ITI1100B

CHAPTER-1

Digital Systems and **Binary Numbers**

Numeric Information is represented in **Number Systems**

- Decimal number system
- Binary number system
- Octal number system
- Hexadecimal number system

Q: Why do we use a number system?
A: To determine the value represented by a number.

Q: What is the value represented by the number 34.25?
A: Tell me the **base** for the number!

OK. It is **base 10**.

A: Then the value represented by the number 34.25

$= 3 \times 10^1 + 4 \times 10^0 \bullet 2 \times 10^{-1} + 5 \times 10^{-2}$

$= 3 \times 10 + 4 \times 1 \bullet 2 \times 1/10 + 5 \times 1/100$

$= (34.25)_{10}$

= thirty four point twenty five

RADIX (BASE) r NUMBER SYSTEM

A number in radix r number system
is represented in terms of POSITIONAL WEIGHTING

RADIX POINT

$$(N)_r = d_{n-1}\, r^{n-1} + d_{n-2}\, r^{n-2} + \ldots + d_1\, r^1 + d_0\, r^0 \bullet d_{-1}\, r^{-1} + \ldots + d_{-m}\, r^{-m}$$

INTEGRAL PART **FRACTIONAL PART**

r = radix or base
. = radix point
d_k = digit in position k, $-m \le k \le n-1$
r^k = weight of position k, $-m \le k \le n-1$
n = number of integral digits (to the left of the radix point) in N
m = number of fractional digits (to the right of the radix point) in N
d_{n-1} = most significant digit (MSD)
d_{-m} = least significant digit (LSD)

(0, ..., r-1) = DIGITS IN RADIX r NUMBER SYSTEM

NOTES:
In the textbook,
digit "d" is called coefficient and represented by "a"

e.g.,

Base 10 Number System

r = 10 (**DECIMAL** number system)

digits are (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

$$(34.25)_{10} = 3 \times 10^1 + 4 \times 10^0 \bullet 2 \times 10^{-1} + 5 \times 10^{-2}$$
$$= 3 \times 10 + 4 \times 1 \bullet 2 \times 1/10 + 5 \times 1/100$$

e.g.,

Base 2 Number System

r = 2 (**BINARY** number system)

digits are (0, 1)

$$(100010.01)_2 = 1x2^5 + 0x2^4 + 0x2^3 + 0x2^2 + 1x2^1 + 0x2^0 \bullet 0x2^{-1} + 1x2^{-2}$$
$$= 1x32 + 0x16 + 0x8 + 0x4 + 1x2 + 0x1 \bullet 0x1/2 + 1x1/4$$

e.g.,

Base 8 Number System

r = 8 (OCTAL number system)

digits are (0, 1, 2, 3, 4, 5, 6, 7)

$(42.2)_8 = 4 \times 8^1 + 2 \times 8^0 \bullet 2 \times 8^{-1}$
$= 4 \times 8 + 2 \times 1 \bullet 2 \times 1/8$

e.g.,

Base 16 Number System

r = 16 (HEXADECIMAL number system)

digits are (0, 1,.2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)

$(22.4)_{16} = 2 \times 16^1 + 2 \times 16^0 \bullet 4 \times 16^{-1}$
$= 2 \times 16 + 2 \times 1 \bullet 4 \times 1/16$

REPRESENTATION OF HEXADECIMAL, DECIMAL, OCTAL DIGITS

$2^3 2^2 2^1 2^0$	r=16	r=10	r=8
0 0 0 0	0	0	0
0 0 0 1	1	1	1
0 0 1 0	2	2	2
0 0 1 1	3	3	3
0 1 0 0	4	4	4
0 1 0 1	5	5	5
0 1 1 0	6	6	6
0 1 1 1	7	7	7
1 0 0 0	8	8	
1 0 0 1	9	9	
1 0 1 0	A		
1 0 1 1	B		
1 1 0 0	C		
1 1 0 1	D		
1 1 1 0	E		
1 1 1 1	F		

Decimal, Binary, Octal and Hexadecimal

Decimal	Binary	Octal	Hexadecimal
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Powers of 2

n	2^n
0	$2^0=1$
1	$2^1=2$
2	$2^2=4$
3	$2^3=8$
4	$2^4=16$
5	$2^5=32$
6	$2^6=64$
7	$2^7=128$

n	2^n	
8	$2^8=256$	
9	$2^9=512$	
10	$2^{10}=1024$	**Kilo**
11	$2^{11}=2048$	
12	$2^{12}=4096$	
20	$2^{20}=1M$	**Mega**
30	$2^{30}=1G$	**Giga**
40	$2^{40}=1T$	**Tera**

NUMBER BASE CONVERSIONS

I- BINARY ⇔ DECIMAL

A) BINARY to DECIMAL
construct the polynomial representation of the number,
then sum the products

e.g., $(100010.01)_2 = 1x2^5 + 0x2^4 + 0x2^3 + 0x2^2 + 1x2^1 + 0x2^0 \bullet 0x2^{-1}+1x2^{-2}$
$= 1x32 + 0 + 0 + 0 + 1x2 + 0 + 0 + 1x1/4$
$= 32 + 2 + 1/4 = 34 + 0.25$
$= (34.25)_{10}$

B) DECIMAL to BINARY

Integral part and Fractional part are converted differently!

i) Covert integral part

ii) Convert fractional part

iii) Combine results of conversions in i) and ii)

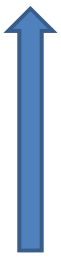
Integral part conversion

- Divide the integral part by the 'Base' (=2)
- Record the remainder (either 0 or 1)
- Take the quotient and repeat the division until quotient becomes 0
- Collect the remainders in reverse order (i.e., bottom-up)

e.g., $(34.25)_{10} = (?)_2$

i) CONVERT INTEGRAL PART $(34)_{10}$
BY DIVIDING IT SUCCESSIVELY BY 2

	Quotient	Remainder	
34 / 2 =	17	0	(least significant bit)
17 / 2 =	8	1	
8 / 2 =	4	0	
4 / 2 =	2	0	
2 / 2 =	1	0	
1 / 2 =	0	1	(most significant bit)



AND
BY READING THE REMAINDERS UP,
THE RESULT BECOMES $(34)_{10} = (100010)_2$

Fractional part conversion

- Multiply the fractional part by the 'Base' (=2)
- Record the integral part of the partial product (either 0 or 1)
- Take the fractional part of the partial product and repeat the multiplication until fractional part of partial product becomes 0 (which may not always happen!!!)
- Collect the integer parts of the partial products in reverse order (i.e., top-down)

e.g., $(34.25)_{10} = (?)_2$ (Continued)

ii) CONVERT FRACTIONAL PART $(.25)_{10}$
BY MULTIPLYING IT SUCCESSIVELY BY 2

	Partial Product	Integral part
0.25 x 2 =	0.50	0
0.50 x 2 =	1.00	1

AND
BY READING THE INTEGRAL DIGITS DOWN,
THE RESULT BECOMES $(.25)_{10} = (0.01)_2$

iii) COMBINE THE RESULTS FROM i) and ii)
$(34.25)_{10} = (100010.01)_2$

More on Fractional Part Conversion

… repeat the multiplication until fractional part of partial product becomes 0 (which may not always happen!!!)

e.g., fractional part of partial product becomes 0
$(0.3125)_{10} = (?)_2$

Partial Product	Integral part
0 . 3125 x 2 = 0 . 6250	0
0 . 6250 x 2 = 1 . 2500	1
0 . 2500 x 2 = 0 . 5000	0
0 . 5000 x 2 = 1 . 0000	1

$(0.3125)_{10} = (0.0101)_2$

e.g., fractional part of partial product does not become 0
$(0.2)_{10} = (?)_2$

	Partial Product	Integral part
0 . 2 x 2 =	0 . 4	0
0 . 4 x 2 =	0 . 8	0
0 . 8 x 2 =	1 . 6	1
0 . 6 x 2 =	1 . 2	1
0 . 2 x 2 =	0 . 4	0
:	:	:

$(0.2)_{10} = (0.001100110011.....)_2$
ROUNDING AND TRUNCATION

II- BINARY ⇔ OCTAL

A) BINARY to OCTAL

Form groups of 3 BITS starting at radix point in both directions

(EACH GROUP OF 3 BITS IS AN OCTAL DIGIT !!!)

e.g., $(100010.01)_2 = \underset{4}{100} \quad \underset{2}{010} \bullet \underset{2}{010}$

$= (42.2)_8$

B) OCTAL to BINARY

Convert each octal digit in its equivalent in Binary

e.g., $(27.5)_8 = \underset{010}{2} \quad \underset{111}{7} \bullet \underset{101}{5}$

$= (10111.101)_2$

III- BINARY ⇔ HEXADECIMAL

A) BINARY to HEXADECIMAL

Form groups of 4 BITS starting at radix point in both directions

(EACH GROUP OF 4 BITS IS AN HEXADECIMAL DIGIT !!!)

e.g., $(100010.01)_2 = \underset{2}{0010} \quad \underset{2}{0010} \bullet \underset{4}{0100}$

$= (22.4)_{16}$

B) HEXADECIMAL to BINARY

Convert each hexadecimal digit in its equivalent in Binary

e.g., $(5D.A)_{16} = \underset{0101}{5} \quad \underset{1101}{D} \bullet \underset{1010}{A}$

$= (1011101.101)_2$

IV- OCTAL ⇔ DECIMAL

A) OCTAL to DECIMAL
construct the polynomial representation of the number,
then sum the products

e.g., $(42.2)_8$ $= 4 \times 8^1 + 2 \times 8^0 \bullet 2 \times 8^{-1}$
$= 4 \times 8 + 2 \times 1 \bullet 2 \times 1/8$
$= (34.25)_{10}$

B) DECIMAL to OCTAL

Integral part and Fractional part are converted differently!

i) Covert integral part

ii) Convert fractional part

iii) Combine results of conversions in i) and ii)

e.g., $(34.25)_{10} = (?)_8$

Integral part conversion

i) CONVERT INTEGRAL PART $(34)_{10}$
BY DIVIDING IT SUCCESSIVELY BY 8

	Quotient	Remainder	
34 / 8 =	4	2	(least significant bit)
4 / 8 =	0	4	(most significant bit)

AND
BY READING THE REMAINDERS UP,
THE RESULT BECOMES $(34)_{10} = (42)_8$

Fractional part conversion

ii) CONVERT FRACTIONAL PART $(.25)_{10}$
BY MULTIPLYING IT SUCCESSIVELY BY 8

	Partial Product	Integral part
0.25 x 8 =	2.00	2

AND
BY READING THE INTEGRAL DIGITS DOWN,
THE RESULT BECOMES $(.25)_{10} = (0.2)_8$

iii) COMBINE THE RESULTS FROM i) and ii)
$(34.25)_{10} = (42.2)_8$

V- OCTAL ⇔ HEXADECIMAL

A) OCTAL to HEXADECIMAL
convert from Octal to Binary and then,
convert from the resulting Binary to Hexadecimal

e.g., $(27.5)_8$ = 2 7 • 5
010 111 101
= $(100010.01)_2$ = 0010 0010 • 0100
2 2 4
= $(22.4)_{16}$

B) HEXADECIMAL to OCTAL
convert from Hexadecimal to Binary and then,
convert from the resulting Binary to Octal

e.g., $(5D.A)_{16}$ = 5 D • A
0101 1101 1010
= $(1011101.101)_2$ = 100 010 • 010
4 2 2
= $(42.2)_8$

VI- HEXADECIMAL ⇔ DECIMAL

A) HEXADECIMAL to DECIMAL
construct the polynomial representation of the number,
then sum the products

e.g., $(22.4)_{16}$ $= 2 \times 16^1 + 2 \times 16^0 \bullet 4 \times 16^{-1}$
$= 2 \times 16 + 2 \times 1 \bullet 4 \times 1/16$
$= (34.25)_{10}$

B) DECIMAL to HEXADECIMAL

Integral part and Fractional part are converted differently!

i) Covert integral part

ii) Convert fractional part

iii) Combine results of conversions in i) and ii)

e.g., $(34.25)_{10} = (?)_{16}$

Integral part conversion

i) CONVERT INTEGRAL PART $(34)_{10}$
BY DIVIDING IT SUCCESSIVELY BY 16

	Quotient	Remainder	
34 / 16 =	2	2	(least significant bit)
2 / 16 =	0	2	(most significant bit)

AND
BY READING THE REMAINDERS UP,
THE RESULT BECOMES $(34)_{10} = (22)_{16}$

Fractional part conversion

ii) CONVERT FRACTIONAL PART $(.25)_{10}$
BY MULTIPLYING IT SUCCESSIVELY BY 16

	Partial Product	Integral part
0.25 x 16 =	4.00	4

AND
BY READING THE INTEGRAL DIGITS DOWN,
THE RESULT BECOMES $(.25)_{10} = (0.4)_{16}$

iii) COMBINE THE RESULTS FROM i) and ii)
$(34.25)_{10} = (22.4)_{16}$

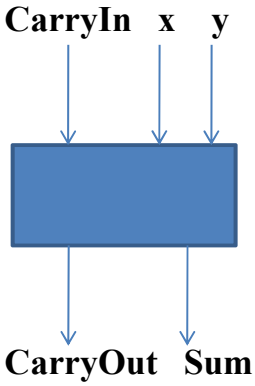
ARITHMETIC OPERATIONS IN BINARY SYSTEM

Binary Addition

	0	0	1	1
	+ 0	+ 1	+ 0	+ 1
	--	--	--	--
Sum	0	1	1	10

(Sum is 0 and CarryOut is 1)

e.g., Addition of Two Binary Digits x and y with a CarryIn



x	y	CarryIn	Sum	CarryOut
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

ARITHMETIC OPERATIONS IN BINARY SYSTEM

Binary Addition (Continued)

e.g., Binary addition of **Unsigned Integers**

	1	1	1	1	1	1		
		1	1	1	1	0	1	= 61
+			1	0	1	1	1	= 23
	1	0	1	0	1	0	0	= 84

e.g., Binary addition of **Unsigned Integers**

```
                     1          1  1
 1 0 0 1          0 0 0 1         1 1 0 0
+0 1 1 0         +1 0 0 1       + 0 1 0 1
----------       ----------     -----------
 1 1 1 1          1 0 1 0      1  0 0 0 1
```


Binary Subtraction

2 ← = $(10)_2$

	0	0	1	1
	- 0	- 1	- 0	- 1
	--	--	--	--
Difference	0	1 1	1	0

(**Difference is 1 and Barrow is 1**)

e.g., Binary subtraction of **Unsigned Integers**

		1			2			
	0	~~2~~	2	0	0	2		
	~~1~~	0	0	~~1~~	~~1~~	0	1	= 77
−			1	0	1	1	1	= 23
	0	1	1	0	1	1	0	= 54

Binary Multiplication

	0	0	1	1
	x 0	x 1	x 0	x 1
	--	--	--	--
Product	**0**	**0**	**0**	**1**

e.g., Binary Multiplication of **Unsigned Integers**

			1	**0**	**1**	**1**	**1**
x				**1**	**0**	**1**	**0**
			0	**0**	**0**	**0**	**0**
		1	**0**	**1**	**1**	**1**	
	0	**0**	**0**	**0**	**0**		
1	**0**	**1**	**1**	**1**			
1	**1**	**1**	**0**	**0**	**1**	**1**	**0**

COMPLEMENTS OF NUMBERS

Complements are used in digital systems for simplifying the subtraction operation and for logical manipulation.

There are two types of complements for each base-r number system: *diminished radix complement* and *radix complement* .

Diminished Radix Complement (also known as (r-1)'s Complement)

Given a number N in base r having n digits,

the (r–1)'s complement of N is defined as:

$$(r^n - 1) - N$$

e.g., 6-digit base-10 (i.e., decimal) numbers:

9's complement is $(r^n - 1) - N = \mathbf{(10^6 - 1) - N} = 999999 - N$

9's complement of 546700 is $999999 - 546700 = 453299$

e.g., 7-digit base-2 (i.e., binary) numbers:

1's complement is $(r^n - 1) - N = \mathbf{(2^7 - 1) - N} = 1111111 - N$

1's complement of 1011000 is $1111111 - 1011000 = 0100111$

Observation:

Subtraction from $(r^n - 1)$ will never require a borrow

Diminished radix complement can be computed digit-by-digit

For binary: 1 – 0 = 1 and 1 – 1 = 0 as the above example illustrates

1's Complement (*Diminished Radix* Complement)

In order to take the 1's complement of a number

- replace each "0" with "1"
- replace each "1" with "0"

e.g., 1's complement of $(10110000)_2$ is $(01001111)_2$

If you add a number and its 1's complement you obtain all "1"s

e.g.,

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

Radix Complement (also known as r's Complement)

Given a number N in base r having n digits,
the r's complement of N is defined as:

$r^n - N$ for $N \neq 0$

0 for $N = 0$

e.g., 6-digit base-10 (i.e., decimal) numbers:

10's complement is $r^n - N = \mathbf{10^6 - N} = 1000000 - N$

10's complement of 546700 is $1000000 - 546700 = 453300$

e.g., 7-digit base-2 (i.e., binary) numbers:

2's complement is $r^n - N = \mathbf{2^7 - N} = 10000000 - N$

2's complement of 1011000 is $10000000 - 1011000 = 0101000$

Observation:

Comparing with the (r – 1) 's complement, note that
the r's complement of N is obtained by adding 1
to the (r – 1) 's complement of N, since $r^n - N = [(r^n - 1) - N] + 1$.
e.g., 453299 +1 = 453300 and 0100111 +1 = 0101000

2's Complement

In order to take the 2's complement of a number

- Take 1's complement of the number
- Add 1 to 1's complement of the number

e.g., 1's complement of $(10110000)_2$ is $(01001111)_2$

2's complement of $(10110000)_2$ is $(01001111)_2 + 1 = (01010000)_2$

If you add a number and its 2's complement you obtain 2^n
but if the number of bits is fixed to n, then you obtain 0.

e.g.,

$$\begin{array}{r} \mathbf{1\,0\,1\,1\,0\,0\,0\,0} \\ +\ \mathbf{0\,1\,0\,1\,0\,0\,0\,0} \\ \hline \mathbf{1\,0\,0\,0\,0\,0\,0\,0\,0} \quad = 2^8 \end{array}$$

$$\Downarrow$$

$$\mathbf{0\,0\,0\,0\,0\,0\,0\,0} \quad = 0$$

Subtraction of Unsigned Numbers with Complements

- **r's complement is used to convert subtraction to addition, which reduces hardware requirements (only adders are needed).**

A - B = A + (-B) (add r's complement of B to A)

- **r's complement has the properties of the minus sign**

A + (-A) = 0

A + r's complement of A = 0

- (- A) = A

r's complement of r's complement of A = A

A - B = A + (-B)

A - B = A + 2's complement of B

Subtraction of two *n*-digit unsigned numbers A – B in base *r* can be done as follows:

Add A to the r's complement of B which is $A + (r^n - B) = A - B + r^n$

If $A \geq B$

then adding A to the r's complement of B will produce an end carry
which can be discarded and the result is A – B

else adding A to the r's complement of B will not produce an end carry
and the result is $r^n - (B - A)$ which is r's complement of (B – A)
which can be normalized by
taking the r's complement of the result and placing a negative sign in front
(because $r^n - (r^n - (B - A)) = (B - A)$) and – (B – A) should be obtained.

Example 1.5

Using 10's complement, subtract 72532 – 3250.

	M =	72532
10's complement of	N =	+ 96750
	Sum =	169282
	Discard end carry 10^5 =	– 100000
	Answer =	69282

96749 + 1

There is end carry.

Example 1.6

Using 10's complement, subtract 3250 – 72532.

	M =	03250
10's complement of	N =	+ 27468
	Sum =	30718

27467 + 1

There is no end carry.

Therefore, the answer
= – (10's complement of 30718)
= – 69282.

Example 1.7

Given the two binary numbers $X = 1010100$ and $Y = 1000011$, perform the subtraction (a) $X - Y$; and (b) $Y - X$, by using 2's complement.

(a)		
	X =	1010100
	2's complement of Y =	+0111101
	Sum =	10010001
	Discard end carry 2^7 =	−10000000
	Answer. X − Y =	0010001

There is end carry.

(b)		
	Y =	1000011
	2's complement of X =	+ 0101100
	Sum =	1101111

There is no end carry.
Therefore, the answer
= – (2's complement of 1101111
= – 0010001.

Subtraction of unsigned numbers can also be done using (r – 1)'s complement. Remember that the (r – 1) 's complement is one less then the r's complement.

Example 1.8

Repeat Example 1.7, but this time using 1's complement.

(a) $X - Y = 1010100 - 1000011$	
X =	1010100
1's complement of Y =	± 0111100
Sum =	10010000
End-around carry =	+ 1
Answer. $X - Y$ =	0010001

There is end carry.

Therefore, add 1 to Sum to get the answer which is 0010001.

(b) $Y \quad X = 1000011 \quad 1010100$	
Y =	1000011
1's complement of X =	+ 0101011
Sum =	1101110

There is no end carry.
Therefore, the answer
= – (1's complement of 1101110)
= – 0010001.

Signed Binary Integers

- There are three representations of signed binary integers:

  A) SIGNED MAGNITUDE FORM (SMF)
  B) 1's COMPLEMENT FORM (1's CF)
  C) 2's COMPLEMENT FORM (2's CF)

- All of these three forms require that
  a number N is represented in a fixed word-length, say n-bit word.

- Let N be a signed binary integer in an n-bit word: $N= d_{n-1}\, d_{n-2}d_{n-3} \ldots d_2d_1d_0$
  Then, d_{n-1} is the sign bit and
  $d_{n-2}d_{n-3} \ldots d_2d_1d_0$ is the magnitude of N, i.e., **|N|**.

- The sign bit is 0 for positive and 1 for negative integers.

e.g., if n = 4, then N is represented in | d_3 | d_2 | d_1 | d_0 |
where d_3 is the sign bit and $d_2d_1d_0$ is the |N|

Representations of signed binary integers in an n-bit word

$N = d_{n-1}\, d_{n-2}d_{n-3} \ldots d_2d_1d_0$
where d_{n-1} represents the sign of N and
$d_{n-2}d_{n-3} \ldots d_2d_1d_0$ represents the |N|.

But, how do we determine $d_{n-2}d_{n-3} \ldots d_2d_1d_0$ for a given |N|?

A) **Signed Magnitude Form (SMF)** in n=4 bits There are two zeros, +0 and -0

$d_2d_1d_0$ is determined by $|N|= d_2x2^2 + d_1x2^1 + d_0\, x2^0$

e.g., N =+3 and N=-3 in SMF
$|3| = 0x2^2 + 1x2^1 + 1x2^0$
Thus,
+3 = 0011
- 3 = 1011

Representations of signed binary integers in an n-bit word

$N = d_{n-1}\, d_{n-2}d_{n-3} \ldots d_2d_1d_0$
where d_{n-1} represents the sign of N and
$d_{n-2}d_{n-3} \ldots d_2d_1d_0$ represents the |N|.

But, how do we determine $d_{n-2}d_{n-3} \ldots d_2d_1d_0$ for a given |N|?

B) 1's Complement Form (1's CF) in n=4 bits There are two zeros, +0 and -0

If $N \geq +0$, then $d_2d_1d_0$ is determined by $|N| = d_2x2^2 + d_1x2^1 + d_0x2^0$
If $N \leq -0$, then $d_2d_1d_0$ is determined by taking the 1's complement of
$d_2d_1d_0$ which was determined by $|N| = d_2x2^2 + d_1x2^1 + d_0x2^0$

e.g., N =+3 and N=-3 in 1's CF
$|3| = 0x2^2 + 1x2^1 + 1x2^0$ and 100 is 1's complement of 011
Thus,

+3 = 0011
- 3 = 1 100

Representations of signed binary integers in an n-bit word

$N = d_{n-1}\, d_{n-2}d_{n-3} \ldots d_2d_1d_0$
where d_{n-1} represents the sign of N and
$d_{n-2}d_{n-3} \ldots d_2d_1d_0$ represents the |N|.

But, how do we determine $d_{n-2}d_{n-3} \ldots d_2d_1d_0$ for a given |N|?

C) 2's Complement Form (2's CF) in n=4 bits There is one zero, +0

If $N \geq 0$, then $d_2d_1d_0$ is determined by $|N| = d_2x2^2 + d_1x2^1 + d_0x2^0$
If $N < 0$, then $d_2d_1d_0$ is determined by taking the 2's complement of
$d_2d_1d_0$ which was determined by $|N| = d_2x2^2 + d_1x2^1 + d_0x2^0$

e.g., N =+3 and N=-3 in 2's CF
$|3| = 0x2^2 + 1x2^1 + 1x2^0$ and 101 is 2's complement of 011
Thus,
+3 = 0011
-3 = 1 101

(exceptions: 0000 is zero and 1000 is $-(2^{n-1})$ by convention)

COMPARISON of SMF, 1's CF, and 2's CF

ASSUME 4 - BIT WORD LENGTH

4-Bit Stream	Unsigned	SMF	1's CF	2's CF
0000	0	+0	+0	0
0001	1	+1	+1	+1
0010	2	+2	+2	+2
0011	3	+3	+3	+3
0100	4	+4	+4	+4
0101	5	+5	+5	+5
0110	6	+6	+6	+6
0111	7	+7	+7	+7
1000	8	-0	-7	-8
1001	9	-1	-6	-7
1010	10	-2	-5	-6
1011	11	-3	-4	-5
1100	12	-4	-3	-4
1101	13	-5	-2	-3
1110	14	-6	-1	-2
1111	15	-7	-0	-1

There is one unique 0 in 2's CF.
There are -0 and +0 in both SMF and 1's CF.

In 2's CF, there are seven positive integers, eight negative integers, and one 0, making a total of 16 unique numbers.

In both SMF and 1's CF, there are seven positive integers, seven negative integers, and two 0's, making a total of 16 unique numbers.

SMF does not utilize complements and thus is more readable by humans.

1's CF is more suitable for logical ops.

2's CF is more suitable for arithmetic ops.

Table 1.3 lists all possible four-bit signed binary numbers in the three representations.

Table 1.3
Signed Binary Numbers

Decimal	Signed-2's Complement	Signed-1's Complement	Signed Magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
−0	—	1111	1000
−1	1111	1110	1001
−2	1110	1101	1010
−3	1101	1100	1011
−4	1100	1011	1100
−5	1011	1010	1101
−6	1010	1001	1110
−7	1001	1000	1111
−8	1000	—	—

ADDITION IN 2's CF

sum = augend + addend

where augend and addend are in 2's CF

- NO SPECIAL TREATMENT OF THE SIGN BIT
-DISCARD THE END CARRY

e.g., ASSUME 4 - BIT WORD LENGTH

```
  (+5)     0101
+ (+2)     0010
  (+7)     0111

  (+5)     0101
+ (- 2)    1110
  (+3)  1  0011      Discard the end carry
        end carry
  (- 5)    1011
+ (+2)     0010
  (- 3)    1101

  (- 5)    1011
+ (- 2)    1110
  (- 7)  1 1001      Discard the end carry
        end carry
```


Note that in 2's CF,
the range of signed binary integers that can be represented in an n-bit word is
$-(2^{n-1}) \le N \le +(2^{n-1}-1)$.
e.g., if n =4, then $-(2^3) \le N \le +(2^3-1)$, i.e., $-8 \le N \le +7$

OVERFLOW in ADDITION

When the sign bits of both operands are the same,
an overflow occurs if
the sign bit of the result is not the same as the common sign bits of the operands
(regardless whether end carry results or not).
In other words: an overflow occurs when: *The addition of two positive numbers results in a negative number* or
The addition of two negative numbers results in a positive number.
The overflow indicates the fact that the result cannot be represented in n-bits!

```
e.g.,    (+5)   0101
       + (+7)   0111
         (?)    1100   OVERFLOW (+12 cannot be represented in 4-bit 2's CF)

         (- 3)  1101
       + (- 7)  1001
         (?)  1 0110   OVERFLOW (-10 cannot be represented in 4-bit 2's CF)
```

SUBTRACTION IN 2's CF

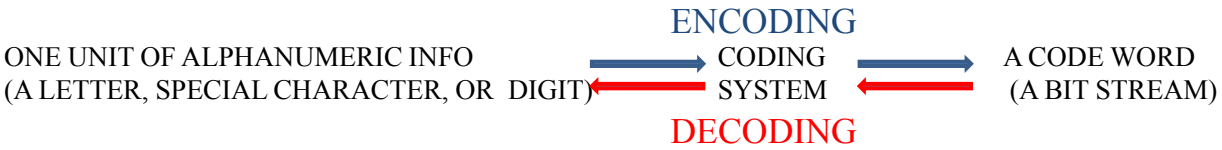
difference = minuend - subtrahend
where minuend and subtrahend are in 2's CF
SUBTRACTION IS PERFORMED BY ADDITION

difference = minuend + (- subtrahend)
where (- subtrahend) is 2's complement of subtrahend

e.g., ASSUME 4 - BIT WORD LENGTH

0101 (+5)	(+5)	0101
0010 - (+2)	+ (- 2)	+ 1110
	(+3)	0011
0101 (+5)	(+5)	0101
1110 - (- 2)	+ (+2)	+ 0010
	(+7)	0111
1011 (- 5)	(- 5)	1011
0010 - (+2)	+ (- 2)	+ 1110
	(- 7)	1001
1011 (- 5)	(- 5)	1011
1110 - (- 2)	+ (+2)	+ 0010
	(- 3)	1101

BINARY CODES
(REPRESENTATION OF ALPHANUMERIC INFO IN BINARY FORM)



BINARY CODES FOR DECIMAL DIGITS

WEIGHTED CODES

- ARE POSITIONALLY WEIGHTED
- e.g., **BCD** : BINARY CODED DECIMAL (OR **8421** CODE)

The code word for the decimal digit **D** in BCD is obtained by
D = W4 x B4 + W3 x B3 + W2 x B2 + W1 x B1
where W4, W3, W2, and W1 are weights and (B4 B3 B2 B1) = code word for **D**
Weights are: 8, 4, 2, and 1 from MSB to LSB (hence, it is called 8-4-2-1 code).

NON-WEIGHTED CODES

- ARE <u>NOT</u> POSITIONALLY WEIGHTED
- e.g., **XS3** : EXCESS-3 CODE

The code word for the decimal digit **D** in XS3 =
The code word for the decimal digit **D** in BCD + 0011
e.g. What is the code word for decimal digit 4 in XS3?

$0111 = 0100 + 0011$
$(4)_{XS3} \quad (4)_{BCD} \quad (3)_2$

SOME BINARY CODES

DECIMAL

DIGIT	8421	XS3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

OCTAL

DIGIT	421
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

BINARY CODES FOR CHARACTERS

ASCII
(American Standard Code for Information Interchange)

It is the most widely used character code.

Table 1.7
American Standard Code for Information Interchange (ASCII)

$b_4b_3b_2b_1$	$b_7b_6b_5$ = 000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	−	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

e.g., ASCII code representation of the word 'Digital'

	Binary	Hex
D	1000100	44
i	1101001	69
g	1100111	67
i	1101001	69
t	1110100	74
a	1100001	61
l	1101100	6C

ASCII Character Code

It uses 7-bits to represent:

94 Graphic printing characters.

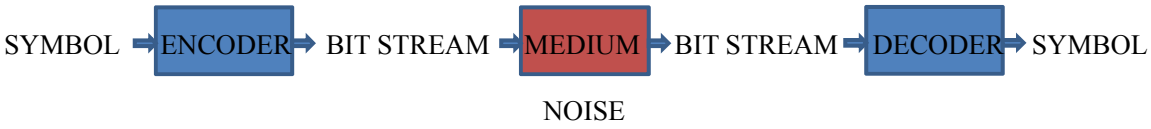
34 Non-printing characters.

Some non-printing characters are used for text formatting (e.g. BS = Backspace, CR = carriage return)
Other non-printing characters are used for record marking and flow control
(e.g. STX and ETX= start and end text areas).

Control Characters

NUL	Null	DLE	Data-link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End-of-transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete

ENCODING/DECODING



- A CODE C IS A COLLECTION OF CODE WORDS, C0, C1, ..., Cq IN WHICH
  THERE IS A CODE WORD FOR EACH SYMBOL S0, S1, ..., Sq IN THE SOURCE ALPHABET

- # OF BITS REQUIRED TO REPRESENT EACH SYMBOL DEPENDS ON

  * FREQUENCY OF OCCURRENCE OF THE SYMBOL IN AVERAGE INFORMATION
    IF S0, S1, ..., Sq ARE NOT EQUALLY LIKELY TO OCCUR ⟹ VARIABLE LENGTH CODE
    IF S0, S1, ..., Sq ARE EQUALLY LIKELY TO OCCUR ⟹ FIXED LENGTH CODE

  * PROBABILITY OF SINGLE, DOUBLE, TRIPLE, ... ERRORS OCCURRING IN A PARTICULAR MEDIUM THROUGH WHICH CODE WORDS WILL BE TRANSMITTED/STORED

  * WHETHER ERRORS IN A RECEIVED/RETRIEVED SEQUENCE SHOULD BE
    DETECTED ONLY ⟹ **ERROR DETECTING CODE**
    OR
    CORRECTED ⟹ **ERROR CORRECTING CODE**

Code words in these codes contain redundancy in terms of additional bits which are called “Parity Bits”.

A parity bit is an extra bit appended in a code word to make the total number of 1's either even or odd in the code word.

A code word has even parity if the number of 1's in the code word is even.
A code word has odd parity if the number of 1's in the code word is odd.

OCTAL DIGIT	421	EVEN PARITY	ODD PARITY
0	000	0000	0001
1	001	0011	0010
2	010	0101	0100
3	011	0110	0111
4	100	1001	1000
5	101	1010	1011
6	110	1100	1101
7	111	1111	1110

DECIMAL DIGIT	8421	EVEN PARITY	ODD PARITY
0	0000	00000	00001
1	0001	00011	00010
2	0010	00101	00100
3	0011	00110	00111
4	0100	01001	01000
5	0101	01010	01011
6	0110	01100	01101
7	0111	01111	01110
8	1000	10001	10000
9	1001	10010	10011

TRANSMISSION OF OCTAL DIGITS USING ODD PARITY

SENDER
- has $(6)_8 = (110)_2$ to send
- performs "parity generation" to find out the value of the parity bit, P
  i.e., P = 1 (parity generation yields 1)
- appends the parity bit P to $(110)_2$ and sends $(1101)_2$

RECEIVER
- receives $(1101)_2$
- performs "parity checking" to find out if the #of 1's is odd in $(1101)_2$
  i.e., Check = 0 (parity check yields 0 meaning no parity error)
- since there is no parity error, it drops the parity bit to obtain $(110)_2$

* **OTHER CODES**
  HUFFMANN CODES
  GRAY CODES
  CYCLIC REDUNDANCY CODES, etc.

SO FAR,

We have seen that discrete information is either numeric or alphanumeric .

We've seen
-that numeric information is represented in Binary number system
- how numeric information in other number systems
such as Octal, Decimal, Hexadecimal are converted into Binary number system
- how some arithmetic operations are performed in Binary number system

We've also seen
- that alphanumeric information is represented in Binary coding systems
such as BCD, XS3, ASCII
- concepts of encoding, decoding, parity, error detecting codes, error correcting codes

Now, we will start to look into discrete information processing which
*** is expressed in a two valued Boolean Algebra as Boolean functions**
- **Boolean Expressions (e)**
- **Truth Tables (TT)**
- **Conversion between e and TT**

*** is realized by means of Logic Circuits**
- **Combinatorial circuits**
- **Sequential circuits**

Binary Storage and Registers

Registers

A binary cell is a device that possesses
two stable states and is capable of storing one of the two states
i.e., on/off, in other words, one bit of information : 0/1 .
e.g., flip-flop

A register is a group of binary cells.
A register with *n* cells can store any discrete quantity of information
that contains *n* bits.
e.g., n cells can store any one of 2^n combinations of n bits.

Register Transfer

A transfer of the information stored in one register to another register.
One of the major operations in a digital system.

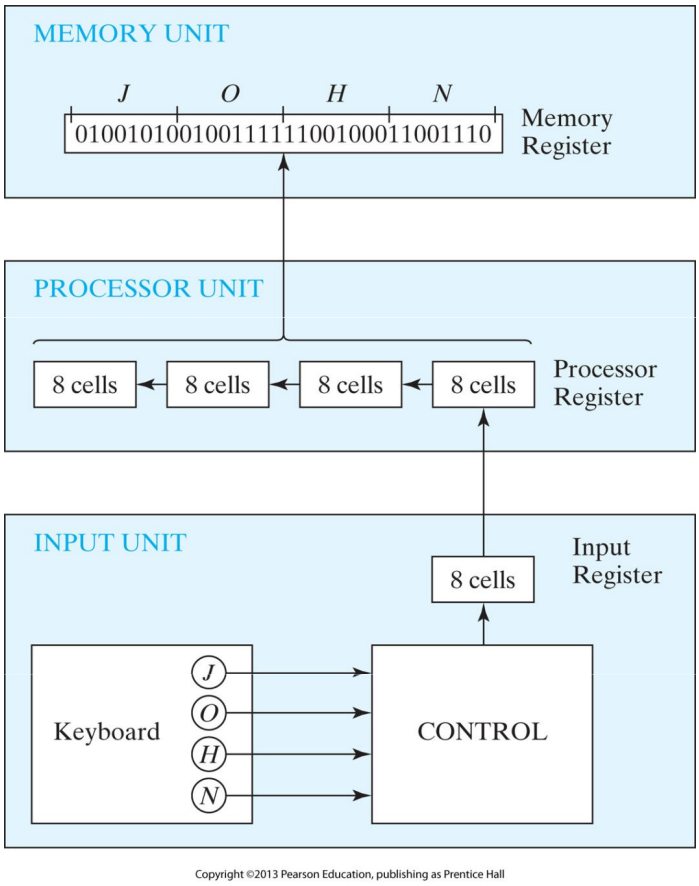
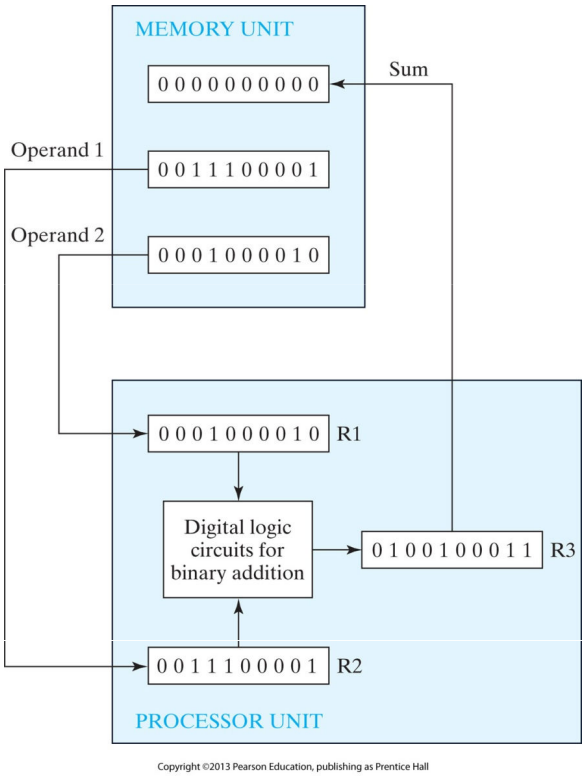


Figure 1.1 Transfer of information among registers



This example shows circuits that manipulate bits of information in a Load/Store machine where the following is executed:

Load	R2
Load	R1
Add	R1, R2, R3
Store	R3

Figure 1.2 Example of binary information processing

A Brief Introduction to Boolean Algebra and Logic gates

Boolean Algebra
deals with

1- Boolean Variables

Designated by letters of the alphabet such as A, B, C, x, y, z, etc.

Each variable can have two and only two distinct values:

{True, False}

However, we will use 1 for True and 0 for False.

2- Boolean Operations

There are three basic logical operations: AND, OR, and NOT

Basic Boolean Operations- ***AND operation***

- Represented by a dot or by the absence of an operator.
  (***Don't confuse this with binary multiplication operation)***

e.g., x.y =z
xy =z

- Read : x AND y is equal to z
- Interpretation: z = 1 when both x = 1 and y = 1. Otherwise z = 0.

Truth Table for AND operation:

Truth table gives the value of z for all possible values of x and y

x	y	x.y
0	0	0
0	1	0
1	0	0
1	1	1

Basic Boolean Operations- *OR operation*

- Represented by a plus sign.

(Don't confuse this with binary addition operation)

e.g., x+y =z

- Read : x OR y is equal to z
- Interpretation: z = 0 when both x =0 and y = 0. Otherwise z = 1.

Truth Table for OR operation:

Truth table gives the value of z for all possible values of x and y

x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	1

Basic Boolean Operations- *NOT operation*

- Represented by a prime or an overbar.

e.g., x'

- Read : complement of x is equal to z
- Interpretation: z = 1 when x =0. z = 0 when x = 1.

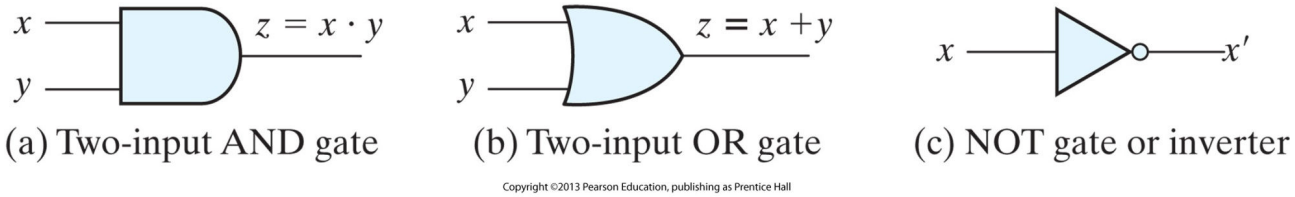
Truth Table for OR operation:

Truth table gives the value of z for all possible values of x

x	x'
0	1
1	0

Logic gates

Logic gates are electronic circuits
that operate on one or more input signal to produce an output signal.



(a) Two-input AND gate (b) Two-input OR gate (c) NOT gate or inverter

Fig. 1.4 Symbols for digital logic circuits

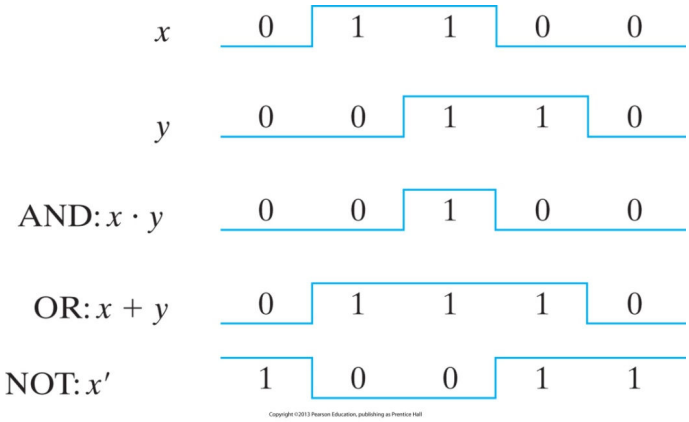
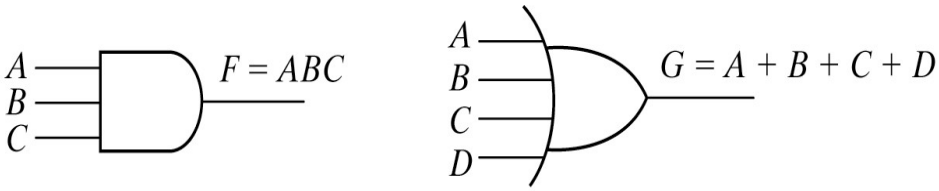


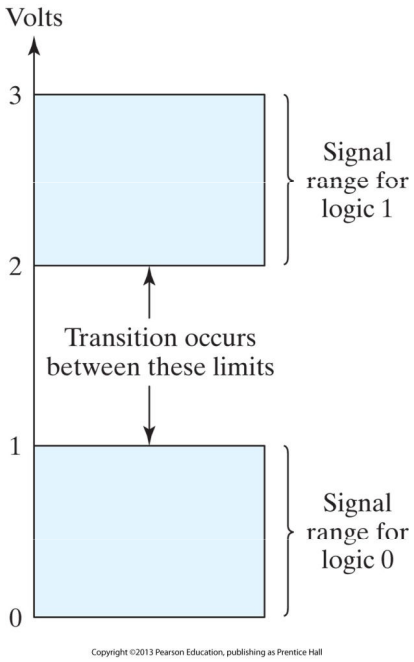
Fig. 1.5 Input-Output signals for gates

Gates with Multiple Inputs

AND and OR
gates may have more than
2 input signals
(we will discuss why this is possible later)



(a) Three-input AND gate (b) Four-input OR gate



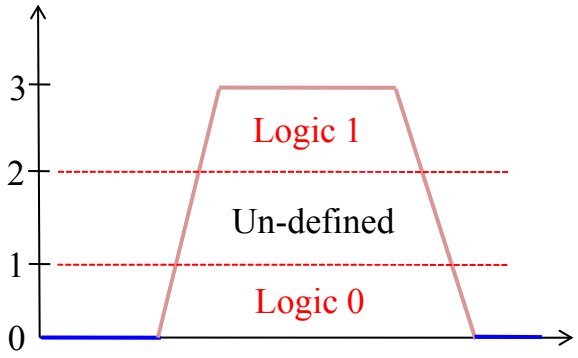


Figure 1.3 Signal levels for Binary signals