XE36SKD

AVR Instruction Set

Arithmetic and Logic Operations

Exercise 5

- Arithmetic operations
- Logic operations
- Rotations and shifts

Basic Register Operations

- Clear a register
- Setting a register
- Moving data between registers

Clear a Register

```
clr r0
```

Examples:

```
clr r1
clr r2
clr r18
```

Setting a register

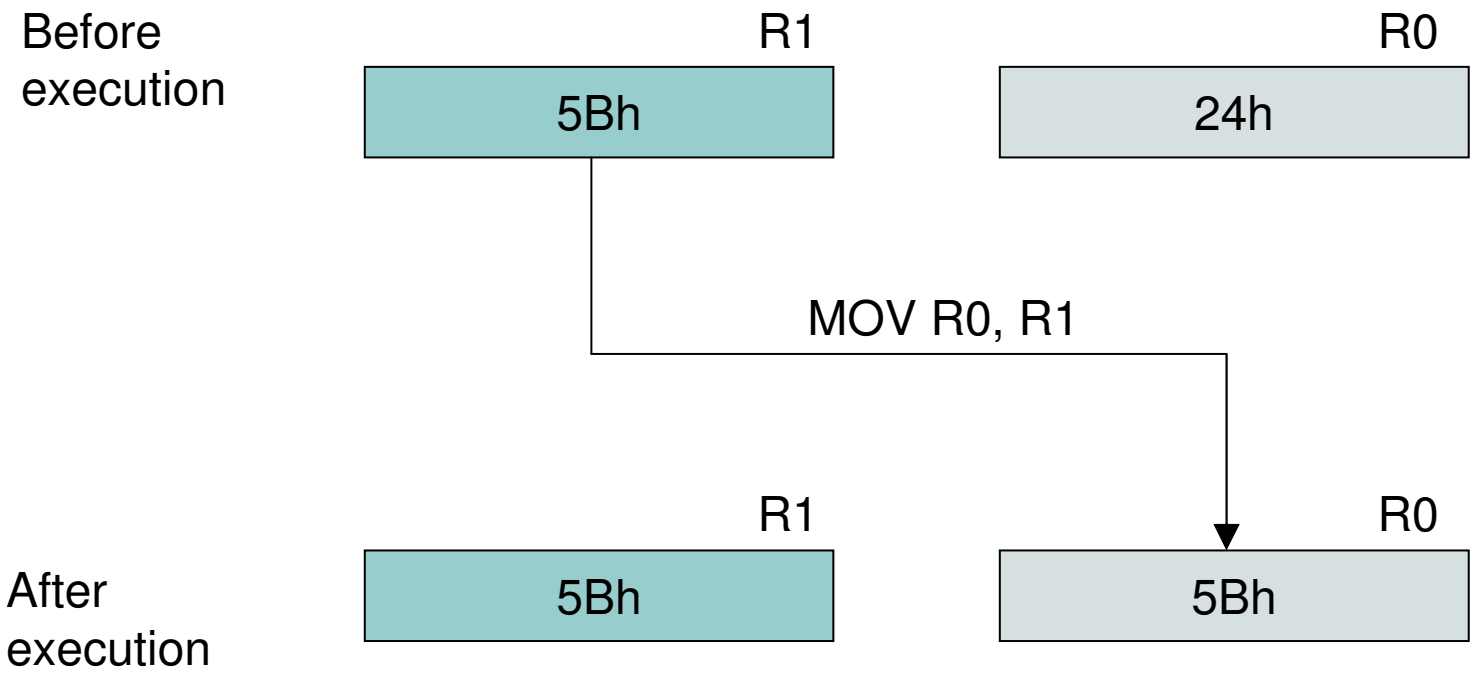
```
ldi r16, 0x3F
```

Examples:

```
ldi r16, 15             ; decimal (positive)
ldi r18, -10            ; decimal (negative)
ldi r24, 0xA4           ; hexadecimálně
ldi r31, 0b10100101     ; binárně
```

Only the R16-R31 registers !

Moving Data Between Registers



In this context, by moving we mean actually **copying** the contents of one register to another. That is, the source register retains its value.

Examples

```
ldi r16, 0x10
ldi r18, 0x23
ldi r23, 0x31

mov r1 , r16  ; moves the content of R16 into R1
mov r16, r18  ; moves the content of R18 into R16
mov r0 , r23  ; moves the content of R23 into R0
mov r5 , r0   ; moves the content of R0 into R5
```

Task: exchange content of the R30 a R31 registers

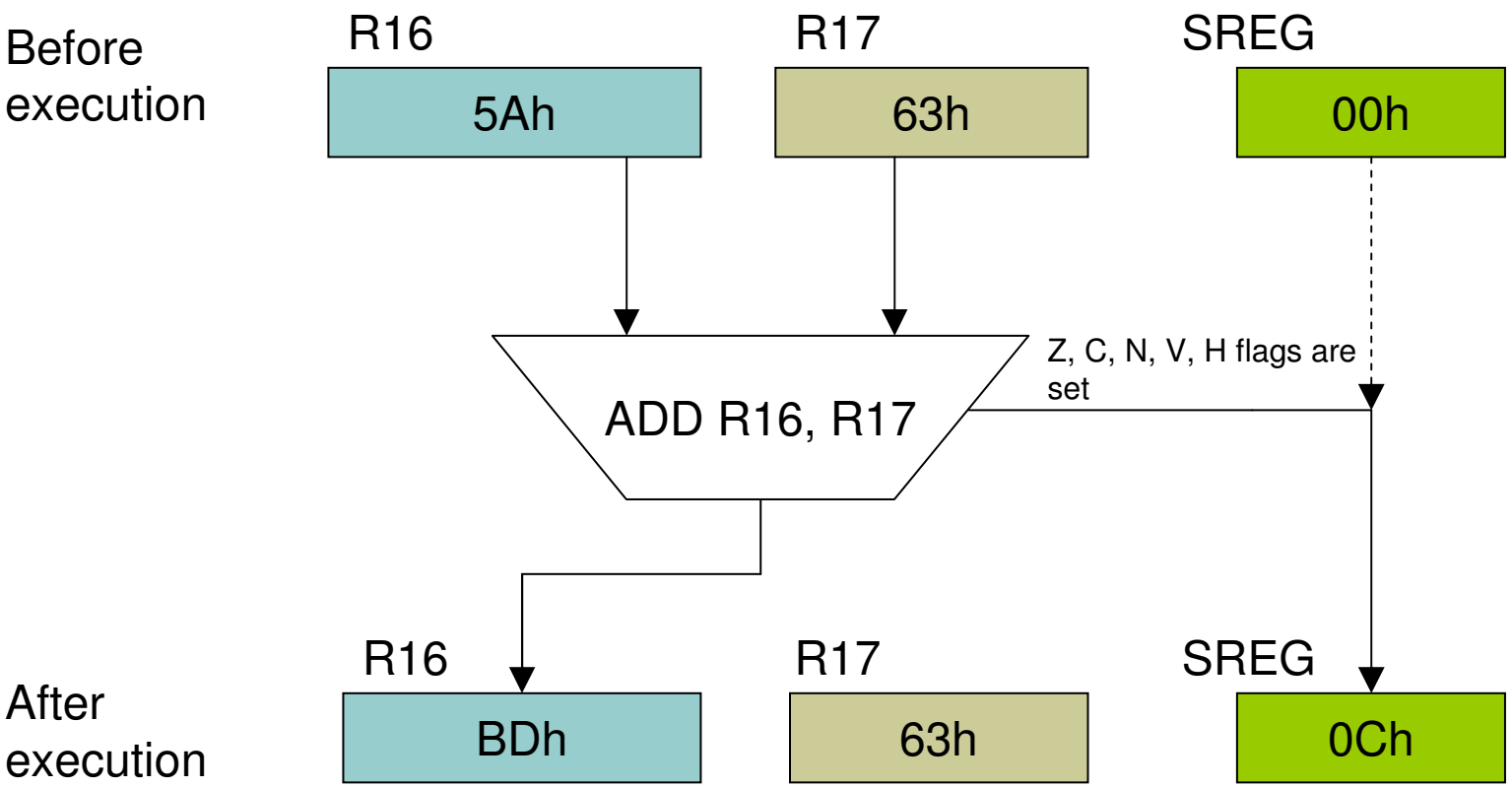
```
ldi r30, 0x10
ldi r31, 0x20
```

… write instructions here …

Result expected:

R30	R31
0x20	0x10

8-bit Number Addition

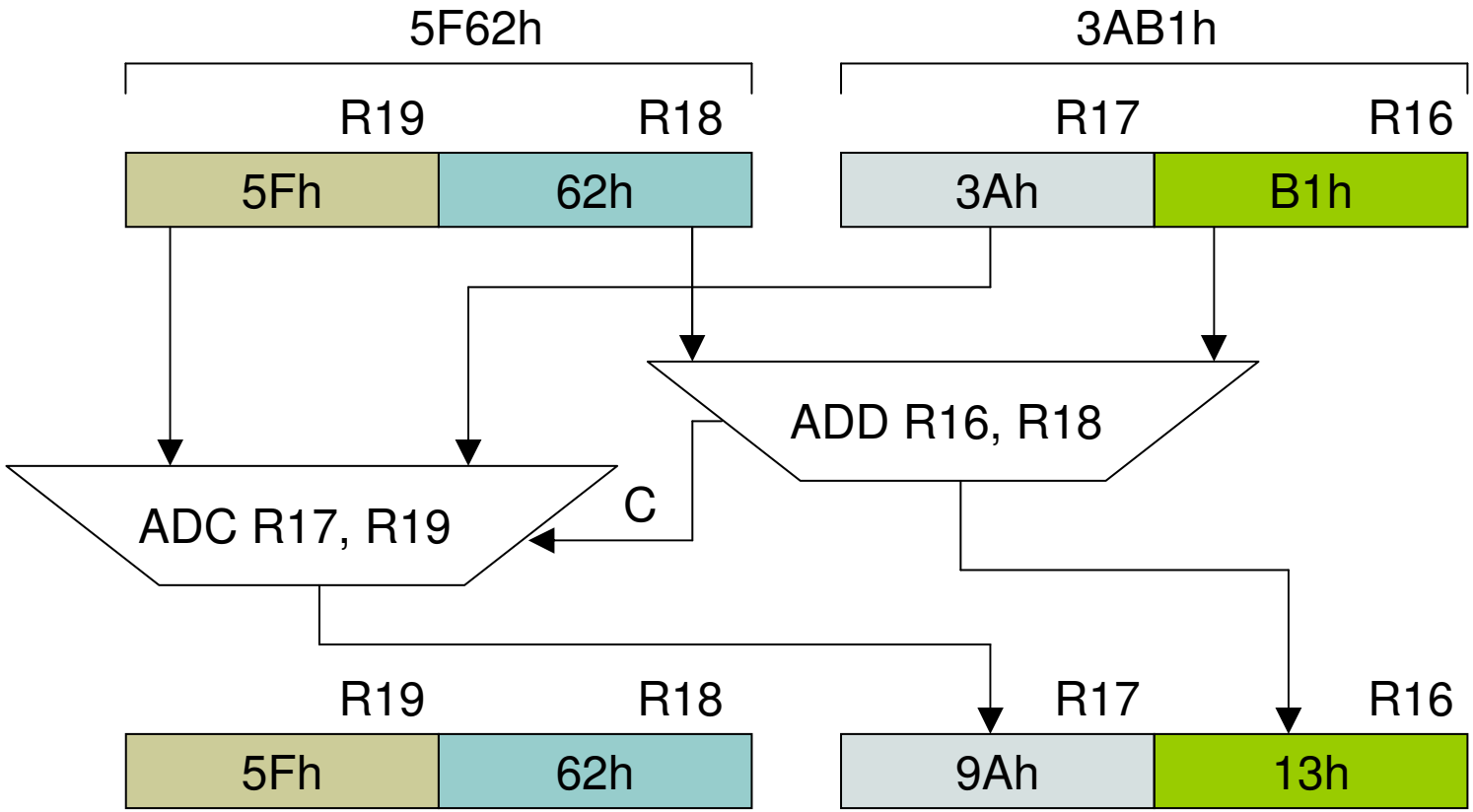


Program Example

```
LDI R16, 0x5A
LDI R17, 0x63
ADD R16, R17
```

Execute this program step-by-step and check the result in the R16 register.

16-bit Number Addition



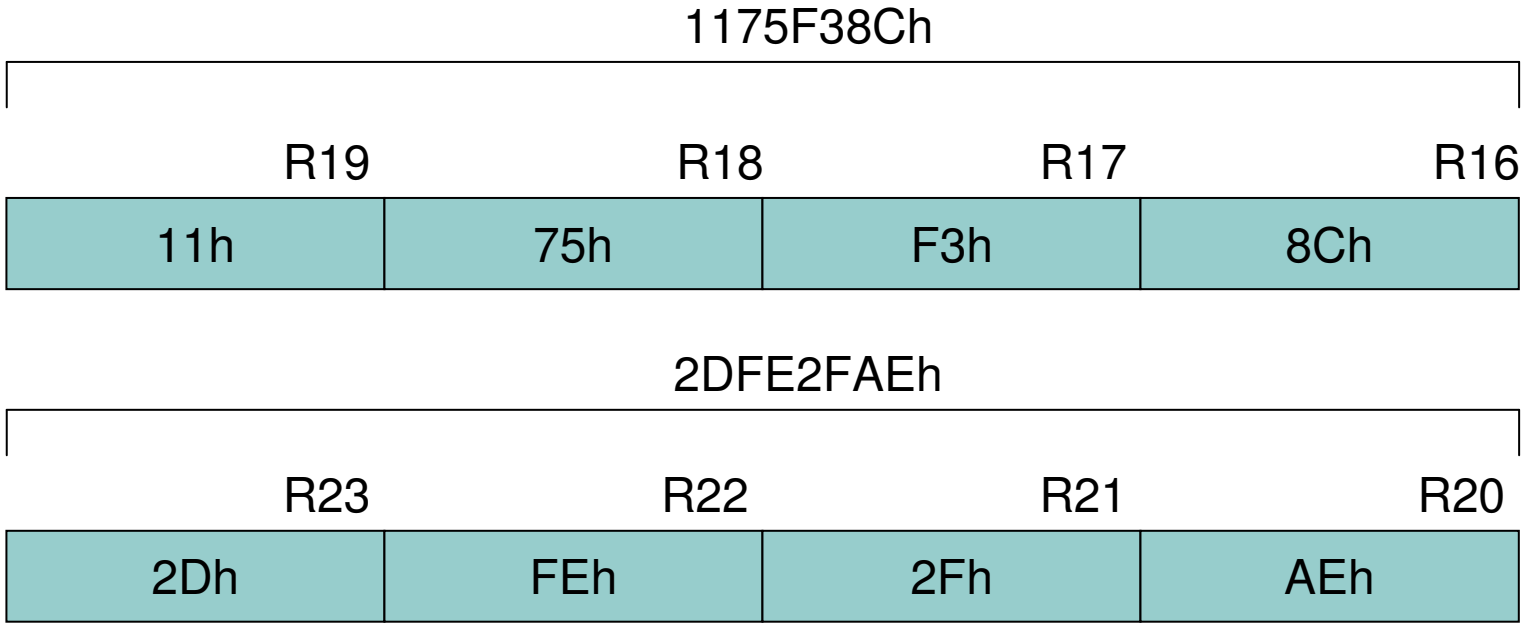
Example

```
LDI R16, 0xB1
LDI R17, 0x3A
LDI R18, 0x62
LDI R19, 0x5F

ADD R16, R18    ; ADD instruction sets the carry bit (C)
ADC R17, R19    ; ADC instruction adds carry bit to the result
```

Execute this program step-by-step and check the result in the R16 and R17 registers.

Task: 32-bit number addition

Write a program for adding of two 32-bits numbers. The numbers are stored in registers as follows:



Answer questions

- What is the result of addition ?
- How many ADD and ADC instructions you used ?
- Write carry bit state after each ADD and ADC instruction.
  Use paper and pencil only.
- Check carry bit states by running the program step-by-step.

Task: calculate following expression

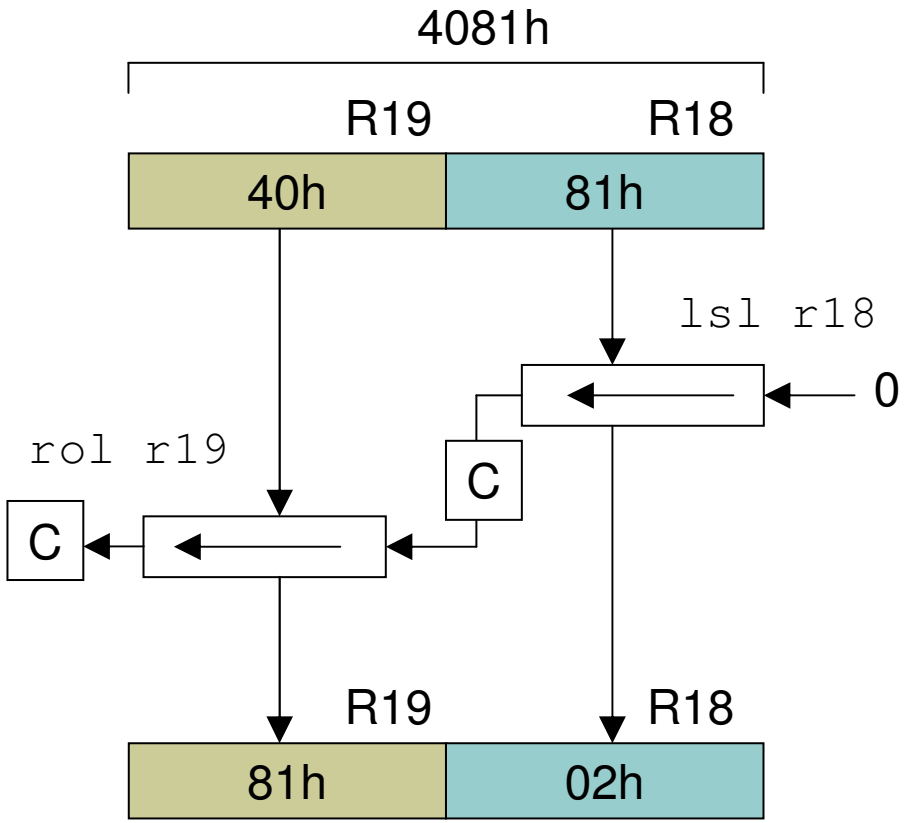
R20 = (4 * R16 + 3 * R17 - R18) / 8

```
LDI R16, 5
LDI R17, 10
LDI R18, 58

… write here next instructions …
```

Answer Questions

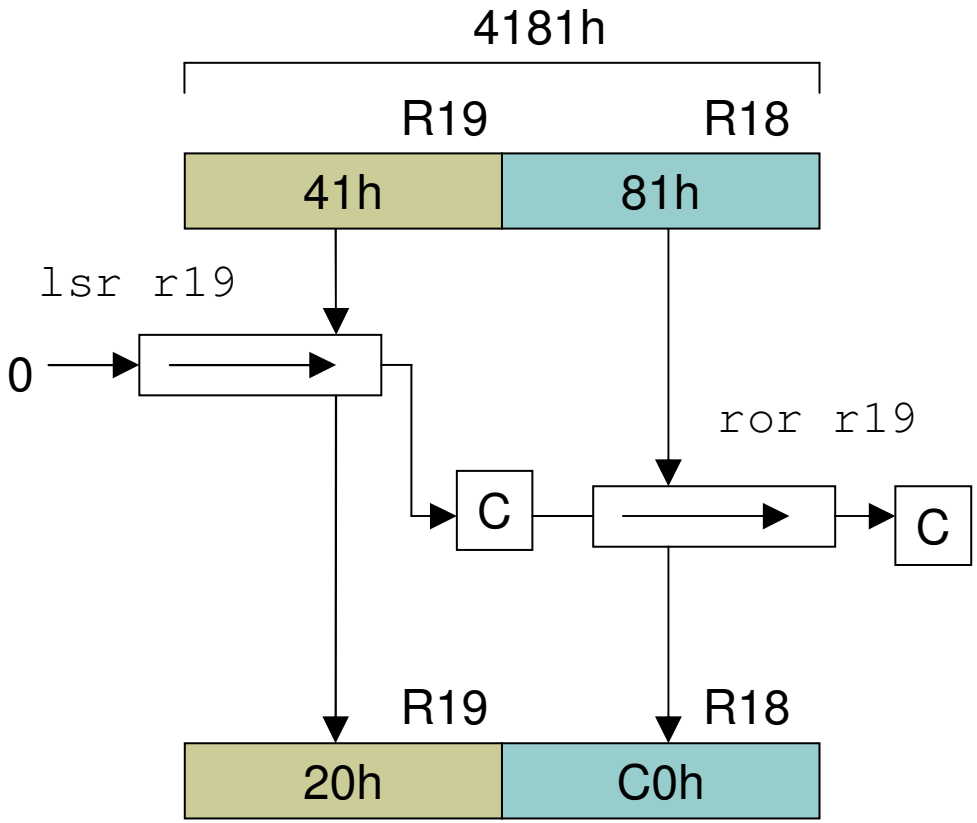
- How we can realize the multiplication by a power of two ?
- How we can realize the multiplication by a small constant (for example 3, 5, 10) ?
- How we can realize the division by a power of two ?
- Is the result correct if we load the R16 and R17 registers by negative values. Test it !
- What is the error ? A solution is to replace the LSR instruction by the ASR instruction.

Logic Shift of the 16-bits Number (1x to the left)



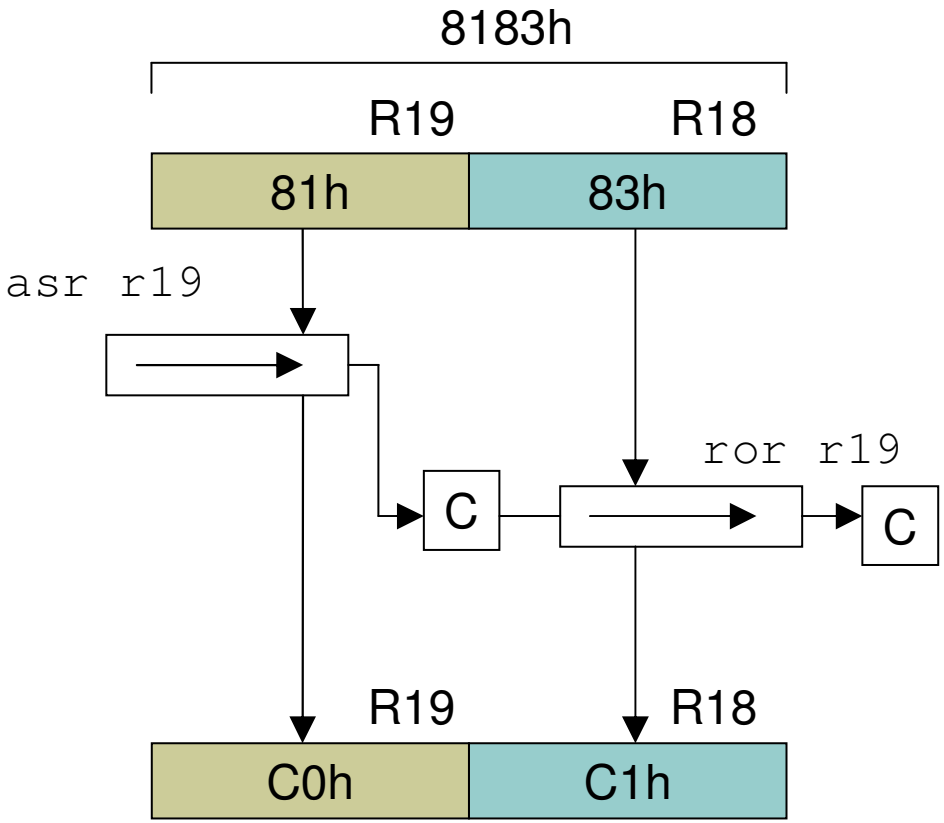

```
lsl     r18
rol     r19
```

Logic Shift of the 16-bits Number (1x to the right)




```
lsr     r19
ror     r18
```

Arithmetic Shift of the 16-bits Number (1x to the right)




```
asr     r19
ror     r18
```

Task: write a program for arithmetic shift of the 32-bit number (1x to the right)

Number: CD1C9BFAh

```
ldi     r16, 0xFA
ldi     r17, 0x9B
ldi     r18, 0x1C
ldi     r19, 0xCD

… write instructions here …
```

Bit Setting by Logical Operations

Set the bit 3 of the R16 register to 1

```
clr     r16
ori     r16, 0b00001000
```

Set the bits 4 and 5 bit of the R16 register to 1

```
clr     r16
ori     r16, 0b00110000
```

Mask

Set bits (bit-by-bit) in R16 to 1

```
clr     r16
ori     r16, 0b00000001 ; bit 0
ori     r16, 0b00000010 ; bit 1
ori     r16, 0b00000100 ; bit 2
ori     r16, 0b00001000 ; bit 3
ori     r16, 0b00010000 ; bit 4
ori     r16, 0b00100000 ; bit 5
ori     r16, 0b01000000 ; bit 6
ori     r16, 0b10000000 ; bit 7
```

Bit Clearing by Logical Operations

Set bit 3 of the R16 register to 0

```
ldi     r16, 0xFF
andi    r16, 0b11110111
```

Set bits 1 and 6 of R16 to 0

```
ldi     r16, 0xFF
andi    r16, 0b10111101
```

Mask

Set bits (bit-by-bit) in R16 to 0

```
ldi     r16, 0xFF
andi    r16, 0b11111110 ; bit 0
andi    r16, 0b11111101 ; bit 1
andi    r16, 0b11111011 ; bit 2
andi    r16, 0b11110111 ; bit 3
andi    r16, 0b11101111 ; bit 4
andi    r16, 0b11011111 ; bit 5
andi    r16, 0b10111111 ; bit 6
andi    r16, 0b01111111 ; bit 7
```

When the mask is in the register …

```
ldi     r16, 0xFF
ldi     r17, 0b11110111
and     r16, r17
```

Mask

```
clr     r16
ldi     r17, 0b01110000
or      r16, r17
```

Mask

If you need to complement bits ...

Complement all bits at once

```
ldi     r16, 0xAA
com     r16

Result: R16 = 55h
```

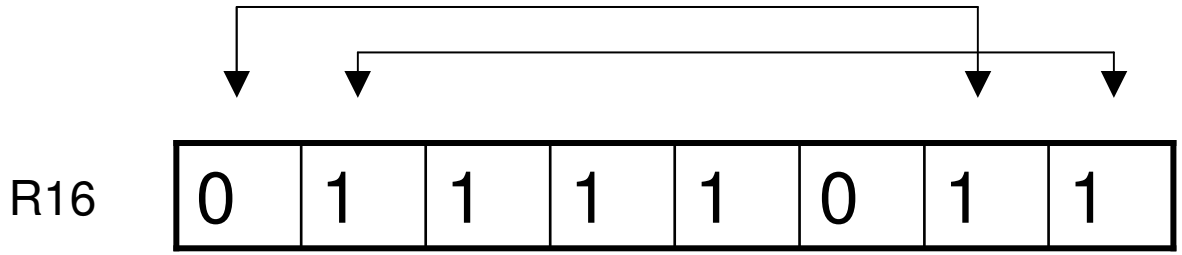
Complement the lower nibble of R16

```
ldi     r16, 0xAA
ldi     r17, 0b00001111

eor     r16, r17

Result: R16 = A5h
```

Two's complement
(positive ↔ negative number conversion)

```
ldi     r16, 0xAA
neg     r16

Result: R16 = 56h
```

Task: exchange two lower bits with two upper bits




```
ldi r16, 0x7B

… write instructions here …
```