UNIT-I

Introduction to Digital Computer

- A computer consists of five functionally independent main parts: input, memory, arithmetic and logic, output, and control units, as shown in Figure 1.

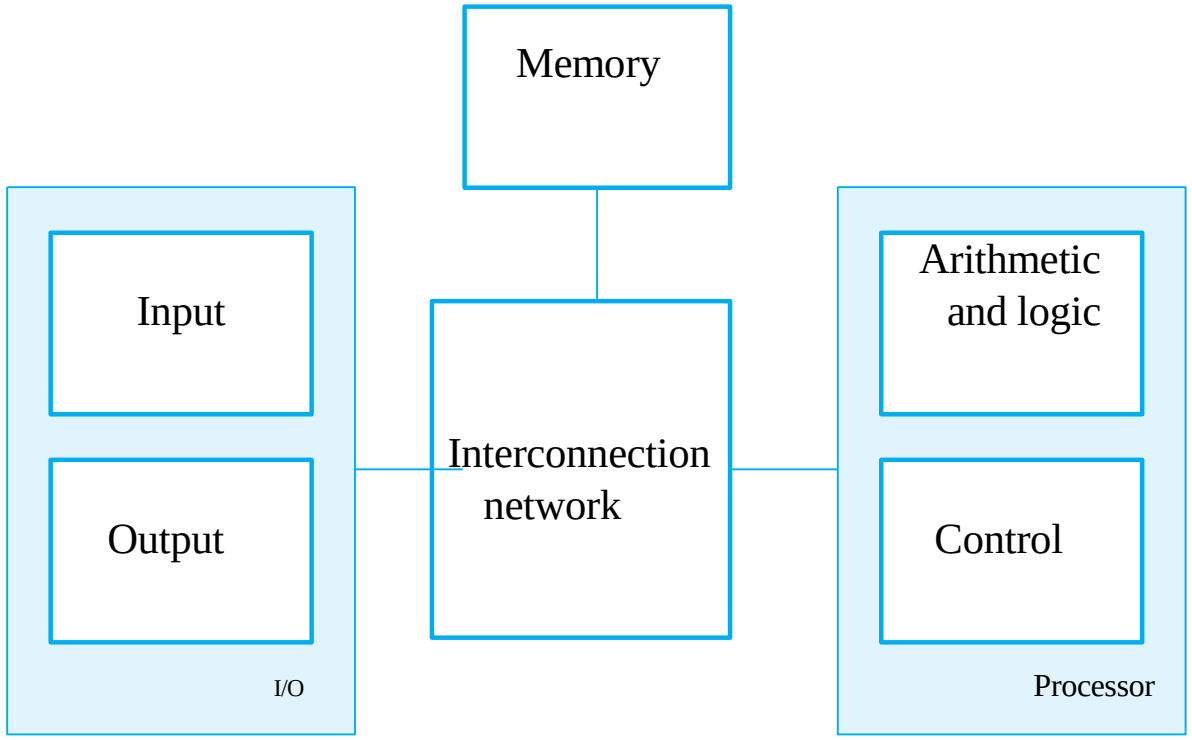


Figure 1

Input Unit

- Computers accept coded information through input units. The most common input device is the keyboard. Whenever a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted to the processor.
- Many other kinds of input devices for human-computer interaction are available, including the touchpad, mouse, joystick, and trackball. Microphones can be used to capture audio input which is then sampled and converted into digital codes for storage and processing. Similarly, cameras can be used to capture video input.

Memory Unit

- The function of the memory unit is to store programs and data. There are two classes of storage, called primary and secondary.
- Primary Memory
  - Primary memory, also called main memory, is a fast memory that operates at electronic speeds. Programs must be stored in this memory while they are being executed.
  - The memory consists of a large number of semiconductor storage cells, each capable of storing one bit of information.
  - Instructions and data can be written into or read from the memory under the control of the processor.
  - A memory in which any location can be accessed in a short and fixed amount of time after specifying its address is called a random-access memory (RAM). The time required to access one word is called the memory access time. This time is independent of the location of the word being accessed. It typically ranges from a few nanoseconds (ns) to about 100 ns for current RAM units.

Memory Unit Contd…

- Cache Memory
  - As an adjunct to the main memory, a smaller, faster RAM unit, called a cache, is used to hold sections of a program that are currently being executed, along with any associated data. The cache is tightly coupled with the processor and is usually contained on the same integrated-circuit chip. The purpose of the cache is to facilitate high instruction execution rates.
  - At the start of program execution, the cache is empty. All program instructions and any required data are stored in the main memory. As execution proceeds, instructions are fetched into the processor chip, and a copy of each is placed in the cache. When the execution of an instruction requires data located in the main memory, the data are fetched and copies are also placed in the cache.

Memory Unit Contd…

- Secondary Storage
  - Although primary memory is essential, it tends to be expensive and does not retain information when power is turned off. Thus additional, less expensive, permanent secondary storage is used when large amounts of data and many programs have to be stored, particularly for information that is accessed infrequently.
  - Access times for secondary storage are longer than for primary memory. A wide selection of secondary storage devices is available, including magnetic disks, optical disks (DVD and CD), and flash memory devices.

Arithmetic and Logic Unit

- Most computer operations are executed in the arithmetic and logic unit (ALU) of the processor. Any arithmetic or logic operation, such as addition, subtraction, multiplication, division, or comparison of numbers, is initiated by bringing the required operands into the processor, where the operation is performed by the ALU. For example, if two numbers located in the memory are to be added, they are brought into the processor, and the addition is carried out by the ALU.
- When operands are brought into the processor, they are stored in high-speed storage elements called **registers**. Each register can store one word of data. Access times to registers are even shorter than access times to the cache unit on the processor chip.

Output Unit

- The output unit is the counterpart of the input unit. Its function is to send processed results to the outside world. A familiar example of such a device is a printer. Most printers employ either photocopying techniques, as in laser printers, or ink jet streams.

Control Unit

- The memory, arithmetic and logic, and I/O units store and process information and perform input and output operations. The operation of these units must be coordinated in some way. This is the responsibility of the control unit. The control unit is effectively the nerve center that sends control signals to other units and senses their states.
- Data transfers between the processor and the memory are also managed by the control unit through timing signals.

Bus Structure

- The bus shown in Figure 2 is a simple structure that implements the interconnection network. Only one source/destination pair of units can use this bus to transfer data at any one time.

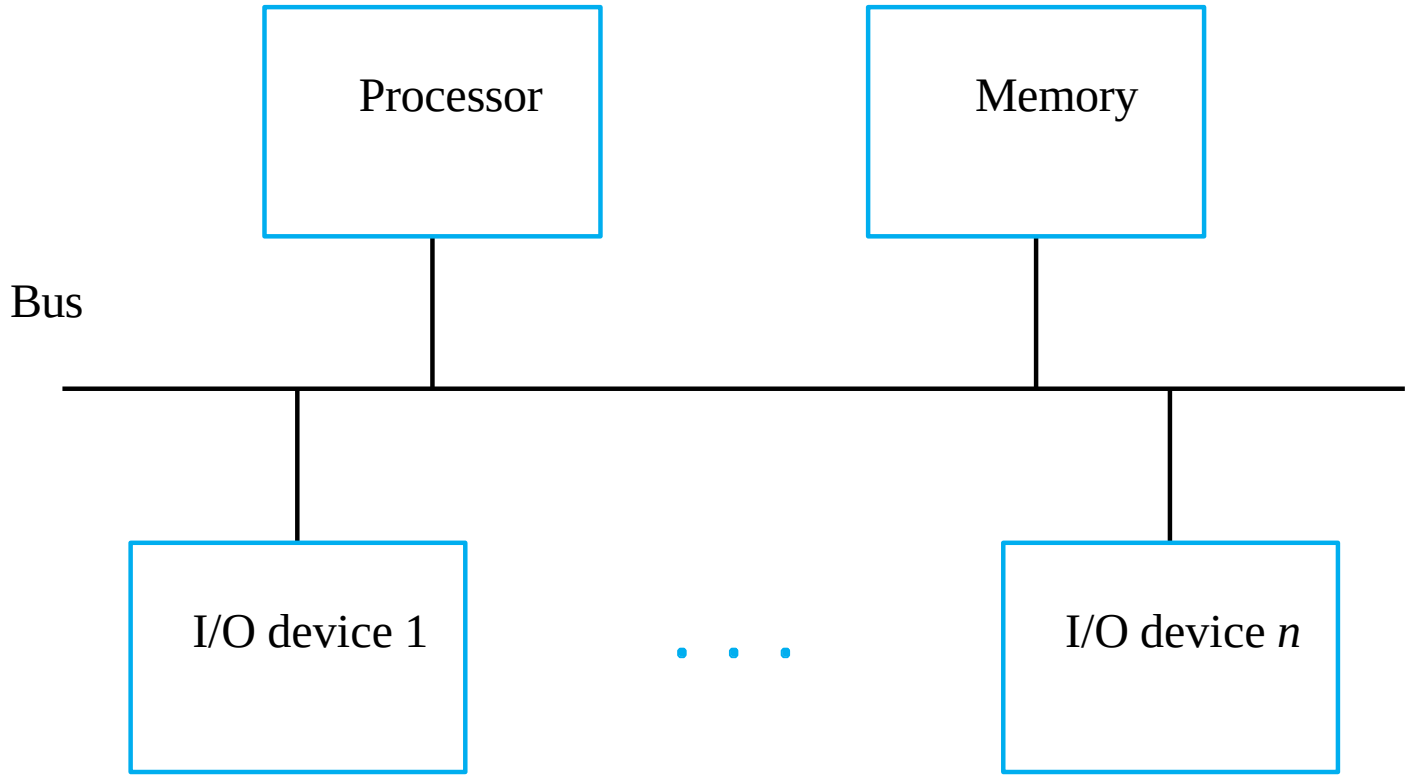


Figure 2

I/O Interface for an input device

- The bus consists of three sets of lines used to carry address, data, and control signals. I/O device interfaces are connected to these lines, as shown in Figure 3 for an input device.
- Each I/O device is assigned a unique set of addresses for the registers in its interface.
- When the processor places a particular address on the address lines, it is examined by the address decoders of all devices on the bus.
- The device that recognizes this address responds to the commands issued on the control lines. The processor uses the control lines to request either a Read or a Write operation, and the requested data are transferred over the data lines.

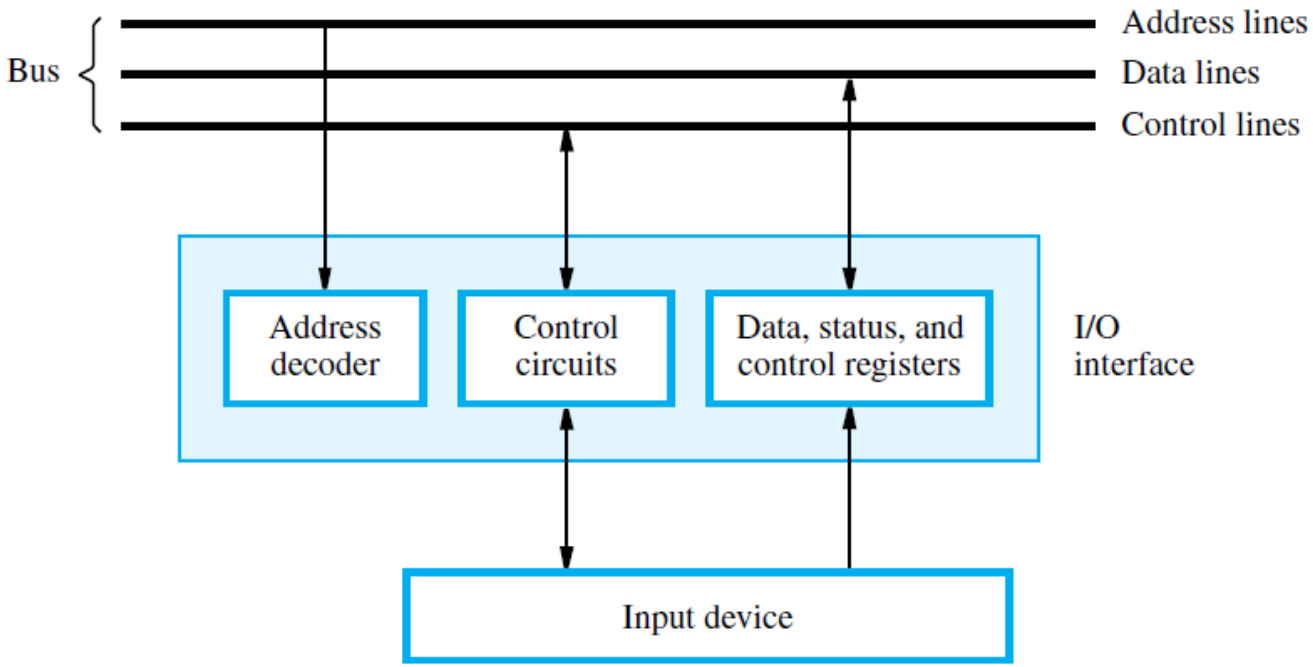
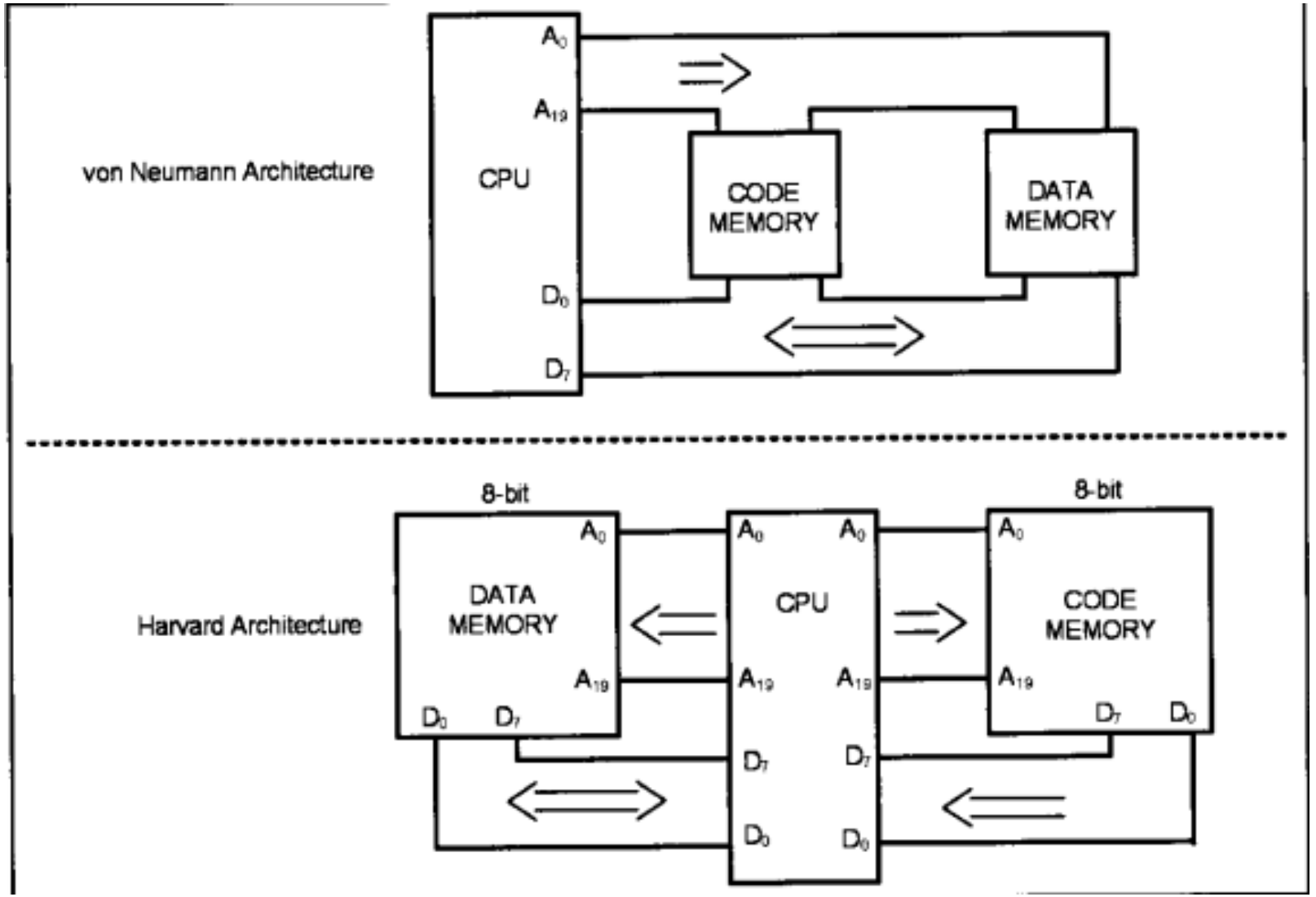


Figure 3

Computer Organization and Architecture



Von Neumann Architecture	Harvard Architecture
It was developed at Princeton University	It was developed at Harvard University
Same physical memory address is used for instructions and data.	Separate physical memory address is used for instructions and data.
There is common bus for data and instruction transfer.	Separate buses are used for transferring data and instruction. There are in all four buses: (a) Data bus for carrying data in and out of CPU (b) Address bus for accessing the data (c) Data bus for carrying code into the CPU (d) Address bus for accessing the code.
Since same buses are used to access data and code which cause them to get in each other's way thus slow down the processing speed of CPU because each have to wait for the other to finish fetching.	Have good speed of program execution.
It is cheaper in cost	It is costly than von Neumann Architecture
It is used in personal computers and small computers	It is used in micro controllers and signal processing

There are four types of number systems. They are

1. Decimal number system

2. Binary number system

3. Octal number system

4. Hexadecimal number system

1. Decimal number system

- The decimal number system contain ten unique digits: **0,1,2,3,4,5,6,7,8 and 9**.

- In decimal system 10 symbols (0-9) are involved, so the base or radix is 10. e.g. $(1245)_{10}$

- The value attached to the symbol depends on its location with respect to the decimal point.

Example: Place value of Decimal Number

In general,

(MSB) dn dn-1 dn-2 …… d0 **.** d -1 d -2 ……d – m **(LSB)**

is given by

(dn x 10n) + (dn-1 x 10n-1) + (dn-2 x 10n-2) + … + (d0 x 100) + (d-1 x 10 -1) + (d-2 x 10 -2) +…+(d -m x 10 –m)

For example:-1 Decimal number- 9256.26

9256.26 = 9 x 1000 + 2 x 100 + 5 x 10 + 6 x 1 + 2 x (1/10) + 6 x (1/100)

$= 9 \times 10^3 + 2 \times 10^2 + 5 \times 10^1 + 6 \times 10^0 + 2 \times 10^{-1} + 6 \times 10^{-2}$

2. Binary number system

- The binary number system is a positional weighted system.

- The base or radix of this number system is 2.

e.g. $\mathbf{(10101)_2}$

- The symbols used are 0 and 1.

- A single binary digit is called a bit.

3. Octal number system

- It is also a positional weighted system.
- Its base or radix is 8.
- It has 8 independent symbols- 0,1,2,3,4,5,6 and 7.
- Its base 8 = 2^3 (2 to the power 3) , every 3-bit group of binary can be represented by an octal digit. e.g.- $(14623)_8$ and $(277.13)_8$

4. Hexa-Decimal number system

- It is also a positional weighted system.
- Its base or radix is 16.
- It has 15 independent symbols- 0,1,2,3,4,5,6,7,9 and 10(A), 11(B),12(C),13(D),14(E),15(F).

Or 0,1,2,3,4,5,6,7,8,9 and 10- A, 11- B, 12- C, 13-D, 14- E and 15-F

- e.g.- $(14ABF)_{16}$ and $(12.CD)_{16}$

CONVERSION FROM ONE NUMBER SYSTEM TO ANOTHER

- Binary number conversion to-

1. Octal
2. Decimal
3. Hexa-Decimal

- OCTAL number conversion to-

1. Binary
2. Decimal
3. Hexa-Decimal

CONVERSION FROM ONE NUMBER SYSTEM TO ANOTHER

- Decimal number conversion to-

1. Binary
2. Octal
3. Hexa-Decimal

- Hexa decimal number conversion to-

1. Binary
2. Octal
3. Decimal

Total 12 types to convert the number system into each other

BINARY NUMBER SYSTEM: Binary to Decimal

- In this method, each binary digit of the number is multiplied by its positional weight and the product terms are added to obtain decimal number

Example: 1

(i) Convert $(10101)_2$ to decimal.

Solution :

(Positional weight) $2^4\ 2^3\ 2^2\ 2^1\ 2^0$

Binary number 10101

$= (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$

$= 16 + 0 + 4 + 0 + 1$

$= (21)_{10}$

Example-2

(ii) Convert $(111.101)_2$ to decimal.

Solution:

$$(111.101)_2 = (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3})$$
$$= 4 + 2 + 1 + 0.5 + 0 + 0.125$$
$$= (7.625)_{10}$$

Example-3

Example 26.3. *Convert the binary number 110001 to its equivalent decimal number.*

Solution. The binary number along with its decimal values of various positions is shown.

$\therefore \quad (110001)_2 = 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^0$

$= 32 + 16 + 1 = 49$

1	1	0	0	0	1
2^5	2^4	2^3	2^2	2^1	2^0

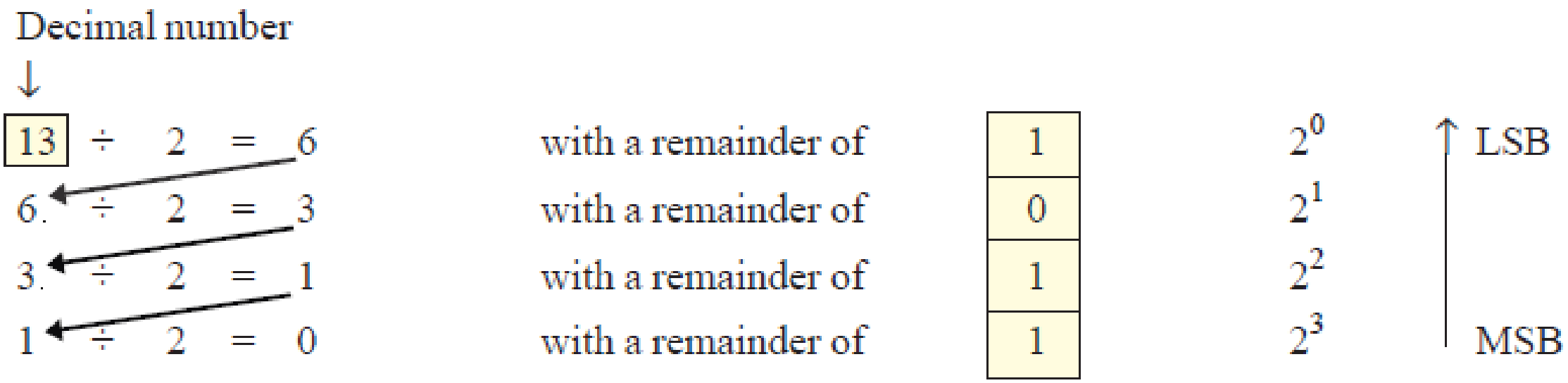
or $\quad (110001)_2 = (49)_{10}$

Decimal to Binary Number

Divide progressively the decimal number by 2 and write down the remainder after each division. Continue this process till you get a quotient of 0 and remainder of 1, the conversion is now complete. The remainders, taken in reverse order, form the binary number [See Fig. 26.4].

Note that 13 is first divided by 2, giving a quotient of 6 with a remainder of 1. This remainder becomes the 2^0 position in the binary number. The 6 is then divided by 2, giving a quotient of 3 with a remainder of 0. This remainder becomes the 2^1 position in the binary number.

Continuing this procedure, the equivalent binary number is 1101.



Example-1

Example 26.1. *Convert the decimal number 37 to its equivalent binary number.*

Solution. Using double-dabble method, we find that the equivalent binary number is 100101. It is a usual practice to mention the base of the number system. The decimal system has a base of 10 while binary system has a base of 2.

$$\therefore \qquad (37)_{10} = (100101)_2$$

Note. This notation avoids the confusion that may arise because decimal number also involves the digits 0 and 1. Thus, $(101)_{10}$ denotes the decimal number hundred one while the binary number $(101)_2$ is equivalent to decimal number 5.

2	37
2	18 – 1
2	9 – 0
2	4 – 1
2	2 – 0
2	1 – 0
	0 – 1

Example-2

Convert $(52)_{10}$ into binary.

2	52	
2	26	— 0
2	13	— 0
2	6	— 1
2	3	— 0
2	1	— 1
	0	— 1

Result of $(52)_{10}$ is **$(110100)_2$**

Example-3

(ii) Convert $(105.15)_{10}$ into binary.

Solution:

Integer part

2 | 105		
2 | 52	—	1
2 | 26	—	0
2 | 13	—	0
2 | 6	—	1
2 | 3	—	0
2 | 1	—	1
0	—	1

Fraction part

0.15 x 2 = 0.30

0.30 x 2 = 0.60

0.60 x 2 = 1.20

0.20 x 2 = 0.40

0.40 x 2 = 0.80

0.80 x 2 = 1.60

Result of $(105.15)_{10}$ is $\mathbf{(1101001.001001)_2}$

Binary to Octal conversion:-

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

For conversion binary to octal the binary numbers are divided into groups of 3 bits each, starting at the binary point and proceeding towards left and right.

Example-1 **Convert $(101111010110.110110011)_2$ into octal.**

Solution :

Group of 3 bits are	101	111	010	110	.	110	110	011
Convert each group into octal =	5	7	2	6	.	6	6	3

The result is **$(5726.663)_8$**

Example-2

Convert $(10101111001.0111)_2$ into octal.

Solution :

Binary number		10	101	111	001	.	011	1
Group of 3 bits are	=	010	101	111	001	.	011	100
Convert each group into octal	=	2	5	7	1	.	3	4

The result is **$(2571.34)_8$**

Binary to Hexadecimal conversion:-

Hexadecimal	Binary	Hexadecimal	Binary
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

Example-1

(i) Convert $(1011011011)_2$ into hexadecimal.

Solution:

Given Binary number		10	1101	1011
Group of 4 bits are		0010	1101	1011
Convert each group into hex	=	2	D	B

The result is $(\mathbf{2DB})_{16}$

Example-2

(ii) Convert $(01011111011.011111)_2$ into hexadecimal.

Solution:

Given Binary number		010	1111	1011	.	0111	11
Group of 3 bits are	=	0010	1111	1011	.	0111	1100
Convert each group into octal	=	2	F	B	.	7	C

The result is $(\mathbf{2FB.7C})_{16}$

Decimal to binary conversion:-

(ii) Convert $(105.15)_{10}$ into binary.

Solution:

Integer part		Fraction part
2 | 105		0.15 x 2 = 0.30
2 | 52	— 1	0.30 x 2 = 0.60
2 | 26	— 0	0.60 x 2 = 1.20
2 | 13	— 0	0.20 x 2 = 0.40
2 | 6	— 1	0.40 x 2 = 0.80
2 | 3	— 0	0.80 x 2 = 1.60
2 | 1	— 1	
0	— 1	

Result of $(105.15)_{10}$ is **$(1101001.001001)_2$**

Decimal to octal conversion:-

To convert the given decimal integer number to octal, successively divide the given number by 8 till the quotient is 0. To convert the given decimal fractions to octal successively multiply the decimal fraction and the subsequent decimal fractions by 8 till the product is 0 or till the required accuracy is obtained.

Example-1

(i) Convert $(378.93)_{10}$ into octal.

Solution:

8 | 378		0.93 x 8 = 7.44
8 | 47	— 2	0.44 x 8 = 3.52
8 | 5	— 7	0.52 x 8 = 4.16
0	— 5	0.16 x 8 = 1.28

Result of $(378.93)_{10}$ is $\mathbf{(572.7341)_8}$

Decimal to hexadecimal conversion:-

The decimal to hexadecimal conversion is same as octal.

Example-1

(i) Convert $(2598.675)_{10}$ into hexadecimal.

Solution:

	Remainder Decimal	Hex		Hex
16 | 2598			0.675 x 16 = 10.8	A
16 | 162	— 6	6	0.800 x 16 = 12.8	C
16 | 10	— 2	2	0.800 x 16 = 12.8	C
0	— 10	A	0.800 x 16 = 12.8	C

Result of $(2598.675)_{10}$ is **$(A26.ACCC)_{16}$**

Octal to binary conversion:

To convert a given a octal number to binary, replace each octal digit by its 3- bit binary equivalent.

Example-1

Convert $(367.52)_8$ into binary.

Solution:

Given Octal number is		3	6	7	.	5	2
Convert each group octal to binary	=	011	110	111	.	101	010

Result of $(367.52)_8$ is **$(011110111.101010)_2$**

Octal to decimal conversion:-

For conversion octal to decimal number, multiply each digit in the octal number by the weight of its position and add all the product terms

Example-1

Convert $(4057.06)_8$ to decimal

Solution:

$$(4057.06)_8 = 4 \times 8^3 + 0 \times 8^2 + 5 \times 8^1 + 7 \times 8^0 + 0 \times 8^{-1} + 6 \times 8^{-2}$$
$$= 2048 + 0 + 40 + 7 + 0 + 0.0937$$
$$= (2095.0937)_{10}$$

Result is **$(2095.0937)_{10}$**

Octal to hexadecimal conversion:-

For conversion of octal to Hexadecimal, first convert the given octal number to binary and then binary number to hexadecimal.

Example-1

Convert $(756.603)_8$ to hexadecimal.

Solution :-

Given octal no.		7	5	6	.	6	0	3
Convert each octal digit to binary	=	111	101	110	.	110	000	011
Group of 4bits are	=	0001	1110	1110	.	1100	0001	1000
Convert 4 bits group to hex.	=	1	E	E	.	C	1	8

Result is $(\mathbf{1EE.C18})_{16}$

HEXADECIMAL NUMBER SYSTEM :-

Hexadecimal to binary conversion:-

For conversion of hexadecimal to binary, replace hexadecimal digit by its 4 bit binary group.

Example-1

Convert $(3A9E.B0D)_{16}$ into binary.

Solution:

Given Hexadecimal number is		3	A	9	E	.	B	0	D
Convert each hexadecimal digit to 4 bit binary	=	0011	1010	1001	1110	.	1011	0000	1101

Result of $(3A9E.B0D)_8$ is **$(0011101010011110.101100001101)_2$**

Hexadecimal to decimal conversion:-

For conversion of hexadecimal to decimal, multiply each digit in the hexadecimal number by its position weight and add all those product terms.

Example-1

Convert $(A0F9.0EB)_{16}$ to decimal

Solution:

$$(A0F9.0EB)_{16} = (10 \times 16^3)+(0 \times 16^2)+(15 \times 16^1)+(9 \times 16^0)+(0 \times 16^{-1})+(14 \times 16^{-2})+(11 \times 16^{-3})$$
$$= 40960 + 0 + 240 + 9 + 0 + 0.0546 + 0.0026$$
$$= (41209.0572)_{10}$$

Result is **$(41209.0572)_{10}$**

Hexadecimal to Octal conversion:-

For conversion of hexadecimal to octal, first convert the given hexadecimal number to binary and then binary number to octal.

Example-1

Convert $(B9F.AE)_{16}$ to octal.

Solution :-

Given hexadecimal no.is		B		9		F	.	A		E	
Convert each hex. digit to binary	=	1011		1001		1111	.	1010		1110	
Group of 3 bits are	=	101	110	011	111		.	101	011	100	
Convert 3 bits group to octal.	=	5	6	3	7		.	5	3	4	

Result is $(\mathbf{5637.534})_8$

Complements

- Complements are used in digital computers for simplifying the subtraction operation and for logical manipulation.
- There are two types of complements for each base r system: the r ’s complement and the (r — l)’s complement.
- When the value of the base r is substituted in the name, the two types are referred to as the 2’s and l’s complement for binary numbers and the 10’s and 9’s complement for decimal numbers.

(r — l)’s Complement

- Given a number N in base r having n digits, the (r — l)’s complement of N is defined as (r^n — 1) — N.
- 9’s Complement
  - For decimal numbers r = 10 and r — 1 = 9, so the 9’s complement of N is (10n — 1) — N. Now, 10n represents a number that consists of a single 1 followed by n 0’s. 10n — 1 is a number represented by n 9’s.
  - For example, with n = 4 we have 104 = 10000 and 104 — 1 = 9999. It follows that the 9’s complement of a decimal number is obtained by subtracting each digit from 9.
  - For example, the 9’s complement of 546700 is 999999 — 546700 = 453299 and the 9’s complement of 12389 is 99999 — 12389 = 87610.

1’s Complement

- For binary numbers, r = 2 and r — 1 = 1, so the 1’s complement of N is (2^n —1) — N. Again, 2^n is represented by a binary number that consists of a 1 followed by n 0’s. 2^n — 1 is a binary number represented by n 1’s. For example, with n = 4, we have 24 = $(10000)_2$ and 24 — 1 = $(1111)_2$. Thus the 1’s complement of a binary number is obtained by subtracting each digit from 1.
- However, the subtraction of a binary digit from 1 causes the bit to change from 0 to 1 or from 1 to 0. Therefore, the 1’s complement of a binary number is formed by changing 1’s into 0’s and 0’s into 1’s. For example, the 1’s complement of 1011001 is 0100110 and the 1’s complement of 0001111 is 1110000.
- The (r — l)’s complement of octal or hexadecimal numbers are obtained by subtracting each digit from 7 or F (decimal 15) respectively.

(r ’s) Complement

- 10’s Complement
  - The r’s complement of an n -digit number N in base r is defined as r^n - N for N ≠ 0 and 0 for N = 0. Comparing with the (r - 1)’s complement, we note that the r’s complement is obtained by adding 1 to the (r - 1)’s complement since r^n - N = [(r^n - 1) - N] + 1. Thus the 10’s complement of the decimal 2389 is 7610 + 1 = 7611 and is obtained by adding 1 to the 9’s complement value.
  - The 2’s complement of binary 101100 is 010011 + 1 = 010100 and is obtained by adding 1 to the l’s complement value.

2’s Complement

- Since 10^n is a number represented by a 1 followed by n 0’s, then 10n - N, which is the 10’s complement of N, can be formed also be leaving all least significant 0’s unchanged, subtracting the first nonzero least significant digit from 10, and then subtracting all higher significant digits from 9. The 10’s complement of 246700 is 753300 and is obtained by leaving the two zeros unchanged, subtracting 7 from 10, and subtracting the other three digits from 9. Similarly, the 2’s complement can be formed by leaving all least significant 0’s and the first 1 unchanged, and then replacing 1’s by 0’s and 0’s by 1’s in all other higher, significant bits.
- The 2’s complement of 1101100 is 0010100 and is obtained by leaving the two low-order 0’s and the first 1 unchanged, and then replacing 1’s by 0’s and 0’s by 1’s in the other four most significant bits.

Subtraction of Unsigned Numbers

- The subtraction of two n -digit unsigned numbers M N(N 0) in base r can be done as follows:

1. Add the minuend M to the r's complement of the subtrahend N. This performs M + (r^n - N) = M - N + r^n.

2. If M ≥ N, the sum will produce an end carry r^n which is discarded, and what is left is the result M - N.

3. If M < N, the sum does not produce an end carry and is equal to r^n - (N - M), which is the r's complement of (N - M). To obtain the answer in a familiar form, take the r's complement of the sum and place a negative sign in front.

Example

- Consider, for example, the subtraction 72532 - 13250 = 59282. The 10's complement of 13250 is 86750. Therefore:

$$\begin{aligned} M &= 72532 \\ \text{10's complement of } N &= +86750 \\ \text{Sum} &= 159282 \\ \text{Discard end carry } 10^5 &= -100000 \\ \text{Answer} &= 59282 \end{aligned}$$

- Now consider an example with M < N. The subtraction 13250 – 72532 produces negative 59282. Using the procedure with complements, we have

$$\begin{aligned} M &= 13250 \\ \text{10's complement of } N &= +27468 \\ \text{Sum} &= 40718 \end{aligned}$$

- There is no end carry Answer is negative 59282 = 10's complement of 40718

Integer Representation

- When an integer binary number is positive, the sign is represented by 0 and the magnitude by a positive binary number. When the number is negative, the sign is represented by 1 but the rest of the number may be represented in one of three possible ways:

1. Signed-magnitude representation
2. Signed-1’s complement representation
3. Signed 2’s complement representation

- The signed-magnitude representation of a negative number consists of the magnitude and a negative sign. In the other two representations, the negative number is represented in either the 1’s or 2’s complement of its positive value.

Example

- Consider the signed number 14 stored in an 8-bit register. +14 is represented by a sign bit of 0 in the leftmost position followed by the binary equivalent of 14: 00001110. Note that each of the eight bits of the register must have a value and therefore 0's must be inserted in the most significant positions following the sign bit. Although there is only one way to represent +14, there are three different ways to represent -14 with eight bits.
- In signed-magnitude representation 1 0001110
- In signed-1's complement representation 1 1110001
- In signed-2's complement representation 1 1110010

Arithmetic Addition

- The addition of two numbers in the signed-magnitude system follows the rules of ordinary arithmetic. If the signs are the same, we add the two magnitudes and give the sum the common sign. If the signs are different, we subtract the smaller magnitude from the larger and give the result the sign of the larger magnitude.
- For example, (+25) + (-37) = -(37 - 25) = 112 and is done by subtracting the smaller magnitude 25 from the larger magnitude 37 and using the sign of 37 for the sign of the result.

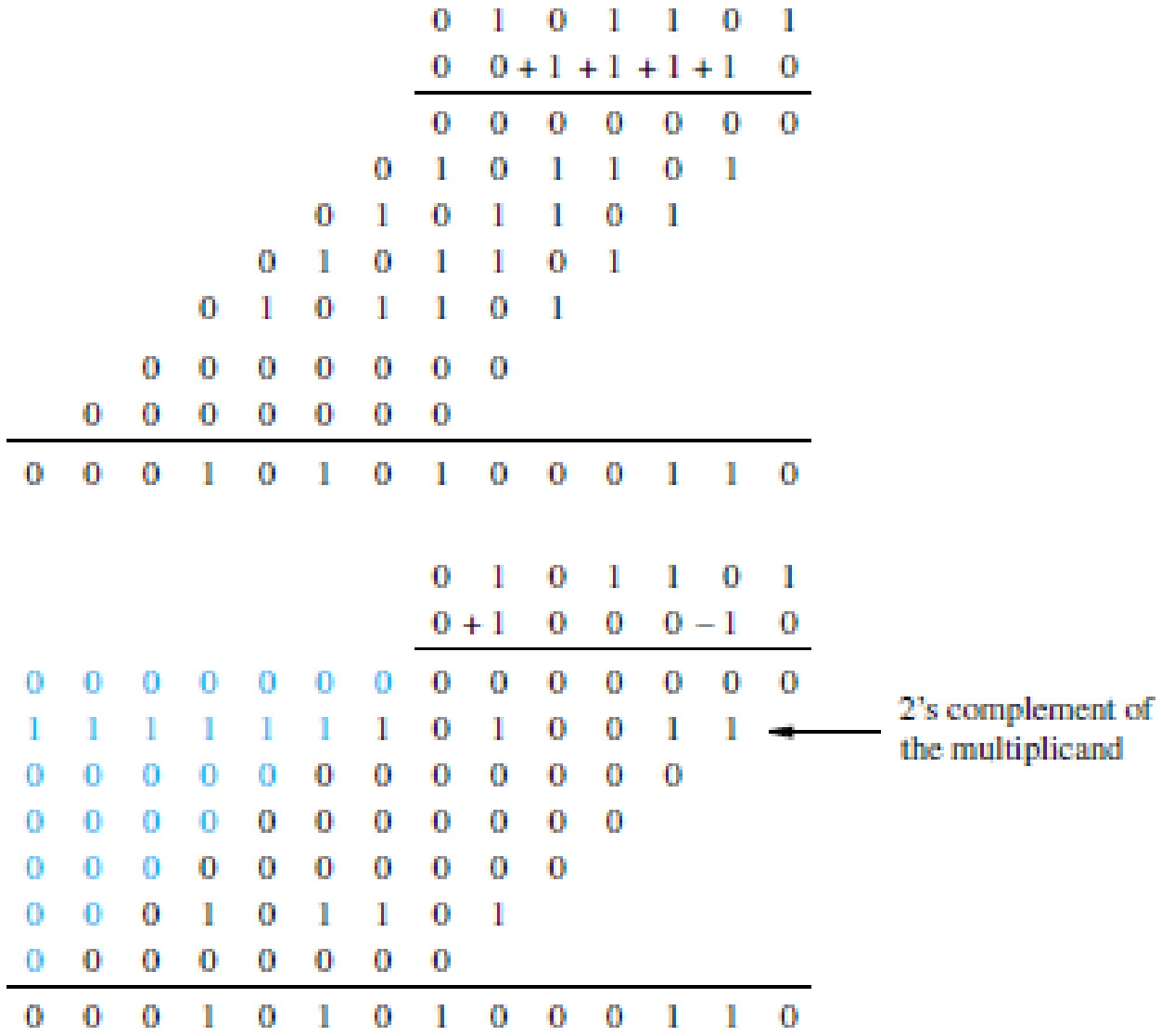
$+6$	00000110	-6	11111010
$+13$	00001101	$+13$	00001101
$+19$	00010011	$+7$	00000111
$+6$	00000110	-6	11111010
-13	11110011	-13	11110011
-7	11111001	-19	11101101

The Booth Algorithm Multiplication

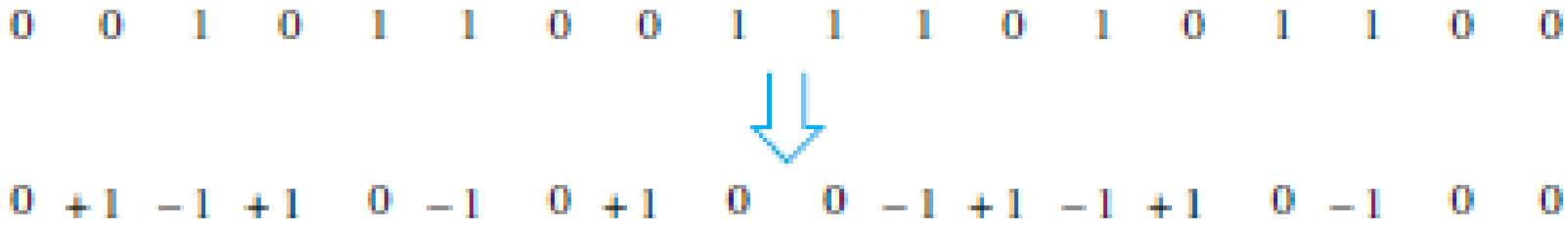
- The Booth algorithm generates a 2n-bit product and treats both positive and negative 2’scomplement n-bit operands uniformly.
- Consider a multiplication operation in which the multiplier is positive and has a single block of 1s, for example, 0011110. To derive the product, we could add four appropriately shifted versions of the multiplicand, as in the standard procedure. However, we can reduce the number of required operations by regarding this multiplier as the difference between two numbers:

	0100000	(32)
–	0000010	(2)
	0011110	(30)

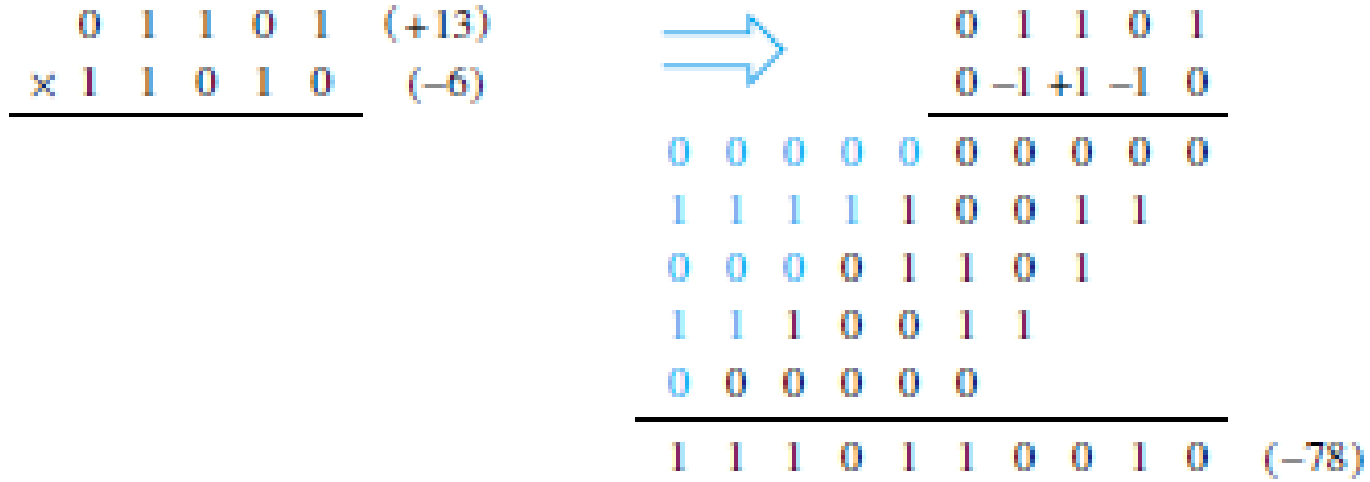
Normal and Booth Multiplication Schemes



Booth Recoding of a Multiplier

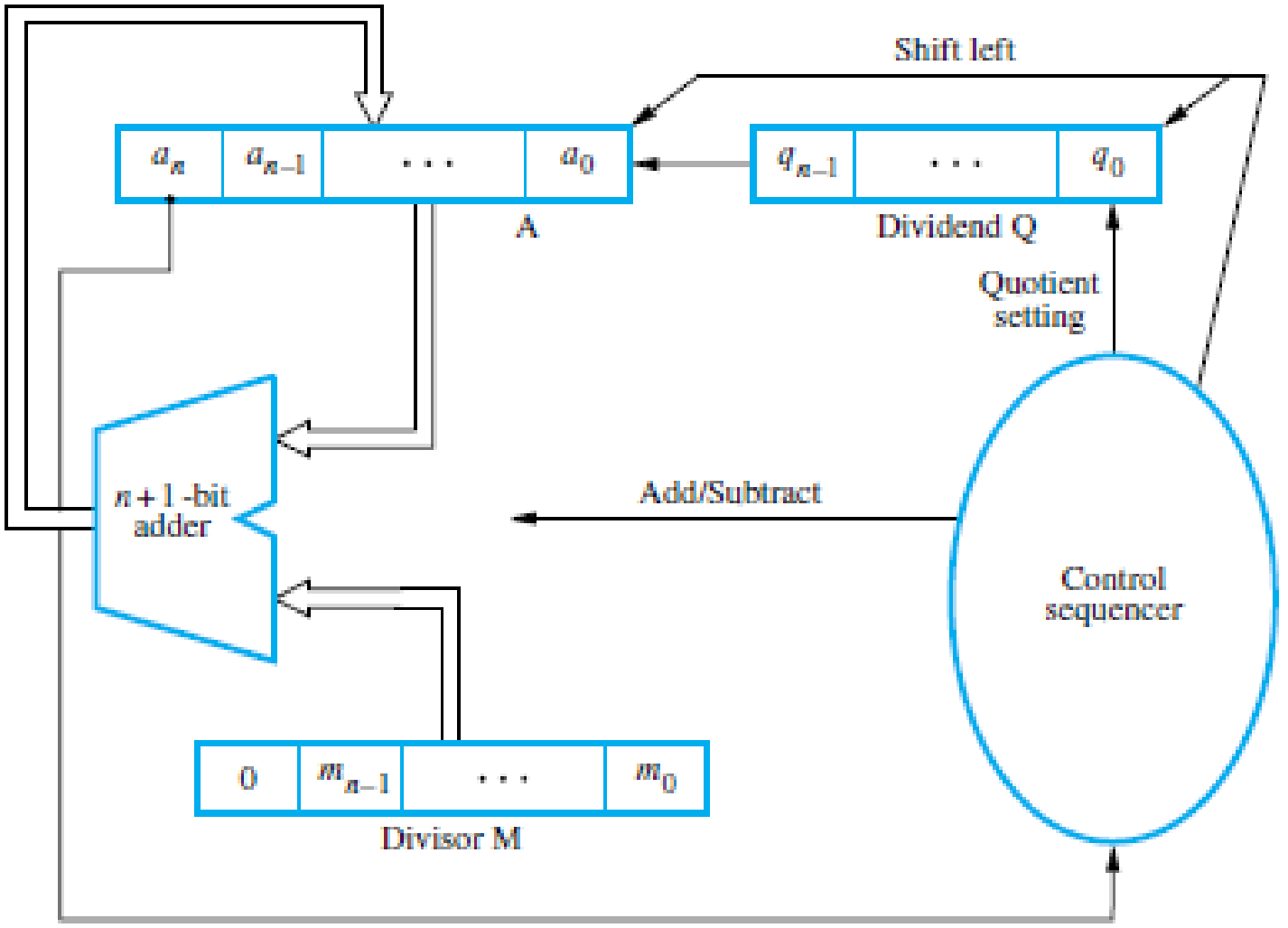


Booth Multiplication with a Negative Multiplier



Circuit arrangement for binary division

- An n-bit positive divisor is loaded into register M and an n-bit positive dividend is loaded into register Q at the start of the operation.
- Register A is set to 0.
- After the division is complete, the n-bit quotient is in register Q and the remainder is in register A.
- The required subtractions are facilitated by using 2's-complement arithmetic.
- The extra bit position at the left end of both A and M accommodates the sign bit during subtractions.



Steps of Restoring Division Algorithm

- The following algorithm performs restoring division.

Do the following three steps n times:

1. Shift A and Q left one bit position.

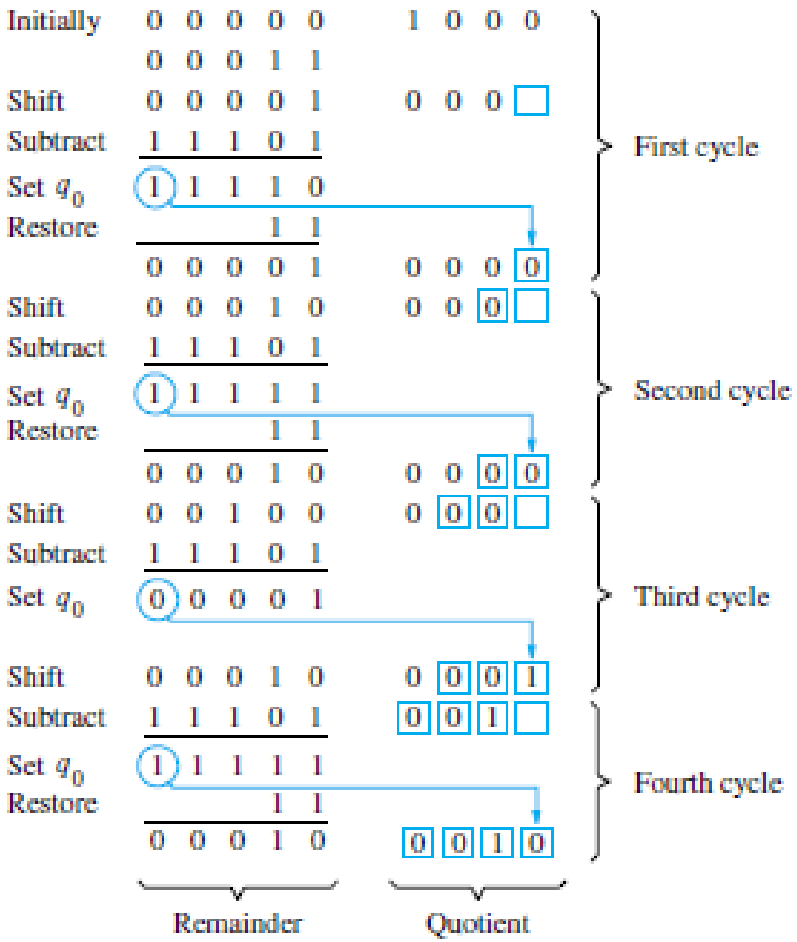
2. Subtract M from A, and place the answer back in A.

3. If the sign of A is 1, set q0 to 0 and add M back to A (that is, restore A); otherwise, set q0 to 1.

Link :
https://www.codingninjas.com/studio/library/introduction-to-division-algorithm-in-computer-architecture

Example

$$
\begin{array}{r}
10 \\
11\overline{)\,1000} \\
\underline{11} \\
10
\end{array}
$$



Fixed-point and floating-point representations of numbers

- A fixed-point representation of a number may be thought to consist of 3 parts: the sign field, integer field, and fractional field. One way to store a number using a 32-bit format is to reserve 1 bit for the sign, 15 bits for the integer part and 16 bits for the fractional part.
- A number whose representation exceeds 32 bits would have to be stored inexactly.

Example. The 32-bit string

$$1 \mid 000000000101011 \mid 1010000000000000$$

represents $(-101011.101)_2 = -43.625$.

Fractions: Two Representations

- Fixed-point: binary point is fixed

1101101.0001001

- Floating-point: binary point floats to the right of the most significant 1 and an exponent is used

$1.1011010001001 \times 2^6$

- Fixed-point representation using 4 integer bits and 3 fraction bits:

0110110

interpreted as **0110.110**

$= 2^2 + 2^1 + 2^{-1} + 2^{-2} = 6.75$

Fixed-Point Numbers

- The binary point is not a part of the representation but is implied
- The number of integer and fraction bits must be agreed upon by those generating and those reading the number

Signed Fixed-Point Numbers

- Negative fractional numbers can be represented two ways:
  - Sign/magnitude notation
  - Two's complement notation
- Represent -7.5_{10} using an 8-bit binary representation with 4 integer bits and 4 fraction bits in Two's complement:
  - +7.5: 01111000
  - Invert bits: 10000111
  - Add 1 to lsb: 10001000

Floating-Point Numbers

- The binary point floats to the right of the most significant digit
- Similar to decimal scientific notation:
  - For example, 273_{10} in scientific notation is
    $$273 = 2.73 \times 10^2$$
- In general, a number is written in scientific notation as:
  $$\pm M \times B^E$$

where:

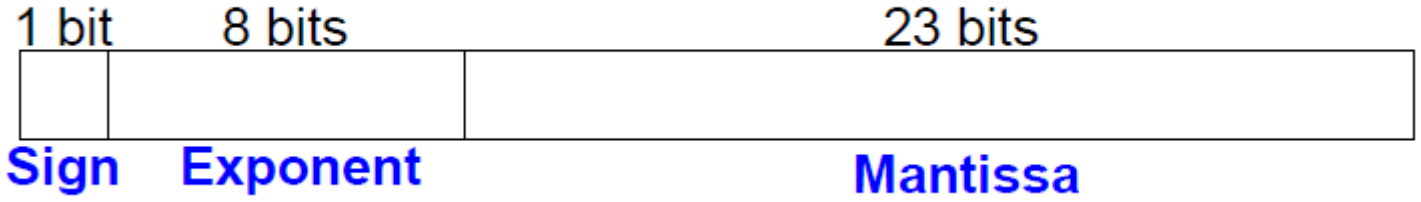
M= mantissa

B= base

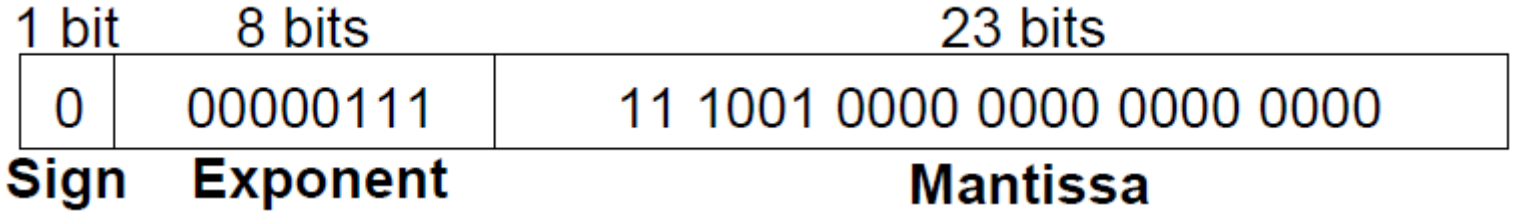
E= exponent

- In the example, M = 2.73, B = 10, and E = 2

Floating-Point Numbers



- Convert the decimal number to binary: 228_{10}= 11100100_2= $1.11001 \times 2_7$
- Fill in each field of the 32-bit number:
- The sign bit is positive (0)
- The 8 exponent bits represent the value 7
- The remaining 23 bits are the mantissa



BIG ENDIAN vs LITTLE ENDIAN

- In a big-endian computer, the two bytes required for the hexadecimal number 4F52 would be stored as 4F52 in storage.
- For example, if 4F is stored at storage address 1000, 52 will be at address 1001.
- In a little-endian system, it would be stored as 524F, with 52 at address 1000 and 4F at 1001.

Register Transfer Language

- Digital system design invariably uses a modular approach. The modules are constructed from such digital components as registers, decoders, arithmetic elements, and control logic.
- The various modules are interconnected with common data and control paths to form a digital computer system.
- Digital modules are best defined by the registers they contain and the operations that are performed on the data stored in them. The operations executed on data stored in registers are called microoperations.
- A microoperation is an elementary operation performed on the information stored in one or more registers. The result of the operation may replace the previous binary information of a register or may be transferred to another register.
- Examples of microoperations are shift, count, clear, and load.

Register Transfer Language Contd…

- The internal hardware organization of a digital computer is best defined by specifying:

1. The set of registers it contains and their function.

2. The sequence of microoperations performed on the binary information stored in the registers.

3. The control that initiates the sequence of microoperations.

- The symbolic notation used to describe the microoperation transfers among registers is called a register transfer language.

Introduction to Register

- Computer registers are designated by capital letters (sometimes followed by numerals) to denote the function of the register. For example, the register that holds an address for the memory unit is usually called a memory address register and is designated by the name MAR. Other designations for registers are PC (for program counter), IR (for instruction register, and R1 (for processor register).

Register Representation

- The most common way to represent a register is by a rectangular box with the name of the register inside, as in Figure 4 (a).
- The individual bits can be distinguished as in Figure 4(b).
- The numbering of bits in a 16-bit register can be marked on top of the box as shown in Figure 4(c).
- A 16-bit register is partitioned into two parts is shown in Figure 4(d).
- Bits 0 through 7 are assigned the symbol L (for low byte) and bits 8 through 15 are assigned the symbol H (for high byte). The name of the 16-bit register is PC. The symbol PC(0–7) or PC(L) refers to the low-order byte and PC(8–15) or PC(H) to the high-order byte.

Register Representation Contd…

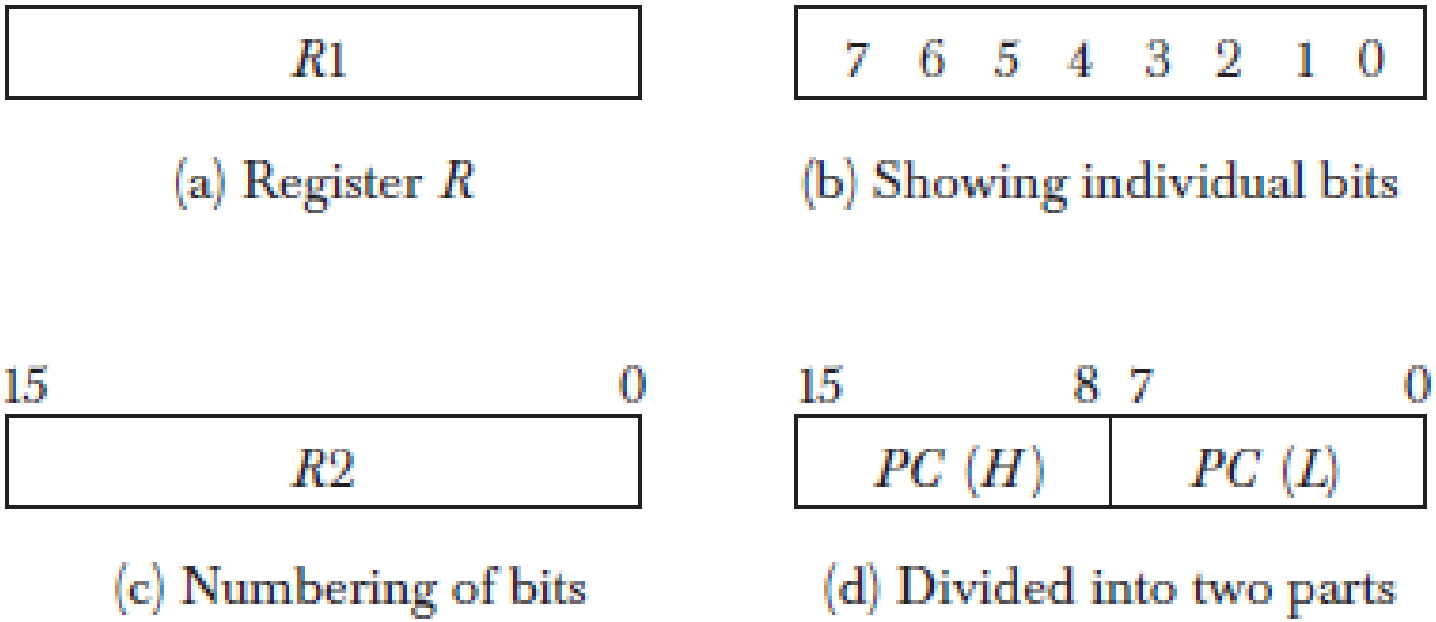


Figure 4

Register Transfer

- Information transfer from one register to another is designated in symbolic form by means of a replacement operator.

$$R2 \leftarrow R1$$

- The above statement denotes a transfer of the content of register R1 into register R2. It designates a replacement of the content of R2 by the content of R1. By definition, the content of the source register R1 does not change after the transfer.

Arithmetic Microoperations

- A microoperation is an elementary operation performed with the data stored in registers. The microoperations most often encountered in digital computers are classified into four categories:

1. Register transfer microoperations transfer binary information from one

register to another.

2. Arithmetic microoperations perform arithmetic operation on numeric data stored in registers.

3. Logic microoperations perform bit manipulation operations on nonnumeric data stored in registers.

4. Shift microoperations perform shift operations on data stored in registers.

ADD Microoperation

- The basic arithmetic microoperations are addition, subtraction, increment, decrement, and shift. Arithmetic shifts are explained later in conjunction with the shift microoperations.

$$R3 \leftarrow R1 + R2$$

- The arithmetic microoperation defined by the above statement specifies an add microoperation. It states that the contents of register R1 are added to the contents of register R2 and the sum transferred to register R3.

SUBTRACT Microoperation

- Subtraction is most often implemented through complementation and addition.

$$R3 \leftarrow R1 + \overline{R2} + 1$$

- $\overline{R2}$ is the symbol for the 1's complement of R2. Adding 1 to the 1's complement produces the 2's complement. Adding the contents of R1 to the 2's complement of R2 is equivalent to R1 - R2.

Summary of Arithmetic Microoperations

Symbolic designation	Description
$R3 \leftarrow R1 + R2$	Contents of $R1$ plus $R2$ transferred to $R3$
$R3 \leftarrow R1 - R2$	Contents of $R1$ minus $R2$ transferred to $R3$
$R2 \leftarrow \overline{R2}$	Complement the contents of $R2$ (1's complement)
$R2 \leftarrow \overline{R2} + 1$	2's complement the contents of $R2$ (negate)
$R3 \leftarrow R1 + \overline{R2} + 1$	$R1$ plus the 2's complement of $R2$ (subtraction)
$R1 \leftarrow R1 + 1$	Increment the contents of $R1$ by one
$R1 \leftarrow R1 - 1$	Decrement the contents of $R1$ by one

Logic Microoperations

- Logic microoperations specify binary operations for strings of bits stored in registers. These operations consider each bit of the register separately and treat them as binary variables. For example, the exclusive-OR microoperation with the contents of two registers R1 and R2 is symbolized by the statement

$$P: \quad R1 \leftarrow R1 \oplus R2$$

Summary of Logic Operations

Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = (x + y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Exclusive-NOR
$F_{10} = y'$	$F \leftarrow \overline{B}$	Complement B
$F_{11} = x + y'$	$F \leftarrow A \vee \overline{B}$	
$F_{12} = x'$	$F \leftarrow \overline{A}$	Complement A
$F_{13} = x' + y$	$F \leftarrow \overline{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow$ all 1's	Set to all 1's

Shift Microoperations

- Shift microoperations are used for serial transfer of data.
- The contents of a register can be shifted to the left or the right.
- During a shift-left operation the serial input transfers a bit into the rightmost position. During a shift-right operation the serial input transfers a bit into the leftmost position.
- There are three types of shifts: logical, circular, and arithmetic.

logical shift

- A logical shift is one that transfers 0 through the serial input.
- For Example:

$$R1 \leftarrow \text{shl } R1$$
$$R2 \leftarrow \text{shr } R2$$

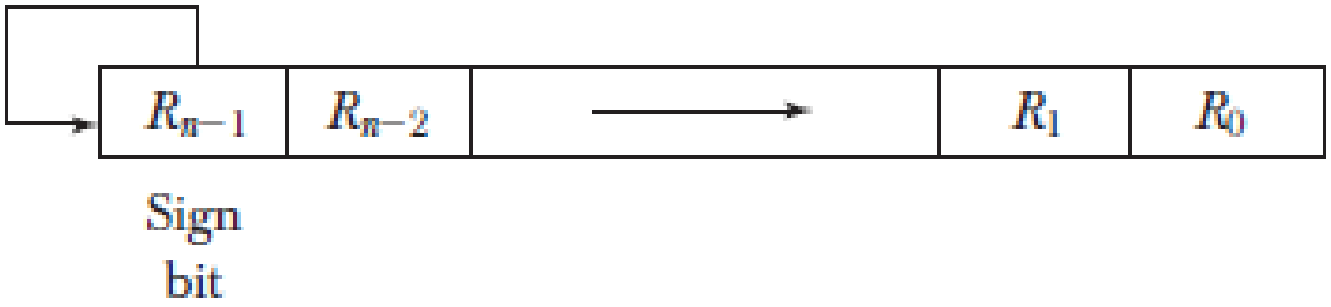
- are two microoperations that specify a 1-bit shift to the left of the content of register R1 and a 1-bit shift to the right of the content of register R2. The register symbol must be the same on both sides of the arrow. The bit transferred to the end position through the serial input is assumed to be 0 during a logical shift.

circular shift

- The circular shift (also known as a rotate operation) circulates the bits of the register around the two ends without loss of information. This is accomplished by connecting the serial output of the shift register to its serial input.

arithmetic shift

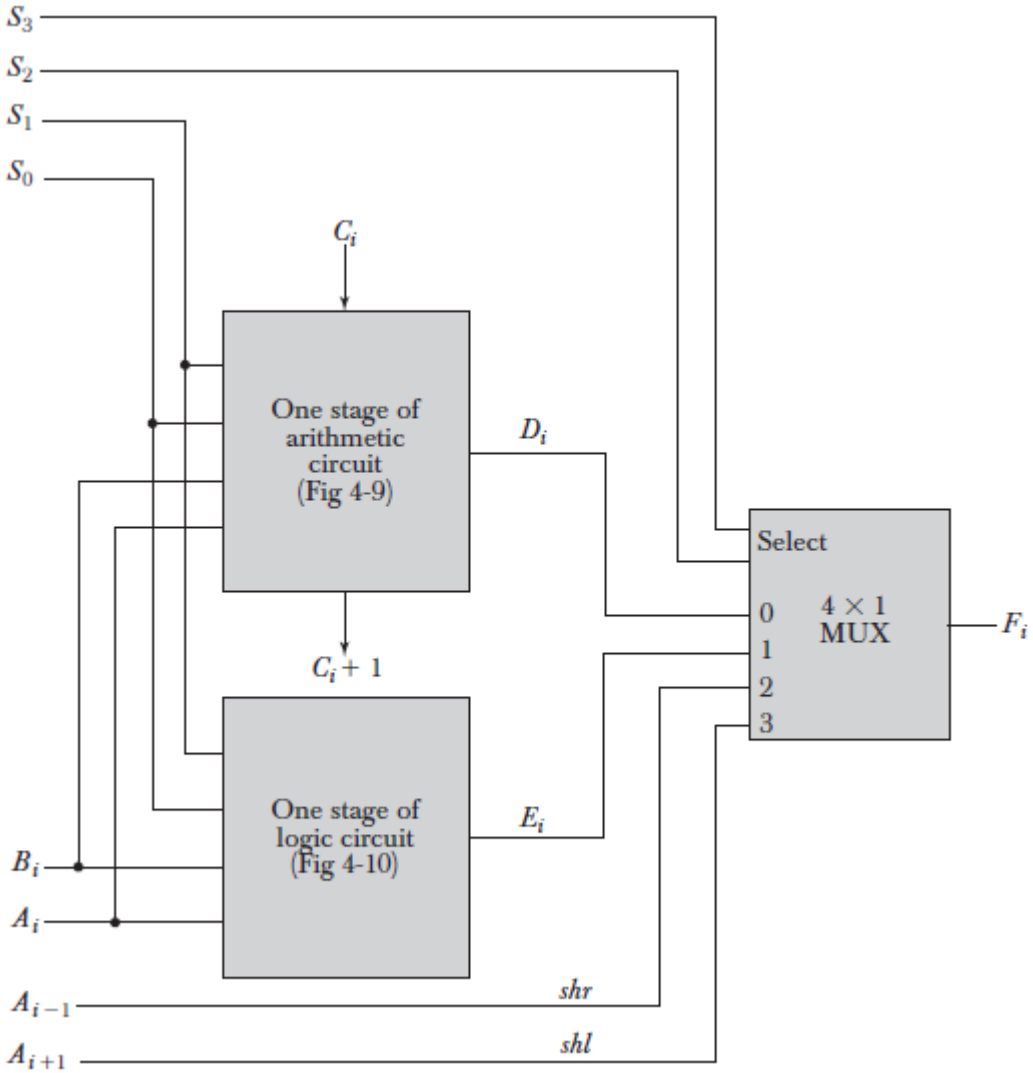
- An arithmetic shift is a microoperation that shifts a signed binary number to the left or right. An arithmetic shift-left multiplies a signed binary number by 2. An arithmetic shift-right divides the number by 2.
- The leftmost bit in a register holds the sign bit, and the remaining bits hold the number. The sign bit is 0 for positive and 1 for negative. Negative numbers are in 2's complement form.



Summary of Shift Microoperations

Symbolic designation	Description
$R \leftarrow \text{shl } R$	Shift-left register R
$R \leftarrow \text{shr } R$	Shift-right register R
$R \leftarrow \text{cil } R$	Circular shift-left register R
$R \leftarrow \text{cir } R$	Circular shift-right register R
$R \leftarrow \text{ashl } R$	Arithmetic shift-left R
$R \leftarrow \text{ashr } R$	Arithmetic shift-right R

Arithmetic Logic Shift Unit



Operation select						
S_3	S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	0	$F = A$	Transfer A
0	0	0	0	1	$F = A + 1$	Increment A
0	0	0	1	0	$F = A + B$	Addition
0	0	0	1	1	$F = A + B + 1$	Add with carry
0	0	1	0	0	$F = A + \overline{B}$	Subtract with borrow
0	0	1	0	1	$F = A + \overline{B} + 1$	Subtraction
0	0	1	1	0	$F = A - 1$	Decrement A
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	×	$F = A \wedge B$	AND
0	1	0	1	×	$F = A \vee B$	OR
0	1	1	0	×	$F = A \oplus B$	XOR
0	1	1	1	×	$F = \overline{A}$	Complement A
1	0	×	×	×	$F =$ shr A	Shift right A into F
1	1	×	×	×	$F =$ shl A	Shift left A into F