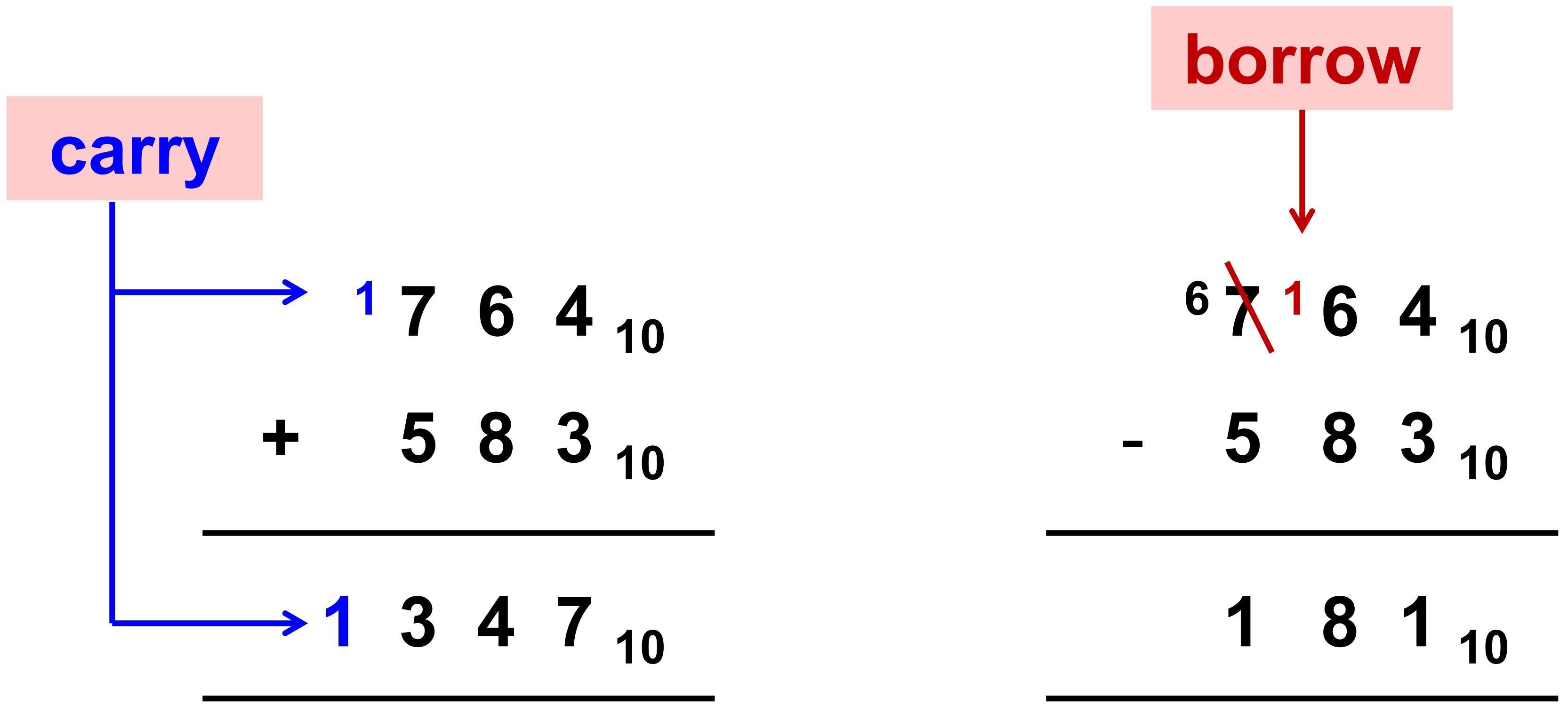
4. Digital Arithmetic

In digital circuits, e.g. digital computers and electronic calculators, arithmetic operations are carried out on binary numbers:

- **Addition, subtraction, multiplication and division**

Binary Addition and Subtraction both begin with the LSB (least significant bit)

Decimal Addition and Subtraction both begin with the LSD:



Binary addition example:

$10111_2 + 1010_2 = ?$ $(23_{10} + 10_{10} = 33_{10})$

$$
\begin{array}{r}
1\;0\;1\;1\;1_2 \\
+\quad 1\;0\;1\;0_2 \\
\hline
\\
\hline
\end{array}
$$

Adder

Half adder (HA)

- combination logic circuit that performs addition of 2 bits

Carry = $A \cdot B$

Sum = $A \oplus B$

Inputs		Outputs	
A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Full adder (FA)

- combination logic circuit that performs addition of 3 bits

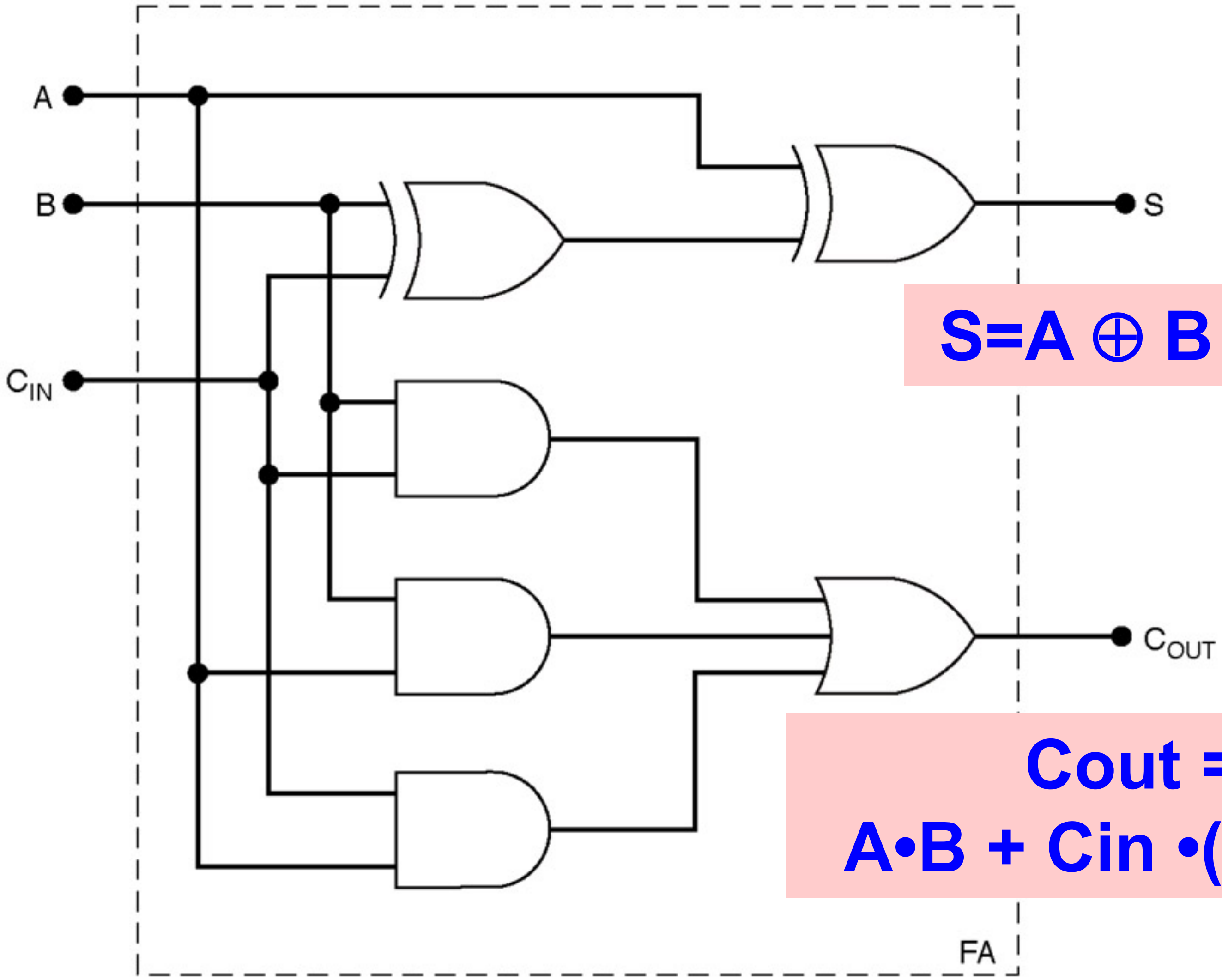
Inputs			Outputs	
A	B	Cin	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Since the truth table is rather simple, we can obtain the Boolean expressions by observation,

Sum = $A \oplus B \oplus Cin$

Cout = $A \bullet B + B \bullet Cin + A \bullet Cin$

Fig 6.81 shows the circuit implementation.



S=A ⊕ B ⊕ Cin

Cout =
A•B + Cin •(A + B)

Fig. 6-81 Full adder circuit

If we re-arrange the Boolean expression,

$$\text{Cout} = A\bullet B + B\bullet \text{Cin} + A\bullet \text{Cin}$$

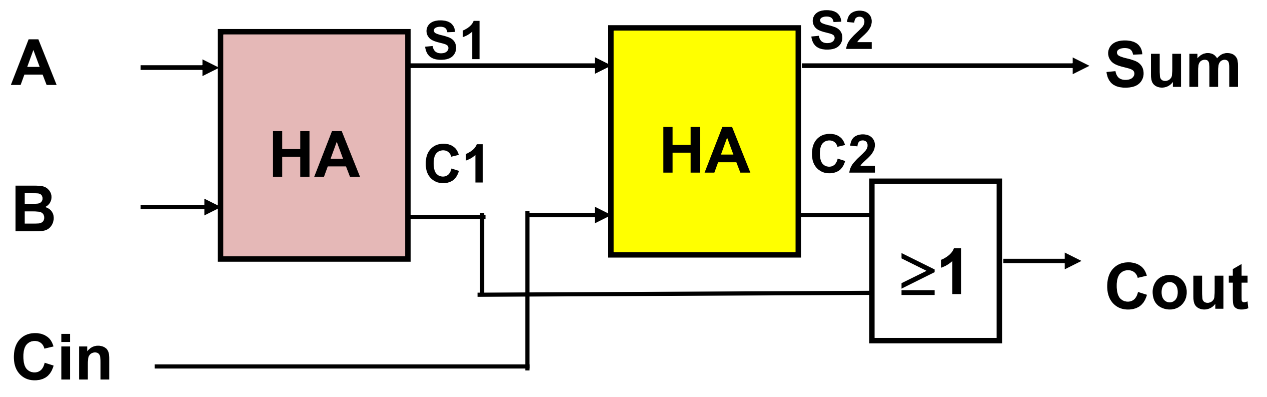
$$= A\bullet B + \text{Cin}\ (A'B+AB) + \text{Cin}\ (AB'+AB)$$

$$= \underbrace{A\bullet B + \text{Cin}\ (AB)} + \text{Cin}\ (A'B+AB')$$

$$= A\bullet B + \text{Cin}\bullet(A \oplus B)$$

- **An alternate circuit implementation using Half-adders is obtained**

Full adder circuit implemented using 2 half-adders and an OR gate



$S1 = A \oplus B$

$C1 = A \bullet B$

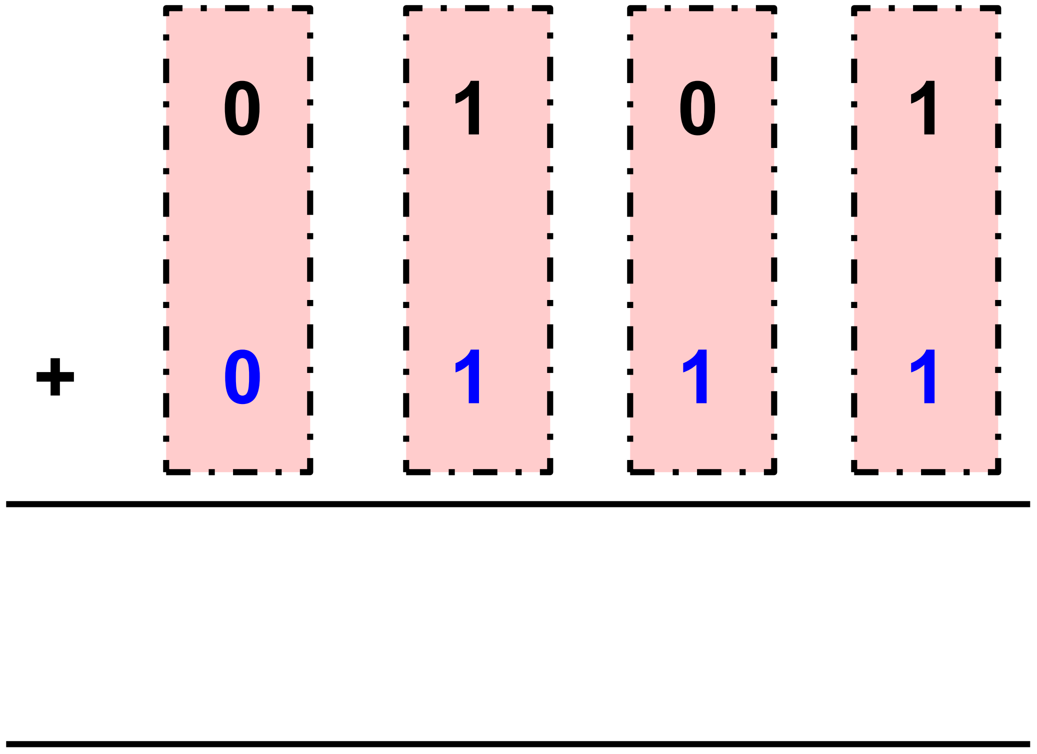
$S2 = (A \oplus B) \oplus Cin$

$C2 = Cin \bullet (A \oplus B)$

$Cout = A \bullet B + Cin \bullet (A \oplus B)$

Binary addition example: 0101_2 + 0111_2 = ?

Need 4 FAs



Parallel Adder

- N full-adders can be cascaded to form an N-bit parallel adder
- also known as ripple adder
- all the bits of the augend and addend are fed into the adder circuits simultaneously
- addition is very fast
- addition speed is limited by propagation delays of FAs - carry propagation
- Fig. 6-82 shows a 4-bit adder

Example: $0011_2 + 0001_2 = 0100_2$

(In decimal: 3 + 1 = 4)

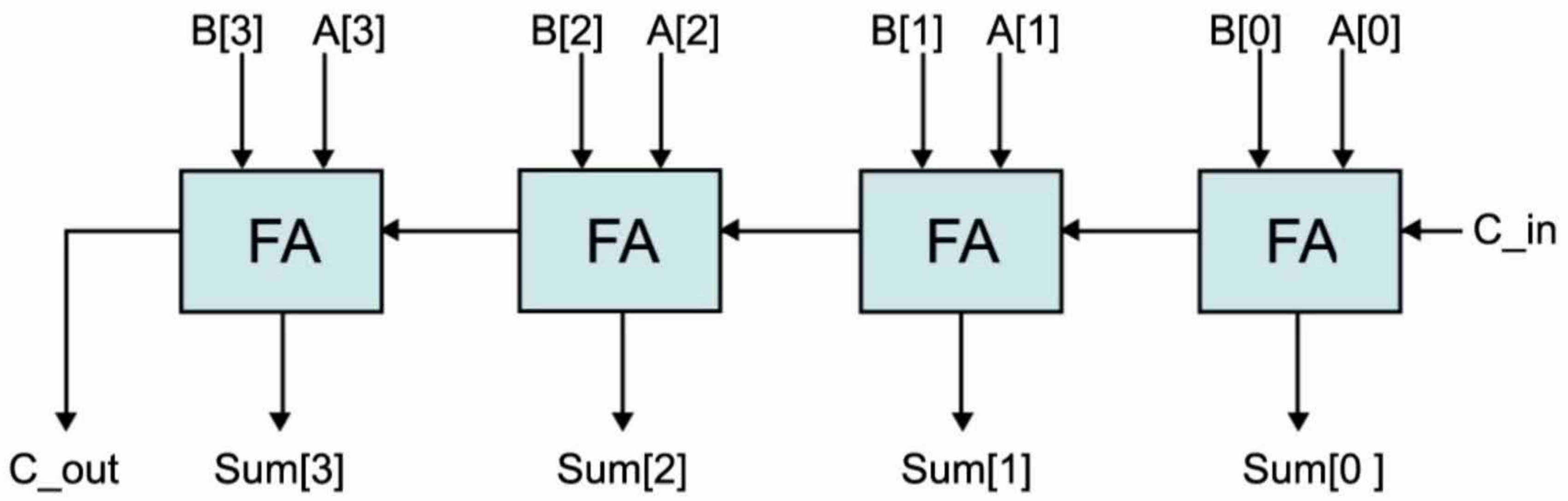


Fig. 6-82 : 4-bit ripple adder

Representing Signed Numbers

Signed-Magnitude system

- sign bit = MSB : 0 for positive numbers
- sign bit = MSB: 1 for negative numbers

$14_{10} = 1\,1\,1\,0_2$

	sign	magnitude
+14	0	1 1 1 0
-14	1	1 1 1 0

Signed-Magnitude system

- There are equal numbers of positive and negative values
- An N-bit value lies in the range

$$\{ -(2^{N-1} - 1) \text{ to } +(2^{N-1} - 1) \}$$

Examples:

$0101\ 0101_2 = +85_{10}$ $1101\ 0101_2 = -85_{10}$

$0000\ 0000_2 = +0_{10}$ $1000\ 0000_2 = -0_{10}$

2's complement system

- **sign bit = MSB: 0 for zero and positive numbers**
- **sign bit = MSB: 1 for negative numbers**

$14_{10} = 1\ 1\ 1\ 0_2$

	sign	
+14	0	1 1 1 0
-14	1	???

2's complement operation:

Step 1: invert every bit of a binary number (i.e. perform 1's complement)

$$1\,1\,1\,0 \longrightarrow 0\,0\,0\,1$$

Step 2: add (arithmetic addition) 1 to it

$$\begin{array}{lr} & 0\,0\,0\,1 \\ + & 1 \\ \hline & 0\,0\,1\,0 \\ \hline \end{array}$$

The 2's complement of 1110 is 0010

2's complement system

$14_{10} = 1\ 1\ 1\ 0_2$

	sign	
+14	0	1 1 1 0
-14	1	0 0 1 0

A short-cut method for 2's complement operation:

- starting from LSB, copy the bit if it is ‘0’ and repeat process with remaining bits
- copy the bit if it is the first ‘1’, then invert all the remaining bits
- a sequential process

This method works on all binary bit patterns

Examples:

$$\underbrace{0\ 1\ 0\ 1\ 0\ 1\ 0}\ \underset{\downarrow}{1}_2 = +85_{10}$$

$$1\ 0\ 1\ 0\ 1\ 0\ 1\ 1_2 = -85_{10}$$

Invert the remaining bits

Copy the bits from the right until it reaches the first 1

$$\underset{\downarrow}{0}\ \underbrace{1\ 0\ 0\ 0\ 0\ 0\ 0}_2 = +64_{10}$$

$$1\ 1\ 0\ 0\ 0\ 0\ 0\ 0_2 = -64_{10}$$

2's complement system

- There is 1 more negative value than positive ones
- An N-bit value lies in the range

$$\{ -(2^{N-1}) \text{ to } +(2^{N-1} - 1) \}$$

- Eg. 4 bits: {-8, 7}
- 8 bits: {-128, 127}

Summary: representing signed numbers in 2's complement system:

If the number is zero or positive

- represent its magnitude in binary
- append a sign bit (0) in front of the MSB if the MSB is 1

If the number is negative

- represent its magnitude in binary
- obtain the 2's complement
- append a sign bit (1) in front of the MSB if the MSB is 0

Examples: Represent the following signed decimal numbers in 2’s complement

(a) 5

- binary of 5 is 101
- Positive 5 is **0101**

(b) -7

- binary of 7 is 111
- 2's complement of 111 is 001
- Negative 7 is **1001**

E.g. A 4-bit 2's complement number system:

Decimal	Binary	
	magnitude	2's comp
-8	1000	1000
-7	0111	1001
-6	0110	1010
-5	0101	1011
-4	0100	1100
-3	0011	1101
-2	0010	1110
-1	0001	1111
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111

Generalised to an N-bit 2's complement number system:

Decimal	2's comp	
$-(2^{N-1})$	1000…00	← Most negative
$-(2^{N-1}) + 1$	1000…01	
$-(2^{N-1}) + 2$	1000…10	
.	.	
.	.	
-1	1111…11	← -1
0	0000…00	← zero
1	0000…01	
2	0000…10	
.	.	
.	.	
$(2^{N-1}) - 2$	0111…10	
$(2^{N-1}) - 1$	0111…11	← Most positive

Some special bit patterns:

- 100….000 (with N-1 zeros)

  the decimal value is $-(2^{N-1})$

- 111….111 (all ones)

  the decimal value is -1

- 000….000 (all zeroes)

  the decimal value is 0

- 011….111 (with N-1 ones)

  the decimal value is $(2^{N-1}-1)$

Obtaining the decimal value of a 2's complement number

If the number is positive:

- perform a binary-to-decimal conversion on the number

If the number is negative:

- First perform 2’s complement to convert it to positive
- Next perform a binary-to-decimal conversion on the positive number

Examples: Obtain the decimal equivalent of the following 2’s complement numbers

(a) 01001

- The magnitude 1001 is 9 in decimal
- 01001 is positive 9 in decimal

(b) 10011

- 2's complement of 10011 is 01101
- The magnitude 1101 is 13 in decimal
- 10011 is negative 13 in decimal

Some observations on the 2's complement number system

•The 2's complement of an N-bit binary number is also of N bits.

•To represent a binary number with N significant bits in its magnitude using the 2's complement system, we need (N+1) bits. The extra bit is the sign bit. *

* note exception

- **If there are more bits than necessary to represent a binary number in the 2's complement system, the more significant bits are filled with the sign bit, which is 0 for positive and 1 for negative – known as sign extension**

101 = 1101 = 11101

011 = 0011 = 00011

- **A 2's complement operation will change a positive number to negative and vice-versa, with no change in the magnitude. ***

*** note exception**

* Exception arises when dealing with the most negative number that can be represented given a number of bits.

e.g.

-8 in a 4-bit system,

-16 in a 5-bit system

-128 in an 8-bit system

$-(2^{N-1})$ in an N-bit system

Reasons for using the 2's complement system:

- Using the 2's complement representation for signed numbers, subtraction of numbers can be carried out in the same way as addition
- Therefore, the same set of hardware circuits can be used for both subtraction and addition

Addition in the 2's complement system

two positive numbers

+10	0	1 0 1 0	(augend)
+3	0	0 0 1 1	(addend)
+13	0	1 1 0 1	(sum)

sign bit

The sign bits are added like the other bits

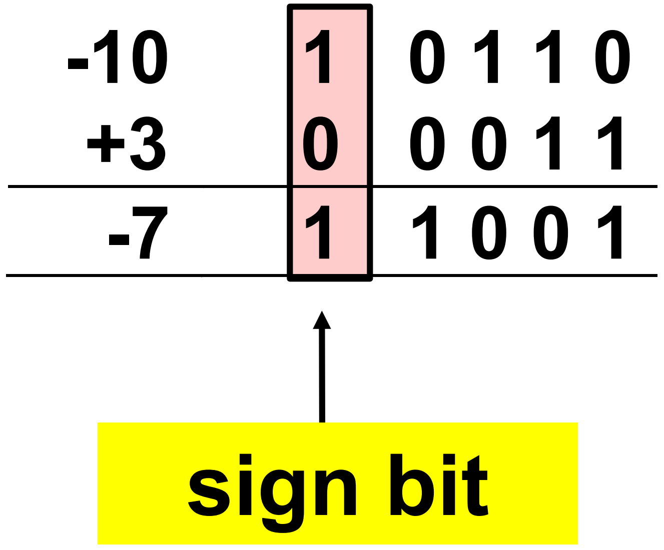
positive number and smaller negative number

```
+10          0  1 0 1 0
 -3          1  1 1 0 1
-----------------------
 +7     1    0  0 1 1 1
-----------------------
```

Ignore carry out

sign bit

positive number and larger negative number



two negative numbers

-10		1	0 1 1 0
-3		1	1 1 0 1
-13	1	1	0 0 1 1

Ignore carry out

sign bit

Subtraction in the 2's complement system

Subtraction can be carried out using 2's complement and addition

A - B = A + (-B)

-B = 2's complement of B

B may be positive or negative

Examples:

10 - 3 = 10 + (-3) = 7

-3 is 2’s complement of 3

$$\begin{array}{r}
{}^{1}0\;1\;0\;1\;0 \\
+\;\;1\;1\;1\;0\;1 \\
\hline
1\;0\;0\;1\;1\;1 \\
\hline
\end{array}$$

Ignore carry out

-10 - (-3) = -10 + 3 = -7

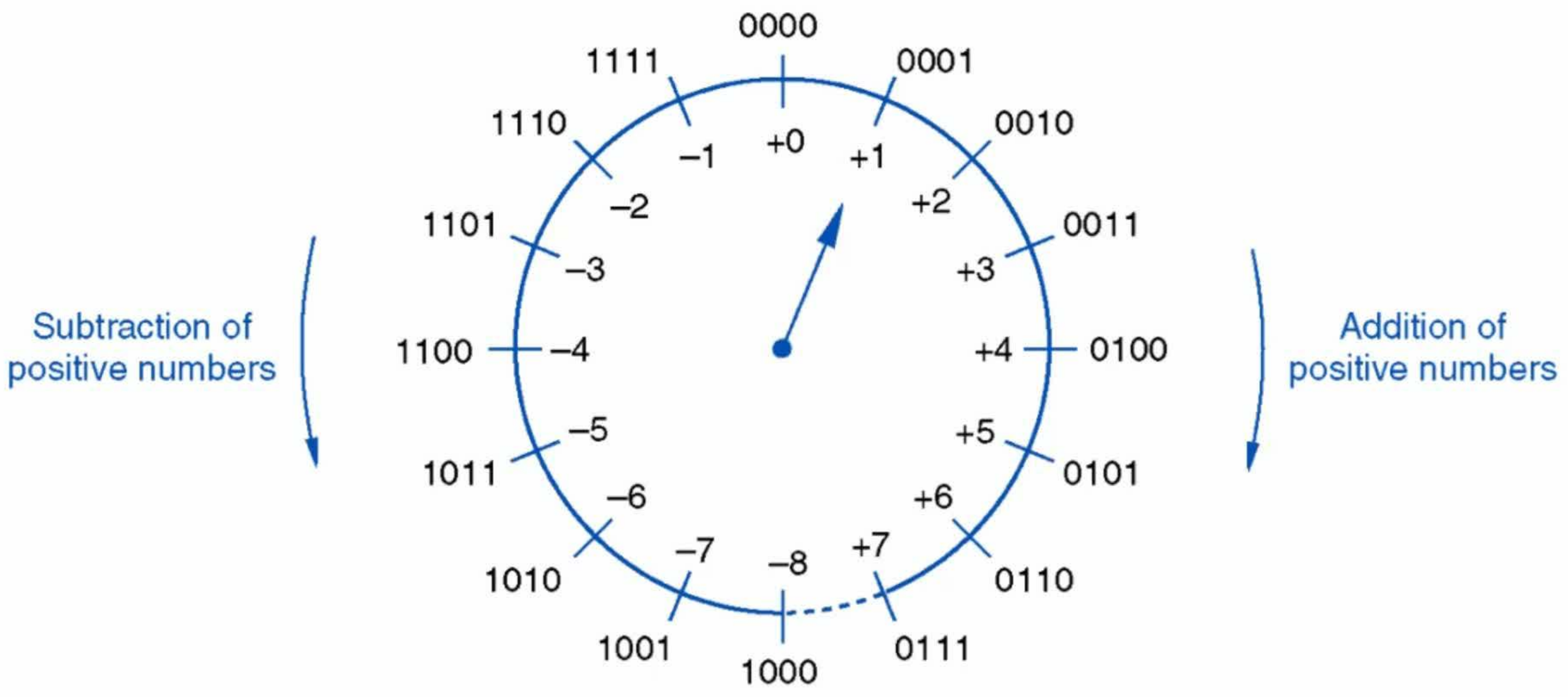
3 is 2’s complement of -3

$$\begin{array}{r}
1\;{}^{1}0\;{}^{1}1\;1\;0 \\
+\;\;0\;\;0\;\;0\;1\;1 \\
\hline
1\;\;1\;\;0\;0\;1 \\
\hline
\end{array}$$

Remember:

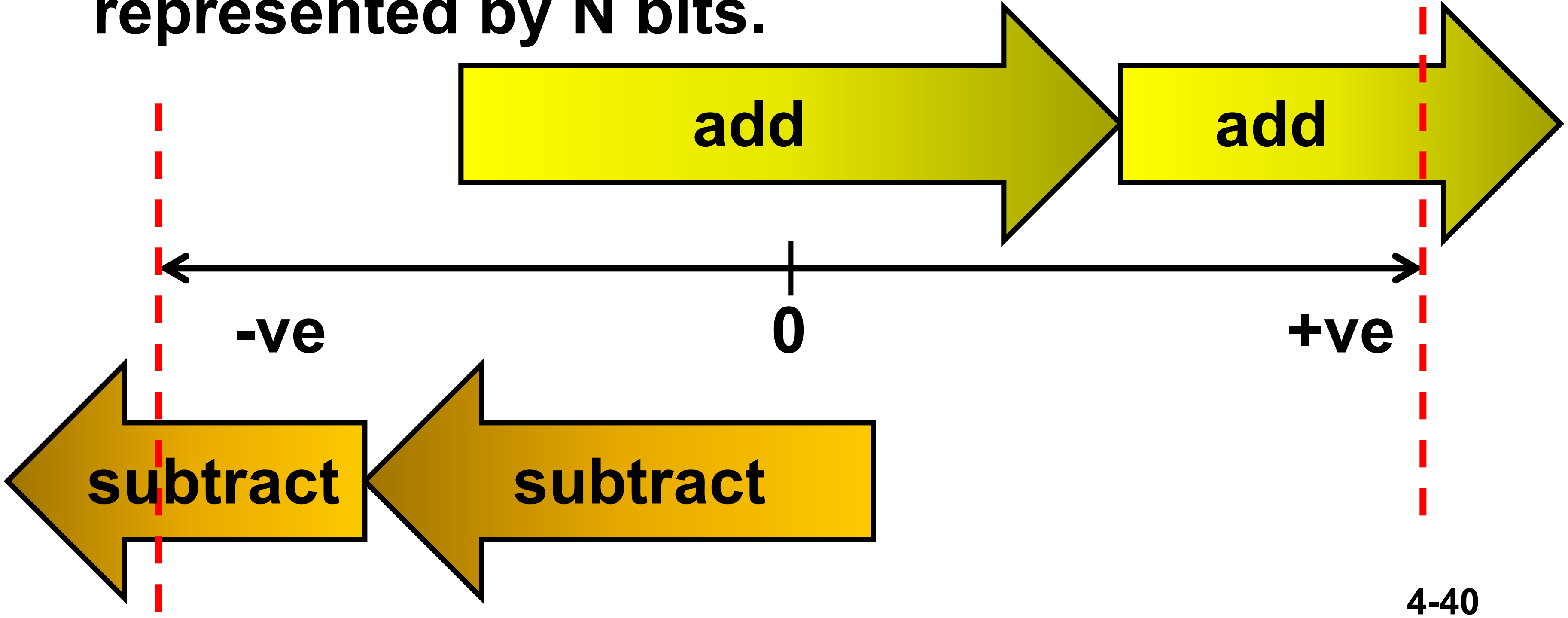
The pair of numbers to be added/subtracted must have the same size, i.e. same number of bits. This ensures that the sign bits are aligned. The resulting sum must also be of the same size.

Addition and Subtraction in the 2's complement system



Arithmetic Overflow

It occurs when an arithmetic operation between two N-bit operands produces a result that cannot be sufficiently represented by N bits.



Rules to detect overflow in 2's complement addition:

No overflow occurs if the operands have opposite signs. Overflow is detected when operands have the **same sign**, but the arithmetic sum has an **opposite sign**.

$$\begin{array}{r} {}^{1}0\;1\;0\;1\;0 \\ +\quad 0\;1\;0\;0\;0 \\ \hline 1\;0\;0\;1\;0 \\ \hline \end{array}$$

10+8=-14?

For **subtraction**, no overflow occurs if the operands have the same sign.

Combined circuit for addition and subtraction

Fig 6-12 (X0 and Y0 are LSB)

- To perform the addition: X + Y
- Make inputs Add/Sub = 0, C0 = 0
- XOR gates do not invert Y
- Inputs to FAs are X and Y, with C0 = 0
- E.g. 0001 + 0011 = 0100
- i.e. 1 + 3 = 4 (in decimal)

Combined circuit for addition and subtraction

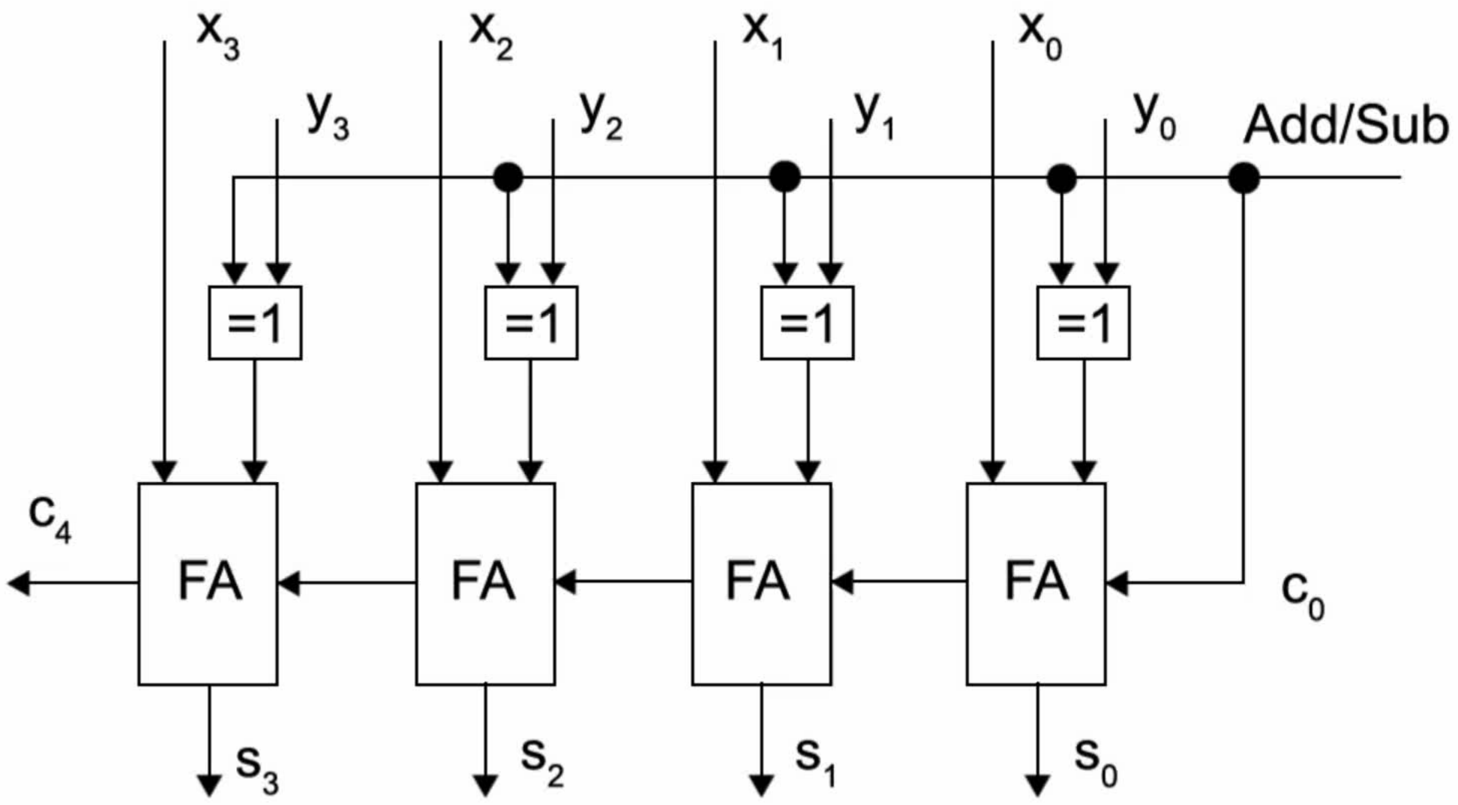


Fig 6-12 (add)

Fig 6-12 (subtraction)

- To perform the subtraction: X - Y
- Recall: -Y is 2's complement of Y
- Make inputs Add/Sub = 1, C0 = 1
- XOR gates invert Y to give Y'
- Inputs to FAs are X and Y', with C0 = 1
- E.g. 0001 - 0100 = 0001 + 1011 + 1 = 1101
- i.e. 1 - 4 = -3 (in decimal)

Combined circuit for addition and subtraction

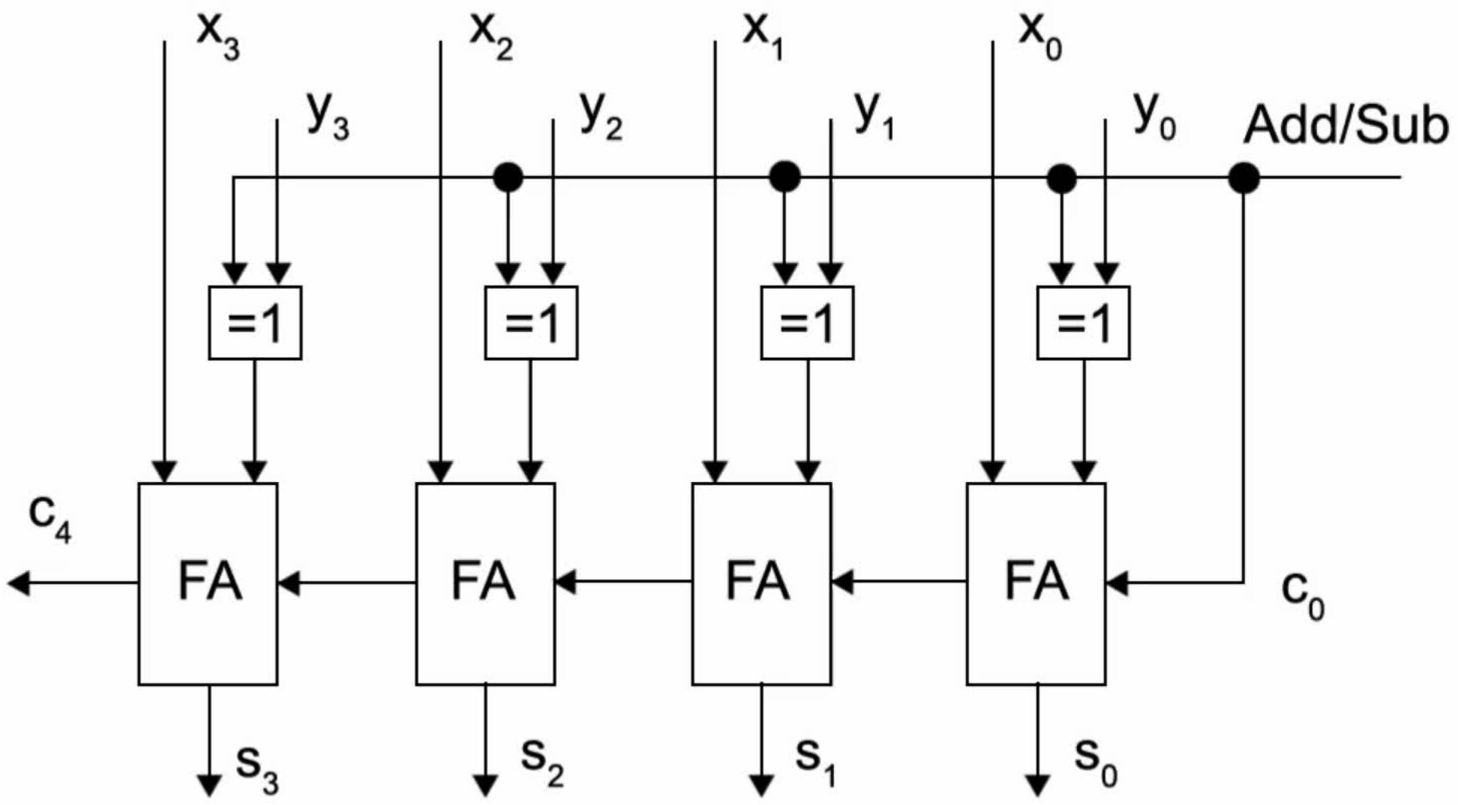


Fig 6-12 (subtract)

Fig. 6-9: Complete 4-bit Parallel Adder with Registers

t1: CLEAR* clears the contents of A register to 0’s

t2: PGT of first LOAD pulse transfers operand X from memory into B register

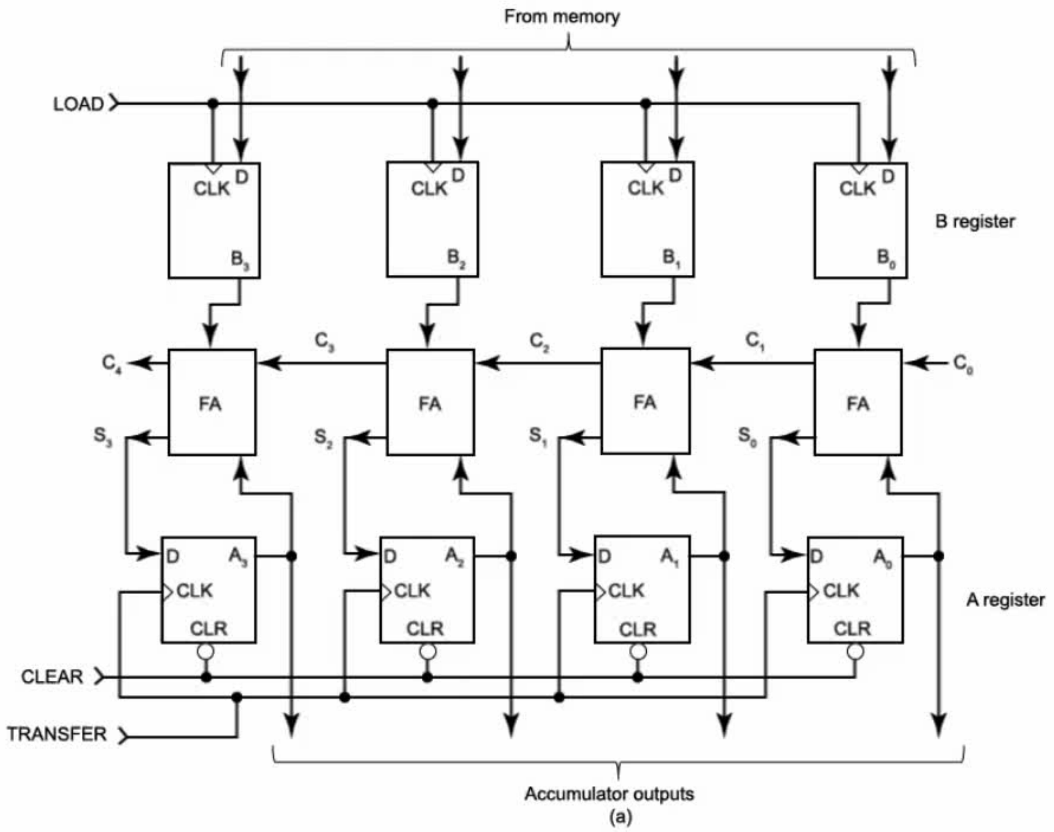
t3: PGT of first TRANSFER pulse transfers FA output (=X) into A register

t4: PGT of second LOAD pulse transfers operand Y from memory into B register

t5: PGT of second TRANSFER pulse transfers FA output (=X+Y) into A register

- **Note that sufficient time (between load and transfer) must be given to FAs to complete addition**
- **E.g. 0001 + 0010 = 0011**

Thus A register holds the result of X+Y


From memory
LOAD
CLK
D
B_3
B_2
B_1
B_0
B register
FA
C_4
C_3
C_2
C_1
C_0
S_3
S_2
S_1
S_0
D
CLK
CLR
A_3
A_2
A_1
A_0
A register
CLEAR
TRANSFER
Accumulator outputs
(a)

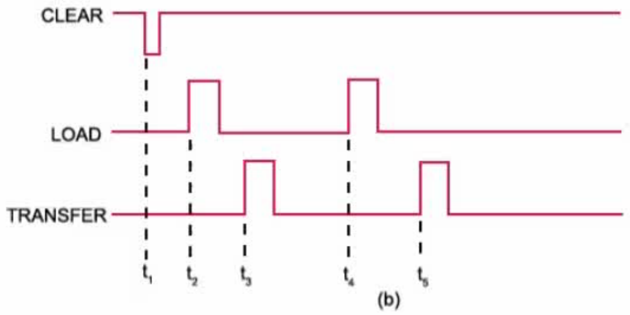

CLEAR
LOAD
TRANSFER
t_1
t_2
t_3
t_4
t_5
(b)

Fig.6-9

Carry propagation can be reduced (hence increasing speed of addition) by a special-purposed logic circuit

- carry-look-ahead circuit

Many IC parallel adders have built-in carry look-ahead circuit to speed up the addition.

Binary Multiplication

Unsigned multiplication:

similar to decimal multiplication

```
      0111     (7)
   x  1100    (12)
  ____________________
```

2’s complement multiplication:

- If the multiplier is positive, same as unsigned multiplication
- If the multiplier is negative, need to take care of negative weight of MSB (i.e. the sign bit)
- Treat multiplier as a sum of 2 parts:
  - MSB – negative part
  - Remaining bits – positive part

Example of 2’s complement multiplication:

```
      -5
x     -3
--------
      15
--------
```

Explanation:

$$1011 \times 1101 = 1011 \times (1000 + 0101)$$

$$= \underbrace{(1011 \times 1000)}_{\text{This is done by shift and 2's complement}} + \underbrace{(1011 \times 0101)}_{\text{This is done by shift and sign-extension}}$$

Binary Division

Unsigned division:

similar to the long division method in decimal arithmetic

$$9 \div 3 = 3$$

Signed division:

convert the signed numbers to unsigned, divide them as above, then convert the result using the appropriate sign representation

BCD addition

When the sum of two BCD digits does not exceed 9_{10}, the operation is the same as binary addition

If the sum of two BCD digits is more than 9_{10} , a correction needs to be made by adding 6 (0110) to skip over the six invalid codes

The correction involves two steps:

- **a carry of decimal value 1 is brought forward and added to the next higher digit**
- **the decimal value 6 is added to the sum to obtain the correct BCD digit**

Example:

$$24_{10} + 47_{10} = 71_{10}$$