Signed Number Concepts and Arithmetic Operations

Hoda Roodaki

hroodaki@kntu.ac.ir

Signed Number Concepts

- To represent a number as signed, the MSB is set aside for sign.
- The sign is represented by 0 for positive (+) numbers and 1 for negative (-) numbers.
- The steps in representing a negative number (in 2's Complement)
  - Represent the number in 8-bit binary
  - Invert the digits
  - Add 1 with the inverted number

Overflow problem in signed number operations

- The AVR indicates the existence of an error by raising the V (overflow) flag, but it is up to the programmer to care of the erroneous result.
- If the result of an operation on signed numbers is too large for the register, an overflow occurs, V flag is set.

		+96	0110 0000
	+	+70	0100 0110
The limit is +127		+166	1010 0110 (-90) INVALID RESULT
			N=1, V=1

When is the V flag set?

- In 8-bit signed operations, V is set to 1 if either of the following conditions occurs:
  - There is a carry from D6 to D7 but no carry out of D7 (C=0)
  - There is a carry from D7 out (C=1) but no carry from D6 to D7
- In other words, the overflow flag is set to 1 if there is a carry from D6 to D7 or from D7 out but not the both.
- It means that if there are carry both from D6 to D7 and from D7 out, V=0.

Example

	+96		0110 0000
+	+70		0100 0110
	+166		1010 0110

(N=0,C=0,V=1) result is -90 WRONG

	- 128		1000 0000
+	- 2		1111 1110
	- 130	1	0111 1110

(N=0,C=1,V=1) result is +126 WRONG

	- 126		1000 0010
+	- 2		1111 1110
	- 128	1	1000 0000

(N=1, C=1, V=0)

Further Consideration on V flag

- In the ADD instruction there are two different conditions, either the operands have the same sign or the signs of the operands are different.
- When we ADD operands of different signs, the absolute values of the operands are less than absolute values of one of the operands, so overflow can not happen.
- Overflow can only happen when operands of same sign, it is possible that the result overflows the limit of the register
- In AVR, the equation of V flag is as follows:

$$V = Rd7.Rr7.\overline{R7} + \overline{Rd7}.\overline{Rr7}.R7$$

- Where Rd7 and Rr7 are 7^{th} bit of the operands and R7 is the 7^{th} bit of the result.

Difference between N and S flag

- The N flag represents D7 bit of the result.
  - If the result is positive, the N flag is zero and if the result is negative N flag is 1.
- In operations of signed numbers, overflow is possible. Overflow corrupts the result and negates (make opposite) the D7 bit.
  - So if we ADD two positive numbers, in case of overflow, the N flag would be 1 showing that the result is negative!
- The S flag helps you to know the real sign of the actual result.

Example 5-15

Examine the following code, noting the role of the V and N flags:

```
LDI   R20,-2        ;R20 = 1111 1110 (R20 = FEH)
LDI   R21,-5        ;R21 = 1111 1110 (R21 = FBH)
ADD   R20,R21       ;R20 = (-2) + (-5) = -7 or F9H
                    ;correct, since V = 0
```

Solution:

```
  -2     1111 1110
+ -5     1111 1011
- 7      1111 1001   and V = 0 and N = 1. Sum is negative
```

According to the CPU, the result is –7, which is correct, and the V indicates that (V = 0).

Example 5-16

Examine the following code, noting the role of the V and N flags:

```
LDI    R20,7       ;R20 = 0000 0111
LDI    R20,18      ;R20 = 0001 0010
ADD    R20,R21     ;R20 = (+7) + (+18)
                   ;R20 = 00000111 + 00010010 = 0001 1001
                   ;R20 = (+7) + (+18) = +25, N = 0, positive
                   ;and correct, V = 0
```

Solution:

```
   + 7 0000 0111
+  +18 0001 0010
   +25 0001 1001   N = 0 (positive 25) and V = 0
```

According to the CPU, this is +25, which is correct, and V = 0 indicates that.

Conditional Jump

- BRSH (branch if same or higher) and BRLO (branch if lower) instructions
  – to compare unsigned numbers.

Example 5-24

Write a program to monitor PORTB continuously for the value 63H. It should stop monitoring only if PORTB = 63H.

Solution:

```
        LDI     R20,0x00
        OUT     DDRB,R20        ;PORT B is input
        LDI     R21,0x63
AGAIN:
        IN      R20,PINB
        CP      R20,R21         ;compare with 0x63, Z = 1 if yes
        BRNE    AGAIN           ;go to AGAIN if PORTB is not equal to 0x63
        ....
```

Example 5-25

Write a program to find the greater of the two values 27 and 54, and place it in R20.

Solution:

```
.EQU  VAL_1=27
.EQU  VAL_2=54

      LDI   R20,VAL_1   ;R20 = VAL_1
      LDI   R21,VAL_2   ;R21 = VAL_2
      CP    R21,R20     ;compare R21 and R20
      BRLO  NEXT        ;if R21<R20 (branch if lower) go to NEXT
      LDI   R20,VAL_2   ;R20 = VAL_2
NEXT:
```

Example 5-26

Assume that Port B is an input port connected to a temperature sensor. Write a program to read the temperature and test it for the value 75. According to the test results, place the temperature value into the registers indicated by the following.

If T = 75	then R16 = T	; R17 = 0 ; R18 = 0
If T > 75	then R16 = 0	; R17 = T ; R18 = 0
If T < 75	then R16 = 0	; R17 = 0 ; R18 = T

Solution:

```
            LDI   R20,0x00          ;R20 = 0
            OUT   DDRB,R20          ;Port B = input

            CLR   R16               ;R16 = 0
            CLR   R17               ;R17 = 0
            CLR   R18               ;R18 = 0

            IN    R20,PINB
            CPI   R20,75            ;compare R20 (PORTB) and 75
            BRSH  SAME_HI
                                    ;executes when R20 < 75
            MOV   R18,R20
            RJMP  CNTNU
SAME_HI:                            ;executes when R20 >= 75

            BRNE  HI

            MOV   R16,R20           ;executes when R20 = 75
            RJMP  CNTNU
HI:                                 ;executes when R20 > 75
            MOV   R17,R20
CNTNU:      ....
```

BRGE and BRLT instructions

- The BRGE makes decisions based on the S flag.
  - If S = 0 (which, after the CP instruction for signed numbers, means that the left-hand operand of the CP instruction was greater than or equal to the right-hand operand) the BRGE instruction branches in a forward or backward direction relative to program counter.
- The BRLT is like the BRGE, but it branches when S = 1.
- Notice that the BRGE, and the BRLT are used with signed numbers.

BRVS and BRVC instructions

- The BRVC and BRVS instructions :
  - let you check the value of the V flag and change the flow of the program if overflow has occurred.

Example 5-27

Write a program to add two signed numbers. The numbers are in R21 and R22. The program should store the result in R21. If the result is not correct, the program should put 0xAA on PORTA and clear R21.

Solution:

```
        LDI    R21,0xFA         ;R21 = 0xFA
        LDI    R22,0x05         ;R22 = 0x05
        LDI    R23,0xFF         ;R23 = 0xFF
        OUT    DDRA,R23         ;Port A is output
        ADD    R21,R22          ;R21 = R21 + R22
        BRVC   NEXT             ;if V = 0 ( no error) then go to next
        LDI    R23,0xAA         ;R23 = 0xAA
        OUT    PORTA,R23        ;send 0xAA to PORTA
        LDI    R21,0x00         ;clear R21
NEXT: ...
```

BRPL and BRMI instruction

- BRPL and BRMI for checking N flag.
- BRPL (branch if plus) really means ``branch only if the N flag is cleared''.
- BRMI (branch if minus) really means ``branch only if the N flag is set.

Logic operation and application

Logic and Compare Instruction

AND

AND Rd, Rr ; Rd = Rd AND Rr

ANDI Rd, K ; Rd = Rd AND K

- The AND instructions can affect Z, S and N flags.
- The AND instructions are often used to mask certain bits of an operand.

LDI R20, 0x35	35H = 0011 0101
ANDI R20, 0x0F	0FH = <u>0000 1111</u>
	0000 0101
	[masking upper nibble]

Logic and Compare Instruction

<u>OR</u>

OR Rd, Rr ; Rd = Rd OR Rr

ORI Rd, K ; Rd = Rd OR K

- OR instructions affects the Z, S and N flags.
- The OR instructions can be used to set certain bits of an operand to 1, for example

```
LDI R20, 0x04
ORI R20, 0x30        ; now R20 will be 34H
```

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
```

Logic and Compare Instruction

EOR

EOR Rd, Rs ; Rd = Rd XOR Rs

- The EX-OR will affect the Z, S and N flags
- EX-OR can be used to check whether the values in two registers are equal or not

```
OVER :    IN R20, PORTB
          LDI R21, 0x45
          EOR R20, R21
```

- Another widely used application of EX-OR is to toggle the bits of an operand.
  - EOR R0, R20 ;and R20 is initialized with 0xFF

Example 5-22

Read and test PORTB to see whether it has the value 45H. If it does, send 99H to PORTC; otherwise, it is cleared.

Solution:

```
        LDI     R20,0xFF        ;R20 = 0xFF
        OUT     DDRC,R20        ;Port C is output
        LDI     R20,0x00        ;R20 = 0
        OUT     DDRB,R20        ;Port B is input
        OUT     PORTC,R20       ;PORTC = 00
        LDI     R21,0x45        ;R21 = 45
HERE:
        IN      R20,PINB        ;get a byte
        EOR     R20,R21         ;EX-OR with 0x45

        BRNE    HERE            ;branch if PORTB has value other than 45
        LDI     R20,0x99        ;R20 = 0x99
        OUT     PORTC,R20       ;PORTC = 99h
EXIT:   JMP     EXIT            ;stop here
```

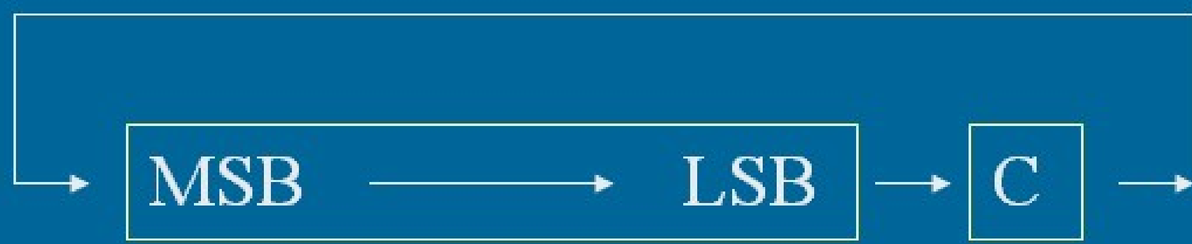
Few Other Logical Instructions

- COM (Complement) – COM R20 ; 1's complement
- NEG (Negative) – NEG R20 ; 2's complement
- CP (Compare) – CP Rd, Rr ;
- Compare is same as subtraction except that compare does not change the register
- There is also compare with constant value

CPI Rd, K

ROTATE AND SHIFT INSTRUCTIONS AND DATA SERIALIZATION

ROR Rd ; Rotate Rd right through carry



ROL Rd ; Rotate Rd left through carry

Example

```
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
```

```
SEC                      ;make C = 1
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
```

Serializing data

- Using the serial port.
- The second method of serializing data is to transfer data one bit at a time and control the sequence of data and spaces between them.
- In many new generations of devices, the serial versions are becoming popular because they take up less space on a printed circuit board.

Serializing a byte of data

- Serializing data is one of the most widely used applications of the rotate instruction.
- We can use the rotate instruction to transfer a byte of data serially (one bit at a time).
  - Shift instructions can be used for the same job.

Example 5-28

Write a program to transfer the value 41H serially (one bit at a time) via pin PB1. Pu one high at the start and end of the data. Send the LSB first.

Solution:

```
.INCLUDE "M32DEF.INC"

      SBI    DDRB, 1          ;bit 1 of Port B is output
      LDI    R20,0x41         ;R20 = the value to be sent

      CLC                     ;clear carry flag
      LDI    R16, 8           ;R16 = 8
      SBI    PORTB, 1         ;bit 1 of PORTB is 1

AGAIN:
      ROR    R20              ;rotate right R20 (send LSB to C flag)
      BRCS   ONE              ;if C = 1 then go to ONE
      CBI    PORTB, 1         ;bit 1 of PORTB is cleared to zero
      JMP    NEXT             ;go to NEXT
ONE:  SBI    PORTB, 1         ;bit 1 of PORTB is set to one
NEXT:
      DEC    R16              ;decrement R16
      BRNE   AGAIN            ;if R16 is not zero then go to AGAIN
      SBI    PORTB, 1         ;bit 1 of PORTB is set to one

HERE: JMP    HERE             ;RB1 = high
```

Example 5-29

Write a program to bring in a byte of data serially via pin RC7 and save it in R20 register. The byte comes in with the LSB first.

Solution:

```
.INCLUDE    "M32DEF.INC"

      CBI   DDRC, 7     ;bit 7 of Port C is input
      LDI   R16, 8      ;R16 = 8
      LDI   R20, 0      ;R20 = 0
AGAIN:
      SBIC  PINC, 7     ;skip the next line if bit 7 of Port C is 0
      SEC               ;set carry flag to one
      SBIS  PINC, 7     ;skip the next line if bit 7 of Port C is 1
      CLC               ;clear carry flag to zero
      ROR   R20         ;rotate right R20. move C flag to MSB of R21
      DEC   R16         ;decrement R16
      BRNE  AGAIN       ;if R16 is not zero go to AGAIN

HERE: JMP   HERE        ;stop here
```

Example 5-30

Write a program that finds the number of 1s in a given byte.

Solution:

```
.INCLUDE    "M32DEF.INC"

      LDI   R20, 0x97
      LDI   R30, 0        ;number of 1s
      LDI   R16, 8        ;number of bits in a byte

AGAIN:
      ROR   R20           ;rotate right R20 and move LSB to C flag
      BRCC  NEXT          ;if C = 0 then go to NEXT
      INC   R30           ;increment R30
NEXT:
      DEC   R16           ;decrement R16
      BRNE  AGAIN         ;if R16 is not zero then go to AGAIN

      ROR   R20           ;one more time to leave R20 unchanged

HERE: JMP   HERE          ;stop here
```

Example 5-32

Assume that R20 has the number 48. Show how we can use ROR to divide R20 by 8.

Solution:

```
                        ;to divide a number by 8 we can
                        ;shift it 3 bits to the right. without
                        ;LSR we have to ROR 3 times and
                        ;clear carry flag before
                        ;each rotation

    LDI     R20,0x30    ;R20 = 0011 0000 (48)
    CLC                 ;clear carry flag
    ROR     R20         ;R20 = 0001 1000 (24)
    CLC                 ;clear carry flag
    ROR     R20         ;R20 = 0000 1100 (12)
    CLC                 ;clear carry flag
    ROR     R20         ;R20 = 0000 0110 (6)
                        ;48 divided by 8 is 6 and
                        ;the answer is correct
```

Example 5-33

(a) Find the contents of the R20 register in the following code .

```
LDI   R20, 0x72
SWAP  R20
```

(b) In the absence of a SWAP instruction, how would you exchange the nibbles? Write a simple program to show the process.

Solution:

(a)

```
LDI   R20, 0x72          ;R20 = 0x72
SWAP  R20                ;R20 = 0x27
```

(b)

```
        LDI   R20,0x72
        LDI   R16,4
        LDI   R21,0
BEGIN:
        CLC
        ROL   R20
        ROL   R21
        DEC   R16
        BRNE  BEGIN
        OR    R20, R21
HERE:   JMP   HERE
```

BCD AND ASCII CONVERSION

BCD (binary coded decimal) number system

- BCD stands for *binary coded decimal.*
- BCD is needed because in everyday life we use the digits 0 to 9 for numbers, not binary or hex numbers.
- Binary representation of 0 to 9 is called BCD.
- Two terms for BCD numbers:
  - unpacked BCD
  - packed BCD

BCD AND ASCII CONVERSION

- <u>Unpacked BCD</u>
  - In unpacked BCD, the lower 4 bits of the number represent the BCD number, and the rest of the bits are 0. For example, "0000 1001" and "0000 0101" are unpacked BCD for 9 and 5, respectively.
  - Unpacked BCD requires 1 byte of memory, or an 8-bit register, to contain it.

BCD AND ASCII CONVERSION

- Packed BCD
  - In packed BCD, a single byte has two BCD numbers in it: one in the lower 4 bits, and one in the upper 4 bits. For example, “0101 1001" is packed BCD for 59H.
  - Only 1 byte of memory is needed to store the packed BCD operands.
  - It is twice as efficient in storing data.

BCD AND ASCII CONVERSION

- On ASCII keyboards, when the key “0" is activated, "0011 0000" (30H) is provided to the computer. Similarly, 31H (0011 0001) is provided for key “1", and so on.
- BCD numbers are universal, although ASCII is standard in the United States (and many other countries).
- Because the keyboard, printers, and monitors all use ASCII, how does data get converted from ASCII to BCD and vice versa?

Packed BCD to ASCII conversion

- In many systems we have what is called a *real-time clock* (RTC).
  - The RTC provides the time of day (hour, minute, second) and the date (year, month, day) continuously, regardless of whether the power is on or off.
  - This data, however, is provided in packed BCD.
  - For this data to be displayed on a device such as an LCD, or to be printed by the printer, it must be in ASCII format

Packed BCD	Unpacked BCD	ASCII
29H	02H & 09H	32H & 39H
0010 1001	0000 0010 &	0011 0010 &
	0000 1001	0011 1001

Example 5-34

Assume that R20 has packed BCD. Write a program to convert the packed BCD to two ASCII numbers and place them in R21 and R22.

Solution:

```
.INCLUDE    "M32DEF.INC"

      LDI    R20,0x29     ;the packed BCD to be converted is 29

      MOV    R21,R20      ;R21 = R20 = 29H
      ANDI   R21,0x0F     ;mask the upper nibble (R21 = 09H)
      ORI    R21,0x30     ;make it ASCII (R21 = 39H)

      MOV    R22,R20      ;R22 = R20 = 29H
      SWAP   R22          ;swap nibbles (R22 = 92H)
      ANDI   R22,0x0F     ;mask the upper nibble (R22 = 02)
      ORI    R22,0x30     ;make it ASCII (R22 = 32H)

HERE: JMP    HERE
```

ASCII to packed BCD

- To convert ASCII to packed BCD, you first convert it to unpacked BCD, and then combine it to make packed BCD.
  - For example, for 4 and 7 the keyboard gives 34 and 37, respectively.
    - The goal is to produce 47H or "0100 0111 ", which is packed BCD.

ASCII to packed BCD

```
LDI   R21,'4'      ;load character 4 to R21
LDI   R22,'7'      ;load character 7 to R22
ANDI  R21,0x0F     ;mask upper nibble of R21
SWAP  R21          ;swap nibbles of R21
                   ;to make upper nibble of packed BCD
ANDI  R22,0x0F     ;mask upper nibble of R22
OR    R22,R21      ;join R22 and R21 to make packed BCD
MOV   R20,R22      ;move the result to R20
```