Published By :

BET Publications

For BHAVYA BOOKS (BET)™

New Delhi

Mob. 9313 852 843

Website : www.bhavyabooks.com

E-mail : bhavyabooks@gmail.com

© ALL RIGHTS STRICTLY RESERVED BY PUBLISHER (BHAVYA BOOKS)

STATUTORY WARNING

All Rights Reserved. No part of this publication may be reproduced, stored in a Retrieval System, or Transmitted in any form or by any means : Electronic, Mechanical, Photocopy, Recording or Otherwise, Without the Prior Written Permission From the Publisher. Any Legal Issues will be Dealt in Delhi.

The Authors of the Book have taken all possible care to ensure that the contents of the Book do not violate any existing copyright or other intellectual property rights of any person in any manner whatsoever. In the event, the Authors have been unable to track any source and if any copyright has been inadvertently infringed, the facts may be brought to the notice of the publisher in writing for corrective action.

First Edition : 2015

Contents

1. INTRODUCTION TO COMPUTER ARCHITECTURE AND ORGANIZATION ..... 1 – 19

1.1. Computing 1
1.2. Computers 2
1.2.1. Characteristics of a Computer 2
1.2.2. Computer Types and Classification 3
1.2.3. Digital Computer 3
1.2.4. Generation of Digital Computers 4
1.2.5. Layers in the Modern Computer 5
1.3. Computer Architecture and Organization 6
1.4. Elements of a CPU 6
1.4.1. CPU Registers 6
1.4.2. Clock 7
1.4.3. CPU Busses 8
1.5. Computer Organization and Stored Program Concept 8
1.6. Performance 11
1.6.1. Processor Speed 12
1.6.2. CPU Time 12
1.6.3. Performance Metrics 13
1.7. Speed Up 16
1.8. Amdahl's Law 17
Solved Questions With Answers 18
Review Questions 19

2. FUNDAMENTALS OF DIGITAL ELECTRONICS ..... 20 – 57

2.1. Number System 20
2.1.1. Binary Number System 21
2.1.2. Octal Number System 21
2.1.3. Decimal Number System 21
2.1.4. Hexadecimal Number System 22
2.1.5. Conversions 22
2.2. Complements 29
2.2.1. 1's Complement 29
2.2.2. (r – 1)'s Complement 30
2.2.3. Subtraction Using 2's Complement 31
2.3. Fixed Point Numbers 32
2.3.1. Signed Binary Numbers 33
2.4. Floating Point Numbers 37
2.4.1. IEEE Standards of Floating Point Representation 38
2.5. Logic Gates 44
Solved Examples 51
Review Questions 56

3. MICROOPERATIONS AND DESIGN OF AN ALU ..... 58 – 86

3.1. Register Transfer and Representation 58
3.2. Microoperations 60
3.3. Register/Data Transfer Microoperations 60
3.4. Arithmetic microoperations 62
3.5. Logical microoperations 67
3.5.1. Selective Microoperations 69
3.6. Shift microoperations 70
3.7. Bus and Memory Transfer 72
3.7.1. Bus Transfer 72
3.7.2. Memory Transfer 75
3.7.3. Bus Organization 75
3.7.4. Bus Architecture 77
3.7.5. Bus Arbitration 79
3.8. Design of an Arithmetic Logic Shift Unit 81
Solved Examples 83
Review Questions 86

4. INTRODUCTION TO MICROPROCESSOR ..... 87 – 126

4.1. Introduction 87
4.2. Intel's Microprocessors 88
4.3. Internal Structure of Microprocessor 98
4.4. Supporting Devices of Microprocessor 98
4.4.1. Clocks and Reset Circuits 99
4.4.2. Buffers 99
4.4.3. Latches 99
4.4.4. Address Decoders 99
4.4.5. Wait State Generators 99
4.4.6. Interrupt Controllers 99
4.4.7. Timers and Counters 99
4.4.8. I/O Interfaces 99
4.4.9. DMA Controllers and Bus Arbitration 100
4.4.10. Memory 100
4.4.11. Buses 100
4.5. Levels of Programming Languages 100
4.5.1. Machine Language 101
4.5.2. Assembly Language 101
4.5.3. Compiler Language 102
4.5.4. Interpreter Language 102
4.6. Introduction to 8085 102
4.7. Features of 8085 Microprocessor 102

www.IPUnotes.com

4.8. Internal Architecture and Functional Block Diagram of 8085 103
4.8.1. Registers 104
4.8.2. Accumulator 104
4.8.3. Register Pairs 104
4.8.4. Stack Pointer 104
4.8.5. Program Counter 104
4.8.6. Instruction Register/Decoder 104
4.8.7. Memory Address Register 104
4.8.8. Flag Register 104
4.8.9. 8085 System Bus 105
4.9. Pin Configuration of 8085 105
4.9.1. Pin Function Description 106
4.9.2. Signal Groups of Microprocessor 8085 109
4.10. Addressing Modes of 8085 109
4.10.1. Immediate Addressing 109
4.10.2. Register Addressing 110
4.10.3. Direct Addressing 110
4.10.4. Indirect Addressing 110
4.11. Instruction Format 110
4.11.1. One-Byte Instructions 110
4.11.2. Two-Byte Instructions 111
4.11.3. Three-Byte Instructions 111
4.12. Instruction set of 8085 112
4.12.1. Data Transfer Instructions 112
4.12.2. Arithmetic Instructions 114
4.12.3. Logical Instructions 116
4.12.4. Branch Instructions 119
4.12.5. Stack I/O Instructions 122
4.12.6. I/O Instructions 122
4.12.7. Machine Control Instructions 122
Solved Examples 124

5. BASIC COMPUTER ORGANIZATION AND DESIGN ..... 127—172

5.1. Instruction Code 127
5.1.1. Store Program Organization 128
5.1.2. Direct Address and Indirect Address 128
5.2. Computer Registers 130
5.3. Computer Instructions 131
5.4. Timing and Control 134
5.5. Instruction Cycle 135
5.5.1. Flow Chart of Instruction Cycle 137
5.6. Input, Output and Interrupts 137
5.7. Control Unit 139
5.7.1. Hardwired Control unit 140
5.7.2. Micro programmed Control Unit 140
5.8. Hardwired Control Unit 140
5.8.1. Advantage of Hardwired Control Unit 142
5.8.2. Disadvantages of Hardwired Control Unit 142
5.9. Specifying a CPU 142
5.10. Design and Implementation of a Simple CPU 142
5.10.1. Specifications for a Very Simple CPU 143
5.10.2. Fetching Instructions from Memory 143
5.10.3. Decoding Instructions 144
5.10.4. Executing Instructions 145
5.10.5. Establishing Required Data Paths 145
5.10.6. Design of a Very Simple ALU 147
5.10.7. Designing the Control Unit Using Hardwired Control 150
5.10.8. Design Verification 152
5.11. Design of Accumulator Logic 152
5.12. Microprogrammed Control Unit 156
5.12.1. Control Memory 157
5.12.2. Address Sequencing 157
5.12.3. Microinstruction Format 158
5.12.4. Micro-program Sequencer 159
5.13. Difference Between Hardwired and Microprogrammed Control Unit 161
5.14. Micro Code 161
5.15. Complete Computer Description 162
5.16. Pentium Microprocessor 163
5.16.1 Architecture of Pentium Processor 164
5.17. Advanced Pentium Processors 167
5.17.1. Pentium PRO 167
5.17.2. Pentium II 167
5.17.3. Pentium III 168
5.17.4. Pentium IV 168
Solved Examples 169
Review Questions 172

6. CPU ARCHITECTURE AND INSTRUCTION SET ..... 173—209

6.1. CPU Organization 174
6.2. General Register Set Organization 175
6.3. Stack Organization 177
6.4. Single Accumulator Organization 182
6.5. Instruction Format 182
6.5.1. General-Register Organization 183
6.5.2. Stack Organization 184
6.5.3. Accumulator-Type Organization 184
6.6. Instruction Set Based Classification of Processors 187
6.6.1. Reduced Instruction Set Computer (RISC) 187
6.6.2. Complex Instruction Set Computer (CISC) 190
6.6.3. Comparison between RISC and CISC 192
6.7. Addressing Modes 193
6.7.1. Implied Addressing 194
6.7.2. Immediate Addressing 194
6.7.3. Direct Addressing 195
6.7.4. Indirect Addressing 195
6.7.5. Register Addressing 195
6.7.6. Register Indirect Addressing 196
6.7.7. Relative Addressing 197
6.7.8. Indexed Addressing 197
6.7.9. Base Register Addressing 198
6.7.10. Auto Increment and Auto Decrement Addressing 199
Solved Examples 201
Review Questions 209

7. PIPELINE AND VECTOR PROCESSING ..... 210—232

7.1. Parallel Computer Systems 210
7.1.1. Instruction-Level Parallel Processors (ILP) 211
7.1.2. Multiprocessors 211
7.2. Classifications Of Parallelism 212
7.2.1. Flynn's Classification based on the Multiplicity of Instruction and Data Streams (1966) 212
7.2.2. Classification based on Coupling between Processing Elements 213
7.3. Instruction Level Parallelism 213
7.3.1. Pipelining 215
7.4. Types of Pipelining 217
7.4.1. Arithmetic Pipelining 217
7.4.2. Instruction Pipelining 219
7.4.3. RISC Pipelining 221
7.4.4. Pipeline Hazards 224
7.5. Processor Level Parallelism 227
7.5.1. Vector Processor 227
Solved Examples 230
Review Questions 232

8. COMPUTER ARITHMETIC ..... 233—252

8.1. Arithmetic Types 233
8.2. Representation of a Number 234
8.2.1. Fixed Point Representation 235
8.3. Fixed Point Arithmetic 235
8.3.1. Addition and Subtraction 235
8.3.2. Multiplication 240
8.3.3. Division 246
Solved Examples 250

9. MEMORY ORGANIZATION ..... 253—283

9.1. Memory Hierarchy 253
9.2. Characteristics/Parameters of Various Memory Devices 255
9.3. Main Memory 257
9.3.1. RAM and ROM Chips 257
9.3.2. Primary Memory 259
9.4. High Speed Memories 265
9.4.1. Interleaved Memory 265
9.4.2. Cache Memory 266
9.4.3. Associative Memories 273
9.4.4. Virtual Memory 273
Solved Problems 277
Review Questions 283

10. INPUT-OUTPUT ORGANIZATION ..... 284—308

10.1. Peripheral Devices 284
10.1.1. Keyboards 284
10.1.2. Mouse 285
10.1.3. Display Devices 285
10.1.4. Printers 285
10.2. Input-Output Interface 285
10.2.1. I/O Versus Memory Bus 287
10.2.2. Isolated Versus Memory-Mapped I/O 287
10.3. Data Transfer Schemes 287
10.3.1. Programmed Data Transfer 288
10.3.2. Synchronous Data Transfer Scheme 289
10.3.3. Asynchronous Data Transfer Scheme 290
10.3.4. Interrupt Driven Data Transfer 290
10.4. Asynchronous Data Transfer 291
10.4.1. Strobe Control 291
10.4.2. Handshaking 292
10.5. Direct Memory Access (DMA) 294
10.5.1. DMA Controller 295
10.5.2. DMA Initialization 296
10.5.3. DMA Transfer 296
10.6. Serial Communication 297
10.6.1. Serial Communication Basics 297
10.6.2. Universal Asynchronous Receiver/Transmitters (UARTs) 300
10.7. Real World Example: Serial Communication Standards 301
10.7.1. The RS-232-C Standard 301
10.7.2 The Universal Serial Bus Standard 303
Solved Examples 304
Review Questions 308

APPENDIX A : Descriptive Questions ..... 309—333
APPENDIX B : Short Answer Type Questions ..... 334—338
INDEX ..... 339—342

www.PTUnotes.com

Introduction to Computer Architecture and Organization

1

Today computers are used in almost all walks of life: banks, hospitals, schools, publishing, manufacturing, entertainment, shops, libraries, factories, travels, hotels, research, messaging, film production, games, meetings and conference, security services, attendance recording, accounting, elections, courts, universities, prisons etc.

www.IPUnotes.DUCE

Computer buyers consider a number of factors before choosing a product such as good performance, low cost, easy programmability, low maintenance cost, less downtime etc. It is the soul aim of the computer architecture and design engineers to provide these features.

The design of a computer is highly sophisticated and complex. Over the years, a variety of concepts and techniques have been applied in designing and building it. Factors such as performance, cost, component technology, types of use and capacity requirement determine the choice of concepts and techniques used in designing a computer. The study of these concepts and techniques is the objective of this book.

This chapter serves as a foundation to the following chapters. It outlines the basic organization of a computer in order to make the reader comfortable in learning about the various architectural and design issues covered in following chapters.

1.1. COMPUTING

The word 'computing' means 'an act of calculating'. The human race has so far seen three different types of computing:

1. Fully manual computing without any tools other than brain and fingers
2. Manual computing using simple tools such as slide rule, abacus etc
3. Automatic computing using a computer machine.

Table 1.1 compares the three types with respect to the following parameters:

1. Speed of computing;
2. Reliability of results;
3. Complexity of problem that can be handled;
4. Extent of human effort involved;
5. Consistency.

Table 1.1. Comparison of Computing Types.

S. No.	Parameter	Manual computing	Manual computing with simple tool	Automatic computing
1.	Speed	Low	Medium	High
2.	Reliability	Poor; varies with individual	Medium	High
3.	Problem complexity possible	Low	Medium	High
4.	Extent of human effort	Very high	Medium	Very low
5.	Consistency	Poor due to tiredness or moodiness	Medium	Very high
6.	Impact	Many problems cannot be solved in reasonable time	Slightly better than manual computing	Any reasonable problem can be solved

www.IPUnation.com

1.2. COMPUTERS

The computer is a tool for solving problems (Fig. 1.1). Its basic ability is to perform arithmetic calculations.

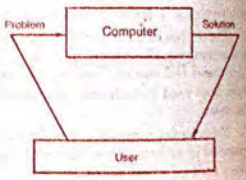


Fig. 1.1. User and computer

The computer is used to solve problems in several fields such as scientific, research, commerce, administration, manufacturing etc. The following are some simple uses:

1. Calculating the average mark of students of a class.
2. Preparing monthly salary statement for a factory.
3. Designing a building.
4. Calculating the tax payable by a business firm.

The various advantages of using a computer are as follows:

1. Quick processing (calculations)
2. Large information storage
3. Relieving manual efforts
4. Messaging and communication

1.2.1. Characteristics of a Computer

The main characteristics of a computer are listed below:

1. Very high speed of computation.

2. Consistency of behavior unaffected by fatique, boredom, likes and dislikes etc.
3. Large storage capacity (for data and programs).
4. High accuracy of computation.
5. General purpose machine which can be programmed as per user's requirement.

1.2.2. Computer Types and Classification

Computers are classified as

1. Based on the processing capability: The following are the types of computers based on the processing capability:

(*a*) Microcomputer
(*b*) Minicomputer
(*c*) Mainframe computer and
(*d*) Super computer

2. Based on the computation: The following are the types of computer based on the computation namely:

(*a*) Analog computer
(*b*) Digital computer
(*c*) Hybrid computer.

3. Based on the area of application: The following are the types of computers based on the area o applications namely

(*a*) General purpose computer and
(*b*) Special purpose computer.

1.2.3. Digital Computer

Most of the present day computers are digital computers though some are also analog computers. An analog computer monitors (senses) input signals whose values keep changing continuously. It generally deals with physical variables such as voltage, pressure, temperature, speed etc. A digital computer operates on discrete (digital) information such as numbers. It uses binary number system in which there are only two digits 0 and 1. Each is called a bit (binary digit). In chapter 2, we discusses the different number systems: decimal, binary, octal and hexa decimal. The digital computer is designed using digital circuits in which there are two different levels for an input or output signal. These two levels are known as logic 0 and 1. The modern computer is a digital computer capable of performing arithmetic (mathematical) and logical operations on data and generates a result (Fig. 1.2).

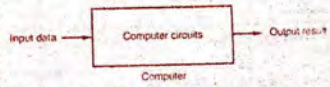


Fig. 1.2. Computer as an electronic machine.

To solve a problem, the user inputs a program to the computer (Fig. 1.3). The program has a sequence of instructions which specify the various operations to be done. Thus the program gives a

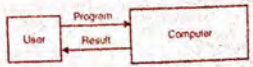


Fig. 1.3. Solving a problem.

step-by-step method to solve a problem. A computer can carry out a variety of instructions such as ADD, SUBTRACT, MULTIPLY, DIVIDE, COMPARE, MOVE, INPUT, OUTPUT etc. The entire list of instructions for a computer is known as instruction set. A program uses combinations of the instructions according to the method (algorithm) to solve the problem. A program can be defined as a sequence of instructions with appropriate data for solving a problem (Fig. 1.4). The computer analyses each instruction and performs action on data.

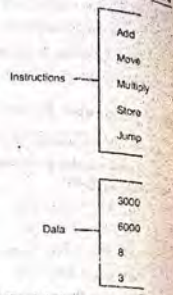


Fig. 1.4. A computer program

The computer program for the electric machine should consist of only 1's and 0's. The operations are specified in the instructions as binary information. The operands (data) are given as binary numbers. Such a program is called a machine language program. Though it is tedious to write programs in machine language, all early programs were written only in machine language as there was no other choice. Different computer models had different machine languages and hence a machine language program for one computer cannot run on the other computer different instruction set. A modern computer programmer does not program in the machine language and he has an easy alternate such as high level language.

1.2.4. Generation of Digital Computers

Computers were classified into generations based on revolutions in the technology used in the development. Computer generation can be classified as shown in the table 1.2.

www.IPUnation.com

Table 1.2. Generation of Digital Computer

Generation:	First	Second	Third	Fourth	Fifth
Years:	1945-58	1959-64	1965-70	1971-till date	Soon
Sample computers	ENIAC, EDVAC, UNIVAC	IBM 1400	IBM 360	Microcomputer	Super micro-computer
Size of computer	A few rooms	Room	A few racks	Table top	"Tablet"
Main memory Device	Magnetic Drum	Magnetic Core	Magnetic Core	RAM Chip	—
Input Devices	Cards, Paper Tape	Cards	Key to Tape/Disk	Keyboard, mouse, optical character recognition	Speech and touch
Output Devices	Cards, print	Cards, print	Print, Video	Print, Audio/ video	Graphics, voice
Software	Machine language	Assembly language	High level language such as FORTRAN, COBOL	DBMS, 4GLs, SW Packages	Natural Language GP Package.

1.2.5. Layers in the Modern Computer

A modern computer is not a mere electronic machine. It is a system consisting of multiple layers. The innermost layer is the hardware unit that is surrounded by other layers of software as shown in Fig. 1.5. A programmer writes an application program in a high level language using decimal numbers and English statements. The compiler is a language translator which converts the high level language program into equivalent machine language program consisting of instructions and binary numbers (Fig. 1.6). A high level language program consists of statements whereas a machine language program has instructions. The operating system is a set of programs that provide a variety of functions with the objective of offering an efficient and friendly environment to the users and programmers. The following are important functions of operating system:

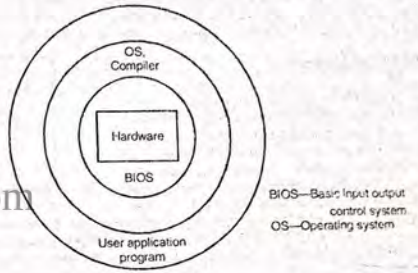


Fig. 1.5. Layers in a computer

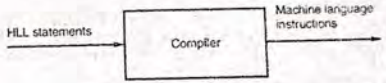


Fig. 1.6. Use of compiler

1. Handling computer user requests for various services;
2. Scheduling of programs;
3. Managing I/O operations;
4. Managing the hardware units.

The Basic Input-Output control System (BIOS) is collection of I/O drivers (programs for performing various I/O operations) for different peripheral devices in the computer. These programs can be called by other programs whenever an I/O operation has to be done.

1.3. COMPUTER ARCHITECTURE AND ORGANIZATION

Computer Architecture is concerned with the structure and behavior of the computer as seen by the user. It includes the information formats, the instruction sets, and techniques for addressing memory. The architectural design of a computer system is concerned with the specifications of the various functional modules such as processors and memories and structuring them together into a computer system.

Computer Organization is concerned with the way the hardware components operate and the way they are connected together to form a computer system. The various components of the computer system are assumed to be in place and the task is to investigate the organizational structure to verify whether the computer parts are operating as intended.

Computer Design is concerned with the hardware design of the computer. Once the computer specifications are formulated, it is the task of the designer to develop hardware for the system. Computer design is concerned with the determination of what hardware should be used and how the parts should be connected. This aspect of computer hardware is sometimes referred to as computer implementation.

1.4. ELEMENTS OF A CPU

The function of the CPU is executing the program stored in the memory. For doing this, the CPU fetches one instruction at a time, executes it and then takes up next instruction. This action is done repeatedly and is known as instruction cycle. As shown in Fig. 1.7, the instruction cycle consists of two phases: fetch phase and execute phase. In fetch phase, an instruction is fetched from memory. In execute phase, the instruction is analyzed and relevant operations are performed.

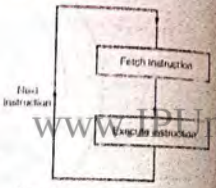


Fig. 1.7. Instruction cycle phases

1.4.1. CPU Registers

The CPU consists of following major registers (Fig. 1.8) :

1. Accumulator (AC)

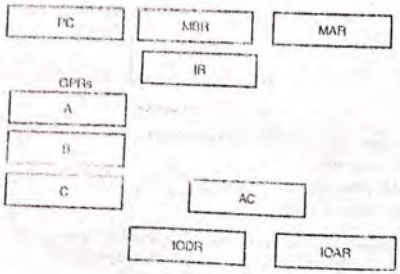


Fig. 1.8. CPU registers

2. Program Counter (PC)
3. Memory Address Register (MAR)
4. Memory Buffer Register (MBR)
5. Instruction Register (IR)
6. General Purpose Registers (GPR)
7. I/O Data Register (IODR)
8. I/O Address Register (IOAR)

AC, PC, MAR, MBR, IR, INPR, OUTR, GPR

AR, PC, DR, AC, INPR, OUTR, IR, TR

The adder performs arithmetic operations such as addition, subtraction etc. The accumulator register holds the result of previous operation in ALU. It is also used as an input register to the adder. The program counter (instruction address counter) contains the address of the memory location from where next instruction has to be fetched. As soon as one instruction fetch is complete, the contents of PC are incremented so as to point to the next instruction address. The instruction register stores the current instruction fetched from memory. The MAR contains the address of memory location during a memory read/write operation. The MBR contains the data read from memory (during read) or data to be written into memory (during write). THE GPRs are used for multiple purposes: storing operands, addresses, constants etc. In addition to GPRs, the CPU also has some working registers known as scratch pad memory. These are used to keep the intermediate results within an instruction cycle for complex instructions such as MULTIPLY, DIVIDE etc.

www.PDUnation.com

1.4.2. Clock

The clock unit generates and supplies a continuous sequence of clock pulses. The clock signal is used as a timing reference by the control unit. The clock signal is a periodic waveform since the waveform repeats itself. The rate at which the periodic waveform repeats is known as the frequency (f). It is specified as cycles per second (cps) or Hz. The clock frequency is an indication of the internal operating speed of the processor. The fixed interval at which the periodic signal repeats is called as its time period (T). The relationship between the frequency and the period is

$$f = \frac{1}{T}$$

The pulse width (tpw) indicates the duration of the (clock) pulse. Another related term is duty cycle. The duty cycle is the ratio (expressed in percentage) of pulse width to period, *i.e.*, duty cycle = $(tpw/T) \times 100\%$. Figure 1.9 illustrates the terms period, frequency, and duty cycle.

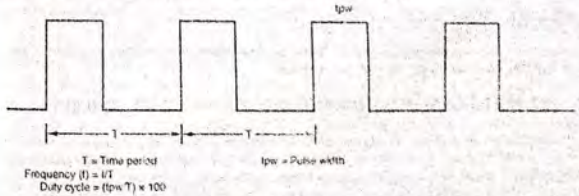


Fig. 1.9. Periodic waveforms

Example 1.1. *A clock signal has a frequency of 10 MHz with a duty cycle of 50%. Calculate its period and pulse width.*

Data : $f = 10\ MHz$; *duty cycle* $= 50\%$;

Solution. Time period $(T) = \frac{1}{f} = \frac{1}{(10 \times 10^6)} = 100\ \eta s$;

$$\text{Duty cycle} = \left(\frac{tpw}{T}\right) \times 100 = 50\%;$$

$$tpw = 0.5\ T = 50\ \eta s.$$

1.4.3. CPU Busses

A group of wires connecting two or more devices and providing a path to perform communication is called Bus. A bus that connects major computer components/modules (CPU, Memory, I/O) is called system bus. These system buses are separated into three functional groups:

1. Data bus.
2. Address bus.
3. Control bus.

(1) ***Data Bus***

The data bus consists of 8, 16, 32 or more parallel lines. The data bus lines are bidirectional. This means that CPU can read data on these lines from memory or from a port as well as data out on these lines to a memory location or to a port.

www.IPUnation.com

(2) ***Address Bus***

It is a unidirectional bus. The address bus consists of 16, 20, 24 or more parallel lines. The CPU sends out the address of the memory location or I/O port that is to be written or read from by using this address bus.

(3) ***Control Bus***

Control lines regulate the activity on the bus. The CPU sends signals on the control bus to enable the outputs of addressed memory device or port device.

Figure 1.10 shows a simple bus structure. There are three different buses: data bus, address bus and control bus. The data bus is used for multiple purposes:

1.5. COMPUTER ORGANIZATION AND STORED PROGRAM CONCEPT

Stored program concept is a process to store the instruction in computer memory to enable it to perform a variety of tasks in sequence or intermittently. 'Stored Program' was the breakthrough that enabled computer to perform complex tasks in fractions of second. Being able to do arithmetic and evaluate formulas fast would be useless if a person had to keep telling the computer what to do next. The

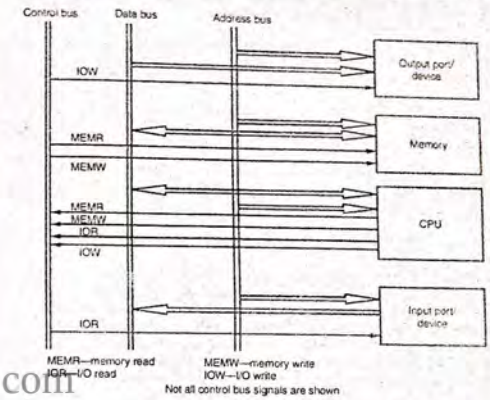


Fig. 1.10. Simple bus structure

trick was to encoding the list of instructions into numbers that could coexist in the computer's memory along with data, and having a 'processor' build into the machine to decode and execute the instructions. Now the computer could do both, calculations and control of the sequence at electronic speed. The idea was introduced by **John Von Neumann**, who proposed that a program can be electronically stored in binary-number format in a memory device so that instructions could be modified by the computer as determined by intermediate computational results.

A Von Neumann Architecture computer have five parts *i.e.*,

(*a*) An arithmetic-logic-unit

(*b*) A control unit

(*c*) A memory

(*d*) Some form of input/output and

(*e*) a bus that provides a data path between these parts.

The ALU and control unit have usually some temporary storage units known as registers. Each register can be considered as a fast memory with single location. Such registers temporarily store certain information such as instruction, data, address, etc. Storing in registers is advantageous since these can be read quickly compared to fetching them from external memory.

The ALU and control unit are together known as Central Processing Unit (CPU) or processor. The memory and CPU consist of electronic circuits and form the nucleus of the computer. The input and

output units are electromechanical units consistsing of both electronic circuits and mechanical assembly. The input and output units are known as peripheral devices.

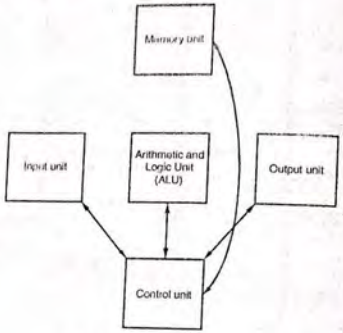


Fig. 1.11. Von Neumann Architecture

A von Neumann Architecture computer performs following sequence of steps:

1. Fetch the next instruction from memory at the address in the program counter.
2. ADD 1 to the program counter.
3. Decode the instruction using control unit. The control unit commands the rest of the computer to perform some operation.
4. Go back to step 1.

www.IPUnation.com

Limitation in Von Neumann Model

Very few computers have a pure von Neumann architecture. Most computers add another step to check for interrupts, electronic events that could occur at any time. An interrupt resembles the ring of a telephone, calling a person away from some lengthy task. Von Neumann computers spend a lot of time moving data to and from the memory, and this slows the computer. So, often we separate the bus into two or more busses, usually one for instruction and the other for data.

Difference between Von Neumann architecture and Harvard architecture?

Harvard architecture has separate data and instruction busses, allowing transfers to be performed simultaneously on both busses.

Von Neumann architecture has only one bus which is used for both data transfers and instruction fetches, and therefore data transfers and instruction fetches must be scheduled - they cannot be performed at the same time.

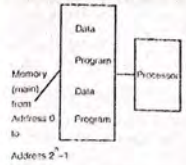


Fig. 1.12. Von Neumann Stored-program computer architecture

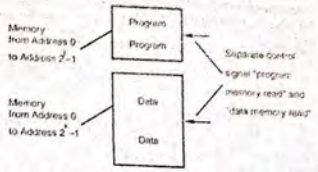


Fig. 1.13. Stored program computer architecture in harvard machine architecture.

1.6. PERFORMANCE

When we say one computer is faster than another, we compare their speeds and observes that the faster computer runs a program in less time than other computers. The computer center manager running a large server system may say a computer is faster when it completes more jobs in an hour. The computer user is always interested in reducing the time between the start and the completion of the program or event, *i.e.*, reducing the execution time. The execution time is also referred to as response time. Reduction in response time increases the throughput (the total amount of work done in a given time). The performance of the computer is directly related to throughput and hence it is reciprocal of execution time.

$$\text{Performance}_A = \frac{1}{\text{Execution time}_A}$$

This means that for two computers A and B if the performance of A is greater than the performance of B, we have

$$\text{Performance}_A > \text{Performance}_B$$

$$\frac{1}{\text{Execution time}_A} > \frac{1}{\text{Execution time}_B}$$

$$\text{Execution time}_B > \text{Execution time}_A$$

That is, the execution time on B is longer than that on A, if A is faster than B.

In discussing a computer design, we often want to relate the performance of two different computers quantitatively. We will use the phrase "A is n times faster than B" or equivalently "A is n times as fast as B" to mean.

$$\frac{\text{Performance}_A}{\text{Performance}_B} = n$$

If A is n times faster than B then the execution time on B is n times longer than it is on A:

$$\frac{\text{Performance}_A}{\text{Performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = n$$

Example 1.2. *If computer A runs a program ing 10 seconds and computer B runs the same program in 25 seconds, how much faster is A than B ?*

Solution. We know that A is n times faster than B is

$$\frac{\text{Performance}_A}{\text{Performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = n$$

Thus, the performance ratio is

$$\frac{25}{10} = 2.5$$

and A is therefore 2.5 times faster than B.

In the above example, we could also say that computer B is 2.5 times slower than computer A, since

$$\frac{\text{Performance}_A}{\text{Performance}_B} = 2.5$$

means that

$$\frac{\text{Performance}_A}{2.5} = \text{Performance}_B$$

For simplicity, we will normally use the terminology faster than when we try to compare computers quantitatively. Because performance and execution time are reciprocals, increasing performance requires decreasing execution time. To avoid the potential confusion between the terms increasing and decreasing, we usually say "improving performance" or "improve execution time" when we mean "increase performance" and "decrease execution time".

www.ITunites.com

The ideal performance of a computer system is achieved when we have a perfect match between the machine capability and the program behaviour. The machine capability can be enhanced with better hardware technology, innovative architectural features, and efficient resources management. However, program behaviour is difficult to predict since it heavily depends on application and run-time conditions. The program behaviour also depends on the algorithm design, data structures used, language efficiency, programmer skill and compiler technlology. Let us see the factors for projecting the performance of a computer.

1.6.1. Processor Clock

In today's digital computer, the CPU or simply the processor is driven by a closk with a constant cycle time called processor clock. The time period of processor clock is denoted by P. The period P of one clock cycle is an important parameter that affects processor performance. The clock rate is given by $R = 1/P$ which is measured in cycles per second (CPS). The electrical unit for this measurement of CPS is hertz (Hz). Today's personal computers and workstations have clock rates in the range of Megahertz (MHz) and Gigahertz (GHz). The computers having clock rate of 800 MHz have 800 million cycles per second.

1.6.2. CPU Time

CPU execution time, or simply CPU time is the time the CPU spends computing for particular task and does not include time spend waiting for I/O or running other programs. CPU time can be divided into

the CPU time spent in the program, called user CPU time and the CPU time spent in the operating system performing tasks on behalf of the program, called system CPU time. Differentiating between system and user CPU time is difficult to do accurately because it is often hard to assign responsibility for operating system activities to one user program rather than another and because of the functionality differences among operating systems. We use CPU performance to refer user CPU time.

1.6.3. Performance Metrics

Users and designers often examine performance using different metrics. If we could relate these different metrics, we could determine the effect of a design change on the performance as seen by the user. Since we are confining ourselves to CPU performance at this point, the bottom-line performance measure is CPU execution time. A simple formula relates the most basic metrics (clock cycles and clock cycle time) to CPU time.

CPU execution time for a program = CPU clock cycles for a program × Clock cycle time

Alternatively, because clock rate and clock cycle time are inverse of each other.

$$\text{CPU execution time for a program} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

This formula makes it clear that the hardware designer can improve performance by reducing either the length of the clock cycle or the number of clock cycles required for a program.

1.6.3.1. Hardware Software Interface

The previous equation do not include any reference to the number of instructions needed for the program. However, since the compiler clearly generated instructions to execute and the computer had to execute the instructions to run the program, the execution time must depend on the number of instructions in a program.

For the execution of program, processor has to execute number of machine language instructions. This number is denoted by N. The number N is the actual number of instructions executed by the processor and is not necessarily equal to the number of machine instructions in the machine language program. This is because some instructions may be executed more than once in the loop and others may not be executed at all. Each machine instruction takes one or more cycle time for execution. This time is required to perform various steps needed to execute machine instruction. The average number of basic steps required to execute one machine instruction is denoted by S, where each basic step is completed in one clock cycle. Thus, the program execution time is given by

$$T = \frac{N \times S}{R} \quad \ldots(i)$$

where N is the actual number of instructions executed by the processor for execution of a program, R is a clock rate measured in clocks per second and S is the average number of steps needed to execute one machine instruction. The above equation is known as **basic performance equation.**

When machine instruction execution time is measured in terms of cycles per instruction (CPI), the program execution time is given as

$$T = \frac{N \times CPI}{R} \quad \ldots(ii)$$

We know that each instruction execution involves cycle of events involving the instruction fetch, decode, operand(s) fetch, execution and store results. We need to access memory to perform instruction fetch, to perform operand(s) fetch or to store results.

$$\text{Speedup} = \frac{1}{\left[0.6 + \left(\frac{0.4}{50}\right)\right]} = 1.64.$$

This example illustrates the impact that the fraction of time an improvement is used has on overall performance. As $\text{Speed}_{\text{improved}}$ goes to infinity, overall speedup converges to $1/\text{Frac}_{\text{unimproved}}$, because the improvement can't do anything about the execution time of the fraction of the program that does not use the improvement.

SOLVED QUESTIONS WITH ANSWERS

Q. 1. What are the functional units of a computer?

Ans. The functional units of a computer are: Input, memory, arithmetic and logic, output and control units.

Q. 2. Define ALU. What are the various operations performed in ALU?

Ans. ALU is a part of computer that performs all arithmetic and logical operations. It is a component of central processing unit.

Arithmetic operations: Addition,Subtraction,multiplication,division,increment and decrement.

Logical operations: AND, OR, XOR, NOT, Compare, Shift, Rotate

Q. 3. Define registers?

Ans. When operands are brought in to the processor, they are stored in high-speed storage elements called registers.

Q. 4. What is a bus? What are the different buses?

Ans. A group of lines that serves as a connecting path for several devices is called a bus.

The different buses in a CPU are:

(*i*) Data bus

(*ii*) Address bus

(*iii*) Control bus.

Q. 5. Give the performance equation?

Ans.

$$T = (N \times S)/R$$

Where S = Average number of basic steps needed to execute one machine instruction.

R = Clock Rate

N = Actual number of machine language instructions

T = Processor time.

Q. 6. What are the registers generally contained in the processor?

Ans. **MAR:** Memory Address Register

MDR: Memory Data Register

IR: Instruction Register

R0-Rn: General purpose Registers

PC: Program Counter

Q. 7. What is elapsed time of computer system?

Ans. The total time to execute the total program is called elapsed time. It is affected by the speed of the processor, the disk and the printer

Q. 8. What is the use of buffer register?

Ans. The buffer register is used to avoid speed mismatch between the I/O device and the processor.

Q. 9. Why is the data bus in most microprocessors bidirectional while the address bus is unidirectional?

Ans. During execution the processor sends the address of the location to be accessed to memory.

The memory sends back the data to the processor; after execution the processor transfer the results to the memory.

Because of this reason the address bus are unidirectional and data bus is bidirectional.

Q. 10. Define machine instruction?

Ans. When a program is being executed in a processor, an instruction must encoded in a compact binary pattern. Such encoded instructions are called machine instruction.

Q. 11. Define *n*-bit ripple-carry-adder?

Ans. When a two *n*-bit numbers are added in a cascaded connection the carries must propagate, or ripple, through this cascade, the configuration is called an *n*-bit ripple-carry-adder.

REVIEW QUESTIONS

1. Explain the design of a basic computer.
2. How is processing of a program controlled in a computer
3. Explain the difference between an instruction and data in a program.
4. Name the smallest unit of memory where information is stored.
5. What is 'word length' of a computer ?
6. What is the difference between 'primary memory' and 'secondary memory' ?
7. Give details of main features related to different units in a processor.
8. How is the action of different units of a processor initiated ?
9. Discuss features of various generations of computers.
10. What is the most important difference between distinct generations of computers ?
11. Describe the architecture of the Von Neumann Machine and explain how the basic features of this architecture are incorporated in present day computers.
12. List the differences between the study of computer organization and computer architecture
13. When a computer launches a new model of computer, what changes in the hardware and software do you expect ?
14. What steps are involved for a software instruction to be executed in the computer hardware ?
15. How does the concept of layered model of computer architecture reduce the complications of study of the subject ?
16. Clarify the advantage of cost involved by studying the subject of computer architecture as a hierarchy of layers.

□□□

www.IPUnation.com

Fundamentals of Digital Electronics

2

Modern computers do not work with decimal numbers. Instead of, they process binary numbers groups of 0's and 1's as electronic devices are most reliable when designed for two states (Binary operation either on or off. People do not like working with binary numbers as the binary numbers are very long. Entering binary numbers into computer becomes tedious. Therefore octal and Hexadecimal numbers are widely used to compress long strings of binary numbers. Logic gate is a circuit with one or more input signals but only one output signal. It is used to construct a logical circuits which are the building blocks of a computer. Therefore it is necessary to study the basic operation of gates and logical circuits. This chapter discusses various number systems, gates and concepts needed to understand computer operation.

2.1. NUMBER SYSTEM

In general, a number system consists of an ordered set of symbols known as digits with rules defined for performing arithmetic operations like addition, subtraction, multiplication and division. A collection of these digits makes a number which in general has two parts – integer and fractional as shown below.

$$(N)_b = \underbrace{d_{n-1}, d_{n-2}, \ldots d_2, \ldots, d_1, \ldots, d_0}_{\text{Integer portion}}, \underbrace{d_{-1}, d_{-2}, \ldots, d_{-f}}_{\text{fractional portion}} \ldots d_{-m}$$

redix point

Table 2.1 gives the details of commonly used number systems.

Table 2.1.

Number system	*Base or radix (b)*	*Symbol used (di or d_f)*	*Weight assigned to position*		*Example*
Binary	2	0, 1	2^i	2^{-f}	1011.11
Octal	8	0, 1, 2, 3, 4	8^i	8^{-f}	3567.23
Decimal	10	0, 1, 2, 3, 4, 5	10^i	10^{-f}	3074.57
Hexadecimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8 A, B, C, D, E, F	16^i	16^{-f}	3FA9.58

2.1.1. Binary Number System

To understand the operation of a digital computer the knowledge of binary system is essential.

Binary number system deals with two digits 0 and 1, so the base or radix of the binary number system is 2. In short a binary digit is known as a bit. All computer systems use binary number system for their internal operation and manipulation.

Let the binary number is $(1011)_2$. This can be represented as: $1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

The weight of Ist digit 1 $= 1 \times 2^0$

The weight of IInd digit 1 $= 1 \times 2^1$

The weight of IIIrd digit 0 $= 0 \times 2^2$

The weight of IVth digit 1 $= 1 \times 2^3$

So the weight of *n*th bit of the number considering from right hand side is

$$= n\text{th bit} \times (2)^{n-1}$$

$$= n\text{th bit} \times (\text{Base})^{n-1}$$

The weight of each bit depends upon the relative position of that bit in the number.

2.1.2. Octal Number System

The base of the octal number system is 8. It uses eight digits 0, 1, 2, 3, 4, 5, 6 and 7. Octal number system is also positional number system, the weight of each digit depends upon the relative position of that digit in the number.

Let the number is $(534)_8$. This can be represented as : $5 \times 8^2 + 3 \times 8^1 + 4 \times 8^0$

The weight of the Ist digit 4 $= 4 \times 8^0$

The weight of the IInd digit 3 $= 3 \times 8^1$

The weight of the IIIrd digit 5 $= 5 \times 8^2$.

This can be generalised as weight of the nth digit of number from right hand side is

$$= n\text{th digit} \times (8)^{n-1}$$

The octal number system is the most popular number system in digital circuit. This system is also used in computer industry.

2.1.3. Decimal Number System

The most familiar number system is decimal number system. To represent any number this system uses 10 digits (0 to 9). Thus, decimal number system is said to have a Base or Radix of 10. The weight of each digit depends upon the relative position of that digit in the number.

Let the number is $(678)_{10}$. This can be represent as : 600 + 70 + 8

$$= 6 \times 10^2 + 7 \times 10^1 \times 8 \times 10^0$$

The weight of Ist digit 8 $= 8 \times 10^0$

The weight of IInd digit 7 $= 7 \times 10^1$

The weight of IIIrd digit 6 $= 6 \times 10^2$

The above expression can be generalised as the weight of nth digit of the number

$$= n\text{th digit} \times (10)^{n-1}$$

$$= n\text{th digit} \times (\text{Base})^{n-1}$$

digit is considered from right hand side.

2.1.4. Hexadecimal Number System

Hexadecimal number system use 16 digits to represent a number. Its radix (base) is 16. Its digits 0 to 9 are same as decimal number system. The alphabets *A, B, C, D, E, F* are used to represent 10, 11, 12, 13, 14 and 15 respectively.

Let the number $(A49)_{16}$. This can be represented as : $A \times 16^2 + 4 \times 16^1 + 9 \times 16^0$

The weight of the Ist digit $9 = 9 \times (16)^0$

The weight of the IInd digit $4 = 4 \times (16)^1$

The weight of the IIIrd digit $A = A \times (16)^2$

So the weight of *n*th digit of a number can be represented as

$$= n\text{th digit} \times (16)^{n-1}$$

$$= n\text{th digit} \times (\text{Base})^{n-1}$$

Again hexadecimal number system is positional number system, weight of each digit depends upon the relative position of the digit in the number.

2.1.5. Conversions

Each and every positional number system is having its own application. There is requirement for converting a given number system into another number system for the internal processing of a computer system.

2.1.5.1. Decimal to Binary Conversion

www.IPUnotes.com

A decimal number consists two parts: integer part and fraction part. So while converting a given decimal number into binary number first we will convert integer part into its equivalent binary number and then fraction part we will be converted into its binary equivalent. For converting the integer part, the number is divided by 2 progressively, until the quotient of zero is obtained. The binary equivalent is obtained by taking the remainder after each division in the reverse order. The binary equivalent of fraction part is obtained by multiplying the number continuously by 2, taking a carry in the integer position each time. The carries in the forward order provide required binary equivalent.

Example 2.1. *Convert the decimal number $(53.125)_{10}$ into an equivalent binary number.*

Solution.

Step 1. Integer part : 53

Fraction part : 0.125

Step 2. Integer part Conversion :

Division	*Remainder*
53 ÷ 2	1 (LSB)
26 ÷ 2	0
13 ÷ 2	1
6 ÷ 2	0
3 ÷ 2	1
1 ÷ 2	1 (MSB)

Reading the remainder from bottom to top

$$(53)_{10} = (110101)_2$$

Step 3. Fractional part conversion :

Multiplication	*Integer part*	*Fraction*
0.125 × 2 = 0.250	0 (MSB)	0.250
0.250 × 2 = 0.500	0	0.500
0.500 × 2 = 1.000	1 (LSB)	0.000 (stop)

Now fraction is 0.000

The binary equivalent of the fraction is written from top to bottom.

So $(0.125)_{10} = (0.001)_2$

On combining the solution of integer part and fractional part.

$$(53.125)_{10} = (110101.001)_2$$

2.1.5.2. Decimal to Octal Conversion

The decimal to octal conversion is similar to decimal to binary conversion. First we will convert integer part into its octal equivalent and then fractional part is converted into its octal equivalent. The integer part is divided by 8 repreatedly until a quotient of 0 is obtained. The octal equivalent is obtained by taking the remainder after each division in the reverse order. The octal equivalent of fraction part is obtained by multiplying the number continuously by 8, taking the whole number in account every time.

Example 2.2. *Convert the decimal number $(159.456)_{10}$ into an equivalent octal number.*

Solution.

Step 1. Integer part : 159

Fraction part : 0.456

Step 2. Integer part conversion :

Division	*Remainder*
159 ÷ 8	7 (LSB)
19 ÷ 8	3
2 ÷ 9	2 (MSB)
0	

So $(159)_{10} = (237)_8$

Step 3. Fractional part conversion :

Multiplication	*Integer part*	*Fraction*
0.456 × 8 = 3.648	3 (MSB)	0.648
0.648 × 8 = 5.184	5	0.184
0.184 × 8 = 1.472	1	0.472
0.472 × 8 = 3.776	3	0.776
0.776 × 8 = 6.208	6 (LSB)	0.208

The process is terminated when significant digits are obtained.

Thus, the octal equivalent of $(234.256)_{10}$ is $(227.2B3D6)_{16}$.

2.1.5.3. Decimal to Hexadecimal Conversion

The hexadecimal equivalent of a decimal number is obtained by dividing the integer part by 16 repeatedly until a quotient of 0 is obtained. The fraction part is multiplied by 16 and integer part is taken into account at every step.

Example 2.3. *Convert the decimal number $(234.14)_{10}$ into an equivalent hexadecimal number.*

Solution.

Step 1 Integer part : 234

Fraction part : 0.14

Step 2 Integer part conversion :

Division	*Remainder*
234 ÷ 16	10 ← *A* (LSB)
14 ÷ 16	14 ← *E* (MSB)
0	

On reading the remainder from bottom to top

$$(234)_{10} = (EA)_{16}$$

Step 3 Fractional part conversion :

Multiplication	*Integer part*	*Fraction*
0.14 × 16 = 2.24	2 (LSB)	0.24
0.24 × 16 = 3.84	3	0.84
0.84 × 16 = 13.44	13 = D	0.44
0.44 × 16 = 7.04	7 (MSB)	0.04

This multiplication can be performed upto some significant bits.

So, $(0.14)_{10} = (0.23D7)_{16}$

Combining the solution of integer and fractional part. The equivalent hexadecimal number is $(EA.23D7)_{16}$

$$(234.14)_{10} = (EA.23D7)_{16}$$

2.1.5.4. Binary to Decimal Conversion

The conversion of the binary number into decimal number is carried out by multiplying each binary bit by its positional weight and then adding all the product of binary bit and positional weight. The binary number has two parts *i.e.*, whole number and fraction.

Let decimal number is written as $b_{m-1}, b_{m-2}, b_{m-3}, ..., b_0, b_{-1}, b_{-2}, ..., b_{-n}$ then binary equivalent can be obtained from the following expression :

$$b_{m-1}(2)^{m-1} + b_{m-2}(2)^{m-2} + ... b_0(2)^0 + b_{-1}(2)^{-1} + b_{-2}(2)^{-2} + ... + b_{-n}(2)^{-n} \quad ...(2.1)$$

This equation represents the weights of the binary number.

Example 2.4. *Convert the binary number $(101010)_2$ into its decimal equivalent.*

Solution. By referring to the equation (2.1), this binary number can be converted into equivalent decimal by multiplying each bit by its positional weight.

So,
$$(101010)_2 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2 + 0 \times 2^0$$
$$= 32 + 8 + 2 + 1$$
$$= 43_{10}$$

2.1.5.5. Binary to Octal Conversion

For converting a given binary number into its equivalent octal number, starting from the least significant bit, each group of 3 bits is replaced by its decimal equivalent. If less than 3 bits remain in grouping then the zeros are placed to make a group of 3 bits.

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

Example 2.5. *Convert the following binary number into its octal equivalent.*

$$(100\ 1011\ 001\ .\ 10101)_2 = (?)_8$$

Solution. The given binary number has fraction part also. The integer part will be converted into octal equivalent by grouping of 3 bits from right to left and fraction part will be converted by grouping of 3 bits from left to right.

001	001	011	001	.	101	010
↓	↓	↓	↓		↓	↓
1	1	3	1	.	5	2

So the octal equivalent of the given number is $(1131.52)_8$.

2.1.5.6. Binary to Hexadecimal Conversion

To convert a given binary number into its equivalent hexadecimal number, we start from least significant bit, each group of 4 bits is replaced by its decimal equivalent. If less than 4 bits remain in grouping then the zeros are placed to make a group of 4 bits.

www.IPUnation.com

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

Example 2.6. *Convert $(0110110111101010.10110101)_2$ into hexadecimal equivalent.*

Solution.

0110	1101	1110	1010	.	1011	0101
↓	↓	↓	↓		↓	↓
6	D	E	A	.	B	2

So, $(0110\ 1101\ 1110\ 1010\ .\ 1011\ 0101)_2$

$= (6\,D\,E\,A\,.\,B\,5)_{16}$.

www.IPUnation.com

2.1.5.7. Octal Decimal Conversion

The conversion of the octal number into decimal number is carried out by multiplying each bit of octal number by its positional weight and then adding all the product of octal bits and positional weights.

Let decimal number is written as $b_{m-1}\, b_{m-2}\, b_{m-3} \ldots b_0\, b_{-1}\, b_{-2}\, b_{-3} \ldots b_{-n}$ then octal equivalent can be obtained from the following expression:

$$b_{m-1}(8)^{m-1} + b_{m-2}(8)^{m-2} + \ldots + b_0(8)^0 + b_{-1}(8)^{-1} + b_{-2}(8)^{-2} + \ldots + b_{-n}(8)^{-n}$$

This equation represents the weights of the octal number.

Example 2.7. *Convert $(117.33)_8$ into decimal.*

Solution.

$$(117.33)_8 = (1 + 8^2 + 1 \times 8^1 + 7 \times 8^0) + (3 \times 8^{-1} + 3 \times 8^{-2})$$
$$= (16 + 8 + 7) + (0.375 + 0.046)$$
$$= (320) + (0.421)$$
$$= (320.421)_{10}.$$

2.1.5.8. Octal to Binary Conversion

The conversion from octal to binary number can be performed by two methods. First the number can be converted into decimal and then decimal number is converted into equivalent binary number. In the second method each digit in the octal number can be replaced by its 3 bit binary equivalent.

Example 2.8. *Perform the following conversion and find (X)*

$$(4057.06)_8 = (X)_{10}$$

Solution. To findout the value of X, we have to perform conversion from octal number to decimal number.

$$(4057.06)_8 = (4 \times 8^3 + 0 \times 8^2 + 5 \times 8^1 + 7 \times 8^0) + (0 \times 8^{-1} + 6 \times 8^{-2})$$
$$= (2048 + 0 + 40 + 7) + (0 + 0.09375)$$
$$= (2095.09375)_{10}$$

So the value of X is 2095.09375.

2.1.5.9. Octal Hexadecimal Conversion

The conversion from octal number to hexadecimal number is performed in two steps:

Step 1. Convert the given octal number into its binary equivalent.

Step 2. Convert the binary equivalent into equivalent hexadecimal number.

Example 2.9. *Convert $(1736.57)_{16}$ to its equivalent hexadecimal number.*

Solution. *Step 1.*

$$(1736)_8 = (001)\ (111)\ (011)\ (110)$$
$$= (0011111\ 011\ 110)_2 \qquad \text{[for integer part]}$$

For fraction part

$$(0.57)_8 = (101)\ (111)$$
$$= (101111)_2$$

So $(1736.57)_8 = (001111\ 011\ 110.101111)_2$

Step 2. Now forming the groups of 4 binary bits. We get

0011	1101	1110	.	1011	1100
↓	↓	↓		↓	↓
3	D	E	.	B	C

So $(3\,D\,E\,.\,B\,C)_{16}$.

This is hexadecimal equivalent.

2.1.5.10. Hexadecimal to Decimal Conversion

The conversion of the hexadecimal number into decimal number is carried out by multiplying each digit by its positional weight and then adding all the product of digits and positional weight.

Let decimal number is represented in the following form :

$$b_{m-1}, b_{m-2}, \ldots, b_0, b_{-1}, b_{-2}, b_{-3}$$

Then hexadecimal equivalent can be obtained from the following expression :

$$b_{m-1}(16)^{m-1} + b_{m-2}(16)^{m-2} + \ldots + b_0(16)^0 + b_{-1}(16)^{-1} + b_{-2}(16)^{-2} + \ldots b_{-n}(2)^{-n} \quad \ldots(2.3)$$

This equation represents the weights of hexadecimal digits.

Example 2.10. *Convert $(8A3.D)_{16}$ into decimal equivalent.*

Solution. The given hexadecimal number is $(8A3.D)_{16}$.

8	A	3	.	D
↓	↓	↓		↓
8	10	3		13

These decimal equivalents are multiplied by positional weights.

$$= (8 \times 16^2 + 10 \times 16^1 + 3 \times 16^0) + (13 \times 16^{-1})$$
$$= (2048 + 160 + 3) + (0.8125)$$
$$= (2211.8125)_{10}.$$

2.1.5.11. Hexadecimal to Binary Conversion

A binary number has a base of 2 while a hexadecimal number has a base of $2^4 = 16$. The conversion from hexadecimal number into binary number can be obtained by replacing each digit of hexadecimal number into its 4 bit binary equivalent. It is very easy method.

Example 2.11. *Convert $(A2F9)_{16}$ into its binary equivalent.*

Solution. The given hexadecimal number is $(A2F9)_{16}$ each digit is replaced by its corresponding

www.IPUnation.com

A	2	F	9
↓	↓	↓	↓
1010	0010	1111	1001

$$(A2F9)_{16} = (1010\ 0010\ 1111\ 1001)_2.$$

2.1.5.12. Hexadecimal to Octal Conversion

This conversion is performed in two steps.

Step 1. Convert the given hexadecimal number into equivalent binary number.

Step 2. Then this binary number is grouped in 3 bits starting from right to left for integer part and from left to right for the fractional part. Then these 3 bit groups are converted into their decimal equivalent and we will get equivalent octal number.

Example 2.12. *Convert the real hexadecimal number 5B.3A to its equivalent octal number.*

Solution.

Step 1. (5B.3A) is converted into binary form

$$5B.3A = (0101)\ (1011)\ .\ (0011)\ (1010)$$
$$= (01011011\ .\ 00111010)_2$$

Step 2. Now forming the groups of 3 bianry bits to obtain equivalent octal number.

$$(01011011.00111010)_2 = (01)\ (011)\ (011)\ .\ (001)\ (110)\ (10)$$
$$= (001)\ (011)\ (011)\ .\ (001)\ (110)\ (100)$$
$$= (133.164)_8.$$

2.2. COMPLEMENTS

In digital computer complements are used to simply the operation of subtraction and for logical manipulation. For each bse system r two types of complements are defined.

1. r's complement
2. $(r-1)$'s complement.

When the value of base r is 2 then 2's complement and 1's complement are defined. When the value of r is 10 then 10's complement and 9's complement are defined.

2.2.1. r's Complement

It is radix complement. The r's complement of a given number N having n digits and base r is defined as

$$= r^n - N \qquad \text{for } N \neq 0$$
$$= 0 \qquad \text{for } N = 0$$

$$r\text{'s complement} = r^n - N$$
$$= [(r^{n-1}) - N] + 1$$
$$= (r-1)\text{'s complement} + 1$$

So adding 1 to $(r-1)$'s complement, we get r's complement of a number.

2.2.1.1. 2's Complement

It is defined for binary number system. The 2's complement of a given number N is defined as

$$2^n - N$$

where n = number of bits in number

2's complement can be findout using two method.

Method 1. (*i*) Findout the 1's complement of the given number by changing 1 into 0 and 0 into 1.

(*ii*) Add 1 to 1's complement to get 2's complement of the number.

Method 2. (*i*) Start from the LSB of the given number.

(*ii*) The first 1 occurs will remain unchanged.

(*iii*) After the first 1 change all significant bits 0 by 1 and 1 by 0.

2.2.1.2. 10's Complement

10's complement is defined for decimal number system. 10's complement is defined as

$$= 10^n - N$$

where N = Given number

n = number of digits

10's complement can be written as

$$= [10^n - 1] - N + 1$$
$$= 9\text{'s complement} + 1$$

So 10's complement of a given number can be obtained by adding 1 to 9's complement.

Example 2.13. *Find out 2's complement of $(100101.11)_2$.*

Solution. *Direct Method.* Scan the number starting from LSB. LSB is 1 so it will remain unchanged and all other bits will change from 0 to 1 and 1 to 0.

So	1	0	0	1	0	1	.	1	1
	↓	↓	↓	↓	↓	↓	.	↓	↓
	0	1	1	0	1	0	.	0	1

Decimal point is placed at the same place so 2's complement is 011010.01.

Example 2.14. *Find out 10's complement of 232140.*

Solution. Here

$$N = 232140$$
$$n = 6$$

So 10's complement is

$$= 10^6 - 232140$$
$$= 1000000 - 232140$$
$$= 767860$$

2.2.2. (r – 1)'s Complement

www.IPUnaCom.com

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

2.2.2.1. 1's Complement

It is defined for binary number system. The 1's complement of any given number N is $(2^n - 1) - N$. 2^n represents a binary number whose MSB is 1 and all other bits are zero. For example if $N = 1001$ then

$$2^n = 2^4(16)_{10} = (10000)_2$$

and

$$2^n - 1 = 10000 - 1 = (1111)_2$$

Then 1's complement of $(1001)_2$ is

$$= [2^n - 1] - N = 1111 - 1001 = 0110.$$

$(2^n - 1)$ has n 1's, so 1's complement of a given number can be obtained when each bit of N is subtracted from 1. Bits are only two either 0 or 1.

So

$$1 - 0 = 1$$
$$1 - 1 = 0$$

Implies 0 is got converted into 1 and 1 is converted into 0.

Therefore, the 1's complement of a given number can be obtained by replacing each bit of the number *i.e.*, '1' into '0' and '0' into 1. It is direct method bny which easily we can get 1's complement of a given binary number.

2.2.2.2. 9's Complement

9's complement of a given number is defined as $[10^n - 1] - N$

where n = Number of digits in number

N = Given decimal number

10^n represents a number that consists a single 1 followed by n 0's. $[10^n - 1]$ will represent a number having n 9's. For example,

$$N = (1234)_{10}$$

then

$$10^n - 1 = 10^4 - 1 = 10000 - 1 = 9999$$

9's complement of $(1234)_{10}$ is

$$= [10^n - 1] - N$$
$$= 9999 - 1234$$
$$= 8765$$

So 9's complement of a given number N can be obtained by subtracting each digit of given number by 0. It is very easy and direct method.

Example 2.15. *Find out 1's complement of $(10\ 101.1011)_2$.*

Solution.

	1	0	1	0	1	.	1	0	1	1
	↓	↓	↓	↓	↓	.	↓	↓	↓	↓
	0	1	0	1	0	.	0	1	0	0

Example 2.16. *Find out the 9's complement of $(413052)_{10}$.*

Solution. 9's complement is

$$= [10^6 - 1] - 413052$$
$$= 999999 - 413052$$
$$= 586947.$$

2.2.3. Subtraction Using 2's Complement

The subtraction of two binary number can be obtained using 2's complement. 2's complement of number can be directly obtained by adding 1 to 1's complement.

Let we have to perform $(A - B)$ then using 2's complement method we have to follow the following steps :

(*i*) Find out the 2's complement of B.

(*ii*) Add $A + B$.

(*iii*) If carry is generated then ignore it. If there is no carry then take 2's complement of the result and change the sign of result.

Example 2.17. *Subtract $(1010)_2$ from $(1111)_2$ using 2's complement method.*

Solution.

$$B = (1010)_2$$
$$A = (1111)_2$$

2's complement of B = 0101 + 1 = 0110

Addition of A and B = 1111 + 0110

= 10101

↑

carry generation

The addition operation is generating carry.

So after ignoring carry bit we will get answer is $(0101)_2$.

Example 2.18. *Subtract 5 from 3 using 2's complement method.*

Solution.

$$(A) = (3)_{10} = (0011)_2$$
$$(B) = (5)_{10} = (0101)_2$$

2's complement of 5 = 1010 + 1
= 1011

Adding A and B

$$\begin{array}{r} 0011 \\ 1011 \\ \hline 1110 \end{array}$$

There is no carry so we take 2's complement of the result and will change the sign of the result.

2's complement of 1110 is

$$\begin{array}{r} 0001 \\ +1 \\ \hline 0010 \\ \hline \end{array}$$

$(0010)_2 = (2)_{10} = (-2)_{10}$ [after changing the sign]

2.3. FIXED POINT NUMBERS

A number having a fixed length is called as fixed point number. In selecting a number representation to be used in a computer, the following factors should be taken into account.

- The numbers types to be represented; for example, integers or real numbers.
- The range of values likely to be encountered.
- The precision of the numbers, which refers to the maximum accuracy of the representation.
- The cost of the hardware require to store and process the numbers.

We have two formats:

- fixed-point and
- floating point.

Fixed point formats allow a limited range of values and also require less hardware for processing. On the other hand, floating-point numbers allow large range of values but require costly processing hardware or lengthy software implementations.

In binary number system, a number can be represented as an integer or a fraction. Depending on the design, the hardware can interpret numbers as an integer or fraction. The radix point is never explicitly specified. It is easier in the design and the hardware interprets it accordingly. In integer numbers, the radix point is fixed and assumed to be the right of the rightmost digit. As radix point is fixed, the number system is referred to as fixed point number system.

Fixed Point Representation

The fixed point/integer numbers are represented in two forms:

- signed integer and
- unsigned integer.

Unsigned integer numbers represent positive numbers. To represent negative numbers various techniques are used because computers do not have any provision to represent negative sign.

2.3.1. Signed Binary Numbers

A binary number can have only two symbols 0 and 1, these symbols are used to indicate positive binary number and negative binary number. In the n bits of signed binary number, one bit is reserved to indicate positive or negative value and the remaining bits are magnitude. The sign bit is placed as the most significant bit. The MSB, 0 indicates that the number is positive and MSB, 1 indicates negative binary number. The signed binary number can be represented in one of the three ways.

1. Sign-magnitude representation.
2. 1's complement representation.
3. 2's complement representation.

Sign-magnitude Representation

The use of unsigned integer is insufficient in many cases when we need to represent negative as well as positive numbers. The simplest form of representation that employs a sign bit is the sign magnitude representation. In an n bit word, the right most $n-1$ bit holds the magnitude of the given number and nth bit is assigned for sign bit. If the sign bit (nth bit) is

0 ⇒	The given number is positive
1 ⇒	The given number is negative

www.IPUnation.com

For example, +18 and −18 are represented as 8 bit signed number as follows.

18 equivalent binary number = 10010

In the given 8 bit signed number, one bit is assigned to be sign bit (MSB). The remaining 7 bits represent the magnitude, but the binary equivalent of 18 is 10010 (5 bits). Two 0's are placed left most to make a 7 bit group (0010010), now we can represent the positive and negative number.

	Sign bit	Magnitude bits						
+18 ⇒	0	0	0	1	0	0	1	0
−18 ⇒	1	0	0	1	0	0	1	0

There are several drawbacks in sign-magnitude representation. One is that addition and subtraction require consideration of both the signs of the numbers and their relative magnitude in order to carry out the required operation. Another drawback is that there are two representations for 0.

+0 = 000000000

−0 = 100000000

This is inconvenient, because it is slightly more difficult to test for 0. The signed magnitude is not normally used because the circuit implementation is more complex than the other systems.

1's Complement Representation

The 1's complement of a binary number is obtained by changing each 0 to 1 and each 1 to 0. For 1's complement representation of signed number we have to find the 1's complement including sign bit. For example, let us see how 20 is represented in 8 bit 1's complement form. The magnitude of 20 is 10100 (5 bits). The 8 bit signed number consists of one bit (MSB) for sign bit and the remaining 7 bits represents magnitude of the given number. Two 0's are placed to make a 7 bit group, and the 1's complement representation of +20 and –20 is given as follows.

$+20_{10} \Rightarrow$

0	0	0	1	0	1	0	0

Sign bit — True binary (magnitude)

1's complement of $20 = -20_{10} \Rightarrow$

1	1	1	0	1	0	1	1

1's complement representation

2's Complement Represenation

If 1 is added to 1's complement of a binary number, the resulting number is known as the 2's complement of the binary number.

For example, $(-20)_{10}$ is represented as 8 bit 2's complement by adding 1 in the LSB to 1's complement. Whenever a signed number has a 1 in the sign bit and all 0's for the magnitude bits, its decimal equivalent is -2^N, where N is the number of bits in the magnitude.

$+20_{10} \Rightarrow$

0	0	0	1	0	1	0	0

1's complement of $+20_{10} \Rightarrow$

1	1	1	0	1	0	1	1

+ 1

2's complement of $20 = -20_{10} \Rightarrow$

1	1	1	0	1	1	0	0

Note: Complementation of positive number is negative number. for example complement of **+8** is the –8.

Example 2.19. *Represent each of the following signed decimal numbers as a signed bginary number in 8 bit sign magnitude representation and 1's complement representation.*

(a) +13. *(b)* –13 *(c)* +48 *(d)* –75

Solution.

(a) +13

Magnitude of 13 = 1101

$+13_{10} \Rightarrow$

0	0	0	0	1	1	0	1

Sign bit

(b) –13

$-13_{10} \Rightarrow$

1	0	0	0	1	1	0	1

Sign bit

(c) +48

$+48_{10} \Rightarrow$

0	0	1	1	0	0	0	0

Sign bit

(d) –75

$-75_{10} \Rightarrow$

1	1	0	0	1	0	1	1

Sign bit

www.JPLnotes.com

1's Complement Representation: In the 1's complement, positive number representation is the same as for signed magnitude representation but 1's complement representation is diffewrent from signed magnitude reprsentation for negative numbers.

(a) +13

$+13_{10} \Rightarrow$

0	0	0	0	1	1	0	1

(b) –13

Magnitude of 13 = 1101

$+13_{10} \Rightarrow$

0	0	0	0	1	1	0	1

$-13_{10} \Rightarrow$

1	1	1	1	0	0	1	0

1's complement

(c) +48

Magnitude of 48 = 11000

$+48_{10} \Rightarrow$

0	0	1	1	0	0	0	0

$-48_{10} \Rightarrow$

1	1	0	0	1	1	1	1

(d) –75

Magnitude of 75 = 1001011

-75_{10} =	0	1	0	0	1	0	1	1
-75_{10} =	1	0	1	1	0	1	0	0

1's complement

Table 2.2 shows representation of 4-bit integer numbers in sign-magnitude, 1's complement and 2's complement form.

Table 2.2

Decimal	Representations		
	Sign-magnitude	*1's complement*	*2's complement*
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
–0	1000	1111	—
–1	1001	1110	1111
–2	1010	1101	1110
–3	1011	1100	1101
–4	1100	1011	1100
–5	1101	1010	1011
–6	1110	1001	1010
–7	1111	1000	1001

www.IPSnotes.com

Advantage of 2's Complement Representation

- From Table 2.2, we can note that there are distinct +0 and –0 representation in both the sign magnitude and 1's complement representations, but the 2's complement representation has only one representation.
- For 4-bit numbers, the value –8 is representable only in the 2's complement representation and not in other representations.
- It is more efficient for logic circuit implementation, and more often used in computers for addition and subtraction operations.

IMPORTANT NOTES

- **Fixed point numbers:** Radix point is a point which separates the integer and fraction part of a number. If the radix point is fixed then the number is called fixed point number.
- Signed number can be represented as signed number, 1's complement representation and 2's complement representation.
- 1's complement representation has two representation for +0 and –0, this is drawback of this representation.
- 2's complement representation is best representation of signed number.

2.4. FLOATING POINT NUMBERS

The floating point numbers contains the binary point variable in its position and hence these numbers are called floating point numbers. The range of numbers that can be represented by a fixed-point number is insufficient. This *fixed point has limitations, very large numbers cannot be represented, nor can very small fractions.* Furthermore, the fractional part of the quotient in division of two large numbers could be lost.

For decimal numbers, one gets around this limitation by using scientific notation. Thus, 976000 can be represented as 9.76×10^5 and 0.0000976 can be represented as 9.76×10^{-5}. To represent binary numbers, it is necessary to consider binary point. To accommodate very large integers and very small fractions, a computer must be able to represent numbers and operate on them in such a way that the position of the binary points variable and is automatically adjusted as computation proceeds. In this way *the binary point is said to float, and the numbers are called floating point numbers.*

The floating point representation has the following three fields:

1. Sign
2. Mantissa
3. Exponent.

The general structure of floating point number is

$$\boxed{\pm M \times B^{\pm E}}$$

Where M is Significant (Mantissa) digits

E is Exponent

B is scaling factor, which is

- 2 for binary number
- 10 for decimal number.

Let us consider the number 1110111.100110 to be represented in the floating point format. To represent the number in floating point format, first binary point is shifted to left of the sign bit and the number is multiplied by correct scaling factor to get the same value.

$$1110001.10010 \longrightarrow \underbrace{1.110111100110}_{\text{Significant or mantissa}} \times \underbrace{2^{6}}_{\text{Scaling factor}} \longrightarrow \underbrace{1.110111100110 \times 2^{110}}_{\text{Normalised form}}$$

Exponent

It is important to note that the base in the scaling factor is fixed as 2 and therefore, it does not have to appear explicitly in the machine representation of a floating point number. The string of the significant digits is commonly known as mantissa.

In the above representation, we can say that

$$\text{Sign} = 0$$

$$\text{Mantissa} = 110111100110$$

$$\text{Exponent} = 110\ (6)$$

In floating point number, bias value is added to the true exponent. Due to this the magnitude of numbers can be compared by doing arithmetic on the exponent part.

The range of floating point is

$$\boxed{-1 \times 8^{8} \times 1 - 2^{-(8-1)}}$$

If x = 32 bit, then

$$-1 \times 8^{8} \times 1 - 4.65661 \times 2873 \times 10^{-10}$$

$$-1 \times 8^{8} \times 0.0000$$

IMPORTANT NOTES

- **Mantissa:** The floating point representation of a number has two parts. The first part represents a signed, fixed-point number called the mantissa.
- **Exponent:** The second part of floating point designates the position of the decimal (or binary) point is called exponent.
- **Floating point number:** A number contains the binary point variable in its position and hence these numbers are called floating point numbers.
- **Normalization:** A floating point number is said to be normalized if the most significant digit of the mantissa is non-zero. For example, the decimal number 350 is normalized but 00035 is not normalized.

www.IPUnation.com

2.4.1. IEEE Standards of Floating Point Representation

The standards for representing floating point numbers is 32 bits and 64 bits have been developed by the Institute of Electrical and Electronics Engineers (IEEE), refers to as IEEE 754 standards.

Based on the total number of bits, we can say the floating point numbers are **single precision** (32 bit) and **double precision** (64 bit).

Single Precision

The 32 bit IEEE standard floating point representation is shown in Fig. 2.1.

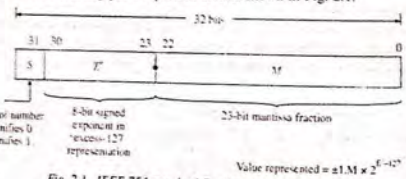


Fig. 2.1. IEEE 754 standard floating point numbers

- The sign of a number is given in the first bit followed by a representation for the exponent (to the base 2) of the scaling factor.
- Instead of the signed exponent E, the value actually stored in the exponent field is $E' = E + \text{bias}$.
- In the 32 bit floating point system (single precision) bias is 127. Hence $E' = E + 127$.
- This representation of exponent is also called the excess-127 format.
- The end values of E' namely 0 and 255 are used to indicate the floating point values of exact zero and infinity respectively in single precision.
- Thus E' is in the range $0 \le E' \le 255$.
- The actual exponent E is in the range $-126 \le E\ 127$. The last 23 bits represent the mantissa.

Note: The values 0 and 255 are used to indicate the floating point values of exact 0 and infinite respectively.

Double Precision

The 64-bit standard representation shown in Fig. 2.2 is called a double precision representation because it occupies two 32-bit words.

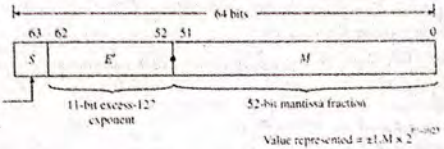


Fig. 2.2. Double precision floating point representation (IEEE 754).

- In the double precision format value actually stored in the exponent field is given as $E' = E + 1023$.
- Here bias value is 1023 and hence it is also called excess-1023.
- The end values of E', namely 0 and 2047 are used to indicate the floating point exact values of zero and infinity respectively.
- Thus the range of E' for normal values in double precision is $0 < E' < 2047$. This means that for 64-bit representation the actual exponent E is the range

$$-1022 \le E \le 1023$$

Paramater	Format	
	Single	Double
Word width (bits)	32	64
Exponent width (bits)	8	11
Exponent bias	127	1023
Maximum Exponent	127	1023
Minimum Exponent	–126	–1022
Number of Exponents	254	2046
Number of Fractions	2^{23}	2^{52}
Number of Values	1.98×2^{31}	1.99×2^{63}

The IEEE floating-point defined formats are specified for the results of overflow, underflow and other exceptional conditions. The IEEE standard's exception formats are intended to set flags in the processor which subsequent instructions can use for error control.

- If the result of a floating-point operation is not a valid floating-point number, then a special code is referred to as Not a Number (NaN). Examples of operations that result in NaNs are dividing zero by zero and taking the square root of a negative number.
- NaN formats are identified in the standard by $M \neq 0$ and $E = 255$ (32-bit format) or $E = 2047$ (64-bit format).
- When overflow occurs which means that a number has been produced whose magnitude is too large to represent this result is referred to an infinity and is identified by $M = 0$ and $E = 255$ (32-bit format) or $E = 2047$ (64-bit format).

In summary, the number N represented by a 32-bit IEEE-standard floating-point number has the following set of interpretations.

If $E = 255$ and $M \neq 0$, then $N = \text{NaN}$

If $E = 255$ and $M = 0$, then $N = (-1)^s \infty$

If $0 < E < 255$ then $N = (-1)^s\, 2^{E-127}\, (1.M)$

If $E = 0$ and $M \neq 0$, then $N = (-1)^s\, 2^{E-126}\, (0.M)$

If $E = 0$ and $M = 0$, then $N = (-1)^s\, 0$.

Example 2.20. *Represent the floating point in the IEEE standard format for the given number $1.00010100 \times 2^{-10}$.*

Solution.

Given number is $1.00010100 \times 2^{-10}$

Sign bit(s) $\Rightarrow$ 0 (1 bit)

Mantissa $(M) \Rightarrow$ 00010100 ... 0 (23 bits)

Actual Exponent $E = -10$

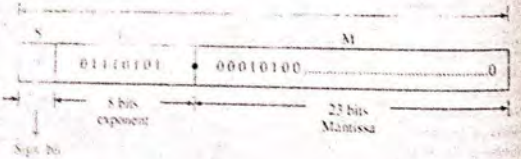
Modified Exponent $E' = E + 127 = -10 + 127$

$= +117$

Equivalent binary representation

$117 \Rightarrow 1110101$

8 bits are allotted to represent the exponent values, but in this example we have only 7 bits. Padding zero to the above 7 bits we to make a group of 8 bits.



www.IPUnation.com

Example 2.21. *Represent the floating point in the IEEE standard format for the given number $1.001010 \ldots 0 \times 2^{-87}$.*

Solution.

Given number's $1.001010 \ldots 0 \times 2^{-87}$

Sign bit $S = 0$ (The given number is positive number)

Mantissa $M = 0010100 \ldots 0$ (23 bits)

Modified Exponent $E' = 127 + E$

Where E is actual given exponent

$E = -87$

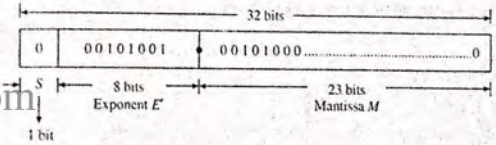
$E' = -87 + 127$

$E' = 40$

Equivalent binary representation $E' = (40)_{10}$

$= 101000$

Padding two zeros to make 8 bit group = 00101000



Example 2.22. *Represent 1460.125_{10} in single precision and double precision formats.*

Solution.

Step 1. Convert the decimal number in binary format

For integer

2	1460	
2	730 – 0	
2	365 – 0	
2	182 – 1	
2	91 – 0	
2	45 – 1	$(1460)_{10} = (10110110100)_2$
2	22 – 1	
2	11 – 0	
2	5 – 1	
2	2 – 1	
2	1 – 0	

For fraction

$$0.125 \times 2 = 0.250 \Rightarrow 0$$
$$0.250 \times 2 = 0.500 \Rightarrow 0$$
$$0.500 \times 2 = 1.000 \Rightarrow 1$$
$$(0.125)_{10} = (0.001)_2$$
$$(1460.125)_{10} = (10110110100.001)_2$$

Step 2. Normalise the number.

$$10110110100.001 = 1.0110110100001 \times 2^{10}$$

Now we represent the above floating point in single precision and double precision.

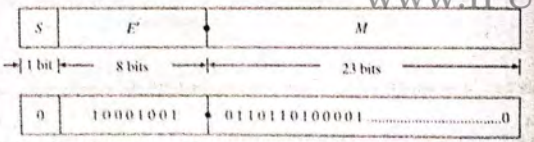
Single precision: From the above floating point

$$S = 0$$
$$E = 10$$
$$M = 0110110100001$$

The modified exponent (an bias for single precision)

$$E' = 127 + E = 127 + 10 = 137_{10}$$
$$= 10001001_2$$

Number in single precision format

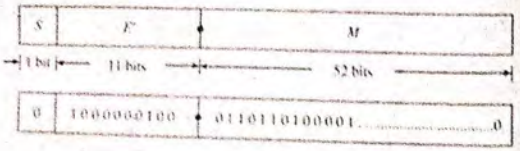


Double Precision: For a given number.

$S = 0$, $E = 10$ and $M = 0110110100001$

Bias (or) modified exponent for double precision

$$E' = E + 1023 = 10 + 1023$$
$$= (1033)_{10} = (10000001001)_2$$



www.IPUnation.com

Example 2.23. *Represent -307.1875_{10} in single precision and double precision formats.*

Solution.

Step 1. Convert the decimal number in binary format.

Integer format

2	307	
2	153 – 1	
2	76 – 1	
2	38 – 0	
2	19 – 0	
2	9 – 1	$(307)_{10} = (100110011)_2$
2	4 – 1	
2	2 – 0	
2	1 – 0	

Fractional format

$$0.1875 \times 2 = 0.370 \Rightarrow 0$$
$$0.3750 \times 2 = 0.750 \Rightarrow 0$$
$$0.750 \times 2 = 1.5 \Rightarrow 1$$
$$0.5 \times 2 = 1.0 \Rightarrow 1$$
$$(0.1875)_{10} = (0.0011)_2$$
$$(307.1875)_{10} = (100110011.0011)_2$$
$$-307.1875_{10} = -100110011.0011_2$$

Step 2. Normalise the number

$$-100110011.0011 = -1.001100110011 \times 2^8$$

Now we will see the represntation of the numbers in single precision and double precision formats.

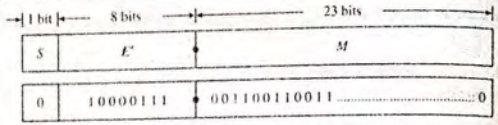
Single precision: For a given number

$$S = 1 \quad \text{(given number is negative number)}$$
$$E = 8$$
$$M = 001100110011$$

Bias of single precision format is = 127

$$E' = 127 + E$$
$$= 127 + 8$$
$$= (135)_{10}$$
$$= (10000\ 0111)_2$$

Number in single precision representation



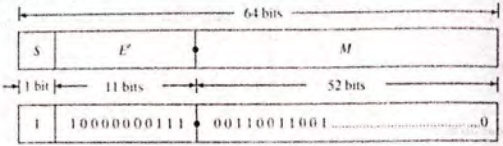
Double precision: For a given number

$S = 1, E = 8, M = 001100110011$

Bias for double precision format is = 1023

$$E' = 1023 + E = 1023 + 8$$
$$= 1031_{10} = 10000000111$$

Number in double precision



2.5. LOGIC GATES

Logic gates are the basic building block of the digital system. The term logic means to make decision based on the combination of inputs. So, a logic gate is an electronic device which makes the logical decision on the basis of given inputs. In case of a logic gate, there is only one input level which depends upon the combination of input levels. The value of inputs and outputs on logic gates can be 1 (High) and 0 (Low).

A table which shows all the possible combinations of input variables and corresponding output levels is called as truth table. If there are n number of inputs in a logic gate, than the possible input combinations will be equal to 2^n. If we use two input variables, at the input of a logic gate, then there are $2^2 = 4$ possible input combinations. The inputs can be represented by Boolean variables *A*, *B*, *C* etc. and the outputs can be boolean variable 1. Each logic gate's output can be represented by boolean expressions. There are seven logic gates which are discussed in the later section.

1. NOT Gate
2. OR Gate
3. AND Gate
4. NAND Gate
5. NOR Gate
6. XOR Gate
7. XNOR Gate

NOT Gate

It is called NOT gate because its output becomes opposite of the input. It is also called an inverter because it inverts the input signal. The inverter changes the one logic level to the opposite level. It changes an input 1 to output 0 and input 0 to output 1. The symbol and truth table for NOT gate is given below.

Fig. 2.5. Not Gate Symbol

www.IPLnation.com

Truth Table

A	Y
0	1
1	0

If the input at the NOT gate is *A*, then the logical expression for NOT gate is given by

$$Y = \overline{A}$$

OR Gate

The OR gate usually have two or more inputs and one output. The output of an OR gate is high (1) when any of the input is high (1). The output of a OR gate is low (0) when all the inputs are low (0). The logic symbol for OR gate is given below :

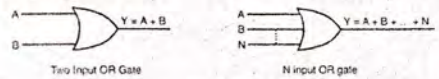


Fig. 2.6.

If the applied inputs are *A* and *B*, then logical expression for OR gate is given by $Y = A + B$ (Logical expression for OR gate).

Truth Table

A	B	Y = A + B
0	0	0
0	1	1
1	0	1
1	1	1

AND Gate

The AND gate also have two or more inputs and one output. The output of an AND gate is equal to the logical multiplication of inputs. The logic symbol for AND gate is given below :

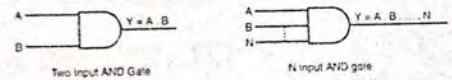


Fig. 2.7.

Truth table : The truth table for two input AND gate is given below :

A	B	Y = A . B
0	0	0
0	1	0
1	0	0
1	1	1

Microoperations and Design of an ALU

3

A digital computer stores data in terms of digits (numbers) and proceeds in discrete steps from one state to the next. The states of a digital computer typically involve binary digits which may take the form of the presence or absence of magnetic markers in a storage medium, on-off switches or relays. In digital computers, even letters, words and whole texts are represented digitally.

The internal hardware of a digital computer is best described by:

1. The set of registers it contains and their function.
2. The sequence of microoperations performed on the binary information stored in the registers.
3. The control that initiates the sequence of microoperations.

In this chapter we will learn how to perform different operations on data stored in the registers, and how the operations are to be initiated.

3.1. REGISTER TRANSFER AND REPRESENTATION

The registers in a digital computer are designated by capital letters (and if required are also followed by numerals) to denote the function of the register. For example, the register that contains the address of the next instruction to be executed is program counter and is designated by PC. Other registers are IR (for instruction register), MAR (memory address register), $R1$ (for processor register) etc.

The most common way of representing a register is with the help of a rectangular box with the name of the register written inside the box as in Fig. 3.1(a) we can also distinguish the individual bits of a register as in Fig. 3.1(b) with each bit assigned a letter with a subscript number. The numbering of bits from right to left can be marked on the top of the box as in Fig. 3.1(c). Also, a 16-bit register is partitioned into two parts as is Fig. 3.1(d), where bits 0 through 7 are symbol L (for low bytes) and bits 8 through 15 are assigned the symbol H (for high byte).

Register transfer is defined as the transfer of information between registers with the help of common bus. The symbolic notation used to describe the microoperation transfer among registers is called a register transfer language.

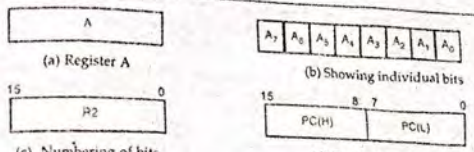


Fig. 3.1. Block Diagram of Register.

The symbolic notation used to describe the microoperation transfer among registers is called a *register transfer language*. This is a way of expressing the different microoperations sequence in the symbolic form among registers of a computer.

For example, the statement,

$$R2 \longleftarrow R1$$

simply denotes that the content of register $R1$ is to be transferred into register $R2$. After this transfer operation takes place the previous content of $R2$ will be replaced by the content of $R1$ but the content of register $R1$ does not change after the transfer. $R1$ is the source register and $R2$ is the destination register. If we want that the transfer can take place only when certain control condition satisfies, this can be shown by means of an if then statement as

$$\text{if } (P = 1) \text{ then } (R2 \leftarrow R1)$$

where P is a control signal, generated in the control section. Also, some time we use control function instead of the control variables from the register transfer operation. A control function is a Boolean variable that is equal to 1 or 0. Thus, the above statement is equivalent as

$$P : R2 \longleftarrow R1$$

where colon is used to terminate the control condition.

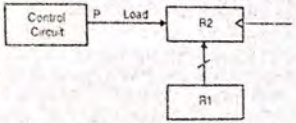


Fig. 3.2. Hardware Implementation of $P : R2 \longleftarrow R1$

Every statement written in a register transfer notation implies a hardware implementation. The block diagram that depicts the transfer from $R1$ to $R2$ is shown in Fig. 3.2.

The basic symbols used for the register transfer notation is shown in Table 3.1.

Table 3.1. Basic Symbols for Register-Transfer Logic.

Symbol	*Description*	*Examples*
Letters (and numerals)	Denotes a register	$R1$, $R2$, PC, IR
Parentheses ( )	Denotes a part of register	PC (0 – 7), PC (L)
Arrow ←	Denotes a transfer of information	$R2 \leftarrow R1$
Comma,	Separates two Microoperation	$R2 \leftarrow R1$, $R3 \leftarrow R1$
Subscript	Denotes a bit of a register	A_2, B_8
Colon :	Terminates a control function	$x' T_0'$
Square brackets [ ]	Specifies an address for memory transfer	MBR ← M[MAR]

www.IRUnotes.com

3.2. MICROOPERATIONS

The operations executed on data stored in registers are called microoperations, for example shift, count, clear, and load. The microoperations are classified into the following four categories:

1. **Data transfer microoperation.**
   (a) *Register transfer microoperations.* These micro operations transfer the data from one register to another register.
   (b) *Memory transfer microoperations.* These microoperations transfer data from register to memory or memory to register.

2. **Arithmetic microoperations.** These microoperations perform arithmetic operations on data stored in registers. Addition, subtraction, multiplication, division all are arithmetic microoperations.

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

4. **Shift microoperations.** These microoperations shift the data in left or right direction stored registers.

3.3. REGISTER/DATA TRANSFER MICROOPERATIONS

Register transfer means that the contents of one register is transferred to another register. The information transfer from one register to another is designated in symbolic form by using the operation. The statement of a simple register transfer is given by

$$B \leftarrow A$$

The above statement denotes the transfer between two registers. The contents of register A are transferred into register B. It is a replacement the contents of B by the contents of A, but the contents of register A do not change A further after this transfer.

www.IPUnation.com

The above statements are very simple and it specifies a register transfer of the contents of source register (A is source register for the given statement) to destination register. This transfer does not depend on any control function. However, some transfers depend on some specified conditions. The control function is a Boolean function that can be equal to 1 (logic HIGH) or 0 (logic LOW). The control function is terminated by colon. This colon indicates that the transfer operation by executed by the hardware only when the control function is equal 0 or 1 (depending on the logic level).

For example, the register transfer statement with control function is given below.

$$xy' : A \leftarrow B$$

The control statement is xy' and the register transfer statement is $A \leftarrow B$. The register transfer is based on this control statement xy'. When this control hardware issues control signal 1, the content of register B is loaded into register A. The hardware implementation is shown in Fig. 3.3.

The output of register B is connected to the input of regsiter A, and the number of connecting wires (bus) is equal to the number of bits in the registers. Registers A must have load control input so that it can be enabled when the control function $xy' = 1$. The register B also has some control input (it is not shown in Fig. 3.3). It is assumed that register B has a synchronized clock pulse input, the content of register B is transferred to register A when the synchronised clock pulse input, the conten of register B is transferred to register A when the synchronized clock plus is HIGH.

The content of register B appears at the input of register A, but the content is not loaded to register A. The register A is loaded when the control function is HIGH.

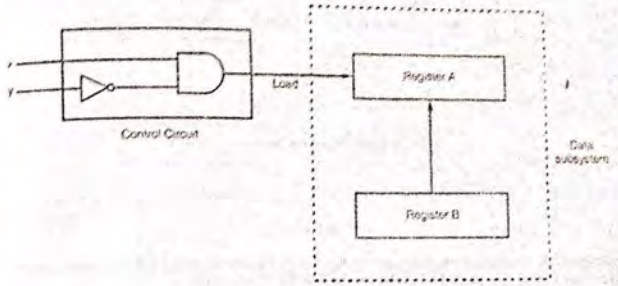


Fig. 3.3. Hardware implementation of data transfer between A and B.

Example 3.1. *Show the block diagram that executes the statement*

$$xT_3 : A \leftarrow B, B \leftarrow A$$

Solution. Control function is xT_3

Register transfer statements are $A \leftarrow B, B \leftarrow A$.

We have the two statements for register transfer, so we cannot perform the transfer function simultaneously. The control function is xT_3. It has two possible values 0 for one register transfer statement and 1 for another register transfer statement.

The function table is given below, it summarises the register transfer with control function.

Function Table

Control function	*Register transfer*	*Comments*
0	$A \leftarrow B$	Load register A
1	$B \leftarrow A$	Load register B

Hardware Implementation

After receiving the load signal 1 from the control subsystem the contents of register A is loaded register B. After receiving the load 0 signal from control sub subsystem the contents of register B is loaded into register A.

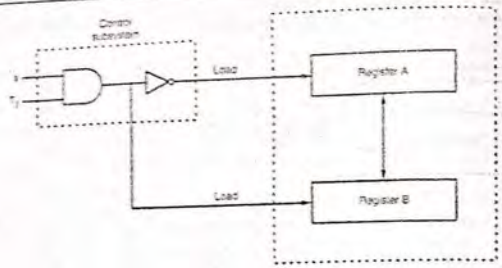


Fig. 3.4.

Example 3.2. *A value can be transferred to a register A by applying to each input a binary signal equivalent to logic 0 or logic 1. Show the implementation of the transfer.*

$$X : A \leftarrow 10010110$$

Solution. The register is loaded when the control function X equals 1. Each bit is stored in each flip-flop of the register.

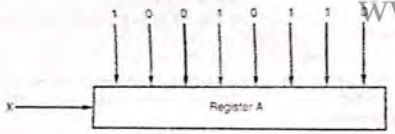


Fig. 3.5.

3.4. ARITHMETIC MICROOPERATIONS

The basic arithmetic microoperations are addition, subtraction, increment, decrement and shift.

Let the arithmetic microoperations defined by the statement

$$R3 \longleftarrow R1 + R2$$

specifies an add microoperation. It states that add the content of register $R1$ and that of register $R2$ and stored the result in register $R3$. Thus, to implement this statement with hardware we need three registers and the digital component that performs the addition operation. Other basic arithmetic microoperations are listed in the Table 3.2.

Arithmetic microoperations are those microoperations that are used to perform arithmetic operations.

Table 3.2. Arithmetic Microoperations.

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

Subtraction is basically implemented through complementation and addition and can be specified by the statement

$$R \leftarrow A + \overline{B} + 1$$

where $\overline{B}$ is the 1's complement of B. When we add 1 to the 1's complement it will give 2's complement of B and adding A to the 2's complement of B gives A minus B *i.e.*, $(A - B)$. The increment and decrement microoperation are implemented with the help of combinational circuit or with a binary up-down counter. They are symbolized by plus-one or minus-one operation executed on the contents of a register. The multiplication operation and division are valid arithmetic operations. Multiplication operation is basically implemented with a sequence of add and shift microoperations and division is implemented with a sequence of subtract and shift microoperations.

www.IPUnotes.com

The block diagram of various arithmetic circuits is discussed in chapter 2 like Half Adder, Full Adder, Adder-Subtractor.

Arithmetic Circuit

The basic arithmetic microoperation listed in Table 3.2 can be implemented by means of a single arithmetic circuit where the basic component of the circuit is parallel adder. Figure 3.6 shows the logic circuit of 4-bit arithmetic circuit.

The arithmetic circuit consists of four full adder that makes the 4-bit adder and four multiplexers for choosing different operations. A and B are the two 4-bit inputs and D is the output. S_1 and S_0 are two selection lines. C_{in} is the input carry connected to the carry input of the full adder in the least significant position and other carries are connected to the next full adder with C_{out} the output carry. The four inputs from A goes directly to the X inputs of the binary adder and four inputs from B are connected to the data inputs of the multiplexers which also receives the complement of B. Other two data inputs of multiplexers are connected to logic 0 and logic 1 respectively.

The output of the circuit is calculated from the following arithmetic sum :

$$D = A + Y + C_{in}$$

where A is the 4-bit input at the X inputs and Y is the 4-bit binary number at the Y input of the circuit. With the help of selection lines S_1 and S_0 and by making input carry C_{in} equal to 0 or 1, the arithmetic microoperation listed in Table 3.3 can be generated.

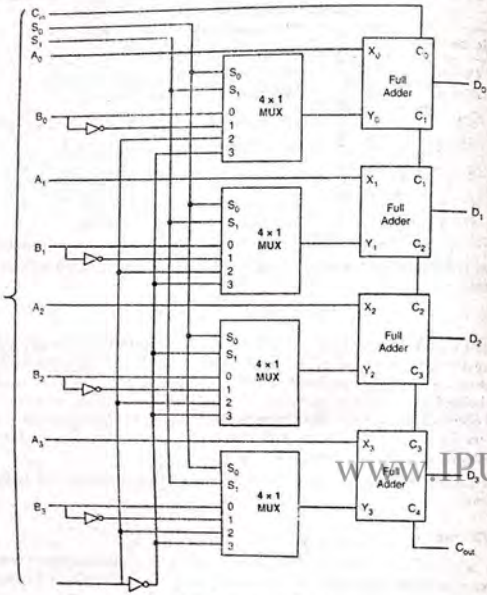


Fig. 3.6. 4-bit Arithmetic Circuit.

Table 3.3. Arithmetic Circuit Function Table.

Select			Input	Output	Microoperation
S_1	S_0	C_{in}	Y	$D = X + Y + C_{in}$	
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\overline{B}$	$D = A + \overline{B}$	Subtract with borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

www.IPUnation.com

Case 1 : When $S_1S_0 = 00$: In this case, the value of B is applied to the Y inputs of the adder. If $C_{in} = 1$, output $D = A + B + 1$ and if $C_{in} = 0$, output $D = A + B$, are the addition microoperation with or without adding the carry input C_{in}.

Case 2 : When $S_1S_0 = 01$: In this case the complement of B is applied to the Y inputs. If $C_{in} = 1$, the output $D = A + \overline{B} + 1$, which produces A plus the 2's complement of B and hence the subtraction $A - B$. If $C_{in} = 0$, the output $D = A + \overline{B}$, which is equivalent to $A - B - 1$ *i.e.* subtract with borrow.

Case 3 : When $S_1S_0 = 10$: In this case, the inputs from B are neglected, and 0's are inputed into the Y inputs. If $C_{in} = 1$, $D = A + 1$, *i.e.* the value of A is incremented by 1 and when $C_{in} = 0$, $D = A$ and hence we have a direct transfer from input A to output D.

Case 4 : When $S_1S_0 = 11$: In this case, all 1's are inserted into the Y inputs. A number with all 1's is basically equal to the 2's complement of 1. Thus, if $C_{in} = 1$, the output $D = A - 1 + 1 = A$, and hence a direct transfer from input A to output D. When $C_{in} = 0$, $D = A - 1$, *i.e.* the value of A is decremented by 1.

Ripple Carry Adder or Parallel Adder

As we know that a single full-adder is capable of adding three one bit binary numbers or two one bit numbers and one previous carry, in order to add binary numbers with more than one bit, additional full adders must be emploted. A four bit parallel adder can be constructed using four full adders as shown in Fig. 3.7. These four full adders are connected in cascade *i.e.*, the carry output of each adder is connected to the carry input of the next higher-order adder. An n-bit parallel adder is constructed using 'n' number of full adders.

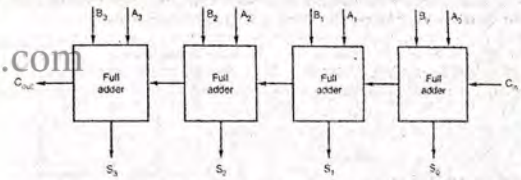


Fig. 3.7. Block diagram of parallel adder

The carry from the right most position (LSB) has to propagate through all the bit positions. If t_d is the time delay for each full-adder stage. For n-bit parallel adder has the maximum propogation delay is nt_d. This is the major drawback of parallel adder.

Serial Adder

The speed of the adder is an important design parameter of adder circuits. If the speed is not important, then the cost effective option is to use a serial adder, in which bits are added one pair at a time. The serial adder method uses only one full adder circuit and a storage device (flip-flop) to hold the generated output carry. Let $A = A_{n-1} A_{n-2} \ldots A_0$ and $B = B_{n-1} B_{n-2} \ldots B_0$ be two unsigned numbers that have to be added to produce sum $S_{n-1} S_{n-2} \ldots S_0$. The two number A and B are stored in the register A and register B respectively. The pair of bits in A_0 and B_0 (which are stored in right shift register) are transferred serially one at a time, through the single full adder to produce a sum and carry and this carry is stored in the flip-flop. The stored output carry from one pair of bits is used as an input carry for the next pair of bits as

shown in Fig. 3.8. The addition of two n-bit numbers starts from the least significant bits and proceeds step by step to the most significant bits in different time sequences. Initially the D flip-flop is cleared.

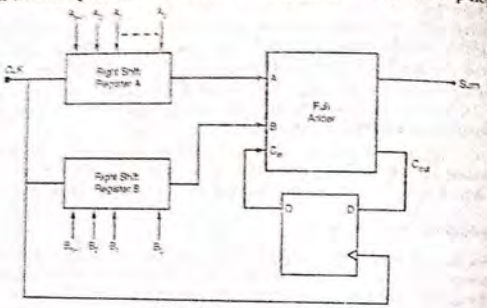
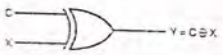


Fig. 3.8. Block diagram of Serial adder.

4-bit Parallel Adder/Subtractor

The 4-bit parallel binary adder/subtractor circuit is shown in Fig. 3.9(*a*). It performs the operation of both addition and subtraction. It has two 4-bit inputs $A_3 A_2 A_1 A_0$ and $B_3 B_2 B_1 B_0$.



Inputs		Outputs
C	X	Y
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 3.9(a) Symbol of EXOR gate.

www.IRUnotes.com

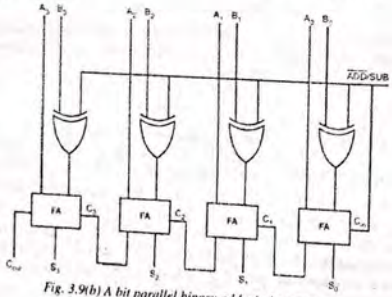


Fig. 3.9(b) A bit parallel binary adder/subtractor

We can not perform the two operations simultaneously, so we need one control signal to perform this operation. The $\overline{ADD/SUB}$ control line, connected to C_{in} of LSB of the full adder, is used to perform the operations of addition and subtraction. The EXOR gates are used as controlled inverters. From the EXOR gate truth table we know that when one of the inputs is LOW the output is of the same value of the other input and when one of the inputs is HIGH the output is the complement of the other input. Fig. 3.9(*a*) shows the symbol EXOR gate and above table shows the truth table.

If $C = 0$, the input variable X, either 0 or 1 wil be complemented and transferred to output. By using these EXOR gate properties, we use this gate in the 4 bit adder/subtractor circuit which is shown in Fig. 3.9(*b*).

Case 1 : $\overline{ADD/SUB} = 1$. Now the controlled inverter (EXOR gate) produces the 1's complement of $B_3 B_2 B_1 B_0$. Since 1 is given to C_{in} of the LSB bit of the adder, it is added to the complemented output of EXOR gate output, it is equal to 2's complement of $B_3 B_2 B_1 B_0$. The 2's complemented $B_3 B_2 B_1 B_0$ will be added to $A_3 A_2 A_1 A_0$ to produce the sum, the produced output of $S_3 S_2 S_1 S_0$ is the difference between $A_3 A_2 A_1 A_0$ and $\overline{B_3 B_2 B_1 B_0}$.

Case 2 : $\overline{ADD/SUB} = 0$. Now the controlled inverter is transferred $B_3 B_2 B_1 B_0$ four bit to full adder, this 4 bit is added with $A_3 A_2 A_1 A_0$ to produce sum and carry.

Binary Incrementor

The increment microoperation adds one to a number in a register. For example, if a 4-bit number 1110, it will be 1111 after it incremented.

The diagram of a 4-bit combinational circuit incrementer is shown in Fig. 3.10. One of the inputs to the least significant half-adder (HA) is connected to logic-1 and the other input is connected to the least significant bit of the number to be incremented. The output carry from one half-adder is connected to one of the inputs of the next-higher-order half-adder. The circuit receives the four bits from A_0 through A_3, adds one to it, and generates the incremented output in S_0 through S_3. The output carry C_4 will be 1 only after incrementing binary 1111. This also causes outputs S_0 through S_3 to go to 0.

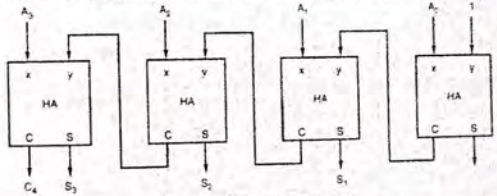


Fig. 3.10. 4-bit binary incrementer

3.5. LOGICAL MICROOPERATIONS

Logic microoperations are those microoperations that are used to perform logical operations, such as AND, OR, NOT etc. The microoperations which perform these operations on individual pairs of bits stored in registers are said to be logic microoperations. These operations consider each bit of the register separately and treat it as a binary variable. The XOR (exclusive OR) operation between the content of two register $R1$ and $R2$ can be represented as

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

The symbol ⊕ represents the XOR operation. Also, special symbols are used for logic microoperations OR, AND and complement operations. The symbol ∨ is used to denote an OR microoperation, and symbol ∧ is used to denote AND microoperation.

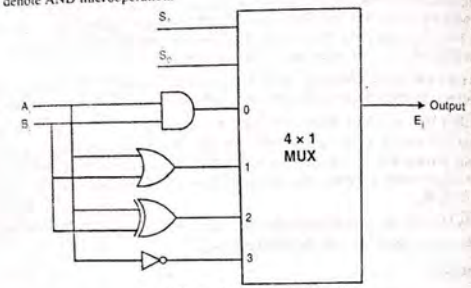


Fig. 3.11. One Stage of Logic Circuit.

There are 16 different logic microoperations that can be performed with two binary variables and are listed below in the Table 3.4.

Table 3.4. Sixteen Logic Microoperations.

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

www.IPUnation.com

Although there are 16 logic microoperations most of the computers uses only four— AND, OR, XOR, and complement from which all others can be derived. Figure 3.11 depicts the hardware implementation of four basic logic microoperations.

The output of the four gates (AND, OR, XOR, NOT) are applied to the data input of the 4 × 1 multiplexer which will give output according to the function Table 3.5.

Table 3.5. Function Table.

S_1	S_0	*Output*	*Operator*
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \overline{A}$	Complement

The selection lines S_1S_0 choose one of the data inputs of the multiplexer and directs it to the output. The diagram shows one stage of circuit. For n-bits the diagram must be repeated n times, for $i = 0, 1, 2, \ldots, n-1$.

3.5.1. Selective Microoperations

Selective-Set Operation

The *selective-set* operation sets to 1 the bits in register A where there are corresponding 1's in register B. It does not affect bit positions that have 0's in B. The following numerical example clarifies this operation :

```
 1010  A before
 1100  B (logic operand)
 ----
 1110  A after
```

It is similar to Logic-OR microoperation.

Selective-complement

The *selective-complement* operation complements bits in A where there are corresponding 1's in B. It does not affect bit positions that have 0's in B. For example :

```
 1010  A before
 1100  B (logic operand)
 ----
 0110  A after
```

It is similar to Exclusive-OR microoperation.

Selective-clear

The *selective-clear* operation clears to 0 the bits in A only where there are corresponding 1's in B. For example :

```
 1010  A before
 1100  B (logic operand)
 ----
 0010  A after
```

It is similar to $A \wedge B'$ microoperation.

Mask Operation

The *mask* operation is similar to the selective-clear operation except that the bits of A are cl only where there are correspoinding 0's in B. The mask operation is an AND microoperation as seen the following numerical example :

```
1010  A before
1100  B (logic operand)
1000  A after masking
```

It is similar to Logic-AND operation.

3.6. SHIFT MICROOPERATIONS

Shift microoperations are used for serial transfer of data. The shifting of bits of register can be in direction left or right, during which the first flip-flop receives its binary information from the serial i When the operation is shift-left, the serial input transfers a bit into the right most position and when operation is shift-right the bit from the serial input is transferred into the left most position.

Shift microoperation can be classified as :

(*a*) Logical, (*b*) Circular, (*c*) Arithmetic.

Shift microoperations are those microoperations that are used to perform shifting of bits in either or right direction.

1. Logical Shift

It is the easiest to perform as the bits are simply shifted to the right or to the left. If it is shift- a 0 bit is transferred at the right most position through serial input and the leftmost bit is lost. Similarly, if shift-right the right most bit is lost and a 0 bit is added at the leftmost place.

For example, $R1 \leftarrow \text{shl } R1$

$R2 \leftarrow \text{shr } R2.$

shl and shr are the symbols used for shift-left and shift-right microoperations. The above two statem specifies a 1-bit shift to the left of the content of register $R1$ and a 1-bit shift to the right of the content of reg $R2$ respectively.

2. Circular Shift

Also known as rotate operation, simply circulates the bits of the register around the two ends. bit is added or lost. If it is left circular shift, the bits are shifted in left direction and the rightmost takes the left most bit position. Similarly, if it is right circular shift, the bits are shift in right direc and the leftmost bit takes the rightmost bit position.

3. Arithmetic Shift

An arithmetic shift is a microoperation that shifts a signed binary number to the left or right. arithmetic left or arithmetic right shift operations, we need to consider the sign bit also, beca arithmetic shift requires that the sign bit (the leftmost bit) remains unchanged. 0 is the sign bit positive and 1 is sign bit for negative numbers.

The Fig. 3.12 shows a register of n-bits, where bit b_{n-1} in the leftmost position contains the sign bit of a number. b_{n-2} is the most significant bit and b_0 is the least significant bit of number.

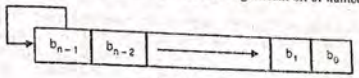


Fig. 3.12.

In case of arithmetic shift-right (ashr) the sign bit remains unchanged and shifts the number, including the sign bit, to the right. The bit b_0 is lost. Thus, b_{n-1} remains unchanged, b_{n-2} receives b_{n-1} and so on.

In case of arithmetic shift-left (ashl), 0 is inserted into bit position b_0 and shifts all other bits to the left. The initial bit b_{n-1} is lost and replaced by bit b_{n-2}. If the bit in b_{n-1} changes the value after the shift a sign reversal occurs.

4. 4-bit Shifter

The 4-bit combinational circuit called shifter constructed with the help of multiplexers is explained in Fig. 3.13. It consists of four data inputs I_0 to I_3 and four data output O_0 to O_3. The two serial inputs, one for shift left and other for shift right are S_L and S_R respectively. The function table for the circuit is given in Table 3.6. When the selection input $S = 0$, the input data are shifted right and when $S = 1$, the input data are shifted left. The function table explains which input goes to each output after the shift operation. In general, a shifter having n data inputs and n data outputs requires n multiplexers.

Table 3.6. Function Table.

Select	Output			
S	O_0	O_1	O_2	O_3
0	S_R	I_0	I_1	I_2
1	I_1	I_2	I_3	S_L

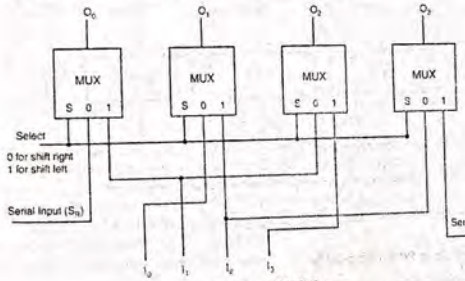


Fig. 3.13. 4-bit Shifter.

www.JBLibrary.com

3.7. BUS AND MEMORY TRANSFER

A computer has many registers and paths must be provided to transfer the information from one register to another register. This transfer done by group wires called bus. To achieve a reasonable speed of operation, a computer must be recognised so that all its units can handle one full word of data. When data is transferred between units, all its bits are transferred in parallel, that is, the bits are transferred simultaneously over different wires. Many wires are required to establish the necessary connections. The number of wires will be excessive if separate lines are used between each register and all other registers in the system. A scheme for transferring information between the registers is a common bus system. Some control lines are required to manage the common bus system.

Bus: A group of wires that connected several devices to carry the data (or) information is called as bus.

Bus Transfer: The data transfer between various blocks connected to the common bus is called bus transfer.

Memory Transfer: The data is stored (write) into memory or read from memory is called memory transfer.

3.7.1. Bus Transfer

A more efficient scheme for transferring information between registers in a multiple register configuration is a common bus system. It is shared by all the units. Switches are required to enable path to be shared as shown in Fig. 3.14. These switches are implemented by multiplexer or tristate non inverting buffers. One common control signal is used to control all the switches, the switches are closed (connection enabled) or open (connections disabled).

www.IPUnation.com

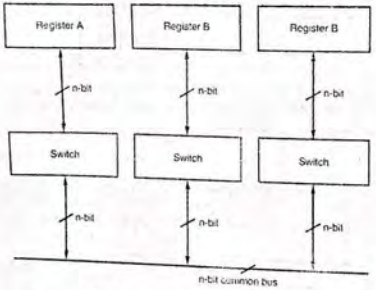


Fig. 3.14. Shared common bus system.

3.7.1.1. Shared Data Path using MUX

Figure 3.15 shows the shared data path using multiplexers. The required multiplexers is based on the number of registers to be used in the data path.

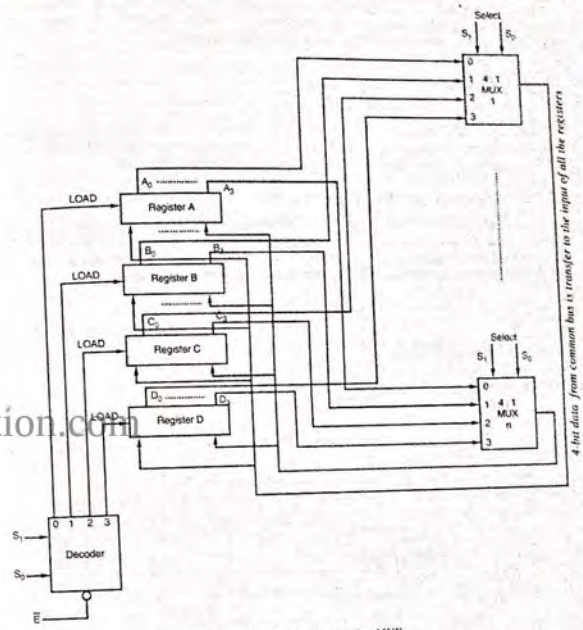


Fig. 3.15. Shared data path using MUX.

The multiplexers select one source register bus, and the decoder selects one destination register to transfer the information. The 4 bits in the same significant position in the register group go through a 4 to 1 multiplexer to form one line of bus. For n bit registers, n multiplexers are needed to produce an n-common line bus (or) shared data path. The transfer of information from the bus into its destination register is accomplished by activbating load control of that register. If the decoder is not enabled, no information will be transferred between the registers.

For example, consider the following statement

$$C \leftarrow B$$

From the statement, we can identify the source register and destination register. The source register is B and the destination register is C. The multiplexer and decoder select the lines which satisfy the following condition to transfer the data from register B into register C.

$$\text{MUX select line } S_1S_0 = 01$$

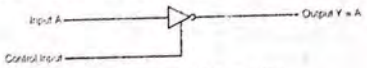
$$\text{Decoder select line } S_1S_0 = 10$$

$$\text{Decoder enable } \overline{E} = 0$$

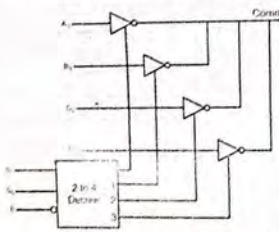
When the select lines are equal to 01, all the MUXs are enabled and the second input places the content of B register to common bus. At next clock pulse, the content of B, being on the bus, is loaded in register C, when the decoder sends the load signal to register C, using $S_1S_0 = 10$.

3.7.1.2. Shared Data Path Using Tristate Buffer

The graphic symbol of tristate buffer is shown in Fig. 3.16(*a*). It has one input and one output controlled by one control input. When the control inpuyt is 1, the buffer output is enabled and the buffer allows to transfer the data from input to output terminal through them. When the control input is equal to 0, the buffer output is disabled and it does not allow transfer of data through it.



(*a*) Graphical symbol of tristate buffer.



(*b*) Shared data path using tristate buffer.

Fig. 3.16.

When more switches are required for sharing the data path line, a shared data path is formed by using tristate buffer in place of multiplexer. Figure 3.16(*b*) shows the 4 bit shared data path using tristate buffer. The output of four buffers are connected together to form a single/common bus line. The connected buffers must be controlled by the control input of each buffer so that only one output of the four tristate buffers has access to the bus line. While all the other buffers act on high impedance (open circuited) state. A decoder is needed to activate the buffer to generate the control signals. When the enable input of decoder is 0, any one of its buffers are enabled based on the selection line of the decoder. When the enable input is high all of its four buffers are disabled.

3.7.2. Memory Transfer

The basic operation of memory transfer is Fetch (or Read) and Store (or Write). The Read operation transfers a copy of the contents from memory location to CPU. The word in memory remains unchanged. The store operation transfers the word information from the CPU to a specific memory location, destroying previous contents of that location. The memory word is symbolised by the letter M. The required information is selected from the memory location by address, it is stored in the address register symbolised by AR. The data is transferred to another register called data register, symbolised by DR. A simple read operation is stated as follows :

$$\text{Read : DR} \leftarrow \text{M[AR]}$$

This **read** operation transfer the information into data register (DR) from the memory word M selected by the address in address register (AR).

The **write** operation transfers the content of data register to a memory word M selected by the address. The write operation is stated as follows :

$$\text{Write : M[AR]} \leftarrow \text{B}$$

The contents of register B is transferred to selected memory location. When it receives the control signal which is high.

3.7.3. Bus Organization

A bidirectional bus used to carry data between two units is data bus. A unidirectional bus used to carry memory addresses is called memory bus. The way in which different bus are connected to form common bus, so that CPU, memory and I/O devices can used common bus when required is called bus organization.

www.IPUnotes.com

A basic computer consists of a memory unit, a control unit and registers. There must be a path that can be used to transfer information between memory and registers or among registers. A most efficient way of transferring information from source to destination in a system with many registers is to use a common bus. Fig. 3.17 shows the connection of eight registers and memory unit of 4096 × 16 to a common bus system. The eight registers are Address Register (AR), Program Counter (PC), Data Register (DR), Accumulator (AC), Instruction register (IR), Temporary Register (TR), Input Register (INPR) and Output Register (OUTR). The Fig. 3.10 consist of 16-bit common bus.

The outputs of seven registers and memory are connected to the common bus. The specific output that is selected for the bus lines at any given time is determined by the binary value of the selection lines S_2, S_1 and S_0 is shown in Table 3.7. The numbers along each output lines shows the decimal equivalent of the required binary selection.

Table 3.7. Function Table.

S_2	S_1	S_0	Register
0	0	1	AR
0	1	0	PC
0	1	1	DR
1	0	0	AC
1	0	1	IR
1	1	0	TR
1	1	1	Memory

For example, when $S_2S_1S_0 = 011$, the 16-bit outputs of DR are placed on the bus lines.

The lines from the common-bus are connected to the input of each registers and the data inputs of the memory. The particular register whose LD (Load) input is enabled receives the data from the bus. The memory receives the data from the bus when its write input is enabled.

The memory places its output onto the bus when the read input is activated and $S_2S_1S_0 = 111$.

The registers DR, AC, IR, TR are of 16-bits Two Register PC and AR have 12-bits since the store address.

When the contents of AR and PC are placed on the bus the four most-significant bits are set to 0. When AR and PC receives the data from the bus only 12 least significant bits are transferred to the regis

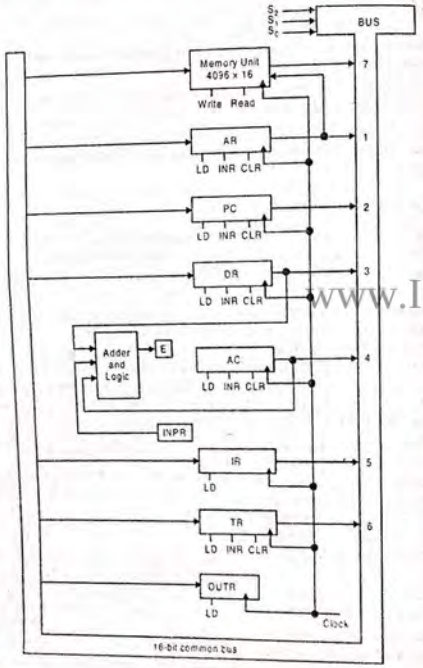


Fig. 3.17. Common Bus Organization.

The input register INPR and OUTR have 8 bits hence they communicate only with the 8 least significant bits in the bus. INPR is connected to bus to provide information but OUTR is connected to bus only to receive information from the bus. INPR receives a character from I/O device which is transferred to AC and OUTR receives a character from AC and delivers to an output device. No transfer is there from OUTR to any of the other registers.

The 16-bit common bus receives information from six registers and the memory unit. Also, the 16-bit common bus are connected to the inputs of six registers and the memory unit. The five registers have three control signals LD (Load), INR (increment) and CLR (clear). Two registers have only LD (load) control signal connected to the common bus. AR is also connected to the memory address. Thus, AR always specify the memory address. During a memory write operation the content of any register can be specified for the memory data and similarly during memory read operation any register except AC receives the memory data. The 16-bit AC receives inputs from adder and logic circuit which receives input from three registers. These three registers are 16-bit AC, 16 bit data register DR and 8-bit inputs come from input register INPR. The inputs from DR and AC are used for arithmetic and logic micro-operations. The function table shown in Table 3.8 shows the binary value of selection line $S_2S_1S_0$ that selects one of the register.

For example, in order to transfer contents of PC to Address Register, the computer requires the following instructions :

(1) Set the selection variables $S_2S_1S_0 = 010$

(2) Transfer the contents of PC to the bus

(3) Enable LD input of AR

(4) Transfer contents of bus into AR

www.IPUnation.com

3.7.4. Bus Architecture

A group of wires connecting two or more devices and providing a path to perform communication is called bus. A bus that connects major computer components/modules (CPU, Memory, I/O) is called a system bus. These system buses are separated into three functional groups.

1. Data Bus
2. Address bus
3. Control bus.

Data Bus

The data bus consists of 8, 16, 32 or more parallel lines. The data bus lines are bidirectional. This means that CPU can read data on these lines from memory or from a port as well as data out on these lines to a memory location or to a port.

Address Bus

It is a unidirectional bus. The address bus consists of 16, 20, 24 or more parallel lines. The CPU sends out the address of the memory location or I/O port that is to be written or read from by using this address bus.

Control Bus

Control lines regulate the activity on the bus. The CPU sends signals on the control bus to enable outputs of addressed memory device or port device.

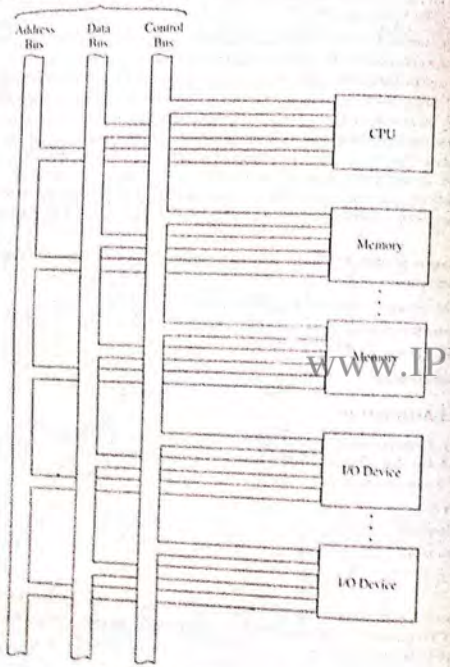


Fig. 3.18 Bus interconnection scheme.

Figure 3.15 shows the bus interconnection scheme. The complexity of bus control logic depends on the amount of translation needed between the system bus and CPU, the timing requirements, whether or not interrupt management is included and size of the overall system. For smaller systems, control signals of the CPU can be used directly to reduce the complexity. But large systems with several interfaces would

need bus driver and receiver circuits connected to the bus in order to maintain adequate signal quality. In most of the processors, multiplexed address and data buses are used to reduce the number of pins. During the first part of the bus cycle, address is present on this bus, afterwards, the same bus is used for data transfer purpose. So latches are required to hold the address sent by the CPU initially.

IMPORTANT NOTES

- **Data Bus:** It is the bidirectional bus used to carry the data between the system units.
- **Address Bus:** It is a unidirectional bus. Its function to select the particular device (or) memory location etc.
- **Control Bus:** It is also unidirectional bus. Its function is to control the units based on the request.

3.7.5. Bus Arbitration

Amongst several masters and slave units are connected to a shared bus, it may happen that more than one master or slave units will request access to the bus at a same time. In such situations bus access is give to the master having highest priority. A mechanism which decides the selection of current master to access bus is known as bus arbitration. Three different mechanisms are commonly used for this.

1. Daisy chaining
2. Parallel arbitration
3. Independent requesting

www.IPUnation.com

IMPORTANT NOTES

- **Bus Arbitration:** A mechanism which decides the selection of current master to access bus is known as bus arbitration.

3.7.5.1. Daisy Chaining

The system connection for daisy chaining method are shown in Fig. 3.23. Daisy chaining method is cheaper and simple method. All masters make use of the same line for bus request. The bus grant signal serially propagates through each master until it encounters the first one that is requesting access to the bus. This master blocks the propagation of the bus grant signal, activates the busy line and gains control of the bus. Therefore, any other requesting module will not receive the grant signal and hence cannot get the bus access.

Advantages

1. It is a simple and cheapter method.
2. It requires the least number of lines and this number is independent of the number of masters in the system.

Disadvantages

1. The propagation time delay of bus grant signal is proportional to the number of masters in the system. This makes arbitration time slow and hence limits the number of masters in the system.
2. The priority of the master is fixed by the physical location of master.
3. Failure of any one master causes the whole system to fail.

and S_2. A_{i-1} and A_{i+1} are the two data inputs of multiplexer for shift-right operation and shift-left operation respectively. The output carry C_{i+1} is connected to the input carry C_i of the next stage in sequence.

Table 3.8 lists eight arithmetic operation, four logic operation and two shift operations. S_3, S_2, S_1, S_0 and C_{in} selects a particular operation. C_{in} is used to select the arithmetic operation only.

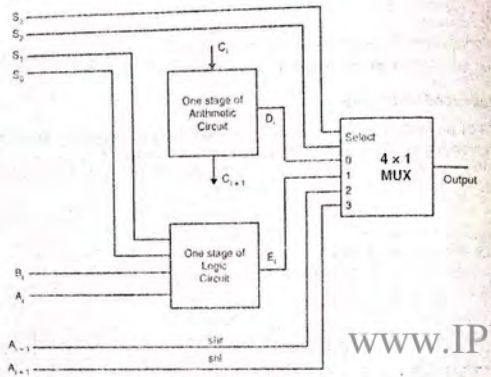


Fig. 3.22. Arithmetic Logic Shift Unit.

Table 3.8. Function Table for Arithmetic Logic Shift Unit.

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

www.IPUnation.com

SOLVED EXAMPLES

Example 3.3. *Show the block diagram of the hardware (similar to Fig. 3.23) that implements the following register transfer statement :*

$$yT_2: R2 \leftarrow R1, R1 \leftarrow R2$$

Solution.

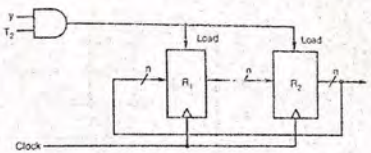


Fig. 3.23

Example 3.4. *Represent the following conditional control statement by two register transfer statements with control functions.*

If (P = 1) then (R1 ← R2) else if (Q = 1) then (R1 ← R3)

Solution.

$$P : R1 \leftarrow R2$$
$$P'Q : R1 \leftarrow R3$$

Example 3.5. *Show the hardware that implements the following statement. Include the logic gates for the control function and a block diagram for the binary counter with a count enable input*

$$xyT_0 + T_1 + y'T_2: \; AR \leftarrow AR + 1$$

Solution.

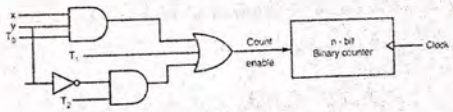


Fig. 3.24

Example 3.6. *A digital computer has a common bus system for 16 registers of 32 bits each. The bus is constructed with multiplexers.*

(a) *How many selection inputs are there in each multiplexer ?*

(b) *What size of multiplexers are needed ?*

(c) *How many multiplexers are there in the bus ?*

Solution.

(*a*) Read memory word specified by address in AR into register R_2.

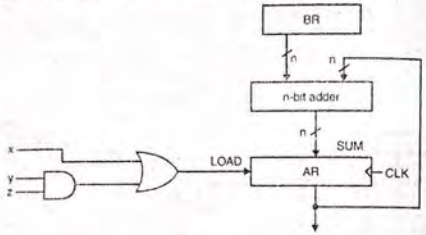


Fig. 3.28.

REVIEW QUESTIONS

1. Write short note on micro-operation.
2. Define bus.
3. Define register transfer language (RTL).
4. What is main advantage of RTL.
5. Explain the concept of memory and bus transfer.
6. Draw the block diagram of ALU to perform the following operstions (use MUX)
   (*i*) AND (*ii*) OR
   (*iii*) NOR (*iv*) NAND
7. Explain the different shift operations with examples.
8. Design a 4-bit arithmetic circuit which is capable of performing the following micro-operation
   (*a*) Addition (with and without carry)
   (*b*) Subtraction (with or without borrow)
   (*c*) Increment
   (*d*) Decrement
   (*e*) Transfer
9. A digital computer has a common bus system for 8 registers of 8 bit each. The bus is construcied with MUX
   (*i*) How many select inputs are there in each MUX.
   (*ii*) What size of MUX are needed.
   (*iii*) How many MUX's are there in bus.

www.IPUnotes.com

Introduction to Microprocessor

4

4.1. INTRODUCTION

Microprocessors are regarded as one of the most important devices in our everyday machines called computers. A microprocessor is an IC (integrated circuit) that contains most of the digital logic required to implement a computer except for memory and I/O (input/output) devices. This is to describe how to interface a microprocessor to memory and I/O devices to make a microcomputer.

Memory and I/O devices usually can't be connected directly to the microprocessor. The logic circuits required to interface the microprocessor and other chips is often called glue logic which is shown in the Fig. 4.1. This additional logic can vary from none in the case of microcontrollers (microprocessors that contain on-chip memory and I/O devices) to "intelligent" peripheral interfaces that are as complex as the microprocessor itself.

We can say that ***microprocessor is an electronic device that works as the central processing unit (CPU) of a computer, providing computational control. Micro-processors are also used in other advanced electronic systems, such as computer printers, automobiles, and jet air lines.*** Typical microprocessors have arithmetic and logic functional units as well as the associated control logic, instruction processing circuitry and a portion of the memory hierarchy. Portions of the interface logic for the input/output (I/O) and memory subsystems may also be infused, allowing cheaper overall systems. While many microprocessors and single chip designs, some high-performance designs rely on a few chips to provide multiple functional units and relatively large caches. When combined with other integrated circuits that provide storage for data and programs, often on a single semiconductor base to form a chip, the microprocessor becomes the heart of a small computer, or microcomputer.

Fig. 4.1. Glue Logic

A microprocessor can perform any information processing task. It is a truly general-purpose information-processing device. It fetches the information, executes the information, control its operation and stored the information electronically. This is the principle of "stored program control". Without a program the microprocessor can do nothing. Furthermore, microprocessors can only perform information-processing tasks. To take action on the outside world, or to receive signals from it, connection must be provided between the microprocessor's representation of information (as digital electronic signals) and the real world representation.

Developed during the 1970s, the microprocessor became most visible as the central processor of the personal computer. Microprocessors also play supporting roles within larger computers as smart controllers for graphics displays, storage devices, and high-speed printers. However, the vast majority of microprocessors are used to control everything from consumer appliances to smart weapons. The microprocessor has made possible the inexpensive hand-held electronic calculator, the digital wristwatch, and the electronic game. Microprocessors are used to control consumer electronic devices, such as the programmable microwave oven and videocassette recorder; to regulate gasoline consumption and antilock brakes in automobiles; to monitor alarm systems; and to operate automatic tracking and targeting systems in aircraft, tanks, and missiles and to control radar arrays to track and identify aircraft, among other defense applications.

4.2. INTEL'S MICROPROCESSORS

1. 4-Bit Microprocessors

Microprocessor 4004 (1971)

www.IPUnotes.com

It was a four bit microprocessor. Historically, the 4-bit microprocessor was the first general purpose microprocessor introduced in the market. The basic design of the early microprocessors was derived from that of the desk calculator. The Intel 4004, a 4-bit design, was the grandfather of microprocessors. Introduced in late 1971, the 4004 was originally designed for a Japanese manufacturer as the processing element of a desk calculator, it was not designed as a general-purpose computer. The shortcomings of the 4004 were recognized as soon as it was introduced. But it was the first general purpose computing device on a chip to be placed on the market. Many of the chips introduced at about the same time by other companies were, in fact, mere calculator chips.

Features

(i) Introduced-November 15, 1971
(ii) Clock rate-740 kHz
(iii) 0.07 MIPS
(iv) Bus Width-4 bits (multiplexed address/data due to limited pins)
(v) PMOS technology
(vi) Number of Transistors-2,300 at 10 μm
(vii) Addressable Memory-640 bytes
(viii) Program Memory-4 KB (4 KB)
(ix) Originally designed to be used in Busicom calculator

2. 8-Bit Microprocessors

Microprocessor 8008 (1972)

The 8008 increased the 4004's word length from four to eight bits, and doubled the volume of information that could be processed. It was still an invention in search of a market however, as the technology world was just beginning to view the microprocessor as a solution to many needs.

Features

(i) Introduced-April 1, 1972
(ii) Clock rate-500 kHz
(iii) 0.05 MIPS
(iv) Bus Width-8 bits (multiplexed address/data due to limited pins)
(v) Enhancement load-PMOS logic
(vi) Number of Transistors-3,500 at 10 μm
(vii) Addressable memory-16 KB
(viii) Typical in dumb terminals, general calculators, bottling machines

Microprocessor 8080 (1974)

The 8080 were 20 times as fast as the 4004 and contained twice as many transistors. The 8080, designed as a successor to Intel's 8008, was the first powerful microprocessor introduced on the market. Several others microprocessors of similar performance were introduced on the market within a year after the 8080 appeared and several additional powerful designs were introduced later. Technically, however, the 8080 was the most powerful product on the market. Furthermore, Intel was the first company to invest in the development of support chips and software for its products. This ensured the continued success of the 8080 because its performance was then sufficient for many applications. The early 8080 competitors were introduced with at least a nine-month delay and failed to dislodge it. The 8080 is still sold today thought It has been largely eclipsed by successor products—most notably the 8085 microprocessor. The 8080 is available in three versions: the standard 8080A with a 2MHZ clock, the 8080A-2, and the 8080A-1 with a 3 MHZ clock.

Features

(i) Introduced-April 1, 1974
(ii) Clock rate-2 MHz
(iii) 0.64 MIPS
(iv) Bus Width-8 bits data, 16 bits address
(v) Enhancement load-NMOS logic
(vi) Number of Transistors-6,000
(vii) Assembly language downwards compatible with 8008.
(viii) Addressable memory-64 KB
(ix) Up to 10X the performance of the 8008
(x) Used in the Altair 8800, Traffic light controller, cruise missile
(xi) Required six support chips versus 20 for the 8008

Microprocessor 8085 (1976)

The 8085 is an 8-bit microprocessor introduced by Intel in 1976.The 8085 was the follow-on processor to the very successful Intel 8080A processor. The 8085 got its name because it was Intel's first 5 volt

microprocessor. The 8085 was 100% software compatible with the 8080A with increased system performance. The initial 8085's were based on NMOS technology and the later "H" versions were based on HMOS technology. The 8085 used 6,500 transistors.

The 8085 incorporated all the features of the 8224 (clock generator) and the 8228 (system controller), increasing the level of system integration.

Features

(*i*) Introduced-March 1976
(*ii*) Clock rate-5 MHz
(*iii*) 0.37 MIPS
(*iv*) Bus Width-8 bits data, 16 bits address
(*v*) Depletion load-NMOS logic
(*vi*) Number of Transistors-6,500 at 3 μm
(*vii*) Binary compatible downwards with the 8080.
(*viii*) Used in Toledo scale. Also was used as a computer peripheral controller - modems, hard disk, printers, etc.
(*ix*) High level of integration, operating for the first time on a single 5 volt power supply, from 12 volts previously. Also featured serial I/O, 3 maskable interrupts, 1 Non-maskable interrupt, 1 externally expandable interrupt [8259], status, DMA.

3. 16-Bit Microprocessors

Microprocessor 8086 (1978)

www.IPUtations.com

Intel's 8086 microprocessor is a *first member* of *x* 86 families of processors. Advertised as a "source code compatible" with Intel 8080 and Intel 8085 processors, the 8086 was not object code compatible with them. The 8086 has complete 16-bit architecture, 16-bit internal registers, 16-bit data bus, and 20-bit address bus (1 MB of physical memory). Because the processor has 16-bit index registers and memory pointers, it can effectively address only 64 KB of memory.

Features

(*i*) Introduced-June 8, 1978
(*ii*) Clock rates:
- 5 MHz with 0.33 MIPS
- 8 MHz with 0.66 MIPS
- 10 MHz with 0.75 MIPS

(*iii*) The memory is divided into odd and even banks. It accesses both the banks simultaneously in order to read 16 bit of data in one clock cycle.
(*iv*) Bus Width-16 bits data, 20 bits address
(*v*) Number of Transistors-29,000 at 3 μm
(*vi*) Addressable memory-1 megabyte
(*vii*) Up to 10X the performance of 8080 (typically lower)
(*viii*) Used in portable computing, and in the IBM PS/2 Model 25 and Model 30. Also used in the AT&T PC6300 / Olivetti M24, a popular IBM PC-compatible (predating the IBM PS/2 line.)
(*ix*) Used segment registers to access more than 64 KB of data at once, which many programmers complained made their work excessively difficult.

Microprocessor 8088 (1979)

Created as a cheaper version of Intel's 8086, the 8088 was a 16-bit processor with an 8-bit external bus. This chip became the most ubiquitous in the computer industry when IBM chose it for its first PC. The success of the IBM PC and its clones gave Intel a dominant position in the semiconductor industry.

Features

(*i*) Introduced-June 1, 1979
(*ii*) Clock rates:
4.77 MHz with 0.33 MIPS
9 MHz with 0.75 MIPS
(*iii*) Internal architecture-16 bits
(*iv*) External bus Width-8 bits data, 20 bits address
(*v*) Number of Transistors-29,000 at 3 μm
(*vi*) Addressable memory-1 megabyte
(*vii*) Identical to 8086 except for its 8 bit external bus (hence an 8 instead of a 6 at the end)
(*viii*) Used in IBM PCs and PC clones

Microprocessor 80186 (1982)

Features

(*i*) Introduced -1982
(*ii*) Included two timers, a DMA controller, and an interrupt controller on the chip in addition to the processor (These were at fixed addresses which differed from the IBM PC, making it impossible to build a 100% PC-compatible computer around the 80186.)
(*iii*) Added a few opcodes and exceptions to the 8086 design; otherwise identical instruction set to 8086 and 8088.
(*iv*) Used mostly in embedded applications - controllers, point-of-sale systems, terminals.
(*v*) Used in several non-PC-Compatible MS-DOS computers including RM Nimbus, Tandy 2000.

Microprocessor 80286 (1982)

With 16 MB of addressable memory and 1 GB of virtual memory, this 16-bit chip is referred to as the first "modern" microprocessor. Many novices were introduced to desktop computing with a "286 machine" and it became the dominant chip of its time. It contained 130,000 transistors and packed serious compute power (12 MHz) into a tiny footprint.

Features

(*i*) Introduced-February 1, 1982
(*ii*) Clock rates:
6 MHz with 0.9 MIPS
8 MHz, 10 MHz with 1.5 MIPS
12.5 MHz with 2.66 MIPS
16 MHz, 20 MHz and 25 MHz are available
(*iii*) Bus Width-16 bit data, 24 bit address.

(iv) Included memory protection hardware to support multitasking operating systems with per-process address space
(v) Number of Transistors-134,000 at 1.5 μm
(vi) Addressable memory-16 MB (16 MB)
(vii) Added protected-mode features to 8086 with essentially the same instruction set
(viii) 3-6X the performance of the 8086
(ix) Widely used in IBM-PC AT and AT clones contemporary to it

4. 32-Bit Microprocessors

Microprocessor 80386 (1985), 80486 (1989)

The price/performance curve continued its steep climb with the 386 and later the 486—32-bit processors that brought real computing to the masses. The 386, which became the best-selling microprocessor in history, featured 275,000 transistors; the 486 had more than a million.

Features of 80386

(i) Introduced-October 17, 1985
(ii) Clock rates:
16 MHz with 5 to 6 MIPS
20 MHz with 6 to 7 MIPS, introduced February 16, 1987
25 MHz with 8.5 MIPS, introduced April 4, 1988
33 MHz with 11.4 MIPS (9.4 SPECint92 on Compaq/i 16K L2), introduced April 10, 1989
(iii) Bus Width-32 bit data, 32 bit address
(iv) Number of Transistors -275,000 at 1 μm
(v) Addressable memory-4 GB (4 GB)
(vi) Virtual memory-64 TB (64 TiB)
(vii) First x86 chip to handle 32-bit data sets
(viii) Reworked and expanded memory protection support including paged virtual memory and virtual 86 mode, features required by Linux,Windows 95 and OS/2 Warp
(ix) Used in Desktop computing

Features of 80486

(i) Introduced-April 10, 1989
(ii) Clock rates:
25 MHz with 20 MIPS
33 MHz with 27 MIPS
50 MHz with 41 MIPS
(iii) Bus Width-32 bits
(iv) Number of Transistors-1.2 million at 1 μm; the 50 MHz was at 0.8 μm
(v) Addressable memory-4 GB
(vi) Virtual memory-1 TB
(vii) Level 1 cache of 8 KB on chip

(viii) Math coprocessor on chip
(ix) 50X performance of the 8088
(x) Used in Desktop computing and servers
(xi) Family 4 model 3

5. 64-Bit Microprocessors

Pentium I (1993)

Adding systems-level characteristics to enormous raw compute power, the Pentium supports demanding I/O, graphics and communications-intensive applications with more than 3 million transistors.

Features

(i) Bus width-64 bits
(ii) System bus clock rate-60 or 66 MHz
(iii) Address bus-32 bits
(iv) Addressable Memory-4 GB
(v) Virtual Memory-64 TB
(vi) Superscalar architecture
(vii) Runs on 5 volts
(viii) Used in desktops
(ix) 16 KB of L1 cache
(x) 0.8 μm process technology
(xi) Introduced March 22, 1993
(xii) Number of transistors 3.1 million
(xiii) Socket 4 273 pin PGA processor package
(xiv) Package dimensions 2.16" × 2.16"
(xv) Family 5 model 1

Pentium Pro (1995)

The newest Pentium has dynamic instruction execution and other performance enhancing features such as a large L2 cache in the chip package, in addition to its more than 5.5 million transistors.

Features

(i) Introduced-November 1, 1995
(ii) Primarily used in server systems
(iii) Socket 8 processor package (387 pins) (Dual SPGA)
(iv) Number of transistors-5.5 million
(v) Family 6 model 1
(vi) 0.6 μm process technology
(vii) 16 KB L1 cache
(viii) 256 KB integrated L2 cache
(ix) 60 MHz system bus clock rate

www.IPUnation.com

(x) Variants
- 150 MHz
- 0.35 μm process technology, or 0.35 μm CPU with 0.6 μm L2 cache
- Number of transistors 5.5 million
- 512 KB or 256 KB integrated L2 cache
- 60 or 66 MHz system bus clock rate

Variants
- 166 MHz (66 MHz bus clock rate, 512 KB 0.35 μm cache) Introduced November 1, 1995
- 180 MHz (60 MHz bus clock rate, 256 KB 0.6 μm cache) Introduced November 1, 1995
- 200 MHz (66 MHz bus clock rate, 256 KB 0.6 μm cache) Introduced November 1, 1995
- 200 MHz (66 MHz bus clock rate, 512 KB 0.35 μm cache) Introduced November 1, 1995
- 200 MHz (66 MHz bus clock rate, 1 MB 0.35 μm cache) Introduced August 18, 1997

Pentium II (1997)

The 7.5 million-transistor Pentium II processor incorporates Intel MMXTM technology, which designed specifically to process video, audio and graphics data efficiently.

Features

(i) Introduced- May 7, 1997
(ii) Pentium Pro with MMX and improved 16-bit performance
(iii) 242-pin Slot 1 (SEC) processor package
(iv) Number of transistors-7.5 million
(v) 32 KB L1 cache
(vi) 512 KB ½ bandwidth external L2 cache
Klamath-0.35 μm process technology (233, 266, 300 MHz)
- 66 MHz system bus clock rate
- Family 6 model 3
- Variants
  233 MHz Introduced May 7, 1997
  266 MHz Introduced May 7, 1997
  300 MHz Introduced May 7, 1997

Pentium II Xeon (1998)

The Pentium II Xeon processors are designed to meet the performance requirements of mid-range and higher servers and workstations. Consistent with Intel's strategy to deliver unique processor products targeted for specific markets segments, the Pentium II Xeon processors feature technical innovations specifically designed for workstations and servers that utilize demanding business applications such as Internet services, corporate data warehousing, digital content creation, and electronic and mechanical design automation. Systems based on the processor can be configured to scale to four or eight processors and beyond

Celeron (PII Based, 1998)

Continuing Intel's strategy of developing processors for specific market segments, the Intel Celeron processor is designed for the value PC market segment. It provides consumers great performance at an exceptional value, and it delivers excellent performance for uses such as gaming and educational software

Features

(i) Covington-0.25 μm process technology
(ii) Introduced-April 15, 1998
(iii) 242-pin Slot 1 SEPP (Single Edge Processor Package)
(iv) Number of transistors-7.5 million
(v) 66 MHz system bus clock rate
(vi) 32 KB L1 cache
(vii) No L2 cache
(viii) Variants
266 MHz Introduced April 15, 1998
300 MHz Introduced June 9, 1998

Pentium III (1999)

The Pentium III processor features 70 new instructions. It was designed to significantly enhance Internet experiences, allowing users to do such things as browse through realistic online museums and stores and download high-quality video. The processor incorporates 9.5 million transistors, and was introduced using 0.25-micron technology.

Features

(i) Katmai-0.25 μm process technology.
(ii) Introduced-February 26, 1999.
(iii) Improved PII, *i.e.* P6-based core, now including Streaming SIMD Extensions (SSE).
(iv) Number of transistors -9.5 million.
(v) 512 KB ½ bandwidth L2 External cache.
(vi) 242-pin Slot 1 SECC2 (Single Edge Contact cartridge 2) processor package
(vii) System Bus clock rate 100 MHz, 133 MHz (B-models)
(viii) Family 6 model 7
(ix) Variants
450 MHz Introduced February 26, 1999
500 MHz Introduced February 26, 1999
550 MHz Introduced May 17, 1999
600 MHz Introduced August 2, 1999
533 MHz Introduced (133 MHz bus clock rate) September 27, 1999
600 MHz Introduced (133 MHz bus clock rate) September 27, 1999
(x) Coppermine-0.18 μm process technology
- Introduced-October 25, 1999
- Number of transistors -28.1 million
- 256 KB Advanced Transfer L2 Cache (Integrated)
- 242-pin Slot-1 SECC2 (Single Edge Contact cartridge 2) processor package, 370-pin FC-PGA (Flip-chip pin grid array) package
- System Bus clock rate 100 MHz (E-models), 133 MHz (EB models)
- Slot 1, Socket 370
- Family 6 model 8

www.IPUnaat.com

Pentium III Xeon (1999)

The Pentium III Xeon processor extends Intel's offerings to the workstation and server market segments providing additional performance for e-Commerce applications and advanced business computing. The processors incorporate the Pentium III processor's 70 SIMD instructions, which enhance multimedia and streaming video applications. The Pentium-III Xeon processor's advance cache technology speeds information from the system bus to the processor, significantly boosting performance. It is designed for systems with multiprocessor configurations.

Features

(i) Introduced-October 25, 1999
(ii) Number of transistors-9.5 million at 0.25 μm or 28 million at 0.18 μm)
(iii) L2 cache is 256 KB, 1 MB, or 2 MB Advanced Transfer Cache (Integrated)
(iv) Processor Package Style is Single Edge Contact Cartridge (S.E.C.C.2) or SC330
(v) System Bus clock rate 133 MHz (256 KB L2 cache) or 100 MHz (1-2 MB L2 cache)
(vi) System Bus Width-64 bit
(vii) Addressable memory-64 GB
(viii) Used in two-way servers and workstations (256 KB L2) or 4- and 8-way servers (1 - 2 MB L2)
(ix) Family 6 model 10
(x) Variants

500 MHz (0.25 μm process) Introduced March 17, 1999
550 MHz (0.25 μm process) Introduced August 23, 1999
600 MHz (0.18 μm process, 256 KB L2 cache) Introduced October 25, 1999
667 MHz (0.18 μm process, 256 KB L2 cache) Introduced October 25, 1999
733 MHz (0.18 μm process, 256 KB L2 cache) Introduced October 25, 1999

Pentium IV

Features

(i) 0.18 μm process technology (1.40 and 1.50 GHz)
(ii) Introduced November 20, 2000
(iii) L2 cache was 256 KB Advanced Transfer Cache (Integrated)
(iv) Processor Package Style was PGA423, PGA478
(v) System Bus clock rate-400 MHz
(vi) SSE2 SIMD Extensions
(vii) Number of Transistors-42 million
(viii) Used in desktops and entry-level workstations

Intel Core 2

Features

(i) Conroe-65 nm process technology
(ii) Desktop CPU (SMP support restricted to 2 CPUs)
(iii) Two cores on one die
(iv) Introduced-July 27, 2006
(v) SSSE3 SIMD instructions

(vi) Number of Transistors-291 Million
(vii) 64 KB of L1 cache per core (32+32 KB 8-way)
(viii) Intel VT, multiple OS support
(ix) TXT, enhanced security hardware extensions
(x) Execute Disable Bit
(xi) EIST (Enhanced Intel Speed Step Technology)
(xii) iAMT2 (Intel Active Management Technology), remotely manage computers
(xiii) LGA 775
(xiv) Variants

Core 2 Duo E6850-3.00 GHz (4 MB L2, 1333 MHz FSB)
Core 2 Duo X6800-2.93 GHz (4 MB L2, 1066 MHz FSB)
Core 2 Duo X6850-2.54 GHz (4 MB L2, 1066 MHz FSB)
Core 2 Duo E6750-2.67 GHz (4 MB L2, 1333 MHz FSB)
Core 2 Duo E6700-2.67 GHz (4 MB L2, 1066 MHz FSB)
Core 2 Duo E6600-2.40 GHz (4 MB L2, 1066 MHz FSB)
Core 2 Duo E6550-2.33 GHz (4 MB L2, 1333 MHz FSB)
Core 2 Duo E6420-2.13 GHz (4 MB L2, 1066 MHz FSB)
Core 2 Duo E6400-2.13 GHz (2 MB L2, 1066 MHz FSB)
Core 2 Duo E6320-1.86 GHz (4 MB L2, 1066 MHz FSB)
Core 2 Duo E6300-1.86 GHz (2 MB L2, 1066 MHz FSB)

www.IPUnation.com

Pentium Dual Core

Features

(i) Allendale - 65 nm process technology
(ii) Desktop CPU (SMP support restricted to 2 CPUs)
(iii) Two CPUs on one die
(iv) Introduced- January 21, 2007
(v) SSSE3 SIMD instructions
(vi) Number of Transistors 167 Million
(vii) TXT, enhanced security hardware extensions
(viii) Execute Disable Bit
(ix) EIST (Enhanced Intel Speed Step Technology)
(x) Variants

Intel Pentium E2220 - 2.40 GHz (1 MB L2, 800 MHz FSB)
Intel Pentium E2200 - 2.20 GHz (1 MB L2, 800 MHz FSB)
Intel Pentium E2180 - 2.00 GHz (1 MB L2, 800 MHz FSB)
Intel Pentium E2160 - 1.80 GHz (1 MB L2, 800 MHz FSB)
Intel Pentium E2140 - 1.60 GHz (1 MB L2, 800 MHz FSB)

(xi) Wolfdale 45 nm process technology

Intel Pentium E6500 - 2.93 GHz (2 MB L2,1066 MHz FSB)

Intel Pentium E6300 - 2.80 GHz (2 MB L2,1066 MHz FSB)
Intel Pentium E5400 - 2.70 GHz (2 MB L2, 800 MHz FSB)
Intel Pentium E5300 - 2.60 GHz (2 MB L2, 800 MHz FSB)
Intel Pentium E5200 - 2.50 GHz (2 MB L2, 800 MHz FSB)
Intel Pentium E2210 - 2.20 GHz (1 MB L2, 800 MHz FSB)

Two most popular series of microprocessors INTEL 8086, 80186, 80286, 80386, 80486, Pentium Motorola 68000, 68010, 68020, 68030, 68040.

4.3. INTERNAL STRUCTURE OF MICROPROCESSOR

The microprocessor has three major components, namely Registers, Arithmetic and Logic Unit (ALU) and Control Unit (CU) as shown in the Fig. 4.2. Registers are used to store the data temporarily during the execution of the instruction. There are two types of registers namely general purpose registers and special purpose registers.

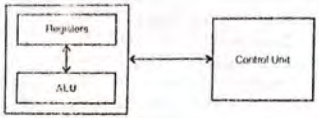


Fig. 4.2. Internal Structure of Microprocessor

Arithmetic and logic unit is responsible for the mathematical operation such as addition, subtraction etc. and logical operation such as AND, OR, etc.The control unit provides the various control and timing signal to all the operations.

www.pbookshop.com

4.4. SUPPORTING DEVICES OF MICROPROCESSOR

The various supporting devices of the microprocessor are shown in the Fig. 4.3 and illustrated below.

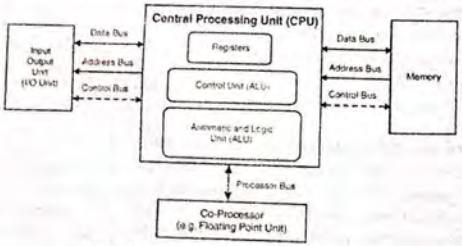


Fig. 4.3. Supporting Devices of Microprocessor

4.4.1. Clocks and Reset Circuits

Microprocessors use a fixed frequency clock to synchronize the operation of their internal logic. This clock signal is usually generated by a quartz-crystal controlled oscillator. The processor must be initialized to a known state when power is first applied or when the system "crashes." Most microprocessors require external circuits to detect the power-on condition or external resets.

4.4.2. Buffers

In larger systems the microprocessor must be connected to more chips than the microprocessor chip's electrical specifications allow. ICs called buffers, bus drivers or transceivers are used between the microprocessor and the other chips.

4.4.3. Latches

Some microprocessors reduce the number of pins required on the chip by using the same pin for two purposes, such as using the same pins first as part of the address bus and then as part of the data bus. An external chip (a latch) is required to temporarily hold the address bus value.

4.4.4. Address Decoders

Since microprocessors can usually address more memory than an individual memory chips contains, there has to be external logic to select the appropriate memory chip for a given address. This function is called address decoding.

4.4.5. Wait State Generators

The access-, hold-, and setup-times of memory and I/O devices often exceed the values provided by the microprocessor's read and write cycles. Circuits are used to force the processor to "wait" one or more clock cycles to extend the durations of read or write cycles.

4.4.6. Interrupt Controllers

I/O devices that require immediate attention can interrupt a program's normal execution by asserting an interrupt signal to the microprocessor. When a system has several sources of interrupts it is useful to be able to distinguish between them and to ensure that higher-priority interrupts are recognized first.

4.4.7. Timers and Counters

It's often useful for programs to be able to measure elapsed time so they can compute the time of day and measure intervals between events. Timers are counters driven from a clock and whose value can be read by the CPU. Timers are often used to generate periodic interrupts to allow an operating system to switch between tasks.

4.4.8. I/O Interfaces

I/O interfaces are used to read and write data to storage and I/O devices such as keyboards, printers, disk drives, etc. Serial interfaces transfer one bit at a time while parallel interfaces transfer several (typically 8) bits at a time. A typical serial interface is the "RS-232" serial interface. Typical parallel interfaces include the "Centronics" (printer), IDE, SCSI and GPIB interfaces.

4.4.9. DMA Controllers and Bus Arbitration

DMA (Direct Memory Access) allows peripherals to directly read or write memory or other peripherals on the system bus independently of the CPU. Special logic is required to ensure that only one device at time tries to use the bus.

4.4.10. Memory

Because the microprocessor alone cannot accommodate the large amount of memory required to store program instructions and data, such as the text in a word-processing program, transistors can be used as memory elements in combination with the microprocessor.

1. Random Access Memory: Integrated circuits, called random-access memory (RAM), which contain large numbers of transistors, is used in conjunction with the microprocessor to provide the needed memory.RAM is a volatile memory. There are different kinds of random-access memory. Static RAM (SRAM) holds information as long as power is turned on and is usually used as cache memory because it operates very quickly.

Another type of memory, dynamic RAM (DRAM), is slower than SRAM and must be periodically refreshed with electricity or the information it holds is lost. DRAM is more economical than SRAM and serves as the main memory element in most computers. Dynamic RAM stores data as electric charge in a capacitor. DRAM requires that every memory cell's content be "refreshed" periodically by recharging the capacitor. Circuits are required to ensure that the DRAM contents are periodically refreshed.

2. Read Only Memory: It is used to store the data permanently. It is a non volatile memory, *i.e.* the contents of the memory cannot be changed by the programmer. There are different types of ROMs such as masked ROM, PROM, EPROM, EEPROM and flash memory.

www.IPUnation.com

3. Cache Memory: Many modern microprocessors require faster access to memory than is possible with inexpensive memory devices. Fast auxiliary memories called cache memories are used to store the contents of frequently used memory locations and thus improve the overall performance of the system.

4.4.11. Buses

A bus is a group of related signals. Basically there are three types of buses, *i.e. data bus, address bus and control bus.* Most microprocessor systems include several buses. The signals appearing on the pins on the microprocessor chip are called the processor bus. Many microcomputers allow peripherals and memory to be placed on physically separate PC cards which plug into connectors on a "motherboard" or "backplane." The signals on these connectors are called the system bus. Examples of system buses include the ISA, VME, and PCI buses. Peripherals (such as modems or disk drives) are connected to the computer using different connectors. The signals on these connectors are called the peripheral bus. Some common peripheral buses include the RS-232 serial interface and the SCSI (Small Computer System Interface) bus. These buses are often further divided into smaller buses. For example, the pins on the microprocessor and system bus can be grouped into a data bus, an address bus, a control bus, and a utility bus. A large part of this course is devoted to the study of typical processor, system and peripheral buses.

4.5. LEVELS OF PROGRAMMING LANGUAGES

A programming language is a notation for writing programs, which are specifications of a computation or algorithm. There is only one programming language that any computer can actually understand and execute its own native binary machine code. This is the lowest possible level of language

in which it is possible to write a computer program. All other languages are said to be *high level* or *low level* according to how closely they can be said to resemble machine code.

In this context, a low-level language corresponds closely to machine code, so that a single low-level language instruction translates to a single machine-language instruction. A high-level language instruction typically translates into a series of machine-language instructions.

Low-level languages have the advantage that they can be written to take advantage of any peculiarities in the architecture of the central processing unit (CPU) which is the "brain" of any computer. Thus, a program written in a low-level language can be extremely efficient, making optimum use of both computer memory and processing time. However, to write a low-level program takes a substantial amount of time, as well as a clear understanding of the inner workings of the processor itself. Therefore, low-level programming is typically used only for very small programs, or for segments of code that are highly critical and must run as efficiently as possible.

High-level languages permit faster development of large programs. The final program which is executed by the computer is not efficient, but the savings in programmer time.

This is because the cost of writing a program is nearly constant for each line of code, regardless of the language. Thus, a high-level language where each line of code translates to 10 machine instructions costs only one tenth as much in program development as a low-level language where each line of code represents only a single machine instruction.

In addition to the distinction between high-level and low-level languages, there is a further distinction between *compiler languages* and *interpreter languages*.

4.5.1. Machine Language

The very lowest possible level at which one can program a computer is in its own native machine code, consisting of strings of 1's and 0's and stored as binary numbers. The main problems with using machine code directly are that it is very easy to make a mistake, and very hard to find it once you realize the mistake has been made.

4.5.2. Assembly Language

Assembly language is the symbolic representation of machine code, which also allows symbolic designation of memory locations. Thus, an instruction to add the contents of a memory location to an internal CPU register called the *accumulator* might be add a number instead of a string of binary digits (*bits*).No matter how close assembly language is to machine code, the computer still cannot understand it.

The assembly-language program must be translated into machine code by a separate program called an *assembler*. The assembler program recognizes the character strings that make up the symbolic names of the various machine operations, and substitutes the required machine code for each instruction. At the same time, it also calculates the required address in memory for each symbolic name of a memory location, and substitutes those addresses for the names. The final result is a machine-language program that can run on its own at any time; the assembler and the assembly language program are no longer needed.

To help distinguish between the "before" and "after" versions of the program, the original assembly language program is also known as the *source code*, while the final machine language program is designated the *object code*.

If an assembly-language program needs to be changed or corrected, it is necessary to make the changes to the source code and then re-assemble it to create a new object program. Assembly Language is a Symbolic Language.

4.5.3. Compiler Language

Compiler languages are the high-level equivalent of assembly language. Each instruction in the compiler language can correspond to many machine instructions. Once the program has been written, it is translated to the equivalent machine code by a program called a *compiler*. Once the program has been compiled, the resulting machine code is saved separately, and can be run on its own at any time.

As with assembly-language programs, updating or correcting a compiled program requires that the original (source) program be modified appropriately and then recompiled to form a new machine-language (object) program.

Typically, the compiled machine code is less efficient than the code produced when using assembly language. This means that it runs a bit more slowly and uses a bit more memory than the equivalent assembled program. To offset this drawback, however, we also have the fact that it takes much less time to develop a compiler-language program, so it can be ready to go sooner than the assembly-language program.

4.5.4. Interpreter Language

An interpreter language, like a compiler language, is considered to be high level. However, it operates in a totally different manner from a compiler language. Rather, the interpreter program resides in memory, and directly executes the high-level program without preliminary translation to machine code.

This uses an interpreter program to directly execute the user's program has both advantages and disadvantages. The primary advantage is that one can run the program to test its operation, make a few changes, and run it again directly. There is no need to recompile because no new machine code is ever produced. This can enormously speed up the development and testing process.

On the down side, this arrangement requires that both the interpreter and the user's program reside in memory at the same time. In addition, because the interpreter has to scan the user's program *one line at a time* and execute internal portions of itself in response; execution of an interpreted program is much slower than for a compiled program.

4.6. INTRODUCTION TO 8085

The *8085 is an 8-bit microprocessor* introduced by Intel in 1976.The 8085 was the follow-on processor to the very successful Intel 8080A processor. The 8085 got its name because it was Intel's first 5 volt microprocessor. The 8085 was 100% software compatible with the 8080A with increased systems performance. The initial 8085's were based on NMOS technology and the later "H" versions were based on HMOS technology. The 8085 used 6,500 transistors.

The *8085 incorporated all the features of the 8224 (clock generator)* and the 8228 (system controller) increasing the level of system integration. The 8085 along with and 8156 RAM and 8355/8755 ROM/PROM constituted a complete system. The 8085 used a multiplexed Data Bus and required the 825X-5 support chips. The address was split between the 8-bit address bus and 8-bit data bus. The on-chip address latch of 8155/8355/8755 memory chips allowed a direct interface with the 8085.

4.7. FEATURES OF 8085 MICROPROCESSOR

1. It is an 8 bit microprocessor.
2. It is manufactured with N-MOS technology.
3. It has 16-bit address bus and hence can address up to 2^{16} memory locations.

4. The first 8 lines of address bus and 8 lines of data bus are multiplexed.
5. Data bus is a group of 8 lines $D_0 - D_7$.
6. It supports external interrupt request.
7. A 16 bit program counter (PC)
8. A 16 bit stack pointer (SP)
9. Six 8-bit general purpose register arranged in pairs: BC, DE, HL.
10. It requires a signal +5V power supply and operates at 3.2 MHZ clock.
11. It is available in 40 pin DIP (Dual in line package).
12. 8085 is one address microprocessor.

4.8. INTERNAL ARCHITECTURE AND FUNCTIONAL BLOCK DIAGRAM OF 8085

The 8085 Architecture follows *the von Neumann architecture* shown in Fig. 4.4, with a 16-bit address bus, and an 8-bit data bus. The 8085 incorporated all the features of the *8224 (clock generator)* and the 8228 (system controller), increasing the level of system integration.

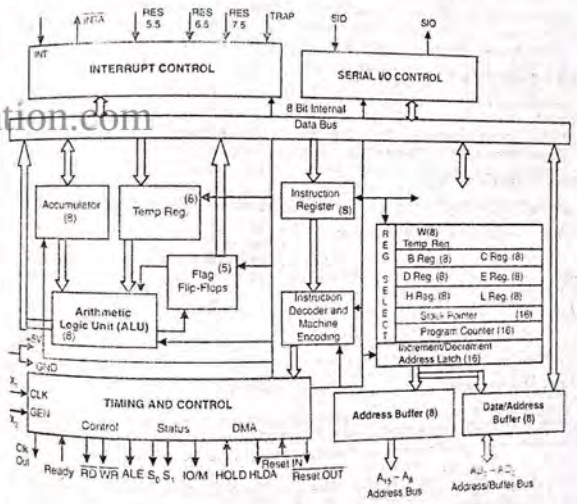


Fig. 4.4. Functional Block Diagram of 8085 microprocessor

4.8.1. Registers

The 8085 can access 2^{16} (= 65,536) individual 8-bit memory locations, or in other words, its address space is 64 kb. Unlike some other microprocessors of its era, it has a separate address space for up to 2^8 (=256) I/O ports.

It also has a built in register array which are usually labeled *A (Accumulator), B, C, D, E, H, and L*. Further special-purpose registers are the *16-bit Program Counter (PC), Stack Pointer (SP), and 8-bit flag register F.*

4.8.2. Accumulator

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

4.8.3. Register Pairs

(1) 8-bit B and 8-bit C registers can be used as one 16-bit BC register pair. When used as a pair, the C register contains low-order byte. Some instructions may use BC register as a data pointer.

(2) 8-bit D and 8-bit E registers can be used as one 16-bit DE register pair. When used as a pair the E register contains low-order byte. Some instructions may use DE register as a data pointer.

(3) 8-bit H and 8-bit L registers can be used as one 16-bit HL register pair. When used as a pair the L register contains low-order byte. HL register usually contains a data pointer used to reference memory addresses.

4.8.4. Stack Pointer

Stack pointer is a *16 bit* register. This register is always *incremented/decremented by 2*. It always holds the address of the top most entity in the stack. The stack pointer is also used as a memory pointer. It points to a memory location in R/W memory, called the stack. The beginning of the stack is defined by loading 16-bit address in the stack pointer.

4.8.5. Program Counter

Program counter is a 16-bit register. It holds *the address of the instruction which is to be executed next.*

4.8.6. Instruction Register/Decoder

Temporary storage for the *current instruction* of a program. Latest instruction sent here from memory prior to execution. Decoder then takes instruction and 'decodes' or interprets the instruction. Decoded instruction then passed to next stage.

4.8.7. Memory Address Register

It holds the address (received from PC) of next program instruction. Feeds the address bus with addresses of location of the program under execution.

4.8.8. Flag Register

The ALU includes *five flip-flops*, which are set or reset after an operation according to data conditions of the result in the accumulator and other registers. They are called *Zero (Z), Carry (CY), Sign (S), Parity (P), and Auxiliary Carry (AC) flags*; and their bit positions in the flag register are shown below.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z	U	AC	U	P	U	CY

The most commonly used flags are Zero, Carry, and Sign. The microprocessor uses these flags to test data conditions.

For example, after an addition of two numbers, if the sum in the accumulator is larger than eight bits, the flip-flop uses to indicate a carry, called the Carry flag (CY) – is set to one.

When an arithmetic operation results in zero, the flip-flop called the Zero (Z) flag is set to one.

4.8.9. 8085 System Bus

Typical system uses a number of busses, collection of wires, which transmit binary numbers, one bit per wire. A typical microprocessor communicates with memory and other devices (input and output) using three busses: *Address Bus, Data Bus and Control Bus.*

1. Address Bus: One wire for each bit, therefore 16 bits = 16 wires. Binary number carried alerts memory to 'open' the designated box. Data (binary) can then be put in or taken out. The Address Bus consists of 16 wires, therefore 16 bits. Its "width" is 16 bits. A 16 bit binary number allows 216 different numbers, or 32000 different numbers, *i.e.*, 0000000000000000 up to 1111111111111111. Because memory consists of boxes, each with a unique address, the size of the address bus determines the size of memory, which can be used. To communicate with memory the microprocessor sends an address on the address bus, e.g. 0000000000000011 (3 in decimal), to the memory. The memory selects box number 3 for reading or writing data. Address bus is unidirectional, *i.e.*, numbers only sent from microprocessor to memory, not other way.

2. Data Bus: Data bus carries 'data', in binary form, between microprocessor and other external units, such as memory. Typical size is 8 or 16 bits. Size determined by size of boxes in memory and microprocessor size helps determine performance of microprocessor. The Data bus typically consists of 8 wires. Data bus used to transmit "data", i.e. information, results of arithmetic, etc. between memory and the microprocessor. Bus is bi-directional. Size of the data bus determines what arithmetic can be done. If only 8 bits wide then largest number is 11111111 (255 in decimal). Therefore, larger numbers have to be broken down into chunks of 255. This slows microprocessor. Data bus also carries instructions from memory to the microprocessor. Size of the bus therefore limits the number of possible instructions to 256, each specified by a separate number.

3. Control Bus: Control Bus has various lines which have specific functions for coordinating and controlling microprocessor operations. Control whether memory is being 'written to' (data stored in memory) or 'read from' (data taken out of memory) 1 = Read, 0 = Write. May also include clock line(s) for timing/synchronizing, 'interrupts', 'reset', etc. generally microprocessor has 10 control lines. The Control Bus carries control signals partly unidirectional, partly bi-directional. Control signals are things like "read or write". This tells memory that we are reading from a location, specified on the address bus, or writing to a location specified. Modern day microprocessors, like 80386, 80486 have much larger busses. Typically 16 or 32 bit busses, which allow larger number of instructions, more memory location, and faster arithmetic.

4.9. PIN CONFIGURATION OF 8085

Microprocessor 8085 is available in 40 pin DIP package shown in Fig. 4.5. The 8085 has some following properties these are:

1. Single + 5V Supply
2. 4 Vectored Interrupts (One is Non Maskable)
3. Serial In/Serial out Port

4. Decimal, Binary, and Double Precision Arithmetic
5. Direct Addressing Capability to 64K bytes of memory

4.9.1. Pin Function Description

The functions of different pins of 8085 are explained in the following manner.

$A_8 - A_{15}$ (Pin-21 to 28, Type-Output, 3 State)

Address Bus: The *most significant 8 bits of the memory address* or the 8 bits of the I/O addresses, 3 stated during Hold and Halt modes.

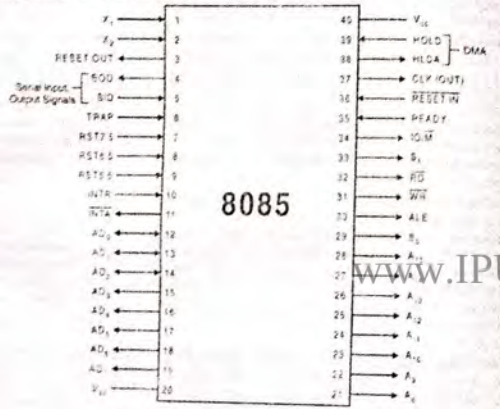


Fig. 4.5. Pin Diagram of 8085 Microprocessor

$AD_0 - AD_7$ (Pin-12 to 19, Type-Input/output, 3 state)

Multiplexed Address/Data Bus: Lower 8 bits of the memory address (or I/O address) appear on the bus during the first clock cycle of a machine state. It then becomes the data bus during the second and third clock cycles. 3 stated during Hold and Halt modes.

ALE (Pin-30, Type-Output)

Address Latch Enable: It occurs during the first clock cycle of a machine state and enables the address to get latched into the on chip latch of peripherals. The falling edge of ALE is set to guarantee setup and hold times for the address information. ALE can also be used to strobe the status information. ALE is never 3stated

S_0, S_1 (Pin-29 and 33, Type-Output)

Data Bus Status of the bus cycle.

S_1	S_0	Action
0	0	Halt
0	1	Write
1	0	Read
1	1	Fetch

S_1 can be used as an advanced R/W status.

$\overline{RD}$ (Pin 32, Type-Output, 3 State)

READ-indicates the selected memory or I/O device is to be read and that the Data Bus is available for the data transfer.

$\overline{WR}$ (Pin 31, Type-Output, 3 State)

WRITE-indicates the data on the Data Bus is to be written into the selected memory or I/O location. Data is set up at the trailing edge of WR. Three stated during Hold and Halt modes.

READY (Pin-35, Type-Input)

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

www.IPUnator.com

HOLD (Pin-39, Type-Input)

HOLD indicates that another Master is requesting the use of the Address and Data Buses. The CPU, upon receiving the Hold request, Will relinquish the use of buses as soon as the completion of the current machine cycle. Internal processing can continue. The processor can regain the buses only after the Hold is removed. When the Hold is acknowledged, the Address, Data, RD, WR, and IO/M lines are 3stated.

HLDA (Pin-38, Type-Output)

HOLD ACKNOWLEDGE indicates that the CPU has received the Hold request and that it will relinquish the buses in the next clock cycle. HLDA goes low after the Hold request is removed. The CPU takes the buses one half clock cycle after HLDA goes low.

INTR (Pin-10, Type-Input)

INTERRUPT REQUEST is used as a general purpose interrupt. It is sampled only during the next to the last clock cycle of the instruction. If it is active, the Program Counter (PC) will be inhibited from incrementing and an INTA will be issued. During this cycle a RESTART or CALL instruction can be inserted to jump to the interrupt service routine. The INTR is enabled and disabled by software. It is disabled by Reset and immediately after an interrupt is accepted.

$\overline{INTA}$ (Pin-11, Type-Output)

INTERRUPT ACKNOWLEDGE is used instead of (and has the same timing as) RD during the Instruction cycle after an INTR is accepted. It can be used to activate the 8259 Interrupt chip or some other interrupt port.

Restart Interrupts

These three inputs have the same timing as I NTR except they cause an internal RESTART to be automatically inserted.

RST 7.5 (Pin-7, Type-Input) Possess Highest Priority

RST 6.5 (Pin-8, Type-Input)

RST 5.5 (Pin-9, Type-Input) Possess Lowest Priority

The priority of these interrupts is ordered as shown above. These interrupts have a higher priority than the INTR.

TRAP (Pin-6, Type-Input)

Trap interrupt is a non-maskable restart interrupt. It is recognized at the same time as INTR. It is unaffected by any mask or Interrupt Enable. It has the highest priority of any interrupt.

$\overline{\text{RESET IN}}$ (Pin-36, Type-Input)

Reset sets the Program Counter to zero and resets the Interrupt Enable and HLDA flip-flops. None of the other flags or registers (except the instruction register) are affected The CPU is held in the reset condition as long as Reset is applied.

RESET OUT (Pin-3, Type-Output)

This pin is used as a system RESET. The signal is synchronized to the processor clock.

www.IPUnation.com

X_1, X_2 (Pin-1 and 2, Type-Input)

Crystal or RC network connections to set the internal clock generator X1 can also be an external clock input instead of a crystal. The input frequency is divided by 2 to give the internal operating frequency.

CLK (Pin-37, Type-Output)

Clock Output for use as a system clock when a crystal or R/ C network is used as an input to the CPU. The period of CLK is twice the X1, X2 input period.

IO/$\overline{\text{M}}$ (Pin-34, Type-Output)

IO/M indicates whether the Read/Write is to memory or I/O tristated during Hold and Halt modes.

SID (Pin-5, Type-Input)

Serial input data line, the data on this line is loaded into accumulator bit 7 whenever a RIM instruction is executed.

SOD (Pin-4, Type-Output)

Serial output data line. The output SOD is set or reset as specified by the SIM instruction.

V_{CC} (Pin-40)

+5 volt supply.

V_{SS} (Pin-20)

Ground Reference.

4.9.2. Signal Groups of Microprocessor 8085

Signals of 8085 are grouped in to six groups; these are shown in Fig. 4.6

1. Power supply and frequency (+5V, GND, 3.2M Hz Crystal Oscillator.)
2. High order address buses (A_{15}–A_8).
3. Multiplexed Low order address bus and data buses. (AD_7–AD_0)
4. Control signals (ALE, S1, S0,IO/M,RD,WR, RESETOUT, CLK OUT)
5. Interrupts (TRAP, RST7.5, 6.5, 5.5, INTR).
6. Externally Initiating signals (Ready, HOLD, RESETIN) and Serial i/o signals.

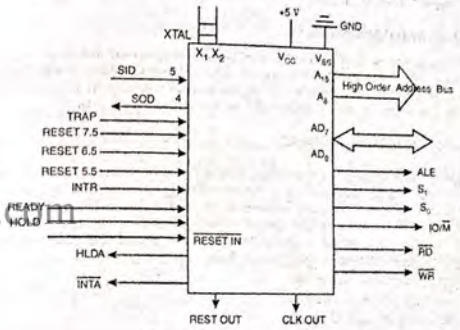


Fig. 4.6. Different Signals of 8085

4.10. ADDRESSING MODES OF 8085

The instructions MOV B, A or MVI A, 82H are to copy data from a source into a destination. In these instructions the source can be a register, an input port, or an 8-bit number (00H to FFH). Similarly, a destination can be a register or an output port. The sources and destination are operands. The various formats for specifying operands are called the addressing modes. For 8085, they are:

1. Immediate addressing.
2. Register addressing.
3. Direct addressing.
4. Indirect addressing.

4.10.1. Immediate Addressing

Data is present in the instruction. Load the immediate data to the destination provided.

Example: MVI R, data

4.10.2. Register Addressing

Data is provided through the registers.

Example: MOV Rd, Rs

4.10.3. Direct Addressing

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

Example: IN 00H or OUT 01H

4.10.4. Indirect Addressing

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

4.11. INSTRUCTION FORMAT

An **instruction** is a command to the microprocessor to perform a given task on a specified data. Each instruction has two parts: one is task to be performed, called the **operation code** (opcode), and the second is the data to be operated on, called the **operand.** The operand (or data) can be specified in various ways. It may include 8-bit (or 16-bit) data, an internal register, a memory location, or 8-bit (or 16-bit) address. In some instructions, the operand is implicit. Instruction word size of 8085 instruction set is classified into the following three groups according to word size:

1. One-word or 1-byte instructions
2. Two-word or 2-byte instructions
3. Three-word or 3-byte instructions

In the 8085, "byte" and "word" are synonymous because it is an 8-bit microprocessor. However, instructions are commonly referred to in terms of bytes rather than words.

4.11.1. One-Byte Instructions

A 1-byte instruction includes the opcode and operand in the same byte. Operand(s) are internal register and are coded into the instruction.

For example :

Task	Opcode	Operand	Binary code	Hex code
Copy the contents of the Acc in the register C	MOV	C,A	01001111	4F
Add the contents of register B to the contents of the Acc	ADD	B	10000000	80H
Invert each bit of the Acc.	CMA		00101111	2FH

These instructions are 1-byte instructions performing three different tasks. In the first instruction, both operand registers are specified. In the second instruction, the operand B is specified and the accumulator is assumed. Similarly, in the third instruction, the accumulator is assumed to be the implicit operand. These instructions are stored in 8-bit binary format in memory; each requires one memory location.

MOV rd, rs

rd <— rs copies contents of rs into rd.

Coded as 01 ddd sss where ddd is a code for one of the 7 general registers which is the destination of the data, sss is the code of the source register.

Example: MOV A,B

Coded as 01111000 = 78H = 170 octal (octal was used extensively in instruction design of such processors).

ADD r

A <— A + r

4.11.2. Two-Byte Instructions

In a two-byte instruction, the first byte specifies the operation code and the second byte specifies the operand. Source operand is a data byte immediately following the opcode. For example:

Task	Opcode	Operand	Binary Code	Hex Code	
Load an 8-bit data byte in the accumulator.	MVI	A, data Data code	00111110 Hex code of data	3E 2nd byte	1st byte

www.pdfbooksfree.com

Assume that the data is 32 H.The assembly language instruction can be expressed as following

Mnemonics	Hex Code
MVI A,32H	3E 32H

The instruction would require two memory locations to store in memory.

MVI r, data

r <— data

Example: MVI A, 30H coded as 3EH 30H as two contiguous bytes.

4.11.3. Three-Byte Instructions

In a three-byte instruction, the first byte specifies the opcode, and the following two bytes specify the 16-bit address. Note that the second byte is the low-order address and the third byte is the high-order address.

For example: opcode + data byte + data byte

Task	Opcode	Operand	Binary Code	Hex Code	
Transfer the program sequence to the memory location 2086H.	JMP	2086H	11000011 10000101 00100000	C3 85 20	1st byte 2nd byte 3rd byte

This instruction would require three memory locations to store in memory.

Three byte instructions – opcode + data byte + data byte

LXI rp, data16

rp is one of the pairs of registers BC, DE, HL used as 16-bit registers. The two data bytes are 16-bit data in L H order of significance.

rp <— data16

Example:

LXI H, 0520H coded as 21H 20H 50H in three bytes. This is also an immediate addressing.

LDA addr

A <— (addr) - Addr is a 16-bit address in L H order. Example: LDA 2134H coded as 3AH 34H 21H. This is also an example of direct addressing.

4.12. INSTRUCTION SET OF 8085

Instruction set of 8085 can be divided into following categories

1. Data transfer instructions-MOV, LDA, MVI etc.
2. Arithmetic operation instructions-Add, subtract, increment and decrement.
3. Logical operation instructions-AND, OR, XOR and rotate.
4. Control transfer instructions-conditional, unconditional, call subroutine, return from subroutine and restarts.
5. Input/output operation instructions, Machine control instructions.
6. Other-setting/clearing flag bits, enabling/disabling interrupts, stack operations, etc.

www.IPUnaionline.com

4.12.1. Data Transfer Instructions

The data transfer instructions move data between registers or between memory and registers.

MOV (Copy data from source to destination)

Syntax: MOV Rd, Rs

This instruction copies the contents of the source M, Rs register into the destination register. The contents of Rd, M the source register are not altered. If one of the operands is a memory location, its location is specified by the contents of the HL registers.

Example: MOV B, C or MOV B, M

MVI (Move Immediate 8-bit data)

Syntax: MVI Rd, data

The 8-bit data is transferred to the destination register or M, data memory. If the operand is a memory location, its location is specified by the contents of the HL registers.

Example: MVI B, 57H or MVI M, 57H

LDA (Load Accumulator Direct from Memory)

Syntax: LDA 16-bit address

The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator. The contents of the source are not altered.

Example: LDA 2034H

LHLD (Load H&L Registers Directly from Memory)

Syntax: LHLD 16-bit address

The instruction copies the contents of the memory location pointed out by the 16-bit address into register L and copies the contents of the next memory location into register H. The contents of source memory locations are not altered.

Example: LHLD 2040H

LXI (Load Register Pair with Immediate data)

Syntax: LXI Reg. pair, 16-bit data

The instruction loads 16-bit data in the register pair designated in the operand.

Example: LXI H, 2034H or LXI H, XYZ

LDAX (Load Accumulator from Address in Register Pair)

Syntax: LDAX B/D Reg. pair

The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory location into the accumulator. The contents of either the register pair or the memory location are not altered.

Example: LDAX B

STA (Store Accumulator Direct in Memory)

Syntax: STA 16-bit address

The contents of the accumulator are copied into the memory location specified by the operand. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address.

Example: STA 4350H

STAX (Store Accumulator in Address in Register Pair)

Syntax: STAX Reg. pair

The contents of the accumulator are copied into the memory location specified by the contents of the operand (register pair). The contents of the accumulator are not altered.

Example: STAX B

SHLD (Store H and L Registers Directly in Memory)

Syntax: SHLD 16-bit address

The contents of register L are stored into the memory location specified by the 16-bit address in the operand and the contents of H register are stored into the next memory location by incrementing the operand. The contents of registers HL are not altered. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address.

Example: SHLD 2470H

XCHG (Exchange H&L with D&E)

Syntax: XCHG none

The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E.

Example: XCHG

XTHL (Exchange Top of Stack with H&L)

Syntax: XTHL none

The contents of the L register are exchanged with the stack location pointed out by the contents of the stack pointer register. The contents of the H register are exchanged with the next stack location (SP+1); however, the contents of the stack pointer register are not altered.

Example: XTHL

SPHL (Move content of H&L to Stack Pointer)

Syntax: SPHL none

The instruction loads the contents of the H and L registers into the stack pointer register, the contents of the H register provide the high-order address and the contents of the L register provide the low-order address. The contents of the H and L registers are not altered.

Example: SPHL

4.12.2. Arithmetic Instructions

The arithmetic instructions are used to perform various arithmetic tasks such as addition, subtraction, division, multiplication, increment, or decrement etc.

ADD (Add Register or Memory to Accumulator)

Syntax: ADD R

The contents of the operand (register or memory) are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition.

Example: ADD B or ADD M

ADI (Add Immediate Data to Accumulator)

Syntax: ADI 8-bit data

The 8-bit data (operand) is added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition.

Example: ADI 45H

ADC (Add Register to Accumulator with Carry)

Syntax: ADC R

The contents of the operand (register or memory) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition.

Example: ADC B or ADC M

ACI (Add Immediate Data to Accumulator with Carry)

Syntax: ACI 8-bit data

The 8-bit data (operand) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition.

Example: ACI 45H

DAD (Double Register Add; Add Content of Register Pair to H&L Register Pair)

Syntax: DAD Reg. pair

The 16-bit contents of the specified register pair are added to the contents of the HL register and the sum is stored in the HL register. The contents of the source register pair are not altered. If the result is larger than 16 bits, the CY flag is set. No other flags are affected.

Example: DAD H

SUB (Subtract Register or Memory From Accumulator)

Syntax: SUB R

The contents of the operand (register or memory) are subtracted from the contents of the accumulator, and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction.

Example: SUB B or SUB M

SUI (Subtract Immediate Data from Accumulator)

Syntax: SUI 8-bit data

The 8-bit data (operand) is subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtraction.

Example: SUI 45H

SBB (Subtract Source From Accumulator with Borrow)

Syntax: SBB R

The contents of the operand (register or memory) and the Borrow flag are subtracted from the contents of the accumulator and the result is placed in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction.

Example: SBB B or SBB M

SBI (Subtract Immediate Data from Accumulator with Borrow)

Syntax: SBI 8-bit data

The 8-bit data (operand) and the Borrow flag are subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtraction.

Example: SBI 45H

INR (Increment Register or Memory by One)

Syntax: INR R

The contents of the designated register or memory are incremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers.

Example: INR B or INR M

DCR (Decrement Register or Memory by One)

Syntax: DCR R

The contents of the designated register or memory are M decremented by 1 and the result is store the same place. If the operand is a memory location, its location is specified by the contents of the HL regis

Example: DCR B or DCR M

INX (Increment Register Pair by One)

Syntax: INX R

The contents of the designated register pair are incremented by 1 and the result is stored in the same pl

Example: INX H

DCX (Decrement Register Pair by One)

Syntax: DCX R

The contents of the designated register pair are decremented by 1 and the result is stored in the same pl

Example: DCX H

DAA (Decimal Adjust After Addition)

Syntax: DAA none

The contents of the accumulator are changed from a binary value to two 4-bit binary co decimal (BCD) digits. This is the only instruction that uses the auxiliary flag to perform the binary to B conversion, and the conversion procedure is described below. S, Z, AC, P, CY flags are altered to refle results of the operation. If the value of the low-order 4-bits in the accumulator is greater than 9 or if AC fla set, the instruction adds 6 to the low-order four bits. If the value of the high-order 4-bits in the accumula greater than 9 or if the Carry flag is set, the instruction adds 6 to the high-order four bits.

Example: DAA

4.12.3. Logical Instructions

This group of instructions performs logical (Boolean) operations on data in registers and mem and no condition flags. The logical operations are AND, OR and Exclusive OR etc.

ANA (Logical AND of Register or Memory Data With Accumulator)

Syntax: ANA R

The contents of the accumulator are logically ANDed with the contents of the operand (regist memory), and the result is placed in the accumulator. If the operand is a memory location, its addre specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. C reset. AC is set.

Example: ANA B or ANA M

ANI (Logical AND of Immediate Data With Accumulator)

Syntax: ANI 8-bit data

The contents of the accumulator are logically ANDed with the 8-bit data (operand) and the resul placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is

Example: ANI 86H

ORA (Logical OR of Register or Memory Data with Accumulator)

Syntax: ORA R

The contents of the accumulator are logically ORed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset.

Example: ORA B or ORA M

ORI (Logical OR of immediate data with Accumulator)

Syntax: ORI 8-bit data

The contents of the accumulator are logically ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset.

Example: ORI 86H

XRA (Logical XOR of the register or memory data with Accumulator)

Syntax: XRA R

The contents of the accumulator are Exclusive ORed with M the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset.

Example: XRA B or XRA M

XRI (Exclusive OR of Immediate Data with Accumulator)

Syntax: XRI 8-bit data

The contents of the accumulator are Exclusive ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset.

Example: XRI 86H

CMP (Compare Register or Memory with Accumulator)

Syntax: CMP R

The contents of the operand (register or memory) are M compared with the contents of the accumulator. Both contents are preserved. The result of the comparison is shown by setting the flags of the PSW as follows:

If (A) < (reg/mem) => carry flag is set

If (A) = (reg/mem) => zero flag is set

If (A) > (reg/mem) => carry and zero flags are reset

Example: CMP B or CMP M

CPI (Compare Immediate Data with accumulator)

Syntax: CPI 8-bit data

The second byte (8-bit data) is compared with the contents of the accumulator. The values being compared remain unchanged. The result of the comparison is shown by setting the flags of the PSW as follows:

Example 4.5. *Assembly language program to subtract two 16 bit numbers.*

Solution. Program:

```
LHLD 8100
XCHG
LHLD 8102
MOV A, E
SUB L
STA 8104
JNC LOOP
DCR D
LOOP MOV A, D
SUB H
STA 8105
RST5
```

Example 4.6. *Assembly language program to multiply two 8 bit numbers.*

Solution. Program:

```
LXI H, 8A00
MOV B, M
XRA A
MOV C, A
INX H
AHEAD ADD M
JNC LOOP-1
INR C
LOOP-1 DCR C
JNZ AHEAD
INX H
MOV M, A
INX H
MOV M, C
RST5
```

□□□

Basic Computer Organization and Design

5

5.1. INSTRUCTION CODE

Basically an instruction in the computer field is a binary code that contains a sequence of microoperations for the computer and an instruction code is a group of bits that instruct the computer to perform a specific operation. It is usually divided into different parts, each having its own particular interpretation. Instruction codes together with the data are stored in memory. The computer reads each instruction from the memory and places it in a control register. The control then interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of microoperations. Every computer has its own instruction set and the ability to store and execute instructions, the store program concept, is the most important property of a general purpose computer.

www.IPhadOnly.com

The most important and the most basic part of the instruction code is its operational part whereas the operation code of an instruction is a group of bits that define following operations such as:

- Add
- Subtract
- Multiply
- Shift
- Compliment

The total number of bits required for the operation code of an instruction depends upon the total number of operations available in the computer. The operation code must consist of at least n bits for a given 2^n distinct operations. Considering an example of a computer with 64 distinct operations, in which one of them is and ADD operation. The operation code must consists of six bits, with a bit configuration 110010 assigned to the ADD operation. When this operation code is decoded in the control unit, the computer issues control signals to read an operand from memory and add the operand to a processor register. The operation must be performed on some data stored in processor register or in memory.

An instruction code must therefore specify not only the operation but also the registers or memory word where the result is to be stored. Processor registers can be specified by assigning to the instruction another binary code of k bits that specifies one of 2^k registers. There are many variations for arranging the binary code of instructions, and each computer has its own particular instruction code format. Instruction code formats are conceived by computer designers who specify the architecture of the computer.

5.1.1. Store Program Organization

A computer can be organized by having a processor register and an instruction code format with two parts. The first part of the format contains the operations, and the second part contains the address. The memory address tells the control, where to find an operand in memory. The operand is read from memory and is used as the data to be operated on together with the data stored in the processor register. In this organization the instructions are stored in one part of memory and data in another.

For a memory unit having 1024 words, we need 10 bits to specify an address and for 4096 words we need 12 bits to specify an address because $2^{12} = 4096$.

If we store each instruction code in one 16-bit memory word, we have available four bits for the opcode to specify one out of 16 possible operations, and 12 bits to specify the address of an operand. The control reads a 16-bit instruction form the portion of memory. It used the 12-bit address part of the instruction from the program portion of memory. Now it used the 12-bit address part of the instruction to read a 16-bit operand from the data portion of memory, and after this it will execute the operation specified by the opcode.

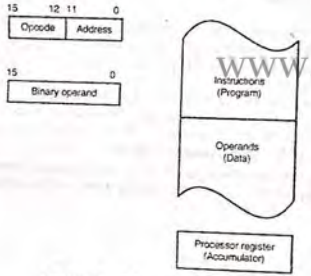


Fig. 5.1. Stored Program Organization

Many computers have a single-processor register they usually name it accumulator with the label name AC, and now the operation is performed with the memory operand and the content of AC. Some time an operation in an instruction code does not need an operand from memory, so in that case, the rest of the bits in the instruction can be used for other purposes.

5.1.2. Direct Address and Indirect Address

Sometime an instruction code can be used as an actual operand, not as an address. In that case when second part of an instruction code specifies an operand, the instruction is said to have an immediate

www.IPUnation.com

operand and when the second part specifies the address of an operand the instruction is said to have a Direct address. When the bits in the second part of the instruction designate an address of a memory word in which the address of the operand is found that is called as Indirect address. One bit of the instruction code can be used to distinguish between a direct and an indirect address. Now, let us consider an instruction format shown in Fig. 5.2 (*a*).

It consists of a 3-bit operation code, a 12-bit address, and an indirect address mode bit designated by I. The mode bit is 0 for a direct address and 1 for an indirect address. A direct address instruction is shown in Fig 5.2 (*b*). It is placed in address 32 in memory. The I bit is 0, so the instruction is recognized as a direct address instruction. The opcode specifies an ADD instruction, and the address part is the binary equivalent of 557. The control finds the operand in memory at address part is the binary equivalent of AC. The instruction in address 45 shown in Fig. 5.2 (*c*) has a mode bit I = 1. Therefore, it is recognized as an indirect address instruction. The address part is the binary equivalent of 400. The control goes to address 400 to find the address of the operand. The address of the operand in this case is 1750. The operand found in address 1750 is then added to the content of AC. The indirect address instruction needs two references to memory to fatch an operand. The first reference is needed to read the address of the operand ; the second is for the operand itself. We define the effective address to be the address of the operand in a computation-type instruction or the target address in a branch-type instruction. Thus the effective address in the instruction of Fig. 5.2 (*b*) is 557 and in the instruction of Fig. 5.2 (*c*) is 1750.

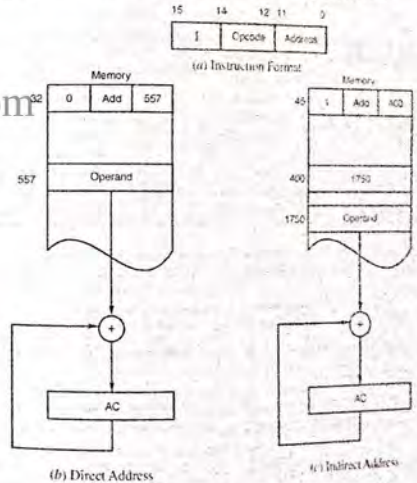


Fig. 5.2. Direct and Indirect Addresses.

The memory word that holds the address of the operand in an indirect address instruction is used as a pointer to an array of data and the pointer could be placed in a processor register instead of memory as done in commercial computers.

5.2. COMPUTER REGISTERS

Computer registers are the part of control unit that are used for storing the instruction code after it is read from memory. The computer needs processor registers for manipulating data and a register for holding a memory address. The memory unit has a capacity of 4096 words and each word contains 16 bits.

Twelve bits of an instruction word are needed to specify the address of an operand. This leaves three bits for the operation part of the instruction and a bit to specify a direct or indirect address.

- The data register (DR) holds the operand read from memory.
- The accumulator (AC) register is a general purpose processing register.
- The instruction read for memory is placed in the Instruction register (IR).
- The temporary register (TR) is used for holding temporary data during the processing.

Here is the list of registers for the basic computer.

Table 5.1

Register symbol	*Number of bits*	*Register name*	*Function*
DR	16	Data register	Holds memory operand
AR	12	Address register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register	Holds temporary data
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character

www.HPUnation.com

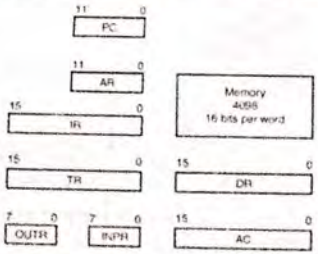


Fig. 5.3. Computer Registers

The memory address register (AR) and program counter (PC) has 12 bits, in which 12 bits of the address register is the width of a memory address and the 12 bits of the PC holds the address of the next instruction to be read from memory after the current instruction is executed.

Also the two register are used for input and output. The input register (INPR) receives an 8-bit character from an input device. The output register (OUTR) holds an 8-bit character for an output device. The basic computer has eight registers, a memory unit and a control unit.

5.3. COMPUTER INSTRUCTIONS

The basic computer contains three formats of instruction codes, and each format contains 16 bits. The opcode part of the instruction contains three bits and the meaning of the rest 13-bits depends upon the opcode for which the instruction format has to be generated. There are different types of instructions such as memory-reference, register-reference, I/O instruction.

1. A memory-reference instruction uses 12 bits to specify an address and one bit to specify the addressing mode I. I is equal to 0 for direct address and to 1 for indirect address.
2. The register-reference instructions are recognized by the operation code 111 with a 0 in the left most bit *i.e.*, bit 15 of the instruction. A register-reference instruction specifies an operation on or a test of the AC register. An operand from memory is not needed therefore the other 12 bits are used to specify the operation or test to be executed.
3. Similarly, an input-output instruction does not need a reference to memory and is recognized by the operation code 111 with a 1 in the left-most bit of the instruction. The remaining 12 bits are used to specify the type of input-output operation or test performed.

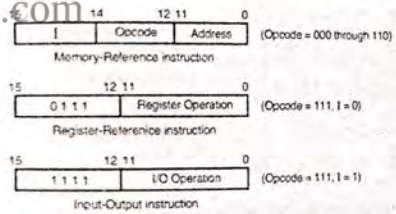


Fig. 5.4. Computer Instructions Format

A computer recognizes the type of instruction through the last four-bits positions *i.e.*, 12 through 15 of the instruction.

1. If the bits in positions 12 to 14 is not equal to 111, the instruction is said to be memory reference type and the bit at position 15 will be taken as addressing modes I.
2. If the bits in positions 12 to 14 are 111 then the control will inspect the bit position of 15, if this bit is 0, the instruction is said to be register reference type.
3. If the bits in the positions 12 to 14 are 111 and the bit of position 15 is 1 the instruction an input-output type.

The bit in position 15 of the instruction code is designated by the symbol I and is not used as a mode bit when the operation code is equal to 111. The instructions for the computer are listed in Table 5.2.

Table 5.2

Symbol	*Description*
AND	AND memory word to AC
ADD	Add memory word to AC
LDA	Load memory word to AC
STA	Store content of AC in memory
BUN	Branch unconditionally
BSA	Branch and save return address
ISZ	Increment and skip if zero
CLA	Clear AC
CLE	Clear E
CMA	Complement AC
CME	Complement E
CIR	Circulate right AC and E
CIL	Circulate left AC and E
INC	Increment AC
SPA	Skip next instruction if AC positive
SNA	Skip next instruction if AC negative
SZA	Skip next instruction if AC zero
SZE	Skip next instruction if E is 0
HLT	Halt computer
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

A computer should have a set of instructions so that the user can construct machine language programs to evaluate any function that is known to be computable. The set of instructions are said to be complete if the computer includes a sufficient number of instructions in each of the following categories :

1. Arithmetic, logical and shift instructions.
2. Instructions for moving information to and from memory and processor registers.
3. Program control instructions together with instructions that check status conditions.
4. Input and output instructions

The control functions and microoperations for the whole computer are given in Table 5.3. The register transfer statements in this table shows in a concise form the internal organization of the basic computer. Register transfer statements also provide the information required for the design of the logic circuits of the computer. Boolean functions for the gates is determined by the control functions and

conditional control statements listed in the table. The microoperations given in the table give the type of control inputs required for the registers and memory. A register transfer language is useful for specifying the logic circuits needed for its design. It is useful for describing the internal organization of a digital system.

Table 5.3. Control Functions and Microoperations for the Basic Computer

Fetch	$R'T_0$:	$AR \leftarrow PC$
	$R'T_1$:	$IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$
Decode	$R'T_2$:	$D_0, ..., D_7 \leftarrow$ Decode IR (12-14),
		$AR \leftarrow IR(0\text{-}11)$, $I \leftarrow IR(15)$
Indirect	D'_7IT_3:	$AR \leftarrow M[AR]$
Interrupt:		
	$T'_0T'_1T'_2$ (IEN) (FGI + FGO):	$R \leftarrow 1$
	RT_0:	$AR \leftarrow 0$, $TR \leftarrow PC$
	RT_1:	$M[AR] \leftarrow TR$, $PC \leftarrow 0$
	RT_2:	$PC \leftarrow PC + 1$, $IEN \leftarrow 0$, $R \leftarrow 0$, $SC \leftarrow 0$
Memory-reference :		
AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR$, $E \leftarrow Cout$, $SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR$, $SC \leftarrow 0$
STA	D_3T_4:	$M[AR] \leftarrow AC$, $SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR$, $SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC$, $AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR$, $SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR$, if (DR = 0) then ($PC \leftarrow PC + 1$), $SC \leftarrow 0$
Register-reference :		
		$D_7I'T_3 = r$ (common to all register-reference instructions)
		$IR(i) = B_i$ (i = 0, 1, 2, ..., 11)
	r:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow \overline{AC}$
CME	rB_8:	$E \leftarrow \overline{E}$
CIR	rB_7:	$AC \leftarrow$ shr AC, $AC(15) \leftarrow E$, $E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow$ shl AC, $AC(0) \leftarrow E$, $E \leftarrow AC(15)$

www.IPUnation.com

INC	rB_7:	AC ← AC + 1
SPA	rB_8:	If (AC(15) = 0) then (PC ← PC + 1)
SNA	rB_9:	If (AC(15) = 1) then (PC ← PC + 1)
SZA	rB_{10}:	If (AC = 0) then PC ← PC + 1)
SZE	rB_{11}:	If (E = 0) then (PC ← PC + 1)
HLT	rB_0:	S ← 0

Input-output:

$D_7IT_3 = p$ (common to all input-output instructions)

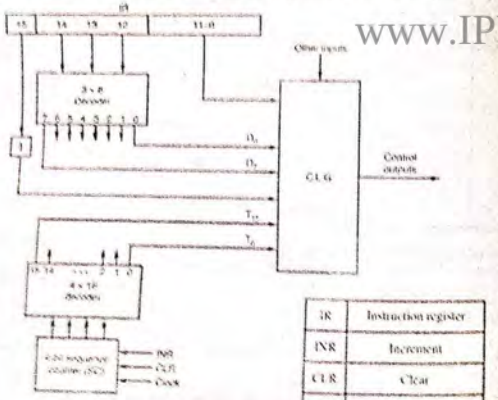
$IR(i) = B_i$ (i = 6, 7, 8, 9, 10, 11)

	p:	SC ← 0
INP	pB_{11}:	AC(0–7) ← INPR, FGI ← 0
OUT	pB_{10}:	OUTR ← AC (0–7), FGO ← 0
SKI	pB_9:	IF (FGI = 1) then (PC ← PC + 1)
SKO	pB_8:	If (FGO = 1) then (PC ← PC + 1)
ION	pB_7:	IEN ← 1
IOF	pB_6:	IEN ← 0

5.4. TIMING AND CONTROL

The timing of all registers in a basic computer system is controlled by a master clock generator. In the computer system the clock pulses are applied to all flip-flops and registers in the system, including the

www.IPUnotes.com



IR	Instruction register
INR	Increment
CLR	Clear
CLG	Control logic gates

Fig. 5.5. Control Unit of Basic Computer

flip-flops in the control unit. The share of a register is changed if the register is enabled by a control signal. Basically the control signals are generated in the control unit and provides the control inputs for the multiplexers in the common bus, processor register and microoperations for the accumulator.

Hardwired control and microprogrammed control are two major types of control organization. The block diagram of the control unit is shown in Fig. 5.5.

The block diagram of the control unit is shown in Fig. 5.5. It consists of two decoders, a sequence counter, and a number of control logic gates. An instruction read from memory is placed in the instruction register (IR). The instruction register is shown again in Fig. 5.5 where it is divided into three parts : the I bit, the operation code, and bits 0 through 11. The operation code in bits 12 through 14 are decoded with a 3 × 8 decoder. The eight outputs of the decoder are designated by the symbols D_0 through D_7. The subscripted decimal number is equivalent to the binary value of the corresponding operation code. Bit 15 of the instruction is transferred to a flip-flop designated by the symbol I. Bits 0 through 11 are applied to the control logic gates. The 4-bit sequence counter can count in binary from 0 through 15. The outputs of the counter are decoded into 16 timing signals T_0 through T_{15}.

The sequence counter SC can be incremented or cleared synchronously. Most of the time, the counter is incremented to provide the sequence of timing signals out of the 4 × 16 decoder. Once in a while, the counter is cleared to 0, causing the next active timing signal to be

For example:

Consider the case where SC is incremented to provide timing signals T_0, T_1, T_2, T_3 and T_4 in sequence. At time T_4, SC is cleared to 0 if decoder output D_3 is active. This is expressed symbolically by the statement:

$$D_3T_4 : SC \leftarrow 0$$

The timing diagram shows the time relationship of the control signals.

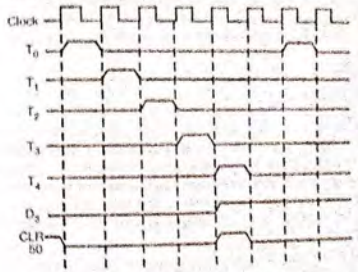


Fig. 5.6. Timing diagram for SC ← 0

5.5. INSTRUCTION CYCLE

A program comprises of many instructions arranged in a sequence. When a program is executed in a computer, it is executed by going through a cycle for each instruction and each instruction cycle in turn is divided into a sequence of sub cycles or phases. The different phases of an instruction cycle are

(a) Fetch

(b) Decode

(c) Read

(d) Execute

In the fetch phase, the instruction is fetched from memory. In the decode phase, the instruction is decoded and in the Read phase the effective address is read form memory. And in the final phase the instruction gets executed.

1. Fetch Phase: In the fetch phase the sequence counter (SC) is initialized to 0. The program counter (PC) contains the address the first instruction of a program under execution. The address of first instruction from PC is loaded into the address register (AR) during first clock cycle (T_0). Then instruction from memory location given by address register (AR) is loaded into the instruction register (IR) and the program counter is incremented to the address of next instruction register (IR) and the program counter is incremented to the address of next instruction in second clock cycle (T_1). These microoperations using register transfer language is shown as ahead.

$T_0 : AR \leftarrow PC$

$T_1 : IR \leftarrow M[AR], PC \leftarrow PC + 1$

2. Decode Phase : In the decode phase all the bits of the instruction under execution are stored in IR and are analyzed and decoded as shown ahead in third clock cycle (T_2).

$T_2\ D_0, D_1, D_2, D_7 \leftarrow$ Decode IR (12-14), $I \leftarrow IR(15)$, $AR \leftarrow IR(0 - 11)$.

3. Decision phase/Read Phase : First of all the [IR (12-14)] bits of instruction are decoded.

(a) If decoder output D_7 (111) is 1, that means instruction must be register reference or input output reference. Then the last bit os instruction register IR (15) stored in flip-flop I is decoded. If IR (15) = 0, that means the instruction is register reference instruction. If IR (15) = 1, that means the instruction is input-output reference instruction. Then first twelve bit of instruction are used for register or I/O reference instruction code.

$T_2 : D_7 \leftarrow$ Decode IR (12-14), $0 \leftarrow IR$ (15) (Register reference).

$T_2 : D_7 \leftarrow$ Decode IR (12-14); $1 \leftarrow IR$ (15) (I/O reference).

(b) If decoder output D_7 (111) is not 1, that means IR (12-14) are decoded in between D_0 to D_6. That implies the instruction is memory reference instruction. Then last bit of instruction register IR (15) stored in flip-flop I is needed. If IR (15) that means memory reference instruction is direct address instruction. If IR(15) = 1, that means memory reference instruction is indirect address instruction. The first twelve bits of instruction are used as address of memory location.

$T_2 : D_0, D_1, D_2, \ldots D_6 \leftarrow$ Decode IR (12-14), $0 \leftarrow IR(15)$

(Direct Address Memory Reference Instruction)

$T_2 : D_0, D_1, D_2, \ldots D_6 \leftarrow$ Decode IR (12-14), $1 \leftarrow IR$ (15)

(Indirect Address Memory Reference Instruction)

4. Execution Phase : Then the instruction is executed as memory or register or input output reference instruction in fourth clock cycle (T_3).

$D_7' I' T_3$: Execute direct address memory reference instruction.

$D_7' I T_3$: Calculate the effective address ($AR \leftarrow MR$ [AR]) and then execute indirect address memory reference instruction.

$D_7' I' T_3$: Execute a register reference instruction.

$D_7' I T_3$: Execute an input output reference instruction.

5.5.1. Flow Chart of Instruction Cycle

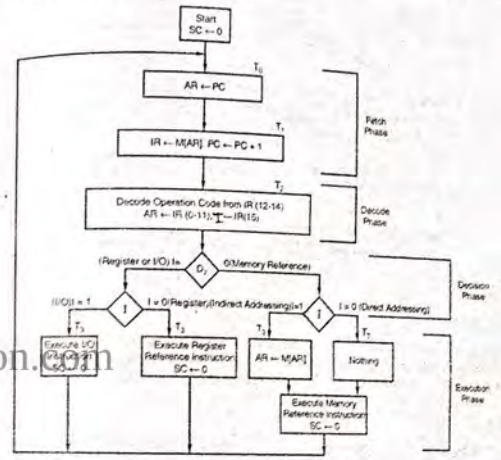


Fig. 5.7. Flow chart of Instruction Cycle

5.6. INPUT, OUTPUT AND INTERRUPTS

To serve usefully a computer should communicate with the external environment. Some input device should provide instructions and data to be stored in memory. Computational results must be showed to the user through some output devices. Many types of input and output devices are found in commercial computers.

An alphanumeric code of eight bits is required for each quantity of information. The serial information from the keyboard is shifted into the input register INPR. The output register OUTR stores the serial information for the printer. These two registers communicate with a communication interface serially and with the accumulator in parallel. The I/O configuration is shown in following Fig. 5.8.

FGI is a 1-bit input flag. It is a control flip-flop. We set the FGI to 1 when new information arrives in the input device and FGI is cleared when the information is taken by the computer. To synchronize the timing rate difference between the input device and the computer, we need flag. The information is transferred as follows. Initially, we clear the input flag FGI. An 8-bit alphanumeric code is shifted into INPR when we hit a keyboard, and the input flag FGI is set to 1. As long as the flag is set, the information

in INPR cannot be changed by striking another key. 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. Once the flag is cleared new information can be shifted into INPR by striking another key.

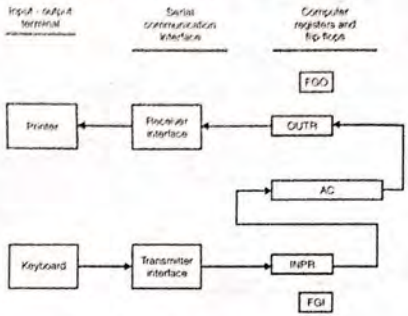


Fig. 5.8. Input-Output Configuration

The output register OUTR works similar to above discussion but the direction of information flow is just the opposite. At the very beginning, the output flag FGO is set. Now if the flag bit is equal to 1, the information from AC is transferred in parallel to OUTR and FGO is clears to 0.

www.firstrankpublication.com

Program Interrupt

The process of communication described above is known as programmed control transfer. In this process the computer has to check the flag bit continually, and when it finds it set, it initiates an information transfer. Obviously if we transfer information in this way it is quiet inefficient because of the difference of information flow rate between the computer and that of the I/O device. We explain why it is so. Let a computer go through an instruction cycle of 1 micro second. Also let us assume that the I/O device can transfer data at the rate of 10 characters per second at most. This is equivalent to one character every 100,000 micro seconds. If it has to execute two instructions, it will check the flag bit and decide not to transfer the information. Hence between each transfer, the computer will check the flag 50,000 times, at the maximum rate. It is clear now that the computer is wasting time while checking the flag while in this time it can do some other useful processing task.

There is another way to perform this transfer. Let the external device inform the computer when it is ready for the transfer. In the meantime the computer can do other work. Interrupt facility is used for this type of transfer when the computer is running a program. There is no need to check the flags in this type of transfer. However when a flag is set, the computer is momentarily interrupted from executing the current program and is informed of the fact that the flag has been set. The computer deviates for a short time from the current job to take care of the input or output transfer. After that it returns to the current program to continue with it. Interrupt cycle is shown in the following Fig. 5.9.

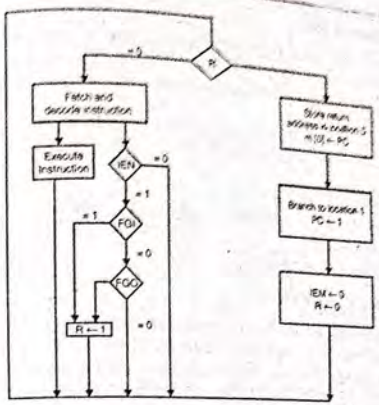


Fig. 5.9. Interrupt cycle

The interrupt enables flip-flop IEN can be set (to 1) and cleared (to 0) with two instructions. When IEN is cleared the flags cannot interrupt the computer. When IEN is set, it means that the computer can be interrupted. With the help of these two instructions a programmer can now decide whether or not to use the interrupt facility. An interrupt flip-flop R is included in the computer. When R = 0, the computer goes through an instruction cycle. While the execution phase of the instruction cycle IEN is checked by the control. If it is 0, it means we do not want to use the interrupt, therefore control continues with the next instruction cycle. If IEN is 1, control checks the flag bits. If both flags are 0, then it shows that neither the input nor the output registers are ready for information transfer. In this case, control continues with the next instruction cycle. If either flag is set to 1 while IEN = 1, flip-flop R is set to 1..

The interrupt cycle is a hardware mechanism to branch and save return address operation. The return address available in a PC is stored in a specific location where it can be found later when the program returns to the instruction at which it was interrupted.

A total computer system consists of both hardware and software. Hardware means the physical components and all other associated equipment. Software means the programs that are written for the computer. It is possible to be familiar with various aspects of computer software without being concerned with details of how the computer hardware operates.

5.7. CONTROL UNIT

The function of control unit is to generate relevant timing and control signals to all operations in the computer. It controls the flow of data between the processor and memory and peripherals

Functions of Control Unit

- The control unit directs the entire computer system to carry out stored program instructions.
- The control unit must communicate with both the arithmetic logic unit (ALU) and main memory.
- The control unit instructs the arithmetic logic unit that which logical or arithmetic operation is to be performed.
- The control unit co-ordinates the activities of the other two units as well as all peripherals and auxiliary storage devices linked to the computer.

Design of a Control Unit

Control unit generates control signals using one of the two organizations:

- Hardwired Control Unit
- Micro-programmed Control Unit

5.7.1. Hardwired Control unit

- It is implemented as logic circuits (gates, flip-flops, decoders etc.) in the hardware.
- This organization is very complicated if we have a large control unit.
- In this organization, if the design has to be modified or changed, requires changes in the wiring among the various components. Thus the modification of all the combinational circuits may be very difficult.

5.7.2. Micro programmed Control Unit

www.IPUnation.com

- A micro-programmed control unit is implemented using programming approach. A sequence of microoperations is carried out by executing a program consisting of micro-instructions.
- Micro-program, consisting of micro-instructions is stored in the control memory of the control unit.
- Execution of a micro-instruction is responsible for generation of a set of control signals.

5.8. HARDWIRED CONTROL UNIT

In the hardwired control, the control units use fixed logic circuits to interpret instructions and generate control signals from them. Figure 5.10 shows the typical hardwired control unit. Here, the fixed logic circuit block includes combinational circuit that generates the required control outputs for decoding and encoding functions. By separating the decoding and encoding functions, we can draw more detail block diagram for hardwired control unit as shown in the Fig. 5.10.

The instruction decoder decodes the instruction loaded in the IR. If IR is an 8-bit register instruction decoder generates 2^8, *i.e.* 256 lines; one for each instruction. According to code in the IR, only one line among all output lines of decoder goes high *i.e.*, set to 1 and all other lines are set to 0. The step decoder provides a separate signal line for each step, or time slot, in a control sequence. The encoder gets in the input from instruction decoder, step decoder, external inputs and condition codes. it uses all these inputs to generate the individual control signals. After execution of each instruction end signal is generated which resets control step counter and make it ready for generation of control step for next instruction.

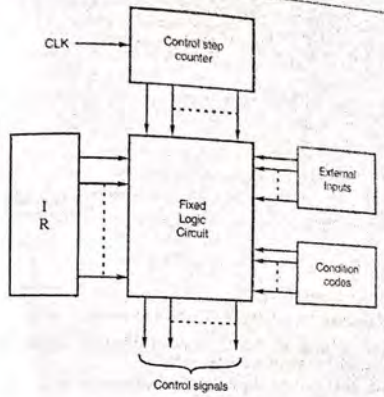


Fig. 5.10. Typical Hardwired Control Unit

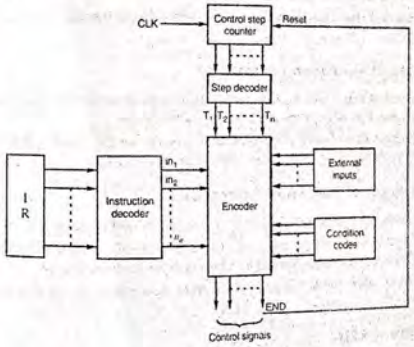


Fig. 5.11. Detail Block Diagram for Hardwired control unit

Let us see how the encoder generates signal for single bus processor organisation shown in Fig. 5.12 Y_{in}. The encoder circuit implements the following logic function to generate Y_{in}:

$$Y_{in} = T_1 + T_3 . ADD + T_4 BRANCH + \ldots$$

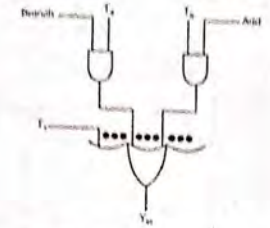


Fig. 5.12. Generation of the Y_{in} Control Signal

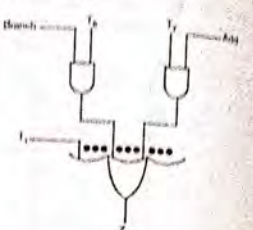


Fig. 5.13. Generation of the Z_{out} Control Signal

The Y_{in} signal is asserted during time interval T_1 for all instructions, during T_3 for an ADD instruction, during T_4 for an unconditional BRANCH instruction, and so on.

As another example, the logic function to generate Z_{out} signal can given by $Z_{out} = T_2 + T_7 . ADD + T_6 . BRANCH + \ldots$

The Z_{out} signal is asserted during time interval T_2 of all instructions, during T_7 for an ADD instruction, during T_6 for an unconditional branch instruction, and so on.

The Fig. 5.12 and 5.13 shows the hardware implementation of logic functions for Y_{in} and Z_{out} control signals.

www.IPUnation.com

5.8.1. Advantage of Hardwired Control Unit

1. Hardwired control unit is fast because control signals are generated by combinational circuit based on present state of flip-flops and other inputs conditions.
2. The delay in generation of control signals depends upon the number of gates. That means one hot method can generate control signals faster than classical method.

5.8.2. Disadvantages of Hardwired Control Unit

1. More is the control signals required by CPU, more complex will be design of control unit.
2. Hardwired control unit is a rigid approach. That means modification in control signal is very difficult. That means it requires rearranging of wires in the hardware circuit.
3. It is difficult to correct mistake in original design or adding new feature in existing design of control unit.

5.9. SPECIFYING A CPU

The first step in designing a CPU is to determine its applicatons. We don't need anything as complicated as Itanium microprocessor to control a microwave oven; a simple 4-bit processor would be

powerful enough to handle this job. However, the same 4 bit processor would be woefully inadequate to power a personal computer. The key is to match the capabilities of the CPU to the tasks it will perform.

Once we have determined the tasks a CPU will perform, we must design an instruction set architecture capable of handling these tasks. We select the instructions a programmer could use to write the application programs and the registers these instructions will use.

After this is done, we design the state diagram for the CPU. We show the micro-operations performed during each state and the conditions that cause the CPU to go from one state to another. A CPU is just a complex finite state machine. By specifying the states and their micro-operations, we specify the steps the CPU must perform in order to fetch, decode and execute every instruction in its instruction set.

In general, a CPU performs the following sequence of operations:

- **Fetch cycle:** Fetch an instruction from memory, then go the decode cycle.
- **Decode cycle:** Decode the instruction—that is, determine which instruction has been fetched—then go to the execute cycle for that instruction.
- **Execute cycle:** Execute the instruction, then go to the fetch cycle and fetch the next instruction.

5.10. DESIGN AND IMPLEMENTATION OF A SIMPLE CPU

In this section, we specify and design a Very Simple CPU, probably the simplest CPU you will ever encounter. This CPU isn't very practical, but it not meant to be. The role application of this CPU is to serve as an instructional aid, to illustrate the design process without burdening the reader with too many design details. In the next section, we design a more complex CPU, which builds on the design methods presented here.

5.10.1. Specifications for a Very Simple CPU

To illustrate the CPU design process, consider this small and somewhat impractical CPU. It can access 64 bytes of memory, each byte being 8 bits wide. The CPU does this by outputting a 6-bit address on its output pins A[5...0] and reading in the 8-bit value from memory to its input D[7...0].

This CPU will have only one programmer-accessible register, an 8-bit **accumulator** labeled AC. It has four four instructions in it instruction set, as shown in Table 5.4.

Table 5.4. Instruction Set for the Very Simple CPU

Instruction	Instruction Code	Operation
ADD	00AAAAAA	AC ← AC + M(AAAAAA)
AND	01AAAAAA	AC ← AC ∧ M(AAAAAA)
JMP	10AAAAAA	GOTO AAAAAA
INC	11XXXXXX	AC ← AC + 1

As noted earlier, this is a fairly impractical CPU for several reasons. For example, although it can perform some computations, it cannot output the results.

In addition to AC, this CPU needs several additional registers to perform the internal operations necessary to fetch, decode, and execute instructions. The registers in this CPU are fairly standard and are found in many CPUs; their sizes vary depending on the CPU in which they are used. This CPU contains the following registers:

- A 6 bit **address register**, AR, which supplies an address to memory via A[5..0].

- A 6 bit **program counter**, PC, which contains the address of the next instruction to be executed.
- An 8-bit **data register**, DR, which receives instructions and data from memory via D[7...0].
- A 2-bit **instruction register**, IR, which stores the opcode portion of the instruction code fetched from memory.

5.10.2. Fetching Instructions from Memory

Before the CPU can execute an instruction, it must fetch the instruction from memory. To do this, the CPU performs the following sequence of actions.

1. Send the address to memory by placing it on the address pins A[5...0].
2. After allowing memory enough time to perform its internal decoding and to retrieve the desired instruction, send a signal to memory so that it outputs the instruction on its output pins. These pins are connected to D[7...0] of the CPU. The CPu reads this data in from those pins.

The address of the instruction to be fetched is stored in the program counter. Since A[5...0] receives their values from the address register, the first step is accomplished by copying the contents of PC to AR. Thus, the first state of the fetch cycle is

FETCH1: AR <- PC

Next, the CPu must read the instruction from memory. The CPU must assert a **READ** signal, which is output from the CPU to memory, to cause memory to output the data to D[7...0]. At the same time, the CPU must read the data in and store it in DR, since this is the only register used to access memory. By waiting until one state after FETCH1, the CPU gives memory the time to access the requested data (which is an instruction in this case). The net result is, at first,

FETCH2: DR <- M

In fact, there is another operation that will be performed here. We must also increment the program counter, so FETCH2 should actually be as follows:

FETCH2: DR <- M, PC <- PC + 1

See Practical Perspective: Why a CPU increments PC during the Fetch Cycle for the reasoning behind this.

Finally, there are two other things that the CPU will do as part of the fetch routine. First, it copies the two high-order bits of DR to IR. As shown in Table 5.4, these two bits indicate which instruction is to be executed. As we will see in the design of the control logic, it is necessary to save this value in a location other than DR so it will be available to the control unit. Also, the CPU copies the six low-order bits of DR to AR during the fetch routine. For the ADD and AND instructions, these bits contain the memory address of one of the operands for the instruction. Moving the address to AR here will result in one less state in the execute routines for these instructions. For the other two instructions, it will not cause a problem. They do not need to access memory again, so they just won't use the value loaded into AR. Once they return to the FETCH routine, FETCH1 will load PC into AR, over-writing the unused value. These two operations can be performed in one state as

FETCH3: IR <- DR[7...6], AR <- DR[5...0]

The state diagram for the fetch cycle is shown in Fig. 5.14.

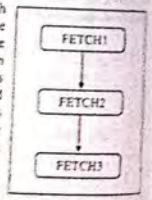


Fig. 5.14. Fetch cycle for the very simple CPU

5.10.3. Decoding Instructions

After the CPU has fetched an instruction from memory, it must determine which instruction it has fetched so that it may invoke the correct execute routine. The state diagram represents this as a series of branches from the end of the fetch routine to the individual execute routines. For this CPU, there are four instructions and thus four execute routines. The value in IR (00, 01, 10, or 11) determines which execute routine is invoked. The state diagram for the fetch and decode cycles is shown in Fig. 5.15.

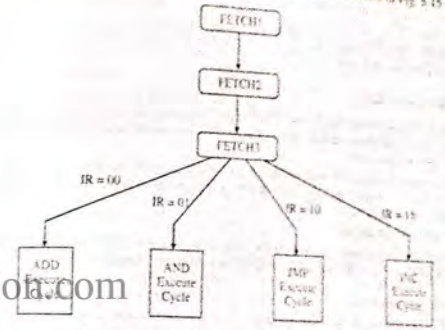


Fig. 5.15. Fetch and decode cycles for the very simple CPU

www.IPLnation.com

5.10.4. Executing Instructions

To complete the state diagram for this CPU, we must develop the state diagram for each execute routine. Now we design the portion of the state diagram for each execute routine and the overall design for the CPU. The state diagrams for the individual execute routines are fairly simple, so they are easily included in the diagram of the finite state machine for the entire CPU.

5.10.4.1. Add Instruction

In order to perform the ADD instruction, the CPU must do two things. First, it must fetch one operand from memory. Then it must add this operand to the current contents of the accumulator and store the result back into the accumulator.

To fetch the operand from memory, the CPU must first make its address available via A[5...0], just as it did to fetch the instruction from memory. This is done by moving the address into AR. However, this was already done in FETCH3, so the CPU can simply read the value in immediately. (This is the time saving mentioned earlier.) Thus,

ADD1. DR ← M

Now that both operands are within the CPU, it can perform the actual addition in one state

ADD2: AC ← AC + DR

These two operations comprise the entire execute cycle for the ADD instruction. At this point, the ADD execute cycle would branch back to the fetch cycle to begin fetching the next instruction.

5.10.4.2. AND Instruction

The execute cycle for the AND instruction is virtually the same as that for the ADD instruction. It must fetch an operand from memory, making use of the address copied to AR during FETCH3. However, instead of adding the two values, it must logically AND the two values. The states that comprise this execute cycle are

AND1: DR ← M

AND2: AC ← AC ∧ DR

5.10.4.3. JMP Instruction

Any JMP instruction is implemented in basically the same way. The address to which the CPU must jump is copied into the program counter. Then, when the CPU fetches the next instruction, it uses this new address, thus realizing the JMP.

The execute cycle for the JMP instruction for this CPU is quite trivial. Since the address is already stored in DR[5...0], we simply copy that value into PC and go to the fetch routine. The single state which comprises this execute cycle is

JMP1: PC ← DR[5...0]

In this case, we actually had a second choice. Since this value was copied into AR during FETCH3, we could have performed the operation PC ← AR instead. Either is acceptable.

www.IPUnotes.com

5.10.4.4. INC Instruction

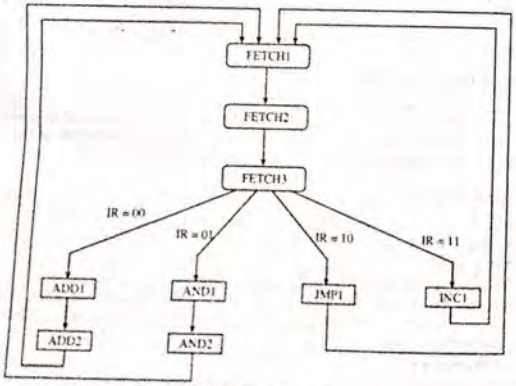


Fig. 5.16. Complete state diagram for the very simple CPU.

The INC instruction can also be executed using a single state. The CPU simply adds 1 to the contents of AC and goes to the fetch routine. The state for this execute cycle is

INC1: AC ← AC + 1

The state diagram for this CPU, including the fetch, decode and execute cycles, is shown in Fig. 5.16.

5.10.5. Establishing Required Data Paths

The state diagram and register transfers specify what must be done in order to realize this CPU. Now we must design the CPU so that it actually does these things. First, we look at what data transfers can take place and design the internal data paths of the CPU so this can be done. The operations associated with each state for this CPU are

FETCH1: AR ← PC

FETCH2: DR ← M, PC ← PC + 1

FETCH3: IR ← DR[7...6], AR←DR[5...0]

ADD1: DR ← M

ADD2: AC ← AC + DR

AND1: DR ← M

AND2: AC ← AC ∧ DR

JMP1: PC ← DR[5...0]

INC1: AC ← AC + 1

(If this looks like RTL code, you're headed in the right direction!). Note that memory supplied its data to the CPU via pins D[7...0]. Also recall that the address pins A[5...0] receive data from the address register, so the CPU must include a data path from the outputs of AR to A.

To design the data paths, we can take one of two approaches. The first is to create **direct paths** between each pair of components that transfer data. We can use multiplexers or buffers to select one of several possible data inputs for registers that can receive data from more than one source. For example, in this CPU, AR can receive data from PC or DR[5...0], so the CPU would need a mechanism to select which one is to supply data to AR at a given time. This approach could work for this CPU because it so small. However, as CPU complexity increases, this becomes impractical. A more sensible approach is to create a bus within the CPU and route data between components via the bus.

To illustrate the bus concept, consider an interstate highway that is 200 miles long and has about as many exits. Assume that each exit connects to one town. When building roads, the states had two choices: They could build a separate pair of roads (one in each direction) between every pair of towns, resulting in almost 40,000 roads, or one major highway with entrance and exit ramps connecting the towns. The bus is like the interstate highway: It consolidates traffic and reduces the number of roads (data paths) needed.

We begin by reviewing the data transfers that can occur to determine the functions of each individual component. Specifically, we look at the operations that load data into each component. It is not necessary to look at operations in which a component supplies the data or one of the operands; that will be taken care of when we look at the component whose value is being changed. First we regroup the operations, without regard for the cycles in which they occur, by the register whose contents they modify. This results in the following:

AR: AR ← PC; AR ← DR[5...0]

PC: PC ← PC + 1; PC ← DR[5...0]

DR: DR ← M

The CPU fetches, decodes and executes eachinstruction following the appropriate state sequences from the state diagram:

ADD4: FETCH1→FETCH2→FETCH3→ADD1→ADD2

AND5: FETCH→FETCH2→FETCH3→AND1→AND2

INC: FETCH1→FETCH2→FETCH3→INC1

JMP 0: FETCH1→FETCH2→FETCH3→JMP1

Table 5.6 shows the trace of the execution of one iteration of this program. We can see that the program processesevery instruction correctly. Initially all registers contain the value 0.

5.11. DESIGN OF ACCUMULATOR LOGIC

The circuits associated with the AC register are shown in Fig. 5.23. The adder and logic circuit has three sets of inputs. One set of 16 inputs comes from the outputs of AC. Another set of 16 inputs comes from the data register DR. A third set of eight inputs comes from the input register INPR. The outputs of the adder and logic circuit provide the data inputs for the register. In addition, it is necessary to include logic gates for controlling the LD, INR, CLR in the register and for controlling the operation of the adder and logic circuit.

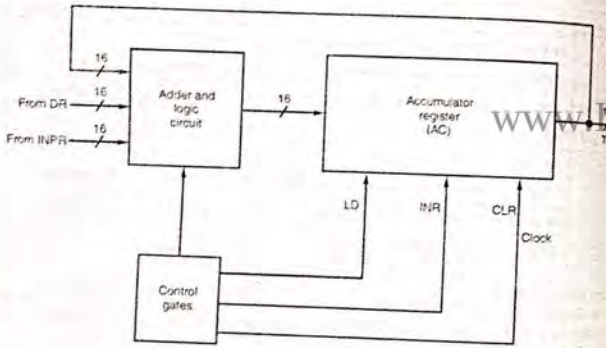


Fig. 5.23. Circuit Related to Accumulator.

In order to design the logic associated with AC, it is necessary to go over the register transfer statements and extract all the statements that change the content of AC.

D_0T_5	: $AC \leftarrow AC \wedge DR$	AND with DR
D_1T_5	: $AC \leftarrow AC + DR$	Add with DR
D_2T_5	: $AC \leftarrow DR$	Transfer from DR
PB_{11}	: $AC(0\text{-}7) \leftarrow INPR$	Transfer from INPR

rB_9	: $AC \leftarrow \overline{AC}$	Complement
rB_7	: $AC \leftarrow shr\ AC, AC(15) \leftarrow E$	Shift right
rB_6	: $AC \leftarrow Shl\ AC, (AC(0) \leftarrow E$	Shift left
rB_{11}	: $AC \leftarrow 0$	Clear
rB_5	: $AC \leftarrow AC + 1$	Increment

From this list we can derive the control logic gates and the adder and Logic circuit.

5.12. MICROPROGRAMMED CONTROL UNIT

A **micro-programmed control unit** is implemented using programming approach. A sequence of microoperations are carried out by executing a program consisting of micro-instructions.

Micro-program, consisting of micro-instructions is stored in the control memory of the control unit. Execution of a micro-instruction is responsible for generation of a set of control signals.

A **micro-instruction** consists of:

- One or more micro-operations to be executed.
- Address of next microinstruction to be executed.

Micro-Operations: The operations performed on the data stored inside the registers are called micro-operations.

Micro-Programs: Microprogramming is the concept for generating control signals using programs. These programs are called micro-programs.

Micro-Instructions: The instructions that make micro-program are called micro-instructions.

Micro-Code: Micro-programs is a group of microinstructions. The micro-program can also be termed as micro-code.

Control Memory: Micro-programs are stored in the read only memory (ROM). That memory is called control memory.

5.12.1. Control Memory

Control Memory means a memory in which control signals are stored. Control signals are in the form of micro-operations. Number of control signals in a given system is finite. When control signal are generated by hardware then they are called hard-wired control signal. Control signals can also be generated by micro-programming.

The simplest way to represent a control signal is by a single binary bit. If binary bit is 1 then it means the corresponding control signal will be executed. If binary bit is 0 then it means the corresponding control signal is not executed. The limitations of using only one bit is that only two different states can be represented by it.

If we use group of bits (bus-organised system) then many combination can be made. Each combination represent a unique control signal.

If we have two binary bits then four different control signals can be made. Similarly If we have n binary bits then 2^n different control signals can be made

Control memory is available in the form of read only memory (ROM). In a computer, there are two types of memories :

1. Main Memory
2. Control Memory

www.PUnotes.com

The main memory is used for storing the programs written by the programmer. The contents of main memory may be changed when program is executed or when some another program is stored in it. The user program in main memory contains instructions and data.

The control memory contains the control signals in the form of micro-operations which are fixed and are not changed by the user.

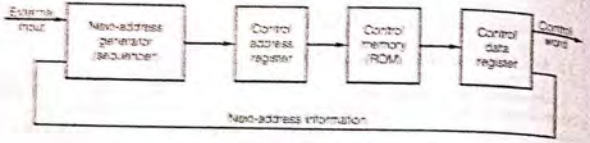


Fig. 5.24 Microprogrammed Control Organization

5.12.2. Address Sequencing

Each instruction has unique address. Making particular sequence of addresses of instructions is known as address sequencing. In the control memory micro-instructions are stored in groups. Each group represent specific routine. For execution of instruction each computer has its own routine. For proper execution of the instructions the hardware of the system must provide the facility to control the address sequencing of the control memory. Because the order of execution of instructions plays an important role. Control memory must also provide the facility of jumping from one routine to another routine if necessary.

www.ifunation.com

Hardware is required to select the address of next micro-instruction in the control memory. It contains a set of control bits. The address of next micro-instruction can also be obtained explicitly by using an address field.

The address of next micro-instructions of control memory can be generated by different way. It can be received by Control Address Register through:

1. Mapping process
2. Branch Logic
3. Increment
4. Sub-routine

This can be shown in Fig. 5.25.

In general we can say that the address sequencing in control memory required the following facilities:

1. A mapping process between the bits of micro-operation and bits of ROM address.
2. Unconditional branch for the address field of micro-instruction.
3. Conditional branch for status bits in the registers.
4. Incrementing of control address register.
5. Facility for sub-routine calls and returns.

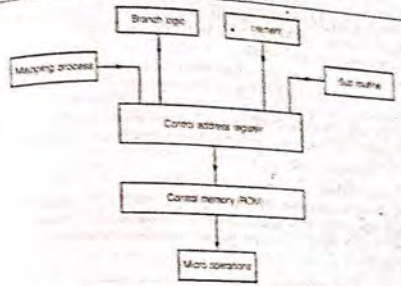


Fig. 5.25. Address Sequencing

5.12.3. Microinstruction Format

The computer instruction format is shown in Fig. 5.26 (a). It consists of three fields : a 1-bit field for indirect addressing symbolized by I, a 4-bit operation code (opcode), and an 11-bit address field. Figure 5.26 (b) lists four of the 16 possible memory reference instructions. The ADD instruction adds the contents of the operand found in the effective address to the content of AC. The BRANCH instruction takes the effective address if the operand in AC is negative. The program proceeds with the next consecutive instruction if AC is not negative. The AC is negative if its sign bit (the bit in the left most position of the register) is a 1. The STORE instruction transfers the content of AC into the memory word specified by the effective address. The EXCHANGE instruction swaps the data between AC and the memory word specified by the effective address.

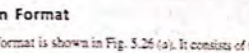


Fig. 5.26 (a). Instruction Format

Symbol	Opcode	Description
ADD	0000	AC ← AC + M [EA]
BRANCH	0001	If (AC < 0) then (PC ← EA)
STORE	0010	M [EA] ← AC
EXCHANGE	0011	AC ← M [EA], M [EA] ← AC

EA is the effective address

Fig. 5.26 (b). Four Computer Instructions

The microinstruction format for the control memory is shown in Fig. 5.27. The 20 bits of the microinstruction are divided into four functional parts. The three fields F1, F2 and F3 specify

microoperations for the computer. The CD filed selects status bit conditions. The BR field specifies the type or branch to be used. The AD field contains a branch address. The address field is seven bits wide, since the control memory has $128 = 2^7$ words.

The microoperations are subdivided into three fields of three bits each. The three bits in each field are encoded to specify seven distinct microoperations. This gives a total of 21 microoperations. No more than three microoperations can be chosen for a microinstruction, one from each field. If fewer than three microoperations are used, one or more of the fields will use the binary code 000 for no operation. As an illustration, a microinstruction can specify two simultaneous from F2 and F3 and none from F1.

DR ← M[AR] with F2 = 100

and PC ← PC + 1 with F3 = 101

The nine bits of the microoperation fields will then be 000 100 101. It is important to realize that two or more conflicting microoperations cannot be specified simultaneously. For example, a microoperation field 010 001 000 has no meaning because it specifies the operations to clear AC to 0 and subtract DR from AC at the same time.

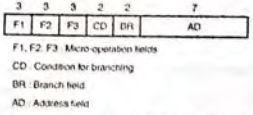


Fig. 5.27. Microinstruction Code format (20-bits)

Microinstruction formats are classified under two categories.

1. Horizontal Format
2. Vertical Format

www.IPUnation.com

5.12.3.1. Horizontal Format

Horizontal microinstruction format is having following characteristics.

Fig. 5.28. Horizontal Format

1. This instruction format contains an individual bit for each control signals. Since each bit represents one control signal. That means more than one control signals are generated by single microinstructions.
2. Since individual bits are used to represent control signals. Hence these instruction formats are having larger length, which can be up to 120-bits per microinstruction.
3. No decoding of control bit signals are required because individual bit represents one control signal. Hence execution of these instructions will be fast.
4. Single microinstruction specifies the number of microoperations. Hence this format is having the ability to represent high degree of parallelism.

5.12.3.2. Vertical Format

Vertical microinstructions format is having following characteristics.

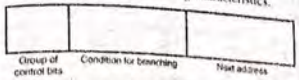


Fig. 5.29. Vertical Format

1. This instruction format contains single field (group of bits) that can produce encoded sequence of control signals. Hence only one control signal is activated at a time.
2. Since group of bits are used to represent control signals. Hence instruction format are shorter in length than that of horizontal microinstruction format.
3. Decoding of control signal is required. Hence this instruction format will take more time to generate control signals.
4. The overall cost of this format is less because microinstructions are smaller in size up to 30-bits. Moreover the number of bits at one location in control memory is less.

5.12.4. Micro-program Sequencer

Micro-Program control unit consists of two part. These are as follows :

1. Control Memory
2. Micro-Program Sequencer

Control memory stores the micro-instructions and its electronic circuit that generated the next address of the micro-instruction that is to be executed.

The electronic circuit which is responsible for the generation of next address of the micro-instruction is known as Micro-Program sequencer because it sequence the micro-instructions in the control memory. A micro-program sequence can be constructed according to the requirement of the given application. It can be constructed by using integrated circuit in the form of ROM.

The aim of micro-program sequencer is to give the address of micro-instruction to the control memory, which is to be executed. The responsible part of the program sequencer which determine the address of next micro-instruction that specific address is loaded into the control address register.

The address of next micro-instruction depends upon the corresponding bits that address-sequencer receives for calculation of address.

Some program-sequencer contains internal-register-stack for storing addresses during micro-program looping and subroutine calls.

The graphical representation of micro-program sequencer in the control unit can be shown as in Fig. 5.30.

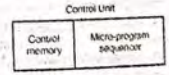


Fig. 5.30. Micro-program Sequencer

5.13. DIFFERENCE BETWEEN HARDWIRED AND MICROPROGRAMMED CONTROL UNIT

Attribute	*Hardwired Control*	*Microprogrammed Control*
Speed	Fast	Slow
Control Functions	Implemented in hardware	Implemented in software
Flexibility	Not flexible, to accommodate new system specifications or new instructions.	More flexible, to accommodate new system specification or new instructions redesign is required.
Ability to Handle Large/ Complex Instruction sets	Some what difficult	Easier
Ability to Support Operating Systems and Diagnostic Features	Very difficult (unless anticipated during design)	Easy
Design Process	Somewhat complicated	Orderly and systematic
Applications	Mostly RISC microprocessors	Mainframes, some microprocessors
Instruction set Size	Usually under 100 instructions	Usually over 100 instructions
ROM size	—	2K to 10K by 20-400 bit microinstructions
Chip Area Efficiency	Uses least area	Uses more area

www.IPUnation.com

5.14. MICRO CODE

Micro Code is a means of implementing a hardware CPU with some assistance from software rather than purely in hardware: typically a user cannot tell whether microcoding was used in implementing a CPU (although there have been some user-modifiable microcoded CPUs).

There is a range of kinds of microcode ranging from "vertical" to "horizontal". Vertical microcoding means that the underlying CPU is a straightforward one-thing-at-a-time instruction set roughly similar to a RISC processor like a Sparc. Horizontal microcoding means that the underlying CPU presents many field per instruction, which each individually control some part of the ALU directly. So one 1-bit field might enable output from register 12 while another 1-bit field enabled input to the AND operation.

This approach gives very precise control over the workings of the ALU, with the ability to specify quite a few things in parallel,but consumes a large number of bits per micro-instruction, since the control fields, in extreme cases, are all just on-off rather than multiplexed.

In the middle of the spectrum, a microcoded CPu might have instructions with only a moderate number of control fields, each multiplexed (*e.g.* a single 4-bit field might specify one of 16 registers to use as output), and hence take up less space than a purely horizontal microcoded engine, but still more than a highly vertical one. All other things being equal, a purely vertical microcoded CPU will be slow and implemented only for prototyping or compatibility, while a highly horizontal microcoded CPu is potentially very fast, but can consume a surprisingly large amount of expensive (in terms of real estate) microcode store.

Microcoding seemed to fall out of favour for some years with the advent of microprocessors, but eventually came into use again as an implementation technique for microprocessor cores.

5.15. COMPLETE COMPUTER DESCRIPTION

Complete Computer Description is given in the flowchart. It consists of instruction cycle, including the interrupt cycle for the basic computer as shown in Figure 5.31. The interrupt flip-flop R may be set at any time during the execution phase. Control returns to timing signal T0 after SC is cleared to 0. If R = 1, the computer goes through an interrupt cycle. If R = 0, the computer goes through an instruction cycle. If the instruction is one of the memory-reference instructions, the computer first checks if there is an indirect address and then continues to execute the decoded instruction according to the flowchart of Figure 5.31.

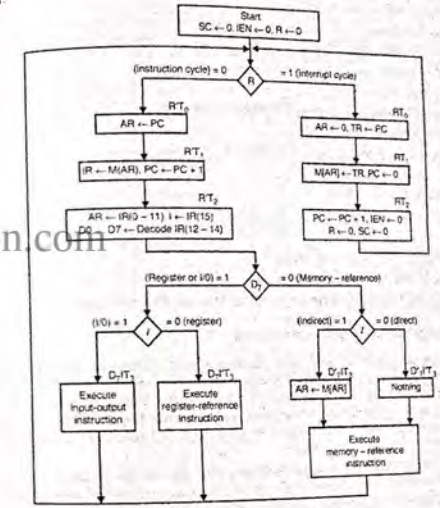


Fig. 5.31. Flowchart for computer operation

5.16. PENTIUM MICROPROCESSOR

The Pentium processor is an advanced 32-bit superscalar processor with 64-bit data bus and 32-bit address bus to address up to 4 Gb of physical memory space. It was released in the year 1993 and consists of 3.1 million transistors. The Pentium employs two general purpose integer pipelines, branch prediction, highly pipelined floating point unit and separate code and data caches to achieve the highest performance

level while preserving the binary code compatibility with 80x86 processors. The Pentium processor can execute two integer instructions simultaneously. The Pentium is available with maximum clock speed ratings of 60 to 233 MHz.

The features of Pentium processor are:

- Superscalar architecture
- Dynamic branch prediction
- Pipelined Floating-Point Unit
- Improved instruction execution time
- Separate code and data caches
- 64-bit data bus
- Bus cycle pipelining
- Address parity
- Internal parity checking
- Functional redundancy checking and lock-step operation
- Execution tracing
- Performance monitoring
- IEEE 1149.1 boundary scan
- System Management Mode
- Virtual mode extensions
- Dual processing support
- Advanced power management feature
- Fractional bus operation
- On-chip local APIC (Advanced Programmable Interrupt Controller) device.

www.IPUnation.com

5.16.1 Architecture of Pentium Processor

The Pentium processor has superscalar architecture which allows parallel execution of two instructions. The various functional blocks of Pentium processor are shown in Fig. 5.32. The functional units of Pentium processor are Bus unit, Paging unit, Prefetch buffer, Instruction decode, Control ROM, Execution unit with two integer pipelines (U-pipe and V-pipe), Branch Target Buffer (BTB), Code cache, Data cache, Floating Point Unit (FPU), Dual Processing (DP) logic, and Advanced Programmable Interrupt Controller (APIC).

The bus unit takes care of the issuing control signals and fetching the code/data from the external memory and I/O devices. The Pentium processor has 64-bit external data bus and supports burst read and burst writeback cycles. In addition, bus cycle pipelining has been added to allow two bus cycles to be in progress simultaneously. The paging unit contains optional extensions to the architecture which allow 2 Mb and 4 Mb page sizes.

The code cache, branch target buffer and prefetch buffers are responsible for getting instructions into the execution unit. Instructions are fetched from the code cache or from the external memory. Branch addresses are remembered by the branch target buffer. The code cache TLB (Translation Lookaside Buffer) translates linear addresses to physical addresses used by the code cache.

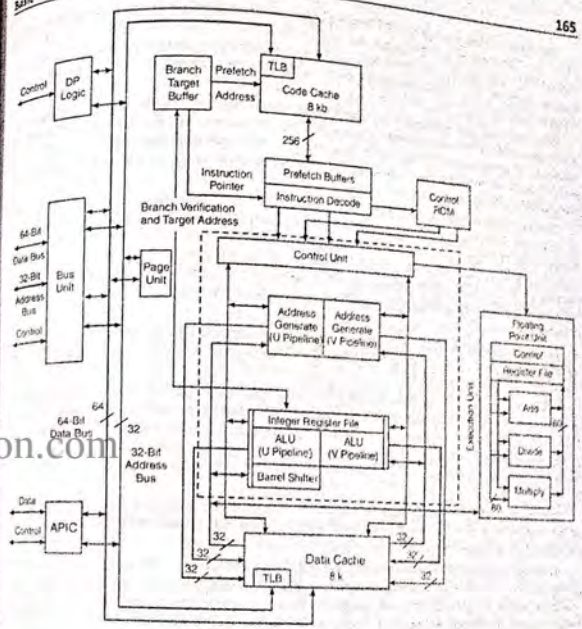


Fig. 5.32. Architecture of a Pentium processor

The Pentium processor has two independent pairs of 32-byte prefetch buffers which operate in conjunction with BTB (Branch Target Buffer). Only one prefetch buffer will be active at any one time. One prefetch buffer will fetch instructions sequentially until a branch instruction is encountered. When a branch instruction is fetched, the BTB will predict whether a branch will occur or not. If the BTB predicts that the branch will not take place then the prefetch buffer will continue fetching instructions linearly. If the BTB predicts that the branch will take place then the other prefetch buffer is enabled and begins to fetch instructions from branch target address. In this way the instructions needed for branching are also prefetched and kept ready for decode and execution.

If a branching is mispredicted then the instruction pipelines are flushed and the first buffer will start prefetch activity. The penalty for misprediction is three clocks in U-pipe and four clock in V-pipe. Mispredicted calls and unconditional jump instructions have a three clock penalty in either pipe.

The execution unit has two parallel integer pipelines U-pipe and V-pipe with individual ALU each pipe. The pipeline has five stages and they are Prefetch (PF), Decode Stage-1 (D1), De Stage-2 (D2), Execute (E) and Write back (WB). The U-pipe can execute all integer and flo point instructions. The V-pipe can execute simple integer instructions and the FXCH floating p instruction.

While executing (previous) instructions, the processor checks the next two instructions. If the exa of one instruction does not depend on the other then the first instruction is issued to U-pipe and the se instruction is issued to V-pipe, so that two instructions can be executed simultaneously. If it is possible to execute two instructions simultaneously then the two instructions are issued to U-pipe an one and no instruction is issued to V-pipe.

The control ROM contains microcodes to control the sequence of operations that must be perfo to implement the Pentium processor architecture. The control ROM unit has direct control over pipelines. The Pentium processor has an 8 kb code cache and 8 kb data cache. These caches are transpa to application software to maintain compatibility with previous INTEL processors. The data cache supports the MESI (Modified/Exclusive/Shared/Invalid) cache consistency protocol. The code cach write protected to prevent code corruption and so supports a subset of MESI protocol, the S (Shared) I (Invalid) states.

The cache is organized as a 2-way set associative cache. There are 128 sets in each cache with set containing 2 lines. Each cache line is 32 bytes wide. The replacement in the cache is handled by LR (Least Recently Used) mechanism.

The code cache is connected to prefetch buffer by a 256-bit bus. Hence, in one clock cycle instruction cache can provide up to 32 bytes (32 × 8 = 256) of raw opcode to prefetch buffer. The cache has two ports and each port is connected to pipelines by a 32-bit. Hence the data cache provide data for two data references simultaneously. The caches are accessed with physical and each cache has its own TLB (Translation Lookaside Buffer) to translate linear addresses to phys address.

The Pentium processor has a pipelined Floating Point Unit (FPU) working independently. The of Pentium processor is up to ten times faster than the FPU of 80486 processor for common operati including add, multiply and load.

The Pentium processor supports clock control. When the clock to the processor is stopp power dissipation is virtually eliminated and this makes the Pentium processor a good choice energy efficient designs. The Pentium processor supports fractional bus operation. This allow processor core to operate at high frequencies, while communicating with the external bus at low frequencies.

The Pentium processor contains an on-chip Advanced Programmable Interrupt Controller (APIC) which supports multiprocessor interrupt management, multiple I/O subsystem support, 82C59 compatibility and inter-processor interrupt support.

The Pentium processor has in-built logic for dual processing mode of operation. In dual processor mode, two identical Pentium processors can be interfaced to a single system bus. The dual processor appears to the system bus as a single, unified processor. Multiprocessor operating systems prompt schedule computing tasks between the two processors. This scheduling of task is transparent to software applications and the end user. Through a private bus, the two processors arbitrate for the external bus and maintains cache coherency. In dual processing mode, both the processor should have the same bus to core frequency ratio.

5.17. ADVANCED PENTIUM PROCESSORS

INTEL has released a number of advanced microprocessors after pentium with advanced features. In the Pentium series, the INTEL has released Pentium Pro, Pentium II, Pentium III and Pentium 4 processors. The Pentium Pro, Pentium II and Pentium III have a common architecture and the architecture of these processors are known as INTEL'S P6 microarchitecture. The architecture of Pentium 4 is known as NetBurst Microarchitecture. The higher clock version of Pentium 4 incorporates Hyper Threading (HT) technology. In all these advanced Pentium processors, the basic data size and memory addressing capability has been retained, but the processors are provided with enhanced features and advanced/sophisticated method of instruction execution. The salient features of the advanced Pentium processors are presented in the following sections:

5.17.1. Pentium PRO

The Pentium pro is a 32-bit processor with 64-bit data bus and 36-bit address bus to address up to 64 Gb of physical memory space. It was released in the year 1995 and consists of 5.5 million transistors. It is a 387 pin IC and available in PGA (Pin Grid Array) package. It is available with maximum internal clock ratings of 150/166/180/200 MHz.

The features of Pentium pro processor are:

- Three-way superscalar architecture.
- Five parallel execution units and 12-stage super pipeline.
- Dual cavity PGA ceramic packages with a CPU die and a secondary cache die.
- Out of order execution and speculative execution.
- DIB (Dual Independent Bus) architecture.
- Register renaming.
- Error checking and correcting codes.
- Improved power management with two extra modes (Stop Grant and Auto HALT modes).
- Internal micro-ops similar to RISC like instructions.
- Transactional 10 bus.
- Scalable up to four processors.
- Fault analysis/recovery.
- Integrated level two (secondary) cache of 256 k/512 k/1 Mb.
- Internal thermal protection.

5.17.2. Pentium II

The Pentium II was released in 1997 and consists of 7.5 million transistors. It is actually a Pentium pro processor with on-chip MMX (Multi Media Extension). It is also a 32-bit processor with 64-bit data bus and 36-bit address bus to address up to 64 Gb of physical memory space. It is available with maximum internal clock ratings of 233 MHz to 450 MHz and in SEC (Single Edge Connector) cartridge packaging or as a boxed processor along with fan/heatsink.

The features of Pentium II processor are:

- Supports the INTEL architecture with dynamic execution.
- Integrated primary (LI) 16 kb instruction cache and 16 kb write back data cache

- Integrated 256 kb second level (L2) cache.
- Fully compatible with previous microprocessors.
- Supports MMX technology.
- Quick start and Deep sleep modes provide extremely low power dissipation.
- Low power GTL + processor system bus interface (GTL : Gunning Transceiver Logic)
- Integrated math co-processor (Floating point unit compatible with IEEE std 754).
- Integrated thermal diode for measuring processor temperature.

5.17.3. Pentium III

The Pentium III was released in 1999 and consists of 9.5 million transistors. The higher clock version of Pentium III consists of 28 million transistors. The Pentium III is a 32-bit processor with a 64-bit data bus and a 36-bit address bus to address up to 64 Gb of physical memory space. It is available with maximum internal ratings of 500 MHz to 1 GHz.

In the IC form, it is available as 370-pin IC in PGA (Pin Grid Array) package. The Pentium III is an advanced version of Pentium II with improved MMX technology and processor serial number. The INTEL has incorporated a processor serial number in Pentium III which supports the concept of processor identification. Each Pentium III processor has a 96-bit processor number accessible by software (of various applications) to identify a system. Some of the applications that may utilize processor serial number are membership authentication, data backup/restore protection, removable storage data protection, managed access to files, etc.

The features of pentium III processor are:

- Dynamic execution microarchitecture.
- Optimized for 32-bit applications running on advanced 32-bit operating systems.
- Fully compatible with previous microprocessors.
- Integrated high performance 16 kb instruction and 16 kb data, nonlocking level one cache.
- Integrated 512 kb full speed level two cache allows for low latency on read/store operation.
- 256-bit cache data bus provides extremely high throughput on read store operation.
- Eight-way cache associativity provides improved cache hit rate on read/store operations.
- Error correcting code for system bus data.
- Data prefetch logic.
- Internet streaming SIMD (Single Instruction Multiple Data) Extensions for enhanced Video, Sound and 3D performance.
- System management mode and multiple low-power states.
- Flip Chip Pin Grid Array (FC-P6A2) packaging technology which offers improved handling protection and socket ability.
- Intel processor serial number.

5.17.4. Pentium IV

The Pentium 4 processor was released in 2000 and consists of 42 million transistors. It is available with maximum internal clock ratings of 1.4 GHz to 2.8 GHz. The Pentium 4 processor with HT (Hyper Threading) Technology was released in the year 2002 and consists of 55 million transistors. It is available

with maximum internal clock ratings of 2.4 GHz to 3.3 GHz. It is available as 478-pin IC in PGA (Pin Grid Array) package.

The features of Pentium 4 processor are:

- INTEL NetBurst microarchitecture
- Hyper Threading (HT) technology.
- Hyperpipelined technology which supports advanced dynamic execution and very deep out-of-order execution.
- Rapid execution engine-ALUs run at twice the processor core frequency.
- System bus frequency at 400/533/800 MHz.
- Binary compatible with applications running on previous members of INTEL processors.
- 8 kb level 1 data cache.
- Level 1 execution trace cache stores 12 k micro-ops and removes decoder latency from main execution loops.
- 512 kb advanced transfer cache with 8-way associativity and error correcting code.
- 144 streaming SIMD Extensions 2 (SSE2) instructions. (SIMD: Single Instruction Multiple Data).
- System management mode and mulitple low power states.

The features of INTEL NetBurst microarchitecture are hyper pipelined technology, a rapid execution engine, 400/533/800 MHz system bus, execution trace cache, advanced dynamic execution, advanced transfer cache, enhanced floating point and multimedia unit and streaming SIMD Extensions 2 (SSE2).

The hyper pipelined technology doubles the pipeline depth in the Pentium-4 processor with 512 kb cache allowing the processor to reach much higher core frequencies. The rapid execution engine allows the two integer ALUs in the processor to run at twice the core frequency, which allows many integer instructions to execute in 1/2 clock tick.

The Hyper Threading (HT) technology allows a single physical Pentium-4 processor to function as two logical processors. Each logical processor has its own architecture state, own set of general purpose registers and control registers to provide increased system performance in multitasking environments.

www.IPUnitedu.com

SOLVED EXAMPLES

Example 5.1. *A computer uses a memory unit with 256 K words of 32 bits each. A binary instruction code is stored in one word of memory. The instruction has four parts: an indirect bit, an operation code, a register code part to specify one of 64 registers, and an address part.*

(a) How many bits are there in the operation code, the register code part, and the address part ?
(b) Draw the instruction word format and indicate the number of bits in each part.
(c) How many bits are there in the data and address inputs of the memory ?

Solution.

$$256\text{ K} = 2^8 \times 2^{10} = 2^{18}$$

$$64 = 2^6$$

(a)

Address :	18 bits
Register code	6 bits
Indirect bit :	1 bit
	25

Example 5.12. *The control memory has 4096 words of 24 bits each.*

(a) How many bits are there in the control address register ?

(b) How many bits are there in each of the four inputs shown going into the multiplexers ?

(c) What are the number of inputs in each multiplexer and how many multiplexers are needed ?

Solution. Control memory = $2^{12} \times 24$

(a) 12 bits

(b) 12 bits

(c) 12 multiplexers, each of size 4-to-1 line.

REVIEW QUESTIONS

1. Discuss the fetch decode execute cycle
2. Explain instruction cycle.
3. What is instruction code.
4. Name all the computer registers.
5. Explain interrupt cycle.
6. Differentiate between all the computer instructions.
7. Explain hardwired and microprogrammed control unit.
8. Discuss the process of implementing a control unit.
9. What do you mean by microinstructions sequencing.
10. Explain the architecture of register based CPU.

www.JKmation.com

CPU Architecture and Instruction Set

6

The part of the computer system that performs the bulk of data processing is known as Central Processing Unit and it is referred as CPU. The CPU is the brain of a computer. The program which is to be executed is stored in the main memory. A program is a sequence of instructions to perform a specified task. The CPU fetches instruction codes from the memory and decodes them. The CPU also reads data from the memory, which are required for instruction execution. When the required data for the execution of instruction is at hand, the CPU executes the instruction.

The CPU is made up of three major parts as depicted in Fig. 6.1.

(*i*) Arithmetic and logic unit (ALU)

(*ii*) Control unit.

(*iii*) Register set.

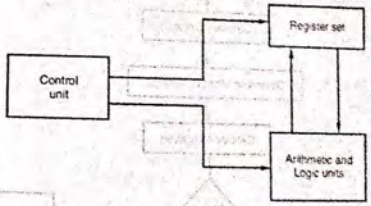


Fig. 6.1. Components of CPU

CPU contains a number of registers to store data temporarily during the execution of a program. The number of registers differ from processor to processor. The function of an ALU is to perform arithmetic and logic microoperations for executing the instructions. The control unit supervises the transfer of information

among various registers and instructs the ALU as to which operation to perform. It also controls all devices such as memory, input and output devices connected to the CPU.

Besides executing programs the CPU also controls input devices, output devices and other components of the computer. It controls input and output devices to receive and send data. Under its control programs and data are stored in the memory and displayed on the CRT screen.

6.1. CPU ORGANIZATION

The primary function of CPU is to execute sequence of instructions, execute the programs that are stored in an external main memory. A program residing in the memory unit of the computer consists of a sequence of instructions. The program is executed in the computer by going through a cycle for each instruction. Each instruction cycle in turn is subdivided into a sequence of phases. In the basic computer each instruction cycle consists of the following phases:

1. Fetch the instruction from memory.
2. Decode the instruction.
3. Fetch the operand.
4. Execute the instruction.

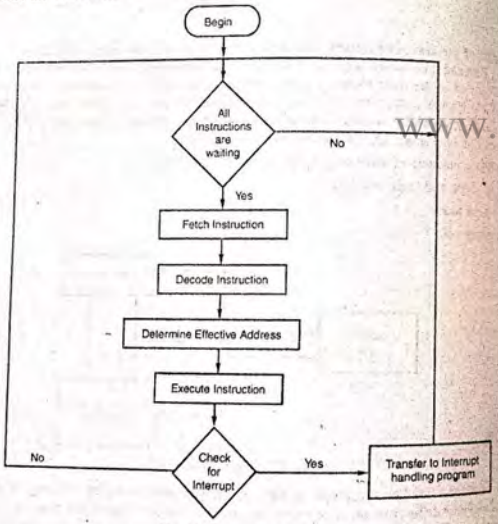


Fig. 6.2. Instruction cycle

Upon completion of step 4 the control goes back to step 1 to fetch the next instruction from memory. This process continues indefinitely unless a HALT instruction is encountered. The overview of CPU behaviour during execution of the instruction can be explained using the following flow chart.

Traditionally, the superiority of a computer is decided on the basis of its instruction set. The total number of instructions and their powerfulness is given utmost importance since these two factors contributed to the efficiency of the computer. An efficient program is the one which is short and occupies less memory space. Execution time is also a key factor.

The modern trend follows the use of simple instructions which results in a simplified control unit. The CPU circuitry is more important then the memory size. Thus, the CPU speed is enhanced for instruction processing of a RISC architecture.

The seelection of an instruction set for a computer depends on the manner in which the CPU is organized. Traditionally, there are three different CPU organizations with certain specific instructions.

1. Register Set Organization
2. Stack Organization
3. Single Accumulator Organization

6.2. GENERAL REGISTER SET ORGANIZATION

A bus organization for seven CPU registers is shown in Fig. 6.3. The output of each registers is

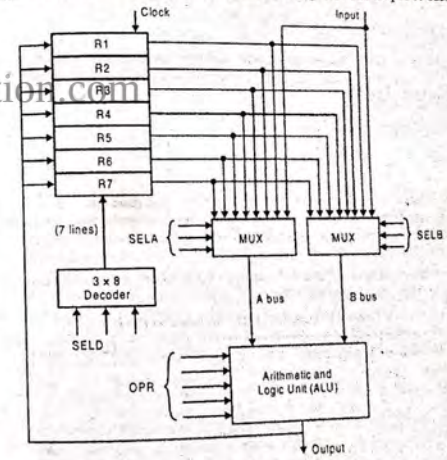


Fig. 6.3. General Register Organization.

www.JNTUnation.com

connected to two multiplexers (MUX) to form the two buses *A* and *B*. The selection lines of each multiplexer select one register or the input data for the particular bus. The *A* and *B* buses form the inputs to a common ALU. The operation selected in the ALU determines the arithmetic or logic microoperations to be performed. The result of the microoperation is available for the output data and also goes into the inputs of seven registers. The register that receives the information from the output bus is selected by the decoder. The decoder activates one of the register load inputs and thus providing a transfer path between the output data bus and the inputs of the selected destination register.

Let the operation be $R1 \longleftarrow R2 + R3$

To perform this operation, the control must provide

(i) SELA : place the contents of *R2* into bus *A*.

(ii) SELB : place the contents of *R3* into bus *B*.

(iii) ALU operation selector OPR : provide the arithmetic addition $A + B$.

(iv) SELD : transfer the contents of the output bus into *R1*.

At the beginning of a clock cycle, the four control selection variables generated *R2* and *R3* must be available in the control unit. Two source registers propagate through multiplexers and the ALU, to the output bus, and to the input of destination register, during the clock cycle interval. At the next clock transition, the information from output bus is transferred to the destination register *R1*.

Control Word

There are 14 binary selection inputs in the units, and their combined value specifies a control word. It consists of four field as shown in Fig. 6.4. The group of binary bits assigned to perform a specific operation is known as control word.

3	3	3	5
SELA	SELB	SELD	OPR

Fig. 6.4. Control Word

Three fields contain three bits each, one field has five bits. The three bits of SELA select a source register for the *A* input of the ALU. The three bits of SELB select a register for the *B* input of the ALU. The three bits of SELD select a destination register using the decoder and its seven load outputs. The five bits of OPR select one of the operations in the ALU.

The 14-bit control word when applied to the selection inputs specify a particular microoperations. The encoding of register selections is specified in Table 6.1.

Table 6.1. Encoding of Register Selection Fields.

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*

When the 3-bit binary code for SELA or SELB is 000, the respective multiplexer selects the external input data as shown in Fig. 6.4, and when the 3-bit binary code for SELD = 000, no destination register is selected and the content of output bus is for external output.

The OPR field has five bits. The encoding for five bit OPR field is specified in the Table 6.2.

Table 6.2. Encoding of ALU Operation.

OPR	Operation	Symbol
00000	Transfer *A*	TSFA
00001	Increment *A*	INCA
00010	Addition	ADD
00101	Subtract	SUB
00110	Decrement *A*	DECA
01000	AND *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

Let the microoperation given by the statement is

$$R1 \longleftarrow R4 \wedge R5$$

This statement specifies *R4* for the *A* input of ALU, *R5* for the *B* input of ALU, and *R1* as the destination register. The microoperation to be performed is AND operation between *R4* and *R5*. The control word for the above statement according to the Table 6.1 and Table 6.2 is as follows :

SELA	SELB	SELD	OPR
R1	*R4*	*R5*	AND
001	100	101	01000

Thus, the control word is 001 100 101 01000.

www.Pnation.com

6.3. STACK ORGANIZATION

The stack is also known as last-in first-out list. The stack can be consider as a storage method in which the item that stored last is the first item to be removed. The most common example of the stack in common, is a pile of trays in a cafeteria. A tray which is placed last on the top of pile is the first one taken off. A stack is an ordered collection of items which permits the insertion or deletion of an item at one end only at one end.

The stack in a digital computer is a part of memory unit. Also, with the stack an address register is associated that holds the address of the last element stored in the stack. This address register is known as Stack Pointer (SP). Thus, the stack pointer always points to the top most element of the stack. The insertion operation is known as push and the deletion operation is known as pop.

Push and Pop Operation

Insertion and deletion of items are the operations related with the stack. The process of insertion an item into the stack is known as *push operation*. The process of deleting an item from the stack is known a *pop operation*. These operation are done by incrementing or decrementing the Stack Pointer.

Register Stack

A stack can be organized by a finite number of registers or a stack can be a finite number of memory words. The stack pointer contains the address of the word that is currently on the top of the stack. A 32-word register stack is shown in Fig. 6.5. The stack pointer contains a binary value. Currently, there are four items X1, X2, X3 and X4 are placed in the stack with X4 at the top of stack so the content of stack pointer is 4. The items are removed from the stack by using POP instruction. When we remove the top item X4 from the stack, X3 is now on top of stack and the content of SP is decrement so that the SP holds address 3. To insert a new item, first the SP will incremented and then the item is inserted so that SP points to the top of the stack.

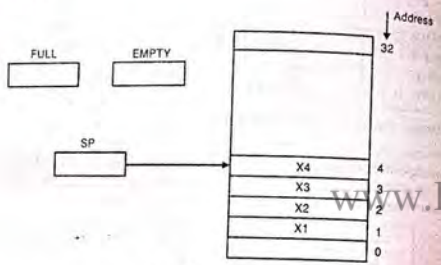


Fig. 6.5. Register Stack.

In a 32-word register stack, the address of each location will be of five bits since $2^5 = 32$. Thus, stack pointer will be of five bits and cannot exceed the value 11111. Thus, when the SP pointer content is 11111, the one-bit register FULL is set to 1, indicating that the stack is full and there is no location empty for any more item. Similarly when the content of SP = 00000 another one-bit register EMPTY is set to 1 indicating that the stack is empty and there is no element in the stack that can be deleted from the stack. The data register DR holds the items that is to written into the stack or read out of the stack.

Initially, the SP is cleared to 0 so the stack pointer points to the word at address 0. Also, the one-bit register FULL is cleared to 0, indicating that the stack is not full and the register EMPTY is set to 1. A new item is inserted into the stack by push operation. The PUSH operation will be the set of following microoperations.

SP ← SP + 1	Increment stack pointer
M[SP] ← DR	Add item on the top of stack
If (SP = 0) then (FULL ← 1)	Check if stack is full
EMPTY ← 0	Mark the stack not empty.

If the stack is not empty, an item can be deleted from the stack using the POP operation. The POP operation is implemented by the following set of microoperations.

DR ← M[SP]	Read item from the top of stack
SP ← SP − 1	Decrement stack pointer
If (SP = 0) then (EMPTY ← 1)	Check if stack is empty
FULL ← 0	Mark the stack not full.

The top item is read from the stack into DR, then the SP is decremented by 1 so that it points to top of stack. The SP is checked whether it is zero on not. If zero, EMPTY sets to 1 indicating that the stack is empty.

Memory Stack

A stack can also be implemented using random-access memory attached to the CPU. This can be implemented by assigning a portion of memory for stack operation using the processor register as a stack pointer. The computer memory is partitioned into three parts as program, data, and stack as shown in Fig. 6.6. PC points to the address of the next instruction stored in memory. The stack pointer (SP) points to the top of the stack.

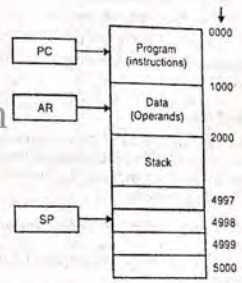


Fig. 6.6. Computer Memory Divided into Segments Program, Data and Stack.

The initial value of SP is 5000 and the first item stored in stack is at address 4999, the second item at address 4998 and so on. The last address that can be used for stack is 2000 *i.e.* the final value of stack is 2000. The stack grows in reverse order with decreasing addresses. A new item into the stack is inserted using PUSH operations as :

SP ← SP − 1
M[SP] ← DR

The stack pointer is decremented first so that it points to the next address of the stack and then the item from the data register is inserted into the top of the stack. An item can be deleted from the stack using POP operation as

www.IPUnation.com

$$DR \leftarrow M[SP]$$
$$SP \leftarrow SP + 1$$

The top item of the stack is read into the data register DR and then the stack pointer is incremented by 1 so that it points to the top item of the stack.

Most of the computer does not provide any method to check the stack overflow or underflow or check whether the stack is full or empty. One possible method is to used two processor registers holding the addresses 2000 (upper limit) and 5000 (lower limit) respectively. Then, stack pointer is compared every time the push operation takes place with the upper-limit register and its with the lower-limit register, after the pop operation takes place.

Reverse Polish Notation

Let us consider an expression $x + y$. The plus operator is placed in between the two operands x and y. Such a notation is known as *infix notation*. If the operator is placed before the two operands as $+xy$, the notation is said to be *prefix notation*, also known as polish notation. If the operator is placed after the two operands as $xy+$, the notation is said to be *postfix notation*, also known as **Reverse polish notation**. Thus, the three notation are

$x + y$ Infix notation

$+ xy$ Prefix or polish notation

$xy +$ Postfix or reverse polish notation.

For Stack manipulation the reverse polish notation is best suited. The reverse polish notation for the expression $A * B + C * D$ is $AB * CD * +$

Conversion to Reverse Polish Notation

The conversion of an expression from infix notation to the reverse polish notation must be done according to the operational hierarchy that follows for infix notation. First perform all arithmetic operations inside the inner parentheses, then inside outer parantheses, then do multiplication and division operations and lastly the addition and subtraction operations.

www.IPUnation.com

Example 6.1. *Convert the infix expression $(x + y) * [z * (w + v) + s]$ into reverse polish notation.*

Solution. The two sub-expression $(x + y)$ and $(w + v)$ will solved first. Thus the postfix expression of these subexpression will be $xy+$ and $wv+$ respectively.

Now, in the square bracket z will be multiplied by $(w + v)$. Thus, the postfix of this multiplication is $zwv++$.

This multiplication result is then added to s will result in $zwv++s+$

Finally, $xy+$ and $zwv++s+$ will be multiplied together to get

$$xy+zwv++s++.$$

The procedure is shown again as

$$\begin{aligned}(x + y) * [z * (w + v) + s] &= xy+ * [z * wv++ s] \\ &= xy+ * [zwv++ + s] \\ &= xy+ * zwv++s+ \\ &= xy+ zwv++s++\end{aligned}$$

Example 6.2. *Convert the infix notation $A * B + A * (B * D + C * E)$ into RPN.*

Solution. $A * B + A * (B * D + C * E)$

$$\begin{aligned}&= AB* + A * (BD* + CE*) \\ &= AB* + A * BD* CE** \\ &= AB** + ABD* CE*** \\ &= AB*ABD* CE****\end{aligned}$$

Evaluation of Arithmetic Expression

Consider an expression $A * B + C * D$ in infix notation. Its reverse polish notation is $AB * CD * +$. This postfix expression will be evaluated as follows : scan the expression from left to right. Whenever an operator is found, perform the operation with the two operands on the left side of operator. Remove the operator and the two operands and replaced them by the result obtained by performing that operation. Continue in the same manner and repeat the procedure for every operator found until there are no more operators.

Thus, for the reverse polish notation $AB * CD* +$ first we find the operator $*$ and the two operands to the left of $*$ are A and B. Thus, we perform $A * B$ and replace A, B and $*$ by the product we get

$$(A * B) CD* +$$

The next operator is $*$ and the two operands to the left of $*$ are C and D. Thus, we perform $C * D$ and replace C, D and $*$ by the product, we get

$$(A * B)(C * D) +$$

The next operator is $+$ and the two operands to the left $+$ are the two products $(A * B)$ and $(C * D)$, hence the result obtained is

$$A * B + C * D.$$

Any arithmetic expression can be evaluated using stack as follows :

(i) Convert the given infix expression into its equivalent reverse polish notation.

(ii) Scan the expression from left to right.

(iii) While scanning when operands are found, pushed them into the stack as they appears.

(iv) When operators are found, pop two top most operands from the stack, perform the operation involving the operator and then pushed back the result into the stack.

(v) Continue, scanning the expression until there are no more operators.

(vi) Finally, the result of the expression will remain on the top of the stack.

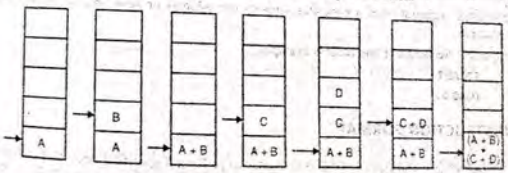


Fig. 6.7. Stack Operation to Evaluate (A + B) and (C + D).

To illustrate this, consider the expression $(A + B) * (C + D)$. In reverse polish notation this expression is $AB + CD + *$. The stack operation is shown in Fig. 6.7. The arrow (→) points to the top of the stack.

Uses of a Stack

(a) Temporary storage.
(b) Reversing strings.
(c) Evaluation of Reverse Polish Notation (RPN) expressions.
(d) Conversion of Infix to Reverse Polish Notation.
(e) Subroutine calls.
(f) Zero address instructions.

6.4. SINGLE ACCUMULATOR ORGANIZATION

An accumulator machine uses a single operational register, called an accumulator to hold the results of arithmetic, logical or shift operation. All operations are performed with an implied accumulator register. The instruction format in this type of computer uses on address field. For example, the instruction that specifies an arithmetic addition is defined by an assembly language instruction as

ADD X

where X is the address of the operand. The ADD instruction in this case result in the operation AC ← AC + M[X]. AC is the accumulator register and M[X] symbolizes the memory word located at address X.

The instruction do not explicitly reference the accumulator. The accumulator is an amplicit operand of each instruction and hence accumulator machine instructions are mostly 1 address instructions. In accumulator machines, LOAD instruction copy a value from memory to an accumulator and a STORE instruction copy the value from the accumulator to memory. Like

STORE T

⇒ M[T] ← AC.

Transfer the contents of accumulator register to a memory location designated by A.

LOAD A

⇒ AC ← M[A].

Transfer the contents of memory location designated by A to the accumulator register.

Accumulator organisation of a computer supports one address or zero address instructions. Zero address example.

Complement Accumulator one address example:

STORE T

LOAD A

6.5. INSTRUCTION FORMAT

An instruction is a command given to a computer to perform a specified operation on some given data and the format in which the instruction is specified is known as Instruction Format. The three common fields found in the instruction are :

(i) An operation code field that specifies the operation to be performed. It is known as opcode field.
(ii) An address field that designates the register address and/or a memory address.
(iii) A mode field that specifies the way the operands or the effective address is determined.

For example,

ADD *R1*, *R0*. ADD is the opcode and *R1*, *R0* are the address field.

Operations specified by computer instructions are executed on some data stored in memory or some registers. Operands residing on memory are specified by memory address and operands residing in processor registers are specified by register address. A register address is a binary number of K-bits that defines one of 2^K registers in the CPU. Thus, if a CPU has processor registers R0 to R15, then address of each registers will be of four bits. For example, the binary information 0101, is the address of register *R5*.

The instruction may be of several different lengths containing different number of addresses. The number of address fields in the instruction format of a computer system depends on the internal architecture/organization of registers. The different types of CPU organization are :

(1) General Register Organization (two and three-address instructions)
(2) Stack Organization (zero-address instructions)
(3) Single Accumulator Organization (one-address instruction)

6.5.1. General-Register Organization

It covers all the Two-address and Three-address instructions.

The instruction format in this type of computer needs two or three addresses. The number of addresses in the instruction can be reduced to two from three if the destination register is same as one of the source registers.

In two address instructions both operand address are specified. The result is placed in one of the specified addresses. In three-address instruction two addresses are specified for the two operands and one address of the result. Thus, general-register-type computers employ two or three address fields in the instruction format. Each address field may specify a processor register or a memory location.

Examples :

ADD *R1*, *R2*, *R3* $\qquad R1 \longleftarrow R2 + R3$

The above instruction contains three register addresses. The operation performed is the add operation between the content of processor register *R2* and *R3* and result is to be placed into the destination register *R1*.

ADD *R1*, *R2* $\qquad R1 \longleftarrow R1 + R2$

The above instruction consists of only two register addresses. *R1* and *R2* are source registers where *R1* also serves the purpose of destination register. The instruction specifies the add operation between the contents of *R1* and *R2* and result to be stored into *R1*.

MOV *R1*, *R2* $\qquad R1 \longleftarrow R2$

Mnemonic MOV is used for transfer instruction. The instruction contains only two register address *R1* and *R2* where *R2* is the source register and *R1* is the destination. Thus, in transfer-type instruction only two addresses are required. The instruction specifies move the content of *R2* into register *R1*.

Add *R1*, X $\qquad R1 \longleftarrow R1 + M[X]$

This instruction has two address field, *R1* the register address and *X* a memory address.

6.5.2. Stack Organization

It covers all the zero-address instructions.

Stack-oriented machines do-not contain any accumulator or general-purpose registers. Com with stack organization have PUSH and POP instructions which requires an address field. Thu instruction

PUSH X $\quad$ TOP $\longleftarrow M[X]$

will push the word/data at address X to the top of the stack. The SP is automatically updated. The ope instruction does not contain any address field because the operation is performed on two top most operan the stack. For example,

ADD

The instruction ADD consist of only operation code with no address field. This instruction p top two operands from the stack, add the numbers and then PUSH the result into the stack.

6.5.3. Accumulator-Type Organization

It covers all the one-address instructions.

All operations are performed with an implied accumulator register. The instruction format con address field, *i.e.* only one operand address is specified in the instruction. The other operand is a accumulator. The result is placed in the accumulator.

For example,

ADD X. $\quad$ $AC \longleftarrow AC + M[X]$

The ADD X instruction means add the content at memory location X, symbolizes as $M[X]$, with content of accumulator. Thus, the previous value of accumulator will be lost and the accumulator the result of above instruction.

www.ITCHOR.com

Example 6.3. *To show how the number of address affects a computer program, we will evaluate arithmetic statement*

$$X = (A + B) * (C + D)$$

using three, two, one or zero address instruction.

Solution. ADD, SUB, DIV and MUL are used for arithmetic operations, MOV for the transfer operat LOAD and STORE for transfers to and from memory and AC register. We will assume that the operands a memory addresses A, B, C and D and the result must be stored in memory address X. $R1$, $R2$ are the register T is the address of temporary memory location used to store intermediate result.

Three-Address Instruction

ADD $R1, A, B$	$R1 \longleftarrow M[A] + M[B]$
ADD $R2, C, D$	$R2 \longleftarrow M[C] + M[D]$
MUL $X, R1, R2$	$X \longleftarrow R1 * R2$

The symbol $M[A]$ denotes the operand at memory address symbolized by A.

The advantage of the three-address format is that it results in short programs when evaluat arithmetic expressions.

Two-Address Instruction

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

One-Address Instruction

One address instruction use an accumulator (AC) register for all data manipulation.

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

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

Zero-Address Instruction

To evaluate arithmetic expression for zero-address machine, the expression must be in reverse-polish notation. Also, the instructions like ADD, MUL does not requires an operand field. It simply pop-up the two top most operands from the stack, perform the operation and places the result on the top of the stack. However, PUSH and POP instructions requires an address field to specify the operand that communicates with the stack. TOS stands for top of stack.

The reverse polish notation of expression.

$$X = (A + B) * (C + D) \text{ is evaluated as}$$
$$= (AB+) * (CD+)$$
$$= AB + CD + *$$

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

Example 6.4. *Evaluate the following arithmetic expressions using 3-address, 2-address, 1-address and 0-address instructions to show the effect on computer program.*

$$X = (A + B) * (C - D)$$

Solution. In the above arithmetic expression, variables A, B, C and D are given. We will assume that the operands are stored at memory addresses, A, B, C and D. The result will be stored at memory address X. In the program, ADD SUB and MUL are used for arithmetic operations. MOV is used for the transfer operations. LOAD and STORE are used for transfers to and from memory and AC register. R1, R2 are general purpose registers and T is the address of tempory memory location used to store intermediate result.

Three-Address Instructions

ADD R1, A, B	R1 ⟵ M[A] + M[B]
SUB R2, C, D	R2 ⟵ M[C] − M[D]
MUL X, R1, R2	M[X] ⟵ R1 * R2

Two-Address Instruction

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

www.IPUnationOnline.com

One-Address Instruction

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

Zero-Address Instruction

Given expression is

$$X = (A + B) * (C - D)$$

Reverse polish notation of the given expression is evaluated as

$$X = AB + * CD -$$
$$= AB + CD - *$$

PUSH A	TOS ⟵ A

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

6.6. INSTRUCTION SET BASED CLASSIFICATION OF PROCESSORS

An instruction set is a list of all the instructions, and all their variations, that a processor can execute. The important aspect of the computer architecture is the design of instruction set for the processors. The instruction set being chosen for a particular computer system determines and decides the way that machine language programs are constructed. The instruction set varies from time to time, earlier computer were having small and simple instruction set where it needs simple and minimized hardware to implement but by the time the instruction set has become large and complex which degraded the expected level of performance.

The classification of the instruction set of the processor includes the two types *i.e.*,

- RISC (Reduced Instruction Set Computer)
- CISC (Complex Instruction Set Computer)

RISC is a computer that was used in early 1980s. The instructions used in RISC were fewer with less constructs because this helps to execute the instructions in a fast manner, within the CPU without having to use much memory whereas a CISC is a computer with large number of instructions mainly in hundreds and are still in use.

6.6.1. Reduced Instruction Set Computer (RISC)

RISC machines can be considered as a renaissance in the architectural evolution of computers. It has changed the conventional approach in the design of CPU architecture followed by computer architects of those days. A RISC machine is simply a computer with less number of vertical microinstructions. In RISC user programs are compiled into the sequences of these microinstructions and then directly executed by the hardware without intervening interpreter. For a RISC machine the instructions generated by a compiler are directly executed by the hardware and even they are not interpreted by the micro code.

The secret behind the speed of a RISC machine is the elimination of this level of interpretation, as a result of which the simple things such as adding of two registers by a program can now be completed in a single microinstruction. It is a fact that, before the invention of microprogrammang, all the computers were RISC machines with simple instructions, directly executed by the hardware.

RISC instruction set contains less than 100 instructions with a fixed instruction format of 32 bits. Only 3 to 5 simple addressing modes are used. Here most instructions are executed in one cycle with hardwired control. The instructions are mostly register based. Memory access is done by load/store instructions only. A large register file of at least 32 has been used to improve fast context switching among multiple user.

Since the complexity of the instruction set has been greatly reduced, the entire processor can be implemented on a single VLSI chip. As a result, the benefit of a higher clock rate and a lower CPI can be

obtained. These again naturally led to higher MIPS ratings as found on commercially available RISC Superscalar Processors.

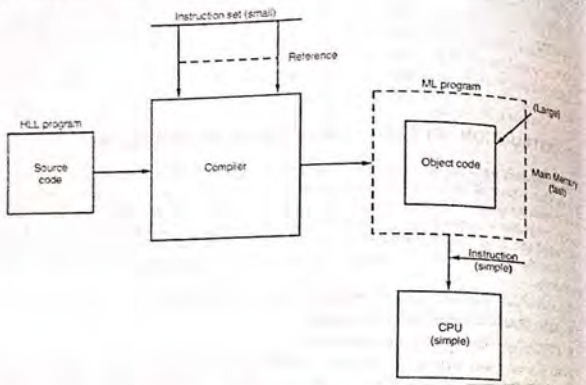


Fig. 6.8. RISC Scenario

www.IPUnation.com

RISC machines are fast due to the advancement of software and not in hardware. The improvement in optimizing compiler technology made it possible for RISC to generate microcode up to the level of handwritten microcode. Nowadays, it is possible to let the compiler produce the microcode directly, skipping the interpreter. RISC machine are somewhat simpler than even vertical micro architectures. The RISC design has further gained power by pushing some of the less frequently used operations in software.

Now a days the RISC chips are prevailed over the market for 32-bit embedded systems. The smaller RISC chips are becoming common even in the cost-sensitive 8-bit embedded system market. The main market for RISC CPUs has been the system that require low power or small size and low cost. In other words, a fast RISC chip is much cheaper than fast CISC chip.

6.6.1.1. RISC Characteristics

The major RISC characteristics are given below:

1. It has few instruction types and less addressing modes.
2. It has fixed and easily decoded instruction formats.
3. Most instructions are executed in one cycle with hardwired control.
4. Fast single-cycle instruction execution.
5. Memory access is limited mainly to load and store instructions.
6. It uses compilers to optimize object code performance.

7. All operations are done within the registers of CPU.

Since complex instructions are not in the instruction set and cannot be microcoded, therefore they must be implemented by routines at the instruction level which prompts the attention of efficient compilation. A machine code generated after compilation for a RISC computer have more instructions than for the corresponding CISC code. Several of the RISC attributes mentioned above are closely related. For example the small size and regularity of the instruction set simplifies the design of a hardwired program control unit, which in turn facilitates the achievement of fast single-cycle execution. The stress and thrust placed on sufficient compilation requires the machine architects and compiler writes to cooperate closely in the design process. Figures 6.9 given below shows the relationship between different categories of processor.

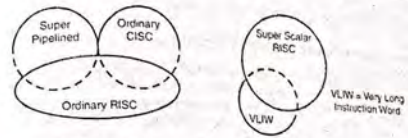


Fig. 6.9. Block diagram of different categories of processors

6.6.1.2. RISC Instruction Format and Design Issue

The basic instruction format used in RISC machine contains :

(a) 7-bit opcode.

(b) Two 5-bit registers namely DEST and SOURCE.

(c) A mode bit (I)

I = 0, not immediate

I = 1, immediate

(d) A condition code bit (C)

C = 0 Do not set condition code

C = 1 Set condition code.

Bits	7	1	5	5	1	13
	OPCODE	C	DEST	SOURCE	I	OFFSET

Fig. 6.10. Generalized RISC instruction format

Now considering the case of a simple ADD instruction, the operands depend on the I-bit. If I = 0, this is Register addressing. Now one operand is taken from the source register, the second operand is taken form the register specified by the low-order 5-bits of the OFFSET field and the result is stored in destination register. If I = 1, the second operand is a 13-bit constant, giving Immediate Addressing.

There are basically three addressing modes for RISC namely Indexex Addressing, Register Indirect, Register Direct and other modes can be constructed at run-time by building an address in a register and then using Register Indirect or Indexed Addressing.

6.6.1.3. RISC Drawback

As there are certain advantages of RISC, it has some disadvantages too.

1. A RISC processor lacks some sophisticated instructions found in CISC.
2. The presence of hardwired control appeared to be another short coming of RISC, since it is less flexible and more error prone.
3. Increase in RISC program length results in more instruction traffic and greater memory demand that drops RISC from favour.
4. RISC uses a large register file which can also cause problems.
5. In case of RISC it is very difficult to determine the optimal sizes of Register set I-cache and D-cache, which directly deals with performance of RISC.

6.6.2. Complex Instruction Set Computer (CISC)

The computer of the earliest time were very simple in their design. The high level languages supported by these computers and used by the users were also simple with the release of IBM 360 series the concept of microprogramming evolved. The instruction set used for these processors were designed not only to manipulate simple binary numbers, but for text, scientific floating point and the binary coded decimal arithmatic needed by accounting systems as well.

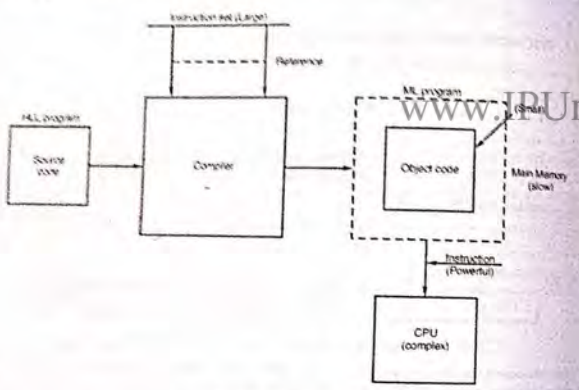


Fig. 6.11. CISC Scenario

As a result of which the instruction set became highly complex and so the micro program became equally complex. With the inclusion of microprogramming, the programming became easier, because a programmer could remember just 30 to 100 instructions, and a set of 3 to 10 addressing modes rather than

thousands of distinct instructions. This was called an "orthogonal instruction set". This strategy is now called as "complex instruction set computer" or CISC.

The slower speed of the main memory compared to the CPU one again encouraged the spread of CISC as the way to reduce frequent memory references which is due to to repeated call to library procedures. The temptation to inject this library procedure into the microprogramming by addressing few routines in the fast ROM inside the CPU was irresistible. As a result, the speed of the operation became faster but the complexity of micro program naturally increased. But the inclusion of all kinds of instructies resulted in the machine language getting larger and more complex, the micro program gets bigger and hence runs slower. So more the instruction, more is the time spent in decoding the op codes. At the same time, a large number of addressing mode means that the process of analyzing the address now can no longer be carried out in line because of the fact that the same lengthy microcode would have to be repeated 100 of times in the micro program.

Thus, micro procedures are needed and almost very instruction must call a micro procedure to analyze the addressing mode. Moreover, all kinds of instructions and modes that have been added are found to be hardly ever used in practice.

6.6.2.1. CISC Characteristics

The major CISC characteristics are :

1. It has a large number of instructions typically in hundreds, ranging form 100 to 350.
2. Some instructions that perform specialized task are seldom used.
3. It has different types of instruction and data formats of variable length.
4. It has wide variety of available addressing mode-typically ranging from 5 to 20 different types.
5. It has a small set of 8 to 24 general-purpose registers (GPRs).
6. Many high level language statements are directly implemented in hardware/firmware.
7. The translation of a program from high level language to low level language is done by compilation. Complex instruction set allow to simplify compilation and improve overall computer performance. The translation is simplified if there are instructions that implements the statement directly.
8. The essential goal of a CISC architecture is to provide single machine instruction for each statement that is written in high level language.

6.6.2.2. CISC Drawback

Certain CISC drawbacks are listed below :

1. With the addition of more and more instruction the machine language become more complicate.
2. With the machine language getting complicated the micro program gets bigger and hence runs slower.
3. More instructions in the instruction set means more time spent in decoding the opcodes.
4. In CISC the address analysis is very difficult to carry in-line, because the same lengthy microcode would have to be repeated hundred of times in the micro program.
5. All kinds of instructions and modes that have been added are found to be hardly ever used in practice.

www.JRUnation.com

6.6.3. Comparison between RISC and CISC

The RISC and CISC are two different computer architectures. RISC machines are somewhat simpler than CISC. However, the fundamental differences between the two machines are listed below :

1. Instruction Format: RISC has fixed format instructions while CISC uses variable format instructions.

2. Instruction Set: RISC has small set of instructions while CISC has a large set of instructions.

3. Instruction Mode: RISC has a few instructions and only 3 to 5, limited instruction mode. While CISC has many instructions and a variety of instruction mode usually 12 to 24.

4. Pipelining: RISC architecture is highly pipelined while CISC machines and not pipelined at all.

5. Role of CPU Hardware: The instructions available in RISC are execute directly by the hardware control while CISC's instruction are firstly interpreted by the corresponding microprogram and then executed accordingly.

6. Clock rate and CPI:

- Clock rate in most RISC machines lies between 50-200 MHz while that of CISC machine lies between 33-66 MHz.
- CPI in case of RISC is 1.5 where as the CPI of CISC is in between 2–15.

7. Memory Reference: RISC has only loads/stores instruction and nothing else to reference memory to obey its criteria to execute it in one cycle, whereas an instruction in CISC, in this regard can reference the memory.

8. Register Set: RISC requires multiple register sets, large number of general purpose registers (GPR's) where as CISC requires single register set of (8-24) GPRs.

Table 6.3. Distinct Features of RISC and CISC

www.IPUnation.com

Differentiating Factor	*Reduced Instruction Set Computers (RISC)*	*Complex Instruction Set Computers (CISC)*
Instruction Set Size	Small set of instructions having mostly register based instructions	Large set of instructions
Instruction Formats	Instructions with fixed format. 32 bits per instruction	Instructions with variable formats. 6-64 bits per instruction
Addressing modes	Limited to 3-5	Normally 12-24
General-Purpose Registers (GPRs)	Large number of GPRs ranging from 32-192	Mostly ranges from 8-24 GPRs
Cache Design and Usage	Mostly use of split caches, separate caches for instructions and data	Use of unified cache for instructions and data. Recend trend to use split caches
Processor Speed (Clock Rate and CPI)	50-200 MHz with none cycle for almost all instructions and an average CPI ≤ 1.5	33-66 MHz and may be a little more in recent release and CPI mostly between 2 and 15
CPU Control Design	Almost all hardwired control without using control memory	Most are microcoded using control memory (ROM), but recent CISC also used hardwired control

6.7. ADDRESSING MODES

There are three fields in an instruction:

(i) Opcode.

(ii) Address field.

(iii) Mode field.

The Opcode field in the instruction represents the operation which is to be performed. The operation is performed on some data that is to be stored in computer registers or memory words. There are several ways to specify an operand. These are called addressing modes. The way the operands are chosen during the execution of a program is depend on the addressing mode of the instruction. Addressing modes define a rule or a way by which the value of the operand is referenced or we reach where the value of operand is present. The operand can be present in processor register, memory, I/O port and in the instruction itself. Addressing modes define a technique to each where the operand that is required in our instruction is present.

Computers use addressing techniques for the purpose of accommodating one or both of the following provisions :

(i) To give programming versatility to the user by providing facilities such as pointer to memory, Counters for loop control, indexing of data and program relocation.

(ii) To reduce the number of bits in the addressing field of the instruction.

To understand the various addressing modes, it is imperative that we understand the basic operation cycle of the computer. An instruction cycle is divided into three major phases.

(i) Fetch the instruction from memory.

(ii) Decode the instruction.

(iii) Execution the instruction.

There is one register PC (Program Counter) that keeps track of the instructions in the program stored in memory. PC holds the address of the next instruction that is to be executed. Whenever an instruction is fetched from memory. PC is incremented by 1. The second phase of the machine cycle determines the operation that we have to performed the addressing mode of the instruction and the location where the operation is present. Then, we execute the instruction and returned to step 1 to fetch the next instruction from memory. The availability of the addressing modes gives the experienced assembly language programmer flexibility for writing program that are more efficient with respect to the number of instructions and execution time.

Now discussing the address modes following standard terms are used :

(i) **Address (A):** Contents of an address field in the instruction that refers to a memory location.

(ii) **Register (R):** Contents of an address field that refers to a register in the instruction.

(iii) **Program Counter (PC):** The Program Counter (PC) keeps tracks of the instructions in the program stored in memory. It holds the address of the instruction that is to be executed and incremented every time when an instruction is fetched from memory.

(iv) **Effective Address (EA):** The effective address is the address of the operand when the operand is actually stored. It is defined to be the memory address obtained from the computation dictated by the given addressing mode.

Table 6.4. Different Addressing Modes

S. No.	Addressing Mode	Mechanism	Remarks
1.	Immediate	Operand is present in the instruction itself	Fast Operand fetch
2.	Direct	Operand is in memory; its location address is given in the instruction	One memory access required for operand fetch
3.	Indirect	Operand is in memory; its address is also in memory; address of the location containing operand address is given in the instruction	Additional memory access required for knowing the operand address
4.	Register direct	Operand is in a register; the register address (number) is given in the instruction	Quickest access of operand without any memory access; faster than direct addressing
5.	Register indirect	Operand is in memory; its address in a register; address (number) of the register containing the operand address is given in the instruction	Faster than indirect addressing
6.	Relative addressing	The operand is in memory; its relative position (offset number) with respect to the contents of program counter is given in the instruction	Quick address calculation without memory access.
7.	Base register addressing	The operand is in memory; its address is specified in two pars; the instruction gives in two parts; the instruction gives an offset number and also specifies the base register; the offset number is added to the base register contents.	
8.	Index addressing	The operand is in memory; the instruction contains an address and the index register contains offset number; the address and the offset are added to get the operand address	

6.7.1. Implied Addressing

In this type of addressing mode operands are defined implicitly as a part of definition of the instruction itself.

For example, increment accumulator instruction.

Complement accumulator instruction are implied mode instructions because the operand in the accumulator register is implied in the definition of the instruction. All register reference instructions that use accumulator are implies mode instruction Zero addrss instructions in stack organised computers are implied mode instructions since it is implied that operand are present on the top of the stack.

6.7.2. Immediate Addressing

This is the simplest form of addressing modes. The operand itself is a part of instruction. In other words, we can say that an immediate mode instruction contains an operand field rather than an address field.

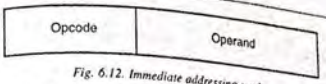
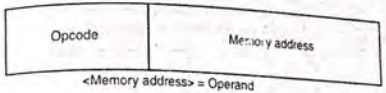


Fig. 6.12. Immediate addressing mode

Since the operand itself is given in the instruction called immediate addressing mode

The merit of immediate addressing is that no memory reference other than the instruction fetch is required to obtain the value of the operand. So it saves one memory cycle in the instruction cycle. The demerit is that the size of the number is restricted to the size of address field.

6.7.3. Direct Addressing



<Memory address> = Operand

Fig. 6.13. Direct addressing

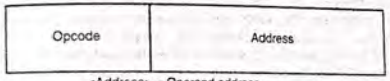
Since the operand address is explicitly given in the instruction, this mode is called direct addressing mode

Merits: Since the operand address is directly available in the instruction, there is no need for the operand address calculation step. Hence, instruction cycle is reduced.

Demerits: The number of bits for the operand address is limited by the operand field in the instruction.

www.IPUnation.com

6.7.4. Indirect Addressing



<Address> = Operand address

Fig. 6.14. Indirect addressing

The instruction gives an address of a location (X) which, in turn conains another location's address (Y), which actually contains the operand. It can be represented as

<X> = Y, <Y> = Operand

Merits: Flexibility in programming; changing the address during program run-time without changing the instruction contents. It is useful for implementing pointers (C language).

Demerits: Instruction cycle time increases since two memory accesses are required even for a single level indirect addressing.

6.7.5. Register Addressing

In this mode the operands are in registers that reside within the CPU. It is similar to direct addressing. The only difference is that the address field refers to a register rather than a main memory address. (Fig 6.15)

Merits :

(a) Only a small address field is needed.

(b) No memory reference is required. The memory access time for a register is much less than the a main memory address because registers are internal to CPU.

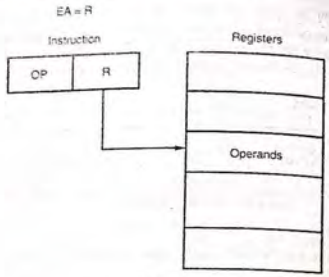


Fig. 6.15.

Demerit : Address space is very limited. Since number of registers are limited.

6.7.6. Register Indirect Addressing

www.IPUnation.com

Register indirect addressing is similar to indirect addressing. In this case address field refers to register and in indirect addressing address field refers to a memory location. The instruction specifies register in the CPU whose contents give the address of the operand in memory. When there is indirect register addressing, the program generates the address of the operand and places it in operational register for later use. A reference to the register is then equivalent specifying a memory address.

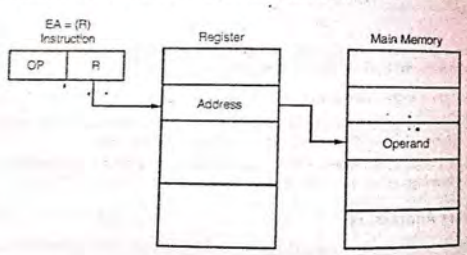


Fig. 6.16.

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

6.7.7. Relative Addressing

In this technique contents of Program Counter (PC) is used as a base address. Contents of program counter is added with the address part of the instruction in order to get the effective address. The address part of the instruction is a signed number. When this number is added to the contents of the program counter register, the result produces an effective address whose position in memory is relative to the address of the next instruction. This addressing is common for branch instructions and sometimes for immediate data but this technique is considered absolute for data addressing. We have to be take care that when we are adding the contents of the program counter then we have to see that value in the decode phase of the instruction cycle. Because when an instruction is fetched from memory, the value of program counter is updated.

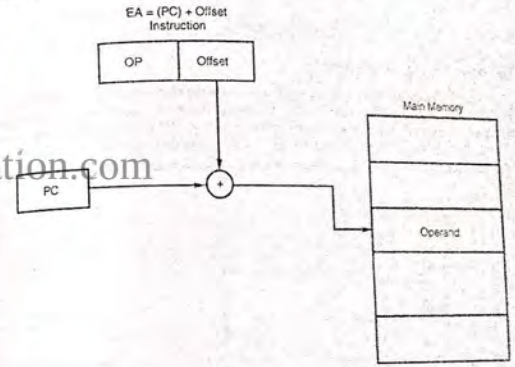


Fig. 6.17.

6.7.8. Indexed Addressing

In this addressing mode the contents of an index register is added with the address part of the instruction to obtain the effective address. An index register is a special CPU register that contains an index value. The distance between the beginning address and the address of the operand is the index value stored in the index register.

$$EA = (XR) + Offset$$

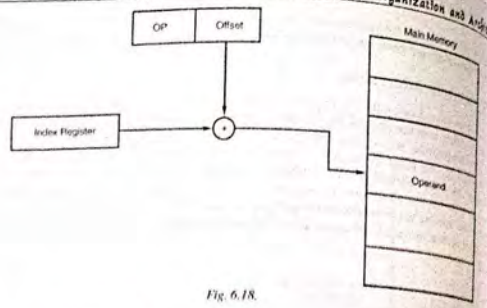


Fig. 6.18.

6.7.9. Base Register Addressing

In this mode the contents of the base register is added with the address part of the instruction to obtain the effective address. It is same as index addressing except now the register is a base register index register in assumed to hold an index number that is relative to the address part of the instruction. The base register may be a general purpose register or a general purpose register set/machine.

The Effective Address (EA) of the operand is then sum of the displacement and the content of the register.

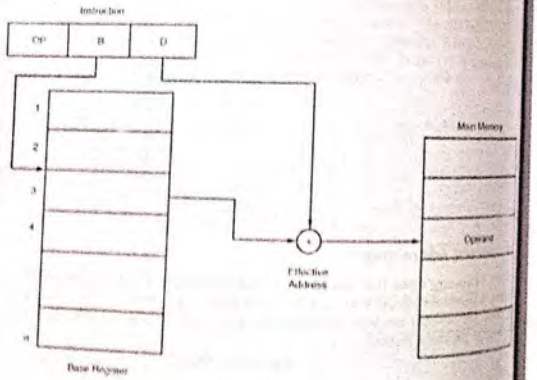


Fig. 6.19.

$$EA = (B) + D$$

Each base register can point to any location in memory.

6.7.10. Auto Increment and Auto Decrement Addressing

It is special addressing mode that automatically increment or decrement that index register. When the address stored in the register refers to a table of data in memory. It is necessary to increment or decrement the register after every access to the table. The address field of an instruction is used by the control unit in the CPU to obtain the operand from memory.

The effective address is defined to be the memory address obtained from the computation dictated by the given addressing mode. The effective address is the address of the operand in a computational type instruction. A programmer can use auto increment addressing to sequently access the characters in a string or the elements in a vector list. The auto increment addressing mode increments the register address before using it and the auto decrement addressing mode decrements the address after using it. That is exactly how typical computers manipulate stack registers.

Example 6.5. *To show the difference between various addressing modes, we will show the effect of the addressing modes on the instructions defined in Fig. 6.20. The instruction is written at the address*

CPUnation.com

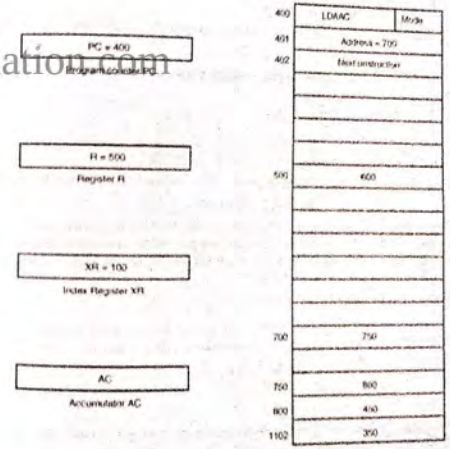


Fig. 6.20

400 and 401. Address field of the instruction is 700. The first word of the instruction specifies the operation code and mode, and the second word specifies the address part. PC has the value 400 before fetching the instruction. After the execution of the instruction accumulator register will receive the value.

Solution. Now for each addressing mode technique we will calculate the Effective Address (EA).

(*i*) **Direct addressing mode.** Address that is mentioned in the instruction will contain the value of the operand.

So, Effective Address = 700

Contents of AC 750

(*ii*) **Immediate addressing mode.** Value of the operand is a part of instruction itself.

So Contents of AC = 700

Effective Address = 401

(*iii*) **Indirect Addressing mode.** Two memory references are required. So when we access 700 location we will get again an address that is 750. When we access 750 then we will get the value of the operand.

So, Effective Address = 750

Contents of AC = 500.

(*iv*) **Register direct addressing mode.** Contents of register R provides the value of the operand.

So, Contents of AC = 500.

(*v*) **Register Indirect addressing mode.** Contents of register R provides the effective address of the operand.

So, Effective address = 500

Contents of AC = 600.

(*vi*) **Relative addressing mode.**

Effective address = Contents of program counter (PC) + Address that is part of the instruction

$$EA = 402 + 700 = 1102$$

Here we are taking the value of program counter as 402. Because addressing mode is determined in the decode phase of the instruction cycle. Value of program counter is updated when an instruction is fetched from memory, so PC is updated as 402. Since 400 and 401 are the location when the instruction is written and next instruction is at address 402.

So, Effective address = 1102

Contents of AC = 350.

(*vii*) **Indexed addressing mode.** In the index mode the effective address is XR + 700 = 100 + 700 = 800 and the operand is 450.

So, Effective address = 800

Contents of AC = 450

Table 6.5 lists the values of the effective address and the operand loaded into the AC for all the addressing modes.

Table 6.5. Tabular list of examples

Addressing Mode	Effective Address	Contents of AC
1. Direct Addressing	700	750
2. Immediate	401	700
3. Indirect Addressing	750	800
4. Register Direct	—	500
5. Register Indirect	500	600
6. Relative	1102	350
7. Indexed	800	450

SOLVED EXAMPLES

Example 6.6. *A two word instruction is stored in memory at an address designated by symbol W. The address field of the instruction (stored at W + 1) is designated by the symbol Y. The operand used during the execution of the instruction is stored at an address symbolized by Z. An index register contains the value X. State how Z is calculated from the other addresses if the addressing mode of the instruction is*

(a) Direct *(b) Indirect*

(c) Relative *(d) Indexed*

Solution. According to question, the memory figure is shown in Fig. 6.21.

www.IPUnation.com

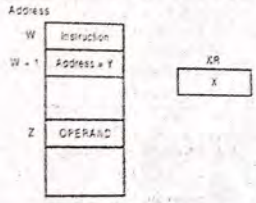


Fig. 6.21. Memory Context.

(*a*) In direct addressing mode the address part of the instruction is the address of the operand.

Thus,

$$Z = Y$$

(*b*) In indirect addressing mode the address part of the instruction gives the memory address, what is the effective address of the operand II.

Thus,

$$Z = M[Y]$$

(*c*) In case of relative addressing mode, the effective address is the address part of the instruction (Y) plus the content of program counter. The instruction is stored at W with address part at W + 1. Thus, when instruction is fetched from memory the PC will increment and points to W + 2.

Solution. $A + \overline{B} + 1$ performs : $A + 2^n - B = 2^n + A - B$

adding B : $(2^n + A - B) + B = 2^n + A$

remove end-carry 2^n to obtain A.

Example 8.5. *Show the step-by-step multiplication process using Booth algorithm (as in Table 8.3) when the following binary numbers are multiplied. Assume 5 bit registers that hold signed numbers. The multiplicand in both cases is +15.*

(a) (+15) × (+13)

(b) (+15) × (−13)

Solution. $(+15) \times (+13) = +195 = (0011000011)_2$

BR = 01111 (+15); $\overline{BR} + 1$ = 10001 (−15); QR = 01101 (+15)

Q_n	Q_{n+1}		AC	QR	Q_{n+1}	SC
		Initial	00000	01101	0	101
1	0	Subtract BR	10001			
			10001			
		ashr	11000	10110	1	100
0	1	Add BR	01111			
			00111			
		ashr	00011	11011	0	011
1	0	Subtract BR	10001			
			10100			
		ashr	11010	01101	1	010
1	1	ashr	11101	00110	1	001
0	1	Add BR	01111			
			01100			
		ashr	00110	00011	0	000
			+ 195			

(b) $(+15) \times (-13) = -195 = (1100111101)_{2's\ cos}$

BR = 011111 (+15); $\overline{BR} + 1$ = 10001 (−15); QR = 10011 (−13)

Q_n	Q_{n+1}		AC	QR	Q_{n+1}	SC
		Initial	00000	10011	0	101
1	0	Subtract BR	10001			
			10001			
		ashr	11000	11001	1	100
1	1	ashr	11100	01100	1	011
0	1	Add BR	01111			
			01011			
		ashr	00101	10110	0	010
0	0	ashr	00010	11011	0	001
1	0	Subtract BR	10001			
			10011			
		ashr	11001	11101	1	010
			−195			

□□□

Memory Organization

9

The memory unit is an essential component of any digital computer system since it is needed for storing programs and data. Memory unit is capable of storing the programs and the data that are needed to perform a particular task. A CPU should have rapid, uninterrupted access to these memories so that it can operate at or near its maximum speed. The goal of every memory system is to provide adequate storage capacity with acceptable level of performance and cost. There is not enough space in one memory unit to accomodate all the programs used in a typical computer, not all the accumulated information is required by the processor at the smae time. Therefore, it is more economical to use low-cost storage devices to serve as a backup for storing the information that is not currently used by the CPU. Following are the some criteria on which we decide which memory unit is to be used.

1. Cost
2. Speed
3. Memory access time
4. Data transfer rate
5. Reliability
6. Memory cycle time.

In this chapter, we will discuss about various memory device organisation. There are various types of memories some provides good speed but they are costly, some provide less speed but they are cheaper so we discuss the organisation of various memory units.

9.1. MEMORY HIERARCHY

Memory in a computer system is required for storage and subsequent retrieval of the instructions and data. A computer system uses variety of devices for storing these instructions and data which are required for its operations. Normally we classify the information to be stored on computer in two basic categories: Data and the Instructions.

"The storage devices along with the algorithm or information on how to control and manage these storage devices constitute the memory system of a computer."

A memory system is a very simple system yet it exhibits a wide range of technology and types. The basic objective of a computer system is to increase the speed of computation. Likewise the basic objective of a memory system is to provide fast, uninterrupted access by the processor to the memory such that the processor can operate at the speed it is expected to work.

But does this kind of technology where there is no speed gap between processor and memory speed exist? The answer is yes they do, but unfortunately as the access time (time taken by CPU to access a location in memory) becomes less and less the cost per bit of memory becomes increasingly higher. In addition, normally these memories require power supply till the information need to be stored. Both these things are not very convenient, but on the other hand the memories with smaller cost have very high access time which will result in slower operation of the CPU. Thus, the cost vs access time anomaly has lead to a hierarchy of memory where we supplement fast memories with larger, cheaper, slower memories. These memory units may have very different physical and operational characteristics, therefore, making the memory system very diverse in type, cost, organisation, technology and performance. This memory hierarchy will work only if the frequency of access to the slower memories are significantly less than the faster memories.

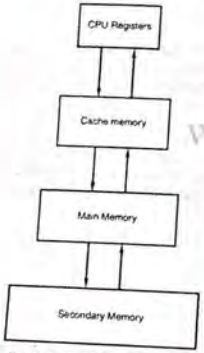


Fig. 9.1. The Memory Hierarchy

Thus, a memory system can be considered to consist of three group of memories. These are:

(a) **Internal Processor Memories:** These consist of the small set of high speed registers which are internal to a processor and are used as temporary locations where actual processing is done.

(b) **Primary Memory or Main Memory:** It is a large memory which is fast but not as fast as internal processor memory. This memory is accessed directly by the processor. It is mainly based on integrated circuits.

(c) **Secondary Memory/Auxiliary Memory/Backing Store:** Auxiliary memory in fact is much larger in size than main memory but is slower than main memory. It normally stores system programs

(programs which are used by system to perform various operational functions), other instructions, programs and data files. Secondary memory can also be used as an overflow memory in case the main memory capacity has been exceeded. Secondary memories cannot be accessed directly by a processor. First the information of these memories is transferred to the main memory and then the information can be accessed as the information of main memory.

There is another kind of memory which is increasingly being used in modern computers, this is called Cache memory. It is logically positioned between the internal memory (registers) and main memory. It stores or catches some of the content of the main memory which is currently in use of the processor. We will discuss about this memory in greater details in a subsequent section of this unit.

Before discussing more about these memories let us first discuss the technological terms commonly used in defining and accessing the memory.

9.2. CHARACTERISTICS/PARAMETERS OF VARIOUS MEMORY DEVICES

The following terms are most commonly used for identifying comparative behavior of various memory devices and technologies.

1. Storage Capacity: It is a representative of the size of the memory. The capacity of internal memory and main memory can be expressed in terms of number of words or bytes. The storage capacity of external memory is normally measured in terms of bytes.

2. Unit of Transfer: Unit of transfer is defined as the number of bits read in or out of the memory in a single read or write operation. For main memory and internal memory, the normal unit of transfer of information is equal to the word length of a processor. In fact it depends on number of data lines in and out of the memory module. (Why?) In general, these lines are kept equal to the word size of the processor. What is a word? You have already learnt about this term in Unit 1 of this block. The unit of transfer of external memory is normally quite large (Why? You will find the answer to this question later in this unit) and is referred to as block of data.

3. Access Modes: Once we have defined the unit of transfer next important-characteristics is the access mode in which the information is accessed from the memory. A memory is considered to consist of various memory locations.

The information from memory devices can be accessed in the following ways:

- **Random Access Memory (RAM):** It is the mode in which any memory location can be accessed in any order in the same amount of time. Ferrite and Semiconductor memories which generally constitute main memory are of this nature. The storage locations can be accessed independently and there exist separate access mechanism for each location.
- **Sequential Access:** On the other hand we have memories which can be accessed in a pre-defined sequences for example, the songs stored on a cassette can be accessed only one by one. The example of sequential access memory is Magnetic Tape. Here the access mechanism need to be shared among different locations. Thus, either the location or the read/write head or both should be moved to access the desired location.
- **Direct Access:** In certain cases the information is neither accessed randomly nor in sequence but something in between. In these kind of access, a separate read/write head exist for a track and on a track the information can be accessed serially. This semi-random mode of operation exist in magnetic disks.

4. Access Time: The access time is the time required between the request made for a read or write operation till the time the data is made available or written at the requested location. Normally it is

measured for read operation. The access time depends on the physical characteristics and access mode used for that device.

5. Permanence or Storage: Is it Possible to lose information by the memories over a period of time? If yes then what can be the reasons of the loss of information and what should be done to avoid it.

There are several reasons for information destruction, these are destructive readout, dynamic storage, volatility and hardware failure.

If for a particular memory the reading process destroys the stored information, we call it Destructive readout. In such memories the information has to be written back on the same location from which it has been read after each read operation. The reading process where the data is not destroyed on reading are referred to as Non-destructive readout.

There can be some memories where the stored 1 looses its strength to become 0 over a period of time. These kind of memories require refreshing. The memories which require refreshing are termed as dynamic memories. In contrast, the memories which do not require refreshing are called static memories.

Another factor which can destroy the contents is die presence and absence of electricity. The memories which loses their content on failure of power am armed as volatile memories, those which do not are called non-volatile. Magnetic memories are non-volatile and semiconductor main memories am volatile in nature.

6. Cycle Time: It is defined as the minimum time elapsed between two consecutive read requests is equal to access time? Yes, for most of the memories except the ones in which destructive readout is encountered. Cycle time for such memories is the access time (time elapsed when a read request is made available) plus writing time as after the data has been made available the information has to be written back in the same memory location as the previous value has been destroyed by reading. But for most of the commonly used semiconductor memories cycle time is equal to the access time.

7. Data Transfer Rate: The amount of information that can be transferred in or on of the memory in a second is termed as data transfer rate or bandwidth. It is measured in bits per second. Maximum number of bits that can be transferred in a second depends on how many bits can be transferred in or out of the memory simultaneously and thus the data bus width become one of the controlling factors.

8. Physical Characteristics: In this respect the memory devices can be categorised into four main categories viz. electronic, magnetic, mechanical and optical. One of the requirement for a gone device is that it should exhibit two well-defined physical states, such that 0 and 1 can be represented in those two states. The data transfer rate of the memory depends on the how quickly the state can be recognised and altered. The following table lists some of the memory technologies along with their physical and other important characteristics.

Technology	*Access time (in Sec)*	*Access mode*	*Permanence of storage*	*Physical nature of storage medium*	*Average cost (Rs/bit) (Approx.)*
Semiconductor memories	10^{-8}	Random	Volatile	Electronic	10^{-2}
Magnetic disk	10^{-2}	Direct	Non-volatile	Magnetic	10^{-5}
Magnetic tape	10^{-1}	Sequential	Non-volatile	Magnetic	10^{-6}
Compact disk ROM	1	Direct	Non-volatile	Optical	10^{-7}

Fig. 9.2. Characteristics of some memory technologies

Some of the main attributes which are considered for the storage devices are physical size, energy consumption and reliability.

The physical size also depends on the storage density of the memories. This is also coupled with the issue of portability of memory. The energy consumption is another key factor which determines the power consumption of the computer and cooling system requirements.

The higher is the power consumption, more costly equipment is required for internal cooling of computer. Reliability is measured as Mean Time To Failure. The storage devices which require mechanical motion e.g. hard disk are more prone to failure rather than the semiconductor memories which are totally electronic in nature. Very high speed semiconductor memories are also prone to failure as technology is moving towards its limits.

9. Cost: Another key factor which is of prime concern in a memory system is cost. It is normally expressed per bit. The cost of a memory system includes not only the device but the cost of access circuitry and peripherals essential for the operation of the memory. Figure 2 also shows per bit cost of these memories. Please note as the access time for memories is increasing, the cost is decreasing.

9.3. MAIN MEMORY

The main memory is the control storage device of a computer system. It is relatively large and fast memory that is used to store the data and programs that are currently needed by CPU. The principal technology behind the main memory is based on semiconductor integrated circuits. Integrated circuit RAM chips are available in two possible operating modes, static and dynamics. The main difference between SRAM and DRAMs is how their bit cells all constructed. The dynamic RAM provides reduced power consumption and larger storage capacity in a single dynamic RAM provides reduced power consumption and larger storage capacity in a single memory chip. The static RAM is easier to use and has shorter read and write cycles. Main memory is made up of RAM and ROM chips. RAM is used to store the programs and data that is to be changed while ROm is used for storing the programs that are permanently resident in the computer.

ROM portion of the main memory is needed for storing the initial program that is known as bootstrap program that runs on the computer when power is turned on. The contents of ROM remains constant when power is turned on and off. RAM and ROM chips are available in a variety of sizes. When memory requirement is large then we combine a number of chips to form the required memory size.

9.3.1. RAM and ROM Chips

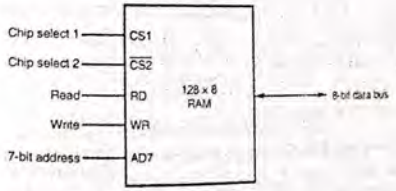


Fig 9.3. RAM chip

Figure 9.3 depicts a typical RAM chip. RAM chips are better suited for communicating with the CPU. It has bi-directional data bus that allows the transfer of data in both the direction. A bidirectional bus can be constructed using three state buffers. A three state buffer output can be placed in one of the possible three states. The capacity of RAM chip is 128 × 8 means there are 128 words each word is of 8 bits.

For 128 words we require an address line of 7 bits since $2^7 = 128$. The read and write inputs specify the memory operation and the two Chips Selects (CS) control inputs are for enabling the chip only when it is selected by the microprocessor. The function table is listed in Table 9.1.

Table 9.1. Function Table

CS_1	$\overline{CS_2}$	RD	WR	*Memory function*	*State of data bus*
0	0	×	×	Inhibit	High impedance
0	1	×	×	Inhibit	High impedance
1	0	0	0	Inhibit	High Impedance
1	0	0	1	Write	Input data to RAM
1	0	1	×	Read	Output data from RAM
1	1	×	×	Inhibit	High impedance

The unit is in operations only when $CS_1 = 1$ and $\overline{CS_2} = 0$. In this combination the memory can be placed in a write or read mode. When WR input is enabled, a byte is stored from the data bus into a location that is specified by the address input lines. When RD input is enabled, the content of the selected byte is placed into the data bus.

A ROM chip is organised in a similar manner. Read Only Memories (ROMs) are memory devices that the CPU can read but cannot write. Computers use them for holding permanent information, for holding the constant values that specify the system configuration. Since a ROM can only read, the data bus can only be in an output mode. The block diagram is shown in Fig. 9.4.

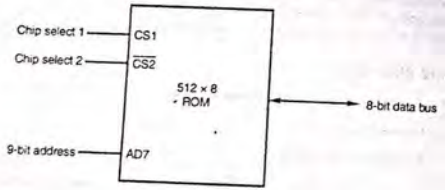


Fig. 9.4. Block diagram ROM chips

The nine address lines in the ROM chip specify anyone of the 512 bytes stored in it. The two chip select inputs must be $CS_1 = 1$ and $\overline{CS_2} = 0$ for the unit to operate.

9.2. Primary Memory

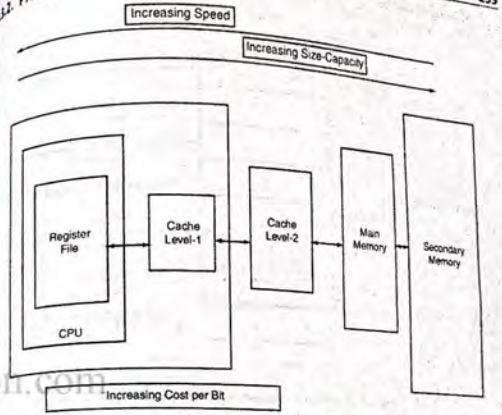


Fig. 9.5. Register File (CPU)

Internal or CPU Registers Memory

Every CPU has a register file or array-which is used by CPU as internal temporary memory for computation. therefore, sometimes this is called **internal scratch memory** of the CPU. A system programmer has access to this memory. Any register can be accessed by the system programmer for usage like system programming.

R/WM (Read Write Memory) or RAM Organisation

This memory is volatile, meaning that when the power is turned off, all the contents are destroyed. This is also called RAM (Random Access Memory) and also called user memory since user can store his program as well as data inside this. This type of memory can be subdivided into two main groups: Static RAM and Dynamic RAM. There are two types.This memory is volatile, meaning that when the power is turned off, all the contents are destroyed. This is also called RAM-Random Access Memory and also called user memory since user can store his/her program as well as data inside it.

Performance of Main Memory

(i) **Memory Access Time :** It is the delay or time between request generation and arrival of word in the CPU.

(*ii*) **Memory Bandwidth :** At what rate data is transferred between CPU and main memory. Higher is the rate better is your system.

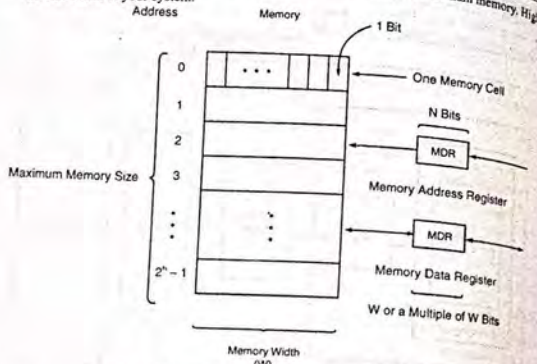


Fig. 9.6. Random Access Memory Addressing by CPU

www.IPLnation.com

Address is Represented in Hexadecimal or Octal

The term comes from the way data is accessed in this memory, and symbolizes the possibility to access any part of the memory at any given time. The contrary would be sequential memory, which mostly has a meaning in tape backup situations. Early computers also used drum memory, which was sequential (kind of) in nature.

(*i*) **SRAM – Static Random Access Memory**

(*ii*) **DRAM – Dynamic Random Access Memory**

- FP – Fast Page RAM
- EDO – Extended Data Output
- SDRAM – Synchronous DRAM

Static RAM Organization and Main Features

(*i*) It is made of flip-flops.

(*ii*) Low density, high power, expensive, fast.

(*iii*) Static: content will last "forever"(until lose power).

The term comes from the way data is accessed in this memory, and symbolizes the possibility to access any part of the memory at any given time. The contrary would be sequential memory, which mostly has a meaning in tape backup situations. Early computers also used drum memory, which was sequential (kind of). All computer memory modules used in computers today are of the RAM type. A RAM is organized typically as bytes to store data and its address is represented in HEX-Format. Any memory configuration can be fabricated using basic SRAM Cell and address decoder circuit.

Write : (*i*) Drive Bit Lines (Bit = 1, Bit = 0) (*ii*) Select Row; Read : (*i*) Precharge Bit and Bit to V_{dd} or 1/2 ⇒ Make Sure Equal! (*ii*) Select Row (*iii*) Cell Pulls One Line Low (*iv*) Sense Amp on Coloumn

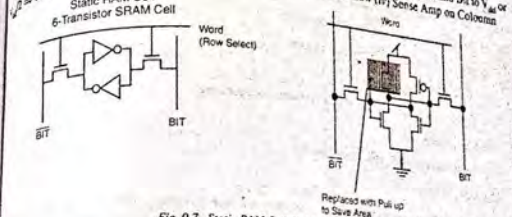


Fig. 9.7. Static RAM Cell With Read and Write

(*i*) Based on flip-flops and requires several transistors.

(*ii*) Fast access time and bits stored as on/off switches.

(*iii*) No charges to leak and No refreshing needed when powered.

(*iv*) Used in Cache memory design.

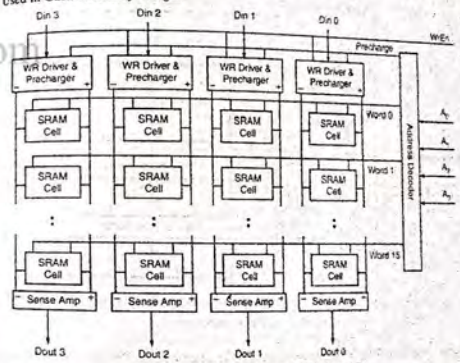


Fig. 9.8. SRAM Organisation 16-Works × 4 Bit

DRAM Organization

DRAM is made up of MOS Transistor Gates. It is a refreshing type memory. The difference between the two is that as long as power is maintained on the memory modules the DRAM will hold its information. The content of DRAM memory disappears from the memory within milliseconds, so in order to maintain

its data it has to be refreshed periodically. This makes the DRAM memory much slower than the SRAM. The computer memories you usually see are a form of DRAM, like SDRAM and DDR-SDRAM.

(*i*) Requires one capacitor and one transistor.

(*ii*) But needs refresh every few milliseconds.

(*iii*) Need refresh circuits.

(*iv*) Slower access time and bits stored as charge in capacitors.

(*v*) Charges may leak.

(*vi*) Simpler construction and smaller per bit.

(*vii*) Less expensive.

Dynamic RAM is constructed of tiny capacitors that leak electricity. DRAM requires a recharge every few milliseconds to maintain its data. Static RAM technology, in contrast, holds its contents as long as power is available. SRAM consists of circuits similar to the D flip-flops. SRAM is faster and much more expensive than DRAM; however, designers use DRAM because it is much denser (can store many bits per chip), uses less power, and generates less heat than SRAM. The different types of SRAM include asynchronous **SRAM**, synchronous **SRAM**, and **pipeline burst SRAM.**

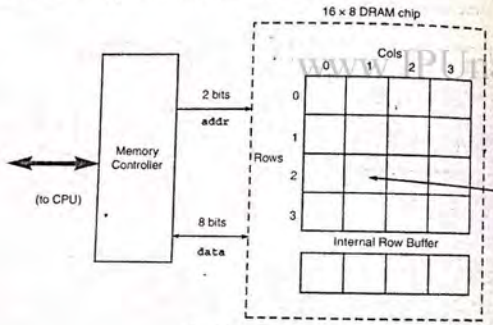


Fig. 9.9. DRAM Organisation

For these reasons, both technologies are often used in combination: DRAM for main memory and SRAM for cache. The basic operation of all DRAM memories is the same, but there are many flavors including Multibank DRAM (MDRAM), Fast-Page Mode (FPM) DRAM, Extended Data Out (EDO) DRAM, Burst EDO DRAM (BEDO DRAM), Synchronous Dynamic Random Access Memory (SDRAM), Synchronous-Link (SL) DRAM, Double Data Rate (DDR) SDRAM, and Direct RAMBUS DRAM.

Circuit Structure and Actual DRAM Chips (DRAM)

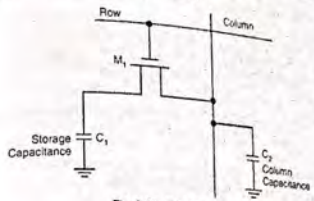


Fig. 9.10. DRAM Circuit

Read Only Memory (ROM)

Mask Programmable (Data "Written" During Device Fabrication")
Field Programmable (Data "Written" by Device Purchaser)
(*i*) PROM–Programmable Read Only Memory
(*ii*) EPROM–Erasable Programmable Read Only Memory
(*iii*) EEPROM–Electrically Erasable Programmable Read Only Memory

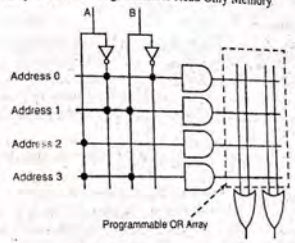


Fig. 9.11. PLD Architecture for PROM

PROM (Programmable Read-Only Memory)

It is a variation on ROM. PROM can be programmed by the user with the appropriate **Programming Device.** Whereas ROMs are hardwired, PROM has fuses that can be blown to program the chip. Once programmed, the data and instructions in PROM cannot be changed.

Flash Memory Applications and Limitations

The Flash memory was invented by Dr. Fujio Masuoka while working for Toshiba in 11084. According to Toshiba, the name 'Flash' was suggested by Dr. Masuoka's colleague, Mr. Shoji Arizumi, because the erasure process of the memory contents reminded him of a flash of a camera.

Flash memory is a form of EEPROM (Electrically Erasable Programmable Read-Only Memory) that allows multiple memory locations to be erased or written in one programming operation. In other terms,

it is a form of rewritable memory chip that, unlike a RAM (Random Access Memory) chip, power supply is not required to hold the contents. It is commonly used in memory cards, USB flash drives, MP3 players, digital cameras and mobile phones.

Fig. 9.12. Flash Memory Device

(*i*) High Density

(*ii*) Read/Write

(*iii*) Non Volatile – Data stored indefinitely without power

(*iv*) Combine best characteristics of traditional RAM/ROM.

Applications of Flash Memory

The Flash Memory technology has evolved into the preferred storage media for a variety of consumer and industrial devices.

1. Notebook computers and Personal computers
2. Digital cameras and Personal Digital Assistants
3. Cell phones and pagers and Global Positioning Systems (GPS)
4. Electronic musical instruments and Solid-state music players such as Television set-top boxes and MP3 players.

Limitations of Flash Memory

One limitation of flash memory is that although it can be read or programmed a byte or a word in a time in a random access fashion, it must be erased a "block" at a time. Starting with a freshly erased block, any byte within that block can be programmed. However, once a byte has been programmed, it cannot be changed again until the entire block is erased. In other words, flash memory (specifically NOR flash) offers random-access read and programming operations, but cannot offer random-access rewrite or erase operations.

(*i*) MRAM - MAGNETIC RAM

(*ii*) Infineon, IBM show prototype 16Mb magnetic RAM. Prototype MRAM chip brings power-saving technology one step closer

(*iii*) MRAM is expected to first be used in mobile applications as a replacement for flash.

(*iv*) At some point in the future, MRAM could also replace SRAM and DRAM, which is cheap to manufacture but is slower than MRAM and requires a constant power supply to retain stored data.

(*v*) Using MRAM instead of DRAM in notebooks PCs could help extend battery life, according to Infineon and IBM.

Types of ROM

Memory Type	*Write Time*	*Order of Read Time*	*Number of Write Cycles allowed*
ROM	Once	Nano seconds	ONE
PROM	Hours	Nano seconds	TENS
EPROM	Minutes (including time of erdsure)	Nano seconds	HUNDREDS
EEPROM	Milliseconds	Nano seconds	THOUSANDS

Fig. 9.13. Features of Read Only and Read Mostly memories

9.4. HIGH SPEED MEMORIES

The Need: Why the high speed memories? Is the main memory not a high speed memory? The answer to this question is definitely "No", but why so, well for this we have to go to the fundamentals of semiconductor technology which is beyond the scope of the unit. Then, if the memories are slower then how slow are they? On an average it has been found that the operating speed of main memories lack by a factor of 5 than that of the speed of processors (such as CPU or Input Output Processors).

In addition each instruction require several memory accesses (It may range from 2 to 7 or even more instances). If an instruction requires even 2 memory accesses even then almost 80% time of executing an instruction, processor waits for memory access.

What can be done to increase this processor-memory interface bandwidth?

There are four possible answers to the questions. These are:

(*i*) Decrease the memory access time, use a faster but expensive technology for main memory probably it will be feasible after few years.

(*ii*) Access more words in a single memory access cycle. That is instead of accessing one word from the memory in a memory access cycle, access more words.

(*iii*) Insert a high speed memory termed as Cache between the main memory and processor.

(*iv*) Use associative addressing in place of random access.

Hardware researchers are taking care of the first point. Let us discuss the schemes for rest three points.

9.4.1. Interleaved Memory

The interleaved memories are the implementation of the concept of "accessing more words in a single memory access cycle". But the question is how can it be achieved? By partitioning the memory into say N separate memory modules. Thus. N accesses can be carried out to the memory simultaneously. Figure 9.13 gives a block diagram for interleaved memory system. But what are the requirements for such a system? For such a system we will be requiring:

- independent addressing circuitry for each of the N modules.
- an appropriate bus control mechanism, in case the physical buses between the process and memory are shared by the modules.
- sufficient buffer storage in the 'processor' to control the increased flow of information

Where this memory can be useful? Well, this kind of interleaved memory has found application in multi-processor systems where many processor require access to common memory simultaneously. But different processor can access the memory simultaneously only if they desire access to different module of the memory. For utilising such a system effectively, the memory references generated by referencing processors should be distributed evenly among the N module. In an ideal case for a set of N consecutive references to the memory every references should be to a separate module. In such a case simultaneous memory access can be carried out. In case, two or more than two references are to the same module then those memory accessess cannot be carried out simultaneously. The maximum processor memory bandwidth in interleaved memory can be equal to the number of modules *i.e.* N words per memory cycle

To achieve the address interleaving consecutive addresses are distributed among the N inter- leaved modules. For example, if we have consecutive addresses and 4 interleaved memory modules then the 1st, 5th, 9th, 13th ... address will be assigned to the first memory module and so on.

1, 5, 9, 13, 17 Addresses to Memory Module 1

2,6,10,14, 18 Addresses to Memory Module 2

3, 7, 11, 15, 19 Addresses to Memory Module 3

4, 8, 12, 16, 20 Addresses to Memory Module 4

For example, Cray 1 super computer which have a CPU cycle time (which is defined as shortest time in executing a microoperation by CPU) of 12.5 ns and main memory of cycle time 50 ns and word size of 64 bits. Since one instruction in CRAY normally read. One instruction word, two operand words and one result words i.e. 4 words, therefore, 4 memory cycles are required for a CPU cycle. Since 1 memory cycle = 4 CPU cycle time, therefore, at least 4x4 = 16 memory modules are used to make it 16 module interleaving for executing the instruction. This way CPU wait time is minimised.

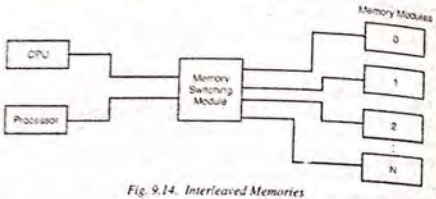


Fig. 9.14. Interleaved Memories

9.4.2. Cache Memory

The active portions of the program and data are placed in fast small memory, the average memory access time can be reduced, thus reducing the total execution time of the program. Such a fast small memory is referred to as a cache memory.

9.4.2.1. Cache Memory System

A cache memory system includes a small amount of fast memory (SRAM) and a large amount of slow memory (DRAM). This system is configured to simulate a large amount of fast memory. Figure 9.15 shows a cache memory system.

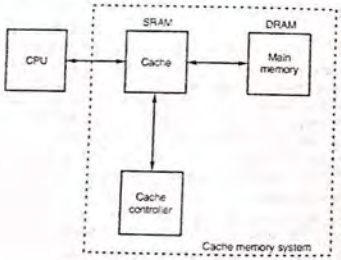


Fig. 9.15. Cache memory system.

The access time of cache memory is very less as compared to the access time of main memory. When a read request is received from the CPU, the contents of a block of memory words containing the location specified are transferred into the cache one word at a time. Subsequently, when the program references any of the location in this block, the desired contents are read directly from the cache. Usually, the cache memory can store a reasonable number of blocks at any given time, but this number of small compared to the total number of blocks in the main memory. If the processor finds that the dressed data is not available in cache, then processor accesses that data from the main memory (DRAM). The percentage of accesses where the processor finds the data word it needs in the cache memory is called the hit rate. The hit rate is normally greater than 90 percent.

$$\text{Hit rate} = \frac{\text{Number of hits}}{\text{Number of read / write bus cycle}} \times 100\%$$

Program Locality

In cache memory system, prediction of memory location for the next access is essential. This is possible because computer systems usually access memory from the consecutive location. This prediction of next memory address from the current memory address is known as program locality. Program locality enables cache contriller.

Locality of Reference

We know that a program may contain a simple loop, nested lops or a few procedures that repeatedly call each other. The actual detailed pattern of instruction sequencing is not important, the point is that many instructions in localised areas of the program are executed repeatedly during some time period, and the remainder of the program is accessed relatively infrequently, this is referred to as locality of reference. It manifests itself in two ways : *temporal* and *spatial*. The temporal means that a recently executed instruction is likely to be executed again very soon. The spatial means that instructions stored near by to the recently executed instructions are also likely to be executed soon.

The temporal aspect of the locality reference suggests that whenever an information of instruction and data is first needed, this information should be brought into cache where it will hope fully remain until it is needed again. The spatial aspects suggests that instead of bringing just one item from the main memory to the cache, it is wise to bring several items that reside at adjacent addresses as well. We use the term block to refer to a set of continuous addresses of size.

9.4.2.2. Element of Cache Design

The cache design elements include cache size, mapping function, replacement algorithm, write policy, block size and number of caches.

Cache Size

The size of the cache should be small enough so that the overall average cost per bit is close to that of main memory alone and large enough so that the overall average access time is close to that of the cache alone.

Mapping Function

The cache memory can store a reasonable number of blocks at any given time, but this number is small compared to the total number of blocks in the main memory. Thus, we have to use mapping

functions to relate the main memory blocks and cache blocks. There are two mapping functions commonly used : Direct mapping and associative mapping.

Replacement Algorithm

When the cache is full and a memory word that is not in the cache is referenced. The cache controller must decide which block should be removed to create space for the new block that contains the referenced word. The collection of rules for making this decision constituents the replacement algorithm. There are four most common replacement algorithms :

- Least-Recently Used (LRU)
- First-In First-Out (FIFO)
- Least-Frequently-Used (LFU)
- Random

Write Policy

It is also known as cache updating policy. In cache system, two copies of the same data can exist as a item, one in cache and one in main memory. If one copy is altered and other is not two different sets of data become associated with the same address. To prevent this the cache system has updating system such as : Write through system, buffered write through system, and write-back system.

Block Size

It should be optimum for cache memory system.

www.IPUnation.com

Number of Caches

When on-cip cache is insufficient the secondary cache is used. The cache design changes as number of caches used in the system changes.

9.4.2.3. Mapping Function

The transformation of data from main memory to cache memory is referred to as a mapping process. The basic characteristics of cache memory is its fast access time. Therefore, very little or no time must be wasted when searching for words in the cache. The transformation of data from main memory cache memory is referred to as a mapping process. There are two mapping techniques which decides the cache organisation :

1. Direct mapping technique
2. Associative mapping technique

The associative mapping technique is faster classified as fully fassociative and set associative technique.

To discuss the mapping technique of cache mapping, we consider the following specifications.

Memory	*Memory capacity*	*Number of block per block*	*Number of words*
1. Cache	2048 (2K)	128	16
2. Main memory	64 K	4 K	16

The group of 128 blocks in main memory form a page. Next we discuss the mapping technique in the following sections.

a) Direct Mapping

The main memory blocks can be placed in one and only one placed in the cache, this mapping is called as direct mapping. The direct mapping technique is the simplest mapping technique. In this technique, each block from the main memory has only one possible location (fixed place) in the cache organisation. The block j of the main memory maps onto block j modules 128 of the cache, illustrated in the Fig. 9.16. Therefore, whenever one of the main memory blocks 0, 128, 256 ... (the first block in each page of main memory is stored in the first block of cache memory), it is stored in cache block 0. Blocks 1, 129, main memory is stored in the first block of cache memory), it is stored in cache block 0. Blocks 1, 129, main memory is stored in the first block of cache memory), it is stored in cache block 0. Blocks 1, 129, 257, ... are stored in cache block 1, and so on.

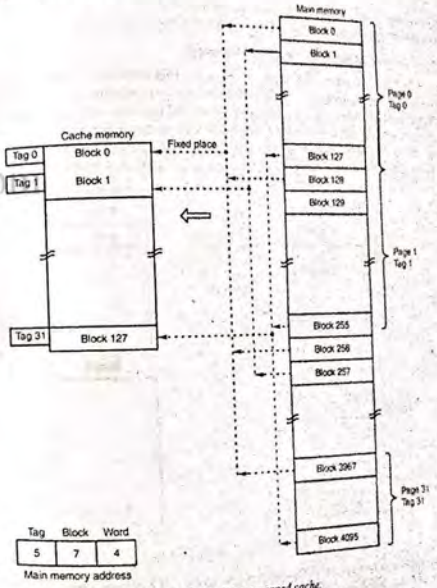


Fig. 9.16. Direct mapped cache.

The placement of a block in the cache is determined from the address of main memory. The address is divided into three fields as shown in Fig. 9.16. The lower order 4-bits select one of the 16 words in a block. This field is known as word field. The second field known as block field is used to distinguish a block from other blocks. When a new block enters the cache, the 7 bit cache block field determines the position in which this block must be stored. This third field is a tag field. Each word in cache consists of the data word and its associated tag field.

When a new word is first brought into the cache, the tag bits are stored alongside the data bits. When the CPU generates a memory request, the word field and block fields are used for the address to access the cache. The tag field of the CPu address is compared with the tag in the word read from the cache. If the two tags match, there is a hit (read from cache memory) and the desired data word is in cache. If there is no match, there is miss (data is not available in the cache, data read from main memory) and the required word read from main memory. It is then stored in the cache together with need tag by replacing the previous word.

The miss occurs, an entire block of 16 words must be transferred from main memory to cache memory. The direct mapping technique is easy to implement, but it is not very flexible.

(b) Fully Associative Mapping (Associative Mapping)

Figure 9.17 shows the associative mapping technique. This technique is faster and must flexible cache organisation. A main memory block can be placed into any cache block position. As there is no fixed block, the memory address has only two fields : Word and tag. This technique is also referred to as fully associative cache. In this case, 12 tag bits are reuqired to identify a memory block when it is placed in the cache. The tag bits of an address received from the CPU are compared to the tag bits of each block of the cache to read if the desired block is present. This is called the asosciative mapping technique.

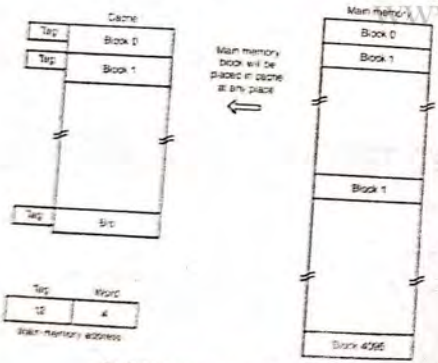


Fig. 9.17. Associative mapped cache.

The cache space can be used more efficiently. A new block that has to be brought into the cache has to replace the existing block only if the cache is full. In this case, we need replacement algorithm to

existing block in the cache. The cost of an associative cache is higher than the cost of direct mapping cache because of the need to search all 128 tag patterns to determine whether a given block is in the cache.

(c) Set Associative Mapping

Set associative mapping is a combination of both direct and associative mapping. It contains several blocks of direct mapped blocks that operate in parallel. A block of data from any page in the main memory can go into a particular block of any direct-mapping cache. Hence the contention problem of the direct mapped technique is eased by having a few choices for block placement. The required address comparisons depend on the number of direct mapped cache in the cache system. These comparisons are always less than the comparisons required in the fully associative mapping.

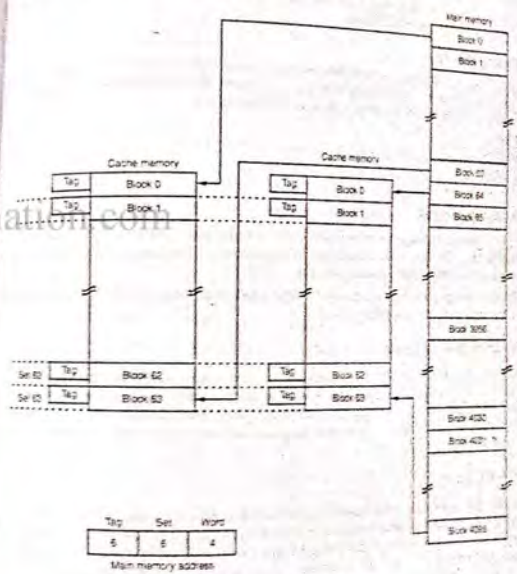


Fig. 9.18. Set associative cache.

Figure 9.18 shows the set asosciative mapped cache with two blocks per set. Each page in the main memory is organised in such a way that the size of each page is same as the size of one directly mapping

cahce. It is called two-way set associative cache because each block from main memory has two choice for block placement. In this technique, block 0, 64, 128, ..., 4032 of main memory can place into any of the two (block 0) blocks of set 0 and they occupy either of two block positions within this set. Having 64 sets means that the 6-bit set field of the address determines which set of the cache might contain the desired block. The tag field of the address must then be associatively compared to the tag of the two blocks of the set to check if the desired block is present. Tis two-way associative search is simple to implement.

The tag field of the CPU address is then compared with both tags in the cache to determine if a match occurs. The comparison logic is done by an associative search of the tags in the set similar to an associative memory search. The hit ratio is improved in this technique because the set size increase more words with same index but different tags can reside in cache. However, an increase in the set size increases the number of bits in words of cache and requires more complex comparison logic. When a miss occurs in a set associative cache and the set is full, it is necessary to replace one of the tag-data items with new data by using any one replacement algorithm.

(d) Cache Updating

In cache system, the copies of same data can exist at a time, one in cache and the other one in main memory. If one copy is altered and the other is not, two different sets of data become associative with the same address. To prevent this the cache system has updating such as

- Write through system
- Buffered write through system and
- Write back system.

Write Through System

In write through updating system, the cache contriler copies data to the memory immediately after it is written to the cache. Due to this, main memory always contains a vald data and any block in the cache can be overwritten immediately without data loss.

This is a simple approach. It requires time to write data in main memory with increase in bus traffic. This effect reduces the system performances.

www.IPUnation.com

Buffer Write Through System

In buffered write through system, the processor can start a new cycle before the write cycle to the main memory is completed. This means that the write accesses to the main memory are buffered. In such systems, read access which is a "cache hit" can be performed simultaneously when main memory is updated. However, to consecutive write operation to the main memory or read operation with "cache miss" require the processor to wait.

Write Back System

In this method only the cache location is updated during the write operation. The location is then marked by a flag so that later when the word is removed from the cache it is copied into main memory. The reason for the write back system is that during the time a word resides in the cache, it may be updated several times; however, as long as the word remains in the cache, it does not matter whether the copy in main memory is out of data, since requests from the word are filled from the cache. It is only when the word is displaced from the cache that an accurate copy need be rewritten main memory.

Due to one time write operation number of write cycles are reduced in write back system. But this system has following disadvantages.

1. Write-Back cache controller is more complex.
2. In case of power failure, the data in the cache memory is lost.

9.3. Associative Memories

In associative memories any stored item can be accessed directly by assigning the contents of the in question, such as name of a person, account number, number etc., as an address. Associative memories are also known as content addressable memories (CAMs). The entity chosen to address the memory is known as the key.

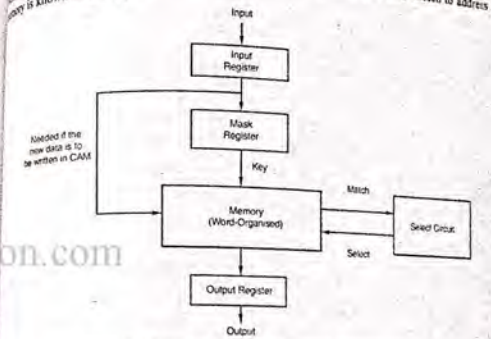


Fig. 9.19. Structure of a simple word-organised associative memory

Figure 9.19 shows the structure of a simple associative memory. The information is stored in CAMs a fixed-length words. Any entity of the word may be chosen as the key field. The desired key is indicated the mask register. Then, the key is compared simultaneously with all stored words. The words that match the key issue a match signal. This match signal then enters a select circuit. The select circuit in turn sign for the access of required data field. In case more than one entries have the same by, then it is the responsibility of the select circuit to determine the data field to be read. For example, the select circuit read out all the matching entries in a pre-determined order. Each word is provided with its own match circuit as all the words in the memory need to compare their keys with the desired key simultaneously. The match and select circuits thus, make these associative memories very complex and expensive than the conventional memory. The VLSI technology has made these associative memories economically feasible. Even now the cost considerations limit the applications of associative memories to relatively small amount of information which needed to be accessed very rapidly. An example, of CAM use is in memory address mapping.

9.4.4. Virtual Memory

In memory hierarchy systems, the physical main memory is not as large as the address space spanned by an address issued by CPU. When a program does not completely fit into the main memory the parts of

it (part of program or data) not currently being executed are stored on secondary storage devices such as magnetic disks and tapes.

If an executing program needs a segment which is not currently in the main memory, the required segment is copied from the secondary storage device. When a new segment of a program is to be copied into a main memory it must replace sanother segment already available in the memory.

In modern computers, the operating system moves program and data automatically between the main memory and secondary storage. Techniques that automatically swaps programmed data blocks between main memory and secondary storage device are called virtual memory. The addresses that CPU issues to access either instruction or data are called virtual or logical addresses. These addresses are translated into physical addresses by a combination of hardwares and software compoents. The virtual address refers to a part of the program or data space that is currently in the main memory, then the contents of the appropriate location in the main memory are accessed immediately. On the other hand, if the reference address is not in the main memory, its contents must be brought into a suitable location in the main memory before they can be used.

Figure 9.20 shows the typical implementation of virtual memory. A special hardware is Memory Management Unit (MMU) is used to translate the virtual address to physical addresses. When the desired data or programs are in the main memory, these data or program are fetched directly from the main memory. If the data are not in the main memory, the MMU comes the operating system to bring the data from the secondary storage like magnetic disk.

Virtual Memory : A memory hierarchy consisting of main memory and auxiliary memory is known as virtual memory.

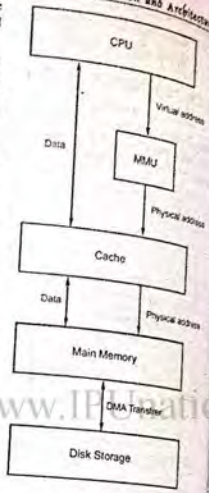


Fig. 9.20. Virtual memory organization.

Address Mapping Using Pages

The table implementation of the address mapping is simplified if the information in the addres space and the memory space are divided into groups of fixed size. The secondary memory or physical memory is broken down into groups of equal size refers called blocks. The term page refers to group of address space of the same size. Let us assume the capacity of physical memory is 8K. We can subdivide memory into 1 K capacity, so we obtain 8K/1K = 8 pages. The memory has 8 physical pages each holding 1K, it is shown in Fig. 9.21.

The programs are also divide into number of pages. The portion of programs and data are transferred from auxiliary memory to main memory to execute it. A page frame is sometimes used to denote a block in the main memory. For example, the capacity of main memory is 4K, it is subdivided into 1K capacity, so we obtain 4 blocks in the main memory, it is illustrated in the Fig. 9.21.

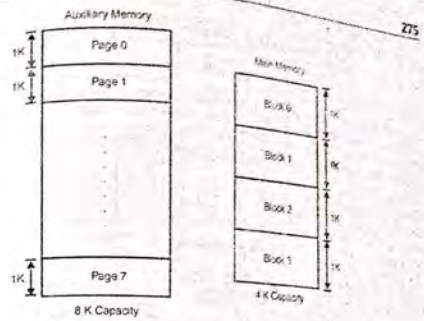


Fig. 9.21. Relation between address and memory space in virtual address.

Logical Address Space

The logical address space in the rage of logical addresses that can be generated by CPU.

- One many systems, the logical address space is dictated by the architecture of the CPU—specifically by the number of bits in the address position of an instruction. For example, a CPU that uses 16 bit address in its instruction, the logical address would be 0 to 65535 (0 to $2^{16} - 1$)
- One CPU whose architecture supports relatively long addresse, logical space any be dictated by the operating system to be some subset of the potential logical space allowed by the architecture.

Physical Address Space

The physical address space is the range of physical addresses that can are recognised by the memory. For example, if 1 MB of physical memory, then the physical address space would be 0 to 1, 048, 575 (2^{20} = 1,048, 5769, the physical address size is 20 bits).

Address Translation

Address translation is defined as the translation of one type of address (logical address) to another type of address (physical address). Figure 9.22 shows the virtual memory address translation.

Recall that in paging scheme each process has a page table which serves to map logical addresses generated by the process to actual physical address. The address translation process can be described as follows :

1. Break the logical address down into page number and offset.
2. Use the page number as an index into the page table to find the corresponding page frame.
3. Using the frame number found there, generate a physical address by concatenating the frame number and the offset from the original address.

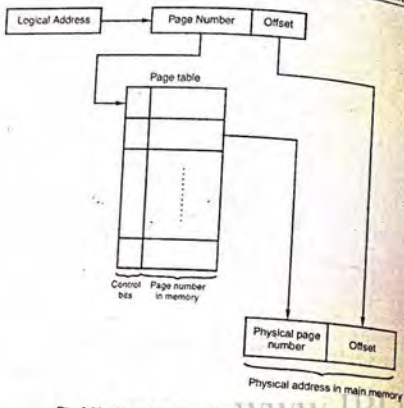


Fig. 9.22. Virtual to physical address translation.

www.iPCuRoom.com

For example, assume that the page size is 256 bytes, that logical addresses are 16 bits long and physical addresses are 24 bits long (all numbers in the table are hexa decimal number). The Fig. 9.23 shows the detailed address translation.

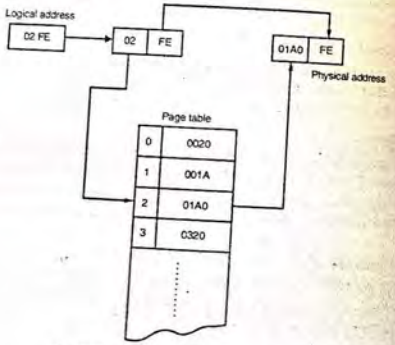


Fig. 9.23. Example of address translation.

SOLVED PROBLEMS

Example 9.1. *What are the memory performance parameters.*

Solution. The most important memory performance parameters are access time t_a, the time interval from the start of a read signal until the assertion of the memory completion signal, and cycle time t_c. The minimum time interval from the start of a read or write to the start of the next memory operation. The cycle time may be slightly longer than the access time, due to various hardware housekeeping tasks that must be performed by the memory.

Some memories read or write values not as individual words, but in blocks of k words. In this memories, there will be latency t_l in accessing the first word of the block, that differs from the bandwidth, w, the rateat which the words can be transmitted, in words per second, once the first word in the block is available. The access time for the entire block $t_{bl} = t_l + k/w$. These parameters are summarised in the following table :

Symbol	*Definition*	*Units*	*Meaning*
t_a	Access time	Time	Time to access a memory word
t_c	Cycle time	Time	Time from start of read to start of next
k	Block size	Word	Number of words per block
(a)	Bandwidth	Words/sec	Word transmission rate
t_l	Latency	Time	Time to access first of a sequence of words
$t_{bl} = tl + k/w$	Block access time	Time	Time to access entire block from start of read.

Example 9.2. *Explain the memory hierarchy with performance.*

Solution.

Components	CPU	Cache Memory	Main Memory	Disk Memory	Tape Memory
Access type	Random access	Random access	Random access	Direct access	Sequential acess
Capacity, bytes	64-124	8-256 KB	8-64 MB	1-10 GB	1 TB
Lasttency	1-10 nS	20 nS	50 nS	10 nS	10S-10S
Block size	1 word	16 words	16 words	4 KB	4 KB
Bandwidth	System clock rate	8 MB/S	1 MB/S	1 MB/S	1 MB/S

Example 9.3. *Explain the cache read operation.*

Solution. Figure 9.24 shows a small cache system that illustrates the relationship between the data stored in the cache M_1 and the data stored in main memory M_2. In this figure, a cache block (line) size is assumed 4 bytes. Each memory address is 12 bit long, so the 10 high-order bits from the tag or block address, and the 2 low-order bits define a displacement address within the block. When a block is assigned to M_1's data memory, its tag is also placed in M_1's tag memory. Figure 9.34 shows the contents of three blocks assigned to the cache data memory. It is important to note the locations of the same blocks in main memory. To read the bold types word, its address A_1 = 101111000101 is sent to M_1, which

Input-Output Organization

10

10.1. PERIPHERAL DEVICES

www.IPU

The input and output units of a computer system are known as Input/Output units. They provide an efficient way of communication between the computer system and the outside world. A computer must have a system to receive information and must be able to communicate results to the external world. The most familiar input devices connected to the computer is keyboard. Devices that are under the control of the computer are said to be connected online. Input/output devices that are attached to the computer are also known as peripherals. Most common peripherals devices are keyboards, mouses, display units, printers etc. The devices which are connected to a computer and controlled by processor are called peripheral devices.

10.1.1. Keyboards

Keyboard is the most familiar input device for entering information into a computer. The layout of the keyboard is similar to that of the traditional typewriter with some extra command and functional keys. When a key on the keyboard is pressed, a binary coded character is send to the computer and the corresponding character is displayed on the screen. If a wrong key is pressed, it can be erased and the correct character can be retyped. The fastest possible speed of entering the information using keyboard depends on the person's typing speed. A keyboard used in a computer system is shown in Fig. 10.1.

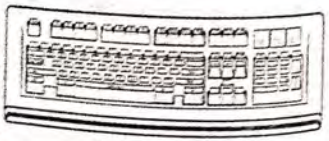


Fig. 10.1. Keyboard.

10.1.2. Mouse

Mouse is a pointing device. It can be moved on a smooth surface so that it points to the desired location on the screen. It can be optical or mechanical. User can move the mouse, stop it at a desired location and with the help buttons can makes selection of choices. The mouse used in a computer system is shown in Fig. 10.2.

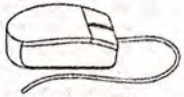


Fig. 10.2. Mouse.

10.1.3. Display Devices

One of the most important peripheral of the computer system is the display moniter. There are different types of display moniter but the most popular is cathode ray tube (CRT). Others are liquid crystal display (LCD) and projection displays. The CRT contains a beam of electrons (cathode rays) emitted by the electron gun, passes through focusing and deflection systems that direct the beam toward specified position on the phosphor-coated screen. The phosphor then emits a small spot of light at each position contacted by the electron beam. The monitor used in a computer system is shown in Fig. 10.3.

Fig. 10.3. Display Monitor

10.1.4. Printers

Printers are used for getting the output on paper. Printers can be classified according to the print quality and the speed of the printing. The three basic types of character printers are daisywheel, dot matrix and laser printers. The daisywheel printers contains a wheel and the characters are placed along the circumference. When the printer has to print a character, the wheel rotates to the proper position and an energized magnet then presses the letter against the ribbon. Dot matrix printers uses impact technology and a print head containing banks of wires moving at high speeds against inked ribbon and paper. In laser printer a laser beam creates a charge distribution on a rotating drum coated with a photoelectric material, which is used to imprint the character images. The pattern is then transferred onto the paper in the same manner as a copying machine. The printer used in a computer system is shown in Fig. 10.4.

Fig. 10.4. Printer

10.2. INPUT-OUTPUT INTERFACE

Input-output interface provides a method for transferring information between internal storage and external I/O devices. Peripherals connected to a computer need special communication links for interfacing them with the central processing unit. The purpose of the communication link is to resolve the differences that exist between the central computer and each peripheral. A hardware unit that plays an important role between the CPU and peripheral devices to supervise the input and output transfers is known as interface.

The major differences are :

(1) Peripherals are electromechanical and electromagnetic devices and their manner of operations are different from the operation of the CPU and memory, which are electronic devices. Therefore, the conversion of signals are required.

(2) The data transfer rate of peripherals are usually slower than the transfer rate of the CPU and hence, a synchronization mechanism are needed.

(3) Data codes and formats in peripherals are different from the word format in the CPU and memory.

(4) The operating modes of peripherals are different from each other and each must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

To resolve these differences, computer systems include special hardware known as "interface units" between the CPU and peripherals to supervise and synchronize all input and output transfers.

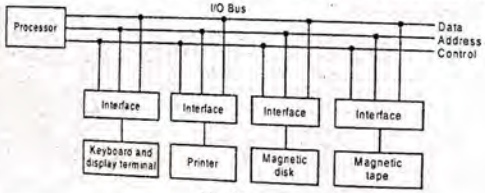


Fig. 10.5.

A typical communication link between the processor and several peripherals are shown in Fig. 10.5. The I/O bus consists of data lines, address lines and control lines.

Each peripheral devices has its associated interface unit. Each interface unit decodes the address and control received from I/O bus, interprets them for the peripheral and provides signals for the peripheral controller. It also synchronizes the data flow and supervises the transfer between peripherals and processor. Each peripheral has its own controller that operates the particular electromechanical device. For example, the printer controller controls the paper motion, the printing timing and the selection of printing characters.

The I/O bus from the processor is attached to all peripherals interfaces. To communicate with a particular device, the processor places a device address on the address lines. Each interface attached to the I/O bus contains an address decoder that monitors the address lines. When the interface detects its own address, it activates the path between the bus lines and the devices that it controls. All peripherals whose address does not correspond to the address in bus are disabled by their interfaces.

At the same time that the address is made available in the address lines, the processor provides a function code in the control lines. The interface selected responds to the function code and proceeds to execute it. These function codes are referred to as I/O commands. There are four types of commands that the interface may receive. They are :

(*i*) **Control Command :** A control command is issued to activate the peripheral and to inform it what to do. The particular control command issued depends on the peripheral.

For example, a magnetic tape unit may be instructed to back space the tape by one record, to rewind the tape or to start the tape moving in the forward direction.

(*ii*) **Status :** A status command is used to test various status conditions in the interface and the peripherals.

(*iii*) **Output Data :** A data output command causes the interface to respond by transferring data from the bus into one of its registers.

(*iv*) **Input Data :** The data input command is the opposite of the data output. In this case the interface receives an item of data from the peripheral and places it in its buffer register. The processor checks if the

data are available by means of a status command and then issues a data input command. The interface places the data on the data lines, where they are accepted by the processor.

10.2.1. I/O Versus Memory Bus

The processor must communicate to the memory unit along with the I/O unit. The memory bus also contains the data, address and readwrite control lines as in I/O bus. The computer buses can communicate memory and I/O in three different ways :

(*i*) Use two separate buses, one of memory and one for I/O.

(*ii*) Use one common bus for both memory and I/O bus having separate control lines for each.

(*iii*) Use one common bus and common control lines for both memory and I/O.

10.2.2. Isolated Versus Memory-Mapped I/O

The transfer of information between the CPU and the memory or I/O are done by means of a common bus. By enabling the specific read or write lines, the CPU specifies whether the address on the address bus is for a memory word or for an interface registers. During a memory transfer, the memory read or memory write control lines are enabled and during I/O transfer, the I/O read or I/O write control lines are enabled. In this way the I/O interface addresses are isolated from the memory addresses and is termed as isolated I/O method for assigning addresses in a common bus.

In isolated I/O configuration, the input and output instructions for both interface register and memory are different. When the CPU fetches and decodes the instructions for I/O interfaces, it enables the I/O read or I/O write control line and informs the other components attached to the common bus that the address in the address bus is for interface register and not for memory word. Similarly, when the CPU fetches and decodes the instruction from memory, it places the memory address on the address bus and enables the memory read or memory write control line, which inform the external components that the address is for memory word and not for I/O interface. Hence, the memory and I/O addresses are isolated in this method.

Alternative, method is known as memory mapped I/O. In this method the computer use the same address space for memory and I/O, both. In memory-mapped I/O there is no specific input or output instructions. The same instruction can be used for manipulating I/O data residing in interfaces and can also be used to manipulate memory words. Computer with memory-mapped I/O can use memory-type instructions to access I/O data and hence, allows the computer to use the same instructions for memory transfers or input-output transfers.

10.3. DATA TRANSFER SCHEMES

The data transfer scheme refers to the method of data transfer between the CPU and the peripheral devices. Data transfer takes place between any two devices : CPU and memory, CPU and I/O devices memory and I/O devices. For effective data transfer between these devices, the timing parameters of the devices should be matched. But most of the devices have incompatible timings. For example and I/O device may be slower than the CPU due to which it cannot send data to the CPU at the expected time.

The semiconductor memories are available with compatible timings, moreover, slow memories can be interfaced using additional hardware to introduce wait states in machine cycles. The data transfer schemes have been boardly classified into the following two categories.

1. Programmed data transfer.
2. Direct memory access (DMA) data transfer.

Programmed data transfer scheme is used when relatively small amount of data are to be transferred. In these schemes, usually one byte or word of data is transferred at a time.

The programmed data transfer scheme can be further classified into the following three types.

1. Synchronous data transfer.
2. Asynchronous data transfer.
3. Interrupt data transfer.

In DMA data transfer the CPU is forced to HOLD state (high impedance state) by an I/O until the data transfer between the device and the memory is complete. The CPU does not execute any instruction during the HOLD period.

The DMA data transfer is used for large block of data transfer between I/O device and memory. The different types of DMA data transfer schemes are :

1. Cycle stealing or single transfer mode DMA.
2. Block or burst mode DMA.
3. Demand transfer mode DMA.

Figure 10.6 shows the various types of data transfer schemes. All the data transfer schemes discussed above requires both software and hardware for their implementation. Within a CPU, more than one scheme can be used for interfacing different I/O devices.

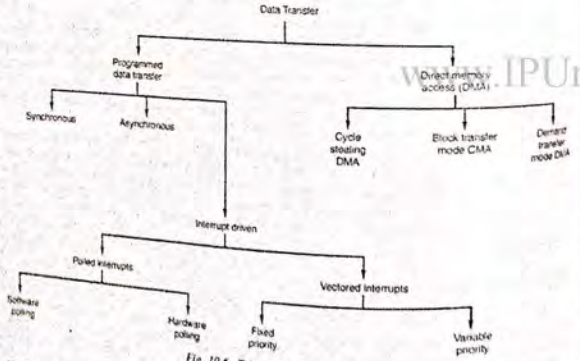


Fig. 10.6. Types of data transfer.

10.3.1. Programmed Data Transfer

Most peripherals respond slowly in comparison with the speed of the CPU. Therefore, it is necessary to set up conditions for data transfer so that data will not be lost during the transfer. CPU controlled data transfer or programmed data transfer can take place between CPU and I/O devices. Any data transfer of this scheme is controlled by CPU (or) program stored in the CPU.

10.3.1.1. Example of Programmed I/O

In the programmed I/O method, the I/O device does not have direct access to memory. The data transfer between I/O device and memory, requires execution of some instruction by CPU. This data transfer is controlled by program, so it is called programmed I/O. Program to be executed for data transfer is includes an input instruction to transfer the data from the device to the CPU and a store instruction to transfer the data from the CPU to the memory. Other instructions may be needed to verify that the data is available from the device and to count the numbers of words transferred.

Figure 10.7 shows the data transfer from I/O device through an interface into the CPU. The device transfers bytes of data one at a time as they are available. When a byte of data is available, the device places it in the I/O bus and enables its data valid line. The interface unit accepts the data into its data register and enables the data accepted line. The interface sets a bit in the status register that we will refer to as an 'F' or 'flag' bit. The device can now disable the data valid line, but it will not transfer another byte until the data accept line is disabled by the interface.

A program is written for the computer to control the above task of transferring data between I/O devices and memory.

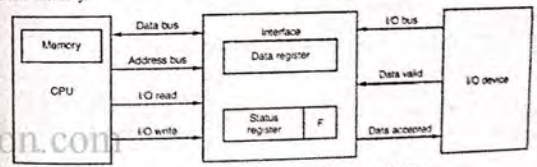


Fig. 10.7. Data transfer between I/O devices to CPU.

10.3.2. Synchronous Data Transfer Scheme

The synchronous data transfer scheme is the simplest of all data transfer schemes. In this scheme the CPU does not check the readiness of the device. The I/O device or peripheral should have matched timing

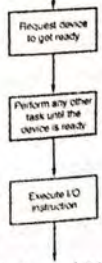


Fig. 10.8. Synchronous data transfer.

www.IPUnation.com

parameters. Whenever data is obtained from the device or transferred to the device, the use program can issue a suitable instruction for the device. At the end of the execution of this instruction, the transfer would have been completed. Figure 10 8 shows the flowchart for synchronous data transfer.

10.3.3. Asynchronous Data Transfer Scheme

The asynchronous data transfer scheme is employed when the speed of the CPU and I/O devices do not match. In this scheme the CPU sends a request to the I/O device for read/write operation. Then the processor keeps on polling the status of the device. Once the device is ready, the CPU executes a data transfer instruction to complete the process. To implement this scheme, the device should provide a signal which may be tested by the CPU to ascertain whether it is ready or not. Fig. 10.9 shows the operation for asynchronous data transfer.

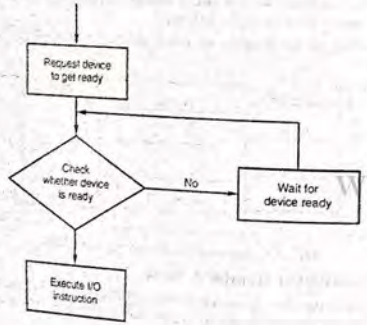


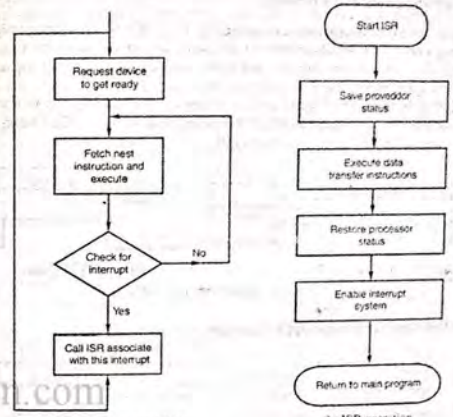
Fig. 10.9. Asynchronous data transfer.

10.3.4. Interrupt Driven Data Transfer

The interrupt driven data transfer scheme is the best method of data transfer for efficient utilization of the CPU time. In this scheme, the processor first initiates an I/O device for data transfer. After initiating the device. CPU will continue the execution of instructions in the program. Also at the end of every instruction the CPU will check for valid interrupt signal. If there is no interrupt then the CPU will continue the execution

When the I/O device is ready, it will interrupt the CPU. On receiving an interrupt signal the CPU will complete the current instruction execution and save the CPU status in stack. Then the processor calls an interrupt service routine (ISR) to service the interrupting device.

At the end of ISR, the CPU status is retrieved from stack and the CPU starts executing its main program. The sequence of operation for an interrupt driven data transfer is shown in Fig. 10.10.



(a) Main program execution (b) ISR operation

Fig. 10.10. Interrupt driven data transfer scheme.

10.4. ASYNCHRONOUS DATA TRANSFER

The data transfer between two independent unit is said to be synchronous data transfer if they share a common clock. But if two independent unit uses its own clock for the data transfer between them the data transfer is said to be asynchronous data trasnfers. Most communications between the computer and the I/O devices are asynchronous. This asynchrouns way of data transfer is achieved by two methods :

(*i*) One way is by means of a strobe pulse supplied by one of the units to indicate to the other unit when the transfer has to occur. This method is known as *strobe control*.

(*ii*) Another method commonly used is to accompany each data item being transferred with a control signal that indicates the presence of data in the bus. The unit receving the data item responds with another control signal to acknowledge receipt of the data. This method of transferring data between two independent units is known as *handshaking*.

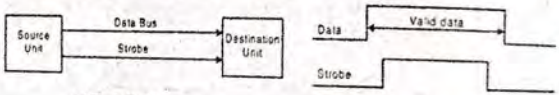
10.4.1. Strobe Control

The strobe control method of asynchronous data transfer employs a single control line. The strobe may be activated by either the source or the destination unit. The data bus carries the binary information from source unit to the destination unit. The bus has multiple lines to transfer an entire byte or word. The strobe is a single line that informs the destination unit when a valid data word is available in the bus.

www.IPUnation.com

Source-Initiated Strobe for Data Transfer

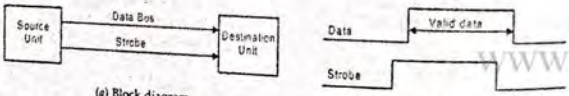
The strobe initiated by the source unit is shown in Fig. 10.11. The source unit first places the data on the data bus. After a brief delay to ensure that the data settle to a steady value, the source activates the strobe pulse. The information on the data bus and strobe control signal remain in the active state for a sufficient period of time to allow the destination unit to receive the data. Actually, the destination unit uses the falling edge of the strobe control to transfer the contents of data bus to one of its internal registers. The source removes the data from the data bus after it disables its strobe pulse. New valid data will be available only after the strobe is enabled again.



(a) Block diagram. (b) Timing diagram.

Fig. 10.11. Source-Initiated Strobe.

Destination-Initiated Strobe for Data Transfer



(a) Block diagram. (b) Timing diagram.

Fig. 10.12. Destination Initiated Strobe.

www.IPUnation.com

The strobe initiated by the destination unit is shown in Fig. 10.12. In this case the destination unit activates the strobe pulse, informing the source to provide the data. The source unit responds by placing the requested binary information on the data bus. The falling edge of the strobe pulse is used to transfer the content of the data bus to one of its internal registers. The destination unit then disables the strobe. The source then removes the data from the data bus after a predetermined period of time.

In the first case, the strobe may be a memory-write control signal from the CPU to a memory unit. The source, CPU places the word on the data bus and informs the memory unit, which is the destination, that this is a write operation.

In the second case, the strobe may be a memory-read control signal from the CPU to a memory unit. The destination, the CPU, initiates the read operation to inform the memory, which is a source unit, to place a selected word into the data bus.

10.4.2. Handshaking

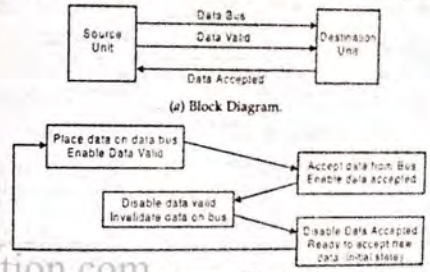
The disadvantage of the strobe method is that the source unit that initiates the transfer has no way of knowing whether the destination unit has actually received the data that was placed in the bus. Similarly, a destination unit that initiates the transfer has no way of knowing whether the source unit has actually placed the data on the bus.

The handshaking problem solves this by introducing a second control signal line that provides a reply to the unit that initiates the transfer.

One control line is in the same direction as the data flow in the bus from the source to destination. It used by the source unit to inform the destination unit whether there are valid data in the bus. The other trol line is in the other direction from the destination to the source.

urce-Initiated Handshaking

The handshaking initiated by the source unit is shown in Fig. 10.13. The two handshaking lines are a valid, which is generated by the source unit and data accepted, generated by the desination unit.



(a) Block Diagram.

(b) Sequence of Events.

(c) Timing Diagram.

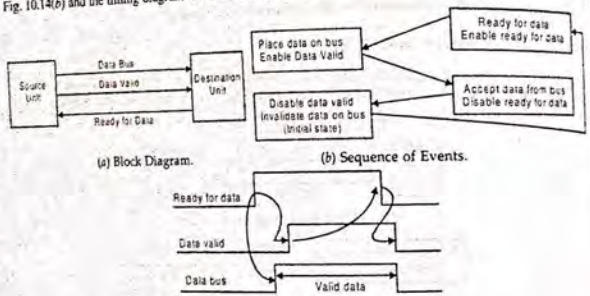
Fig. 10.13. Source-Initiated Handshaking.

The source unit initiates the transfer by placing the data on the bus and enabling its data valid signal. The data accepted signal is then activated by the destination unit after it accepts the data from the bus. The source unit then disable its data valid signal, which invalidates the data on the bus. After this the destination unit disables its data accepted signal and the system goes into initial state. The source does not send the next data item until after the destination unit shows its readiness to accept new data by disabling its data accepted signal. The sequence of event is described in Fig. 10.13(b) and the timing diagram is shown in Fig. 10.13(c) which shows the exchange of signals between two units.

Destination-Initiated Handshaking

The handshaking initiated by the destination unit is shown in Fig. 10.14. The two handshaking lines are data valid, generated by the source unit and ready for data, generated by the destination unit. The source unit does not place the data on the data bus until it receives the ready for data signal from the destination unit. From

there on the handshaking process is same as that of source initiated. The sequence of event is described in Fig. 10.14(*b*) and the timing diagram is shown in Fig. 10.14(*c*).



(*a*) Block Diagram. (*b*) Sequence of Events.

(*c*) Timing Diagram.

Fig. 10.14. Destination-Intiated Handshaking.

The handshaking process provides a high degree of flexibility and reliability becaue the successful completion of a data transfer relies on active participation by both units. If one unit is faulty, the data transfer will be not completed. Such an error is detected by means of a timeout mechanisms, which produces an alarm if the data transfer is not completed within a predetermined time period. The time out is implemented by means of an internal clock that starts counting time when the unit enables one of its handshaking control signals. If return handshake signal does not respond within a given time period, the unit assumes that an error has been occurred. The time out signal can be used to interrupt the processor and hence execute a service routine that takes appropriate error recovery action.

10.5. DIRECT MEMORY ACCESS (DMA)

Direct memory access are system pathways that are used by many I/O devices for transferring information directly to and from memory. Today DMAs are used most commonly by floppy drive, tape drives and sound cards.

Direct memory access is a technique for moving data directly between the main memory and the peripheral devices without the intervention of CPU. Thus, removing the CPU from the path and letting the peripherals device manage the memory buses directly would improve the speed of transfer. This transfer technique is called *Direct Memory Access*.

During DMA transfer, the CPU is idle and has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between the I/O device and memory.

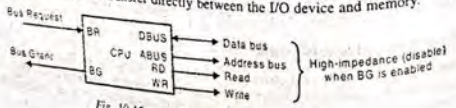


Fig. 10.15. Bus Signals for DMA Transfer.

Figure 10.15 shows two control signals in the CPU that facilitate the DMA transfer. The bus request (BR) input is used by the DMA controller to request the CPU to relinguish the control of the buses. When this input is active, the CPU terminates the execution of the current instruction and places the address bus, the data bus and the read and write lines into a high impedance state. Then the CPU activates the bus grant (BG) output to inform the external DMA that the buses are in the high impedance state. The DMA that originated the bus request, can now take control of the buses to conduct memory transfers without processor intervention. When the DMA terminates the transfer, it disables the bus request line. The CPU then disables the bus grant line and takes control of the buses and returns to the normal operation.

When DMA takes control of the bus system, it communicates directly with the memory. The transfer can be made in several ways.

(*i*) **Burst Transfer :** In DMA burst transfer, a block sequence consisting of a number of memory words is transferred into a continuous burst while the DMA controller is master of the memory buses.

(*ii*) **Cycle Stealing :** In this method, the DMA controller transfers one data word at a time, after which it must return control of the buses to the CPU. The CPU merely delays its operation for one memory cycle to allow the direct memory I/O transfer to "Steal" one memory cycle.

10.5.1. DMA Controller

The DMA controller needs the circuits of an interface to communicate with the CPU and I/O device. It also needs an address register, a word count register, a set of address lines. The address register and address lines are used for direct communication with the memory. The word count register specifies the number of words that must be transferred.

PUnation.com

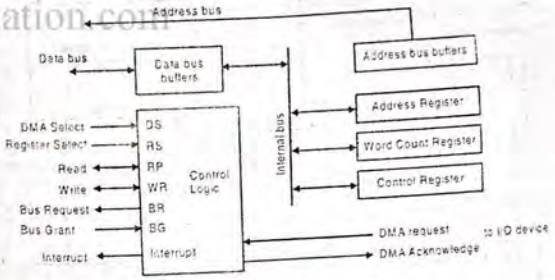


Fig. 10.16. Block Diagram of DMA Controller.

Figure 10.16 shows the block diagram of DMA controller. The unit communicates with the CPU via the data bus and control lines. The registers in the DMA are selected by the CPU through the address bus by enabling the DS (DMA select) and RS (register select) inputs. When the BG (bus grant) input is 0, the CPU can communicate with the DMA registers through the data bus to read from or write to the DMA registers. When BG = 1, the CPU relinguished the buses and the DMA can communicate directly with the memory by specifying an address in the address bus and activating the RD or WR control.

The DMA controller has three registers. The *address register* contains an address to specify the desired location in memory and the address bits goes into the address bus through address bus buffers. The address register is incremented after each word that is transferred to memory. The *word count register* holds the number of words to be transferred to memory. This register is decremented by one after each word is transferred into memory and internally tested for zero. The *control register* specifies the mode of transfer. All registers in DMA appears to the CPU as I/O interface registers and hence the CPU can read from or write into the DMA registers.

10.5.2. DMA Initialization

The CPU initializes the DMA by sending the following information through the data bus :

(*i*) The starting address of the memory block where data are available (for read) or where data are to be stored (for write).

(*ii*) The word count, which is the number of words in the memory block.

(*iii*) Control to specify the mode of transfer such as read and write.

(*iv*) A control to start the DMA transfer.

The starting address is stored in the address register. The word count is stored in the word count register and the control information is stored in the control register. Once the DMA is initialized, the CPU stops communicating with the DMA unless it receives an interrupt signal or if it wants to check how many words have been transferred.

10.5.3. DMA Transfer

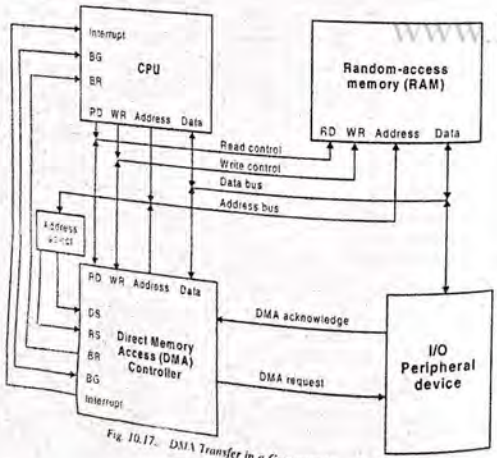


Fig. 10.17. DMA Transfer in a Computer System.

Figure 10.17 shows the DMA transfer in a computer system. With the help of address and data bus, CPU communicates with the DMA. DMA has its own address, which activates DS and RS lines. The CPU initializes the DMA through data bus. The DMA can start data transfer between the peripheral device and the memory once it receives the start control command.

When the peripheral device sends a DMA request, the DMA controller activates the BR line, requesting the CPU to give control over the buses. Then CPU activates BG line and informs the DMA controller that its buses are now disabled. Then the DMA sends a DMA acknowledge to the device by putting the current value of its address register into address bus and initiates the RD or WR signals. After receiving the acknowledge signal, the device puts a word in the data bus (for write) or receives a word from data bus (for read). The DMA increments its address register and decrements the word count register after each word that is transfered. Internally, the word count register is tested for zero. If the word count register reaches zero, the DMA stops any further transfer and disables its bus request signals and informs the CPU through an interrupt. When CPU receives the interrupt signals, it reads the word-count register. The zero value in word-count register signifies that all the words are transferred successfully.

DMA transfer is very useful in many applications. It is used for fast transfer of information between magnetic disk and memory.

10.6. SERIAL COMMUNICATION

The input/output devices, DMA controllers, and I/O processors presented in this chapter all use parallel communication. They transmit more than one bit of data at a time. Some devices cannot handle more than one bit of data at any given time by design; they utilize **serial communication.** This may be done to reduce cost or to facilitate communication with devices outside of the computer system.

It is important to note that, almost always, the CPU does not communicate serially with such devices. Instead, the CPU interacts with the device using parallel communications; an interface or the device itself converts the data between serial and parallel forms.

There are several modes of serial communication. **Asynchronous serial communication** is used to interact with devices outside of the computer, such as when connecting to another computer via modem. The connected devices do not share a common clock and must synchronize their data transfers. They transmit individual bytes of data, rather than large blocks.

Synchronous serial transmission is more efficient. It transmits blocks of data in **frames,** which consist of leading transmission information, the data, and trailing transmission information. In spite of the word *synchronous* in the name, the two systems communicating generally do not share a clock. Instead, information is included as part of the transmission that allows the devices to synchronize their clocks.

10.6.1. Serial Communication Basics

When two devices communicate using asynchronous serial transmission, they do not share a common clock. They must have some means of synchronizing the flow of data. In order to do this, they must agree beforehand on several transmission parameters. One of these parameters is the speed, the number of **bits per second, or bps.** (formerly referred to as the **baud rate**). Not agreeing on transmission parameters will cause data to be lost or corrupted. For example, if data is being transmitted at 28,800 bps but the receiver is reading the data at 14,400 bps, half the bits are being lost, including control bits, which will ultimately lead to corruption of the received data. In addition, the two devices must agree on the number of data bits per data transmission, whether or not a **parity bit** is transmitted along with the data (and, if so, whether odd or even parity checking is used), and the number of **stop bits** at the end of the transmission.

In asynchronous serial communication, each byte of data is transmitted as a separate entity. The main problems to be overcome in this transmission mode are for the receiving device to recognize when a transmission is occurring; when to read a bit of data; when the transmission is ending; and when the transmission line is idle, that is, no data is being transmitted.

When the transmission line is idle, its value is logic 1. To signal the start of transmission, the transmitting device outputs a single **start bit** of 0 onto the line. The line is set to 0 for one **bit time**, which is 1 divided by the number of bits per second. (For example, a system that transmits data at 28,800 bps has a bit time of 1 ÷ 28,800 = 34.72 μs.) The receiver notes the 1 to 0 transition on the line, waits ½ bit time (so that it samples data in the middle of the bit transmission to increase reliability), and resamples the value on the transmission line. If it is still 0, the receiver confirms that the initial 1 to 0 transition was a start bit, and not a glitch on the line. It then waits 1 bit time and reads a data bit off of the line, repeating this process for however many data bits are in the transmission. (The two devices agreed on the number of data bits before beginning the transmission.) The least significant bit of data is transmitted first, then the remaining bits are transmitted in order; the most significant bit is transmitted last. If there is a parity bit, the receiver waits the requisite 1-bit time and reads that bit in as well. Whether or not a parity bit is used, the receiver then reads in the stop bit or bits, which must be logic 1. Usually, 1, 1½, or 2 stop bits are used; the number of stop bits indicates the number of bits times for which the tarnsmission line is set to 1, not the number of 1's placed on the line. For a system transmitting data at 28,800 bps that uses 1½ stop bits, the transmitting device sets the line to 1 for 1½ . 34.72 μs = 52.08 μs. Any time after this, the transmitter may begin sending another piece of data. Figure 10.18 shows the value of a transmission line for the transmission of 2 bytes of data with a gap between their transmission. This system uses the N81 setting typical for many modem transmissions; no parity bit; 8 data bits; and 1 stop bit.

Transmitting numerical data this way is pretty straightforward, but character data requires a bit of work. We must encode each character as a unique binary value that both the sender and receiver recognize. Several standards exist for encoding characters. The most commonly used in **ASCII**, the **American Standard Code for Information Interchange.** The ASCII standard specifies 128 characters, as shown in Table 10.1. The first 32 characters are **control codes**, or **nonprinting characters**. They specify characters used in printer and screen control (such as the carriage return), serial transmission (such as the acknowledge), or other device controls. The remaining characters are printable letters, numbers and symbols.

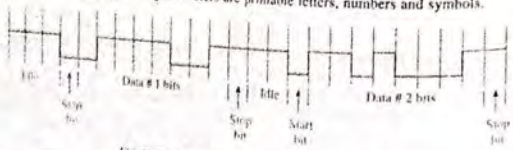


Fig. 10.18. Sample transmission of 2 bytes of data

Since most transmissions use eight data bits and ASCII codes require only seven bits, designers have developed extended ASCII implementations that add 128 more characters to the basic ASCII character set. There are several varieties of extended ASCII characters, which usually include special symbols and characters used in languages other than English.

In addition to ASCII, there are other standards for representing characters, such as **unicode**, which is used by the Java programming language. Unicode uses 16-bit characters and includes characters from many different languages. As with most non ASCII standards, its first 128 characters are identical to those in ASCII.

Table 10.1. ASCII Codes

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

This system introduces a fairly large amount of overhead into the transmission. In the N81 example just given, every 8-bit data chunk requires the transmission of 10 bits, one start bit, the eight data bits, and one stop bit. The start and stop bits account for 20 percent of the total transmission, yet carry no data in and of themselves. They merely synchronize the devices so they can successfully transmit the other eight bits.

Synchronous transmission seeks to improve trasmission performance by reducing this overhead. Instead of sending a start and stop bit for each data value, it concatenates several data values into a single data block. It appends appropriate information in front of and behind the data block to create a frame. This additional information represents overhead but normally at a lower percentage than the overhead associated with asynchronous serial transmission.

A common synchronous serial transmission standard, the **High-level Data Link Control**, or **HDLC**, is shown in Fig. 10.19. The leading and trailing flags are used to synchronize and delimit the frames. The address and control fields specify the address of the destination and the function of the frame. The **cyclic redundancy check (CRC)** is used to catch errors in transmission, just as parity can be used in asynchronous transmissions.

To determine the efficiency of the transmission, we calculate its overhead. If a frame contains 256 data bits, and the other fields contain 48 bits which are treated as overhead, 15.79 percent of the total 304 (256 + 48) bits are overhead. Larger data fields generally required the same overhead as smaller data

fields, so they have lower overhead rates. If the frame contained 256 bytes of data, instead of 256 bits, its overhead would comprise only 2.3 percent of the total bits transmitted.

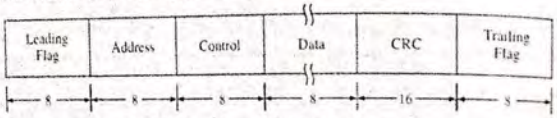


Fig. 10.19. Frame format for high-level data link control

10.6.2. Universal Asynchronous Receiver/Transmitters (UARTs)

Asynchronous serial communication is a common function in computer systems. As with many popular functions, manufacturers have designed special chips to perform these functions and relieve the CPU of this task. For asynchronous serial communication, these specialized chips are called **universal asynchronous receiver/transmitters** or **UARTs**.

As shown in Fig. 10.20, the CPU sees the UART as just another parallel I/O device and interfaces with it accordingly. On its back end, the UART inputs and outputs (receives and transmits) data serially. It can interact with any device that can access serial data. In Fig. 10.20, the UART exchanges data with a modem. To transmit data, the UART outputs sequential data to the modem, which modulates (the "MO" in *modem*) the data, combining it with a carrier frequency and transmitting it. To receive data, the modem accepts a signal at a different carrier frequency and demodulates the ("DEM" in *modem*) it, extracts the data, and transmits it serially to the UART.

As shown in Fig. 10.21, the UART has several internal registers, just as in the DMA controller. Like the DMA controller, the CPU treats these registers as individual I/O ports. A typical UART contains registers to hold transmitted and received data, a control register, and a status register, all of which are accessible to the CPU. It also contains shift registers, which convert data from parallel to serial (for data transmission) and from serial to parallel (for data reception). The UART's control unit also coordinates the insertion and removal of start, parity and stop bits. A command sent from the CPU to the control register of the UART sets the number of data bits, parity and the number of stop bits to be used in the transmission.

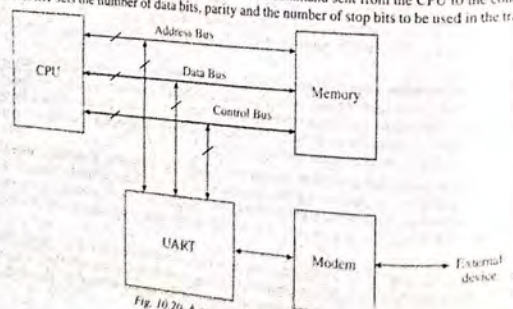


Fig. 10.20. A computer system incorporating a UART

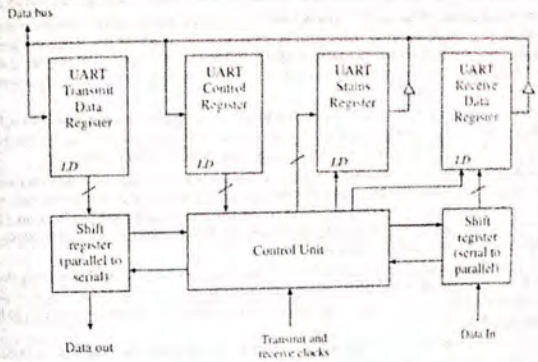


Fig. 10.21.

10.7. REAL WORLD EXAMPLE: SERIAL COMMUNICATION STANDARDS

www.IPUnation.com

The standards for two serial connections typically found on IBM compatible personal computers are RS-232-0C and the universal serial bus standard. The RS-232-C standard has been used by the serial ports of personal computers since their inception in 1981. This standard was originally defined in 1969 and was intended for connecting mainframe computers and terminals. More recently, personal computers have incorporated universal serial bus ports, which connect to up to 127 devices on a single port.

10.7.1. The RS-232-C Standard

The standard serial port of an IBM-compatible personal comuter follows the **RS-232-C standard**. (Apple Macintosh computers follow a different standard.) The RS-232-C standard specifies nine signals, as follows. The serial port of most personal computers can transmit data at up to 115,200 bps.

- Request to send
- Clear to Send
- Transmission Data
- data Terminal Ready
- Data Set Ready
- Received Data
- Data Carrier Detect
- Ring Indicator
- Ground

The first two signals, Request to Send and Clear to Send, are used to specify whether the computer and the device it is communicating with, such as its modem are powered on. When the computer is powered on, it sets the Data Set Ready signal to logic 1. The modem (or other serial device) sets the Data Terminal Ready signal to logic 1 when its power is on. Nothing else happens between these two devices unless both signals are asserted.

The Ring Indicator signal is used by the modem to inform the computer that it detects a ringing on its telephone line. This is used by modems that are set to answer incoming calls, but is not needed when the computer dials out to connect to an internet service provider (ISP) or another computer.

When a modem connects to another computer, it negotiates a connection with the other computer's modem. This includes setting the bit rate of the transmission. (When you dial into an ISP, you may see a message saying "Connected at xxxxx bps;" this is the bit rate negotiated by the two modems.) Once the connection is set, the modem on your computer asserts the Data Carrier Detect signal, which remains asserted as long as the two modems are connected.

Once the modems have established communications, they can transmit data. The computer asserts the Request to Send signal, which tells the modem that it is ready to transmit and receive data. The modem asserts the Clear to Send signal, indicating that it is ready. No transmission occurs unless both signals are asserted.

From here, the computers and modem can transfer data using asynchronous data transfer. The computer sends data to the modem on the Transmission Data line. The modem receives this data, modulates it, and sends it out to the other computer. If the computer transmits data too quickly, the modem deasserts the Clear to Send signal, causing the computer to stop sending data. Once the modem has caught up, it reasserts the Clear to Send signal, enabling the computer to send more data. The modem sends data to the computer via the Received Data signal. Here, the computer deasserts the Request to Send signal if it falls behind in accepting data.

Practical Perspective: The RS-422 Serial Standard

Unlike IBM-compatible personal computers, which use the RS-232-C standard, Apple's Macintosh computers follow the **RS-422 standard** for their serial ports. The signals for this standard are shown in Table 10.2.

Table 10.2. RS-422 Signals

Symbol	Signal name
HSKo	Output Handshake
HSKi/Clock	Input Handshake or External Clock
TxD+	Transmission Data (plus)
TxD-	Transmission Data (minus)
RxD+	Receive Data (plus)
RxD-	Receive Data (minus)
GPi	General Purpose Input
Ground	Signal Ground

The HSKo and HSKi/Clock signals are used to implement handshaking. Their functions are similar to the Request to Send, Data Terminal Ready, and Clear to Send signals of the RS-232-C standard. The General Purpose Input signal is often used like the RS-232-C Data Carrier Detect signal.

Unlike the RS-232-C standard, the RS-422 standard uses **differential voltage** to send data. In this form, transmitted data occupies two signals in each direction. The transmitted value depends on the difference in the voltage on the two signal lines. This improves the quality of the signal, making it more immune to electrical noise, and allows communication over longer distances.

10.7.2. The Universal Serial Bus Standard

The RS-232-C bus does not supply address information to its connected device. Typically, only one device is connected to a single RS-232-C port, so no address is needed. In contrast, the **universal serial bus (USB)**, available on many newer computers, can connect to several devices. The universal serial bus transmits data in packets, similar to the frames described in the previous section. This packet can specify an address, allowing several devices to be connected to one USB port.

The USB port is much faster than the RS-232-C port. The USB specification, version 1.1, allows transmission speeds upto 1.5 Mb/s. The new version 2.0 standard allows for speeds up to 480 Mb/s.

The USB standard specifies four types of packets, which are used to communicate between a computer and its USB peripherals. The **token packets** specify addresses and end points for a transfer, or a frame marker. **Data packets**, as their name implies, contain data transferred to or from a device. The **handshake packets** transfer information used to coordinate data transfers, such as the ACK (acknowledge) packet. Finally, there are **special packets** with several different functions.

Figure 10.22 shows the format for three packets used to transfer data. The **token pocket** is used to initiate data transfers. It specifies the address (ADDR) of the device to send or receive data, and the direction of data flow; the direction is specified in the **packet identifier (PID)**. The other two fields indicate the **end of packet (ENDP)** and a **cyclic redundancy check (CRC)**.

After sending a token packet to initiate transfers, the computer sends a data packet. Each data packet contains a PID, a data field, and a 16-bit CRC field. The data field can contain up to 8192 bits of data. This greatly reduces the overhead, which is normally much less than the 20 percent (or more) overhead for RS-232-C devices. Notice that this packet does not contain an address. The address was specified in the token packet sent before data was transmitted.

After receiving this data, the receiving device sends a handshake packet. This packet contains only an 8-bit PID. If the packet was received without any errors, the receiving device acknowledges successful receipt of the data. If there was an error in the packet, this PID indicates a negative acknowledge. (This happens when the CRC for the transmitted data, as calculated by the receiving device, does not match the CRC transmitted in the data packet.)

Because the USB can support multiple devices, a system that would require several RS-232-C ports might require only one USB. In this respect, it is very hardware efficient. However, there are significantly greater software requirements for USB as opposed to RS-232, since it must distinguish among interconnected devices.

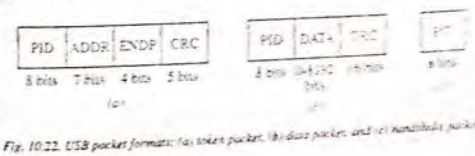


Fig. 10.22. USB packet formats: (a) token packet, (b) data packet, and (c) handshake packet.

SOLVED EXAMPLES

Example 10.1. *Explain the different data transfer schemes.*

Solution. The data transfer scheme on refers to the method of data transfer between the CPU and the peripheral devices. Data transfer takes place between any two devices , CPU and memory, CPU and I/O devices, memory and I/O devices. For effective data transfer between these devices, the timing parameters of the devices should be matched. But most of the devices have incompatible timings. For example an I/O device may be slower than the CPU due to which it cannot be send data to the CPU at the expected time.

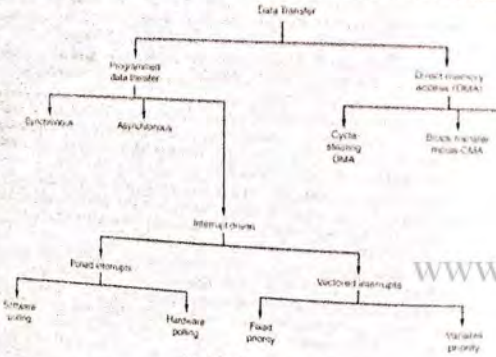


Fig. 10.23 Types of data transfer.

The semiconductor memories are available with compatible timings, moreover, slow memories can be interfaced using additional hardware to introduce wait states in machine cycles. The data transfer schemes have been broadly classified into the following two categories.

1. Programmed data transfer.
2. Direct memory access (DMA) data transfer.

Programmed data transfer scheme is used when relatively small amount of data are to be transferred. In these schemes, usually one type or word of data is transferred at a time.

The programmed data transfer scheme can be further classified into the following three types.

1. Synchronous data transfer.
2. Asynchronous data transfer.
3. Interrupt driven data transfer.

In DMA data transfer, the CPU is forced to HOLD state (high impedance state) by an I/O device until the data transfer between the device and the memory is complete. The CPU does not execute any instructions during the HOLD period.

The DMA data transfer is used for large block of data transfer between I/O device and memory. The different types of DMA data transfer schemes are :

1. Cycle stealing or single transfer mode DMA.
2. Block or burst mode DMA.
3. Demand transfer mode DMA.

Figure 10.18 shows the various types of data transfer schemes. All the data transfer schemes discussed above requires both software and hardware for their implementation. Within a CPU, more than one scheme can be used for interfacing different I/O devices.

Example 10.2. *What is meant by software for programmed I/O?*

Solution. The software that manages data transfer to and from I/O devices is of critical importance. It manages the communications of data and control information between the device and the CPU. This includes initiating, controlling and terminating the data transmission and the handling exceptional events such as 'out of paper' or 'device not ready'. This software module is known as a device driver or device handler.

Example 10.3. *What is the basic advantage of using interrupt initiated data transfer over transfer under program control without an interrupt?*

Solution. The main problem with programmed I/O is that the processor may waste many instructions polling a device to see if it is ready for the next operation. It can be avoided by using an interrupt facility and special commands to inform the interface to issue an interrupt request signal when the data are available from the device. Upon detecting the external interrupt signal, the CPU momentarily stops the task it is processing, branches to a service program to process the I/O transfer and then returns to the task it was originally performing.

Example 10.4. *Why are the read and write control lines in a DMA controller bidirectional? Under what conditions and for what purpose are these used as inputs? Under what condition and for what purpose are they used as outputs?*

Solution. The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU. Removing the CPU from the path and peripheral device manage the memory buses directly would improve the speed of transfer. This transfer technique is called direct memory access (DMA). During DMA transfer, the CPU is idle and has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between the I/O device and memory.

In DMA controller has Read (RD) write (WR) signal in bidirectional. The direction of transfer depends on the status of the bus grant (BG) line. When BG = 0, the RD and WR are input lines allowing the CPU to communicate with the internal DMA register. When BG = 1, the RD and WE are output lines from the DMA controller to the random access memory (RAM) to specify the read or write operation for the data.

Example 10.5. *List four peripheral devices that produce an acceptable output for a person to understand.*

Solution. Character printer; Line printer; Laser printer; Digital plotter; Graphic display; Voice output; Digital to analog converter; Instrument indicator.

www.IPUPC.com