Integer Arithmetic

Reading: pp. 296-302

Addition

In general, we know the following is true:

```
0 + 0 = 0 0
0 + 1 = 0 1
1 + 0 = 0 1
1 + 1 = 1 0
```

P1: Perform the following addition and interpret the result in: (a) modulo 2^n and (b) two's complement notation.

```
 010010101         10101010
+000111001        +10101010
 ---------         --------
```

We must become familiar with the use of the carry. There are a couple of different carries that we must concern ourselves with. The carry-in and carry-out.

P2: Add the following two binary values discussing what is meant by carry-in and carry-out.

```
 00001111
+01010101
 --------
```

Note: The carry-out of the MSB is the value that the external carry flag found in the flags register takes on.

Q1: In the previous example, what would be the value of the external carry? Why?

Half-Adders and Adders

Subtraction

Subtraction works a little differently than one would think. In particular, subtraction is performed by taking the two's complement of the subtrahend and adding this value to the minuend. Let's look at the following example for some clarification.

Perform the following subtraction:

```
 00110011    (Minuend)
-00001111    (Subtahend)
 --------
```

Q2: Before performing the subtraction, identify the two numbers being subtracted. Assume the numbers are represented in modulo 2^n.

P3: Now perform the subtraction.

Q3: Interpret the result. Is it what you would expect it to be?

P4: Switch the minuend and the subtrahend from the above problems and then perform the subtraction. Interpret the result if the numbers are: (a) modulo 2^n and (b) two's complement numbers.

Arithmetic Overflow

Remember that the range of values that can be represented using 8-bits for modulo 2^n numbers is 0 to 255 and for two's complement is -128 to 127. The microprocessor will perform the addition or subtraction of two numbers, but the question is how do we know if the result is correct. That is, if we add two 8-bit numbers and the result is larger than the representation allows, how do we know this happened. The answer lies with two flags: (a) the carry flag and (b) the overflow flag.

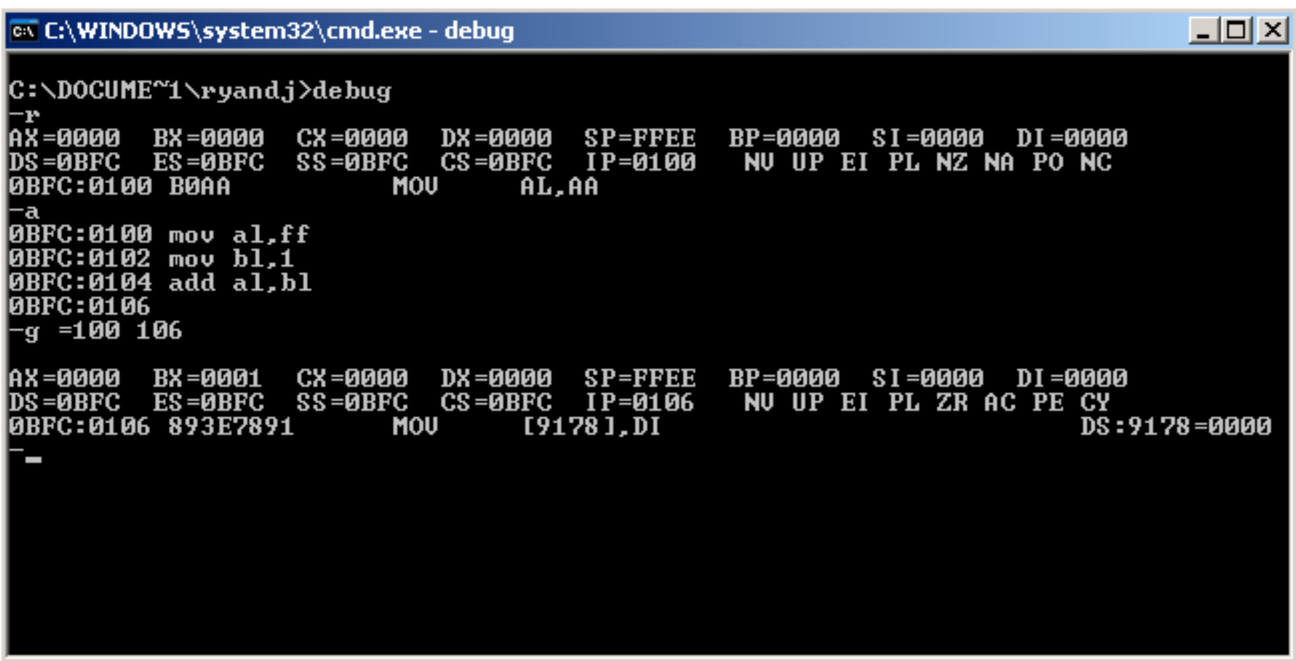
First we will define overflow as a condition such that an arithmetic operation produces a result outside the range of the number system being used.

P5: Perform the operations below and interpret the result in: (a) modulo 2^n and (b) two's complement notation.

```
 11111111      01111111
+00000001     +00000001
 --------      --------
```

Q4: Were there any examples of overflow? Identify each case and briefly explain why.

Test each example using debug in Windows XP.



```
C:\WINDOWS\system32\cmd.exe - debug

C:\DOCUME~1\ryandj>debug
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0BFC  ES=0BFC  SS=0BFC  CS=0BFC  IP=0100   NV UP EI PL NZ NA PO NC
0BFC:0100 B000          MOV     AL,00
-a
0BFC:0100 mov al,7f
0BFC:0102 mov bl,1
0BFC:0104 add al,bl
0BFC:0106
-g =100 106

AX=0080  BX=0001  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0BFC  ES=0BFC  SS=0BFC  CS=0BFC  IP=0106   OV UP EI NG NZ AC PO NC
0BFC:0106 893E7891      MOV     [9178],DI                      DS:9178=0000
-_
```

Let's perform the following additions and determine where any overflows occurred.

```
 1001                    1100
+0101                   +0100
-----                   -----

 0011                    1100
+0100                   +1111
-----                   -----

 0101                    1001
+0100                   +1010
-----                   -----
```

Let's perform the following subtractions and determine where any overflows occurred. Represent the numbers in 2's complement.

```
2 – 7               5 – 2


(-5) – 2            5 – (-2)


7 – (-7)            (-6) - 4
```

For each of the following, perform the additions, convert the binary numbers to decimal where the binary numbers are represented in two's complement, and indicate where a carry-out or overflow occurs.

11011001 +01011100 ---------	11101101 +11111001 ---------	00101100 +00101101 ---------
01101000 +00101101 ---------	10110101 +00111011 ---------	10011001 +10111011 ---------
00001010 +11111101 ---------	01111111 +00000001 ---------	11111111 +00000001 ---------