

Digital Computer Fundamentals

This book is a part of the course by **Jaipur National University**, Jaipur.
This book contains the course content for Digital Computer Fundamentals.

JNU, Jaipur
First Edition 2013

The content in the book is copyright of **JNU**. All rights reserved.

No part of the content may in any form or by any electronic, mechanical, photocopying, recording, or any other means be reproduced, stored in a retrieval system or be broadcast or transmitted without the prior permission of the publisher.

JNU makes reasonable endeavours to ensure content is current and accurate. **JNU** reserves the right to alter the content whenever the need arises, and to vary it at any time without prior notice.

Index

I. Content.....	II
II. List of Figures	VI
III. List of Tables.....	VII
IV. Abbreviations	VIII
V. Application.....	105
VI. Bibliography	112
VII. Self Assessment Answers.....	115
Book at a Glance	

Contents

Chapter I.....	1
Number System and Codes	1
Aim	1
Objectives	1
Learning outcome	1
1.1 Introduction.....	2
1.2 Binary System.....	2
1.2.1 Binary to Decimal Conversion	4
1.3 Decimal to Binary Conversion.....	4
1.4 Octal Number System	6
1.4.1 Octal to Decimal Conversion.....	7
1.4.2 Decimal to Octal Conversion.....	7
1.4.3 Octal to Binary Conversion	8
1.4.4 Binary to Octal Conversion	9
1.5 Hexadecimal Number System.....	9
1.5.1 Hex to Decimal Conversion.....	10
1.5.2 Decimal to Hex Conversion.....	10
1.5.3 Hex to Binary Conversion	11
1.5.4 Binary to Hex Conversion	11
1.5.5 Hex to Octal Conversion.....	12
1.5.6 Octal to Hex Conversion.....	12
1.6 Codes.....	12
1.6.1 BCD Code.....	12
1.6.2 ASCII Code.....	13
1.6.3 Code Gray	13
1.6.4 Excess 3	13
1.7 Binary Arithmetic.....	13
1.7.1 Addition	14
1.7.2 Addition of Signed Numbers	15
1.7.3 Subtraction	15
1.7.4 Multiplication.....	15
1.7.5 Division.....	16
Summary.....	18
References.....	18
Recommended Reading.....	18
Self Assessment.....	19
 Chapter II	 21
Logic Gates and Boolean Algebra	21
Aim	21
Objectives	21
Learning outcome	21
2.1 Introduction.....	22
2.2 Logic Gates	22
2.2.1 NOT Gate.....	22
2.2.2 OR Gate	22
2.2.3 AND Gate	22
2.2.4 NAND Gate	23
2.2.5 NOR Gate	23
2.2.6 XOR Gate	23
2.2.7 XNOR	23
2.3 Boolean Algebra.....	24
2.3.1 Fundamental Laws	24

2.3.2 DeMorgan's Theorems.....	24
2.3.3 Boolean Identities	25
2.4 Logic Minimisation.....	26
2.5 Karnaugh Maps (K-Maps)	27
2.5.1 Sum-of-Products Equations and Logic Circuits (SOP)	27
2.5.2 Product of Sums (POS).....	28
2.5.3 Drawing Karnaugh Maps.....	28
2.5.4 Don't Care Conditions	29
2.6 Quine - McCluskey Method.....	29
Summary.....	30
References.....	30
Recommended Reading.....	30
Self Assessment.....	31
 Chapter III.....	 33
Introduction to 8085 Microprocessor	33
Aim	33
Objectives	33
Learning outcome	33
3.1 Introduction	34
3.2 Features	34
3.3 Architecture of 8085 Microprocessor	35
3.4 I/O and Memory Interfacing	35
3.4.1 I/O Devices and their Interfacing.....	36
3.4.2 IO Addressing	37
3.4.3 Interfacing of Input Device.....	37
3.4.4 Interfacing Output Data	39
3.5 The 8085 Programming Model.....	40
3.6 The 8085 Addressing Modes.....	42
3.7 Instruction Set Classification	42
Summary.....	44
References.....	44
Recommended Reading.....	44
Self Assessment.....	45
 Chapter IV	 47
System Buses.....	47
Aim	47
Objectives	47
Learning outcome	47
4.1 Introduction	48
4.2 Peripheral Component Interconnect (PCI).....	49
4.3 Features of PCI	49
4.4 Concept of PCI Arbitration	50
4.4.1 The Arbitration Process.....	50
4.4.2 Clock.....	51
4.4.3 An Example of Fairness.....	51
4.4.4 Bus Parking.....	52
4.4.5 Latency	52
4.4.6 Arbitration Latency	53
4.4.7 Acquisition Latency	53
4.4.8 Initial Target Latency	53
4.4.9 Latency Timer	53
4.4.10 Bandwidth vs. Latency	53
4.5 Cache Memory Organisations.....	56

4.5.1 Fully Associative Mapping	56
4.5.2 Direct Mapping	58
4.5.3 Set-associative Mapping	59
4.5.4 Sector Mapping	60
Summary	62
References	62
Recommended Reading	62
Self Assessment	63
 Chapter V	 65
Input/Output Modules	65
Aim	65
Objectives	65
Learning outcome	65
5.1 Introduction	66
5.2 Functions of I/O Module	66
5.3 Structure of I/O Module	67
5.4 Programmed Input/ Output I/O Instructions Four types of I/O Commands	68
5.5 Interrupt Driven Input/ Output	69
5.5.1 Interrupt	70
5.5.2 Software Poll	70
5.6 Direct Memory Access (DMA)	70
Summary	72
References	72
Recommended Reading	72
Self Assessment	73
 Chapter VI	 75
Operating System Support	75
Aim	75
Objectives	75
Learning outcome	75
6.1 Introduction	76
6.2 Types of Operating System	76
6.2.1 Batch Processing Operating System	76
6.2.2 Time Sharing	76
6.2.3 Real Time Operating System (RTOS)	76
6.2.4 Multiprogramming Operating System	77
6.2.5 Multiprocessing System	78
6.2.6 Networking Operating System	78
6.2.7 Distributed operating system	78
6.2.8 Operating Systems for Embedded Devices	79
6.3 Scheduling	79
6.4 Virtual Memory	80
6.5 Memory Management	80
6.5.1 Relocation	81
6.5.2 Protection	81
6.5.3 Sharing	81
6.5.4 Logical Organisation	81
6.5.5 Physical Organisation	81
Summary	82
References	82
Recommended Reading	82
Self Assessment	83

Chapter VII	85
Central Processing Unit.....	85
Aim	85
Objectives	85
Learning outcome	85
7.1 Introduction	86
7.2 CPU Operation.....	86
7.3 Processor Organisation	87
7.3.1 General Register	87
7.3.2 Addressing Modes	87
7.4 RISC, CISC and VLIW	87
7.4.1 RISC	88
7.4.2 VLIW	88
7.4.3 CISC.....	89
7.5 Instruction Set	89
7.5.1 Code Density.....	89
7.5.2 Number of Operands.....	90
Summary.....	91
References	91
Recommended Reading	91
Self Assessment.....	92
 Chapter VIII.....	 94
Introduction to Multiprocessor System	94
Aim	94
Objectives	94
Learning outcome	94
8.1 Introduction	95
8.2 Flynn's Classification.....	95
8.3 Characteristics of Multiprocessors.....	96
8.3.1 Interprocessor Arbitration	97
8.3.2 Inter Processor Communication and Synchronisation.....	97
8.3.3 Cache Coherence	98
8.4 Parallel Processing.....	98
8.5 Types of Parallel Organisation.....	99
8.6 Pipelining	99
Summary.....	102
References	102
Recommended Reading	102
Self Assessment.....	103

List of Figures

Fig. 1.1 Positional value (weight) of each bit	2
Fig. 1.2 1's complement.....	3
Fig. 1.3 (a) Decimal to binary conversion	5
Fig. 1.3 (b) Decimal to binary conversion	5
Fig. 1.4 (a) Decimal to binary conversion	6
Fig. 1.4 (b) Decimal to binary conversion	6
Fig. 1.5 Octal number system	7
Fig. 1.6 (a) Decimal to octal conversion.....	7
Fig. 1.6 (b) Decimal to octal conversion.....	8
Fig. 1.7 Hexadecimal number showing positional values (weight) of digits.....	9
Fig. 1.8 (a) Decimal to hex conversion.....	10
Fig. 1.8 (b) Decimal to hex conversion.....	11
Fig. 2.1 Logic gates (a) Basic logic gates; (b) Compound logic gates (c) Truth tables for compound gates.....	24
Fig. 2.2 K-map	26
Fig. 2.3 Steps for construction of 2-variable K-map (a); (b); (c).....	29
Fig. 2.4 (a) Overlapped groups; (b) Rolling; (c) Don't care conditions of K-maps	29
Fig. 3.1 8085 Pin out diagram.....	35
Fig. 3.2 Architecture of 8085 Microprocessor	36
Fig. 3.3 I/O interfacing.....	37
Fig. 3.4 DIP switches	37
Fig. 3.5 Pin and logic diagram of 74244.....	38
Fig. 3.6 Interfacing of input device.....	38
Fig. 3.7 LED display	39
Fig. 3.8 Internal diagram of latch IC 74273.....	39
Fig. 3.9 Principal of interfacing an output device.....	40
Fig. 3.10 8085 Microprocessor programming model	40
Fig. 3.11 Flag register	41
Fig. 4.1 Functional diagram of a computer bus	48
Fig. 4.2 Arbitration process under PCI	50
Fig. 4.3 Timing diagram for arbitration process involving two masters.....	51
Fig. 4.4 Example of fairness in arbitration	52
Fig. 4.5 Components of bus latency.....	53
Fig. 4.6 A cache memory reference	55
Fig. 4.7 The logical organisation of a four-way set-associate cache	56
Fig. 4.8 Cache with fully associative mapping.....	57
Fig. 4.9 Fully associative mapped cache with multi-word lines 5	57
Fig. 4.10 Cache with direct mapping.....	58
Fig. 4.11 Direct mapped cache with a multi-word block	59
Fig. 4.12 Cache with set-associative mapping	60
Fig. 5.1 General I/O module	67
Fig. 5.2 Three techniques for an input of a block of data	68
Fig. 5.3 Structure of memory mapped I/O.....	69
Fig. 5.4 I/O-mapped I/O.....	69
Fig. 6.1 Multiprogramming operating system	77
Fig. 6.2 Multiprocessing system	78
Fig. 6.3 Networking Operating System	78
Fig. 6.4 Virtual memory.....	80
Fig. 8.1 SIMD	95
Fig. 8.2 MISD	96
Fig. 8.3 MIMD.....	96
Fig. 8.4 Two ways of executing N-unit jobs in a stream	100
Fig. 8.5 Four segment pipeline	100
Fig. 8.6 Space time diagram for a segment pipeline.....	101

List of Tables

Table 1.1 Counting binary numbers.....	3
Table 1.2 Binary equivalent of octal digit.....	8
Table 1.3 Binary and decimal equivalent of each hex digit.....	9
Table 1.4 Gray code	13
Table 1.5 One-bit binary addition	14
Table 2.1 Fundamental products for two inputs.....	27
Table 2.2 Fundamental products for three inputs.....	27
Table 2.3 Truth table for SOP equation.....	28
Table 2.4 Truth table for SOP equation.....	28
Table 3.1 Types of data transfer	42
Table 4.1 Bus parameters	49
Table 4.2 Bandwidth vs. latency	54
Table 8.1 Critical sections.....	98

Abbreviations

BPOS	-	Batch Processing Operating System
DMA	-	Direct Memory Access
I/O	-	Input Output
MIMD	-	Multiple Instruction Multiple Data
MISD	-	Multiple Instruction Single Data
MMU	-	Memory Management Unit
OS	-	Operating System
PIO	-	Programmed Input Output
ROM	-	Read Only Memory
RTOS	-	Real Time Operating System
SIMD	-	Single Instruction Multiple Data
SISD	-	Single Instruction Single Data

Chapter I

Number System and Codes

Aim

The aim of this chapter is to:

- introduce different types of number system
- explain binary arithmetic
- explicate different types of syntax

Objectives

The objectives of this chapter are to:

- explain conversion of digits from a number system to another
- explicate binary system and binary arithmetic
- elucidate inter-conversion of digits from one number system to another

Learning outcome

At the end of this chapter, you will be able to:

- identify decimal numbers in binary system
- define numbers from one system to another
- understand binary and decimal equivalent of each hex digit

1.1 Introduction

Number system is simply the ways to count things. Aim of any number system is to deal with certain quantities which can be measured, monitored, recorded, manipulated arithmetically, observed and utilised. Each quantity has to be represented by its value as efficiently and accurately as is necessary for any application. The numerical value of a quantity can be basically expressed in either analog (continuous) or digital (step by step) method of representation. In analog method, a quantity is expressed by another quantity which is proportional to the first. For example, the voltage output of an amplifier is measured by a voltmeter. The angular position of the needle of the voltmeter is proportional to the voltage output of the amplifier.

Yet another example is of a thermometer. The height to which the mercury rises is proportional to the temperature. In both these examples, the value of voltage and temperature can be anywhere between zero and the maximum limit. In digital method, the value of a quantity is expressed by some symbols which are called digits, and not by a quantity which is proportional to the first. In a digital watch, the time, which changes continuously, is expressed by digits which do not change continuously. It is clear from the examples that the accuracy of the value of an analog quantity generally depends upon the judgement of the observer.

Digital technology is different from analog technology. Many number systems are being used in digital technology. Most common amongst them are decimal, binary, octal, and hexadecimal systems. We are most familiar with the decimal number system, because we use it every day.

It is the base-10 or radix-10 system. Note that there is no symbol for “10” or for the base of any system. We count 1 2 3 4 5 6 7 8 9, and then insert a 0 in the first column and add a new left column, starting at 1 again. Then we count 1-9 in the first column again. (People use the base-10 system because we have 10 fingers!). Each column in our system stands for a power of 10 starting at 100.

1.2 Binary System

All computers use the binary system. The following points provides an overview of the binary system:

- In the binary number system (base of 2), there are only two digits: 0 and 1 and the place values are 2⁰, 2¹, 2², 2³ etc. Binary digits are abbreviated as bits. For example, 1101 is a binary number of 4 bits (i.e. it is a binary number containing four binary digits.)
- A binary number may have any number of bits. Consider the number 11001.01. Note the binary point (counterpart of decimal point in decimal number system) in this number.
- Each digit is known as a bit and can take only two values 0 and 1. The left most bit is the highest-order bit and represents the most significant bit (MSB) while the lowest-order bit is the least significant bit (LSB). Some useful definitions are:
 - Word is a binary number consisting of an arbitrary number of bits.
 - Nibble is a 4-bit word (one hexadecimal digit) 16 values.
 - Byte is an 8-bit word 256 values.

2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	2 ⁻¹	2 ⁻²	2 ⁻³	Positional values or weight
1	1	0	0	1	.	0	1	1
MSB				Binary Point			LSB	

Fig. 1.1 Positional value (weight) of each bit

(Source: <http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf>)

Any number can be expressed in binary form in the usual way. Table 1.1 shows expression of binary numbers.

2^3	2^2	2^1	2^0	Binary Number	Decimal Number
0	0	0	0	0000	0
0	0	0	1	0001	1
0	0	1	0	0010	2
0	0	1	1	0100	3
0	1	0	0	0101	4
0	1	0	1	0110	5
0	1	1	0	0111	6
0	1	1	1	1000	7
1	0	0	0	1001	8
1	0	0	1	1010	9
1	0	1	0	1011	10
1	0	1	1	1100	11
1	1	0	0	1101	12
1	1	0	1	1101	13
1	1	1	0	1110	14
1	1	1	1	1111	15

Table 1.1 Counting binary numbers

(Source: <http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf>)

1's complement

The 1's complement of a binary number is obtained just by changing each 0 to 1 and each 1 to 0.

Binary number	1	0	1	1	1	0	1	0
	↓	↓	↓	↓	↓	↓	↓	↓
1-complement	0	1	0	0	0	1	0	1

Fig. 1.2 1's complement

(Source: <http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf>)

2's complement

The 2's complement of a binary number is obtained adding 1 to the 1's complement of this number:

2's complement = 1's complement+1

Binary number 1 0 1 1 1 0 1 0

1's complement 0 1 0 0 0 1 0 1

$$\frac{1}{2} + 1$$

2's complement 0 1 0 0 0 1 1 0

There is a simple method to obtain the 2's complement:

- Beginning with the LSB, just write down bits as they are moving to left till the first 1, including it.
- Substitute the rest of bits by their 1's complement.

Signed numbers

In a signed number, the left most bit is the so called sign bit: 0=positive number 1=negative number.

Sign-value notation

In this notation, the left-most bit is the sign bit and the others are used to represent the absolute value notation.

1's complement

In this notation, the positive numbers have the same representation as the sign-value notation, and the negative numbers are obtained by taking the 1's complement of the positive correspondents.

2's complement

The positive numbers have the same representation as the sign-value notation, and the negative numbers are obtained by taking the 1's complement of the positive correspondents.

Positive	All	00011001 (+25)
Negative	Sign-value	10011001
1's complement		11100110
2's complement		11100111

1.2.1 Binary to Decimal Conversion

- Binary number can be converted into its decimal equivalent, by simply adding the weights of various positions in the binary number which have bit 1.

- Example 1:**

Find the decimal equivalent of the binary number $(11111)_2$

The equivalent decimal number is

$$=1X2^4+1X2^3+1X2^2+1X2^1+1X2^0$$

$$=16+8+4+2+1$$

$$= (31)_{10}$$

- To differentiate between numbers represented in different number systems, either the corresponding number system may be specified along with the number or a small subscript at the end of the number may be added signifying the number system. Example $(1000)_2$ represents a binary number and is not one thousand.

- Example 2:**

Consider the conversion of $(100011.101)_2$

$$1\ 0\ 0\ 0\ 1\ 1\ .\ 1\ 0\ 1$$

$$= 25+0+0+0+2^1+2^0+2^{-1}+0+2^{-3}$$

$$=32+2+1+0.5+0.125$$

$$=(35.625)_{10}$$

Consider the following examples.

$$1111.00 = 15$$

$$11110.0 = 30$$

$$111100.0 = 60$$

From these examples, it is clear that if the binary point is shifted towards right side, then the value of the number is doubled.

Now consider the following examples.

$$111.100 = 7.5$$

$$11.1\ 100 = 3.75$$

$$1.1\ 1100 = 1.875$$

From these examples it is clear that if the binary point is shifted towards the left side, then the value of the number is halved.

1.3 Decimal to Binary Conversion

A decimal number is converted into its binary equivalent by its repeated divisions by 2. The division is continued

till we get a quotient of 0. Then all the remainders are arranged sequentially with first remainder taking the position of LSB and the last one taking the position of MSB. Consider the conversion of 27 into its binary equivalent as follows.

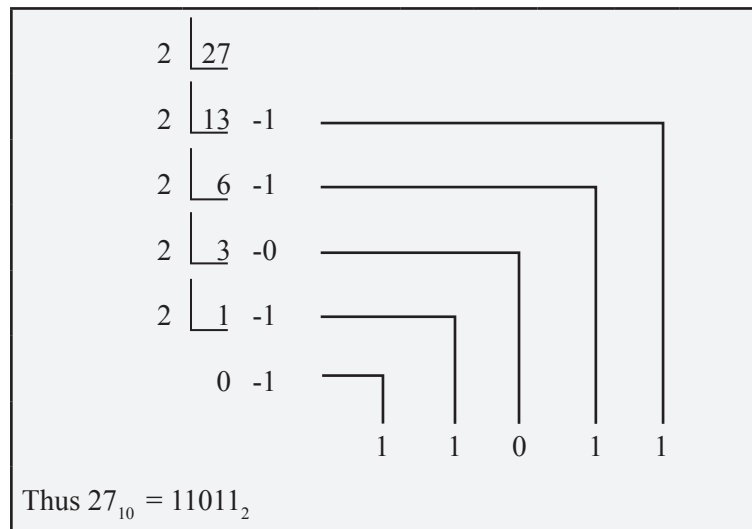


Fig. 1.3 (a) Decimal to binary conversion

(Source: <http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf>)

If the number also has some figures on the right of the decimal point, then this part of the number is to be treated separately. Multiply this part repeatedly by two. After first multiplication by 2, either 1 or 0 will appear on the left of the decimal point. Keep this 1 or 0 separately and do not multiply it by 2 subsequently. This should be followed for every multiplication. Continue multiplication by 2 till you get all 0s after the decimal point or up to the level of the accuracy desired. This will be clear from the following example. Consider the conversion of 27.62510 into its binary equivalent. We have already converted 27 into its binary equivalent which is (11011)₂. Now for the conversion of 0.625, multiply it by 2 repeatedly as follows:

Thus, $27.62510 = 11011.101$

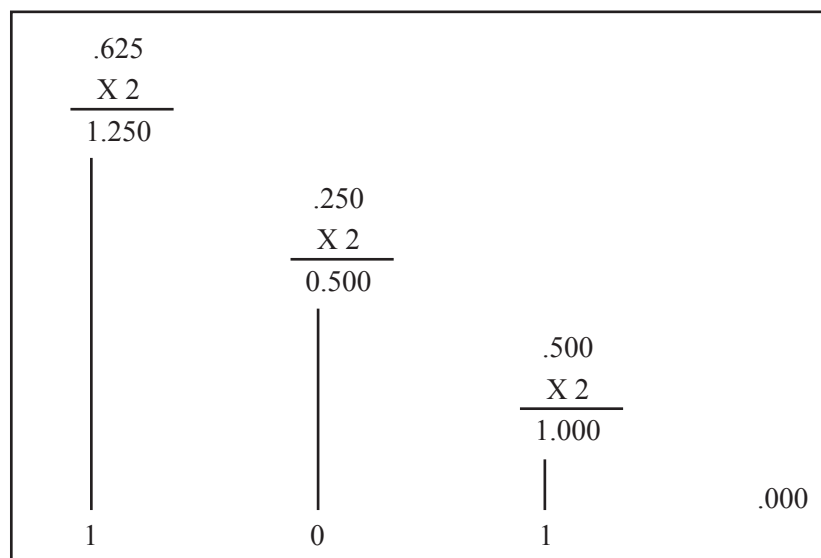
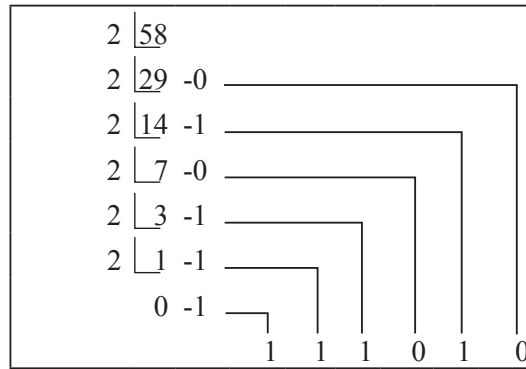


Fig. 1.3 (b) Decimal to binary conversion

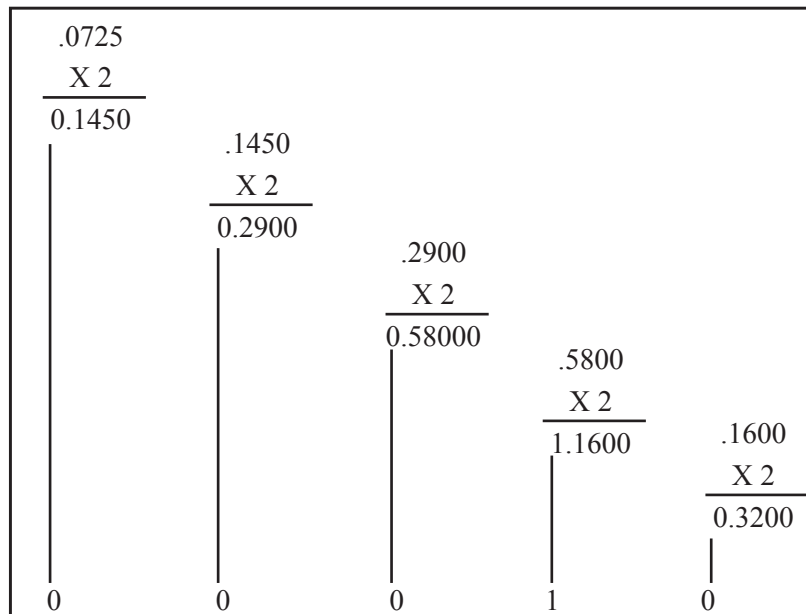
(Source: <http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf>)

Let us try another example, conversion of (58.0725)₁₀ into binary. Split this number in two parts, i.e. 58 and .0725 and convert them into binary separately as described above.

**Fig. 1.4 (a) Decimal to binary conversion**(Source: <http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf>)

Now let's look at the conversion of 0.725

Thus, $(58.0725)_{10} = 111010.00010$

**Fig. 1.4 (b) Decimal to binary conversion**(Source: <http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf>)

Representing numbers in binary is very tedious, since binary numbers often consist of a large chain of 0's and 1's. Convenient shorthand forms for representing the binary numbers are developed such as octal system and hexadecimal system. With these number systems long strings of 0's and 1's can be reduced to a manageable form. The section below gives an overview of these systems.

1.4 Octal Number System

The octal number system has base-8 that is, there are 8 digits in this system. These digits are 0, 1, 2, 3, 4, 5, 6, and 7. The weight of each octal digit is some power of 8 depending upon the position of the digit. Octal numbers showing positional values (weights) of each digit are as follows:

The conversion of .38 is as follows

Thus, $(126.38)_{10} = 176.3024$

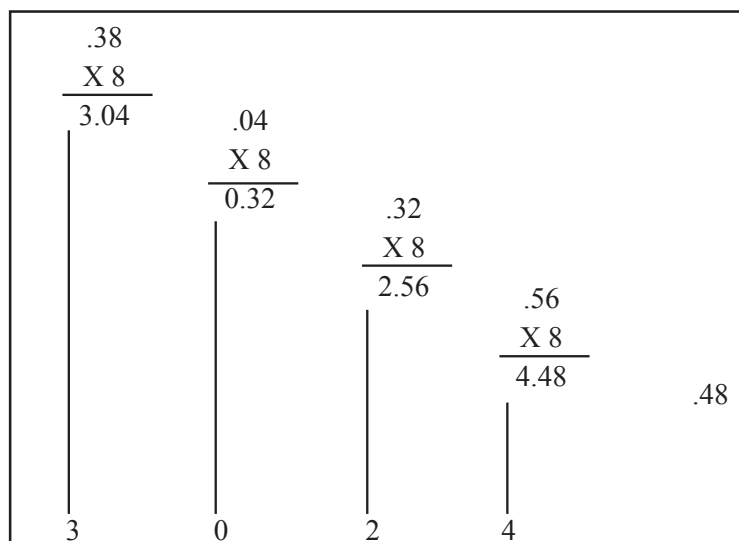


Fig. 1.6 (b) Decimal to octal conversion

(Source: <http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf>)

1.4.3 Octal to Binary Conversion

In the octal number system the highest octal digit i.e.7 can be expressed as a 3-bit binary number. Therefore, all the octal digits have to be represented by a 3-bit binary number. The binary equivalent of each octal digit is shown in Table 1.2. The main advantage of the octal number system is the easiness with which any octal number can be converted into its binary equivalent.

Octal digit	3-bit binary equivalent
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Table 1.2 Binary equivalent of octal digit

(Source: <http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf>)

Using this conversion of octal digit into 3-bit binary number, any octal number can be converted into its binary equivalent by simply replacing each octal digit by a 3-bit binary number. For example, conversion of 567, into its binary equivalent is:

$567 = 101110111$

1.5.1 Hex to Decimal Conversion

Hex to decimal conversion is done in the same way as in the cases of binary and octal to decimal conversions. A hex number is converted into its equivalent decimal number by summing the products of the weights of each digit and their values. This is clear from the example of conversion of 514.AF16 into its decimal equivalent.

$$\begin{aligned} 514.AF_{16} &= 5 \times 16^2 + 1 \times 16^1 + 4 \times 16^0 + 10 \times 16^{-1} + 15 \times 16^{-2} \\ &= 1280 + 16 + 4 + 0.625 + 0.0586 \\ &= (1300.6836)_{10} \end{aligned}$$

1.5.2 Decimal to Hex Conversion

A decimal number is converted into hex number in the same way as a decimal number is converted into its equivalent binary and octal numbers. The part of the number on the left of the decimal point is to be divided repeatedly by 16 and the part on the right of the decimal point is to be repeatedly multiplied by 16. This will be clear from the examples of conversion of $(579.26)_{10}$ into hex equivalent. Split the number into two parts, 579 and .26.

Thus, $(579)_{10} = (2443)_{16}$

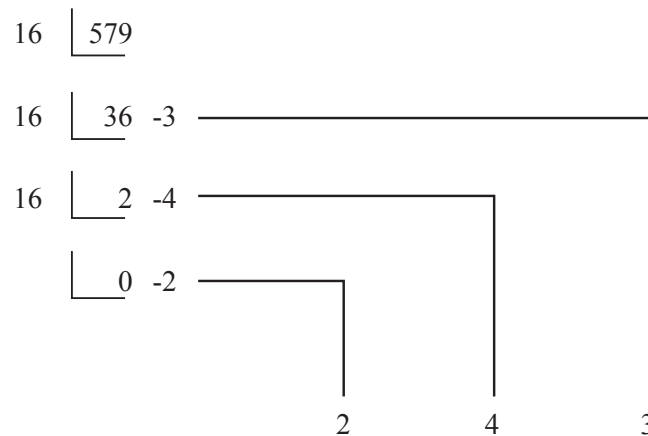


Fig. 1.8 (a) Decimal to hex conversion

(Source: <http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf>)

Now .26 is converted into hex number as follows:

$$\begin{array}{r}
 .26 \\
 \times 16 \\
 \hline
 4.16 \\
 \downarrow \\
 4
 \end{array}
 \qquad
 \begin{array}{r}
 .16 \\
 \times 16 \\
 \hline
 2.56 \\
 \downarrow \\
 2
 \end{array}
 \qquad
 \begin{array}{r}
 .56 \\
 \times 16 \\
 \hline
 8.96 \\
 \downarrow \\
 8
 \end{array}
 \qquad
 \begin{array}{r}
 .96 \\
 \times 16 \\
 \hline
 15.36 \\
 \downarrow \\
 F
 \end{array}$$

Thus $579.26_{10} = 243.428_{16}$

Fig. 1.8 (b) Decimal to hex conversion

(Source: <http://www.egyankosh.ac.in/bitstream/123456789/10991/1/Unit-10.pdf>)

1.5.3 Hex to Binary Conversion

As in octal number system, a hex number is converted into its binary equivalent by replacing each hex digit by its equivalent 4-bit binary number. This is clear from the following example:

$$\begin{aligned}
 (BA6)_{16} &= B & A & 6 \\
 &= 1011 & 1010 & 010 \\
 &= (101110100110)_2
 \end{aligned}$$

1.5.4 Binary to Hex Conversion

By a process that is reverse of the process described in the above section, a binary number can be converted into its hex equivalent. Starting from the LSB side, group the binary number bits into groups of four bits. If towards the MSB side, the number of bits is less than four, then add zeros on the left of the MSB so that the group of four is complete. Replace each group by its equivalent hex digit. This is clear from the following example:

$$\begin{aligned}
 (1001101110)_2 &= 0010 & 0110 & 1110 \\
 &= 2 & 6 & E \\
 &= (26E)_{16}
 \end{aligned}$$

1.5.5 Hex to Octal Conversion

Each digit of the hex number is first converted into its equivalent four bit binary number. Then the bits of the equivalent binary number are grouped into groups of three bits. Then each group is replaced by its equivalent octal digit to get the octal number.

For example:

$$\begin{aligned}
 (5AF)_{16} &= 0101 \quad 1010 \quad 1111 \\
 &= 010110101111 \\
 &= 010 \quad 110 \quad 101 \quad 111 \\
 &= 2 \quad 6 \quad 5 \quad 7 \\
 &= (2567)_8
 \end{aligned}$$

1.5.6 Octal to Hex Conversion

For octal to hex conversion, just reverse the process described in the section above.

This is clear from the following example:

$$\begin{aligned}
 (5457)_8 &= 101 \quad 100 \quad 101 \quad 111 \\
 &= 1011 \quad 0010 \quad 1111 \\
 &= B \quad 2 \quad F \\
 &= (B2F)_{16}
 \end{aligned}$$

This method can also be applied to hex to decimal and decimal to hex conversions. For example, consider the conversion of 3C16, into its decimal equivalent:

$$\begin{aligned}
 3C16 &= 0011 \quad 1100 \\
 &= 1111002
 \end{aligned}$$

Check the conversion.

$$3C16 = 3 \times 16^1 + C \times 16^0$$

$$= 3 \times 16^1 + 12 \times 16^0$$

$$= 48 + 12$$

$$= (60)_{10}$$

$$1111002 = 2^5 + 2^4 + 2^3 + 2^2$$

$$= 32 + 16 + 8 + 4$$

$$= (60)_{10}$$

$$\text{Thus, } 3C16 = (111100)_2 = 6010$$

1.6 Codes

We had an overview of binary, octal and hexadecimal number system. For any number system with n base B and digits N_0 (LSB), N_1 N_2 N_{10} (MSB), the decimal equivalent N_{10} is given by

$$N_{10} = N_m \times B^m + \dots N_3 \times B^3 + N_2 \times B^2 + N_1 \times B^1 + N_0 B^0$$

When numbers, letters or words are represented by a specific group of symbols, it is said that the number, letter or word is being encoded. The group of symbols is called as the code.

Few codes will be discussed in the following sections.

1.6.1 BCD Code

In BCD (BCD stands Binary coded decimal) code, each digit of a decimal number is converted in to its binary equivalent. The largest decimal digit is 9; therefore the largest binary equivalent is 1001. This is illustrated as follows

$$\begin{aligned}
 95110 &= 1001 \quad 0101 \quad 0001 \\
 &= (100101010001)_{BCD}
 \end{aligned}$$

1.6.2 ASCII Code

The word ASCII is run acronym of American Standard Code for Information Interchange. This is the alphanumeric code most widely used in computers. The alphanumeric code is one that represents alphabets, numerical numbers, punctuation marks and other special characters recognised by a computer. The ASCII code is a 7-bit code representing 26 English alphabets, 0 through 9 digits, punctuation marks, etc. A 7-bit code has $2^7 = 128$ possible code groups which are quite sufficient.

1.6.3 Code Gray

Gray Code is a form of binary that uses a different method of incrementing from one number to the next. With Gray Code, only one bit changes state from one position to another. This feature allows a system designer to perform some error checking (i.e., if more than one bit changes, the data must be incorrect).

Decimal	Binary	Gray	Decimal	Binary	Gray
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	12	1100	1110
4	0100	0110	13	1101	1010
5	0101	0111	14	1110	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

Table 1.4 Gray code

(Source: <http://dpnc.unige.ch/tp/elect/doc/07-Digital.pdf>)

1.6.4 Excess 3

Excess-3 binary-coded decimal (XS-3), also called biased representation or Excess-N, is a numeral system used on some older computers that uses a pre-specified number N as a biasing value. It is a way to represent values with a balanced number of positive and negative numbers. In XS-3, numbers are represented as decimal digits, and each digit is represented by four bits as the BCD value plus 3 (the "excess" amount)

The smallest binary number represents the smallest value. (i.e. 0 – Excess Value)

The greatest binary number represents the largest value. (i.e. $2_{N+1} - \text{Excess Value} - 1$)

The primary advantage of XS-3 coding over BCD coding is that a decimal number can be 9's complemented (for subtraction) as easily as a binary number can be 1's complemented; just invert all bits. In addition, when the sum of two XS-3 digits is greater than 9, the carry bit of a four bit adder will be set high. This works because, when adding two numbers that are greater or equal to zero, an "excess" value of six results in the sum. Since a four bit integer can only hold values 0 to 15, an excess of six means that any sum over nine will overflow.

1.7 Binary Arithmetic

Majority of arithmetic performed by computers is binary arithmetic, that is, arithmetic on base two numbers. Decimal and floating-point numbers, also used in computer arithmetic, depend on binary representations, and an understanding of binary arithmetic is necessary in order to understand either one.

Computers perform arithmetic on fixed-size numbers. The arithmetic of fixed size numbers is called finite-precision arithmetic. The rules for finite-precision arithmetic are different from the rules of ordinary arithmetic. The sizes of numbers which can be arithmetic operands are determined when the architecture of the computer is designed. Common sizes for integer arithmetic are eight, 16, 32, and recently 64 bits. It is possible for the programmer to perform arithmetic on larger numbers or on sizes which are not directly implemented in the architecture. However, this is usually so painful that the programmer picks the most appropriate size implemented by the architecture. This puts a burden on the computer architect to select appropriate sizes for integers, and on the programmer to be aware of the limitations of the size he has chosen and on finite-precision arithmetic in general.

We are considering binary arithmetic in the context of building digital logic circuits to perform arithmetic. Not only do we have to deal with the fact of finite precision arithmetic, we must consider the complexity of the digital logic. When there is more than one way of performing an operation we choose the method which results in the simplest circuit.

1.7.1 Addition

Addition of binary numbers can be carried out in a similar way by the column method. But before this, view four simple cases. In the decimal number system, $3 + 6 = 9$ symbolizes the combination of 3 with 6 to get a total of 9.

View the four simple cases.

- Case 1: When nothing is combined with nothing, we get nothing. The binary representation of this is $0 + 0 = 0$.
- Case 2: When nothing is combined with 1, we get 1. Using binary numbers to denote this gives $0 + 1 = 1$.
- Case 3: Combining 1 with nothing gives 1. The binary equivalent of this is $1 + 0 = 1$.
- Case 4: When we combine 1 with 1, the result is 2. Using binary numbers, we symbolize $1 + 1 = 10$.

The last result is sometimes confusing because of our long time association with decimal numbers. But it is correct and makes sense because we are using binary numbers. Binary number 10 stands for '1','0' and not for 10 (ten). To summarize our results for binary addition,

- $0+0 = 0$
- $0+1 = 1$
- $1+0 = 1$
- $1+1=10$
- One can also express the rules of binary addition with a truth table. This is important because there are techniques for designing electronic circuits that compute functions expressed by truth tables. The fact that we can express the rules of binary addition as a truth table implies that we can design a circuit which will perform addition on binary numbers, and that turns out to be the case.
- We only need to write the rules for one column of bits; we start at the right and apply the rules to each column in succession until the final sum is formed. Call the bits of the addend and augend A and B, and the carry in from the previous column C_i . Call the sum S and the carry out C_o .
- The truth table for one-bit binary addition looks like this:

A	B	C_i	S	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 1.5 One-bit binary addition

This says if all three input bits are zero, both S and Co will be zero. If any one of the bits is one and the other two are zero, S will be one and Co will be zero. If two bits are 1's, S will be zero and Ci will be one. Only if all three bits are 1's, both S and Co will be 1's.

1.7.2 Addition of Signed Numbers

Binary addition of 2's complement signed numbers can be performed using the same rules given above for unsigned addition. If there is a carry out of the sign bit, it is ignored. It is possible for the result of an addition to be too large to fit in the available space. The answer will be truncated, and will be incorrect. This is the overflow condition discussed above. There are two rules for determining whether overflow has occurred:

- If two numbers of opposite signs are added, overflow cannot occur.
- If two numbers of the same sign are added, overflow has occurred if and only if the result is of the opposite sign.

1.7.3 Subtraction

Addition has the property of being commutative, that is, $a+b = b+a$. This is not true of subtraction. $5 - 3$ is not the same as $3 - 5$. For this reason, we must be careful of the order of the operands when subtracting. We call the first operand, the number which is being diminished, the minuend; the second operand, the amount to be subtracted from the minuend, is the subtrahend. The result is called the difference.

```
51 minuend
- 22 subtrahend
 29 difference
```

It is possible to perform binary subtraction using the same process we use for decimal subtraction, namely subtracting individual digits and borrowing from the left. This process quickly becomes cumbersome as you borrow across successive zeroes in the minuend. Further, it doesn't lend itself well to automation. Jacobowitz describes the "carry" method of subtraction which some of you may have learned in elementary school, where a one borrowed in the minuend is "paid back" by adding to the subtrahend digit to the left. This means that one need look no more than one column to the left when subtracting.

Subtraction can thus be performed a column at a time with a carry to the left, analogous to addition. This is a process which can be automated, but we are left with difficulties when the subtrahend is larger than the minuend or when either operand is, signed. Since we can form the complement of a binary number easily and can add signed numbers easily, the obvious answer to the problem of subtraction is to take the 2's complement of the subtrahend, then add it to the minuend. That is $51 - 22 = 51 + (-22)$.

Not only does this approach remove many of the complications of subtraction by the usual method, but it also means special circuits to perform subtraction need not be built. All that is needed is a circuit which can form the bitwise complement of a number and an adder.

1.7.4 Multiplication

A simplistic way to perform multiplication is by repeated addition. In the example below, we could add 42 to the product register 27 times. In fact, some early computers performed multiplication this way. However, one of our goals is speed, and we can do much better using the familiar methods we have learned for multiplying decimal numbers. Recall that the multiplicand is multiplied by each digit of the multiplier to form a partial product, and then the partial products are added to form the total product. Each partial product is shifted left to align on the right with its multiplier digit.

```
42 multiplicand
x 27 multiplier
-----
294 first partial product (42 X 7)
 84 second partial product (42 X 2)
-----
1134 total product
```

Binary multiplication of unsigned (or positive 2's complement) numbers works exactly the same way, but is even easier because the digits of the multiplier are all either zero or one. That means the partial products are either zero or a copy of the multiplicand, shifted left appropriately. Consider the following binary multiplication:

0111	multiplicand	
x 0101	multiplier	
0111	first partial product	(0111 X 1)
0000	second partial product	(0111 X 0)
0111	third partial product	(0111 X 1)
0000	fourth partial products	(0111 X 0)
0100011	total product	

Notice that no true multiplication is necessary in forming the partial products. The fundamental operations required are shifting and addition. This means we can multiply unsigned or positive integers using only shifters and adders.

1.7.5 Division

As with the other arithmetic operations, division is based on the paper-and-pencil approach we learned for decimal arithmetic. We will show an algorithm for unsigned long division that is essentially similar to the decimal algorithm we learned in grade school. Let us divide 0110101 (5310) by 0101 (510). Beginning at the left of the dividend, we move to the right one digit at a time until we have identified a portion of the dividend which is greater than or equal to the divisor. At this point, a one is placed in the quotient; all digits of the quotient to the left are assumed to be zero. The divisor is copied below the partial dividend and subtracted to produce a partial remainder as shown below.

$$\begin{array}{r}
 \text{divisor } 0101 \quad \overline{) 0110101} \quad \begin{array}{l} \text{quotient} \\ \text{dividend} \end{array} \\
 \underline{0101} \\
 1 \quad \text{partial remainder}
 \end{array}$$

Now digits from the dividend are “brought down” into the partial remainder until the partial remainder is again greater than or equal to the divisor. Zeroes are placed in the quotient until the partial remainder is greater than or equal to the divisor, and then a one is placed in the quotient, as shown below.

$$\begin{array}{r}
 \overline{) 0110101} \\
 \underline{0101} \downarrow \downarrow \\
 110
 \end{array}$$

The divisor is copied below the partial remainder and subtracted from it to form a new partial remainder. The process is repeated until all bits of the dividend have been used. The quotient is complete and the result of the last subtraction is the remainder.

$$\begin{array}{r}
 \overline{) 0110101} \\
 \underline{0101} \downarrow \\
 110 \downarrow \\
 \underline{0101} \downarrow \\
 11
 \end{array}$$

This completes the division. The quotient is $(1010)_2$ (1010) and the remainder is $(11)_2$ (310), which is the expected result. This algorithm works only for unsigned numbers, but it is possible to extend it to 2's complement numbers.

As with the other algorithms, it can be implemented using only shifting, complementation, and addition. Digital computers can perform arithmetic operations using only binary numbers. And hence the above section of binary arithmetic is the basic step of digital electronics.

Summary

- There are mainly four number systems mainly binary, octal, decimal and hexadecimal which have 2, 8, 10 and 16 digits respectively. But it is the ease in applications that decides which kind of number system should be defined and used. Every computer uses two or more of the above mentioned number systems simultaneously.
- The binary number system has only two digits; 0 and 1. A binary digit is called bit. A binary number can be converted into its equivalent octal, decimal and hex numbers as described in the text. And octal, decimal and hex numbers can be converted into equivalent binary numbers.
- The octal number system has 8 digits; 0 through 7. An octal number can be converted into its equivalent binary, decimal and hex numbers and vice versa as described in the text.
- The hex number system has 16 digits; 0 through 9, A through F. As in the other systems, the hex numbers can be converted as described in the text into their binary, octal and decimal equivalents and vice versa.
- It is possible to arrange sets of binary digits to represent numbers, letters of the alphabet or other information by using a given code. Some of the important codes are BCD and ASCII codes. In the BCD code, each decimal digit is replaced by its 4-bit binary equivalent.
- Conversion of BCD code into its decimal equivalent and vice versa is quite easy. Therefore, it is quite often used in computers.
- The ASCII code is the most widely used alphanumeric code. It is a 7-bit binary number and has $2^7 = 128$ possible 7-bit binary numbers which are quite sufficient to describe the capital and small letters of the alphabet, digits, punctuation marks and other symbols.
- The process of dividing a binary number is once again the same as followed in the decimal system.

References

- Subramanyam, M.V. & Bhatia, B., 2008. *Basic Digital Electronics*, Laxmi Publications, Ltd.
- Holdsworth, B. & Woods, C., 2002. *Digital Logic Design*, 4th ed. Newnes.
- *UNIT 1 NUMBER SYSTEM AND BINARY CODES* [Pdf] Available at: <http://www.b-u.ac.in/sde_book/bca_fund.pdf> [Accessed 29 May 2013].
- *NUMBER SYSTEMS AND CODES* [Pdf] Available at: <<http://www.inf.fu-berlin.de/lehre/WS00/19504-V/Chapter1.pdf>> [Accessed 29 May 2013].
- *Binary Number System - One's Complement, Two's Complement, Conversion* [Video online] Available at: <<http://www.youtube.com/watch?v=INGWoFHLoA>> [Accessed 29 May 2013].
- Prof. Kleitz., 2011. *Digital Electronics: Textbook Preface* [Video online] Available at: <<http://www.youtube.com/watch?v=nM25ejvGn0Y&list=PLE7E5C88AA6AEA0A2>> [Accessed 29 May 2013].

Recommended Reading

- Rafiquzzaman, M., 2005. *Fundamentals of Digital Logic and Microcomputer Design*, 5th ed. John Wiley & Sons.
- Das, S., 2010. *A Complete Guide to Computer Fundamentals*, Laxmi Publications, Ltd.
- Kumar, A. A., 2010. *Switching Theory And Logic Design*, PHI Learning Pvt. Ltd..

Self Assessment

1. Aim of any number system is to deal with certain quantities which can be measured, monitored, _____, manipulated arithmetically, observed and utilised.
 - a. recorded
 - b. stored
 - c. read
 - d. used
2. Which of the following is not a common number system?
 - a. Decimal system
 - b. Tetra decimal system
 - c. Binary system
 - d. Octal systems
3. What are binary digits abbreviated as?
 - a. Bytes
 - b. Giga bytes
 - c. Bits
 - d. Nibble
4. The 2's complement of a binary number is obtained adding 1 to the _____ of this number.
 - a. complement
 - b. end
 - c. beginning
 - d. 1's complement
5. Binary number can be converted into its decimal equivalent by simply adding _____ of various positions in the binary number which has bit 1.
 - a. weights
 - b. number
 - c. ten
 - d. two
6. A decimal number is converted into its binary equivalent by its repeated _____ by 2.
 - a. addition
 - b. division
 - c. multiplication
 - d. subtraction
7. Advantage of the octal number system is the ease with which any octal number can be converted into its _____ equivalent.
 - a. decimal
 - b. hex
 - c. binary
 - d. tetra

8. Which digits are excluded from the octal number system?
 - a. 1 and 2
 - b. 0 and 1
 - c. 7 and 8
 - d. 8 and 9

9. What should be added to last group of 3 bits if the MSB side does not have 3 bits while converting numbers from binary to octal?
 - a. Zero
 - b. One
 - c. Two
 - d. Nine

10. In subtraction number which is being diminished is called the minuend; the amount to be subtracted from the minuend is the called the _____.
 - a. difference
 - b. subtrahend
 - c. subtraction
 - d. minus

Chapter II

Logic Gates and Boolean Algebra

Aim

The aim of this chapter is to:

- introduce the concept of logic gates
- explain Boolean algebra
- explicate logic minimisation techniques

Objectives

The objectives of this chapter are to:

- elucidate basic logic gates
- explicate compound logic gates
- explain laws and theorems of Boolean algebra

Learning outcome

At the end of this chapter, you will be able to:

- identify AND, NOT, OR, XOR, XNOR and NOR gates
- understand Boolean identities for basic and compound gates
- describe K-maps and Quine McCluskey techniques of logic minimisation

2.1 Introduction

Digital computers understand the language of 1s and 0s. This number system is also called as binary number system (bi means two). Note that the operation of digital circuits can be described in two corresponding voltage levels. The more positive level is denoted by high ($H = 1$) and the other is denoted by low ($L = 0$). And the logical operations are represented by true (T) and false (F). For instance, $H = 1 = T$ and $L = 0 = F$ is a positive logic whereas $H = 0 = F$ and $L = 1 = T$ is a negative logic.

A digital circuit having one or more input signals but only one output signal is called "a gate". A gate, which implements Boolean algebraic equations, is called "Logic gates". The basic building blocks of digital electronics are logic gates which perform simple binary logic functions (AND, OR, NOT, etc.). From these devices, more complex circuits can be constructed to do arithmetic, that act as memory elements.

2.2 Logic Gates

Features of logic gates are as follows:

- The flow of digital signals is controlled by transistors in various configurations depending on the logic family. For most purposes we can imagine that the logic gates are composed of ideal switches with just two states: OPEN and CLOSED. The state of a switch is controlled by a digital signal. The switch remains closed so long as a logical (1) signal is applied. A logical (0) control signal keeps it open.
- Logic signals interact by means of gates. The three fundamental gates AND, OR, and NOT, are named after the three fundamental operations of logic that they carry out. The AND and OR gates each have two inputs and one output. The output state is determined by the states of the two inputs.
- The function of each gate is defined by a truth table, which specifies the output state for each possible combination of input states. The output values of the truth tables can be understood in terms of two switches. If the switches are in series, you get the AND function. Parallel switches perform the OR operation. A bubble after a gate or at an input indicates NOT.
- The three compound gates NAND, NOR and XOR can be made from AND, OR, and NOT. NAND means an AND gate followed by a NOT, while NOR means an OR gate followed by a NOT. The EXCLUSIVE-OR (XOR) is similar to OR but it has a LO output if both inputs are HI, so you can think of it as one OR the other but NOT both. NAND and NOR are more common than AND and OR because with the help of DeMorgan's theorems they can be used to simplify complex circuits.
- When several gates are combined to perform a complex logical operation, a good design uses as few as possible. Boolean algebra, the mathematics of two valued variables, is the theoretical tool used to simplify complex logical expressions.

2.2.1 NOT Gate

The NOT gate has a single input and a single output and its symbol and truth table (truth table contains a table of all possible input values and their corresponding outputs values). It performs the operation of inversion, i.e., the output of a NOT gate takes a 1 (high) state if the input takes the 0 (low) state and vice-versa. A circuit, which performs a logic negation, is called a NOT circuit or inverter since it inverts the output with respect to the input.

2.2.2 OR Gate

The OR gate has two or more inputs and a single output. The output of an OR gate is in 1 state if any of the inputs is in the 1 state.

2.2.3 AND Gate

The AND gate has two or more inputs and a single output. The output of an AND gate is in 1 state if all inputs are in the 1 state or the output will be zero if any of the inputs is zero.

2.2.4 NAND Gate

- The Negated AND, NOT AND or NAND gate is the opposite of the digital AND gate, and behaves in a manner that corresponds to the opposite of AND gate, as shown in the truth table. A LOW output results only if both the inputs to the gate are HIGH. If one or both inputs are LOW, a HIGH output results. The NAND gate is a universal gate in the sense that any Boolean function can be implemented by NAND gates.
- Digital systems employing certain logic circuits take advantage of NAND's functional completeness. In complicated logical expressions, normally written in terms of other logic functions such as AND, OR, and NOT, writing these in terms of NAND saves on cost, because implementing such circuits using NAND gate yields a more compact result than the alternatives.
- NAND gates can also be made with more than two inputs, yielding an output of LOW if all of the inputs are HIGH, and an output of HIGH if any of the inputs is LOW.

2.2.5 NOR Gate


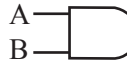
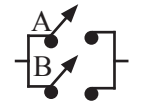
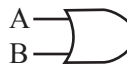
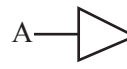
A contraction of NOT and OR, a NOR gate is one of the basic digital logic gates. With it, it is possible to build any of the other basic logic gates and thus, nearly any type of digital circuit.

2.2.6 XOR Gate

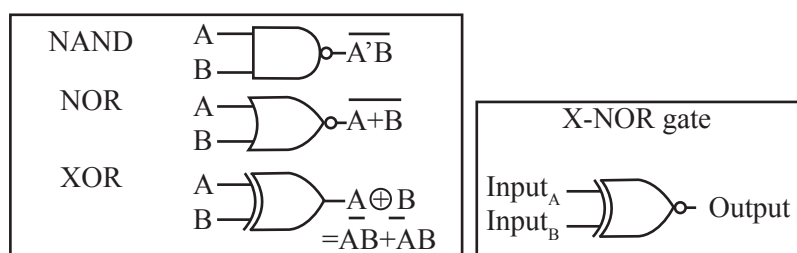
Abbreviated X-OR, is a unique and very useful type of gate. As shown in the truth table, any time both of its inputs are held in the same state (both low or both high), its output will be low. If one of the inputs is pulled high while the other is held low, then its output will be high.

2.2.7 XNOR

XNOR is a digital logic gate whose function is the inverse of the Exclusive OR (XOR) gate. The two-input version implements logical equality, behaving according to the truth table. A high output results if both of the inputs to the gate are the same. If one but not both inputs are high, a low output results.

Operation	Switches	Condition that circuit is closed	Boolean Notation	Symbol	Truth Table															
AND	 Series	(A AND B is closed)	$A \bullet B$ or AB	 A-B	<table><tr><th>A</th><th>B</th><th>A B</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	A B	0	0	0	0	1	0	1	0	0	1	1	1
A	B	A B																		
0	0	0																		
0	1	0																		
1	0	0																		
1	1	1																		
OR	 Paralle	(A OR B is closed)	$A + B$	 A+B	<table><tr><th>A</th><th>B</th><th>A+B</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	A+B	0	0	0	0	1	1	1	0	1	1	1	1
A	B	A+B																		
0	0	0																		
0	1	1																		
1	0	1																		
1	1	1																		
NOT (same as invert)	Diifferent kind of switch	1 means open 0 means closed	NOT $A = \bar{A}$	 A- \bar{A}	<table><tr><th>A</th><th>\bar{A}</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	\bar{A}	0	1	1	0									
A	\bar{A}																			
0	1																			
1	0																			

(a)



(b)

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

(c) NAND

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

NOR

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

XOR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

XNOR

Fig. 2.1 Logic gates (a) Basic logic gates; (b) Compound logic gates (c) Truth tables for compound gates
 (Source: <http://www.beam-wiki.org/wiki/NOR>; <http://www.pdf-finder.com/Digital-Electronics-I:-Logic,-Flip-Flops,-and-Clocks.html>)

2.3 Boolean Algebra

Boolean algebra (or Boolean logic) is a logical calculus of truth values, developed by George Boole in the 1840s. It resembles the algebra of real numbers, but with the numeric operations of multiplication xy , addition $x + y$, and negation $\neg x$ replaced by the respective logical operations of conjunction xoy , disjunction $x \vee y$, and negation $\neg x$.

2.3.1 Fundamental Laws

We imagine a logical variable, A , that takes on the values 0 or 1. If $A = 0$ then $\bar{A} = 1$ and if $A = 1$ then $\bar{A} = 0$. Here are some obvious identities using the AND, OR and NOT operations.

Looking at these identities you can see why the 'plus' symbol was chosen for OR and 'times' was chosen for AND.

OR AND NOT

$$A + 0 = A \quad A \cdot 0 = 0 \quad A + A = 1$$

$$A + 1 = 1 \quad A \cdot 1 = A \quad A \cdot A = A$$

$$A + A = A \quad A \cdot A = A \quad A = A$$

$$A + \bar{A} = 1 \quad A \cdot \bar{A} = 0$$

Equality

Two Boolean expressions are equal if and only if their truth tables are identical.

Associative laws

$$(A + B) + C = A + (B + C)$$

$$(AB)C = A(BC)$$

Distributive laws

$$A(B + C) = AB + AC$$

Related identities

$$(A + \bar{A}) = 1$$

$$(A + \bar{A}) = A + \bar{A}$$

$$(A + B) \cdot (A + C) = A + BC$$

2.3.2 DeMorgan's Theorems

DeMorgan's first Theorem says that complement of sum equals the product of complements.

The LHS of the equation describes a NOR (NOT-OR) gate and the RHS of the equation describes an AND gate

$$\overline{A + B} = \bar{A} \bar{B}$$

with inverted inputs and both the equations have the same truth table.

DeMorgan's second Theorem says that complement of product equals the sum of complements.

The LHS of the equation describes a NAND (NOT-AND) gate and the RHS of the equation describes an OR gate with inverted inputs and both the equations have the same truth table.

$$\overline{AB} = \overline{A} + \overline{B}$$

2.3.3 Boolean Identities

Boolean Identities for NOT operation

The NOT operation of a Boolean variable A is denoted by its complement \bar{A} . Following are the Boolean identities pertaining to NOT operation.

$$\bar{\bar{A}} = A$$

It implies that the double complement of a Boolean variable is the variable itself.

For $A = 0$, $\bar{A} = 1$, $A = 0$ and for $A = 1$, $\bar{A} = 0$, $A = 1$ is true.

$$\bar{A} + A = 1$$

It means that a variable ORed with its complement always equals 1. If $A = 0$, $\bar{A} = 1$ and $A + \bar{A} = 1$ and when $A = 1$, $\bar{A} = 0$ and $A + \bar{A} = 1$ is correct.

$$A \bar{A} = 0$$

It means that a variable ANDed with its complement always equals 0. For two possible values of A, $0.1 = 0$ when $A = 0$, $1.0 = 0$ when $A = 1$ is true.

$$A + \bar{A} B = A + B$$

Proof: $A + \bar{A} B = A(B + 1) + \bar{A} B = AB + A + \bar{A} B = (A + \bar{A}) B + A = B + A = A + B$

Boolean identities for OR operation

The OR operation in the Boolean algebra is denoted by (+). Following are the Boolean identities for the OR operation.

Commutative Law

$$A + B = B + A$$

It implies that the input A and B of the OR gate can be interchanged without changing the output Y. It can be justified from the truth table of two-input OR gate. For $A = B$, it is obvious that it doesn't matter when we interchange A and B. When $A = 0$ and $B = 1$, if we interchange A and B, then it will become the case of $A = 1$ and $B = 0$ and for both these case the output is 1. Hence it doesn't matter to the output if we interchange A and B inputs.

Associative Law

$$A + B + C = (A + B) + C = A + (B + C)$$

It means that the order of combining the input variables has no effect on the output variables. This can be verified from the truth table for three-input OR gate.

$$A + A = A$$

It means that any variable ORed with itself equals the variable. We can justify this Boolean identity by substituting the two possible values of A. For $A = 0$, $0 + 0 = 0$ and for $A = 1$, $1 + 1 = 1$ is true (refer to truth table for two-input OR gate).

$$A + 1 = 1$$

If one input of the OR gate is high the output is high no matter what is the other input. For $A = 0$, $0 + 1 = 1$ and for $A = 1$, $1 + 1 = 1$ is true.

$$A + 0 = A$$

It means a Boolean variable ORed with 0 equals the variable. For $A = 0$, $0 + 0 = 0$ and $A = 1$, $1 + 0 = 1$ is true.

Boolean Identities for AND Operation

The AND operation is denoted by (.) in the Boolean algebra. Following are the Boolean identities pertaining to AND operation.

Commutative Law

$$A \cdot B = B \cdot A$$

It implies that the inputs of the AND gate can be interchanged without changing the output Y which can be justified from the truth table of two-input AND gate. Note that (.) is suppressed many times and we can write the Commutative law as $A \cdot B = B \cdot A$.

Associative Law

$$A \cdot B \cdot C = (A \cdot B) \cdot C = A \cdot (B \cdot C)$$

It means that the order of combining the input variables has no effect on the output variables. This can be verified from the truth table for three-input AND gate.

$$A \cdot A = A$$

It means that any variable ANDed with itself equals the variable. For $A = 0$, $0 \cdot 0 = 0$ and for $A = 1$, $1 \cdot 1 = 1$ is true (refer to truth table for two-input AND gate).

$$A \cdot 1 = A$$

If one input of the AND gate is high the output is equal to the input. For $A = 0$, $0 \cdot 1 = 0$ and for $A = 1$, $1 \cdot 1 = 1$ is true.

$$A \cdot 0 = 0$$

If one input of the AND gate is low the output is low irrespective of the other input. For $A = 0$, $0 \cdot 0 = 0$ and for $A = 1$, $1 \cdot 0 = 0$ is true.

Distributive law

$$A \cdot (B + C) = A \cdot B + A \cdot C$$

We can write down the truth tables for LHS and RHS of the Boolean equation and verify they are same.

2.4 Logic Minimisation

Given a truth table, it is always possible to write down a correct logic expression simply by forming an OR of the ANDs of all input variables for which the output is true. However, for an arbitrary truth table such a procedure could produce a very lengthy and cumbersome expression which might be needlessly inefficient to implement with gates.

There are several methods for simplification of Boolean logic expressions. The process is usually called "logic minimization" and the goal is to form a result which is efficient. Two methods we will discuss are algebraic minimization and Karnaugh maps. For more complex expressions the Quine-McKluskey method may be appropriate. Karnaugh maps are also limited to problems with up to 4 binary inputs.

		AB			
CD		00	01	11	10
				X	
00				X	
01		X	1	X	1
11		1	1	X	X
10			1	X	X

Fig. 2.2 K-map

(Source: <http://www.ee.surrey.ac.uk/Projects/Labview/minimisation/isf.html>)

2.5 Karnaugh Maps (K-Maps)

The Karnaugh map uses a rectangle divided into rows and columns in such a way that any product term in the expression to be simplified can be represented as the intersection of a row and a column. The rows and columns are labelled with each term in the expression and its complement. The labels must be arranged so that each horizontal or vertical move changes the state of one and only one variable. A Karnaugh map is visual display of minterms required for a sum-of-product solution.

2.5.1 Sum-of-Products Equations and Logic Circuits (SOP)

There are four possible ways to AND two input signals that are in the complemented and un-complemented form (B, A, AB) also called as fundamental products. Table 2.1 lists each of the fundamental product producing high outputs.

A	B	Fundamental Product	Minterms
0	0	$\overline{A} \overline{B}$	m_0
0	1	B	m_1
1	0	$A \overline{B}$	m_2
1	1	AB	m_3

Table 2.1 Fundamental products for two inputs

(Source: <http://www.egyankosh.ac.in/bitstream/123456789/13358/1/Unit-8.pdf>)

For instance, $\overline{A} \overline{B}$ is high when A and B are low. The fundamental products B, $A \overline{B}$ and AB are also represented by minterms m_0 , m_1 , m_2 and m_3 , where the suffix i of m_i comes from the decimal equivalent of the binary number (Each individual term in standard SOP form is called as minterm). For 3 inputs A, B and C, there are 23 minterms, m_0 , m_1 , m_2 , m_3 , m_4 , m_5 , m_6 and m_7 as listed in table 2.2.

A	B	C	Fundamental Product	Minterms
0	0	0	$\overline{A} \overline{B} \overline{C}$	m_0
0	0	1	$\overline{A} \overline{B} C$	m_1
0	1	0	$\overline{A} B \overline{C}$	m_2
0	1	1	$\overline{A} B C$	m_3
1	0	0	$A \overline{B} \overline{C}$	m_4
1	0	1	$A \overline{B} C$	m_5
1	1	0	$A B \overline{C}$	m_6
1	1	1	ABC	m_7

Table 2.2 Fundamental products for three inputs

(Source: <http://www.egyankosh.ac.in/bitstream/123456789/13358/1/Unit-8.pdf>)

For example, when $A = 1$, $B = 1$, $C = 0$, the fundamental product results a high output for the case $Y = AB = 11 = 1$. SOP equation is a function of Boolean variables, which states the fundamental product terms or minterms, which will give a high output for the given inputs. For instance, the output Y is a function of three Boolean variables A, B and C whose minterms are listed which will give a high outputs.

$$Y = F(A, B, C) = \sum m(1, 2, 3, 4) = + + + +$$

Let us try to find the truth table from the given SOP equation. List all the possible values of A, B and C in the order of increasing minterm index i.e., In the truth table, place 1s whose minterms are listed in the SOP equation and at the other places draw 0s as in Table 2.3.

A	B	C	Minterms	Y
0	0	0	m_0	0
0	0	1	m_1	$1(\bar{A} \bar{B} C)$
0	1	0	m_2	$1(\bar{A} B \bar{C})$
0	1	1	m_3	$1(\bar{A} B C)$
1	0	0	m_4	$1(A \bar{B} \bar{C})$
1	0	1	m_5	0
1	1	0	m_6	0
1	1	1	m_7	0

Table 2.3 Truth table for SOP equation

(Source: <http://www.egyankosh.ac.in/bitstream/123456789/13358/1/Unit-8.pdf>)

2.5.2 Product of Sums (POS)

The Product of Sums form represents an expression as a product of maxterms. Each individual term in standard POS form is called as maxterm.

$F(X, Y \dots) = \text{Product}(b_k + M_k)$, where b_k is 0 or 1 and M_k is a maxterm.

2.5.3 Drawing Karnaugh Maps

Consider the following example:

$$Y = F(A, B) = \sum m(1, 2, 3) = B + +$$

The truth table for this SOP equation is shown in Table 2.4

A	B	Y	Minterms
0	0	0	m_0
0	1	1	m_1
1	0	1	m_2
1	1	1	m_3

Table 2.4 Truth table for SOP equation

(Source: <http://www.egyankosh.ac.in/bitstream/123456789/13358/1/Unit-8.pdf>)

Following are the steps required to produce the K-map from the truth table:

Begin by drawing Fig. 2.3 (a). Write down the value of suffix of minterms or the equivalent decimal numbers as shown in Fig. 2.3 (b).

From the truth table of Table 2.4, write down the values of Y for the corresponding minterms. The values of Y for the minterms which are there in the SOP equation is 1, for other minterms it is 0.

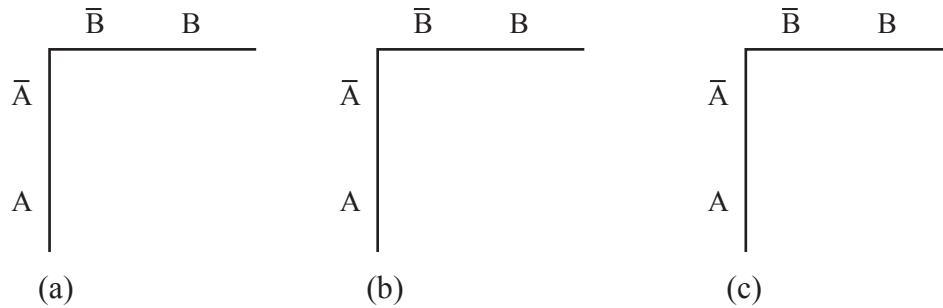


Fig. 2.3 Steps for construction of 2-variable K-map (a); (b); (c)

(Source: <http://www.egyankosh.ac.in/bitstream/123456789/13358/1/Unit-8.pdf>)

2.5.4 Don't Care Conditions

In some digital systems, some inputs are never supplied, thereby no output is visible. We call such cases "don't care conditions" and they are represented by X in the truth table as depicted in Fig. 2.4(c). Whenever we see an X in the truth table while encircling the 1s in the K-map to form the largest group, we can assume Xs as 1s. After it has been included in all the groups, disregard the Xs in the truth tables by assuming them as 0s. As for Fig. 2.4(c), the SOP equation after Karnaugh simplifications are $Y = B$.

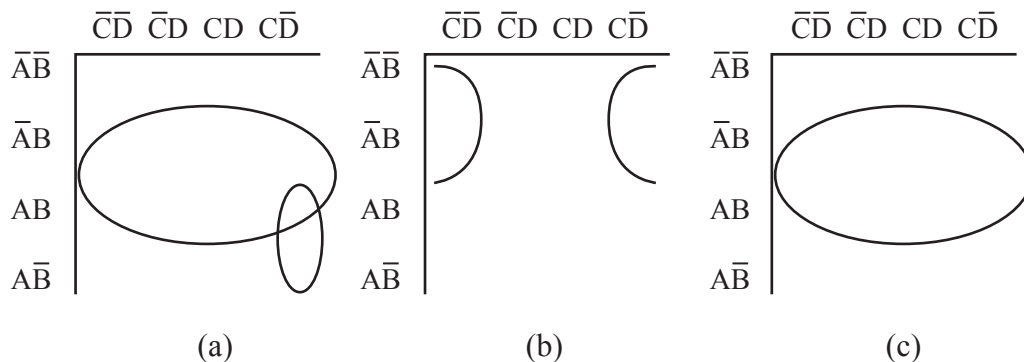


Fig. 2.4 (a) Overlapped groups; (b) Rolling; (c) Don't care conditions of K-maps

(Source: <http://www.egyankosh.ac.in/bitstream/123456789/13358/1/Unit-8.pdf>)

2.6 Quine - McCluskey Method

The Quine - McCluskey algorithm (or the method of prime implicants) is a method used for minimization of Boolean functions which was developed by W.V. Quine and Edward J. McCluskey. It is functionally identical to Karnaugh mapping, but the tabular form makes it more efficient for use in computer algorithms, and it also gives a deterministic way to check that the minimal form of a Boolean function has been reached. It is sometimes referred to as the tabulation method.

The method involves two steps:

- Finding all prime implicants of the function.
- Use those prime implicants in a prime implicant chart to find the essential prime implicants of the function, as well as other prime implicants that are necessary to cover the function.

Thus, logic expressions can be represented in one of the standard forms: SOP or POS and then simplified using K-maps or Quine- McCluskey method.

Summary

- A digital circuit having one or more input signals but only one output signal is called a gate. A gate, which implements Boolean algebraic equations, is called Logic gates.
- The basic building blocks of digital electronics are logic gates which perform simple binary logic functions (AND, OR, NOT, etc.). From these devices, more complex circuits can be constructed to do arithmetic, act as memory elements.
- The flow of digital signals is controlled by transistors in various configurations depending on the logic family. For most purposes we can imagine that the logic gates are composed of ideal switches with just two states: OPEN and CLOSED. The state of a switch is controlled by a digital signal. The switch remains closed so long as a logical (1) signal is applied. A logical (0) control signal keeps it open.
- Logic signals interact by means of gates. The three fundamental gates AND, OR, and NOT, are named after the three fundamental operations of logic that they carry out. The AND and OR gates each have two inputs and one output. The output state is determined by the states of the two inputs.
- The three compound gates NAND, NOR and XOR can be made from AND, OR, and NOT. NAND means an AND gate followed by a NOT, while NOR means an OR gate followed by a NOT. The EXCLUSIVE-OR (XOR)
- Boolean algebra (or Boolean logic) is a logical calculus of truth values, developed by George Boole in the 1840s. and resembles the algebra of real numbers.
- There are several methods for simplification of Boolean logic expressions. The process is usually called "logic minimization" and the goal is to form a result which is efficient. Two methods we will discuss are algebraic minimization and Karnaugh maps. For more complex expressions the Quine-McKluskey method may be appropriate.
- The Karnaugh map uses a rectangle divided into rows and columns in such a way that any product term in the expression to be simplified can be represented as the intersection of a row and a column. The rows and columns are labelled with each term in the expression and its complement. The labels must be arranged so that each horizontal or vertical move changes the state of one and only one variable.
- The Quine - McCluskey algorithm (or the method of prime implicants) is a method used for minimization of Boolean functions which was developed by W.V. Quine and Edward J. McCluskey.

References

- Patrick, R. D., Fardo, W. S. & Chandra, V., 2008. *Electronic Digital System Fundamentals*, The Fairmont Press, Inc.
- Rajaraman, V. & Radhakrishnan, T., 2006. *DIGITAL LOGIC AND COMPUTER ORGANIZATION*, PHI Learning Pvt. Ltd.
- *Chapter 3 Boolean Algebra and Logic Gates* [Pdf] Available at: <<http://larc.ee.nthu.edu.tw/~cww/n/228/03.pdf>> [Accessed 29 May 2013].
- Ryu, J. H., *Boolean Algebra and Logic Gates* [Pdf] Available at: <http://robot.kut.ac.kr/download/dd/chapter2_boolean_algebra.pdf> [Accessed 29 May 2013].
- *An Introduction to Logic Gates* [Video online] Available at: <<http://www.youtube.com/watch?v=95kv5BF2Z9E>> [Accessed 29 May 2013].
- *Boolean Algebra Part 1* [Video online] Available at: <<http://www.youtube.com/watch?v=lEgyiudC-nk>> [Accessed 29 May 2013].

Recommended Reading

- Bhattacharya, J., *Rudiments of Computer Science*, Academic Publishers.
- Tocci, J. R., 1980. *Digital Systems: Principles and Applications*, Pearson Education India.
- Balabanian, N. & Carlson, B., 2007. *DIGITAL LOGIC DESIGN PRINCIPLES*, John Wiley & Sons.

Self Assessment

1. A digital circuit having one or more input signals but only one output signal is called a _____.
 - a. gate
 - b. gateway
 - c. circuit
 - d. route
2. The flow of digital signals is controlled by _____ in various configurations depending on the logic family.
 - a. resistors
 - b. transistors
 - c. capacitors
 - d. circuits
3. Which of the following statements is true?
 - a. The circuit remains closed so long as a logical (1) signal is applied.
 - b. The switch remains open so long as a logical (1) signal is applied.
 - c. The switch remains closed so long as a logical (1) signal is applied.
 - d. The switch remains closed so long as a logical (0) signal is applied.
4. The function of each gate is defined by _____.
 - a. input
 - b. output
 - c. signal
 - d. truth table
5. Which is the theoretical tool used to simplify complex logical expressions?
 - a. Boolean algebra
 - b. Logic
 - c. Summation
 - d. Trigonometry
6. Which is the digital logic gate whose function is the inverse of the Exclusive OR?
 - a. XOR
 - b. XNOR
 - c. NOR
 - d. OR
7. Two Boolean expressions are equal if and only if their _____ are identical.
 - a. values
 - b. figures
 - c. truth tables
 - d. diagrams

8. According to which theorem/law, the complement of product equals the sum of complements?
 - a. DeMorgan's first theorem
 - b. Associative law
 - c. Distributive law
 - d. DeMorgan's second theorem

9. _____ of Boolean logic an expression is usually called logic minimization.
 - a. Simplification
 - b. Verification
 - c. Addition
 - d. Division

10. The Product of Sums form represents an expression as a product of _____.
 - a. minterms
 - b. maxterms
 - c. related terms
 - d. terms

Chapter III

Introduction to 8085 Microprocessor

Aim

The aim of this chapter is to:

- explain 8085 microprocessor
- elucidate features and architecture of 8085 microprocessor
- introduce 8085 programming model and addressing modes

Objectives

The objectives of this chapter are to:

- determine 8085 microprocessor
- explain interfacing of input device
- elucidate instruction set classification

Learning outcome

At the end of this chapter, you will be able to:

- identify features and architecture of 8085 microprocessor
- describe I/O devices and their interfacing
- understand instruction set classification

3.1 Introduction

The 8085A is an 8-bit microprocessed suitable for a wide range of applications. It is a single-chip, NMOS device implemented with approximately 6200 transistors on 164X222 mil chip contained in a 40-pin dual-in-line package.

3.2 Features

Following are the features of 8085A:

- It is an 8-bit microprocessor i.e., it can accept, process, or provide 8-bit data simultaneously.
- It operates on a single +5V power supply connected to V_{cc} ; power supply ground is connected to V_{ss} .
- It operates on clock cycle with 50% duty cycle.
- It has no chip clock generator. This internal clock generator requires tuned circuit like LC, RC or crystal. The internal clock generator divides oscillator frequency by 2 and generates clock signal, which can be used for synchronising external devices.
- It can operate with a 3MHz clock frequency. The 8085A-2 version can operate at the maximum frequency of 5MHz.
- It has 16 address lines, hence it can access (2^{16}) 64 Kbytes of memory.
- It provides 8 bit I/O address to access (2^8) 256 I/O ports.
- In 8085, the lower 8-bit address bus (A_0-A_7) and data bus (D_0-D_7) are multiplexed to reduce number of external pins. But due to this, external hardware (latch) is required to separate address lines and data lines.
- It supports 74 instructions with following addressing modes:
 - Immediate
 - Register
 - Direct
 - Indirect
 - Implied
- The Arithmetic Logic Unit (ALU) of 8085 performs:
 - 8 bit binary addition with or without carry
 - 16 bit binary addition
 - 2 digit BCD addition
 - 8-bit binary subtraction with or without borrow
 - 8-bit logical AND, OR, EX-OR, complement (NOT) and bit shift operations
- It has 8-bit accumulator, flag register, instruction register, six 8-bit general purpose registers (B, C, D, E, H and L) and two 16-bit registers (SP and PC). Getting the operand from the general purpose registers is faster than from memory. Hence skilled programmers always prefer general purpose registers to store program variables than memory.
- It provides four hardware interrupts: TRAP, RST 7.5, RST 5.5 and INTR.
- It has serial I/O control which allows serial communication.
- It provides control signals ($\overline{IO/\overline{MM}}$, \overline{RDRD} , \overline{WRWR}) to control the bus cycles, hence, external bus controller is not required.
- The external hardware (another microprocessor or equivalent master) can detect which machine cycle microprocessor is executing using status signals ($\overline{IO/\overline{MM}}$, S_0 , S_1). This feature is very useful when more than one processors are using common system resources (memory and I/O devices).
- It has a mechanism by which it is possible to increase its interrupt handling capacity.
- The 8085 has an ability to share system bus with Direct Memory Access controller. This feature allows transferring large amount of data form I/O device to memory or from memory to I/O device with high speeds.
- It can be used to implement three chip microcomputer with supporting I/O devices like IC 8155 and IC 8355.

X ₁	<input type="checkbox"/>	1	40	<input type="checkbox"/>	V _{cc}
X ₂	<input type="checkbox"/>	2	39	<input type="checkbox"/>	HOLD
RESET OUT	<input type="checkbox"/>	3	38	<input type="checkbox"/>	HLDA
SOD	<input type="checkbox"/>	4	37	<input type="checkbox"/>	CLK (OUT)
SID	<input type="checkbox"/>	5	36	<input type="checkbox"/>	RESET IN
TRAP	<input type="checkbox"/>	6	35	<input type="checkbox"/>	READY
RST 7.5	<input type="checkbox"/>	7	34	<input type="checkbox"/>	IO / M
RST 6.5	<input type="checkbox"/>	8	33	<input type="checkbox"/>	S ₁
RST 5.5	<input type="checkbox"/>	9	32	<input type="checkbox"/>	RD
INTR	<input type="checkbox"/>	10	31	<input type="checkbox"/>	WR
INTA	<input type="checkbox"/>	11	30	<input type="checkbox"/>	ALE
AD ₀	<input type="checkbox"/>	12	29	<input type="checkbox"/>	S ₀
AD ₁	<input type="checkbox"/>	13	28	<input type="checkbox"/>	A ₁₅
AD ₂	<input type="checkbox"/>	14	27	<input type="checkbox"/>	A ₁₄
AD ₃	<input type="checkbox"/>	15	26	<input type="checkbox"/>	A ₁₃
AD ₄	<input type="checkbox"/>	16	25	<input type="checkbox"/>	A ₁₂
AD ₅	<input type="checkbox"/>	17	24	<input type="checkbox"/>	A ₁₁
AD ₆	<input type="checkbox"/>	18	23	<input type="checkbox"/>	A ₁₀
AD ₇	<input type="checkbox"/>	19	22	<input type="checkbox"/>	A ₉
V _{ss}	<input type="checkbox"/>	20	21	<input type="checkbox"/>	A ₈

Fig. 3.1 8085 Pin out diagram

3.3 Architecture of 8085 Microprocessor

8085 consists of various units and each unit performs its own functions. The various units of a microprocessor are listed below:

- Accumulator
- Arithmetic and logic unit
- General purpose register
- Program counter
- Stack pointer
- Temporary register
- Flags
- Instruction register and decoder
- Timing and control unit
- Interrupt control
- Serial Input/output control
- Address buffer and address-data buffer
- Address bus and data bus

3.4 I/O and Memory Interfacing

Input and output (IO) devices are used to enter data to the microprocessor and to take out data from the microprocessor from external world. General examples of input and output devices are used to take output from the microprocessor.

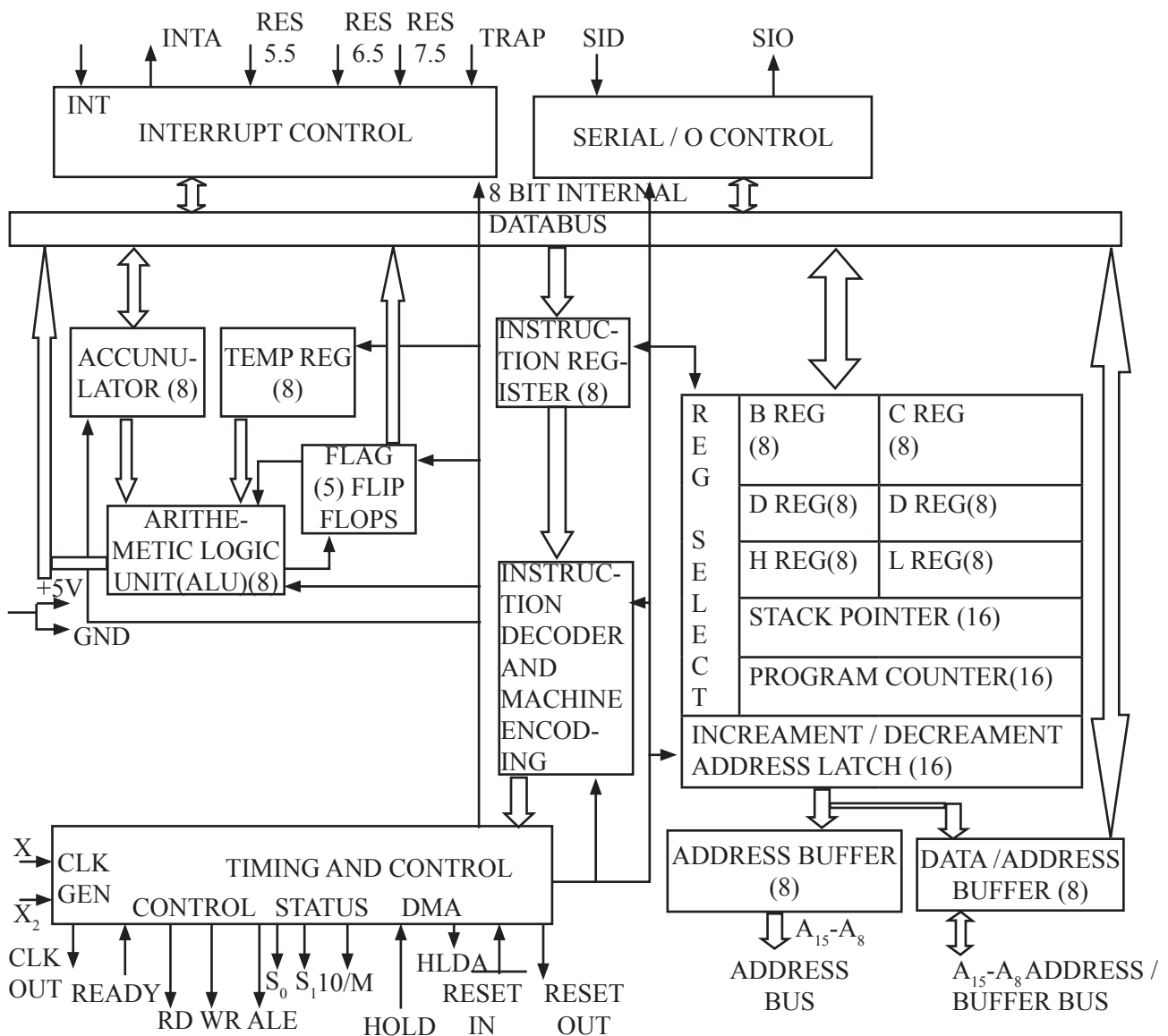


Fig. 3.2 Architecture of 8085 Microprocessor

Memory is basically a data storage device. For any microprocessor system, we require memories to store monitor program, to store user program and to store data. So memory is an essential component in microprocessor system which will allow the user to store program and data. Memory consists of thousands of memory cells. Each memory cell is capable of storing 1-bit.

Memories are broadly classified as primary and secondary memory. Primary memory is a part of memory which can be directly accessed by the microprocessor, whereas secondary memories are those memories which are portable such as hard disk, floppy disc, and compact disk (CD).

3.4.1 I/O Devices and their Interfacing

Input/Output, or IO, refers to the communication between an information processing system and the outside world—possibly a human or another information processing system. Inputs are the signals or data received by the system and outputs are the signals or data sent from it. The term can also be used as part of an action; “perform I/O” means to perform an input or output operation. I/O devices are used by a person to communicate with a computer. For instance, keyboard and mouse are considered input devices of a computer, while monitors and printers are considered output devices of a computer. Devices for communication between computers, such as modems and network cards, typically serve for both input and output. Input and output devices are connected to the microprocessor through some interfacing devices as shown in Fig. 3.3

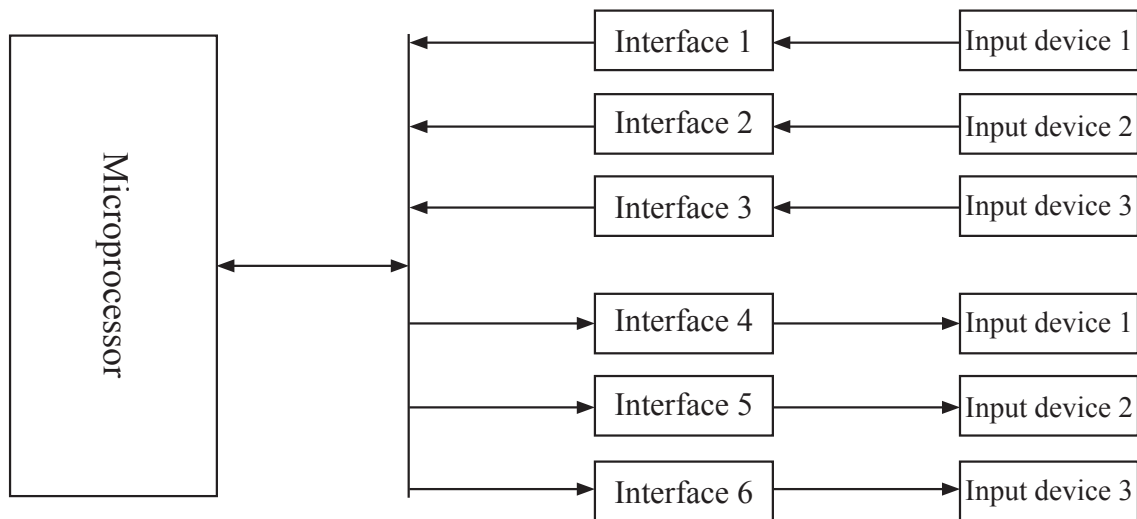


Fig. 3.3 I/O interfacing

3.4.2 IO Addressing

The IO devices in 8085 may be interface with 8085 in two different ways i.e. IO mapped IO and memory mapped IO. These two methods of IO interfacing give rise to two different IO address spaces.

3.4.3 Interfacing of Input Device

Input devices are used to input data to the microprocessor. A common example of an input device is the keyboard as shown in fig. 3.4. Such an IO device is not directly connected to the microprocessor but connected through an interface i.e. a buffer gate IC.

The DIP switches shown in Fig. 3.4 cannot be directly connected to the data lines of the microprocessor. These are connected to the microprocessor through the buffer gate IC. One such chip is the 74LS244 chip shown in fig. 3.5 along with its internal logic diagram.

This buffer chip performs two important functions: firstly, it increases the driving capabilities of the buses and secondly, it is used to select the input device through its OE signal.

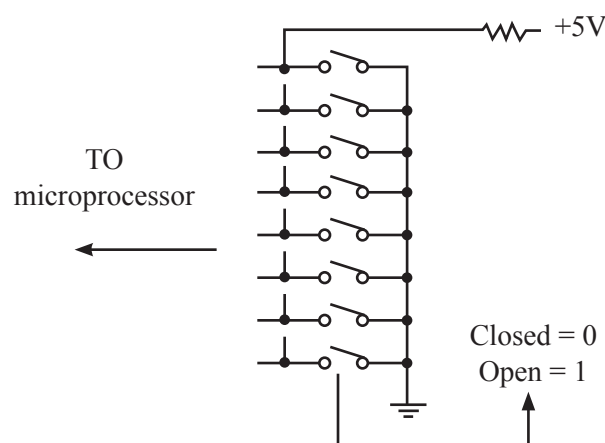


Fig. 3.4 DIP switches

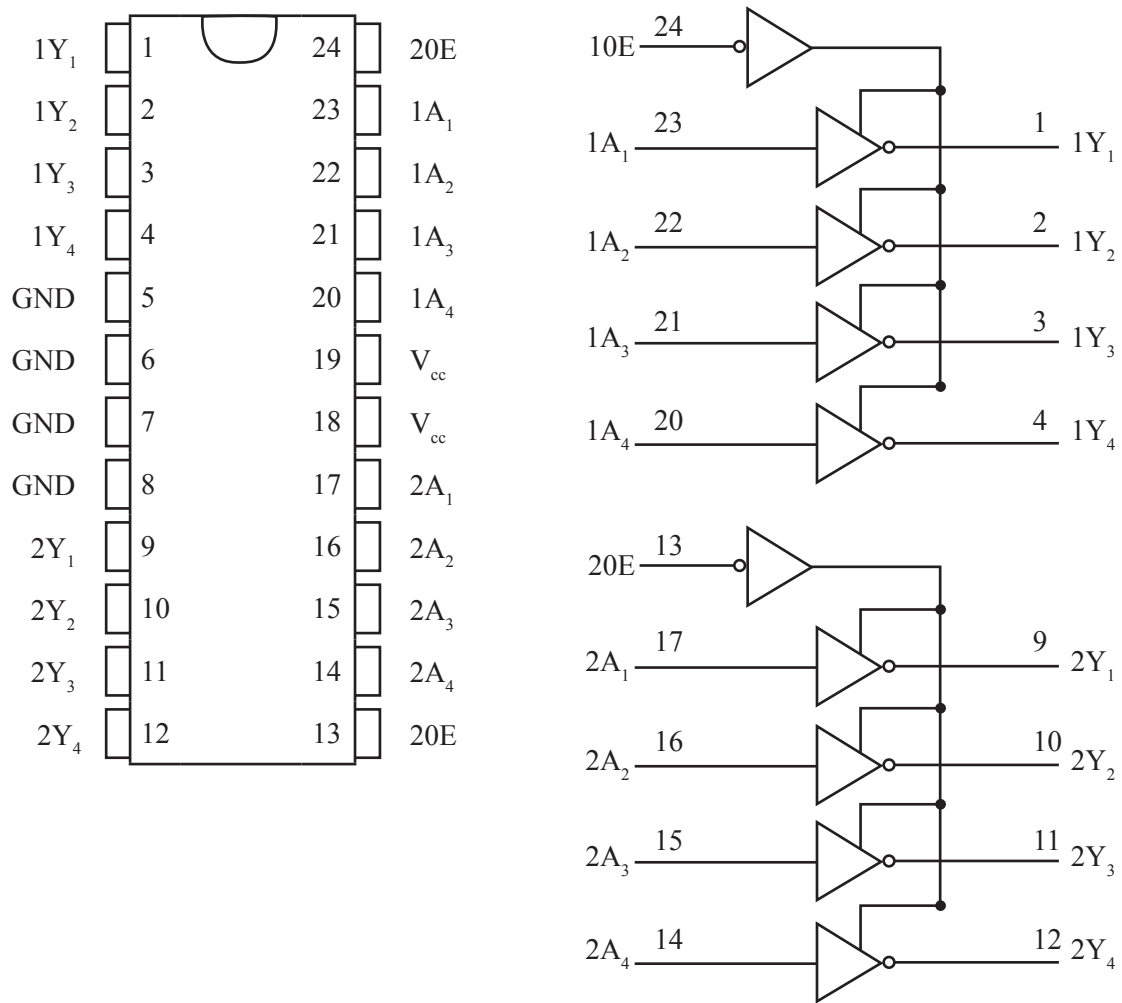


Fig. 3.5 Pin and logic diagram of 74244

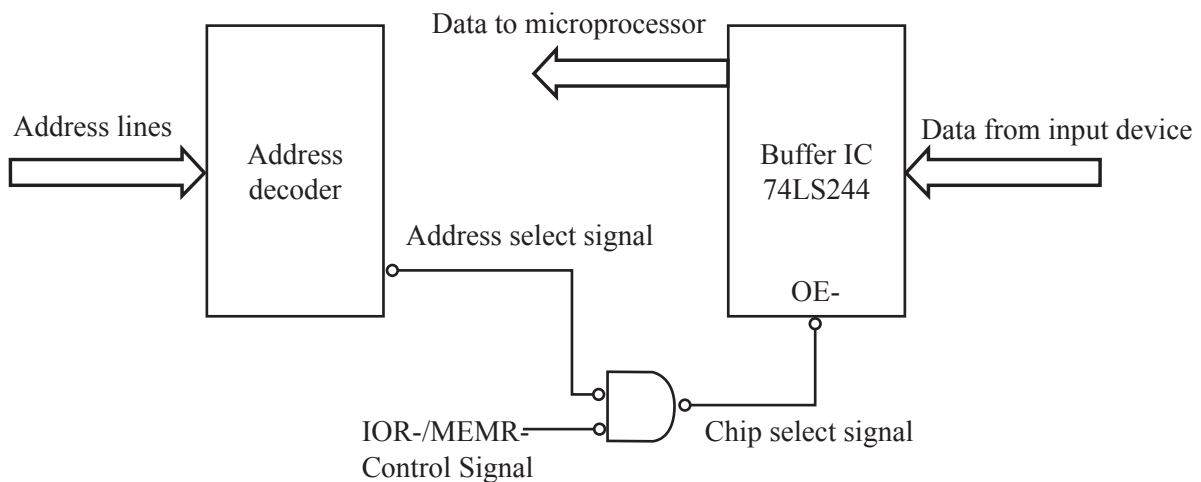


Fig. 3.6 Interfacing of input device

As shown in Fig. 3.6, the address lines are decoded, either using a NAND (or AND) gate or using a decoder, to generate an address select signal. The address lines may be A0-A15 in case of memory mapped IO scheme. Then the address select signal is NANDed with the control signal IOR (in case of IO mapped IO). The output of the NAND gate will generate the chip select signal, which is applied to the OE signal of the 74244 buffer chip. When the OE signal goes low it will enable the buffer and microprocessor will start reading the switch position and load it into the accumulator.

3.4.4 Interfacing Output Data

Output devices are used to take data from the microprocessor. A common example of an output device is the display devices such as LED or LCD or seven segment displays. Fig. 3.6 shows a simple seven segment display. Such an IO device is not directly connected to the microprocessor, but connected through a latch IC. Fig. 3.7 shows the pin and internal diagram of latch 74273. The eight latches of the DM74ALS373 are transparent D-type latches. While the enable is taken low the output will be latched at the level of the data that was set up.

A buffered output control input can be used to place the eight outputs in either a normal logic state (high or low logic levels) or a high-impedance state. In the high-impedance state the outputs neither load nor drive the bus lines significantly. The output control does not affect the internal operation of the latches. That is, the old data can be retained or new data can be entered even while the outputs are off.

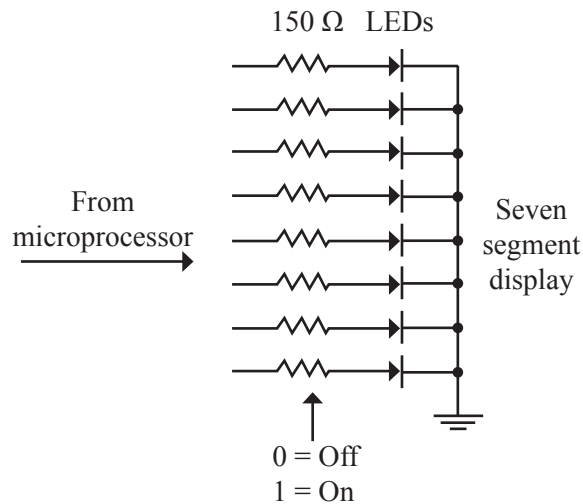


Fig. 3.7 LED display

Fig. 3.9 gives the basic principal of interfacing an output device. To generate an address, select signal. As shown in fig. 3.8, the address lines are decoded, either using NAND (or AND) gate or using a decoder. The address lines may be A0-A7, in case of IO mapped IO or may be A0-A15 in case of memory mapped IO scheme. Then the address select signal is Nanded with control signal IOW (in case of IO mapped IO) or with MEMW (in case of memory mapped IO).

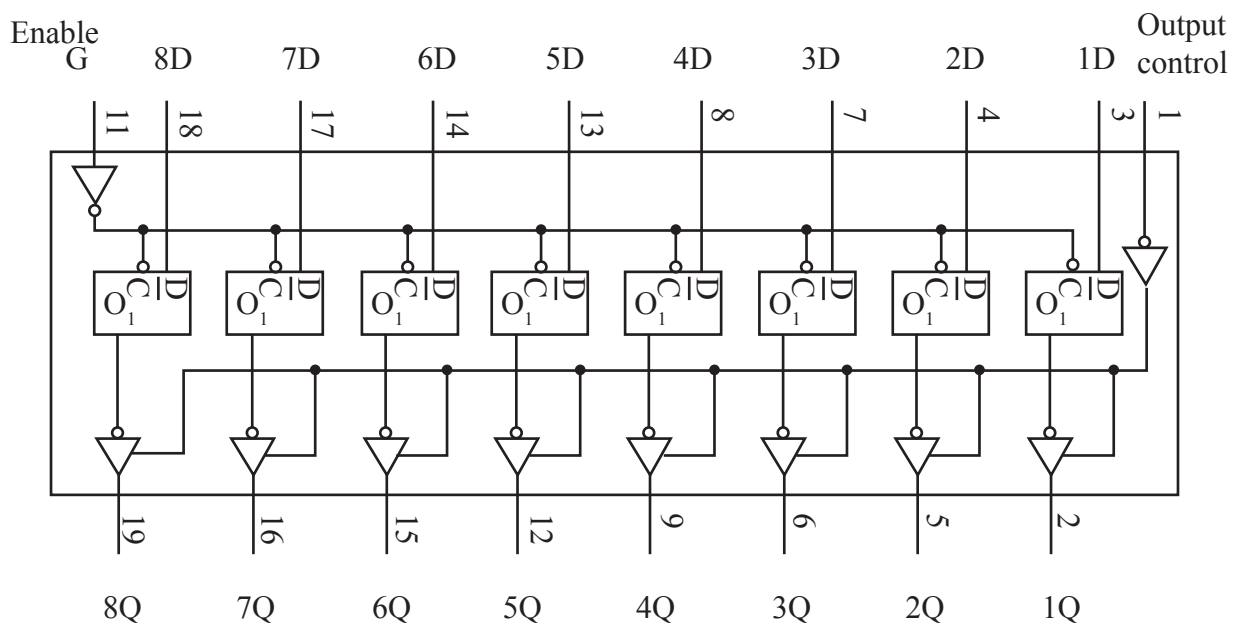


Fig. 3.8 Internal diagram of latch IC 74273

The output of NAND gate will generate the chip select signal, which is applied to the output control signal OE of the 74273 latch chip. The OE signal enabled the tri-stated buffer gates. The control input G is connected to the clock so as to trigger the D flip-flops.

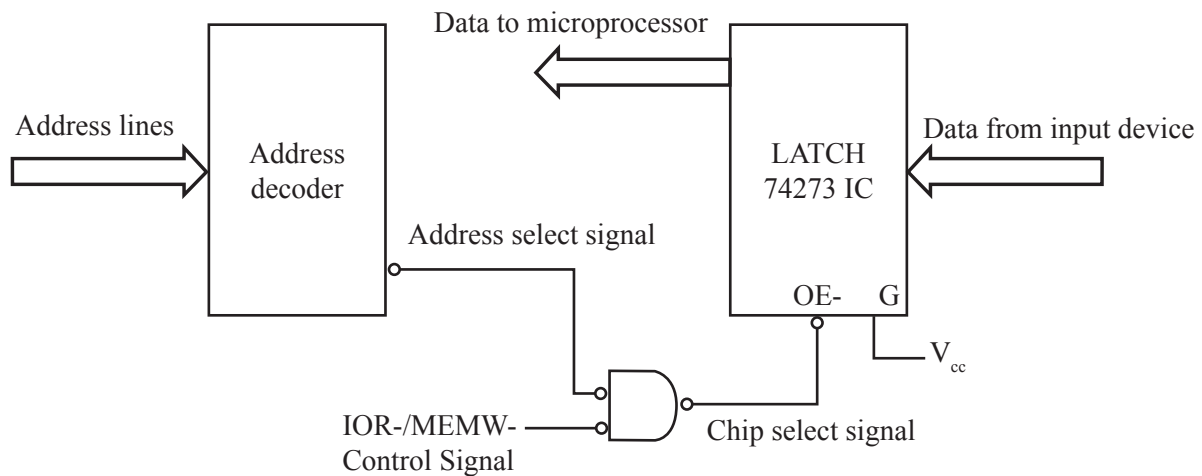


Fig. 3.9 Principal of interfacing an output device

When microprocessor wants to send some data for display, the OE signal goes low, it will trigger the flip-flops and the microprocessor data will be latched into the latches, which will then be available at the display section.

3.5 The 8085 Programming Model

The 8085 microprocessor registers with reference to the internal data operations. The same information is repeated here briefly to provide the continuity and the context to the instruction set and to enable the readers who prefer to focus initially on the programming aspect of the microprocessor.

The 8085 programming model includes six registers, one accumulator, and one flag register, as shown in figure. In addition, it has two 16-bit registers: the stack pointer and the program counter. They are described briefly as follows.

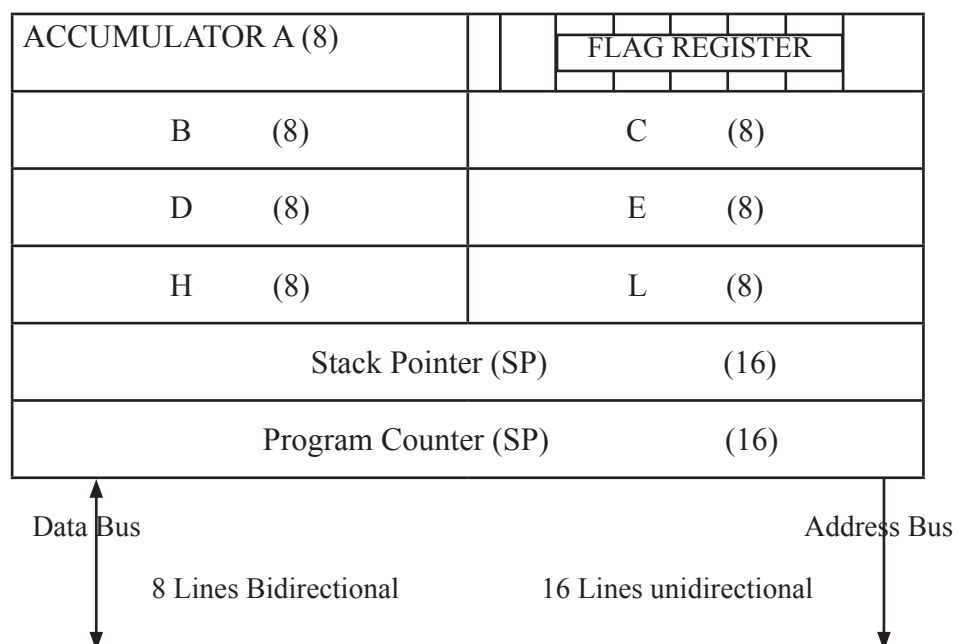


Fig. 3.10 8085 Microprocessor programming model

Registers

The 8085 has six general-purpose registers to store 8-bit data; these are identified as B, C, D, E, H, and L as shown in the figure. They can be combined as register pairs -BC, DE, and HL - to perform some 16-bit operations. The programmer can use these registers to store or copy data into the registers by using data copy instructions.

Accumulator

The accumulator is an 8-bit register that is a part of 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, which are set or reset after an operation according to data conditions of the result in the accumulator and other registers. They are called Zero (Z), Carry (CY), Sign (S), Parity (P), and Auxiliary Carry (AC) flags; their bit positions in the flag register are shown in the Figure below. The most commonly used flags are Zero, Carry, and Sign. The microprocessor uses these flags to test data conditions.

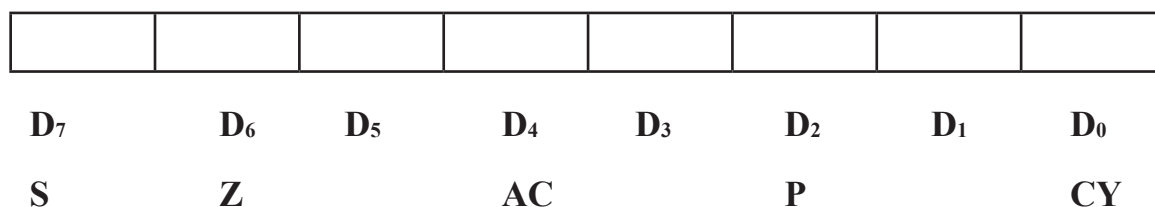


Fig. 3.11 Flag register

For example, after an addition of two numbers, if the sum in the accumulator is larger than eight bits, the flip-flop used to indicate a carry called the Carry flag (CY) – is set to one. When an arithmetic operation results in zero, the flip-flop called the Zero (Z) flag is set to one. The first Figure shows an 8-bit register, called the flag register, adjacent to the accumulator. However, it is not used as a register; five bit positions out of eight are used to store the outputs of the five flip-flops. The flags are stored in the 8-bit register so that the programmer can examine these flags (data conditions) by accessing the register through an instruction.

These flags have critical importance in the decision-making process of the micro-processor. The conditions (set or reset) of the flags are tested through the software instructions. For example, the instruction JC (Jump on Carry) is implemented to change the sequence of a program when CY flag is set. The thorough understanding of flag is essential in writing assembly language programs.

Program counter (PC)

This 16-bit register deals with sequencing the execution of instructions. This register is a memory pointer. Memory locations have 16-bit addresses, and that is why this is a 16-bit register. The microprocessor uses this register to sequence the execution of the 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. It points to a memory location in R/W memory, called the stack. The beginning of the stack is defined by loading 16-bit address in the stack pointer. This programming model will be used in subsequent tutorials to examine how these registers are affected after the execution of an instruction.

3.6 The 8085 Addressing Modes

The instructions MOV B, A or MVI A, 82H are to copy data from a source into a destination. In these instructions the source can be a register, an input port, or an 8-bit number (00H to FFH). Similarly, a destination can be a register or an output port. The sources and destination are operands. The various formats for specifying operands are called the addressing modes, which are explained below.

Direct addressing

Used to accept data from outside devices to store in the accumulator or send the data stored in the accumulator to the outside device. Accept the data from the port 00H and store them into the accumulator or Send the data from the accumulator to the port 01H.

Example: IN 00H or OUT 01H

Indirect addressing

This means that the effective address is calculated by the processor and the contents of the address (and the one following) are used to form a second address. The second address is where the data is stored. Note that this requires several memory accesses; two accesses to retrieve the 16-bit address and a further access (or accesses) to retrieve the data which is to be loaded into the register.

3.7 Instruction Set Classification

An instruction is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions, called the instruction set, determines what functions the microprocessor can perform. These instructions can be classified into the following five functional categories: data transfer (copy) operations, arithmetic operations, logical operations, branching operations, and machine-control operations.

Data transfer (copy) operations

This group of instructions copy data from a location called a source to another location called a destination, without modifying the contents of the source. In technical manuals, the term data transfer is used for this copying function. However, the term transfer is misleading; it creates the impression that the contents of the source are destroyed when, in fact, the contents are retained without any modification.

The various types of data transfer (copy) are listed below together with examples of each type:

Types	Examples
Between Registers	Copy the contents of the register B into register D
Specific data byte to a register or a memory location.	Load register B with the data byte 32 H
Between a memory location and a register.	From a memory location 2000H to register B
Between an I/O device and the accumulator.	From an input keyboard to the accumulator.

Table 3.1 Types of data transfer

Arithmetic operations

These instructions perform arithmetic operations such as addition, subtraction, increment, and decrement.

Addition

Any 8-bit number, or the contents of a register or the contents of a memory location can be added to the contents of the accumulator and the sum is stored in the accumulator. No two other 8-bit registers can be added directly (e.g., the contents of register B cannot be added directly to the contents of the register C). The instruction DAD is an exception; it adds 16-bit data directly in register pairs.

Subtraction

Any 8-bit number, or the contents of a register, or the contents of a memory location can be subtracted from the contents of the accumulator and the results stored in the accumulator. The subtraction is performed in 2's complement, and the results if negative, are expressed in 2's complement. No two other 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. Similarly, the 16-bit contents of a register pair (such as BC) can be incremented or decremented by 1. These increment and decrement operations differ from addition and subtraction in an important way; i.e., they can be performed in any one of the registers or in a memory location.

Logical operations

These instructions perform various logical operations with the contents of the accumulator.

AND, OR Exclusive-OR

Any 8-bit number, or the contents of a register, or of a memory location can be logically ANDed, Ored, or Exclusive-ORed with the contents of the accumulator. The results are stored in the accumulator.

Summary

- The 8085A is an 8-bit microprocessor suitable for a wide range of applications.
- It operates on a single +5V power supply connected at V_{cc} ; power supply ground is connected to V_{ss} .
- It operates on clock cycle with 50% duty cycle.
- It has 16 address lines, hence it can access (2^{16}) 64 Kbytes of memory.
- It provides 8 bit I/O address to access (2^8) 256 I/O ports.
- It has serial I/O control which allows serial communication.
- The architecture of the 8085 microprocessor describes various units like accumulator, program counter, stack pointer, flags, etc.
- I/O devices are used to enter as well as take out data from the microprocessor fro external world.
- I/O refers to the communication between an information processing system and the outside world-possibly a human or another information processing system.
- 8085 programming model includes six registers, one accumulator, and one flag register.
- Instruction is a binary pattern designed inside a microprocessor to perform a specific function.

References

- Srinath, N. K., 2005. *8085 MICROPROCESSOR: PROGRAMMING AND INTERFACING*, PHI Learning Pvt. Ltd.
- Godse, D. A. & Godse, A. P., 2007. *Introduction To Microprocessor*, Technical Publications.
- Fawwaz, W., *The 8085 microprocessor* [Pdf] Available at: <<http://www.uotechnology.edu.iq/dep-cse/lectures/3/control/mico.pdf>> [Accessed 30 May 2013].
- *MICROPROCESSOR 8085* [Pdf] Available at: <<http://itsnka.files.wordpress.com/2010/10/mpmanual-first-ver.pdf>> [Accessed 30 May 2013].
- Prof. Pal, A., 2012. *lec 2 - Architecture and Organization of 8085* [Video online] Available at: <<http://www.youtube.com/watch?v=p4RcMLFIr5o>> [Accessed 30 May 2013].
- Prof. Pal, A., 2012. *lec 1 - Introduction to Microprocessors & Microcontrollers* [Video online] Available at: <<http://www.youtube.com/watch?v=liRPtvj7bFU>> [Accessed 30 May 2013].

Recommended Reading

- Kumar, U. K., 2008. *The 8085 Microprocessor: Architecture, Programming and Interfacing*, Pearson Education India.
- Godse, D. A. & Godse, A. P., 2008. *Microprocessors And Its Applications*, Technical Publications.
- Seeger, A. S., 1988. *Introduction to Microprocessors With the Intel 8085*, Harcourt Brace Jovanovich.

Self Assessment

1. Microprocessor is _____ microprocessor, i.e., it can accept, process, or provide 8-bit data simultaneously.
 - a. 8-bit
 - b. 16-bit
 - c. 32-bit
 - d. 64-bit
2. It provides four hardware interrupts: _____, RST 7.5, RST 5.5 and INTR.
 - a. RAPT
 - b. TRAP
 - c. PART
 - d. APRT
3. The 8085 has an ability to share system bus with _____ controller.
 - a. accumulator
 - b. I/O devices
 - c. IO Addressing
 - d. direct memory access
4. _____ are used to enter as well as take out data from the microprocessor for external world.
 - a. I/O devices
 - b. IO Addressing
 - c. Direct memory access
 - d. Accumulator
5. For any microprocessor system, we require memories to store monitor program, to store _____ and to store data.
 - a. user program
 - b. internal data operations
 - c. registers
 - d. flags
6. Which of the following statements is true?
 - a. Memories are broadly classified as primary and secondary memory.
 - b. Memories are broadly classified as internal and external memory.
 - c. Memories are broadly classified as first and second memory.
 - d. Memories are broadly classified as intermediate and advanced memory.
7. Which of the following statements is true?
 - a. Outputs are the signals or data received by the system and inputs are the signals or data sent from it.
 - b. Inputs are the signals or data received by the system and outputs are the signals or data sent from it.
 - c. Inputs are the indicator received by the system and outputs are the signs sent from it.
 - d. Feeds are the signals or data received by the system and results are the signals or data sent from it.

8. Which of the following statements is true?
 - a. Input devices are used to input data to the registers.
 - b. Input devices are used to input data to the microprocessor.
 - c. Output devices are used to input data to the microprocessor.
 - d. Input devices are used to output data to the microprocessor.

9. Which of the following statements is true?
 - a. The output of NAND gate will generate the chip select signal, which is applied to the output control signal OE of the 74732 latch chip.
 - b. The output of NAND gate will generate the chip select signal, which is applied to the output control signal OE of the 7473 latch chip.
 - c. The output of NAND gate will generate the chip select signal, which is applied to the output control signal OE of the 74273 latch chip.
 - d. The output of NAND gate will generate the chip select signal, which is applied to the output control signal OE of the 7273 latch chip.

10. Which of the following statements is true?
 - a. The 8085 programming model includes six registers, one accumulator, and one flag register.
 - b. The 8085 programming model includes five registers, two accumulators, and one flag register.
 - c. The 8085 programming model includes six registers, three accumulators, and two flags register.
 - d. The 8085 programming model includes eight registers, one accumulator, and one flag register.

Chapter IV

System Buses

Aim

The aim of the chapter is to:

- introduce the concept of bus system
- explain PCI and its features
- explicate the concept of PCI arbitration

Objectives

The objectives of the chapter are to:

- explain bus system
- define bus taxonomy
- enlist PCI and its features

Learning outcome

At the end of this chapter, you will be able to:

- understand bus system and bus taxonomy
- identify PCI and its features
- determine basic cache structure

4.1 Introduction

Fundamentally, a computer bus consists of a set of parallel “wires” attached to several connectors into which peripheral boards may be plugged, as shown in Fig. 4.1. Typically, the processor is connected to one end of these wires. Memory may also be attached via the bus.

The wires are split into several functional groups such as:

- Address: Specifies the peripheral and register within the peripheral that is being accessed.
- Data: The information being transferred to or from the peripheral.
- Control: Signals that affect the data transfer operation are the control signals and the way they are manipulated embodies the bus protocol.

Beyond basic data transfer, busses typically incorporate advanced features such as:

- Interrupts
- DMA
- Power distribution

The classic concept of a bus is a set of boards plugged into a passive backplane as shown in Fig. 4.1. But there are also many bus implementations based on cables interconnecting stand-alone boxes. The GPIB (general purpose interface bus) is a classic example.

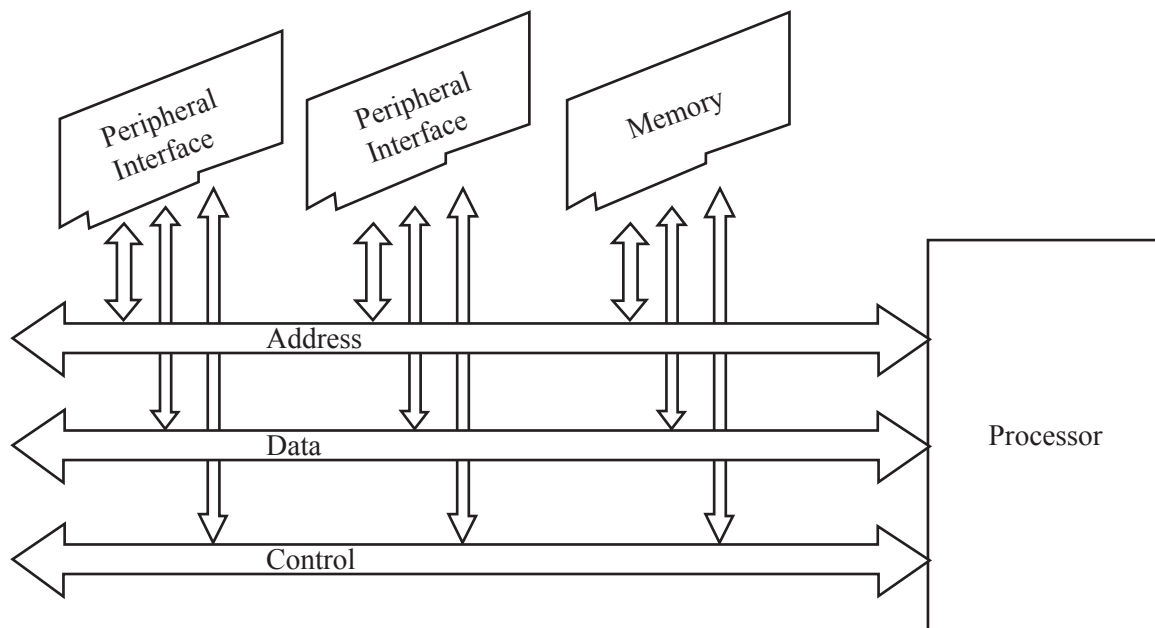


Fig. 4.1 Functional diagram of a computer bus

Contemporary examples of cable busses include USB (universal serial bus) and IEEE-1394 (trademarked by Apple Computer under the name Firewire™). The backplane is not restricted to being passive as illustrated by the typical PC motherboard implementation.

Bus Taxonomy

Computer busses can be characterised along a number of dimensions. Architecturally, busses can be characterised along two binary dimensions: synchronous vs. asynchronous and multiplexed vs. non-multiplexed. In a synchronous bus, all operations occur on a specified edge of a master clock signal. In asynchronous busses operations occur on specified edges of control signals without regard to a master clock. Early busses tended to be asynchronous. Contemporary busses are generally synchronous.

A bus can be either multiplexed or non-multiplexed. In a multiplexed bus data and address share the same signal lines. Control signals identify when the common lines contain address information and when they contain data. A non-multiplexed bus has separate wires for address and data.

The basic advantage of a multiplexed bus is fewer wires which in turn means fewer pins on each connector, fewer high-power driver circuits and so on. The disadvantage is that it requires two phases to carry out a single data transfer: first the address must be sent and then the data is transferred. Contemporary busses are about evenly split between multiplexed and non-multiplexed.

Table 4.1 lists some of the quantifiable dimensions of bus design. Busses can be characterised in terms of the number of bits of address and data. Contemporary busses are typically either 32 or 64 bits wide for both address and data. Not surprisingly, multiplexed busses tend to have the same number of address and data bits.

Address width	8, 16, 32, 64
Data width	1, 8, 16, 32, 64
Transfer rate	1 MHz up to several hundred MHz
Maximum length	Several centimeters to several meters
Number of devices	A few up to many

Table 4.1 Bus parameters

A key element of any bus protocol is performance. How fast can it transfer data? Early busses were limited to a few megahertz, which closely matched processor performance of the era. The problem in contemporary systems is that the processor is often many times faster than the bus and so the bus becomes a performance bottleneck. Bus length is related to transfer speed. Early busses with transfer rates of one or two megahertz allowed maximum lengths of several meters. But with higher transfer rates comes shorter lengths so that propagation delay doesn't adversely impact performance.

The maximum number of devices that can be connected to a bus is likewise restricted by high performance considerations. Early busses could tolerate high-power, relatively slow driver circuits and could thus support a large number of attached devices. High performance busses such as PCI limit driver power and so are severely restricted in terms of number of devices.

4.2 Peripheral Component Interconnect (PCI)

PCI is an interconnection system between a microprocessor and attached devices in which expansion slots are spaced closely for high speed operation. Using PCI, a computer can support both new PCI cards while continuing to support Industry Standard Architecture (ISA) expansion cards, an older standard. Designed by Intel, the original PCI was similar to the VESA Local Bus. However, PCI 2.0 is no longer a local bus and is designed to be independent of microprocessor design. PCI is designed to be synchronised with the clock speed of the microprocessor.

PCI is now installed on most new desktop computers, not only those based on Intel's Pentium processor but also those based on the PowerPC. PCI transmits 32 bits at a time in a 124-pin connection (the extra pins are for power supply and grounding) and 64 bits in a 188-pin connection in an expanded implementation. PCI uses all active paths to transmit both address and data signals, sending the address on one clock cycle and data on the next. Burst data can be sent starting with an address on the first cycle and a sequence of data transmissions on a certain number of successive cycles.

4.3 Features of PCI

The features of PCI are mentioned as follows:

- The maximum theoretical transfer rate of the base configuration is 132 Mbytes/sec. Currently defined extensions can boost this by a factor of four to 528 Mbytes/sec.
- Any device on the bus can be a bus master and initiate transactions. One consequence of this is that there is no need for the traditional notion of DMA.
- The transfer protocol is optimised around transferring blocks of data. A single transfer is just a block transfer with a length of one.

- Although PCI is officially processor-independent, it inevitably reflects its origins with Intel and its primary application in the PC architecture. Among other things it uses little-endian byte ordering.
- PCI implements Plug and Play configurability. Every device in a system is automatically configured each time the system is turned on. The configuration protocol supports up to 256 devices in a system.
- The electrical specifications emphasise low power use including support for both 3.3 and 5 volt signaling environments. PCI is a “green” architecture.

4.4 Concept of PCI Arbitration

Since the PCI Bus accommodates multiple masters, any of which could request the use of the bus at any time there must be a mechanism that allocates use of bus resources in a reasonable way and resolves conflicts among multiple masters wishing to use the bus simultaneously. Fundamentally, this is called bus arbitration.

4.4.1 The Arbitration Process

Before a bus master can execute a PCI transaction, it must request, and be granted, use of the bus. For this purpose, each bus master has a pair of REQ# and GNT# signals connecting it directly to a central arbiter as shown in fig. 2.2. When a master wishes to use the bus, it asserts its REQ# signal. Some time later the arbiter will assert the corresponding GNT# indicating that this master is next in line to use the bus.

Only one GNT# signal can be asserted at any instant in time. The master agent who sees his GNT# asserted may initiate a bus transaction when it detects that the bus is idle. The bus idle state is defined as both FRAME# and IRDY# de-asserted. Fig.4.3 is a timing diagram illustrating how arbitration works when two masters request use of the bus simultaneously.

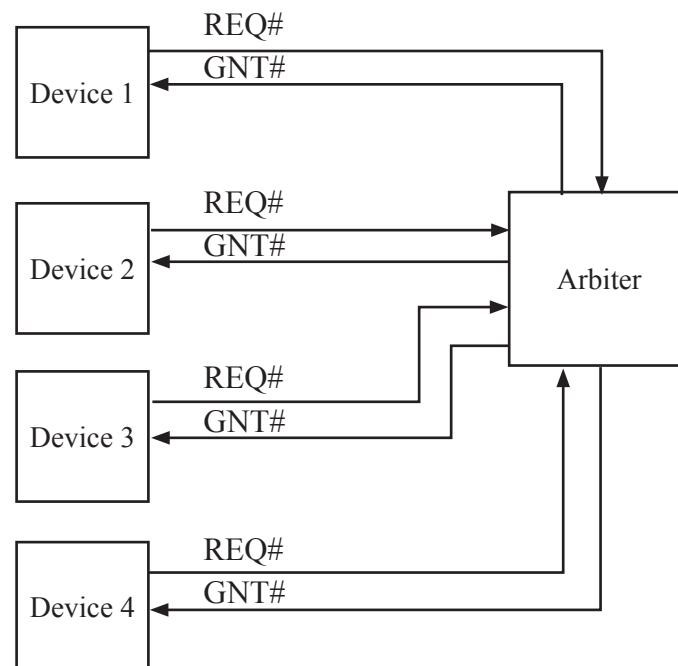


Fig. 4.2 Arbitration process under PCI

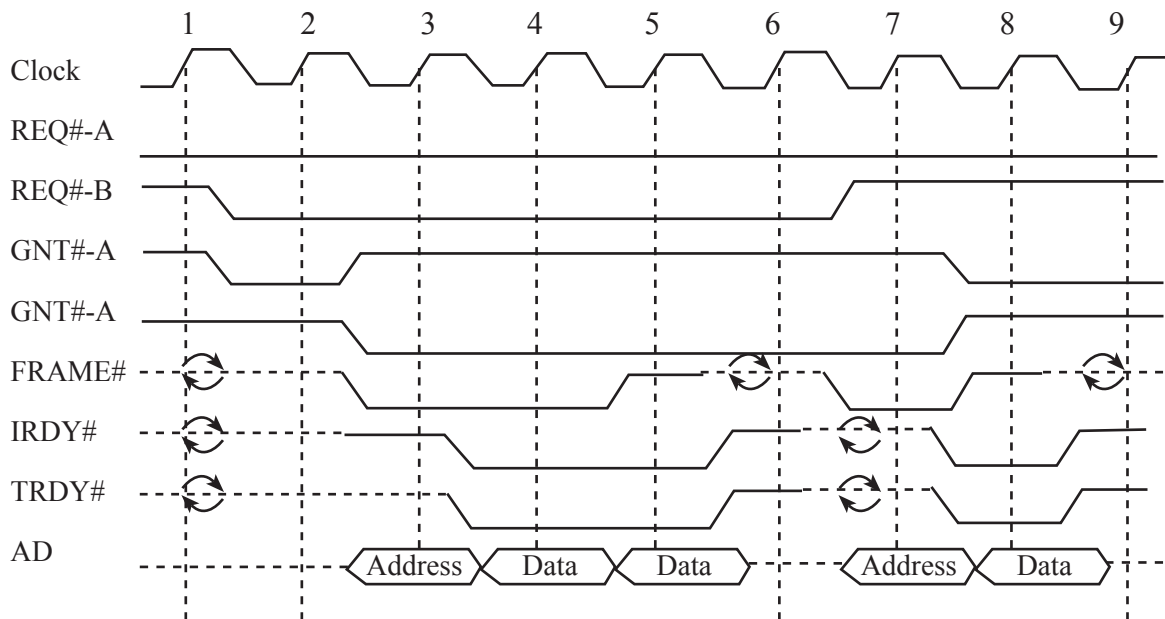


Fig. 4.3 Timing diagram for arbitration process involving two masters

4.4.2 Clock

The arbiter detects that device A has asserted its REQ#. No one else is asserting a REQ# at the moment so the arbiter asserts GNT#-A. In the meantime device B asserts its REQ#. Device A detects its GNT# asserted, the bus is idle and so it asserts FRAME# to begin its transaction. Device A keeps its REQ# asserted indicating that it wishes to execute another transaction after this one is complete. Upon detecting REQ#-B asserted, the arbiter deasserts GNT#-A and asserts GNT#-B.

Device B detects its GNT# asserted but can't do anything yet because a transaction is in process. Device B detects that the bus is idle because both FRAME# and IRDY# are deasserted. In response, it asserts FRAME# to start its transaction. It also deasserts its REQ# because it does not need a subsequent transaction. The arbiter detects REQ#-B deasserted. In response it deasserts GNT#-B and asserts GNT#-A since REQ#-A is still asserted.

Arbitration is "hidden," which means that arbitration for the next transaction occurs at the same time as, or in parallel with, the current transaction. So the arbitration process doesn't take any time. The specification does not stipulate the nature of the arbitration algorithm or how it is to be implemented other than to say that arbitration must be "fair." This is not to say that there cannot be a relative priority scheme among masters but rather that every master gets a chance at the bus. Note in Fig. 4.3 that even though Device A wants to execute another transaction, he must wait until Device B has executed his transaction.

4.4.3 An Example of Fairness

Fig. 2.4 offers an example of what the specification means by fairness. This is taken directly from the specification. In this example, a bus master can be assigned to either of two arbitration levels. Agents assigned to Level 1 have a greater need for use of the bus than those assigned to Level 2. Agents at Level 2 have equal access to the bus with respect to other second level agents. Furthermore, Level 2 agents, as a group, have equal access to the bus as Level 1 agents.

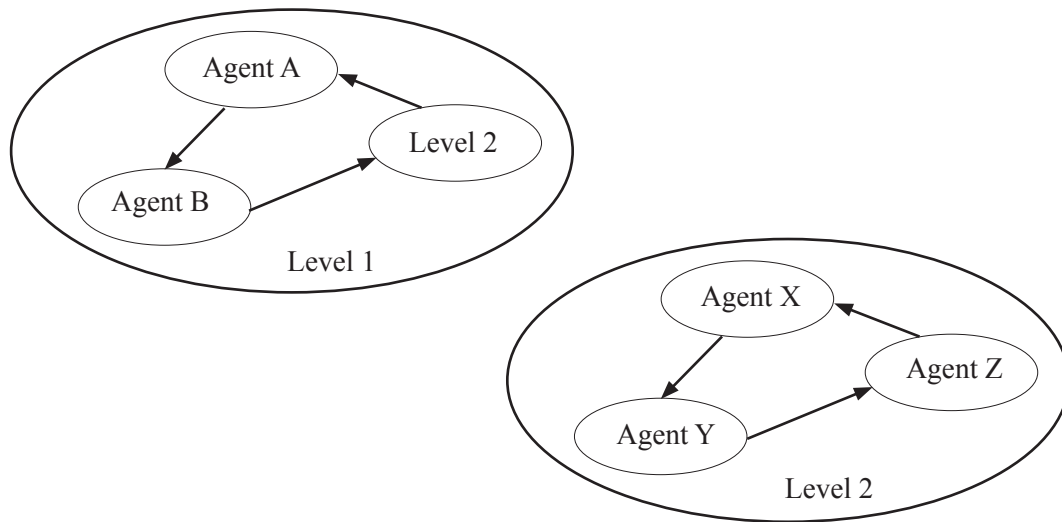


Fig. 4.4 Example of fairness in arbitration

Consider the case that all agents in the figure above have their REQ# signals asserted and continue to assert them. If Agent A is the next Level 1 agent to receive the bus and Agent X is next for Level 2, then the order of bus access would be:

- A, B, Level 2 (X)
- A, B, Level 2 (Y)
- A, B, Level 2 (Z) and so forth.

If only Agents B and Y had their REQ# signals asserted, the order would be:

- B, Level 2 (Y)
- B, Level 2 (Y)

Typically, high performance agents like video, ATM or FDDI would be assigned to Level 1 while devices like a LAN or SCSI disk would go on Level 2. This allows the system designer to tune the system for maximum throughput and minimal latency without the possibility of starvation.

It is often the case that when a standard offers an example or suggestion of how some feature may be implemented, it becomes a de facto standard as most vendors choose that particular implementation. So it is with arbitration algorithms. Many chipset and bridge vendors have implemented the priority scheme described by this example.

4.4.4 Bus Parking

A master device is only allowed to assert its REQ# when it actually needs the bus to execute a transaction. In other words, it is not allowed to continuously assert REQ# in order to monopolise the bus. This violates the low-latency spirit of the PCI spec. On the other hand, the specification does allow the notion of “bus parking.”

The arbiter may be designed to “park” the bus on a default master when the bus is idle. This is accomplished by asserting GNT# to the default master when the bus is idle. The agent on which the bus is parked can initiate a transaction without first asserting REQ#. This saves one clock. While the choice of a default master is up to the system designer, the specification recommends parking on the last master that acquired the bus.

4.4.5 Latency

When a bus master asserts REQ#, a finite amount of time expires until the first data element is actually transferred. This is referred to as bus access latency and consists of several components as shown in fig. 4.5.

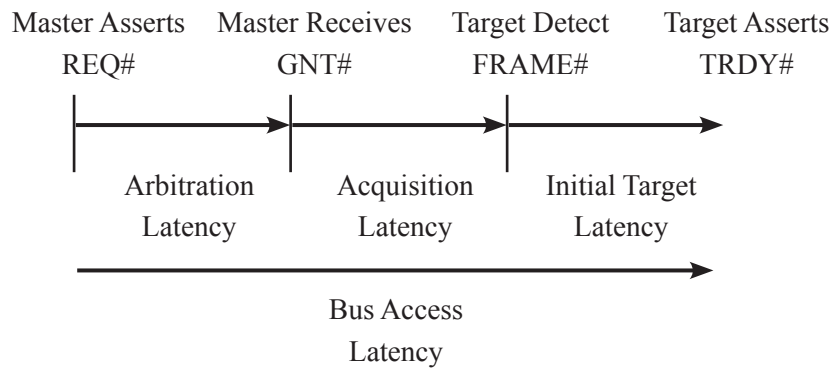


Fig. 4.5 Components of bus latency

4.4.6 Arbitration Latency

The time from when the master asserts REQ# until it receives GNT#. This is a function of the arbitration algorithm and the number of other masters requesting use of the bus that may be ahead of this one in the arbitration queue.

4.4.7 Acquisition Latency

The time from when the master receives GNT# until the targets recognise that FRAME# is asserted. If the bus is idle, this is only one or two clock cycles. Otherwise, it is a function of the Latency Timer in the master currently using the bus.

4.4.8 Initial Target Latency

The time from when the selected target detects FRAME# asserted until it asserts TRDY#. Target latency for the first data transfer is often longer than the latency on subsequent transfers because the device may need extra time to prepare a block of data- a disk may have to wait for the sector to come around for example. The specification limits initial target latency to 16 clocks and subsequent latency to 8 clocks.

4.4.9 Latency Timer

The PCI specification goes to great lengths to give designers and integrators facilities for balancing and fine tuning systems for optimal performance. One of these facilities is the Latency Timer that is required in every master device that is capable of burst lengths greater than two.

The purpose of the Latency Timer is to prevent a master from hogging the bus if other masters require access. The value programmed into the Latency Timer (or hardwired) represents the minimum number of clock cycles a master gets when it initiates a transaction.

When a master asserts FRAME#, the Latency Timer is loaded with the hardwired or configuration-programmed value. Each clock cycle thereafter decrements the counter. If the counter reaches 0 before the transaction completes and the master's GNT# is not asserted, that means another master needs to use the bus and so the current master must terminate its transaction. The current master will most likely immediately request the bus so it can finish its transaction. But of course it won't get the bus until all other masters currently requesting the bus have finished.

4.4.10 Bandwidth vs. Latency

In PCI, there is a tradeoff between the desire for low latency and the complementary desire for high bandwidth (throughput). High throughput is achieved by allowing devices to use long burst transfers. Conversely, low latency results from reducing the maximum burst length.

A master is required to assert its IRDY# within eight clocks for any given data phase. The selected target is required to assert TRDY# within 16 clocks from the assertion of FRAME# for the first data phase (32 clocks if the access

hits a modified cache line). For subsequent data phases the target must assert TRDY# or STOP# within 8 clocks. If we ignore the effects of the Latency Timer, it is a straightforward exercise to develop equations for worst case latencies. If a modified cache line is hit:

$$\text{Latency}_{\text{max}} = 32 + 8 \cdot (n - 1) + 1 \text{ (clocks)}$$

Otherwise:

$$\text{Latency}_{\text{max}} = 16 + 8 \cdot (n - 1) + 1 \text{ (clocks)}$$

where n is the total number of data transfers. The extra clock is the idle cycle introduced between most transactions.

Nevertheless, it is more useful to consider transactions that exhibit typical behavior. PCI bus masters typically don't insert wait states because they only request transactions when they are prepared to transfer data. Likewise, once a target begins transferring data it can usually sustain the full data rate of one transfer per clock cycle. Targets typically have an initial access latency of less than 16 (or 32) clock cycles. Again ignoring the effects of the Latency Timer, typical latency can be expressed as:

$$\text{Latency}_{\text{typ}} = 8 + (n - 1) + 1 \text{ (clocks)}$$

The Latency Timer effectively controls the tradeoff between high throughput and low latency.

Table 4.2 illustrates this tradeoff between latency and throughput for different burst lengths based on the typical latency equation just developed.

Data Phases	Bytes Transferred	Total Clocks	Bandwidth (Mb/sec)	Latency(us)
8	32	16	60	0.48
16	64	24	80	0.72
32	128	40	96	1.20
64	256	72	107	2.16

Table 4.2 Bandwidth vs. latency

Total Clocks: total number of clocks required to complete the transaction.

Latency Time: The Latency Timer is set to expire on the next to the last data transfer.

Bandwidth: calculated bandwidth in MB/sec

$$\text{Bandwidth} = \text{bytes transferred} / (\text{total clocks} \times 30\text{ns})$$

Latency: latency in microseconds resulting from the transaction

$$\text{Latency} = \text{total clocks} \times 0.030 \text{ us}$$

Notice that the amount of data transferred per transaction doubles from row to row but the latency doesn't quite double. From first row to last row the amount of data transferred increases by a factor of 8 while latency increases by about 4.5. This reflects the fact that there is some overhead in every PCI transactions and so the longer the transaction, the more efficient the bus is.

Note that it's not uncommon to find devices that routinely violate the latency rules, particularly among older devices derived from ISA designs. How should an agent respond to excessive latency, or indeed any protocol violations? The specification states "A device is not encouraged actively to check for protocol errors."

In effect, the protocol rules define "good behaviour" that well-behaved devices are expected to observe. Devices that aren't so well behaved are tolerated.

Basic cache structure

Processors are generally able to perform operations on operands faster than the access time of large capacity main memory. Though semiconductor memory which can operate at speeds comparable with the operation of the processor exists, it is not economical to provide all the main memory with very high speed semiconductor memory. The problem can be alleviated by introducing a small block of high speed memory called a cache between the main memory and the processor.

The idea of cache memories is similar to virtual memory in that some active portion of a low-speed memory is stored in duplicate in a higher-speed cache memory. When a memory request is generated, the request is first presented to the cache memory, and if the cache cannot respond, the request is then presented to main memory.

The difference between cache and virtual memory is a matter of implementation; the two notions are conceptually the same because they both rely on the correlation properties observed in sequences of address references. Cache implementations are totally different from virtual memory implementation because of the speed requirements of cache. We define a cache miss to be a reference to a item that is not resident in cache, but is resident in main memory. The corresponding concept for cache memories is page fault, which is defined to be a reference to a page in virtual memory that is not resident in main memory. For cache misses, the fast memory is cache and the slow memory is main memory. For page faults the fast memory is main memory, and the slow memory is auxiliary memory.

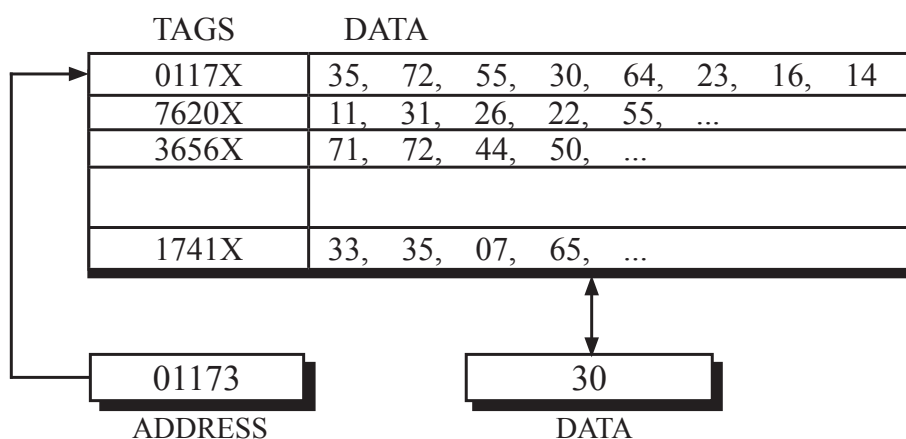


Fig. 4.6 A cache memory reference

The tag 0117X matches address 01173, so the cache returns the item in the position X=3 of the matched block. Fig. 4.6 shows the structure of a typical cache memory. Each reference to a cell in memory is presented to the cache. The cache searches its directory of address tags shown in the figure to see if the item is in the cache. If the item is not in the cache, a miss occurs.

For read operations that cause a cache miss, the item is retrieved from main memory and copied into the cache. During the short period available before the main-memory operation is complete, some other item in cache is removed from the cache to make room for the new item.

The cache-replacement decision is critical; a good replacement algorithm can yield somewhat higher performance than can a bad replacement algorithm. The effective cycle-time of a cache memory (t_{eff}) is the average of cache-memory cycle time (t_{cache}) and main-memory cycle time (t_{main}), where the probabilities in the averaging process are the probabilities of hits and misses.

If we consider only READ operations, then a formula for the average cycle-time is:

$$t_{eff} = t_{cache} + (1 - h) t_{main}$$

where h is the probability of a cache hit (sometimes called the hit rate), the quantity $(1 - h)$, which is the probability of a miss, is known as the miss rate.

In Fig.4.6 we show an item in the cache surrounded by nearby items, all of which are moved into and out of the cache together. We call such a group of data a block of the cache.

4.5 Cache Memory Organisations

Fig.4.7 shows a conceptual implementation of a cache memory. This system is called set associative because the cache is partitioned into distinct sets of blocks, and each set contains a small fixed number of blocks. The sets are represented by the rows in the figure. In this case, the cache has N sets, and each set contains four blocks. When an access occurs to this cache, the cache controller does not search the entire cache looking for a match. Instead, the controller maps the address to a particular set of the cache and searches only the set for a match. Some of the features of cache memory organisation are as follows.

Fig. 4.7 The logical organisation of a four-way set-associate cache

- If the block is in the cache, it is guaranteed to be in the set that is searched. Hence, if the block is not in that set,

	Associativity			
	1	2	3	4
Set 0				
Set 1				
Set 2				
Set 3				
Set 4				
Set 5				
	⋮	⋮	⋮	⋮
Set $N-2$				
Set $N-1$				

the block is not present in the cache, and the cache controller searches no further.

- Because the search is conducted over four blocks, the cache is said to be four-way set associative or, equivalently, to have an associativity of four.
- Fig.4.7 is only one example, there are various ways that a cache can be arranged internally to store the cached data.
- In all cases, the processor refers the cache with the main memory address of the data it wants. Hence each cache organisation must use this address to find the data in the cache if it is stored there, or to indicate to the processor when a miss has occurred.
- The problem of mapping the information held in the main memory into the cache must be totally implemented in hardware to achieve improvements in the system operation. Various strategies are possible, as described below.

4.5.1 Fully Associative Mapping

Perhaps the most obvious way of relating cached data to the main memory address is to store both memory address and data together in the cache. This is the fully associative mapping approach. Some of the features of fully associative mapping are as follows.

- A fully associative cache requires the cache to be composed of associative memory holding both the memory address and the data for each cached line.
- The incoming memory address is simultaneously compared with all stored addresses using the internal logic of the associative memory, as shown in Fig.4.8.
- If a match is found, the corresponding data is read out. Single words from anywhere within the main memory could be held in the cache, if the associative part of the cache is capable of holding a full address.

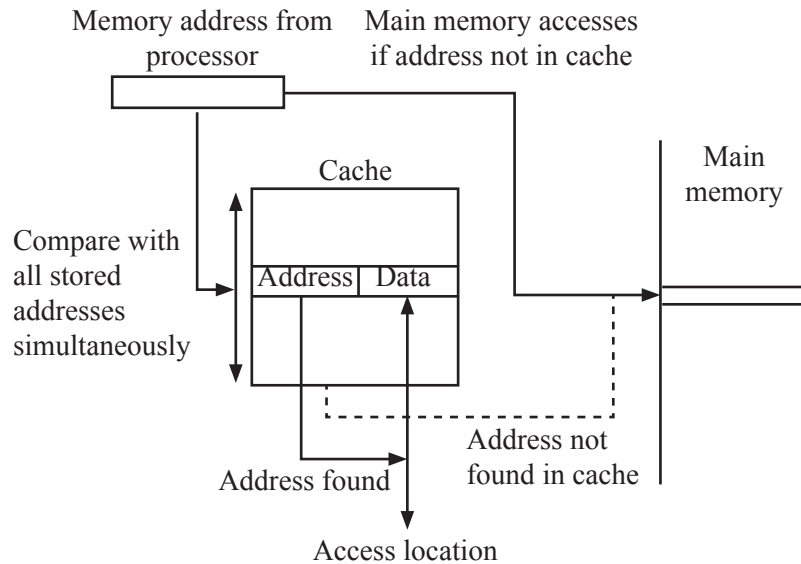


Fig. 4.8 Cache with fully associative mapping

- In all organisations, the data can be more than one word, i.e., a block of consecutive locations to take advantage of spatial locality.
- In Fig.4.9 a line constitutes four words, each word being 4 bytes. The least significant part of the address selects the particular byte, the next part selects the word, and the remaining bits form the address compared to the address in the cache.
- The whole line can be transferred to and from the cache in one transaction if there are sufficient data paths between the main memory and the cache.
- With only one data word path, the words of the line have to be transferred in separate transactions.

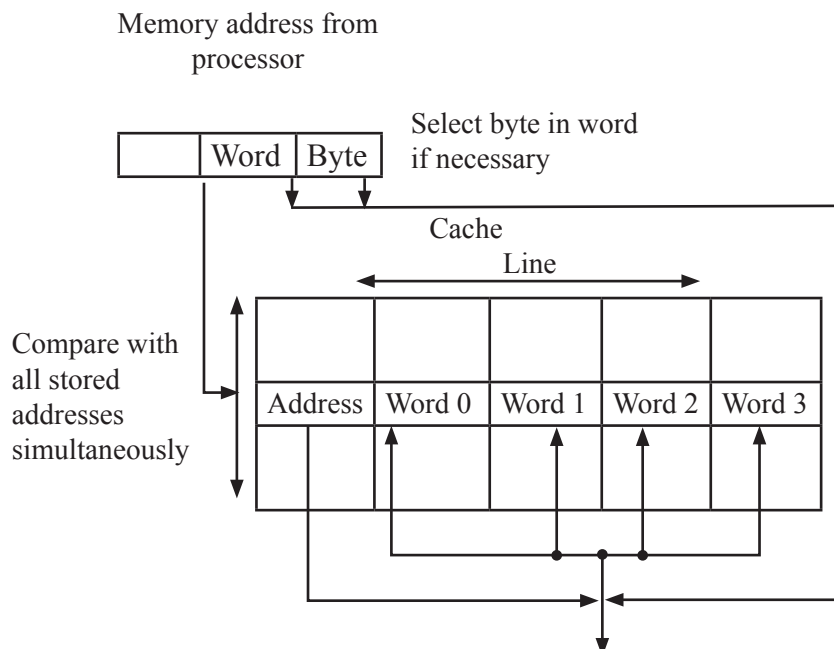


Fig. 4.9 Fully associative mapped cache with multi-word lines

- The fully associate mapping cache gives the greatest flexibility of holding combinations of blocks in the cache and minimum conflict for a given sized cache, but is also the most expensive, due to the cost of the associative memory.

- It requires a replacement algorithm to select a block to remove upon a miss and the algorithm must be implemented in hardware to maintain a high speed of operation.
- The fully associative cache can only be formed economically with a moderate size capacity.
- Microprocessors with small internal caches often employ the fully associative mechanism.

4.5.2 Direct Mapping

The fully associative cache is expensive to implement because of requiring a comparator with each cache location, effectively a special type of memory. In direct mapping, the cache consists of normal high speed random access memory, and each location in the cache holds the data, at an address in the cache given by the lower significant bits of the main memory address. This enables the block to be selected directly from the lower significant bits of the memory address. The remaining higher significant bits of the address are stored in the cache with the data to complete the identification of the cached data.

Consider the example shown in fig.4.10. The address from the processor is divided into two fields, a tag and an index. The tag consists of the higher significant bits of the address, which are stored with the data. The index is the lower significant bits of the address used to address the cache.

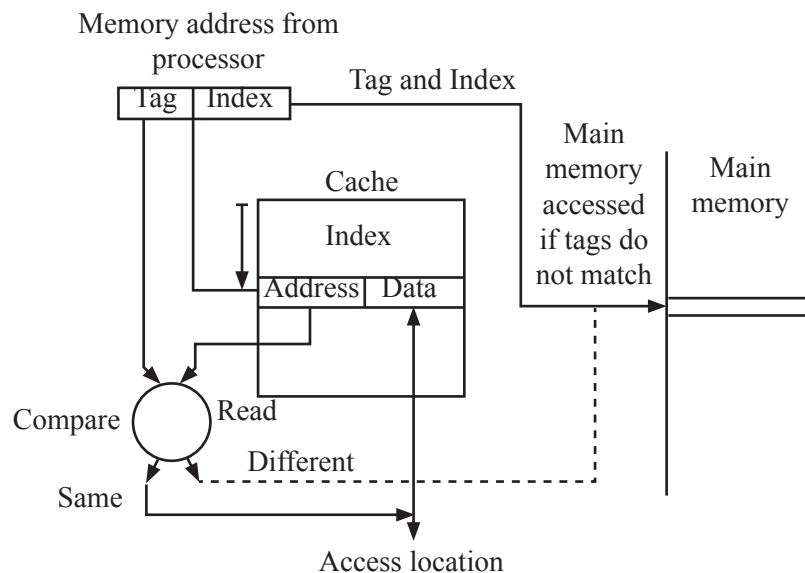


Fig. 4.10 Cache with direct mapping

When the memory is referenced, the index is first used to access a word in the cache. Then the tag stored in the accessed word is read and compared with the tag in the address. If the two tags are the same, indicating that the word is the one required, access is made to the addressed cache word. However, if the tags are not the same, indicating that the required word is not in the cache, reference is made to the main memory to find it.

For a memory read operation, the word is then transferred into the cache where it is accessed. It is possible to pass the information to the cache and the processor simultaneously i.e. to read-through the cache, on a miss. The cache location is altered for a write operation. The main memory may be altered at the same time (write-through) or later. Fig. 4.10 shows the direct mapped cache with a line consisting of more than one word.

The main memory address is composed of a tag, an index, and a word within a line. All the words within a line in the cache have the same stored tag. The index part to the address is used to access the cache and the stored tag is compared with required tag address.

For a read operation, if the tags are the same the word within the block is selected for transfer to the processor. If the tags are not the same, the block containing the required word is first transferred to the cache.

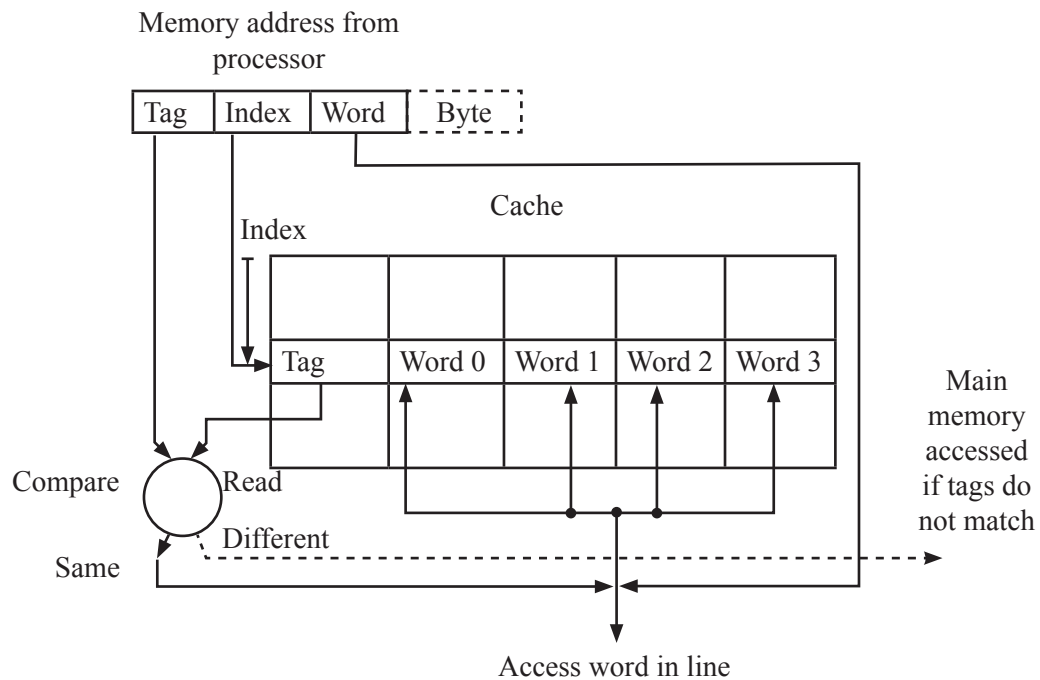


Fig. 4.11 Direct mapped cache with a multi-word block

In direct mapping, the corresponding blocks with the same index in the main memory will map into the same block in the cache, and hence only blocks with different indices can be in the cache at the same time. A replacement algorithm is unnecessary, since there is only one allowable location for each incoming block. Efficient replacement relies on the low probability of lines with the same index being required. However, there are such occurrences, for example, when two data vectors are stored starting at the same index and pairs of elements need to be processed together.

To gain the greatest performance, data arrays and vectors need to be stored in a manner which minimises the conflicts in processing pairs of elements. Fig. 4.11 shows the lower bits of the processor address used to address the cache location directly. It is possible to introduce a mapping function between the address index and the cache index so that they are not the same.

4.5.3 Set-associative Mapping

In the direct scheme, all words stored in the cache must have different indices. The tags may be the same or different. In the fully associative scheme, blocks can displace any other block and can be placed anywhere, but the cost of the fully associative memories operate relatively slow. Set-associative mapping allows a limited number of blocks, with the same index and different tags, in the cache and can therefore be considered as a compromise between a fully associative cache and a direct mapped cache.

The organisation is shown in Fig.4.12. The cache is divided into “sets” of blocks. A four-way set associative cache would have four blocks in each set. The number of blocks in a set is known as the associativity or set size. Each block in each set has a stored tag which, together with the index, completes the identification of the block. First, the index of the address from the processor is used to access the set. Then, comparators are used to compare all tags of the selected set with the incoming tag. If a match is found, the corresponding location is accessed, other wise, as before, an access to the main memory is made.

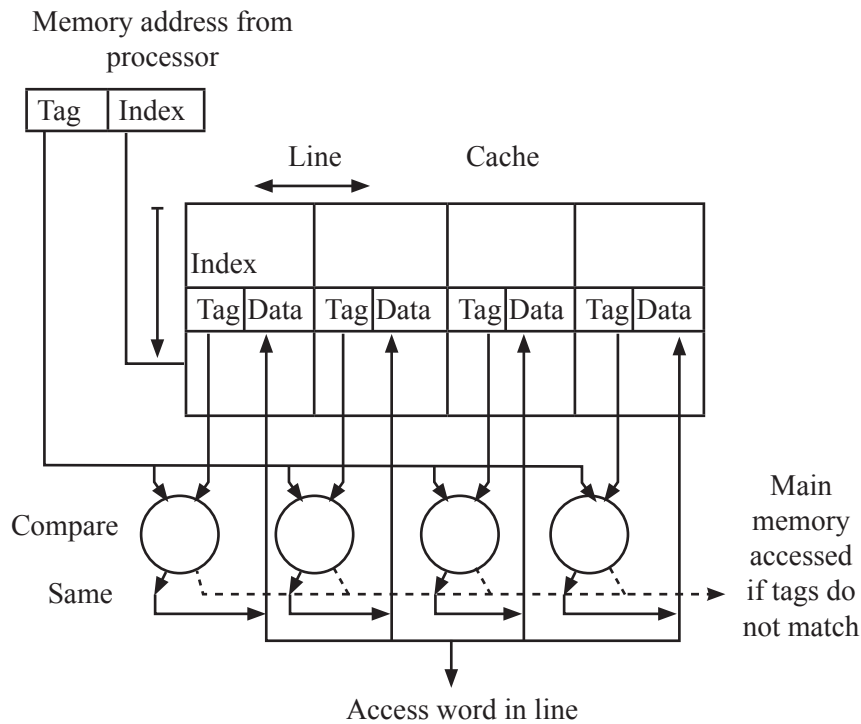


Fig. 4.12 Cache with set-associative mapping

The tag address bits are always chosen to be the most significant bits of the full address, the block address bits are the next significant bits and the word/byte address bits form the least significant bits as this spreads out consecutive main memory blocks throughout consecutive sets in the cache. This addressing format is known as bit selection and is used by all known systems.

In a set-associative cache it would be possible to have the set address bits as the most significant bits of the address and the block address bits as the next significant, with the word within the block as the least significant bits, or with the block address bits as the least significant bits and the word within the block as the middle bits. Notice that the association between the stored tags and the incoming tag is done using comparators and can be shared for each associative search, and all the information, tags and data, can be stored in ordinary random access memory.

The number of comparators required in the set-associative cache is given by the number of blocks in a set, not the number of blocks in all, as in a fully associative memory. The set can be selected quickly and all the blocks of the set can be read out simultaneously with the tags before waiting for the tag comparisons to be made. After a tag has been identified, the corresponding block can be selected. The replacement algorithm for set-associative mapping need only consider the lines in one set, as the choice of set is predetermined by the index in the address. Hence, with two blocks in each set, for example, only one additional bit is necessary in each set to identify the block to replace.

4.5.4 Sector Mapping

In sector mapping, the main memory and the cache are both divided into sectors; each sector is composed of a number of blocks. Some of the features of sector mapping are as follows.

- Any sector in the main memory can map into any sector in the cache and a tag is stored with each sector in the cache to identify the main memory sector address.
- However, a complete sector is not transferred to the cache or back to the main memory as one unit. Instead, individual blocks are transferred as required.
- On cache sector miss, the required block of the sector is transferred into a specific location within one sector.
- The sector location in the cache is selected and all the other existing blocks in the sector in the cache are from a previous sector.

- Sector mapping might be regarded as a fully associative mapping scheme with valid bits, as in some microprocessor caches.
- Each block in the fully associative mapped cache corresponds to a sector, and each byte corresponds to a “sector block”.

Summary

- The chapter explains the concept system bus and different functional groups within the bus.
- The classic concept of a bus is a set of boards plugged into a passive backplane.
- Bus taxonomy is characterised along a number of dimensions.
- In asynchronous busses, operations occur on specified edges of control signals, regardless of a master clock. Early busses tend to be asynchronous.
- The basic advantage of a multiplexed bus is fewer wires which in turn means fewer pins on each connector, fewer high-power driver circuits and so on.
- PCI is an interconnection system between a microprocessor and attached devices in which expansion slots are spaced closely for high speed operation.
- The PCI specification gives designers and integrators facilities for balancing and fine tuning systems for optimal performance, up to a great extent.

References

- Mathur, S., 2010. *Microprocessor 8085 And Its Interfacing*, PHI Learning Pvt. Ltd.
- Brey, 2008. *The Intel Microprocessors: 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro Processor, Pentium Ii, Pentium Iii, Pentium 4*, 8th ed. Pearson Education India.
- *Tutorial On Introduction to 8085 Architecture and Programming* [Pdf] Available at: <<http://www.phy.davidson.edu/fachome/dmb/py310/8085.pdf>> [Accessed 30 May 2013].
- *8085 Microprocessor* [Pdf] Available at: <http://www.nptel.iitm.ac.in/courses/Webcourse-contents/IISc-BANG/Microprocessors%20and%20Microcontrollers/pdf/Lecture_Notes/LNm1.pdf> [Accessed 30 May 2013].
- *System bus explanation* [Video online] Available at: <<http://www.youtube.com/watch?v=A71WOpA6IeU>> [Accessed 30 May 2013].
- *8086 Microprocessor Architecture: Tutorial 4 On 8086 Architecture* [Video online] Available at: <<http://www.youtube.com/watch?v=WpOJDw6C0Mk>> [Accessed 30 May 2013].

Recommended Reading

- Nagoorkani, 2013. *8085 Microprocessors & Its Application*, 3rd ed. Tata McGraw-Hill Education.
- Mathivanan, N., 2003. *MICROPROCESSORS, PC HARDWARE AND INTERFACING*, PHI Learning Pvt. Ltd.
- Godse, D. A. & Godse, A. P., 2010. *Microprocessor & Microcontroller*, Technical Publications.

Self Assessment

1. A computer bus consists of a set of _____ “wires” attached to several connectors into which peripheral boards may be plugged.
 - a. parallel
 - b. overlapping
 - c. transacting
 - d. intersecting
2. _____ specifies the peripheral and register within the peripheral that is being accessed.
 - a. Address
 - b. Data
 - c. Control
 - d. PCI
3. _____ is the information being transferred to or from the peripheral.
 - a. Address
 - b. Data
 - c. Control
 - d. PCI
4. _____ are signals that effect the data transfer operation. It is the control signals and how they are manipulated that embodies the bus protocol.
 - a. Address
 - b. Data
 - c. Control
 - d. PCI
5. Contemporary busses are typically either _____ wide for both address and data.
 - a. 256 or 512 bits
 - b. 128 or 256 bits
 - c. 64 or 128 bits
 - d. 32 or 64 bits
6. Which of the following statements is true?
 - a. Early busses could tolerate low-power, relatively fast driver circuits and could thus support a large number of attached devices.
 - b. Early busses could tolerate high-power, relatively fast driver circuits and could thus support a few number of attached devices.
 - c. Early busses could tolerate low-power, relatively slow driver circuits and could thus support a large number of attached devices.
 - d. Early busses could tolerate high-power, relatively slow driver circuits and could thus support a large number of attached devices.

7. Which of the following statements is true?
- The basic advantage of a multiplexed bus is fewer wires which in turn means fewer pins on each connector, fewer high-power driver circuits and so on.
 - The basic disadvantage of a multiplexed bus is fewer wires which in turn means fewer pins on each connector, fewer high-power driver circuits and so on.
 - The basic advantage of a multiplexed bus is more wires which in turn means fewer pins on each connector, fewer high-power driver circuits and so on.
 - The basic advantage of a multiplexed bus is fewer wires which in turn means more pins on each connector, more high-power driver circuits and so on.
8. Which of the following statements is true?
- A fully associative cache requires the cache to be composed of associative memory holding the memory address for each cached line.
 - A fully associative cache requires the cache to be composed of associative memory holding both the memory address and the control for each cached line.
 - A fully associative cache requires the cache to be composed of associative memory holding both the memory control and the data for each cached line.
 - A fully associative cache requires the cache to be composed of associative memory holding both the memory address and the data for each cached line.
9. Which of the following statements is true?
- In set associative mapping, the cache consists of normal high speed random access memory, and each location in the cache holds the data, at an address in the cache given by the lower significant bits of the main memory address.
 - In fully associative mapping, the cache consists of normal high speed random access memory, and each location in the cache holds the data, at an address in the cache given by the lower significant bits of the main memory address.
 - In sector mapping, the cache consists of normal high speed random access memory, and each location in the cache holds the data, at an address in the cache given by the lower significant bits of the main memory address.
 - In direct mapping, the cache consists of normal high speed random access memory, and each location in the cache holds the data, at an address in the cache given by the lower significant bits of the main memory address.
10. Which of the following statements is true?
- In sector mapping, the main memory and the cache are both divided into sectors; each sector is composed of a number of blocks.
 - In sector mapping, the main memory and the cache are both divided into sectors; each sector is composed of a number of blocks.
 - In direct mapping, the main memory and the cache are both divided into sectors; each sector is composed of a number of blocks.
 - In fully associative mapping, the main memory and the cache are both divided into sectors; each sector is composed of a number of blocks.

Chapter V

Input / Output Modules

Aim

The aim of the chapter is to:

- explain the concept of Input/output module
- elucidate the structure of Input/output module
- introduce interrupt driven Input/output

Objectives

The objectives of this chapter are to:

- define input/ output modules
- elucidate the structure of input/ output modules
- explain the concept of direct memory access

Learning outcome

At the end of this chapter, you will be able to:

- identify Input/ output modules
- understand interrupt driven modules
- describe the functions of Input/ output modules

5.1 Introduction

The input/output module (I/O module) is normally connected to the system Bus of the system on one end and one or more input /Output devices on the other. Normally an I/O module can control one or more peripheral devices. It performs additional functions such as communicating with the CPU and with the peripherals.

5.2 Functions of I/O Module

An I/O module is a mediator between the processor and I/O devices. It controls the data exchange between the external devices and main memory, or external devices and CPU registers. The functional requirements of an I/O module are discussed below.

Ability to provide control and timing signals

- The need of I/O from various I/O devices by the CPU is quite unpredictable. In fact, it depends on I/O needs of particular programs and normally does not follow any pattern.
- Since the I/O module also shares system Bus and memory for data Input/output, control and timing are needed to coordinate the flow of data from external devices to CPU or memory.
- A typical control cycle for transfer of data from I/O device to CPU is:
- Enquire status of an attached device from I/O module. The status can be “busy”, “ready” or “out of order”.
 - I/O module responds back with the status of the device.
 - If device is ready, CPU commands I/O module to transfer data from the I/O device.
 - I/O module accepts data from the I/O device.
 - The data is then transferred from I/O module to the CPU.

Communication with the CPU

- The example given above clearly specifies the need of communication between the CPU and I/O module. This communication can be: commands such as read sector, write sector, seek track number (which are issued by CPU to I/O module); or data (which may be required by the CPU or transferred out); or status information such as “busy”, “ready” or any error condition from I/O modules.
- The status recognition is necessary because of the speed gap between the CPU and I/O device. An I/O device might be busy in doing the I/O of previous instruction when it is needed again.
- Another important communication from the CPU is the unique address of the peripheral from which I/O is expected or is to be controlled.

Communication with the I/O device

- Communication between I/O module and I/O device is needed to complete the I/O operation. This communication involves commands, status or data.

Provision for data buffering

- Data buffering is quite useful for the purpose of smoothing out the gaps in speed of CPU and I/O devices. The data buffers are registers, which hold the I/O information temporarily.
- The I/O is performed in short bursts, in which data is stored in buffer area while the device can take its own time to accept it. I/O device to CPU transfer, data is first transferred to the buffer and then passed on to CPU from these buffer registers.
- Thus, the I/O operation does not tie up the bus for the slower I/O devices.

In-built error detection mechanism

- The error detection mechanism may involve checking the mechanical as well as data communication errors. These errors should be reported to the CPU.

- The examples of the kind of mechanical errors that can occur in devices are paper jam in printer, mechanical failure, electrical failure etc.
- The data communication errors may be checked by using parity bit.

5.3 Structure of I/O Module

General characteristics of the structure of I/O module:

- There is a need of I/O logic, which should interpret and execute the dialogue between the CPU and I/O module. Therefore, control lines are needed between CPU and this I/O module. In addition, the address lines are needed to recognise the address of the I/O module and its specific device.
- The data lines connecting I/O module to system bus must exist. These lines serve the purpose of data transfer.
- Data registers may act as buffer between CPU and I/O module.
- The interface of I/O module with the device should have interface logic to control the device, to get the status information and transfer of data.
- The figure below is a typical I/O module, which in addition to the above features, has status/control registers which might be used to pass on the status information or can store control information.

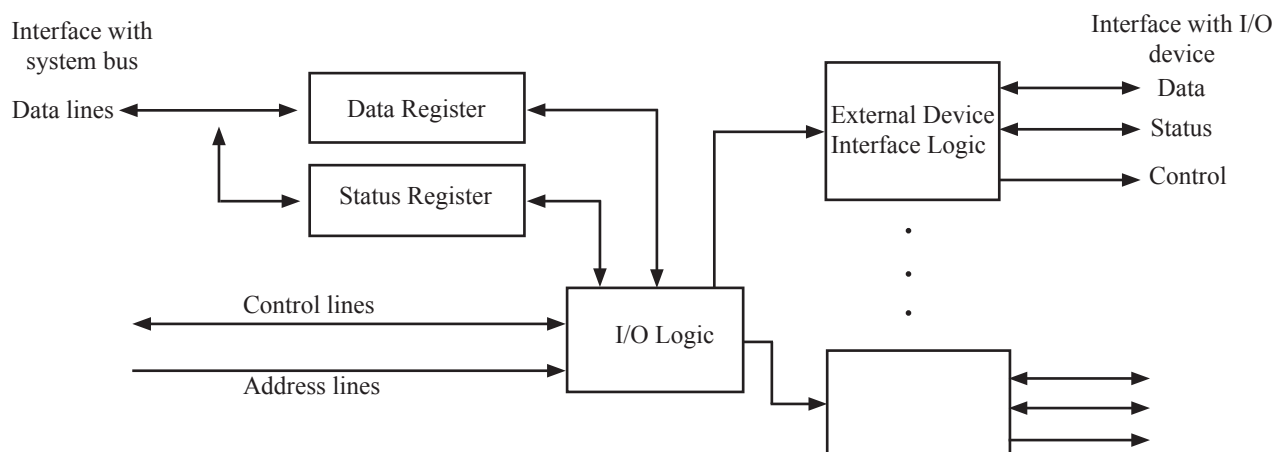


Fig. 5.1 General I/O module

- The data is stored or buffered in data registers. The status registers are used to indicate the current state of a device. For example, the device is busy, the system BUS is busy, the device is out of order etc.
- If an I/O module takes more processing burden, then it is termed as I/O channel or processor. The primitive I/O modules, which require detailed control by processor is called I/O controller or device controller.
- These I/O controllers are normally used in micro-computers while I/O processors are mainly used for mainframes because the I/O work for micro-computer is normally limited to single user's job, therefore, we do not expect a very huge amount of I/O to justify the investment in I/O processors, which are expensive.
- Once the general structure for an I/O module is defined, the next subject is I/O operations.

5.4 Programmed Input/ Output

Programmed Input/Output is useful I/O method for computers where hardware costs need to be minimised. The input or output operation in such cases may involve:

- Transfer of data from I/O device to the CPU registers.
- Transfer of data from CPU registers to memory.

In addition, in a programmed I/O method, the responsibility of CPU is to constantly check the status of the I/O device, to check whether it has become free (when output is desired) or it has finished inputting the current series

of data (when input is going on).

Thus, Programmed I/O is a very time consuming method, as CPU takes a lot of time for checking and verifying the status of an I/O device.

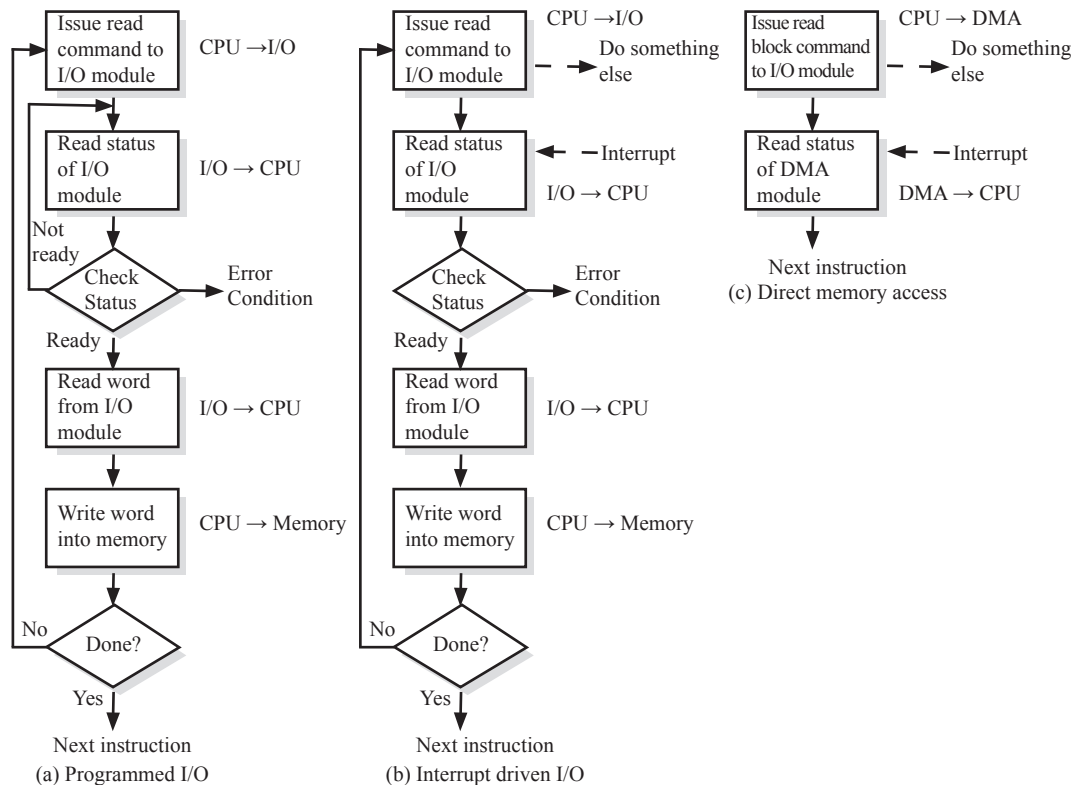


Fig. 5.2 Three techniques for an input of a block of data

I/O Instructions

To carry out Input/Output, CPU issues I/O related instruction. These instructions consist of two components:

- the address of the Input/Output device specifying the I/O device and I/O module
- an Input/Output command

Four types of I/O Commands

- **Control command:** Control commands are device specific and are used to control the specific instructions to the device. For example, a magnetic tape requires rewinding or moving forward by a block.
- **Test command:** It checks the status, such as, if a device is ready or not or is in error condition.
- **Read command:** It is used for input of data from input device.
- **Write command:** It is used for output of data to output device.

The other part of I/O instruction is the address of the I/O device. In systems with programmed I/O the I/O module, the main memory and the CPU normally share the system bus. Thus, each I/O module should interpret the address lines to determine if the command is for itself. That is, the way CPU specifies which device to access. There are two methods of doing so. These are called memory mapped I/O and I/O-mapped I/O.

If we use the single address space for memory locations and I/O devices, i.e., the CPU treats the status and data registers of I/O module as memory locations, then memory and I/O devices can be accessed using the same instructions. This is referred to as memory mapped I/O. For a memory mapped I/O, only a single read and a single write line are needed for memory or I/O module read or write operations. These lines are activated by CPU for either memory access or I/O device access. Fig. 5.3 shows the memory mapped I/O system structure. This scheme is used in Motorola 68000.

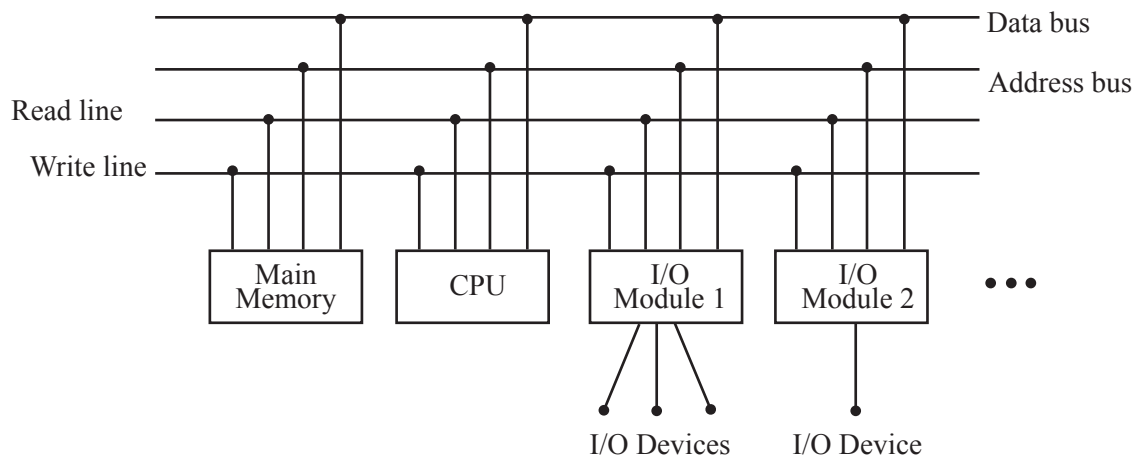


Fig. 5.3 Structure of memory mapped I/O

In I/O-mapped I/O, the I/O devices and memory are addressed separately (refer to fig. 5.4). There are separate control lines for memory and I/O device read or write operations, thus, a memory reference instruction does not affect an I/O device. Here, separate Input/ Output instructions are needed which cause data transfer between addressed I/O module and CPU. This structure is used in Intel 8085 and 8686 series.

5.5 Interrupt Driven Input/ Output

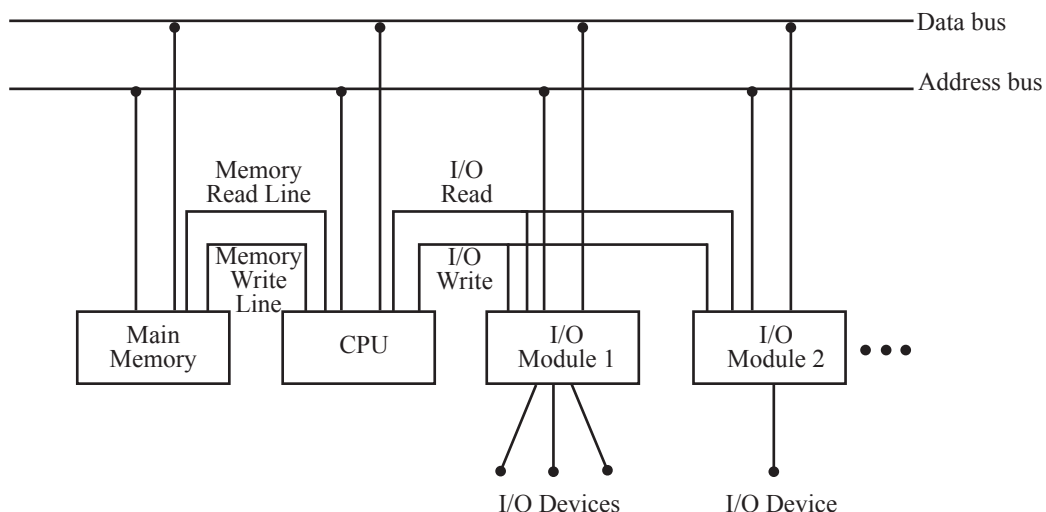


Fig. 5.4 I/O-mapped I/O

The speed of I/O devices is much slower in comparison to that of CPU and because the CPU has to repeatedly check whether a device is free; or wait till the completion of I/O, therefore, the performance of CPU in programmed I/O goes down tremendously. A well-designed mechanism was conceived for this, which is referred to as Interrupt driven I/O. In this mechanism, provisions of interruption of CPU work, once I/O device has finished the I/O or when it is ready for the I/O, has been provided.

The interrupt driven I/O mechanism for transferring a block of data is shown in Fig. 5.2. After issuing a READ command (for input) the CPU goes off to do other useful work (it may be an execution of a different program), while I/O module proceeds for reading of data from associated device. CPU saves the important register and processor status of the executing program in a stack and request I/O device to provide its data, which is placed on data bus by I/O device. After taking the required action with the data, the CPU can go back to the program it was executing before the interrupt.

5.5.1 Interrupt

The term “interrupt” is used for any exceptional event that causes temporary transfer of control of CPU from one program to the other, which causes obstruction. Interrupts are primarily issued on:

- initiation of Input/Output operation
- completion of an Input/Output operation
- occurrence of hardware or software errors

Interrupts can be generated by various sources internal/external to the CPU. An interrupt generated internally by CPU is sometimes termed as “Traps”. The Traps are normally results of programming errors such as division by zero while execution of a program.

The two key issues in Interrupt driven Input/Output are:

- to determine the device which has issued an interrupt
- in case of occurrence of multiple interrupts which are to be processed first

There are several solutions to these problems. The simplest of them is to provide multiple interrupt lines, which will result in immediate recognition of the interrupting device. The priorities can be assigned to various interrupts and the interrupt with highest priority should be selected for service in case multiple interrupt occurs. But providing multiple interrupt lines is an impractical approach because only a few lines of the system bus can be devoted for the interrupt. Other methods for this are software poll, daisy chaining etc.

5.5.2 Software Poll

On occurrence of an interrupt, CPU starts executing software; routine termed as interrupt service program or routine which poll to each I/O module to determine which I/O module has caused the interrupt. This may be achieved by reading the status register of the I/O modules. The priority here can be implemented easily by defining the polling sequence, since the device polled first will have higher priority. After identifying the device, the next set of instructions to be executed will be the device service routines of that device, resulting in the desired input or output. As far as daisy chaining is concerned, we have one interrupt acknowledge line, which is chained through various interrupt devices. There is just one interrupt request line.

On receiving an interrupt request the interrupt acknowledge line is activated, which in turn passes this signal, device by device. The first device which has made the interrupt request thus grasps the signal and responds by putting a word which is normally an address of interrupt servicing program or a unique identifier on the data lines. This word is also referred to as interrupt vector. This address or identifier in turn is used for selecting an appropriate interrupt-servicing program. The daisy chaining has an in-built priority scheme, which is determined by the sequence of devices on interrupt acknowledge line.

In bus arbitration technique, the I/O module first needs to control the bus and only after that it can request for an interrupt. In this scheme, since only one of the modules can control the bus, therefore, only one request can be made at a time. The interrupt request is acknowledged by the CPU. On its response, I/O module places the interrupt vector on the data lines. An interrupt vector normally contains the address of interrupt serving program.

5.6 Direct Memory Access (DMA)

When large amount of data is to be transferred from CPU, a DMA module can be used. In both, Interrupt driven and programs I/O, the CPU is tied up in executing input/output instructions while DMA acts as if it has taken over control from the CPU.

Direct memory access (DMA) is a feature of modern computers and microprocessors that allow certain hardware subsystems within the computer to access system memory for reading and/or writing independently of the central processing unit.

Many hardware systems use DMA including disk drive controllers, graphics cards, network cards and sound cards. DMA is also used for intra-chip data transfer in multi-core processors, especially in multiprocessor system-on-chips,

where its processing element is equipped with a local memory (often called scratchpad memory) and DMA is used for transferring data between the local memory and the main memory.

Computers that have DMA channels can transfer data to and from devices with much less CPU overhead than computers without a DMA channel. Similarly, a processing element inside a multi-core processor can transfer data to and from its local memory without occupying its processor time and allowing computation and data transfer concurrency.

Without DMA, using programmed input/output (PIO) mode for communication with peripheral devices, or load/store instructions in the case of multicore chips, the CPU is typically fully occupied for the entire duration of the read or write operation, and is thus unavailable to perform other work. With DMA, the CPU would initiate the transfer, do other operations while the transfer is in progress, and receive an interrupt from the DMA controller once the operation has been done. This is especially useful in real-time computing applications where not stalling behind concurrent operations is critical. Another and related application area is various forms of stream processing where it is essential to have data processing and transfer in parallel, in order to achieve sufficient throughput.

Summary

- The I/O module acts as a mediator between the processor and the I/O device.
- An I/O module provides an interface internal to the computer which connects it to the CPU and main memory and an interface external to the computer connecting it to external device or peripheral.
- Components of I/O module are data registers, control lines, address lines etc.
- The interface of I/O module with the device should have interface logic to control the device, to get the status information and transfer of data.
- A programmed input/output is useful I/O method for computers where hardware costs need to be minimised.
- In a programmed I/O method, the responsibility of CPU is to constantly check the status of the I/O device, to check whether it has become free or it has finished inputting the current series of data.
- Interrupt driven I/O device is a well-designed mechanism which overcomes the limitation of the slow working of I/O device.
- The direct memory access (DMA), a feature of microprocessor allows certain hardware subsystems within the computer to access system memory for reading and/or writing independently of the central processing unit.

References

- Jones, T. C., 2006. *Programmable Logic Controllers: The Complete Guide to the Technology*, Brilliant-Training.
- Godse, D. A. & Godse, A. P., 2008. *Microprocessor Techniques*, Technical Publications.
- *Input/Output Module* [Pdf] Available at: <<http://www.docs.is.ed.ac.uk/skills/documents/3663/SPSSInput-OutputModule.pdf>> [Accessed 30 May 2013].
- *I/O Modules* [Pdf] Available at: <http://shpat.com/docs/grayhill/io_modules_individual.pdf> [Accessed 30 May 2013].
- Prof. Raman, S., *Lecture 25 - Interrupt Driven I/O* [Video online] Available at: <<http://www.youtube.com/watch?v=C02weCM9yWA>> [Accessed 30 May 2013].
- Prof. Kumar, A., *Lecture - 33 Input / Output Subsystem: Introduction* [Video online] Available at: <<http://www.youtube.com/watch?v=0XybwAbup-w>> [Accessed 30 May 2013].

Recommended Reading

- Lipovski, J. G., 1999. *Single and Multi-Chip Microcontroller Interfacing: For the Motorola 6812*, Academic Press.
- El-Abiad, H. A., 2006. *Adv Microprocessors & Periph 2E*, 2nd ed. Tata McGraw-Hill Education.
- Wadhwa, A., 2010. *Microprocessor 8085: Architecture Programming And Interfacing*, PHI Learning Pvt. Ltd.

Self Assessment

1. An I/O module is a mediator between the _____ and I/O device devices.
 - a. processor
 - b. DMA
 - c. hard disk
 - d. ROM
2. The I/O module should not only communicate the information from _____ to I/O device, but it should also coordinate these two.
 - a. DMA
 - b. hard disk
 - c. CPU
 - d. ROM
3. _____ may act as buffer between CPU and I/O module.
 - a. DMA
 - b. Hard disk
 - c. Data registers
 - d. ROM
4. _____ commands are device specific and are used to control the specific instructions to the device.
 - a. Control
 - b. Read
 - c. Write
 - d. Test
5. In bus arbitration technique, the _____ first need to control the bus and only after that it can request for an interrupt.
 - a. DMA
 - b. hard disk
 - c. data registers
 - d. I/O module
6. Which of the following statements is true?
 - a. Programmed data registers are useful I/O methods for computers where hardware costs need to be minimised.
 - b. Programmed input/output is a useful I/O method for computers where software costs need to be minimised.
 - c. Programmed input/output is a useful I/O method for computers where hardware costs need to be maximised.
 - d. Programmed input/output is a useful I/O method for computers where hardware costs need to be minimised.

7. Which of the following statements is true?
 - a. The interrupt request is acknowledged by the CPU. On its response, I/O module places the interrupt vector on the data lines.
 - b. The interrupt request is acknowledged by the CPU. On its response, I/O module places the interrupt vector on the address lines.
 - c. The interrupt request is acknowledged by the DMA. On its response, I/O module places the interrupt vector on the data lines.
 - d. The interrupt request is acknowledged by the CPU. On its response, I/O module places the interrupt vector on the status registers.
8. Which of the following statements is true?
 - a. When small amount of data is to be transferred from CPU, a DMA module can be used.
 - b. When small amount of data is to be transferred from data register, a DMA module can be used.
 - c. When large amount of data is to be transferred from CPU, a DMA module can be used.
 - d. When large amount of data is to be transferred from hard disk, a DMA module can be used.
9. Which of the following statements is true?
 - a. "Write" command checks the status such as, if a device is ready or not or is in error condition.
 - b. "Test" command checks the status such as, if a device is ready or not or is in error condition.
 - c. "Read" command checks the status such as, if a device is ready or not or is in error condition.
 - d. DMA command checks the status such as, if a device is ready or not or is in error condition.
10. Which of the following statements is true?
 - a. The "Read" command is used for input of data from input device and Test command is used for output of data to output device.
 - b. The "Test" command is used for input of data from input device and Write command is used for output of data to output device.
 - c. The "Write" command is used for input of data from input device and Read command is used for output of data to output device.
 - d. The "Read" command is used for input of data from input device and Write command is used for output of data to output device.

Chapter VI

Operating System Support

Aim

The aim of the chapter is to:

- explain the concept of operating system support
- introduce the types of operating system
- elucidate scheduling in operating system

Objectives

The objectives of this chapter are to:

- enlist the types of operating system
- elucidate memory management
- explain the concept of scheduling

Learning outcome

At the end of this chapter, you will be able to:

- identify various types of operating system
- describe the concept of scheduling
- recognise virtual memory

6.1 Introduction

The primary need for the operating system arises from the fact that user needs to be provided with services and operating system. The central part of a computer system is a processing engine called CPU. A system should make it possible for a user's application to use the processing unit. A user application would need to store information. The OS makes memory available to an application when required. Similarly, user applications need the use of input facility to communicate with the application. This is often in the form of a keyboard, or a mouse or even a joystick.

The output is usually provided by a video monitor or a printer as sometimes the user may wish to generate an output in the form of a printed document. Output may be available in some other forms.

6.2 Types of Operating System

Modern computer operating systems may be classified into three groups, which are distinguished by the nature of interaction that takes place between the computer user and his or her program during its processing. The types of operating systems are described below:

6.2.1 Batch Processing Operating System

In a batch processing operating system environment, users submit jobs to a central place where these jobs are collected into a batch, and subsequently placed on an input queue at the computer where they will be run. In this case, the user has no interaction with the job during its processing, and the computer's response time is the turnaround time: the time from submission of the job until execution is complete, and the results are ready for return to the person who submitted the job.

6.2.2 Time Sharing

Another mode for delivering computing services is provided by time sharing operating systems. In this environment, a computer provides computing services to several or many users concurrently on-line. Here, the various users are sharing the central processor, the memory and other resources of the computer system in a manner facilitated, controlled and monitored by the operating system. The user, in this environment, has nearly full interaction with the program during its execution, and the computer's response time may be expected to be no more than a few seconds.

Time-sharing is the sharing of a computing resource among many users by means of multiprogramming and multi-tasking. Its introduction in the 1960s and emergence as the prominent model of computing in the 1970s, represents a major technological shift in the history of computing. By allowing a large number of users to interact concurrently with a single computer, time-sharing dramatically lowered the cost of providing computing capability, which makes possible for individuals and organisations to use a computer without owning one, and promoted the interactive use of computers and the development of new interactive applications.

6.2.3 Real Time Operating System (RTOS)

The third type is the real time operating systems, which are designed to service those applications where response time is of great importance in order to prevent error, misrepresentation or even a disaster. Examples of real time operating systems are those which handle airlines reservations, machine tool control, and monitoring of a nuclear power station. The systems, in this case, are designed to be interrupted by external signals that require the immediate attention of the computer system. These real time operating systems are used to control machinery, scientific instruments and industrial systems.

An RTOS typically has very little user-interface capability and no end-user utilities. A very important part of an RTOS is managing the resources of the computer so that a particular operation executes in precisely the same amount of time every time it occurs. In a complex machine, having a part move more quickly just because system resources are available, may be just as catastrophic as having it not move at all because the system is busy.

6.2.4 Multiprogramming Operating System

Computer multiprogramming is the allocation of a computer system and its resources to more than one concurrent application, job or user (“program” in this nomenclature). Initially, this technology was sought in order to optimise use of a computer system, since time and processing resources were often wasted when a single job waited for human interaction or other data input/output operations. Multiprogramming capability was developed as a feature of operating systems in the late 1950s and came into common use in mainframe computing in the mid- to late 1960s. This followed the development of hardware systems that possessed the requisite circuit logic and instruction sets to facilitate the transfer of control between the operating system and one or more independent applications, users or job streams.

The use of multiprogramming was enhanced by the arrival of virtual memory and virtual machine technology, which enabled individual programs to make use of memory and operating system resources, as if other concurrently running programs were, for all practical purposes, non-existent and invisible to them.

Multiprogramming should be differentiated from multi-tasking since not all multiprogramming entails or has the capability for “true” multi-tasking. This is the case even though the use of multi-tasking generally implies the use of some multiprogramming methods.

- In this context, the root word “program” does not necessarily refer to a compiled application. It is, rather, any set of commands submitted for execution by a user or an operator. Such could include a script or job control stream and any included calls to macro-instructions, system utilities or application program modules.
- An entire, interactive, logged-in user session can be thought of as a “program” in this sense.
- A program generally comprises of numerous tasks, a task being a relatively small group of processor instructions which together achieve a definable logical step in the completion of a job or the execution of a continuous-running application program.
- A task frequently ends with some request requiring the moving of data, a convenient opportunity to allow another program to have system resources, particularly CPU time.
- In multiprogramming, concurrent running (sharing of the processor) is achieved when the operating system identifies opportunities to interrupt the handling of one program between tasks (e.g., when it is waiting for input/output) and to transfer process control to another program (application, job or user).
- To a great extent, the ability of a system to share its resources equitably or according to certain priorities is dependent upon the design of the programs being handled and how frequently they may be interrupted.
- Multi-tasking eliminates that dependency and expands upon multiprogramming by enabling the operating system supervisor to interrupt programs in the middle of tasks and to transfer processor control so rapidly that each program is now assured a portion of each processing second, making the interruptions imperceptible to most human-interactive applications.

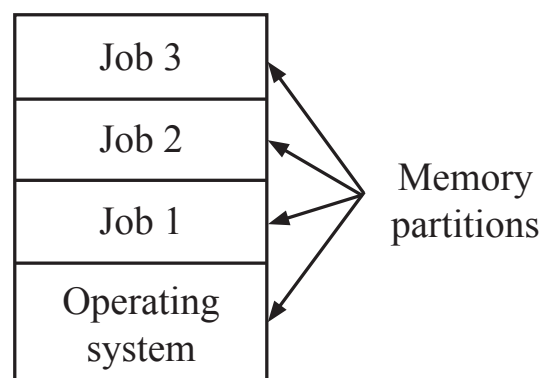


Fig. 6.1 Multiprogramming operating system

6.2.5 Multiprocessing System

A multiprocessing system is a computer hardware configuration that includes more than one independent processing unit. The term multiprocessing is generally used to refer to large computer hardware complexes found in major scientific or commercial applications.

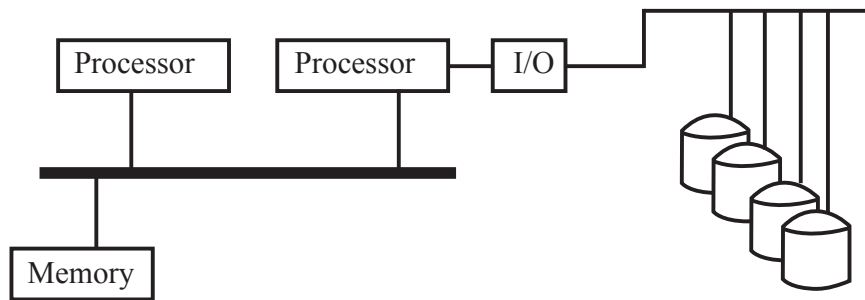


Fig. 6.2 Multiprocessing system

6.2.6 Networking Operating System

- A networked computing system is a collection of physical interconnected computers. The operating system of each of the interconnected computers must contain, in addition to its own stand-alone functionality, provisions for handling communication and transfer of program and data among the other computers with which it is connected.
- Network operating systems are not fundamentally different from single processor operating systems.
- They obviously need a network interface controller and some low-level software to drive it, as well as programs to achieve remote login and remote files access, but these additions do not change the essential structure of the operating systems.

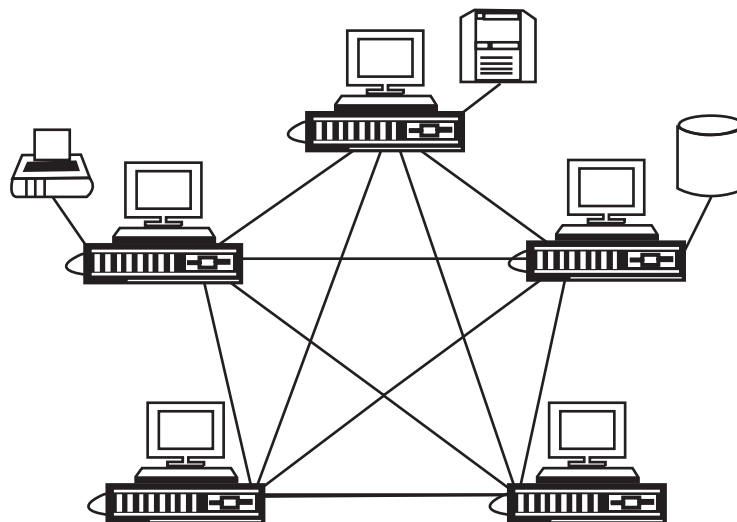


Fig. 6.3 Networking Operating System

6.2.7 Distributed operating system

A distributed computing system consists of a number of computers that are connected and managed, so that they automatically share the job processing load among the constituent computers, or separate the job load as appropriate particularly configured processors. Such a system requires an operating system which, in addition to the typical stand-alone functionality, provides coordination of the operations and information flow among the component computers.

The networked and distributed computing environments and their respective operating systems are designed with more complex functional capabilities. In a network operating system, the users are aware of the existence of multiple computers, and can log in to remote machines and copy files from one machine to another. Each machine runs its own local operating system and has its own user (or users).

A distributed operating system, in contrast, is the one that appears to its users as a traditional uni-processor system, even though it is actually composed of multiple processors. In a true distributed system, users should not be aware of where their programs are being run or where their files are located; that should all be handled automatically and efficiently by the operating system.

True distributed operating systems require more than just adding a little code to a uni-processor operating system, because distributed and centralised systems differ in critical ways. Distributed systems, for example, often allow program to run on several processors at the same time, thus requiring more complex processor scheduling algorithms in order to optimise the amount of parallelism achieved.

6.2.8 Operating Systems for Embedded Devices

As embedded systems (PDAs, cell phones, point-of-sale devices, VCR's, industrial robot control, or even your toaster) become more complex hardware-wise with every generation, and more features are put into them day-by-day, applications they run require more and more to run on actual operating system code in order to keep the development time reasonable. Some of the popular OS are:

- Nexus's Conix - An embedded operating system for ARM processors.
- Sun's Java OS - A standalone virtual machine not running on top of any other OS; mainly targeted at embedded systems.
- Palm Computing's Palm OS - Currently the leader OS for PDAs, has many applications and supporting companies.
- Microsoft's Windows CE and Windows NT Embedded OS

6.3 Scheduling

Operating system scheduling is the process of controlling and prioritising messages sent to a processor. An internal operating system program, called the scheduler, performs this task. The goal of operating system scheduling is maintaining a constant amount of work for the processor, eliminating highs and lows in the processor's workload and making sure each process is completed within a reasonable time frame.

While operating system scheduling is important to all systems, it is especially important in a real-time system. Since nearly every operation on a computer has at least a small amount of processor time involved, the processor can be a major source of slowdowns and bottlenecks. In order to alleviate the strain on the processor, and make sure tasks are completed in a timely manner, most operating systems use some form of task scheduling. The operating system scheduling process varies based on the system, but they tend to fall within familiar categories.

Operating system scheduling is typically broken down into three parts: long-, mid- and short-term scheduling. Not every operating system fully utilises each type midterm and long-term are often combined but they will use some combination of them.

- Each type of scheduling provides a slightly different benefit to the system.
- Long-term scheduling revolves around admitting the programs to the scheduling process.
- When a new program initiates, the long-term scheduler determines if there is enough space for the new entrant. If there isn't, then the long-term scheduler delays the activation of the program until there is enough room.
- The midterm scheduler decides which processes have been idle and which are active. It leaves the active processes alone and writes idle ones to the hard drive. This frees up memory for other programs to come in through the long-term scheduler.
- When the mid- and long-term schedulers are combined, instead of delaying activation of a new process the scheduler simply swaps it into storage.

- The short-term scheduler is the part that works directly with the processor. This portion activates processes, sets priorities and oversees the processor's workload. The short-term scheduler is constantly trying to anticipate computer needs to keep the processor running smoothly.
- In most circumstances, operating system scheduling is a way of making a computer function more efficiently, but in a real-time operating system, it is vital to its purpose. A real-time system needs to execute processes within a set time.
- If these processes lag, then their purpose is lost. These important programs require very specific system scheduling in order to make sure information and responses.

6.4 Virtual Memory

The purpose of virtual memory is to enlarge the address space, the set of addresses a program can utilise. For example, virtual memory might contain twice as many addresses as main memory. A program using all of virtual memory, therefore, would not be able to fit in main memory all at once. Nevertheless, the computer could execute such a program by copying those portions of the program into the main memory, which are needed at any given point during execution. To facilitate copying virtual memory into real memory, the operating system divides virtual memory into pages, each of which contains a fixed number of addresses. Each page is stored on a disk until it is needed.

When the page is needed, the operating system copies it from disk to main memory, translating the virtual addresses into real addresses. Addresses generated by programs are virtual addresses. The actual memory cells have physical addresses. A piece of hardware called a memory management unit (MMU) translates virtual addresses to physical addresses at run-time. The process of translating virtual addresses into real addresses is called mapping. The copying of virtual pages from disk to main memory is known as paging or swapping.

Some physical memory is used to keep a list of references to the most recently accessed information on an I/O (input/output) device, such as the hard disk. The optimisation it provides is that it is faster to read the information from physical memory than use the relevant I/O channel to get that information. This is called caching. It is implemented inside the OS.

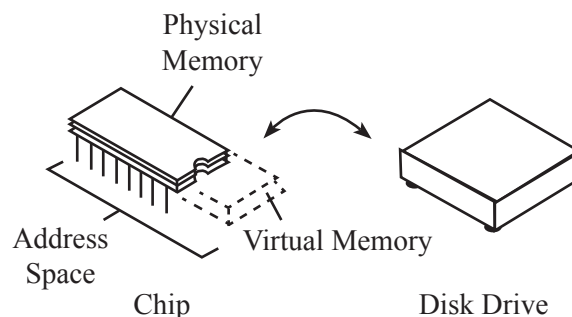


Fig. 6.4 Virtual memory

6.5 Memory Management

Memory is central to the operation of a modern computer system. Memory is a large array of words or bytes, each location with its own address. Interaction is achieved through a sequence of reads/writes of specific memory address. The CPU fetches from the program from the hard disk and stores in memory. If a program is to be executed, it must be mapped to absolute addresses and loaded into memory. In a multiprogramming environment, in order to improve both, the CPU utilisation and the speed of the computer's response, several processes must be kept in memory. There are many different algorithms depending on the particular situation to manage the memory. Selection of a memory management scheme for a specific system depends upon many factors, but especially upon the hardware design of the system. Each algorithm requires its own hardware support. The Operating System is responsible for the following activities in connection with memory management:

- Keep track of which parts of memory are currently being used and by whom.
- Decide which processes are to be loaded into memory when memory space becomes available.
- Allocate and reallocate memory space as needed.

In the multiprogramming environment, operating system dynamically allocates memory to multiple processes. Thus, memory plays a significant role in the important aspects of computer system like performance, S/W support, reliability and stability.

Memory can be broadly classified into two categories:

- the primary memory (like cache and RAM) and
- the secondary memory (like magnetic tape, disk etc.).

The memory is a resource that needs effective and efficient management. The part of OS that perform this vital task of memory management is known as memory manager.

In multiprogramming system, as the available memory is shared among number of processes, the allocation speed and the efficient memory utilisation (in terms of minimal overheads and reuse/relocation of released memory block) are of prime concern. Protection is difficult to achieve with relocation requirement, as location of process and absolute address in memory is unpredictable. But at run-time, it can be done. Fortunately, we have mechanisms supporting protection like processor (hardware) support that is able to abort the instructions violating protection and trying to interrupt other processes. Memory management systems on multi-tasking operating systems usually deal with the issues that are discussed briefly below.

6.5.1 Relocation

In systems with virtual memory, programs in memory must be able to reside in different parts of the memory at different times. This is because, when the program is swapped back into memory after being swapped out for a while, it can not always be placed in the same location. The virtual memory management unit must also deal with concurrency. Memory management in the operating system should therefore be able to relocate programs in memory and handle memory references and addresses in the code of the program so that they always point to the right location in memory.

6.5.2 Protection

Processes should not be able to refer the memory for another process without permission. This is called memory protection, and prevents malicious or malfunctioning code in one program from interfering with the operation of other running programs.

6.5.3 Sharing

Even though the memory for different processes is normally protected from each other, different processes sometimes need to be able to share information and therefore access the same part of memory. Shared memory is one of the fastest techniques for Inter-process communication.

6.5.4 Logical Organisation

Programs are often organised in modules. Some of these modules could be shared between different programs, some are read only and some contain data that can be modified. The memory management is responsible for handling this logical organisation that is different from the physical linear address space. One way to arrange this organisation is segmentation.

6.5.5 Physical Organisation

Memory is usually divided into fast primary storage and slow secondary storage. Memory management in the operating system handles moving information between these two levels of memory.

Summary

- The primary need for the operating system arises from the fact that user needs to be provided with services and operating system.
- The operating system makes memory available to an application when required.
- In a batch processing operating system environment, users submit jobs to a central place where these jobs are collected into a batch and placed on an input queue at the computer where they will be run.
- In time sharing operating systems, a computer provides computing services to several or many users concurrently on-line.
- The third type is the real time operating systems, which are designed to service those applications where response time is of the essence in order to prevent error, misrepresentation or even disaster.
- Computer multiprogramming is the allocation of a computer system and its resources to more than one concurrent application, job or user.
- A multiprocessing system is a computer hardware configuration that includes more than one independent processing unit.
- A networked computing system is a collection of physical interconnected computers.
- A distributed computing system consists of a number of computers that are connected and managed, so that they automatically share the job processing load among the constituent computers, or separate the job load as appropriate particularly configured processors.
- Virtual memory satisfies the purpose to enlarge the address space, the set of addresses a program can utilise.
- Memory is central to the operation of a modern computer system. Memory is a large array of words or bytes, each location with its own address.

References

- Flynn, M. I. & MacHoes, M. A., 1997. *Understanding operating systems*, 2nd ed. Taylor & Francis.
- Yadav, A., 2008. *Microprocessor 8085, 8086*, Firewall Media.
- Prof. Bhor, H., Prof. Rote, U. & Prof. Shinde, U., *Operating System* [Pdf] Available at: <http://www.mu.ac.in/myweb_test/MCA%20study%20material/OS%20-%20PDF.pdf> [Accessed 30 May 2013].
- *Operating Systems* [Pdf] Available at: <<http://www.uow.edu.au/~nabg/ABC/C3.pdf>> [Accessed 30 May 2013].
- Dijkstra, E. W., *Lecture 2: THE multiprogramming system* [Video online] Available at: <<http://www.youtube.com/watch?v=Uiew30G2D1c>> [Accessed 30 May 2013].
- *Lecture10MultiprogramminAndFixedPartitions* [Video online] Available at: <http://www.youtube.com/watch?v=jKxFivAja_8> [Accessed 30 May 2013].

Recommended Reading

- Ramesh, V., 2010. *Principles of Operating Systems*, Laxmi Publications, Ltd.
- Ball, S., 2002. *Embedded Microprocessor Systems: Real World Design*, 3rd ed. Newnes.
- Mathivanan, N., 2003. *MICROPROCESSORS, PC HARDWARE AND INTERFACING*, PHI Learning Pvt. Ltd.

Self Assessment

1. In _____ environment, a computer provides computing services to several or many users concurrently on-line.
 - a. time sharing
 - b. batch processing operating system
 - c. multi programming
 - d. real time operating system
2. Computer _____ is an allocation of a computer system and its resources to more than one concurrent application, job or user.
 - a. batch processing operating system
 - b. multi programming
 - c. time sharing
 - d. real time operating system
3. A _____ system is a computer hardware configuration that includes more than one independent processing unit.
 - a. batch processing operating
 - b. multi processing
 - c. time sharing
 - d. real time operating system
4. _____ systems are not fundamentally different from single processor operating systems.
 - a. Batch processing operating
 - b. Network operating
 - c. Time sharing
 - d. Real time operating
5. In the _____, environment operating system dynamically allocates memory to multiple processes.
 - a. batch processing operating system
 - b. time sharing
 - c. real time operating system
 - d. multi programming
6. Which of the following statements is true?
 - a. In real time operating system, scheduling is the process of controlling and prioritising messages sent to a processor.
 - b. In operating system, multi programming is the process of controlling and prioritising messages sent to a processor.
 - c. In operating system, time sharing is the process of controlling and prioritising messages sent to a processor.
 - d. In operating system, scheduling is the process of controlling and prioritising messages sent to a processor.

7. Which of the following statements is true?
- a. Time-sharing is the sharing of a computing resource among many users by means of multiprogramming and multi-tasking.
 - b. Operating system is sharing of a computing resource among many users by means of multiprogramming and multi-tasking.
 - c. Memory management is the sharing of a computing resource among many users by means of multiprogramming and multi-tasking.
 - d. Time-sharing is the sharing of a computing resource among many users by means of memory management and multi-tasking.
8. Which of the following statements is true?
- a. Short-term scheduling revolves around admitting the programs to the scheduling process.
 - b. Long-term scheduling revolves around admitting the programs to the scheduling process.
 - c. Long-term scheduling revolves around admitting the programs to the multiprocessing system.
 - d. Long-term scheduling revolves around admitting the programs to the multi programming system.
9. Which of the following statements is true?
- a. The purpose of time sharing is to enlarge the address space, the set of addresses a program can utilise.
 - b. The purpose of memory management is to enlarge the address space, the set of addresses a program can utilise.
 - c. The purpose of virtual memory is to enlarge the address space, the set of addresses a program can utilise.
 - d. The purpose of scheduling is to enlarge the address space, the set of addresses a program can utilise.
10. Which of the following statements is true?
- a. Operating system is usually divided into fast primary storage and slow secondary storage.
 - b. Memory is usually divided into slow primary storage and fast secondary storage.
 - c. Memory is usually divided into fast primary storage and slow secondary storage.
 - d. Time sharing is usually divided into fast primary storage and slow secondary storage.

Chapter VII

Central Processing Unit

Aim

The aim of this chapter is to:

- introduce CPU organisation
- explain CPU operation
- elucidate processor organisation

Objectives

The objectives of the chapter are to:

- enlist the number of operands
- determine CPU operation
- explain addressing modes

Learning outcome

At the end of this chapter, you will be able to:

- enlist RISW, CISW and VLIW
- recognise addressing modes
- describe CPU organisation

7.1 Introduction

The CPU is the brain of computer. Sometimes simply referred to as the central processor, CPU is more commonly called “Processor”. The CPU is where most calculations take place. In terms of computing power, the CPU is the most important element of a computer system. On large machines, CPUs require one or more printed circuit boards. On personal computers and small workstations, the CPU is housed in a single chip called a microprocessor. Since the 1970’s, the microprocessor class of CPUs has almost completely overtaken all other CPU implementations.

The CPU itself is an internal component of the computer. Modern CPUs are small and square in shape and contain multiple metallic connectors or pins on the underside. The CPU is inserted directly into a CPU socket, pin side down, on the motherboard. Each motherboard will support only a specific type or range of CPU, so one must check the motherboard manufacturer’s specifications before attempting to replace or upgrade a CPU. Modern CPUs also have an attached heat sink and small fan that is located at the top of the CPU to help dissipate heat.

7.2 CPU Operation

The fundamental operation of most CPUs, regardless of the physical form they take, is to execute a sequence of stored instructions called a program. The program is represented by a series of numbers that are kept in some kind of computer memory. There are four steps that nearly all CPUs use in their operation: fetch, decode, execute, and write back.

Fetch

This involves retrieving an instruction (which is represented by a number or sequence of numbers) from program memory. The location in program memory is determined by a program counter (PC), which stores a number that identifies the current position in the program.

In other words, the program counter keeps a track of the CPU’s place in the program. After an instruction is fetched, the PC is incremented by the length of the instruction word in terms of memory units. Often the instruction to be fetched must be retrieved from relatively slow memory, causing the CPU to stall while waiting for the instruction to be returned. This issue is largely addressed in modern processors by caches and pipeline architectures. The instruction that the CPU fetches from memory is used to determine what the CPU is to do.

Decode

In the decode step, the instruction is broken up into parts that have significance to other portions of the CPU. The way in which the numerical instruction value is interpreted is defined by the CPU’s instruction set architecture (ISA). Often, one group of numbers in the instruction, called the opcode, indicates which operation to perform.

The remaining parts of the number usually provide information required for that instruction, such as operands for an addition operation. Such operands may be given as a constant value (called an immediate value), or as a place to locate a value: a register or a memory address, as determined by some addressing mode. In older designs, the portions of the CPU responsible for instruction decoding, were unchangeable hardware devices. However, in more abstract and complicated CPUs and ISAs, a micro program is often used to assist in translating instructions into various configuration signals for the CPU. This micro program is sometimes rewritable, so that it can be modified to change the way the CPU decodes instructions even after it has been manufactured.

Execute

After the fetch and decode steps, the execute step is performed. During this step, various portions of the CPU are connected, so they can perform the desired operation. If, for instance, an addition operation was requested, an arithmetic logic unit (ALU) will be connected to a set of inputs and a set of outputs.

The inputs provide the numbers to be added, and the outputs will contain the final sum. The ALU contains the circuitry to perform simple arithmetic and logical operations on the inputs (like addition and bitwise operations). If the addition operation produces a result too large for the CPU to handle, an arithmetic overflow flag in a flags register may also be set.

Write

The final step, write back, simply “writes back” the results of the execute step to some form of memory. Very often, the results are written to some internal CPU register for quick access by subsequent instructions. In other cases, results may be written to slower, but cheaper and larger, main memory. Some types of instructions manipulate the program counter rather than directly produce result data. These are generally called “jumps” and facilitate behaviour like loops, conditional program execution (through the use of a conditional jump), and functions in programs.

Many instructions will also change the state of digits in a “flags” register. These flags can be used to influence how a program behaves, since they often indicate the outcome of various operations. For example, one type of “compare” instruction considers two values and sets a number in the flags register according to which one is greater. This flag could then be used by a later jump instruction to determine program flow. After the execution of the instruction and write back of the resulting data, the entire process repeats, with the next instruction cycle normally fetching the next-in-sequence instruction because of the incremented value in the program counter.

If the completed instruction was a jump, the program counter will be modified to contain the address of the instruction that was jumped to, and program execution continues normally. In more complex CPUs than the ones described here, multiple instructions can be fetched, decoded, and executed simultaneously. This section describes what generally is referred to as the “Classic RISC pipeline”, which in fact is quite common among the simple CPUs used in many electronic devices (often called microcontroller). It largely ignores the important role of CPU cache, and therefore the access stage of the pipeline.

7.3 Processor Organisation

The two main components of processor organisation are:

7.3.1 General Register

The internal processor memory of a CPU is served by its registers. One of the key differences among various computers is the difference in their register sets. Some computers have very large, while some has smaller sets. But on the whole, from a user’s point of view, the register set can be classified under two basic categories.

- Programmer visible registers-These registers can be used by machine or assembly language programmers to minimise the references to main memory.
- Status control and registers-These registers can not be used by the programmers but are used to control the CPU or the execution of a program.

Different vendors have used some of these registers interchangeably; therefore one should not stick to these definitions rigidly.

7.3.2 Addressing Modes

Addressing modes are the aspects of the instruction set architecture in most central processing unit (CPU) designs. The various addressing modes that are defined in a given instruction set architecture define how machine language instructions in that architecture identify the operand (or operands) of each instruction.

An addressing mode specifies how to calculate the effective memory address of an operand by using information held in registers and/or constants contained within a machine instruction or elsewhere. In computer programming, addressing modes are primarily of interest to compiler writers and to those who write code directly in assembly language.

7.4 RISC, CISC and VLIW

RISC, CISC and VLIW design classifications are the modern equivalents of the micro, mini and supermini, scheme of the old.

7.4.1 RISC

Reduced instruction set computing or RISC, is a CPU design strategy based on the insight that simplified (as opposed to complex) instructions can provide higher performance, if this simplicity enables much faster execution of each instruction.

A computer based on this strategy is a reduced instruction set computer (also RISC). There are many proposals for precise definitions, but the term is slowly being replaced by the more descriptive load-store architecture.

Well known RISC families include DEC Alpha, AMD 29k, ARC, ARM, Atmel AVR, MIPS, PA-RISC, Power (including PowerPC), SuperH, and SPARC.

Some aspects attributed to the first RISC-labeled designs around 1975 include the observations that the memory-restricted compilers of the time were often unable to take advantage of features intended to facilitate manual assembly coding, and that complex addressing modes take many cycles to perform due to the required additional memory accesses.

It was argued that such functions would be better performed by sequences of simpler instructions if this could yield implementations small enough to leave room for many registers, reducing the number of slow memory accesses.

In these simple designs, most instructions are of uniform length and similar structure, arithmetic operations are restricted to CPU registers and only separate load and store instructions access memory.

These properties enable a better balancing of pipeline stages than before, making RISC pipelines significantly more efficient and allowing higher clock frequencies.

7.4.2 VLIW

Very long instruction word or VLIW refers to a CPU architecture designed to take advantage of instruction level parallelism (ILP). A processor that executes every instruction one after the other (i.e., a non-pipelined scalar architecture) may use processor resources inefficiently, potentially leading to poor performance. The performance can be improved by executing different sub-steps of sequential instructions simultaneously (this is pipelining), or even executing multiple instructions entirely simultaneously as in superscalar architectures. Further improvement can be achieved by executing instructions in an order different from the order they appear in the program; this is called out-of-order execution. As often implemented, these three techniques all come at a cost: increased hardware complexity. Before executing any operations in parallel, the processor must verify that the instructions do not have interdependencies.

For example, a result of first instruction is used as an input for the second instruction. Clearly, they cannot execute at the same time, and the second instruction can't be executed before the first. Modern out-of-order processors have increased the hardware resources which do the scheduling of instructions and determining of interdependencies.

The VLIW approach, on the other hand, executes operations in parallel based on a fixed schedule determined when programs are compiled. Since determining the order of execution of operations (including which operations can execute simultaneously) is handled by the compiler, the processor does not need the scheduling hardware that the three techniques described above require. As a result, VLIW CPUs offer significant computational power with less hardware complexity (but greater compiler complexity) than is associated with most superscalar CPUs.

As is the case with any novel architectural approach, the concept is only as useful as code generation makes it. That is, the fact that a number of special-purpose instructions are available to facilitate certain complicated operations—say, Fast Fourier Transform (FFT) computation or certain calculations that recur in tomography contexts—is useless if compilers are unable to spot relevant source code constructs and generate target code that duly utilises the CPU's advanced offerings. A fortiori, the programmer must be able to express his algorithms in a manner that makes the compiler's task.

7.4.3 CISC

A complex instruction set computer (CISC), is a computer where single instructions can execute several low-level operations (such as a load from memory, an arithmetic operation, and a memory store) and/or are capable of multi-step operations or addressing modes within single instructions. The term was retroactively coined in contrast to reduced instruction set computer (RISC).

7.5 Instruction Set

Any given instruction set can be implemented in a variety of ways. All ways of implementing an instruction set give the same programming model, and they all are able to run the same binary executables. The various ways of implementing an instruction set give different tradeoffs between cost, performance, power consumption, size, etc. When designing micro architectures, engineers use blocks of “hard-wired” electronic circuitry (often designed separately) such as adders, multiplexers, counters, registers, ALUs etc. Some kind of register transfer language is then often used to describe the decoding and sequencing of each instruction of an ISA using this physical micro architecture.

There are two basic ways to build a control unit to implement this description (although many designs use middle ways or compromises):

- Early computer designs and some of the simpler RISC computers “hard-wired” the complete instruction set decoding and sequencing (just like the rest of the micro architecture).
- Other designs employ microcode routines and/or tables to do this—typically as on chip ROMs and/or PLAs (although separate RAMs have been used historically).

There are also some new CPU designs which compile the instruction set to a writable RAM or FLASH inside the CPU or an FPGA (reconfigurable computing). The Western Digital MCP-1600 is an older example, using a dedicated, separate ROM for microcode. An ISA can also be emulated in software by an interpreter. Naturally, due to the interpretation overhead, this is slower than directly running programs on the emulated hardware, unless the hardware running the emulator is an order of magnitude faster. Today, it is common practice for vendors of new ISAs or micro architectures to make software emulators available to software developers before the hardware implementation is ready.

Often the details of the implementation have a strong influence on the particular instructions selected for the instruction set. For example, many implementations of the instruction pipeline only allow a single memory load or memory store per instruction, leading to load-store architecture (RISC). In another example, some early ways of implementing the instruction pipeline led to a delay slot.

7.5.1 Code Density

In early computers, program memory was expensive, so minimising the size of a program to make sure it would fit in the limited memory was often central. Thus, the combined size of all the instructions needed to perform a particular task, the code density, was an important characteristic of any instruction set.

Computers with high code density also often had (and have still) complex instructions for procedure entry, parameterised returns, loops etc. (therefore retroactively named as Complex Instruction Set Computers, CISC). However, more typical or frequent “CISC” instructions merely combine a basic ALU operation, such as “add”, with the access of one or more operands in memory (using addressing modes such as direct, indirect, indexed etc.). Certain architectures may allow two or three operands (including the result) directly in memory or may be able to perform functions such as automatic pointer increment etc.

Software-implemented instruction sets may have even more complex and powerful instructions. Reduced instruction-set computers, RISC, were first widely implemented during a period of rapidly-growing memory subsystems and sacrifice code density in order to simplify implementation circuitry and thereby try to increase performance via higher clock frequencies and more registers. RISC instructions typically perform only a single operation, such as an “add” of registers or a “load” from a memory location into a register; they also normally use a fixed instruction width, whereas a typical CISC instruction set has many instructions shorter than this fixed length.

Fixed-width instructions are less complicated to handle than variable-width instructions for several reasons and are therefore somewhat easier to optimise for speed. However, as RISC computers normally require more and often longer instructions to implement a given task, they inherently make less optimal use of bus bandwidth and cache memories.

Minimal instructions set computers (MISC) are a form of stack machine, where there are few separate instructions (16-64), so that multiple instructions can be fit into a single machine word. This often takes little silicon to implement, so they can be easily realised in an FPGA or in a multi-core form. Code density is similar to RISC; the increased instruction density is offset by requiring more of the primitive instructions to do a task.

7.5.2 Number of Operands

Instruction sets may be categorised by the maximum number of operands explicitly specified in instructions. 0-operand (zero address machines), so called “stack machines”: All arithmetic operations take place using the top one or two positions on the stack; 1-operand **push** and **pop** instructions are used to access memory: **push a**, **push b**, **add**, **pop c**.

- 1-operand: One address machines, called “accumulator machines”, include most early computers and many small microcontrollers. Most instructions specify a single explicit right operand (a register, a memory location, or a constant) with the implicit accumulator as both destination and left (or only) operand: **load a**, **add b**, **store c**. A related class is practical stack machines which often allow a single explicit operand in arithmetic instructions: **push a**, **add b**, **pop c**.
- 2-operand: Many CISC and RISC machines fall under this category:
 - CISC: **load a, reg1**; **add reg1, b**; **store reg1, c**
 - RISC: Requiring explicit memory loads, the instructions would be: **load a, reg1**; **load b, reg2**; **add reg1, reg2**; **store reg2, c**
- 3-operand: Allows better reuse of data:
 - CISC: It becomes either a single instruction: **add a, b, c**, or more typically: **move a, reg1**; **add reg1, b, c** as most machines are limited to two memory operands.
 - RISC: Due to the large number of bits needed to encode three registers, this scheme is typically not available in RISC processors using small 16-bit instructions; arithmetic instructions use registers only, so explicit 2-operand load/store instructions are needed: **load a, reg1**; **load b, reg2**; **add reg1+reg2->reg3**; **store reg3, c**;
- More operands: Some CISC machines permit a variety of addressing modes that allow more than 3 operands (registers or memory accesses), such as the VAX “POLY” polynomial evaluation instruction. Each instruction specifies some number of operands (registers, memory locations, or immediate values) explicitly. Some instructions give one or both operands implicitly, such as by being stored on top of the stack or in an implicit register. When some of the operands are given implicitly, the number of specified operands in an instruction is smaller than the parity of the operation. When a “destination operand” explicitly specifies the destination, the number of operand specifies in an instruction is larger than the parity of the operation. Some instruction sets have different numbers of operands for different instructions.

Summary

- The chapter explains central processing unit as the brain of the computer.
- The instruction that the CPU fetches from memory is used to determine what the CPU is to do.
- The internal processor memory of a CPU is served by its registers.
- Addressing modes are aspects of the instruction set architecture in most central processing unit (CPU) designs.
- Reduced instruction set computing, or RISC, is a CPU design strategy based on the insight that simplified instructions can provide higher performance if this simplicity enables much faster execution of each instruction.
- Very long instruction word or VLIW refers to a CPU architecture designed to take advantage of instruction level parallelism.
- A complex instruction set computer (CISC), is a computer where single instructions can execute several low-level operations and/or are capable of multi-step operations or addressing modes within single instructions.
- The various ways of implementing an instruction set give different tradeoffs between cost, performance, power consumption, size, etc.
- Instruction sets may be categorised by the maximum number of operands explicitly specified in instructions.

References

- Fisher, A. J., Zaraboschi, P. & Young, C., 2005. *Embedded Computing: A Vliw Approach To Architecture*, Compilers And Tools, Elsevier.
- Srinath, N. K., 2005. *8085 MICROPROCESSOR: PROGRAMMING AND INTERFACING*, PHI Learning Pvt. Ltd.
- *8085 INSTRUCTION SET* [Pdf] Available at: <http://engeletrica.sobral.ufc.br/professores/marcelo/Micro/8085_is_details.pdf> [Accessed 30 May 2013].
- Tatla, G. H., *INSTRUCTION SET OF 8085* [Pdf] Available at: <<http://www.eazynotes.com/notes/microprocessor/Slides/instruction-set-of-8085.pdf>> [Accessed 30 May 2013].
- Prof. Kumar, A., *Lecture -3 Instruction Set Architecture* [Video online] Available at: <<http://www.youtube.com/watch?v=HbsuwpJgKao>> [Accessed 30 May 2013].
- Prof. Pal, A., 2012. *lec 4 - Instruction Set : Vocabulary of the Machine* [Video online] Available at: <http://www.youtube.com/watch?v=tjZ2Mh_MV6g> [Accessed 30 May 2013].

Recommended Reading

- Rajaraman, 2004. *Introduction To Information Technology*, PHI Learning Pvt. Ltd..
- Swamy, G. T., 2006. *Microprocessor (8085) Lab Manual*, Firewall Media.
- Ghosh, P. K. & Sridhar, P. R., 2004. *Introduction To Microprocessors For Engineers And Scientists* 2Nd Ed., 2nd ed. PHI Learning Pvt. Ltd.

Self Assessment

1. In terms of computing power, the _____ is the most important element of a computer system.
 - a. bus
 - b. CPU
 - c. input device
 - d. output device
2. Modern CPUs are _____, square and contain multiple metallic connectors or pins on the underside.
 - a. small
 - b. large
 - c. complex
 - d. simple
3. The fundamental operation of most CPUs, regardless of the physical form they take, is to execute a sequence of stored instructions called _____.
 - a. data
 - b. algorithm
 - c. program
 - d. software
4. There are four steps that nearly all CPUs use in their operation: fetch, _____, execute, and write back.
 - a. code
 - b. read
 - c. store
 - d. decode
5. Very long instruction word or VLIW refers to CPU _____ to take advantage of instruction level parallelism.
 - a. architecture
 - b. design
 - c. program
 - d. components
6. Which of the following statements is true?
 - a. The first step, write back, simply “writes back” the results of the execute step to some form of memory.
 - b. The final step, read back, simply “reads back” the results of the execute step to some form of memory.
 - c. The final step, write back, simply “writes back” the results of the execute step to some form of memory.
 - d. The final step, write back, simply “writes back” the results of the execute step to some form of data.
7. Which of the following statements is true?
 - a. Status Control and Registers cannot be used by the programmers, but are used to control the CPU or the execution of a program.
 - b. Status Control and Registers can be used by the programmers, but are used to control the CPU or the execution of a program.
 - c. Status Control and Registers cannot be used by the programmers and thus, no control is maintained.
 - d. Status Control and Registers cannot be used by the programmers, but are used to control the bus or the execution of a program.

8. Which of the following statements is true?
- a. General registers are an aspect of the instruction set architecture in most central processing unit (CPU) designs.
 - b. RISC is an aspect of the instruction set architecture in most central processing unit (CPU) designs.
 - c. CISC are an aspect of the instruction set architecture in most central processing unit (CPU) designs.
 - d. Addressing modes are an aspect of the instruction set architecture in most central processing unit (CPU) designs.
9. Which of the following statements is true?
- a. The RISC approach, on the other hand, executes operations in parallel, based on a fixed schedule determined when data is compiled.
 - b. The CISC approach, on the other hand, executes operations in parallel, based on a fixed schedule determined when programs are compiled.
 - c. The VLIW approach, on the other hand, executes operations in parallel, based on a fixed schedule determined when programs are compiled.
 - d. The VLIW approach, on the other hand, executes operations in parallel, based on a fixed schedule determined when data are compiled.
10. Which of the following statements is true?
- a. CISC instructions typically perform only a single operation, such as an “add” of registers or a “load” from a memory location into a register.
 - b. RISC instructions typically perform only a single operation, such as “store” of registers or a “load” from a memory location into a register.
 - c. RISC instructions typically perform only a single operation, such as an “add” of registers or a “load” from a memory location into a register.
 - d. VLIW instructions typically perform only a single operation, such as an “add” of registers or a “load” from a memory location into a register.

Chapter VIII

Introduction to Multiprocessor System

Aim

The aim of this chapter is to:

- explain multiprocessor system
- elucidate Flynn's classification
- elucidate pipelining

Objectives

The objectives of this chapter are to:

- enlist the characteristics of multiprocessor
- explain multi-port memory
- determine inter processor arbitration

Learning outcome

At the end of this chapter, you will be able to:

- enlist the types of parallel organisation
- understand the characteristics of multiprocessor
- describe parallel processing

8.1 Introduction

The computers can be made faster by increasing their clock speed or by using improved VLSI technology. But there is a practical limitation to both of them. The only way that seems possible to theoretically increase the speed in an unlimited fashion is to have multiple modules, that is, split the original problems into independent modules and execute them simultaneously.

When more number of processors do the job simultaneously, the speed also is more. “Parallel processing” is a term used to denote a large class of techniques that are used to provide simultaneous data processing tasks for the purpose of increasing the computational speed of a computer.

Most basic parallel processing can be seen in uniprocessors (the systems with one CPU) as well. The simplest parallel processing is the parallel transfer of data from one module to another in the computer. Instead of transferring the data bit by bit, 8/16/32 bit are transferred simultaneously through the bus. Various cases of instruction execution also take place simultaneously. For example, while one instruction is being executed in the 8088 based PC, the next instruction is fetched and kept in the queue, ready for execution. In a similar manner, CPU and I/O operations are also overlapped to increase the overall speed of the computer.

8.2 Flynn’s Classification

There are varieties of ways in which parallel processing can be classified. One of the most popular classifications of parallel computer is by M.J. Flynn, called Flynn’s classification, and it is based on the multiplicity of instruction streams and data streams in a computer system. The sequence of instructions read from the memory constitutes the instruction stream, and the data they operate of in the processor constitutes the data stream. Flynn’s classification divides the computers into four categories:

- Single instruction single data (SISD)
- Single instruction multiple data (SIMD)
- Multiple instruction single data (MISD)
- Multiple instruction multiple data (MIMD)

SISD organisation is available in most serial computers today. Instructions are executed sequentially, though may be overlapped in their execution stage (pipelining). All the functional units are under the supervision of one control unit. The Von Neumann architecture falls under this category. SIMD organisation consists of multiple processing elements supervised by the same control unit. All processing units receive the same instruction broadcast but works on different data streams. One of the very common example is the execution of “For Loop”, in which the same set of instructions are executed for, may be a different set of data.

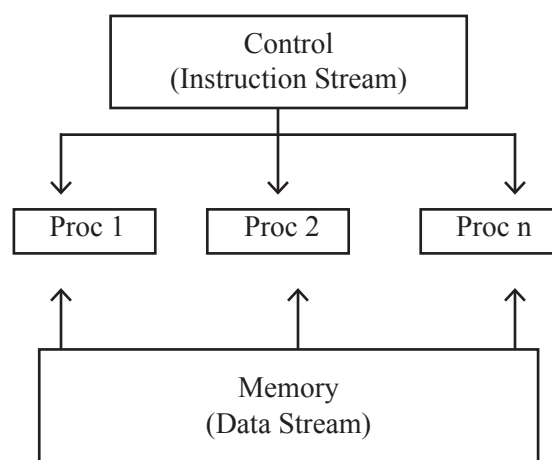


Fig. 8.1 SIMD

MISD organisation consists of N processor units, each working on a different set of instructions but working on the same set of data. The output of one unit becomes the input to the other unit. There is a commercial computer of this kind.

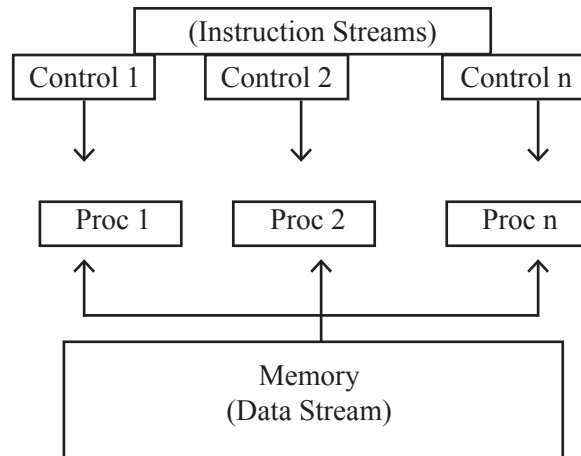


Fig. 8.2 MISD

MIMD organisation implies interactions between N processors because all memory streams are derived from the same data stream shared by all processors. If the interaction between the processors is high, it is called a tightly coupled system, or else it is called a loosely coupled system. Most multiprocessors fit into this category.

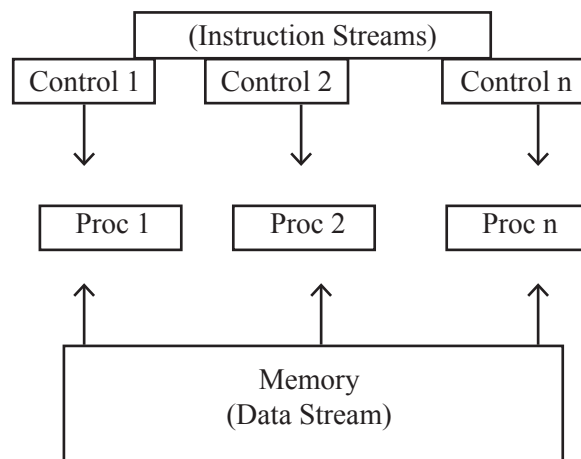


Fig 8.3 MIMD

8.3 Characteristics of Multiprocessors

Following are the characteristics of multiprocessors:

- A multiprocessor system is an interconnection of two or more CPUs, with memory and input-output equipment. As defined earlier, multiprocessors can be put under MIMD category.
- The term multiprocessor is some times confused with the term multicomputer. Though both of them support concurrent operations, there is an important difference between a system with multiple computers and a system with multiple processors. In a multicomputer system, there are multiple computers with their own operating systems, which communicate with each other, if needed, through communication links.
- A multiprocessor system, on the other hand, is controlled by a single operating system, which coordinates the activities of the various processors, either through shared memory or through interprocessor messages.
- The advantages of multiprocessor systems are:
 - increased reliability because of redundancy in processors

- increased throughput because of execution of - multiple jobs in parallel - portions of the same job in parallel
- A single job can be divided into independent tasks, either manually by the programmer, or by the compiler, which finds the portions of the program that are data independent, and can be executed in parallel.
- The multiprocessors are further classified into two groups, depending on the way their memory is organised. The processors with shared memory are called tightly coupled or shared memory processors.
- The information in these processors is shared through the common memory. Each of the processors can also have their local memory too.
- The other class of multiprocessors is loosely coupled or distributed memory multiprocessors. In this, all the processors have their own private memory, and they share information with each other through interconnection switching scheme or message passing.
- The principal characteristic of a multiprocessor is its ability to share a set of main memory and some I/O devices. This sharing is possible through some physical connections between them called the interconnection structures.

8.3.1 Interprocessor Arbitration

Computer system needs buses to facilitate the transfer of information between its various components. For example, even in a uniprocessor system, if the CPU has to access a memory location, it sends the address of the memory location on the address bus. This address activates a memory chip. The CPU then sends a read signal through the control bus, in response of which the memory puts the data on the address bus. This address activates a memory chip. The CPU then sends a read signal through the control bus, in response of which the memory puts the data on the data bus. Similarly, in a multiprocessor system, if any processor has to read a memory location from the shared area, it follows the similar routine.

There are buses that transfer data between the CPUs and memory. These are called memory buses. An I/O bus is used to transfer data to and from input and output devices. A bus that connects major components in a multiprocessor system, such as CPUs, I/Os, and memory is called system bus. A processor, in a multiprocessor system, requests the access of a component through the system bus. If there is no processor accessing the bus at that time, it is then given the control of the bus immediately. If there is a second processor utilising the bus, then this processor has to wait for the bus to be freed. If at any time, there is a request for the services of the bus by more than one processor, then the arbitration is performed to resolve the conflict. A bus controller is placed between the local bus and the system bus to handle this.

8.3.2 Inter Processor Communication and Synchronisation

In a multiprocessor system, it becomes very necessary, that there is a proper communication protocol between the various processors. In a shared memory multiprocessor system, a common area in the memory is provided, in which all the messages that need to be communicated to other processors are written.

A proper synchronisation is also needed whenever there is a race of two or more processors for shared resources like I/O resources. The operating system in this case is given the task of allocating the resources to the various processors in a way that at any time not more than one processor uses the resource.

A very common problem can occur when two or more resources are trying to access a resource, can be modified. For example, processor 1 and 2 are simultaneously trying to access memory location 100. Processor 1 is writing on to the location while processor 2 is reading it.

The chances are that, processor 2 will end up reading erroneous data. Such kind of resources which need to be protected from simultaneous access of more than one processors are called critical sections. The following assumptions are made regarding the critical sections:

Mutual exclusion	At most one processor can be in a critical section at a time.
Termination	The critical section is executed in a finite time.
Fair scheduling	A process attempting to enter the critical section will eventually do so in a finite time.

Table 8.1 Critical sections

- A binary value called a semaphore is usually used to indicate whether a processor is currently executing the critical section.

8.3.3 Cache Coherence

Cache memories are high speed buffers which are inserted between the processor and the main memory to capture those portions of the contents of main memory which are currently in use. These memories are five to ten times faster than main memories and therefore reduce the overall access time. In a multiprocessor system, with shared memory, each processor has its own set of private cache.

Multiple copies of the cache are provided with each processor to reduce the access time. Each processor, whenever accesses the shared memory, also updates its private cache. This introduced the problem of cache coherence, which may result in data inconsistency. That is, several copies of the same data may exist in different caches at any given time. For example, let us assume there are two processors x and y. Both have the same copy of the cache. Processor x produces data 'a', which is to be consumed by processor y. Processor updates the value of 'a' in its own private copy of the cache. As it does not have any access to the private copy of cache of processor y, the processor y continues to use the variable 'a' with old value, unless it is informed of the change.

8.4 Parallel Processing

Parallel processing means simultaneous use of more than one CPU or processor core to execute a program or multiple computational threads. Ideally, parallel processing makes programs run faster because there are more engines (CPUs or cores) running it. In practice, it is often difficult to divide a program in such a way that separate CPUs or cores can execute different portions without interfering with each other. Most computers have just one CPU, but some models have several, and multi-core processor chips are becoming the norm.

There are even computers with thousands of CPUs. With single CPU, single-core computers; it is possible to perform parallel processing by connecting the computers in a network. However, this type of parallel processing requires very sophisticated software called distributed processing software.

- Parallel processors can be categorised into several categories. These include:
 - Array processors
 - Distributed architecture
 - Multiprocessors
 - Data flow architectures

Array processors are parallel architectures which deal with repetitive operations. These are the examples of SIMD architecture, with applications varying from mathematical array operations and in structures in which data objects are known well in advance.

Distributed architecture are composed of relatively autonomous subsystems which are capable of handling complete system and execution functions, and which cooperate together to run a large application. Multiprocessors are architectures composed of multiple computing units which operate in a synchronous mode. Generally, both shared and local memory can be available for these processors. The communication between the processors takes place either through the shared memory area or through the interprocessor messages.

Dataflow architectures are functionally distributed architectures, in which the operations are triggered with the arrival of data at these processors. They may be viewed as very general MIMD parallel architectures.

8.5 Types of Parallel Organisation

The types of parallel organisation are as follows:

Time sharing bus

- An extensible time-sharing bus structure is provided for a microprocessor to read data from, or write data to a memory. An address/data bus transfers addresses and data between the microprocessor and the memory in a time sharing manner.
- The combination of the logic levels of two control lines is used to determine that the address/data bus is utilised to transfer addresses, to read data or to write data.
- Thus, the pin number required in the bus interface is reduced, and the memory capacity can be increased flexibly.

Multiport memory

- A multiport memory has a RAM port includes a memory cell array having a plurality of memory cells arranged in a matrix form.
- It has a sense amplifier circuit for sensing potential of a bit line after the storage potential has been transferred from the memory cells and restore circuit connected to the bit line for pulling up the potential of the bit line at the predetermined timing after sense operation has been started.
- It has a barrier circuit connected between the bit line and the sense amplifier circuit; and a SAM port including a data register, transfer gate and functional means for transferring serial data in the column direction.
- In this memory, the RAM port is connected to the SAM port by the transfer gate with the bit line directly connected to the data register, and the potentials at the bit line are amplified by the sense amplifier circuit and are directly transferred to the data register.

8.6 Pipelining

The basic idea behind pipeline design is quite natural; it is not specific to computer technology. In fact, the name pipeline stems from the analogy with petroleum pipelines, in which a sequence of hydrocarbon products is pumped through a pipeline.

The last product might well be entering the pipeline before the first product has been removed from the terminus. The key contribution of pipelining is that it provides a way to start a new task before an old one has been completed. The concept can be better understood by taking an example of an industrial plant.

To achieve pipelining, one must subdivide the input task into a sequence of subtasks, each of which can be executed by a specialised hardware stage that operates concurrently with other stages in the hardware. Successive tasks are streamed into the pipe and get executed in an overlapped fashion at the subtask level. Hence, the completion rate is not a function of the total processing time, but rather of how soon a new process can be introduced. The subdivision of labour in assembly lines has contributed to the success of mass production in modern industry. By the same token, pipeline processing has led to the tremendous improvement of system throughput in the modern digital computers.

Now consider that for execution of an instruction, a computers performs N processing steps. A server unit as represented in Fig. 8.4 (a) can perform any one of the N steps in one unit time. If we process M instructions by this server, then the rate of completion is one instruction of every N steps, and the time behaviour of the job stream is as described in Fig. 8.4 (a). Compare Fig. 8.4 (a) with Fig. 8.4 (b) which shows N servers concatenated in a sequence each performing only a single step flows through the collection of servers by visiting server 1, then server 2, and so on, and finally emerging from server N after N steps. The time behaviour of this system is shown in Fig. 8.4 (b). Fig. 8.4 (b) is an ideal model of a constant speed assembly line, such as an automobile assembly plant.

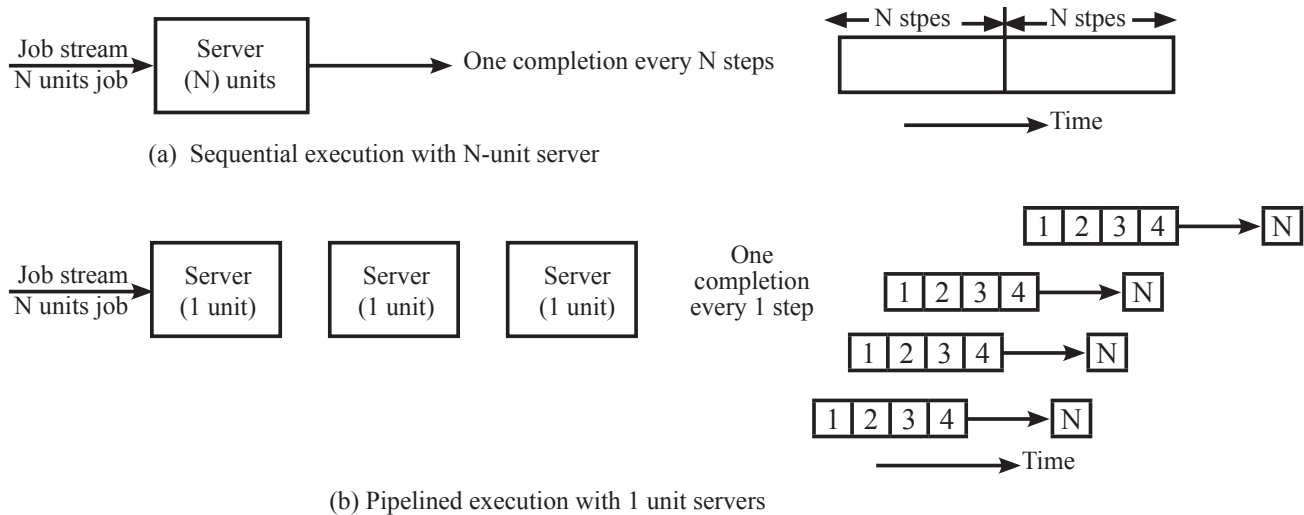


Fig. 8.4 Two ways of executing N-unit jobs in a stream

Thus, any arithmetic task is divided into steps. Thus, a sort of concurrent operations can be obtained in the pipeline. This will help in increasing the throughput of the system. Though superficially it seems, that it should be possible to execute any application in various pipeline stages, it is not actually so. Only those applications, which can be broken into independent subtasks, can take the advantage of pipelining. It is most efficient for those applications that need to repeat the same task many times with different sets of data. Fig. 8.5 shows the four segment pipeline.

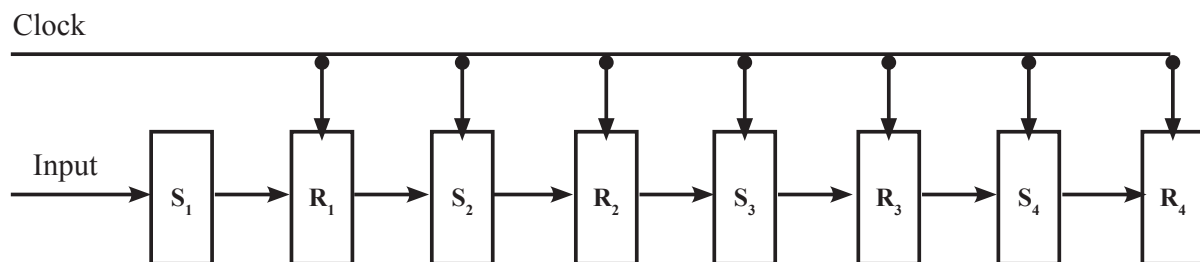


Fig. 8.5 Four segment pipeline

The figure shows two kinds of modules, S_i and R_i . These specify the processing circuits and the registers to hold the intermediate results, respectively. The information flows from left to right, after being input from the left most modules.

The task can be defined as the total operation going from all the segments of the pipeline. The behaviour of a pipelined operation can be explained with the help of a space-time diagram, Fig. 8.6. This illustrates the overlapped behaviour in a pipelined processor having four segments. The horizontal axis displays the time in clock cycles, and the vertical axis displays the segment number.

The diagram shows the six tasks, T_1 through T_6 . A task is the total operation performed after going through all the segments of a pipeline. At the initial clock cycles, the segment 1 is busy with task T_1 while all other segments are idle. In the second clock cycle, the second segment starts with the first task output from the segment 1, while the segment 1 starts with the second task (T_2). Once the pipe is filled up, it will output one result per clock period, independent of the number of stages in the pipe.

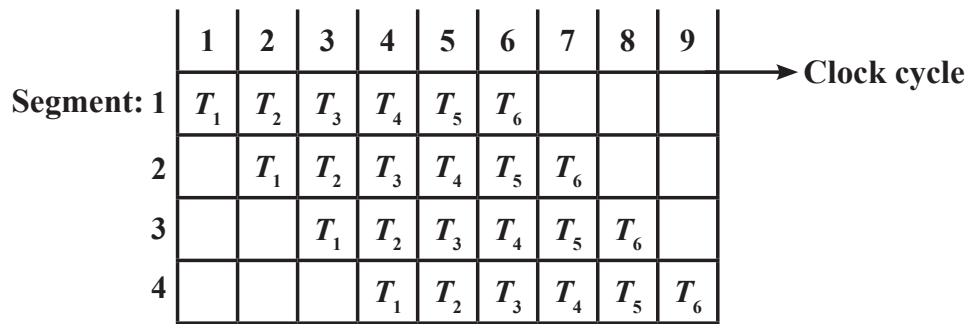


Fig. 8.6 Space time diagram for a segment pipeline

Let us now expand our example, by considering the case of k-segment pipeline. Let us assume the clock cycle time is T_p and a total of n tasks are to be executed. The first task requires a time equal to $k T_p$ to complete its operation as it has to go through k stages of the pipeline. The remaining (n-1) tasks emerge from the pipe at the rate of one task per clock cycle and they will be completed after a time equal to $(n-1) T_p$. Therefore, to complete n tasks using a k segment pipeline, the total time required is $k + (n-1)$ clock periods.

For example, in the pipelined processing of fig. 8.5, with four segments and six tasks. The time required to complete all the operations is $4 + (6-1) = 9$ clock cycles, as indicated in the Fig. 8.6

- Now let us assume that in a non-pipelined processor, a task is executed in T_n times, then, the speedup obtained from a k stage, n tasks pipeline can be given as:

$$S_k = \text{Time taken in non pipelined system} / \text{Time taken in a k stage pipelined system} \\ = n T_n / (k + n - 1) T_p$$

- As the number of tasks increases, n becomes much larger than $k - 1$, and $k + n - 1$ approaches the value of n, the speedup becomes

$$S_k = T_n / T_p$$

- If we assume that the time it takes to process the task is same in the pipeline and the non-pipeline circuits, we will have $T_n = k T_p$. Including this assumption, the speedup becomes:

$$S_k = k T_p / T_p = k$$

- Example: Let us consider a pipeline system with the following specifications.

time takes to process a sub operation in each segment = 20 nsecs
 number of segments in the pipeline = 4
 number of tasks being executed in sequence = 100
 time takes to complete the pipeline = $(k + n - 1) T_p$
 $= (4 + 100 - 1) 20$
 $= 2060 \text{ nsecs}$

Drawback of a pipeline

- The performance of a pipeline highly depends on the data dependency and branches in the program.
- In this memory, the RAM port is connected to the SAM port by the transfer gate with the bit line directly connected to the data register, and the potentials at the bit line are amplified by the sense amplifier circuit and are directly transferred to the data register.

Summary

- The chapter explains multiprocessor system and its functioning.
- Flynn's Classification is based on the multiplicity of instruction streams and data streams in a computer system.
- Flynn's classification divides the computers into four categories that are SISD, SIMD, MISD and MIMD.
- A multiprocessor system is an interconnection of two or more CPU, with memory and input-output equipment.
- Cache memories are high speed buffers which are inserted between the processor and the main memory to capture those portions of the contents of main memory which are currently in use.
- Parallel processing means simultaneous use of more than one CPU or processor core to execute a program or multiple computational threads.
- The key contribution of pipelining is that, it provides a way to start a new task before an old one has been completed.

References

- Bamakhrama, M., 2008. *Embedded Multiprocessor System-on-chip for Access Network Processing*, GRIN Verlag.
- Jerraya, A. A. & Wolf, W. H., 2005. *Multiprocessor Systems on Chips*, 2nd ed. Morgan Kaufmann.
- *UNIT 1 MULTIPROCESSOR SYSTEMS* [Pdf] Available at: <<http://dspace.sngce.ac.in/bitstream/handle/123456789/1953/Media-B3U1mcs-041.pdf?sequence=10>> [Accessed 30 May 2013].
- Kotz, D., *INTRODUCTION TO MULTIPROCESSOR I/O ARCHITECTURE* [Pdf] Available at: <<http://www.cs.dartmouth.edu/~dfk/papers/kotz-pioarch.pdf>> [Accessed 30 May 2013].
- Dr. Mall, R., 2011. *Mod-01 Lec-17 Real-Time Task Scheduling on Multiprocessors and Distributed Systems* [Video online] Available at: <<http://www.youtube.com/watch?v=kgcUVZARftQ>> [Accessed 30 May 2013].
- Dr. Mall, R., 2011. *Mod-01 Lec-18 Real-Time Task Scheduling on Multiprocessors and Distributed Systems (Contd.)* [Video online] Available at: <<http://www.youtube.com/watch?v=kgcUVZARftQ>> [Accessed 30 May 2013].

Recommended Reading

- Godse, A. P. & Godse, D. A., 2009. *Microprocessor, Microcontroller And Embedded Systems*, Technical Publications.
- Kempf, T., Ascheid, G. & Leupers, R., 2011. *Multiprocessor Systems on Chip: Design Space Exploration*, Springer.
- *Introduction To Computer Science*, Pearson Education India.

Self Assessment

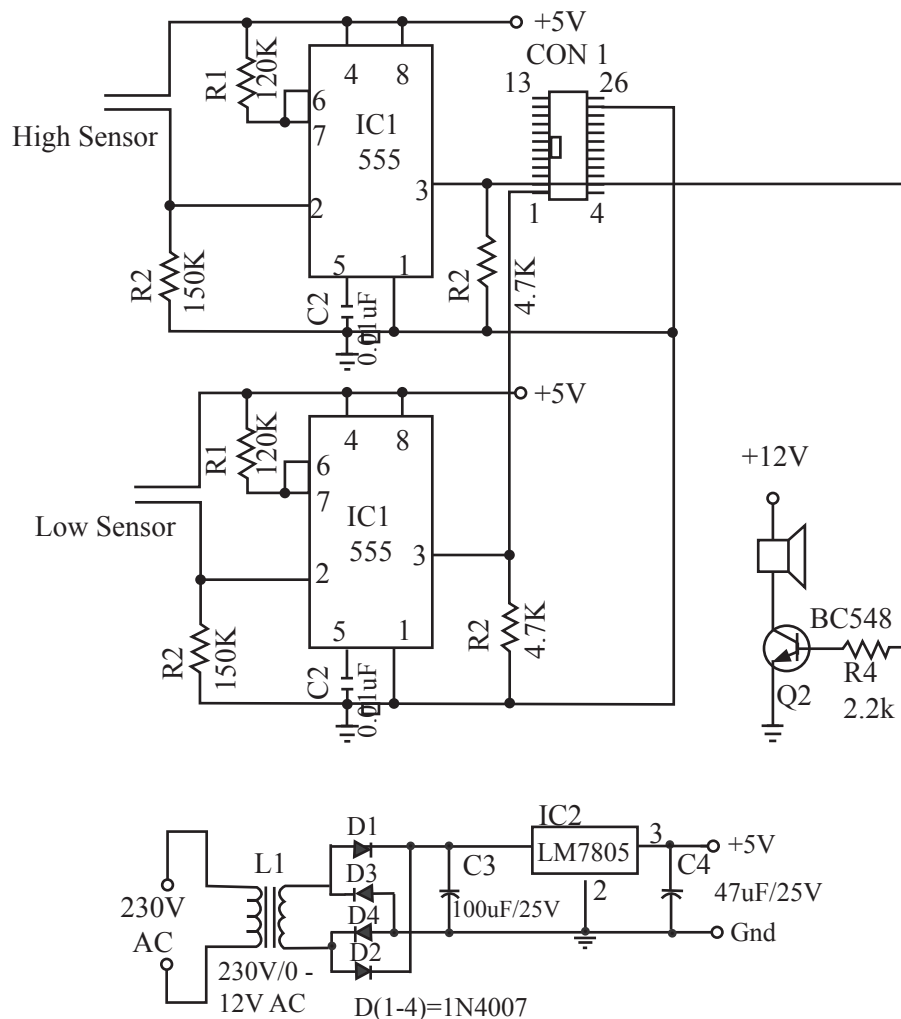
1. The _____ is the parallel transfer of data from one module to another in the computer.
 - a. parallel processing
 - b. multi programming
 - c. inter processor arbitration
 - d. array processing
2. _____ organisation is available in most serial computers today.
 - a. SIMD
 - b. SISD
 - c. MIMD
 - d. MISD
3. _____ organisation implies interactions between N processors because all memory streams are derived from the same data stream shared by all processors.
 - a. SIMD
 - b. SISD
 - c. MIMD
 - d. MISD
4. _____ are high speed buffers which are inserted between the processor and the main memory, to capture those portions of the contents of main memory which are currently in use.
 - a. Cache memories
 - b. Virtual memories
 - c. Bus
 - d. Array processor
5. Computer system needs _____ to facilitate the transfer of information between its various components.
 - a. cache memories
 - b. virtual memories
 - c. bus
 - d. array processor
6. Which of the following statements is true?
 - a. In the bus system, it becomes very necessary that there is a proper communication protocol between the various processors.
 - b. In the multiprocessor system, it becomes very necessary that there is a proper communication protocol between the various processors.
 - c. In the virtual memory system, it becomes very necessary that there is a proper communication protocol between the various processors.
 - d. In the inter processor arbitration system, it becomes very necessary that there is a proper communication protocol between the various processors.

7. Which of the following statements is true?
 - a. Parallel processing means simultaneous use of more than one CPU or processor core to execute a program or multiple computational threads.
 - b. Inter processor arbitration means simultaneous use of more than one CPU or processor core to execute a program or multiple computational threads.
 - c. Cache coherence means simultaneous use of more than one CPU or processor core to execute a program or multiple computational threads.
 - d. Time sharing bus means simultaneous use of more than one CPU or processor core to execute a program or multiple computational threads.
8. Which of the following statements is true?
 - a. MIMD organisation consists of multiple processing elements supervised by the same control unit.
 - b. MISD organisation consists of multiple processing elements supervised by the same control unit.
 - c. SISD organisation consists of multiple processing elements supervised by the same control unit.
 - d. SIMD organisation consists of multiple processing elements supervised by the same control unit.
9. Which of the following statements is true?
 - a. Cache coherence is parallel architectures which deals with repetitive operations.
 - b. Parallel processing means parallel architectures which deal with repetitive operations.
 - c. SISD are parallel architectures which deal with repetitive operations.
 - d. Array processors are parallel architectures which deal with repetitive operations.
10. Which of the following statements is true?
 - a. An extensible time-sharing bus structure is provided for a microprocessor to read data from or write data to a memory.
 - b. An extensible pipelining structure is provided for a microprocessor to read data from or write data to a memory.
 - c. An extensible parallel processing structure is provided for a microprocessor to read data from or write data to a memory.
 - d. An extensible SIMD structure is provided for a microprocessor to read data from or write data to a memory.

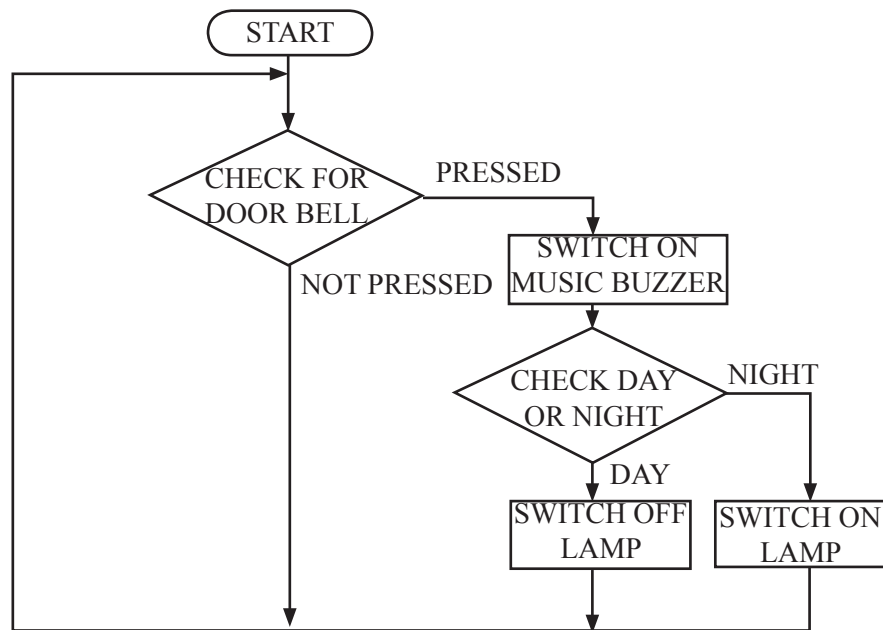
Application I

Water Level Controller Using 8085 Microprocessor

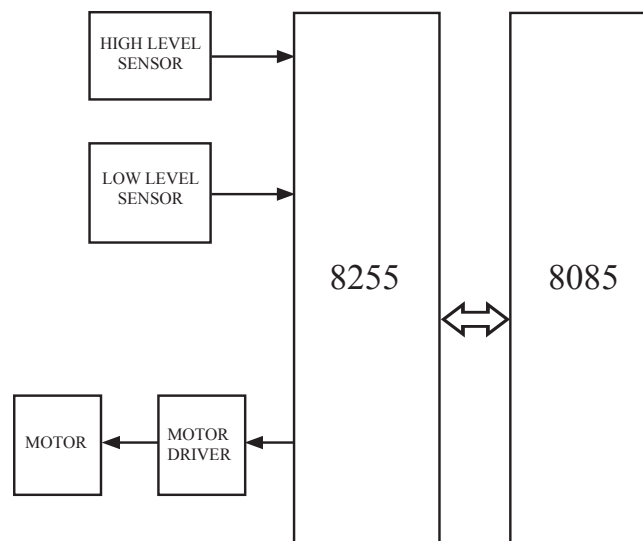
The water level controller is a reliable circuit. It performs the task of indicating and controlling the water level in the overhead water tanks. The level of the water is displayed in the LED Bar graph. The copper probes are used to sense the water level. The probes are inserted into the water tank which is to be monitored. This water-level controller-cum-alarm circuit is configured around the well-known 8 bit microprocessor 8085. It continuously monitors the overhead water level, displays it and also switches off the motor when the tank fills and it will automatically switch on the motor when the water level is low. The microprocessor will also indicate the water level over the LED display. All the input and output functions are done through the programmable peripheral interface IC 8255.



Water level controller - Circuit diagram



Flow chart



Block diagram

(Source: <http://www.8085projects.info/forum/tag.asp?tag=water+level+controller+using+8085+microprocessor>)

Questions

1. How does water level controller function?

Answer:

The water level controller takes over the task of indicating and controlling the water level in the overhead water tanks. The level of the water gets displayed in the LED bar graph. The copper probes are inserted into the water tank which is to be monitored. The water-level controller-cum-alarm circuit is configured around the 8 bit microprocessor 8085. It then monitors the overhead water level and displays it. It also switches off the motor when the tank fills and automatically switches on the motor when the water level is low.

2. What are the inputs used in water level controller?

Answer:

The inputs used in water level controller are LED display, copper probes and motor.

3. What is the role of 8085 microprocessor in water level controller?

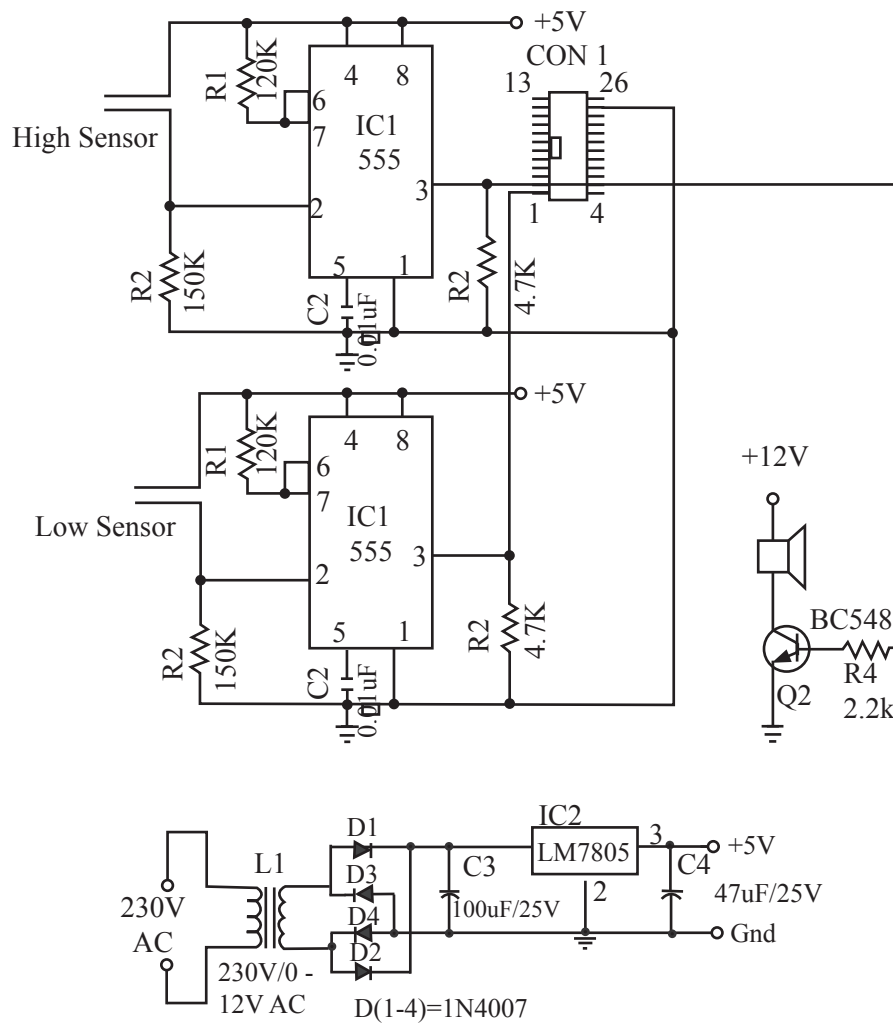
Answer:

The water-level controller-cum-alarm circuit is configured around the 8 bit microprocessor 8085. The 8085 microprocessor indicates the water levels over the LED display. All the input and output functions are done through the programmable peripheral interface IC 8255.

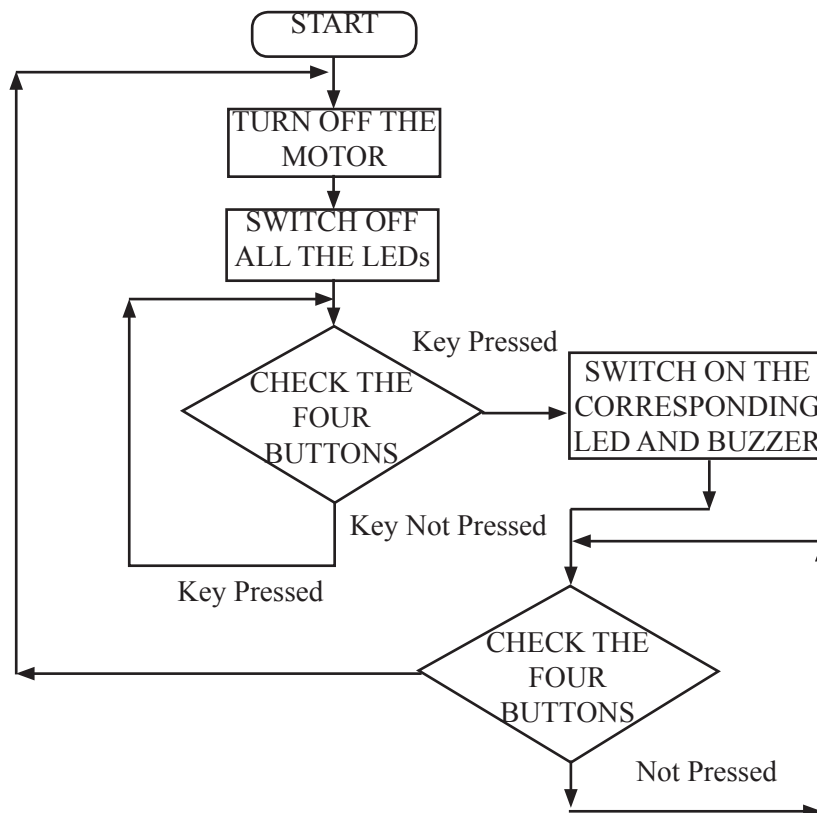
Application II

Door Bell with Security Feature

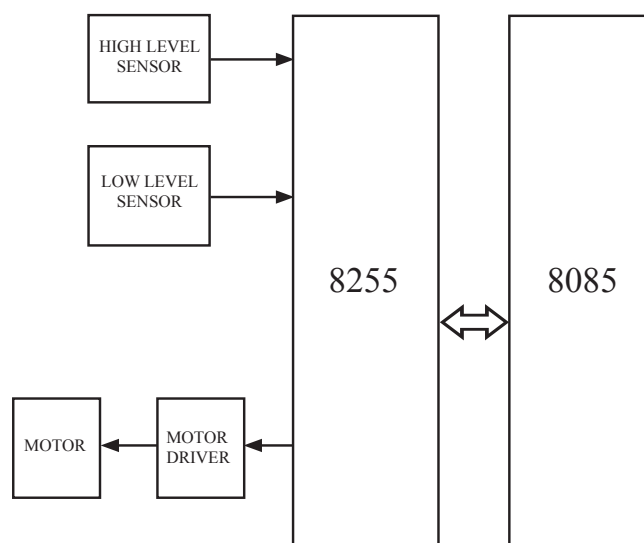
This Project “Door Bell with Security Feature” works in such a way that when someone presses the switch during night, not only the bell rings but the bulb connected to it also glows. In the day time, the bell rings but the bulb does not glow. The above systems are controlled by the popular 8 bit Intel microprocessor 8085A. It monitors the bell switch until it is pressed, and also identifies whether it is day or night. The same system continues until the reset switch is pressed. The monitoring for Microprocessor is done by the PPI (Programmable Peripheral Interface) IC8255.



Door bell circuit diagram



Door bell flow chart



Door bell block diagram

(Source: <http://www.8085projects.info/proj.asp?ID=5>)

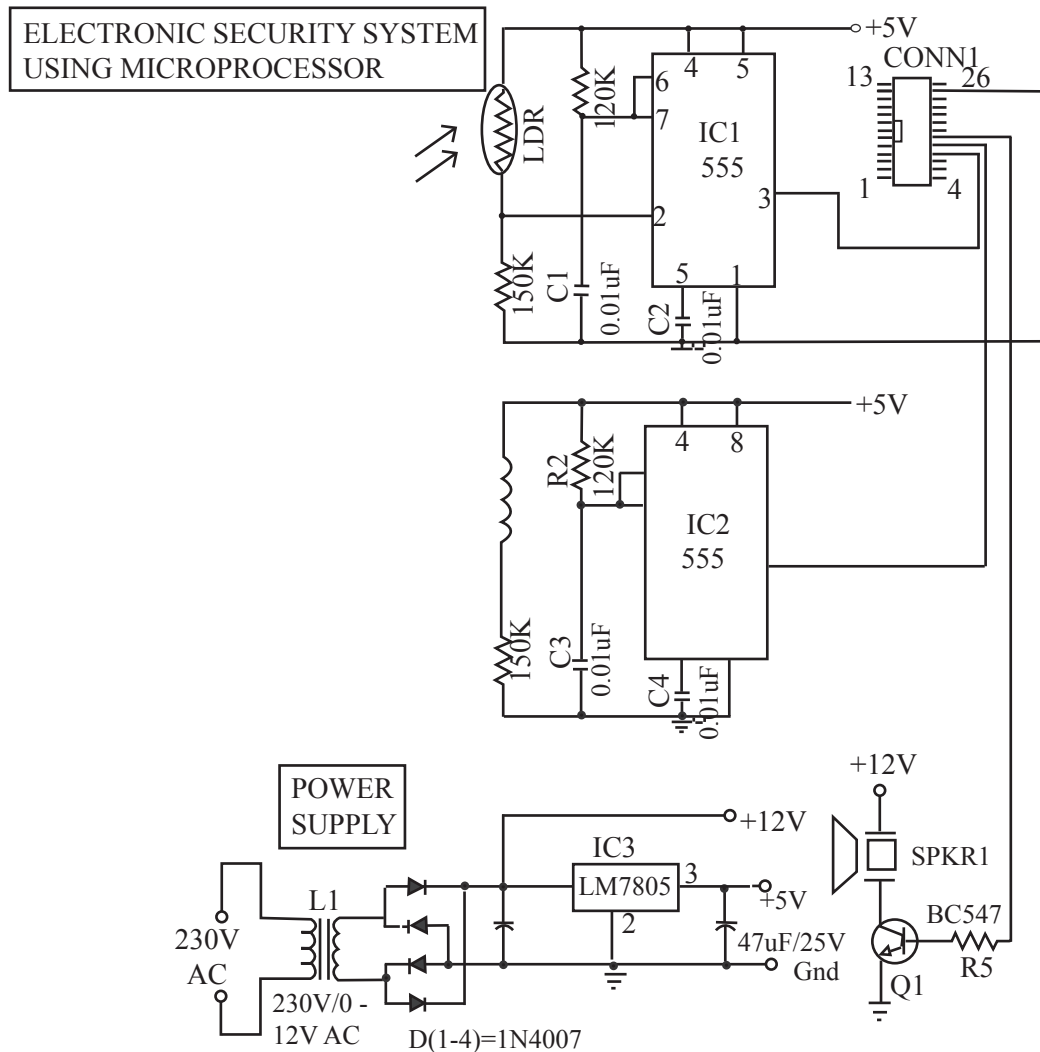
Questions

1. How does door bell function?
2. What are the inputs used in door bell functioning?
3. What is the significance of 8085 microprocessor in door bell security feature?

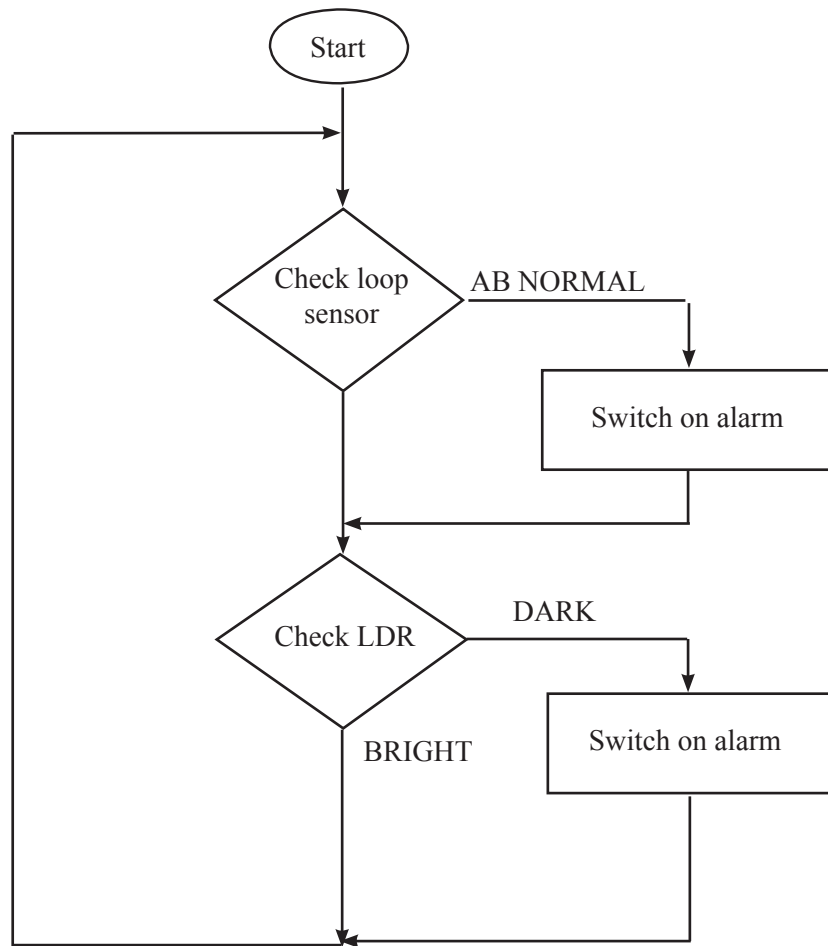
Application III

Electronic Security System

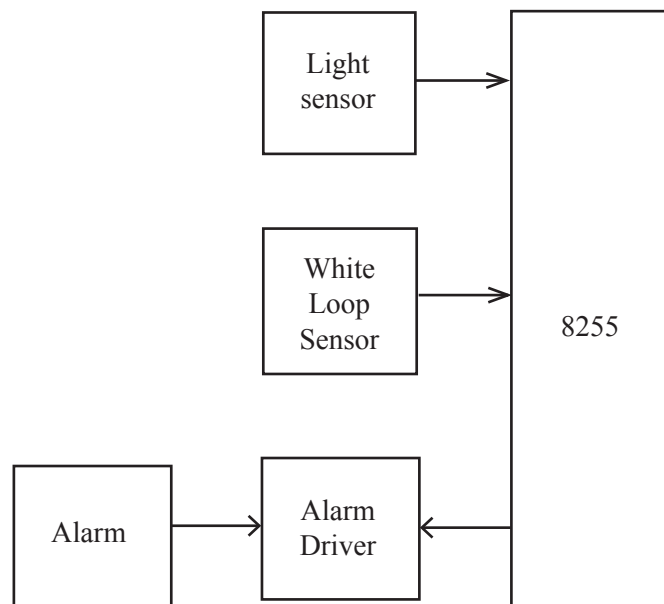
This reliable Electronic Security system can be used in banks, factories, commercial establishments etc. The system comprises of two monitoring systems, one is a light detecting system and the other is a Loop sensor. When a light falls on the LDR (Light Depending Resistor), the system generates an alarm sound and when the loop wire gets opened at any condition, the alarm is switched on. The above system is monitored and controlled by the 8 bit microprocessor 8085A and using the ports of 8255. The microprocessor continuously monitors the loop wire and the LDR through the ports of 8255. If there is an abnormal condition is detected then the alarm is switched on until the fault is cleared. The monitoring for microprocessor is done by the PPI (Programmable Peripheral Interface) IC8255.



Electronic security system - Circuit diagram



Flow chart



Block diagram

(Source: <http://www.8085projects.info/forum/tag.asp?tag=8085+microprocessor>)

Questions

1. What is the work procedure of electronic security system?
2. What are the inputs used in electronic security system?
3. What is the function of 8085 microprocessor in electronic security system?

Bibliography

References

- *8085 INSTRUCTION SET* [Pdf] Available at: <http://engeletrica.sobral.ufc.br/professores/marcelo/Micro/8085_is_details.pdf> [Accessed 30 May 2013].
- *8085 Microprocessor* [Pdf] Available at: <http://www.nptel.iitm.ac.in/courses/Webcourse-contents/IISc-BANG/Microprocessors%20and%20Microcontrollers/pdf/Lecture_Notes/LNm1.pdf> [Accessed 30 May 2013].
- *8086 Microprocessor Architecture: Tutorial 4 On 8086 Architecture* [Video online] Available at: <<http://www.youtube.com/watch?v=WpOJDw6C0Mk>> [Accessed 30 May 2013].
- *An Introduction to Logic Gates* [Video online] Available at: <<http://www.youtube.com/watch?v=95kv5BF2Z9E>> [Accessed 29 May 2013].
- Bamakhrama, M., 2008. *Embedded Multiprocessor System-on-chip for Access Network Processing*, GRIN Verlag.
- *Binary Number System - One's Complement, Two's Complement, Conversion* [Video online] Available at: <<http://www.youtube.com/watch?v=INGWoFHoLoA>> [Accessed 29 May 2013].
- *Boolean Algebra Part 1* [Video online] Available at: <<http://www.youtube.com/watch?v=IEgyiudC-nk>> [Accessed 29 May 2013].
- Brey, 2008. *The Intel Microprocessors: 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro Processor, Pentium II, Pentium III, Pentium 4*, 8th ed. Pearson Education India.
- *Chapter 3 Boolean Algebra and Logic Gates* [Pdf] Available at: <<http://larc.ee.nthu.edu.tw/~cww/n/228/03.pdf>> [Accessed 29 May 2013].
- Dijkstra, E. W., *Lecture 2: THE multiprogramming system* [Video online] Available at: <<http://www.youtube.com/watch?v=Uiew30G2D1c>> [Accessed 30 May 2013].
- Dr. Mall, R., 2011. *Mod-01 Lec-17 Real-Time Task Scheduling on Multiprocessors and Distributed Systems* [Video online] Available at: <<http://www.youtube.com/watch?v=kgcUVZARftQ>> [Accessed 30 May 2013].
- Dr. Mall, R., 2011. *Mod-01 Lec-18 Real-Time Task Scheduling on Multiprocessors and Distributed Systems (Contd.)* [Video online] Available at: <<http://www.youtube.com/watch?v=kgcUVZARftQ>> [Accessed 30 May 2013].
- Fawwaz, W., *The 8085 microprocessor* [Pdf] Available at: <<http://www.uotechnology.edu.iq/dep-cse/lectures/3/control/mico.pdf>> [Accessed 30 May 2013].
- Fisher, A. J., Zaraboschi, P. & Young, C., 2005. *Embedded Computing: A Vliw Approach To Architecture, Compilers And Tools*, Elsevier.
- Flynn, M. I. & MacHoes, M. A., 1997. *Understanding operating systems*, 2nd ed. Taylor & Francis.
- Godse, D. A. & Godse, A. P., 2007. *Introduction To Microprocessor*, Technical Publications.
- Godse, D. A. & Godse, A. P., 2008. *Microprocessor Techniques*, Technical Publications.
- Holdsworth, B. & Woods, C., 2002. *Digital Logic Design*, 4th ed. Newnes.
- *I/O Modules* [Pdf] Available at: <http://shpat.com/docs/grayhill/io_modules_individual.pdf> [Accessed 30 May 2013].
- *Input/Output Module* [Pdf] Available at: <<http://www.docs.is.ed.ac.uk/skills/documents/3663/SPSSInput-OutputModule.pdf>> [Accessed 30 May 2013].
- Jerraya, A. A. & Wolf, W. H., 2005. *Multiprocessor Systems on Chips*, 2nd ed. Morgan Kaufmann.
- Jones, T. C., 2006. *Programmable Logic Controllers: The Complete Guide to the Technology*, Brilliant-Training.
- Kotz, D., *INTRODUCTION TO MULTIPROCESSOR I/O ARCHITECTURE* [Pdf] Available at: <<http://www.cs.dartmouth.edu/~dfk/papers/kotz-pioarch.pdf>> [Accessed 30 May 2013].
- *Lecture10 Multiprogramming and Fixed Partitions* [Video online] Available at: <http://www.youtube.com/watch?v=jKxFivAja_8> [Accessed 30 May 2013].

- Mathur, S., 2010. *Microprocessor 8085 And Its Interfacing*, PHI Learning Pvt. Ltd.
- *MICROPROCESSOR 8085* [Pdf] Available at: <<http://itsnka.files.wordpress.com/2010/10/mpmanual-first-ver.pdf>> [Accessed 30 May 2013].
- *NUMBER SYSTEMS AND CODES* [Pdf] Available at: <<http://www.inf.fu-berlin.de/lehre/WS00/19504-V/Chapter1.pdf>> [Accessed 29 May 2013].
- *Operating Systems* [Pdf] Available at: <<http://www.uow.edu.au/~nabg/ABC/C3.pdf>> [Accessed 30 May 2013].
- Patrick, R. D., Fardo, W. S. & Chandra, V., 2008. *Electronic Digital System Fundamentals*, The Fairmont Press, Inc.
- Prof. Bhor, H., Prof. Rote, U. & Prof. Shinde, U., *Operating System* [Pdf] Available at: <http://www.mu.ac.in/myweb_test/MCA%20study%20material/OS%20-%20PDF.pdf> [Accessed 30 May 2013].
- Prof. Kleitz., 2011. *Digital Electronics: Textbook Preface* [Video online] Available at: <<http://www.youtube.com/watch?v=nM25ejvGn0Y&list=PLE7E5C88AA6AEA0A2>> [Accessed 29 May 2013].
- Prof. Kumar, A., *Lecture - 33 Input / Output Subsystem: Introduction* [Video online] Available at: <<http://www.youtube.com/watch?v=0XybwAbup-w>> [Accessed 30 May 2013].
- Prof. Kumar, A., *Lecture -3 Instruction Set Architecture* [Video online] Available at: <<http://www.youtube.com/watch?v=HbsuwpJgKao>> [Accessed 30 May 2013].
- Prof. Pal, A., 2012. *lec 1 - Introduction to Microprocessors & Microcontrollers* [Video online] Available at: <<http://www.youtube.com/watch?v=liRPtvj7bFU>> [Accessed 30 May 2013].
- Prof. Pal, A., 2012. *lec 2 - Architecture and Organization of 8085* [Video online] Available at: <<http://www.youtube.com/watch?v=p4RcMLFIr5o>> [Accessed 30 May 2013].
- Prof. Pal, A., 2012. *lec 4 - Instruction Set : Vocabulary of the Machine* [Video online] Available at: <http://www.youtube.com/watch?v=tjZ2Mh_MV6g> [Accessed 30 May 2013].
- Prof. Raman, S., *Lecture 25 - Interrupt Driven I/O* [Video online] Available at: <<http://www.youtube.com/watch?v=C02weCM9yWA>> [Accessed 30 May 2013].
- Rajaraman, V. & Radhakrishnan, T., 2006. *DIGITAL LOGIC AND COMPUTER ORGANIZATION*, PHI Learning Pvt. Ltd.
- Ryu, J. H., *Boolean Algebra and Logic Gates* [Pdf] Available at: <http://robot.kut.ac.kr/download/dd/chapter2_boolean_algebra.pdf> [Accessed 29 May 2013].
- Srinath, N. K., 2005. *8085 MICROPROCESSOR: PROGRAMMING AND INTERFACING*, PHI Learning Pvt. Ltd.
- Subramanyam, M.V. & Bhatia, B., 2008. *Basic Digital Electronics*, Laxmi Publications, Ltd.
- *System bus explanation* [Video online] Available at: <<http://www.youtube.com/watch?v=A71WOpA6leU>> [Accessed 30 May 2013].
- Tatla, G. H., *INSTRUCTION SET OF 8085* [Pdf] Available at: <<http://www.eazynotes.com/notes/microprocessor/Slides/instruction-set-of-8085.pdf>> [Accessed 30 May 2013].
- *Tutorial On Introduction to 8085 Architecture and Programming* [Pdf] Available at: <<http://www.phy.davidson.edu/fachome/dmb/py310/8085.pdf>> [Accessed 30 May 2013].
- *UNIT 1 MULTIPROCESSOR SYSTEMS* [Pdf] Available at: <<http://dspace.sngce.ac.in/bitstream/handle/123456789/1953/Media-B3U1mcs-041.pdf?sequence=10>> [Accessed 30 May 2013].
- *UNIT I NUMBER SYSTEM AND BINARY CODES* [Pdf] Available at: <http://www.b-u.ac.in/sde_book/bca_fund.pdf> [Accessed 29 May 2013].
- Yadav, A., 2008. *Microprocessor 8085, 8086*, Firewall Media.

Recommended Reading

- Balabanian, N. & Carlson, B., 2007. DIGITAL LOGIC DESIGN PRINCIPLES, John Wiley & Sons.
- Ball, S., 2002. *Embedded Microprocessor Systems: Real World Design*, 3rd ed. Newnes.
- Bhattacharya, J., *Rudiments of Computer Science*, Academic Publishers.
- Das, S., 2010. *A Complete Guide to Computer Fundamentals*, Laxmi Publications, Ltd.
- El-Abiad, H. A., 2006. *Adv Microprocessors & Periph 2E, 2nd ed.* Tata McGraw-Hill Education.
- Ghosh, P. K. & Sridhar, P. R., 2004. *Introduction To Microprocessors For Engineers And Scientists 2Nd Ed.*, 2nd ed. PHI Learning Pvt. Ltd.
- Godse, A. P. & Godse, D. A., 2009. *Microprocessor, Microcontroller And Embedded Systems*, Technical Publications.
- Godse, D. A. & Godse, A. P., 2008. *Microprocessors And Its Applications*, Technical Publications.
- Godse, D. A. & Godse, A. P., 2010. *Microprocessor & Microcontroller*, Technical Publications.
- *Introduction To Computer Science*, Pearson Education India.
- Kempf, T., Ascheid, G. & Leupers, R., 2011. *Multiprocessor Systems on Chip: Design Space Exploration*, Springer.
- Kumar, A. A., 2010. *Switching Theory And Logic Design*, PHI Learning Pvt. Ltd.
- Lipovski, J. G., 1999. *Single and Multi-Chip Microcontroller Interfacing: For the Motorola 6812*, Academic Press.
- Mathivanan, N., 2003. *MICROPROCESSORS, PC HARDWARE AND INTERFACING*, PHI Learning Pvt. Ltd.
- Nagoorkani, 2013. *8085 Microprocessors & Its Application*, 3rd ed. Tata McGraw-Hill Education.
- Rafiquzzaman, M., 2005. *Fundamentals of Digital Logic and Microcomputer Design*, 5th ed. John Wiley & Sons.
- Rajaraman, 2004. *Introduction To Information Technology*, PHI Learning Pvt. Ltd..
- Ramesh, V., 2010. *Principles of Operating Systems*, Laxmi Publications, Ltd.
- Seeger, A. S., 1988. *Introduction to Microprocessors With the Intel 8085*, Harcourt Brace Jovanovich.
- Swamy, G. T., 2006. *Microprocessor (8085) Lab Manual*, Firewall Media.
- Tocci, J. R., 1980. *Digital Systems: Principles and Applications*, Pearson Education India.
- Udaya Kumar, K., 2008. *The 8085 Microprocessor: Architecture, Programming and Interfacing*, Pearson Education India.
- Wadhwa, A., 2010. *Microprocessor 8085: Architecture Programming And Interfacing*, PHI Learning Pvt. Ltd.

Self Assessment Answers

Chapter I

1. a
2. b
3. c
4. d
5. a
6. b
7. c
8. d
9. a
10. b

Chapter II

1. a
2. b
3. c
4. d
5. a
6. b
7. c
8. d
9. a
10. b

Chapter III

1. a
2. b
3. d
4. a
5. a
6. a
7. b
8. b
9. c
10. a

Chapter IV

1. a
2. a
3. b
4. c
5. d
6. d
7. a
8. d
9. d
10. b

Chapter V

1. a
2. c
3. c
4. a
5. d
6. d
7. a
8. c
9. b
10. d

Chapter VI

1. a
2. b
3. b
4. b
5. d
6. d
7. a
8. b
9. c
10. c

Chapter VII

1. b
2. a
3. c
4. d
5. a
6. c
7. a
8. d
9. c
10. c

Chapter VIII

1. a
2. b
3. c
4. a
5. c
6. b
7. a
8. d
9. d
10. a