EC7551: COMPUTER ARCHITECTURE AND ORGANIZATION

UNIT I: INTRODUCTION

Presented By,

Dr. V. SATHIESH KUMAR

Assistant Professor

Department of Electronics Engineering

MIT-Anna University

SYLLABUS:

UNIT I INTRODUCTION

Computing and Computers, Evolution of Computers, VLSI Era, System Design- Register Level, Processor Level, CPU Organization, Data Representation, Fixed – Point Numbers, Floating Point Numbers, Instruction Formats, Instruction Types. Addressing modes.

TEXT BOOKS:

1. John P.Hayes, _Computer architecture and Organization‘, Tata McGraw-Hill, Third edition, 1998.
2. V.Carl Hamacher, Zvonko G. Varanesic and Safat G. Zaky, — Computer Organization—, V edition, McGraw-Hill Inc, 1996.

Presentation Slides:

www.sathieshkumar.com/tutorials

COMPUTING AND COMPUTERS

THE NATURE OF COMPUTING:

- Human brain to perform computations.
- People used fingers, pebbles or tally sticks for counting purpose.
- Abacus and slide rule are also used as an computational aid.
- As the size and complexity of the calculations being carried out increases, two serious limitations of manual computation become apparent.

1. The speed at which a human computer can work is limited.
2. Humans are notoriously prone to error, so long calculations done by hand are unreliable unless elaborate precautions are taken to eliminate mistakes.

COMPUTING AND COMPUTERS

THE NATURE OF COMPUTING: THE ELEMENTS OF COMPUTERS

- Every computer, human or artificial, contain the following components:

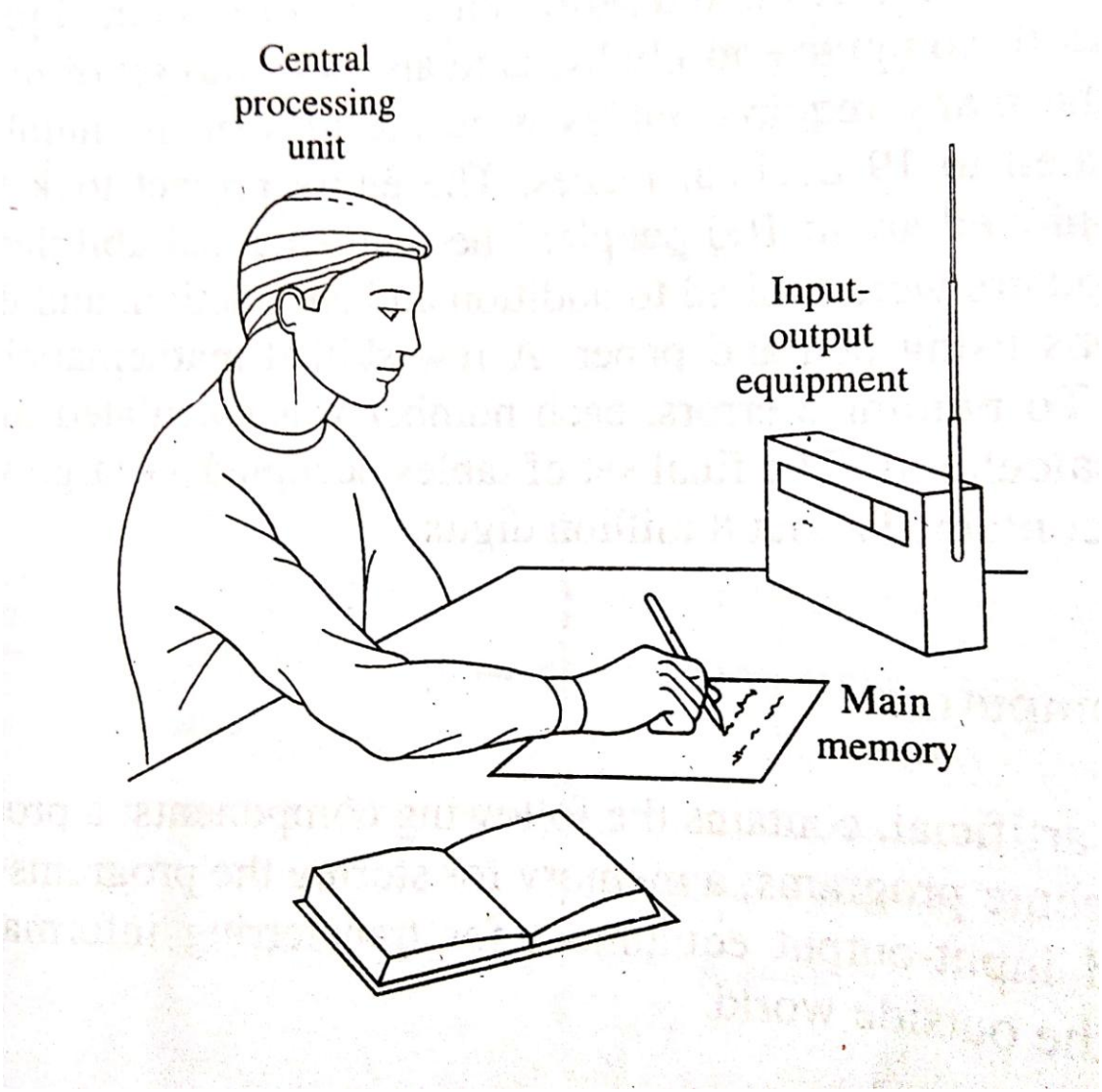
1. A processor able to interpret and execute programs.
2. A memory for storing the programs and the data they can process.
3. Input-output equipment for transferring information between the computer and the outside world.

- Human Computation: Manual calculation using pencil and paper.

1. Paper – Information storage or memory (Program, algorithm or procedure, data - Intermediate or final results are recorded)
2. Human brain – Data processing (central processor) – Performs two basic functions, namely, control function to interpret the instructions and ensures that they are performed in the proper sequence and execution function to perform arithmetic operations

COMPUTING AND COMPUTERS

THE NATURE OF COMPUTING: THE ELEMENTS OF COMPUTERS



COMPUTING AND COMPUTERS

THE NATURE OF COMPUTING: THE ELEMENTS OF COMPUTERS

- Main memory – Paper – To store instructions and data
- Central processing unit – Brain – Program control unit or Instruction unit (Function is to fetch instructions from memory and interpret them) – Arithmetic logic unit (data processing or execution unit)

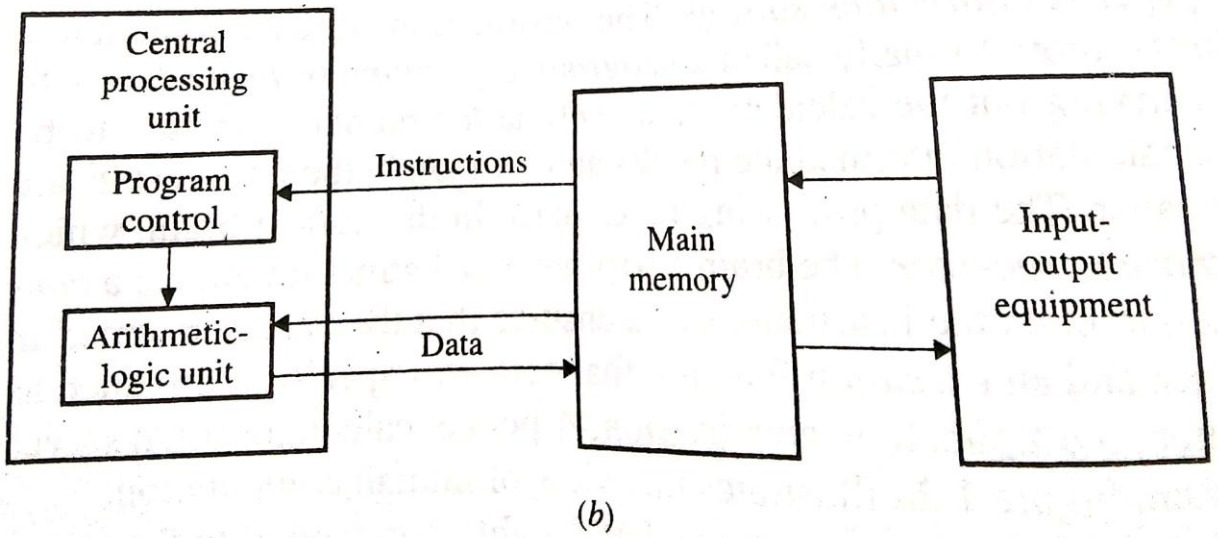


Figure 1.2
Main components of (*a*) human computation and (*b*) machine computation.

COMPUTING AND COMPUTERS

THE NATURE OF COMPUTING: THE ELEMENTS OF COMPUTERS

- Similarities and differences between human being and artificial computers :

1. In both cases information is usually in digital or discrete form, results in high accuracy.
2. Human employ languages with a wide range of digital symbols (decimal – base 10 format) whereas, computer process data in binary form, using the two symbols 0 and 1 called bits (binary digits).
3. To provide communication between an computer and its human users, a means of translating information between human and machine (binary) format is necessary. The input-output equipment perform this task.

COMPUTING AND COMPUTERS

THE NATURE OF COMPUTING: THE ELEMENTS OF COMPUTERS

- An Abstract computer : Computational abilities of general purpose digital computers.
- Three notions of reasonableness are widely accepted :

1. The computer should not store the answers to all possible problems.
2. The computer should only be required to solve problems for which a solution procedure or program can be given.
3. The computer should process information at a finite speed.

- A reasonable computer can therefore solve a particular problem only if it is supplied with a program that can generate the answer in a finite amount of time.

COMPUTING AND COMPUTERS

THE NATURE OF COMPUTING: THE ELEMENTS OF COMPUTERS

- Alan M. Turing introduced an abstract model of a computer – **Turing machine**

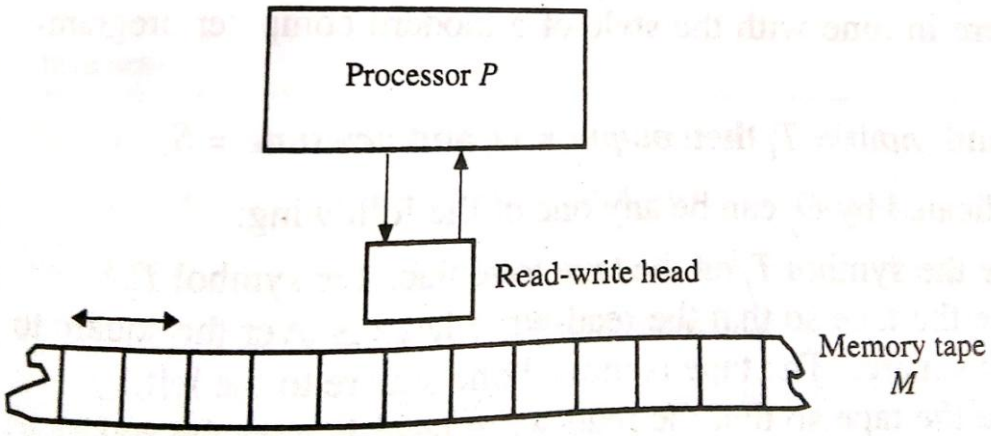


Figure 1.3
A Turing machine.

- Memory – Tape M (is of unbounded length and is divided lengthwise into squares M). Each square can be blank or it can contain one of a small set of symbols.
- Processor P – small device with a small number of internal configurations or states. It is linked to M by a read-write head that can read the content of one square Q and write a new symbol into Q to replace the old one in a single step.

COMPUTING AND COMPUTERS

THE NATURE OF COMPUTING: THE ELEMENTS OF COMPUTERS

- Turing machine as having a set of instructions, four part format,

$$S_h \quad T_i \quad O_j \quad S_k$$

S_h – Present state of processor P (or old state)

T_i – Symbol read by the processor on a square of M using read-write head (or input)

O_j – perform the action (write a new symbol or move the tape) (or output)

S_k – new state of processor.

- The output operation O_j can be any of the following:

1. $O_j=T_j$, meaning write the symbol T_j on the tape to replace the symbol T_i.
2. $O_j=R$, meaning move the tape so that the read-write head is over the square to the right of the current square (The tape is moved one square to the left).
3. $O_j=L$, meaning move the tape so that the read-write head is over the square to the left of the current square (The tape is moved one square to the right).
4. $O_j=H$, meaning halt the computation.

COMPUTING AND COMPUTERS

THE NATURE OF COMPUTING: LIMITATIONS OF COMPUTERS

- Unsolvable problems : Goldbech conjecture – states that every even integer greater than 2 is sum ofexactly two prime numbers (Eg: 8=3+5, 108=37+71). The number of even integers is infinite, so a complete or exhaustive examination of all even integers and their prime numbers is not feasible.
- Undecidable problem: To determine if an arbitrary polynomial equation of the form,

  $a_0+a_1x+a_2x^2+\ldots+a_nx^n = b$

  has a solution consisting entirely of integers. The general program or procedure can never be constructed that can analyze any possible polynomial equation and decide if it has an integer solution.
- Turing machine halting problem is also undecidable (infinite loops).
- The length of the tape memory in Turing machine is infinite and hence the total number of states in the Turing machine is infinite.
- Real computers have finite amount of memory – Finite state machines.

COMPUTING AND COMPUTERS

THE NATURE OF COMPUTING: LIMITATIONS OF COMPUTERS

- Intractable problems: Real (finite state machines) computers can solve most computational problems to an acceptable degree of accuracy.
- Tractable – A computer of reasonable size and cost solve a given problem in a reasonable amount of time.
- Whether a given problem is tractable depends on several factors,

1. The nature of the problem itself
2. The solution method or the program used
3. The computing speed or the performance of the computer available to solve it.

- The given problem is intractable if all its known solution methods grow exponentially with the size of the problem.

COMPUTING AND COMPUTERS

THE NATURE OF COMPUTING: LIMITATIONS OF COMPUTERS

- Influence of hardware technology on computing speed:

Component technology	Date	Number of basic operations per second
Electromechanical: relays	1940	10
Electronic: vacuum tubes (valves)	1945	10^3
Electronic: transistors	1950	10^4
Small-scale integrated circuits	1960	10^5
Medium-scale integrated circuits	1980	10^6
Very large-scale integrated circuits	2000	10^9

Figure 1.5
Influence of hardware technology on computing speed.

COMPUTING AND COMPUTERS

THE NATURE OF COMPUTING: LIMITATIONS OF COMPUTERS

- Speed limitations: An algorithm A has time complexity of order f(n), denoted **O(f(n)).**
- **A** uses to solve a problem of size n is at most **cf(n)**, where f(n) is some function of n and c is a constant.
- The function f(n) therefore indicates the rate at which the computing time that A needs to obtain a solution grows with the problem size n.
- To study the impact of computing speed on the size n_{MAX} of the largest solvable problem, four algorithms A_1, A_2, A_3 and A_4 of varying degrees of difficulties are considered.
- Let the time complexities of A_1, A_2, A_3 and A_4 be **O(n), $O(n^2)$, $O(n^{100})$** and **$O(2^n)$**, respectively.
- A_4 has a time complexity that is exponential in n – intractable procedure.
- All four algorithms are programmed on a computer M having a speed of S basic operations per second.

COMPUTING AND COMPUTERS

THE NATURE OF COMPUTING: LIMITATIONS OF COMPUTERS

- Let n_i denote the size of the largest problem that algorithm A_i can solve in a fixed time period of T seconds.
- Let n_i' denote the size of the largest problem that the same algorithm A_i can solve in T seconds on a new computer M' that is 100 times faster than M; the speed of M' is therefore 100S operations per second.
- M' could be implemented by a different and faster hardware technology than M or be implemented by a "supercomputer" consisting of 100 copies of M all working in parallel on the same problem, a technique referred to as parallel processing.

COMPUTING AND COMPUTERS

THE NATURE OF COMPUTING: LIMITATIONS OF COMPUTERS

Algorithm	Time complexity	Maximum problem size	
		Computer M	Computer M'
A_1	$O(n)$	n_1	$n_1' = 100n_1$
A_2	$O(n^2)$	n_2	$n_2' = 10n_2$
A_3	$O(n^{100})$	n_3	$n_3' = 1.047n_3$
A_4	$O(2^n)$	n_4	$n_4' = n_4 + 6.644$

COMPUTING AND COMPUTERS

THE NATURE OF COMPUTING: LIMITATIONS OF COMPUTERS

➢ Devise approximate or inexact methods to solve the intractable problems. Two major techniques follow.

1. We replace the intractable problem Q with a tractable problem Q' whose solution approximates that of Q.
2. We examine a relatively small set of possible solutions to Q using reasonable, intuitive, and often poorly understood selection criteria and take the "best" of these as the solution to Q. Methods that are designed to produce acceptable, if not optimal, answers using a reasonable amount of computing time are sometimes called heuristic procedures.

COMPUTING AND COMPUTERS

THE NATURE OF COMPUTING: LIMITATIONS OF COMPUTERS

- Computers are continually being applied to new problems whose computational requirements far exceed those of older problems.
- Example: Processing of high-quality speech and visual images for multimedia applications can require speeds measured in trillions of basic operations per second.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : THE MECHANICAL ERA

- 16th Century – Calculating machines capable of performing the elementary operations of arithmetic (addition, subtraction, multiplication and division).
- Blaise Pascal – invented an early and influential mechanical calculator that could add and subtract decimal numbers. Used gears, levers etc.
- Gottfried Leibniz – extended Pascal's design to one that could also perform multiplication and division.
- Babbage Difference Engine : First computer was designed in 19th Century to perform multistep operations automatically, that is, without a human intervening in every step.
- Technology is entirely mechanical.
- It was intended to compute and print mathematical tables automatically.
- It performed only one arithmetic operation – Addition.
- However, the method of (finite) differences embodied in the Difference Engine can calculate many complex and useful functions by means of addition alone.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : THE MECHANICAL ERA

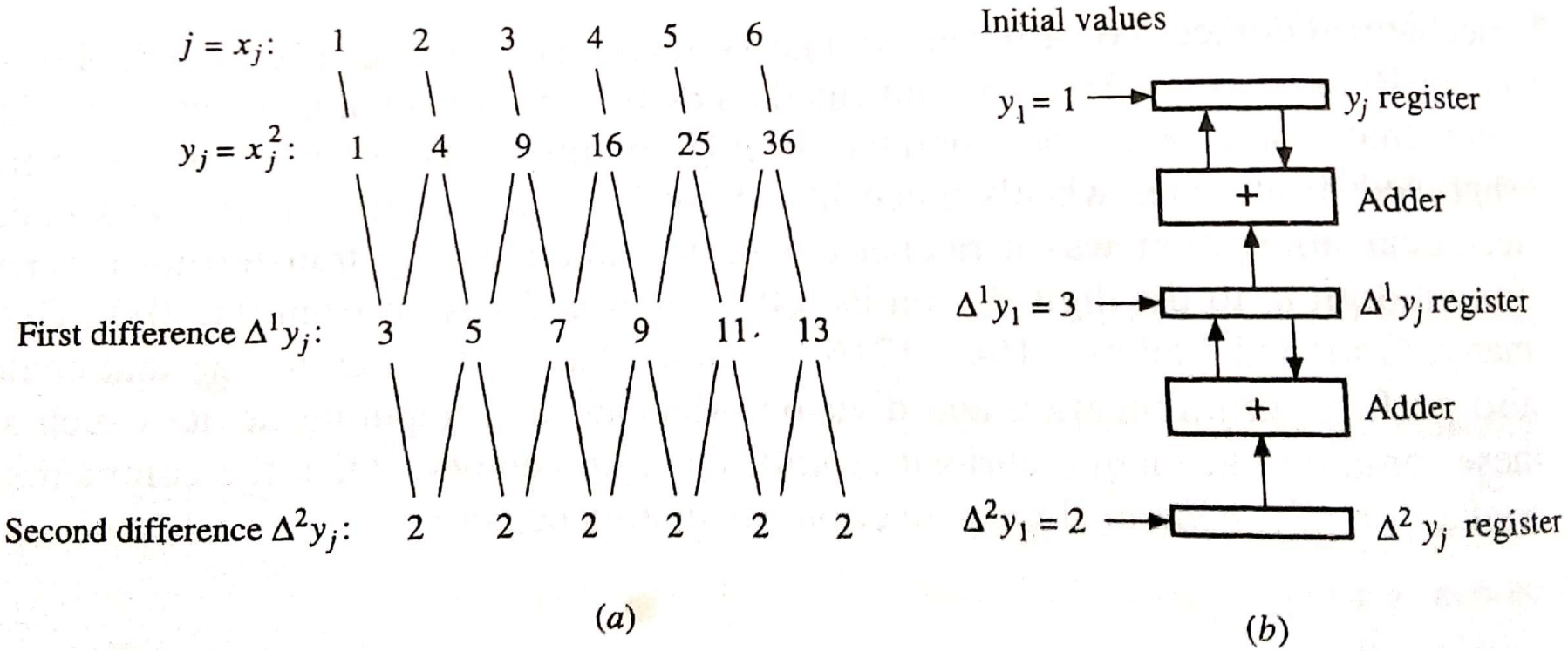


Figure 1.8
Computing x^2 by the method of differences: (*a*) a representative computation and (*b*) the corresponding Difference Engine configuration.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : THE MECHANICAL ERA

- Computing x^2 by the method of differences.
- Calculating a table of squares $y_j=x_j^2$, for $x_j=1, 2, 3,\ldots$ using the method of differences.
- First difference of y,

$$\Delta^1 y_j = (x_j + 1)^2 - x_j^{\,2} = 2x_j + 1$$

- Second difference of y,

$$\Delta^2 y_j = 2(x_j + 1) + 1 - (2x_j + 1)$$

$$\Delta^1 y_{j+1} = \Delta^1 y_j + \Delta^2 y_j$$

$$y_{j+1} = (x_j + 1)^2 = y_j + \Delta^1 y_j$$

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : THE MECHANICAL ERA

- The Analytical Engine : Designed by Charles Babbage.
- This machine considered to be the first general purpose programmable computer ever designed.

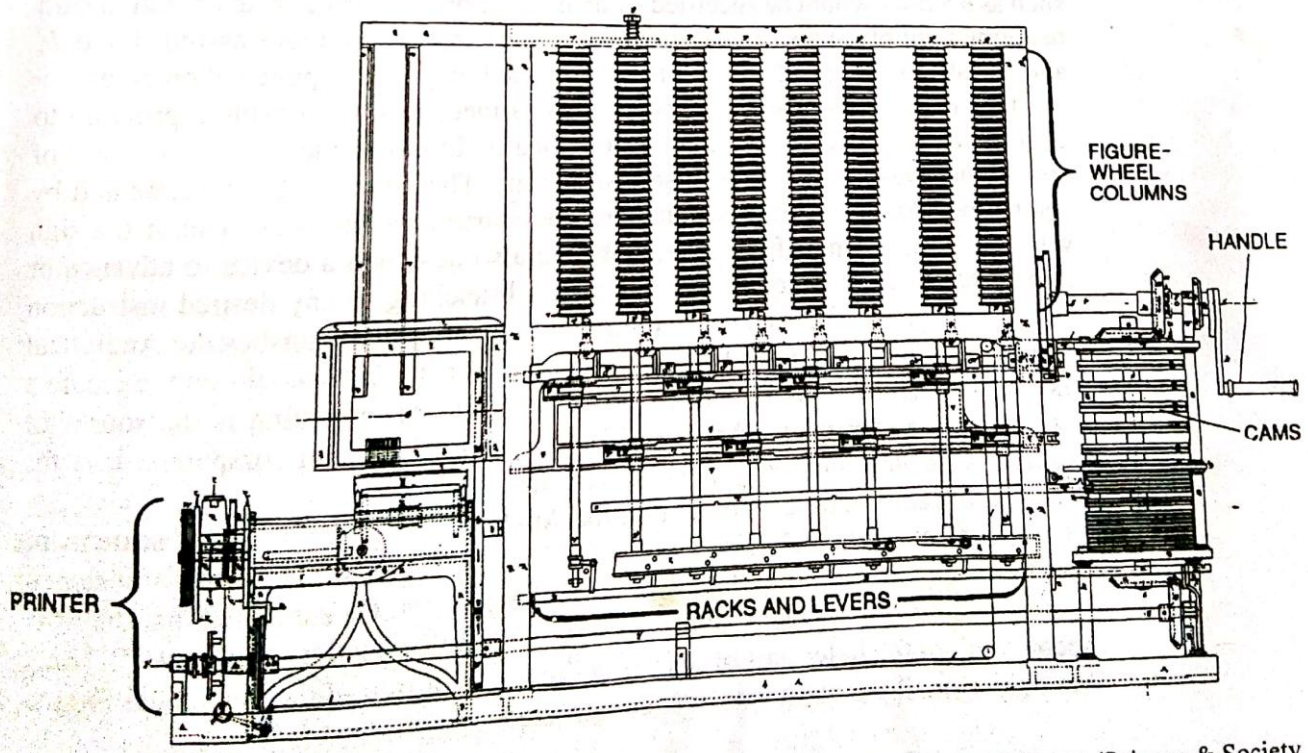


Figure 1.9
Diagram by Babbage of Difference Engine No. 2 [Courtesy of the National Science Museum/Science & Society Picture Library].

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : THE MECHANICAL ERA

➢ The Analytical Engine : The main components are a memory called the **store** and an ALU called **the mill** (can perform four basic arithmetic operations).

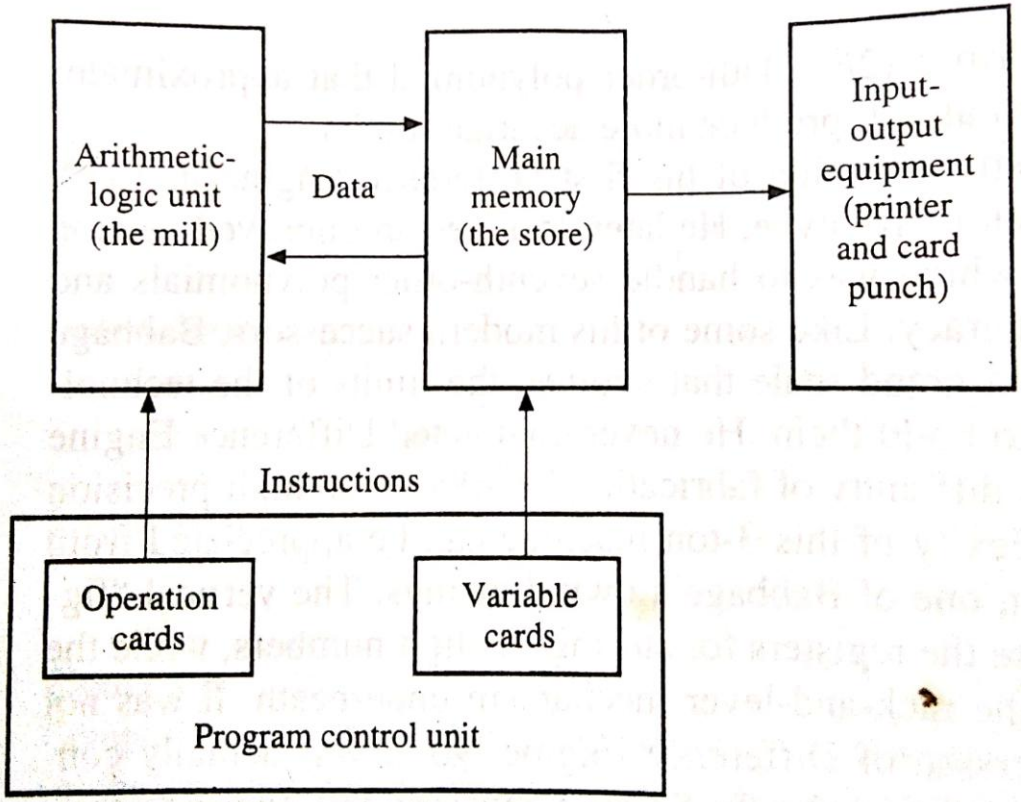


Figure 1.10
Structure of Babbage's Analytical Engine.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : THE MECHANICAL ERA

- A program for the Analytical Engine was composed of two sequences of punched cards : **operation cards** used to select the operation to be performed by the mill, and **variable cards** to specify the locations in the store from which the inputs were to be taken or results sent.
- Babbage intended the results to be printed on paper or punched on cards.
- Key innovations are : mechanism to enable a program to alter the sequence of its operations automatically.
- He also designed the device to advance or reverse the flow of punched cards to permit branching to any desired instruction within the program.
- This type of conditional branching distinguishes the Analytical Engine from Difference Engine.
- The store have a capacity of a thousand 50-digit numbers.
- Addition of two numbers would take a second, and multiplication, a minute.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : THE MECHANICAL ERA

- In Germany, Konrad Zuse built a small mechanical computer, the Z1.
- The Z1 used binary instead of decimal, arithmetic.
- The subsequent version Z3 is believed to have been the first operational general purpose computer.
- Drawbacks of Mechanical Computers are :

1. Its computing speed is limited by the inertia of its moving parts.
2. The transmission of digital information by mechanical means is quite unreliable.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

- The "moving parts" in electronic computers are electrons, which can be transmitted and processed reliably at speeds approaching that of light (300,000 km/s).
- Development of Vacuum tubes or electronic valve permit the processing and storage of digital signals at speeds far exceeding those of mechanical device.
- The first Generation : **Electronic Numerical Integrator and Calculator (ENIAC)** – First widely known general purpose electronic computer was designed by John W. Mauchly and J. Presper Eckert at University of Pennysylvania.
- Enormous machine weighing 30 tons and containing more than 18000 vacuum tubes.
- It was substantially faster than any previous computer.
- ENIAC required only 3 ms to perform 10-digit multiplication.
- ENIAC had a set of electronic memory units called **accumulators** with a combined capacity of twenty 10-digit decimal numbers.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

- Each digit was stored in a 10-bit ring counter, where the binary pattern 1000000000 denoted the decimal digit 0, 0100000000 denoted 1, 0010000000 denoted 2, and so on.
- ENIAC's accumulators combined the function of storage with addition and subtraction.
- Additional units performed multiplication, division and the extraction of square roots.
- Programmed by plugging and unplugging cables and by manually setting a master programming unit to specify multistep operations.
- Results were punched on cards or printed on an electric typewriter.
- ENIAC – Separate memories for storing program and data.
- Entering or altering the programs was a tedious task.
- John von Neumann proposed the idea of storing programs and data in the same high speed memory – **the stored program concept**.
- Resulted in a new model of computer namely, Electronic Discrete Variable Computer (EDVAC).

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

- EDVAC stored and processed the numbers in binary or base 2 form.
- To minimize the hardware costs, data was processed serially or bit by bit.
- It had two kinds of memory: a fast main memory with a capacity of 1024 or 1K words (numbers or instructions) and a slower secondary memory with a capacity of 20K words.
- Prior to their execution, a set of instructions forming a program was placed in the EDVAC's main memory.
- The instructions were then transferred one at a time from the main memory to the CPU for execution.
- Each instruction had a well defined structure of the form,

A_1 A_2 A_3 A_4 OP

A_1 and A_2 – contents of main memory locations or addresses.

A_3 – Place the result in the address

A_4 – location of the next instruction to be executed, OP- operation (+, -, * etc)

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

- Another instruction type specifies input-output operations that transfer words between main memory and secondary memory or between secondary memory and a printer.
- **IAS Computer** – Von Neumann and his colleges designed a new stored program electronic computer.

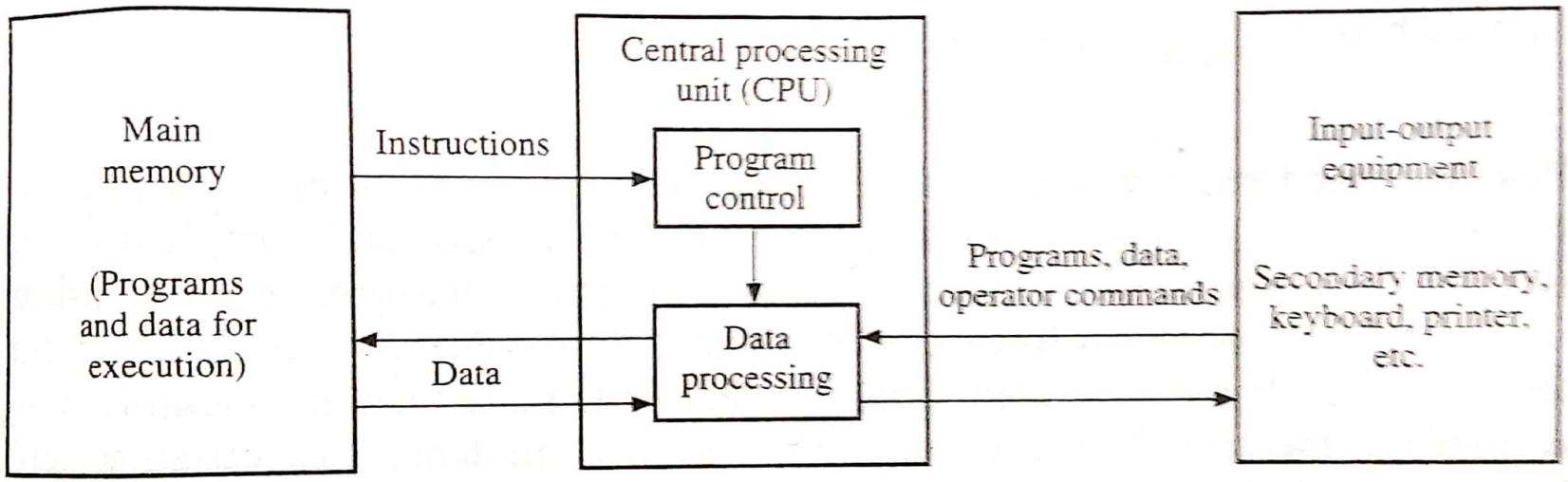


Figure 1.11
Organization of a first-generation computer.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

- CPU for executing instructions
- a main memory for storing active programs
- a secondary memory for backup storage,
- and miscellaneous input-output equipment.
- IAS machine was designed to process all bits of a binary number simultaneously or in parallel.
- First generation computers experimented with various technologies for main and secondary memory.
- In Whirlwind computers – the ferrite core memory in which a bit of information was stored in magnetic form on a tiny ring of magnetic material.
- Earliest computers had their instructions written in a binary code known as **machine language** that could be executed directly.

00111011000000001001100100000111

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

- Machine languages programs are extremely difficult for humans to write and so are very error prone.
- Hence resulted in the development of writing the instructions in symbolic format, referred to as **an assembly language**.

ADD X1, X2

- An assembly language requires a special system program (an **assembler**) to translate it into machine language before it can be executed.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

The Princeton IAS Computer:

- Small instruction set and simple design.
- Fast main memory with a initial capacity of 1K words but expandable to 4K.
- a larger 16K words secondary memory based on electromechanical magnetic drum technology.
- The basic unit of information is a 40-bit word, which is the standard packet of information stored in a main memory location or transferred in one step between the CPU and main memory M.
- Each location in M can be used to store either a single 40-bit number or else a pair of 20-bit instructions.
- Number format : fixed-point, meaning that it contains an implicit binary point in some fixed position.
- Numbers are usually treated as signed binary fractions lying between -1 and +1.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

The Princeton IAS Computer:

- Numbers that lie outside the range + or -1 must be suitable scaled for processing by IAS.
- An IAS instruction consists of an 8-bit opcode (operation code) **OP** followed by a 12-bit address **A** that identifies one of up to 2^{12}=4K 40-bit words stored in M.
- One address instruction format,

  OP A

- The IAS's shorter format clearly saves memory space.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

The Princeton IAS Computer:

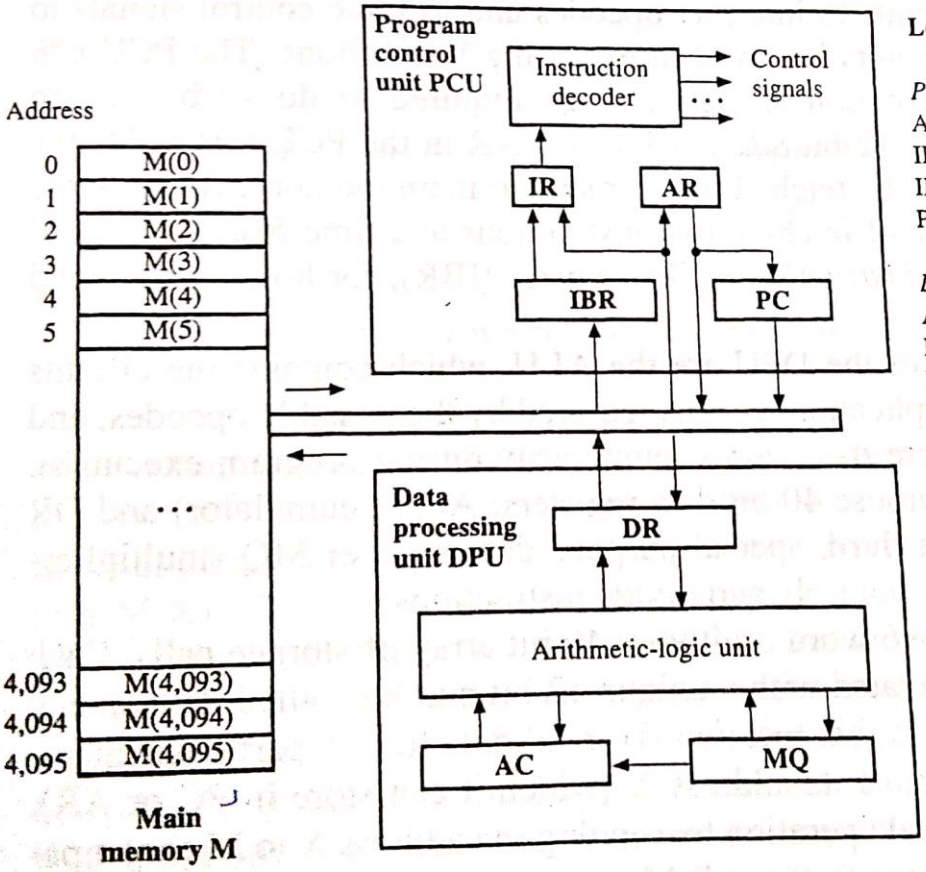


Figure 1.12
Organization of the CPU and main memory of the IAS computer.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

The Princeton IAS Computer:

- Two key aspects of IAS design are:

1. The CPU contains a small set of high-speed storage devices called **registers**, which serve as implicit storage locations for operands and results.
2. A program's instructions are stored in M in approximately the sequence in which they are executed. Hence the address of the next instruction word is usually that of the current instruction plus one. **PC (Program counter)** stores the address of the current instruction word and is increments by one when the CPU needs a new instruction word. Branch instructions are provided to permit the instruction execution sequence to be varied.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

The Princeton IAS Computer:

- Two main parts of CPU are **program control unit (PCU)** or **I-unit (instruction unit)** and data processing unit (DPU) or **E-unit (execution unit).**

1. PCU unit – responsible for fetching instructions from main memory and interpreting them.

- main components of PCU are **: instruction register (IR)** – which stores the opcode that is currently being executed, **program counter (PC)** – which automatically stores and keeps track of address of the next instruction to be fetched, **12-bit address register (AR)** holds the address of a data operand to be fetched from or sent to main memory. Two instructions are fetched at a time from M, hence the second instruction is stored in the **instruction buffer register (IBR).**

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

The Princeton IAS Computer:

2. DPU unit – responsible for executing instructions.

➢ main components of DPU are : **ALU** which contains the circuits to perform addition, multiplication etc, several data registers to store data words temporarily during program execution, two general purpose 40-bit data registers namely, **AC (accumulator)** and **DR (data register)**, it also has a third, special purpose data register **MQ (multiplier-quotient)** intended for use by multiply and divide instructions.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

Instruction set:

- 30 types of instructions
- To represent instructions, we will use a notation called a **hardware description language (HDL)** or **register-transfer language (RTL)** that approximates the assembly language used to prepare programs for the computer.

Instruction	Comment
AC := M(100)	Load the contents of memory location 100 into the accumulator.
AC := AC + M(101)	Add the contents of memory location 101 to the accumulator.
M(102) := AC	Store the contents of the accumulator in memory location 102.

Figure 1.13
An IAS program to add two numbers stored in main memory.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

Instruction type	Instruction	Description
Data transfer	AC := MQ	Transfer contents of register MQ to register AC.
	AC := M(X)	Transfer contents of memory location X to AC.
	M(X) := AC	Transfer contents of AC to memory location X.
	MQ := M(X)	Transfer M(X) to MQ.
	AC := –M(X)	Transfer minus M(X) to AC.
	AC := |M(X)|	Transfer absolute value of M(X) to AC.
	AC := –|M(X)|	Transfer minus |M(X)| to AC.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

Data processing	AC := AC + M(X)	Add M(X) to AC putting the result in AC.
	AC := AC + |M(X)|	Add absolute value of M(X) to AC.
	AC := AC – M(X)	Subtract M(X) from AC.
	AC := AC – |M(X)|	Subtract |M(X)| from AC.
	AC.MQ := MQ × M(X)	Multiply MQ by M(X) putting the double-word product in AC and MQ.
	MQ.AC := AC ÷ M(X)	Divide AC by M(X) putting the quotient in AC and the remainder in MQ.
	AC := AC × 2	Multiply AC by two (1-bit left shift).
	AC := AC ÷ 2	Divide AC by two (1-bit right shift).

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

Program control	**go to** *M(X, O:19)*	Take next instruction from left half of M(X)
	go to *M(X, 20:39)*	Take next instruction from right half of M(X).
	if *AC ≥ 0* ***then*** **go to** *M(X, 0:19)*	If AC contains a nonnegative number, then take next instruction from left half of M(X).
	if *AC ≥ 0* ***then*** **go to** *M(X, 20:39)*	If AC contains a nonnegative number, then take next instruction from right half of M(X).
	M(X, 8:19) := AC (28:39)	Replace left instruction address field in M(X) by 12 right-most bits of AC.
	M(X, 28:39) := AC(28:39)	Replace right instruction address field in M(X) by 12 right-most bits of AC.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

Instruction execution :

- The IAS fetches and executes instructions in several steps that form an **instruction cycle**.
- Since two instructions are packed into a 40-bit word, the IAS fetches two instructions in each instruction cycle.
- One instruction has its opcode placed in the instruction register (IR) and its address field placed in the address register (AR).
- The other instruction is transferred to the IBR register for possible later execution.
- Whenever the next instruction needed by the CPU is not in IBR, the program counter (PC) is incremented to generate the next instruction address.
- Once the desired instruction has been loaded into the CPU, its execution phase begins.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

- The PCU decodes the instructions opcode, and the subsequent actions depend on the opcode's bit pattern.
- Typically, these actions involve one or two register-transfer (micro) operations of the form,

$$S:=f(S_1,S_2,\ldots,S_k),$$

Where S_i's are the locations of operands and f is a data-transfer or arithmetic operations.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

Second Generation (1954 - 64):

- Transistors replaced vacuum tubes.
- Transistor serves as a high-speed electronic switch for binary signals, but it is smaller, cheaper, sturdier and requires much less power than a vacuum tube.
- Transistor based memories replaced ferrite core memories.
- Magnetic disks are used as secondary memory.
- More registers were added to the CPU to facilitate data and address manipulations.
- Example Index registers C(I):=A(I)+B(I) - Indexed Instruction
- Two program control instructions were added – **call** and **return** – for linking of programs.
- IBM 7094 – introduced floating point number formats and supporting instructions to facilitate numerical processing.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

Second Generation (1954 - 64):

- A floating-point number consists of a pair of fixed-point numbers, a mantissa M and an exponent E and has the value $\mathbf{MxB^{-E}}$
- Eg: 0.0000000709 is represented as $7.09x10^{-8}$
- M and E are encoded in binary an embedded in a word of suitable size, the base B is implicit.
- Floating point numbers eliminate the need for number scaling.
- IO operation – Transfer of information to and from peripheral devices can severely degrade overall computer performance if done inefficiently.
- Transfer can take place via CPU.
- IO device transfer data at low speeds compared to that of CPU because of their inherent reliance on electromechanical rather than electronic technology.
- Thus CPU is Idle for long time.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

Second Generation (1954 - 64):

- IBM 7094 introduced **input-output processors (IOPs)** – special type of processing units designed exclusively to control IO operations.
- It utilizes the registers in the IOP processor during data transfer.
- IO data transfer take place independently of the CPU, permitting the CPU to execute user programs while IO operations are taking place.
- **Programming Languages:** High level programming languages.
- A **high-level language** is intended to be usable on many different computers.
- **Compiler** – translates a user program from the high-level language to machine language of the particular computer on which the program is to be executed.
- First high level language : FORTRAN – Formula Translation.
- FORTRAN permits the specification of numerical algorithms in a form approximating normal algebraic notations.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

Second Generation (1954 - 64):

➢Eg: FORTRAN Vector Addition: C(1:1000) = A(1:1000)+B(1:1000)

```
      DO 5 I = 1,1000
5     C(I) = A(I)+B(I)
```

➢ Other High-level languages are COBOL, Basic, Pascal, C and JAVA.

➢System Management : **Batch Processing** is introduced.

➢ In early days, All programs or jobs were run separately, and the computer had to be halted and prepared manually for each new program to be executed.

➢In second generation computers, a batch of jobs are prepared in advance, stored on magnetic tape and then have the computer process the jobs in one continuous sequence, placing the results on another magnetic tape.

➢ Batch processing requires the use of a supervisory program called **batch monitor**, which is permanent resident in main memory – rudiment version of operating system.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

Second Generation (1954 - 64):

- Single CPU is designed to process a set of independent user programs concurrently, a technique called **multiprogramming**.
- Multiprogramming is accomplished by the CPU temporarily suspending execution of its current program, beginning execution of a second program, and returning to the first program later.
- Multiprogrammed computers that process many user programs concurrently and support users at interactive terminals or workstations are sometimes called **time-sharing systems**.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

Third Generation:

- Introduction of Integrated Circuits (ICs)
- ICs allowed large number of transistors and associated components to be combined on a tiny piece of semiconductor material, usually silicon.
- Smaller size, higher speed and lower hardware costs.
- The cost of writing and maintaining programs for a particular computer – the software cost – began to exceed that of the computer hardware.
- Switching to a different computer and making one's old software obsolete was thus an increasingly unattractive proposition.
- IBM developed **System 360** – software compatible (all models in the series shared a common instruction set) versions of the model is being developed.
- Only the execution time, memory usage will change from one version to another.
- OS/360 – operating system used in System360

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

Third Generation:

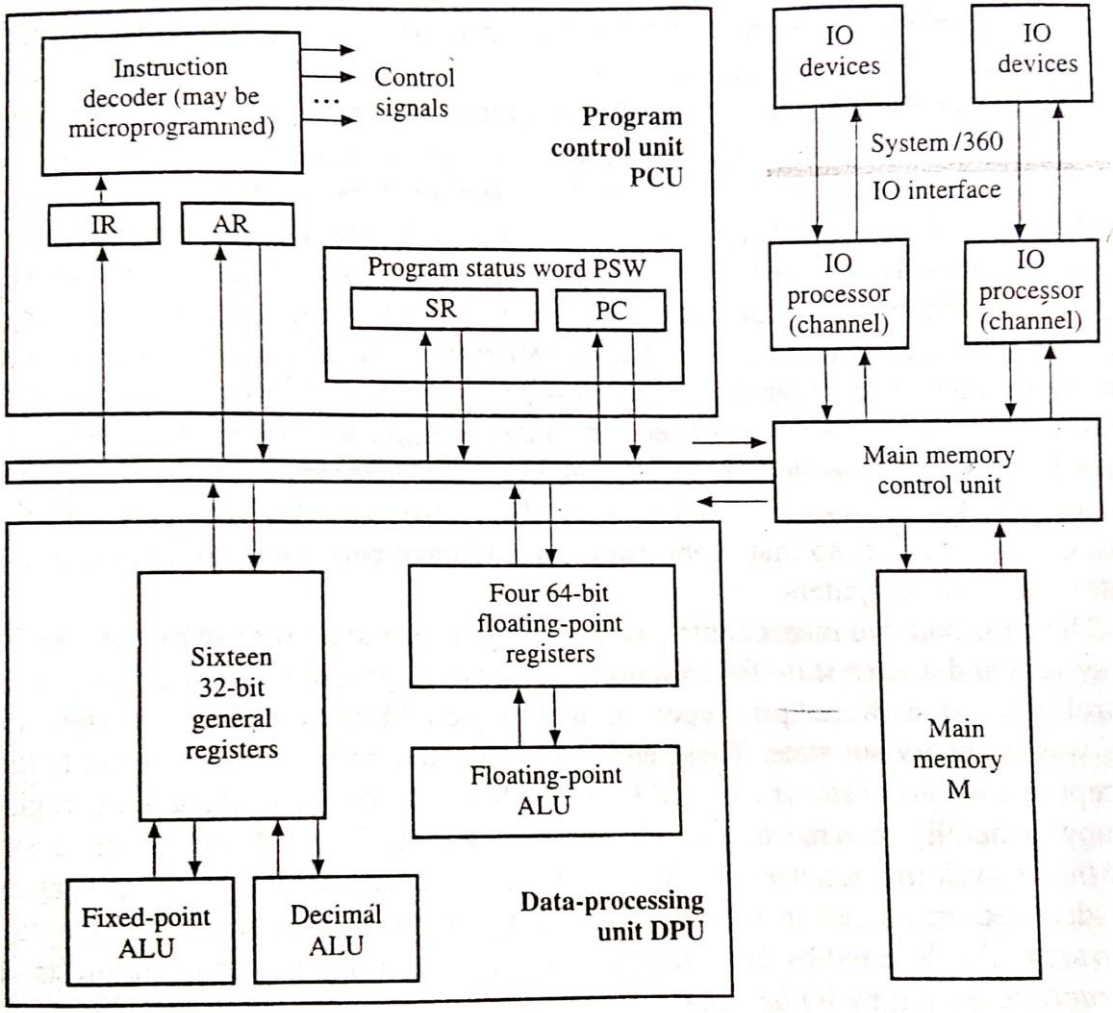


Figure 1.17
Structure of the IBM System/360.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

Third Generation:

- it became a de-facto standard for mainframe computers.
- 200 distinct instruction types, many addressing modes and data types, including fixed and floating point numbers of various sizes.
- 8-bit unit **byte** was defined as the smallest unit of information for data transmission and storage.
- 32 bits (4 bytes) the main CPU **word** size.
- The CPU had two major control states : **a supervisor state** for use by the operating system and a **user state** for executing application programs.
- Privileged mode (certain program control instructions could be executed only in this mode).

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

Third Generation:

➢**SR (Status register)** – encapsulates the key information used by the CPU to record exceptional conditions such as CPU-detected errors, hardware faults detected by error-checking circuits, and urgent service request or interrupts generated by IO device.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA

- ICs containing thousands or millions of transistors – **very large scale integration (VLSI).**
- It allows the manufacturers to fabricate a CPU, main memory, or even all the electronic circuits of a computer on a single IC that can be mass-produced at a very low cost.
- Supercomputers with thousands of CPUs are developed.
- A **multichip module** is a package containing several IC chips attached to a substrate that provides mechanical support, as well as electrical connections between the chips.
- Packaged ICs are often mounted on a printed circuit board (PCB).
- A contemporary computer consists of a set of ICs, a set of IO devices and a power supply.
- No of ICs can range from one IC to several thousand, depending on the computers size and the IC types it uses.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA

➢**IC density :** Number of transistors contained in the chip.

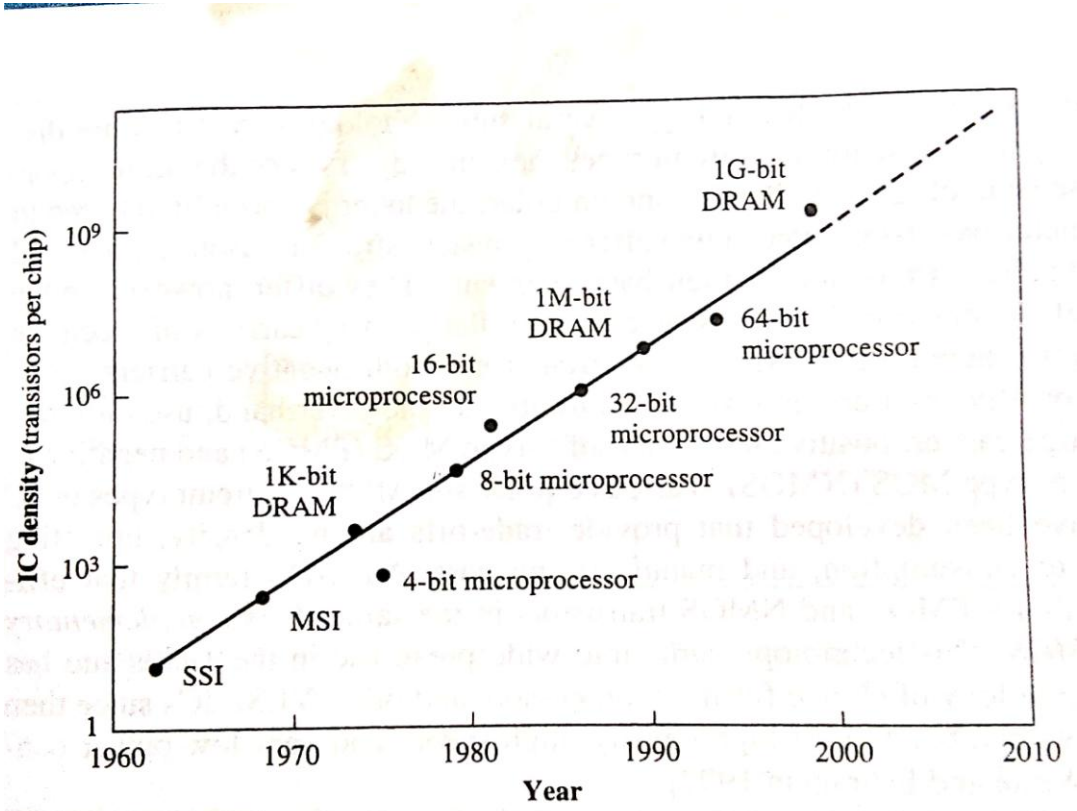


Figure 1.19
Evolution of the density of commercial ICs.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA

- **SSI** or small scale integration – fewer than 100 transistors.
- **MSI** or Medium scale integration – hundred transistors
- **LSI** or large scale integration – thousand transistors
- **VLSI** or very large scale integration – millions of transistors
- Two of the densest chip types are : **dynamic random access memory (DRAM)** and a **single-chip CPU** or **microprocessor**.
- 1K=2^{10} bits – DRAM
- 1M=2^{20} bits – DRAM
- Intel 4004 – 4-bit CPU
- The combination of a CPU, memory and IO circuits in one IC (or a small number of ICs) is called a **microcomputer**.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA

- IC Technologies : **Bipolar** and **Unipolar (MOS** – metal oxide semiconductor).
- Bipolar circuits use both negative carriers (**electrons**) and positive carriers **(holes**).
- MOS circuits use only one type of charge carrier : positive in the case of **P-type MOS (PMOS)** and negative in the case of **N-type MOS (NMOS).**
- A MOS family that efficiently combines PMOS and NMOS transistors in the same IC is **complementary MOS** or **CMOS**. – It resulted in high density, high speed and very low power consumption.
- Behavior (analyze the computing functions) of a digital circuit at a low level of abstraction called the **switch level**.
- Higher level of abstraction are **gate** or **logic level**, **register** or **register transfer level**.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA

- At the gate level or logic level, we represent certain common sub circuits by symbolic components called **logic gates.**
- Register level treats the entire circuit (zero-detection circuit) as a primitive or indivisible component.

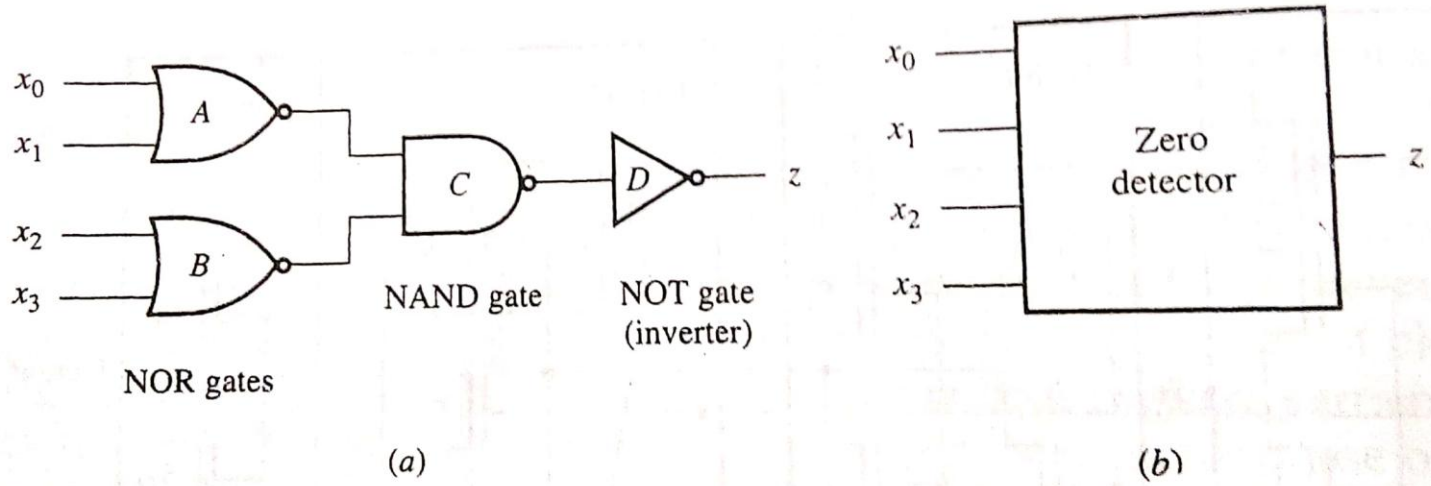


Figure 1.21
The zero-detection circuit of Figure 1.20 modeled at (*a*) the gate level and (*b*) the register level of abstraction.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA

- When we treat an entire CPU, memory or computer as a primitive component – highest level of abstraction, which is called the **processor** or **system level**.
- Processor Architecture: In 1980 – Computers were classified into three main types :

1. Mainframe computers – large computer, containing thousands of ICs. It served as a central computing facility for an organization or university.
2. Minicomputer – smaller computer (desk size), slower version of mainframe computers.
3. Microcomputer – even smaller, slower and cheaper, packing all the electronics of a computer into a handful of ICs, including microprocessor (CPU), memory and IO chips.

- Microcomputers gave rise to a new class of general purpose machines called **personal computers (PCs)**, which are intended for a single user.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA

➢ The more powerful desktop computers intended for scientific computing are referred as **workstations**.

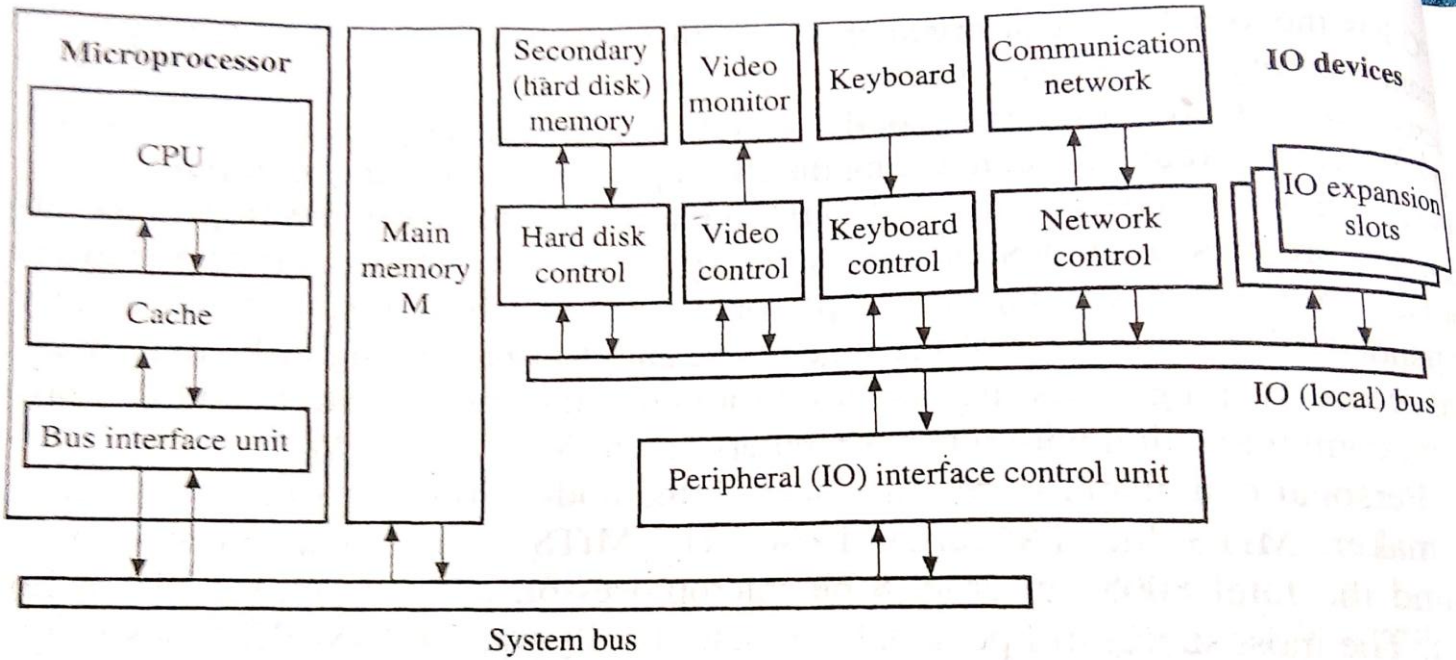


Figure 1.22
A typical personal computer system.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA

- CPU – Pentium or PowerPC
- System Bus – connects the microprocessor to a main memory based on semiconductor DRAM technology and to an IO subsystem.
- A separate IO bus (**PCI** – peripheral component interconnect) – connects directly to the IO devices and their individual controllers.
- **Bridge** – used to interconnect peripherals, microprocessor and memory.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA : PERFORMANCE CONSIDERATIONS

- Computer designers increased the use of complex, multistep instructions. This reduces the total number instructions that must be executed for a given task.
- **Complex instruction set computers (CISC)**
- 8086 microprocessor contains 20000 transistors – processed 16 bit data words and had no instructions for operating on floating point numbers.
- Pentium – over 3 million transistors – processed 32-bits and 64-bits words and executed a comprehensive set of floating point numbers.
- CISC computers reduce program size but does not translate into faster program execution.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA : PERFORMANCE CONSIDERATIONS

- **Reduced instruction set computers (RISC)** were introduced in 1980 – IBM RISC System/6000 and SPARC.
- It is designed for achieving high performance.
- Performance is also strongly affected by other factors besides its instruction set, especially the time required to move instructions and data between the CPU and main memory M and, to a lesser extent, the to move information between M and IO devices.
- RISC computers usually limit access to main memory to a few load and store instructions, other instructions, including all data processing and program-control instructions, must have their operands in CPU registers – **load-store architecture**.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA : PERFORMANCE MEASURES

- CPU speed is the number of "basic" operations that it can perform per unit of time.
- R1:=R1+R2
- Such operations are timed by a regular stream of signals (ticks or beats) issued by a central timing signal, the **system clock**.
- The speed of the clock is its **frequency f** measured in millions of ticks per second; the unit for this are megahertz (MHz).
- Each tick of the clock triggers a basic operation; hence the time required to execute the operation if 1/f (μs).
- Complicated operations such as division or operations on floating-point numbers require more than one clock cycle to complete their execution.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA : PERFORMANCE MEASURES

- The CPU's processing of an instruction involves several steps, each of which requires at least one clock cycle:

1. Fetch the instruction from main memory M
2. Decode the instruction opcode
3. Load (read) from M any operands needed unless they are already in CPU registers
4. Execute the instruction via a register to register operation using an appropriate functional unit of the CPU, such as a fixed-point adder.
5. Store (write) the results in M unless they are to be retained in CPU registers.

- The fastest instruction have all their operands in CPU registers and can be executed by the CPU in a single clock cycle.
- The slowest instructions require multiple memory accesses and multiple register to register operations to complete their execution.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA : PERFORMANCE MEASURES

- Let us consider a benchmark program Q takes T seconds on a CPU to execute a total of N machine (object) instructions.
- **N** is the actual number of instructions executed, including repeated executions of the same instructions; it is not the number of instructions appearing in Q.

$$T=N/IPS \text{ s}$$

IPS – average number of instructions executed per second.

$$T = \frac{N \times CPI}{f \times 10^6} \text{ s}$$

CPI – average number of cycles per instructions $=(fx10^6)/IPS$

$$MIPS=f/CPI$$

MIPS – millions of instructions executed per second

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA : PERFORMANCE MEASURES

- Three factors determine the computer performance are:

1. Software : The efficiency with which the programs are written and compiled into object code influences N, the number of instructions executed.
2. Architecture : The efficiency with which individual instructions are processed directly affects CPI, the number of cycles per instruction executed.
3. Hardware : The raw speed of the processor circuits determine f, the clock frequency. Increasing f tends to reduce T.

- CISC – reduce N at the expense of CPI
- RISC – reduce CPI at the expense of N

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA : SPEEDUP TECHNIQUES

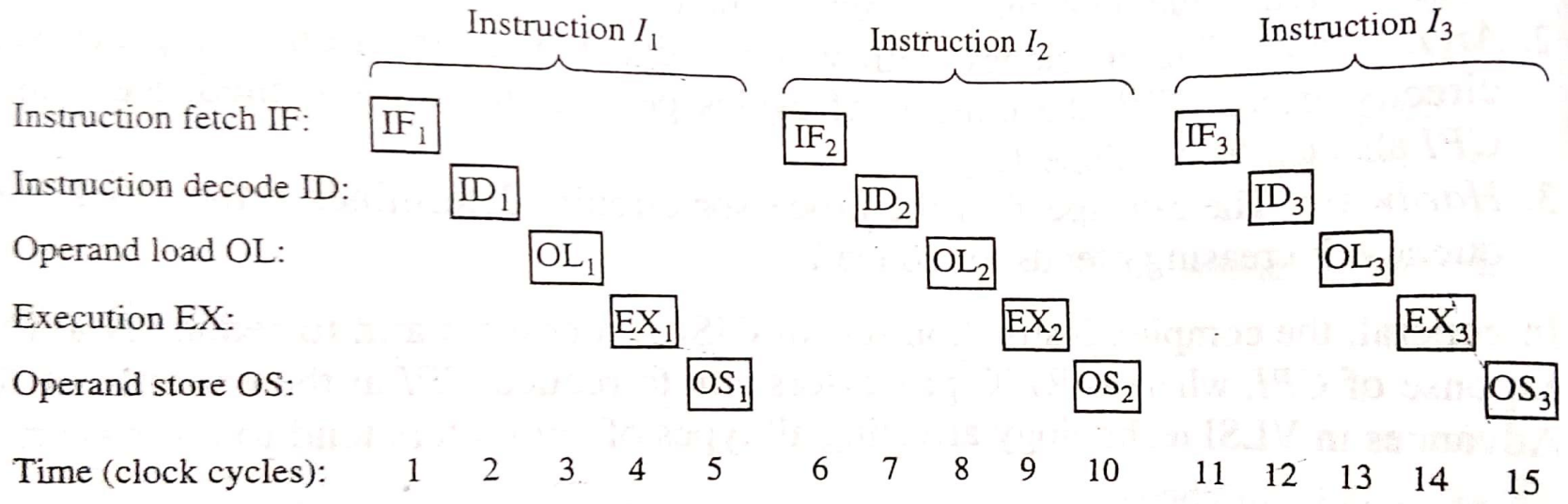
1. Cache memory : To provide the CPU with faster access to instructions and data.
2. Pipelined processing : To increase performance by allowing the processing of several instructions to be partially overlapped.
3. Superscalar processing : To increase performance by allowing several instructions to be processed in parallel (full overlapping)

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA : SPEEDUP TECHNIQUES

Pipelined processing:



COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA : SPEEDUP TECHNIQUES

Pipelined processing:

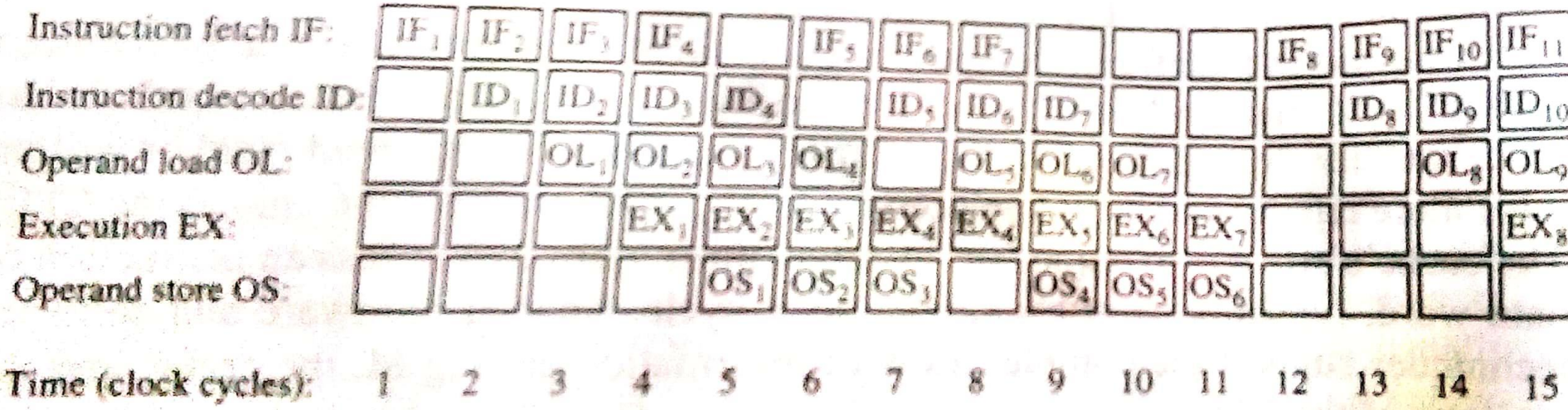


Figure 1.24
Instruction processing: (*a*) sequential or nonpipelined and (*b*) pipelined.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA : SPEEDUP TECHNIQUES

Pipelined processing:

Note :

1. The performance reducing delays occur, as in the case of instruction I_4 (shaded), which must use the EX stage for two consecutive cycles.

2. Branch instruction I_7 also reduces the performance, outcome of I_7's EX step must be known before the location of the next instruction I_8 to be processed can be identified.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA : SPEEDUP TECHNIQUES

Superscalar processing:

- A microprocessor's effective MIPS rate can also be increased by replicating various instruction-processing circuits so that several instructions can be in the same processing phase at the same time.
- This makes it possible to start the processing of, or issue, two or more instructions simultaneously or in parallel; in other words, the instructions can be completely overlapped.
- The two instructions should be independent of each other (they should not share the same resource or one's input depends on another output).
- Pipelining and superscalar design are both instances of instruction-level parallelism.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA : SYSTEM ARCHITECTURE

Basic organization:

- Standalone computer or PC or workstation – intended for a single user.
- Operating system – performs system management
- Microprocessor – responsible for fetching, decoding and executing instructions
- Data and instructions are typically composed of 32-bit words.
- Instruction set containing up to 200 or so instruction types, which perform data transfer, data processing and program control operations.
- The CPU may be augmented by on-chip or off-chip **coprocessors** that implement such specialized functions as managing the graphical user interface (GUI).

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA : SYSTEM ARCHITECTURE

Basic organization:

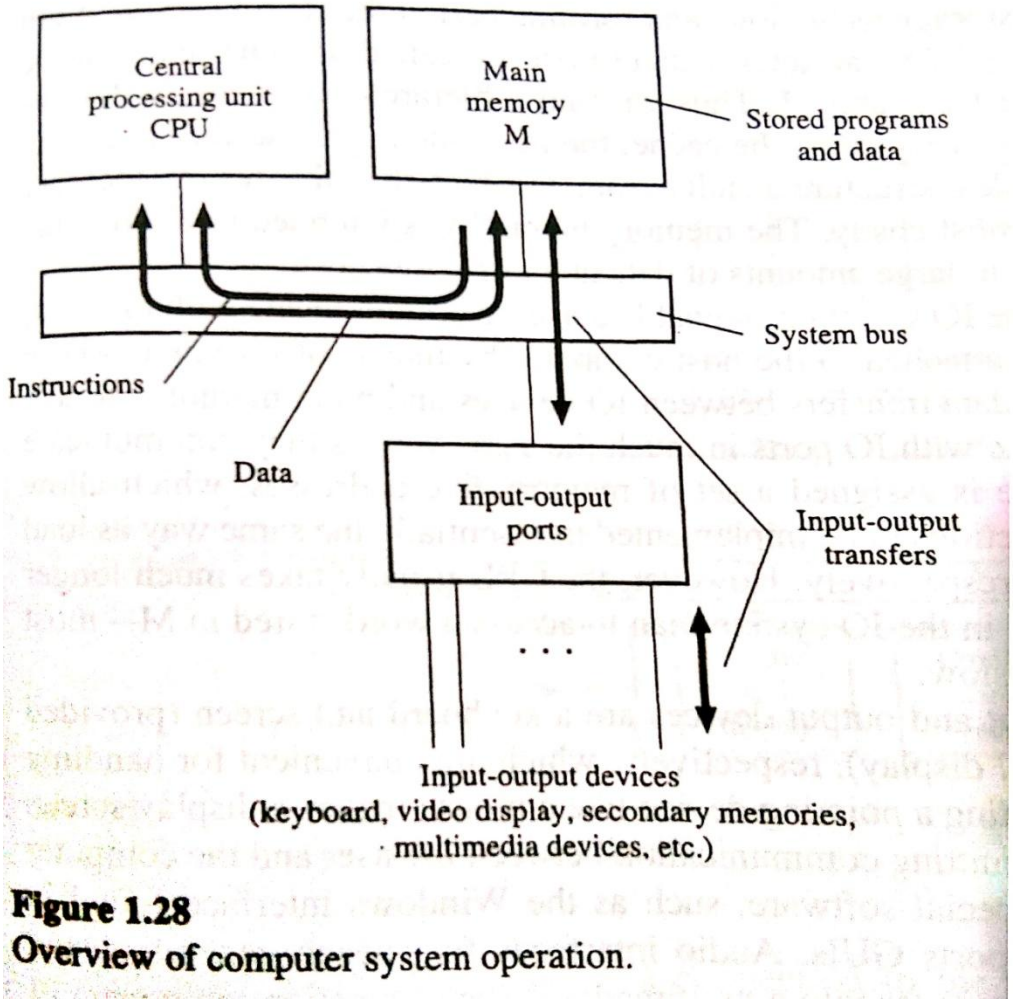


Figure 1.28
Overview of computer system operation.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA : SYSTEM ARCHITECTURE

Basic organization:

- The hierarchy of memory devices composed of the CPU's registers, the cache, the main memory and the secondary memory.
- The fastest memory has higher cost.
- The memory hierarchy is intended to provide the CPU with fast access to large amounts of data at a fairly low cost.
- IO devices are attached to the host computer by means of **IO ports**.
- An IO device is assigned a set of memory-like addresses, which allow input and output instructions to be implemented in essentially the same way as load and store instructions, respectively.
- IO operations are quite slow compared to that of memory operation.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA : SYSTEM ARCHITECTURE

Basic organization: Microcontrollers

- Small size and low cost have made it feasible to use miniature general-purpose computers, referred to as microcontrollers.
- Eg: Applications – Washing machine, control the ignition system of the car.
- Programs stored in read-only memory (ROM) that forms a part of main memory.
- It is used in application specific control circuits.
- Application : a point of sale terminal (POS) has replaced cash registers in retail stores.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA : SYSTEM ARCHITECTURE

Basic organization: Microcontrollers

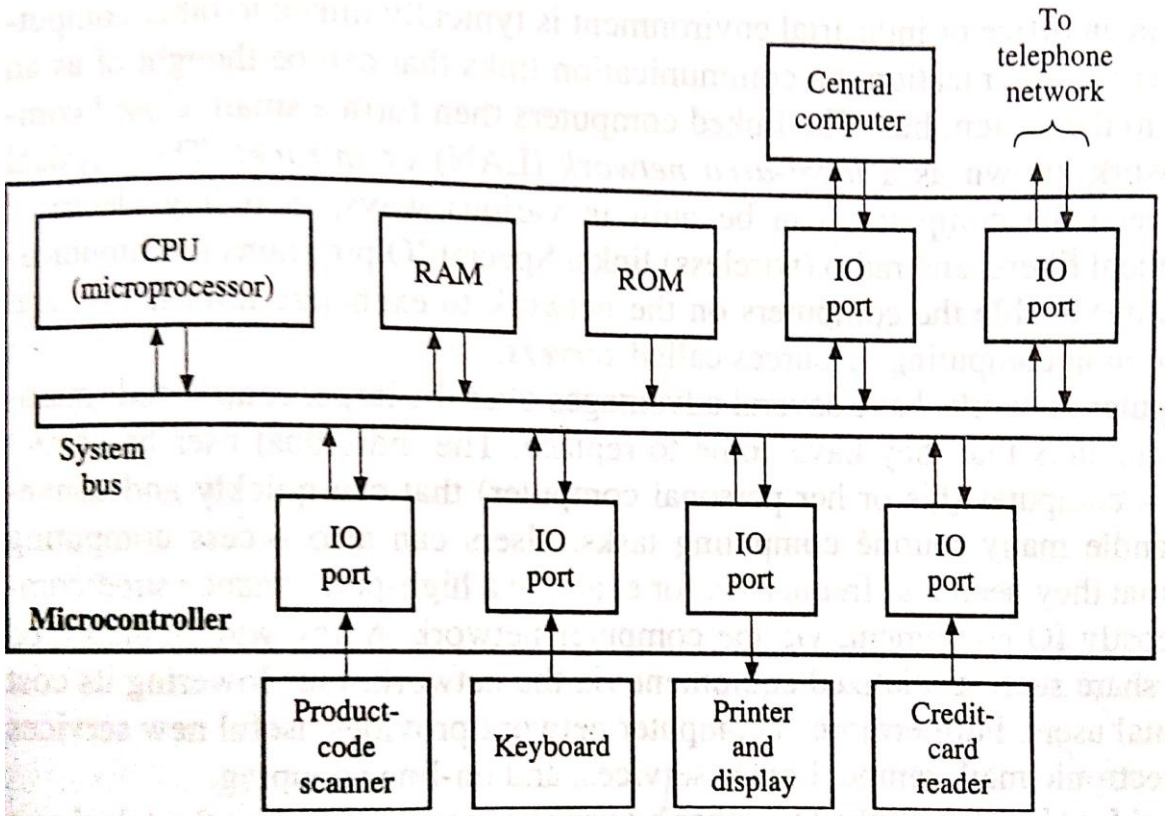


Figure 1.29
A microcontroller-based point-of-sale terminal.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA : SYSTEM ARCHITECTURE

Basic organization: Computer networks

➢ The linking of computers to form networks of various types has become an increasingly important feature of modern computing.

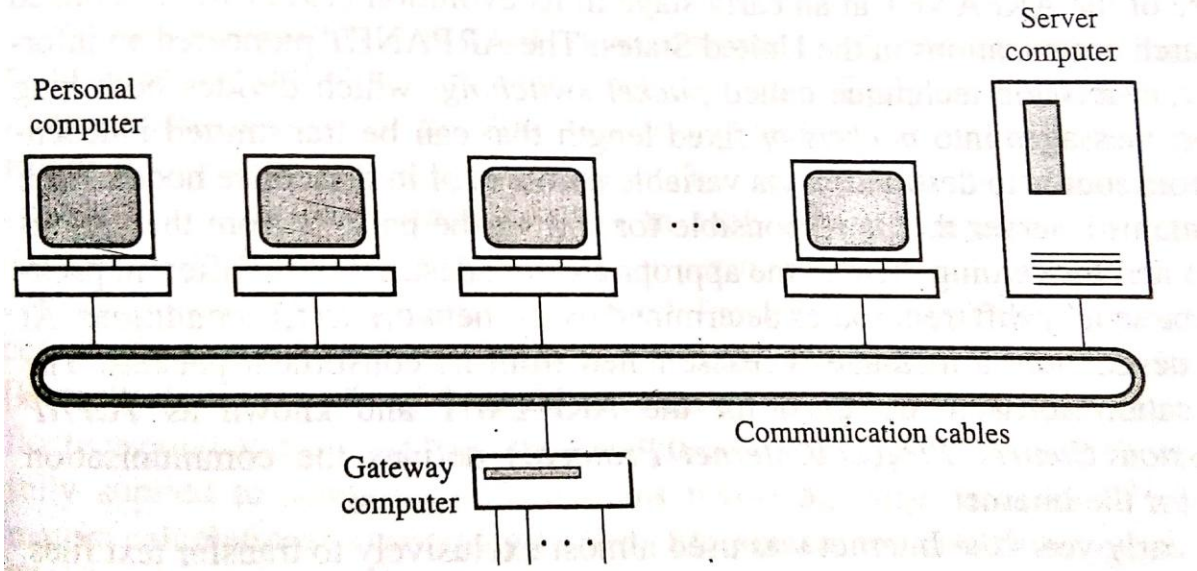


Figure 1.30
A local-area computer network.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA : SYSTEM ARCHITECTURE

Basic organization: Computer networks

- The linked computers in an organization or industrial environment via communication links form a small, closed computer network known as a **local area network (LAN)** or **intranet**.
- The physical links between the computers can be built in various ways, including electrical cables, optical fibers and radio (wireless) links.
- Special IO programs (communication software) enable the computers on the network to exchange information and access common computing resources called **servers**.
- Computer network provides useful new services such as electronic mail, remote library services and online shopping etc.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA : SYSTEM ARCHITECTURE

Basic organization: Computer networks

- Several LANs can be linked together using **gateway device** (manages communication between the LAN and other computer networks).
- A collection of linked LANs form a large computer network known as **Internet.**
- Internet – utilizes **packet switching technique** for information transmission.
- Both long and short messages are divided into packets of fixed length that can be transmitted independently from source to destination via variable numbers of intermediate nodes.
- Each node contains a server that is responsible for sorting the packet from various messages and forwarding them to the appropriate next destinations.
- At the final destination, a message is reassembled from its constituent packets.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA : SYSTEM ARCHITECTURE

Basic organization: Computer networks

- **Transmission control protocol/Internet protocol (TCP/IP)** – defines the communication standard for the Internet.
- Initially Internet was used to send electronic mails.
- A high-level protocol (**http - hypertext transport protocol**) and an associated programming language (**html – hypertext markup language**) permit the linking of diverse file types – texts, still pictures, movies, sound etc.
- The result is an enormously rich collection of easily accessible data that has come to be known as the **World wide web (www).**

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA : SYSTEM ARCHITECTURE

Basic organization: Parallel processing

- Supercomputers capable of executing many instructions in parallel.
- Single computer with pipelining strategy (one or more multistage pipelines)
- Degree of parallelism n possible with a pipeline is small, typically less than 10.
- Alternate approach – Use many independent processors operating in unison.
- Loosely coupled or **distributed system** is useful for computing tasks that can be easily be partitioned into independent subtasks, with infrequent communication of results among the subtasks.
- Exchange of results between the subtasks needs to be faster – slow IO transfers.

COMPUTING AND COMPUTERS

THE EVOLUTION OF COMPUTERS : ELECTRONIC COMPUTERS

THE VLSI ERA : SYSTEM ARCHITECTURE

Basic organization: Parallel processing

- To address the interprocess communication problem, computers have been built that employ n separate CPUs that are tightly coupled, both physically and logically. Processors can access one another's data rapidly are called **multiprocessors**.
- Two types of multiprocessors are :

1. Shared-memory machines : All the processors have access to a common main memory through which they communicate to share programs and data.
2. Distributed-memory machines: Each processor has only a private or local main memory and communicates with other processors by sending them messages through an IO subsystem linking the processors.

- Key issue – high speed and low cost processor to memory and processor to processor interconnects are required.

DESIGN METHODOLOGY : SYSTEM DESIGN

- The design process for digital systems at three basic levels of abstraction: the **gate**, the **register** and **the processor levels**.
- A computer is an example of a system – collection of a large and complex of objects called components, that are connected to form a coherent entity with a specific function or purpose.
- System representation – **graph**.
- A (directed) graph consists of a set of objects $V = \{v_1, v_2, v_3, \ldots, v_n\}$ called **nodes** or **vertices** and a set of edges E whose members are (ordered) pairs of nodes taken from the set $\{(v_1, v_2), (v_1, v_3), \ldots, (v_{n-1}, v_n)\}$ of all such pairs.
- The edge $e = (v_i, v_j)$ joins or connects node v_i to node v_j.

DESIGN METHODOLOGY : SYSTEM DESIGN

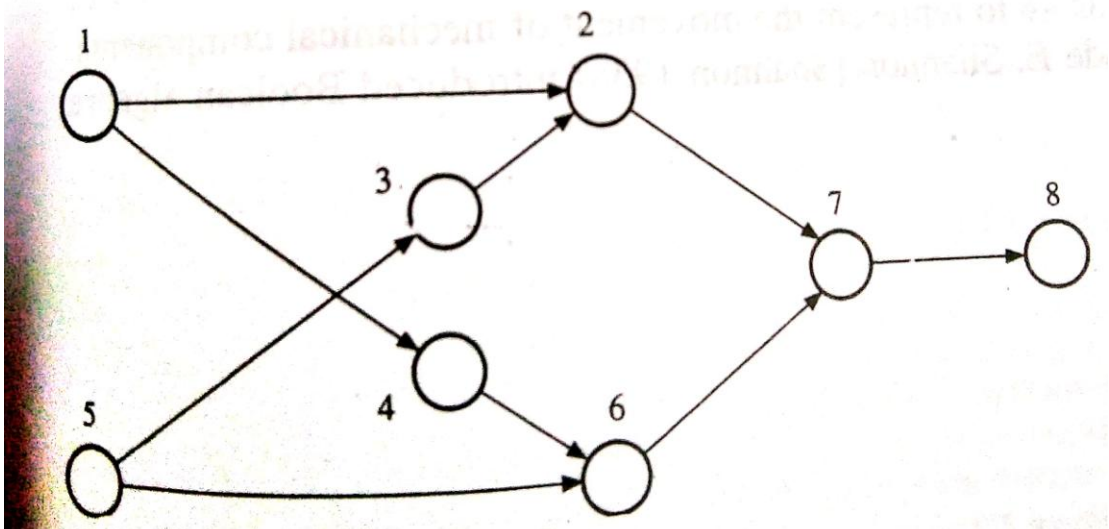


Figure 2.1
A graph with eight nodes and nine edges.

- The system of interest comprises two classes of objects : a set of information processing components C and a set of lines S that carry information signals between components.
- C is similar to nodes and S is similar to edge in a graph G.
- The resulting graph is called as a **block diagram**.

DESIGN METHODOLOGY : SYSTEM DESIGN

- Structure vs behavior : Two central properties of any system.
- Structure of a system as the abstract graph consisting of its block diagram with no functional information.
- A **structural description** merely names components and defines their interconnections.
- A **behavioral description** enables one to determine for any given input signal a to the system, the corresponding output f(a). The function f to be the behavior of the system.
- The tabulation of all possible combinations of input-output values is called the **truth table**.

DESIGN METHODOLOGY : SYSTEM DESIGN

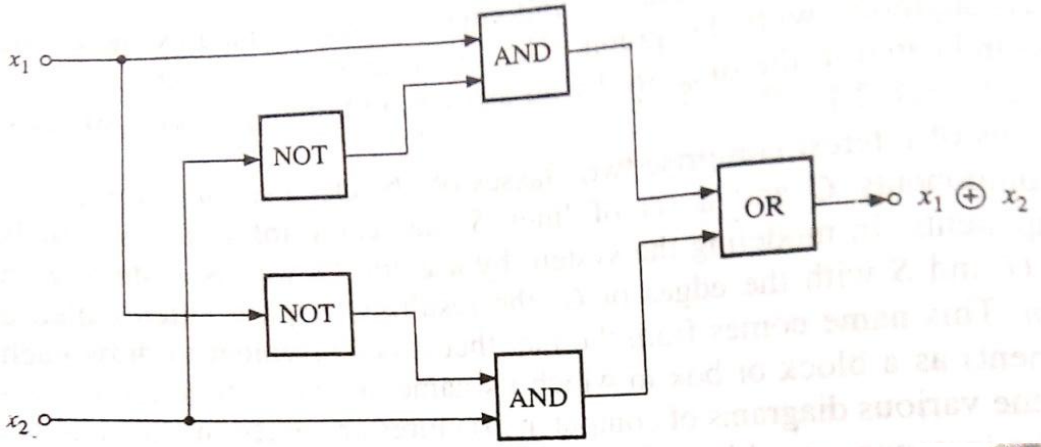


Figure 2.2
A block diagram representing an EXCLUSIVE-OR logic circuit.

Input a		Output
x_1	x_2	$f(a)$
0	0	0
0	1	1
1	0	1
1	1	0

DESIGN METHODOLOGY : SYSTEM DESIGN

- Alternate : using mathematical equations,

  $f(a) = f(x_1, x_2)$

  f(0, 0) = 0

  f(0, 1) = 1

  f(1, 0) = 1

  f(1, 1) = 0.

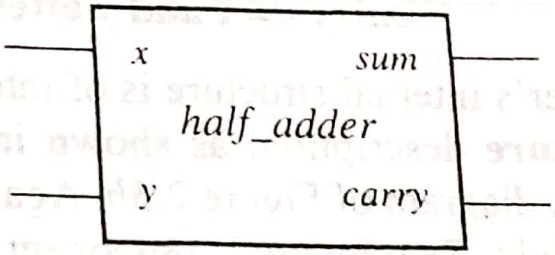
- Hardware description language : used to represent the functions of each components.
- **VHDL (Very High Speed Integrated Circuits Hardware Description Language)** and **VerilogHDL**
- VHDL has several advantages : precise, technology-independent description of digital circuits at various levels of abstraction, primarily the gate and register levels.

DESIGN METHODOLOGY : SYSTEM DESIGN

```
entity half_adder is
  port (x,y: in bit; sum, carry: out bit);
end half_adder;

architecture behavior of half_adder is
begin
  sum <= x xor y;
  carry <= x and y;
end behavior;
```

(a)



(b)

Inputs		Outputs	
x	*y*	*sum*	*carry*
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(c)

Figure 2.4
Half adder: (*a*) behavioral VHDL description; (*b*) block symbol; and (*c*) truth table.

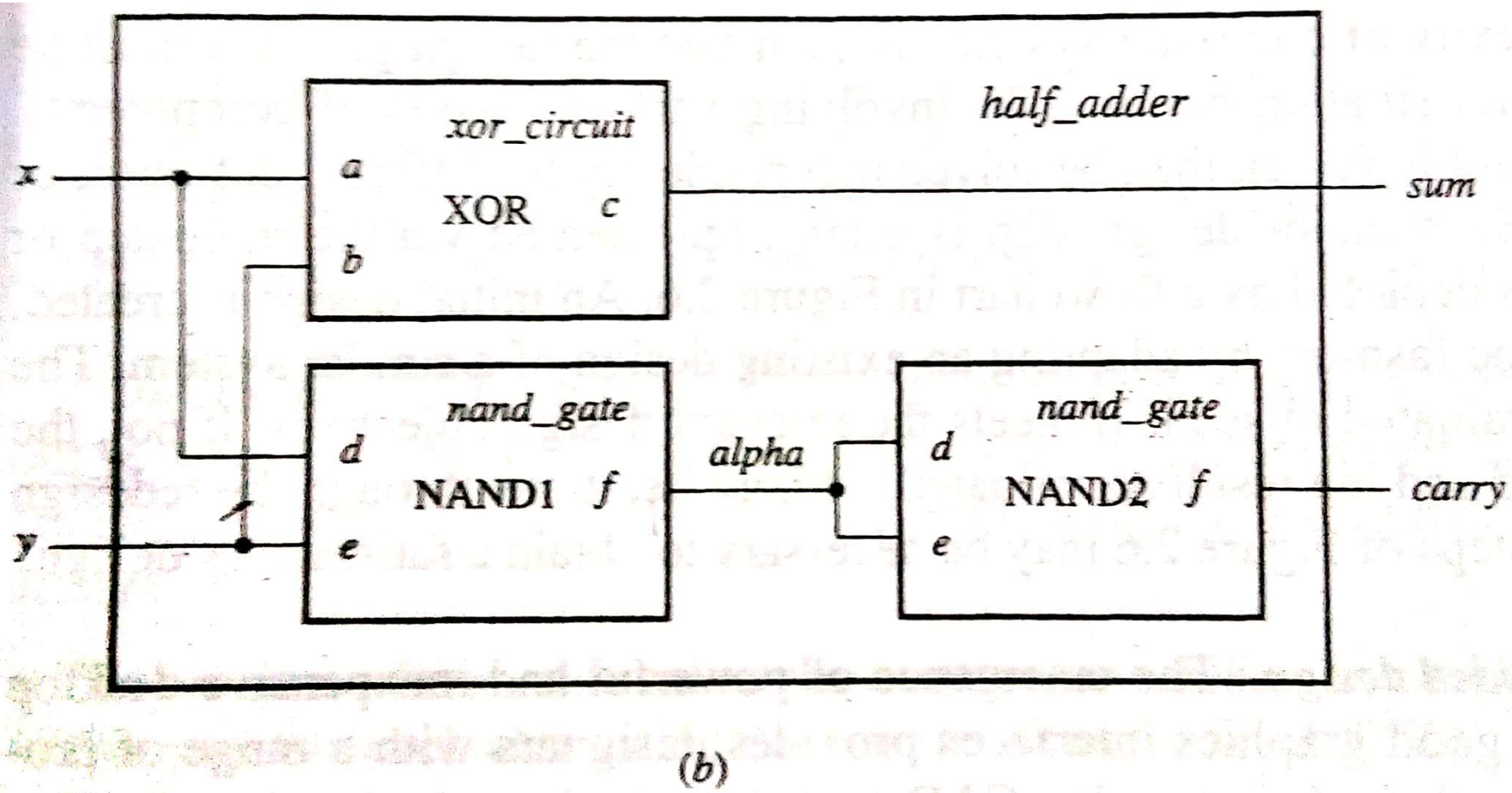
DESIGN METHODOLOGY : SYSTEM DESIGN

```
entity half_adder is
  port (x,y: in bit; sum, carry: out bit);
end half_adder;

architecture structure of half_adder is
  component xor_circuit port (a,b: in bit; c: out bit); end component;
  component nand_gate port (d,e: in bit; f: out bit); end component;
  signal alpha: bit;
begin
  XOR: xor_circuit port map (a => x, b => y, c => sum);
  NAND1: nand_gate port map (d => x, e => y, f => alpha);
  NAND2: nand_gate port map (d => alpha, e => alpha, f => carry);
end structure;
```

(a)

DESIGN METHODOLOGY : SYSTEM DESIGN



(b)

Figure 2.5
Half adder: (*a*) structural VHDL description; (*b*) block diagram.

DESIGN METHODOLOGY : SYSTEM DESIGN

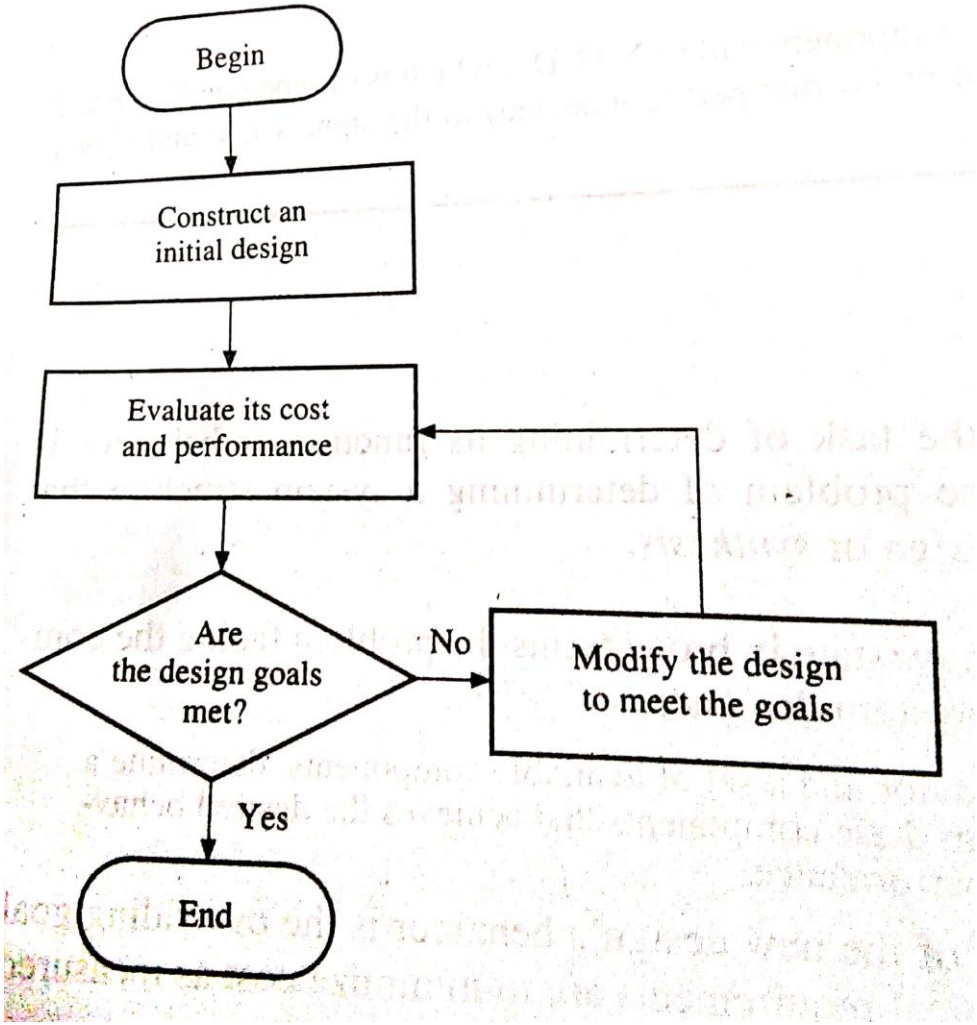
Design Process:

- Given a system structure, the task of determining its functions or behavior is termed **analysis**.
- The converse problem of determining a system structure that exhibits a given behavior is **design** or **synthesis**.
- Given a desired range of behavior and a set of available components, determine a structure (design) formed from these components that achieves the desired behavior with acceptable cost and performance.
- Major requirement of the design : assuring the correctness of the new design's behavior.
- Other typical requirements are : maximize performance, high reliability, low power consumption and compatibility with existing system.

DESIGN METHODOLOGY : SYSTEM DESIGN

Design Process:

Flowchart of an iterative design process



DESIGN METHODOLOGY : SYSTEM DESIGN

Design Process: Computer aided design

➢ **CAD tools** are used to automate, at least in part, the more tedious design and evaluation steps and contribute in three important ways to the overall design process.

1. **CAD editors** or **translators** – convert design data into forms such as HDL descriptions or schematic diagrams.
2. **Simulators** – create computer models of a new design, which can mimic the design's behavior and help designers determine how well the design meets various performance and cost goals.
3. **Synthesizers** – automate the design process itself by deriving structures that implement all or part of some design step.

DESIGN METHODOLOGY : SYSTEM DESIGN

Design Process: Design Levels

- The design of a complex system is carried out at several levels of abstraction.

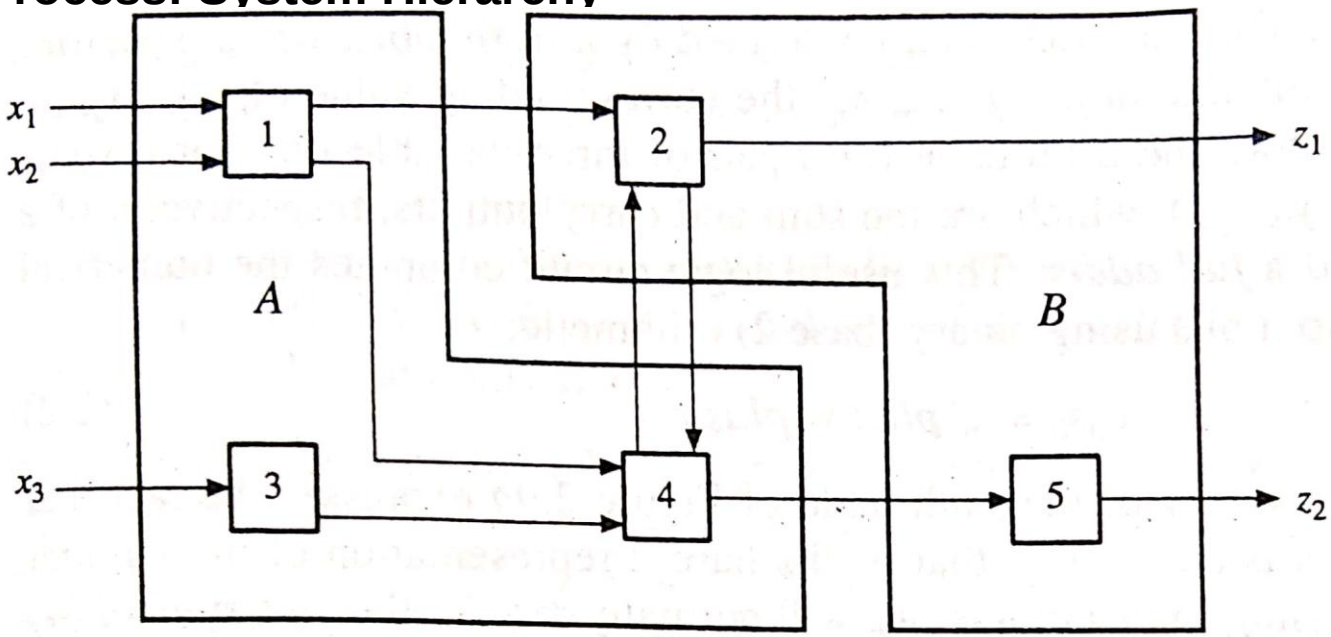
1. The **processor level** also called the **architecture, behaviour** or **system level**
2. The **register level**, also called the **register-transfer level (RTL)**
3. The **gate level**, also called the **logic level**.

Level	Components	IC density	Information units	Time units
Gate	Logic gates, flip-flops.	SSI	Bits	10^{-12} to 10^{-9} s
Register	Registers, counters, combinational circuits, small sequential circuits.	MSI	Words	10^{-9} to 10^{-6} s
Processor	CPUs, memories, IO devices.	VLSI	Blocks of words	10^{-3} to 10^{3} s

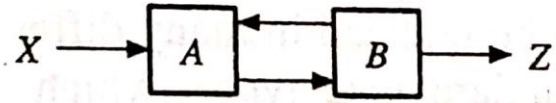
Figure 2.7
The major computer design levels.

DESIGN METHODOLOGY : SYSTEM DESIGN

Design Process: System Hierarchy



(a)



(b)

Figure 2.8
Two descriptions of a hierarchical system: (*a*) low level; (*b*) high level.

DESIGN METHODOLOGY : SYSTEM DESIGN

Design Process: System Hierarchy

- Top down design involves 3 steps, namely,

1. Specify the processor level structure of the system.
2. Specify the register level structure of each component type identified in step 1.
3. Specify the gate-level structure of each component type identified in step 2.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Gate level:

- Gate level design is concerned with processing binary variables (0 or 1).
- The design components are logic gates (memory-less processing elements) and flip-flops (bit storage devices).
- Combinational logic – referred to as a logic or a Boolean function, is a mapping from the set of 2^n input combinations of n binary variables onto the output values 0 and 1.

$$z(x_1, x_2, .., x_n)$$

- The function z can be defined by a truth table.

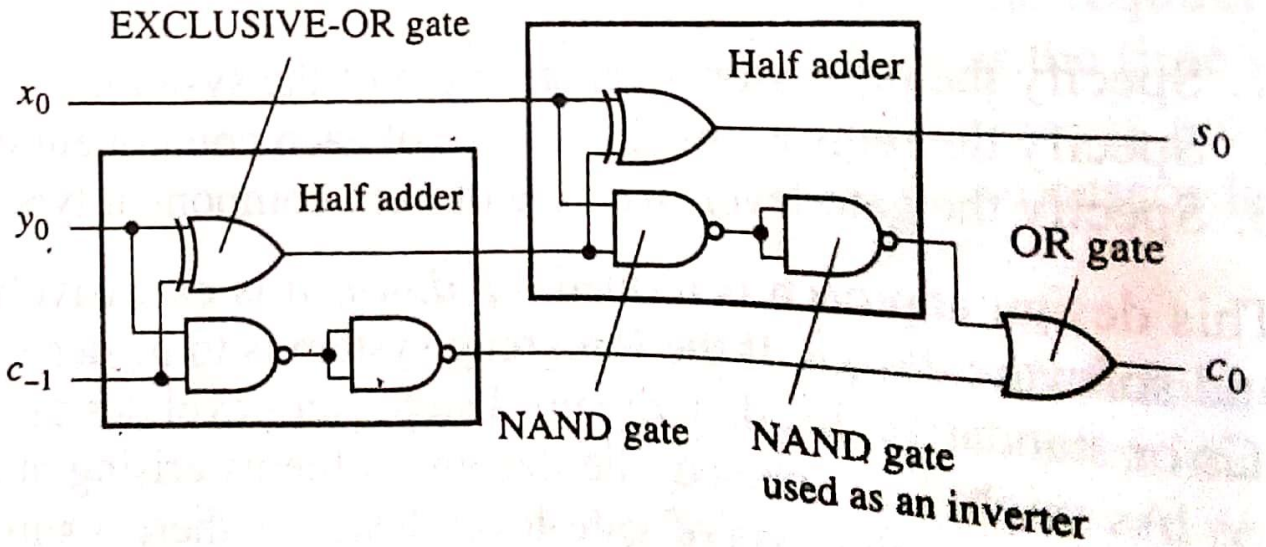
DESIGN METHODOLOGY : SYSTEM DESIGN

The Gate level:

Full Adder : A combinational function z can be realized in many different ways by combinational circuits built from standard gate types, which include AND, OR, EXCLUSIVE-OR, NOT (inverter), NAND and NOR.

Inputs			Outputs	
x_0	y_0	c_{-1}	c_0	s_0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

(a)



(b)

DESIGN METHODOLOGY : SYSTEM DESIGN

The Gate level:

Full Adder

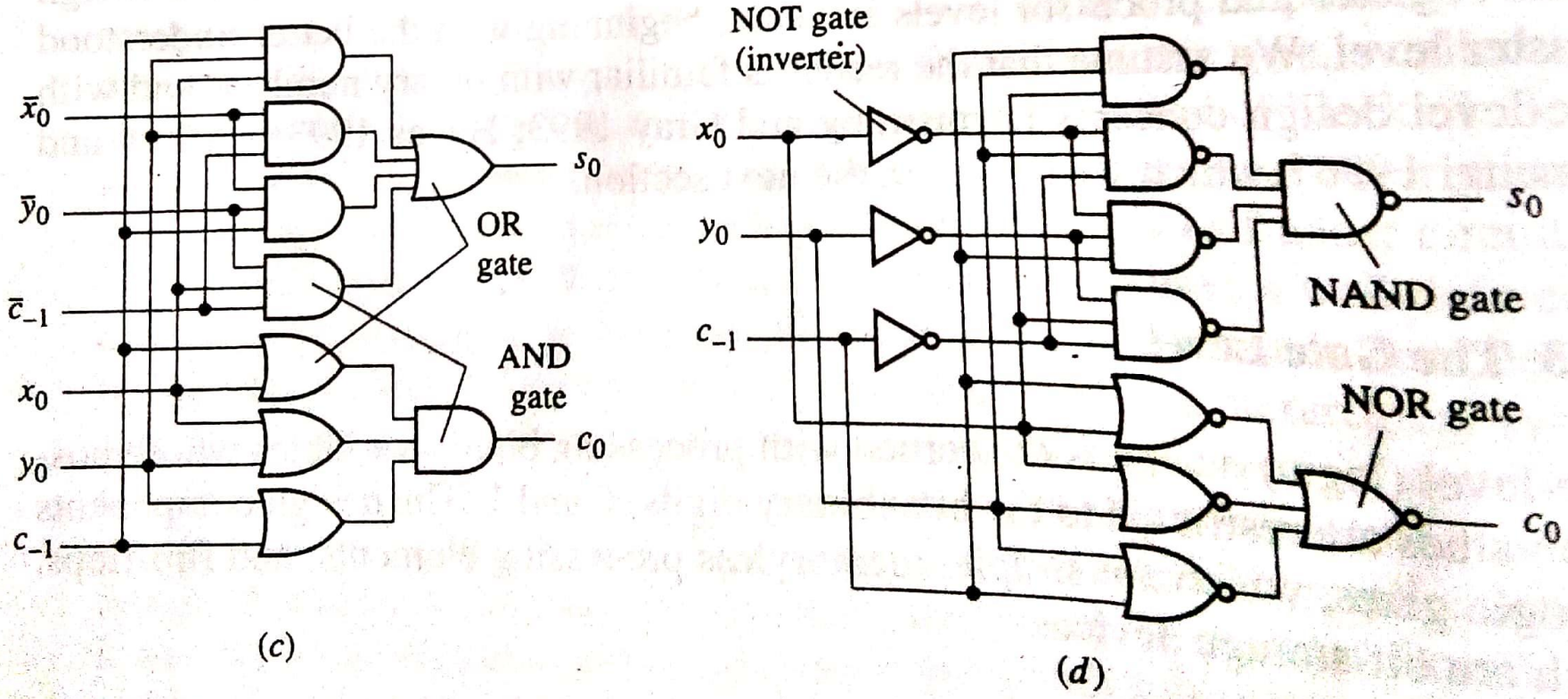


Figure 2.9
Full adder: (*a*) truth table; (*b*) realization using half adders; (*c*) realization using AND and OR gates; (*d*) realization using NAND, NOR, and NOT gates.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Gate level:

- NANDs and NORs are particularly important in logic design because they are easily manufactured using most IC technology and are the only standard gate types that are functionally complete by themselves.
- **Sum-of-products (SOP)** and **product-of-sums (POS)**

$$s_0 = x_0 y_0 c_{-1} + \overline{x_0} y_0 \overline{c_{-1}} + \overline{x_0}\,\overline{y_0} c_{-1} + x_0 \overline{y_0}\,\overline{c_{-1}}$$

$$c_0 = (x_0 + c_{-1})(x_0 + y_0)(y_0 + c_{-1})$$

- Full adder realization using two half adder and OR gate is **four level** design
- Full adder realization using AND and OR gate is **two level** design.
- The number of logic levels is defined by the number of gates along the circuit's longest IO path.
- If all the gates have the same propagation delay, then two-level adder is twice as fast as the four level design.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Gate level:

- However the two-level adder has more gates and so has a higher hardware cost.
- **Fan-in** – upper bound on the number of inputs of a gate G.
- **Fan-out** – upper bound on the number of inputs of other gates to which G's output line may be connected.
- Once a good design of a useful function is known, it can be placed in a library for future use.
- A full adder, for instance, can be used to build a multibit, multilevel adder.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Gate level:

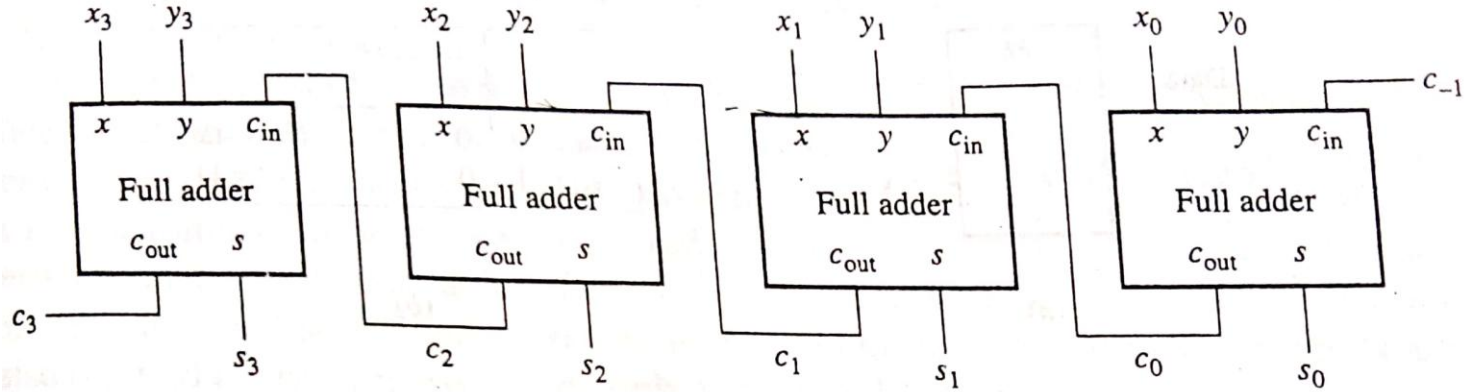


Figure 2.10
Four-bit ripple-carry: (*a*) logic structure; (*b*) high-level symbol.

DESIGN METHODOLOGY : SYSTEM DESIGN

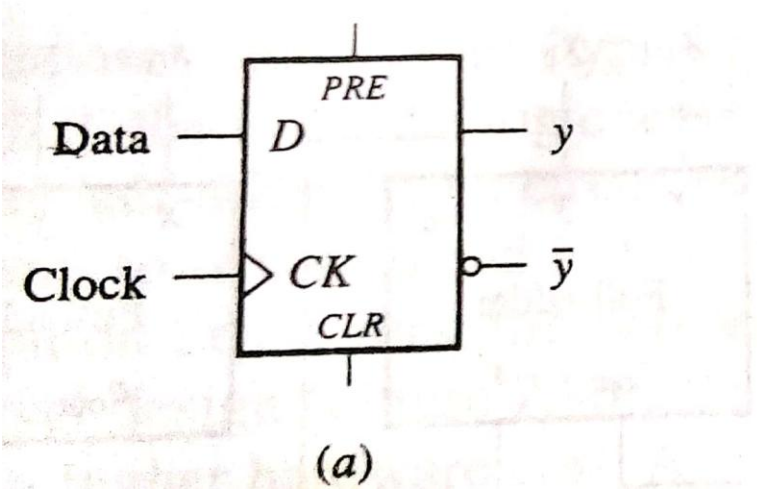
The Gate level: Flipflops

- By adding memory to a combinational circuit in the form of 1-bit storage elements called **flipflops**, we obtain a sequential logic circuits.
- Flipflops rely on an external clock signal CK to synchronize the times at which they respond to changes on their input data lines.
- Edge triggering – confines the flipflops state changes to a narrow window of time around one edge (0-1 or 1-0 transition point) of CK.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Gate level: Flipflops

- Edge-triggered D (delay) flipflop



(a)

	Input $D(i)$		
	0	1	
State $y(i)$ 0	0	1	Next state $y(i+1)$
State $y(i)$ 1	0	1	Next state $y(i+1)$

(b)

- The output signal y constitutes the stored data or **state** of the flipflop.
- The D flipflop reads in the data value on its D line when the 0-1 triggering edge of the clock signal CK arrives; this D value becomes the new value of y.
- The omission of the triangular symbol on the clock's input port indicates **level triggering**, in which case the flipflop responds to all changes in the signal value on D – **latch.**

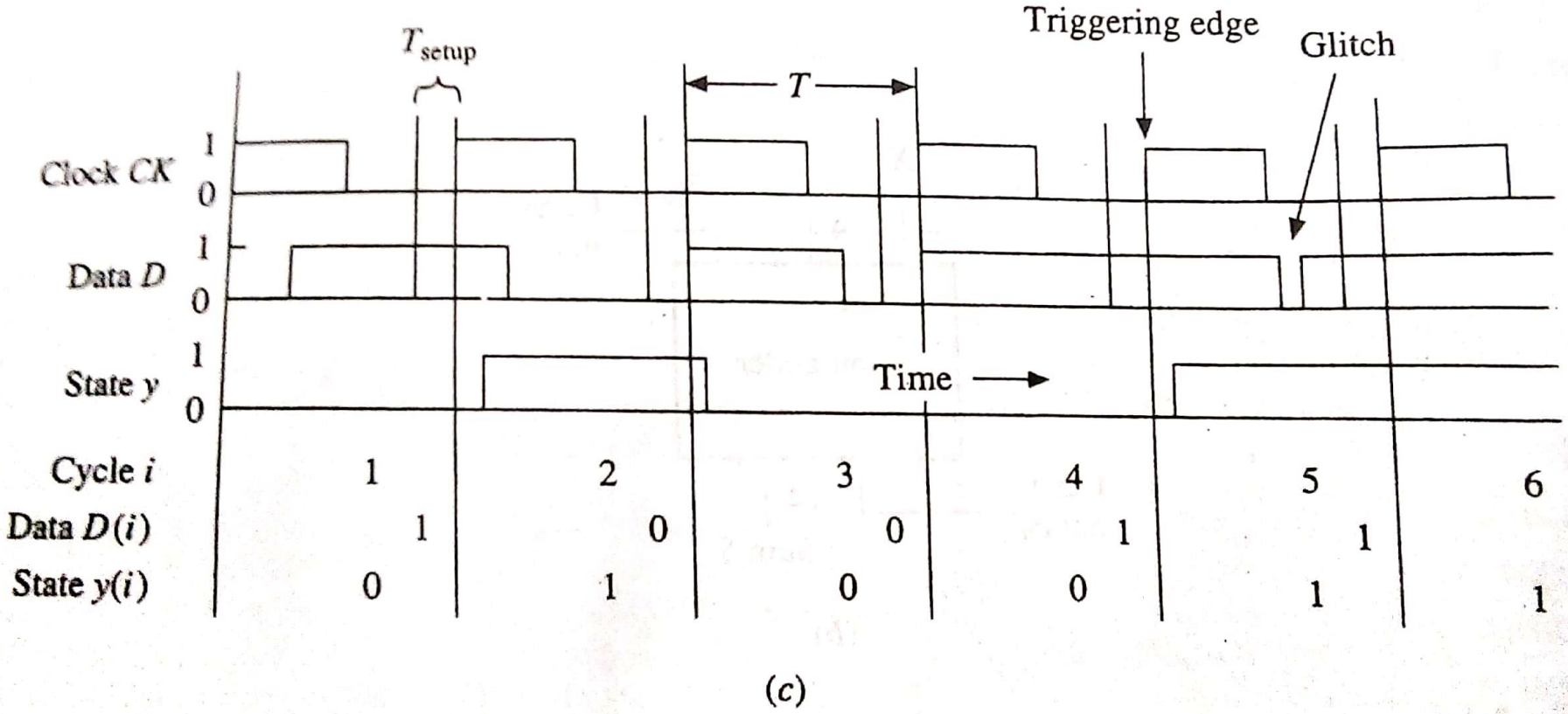
DESIGN METHODOLOGY : SYSTEM DESIGN

The Gate level: Flipflops

- In edge triggering, there is just one triggering edge in each clock cycle, there can be just one change in y per clock cycle.
- The input data line D can be varied independently and so can go through several changes in any clock cycle i.
- To change the flipflops state, the D signal must be held steady for a minimum period known as the **setup time (T_{setup})** before the flipflop is triggered.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Gate level: Flipflops



(c)

Figure 2.11
D flip-flop: (*a*) graphic symbol; (*b*) state table; (*c*) timing diagram.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Gate level: Flipflops

- **Glitch** – sudden change in input signal.
- Edge triggered flipflops have he very useful property of filtering out noise signals appearing at their inputs.
- When flipflop is switched on, its initial state is uncertain.
- Flipflop can be set or reset (initial state) asynchronously (independent of clock signal CK) at the start of operation.
- One or two asynchronous control inputs, **CLR (clear – y to 0)** and **PRE (preset – y to 1)** is used.
- Characteristic equation of D flipflop is y(i+1)=D(i).

DESIGN METHODOLOGY : SYSTEM DESIGN

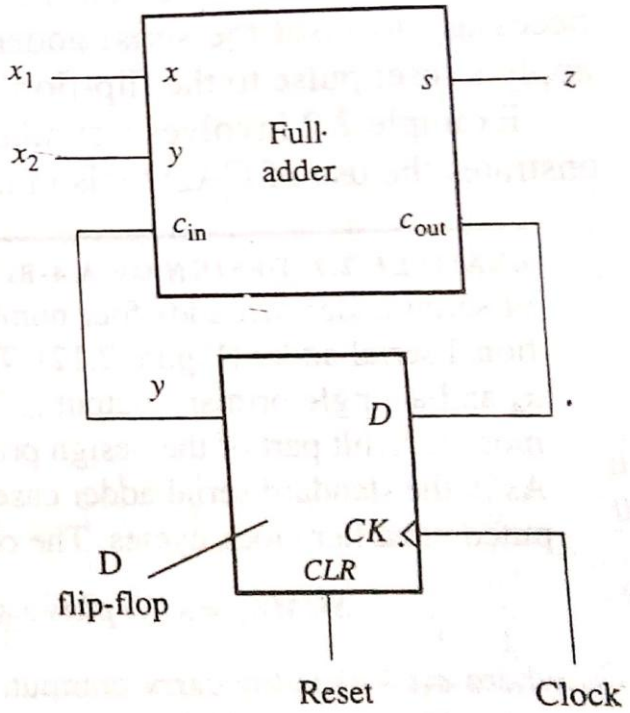
The Gate level: Sequential circuits

- A sequential circuit consists of a combinational circuit and a set of flipflops.
- Example : Serial adder

		Input $x_1 x_2$ 00	01	10	11
Present state	$S_0 (y = 0)$	S_0,0	S_0,1	S_0,1	S_1,0
	$S_1 (y = 1)$	S_0,1	S_1,0	S_1,0	S_1,1

Next state — Present output

(a)



(b)

Figure 2.12
(a) State table; (b) logic circuit for a serial adder.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Gate level: Sequential circuits

- The flipflops store information on the circuit's past behaviour; this stored information defines the circuit's **internal state Y**.
- If the primary inputs are X and the primary outputs are Z, then Z is a function of both X and Y, denoted Z(X,Y).
- Each tick (cycle or period) of the clock permits a single change in the circuit's state Y; it can also trigger changes in the primary output Z – Reflecting the state behaviour – **finite state machine (FSM).**
- Internal states S_0 – meaning that the previous carry signal c(i-1)=0 and S_1, meaning that c(i-1)=1.
- Before entering two new numbers to be added, it is necessary to reset the serial adder to the S_0 state – apply a reset pulse to a flipflop asynchronous clear (CLR) input.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level:

- At the register or register-transfer level, related information bits are grouped into ordered sets called words or vectors.
- The primitive components are small combinational or sequential circuits intended to process or store words.

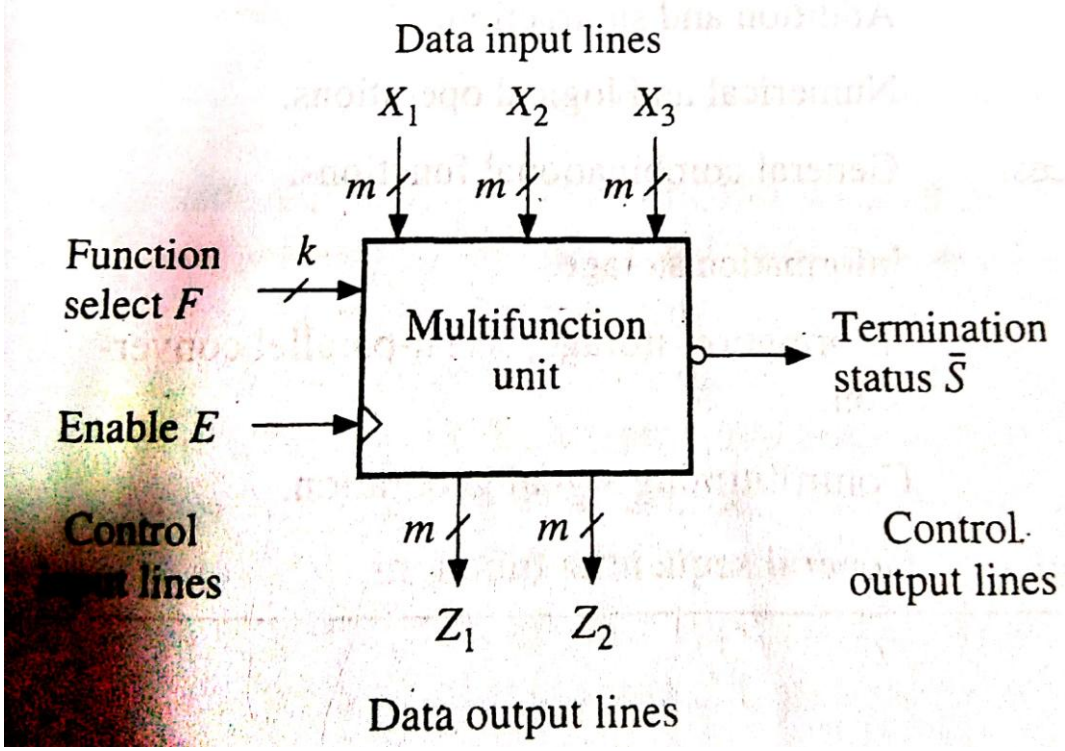
Type	Component	Functions
Combinational	Word gates.	Logical (Boolean) operations.
	Multiplexers.	Data routing; general combinational functions.
	Decoders and encoders.	Code checking and conversion.
	Adders.	Addition and subtraction.
	Arithmetic-logic units.	Numerical and logical operations.
	Programmable logic devices.	General combinational functions.
Sequential	(Parallel) registers.	Information storage.
	Shift registers.	Information storage; serial-parallel conversion.
	Counters.	Control/timing signal generation.
	Programmable logic devices.	General sequential functions.

Figure 2.15
The major component types at the register level.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level:

- Register-level components are linked to form circuits by means of word-carrying group of lines, referred to as **buses**.
- no universally accepted graphic symbol for register level components.
- represented in circuit diagrams by blocks containing an abbreviated description of their behavior.



DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level:

- A components IO lines are separated into **data** and **control lines**.
- A control lines name indicates the operation determined by the line in its active, enabled or asserted state.
- A small circle representing inversion.
- Enable line – specify the time or condition for a selected operation to be performed.
- Enable lines are often connected to the clock sources.
- In adequate design theory of word based logical operations are :

1. Operations performed are numerical rather than logical.
2. Logical operations are complex and do not have the properties of the gates – interchangeability of inputs.
3. Some buses carry signals with a different number of bits – makes it difficult to define a useful algebra to describe operations on these signals.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Word gates

- Let $X=(x_1, x_2, \ldots, x_m)$ and $Y=(y_1, y_2, \ldots, y_m)$ be two m-bit binary words.
- Let the output $Z=(z_1, z_2, \ldots, z_m) = f(X,Y)$
- Example : $Z = \overline{XY}$

$$Z = (z_1, z_2, \ldots, z_m) = (\overline{x_1 y_1}, \overline{x_2 y_2}, \ldots, \overline{x_m y_m})$$

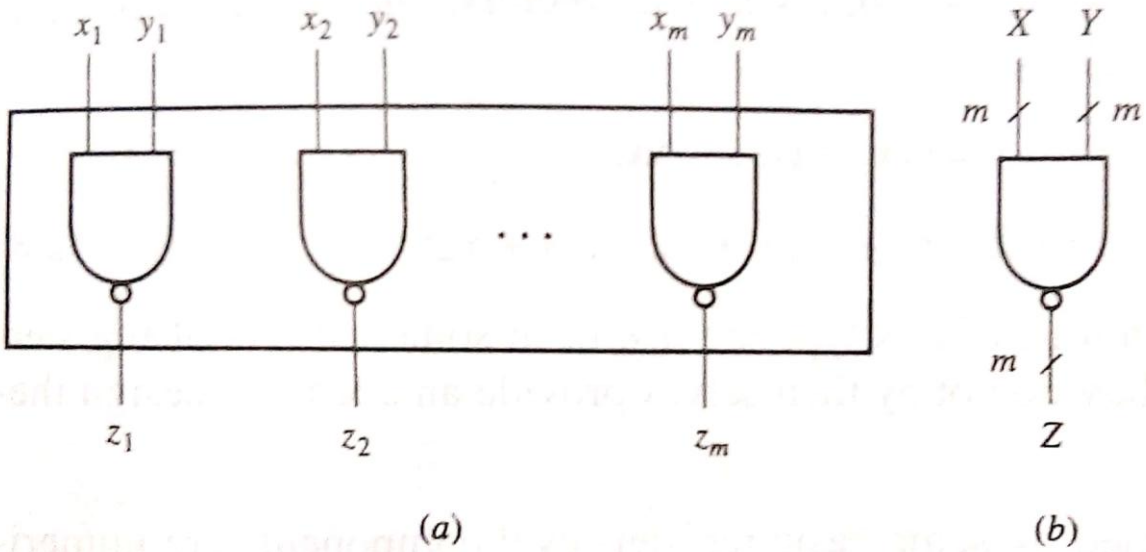


Figure 2.17
Two-input, *m*-bit NAND word gate: (*a*) logic diagram and (*b*) symbol.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Word gates

- Word gate circuits can be analyzed using Boolean Algebra.

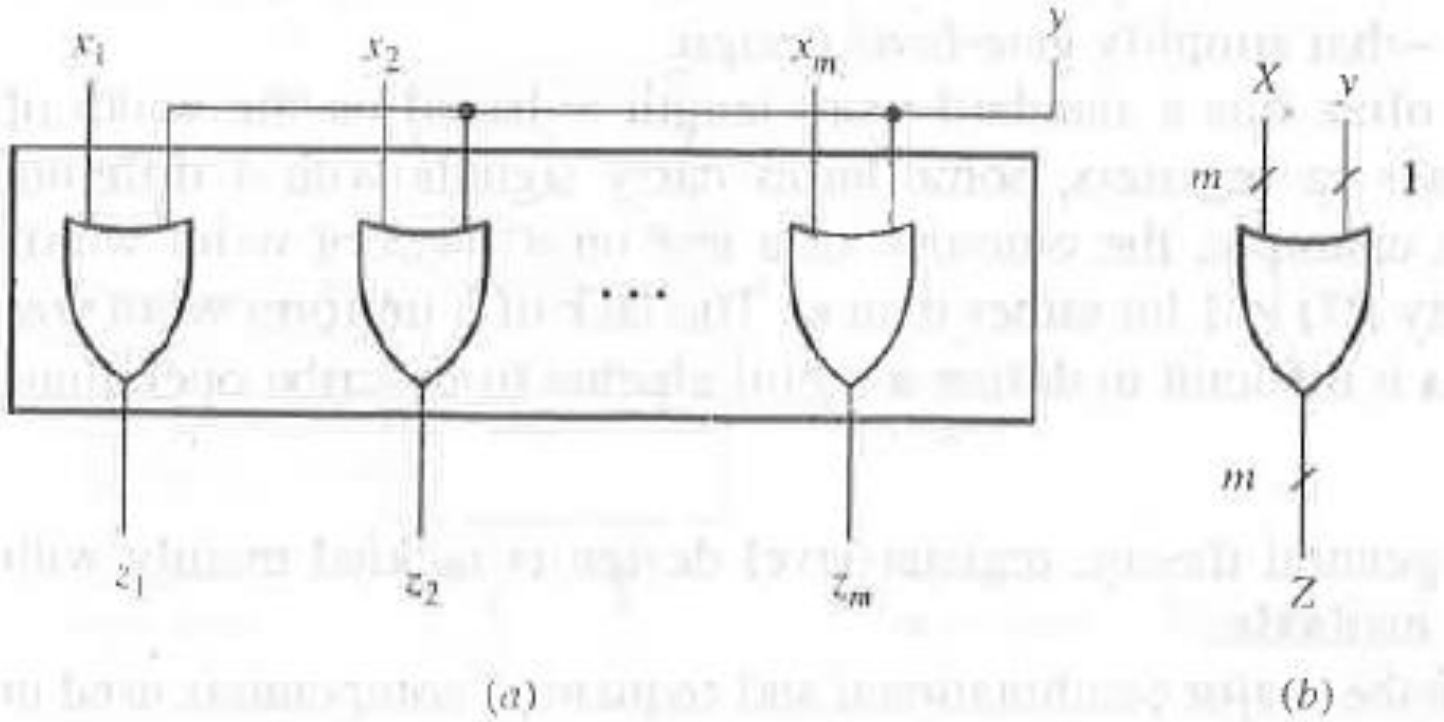


Figure 2.18
OR word gate implementing $y + X$: (*a*) logic diagram; (*b*) symbol.

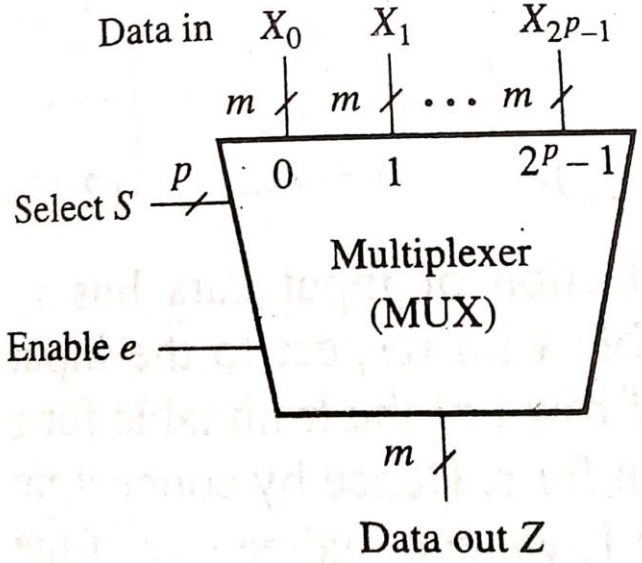
DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Multiplexers

➢ A **multiplexer** is a device intended to route data from one of several sources to a common destination; the source is specified by applying appropriate control (select) signals to the multiplexer.

➢ If the maximum number of data sources is k and each IO data line carries m-bit, the multiplexer is referred to as a **k-input (or k-way), m-bit multiplexer**.

➢K=2^p , data source selection is determined by an **encoded pattern** or **address** of p bits.



DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Multiplexers

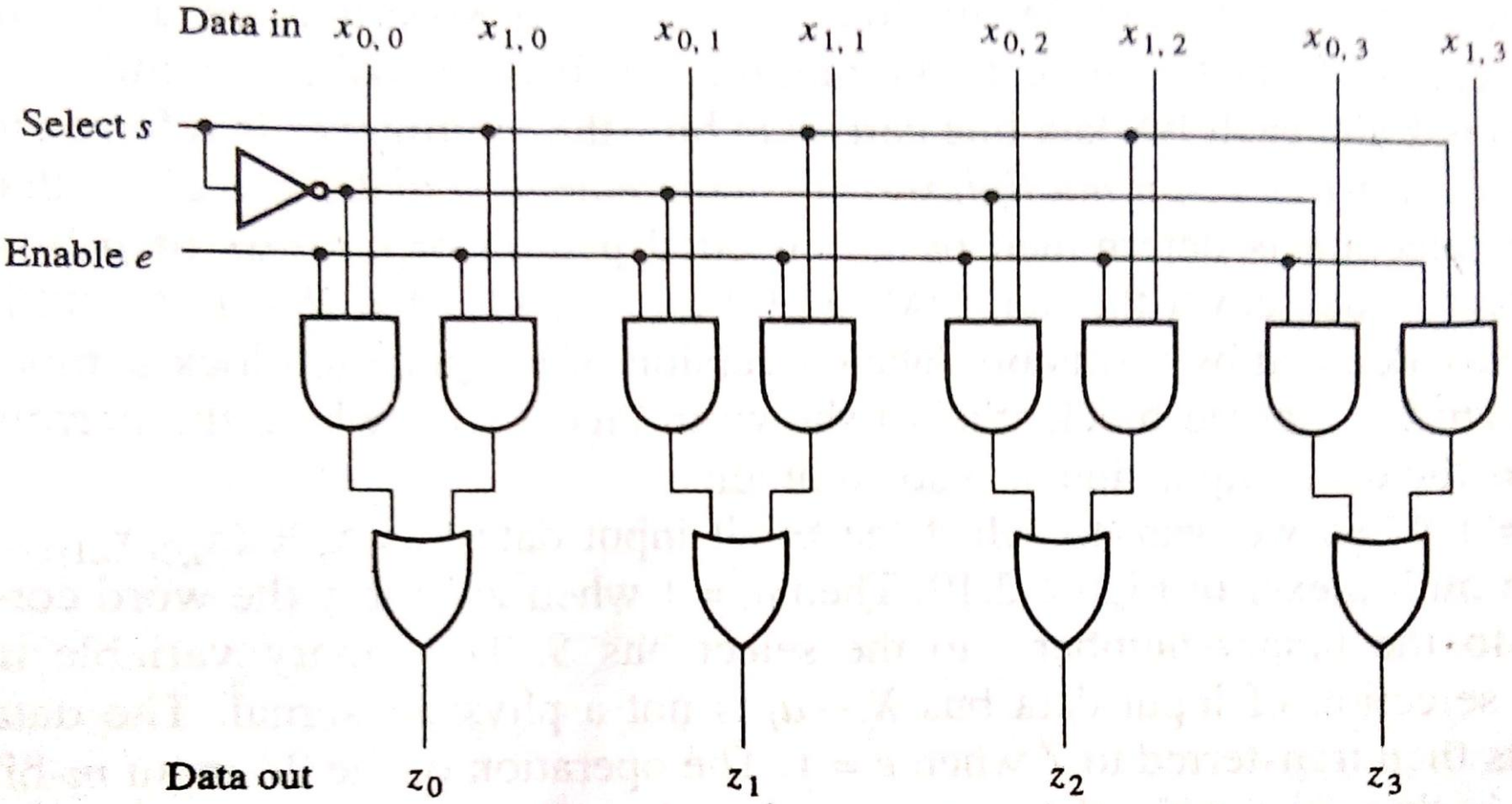


Figure 2.20
Realization of a two-input, 4-bit multiplexer.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Multiplexers

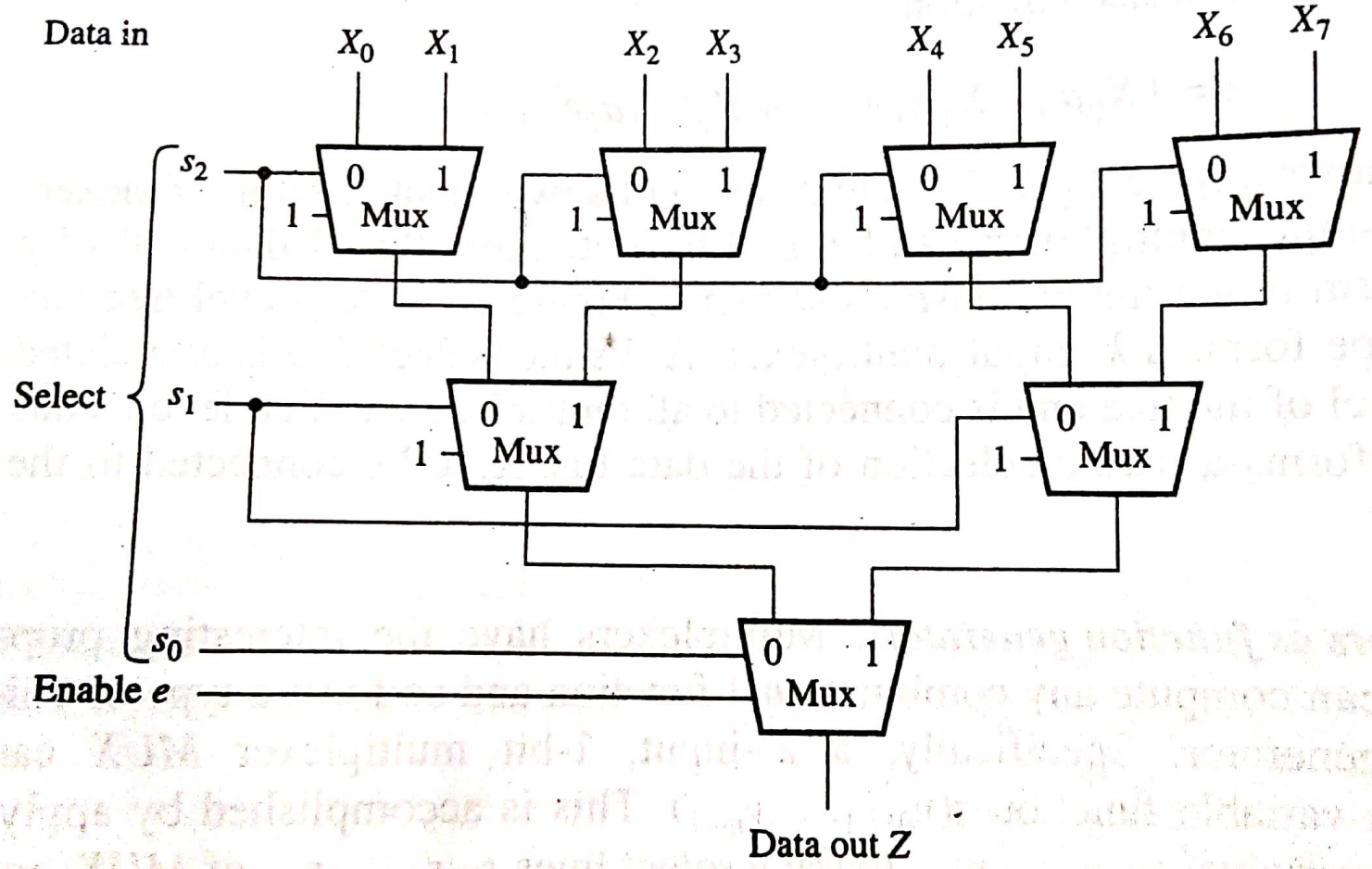


Figure 2.21
An eight-input multiplexer constructed from two-input multiplexers.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Multiplexers as function generators

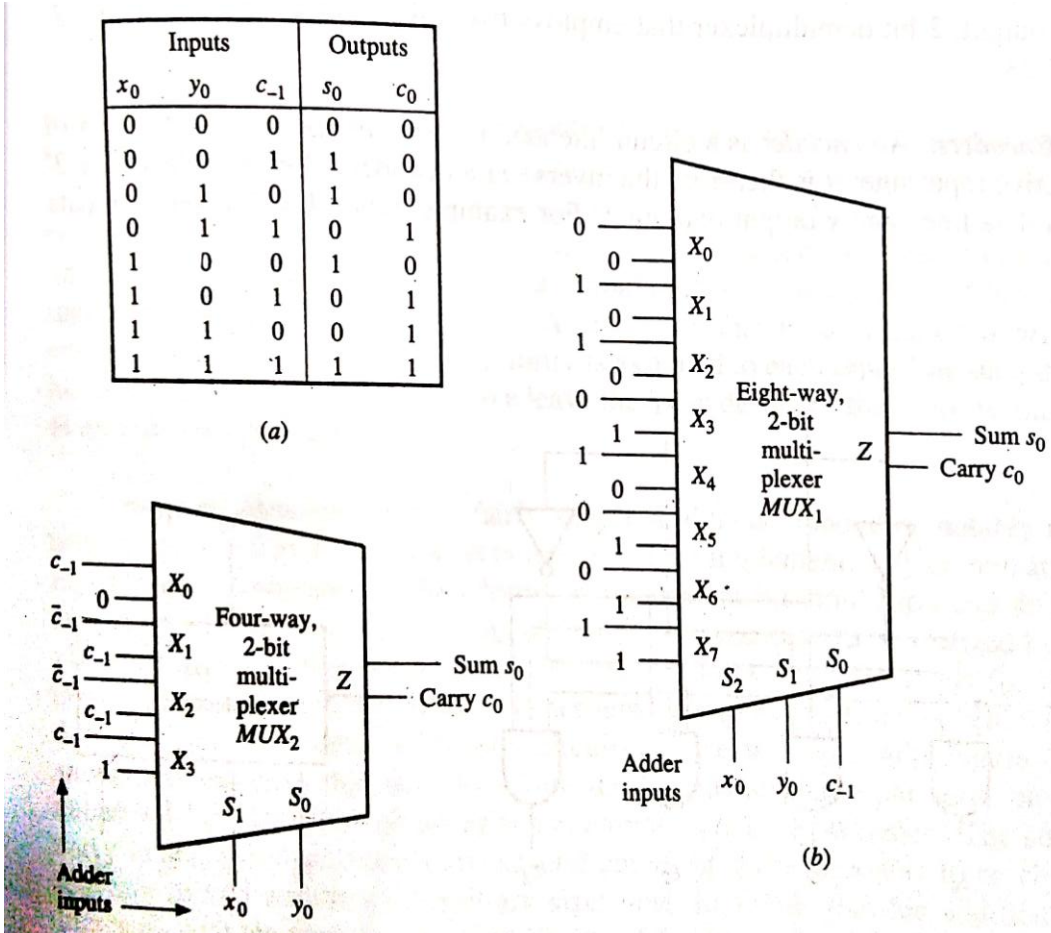
- Interesting property of multiplexer is they can be used to compute any combinational function – **universal logic generator**.
- Specifically, a 2^n – input, 1-bit multiplexer MUX can generate any n-variable function $z(v_0, v_1,\ldots,v_{n-1})$.
- This is accomplished by applying the n input variables $v_0,v_1,\ldots,v_{n-1}$ to the n select lines $s_0,s_1,\ldots,s_{n-1}$ of MUX, and 2^n function specific constant values (0 or 1) to MUX's 2^n input data lines $x_0,x_1\ldots,x_{2^n-1}$.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Multiplexers to implement a full adder

Inputs			Outputs	
x_0	y_0	c_{-1}	s_0	c_0
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a)



(b)

(c)

Figure 2.22
Multiplexer-based full adder: (*a*) truth table; (*b*) first version; (*c*) second version.

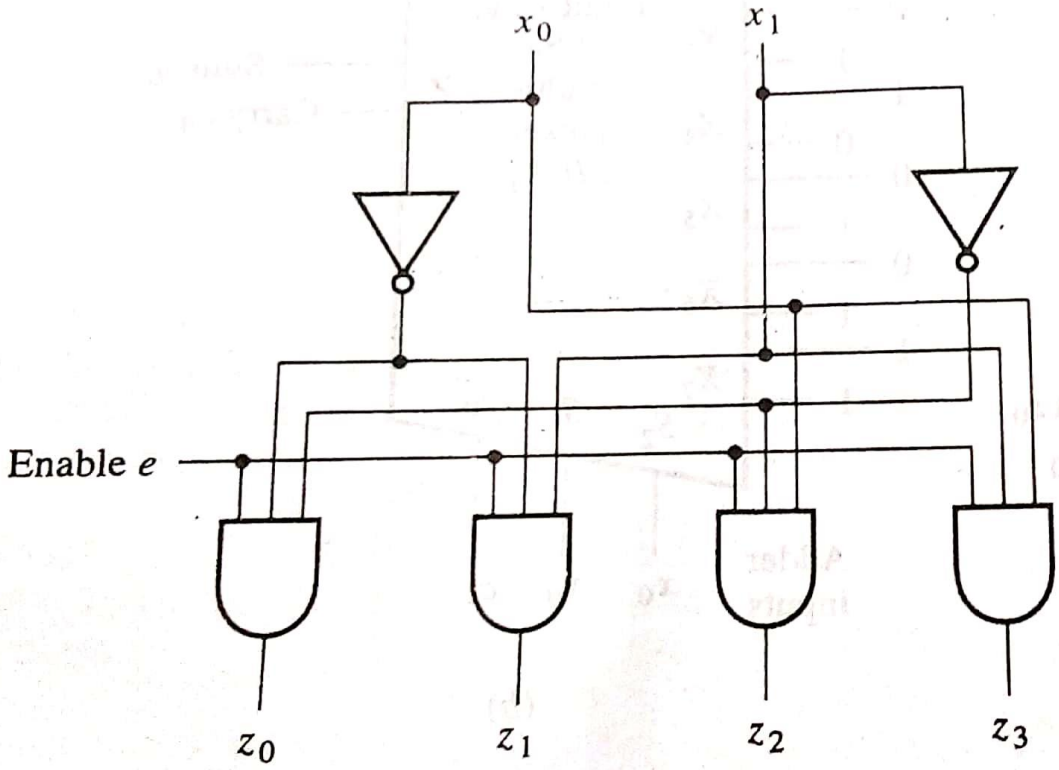
DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Decoders

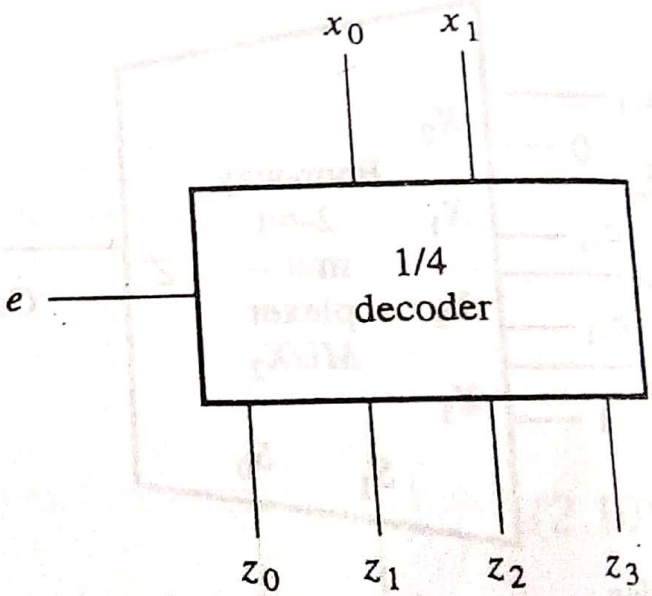
- A 1-out-of-2^n or $1/2^n$ decoder is a combinational circuit with n input lines X and 2^n output lines Z such that each of the 2^n possible input combinations A_i applied to X activates a corresponding output line z_i
- Application of decoder is in address decoding.
- Decoders are used in RAMs to select storage cells to be read from or written to.
- Another application : Routing data from a common source to one of several destinations – **demultiplexer.**

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Decoders



(a)



(b)

Figure 2.23
A 1/4 decoder: (*a*) logic diagram; (*b*) symbol.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Decoders

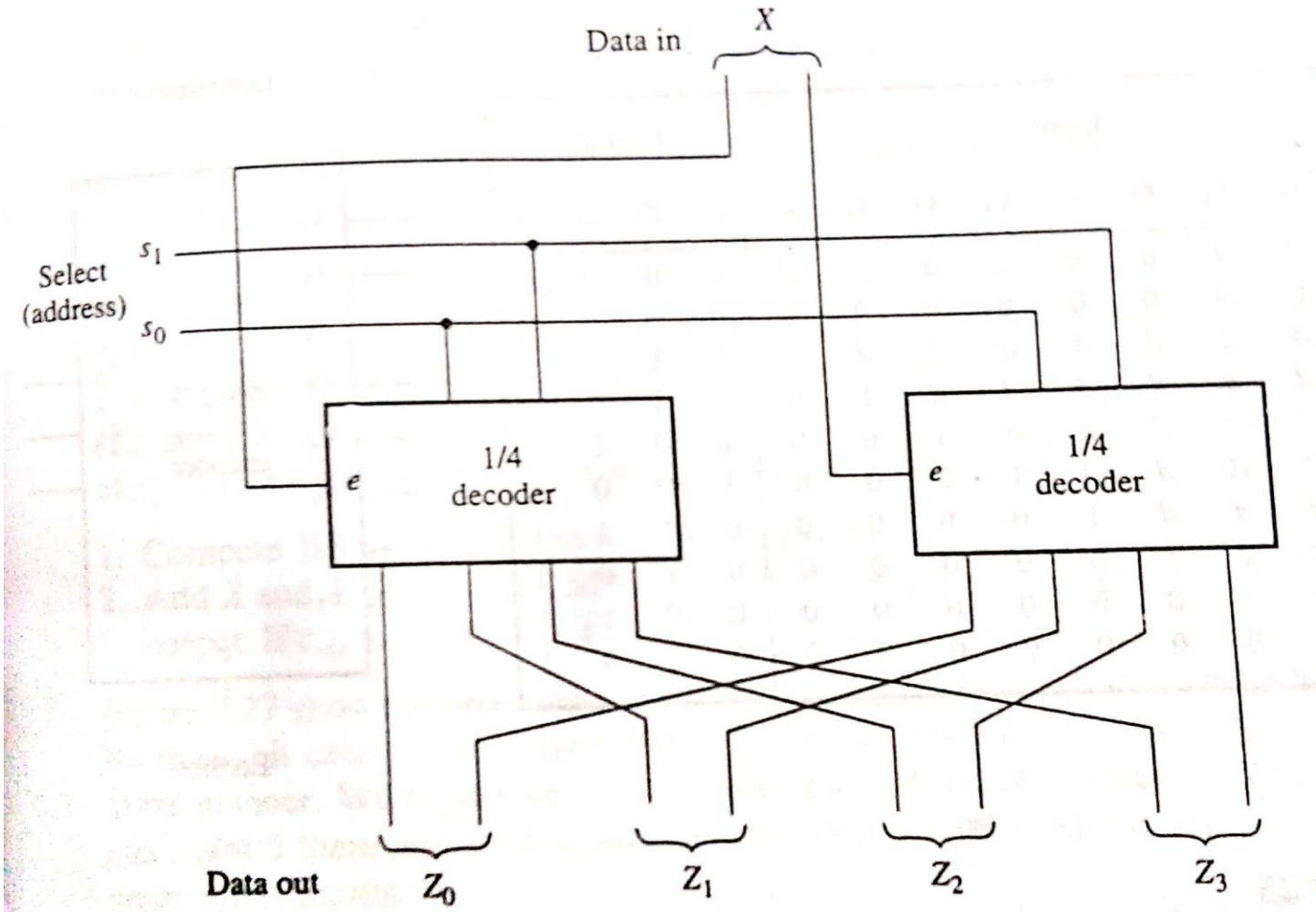


Figure 2.24
A four-output, 2-bit demultiplexer.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Encoders

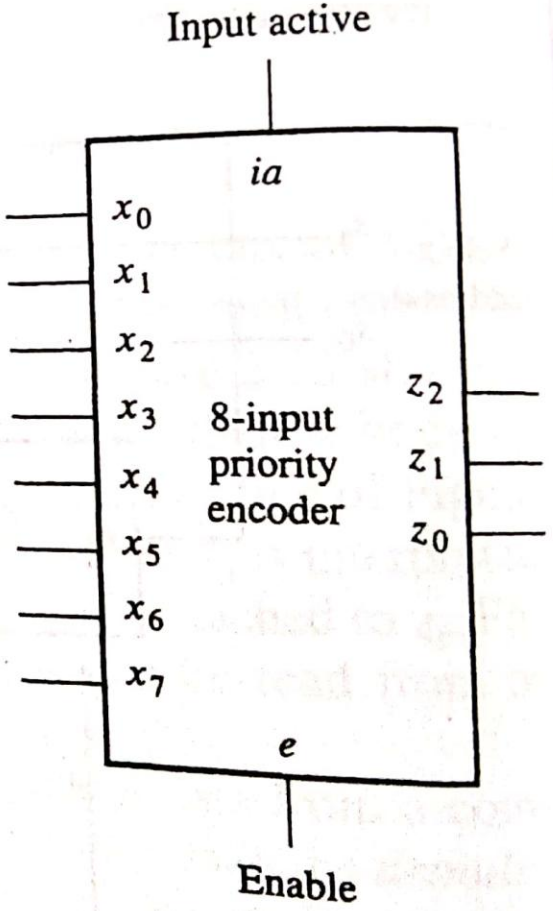
- An **encoder** is a circuit intended to generate the address or index of an active input line; it is therefore the inverse of a decoder.
- 2^k input data lines and k output data lines.
- It is also necessary to assign priorities to the input lines and design the encoder so that the output address is always that of the active input line with the highest priority – **Priority encoder**.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Encoders

Inputs									Outputs			
e	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	z_2	z_1	z_0	ia
0	d	d	d	d	d	d	d	d	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0
1	d	d	d	d	d	d	d	1	1	1	1	1
1	d	d	d	d	d	d	1	0	1	1	0	1
1	d	d	d	d	d	1	0	0	1	0	1	1
1	d	d	d	d	1	0	0	0	1	0	0	1
1	d	d	d	1	0	0	0	0	0	1	1	1
1	d	d	1	0	0	0	0	0	0	1	0	1
1	d	1	0	0	0	0	0	0	0	0	1	1
1	1	0	0	0	0	0	0	0	0	0	0	1

(a)



(b)

Figure 2.25
An 8-input priority encoder: (*a*) truth table; (*b*) symbol.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Arithmetic elements

- Addition and subtraction of fixed-point numbers can be implemented by combinational register-level components.
- 4-bit adder – adds two 4-bit data words and an input carry bit.
- Magnitude comparator – compare the magnitudes of two binary numbers.

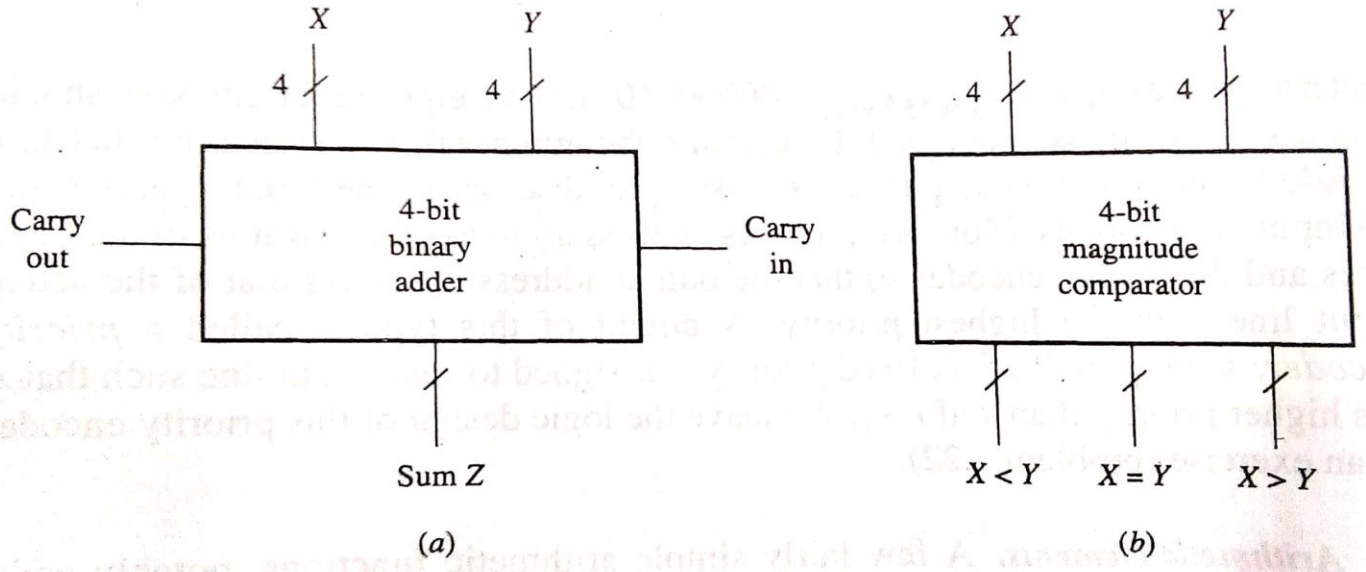


Figure 2.26
Symbols for (*a*) a 4-bit parallel adder; (*b*) a 4-bit magnitude comparator.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Arithmetic elements

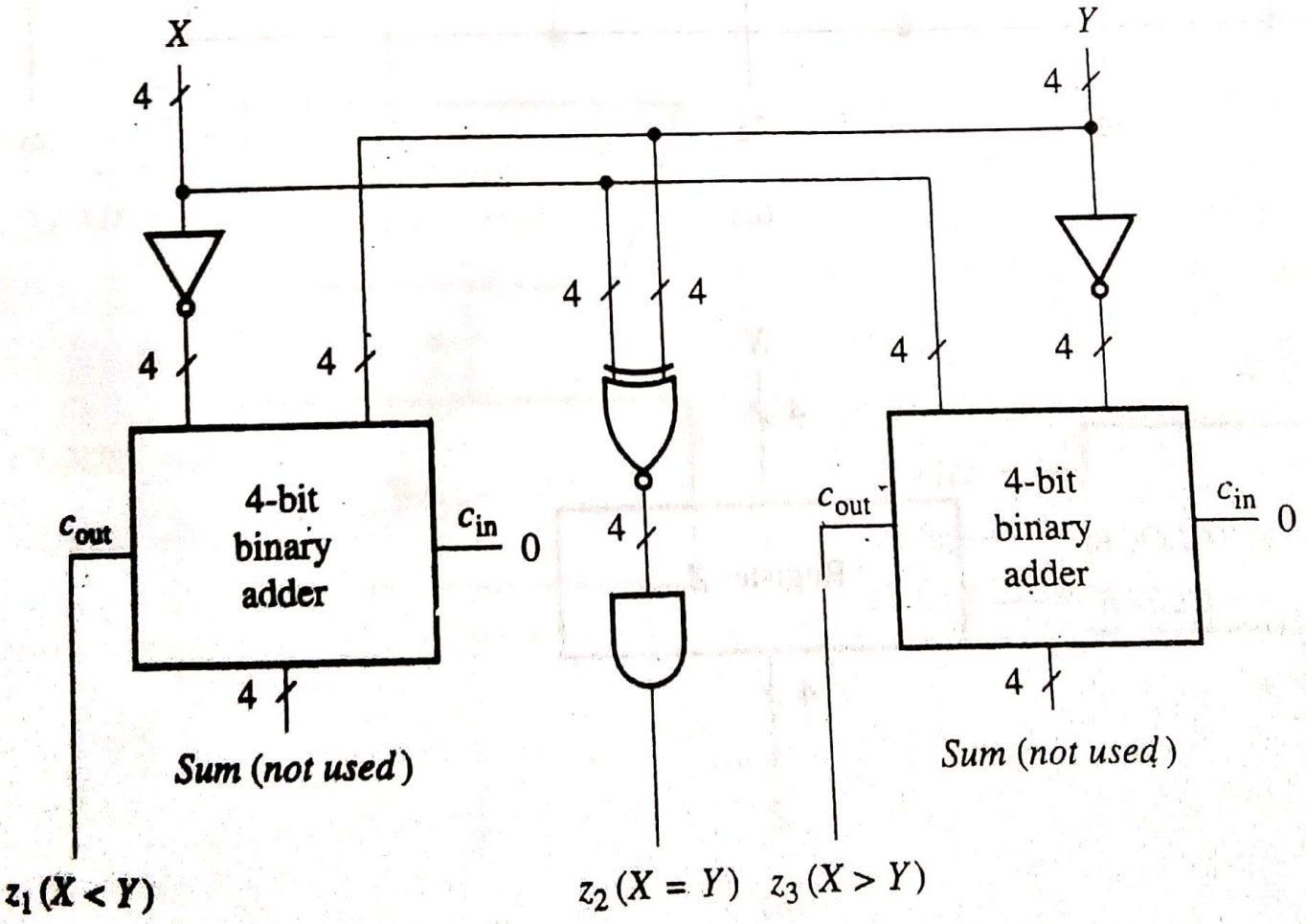
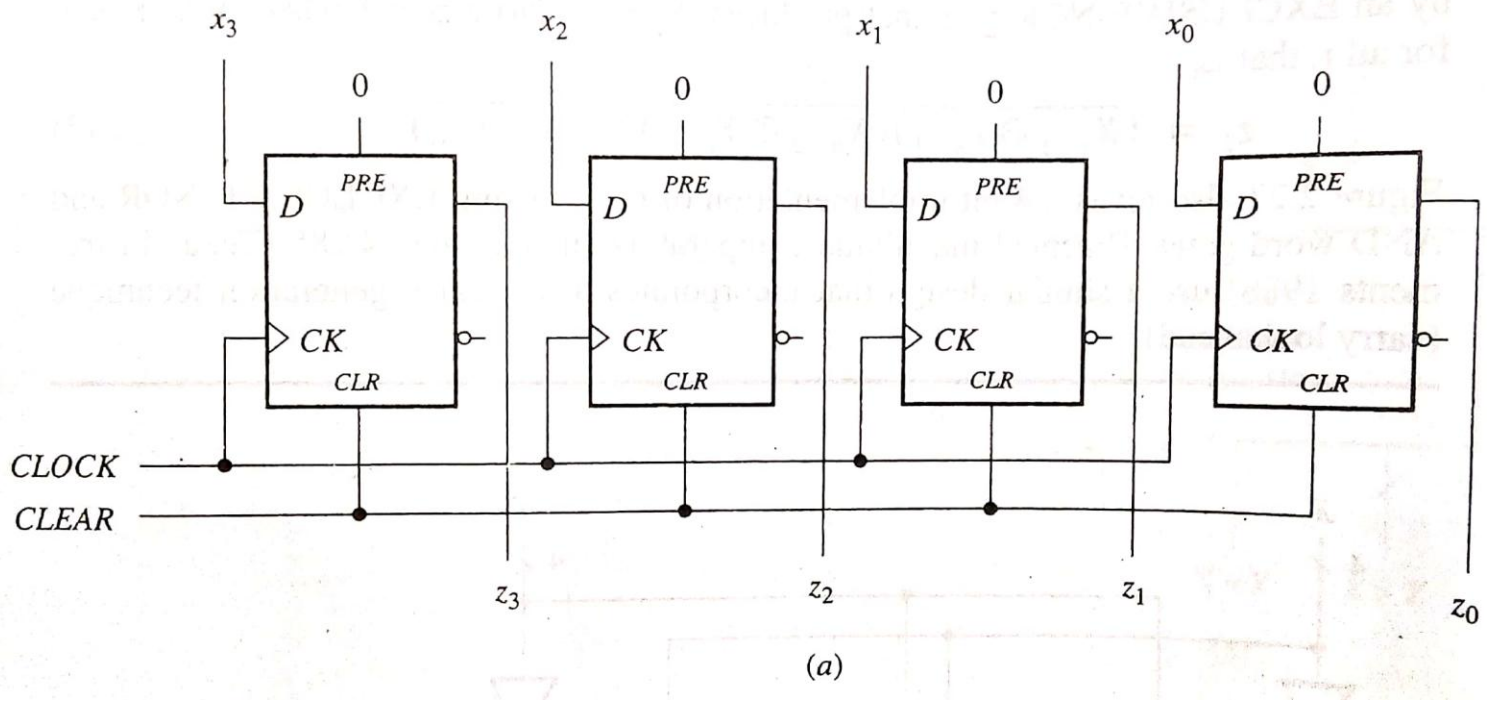


Figure 2.27
Register-level design of a 4-bit magnitude comparator.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Registers

- An **m-bit register** is an ordered set of m flipflops designed to store an m-bit word $(z_0, z_1, \ldots, z_{m-1})$.
- Each bit of the word is stored in a separate flipflop, but the flipflops have common control lines (clock, clear, and so on).



DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Registers

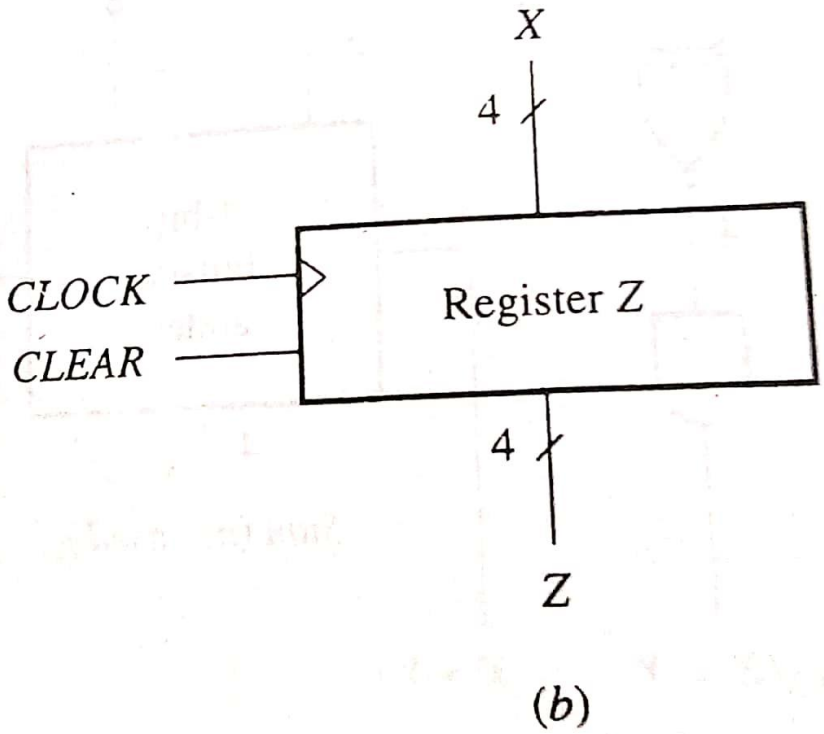


Figure 2.28
A 4-bit D register with parallel IO; (*a*) logic diagram; (*b*) symbol.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Registers

- LOAD feature enables **parallel input-output**.
- If LOAD=1, then X is loaded into the register from the input bus; that is Z=X
- If LOAD=0, then the old value of Z is loaded back into the register; that is Z=Z.

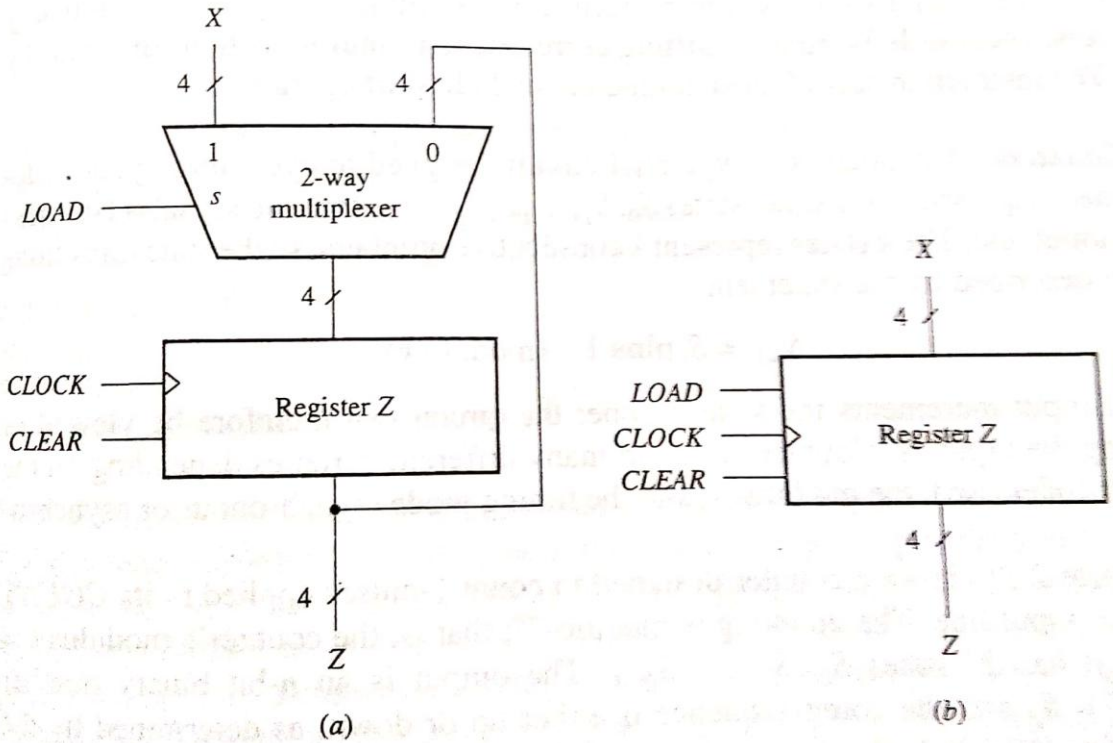


Figure 2.29
A 4-bit D register with parallel load: (*a*) logic diagram; (*b*) symbol.

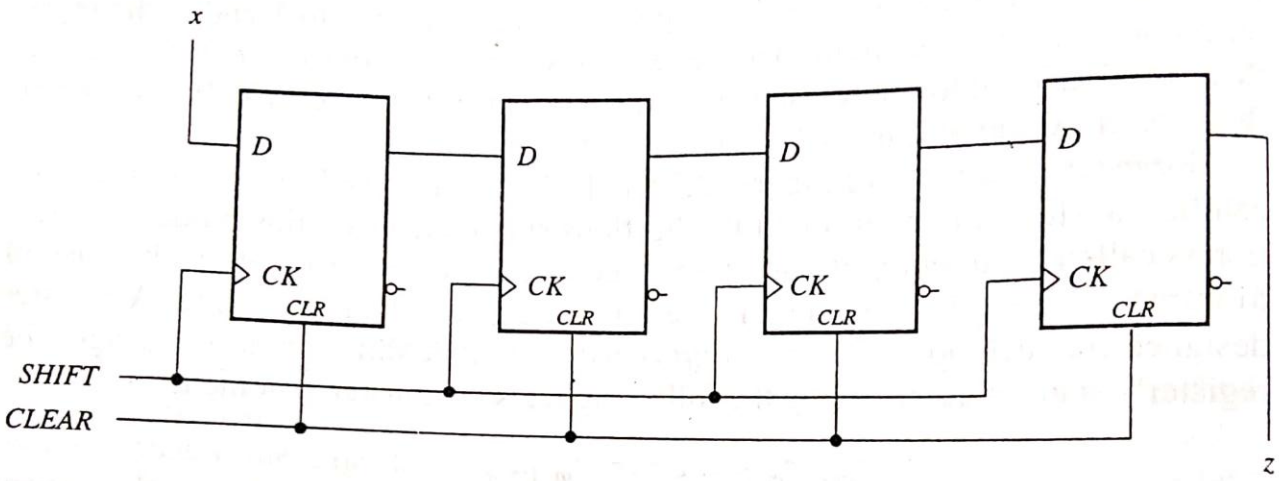
DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Registers

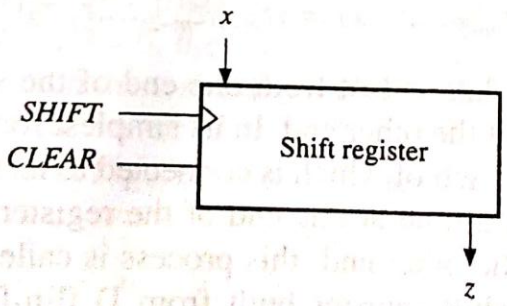
- Shift register - **serial input-output.**
- Data can be entered 1 bit at a time at one end of the register and can be removed (read) 1 bit at a time from the other end.
- A right shift is accomplished by activating the SHIFT enable line connected to the clock input CK of each flipflop.
- In addition to the serial data lines, m input or output lines are often provided to permit parallel data transfers to or from the shift register.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Registers



(a)



(b)

Figure 2.30
A 4-bit, right-shift register: (*a*) logic diagram; (*b*) symbol.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Counters

- A **counter** is a sequential circuit designed to cycle through a predetermined sequence of k distinct states $S_0, S_1,\ldots,S_{k-1}$ in response to signals (1-pulses) on an input line.

$$S_{i+1}=S_i \text{ plus } 1 \text{ (modulo k)}$$

- For 2^n states, $S_0, S_1,\ldots,S_{2^n-1}$,

$S_{i+1}=S_i$ plus 1 (modulo 2^n) - up counting mode

$S_{i+1}=S_i$ minus 1 (modulo 2^n) - down counting mode

- Applications – store the state of control unit (program counter), generate timing signals and introduce precise delay s into a system.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Buses

- A **Bus** is a set of lines (wires) designed to transfer all bits of a word from a specified source to a specified destination on the same or a different IC; the source and destination are typically registers.
- Unidirectional or bidirectional
- Bus performs no logical function
- cost – they require logic circuits to control access to them and when used over long distances, signal amplification circuits (drivers and receivers).
- To reduce cost – shared bus is used
- A **shared bus** is one that can connect one of several sources to one of several destinations. – but requires more complex bus-control mechanisms.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Programmable Logic Devices (PLDs)

- ICs containing many gates or other general purpose cells whose interconnections can be configured or programmed to implement any desired combinational or sequential function.
- Easy to design and inexpensive to manufacture.
- Key technology for building application specific integrated circuits (ASICs).
- Two techniques are used to program PLDs : mask programming and field programming.
- Some field-programmable PLDs are erasable, implying that the same IC can be reprogrammed many times.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Programmable Logic Devices (PLDs)

- Programmable arrays : The connections leading to and from logic elements in a PLD contain transistors switches that can be programmed to be permanently switched on or switched off.
- Switches are laid out in two dimensional array so that large gates can be implemented with minimum IC area.
- x denote a programmable connection or **crosspoint** in a gate's input line.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Programmable Logic Devices (PLDs)

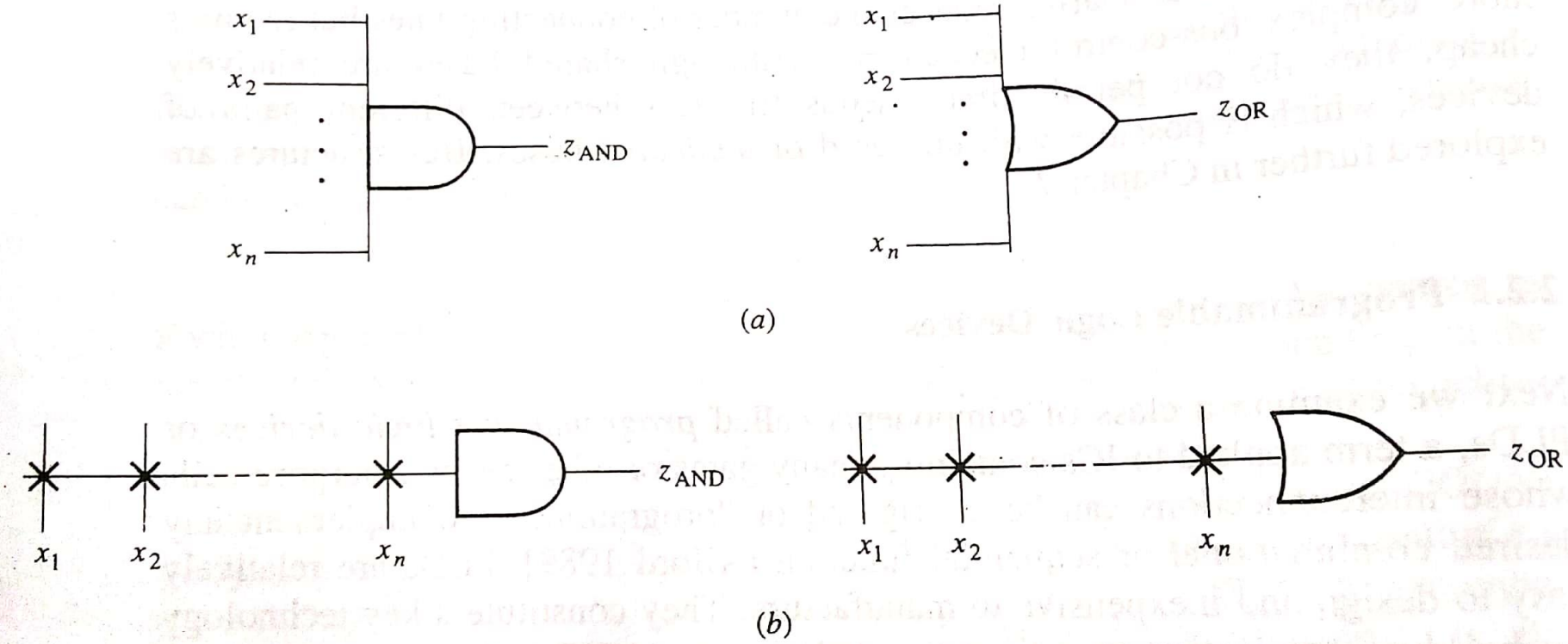


Figure 2.32
AND and OR gates: (*a*) normal notation; (*b*) PLD notation.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Programmable Logic Devices (PLDs)

➢ The **programmable logic array (PLA)** is intended to realize a set of combinational logic functions in minimal SOP form.

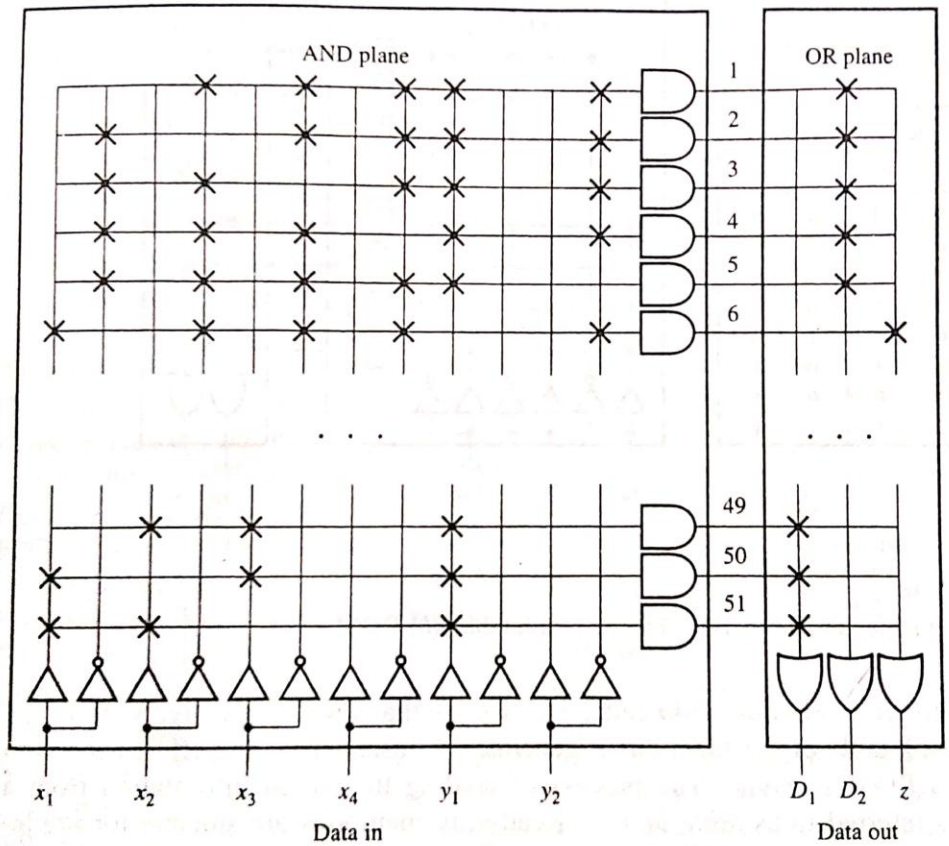


Figure 2.33
PLA implementing the combinational part *C* of the adder of Figure 2.13.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Programmable Logic Devices (PLDs)

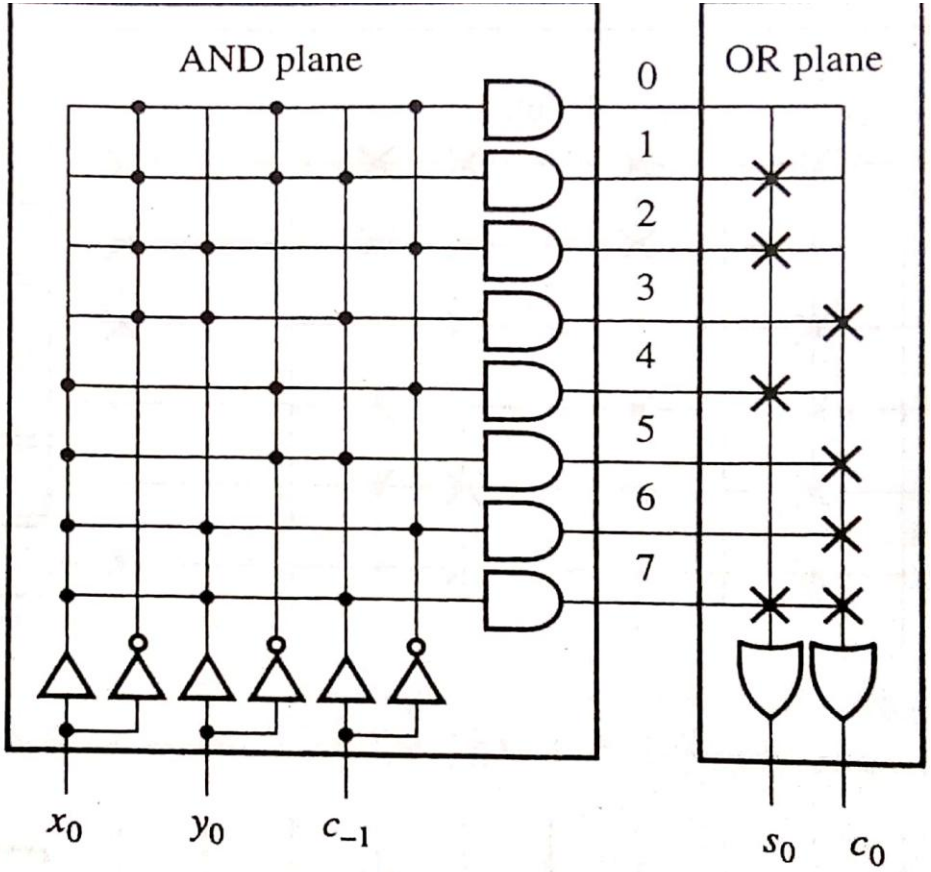
- Read-only memory (ROM) – generates all 2^n possible n-variable product terms (minterms) in its AND plane.
- This enables each output column of the OR plane to realize any desired function of n or fewer variables in sum of minterms form.
- dots denote fixed connections in the AND plane.
- The process of reading the stored information from a ROM is reffered to as table lookup.
- **Programmable array logic (PAL)** circuits have an AND plane that is programmable, but an OR plane with fixed connections designed to link each output line to a fixed set of AND rows, typically about eight rows.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Programmable Logic Devices (PLDs)

Inputs			Outputs	
x_0	y_0	c_{-1}	s_0	c_0
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a)



(b)

Figure 2.34
ROM implementation of a full adder: (*a*) truth table; (*b*) ROM array.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Field Programmable Gate Arrays (FPGA)

- A FPGA is a two-dimensional array of general purpose logic circuits, called **cells** or **logic blocks**, whose functions are programmable; the cells are linked to one another by programmable buses.
- The cells are not restricted to gates.
- They are small multifunction circuits capable of realizing all Boolean functions of a few variables; a cell may also contain one or two flip-flops.
- FPGAs can store the program that determines the circuit to be implemented in a RAM or PROM on the FPGA Chip.
- The pattern of the data in this configuration memory CM determines the cells functions and their interconnections wiring.
- Each bit of the CM controls a transistor switch in the target circuit that can select some cell function or make (break) some connection.

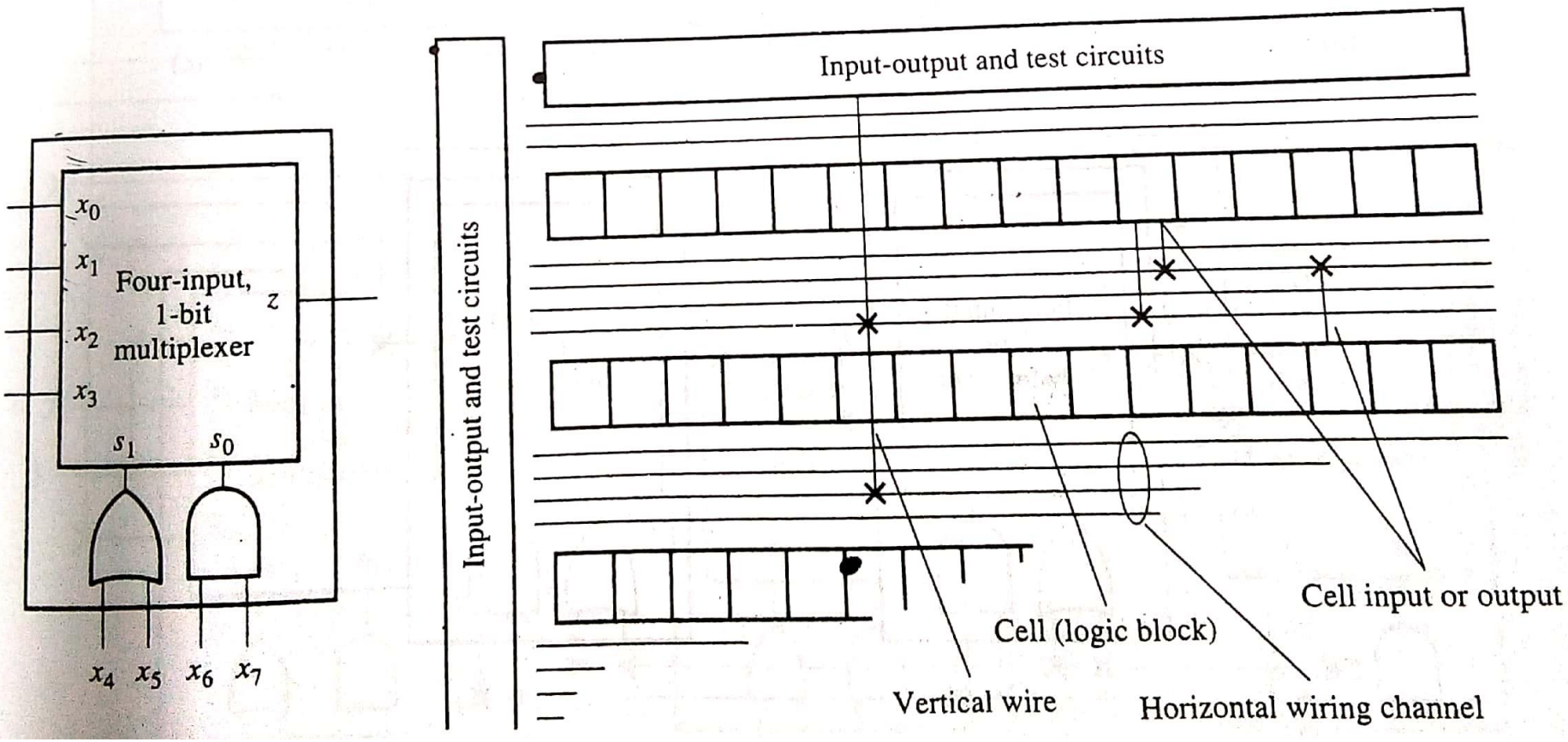
DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Field Programmable Gate Arrays (FPGA)

- By replacing the contents of CM, designers can make design changes or correct design errors.
- Two types of logic cells found in FPGAs are those based on multiplexers and those based on PROM table-lookup memories.

DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level: Field Programmable Gate Arrays (FPGA)



DESIGN METHODOLOGY : SYSTEM DESIGN

The Register level Design :

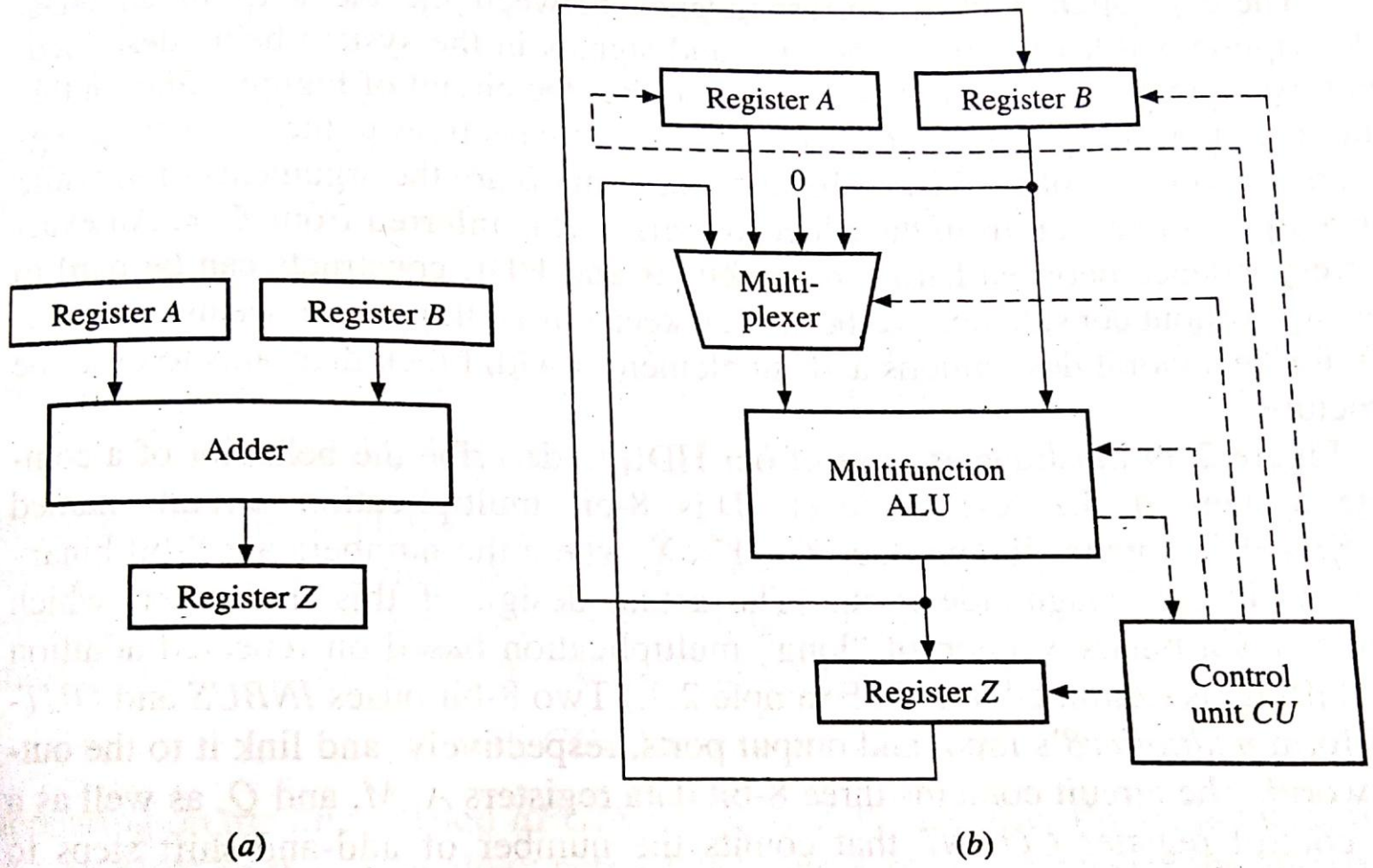
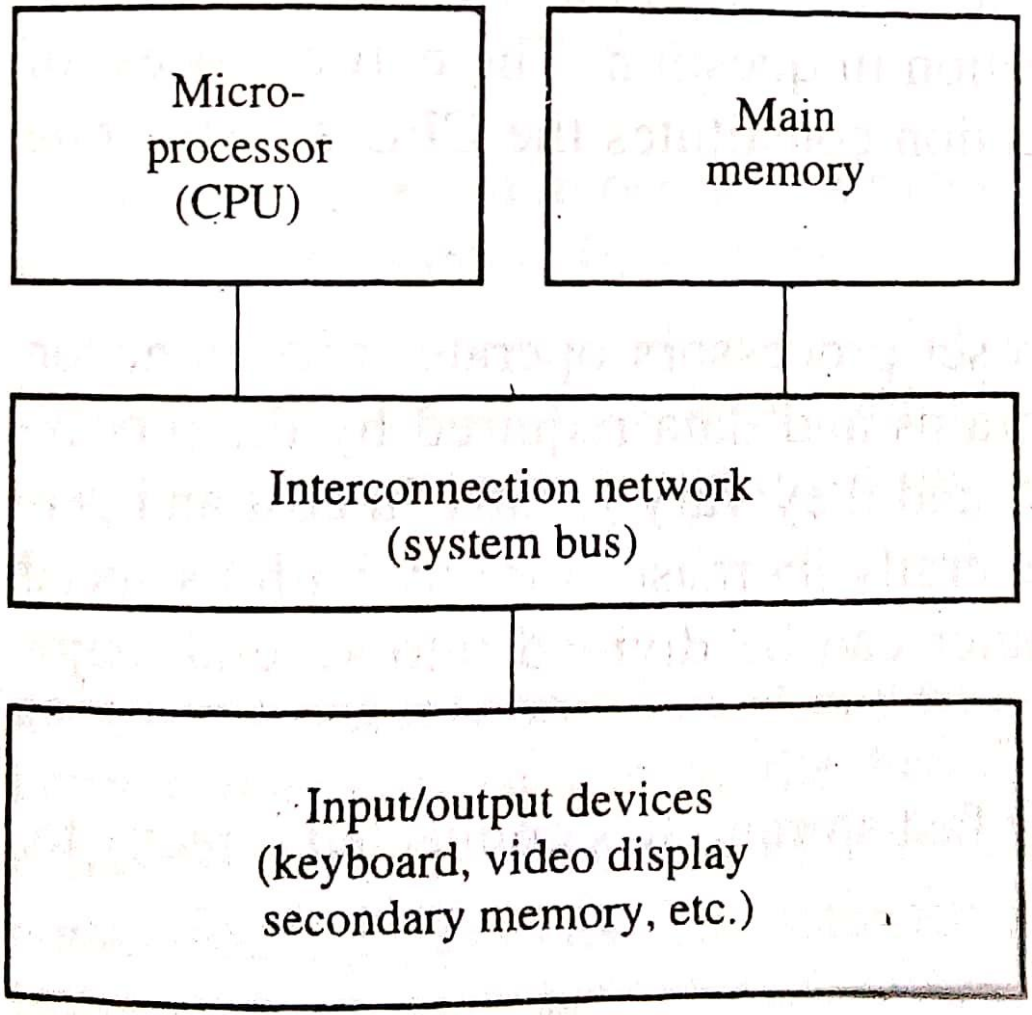


Figure 2.38
(*a*) Single-function circuit performing $Z := A + B$; (*b*) a multifunction circuit.

DESIGN METHODOLOGY : SYSTEM DESIGN

Major components of computer system:



DESIGN METHODOLOGY : SYSTEM DESIGN

Internal organization of a CPU and Cache memory:

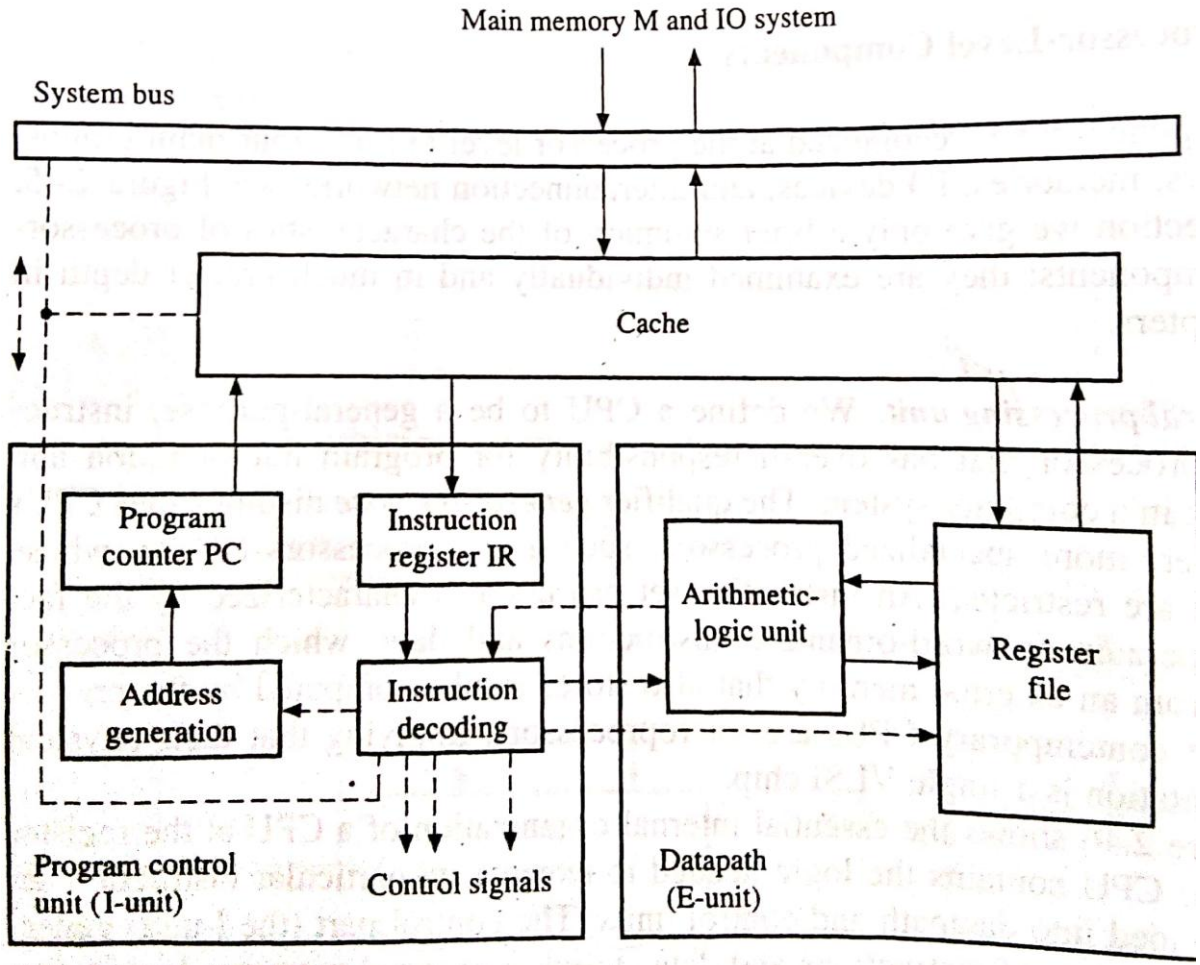
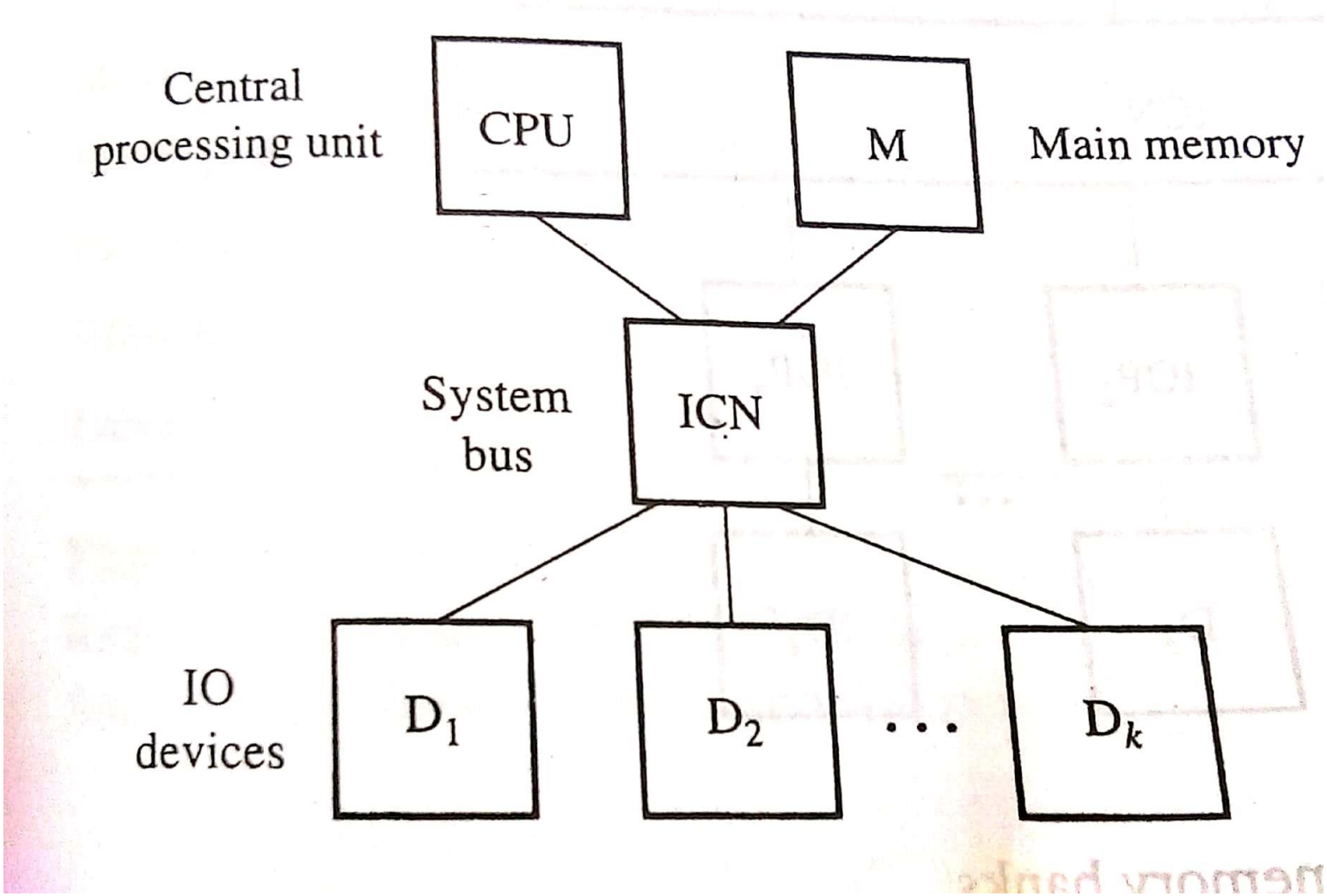


Figure 2.46
Internal organization of a CPU and cache memory.

DESIGN METHODOLOGY : SYSTEM DESIGN

Basic Computer Structure:



DESIGN METHODOLOGY : SYSTEM DESIGN

Computer with Cache and I/O processor:

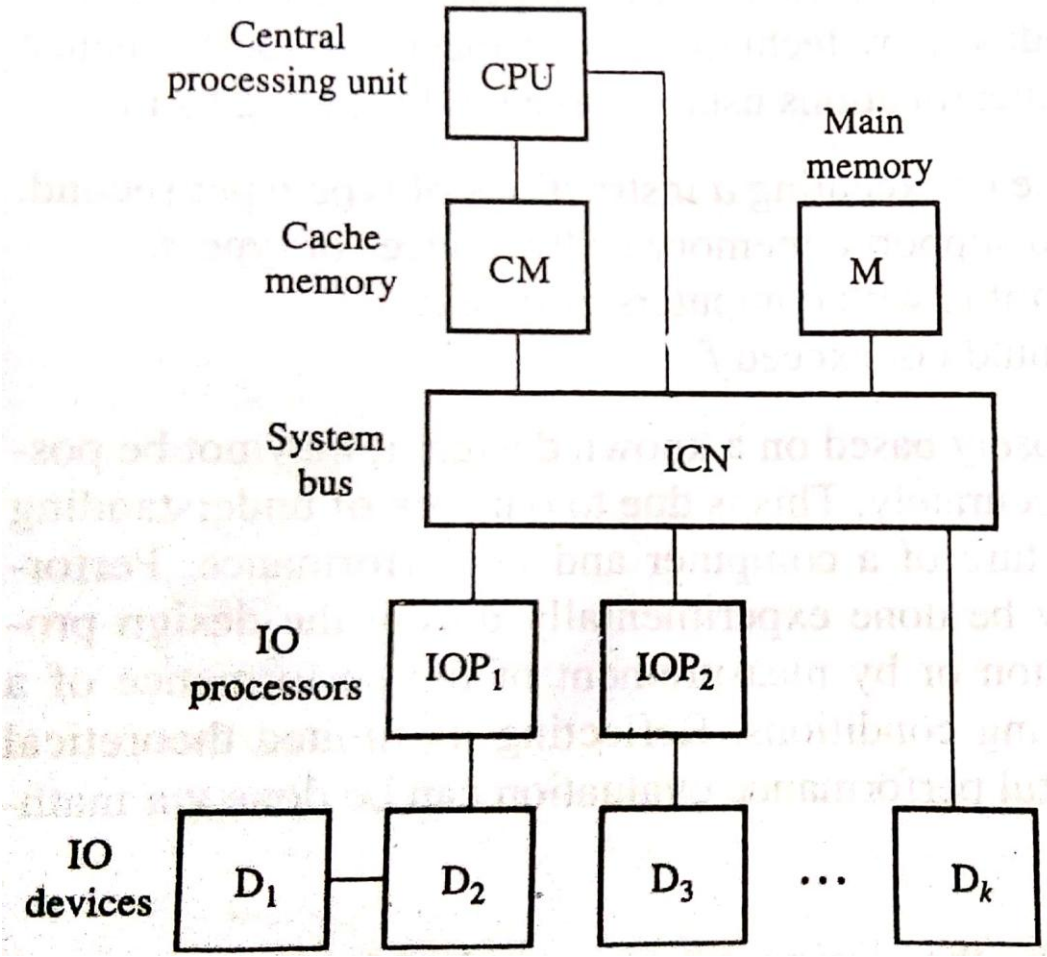


Figure 2.49 Computer with cache and IO processors.

DESIGN METHODOLOGY : SYSTEM DESIGN

Computer with multiple CPU and main memory bank:

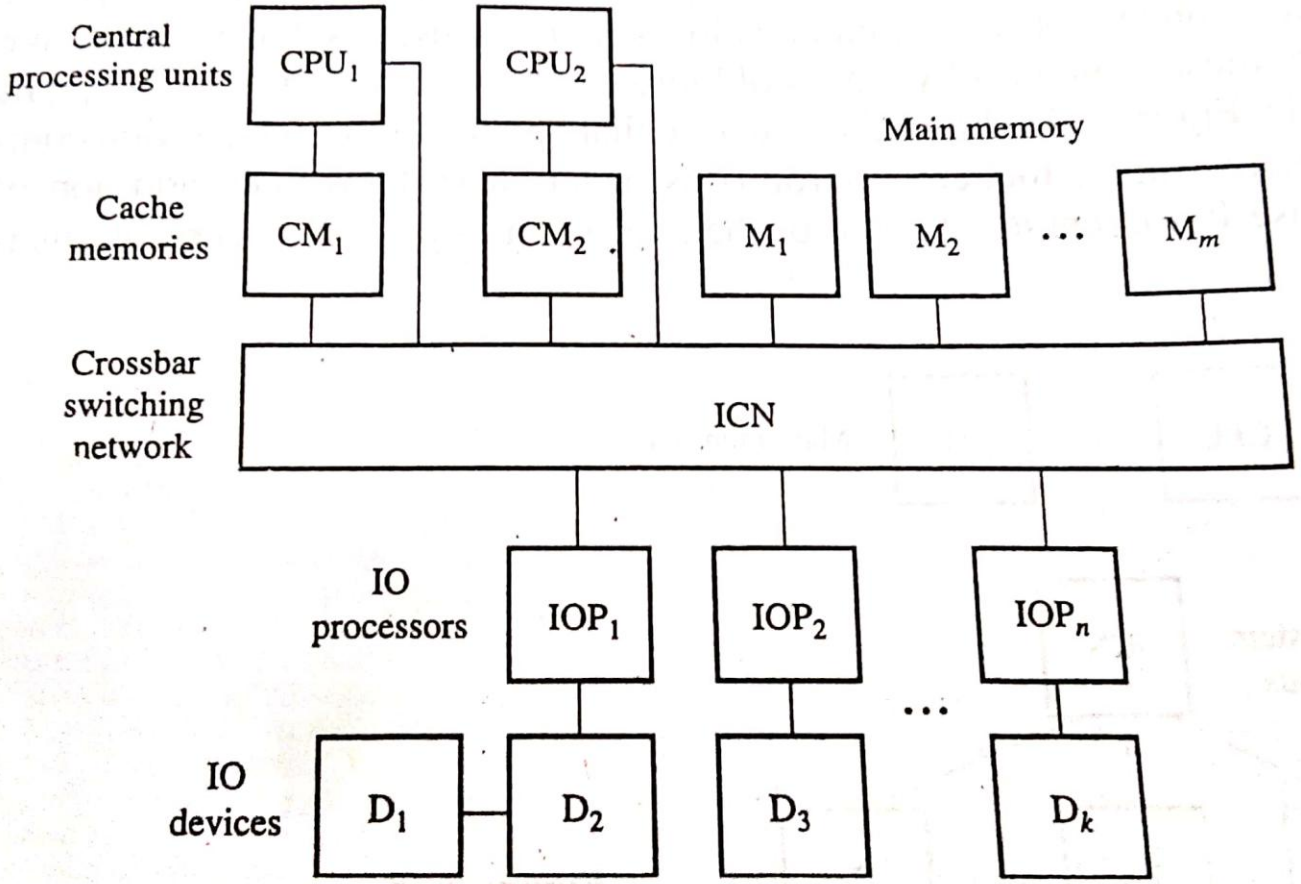


Figure 2.50
Computer with multiple CPUs and main memory banks.

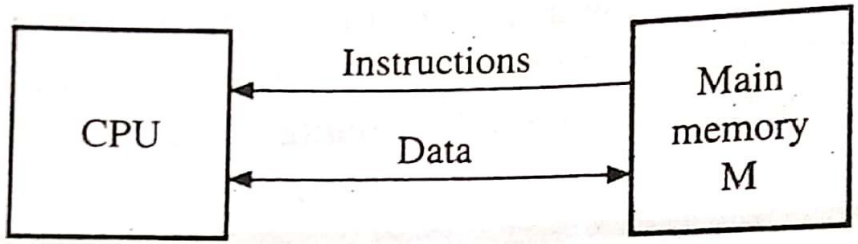
PROCESSOR BASICS : CPU ORGANIZATION

➢ Primary function of CPU – To execute sequences of instructions (or programs) stored in an external main memory.

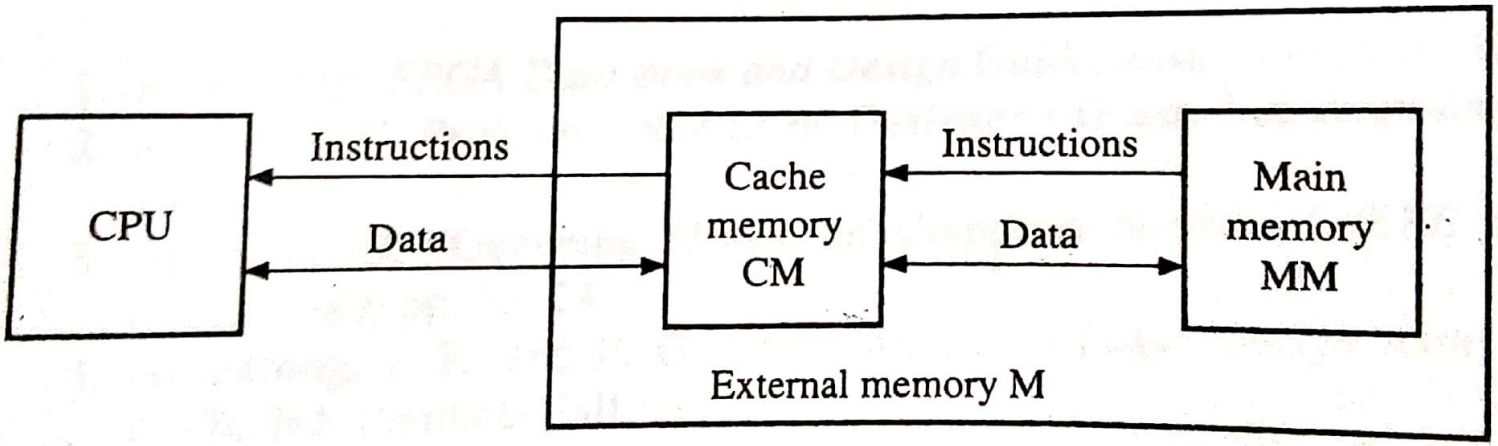
➢Program execution is carried out as follows:

1. The CPU transfers instructions and, when necessary, their input data (operands) from main memory to registers in the CPU.
2. The CPU executes the instructions in their stored sequence except when the execution sequence is explicitly altered by a branch instruction.
3. When necessary, the CPU transfers output data (results) from the CPU registers to main memory.

PROCESSOR BASICS : CPU ORGANIZATION



(*a*)



(*b*)

Figure 3.1
Processor-memory communication: (*a*) without a cache and (*b*) with a cache.

PROCESSOR BASICS : CPU ORGANIZATION

- External communication : If no cache memory is present, the CPU communicates directly with the main memory M, which is typically a high capacity multichip random access memory (RAM).
- CPU is significantly faster than M
- CPU can read or write the registers, 5 to 10 times faster than it can read or write to M.
- To remedy this, many computers have a cache memory CM positioned between the CPU and main memory.
- The cache CM is smaller and faster than main memory and may reside, wholly or in part, on the same chip as the CPU.
- It typically permits the CPU to perform a memory load or store operation in a single clock cycle.
- The cache is designed to be transparent to the CPU instructions, which "see" the cache and main memory as forming a single, seamless memory space consisting of 2^m addressable storage locations M(0), M(1), …. , M(2^m-1)

PROCESSOR BASICS : CPU ORGANIZATION

- The CPU communicates with IO devices the same way as it communicates with external memory.
- The IO devices are associated with addressable registers called **IO ports** to which the CPU can store a word (an output operation) or from which it can load a word (an input operation).
- IO data transfers are implemented by memory-referencing instructions, an approach called **memory-mapped IO.**
- Memory locations and IO ports share the same set of addresses.
- Memory mapped IO – no IO instructions, only memory-referencing instructions
- **IO-mapped IO** – IO instructions produce control signals to which IO ports but not memory locations, respond.

PROCESSOR BASICS : CPU ORGANIZATION

User and supervisor modes:

1. A user or application program handles a specific application, such as word processing.
2. A supervisor program, manages various routine aspects of the computer system on behalf of its users; it is typically part of the computer's operating system. Eg: Controlling a graphics interface and transferring data between secondary and main memory.

- In normal operation the CPU continually switches back and forth between the user and supervisor programs.
- Interrupt mechanism – The CPU suspends execution of the program that it is currently executing and transfers to an appropriate interrupt-handling program.
- As interrupts, particularly from IO devices, require a rapid response from the CPU, it checks frequently for the presence of interrupt request.

PROCESSOR BASICS : CPU ORGANIZATION

CPU operation :

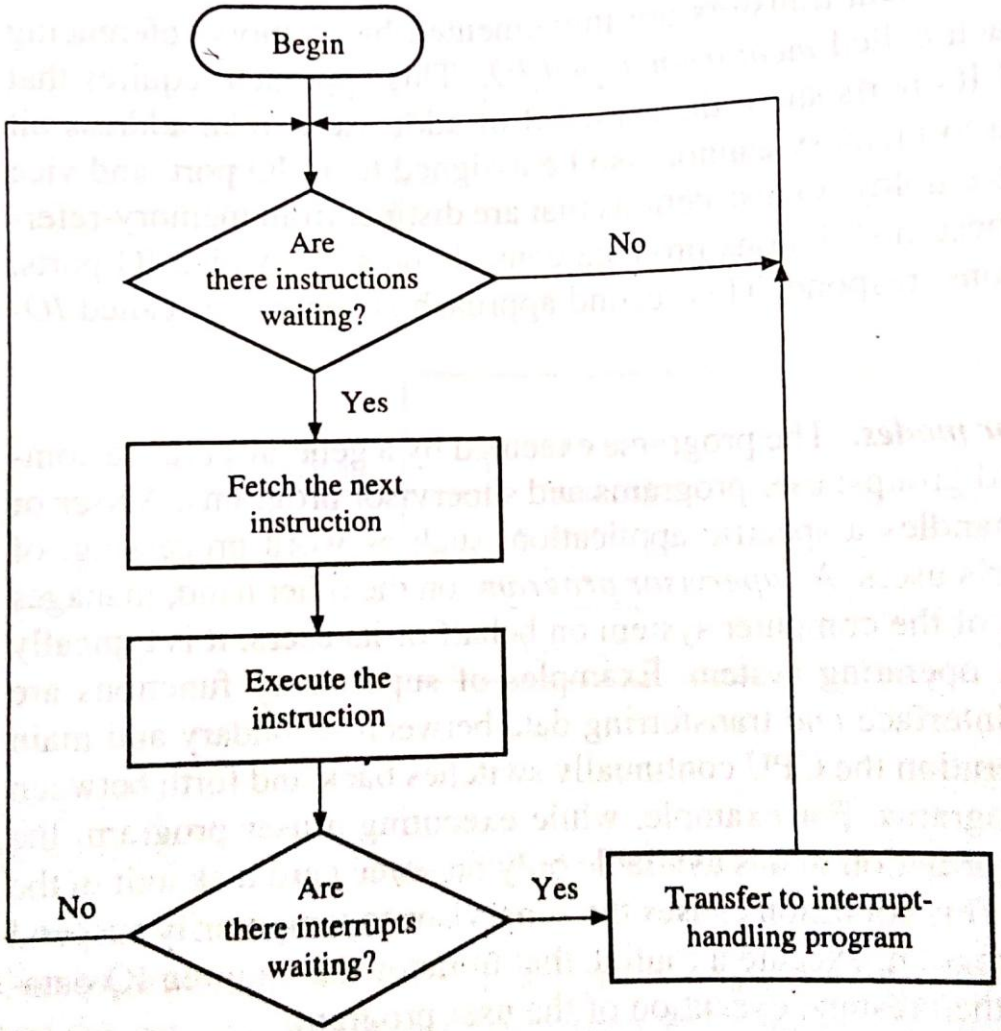


Figure 3.2
Overview of CPU behavior.

PROCESSOR BASICS : CPU ORGANIZATION

CPU operation :

- The sequence of operations performed by the CPU in processing an instruction constitutes an **instruction cycle**.
- Instruction cycle vary with the type of instruction
- Instruction cycle requires two major steps :

1. a **fetch step** – a new instruction is read from the external memory M
2. an **execute step** – operations specified by the instruction are executed.

- A check for pending interrupt requests is also usually included in the instruction cycle.
- The actions of the CPU during an instruction cycle are defined by a sequence of microoperations.
- The time required for the shortest well-defined CPU microoperation is the **CPU cycle time** or **clock period T_{clock}** – basic unit of time for measuring CPU actions.

PROCESSOR BASICS : CPU ORGANIZATION

Accumulator based CPU:

- CPU organization proposed by von Neumann and his colleagues for the IAS computer is the basics for most subsequent designs.
- It comprises a small set of registers and the circuits needed to execute a functionally complete set of instructions.
- One of the CPU registers, the **accumulator**, played a central role being used to store an input or output operand (result) in the execution of many instructions.
- Assume for simplicity that instructions and data have some fixed word size n bits.

PROCESSOR BASICS : CPU ORGANIZATION

Accumulator based CPU:

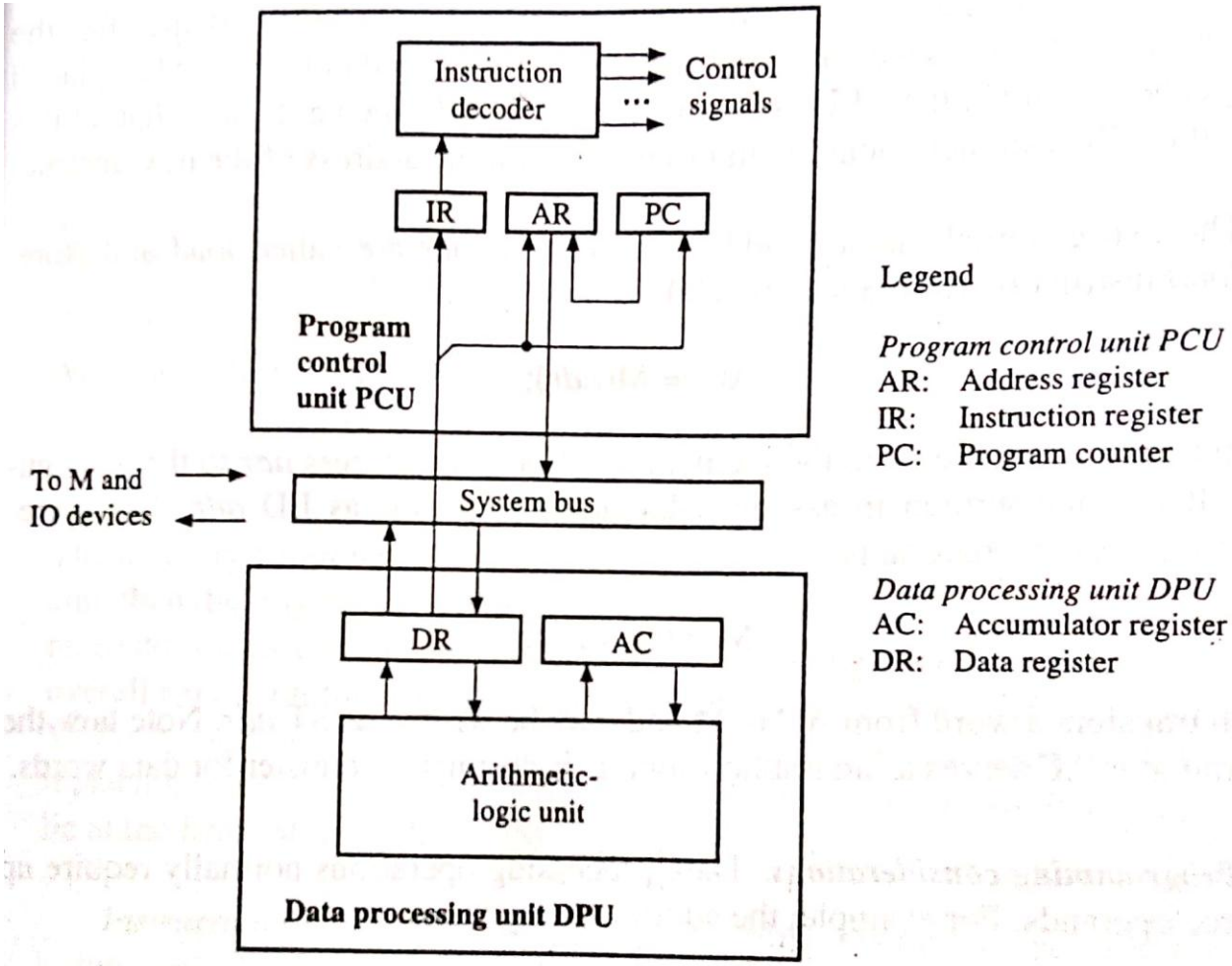


Figure 3.3
A small accumulator-based CPU.

PROCESSOR BASICS : CPU ORGANIZATION

Accumulator based CPU:

- Instructions are fetched by program control unit PCU, whose main register is the program counter PC.
- They are executed in the data processing unit DPU, which contains an n-bit arithmetic and logic unit (ALU) and two data registers Accumulator (AC) and data register (DR).
- Most instructions perform operations of the form,

$$X1 := f_i(X1, X2)$$

Where X1 and X2 denote a CPU register (AC, DR or PC) or an external memory location M(adr).

- The operations f_i performed by the ALU are limited to fixed point (integer) addition and subtraction, shifting and logical (word-gate) operations.
- Memory addresses are stored in two address registers in the PCU : the program counter PC (which stores instruction addresses only) and the general-purpose (data) address register AR.

PROCESSOR BASICS : CPU ORGANIZATION

Accumulator based CPU:

- An instruction I that refers to a data word in M contains two parts, an opcode **op** and a memory address **adr**, and may be written as I=op.adr

IR.AR := M(PC)

Which transfers the instruction word I from M to the CPU.

- The opcode op is loaded into the PCU's instruction register IR, and the address adr is loaded into address register AR.

IR := op , AR := adr

- Instructions that do not reference M do not use AR; their opcode part specifies the CPU registers to use, as well as the operation f_i to be carried out.
- The two essential memory-addressing instructions are called load and store.

PROCESSOR BASICS : CPU ORGANIZATION

Accumulator based CPU:

- The load instruction,

AC := M(adr)

Which transfers a word from the memory location with address adr to the accumulator.

- Assembly language program - LD adr.
- The store instruction is,

M(adr) := AC

Which transfers a word from AC to M

- Assembly language program - ST adr

PROCESSOR BASICS : CPU ORGANIZATION

Programming considerations:

- Data processing operations normally require up to three operands.

Z:=X+Y

- Accumulator based CPU supports only single-address instructions, that is, instructions with one explicit memory address
- AC and DR can serve as implicit operand locations.

HDL format	Assembly-language format	Narrative format (comment)
AC := M(X);	LD X	Load X from M into accumulator AC.
DR := AC;	MOV DR, AC	Move contents of AC to DR.
AC := M(Y);	LD Y	Load Y into accumulator AC.
AC := AC + DR;	ADD	Add DR to AC.
M(Z) := AC;	ST Z	Store contents of AC in M.

PROCESSOR BASICS : CPU ORGANIZATION

Programming considerations:

- The preceding program fragment uses only the load and store instructions to access memory, a feature called **load/store architecture**.
- A CPU can be designed to implement memory referencing instructions of the form,

$$AC := f_i(AC, M(adr))$$

HDL Format	Assembly-language format	Narrative format (comment)
AC := M(*X*);	LD *X*	Load *X* from M into accumulator AC.
AC := AC + M(*Y*);	ADD *Y*	Load *Y* into DR and add to AC.
M(Z) := AC;	ST *Z*	Store contents of AC in M.

- The memory referencing ADD Y instruction can be expected to take longer to execute than the original ADD instruction that references only CPU registers.

PROCESSOR BASICS : CPU ORGANIZATION

Instruction set:

Type	Instruction	HDL format	Assembly-language format	Narrative format (comment)
Data transfer	Load	AC := M(*X*)	LD X	Load *X* from M into AC.
	Store	M(*X*) := AC	ST X	Store contents of AC in M as *X*.
	Move register	DR := AC	MOV DR, AC	Copy contents of AC to DR.
	Move register	AC := DR	MOV AC, DR	Copy contents of DR to AC.
Data processing	Add	AC := AC + DR	ADD	Add DR to AC.
	Subtract	AC := AC – DR	SUB	Subtract DR from AC.
	And	AC := AC ***and*** DR	AND	And bitwise DR to AC.
	Not	AC := ***not*** AC	NOT	Complement contents of AC
Program control	Branch	PC := M(*adr*)	BRA adr	Jump to instruction with address *adr*.
	Branch zero	**if** AC = 0 **then** PC := M(*adr*)	BZ adr	Jump to instruction *adr* if AC = 0.

Figure 3.4
Instruction set for the CPU of Figure 3.3.

PROCESSOR BASICS : CPU ORGANIZATION

Instruction set:

➢ The arithmetic operation negation, for which many CPUs have a single instruction of the type AC := -AC.

HDL format	Assembly-language format	Narrative format (comment)
DR := AC;	MOV DR, AC	Copy contents X of AC to DR.
AC := AC – DR;	SUB	Compute AC = $X - X = 0$.
AC := AC – DR;	SUB	Compute AC = $0 - X = -X$.

PROCESSOR BASICS : CPU ORGANIZATION

Additional Features:

- Multipurpose register set for storing data and addresses – Replaces accumulator and auxiliary registers DR and AR – general register organization – The set of general registers is now usually referred to as a **register file**.
- Additional data, instruction and address types – several different word sizes and formats for instruction and data.
- Register to indicate computation status - A **status register** (also called a **condition code** or **flag register**) indicates infrequent or exceptional conditions resulting from the instruction execution – Also indicate user and supervisor states – Conditional branch instructions can test the status register.
- Program control stack – Various special registers and instructions facilitate the transfer of control among programs due to procedure calling or external interrupts – a part of external memory M as a push-down stack – It is intended for saving key information about an interrupted program via push operation.

PROCESSOR BASICS : CPU ORGANIZATION

Additional Features:

➢ Program control stack - So that the information can be retrieved later by pop operation. – A CPU address register called a **stack pointer** automatically keeps track of the stack's entry point.

PROCESSOR BASICS : CPU ORGANIZATION

Additional Features:

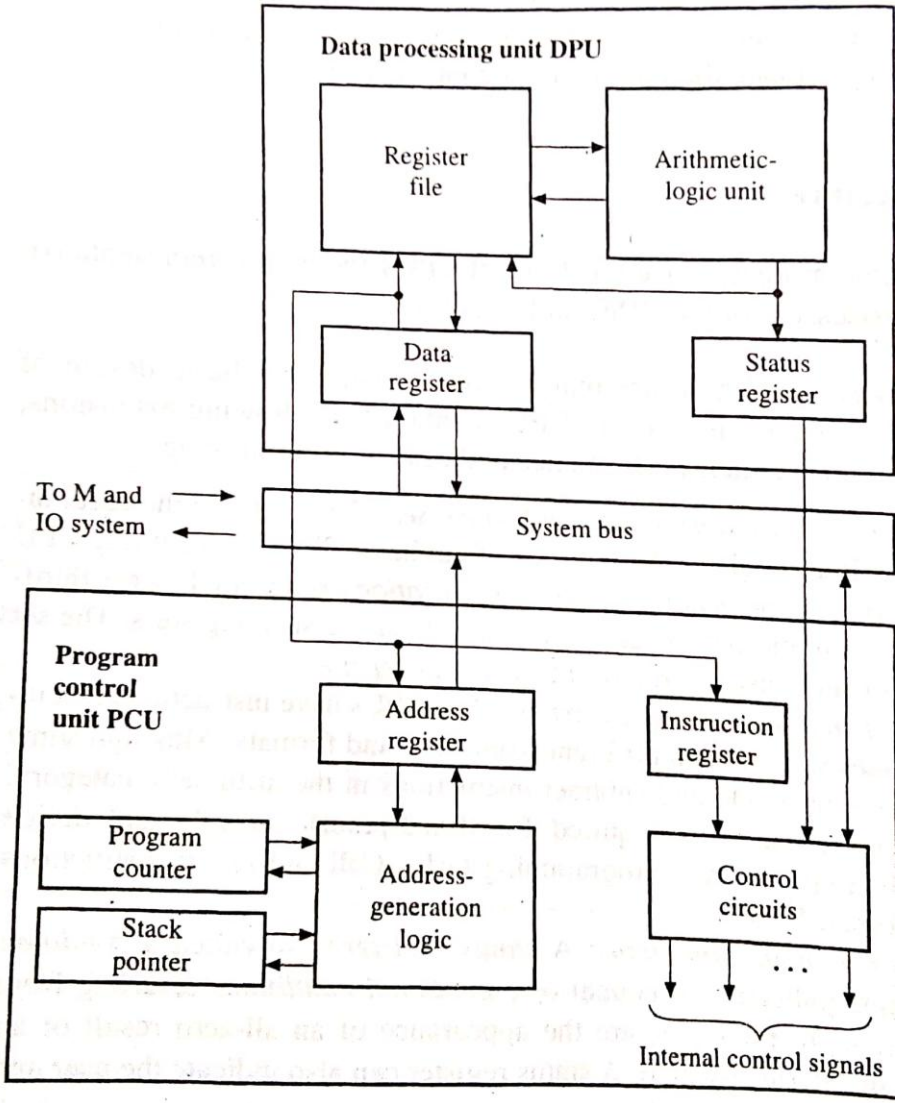


Figure 3.7
A typical CPU with the general register organization.

PROCESSOR BASICS : CPU ORGANIZATION

Pipelining:

- Performance can be improved by overlapping of the operations carried out by the DPU and PCU.
- Example : Negation

Clock cycle	Instruction cycle	PC	PCU actions	DPU actions
1	MOV DR, AC	2000	IR.AR := M(PC), PC := PC + 1	
2		2001		DR := AC
3	SUB	2001	IR.AR := M(PC), PC := PC + 1	
4		2002		AC := AC – DR
5	SUB	2002	IR.AR := M(PC), PC := PC + 1	
6		2003		AC := AC – DR

PROCESSOR BASICS : CPU ORGANIZATION

Pipelining:

➢ By merging the execution part of each instruction cycle with the fetch part of the following instruction cycle, we can reduce the overall execution time from six clock cycles to four.

Clock cycle	Instruction cycle	PC	PCU actions	DPU actions
1	MOV	2000	IR.AR := M(PC), PC := PC + 1	
2	MOV/SUB_1	2001	IR.AR := M(PC), PC := PC + 1	DR := AC
3	SUB_1/SUB_2	2002	IR.AR := M(PC), PC := PC + 1	AC := AC – DR
4	SUB_2	2003		AC := AC – DR

PROCESSOR BASICS : CPU ORGANIZATION

Pipelining:

- Branch instruction reduce the efficiency of instruction pipelining

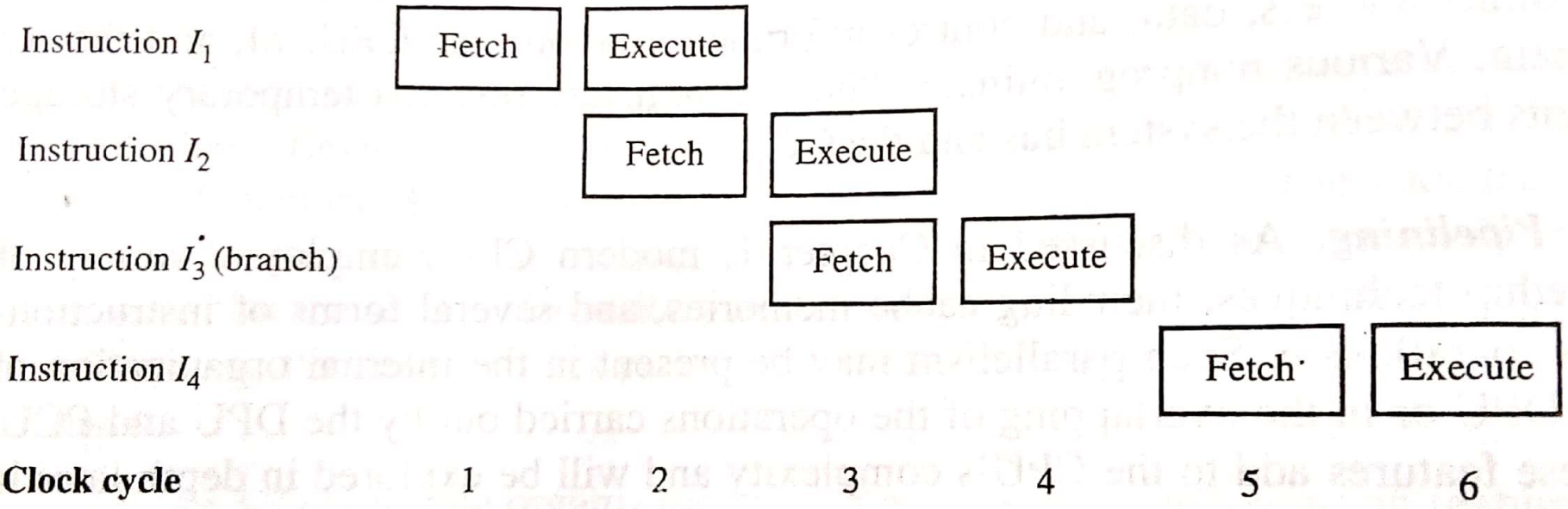


Figure 3.8
Overlapping instructions in a two-stage instruction pipeline.

PROCESSOR BASICS : CPU ORGANIZATION

A CISC machine:

- Motorola 68020 – a 32 bit machine – third generation mainframe computer – personal computer or workstation.
- Addresses – 32 bit long – 2^{32} different memory locations each storing 1 byte.
- Memory-mapped IO
- Register file containing – sixteen 32-bit registers
- ALU – can execute fixed-point instructions.
- 70 distinct instruction types (or around 200 if all opcode variants are distinguished)

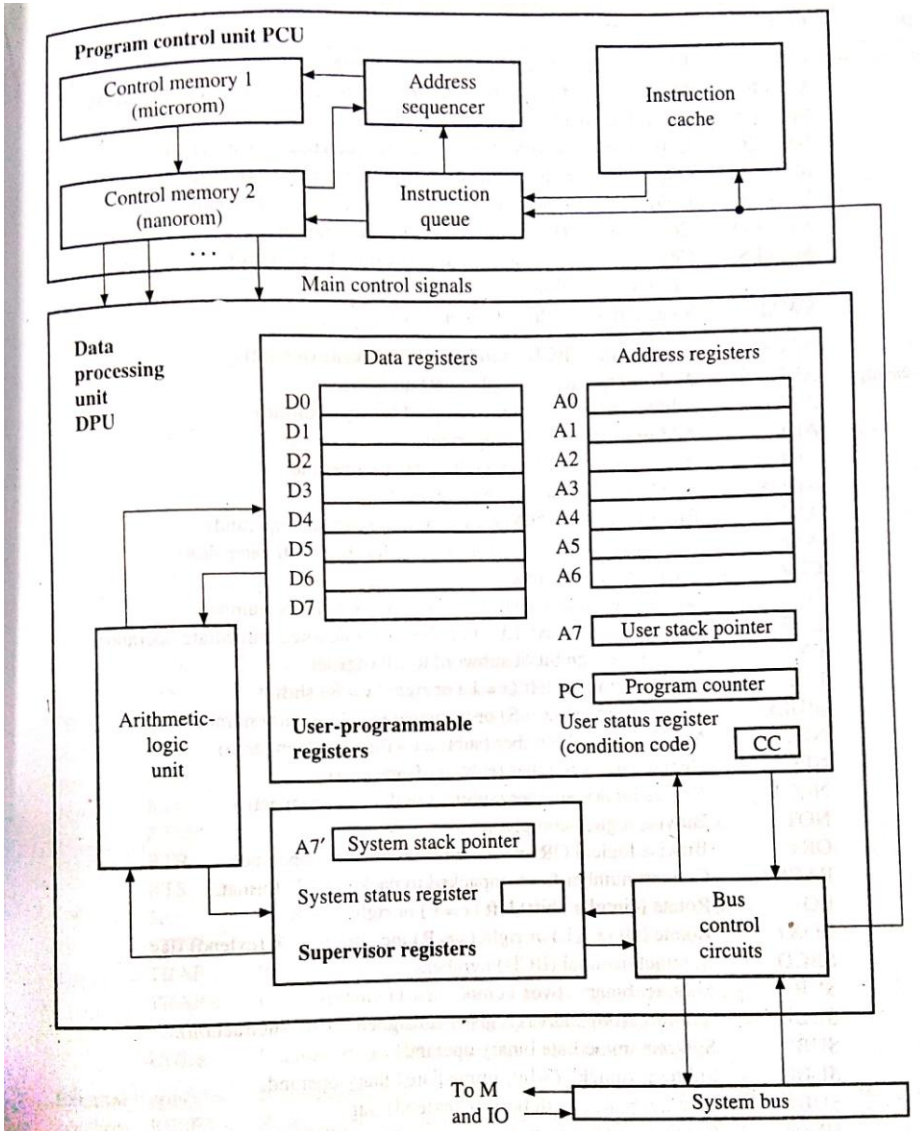


Figure 3.11
Organization of the 68020.

PROCESSOR BASICS : CPU ORGANIZATION

Coprocessors:

- Hardware implemented floating point instructions are not available in Motorola 68020, however they are provided indirectly by means of an auxiliary IC, the 68881 floating point coprocessor.
- A **coprocessor P** is a specialized instruction execution unit that can be coupled to a microprocessor so that instructions to be executed by P can be included in programs fetched by the microprocessor.
- Thus the coprocessor serves as a extension to the microprocessor and forms part of the CPU.

PROCESSOR BASICS : CPU ORGANIZATION

Coprocessors:

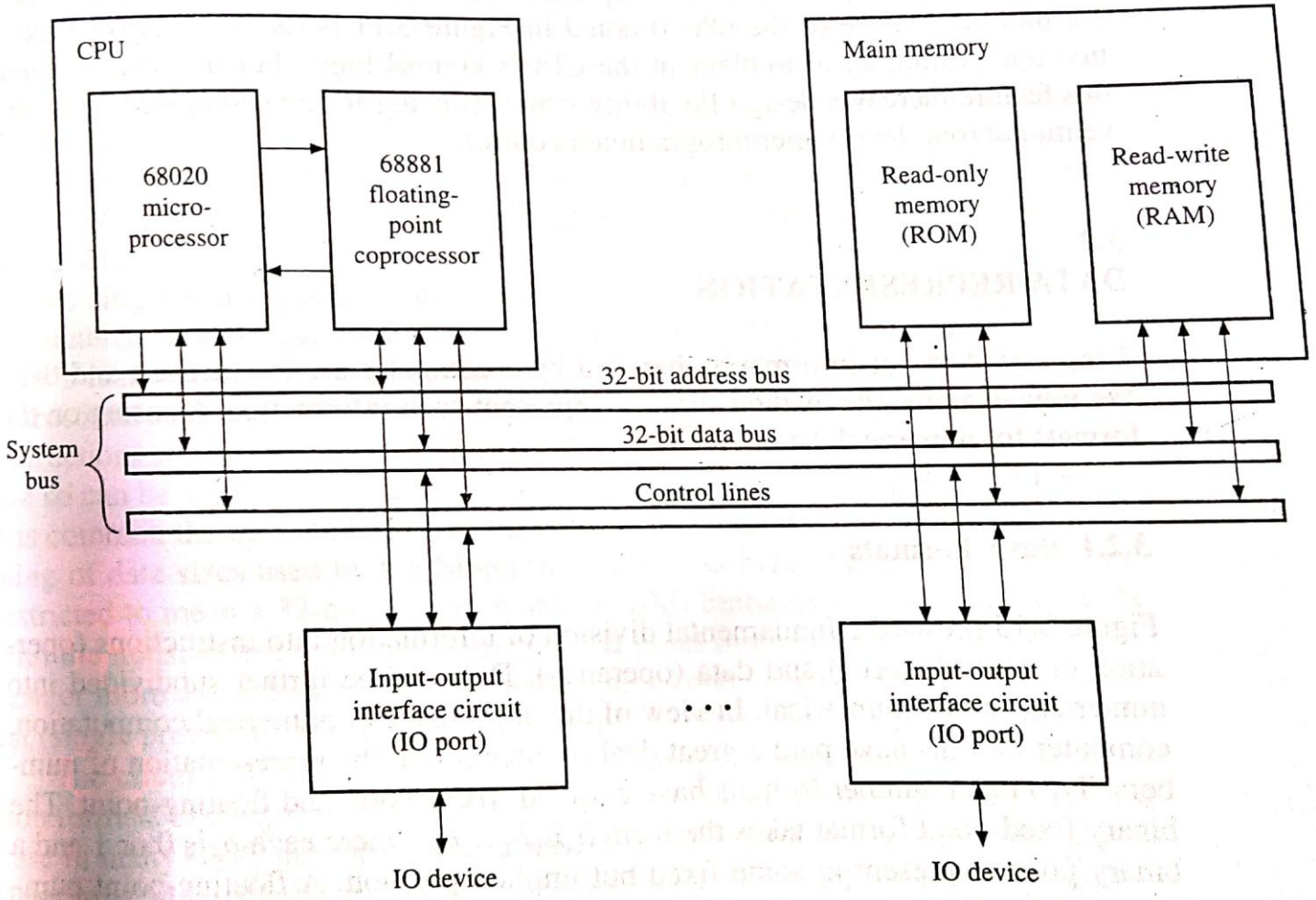


Figure 3.14
68020-based microcomputer with floating-point coprocessor.

PROCESSOR BASICS : CPU ORGANIZATION

Coprocessors:

- Coprocessor 68881 contains a set of eight 80-bit registers for storing floating point numbers of various formats, including 32 and 64-bit numbers conforming to the standard IEEE 754 format.
- Additional control registers allow it to communicate with the 68020.
- A set of coprocessor instructions are defined for the 68020; they contain command fields specifying floating point operations that the 68881 can execute.
- When the 68020 fetches and decodes such an instruction, it transfers the command portion to the coprocessor, which then executes it.
- Further exchanges take place between the main processor and the coprocessor until the coprocessor completes execution of its current operation, at which point the 68020 proceeds to its next instruction.
- Some operations performed by coprocessor are add, subtract, multiply, divide, square root, logarithms and trigonometric functions.

DATA REPRESENTATION : BASIC FORMATS

➢The fundamental division of information into instructions (operation or control words) and data (operands).

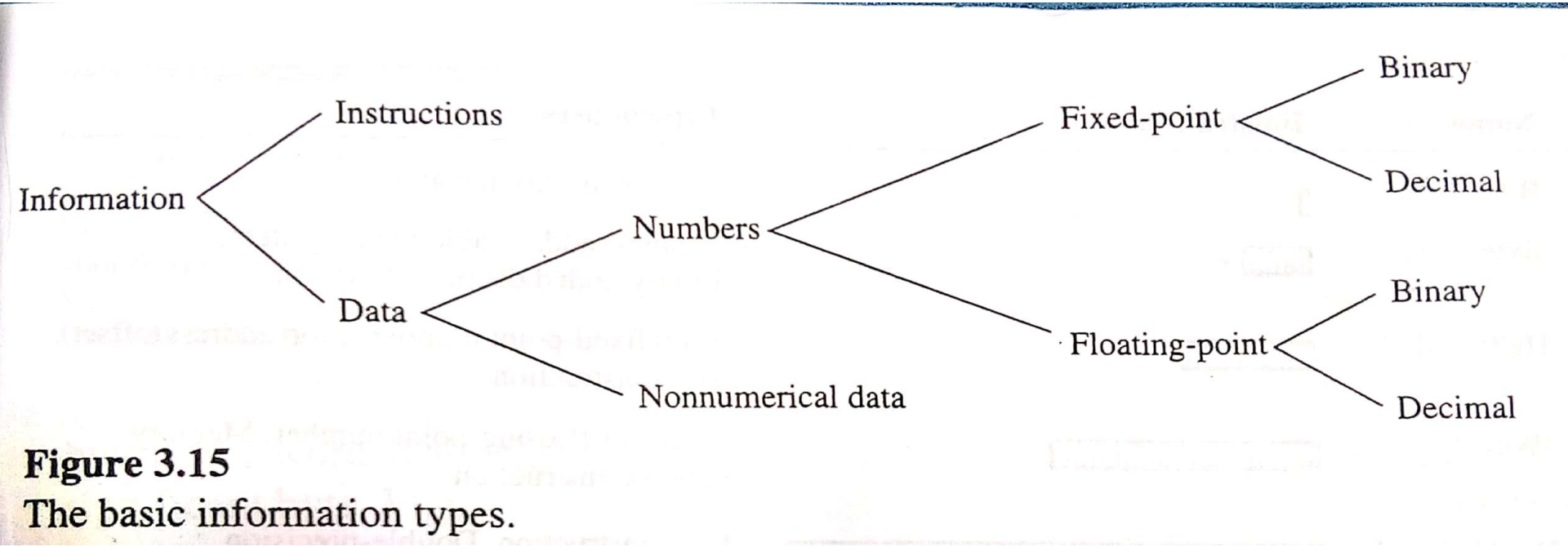


Figure 3.15
The basic information types.

➢ Non-numerical data usually take the form of variable length character strings encoded in one of several standard codes, such as **ASCII (American Standard Committee on Information Exchange) code**.

DATA REPRESENTATION : BASIC FORMATS

Word Length :

- Information – binary words
- Word – unit of information of some fixed length n.
- n-bit word allows up to 2^n different items to be represented.
- To encode alphanumeric symbols or character, 8-bit words called **bytes** are commonly used.
- Most computers have the 8-bit byte as the smallest addressable unit of information in their main memories.
- Word sizes is typically a multiple of 8, common CPU word sizes being 8, 16, 32 and 64 bits.
- Computers instruction set, we often find several different word sizes.
- Example : load and store instructions that reference memory need long address fields.

DATA REPRESENTATION : BASIC FORMATS

Word Length :

Bits	Name	Illustration	Typical uses
1	Bit		Status flag. Logic variable.
8	Byte		Smallest addressable memory item. Binary-coded decimal digit pair.
16	Halfword		Short fixed-point number. Short address (offset). Short instruction.
32	Word		Fixed- or floating-point number. Memory address. Instruction.
64	Double word		Long instruction. Double-precision floating-point number.

DATA REPRESENTATION : BASIC FORMATS

Word Length :

- Fixed-point numbers come in lengths of 1, 2, 4 or more bytes.
- Floating-point numbers also come in several lengths, the shortest (single precision) number being one word (32-bits) long.
- In CISC computers – instruction length varies – the program control unit must be designed to determine an instruction length from its opcode and to fetch a variable number of instruction bytes from memory.
- It must also increment program counter by a variable amount to obtain the address of the next consecutive instruction.

DATA REPRESENTATION : BASIC FORMATS

Word Length :

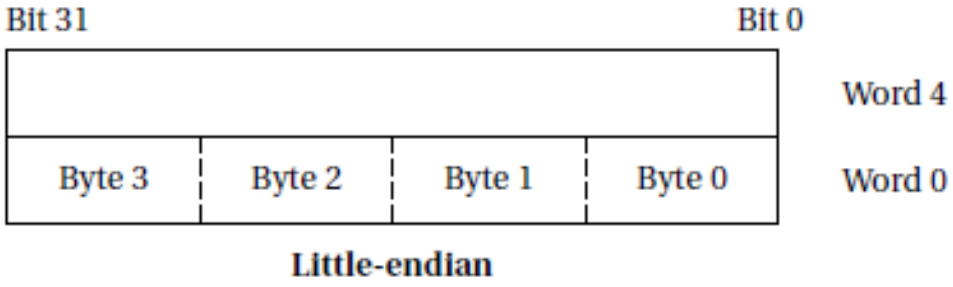
- To add two unsigned 64-bit binary integers in ARM 6 processor
- r1 , r0 holds the first operand
- r3, r2 holds the second operand
- r5, r4 holds the result

HDL format	ARM6 assembly-language format		Narrative format (comment)
C.R4 := R0 + R2	ADDS	R4,R0,R2	Add right words and store carry signal C.
R5 : = R1 + R3 + C	ADC	R5,R1,R3	Add left words plus C.

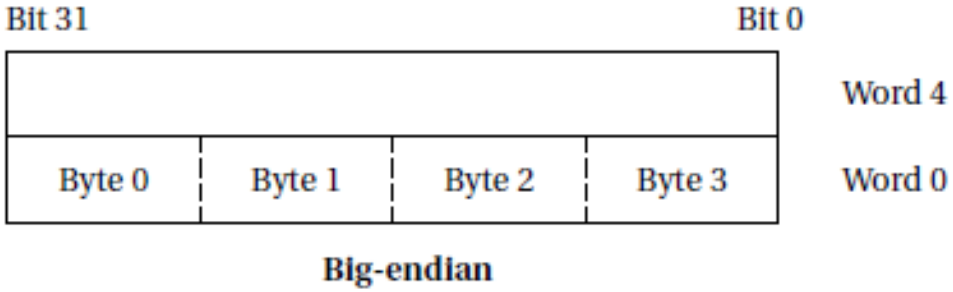
DATA REPRESENTATION : BASIC FORMATS

Storage order:

- Words are stored as individually addressable bytes in memory M, a question arises as to the storage order in M of the bytes within each word.
- The ARM processor can be configured at power-up to address the bytes in a word in either **little-endian mode** (the lowest order byte residing in the low-order bits of the word) or **big-endian mode** (the lowest-order byte stored in the highest bits of the word).



Little-endian



Big-endian

DATA REPRESENTATION : BASIC FORMATS

Storage order:

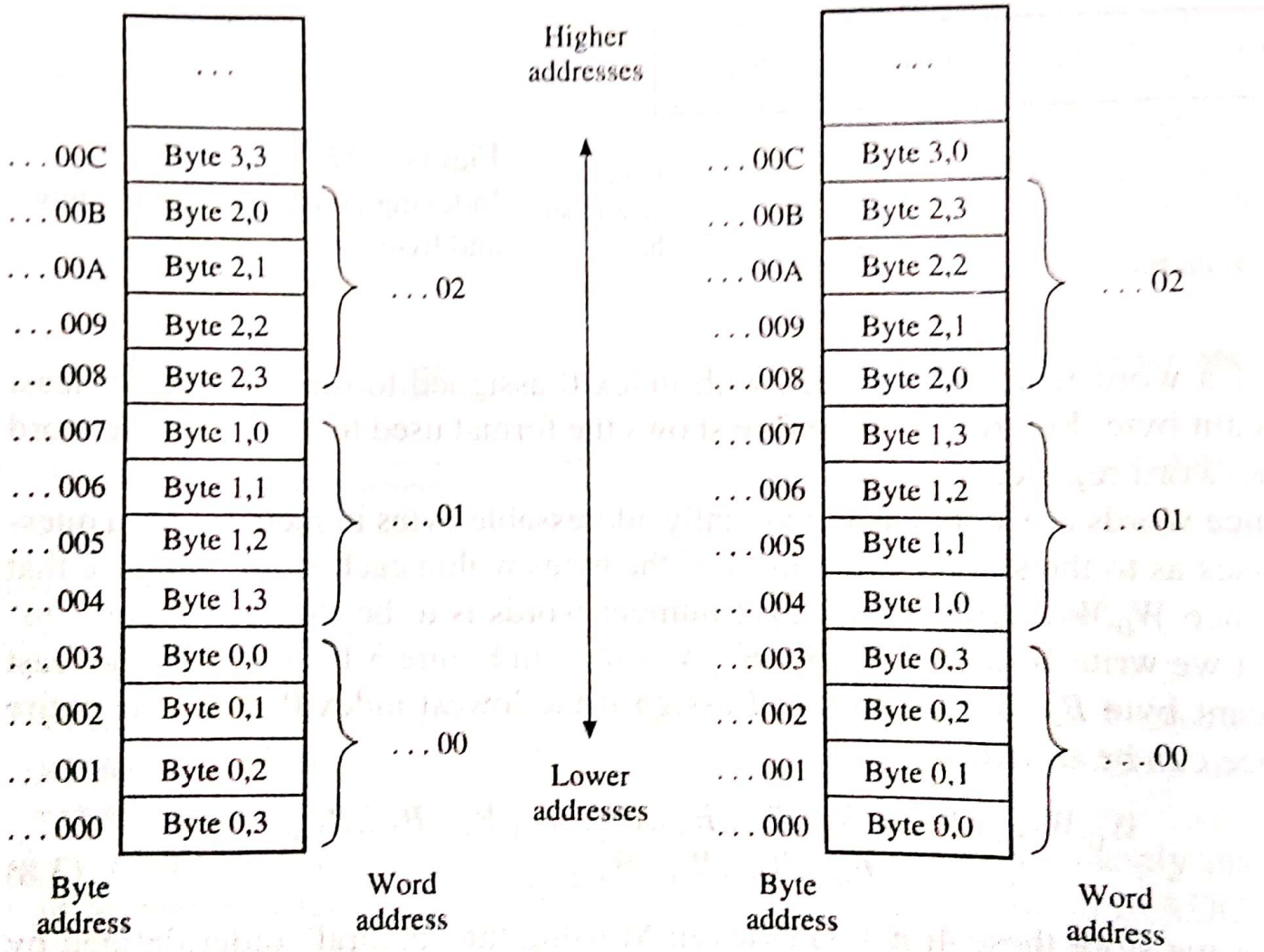


Figure 3.18
Basic byte storage methods: (*a*) big-endian and (*b*) little-endian.

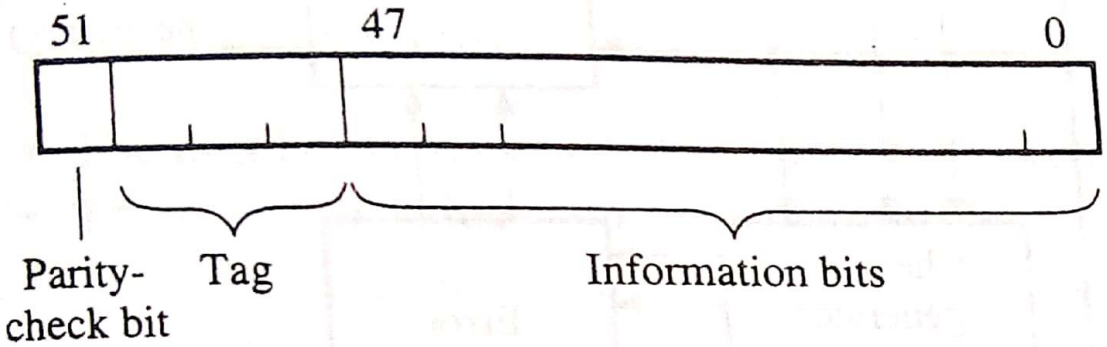
DATA REPRESENTATION : BASIC FORMATS

Tags:

- In the Von-Neumann computer, instruction and data words are stored together in main memory and are indistinguishable from one another – stored program concept.
- An item plucked at random from memory cannot be identified as an instruction or data.
- Different data types such as fixed-point and floating-point numbers also cannot be distinguished by inspection.
- A **Tag** (a group of bits) is associated with each information word.
- Tag inspection permits the hardware to check for software errors, such as an attempt to add operands whose types are incompatible.
- Tags have a serious cost disadvantage.
- Increase memory size and add to the system hardware costs without increasing computing performance.

DATA REPRESENTATION : BASIC FORMATS

Tags:



Tag	Interpretation
000	Single-precision number.
001	Indirect reference word.
010	Double-precision number.
011	Segment descriptor.
100	Step-index control word.
101	Data descriptor.
110	Uninitialized operand.
111	Instruction.

Figure 3.19
Tagged-word format of the Burroughs B6500/7500 series.

DATA REPRESENTATION : BASIC FORMATS

Error detection and correction:

- Manufacturing defects and environmental effects cause errors in computation.
- It happens when information is being transmitted between two relatively distant points within a computer or is being stored in a memory unit.
- Noise – corrupt a bit x that is being sent from A to B so that B receives x' instead of x.
- To guard against errors of this type, the information can be encoded so that special logic circuits can detect, and possibly even correct, the errors.
- A general way to detect or correct errors is to append special check bits to every work – **Parity bit**.

n-bit word - $X = (x_0, x_1, x_2, \ldots, x_{n-1})$

c0 parity bit is added with X - $X^* = (x_0, x_1, \ldots, x_{n-1}, c_0)$

- Bit c0 is assigned the value 0 or 1 that makes the number of ones in X* even – **even parity**, or odd, in the case of **odd-parity codes**.

DATA REPRESENTATION : BASIC FORMATS

Error detection and correction:

- In case of even parity, c_0 is defined by the logic equation,

$$c_0 = x_0 \oplus x_1 \oplus \ldots \oplus x_{n-1}$$

- In case of odd parity,

$$\overline{c_0} = x_0 \oplus x_1 \oplus \ldots \oplus x_{n-1}$$

- Suppose that the information X is to be transmitted from A to B.
- The value of c_0 is generated at the source point A and X* is sent to B.
- Let B receive the word X',

$$X' = (x_0', x_1', \ldots, x_{n-1}', c_0')$$

- B then determines the parity of the received word by recomputing the parity bit,

$$c_0^* = x_0' \oplus x_1' \oplus \ldots \oplus x_{n-1}'$$

- The received parity bit c'_0 and the reconstituted parity bit c^*_0 are then compared.

DATA REPRESENTATION : BASIC FORMATS

Error detection and correction:

- If $c_0^* \neq c_0^{'}$, the received information contains an error (single bit error).
- This fails in identifying the multiple bit errors or double error (0 changes to 1 and 1 changes to 0).
- It does not detect all multiple errors, much less provide any information about the location of the erroneous bits.
- The parity checking concept can be extended to the detection of multiple errors or to the location of single or multiple errors.
- It is achieved by providing additional parity bits, each of which checks the parity of some subset of the bits in the word X*.
- By appropriately overlapping these subsets, the correctness of every bit can be determined.

DATA REPRESENTATION : BASIC FORMATS

Error detection and correction:

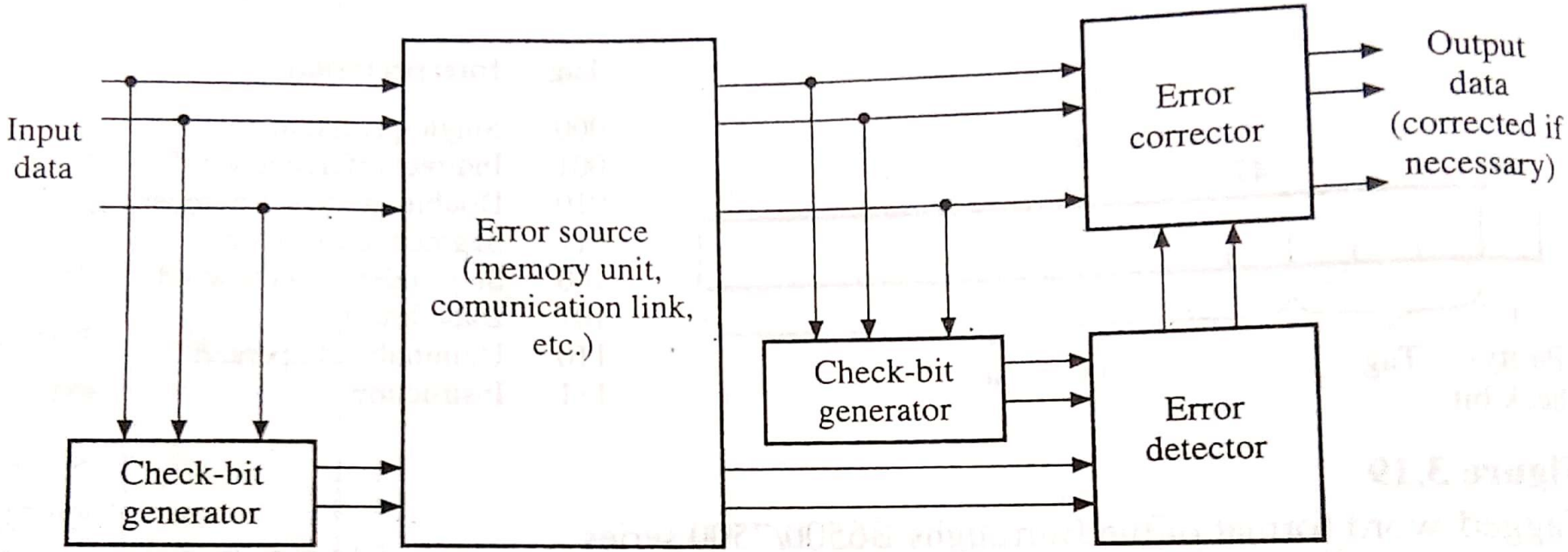


Figure 3.20
Error detection and correction logic.

DATA REPRESENTATION : FIXED POINT NUMBERS

➢The following factors needs to be considered while selecting a number representation to be used in a computer,

1. The number types to be represented; for example, integers or real numbers.
2. The range of values (number magnitudes) likely to be encountered
3. The precision of the numbers, which refers to the maximum accuracy of the representation.
4. The cost of the hardware required to store and process the numbers.

➢ The two principal number formats are **fixed-point** and **floating-point.**

➢ Fixed-point formats allow a limited range of values and have relatively simple hardware requirements.

➢ Floating-point numbers allow a much larger range of values but require either costly processing hardware or lengthy software implementations.

DATA REPRESENTATION : FIXED POINT NUMBERS

Binary numbers:

- The fixed-point format is derived directly from the ordinary (decimal) representation of a number as a sequence of digits separated by a decimal point.
- The digit to the left of the decimal point represents an **integer**; the digit to the right represent a **fraction**.
- This is **positional notation** in which each digit has a fixed weight according to its position relative to the decimal point.
- Example : 192.73 is equivalent to,

$$1x10^{2}+9x10^{1}+2x10^{0}+7x10^{-1}+3x10^{-2}$$

- Assign weights of the form r^{i}, where **r** is the base or radix of the number system, to each digit.

DATA REPRESENTATION : FIXED POINT NUMBERS

Binary numbers:

- The most fundamental number representation used in computers employs a base-two positional notation.
- A binary word of the form,

$$b_N \ldots b_3 b_2 b_1 b_0 . b_{-1} b_{-2} b_{-3} \ldots b_M$$

represents the number,

$$\sum_{i=M}^{N} b_i 2^i$$

- Binary 10_2 denotes Decimal 2_{10}
- Suppose an n-bit word is to contain a signed binary number.
- One bit is reserved to represent the sign of the number, while the remaining bits indicate its magnitude.
- 0 or 1 in the MSB position to indicate a number as plus or minus.

DATA REPRESENTATION : FIXED POINT NUMBERS

Binary numbers:

- It leads to the format,

$$x_{n-1}x_{n-2}x_{n-3}\ldots x_2x_1x_0$$

where $\mathbf{x_{n-1}}$ is he sign bit and $\mathbf{x_{n-2}x_{n-3}\ldots x_2x_1x_0}$ is the magnitude

- Suppose that both positive and negative binary numbers are to be represented by an n-bit word $X=x_{n-1}x_{n-2}x_{n-3}\ldots x_2x_1x_0$.
- The standard format for positive numbers is given by with a sign bit of 0 on the left and the magnitude to the right in the usual positional notation.
- A natural way to represent negative numbers is to employ the same positional notation for the magnitude and simply change the sign bit x_{n-1} to 1 to indicate minus.
- This number code is called **sign magnitude**.
- Operations like subtraction are costly to implement by logic circuits when sign-magnitude codes are used.

DATA REPRESENTATION : FIXED POINT NUMBERS

Binary numbers:

➢Several number codes have been devised that use the same representation for positive numbers as the sign-magnitude code but represent negative numbers in different ways. Example : 1's complement code and 2's complement code.

➢ The primary advantage of the complement codes is that subtraction can be performed by logical complementation and addition only.

DATA REPRESENTATION : FIXED POINT NUMBERS

Binary numbers:

Decimal representation	Binary code		
	Sign magnitude	Ones complement	Twos 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	0000
–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

Figure 3.21
Comparison of three 4-bit codes for signed binary numbers.

DATA REPRESENTATION : FIXED POINT NUMBERS

Exceptional conditions:

- If the result of an arithmetic operation involving n-bit number is too large (small) to be represented by n-bits, **overflow (underflow)** is said to occur.
- It is generally necessary to detect overflow and underflow, since they may indicate bad data or a programming error.
- **Round-off error** – results from the fact that every number must be represented by a limited number of bits.
- An operation involving n-bit numbers frequently produces a result of more than n bits. Retaining the n most significant bits of the result without modification is called **truncation**. This error can be reduced by a process called rounding.

DATA REPRESENTATION : FIXED POINT NUMBERS

Decimal numbers:

- Numbers being entered into a computer must first be converted from decimal to some binary representation.
- Similarly, binary to decimal conversion is a normal part of the computer's output process.
- Conversion should be rapid.
- Several number codes exists that facilitate rapid binary-decimal conversion by encoding each decimal digit separately by a sequence of bits – **Decimal codes**
- Example : Binary coded decimal (BCD), ASCII code, Excess-three code, Two-out-of-five code

DATA REPRESENTATION : FIXED POINT NUMBERS

Decimal numbers:

Decimal digit	Decimal code			
	BCD	ASCII	Excess-three	Two-out-of-five
0	0000	0011 0000	0011	11000
1	0001	0011 0001	0100	00011
2	0010	0011 0010	0101	00101
3	0011	0011 0011	0110	00110
4	0100	0011 0100	0111	01001
5	0101	0011 0101	1000	01010
6	0110	0011 0110	1001	01100
7	0111	0011 0111	1010	10001
8	1000	0011 1000	1011	10010
9	1001	0011 1001	1100	10100

Figure 3.23
Some important decimal number codes.

DATA REPRESENTATION : FIXED POINT NUMBERS

Decimal numbers:

- In **Two-out-of-five code** – each decimal digit is represented by a 5-bit sequence containing two 1's and three 0's; there are exactly 10 distinct sequences of this type.
- It is capable of single error detecting, since changing any one bit results in a sequence that does not correspond to a valid code word.
- Its drawbacks are that it is a non-weighted code and uses 5 rather than 4 bits per decimal digit.
- Decimal codes have two disadvantages :

1. They use more bits to represent a number than the binary codes.
2. The circuitry required to perform arithmetic using decimal operands is more complex than that needed for binary arithmetic.

DATA REPRESENTATION : FIXED POINT NUMBERS

Hexadecimal numbers:

- Hexadecimal numbers are characterized by a base r=16 and the use of 16 digits, consisting of the decimal digits 0,1,2,…,9 augmented by the six digits A,B,C,D,E and F, which have the numerical values 10, 11, 12, 13, 14 and 15, respectively.
- Hexadecimal code is useful for representing long binary numbers, a consequently of the fact that the base 16 is a power of two.
- Example : $2FA0C_{16} = 195084_{10} = 00101111101000001100_2$
- Hex code provides a very convenient shorthand for binary information.

DATA REPRESENTATION : FLOATING POINT NUMBERS

- Scientific notation permits us to represent large numbers using relatively few digits.
- Example : Quintillion – 1.0×10^{18}
- The floating point codes used in computers are binary (or binary-coded) versions.
- Three numbers are associated with a floating-point number : a **mantissa M**, an **exponent E** and a **base B**.
- The mantissa M is also referred to as the **significand** or **fraction** in the literature.
- These three components together represent the real number $M \times B^E$
- For machine implementation the mantissa and exponent are encoded as fixed-point numbers with a base r that is usually 2 or 10.
- Since B is a constant, it need not be included in the number code.
- A floating-point number is therefore stored as a word (M,E) consisting of a pair of signed fixed-point numbers : a mantissa M which is usually a fraction or an integer, and an exponent E, which is an integer.

DATA REPRESENTATION : FLOATING POINT NUMBERS

- The number of digits in M determines the precision of (M, E); B and E determines its range.
- With a word size of n bits, 2^n is the most real numbers that (M,E) can represent.
- Example : Suppose that M and E are both 3-bit, sign-magnitude integers and B=2.
- Then M and E can assume the values ±0, ±1, ±2 and ±3.
- All the binary words of the form (M, E) = (x00, xxx) represent zero, where x denotes either 0 or 1.
- The smallest non-zero positive number is (001, 111) = $1 \times 2^{-3} = 0.125$
- (101, 111) = $-1 \times 2^{-3} = -0.125$
- The largest representable positive number is (011, 011) = $3 \times 2^3 = 24$
- (111, 011) = $-3 \times 2^3 = -24$
- The floating point representation of most real numbers is only approximate.
- The results of floating-point operations must be carefully rounded off to minimize the errors inherent in floating-point representation.

DATA REPRESENTATION : FLOATING POINT NUMBERS

➢It is common practice for floating point processing circuits to include a few extra mantissa digits termed **guard digits** to reduce approximation errors; the guard digits are removed automatically from the final result.

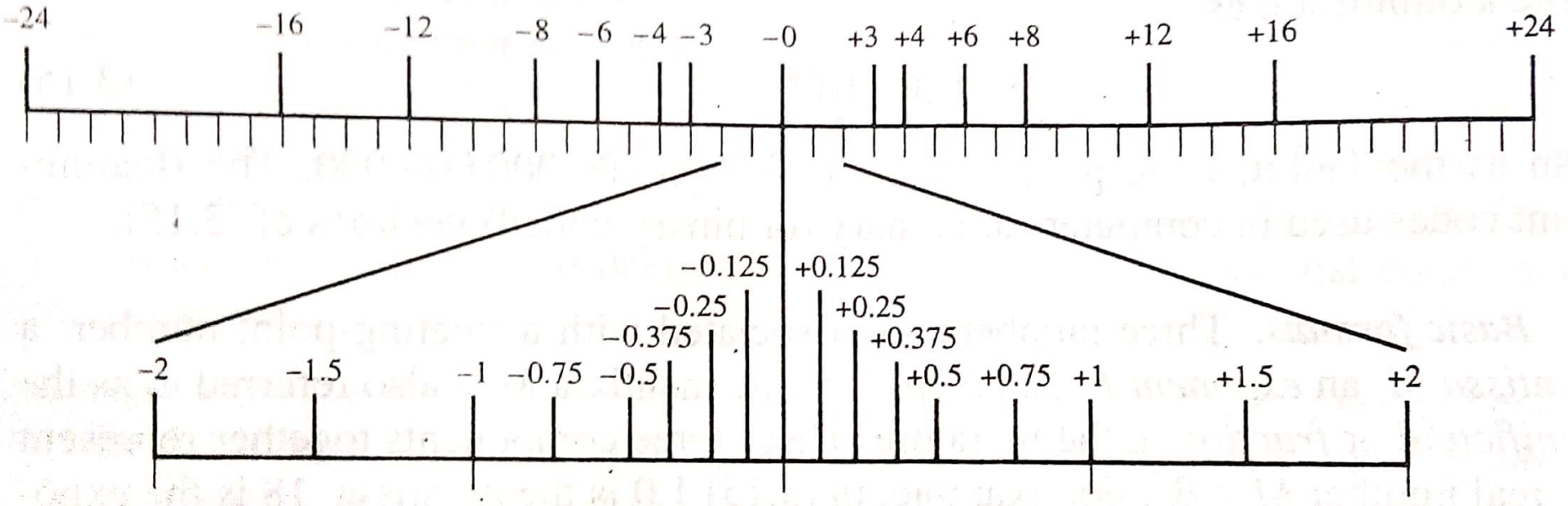


Figure 3.24
The real numbers representable by a hypothetical 6-bit, floating-point format.

DATA REPRESENTATION : FLOATING POINT NUMBERS

Normalization and Biasing :

- Floating point representation is redundant (same number can be represented in more than one way).
- Quintillion = 1.0×10^{18}, 0.1×10^{19}, 1000000×10^{12} and 0.000001×10^{24}
- It is generally desirable to have a unique or normal form for each representable number in a floating point system.
- Consider the common case where the mantissa is a sign magnitude fraction and a base r is used.
- The mantissa is said to be normalized if the digit to the right of the radix point is not zero, that is appear in the magnitude part of the number.
- Example : 0.1×10^{19} is the normalized format

DATA REPRESENTATION : FLOATING POINT NUMBERS

Normalization and Biasing :

- Floating-point exponents should be encoded in excess-K code.
- The exponent field E contains an integer that is the desired exponent value plus K.
- The quantity K is called the **bias** and an exponent encoded in this way is called a **biased exponent** or **characteristics**.

Exponent bit pattern *E*	Unsigned value	Number represented	
		Bias = 127	Bias = 128
111 . . . 11	255	+128	+127
111 . . . 10	254	+127	+126
. . .		. . .	. . .
100 . . . 01	129	+2	+1
100 . . . 00	128	+1	0
011 . . . 11	127	0	–1
011 . . . 10	126	–1	–2
. . .		. . .	. . .
000 . . . 01	1	–126	–127
000 . . . 00	0	–127	–128

Figure 3.25
Eight-bit biased exponents with bias = 127 (excess-127 code) and bias = 128 (excess-128 code).

DATA REPRESENTATION : FLOATING POINT NUMBERS

Standards :

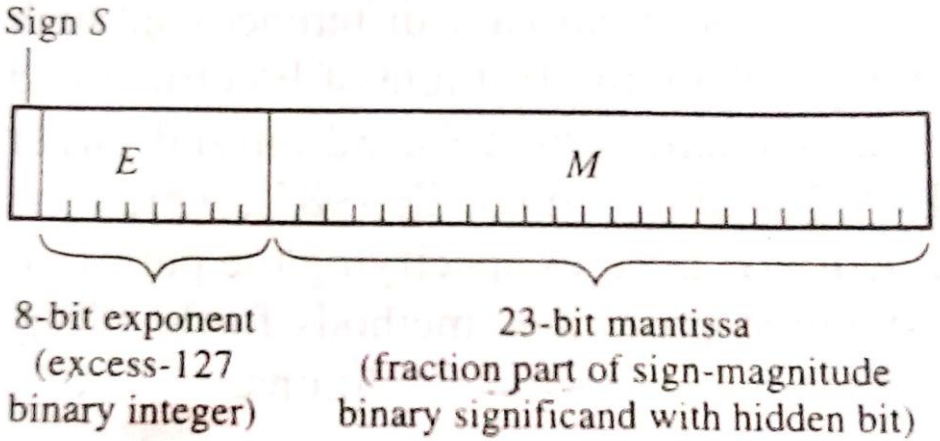
- The Institute of Electrical and Electronics Engineers (IEEE) sponsored a standard format for 32-bit and larger floating point numbers, known as **the IEEE 754 standards**.
- Besides specifying the permissible formats for M, E and B, the IEEE standard prescribes methods for handling round-off errors, overflow, underflow and other exceptional conditions.
- A 32-bit floating-point number conforming to the IEEE 754 standard represents the real number N given by the formula,

$$N = (-1)^s \, 2^{E-127} \, (1.M)$$

Example : N = -1.5 is represented by,

1 01111111 10000000000000000000000

S = 1, E = 127, M = 0.5



INSTRUCTION SETS

Instruction Formats :

- The purpose of an instruction is to specify both an operation to be carried out by a CPU or other processor and the set of operands or data to be used in the operation.
- Register transfer operation of the form $X_1 := op(X_1, X_2,\ldots,X_n)$ - which applies the operation **op** to **n** operands.
- Assembly language notation – op $X_1,X_2,\ldots,X_n$
- The operation **op** is specified by the field called the **opcode (operation code).**
- The n $X_1,X_2,\ldots,X_n$ fields are referred to as **addresses**.
- An address X_i typically names a register or a memory location that stores an operand value.
- In some instances X_i itself is the desired value, in which case it is called an immediate address.

INSTRUCTION SETS

Instruction Formats :

- If m is the normal maximum number of explicit main-memory addresses allowed in any processor instruction, the processor is called an **m-address machine**.
- CISC – Motorola 680X0 – More addressing modes and many different instruction formats.
- Instruction length varies from 2 to 10 bytes.
- Add register instruction,

  ADD.L D1,D2 – register to register addition of 32-bit (long word) operands

  D2 := D2 + D1 – fits in the third 2-byte format F3

- Variant: ADD.L ADR1,D2 - combine load and add operation

  D2 := D2 +M(ADR1)

INSTRUCTION SETS

Instruction Formats :

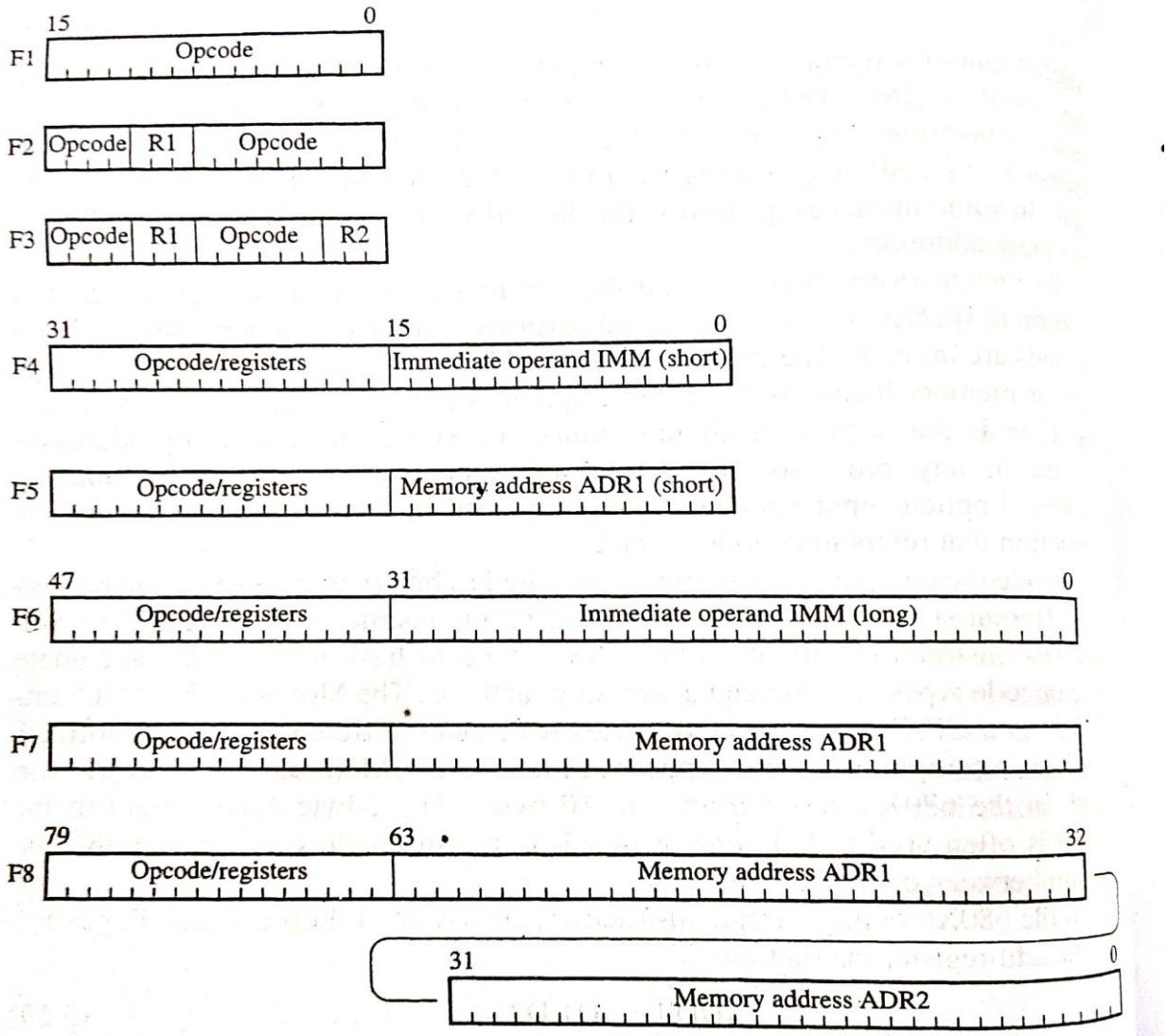


Figure 3.27
A selection of instruction formats of the Motorola 680X0.

INSTRUCTION SETS

Instruction Formats : RISC formats

- Fast, single cycle execution.
- Instructions of fixed length.
- Memory addressing is restricted to load and store instructions, so the operands of most instructions are register addresses.
- Register to register operation – Rd := F(Rs, S2)

Where Rd – Destination Register, Rs – first source register and S2 – Second source register.

- If bit 13 of the instruction is set to one, then S2 is interpreted as an immediate address, that is, as a 13-bit constant.

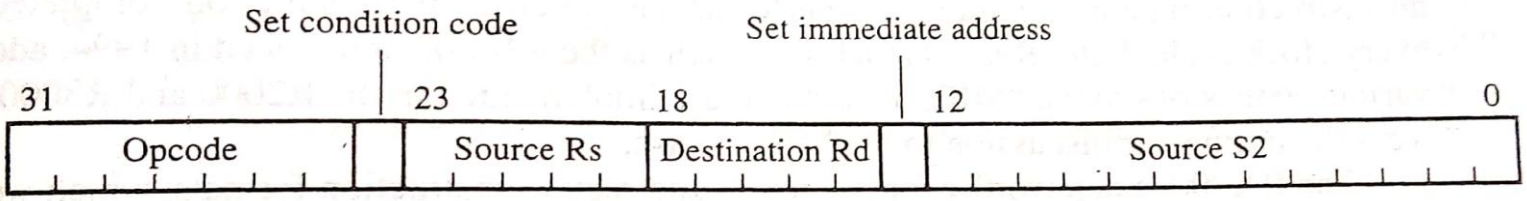


Figure 3.28
Instruction format of the Berkeley RISC 1.

INSTRUCTION SETS

Operand extension :

- A CPU is designed primarily to process data words and addresses of one specific length – a 32-bit word in the case of ARM6 – although some instructions handle longer or shorter operands.
- Instructions often contain operand fields that are shorter than the standard word sizes.
- This problem is unavoidable in RISC instruction sets where the instruction length and the standard word sizes are the same.
- Consequently a systematic method is needed to extend short operand values to full-size, signed or unsigned numbers.
- When a short m-bit, two's complement number is used in an n-bit arithmetic operation where n>m, a technique called **sign-extension** is employed.

INSTRUCTION SETS

Operand extension :

- 13- bit operand (-2646) – N= 10101 01010101 gets converted to a 32-bit word as,

  $N_{sign\text{-}extended}$ = 11111111 11111111 11110101 01010101

  S = 1, n-m = 19.

- Sign extension maintains a number's correct sign and magnitude because it introduces only numerically insignificant leading 0's (positive numbers) or insignificant leading 1's (negative numbers).
- If N is to be treated as an unsigned binary number, then it is always extended by leading 0's, independent of the value of s – **zero extension**.

INSTRUCTION SETS

Operand extension :

- How is an n-bit memory address, which is a long (typically 32-bit) unsigned integer, constructed from a short m-bit address field, when n>m?

- To treat a short memory address as a **modifier**, or **offset**, which is added (in zero-extended form) to a full –length memory address stored in a designated CPU register, called a **base register**.

- The final memory address obtained is an **effective address**.

INSTRUCTION SETS

Addressing Modes :

- The purpose of an address field is to point to the current value V(X) of some operand X used by an instruction. This value can be specified in various ways, which are termed as **addressing modes**.
- The addressing mode of X affects the following issues :

1. The speed with which V(X) can be accessed by the CPU.
2. The ease with which V(X) can be specified and altered.

- Operand values located in CPU registers and program counter PC can be accesses faster than operands in M.

INSTRUCTION SETS

Addressing Modes :

- If the value V(X) of the target operand is contained in the address field itself, then X (constant) is called an immediate operand and the corresponding addressing mode is **immediate addressing**.
- If X is a variable, and its value V(X) can be varied without modifying the instruction address field. Operand specification of this type is called **direct addressing**.
- If the operands are stored in a register – **Register addressing mode**.
- It is sometimes useful to change the location (as opposed to the value) of X without changing the address fields of any instructions that refer to X.
- Instruction contains the address W of a storage location, which in turn contains the address X of the desired operand value V(X) – **Indirect addressing mode**.

INSTRUCTION SETS

Addressing Modes :

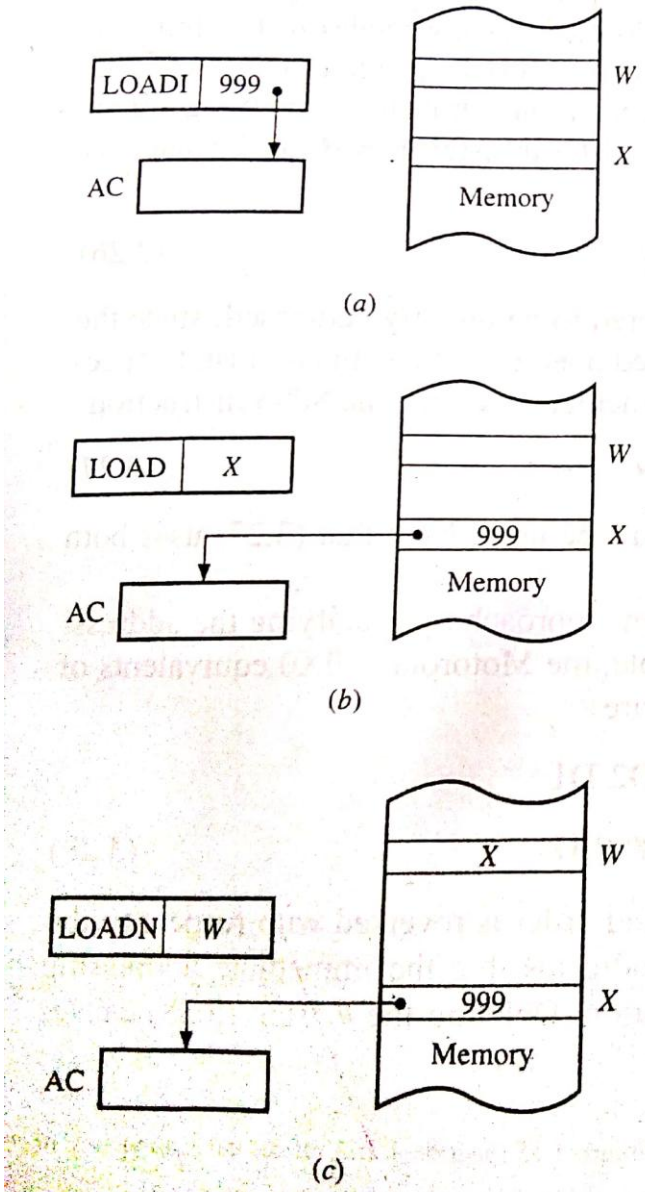


Figure 3.30
Three basic addressing modes: (*a*) immediate; (*b*) direct; (*c*) indirect.

INSTRUCTION SETS

Addressing Modes :

- **Absolute addressing** – requires the complete operand address to appear in the instruction operand field. This address is used without modification (except, zero or sign-extension in the case of short address fields).
- **Relative addressing** – Operand field in an instruction contains the relative address, also called an **offset** or displacement D. The instruction also implicitly or explicitly identifies other storage locations R_1, R_2,…, R_k (usually CPU registers) containing additional addressing information.
- The effective address A of an operand is then some function f(D, R_1, R_2,…,R_k).

$$A := R+D$$

R is referred as **base register** and its **content as a base address**.

INSTRUCTION SETS

Addressing Modes :

➢Advantages in using relative addressing,

1. Since all the address information need not be included in the instructions, instruction length is reduced.
2. By changing the content of R, the processor can change the absolute addresses referred to by a block of instructions B.
3. R can be used for storing indexes to facilitate the processing of indexed data (datas stored in consecutive memory addresses)

➢ Drawback of relative addressing : Requirement of extra logic circuits and processing time needed to compute address.

INSTRUCTION SETS

Addressing Modes :

➢To facilitate stepping through a sequence of items in a memory, addressing modes that automatically increment or decrement an address can be defined; the resulting address-modification process is called **autoindexing**.

➢ **Register indirect addressing** – indirect addressing with a register name R in the address field. It is often used to access memory, in which case R becomes a memory address register.

Example: MOVE.B (A0), D1 – parenthesis to indicate that (A0) is an indirect address involving A0 address register. – D1[7:0] := M(A0)

➢ **Register indirect with offset** – type of base addressing or indexed addressing.

Example: Store word instruction – SW Rt, OFFSET(Rs) = M(Rs+OFFSET) := Rt

INSTRUCTION SETS

Number of Addresses:

- **Zero address machine** : most instructions contain no address.

Example : ADD

X+Y, where X and Y and two memory locations in a stack.

- **One address** add instruction,

Example: ADD X - AC := AC+X

- **Two address** add instruction,

Example: ADD X,Y - AC := X+Y

- **Three address** add instruction,

Example: ADD Z, X, Y - Z := X+Y

INSTRUCTION SETS

Number of Addresses:

Instruction	Comments
LOAD A	Transfer A to accumulator AC.
MULTIPLY B	AC := AC × B
STORE T	Transfer AC to memory location T.
LOAD C	Transfer C to accumulator AC.
MULTIPLY C	AC := AC × C
ADD T	AC := AC + T
STORE X	Transfer result to memory location X.

(*a*) One-address machine

INSTRUCTION SETS

Number of Addresses:

Instruction	Comments
MOVE T,A	T := A
MULTIPLY T,B	T := T × B
MOVE X,C	X := C
MULTIPLY X,C	X := X × C
ADD X,T	X := X + T

(*b*) Two-address machine

Instruction	Comments
MULTIPLY T,A,B	T := A × B
MULTIPLY X,C,C	X := C × C
ADD X,X,T	X := X + T

(*c*) Three-address machine

Figure 3.32
Programs to execute the operation $X := A \times B + C \times C$ in one-address, two-address, and three-address processors.

INSTRUCTION SETS

Number of Addresses:

Instruction	Comments
PUSH A	Transfer A to top of stack.
PUSH B	Transfer B to top of stack.
MULTIPLY	Remove A, B from stack and replace by $A \times B$.
PUSH C	Transfer C to top of stack.
PUSH C	Transfer second copy of C to top of stack.
MULTIPLY	Remove C, C from stack and replace by $C \times C$.
ADD	Remove $C \times C$, $A \times B$ from stack and replace by their sum.
POP X	Transfer result from top of stack to X.

Figure 3.33
Program to execute $X := A \times B + C \times C$ in a zero-address, stack processor.

INSTRUCTION SETS

Completeness:

- Instructions are conveniently divide into the following five types:

1. **Data-transfer instructions** – which copy information from one location to another either in the processor internal register set or in the external main memory.
2. **Arithmetic instructions** – which perform operations on numerical data.
3. **Logical instructions** – which include Boolean and other nonnumerical operation.
4. **Program-control instructions** – Branch instructions – which change the sequence in which programs are executed.
5. **Input-output (IO) instructions** – which cause information to be transferred between the processor or its main memory and external IO devices.

INSTRUCTION SETS

Completeness:

Type	Operation name(s)	Description
Data transfer	MOVE	Copy word or block from source to destination.
	LOAD	Copy word from memory to processor register.
	STORE	Copy word from processor register to memory.
	SWAP (EXCHANGE)	Swap contents of source and destination.
	CLEAR	Transfer word of 0s to destination.
	SET	Transfer word of 1s to destination.
	PUSH	Transfer word from source to top of stack.
	POP	Transfer word from top of stack to destination.

INSTRUCTION SETS

Completeness:

Arithmetic	ADD	Compute sum of two operands.
	ADD WITH CARRY	Compute sum of two operands and a carry bit.
	SUBTRACT	Compute difference of two operands.
	MULTIPLY	Compute product of two operands.
	DIVIDE	Compute quotient (and remainder) of two operands.
	MULITPLY AND ADD	Compute product of two operands; add it to a third operand.
	ABSOLUTE	Replace operand by its absolute value.
	NEGATE	Change sign of operand.
	INCREMENT	Add 1 to operand.
	DECREMENT	Subtract 1 from operand.
	ARITHMETIC SHIFT	Shift operand left (right) with sign extension.

INSTRUCTION SETS

Completeness:

Logical	AND OR NOT EXCLUSIVE-OR	Perform the specified logical operation bitwise.
	LOGICAL SHIFT	Shift operand left (right) introducing 0s at end.
	ROTATE	Left- (right-) shift operand around closed path.
	CONVERT (EDIT)	Change data format, for example, from binary to decimal.

INSTRUCTION SETS

Completeness:

Program control	JUMP (BRANCH)	Unconditional transfer; load PC with specified address.
	JUMP CONDITIONAL	Test specified conditions; if true, load PC with specified address.
	JUMP TO SUBROUTINE (BRANCH-AND-LINK)	Place current program control information including PC in known location, for example, top of stack; jump to specified address.
	RETURN	Restore current program control information including PC from known location, for example, from top of stack.
	EXECUTE	Fetch operand from specified location and execute as instruction; note that PC is not modified.
	SKIP CONDITIONAL	Test specified condition; if true, increment PC to skip next instruction.
	TRAP (SOFTWARE INTERRUPT)	Enter supervisor mode.
	TEST	Test specified condition; set flag(s) based on outcome.
	COMPARE	Make logical or arithmetic comparison of two or more operands; set flag(s) based on outcome.

Figure 3.34
List of common instruction types.

INSTRUCTION SETS

Completeness:

Type	Operation name(s)	Description
Program control	SET CONTROL VARIABLES	Large class of instructions to set controls for protection purposes, interrupt handling, timer control, and so forth (often privileged).
	WAIT (HOLD)	Stop program execution; test a specified condition continuously; when the condition is satisified, resume instruction execution.
	NO OPERATION	No operation is performed, but program execution continues.
Input-output	INPUT (READ)	Copy data from specified IO port to destination, for example, output contents of a memory location or processor register.
	OUTPUT (WRITE)	Copy data from specified source to IO port.
	START IO	Transfer instuctions to IOP to initiate an IO operation.
	TEST IO	Transfer status information from IO system to specified destination.
	HALT IO	Transfer instructions to IOP to terminate an IO operation.

Figure 3.34
(continued).

INSTRUCTION SETS

Completeness:

➢ Advantages of RISC processor are:

1. Relatively few instruction types and addressing modes.
2. Fixed and easily decoded instruction formats
3. Fast, single-cycle instruction execution
4. Hardwired rather than microprogrammed control
5. Memory access limited mainly to load and store instructions
6. Use of compilers to optimize object code performance.

SUMMARY

- Evolution of Computers – Mechanical Era and Electronics Era
- VLSI Era
- Different levels of Abstraction : Register Level, Processor Level
- CPU Organization
- Number Representation – fixed-point and floating-point numbers
- Instruction set – Instruction format, Types and Addressing Modes.