CORE - II COMPUTER ORGANIZATION

OBJECTIVES:

- To understand the basic organization of computers and the working of each component and CPU
- To bring the programming features of 8085 Microprocessor and know the features of latest microprocessors.
- To understand the principles of Interfacing I/O devices and Direct Memory accesses

OUTCOMES:

- Describe the major components of a computer system and state their function and purpose
- Describe the microstructure of a processor
- Demonstrate the ability to program a microprocessor in assembly language.
- Classify and describe the operation DMA and peripheral Interfaces.

UNIT - I

Data Representation: Data Types – Complements – Fixed-Point and Floating-Point Representation Other Binary Codes. **Register Transfer and Microoperations**: Register Transfer Language - Register Transfer - Bus and Memory Transfers – Arithmetic, Logic and Shift Microoperations.

UNIT - II

Central Processing Unit: General Register and Stack Organizations - Instruction Formats - Addressing Modes - Data Transfer and Manipulation - Program Control - RISC - Pipelining - Arithmetic and Instruction - RISC Pipeline - Vector Processing and Array Processors.

UNIT - III

Microprocessor Architecture and its Operations - 8085 MPU - 8085 Instruction Set and Classifications. **Programming in 8085: Code Conversion** – BCD-to-Binary and Binary-to-BCD Conversions – ASCII-to-BCD and BCD-to-ASCII Conversions – Binary-to-ASCII and ASCII-to -Binary Conversions.

UNIT - IV

Programming in 8085: BCD Arithmetic - BCD addition and Subtraction - Multibyte Addition and Subtraction - Multiplication and Division. **Interrupts**: The 8085 Interrupt – 8085 Vectored Interrupts.

UNIT - V

Direct Memory Access (DMA) and 8257 DMA Controller - 8255A Programmable Peripheral Interface. Basic Features of Advanced Microprocessors - Pentium - I3, I5 and I7

TEXT BOOKS:

1. M.M. Mano, “Computer System Architecture”. Pearson, Third Edition, 2007
2. R. S. Gaonkar- "Microprocessor Architecture- Programming and Applications with 8085"- 5th Edition- Penram-2009.
3. Tripti Dodiya & Zakiya Malek, “Computer Organization and Advanced Microprocessors”, Cengage Learning,2012.

REFERENCE BOOKS:

1. Mathur- “Introduction to Microprocessor”- 3rd Edition- Tata McGraw-Hill-1993.
2. P. K. Ghosh and P. R. Sridhar- “0000 to 8085: Introduction to Microprocessors for Engineers and Scientists”-2nd Edition- PHI- 1995.
3. NagoorKani- “Microprocessor (8085) and its Applications”- 2nd Edition- RBA Publications- 2006.
4. V. Vijayendran- “Fundamentals of Microprocessors – 8085”- S. Viswanathan Pvt. Ltd.- 2008.

WEB REFERENCES:

- NPTEL & MOOC courses titled Computer organization
- https://nptel.ac.in/courses/106105163/
- https://nptel.ac.in/courses/106103068/

UNIT - I

Chapter 1 Data Re

U

tral Processing Unit

eneral Register Organization
tack Organization
nstruction Formats
Addressing Modes
Data Transfer and Manipulation
Program Control
Reduced Instruction Set Computer (RISC)

4 Pipeline and Vector Processing

Pipelining
Arithmetic Pipeline
Instruction Pipeline
RISC Pipeline
Vector Processing
Array Processors

UNIT – I

Chapter 1 Data Representation

This Chapter shows how the various data types found in digital computers are represented in binary form in computer registers. Emphasis is on the representation of numbers employed in arithmetic operations, and on the binary coding of symbols used in data processing.

1.1 Data Types

Data Types are the classification or categorization of data items.

It represents the kind of value that tells what operations can be performed on a particular data.

Data are numbers and other binary-coded information that are operated on to achieve required computational results.

The data types found in the registers of digital computers may be classified as being one of the following categories:

1) Numbers used in arithmetic computations,
2) Letters of the alphabet used in data processing and
3) Other discrete symbols used for specific purposes.

All types of data, except binary numbers, are represented in computer organization registers in binary-coded form.

Registers are made up of flip-flops and flip-flops are two-state devices that can store only 1's and 0's.

The Binary Number System is the most natural system to use in a digital computer.

Number System

- A Number System of base, or radix, r is a system that uses distinct symbols for r digits.
- Numbers are represented by a string of digit symbols.
- To determine the quantity that the number represents, it is necessary to multiply each digit by aninteger power of r and then form the sum of all weighted digits.
- Computer Architecture supports the following 4 Types of Number Systems:
  1. Decimal Number System
  2. Binary Number System
  3. Octal Number System
  4. Hexadecimal Number System

1. **Decimal Number System**

- The Decimal Number System is also known as the base-10 number system.
- It uses ten symbols for representing numbers.
- They are 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9.

2. **Binary Number System**

- The Binary Number System is also known as the base-2 number system.
- It has only two digits **0** and **1**.
- The Binary Number System is the most commonly used number system used in Digital Computers.

3. **Octal Number System**

- The octal number system is also known as the base-8 number system.
- It uses eight different symbols for representing numbers.
- They are 0, 1, 2, 3, 4, 5, 6 and 7.

4. **Hexadecimal Number System**

- The Hexadecimal Number System is also known as the base-16 number system.
- It uses sixteen different symbols for representing numbers.
- They are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F. Here **A is 10**, **B is 11**, **C is 12**, **D is 13**, **E is 14** and **F is 15**.

Number System Conversion

Decimal to Binary Conversion

a) **Converting the Decimal Integer to Binary**

Steps:

1) Divide the Number by 2.
2) Write the Quotient down.
3) Write the remainder on the right hand side. This will be 0 or 1.
4) Repeat the above steps till the quotient becomes zero.
5) Read upwards the sequence of remainders.

Example 1:

Convert the Decimal Number $(34)_{10}$ to Binary.

	Remainder
2)34	
2)17	0
2) 8	1
2) 4	0
2) 2	0
2) 1	0
0	1

The remainders are to be read from bottom to top to obtain the binary equivalent.

Thus, $(34)_{10} = (100010)_2$

b) **Converting Decimal Fraction to Binary**

Steps:

1. Multiply the number by 2.
2. Write the value of the integer part.
3. Repeat the above two steps until the fractional part becomes zero.
4. Read downwards the sequence of integer parts.

Example 2:

Convert the Decimal fraction $(0.4375)_{10}$ to Binary

0.4375 x 2 = 0.8750 with a carry of 0
0.8750 x 2 = 1.75 with a carry of 1
0.75 x 2 = 1.5 with a carry of 1
1.5 x 2 = 1.0 with a carry of 1

Thus, $(0.4375)_{10} = (0.0111)_2$

Example 3:

Convert the Decimal Number $(98.46)_{10}$ to Binary

Binary equivalent of 98 is 1100010_2

Fractional part:
$0.46 \times 2 = 0.92 = 0$
$0.92 \times 2 = 1.84 = 1$
$0.84 \times 2 = 1.68 = 1$
$0.68 \times 2 = 1.36 = 1$
$0.36 \times 2 = 0.72 = 0$

2	98	
2	49	– 0
2	24	– 1
2	12	– 0
2	6	– 0
2	3	– 0
	1	– 1

Thus, $(98.46)_{10} = (1100010.01110)_2$

Binary to Decimal Conversion

Steps:

1. Separate the digits of the given Binary number, if it contains more than 1 digit.
2. Multiply each digit of the Binary number with its increasing power of 2 from right to left.
3. Add all the individual results provide the equivalent decimal number

Example 1:

Convert the Binary Number $(100011)_2$ to Decimal.
Table 1 shows the conversion from Binary to Decimal.
Table 1 Conversion table from Binary to Decimal

Binary Number:	1	0	0	0	1	1
Power of 2:	2^5	2^4	2^3	2^2	2^1	2^0

$(100011)_2 = 1 \times 2^5+0 \times 2^4+0 \times 2^3+0 \times 2^2+1 \times 2^1+1 \times 2^0 = (35)_{10}$
Thus, $(100011)_2 = (35)_{10}$

Example 2:

Convert the Binary Number $(1101011.1011)_2$ to Decimal.

Table 2 shows the conversion from Binary $(1101011.1011)_2$ to Decimal
Table 2 Conversion table from Binary to Decimal, for the integer part and fractions

Binary Number (Integer Part) 1101011	1	1	0	1	0	1	1
Power of 2:	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Binary Number (Fraction Part) .1011	1	0	1	1			
Power of 2:	2^{-1}	2^{-2}	2^{-3}	2^{-4}			

Integer Part $(1101011)_2 = 1 \times 2^6+1 \times 2^5+0 \times 2^4+1 \times 2^3+0 \times 2^2+1 \times 2^1+1 \times 2^0 = (107)_{10}$
Fraction Part $(.1011) = 1 \times 2^{-1}+0 \times 2^{-2}+1 \times 2^{-3}+1 \times 2^{-4} = (0.6875)_{10}$

Thus, $(1101011.1011)_2 = (107.6875)_{10}$

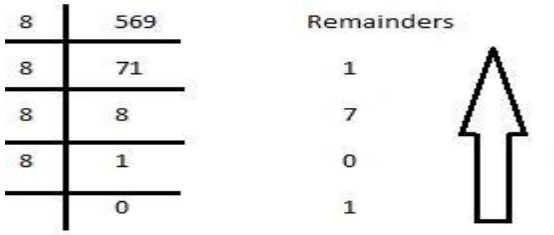
Decimal to Octal Conversion

a) **Converting an Integer Decimal Number to Octal Steps:**

1. Divide the number by 8.
2. Write the Quotient down.
3. Write the remainder on the right hand side.
4. Repeat the above steps till the quotient becomes zero.
5. Read upwards the sequence of remainders.

Example 1:

Convert the Decimal number $(569)_{10}$ to Octal.



Thus, $(569)_{10} = (1071)_8$

b) **Converting a fractional Decimal Number to Octal Steps:**

1. Multiply the number by 8.
2. Write the value of the integer part.
3. Repeat the above two steps until the fractional part becomes zero.
4. Read downwards the sequence of integer parts.

Example 2:

Convert the fractional decimal number $(0.140869140625)_{10}$ to Octal.

Multiplication	Resultant integer part
0.140869140625 x 8 = 0.12695313	1
0.12695313 x 8 = 0.01562504	1
0.01562504 x 8 = 0.12500032	0
0.12500032 x 8 = 0.00000256	1
0.00000256 x 8 = 0.000020544	0

Thus, $(0.140869140625)_{10} = (11010)_8$

Octal to Decimal Conversion

Steps:

1. Separate the digits of the given octal number, if it contains more than 1 digit.
2. Multiply each digit of octal number with its increasing power of 8 from right to left
3. Add all the individual results provide the equivalent decimal number.

Example 1:

Convert the Octal number $(75)_8$ to Decimal

Table 3 shows the conversion from Octal $(75)_8$ to Decimal.
Table 3 Conversion Table from Octal to Decimal, for the integer part

Octal Number:	7	5
Power of 8:	8^1	8^0

$(75)_8 = 7 \times 8^1 + 5 \times 8^0 = (61)_{10}$
Thus, $(75)_8 = (61)_{10}$

Example 2:

Convert the Octal number $(45.6)_8$ to Decimal.

Table 4 shows the conversion from Octal $(45.6)_8$ to Decimal.

Conversion Table from Octal to Decimal for the integer part and fractions

Octal Number:	4	5	.	6
Power of 8:	$8^1 = 8$	$8^0 = 1$	.	8^{-1}
	4 x 8	5 x 1	.	6 x 1/8

$(45.6)_8 = 4 \text{ X } 8^1 + 5\text{X } 8^0 + 6\text{X}8^{-1} = 32+5+0.75 = (37.75)_{10}$

Thus, **$(45.6)_8$** = $(37.75)_{10}$

Octal to Binary Conversion

Steps:

1. Separate the digits of the given Octal Number, if it contains more than one digit.
2. Find the equivalent Binary Number for each digit of the Octal Number. Add 0's to the left if any of the Binary equivalent is shorter than 3 bits.
3. Write all the binary numbers together, maintaining the same group order provides the equivalent Binary for the given octal number.

Example:

Convert the Octal $(7631)_8$ to its Binary equivalent.

7	6	3	1
111	110	011	001

Thus, $(7631)_8 = (111110011001)_2$

Binary to Octal Conversion

Steps:

1. Separate the digits of a given Binary number into groups of 3 bits starting from right to left.
2. Find the equivalent octal number for each group.
3. Write all the group octal numbers together, maintaining group order provides the equivalent octal number for the given binary.

Example:

Convert the Binary number $(111110011001)_2$ into Octal.

111	110	011	001
7	6	3	1

Thus, $(111110011001)_2 = (7631)_8$

Octal to Hexadecimal Conversion

Steps:

1. Convert the given Octal Number to Binary.
2. Then Convert the Binary Number to Hexadecimal.

Example:

Convert the Octal number $(536)_8$ to Hexadecimal number.

Converting 536 (Octal) into its Binary equivalent we get

$(536)_8 = 101\ 011\ 110$

$= (101011110)_2$

Now forming the group of 4 binary bits to obtain its hexadecimal equivalent,

$(101011110)_2 = 0001\ 0101\ 1110$

$= (15E)_{16}$

Thus, $(536)_8 = (15E)_{16}$

Hexadecimal to Octal Conversion

Steps:

1. Convert the given Hexadecimal Number to Binary.
2. Then convert the Binary Number to Octal.

Example:

Convert the Hexadecimal $(1BC)_{16}$ into its Octal equivalent.

Given, $(1BC)_{16}$ is a Hexadecimal number.

1 → 0001, B → 1011, C →1100

Now group them from right to left, each having 3 digits.

000 110 111 100

000→0, 110 →6, 111→7, 100→4

Thus, $(1BC)_{16} = (674)_8$

Decimal to Hexadecimal Conversion

a) Converting an Integer Decimal Number to Hexadecimal Steps:

1. Divide the number by 16.
2. Write the Quotient down.
3. Write the remainder on the right hand side.
4. Repeat the above steps till the quotient becomes zero.
5. Read upwards the sequence of remainders.

Example:

Convert the Decimal $(423)_{10}$ to Hexadecimal.

		Remainder
16	423	
16	26	7
16	1	A
	0	1

Thus, $(423)_{10} = (1A7)_{16}$

b) Converting an Fractional Decimal Number to Hexadecimal

Steps:

1. Multiply the number by 16.
2. Write the value of the integer part.
3. Repeat the above two steps until the integer part became zero.
4. Read downwards the sequence of integer parts.

Example:

Convert the Decimal fraction $(0.06640625)_{10}$ to Hexadecimal.

Multiplication	Resultant integer part
0.06640625 x 16=1.0625	1
0.0625 x 16 =1.0	1
0 x 16=0.0	0

Thus, $(0.06640625)_{10} = (0.110)_{16}$

Hexadecimal to Decimal Conversion

Steps:

1. Get the Decimal equivalent of the Hexadecimal number.
2. Multiply every digit with 16 power of digit location.
3. Sum all the multipliers.

Example:

Convert the Hexadecimal $(F3)_{16}$ to Decimal.

Table 5 shows the conversion from Hexadecimal (F3)16 to Decimal.

Table 5 Conversion Table from Hexadecimal to Decimal

Hexadecimal Number:	F	3	$(F3)_{16}$
Power of 16:	$16^1 = 16$	$16^0 = 1$	
Decimal Number	15 x16	3 x 1	$(243)_{10}$

= 15 x 16 +3

Thus, $(F3)_{16} = (243)_{10}$

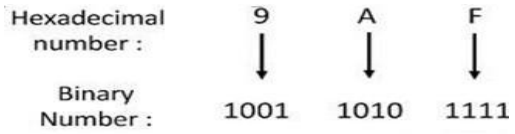
Hexadecimal to Binary Conversion

Steps:

1. Separate the digits of the given Hexadecimal Number, if it contains more than one digit.
2. Find the equivalent Binary Number for each digit of the Hexadecimal Number. Add 0's to the left if any of the Binary equivalent is shorter than 4 bits.
3. Write all the Binary numbers together, maintaining the same group order provides the equivalent Binary for the given Hexadecimal number.

Example:

Convert the Hexadecimal number $(9AF)_{16}$ into its Binary equivalent.



Thus. $(9AF)_{16} = (1001\ 1010\ 1111)_2$

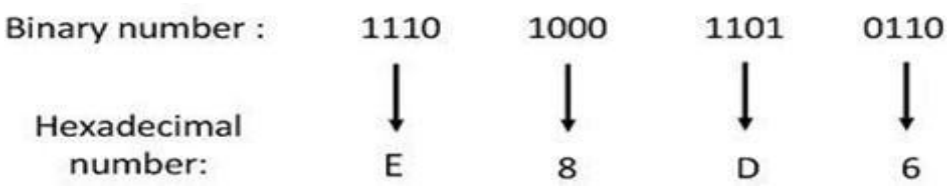
Binary to Hexadecimal Conversion

Steps:

1. Separate the digits of a given binary number into groups of 4 bits starting from right to left.
2. Find the equivalent Hexadecimal number for each group.
3. Write all the Hexadecimal numbers together, maintaining group order provides the equivalent octal number for the given binary.

Example:

Convert the given Binary Number $(1110100011010110)_2$ into its Hexadecimal equivalent.



Thus, $(1110100011010110)_2 = (E8D6)_{16}$

Hexadecimal to Octal Conversion

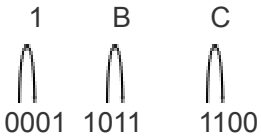
Conversion of Hexadecimal to Octal cannot be done directly.

Steps:

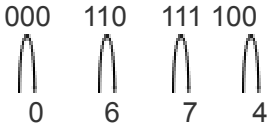
1. For each Hexadecimal digit, write the equivalent binary number. If any of the binary equivalents are less than 4 digits, add 0's to the left side.
2. Combine and make the groups of binary digits from right to left, each containing 3 digits. Add 0's to the left if there are less than 3 digits in the last group.
3. Find the Octal equivalent of each binary group.

Example:

Convert the Hexadecimal Number $(1BC)_{16}$ into its Octal equivalent.



Now group them from right to left, each having 3 digits.



Thus, $(1BC)_{16} = (674)_8$

Octal to Hexadecimal Conversion

Conversion of Octal to Hexadecimal cannot be done directly.
Steps:
Convert the given Octal Number to Binary.
Then Convert the Binary number to Octal.

Example:
Convert the Octal Number $(752)_8$ to Hexadecimal.
Step 1:

Octal to Binary

111 101 010

So the Binary equivalent is 111101010

Step 2:

Binary to Hexadecimal Conversion

0001 1110 1010

1 E A

Thus, $(752)_8$ =

$(1EA)_{16}$

Decimal, Binary, Octal and Hexadecimal Equivalence Table

Decimal	Binary	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	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

Decimal Representation:

- The Binary Number System is the most natural system for a computer.
- A Binary Code is a group of n bits that assumes up to 2^n distinct combinations of 1's and 0's witheach combination representing one element of the set that is being coded.
- Binary Code that distinguishes among 10 elements must contain at least four bits.
- The bits assignment is most commonly used for the decimal digits.
- Table 6 shows the difference between the conversion of decimal numbers into binary and thebinary coding of decimal numbers.

Example:

- When converted to binary number, the decimal number 99 is represented by the string of bits 1100011.
- When represented in BCD, it becomes 1001 1001.

Table 6 Decimal and BCD

Decimal Number	Binary-Coded Decimal (BCD) number
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	0001 0000
20	0010 0000
99	1001 1001

Alphanumeric Representation:

- **An Alphanumeric character set is a set of elements that includes the 10 decimal digits, the 26 letters of the alphabet and a number of special characters, such as $, +, =.**
- Such a set contains between 32 and 64 elements (if only uppercase letters are included) or between 64 and 128 (if both uppercase and lowercase letters are included).
  - In the first case, the binary code will require six bits and in the second case, seven bits.
- **The standard alphanumeric binary code is the ASCII (American Standard Code for Information Interchange) which uses seven bits to code 128 characters.**
  - The binary code for the uppercase letters, the decimal digits, and a few special characters is listed in Table 7.
- Binary Codes play an important part in Digital Computer operations.
- The codes must be in binary because registers can only hold binary information.
- Binary Codes change the symbols only not the meaning of the elements.

Table 7 ASCII

Character	Binary Code	Character	Binary Code
A	100 0001	0	011 0000
B	100 0001	1	011 0001
C	100 0001	2	011 0010
D	100 0100	3	011 0011
E	100 0101	4	011 0100
F	100 0110	5	011 0101
G	100 0111	6	011 0110
H	100 1000	7	011 0111
I	100 1001	8	011 1000
J	100 1010	9	011 1001
K	100 1011	SPACE	010 0000
L	100 1100	.	010 1110
M	100 1101	(	010 1000
N	100 1110	+	010 1011
O	100 1111	$	010 0100
P	101 0000	*	010 1010
Q	101 0001	)	010 1001
R	101 0010	-	010 1101
S	101 0011	/	010 1111
T	101 0100	,	010 1100
U	101 0101	=	011 1101
V	101 0110		
W	101 0111		
X	101 1000		
Y	101 1001		
Z	101 1010		

1.2 Complements

Complements are used in Digital Computers for simplifying the subtraction operation and forlogical manipulation.

There are two types of complements for each base r system: (Figure 1)

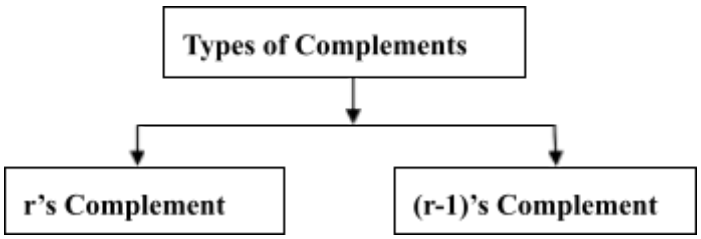


Figure 1 Types of Complements

When the value of the base r is substituted in the name, the two types are referred to as the 2's and I's complement for binary numbers and the ID's and 9's complement for decimal numbers.

(Table 8 shows the relation between r's and (r-1)'s complement.

Table 8 Relation between r's and (r-1)'s complement

Sl. No.	r's Complement	(r-1)'s Complement
1	10's	(10-1 = 9's
2	2's	(2-1) = 1's

(r-1)'s Complement

Given a number N in base r having n digits, the (r-1)'s complement of N is defined as $(r^n - 1) - N$.

For decimal numbers r = 10 and r-1 = 9.

So the 9's complement of N is $(10^n - 1) - N$.

- **The 9's Complement of a decimal number is obtained by subtracting each digit from 9**.

Example 1:

The 9's complement of 28 is 99-28 = 71.

The 9's complement of 12389 is 9999 – 12389 = 87610.

For binary numbers, r = 2 and r – 1 =1, so the 1's complement of N is $(2^n - 1) - N$.

❖ **The 1's Complement of a binary number is obtained by subtracting each digit from 1.**

The 1's complement of a binary number is formed by changing 1's into 0's and 0's into 1's.

The (r - I)'s complement of octal or hexadecimal numbers are obtained by subtracting each digit from 7 or F (decimal 15) respectively.

Example 2:

1’s complement of 1011001 = 0100110

(r’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.

The 10's complement of the decimal number is obtained by adding 1 to the 9's complement value.

The 2's complement of the binary number is obtained by adding 1 to the 1's complement value.

The complement of the complement restores the number to its original value. The r's complement of N is r^n - N. The complement of the complement is r^n - (r^n - N) = N giving back the original number.

Example 1:

The 10’s complement of the Decimal $(7)_{10}$ = 10-7 = 3

Example 2:

F i n d t h e 2’s Complement of the Binary $(0010)_2$ number.

Step 1: Convert the binary into 1’s complement by replacing 1’s by 0’s and 0s, then get
0010 □ 1101

Step 2: Add 1 into 1’s complement 1101 + 1 □ 1110

Step 3: 2’s complement of (0010) = 1110.

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) = M - N + r'.
2. If M "" N, the sum will produce an end carry r' 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 i s equal to r' - (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.

Consider, for example, the subtraction 72532 - 13250 = 59282. The l0's complement of 13250 is 86750. Therefore:

M = 72532
10's complement of N = +86750

Sum = 159282

Discard end carry 10' = - 1 00000
Answer = 59282

Now consider an example with M < N. The subtraction 13250 – 72532 produces negative 59282.

Using the procedure with complements, we have

M = 13250

10's complement of N = +27468

Sum = 40718
There is no end carry

Answer is negative 59282 = 10's complement of 40718

Subtraction with complements is done with binary numbers in a similar manner using the same procedure outlined above. Using the two binary numbers X = 1010100 and Y = 1000011, we perform the subtraction X – Y and Y - X using 2's complements:

X = 1010100
2' s complement of Y = +0111101
Sum = 10010001
Discard end carry 27 = - 10000000
Answer: X - Y = 0010001
y = 1000011

2's complement of X = +0101 100
There is no end carry
Sum = 1101111
Answer is negative 0010001 = 2's complement of 1101111

1.3 **Fixed - Point Representation**

- Positive integers, including zero, can be represented as unsigned numbers.
- To represent negative integers, we need a notation for negative values.
- In ordinary arithmetic, a negative number is indicated by a minus sign and a positive number by aplus sign.

- The convention is to make the sign bit equal to 0 for positive and to 1 for negative.
- A number may have a binary or decimal point.
- The position of the binary point is needed to represent fractions, integers, or mixed integer-fraction numbers.
- There are two ways of specifying the position of the binary point in a register:
  1. **Fixed - Point Representation:** The Binary point is always fixed in one position.
  2. **Floating - Point Representation:** It uses a second register to store a number.

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 thesign of the larger magnitude.

Example 1:

Add (+ 15) , (- 18)

= (+ 15) + (-18)

= (- 3)

- The rule for adding numbers in the signed 2's complement system is very simple.

Example 2:

2's Complement Addition of (+6) and (+13)

+6 = 0000 0110	**-6 = 1111 1010**
+13 = 0000 1101	**+13 = 0000 1101**
+19 = 00010011	**+7 = 00000111**

Arithmetic Subtraction

- Subtraction of two signed binary numbers when negative numbers are in 2's complement formis very simple and can be stated as follows:
  - Take the 2's complement of the subtrahend.
  - Add it to the minuend.
  - A carry out of the sign bit position is discarded.
- Changing a positive number to a negative number is done by taking its 2's complement.

- The reverse is also true because the complement of a negative number in complement form produces the equivalent positive number.

Example:

The subtraction of (-6) - (-13) = +7

+6 = 0000 0110 -6 = 1111 1010
+13 = 0000 1101 -13 = 1111 0011

+7 = 0000 0111

Overflow

When two numbers of n digits each are added and the sum occupies n + 1 digits, then overflow has occurred.

- An overflow is a problem in Digital Computers because the width of registers is finite.
- A result that contains n + 1 bits cannot be accommodated in a register with a standard length of n bits.
- For this reason, many computers detect the occurrence of an overflow.
- When it occurs, a corresponding flip-flop is set which can then be checked by the user.
- The detection of an overflow after the addition of two binary numbers depends on whether thenumbers are considered to be signed or unsigned.
- An overflow cannot occur after an addition if one number is positive and the other is negative.

Decimal Fixed-Point Representation

- The representation of decimal numbers in registers is a function of the binary code used to represent a decimal digit.
- A 4-bit decimal code requires four flip-flops for each decimal digit.
- The representation of 4385 in BCD requires 16 flip-flops, four flip-flops for each digit. The number will be represented in a register with 16 flip-flops as follows: 0100 0011 1000 010I
- The representation of signed decimal numbers in BCD is similar to the representation of signed numbers in binary. We can either use the familiar signed-magnitude system or the signed-complement system. The sign of a decimal number is usually represented with four bits to conform with the 4-bit code of the decimal digits.
-

- The signed-magnitude system is difficult to use with computers. The signed-complement system can be either the 9's or the 10's complement, but the 10's complement is the one most often used. To obtain the 10' s complement of a BCD number, we first take the 9's complement and then add one to the least significant digit. The 9's complement is calculated from the subtraction of each digit from 9.

1.4 **Floating – Point Representation**

- The floating-point representation of a number has two parts.

❖ The first part represents a signed, fixed-point number called the **mantissa**.

❖ The second part designates the position of the decimal point and is called the **exponent**. **Example**:

Table 9 shows the Floating-Point with a fraction and an exponent as follows:

Table 9 Examples for Floating Point Representation

DecimalNumber	Fraction	Exponent
+6132.789	+0.6132789	+04
+1001.11	01001110 (0 in the leftmost position to denote positive)	000100

1.5 **Other Binary Codes**

- Digital Computers also employ other binary codes for special applications.
- A few additional binary codes encountered in Digital Computers are:
  - o Gray Code
  - o Other Decimal Codes
  - o Other Alphanumeric Codes

Gray Code

- **The Gray Code is a sequence of Binary Number System in which every successive pair of numbers differs in only one bit**.
- The Gray Code is also known as **Reflected Binary Code** (**RBC**) or **Reflected Binary.**
- The Gray Code was invented by Frank Gray (1939) for sending sequence of bits using a frequency transmitting device.
- Gray Code has some applications in analog to digital converters, as well as being used for error correction in digital communication.

Gray Code counters are sometimes used to provide the timing sequences that control operations in a digital system.

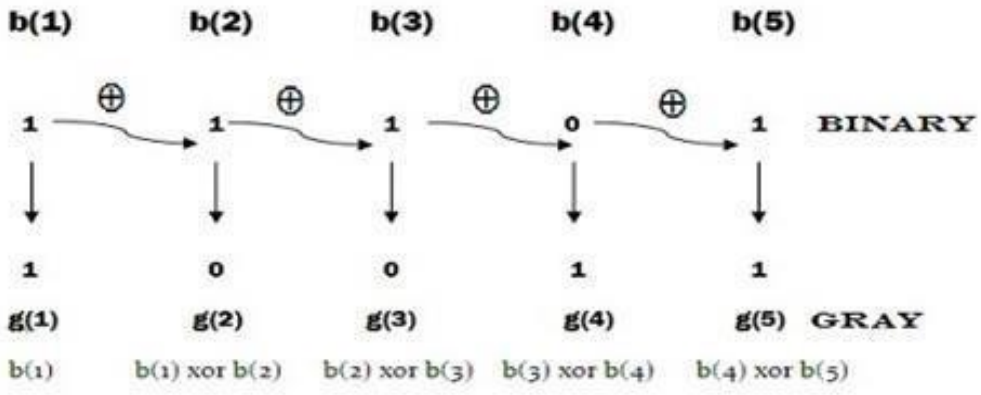
Binary to Gray Conversion

Steps:

1. The MSB of the Gray Code is the same as the MSB of the Binary.
2. Coding from left to right, add each adjacent pair of bits to get the next bit of the Gray Code.
3. Omit the carries if occurs.

Example:

Convert the Binary $(11101)_2$ to its equivalent Gray Code.



The answer is 10011.

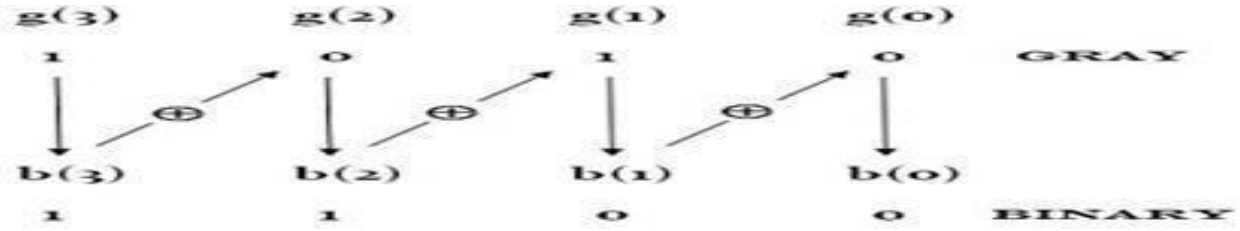
Gray Code to Binary Conversion

Steps:

1. The Most Significant Bit (MSB) of the Binary number is the same as the MSB of the Gray code.
2. Add the next significant bit of the binary number to the next significant bit of the gray coded number to obtain the next binary bit.
3. Repeat step 2 till all bits of the gray coded number have been added.
4. The resultant number is the binary equivalent of the gray number.

Example:

Convert the Gray Code 1010 to its equivalent Binary.



i.e
$b(3) = g(3)$
$b(2) = b(3) \oplus g(2)$
$b(1) = b(2) \oplus g(1)$
$b(0) = b(1) \oplus g(0)$

The answer is 1100.

Other Decimal Codes

- Binary Codes for decimal digits require a minimum of four bits.
- The 2421 is an example of a weighted code.
- In a weighted code, the bits are multiplied by the weights indicated and the sum of the weightedbits given the decimal digit.
- For Example: The bit combination 1101, when weighted by the respective digits 2421,

  2 x 1 + 4 x 1 + 2 x 0 + 1 x 1 = 7
- The excess-3 code is a decimal code that has been used in older computers.
- This is an un-weighted code.

Other Alphanumeric Codes

- The ASCII code is the standard code commonly used for the transmission of binary information.
- Each character is represented by a 7-bit code and usually an eighth bit is inserted for parity.
- The code consists of 128 characters.
- Alphanumeric Code used in IBM equipment is the EBCDIC (Extended BCD Interchange Code).
- It uses eight bits for each character.
- EBCDIC has the same character symbols as ASCII but the bit assignment to characters isdifferent.
- When alphanumeric characters are used internally in a computer for data processing.

.EXERCISES:

1. Convert the following Decimal Numbers to Binary.

a) $(120)_{10}$ b) $(24.8)_{10}$

2. Convert the following Binary Numbers to Decimal.
   a) $(11011)_2$ b) $(101011.11)_2$
2. Convert the following Hexadecimal Numbers to Decimal. a) $(4CE)_{16}$ b) $(8B.A)_{16}$
3. Convert the following Decimal Numbers to Hexadecimal. a) $(48)_{10}$ b) $(27.7)_{10}$
4. Convert the following Hexadecimal Numbers to Binary. a) $(C7)_{16}$ b) $(FF.A)_{16}$
5. Convert the following Binary Numbers to Hexadecimal. a) $(111001)_2$ b) $(11101.01)_2$
6. Convert the following Octal Numbers to Decimal. a) $(24)_8$ b) $(23.43)_8$
7. Convert the following Decimal numbers to Octal. a) $(99)_{10}$ b) $(19.6)_{10}$
8. Convert the following Octal Numbers to Binary.
   a) $(132)_8$ b) $(76.9)_8$
9. Convert the following Binary Numbers to Octal. a) $(10101)_2$ b) $(110.11)_2$
10. Perform the following additions in the binary number system. a) +95 and +27 b) -95 and -27
11. Perform the following subtraction in the binary number system.
    a) 4 from 13 b) 14 from 17
12. Give the 1's and 2's complement forms for the following binary numbers.
    a) 11011 b) 11001 c) 101011
13. Convert the following decimal numbers into their 9's and 10's complements.
    a) 6 b) 34 c) 99

****

****

Chapter 2 Register Transfer and Microoperations

****

This Chapter introduces a Register Transfer Language and shows how it is used to express microoperations in symbolic form. Symbols are defined for arithmetic, logic, and shift microoperations.

2.1 Register Transfer Language

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.

They are low-level instructions used in some designs to implement complex machine instructions.

Examples of microoperations are shift, count, clear, and load.

For example, a counter with parallel load is capable of performing the microperations increment and load.

A bidirectional shift register is capable of performing the shift right and shift left microoperations.

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.

Register Transfer Language (RTL) is used to describe the microoperations transfer amongregisters.

It is a kind of Intermediate Representation (IR) that is near to assembly language.

The term “Register Transfer” can perform microoperations and transfer the result of operation tothe same or other register.

2.2 Register Transfer

Computer registers are designated by capital letters 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).

The individual flip-flops in an n-bit register are numbered in sequence from 0 through n - 1, starting from 0 in the rightmost position and increasing the numbers toward the left.

Figure 2.1 shows the representation of registers in block diagram form. The most common way to represent a register is by a rectangular box with the name of the register inside, as in Fig. 2.1(a). The individual bits can be distinguished as in (b). The numbering of bits in a 16-bit register can be marked on top of the box as shown in (c). A 16-bit register is partitioned into two parts in (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 (S-15) or PC (H) to the high-order byte.

The Information transformed from one register to another register is represented in symbolic form by replacement operator is called Register Transfer.

The

Statement R2

<--R1

denotes the transfer of content of register R1 into register R2.
It designates a replacement of the content of R2 by the content of R l.

- By definition, the content of the source register R1 does not change after the transfer.

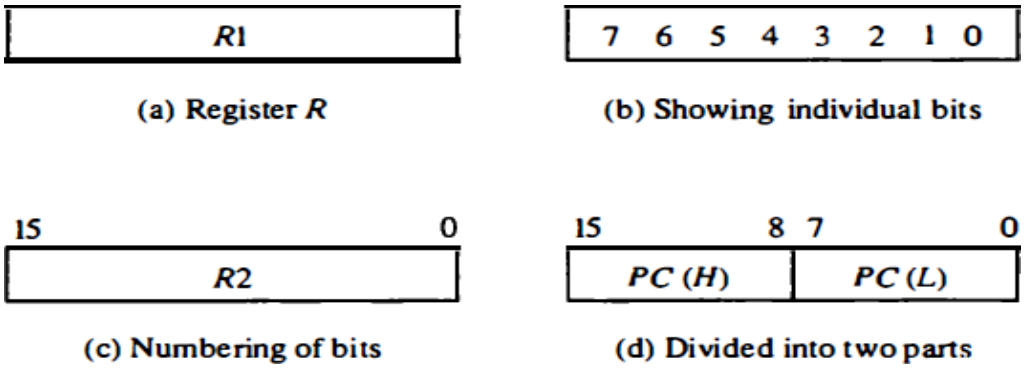


Figure 2.1 Block diagram of register

The basic symbols of the register transfer notation are listed in Table 2.1.

Table 2.1 Basic Symbols for Register Transfers

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	*MAR*, *R2*
Parentheses ( )	Denotes a part of a register	*R2*(0–7), *R2*(*L*)
Arrow ←	Denotes transfer of information	*R2* ← *R1*
Comma ,	Separates two microoperations	*R2* ← *R1*, *R1* ← *R2*

Different types of Register Transfer operations:

1. **Simple Transfer R2 R1:**
   a. The content of R1 is copied into R2 without affecting the content of R1.
   b. It is an unconditional type of transfer operation.
2. **Conditional Transfer**
   - **Figure 2.2** shows the block diagram that depicts the transfer from R1 to R2.
   - The n outputs of register R1 are connected to the n inputs of register R2.
   - The letter n indicates any number of bits for the register.
   - Register R2 has a load input that is activated by control variable P.
   - Control variable is synchronized with the same clock as the one applied to the register.

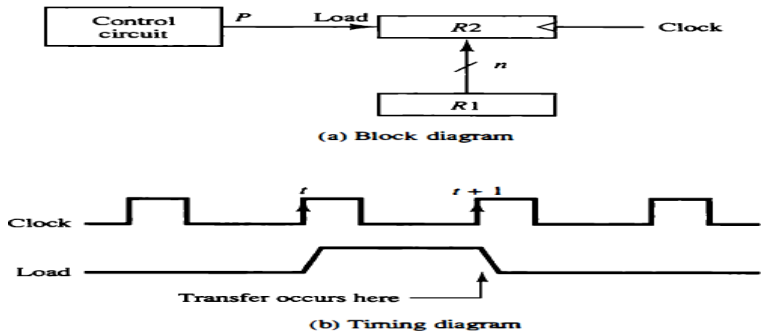


Figure 2.2 Block Diagram, Transfer from R1 to R2 when P=1

2.3 Bus and Memory Transfers

- **A bus is a group of wires through which binary information is transferred among registers**.
- Control signals determine which register is selected by the bus during each particular registertransfer.
- One way of constructing a common bus system is with multiplexers.
- The multiplexers select the source register whose binary information is then placed on the bus.
- The construction of a bus system for four registers is shown in Figure 2.3.
- Each register has four bits.
- The bus consists of four 4 X 1 multiplexers each having four data inputs.
- Two selection inputs, S_1 and S_0.

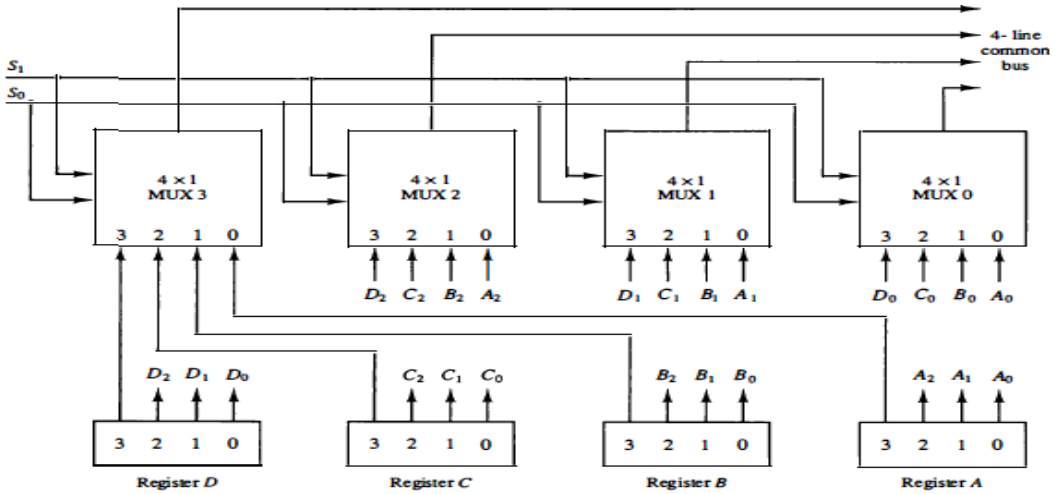


Figure 2.3 Bus Systems for four registers

- The output 1 of register A is connected to input 0 of MUX 1and similarly other connections aremade as shown in the diagram.
- The data transferred to the bus depends upon the selection lines.
- Table 2.3 denotes the combinations of selection lines are shown below.

Table 2.3 Function Table for Bus of Figure 2.3

S1	S2	Register Selected
0	0	A
0	1	B
1	0	C
1	1	D

i) **Three-State Bus Buffers**

- A bus system can be constructed with three-state gates instead of multiplexers.
- A three-state gate is a digital circuit that exhibits three states.
- Two of the states are signals equivalent to logic 1 and 0 as in a conventional gate.
- The third state is a high-impedance state.
- The high-impedance state behaves like an open circuit.
- Open circuit means that the output is disconnected and does not have logic significance.
- Three- state gates may perform any conventional logic, such as AND or NAND.
- Most commonly used in the design of a bus system is the buffer gate.

ii) Memory Transfer

- The transfer of information from a memory is called ***read operation.***
- The transfer of new information to be stored into the memory is called ***write operation.***
- A memory symbolized by the letter M.

- The particular memory is selected by the memory address during the transfer.
- A memory unit that receives the address from a register, called the ***Address Register (AR).***
- The data are transferred to another register, called the ***Data Register (DR).***
- The read operation can be stated as follows: Read

  : DR M [AR]
- The write operation can be

  stated as followsWrite: M [AR]

  R1.

2.4 Arithmetic Microoperations

Arithmetic microoperations perform arithmetic operations on numeric data stored in registers.

Table 2.4 lists the various Arithmetic microoperations performed on the registers.

Table 2.4 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

Addition:

In addition microoperation, the value in register R1 is added to the value in the register R2 andthen the sum is transferred into register R3.

R3 R1 + R2

Subtraction:

In subtraction microoperation, the contents of register R2 are subtracted from contents of theregister R1, and then the result is transferred into R3.

R3 R1 - R2

Increment:

In increment micro-operation, the value inside the R1 register is increased by 1.R1 R1 + 1

Decrement:

In Decrement micro-operation, the value inside the R1 register is decreased by 1.R1 R1 – 1

i) Binary Adder

- **The digital circuit that generates the arithmetic sum of two binary numbers of any length s called a binary adder.**
- The binary adder is constructed with full-adder circuits connected in cascade, with the output carry from one full-adder connected to the input carry of the next full-adder.
- Figure 2.4 shows the interconnections of four full-adders (FA) to provide a 4-bit binary adder.
- The augend bits of A and the addend bits of B are designated by subscript numbers from right to left, with subscript 0 denoting the low-order bit.
- The carries are connected in a chain through the full-adders.
- The input carry to the binary adder is C0 and the output carry is C,.
- The S outputs of the full-adders generate the required sum bits.
- An n-bit binary adder requires n full-adders.
- The output carry from each full-adder is connected to the input carry of the next-high-order full-adder.
- The n data bits for the A inputs come from one register (such as R1), and the n data bits for the B inputs come from another register (such as R2).
- The sum can be transferred to a third register or to one of the source registers (R1 or R2), replacing its previous content.

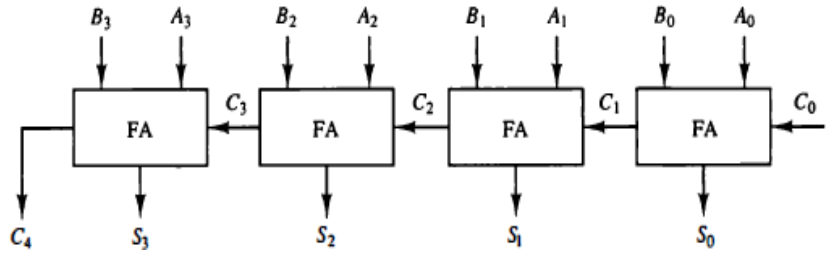


Figure 2.4 4-bit Binary Adder

ii) Binary Adder-Subtractor

The addition and subtraction operations can be combined into one common circuit by including an exclusive-OR gate with each full-adder.

A 4-bit Adder-Subtractor circuit is shown in Figure 2.5.

The mode input M controls the operation.

When M = 0 the circuit is an adder and when M = 1 the circuit becomes a subtractor.

Each exclusive-OR gate receives input M and one of the inputs of B. When M = 0, we have B a1 0 = B.

The full-adders receive the value of B, the input carry is O, and the circuit performs A plus B.

When M = 1, we have B a11 = B' and C0 = 1. The B inputs are all complemented and a 1 is added through the input carry.

The circuit performs the operation A plus the 2's complement of B.
For unsigned numbers, this gives A - B if A '" B or the 2's complement of (8 - A) if A < B.
For signed numbers, the result is A – B provided that there is no overflow.

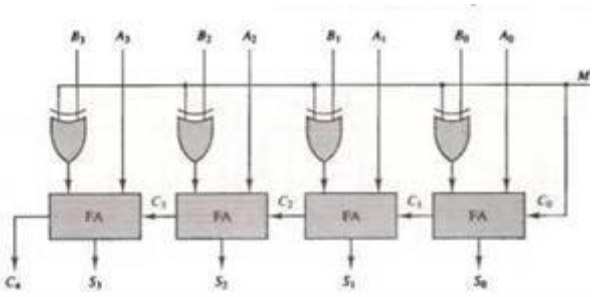


Figure 2.5 4-bit Adder-Subtractor.

iii) Binary lncrementer

The increment microoperation adds one to a number in a register.

For example, if a 4-bit register has a binary value 0110, it will go to 0111 after it is incremented.

This microoperation is easily implemented with a binary counter.

Every time the count enable is active, the clock pulse transition increments the content of the register by one.

There may be occasions when the increment microoperation must be done with a combinational circuit independent of a particular register.

This can be accomplished by means of half-adders connected in cascade.

The diagram of a 4-bit combinational circuit incrementer is shown in Figure 2.6.

One of the inputs to the least significant half-adder (HA) is connected to logic-1 and the other input is connected to the least significant bit of the number to be incremented.

The output carry from one half-adder is connected to one of the inputs of the next-higher-order half-adder.

The circuit of Figure 2.6 can be extended to an n-bit binary incrementer by extending the diagram to include n half-adders.

The least significant bit must have one input connected to logic-1. The other inputs receive the number to be incremented or the carry from the previous stage.

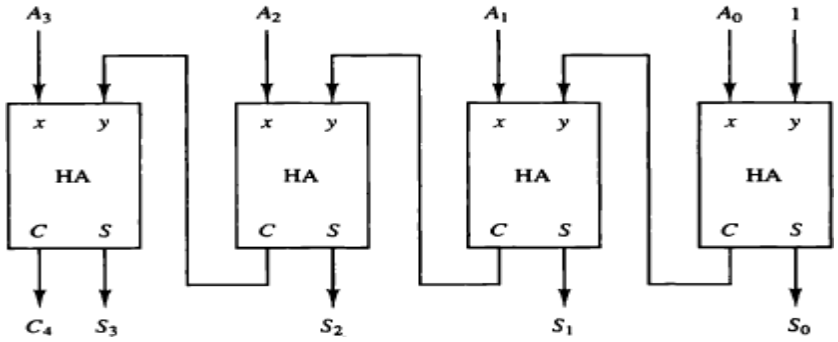


Figure 2.6 4 bit Binary Incrementer.

2.5 **Logic Microoperations**

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

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

- It specifies a logic microoperation to be executed on the individual bits of the registers provided that the control variable P = 1.

Assume that each register has 4bits.

1 0 1 0	Contentof R1
1 1 0 0	Content of R2
0 1 1 0	Content of R1 after P = 1

- The content of R1, after the execution of the microoperation, is equal to the bit-by-bit exclusive-OR operation on pairs of bits in R2 and previous values of R 1 .
- The logic microoperations are seldom used in scientific computations, but they are very useful for bit manipulation of binary data and for making logical decisions.
- Table 2.5 lists the various Logic Microperations.

Table 2.5 Logic Microoperations

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

2.6 **Shift Microoperations**

- **Shift Microoperations are used for serial transfer of data.**
- They are also used in conjunction with arithmetic, logic, and other data-processing operations.
- The contents of a register can be shifted to the left or the right.
- At the same time that the bits are shifted, the first flip-flop receives its binary information fromthe serial input.
- During a shift-left operation the input transfers a bit into the rightmost position.
- During a shift-right operation the input transfers a bit into the leftmost position.
- The information transferred through the serial input determines the type of shift.
- There are three types of shifts micro-operations**:**

1. Logical Shift

- **Logical Shift Left**
- **Logical Shift Right**

2. **Arithmetic Shift**

3. **Circular**

Shift Logical

Shift

- Logical shift transfers the 0 through the serial input.
- **Logical Shift Left:**
  - Figure 2.7(a) shows the Logical Shift Left Microoperation.
  - In this shift one position moves each bit to the left one by one.

- The Empty Least Significant Bit (LSB) is filled with zero
- The Most Significant Bit (MSB) is rejected.

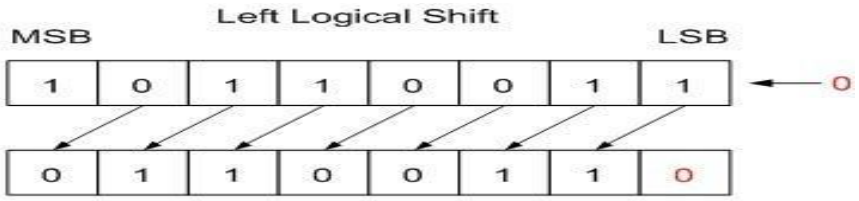


Figure 2.7(a) Logical Shift Left

- **Logical Shift Right:**
  - Figure 2.7(b) shows the Logical Shift Right Microoperation.
  - In this one position moves each bit to the right one by one.
  - Most Significant Bit (MSB) is filled with zero.
  - Least Significant Bit (LSB) is rejected.

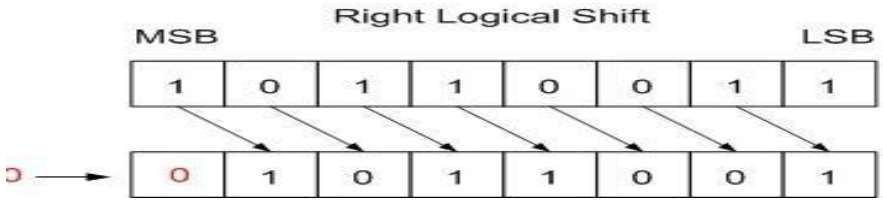


Figure 2.7(b) Logical Shift Right

Arithmetic Shift Microoperation

- Arithmetic Shift Microoperation shifts a signed binary number to the left or to the right position.
- This Microoperation is divided into two categories:
  1. Arithmetic Shift Left
  2. Arithmetic Shift Right
  - **Arithmetic Shift Left:**
    - Figure 2.7(c) shows the Arithmetic Shift Left Microoperation.
    - In this one position moves each bit to the left one by one.
    - An Arithmetic Shift Left multiplies a signed binary number by 2.
    - The leftmost bit in a register holds the sign bit.
    - The reaming bits hold the number.
    - The sign bit is 0 for positive and 1 for negative.
    - Most Significant Bit (MSB) is rejected.

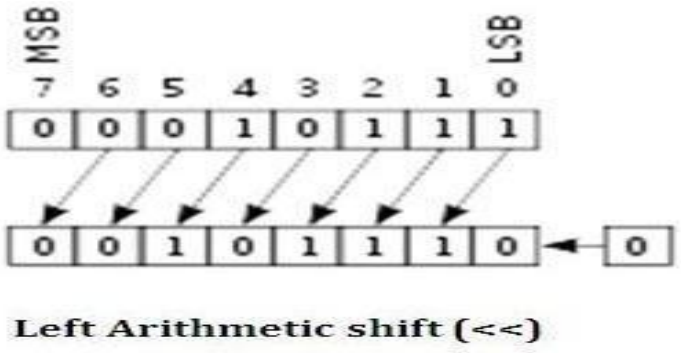


Figure 2.7(c) Arithmetic Shift Left

- **Arithmetic Shift Right**
  - Figure 2.7(d) shows the Arithmetic Shift Right Microoperation.
  - In this one position moves each bit to the right one by one.
  - Least Significant Bit (LSB) is rejected.

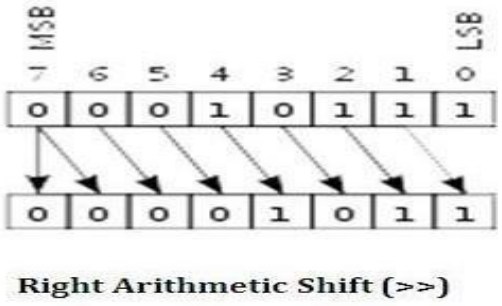


Figure 2.7(d) Arithmetic Shift Right

Circular Shift Micro-operation (Rotate Operation)

- The circular shift circulates the bits in the sequence of the register around the both endswithout any loss of information.
- This operation connecting the serial output of the shift registers to its serial input.
- Figure 2.7 (e) and (f) are the Circular Shift Left and Right Operation.
- The Circular Shift Operation is divided into two categories:

1. **Circular Shift Left**

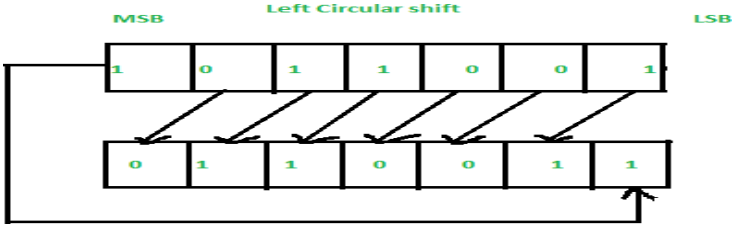


Figure 2.7 (e) Circular Shift Left

2. **Circular Shift Right**

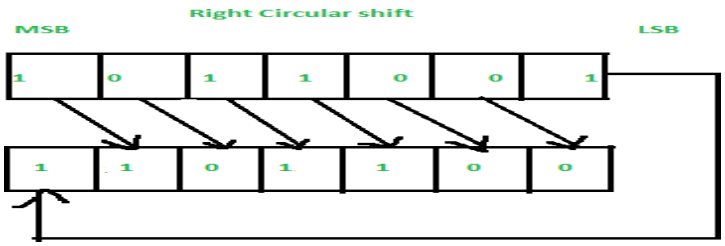


Figure 2.7 (f) Circular Shift Right

EXERCISES:

1. What is meant by Microoperation in Computer Organization?
2. What is Register Transfer Language?
3. Describe about Register Transfer in Computer Organization with neat Block Diagram
4. Write the basic symbols for Register Transfer.
5. Explain about Bus and Memory Transfers in Computer Organization.
6. Describe about Microoperations in Computer Organization.
7. Explain with an example for Logic Microoperations.
8. Describe about Shift Microoperation with suitable diagram.

****

UNIT – II

****

Chapter 3 Central Processing Unit

****

The main part of the Computer that performs the Data Processing is called the Central Processing Unit and is referred to as the CPU. The Central Processing Unit consists of three main parts, such as, Register Set, ALU, and Control Unit. The Register Set stores intermediate data used during the execution of the instructions. Arithmetic and Logic Unit performs the microoperations for executing the instructions. The control unit supervises the transfer of information among the registers and instructs the ALU as to which operation to perform.

3.1 General Register Organization

- The CPU consists of large number of registers.
- It is most efficient to connect them through a common bus system.
- The registers communicate with each other not only for direct data transfers, but also whileperforming various micro-operations.
- It can perform all the arithmetic, logic, and shift microoperations in the processor.
- A bus organization for seven CPU registers is shown in Figure 3.1.
- The output of each register is connected to two multiplexers (MUX) to form the two buses A andB.
- The Selection lines in each multiplexer select one register or the input data for the particular bus.
- The A and B buses form the inputs to a common arithmetic logic unit (ALU).
- The operation selected in the ALU determines the arithmetic or logic microoperation that is to be performed.
- The result of the microoperation is available for output data and also goes into the inputs of all the registers.
- The register that receives the information from the output bus is selected by a decoder.
- The decoder activates one of the register.
- The control unit that operates the CPU bus system directs the information flow through theregisters and ALU.
- The control must provide binary selection variables to the following selection inputs:

1. MUX A selector (SELA) : To place the content of R2 into bus A.
2. MUX B selector (SELB) : To place the content of R3 into bus B.
3. ALU operation selector (OPR) : To provide the arithmetic addition A+B.
4. Decoder destination selector (SELD) : To transfer the content of the output bus into R1.

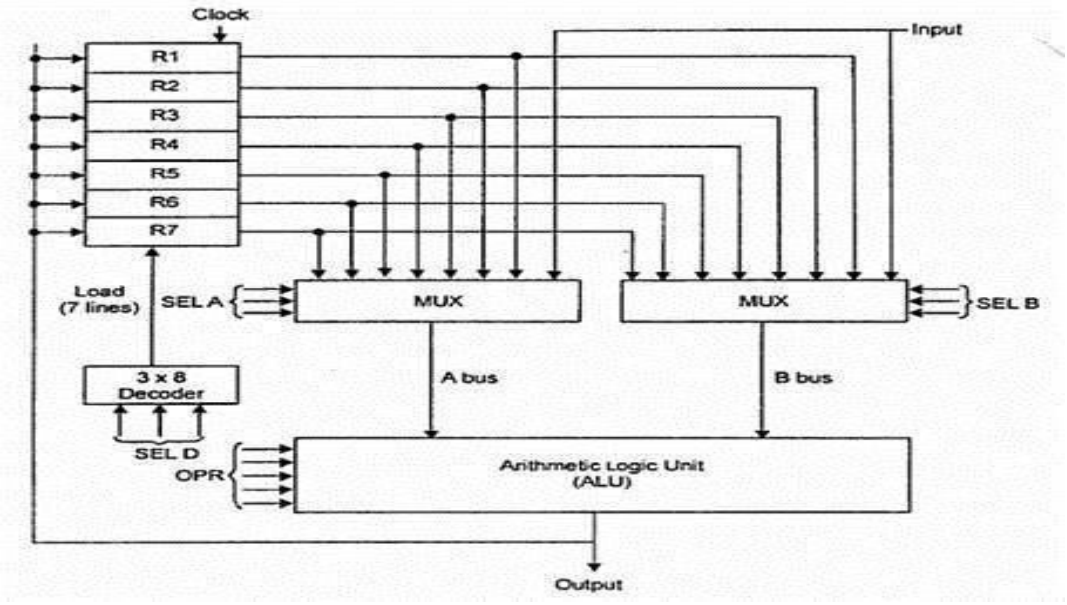


Figure 3.1 Bus Organization for seven CPU Register

3.2 Stack Organization

- **A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved.**
- Stack is a Last-In, First-Out (LIFO) list.
- The operation of a stack can be compared to a stack of trays.
- The last tray placed on top of the stack is the first to be taken off.
- The register that holds the address for the stack is called a stack pointer (SP).
- Figure 3.2 shows the organization of a 64-word register stack.

Stack Operations

- The two operations of a stack are

1. **PUSH:** Insert an item on the top of stack.
2. **POP:** Deletes an item from top of stack

Implementation of Stack

- **Stack** can be implemented in two ways:

i) Register Stack

ii) **Memory Stack**

iii) **Reverse Polish Notation**

iv) **Evaluation of Arithmetic Expressions**

i) Register Stack

o A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or registers.

o The stack pointer register SP contains a binary number whose value is equal to theaddress of

the word that is currently on top of the stack.

o Three items are placed in the stack: A, B & C.

o Item C is on top of the stack.

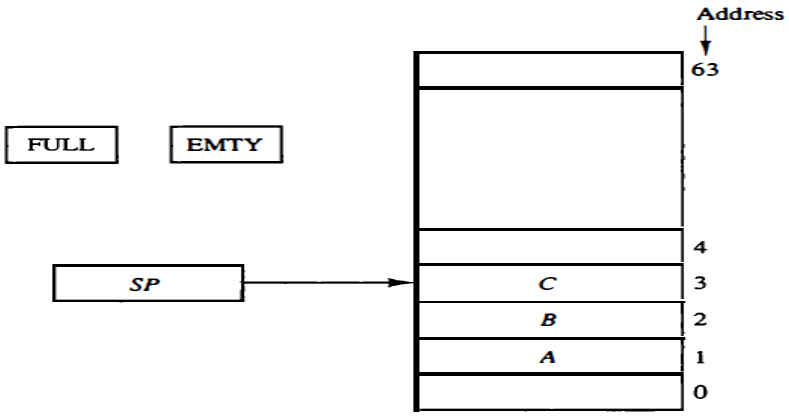


Figure 3.2 Block Diagram of a 64-word stack

PUSH

- The one-bit register FULL is set to 1 when the stack is FULL.
- The one-bit register EMPTY is set to 1 when the stack is empty.
- DR is the Data Register that holds the data to be written into or read from the stack.
- Initially, SP is cleared to 0, EMTY is set to 1, and FULL is cleared to 0.
- If the stack is not full (if FULL = 0), a new item is inserted with a push operation. The push operation is implemented with the following sequence of microoperations;

$SP \leftarrow SP + 1$	Increment stack pointer
$M[SP] \leftarrow DR$	Write item on top of the stack
If $(SP = 0)$ then $(FULL \leftarrow 1)$	Check if stack is full
$EMTY \leftarrow 0$	Mark the stack not empty

- The stack pointer is incremented so that it points to the address of the next-higher word.
- A memory write operation inserts the word from DR into the top of the stack.
- SP holds the address of the top of the stack and that M [SP] denotes the memory word specified by the address presently available in SP.
- The first item stored in the stack is at address L The last item is stored at address 0. If SP reaches 0, the stack is full of items, so FULL is set to L

POP

- A new item is deleted from the stack if the stack is not empty (if EMTY = 0).
- The pop operation consists of the following sequence of microoperations:

$DR \leftarrow M[SP]$	Read item from the top of stack
$SP \leftarrow SP - 1$	Decrement stack pointer
If $(SP = 0)$ then $(EMTY \leftarrow 1)$	Check if stack is empty
$FULL \leftarrow 0$	Mark the stack not full

- The top item is read from the stack into DR.
- The stack pointer is then decremented.
- If its value reaches zero, the stack is empty, so EMTY is set to 1 This condition is reached if the item read was in location 1
- Once this item is read out, SP is decremented and reaches the value 0, which is the initial value of SP.
- An erroneous operation will result if the stack is pushed when FULL = 1 or popped when EMTY = 1

ii) Memory Stack

A Stack can exist as a stand-alone unit a s shown in Figure 3.2 or can be implemented in a random-access memory attached to a CPU.

The implementation of stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer.

Figure 3.3 shows a portion of computer memory stack partitioned into three segments:

- **Program**
- **Data**
- **Stack**

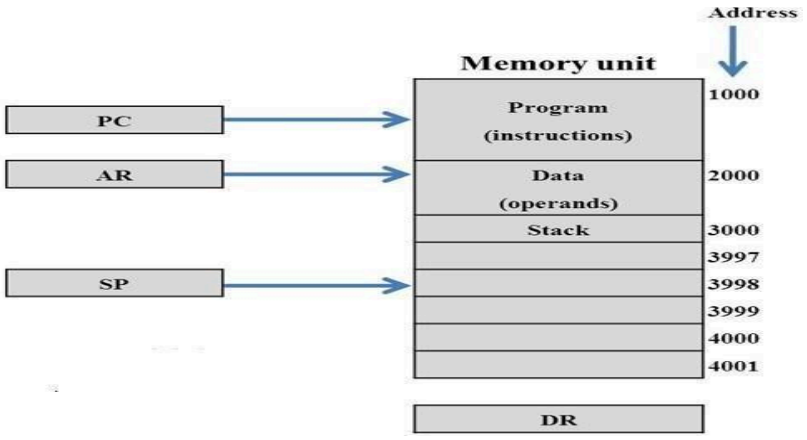


Figure 3.3 Computer Memory Stack

The **Program Counter (PC)** points at the address of the next instruction in the program. It is used during the fetch phase to read an instruction.

The **Address Register (AR)** points at an array of data. It is used during the execute phase to read

an operand.

The **Stack Pointer (SP)** points at the top of the stack. It is used to **Push or Pop** items into the stack. A stack pointer is loaded with an in initial value. This initial value must be at the bottom address of an assigned stack in memory. SP is automatically decremented or incremented with ever push or pop operation.

Assume that the item in the stack with data register (DR).

A new item is inserted with the push operation as follows: SP SP – 1

M[SP] DR

- **The stack pointer is decremented.**
- A memory write operation inserts the word from DR into the top of the stack.

A new item is deleted with a pop operation as

follows: DR M[SP]

SP SP + 1

- **The stack pointer is incremented**
- The top item is read from the stack into DR.

iii) Reverse Polish Notation

Reverse Polish notation (RPN) is a method for representing expressions in which the operator symbol is placed after the arguments being operated on.

- The Polish mathematician Lukasiewicz showed that arithmetic expressions can be represented in prefix notation. This representation, often referred to as **Polish Notation, places the operator before the operands**. The postfix notation, referred to as **Reverse Polish Notation (RPN), places the operator after the operands.**
- The following examples demonstrate the three representations: A + B Infix notation

  + AB Prefix or Polish notation

  AB + Postfix or reverse Polish notation
- The reverse Polish notation is in a form suitable for stack manipulation. The expression A * B + C * D

is written in reverse Polish notation as

AB * CD * +

iv) Evaluation of Arithmetic Expressions

- Reverse Polish notation, combined with a stack arrangement of registers, is the most efficient way known for evaluating arithmetic expressions.

- This procedure is employed in some electronic calculators and also in some computers.
- The stack is particularly useful for handling long, complex problems involving chain calculations.
- It is based on the fact that any arithmetic expression can be expressed in parentheses-free Polish notation.
- The procedure consists of first converting the arithmetic expression into its equivalent reverse Polish notation.
- The operands are pushed into the stack in the order in which they appear.
- The following rnicrooperations are executed with the stack when an operation is entered in a calculator or issued by the control in a computer:
  (1) the two topmost operands in the stack are used for the operation, and
  (2) the stack is popped and the result of the operation replaces the lower operand.
- By pushing the operands into the stack continuously and performing the operations as defined above, the expression is evaluated in the proper order and the final result remains on top of the stack.
- The following numerical example may clarify this procedure. Consider the arithmetic expression

  (3 * 4) + (5 * 6)

- In reverse Polish notation, it is
  expressed as 34 * 56 * +
- Now consider the stack operations shown in Figure 3.4.

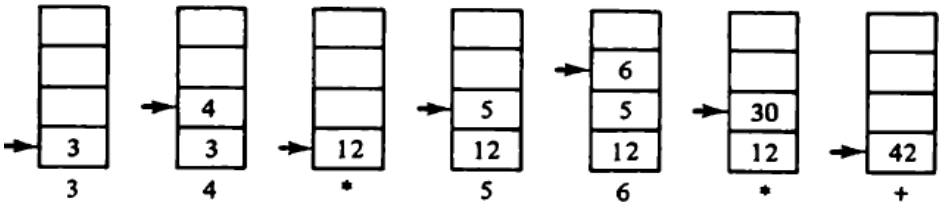


Figure 3.4 Stack operations to evaluate 3 • 4 + 5 • 6.

3.3 Instruction Formats

- A computer usually has a variety of Instruction Code Format.
- Figure 3.4 shows the Instruction Format with Mode Field.
- It is the function of the control unit within the CPU.
- The bits of the instruction are divided into groups called fields.
- The most common fields in Instruction Formats are:

- **Operation Code:** Specifies the operation to be performed.
- **Address Field :** Designates a memory address.
- **Mode Field :** The effective address is determined.

Figure 3.4 Instruction Format

Most computers fall into one of three types of CPU organizations:

1. **Single Accumulator Organization**
2. **General Register Organization**
3. **Stack Organization**

1. **Single Accumulator Organization**

- All operations are performed with an implied accumulator register.
  - The instruction format in this type of computer uses **one address field.**
  - Example: **ADD X**
  - X - Address of the operand.
  - ADD - instruction
  - The ADD instruction in this case results in the operation
  - **AC AC + M[X]**

2. **General Register Organization**

- The instruction format in this type of computer needs **three register address fields.**
  - Example: **ADD R1, R2, R3**
  - To denote the operation **R1 R2 + R3.**

3. Stack Organization

- **Stack organization** would have PUSH and POP instructions which require an address field.
  - **The instruction PUSH X**
  - Push the word at address X to the top of the stack.
    - ❖ The stack pointer is updated automatically.
    - ❖ The instruction **ADD** in a stack computer consists of an operation code only with noaddress field.
    - ❖ This operation has the effect of popping the two top numbers from the stack, adding thenumbers, and pushing the sum into the stack.

Three- Address Instructions:

- **Three-address instruction** formats can use each address field to specify eithera processor register or a memory operand.

- The program in assembly language that evaluates X= (A+B) * (C+D)

ADD R1, A, B R1 ← M[A] + M[B]

ADD R2, C, D R2 ← M[C] + M[D]

MUL X, R1, R2 M[X] ← R1 + R2

Two-Address Instructions:

- Two-address instructions are the most common in commercial computers.
- Each address field can specify either a processor register or a memory word.
- The program to evaluate X = (A+B) * (C+D) is as follows:

MOV	**R1, A**	**R1←M[A]**
ADD	**R1, B**	**R1←R1 +M[B]**
MOV	**R2, C**	**R2←M[C]**
ADD	**R2, D**	**R2←R2+M[D]**
MUL	**R1, R2**	**R1←R1, R2**
MOV	**X, R1**	**M[X] ←R1**

- **The MOV** instruction moves or transfers the operands to and from memory and processor registers.

One-Address Instructions:

- One-address instruction uses an implied accumulator (AC) register for all data manipulation.
- For multiplication and division there is a need for a second register.
- The program to evaluate X = (A + B) * (C + D) is as follows:

LOAD	**A**	**AC**	←	**M[A]**
ADD	**B**	**AC**	←	**AC + M[B]**
STORE	**T**	**M[T]**	←	**AC**
LOAD	**C**	**AC**	←	**M[C]**
ADD	**D**	**AC**	←	**AC + M[D]**
MUL	**T**	**AC**	←	**AC * M[T]**
STORE	**X**	**M[X]**	←	**AC**

- All operations are done between the AC register and a memory operand.
- T is the address of a temporary memory location required for storing the intermediate result.

Zero-Address Instructions:

- Stack organized computer does not use an address field for the instructions ADD and MUL.
- The PUSH and POP instructions need an address field to specify the operand that

communicates with the stack.

- The following program shows how X = (A + B) * (C + D) w i l l be written for astack organized computer.

PUSH	**A**	**TOS**	**←**	**A**
PUSH	**B**	**TOS**	**←**	**B**
ADD		**TOS**	**←**	**(A + B)**
PUSH	**C**	**TOS**	**←**	**C**
PUSH	**D**	**TOS**	**←**	**D**
ADD		**TOS**	**←**	**(C + D)**
MUL		**TOS**	**←**	**(C + D) * (A + B)**
POP	**X**	**M[X]**	**←**	**TOS**

- The name "zero address" is given to this type of computer because of the absence of an address field in the computational instructions.

3.4 Addressing Modes

- The operation field of an instruction specifies the operation to be performed.
- This operation must be executed on some data stored in computer registers or memory words.
- The way the operands are chosen during program execution is dependent on the addressing modeof the instruction.
- **Addressing Mode specifies a rule for interpreting or modifying t he address field of theinstruction before the operand is actually referenced.**
- Computers use addressing mode techniques for the purpose of accommodating one or both of the following provisions:
- To give programming versatility to the user by providing such facilities as
  - **Pointers to memory**
  - **Counters for loop control,**
  - **Indexing of data**
  - **Program relocation**
- To reduce the number of bits in the addressing field of the instruction.
- The control unit of a computer is designed to go through an instruction cycle that is divided into three major phases:
  - **Fetch the instruction from memory**
  - **Decode the instruction**
  - **Execute the instruction**

Program Counter (PC):

- There is one register in the computer called Program Counter (PC) that the instruction in the program stored in memory.
- **PC holds the address of the instruction to be executed next and is incremented each time an instruction is fetched from memory.**

Type of Addressing Modes

1. **Implied Mode:**

- In this mode the operands are specified implicitly in the definition of the instruction.
- For example, the instruction "complement accumulator" is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction.
- In fact, all register reference instructions that use an accumulator are implied-mode instructions.
- Zero-address instructions in a stack-organized computer are implied-mode instructions since the operands are implied to be on top of the stack.

2. Immediate Mode:

- In this mode the operand is specified in the instruction itself.
- In other words, an immediate-mode instruction has an operand field rather than an address field.
- The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction.
- Immediate-mode instructions are useful for initializing registers to a constant value.

3. Register Mode:

- In this mode the operands are in registers that reside within the CPU. The particular register is selected from a register field in the instruction.
- A k-bit field can specify any one of 2' registers.

4. **Register Indirect Mode**:

- In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory.
- In other words, the selected register contains the address of the operand rather than the operand itself.
- Before using a register indirect mode instruction, the programmer must ensure that the memory address of the operand is placed in the processor register with a previous instruction. A reference to the register is then equivalent to specifying a memory address.

- The advantage of a register indirect mode instruction is that the address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.

5. **Autoincrement or Autodecrement Mode**:

- This is similar to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory.
- When the address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table.
- This can be achieved by using the increment or decrement instruction.
- The address field of an instruction is used by the control unit in the CPU to obtain the operand from memory.
- The effective address is defined to be the memory address obtained from the computation dictated by the given addressing mode. **The effective address is the address of the operand in a type instruction**. It is the address where control branches in response to a branch-type instruction.

6. Direct Addressing Mode

- In this mode the effective address is equal to the address part of the instruction.
- The operand resides in memory and its address is given directly by the address field of the instruction.
- In a branch-type instruction the address field specifies the actual branch address.

7. Indirect Addressing Mode

- In this mode the address field of the instruction gives the address where the effective address is stored in memory.
- Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.
- The effective address in these modes is obtained from the following computation:
  effective address = address part of instruction + content of CPU register
- The CPU register used in the computation may be the program counter, an index register, or a base register.

8. Relative Addressing Mode

- In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address.

- The address part of the instruction is usually a signed number (in 2's complement representation) which can be either positive or negative.
- When this number is added to the content of the program counter, the result produces an effective address whose position in memory is relative to the address of the next instruction.

9. Indexed Addressing Mode

- In this mode the content of an index register is added to the address part of the instruction to obtain the effective address.
- The index register is a special CPU register that contains an index value.
- The address field of the instruction defines the beginning address of a data array in memory.
- Each operand in the array is stored in memory relative to the beginning address.
- The distance between the beginning address and the address of the operand is the index value stored in the index register.
- Any operand in the array can be accessed with the same instruction provided that the index register contains the correct index value.
- The index register can be incremented to facilitate access to consecutive operands.

10. Base Register Addressing Mode

- In this mode the content of a base register is added to the address part of the instruction to obtain the effective address.
- This is similar to the indexed addressing mode except that the register is now called a base register instead of an index register.
- The difference between the two modes is in the way they are used rather than in the way that they are computed.
- An index register is assumed to hold an index number that is relative to the address part of the instruction.
- A base register is assumed to hold a base address and the address field of the instruction gives a displacement relative to this base address.
- The base register addressing mode is used in computers to facilitate the relocation of programs in memory.

3.5 Data Transfer and Manipulation

Computers provide an extensive set of instructions to give the user the flexibility to carry out various computational tasks.

The instruction set of different computers differ from each other mostly in the way the operands are determined from the address and mode fields.

Most computer instructions can be classified into three categories:

- Data Transfer Instructions
- Data Manipulation Instructions
- Program Control Instructions

Data Transfer Instructions

Data Transfer Instructions move data from one place in the computer to another withoutchanging the data content.

The most common transfers are between memory and processor registers, between processor registers and input or output, and between the processor register's themselves.

Table 3.1 gives a list of eight data transfer instructions used in many computers.

Table 3.1 Typical Data Transfer Instructions

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

The **Load** instruction has been used mostly to designate a transfer from memory to a processor register, usually an accumulator.

The **Store** instruction designates a transfer from a processor register into memory.

The **Move** instruction has been used in computers with multiple CPU registers to designate a transfer from one register to another. It has also been used for data transfers between CPU registers and memory or between two memory words.

The **Exchange** instruction swaps information between two registers or a register and a memory word.

The **Input** and **Output** instructions transfer data among processor registers and input or output terminals.

The **Push** and **Pop** instructions transfer data between processor registers and a memory stack.

Data Manipulation Instructions

Data Manipulation Instructions perform operations on data and provide the computational capabilities for the computer.
The data manipulation instructions in a typical computer are usually divided into three basic types:

1. Arithmetic Instructions
2. Logical and Bit Manipulation Instructions
3. Shift Instructions

1. Arithmetic Instructions

Arithmetic Instructions are the instructions which perform basic arithmetic operations such as addition, subtraction, multiplication and division.

Most computers provide instructions for all four operations.

Some small computers have only addition and possibly subtraction instructions. The multiplication and division must then be generated by means of software subroutines.

A list of typical arithmetic instructions is given in Table 3.2.

Table 3.2 Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

- The **Increment** instruction adds 1 to the value stored in a register or memory word.
- The **Decrement** instruction subtracts 1 from a value stored in a register or memory word.

- The **Add, Subtract, Multiply**, and **Divide** instructions may be available for different types of data.
- An arithmetic instruction may specify fixed-point or floating-point data, binary or decimal data, single-precision or double-precision data.

2. Logical and Bit Manipulation Instructions

- **Logical Instructions perform binary operations on strings of bits stored in registers**.
- They are useful for manipulating individual bits or a group of bits that represent binary-coded information.
- The Logical Instructions consider each bit of the operand separately and treat it as a Boolean variable.
- Some typical logical and bit manipulation instructions are listed in Table 3.3.

Table 3.3 Logical and Bit Manipulation Instructions

Name	Mnemonic
Clear	**CLR**
Complement	**COM**
AND	**AND**
OR	**OR**
Exclusive-OR	**XOR**
Clear Carry	**CLRC**
Set Carry	**SETC**
Complement Carry	**COMC**
Enable Interrupt	**EI**
Disable Interrupt	**DI**

- The **Clear** instruction causes the specified operand to be replaced by D's.
- The **Complement** instruction produces the 1's complement by inverting all the bits of the operand.
- The **AND, OR**, and **XOR** instructions produce the corresponding logical operations on individual bits of the operands.

3. Shift Instructions

- **Shift Instructions are used to shift the content of an operand.**
- Shifts are operations in which the bits of a word are moved to the left or right.
- The bit shifted in at the end of the word determines the type of shift used.

- Shift Instructions may specify either ogical Shifts, Arithmetic Shifts, or Rotate-type operations.
- In either case the shift may be to the right or to the left.
- Table 3.4 lists four types of shift instructions.

Table 3.4 Typical Shift Instructions

Name	Mnemonic
Logical Shift Right	**SHR**
Logical Shift Left	**SHL**
Arithmetic Shift Right	**SHRA**
Arithmetic Shift Left	**SHLA**
Rotate Right	**ROR**
Rotate Left	**ROL**
Rotate Right Through Carry	**RORC**
Rotate Left Through Carry	**ROLC**

The logical shift inserts 0 to the end bit position. The end position is the leftmost bit for shift right and the rightmost bit position for the shift left.

The arithmetic shift-right instruction must preserve the sign bit in the leftmost position.

The arithmetic shift-left instruction inserts 0 to the end position and is identical to the logical shift-left instruction.

The rotate instructions produce a circular shift.

The rotate through carry instruction treats a carry bit as an extension of the register whose word is being rotated.

Thus a rotate-left through carry instruction transfers the carry bit into the rightmost bit position of the register, transfers the leftmost bit position into the carry, and at the same time, shifts the entire register to the left.

3.6 Program Control

- **Program Control Instructions change or modify the flow of a program.**
- These instructions specify conditions for altering the content of the program counter.
- Some typical program control instructions are listed in Table 3.5**.**

Table 3.5 Typical Program Control Instructions

Name	Mnemonic

Branch	**BR**
Jump	**JMP**
Skip	**SKP**
Call	**CALL**
Return	**RET**
Compare (by subtraction)	**CMP**
Test (by ANDing)	**TST**

- **Branch** and **Jump** instructions may be conditional or unconditional. An unconditional branch instruction causes a branch to the specified address without any conditions. The conditional branch instruction specifies a condition such as branch if positive or branch if zero.
- The **Skip** instruction does not need an address field and is therefore a zero-address instruction. A conditional skip instruction will skip the next instruction if the condition is met. If the condition is not met, control proceeds with the next instruction in sequence where the programmer inserts an unconditional branch instruction.
- The **Call** and **Return** instructions are used in conjunction with subroutines.
- The **Compare** and **Test** instructions do not change the program sequence directly. The compare instruction performs a subtraction between two operands, but the result of the operation is not retained. However, certain status bit conditions are set as a result of the operation. Similarly, the test instruction performs the logical AND of two operands and updates certain status bits without retaining the result or changing the operands. The status bits of interest are the carry bit, the sign bit, a zero indication, and an overflow condition.

Status Bit Conditions

- **Status Bits are also called condition code bits or flag bits.**
- The Four status bits are symbolized by C, S, Z, and V.
- The bits are set or cleared as a result of an operation performed in the ALU.

1. **Bit C (Carry):**
    a. Set to 1, if carry is 1
    b. Set to 0, if carry is 0

2. **Bit S (Sign):**
    a. Set to 1, if the highest order bit is 1.
    b. Set to 0, if the bit is 0.

3. **Bit Z (Zero):**
   a. **Z=1,** if the output is 0
   b. **Z=0,** if the output is not zero.
4. **Bit V (Overflow):**
   a. **V=1** , if the output is greater than +127 or less then -128
   b. V=0

Conditional Branch Instructions

Table 3.6 gives a list of the most common branch instructions.

Each mnemonic is constructed with the letter B (for branch) and an abbreviation of the condition name. When the opposite condition state is used, the letter N (for no) is inserted to define the 0 state. Thus BC is Branch on Carry, and BNC is Branch on No Carry.

Table 3.6 Conditional Branch Instructions

Mnemonic	Branch Condition	Test Condition
BZ	Branch if zero	Z = 1
BNZ	Branch if not zero	Z = 0
BC	Branch if carry	C = 1
BNC	Branch if no carry	C = 0
BP	Branch if plus	S = 0
BM	Branch if minus	S = 1
BV	Branch if overflow	V = 1
BNV	Branch if no overflow	V = 0
Unsigned Compare Conditions (A – B)		
BHI	Branch if higher	A > B
BHE	Branch if higher or equal	A >= B
BLO	Branch if lower	A < B
BLOE	Branch if lower or equal	A <= B
BE	Branch if equal	A = B
BNE	Branch if not equal	A # B
Signed Compare Conditions (A –B)		
BGT	Branch if greater than	A > B
BGE	Branch if greater than	A >= B
BLT	Branch if less than	A < B
BLE	Branch if less or equal	A <= B
BE	Branch if equal	A = B

B		A # B
N		
E		

Subroutine Call and Return

- **A Subr** **uctions that perf**
- Durin nction man main program.
- Ea ed, a branch is executed to utine t instructions.
- s been executed, a branch is made back
- transfers program control to a subroutine is k nes.
- on names used are call subroutine, jump to o ranch and save address.
- ine instruction consists of an operation code together w beginning of the subroutine.
- tion is executed by performing two operations: (1) the add available in the program counter (the return address) is stored so the subroutine knows where to return, and (2) control is tran g of the subroutine.
- st instruction of every subroutine, commonly called return from rs the return address from the temporary location into the program coun results in a transfer of program control to the instruction whose ad nally stored in the temporary location.

broutine call is implemented with the following microoperations:

SP – 1 **Decrement stack pointer**

SP]PC **Push content of PC onto the stack**

Effective Address Transfer control to the subroutine

instruction that returns from the last subroutine is implemented by the microope

M[SP] **Pop stack and transfer to PC**

SP + 1 **Increment stack pointer**

Program Interrupt

- **Program Interrupt refers to the transfer of program control from a currently running program to another service program as a result of an external or internal generated**

request.

- Control returns to the original program after the service program is executed.
- The interrupt procedure is quite similar to a subroutine call except for three variations:
  (1) The interrupt is usually initiated by an internal or external signal rather than from the execution of an instruction;
  (2) the address of the interrupt service program is determined by the hardware rather than from the address field of an instruction; and
  (3) an interrupt procedure usually stores all the information necessary to define the state of the CPU rather than storing only the program counter.
- The state of the CPU at the end of the execute cycle (when the interrupt is recognized) is determined from:
  1. The content of the program counter
  2. The content of all processor registers
  3. The content of certain status conditions
- The collection of all status bit conditions in the CPU is sometimes called a **program status word** or **PSW**.
- When the CPU is executing a program that is part of the operating system, it is said to supervisor mode be in the **supervisor or system mode**.

Types of Interrupts

- **An interrupt is a signal sent to the processor that interrupts the current process.** It may be generated by a hardware device or a software program.
- There are three major types of interrupts that cause a break in the normal execution of a program. They can be classified as:

1. **External Interrupts**
2. **Internal Interrupts**
3. **Software Interrupts**

External Interrupts

- Come from I/O device, from a timing device, from a circuit.
- Monitoring the power supply, or from any other external source.

Internal Interrupts or TRAP

- Interrupts caused by register overflow,
- Attempt to divide by zero,
- Invalid operation code,

- Stack overflow, and
- Protection violation

Software Interrupts

- Initiated by executing an instruction.
- Used by the programmer to initiate an interrupt procedure at any desired point in the program.

3.7 Reduced Instruction Set Computer (RISC)

- An important aspect of computer architecture is the design of the instruction set for the processor.
- The instruction set chosen for a particular computer determines the way that machine language programs are constructed.
- **A Computer with a large number of instructions is classified as a complex instruction set computer, abbreviated CISC**.
- **RISC stands for Reduced Instruction Set Computer. It is a type of Computer that has a limited number of instructions. They can execute their instructions very fast because instructions are very small and simple.**

CISC Characteristics

- The major characteristics of CISC architecture are:

1. A large number of instructions-typically from 100 to 250 instructions
2. Some instructions that perform specialized tasks and are used infrequently
3. A large variety of addressing modes-typically from 5 to 20 different modes
4. Variable-length instruction formats
5. Instructions that manipulate operands in memory

RISC Characteristics

- The major characteristics of a RISC processor are:

1. Relatively few instructions
2. Relatively few addressing modes
3. Memory access limited to load and store instructions
4. All operations done within the registers of the CPU
5. Fixed-length, easily decoded instruction format
6. Single-cycle instruction execution
7. Hardwired rather than microprogrammed control

- A characteristic of RISC processors is their ability to execute one instruction per dock cycle. This is done by overlapping the fetch, decode, and execute phases of two or three instructions by using a procedure referred to as pipe lining.
- A load or store instruction may require two clock cycles because access to memory takes more time than register operations.
- Other characteristics attributed to RISC architecture are:

1. A relatively large number of registers in the processor unit
2. Use of overlapped register windows to speed-up procedure call and return Efficient instruction pipeline
3. Compiler support for efficient translation of high-level language programs into machine language programs

Overlapped Register Windows

- A characteristic of some RISC processors is their use of overlapped register windows to provide the passing of parameters and avoid the need for saving and restoring register values.
- Each procedure call results in the allocation of a new window consisting of a set of registers from the register file for use by the new procedure.
- Each procedure call activates a new register window by incrementing a pointer, while the return statement decrements the pointer and causes the activation of the previous window.
- Windows for adjacent procedures have overlapping registers that are shared to provide the passing of parameters and results.

Differences between RISC and CISC

Table 3.7 shows the Differences between RISC and CISC

Table 3.7 Differences between RISC and CISC

S. No	RISC	CISC
1	Focus on software	Focus on hardware
2	Uses only hardwired control unit	Uses both hardwired and microprogrammed control unit.
3	Transistors are used for more registers	Transistors are used for storing complex instructions
4	Fixed sized instructions	Variable sized instructions
5	Can perform only register to egister Arithmetic operations	Can perform REG to REG or REG to MEM or MEM to MEM
6	Requires more number of registers	Requires less number of registers
7	Code size is large	Code size is small

8	An instruction execute in single clock cycle	An instruction take more than one clock cycle
9	An instruction fit in one word	An instruction are larger than size of one word.

EXERCISES:

1. Define: Control Word.
2. What is meant by Microoperation?
3. Explain about Stack Organization with an example.
4. Explain the various Instruction types.
5. What is a subroutine?
6. What is the purpose of CALL and RET instruction to implement subroutine?
7. Discuss about different types of addressing modes.
8. Discuss about Data Transfer Instructions.
9. Explain in detail about Data Manipulation Instructions.
10. Write about conditional branch instructions with example.
11. What is meant by subroutine call and return?
12. Which types of multiplexers are used to design seven CPU register organization? Explain indetail.
13. Give the difference between RISC and CISC.

****

****

Chapter 4 Pipeline and Vector Processing

****

In this Chapter, the concept of pipelining is explained and the way it can speed-up processing is illustrated with several examples. Both arithmetic and instruction pipeline is considered. It is shown how RISC processors can achieve single-cycle instruction execution by using an efficient instruction pipeline together with the delayed load and delayed branch techniques. Vector processing is introduced and examples are shown of floating-point operations using pipeline procedures.

4.1 Pipelining

Pipelining is a technique of decomposing a sequential process into sub operations, with each sub process being executed in a special dedicated segment that operates concurrently with all other segments.

Pipelining can be visualized as a collection of processing segments through which binary information flows.

Each segment performs partial processing dictated by the way the task is partitioned.

The result obtained from the computation in each segment is transferred to the next segment in the pipeline.

The final result is obtained after the data have passed through all segments.

The Pipeline organization will be demonstrated by means of a simple

example: $A_i * B_i + C_i$ for i = 1, 2, 3, …7

Each sub operation is to be implemented in a segment within a pipeline.

Each segment has one or two registers and a combinational circuit as shown in Figure 4.1.

R1 through R5 are registers that receive new data with every clock pulse.

The multiplier and adder are combination al circuits.

The sub operations pe3rformed in each segment of the pipeline are as follows:

R1 ← A_i,	**R2 ← B_i**	**Input A_i, and B_i**
R3 ← R1 * R2,	**R4 ← C_i**	**multiply and input C_i**
R5 ← R3 + R4		**Add C_i to product**

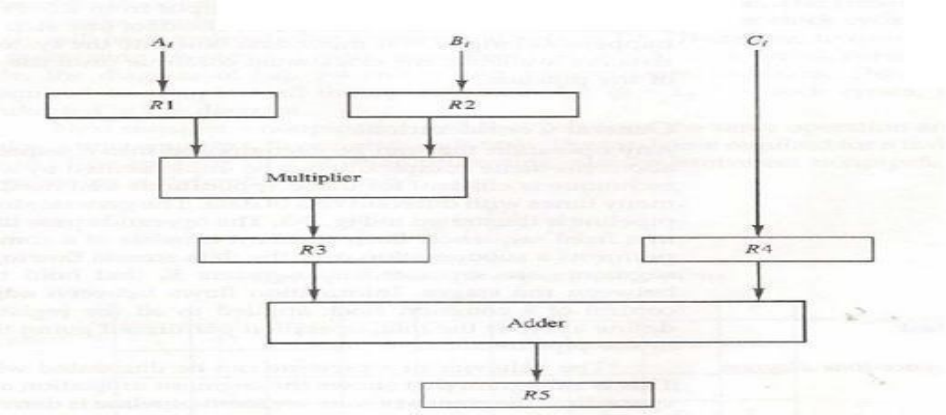


Figure 4.1 Pipeline Processing

The five registers are loaded with new data every clock pulse.

The effect of each clock is shown in Table 4.1.

Table 4.1 Effect of Registers with Clock

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A_1	B_1	—	—	—
2	A_2	B_2	$A_1 * B_1$	C_1	—
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
6	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
7	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
8	—	—	$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
9	—	—	—	—	$A_7 * B_7 + C_7$

The first clock pulse transfers A_1 and B_1 into R1 and R2.

The second clock pulse transfers the product of R1 and R2 into R3, C1 into R4.

The same clock pulse transfers A_2 and B_2 into R1 and R2.

The third clock pulse operates on all three segments simultaneously.

It places A_3 and B_3 into R1 and R2.

Transfers the product of R1 and R2 into R3.

Transfers C_2 into R4.

Place the sum of R3 and R4 into R5.

It takes three clock pulses to fill up the pipe and retrieve the first output from R5.

Each clock produces a new output and moves the data one step down the pipeline.

This happens as long as new input data flow into the system.

When no more input data are available, the clock must continue until the last output emerges one

of the pipelines.

4.2 Arithmetic Pipeline

Arithmetic Pipeline units are usually found in very high speed computers.

They are used to implement floating-point operations, multiplication of fixed-point numbers, and similar computations encountered in scientific problems.

A pipeline multiplier is essentially an array multiplier.

Special adders designed to minimize the carry propagation time through the partial products.

Floating-point operations are easily decomposed into sub operations.

Inputs to the floating-point adder pipeline are two normalized floating-point binary numbers**:** $\mathbf{X = A \times 2^a}$

$\mathbf{Y = B \times 2^b}$

A and B are two fractions representing mantissas (two Fractions) and a, b are the exponents.

The Floating-point addition and subtraction can be performed in four segments, as shown in Figure.4.2. The registers labeled R are placed between the segments to store intermediate results.

The sub operations that are performed in the four segments are:

1. **Compare the exponents**
2. **Align the mantissas**
3. **Add or subtract the mantissas**
4. **Normalize the result**

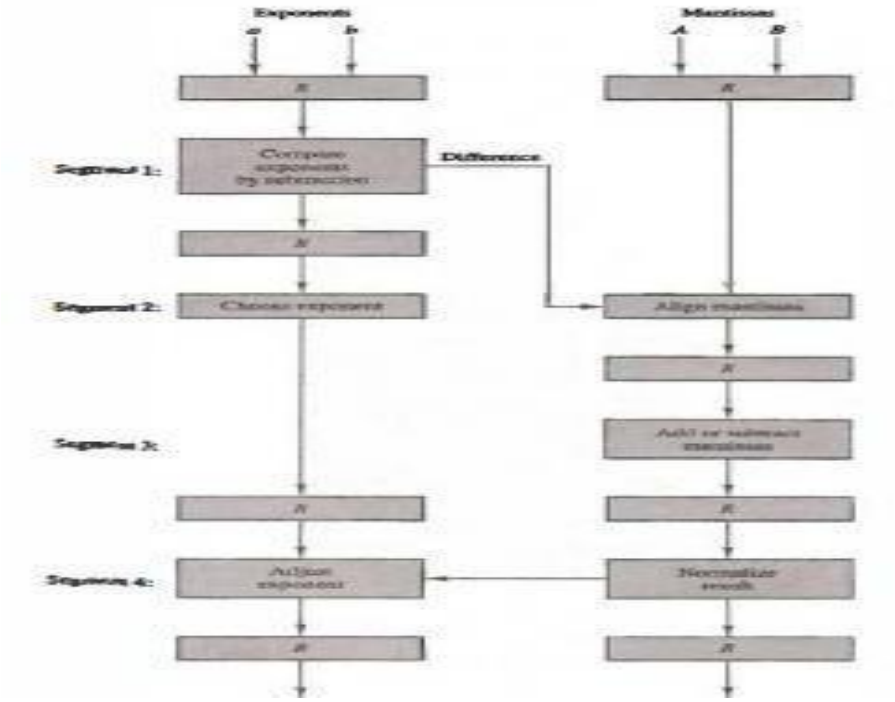


Figure 4.2 Pipeline for floating-point addition and subtraction

Consider the two normalized floating-point numbers:

X = 0.9504×10^3 **Y = 0.8200×10^2**

Segment 1: Two exponents are subtracted 3-2 = 1.

The larger exponent 3 is chosen as the exponent of the result.

Segment 2: Shift the mantissa of Y to the right to obtain

$\mathbf{X = 0.9504 \times 10^3\ Y = 0.0820 \times 10^3}$

This aligns the two mantissas under the same exponent.

Segment 3: Produces the sum

$\mathbf{X = 0.9504 \times 10^3}$

$\mathbf{Y = 0.0820 \times 10^3}$

$\mathbf{Z = 1.0324 \times 10^3}$

Segment 4: The sum is adjusted by normalized the result.

Therefore, it has fraction with a nonzero first digit.

This is done by shifting the mantissa once to the right and increment the exponent byone to obtain the normalized sum.

$\mathbf{Z = 0.10324 \times 10^4}$

4.3 Instruction Pipeline

An Instruction Pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments.

This causes the instruction fetch and execute to overlap and perform simultaneous operations. Computers with complex instructions require other phases in addition to the fetch and execute to process an instruction completely.
In the most general case, the computer needs to process each instruction with the following sequence of steps.

1. Fetch the Instruction from Memory.
2. Decode the Instruction.
3. Calculate the effective address.
4. Fetch the operands from memory.
5. Execute the instruction.
6. Store the result in the proper place.

1. **Fetch the Instruction from Memory**:
   - The Fetch Instruction is responsible for obtaining the requested instruction from memory.
   - The instruction and the program counter are stored in the register as temporary storage.

2. **Decode the Instruction** :

- The Decode Instruction stage is responsible for decoding the instruction
- Sending it into out the various control lines to the other parts of the processor.

3. **Calculate the effective address**:
   - The Calculate operands stage is where any calculations are performed.
   - The main component in this stage is the ALU.
   - The ALU is made up of arithmetic, logic and capabilities.

4 & 5 Fetch the operands from memory & Execute the instruction:

- The Fetch operand and Execute Instruction stages are responsible for storing and loading values to and from memory.
- They also responsible for input and output from the processor respectively.

6. Store the result in the proper place:

- This stage is responsible for store the result of a calculation.

Example: Four-Segment Instruction Pipeline

- Figure 4.3 shows how the instruction cycle in the CPU can be processed with a four-segment pipeline.
- **Segment 1:** The Instruction Fetch segment can be implemented using first in first out (FIFO) buffer.
- **Segment 2:** The instruction fetched from memory is decoded and effective address is calculated.
- **Segment 3:** An operand from memory is fetched in the third segment.
- **Segment 4:** The Instructions are finally executed in the last segment of the pipeline organization.

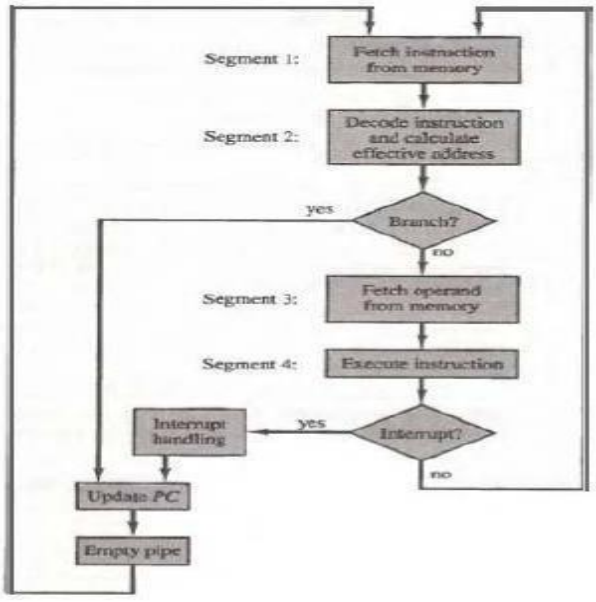


Figure 4.3 Instructio n Cycle in the CPU

- Figure 4.4 shows the operation of the instruction pipeline.
- The time in the horizontal axis is divided into steps of equal duration.
- The four segments are represented in the diagram with an abbreviated symbol.
  1. FI is the segment that fetches an instruction.
  2. DA is the segment that decodes the instruction and calculates the effective address.
  3. FO is the segment that fetches the operand.
  4. EX is the segment that executes the instruction.

Step:		1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction:	1	FI	DA	FO	EX									
	2		FI	DA	FO	EX								
(Branch)	3			FI	DA	FO	EX							
	4				FI	–	–	FI	DA	FO	EX			
	5					–	–	–	FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

Figure 4.4 Timing of Instruction pipeline

Pipelining Conflicts

- There are three major difficulties that cause the instruction pipeline to deviate from its normal operation.

1. **Resource Conflicts**
2. **Data Dependency**
3. **Branch Difficulties**

1. **Resource Conflicts**
   - Caused by access to memory by two segments at the same time.
   - Most of these conflicts can be resolved by using separate instruction and data memories.

2. **Data Dependency**
   - A Data Dependency occurs when an instruction needs data that are not yet available.
   - The data dependency conflict can be solved by using the following methods:
     - **Hardware Interlocks:** The most straight forward method is to insert hardware interlocks.
     - **An interlock is a circuit that detects instructions whose source operands are**

destination of instructions farther up in the pipeline.

- Detection of this situation causes the instruction whose source is not available to be delayed by enough clock cycles to resolve the conflict.
- This approach maintains the program sequence by using hardware to insert the required delay.
- **Operand Forwarding:** The operand forwarding technique uses special hardware to detect a conflict and avoid the conflict path by using a special path to forward the values between the pipeline segments.
- **Delayed Load:** The delayed load operation is nothing but when executing an instruction in the pipeline.
- Simply delay the execution starting the instruction such that all the data that is needed for the instruction can be successfully updated before execution.

3. Branch Difficulties

- Arise from branch and other instructions that change the value of PC.
- The following are the solutions for solving the branch conflicts that obtained in the pipelining concept.
  - **Pre-fetch Target Instruction:** In this branch instructions which are to be executed are pre-fetch detect if any errors are present in the branch before execution.
  - **Branch Target Buffer: BTB** is the Associative memory implementation of the branch conditions.
  - **Loop Buffer:** The loop buffer is a very high speed memory device.
  - **Branch Prediction:** Before a branch is to be executed, the instructions along with the error checking conditions are checked.
  - **Delayed Branch:** This concept is same as the delayed load process in which we are delaying the execution of a branch process, before all the data is fetched by the system for beginning the CPU.

4.4 RISC Pipeline

The ability to use the instruction pipelining concept in the RISC architecture is very efficient.

The simplicity of the instruction set can be utilized to implement an instruction pipeline using a small number of sub operations, with each being executed in one clock cycle.

Due to fixed length instruction format, the decoding of the operation can occur at thesame time as the register selection.

Since the arithmetic, logic and shift operations are done on register basis, there is no need for extra fetching or effective address decoding steps to perform the operation.

So pipelining concept can be effectively used in this scenario.

Therefore the total operations can be categorized as one segment will be fetching the instruction from program memory, the other segment executes the instruction in the ALU and the third segment may be used to store the result of the ALU operation in a destination register.

The data transfer instructions in RISC are limited to only Load and Store instructions.

To prevent conflicts in data transfer, we will be using two separate buses one for storing the instructions and other for storing the data.

Example: Three Segment Instruction Pipeline

There are three types of instructions for a RISC processor.

The data manipulation instructions operate on data in processor registers.

The data transfer instructions are load and store instructions that use an effective address obtained from the addition of the contents of two registers or a register and a displacement constant provided in the instruction.

The program control instructions use register values and a constant to evaluate the branch address, which is transferred to a register or the program counter PC.

The instruction cycle can be divided into three suboperations and implemented in three segments:.

I: Instruction Fetch

A: ALU Operation

E: Execute Instruction

The **I** segment fetches the instruction from program memory.

The instruction is decoded and an ALU operation is performed in the **A** segment. The ALU is used for three different functions, depending on the decoded instruction. It performs an operation for a data manipulation instruction, it evaluates the effective address for a load or store instruction, or it calculates the branch address for a program control instruction.

The E segment directs the output of the ALU to one of three destinations, depending on the decoded instruction.

Delayed Load

Consider the following instructions:

1. LOAD: R1 ← M[address 1]
2. LOAD: R2 ← M[address 2]
3. ADD: R3 ← R1 + R2

4. STORE: M[address 3] ← R3

- If the three-segment pipeline proceeds without interruptions, there will be a data conflict in instruction 3 because the operand in R2 is not yet available in the A segment.
- This can be seen from the timing of the pipeline shown in Figure 4.6(a).
- Figure 4.6(b) shows the same program with a no-op instruction inserted after the load to R2 instruction.

Clock cycles:	1	2	3	4	5	6
1. Load *R1*	I	A	E			
2. Load *R2*		I	A	E		
3. Add *R1* + *R2*			I	A	E	
4. Store *R3*				I	A	E

(a) Pipeline timing with data conflict

Clock cycle:	1	2	3	4	5	6	7
1. Load *R1*	I	A	E				
2. Load *R2*		I	A	E			
3. No-operation			I	A	E		
4. Add *R1* + *R2*				I	A	E	
5. Store *R3*					I	A	E

(b) Pipeline timing with delayed load

Figure 4.6 Three-segment Pipeline Timing.

4.5 Vector Processing

The vectors are considered as the large one dimensional array of data.

The term Vector Processing involves the data processing on the vectors of such large data. In many science and engineering applications, the problems can be formulated in terms of Vectors and Matrices that lend themselves to Vector Processing.

Computers with Vector Processing capabilities are in demand in specialized applications.

The following are representative application areas where Vector Processing is of the utmost importance.

- Long-range Weather Forecasting
- Petroleum Explorations
- Seismic Data Analysis
- Medical Diagnosis
- Aerodynamics and space flight simulations
- Artificial Intelligence and Expert Systems
- Mapping the human genome

- Image Processing

Vector Operations

- A Vector is an ordered set of a one-dimensional array of data items.
- A Vector V of length n is represented as a row vector by $V = [V_1, V_2, V_3, \ldots, V_n]$.
- It may be represented as a column vector if the data items are listed in a column.
- A conventional sequential computer is capable of processing operands one at a time.
- Consequently, operations on Vectors must be broken down into single computations with subscripted variables.
- The element V_i of Vector V is written as V(I) and the index I refers to a memory address or register where the number is stored.
- To examine the difference between a conventional Scalar Processor and a Vector Processor, consider the following Fortran DO loop:

```
      DO 20 I = 1, 100
20 C (I) = B (I) + A (I)
```

- This is a program for adding two vectors A and B of length 100 to produce a vector C. This is implemented in machine language by the following sequence of operations.

```
     Initialize I = 0
20   Read A (I)
     Read B (I)
     Store C (I) = A (I) + B (I)
     Increment I = I + 1
     If I<=100 go to 20
     Continue
```

- A computer capable of vector processing eliminates the overhead associated with the time it takes to fetch and execute the instructions in the program loop.
- It allows operations to be specified with a single vector instruction of the form

C(1 : 100) = A(1 : 100) + B(1 : 100)

- The vector instruction includes the initial address of the operands, the length of the vectors, and the operation to be performed, all in one composite instruction.
- A possible instruction format for a vector instruction is shown in Figure 4.7.
- This is essentially a three-address instruction with three fields specifying the base address of the operands and an additional field that gives the length of the data items in the vectors.
- This assumes that the vector operands reside in memory.
- It is also possible to design the processor with a large number of registers and store all operands

in registers prior to the addition operation.

- In that case the base address and length in the vector instruction specify a group of CPU registers.

Figure 4.7 Instruction Format for Vector Processor

Operation code	Base address source 1	Base address source 2	Base address destination	Vector length

Matrix Multiplication

- Matrix multiplication is one of the most computational intensive operations performed in computers with vector processors.
- The multiplication of two n x n matrices consists of n' inner products or n' multiply-add operations.
- An n x m matrix of numbers has n rows and m columns and may be considered as constituting a set of n row vectors or a set of m column vectors.
- Consider, for example, the multiplication of two 3 x 3 matrices A and B.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

- The product matrix C is a 3 x 3 matrix whose elements are related to the elements of A and B by the inner product:

$$c_{ij} = \sum_{k=1}^{3} a_{ik} \times b_{kj}$$

- For example, the number in the first row and first column of matrix C is calculated by letting i = 1, j = 1, to obtain

$$c_{11} = a_{11} b_{11} + a_{12} b_{21} + a_{13} b_{31}$$

- In general, the inner product consists of the sum of k product terms of the form

$$C = A_1 B_1 + A_2 B_2 + A_3 B_3 + A_4 B_4 + \cdots + A_k B_k$$

- The inner product calculation on a pipeline vector processor is shown in Figure 4.8.

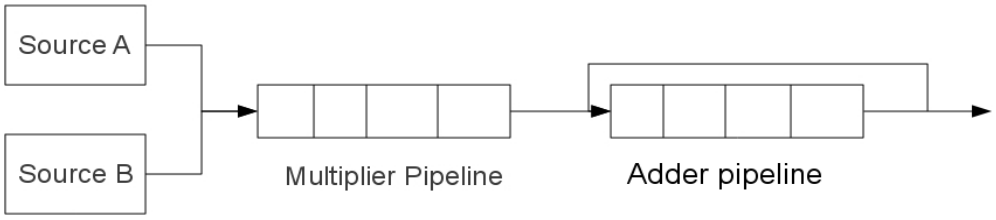


Figure 4.8 Pipeline for calculating an inner product.

- The values of A and B are either in memory or in processor registers.

- The floating-point multiplier pipeline and the floating-point adder pipeline are assumed to have four segments each.
- All segment registers in the multiplier and adder are initialized to 0.
- When addition operation is taking place in the adder pipeline the next set of values will be brought into the multiplier pipeline.
- All the operations can be performed simultaneously using the parallel processing concepts by the implementation of pipeline.

Memory Interleaving

- **Memory Interleaving is a technique which divides the memory into a number of modules. In an interleaved memory, different sets of addresses are assigned to different memory modules.**
- A memory module is a memory array together with its own address and data registers.
- The modular system permits one module to initiate a memory access while other modules are in the process of reading or writing a word and each module can honor a memory request independent of the state of the other modules.
- A modular memory is useful in systems with pipeline and vector processing.
- A vector processor that uses an n-way interleaved memory can fetch n operands from n different modules.

Supercomputers

- A commercial computer with vector instructions and pipelined floating-point arithmetic operations is referred to as a supercomputer.
- Supercomputers are very powerful, high-performance machines used mostly for scientific computations.
- Supercomputers also use special techniques for removing the heat from circuits to prevent them from burning up because of their close proximity.
- The instruction set of supercomputers contains the standard data transfer, data manipulation, and program control instructions of conventional computers.
- A supercomputer is a computer system best known for its high computational speed, fast and large memory systems, and the extensive use of parallel processing.
- It is equipped with multiple functional units and each unit has its own pipeline configuration.
- Supercomputers are not suitable for normal everyday processing of a typical computer installation. They are limited in their use to a number of scientific applications, such as numerical weather forecasting, seismic wave analysis, and space research.
- They have limited use and limited market because of their high price.

- A measure used to evaluate computers in their ability to perform a given number of floating-point operations per second is referred to as **flop**s.
- The term **megaflops** is used to denote million flops and **gigaflops** to denote billion flops.
- **The first supercomputer developed in 1976 is the Cray-1 supercomputer.**

4.6 Array Processors

- **An Array Processor is a processor that performs computations on large arrays of data. The term is used to refer to two different types of processors**.
- An attached array processor is an auxiliary processor attached to a general-purpose computer.
- It is intended to improve the performance of the host computer in specific numerical computation tasks.
- An SIMD array processor is a processor that has a single-instruction multiple-data organization.
- It manipulates vector instructions by means of multiple functional units responding to a common instruction. Although both types of array processors manipulate vectors, their internal organization is different.

Attached Array Processor

- An attached array processor is designed as a peripheral for a conventional host computer, and its purpose is to enhance the performance of the computer by providing vector processing for complex scientific applications.
- It achieves high performance by means of parallel processing with multiple functional units.
- It includes an arithmetic unit containing one or more pipelined floating point adders and multipliers.
- The array processor can be programmed by the user to accommodate a variety of complex arithmetic problems.
- Figure 4.9 shows the interconnection of an attached array processor to a host computer.
- The host computer is a general-purpose commercial computer and the attached processor is a back-end machine driven by the host computer.
- The array processor is connected through an input-output controller to the computer and the computer treats it like an external interface.
- The data for the attached processor are transferred from main memory to a local memory through a high-speed bus.
- The general-purpose computer without the attached processor serves the users that need conventional data processing.
- The system with the attached processor satisfies the needs for complex arithmetic applications.

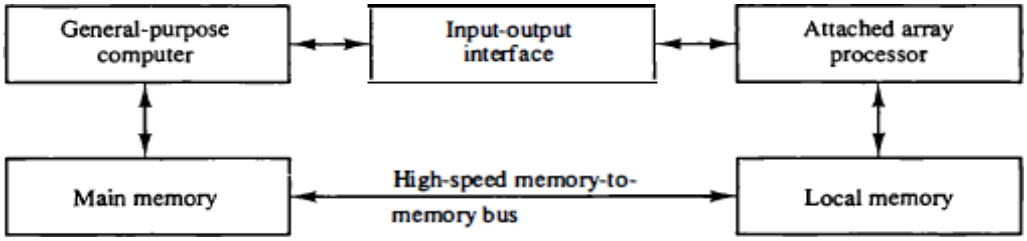


Figure 4.9 Attached Array Processor with host computer

SIMD Array Processor

- **An SIMD array processor is a computer with multiple processing units operating in parallel.**
- The processing units are synchronized to perform the same operation under the control of a common control unit, thus providing a single instruction stream, multiple data stream (SIMD) organization.
- A general block diagram of an array processor is shown in Figure 4.10.

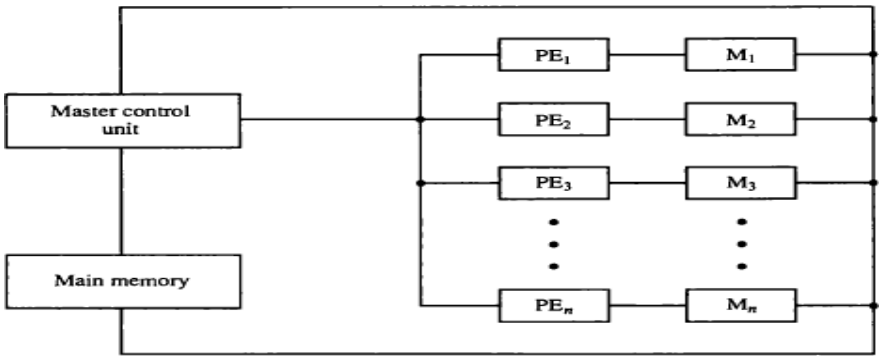


Figure 4.10 SIMD Array Processor Organization

- It contains a set of identical processing elements (PEs), each having a local memory M.
- Each processor element includes an ALU, a floating-point arithmetic unit, and working registers.
- The master control unit controls the operations in the processor elements.
- The main memory is used for storage of the program.
- The function of the master control unit is to decode the instructions and determine how the instruction is to be executed.
- Scalar and program control instructions are 15. It contains a set of identical processing elements (PEs), each having a local memory M.
- Each processor element includes an ALU, a floating-point arithmetic unit, and working registers.
- The master control unit controls the operations in the processor elements.
- The main memory is used for storage of the program.
- The function of the master control unit is to decode the instructions and determine how the instruction is to be executed. Scalar and program control instructions are

EXERCISES:

1. Explain about pipelining with an example.
2. Discuss about Arithmetic pipeline with an example.
3. What is the six and four segment instruction pipeline in computer organization?
4. Explain the six stages of instruction pipelining.
5. How many segments are there in the pipeline?
6. Write in detail about RISC pipeline concept.
7. Explain about Vector Processing with an example.
8. Discus about Array Processors.

****

UNIT – III

****

Chapter 5 Microprocessor Architecture and Microcomputer Systems

****

* This chapter describes about the Microcomputer Systems that consist of main components such as Microprocessor, Memory, and I/O (Input / Output). The internal logic design of the microprocessor, called its architecture. Also, the internal architecture of the 8085 Microprocesssor is discussed.

5.1 Microprocessor Architecture and its Operations

Microprocessor is a programmable digital device, designed with registers, flip-flop and timing elements.

The Microprocessor has a set of instructions, designed internally, to manipulate data and communicate with peripherals.

Data manipulation and communication is determined by the logic design of the microprocessor, called the architecture.

The microprocessor can be programmed to perform functions on given data by selecting necessary instructions from its set.

These instructions are given to the microprocessor by writing them into its memory.

Writing (Or entering) instruction and data is done through an input device such as a keyboard.

The microprocessor reads or transfers each instruction one at a time, matches it with its instruction set, and performs the data manipulation indicated by the instruction.

The result can be stored in memory or sent to such output devices as LEDs or a CRT terminal.

The various functions performed by the microprocessor can be classified in three general categories :

- Microprocessor-initiated operations
  - Internal data operations
  - Peripheral (or externally) initiated operations.

To perform these functions, the microprocessor requires a group of logic circuits and a set of signals called control signals.

The first microprocessor was the **Intel 4004**, introduced in 1971. The 4004 was not very powerful.

[Computer's Central Processing Unit (CPU) built on a **single Integrated Circuit (IC)** is called a **microprocessor**.

A digital computer with one microprocessor which acts as a CPU is called microcomputer.]

Applications of Microprocessor

Microprocessors are mainly used in devices like:

- Calculators
- Accounting System
- Games Machine
- Complex Industrial Controllers
- Traffic Light
- Control Data
- Military Applications
- Defense Systems
- Computation Systems

5.1.1 Microprocessor-Initiated Operations and 8085/8080A Bus Organization The MPU performs primarily four operations.

1. **Memory Read:** Reads data from memory. **2.Memory Write:** Writes data into memory. **3.I/O read:** Accepts data from input devices. **4.I/O Write:** Sends data to output devices.

- All these operations are part of the communication process between the MPU and peripheral devices (including memory).
- To communicate with a peripheral (or a memory location), the MPU needs to perform the following steps:

Step 1: Identify the peripheral or the memory location (with its address).

Step 2: Transfer data

Step 3: Provide timing or synchronization signals.

- The 8085/8080A MPU performs these functions using three sets of communication lines called buses :
  - The address bus,
  - The data bus,
  - The control bus.
- Figure 5.1 shows the 8085 Bus Structure as one group, called the system bus.

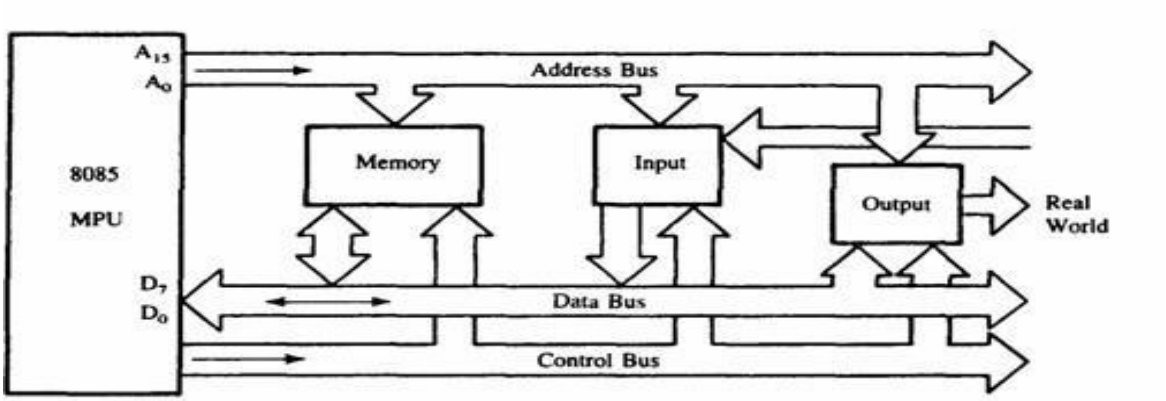


Figure 5.1The 8085 Bus Structure

- **Address Bus:**
  - The address bus is a group of 16 lines generally identified as A0 to A15.
  - The address bus is unidirectional: Bits flow in one direction – from theMPU to peripheral devices.
  - The MPU uses the address bus to perform the first function: identifying a peripheral or a memory location.
- **Data Bus:**
  - This is a bi-directional bus, because data can flow to or from the CPU.
  - T he CPU's eight data pins, D_0 through D_7, can be either inputs or outputs, depending on whether the CPU is performing a read or a write operation.
  - During data bus by the memory or I/O element. During a write operation the CPU's data pins act as outputs and place data on the data bus, which are then sent to the selected memory or I/O element.

- **Control Bus:**

- This is the set of signals that is used to synchronize the activities of the separate μC elements.
- Some of these control signals, such as RD and WR are sent by the CPU tothe other elements to tell them what type of operation is currently in progress.
- The I/O elements can send control signals to the CPU.

- **Memory Unit:**
  - Operation of the memory is controlled by the control unit which signals for either a read or a write operation.
  - A given location in memory is accessed by the control unit that provides the appropriate address code.
  - Information can be written into the memory from the ALU or the input unit, again under control of the control unit.
  - Information can be read from memory into the ALU or into the output unit.
- **Input Unit:**
  - The input unit consists of all of the devices used to take information and data that are external to the computer and put them into the memory unit or the ALU.
  - The control unit determines where the input information is sent.
  - The input unit is used to enter the program and data into the memory unit or in to the ALU from an external device during the execution of a program.
  - Some of the common input devices are keyboards, Joysticks, mouse, OCR, OMR etc.
- **Output Unit:**
  - The output unit consists of the devices used to transfer data and information from the computer to the "outside world."
  - The output devices are directed by the control unit and can receive data from memory or the ALU.
  - Examples of common output devices are printers, disk or tape units, video monitors, and digital-to-analog converters (DACs).
- **Central Processing Unit (CPU):**
  - The ALU and Control Units are combined into one unit called the Central Processing

Unit (CPU).

- This is commonly done to separate the actual “brains” of the computer from the other units.
- In a microcomputer the CPU is usually implemented on a single chipsilicon wafer.

5.1.2 Internal Data Operations and the 8085 Registers

- The internal architecture of the 8085/8080A microprocessor determines how and what operations can be performed with the data.
- These operations are
  1. Store 8-bits data.
  2. Perform arithmetic and logical operations.
  3. Test for conditions.
  4. Sequence the execution of instructions.
  5. Store data temporarily during execution in the defined R/W memory locations called the stack.
- To perform these operations, the microprocessor requires registers, arithmetic logic unit (ALU) and control logic, and internal buses (path for information flow).

Registers

- The 8085/8080A has six general - purpose registers to perform the first operation listed above, that is, to store 8-bit data during a program execution.
- These registers are identified as B, C, E, H, and L.
- They can be combined as register pairs - BC, DE, and HL - to performsome 16-bit operation.
- These registers are programmable, meaning that a programmer can use them to load or transfer data from the registers by using instructions.

Accumulator

- The accumulator is an 8-bit register that is part of the arithmetic logic unit (ALU).
- This register is used to store 8-bit data and to perform arithmetic and logical operations.
- The result of an operation is stored in the accumulator.
- The accumulator is also identified as register A.

Flags

- The ALU includes five flip-flops that are set or reset according to data conditions in the accumulator and other registers.

- The microprocessor uses them to perform the third operation; namely testing for data conditions.
- The 8085/8080A has five flags to indicate five different types of dataconditions. They are called Zero (Z), Carry (CY), Sign (S), Parity (P), and Auxiliary Carry (AC) flags.
- The most commonly used flags are Zero and Carry.

Program Counter (PC)

- This 16 bit register deals with the fourth operation, sequencing the execution of instructions.
- This register is a memory pointer.
- Memory locations have 16-bit address, and 16-bit register.
- The microprocessor uses this register to sequence the execution of instructions.
- The function of the program counter is to point to the memory address from which the next byte is to be fetched.
- When a byte (machine code) is being fetched the program counter is incremented by one to point to the next memory location.

Stack Pointer (SP)

- The stack pointer is also a 16-bit register used as a memory pointer; initially, it is called the stack pointer register to emphasize that it is a register.
- It pointers of a memory location in R/W memory, called the stack.
- The beginning of the stack is defined by loading a 16-bit address in the stack pointer (register).

EXERCISES:

1. What is CPU?
2. What basic operations occur in computer?
3. Name the five basic units of a computer and describe the major functions of each.
4. What is the function of central processing unit?
5. Draw the typical structure of a micro computer and show its bus organization.

****

****

Chapter 6 Microprocessor Architecture and Memory Interfacing

****

This chapter deals with the Architecture of 8085 Microprocessor and its Memory Interfacing concepts. The term microprocessor unit (MPU) is similar to the term central processing unit (CPU) used in computers.

6.1 The 8085 MPU

- The 8085 is an 8-bit general purpose microprocessor capable of addressing 64K of memory.
- The device has 40 pins.
- It requires a +5 V single power supply, and can operate with a 3-MHz single-phase clock.
- Figure 6.1 shows the logic pin out of the 8085 microprocessor.
- All the signal can be classified into six groups
  - **Address bus**
  - **Data bus**
  - **Control and status signals,**
  - **Power supply and frequency signals,**
  - **Externally initiated signals**
  - **Serial I/O ports**
  - **Address Bus:**
    - The 8085 has 8 signal lines,
    - A15-A8, which are unidirectional and used as the high order address bus.
  - **Multiplexed Address/ Data Bus:**
    - The signal lines AD7-AD0 are bidirectional; they serve a dual purpose.
    - They are used as the low-order address bus as well as the data bus.
  - **Control and Status Signals:**
    - This group of signals includes two control signals ($\overline{RD}$ and $\overline{WR}$),
    - These status signals (IO/M, s1, and s0) to identify the nature of the operation.
    - One special signal (ALE) to indicate the beginning of the operation.
    - **$\overline{ALE}$ (Address Latch Enable)**: This is a positive going pulse generated every time the 8085 begins an operation; it indicates that the bits on AD7-AD0 are address bus.

- **$\overline{\text{RD}}$ (Read)**: This is a Read control signal (active low). This signal indicates that the selected I/O or memory device is to be read and data are available on the data bus.
- **WR (Write):** This is a write control signal (active low). This signal indicates that the data on the data bus are to be written into a selected memory or I/O location.
- **IO/$\overline{\text{M}}$:** This is a status signal used to differentiate between I/O and memory operations. When it is high, it indicates an I/O operation; when it is low, it indicates a memory operation.
- **S1 and S0:** These status signals can identify various operations, nut they are rarely used in small systems.

- **Power Supply and Clock Frequency:**

  The power supply and frequency signals are as follows:

  1. **Vcc** : +5 V Power supply.
  2. **Vss** : Ground Reference
  3. **X1, X2** : A crystal is connected at these two pins.
  4. **CLK (OUT) :** This signal can be used as the system clock forother devices.

- **READY (Input):**

  - If Ready is high during a read or writes cycle, it indicates that the memory or peripheral is ready to send or receive data.
  - If Ready is low, the CPU will wait for Ready to go high before completing the read or write cycle.

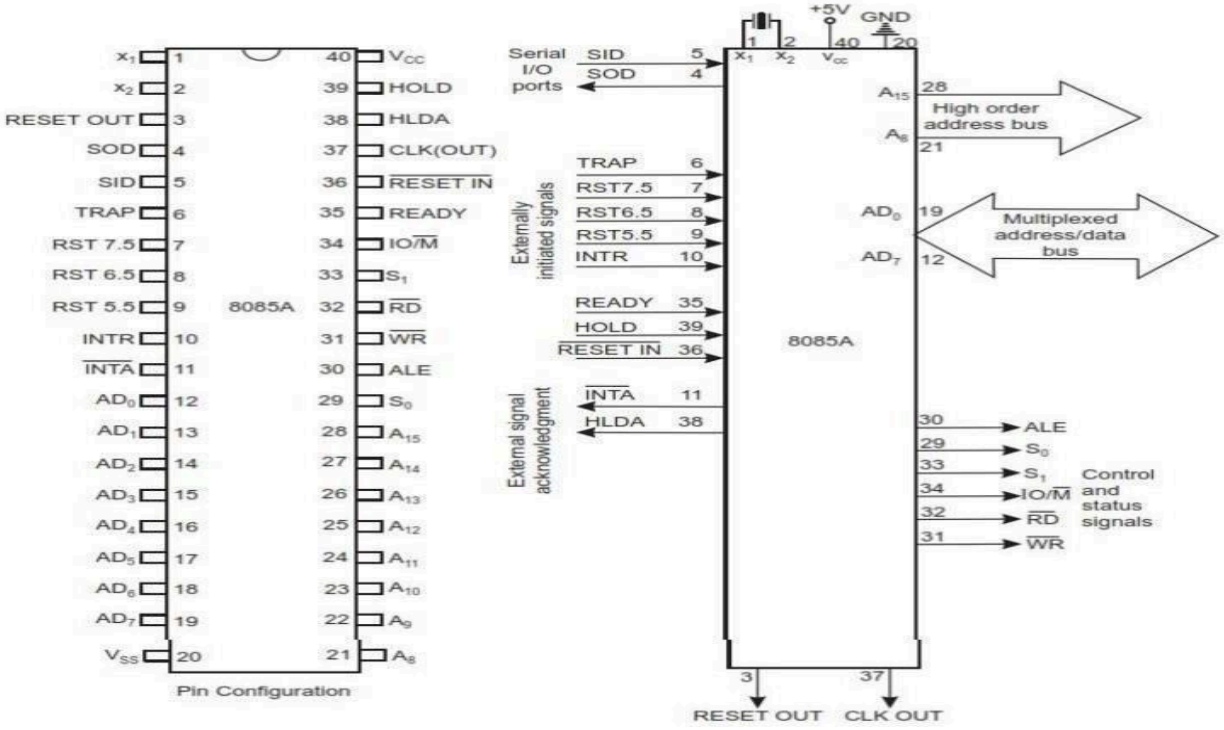
- **HOLD (Input):**
  - HOLD: indicates that another Master is requesting the use of the Address and Data Buses.
  - The CPU, upon receiving the Hold request will relinquish the use of buses assoon as the completion of the current machine cycle.
  - Internal processing can continue. The processor can regain the buses only afterthe Hold is removed.
  - When the Hold is acknowledged, the Address, Data, RD, WR, and IO/M lines

are 3stated.

- **HLDA (Output):**
  - HOLD ACKNOWLEDGE: indicates that the CPU has received the Hold request and that it will relinquish the buses in the next clock cycle. HLDA goes low after the Hold request is removed.
  - The CPU takes the buses one half clock cycle after HLDA goes low.
- **INTR (Input):**
  - INTERRUPT REQUEST is used as a general purpose interrupt.
  - It is sampled only during the next to the last clock cycle of the instruction.
  - If it is active, the Program Counter (PC) will be inhibited from incrementingand an INTA will be issued.
  - During this cycle a RESTART or CALL instruction can be inserted to jump to the interrupt service routine.
  - The INTR is enabled and disabled by software. It is disabled by Reset and immediately after an interrupt is accepted.
- **INTA (Output):**
  - INTERRUPT ACKNOWLEDGE: is used instead of (and has the same timing as) RD during the Instruction cycle after an INTR is accepted.
- **RESTART INTERRUPTS:**
  - These three inputs have the same timing as INTR except they cause an internal RESTART to be automatically inserted.
  - RST 7.5 Highest Priority
  - RST 6.5
  - RST 5.5 Lowest Priority
- **TRAP (Input):**
  - Trap interrupt is a non maskable restart interrupt.
  - It is recognized at the same time as INTR.
  - It is unaffected by any mask or Interrupt Enable.
  - It has the highest priority of any interrupt.
- **RESET IN (Input):**
  - Reset sets the Program Counter to zero and resets the Interrupt Enable and HLDA flip-flops.
  - None of the other flags or registers (except the instruction register) are affectedThe CPU is held in the reset condition as long as Reset is applied.

- **RESET OUT (Output):**
  - Indicates CPU is being reset. Can be used as a system RESET.
  - The signal is synchronized to the processor clock.

Figure 6.1 Pin Diagram of 8085



The 8085 MPU Architecture

- Figure 6.2 shows the internal architecture of the 8085.

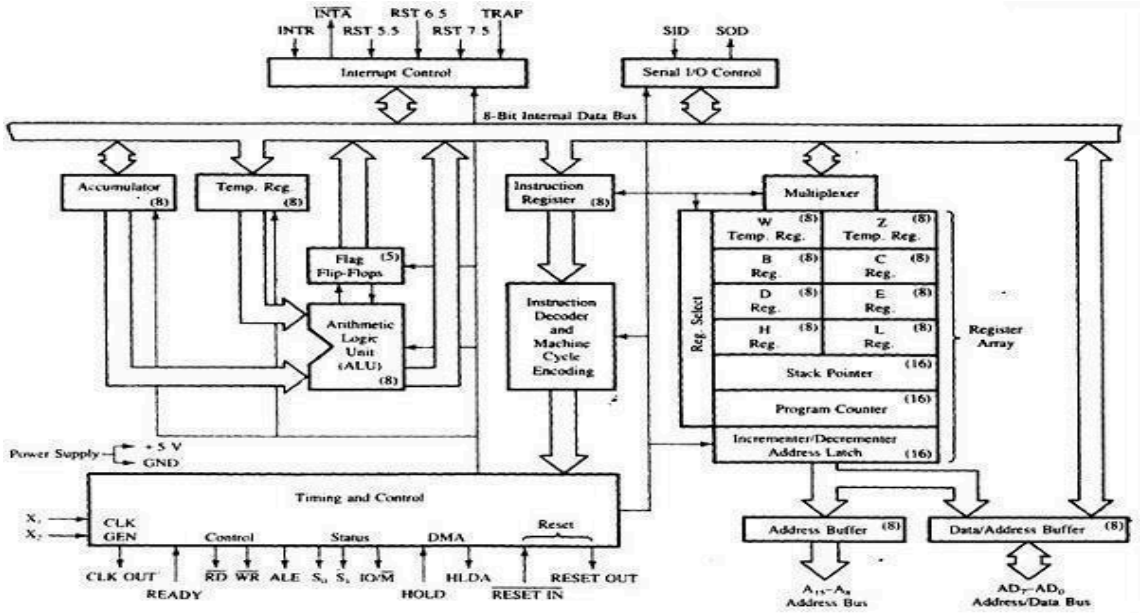


Figure 6.2 8085 Microprocessor Architecture

- **Accumulator:** It is an 8-bit register used to perform arithmetic, logical, I/O & LOAD/STORE operations. It is connected to internal data bus & ALU.
- **Arithmetic and logic unit:** It performs arithmetic and logical operations like Addition, Subtraction, AND, OR, etc. on 8-bit data.
- **General purpose register:** There are 6 general purpose registers in 8085 processor, i.e. B, C, D, E, H & L. Each register can hold 8-bit data. These registers can work in pair to hold 16-bit data and their pairing combination is like B-C, D-E & H-L.
- **Program Counter:** It is a 16-bit register used to store the memory address location of the next instruction to be executed. Microprocessor increments the program whenever an instruction is being executed, so that the program counter points to the memory address of the next instruction that is going to be executed.
- **Stack Pinter:** It is also a 16-bit register works like stack, which is always incremented / decremented by 2 during push & pop operations.
- **Temporary register:** It is an 8-bit register, which holds the temporary data of arithmetic and logical operations.
- **Flag register:** Table 6.1 shows the flag register. It is an 8-bit register having five 1-bit flip-flops, which holds either 0 or 1 depending upon the result stored in the accumulator.

  These are the set of 5 flip-flops −

  - **Sign Flag (S):** After the execution of an arithmetic or logic operation, if bitD7 of the result is 1, the sign flag is set.
  - **Zero Flag (Z):** The Zero flag is set if the ALU operation results in 0, and theflag is reset if the result is not 0.
  - **Auxiliary Carry Flag (AC):** In an arithmetic operation, when a carry is generated by digit D3 and passed on to digit D4, the AC flag is set.
  - **Parity Flag (P):** After an arithmetic or logical operation, if the result has an even number of 1's, the flag is set. If it has an odd number of 1's, the flag is reset.
  - **Carry Flag (CY):** If **an** arithmetic operation results in a carry, the carry, theCarry Flag is set; otherwise it is reset.

Table 6.1 Flag Register

D7	D6	D5	D4	D3	D2	D1	D0
S	Z		AC		P		CY

- **Instruction register and decoder:** It is an 8-bit register. When an instruction is fetched from memory then it is stored in the Instruction register. Instruction decoder decodes the information present in the Instruction register.
- **Timing and control unit:** It provides timing and control signal to the microprocessor to perform operations. Following are the timing and control signals, which control external and internal circuits −
    - **Control Signals:** READY, RD, WR, ALE
    - **Status Signals:** S0, S1, IO/M
    - **DMA Signals:** HOLD, HLDA
    - **RESET Signals:** RESET IN, RESET OUT
- **Interrupt control:** It controls the interrupts during a process. When a microprocessor is executing a main program and whenever an interrupt occurs, the microprocessor shifts the control from the main program to process the incoming request. After the request is completed, the control goes back to the main program. There are 5 interrupt signals in 8085 microprocessor: INTR, RST 7.5, RST 6.5, RST 5.5, and TRAP.
- **Serial Input/output control:** It controls the serial data communication by using these two instructions: SID (Serial input data) and SOD (Serial output data).
- **Address buffer and address-data buffer:** The content stored in the stack pointer and program counter is loaded into the address buffer and address-data buffer to communicate with the CPU. The memory and I/O chips are connected to these buses; the CPU can exchange the desired data with the memory and I/O chips.
- **Address bus and data bus:** Data bus carries the data to be stored. It is bidirectional, whereas address bus carries the location to where it should be stored and it is unidirectional. It is used to transfer the data & Address I/O devices.

Memory Interfacing

Interfacing is a technique to be used for connecting the Microprocessor to Memory.

Memory requires some signals to read from and write to registers and microprocessor transmits

some signals for reading or writing data.

The interfacing process includes some key factors to match with the memoryrequirements

and microprocessor signals.

The interfacing circuit therefore should be designed in such a way that it matches the memory signal requirements with the signals of the microprocessor.

IO Interfacing

Figure 6.3 shows the Memory Interface diagram. This consists of various communication devices like keyboard, mouse, printer, etc.

To interface the keyboard and other devices with the microprocessor, latches and buffers are used.

This type of interfacing is known as I/O interfacing.

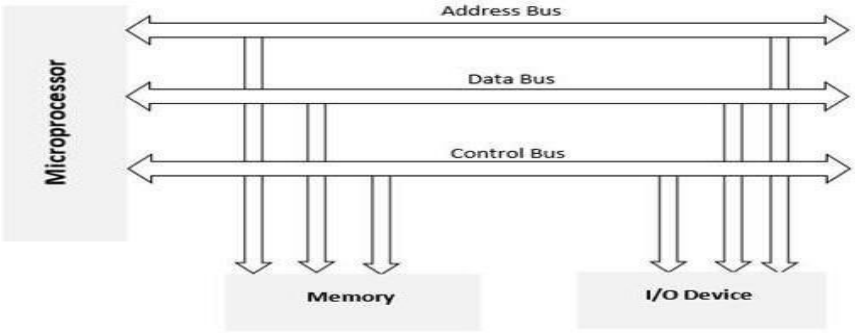


Figure 6.3 Memory Interface

6.2 8085 Instruction Set and Classifications

An instruction is a binary pattern designed inside a microprocessor to perform a specific function.

The entire group of instructions that a microprocessor supports is called Instruction Set.

8085 has 346 instructions.

Each instruction is represented by an 8-bit binary value.

- This set of instructions, written in a sequence, is called a **Program.**

These 8-bits binary value is called Op-Code or Instruction Byte.

Classification of Instruction Set

The Instruction set of 8085 can be c lassified into five different categor ies:

1. Data Transfer Instructions
2. Arithmetic Instructions
3. Logical Instructions
4. Branch Instructions

5. Control Instructions

Data Transfer Instructions

These instructions move data between registers, or between memory and registers.

These instructions copy data from source to destination.

While copying, the contents of the source are not modified.

Arithmetic Instructions

These instructions perform the operations like:

- Addition
- Subtraction
- Increment
- Decrement

Addition

- Any 8-bit number or the contents of register, or the contents of memory location can be added to the contents of accumulator.
- The result (sum) is stored in the accumulator.
- No two 8-bit registers can be added directly.

Subtraction

- Any 8-bit number or the contents of register, or the contents of memory location can be subtracted from the contents of accumulator.
- The result is stored in the accumulator.
- Subtraction is performed in 2's complement form.
- If the result is negative, it is stored in 2's complement form.
- No two other 8-bit registers can be subtracted directly.

Increment / Decrement

- The 8-bit contents of a register or a memory location can be incremented or decremented by 1.
- The 16-bit contents of a register pair can be incremented or decremented by 1.
- Increment or decrement can be performed on any register or a memory location.

Logical Instructions

- These instructions perform logical operations on data stored in registers, memory and

status flags.

❖ The logical operations are:

- AND
- OR
- XOR
- Rotate
- Compare
- Complement

AND, OR, XOR

- Any 8-bit data, or the contents of register, ot memory location can logically have
  - AND operation
  - OR operation
  - XOR operation

  with the contents of accumulator
- The result is stored in accumulator.

Rotate

- Each bit in the accumulator can be shifted either left or right to the next position.

Compare

- Any 8-bit data or the contents of register or memory location can be compared for:
  - Equality
  - Greater Than
  - Less Than

  with the contents of accumulator .
- The result is reflected in status flags.

Complement

- The contents of accumulator can be complemented.
- Each 0 is replaced by 1 and each 1 is replaced by 0.

Branching Instructions

- The branching instructions alter the normal sequential flow.
- These instructions alter either unconditionally or conditionally.

Control Instructions

- The control instructions control the operation of microprocessor.

EXERCISES:

1. What is bus?
2. What is the function of the accumulator?
3. What is a flag? What are its types?
4. Specify the four control signals commonly used by the 8085 MPU.
5. Explain about 8085 MPU Architecture with neat diagram.
6. Discuss about 8085 MPU pin concept with neat diagram.
7. Explain in detail about memory interface.
8. Explain about 8085 MPU instruction set.

****

Chapter 7 Programming in 8085: Code Conversion

****

In microcomputer applications, various number systems and codes are used to input data or to display results. In the microprocessor, Data Processing is usually performed in binary. This Chapter discusses about the various programming techniques used in the 8085 Microprocessor.

Code Conversion allows the user to translate a number that is represented using one coding system to other coding system.

The code conversion involves operations like:

1) BCD to Binary
2) Binary to BCD
3) ASCII to BCD
4) BCD to ASCII
5) Binary to ASCII
6) ASCII to Binary

7.1 BCD to Binary Conversion

- Converting a number represented in BCD to a number represented in Binary is equivalent to converting a Decimal number to Hexadecimal.
- An 8-bit BCD or 2 digit decimal has a range from $(00)_{10}$ to $(99)_{10}$.
- We know,

  $(98)_{10}$ = (1001 1000) in BCD and is equal to $(63)_{16}$ = (0110 0011) in Binary

Steps:

- To convert a BCD number to Binary is to separate the BCD digits as tens digit and units digit.
- For example, the BCD number 98 has 9 tens and 8 units.
- Therefore, 98 is separated as 09 and 08.
- Multiply 09 with $0A_H$, we get the result as $5A_H = (90)_{10}$.
- Add 08 to this product. $5A_H + 08_H =$ 62_H

  In this way the Decimal number $(98)_{10}$ is converted into a Hexadecimal number 62_H. Inside the registers and memory, these digits are represented in 1's and 0's. in effect, the BCD number 1001 1000 is converted to a binary number 0110 0010

Algorithm:

1. Load the BCD number into the accumulator
2. Move the accumulator value into the B register.
3. Obtain BCD_2 by ANDing the accumulator with 0FH and store it in C

4. Restore the original value of the accumulator by moving the value in B to A. AND the accumulator with F0H
5. If the value in the accumulator equals 0, then BCD_2 is the final answer and store it in the memory location, 2020H (arbitrary)
6. Else, shift the accumulator to right 4 times to obtain BCD_1. Next step is to multiply BCD_1 by 0AH
7. Multiplication: Move BCD_1 to D and initialize E with 0AH as the counter. Clear the accumulator to 0 and add D to it E number of times
8. Finally, add C to the accumulator and store the result in 2020H

ADDRESS	LABEL	MNEMONIC
2000H		LDA 201FH
2001H		
2002H		
2003H		MOV B, A
2004H		ANI 0FH
2005H		
2006H		MOV C, A
2007H		MOV A, B
2008H		ANI F0H
2009H		
200AH		JZ LOOP1
200BH		
200CH		
200DH		RRC
200EH		RRC
200FH		RRC
2010H		RRC
2011H		MOV D, A
2012H		XRA A
2013H		MVI E, 0AH
2014H		
2015H	LOOP2	ADD D
2016H		DCR E
2017H		JNZ LOOP2
2018H		
2019H		
201AH	LOOP1	ADD C
201BH		STA 2020H

201CH		
201DH		
201EH		HLT

Input:

Address	Data
201F	72

Output:

Address	Data
2020	48

7.2 Binary to BCD Conversion

- The conversion of binary to BCD is performed by dividing the number by the power of ten: the division is performed by the subtraction method.
- For example: Assume the Binary Number is

  $1\ 1\ 1\ 1\ 1\ 1\ 1\ 1_2$ (FFH) = $(255)_{10}$
- To represent this number in BCD requires 12- bits or 3 BCD digits.

Example:

A binary number is stored at location 800H. Convert the number into its BCD equivalent and store it to the memory location 8050H.

Input:

Address	Data
8000	34

Output:

Address	Data
8050	52

Address	HEX Codes	Labels	Mnemonics	Comments
F000	21, 00, 80		LXI H,8000H	Initialize memory pointer
F003	16, 00		MVI D,00H	Clear D register for Most significant Byte
F005	AF		XRA A	Clear Accumulator
F006	4E		MOV C, M	Get HEX data
F007	C6, 01	LOOP	ADI 01H	Count the number one by one
F009	27		DAA	Adjust for BCD count
F00A	D2, 0E, F0		JNC SKIP	Jump to SKIP
F00D	14		INR D	Increase D
F00E	0D	SKIP	DCR C	Decrease C register
F00F	C2, 07, F0		JNZ LOOP	Jump to LOOP
F012	6F		MOV L, A	Load the Least Significant Byte
F013	62		MOV H, D	Load the Most Significant Byte
F014	22, 50, 80		SHLD 8050H	Store the BCD
F017	76		HLT	Terminate the program

7.3 **ASCII to BCD Conversion**

Write an Assembl

y Language Program to convert ASCII to BCD.

ALGORITHM:

STEP 1 : Start.
STEP 2: Take the ASCII number in accumulator.
STEP 3: Subtract NUMBER from the accumulator.
STEP 4: Store the BCD result in memory
STEP 5: End.

PROGRAM:

ADDRESS	OPCODE	MNEMONICS	COMMENTS
4000	3A,50,40	LDA 4050_H	Take the ASCII number in accumulator
4003	D6,30	SUI 30_H	Subtract the number 30 from the accumulator.
4005	32, 51, 40	STA 4051_H	Store the Binary result in memory
4008	76	HLT	End of program

Input:

Address	Data
4050	39

Output:

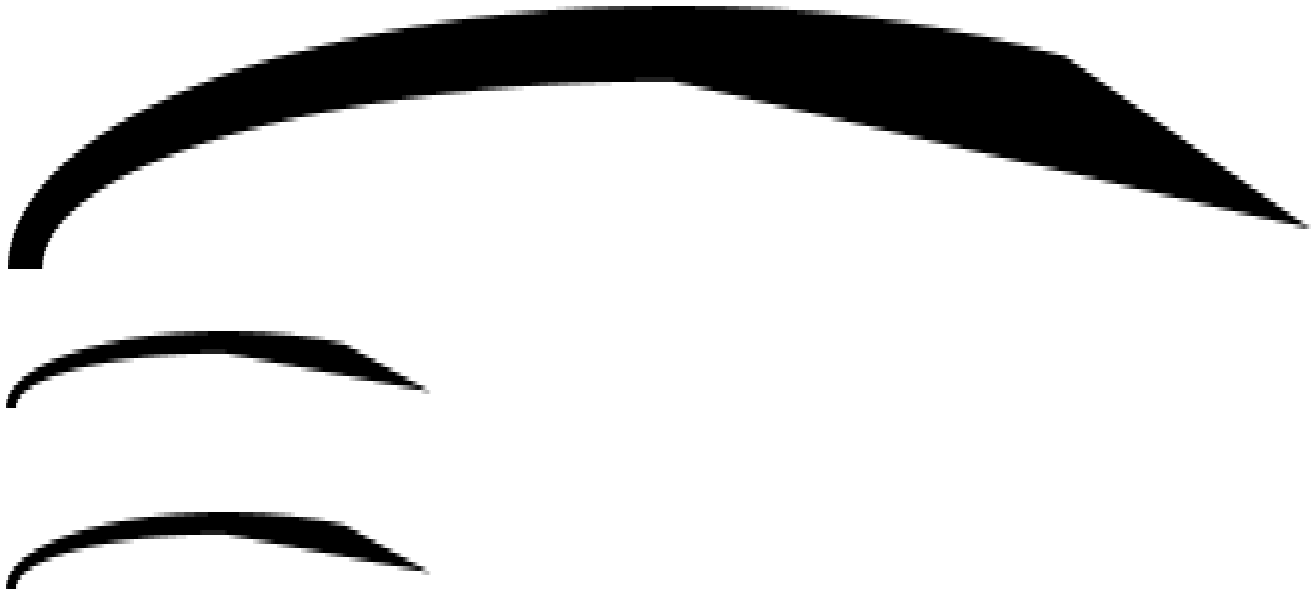
Address	Data
4051	09

7.4 **BCD to ASCII Conversion**

Write an assembly language program to convert 8 bit BCD number to its respective ACSII Code.

Algorithm:

1. Input the content of 2050 in accumulator
2. Move content of Accumulator to register B
3. Separate the least significant digit using AND with 0F and ADD 30 to accumulator
4. Store content of accumulator to memory location 3050
5. Move content of register B to Accumulator
6. Separate the most significant digit using AND with F0
7. Rotate Content of Accumulator 4 times
8. ADD 30 to accumulator
9. Store content of accumulator to memory location 3051



Program

Address	Mnemonics	Comments
2000	LDA 2050	Load the content of memory location 2050 to accumulator
2003	MOV B, A	Copy the content of accumulator to register B
2004	ANI 0F	AND the content of accumulator with immediate data 0F
2006	ADI 30	ADD 30 to accumulator
2008	STA 3050	Store the content of accumulator to memory location 3050
200B	MOV A, B	Copy the content of register B to accumulator
200C	ANI F0	AND the content of accumulator with immediate data F0
200E	RLC	Rotate the content of accumulator left without carry
200F	RLC	Rotate the content of accumulator left without carry
2010	RLC	Rotate the content of accumulator left without carry
2011	RLC	Rotate the content of accumulator left without carry
2012	ADI 30	ADD 30 to accumulator
2014	STA 3051	Store the content of accumulator to memory location 3051
2017	HLT	Stop the Execution

Input:

Address	Data
2050	98

Output:

Address	Data
3050	38
3051	39

7.5. Binary to ASCII Conversion

Write an assembly language program to convert Binary Code in to ASCII Code

ADDRESS	OPCODE	LABEL	MNEMONICS	COMMENTS
4000	3A		LDA 4050	Load the accumulator with Binary number
4001	50			
4002	40			
4003	FE		CPI 0A	Compare immediately the accumulator content with $0A_H$
4004	0A			
4005	DA		JC L1	If the number is less than 9 no need to add 7
4006	0A			
4007	40			
4008	C6		ADI 07	Add immediate data 07_H to the accumulator
4009	07			
400A	C6	L1	ADI 30	Add immediate data 30_H to the accumulator
400B	30			
400C	32		STA 4051	Store the Accumulator content at 4051
400D	51			
400E	40			
400F	76		HLT	Halt the execution

Input:

Address	Data
4050	0A

Output:

Address	Data
4051	41

Test Data:

4050 0A ; Binary Number (Input Data)

4051 41 ‘ ;ASCII Number (Output Data)

7.7. ASCII to Binary Conversion

Steps:

1. Read the ASCII number.
2. If the ASCII number is less than 3AH then 30H is subtracted from the number to get its Binary equivalent.
3. If the number is between 41H and 5AH then 37H is subtracted to get the Binary equivalent of the letter A-F.

Example:

41 H (ASCII) **= 41 H – 37 H**
= 04 H (BINARY)

Program:

ADDRESS	OPCODE	LABEL	MNEMONICS	COMMENTS
4000	3A		LDA 4050	Load the accumulator with Binary number
4001	50			
4002	40			
4003	D6		SUI 30	Subtract the number 30 from the Accumulator
4004	30			
4005	FE		CPI 0A	Compare immediately the accumulator content with 0AH
4006	0A			
4007	DA		JC L1	Add immediate data 07H to the accumulator
4008	0C			
4009	40			
400A	D6		SUI 07	Subtract the number 7 from the Accumulator
400B	07			
400C	32	L1	STA 4051	Store the Accumulator content at 4051
400D	51			

400E	**40**			
400F	**76**		**HLT**	**Halt the execution**

TEST DATA:

INPUT:

4050: 42_H

OUTPUT:

4051: $0B_H$

EXERCISES:

1. Discuss the conversion process of BCD to Binary form.
2. Discuss the conversion process of Binary to BCD form.
3. Write the steps for conversion process of ASCII to Binary and Binary to ASCII.
4. Write an assembly language program to convert Binary to ASCII.
5. Write an assembly language program to convert ASCII to Binary

****

Chapter 8 Programming in 8085: BCD Arithmetic

****

In microcomputer applications, various number systems and codes are used to input data or to display results. The ASCII (American Standard Code for Information Interchange) keyboard is a commonly systems. However, in this chapter includes arithmetic operations that are performed in BCD numbers.

8.1 BCD Addition:

- In some application, input/output data are presented in decimal numbers.
- The addition of two BCD numbers may not represent an appropriate BCD value.

Example: The addition of BCD (34, 26) results is 5AH.

34_{10}	**=**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**BCD**
26_{10}	**=**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**BCD**
60_{10}	**=**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**5A (Result)**

- The microprocessor cannot recognize BCD numbers.
- It adds any two numbers in binary.
- Table 8.1(b) shows the illustrative program for Addition of two BCD numbers.

Table 8.1(b) Program for 8-bit Addition of BCD Numbers

LDA	**2050H**	Get the addend from memory into accumulator
MOV	**B, A**	Transfer the number to B register
LDA	**2051H**	Take the second number to A
ADD	**B**	Add the two numbers in A and B
DAA		Convert the Hex result in accumulator to decimal
STA	**2052H**	Store the result in memory
HLT		End of program

8.2 BCD Subtraction:

- When subtracting two BCD numbers, the instruction DAA cannot be used directly to perform

BCD subtraction because DAA instruction requires an addition to be performedfirst.

- To subtract two BCD numbers by using the 10's compliment,
- Tables 8.2(c) illustrate the BCD subtraction program.

Example: 1 Subtract 39 from 85

Step 1: Take the 10's complement of 39 and then add 85.

Step 2: The result will be a two digit BCD with a carry and the carry is omitted.

Step 3: The result 85 – 39 = 46, in decimal.

Minuend = 85

Subtrahend = 39

9's complement of 39 = 99 – 39 = 60 (Each digit is subtracted from 9)

10's complement of 39 = 9's complement + 1 = 60 + 1 = 61

```
    85
+   61
-------
    1   46    (Omitting the carry, then the result is 46
```

Table 8.2(c) Program for BCD subtraction

LDA	**2050**	Take the minuend in accumulator
MOV	**B, A**	Transfer the minuend to B register
LDA	**2051**	Take the subtrahend in accumulator
MOV	**C, A**	Transfer the subtrahend to C register
MVI	**A, 99**	Load 99 in accumulator
SUB	**C**	Find the 9's complement of the subtrahend
INR	**A**	Increment the accumulator content by one to find the 10's complement of the subtrahend.
ADD	**B**	Add the minuend with the 10's complement of the subtrahend.
DAA		Convert the result to decimal
STA	**2052**	Store the result in memory
HLT		End of program

8.3 **Multibyte Addition and**

Subtraction: Multibyte Addition:

- Multibyte Addition using 4-byte BCD numbers.
- The numbers are stored into the memory location.

- The result is stored at location 85F0H.
- The HL pair is storing the address of first operand bytes, the DE is storing the address ofsecond operand bytes.
- C is holding the byte count.
- Multibyte Addition using the stack to store the intermediate bytes of the result.
- After completion of the addition operation, popping from the stack and storing into the destination.
- Table 8.3(a) (b) (c) shows the Input, Program, and Output of the Multibyte Addition.

Table 8.3(a) Input of Multibyte Addition

Address	8500	8501	8502	8503	8504	8505	8506	8507	8508
Data	04	19	68	12	85	88	25	17	20

Table 8.3(b) Multibyte Addition Program

Address	**HEX Codes**	**Labels**	**Mnemonics**	**Comments**
F000	31,00, 20		LXI SP, 2000H	Initialize Stack Pointer
F003	21,00, 85		LXI H, 8500H	Load memory address to get byte count
F006	4E		MOV C, M	Load memory content into C register
F007	06,00		MVI B, 00H	Clear B register
F009	21,01, 85		LXI H,8501H	Load first argument address
F00C	11,05, 85		LXI D,8505H	Load second argument address
F00F	1A	LOOP	LDAX D	Load DE with second operand address
F010	8E		ADC M	Add memory content and carry with Acc
F011	27		DAA	Decimal adjust the acc content
F012	F5		PUSH PSW	Store the accumulator content into the stack
F013	4		INR B	Increase b after pushing into a stack
F014	23		INX H	Increase HL pair to point next address
F015	13		INX D	Increase DE pair to point next address
F016	0D		DCR C	Decrease c to while all bytes are not exhausted

F017	C2,0F,F0		JNZ LOOP	When bytes are not considered, loop again
F01A	D2,21,F0		JNC SKIP	When carry = 0, jump to store
F01D	3E,01		MVIA, 01H	When carry = 1, push it into stack
F01F	F5		PUSH PSW	Store the accumulator content into the stack
F020	04		INR B	Increase b after pushing into the stack
F021	21,F0, 85	SKIP	LXIH,85F0H	Load the destination pointer
F024	F1	L1	POP PSW	Pop AF to get back bytes from the stack
F025	77		MOV M, A	Store Acc data at the memory location pointed by HL
F026	23		INX H	Increase HL pair to point next address
F027	05		DCR B	Decrease B
	C2,24,F0		JNZ L1	Go to L1 to store stack contents
F02B	76		HLT	Terminate the program

Table 8.3(c) Output of Multibyte Addition

Address	8500	8501	8502	8503	8504	8505	8506	8507	8508
Data	04	19	68	12	85	88	25	17	20

Multibyte Subtraction:

- The numbers are stored into memory, and one additional information is stored and the byte count of the multi-byte BCD number.
- This program uses 3-byte BCD numbers for multibyte subtraction.
- Numbers are stored at location 8001H to 8003H, and another number is stored at location8004H to 8006H.
- The location 8000H is holding the byte count. In this case the byte count is 03H.
- For the subtraction, using the 10's complement method for subtraction.
- In this case the numbers are: 672173 – 275188 = 376985
- Table 8.3(d) (e) (f) shows the Input, Program, and Output of the Multibyte Subtraction.

Table 8.3(d) Input of Multibyte Subtraction

Address	8000	8001	8002	8003	8004	8005	8006
Data	03	73	21	67	88	51	27

Table 8.3(e) Multibyte Subtraction Program

Address	**OP Code**	**Labels**	**Mnemonics**	**Comments**
F000	21, 00, 80		LXI H,8000H	Point to get the count
F003	4E		MOV C,M	Get the count to C
F004	11, 01, 80		LXI D, 8001H	Point to first number
F007	21, 04, 80		LXI H,8004H	Point to second number
F00A	37		STC	Set the carry flag
F00B	3E, 99	LOOP	MVI A,99H	Load 99H into A
F00D	CE,00		ACI 00H	Add 00H and Carry with A
F00F	96		SUB M	Subtract M from A
F010	EB		XCHG	Exchange DE and HL
F011	86		ADD M	Add M to A

F012	27		DAA	Decimal adjust
F013	77		MOV M,A	Store A to memory
F014	EB		XCHG	Exchange DE and HL
F015	23		INX H	Point to next location by HL
F016	13		INX D	Point to next location by DE
F017	0D		DCR C	Decrease C by 1
F018	C2, 0B, F0		JNZ LOOP	Jump to LOOP if Z = 0
F01B	76		HLT	Terminate the program

Table 8.3(f) Output of Multibyte Subtraction

Address	8001	8002	8003
Data	85	69	37

8.4 BCD Multiplication and Division

BCD Multiplication:

- The Program multiplies two decimal numbers and gives a two digit decimal result (Less than 99).
- It is using repeated addition method and after each addition, DAA is used to get the partial sum in BCD.
- When the multiplier is decremented, decrement in BCD.
- The DAA instruction does not work for decrement operations.
- By adding 99 and using the instruction DAA, the multiplier can be decremented by one in BCD.
- Table 8.4 (a) (b) (c) shows the Input, Program, and Output of the BCD Multiplication.

Table 8.4(a) Input of BCD Multiplication

Address	2050	2051
Data	08	12

Table 8.4(b) Program for BCD Multiplication

LDA	2050	Get the Multiplicand from memory
MOV	B, A	Transfer the multiplicand to B register
LDA	2051	Get the multiplier from memory
MOV	C, A	Transfer the multiplier to C register
XFA	A	Clear the accumulator carry and auxiliary flags
AGAIN: ADD	B	(A)□(A) +(B)
DAA		Convert the Hex number in accumulator to decimal
MOV	D, A	Save the content of accumulator in D register.
MOV	A, C	Transfer the multiplier into A register
ADI	99	Add 99 to the multiplier
DAA		Convert into decimal
MOV	C, A	Save it in C register
MOV	A, D	Transfer the saved value in D register to accumulator
JNZ	AGAIN	If (C) is not zero, continue in AGAIN
STA	2052	Store the product in memory
HLT		Terminate the program

Table 8.4(c) Output of the BCD Multiplication

Address: 2052	Data: 96

Division:

- The 8085 has no division operation. To get the result of the division, we should use the repetitive subtraction method.
- By using this program, we will get the quotient and the remainder.
- 8020H will hold the quotient, and 8021H will hold the remainder.
- We are saving the data at location 8000H and 8001H.
- The result is storing at location 8050H and 8051H.
- The table 8.4(d) (e) (f) shows the Input, Program, and Output of Division.

Table 8.4(d) Input of 8085 Division

The Dividend: 0EH
The Divisor 04H
The Quotient will be 3, and the remainder will be 2

Table 8.4(e) 8085 Division Program

Address	OPCODE	Labels	Mnemonics	Comments
F000	21,0E, 00	START	LXIH,0CH	Load 8-bit dividend in HL register pair
F003	06,04		MVIB,04H	Load divisor in B to perform num1 / num2
F005	0E,08		MVIC, 08	Initialize the counter
F007	29	UP	DADH	Shifting left by 1 bit HL = HL + HL
F008	7C		MOVA, H	Load H in A
F009	90		SUB B	perform A = A – B
F00A	DA,0F, F0		JC DOWN	If MSB < divisor then shift to left
F00D	67		MOVH, A	If MSB > divisor, store the current valueof A in H
F00E	2C		INR L	Tracking quotient
F00F	0D	DOWN	DCRC	Decrement the counter
F010	C2,07, F0		JNZ UP	If not exhausted then go again

Table 8.4(f) Output of 8085 Division Program

Address	Data
8020	03
8021	02

EXERCISE:

1. Define the term BCD.
2. Illustrate the program for addition of two BCD numbers.
3. Write the steps and program to calculate the BCD subtraction.
4. Describe about Multibyte Addition with a simple program.
5. Explain about BCD Multiplication with an example.
6. What are the steps followed to calculate the BCD division.

****

Chapter 9 Interrupts

****

An interrupt I/O is a process of data transfer whereby an external device or a peripheral can inform the processor that it is ready for communication and it requests attention. The process is initiated by an external device and is asynchronous, meaning that it can be initiated at any time without reference to the system clock. However, the response to an interrupt request is directed or controlled by the microprocessor.

This chapter includes a brief explanation of the interrupt requests that are classified in two categories: Maskable and Non-Maskable. The 8085 microprocessor includes four maskable and one Non-maskable interrupt. The interrupts in 8085 microprocessor and vector interrupts are also described later in the chapter.

When microprocessor receives any interrupt signal from peripheral(s) which are requesting its services, it stops its current execution and program control is transferred to a sub-routine by generating **CALL** signal and after executing sub-routine by generating **RET** signal again program control is transferred to main program from where it had stopped.

When microprocessor receives interrupt signals, it sends an acknowledgement (INTA) to the peripheral which is requesting for its service.

Interrupts can be classified into various categories based on different parameters:

1. **Hardware and Software Interrupts –**

   When microprocessors receive interrupt signals through pins (hardware) of microprocessor, they are known as *Hardware Interrupts*.

   There are 5 Hardware Interrupts in 8085 microprocessor.

   They are – *INTR, RST 7.5, RST 6.5, RST 5.5, TRAP*

   Software Interrupts are those which are inserted in between the program which means these are mnemonics of microprocessor.

   There are 8 software interrupts in 8085 microprocessor.

   They are – *RST 0, RST 1, RST 2, RST 3, RST 4, RST 5, RST 6, RST 7***.**

2. **Vectored and Non-Vectored Interrupts –**

Vectored Interrupts are those which have fixed vector address (starting address of sub-routine) and after executing these, program control is transferred to that address.

3. ***Non-Vectored Interrupts***

In this vector address is not predefined and the interrupting device gives the address of sub-routine for these interrupts.

INTR is the only non-vectored interrupt in 8085 microprocessor.

4. **Maskable and Non-Maskable Interrupts –**

Maskable Interrupts are those which can be disabled or ignored by the microprocessor.

These interrupts are either edge-triggered or level-triggered, so they can be disabled.

Maskable interrupts in 8085 microprocessor *INTR, RST 7.5, RST 6.5, RST 5.5*.

- **Non-Maskable Interrupts** are those which cannot be disabled or ignored by microprocessor.
- *TRAP* is a non-maskable interrupt.
- It consists of both level as well as edge triggering and is used in critical power failureconditions.

9.1 The 8085 Interrupt

- The 8085 interrupt process is controlled by the Interrupt Enable flip-flop, which isinternal to the processor and can be set or reset by using software instructions.
- If the flip-flop is enabled and the input to the interrupt signal INTR (pin 10) goes high,the microprocessor is interrupted.
- This is a maskable interrupt and it can be disabled. The 8085 has a non-maskable andthree additional vectored interrupt signals also.

The 8085 interrupt process has the following eight steps:

The interrupt process should be enabled by writing the instruction EI in the main program.

Step 1:

- ❖ The instruction EI sets the Interrupt Enable flip-flop.
- ❖ The Instruction DI resets the flip-flop and disables the interrupt process.
- ❖ EI (Enable Interrupt) is a 1-byte instruction
- ❖ DI (Disable Interrupt) is a 1-byte instruction.

Step 2:

- ❖ When the microprocessor is executing a program, it checks the INTR line during the execution of each instruction.

Step 3:

- If the line INTR is high and the interrupt is enabled, the microprocessor completes thecurrent instruction.
- It disables the Interrupt Enable flip-flop and sends a signal called INTA (Interrupt Acknowledge (active low)).
- This processor cannot accept any interrupt requests until the interrupt flip-flop is enabled again.

Step 4:

- The signal INTA is used to insert a restart (RST) instruction or CALL instruction through external hardware.
- The RST instruction is a 1-byte call instruction that transfers the program control to a specific memory location.

Step 5:

- When the microprocessor receives an RST instruction or CALL instruction, it saves the memory address of the next instruction on the stack.
- The program is transferred to the CALL location.

Step 6:

- The task to be performed is written as a subroutine at the specified location, the processor performs the task.
- This subroutine is known as a service routine.

Step 7:

- The service routine should include the instruction EI to enable the interrupt again.

Step 8:

- At the end of the subroutine, the RET instruction retrieves the memory address where the program was interrupted and continues the execution.

Multiple Interrupt and Priorities:

- Figure 9.1(a) shows the priority of the interrupts, when microprocessor receives multipleinterrupt requests simultaneously.
- It will execute the interrupt service request (ISR) according to the priority of the interrupts.

Figure 9.1(a) Multiple Interrupt and Priorities

- The schematic in Figure 9.1(b) implements multiple interrupting devices using an 8-to-3 priority encoder that determines the priorities among interrupting devices.
- The encoder provides appropriate combinations on its output lines A_0, A_1, and A_2, which are connected to data lines D_3, D_4, and D_5 through a tri-state buffer.
- The eight inputs to the encoder are connected to eight different interrupting devices.
- When an interrupting device requests service, one of the input lines goes low; this makes line E_0 high and interrupts the microprocessor.
- When the interrupt is acknowledged and the signal INTA enables the tri-state buffer, the code corresponding to the input is placed on lines D_5, D_4, and D_3.

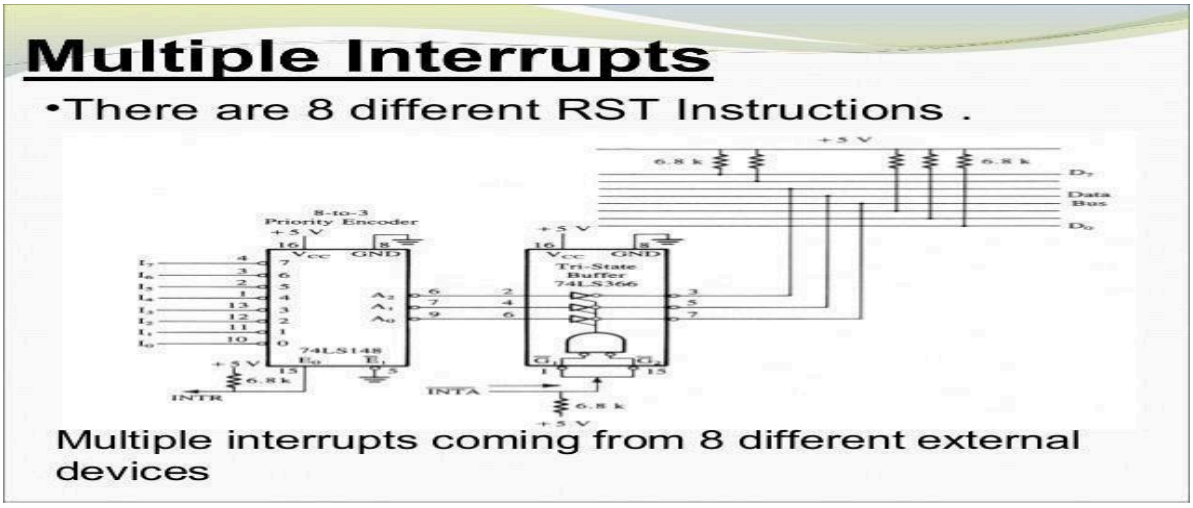


Figure 9.1 (b) Multiple Interrupts using priority encoder

9.2 **8085 Vector Interrupt**

- The 8085 has five interrupt inputs and vector locations, which is shown in the figure9.2 (a).
- One is called INTR, which is already discussed in the previous section, three are called RST 5.5, 6.5, and 7.5, respectively, and the fifth is called TRAP a non-maskable interrupt.
- These last four (RST and TRAP) are automatically vectored (transferred) to specific

locations on memory.

- They do not require the INTA signal or an input port; the necessary hardware is already implemented inside the 8085. The Table 9.2(a) shows the Vectored Interrupt with address.

Table 9.2(a) Vectored Interrupt

INTERRUPT	VECTOR ADDRESS
TRAP (RST 4.5)	24 H
RST 5.5	2C H
RST 6.5	34 H
RST 7.5	3CH

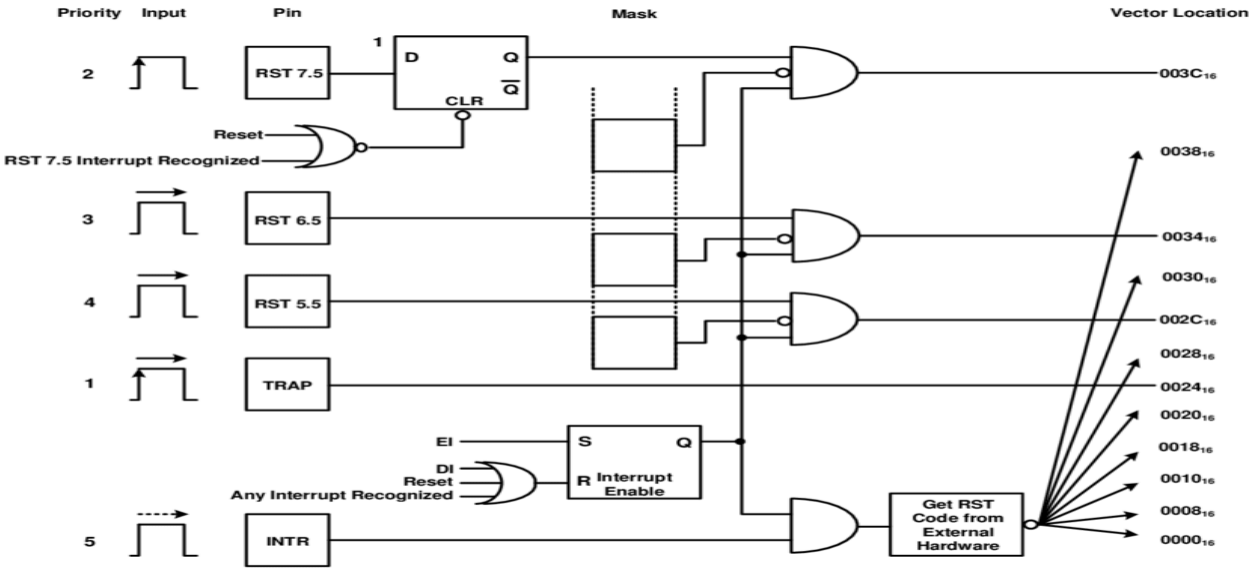


Figure 9.2(a) 8085Interrupt and Vector Locations

- The TRAP has the highest priority, followed by RST 7.5, 6.5, 5.5, and INTR, in that order.
- The TRAP has a lower priority than the hold signal used for DMA(will discuss next chapter).

TRAP

- TRAP, a nonmaskable interrupt known as NMI, is analogous to the smoke detector described earlier.
- It has the highest priority among the interrupt signals, it need not be enabled and it cannot be disabled.

- It is level and edge sensitive, meaning that the input should go high and stay high to be acknowledged.
- It cannot be acknowledged again until it makes a transition from high to low to high.
- TRAP is generally used for critical events as power failure and emergency shut-off.

RST 7.5, 6.5, and 5.5

- These maskable interrupts are enabled under program control with two instructions: EI(Enable Interrupt) described earlier, and SIM(Set Interrupt Mask) described below:
- Figure 9.2(b) shows the interpretation of the accumulator bit pattern for the SIM Instruction.

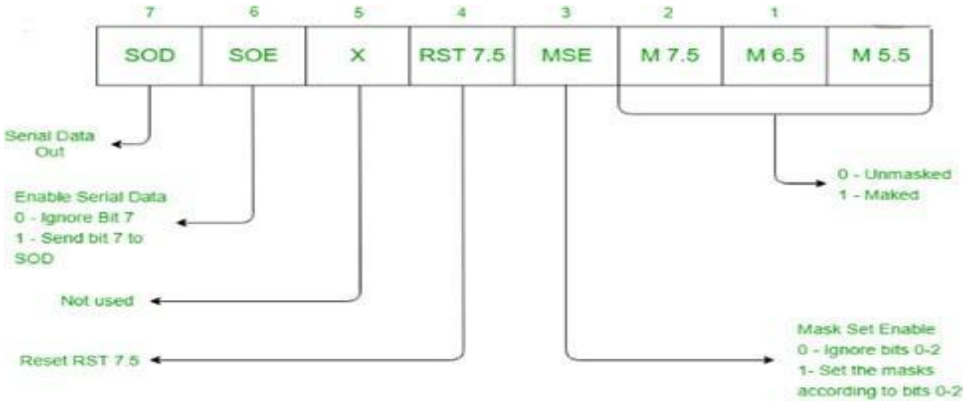


Figure 9.2(b) SIM (Set Interrupt Mask) Instruction

Instruction SIM (Set Interrupt Mask):

- This is a 1-byte instruction and can be used for three different functions.
- One function is to set mask for RST 7.5, 6.5, and 5.5 interrupts.
- This instruction reads the content of the accumulator and enables or disables the interrupts according to the content of the accumulator.
- Bit D3 is a control bit and should=1for bits D0, D1, and D2 to the effective.
- Logic 0 on D0, D1, and D2 will enable the corresponding interrupts, and logic 1 will disable the interrupts.
- The second function is to reset RST 7.5 flip-flop.

- Bit D4 is additional control for RST 7.5.

- If D4 =1, RST 7.5 is reset.
- This is used to override RST 7.5 without servicing it.
- The third function is to implement serial I/O.
- Bit D7 and D6 of the accumulator are used for serial I/O abd do not affect the interrupts. Bit D6=1 enables the serial I/O and bit D7 is used to transmit(output) bits.

EXERCISE

1. Define an Interrupt.
2. Explain the 8085 interrupt service execution in detail.
3. Describe about interrupts Classification categories based on different parameters.
4. Explain about Hardware and Software interrupts in 8085.
5. What is meant by Maskable and Non-masable interrupts in 8085?
6. Explain about 8085 vector Interrupt with neat diagram.
7. What is the use of TRAP interrupt signal?
8. What is the use of RST 7.5, 6.5, 5.5 interrupts?
9. What is meant by SIM?

UNIT – V

****

Chapter 10 DMA

****

This chapter describes about Direct Memory Access (DMA) and its working principle. This chapter discusses DMA Controller and several programmable devices from the Intel: The 8257 DMA controller and 8255A Peripheral Interface. The 8255A is the widely used general-purpose programmable devices, can be compatible with any microprocessor. It includes three programmable ports, one of which can be used for bidirectional data transfer. Later part of this chapter describes about the topics Basic features of Advanced Microprocessors and Pentium – I3, I5, and I7.

10.1 Direct Memory Access (DMA)

- The Direct Memory Access (DMA) is a process of communication or data transfer controlled by an external peripheral.
- In these situations in which the microprocessor-controlled data transfer is very slow, the DMA is generally used.
- DMA (Direct Memory Access) controller is being used in graphics cards, network cards, sound cards etc. DMA is also used for intra-chip transfer in multi-core processors.
- Ex: Data transferred between a floppy disk and R/W memory of the system.
- The 8085 Microprocessor has two pins available for this type of I/O communication:

1. HOLD (Hold):

- This is an active high input signal to the 8085 from another master requesting the use of the address and data buses.
- After receiving the HOLD request, the MPU relinquishes the buses in the following machine cycle.
- All buses are tri-stated and a Hold Acknowledge (HLDA) signal is sent out.
- The MPU regains the control of buses after HOLD goes low.

2. HLDA (Hold Acknowledge):

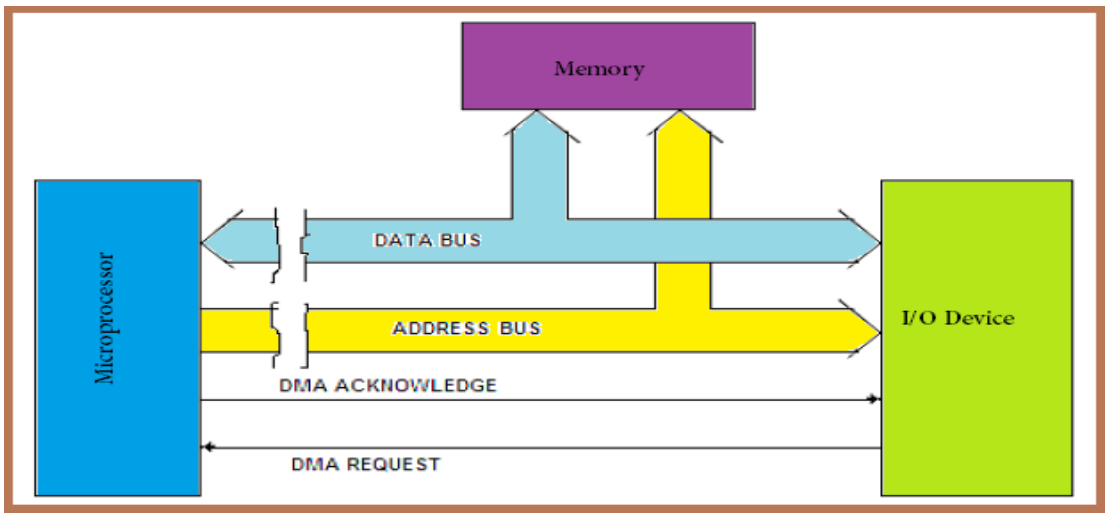
- This is an active high output signal indicating that the MPU is relinquishing the control of the buses.

- Figure 10.1(a) shows the DMA Data Transfer, In that DMA sends request to the microprocessor in form of HOLD signal by the I/O device.
- When microprocessor receipt of such request, the microprocessor relinquishes the

address and data buses.

- And informs the I/O devices of the situation by sending Acknowledge signal HLDA.
- The I/O device withdraws the request when the data transfer between the I/O device and external memory will complete.

Figure 10.1(a) DMA Data Transfer



- A DMA controller uses these signals as if it were a peripheral requesting the MPU for the control of the buses.
- The MPU communicates with the controller by using the chip select line, buses, and control signals.
- Once the controller has gained control, it plays the role of a processor for data transfer.
- To perform this function the DMA controller should have
  1. A Data Bus
  2. An Address Bus
  3. Read/Write control signals, and
  4. Control signals to disable its role as a peripheral and to enable its role as a processor.

- This process is called switching from the slave mode to master mode.
- The DMA transfers the data in three modes which include the following.

a) **Burst Mode**: In this mode DMA handover the buses to CPU only after completion of whole data transfer. Meanwhile, if the CPU requires the bus it has to stay ideal and wait for data transfer.

b) **Cycle Stealing Mode**: In this mode, DMA gives control of buses to CPU after transfer of every byte. It continuously issues a request for bus control, makes the transfer of one byte and returns the bus. By this CPU doesn't have to wait for a long time if it needs a bus for higher priority task.

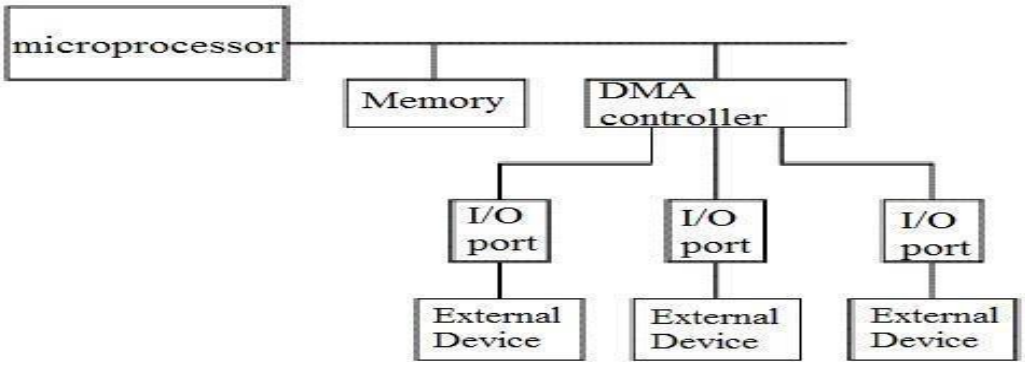
c) **Transparent Mode:** Here, DMA transfers data only when CPU is executing the instruction which does not require the use of buses.

10.2 8257 DMA Controller

DMA Controller

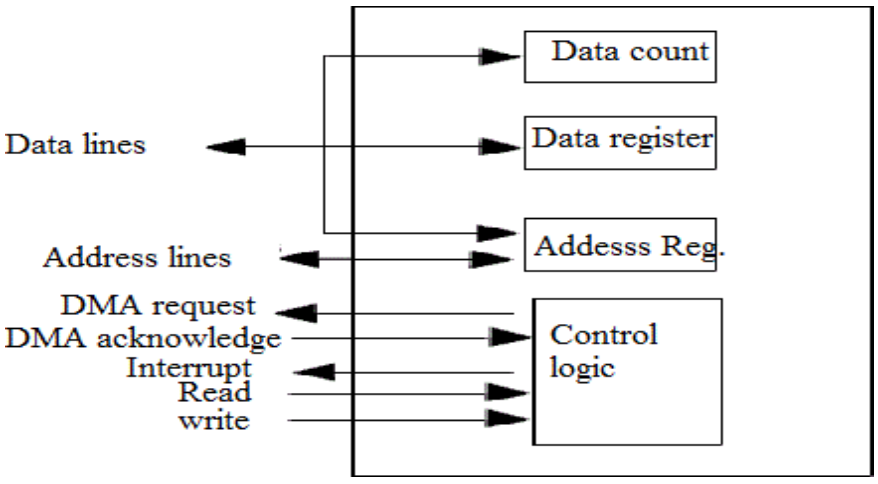
- DMA controller provides an interface between the bus and the input-output devices.
- It transfers data without intervention of processor and it is controlled by the processor.
- The processor initiates the DMA controller by sending the starting address.
- Direction of data transfer between I/O devices to the memory or from main memory to I/O devices shows in the figure 10.2(a).
- More than one external device can be connected to the DMA controller.

Figure 10.2(a) DMA Controller



- DMA controller contains an address unit, for generating addresses and selecting I/O device for transfer.
- It also contains the control unit and data count for keeping counts of the number of blocks transferred and indicating the direction of transfer of data.
- When the transfer is completed, DMA informs the processor by raising an interrupt.
- The typical block diagram of the DMA controller is shown in the figure 10.2(b) below.

Figure 10.2(b) Block Diagram of DMA Controller



8257 DMA Controller

The 8257 DMA controller forms a complete 4 channel **DMA controller**. Upon receiving a transfer request the 8257 controller-

Acquires the control over system bus from the processor.

The peripheral connected to the highest priority channel is acknowledged.

The least significant bits of the memory address are moved over the address lines A0-A7 of the system bus.

The most significant 8 bits of the memory address are driven to 8212 I/O port through data lines.

Generates the appropriate controls signals for the transfer of data between peripherals and addressed memory locations.

When the specified number of bytes is transferred, the controller informs the CPU end of transfer by activating the terminal count (TC) output.

For each channel 8257 contains two 16-bit registers.

1) DMA address register
2) Terminal count register, which should be initialized before a channel is enabled.

The address of first memory location to be accessed is loaded in the DMA address register.

The lower order 14 bits of the value loaded in the terminal count register indicates the number of DMA cycles minus one before the activation of Terminal count output.

Type of operation for a channel is indicated by the most significant two bits of the Terminal count register.

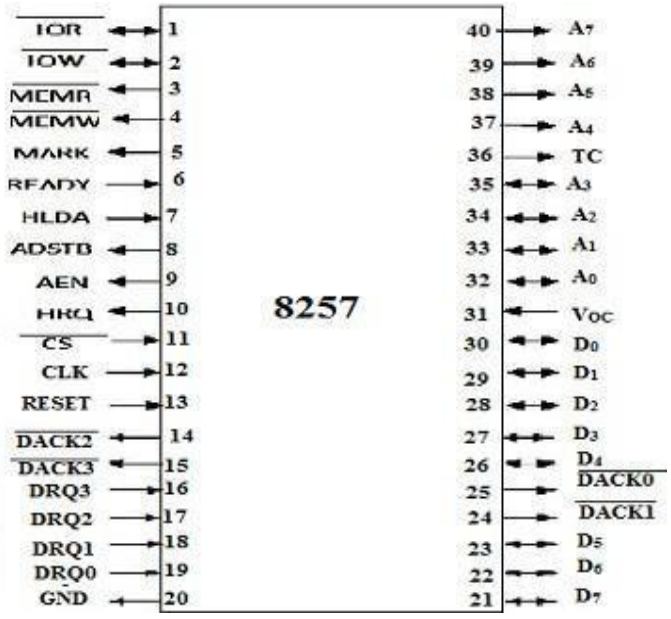


Figure 10.2(c) 8257 Pin Diagram

Advantages and Disadvantages of DMA Controller

The advantages and disadvantages of DMA controller include the following.

Advantages

- DMA speedups the memory operations by bypassing the involvement of the CPU.

The work overload on the CPU decreases.

For each transfer, only a few numbers of clock cycles are required

Disadvantages

Cache coherence problem can be seen when DMA is used for data transfer.

Increases the price of the system.

10.3 8255A Programmable Peripheral Interface

- The 8255A is a widely used, programmable, parallel I/O device.
- It can be programmed to transfer data under various conditions, from simple I/O to interrupt I/O.
- It is flexible and economical, but complex.
- As shown in the figure 10.3(a) The 8255A has 24 I/O pins that can be grouped primarilyin two 8-bit parallel ports:
  - Port A and Port B
    - The remaining eight bits as port C.
    - eight bits of port C can be used as individual bits or grouped in two 4-bits ports:
  - CUPPER (CU)
  - CLOWER(CL**(b) 8255A Modes**

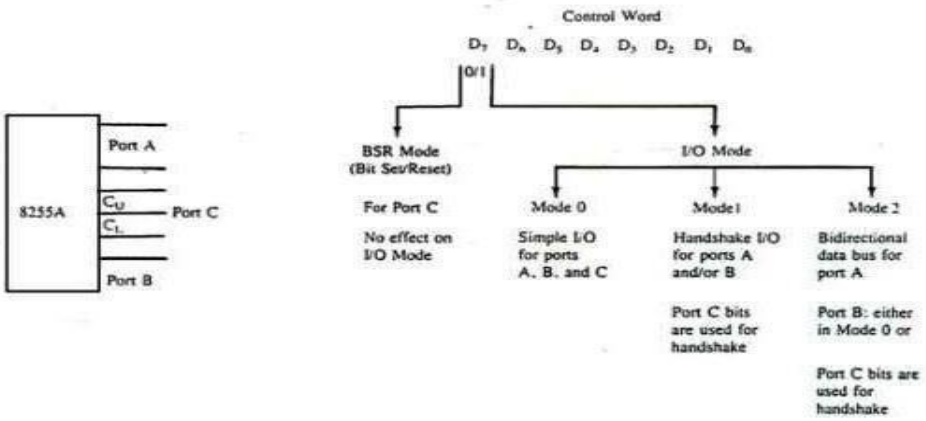


Figure 10.3 (a) 8255A I/O Ports

Operating Modes

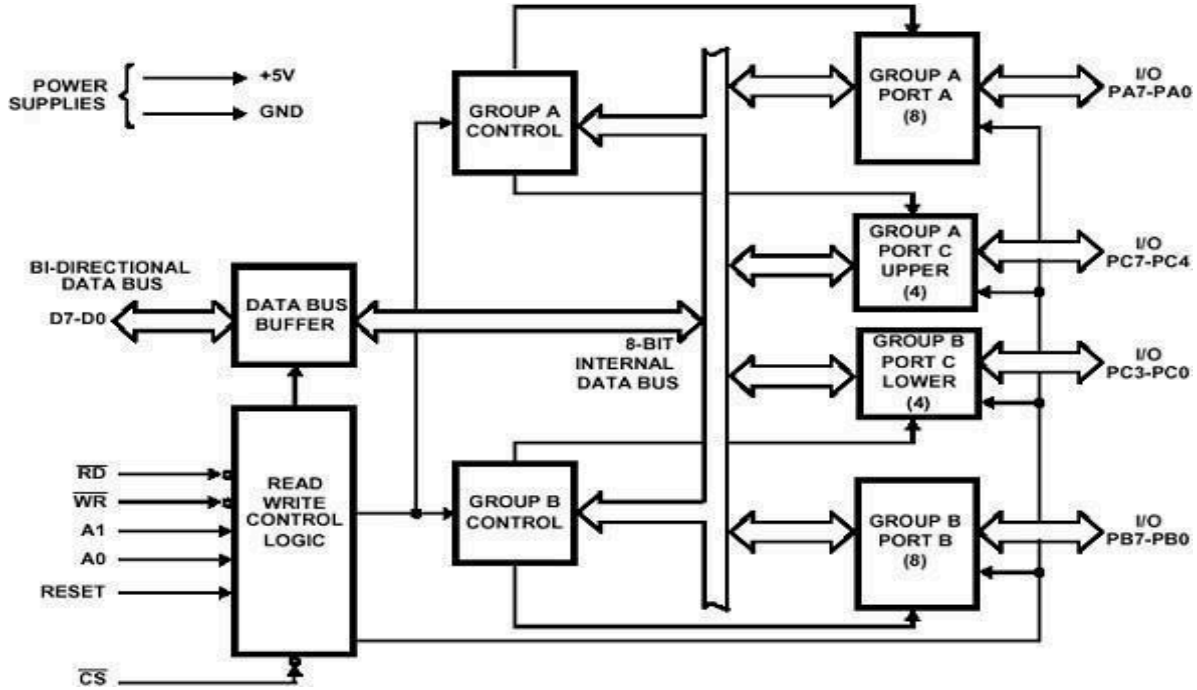
8255A has three different operating modes −

- **Mode 0** − In this mode, Port A and B is used as two 8-bit ports and Port C as two 4-bit ports. Each port can be programmed in either input mode or output mode where outputs are latched and inputs are not latched. Ports do not have interrupt capability.
- **Mode 1** − In this mode, Port A and B is used as 8-bit I/O ports. They can be configured as either input or output ports. Each port uses three lines from port C as handshake signals. Inputs and outputs are latched.
- **Mode 2** − In this mode, Port A can be configured as the bidirectional port and Port B either in Mode 0 or Mode 1. Port A uses five signals from Port C as handshake signals for data transfer. The remaining three signals from Port C can be used either as simple I/O or as handshake for port B.

8255 Architecture

Figure 10.3(c) shows the architecture of 8255A

Figure 10.3(c) Architecture of 8255A



Data Bus Buffer

It is a tri-state 8-bit buffer, which is used to interface the microprocessor to the system data bus. Data is transmitted or received by the buffer as per the instructions by the CPU. Control words and status information is also transferred using this bus.

Read/Write Control Logic

This block is responsible for controlling the internal/external transfer of data/control/status word. It accepts the input from the CPU address and control buses, and in turn issues command to both the control groups.

Chip Select (CS)

It stands for Chip Select. A LOW on this input selects the chip and enables the communication between the 8255A and the CPU. It is connected to the decoded address, and A_0 & A_1 are connected to the microprocessor address lines. Table 10.3(a) shows the result depends on conditions.

Table 10.3(a) Result that Depends on Conditions

CS	A_1	A_0	Result
0	0	0	PORT A
0	0	1	PORT B
0	1	0	PORT C
0	1	1	Control Register
1	X	X	No Selection

Write (WR)

It stands for write. This control signal enables the write operation. When this signal goes low, the microprocessor writes into a selected I/O port or control register.

RESET

This is an active high signal. It clears the control register and sets all ports in the input mode.

Read (RD)

It stands for Read. This control signal enables the Read operation. When the signal is low, the microprocessor reads the data from the selected I/O port of the 8255.

A_0 and A_1

These input signals work with RD, WR, and one of the control signal. Following is the table 10.3(b) showing their various signals with their result.

Table 10.3(b) Various Signals with their Result

A_1	A_0	RD	WR	CS	Result
0	0	0	1	0	**Input Operation** PORT A → Data Bus
0	1	0	1	0	PORT B → Data Bus
1	0	0	1	0	PORT C → Data Bus
0	0	1	0	0	**Output Operation** Data Bus → PORT A
0	1	1	0	0	Data Bus → PORT A
1	0	1	0	0	Data Bus → PORT B
1	1	1	0	0	Data Bus → PORT D

10.4 Basic Features of Advanced Microprocessors

Intel 80186

- 80186 contain a 16-bit data bus.
- The internal register structure of the 80186 is virtually identical to the 8086.
- The only difference is that the 80186 contain additional reserved interruptvectors and some very powerful built in I/O features.
- The 80186 and 80188 are often called embedded controllers because of their application as a controller, not as a microprocessor-based computer.

Features

- The 80186 contains 16 bit data bus.
- The internal register structure of 80186 is virtually identical to the 8086.
- Enhanced 8086-2 CPU.
- Clock Generator.
- 2 Independent DMA Channels.
- Programmable Interrupt Controller.
- 3 Programmable 16-bit Timers.
- Programmable Memory and Peripheral Chip-Select Logic.
- Available in 10 MHz and 8 MHz Versions.

I/O.

Direct Addressing Capability to 1 Mbyte of Memory and 64 Kbyte

- Following figure 10.4(a) shows the Architecture Diagram of 80186.

Intel 80286

- The 80286 is the first member of the family of advanced microprocessors with memory management

and protection abilities.

Features

- The 80286 microprocessor is an advanced version of the 8086 microprocessor that is designed for multi user and multitasking environments.
- The 80286 addresses 16 M Byte of physical memory and 1G Bytes of virtualmemory by using its memory-management system.
- The 80286 is basically an 8086 that is optimized to execute instructions in fewer locking periods than the 8086.

- Various versions of 80286 are available that runs on 12.5 MHz , 10 MHz and8 MHz clock frequencies.
- Like the 80186, the 80286 doesn't incorporate internal peripherals; instead it contains a memory management unit (MMU).
- The 80286 operates in both the real and protected modes
- In the real mode, the 80286 addresses a 1MByte memory addressspace and is virtually identical to 8086.
- In the protected mode, the 80286 addresses a 16MByte memory space.

The CPU contain four functional blocks

1. Address Unit (AU)
2. Bus Init (BU)
3. Instruction Unit(IU)
4. 4.Execution Unit (EU)

Address Unit: The address unit is responsible for calculating the physical address of instructions and data that the CPU wants to access.

Bus Unit: Major function of the bus unit is to fetch instruction bytes from the memory. Instructions are fetched in advance and stored in a queue to enable faster execution of the instructions.

Instruction Unit: The instruction unit (IU) accepts instructions from the prefetch queue and an instruction decoder decodes them one by one.

Execution Unit: The execution unit (EU) is responsible for executing the instructions received from decoded instruction queue. The decoded instruction queue sends the data part of the instruction over the data bus.

Intel 80386 Features

- 32-bit general and offset registers.
- 16-byte prefetch queue.
- Memory Management Unit with a Segmentation Unit and a Paging Unit.
- 32-bit Address and Data Bus.
- 4-Gbyte Physical address space.
- 64-Tbyte virtual address space.
- 64K 8-, 16-, or 32-bit ports.
- Implementation of real, protected and virtual 8086 modes.

Intel 80486

Features

- Improved 80386 CPU (6 extra instructions)
- Hard-wired implementation of frequently used instructions (as in RISCs)
- A 5 stage instruction pipeline
- An 8K Cache Memory + cache controller (previously a separate device)
- An on-chip Floating Point coprocessor Longer Prefetch Queue (32-bytes as opposed to 16 on the 80386)
- Higher frequency operation
- About a million transistors
- Like the 80386 it uses real, protected and virtual 8086 modes and its Memory Management Unit include a Segmentation Unit and a PagingUnit.

The Pentium Processors

- The Pentium microprocessor signals an improvement to the architecture found in the 80486 microprocessor.
- The changes include an improved cache structure, a wider data bus width, a faster numeric coprocessor, a dual integer processor, and branch prediction logic.
- Table 10.4 describes the comparison between the Pentium processors.

Features of Pentium Processors:

- It is a highly integrated device containing about 1.2 million transistors.
- **Faster Floating Point Unit:** Faster algorithm provides up to ten times speed-upfor common operations including add, multiply and load.
- **Improved Cache Structure:** Pentium processors include separate code and datacaches integrated on-chip to meet performance goals.

- **D**

  ual Integer Processor: Pentium processor has integer processor. It allows

execution of two instructions per clock.

Data Integrity and Error Detection: The Pentium processors have added significant data integrity and error detection capability.

Pentium II

Pentium II is also a 32-bit processor with 64-bit data bus and 36-bit address bus to address up to 64GB of physical memory space. It is actually a Pentium pro processor with on-chip MMX (Multi Media Extension).

The features of Pentium II processor are;

(i) Supports the INTEL architecture with dynamic execution.

(ii) Integrated primary (L1) 16-kb instruction cache and 16-kb write back data cache.

(iii) Integrated 256kb second level (L2) cache.

(iv) Fully compatible with previous microprocessors.

(v) Supports MMX technology.

(vi) Quick start and Deep sleep modes provide extremely low power dissipation.

(vii) Integrated math co-processor.

Pentium III

The Pentium III microprocessor is an improved version of the Pentium II microprocessor. Even though it is newer than the Pentium II, it is still based on thePentium Pro architecture.

The features are:

1. P-III CPU has been developed using 0.25 micron technology and includes over 9.5 million transistors. It has three versions operating at 450 MHz, 500 MHz and 550MHz which are commercially available.
2. P-III incorporates multiple branch prediction algorithms.
3. Dual independent bus architecture increases bandwidth.
4. P-III employs dynamic execution technology.
5. A 512Kbyte unified, non-blocking level 2 cache has been used.
6. Eight 64-bit wide Intel MMX registers along with a set of 57 instructions for multimedia applications are available.

Pentium IV

The most recent version of the Pentium Pro architecture microprocessor is the Pentium 4 microprocessor from Intel. The Pentium 4 was released initially in November2000 with a speed of 1.3 GHz. It is currently available in speeds up to

2.1 GHz. There are two packages available for this integrated microprocessor, the 423-pin PGA and the 478-pin FC- PGA2. Both versions use the 1.8 micron

technology for fabrication. As with earlier versions of the Pentium, the Pentium 4 uses a 100-MHz memory bus speed.

Table 10.4 Comparison between different Processors

Name	Year	Internal Reg:	Clock Speed	Data Width	Address Lines	Max. Memory Space
8086	1974	16 Bit	2 MHZ	16 bits	20 Bit	1MB
80286	1982	16 Bit	6 MHZ	16 bits	24 Bit	16 MB
80386	1985	32 Bit	16 MHZ	32 bits	32 Bit	4 GB
80486	1989	32 Bit	25 MHZ	32 bits	32 Bit	4 GB
Pentium	1993	32 Bit	60 MHZ	32 bits, 64 bit bus	32 Bit	4 GB
PentiumII	1997	32 Bit	233 MHZ	32 bits, 64 bit bus	32 Bit	64 GB
PeniumIII	1999	32 Bit	450 MHZ	32 bits , 64 bit bus	32 Bit	64 GB
PentiumIV	2000	32 Bit	1.5 MHZ	32 bits, 64 bit bus		64 GB

10.5 Pentium – I3, I5, and I7

Intel launched its Pentium processor in 1993. It was a combination of two processors on a single chip with 3.1 million transistors. Pentium does not support HD video technology and gaming technologies. It also lacks multitasking support. Pentium processor is a low- end version of core processors.

Intel developed and manufactured dual-core i3 processors. It is the first processor in the i-series.

Pentium and Core processors both do the work according to their capacity. Both are designed for modern computer systems.

Intel's current core processors are divided into **three ranges**; Intel Core i3, Intel Core i5and Intel Core i7. Their relative levels of processing power are based on a collection of criteria involving their **Number of cores, clock speed (in GHz), size of cache, Turbo Boost and Hyper-Threading.**

- **Number of cores:**

- A core is usually the basic computation **unit of the CPU** which is describe in the table 10.5(a). It receives instructions and performs calculations, or actions, basedon those instructions.
- A processor with two cores is called a **dual-core processor** and four cores is called a **quad-core processor**.
- For example, an i5 4690k has 4 cores.

Table 10.5(a) Number of Cores in i3, i5, and i7

	Core i3	Core i5	Core i7
Number of cores	2	4	4

Clock speed:

- The clock speed is the most common way to measure a CPU's speed. The CPU requires a fixed number of clock cycles, to execute each instruction. In the table 10.5(b) shows the clock speed of the core i3, i5, and i7. The CPU speed determines how many calculations it can perform in one second of time**.**
- The components like RAM, hard drive, motherboard, and the number of processor cores (e.g., dual core or quad core) are also have to upgrade for improve the computer speed.

Table 10.5 (b) Clock Speed of Cores in i3, i5, and i7

	Core i3	Core i5	Core i7
Clock speed	3.4–4.2GHz	2.4–3.8GHz	2.9–4.2GHz

- **Hyper-Threading:**

The table 10.5(c) shows the Hyper Threading technology in i3, i5, and i7.Intel's implementation of Simultaneous Multi-Threading is known as Hyper-Threading Technology. It's available on Core i7 and Core i3 but not on the mid-range core i5.

- Hyper-Threading Technology uses processor resources more efficiently, enabling multiple threads to run on each core.

Table 10.5(c) Hyper-Threading Technology

	Core i3	Core i5	Core i7
Hyper-Threading	Yes	No	Yes

- **Turbo Boost**

- Table 10.5(d) describes about Turbo boost, which monitors the present processor

usage to determine how close the processor is to the maximum **thermal design power**.

- When the total number of cores is being used, the processor can turn off the unused cores and increase the clock speed on the rest of the cores. This is like **dynamic over clocking**, when the system demands.
- For example, by default the processor runs at 2.3 GHz, and when under heavy load, it will automatically speed up the cores up to 3.3 GHz. Turbo boost is present in most modern Intel processors (not in i3). Turbo boost is used to save battery and enhance usage based performance.

Table 10.5(d) Turbo Boost in Intel

	Core i3	Core i5	Core i7
Turbo Boost	No	Yes	Yes

- **Cache memory**

- A cache is a smaller, **faster memory**, closer to a processor core, which stores copies of the data from frequently used main memory locations.
- They were built in to decrease the time taken to access data by the **processor**. This time taken is called latency.

EXERCISES:

1. Define the term DMA.
2. What is the use of HOLD and HLDA Pins in 8085 microprocessor?
3. Explain about the working principle of DMA with neat diagram.
4. Define DMA Controller.
5. What are the three DMA data transfer modes?
6. Describe about DMA Controller.
7. Explain about 8257 DMA Controller with neat diagram.
8. Explain the concept of 8255A programmable peripheral interface.
9. Describe about the basic features of Advanced Microprocessors.
10. Compare the i3, i5, and i7 processors.

REFERENCE BOOKS:

1. M. M. Mano, “Computer System Architecture”. Pearson, Third Edition, 2007 R. S. Gaonkar- "Microprocessor Architecture- Programming and Applications with 8085"- 5th Edition- Penram-2009.
2. Mathur- “Introduction to Microprocessor”- 3rd Edition- Tata McGraw-Hill-1993.
3. V. Vijayendran- “Fundamentals of Microprocessors – 8085”- S. Viswanathan Pvt. Ltd.- 2008.
4. V. Vijayendran - Digital Electronics & Microprocessor – 8085