LOVELY PROFESSIONAL UNIVERSITY

COMPUTER ORGANIZATION AND ARCHITECTURE/INTRODUCTION TO COMPUTER ORGANIZATION AND ARCHITECTURE

Edited By

Dr. Manmohan Sharma

ISBN: 978-93-87034-67-9

Printed by
EXCEL BOOKS PRIVATE LIMITED
Regd. Office: E-77, South Ext. Part-I, Delhi-110049
Corporate Office: 1E/14, Jhandewalan Extension, New Delhi-110055
+91-8800697053, +91-011-47520129
info@excelbooks.com/projects@excelbooks.com
internationalalliance@excelbooks.com
www.excelbooks.com

for
Lovely Professional University
Phagwara

CONTENTS

Unit 1:	Review of Basics of Digital Electronics *Anuj Sharma, Lovely Professional University*	1
Unit 2:	Devices Used in Digital Electronics *Sahil Rampal, Lovely Professional University*	23
Unit 3:	Data Representation and Data Transfer *Yadwinder Singh, Lovely Professional University*	43
Unit 4:	Computer Organization I *Avinash Bhagat, Lovely Professional University*	65
Unit 5:	Computer Organization II *Avinash Bhagat, Lovely Professional University*	81
Unit 6:	Control Unit *Pooja Gupta, Lovely Professional University*	95
Unit 7:	Central Processing Unit *Manmohan Sharma, Lovely Professional University*	111
Unit 8:	Addressing Modes *Ajay Kirani Khuswaha, Lovely Professional University*	125
Unit 9:	Computer Arithmetic I *Sarabjit Kumar, Lovely Professional University*	143
Unit 10:	Computer Arithmetic II *Avinash Bhagat, Lovely Professional University*	155
Unit 11:	Input/Output Organization *Ajay Kumar Bansal, Lovely Professional University*	165
Unit 12:	Memory Organization Concepts *Pooja Gupta, Lovely Professional University*	185
Unit 13:	Multiprocessors *Manmohan Sharma, Lovely Professional University*	205
Unit 14:	Introduction to Parallel Processing *Yadwinder Singh, Lovely Professional University*	229

SYLLABUS

Computer Organization and Architecture/Introduction to Computer Organization and Architecture

Objectives: The objectives of this course are:

- To understand how computers are constructed out of a set of functional units
- To understand how these functional units operate, interact and communicate
- To understand the factors and trade-offs that affect computer performance
- To understand concrete representation of data at the machine level
- To understand how computations are actually performed at the machine level
- To understand how problems expressed by humans are expressed as binary strings in a machine

DCAP502 COMPUTER ORGANIZATION AND ARCHITECTURE

Sr. No.	Description
1.	**Review of Basics of Digital Electronics:** Codes, logic gates, flip flops, registers, counters, multiplexer, demultiplexer, decoder, and encoder.
2.	**Integers Representation:** Signed Magnitude, 1s & 2s Complement) & Real numbers (Fixed point & Floating Point representation), Register Transfer and Micro operations: Register transfer language Bus & memory transfer, logic micro operation, shift micro operation, Arithmetic Logic Shift Unit
3.	**Basic Computer Organization:** Instruction codes, computer instructions, timing & control, instruction cycles
4.	**Memory reference instruction, Input/output & interrupts, Design of basic computer Control Unit:** Hardwired vs. micro programmed control unit, Control Memory, Address Sequencing, Micro program Sequencer
5.	**Central Processing Unit:** General register organization, stack organization, instruction format, Addressing Modes Data transfer & manipulation, program control, RISC, CISC.
6.	**Introduction to Parallel Processing:** Pipelining, Instruction pipeline, RISC Pipeline, Vector Processing
7.	**Computer Arithmetic:** Addition, Subtraction, Multiplication & Division Algorithm(s), Decimal arithmetic units & Operations.
8.	**Input-Output Organization:** Peripheral devices, I/O interface, data transfer schemes, program control, interrupt, DMA transfer, I/O Processor
9.	**Memory Organization Concepts:** Cache & Virtual memory
10.	**Multiprocessors:** Characteristics, Interconnection Structures, Interprocessor Communication and synchronization

DCAP206 INTRODUCTION TO COMPUTER ORGANIZATION & ARCHITECTURE

Sr. No.	Description
1.	**Tools for course understanding:** Awareness of ISA bus interface, a popular bus architecture used in IBM and compatible personal computer systems. **Digital Logic Circuits:** Digital computers, Logic gates, Boolean Algebra, Map Simplification, Half Adder, Full Adder, Flip flops – SR, JK, D, T, Edge triggered flip flops, Sequential Circuits
2.	**Digital Components:** Integrated circuits, Decoders – NAND gate decoder, Encoders, Multiplexers, Demultiplexers, Registers, Shift registers, Bidirectional Register with parallel load, Binary counters, Memory Unit – RAM, ROM, Types of ROMs
3.	**Data Representation:** Number systems – decimal, octal, hexadecimal, Complement – (r-1)'s complement, r's complement, Fixed point representation, floating point representation, Gray code, Decimal codes, alphanumeric codes, Error detection codes
4.	**Register Transfer and Micro-operations:** Register transfer language, Register transfer, Bus and memory transfers – three state bus buffers, Arithmetic micro-operations – binary adder, binary adder subtractor, binary incrementer, arithmetic circuit
5.	Logic micro-operations and its hardware implementation, Shift micro-operations and hardware implementation, Arithmetic Logic Shift unit, Hardware description languages
6.	**Basic Computer Organization and Design:** Instruction Codes, Stored program organization, Computer registers, Common bus system, Computer instructions, Timing and Control, Instruction cycle, Memory reference instructions, Input output and interrupt, complete design of basic computer
7.	**Central Processing Unit:** General register organization, control word, Stack organization, register stack, memory stack, Instruction formats – three address, two address, one address, zero address instructions, Addressing modes, Data transfer and manipulation, arithmetic, logical, bit manipulation, Program control, Reduced Instruction Set Computer (RISC), CISC characteristics
8.	**Input-Output Organization:** Input output interface, I/O bus and interface modules, I/O vs memory bus, Isolated vs Memory mapped I/O
9.	Asynchronous data transfer, handshaking, Programmed I/O, Interrupt-initiated I/O, Priority Interrupt – Daisy chaining, parallel priority, priority encoder, interrupt cycle, DMA controller and transfer
10.	**Memory Organization:** Memory hierarchy, RAM, ROM chips, memory address map, Associative memory, Cache memory, Virtual memory, Memory management hardware

Notes

Unit 1: Review of Basics of Digital Electronics

CONTENTS

Objectives
Introduction
1.1 Codes in Digital Electronics
1.1.1 Classification of Binary Codes
1.2 Logic Gates
1.3 Summary
1.4 Keywords
1.5 Self Assessment
1.6 Review Questions
1.7 Further Readings

Objectives

After studying this unit, you will be able to:

- Discuss the codes in digital electronics
- List the functions of different logic gates
- Discuss the truth table for all the logic gates

Introduction

Digital electronics is a field of computer science. It deals with devices that are used to carry out computer applications.

In digital electronics, we use two-state or binary logic. The two logic states are "0" (low) and "1" (high).

Computer uses binary number system for its operations. Digital electronics represents the two binary numbers, 1 and 0, using two voltage levels in a device called a logic gate. Sometimes the two states can also be represented using Boolean logic functions, "true" or "false" states, or using an "on" or "off" state.

Logic gates are important components of a digital circuit. A logic gate takes two inputs and generates a single output. In this unit we will discuss about the basic logic gates and their corresponding truth tables.

1.1 Codes in Digital Electronics

Basically, digital data is represented, stored, and transmitted as groups of binary digits which are called bits. The group of bits is known as binary code. Binary codes are used in computers as they allow computers to perform complex calculations quickly and efficiently. Binary codes are used in financial, commercial, and industrial applications. To understand how binary codes are applied in these fields, we first have to understand the classification of binary codes.

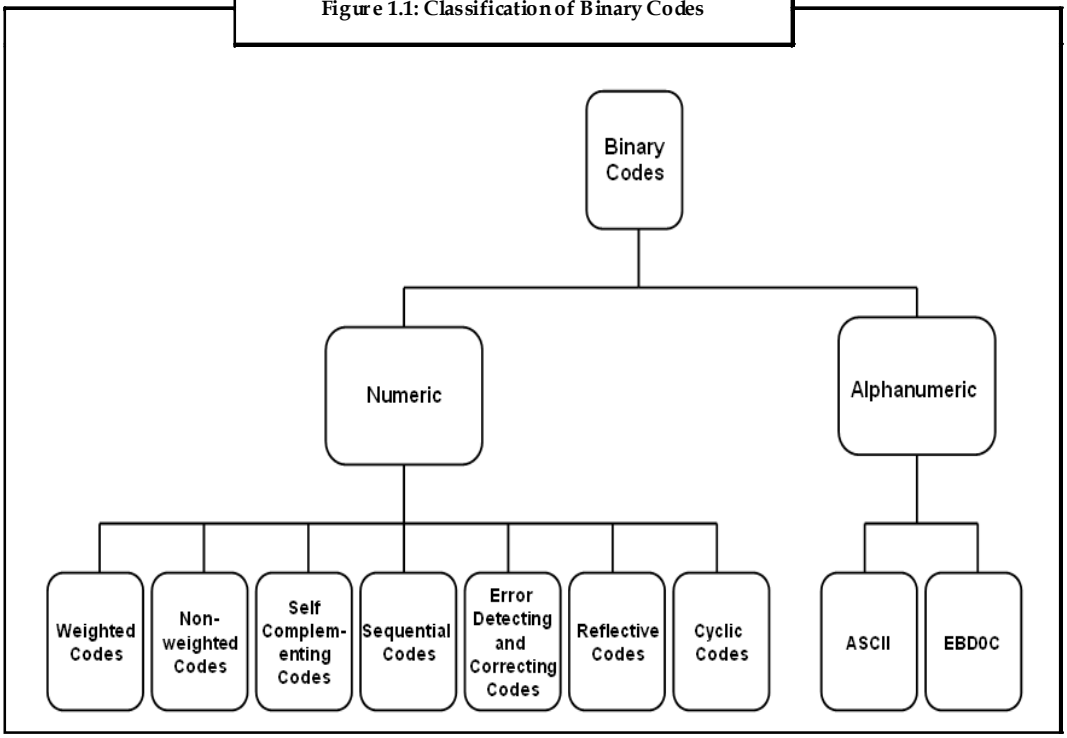
1.1.1 Classification of Binary Codes

Binary codes can be represented as numbers and letters of the alphabets as well as many special characters and control functions. They are classified as numeric or alphanumeric codes. Numeric codes are used to represent numbers, whereas alphanumeric codes are used to represent alphabetic letters and numerals.

Notes

The binary codes are classified as shown in figure 1.1.

Figure 1.1: Classification of Binary Codes



As mentioned earlier, numeric codes are used to represent numbers. The following are the numeric codes:

1. ***Weighted Binary Codes:*** Weighted binary codes are those which follow the positional weighting principles. In weighted codes, each position of the number represents a specific weight. For example, in decimal code, if the number is 345, then the weight of 3 is 100, 4 is 10, and 5 is 1. In the 8421 weighted binary code, each digit has a weight of 8, 4, 2 or 1 corresponding to its position.

Example: The codes 8421, 2421 and 5211 are examples of weighted binary codes.

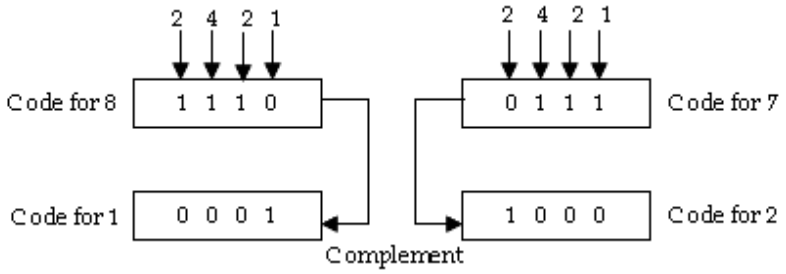
2. ***Non-Weighted Binary Codes:*** Non-weighted codes do not follow the positional weighting principles. In non-weighted codes, each digit position within the number does not have any fixed value.

*Example:*Excess-3 and Gray codes are examples of non-weighted codes. Excess-3 codes are used to express decimal numbers. The code can be derived from the natural BCD (8421) code by adding 3 (011 in binary) to the coded number. It is used in decimal arithmetic units. The excess-3 code of 1000 in 8421 is 1011. Gray codes represent each number in the sequence of integers {0...2^N-1} as a binary string of length N such that the adjacent integers have Gray code representations which differ in only one bit position.

3. ***Reflective Codes:*** A code is said to be reflective when the code for 9 is complement for the code for 0, 8 for 1, 7 for 2, 6 for 3, and 5 for 4.

Notes

Example: Codes 2421 and excess-3 are reflective. In code 2421: we know that the positional weights are 2, 4, 2 and 1. The following figure shows an example of reflective code where the complement of 8 is 1 and the complement of 7 is 2.



4. ***Sequential Codes:*** Sequential codes are codes in which the succeeding code is one binary number greater than its preceding code. This assists in mathematical data manipulation.

Example: 8421 and excess-3 codes are sequential codes.

5. ***Cyclic Codes:*** In cyclic codes, only one bit in the code changes at a time while moving from one number to the next. It is a non-weighted code, which means that the position of bit does not contain any weight.

Table1.1 shows the bit patterns assigned for gray code from decimal 0 to 10.

Table 1.1: Decimal Code to Gray Code Conversion

Decimal code	Gray code
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111

6. ***Error Detecting Codes:*** Whenever data is transmitted from one point to another, there is a probability that the data may get corrupted. In order to detect these data errors, some special codes called error detection codes are used.

Notes

7. ***Error Correcting Codes:*** These codes not only detect errors in data, but also correct them significantly. Error correction codes are a method by which a set of symbols can be represented such that even if any 1 bit of the representation gets accidentally flipped, we can still clearly identify the earlier symbol. Error correcting codes depend mainly on the notations and results of linear algebra. Error correction can be done using many methods like parity checking, Hamming codes, Single-bit Error Correction Double-bit Error Correction (SECDED), and so on.

Notes Error correcting codes are used in memories, networking, CDROM, and so on.

Example: Error correction using parity checking is as follows:

In parity check, an extra bit is added to the binary number to make all the digits in the binary number to sum up to an even or odd value. When the number adds up to an even number, we call it even parity and when the number sums up to an odd number, we call it odd parity. Consider the following two binary numbers:

1011010

1101011

Now, if we want to use even parity, we can add a parity bit to these numbers to obtain an even number as shown below:

01011010 4

11101011 6

If we want to use odd parity, we can add a parity bit to the number as follows:

11011010 5

01101011 5

Most of the modern applications use even parity. Let us consider even parity in our example.

The two binary numbers that need to be transmitted are:

01011010…….The even parity

11101011…….The even parity

Suppose during transmission the bits get changed as follows:

01111010 5

10101011 5

We can observe that the digits in the number sum up to odd numbers. Since we are using even parity, the computer knows that there is an error in the transmission.

8. ***Alphanumeric Codes***: These are codes that consist of both numbers and alphabets. The most commonly used alphanumeric codes are ASCII and EBCDIC.

   (a) ***EBCDIC Code:*** EBCDIC (Extended Binary Coded Decimal Interchange) is mainly used with large computer systems like mainframe computers. It is an 8-bit code which accommodates up to 256 characters.

Notes

(b) ***ASCII Code:*** ASCII (American Standard Code for Information Interchange) has become a standard alphanumeric code for microcomputers and computers. It is a 7-bit code, which represents 128 different characters. These 128 characters include 52 alphabets, which include A to Z and a to z, numbers from 0 to 9 (that is, 10 numbers), 33 special characters and symbols, and 33 control characters.

1.2 Logic Gates

A logic gate is an electronic device that makes logical decisions based on the different combinations of digital signals available on its inputs. A digital logic gate can have more than one input signal but has only one digital output signal.

Integrated Circuits or ICs can be grouped together into families according to the number of transistors or gates. Integrated circuits are categorized according to the number of logic gates or the circuit complexities within a chip.

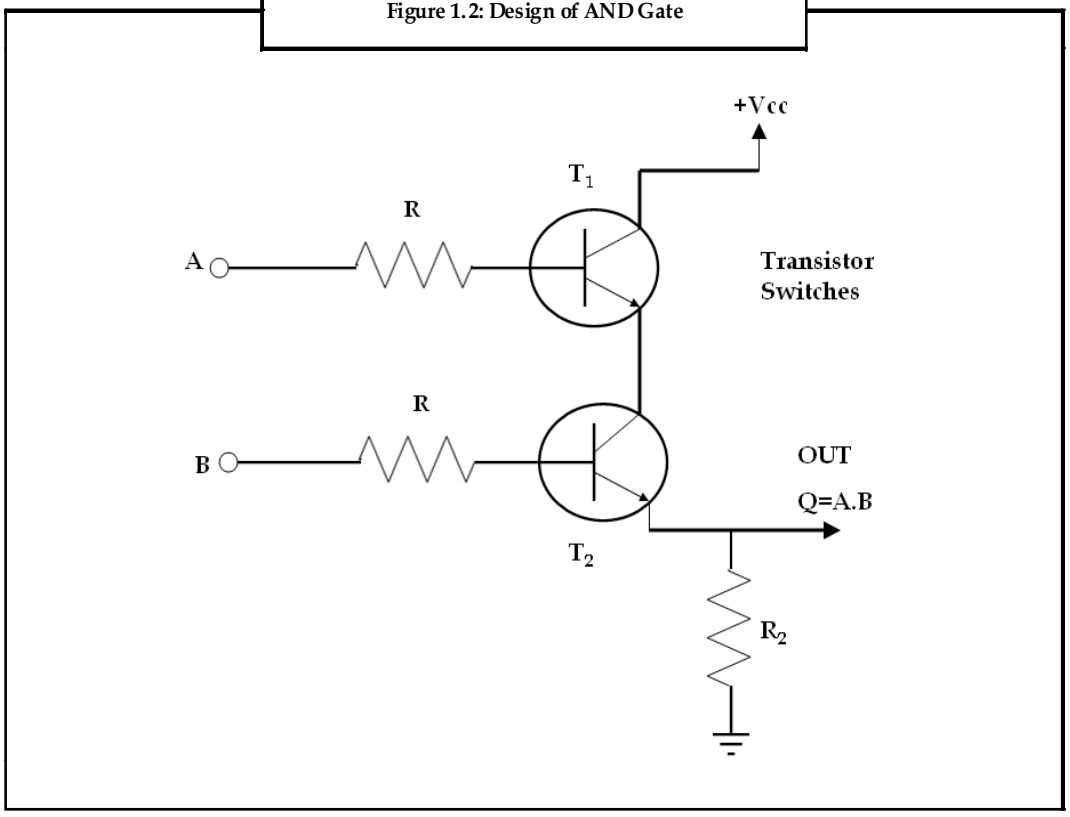
AND Gate

AND gate is a type of digital logic gate, which has an output that is normally at logic level "0" and goes "HIGH" to a logic level "1" when all of its inputs are at logic level "1". The output of AND gate returns "LOW" when any of its inputs are at a logic level "0".

The Boolean expression for AND gate is Q= A.B.

A simple 2-input logic AND gate can be constructed using RTL structure connected together as shown below in figure 1.2. Both transistors, T_1 and T_2 must be saturated "ON" to produce an output at Q. In the figure 1.2, we can observe that the output from T_1 is the input to T_2. Vcc is the input to transistor 1 and the final output, Q=A.B, is obtained from T_2.

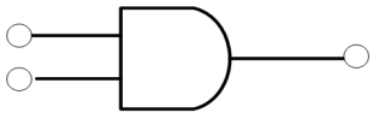
Figure 1.2: Design of AND Gate



Notes

The figure 1.3 shows the truth table and symbol of AND gate.

Figure 1.3: Logic Symbol and Truth Table of AND Gate



2-input And Gate

Boolean Expression Q = A.B

Truth Table

A	B	Q
0	0	0
1	0	0
0	1	0
1	1	1

Example: Commonly available digital logic AND gate ICs are:

TTL Logic Types

74LS08 Quad 2-input

74LS11 Triple 3-input

74LS21 Dual 4-input

CMOS Logic Types

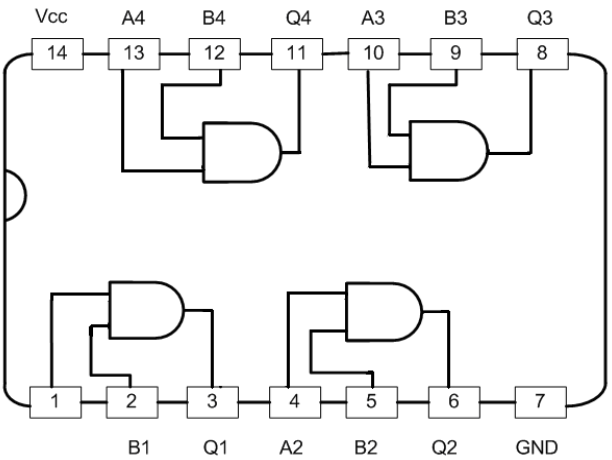
CD4081 Quad 2-input

CD4073 Triple 3-input

CD4082 Dual 4-input

Internal structure of IC 7408

Figure 1.4: Internal Structure of IC 7408



Notes

The figure 1.4 depicts how AND gates are placed within an IC. In the figure, pin number 1, 2, 4, 5, 9, 10, 12, and 13 are the inputs to the AND gate, while 3, 6, 8, and 11 are the AND outputs. Pin number 7 is connected to ground and pin number 14 is connected to the power supply.

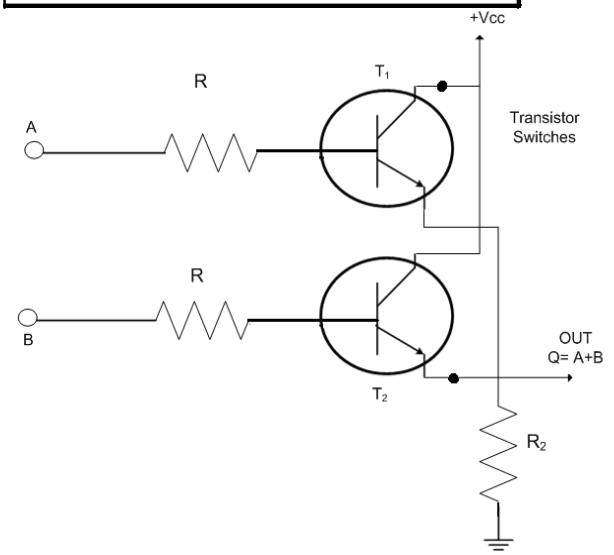
OR Gate

OR gate is a type of digital logic gate which has an output that is normally at logic level '0', but goes 'HIGH' to a logic level '1' when any of its inputs are at logic level '1'. The output of a logic OR gate returns 'LOW' again when all of its inputs are at a logic level '0'.

The Boolean expression for OR gate is denoted as Q = A+B.

A simple 2-input logic OR gate can be constructed using RTL structure connected together as shown in the figure 1.5 Either transistor T_1 or transistor T_2 must be saturated "ON" to produce an output at Q. In the figure 1.5, we can observe that Vcc is the input to both T_1 and T_2. The output of T_1 and T_2 constitutes the final output - Q=A+B.

Figure 1.5: Design of OR Gate



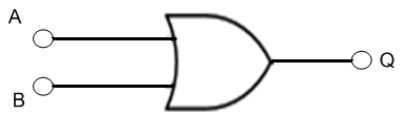
Example: Commonly available OR gate ICs are as follows:

TTL Logic Types	CMOS Logic Types
74L S32 Quad 2-input	CD4071 Quad 2-input
	CD4075 Triple 3-input
	CD4072 Dual 4-input

Notes

Figure 1.6 depicts logic symbol and truth table of OR gate.

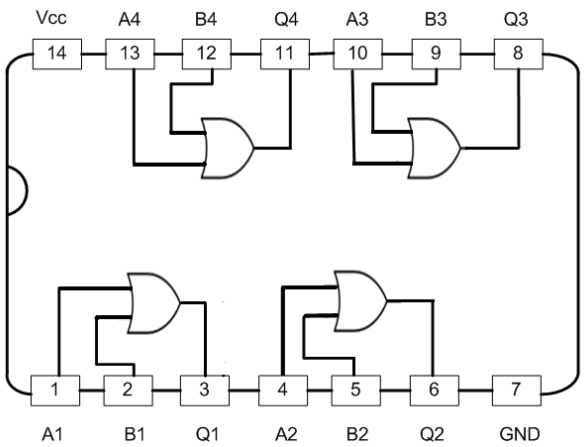
Figure 1.6: Logic Symbol and Truth Table of OR Gate



Truth Table		
A	B	Q
0	0	0
1	0	1
0	1	1
1	1	1

Figure 1.7 depicts how OR gates are placed within an IC.

Figure 1.7: Internal Structure of IC 7432



In the figure, pin number 1, 2, 4, 5, 9, 10, 12, and 13 are the inputs to the OR gate, while 3, 6, 8, and 11 are the OR outputs. Pin number 7 is connected to the ground and pin number 14 is connected to the power supply.

NOT Gate

In digital electronics, the NOT gate is also known as inverting buffer or a digital inverter element. A NOT gate is basically a single input device. It has an output level that is often at logic level '1'. However, it goes 'LOW' to a logic level '0' whenever the single input is at logic level '1'. The output from a NOT gate returns 'HIGH' when its input is at logic level '0'.

The Boolean expression of NOT gate is $Q=\overline{A}$

Notes

A simple 2-input logic NOT gate can be constructed using a RTL structure as shown in figure 1.8 with the input connected directly to the transistor base. If the transistor is saturated 'ON', we receive an inversed output 'OFF' at Q.

Figure 1.8: Design of NOT Gate

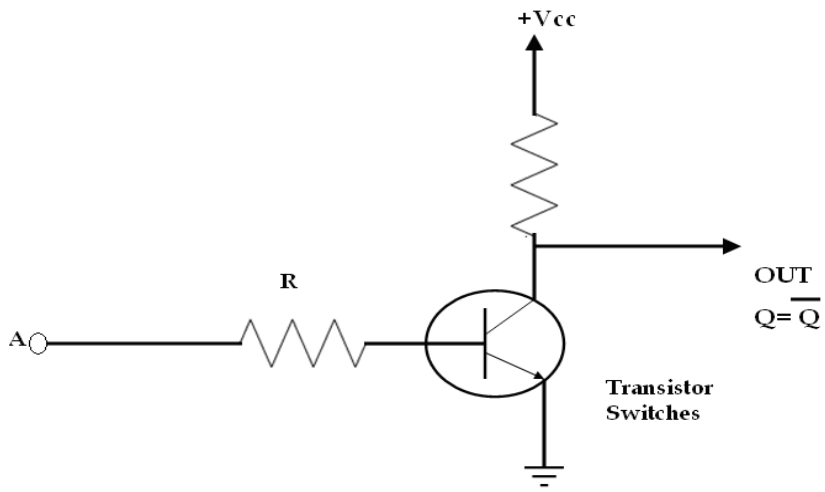
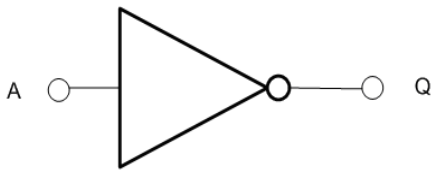


Figure 1.9 depicts the logic symbol and the truth table of a NOT gate.

Figure 1.9: Logic Symbol and Truth Table of NOT Gate



Inverter or NOT Gate

Truth Table	
A	Q
0	1
1	0

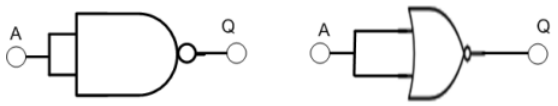
Boolean Expression Q = not A or $\overline{A}$

Notes

NAND and NOR Gate Equivalents for NOT Gate

A NOT gate can be constructed using standard NAND and NOR gates by connecting all their inputs together to a common input signal as shown in figure 1.10.

Figure 1.10: NAND and NOR Gate

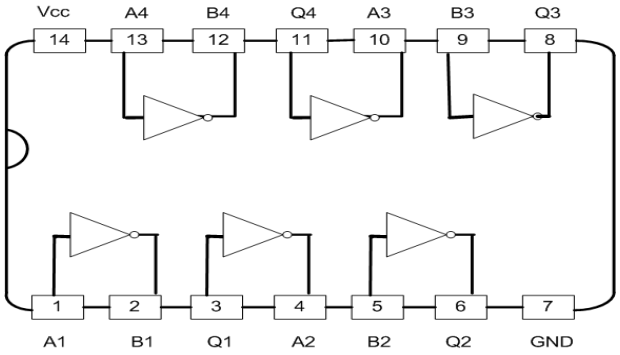


Example: Commonly available logic NOT gate and Inverter IC's are as follows:

TTL Logic Types	CMOS Logic Types
74LS04 Hex Inverting NOT Gate	CD4009 Hex Inverting NOT Gate
74LS04 Hex Inverting NOT Gate	CD4069 Hex Inverting NOT Gate
74LS14 Hex Schmitt Inverting NOT Gate	
74LS1004 Hex Inverting Drivers	

The figure 1.11 depicts how NOT gates are placed within an IC.

Figure 1.11: Internal Structure of IC 7404



In the figure, pin number 1, 3, 5, 9, 11, and 13 are the inputs to the NOT gate, while 2, 4, 6, 8, 10, and 12 are the NOT outputs. Pin number 7 is connected to ground and pin number 14 is connected to the power supply.

Universal Gates

NAND gate and NOR gate are called universal gates because these gates can be connected together in various combinations to form other gates like AND, OR, and NOT.

NAND Gate

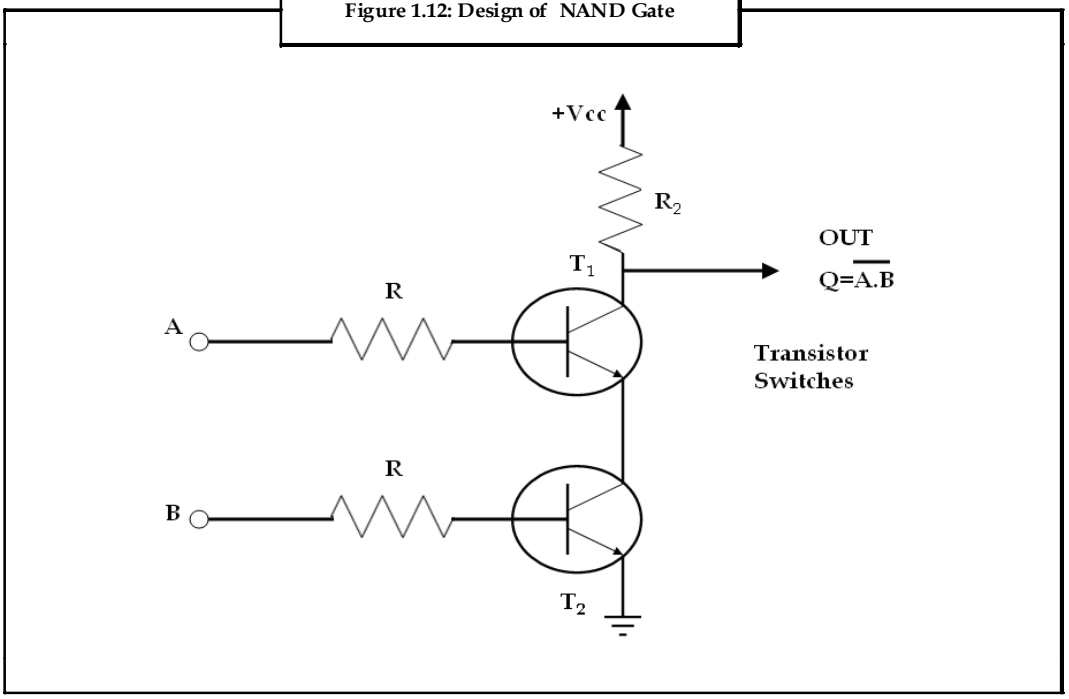
NAND gate is a combination of AND gate with an inverter or NOT gate connected together in series. NAND gate has an output that is normally at logic level '1' and only goes 'LOW' to logic level '0' when all of its inputs are at logic level "1".

The Boolean expression of NAND gate is Q= $\overline{A.B}$

Notes

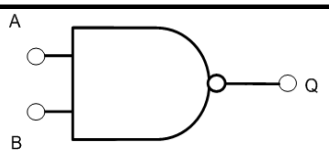
A simple 2-input logic NAND gate can be constructed using RTL structure connected together as shownin figure 1.12, with the inputs connected directly to the transistor bases. Either transistor T_1 or T_2 must be cut-off 'OFF' to receive an output at Q. Figure 1.12 depicts a design of NAND gate. In figure 1.12, we can observe that the output of transistor T_1 is the input to transistor T_2. The output, $Q = \overline{A.B}$ is received from T_1.

Figure 1.12: Design of NAND Gate



The figure 1.13 illustrates the logic symbol and truth table of NAND gate.

Figure 1.13: Logic Symbol and Truth Table of NAND Gate



Truth Table

A	B	Q
0	0	1
1	0	1
0	1	1
1	1	0

Did u know? NAND gate and NOR gate are called universal gates because these gates can be connected together in various combinations to form other gates like AND, OR, and NOT gates.

Notes

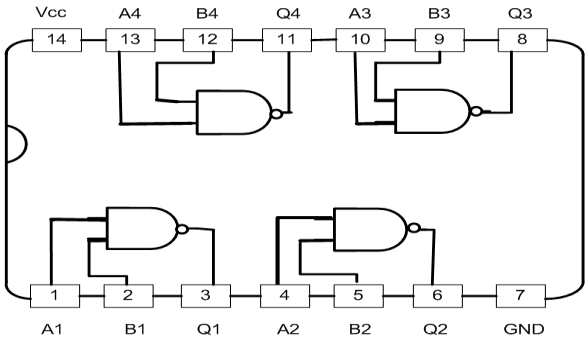
The various logic gates formed using NAND gates are as follows:

Example: Commonly available logic NAND gate ICs are as follows:

TTL Logic Types	CMOS Logic Types
74LS00 Quad 2-input	CD4011 Quad 2-input
74LS10 Triple 3-input	CD4023 Triple 3-input
74LS20 Dual 4-input	CD4012 Dual 4-input
74LS30 Single 8-input	

The figure 1.14 depicts how NAND gates are placed within an IC.

Figure 1.14: Internal Structure of IC 7400



In the figure, pin number 1, 2, 4, 5, 9, 10, 12, and 13 are the inputs to the NAND gate, while 3, 6, 8, and 11 are the NAND outputs. Pin number 7 is connected to ground and pin number 14 is connected to the power supply.

NOR Gate

NOR gate is a combination of OR gate with a NOT gate connected together in a series. The NOR gate has an output that is normally at logic level '1' and only goes 'LOW' to logic level '0' when any of its inputs are at logic level '1'.

The Boolean expression of NOR gate is $Q= \overline{A+B}$

To construct a 2-input logic NOR gate, an RTL Resistor-transistor switches can be used. The RTL must be connected together as shown in the figure 1.15 with the inputs connected directly to the transistor bases. Both transistors, T_1 and T_2 must be cut-off 'OFF' to receive an output at Q. Figure 1.15 shows the design of NOR gate.

Notes

In figure 1.15, we can observe that Vcc is the input to both transistors T_1 and T_2. The output of T_1 and T_2 is grounded. The inputs of both T_1 and T_2 constitute the output, Q=A+B.

Figure 1.15: Design of NOR Gate

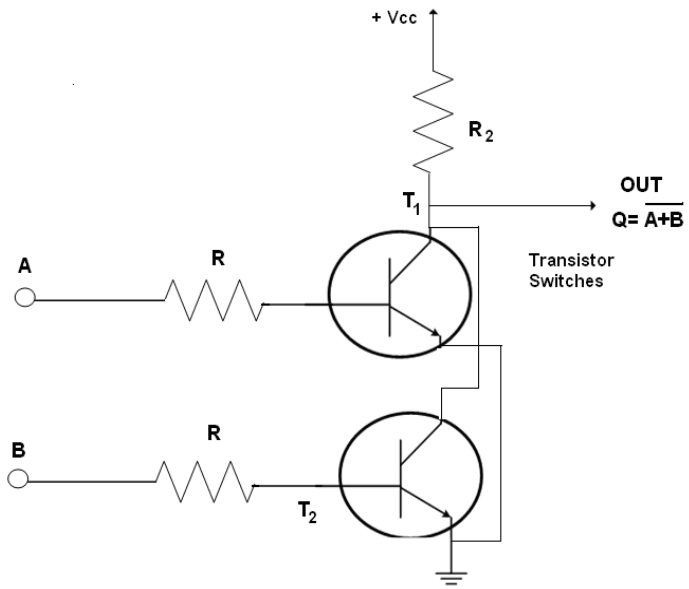
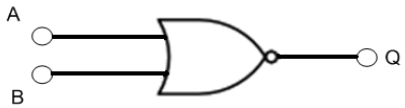


Figure 1.16 depicts the logic symbol and truth table of NOR gate.

Figure 1.16: Logic Symbol and Truth Table of NOR Gate



Truth Table		
A	B	Q
0	0	1
1	0	0
0	1	0
1	1	0

Example: Commonly available NOR gate ICs are as follows:

TTL Logic Types	CMOS Logic Types
74LS02 Quad 2-input	CD4001 Quad 2-input
74LS27 Triple 3-input	CD4025 Triple 3-input
74LS260 Dual 4-input	CD4002 Dual 4-input

Notes

Figure 1.17 shows the internal structure of IC 7402.

Figure 1.17: Internal Structure of IC 7402

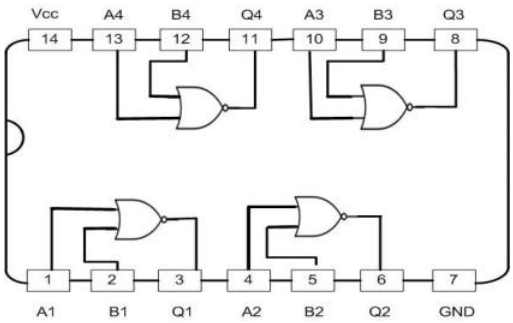


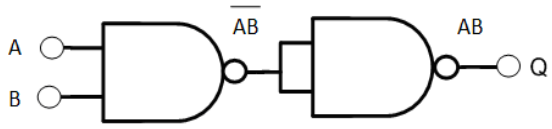
Figure 1.17 depicts how NOR gates are placed within an IC. In the figure 1.17, pin number 1, 2, 4, 5, 9, 10, 12, and 13 are the inputs to the NOR gate, while 3, 6, 8, and 11 are the NOR outputs. Pin number 7 is connected to ground and pin number 14 is connected to the power supply.

Application of Universal Gates

The NAND and NOR gates can be used to construct other forms of gates. The figures 1.18(a), 1.18 (b), 1.18 (c) depict the application of these universal gates.

The various logic gates formed using NAND gates and their truth tables are as follows:

Figure 1.18 (a): Construction of AND Gate Using NAND Gate



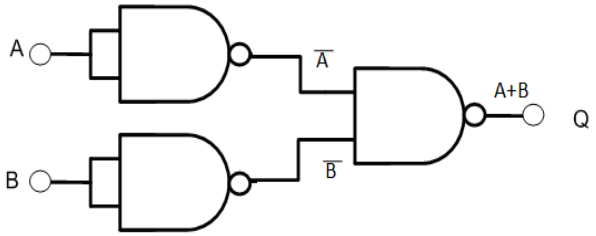
Truth Table		
A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

Notes

Figure 1.18 (b): Construction of NOT Gate Using NAND Gate

Truth Table	
A	$\overline{A}$
0	1
1	0

Figure 1.18 (c): Construction of OR Gate Using NAND Gate



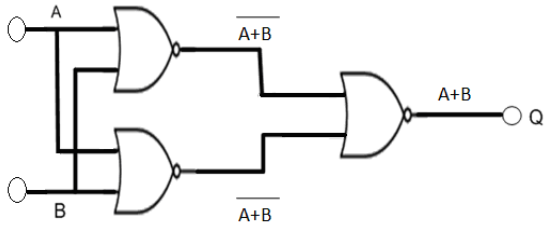
Truth Table		
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

Notes

NOR gates

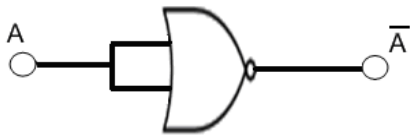
The construction of logic gates using NOR gates and their truth tables are given in the below figures.

Figure 1.19 (a): Construction of OR Gate Using NOR Gate



Truth Table		
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

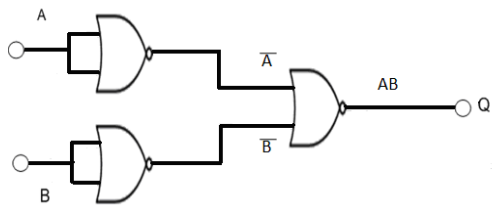
Figure 1.19 (b): Construction of NOT Gate Using NOR Gate



Truth Table	
A	$\overline{A}$
0	1
1	0

Notes

Figure 1.19 (c): Construction of AND Gate Using NOR Gate



Truth Table		
A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

Exclusive-OR Gate

The output of an Exclusive-OR gate goes 'HIGH' when its two input terminals are at different logic levels with respect to each other and they can be at logic level '1' or both at logic level '0'.

The Boolean expression is Q= (A $\oplus$ B) = A.B+A.B

Figure 1.20: Logic Symbol and Truth Table of Exclusive OR Gate

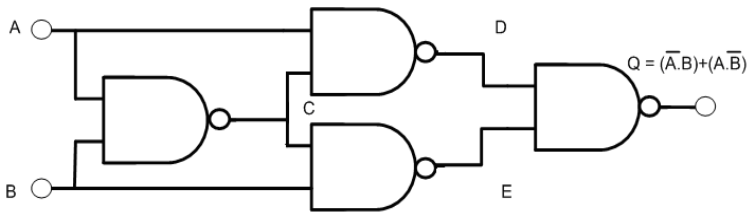
Truth Table		
A	B	Q
0	0	0
1	0	1
0	1	1
1	1	0

One of the main disadvantages of implementing the Ex-OR function is that it contains three different types of logic gates OR, NAND, and AND within its design.

Notes

One simpler way of producing the Ex-OR function from a single gate is to use NAND gate as shown below.

Figure 1.21: Ex-OR Gate Equivalent Circuit using NAND Gate

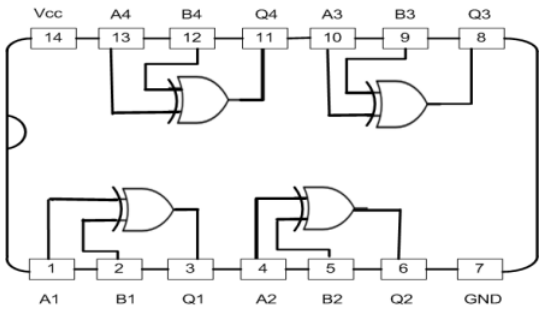


Example: Some of the commonly available Exclusive-OR gate ICs are as follows:

TTL Logic Types	CMOS Logic Types
74LS86 Quad 2-input	CD4030 Quad 2-input

The figure 1.22 depicts how Ex-OR gates are placed within an IC.

Figure 1.22: Internal Structure of IC 7486



In the figure, pin number 1, 2, 4, 5, 9, 10, 12, and 13 are the inputs to the Ex-OR gate, while 3, 6, 8, and 11 are the Ex-OR outputs. Pin number 7 is connected to ground and pin number 14 is connected to the power supply.

Exclusive-NOR Gate

The Exclusive-NOR gate function is a digital logic gate that is the complementary form of the Exclusive-OR function. Normally, this function is at logic level '1', but it goes 'LOW' to logic level '0' whenever any of its inputs are at logic level '1'. However, another instance where an output '1' is obtained is when both of its inputs are at logic level "1".

The Boolean expression is $Q = \overline{(A \quad B)} = A.B + \overline{A.B}$. It can also be represented as A B

Notes

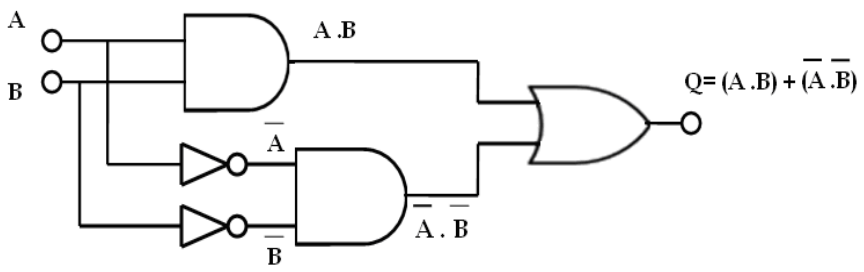
Figure 1.23 shows logic symbol and truth table of exclusive NOR gate.

Figure 1.23: Logic Symbol and Truth Table of Exclusive NOR Gate

Truth Table		
A	B	Q
0	0	1
1	0	0
0	1	0
1	1	1

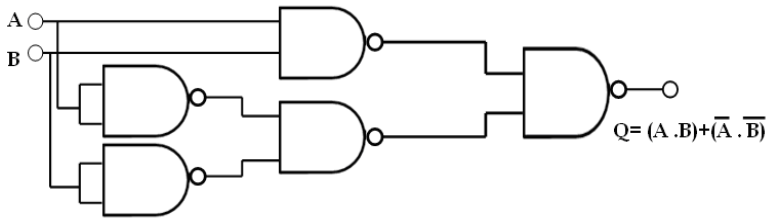
The following figure depicts the Ex-NOR equivalent circuit.

Figure 1.24: Ex-NOR Gate Equivalent Circuit



One of the main disadvantages of implementing the Ex-NOR function is that it contains three different types of logic gates which are AND, NOT, and OR gate within its basic design. One simpler way of producing the Ex-NOR function from a single gate type is to use NAND gates as shown in figure 1.25.

Figure 1.25: Ex-NOR Equivalent Circuit Using NAND Gate



Notes

Figure 1.26 shows the internal structure of IC 74266.

Figure 1.26: Internal Structure of IC 74266

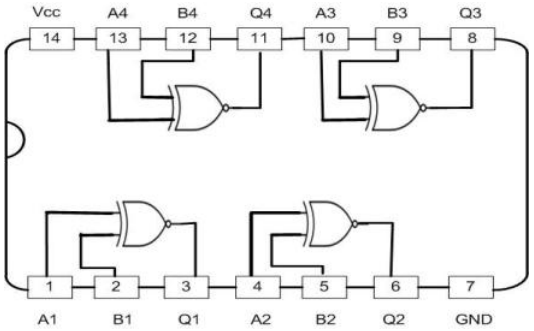


Figure 1.26 depicts how EX-NOR gates are placed within an IC. In the figure, pin number 1, 2, 4, 5, 9, 10, 12, and 13 are the inputs to the EX-NOR gate, while 3, 6, 8, and 11 are the EX-NOR outputs. Pin number 7 is connected to ground and pin number 14 is connected to the power supply.

We can classify integrated circuits as follows:

1. ***Small Scale Integration (SSI):*** It contains up to 10 transistors or a few gates within a single package such as AND, OR, and NOT gates.
2. ***Medium Scale Integration (MSI):*** It contains between 10 and 100 transistors or gates within a single package and performs digital operations such as adders, decoders, counters, flip-flops, and multiplexers.
3. ***Large Scale Integration (LSI):*** It contains between 100 and 1000 transistors or hundreds of gates and performs specific digital operations on I/O chips, memory, arithmetic, and logic units.
4. ***Very-Large Scale Integration (VLSI):*** It contains between 1,000 and 10,000 transistors or thousands of gates and performs computational operations such as processors, large memory arrays, and programmable logic devices.
5. ***Super-Large Scale Integration (SLSI):*** It contains between 10,000 and 100,000 transistors within a single package and performs computational operations like microprocessor chips, micro-controllers, and basic calculators.

1.3 Summary

- Binary codes are classified into many forms like weighted codes, reflective codes, sequential codes, alphanumeric codes, and so on.
- Alphanumeric codes include ASCII code and EBCDIC code.
- There are various logic gates in digital electronics like AND gate, OR gate, NOT gate, NAND gate, NOR gate, and so on, which have their own significance in digital electronics.
- NAND gate and NOR gate are called universal gates as other basic gates can be constructed using these gates.

Notes

1.4 Keywords

Boolean Expression: An expression that produces a Boolean value as a result.

Integrated Circuit: Complex circuits that are etched onto tiny chips of semiconductor.

Micro-controller: A computer-on-a-single integrated circuit containing a processor core, memory, and programmable input/output peripherals.

RTL: Resistor Transistor Switches.

1.5 Self Assessment

1. State whether the following statements are true or false:
   (a) Alphanumeric code is mainly used with large computer systems like mainframe computers.
   (b) The Boolean expression for NOR gate is denoted as Q = A+B.
   (c) The Ex-NOR function contains three different types of logic gates which are AND, NOT and OR gate within its basic design.
2. Fill in the blanks:
   (a) Digital data is represented, stored, and transmitted as groups of binary digits which are called _____.
   (b) In _____ code, each decimal digit is represented by a 4-bit binary number.
   (c) The Boolean expression for OR gate is denoted as _____.
3. Select a suitable choice for every question:
   (a) Gray codes are an example of which of the following codes:
      (i) Weighted codes
      (ii) Non-weighted codes
      (iii) Reflective codes
      (iv) Sequential codes
   (b) Which of the following ICs contain up to 10 transistors or a few gates within a single package such as AND, OR, and NOT gates.
      (i) SSI
      (ii) MSI
      (iii) LSI
      (iv) VLSI

1.6 Review Questions

1. "The digital logic gate is the building block from which all digital electronic circuits and microprocessor based systems are built." Discuss with the help of circuit diagrams.
2. "The Boolean expression for AND gate is A.B = Q." Elaborate.
3. "NAND gate is a combination of AND gate with an inverter or NOT gate connected together in series." Elaborate.

Answers: Self Assessment

1. (a) False (b) False (c) True
2. (a) Bits (b) BCD (c) Q = A+B
3. (a) Non-weighted code (b) SSI

Notes

1.7 Further Readings

Books

Radhakrishnan, T., & Rajaraman, V. (2007). Computer Organization and Architecture. New Delhi :Rajkamal Electric Press. Godse, A.P., & Godse D.A. (2008). Digital Electronics, 3rd ed. Pune: Technical Publications.

Online links

http://nptel.iitm.ac.in/courses/Webcourse-contents/IIT-KANPUR/esc102/node32.html

http://www.upscale.utoronto.ca/IYearLab/digital.pdf

http://www.electronicdesignworks.com/digital_electronics/multiplexer/multiplexer.htm

http://www.scribd.com/doc/26296603/DIGITAL-ELECTRONICS-demultiplexer

Notes

Unit 2: Devices Used in Digital Electronics

CONTENTS

Objectives
Introduction
2.1 Combinational Circuit and Sequential Circuit
2.2 Latches and Flip-flops
2.2.1 Latches
2.2.2 Flip-flops
2.3 Registers
2.3.1 Data Register
2.3.2 Shift Register
2.4 Counters
2.4.1 Synchronous Counters
2.4.2 Asynchronous Counters
2.5 Multiplexer
2.6 Demultiplexer
2.7 Decoder and Encoder
2.7.1 Binary Decoder
2.7.2 3 to 8 Decoder
2.7.3 Encoder
2.8 Summary
2.9 Keywords
2.10 Self Assessment
2.11 Review Questions
2.12 Further Readings

Objectives

After studying this unit, you will be able to:

- Differentiate between combinational and sequential circuits
- Explain latches and flip-flops
- Discuss the working of registers and counters
- Describe the functions of multiplexers and demultiplexers
- Identify the features of an encoder and a decoder

Introduction

Various devices are used in digital electronics. In the previous unit, we discussed logic gates which are devices used to implement basic logical expressions.

Electronic gadgets, despite their complexity operate using basic rules and circuits, which are discussed in this unit. These circuits and rules are applicable to computer organization as well.

Notes

We will also discuss about latches and flip-flops, registers, counters, multiplexer, de-multiplexer, and encoder and decoder.

2.1 Combinational Circuit and Sequential Circuit

There are two types of circuits that can be constructed using logic gates. They are:

1. Combinational circuits
2. Sequential circuits

Combinational Circuits: A circuit in which the output/outputs depend on the present state of combination of the logic inputs is called as combinational circuits. These circuits do not consider past state of inputs. Logic gates are the basic building blocks of these circuits. Multiplexers, demultiplexers are combinational circuits.

Sequential Circuits: A circuit in which the output/outputs depend on both the present state and past state of input is called as sequential circuit. Flip-flops are the basic building block of a sequential circuit. Sequential circuits have a memory unit which stores the past state of inputs. Sequential circuits can be synchronous or asynchronous.

2.2 Latches and Flip-flops

Just as gates are the building blocks of combinatorial circuits, flip-flops are the building blocks of sequential circuits. Gates are built directly from transistors and flip-flops are built from latches.

The output of latches and flip-flops depends not only on the current inputs but also on previous inputs and outputs. One of the contrasting features between a latch and a flip-flop is that a latch does not have a clock signal, whereas a flip-flop always has clock signals.

2.2.1 Latches

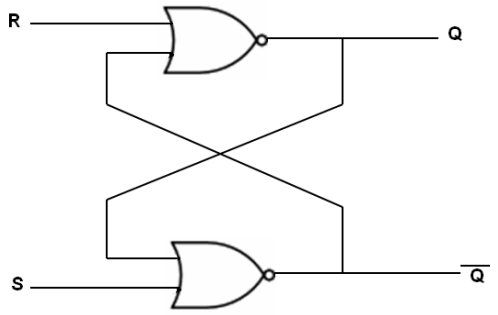
A latch is a device with exactly two stable states and these states are high-output and low-output. A latch has a feedback path, to retain the information. Hence, latches can be memory devices and can store one bit of data. As the name suggests, latches can be used to "latch onto" information and store it in the required place. One of the most commonly used latches is the SR latch.

SR Latch

An SR latch is an asynchronous device. An SR latch does not depend on control signals but depends only on the state of the S and R inputs. An SR latch can be constructed by interconnecting two NOR gates with a cross-feedback loop. SR latches can also be formed by interconnecting NAND gates but the inputs are swapped and negated.

A simple SR latch circuit is shown in the figure 2.1, where the inputs S and R represents 'set' and 'reset'. The current output of a latch is dependent on the state of the latch. Thus, the output at n^{th} instant represented as Q_n is dependent on output at $(n-1)^{th}$ instant represented by Q_n-1.

Figure 2.1: A Latch Circuit



Notes

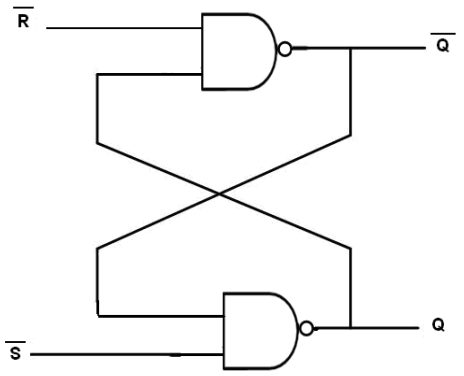
The table 2.1 shows the truth table for SR latch.

Table 2.1: Truth Table for SR Latch

S	R	Qn	$\overline{Qn}$
1	0	1	0
0	1	0	1
1	1	0	0
0	0	Qn-1	$\overline{Qn-1}$

Similar SR latch can be constructed using NAND gates. The figure 2.2 depicts how an SR latch can be constructed using NAND gate.

Figure 2.2: A Latch Circuit



The table 2.2 shows the truth table for an SR latch that is constructed using NAND gate.

Table2.2: Truth Table for SR Latch that Uses NAND Gate

$\overline{S}$	$\overline{R}$	Qn	$\overline{Qn}$
0	1	1	0
1	0	0	1
0	0	1	1
1	1	Qn-1	$\overline{Qn-1}$

Notes

This truth table shows the working of the SR latch.

Notes SR latch stores the last state of the inputs that is, it remembers which of the two inputs, 's' or 'r', last had the value of 1.

Task Using the concept of SR latch, find out the truth table for gated SR latch.

2.2.2 Flip-flops

We know that latches are asynchronous elements, which means that the output varies when the input changes. But most computers have synchronous elements. Synchronous elements are the elements in which the outputs of all the sequential circuits change simultaneously according to the clock signal.

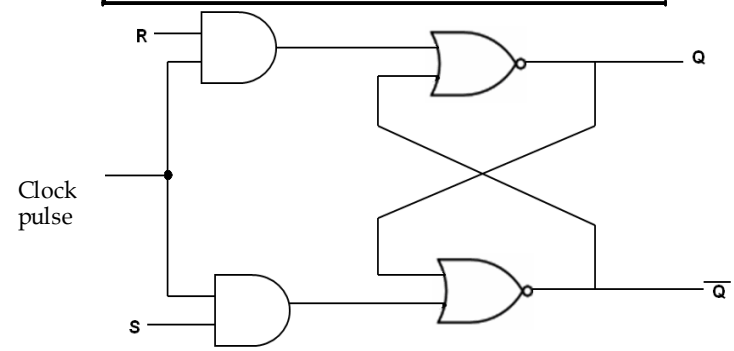
A flip-flop is therefore a synchronous version of the latch.

Set-Reset (SR) Flip-flop

The SR flip-flop has two inputs namely, a 'Set' input and a 'Reset' input. The two outputs of SR flip-flop are: the main output Q and its complement $\overline{Q}$.

The figure 2.3 depicts the circuit diagram of an SR flip-flop.

Figure 2.3: An SR Flip-Flop



Notes

The truth table of SR flip flop is shown in table 2.3.

Table 2.3: Truth Table of SR Flip Flop

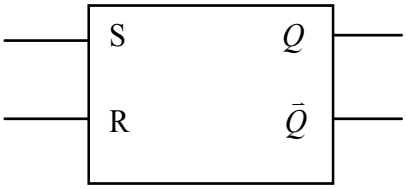
S	R	Q_{N+1}	$\overline{Q}_{N+1}$
0	0	Q_N	$\overline{Q}_N$
0	1	0	1
1	0	1	0
1	1	Indeterminate	

A pair of cross-coupled NOR gates is used to represent an SR flip-flop, wherein, the output of one gate is connected to one of the two inputs of the other gate and vice versa. The free input of one NOR gate is 'R' while the free input of the other gate is 'S'.

The input 'R' produces the output Q and the gate with the 'S' input produces the output $\vec{Q}$.

The logic symbol of SR flip-flop is shown in figure 2.4.

Figure 2.4: Logic Symbol of SR Flip-Flop



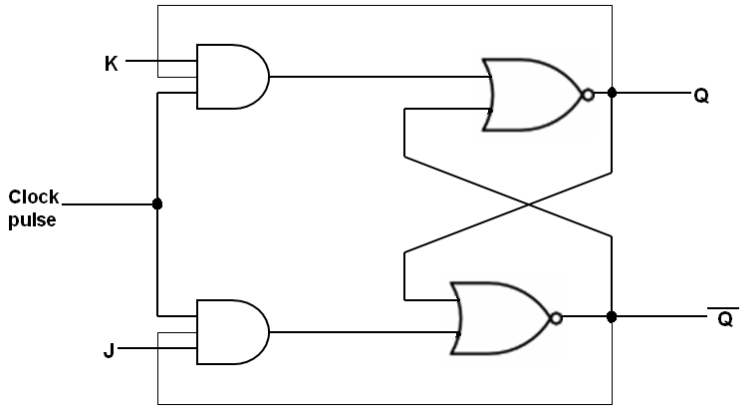
JK Flip-flop

In JK flip-flop, apart from the states of its inputs, the output is determined by its present output state as well. In the JK flip-flop, the 'S' input is called the 'J' input and the 'R' input is called the 'K' input. The output of the JK flip-flop does not change if both 'J' and 'K' are '0'. However, if both the inputs are '1', then the output toggles to its complement.

Notes

Figure 2.5 depicts the circuit diagram of a JK flip-flop.

Figure 2.5: An JK Flip-Flop



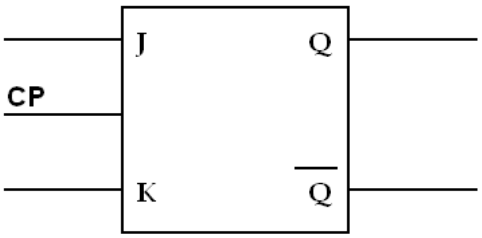
The truth table of JK flip-flop is shown in table 2.4.

Table 2.4: Truth Table of JK Flip-Flop

J	K	Q_{N-1}
0	0	Q_N
0	1	0
1	0	1
1	1	$\overline{Q}_N$

The logic symbol for JK flip-flop is shown in figure 2.6.

Figure 2.6: Logic Symbol of JK Flip-Flop



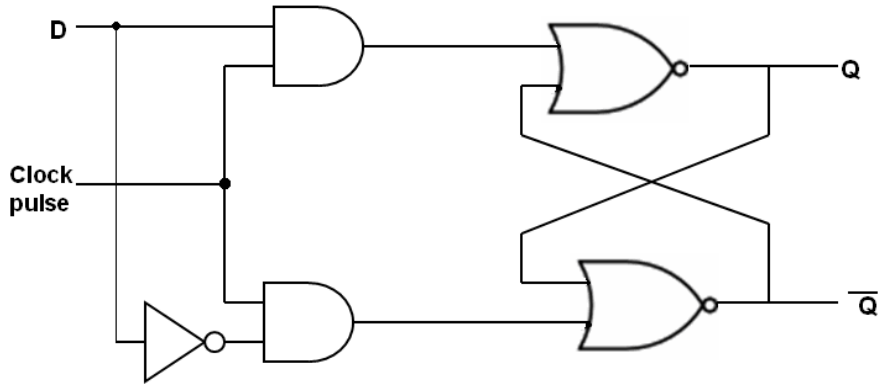
D Flip-flop

The D flip-flop is a clocked flip-flop with a single digital input 'D'. Every time a D flip-flop is clocked, its output follows the state of 'D'.

Notes

Figure 2.7 depicts the circuit diagram of a D flip-flop.

Figure 2.7: A D Flip-Flop



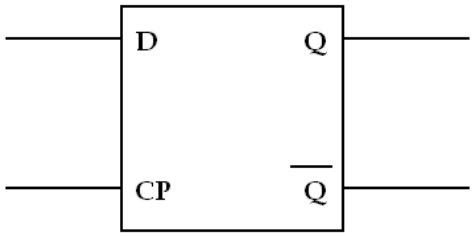
The truth table for D flip-flop is as shown in table 2.5.

Table 2.5: Truth Table of D Flip-Flop

CP	D	Qn+1
Positive	0	0
Positive	1	1
0	X	Qn

The logic symbol for D flip-flop is shown in figure 2.8.

Figure 2.8: Logic Symbol of D Flip-Flop



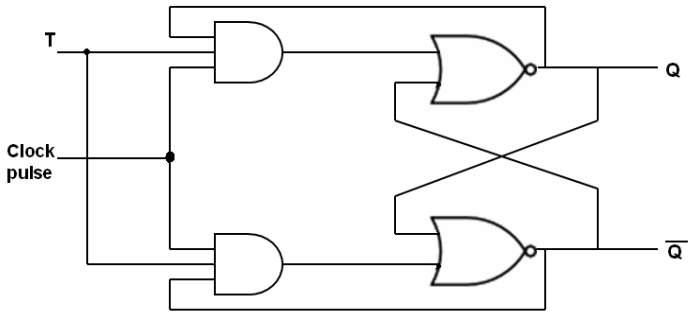
T Flip-flop

The T flip-flop is also known as toggle flip-flop. It is a modification of the JK flip-flop. The T flip-flop is obtained by connecting both inputs of a JK flip-flop that is, T flip-flop is obtained by connecting the inputs 'J' and 'K' together. When T = 0, both AND gates are disabled. Hence, there is no change in the output. When T= 1, the output toggles.

Notes

Figure 2.9 depicts the circuit diagram of a D flip-flop.

Figure 2.9: A T Flip-Flop



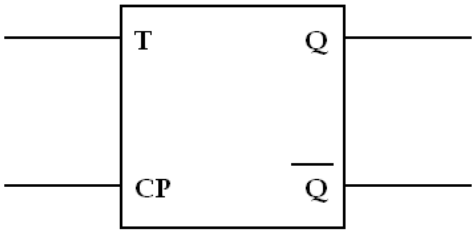
The truth table of T flip-flop is shown in table 2.6.

Table 2.6: Truth Table of T Flip-Flop

Qn	T	Qn+1
0	0	0
0	1	1
1	0	1
1	1	0

The logic symbol of T flip-flop is shown in figure 2.10.

Figure 2.10: Logic Symbol of T Flip-Flop



Notes

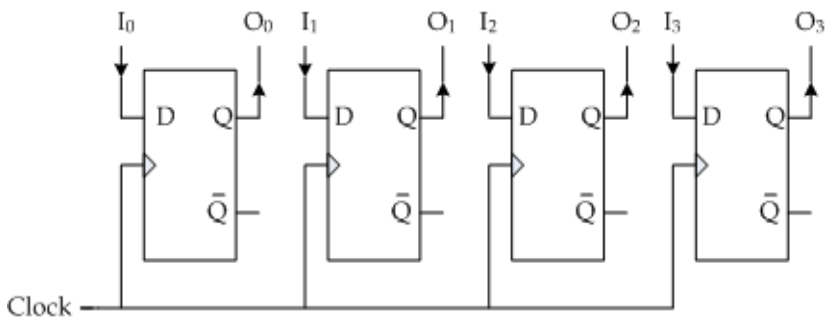
2.3 Registers

Flip-flops are the binary cells that are capable of storing one bit information. The group of flip-flops can be used to store a word, which is called register. Since flip flops can store 1-bit information, an n-bit register has a group of n flip-flops and is capable of storing binary information containing n-bits.

2.3.1 Data Register

Data register is the simplest type of register that is used for the temporary storage of a data word. In its simplest form, it consists of a set of n-D flip-flops, which share a common clock pulse. All the digits that are present in the n-bit data word are connected to the data register by an n-line data bus. The following figure depicts a four bit data register, implemented with four D flip-flops.

Figure 2.11: 4 Bit Data Register



Since all the flip-flops change state at the same time, the data register can be called as a synchronous device. The number of flip-flops in a register determines its total storage capacity. Therefore, when the first clock pulse arrives, the stored binary information becomes

QAQBQCQD = ABCD

2.3.2 Shift Register

Another common form of register used in many types of logic circuits is shift register. It is a set of flip-flops (usually D latches or S-R flip-flops) connected together in a series such that the output of one becomes the input of the next, and so on. It is called a shift register because the data is shifted through the register by one bit position on each clock pulse.

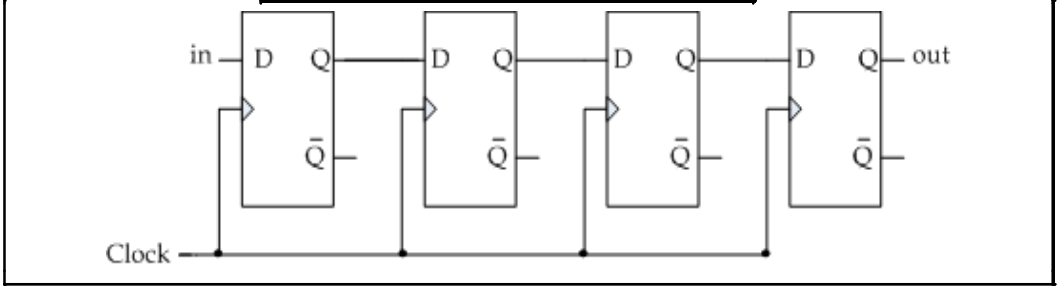
Let us next discuss the modes of operation of shift registers.

Notes

Serial-In Serial-Out Shift Register

The direction of shifting of data determines the 'serial-in serial-out' shift register into right shift and left shift register. The figure 2.12 shows a 'serial-in serial-out' right shift register.

Figure 2.12: Serial-in Serial-out Right Shift Register

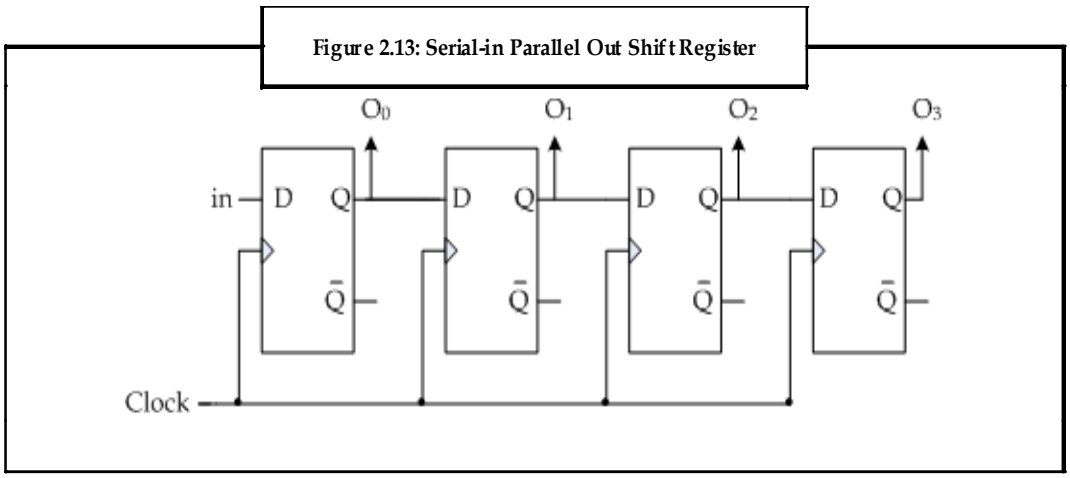


During the first clock pulse, the signal on the data input is latched in the first flip-flop. During the next clock pulse, the contents of the first flip-flop are stored in the second flip-flop, and the signal which is present at the data input is stored in the first flip-flop, and so on. Because the data is entered one bit at a time, we call it as a serial-in shift register. If there is only one output, and data leaves the shift register one bit at a time, then it is also a serial-out shift register.

Serial-In Parallel Out Shift Register

The 'preset' and 'clear' input commands can be provided to the flip-flops to obtain a parallel input. The parallel loading of the flip-flop can be synchronous (that is, it occurs with the clock pulse) or asynchronous (independent of the clock pulse), depending on the design of the shift register. The outputs of each flip-flop produce a parallel output.

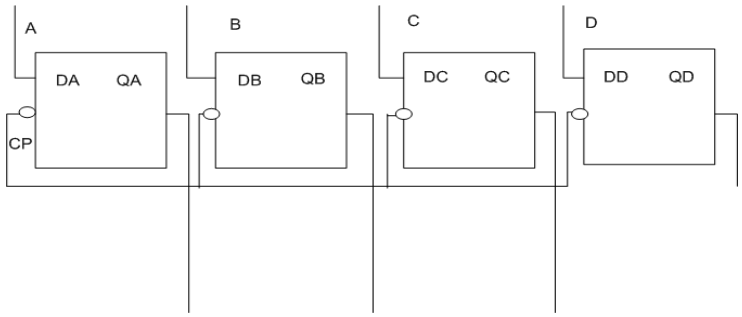
Figure 2.13: Serial-in Parallel Out Shift Register



Parallel In Parallel Out Shift Register

In 'parallel in parallel out' register there is simultaneous entry of all data bits and the bits appear on parallel outputs simultaneously. The following figure shows a 'parallel in parallel out' shift register.

Figure 2.14: Parallel In Parallel Out Shift Register



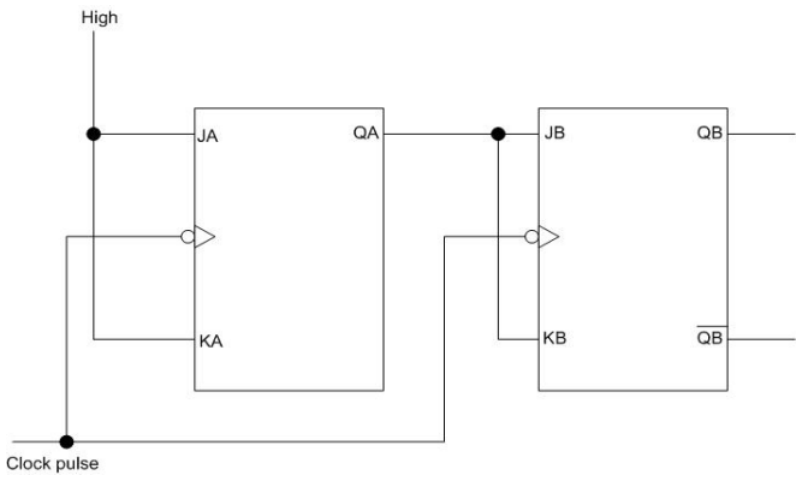
2.4 Counters

A counter is a register which is capable of counting the number of clock pulses arriving at its clock input. There are two types of counters namely, synchronous and asynchronous counters. As the common clock in a synchronous counter is connected to all of the flip-flop, they are clocked simultaneously. In asynchronous counter, the first flip-flop is clocked by the external clock pulse and then each successive flip-flop is clocked by the Q or $\overline{Q}$ output of the previous flip-flop.

2.4.1 Synchronous Counters

Figure 2.15 depicts a synchronous counter.

Figure 2.15: Synchronous Counters



Here, the clock signal is connected in parallel to clock inputs of all the flip -flops. Initially, we assume that QA = QB = 0. When positive edge of the first clock pulse is applied, flip-flop A will toggle, whereas flip-flop B output will remain zero because JB = KB = 0. After first clock pulse,

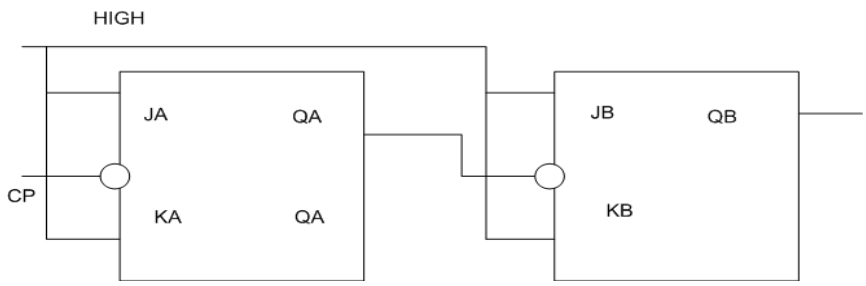
Notes

QA = 1 and QB = 0. After second clock pulse, QA = 0 and QB = 1. After the fourth clock pulse the counter recycles back to its original state.

2.4.2 Asynchronous Counters

An asynchronous counter consists of a series connection of complementing flip-flops, with the output of each flip-flop connected to the clock input of the next higher order flip-flop. To obtain a complementing flip-flop a JK flip flop can be used by connecting the J and K inputs together. The figure 2.16 shows a 2-bit asynchronous counter using JK flip-flops.

Figure 2.16: Asynchronous Counter

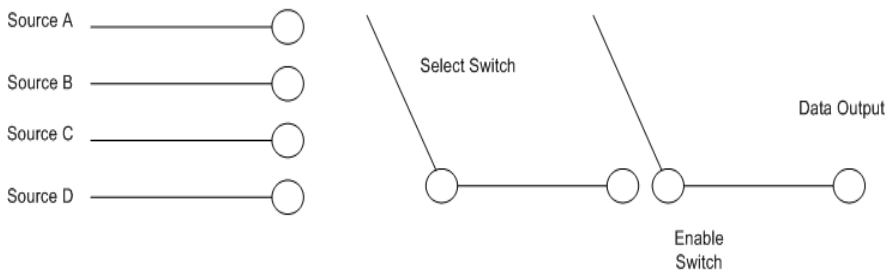


2.5 Multiplexer

A multiplexer is a digital switch which allows digital information from several sources to be routed onto a single output line. A set of selection lines control the selection of a particular input line. Therefore, a multiplexer is a multiple-input and single-output switch. It provides the digital equivalent of an analog selector switch.

A multiplexer is also called as data selector as it accepts many digital data inputs and selects one of them at any given time to pass onto the output. In some cases, two or more multiplexers are enclosed within an IC package. Figure 2.17 depicts the switching concept of a multiplexer.

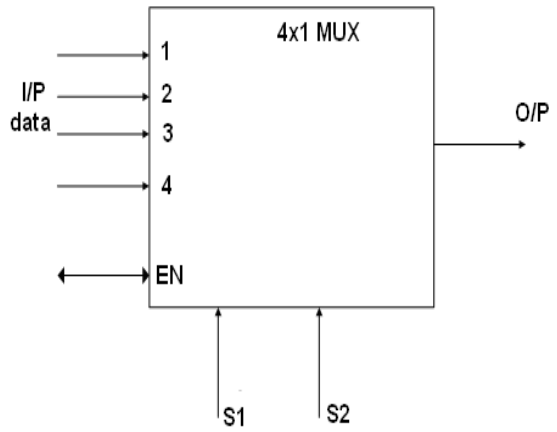
Figure 2.17: Switching Concept of a Multiplexer



Notes

Figure 2.18 depicts a 4 to 1 line multiplexer.

Figure 2.18: 4x1 Multiplexer



The truth table of a multiplexer circuit is shown in table 2.7.

Table 2.7: Truth Table of Multiplexer

S1	S0	Y
0	0	D0
0	1	D1
1	0	D2
1	1	D3

Example: 74xx151 is an 8 to 1 multiplexer which has 8 inputs and two outputs. One of the outputs is an active high output and the other is an active low output. These circuits are used mostly in digital systems of all types, such as data selection, data routing, operation sequencing, parallel-to-serial conversion, waveform generation, and logic-function generation.

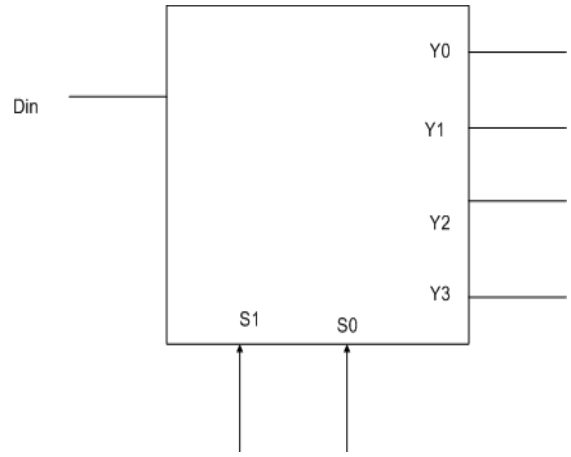
Notes

2.6 Demultiplexer

A circuit that receives information on a single line and transmits the information on any of the 2^n possible output lines is called as a demultiplexer. Therefore, a demultiplexer is called a single-input multiple-output switch. The values of n selection lines control the selection of specific output line.

A demultiplexer is shown in figure 2.19.

Figure 2.19: 1x4 Demultiplexer



The truth table of a demultiplexer circuit is shown in table 2.8.

Table 2.8: Truth Table of Demultiplexer

Enable	S1	S0	Din	Y0	Y1	Y2	Y3
0	X	X	X	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	0	0	0	
1	0	1	1	0	1	0	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	0
1	1	1	0	0	0	0	0
1	1	1	1	0	0	0	1

A demultiplexer is used extensively in clock demultiplexer, security monitoring system, synchronous data transmission system, and so on.

2.7 Decoder and Encoder

A decoder is a multiple input, multiple output logic circuit. A decoder converts coded inputs into coded outputs, where the input and output codes are different. Often, the input code has fewer bits than the output code. Each input code word produces a different output code word. The following figure shows the general structure of the decoder circuit which shows that n inputs produce 2^n possible outputs. The 2^n output values are from 0 through 2^{n-1}. Usually, a decoder is provided with enabled inputs to activate decoded output based on data inputs. When any one enabled input is unasserted, all outputs of decoder are disabled.

Notes

2.7.1 Binary Decoder

A binary decoder is a decoder which has an n-bit binary input code and one activated output which is selected from 2^n output codes. This decoder is applicable in instances where it is necessary to activate exactly one of 2^n outputs based on an n-bit input value.

In a 2 to 4 decoder, 2 inputs are decoded into four outputs, each output representing one of the minterms of the 2 input variables. The two inverters provide the complement of the inputs and each one of the four AND gates generate one of the minterms.

The figure 2.20 illustrates the circuit diagram of a 2 to 4 decoder.

Figure 2.20: 2 to 4 Decoder Circuit Diagram

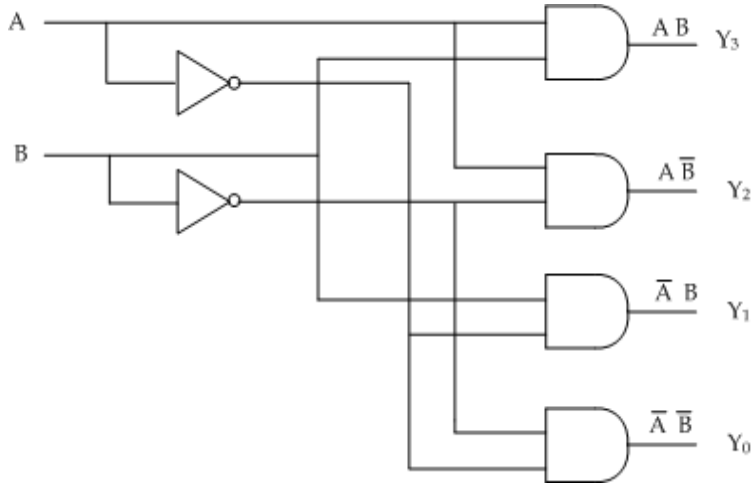


Table 2.9 shows the truth table for a 2 to 4 decoder.

Table 2.9: Truth Table of a 2 to 4 Decoder

Inputs			Outputs			
EN	A	B	Y3	Y2	Y1	Y0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

2.7.2 3 to 8 Decoder

The 74x138 is a 3 to 8 decoder. It accepts three binary inputs, namely A, B, and C and it provides eight individual active low outputs (Y0-Y7) when enabled. The device has three enable inputs, that is, two active low and one active high.

Notes

The figure 2.21 depicts a 3 to 8 decoder.

Figure 2.21: A 3 to 8 Decoder

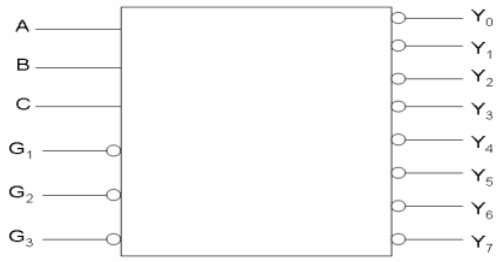


Figure 2.22 depicts the circuit diagram of a 3 to 8 decoder.

Figure 2.22: A 3 to 8 Decoder Circuit Diagram

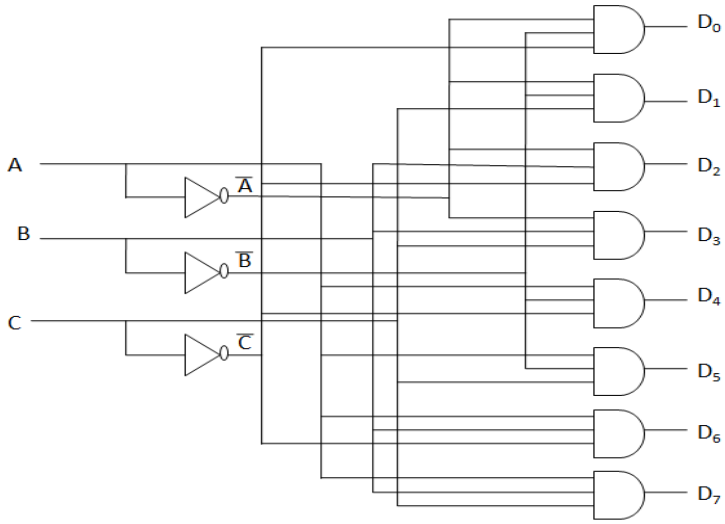


Table 2.10 shows the truth table of a 3 to 8 decoder.

Table 2.10: Truth Table of a 3 to 8 Decoder

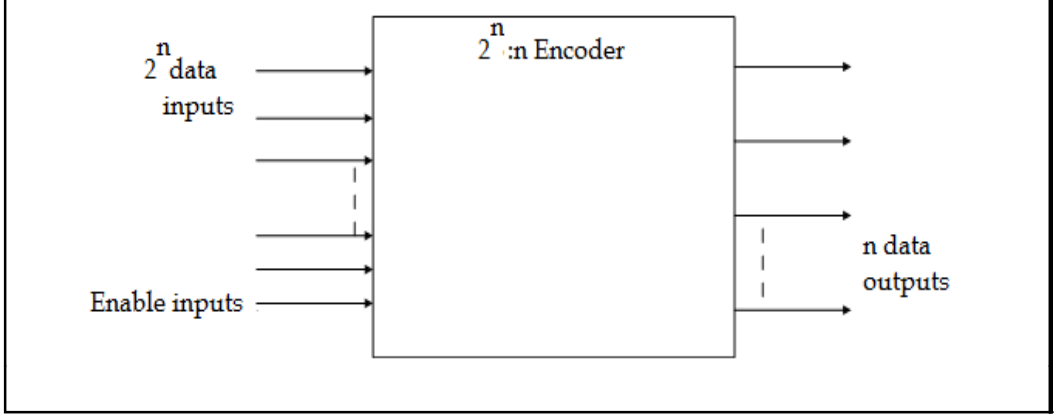
A	B	C	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Notes

2.7.3 Encoder

A digital circuit that performs the inverse operation of a decoder is called as an encoder. It has 2^n input lines and n output lines. In an encoder, the output lines generate the binary code corresponding to the input value. The figure 2.23 depicts the general structure of an encoder circuit.

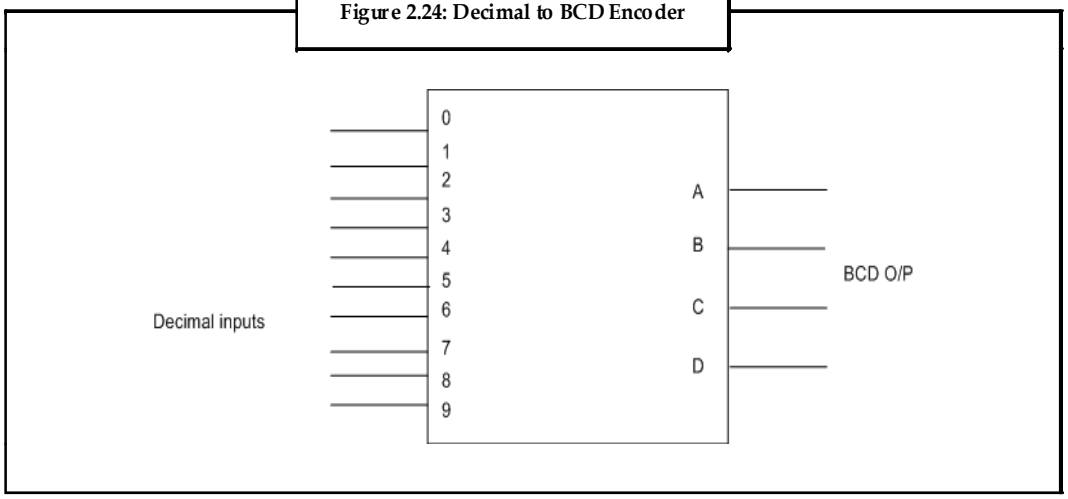
Figure 2.23: An Encoder Circuit



Decimal to BCD Encoder

The decimal to BCD encoder has ten input lines and four output lines. The input for the encoder is the decoded decimal data and encoded BCD is the output available on the four output lines. The figure 2.24 shows the logic symbol for decimal to BCD encoder IC.

Figure 2.24: Decimal to BCD Encoder



Notes

Figure 2.25 depicts the decimal to BCD encoder circuit diagram.

Figure 2.25: Decimal to BCD Encoder Circuit Diagram

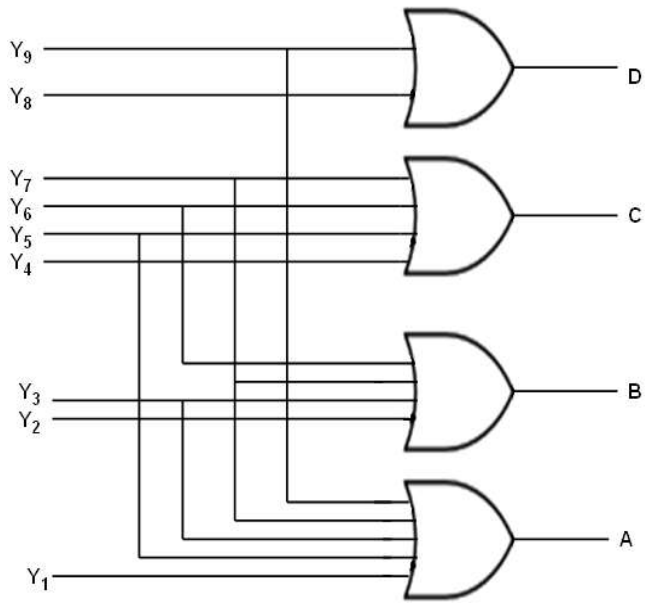


Table 2.11 shows the truth table for a BCD encoder.

Table 2.11: Truth Table of a BCD Encoder

Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	Y_8	Y_9	D	C	B	A	BCD
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	1
0	1	0	0	0	0	0	0	0	0	0	1	0	2
0	0	1	0	0	0	0	0	0	0	0	1	1	3
0	0	0	1	0	0	0	0	0	0	1	0	0	4
0	0	0	0	1	0	0	0	0	0	1	0	1	5
0	0	0	0	0	1	0	0	0	0	1	1	0	6
0	0	0	0	0	0	1	0	0	0	1	1	1	7
0	0	0	0	0	0	0	1	0	1	0	0	0	8
0	0	0	0	0	0	0	0	1	1	0	0	1	9

2.8 Summary

- Binary codes are classified into many forms like weighted codes, reflective codes, sequential codes, alphanumeric codes, and so on.
- There are various logic gates in digital electronics like AND gate, OR gate, NOT gate, NAND gate, NOR gate, and so on which have their own significance.
- The output of latches and flip-flops depends not only on the current inputs but also on previous inputs and outputs.
- The group of flip-flops can be used to store a word which is called a register.

- A counter is a register which is capable of counting the number of clock pulses arriving at its clock input.
- A multiplexer is a digital switch which allows digital information from several sources to be routed onto a single output line.
- A demultiplexer is a circuit that receives information on a single line and transmits the same information on any of the 2^n possible output lines.
- A multiple input, multiple output logic circuit is called as a decoder. It converts coded inputs into coded outputs.
- An encoder has 2^n input lines and n output lines.

2.9 Keywords

Clock Pulse: A circuit in a processor that generates a regular sequence of electronic pulses used to synchronize operations of the electronic system.

Register: An electronic component that offers a known resistance to the flow of electricity.

Switch: A device that directs incoming data from any of multiple input ports to the specific output port.

2.10 Self Assessment

1. State whether the following statements are true or false:
   (a) A decoder has 2^n input lines and n output lines.
   (b) In a 2 to 4 decoder, 2 inputs are decoded into four outputs, each output representing one of the minterms of the 2 input variables.
   (c) A demultiplexer is used extensively in clock multiplexer, security monitoring system, synchronous data transmission system, and so on.
2. Fill in the blanks:
   (a) The decimal to BCD encoder has ten input lines and _____ output lines.
   (b) A multiplexer is also called as _____ as it accepts many digital data inputs and selects one of them at any given time to pass onto the output.
   (c) A counter is a _____ which is capable of counting the number of clock pulses arriving at its clock input.
3. Select a suitable choice for every question:
   (a) Apart from the states of its inputs, the output of _____ is determined by its present output state as well.
      (i) JK flip-flop
      (ii) S-R flip-flop
      (iii) D flip-flop
      (iv) T flip-flop
   (b) _____ is a multiple input, multiple output logic circuit.
      (i) Decoder
      (ii) Encoder
      (iii) Multiplexer
      (iv) Demultiplexer

Notes

2.11 Review Questions

1. "Flip-flops are built from latches." Discuss.
2. "A pair of cross-coupled NOR gates is used to represent an S-R flip-flop." Explain with the help of circuit diagrams.
3. "Data register is the simplest type of register which is used for the temporary storage of a data word." Elaborate.
4. "Shift registers can be operated in different modes." Discuss.
5. "Multiplexer is a multiple-input and single-output switch." Explain with the help of circuit diagrams.
6. "Demultiplexer is called a single-input, multiple-output switch." Explain with the help of circuit diagrams.
7. "In an encoder, the output lines generate the binary code corresponding to the input value." Elaborate.

Answers: Self Assessment

1.	(a) False	(b) True	(c) False
2.	(a) Four	(b) Data selector	(c) Register
3.	(a) JK flip-flop	(b) Decoder	

2.12 Further Readings

Books

Radhakrishnan, T., & Rajaraman, V. (2007). Computer Organization and Architecture. New Delhi :Rajkamal Electric Press.

Godse, A.P., & Godse D.A. (2008). Digital Electronics, 3rd ed. Pune: Technical Publications.

Online links

http://nptel.iitm.ac.in/courses/Webcourse-contents/IIT-KANPUR/esc102/node32.html

http://www.upscale.utoronto.ca/IYearLab/digital.pdf

http://www.electronicdesignworks.com/digital_electronics/multiplexer/multiplexer.htm

http://www.scribd.com/doc/26296603/DIGITAL-ELECTRONICS-demultiplexer

Notes

Unit 3: Data Representation and Data Transfer

CONTENTS

Objectives
Introduction
3.1 1's and 2's Complement
3.2 Fixed-Point and Floating-Point Number
3.2.1 Decimal Fixed-Point Representation
3.2.2 Floating-Point Representation
3.3 Register Transfer
3.3.1 Bus Transfer
3.3.2 Memory Transfer
3.4 Microoperation
3.4.1 Logic Microoperation
3.4.2 Shift Microoperation
3.4.3 Arithmetic Logic Shift Unit
3.5 Summary
3.6 Keywords
3.7 Self Assessment
3.8 Review Questions
3.9 Further Readings

Objectives

After studying this unit, you will be able to:

- Analyse 1s and 2s complement
- Discuss representation of fixed-point and floating-point numbers
- Discuss register transfer language and microoperations

Introduction

We are aware that computer organization deals with the functional units and the interconnectivity among them by specifying the details of its architecture. These details may include instruction set, number of bits used for the representation of different types of data (numbers, characters, etc.), and so on. Any computer's organization relies on the way it represents numbers, character, and other information. This information in an organized form is referred to as data.

The information received are stored in memory or used by the CPU to perform required operations. The information received is either data or instructions. An instruction is a command to perform a specific type of operation and the data are the numbers or encoded characters that are represented by signed integers 0 and 1. These signed integers are called as binary numbers. The basic arithmetic operations such as addition, subtraction, multiplication and division can be performed using the basic data types.

Notes Instruction set is the basic set of commands that a microprocessor understands.

Notes

3.1 1's and 2's Complement

In computers, binary number system is used for storing information as 0's and 1's.

Did u know? The basic unit of computer storage is a bit, which can be either 0 (off/positive) or 1 (on/negative).

1's Complement

1's complement is a method of representation of negative numbers in computers. A binary number's 1's complement is obtained by inverting all 0s to 1s and all 1s to 0s.

Example: 0 in 8-bits is represented as 00000000 and -0 is represented in 1's complement as 11111111.

Task Find out 1's complement of (1001)2

2's Complement

2's complement is a method of representation of negative binary numbers in computers. A binary number's 2's complement is obtained by adding 1 to that number's 1's complement.

Example: We can represent -28 in 2's complement as follows:

1. Write 28 in binary form.
   00011100
2. Invert the digits by converting 0's to 1's and 1's to 0's.
   11100011
3. Add 1 to 11100011

The result is 11100100, which represents -28.

Task Find out 2's complement of (1010 0011)2

Computer circuits perform different operation on binary number system. Binary operations are the key to perform all basic arithmetic operations, such as addition, subtraction, multiplication, and division. In these operations, the Most Significant Bit (MSB) is reserved to indicate the sign (+ or -). MSB is **0** for positive numbers and **1** for negative numbers. The rules for the binary additions are depicted in the table 3.1.

Notes

As per the table 3.1, when two positive numbers are to be added, bit pairs are added, starting from lower-order (right end) bits, going up to the higher-order bits. While adding 1+1, a carry is generated.

Table 3.1: Binary Operation (Addition)

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Just like binary additions, binary subtractions also have some rules to be followed. These rules are depicted in table 3.2.

As per table 3.2, when two positive numbers are to be subtracted, bit pairs are subtracted, starting from lower-order (right end) bits, going up to the higher-order bits. The subtraction of binary numbers is similar to the subtraction of decimal numbers. Just as decimal subtraction has the concept of "borrow", subtraction of binary numbers also has the concept of "carry". If you have to subtract a one from a zero, you need to "carry" from the left, just as in decimal subtraction.

Table 3.2: Binary Operation (Subtraction)

A	B	Difference	Carry
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Did u know? Signed number representations are required to encode negative numbers in binary number systems.

1's Complement Addition

In 1's complement addition, when two numbers are added, the two binary numbers may be added including the sign bit. If there is a carry after the MSB position, it is called carry-out. In case of signed number addition, it is called end-around-carry. In 1's complement, the representation of positive numbers is identical to sign magnitude system. To represent a negative number, the convention is different and is done by bit- complementing method, meaning replacing 0 by 1 and 1 by 0.

Example: 5 is represented as 10 in bits and -5 is represented as 01

Notes

Example: Add $(1110)_2$ and $(1010)_2$.

1	Carry	
1 1 1 0		Decimal 1 8
+ 1 0 1 0		+ Decimal 1 0
1 1 0 0 0		Decimal 2 4

Task Add 6+8 using 1's complement.

1's Complement Subtraction

1's complement has two cases of subtraction. These cases are:

1. Subtraction of a smaller number from a larger number. The steps followed for subtraction are:
   (a) Determine the 1's complement of the smaller number.
   (b) Add 1's complement to the larger number.
   (c) Add the carry that is obtained, to the result.

Example: Subtract $(101011)_2$ from $(111001)_2$.

	1 1 1 0 0 1	
	+ 0 1 0 1 0 0	1's complement of 101011
1	0 0 1 1 0 1	
	1	Add end-round carry
	0 0 1 1 1 0	Answer

2. Subtraction of a larger number from a smaller number. The steps followed for this type of subtraction are:
   (a) Determine the 1's complement of the larger number.
   (b) Add 1's complement to the smaller number.
   (c) As the result is in 1's complement, to obtain the required result, the answer is converted to 1's complement with a negative sign.

Example: Subtract $(111001)_2$ with $(101011)_2$.

	1 0 1 0 1 1	
+	0 0 0 1 1 0	1's complement of 111001
	1 1 0 0 0 1	→ Answer in 1's complement
-	0 0 1 1 1 0	Answer in negative

Notes

Notes 1's complement arithmetic was common in older computers; the PDP-1, CDC 160A and UNIVAC 1100/2200 series are some computers that used ones'-complement arithmetic.

2's Complement Addition

When 2's complement system is used to represent negative numbers, the addition operation is similar to the 1's complement system. The calculation steps that are followed to add using 2's complement include:

1. Representing both numbers in signed-2's complement format.
2. Adding operands and discard carry-out of the sign bit MSB (if any).

Example:

Add 69 and 12. 1 1 Carry Row

```
  0000 0000 0000 0000 0000 0000 0100 0101   (69)
+ 0000 0000 0000 0000 0000 0000 0000 1100   (12)
-------------------------------------------------
  0000 0000 0000 0000 0000 0000 0101 0001   (81)
```

The result obtained is automatically in signed-2's complement form.

2's Complement Subtraction

The following steps explain subtraction using 2's complement.

If A and B are the numbers that are to be subtracted, A is called as minuend and B is called the subtrahend.

1. Represent both numbers in signed-2's complement format.
2. Obtain 2's complement of the subtrahend B (which may be in complement form already if it is negative).
3. Add it to A.

Example: Subtract 12 from 7.

+7 –(-12)

```
  0000 0111   (+7)
+ 1111 0100   (-12)
-------------
  1111 1011   (-5)
```

The result is automatically in signed-2's complement form.

Notes ***The 2's complement system has the following advantage:*** It is not required for the addition or subtraction circuitry of the 2's complement to examine the signs of the operands to determine whether to perform addition or subtraction.

2's complement has two cases of subtraction. These cases are:

1. Subtraction of a smaller number from a larger number. The subtraction steps are:
   (a) The smaller number of 2's complement is determined.
   (b) The bigger number is added with 2's complement.
   (c) The carry is discarded.

Notes

Example: Subtract $(101011)_2$ from $(111001)_2$.

```
  1 1 1 0 0 1
+ 0 1 0 1 0 1          2's complement of 101010
-------------
  0 0 1 1 1 0    →     Answer
```

2. Subtraction of a larger number from a smaller number. The steps for this type of subtraction are:
   (a) The 2's complement of bigger number is determined.
   (b) The 2's complement is added to the smaller number.
   (c) The 2's complement answer is prefixed with negative value to get a true value.

Example: Subtract 69 from 12

Here, 12-69 = 12+(-69)

69 in 2's complement is 0000 0000 0000 0000 0000 0000 0100 0101

-69 would be 1111 1111 1111 1111 1111 1111 1011 1011 (converting 0's to 1's and 1's to 0's, and adding 1)

```
                                          111      Carry Row
  0000  0000  0000  0000  0000  0000  0000  1100     (12)
+ 1111  1111  1111  1111  1111  1111  1011  1011     (-69)
-----------------------------------------------------
  1111  1111  1111  1111  1111  1111  1100  0111     (-57)
```

Task Subtract 9-2 using 2's complement.

3.2 Fixed-Point and Floating-Point Number

Fixed-points are considered as a simple and easy way to represent factorial numbers. The fixed-point numbers use a fixed amount of bits. The term "fixed-point" refers to the decimal point of a number. There is no separate symbol to designate where the decimal point lies. However, in binary, bits can either be 0 or 1.

In floating-point representation, a decimal point number is multiplied by a base value and it is scaled up with an exponent value. The usage of floating-point number is limited when compared to fixed-point number because floating-point number carries a wide range of real numbers values.

3.2.1 Decimal Fixed-Point Representation

Fixed-point representation has a radix point called decimal point. Fixed-point numbers having decimal points at the right end of the number are considered as integers, whereas the fixed-point numbers having decimal points at the left end of the number are considered as fractions. In either case, the decimal point position is fixed because the number stored in the memory is treated as an integer or as a fraction.

The binary numbers that are unsigned are always treated as positive integers and are represented as **0**s in the MSB. As we are aware, the binary numbers that are signed differ for negative numbers and are represented as **1**s in the MSB.

Notes

The magnitude of the signed binary numbers can be represented using three approaches. They are:

1. Sign and magnitude representation.
2. Signed 1's complement representation.
3. Signed 2's complement representation.

The following section deals with a brief explanation of these three approaches.

Sign and Magnitude Representation

In this approach, the leftmost bit in the number is used for indicating the sign; 0 indicates a positive integer, and 1 indicates a negative integer. The remaining bits in the number give the magnitude of the number.

Example: -2410 is represented as:

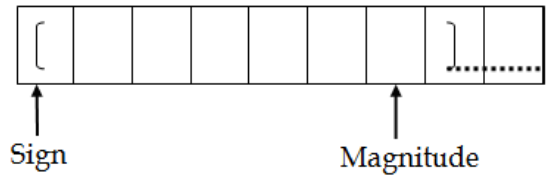
1001 1000

In this example, the leftmost bit 1 means negative, and the magnitude is 24.

The magnitude for both positive and negative value is same, but they differ only with their signs.

The range of values for the sign and magnitude representation is from -127 to 127.

Figure 3.1: Representation of Sign and Magnitude in Memory



Example: 1-sign bit and 7-bit magnitude.

01001001 = +73 0 at the LHS indicates the positive value

10010010 = - 18 → 1 at the LHS indicates the negative value

Signed 1's Complement Representation

In signed 1's complement representation, a negative value is obtained by taking the 1's complement of the corresponding positive number. Also, a signed 1's complement method produces end carry during arithmetic operation that cannot be discarded.

The range of values for the signed 1's complement representation is from -127 to 128.

Example: Consider 8-bit numbers for 1's complement.

$(29)_{10}$ = $(00011101)_2$ = 000011101 1's complement for positive value

-(29)10 = -(00011101)2 = 111100010 → 1's complement for negative value

Notes

Signed 2's Complement Representation

In signed 2's complement representation, the 2's complement of a number is found by first taking the 1's complement of that number, then incrementing the result by 1.

The range of values for the signed 2's complement representation is from -128 to 127.

Example:

Consider 8-bit numbers for 2's complement.

$(29)_{10} = (00011100)_2 = (000011100)_{2s}$ 2's complement for positive value

$-(29)_{10} = -(00011100)_2 = (11110010)_{2s}$ → 2's complement for negative value

3.2.2 Floating-Point Representation

The floating-point representation is used to perform operations for high range values. The scientific calculations are carried out using floating-point values. To make calculations simple, scientific numbers are represented as follows:

The number 5,600,000 can be represented as $0.56 * 10^7$

Here, 0.56 is the mantissa and 7 is the value of the exponent.

Similar to the above example, binary numbers can also be represented in the exponential form. The representation of binary numbers in the exponential form is known as floating-point representation. The floating-point representation divides the number in two parts, the left hand side is a signed, fixed-point number called **mantissa** and the right hand side of the number is called the **exponent**. The floating-point values are also assigned with a sign; **0** indicating the positive value and **1** indicating the negative value.

General form of floating-point representation of a binary number:

$$x = (x_0 * 2^0 + x_1 * 2^1 + x_2 * 2^2 + \text{------} + b_{-(n-1)} * 2^{-(n-1)})$$

$$\text{mantissa} * 2^{\text{exponent}}$$

In the above syntax, the decimal point is moved left for negative exponents of two and right for positive exponents of two. Both the mantissa and the exponent are signed values allowing for negative numbers and for negative exponents respectively.

Example: Convert 111101.1000110 into floating-point value.

$111101.1000110 = 1.111011000110 * 2^5$ converted to floating-point value.

→ indicates the negative sign value

In this example, the integer value is converted to floating-point value by shifting the radix point next to the signed integer and scaling up the number to the exponential form by multiplying the value with the base 2. The value remains unchanged and this procedure is called the normalized method.

The floating-point numbers are of two types, which are:

1. Normalized and Un-normalized
2. Single precision and Double precision

Notes

Normalized and Un-Normalized Floating-Point Numbers

In normalized floating-point representation, the most significant digit of the mantissa is non-zero. Thus, the number is normalized only if its leftmost digit is non-zero. Normalized floating-point numbers provide the maximum number of possible precisions for the floating-point number.

Example: The number 450 is normalized, but the number 000045 is not normalized.

$0.0035*10^5$ is un- normalized, whereas $0.35000 * 10^3$ is normalized.

Eight bit numbers are not normalized because of leading **0**s. These numbers can be normalized by shifting three places to the left to obtain 10010000 in a number.

Normalized Version representation is shown below:

Value represented = $+1.0110\ldots *2^6$ is a normalized value

Un-normalized version representation is shown below:

Value represented = $+0.0010110\ldots * 2^9$ is an unsigned value.

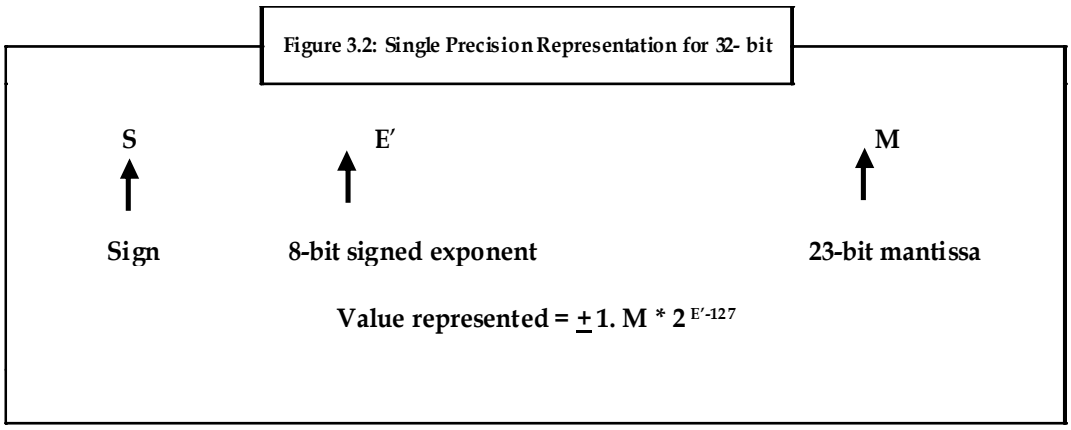
There is no implicit point to the left of the binary point.

Single Precision and Double Precision

As depicted in the figure 3.2, the 23 bit, which represents the **mantissa** whose most significant bit is always equal to 1, is normalized. This bit is immediate to the left of the binary point. Hence, the 23 bits stored in the **M** field represent the fractional part of the **mantissa** and this 32 bit representation is called the single precision because it occupies a single 32 bit word.

Double precision floating point numbers are used to improve the accuracy and range of floating - point numbers. The excess -1023 exponent **E′** has the range **1<=E′<=2064** for normal values. Thus, the 53-bit **mantissa** provides a precision equivalent to 16 decimal digits.

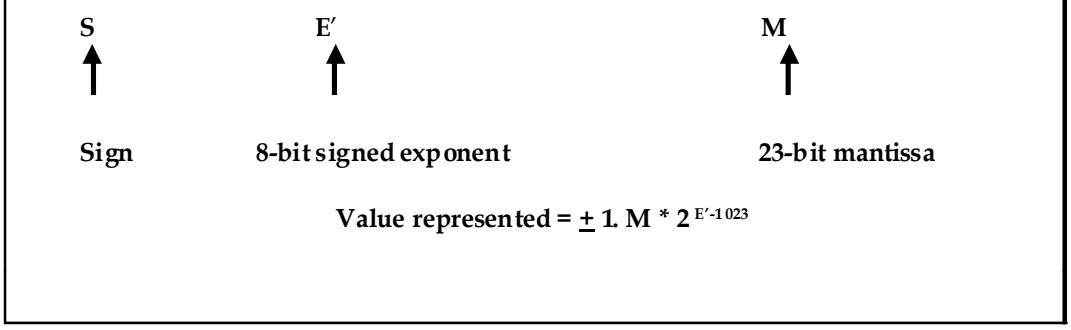
Figure 3.2: Single Precision Representation for 32- bit



Notes

Figure 3.3 depicts double precision representation for 64-bit.

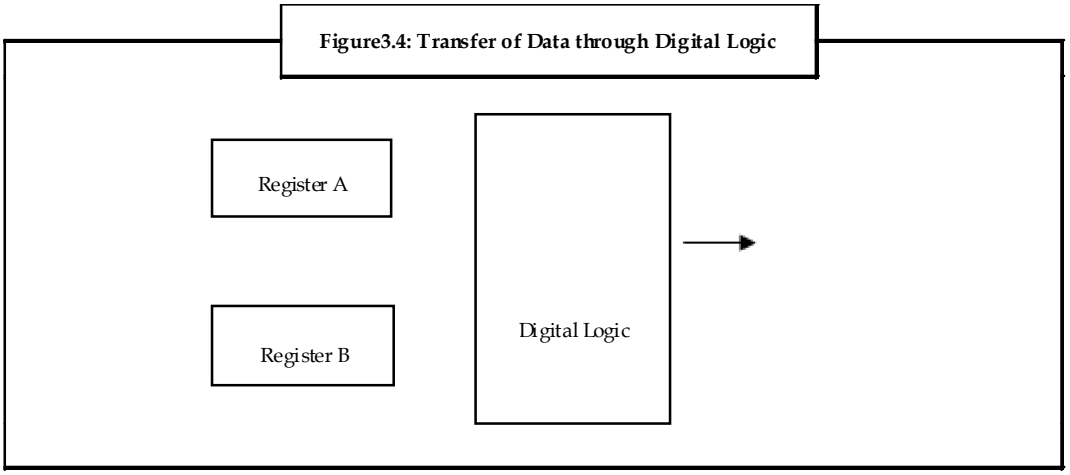
Figure 3.3: Double Precision Representation for 64 -bit



3.3 Register Transfer

Registers refer to the storage space that holds the data and instructions. To transmit data and instructions from one register to another register, memory to register and memory to memory, the register transfer method is used. This register helps in the transfer of data and instructions between memory and processors to perform the specified tasks. The transfer of data is more concise, precise, and is provided in an organized manner. Digital logic is used to process the data. Figure 3.4 depicts the transfer of data though digital logic.

Figure3.4: Transfer of Data through Digital Logic



There are two types of register transfers. They are:

1. Bus transfer
2. Memory transfer

3.3.1 Bus Transfer

The most efficient way to transfer data is by using a common bus system, which is configured using common bus registers in a multiple register. The structure of the bus consists of a set of lines. These lines are registers of one bit each that transfer only one data at a time. The data transfer is controlled by the control signals. Control signals determine which register is to be selected during each register transfer. To construct a common bus system, two methods are used:

1. Using multiplexer
2. Using three states bus buffers

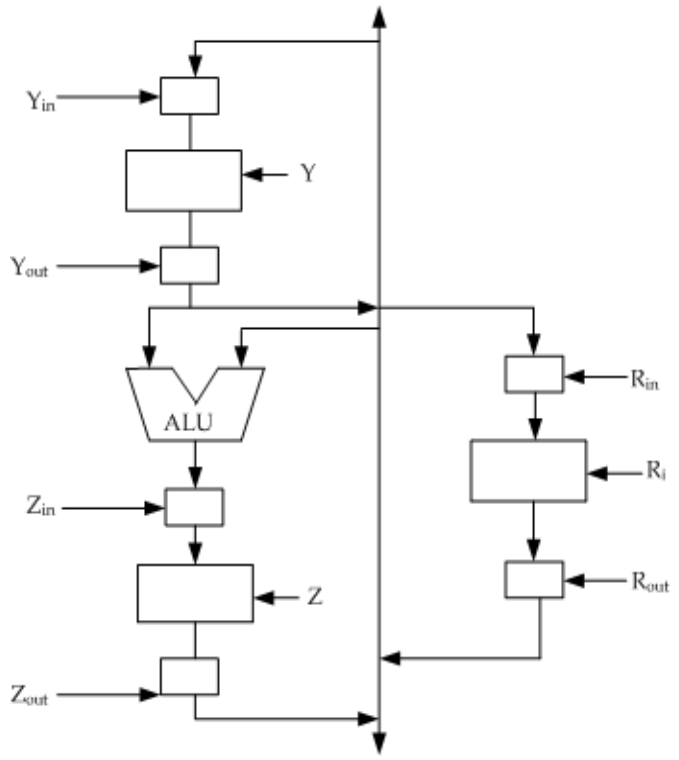
Using Multiplexer

Notes

A common bus can be constructed using a multiplexer. Multiplexer helps in selecting the source register to place the binary information on the bus. The bus register has input and output gating controlled by control signals.

The figure 3.5 depicts the input and output gating of registers.

Figure 3.5: Input and Output Gating of Registers



In figure 3.5:

1. Ri is the register and Rin and Rout are the input and output gating signals of Ri.
2. Z is the register and Zin and Zout are the input and output gating signal of register Z.
3. Y is the register and Yin and Yout are the input and output signals of Y.

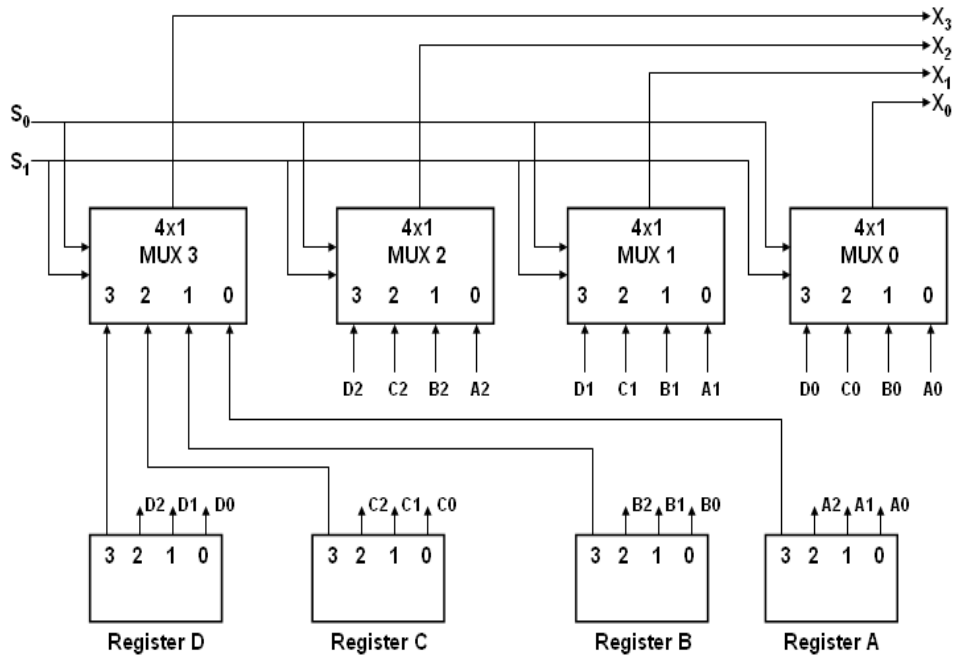
The figure 3.5 illustrates input and output gating, that is, the switches controlled by control signals. In the figure 3.5, **Rin** and **Rout** are the input and output gating of the register **Ri**. When the signal is **ON**, **Ri** is set to **1** and when the signal is **OFF**, **Ri** is set to **0**.

When the input gating Rin is set to 1, the data is loaded into the register bus Ri available on the common bus. Similarly, when Rout is set to 1, the contents of the register Ri are placed on the data bus. Hence, they are known as input enabled and output enabled signals. The operation that takes place within the processor is in sync with the clock pulse.

Notes

The figure 3.6 depicts a bus that uses a 4-bit multiplexer.

Figure 3.6: Bus Using 4-bit Multiplexer



Source: Computer organization and architecture by Krishnananda.

In the figure 3.6, the multiplexer with four bit register is illustrated. Each register is of four bits from **0** to **3** and the bus carries **4*1** multiplexers with four data inputs from **0** to **3** through **X0** and X3. Here, **S1** and **S0** are the selection inputs for all the four multiplexers. The connection is made from the output of the registers through the input of the multiplexers.

In the figure 3.6, the output 1 of the register **A** is connected to the input 0 of **multiplexer 1**. Thus, the selected data is loaded into the registers and placed on the data bus to perform the operations by using the register bus transfer. Table 3.3 depicts the register that is selected on the basis of the two switches **S0** and **S1**.

Table 3.3: Register Selected On the Basis of the Two Switches S0 and S1

S0	S1	Register Selected
0	0	A
0	1	B
1	0	C
1	1	D

Three-State Buffers

Three-state buffers can also be used to construct a common bus. The buffer is a region of the memory, which is inserted in between the other devices to prevent multiple interactions and to match the impedance. The buffers supply additional drive and relay capabilities to the bus registers.

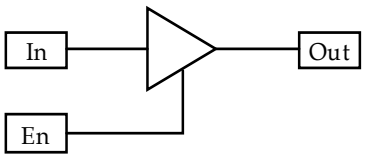
Notes

The three-state buffers are based on the three states, 1, 0, and the open circuit. These three states describe that:

1. The logic 0 and 1 are the two signals equivalent to the ones in conventional gate.
2. The high impedance state means that it does not have the logic significance and the output is disconnected. It also behaves like an open circuit.
3. These three-state gates can perform any conventional logic AND or NAND, OR or NOR, but the most commonly used gate is the buffer gate in the bus register system.

Figure 3.7 depicts the logic symbols and the associated truth table.

Figure 3.7: Logic Symbol and Truth Table



Logical Symbol

En	In	Out
0	X	Z
1	Y	0
1	1	1

Truth Table

According to figure 3.7:

1. When the output is enabled and the control input is equal to 1, the logic gate acts like buffer with the output equal to the input.
2. When the input given is 0, the gate goes to high impedance state Z and the output is disabled.
3. The impedance in three-state buffers connects all the outputs with a wire to form a common bus line and does not endanger the loading effect.
4. The truth table explains that when some input is given and the gate is disabled, it results in high impedance.
5. When the gate is enabled with some input given, then the output results are not in disabled mode.
6. When the gate is enabled with input as 1, the output is equal to 1.

3.3.2 Memory Transfer

The read operation in memory transfer is defined as the transfer of data from the address register (AR) with the selected word M for the memory into the memory buffer register (MBR).

[AR]M MBR=Read Operation

The control signal of the **read** operation initiates the read operation. The read operation statement causes the information transfer from the selected memory register M into the MBR.

The memory transfer in write operation is defined as the transfer of data from the memory buffer register (MBR) to the address register (AR) with the selected word M for the memory.

MBR M[AR]=Write Operation.

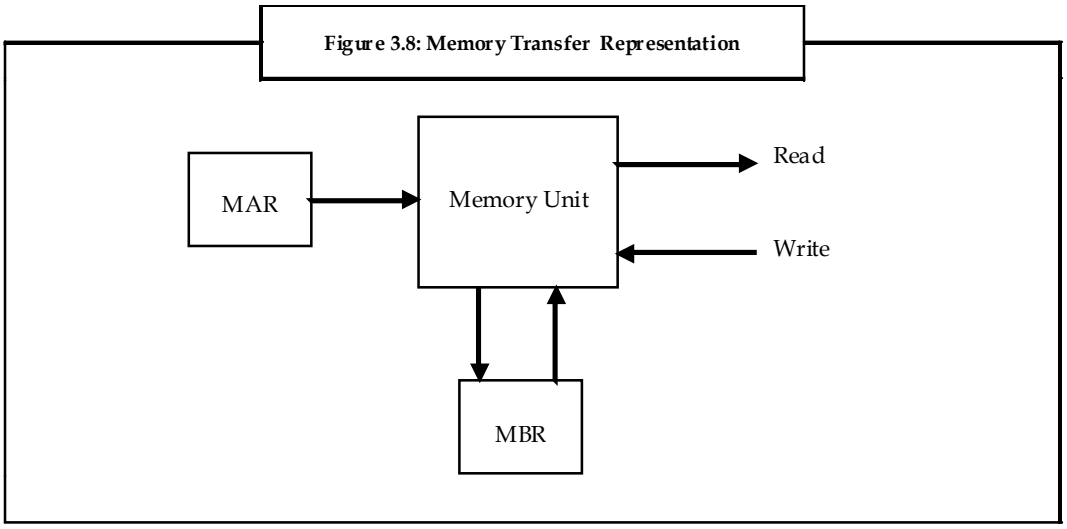
The control signal of the write operation initiates the write operation. The write operation statement causes the information transfer from the MBR into the selected memory register by the address present in the memory M[AR].

To perform either read or write operation, first the memory register (M) must be selected by a specified address.

Notes

The figure 3.8 depicts the memory transfer representation.

Figure 3.8: Memory Transfer Representation



The figure 3.8 shows that the memory unit is used to transfer the information from the memory address register and memory buffer register to perform read and write operations in the memory transfer.

3.4 Microoperation

Microoperations specify the transfer of data to storage. The following sections deals with logic microoperation, shift microoperation and arithmetic logic shift unit.

3.4.1 Logic Microoperation

The logic microoperations for the string of bits stored in registers specify the binary operations. This logic microoperation considers each bit of register separately and treats them as binary variables.

The two registers with exclusive-or microoperation are symbolized as:

A : R1 R1 R2.

The control variable A =1, specifies that the microoperations are to be executed on the individual bits.

Example: Consider registers R1 and R2 each having 4-bits binary numbers .

1010	R1
1100	R2
0110	A=1

In this example, 1010 is the content of register R1, 1100 is the content of R2. After the execution of the microoperation, the content of R1 is equal to the bit-by-bit XOR operation on pairs of bits in R2 and the previous value of R1.

Logical symbols like OR, AND, and their complements are adopted to differentiate them from the corresponding symbols that are used to represent Boolean functions.

The symbol **^** is used to denote **AND** microoperation, the symbol **v** is used to denote **OR** microoperation, and to denote the complement of **AND** and **OR**, **1's complement** is taken with a **bar** on top of the symbol that denotes the register name.

There are some applications of logic microoperations. Some of the applications of microoperations are:

1. It manipulates a part of the memory word or individual bits stored in the registers.
2. It is used to change, delete, and insert new bit values within the memory.

Notes

3. It operates on Selective Set,

A A v B,

Example: 1010 A
1100 B
1110 A

As shown in the above example, the microoperations set a group of bit values.

In the bit register, the microoperation performs the following function:

1. It changes the values in register A to correspond to the values in register B, that is, in places where the register B holds the value 1, the corresponding bits of 'A' are also changed to 1.
2. It operates on Selective ê Complement,

A ê A ⊕ B,

Example: 1010 A
1100 B
0110 A

In this example, microoperation complements the 1 in bit register A, where there are corresponding 1 bits in register B. It does not affect the 0 bits in the register.

3. It operates on Selective Clear,

A ← A ^ $\overline{B}$,

Example: 1010 A
1100 B
0010 A

In this example, microoperation clears the 0 in bit register A, where there are corresponding 1 bits in register B. It affects the 0 bits in the register.

4. Microoperation operates on Selective ê Mask,

A ê A ^B,

Example: 1010 A
1100 B
0010 A

It masks the group of bit values. It is same as that of selective clear, but the bits of 'A' are cleared only when there are corresponding 0 bits in 'B'.

Notes

3.4.2 Shift Microoperation

The transfer of data is done using the shift operation. Shift Microoperation is used in logic, arithmetic and other data processing conventions in conjunction. The content of bits from the registers is shifted to left and right. After the bits are shifted, the flip-flops receive the information from the serial input.

During the left shift operation, the bits are shifted to the rightmost position by the serial input operations. During the right shift operation, the bits arte shifted to the leftmost position by the serial input operations.

Many arithmetic operations like multiplications and divisions require shifting of the operands. For general operands, logical shift is used, which preserves the sign of the number. The table 3.4 shows the different shift operations.

Table 3.4: Different Shift Operations

Register	Description
R	Lshl R Shift left register
R	Lshr R Shift right register
R	cil R Circular left shift register
R	cir R Circular right shift register
R	ashl R Arithmetic left shift register
R	ashl R Arithmetic right shift register

There are three different types of shift operations. They are:

1. Logical shift operation
2. Circular shift operation
3. Arithmetic shift operation

The following section deals with the explanation of each of these shift operations.

Logical Shift Operation

There are two logical shift operations **Lshl** for the logical left shift register and **Lshr** for the logical right shift register. Logical shift instructions shift an operand by a number of bit positions specified in the logical shift instruction.

General form of logical left shift operation

Lshl count, R

In this form of logical shift operation, the count is the immediate operand or it may be given in a processor register. When bits are shifted left, the rightmost bit is filled with zeros. The bits shifted from the MSB position are passed through a carry flag **C** and are lost. **R** is the processor register.

General form of logical right shift operation

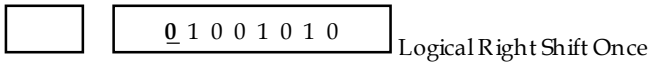
Lshr count, R

In this form of logical shift operation, the leftmost positions are filled with zeros when bits are shifted to the right. The bits that are shifted from the MSB position and passed through a carry flag **C** are lost. Involving carry flag in the shift operations helps in performing operations on large numbers.

Notes

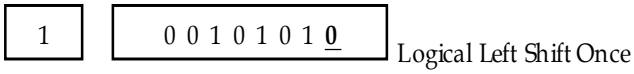
Suppose the register holds the value 10010100, the logical right shift operation performed on this value fetches the result as shown in figure 3.9.

Figure 3.9: Logical Right Shift Operation



Suppose the register holds the value 01010101, the logical left shift operation performed on this value fetches the result as shown in figure 3.10.

Figure 3.10: Logical Left Shift Operation



The figure 3.9 depicts a logical right shift by 1 bit and the figure 3.10 depicts a logical left shift by 1 bit.

3.4.3 Arithmetic Logic Shift Unit

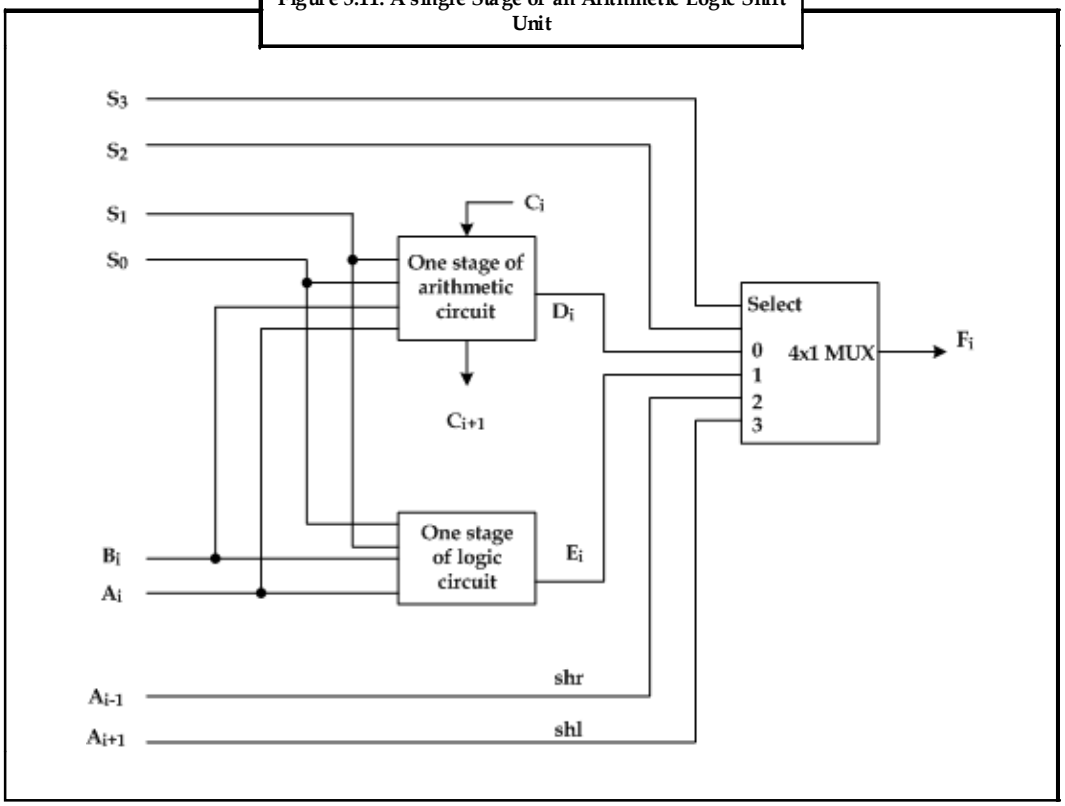
Computer systems make use of a number of storage registers that are connected to common operational unit. This unit is called as an arithmetic and logic unit (ALU). The contents of the specified source register form the inputs of the ALU. The ALU performs an operation and transfers the result to the destination register. The ALU comprises combination of circuits that transfers the content from source register to destination register in one clock pulse period. Usually, the shift microoperation is performed in a separate unit. However, sometimes the shift unit becomes part of the overall unit. The arithmetic, logic, and shift circuits can be incorporated into one ALU with common selection variable.

The figure 3.11 shows a single stage of an arithmetic logic shift unit. The subscript 'i' indicates a typical stage. A_i and B_i are the two inputs assigned to arithmetic as well as logic units. A microoperation is selected with inputs S_1 and S_0. E_i is the arithmetic output and D_i is the logic output. Multiplexer 4x1 chooses between arithmetic output and logic output. S_3 and S_2 are the inputs used to select data in the multiplexer. The other inputs to the multiplexer are A_{i-1} for the shift-right operation and A_{i+1} for the shift-left operation.

Notes

Figure 3.11 depicts a single stage of the arithmetic logic sift operation.

Figure 3.11: A single Stage of an Arithmetic Logic Shift Unit



This circuit is repeated 'n' number of times to obtain an 'n' bit ALU. The output carry denoted by C_{i+1} of one stage is connected to the input carry denoted by C_i of the next stage. The connection between the output carry and the input carry happens in a sequential manner. The input carry of the first stage is denoted by C_i and helps in selecting the variable for the arithmetic operation.

Notes

The table 3.5 gives the 14 operation of the ALU.

Table 3.5: ALU Function Table

Select Operation

S_0	S_1	S_2	S_3	C_i	Operation Performed	Description
0	0	0	0	0	$F - A$	Transfer A
0	0	0	0	1	$F - A - 1$	Increment A
1	0	0	0	0	$F - A \quad B$	Addition
1	0	0	0	1	$F - A - B - 1$	Add with carry
0	1	0	0	0	$F - A \quad \overline{B}$	Subtract with borrow
0	1	0	0	1	$F - A - \overline{B} - 1$	Subtraction
1	1	0	0	0	$F - A - 1$	Decrement A
1	1	0	0	1	$F - A$	Transfer A
0	0	1	0	x	$F - A \wedge B$	AND
1	0	1	0	x	$F - A \vee B$	OR
0	1	1	0	x	$F - A \oplus B$	XOR
1	1	1	0	x	$F - \overline{A}$	Complement A
x	X	0	1	x	$F - shrA$	Shift right A into F
x	X	1	1	x	$F - shlA$	Shift left A into F

The first eight operations relate to arithmetic operations and are selected based on the variables $S_3, S_2 = 00$. The next four operations relate to logical operations and are selected based on variables $S_3, S_2 = 01$.

Notes The Input carry denoted by Ci does not have any kind of impact on the logic operations and hence is marked as 'don't cares' (X). The last two operations relate to the left shift and right shift operations and are selected based on the variables S3, S2 = 10 and S3, S2 = 11.

3.5 Summary

- Binary numbers are used to represent the 0s and 1s information in computers.
- 1's complement addition can be performed by adding two binary numbers including the signed bit.
- 1's complement subtraction has two cases; subtracting a smaller number from a larger number and subtracting the smaller number from a larger number.
- 2's complement addition and subtraction is similar to 1's complement. But in 2's complement the numbers are represented by adding 1 to a 1's complement number.
- Fixed-point numbers are used to represent the factorial numbers.

Notes

- Floating-point number is used to represent a decimal point which is multiplied by a base value and it is scaled up with some exponent value.
- There are three ways to represent the magnitude of the signed binary numbers namely, the sign and magnitude representing, the signed and 1's complement representation, and the signed and 2's complement representation.
- The term register is the storage space that holds the data and instructions within itself.
- There are two ways for register transfer namely bus transfer and memory transfer.
- Logical microoperations consider each bit of register separately and treat them as binary variables.

3.6 Keywords

Carry Flag: A single bit in a system status (flag) register used to indicate when arithmetic carry or borrow has been generated out of the most significant ALU bit position.

Impedance: A measure of opposition that a circuit presents to an alternating current.

Operand: The part of a computer instruction which specifies what data is to be manipulated or operated on.

Radix Point: A symbol used in number representations to separate the integer part of the number from the fractional part of the number.

3.7 Self Assessment

1. State whether the following statements are true or false:
   (a) During the left shift operation, the bits are shifted to the leftmost position by the serial input operations.
   (b) The logical microoperations for the string of bits stored in registers specify the binary operations.
   (c) The buffers supply additional drive and relay capabilities to the bus registers.
   (d) Control signals determine which register to be selected during each register transfer.
   (e) Microoperation is an operation that operates on the data stored in the registers.
   (f) The binary numbers that are unsigned are always treated as positive integers and are represented as 1s in the MSB.
2. Fill in the blanks:
   (a) The _____ is 0 for positive numbers and 1 for negative.
   (b) Many arithmetic operations like multiplications and divisions require shifting of _____
   (c) The bus can be constructed by using the _____ instead of using multiplexers.
   (d) The _____ is the one way of constructing a common bus.
   (e) The _____ representations are used to perform operations for high range values.
3. Select a suitable choice for every question:
   (a) Which of the following is obtained by inverting all 0s to 1s and all 1s to 0s?
      (i) 1's complement
      (ii) 2's complement
      (iii) 3's complement
      (iv) Factorial number

Notes

(b) Which is the operation that is considered as the key to perform basic arithmetic operations such as, addition, subtraction, multiplication and division?

(i) Decimal

(ii) Binary

(iii) Hexadecimal

(iv) Octal

(c) Which is the easy way to represent factorial numbers?

(i) Floating-point numbers

(ii) Decimal numbers

(iii) Fixed-point numbers

(iv) Overflow

(d) Which flag is used by the unsigned numbers?

(i) Carry flag

(ii) Overflow flag

(iii) Parity flag

(iv) Check flag

3.8 Review Questions:

1. "In computers, binary number system is used for storing this information as 0's and 1's." Elaborate.
2. "In 1's complement addition, when two numbers are added, the two binary numbers may be added including the sign bit." Comment.
3. "The rule for the subtraction of n-bit signed integer numbers using 2's complement is to subtract two negative numbers, add the two negative numbers." Justify.
4. "Overflow is defined as a result of arithmetic operations that does not fit into the given integer size or the value that moves out-of-range." Comment.
5. "Fixed-point representation has a radix point called decimal point'." Justify.
6. "Three approaches can be used to represent the magnitude of the signed binary numbers." Comment.
7. "Just like a fixed-point representation, the floating-point also contains integers and fractions separated by a radix point." Justify.
8. "In normalized floating-point representation, the most significant digit of the mantissa is non-zero." Comment.
9. "To transmit data and instructions form one register to another register, memory to register and memory to memory, the register transfer method is used." Elaborate.
10. "The bus register has input and output gating controlled by control signals." Elaborate.
11. "The three-state buffers are based on the three states, 1, 0, and the open circuit." Elaborate.
12. "The logic microoperations for the string of bits stored in registers specify the binary operations." Comment.

Notes

Answers: Self Assessment

1. (a) False (b) True (c) True
   (d) True (e) True (f) False
2. (a) Most Significant bit (MSB) (b) Operand (c) Three-state buffer
   (d) Multiplexer (e) Floating-point
3. (a) 1's complement (b) Binary (c) Fixed-point number
   (d) Carry flag

3.9 Further Readings

Books

Godse, A.P., & Godse, D.A. (2010). Computer Organization and Architecture, 1st ed. Pune: Technical Publications.

Rajaraman, V., & Radhakrishnan, T. (2007). Computer Organization and Architecture. New Delhi: PHI Learning Private Limited.

Online links

http://www.dspguide.com/ch28/4.htm

http://www.google.co.in search?hl=en&biw=1024&bih=578&q=what+is+a+fkip+flop&aq=f&aqi=&aql=&oq=

Notes

Unit 4: Computer Organization I

CONTENTS

Objectives
Introduction
4.1 Instruction Codes and Operands
4.2 Computer Registers
4.2.1 Common Bus System
4.3 Computer Instructions
4.3.1 Memory Reference Instruction
4.3.2 Register Reference Instruction
4.3.3 Input/Output Instructions
4.4 Summary
4.5 Keywords
4.6 Self Assessment
4.7 Review Questions
4.8 Further Readings

Objectives

After studying this unit, you will be able to:

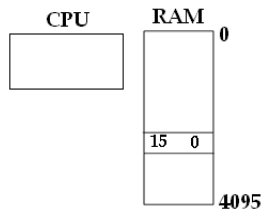
- Define instruction codes and operands
- Discuss computer registers
- Explain computer instructions

Introduction

Computer processors can be classified into various categories depending on their components such as registers, buses, micro operations, machine instructions, and so on. These processors are complex devices. They contain a number of registers and arithmetic units (for both integer and floating point calculations). They can work on multiple instructions in a simultaneous manner and speed up the execution.

The figure 4.1 depicts the components of a basic computer. In general, a computer consists of two components , a processor and a memory. The memory has **4096 = 2^{12}** bytes in it.

Figure 4.1: Computer Components



Notes

A program refers to a sequence of instructions, which are stored in the memory of a computer. These instructions are a group of bits that enable the computer to perform a specific operation. They are read by the Central Processing Unit (CPU) and are placed in Instruction Register (IR). The circuitry control then converts these instructions into a sequence of micro operations that are required for their implementation.

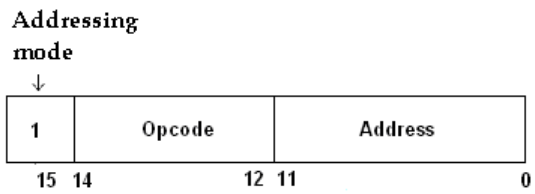
4.1 Instruction Codes and Operands

A computer instruction is a binary code that specifies the micro-operations in a sequence for a computer. They are stored in the memory along with the data. Every computer has its own unique set of instructions. They can be divided into two parts namely, **Operation codes** (Opcodes) and **Address**. Opcodes specifies the operation for a particular instruction. An address specifies the registers or the locations that can be used for that operation. **Operands** are specific parts of computer instruction that depict what data is to be operated on.

For a basic computer, **12 bits** of memory is required to specify the address as the memory contains **4096 words**. The **15th bit** of the instruction specifies the addressing mode (where direct addressing corresponds to 0, indirect addressing corresponds to 1). Since the instruction format consists of **12 bits** of address and **1 bit** for the addressing mode, **3 bits** are left for Opcodes.

Figure 4.2 gives a pictorial representation of the above said instruction format.

Figure 4.2: Instruction Format



As we can see in figure 4.2, instruction format has three parts.

Addressing Modes

Instructions generally refer to the address of a specific memory location. These are called memory reference instructions. The way in which a target address or effective address is identified within the instruction is called addressing mode.

The address field for an instruction can be represented in two different ways. They are:

1. ***Direct Addressing:*** It uses the address of the operand.
2. ***Indirect Addressing:*** It uses the address as pointer to operand.

The address of the operand or the target address is known as the effective address.

Effective Address (EA) – It refers to the address that can be implemented as a target address for a branch type instruction or the address that can be used directly to access an operand for a computation type instruction, without developing any changes.

Opcodes

An opcode is a set of bits that defines the basic operations such as add, subtract, multiply, complement, and shift. The total number of operations supported by the computer decides the number of bits required for the opcode. The minimum bits available to the opcode must be n for 2n operations. These operations are performed on data that is stored in processor registers or in memory.

Notes

Notes Processors have registers to hold instructions, addresses, and data. Program Counter (PC) is the register used to store the memory address of the instruction that is to be executed next. A PC requires 12 bits to hold the memory address.

An effective way of organizing a computer is to divide it with one processor register and one instruction code format that has two parts. The former part of the instruction code specifies the operation that is to be performed and the latter indicates the address for the same. It indicates the address of the operand to the control.

Commonly, computers come with a single processor register assigned to them named, Accumulator (AC). Operations are performed with the content in AC and the memory operand. In case an operation in the instruction code does not require the operand from memory, the remaining bits can be utilized for other purposes.

Example: Operations in the instruction code include clear AC, increment AC, complement AC, and so on. These operations do not need the second part of the instruction code.

Did u know? Address bits are also used as operands in some cases.

Address

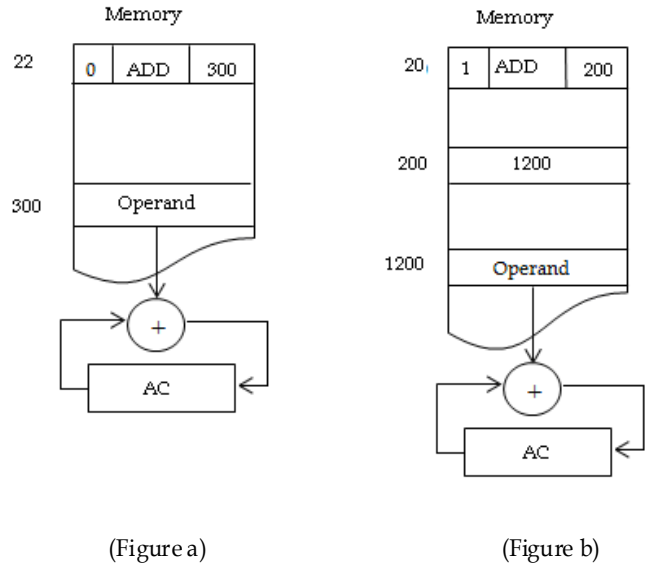
Address is defined as the location where a particular instruction is found in the memory.

Sometimes the address bits of an instruction code are used as operand and not as an address. In such cases, the instruction has an immediate operand. If the second part has an address, the instruction is known to have a direct address. Another possibility is, the second part having the address of the operand. This is known as indirect address. In the instruction code, one bit is used to indicate if direct or indirect address is implemented.

Notes

Figure 4.3 depicts a diagram showing direct and indirect address.

Figure 4.3: Diagram Showing Direct and Indirect Address



(Figure a) (Figure b)

Example:

Referring to figure 4.3,

Consider Figure a,

Here,

Addressing mode is 0, which indicates direct addressing.

Operand is ADD.

Address is 300. This indicates the address of the operand.

The instruction is stored in the 22nd location. The control jumps to the 300th location to access the operand.

Consider Figure b,

Here,

Addressing mode is 1, which indicates indirect addressing.

Operand is ADD.

Address is 200. This indicates the address of the operand. The control goes to the address 200 to get the address of the operand. Here, the address is 1200. The operand found in this address location is added to the data in accumulator.

Here the address of the operand is known as the effective address. In the first figure the effective address is 300 and in the second figure it is 1200. The memory word that contains the address of the operand in the indirect address is used as a pointer to an array of data. The pointer is located in the processor register.

Notes

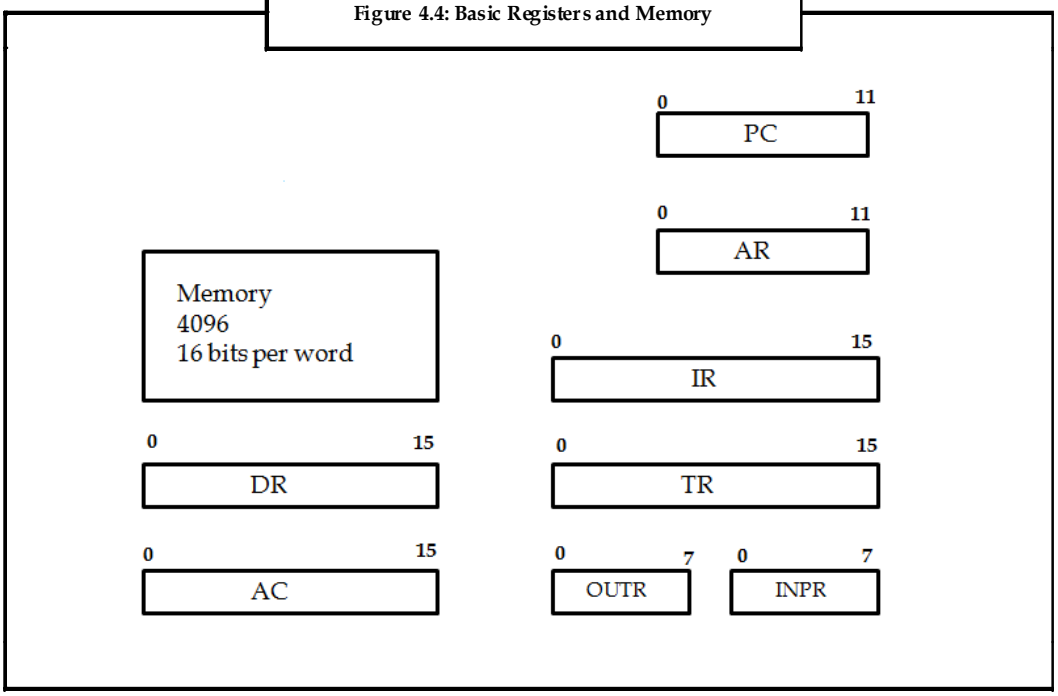
4.2 Computer Registers

Computer registers are high speed memory storing units. They are a part of the computer processor. They can hold any kind of data, such as a bit sequence or an individual data. Instructions specify registers as part of them in most cases. A register must be 32 bits in length for a 32 bit instruction computer. Registers can be numbered depending on the processor design and language rules.

Notes If two numbers are to be added, both of them must be contained in registers and their sum is also stored in a register. The register can also contain the address of a memory location where the data is stored and not the data itself.

The instructions in a computer are stored in memory locations and executed one after another at a time. It is the function of control unit to retrieve the instruction from the memory and execute it. The control does the same for all the instructions in the memory in a serial order. A counter is required to keep a track of the next instruction to be executed and calculate its address. In addition to counter, a register is also required to store the instruction code after it is read by the control from memory. Processor registers are needed for data manipulation. The figure 4.4 displays the registers with their memories. The memory addresses are stored in a different register. These requirements clearly state the need of registers in a computer. The figure 4.4 depicts basic registers and memory.

Figure 4.4: Basic Registers and Memory



In figure 4.4, different registers are shown with their memory capacity.

Notes

Table 4.1 lists the registers and their functions.

Table 4.1: Registers and their Functions

Register Symbol	Number of Bits	Register Name	Function
OUTR	8	Output register	It holds output character
INPR	8	Input register	It holds input character
PC	12	Program Counter	It holds address of instruction
AR	12	Address register	It holds address for memory
DR	16	Data register	It holds memory operand
AC	16	Accumulator	It's a processor register
IR	16	Instruction register	It holds instruction code
TR	16	Temporary register	It holds temporary data

The explanation for each of the registers specified in figure 4.4 is as follows:

1. The data register holds the operand read from the memory.
2. The accumulator is a general purpose register used for processing.
3. The instruction register holds the read memory.
4. The temporary data used while processing is stored in the temporary register.
5. The address register holds the address of the instruction that is to be executed next from the memory.
6. The Program Counter (PC) controls the sequence of instructions to be read. In case a branch instruction is encountered, the sequential execution does not happen. A branch execution calls for a transfer to an instruction that is not in sequence with the instructions in the PC. The address of this non-consecutive instruction is passed on to the PC to replace the existing instruction address. To read an instruction, the memory cycle is initiated again with the PC content taken as address for memory. Next, the PC is incremented by one and the previous order continues.
7. The input register (INPR) and output register (OUTPR) are the registers used for the I/O operations. The INPR receives an 8 bit character from the input device, same with the OUTPR.

Example: Here is a list of registers and some of their examples:

Data register - MIX, Motorola 680*0, 68300

Accumulator - Intel 8086/80286

Address register - Motorola 680x0, 68300

Program counter - IBM 360/370, Motorola 680x0, 68300

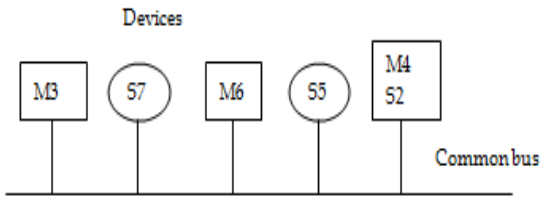
Notes

4.2.1 Common Bus System

A collection of signal lines that help in the transfer of multi bit information from one system to another is called a bus.

Figure 4.5 shows three master devices M3, M6, and M4.

Figure 4.5: Common Bus System



The master device initiates and controls the communication. S7, S5, and S2 are the slave devices. Slave devices respond to the commands given by master devices. If M3 wants to give a command to S5, it must send its instruction through the bus. Then, the S5 receives the instruction and takes action on the instruction through the bus.

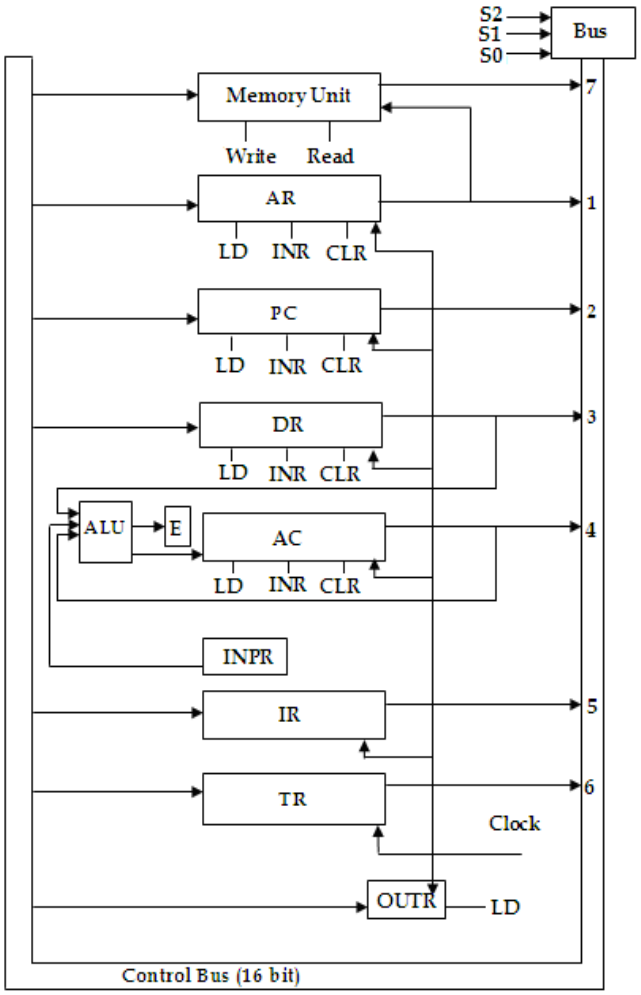
A basic computer consists of eight registers, a memory unit, and a control unit. These units need to communicate often. Bus provides the medium through which communication can take place.

Previously, wires were used to connect the outputs of each register to the input of other registers. However now, the common bus system has replaced them making communication more efficient. The common bus provides a path between the memory unit and the registers.

Notes

In the figure 4.6, the outputs of the seven registers and memory are connected to a common bus. S_0, S_1, S_2 are the selection values that are connected to the bus.

Figure 4.6: Basic Computer Registers Connected to the Memory Unit through Common Bus



The selection values determine the output that is selected for the bus lines at any given time. Each of the output has a number along with it, which indicates the decimal equivalent of the required binary selection.

Example: The number along the 16 bit output of common bus for DR is 3. When the S2S1S0 = 011, the 16 bit outputs are placed on the bus lines, since it is the binary value of decimal 3. These lines are linked with the data inputs from the registers and the memory.

Notes

Now let us discuss the different registers and their functions:

1. ***Load (LD):*** During the next clock pulse transition the data from the bus is sent to the register whose load (LD) input is enabled.

2. ***Memory Unit:*** When the write input of the memory is activated, it receives the content of the bus. When the read input is activated, the memory places the 16 bit output onto the bus with the selection variables being $S_2S_1S_0$=111.

3. ***Increment (INR) and Clear (CLR):*** When the INR signal is enabled, the contents of the specific register are incremented. The contents are cleared when the CLR signal is enabled.

4. ***Address Registers (AR):*** Here, the address of the memory for the next read and write operation is specified. It receives or sends address from or to bus when selection inputs $S_2S_1S_0$=001 is applied and the load is enabled. With inputs INR and CLR, the address gets incremented or cleared.

5. ***Program Counter (PC):*** Here, the address of the next instruction that is to be read from the memory is stored. It receives or sends address from or to bus when selection inputs $S_2S_1S_0$=010 is applied and the load input is enabled. With inputs INR and CLR, the address gets incremented or cleared.

6. ***Data Register (DR):*** The data register contains the data to be written into memory or data that is to be read from the memory. It receives or sends address from or to bus when selection inputs $S_2S_1S_0$=011 is applied and the load input is enabled. With inputs INR and CLR, the address gets incremented or cleared.

7. ***Accumulator (AC):*** Accumulators are useful in implementing the register micro operations such as complement, shift, and so on. The results obtained are again sent to accumulator. An accumulator stores the intermediate arithmetic and logic results. The processor register AC receives or sends its data to the bus when the selection inputs $S_2S_1S_0$=100 is applied and the load input of DR is enabled. The contents of DR are sent via the adder/logic circuit into AC with the load enabled. With inputs INR and CLR, the address gets incremented or cleared.

8. ***Instruction Registers (IR):*** The IR stores the copy of the instruction that the processor has to execute. The instruction that is read from the memory is stored in the IR. It receives or sends instruction code from or to bus when selection inputs $S_2S_1S_0$=111 is applied and the load input is enabled.

9. ***Temporary Register (TR):*** The temporary storage for variables or results is provided by the temporary register. It receives or sends the temporary data from or to bus when selection inputs $S_2S_1S_0$=011 is applied and the load input is enabled. With inputs INR and CLR, the address gets incremented or cleared.

10. ***Input Registers (INPR):*** It consists of 8 bits to hold the alpha numeric input information. Input device shifts its serial information into the 8 bit register. The information is transferred to AC via the adder/logic circuit with load enabled.

11. ***Output Registers (OUTPR):*** The information here is received from AC and transferred to the output device.

Notes The content from any of the registers can be added to the bus and the operation can be performed by adder/logic circuit in the same clock cycle.

Notes

Example: Suppose two micro operations need to be executed simultaneously,

PC ê AC and AC PC

These operations can be performed by placing the content of AC onto the bus with the selection inputs S2S1S0 = 100 and enabled load (LD) of PC, placing content from PC to AC through adder/logic circuit and enabling load input of AC during the unchanged clock cycle.

4.3 Computer Instructions

A computer has programs stored in its RAM in the form of 1s and 0s that are interpreted by CPU as instructions. One word of RAM contains one instruction in the machine language. These instructions are loaded to the CPU one at a time, where it gets decoded and executed. A basic computer has three instruction code formats, namely the memory reference instruction, the register-reference instruction, and the input-output instruction format.

4.3.1 Memory Reference Instruction

Usually programmers write codes in assembly language which has a very close reference with machine language. These instruction formats are known as memory reference instructions.

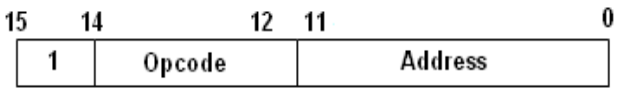
The different memory reference instructions are tabulated in table 4.2.

Table 4.2: Memory Reference Instructions

Opcode	Instruction	Function
000	LDA	Load A register from memory
001	LDB	Load B register from memory
010	STA	Store A register in memory
011	STB	Store B register in memory
100	ADDA	Add A register to memory
101	SUBA	Subtract A register from memory

The memory reference instruction has a specific format. It consists of three parts as shown in figure 4.7.

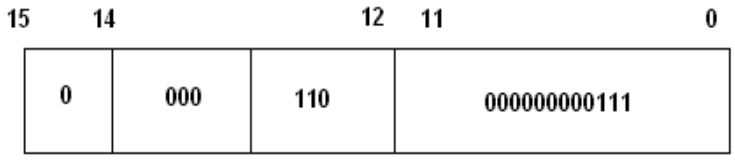
Figure 4.7: Memory Reference Instruction Format



Notes

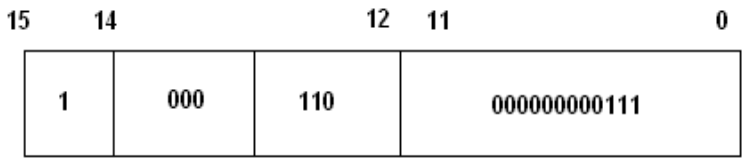
In figure 4.8, the opcode varies from 000 to 110. The first bit is the indirection bit. The memory reference instruction is performed on the contents of the address in bits 0-11 of the instruction word, if the indirect bit is 0.

Figure 4.8: Memory Reference Instruction Format for Direct Addressing



In figure 4.8, the indirection bit is given as 0 and the opcode as 000 indicating LDA. The address, given in binary as 000000000111 is equivalent to decimal 7. This states that the Register A will be loaded with contents of word 7 from memory. In case the indirect bit is changed to 1, the word 7 in memory would contain the address of the word that is to be loaded by Register A and not direct content. Figure 4.8 illustrates the memory reference instruction format for direct addressing. The 15th bit here is 0.

Figure 4.9: Memory Reference Instruction Format for Indirect Addressing

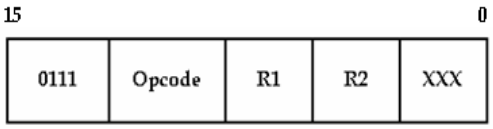


In figure 4.9, the indirection bit is given as 1 and the opcode as 000 indicating LDA. The address, given in binary as 000000000111 is equivalent to decimal 7. This states that the register A will be loaded with contents of word 7 from memory. As the indirect bit is 1, the memory contains the address of word that is to be loaded by register A and not direct content. Figure 3.9 illustrates the memory reference instruction format for indirect addressing. The 15th bit here is 1.

4.3.2 Register Reference Instruction

The register reference instructions use 16 bits to specify any operation. All these instructions take 1 clock cycle during execution. Figure 4.10 depicts its format.

Figure 4.10: Register Reference Instruction Format



Notes

In figure 4.10, the four leftmost bits are always **0111** which is the binary equivalent of **hexadecimal 7.** The opcodes and the registers that can be implemented are listed in table 4.3.

Table 4.3: Register and their Description

R1 or R2	Instruction	Description
000	A	General purpose register A
001	B	General purpose register B
010	X	Index register
011	SP	Stack pointer
100	CCR	Condition code register
101	SR	Shift register

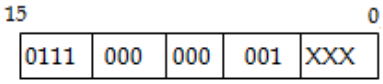
Table 4.4 depicts opcodes and their description.

Table 4.4 : Opcodes and their Description

Opcode	Instruction	Description
000	MOV	Move R1 to R2
001	INC	Increment
010	DEC	Decrement
011	ADD	Add R1 to R2
100	SUB	Subtract R1 from R2
101	XOR	Exclusive OR
110	CMP	Complement
111	SHIFT	Shift left

Other than the leftmost 4 bits, the next nine bits hold the operand and registers, and the last three are don't cares. Consider an instruction as shown in figure 4.11.

Figure 4.11: Instruction



In figure 4.11, **0111** indicates that the instruction is register reference. When the processor loads and executes the instruction, it moves the contents of Register A to Register B. A programmer would actually write this instruction in the coding format as:

MOV A, B

This is called assembly language and is translated into machine language by an assembler. The machine language format of this instruction is stored in a disk. The operating system then loads it from the disk to memory and causes the CPU to execute it.

Notes

4.3.3 Input/Output Instructions

The I/O instructions are similar to the register reference instructions. They also use a 16-bit format to specify an operation. The difference is that the leftmost bits here are always **1111,** which is the binary equivalent of hexadecimal **F.**

Types of Instructions

A basic computer must have a specific set of instructions so that it is helpful for the user to develop the machine language programs with ease and evaluate the computable functions. Following are the different types of instructions that a basic computer should possess:

1. Arithmetic, logical, and shift instructions.
2. Instructions for the movement of data from memory and registers.
3. Program control and status check instructions.
4. Input and output instructions.

Table 4.5 lists some of the basic computer instructions available.

Table 4.5: Basic Computer Instructions

Type of instruction	Symbol	Description
Arithmetic instruction	ADD	Add memory word to AC
Store instruction	STA	Store content to AC in memory
Branch instruction	BUN	Branch unconditionally
	BSA	Branch and save return address
	ISZ	Increment and skip if zero
Logical instruction	AND	AND memory word to AC
	CLA	Clear AC
	CMA	Complement AC
Check status instruction	CLS	Clear all status flags
Circulate instruction	CIR	Circulate right AC
	CIL	Circulate left AC
Increment instruction	INC	Increment AC
Skip instructions	SPA	If AC is positive, skip next instruction
	SNA	If AC is negative, skip next instruction
	SZA	If AC is zero, skip next instruction
	SZE	Skip instruction if E is 0
	HLT	Halt computer
Input/Output instruction	INP	Input character to AC
	OUT	Output character from AC
	SKI	Skip on input flag
	SKO	Skip on output flag
	ION	Interrupt on
	IOF	Interrupt off

Although the set of instructions is complete for a basic computer, it can become more efficient only if it has a set of instructions like subtract, multiply, OR, and exclusive OR.

Did u know? 1000 gigabytes = Storing a stack of documents that is greater than 16 times the height of New York's empire state building.

Notes

4.4 Summary

- The binary codes that specify the micro operations in a sequence are called computer instructions. An instruction code format consists of addressing mode, operation codes, and address.
- For a basic computer, 12-bits of memory is required to specify the address since the memory contains 4096 words.
- Computer registers are used to store the data. They are high speed storage areas within the processor. Before every processing, the data must be represented in the registers.
- A basic computer consists of eight registers, a memory unit, and a control unit. These units need to communicate often. Bus provides the medium through which communication can take place.
- Computer instructions are programs written in the form of 0's and 1's as interpreted by the CPU. The instruction codes are divided into three categories, memory reference instructions, register reference instructions, and Input/Output instructions.

4.5 Keywords

Accumulator: A register that contains a built-in adder, which is used to add an input number to the contents of the register.

Master Device: A device that controls one or more other devices.

RAM: Random Access Memory

4.6 Self Assessment

1. State whether the following statements are true or false:
   (a) Opcodes specifies the operation for a particular instruction.
   (b) Effective address cannot be implemented as a target address for a branch type instruction.
   (c) Previously buses were used for connecting devices but now wires are used for the same.
2. Fill in the blanks:
   (a) The main components of a computer are _____ and processor.
   (b) _____ can be numbered depending on the processor design and language rules.
   (c) The minimum bits available to the opcode must be _____ for 2n operations.
3. Select a suitable choice for every question:
   (a) The INP instruction receives data that is sent by…… to AC.
      (i) OUTPR
      (ii) INPR
      (iii) DR
      (iv) TR
   (b) Which instruction is used to branch a part of the program?
      (i) Increment if zero
      (ii) Branch unconditionally
      (iii) Load register
      (iv) Branch and save

Notes

(c) When the load is …… the content from the source register or memory can be transferred to/from bus.

(i) Enabled

(ii) Disabled

(iii) Stored

(iv) Retrieved

(d) What is a binary code called when it specifies the micro operations in a sequence?

(i) Sequence counter

(ii) Computer instructions

(iii) Computer codes

(iv) Instruction codes

4.7 Review Questions

1. "The format of instruction code has three sections, namely addressing mode, opcode, and address." Explain.
2. Discuss the different computer registers and their importance.
3. "Computer buses help in the transmission of data between the components connected to it." Discuss.
4. "Instruction cycles are concerned with the fetch and execute cycle." Justify.

Answers: Self Assessment

1. (a) True

   (b) True

   (c) False

2 (a) Memory

   (b) Registers

   (c) n

3. (a) INPR

   (b) Branch and save

   (c) Enabled

   (d) Computer instructions

Notes

4.8 Further Readings

Books

Radhakrishnan, T., & Rajaraman, V. (2007). Computer Organization and Architecture. New Delhi: Raj Kamal Electric Press.

Rauss, R. (1998). Essentials of computer science 2. USA: Research and education association

Online links

http://www.scribd.com/doc/19731285/Computer-organisation

http://cnx.org/content/m29708/latest/

Notes

Unit 5: Computer Organization II

CONTENTS

Objectives
Introduction
5.1 Timing and Control
5.2 Instruction Cycle
5.3 Memory Reference Instructions
5.4 Input/Output
5.4.1 Input/Output Configuration
5.4.2 Input/Output Instruction
5.5 Design of a Basic Computer
5.5.1 Control of Logic Gates
5.5.2 Control of Registers and Memory
5.5.3 Control of Single Flip-Flop
5.5.4 Control of Common Bus
5.6 Summary
5.7 Keywords
5.8 Self Assessment
5.9 Review Questions
5.10 Further Readings

Objectives

After studying this unit, you will be able to:

- Elucidate timing and control
- Explain instruction cycles
- Discuss memory reference instructions
- Discuss I/O and interrupts
- Analyze the design of basic computers

Introduction

The main function that a computer performs is to execute a program. A program involves a set of instructions that are stored in the memory. A processor executes these instructions. Instruction codes are used to execute an instruction. Timing and control is required to execute the instruction codes that were discussed in the previous unit.

Instructions are executed using two cycles, which are discussed in this unit. The memory reference instructions are also discussed here.

5.1 Timing and Control

Essentially, timing and control are used to execute the instruction codes, register and computer instructions. In a basic computer, the timing for all the registers is controlled by a master clock

Notes

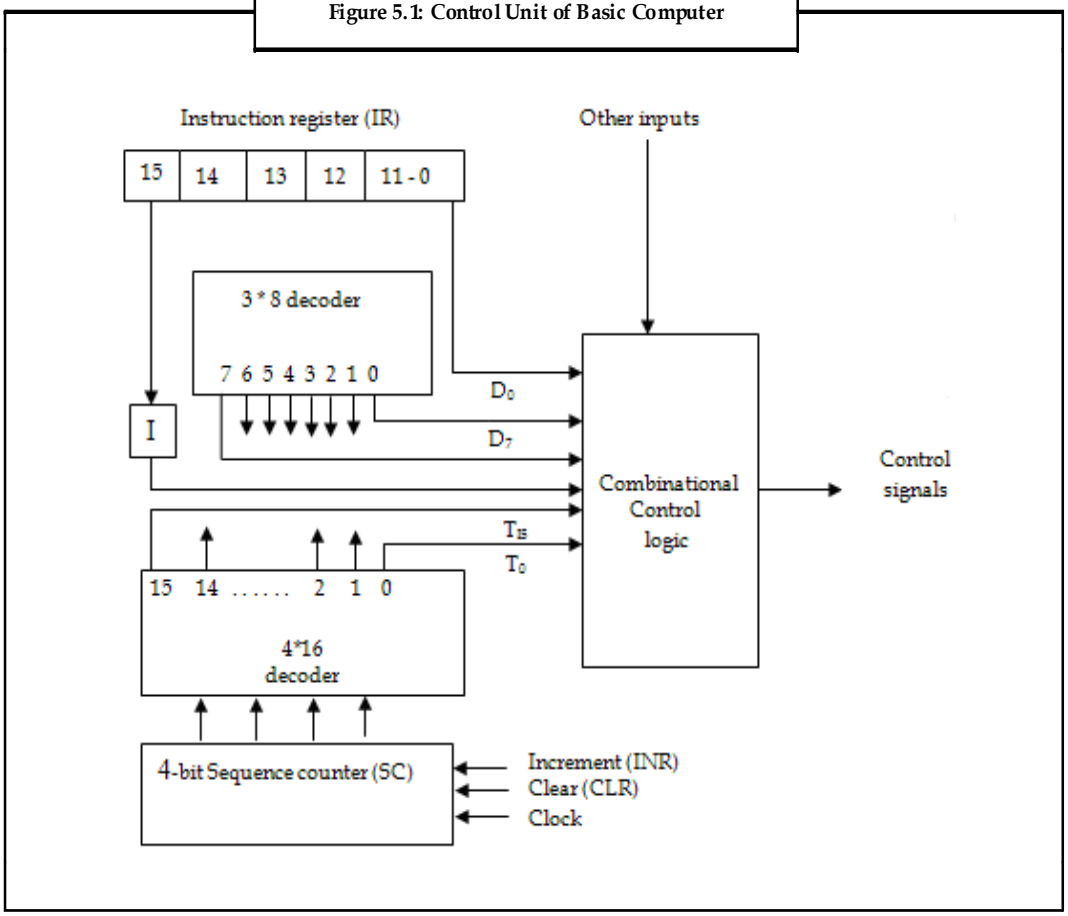
generator. The signals for control are generated by a control unit. They provide control inputs for multiplexer in common bus (we know that a bus gets inputs from multiplexer), control inputs in processor registers and micro-operations for an accumulator. The clock pulses from the master clock must be implemented to all the flip-flops and registers in the systems control unit. When the register is in the disabled mode, the clock pulse remains unchanged.

The control organization can be divided into two major types, the hardwired control and the micro programmed control. The main advantage of the hardwired control is that, it can be optimized to a result in a fast mode of operation. It is implemented with flip flops, gates, decoders, and other digital circuits. If a change or modification is to be done in a hard wired control, it must be done to the wiring among various components.

On the other hand, the micro programmed control stores its control information in a control memory. To initiate the necessary set of micro operations, the control memory is programmed. The changes and modifications in a micro programmed control can be done by updating the micro program in control memory.

The following section deals with the control unit. As we can see in figure 5.1, the control unit consists of two decoders, a sequence counter, and logic gates.

Figure 5.1: Control Unit of Basic Computer



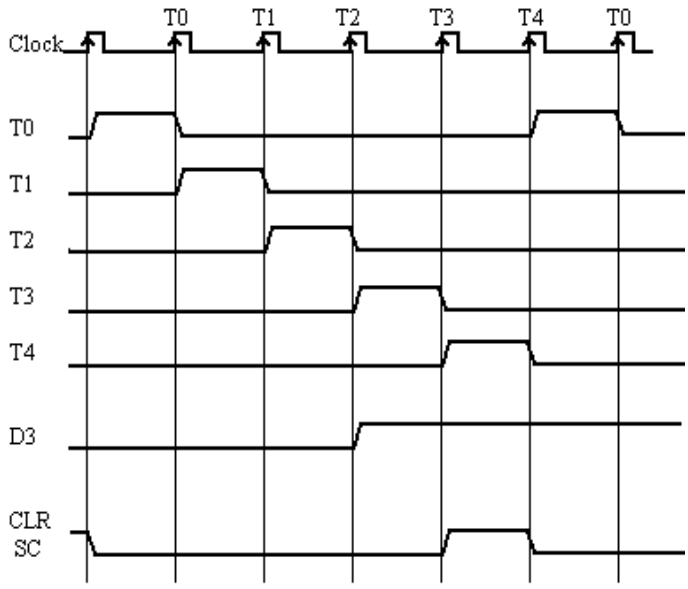
As depicted in figure 5.1, any instruction which is read from the memory is placed in the Instruction Register (IR). Here, the IR is split into three parts, namely, I bit, opcode, and bits from 0 through 11. The opcodes are decoded with a 3 * 8 decoder whose outputs are denoted by symbols D_0 through D_7. The binary value of the respective opcode is the subscripted number in the symbol. The symbol I which is the 15th bit of the instruction is moved to a flip flop. The control logic gates have the bits that are applied from 0 through 11. The sequence counter is 4-bit counts in binary from 0 through 15. It can be incremented or cleared synchronously. The timing signals from T_0 through T_{15} are the decoded outputs of decoder.

Notes

A memory that is read or written is always initiated with the rising timing signal. We are aware that a clock cycle time is greater than a memory cycle time. So when the clock goes through its next positive transition, the memory read/written gets completed. This transition is used to load the memory word to register. But in usual cases, the memory cycle time is longer than the processor cycle. So, it contradicts the timing relation. Therefore, wait cycles are provided till the memory word is made available.

Consider figure 5.2 that shows the time relation between the control signals.

Figure 5.2: Timing Diagram



In figure 5.2, the control signals are generated by the 4-bit sequence counter and a 4 * 16 decoder. In this figure, consider at time T_4, Sequence Counter (SC) is cleared to 0 if the output D_3 of decoder is active. This can be represented as:

D3T4: SC ← 0

The SC gives response to the positive transition of the clock. The CLR input of the SC is active in the beginning.

5.2 Instruction Cycle

The main execution process is done by the processor. The processing of instruction involves two steps, instruction fetch and instruction execution. Each instruction is fetched from the memory separately and executed. Depending on the nature of the instruction its execution may deal with a number of operations.

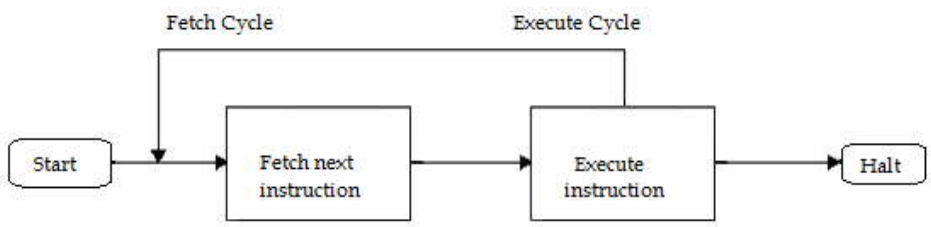
An instruction cycle refers to the processing of a particular instruction. Each instruction cycle goes through the following phases during its processing:

1. Fetching instruction from memory.
2. Decoding the instruction.
3. Reading the effective address from memory in case of indirect address.
4. Executing the instruction.

Notes

After the above four steps are completed, the control switches back to the first step and repeats the same process for the next instruction. Hence, the cycle continues until a HALT condition is met. Figure 5.3 depicts the phases involved in the instruction cycle.

Figure 5.3: Instruction Cycle



As shown in figure 5.3 the halt condition occurs when the machine gets turned off, on occurrence of errors that are unrecoverable, and so on.

Fetch Cycle

The address instruction to be executed is held in the program counter. The processor fetches the instruction from the memory that is pointed by PC. Next, the PC is incremented to show the address of the next instruction. This instruction is loaded on to the instruction register. The processor reads the instruction and implements the necessary actions.

Execute Cycle

The data transfer for execution takes place in two ways:

1. Processor-memory, that is, data transfer from processor to memory or from memory to processor.
2. Processor-Input/Output, that is, data may be transferred to or from a peripheral device by the transfer between a processor and an I/O device.

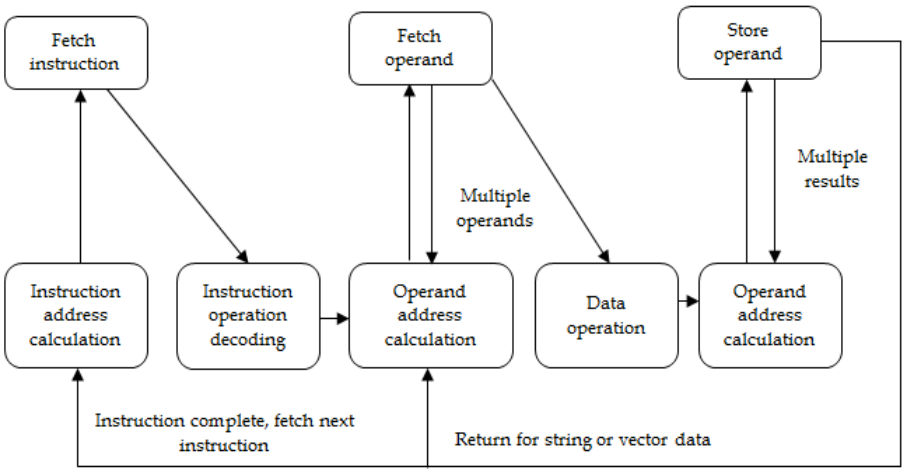
In the execute cycle, the processor performs the necessary operations on the data, and sometimes the control calls for the alteration in the sequence of data execution. These two cases combine and complete the execute cycle.

Notes

State Diagram for Instruction Cycle

The figure 5.4 gives a broader perspective of the instruction cycle of a basic computer, which is in the form of a state diagram. For an instruction cycle, some of the states may be null, while others may be visited more than once.

Figure 5.4: Instruction Cycle State Diagram



Explanation for figure 5.4 is as follows:

1. ***Instruction Address Calculation***: Here, the address of the next instruction is calculated. A fixed number is added to the address of previous instruction.
2. ***Instruction Fetch***: The instruction is read from its respective memory location to the processor.
3. ***Instruction Operation Decoding***: Here, the instruction is analyzed and the type of operation to be performed and the operand(s) to be used are determined.
4. ***Operand Address Calculation:*** The address of operand is calculated if it has a reference to an operand in memory or is available through the Input/Output.
5. ***Operand Fetch***: The operand is read from the memory or the I/O.
6. ***Data Operation:*** The actual operation that the instruction involves is performed.
7. ***Store Operands:*** Stores the result obtained in the memory or sends it to the I/O.

5.3 Memory Reference Instructions

The function that the microoperation intends to perform must be specified accurately. It can be explained in a better way if we consider the register transfer notation. The memory instructions were mentioned in the previous sections.

Now, let us study them in detail.

AND

The AND instruction performs the AND logic operation on the bit pairs from the register and the memory word that is specified by the effective address. The result of this operation is transferred back to the register.

Notes

Example:

DR ê M [AR]

AC ê AC^DR, SC 0

The AND operation is performed on AC and DR and transferred to AC.

ADD

The ADD instruction adds the content of the memory word that is indicated by the effective address to the value of the register. The output calculated is transferred to register and if there is a carry left it is sent to a flip flop.

Example: DR ê M [AR]

AC ê AC+DR, E ê Cout, SC ê 0

Here, content of memory is transferred into DR, the sum is sent to AC. The output carry Cout is transferred to Flip-flop E.

LDA

The LDA instruction transfers the memory word indicated by the effective address to the register.

Example: DR ê M [AR]

AC ê DR, AC ê 0

The memory word is read to **DR** and this content is loaded to **AC.** It is not directly loaded to **AC** due to delay reasons in the adder/logic circuit.

STA

STA stores the content of the register into the memory word that is specified by the effective address. The output is next applied to the common bus and the data input is connected to the bus. It requires only one micro operation.

Example: M [AR] ê AC, SC ê 0

Here, content of **AC** is stored in memory word.

BUN

The **Branch Unconditionally (BUN)** instruction is used to transfer the instruction that is specified by the effective address. We know that the address of next instruction to be executed is held by the PC and it must be incremented by one to get the address of the next instruction in the sequence. If the control wishes to execute some other instruction which is not next in the sequence, it implements the BUN instruction. Hence, we say that the program jumps unconditionally.

Example: PC ê AR, SC ê 0

Here, effective address from **AR** is sent to **PC. SC** is reset to **0.** The new value of **PC** is taken as the next instruction's address.

BSA

The **Branch and Save return Address (BSA)** instruction is used to branch a part of the program (called as subroutine or procedure). When this instruction is executed, BSA will save the address of the next instruction from PC into a memory location that is specified by the effective address.

Example: M [AR] ê PC, PC ê AR + 1

PC gets address for the first instruction in subroutine as effective address plus one.

ISZ

The I**ncrement if Zero (ISZ)** instruction increments the word specified by effective address. If the incremented value is zero, then PC is incremented by 1. A negative value is stored in the memory word by the programmer. At one point of time, it reaches the zero value after getting incremented again and again. At that moment, the PC is incremented and the next instruction is skipped.

Example: DR ê M [AR]

DR ê DR + 1

M [AR] ← DR, if (DR=0) then (PC ê PC+1), SC ê 0

Here, a word is read from memory to **DR,** next **DR** is incremented by **1** and the word is saved in **DR.** When **DR** reaches **0, PC** is incremented and instruction is skipped.

The execution of these microoperations depends on the opcode values. The longest instruction among them is the **INZ.** A 3-bit sequence counter is adequate. However, usually a 4 bit sequence counter is implemented while designing a computer, to provide with additional timing signals for other instructions to get executed.

5.4 Input/Output

The instructions and data that have to be computed must be entered into the computer through some medium. Similarly, the results must be given to user through a medium. The Input/Output module of the computer provides a means to interact with the outside world and equip the operating system with the information it needs to manage the I/O activity effectively. Figure 5.5 is an illustration of input and output device. The input device here is the keyboard and the output device is the printer.

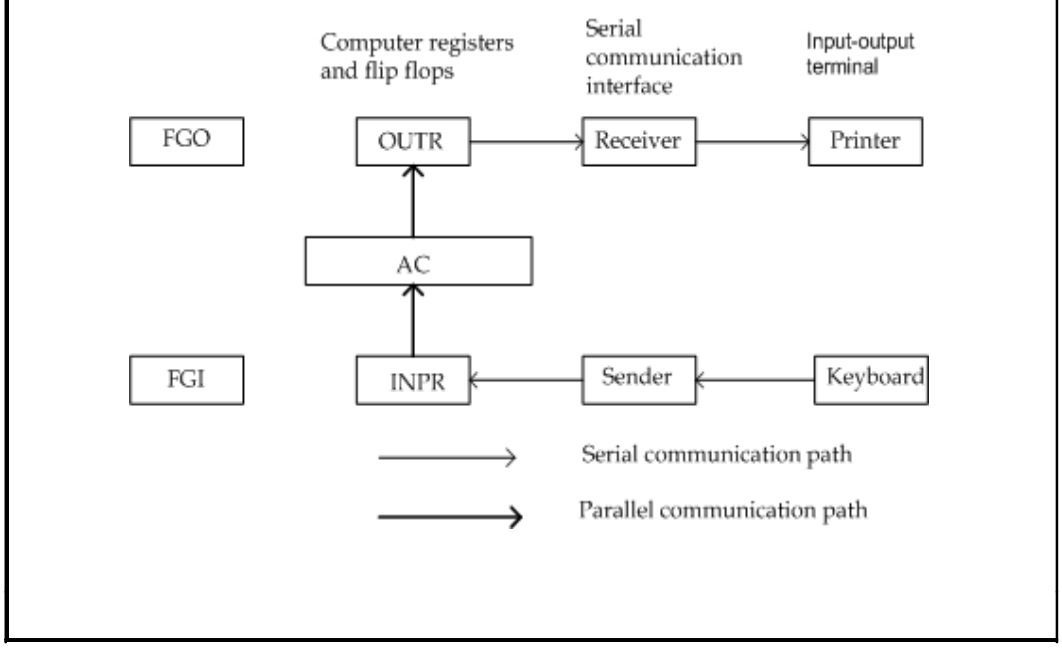
5.4.1 Input/Output Configuration

Consider the figure 5.5 that illustrates the Input/Output configuration. The terminals here are the keyboard and printer. They send and receive the information serially. The information is alphanumeric and 8-bits in size. The input provided through the keyboard is sent to the input register **INPR.** The data is stored in the **OUTPR (output register)** in the serial order for the printer. The OUTR stores the serial information for the printer.

Notes

Figure 5.5 shows input/output configuration.

Figure 5.5: Input/ Output Configuration



As depicted in figure 5.5, the I/O registers interact serially with interfaces (keyboard, printer) and parallels with **AC.** The sender interface receives information from keyboard and sends it to **INPR.** The receiver interface obtains the information and transmits it to printer. The **INPR** holds the **8-bit** alphanumeric input information. **FGI** refers to 1-bit input flag which is a flip-flop. When the input device receives any new data, the flip flop is set to 1. It is cleared to **0** when data is accepted by output device. The difference between the timing rates of input devices and computer is synchronized by the flag. The case is similar with the output device; the difference being the change in direction of the data flow. The output device sets the **FGO** to **1** after accepting, decoding and printing the data. **FGO** in the **0** mode indicates that the device is printing data.

5.4.2 Input/Output Instruction

The I/O devices are given unique addresses. The processor views the I/O operations in a similar manner as memory operations. It issues commands that contain the address for the device. Basically, the I/O instructions are required for the following purposes:

1. Checking flag bits.
2. Transferring data to or from AC register.
3. Controlling interrupts.

The I/O instructions have the opcode as **1111.** They are identified by the control when $D_7 = 1$ and **I=1.** The operation to be performed is specified by the other remaining bits.

Notes

The different I/O instructions are listed in table 5.1.

Table 5.1: I/O Instructions with their Descriptions

Symbol	Description
INP	The INP instruction transmits the data from the INPR to AC which has 8 low order bits. It also clears input flag to 0.
OUT	It transfers the 8 low order bits from AC into output register OUTPR. It also clears the output flag to 0.
SKI	These are the status flags. They skip the next instructions when flag = 1, They are mainly branch instructions.
SKO	
ION	Enables (set) interrupt.
IOF	Disables (clear) interrupt.

5.5 Design of a Basic Computer

The essential components for designing a basic computer are as follows:

1. Registers including AR, PC, AC, DR, IR, SC, TR, INPR, and OUTR.
2. Memory unit that has the capacity to hold 4096 words of 16 bits each. It is the standard component that is available easily.
3. Flip-flops including I, S, E, R, IEN, FGI, and FGO. They can be the D or JK type, which was discussed in the previous units.
4. Decoders (3 * 8 operation decoder and a 4 * 16 timing decoder).
5. Common bus (16-bit), it can be developed by combining sixteen 8 * 1 multiplexers.
6. Adder or logic circuit connected to AC input.

The memory unit and the decoders are components that can be obtained from any commercial resource. A basic computer uses 9 registers generally. The flip flops belonging to two categories, either the D or the JF type can be implemented. The 8*1 multiplexers are used to create the common bus system.

Let us next discuss the design of the control unit.

5.5.1 Control of Logic Gates

The inputs for control logic circuit are:

1. Two decoders.
2. I flip-flop.
3. 0 - 11 bits of instruction register.
4. 0-15 bits from AC to check if A=0 and to determine the sign bit.
5. 0-15 bits from DR to check if DR=0 and the values of flip flops.

Notes

The outputs for the control logic circuit are:

1. Signals controlling register inputs.
2. Signals controlling the read/write inputs of memory.
3. Signals setting, clearing, and complementing flip flops.
4. Signals (S_2, S_1, S_0) selecting register for the bus.
5. Signals controlling the AC adder/logic circuit.

The specification of each has already been discussed previously. (Refer figure 5.1 of control unit)

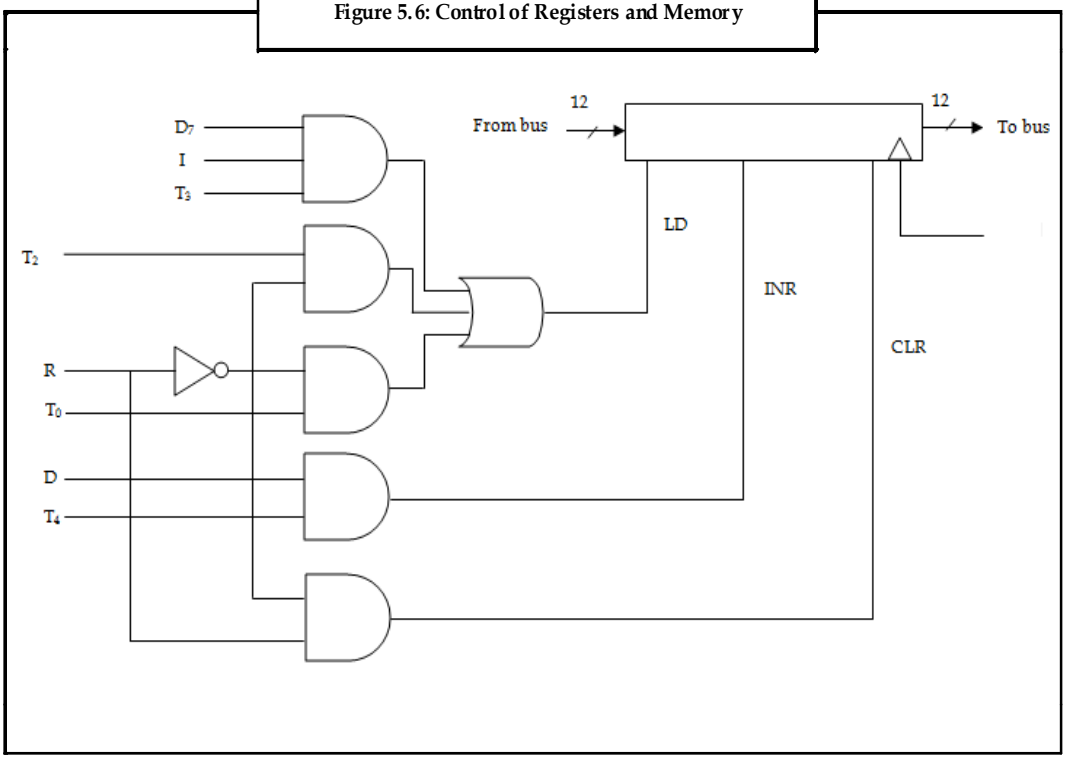
5.5.2 Control of Registers and Memory

The control inputs to registers and memory are:

1. ***Load (LD):*** When it is enabled, the content from the source register or memory can be transferred to/from bus.
2. ***Increment (INR):*** It is used to increment the register by 1.
3. ***Clear (CLR):*** It is used to clear the register.

The figure 5.6 depicts a scenario for the control of registers and memory.

Figure 5.6: Control of Registers and Memory



As depicted in figure 5.6, all the register transfer statements are scanned initially that change the content of AR.

The instructions are as follows:

$R'T_0$: AR ê PC LD (AR)

Here, R is negated, added to T_0, sent to a NAND gate, and loaded to AR.

$R'T_2$: AR ê IR (0-11) LD (AR)

When the clock is incremented, the negated R is added to T_2, NANDed and loaded to AR.

Notes

D'_7IT_3: AR ê M [AR] LD (AR)

RT_0: AR ê 0 CLR (AR)

D_5T_4: AR ê AR + 1 INR (AR)

The first three statements depict the information transfer from register or memory to the AR. AR is cleared to 0 in the fourth statement. AR is incremented in the last statement.

In total the inputs given to address register will be:

LD (AR) $=R'T_0+R'T_2+D'_7IT_3$

CLR (AR) $=RT_0$

INR (AR) $=D_5T_4$

5.5.3 Control of Single Flip-Flop

The control gates associated with flip–flops are determined in the same manner as register and memory. In some cases, there is a possibility that the IEN changes because of the instructions ION and IOF.

Consider the following scenario:

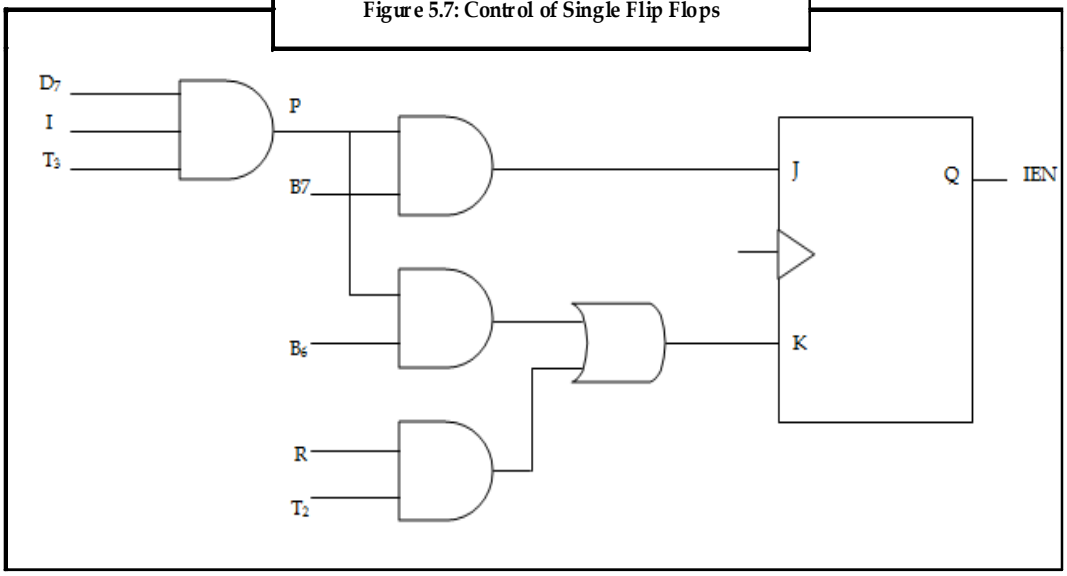
PB_7: IEN ê 1 (I/O instruction)

PB_6: IEN ê 0 (I/O instruction)

RT_2: IEN ê 0 (Interrupt)

Where P = D_7IT_3 (I/O instruction)

Figure 5.7: Control of Single Flip Flops



As depicted in figure 5.7, IEN refers to interrupt enable flag.

For the first two inputs, the value is considered as Input/Output instruction. The third input is an interrupt.

5.5.4 Control of Common Bus

A bus is a structure that handles the data transmission in a computer system or network. The common bus of 16-bit is controlled by the selection inputs S_2, S_1, and S_0. The decimal number shown with each bus input indicates the binary equivalent that must be applied to the selection inputs. This is helpful while selecting the appropriate register.

Notes

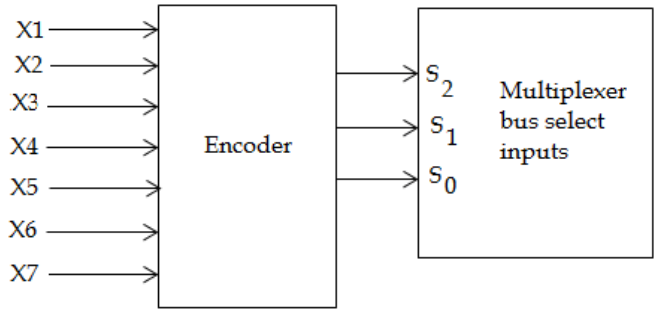
Table 5.2 shows the register that is to be chosen for the particular binary numbers of selection inputs.

Table 5.2: Selection Inputs and their Corresponding Registers

Inputs							Outputs			Register to be selected for bus
X1	X2	X3	X4	X5	X6	X7	S2	S1	S0	
0	0	0	0	0	0	0	0	0	0	None
1	1	1	1	1	1	1	0	0	0	AR
0	0	0	0	0	0	0	0	1	0	PC
0	0	0	0	0	0	0	0	1	0	DR
0	0	0	0	0	0	0	1	0	1	AC
0	0	0	0	0	0	0	1	0	1	IR
0	0	0	0	0	1	0	1	1	0	TR
0	0	0	0	0	0	1	1	1	1	Memory

Each of the binary number is related with a Boolean variable x_1 through x_7 in figure 5.8.

Figure 5.8: Encoder for Bus Selection Structure



As depicted in figure 5.8, the encoder Boolean functions are:

$S_0 = x_1 + x_3 + x_5 + x_7$

$S_1 = x_2 + x3 + x_6 + x_7$

$S_2 = x_4 + x_5 + x_6 + x_7$

The logic for each encoder input can be determined if control functions that place the corresponding register onto the bus, are found.

5.6 Summary

- The timing and control functions are used to execute the instruction codes and registers in an efficient manner. The control organization is divided into hardwired control and micro programmed control.
- The instruction cycle involves the fetch and the execute cycle.
- The data transfer in execute cycle happens through processor to memory and processor to I/O device.

Notes

- The codes written in assembly language which have close reference with machine language are known as memory reference instructions.
- The register reference instructions take only 1 clock cycle to execute. The inputs that can be given to a register are CLR, Load, and INR.
- The I/O instructions are mainly used to check flags, transfer data, and control flags.
- The I/O module provides the interface to interact with the external environment.
- The basic components required for designing a computer are registers, flip flops, decoders, a common bus, a memory unit, and an adder/logic circuit.
- Buses are controlled by selection inputs S2, S1, and S0. The decimals shown with bus input represent the binary equivalent that must be applied to selection inputs.

5.7 Keywords

Decoder: A device that reverses the result of an encoder to retrieve the original information.

Flag: One or more bits that stores a binary value or code that has an assigned meaning.

Operand: Objects in an expression that are manipulated.

Sequence Counter: A counter that produces a series of code combinations separately.

5.8 Self Assessment

1. State whether the following statements are true or false:
   (a) Signals provide control inputs for multiplexer in common bus.
   (b) The INZ instruction increments the word specified by the target address.
   (c) The execution of micro operations depends on the counter values.
2. Fill in the blanks:
   (a) STA stores the content of the _____ into the memory word that is specified by the effective address.
   (b) FGO in the _____ mode indicates that the device is printing data.
   (c) The instruction being read from its respective memory location to the processor is called _____.
3. Select a suitable choice for every question:
   (a) A bus is a structure that handles the _____ in a computer system or network.
      (i) Communication
      (ii) Data transmission
      (iii) Registers
      (iv) Memory unit
   (b) The control gates associated with _____ are determined in the same manner as register and memory.
      (i) Flip-flops
      (ii) Multiplexer
      (iii) Decoder
      (iv) Sequence control

Notes

5.9 Review Questions

1. "The assembly language written by programmers is known as memory reference instructions." Elaborate.
2. "The I/O module provides an interface to the outside world to interact with the computer." Explain.
3. "Each binary code of register has an instruction." Describe the same.
4. "The decimal number shown with each bus input is helpful in selecting the relevant register." Discuss.
5. Logic gates have a number of inputs that are manipulated to provide outputs. Explain the various inputs and their results.
6. "Various components are required while designing a basic computer." List the same.
7. With the help of an example explain the input output configuration of a basic computer.
8. "Sometimes the address bits of an instruction code are used as operand and not as an address." Justify.

Answers: Self Assessment

1.	(a) True	(b) True	(c) False
2.	(a) Register	(b) Zero	(c) Instruction fetch
3.	(a) Data transmission	(b) Flip-flops	

5.10 Further Readings

Books

Radhakrishnan, T., & Rajaraman, V. (2007). Computer Organization and Architecture. New Delhi: Raj Kamal Electric Press.

Rauss, R. (1998). Essentials of computer science 2. USA: Research and education association

Online links

http://www.scribd.com/doc/19731285/Computer-organisation

http://cnx.org/content/m29708/latest/

Notes

Unit 6: Control Unit

CONTENTS

Objectives
Introduction
6.1 Control Memory
6.2 Hardwired Control and Micro Programmed Control Unit
6.2.1 Micro Programmed Control
6.2.2 Difference between Hardwired Control and Micro Programmed Control
6.3 Address Sequencing
6.3.1 Conditional Branching
6.3.2 Instruction Mapping
6.3.3 Subroutines
6.4 Micro Program Sequencing
6.4.1 Micro Instruction Format
6.4.2 Symbolic Micro Instructions
6.5 Summary
6.6 Keywords
6.7 Self Assessment
6.8 Review Questions
6.9 Further Readings

Objectives

After studying this unit, you will be able to:

- Explain control memory
- Discuss hardwired control and micro programmed control unit
- Describe address sequencing
- Elaborate on micro-program sequencing

Introduction

A control unit drives the corresponding processing hardware by generating a set of signals that are in sync with the master clock. The two major operations performed by control unit are instruction interpretation and instruction sequencing.

Control unit is a part of Central Processing Unit (CPU). The CPU is divided into arithmetic logic unit and the Control unit. The control unit generates the appropriate timing and control signals to all the operations involved with a computer. The flow of data between the processor, memory, and other peripherals are controlled using timing signals of the control unit.

The main function of a control unit is to fetch the data from the main memory, determine the devices and the operations involved with it, and produce control signals to execute the operations.

Notes

The functions of control unit are as follows:

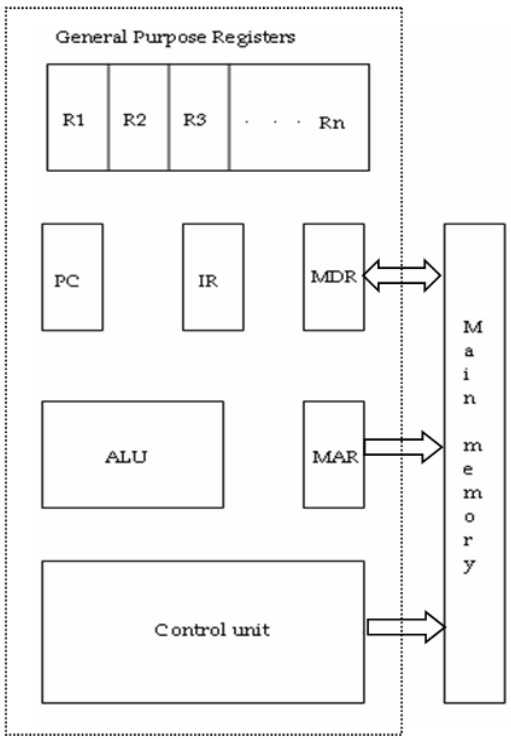
1. It helps the computer system on the process of carrying out the stored program instructions.
2. It interacts with both the main memory and arithmetic logic unit.
3. It performs the arithmetic or logical operations.
4. It coordinates with all the activities related to the other units and the peripherals.

Did u know? The first electronic computers were developed in the mid-20th century (1940 – 1945). They were as large as the size of a room, consuming as much power as several hundred modern PCs (Personal Computers) put together.

As discussed in the previous unit, processor contains a number of registers and special function registers for temporary storage purposes, in addition to the arithmetic logic unit and control unit. Program Counters (PC), Instruction Registers (IR), Memory Address Registers (MAR) and Memory Data Register (MDR) are special function registers. Figure 6.1 depicts these special function registers.

PC is one of the main registers in the CPU. The instructions in a program must be executed in the right order to obtain the correct results. The sequence of instructions to be executed is maintained by the PC.

Figure 6.1: Special Function Registers of the CPU



The IR holds the instruction that is presently being executed. The timing signals generated by the control unit are based on the content of IR. The signals help in controlling the various processing elements that are necessary to execute the instruction.

Notes

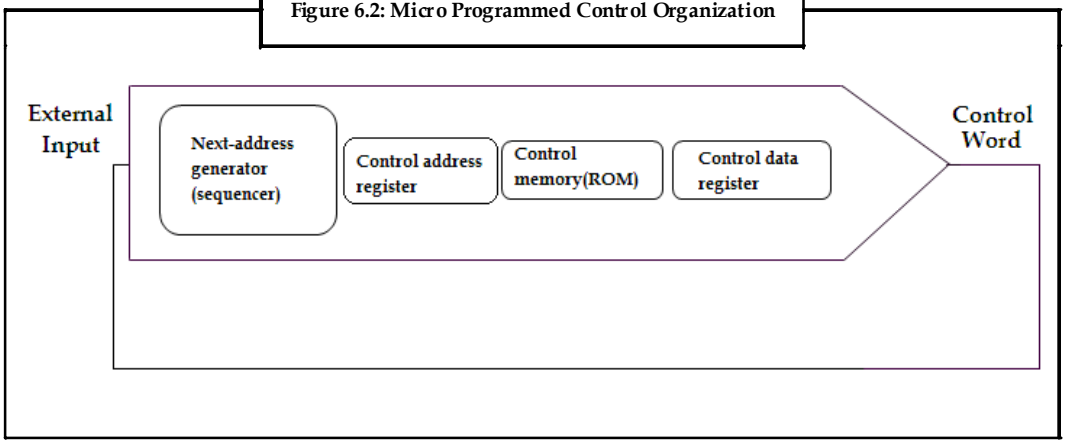
The function of the other registers MAR and MDR is to transfer data. The address of the main memory to/from which data is transferred is stored in MAR. The data that is to be read/written from the specified address to the main memory is stored in MDR.

6.1 Control Memory

A control memory is a part of the control unit. Any computer that involves micro programmed control consists of two memories. They are the main memory and the control memory. Programs are usually stored in the main memory by the users. Whenever the programs change, the data is also modified in the main memory. They consist of machine instructions and data.

On the other hand, the control memory consists of micro programs that are fixed and cannot be modified frequently. They contain micro instructions which specify the internal control signals required to execute register micro operations. The machine instructions generate a chain of micro instructions in control memory. Their function is to generate micro operations that can fetch instructions from main memory, compute the effective address, execute the operation, and return control to fetch phase and continue the cycle. The figure 6.2 represents the general configuration of a micro programmed control organization.

Figure 6.2: Micro Programmed Control Organization



Here, the control is presumed to be a Read Only Memory (ROM), where all the control information is stored permanently. ROM provides the address of the micro instruction. The other register, that is, the control data register stores the micro instruction that is read from the memory. It consists of a control word that holds one or more micro operations for the data processor. The next address must be computed once this operation is completed. It is computed in the next address generator. Then, it is sent to the control address register to be read. The next address generator is also known as the micro program sequencer. Based on the inputs to a sequencer, it determines the address of the next micro instruction. The micro instructions can be specified in number of ways.

The main functions of a micro program sequencer are as follows:

1. Increment the control register by one.
2. Load the address from control memory to control address register.
3. Transfer external address or load an initial address to begin the start operation.

The data register is also known as pipeline register. It allows two operations to be performed at the time. It allows performing the micro operation specified by the control word and also the generation of the next micro instruction. A dual phase clock is required to be applied to the address register and the data register. It is possible to apply a single phase clock to the address register and work without the control data register.

The main advantage of using a micro programmed control is that, if the hardware configuration is established once, no further changes can be done. However, if a different control sequence is to be implemented, a new set of micro instructions for the system must be developed.

Notes

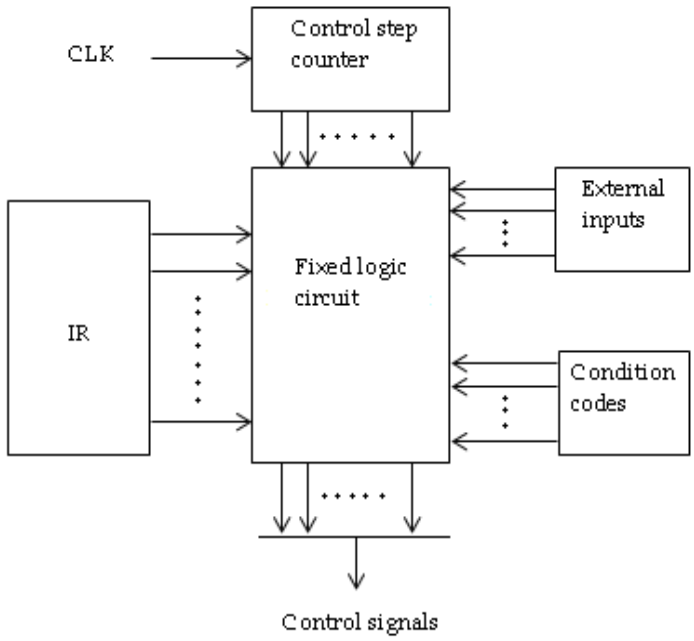
Task Find out the configuration of the latest processors available in the market.

6.2 Hardwired Control and Micro Programmed Control Unit

A hardwired control is a mechanism of producing control signals using Finite State Machines (FSM) appropriately. It is designed as a sequential logic circuit. The final circuit is constructed by physically connecting the components such as gates, flip flops, and drums. Hence, it is named as hardwired controller.

Figure 6.3 depicts a 2-bit sequence counter, which is used to develop control signals. The output obtained from these signals is decoded to generate the required signals in a sequential order.

Figure 6.3: Sequential Counter



The hardwired control consists of combinational circuit that outputs desired controls for decoding and encoding functions. The instruction that is loaded in the IR is decoded by the **instruction decoder.** If the IR is an 8 bit register, then the instruction decoder generates 2^8 (256) lines. Inputs to the encoder are given from the instruction step decoder, external inputs, and condition codes. All these inputs are used and individual control signals are generated. The end signal is generated after all the instructions get executed. Furthermore, it results in the resetting of the control step counter, making it ready to generate control step for the next instruction.

The major goal of implementing the hardwired control is to minimize the cost of the circuit and to achieve greater efficiency in the operation speed. Some of the methods that have come up for designing the hardwired control logic are as follows:

1. ***Sequence Counter Method:*** This is the most convenient method employed to design the controller of moderate complexity.
2. ***Delay Element Method:*** This method is dependent on the use of clocked delay elements for generating sequence of control signals.

Notes

3. ***State Table Method:*** This method involves the traditional algorithmic approach to design the controller using classical state table method.

When the complexity of the control function increases, it is difficult to debug the hardwired controller.

Did u know? The concept of micro programmed control was introduced by Wikes in the year 1051. However, its actual version came into existence in the 1960s.

6.2.1 Micro Programmed Control

A control unit whose binary control values are stored as words in memory is known as a micro programmed control unit.

A controller results in the instructions to be executed by generating a specific set of signals at each system clock beat. Each of these output signals causes one micro operation such as register transfer. Here, the sets of control signals are said to cause specific micro operations that can be stored in the memory. Each bit that forms the micro instruction connects to one control signal. When the bit is set, the control signal is active. When it is cleared the control signal becomes inactive. These micro instructions in a sequence can be stored in the internal 'control' memory. Basically, the control unit of a micro program controlled computer is a computer within a computer.

The steps followed by the micro programmed control are:

1. To execute any instruction, the CPU must break it down into a set of sequential operations (each stating a register transfer level (RTL). These set of operations are known as micro instruction). The sequential micro operations use the control signals to execute. (These are stored in the ROM).
2. Control signals stored in the ROM are accessed to implement the instructions on the data path. These control signals are used to control the micro operations concerned with a micro instruction that is to be executed at any time step.
3. The address of the micro instruction that is to be executed next is generated.
4. The previous 2 steps are repeated until all the micro instructions related to the instruction in the set are executed.

The address that is provided to the control ROM originates from micro counter register. The micro counter gets its inputs from a multiplexer that selects the output of an address ROM, a current address incrementer, and address that is stored in the next-address field of current micro instruction.

The advantages of micro programmed control are:

1. More systematic design of control unit.
2. Easier to debug and modify.
3. Retains the underlying structure of control function.
4. Makes the design of control unit much simpler. Therefore, it is cheaper and less error prone.
5. Orderly and systematic design process.
6. Control function implemented in software and not hardware.
7. More flexible.
8. Complex function is carried out easily.

The disadvantages of micro programmed control are:

1. Flexibility is achieved at extra cost.
2. It is slower than hardwired control unit.

In a micro programmed control, the control memory is assumed to be ROM, where all the data is stored permanently. The memory address of control unit denotes the address of micro instruction.

Notes

The micro instruction has a control word. The control word denotes the operations for the data processor. After the completion of these operations the next address must be determined by the control. The next address may be the one that is next in sequence or the one that is located elsewhere. Due to this reason, it is required that some bits of the present micro instruction are used in the next instruction. Another term for the next address generator is micro program sequencer. The present address is held by the control data register until the next address is computed and read from the memory. The data register is also called pipeline register. A two phase clock is required for the same.

6.2.2 Difference between Hardwired Control and Micro Programmed Control

Table 6.1: Comparison between Hardwired and Micro Programmed Control

Hardwired Control	Micro Programmed Control
It is not possible to modify the architecture and instruction set, once it is built.	It is possible to make modifications by changing the micro program stored in the control memory.
Designing of computer is complex.	Designing of computer is simplified.
Architecture and instructions set is not specified.	Architecture and instruction set is specified.
It is faster.	It is slower comparatively.
It has a processor to generate signals to be implemented in correct sequence.	It uses the micro sequencer from which instruction bits are decoded and implemented.
It works through the use of drums, flip flops, flip chips, and sequential circuit.	It controls the sub devices such as ALU, registers, buses, instruction registers.

Task Study the applications of both hardwired and micro programmed controls and compare.

Refer to the link- http://www.cs.binghamton.edu/~reckert/hardwire3new.html

6.3 Address Sequencing

Micro instructions are stored in control memory in groups. These groups define routines. Each computer instruction has its own micro program routine that is used to generate micro operations. These micro operations are used to execute instructions. The hardware involved controls the address sequencing of the micro instructions of the same routine. They also branch the micro instructions.

Following are the steps that the control undergoes while executing a computer instruction:

1. When power is turned on, an address is initially loaded into the control address register. (This is the address of the first micro instruction).
2. The control address register is incremented resulting in sequencing the fetch routine.
3. After the fetch routine, the instruction is present in the IR of the computer.
4. Next, the control memory retrieves the effective address of the operand from the routine. (The effective address computation routine is achieved through branch micro instruction. It depends on the status of mode bits of instruction. After its completion, the address is made available in the address register).
5. Then, the mapping process happens from the instruction bits to a control memory address.

Notes

6. Based on the opcodes of instruction the micro instructions of the processor registers are generated. Each of these micro instructions has a separate micro program routine stored. The instruction code bits are transformed into the address where routine is located and is called as the mapping process. A mapping procedure converts the micro instruction into control memory address.
7. Next, subroutines are called and procedures are returned.
8. After the completion of the routine, control address register is incremented to sequence the instruction that is to be executed. They also depend on values of status bits in processor registers. External registers are required by micro programs to store return address that use subroutines. After the instruction is executed, the control returns to the fetch routine. This is done by branching the micro instruction to the first address in the fetch routine.

Figure 6.4: Selection of Address for Control Memory

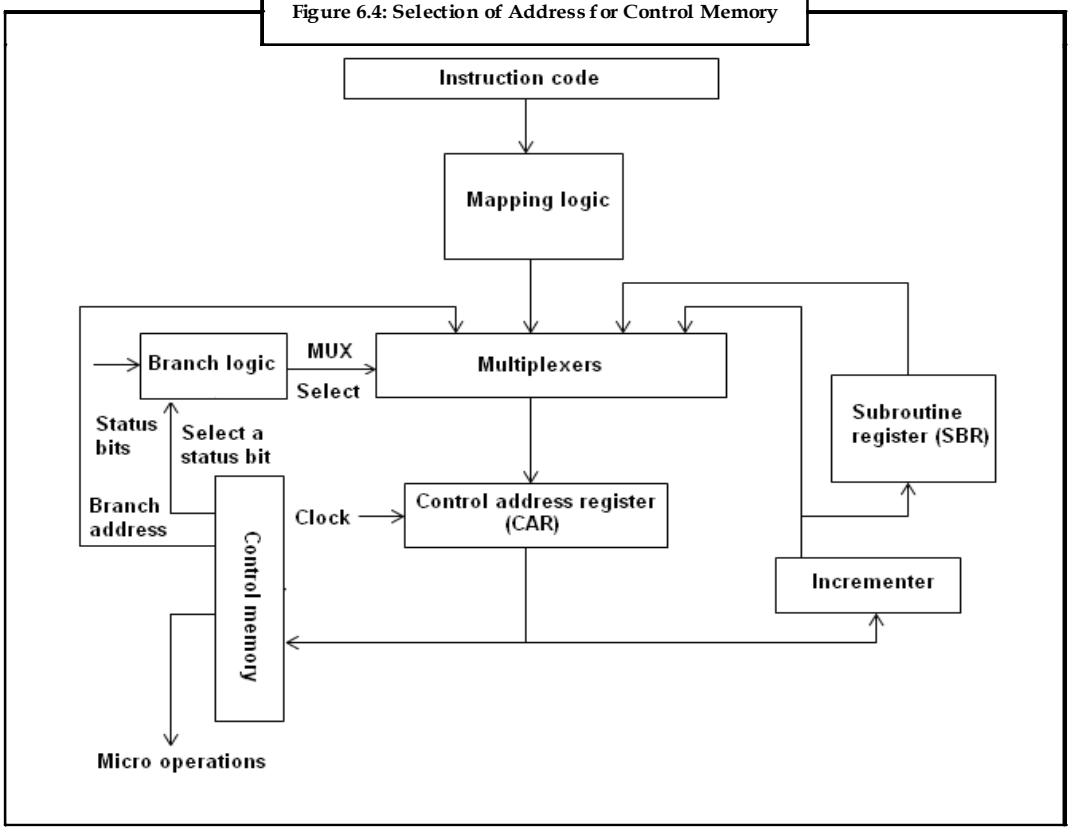


Figure 6.4 depicts the block diagram of a control memory and its related hardware to help in selecting the next micro instruction. The micro instruction present in the control memory has a set of bits that help to initiate the micro operations in registers. They also have bits and the method that can be used to obtain the instruction of the next address. Four different paths are displayed in the figure from where the control address register retrieves its address. The CAR is incremented by the incrementer and selects the next instruction. In one of the fields of the micro instruction, the branching address can be specified to result in branching. To determine the condition of the status bits of micro instruction, conditional branching may be used. A mapping logic circuit is used to transfer an external address. A special register is used to store the return address, so that whenever the micro program wants to return from subroutine, it can use the value from special register.

6.3.1 Conditional Branching

From the figure 6.4, the flow of control is clear. The carry out of an adder, mode bits of an instruction, Input/Output status conditions are the special bits of status conditions. Based on conditions whether their value is 0 or 1, their information is tested and actions are initiated. These bits combine with

Notes

the other fields of the micro instructions and control the decisions regarding the conditions in the branch logic.

The hardware associated with branch logic can be employed in a number of ways. One of the common ways is to test the condition, and if it is satisfied, then branch to the specified address. Else, increment the address register. A multiplexer can be employed to work through the branch logic.

Suppose the number of status bit conditions = 8. Out of the eight bits, three are used to specify selection variables for multiplexer. If selected status bit = 1, multiplexer output = 1, else it would be 0. When the MUX O/P = 1, it produces a control signal and transfers the branch address to CAR from micro instruction. When MOX O/P = 0, the address register gets incremented.

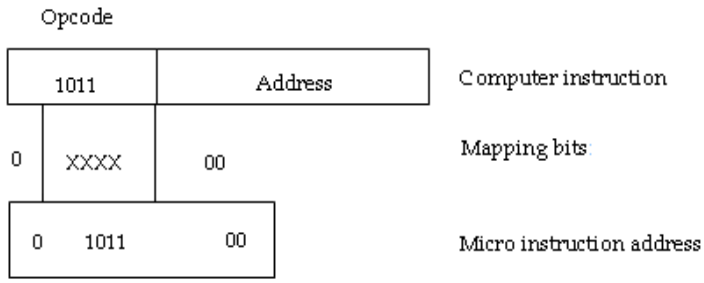
Unconditional Branch Micro Instruction

The unconditional branch micro instruction can be achieved by transferring the branch address from control memory to CAR. Here, the status bit at the input of MUX is fixed to 1. When there is a reference to these status bit lines, the branch address is loaded into CAR causing the unconditional branching.

6.3.2 Instruction Mapping

When a micro instruction specifies the branch to the first word in the control memory where the routine for micro instruction is placed, it leads to a special type of branch. The branch has its status bits placed in the instruction's opcode part.

Figure 6.5: Instruction Code to Micro Instruction Address Mapping



As depicted in figure 6.5 the instruction format of a simple computer has an opcode of four bits. They can specify up to 16 different instructions, if the control memory has 128 words that require 7-bit address. Each of the operations has a micro program routine that helps in executing the instruction. A mapping process can transform a 4-bit opcode to a 7-bit address for control memory. In this process, a 0 is placed in the most significant bit of the address, 4 opcode bits are transferred and 2 least significant bits are cleared. This way each computer instruction has a micro program routine that has a capacity to group 4 micro instructions.

Sometimes, a mapping function needs to use the integrated circuit called Programmable Logic Device (PLD). It uses the AND/OR gates that consist of electrical fuses internally. They are commonly implemented in mapping function that is expressed in terms of Boolean expressions.

6.3.3 Subroutines

Certain tasks cannot be performed by the program alone. They need additional routines known as subroutines. Subroutines are routines that can be called within the main body of the program at any point of time. There are cases when many programs contain identical code sections. These code sections can be saved in subroutines and used wherever common codes are used.

Notes

Example: The code needed to generate effective address of operand for a sequence of micro instruction is common for all the memory reference instructions. This code can be a subroutine and called from within other routines.

All the micro programs that implement subroutine must have extra memory space to store the return address. The extra space is called the subroutine register. It is used to store the return address during a subroutine call and restore during subroutine return. The incremented output must be placed in a subroutine register from a CAR. This must be branched to the beginning of subroutine. Now, the register becomes a means of transferring address to return to main routine. The registers must be arranged in the LIFO (Last In First Out) stack so that it is easy to get the addresses.

6.4 Micro Program Sequencing

The micro code for the control memory must be generated by the designer once the configuration of a computer is established. The generation of code is called **micro programming.**

The important points to be considered while designing the micro program sequencer are:

1. Size of micro instruction
2. Time of address generation

The micro instruction's size must be in the least, so that the control memory required is less and the cost is reduced. Micro instructions can be executed at a faster rate if the time to generate an address is less. This results in increased throughput.

The disadvantage of micro program sequencing is as follows:

1. If each machine instruction has a separate micro routine, then it results in the usage of larger areas for storage.
2. The branching requires more time for execution.

Example: Consider an instruction Add X, AR

This instruction will require four addressing modes - register, auto increment, auto decrement, and indexing in case of indirect forms.

Computer Configuration

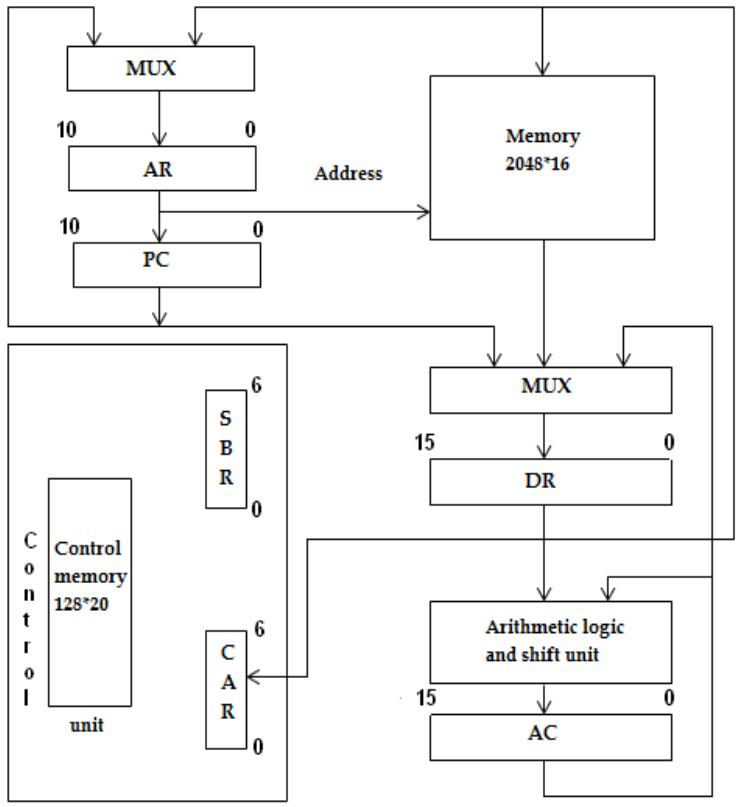
The micro code for the control memory is generated after the computer configuration and its micro programmed control unit is established. The figure 6.6 displays the simple digital computer and the way it is micro programmed. There are two memory units, the instructions and data is stored in the main memory and the micro programs are stored in the control memory. The processor unit consists of four registers, Program Counter (PC), Address Register (AR), Data Register (DR), and an Accumulator (AC). The control unit contains two registers. They are Control Address Register (CAR) and a Subroutine Register (SBR).

Did u know? Referring to a person who carried out calculations, or computations, the first use of the word "computer" was recorded in the year 1613. This word continued with the same meaning until the middle of the 20th century.

Notes

Figure 6.6 shows configuration of computer hardware.

Figure 6.6: Configuration of Computer Hardware



As depicted in figure 6.6, multiplexers are used to transfer information within the registers in the processor. **AR** can get data from **PC** or **DR**. **DR** can receive data from **AC, PC,** or **memory. PC** gets data only from **PC.** The data from **AC** and **DR** can undergo arithmetic and logic operations and be placed in the **AC.** The **DR** is the source of data for memory, where the data that is read can go to **DR** and no other register.

Notes

6.4.1 Micro Instruction Format

A micro instruction format consists 20 bits in total. They are divided into four parts as shown in the figure 6.7.

Figure 6.7: Micro Instruction Code Format

3	3	3	2	2	7
F1	F2	F3	CD	BR	AD

In figure 6.7:

F1, F2, F3 are the micro operation fields. They specify micro operations for the computer.

CD is the condition for branching. They select the status bit conditions.

BR is the branch field. It specifies the type of branch.

AD is the address field. It contains the address field whose length is 7 bits.

The micro operations are further divided into three fields of three bits each. These three bits can specify seven different micro operations. In total there are 21 operations as shown in table 6.2.

Table 6.2: Symbols with their Binary Code for Micro Instruction Fields

	Code	Micro Operation	Symbol
F1	000		NOP
	001	AC ? AC + DR	ADD
	010	AC ? 0	LRAC
	01	AC ? AC + 1	INCAC
	100	AC ? DR	DRTAC
	101	AR ? DR (0 - 10)	DRTAR
	1 0	AR ? PC	PCTAR
	111	M[AR] ? DR	WRITE
F2	000	None	NOP
	001	AC ? AC – DR	SUB
	010	AC ? AC ∨ DR	OR
	011	AC ? AC^DR	AND
	100	DR ? M[AR]	READ
	101	DR ? AC	ACTDR
	110	DR ? DR + 1	IN DR
	111	DR (0 - 10) ? PR	PCTDR
F3	000	None	NOP
	001	AC ? AC ⊕ DR	XOR
	010	AC ? AC	COM
	011	AC ? shl AC	SHL
	100	AC ? shr AC	SHR
	101	PC ? PC + 1	INCPC
	110	PC ? AR	ARTPC
	111	Reserved	

As depicted in table 6.2, each micro instruction can have only three micro operations, one from every field. If it uses less than three, it will result in more than one operation using the no operation binary code.

Notes

Example: Consider **F2** and **F3** as two consecutive micro operations specified by micro instructions. It does not describe **F1.**

DR ← M [AR] with F2 = 100

PC ← PC + 1 with F3 = 101

Here, the micro operation will be **000 100 101.** The **F1** field remains **000** as nothing is specified for the same. Also, two or more conflicting micro operation cannot be specified consecutively.

Condition Field

A condition field consists of 2 bits. They are encoded to specify four status bit conditions. As specified in the table, the first condition is always a **1,** with **CD = 0.** The symbol used to denote this condition is **'U'.**

The table 6.3 depicts the different condition fields and their descriptions in a clear manner.

Table 6.3: Condition Field Symbols and Descriptions

	Condition	Symbol	Comments
00	Always =1	U	Unconditional branch
01	DR(15)	I	Indirect address bit
10	AC(15)	S	Sign bit of AC
11	AC=0	Z	Zero value in AC

As depicted in table 6.3, when the condition 00 is combined with **BR** (branch) field, it results in unconditional branch operation. After the execution is read from memory the **indirect bit I** is available from **bit 15** of **DR.** The status of next bit is provided by **AC** sign bit. If all the bits in **AC** are **1,** then it is denoted as **Z** (its binary variable whose value is 1). The symbols **U, I, S** and **Z** are used to denote status bits while writing micro programs.

Branch Field

The BR (branch) field consists of 2 bits. It is used by combining with the AD (address) field. The reason for combining with AD field is to choose the address for the next micro instruction. The table 6.4 explains the different branch fields and their functions.

Table 6.4 : Branch Field Symbols and Descriptions

BR	Symbol	Function
00	JMP	CAR ? AD, if condition = 1
		CAR ? CAR + 1, if condition = 0
01	CALL	CAR ? AD, SBR ? CAR + 1, if condition = 1
		CAR ? CAR + 1, if condition =0
10	RET	CAR ? SBR (return from subroutine)
11	MAP	CAR (2 - 5) ? DR (11 - 4), CAR (0, 1, 6) ? 0

Notes

As depicted in table 6.4, when **BR = 00,** a **JMP** operation is performed and when **BR = 01,** a **subroutine** is called. The only difference between the two instructions is that when the micro instruction is stored, the return address is stored in the **Subroutine Register (SBR).** These two operations are dependent on the **CD** field values. When the status bit condition of **CD** field is specified as **1,** the address that is next in order is transferred to **CAR.** Else, it gets incremented. If the instruction wants to return from the subroutine, its **BR** field is specified as **10.** This results in the transfer of return address from **SBR** to **CAR**. The opcode bits of instruction can be mapped with an address for **CAR** if **BR** field is **11.** They are present in **DR (11 - 14)** after an instruction is read from memory. The last two conditions in the **BR** fields are not dependent on the **CD** and **AD** field values.

6.4.2 Symbolic Micro Instructions

The micro instructions can be specified by symbols. It is translated to its binary format with an assembler. The symbols must be defined for each field in the micro instruction. Furthermore, the users must be allowed to define their own symbolic addresses. Every line in an assembly language defines a symbolic instruction. These instructions are split in five fields, namely, **label, micro operations, CD, BR, and AD** as explained in table 6.5.

Table 6.5: Fields and their Descriptions

Field	Description
Label	It may be empty or specified by a symbolic address. It is ended by a colon (:).
Micro operations	It consists of one or more symbols separated by commas. But each F field consists of only a single symbol.
CD	It has one of the letters U, I ,A, or Z.
BR	It consists of one among the four symbols defined in table 6.4.
AD	It specifies the value for address field of micro instruction in any of the following ways described below: - With address in symbols that appears as label. - With NEXT symbol that point to the next address in a sequence. - With BR field containing a Ret or a MAP symbol, the AD field is left empty and converted to seven zeroes by the assembler.

Fetch Routine

Control unit consists of **128 words,** each containing **20 bits.** The value of every bit must be determined to micro program the control memory. Among the 128 words, the first **64** are reserved for the routines of **16** instructions. The remaining **64** may be used for other purposes. The best starting location for the fetch routine to begin is the **64th address.** The micro instructions necessary for fetch routine are:

AR ê OC

DR ê M [AR], PC ê PC + 1

AR ê DR (0-10), CAR (2 - 5) ê DR (11 - 14), CAR (0, 1, 6) ê 0

From **PC** the address is transferred to **AR** and read from the memory into **DR.** The instruction code remains in **DR** as no instruction register exists. Next, the address is moved to **AR,** and the control to one of the **16 routines.** They take place by mapping the opcode of instruction into **CAR** from **DR.**

Micro instructions that are located in the addresses **64, 65,** and **66** are necessary for the fetch routine. The symbolic language to describe the same is shown below:

ORG 64

FETCH: PCTARUJMPNEXT

Notes

READ, INCPCUJMPNEXT

DRTAR UMAP

Table 6.6 depicts the results of binary translation for the assembly language.

Table 6.6: Binary Translation for Assembly Language

Binary Address	F1	F2	F3	CD	BR	AD
1000000	110	000	000	00	00	1000001
1000001	000	100	101	00	00	1000010
1000010	101	101	000	00	11	0000000

Each micro instruction implements the internal register transfer operation shown by the register transfer representation. The representation in symbols is necessary while writing micro programs in assembly language format. The actual internal content which is stored in the control memory is in binary representation. In normal practice, programs are written in symbolic form initially and later converted into binary using an assembler.

6.5 Summary

- A control unit controls the data flow through the processor and coordinates the activities of the other units that are involved with it.
- The processor consists of many registers within it. One of the main registers is Program Counter (PC). It holds the instruction that is to be executed next in a sequence. The function of the other registers such as MAR and MDR is to transfer data.
- Control memory is a part of control unit. It stores all the micro programs that cannot be modified frequently. They are fixed programs.
- The data register is also known as pipeline register. It allows two operations to be performed at a time. It allows performing the micro operation specified by the control word and also the generation of the next micro instruction.
- The hardwired control uses the finite state machines to generate control signals.
- The hardwired control consists of physical components such as flip flops, drums, and so on. The methods used to design the hardwired control are sequence counter, delay element, and state table method.
- Micro programmed control is another way of generating control signals. They consist of sequence of micro instructions that correspond to a sequence of steps in an instruction execution.
- The next address generator is called a micro program sequencer.
- Each computer instruction has its own micro program routine that is used to generate micro operations. An address sequencer is a circuit used to generate addresses for accessing the memory device.
- In a conditional branching the information is tested and actions are initiated based on conditions, whether their value is 0 or 1.
- In an unconditional branching, the branch address is transferred to CAR from the control memory.
- Instruction mapping function uses the integrated circuit called Programmable Logic Device (PLD).
- Subroutines are the additional routines that are used by programs to perform some tasks.
- The micro programs need to be sequenced in a certain order for their smooth execution. This is called micro program sequencing.

- A micro instruction format has 20 bits, consisting of micro operation fields, condition field, branch filed, and the address field.

6.6 Keywords

Micro Instruction: An elementary instruction that controls the sequencing of instruction execution and the flow of data in a processor. Execution of a machine language instruction requires the execution of a series of micro instructions.

***Micro Program*:** It consists of a sequence of micro instructions corresponding to the sequence of steps in the execution of a given machine instruction.

Micro Programming: It is the method of generating control signals by setting the individual bits in a control word of a step.

6.7 Self Assessment

1. State whether the following statements are true or false:
   (a) The control memory interacts with both the main memory and the arithmetic logic unit.
   (b) ROM provides address of all the micro instructions in a control memory.
   (c) When a bit is cleared the control signal becomes active.
   (d) In a micro instruction format 7 bits are assigned for address field and 3 bits for the condition field.
   (e) The actual internal content which is stored in the control memory is always in binary representation.
2. Fill in the blanks:
   (a) Designing of controls is complex in _____.
   (b) Multiplexers are used to transfer information within the _____ in the processor.
   (c) The best starting location for the fetch routine to begin is the _____ address.
   (d) The unconditional branch micro instruction is achieved by transferring the _____ from control memory to CAR.
   (e) Programs are written in symbolic form initially and later converted into binary using a _____.
3. Select a suitable choice in every question:
   (a) Which of the following properties of the micro instruction must be least so that it uses less control memory space?
      (i) Size (ii) Time taken (iii) Cost (iv) Steps
   (b) The control unit consists of …….. words, each containing 20 bits.
      (i) 120 (ii) 168 (iii) 128 (iv) 150
   (c) Which of the following methods involve the traditional algorithmic approach to design the controller?
      (i) State table method
      (ii) Sequence counter method
      (iii) Finite state machine method
      (iv) Delay element method

Notes

(d) A condition field consists of …….. bits. They are encoded to specify …….. status bit conditions.

(i) 2, 4 (ii) 3, 6 (iii) 1, 2 (iv) 2, 3

(e) What are the F1, F2 and F3 fields called?

(i) Condition fields

(ii) Branch fields

(iii) Address fields

(iv) Micro operations fields

6.8 Review Questions

1. "A control memory is a part of control unit." Explain.
2. "The function of a control unit is to control the flow of data and coordinate with the components involved in the process." Elaborate.
3. A micro programmed control is easier to design a control unit. Do you agree? Justify.
4. "A computer system consists of two memory units, namely, the main memory and the control memory." Discuss.
5. A control has to undergo a number of steps while executing an instruction. List all the steps.
6. The special bits of status conditions involve the carry out of an adder, mode bits of an instruction, Input/Output status conditions. Do you agree that all these conditions are required for the proper execution of the instruction?
7. The instruction format of a simple computer has an opcode of four bits. Explain the instruction mapping.
8. "A program alone cannot perform all the tasks. It needs some additional routines." Explain
9. "Micro programming involves generating micro code for the control memory." Describe.
10. "The format of micro instruction consists of 20 bits in total." Discuss each of the fields.
11. "The branch field is assigned two bits in the micro instruction format." Describe.
12. Micro instructions that are located in the addresses 64, 65, and 66 are necessary for the fetch routine. Explain the fetch routine in detail.

Answers: Self Assessment

1. (a) False (b) True (c) False (d) False (e) True
2. (a) Hardwired control (b) Registers (c) 64th (d) Branch address (e) Assembler
3. (a) Size (b) 168 (c) State table method (d) 2, 4 (e) Micro operations field

6.9 Further Readings

Books

Radhakrishnan, T., & Rajaraman, V. (2007). Computer Organization and Architecture. New Delhi: Raj Kamal Electric Press.

Godse A.P & Godse D.A. (2008). Digital Electronics, 3rd ed. Pune: Technical Publications.

Online links

http://www.ustudy.in/node/632

http://homepage3.nifty.com/alpha-1/computer/Control_E.html

http://www.slideshare.net/RRoshan/co-unit2-by-rakesh-roshan

Notes

Unit 7: Central Processing Unit

CONTENTS

Objectives
Introduction
7.1 An Overview of the CPU
7.2 General Register Organization
7.3 Stack Organization
7.3.1 Register Stack
7.3.2 Memory Stack
7.4 Instruction Format
7.5 Summary
7.6 Keywords
7.7 Self Assessment
7.8 Review Questions
7.9 Further Readings

Objectives

After studying this unit, you will be able to:

- Explain the functions of CPU
- Explain the general register organization
- Describe stack organization
- Discuss instruction format

Introduction

The Central Processing Unit (CPU) is the heart of a computer system. The CPU along with the memory and the I/O sub-systems develops a powerful computer system.

Companies such as AMD, IBM, Intel, Motorola, SGI, and Sun manufacture CPUs that are used in various kinds of computers such as desktops, mainframes, and supercomputers. A CPU comprises thin layers of thousands of transistors. Transistors are microscopic bits of material that block electricity at one voltage (non-conductor) and allow electricity to pass through them at different voltage (conductor). These tiny bits of materials are the semiconductors that take two electrical inputs and generate a different output when one or both inputs are switched on. As CPUs are small, they are also referred to as microprocessors.

Did u know? CPU is an old term that was used for processor or multiprocessor.

Modern CPUs are called as integrated chips. It is so called because several types of components such as execution core, Arithmetic Logic Unit (ALU), registers, instruction memory, cache memory, and the input/output controller are integrated into a single piece of silicon.

Notes

Example: Intel makes Pentium series of processors, whereas AMD makes the Athlon and Duron processors.

7.1 An Overview of the CPU

Central Processing Unit (CPU) is the most important unit in a computer system. It is the component which controls all internal and external devices as well as performs arithmetic and logic operations to execute the set of instructions stored in the computer's memory.

A CPU comprises three major components. They are:

- Register Set
- ALU
- Control Unit (CU)

Register Set

The register set differs from one system to another. The register set comprises many registers which include general purpose registers and special purpose registers. The general purpose registers do not perform any specific function. They store the temporary data that is required by a program. The special purpose registers perform specific functions for the CPU.

Example: Instruction Register (IR) is a special purpose register that stores the instruction that is currently being executed.

ALU

The ALU performs all the arithmetic, logical, and shift operations by providing necessary circuitry that supports these computations.

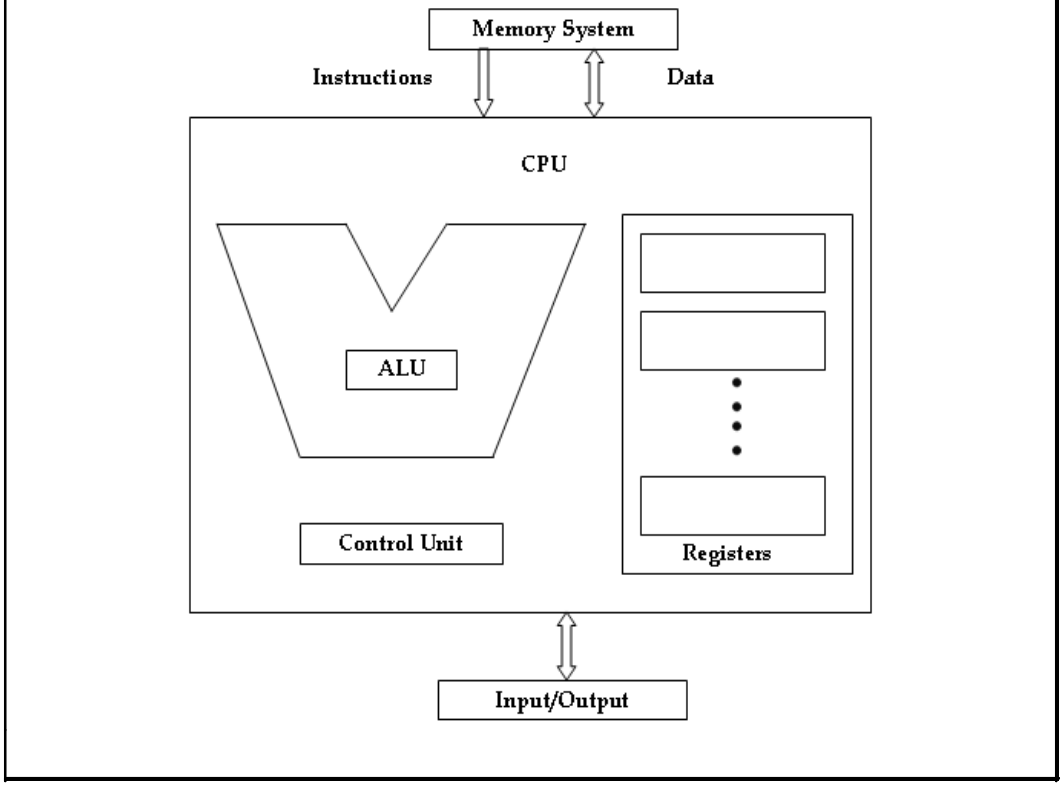
Control Unit

The control unit fetches the instructions from the main memory, decodes the instructions, and then executes it. The control unit is discussed in detail in the units ahead.

Notes

Figure 7.1 illustrates the components of the CPU.

Figure 7.1: Components of CPU



As shown in figure 7.1, the CPU consists of the register set, ALU, and CU.

The CPU interacts with the main memory and input/output devices. The CPU reads and writes data to and from the memory system and transfers data to and from the I/O devices.

A simple execution cycle in the CPU can be described as below:

1. The CPU fetches the instruction to be executed from the main memory and stores it in the Instruction Register (IR).
2. The instruction is decoded.
3. The operands are fetched from the memory system and stored in the CPU registers.
4. The instructions are then executed.
5. The results are transferred from the CPU registers to the memory system.

Notes Operand is the part of a computer instruction that is manipulated and operated. In the addition of 5 + x, '5' and 'x' are operands and '+' is the operator.

If there are more instructions to be executed, the execution cycle repeats. Any pending interrupts are also checked during the execution cycle.

Example: The interrupts such as I/O device request, arithmetic overflow, or pages is checked during the execution cycle.

Notes

The actions of the CPU are defined by the micro-orders issued by the control unit. The micro-orders are the control signals, which are sent over specified control lines.

Example: Suppose you need to execute an instruction that moves the contents of register A to register B. If both the registers are connected to data bus C, then the control unit issues a micro-order (control signal) to register A to place its contents on the data bus C. Another micro-order is sent to register B to read from data bus C. The control signals are activated either through hardwired control or microprogramming.

Thus, CPU is the primary element of a computer system, which carries out each instruction of a program to perform basic arithmetical, logical, and input/output operations.

7.2 General Register Organization

A group of flip-flops form a register. A register is a special high speed storage area in the CPU. They comprise combinational circuits that perform data processing. The data is always represented in a register before processing. The registers speed up the execution of programs.

Registers perform two important functions in the CPU operation. They are:

1. Providing a temporary storage area for data. This helps the currently executing programs to have a quick access to the data, if needed.
2. Storing the status of the CPU as well as information about the currently executing program.

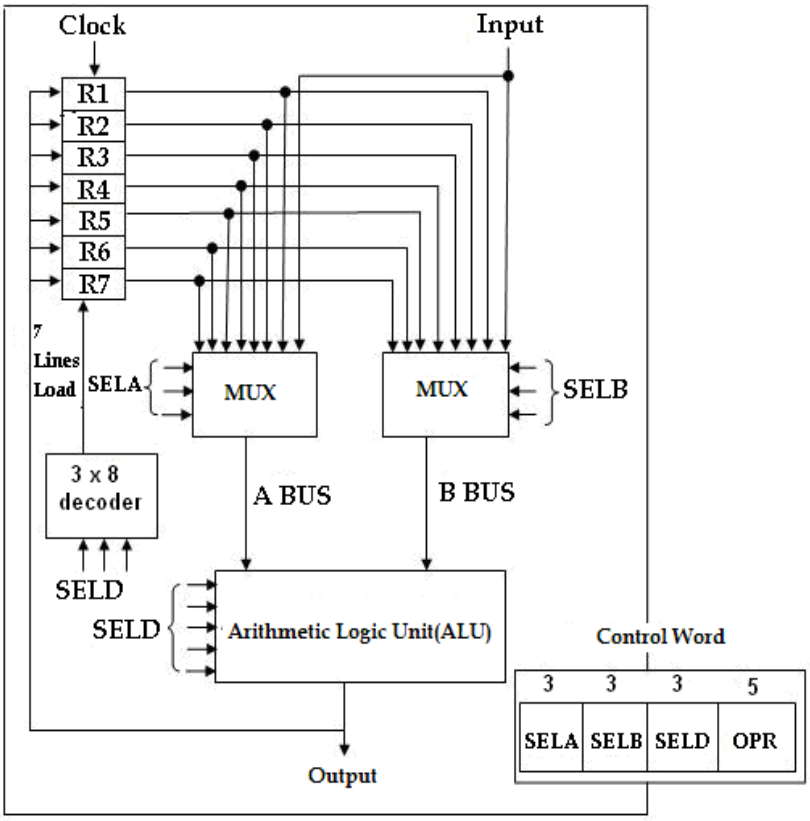
Example: Address of the next program instruction, signals received from the external devices and error messages, and such other information is stored in the registers.

We know that referring to memory locations is considered difficult and time consuming. Hence, storing the pointers, return addresses, temporary results, and program counters into the register is more efficient than the memory. The number of registers varies from one computer system to another.

Notes

If a CPU contains a number of registers, then a common bus is used to connect these registers. A general organization of seven CPU registers is shown in figure 7.2.

Figure 7.2: General Organization of Registers



As observed in figure 7.2, the CPU bus system is operated by the control unit. The control unit directs the information flow through the ALU by selecting the function of the ALU as well as components of the system.

Consider R1 ← R2 + R3, the following are the functions performed within the CPU:

MUX A Selector (SELA): It is used to place R2 into bus A

MUX B Selector (SELB): It is used to place R3 into bus B

ALU Operation Selector (OPR): It selects the arithmetic addition (ADD)

Decoder Destination Selector (SELD): It transfers the result into R1.

The multiplexers of 3-state gates are implemented with the buses. The state of 14 binary selection inputs specifies the control word. The 14-bit control word specifies a micro-operation.

Notes

The encoding of register selection fields are specified in table 7.1.

Table 7.1: Encoding of Register Selection Field

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

Notes 3-state gates are the types of logic gates having three states of output: high (H), low (L) and high-impedance (Z).

Various micro-operations are performed by the ALU. Some of the operations performed by the ALU are listed in table 7.2.

Table 7.2: Encoding of ALU Operations

OPR Select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	Add A + B	ADD
00101	Subtract A - B	SUB
00110	Decrement A	DECA
01000	ADD A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

Some of the ALU micro-operations are shown in the table 7.3:

Table 7.3: ALU Micro-Operations

Micro-operation	SELA	SELB	SELD	OPR	Control Word			
R1 ← R2 – R3	R2	R3	R1	SUB	010	011	001	00101
R4 ← R4 R5	R4	R5	R4	OR	100	101	100	01010
R6 ← R6 + 1	R6	-	R6	INCA	110	000	110	00001
R7 ← R1	R7	-	R1	TSFA	001	000	111	00000
Output ← R2	R2	-	None	TSFA	010	000	000	00000
Output ← Input	Input	-	None	TSFA	000	000	000	00000
R4 ← shl R4	R4	-	R4	SHLA	100	000	100	11000
R5 ← 0	R5	R5	R5	XOR	101	101	101	01100

Notes

Types of Registers

There are many different types of register available in the market. Some of them are:

1. ***Data Register:*** It is used to store data.
2. ***Accumulator Register***: It is considered as a special data register.
3. ***Address Register***: It holds the memory address.
4. ***Index Register:*** It holds the index of the memory address.
5. ***Condition Register***: It determines whether the instruction should be executed or not.
6. ***General Purpose Register***: It stores data and addresses.
7. ***Special Purpose Register:*** It stores the status of the programs.
8. ***Floating Point Register:*** It is a kind of data register that stores the floating point numbers.
9. ***Constant Register:*** It stores read-only values.

General purpose registers are also called as processor registers. These processor registers provide the fastest means to access data.

Did u know? Processor register is mostly found at the top of the memory hierarchy.

7.3 Stack Organization

Stack, also called as Last In First Out (LIFO) list, is the most useful feature in the CPU. It stores information such that the item stored last is retrieved first. Stack is a memory unit with an address register. This register holds the address for the stack, which is called as Stack Pointer (SP). The stack pointer always holds the address of the item that is placed at the top of the stack.

You can insert an item into or delete an item from the stack. The insertion operation is called as push operation and the deletion operation is called as pop operation. In a computer stack, these operations are simulated by incrementing or decrementing the SP register.

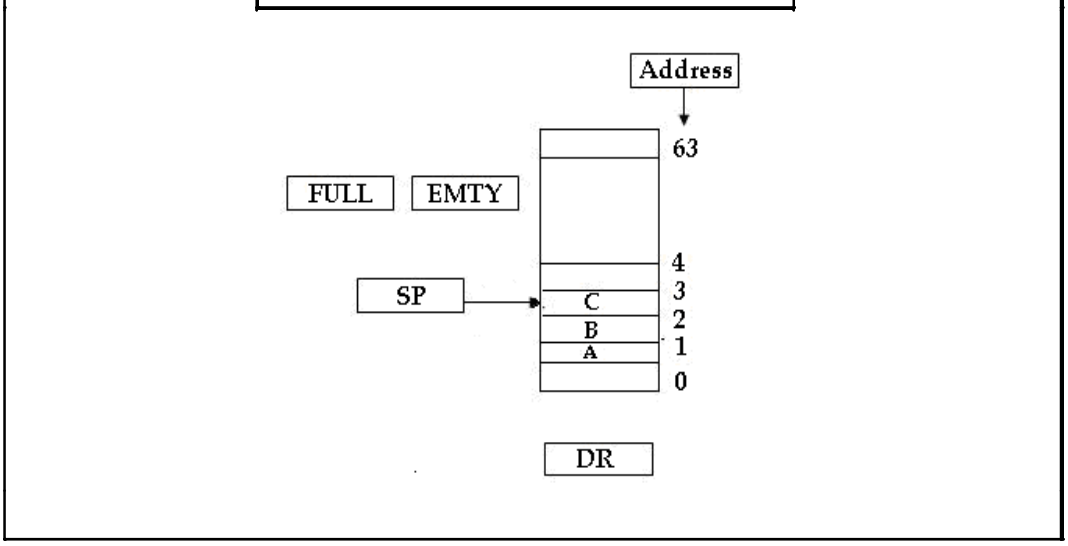
7.3.1 Register Stack

Stack can be organized as a collection of memory words or registers. Consider a 64-word register stack organized as shown in figure 7.3. The stack pointer register contains a binary number, which is the address of the item present at the top of the stack. The three items A, B, and C are placed in the stack. The item C is at the top of the stack and the stack pointer holds the address of C that is, 3. The top item is popped from the stack by reading memory word at address 3 and decrementing the stack pointer by 1. Now, B is at the top of the stack and the SP holds the address of B that is, 2. To insert a new word, the stack is pushed by incrementing the stack pointer by 1 and inserting a word in that incremented location.

Notes

Figure 7.3 depicts 64-word stack.

Figure 7.3: 64-word Stack



Here, the stack pointer contains 6 bits, since $2^6 = 64$, and the SP cannot exceed 63 (111111 in binary) because if 63 is incremented by 1, then the result is 0(111111 + 1= 1000000). SP holds only the six least significant bits. If 000000 is decremented by 1 then the result is 111111. Thus, when the stack is full, the one bit register 'FULL' is set to 1. If the stack is empty, then the one bit register 'EMTY' is set to 1. The data register DR holds the binary data which is written into or read out of the stack.

First the SP is set to 0, EMTY is set to 1, and FULL is set to 0. Now, as the stack is not full (FULL = 0), a new item is inserted using the push operation. The push operation is performed as below:

SP SP + 1, stack pointer is incremented

K[SP] – DR, place an item on the top of the stack

If (SP = 0) then (FULL 1), check if stack is full

EMTY 0, if stack is full, then mark the stack as not empty

The stack pointer is incremented by 1 and the address of the next higher word is stored in the SP. The word from DR is inserted into the stack using the memory write operation. As per figure 5.3, the first item is stored at address 1 and the last item is stored at address 0. If the stack pointer is at 0, then the stack is full and 'FULL' is set to 1. This is the condition when the SP was in location 63 and after incrementing SP , the last item is stored at address 0. Once an item is stored at address 0, there are no more empty registers in the stack. The stack is full and the 'EMTY' is set to 0.

You can perform pop operation (deletion) only if the stack is not empty. To delete an item from the stack, the following micro-operations are performed.

DR K[SP], an item is read from the top of stack

SP SP – 1, stack pointer is decremented

If(SP = 0) then (EMTY 1), check if stack is empty

FULL 0, if stack empty, then mark the stack empty

The top item from the stack is read and sent to DR and then the stack pointer is decremented. If the stack pointer reaches 0, then the stack is empty and 'EMTY' is set to 1. This is the condition when the item in location 1 is read out and the SP is decremented by 1.

Notes

Notes When the pop operation reads an item from location 0, the stack pointer is decremented and reaches 63 (SP changes to 111111).

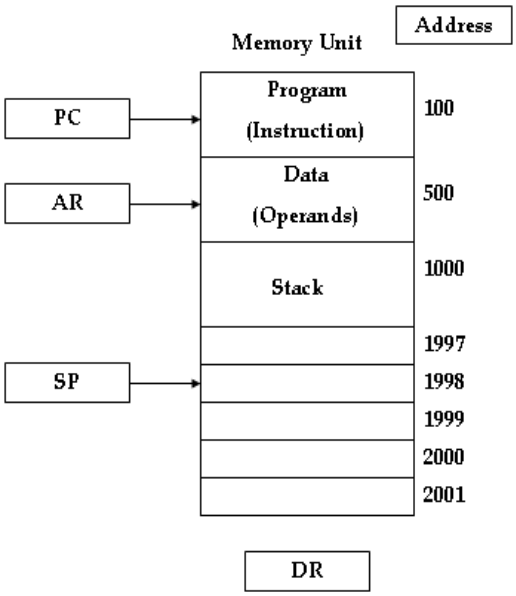
Caution If stack is pushed when FULL = 1 or popped when EMTY = 1, you get an erroneous result.

7.3.2 Memory Stack

Stack can be implemented in the CPU by allotting a portion of the computer memory to a stack operation and using a processor register as a stack pointer. In this case, it is implemented in a random access memory attached to the CPU.

In figure 7.4, a portion of the computer memory is divided into three segments: program, data and stack. The address of the next instruction in the program is stored in the pointer Program Counter (PC). The Address Register (AR) points to an array of the data. SP always holds the address of the item present at the top of the stack. The three registers that are connected to the common bus are PC, AR, and SP. PC is used to read the instruction during fetch phase. An operand is read during execute phase using address register. An item is pushed into or popped from the stack using stack pointer. Figure 7.4 depicts the memory stack.

Figure 7.4: Memory Stack



In figure 7.4, the SP points to an initial value '2001'. Here, the stack grows with decreasing addresses. The first item is stored at address 2000, the next item is stored at address 1999 and the last item is stored at address 1000.

As we already know, data register is used to read an item into or from the stack. You use push operation to insert a new item into the stack.

SP SP -1

K[SP] DR

Notes

To insert another item into the stack, the stack pointer is decremented by 1 so that it points at the address of the next location/word. A word from DR is inserted into the top of the stack using memory write operation.

To delete an item from the stack, you need to use the pop operation:

DR K[SP]

SP SP + 1

The top item is read into the DR and then the stack pointer is decremented to point to the next item in the stack.

Here, two processor registers are used to check the stack limits. One processor register holds the upper limit (1000) and the other holds the lower limit (2001). During push operation, the SP is compared with the upper limit to check if the stack is full. During pop operation, the SP is compared with the lower limit to check if the stack is empty.

An item in the stack is pushed or popped using two micro-operations. They are:

1. Accessing the memory through SP
2. Updating SP

A stack pointer is initially loaded with the bottom address of the stack in memory. Thereafter, SP is automatically incremented or decremented depending on the operation performed (push or pop). As the address is automatically updated in the stack pointer, the CPU can refer to the memory stack without specifying the address.

7.4 Instruction Format

An instruction consists of a combination of operation codes and operands that deal with the operation codes. Instruction format basically provides the layout of bits in an instruction. It includes fields such as opcode, operands, and addressing mode. The instruction length is usually kept in multiples of the character length, which is 8 bits. When the instruction length is fixed, a number of bits are allocated to opcode, operands, and addressing modes. The bits are distributed such that if more number of bits is allocated to the opcode field, then less number of bits are allocated to the operands and addressing. The task of allocating bits in the instruction can be simplified by considering the following factors:

1. Number of addressing modes
2. Number of operands
3. Number of CPU registers
4. Number of register sets
5. Number of address lines

Figure 7.5 shows the general IA-32 (Intel Architecture- 32 bits) instruction format. IA-32 is the instruction format that is used in Intel's most successful microprocessors. This instruction format consists of four fields, namely opcode field, addressing mode field, displacement field, and immediate field.

Figure 7.5: IA-32 Instruction Format

Opcode	Addressing mode	Displacement	Immediate
1 or 2 bytes	1 or 2 bytes	1 or 4 bytes	1 or 4 bytes

Notes

As shown in figure 7.5, the opcode field has 1 or 2 bytes. The addressing mode field also includes 1 or 2 bytes. In the addressing mode field, an instruction needs only one byte if it uses only one register to generate the effective address of an operand. The field that immediately follows the addressing mode field is the displacement field. If an effective address for a memory operand is calculated using the displacement value, then it uses either one or four bytes to encode. If an operand is an immediate value, then it is placed in the immediate field and it occupies either one or four bytes.

Instructions in a computer can be of different lengths with varying number of addresses. The number of address fields in the instruction format of a computer varies according to the organization of its registers. Based on the number of address fields the instruction can be classified as three address instructions, two address instructions, one address instruction, and zero address instruction.

Three Address Instructions

The general format of a three address instruction is represented as:

operation source 1, source 2, destination

ADD A, B, C

where A, B and C are the three variables that are assigned to a distinct location in the memory. 'ADD' is the operation that is performed on the operands. 'A' and 'B' are the source operands and 'C' is the destination operand.

Here, bits are required to specify the three operands. n bit is required to specify one operand (one memory address). Similarly, 3n bits are required to specify three operands (three memory addresses). Bits are also required to specify the ADD operation.

Two Address Instructions

The general format of a two address instruction is represented as:

operation source, destination

ADD A, B

where A and B are the two variables that are assigned to a distinct location in the memory. 'ADD' is the operation that is performed on the operands. This instruction adds the content of the variables A and B and stores the result in variable B. Here, 'A' is the source operand and 'B' is considered as both source and destination operands.

Here, bits are required to specify the two operands. n bit is required to specify one operand (one memory address). Similarly, 2n bits are required to specify two operands (two memory addresses). Bits are also required to specify the ADD operation.

One Address Instruction

The general format of one address instruction is represented as:

operation source

ADD A

where A is the variable that is assigned to a distinct location in the memory. 'ADD' is the operation that is performed on the operand A. This instruction adds the content of the variable A into the accumulator and stores the result in the accumulator by replacing the content of the accumulator.

Some more examples of one address instructions are:

LOAD A: The content of memory location A is stored in the accumulator.

STORE B: The content of accumulator is stored in the memory location B.

The operand in the instruction can either be the source or the destination, depending on the instruction.

Notes

Zero Address Instructions

The locations of the operands in zero address instructions are defined implicitly. These instructions store operands in a structure called pushdown stack.

We now know that one address instruction uses less number of bits, whereas three address instructions are uses more number of bits. Similarly, the three address instructions require more memory access when compared to one address instructions. Thus, three address instructions take more time to execute instructions when compared to one address instructions.

To reduce the execution time of the instructions, it is advised to refer the operands from the processor registers instead of referring the operands from the memory.

Lab Exercise

1. Write an assembly language program to subtract two numbers.
2. Write an assembly language program to find the average of two numbers.

7.5 Summary

- A Central Processing unit (CPU) is the main unit of a computer system.
- There are three main components of CPU, namely register set, ALU, and control unit.
- Registers are the temporary storage, which are constructed from flip-flops. They store the status of the CPU.
- Stack is considered as a memory unit with an address register. It has a stack pointer, which always points at the top item in the stack.
- If stack is organized as a collection of registers, then stack is considered as register stack. If implemented in a random access memory attached to the CPU, then stack is considered as memory stack.
- Instruction format is the layout of bits in an instruction. An instruction includes fields such as opcode, operands, and addressing mode.

7.6 Keywords

Bus: An electrical conductor that connects all internal computer components to the CPU and the main memory.

Displacement Value: A value added to the contents of the address register and the resulting value is used as the address of the operand.

Immediate Field: A field that includes the address offset field for branches, load/store instructions, and jump target fields.

Multiplexer: A device that can combine several input signals into one output.

Notes

7.7 Self Assessment

1. State whether the following statements are true or false:
   (a) The control unit fetches the instructions from the registers, decodes and then executes it.
   (b) The registers store the status of the CPU as well as information about the currently executing program.
   (c) Memory stack is implemented in a random access memory attached to the CPU.
2. Fill in the blanks:
   (a) Pending interrupts are checked during _____.
   (b) The CPU bus system is operated by the _____.
   (c) Stack can be implemented in the CPU by allotting a portion of the computer memory to a stack operation and using a _____ as a stack pointer.
   (d) As instruction consists of a combination of _____and operands that deal with these operation codes.
3. Select a suitable choice for every question:
   (a) The CPU reads and writes data to and from _____and transfers data to and from I/O devices.
      (i) Memory system
      (ii) Register
      (iii) Control unit
      (iv) Arithmetic and logic unit
   (b) Constant registers is used to store _____.
      (i) Floating point numbers
      (ii) Read-only values
      (iii) Addresses
      (iv) Programs
   (c) The push and pop operations are simulated by incrementing or decrementing the _____ register.
      (i) Accumulator
      (ii) Address
      (iii) Stack pointer
      (iv) Index

7.8 Review Questions

1. "Central Processing Unit (CPU) is the most important unit in a computer system." Justify.
2. "A register is a special high speed storage area in the CPU." Justify with diagram.
3. "Stack can be implemented in the CPU by allotting a portion of the computer memory to a stack operation and using a processor register as a stack pointer." Explain.
4. "Instruction format basically provides the layout of bits in an instruction". Discuss.
5. "The bits of the status register are modified according to the operations performed in the ALU." Discuss.

Notes

Answers: Self Assessment

1. (a) False (b) True (c) True
2. (a) Execution cycle (b) Control unit (c) Processor register (d) Operation codes
3. (a) Memory system (b) Read-only values (c) Stack pointer

7.9 Further Readings

Books

Morris M. Computer System Architecture. Pearson Education.

A.P.Godse & D.A.Godse (2010). Computer Organization And Architecture. Pune: Technical Publications.

Stallings. W (2009). Computer Organization and Architecture: Designing for Performance. Prentice Hall.

Online links

www.mans.edu.eg/faceng/english/computers/PDFS/PDF4/1.2.pdf

www.ehow.com/list_7332165_types-addressing-modes-computers.html

Notes

Unit 8: Addressing Modes

CONTENTS

Objectives
Introduction
8.1 Need for Addressing Modes
8.2 Data Transfer and Manipulation
8.3 Program Control
8.4 RISC and CISC
8.4.1 RISC Instruction Set
8.4.2 RISC Versus CISC
8.5 Summary
8.6 Keywords
8.7 Self Assessment
8.8 Review Questions
8.9 Further Readings

Objectives

After studying this unit, you will be able to:

- List the addressing modes
- Explain data transfer and manipulation process
- Discuss program control instructions
- Differentiate between RISC and CISC architecture

Introduction

In almost all CPU designs, addressing modes are a part of the instruction set architecture. Various addressing modes are defined in a given instruction set architecture. These addressing modes describe the procedure by which language instructions in instruction set architecture identify the operands of each instruction. We can specify how to calculate the effective memory address of an operand by using addressing modes. This is done by using information held in registers and/or constants contained within a machine instruction or elsewhere.

RISC and CISC are the two most commonly used instruction sets, which are discussed in this unit.

8.1 Need for Addressing Modes

The operands of the instructions can be located either in the main memory or the CPU registers. If the operand is placed in the main memory, then the instruction provides the location address in the operand field. Many methods are followed to specify the operand address. The different methods/modes for specifying the operand address in the instructions are known as addressing modes. The exact addressing mode used in the instruction can be specified to the control unit by using any of the following two methods:

1. The opcode explicitly specifies the addressing mode in the instruction.
2. A separate addressing mode field is specified in the instruction.

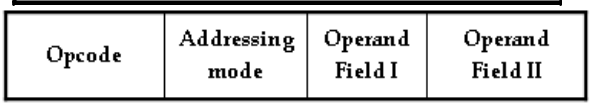
Notes

Some of the most common addressing modes used by the computers are:

1. Direct Addressing Mode
2. Indirect Addressing Mode
3. Register Addressing Mode
4. Immediate Addressing Mode
5. Index Addressing Mode

Figure 8.1 depicts the addressing mode field.

Figure 8.1: Addressing Mode Field

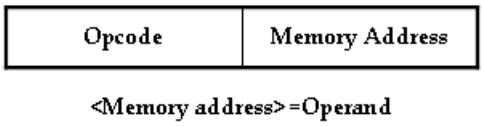


As seen in figure 8.1, the addressing mode field comprises an opcode, an addressing mode, an operand field I, and an operand field II. The addressing mode could be any of the five addressing modes mentioned above.

The addressing modes can be described as follows:

Direct Addressing Mode

Figure 8.2: Direct Addressing



In direct addressing mode, the operand address is explicitly specified in the instruction. This mode can be illustrated using the below assembly language statements:

LOAD R1, A //The content of memory location A is loaded into register R1

MOV B, A //The content of memory location A is moved to memory location B

JUMP A //The program control is transferred to the instruction at memory location A

In direct addressing mode, the operand address is directly available in the instruction. Hence, eliminating the operand address calculation step the instruction cycle time decreases. However, the operand field in the instruction limits the number of bits for the operand address.

Notes Instruction cycle is the process in which the computer retrieves an instruction from its memory, determines the actions, and performs those actions.

Indirect Addressing Mode

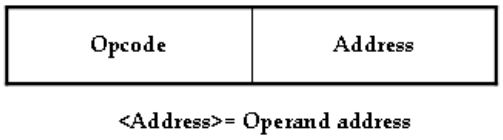
In indirect addressing mode, the address of the location 'A' contains address of another location 'B', which actually holds the operand. It is represented as below:

A = B, B = operand

Thus, 'A' is considered as the pointer. By modifying the content of location 'A', you can change the value of 'B', without changing the instruction. The below assembly language statement illustrates this mode.

MOVE (A), R1 //The content present in the address A is loaded into the register R1.The figure 8.3 illustrates indirect addressing mode.

Figure 8.3: Indirect Addressing



The indirect addressing mode provides flexibility in programming. You can change the address during program run-time without altering the contents of the instruction. However, as there are two memory accesses even for the single level indirect addressing, the instruction cycle time increases.

Register Addressing Mode

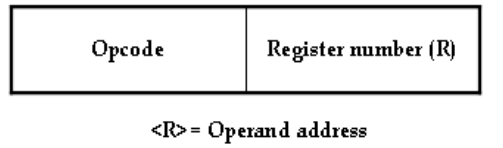
In register addressing mode, the register holds the operand. In the instruction, the register number that holds the operand is specified. The long programs find this mode useful as it helps to store the intermediate results in the registers. The following assembly language statements illustrate this mode:

ADD R1, R2 //The contents of the registers R1 and R2 are added and the result of the addition is stored in register R1.

STORE R1, M1 //The contents of the register R1 are stored in memory address M1.

Here, the first operand uses register addressing mode and the second operand uses direct addressing mode. The figure 8.4 depicts the register addressing mode

Figure 8.4: Register Addressing



The register addressing mode provides faster operand fetch without memory access. However, the number of registers is limited. Hence, the programmers must effectively utilize the registers.

Immediate Addressing Mode

In immediate addressing mode, the operand is a part of the instruction. Hence, memory reference is not required to retrieve the operand. This mode is used to define constants and set initial values to the variable. The below assembly language statements illustrate the immediate addressing mode:

MOVE #14, R1 or MVI R1, 14 //The binary equivalent of 14 is loaded in the register R1

ADD #14, R1 //The binary equivalent of 14 and the contents of R1 are added and the result is stored in register R1

CMP #14, R1 //The binary equivalent if 14 is compared with the contents of R1.

The '#' sign indicates that the constant following the sign is the immediate operand.

Notes

The figure 8.5 depicts the immediate addressing mode.

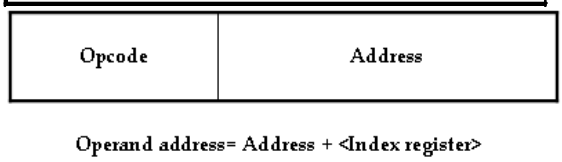
Figure 8.5: Immediate Addressing

Once the instruction is fetched, the operand is also fetched in the instruction. This reduces the instruction cycle time. However, the value of the operand is limited because this mode is limited to the size of the address field.

Index Addressing Mode

Figure 8.6 depicts index addressing mode.

Figure 8.6: Index Addressing



The index addressing mode includes an index register which holds an offset/displacement. The effective address of the operand is obtained by adding the offset with the contents of the registers present in the instruction.

The start address of an array in the memory is obtained from the address field in the instruction. The difference between the start address and the operand address provides an index value for the operand. The index value is stored in the index register. The operands are stored in consecutive locations in the memory. Thus, by changing the value of the index or by incrementing the index value, you can access any operand in the array.

Some CPUs possess auto-indexing feature, which automatically increments the index registers when an instruction with index addressing is executed.

These addressing modes provide flexibility in writing efficient programs. Addressing modes help in reducing the instruction length by including a separate field for the address. This helps the programmers to handle complex tasks such as loop control, program relocations, and indexing of an array.

8.2 Data Transfer and Manipulation

Computer systems consist of a set of instructions that help the users to easily carry out their computational tasks. The instruction set differs from one computer system to another. The operations are represented by binary codes and the binary code assignment in the operation field of the instruction can be different in different computers. However, the actual operations in the instruction set are not very different from one computer system to another. The symbolic names of the instruction (assembly language notation) may also differ from one computer to another. An instruction usually contains opcode, addressing field, and operand field. There are different types of opcodes and based on the type of opcode, the instructions can be classified as follows:

1. Data Transfer Instructions
2. Data Manipulation Instructions
3. Program Control Instructions

Notes

Data transfer instructions transfer data from one location to another without causing any change in the content present in the binary form. Data manipulation instructions perform arithmetic, logic, and shift operations. Program control instructions provide decision making abilities and are able to change the execution sequence. The program control instructions are explained in the next section.

Data Transfer Instructions

Data transfer instructions move the data between memory and processor registers, processor registers and I/O devices, and from one processor register to another. There are eight commonly used data transfer instructions. Each instruction is represented by a mnemonic symbol. Table 8.1 shows the eight data transfer instructions and their respective mnemonic symbols.

Table 8.1: Data Transfer Instructions

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

The instructions can be described as follows:

1. ***Load:*** The load instruction is used to transfer data from the memory to a processor register, which is usually an accumulator.
2. ***Store***: The store instruction transfers data from processor registers to memory.
3. ***Move:*** The move instruction transfers data from processor register to memory or memory to processor register or between processor registers itself.
4. ***Exchange:*** The exchange instruction swaps information either between two registers or between a register and a memory word.
5. ***Input:*** The input instruction transfers data between processor register and input terminal.
6. ***Output:*** The output instruction transfers data between processor register and output terminal.
7. ***Push and Pop:*** The push and pop instructions transfer data between a processor register and memory stack.

All these instructions are associated with a variety of addressing modes. Some assembly language instructions use different mnemonic symbols just to differentiate between the different addressing modes.

Example: The mnemonic symbols for load immediate is LDI

Thus, it is necessary to be familiar with various addressing modes and different types of instructions to write efficient assembly language programs for a computer.

Notes

Data Manipulation Instructions

Data manipulation instructions have computational capabilities. They perform arithmetic, logic, and shift operations on data. There are three basic types of data manipulation instructions:

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

During execution of the instruction, each instruction goes through the fetch phase, where it reads the binary code of the instruction from the memory. According to the rules of the instruction addressing mode, the operands are brought in processor registers. Finally, the instruction in the processor is executed.

Arithmetic Instructions

Arithmetic operations include addition, subtraction, multiplication and division. Some computers provide instructions only for addition and subtraction operations, and generate multiplication and division operations from these two operations. Each instruction is represented by a mnemonic symbol. Table 8.2 illustrates some of the arithmetic instructions and their respective mnemonic symbols.

Table 8.2: Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with Carry	ADDC
Subtract with Borrow	SUBB
Negation	NEG

The description of these instructions is as follows:

1. ***Increment:*** The increment instruction adds 1 to the value stored in register or memory word.
2. ***Decrement:*** The decrement instruction subtracts 1 from the contents stored in register or memory word.
3. ***Arithmetic Instructions:*** The arithmetic instructions are available for different types of data such as floating point, binary, single precision, or double precision data.

During execution of arithmetic instructions, the processor status flags or conditional codes are set to designate the outcome of the operation.

Example: For the conditions generated as a carry or borrow, the outcome is either 0 or negative.

A flip-flop is used to store the carry from an addition operation. The add with carry instruction performs the addition of two numbers along with the value of carry from the previous computation. Similarly, the subtract with borrow instruction performs the subtraction of two numbers and a borrow if any, from the previous computation. The negation instruction represents the 2's complement of a number.

Notes

Logical and Bit Manipulation Instructions

Logical instructions carry out binary operations on the bits stored in the registers. In logical operations, each bit of the operand is treated as a Boolean variable. Logical instructions can change bit value, clear a group of bits, or can even insert new bit value into operands that are stored in registers or memory words. Each logical instruction is represented by mnemonic symbols.

Notes Boolean variable is a numerical variable that can hold a single binary bit (0 or 1).

Table 8.3 illustrates some of the logical instructions andtheir respective mnemonic symbols.

Table 8.3: Logical Instructions

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

The clear instruction replaces the specific operand by 0's. The complement instruction inverts all the bits of the operand and produces 1's complement. The AND, OR, and XOR instructions perform logical operations on each bit or group of bits of the operand.

Logical instructions can also manipulate individual bits or group of bits. The bit manipulation operation can clear a bit to 0, can set a bit to 1, or can complement a bit.

The AND instruction can clear a bit or group of bits of an operand. For Boolean variable a, the relationship 'ab0 = 0' and 'ab1 = a' indicates that the binary variable when ANDed with 0 changes the value to 0. However, the variable when ANDed with 1 does not change the value. Thus, bits of an operand can be cleared by ANDing the operand with another operand that has to clear all 0 bits in its position. It is also known as mask because it masks 0s in selected bit positions of an operand.

The OR instruction can set a bit or group of bits of an operand. For Boolean variable a, the relationship 'a + 1 =1' and 'a + 0 = a' indicates that the binary variable when ORed with 1, changes the value to 1. However, the variable when ORed with 0 does not change the value. Thus, OR instruction is used to set the bits to 1 by ORing the bits of an operand with another operand that has 1s in its bit positions.

The XOR instruction can complement bits of an operand. For Boolean variable a, the relationship 'a + 1 = a' and 'a + 0 = a' indicates that the binary variable is complemented when XORed with 1. However, the variable does not change value when XORed with 0.

The carry bits can be cleared, set, or complemented with appropriate instructions. The bit manipulation instructions can also enable or disable the interrupt facility, which is controlled by the flip-flops.

Shift Instructions

Shift instruction helps to shift the bits of an operand to the right or to the left. The direction of shift is based on specific instructions. The operand is first loaded into the accumulator and then the shift operation is performed bit by bit.

Notes

The shift-left operation shifts the zero into low-order vacated position.

Example: Operand in Accumulator

1111 0101 1000 1011

After a shift-left operation

1 1110 1011 0001 0110 zero is shifted in

↑

High order bit is shifted out

In shift-right operations, zeros are shifted into high-order vacated position. The bits that are shifted can also be the original value of the sign bit as in case of arithmetic right shift or can be the bits that are shifted out of low-order position of the accumulator-extension as in case of Rotate Right Accumulator and Extension (RRAE). The main purpose of this RRAE is to fetch the bits from the accumulator-extension position 15 and shift the bits back to the accumulator position 0. This ensures that no bits are lost during the shift.

The shift operation can be ended either by decrementing the shift count to zero or by shifting the bit value of 1 into the high-order position (bit 0) of the accumulator.

Example: Consider the below example program to add two numbers:

```
NAME   Addition                    //name of the program
PAGE   52,80
TITLE  8086 assembly language program to add two numbers //Title of the
program
.model small                 //implies that the program is a small program
.stack 100                       //memory allocation is 100
.data                            // data for the program
   Number1     DB   63H
   Number2     DB   2EH
   Result      DW   ?
.code                            //marks the beginning of the code
START:  MOV   AX, @data
        MOV   DS, AX
        MOV   AL, Number1
        ADD   AL,  Number2
        MOV   AH, 00H
        ADC   AH, 00H
        MOV   Result, AX
        END START
```

Notes

In the above program,

1. .model specifies the mode for assembling the program.
2. DB allocates and initializes the bytes of storage.
3. Number1 DB 63H indicates the first number storage and Number2 DB 2EH indicates the second number storage.
4. DW allocates and initializes the words of storage.
5. Result DW indicates that double byte is reserved for the result.
6. The instructions MOV AX, @data and MOV DS, AX initialize the data segment.
7. MOV AL, Number1 transfers the first number to AL.
8. ADD AL, Number2 adds the second number to AL.
9. MOV AH, 00H makes Most Significant Bit of result zero.
10. ADC AH, 00H puts carry in AH.
11. MOV Result, AX copies result to the memory.

8.3 Program Control

Instructions of the computer are always stored in consecutive memory locations. These instructions are fetched from successive memory locations for processing and executing. When an instruction is fetched from the memory, the program counter is incremented by 1 so that it points to the address of the next consecutive instruction in the memory. Once a data transfer and data manipulation instruction is executed, the program control along with the program counter, which holds the address of the next instruction to be fetched, is returned to the fetch cycle.

Data transfer and manipulation instructions specify the conditions for data processing operations, whereas the program control instructions specify the conditions that can alter the content of the program counter. The change in the content of program counter can cause an interrupt/break in the instruction execution. However, the program control instructions control the flow of program execution and are capable of branching to different program segments.

Some of the program control instructions are listed in table 8.4.

Table 8.4: Program Control Instructions

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

The branch is a one-address instruction. It is represented as BR ADR, where ADR is a mnemonic for an address. The branch instruction transfers the value of ADR into the program counter. The branch and jump instructions are interchangeably used to mean the same. However, sometimes they denote different addressing modes.

Notes

Branch and jump instructions can be conditional or unconditional. The unconditional branch instruction creates a branch to the specified address without any conditions.

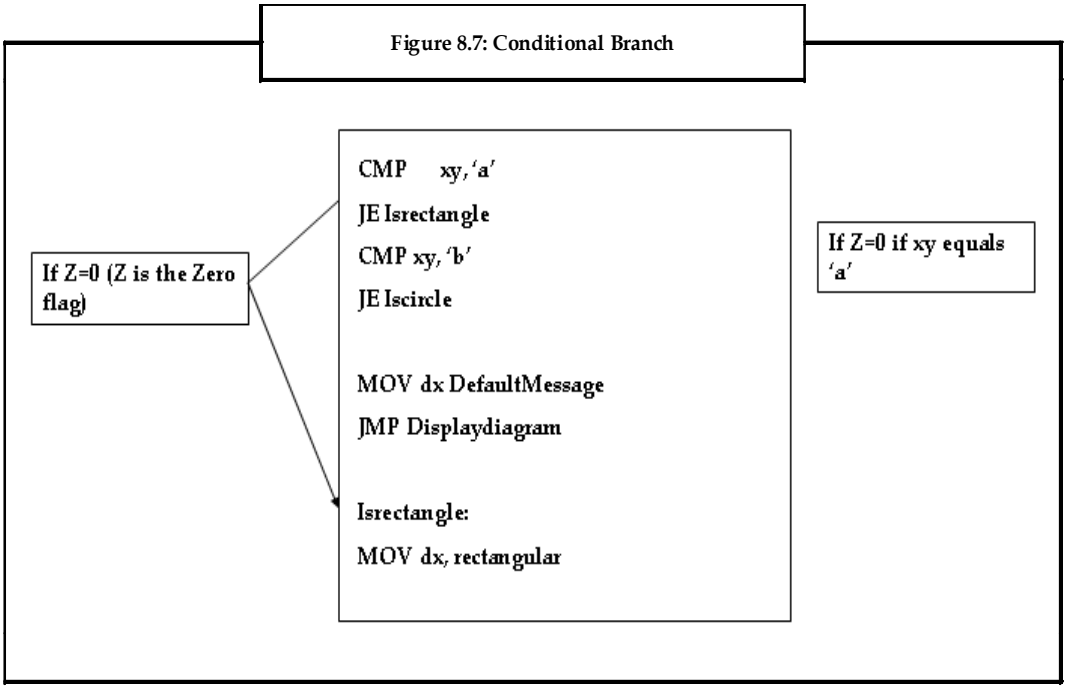
Example: JMP Displaydiagram

The JMP instruction transfers the flow of execution, without considering the actual condition of the flags, to the indicated operator. The above instruction makes the control jump to the part of the code where Displaydiagram is specified.

The conditional branch instructions such as 'branch if positive', or 'branch if zero' specify the condition to transfer the flow of execution. When the condition is met, the branch address is loaded in the program counter.

Figure 8.7 depicts the conditional branch instructions.

Figure 8.7: Conditional Branch



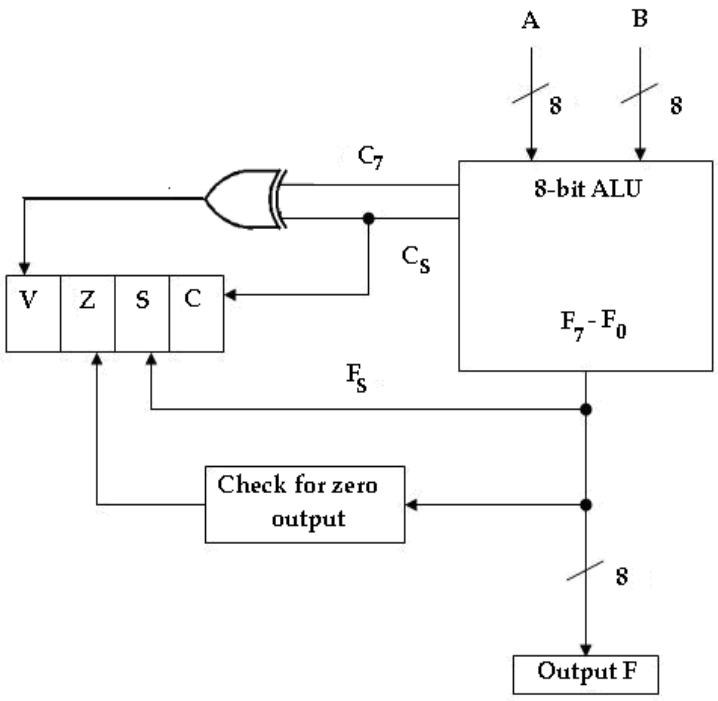
The compare instruction performs an arithmetic subtraction. Here, the result of the operation is not saved; instead, the status bit conditions are set. The test instruction performs the logical AND operation on two operands and updates the status bits.

Notes

Status Bit Conditions

The status register comprises the status bits. The bits of the status register are modified according to the operations performed in the ALU. Figure 8.8 depicts a block diagram of an 8-bit ALU with a 4-bit status register.

Figure 8.8: Status Register Bits



In figure 8.8, if the end carry C_8 is 1, then carry (C) is set to 1. If C_8 is 0, then C is cleared to 0.

If the highest order bit F_7 is 1, then Sign (S) is set to 1. If F_7 is 0, then S is set to 0.

If the output of ALU is 0, then zero (Z) is set to 1, otherwise Z is set to 0.

If XOR of the last two carries is equal to 1, then overflow (V) is set to 1, otherwise V is cleared to 0.

The result of the 8-bit ALU operation is either 127 or -127.

Z is a status bit used to indicate the result obtained after comparing A and B. Here, XOR operation is used to compare two numbers (Z = 0 if A = B).

Notes

Conditional Branch Instruction

The conditional branch instruction checks the conditions for branching using the status bits. Some of the commonly used conditional branch instructions are shown in table 8.5.

Table 8.5: Conditional Instructions

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

Thus, when the status condition is true, the program control is transferred to the address specified in the instruction, otherwise the control continues with the instructions that are in the subsequent locations. The conditional instructions are also associated with the program control instructions such as jump, call, or return.

The zero status bit checks if the result of the ALU is zero or not. The carry bit checks if the most significant bit position of the ALU has a carry out. It is also used with rotate instruction to check whether or not the bit is shifted from the end position of a register into a carry position. The sign bit indicates the state of the most significant bit of the output from the ALU (S = 0 denotes positive sign and S = 1 denotes negative sign). The branch if plus and branch if minus are used to check whether the value of the most significant bit represents a sign or not. The overflow and underflow instructions are used in conjunction with arithmetic operations performed on signed numbers. The higher and lower words are used to denote the relations between unsigned numbers, whereas the greater and lesser words are used to denote the relations between signed numbers.

Notes

Example: Consider two numbers M = 11110000 and N = 00010100. When we perform M - N operation, the result obtained is

M:	11110000
N:	00010100
M-N:	11011100

The compare instruction updates the status bits:

C =1: Carry out of the last stage

S = 1: Left most bit is 1

Z = 0: Last two carries are equal to 1

V = 0: Result is not equal to 0

Consider the numeric value, where M = 60 and B =10 (unsigned numbers). Here, M > N and M $\neq$ N. Therefore, the instructions that will cause branch are BHI, BHE and BNE.

If M = -8 and N = 10, then we have M < N and M $\neq$ N. Therefore, the instructions that will cause branch are BLT, BLE, and BNE

8.4 RISC and CISC

The design of the instruction set for the processor is considered as an important aspect of computer architecture. The machine language program is developed based on the instruction set chosen for that particular computer.

Earlier, the hardware components of the computer were expensive and to minimize this expense, the programmers started to build simple and small instructions. With the advent of integrated circuits, the digital hardware became cheaper and the computer instructions started to increase in number and complexity. Many computers have more than 100 instruction sets. Such computers with large number of instructions are classified as a Complex Instruction Set Computers (CISC).

In 1980s, computer architects started to design computers with fewer instructions in order to execute programs at a much faster rate within the CPU. Such computers with less number of instructions are classified as a Reduced Instruction Set Computer (RISC).

Complex Instruction Set Computer (CISC)

A Complex Instruction Set Computer (CISC) comprises a complex instruction set. It incorporates variable length instruction format. Instructions that require register operands may take only two bytes. However, the instructions that require two memory addresses may take five bytes to include the complete instruction code. Thus, CISC has variable length encoding of instructions and the execution of instructions may take varying number of clock cycles. The CISC processor provides direct manipulation of operands that are in memory.

Example: An ADD instruction will use index addressing to specify one operand in memory and direct addressing to specify second operand in memory. This instruction would use another memory location to store the result. Thus, this instruction would use three memory references for execution.

Many CISC architectures read the inputs and write their outputs in the memory system instead of a register file. As CISC architecture takes large number of addressing modes, more hardware logic is required to implement them. This reduces the computation speed.

Basically, the CISC architecture attempts to provide a single machine instruction for the statements that are written in a high-level language.

Notes

Example: The IBM 370 computer uses the CISC architecture.

Reduced Instruction Set Computer (RISC)

In Reduced Instruction Set Computer (RISC) architecture, the instruction set of the computer is simplified to reduce the execution time. RISC has a small set of instructions, which generally include register-to-register operations. Thus, data is stored in processor registers for computations and results of the computations are transferred to the memory using store instructions. All operations are performed within the registers of the CPU. In RISC, all instructions have simple register addressing and hence use less number of addressing modes.

RISC uses relatively a simple instruction format and is easy to decode. Here, the instruction length can be fixed and aligned on word boundaries. The RISC processors have the ability to execute one instruction per clock cycle. This is done using pipelining, which involves overlapping the fetch, decode, and execute phases of two or three instructions.

As RISC takes relatively a large number of registers in the processor unit, it takes less time to execute its program when compared to CISC.

Example: The Scalable Processor Architecture (SPARC) is an example of RISC architecture.

8.4.1 RISC Instruction Set

RISC instruction set includes simpler instructions with hard-wired control, large number of registers, simpler processor pipeline and increased clock-rate. The RISC processor's instruction set is restricted to load and store instructions when there is an interaction between memory and CPU. All other instructions are executed within registers without any reference to memory.

Example: Consider an example program for a RISC-type CPU, which include load and store instructions that have one memory and one register address, and arithmetic instructions that are specified by processor registers. The below program evaluates X = (P + Q)*(R + S)

```
LOAD   R1  P          R1    M[P]
LOAD   R2  Q          R2    M[Q]
LOAD   R3  R          R3    M[R]
LOAD   R4  S          R4    M[S]
ADD    R1, R1,  R2    R1    R1 + R2
ADD    R3, R3, R4     R3    R3 + R4
MUL    R1, R1, R3     R1    R1 + R3
STORE  X,  R1         M[X]  R1
```

The LOAD instructions transfer the operand P, Q, R and S from memory to CPU registers R1, R2, R3, and R4 respectively. The ADD and MUL instructions execute the addition and multiplication operations with the data in the registers without referring to the memory. The STORE instruction stores the result of the computation in the memory (M[X]).

Notes

8.4.2 RISC Versus CISC

There are some significant differences between RISC and CISC processors. The comparison between the common characteristics of RISC and CISC processor is shown in table 8.6:

Table 8.6: Comparison Between RISC and CISC

RISC	CISC
Few instructions	Many instructions
Few addressing modes. Most instructions have register to register addressing modes	Many addressing modes
Includes simple instructions and takes one cycle	Includes complex instructions and takes multiple cycles
Some of the instructions refer to memory	Most of the instructions refer to memory
Hardware executes the instructions	Microprogram executes the instructions
Fixed format instructions	Variable format instructions
Easier to decode as instructions have fixed format	Difficult to decode as instructions have variable format
Multiple register sets are used	Single register set is used
RISC is highly pipelined	CISC is not pipelined or less pipelined
Load and store functions are separate instructions	Load and store functions are found in a single instruction

Today, RISC and CISC architectures are becoming more alike. Many RISC chips now support instructions of CISC chips also. Similarly, CISC chips are using many techniques associated with RISC chips.

8.5 Summary

- Addressing modes provide different methods for specifying operand address in the instruction.
- Some of the commonly used addressing modes are direct addressing mode, indirect addressing mode, register addressing mode, immediate addressing mode, and index addressing mode.
- Data transfer instructions help to move the data from one location to another. Data manipulation instructions perform arithmetic, logic, and shift operations on data.
- Program control instructions specify the conditions for data processing operations.
- The Complex Instruction Set Computer (CISC) consists of many complex instruction sets.
- The Reduced Instruction Set Computer (RISC) consists of less instruction sets and executes the instructions at a greater speed.

8.6 Keywords

Accumulator: A processor register that stores intermediate arithmetic and logic results.

Flip-flops: An electronic circuit that is interconnected to form logic gates. It changes its state when it receives the input pulse (trigger). Hence, it is also known as bistable gate.

Microprogram: A computer program that has basic elemental commands which control the operation of each components of a microprocessor.

Pipelining: In pipelining, while the processor is performing arithmetic operations, the computer architecture allows the next instructions to be fetched, holding them in a buffer close to the processor until each instruction operation can be performed.

Notes

8.7 Self Assessment

1. State whether the following statements are true or false:
   (a) The number of address fields in the instruction format of a computer varies according to the organization of the stack.
   (b) In direct addressing mode, the register holds the operand.
   (c) The store instruction transfers data from processor register to memory.
   (d) Data transfer and manipulation instructions specify the conditions that can alter the content of the program counter.
   (e) Complex Instruction Set Computer (CISC) incorporates variable length instruction format.
2. Fill in the blanks:
   (a) The different methods/modes for specifying the operand address in the instructions are known as _____.
   (b) The push and pop instructions transfer data between a processor register and _____.
   (c) In conditional branch instruction, when the condition is met, the branch address is loaded in the _____.
   (d) RISC has small set of instructions, which generally include _____ operations.
3. Select a suitable choice for every question:
   (a) In register addressing mode, the register holds the _____
      (i) Operand
      (ii) Opcode
      (iii) Address
      (iv) Register number
   (b) A flip-flop is used to store the carry from _____ operation.
      (i) Addition
      (ii) Subtraction
      (iii) Comparison
      (iv) Division
   (c) The test instruction performs the logical _____ operation on two operands and updates the status bits.
      (i) SUB
      (ii) OR
      (iii) XOR
      (iv) AND

Notes

(d) The conditional branch instruction checks the conditions for branching using the _____.

(i) Clock cycles

(ii) Registers

(iii) Instruction codes

(iv) Status bits

Answers: Self Assessment

1. (a) False
   (b) False
   (c) True
   (d) False
   (e) True
2. (a) Addressing modes
   (b) Memory stack
   (c) Program counter
   (d) Register-to-register
3. (a) Operand
   (b) Addition
   (c) AND
   (d) Status bits

8.8 Review Questions

1. "In direct addressing mode, the operand address is explicitly specified in the instruction." Explain with examples.
2. "Data manipulation instructions have computational capabilities." Comment.
3. "Data transfer and manipulation instructions specify the conditions for data processing operations." How?
4. "The conditional branch instruction checks the conditions for branching using the status bits." How?
5. "Complex Instruction Set Computer (CISC) comprises complex instruction set." Justify
6. "RISC instruction set includes simpler instructions." Explain with an example.
7. "There are some significant differences between RISC and CISC processor." Provide the differences.

Notes

8.9 Further Readings

Books

Morris M. Computer System Architecture. Pearson Education.

A.P.Godse & D.A.Godse (2010). Computer Organization And Architecture. Pune: Technical Publications.

Stallings. W (2009). Computer Organization and Architecture: Designing for Performance. Prentice Hall.

Online links

www.mans.edu.eg/faceng/english/computers/PDFS/PDF4/1.2.pdf

www.ehow.com/list_7332165_types-addressing-modes-computers.html

Notes

Unit 9: Computer Arithmetic I

CONTENTS

Objectives
Introduction
9.1 Addition and Subtraction
9.2 Multiplication
9.2.1 Hardware Algorithm
9.2.2 Booth Multiplication Algorithm
9.3 Summary
9.4 Keywords
9.5 Self Assessment
9.6 Review Questions
9.7 Further Readings

Objectives

After studying this unit, you will be able to:

- Explain the addition of binary numbers
- Define the method to subtract binary numbers
- Describe the method of multiplying binary numbers

Introduction

Today, the knowledge on hardware and software utilization to perform computations has improved to a great extent. Digital computer arithmetic that is used in digital computers has emerged in two ways: one, as an aspect of logic design and the other as development of effective algorithms to use the available hardware.

Computers carry out arithmetic instructions at the bit level. That is, they follow the binary system, which consists of 0s and 1s. The arithmetic instructions perform arithmetic calculations and are responsible for processing data in computers. Addition, subtraction, multiplication, and division are the four basic arithmetic operations. These four basic operations are utilized to develop other arithmetic functions. The addition, subtraction and multiplication operation are discussed in this unit. This unit describes various algorithms required to perform arithmetic operations.

9.1 Addition and Subtraction

Addition of binary numbers is easy yet tedious at the same time. It is a fundamental feature of digital computers, and hence it is important to know how to add the binary digits. Almost all the operations of a computer depend on binary addition. Once we understand the addition of two binary digits, it is easier to understand subtraction, multiplication, and division of binary digits.

We shall begin by adding two binary bits. As you are aware a bit can be either 0 or 1. Therefore, we can have only four possible input combinations. The four possible input combinations and their output are as follows:

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

Notes

1 + 1 = 10

In the above four possibilities, we can observe that the fourth possibility results in a 2-bit output. Table 9.1 depicts the method of handling such output.

Table 9.1: Binary Addition

Input		Output	
P	Q	Carry	Sum (P + Q)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

As depicted in table 9.1, the carry digit handles the possibility of overflow. Here, overflow refers to the extra digit that we obtain on adding 1 and 1. The overflow or carry digit is carried forward to the next most significant digit in the operation.

We shall now look at a more complex example of adding binary digits.

Example: Add **1011011 + 100111**

```
  1 0 1 1 0 1 1
  + 1 0 0 1 1 1
 ---------------
 1 0 0 0 0 0 1 0
 ---------------
  1 1 1 1 1 1     ---->   Carry bits
```

In the above example:

1 + 1 = 0 (one carry)

1 + 1 (+ the carried digit 1) = 1 (one carry)

0 + 1 (+ the carried digit 1) = 0 (one carry)

1 + 0 (+ the carried digit 1) = 0 (one carry)

1 + 0 (+ the carried digit 1) = 0 (one carry)

0 + 1 (+ the carried digit 1) = 0 (one carry)

1 + 0 (+ the carried digit 1) = 0 (one carry)

The last digit that is carried is placed on the left hand side of the result. Therefore, the output is **10000010.**

Notes While performing addition operations on binary numbers always start adding from the right side and move to the left.

Notes

Task Add binary numbers 1100110 and 1110001 using the method of addition of binary numbers.

Once we understand the concept of addition of binary numbers, it is easy to learn the subtraction process of binary numbers. Binary numbers are subtracted by performing two's complement on the subtrahend. Two's complement is done through the following steps:

1. Complement every digit. That is, change 1 to 0 and 0 to 1.
2. Add 1 to the output.

The following example illustrates the subtraction operation of binary digits using the above mentioned steps.

Example: **11101011 - 01100110**

The second value 01100110 is to be subtracted from the first value 11101011.

First apply two's complement to the second value 01100110, i.e., follow the two steps as shown.

Step 1: 0 1 1 0 0 1 1 0

1 0 0 1 1 0 0 1 (change 1 to 0 and 0 to 1)

Step 2: 1 0 0 1 1 0 0 1

+1 (add 1)

1 0 0 1 1 0 1 0 (resultant)

1

Then, add the resultant to the first value.

1 1 1 0 1 0 1 1 (first value)

+1 0 0 1 1 0 1 0 (resultant)

1 0 0 0 0 1 0 1 (output)

ignore ← $\boxed{1}$ 1 1 1 1 1

The output is **10000101.**

Note that we have to ignore the carry bit.

We now know the process of subtracting smaller number from larger number. We shall now examine the subtraction of a greater number from a smaller number.

The Most Significant Bit (MSB) or the leftmost bit is set to 1 to indicate a negative number. The MSB is known as the sign bit. The remaining 7 bits are used to express the value.

The following are the steps to subtract a greater number from a smaller number:

1. Apply two's complement to the smaller number.
2. Add the resultant value to the smaller number.
3. Change MSB to 0.
4. Apply two's complement to the resulting number.

MSB now indicates a negative value.

Notes

We shall understand this subtraction process by considering an example.

Example: **10010101 – 10110100**

Step 1: 1 0 1 1 0 1 0 0 (greater number)
0 1 0 0 1 0 1 1 (change 1 for 0 and 0 for 1)

0 1 0 0 1 0 1 1
+1 (add 1)
0 1 0 0 1 1 0 0
1 1

Step 2: 1 0 0 1 0 1 0 1 (smaller number)
+**0 1 0 0 1 1 0 0** (add the resultant value to smaller number)
1 1 1 0 0 0 0 1
1 1 1

Step 3: 1 1 1 0 0 0 0 1
0 1 1 0 0 0 0 1 (change MSB bit to 0)

Step 4: 0 1 1 0 0 0 0 1
1 0 0 1 1 1 1 0 (change 1 for 0 and 0 for 1)

1 0 0 1 1 1 1 0
+1
1 0 0 1 1 1 1 1

Output = **10011111** (MSB indicates a negative value)

Task Subtract binary numbers 1111000 and 1010111.

We can represent negative fixed-point binary using the following three methods:

1. Signed-magnitude
2. Signed-1's complement
3. Signed-2's complement

Signed-magnitude method is used by computers to perform floating-point operations. Signed-2's complement method is used by most computers for arithmetic operations performed on integers.

In this section, we will develop algorithms for addition and subtraction to represent data in signed-magnitude and signed 2's complement methods.

The adopted representation for negative numbers relates to representation of numbers in the register before and after executing the arithmetic operation. However, it does not mean that we should not use complement arithmetic in intermediate steps.

Example: Employment of complement arithmetic is suitable to perform subtraction operations with numbers in signed-magnitude representation.

Complements used in an intermediate step do not affect the fact that the representation is in signed-magnitude. This holds true as long as the initial minuend and subtrahend and the final difference are in signed-magnitude.

Notes

Performing Addition and Subtraction with Signed-Magnitude Data

Performing addition and subtraction using signed-magnitude data is relatively easy. It is essential to understand this process to derive the hardware algorithm.

There are eight conditions to consider while adding or subtracting signed numbers. These conditions depend on the operations performed and the sign of the numbers. Table 9.2 depicts the algorithm for addition and subtraction. The first column in table 9.2 depicts these conditions. The other columns of the table depict the actual operations to be performed with the magnitude of numbers. The last column of the table is required to avoid a negative zero. This means that when two equal numbers are subtracted, the output should not be **- 0.** It should always be **+0.**

In table 9.2, the magnitude of the two numbers is represented by **P** and **Q.**

Table 9.2: Addition and Subtraction of Signed-Magnitude Numbers

Operation	Addition of Magnitudes	Subtraction of Magnitudes		
		P>Q	P<Q	P=Q
(+P) + (+Q)	+(P+Q)			
(+P) + (-Q)		+(P-Q)	-(Q-P)	+(P-Q)
(-P) + (+Q)		-(P-Q)	+(Q-P)	+(P-Q)
(-P) + (-Q)	-(P+Q)			
(+P) - (+Q)		+(P-Q)	-(Q-P)	+(P-Q)
(+P) - (-Q)	+(P+Q)			
(-P) - (+Q)	-(P+Q)			
(-P) - (-Q)		-(P-Q)	+(Q-P)	+(P-Q)

In table 9.2 the addition algorithm states that:

1. When the signs of **P** and **Q** are the same, add the two magnitudes and attach the sign of P to the output.
2. When the signs of **P** and **Q** are different, compare the magnitudes and subtract the smaller number from the greater number.
3. The signs of the output have to be the same as **P** in case **P > Q** or the complement of the sign of **P** in case **P < Q**.
4. When the two magnitudes are the same, subtract **Q** from **P** and change the sign of the output to positive.

The subtraction algorithm states that:

1. When the signs of **P** and **Q** are different, add the two magnitudes and attach the signs of **P** to the output.
2. When the signs of **P** and **Q** are the same, compare the magnitudes and subtract the smaller number from the greater number.
3. The signs of the output have to be the same as **P** in case **P > Q** or the complement of the sign of **P** in case **P < Q**.
4. When the two magnitudes are the same, subtract **Q** from **P** and change the sign of the output to positive.

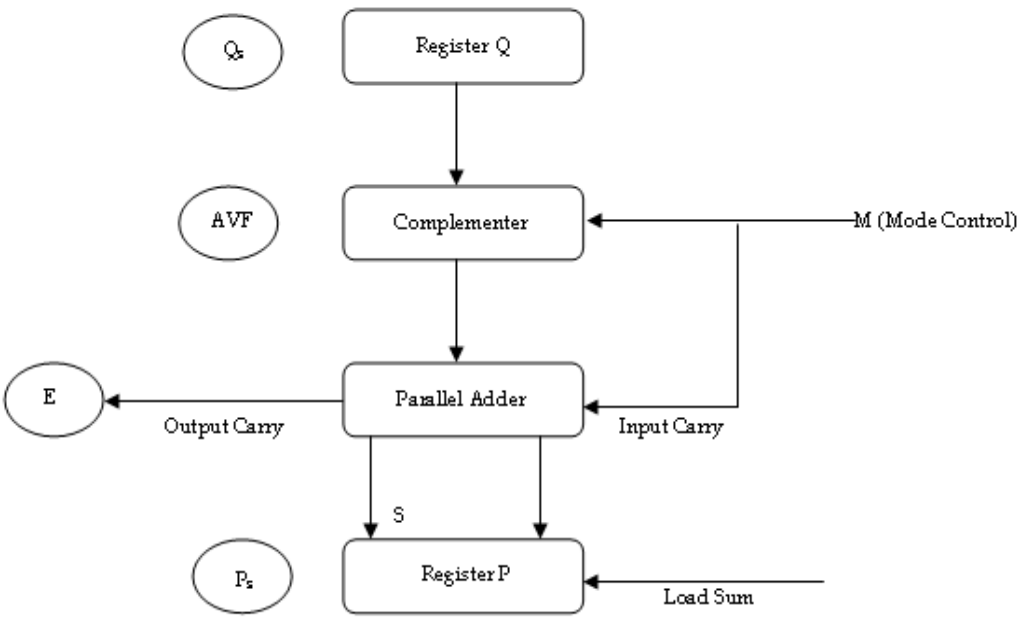
Notes

Notes There is not much difference between addition algorithm and subtraction algorithm, except for the sign comparison. The procedure that we need to follow for similar signs in addition algorithm is the same as for different signs in subtraction algorithm.

Implementation of the Two Arithmetic Operations with Hardware

It is essential to store numbers in registers to implement the two arithmetic operations with hardware. Figure 9.1 depicts a block diagram of hardware to implement the addition and subtraction operation.

Figure 9.1: Block Diagram of Hardware



In figure 9.1, two registers **P** and **Q** are created to hold the magnitudes of the numbers. $\mathbf{P_s}$ and $\mathbf{Q_s}$ are two flip-flops that hold the corresponding signs. A third register can be used to transfer the output of the operation. However, a saving is achieved if the output is transferred to **P** and $\mathbf{P_s}$. Therefore, when **P** and $\mathbf{P_s}$ are put together and an accumulator register is formed.

Now consider the implementation of the above algorithm in the hardware. First, a parallel-adder is required to perform the micro-operation **P + Q** Second, a comparator circuit is required to identify if **P > Q, P < Q or P = Q**. Third, two parallel-subtractor circuits are required to perform the micro-operation **P - Q and Q - P**. The sign relationship is ascertained using an exclusive OR gate with $\mathbf{P_s}$ and $\mathbf{Q_s}$ as input.

The procedure to ascertain sign relationship needs a magnitude comparator, an adder, and two subtractors.

Notes A different procedure that needs less equipment can be used to ascertain sign relationships.

Notes

We know that it is possible to perform subtraction through **complement** and **add.** We can determine the output of a comparison from the end carry after the subtraction. When we carefully investigate the alternatives, we can find that the utilization of two's complement for subtraction and comparison is an efficient method that needs only one adder and one complementer.

Subtraction is performed by adding P to the two's complement of **Q.** The resultant carry is taken to the **flip-flop E** where it is checked for the relative magnitudes of the two numbers **P** and **Q.** When **P** and **Q** are added, the overflow bits are stored in the **Add-Overflow Flip-flop (AVF).** The other micro-operations are rendered by the **Register P.** The micro-operations may be required to specify the sequence of steps in the algorithm.

The parallel adder is used to add **P** and **Q.** The **S(sum)** resultant of the adder is applied to the input of the **Register P.** Depending on the state of the **mode control M,** the complementer generates an output **Q** or the complement of **Q.** The parallel adder comprises full-adder circuits while the complementer comprises **exclusive-OR gates.** The **M** signal here is applied to the input carry of the adder.

In case **M = 0**:

1. The resultant of Q is transferred to adder.
2. The input carry is 0.
3. The output of adder is equal to the total of P + Q.

In case **M = 1**:

1. The one's complement of Q is applied to adder.
2. The input carry is 1.
3. The output S = P + Q + 1.

9.2 Multiplication

Binary multiplication is similar to decimal multiplication. The first step is to multiply each number, and the second step is to add the values together.

The following example depicts the multiplication of binary digits.

Example: Multiply **1011** by **11**

Step 1:

```
  1 0 1 1
    X 1 1
  1 0 1 1    (Multiply 1011 by 1)
1 0 1 1 0    (Multiply 1011 by 1 by adding a 0 as a placeholder)
```

Step 2:

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

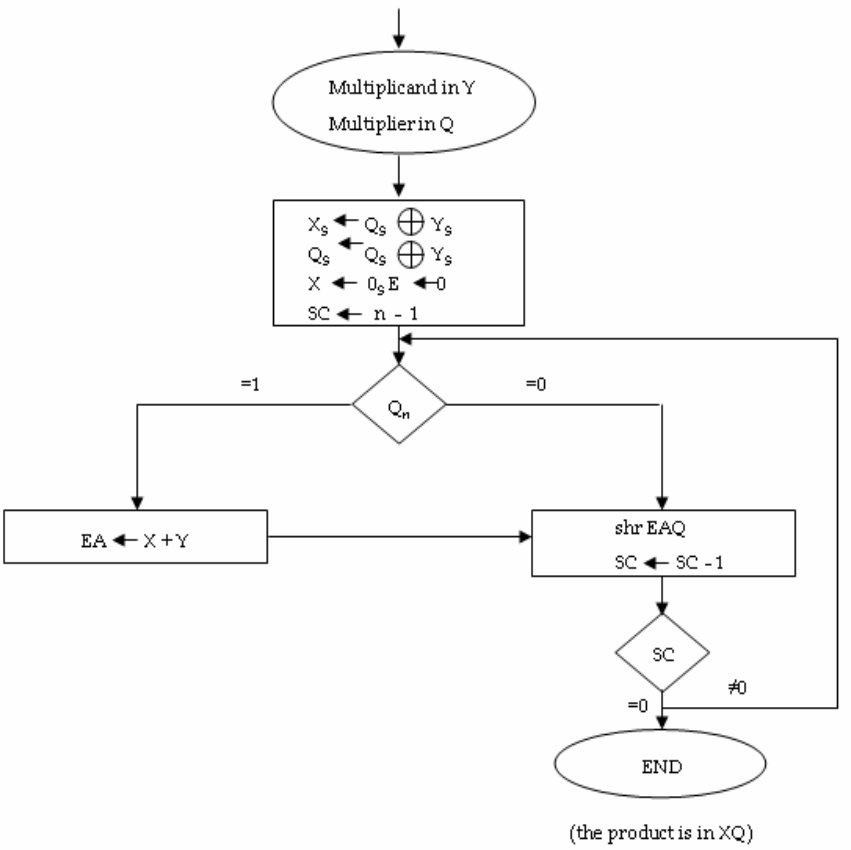
Notes The sign of the product is ascertained from the signs of the multiplier and multiplicand. If the signs of the multiplier and multiplicand are the same, the product sign is positive. If they are different, the product sign is negative.

Notes

9.2.1 Hardware Algorithm

An algorithm to multiply two numbers is known as the multiplication algorithm. The hardware multiply algorithm is used in digital electronics such as computers to multiply binary digits. Figure 9.2 depicts the flowchart for hardware multiply algorithm.

Figure 9.2: Hardware Multiply Algorithm Flowchart



In the flowchart shown in figure 9.2, the **multiplicand** is in **Y** and the **multiplier** is in **Q.** The signs related to **Y** and **Q** are in Y_s and Q_s respectively. These signs are compared and both **X** and **Q** are set to correspond to the sign of the product because a double length product will be stored in **registers X** and **Q.** The **registers X** and **E** are cleared. Then, the **Sequence Counter (SC)** is set to a number that is equal to the number of bits of the multiplier.

It is assumed that the operands are transferred from a memory unit to the registers having words of **n bits.** One bit of the word is occupied by the sign and the magnitude comprises **n - 1 bits** because the operand has to be stored with its sign.

Once the initialization is done, the low-order bit of multiplier in Q_n is tested. In case the bit is 1, the multiplicand in Y is added to the present partial product that is stored in **X.** In case the bit is **0,** no action is performed.

Then the EAQ register is shifted once to the right in order to form the new partial product. The **SC** is decreased by **1** and its new value is checked. In case it is not equal to **0,** the process is repeated and a new partial product is formed. This process is halted when **SC** is equal to **0.**

Notes

The partial product that is generated in X is shifted to Q, one bit at a time and replaces the multiplier eventually. The final product is stored in **X** as well as **Q.** Here, **X** contains the **MSBs** and **Q** contains the **Least Significant Bits (LSBs).**

9.2.2 Booth Multiplication Algorithm

The Booth multiplication algorithm refers to a multiplication algorithm that is used to multiply two signed binary numbers in two's complement. This algorithm helps in the study of computer architecture.

Did u know? The Booth multiplication algorithm was invented by Andrew Donald Booth in 1951 while researching on crystallography at Birkbeck College in Bloomsbury, London.

Booth's algorithm involves the addition of one of two predetermined values (**A** and **S**) to a product (**P**) repeatedly, and then performing a rightward arithmetic shift on the product (**P**). Let us consider the predetermined values to be **A** and **S,** and the product to be **P.** Consider that the multiplicand and multiplier are **m** and **r** respectively. Let the number of bits in **m** and **r** be **x** and **y** respectively.

The Booth's multiplication algorithm involves the following steps:

Step 1: The values of **A** and **S** and the initial value of **P** are determined. These values should have a length that is equal to **(x + y + 1).**

1. For **A,** the **MSB** is filled with the value of **m** and the remaining **(y+1)** bits are filled with **zeros.**
2. For **S,** the **MSB** is filled with the value of **(-m)** in two's complement notation, and the remaining **(y + 1)** bits are filled with zeros.
3. For **P,** the **MSB** for **x** is filled with **zeros.** To the right of this value, the value of **r** is appended. Then, the **LSB** is filled with a **zero.**

Step 2: The **LSBs** of **P** are determined.

1. In case they are **01,** find the value of **P + A,** and ignore the overflow or carry if any.
2. In case they are **10,** find the value of **P + S,** and ignore the overflow or carry if any.
3. In case they are **00,** use **P** directly in the next step.
4. In case they are **11,** use **P** directly in the next step.

Step 3: The value obtained in the second step is arithmetically shifted by one place to the right. **P** is now assigned the new value.

Step 4: Step **2** and Step **3** are repeated for **y** number of times.

Step 5: The **LSB** is dropped from **P,** which gives the product of **m** and **r.**

Let us consider a simple example.

Example: Find the product of **3 x (-4),** where **m = 3, r = -4, x = 4 and y = -4.**

A = 001100001
S = 110100000
P = 000011000

The loop has to be performed four times since **y = 4.**

P = 000011**00**
Here, the last two bits are **00.**
Therefore, **P = 000001100** after performing the arithmetic right shift.

Notes

P = 000001100
Here, the last two bits are **00.**
Therefore, **P = 000000110** after performing the arithmetic right shift.

P = 000000110
Here, the last two bits are **10.**
Therefore, **P = P + S, which is 110100110.**
P = 111010011 after performing the arithmetic right shift.

P = 111010011
Here, the last two bits are **11.**
Therefore, **P = 111101001** after performing the arithmetic right shift.

The product is **11110100** after dropping the LSB from P.
11110100 is the binary representation of **-12**.

Booth's algorithm is useful in two ways. One is for fast multiplication, that is, when there are consecutive 0's and 1's in multiplier. The other is in signed multiplication.

9.3 Summary

- The addition of binary numbers is a fundamental feature of the digital computers.
- Every binary bit consists of 0s and 1s.
- In addition of binary numbers, the possibility of overflow is handled by introducing a carry digit.
- Binary numbers are subtracted by taking the second value that needs to be subtracted and then applying the 2's complement.
- To subtract larger numbers from smaller numbers first apply the 2's complement, and then add the resultant value to the smaller number. Next change the MSB to 0 and again apply 2's complement. MSB would then indicate a negative value.
- Negative fixed-point binary is represented in three ways, which includes signed-magnitude, signed-1's complement, and signed-2's complement.
- The binary digits have to be stored in registers to implement the two arithmetic operations with hardware.
- The use of 2's complement for subtraction and comparison is an efficient method requiring only an adder and a complementer.
- The binary multiplication of digits is performed by multiplying every digit and then adding the values together.
- In digital computers, hardware multiply algorithms can be used to multiply binary numbers.
- In Booth's multiplication algorithm one of two predetermined values are added to a product repeatedly and rightward arithmetic shift on the product is performed.

9.4 Keywords

Addend Digit: It is a number that is added to the augend.

Augend Digit: It is a number to which another number is added.

Registers: These are memory devices that are part of the computer memory. Registers have a particular address and are used to hold a specific type of data.

Subroutine: It is a set sequence of steps that is part of a larger computer program.

Notes

9.5 Self Assessment

1. State whether the following statements are true or false:
   (a) The possibility of overflow is handled by introducing a carry digit in the subtraction of binary digits.
   (b) The signed-2's complement is used by most computers when arithmetic operations are performed with integers.
   (c) The Booth's multiplication algorithm is useful when there are consecutive 0s and 1s in multiplier, and in signed multiplication.
2. Fill in the blanks:
   (a) Binary numbers are subtracted by first taking the second value to be subtracted and then applying the _____.
   (b) To indicate a negative number, the _____ or the left side bit is set to 1.
   (c) The _____ algorithm refers to a multiplication algorithm that is used to multiply two signed binary numbers in 2's complement notation.
3. Select a suitable choice in every question.
   (a) The two's complement is performed by:
      (i) Adding one to output and complementing every digit in turn.
      (ii) Complementing every digit in turn and adding one to output.
      (iii) Complementing every digit in turn.
      (iv) Adding one to output
   (b) Which of the following methods are used to represent negative fixed-point binary?
      (i) Using signed-2's complement
      (ii) Using signed-9's complement
      (iii) Using LSB
      (iv) Using MSB
   (c) In order to implement the two arithmetic operations with hardware, which of the following is essential to store numbers?
      (i) Complementer
      (ii) Flip-flop
      (iii) Parallel Adder
      (iv) Registers

9.6 Review Questions

1. "Addition of binary numbers is easy and at the same time quite tedious." Discuss.
2. Discuss the process of subtracting smaller number from larger number.
3. Analyze the subtraction of a larger number from a smaller number.
4. Analyze the algorithms used for addition and subtraction.
5. "In order to implement the two arithmetic operations with hardware, it is essential to store the numbers in registers." Justify.
6. Examine the process involved in the multiplication of binary digits.

Notes

7. "The hardware multiply algorithm is used in digital electronics such as computers to multiply binary digits." Explain.
8. "The Booth's multiplication algorithm involves a few steps." Discuss.

Answers: Self Assessment

1. (a) False (b) True (c) True
2. (a) Two's complement (b) Most Significant Bit (MSB) (c) Booth's multiplication
3. (a) Complementing every digit in turn and adding one to output
   (b) Using signed-2's complement (c) Registers

9.7 Further Readings

Books

Swartzlander, E.; Lemonds, Carl. (2011), Computer Arithmetic: A Complete Reference, 1st ed.

Parhami, Behrooz. (2000), Computer Arithmetic: Algorithms and Hardware Designs, Oxford University Press, New York.

Online links

http://www.quadibloc.com/comp/cp02.htm

http://lapwww.epfl.ch/courses/comparith/Arithm-CRC.pdf

Notes

Unit 10: Computer Arithmetic II

CONTENTS

Objectives
Introduction
10.1 Division Algorithm
10.1.1Divide Overflow
10.1.2 Hardware Algorithm
10.2 Decimal Arithmetic Unit
10.2.1 BCD Adder
10.2.2 BCD Subtraction
10.3 Decimal Arithmetic Operation
10.4 Summary
10.5 Keywords
10.6 Self Assessment
10.7 Review Questions
10.8 Further Readings

Objectives

After studying this unit, you will be able to:

- Explain the division of binary numbers
- Describe decimal arithmetic unit
- Discuss decimal arithmetic operations

Introduction

The processor unit comprises an arithmetic processor to execute arithmetic operations. The designer of the hardware should be aware of the sequence of steps that need to be followed to carry out the arithmetic operations to provide accurate output. This sequence of steps is known as an algorithm. This unit describes various algorithms required to perform arithmetic operations.

Division operation, decimal arithmetic unit, and decimal arithmetic operation are discussed in this unit.

10.1 Division Algorithm

Binary division is similar to division in decimals. The process involves **successive comparison, shifting,** and **subtraction.** Division of binary numbers is easy compared to division of decimal numbers because the quotient is either **0** or **1.** It is also not necessary to check the number of times the dividend (partial remainder) fits into the divisor.

Notes

The following example illustrates the division of binary digits.

Example: Divide **1101** by **11**

```
   100
11) 1 1 0 1   (comparison of partial number with divisor)
   1 1
   0 0 0      (shifting of divisor to right)
     0
       0 1    (subtraction from partial remainder)
         0
         1
```

Task Divide binary number 11011 by 111 using the division algorithm.

10.1.1 Divide Overflow

In a computer system the division operation can lead to a quotient with an overflow because the registers cannot hold a number that exceeds the standard length. To understand this better, consider a system with a standard **5-bit** register. One register is used to hold the divisor and the other to hold the dividend. In case the quotient consists of **6 bits**, **5 bits** of the quotient will be stored in a **5-bit register.** Therefore, the overflow bit needs a **flip-flop** to store the **sixth bit.**

Caution The divide overflow condition should be avoided in normal computer operations. This is because the quotient may be very long and hence, cannot be accommodated in the memory unit. Hence, it is recommended to make provisions in the computer hardware or software to ensure that such conditions are detected.

The divide overflow condition occurs in case the high-order half bits of the dividend comprises a number that is greater than or equal to the divisor. One other point that needs to be considered in division is that, it is advisable to **avoid** division by **zero.** The overflow condition is generally detected when a flip-flop is set. This flip-flop is known as **DVF.**

There are many ways to handle the occurrence of divide overflow. Programmers are sometimes responsible for checking whether the DVF is set after each divide command. After checking for the DVF, programmers can branch to a **subroutine.** The subroutine takes the required corrective measures to avoid the overflow.

Did u know? In a few older computers, the working of the computer used to stop on the occurrence of a divide overflow. This condition was known as divide stop.

Stopping the operation of a computer is not recommended since it is time consuming.

Generally in most of the computers, an interrupt request is provided when a DVF is set. This interrupt causes the computer to cancel the current program and branch to a service routine to take the required corrective measures.

Corrective measure usually refers to removal of the program and display of an error message stating the reason for not completing the program. It is the responsibility of the user of the program to rescale the data or to take corrective measures.

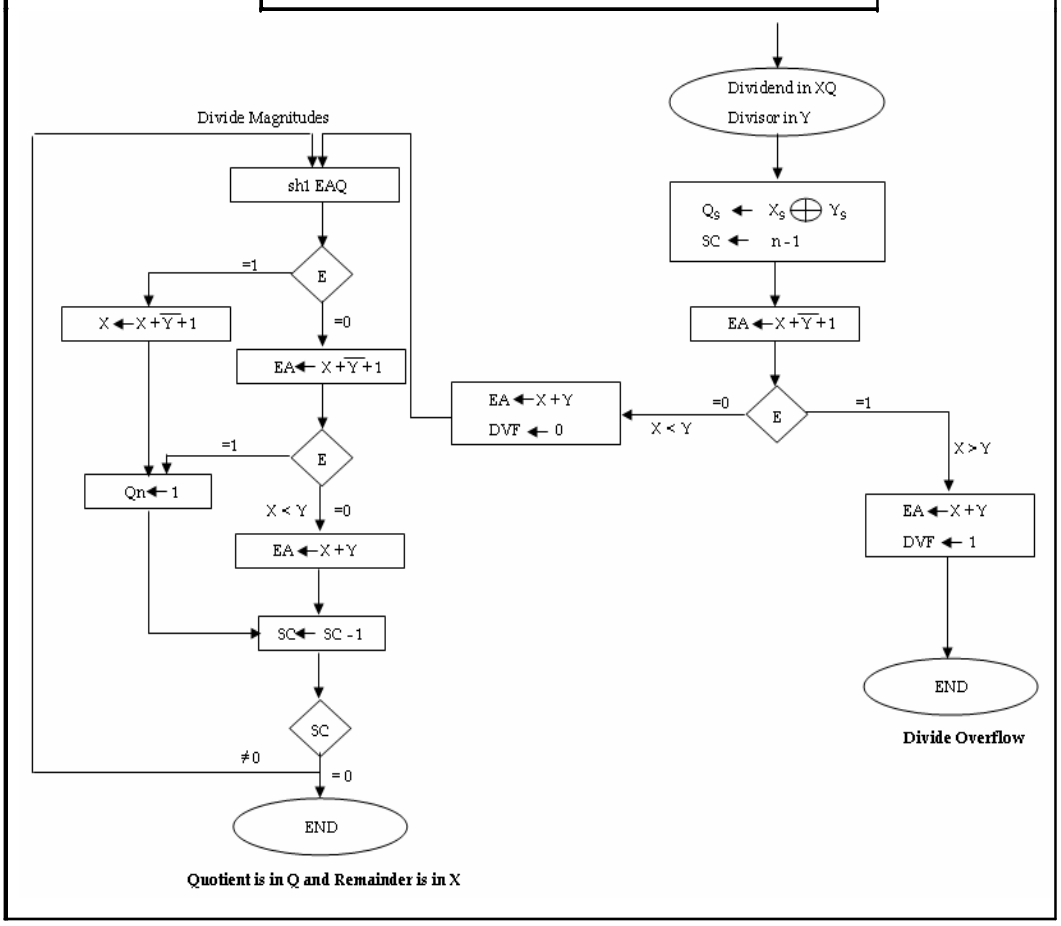
Notes

Notes Use of floating point data is the best way to avoid a divide overflow.

10.1.2 Hardware Algorithm

The hardware divide algorithm can be interpreted easily with the help of a figure. Figure 10.1 illustrates the hardware divide algorithm using a flow chart.

Figure 10.1: Hardware Divide Algorithm Flowchart



In the flowchart shown in figure 10.1, the **dividend** is in **X** and **Q** and the **divisor** is in **Y.** The sign of the resultant is stored in Q_S, which is a part of the quotient. A constant is set into the **SC** to specify the number of bits in the quotient**.** As discussed in the multiplication algorithm, we assume that the operands are transferred from a memory unit to registers. The memory unit comprises words of **n bits.** The sign occupies one bit of the word. The magnitude comprises **n - 1 bits** because it is essential to store an operand with its sign.

The testing of the divide overflow condition takes place by subtracting the divisor present in **Y** from half of the bits of the dividend that is stored in **X.**

1. In case **X ≥ Y,** the DVF is set and an untimely cancellation of the operation takes place.
2. In case **X < Y,** the divide overflow does not occur. Therefore, the value of the dividend is restored by adding **Y** to **X.**

The division of the magnitudes begins by transferring the dividend present in **XQ** to the left and by transferring the high-order bit to **E.** In case the bit shifted into **E** is **1, EA > Y** because **EA**

Notes

comprises a 1 followed by **n - 1 bits** and **Y** comprises only **n - 1 bits.** Therefore, **Y** is subtracted from **EA** and **1** is placed in **QS** for the quotient bit. Because the **Register X** does not have the high-order bit of the dividend, its value is $EA - 2^{n-1}$. If the 2's complement of **Y** is added to this value, the output would be as follows:

$$(EA - 2^{n-1}) + (2^{n-1} - Y) = EA - Y$$

In case **E** should remain as **1,** the carry from the addition should not be transferred to **E.**

In case the shift left operation inserts **0** into **E,** the divisor is subtracted by adding its 2's complement value. The carry has to be transferred to **E.** In case **E = 1,** it means that $X \geq Y$ and as such, Q_S is set to **1.** In case **E = 0,** it means that $X < Y$ and therefore, the original number is restored by adding **Y** to **X.** In this case, the **0** that was inserted during shift is left in Q_S.

The above process is repeated with **Register X**, which holds the partial remainder. The quotient magnitude is formed in **Register Q** and the remainder is found in **Register X** after **n - 1** times. The sign of the quotient in Q_S and the sign of remainder in X_S are the same as the original sign of the dividend.

10.2 Decimal Arithmetic Unit

Decimal arithmetic unit refers to a digital function that does decimal micro-operations. This function adds or subtracts decimal numbers by forming **9's** or **10's** complement of the subtrahend. This decimal arithmetic unit first accepts coded decimal numbers and then generates output in the binary form. Since four bits are necessary to represent every coded decimal digit, a single-stage decimal arithmetic unit comprises nine binary input variables and five binary output variables.

Every stage has to comprise four sets of input for the augend digit, four sets of input for the addend digit, and an input-carry. The output comprises four terminals for the sum digit and one for the output-carry.

10.2.1 BCD Adder

BCD adder refers to a 4-bit binary adder that can add two **4-bit** words of **BCD** format. The output of the addition is a BCD-format 4-bit output word, which represents the decimal sum of the addend and augend and a carry that is generated in case this sum exceeds a decimal value of **9.** Therefore, BCD adders can perform decimal addition.

Let us examine an arithmetic operation consisting of two decimal digits in BCD, along with a possible carry from a previous stage. The output of the arithmetic operation cannot be more that **9 + 9 + 1 = 19,** because no input digit exceeds **9.** In case two BCD numbers are applied to a 4-bit binary adder, the adder forms the sum in binary and gives an output ranging from **0** to **19.** Here, **1** refers to an output carry.

Notes

Table 10.1 discusses the construction of a BCD adder, wherein the binary numbers are labeled using symbols **K, Z_8, Z_4, Z_2,** and **Z_1.**

Table 10.1: Construction of BCD Adder (Cont'd)

Sum of Binary Digits					Sum of BCD Digits					
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	Decimal
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

In table 10.1, **K** is the **carry.** The subscripts below the letter **Z** represent the **weights.** The weights, according to the table, are **8, 4, 2,** and **1.** These weights can be allotted to the four bits in BCD code. The first column contains the binary sum as in the outputs of a 4-bit binary adder. The second column contains of the output sum of two decimal numbers that is represented in BCD. Now, a simple rule is essential to convert the binary numbers in the first column to the correct BCD digit representation in the second column.

It is evident from table 10.1 that if the binary sum is less than or equal to **1001,** then the corresponding BCD number is identical and therefore, there is no need for conversion. However, in case the binary sum is more than **1001,** a non valid BCD representation is obtained. Therefore, the binary **6 (0110)** has to be added to the binary sum to convert it to the correct BCD representation and to produce an output carry.

The decimal numbers in BCD are added by employing one 4-bit binary adder and by performing arithmetic operation one digit at a time. To produce a binary sum, first addition is performed on the low-order pair of BCD digits.

In case the output is equal to or greater than **1010,** it can be set right by adding **0110** to the binary sum. This would produce an output-carry automatically for the next pair of significant numbers. Then, the subsequent high-order pair of numbers along with input-carry is added to produce their binary sum. In case this output is greater than or equal to **1010,** it is set right by adding **0110.** This process is repeated until every decimal digit is added.

The entries in table 10.1 help derive the logic circuit that detects the required corrections. When the binary sum has an output carry **K = 1,** a correction is required. The other six combinations starting from **1010** to **1111** that require corrections have a **1** in position **Z_8.** To differentiate them from binary **1000** and **1001,** which also have a **1** in position **Z_8,** it is specified that either Z_4 or Z_2 must have a 1.

Notes

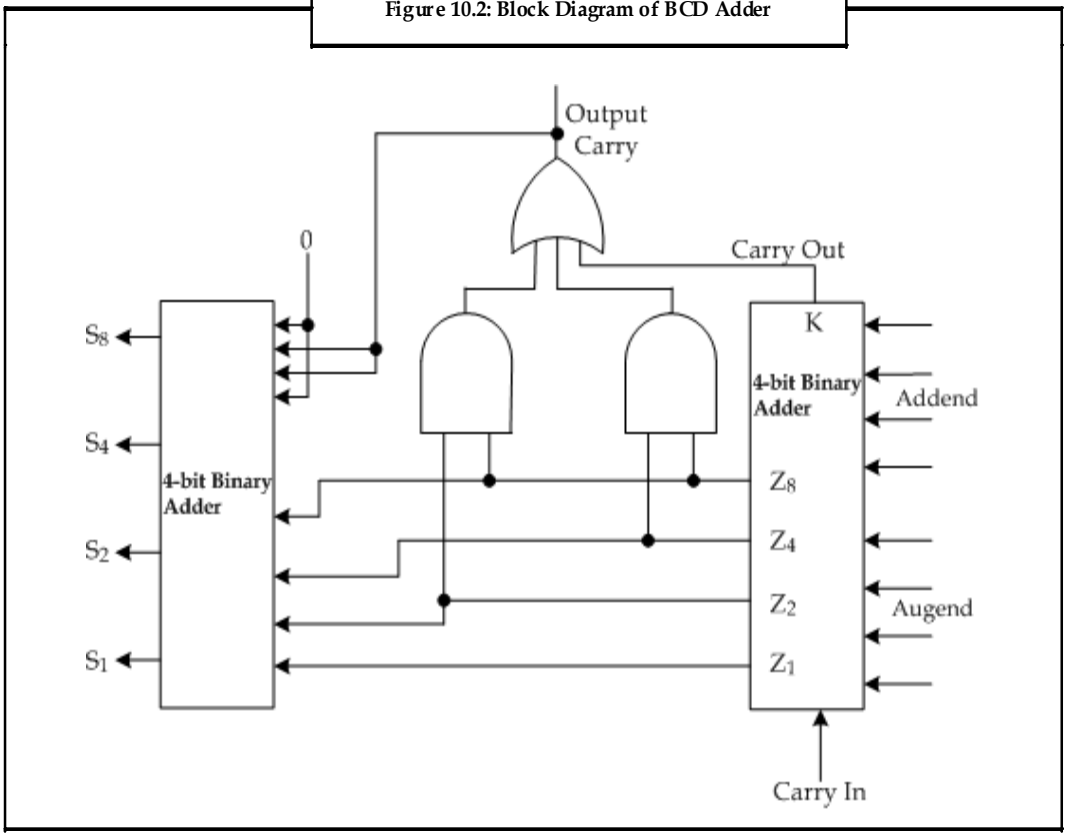
The following Boolean function is used to express the condition for a correction and an output-carry:

$C = K + Z_8Z_4 + Z_8Z_2$

In case **C = 1**, **0110** is added to the binary sum and an output-carry is provided for the next stage.

Figure 10.2 depicts a BCD adder circuit to add two BCD numbers in parallel and to produce a sum digit, which is also in BCD. The internal construction of the BCD adder must include the correction logic.

Figure 10.2: Block Diagram of BCD Adder



As depicted in figure 10.2, a second 4-bit binary adder is used to add **0110.** To produce the binary sum, the two decimal digits along with the input-carry are added in the top 4-bit binary adder.

In case the output-carry is equal to **0,** no binary number is added to the binary sum. However, in case the output-carry is equal to **1,** binary number **0110** is added to the binary sum through the 4-bit binary adder on the left side of figure 6.4. The output-carry that is generated from the binary adder on the left side of figure 6.4 can be ignored, because it provides information that is already present in the output-carry terminal.

A decimal parallel-adder adding **n** decimal numbers requires **n** BCD adder stages along with the output-carry from one stage connected to the input-carry of the next-higher order stage. BCD adders comprise the required circuits for carry look-ahead to achieve shorter propagation delays. It should be noted that the adder circuit for correction may not require all four full-adders. It is also possible to optimize the adder circuits.

10.2.2 BCD Subtraction

A subtractor circuit is required to perform a subtraction operation on two decimal numbers. BCD subtraction is slightly different from BCD addition. Performing subtraction operation by taking the **9's** or **10's** complement of the **subtrahend** and adding it to the **minuend** is economical. It is not possible to obtain the 9's complement by complementing every bit in the code because the BCD

is not a self- complementing code. The 9's complement has to be formed by a circuit that subtracts every BCD number from 9.

The 9's complement of a decimal digit that is represented in BCD can be obtained by complementing the bits in the coded representation of the digit. However, it is essential that a correction is included. There are two methods of correction. They are:

1. ***First Method:*** The binary 1010 is added to every complemented digit. The carry is discarded after performing the addition.
2. ***Second Method:*** The binary 0110 is added before the digit is complemented.

For instance, the 9's complement of BCD **0111** is calculated by complementing every bit to get **1000.** The value **0010** is obtained by adding binary **1010** and ignoring the carry. Using the second method, **0110** and **0111** can be added to obtain **1101.** The required output, that is, **0010** can be obtained by complementing every bit.

Complementing every bit of a 4-bit binary digit **N** is the same as subtracting the digit from **1111.** When the decimal equivalent of **10** is added, the value obtained is **15 - N + 10 = 9 - N + 16.** However, the digit **16** signifies the carry that is discarded, hence, the result equals to **9 - N** as required. Adding and then complementing the binary equivalent of decimal **6** provides **15 - (N + 6) = 9 - N** as needed.

A combination circuit can also be used to obtain the 9's complement of a BCD digit. When this combination circuit is attached to a BCD adder, it results in a **BCD adder or subtractor.**

Consider that the subtrahend digit is denoted by the four binary variables $\mathbf{B_8}$, $\mathbf{B_4}$, $\mathbf{B_2}$ and $\mathbf{B_1}$**.** Also consider **M to be a mode** bit that controls add or subtract operation. Therefore, when **M = 0,** the two digits are added and when **M = 1,** the digits are subtracted.

Consider the binary variables $\mathbf{x_8}$, $\mathbf{x_4}$, $\mathbf{x_2}$ **and** $\mathbf{x_1}$ to be the outputs of the 9's complementer circuit. According to the truth table for circuits:

1. **B1** needs to be complemented at all times.
2. **B2** is the same every time in 9's complement as in original number.
3. **x4** is **1** if the exclusive OR of **B2** and **B4** is **1**.
4. **x8** is **1** if **B8B4B2 = 000**.

For the 9's complement circuit, the Boolean functions are as follows:

1. $x1 = B1M^1 + B^11M$
2. $x2 = B2$
3. $x4 = B4M^1 + (B^14B2 + B4B^12)M$
4. $x8 = B8M^1 + B^18B^14B^12M$

In the above equations it can be observed that **x = B** if **M= 0.** If **M = 1,** the **x** outputs produce the 9's complement of **B.**

10.3 Decimal Arithmetic Operation

Algorithms that are used for arithmetic operations with decimal data and binary data are alike. If the micro-operations symbol is interpreted correctly the same flowchart can be used for both multiplication and division. The decimal numbers in BCD are stored in groups of four bits in the computer registers. When performing decimal micro-operations, every 4-bit group represents a decimal digit and has to be taken as a group.

Notes

For convenience, the same symbol can be used for binary and decimal arithmetic micro-operations with different interpretation. Table 10.2 depicts symbols for decimal arithmetic micro-operations.

Table 10.2: Symbols for Decimal Arithmetic Micro-Operations.

Symbolic Representation	Meaning
X? X +	Adds decimal numbers and transfers the output to X.
$\overline{Y}$	9's complement of Y.
X? X + $\overline{Y}$ + 1	Adds content of X and 10's complement of Y and transfers the output to X.
dshr X	Shifts the decimal number one digit towards right in register X.
d X	Shifts the decimal number one digit towards left in register X.

In table 10.2, we can see **a bar over the symbol** for the **register letter**. This refers to the **9's complement** of decimal number that is stored in the register. When **1** is added to the **9's complement** the **10's complement** is produced. Therefore, the symbol **X ← X+ Y + 1** for decimal digits denotes, transfer of decimal sum that was formed by adding the original content **X** to the **10's complement of Y.** It may be confusing to use similar symbols for **9's complement and 1's complement** in case both types of data are used in the same system. Therefore, it would be better to implement a different symbol for the **9's complement.** In case only one type of data is taken into consideration, the symbol would apply to the type of data used.

Consider that a register **X** holds a decimal **7860** in BCD. The bit pattern of the **12 flip-flops** equal to:

0111 1000 0110 0000.

The micro-operation **dshr** X moves the decimal number one digit to the right to provide **0786.** This shift is over the four bits and as such, changes the content of register to **0000 0111 1000 0110.**

10.4 Summary

- Binary division involves successive comparison, shifting, and subtraction.
- In division, the overflow condition is identified when a flip-flop is set. This flip-flop is named as DVF.
- The interrupt makes the computer to terminate the current program and branch to a service routine that would take the required corrective measures.
- The decimal arithmetic unit is a digital function that does decimal micro-operations, which performs addition or subtraction of decimal numbers by forming the 9's or 10's complement of the subtrahend.
- The decimal arithmetic unit accepts coded decimal numbers and then generates output in binary form.
- BCD adders are used to perform decimal additions. BCD adder is a 4-bit binary adder used to add two 4-bit words having a BCD format.
- It is economical to perform subtraction on two decimal numbers by taking 9's or 10's complement of the subtrahend and adding it to the minuend.
- The 9's complement is formed by a circuit that subtracts every BCD number from 9.
- In decimal arithmetic operation, if micro-operations symbol is interpreted properly, the same flowchart can be used for both multiplication and division.
- When decimal micro-operations are performed each 4-bit group represents a decimal digit and has to be taken as a group.

Notes

10.5 Keywords

BCD: Binary Coded Decimal is a method of representing decimal numbers.

DVF: Divide-Overflow Flip-flop.

Micro-operation: Detailed low-level instructions used in some designs to implement complex machine instructions.

10.6 Self Assessment

1. State whether the following statements are true or false:
   (a) The divide overflow condition occurs in case the high-order half bits of the dividend comprises a number that is equal to the divisor.
   (b) Decimal arithmetic unit refers to a digital function that does decimal micro-operations.
   (c) The 9's complement has to be formed by a circuit that subtracts every BCD number from 10.
   (d) When performing decimal micro-operations, every 4-bit group represents a decimal digit and has to be taken as a group.
2. Fill in the blanks:
   (a) The _____ refers to a 4-bit binary adder that can add two 4-bit words having a BCD (Binary-Coded Decimal) format.
   (b) A combination circuit can also be used to obtain the _____ of a BCD digit.
   (c) The decimal numbers in BCD are stored in groups of _____ bits in the computer registers.
3. Select a suitable choice in every question.
   (a) While performing the division operation, which of the following stores the overflow bit?
      (i) Subroutine
      (ii) Registers
      (iii) Flip-flop
      (iv) Floating point data
   (b) Which of the following symbolic representation shifts the decimal number one digit towards left in register X?
      (i) dshr X
      (ii) dshl X
      (iii) Y
      (iv) $X \leftarrow X + Y$

10.7 Review Questions

1. "In a computer system the division operation could lead to quotient with an overflow, since the registers cannot hold a number that exceeds the standard length." Discuss.
2. Illustrate the division algorithm using a flowchart.
3. "BCD adder refers to a 4-bit binary adder that can add two 4-bit words having a BCD (Binary-Coded Decimal) format." Explain.
4. Analyze the symbols used for decimal arithmetic micro-operations.

Notes

Answers: Self Assessment

1. (a) False (b) True (c) False (d) True
2. (a) BCD adder (b) 9's complement (c) Four
3. (a) Flip-flop (b) dshl X

10.8 Further Readings

Books

Swartzlander, E.; Lemonds, Carl. (2011), Computer Arithmetic: A Complete Reference, 1st ed.

Parhami, Behrooz. (2000), Computer Arithmetic: Algorithms and Hardware Designs, Oxford University Press, New York.

Online links

http://www.quadibloc.com/comp/cp02.htm

http://lapwww.epfl.ch/courses/comparith/Arithm-CRC.pdf

Notes

Unit 11: Input/Output Organization

CONTENTS

Objectives
Introduction
11.1 Peripheral Devices
11.2 I/O Interface
11.3 Data Transfer Schemes
11.4 Program Control
11.5 Interrupts
11.5.1 Interrupt I/O
11.5.2 Enabling and Disabling Interrupts
11.5.3 Program Interrupts
11.5.4 Interrupt Cycle
11.6 Direct Memory Access
11.7 I/O Processor
11.8 Summary
11.9 Keywords
11.10 Self Assessment
11.11 Review Questions
11.12 Further Readings

Objectives

After studying this unit, you will be able to:

- Define peripheral devices
- Explain input/output (I/O) interfaces
- List the data transfer schemes
- Explain the concept of program control
- Describe interrupts
- Explain the concept of DMA transfer
- Define I/O Processors

Introduction

We are aware that computer organization refers to operational units and their interconnections that conform to the architecture specifications. A computer is a complex system that contains millions of elementary electronic components. A computer serves no purpose unless it communicates with the external environment. A user provides instructions to the computer through input devices, and the input data from the user after processing is stored in the memory. The computational results are given to the user through an output device.

Notes

Commercial computers include many types of input and output (I/O) devices. I/O devices are interconnected with the Central Processing Unit (CPU) and the main memory, and each of these devices controls one or more external devices. These devices constitute the Input/Output system.

The I/O devices of a computer provide an efficient mode of communication between the central system and the outside environment. Programs are stored in the computer memory for computation. The results obtained from the computations are displayed to the users. The common means of entering information in a computer is using a keyboard that allows the user to directly enter alphanumeric information.

Did u know? Once a key is pressed, the terminal sends a binary-coded character to the computer.

CPU is a device that is capable of performing computations at a high speed.

The I/O organization of a computer depends on the size of the computer and the I/O devices connected to it. The amount of hardware in the computer available for communicating with the peripheral units helps us know the difference between a small and a large computer. Some techniques involved with peripheral devices are presented in this unit.

11.1 Peripheral Devices

Peripheral devices are devices that are connected either internally or externally to a computer. These devices are commonly used to transfer data. The most common processes that are carried out in a computer are entering data and displaying processed data. Several devices can be used to receive data and display processed data. The devices used to perform these functions are called as peripherals or I/O devices.

Peripherals read information from or write in the memory unit on receiving a command from the CPU. They are considered to be a part of the total computer system. As they require a conversion of signal values, these devices can be referred to as electromechanical and electromagnetic devices. The most common peripherals are printer, scanner, keyboard, mouse, tape device, microphone and external modem that are externally connected to the computer.

The following are some of the commonly used peripherals:

Keyboard

Keyboard is the most commonly used input device. It is used to provide commands to the computer. The commands are usually in the form of text. The keyboard consists of many keys such as function keys, numeric keypad, character keys, and various symbols.

Monitor

The most commonly used output device is the monitor. A cable connects the monitor to the video adapters in the computer's motherboard. These video adapters convert the electrical signals to the text and images that are displayed. The images on the monitor are made of thousands of pixels.

The cursor is the characteristic feature of display devices. It marks the position on the screen where the next character will be inserted.

Printer

Printers provide a permanent record of computer data or text on paper. We can classify printers as impact and non-impact printers. Impact printers print characters due to the physical contact of the print head with the paper. In non-impact printers, there is no physical contact. Some examples of printers are:

1. Daisywheel
2. Dot Matrix
3. Laser Printer

Notes

Let us understand in detail about these printers:

1. ***Daisywheel:*** The daisywheel consists of a wheel with characters placed along its circumference. To print a character, the wheel rotates to the required position, and an energized magnet presses the letter against the ribbon.
2. ***Dot Matrix:*** The dot matrix printer has a set of dots along the printing mechanism. The decision to print a dot depends on the specific characters that are printed on the line.

Example: A 5 X 7 dot matrix printer that prints 80 characters per line has seven horizontal lines, each consisting of 5 X 80 = 400 dots.

3. ***Laser Printer****:* The laser printer imprints the character image by using a rotating photographic drum. The resulting pattern is transferred to paper in the same way as a photocopier.

Daisywheel and dot matrix printers are impact printers. Laser printer is a non-impact printer.

Magnetic Tape

Magnetic tapes are used in most companies to store data files.

Magnetic tapes use a read-write mechanism. The read-write mechanism refers to writing data on or reading data from a magnetic tape. The tapes store the data in a sequential manner. In this sequential processing, the computer must begin searching at the beginning and check each record until the desired data is available. As the tape moves along the stationary read-write mechanism, the access is sequential and the records are accessed one after another. The magnetic tape is the cheapest medium for storage because it can store large number of binary digits, bytes, or frames on every inch of the tape. The advantages of using magnetic tape include unlimited storage, low cost, high data density, rapid transfer rate, portability and ease of use.

Magnetic Disk

Another medium for storing data is magnetic disks. Magnetic disks have high speed rotational surfaces coated with magnetic material. A read-write mechanism is used to achieve the access to write on or read from the magnetic disk. Magnetic disks are mostly used for bulk storage of programs and data.

There are many other peripheral devices found in computer systems such as digital incremental plotters, optical and magnetic readers, analog to digital converters, and various data acquisition equipment.

The following section deals with the method used to transfer information from the computer to the peripherals.

ASCII Alphanumeric Characters

Alphanumeric characters are used for the transfer of information to and from the I/O devices and the computer. American Standard Code for Information Interchange (ASCII) is the standard binary code used to represent alphanumeric characters. This standard uses seven bits to code 128 characters. However, there is an additional bit on the left that is always assigned 0. Therefore, there are 8 bits in total.

Example: Letter A is represented as 01000001 in ASCII.

The ASCIII code consists of 34 nonprinting characters and 94 characters used for various control operations. There are 26 uppercase letters A through Z, 26 lowercase letters a through z, numerals from 0 to 9, and 32 printable characters such as %,*.

Task Write your full name in ASCII using eight bits per character with the leftmost bit always 0. You may check for all ASCII codes online.

The control characters are used to route the data and arrange the printed text into a prescribed format.

Notes

A list of control characters is shown in table 11.1.

Table 11.1: Control Character and Description

Control Character	Description
NUL	Null
SOH	Start of Heading
STX	Start of text
EOT	End of transmission
ENQ	Enquiry
ACK	Acknowledge
DLE	Data Link Escape
ETB	End of transmission block
EM	End of medium

There are three types of control characters. They are:

1. Format Effectors
2. Information Separators
3. Communication Control Characters

The functions of these control characters are:

1. ***Format Effectors:*** They control the layout of printing. They include familiar typewrite controls such as Back Space (BS), Horizontal Tabulation (HT), and Carriage Return (CR).
2. ***Information Separators***: They separate the data into divisions such as paragraphs and pages. They include Record Separator (RS) and File Separator (FS).
3. ***Communication Control:*** They are used during the transmission of text between remote terminals.

Example: STX (start of text) and ETX (end of text) are examples of communication control characters. They are used to frame a text message when they are transmitted through a communication medium.

11.2 I/O Interface

The concept of I/O interface helps us understand how devices intercommunicate. I/O interface provides a method by which information is transferred between internal storage and external I/O devices. All the peripherals connected to a computer require special communication connections for interfacing them with the CPU. The importance of communication connections is to resolve the differences that occur between the computer and each peripheral.

Some of the major differences are:

1. The operation of electromagnetic devices and peripherals are different from the operation of CPU and memory, which are electronic devices. There is a need for conversion of signal values to resolve this difference.
2. The transfer rate of CPU is faster than the transfer rate of the peripherals. Additionally, the peripherals require a synchronization mechanism to transfer the data.
3. Data codes and formats in peripherals differ from the word format in the CPU and memory.
4. Each operating mode is different and each mode must be controlled to prevent disturbing the operation of the peripheral devices connected to the CPU.

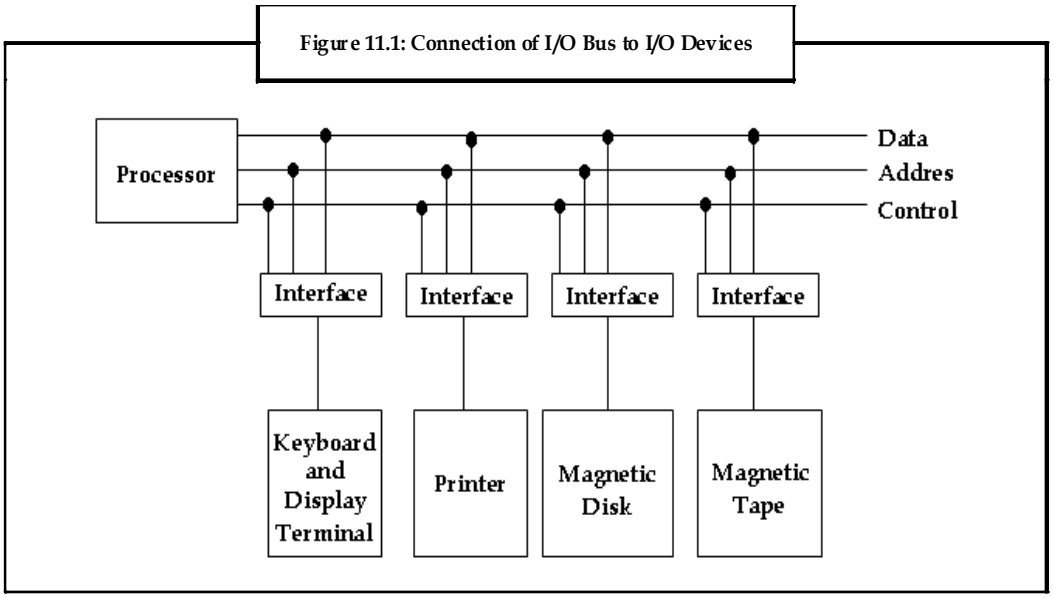
Therefore, computer systems include special hardware components between the CPU and peripherals to resolve these differences. This special hardware supervises and synchronizes all input and output transfers. They are named interface units as they interface the processor bus and

peripheral bus. Each device also has its own controller that supervises the operations of the particular mechanism in the peripheral device.

I/O Bus and Interface Modules

The I/O bus is the pathway used for peripheral devices to communicate with the computer processor. A typical connection of I/O bus to I/O devices is shown in figure 11.1.

Figure 11.1: Connection of I/O Bus to I/O Devices



The I/O bus consists of data lines, address lines, and control lines. In any general purpose computer, the magnetic disk, printer, and keyboard and display terminal are commonly employed. Each peripheral unit has an interface unit associated with it. Each interface decodes the control and address received from the I/O bus. It interprets the address and control received from the peripheral, and provides signals for the peripheral controller. It also supervises the transfer of data between peripheral and processor and also synchronizes the data flow.

Example: Printer can perform actions such as control of paper motion, print timing, and selection of printing characters.

The I/O bus is connected to all peripheral interfaces from the processor. The processor places a device address on the address line to communicate with a particular device. Each interface contains an address decoder attached to the I/O bus that monitors the address lines. When the address is detected by the interface, it activates the path between the bus lines and the device that it controls. The interface disables the peripherals whose address does not correspond to the address in the bus.

At the same time, when the address is made available in the address lines, the processor provides a function code in the control lines. The interface that is selected replies to the function code and executes it. This function code is referred to as an I/O command. This also corresponds to an instruction that is executed in the interface and the peripheral unit connected to that interface. An interface receives any of the following four commands:

1. Control
2. Status
3. Data Output
4. Data Input

Notes

These interface commands are used for the following purposes:

1. ***Control:*** A command control is given to activate the peripheral and to inform its next task. This control command depends on the peripheral, and each peripheral receives its own sequence of control commands, depending on its mode of operation.

   Example: A magnetic tape may be given the instruction to rewind the tape by one record, to rewind the whole tape, or to start the tape moving in the forward direction.

2. ***Status:*** A status command is used to test different test conditions in the interface and the peripheral.

   Example: The computer may require checking the status of the peripheral before a transfer is made. During this transfer, the errors that occur are detected by the interface. These errors are cleared by setting bits in a status register that the processor can read at certain intervals.

3. ***Data Output:*** A data output command makes the interface respond to the command by transferring data from the bus to one of its registers.

   Example: In a tape unit, when a control command is issued, the computer starts the tape to move.

4. ***Data Input:*** The data input command is opposite to the data output command. In data input, the interface receives an item of data from the peripheral and places it in its buffer register. Later, the processor checks if the data is available by means of a status command and then issues a data input command.

I/O Versus Memory Bus

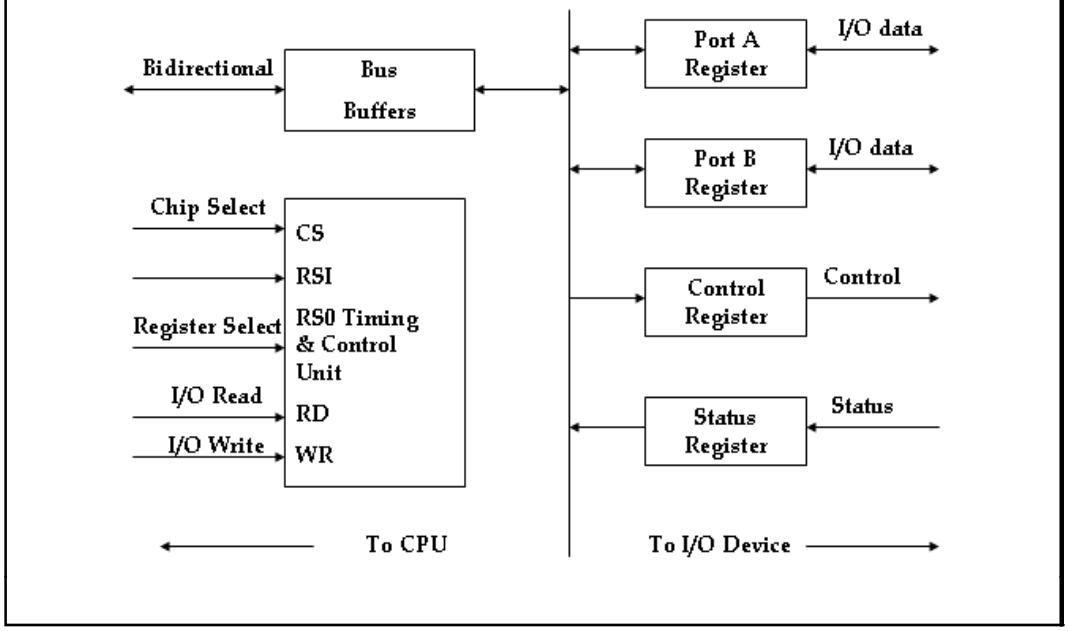
In addition to communicating with I/O devices, the processor must also communicate with the memory unit. Similar to the I/O bus, the memory bus contains independent sets of data, address, and control lines. There are three ways for buses to communicate with memory and I/O. They are:

1. Use two separate buses, one for memory and the other for I/O.
2. Use one common bus for both memory and I/O, but have separate control lines for each.
3. Use one common bus for memory and I/O with common control lines.

Notes

In figure 11.2, an example of an interface unit is shown.

Figure 11.2: Example of an Interface Unit



In figure 11.2, the I/O interface consists of two data registers, a control register, a status register, bus buffers, and timing and control circuits. The interface communicates with the CPU through the data bus. The Chip Select (CS) and the Register Select (RS) inputs determine the address assigned to the interface. The I/O read and I/O write are control lines that specify input and output respectively. There are four registers that communicate with the I/O device attached to the interface.

The I/O data from the device can be transferred to either port A or port B. There is a magnetic disk unit that transfers data in both directions but not at the same time. In this kind of system, the function code in the I/O bus is not needed because control is sent to the control register, status information is received from the status register, and data is transferred between ports A and B.

The control register receives control information from the CPU. The interface and the I/O device attached to it can be placed in various operating modes by loading the corresponding bits into the control register.

Example: Port A is defined as an input port and port B as an output port. A magnetic tape unit is instructed to rewind the tape or to start the tape moving in the forward direction. The bits in the status register are used for status conditions and for recording errors that occur during the data transfer. A status bit represents port A, which received a new data item from the I/O device.

Notes

Table 11.2 shows a list of control characters.

Table 11.2: List of Control Characters

CS	RS_1	RS_0	Register Selected
0	X	X	None
1	0	0	Port A register
1	0	1	Port B register
1	1	0	Control register
1	1	1	Status register

The interface registers communicate with the CPU through the bidirectional data bus. The address bus selects the interface unit through the Chip Select (CS) and the two Register Select (RS_0 and RS_1) inputs. A circuit is provided externally to detect the address assigned to the interface registers. This enables the CS input when the interface is selected. RS_1 and RS_0 are usually connected to the least significant lines of the address bus. These two bits select any of the two inputs listed in table 7.2. The content from the selected register is transferred to the CPU through the data bus when I/O read signal is enabled.

11.3 Data Transfer Schemes

It is important to know the data transfer schemes because they provide an efficient means of transmitting data between the processing unit and the I/O devices. In a computer, the data transfer happens between any of these combinations – CPU and memory, CPU and I/O devices, and memory and I/O devices. A computer is interfaced with many devices of different speeds. Therefore, I/O devices may not be ready to transfer data as soon as the microprocessor issues the instruction for this purpose. Many data transfer schemes have been developed to solve this problem. The data transfer schemes have been broadly classified into two categories:

1. Programmed data transfer schemes
2. Direct Memory Access (DMA) data transfer scheme

Let us now discuss these data transfer schemes.

Programmed Data Transfer Schemes

In programmed data transfer scheme, data transfer takes place between the CPU and I/O device under the control of a program that resides in the memory. In this scheme, the program is executed by the CPU. This scheme is used when a small amount of data is to be transferred. The three important types of programmed data transfer schemes are:

1. ***Synchronous Data Transfer Scheme:*** This type of programmed data transfer scheme is used when the processor and the I/O devices match in speed. Some suitable instructions such as IN and OUT are used for 'to and from' data transfer of I/O devices.
2. ***Asynchronous Data Transfer Scheme:*** This type of programmed data transfer scheme is used when the speeds of I/O devices and the microprocessor do not match and also when the timing characteristics of the I/O devices are not predictable.
3. ***Interrupt Driven Data Transfer Scheme:*** In this programmed data transfer scheme, the processor enables the I/O devices and then continues to execute its original program instead of wasting time on checking the status of the I/O devices. When the I/O devices are ready to send and receive data, then the processor is informed through a specific control line called the 'Interrupt line'.

DMA Data Transfer Scheme

In DMA data transfer, data is directly transferred from the memory to the I/O device or vice versa without going through the microprocessor. This scheme is used when there is a need to transfer bulk data. Transferring bulk data using a microprocessor consumes more time. Therefore, the microprocessor performs the data transfer between an I/O device and memory using this

Notes

DMA technique. For a DMA transfer, I/O devices must also contain an electronic circuitry to generate control signals. But most I/O devices are not equipped with such facilities. Hence, to solve this problem, manufacturers have developed a single chip programmable DMA controller to interface I/O devices with the microprocessor for DMA transfer.

Example: Intel 8237A, 82307 and so on are examples of single chip programmable DMA controller. These are DMA control electronic circuitry that is used to generate control signals for a DMA transfer.

11.4 Program Control

Data transfer instructions are stored in successive memory addresses. When these instructions are processed in the CPU, the data transfer instructions fetch the data from consecutive memory locations and process it. Once the data transfer is complete, the program control moves from the current memory location to the fetch cycle with the program counter containing the address of next instruction to be executed. Sometimes, the program control instruction specifies conditions for altering the content of the program counter while transferring data.

Some of the typical program control instructions are shown in table 11.3:

Table 11.3: Typical Program Control Instructions

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by subtraction)	CMP

The **branch** and **jump** instructions are used interchangeably, but sometimes they are used to denote different addressing modes. **Branch** is usually a one-address instruction. In assembly language, it is represented as ADR for an address. When this instruction is executed, the **branch** instruction transfers the value of ADR into the Program Counter (PC).

Branch and **jump** may be conditional or unconditional. A conditional branch instruction specifies a condition such as branch if positive or branch if zero. If the condition is satisfied, the PC is loaded with branch address and the next instruction is taken from this address. If the condition is not satisfied, the PC is not changed and the next instruction is taken from the next location. On the other hand, an unconditional branch instruction causes a branch to specify an address without any condition.

The **skip** instruction is a zero-address instruction as it does not need an address field. If the condition is met, the conditional skip instruction skips the next instruction. If the condition is not met, control moves to the next instruction where the programmer inserts an unconditional branch instruction.

The **call** and **return** instructions are used with subroutines. The **compare** and **return** instructions do not change the program execution. They are listed in the table because of their setting conditions for subsequent conditional branch instructions. The **compare** instruction subtracts two operands, but the result is not retained. But some status bit conditions are set as a result of the operation. The status bits may be the carry bit, the sign bit, a zero indication, or an overflow condition.

Notes

11.5 Interrupts

Whenever certain condition exists within a program or the computer system, it becomes necessary to have the computer automatically execute one special routine or a collection of special routines.

Example: It is necessary for a computer system to respond to devices such as keyboard, sensor and other components when they request for service.

The common method used to service an interrupt device is the polled approach. In this process, the processor tests each device in sequence and in effect asks each device if it needs to communicate with the processor. However, this method has a decremented effect on the system throughout, thus limiting the tasks and reducing the effectiveness of using such devices.

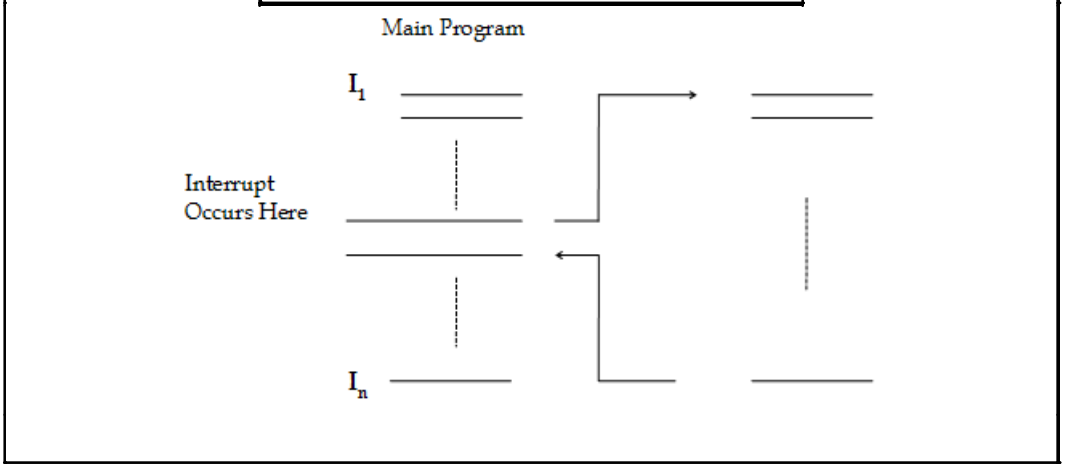
The other desirable approach is the one that allows the processor to execute the main program and stop the execution of the program only to service the peripheral devices when the device itself instructs the processor to do so. Also, the method provides an external asynchronous input that instructs the processor to complete the instruction that is currently being executed. The method also requests to fetch a new routine that will service the requesting device. In this method, the processor resumes program execution from the last instruction executed. This method of servicing the I/O request is called Interrupt driven I/O. System throughput is increased by using this method. The event that causes the interruption is called interrupt and the special routine executed to service the interrupt is called an Interrupt Service Routine (ISR).

There three ways to interrupt a normal program are:

1. By an external signal
2. By a special instruction in the program
3. By the occurrence of some condition

Figure 11.3 shows an interrupt operation.

Figure 11.3: Interrupt Operation



Hardware interrupts are caused by an external signal, whereas software interrupts are caused by special instructions.

Notes

Hardware Interrupt

A hardware interrupt is generated by a hardware device such as a key-press or a mouse-click. The hardware interrupts are classified as:

1. Single level interrupts
2. Multilevel interrupts

Let us now discuss these interrupts.

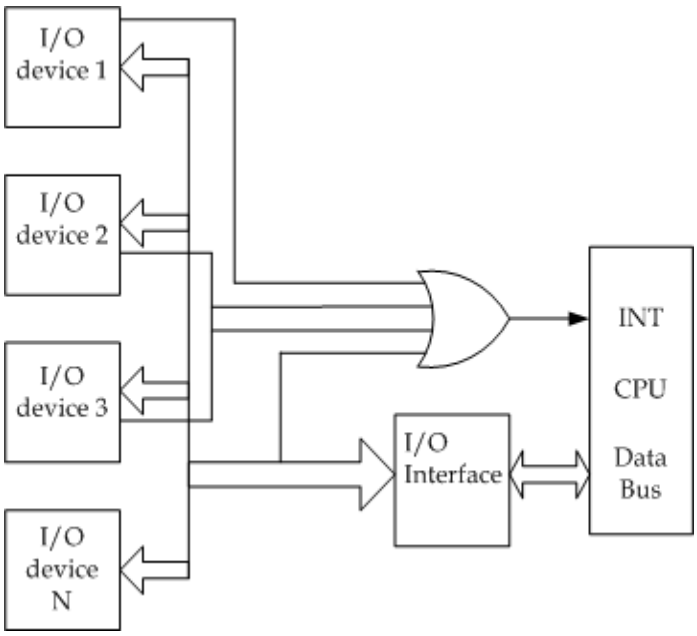
Single Level Interrupts

In single level interrupts, many devices can interrupt the processor at the same time to attend to their requests. But all the devices raise requests through a single input pin of the CPU. When interrupted, the CPU must identify the device that raised the request. Once the I/O port is identified, the CPU attends to the request of I/O device and then continues to carry out the task that it was performing before being interrupted.

In single level interrupts, interrupt requests from all the devices are logically ORed and connected to the interrupt input of the processor. Hence, the interrupt request from any device is routed to the processor interrupt input. After the processor is interrupted, it identifies the requesting device by reading the interrupt status of each device.

Figure 11.4 shows the equivalent circuit of the single interrupt line used in devices. All the I/O devices are connected to INTR through switches to ground. A device closes its associated switch to request an interrupt. When all the interrupt request signals from $INTR_1$ to $INTR_n$ are inactive and all the switches are open, then the voltage on the interrupt request will be equal to V_{DD}. When the device requests an interrupt by closing its switch, then the voltage line drops to 0, causing the interrupt request signal INTR to go to 1. Closing of one or more switching will cause the line voltage to drop to 0. This results in an INTR signal.

Figure 11.4: Single Level Interrupt System



Notes

Multilevel Interrupts

In multilevel interrupts, more than one interrupt pin is present in the processor. Therefore, interrupts can be identified by the CPU on receiving an interrupt request from any of the interrupt pins.

Figure 11.5 shows the multilevel interrupt system.

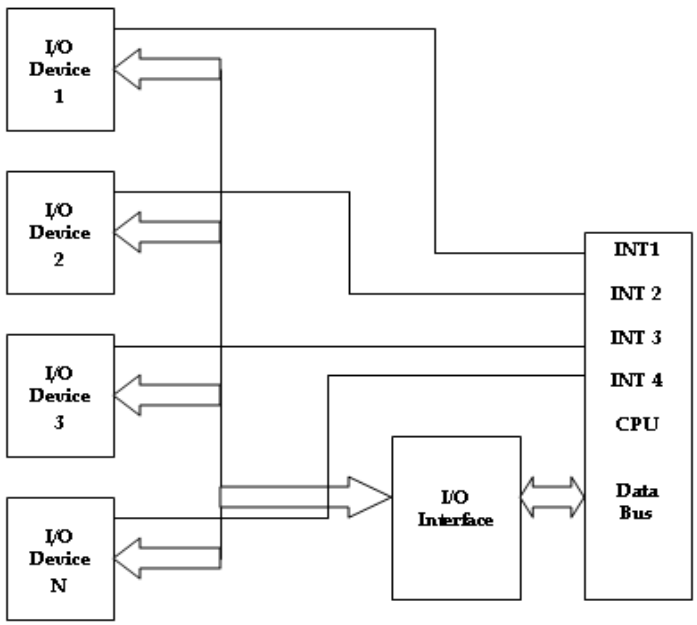
Let us now discuss the process flow of a multilevel interrupt. When the external asynchronous input is inserted, the logic control of a multilevel interrupt is as follows:

1. The processor completes its current instruction.
2. The current contents of the program counter are stored in the stack.

   Caution Remember, during the execution of an instruction, the program counter is pointing to the memory location for the next instruction.

3. The program counter is loaded with the address of an interrupt service routine.
4. Program execution continues with the instruction taken from the memory location pointed by the new program counter contents.
5. The interrupt program is executed until a return instruction is executed.
6. Once the RET instruction receives the old address from the stack, it puts back the address into the program counter. This allows the interrupted program to continue executing at the instruction following the one where it was interrupted.

Figure 11.5: Multilevel Interrupt System



Notes

Notes To handle interrupts from multiple devices, the processor has to perform the following tasks:

1. It has to recognize the device requesting an interrupt.
2. It has to allow the device to interrupt while another interrupt is being serviced.

11.5.1 Interrupt I/O

An interrupt I/O is a process of data transfer in which an external device or a peripheral informs the CPU that it is ready for communication and requests the attention of the CPU.

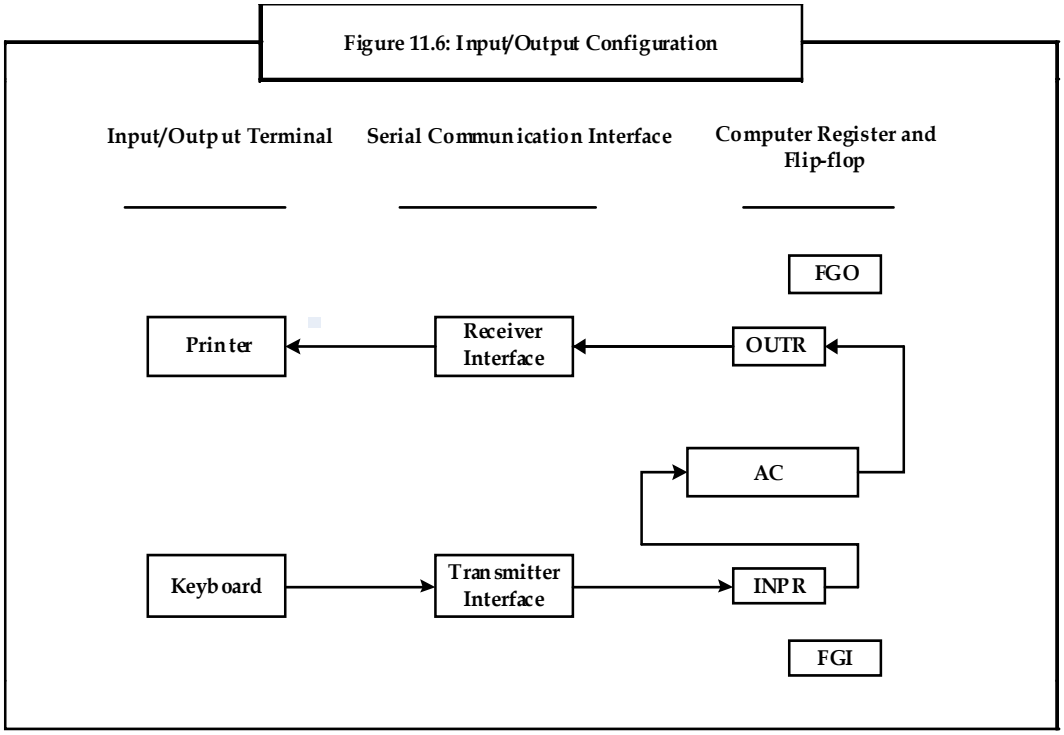
To demonstrate the basic requirements for input and output communication, let us consider an illustration of a terminal unit with a keyboard and printer.

I/O Configuration

The terminals send and receive serial information. Each quantity of serial information has eight bits of alphanumeric code, where the leftmost bit is always 0. The serial information from the input register is shifted into the input register INPR. The output register OUTR is used to store the serial information for the printer. These two registers communicate with the Accumulator (AC) in parallel and with a communication interface in a serial form.

The Input/Output configuration is shown in figure 11.6. The transmitter interface receives serial information from the keyboard and transmits it to INPR. The receiver interface receives information from OUTR and sends it to the printer serially.

Figure 11.6: Input/Output Configuration



The input/output registers consist of eight bits. The FGI is a 1-bit input flag, which is a control flip-flop. The flag bit is set to 1 when new information is available in the input device and is cleared to 0 when the information is accepted by the computer.

The process of information transfer is as follows. Initially, the FGI is cleared to 0. When a key is pressed on the keyboard, the alphanumeric code corresponding to the key is moved to INPR and the input flag FGI is set to 0. The information in INPR cannot be changed as long as the flag is set. The computer checks the flag bit; if it is 1, the information from INPR is transferred in parallel into AC and FGI is cleared to 0.

Notes

The output register OUTR works similar to the input register INPR.

The flow of information through OUTR is the reverse of INPR. Here, the output flag FGO is set to 1 initially. The computer checks the flag bit; if it is 1, the information from AC is transferred in parallel to OUTR and FGO is cleared to 0. New information cannot be loaded into OUTR when the FGO is 0 because this condition indicates that the output device is in the process of printing a character.

Input Register

The INPR input register is a register that consists of eight bits and holds alphanumeric input information. The 1-bit input flag FGI is a control flip-flop. When new information is available in the input device, the flag bit is set to 1. It is cleared to 0 when the information is accepted by the computer. The flag is required to synchronize the timing rate difference between the input device and the computer.

The process of information transfer is as follows:

1. The input flag FGI is set to 0. When a user presses any key on the keyboard, an 8-bit alphanumeric code is moved into INPR and the input flag FGI is set to 1.
2. The computer checks the flag bit. If the bit is 1, then the information from INPR is transferred to AC and simultaneously FGI is cleared to 0.
3. After the flag is cleared, new information can be shifted into INPR by entering another key.

Output Register

The working of the output register OUTR is similar to that of the input register INPR, but the direction of information flow is in reverse.

The process of information transfer is as follows:

1. The output flag FGO is set to 1.
2. The computer checks the flag bit. If the bit is 1, the information from AC is transferred to OUTR and simultaneously FGO is cleared to 0.
3. Then the output device accepts the coded 8-bit information and prints the corresponding character.
4. After this operation is complete, the output device sets the FGO to 1.

11.5.2 Enabling and Disabling Interrupts

Interrupts can be classified as maskable and non-maskable interrupts.

1. ***Maskable Interrupt:*** It is a hardware interrupt that can be ignored.
2. ***Non-maskable Interrupt:*** It is a hardware interrupt that cannot be ignored. It is usually signaled when a non-recoverable hardware error occurs.

Interrupts can be masked or unmasked by setting the particular flip-flop in the processor. When the interrupt is masked, processor does not respond even though the interrupt is activated. These days most of the processors provide this masking facility.

These interrupts can be enabled and disabled under program control. The enabling and disabling of interrupts can be done using an Interrupt Flag (IF). The IF controls the external interrupts that are signaled through the INTR pin. When IF=0, INTR interrupts are restrained; when IF=1, INTR interrupts are enabled.

Notes

11.5.3 Program Interrupts

A program interrupt is an interrupt that is initiated by an internal or an external signal rather than from the execution of an instruction.

There are three types of program interrupts, namely:

1. Internal Interrupt
2. External Interrupt
3. Software Interrupt

1. ***Internal Interrupts:*** These are often referred to as traps. These interrupts occur when an instruction is terminated before the time allocated for it is over.

Example: Whenever we try to divide a number by '0', an error occurs during the execution. The execution is terminated there and an error message pops up on the screen.

The subroutine of the interrupt is responsible for the corrective measures that process the internal interrupt.

2. ***External Interrupts:*** These are caused by I/O devices and peripherals. They are generated when an external device requests for transfer of data and so on. Some of the differences between internal interrupt and external interrupt are shown in table 11.4.

Table 11.4: Differences Between Internal Interrupt and External Interrupt

Internal Interrupt	External Interrupt
Internal interrupts are synchronous with the program.	External interrupts are not synchronous with the program.
Internal interrupts are generated due to some condition violated by the main program.	External interrupts are generated by peripherals.
Internal interrupts are generated at the same place each time.	External interrupts do not depend on re-execution of the same.

3. ***Software Interrupts:*** These interrupts are not initiated by any hardware; instead they are generated when appropriate instructions are executed.

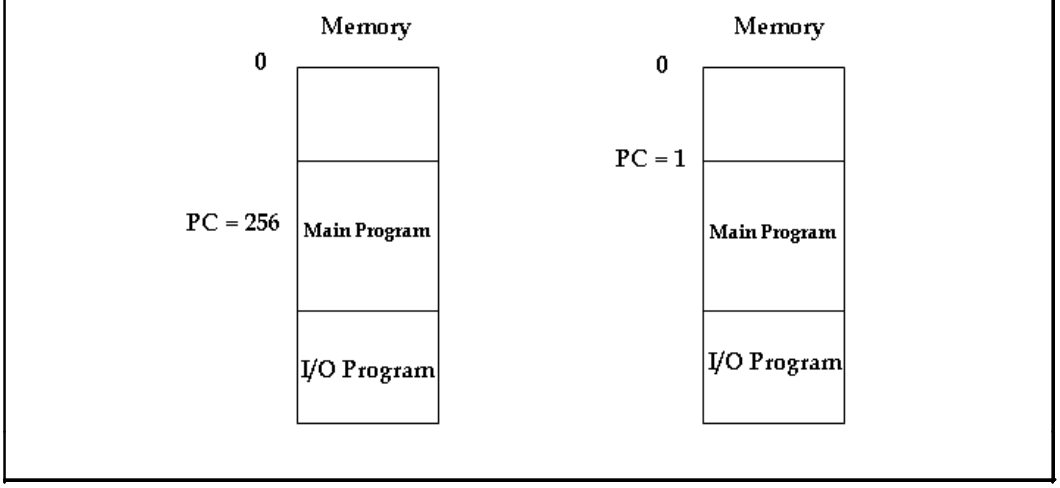
11.5.4 Interrupt Cycle

The interrupt cycle is a hardware implementation of a branch and save return address operation. The available return address is stored in PC in a specific location where it can be taken later when the program returns to the instruction at which it was interrupted. The location may be a processor, a register, a memory stack, or a specific memory location. The memory location is chosen at address 0 for storing the return address. After this, control inserts address 1 into PC and clears IEN and R so that no more interruptions occur.

Notes

A simple example of the interrupt cycle process is shown in figure 11.7.

Figure 11.7: Interrupt Cycle



Let us assume that an interrupt occurs and R is set to 1 when the control is executing the instruction at address 255. At this instant, the return address 256 is in PC. The programmer had previously placed an I/O service program in memory from address 1120.

When the control reaches the timing signal T_0 and finds that R = 1, it continues with the interrupt cycle. Then the content of PC (256) is stored in memory location 0, PC is set to 1, and R is cleared to 0. But for the next instruction cycle, the instruction is read from address 1 as it is the content of PC. Next, the branch instruction at address 1 causes the program to transfer the service to 1120. Once this is completed, ION instruction is executed to set IEN to 1. Then the program returns to the location where it was interrupted.

Later, the instruction that returns the computer to the initial place is the branch indirect function with an address part of 0. Next, the control goes to the indirect phase to read the effective address. The effective address is in location 0 and contains the return address that was stored there during the previous interrupt cycle.

11.6 Direct Memory Access

Direct Memory Access (DMA) is a hardware controlled data transfer technique. An external device is used to control data transfer. External device generates address and control signals that are required to control data transfer. External devices also allow peripheral devices to directly access memory. The external device which controls the data transfer is called the DMA controller.

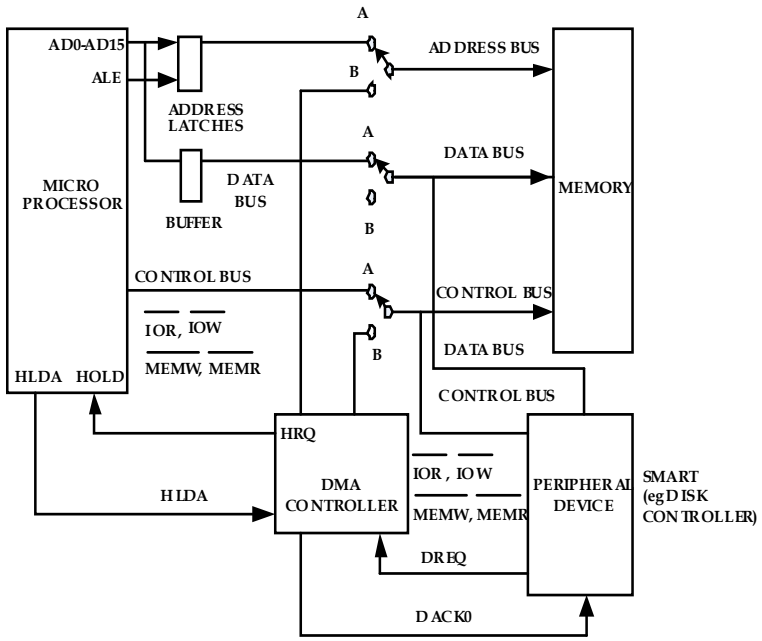
DMA Idle Cycle

When the system is turned on, the switches are in position A. The processor starts executing the program until it needs to read a block of data from the disk. The disk processor sends a series of commands to the disk controller to search and read the desired block of data from the disk. When the disk controller is ready to send the data from the disk, it sends DMA request (DRQ) signal to the DMA controller. Then the DMA controller sends a HOLD signal to the processor HOLD input. The processor responds to this signal by suspending the buses and sending an HLDA acknowledgement signal. When the DMA controller receives the HLDA signal, it sends a control signal to change the switch position from A to B.

Notes

Figure 11.8 shows how a DMA controller operates in a computer system.

Figure 11.8: DMA Controller Operating in Microprocessor System



DMA Active Cycle

When the DMA controller gets control of the buses, it sends the memory address where the first byte of data from the disk is to be written. It also sends a DMA acknowledge (DACK) signal to the disk controller device signaling it to get ready to send the output byte. Then it asserts both the $\overline{IOR}$ and $\overline{MEMW}$ signals on the control bus. $\overline{IOR}$ enables the disk controller to yield the byte of data from the disk and $\overline{MEMW}$ signal enables the addressed memory to accept data from the data bus.

Cycle Stealing Mode

In this data transfer mode, the device can make only one transfer (byte or word). After each transfer, DMAC gives the control of all buses to the processor. This is a single transfer mode with the process as follows:

1. I/O device asserts DRQ line when it is ready to transfer data.
2. The DMAC asserts HLDA line to request use of the buses from the processor.
3. The processor asserts HLDA, granting the control of buses to the DMAC.
4. The DMAC asserts $\overline{DACK}$ to the requesting I/O device and executes DMA bus cycle, resulting in data transfer.
5. I/O device deasserts its DRQ after data transfer of one byte or word.
6. DMA deasserts $\overline{DACK}$ line.
7. The word/byte transfer count is decremented and the memory address is incremented.
8. The HOLD line is deasserted to give control of all buses back to the processor.
9. HOLD signal is reasserted to request the use of buses when I/O device is ready to transfer another byte or word. The same process is then repeated until the last transfer.

Notes

10. When the transfer count is exhausted, terminal count is generated to indicate the end of the transfer.

11.7 I/O Processor

Sometimes it is necessary for a processor to have some special features like controlling the I/O operations along with the ability to execute I/O instructions according to the requirements. Therefore, it is necessary to study the features of I/O processors (IOPs). An I/O processor is an I/O channel that has a special-purpose processor. This processor has the ability to execute I/O instructions and it can also have control over I/O operation. The I/O instructions are stored in the main memory. The CPU initiates an I/O transfer by instructing the I/O channel to execute an I/O program when I/O transfer is required. The I/O program specifies the area of memory storage priority and action that needs to be taken for some conditions.

Features

1. An IOP can fetch and execute its own instructions.
2. Instructions are specially designed for I/O processing.
3. In addition to data transfer, an IOP can perform arithmetic and logic operations, branches, searches, and translations.
4. IOP performs all work involved in I/O transfer including device setup, programmed I/O, and DMA operation.
5. IOP can transfer data from an 8-bit source to a 16-bit destination and vice versa.
6. Memory-based control blocks are used for communication between IOP and CPU.
7. IOP supports multiprocessing environment. Both IOP and CPU can process simultaneously.

11.8 Summary

- I/O devices enable an efficient mode of communication.
- Peripheral devices are devices that are externally connected to the computer such as the printer and scanner.
- The means of transferring information between internal storage and external storage is provided by I/O interfaces.
- There are mainly two types of data transfer schemes used to transfer data between two devices such as CPU and memory, CPU and I/O devices, and memory and I/O devices.
- Interrupts exist within a program that operates the computer to stop and understand the next action.
- There are two types of hardware interrupts, namely single level interrupt and multilevel interrupt.
- An interrupt cycle is a hardware implementation of a branch and save return address operation.
- Direct Memory Access is a hardware controlled data transfer technique.
- I/O processor is an I/O channel that has a special purpose processor.

11.9 Keywords

Deassert: Set a signal to its inactive state

Microprocessor: It is an integrated circuit semiconductor chip that performs the bulk of the processing and controls the different parts of a system.

Program Counter (PC): It holds the address of instruction.

Synchronous: Occurs at the same time.

Notes

11.10 Self Assessment

1. State whether the following sentences are true or false:
   (a) In cycle stealing data transfer mode, the device can make only two transfers.
   (b) After execution the current instruction, the CPU serves an interrupt depending on the device of the interrupt.
   (c) The compare and return instructions do not change the program execution.
   (d) The working of the output register is similar to that of the input register, and the direction of the flow is also similar.
   (e) The jump instruction is a zero-address instruction because it does not need an address field.
2. Fill in the blanks:
   (a) The _____ consists of a wheel with the characters placed along the circumference.
   (b) The _____ places a device address on the address line to communicate with a particular device.
   (c) Data are directly transferred from memory to the I/O device or vice versa without going through the microprocessor in _____ data transfer scheme.
   (d) The call and return instructions are used with _____
   (e) The _____ is a hardware implementation of a branch and save return address operation.
3. Select a suitable choice for every question:
   (a) What type of information is entered in computer by means of a keyboard?
      (i) Alphanumeric
      (ii) Beta numeric
      (iii) Binary information
      (iv) Octal information
   (b) Which is the most commonly used video monitor?
      (i) Liquid Crystal Display (LCD)
      (ii) Cathode Ray Tube (CRT)
      (iii) Light Emitting Diode (LED)
      (iv) Plasma Display
   (c) Which command is used to test different test conditions in the interface and the peripheral?
      (i) Data output
      (ii) Status
      (iii) Control
      (iv) Data input
   (d) Which type of instruction occurs when an instruction is terminated before the time allocated for it is over?
      (i) External interrupt (ii) Software interrupt
      (iii) Internal interrupt (iv) Hardware interrupt

Notes

11.11 Review Questions

1. "The most commonly used peripherals are video monitors." Elaborate.
2. "Input/Output interface provides a way for transferring information between internal storage and external I/O devices." Elaborate.
3. "The I/O bus consists of data lines, address lines, and control lines." Comment.
4. "In a computer, the data transfer happens between two devices such as CPU and memory, CPU and I/O devices, and memory and I/O devices." Justify.
5. "Instructions are stored in successive memory addresses. When these instructions are processed in the CPU, they are fetched from a consecutive memory location and executed." Comment.
6. "The interrupts are classified as single level interrupts and multilevel interrupts." Elaborate
7. "The INPR input register consists of eight bits and holds alphanumeric input information." Comment.
8. "The working of the output register is similar to that of the input register but the direction of information flow is in reverse." Justify.
9. "An interrupt I/O is a process of data transfer through which an external device or a peripheral informs the CPU that it is ready for communication and requests the attention of the CPU." Elaborate.
10. "The interrupt cycle is a hardware implementation of a branch and save return address operation." Comment.
11. "Direct Memory Access is a hardware-controlled data transfer technique." Elaborate.
12. "In cycle stealing data transfer mode, the device can make only one transfer." Comment.

Answers: Self Assessment

1. (a) False (b) False (c) True
   (d) False (e) False
2. (a) Daisywheel (b) Processor (c) Direct Memory Access (DMA)
   (d) Subroutines (e) Interrupt Cycle
3. (a) Alphanumeric (b) Cathode Ray Tube (CRT) (c) Status
   (d) Internal interrupt

11.12 Further Readings

Books

Godse.A.P & GodseRandall Raus D.A. Computer Organization And Architecture, Essentials of computer science 2. Research and education association

Morris Mano. Computer System Architecture, 3rd ed, Pearson Education, Inc.

Online links

http://whatis.techtarget.com/definition/0,,sid9_gci212374,00.html

http://www.google.co.in/#hl=en&biw=986&bih=560&q=Quotes+on+computer+organisation&aq=f&aqi=&aql=&oq=&fp=19f06de9fa513ae8

http://en.wikipedia.org/wiki/Non-maskable_interrupt

Notes

Unit 12: Memory Organization Concepts

CONTENTS

Objectives
Introduction
12.1 Memory Hierarchy
12.2 Main Memory
12.3 Cache
12.3.1 Direct Mapping
12.3.2 Associative Mapping
12.3.3 Set Associative Mapping
12.4 Virtual Memory
12.4.1 Address Space and Memory Space
12.4.2 Address Mapping Using Pages
12.4.3 Associative Memory Page Table
12.4.4 Page Replacement
12.5 Summary
12.6 Keywords
12.7 Self Assessment
12.8 Review Questions
12.9 Further Readings

Objectives

After studying this unit, you will be able to:

- Explain memory hierarchy
- Describe main memory
- Describe cache memory
- Define virtual memory

Introduction

"Ideally one would desire an indefinitely large memory capacity such that any particular word would be immediately available.

We are forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible."

A. W. Burks, H. H. Goldstine, and J. Von Neumann

Computer architects rightly predicted that programmers would want unlimited amounts of memory. A cost-effective way to provide large amounts of memory is by using a memory hierarchy, which takes advantage of locality and performance of memory technologies (such as SRAM, DRAM, and so on). The principle of locality states that most programs do not refer all code or data in a uniform manner. Locality occurs in time and in space. The belief that smaller hardware can be made faster, contributed to the development of memory hierarchies based on different speeds and sizes.

Notes

The significance of memory hierarchy has improved with advances in performance of processors. As the performance of the processor increases, the memory should also support this performance. Therefore, computer architects are working towards minimizing the processor-memory gap.

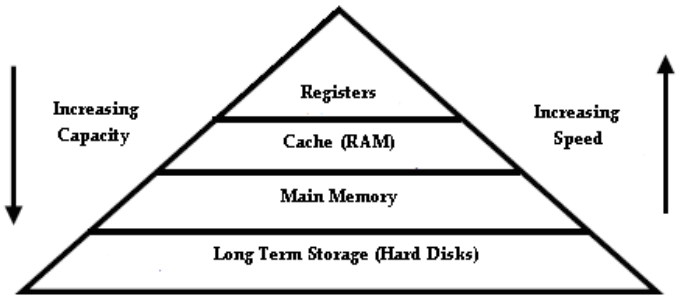
12.1 Memory Hierarchy

You may be aware that the data required to operate a computer is stored in the hard drive. However, other storage methods are also required. This requirement is for a number of reasons, but mostly it is because the hard drive is slow and executing programs using it could be impractical. When the processor requires data or functions, it first fetches the required data from the hard drive and then loads them into the main memory (RAM). This increases the operation speed and programs are executed faster.

Main memory and the hard drive form two levels of the computer's ***memory hierarchy***. In memory hierarchy, the storage devices are arranged in such a way that they take advantage of the characteristics of different storage technologies in order to improve the overall performance of a computer system.

Figure 12.1 shows the standard memory hierarchy of a computer.

Figure 12.1: Representation of Standard Memory Hierarchy



The memory unit is an essential component of any digital computer, because all the programs and data are stored here. A very small computer with a limited purpose may be able to accomplish its intended task without the need for additional storage capacity. Nearly all conventional computers would run more efficiently if they were provided with additional storage space, excluding the capacity of the main memory. It is not possible for one memory unit to accommodate all the programs used in a conventional computer due to lack of storage space. Additionally, almost all computer users collect and continue to amass large amount of data processing software. Not all information that is gathered is needed by the processor at the same time. Thus, it is more appropriate to use low-cost storage devices to serve as a backup for storing the information that is not currently used by the CPU.

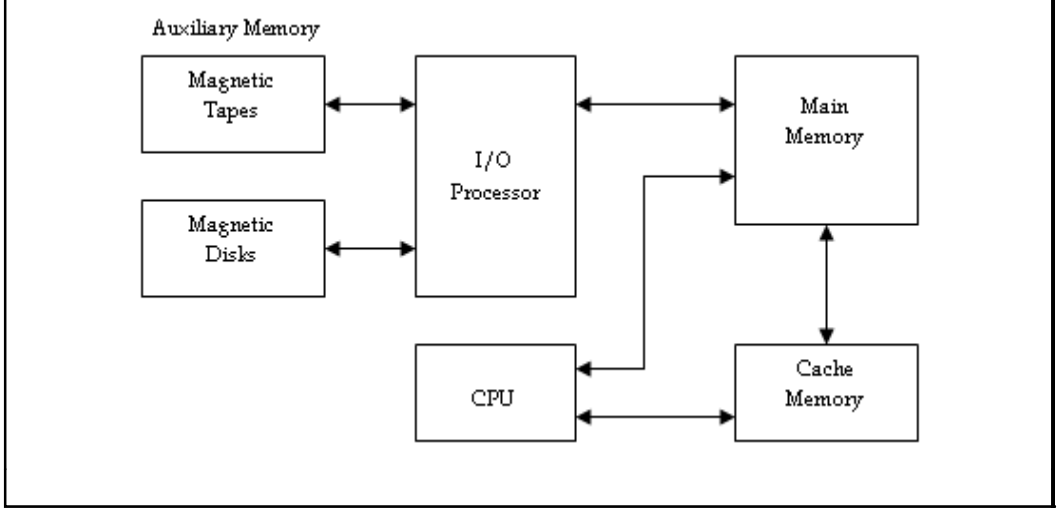
The memory unit that exchanges information directly with the CPU is called the main memory. Devices that support backup storage are referred to as auxiliary memory. The most common devices that support storage in typical computer systems are magnetic disks and magnetic tapes. They are mainly used to store system programs, large data files, and other backup information. Only programs and data that are currently required by the processor reside in the main memory. All other information is stored in auxiliary/secondary memory and moved to the main memory when needed.

The overall memory capacity of a computer can be pictured as being a hierarchy of components. The memory hierarchy system includes all storage devices used in a computer system. They range from the slow but large capacity auxiliary memory to a comparatively faster main memory, to an even smaller but faster cache memory accessible to the high-speed processing logic.

Notes

Figure 12.2 shows the components in a typical memory hierarchy.

Figure 12.2: Memory Hierarchy in a Computer System



Placed at the bottom of the hierarchy is the comparatively slow magnetic tape used to store removable files. Next is the magnetic disk that is used as backup storage.

The main memory takes up a central position because of its ability to communicate directly with the CPU and with auxiliary memory devices, through an Input/Output (I/O) processor. When the CPU needs programs that are not present in the main memory, they are brought in from the auxiliary memory. Programs that are not currently required in the main memory are moved into the auxiliary memory to provide space for currently used programs and data.

To increase the speed of processing, a very-high-speed memory, known as cache, is used. It helps in making the current data and programs available to the CPU at high speed. The cache memory is used in computer systems to compensate for the difference between the main memory access time and processor logic speed. CPU logic is quicker than main memory access time. Yet, the processing speed of the CPU is limited by the main memory's speed. The difference in these operating speeds can be balanced by using an extremely fast, small cache between the CPU and main memory. The cache access time is nearly equal to the processor logic clock cycle time. The cache is used to store the following:

1. Fragments of program that is presently being executed in the CPU
2. Temporary data that is repeatedly needed in the current operation

When programs and data are available at a high rate, it is possible to increase the performance rate of the computer.

Although the I/O processor manages data transfers between main memory and auxiliary memory, the cache organization deals with the transfer of information between the main memory and the CPU. Thus, the cache and the CPU interact at different levels in the memory hierarchy system. The main reason for having two or three levels of memory hierarchy is to achieve cost-effectiveness. As the storage capacity of the memory increases, the cost-per-bit of storing binary information decreases and the memory access time increases. When compared to main memory, the auxiliary memory has a large storage capacity and is relatively inexpensive. However, the auxiliary memory has low access speed. The cache memory is very small, quite expensive, and has very high access speed. Consequently, as the memory access speed increases, so does its relative cost. The main purpose of employing a memory hierarchy is to achieve the optimum average access speed while minimizing the total cost of the entire memory system.

Notes

Did u know? Many operating systems are designed in such a way that they allow the CPU to process a number of independent programs concurrently. This concept, called multiprogramming, refers to the presence of two or more programs in different parts of the memory hierarchy at the same time. In this way, it is possible to make use of the computer by processing several programs in sequence.

Example: Assume that a program is being executed in the CPU and an I/O operation is required. The CPU instructs the I/O processor to start processing the transfer. At this point, the CPU is free to execute another program. In a multiprogramming system, when one program is waiting for an input or output process to take place, there is another program ready to use the CPU for execution.

12.2 Main Memory

The main memory is the fundamental storage unit in a computer system. It is a relatively large and fast memory, and stores programs and data during computer operations. The technology that makes the main memory work is based on semiconductor integrated circuits.

As mentioned earlier, RAM is the main memory. Integrated circuit Random Access Memory (RAM) chips are available in two possible operating modes. They are:

1. ***Static***: It basically consists of internal flip-flops, which store the binary information. The stored information remains valid as long as power is supplied to the unit. The static RAM is simple to use and has shorter read and write cycles.
2. ***Dynamic***: It stores the binary information in the form of electric charges that are applied to capacitors. The capacitors are made available inside the chip by Metal Oxide Semiconductor (MOS) transistors. The stored charge on the capacitors tends to discharge with time and thus, the capacitors must be regularly recharged by refreshing the dynamic memory. Refreshing is done by progressively supplying the capacitor with words every few milliseconds to restore the decaying charge. The dynamic RAM uses minimum power and provides ample storage capacity in a single memory chip.

Notes Metal Oxide Semiconductor (MOS) transistor is the basic building block of almost all modern digital memories, processors and logic chips. MOS transistors find many uses and can function as a switch, an amplifier or a resistor.

Most of the main memory in a computer is typically made up of RAM integrated circuit chips, but a part of the memory may be built with Read Only Memory (ROM) chips. Formerly, RAM was used to refer to a random-access memory. But now it is used to address a read/write memory to distinguish it from a read-only memory, although ROM is also random access. RAM is used for storing the volumes of programs and data that are subject to change. ROM is used for storing programs that permanently reside in the computer. It is also used for storing tables of constants whose value does not change once the computer is constructed.

Notes The ROM part of main memory is used for storing an initial program called a bootstrap loader. The bootstrap loader is a program whose function is to initiate the computer software when power is turned on. As RAM is volatile, its contents are lost when power is turned off. The contents of ROM are not lost even when power is turned off.

RAM and ROM chips are available in a variety of sizes. If the memory requirement for the computer is larger than the capacity of a single chip, a number of chips can be combined to form the required memory size.

Random Access Memory

The term Random Access Memory or RAM is typically used to refer to memory that is easily read from and written to by the microprocessor. In reality, it is not right to use this term. For a memory to be called random access, it should be possible to access any address at any time. This differentiates RAM from storage devices such as tapes or hard drives where the data is accessed sequentially.

Practically, RAM is the main memory of a computer. Its objective is to store data and applications that are currently in use. The operating system controls the usage of this memory. It gives instructions like when the items are to be loaded into RAM, where they are to be located in RAM, and when they need to be removed from RAM. RAM is intended to be very fast both for reading and writing data. RAM also tends to be volatile, that is, all the data is lost as soon as power is cut off.

Read Only Memory

In every computer system, there must be a segment of memory that is stable and unaffected by power loss. This kind of memory is called ***Read Only Memory or ROM***. Once again, this is not the right term. If it was not possible to write to this type of memory, it would not have been possible to store the code or data that is to be contained in it. It simply indicates that without special mechanisms in place, a processor may not be able to write to this type of memory.

Did u know? ROM also stores the computer's BIOS (Basic Input/Output System). BIOS is the code that guides the processor to access its resources when power is turned on, it must be present even when the computer is powered down. The other function of BIOS is storing the code for embedded systems.

Example: It is important for the code in your car's computer to persist even if the battery is disconnected.

There are some categories of ROM that the microprocessor can write to. However, the time taken to write to them, or the programming requirements needed to do so, makes it difficult to write to them regularly. This is why these memories are still considered read only.

There are few situations where the processor cannot write to a ROM under any conditions.

Example: There is no need to modify the code in your car's computer. This ROM is programmed before installing. In order to install a new program in the car's computer, the old ROM is removed and a new ROM is installed in its place.

SRAM

RAMs that are made up of circuits and can preserve the information as long as power is supplied are referred to as Static Random Access Memories (SRAM). Flip-flops form the basic memory elements in a SRAM device. A SRAM consists of an array of flip-flops, one for each bit.

Since SRAM consists of an array of flip-flops, a large number of flip-flops are needed to provide higher capacity memory. Because of this, simpler flip-flop circuits, BJT and MOS transistors are used for SRAM. This helps to save chip area and provides memory integrated circuits at relatively reduced cost, increased speed and reduces the power dissipation as well. SRAMs have very short access times typically less than 10 ns. SRAMs with battery backup are commonly used to provide stability to data during power loss.

Notes

DRAM

SRAMs are faster but their cost is high, because their cells require many transistors. RAMs can be obtained at a lower cost if simpler cells are used. A MOS storage cell based on capacitors can be used to replace the SRAM cells. Such a storage cell cannot preserve the charge (that is, data) indefinitely and must be recharged periodically. Therefore, these cells are called as dynamic storage cells. RAMs using these cells are referred to as Dynamic RAMs or simply DRAMs.

12.3 Cache

Even with improvements in hard drive performance, it is still not practical to execute programs or access data directly from the mechanical devices like hard disk and magnetic tapes, because they are very slow. Therefore, when the processor needs to access data, it is first loaded from the hard drive into the main memory where the higher performance RAM allows fast access to the data. When the processor does not require the data anymore, it can either be discarded or used to update the hard drive.

The cost makes the capacity of a computer's main memory inadequate when compared to its hard drive. However, this would not have a major impact as all of the data on a hard drive need not be accessed all the time by the processor. Only the currently active data or programs need to be in RAM. Additional performance improvements can be achieved by taking this concept to another level.

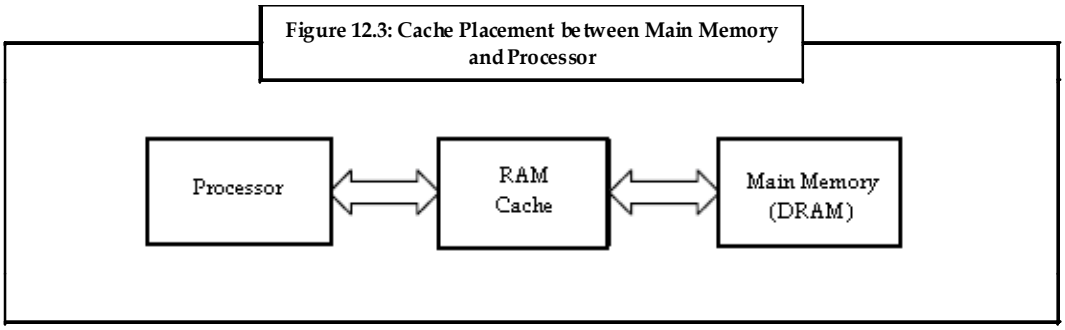
As discussed earlier, there are two main classifications of RAM: Static RAM (SRAM) and Dynamic RAM (DRAM). SRAM is faster, but that speed comes at a high cost - it has a lower density and it is more expensive. Since main memory needs to be relatively large and inexpensive, it is implemented with DRAM.

Main memory improves the performance of the system by loading only the data that is currently required or in use from the hard drive. However, the system performance can be improved considerably by using a small, fast SRAM to store data and code that is in immediate use. The code and data that is not currently needed can be stored in the main memory.

You must be aware that in a programming language, instructions that are executed often tend to be clustered together. This is mainly due to the basic constructs of programming such as loops and subroutines. Therefore, when one instruction is executed, the chances of it or its surrounding instructions being executed again in the near future are high. Over a short interval, a cluster of instructions may execute over and over again. This is referred to as the **principle of locality**. Data also behaves as per this principle as related data is often stored in consecutive locations.

To benefit from this principle, a small, fast SRAM is placed between the processor/CPU and main memory to hold the most recently used instructions/programs and data under the belief that they will most likely be used again soon. This small, fast SRAM is called a **RAM cache** or just a **cache**. The location of a cache in a memory hierarchy is shown in Figure 12.3.

Figure 12.3: Cache Placement between Main Memory and Processor



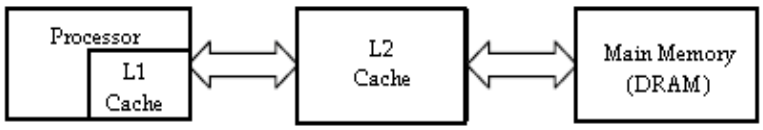
The SRAM of the cache needs to be small, the reason being that the larger address decoder circuits are slower than small address decoder circuits. As the memory increases, the complexity of the address decoder circuit also increases. As the complexity of the address decoder circuit increases, the time taken to select a memory location based on the address it received also increases. For this reason, making a memory smaller makes it faster.

Notes

The concept of reducing the size of memory can be optimized by placing an even smaller SRAM between the cache and the processor, thereby creating two levels of cache. This new cache is usually contained inside of the processor. As the new cache is put inside the processor, the wires connecting the two become very short, and the interface circuitry becomes more closely integrated with that of the processor. These two conditions together with the smaller decoder circuit facilitate faster data access. When two caches are present, the cache within the processor is referred to as a level 1 or L1 cache. The cache between the **L1 cache** and memory is referred to as a level 2 or **L2 cache**.

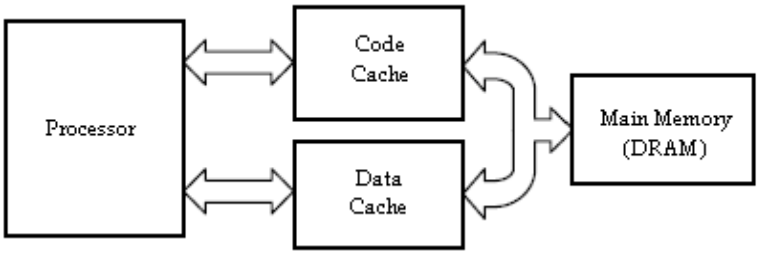
Figure 12.4 shows the placement of L1 and L2 cache in memory.

Figure 12.4: L1 and L2 Cache Placement



The **split cache,** another cache organization, is shown in figure 12.5. Split cache requires two caches. In this case, a processor uses one cache to store code/instructions and a second cache to store data. This cache organization is typically used to support an advanced type of processor architecture such as pipelining. Here, the mechanisms used by the processor to handle the code are so distinct from those used for data that it does not make sense to put both types of information into the same cache.

Figure 12.5: Split Cache Organization



The success of caches depends upon the principle of locality. The principle proposes that when one data item is loaded into a cache, the items close to it in memory should be loaded too.

If a program enters a loop, most of the instructions that are part of that loop are executed multiple times. Therefore, when the first instruction of a loop is being loaded into the cache, it loads its bordering instructions simultaneously to save time. In this way, the processor does not have to wait for the main memory to provide subsequent instructions. As a result of this, caches are organized in such a way that when one piece of data or code is loaded, the block of neighboring items are loaded too. Each block loaded into the cache is identified with a number known as a tag. This tag can be used to find the original addresses of the data in the main memory. Therefore, when the processor is in search of a piece of data or code (hereafter referred to as a word), it only needs to check the tags to see if the word is contained in the cache.

Each block of words and its corresponding tag is combined in the cache to form a **line**. The lines are structured into a table. When the main memory needs a word from within a block, the whole block is moved into one of the lines of the cache along with its tag, which is used to identify the address of the block.

Notes

Cache Operation

The processor first checks the cache when it needs a word from the memory. If the word is not in the cache, a condition known as miss occurs. In order to ensure that no time is lost due to a miss, the process for accessing the same word from main memory is simultaneously activated. If the word is present in the cache, then the processor uses the cache's word and disregards the results from the main memory. This mechanism is referred to as a hit.

In case of a miss, the entire block containing the word is loaded into a line of the cache, and the word is sent to the processor. Depending on the design of the cache/processor interface, the word is either loaded into the cache first and then given to the processor or it is loaded into the cache and sent to the processor simultaneously. In the first case, the cache is controlled by the memory interface and lies between memory and the processor. In the second case, the cache acts like an extra memory on the same bus with the main memory.

Suppose the main memory is divided into ***n*** blocks and the cache has space to accommodate exactly ***m*** blocks. By virtue of the nature of the cache, ***m*** is much smaller than ***n***. If we divide ***m*** into ***n*** , get an integer which illustrates the number of times that the main memory could fill the cache with different blocks from its contents.

Example: Suppose main memory is 128 M (2^{27}) and a block size is four words (2^2), then main memory contains $n = 2^{27-2} = 2^{25}$ blocks. If the cache for this system can hold 256 K (2^{18}) words, then $m = 2^{18-2} = 2^{16}$ blocks. Therefore, the main memory could fill the cache $n/m = 2^{25}/2^{16} = 2^{25-16} = 2^9 = 512$ times.

The main memory is much larger than a cache, so each line in the cache is responsible for storing one of many blocks from main memory. In our above example, each line of the cache is responsible for storing one of 512 different blocks from main memory at a specific time.

The process of transferring data from main memory to cache memory is called as mapping. There are three methods used to map a line in the cache to an address in memory so that the processor can quickly find a word. They are:

1. Direct Mapping
2. Associative Mapping
3. Set Associative Mapping

Caution The only one mistake that can be made in computer design is not having enough address bits for memory addressing and memory management. It is difficult to correct this.

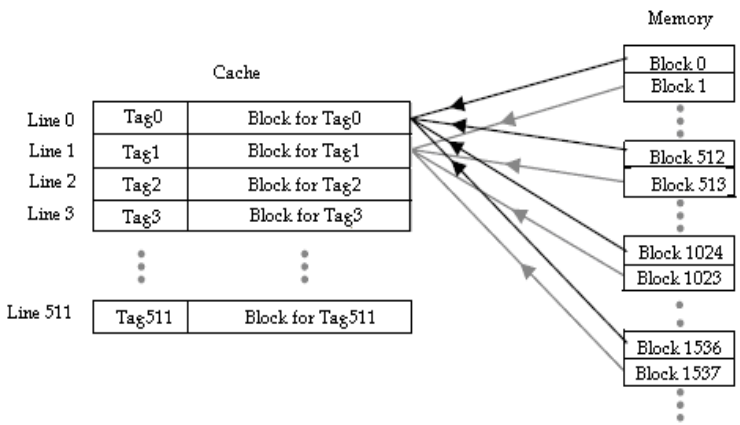
12.3.1 Direct Mapping

Direct mapping is a procedure used to assign each memory block in main memory to a specific line in the cache. If a line is already filled with a memory block and a new block needs to be loaded, then the old block is discarded from the cache.

Notes

Figure 12.6 shows how multiple blocks from the above example are mapped to each line in the cache.

Figure 12.6: Direct Mapping of Main Memory to Cache

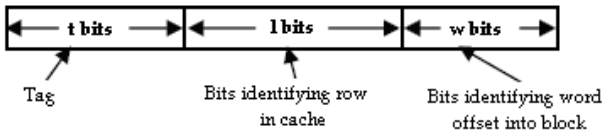


Just like locating a word within a block, bits are taken from the main memory address to uniquely describe the line in the cache where a block can be stored.

Example: Consider a cache with 2^9 = 512 lines, then a line would need 9 bits to be uniquely identified.

Therefore, the 9 bits of the address immediately to the left of the word identification bits would recognize the line in the cache where the block is to be stored. The bits of the address that were not used for the word offset or the cache line would be used for the tag. Figure 12.7 represents this partitioning of the bits.

Figure 12.7: Direct Mapping Partitioning of Memory Address



As soon as the block is stored in the line of the cache, the tag is copied to the tag location of the line. From the cache line number, the tag, and the word position within the block, the original address of the word can be reproduced.

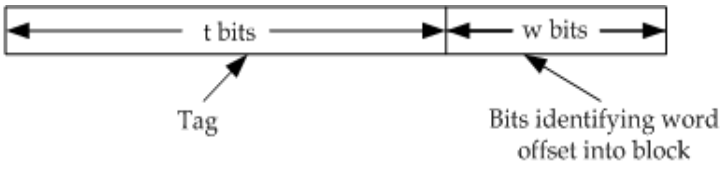
In short, direct mapping divides an address into three parts: **t** tag bits, **l** line bits, and **w** word bits. The word bits are the least significant bits that identify the specific word within a block of memory. The line bits are the next least significant bits that identify the line of the cache in which the block is stored. The remaining bits are stored along with the block as the tag which locates the block's position in the main memory.

Notes

12.3.2 Associative Mapping

Associative mapping or fully associative mapping does not make use of line numbers. It breaks the main memory address into two parts - the word ID and a tag as shown in Figure 12.8. In order to check for a block stored in the memory, the tag is pulled from the memory address and a search is performed through all of the lines of the cache to see if the block is present.

Figure 12.8: Associative Partitioning of Memory Address



This method of searching for a block within a cache appears like it might be a slow process, but it is not. Each line of the cache has its own compare circuitry, which can quickly analyze whether or not the block is contained at that line. With all of the lines performing this comparison process in parallel, the correct line is identified quickly.

This mapping technique is designed to solve a problem that exists with direct mapping where two active blocks of memory could map to the same line of the cache. When this happens, neither block of memory is allowed to stay in the cache as it is replaced quickly by the competing block. This leads to a condition that is referred to as **thrashing.** ***In*** thrashing, a line in the cache goes back and forth between two or more blocks, usually replacing a block even before the processor goes through it. Thrashing can be avoided by allowing a block of memory to map to any line of the cache.

However, this advantage comes with a price. When an associative cache is full and the processor needs to load a new block from memory, a decision has to be made regarding which of the existing blocks should be discarded. The selection method, known as a replacement algorithm, should aim to replace the block least likely to be needed by the processor in the near future.

There are many replacement algorithms, none of which has any precedence over the others. In an attempt to realize the fastest operation, each of these algorithms is implemented in hardware.

1. ***Least Recently Used (LRU):*** This method replaces the block that has not been read by the processor in the longest period of time.
2. ***First In First Out (FIFO):*** This method replaces the block that has been in cache the longest.
3. ***Least Frequently Used (LFU):*** This method replaces the block which has had fewest hits since being loaded into the cache.
4. ***Random:*** This method randomly selects a block to be replaced. Its performance is slightly lower than LRU, FIFO, or LFU.

The objective of a replacement algorithm is to try to remove the page least likely to be referenced in the immediate future.

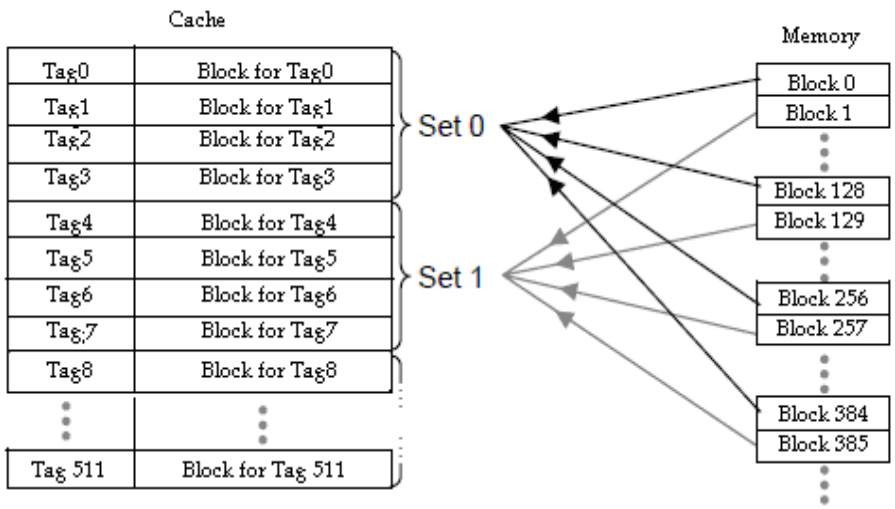
12.3.3 Set Associative Mapping

Set associative mapping merges direct mapping with fully associative mapping by grouping together lines of a cache into sets. The sets are determined using a direct mapping scheme. However, the lines within each set are considered as tiny fully associative cache where any block that is to be stored in the set can be stored to any line within the set.

Notes

Figure 12.9 represents this arrangement using a sample cache that uses four lines to a set.

Figure 12.9: Set Associative Mapping of Main Memory to Cache



A set associative cache that contains *k* lines per set is called as a *k* way set associative cache. Since the mapping technique uses the memory address just like direct mapping does, the number of lines contained in a set must be equal to an integer power of two, for example, two, four, eight, sixteen, and so on.

Example: **Description of set associative mapping**.

Consider a cache with $2^9 = 512$ lines, a block of memory contains $2^3 = 8$ words, and the full memory space contains $2^{30} = 1G$ words. In a direct mapping scheme, this would leave 30 – 9 – 3 = 18 bits for the tag.

Caution Note that the direct mapping method is the same as set associative method where the set size is equal to one line.

By shifting from direct mapping to set associative with a set size of two lines per set, the number of sets obtained equals to half the number of lines. In the instance of the cache having 512 lines, we would obtain 256 sets of two lines each, which would need eight bits from the memory address to identify the set. This would leave 30 – 8 – 3 = 19 bits for the tag. By shifting to four lines per set, the number of sets is reduced to 128 sets requiring 7 bits to identify the set and twenty bits for the tag.

Notes

Every time the number of lines per set in the example is doubled, the number of bits used to identify the set is reduced by one, thus increasing the number of tag bits by one. This is shown in the Figure 12.10.

Figure 12.10: Effect of Cache Set Size on Address Partitioning

Tag bits	Set ID bits	Word ID bits	
18 bits	9 bits	3 bits	Direct Mapping (1 line/set)
19 bits	8 bits	3 bits	2-way Set Associative (2^1 lines/set)
20 bits	7 bits	3 bits	4-way Set Associative (2^2 lines/set)
21 bits	6 bits	3 bits	8-way Set Associative (2^3 lines/set)
⋮	⋮	⋮	
25 bits	2 bits	3 bits	128-way Set Associative (2^7 lines/set)
26 bits	1bit	3 bits	256-way Set Associative (2^8 lines/set)
27 bits		3 bits	Full Associative (1 big set)

Whenever a block from memory has to be stored in a set already filled with other blocks, one of the replacement algorithms described for fully associative mapping is used.

12.4 Virtual Memory

In typical computer systems, data and programs are initially stored in auxiliary memory. Fragments of data or programs are brought into main memory as and when the CPU requires them. Virtual memory is a method or approach used in some large computer systems. It allows the user to construct programs as though a large memory space was available, which is equal to the whole of auxiliary memory. Every address that is referenced by the CPU goes through an address mapping from the so-called virtual address to a physical address in the main memory. Virtual memory is made use of to give programmers the impression that they have a very large memory at their disposition, even though the computer actually has a relatively small main memory. A virtual memory system implements a mechanism that translates program-generated addresses into correct main memory locations. This translation happens dynamically even as programs are being executed in the CPU. The translation or mapping is automatically handled by the hardware by means of a mapping table.

12.4.1 Address Space and Memory Space

Addresses that are used by programmers are called virtual addresses, and the set of such addresses is called the address space. The space or spot where the address is stored in the main memory is called a location or physical address and the set of such locations is called the memory space. Therefore, the address space is the set of addresses generated by programs as they reference instructions and data. The memory space holds the actual main memory locations that are directly addressable for processing. In most computers, the address and memory spaces are the same.

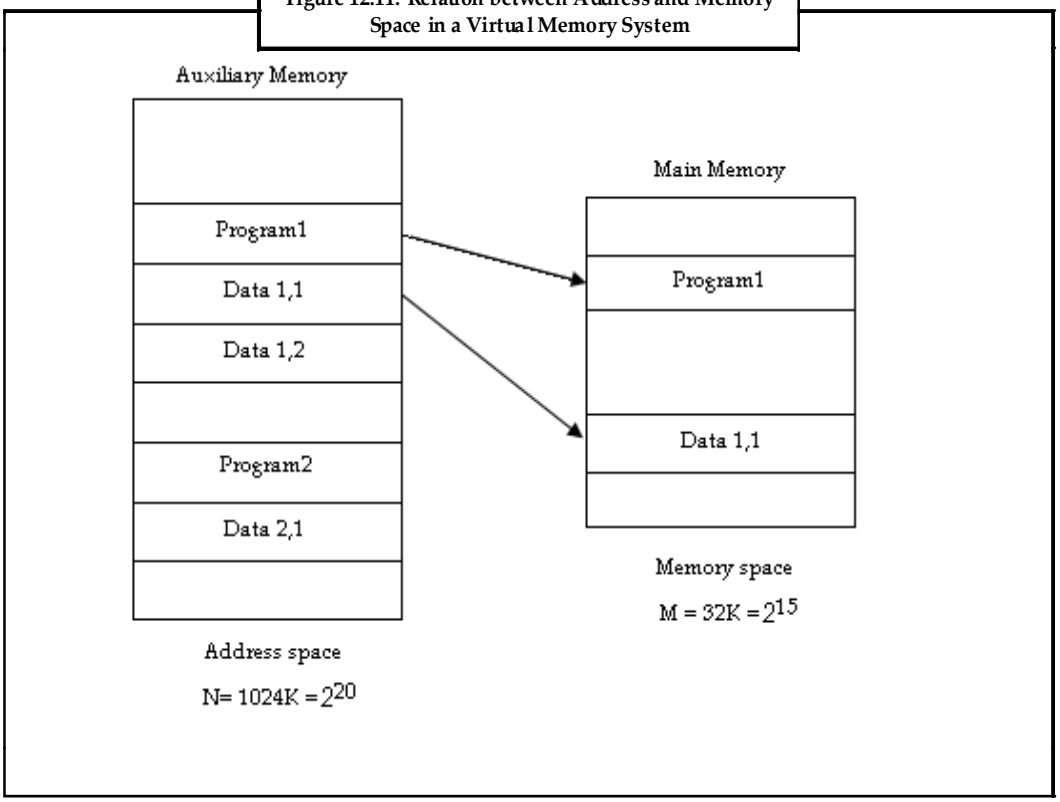
Did u know? The address space is permitted to be bigger than the memory space in computers with virtual memory.

Notes

Example: Consider, main-memory having capacity of 32K words (K = 1024). 15 bits are required to specify a physical address in memory since 32K = 2^{15}. Assuming that the computer has available auxiliary memory for storing 2^{20} = 1024K words. Thus auxiliary memory has a storage capacity equivalent to the capacity of 32 main memories. If the address space is denoted by N and the memory space by M, we then have for this example N = 1024K and M = 32K.

In a multiprogramming computer system, programs and data are transferred to and from auxiliary memory and main memory when required by the CPU. Suppose Program1 is currently being executed in the CPU, Program1 and a section of its associated data are transferred from auxiliary memory into main memory as shown in figure 12.11. The associated programs and data need not be in adjacent locations in the memory, since information is being moved in and out, and empty spaces may be scattered in the memory.

Figure 12.11: Relation between Address and Memory Space in a Virtual Memory System



In a virtual memory system, programmers are made to believe that they have the total address space for their use. Additionally, the address field of the instruction code has a sufficient number of bits to specify all virtual addresses. Suppose, the address field of an instruction code consists of 20 bits, but physical memory addresses can only be specified with 15 bits. As a result, CPU will reference instructions and data with a 20-bit address, but the information at this address must be taken from physical memory because access to auxiliary storage for individual words would be extremely long.

Thus, a table is needed to map a virtual address of say 20 bits to a physical address of say 15 bits. Mapping is a dynamic process, which means that every address is translated instantly as a word is referenced by CPU.

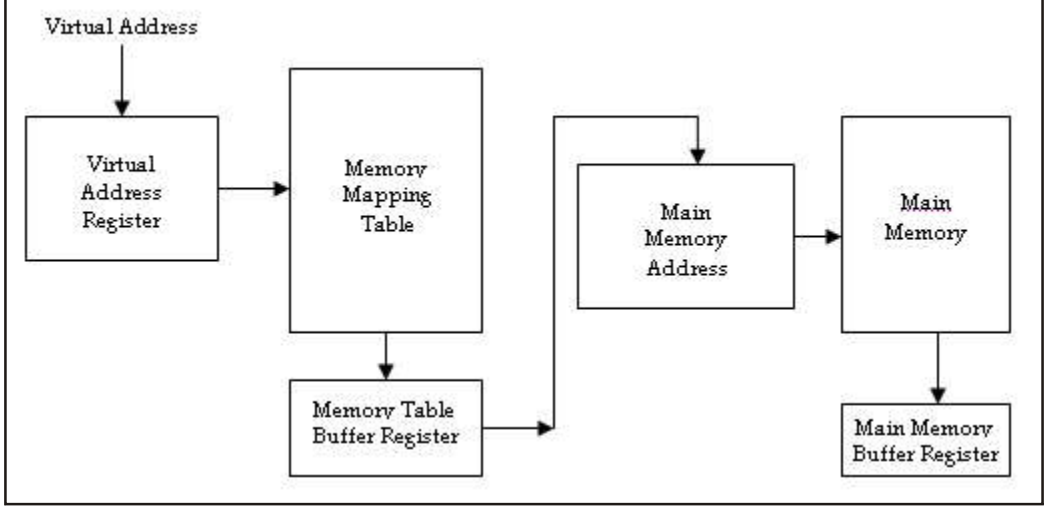
A separate memory or main memory may be used to store the mapping as shown in figure 12.12.

1. In the first case, an additional memory unit is needed along with one extra memory access time.

Notes

2. In the second case, the table takes space from main memory and two accesses to memory are required with the program running at half speed.
3. In the third case, an associative memory can be used. The associative mapping technique is discussed later.

Figure 12.12: Memory Table for Mapping a Virtual Address



In figure 12.12, a virtual address of 20 bits is mapped and listed in the memory mapping table. It is then mapped to a 15 bit physical address which refers to a location in the main memory.

12.4.2 Address Mapping Using Pages

Presenting the address mapping in table form is simplified, if the information in the address space and the memory space are each divided into groups of fixed size. The physical memory is divided into clusters of equal size called ***blocks,*** which may range from 64 to 4096 words each. The term page refers to clusters of address space of the same size.

Example: Suppose a page or block consists of 1K words, then address space can be divided into 1024 pages and main memory can be divided into 32 blocks.

Even though both a page and a block are split into groups of 1K words, a page refers to the cluster of address space, while a block refers to the cluster of memory space. The programs are also split into pages. Segments of programs are moved from auxiliary memory to main memory in records equal to the size of a page. The term page frame is at times used to identify a block.

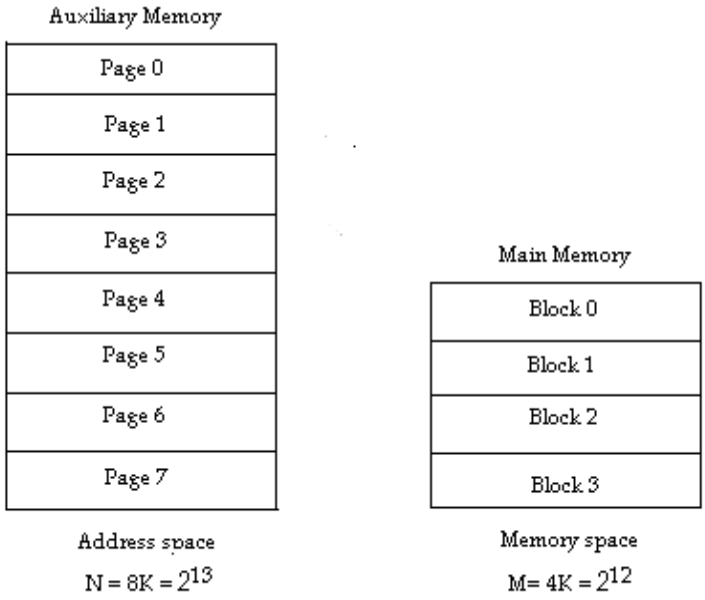
Example: Suppose a computer has an address space of 8K and a memory space of 4K. If they are each split into groups of 1K words, we get eight pages and four blocks as shown in figure 12.13. At any given time, up to four pages of address space may be available in main memory in any one of the four blocks.

The mapping from address space to memory space is made possible if each virtual address is considered to be represented by two numbers - both a page number address and a line within the page. In a computer with 2^p words per page, p bits are used to specify a line address and the remaining high-order bits of the virtual address specify the page number.

Notes

Example: The figure 12.13 shows a virtual address having 13 bits. As each page contains 2^{10} = 1024 words, the higher-order three bits of a virtual address will specify one of the eight pages and the lower-order 10 bits specify the line address within the page.

Figure 12.13: Address Space and Memory Space Split into Groups of 1K Words



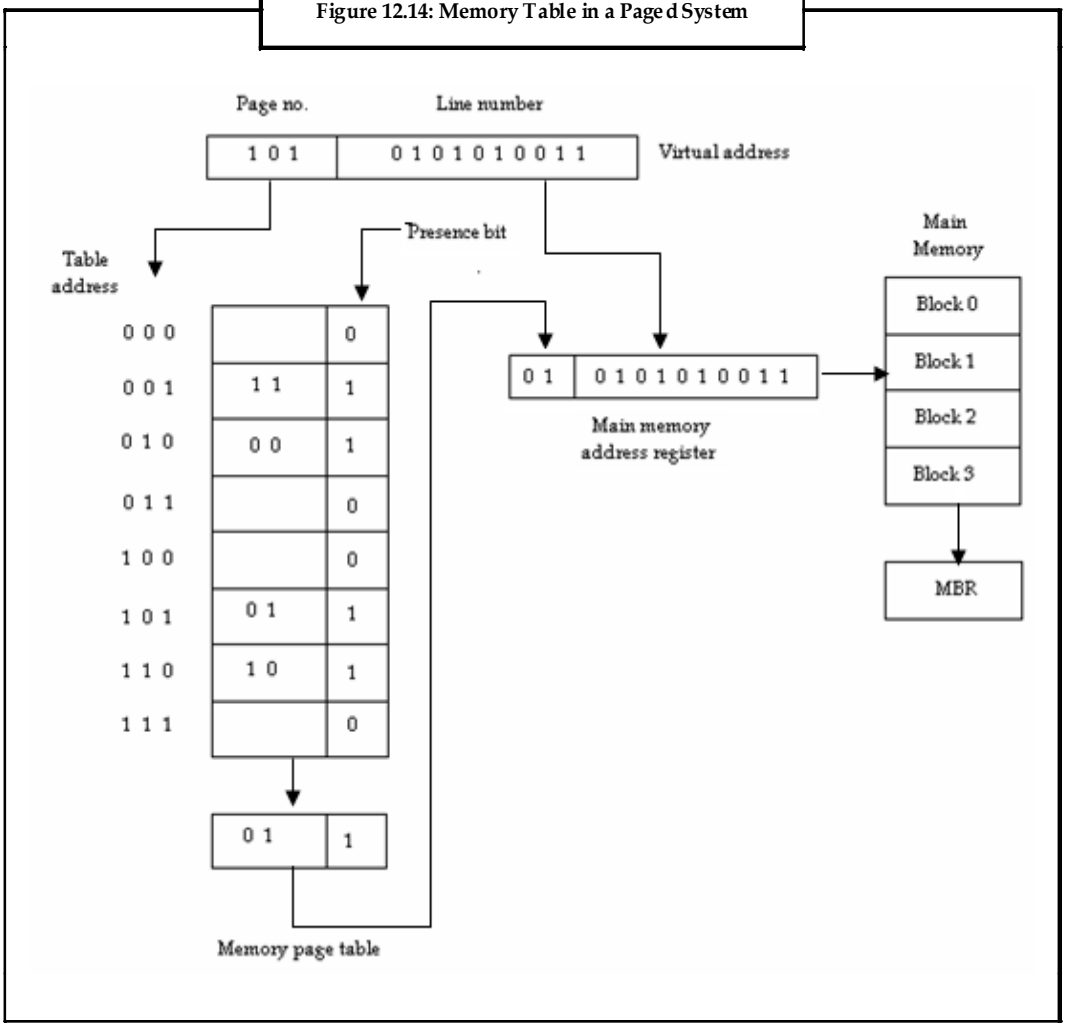
Notes The line address is the same in address space as well as memory space; the only mapping needed is from a page number to a block number.

The structure of the memory mapping table in a paged system is shown in Figure 12.14. The memory-page table comprises eight words, one for each page. The address in the page table denotes the page number and the content of the word gives the block number where that page is stored in main memory. The table shows that pages 1, 2, 5, and 6 are now present in the main memory in blocks 3, 0, 1, and 2, respectively. A presence bit in each location signifies whether the page has been moved from auxiliary memory into main memory. Zero in the presence bit signifies that this page is not available in main memory. The CPU points to a word in memory with a virtual address of 13 bits. The three high-order bits of the virtual address specify a page number and also an address for the memory-page table.

Notes

Figure 12.14 shows memory table in a paged system.

Figure 12.14: Memory Table in a Paged System



The content of the word in the memory page table at the page number address is copied into the memory table buffer register. If the presence bit is 1, the block number thus copied is transferred to the two high-order bits of the main memory address register. The line number from the virtual address is moved into the 10 lower-order bits of the memory address register. A read signal to main memory moves the content of the word to the main memory buffer register that is ready to be used by the CPU. If the presence bit of the word copied from the page table is 0, it indicates that the content of the word referenced by the virtual address is not present in the main memory. A request to the operating system is then generated to get the required page from auxiliary memory and place it into main memory before resuming computation.

12.4.3 Associative Memory Page Table

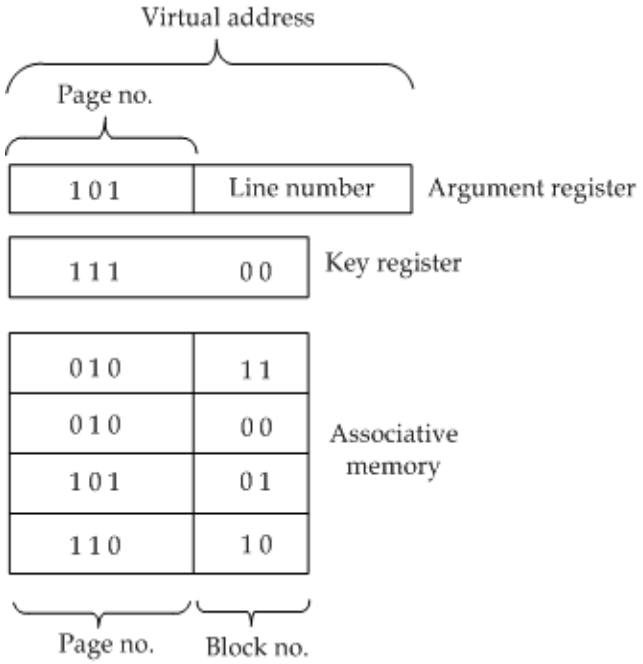
A random-access memory page table is not appropriate when it comes to storage utilization. In the illustration in figure 12.14, we noticed that eight words of memory are needed, one for each page. But at least four words are always marked empty because the main memory cannot accommodate more than four blocks. Normally, a system with *m* blocks and *n* pages would require a memory-page table of *n* locations of which, up to *m* blocks is marked with block numbers and all others are empty.

Notes

Example: Consider a computer with an address space of 1024K words and memory space of 32K words. If each page or block is composed of 1K words, the number of pages will be 1024 and the number of blocks will be 32. The capacity of the memory-page table will be 1024 words and there will be only 32 locations having a presence bit equal to 1. At any given time, at least 992 locations will be free.

A better approach towards organizing the page table would be to construct it with a number of words equal to the number of blocks in the main memory. In this way, the memory size is reduced and each location is fully utilized. This method can be implemented using an associative memory where each word in memory includes a page number together with its related block number. Each word's page field is compared with the page number present in the virtual address and if a match is found, the word is read from memory and its corresponding block number is determined.

Figure 12.15: An Associative Memory Page Table



Let us consider the same case of eight pages and four blocks shown in the example of figure 12.14. If we replace the random access memory-page table with an associative memory of four words as shown in figure 12.15, then each entry in the associative memory array would consist of two fields. By observing the figure, you would find that the first three bits specify a field for storing the page number and the last two bits make up a field for storing the block number. The virtual address is stored in the argument register. The page number bits in the page field of the associative memory and the page numbers in the argument register are matched against each other. If a match is found, the 5-bit word is read out from memory and the related block number is transferred to the main memory address register. If no match occurs, a request to the operating system is generated to fetch the required page from auxiliary memory.

Notes

12.4.4 Page Replacement

A virtual memory organization is a combination of hardware and software systems. To make efficient utilization of memory space all the software operations are handled by the memory management software. The memory management software must decide on the following three things,

1. Which page in main memory must to be removed to create space for a new page?
2. When a new page is to be moved from auxiliary memory to main memory?
3. Where the page is to be located in main memory?

The hardware mapping system and the memory management software together form the architecture of a virtual memory.

When the program execution starts, one or more pages are moved into main memory and the page table is set to indicate their location. The program is executed from main memory until a reference is made for a page that is not in memory. This event is termed as page fault. When page fault occurs, the program that is currently in execution is stopped until the required page is moved into main memory. Since the act of loading a page from auxiliary memory to main memory is basically an I/O operation, the operating system assigns this task to the I/O processor. In this interval, control is transferred to the next program in main memory that is waiting to be processed in the CPU. Soon after the memory block is assigned and then moved, the suspended program can resume execution.

If main memory is full, a new page cannot be moved in. Therefore, it would be necessary to remove a page from a memory block to accommodate the new page. The decision of removing specific pages from memory is determined by the replacement algorithm. The different replacement algorithms have been discussed briefly in the previous sections.

12.5 Summary

- Memory hierarchy is essential in computers as it provides an optimized low-cost memory system.
- Main memory and the hard drive form two levels of the computer's memory hierarchy.
- The main memory plays an important role as it can communicate directly with the CPU and auxiliary memory.
- Most of the main memory in a computer is typically made up of RAM integrated circuit chips, but a part of the memory may be built with ROM chips also.
- SRAM can be used to construct high capacity memory. SRAMs have very short access times.
- Cache memory is built using SRAMs.
- Cache is used to store temporary data that will be used in the immediate future.
- In order to have data that would be needed frequently, a mapping system is employed.
- There are three important methods used to map a line in the cache to an address in memory. They are direct mapping, associative mapping, and set associative mapping.
- Virtual memory is a memory management system that will ensure that the CPU is not left idle at any given time.

12.6 Keywords

Access Time: The time taken to locate the address and perform the transfer.

Argument Register: It is the small amount of storage available on CPU to store arguments.

Memory Access Time: It is the time interval between a memory operation request (read or write) and the time the memory operation completes.

Processor Cycle Time: It is the time taken to perform a basic operation performed by a CPU.

Transfer Rate: Rate at which data is transferred to/from the memory device.

Notes

12.7 Self Assessment

1. State whether the following statements are true or false:
   (a) Main memory and the hard drive form two levels of the computer's memory hierarchy.
   (b) Main memory stores programs and data permanently.
   (c) RAM is the main memory of a computer.
   (d) It is possible to make memory smaller and faster by placing an even smaller SRAM between the cache and the processor.
   (e) A random-access memory page table is not a good choice when it comes to storage utilization.
   (f) Addresses that are used by programmers are called physical addresses.
2. Fill in the blanks:
   (a) To increase the speed of processing, a very-high-speed memory known as_____ is used.
   (b) The _____ processor manages data transfers between main memory and auxiliary memory.
   (c) Flip-flops form the basic memory elements in a _____ device.
   (d) _____ breaks the main memory address into two parts, namely, the word ID and a tag.
   (e) If a reference is made for a page that is not in memory, it results in an event called a _____.
3. Select the suitable choice for every question:
   (a) An essential component of any digital computer is
      (i) Virtual memory (ii) Main memory (iii) Auxiliary memory (iv) I/O processor
   (b) RAM is typically used to refer memory that can
      (i) Only read (ii) Only write (iii) Both read and write (iv) Read but not write
   (c) Over a short interval, a cluster of instructions may execute over and over again. This is called
      (i) Page replacement (ii) LFU (iii) Principle of locality (iv) None of the above
   (d) Identify which of the following is not a replacement algorithm.
      (i) LRU (ii) FIFO (iii) LFU (iv) LIFO
   (e) A better approach towards organizing the page table is by using:
      (i) Set-associative mapping
      (ii) Associative mapping
      (iii) Direct mapping
      (iv) Page replacement

12.8 Review Questions

1. "Not all information that is gathered is needed by the processor at the same time." Explain.
2. "A method that is used to compensate for the difference in operating speeds is to use an extremely fast, small cache between the CPU and main memory." Justify.
3. "The term Random Access Memory is typically used to refer to memory that is easily read from and written to by the microprocessor. In reality, it is not right to use this term." Why?

Notes

4. "Set associative mapping merges direct mapping with fully associative mapping by grouping together lines of a cache into sets." Elaborate.
5. "If a line is already filled with a memory block and a new block needs to be loaded then the old block is discarded from the cache." Explain.
6. "Flip-flops form the basic memory elements in a SRAM device." Why?
7. "In a virtual memory system, programmers are made to believe that they have the total address space for their use." Why and how?
8. "A presence bit in each location signifies whether the page has been moved from auxiliary memory into main memory." Explain with an example.
9. Discuss about address space and memory space.
10. "Even with improvements in hard drive performance, it is still not practical to execute programs or access data directly from these mechanical devices because they are far too slow." How is this drawback overcome?
11. When does a page fault occur?
12. Explain operation of cache.

Answers: Self Assessment

1. (a) True (b) False (c) True (d) True (e) True (f) False
2. (a) Cache (b) I/O processor (c) SRAM
   (d) Associative mapping (e) Page fault
3. (a) Main memory (b) Both read and written (c) Principle of locality
   (d) LIFO (e) Associative mapping

12.9 Further Readings

Books

Mano. M. Computer System Architecture, 3rd ed. Pearson Education, Inc

Null, L., & Lobur, J. (2006). The Essentials of Computer Organization and Architecture. 2nd ed. U.S.A.: Jones and Bartlett Publishers.

Singh. A. K. Digital Principles Foundation of Circuit Design and Application. New Age International Pvt. Ltd

Stallings, W. (2006). Computer Organization and Architecture. Prentice Hall.

Online links

http://www.scribd.com/doc/47360147/CODF-v02b

ftp://ftp.prenhall.com/pub/esm/computer_science.s-041/stallings/COA4e-Notes/Ch4-5.pdf

http://www.csit-sun.pub.ro/courses/cn2/Carte_H&P/H%20and%20P/chapter_5.pdf

Notes

Unit 13: Multiprocessors

CONTENTS

Objectives
Introduction
13.1 Multiprocessors
13.1.1 Coupling of Processors
13.2 Uses of Multiprocessors
13.3 Interconnection Structures
13.3.1 Time-shared Common Bus
13.3.2 Multiport Memory
13.3.3 Crossbar Switch
13.3.4 Multistage Switching Network
13.3.5 Hypercube Interconnection
13.4 Interprocessor Communication and Synchronization
13.5 Summary
13.6 Keywords
13.7 Self Assessment
13.8 Review Questions
13.9 Further Readings

Objectives

After studying this unit, you will be able to:

- Describe the characteristics of multiprocessor
- Discuss the uses of multiprocessors
- Explain interconnection structures
- Explain interprocessor communication and synchronization

Introduction

Multiprocessor is a single computer that has multiple processors. It is possible that the processors in the multiprocessor system can communicate and cooperate at various levels of solving a given problem. The communications between the processors take place by sending messages from one processor to another, or by sharing a common memory.

Both multiprocessors and multicomputer systems share the same fundamental goal, which is to perform the concurrent operations in the system. However, there is a significant difference between multicomputer systems and multiprocessors. The difference exists depending on the extent of resource sharing and cooperation in solving a problem. A multicomputer system includes numerous autonomous computers which may or may not communicate with each other. However, a single operating system that provides communication between processors and their programs on the process, data set, and data element level, controls a multiprocessor system.

Notes

The interprocessor communication is carried out with the help of shared memories or through an interrupt network. Most significantly, a single operating system that provides interactions between processors and their programs at different levels, control the whole system.

Did u know? Processors share access to general sets of memory modules, Input/Output channels, and also peripheral devices. All processors have their individual local memory and Input/Output devices along with shared memory.

13.1 Multiprocessors

Multiprocessor is a data processing system that can execute more than one program or more than one arithmetic operation simultaneously. It is also known as multiprocessing system. Multiprocessor uses with more than one processor and is similar to multiprogramming that allows multiple threads to be used for a single procedure. The term 'multiprocessor' can also be used to describe several separate computers running together. It is also referred to as clustering. A system is called multiprocessor system only if it includes two or more elements that can implement instructions independently. A multiprocessor system employs a distributed approach. In distributed approach, a single processor does not perform a complete task. Instead more than one processor is used to do the subtasks.

Some of the major characteristics of multiprocessors include:

1. ***Parallel Computing***: This involves simultaneous application of multiple processors. These processors are developed using a single architecture in order to execute a common task. In general, processors are identical and they work together in such a way that the users are under the impression that they are the only users of the system. In reality, however, there are many users accessing the system at a given time.
2. ***Distributed Computing:*** This involves the usage of a network of processors. Each processor in this network can be considered as a computer in its own right and have the capability to solve a problem. These processors are heterogeneous, and generally one task is allocated to a single processor.
3. ***Supercomputing:*** This involves usage of the fastest machines to resolve big and computationally complex problems. In the past, supercomputing machines were vector computers but at present, vector or parallel computing is accepted by most of the people.
4. ***Pipelining:*** This is a method wherein a specific task is divided into several subtasks that must be performed in a sequence. The functional units help in performing each subtask. The units are attached in a serial fashion and all the units work simultaneously.
5. ***Vector Computing:*** It involves usage of vector processors, wherein operations such as 'multiplication' is divided into many steps and is then applied to a stream of operands ("vectors").
6. ***Systolic:*** This is similar to pipelining, but units are not arranged in a linear order. The steps in systolic are normally small and more in number and performed in a lockstep manner. This is more frequently applied in special-purpose hardware such as image or signal processors.

A multiprocessor system has the following advantages:

1. It helps to improve the cost or performance ratio of the system.
2. It helps to fit the needs of an application, when several processors are combined. At the same time, a multiprocessor system avoids the expenses of the unnecessary capabilities of a centralized system. However, this system provides room for expansion.
3. It helps to divides the tasks among the modules. If failure happens, it is simple and cheaper to identify and replace the malfunctioning processor, instead of replacing the failing part of complex processor.

4. It helps to improve the reliability of the system. A failure that occurs in any one part of a multiprocessor system has a limited effect on the rest of the system. If error occurs in one processor, a second processor may take up the responsibility of doing the task of the processor in which the error has occurred. This helps in enhancing the reliability of the system at the cost of some loss in the efficiency.

13.1.1 Coupling of Processors

There are two types of multiprocessor systems and they are:

1. ***Tightly-coupled Multiprocessor System***: This system has many CPUs that are attached at the bus level. Tasks and/or processors interact in a highly synchronized manner. The CPUs have access to a central shared memory and communicate through a common shared memory.
2. ***Loosely-coupled Multiprocessor System***: This multiprocessor system is often referred to as clusters. These systems operate based on single or dual processor commodity computers interconnected through a high speed communication system. Tasks or processors do not communicate in a synchronized manner as done in tightly-coupled multiprocessor systems. They communicate through message passing packets. This system has a high overhead for data exchange and uses distributed memory system.

Example: The best example for a loosely-coupled multiprocessor system is a Linux Beowulf cluster. The example for a tightly-coupled multiprocessor system is mainframe system.

Granularity of Parallelism

When you talk about parallelism, you need to know the concept of granularity. The granularity of parallelism specifies the size of the computations that are carried out at the same time between synchronizations. Granularity is referred to as the level to which a system is divided into small parts, either the system itself or its explanation or observation. Granularity of parallelism is of three types. They are:

1. ***Coarse-grain***: A task is divided into a handful of pieces, where each piece is performed with the help of a powerful processor. Processors are heterogeneous. Communication/computation ratio is very high.
2. ***Medium-grain***: A task is divided into tens to few thousands of subtasks. Processors here usually run the same code. Computation ratio is more often hundreds or more.
3. ***Fine-grain:*** A task is divided into thousands to millions of small subtasks that are implemented using very small and simple processors, or through pipelines. Processors have instructions broadcasted to them. The computation ratio is more often 1 or less.

Memory

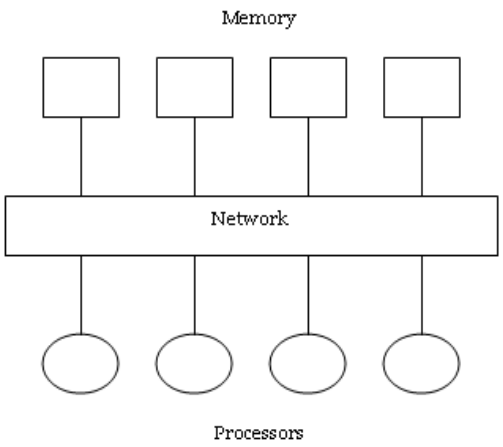
We are aware of the concepts of memory and the memory hierarchy. The different categories of memory discussed in the previous units are main memory, cache memory, and virtual memory. In this section, the different types of memory are listed. They are:

1. ***Shared (Global) Memory:***
   (a) All processors can access a global memory space.
   (b) Processors can also have some local memory.

Notes

Figure 13.1 depicts shared memory.

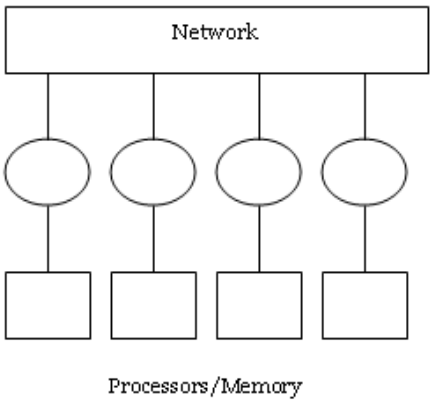
Figure 13.1: Shared Memory



2. ***Distributed (Local, message-passing) Memory*:**
   (a) All the memory units are associated with the processors.
   (b) A message must be sent to another processor's memory to retrieve information from that memory.

Figure 13.2 depicts distributed memory.

Figure 13.2: Distributed Memory



3. ***Uniform Memory:*** Every processor takes the same time to reach all memory locations.
4. ***Non-uniform Memory Access:*** Memory access is not uniform. It is in contrast to the uniform memory.

Notes

Shared Memory Multiprocessors

In shared-memory multiprocessors, there are numerous processors accessing one or more shared memory modules. The processors may be physically connected to the memory modules in many ways, but logically every processor is connected to every memory module.

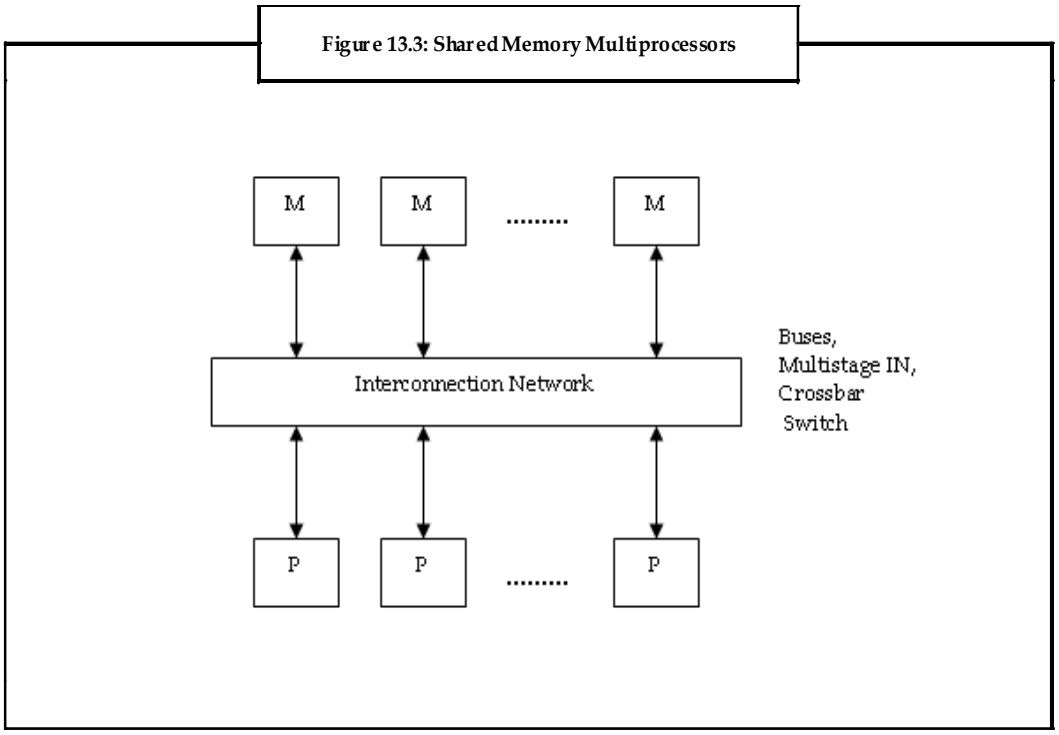
One of the major characteristics of shared memory multiprocessors is that all processors have equally direct access to one large memory address space.

The limitation of shared memory multiprocessors is memory access latency.

Example: Bus and cache-based systems: Encore Multimax, Sequent BalanceMultistage IN-based systems: Ultracomputer, Butterfly, RP3, HEPCrossbar switch-based systems: C mmp, Alliant FX/8

Figure 13.3 depicts shared memory multiprocessors.

Figure 13.3: Shared Memory Multiprocessors



Shared memory multiprocessors have a major benefit over other multiprocessors, because all the processors share the same view of the memory.

These processors are also termed as Uniform Memory Access (UMA) systems. This term denotes that memory is equally accessible to every processor, providing the access at the same performance rate.

Message-Passing Multiprocessors

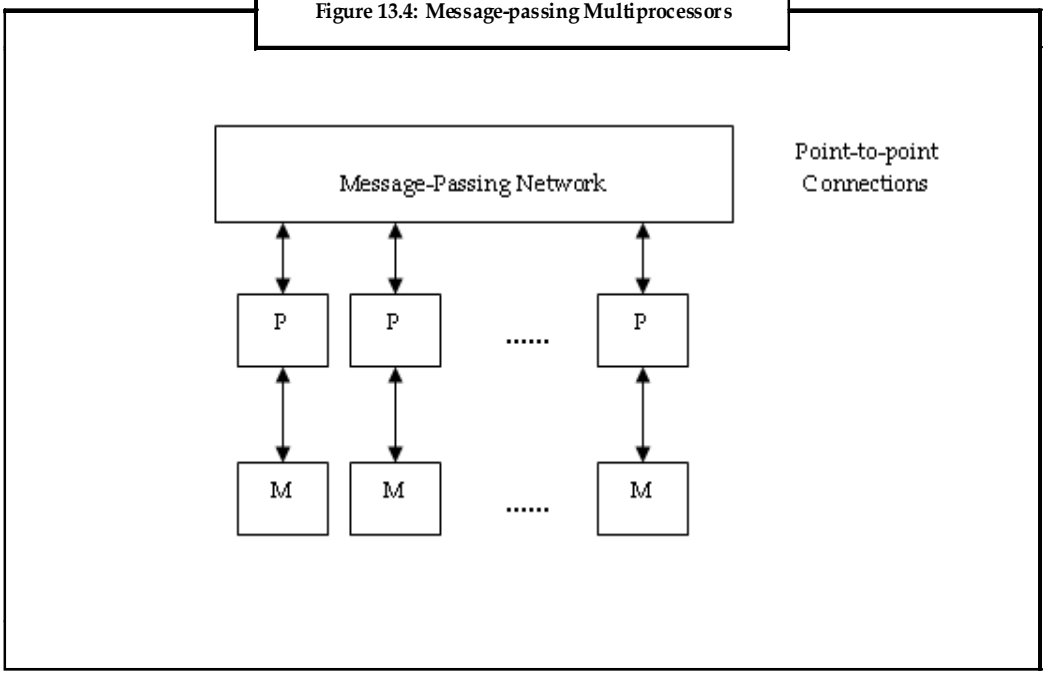
In a message-passing multiprocessor system, a method for conveying messages between nodes, and a node and a method, in order to format the same in a message-passing computer system is specified. Network interface is an example of the message-passing multiprocessor system. In the network interface for a computer system, there exists:

1. Multiple nodes linked with one another through an interconnection network for communication of messages.
2. More than one processor and a local shared memory that are linked with one another through a node bus.

Notes

Figure 13.4 depicts message-passing multiprocessors.

Figure 13.4: Message-passing Multiprocessors



Some of the important characteristics of message-passing multiprocessors are:

1. Computers are interconnected.
2. All processors have their own memory and they communicate through message -passing.

Example: Tree structure: Teradata, DADO

Mesh-connected: Rediflow, Series 2010, J-MachineHypercube: Cosmic Cube, iPSC, NCUBE, FPS T Series, Mark III

Limitations of message-passing multiprocessors are communication overhead and difficulty in programming.

13.2 Uses of Multiprocessors

Use of multiprocessor systems in real-time applications is becoming popular. One of the major reasons for this popularity is the recent drop in the cost of these systems. At present, dual processor machines are available at fifty to sixty thousand rupees, and it is predicted that the prices are going to drop even further. The faster response time and fault-tolerance feature of such systems are the other reasons that attract real-time system developers to install multiprocessor systems.

It is to be noted that using a multiprocessor is more beneficial than using independent processors. The parallelism existing within each multiprocessor helps in gaining localized high performance and also maintains extensive multithreading for the fine-grained parallel programming models. The thread block has individual threads that execute together within a multiprocessor to allocate data.

For maintaining area and power efficiency, the multiprocessor shares large and complex units among the different processor cores, which also include the instruction cache, the multithreaded instruction unit, and the shared memory RAM.

One of the main advantages of multiprocessor is shared memory programming model. Shared-memory multiprocessors have a major advantage over other multiprocessors, as all the other processors share the same view of the memory. These processors are also termed as Uniform Memory Access (UMA) systems. This term indicates that all processors can equally access the memory with the same performance.

Notes

The popularity of the shared-memory systems is just not due to the demand for high performance computing. These systems also provide high throughput for a multiprocessing load. They also work efficiently as high-performance database servers, Internet servers, and network servers. As more processors are added, the throughput of these systems is increased linearly.

Multiprocessors also find their applications in various domains which include:

1. ***Server Workload:*** This includes many concurrent updates, lookups, searches, queries, and so on. Processors deal with different requests.

Example: Database for airline reservation

2. ***Media Workload:*** Processors compress/decompress different parts of image/frames. This includes compressing/decompressing of different parts of image/frames.
3. ***Scientific Computing:*** This includes large grids that integrate changes over time, and each processor computes for a part of the grid.

Example: Protein folding, aerodynamics, and weather simulation.

13.3 Interconnection Structures

The structures that are used to connect the memories and processors (and between memories and I/O channels if required), are called interconnection structures. A multiprocessor system is formed by elements such as CPUs, peripherals, and a memory unit that is divided into numerous separate modules. There can exist different physical configurations for the interconnection between the elements. The physical configurations are based on the number of transfer paths existing between the processors and memory in a shared memory system or among the processing elements in a loosely coupled system. An interconnection network is established using several physical forms available. Some of the physical forms include:

1. Time-shared common bus
2. Multiport memory
3. Crossbar switch
4. Multistage switching network
5. Hypercube system

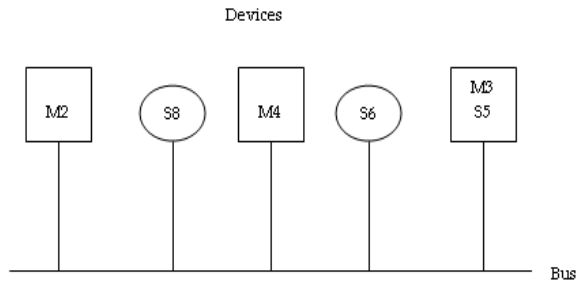
Operation of Bus

Bus is defined as a group of signal lines that carry module-to-module communication. Here, data highways connect several digital system elements. Each processor (and memory) is connected to a common bus. Memory access is moderately uniform, but it is less scalable.

Notes

Figure 13.5 depicts operation of bus.

Figure 13.5: Operation of Bus



In figure 13.5:

Master Device (M2, M3, M4): This is a device that initiates and controls the communication.

Slave Device (S5, S6, S8): This is a responding device.

As depicted in figure 13.5, if M2 wishes to communicate with S6,

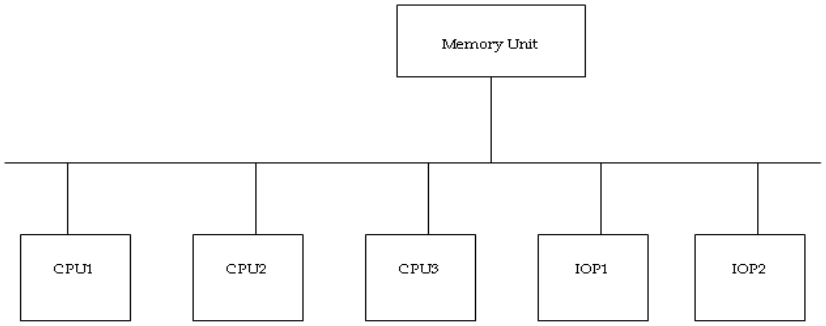
1. M2 sends signals (address) on the bus that causes S6 to respond.
2. M2 sends data to S6, or S6 sends data to M2. (determined by the command line)

13.3.1 Time-shared Common Bus

In time-shared common bus, there are numerous processors connected through a common path to the memory unit in a common-bus multiprocessor system. Figure 13.6 shows organization of time-shared common bus for five processors. At any specified time, only one processor can communicate with the memory or another processor. The processor that is in control of the bus at the time performs transfer operations. Any processor that wants to initiate a transfer must first verify the availability status of the bus.

Once the bus is available, the processor can establish a connection with the destination unit to initiate the transfer. A command is issued to inform the destination unit about the function to be performed. The receiving unit identifies its address in the bus, and then responds to the control signals from the sender, after which the transfer is initiated. As all processors share a common bus, it is possible that the system may display some transfer conflicts. Incorporation of a bus controller that creates priorities among the requesting units helps in resolving the transfer conflicts.

Figure 13.6: Organization of a Time-Shared Common Bus

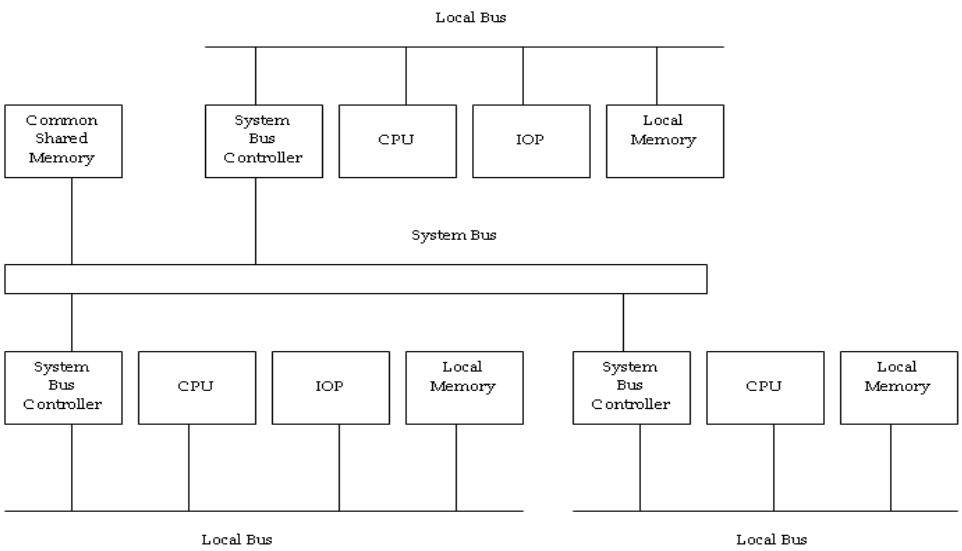


Notes

There is a restriction of one transfer at a time for a single common-bus system. This means that other processors are busy with internal operations or remain idle waiting for the bus when one processor is communicating with the memory. Hence, the speed of the single path limits the total overall transfer rate within the system. The system processors are kept busy through the execution of two or more independent buses, to allow multiple bus transfers simultaneously. However, this leads to increase in the system cost and complexity.

Figure 13.7 depicts a more economical execution of a dual bus structure for multiprocessors.

Figure 13.7: System Bus Structure for Multiprocessors



In figure 13.7 we see that there are many local buses, and each bus is connected to its own local memory, and to one or more processors. Each local bus is connected to a peripheral, a CPU, or any combination of processors. Each local bus is linked to a common system bus using a system bus controller.

The I/O devices connected to both the local I/O peripherals and the local memory is available to the local processor. All processors share the memory connected to the common system bus. When an IOP is connected directly to the system bus, the Input/Output devices attached to it are made available to all processors. At any specified time, only one processor can communicate with the shared memory, and other common resources through the system bus. All the other processors are busy communicating with their local memory and I/O devices.

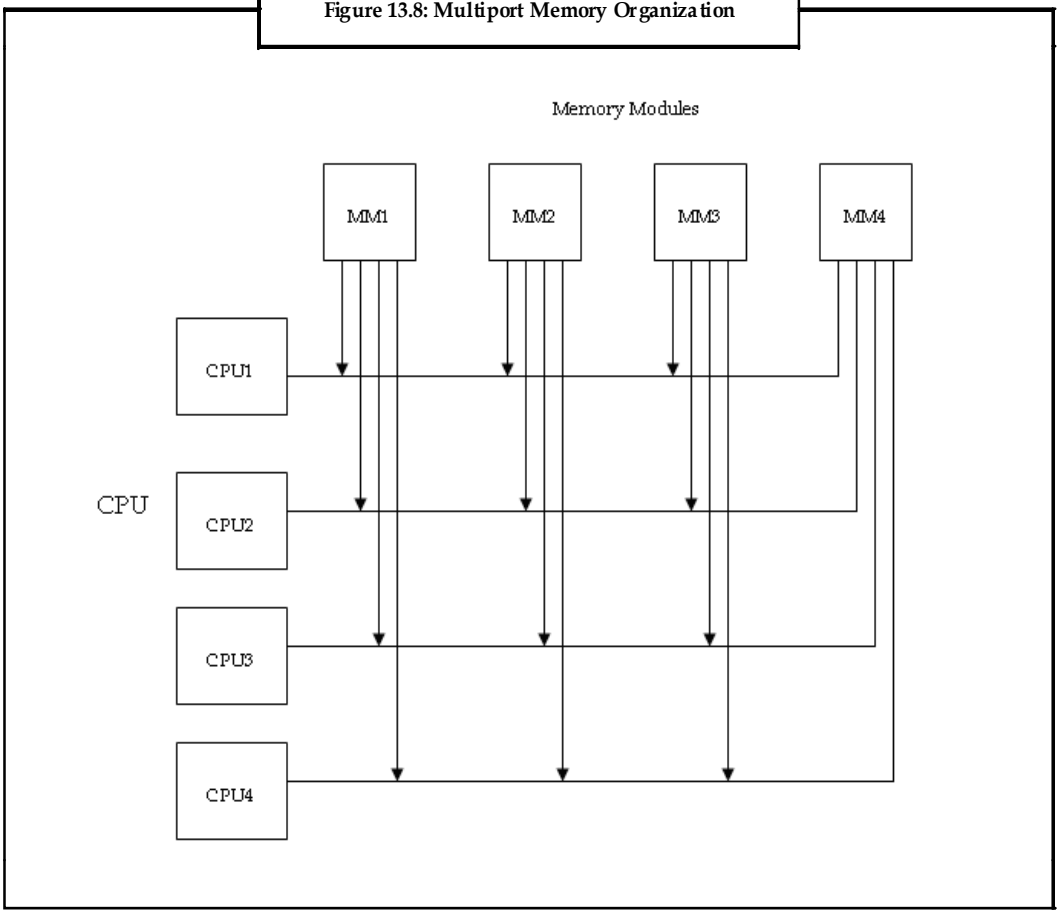
13.3.2 Multiport Memory

Multiport memory is a memory that helps in providing more than one access port to separate processors or to separate parts of one processor. A bus can be used to achieve this kind of an access. This mechanism is applicable to interconnected computers too. A multiport memory system uses separate buses between each CPU and each memory module. Figure 9.8 depicts a multiport memory system for four CPUs and four Memory Modules (MMs). Every processor bus is connected to each memory module. A processor bus consist three elements; namely: address, data, and control lines. These elements are needed to communicate with memory. Memory module has four ports and each port contains one of the buses. It is necessary for a module to have internal control logic to verify which port will have access to memory at any specified time. Assigning fixed priorities to each memory port helps in resolving memory access conflicts. The priority for memory access

Notes

related to each processor is created with the physical port position that its bus occupies in each module. Consequently, CPU1 has priority over CPU2, CPU2 has priority over CPU3, and CPU4 has the least priority.

Figure 13.8: Multiport Memory Organization



The multiport memory organization has an advantage of high transfer rate. This is because of several paths between memory and processors. The only drawback is that it needs expensive memory control logic and more number of cables and connectors. Therefore, this interconnection structure is usually suitable for systems having small number of processors.

13.3.3 Crossbar Switch

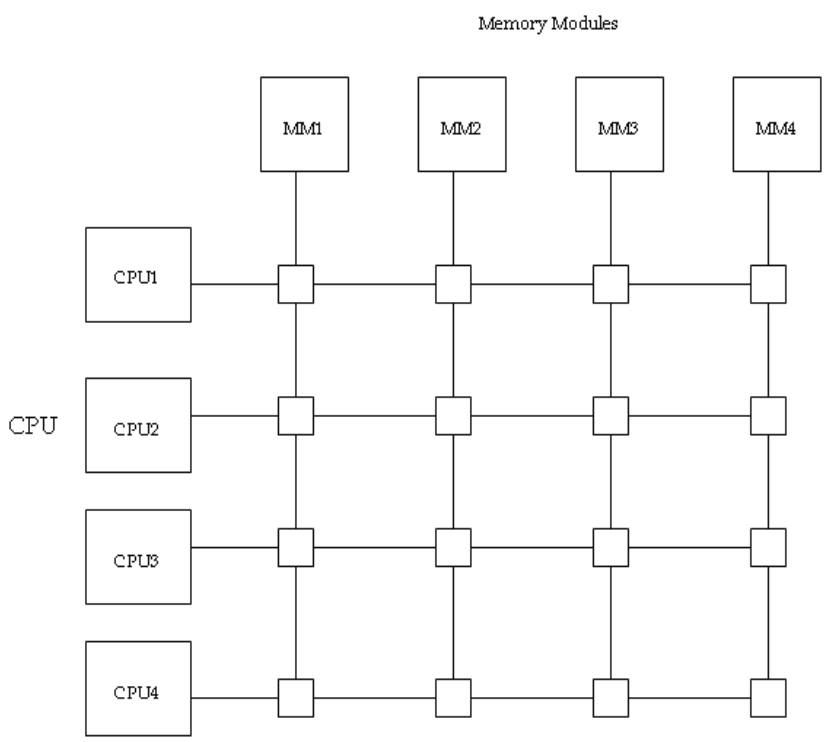
In a network, a device that helps in channeling data between any two devices that are connected to it, up to its highest number of ports is a crossbar switch. The paths set up between devices can be fixed for some period of time or changed when wanted.

In a crossbar switch organization, there are several cross points that are kept at intersections between processor buses and memory module paths.

Notes

Figure 13.9 shows a crossbar switch interconnection between four memory modules and four CPUs.

Figure 13.9: Crossbar Switch

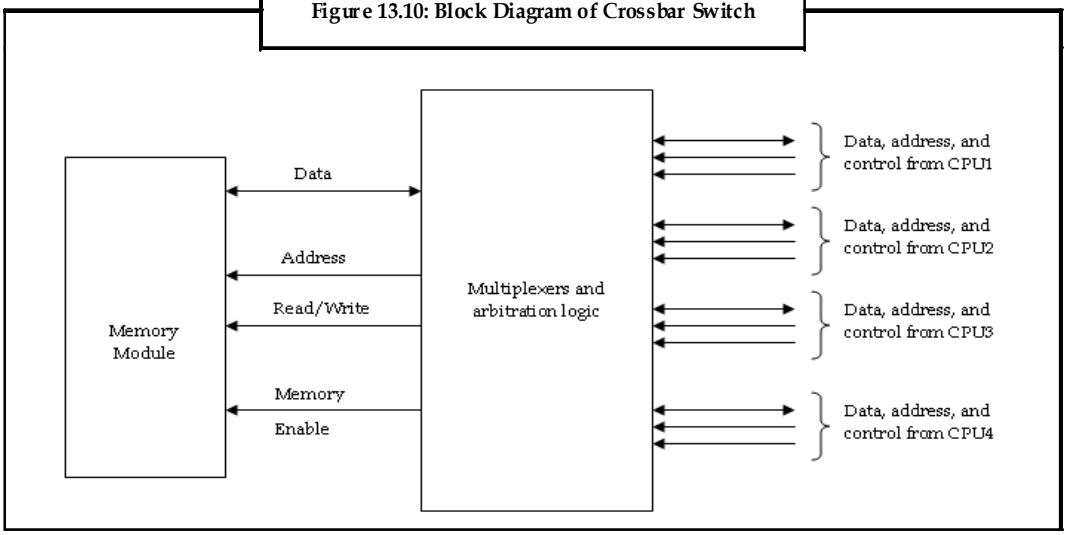


In figure 13.9, the small square in each crosspoint indicates a switch. This switch determines the path starting from a processor to a memory module. There is control logic for each switch point to set up the transfer path between a memory module and a processor. It checks the address that is placed in the bus to verify if its particular module is addressed. It also allows resolving multiple requests to get access to the same memory module on a predetermined priority basis.

Notes

The functional design of a crossbar switch connected to one memory module is depicted in figure 13.10.

Figure 13.10: Block Diagram of Crossbar Switch



The circuit includes multiplexers that choose the data, address, and control from one CPU for communication with the memory module. The arbitration logic establishes priority levels to choose one CPU when two or more CPUs try to get access to the same memory. The binary code controls the multiplexers. A priority encoder generates this binary code within the arbitration logic.

Notes A crossbar switch organization maintains and supports simultaneous transfers from memory modules, since there is a separate path related with each module. On the other hand, the hardware necessary to implement the switch may be quite large and complex.

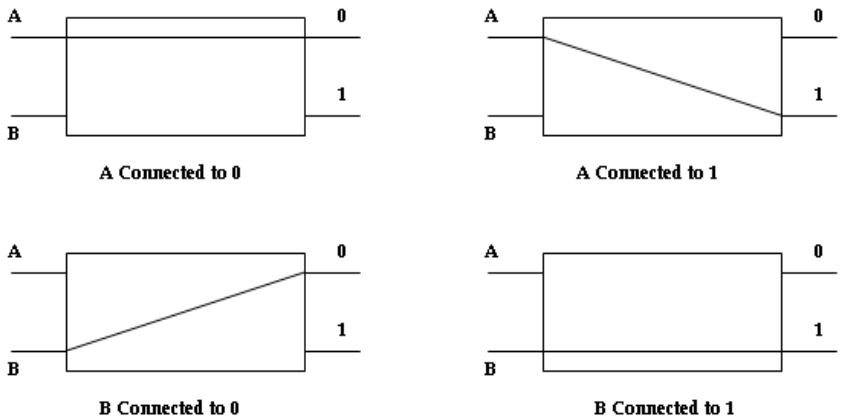
13.3.4 Multistage Switching Network

The network that is built from small (for example, 2 x 2 crossbar) switch nodes along with a regular interconnection pattern is a multistage switching network. Two-input, two-output interchange switch is a fundamental element of a multistage network. There are two inputs marked A and B, and two outputs marked 0 and 1 in the 2 x 2 switch as shown in figure 13.11.

Notes

Figure 13.11 depicts operation of a 2x2 interchange switch.

Figure 13.11: Operation of a 2 x 2 Interchange Switch



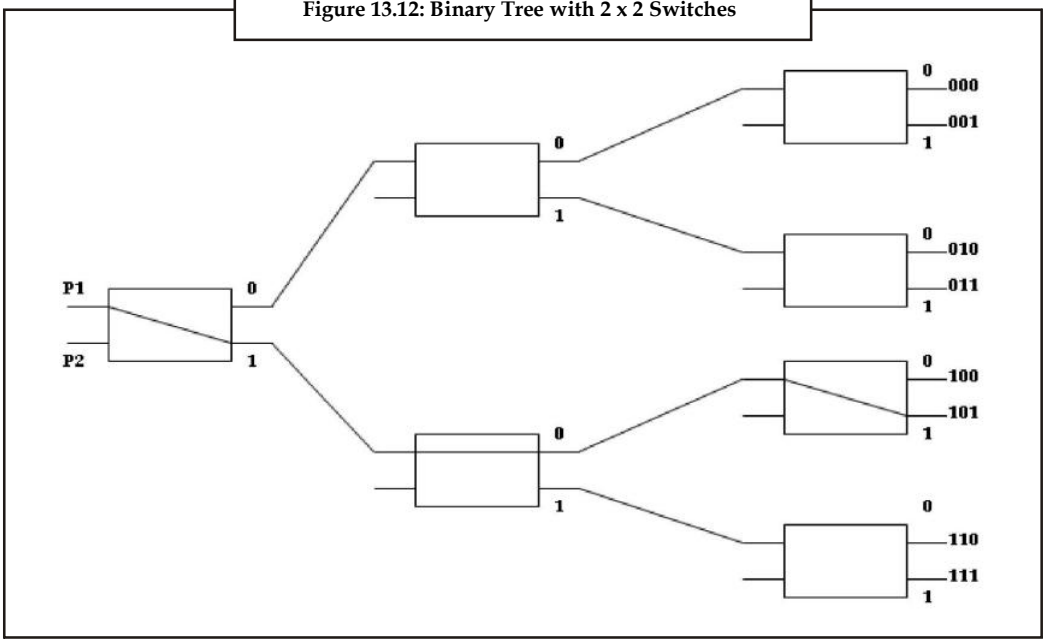
As depicted in figure 13.11, there are control signals associated with the switch. The control signals establish interconnection between the input and output terminals. The switch can connect input A to either of the outputs. Terminal B of the switch acts in a same way. The switch can also arbitrate between conflicting requests. In case, inputs A and B request the same output terminals, it is possible that only one of the inputs is connected and the other is blocked.

It is possible to establish a multistage network to control the communication between numerous sources and destinations. The multistage network is established with the help of 2 x 2 switch as a building block.

Notes

Consider the binary tree shown in figure 13.12 to see how this is carried out.

Figure 13.12: Binary Tree with 2 x 2 Switches



The two processors P1 and P2 are linked through switches to eight memory modules labeled in binary, starting from 000 through 111. The path starting from source to destination is determined from the binary bits of destination number. The first bit of the destination number helps in indicating the first level's switch output. The second bit identifies the second level's switch output, and the third bit specifies the third level's switch output.

Example: As shown in figure 13.12, in order to make a connection between P1 and memory 101, it is important to create a path from P1 to output 1 in the third-level switch, output 0 in the second-level switch, and output 1 in the third-level switch. Hence, it is evident that either P1 or P2 must be connected to any one of the eight memories.

It is also evident that certain request patterns however cannot be satisfied simultaneously.

Example: As shown in figure 13.12, if P1 is connected to one of the destinations 000 through 011, then it is possible to connect P2 to only one of the destinations 100 through 111.

There are many topologies for multistage switching networks that help to:

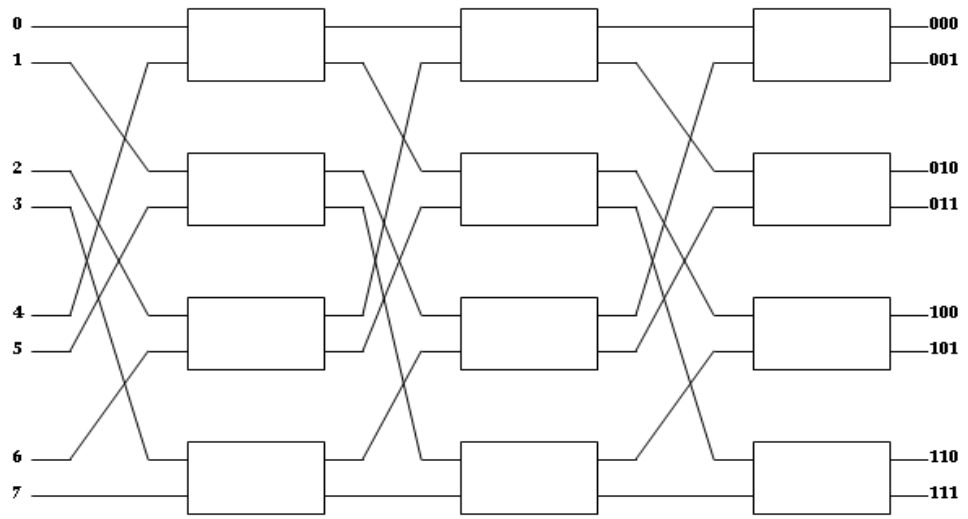
1. Control the processor-memory communication in a tightly-coupled multiprocessor system.
2. Control the communication between the processing components in a loosely-coupled system.

Omega switching network is one such topology that is depicted in the figure 13.13. There exists exactly one path from source to any specific destination in this configuration. However, certain request patterns cannot be connected simultaneously. For example, it is not possible to connect any two sources simultaneously to destinations 000 and 001.

Notes

Figure 13.13 depicts 8x8 Omega switching network.

Figure 13.13: 8 x 8 Omega Switching Network



As depicted in figure 13.13, a specific request is started in the switching network through the source that sends a 3-bit pattern depicting the destination number. Every level checks a different bit to determine the 2 x 2 switch setting as the binary pattern moves through the network. Level 1 examines the most important bit, level 2 examines the middle bit, and level 3 examines the least important bit. When the request appears on input 2 x 2 switch, it is routed to the lower output if the specified bit is 1 or to the upper output if the specified bit is 0.

The source is considered to be a processor and the destination is considered as a memory module in a tightly-coupled multiprocessor system. The path is set when the first pass is through the network. If the request is read or write the address is transferred into memory, and then the data is transferred in either direction using the succeeding passes. Both the destination and the source are considered to be processing elements in a loosely-coupled multiprocessor system. The source processor transfers a message to the destination processor once the path is established.

13.3.5 Hypercube Interconnection

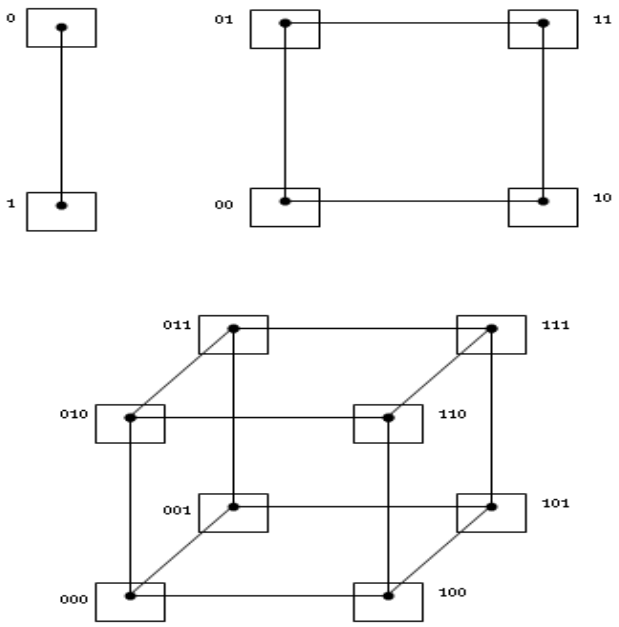
The hybercube is considered to be a loosely coupled system. This system is composed of $N = 2^n$ processors that are interconnected in an n-dimensional binary cube. Each processor indicates a node of the cube. Although it is expected to refer to every node as having a processor, in effect it not only has a CPU but also local memory and I/O interface. Every processor contains direct communication paths to n other neighbor processors. These paths relates to the edges of the cube. The processors can be assigned with 2^n distinct n-bit binary addresses. Each processor address differs from that of each of its n neighbors by exactly one bit position.

Did u know? The hypercube interconnection is also referred to as binary *n*-cube multiprocessor structure.

Notes

Figure 13.14 depicts the hypercube structure for n, wherein n = 1, 2, and 3.

Figure 13.14: Hypercube Structures for n = 1, 2, and 3



A depicted in figure 13.14, a one-cube structure contains n = 1 and 2^n = 2. It has two processors that are interconnected by a single path. A two-cube structure contains n = 2 and 2^n = 4. It has four nodes that are interconnected as a square. There are eight nodes interconnected as a cube in a three-cube structure. There are 2^n nodes in an n-cube structure with a processor existing in every node.

A binary address is assigned to every node such that the addresses of two neighbors vary in exactly one bit position.

Example: As shown in figure 13.13, the three neighbors of the node having address 100 in a three-cube structure are 000, 110, and 101. Each of these binary numbers vary from address 100 by one bit value.

Routing messages through an n-cube structure may require one to n links, starting from a source node to a destination node.

Example: As shown in figure 13.12, it is possible for node 000 to communicate directly with node 001 in a three-cube structure. To communicate from node 000 to node 111, the message has to travel through at least three links.

Computing the exclusive-OR of the source node address with the destination node address helps in developing a routing procedure. The resulting binary value has 1 bit relating to the axes on which the two nodes vary. Later, the message is sent along any one of the axes.

Example: A message at 010 being sent to 001 generates an exclusive-OR of the two addresses equivalent to 011 in a three-cube structure. It is possible to send the message along the second axis to 000 and then through the third axis 001.

Notes

Example: The Intel iPSC complex is considered to be a representative of the hypercube architecture.

The Intel iPSC has 128 (n = 7) microcomputers connected through communication channels. Each node has a CPU, local memory, floating-point processor, and serial communication interface units. The individual nodes work independently on data saved in local memory according to the resident programs. It is evident that the programs and data at every node is received through a message-passing system from other nodes or from a cube manager. Application programs are developed and gathered on the cube manager and then downloaded to the individual nodes. Computations are allocated through the system and implemented concurrently.

Task Visit http://ed-thelen.org/comp-hist/intel-iPSC-860.html and discuss about the hypercube architecture and special features of Intel iPSC/860.

13.4 Interprocessor Communication and Synchronization

A multiprocessor system has various processors that must be provided with a facility to communicate with each other. Using a common I/O channel, a communication path is established. The most frequently used procedure in a shared memory multiprocessor system is to set aside a part of the memory that is available to all processors. The major use of the common memory is to work as a message center similar to a mailbox, where every processor can leave messages for other processors and pick up messages meant for it.

The sending processor prepares a request, a message, or a procedure, and then places it in the memory mailbox. The receiving processor can check the mailbox periodically to determine if there are valid messages in it, as a processor identifies a request only while polling messages. However, the response time of this procedure may be time consuming. The sending processor has a more efficient procedure, and the procedure involves alerting the receiving processor directly using an interrupt signal. This procedure is achieved with the help of software initiated interprocessor interrupt initialized in one processor, which when implemented generates an external interrupt condition in a second processor. This interrupt informs the second processor that processor one has inserted a new message in its mailbox.

Notes Status bits present in common memory are usually used to determine the condition of the mailbox, if it has meaningful data, and for which processor it is intended.

A multiprocessor system has other shared resources in addition to shared memory.

Example: An IOP to which a magnetic disk storage unit is connected, is available to all CPUs. This helps in providing a facility for sharing of system programs stored in the disk.

A communication path can be established between two CPUs through a link between two IOPs, which connects two different CPUs. This kind of link allows each CPU to treat the other as an I/O device, such that messages can be transferred through the I/O path.

There should be a provision for assigning resources to processors to avoid inconsistent use of shared resources by many processors. This job is given to the operating system. The three organizations that are used in the design of operating system for multiprocessors include:

1. Master-slave configuration
2. Separate operating system
3. Distributed operating system

In a master-slave configuration mode, one processor, designated the master, always implements the operating system functions. The remaining processors, designated as slaves, do not execute

Notes

operating system functions. If a slave processor requires an operating system service, then it should request it by interrupting the master.

Each processor can implement the operating system routines that it requires in the separate operating system organization. This kind of organization is more appropriate for loosely-coupled systems wherein, every processor needs to have its own copy of the entire operating system.

The operating system routines are shared among the available processors in the distributed operating system organization. However, each operating system function is allocated to only one processor at a time. This kind of organization is also termed as a floating operating system because the routines float from one processor to another, and the implementation of the routines are allocated to different processors at different times.

The memory is distributed among the processors and there is no shared memory for sending information in a loosely-coupled multiprocessor system. Message passing system through I/O channels is used for communication between processors. The communication is started by one processor calling a procedure that exists in the memory of the processor with which it has to communicate. A communication of channel is established when both the sending processor and the receiving processor recognize each other as source and destination. A message is then sent to the nodes with a header and different data objects that are required for communication between the nodes. In order to send the message between any two nodes, several possible paths are available. The operating system of each node has the routing information which indicates the available paths to send a message to different nodes.

The communication efficiency of the interprocessor network depends on four major factors and they are:

1. Communication routing protocol
2. Processor speed
3. Data link speed
4. Topology of the network

Interprocessor Synchronization

Synchronization is a communication of control information between processors. Synchronization helps to:

1. Implement the exact sequence of processes.
2. Ensure mutually exclusive access to allocated writable data.

Synchronization refers to a special case where the control information is the data employed to communicate between processors. Synchronization is necessary to implement the exact sequence of processes and to ensure mutually exclusive access to shared writable data.

There are many mechanisms in multiprocessor systems to handle the synchronization of resources. The hardware directly implements low-level primitives. These primitives act as essential mechanisms that enforce mutual exclusion for more difficult mechanisms executed in software. Many hardware mechanisms for mutual exclusion are developed. However, the use of a binary semaphore is considered to be one of the most popular mechanisms.

The following are the methods to achieve synchronization.

Synchronization can be achieved by mutual exclusion with a semaphore. Semaphores are considered to be the means of addressing the requirements of both task synchronization and mutual exclusion. Mutual exclusion includes a processor to eliminate or lock out access to allocated resource by other processors when it is in a **Critical Section.**

Notes

Mutual Exclusion with a Semaphore

Appropriately operating multiprocessor system must provide a mechanism that would ensure systematic access to shared memory and other shared resources. This is required to protect data, since two or more processors can change the data simultaneously. This mechanism is referred to as mutual exclusion. A multiprocessor system must have mutual exclusion to allow one processor to rule out or lock out access to an allocated resource by other processors when it is in a critical section. A critical section is defined as a program sequence which once started must complete implementation before another processor accesses the same allocated resource.

Notes A semaphore is considered to be a software-controlled flag stored in a memory location such that all processors can access.

When the semaphore is set to one, it indicates that a processor is implementing a critical program, and the shared memory is unavailable to other processors. When the semaphore is set to zero, it indicates that the shared memory is available to any requesting processor. Processors sharing the same memory segment agree to not use the memory segment unless the semaphore is 0, showing that memory is available. The processors also concur to set the semaphore to 1, while they are implementing a critical section, and then to clear it to 0 when they are done.

Testing and setting the semaphore is considered to be a critical function, and needs to be carried out as a single indivisible operation. Otherwise, two or more processors may check the semaphore simultaneously and set the semaphore in such a way that it can enter a critical section at the same time. This action allows the simultaneous execution of these critical sections resulting in incorrect initialization of control factors and a loss of necessary information.

A semaphore is initialized using a test and set instruction together with a hardware lock mechanism. A hardware lock is defined as a processor-generated signal that helps in preventing other processors from using the system bus as long as the signal is active. When the instruction is being executed, the test-and-set instruction tests and sets a semaphore and activates the lock mechanism. This helps in preventing the other processors from changing the semaphore between the time that the processor is testing it and the time that the processor is setting it. Consider that the semaphore is a bit in the least significant position of a memory word whose address is symbolized by SEM. Let the mnemonic TSL designate the "test and set while locked" function. The instruction TSL SEM is executed in two memory cycles, that is, the first one to read and the second to write without any interference as given below:

R ê M[SEM] Test semaphore

M[SEM] ê 1 Set semaphore

In order to test the semaphore, its value is transferred to a processor register R and then set to 1. The value of R indicates what to do next. If the processor identifies that R = 1, it means that the semaphore was initially set. Even if the register is set again, it does not change the value of the semaphore. This indicates that another processor is executing a critical section and therefore, the processor that checked the semaphore does not access the shared memory. The common memory or the shared resource that the semaphore represents is available when R = 0. In order to avoid other processors from accessing memory, the semaphore is set to 1. Now, it is possible for the processor to execute the critical section. To release the shared resource to other processors, the final instruction of the program must clear location SEM to zero.

It is crucial to note that the lock signal must be active at the time of execution of the test-and-set instruction. Once the semaphore is set, the lock signal does not have to be active. Therefore, the lock mechanism prevents other processors from accessing memory while the semaphore is being set. Once set, the semaphore itself will prevent other processors from accessing shared memory while one processor is implementing a critical section.

Notes

Task Visit http://www.articlesbase.com/information-technology-articles/multiprocessor-semaphore-318193.html and discuss how shared memory semaphores act as essential tools for interprocessor synchronization.

13.5 Summary

- A multiprocessor generally refers to a single computer that has many processors.
- The term 'multiprocessor' can also be used to describe several separate computers running together. It is also referred to as clustering.
- Processors in the multiprocessor system communicate and cooperate at various levels of solving a particular problem.
- The difference that exists between multicomputer systems and multiprocessors depends on the extent of resource sharing and cooperation in solving a problem.
- Multiprocessor system uses a distributed approach, wherein a single processor does not perform a complete task but more than one processor is used to perform the subtasks.
- There are two types of multiprocessor systems; they are tightly-coupled multiprocessor system and loosely-coupled multiprocessor system.
- One of the major characteristics of shared memory multiprocessors is that all processors have equally direct access to one large memory address space.
- Multiprocessor systems work efficiently as high-performance database servers, Internet servers, and network servers.
- Common-bus multiprocessor system has numerous processors connected through a common path to a memory unit.
- A multiport memory system uses separate buses between each CPU and each memory module.
- There are many cross points located at intersections between processor buses and memory module paths in a crossbar switch organization.
- The most frequently used procedure in a shared memory multiprocessor system is to set aside a part of memory that is available to all processors.
- Appropriately functioning multiprocessor system must provide a mechanism that will ensure systematic access to shared memory and other shared resources.

Notes

13.6 Keywords

Autonomous Computers: A network administered by a single set of management rules that are controlled by single person, group, or organization. Autonomous systems frequently use only one routing protocol even though it is possible to use multiple protocols.

Control Logic: It is the part of a software architecture that helps in controlling what the program will do. This part of the program is also termed as controller.

Multithreading: It is a process wherein the same job is broken logically and performed simultaneously and the output is combined at the end of processing.

Real-time Applications: A real-time application is an application program that works within a given time frame that the user assumes as immediate or current.

13.7 Self Assessment

1. State whether the following statements are true or false:
   (a) The communication between the processors happens by sending messages from one processor to another or by sharing a common memory.
   (b) If failure happens, it is difficult and expensive to identify and replace the malfunctioning processor instead of replacing the failing part of complex processor.
   (c) One of the major characteristics of shared memory multiprocessors is that all processors have equally direct access to one large memory address space.
   (d) It is noted that using a multiprocessor is not as beneficial as using independent processors.
   (e) There is a restriction of one transfer at a time for a single common-bus system.
   (f) There is control logic for each switch point to set up the transfer path between memory and a processor.
2. Fill in the blanks:
   (a) The _____ communication is carried out with the help of shared memories or through an interrupt network.
   (b) A method wherein a specific task is divided into several subtasks that must be performed in a sequence is _____.
   (c) There are numerous processors connected through a common path to a memory unit in a _____ multiprocessor system.
   (d) The _____ establishes priority levels to choose one CPU when two or more CPUs try to get access to the same memory.
   (e) A semaphore is initialized using a test and set instruction together with a _____ mechanism.

Notes

3. Select a suitable choice in every question.
   (a) Which of the following involves single architecture in order to execute a common task?
      (i) Distributed computing
      (ii) Parallel computing
      (iii) Super computing
      (iv) Vector computing
   (b) Which of the following is the memory wherein all memory units are associated with processors?
      (i) Shared (Global) memory
      (ii) Uniform memory
      (iii) Distributed memory
      (iv) Non-uniform memory
   (c) Which of the following is used to establish a network to control the communication between numerous sources and destinations?
      (i) Time-shared common bus
      (ii) Multistage switching network
      (iii) Memory unit
      (iv) Hypercube system
   (d) Which of the following is a kind of organization used in the design of operating system for multiprocessors that is also termed as a floating operating system?
      (i) Distributed operating system
      (ii) Separate operating system
      (iii) Master-slave configuration
      (iv) Loosely-coupled multiprocessor system
   (e) Which of the following is a major factor that communication efficiency of the interprocessor network depends on?
      (i) Writable data
      (ii) Communication routing protocol
      (iii) External interrupt condition
      (iv) Receiving processor

13.8 Review Questions

1. "Multiprocessor system has many advantages". Elaborate.
2. "Multiprocessor has many major characteristics". Explain some of the characteristics of multiprocessor system.
3. Explain why loosely-coupled multiprocessor system is more often referred to as clusters.
4. "Granularity of parallelism is of three types". Briefly explain the three types of granularity of parallelism.
5. "Use of multiprocessor systems in real-time applications is becoming popular". Justify.
6. "Multiprocessors find their applications in various domains". Elaborate.

Notes

7. "There are numerous processors connected through a common path to a memory unit in a common-bus multiprocessor system". Explain this concept for a time-shared common bus for five processors with the figure.
8. "A multiport memory system uses separate buses between each CPU and each memory module". Elaborate this concept with a figure for four CPUs and four memory modules (MMs).
9. "There are numerous cross points that are located at intersections between processor buses and memory module paths in a crossbar switch organization". Explain a crossbar switch interconnection between four memory modules and four CPUs with a diagram.
10. "Two-input, two-output interchange switch is a fundamental element of a multistage network". Elaborate.
11. "There are many topologies for multistage switching networks and omega switching network is one such topology". Explain.
12. "The hypercube system is considered to be a loosely coupled system and this system is composed of N = 2n processors that are interconnected in an n-dimensional binary cube". Elaborate and explain the hypercube structure for n = 1, 2, and 3.
13. "There are three organizations that are used in the design of operating system for multiprocessors". Elaborate.

Answers: Self Assessment

1. (a) True
   (b) False
   (c) True
   (d) False
   (e) True
   (f) True
2. (a) Interprocessor
   (b) Pipelining
   (c) Common-bus
   (d) Arbitration logic
   (e) Hardware lock
3. (a) Parallel computing
   (b) Distributed memory
   (c) Multistage switching network
   (d) Distributed operating system
   (e) Communication routing protocol

Notes

13.9 Further Readings

Books

Godse, A. P., & Godse, D. A. (2009). Computer Organization.1st ed. Pune: Technical Publications.

Stallings, W. Computer Organization and Architecture: Designing for Performance.

Morris, M, Computer System Architecture, 3rd ed.

Online links

http://cnx.org/content/m32797/latest/

Notes

Unit 14: Introduction to Parallel Processing

CONTENTS

Objectives
Introduction
14.1 Pipelining
14.1.1 Pipelining Conflicts
14.1.2 Techniques for Overcoming Pipelining Conflicts
14.2 Instruction Pipeline
14.3 RISC Pipeline
14.4 Vector Processing
14.4.1 Characteristics of Vector Processing
14.4.2 Advantages of Vector Processing
14.5 Parallel Processing
14.6 Summary
14.7 Keywords
14.8 Self Assessment
14.9 Review Questions
14.10 Further Readings

Objectives

After studying this unit, you will be able to:

- Describe the process of pipelining
- Explain the working of an instruction pipeline
- Define RISC pipeline
- Discuss vector processing
- Discuss parallel processing

Introduction

A large class of techniques is used to speed the processing of a computer system. The two basic techniques which can increase the instruction execution rate of a processor are to increase the clock rate and to increase the number of instructions that can be executed at the same time. Pipelining and instruction-level parallelism are examples of the second technique. Parallel processing is a technique used to permit data processing tasks to happen simultaneously in order to increase the speed of processing of a computer system.

The instruction executions in conventional computers were done in a sequential manner wherein, if one program is being executed the other one waits till the first one is completed. In contrast, in parallel processing, the execution is done concurrently, resulting in faster execution time and higher throughput. The hardware requirement for a parallel processing system is higher than a conventional computer system.

Parallel processing is fast since it utilizes concurrent data processing to achieve faster execution. Parallel processing is achieved by dividing the data among different units wherein each unit is

Notes

processed simultaneously. The control unit governs the timing and sequencing in order to obtain the desired results in a minimum amount of time.

As we already know, the operation sequence in a computer is to first fetch instructions from memory and then execute them in the processor. The sequence of instructions read from memory, makes an instruction stream. The data operations performed in the processor are consisted in the data stream. Parallel processing may occur in the instruction stream, data stream, or in both.

Parallel processing can be classified in a number of ways—from the processors internal organization, from the interconnection structure between processor, or from the information flow through the system point of view.

Pipelining and vector processing are different aspects of parallel processing which are discussed in the subsequent sections of this unit.

14.1 Pipelining

Pipelining is a technique of breaking a sequential process into small fragments or sub operations. The execution of each of these sub process takes place in a special dedicated segment that functions concurrently with all other segments. The pipeline has a collection of processing segments which helps the flow of binary information. The internal working in a pipeline is such that the outcome of one segment is conveyed to the next segment in the pipeline until the desired result is obtained. The final outcome is obtained after the data is passed through all segments.

The term "pipeline" indicates that the flow of information takes place in parallel. Pipelining refers to the temporal overlapping of processing. The overlapping of processing is done by associating a register with each segment in the pipeline. The registers help in providing isolation between each segment so that every segment can work on distinct data simultaneously.

Did u know? There is a common analogy between pipelining in a computer and manufacturing parts on an assembly line. In both the cases the aim is to keep equipments busy for maximum time, increase throughput by increasing the number of stages and decreasing the amount of work done at a given stage.

A segment consists of an input register and a combinational circuit. The register stores the data and the combinational circuit performs the operation in the particular segment. The output of the combinational circuit of a segment is sent to the input register of the next segment. To perform the activity in each segment, a clock is set for each register.

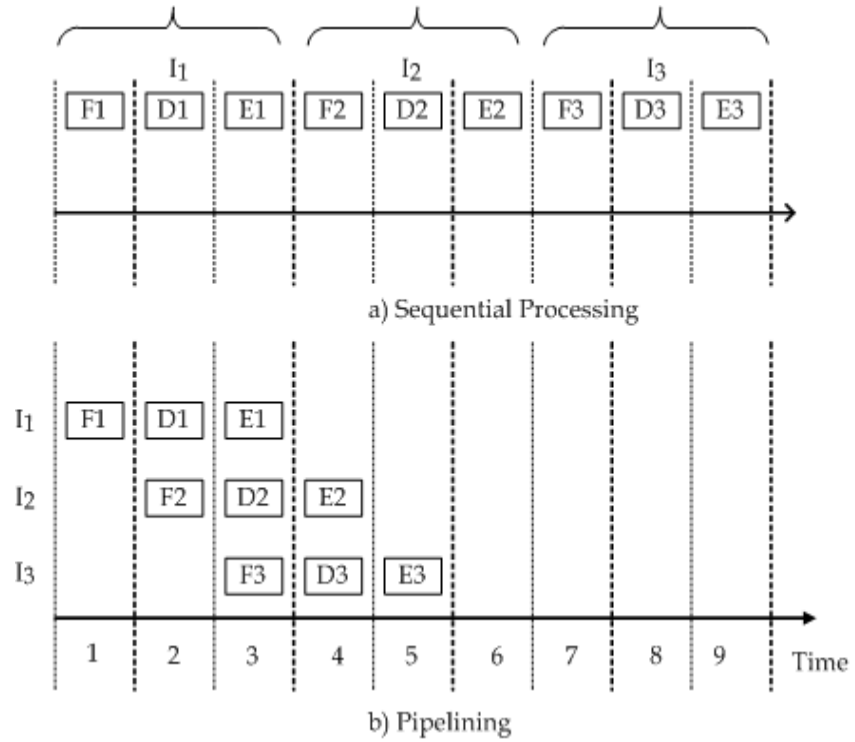
When we consider a sequential execution of three instructions, each having three stages of execution, the sequence got is the following instruction cycle.

If each stage requires one unit time and a separate unit for each action, then the total time taken would be nine units. In the case of pipelined execution of the same instruction set, the sequence would require only five units. This is shown in figure 14.1.

Notes

Figure 14.1 depicts the comparison between parallel processing and sequential processing.

Figure 14.1: Pipelining Versus Sequential Processing



As observed in figure 14.1, we can save approximately 50% of the execution time by using pipelining.

Figure 14.1 depicts a space time chart which helps in depicting the performance measures of using pipeline. The chart depicts the working of the subtasks with respect to time. From figure 14.1 it can be observed that the time required to process three instructions (I_1, I_2, I_3) is only five time units if three stage pipelining is used and nine time units if sequential processing is used.

Example: The pipeline organization can be understood with the help of an example to perform the combined multiply and add operations

Ex: An * Bn + Cn * Dnn =1,2,3….

The sub operations performed in each segment of the pipeline is as shown in the table 14.1

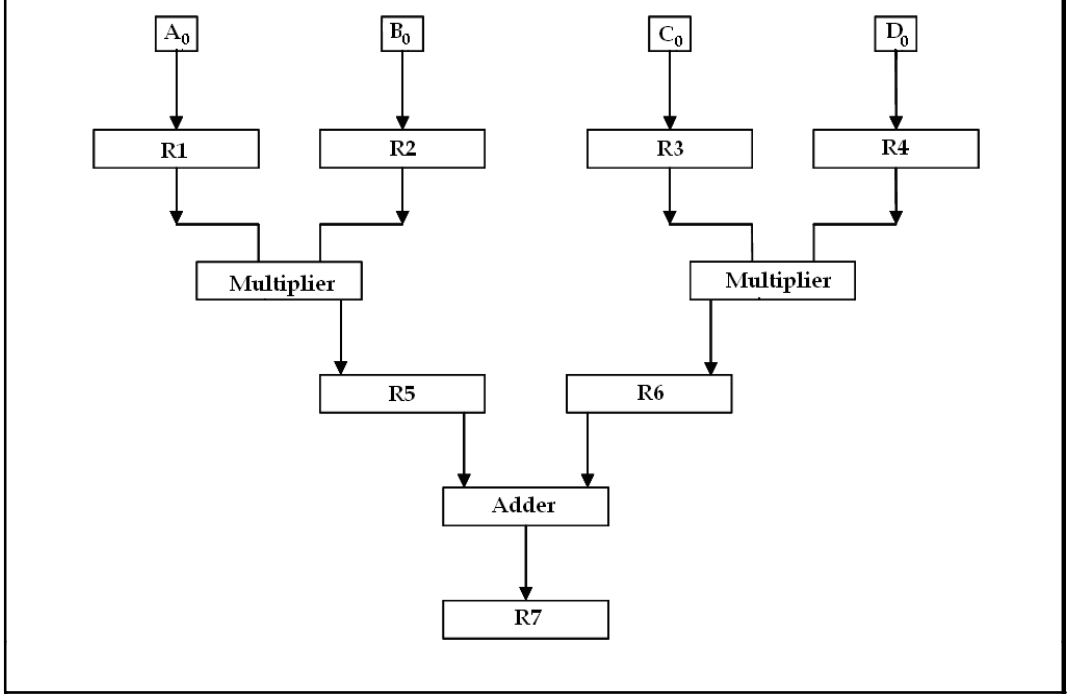
Table 14.1: Sub Operations

R1? An, R2 ? Bn, R3 ? Cn, R4 ? Dn	Input An, Bn, Cn, Dn
R5 ? An * Bn, R6 ? Cn * n	Multiply
R7 ? R5 + R6	Add and store in Register R7

Notes

Each sub operation is implemented in a segment within a pipeline. The separation between each segment is provided by registers for each segment to work on different data simultaneously. Each segment uses one or three registers with combinational circuits. R1 through R7 are registers that receive new data with every clock pulse. The combinational circuits are multiplier and adder.

Figure 14.2: Example of Pipeline Processing



The seven registers are loaded with new data during every clock pulse. The effect of each clock pulse in the register is shown in table 14.2.

The following is the sequence in which the instructions are carried out:

1. ***Clock Pulse 1***: The first clock pulse transfers:
   (a) A1 and B1 into R1 and R2
   (b) C1 and D1 into R3 and R4
2. ***Clock Pulse 2:*** The second clock pulse transfers:
   (a) The product of R1 and R2 into R5
   (b) The product of R3 and R4 into R6
      (i) A2 and B2 into R1 and R2, (ii) C2 and D2 into R3 and R4.
3. ***Clock Pulse 3:*** The third clock pulse operates simultaneously on all three segments. It transfers
   (a) A3 and B3 into R1 and R2
   (b) C3 and D3 into R3 and R4
   (c) The product of R1 and R2 into R5
   (d) The product of R3 and R4 into R6
   (e) The sum of R5 and R6 in R7

Therefore, it takes three clock pulses to fill up the pipeline and get the first output from R7. After these three clock pulses, each clock pulse produces a new output and moves the data one step

Notes

down the pipeline. This continues until new input data flows into the system, and even if the input of data stops, the clock must continue till the last output emerges from the pipeline.

Table 14.2 depicts the contents of the registers.

Table 14.2: Contents of Registers

Clock Pulse	Segment 1		Segment 2		Segment 3		Segment 4
	R_1	R_2	R_3	R_4	R_5	R_6	R_7
1	A_1	B_1	C_1	D_1			
2	A_2	B_2	C_2	D_2	A_1*B_1	C_1*D_1	
3	A_3	B_3	C_3	D_3	A_2*B_2	C_2*D_2	$A_1*B_1+C_1*D_1$
4					A_3*B_3	C_3*D_3	$A_2*B_2+C_2*D_2$
5							$A_3*B_3+C_3*D_3$

The performance parameters that are important with respect to pipelining are speedup, efficiency and throughput.

Speedup

Speedup is defined as the time in pipeline process over the time in non pipeline process.

$$S = \frac{T_1}{T_k} = \frac{n*k}{k+(n-1)}$$

Time in pipeline process is $\mathbf{T_k = k + (n-1)}$ periods, where **k** is the cycle used to fill up the pipeline or to complete the execution of the first task and $\mathbf{n-1}$ number of cycles are needed to complete the remaining **n " 1** tasks. The same number of tasks can be executed in a non-pipeline processor with an equivalent function in $\mathbf{T_1 = n*k}$ time delay.

The maximum speedup that a linear pipeline provides is **k**, where **k** is the number of stages in the pipe. It should be noted that the maximum speedup is $\mathbf{S_k \rightarrow k}$, for **n >> k**. Due to data dependencies between instructions, interrupts, and other factors, the maximum speedup is difficult to achieve.

Efficiency

The percentage of busy time-space spans over the total time-space span gives the efficiency of a linear pipeline. This is equal to the sum of all busy and idle time-space spans. Let **n**, **k** be the number of instructions, the number of pipeline stages, and the clock period of a linear pipeline, respectively. The pipeline efficiency is defined by:

$$\eta = \frac{n}{k+(n-1)}$$

Throughput

Throughput can be defined as the number of results that can be completed by a pipeline per unit time. This helps us to understand the computing power of a pipeline. In terms of efficiency η and clock period τ of a linear pipeline, we define the throughput as follows:

$$w = \frac{\eta}{\tau} = \frac{n}{k\tau+(n-1)\tau}$$

Notes

14.1.1 Pipelining Conflicts

The potential increase in performance, results only when pipelining is proportional to the number of pipeline stages. However, this increase would be achievable only if pipelined operations are performed without any interruption throughout program execution. The ideal situation is not so, the potential increase can be delayed for some reason. The pipeline stages may not always be able to complete its operations in the set time.

Consider that the execution stage E is responsible for arithmetic and logic operations. The set time for this operation would be one clock period. However, an operation like division can take more than one clock cycle and require more time to complete execution. Figure 14.3 shows an example of an instruction requiring three cycles to complete, from cycle 4 through cycle 6.

Figure 14.3: Example of Pipelining Conflict

Clock cycle	1	2	3	4	5	6	7	8	9
Instruction									
I1	F1	D1	E1	S1					
I2		F2	D2	E2			S2		
I3			F3	D3			E3	S3	
I4				F4			D4	E4	S4
I5							F5	D5	E5

As seen in figure 14.3, in cycles 5 and 6, the information in buffer must wait until the instruction execution stage has completed its operation. This blocks stage2 and in turn, stage 1 from accepting new instructions because the information in B1 cannot be overwritten. Thus, decode step for instruction and fetch step for instruction 5 is delayed. A phenomenon that causes the pipeline to stall is called a hazard or conflict.

Types of Conflicts

There are three types of conflicts:

1. Resource conflicts
2. Data dependency conflicts
3. Branch difficulties

Let us now discuss these conflicts.

Resource Conflicts

These conflicts arise due to insufficient resources wherein it is not possible to overlap the operations. The performance of pipelined processor depends on either of the two conditions. They are:

1. Whether or not the functional units are pipelined?
2. Do the multiple execution units allow all possible combination of instructions in the pipeline?

If for a particular combination, pipeline is stalled to avoid the resource conflicts, then there is a structural hazard.

Structural hazard occurs if two instructions require the use of a given hardware resource at the same time. The most common situation in which this hazard occurs is when resources request for memory. Consider a situation where one instruction needs to access memory for storage of the result while another instruction is being fetched. If the instructions and data reside in the same cache unit, only one instruction can proceed and the other instruction is delayed. To avoid this conflict many processors use separate caches for instruction and data.

Data Dependency Conflicts

These conflicts arise when the instruction in the pipeline depends on the result of the previous instructions and these instructions are still in pipeline and are not executed yet. When either the source or the destination operand of an instruction is not available at the expected time in the pipeline, the pipeline is stalled. Such a situation is termed as a data hazard or data conflict.

Consider a program with two instructions, I1 followed by I2. When this program is executed in a pipeline, the execution of these two instructions is performed concurrently. If the result of I1 and I2 are dependent on each other, then the result of I1 may not be available for the execution of I2.

Example: Consider the following operations to understand the data dependency conflict:

I1: A ê A+5

I2: B ê A*2

The result of I2 is dependent on the result of I1, therefore we may get incorrect result if both are executed concurrently.

Assume A = 10, if the given operations are performed sequentially, the result obtained will be 30. But if they are performed concurrently, the value of A used while computing B would be its original value 10, which leads to an incorrect result. The conflict due to such a situation is called data conflict or data dependent conflict. To avoid incorrect result, the dependent instructions are to be executed one after the other (in order).

Branch Difficulties

This difficulty is faced when branch and other instructions change the contents of program counter. There are two types of branches - conditional and unconditional. Conditional branches may or may not cause branching but an unconditional branch always causes branching. This difficulty is termed as control hazard.

The following critical actions can be followed during pipelining process which helps in handling control hazard:

1. Timely detection of a branch instruction
2. Early calculation of branch address
3. Early testing of branch condition (fate) for conditional branch instructions

14.1.2 Techniques for Overcoming Pipelining Conflicts

The following are the techniques that can be followed to avoid pipelining conflicts:

1. ***Hardware Interlocks***: Hardware interlocks are electronic circuits that detect instructions whose source operands are destinations of instructions further up in the pipeline. After detecting this situation, the instruction whose source is not available is delayed by a suitable number of clock periods. In this way, the conflict is resolved.
2. ***Operand Forwarding:*** This procedure uses a special hardware to detect a conflict and circumvent it, by routing the data through special paths between pipeline segments. This method requires additional hardware paths through MUXs (multiplexers).

Notes

3. ***Delayed Branching:*** In this procedure, the compiler is responsible for resolving the pipelining conflicts. The compiler detects the branch instructions and organizes the machine language code sequence by inserting useful instructions that keep the pipeline functioning without obstructions.

4. ***Branch Prediction:*** This method utilizes some sort of intelligent forecasting through appropriate logic. A pipeline with branch prediction guesses the result of a conditional branch instruction before it is executed. The pipeline fetches the stream of instructions from a predicted path, thus saving the time that is wasted by branch penalties.

5. ***Speculative Execution***: The advantage of branch prediction is that it helps the instructions following the branch to be fetched and executed without any delay. However, this must be done on a speculative basis. Speculative execution means that instructions are executed before the processor is certain that they are in the correct execution path. Hence, care must be taken that no processor registers or memory locations are updated until it is confirmed that these instructions have to be executed. If the branch decision indicates otherwise, the instructions and all their associated data in the execution units must be purged and the correct instructions must be fetched and executed.

14.2 Instruction Pipeline

An instruction pipeline reads consecutive instructions from memory while in the other segments the previous instructions are being executed. Pipeline processing occurs both in the data stream and in the instruction stream. This leads to the overlapping of the fetch and execute instruction and hence simultaneous operations are performed. One possible extra activity associated with such a scheme is that an instruction may cause a branch out of a sequence. In which case, the pipeline is emptied and all the instructions that have previously been read from memory after the branch instruction should be discarded.

A computer can be designed to provide a two segment unit, with an instruction fetch unit and an instruction execution unit. By means of a first-in, first-out (FIFO) buffer the instruction fetch segment is implemented. This is a type of unit forming a queue rather than a stack. When the execution unit is not accessing the memory, the control increments the program counter and uses its address value to read consecutive instructions from memory. The instructions are inserted into the FIFO buffer so that the execution occurs on a FIFO basis. Thus an instruction stream can be placed in a queue to wait for decoding and processing by the execution segment. Therefore, we can say that the instruction stream queuing mechanism provides an efficient way for reducing the average access time for memory to read instructions. Whenever there is space in the FIFO buffer, the control unit initiates the next instruction fetch phase. The buffer acts as a queue from which control then extracts the instructions for the execution unit.

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

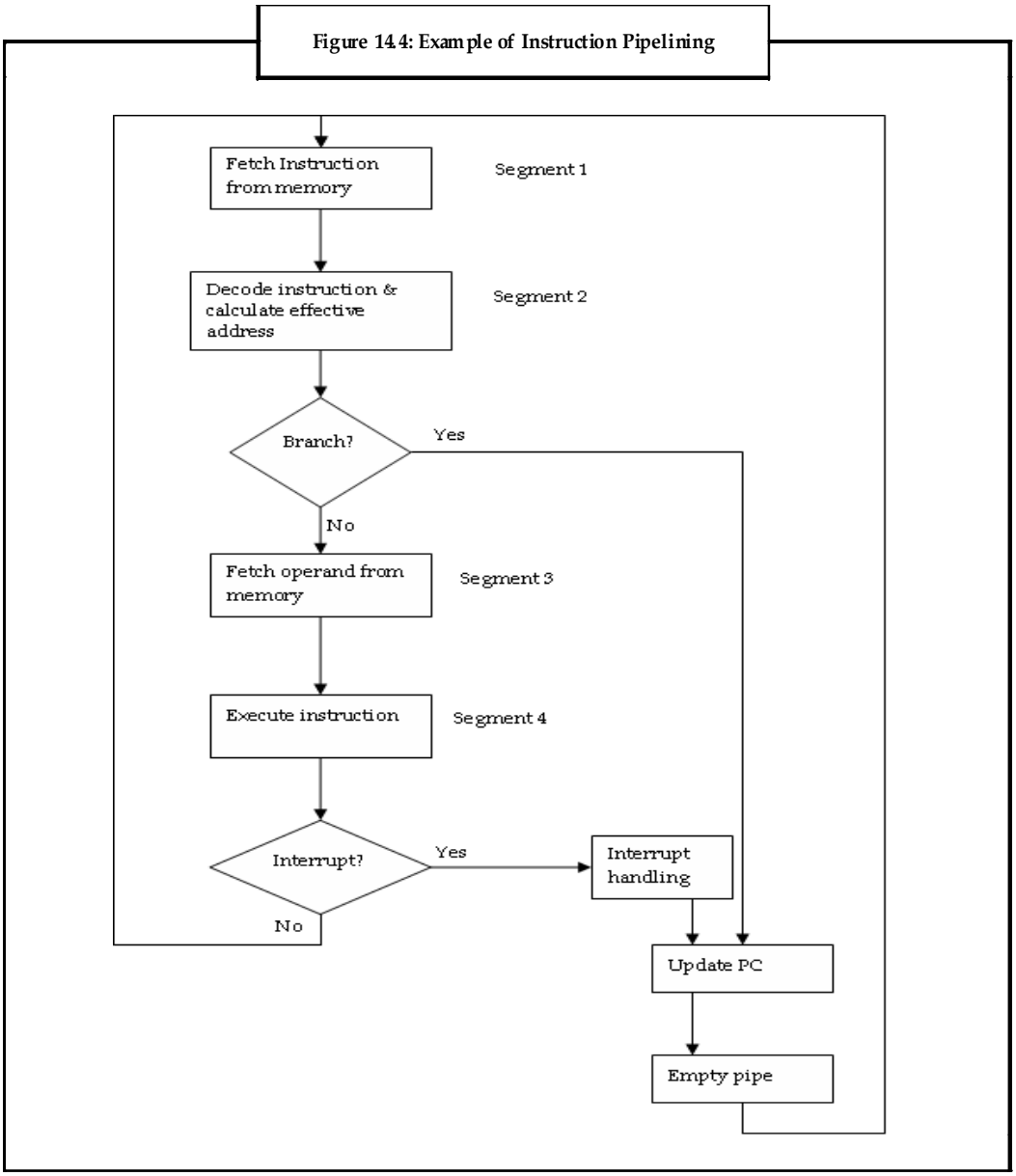
1. Fetch instruction from memory.
2. Decode instruction.
3. Calculate effective address.
4. Fetch operands from memory.
5. Execute instruction.
6. Store result in the proper place.

Different segments can take different time durations to operate on the incoming information. This could prevent the instruction pipeline from operating at its maximum speed. Some segments are skipped for certain operations.

Notes

Example: A register mode instruction does not need an effective address calculation. But at certain times two or more segments may require memory access at the same time, causing one segment to wait until another is finished with the memory. By using two memory buses for accessing instructions and data in separate modules, memory access conflicts are sometimes resolved. In this way, an instruction word and a data word can be read simultaneously from two different modules. Figure 14.4 illustrates an example of instruction pipelining.

Figure 14.4: Example of Instruction Pipelining



The time that each step takes to fulfill its function depends on the instruction and the way it is executed. The design of an instruction pipeline would be most efficient if the instruction cycle is divided into segments of equal duration.

Notes

Example: **Four-Segment Instruction Pipeline**

The instruction pipeline can be reduced to four segments by making two assumptions. The first assumption is that the decoding of the instruction can be combined with the calculation of the effective address and made into one segment. The second assumption is that most of the instructions places the result into a processor register so that the instruction execution and storing of the result can be combined into one segment.

Figure 14.4 depicts how the instruction cycle in the CPU can be processed with a four-segment pipeline. The simultaneous operations which can take place are, while an instruction is being executed in segment 4, the next instruction in sequence is busy fetching an operand from memory in segment 3. The effective address calculation can be done in a separate arithmetic circuit for the third instruction, and when the memory is available, the fourth and all subsequent instructions can be fetched and placed in an instruction FIFO.

Therefore maximum up to four sub-operations in the instruction cycle can overlap and up to four different instructions can be processed at the same time. At times, an instruction in the sequence may be a program control type that causes a branch out of normal sequence. In such a case, the pending operations in the last two segments are completed and all information stored in the instruction buffer is deleted. The pipeline then restarts from the new address stored in the program counter. Similarly, an interrupt request when acknowledged causes the pipeline to empty and start again from a new address value.

The operation of the instruction pipeline is depicted in table 14.3.

Table 14.3: Instruction Pipeline with Branch

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

The four segments in the instruction pipeline are represented in the table 14.3 with the following abbreviated symbols:

1. Fl is the segment to fetch an instruction.
2. DA is the segment to decode the instruction and calculate the effective address.
3. FO is the segment to fetch the operand.
4. EX is the segment to execute the instruction and store result.

The time in the horizontal axis is divided into steps of equal duration. By assuming that the processor has separate instruction and data memories, the operation in Fl and FO can proceed at the same time.

Each segment operates on different instructions in the absence of a branch instruction. In step 4, instruction 1 is being executed in segment EX; the operand for instruction 2 is being fetched in segment FO; instruction 3 is being decoded in segment DA; and instruction 4 is being fetched from memory in segment FI.

Assume instruction 3 as a branch instruction. When this instruction is decoded in segment DA in step 4, the transfer from FL to DA of the other instructions is halted until the branch instruction is executed in step 6. If the branch is taken into consideration, a new instruction is fetched in step 7,

else, the instruction fetched previously in step 4 can be used. The pipeline then continues until a new branch instruction is encountered.

Another delay likely to occur in the pipeline is when the EX segment needs to store the result of the operation in the data memory while the FO segment needs to fetch an operand. In which case, segment FO must wait until segment EX has finished its operation.

Task Consider a five stage pipeline, with an extra execution task of storing the result at a particular memory location. Assume the branch instruction to be taken into consideration at instruction 4. If so, at which stage will the operation end if the total numbers of instructions are 8?

14.3 RISC Pipeline

Reduced Instruction Set Computers (RISC) was introduced to execute as fast as one instruction per clock cycle. This RISC pipeline helps to simplify the computer architecture's design. It relates to what is known as the Semantic Gap, that is, the difference between the operations provided in the high level languages (HLLs) and those provided in computer architectures. Generally, wider the semantic gap, larger the number of undesirable consequences, which are execution inefficiency, excessive machine program size, and increased compiler complexity.

To avoid these consequences, the conventional response of the computer architects is to add layers of complexity to newer architectures. This also increases the number and complexity of instructions together with increase in the number of addressing modes. The architecture which resulted from the adoption of this "add more complexity" are known as Complex Instruction Set Computers (CISCs). But this brought in disadvantages like a complex instruction decoding scheme, an increased size of the control unit, and increased logic delays. The advantage of RISC over CISC is that RISC achieves pipeline segments in just one clock cycle, but CISC uses many segments in its pipeline, requiring two or more clock cycles for the longest segment.

The significant advantageous capability attributed to RISC is the use of an efficient instruction pipeline. Using a small number of sub operations, the simplicity of the instruction set can be used to implement an instruction pipeline with each sub-operation being executed in one clock cycle. The fixed length instruction format helps the decoding of the operation to occur at the same time as the register selection. The data manipulation instructions have register-to- register operations.

All operands are in registers, hence there is no need for calculating an effective address or fetching of operands from memory. Therefore, the instruction pipeline can be implemented with two or three segments. One segment fetches the instruction from program memory, and the other segment executes the instruction in the ALU. A third segment may be used to store the result of the ALU operation in a destination register.

The data transfer instructions use register indirect addressing. These instructions in RISC are limited to load and store instructions. They require three or four stages in the pipeline. To prevent conflicts between a memory access and to load or store an operand, most RISC machines use two separate buses with two memories. The two memories can sometimes operate at the same speed as the CPU clock and are referred to as cache memories wherein one memory stores the instructions and the other stores the data.

The said major advantage of RISC to execute instructions at the rate of one per clock cycle is always not possible, since every instruction cannot be fetched from memory and executed in one clock cycle ideally under all circumstances. The method to achieve the execution of an instruction per clock cycle is to start each instruction with each clock cycle and to pipeline the processor to achieve the goal of single-cycle instruction execution.

RISC compiler gives support to translate the high level language program into machine language program. Problems in handling difficulties in relation to data conflicts and branch penalties are taken care by the RISC processors, which relies on the efficiency of the compiler to detect and minimize the delays encountered with these problems.

Notes

RISCs Design Principles

The RISC design principles are:

1. Keep the most frequently accessed operands in CPU registers.
2. Minimize the register-to-memory operations.

These principles can be achieved using the following mechanisms:

1. Use a large number of registers to optimize operand referencing and reduce the processor memory traffic.
2. Optimize the design of instruction pipelines such that minimum compiler code generation can be achieved.
3. Use a simplified instruction set and leave out those complex and unnecessary instructions.

Let us consider a three segment instruction pipeline which shows how a compiler can optimize the machine language program to compensate for pipeline conflicts.

A typical set of instructions for a RISC processor is of three types. They are:

1. ***Data Manipulation Instructions***: Manage the data in processor registers.
2. ***Data Transfer Instructions:*** These are load and store instructions which use an effective address that is obtained by adding the contents of two registers or a register and a displacement constant provided in the instruction.
3. ***Program Control Instructions:*** These instructions use register values and a constant to evaluate the branch address, which is transferred to a register or the program counter (PC).

Now, consider the hardware operation for a computer with RISC processor. The control section fetches the instruction from program memory into an instruction register. The instruction is decoded at the same time that the registers needed for the execution of the instruction are selected. The processor unit consists of a number of registers and an arithmetic logic unit (ALU) that performs the necessary arithmetic, logic, and shift operations. A data memory is used to load or store the data from a selected register in the register file.

The instruction cycle can be divided into three sub operations and implemented in three segments:

I: Instruction fetch

A: ALU operation

E: Execute instruction

The I segment fetches the instruction from program memory. The instruction is decoded and an ALU operation is performed in the A segment. The ALU is used for three different functions. Depending on the decoded instruction, the ALU performs the following functions:

1. An operation for a data manipulation instruction
2. An evaluation of the effective address for a load or store instruction
3. A calculation of the branch address for a program control instruction

The E segment directs the output of the ALU to one of a destination, depending on the decoded instruction. It transfers the result of the ALU operation into a destination register in the register file, it transfers the effective address to a data memory for loading or storing, or it transfers the branch address to the program counter.

Notes

14.4 Vector Processing

High computational power is a never ending requirement. Certain classes of computational problems are beyond the ability of a conventional computer. The scientific and research computations involve many computations which require extensive and high power computers. These computations when run in a conventional computer may take days or weeks to complete. The science and engineering problems can be formulated in terms of vectors and matrices using vector processing.

The following are some examples or application areas in which vector processing is used.

1. Radar and signal processing to detect space/underwater targets
2. Remote sensing for earth resources exploration
3. Wind tunnel experiments
4. Medical diagnosis
5. Long-range weather forecasting
6. Petroleum explorations
7. Seismic data analysis
8. Medical diagnosis
9. Aerodynamics and space flight simulations
10. Artificial intelligence and expert systems
11. Mapping the human genome
12. Image processing

To achieve high performance in all these processes, a fast, reliable hardware and innovative procedures from vector and parallel processing techniques have to be used.

14.4.1 Characteristics of Vector Processing

A vector is a structured set of elements. The elements in a vector are scalar quantity. A vector operand contains an ordered set of **n** elements, where **n** is called the length of the vector. Each clock period processes two successive pairs of elements. During one single clock period the dual vector pipes and the dual sets of vector functional units allow the processing of two pairs of elements. As the completion of each pair of operation takes place, the results are delivered to appropriate elements of the result register. The operation continues until the number of elements processed is equal to the count specified by the vector length register.

In parallel vector processing, more than two results are generated per clock cycle. The parallel vector operations are automatically initiated under the following two circumstances:

1. Firstly, when successive vector instructions use different functional units and different vector registers.
2. Secondly, when successive vector instructions use the result stream from one vector register as the operand of another operation using a different functional unit. This process is termed as chaining.

A vector processor performs better with longer vectors due to the startup delay in a pipeline.

Vector processing reduces the overhead associated with maintenance of the loop-control variables which makes it more efficient than scalar processing.

14.4.2 Advantages of Vector Processing

Although vector processing is not for general purpose computation, it does offer significant advantages for scientific computing applications. The following are the advantages of using vector processing:

1. Flynn's bottleneck can be reduced by using vector instructions as each vector instruction specifies a lot of work. Vector instructions can specify the work equivalent of an entire loop.

Notes

Thus, fewer instructions are required to execute programs. This reduces the bandwidth required for instruction fetch.

Notes Flynn's bottleneck states that maximum processor element (PE) can process only one instruction per clock cycle.

2. Data hazards can be eliminated due to the structured nature of the data used by vector machines. We can determine the absence of a data hazard at compile-time, which not only improves performance but also allows for planned pre-fetching of data from memory.
3. Memory latency can be reduced by using pipelined load and store operations.

Example: When we fetch the first element in a 64-element addition operation, we can schedule fetching the remaining 63 elements from memory. By using interleaved memory designs, vector machines can amortize the high latency associated with memory access over the entire vector.

4. Control hazards are reduced as a result of specifying a large number of iterations in a single vector instruction. The number of iterations depends on the size of the vector registers.

Example: If the machine has 64-clement vector registers, each vector instruction can specify work equivalent to 64 loop iterations.

5. Pipelining can be exploited to the maximum extent. This is facilitated by the absence of data and control hazards. Vector machines not only use pipelining for integer and floating-point operations, but also to feed data from one functional unit to another. This process is known as chaining. In addition, as mentioned before, load and store operations also use pipelining.

Did u know? The most powerful computers of the 1970s and the 1980s were vector machines. But with increasingly higher degrees of semiconductor integration, the mismatch between instruction bandwidth and operand bandwidth essentially deteriorated. As of 2009, only one of the world's top 500 supercomputers was still based on vector architecture.

14.5 Parallel Processing

Parallel processing helps in the concurrent execution of many programs in the computer. It is in contrast to sequential processing. Parallel processing method of information processing emphasizes on the running of concurrent events during computing processes. Concurrency implies parallelism, simultaneity, and pipelining. The concurrent events can be attained in a computer system at various processing levels. Parallel events can occur in multiple resources during the same time interval, simultaneous events can occur at the same time instant, and pipelined events can occur in overlapped time spans. Therefore, parallel processing is a cost-effective means to improve system performance through concurrent activities in the computer.

Parallel processing can be done in four ways while considering the following programmatic levels:

1. Job or program level
2. Task or procedure level
3. Inter-instruction level
4. Intra-instruction level

The highest level of parallel processing, job or program level, requires the development of parallel processable algorithms. This level is conducted among multiple jobs or programs through multiprogramming, time sharing, and multiprocessing. The implementation of parallel algorithms depends on the efficient allocation of limited hardware-software resources to the multiple programs that are used to solve a large computational problem. The next highest level of parallel processing

is task or procedure level conducted among procedures or program segments within the same program. This involves the decomposition of a program into multiple tasks (segments). The third level is the inter-instruction level which brings in concurrency among multiple instructions. Data dependency analysis is often performed to reveal parallelism among instructions. In the last level, which is the intra-instruction level, faster and concurrent operations can be brought in within each instruction.

Parallel processing is part of the architectural trend of multiprocessor systems, which has shared memory space and peripherals under the control of one integrated operating system. Computer manufacturers started the development of systems with a single central processor called a uniprocessor system, but their goal is to achieve high performance.

Did u know? Parallel processing differs from multitasking, in which a single CPU executes several programs at once.

The general operation involved in a computer is to fetch instructions from memory and execute them in the processor and then store the results in the main memory. The sequence of instructions which is read from memory comprises an instruction stream flowing from memory to processor. Operation on data constitutes a data stream flowing to and from the processor.

According to Michael J Flynn, parallel processors can be divided into the following four groups based on the number of instructions and data streams that they have:

1. Single Instruction Stream Single Data stream (SISD)
2. Single Instruction Stream Multiple Data stream (SIMD)
3. Multiple Instruction Stream Single Data stream (MISD)
4. Multiple Instruction Stream Multiple Data stream (MIMD)

Let us now briefly discuss the four groups of parallel processors.

SISD Computer Organization

SISD represents a computer organization with a control unit, a processing unit and a memory unit. SISD is like the serial computer in use. SISD executes instructions sequentially and they may or may not have parallel processing capabilities. Instructions executed sequentially may get overlapped in their execution stages. An SISD computer may have more than one functional unit in it. But all the functional units are under the supervision of one control unit. Parallel processing in such systems can be attained by pipeline processing or by using multiple functional units.

SIMD Computer Organization

SIMD organization includes multiple processing elements. All these elements are under the supervision of a common control unit. All processors receive identical instruction from the control unit, but operate on different data items. The shared subsystem contains multiple modules which help in communicating with all the processors simultaneously. This is further divided into word slice and bit slice mode organizations.

MISD Computer Organization

MISD organization includes multiple processing units, each receiving separate instructions operating over the same data stream. The result of one processor becomes the input of the next processor. The introduction of this organization received less attention and was not practically implemented on architecture. The structure was of only theoretical interest.

MIMD Computer Organization

An MIMD computer organization involves interactions among the multi processors since all memory streams are derived from the common data space shared by all processors. If the multi data streams were derived from different shared memories then it is a multiple SISD operation which is equal to a set of '**n**' independent SISD systems.

Notes

An MIMD is tightly coupled when the degree of interactions among the processors is high, otherwise they are said to be loosely coupled. This kind of organization refers to a computer system capable of processing several programs simultaneously. Most multiprocessor computer systems come under this category.

14.6 Summary

- Parallel processing is a technique of executing several jobs simultaneously in order to enhance the speed of processing and throughput.
- Pipelining is a technique of dividing a sequential process into sub operations.
- The performance parameters important with respect to pipelining are speedup, efficiency and throughput.
- There are three major pipelining conflicts: resource conflicts, data dependency conflicts and branch difficulties
- The pipelining conflicts can be removed by the following schemes:
  - (a) Hardware interlocks
  - (b) Operand forwarding
  - (c) Delayed branching
  - (d) Branch prediction
  - (e) Speculative execution
- Computers with vector processing capabilities are in demand to run specialized applications involving computations which are beyond the capabilities of a conventional computer.
- Parallel vector processing permits the generation of more than two results per clock period.
- Flynn's classification of computers defines the following four major groups of parallel processing:
  - (a) SISD (Single Instruction stream, Single Data stream)
  - (b) SIMD (Single Instruction stream, Multiple Data stream)
  - (c) MISD (Multiple Instruction stream, Single Data stream)
  - (d) MIMD (Multiple Instruction stream, Multiple Data stream)

14.7 Keywords

Branch Penalty: The delay caused due to a branch instruction in a pipeline.

Combinational Circuit: A combinational circuit is one for which the output value is determined solely by the values of the inputs.

Concurrent: Executed at the same time.

Latency: Measure of time delay that is experienced in a system.

14.8 Self Assessment

1. State whether the following statements are true or false:
   - (a) The overlapping of processing is done by associating a combinational circuit with each segment in the pipeline.
   - (b) Even after the input of data stops, the clock must continue till the last output emerges out of the pipeline.
   - (c) The said major advantage of RISC to execute instructions at the rate of one per clock cycle is always not possible.

Notes

(d) Data hazard occurs if two instructions require the use of a given hardware resource at the same time.

(e) The percentage of the total time-space span over the busy time-space spans gives the efficiency of a linear pipeline.

(f) Pipeline processing occurs both in the data stream and in the instruction stream.

2. Fill in the blanks:

(a) _____ use register values and a constant to evaluate the branch address.

(b) _____ is defined as the number of results that can be completed by a pipeline per unit time.

(c) _____ method requires additional hardware paths through MUXs (multiplexers).

(d) _____ brings in concurrency among multiple instructions.

(e) _____ is the difference between the operations provided in the high level languages (HLLs) and those provided in computer architectures.

3. Select a suitable choice for every question.

(a) Which of the following level is conducted among multiple jobs or programs through multiprogramming, time sharing, and multiprocessing.

(i) Job or program level

(ii) Task or procedure level

(iii) Inter-instruction level

(iv) Intra-instruction level

(b) _____ are electronic circuits that detect instructions whose source operands are destinations of instructions further up in the pipeline.

(i) Hardware interlocks

(ii) Combinational circuits

(iii) Arithmetic circuit

(iv) Delayed branching

(c) _____ help in providing isolation between each segment so that every segment can work on distinct data simultaneously.

(i) Overlapping of processes

(ii) Registers

(iii) Memory interleaving

(iv) Speculative execution

(d) A pipeline with _____ guesses the result of a conditional branch instruction before it is executed.

(i) Branch prediction

(ii) Delayed branching

(iii) Open forwarding

(iv) Hardware interlocks

Notes

(e) Which computer organization is further divided into word slice and bit slice mode organizations?

(i) Single Instruction stream Single Data stream (SISD)

(ii) Single Instruction stream Multiple Data stream (SIMD)

(iii) Multiple Instruction stream Single Data stream (MISD)

(iv) Multiple Instruction stream Multiple Data stream (MIMD)

14.9 Review Questions

1. "A compiler can optimize the machine language program to compensate for pipeline conflicts." Justify with example.
2. "When compared with sequential processing, pipelining reduces the amount of time used." Discuss with the help of a time space chart.
3. "The pipeline stages are not always able to complete its operations in the set time." Explain with an example.
4. "Reduced Instruction Set Computers (RISC) was introduced to execute as fast as one instruction per clock cycle." Explain in detail.
5. "Resource conflict arises due to insufficient resources where in it is not possible to overlap the operations." Discuss considering a situation.
6. "The instruction pipeline can be reduced to segments by making assumptions." Explain with example.
7. "Pipelining conflicts can be overcome." Discuss with regard to techniques.
8. "Memory latency can be reduced by using pipelined load and store operations." Explain with example.
9. "When the pipeline is stalled due to the unavailability of either the source or the destination operand at the expected time it is termed as a data hazard." Discuss with example.
10. "According to Flynn, Parallel processors can be divided into four groups based on the number of instructions and data streams." Name the types and explain.
11. "The advantage of RISC over CISC is that RISC achieves pipeline segments in just one clock cycle." Justify and discuss the differences between the two.
12. "Different sets of addresses can be assigned to different memory with the help of interleaving" Explain.

Answers: Self Assessment

1. (a) False (b) True (c) True (d) False (e) False (f) True
2. (a) Program control instructions (b) Throughput (c) operand forwarding (d) Inter-instruction level (e) Semantic gap
3. (a) Job or program level (b) Hardware interlocks (c) Registers (d) Branch prediction (e) SIMD

14.10 Further Readings

Notes

Books

Stallings, W. (2009). Computer Organisation and Architecture: Designing for performance. Prentice Hall.

Null,L., & Lobur, J. (2006). The essentials of Computer Organization and Architecture. Jones & Barlett Learning.

Godse, A.P., & Godse D.A. (2008). Digital Electronics. Pune: Technical Publications

Online links

http://brahms.emu.edu.tr/rza/chapter1.pdf

http://www.cs.unb.ca/profs/gdueck/courses/Ch_12.ppt.pdf

http://www.scribd.com/doc/45907864/Pipeline-and-Vector-Processing