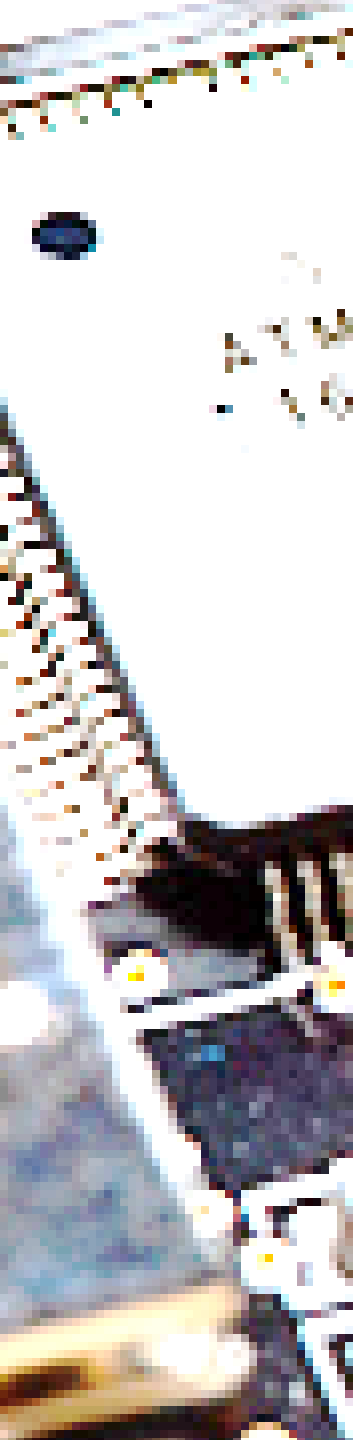
Check if switch is pressed.



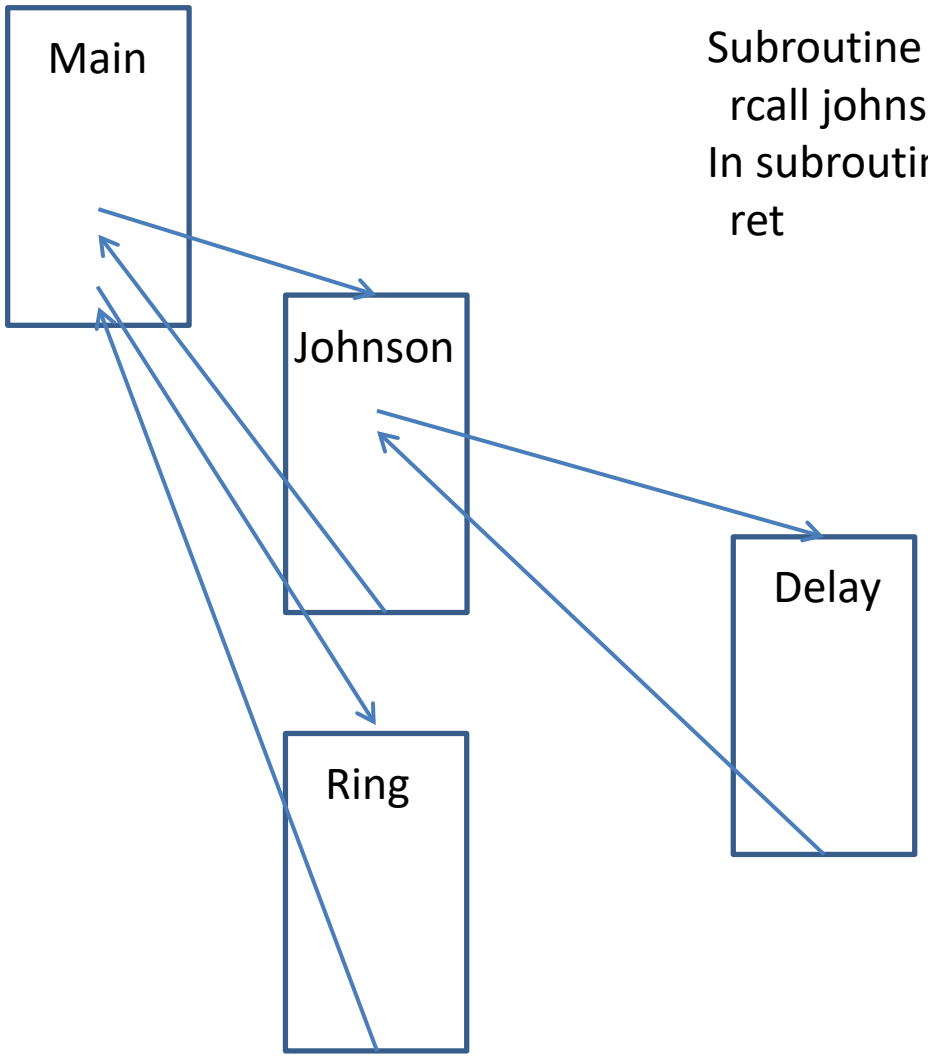
Example:



```
sub   r0,r1      ; Subtract r1 from r0
sbrc  r0,7       ; Skip if bit 7 in r0 cleared
sub   r0,r1      ; Only executed if bit 7 in r0 not cleared
nop              ; Continue (do nothing)
```



Program structure, example.



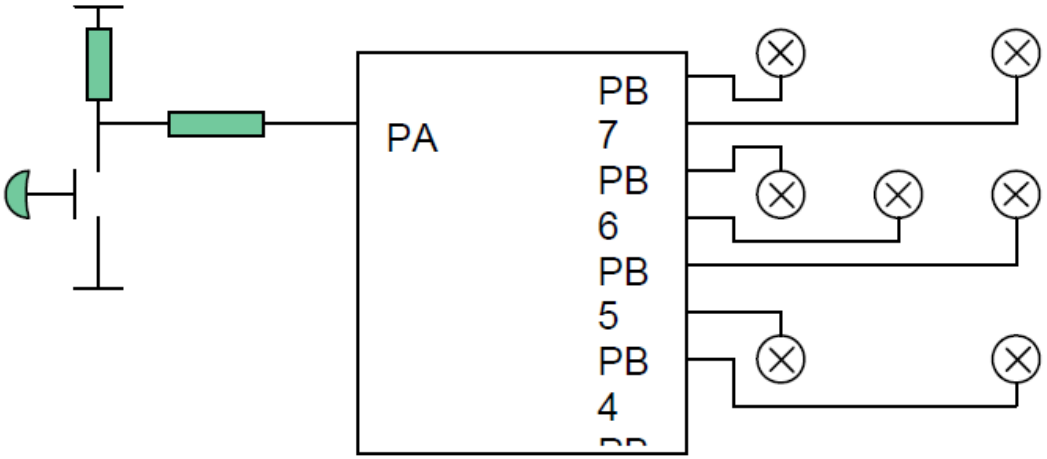
Subroutine call, example:
rcall johnsson
In subroutine:
ret

Question:
Where to check the switch?

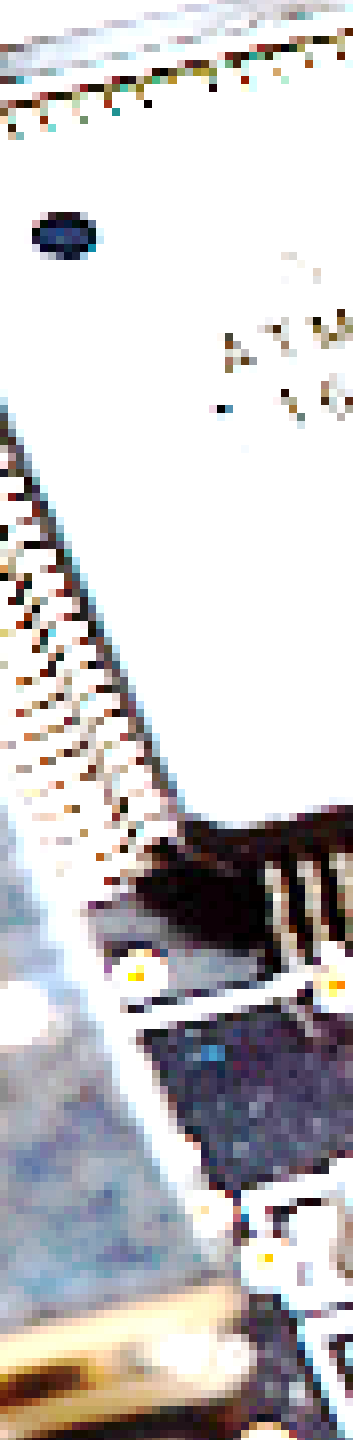
Lab 2, task 2.

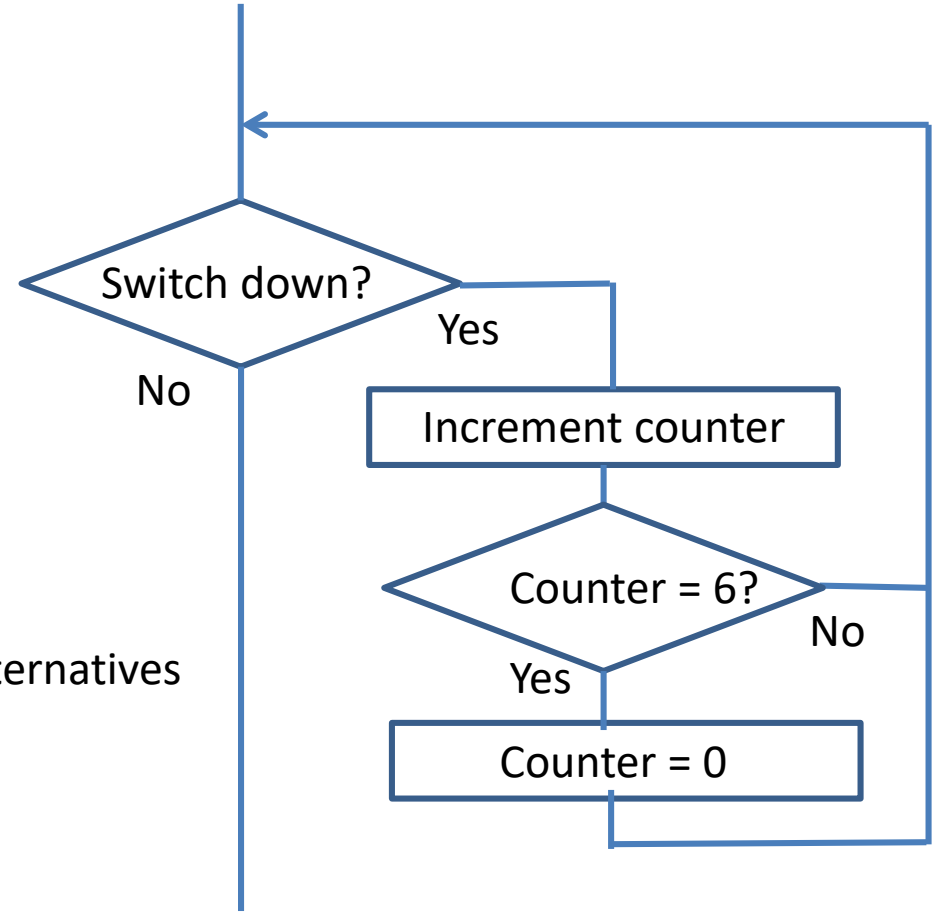
Task 2:
Electronic dice

You should create an electronic dice. Think of the LEDs placed as in the picture below. The number 1 to 6 should be generated randomly. You could use the fact that the time you press the button varies in length.

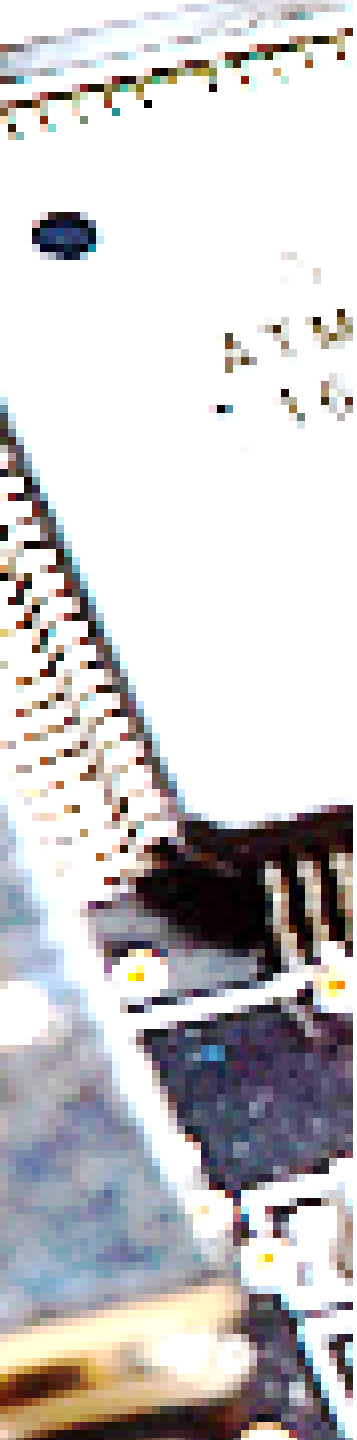


Example, random generator.

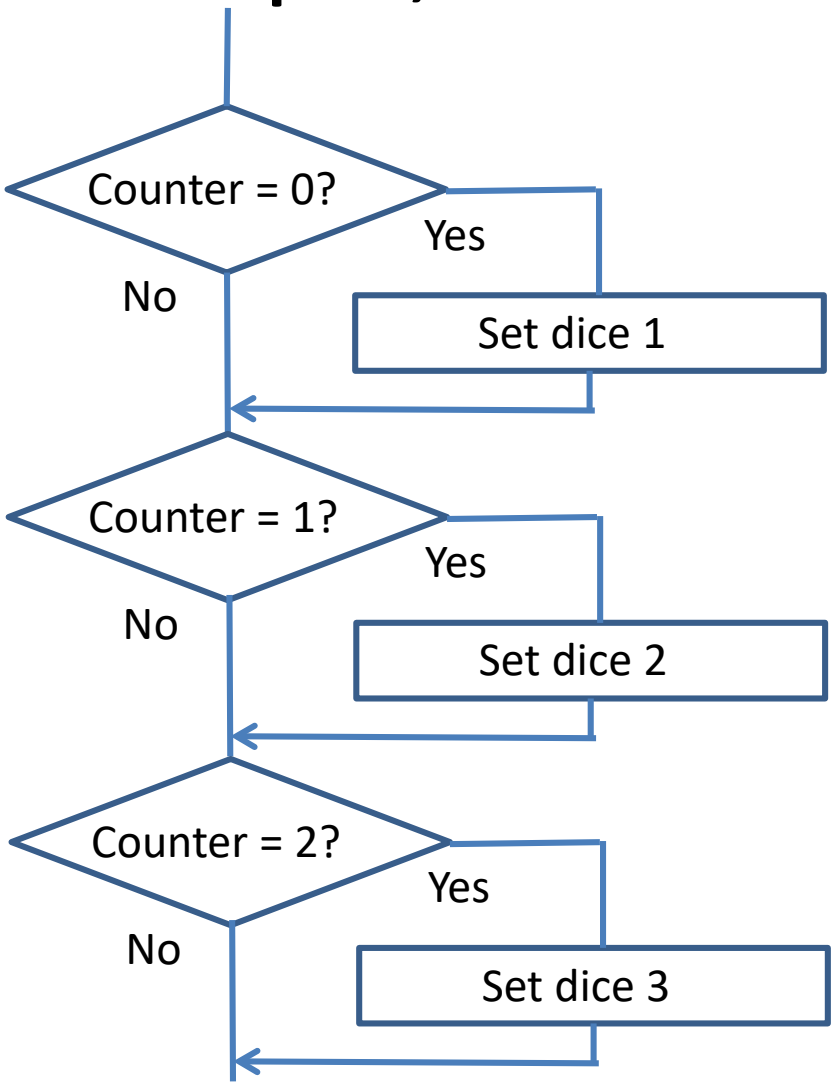




Important:
Check that all alternatives are equal.



Example, dice.

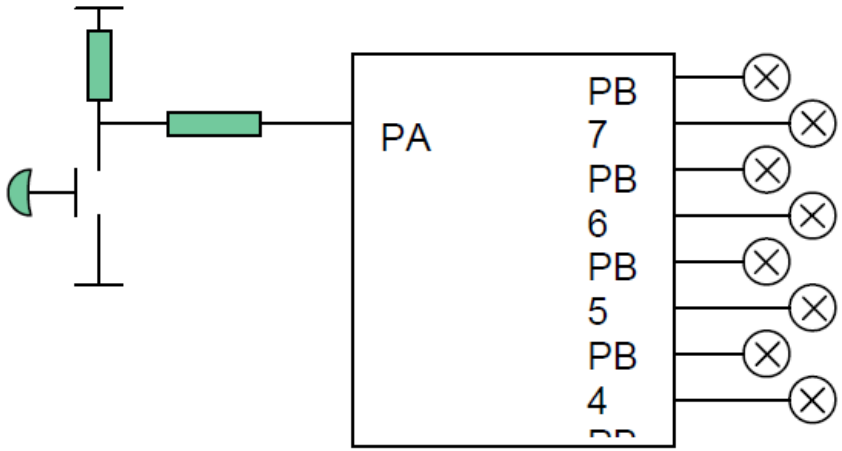


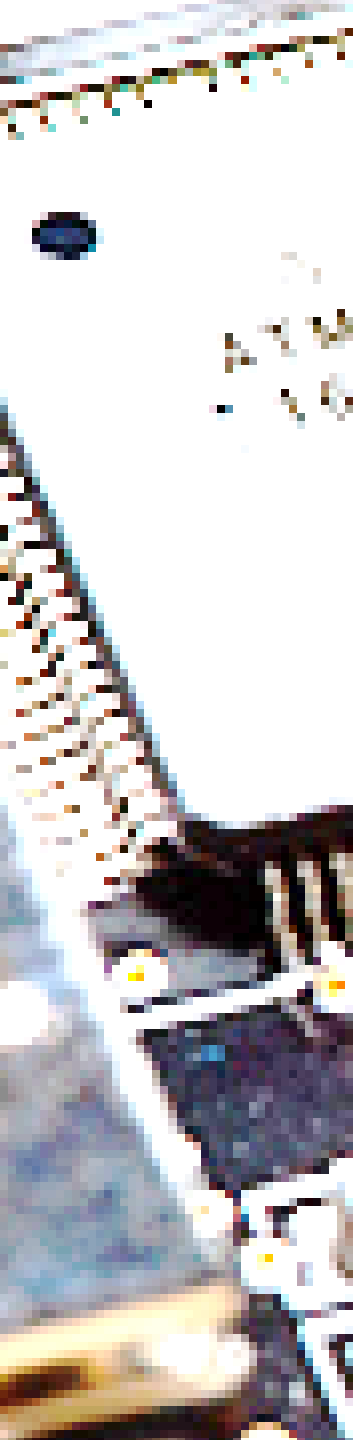
Und so weiter…

Lab 2, task 3.

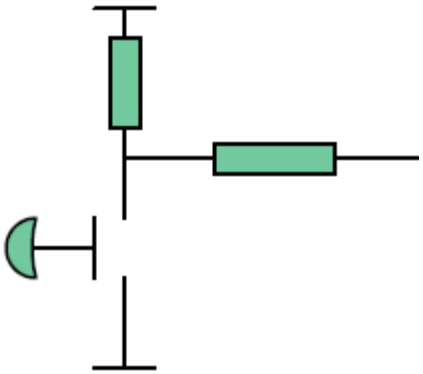
Task 3:
Change counter

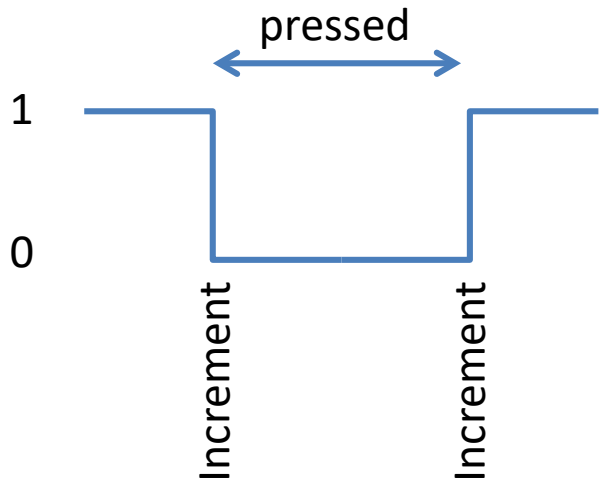
Skriv ett program som klarar av att beräkna förändringarna på en strömställare. Som förändring räknar vi när strömställaren SW0 går från 0 till 1 respektive 1 till 0, vi räknar alltså positiva och negativa flanker. Vi räknar förändringarna i en byte variabel och lägger ut dess värde på PORTB.
Write a program that is able to calculate the changes on a switch. As a change we expect when the switch SW0 goes from 0 to 1 and from 1 to 0, we expect therefore positive and negative edges. We calculate the changes in a byte variable and puts out its value on PORTB.

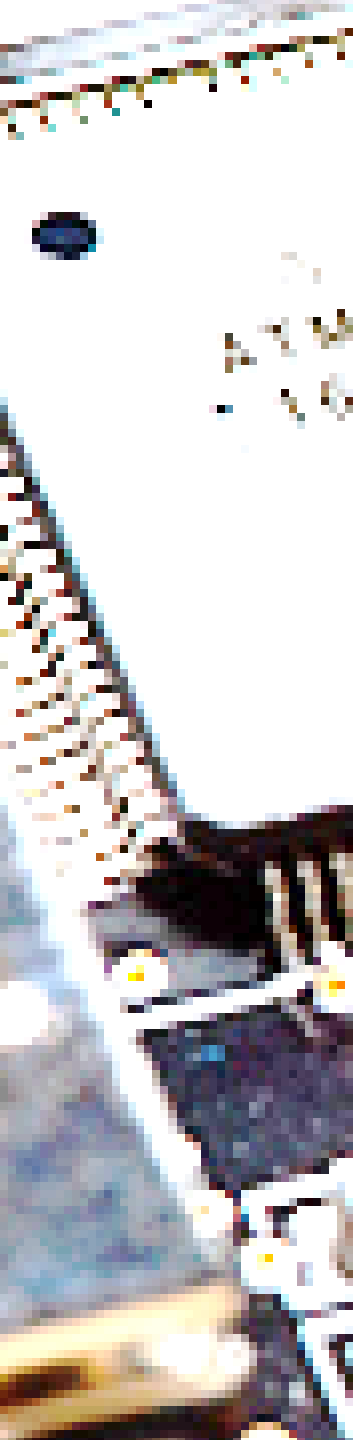




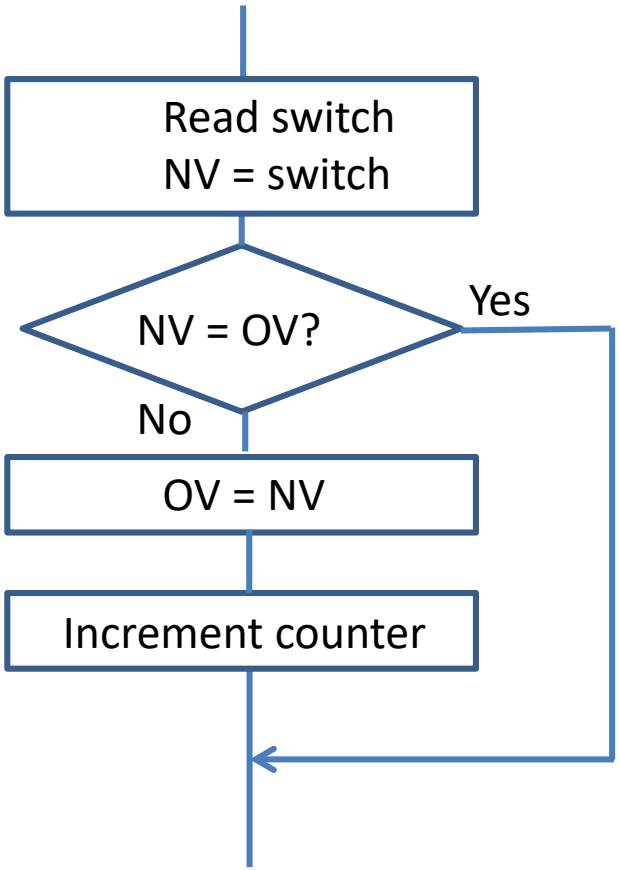
Switch.





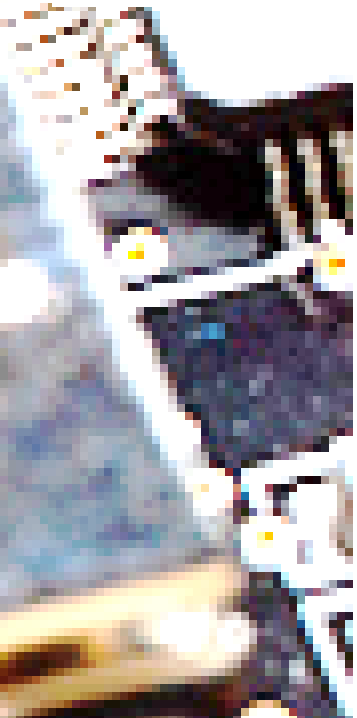


Example, counter.



Lab 2, task 4.

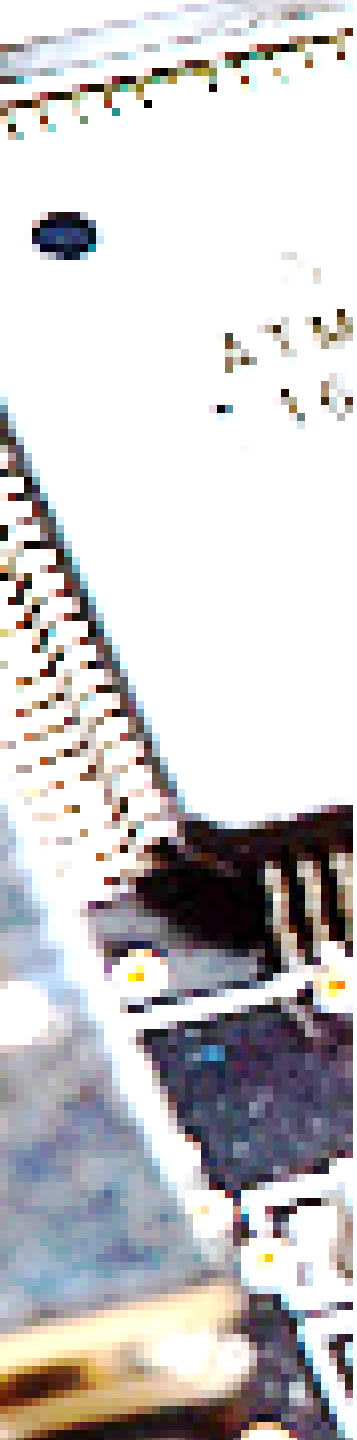
Task 4:
Delay subroutine with variable delay time
Modify the program in task 5 in Lab 1 to a general delay routine that can be called from other programs. It should be named **wait_milliseconds.** The number of milliseconds should be transferred to register pair R24, R25.



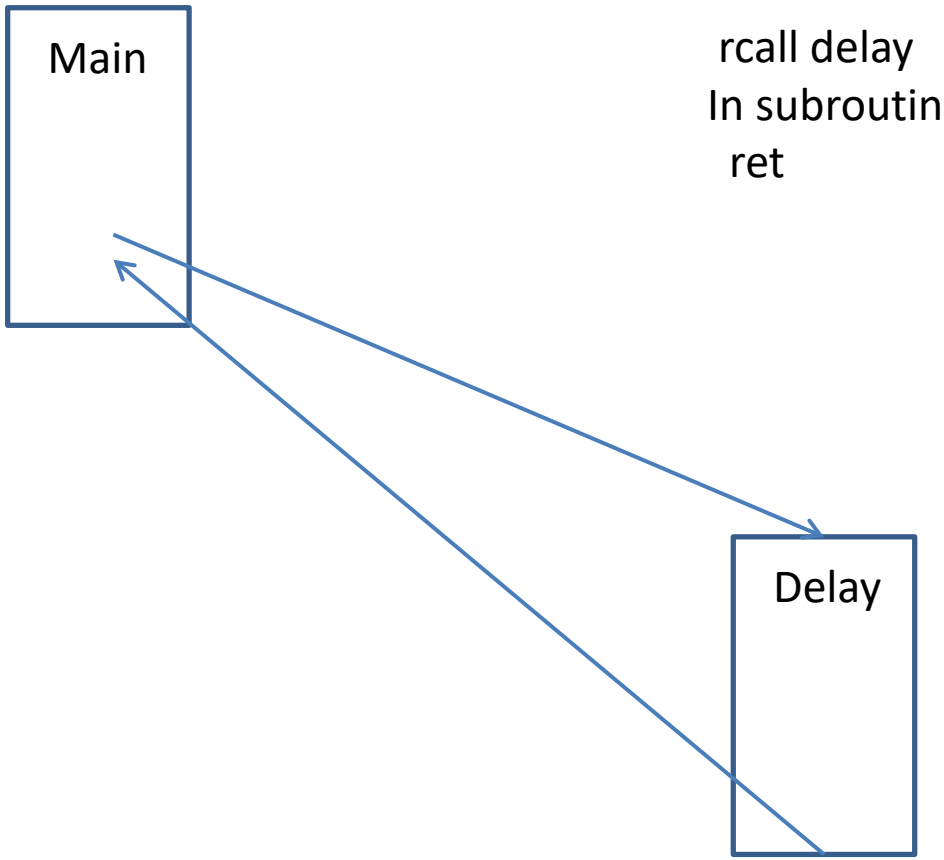
Example, C language:

num_day = function3(number);

function3() is a routine or function, *number* is a variable transferred to the subroutine.
The variable *num_day* will get the return value from the subroutine *function3().*



Program structure, example.

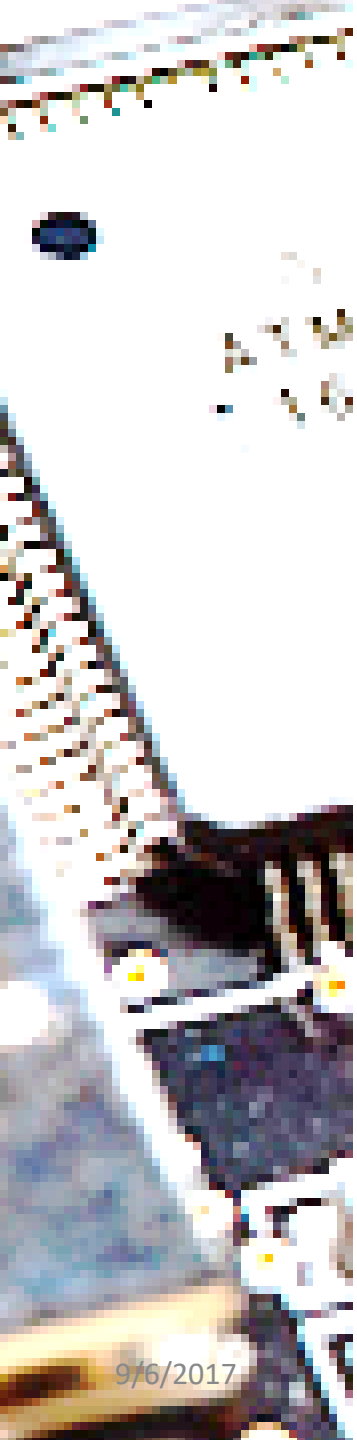


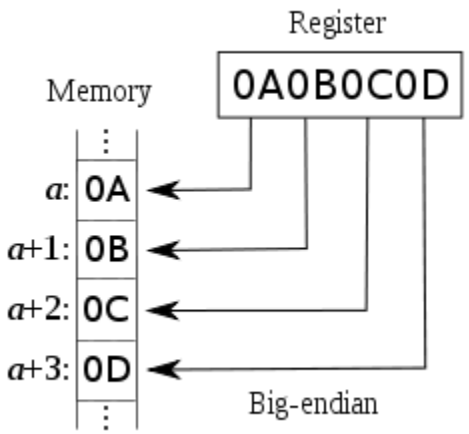
rcall delay
In subroutine:
ret

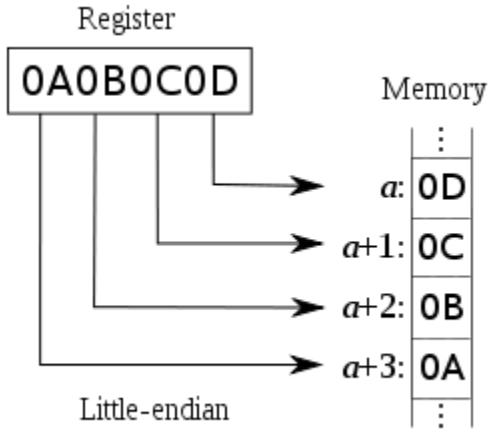
Question:
Where to check the switch?

Big-Endian – Little-Endian

(Pictures from Wikipedia)

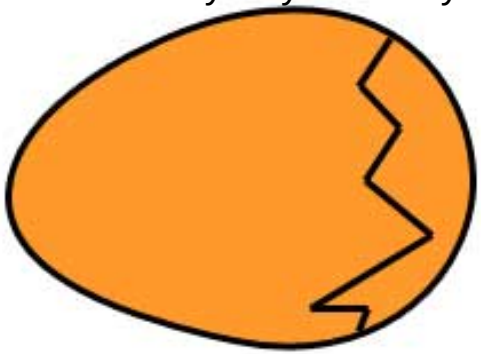




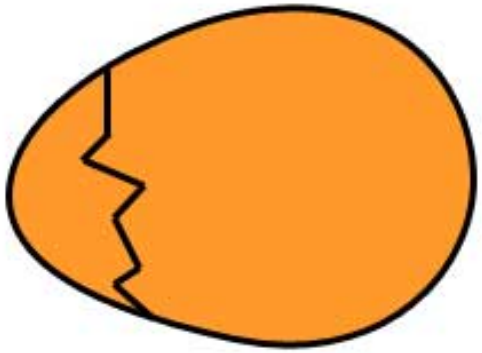


Big Endian – Little Endian

- The term big endian originally comes from Jonathan Swift's satirical novel Gulliver's Travels by way of Danny Cohen in 1980.[1]



BIG ENDIAN - The way people always broke their eggs in the Lilliput land



LITTLE ENDIAN - The way the king then ordered the people to break their eggs

Picture from: http://flickeringtubelight.net/blog/2004/05/big-endian-and-little-endian-storage-schemes-how-to-remember/

- **Atmel AVR Endianess Cheat Sheet**:

Example Value: 0x0A0B

All 8-bt AVRs: Little Endian

in Memory 0x0B 0x0A (*adr: 0x0B, *(adr+1): 0x0A)

(AVR32: Big Endian in Memory 0x0A 0x0B)

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 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

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

S: N ⊕ V, For signed tests.

V: $\overline{Rdh7} \bullet R15$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R15
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $0000; cleared otherwise.

C: $\overline{R15} \bullet Rdh7$
Set if there was carry from the MSB of the result; cleared otherwise.

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

S: $N \oplus V$, For signed tests.

V: $\overline{Rdh7} \bullet R15$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R15
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $0000; cleared otherwise.

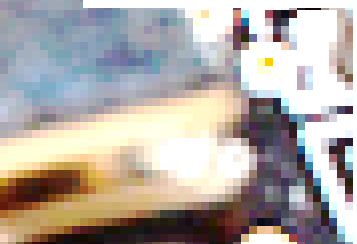
C: $\overline{R15} \bullet Rdh7$
Set if there was carry from the MSB of the result; cleared otherwise.

R (Result) equals Rdh:Rdl after the operation (Rdh7-Rdh0 = R15-R8, Rdl7-Rdl0=R7-R0).

Example:

```
adiw  r25:24,1  ; Add 1 to r25:r24
adiw  ZH:ZL,63  ; Add 63 to the Z-pointer(r31:r30)
```

Words: 1 (2 bytes)
Cycles: 2



SBIW – Subtract Immediate from Word

Description:

Subtracts an immediate value (0-63) from a register pair and places the result in the register pair. This instruction operate 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)	SBIW Rd+1:Rd,K	d ∈ {24,26,28,30}, 0 ≤ K ≤ 63	PC ← PC + 1

16-bit Opcode:

1001	0111	KKdd	KKKK

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

S: N ⊕ V, For signed tests.

V: Rdh7 •$\overline{R15}$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R15
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R15}$• $\overline{R14}$ •$\overline{R13}$ •$\overline{R12}$ •$\overline{R11}$• $\overline{R10}$• $\overline{R9}$• $\overline{R8}$• $\overline{R7}$• $\overline{R6}$ •$\overline{R5}$• $\overline{R4}$• $\overline{R3}$ •$\overline{R2}$• $\overline{R1}$• $\overline{R0}$
Set if the result is $0000; cleared otherwise.

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

S: $N \oplus V$, For signed tests.

V: $Rdh7 \bullet \overline{R15}$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R15
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $0000; cleared otherwise.

C: $R15 \bullet \overline{Rdh7}$
Set if the absolute value of K is larger than the absolute value of Rd; cleared otherwise.

R (Result) equals Rdh:Rdl after the operation (Rdh7-Rdh0 = R15-R8, Rdl7-Rdl0=R7-R0).

Example:

```
sbiw    r25:r24,1   ; Subtract 1 from r25:r24
sbiw    YH:YL,63    ; Subtract 63 from the Y-pointer(r29:r28)
```

Words: 1 (2 bytes)
Cycles: 2