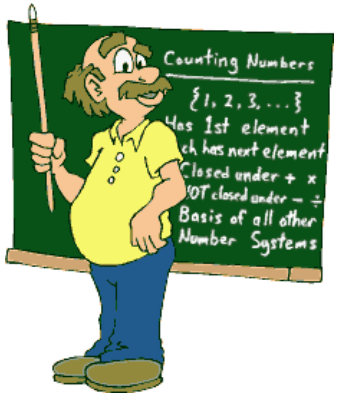
iha.dk

IECA

Embedded Computer Architecture

Lesson 9
Arithmetic and Logical Instructions



Version: 6-11-2015 Henning Hargaard

Binary addition

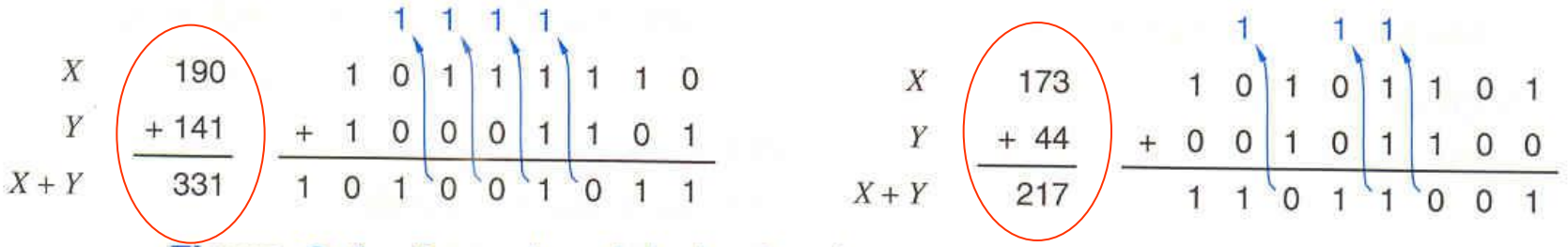


Figure 2–1 Examples of decimal and corresponding binary additions.

Binary subtraction

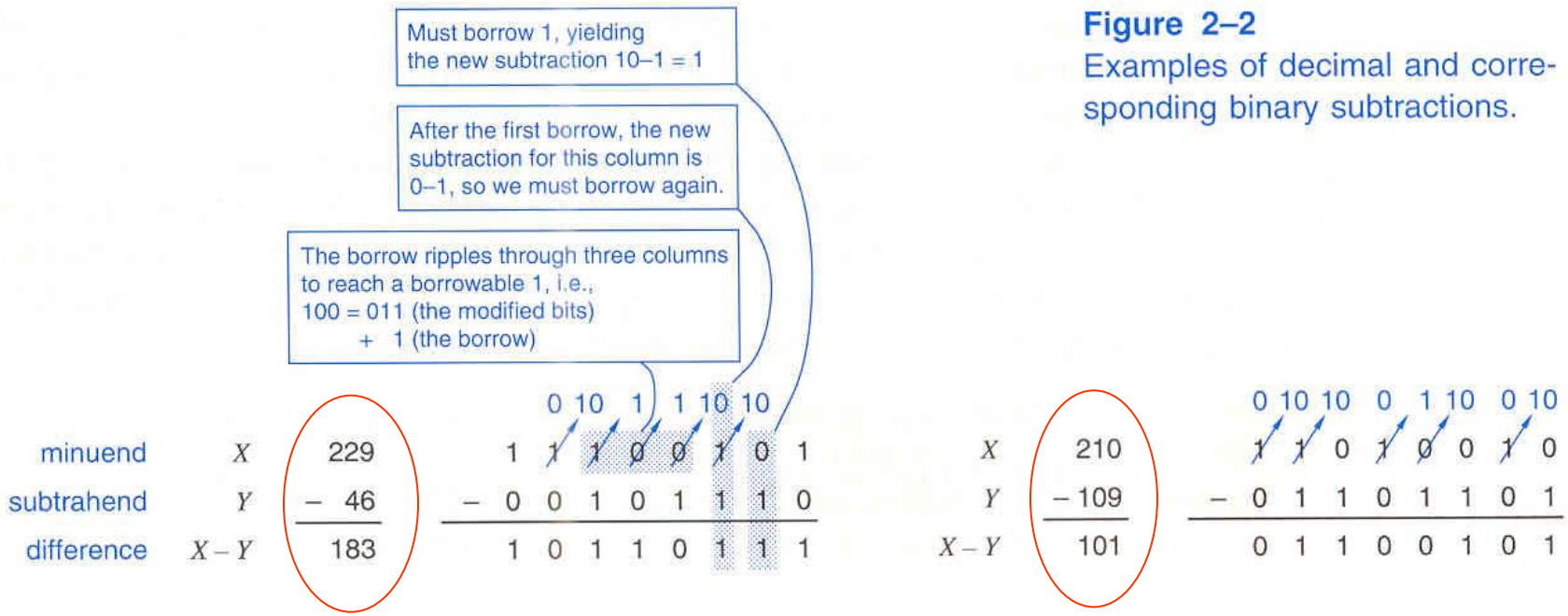
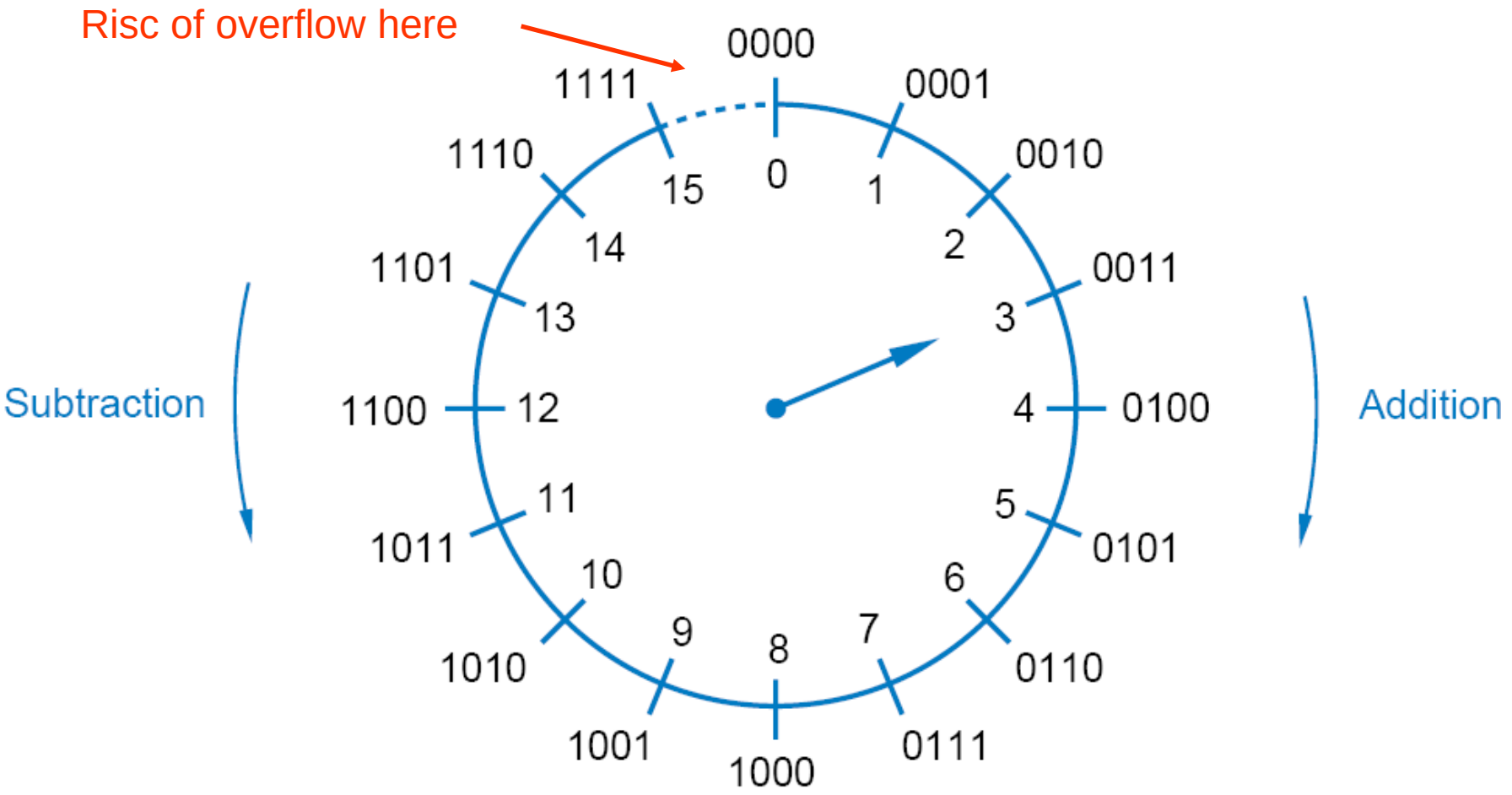


Figure 2–2
Examples of decimal and corresponding binary subtractions.

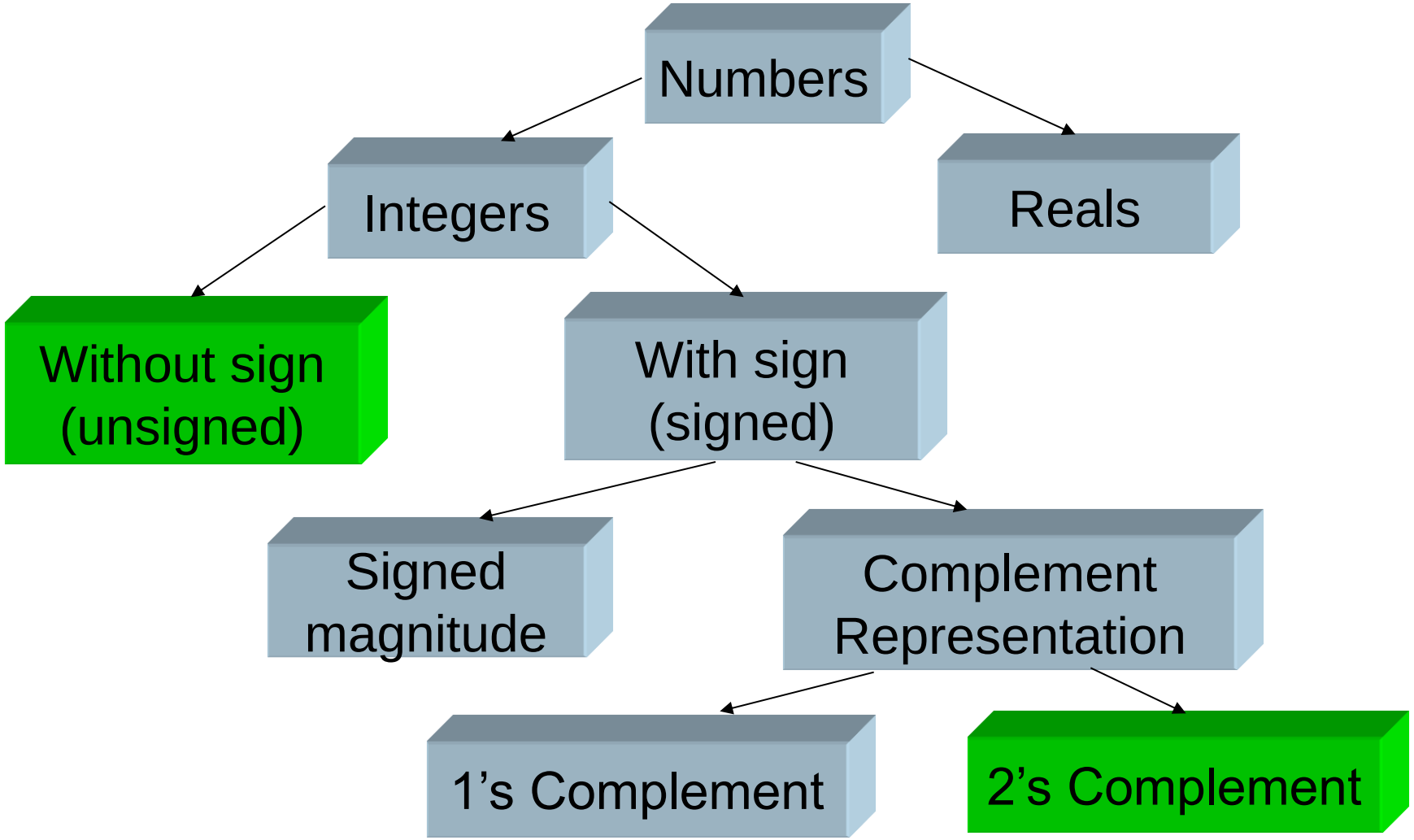
Unsigned addition / subtraktion



Easy having 16 fingers :

C	1 1 0 0		1	1	0	0
X	$1\ 9\ B\ 9_{16}$		1	9	11	9
Y	$+\ C\ 7\ E\ 6_{16}$	+	12	7	14	6
$X+Y$	$E\ 1\ 9\ F_{16}$		14	17	25	15
			14	16+1	16+9	15
			E	1	9	F

Representing numbers



Signed magnitude

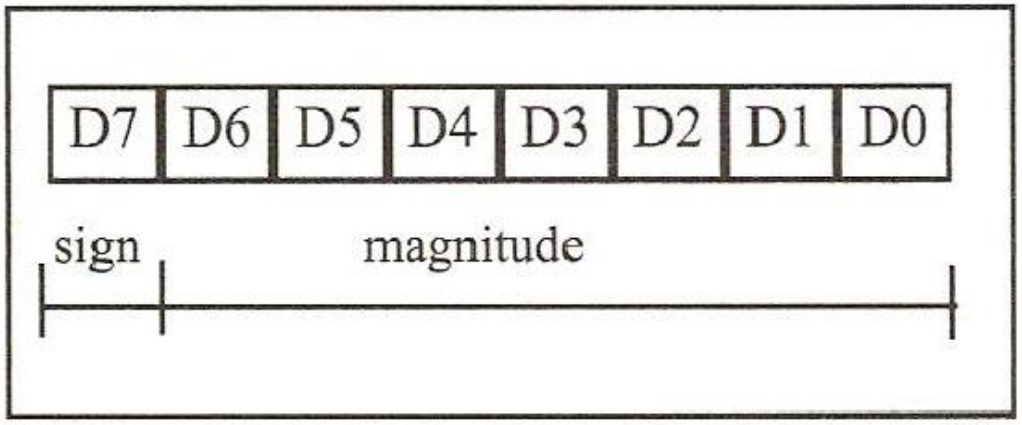
- Typically we use the MSB to represent the sign.

$01010101_2 = +85_{10}$ $\quad$ $11010101_2 = -85_{10}$

$01111111_2 = +127_{10}$ $\quad$ $11111111_2 = -127_{10}$

$00000000_2 = +0_{10}$ $\quad$ $10000000_2 = -0_{10}$

- Notice: We have 2 values representing 0 !



Forming the 2's complement

$17_{10} = 00010001_2$

$\Downarrow$ complement bits

11101110

$+1$

$11101111_2 = -17_{10}$

$-99_{10} = 10011101_2$

$\Downarrow$ complement bits

01100010

$+1$

$01100011_2 = 99_{10}$

$119_{10} = 01110111_2$

$\Downarrow$ complement bits

10001000

$+1$

$10001001_2 = -119_{10}$

$-127_{10} = 10000001_2$

$\Downarrow$ complement bits

01111110

$+1$

$01111111_2 = 127_{10}$

$0_{10} = 00000000_2$

$\Downarrow$ complement bits

11111111

$+1$

$1\,00000000_2 = 0_{10}$

$-128_{10} = 10000000_2$

$\Downarrow$ complement bits

01111111

$+1$

$10000000_2 = -128_{10}$

- Rule : Invert all bits and the add 1.

Forming the 1's complement

$17_{10} = 00010001_2$

$\Downarrow$

$11101110_2 = -17_{10}$

$-99_{10} = 10011100_2$

$\Downarrow$

$01100011_2 = 99_{10}$

$119_{10} = 01110111_2$

$\Downarrow$

$10001000_2 = -119_{10}$

$-127_{10} = 10000000_2$

$\Downarrow$

$01111111_2 = 127_{10}$

$0_{10} = 00000000_2$ (positive zero)

$\Downarrow$

$11111111_2 = 0_{10}$ (negative zero)

- Notice: 2 values for 0 !

Number systems

Table 2-6 Decimal and 4-bit numbers.

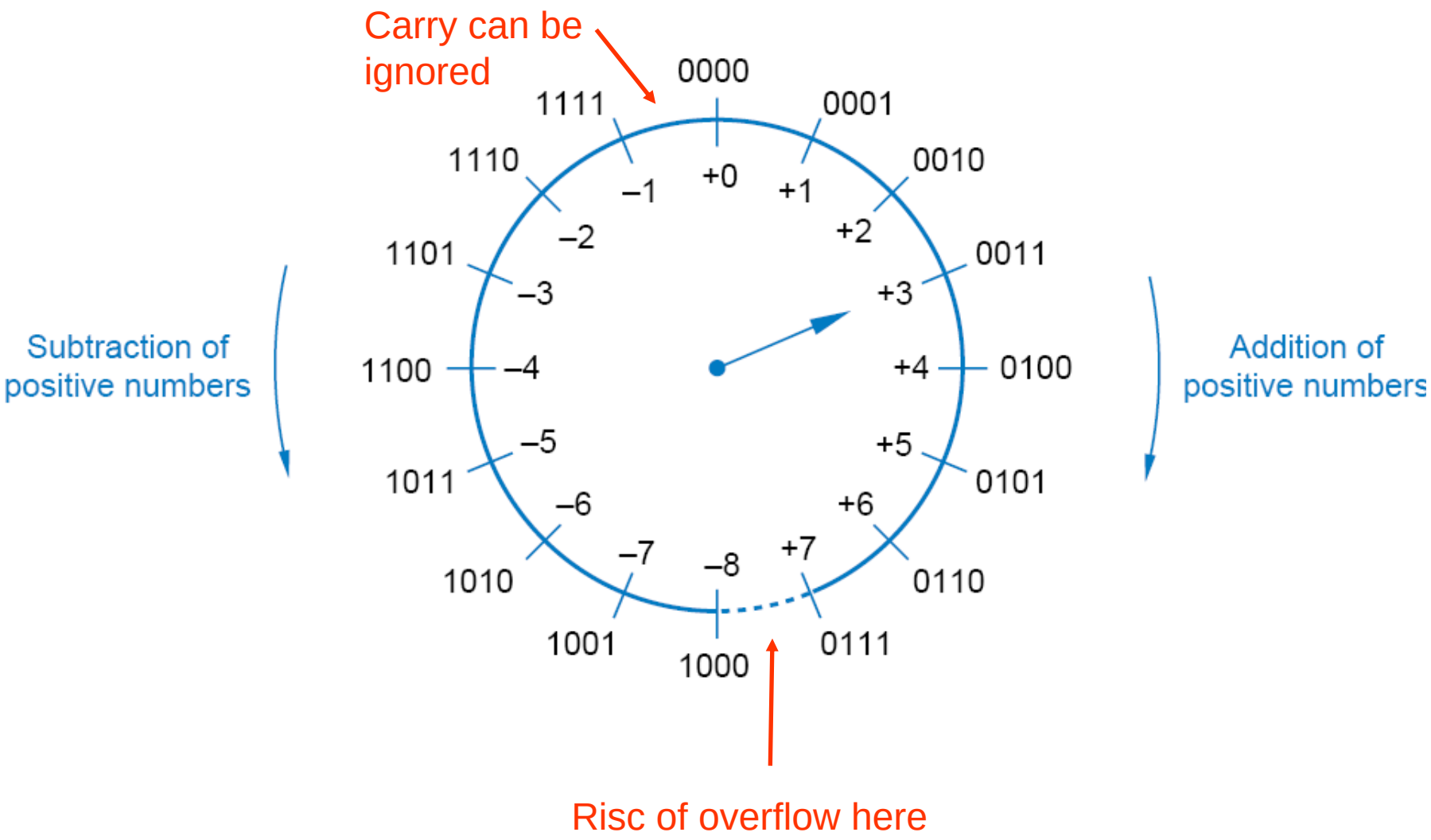
Decimal	Two's Complement	Ones' Complement	Signed Magnitude	Excess 2^{m-1}
–8	1000	—	—	0000
–7	1001	1000	1111	0001
–6	1010	1001	1110	0010
–5	1011	1010	1101	0011
–4	1100	1011	1100	0100
–3	1101	1100	1011	0101
–2	1110	1101	1010	0110
–1	1111	1110	1001	0111
0	0000	1111 or 0000	1000 or 0000	1000
1	0001	0001	0001	1001
2	0010	0010	0010	1010
3	0011	0011	0011	1011
4	0100	0100	0100	1100
5	0101	0101	0101	1101
6	0110	0110	0110	1110
7	0111	0111	0111	1111

2's complement addition

+3	0011	−2	1110
+ +4	+ 0100	+ −6	+ 1010
+7	0111	−8	1 1000
+6	0110	+4	0100
+ −3	+ 1101	+ −7	+ 1001
+3	1 0011	−3	1101

- Note: The same hardware / instructions as for unsigned addition. It's just our way of coding numbers that differs !

The ”2’s complement – watch”



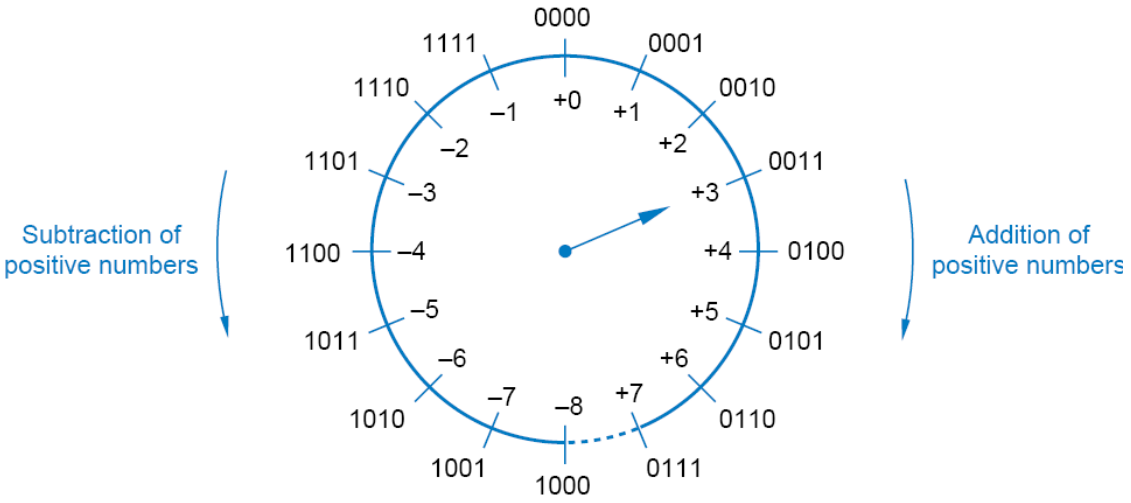
2's complement overflow

$$\begin{array}{rr} -3 & 1101 \\ +\ -6 & +\ 1010 \\ \hline -9 & 10111 = +7 \end{array}$$

$$\begin{array}{rr} +5 & 0101 \\ +\ +6 & +\ 0110 \\ \hline +11 & 1011 = -5 \end{array}$$

$$\begin{array}{rr} -8 & 1000 \\ +\ -8 & +\ 1000 \\ \hline -16 & 10000 = +0 \end{array}$$

$$\begin{array}{rr} +7 & 0111 \\ +\ +7 & +\ 0111 \\ \hline +14 & 1110 = -2 \end{array}$$



Addition Instructions

```
ADD   Rd,Rr          ;Rd = Rd + Rr

ADC   Rd,Rr          ;Rd = Rd + Rr + C

ADIW  Rd:Rd,K        ;Rd+1:Rd = Rd+1:Rd + K
```

Multibyte addition

Example 5-3

Write a program to add two 16-bit numbers. The numbers are 3CE7H and 3B8DH. Assume that R1 = 8D, R2 = 3B, R3 = E7, and R4 = 3C. Place the sum in R3 and R4; R3 should have the lower byte.

Solution:

```
;R1 = 8D
;R2 = 3B
;R3 = E7
;R4 = 3C

ADD   R3,R1        ;R3 = R3 + R1 = E7 + 8D = 74 and C = 1
ADC   R4,R2        ;R4 = R4 + R2 + carry, adding the upper byte
                   ;with carry from lower byte
                   ;R4 = 3C + 3B + 1 = 78H (all in hex)
```

Notice the use of ADD for the lower byte and ADC for the higher byte.

Subtraction Instructions

```
SUB   Rd,Rr          ;Rd = Rd – Rr
SBC   Rd,Rr          ;Rd = Rd - Rr - C
SUBI  Rd,K           ;Rd = Rd - K
SBCI  Rd,K           ;Rd = Rd - K – C

SBIW  Rd:Rd,K        ;Rd+1:Rd = Rd+1:Rd - K
```

Our microcontroller uses this method of subtraction (it “adds to subtract”).

1. Forms the 2’s complement for the number to be subtracted.
2. Adds the two numbers.
3. Inverters the Carry flag.

Multiplication

Table 5-1: Multiplication Summary

Multiplication	Application	Byte1	Byte2	High byte of result	Low byte of result
MUL Rd, Rr	Unsigned numbers	Rd	Rr	R1	R0
MULS Rd, Rr	Signed numbers	Rd	Rr	R1	R0
MULSU Rd, Rr	Unsigned numbers with signed numbers	Rd	Rr	R1	R0

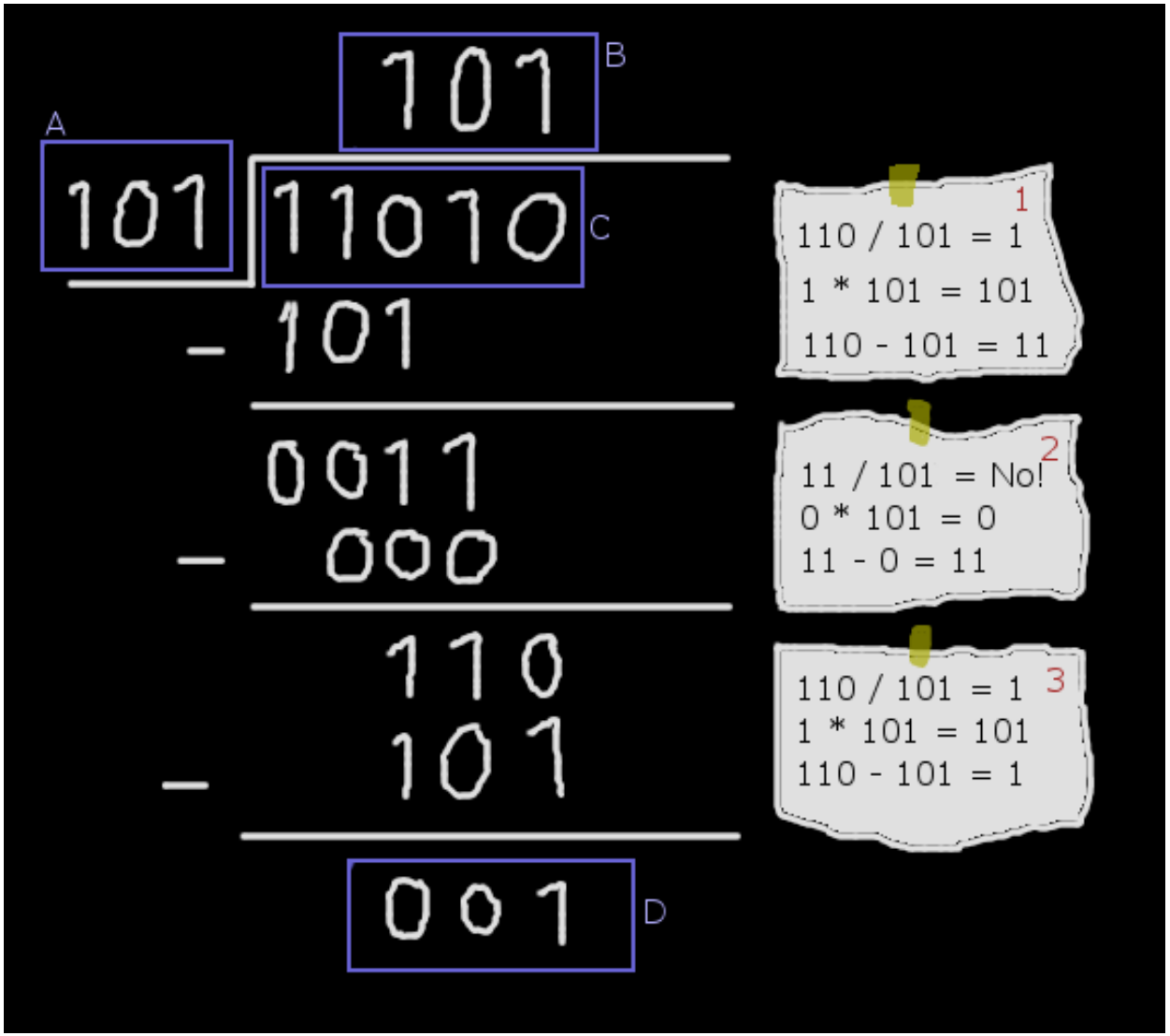
The following example multiplies 25H by 65H.

```
LDI    R23,0x25     ;load 25H to R23
LDI    R24,0x65     ;load 65H to R24
MUL    R23,R24      ;25H * 65H = E99 where
                    ;R1 = 0EH and R0 = 99H
```

Example of an algorithm for division

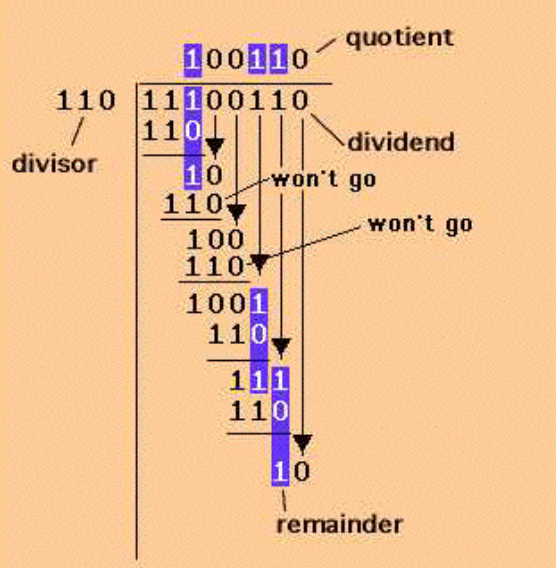
```
.DEF   NUM = R20
.DEF   DENOMINATOR = R21
.DEF   QUOTIENT = R22

       LDI    NUM,95              ;NUM = 95
       LDI    DENOMINATOR,10      ;DENOMINATOR = 10
       CLR    QUOTIENT            ;QUOTIENT = 0

L1:    INC    QUOTIENT
       SUB    NUM, DENOMINATOR
       BRCC   L1                  ;branch if C is zero

       DEC    QUOTIENT            ;once too many
       ADD    NUM, DENOMINATOR    ;add back to it

HERE:  JMP HERE                   ;stay here forever
```

Binary division using shift / subtract



Binary division using shift / subtract

- **Set** quotient to 0
- Align leftmost digits in dividend and divisor
- **Repeat**
  - **If** that portion of the dividend above the divisor is greater than or equal to the divisor
    - **Then** subtract divisor from that portion of the dividend and
    - Concatentate 1 to the right hand end of the quotient
    - **Else** concatentate 0 to the right hand end of the quotient
  - Shift the divisor one place right
- **Until** dividend is less than the divisor
- quotient is correct, dividend is remainder
- **STOP**



How (-5) is stored

Example 5-10

Show how the AVR would represent –5.

Solution:

Observe the following steps.

```
1.     0000 0101          5 in 8-bit binary
2.     1111 1010          invert each bit
3      1111 1011          add 1 (which becomes FB in hex)
```

Therefore, –5 = FBH, the signed number representation in 2's complement for –5. The D7 = N = 1 indicates that the number is negative.

How (-0x34) is stored

Example 5-11

Show how the AVR would represent –34H.

Solution:

Observe the following steps.

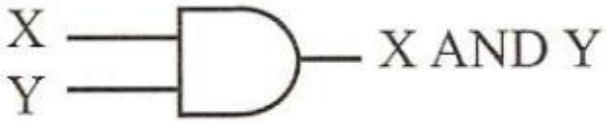
```
1.    0011 0100          34H given in binary
2.    1100 1011          invert each bit
3     1100 1100          add 1 (which is CC in hex)
```

Therefore, -34 = CCH, the signed number representation in 2's complement for 34H. The D7 = N = 1 indicates that the number is negative.

AND Rd,Rr (Rd = Rd AND Rr)

Logical AND Function

Inputs		Output
X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1



Suitable for RESETING certain bit(s).

Also possible to AND with a constant:

ANDI Rd,K ;rd = Rd AND K

AND example

Example 5-18

Show the results of the following.

```
LDI    R20,0x35     ;R20 = 35H
ANDI   R20,0x0F     ;R20 = R20 AND 0FH (now R20 = 05)
```

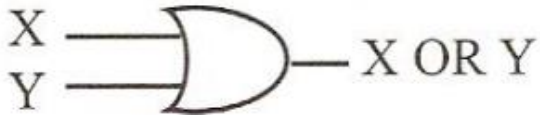
Solution:

```
       35H   0011 0101
 AND   0FH   0000 1111
       ---------------
       05H   0000 0101          ;35H AND 0FH = 05H, Z = 0, N = 0
```

OR Rd,Rr (Rd = Rd OR Rr)

Logical OR Function

Inputs		Output
X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1



Suitable for SETTING certain bit(s).

Also possible to OR with a constant:

ORI Rd,K ;rd = Rd OR K

OR example

Example 5-19

(a) Show the results of the following:

```
      LDI    R20,0x04                ;R20 = 04
      ORI    R20,0x30                ;now R20 = 34H
```

(b) Assume that PB2 is used to control an outdoor light, and PB5 to control a light inside a building. Show how to turn "on" the outdoor light and turn "off" the inside one.

Solution:

```
(a)           04H      0000 0100
      OR      30H      0011 0000
              -------------------
              34H      0011 0100      04 OR 30 = 34H, Z = 0 and N = 0
(b)
      SBI     DDRB, 2              ;bit 2 of Port B is output
      SBI     DDRB, 5              ;bit 5 of Port B is output
      IN      R20, PORTB           ;move PORTB to R20.(Notice that we read
                                   ;the value of PORTB instead of PINB
                                   ;because we want to know the last value
                                   ;of PORTB, not the value of the AVR
                                   ;chip pins.)
      ORI     R20, 0b00000100      ;set bit 2 of R20 to one
      ANDI    R20, 0b11011111      ;clear bit 5 of R20 to zero
      OUT     PORTB, R20           ;out R20 to PORTB

HERE: JMP HERE                     ;stop here
```

EOR Rd,Rr (Rd = Rd XOR Rr)

Logical XOR Function

Inputs		Output
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Suitable for INVERTING certain bit(s).

EOR example

Example 5-20

Show the results of the following:

```
LDI   R20, 0x54
LDI   R21, 0x78
EOR   R20, R21
```

Solution:

```
      54H   0101 0100
XOR   78H   0111 1000
      ----------------
      2CH   0010 1100        54H XOR 78H = 2CH, Z = 0, N = 0
```

COM and NEG

COM (complement)

This instruction complements the contents of a register. The complement action changes the 0s to 1s, and the 1s to 0s. This is also called *1's complement*.

```
LDI    R20,0xAA      ;R20 = 0xAA
COM    R20           ;now R20 = 55H
```

Logical Inverter

Input	Output
X	NOT X
0	1
1	0

NEG (negate)

This instruction takes the 2's complement of a register. See Example 5-23.

Example 5-23

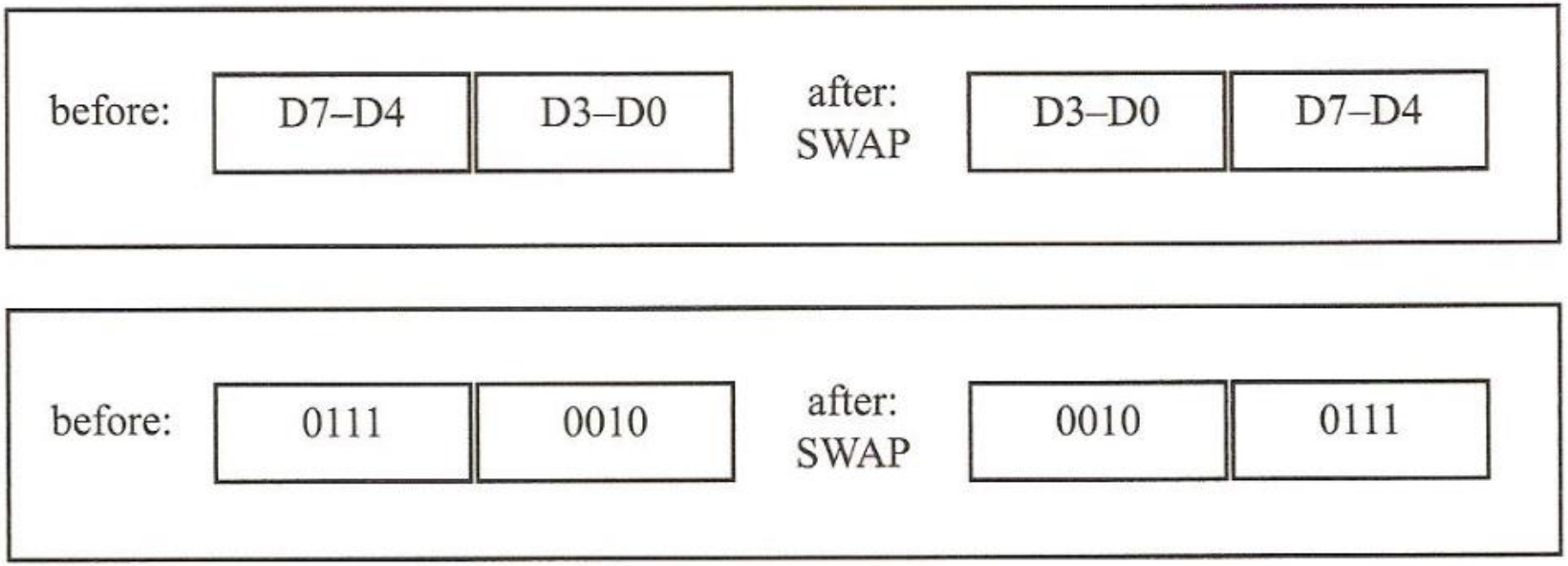
Find the 2's complement of the value 85H. Notice that 85H is –123.

Solution:

```
LDI    R21, 0x85              ;85H = 1000 0101
                              ;1's = 0111 1010
                                     + 1
NEG    R21            ;2's comp      0111 1011 = 7BH
```

SWAP Register

Example: SWAP R16



Compare

```
CP    Rd,Rr     ;Rd is compared to Rr

CPI   Rd,K      ;Rd is compared to K
```

Table 5-2: AVR Compare Instructions

BREQ	Branch if equal	Branch if Z = 1
BRNE	Branch if not equal	Branch if Z = 0
BRSH	Branch if same or higher	Branch if C = 0
BRLO	Branch if lower	Branch if C = 1
BRLT	Branch if less than (signed)	Branch if S = 1
BRGE	Branch if greater than or equal (signed)	Branch if S = 0
BRVS	Branch if Overflow flag set	Branch if V = 1
BRVC	Branch if Overflow flag clear	Branch if V = 0

Conditional Branch and the flags

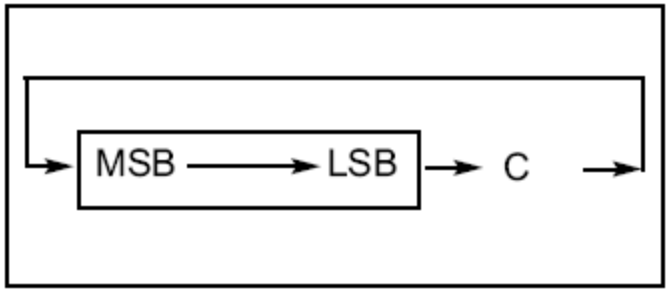
Test	Boolean	Mnemonic	Complementary	Boolean	Mnemonic	Comment
Rd > Rr	Z•(N ⊕ V) = 0	BRLT(1)	Rd ≤ Rr	Z+(N ⊕ V) = 1	BRGE*	Signed
Rd ≥ Rr	(N ⊕ V) = 0	BRGE	Rd < Rr	(N ⊕ V) = 1	BRLT	Signed
Rd = Rr	Z = 1	BREQ	Rd ≠ Rr	Z = 0	BRNE	Signed
Rd ≤ Rr	Z+(N ⊕ V) = 1	BRGE(1)	Rd > Rr	Z•(N ⊕ V) = 0	BRLT*	Signed
Rd < Rr	(N ⊕ V) = 1	BRLT	Rd ≥ Rr	(N ⊕ V) = 0	BRGE	Signed
Rd > Rr	C + Z = 0	BRLO(1)	Rd ≤ Rr	C + Z = 1	BRSH*	Unsigned
Rd ≥ Rr	C = 0	BRSH/BRCC	Rd < Rr	C = 1	BRLO/BRCS	Unsigned
Rd = Rr	Z = 1	BREQ	Rd ≠ Rr	Z = 0	BRNE	Unsigned
Rd ≤ Rr	C + Z = 1	BRSH(1)	Rd > Rr	C + Z = 0	BRLO*	Unsigned
Rd < Rr	C = 1	BRLO/BRCS	Rd ≥ Rr	C = 0	BRSH/BRCC	Unsigned
Carry	C = 1	BRCS	No carry	C = 0	BRCC	Simple
Negative	N = 1	BRMI	Positive	N = 0	BRPL	Simple
Overflow	V = 1	BRVS	No overflow	V = 0	BRVC	Simple
Zero	Z = 1	BREQ	Not zero	Z = 0	BRNE	Simple

Note: 1. Interchange Rd and Rr in the operation before the test, i.e., CP Rd,Rr → CP Rr,Rd

ROR instruction

```
ROR     Rd                  ;Rd  (only flags are set)
```

In ROR, as bits are rotated from left to right, the carry flag enters the MSB and the LSB exits to the carry flag. In other words, **in ROR the C is moved to the MSB, and the LSB is moved to the C.**



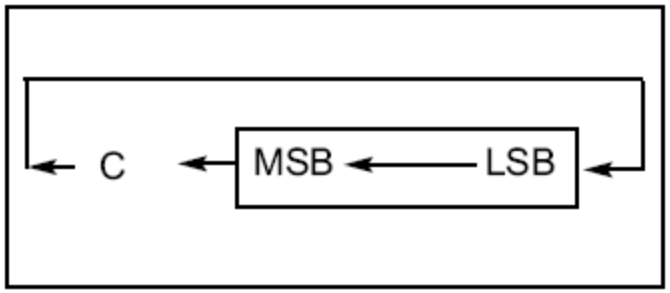
See what happens to 0010 0110 after running 3 ROR instructions:

```
CLC                     ;make C = 0 (carry is 0 )
LDI     R20 , 0x26      ;R20 = 0010 0110
ROR     R20             ;R20 = 0001 0011 C = 0
ROR     R20             ;R20 = 0000 1001 C = 1
ROR     R20             ;R20 = 1000 0100 C = 1
```

ROL instruction

```
ROL     Rd                  ;Rd  (only flags are set)
```

ROL. In ROL, as bits are shifted from right to left, the carry flag enters the LSB and the MSB exits to the carry flag. In other words, **in ROL the C is moved to the LSB, and the MSB is moved to the C.**

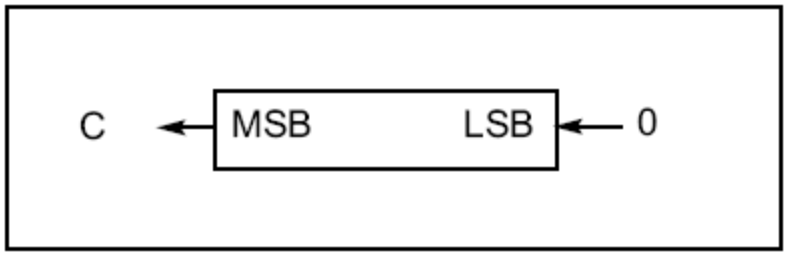



```
SEC                 ;make C = 1 (carry is 0)
LDI R20,0x15         ;R20 = 0001 0101
ROL R20              ;R20 = 0010 1011 C = 0
ROL R20              ;R20 = 0101 0110 C = 0
ROL R20              ;R20 = 1010 1100 C = 0
ROL R20              ;R20 = 0101 1000 C = 1
```

LSL instruction

```
LSL    Rd          ;logical shift left
```

In LSL, as bits are shifted from right to left, 0 enters the LSB and the MSB exits to the carry flag. In other words, **in LSL 0 is moved to the LSB, and the MSB is moved to the C.**



this instruction multiplies content of the register by 2 assuming that after LSL the carry flag is not set.

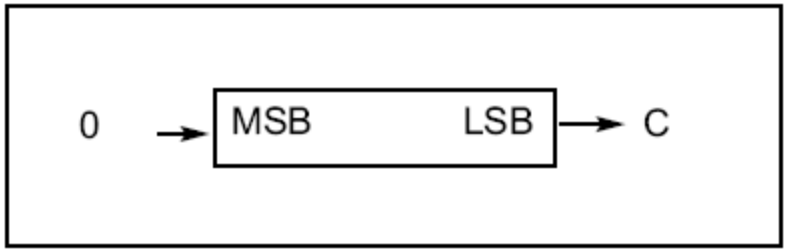
In the next code you can see what happens to 00100110 after running 3 LSL instructions.

```
CLC                 ;make C = 0 (carry is 0 )
LDI R20 , 0x26      ;R20 = 0010 0110(38) c = 0
LSL R20             ;R20 = 0100 1100(74) C = 0
LSL R20             ;R20 = 1001 1000(148) C = 0
LSL R20             ;R20 = 0011 0000(98) C = 1 as C=1 and content of R20
                    ;is not multiplied by 2
```

LSR Instruction

```
LSR     Rd                  ;Rd  (only flags are set)
```

In LSR, as bits are shifted from left to right, 0 enters the MSB and the LSB exits to the carry flag. In other words, **in LSR 0 is moved to the MSB, and the LSB is moved to the C.**



this instruction divides content of the register by 2 and carry flag contains the remainder of division.

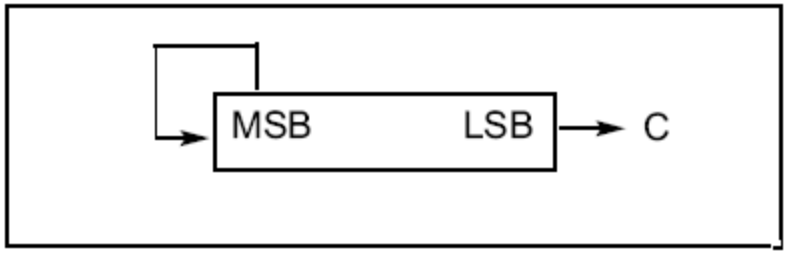
In the next code you can see what happens to 0010 0110 after running 3 LSL instructions.

```
LDI R20,0x26        ;R20 = 0010 0110 (38)
LSR R20             ;R20 = 0001 0011 (19) C = 0
LSR R20              ;R20 = 0000 1001 (9) C = 1
LSR R20              ;R20 = 0000 0100 (4) C = 1
```

ASR Instruction

```
ASR     Rd                  ;Rd  (only flags are set)
```

ASR means *arithmetic shift right*. ASR instruction can divide signed number by 2. In LSR, as bits are shifted from left to right, MSB is held constant and the LSB exits to the carry flag. In other words **MSB is not changed but is copied to D6, D6 is moved to D5, D5 is moved to D4 and so on.**



In the next code you can see what happens to 0010 0110 after running 5 ASL instructions.

```
LDI   R20 , 0D60          ;R20 = 1101 0000(-48) c = 0
ASR   R20                 ;R20 = 1110 1000(-24) C = 0
ASR   R20                 ;R20 = 1111 0100(-12) C = 0
ASR   R20                 ;R20 = 1111 1010(-6) C = 0
ASR   R20                 ;R20 = 1111 1101(-3) C = 0
ASR   R20                 ;R20 = 1111 1110(-1) C = 1
```

LAB5, part 1

(bit 31)	**Number 1:**		**LSB** (bit 0)
R19	**R18**	**R17**	**R16**

MSB (bit 31)	**Number 2:**		**LSB** (bit 0)
R23	**R22**	**R21**	**R20**

MSB (bit 31)	**Sum after addition:**		**LSB** (bit 0)
R19	**R18**	**R17**	**R16**

LAB5, part 2

- If the button SW7 is activated:
  Increment the numerical value of PORTC.
- If the button SW6 is activated:
  Decrement the numerical value of PORTC.
- If the button SW5 is activated:
  The values of [LED7,LED6,LED5,LED4] ”swaps with” [LED3,LED2,LED1,LED0].
- If the button SW4 is activated:
  All LEDs changes state (from ON to OFF and vice verca).
- If the button SW3 is activated:
  The numerical value of PORTC is divided by 8.
- If the button SW2 is activated:
  The numerical value of PORTC is divided by 7. *(Hint: Page 167 in the textbook).*
- If the button SW1 is activated:
  LED7 and LED0 is turned OFF, while the rest of the LEDs must be unchanged.
- If the button SW0 is activated:
  LED7 and LED0 is turned ON, while the rest of the LEDs must be unchanged.

LAB5, part 3 (optional task)

```
;********* LED_ON **********
;* Turns ON one LED at PC  *
;* Bit no.(0-7) is in R20  *
;**************************
LED_ON:
   LDI  R21,1              ;R21 = 0b00000001
   CPI  R20,0
   BREQ READY1             ;Branch if LED no. = 0
AGAIN1:
   LSL  R21                ;Left shift R21
   DEC  R20                ;totally "LED no." times
   BRNE AGAIN1
READY1:
   COM  R21                ;Invert "the mask"
   IN   R20,PINC           ;Read all LEDs
   AND  R20,R21            ;- do bitwise AND
   OUT  PORTC,R20          ;- and output to LEDs again
   RET
;**********************************

;********* LED_OFF **********
;* Tourns OFF one LED at PC *
;* Bit no.(0-7) is in R20   *
;**************************
LED_OFF:

;-----> Write the LED_OFF code here

   RET
;**********************************
```

End of lesson 9