Topic 4. Arithmetic operations

Binary Addition and Subtraction

- The rules for addition:
  0+0=0
  0+1=1
  1+0=1
  1+1=0 (carry)

Example:

			11
	6		0110
+	3	+	0011
	9		1001

- The rules for subtraction:
  0-0=0
  0-1=1 (1-borrow)
  1-0=1
  1-1=0

Example:

	6		0110
-	3	-	0011
	3		0011

Sign-magnitude system (DC)

Direct code (DC):

x = 3 = 00011

y = -9 = 11001

Find x + y.

Steps:

1. Compare numbers and find the largest one by absolute value.
2. Determine if the signs are the same.
3. Change the arithmetic operation if the signs are different.

```
1001 -
0011
----
0110
```

4. Assign the sign of the largest number to the result. (`10110` = -6)

Addition is not done in DC because:

- We need to have an adder, a subtractor, a comparator and a more complex control unit.
- The sign is processed separately from the module.

One’s complement system (IC)

Advantages:

- Subtraction is substituted by addition.
- The sign is processed with the whole number.

Disadvantages:

- Addition is done in 2 steps because the carry out from the sign position is added to the number.
- 0 has two representations.

x = -1

$x_{DC} = 10001$

$x_{IC} = 11110$

y = 12

$y_{IC} = 01100$

```
11110 +
01100
-----
01011 #carried the one from the leftmost to the rightmost
```

Addition and Subtraction in Two's Complement System

- If we represent the negative number in the two's complement system we can substitute the subtraction with addition: $A-B=A+B_{CC}$

- In a two's complement system representation the sign and the significant are examined together and the result is obtained in two's complement representation.

Example 1:

$A=-27_{10}=-11011_2$ $\quad$ $B=31_{10}=11111_2$

$A_{DC}=1.11011$ $\quad$ $B_{DC}=0.11111$

$A_{CC}=1.00101$ $\quad$ $B_{CC}=0.11111$

A+B

$$
\begin{array}{lcr}
A_{CC}= & & 1.00101 \\
B_{CC}= & + & 0.11111 \\
\hline
C_{CC}= & & 0.00100
\end{array}
$$

$C=4_{10}$

Example 2:

$A = 8_{10} = 01000_2$

$A_{DC} = 0.01000$

$A_{CC} = 0.01000$

$B = -25_{10} = -11001_2$

$B_{DC} = 1.11001$

$B_{CC} = 1.00111$

A+B

$A_{CC}=$		0.01000
$B_{CC}=$	+	1.00111
$C_{CC}=$		1.01111

complementing 110000

+1

110001

$C = 10001_2 = -17_{10}$

Example 3:

$A = -13_{10} = -01101_2$ $B = 17_{10} = 10001_2$

$A_{CC} = 1.10011$ $B_{CC} = 0.10001$

$A\text{-}B = A + (-B)$

$-B_{CC} = 1.01111$

$$
\begin{array}{lccr}
A_{CC} = & + & & 1.10011 \\
-B_{CC} = & & & 1.01111 \\
\hline
C_{CC} = & & \textcircled{1} & 1.00010
\end{array}
$$

$C_{CD} = 1.11110$

$C = 30$

Overflow and Underflow

Example 4:

$A=21_{10}$

$A_{CC}=0.10101$

$B=17_{10}$

$B_{CC}=0.10001$

$A_{CC}=$	+	0.10101	
$B_{CC}=$		0.10001	
$C_{CC}=$		1.00110	positive overflow

Example 5:

$A=-26_{10}=-11010_2$

$A_{CC}=1.00110$

$B=-22_{10}=-10110_2$

$B_{CC}=1.01010$

$A_{CC}=$	+	1.00110	
$B_{CC}=$		1.01010	
$C_{CC}=$		0.00110	negative overflow

An addition overflows the result if the signs of the addends are the same and the sign of the result differs from the sign of the addends.

Binary Multiplication

```
   11          1011   multiplicand
   13          1101   multiplier
 ----        ------
   33          1011
  11          0000    shifted
 ----        1011     multiplicands
  143       1011
           --------
           10001111   product
```

```
 11          1011
 13          1101
----        ------
 11          1011
  33          1011
----           0000
 143             1011
            --------
             10001111
```

Multiplication algorithms

There are four multiplication algorithms:

- **Starting with the LSB of the multiplier shifting the multiplicand left**
- **Starting with the MSB of the multiplier shifting multiplicand right**
- Starting with the LSB of the multiplier shifting partial products right
- Starting with the MSB of the multiplier shifting partial products left

Multiplication in Signed-Magnitude System

Multiplication of signed numbers can be accomplished using unsigned multiplication.

Steps:

1. Use XOR function to determine the product sign $Sg\ X \oplus Sg\ Y = Sg\ P$
2. Perform an unsigned multiplication of the magnitudes.
3. Convert the result to two's complement system (if the sign is negative).

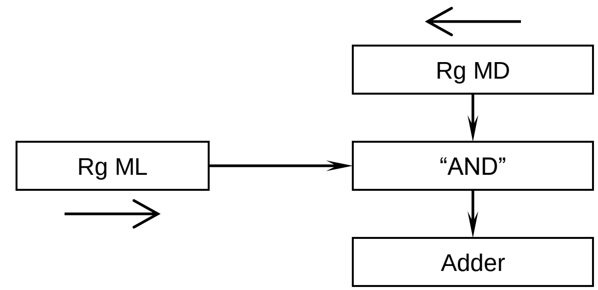
Example 1: Algorithm 1

$A = -10$ $B=13$

$A_{CC}=10110$ $B=01101$

1. $1 \oplus 0 = 1$
2. $|A| = 0.1010$ $|B| = 0.1101$

2)



Multiplier	Adder	
	0000 0000	
110[1]	1010	+A
	0000 1010	
011[0]	00000	+0, A←, B→
	0000 1010	
001[1]	10 1000	+A←, B→
	0011 0010	
000[1]	101 0000	+ A←, B→
	1000 0010	

$|C| = 0.\ 1000\ 0010$

$C_{CC}=1.\ 0111\ 1110$

$C= -130$

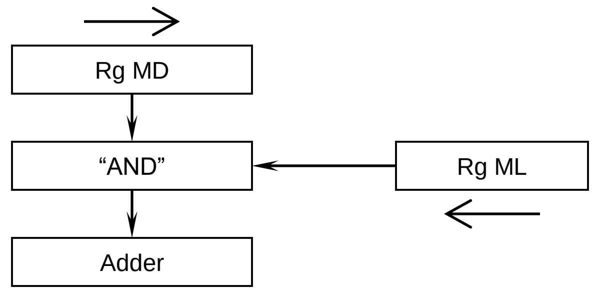
Example 2: Algorithm 2

$A = -10$ $\quad$ $B = 13$

$A_{CC} = 10110$ $\quad$ $B = 01101$

1. $1 \oplus 0 = 1$
2. $|A| = 0.1010$ $\quad$ $|B| = 0.1101$

4)



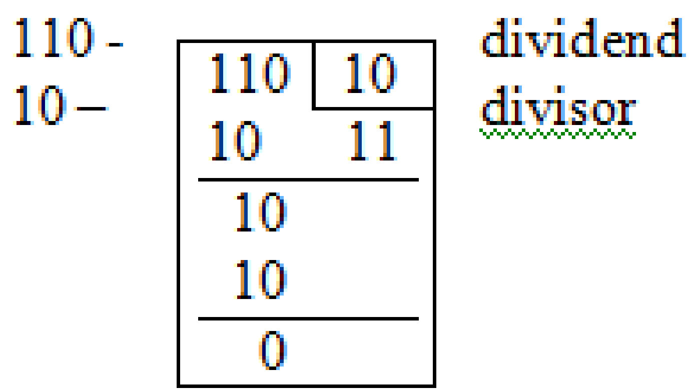
Multiplier	Adder	
1101	0000 0000	
	01010	+A→, We start with the first shift
	0101 0000	
1010	001010	+A→, B←
0100	0111 1000	
	0000 000	+0, B←
	0111 1000	
1000	0000 1010	+A→, B←
	1000 0010	

$|C| = 0.\ 1000\ 0010$

$C_{CC} = 1.\ 0111\ 1110$ $\quad$ $C = -130$

Binary division

For unsigned decimal and binary numbers we mentally compare the reduced dividend with multiples of the divisor to determine which multiple of the shifted divisor to subtract.



In the binary case the choice is somewhat simpler, since the only two choices can exist: 0 and 1.

```
(dividend)    1101110 | 10      divisor
              1010      1011    (quotient)
              -------
              00111
               1010
              -------
              -1101
               1010
              -------
               01111
                1010
              -------
                01010
                 1010
              -------
                    0
```

- If the remainder is positive, then the quotient bit is 1.
- If the remainder is negative, then the quotient bit is 0.
- In case of negative remainder, we first reestablish the last positive remainder and then subtract the shifted divisor.

Division algorithms

1. With reestablishment of the remainder, and shifting it left.
2. With reestablishment of the remainder and shifting the divisor to right.

Using these algorithm we obtain the quotient bit in 2 steps if the remainder is positive and in 3 steps if the remainder is negative. So this process is not very convenient.

3. Without reestablishment of the remainder and shifting it to left.
4. Without reestablishment of the remainder and shifting the divisor to right.

Division with reestablishment of the remainder in signed-magnitude system

Division of signed numbers can be accomplished using unsigned division.

If we have fixed-point fraction we can divide numbers if only **|A|<|B|**, because otherwise we can obtain a integer part of a quotient and this is overflow.

1. Sign bit is computed as XOR of input sign bits.
2. Find absolute values of A and B (operands).
3. First subtraction |A|-|B| (is substituted with addition in two's complemented system).
4. a) If the remainder is positive, then operation is stopped and the pseudo sign bit is 1.

   b) If the remainder is negative, |A|<|B| and the pseudo sign bit is 0.
5. The reestablishment of the dividend is done by addition of the divisor to the remainder.
6. The dividend is shifted left.
7. Subtraction of the divisor.

   a) If the remainder is positive the quotient bit is 1. The remainder is shifted left.

   b) If the remainder is negative, the quotient bit is 0 and the reestablishment of the last positive remainder is done. Then it is shifted left.

- The number of iterations depends on the required precision.
- The algorithm can be stopped when the remainder is 0.

E.g.:

A=1.0111

B=0.1101

1) $1 \oplus 0 = 1$

2) $\lvert A\rvert = 01001$

$\lvert B\rvert = 01101$

$-\lvert B\rvert = 10011$

C=10101

Rg C	Adder	
	01001	$\lvert A\rvert - \lvert B\rvert$
	10011	
0.	11100	
	01101	$+\lvert B\rvert$(reest. $\lvert A\rvert$)
	01001	
	10010	$\overline{Ad}$
	10011	$-\lvert B\rvert$
01	00101	remainder
	01010	$\overline{Ad}$
	10011	$-\lvert B\rvert$
010	11101	
	01101	reest. rem. $+\lvert B\rvert$
	01010	
	10100	$\overline{Ad}$
	10011	$-\lvert B\rvert$
0101	00111	
	01110	$\overline{Ad}$
	10011	$-\lvert B\rvert$
01011	00010	

Division without reestablishment of the remainder in signed-magnitude system

- neg. rem. $R_i=2R_{i-1}-|B|$
- reest. $R_i=2R_{i-1}-|B|+|B|=2R_{i-1}$
- shift $R_i`=4R_{i-1}$
- subtracting $\mathbf{R_i`=4R_{i-1}-|B|}$

without reest.

- neg. rem. $R_i=2R_{i-1}-|B|$
- shift $R_i`=4R_{i-1}-2|B|$
- addition $R_i`=4R_{i-1}-2|B|+|B|=$ $\mathbf{4R_{i-1}-|B|}$

Division without reestablishment of the remainder in signed-magnitude system

Rule: After first control subtraction, the sign of the remainder is examined.

If the sign is positive, the remainder is shifted and then subtraction of |B| is done.

If the sign is negative, the remainder is shifted and the addition of |B| is done.

The quotient is obtained using the same rules as in first algorithm.

Rg C	Adder	
	01001	$\lvert A\rvert - \lvert B\rvert$
	10011	
0.	11100	
	11000	$\overline{Ad}$
	01101	$+\lvert B\rvert$
01	00101	rem
	01010	$\overline{Ad}$
	10011	$-\lvert B\rvert$
010	11101	rem
	11010	$\overline{Ad}$
	01101	$+\lvert B\rvert$
0101	00111	rem
	01110	$\overline{Ad}$
	10011	$-\lvert B\rvert$
01011	00001	rem

BCD arithmetic

- Two main differences between decimal and binary arithmetic:

  1. In decimal arithmetic, the carry out takes 10 1's from position, but when we add two 4 bit binary strings, carry out takes 16.

  2. In decimal arithmetic, the carry out appears when the sum is larger than 9, for BCD it's 15.

These differences require the correction of the result in certain cases:

1. $a_i + b_i + c_{in} \leq 9$

   In this case correction is not necessary.

0011
0100
0
0111 (7)

2. $9 < a_i + b_i + c_{in} \leq 15$

   Decimal carry out appears, but binary not.

a_i=5	0101
b_i=8	1000
c_{i-1}=1	1
14	------
	1110 (illegal combination)
	0110

0001	0100
(1)	(4)

3. $a_i + b_i + c_{in} > 15$

   Binary and decimal carry outs appear, the correction is still +6.

a_i=7	0111
b_i=9	1001
c_{i-1}=0	0
16	------
	0000 (carry out)
	0110

0001	0110
(1)	(6)

Rules for BCD addition

1. If the sum is smaller or equal to 9 the addition is done without correction.
2. If after addition illegal combination appears or carry out occurs the correction is 6 (0110).
3. Carry out which appears after correction is added to the next nibble.

E.g.:
A=57985
B=24593

0101 0010	0111 0100	1001 0101	1000 1001	0101 0011
0111	1011 0110	1111 0110	0001 0110	1000
1000 8	0010 2	0101 5	0111 7	1000 8

E.g.: A=032891
B=067584

0000 0000	0011 0110	0010 0111	1000 0101	1001 1000	0001 0100
0000	**1001**	**1001**	1110 0110	0001 0110	0101
000**1** 1	**0000** 0	**0000** 0	0100 4	0111 7	0101 5

Types of Shifts

- **Logic shift**

A logic shift is the shift of bits in a constant number of cells (corresponding to a processor register – that's why the number of bits is limited to a constant); in case of shifting left, we loose the MSB, when shifting right we loose the LSB, the blank position is filled with a zero value.

Example 1:
A=38 A=00100110

SHL=01001100 (76_{10})
SHR=00010011 (19_{10})

Arithmetic shift

In this shift the sign bit is not changed. The digit next to the sign is lost if the number is shifted left and the sign bit is doubled if the number is shifted right. In this case the LSB is lost.

Example 2:

A_{CC}=00010101 (21_{10})

- SAL=00101010 (42_{10})
- SAR=00001010 (10_{10})

B=11011111 (-33_{10})

- SAL=10111110 (-66_{10})
- SAR=11101111 (-17_{10})

Round shift

No bit is lost in this case, because MSB and LSB are connected so that each bit moved out of the number is displaced on the other blank side of it.

Example 3:

A=11010010

- ROL=10100101
- ROR=01101001