DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE MATERIALS



CS 202 COMPUTER ORGANIZATION AND ARCHITECTURE

VISION OF THE INSTITUTION

To mould true citizens who are millennium leaders and catalysts of change through excellence in education.

MISSION OF THE INSTITUTION

NCERC is committed to transform itself into a center of excellence in Learning and Research in Engineering and Frontier Technology and to impart quality education to mould technically competent citizens with moral integrity, social commitment and ethical values.

We intend to facilitate our students to assimilate the latest technological know-how and to imbibe discipline, culture and spiritually, and to mould them in to technological giants, dedicated research scientists and intellectual leaders of the country who can spread the beams of light and happiness among the poor and the underprivileged.

ABOUT DEPARTMENT

- Established in: 2002
- Course offered : B.Tech in Computer Science and Engineering

  M.Tech in Computer Science and Engineering

  M.Tech in Cyber Security
- Approved by AICTE New Delhi and Accredited by NAAC
- Affiliated to the University of A P J Abdul Kalam Technological University.

DEPARTMENT VISION

Producing Highly Competent, Innovative and Ethical Computer Science and Engineering Professionals to facilitate continuous technological advancement.

DEPARTMENT MISSION

1. To Impart Quality Education by creative Teaching Learning Process
2. To Promote cutting-edge Research and Development Process to solve real world problems with emerging technologies.
3. To Inculcate Entrepreneurship Skills among Students.
4. To cultivate Moral and Ethical Values in their Profession.

PROGRAMME EDUCATIONAL OBJECTIVES

PEO1: Graduates will be able to Work and Contribute in the domains of Computer Science and Engineering through lifelong learning.

PEO2: Graduates will be able to Analyse, design and development of novel Software Packages, Web Services, System Tools and Components as per needs and specifications.

PEO3: Graduates will be able to demonstrate their ability to adapt to a rapidly changing environment by learning and applying new technologies.

PEO4: Graduates will be able to adopt ethical attitudes, exhibit effective communication skills, Teamworkand leadership qualities.

PROGRAM OUTCOMES (POS)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSO)

PSO1: Ability to Formulate and Simulate Innovative Ideas to provide software solutions for Real-time Problems and to investigate for its future scope.

PSO2: Ability to learn and apply various methodologies for facilitating development of high quality System Software Tools and Efficient Web Design Models with a focus on performance optimization.

PSO3: Ability to inculcate the Knowledge for developing Codes and integrating hardware/software products in the domains of Big Data Analytics, Web Applications and Mobile Apps to create innovative career path and for the socially relevant issues.

COURSE OUTCOMES

CO1	To identify the basic structure and functional units of a digital computer. And analyze the effect of addressing modes on the execution time of a program
CO2	To design processing unit using the concepts of ALU and control logic design.
CO3	To select appropriate interfacing standards for I/O devices.
CO4	To identify the pros and cons of different types of Memory systems and understand mapping functions.
CO5	To select appropriate interfacing standards for I/O devices.
CO6	To identify the roles of various functional units of a computer in instruction execution. And analyze the types of control logic design in processors.

MAPPING OF COURSE OUTCOMES WITH PROGRAM OUTCOMES

CO'S	**PO 1**	**PO 2**	**PO 3**	**PO 4**	**PO 5**	**PO 6**	**PO 7**	**PO 8**	**PO 9**	**PO 10**	**PO 11**	**PO 12**
CO1	3	2	2	-		-	-	-	-	-	-	2
CO2	3	3	-	-	2	-	-	-	-	-	-	2
CO3	3	2	-	-	2	-	-	-	-	-	-	2
CO4	3	2	2	-		-	-	-	-	-	-	2
CO5	3	3	3	-	-	-	-	-	-	-	-	-
CO6	3	2	-	-	2	-	-	-	-	-	-	2

MAPPING OF COURSE OUTCOMES WITH PROGRAM SPECIFICOUTCOMES

CO'S	PSO1	PSO2	PSO3
CO1	3	-	-
CO2	3	-	2
CO3	3	-	2
CO4	-	-	-
CO5	2	3	-
CO6	2	3	-

Note: H-Highly correlated=3, M-Medium correlated=2, L-Less correlated=1

SYLLABUS

Course code	Course Name	L-T-P -Credits	Year of Introduction
CS202	**Computer Organization and Architecture**	**3-1-0-4**	**2016**

Pre-requisite: CS203 Switching theory and logic design

Course Objectives

1. To impart an understanding of the internal organization and operations of a computer.
2. To introduce the concepts of processor logic design and control logic design.

Syllabus

Fundamental building blocks and functional units of a computer. Execution phases of an instruction. Arithmetic Algorithms. Design of the processing unit – how arithmetic and logic operations are performed. Design of the control unit – hardwired and microprogrammed control. I/O organisation – interrupts, DMA, different interface standards. Memory Subsystem – different types.

Expected outcome

Students will be able to:

1. identify the basic structure and functional units of a digital computer.
2. analyze the effect of addressing modes on the execution time of a program.
3. design processing unit using the concepts of ALU and control logic design.
4. identify the pros and cons of different types of control logic design in processors.
5. select appropriate interfacing standards for I/O devices.
6. identify the roles of various functional units of a computer in instruction execution.

Text Books:

1. Hamacher C., Z. Vranesic and S. Zaky, *Computer Organization* ,5/e, McGraw Hill, 2011.
2. Mano M. M., Digital Logic & Computer Design, 4/e, Pearson Education, 2013.

References:

1. Mano M. M., Digital Logic & Computer Design, 4/e, Pearson Education, 2013.
2. Patterson D.A. and J. L. Hennessey, Computer Organization and Design, 5/e, Morgan Kauffmann Publishers, 2013.
3. William Stallings, Computer Organization and Architecture: Designing for Performance, Pearson, 9/e, 2013.
4. Chaudhuri P., Computer Organization and Design, 2/e, Prentice Hall, 2008.
5. Rajaraman V. and T. Radhakrishnan, Computer Organization and Architecture, Prentice Hall, 2011.
6. Messmer H. P., The Indispensable PC Hardware Book, 4/e, Addison-Wesley, 2001

Course Plan

Module	Contents	Hours (51)	Sem.ExamMarks
I	**Basic Structure of computers**–functional units – basic operational concepts –bus structures – software. Memory locations and addresses – memory operations – instructions and instruction sequencing – addressing modes – ARM Example (programs not required). Basic I/O operations – stacks subroutine calls.	6	15%
II	**Basic processing unit – fundamental concepts – instruction cycle - execution of a complete instruction –multiple- bus organization – sequencing of control signals.** **Arithmetic algorithms: Algorithms for multiplication and division of binary and BCD numbers — array multiplier —Booth's multiplication algorithm — restoring and non-restoring division — algorithms for floating point, multiplication and division.**	10	15%
FIRST INTERNAL EXAMINATION			
III	**I/O organization:** accessing of I/O devices – interrupts –direct memory access –buses –interface circuits –standard I/O interfaces (PCI, SCSI, USB)	8	15%
IV	**Memory system :** basic concepts –semiconductor RAMs –memory system considerations – semiconductor ROMs –flash memory –cache memory and mapping functions.	9	15%

SECOND INTERNAL EXAMINATION			
V	**Processor Logic Design:** Register transfer logic – inter register transfer – arithmetic, logic and shift micro operations –conditional control statements. **Processor organization:**–design of arithmetic unit, logic unit, arithmetic logic unit and shifter –status register –processor unit –design of accumulator.	9	20%
VI	**Control Logic Design:** Control organization – design of hardwired control –control of processor unit –PLA control. **Micro-programmed control:** Microinstructions –horizontal and vertical micro instructions – micro-program sequencer –micro programmed CPU organization.	9	20%
END SEMESTER EXAM			

Question Paper Pattern:

1. There will be *five* parts in the question paper – A, B, C, D, E
2. Part A
    a. Total marks : 12
    b. *Four* questions each having *3* marks, uniformly covering module I and II; All *four* questions have to be answered.
3. Part B
    a. Total marks : 18
    b. *Three* questions each having *9* marks, uniformly covering module I and II; T*wo* questions have to be answered. Each question can have a maximum of three subparts
4. Part C
    a. Total marks : 12
    b. *Four* questions each having *3* marks, uniformly covering module III and IV; All *four* questions have to be answered.
5. Part D
    a. Total marks : 18
    b. *Three* questions each having *9* marks, uniformly covering module III and IV; T*wo* questions have to be answered. Each question can have a maximum of three subparts
6. Part E

a. Total Marks: 40
b. *Six* questions each carrying 10 marks, uniformly covering modules V and VI; *four* questions have to be answered.
c. A question can have a maximum of three sub-parts.

There should be at least 60% analytical/numerical

QUESTION BANK

MODULE I				
Q:NO:		**CO**	**KL**	**PAGE NO:**
1	Write the three-address, two-address and one-address representations of the operation below with relevant assumptions: $C \leftarrow [A] + [B]$	CO1	K3	26
2	Write the use of linkage register in subroutine invocation.	CO1	K3	66
3	Illustrate various addressing modes with proper examples	CO1	K3	36
4	Describe any 5 addressing modes with examples	CO1	K2	36
5	Explain single bus organization with the help of a diagram. Specify with examples, how memory operations are done in the given organization.	CO1	K5	12
6	Write notes on condition codes.	CO1	K3	34
7	Explain indirect addressing with an example	CO1	K5	38
8	Describe , with proper examples, the role of processor stack in subroutine call and return.	CO1	K2	64
9	Write notes on three address, two address and one address instructions, giving example for each.	CO1	K3	26
10	Differentiate between big-endian and little-endian byte assignments.	CO1	K4	21

	MODULE II			
1	Give the control sequence for execution of instruction Add[R2],R1 using a single bus organization.	CO2	K2	78
2	Explain the process of storing a word in memory using a single bus organization. Specify which all control signals will be activated.	CO2	K5	77
3	Explain restoring method of division with the help of a flow chart.	CO2	K5	112
4	Explain the flow charts for floating point multiplication and division.	CO2	K5	120
5	Explain circuit arrangement for binary division with the help of a diagram.	CO2	K5	108
6	Write notes on control unit organization with the help of a diagram.	CO2	K3	86
7	Draw the flow chart for Booth's Multiplication algorithm.	CO2	K1	101
8	Design a 2x2 array multiplier.	CO2	K6	97
9	Explain non-restoring method of division with the help of a flow chart.	CO2	K5	113
	MODULE III			
1	Differentiate between programmed I/O and interrupt driven I/O.	CO3	K4	125
2	Write about vectored interrupts.	CO3	K3	131
3	Explain the important Data transfer signals on the PCI bus.	CO3	K5	161
4	Distinguish between centralized and	CO3	K4	140

	distributed bus arbitration.			
5	Explain the procedure and the packets used for an output transfer in USB interface.	CO3	K5	172
6	Write down the sequence of events take place in SCSI controller.	CO3	K3	166
7	Explain Direct Memory Access. What is burst mode DMA?	CO3	K5	139
8	Describe the functions of I/O interface.	CO3	K2	125
9	Explain the different phases in SCSI bus.	CO3	K5	166
10	Distinguish between Parallel port and Serial Port.	CO3	K4	150
	MODULE IV			
1	Show that dynamic RAMs need constant refreshing. How is this done?	CO4	K2	185
2	Explain the LRU cache replacement algorithm.	CO4	K5	198
3	Distinguish between synchronous and asynchronous DRAMs.	CO4	K4	185
4	Write notes on set associative cache mapping.	CO4	K3	198
5	Explain Direct-mapped cache.	CO4	K5	196
6	Write notes on flash memory.	CO4	K3	192
7	Distinguish between Memory Access Time and Memory Cycle Time.	CO4	K4	181
8	Discuss about the different types of Read only memories.	CO4	K2	190

MODULE V				
1	Give a sample design for generating status bits for a 8-bit ALU.	CO5	K2	244
2	Draw a labelled block diagram of a processor unit with seven registers R1 to R7, a status register, ALU with 3-selection variables and C_{in} , and shifter with 3 selection variables.	CO5	K1	248
3	Discuss shift and conditional control micro operations.	CO5	K2	209
4	Design a bus system for interconnecting four n bit registers.	CO5	K6	211
5	Describe the different ways in which a general-purpose processor unit can be organized. Write notes on conditional control statements.	CO5	K2	212
6	Describe processor organization with diagram using a) scratchpad memory b) Two port memory.	CO5	K2	217
7	Write notes on status register.	CO5	K3	243
MODULE VI				
1	Using a block diagram analyze the design of a microprogram control for a processor unit.	CO6	K4	284
2	What is a control word? With the help of proper illustrations and assumptions show how a designer would compose a control word for the processor unit.	CO6	K3	297
3	With the help of a diagram establish the functioning of microprogram sequencer in a microprogram controlled processor.	CO6	K5	292
4	Compare vertical and horizontal microinstruction formats, giving examples.	CO6	K5	299
5				
6	Describe the purpose of microprogram sequencing. How is it carried out?	CO6	K2	290

7	Briefly explain, with diagrams, the different methods for control organization. Write notes on microprogrammed CPU organization.	CO6	K5	294
8	Explain with the help of a diagram, the working of microprogram sequencer.	CO6	K5	290
9	Describe the steps in control logic design with the help of an example.(Example can be realized using either hardwired or microprogrammed control organization.)	CO6	K2	275
10	What is the significance of a micro program sequencer? Explain its working with the help of a diagram.	CO6	K5	290

APPENDIX 1

CONTENT BEYOND THE SYLLABUS

SL.NO:	TOPIC	PAGE NO:
1	Graphene Processors	301
2	Memristor Storage	302
3	Micron Hybrid Memory Cube	303
4	Neuromorphic Chips	303
5	The top 10 technology trends	304

MODULE NOTES

CS202 Computer Organization and Architecture

Module I

Basic Structure of computers.

i) Functional Units.

A computer consists of five functionally independent main parts: input, memory, arithmetic and logic, output and control units as shown in Figure I-1.

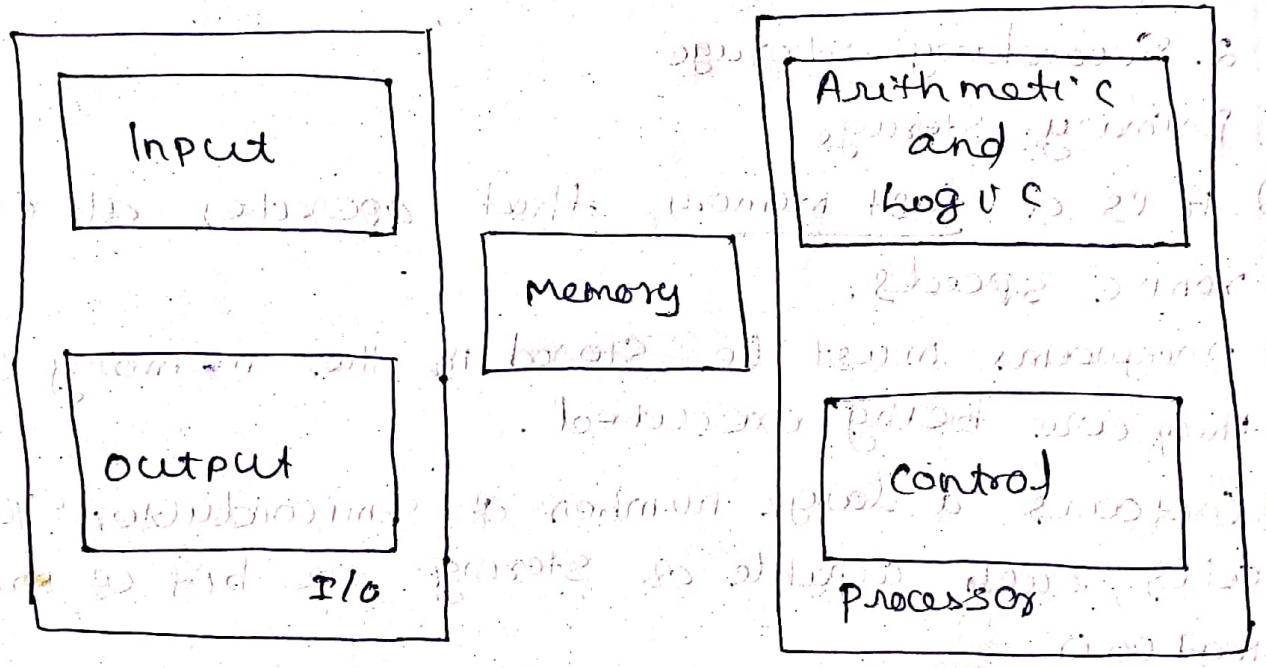


Figure 1.1 Basic functional units of a computer.

I Input unit

1. The input unit accepts coded information from
   (a) Human operators
   (b) Electro mechanical devices such as keyboards, mouses, joysticks, trackballs, microphone
   (c) Other computers over digital communication line.

Scanned with CamScanner

II Memory Unit

The information received from the input device is either stored in the computer's memory for later reference or immediately used by the arithmetic and logic circuitry to perform the desired operations.

There are two classes of storage,

1. primary storage and
2. Secondary storage

1. Primary Storage

a) It is a fast memory that operates at electronic speeds.

b) Programs must be stored in the memory while they are being executed.

c) Contains a large number of semiconductor storage cells, each capable of storing one bit of information

d) Cells are rarely read or written as individual cells but instead are processed in group of fixed size called words.

e) The memory is organized so that the contents of one word, containing n bits, can be stored or retrieved in one basic operation.

f) To access any word in the memory a distinct address is associated with each word location.

Scanned with CamScanner

- Addresses are numbers that identify successive locations.

g) word length: The number of bits in each word, range from 16 to 64 bits.

(i) Data are usually processed within a machine in units of words, multiples of words, or parts of words.

(ii) When the memory is accessed, usually only one word of data is read or written

h) Random-access memory (RAM):

Memory in which any location can be reached in a short and fixed amount of time after specifying its address is called random-access memory.

(i) Memory access time: (This time is fixed).

The time required to access one word is called the memory access time. range: a few nanoseconds (ns) to about 100s for modern RAM units.

(ii) The small, fast, RAM units are called caches.

(iii) The largest and slowest RAM unit is referred to as the main memory.

2. secondary storage

Although primary storage is essential, it tends to be expensive. This additional, cheaper, secondary storage is used when large amounts

Scanned with CamScanner

of data and many programs have to be stored particularly for information that is accessed infrequently.
eg: Magnetic disks, tapes and optical disks (CD-ROMs).

III ARITHMETIC AND LOGIC UNIT

1. Most computer operations are executed in the arithmetic and logic unit (ALU) of the processor.
2. When operands are brought into the processor, they are stored in high-speed storage elements called registers.
   (i) Each register can store one word of data.
   (ii) Access times to registers are somewhat faster than access times to the fastest cache unit in the memory hierarchy.
3. The processed data are stored in the memory or retained in the processor.
4. The control and arithmetic and logic units are many times faster than other devices connected to a computer system. This enables a single processor to control number of external devices.

IV OUTPUT UNIT

1) The output unit is the counterpart of the input unit.
2) send processed results to the outside world.
   eg: printer.
3) Some units, such as graphic displays, provide both an output function and an input function. The dual role of such units is the reason for

Scanned with CamScanner

using the single name I/O unit in many cases.

I5 CONTROL UNIT

1) The task of coordination among the memory, arithmetic and logic and input and output units, store and process information and perform inputs and output operations, is done by the control unit.

2) The control unit is effectively the nerve center that sends control signals to other units and senses their states.

3) Timing signals:
Timing signals are signals that determine when a given action is to taken place.
- Generated by the control circuit.
- controlling of data transfers between the processor and the memory.

4) Much of the control circuitry is physically distributed throughout the machine.

5) A large set of control lines carries the signals used for timing and synchronization of events in all units.

The information received is either stored in the computer's memory for later reference or immediately used by the arithmetic and logic circuitry to perform the desired operations. The processing steps are determined by a program stored in the memory. Finally, the results are sent back to the outside world through the output unit. All of these actions are coordinated by the control unit.

Scanned with CamScanner

Information handled by a computer.

1. Instructions or machine instructions, are explicit commands that
   a) Govern the transfer of information within a computer as well as between the computer and its I/O devices.
   b) Specify the arithmetic and logic operations to be performed.

2. Program
   A list of instructions that performs a task is called a program.
   - Stored in the memory
   - Processor then fetches the instructions that make up the program from the memory, one after another, and performs the desired operations.
   - The computer is completely controlled by the stored program, except for possible external interruption by an operator or by I/O devices connected to the machine.

3. Data
   Data are numbers and encoded characters that are used as operands by the instructions.
   a) The term data, however, is often used to mean any digital information
      - an entire program (a list of instructions) may be considered as data if it is to be processed by another program.
        eg: source program to the compiler.

Scanned with CamScanner

4. Format of information

Information handled by a computer must be encoded in a suitable format.

a) Most present-day hardware employs digital circuits that have only two stable states, ON and OFF.

b) Each number, character, or instruction is encoded as a string of binary digits called bits, each having one of two possible values, 0 or 1.

eg: 1) positional binary notation for numbers - BCD (Binary-coded decimal) - each decimal digit is encoded by four bits.

2) Alphanumeric characters
- ASCII (American Standard Code for Information Interchange)
: each character is represented as a 7-bit code
- EBCDIC (Extended Binary-Coded Decimal Interchange Code)
: each character is denoted by eight bits.

- The operation of a computer can be summerized as follows:

1. The computer accepts information in the form of programs and data through an input unit and stores it in the memory.
2. Information stored in the memory is fetched, under program control, into an arithmetic and logic unit, where it is processed.
3. Processed information leaves the computer through an output unit.

Scanned with CamScanner

4. All activities inside the machine are directed by the control unit.

ii) BASIC OPERATIONAL CONCEPTS

1. The activity in a computer is governed by instructions.
2. To perform a given task, an appropriate program consisting of a list of instructions is stored in the memory.
3. Individual instructions are brought from the memory into the processor.
4. Data to be used as operands are also stored in the memory.

eg: Add LOCA, R0

- This instruction adds the operand at memory location LOCA to the operand in a register in the processor, R0, and places the sum into register R0.
- The original contents of location LOCA are preserved, whereas those of R0 are overwritten.

Steps required

1. The instruction is fetched from the memory to the processor.
2. The operand at LOCA is fetched and added to the contents of R0.
3. The resulting sum is stored in register R0.
5. Transfers between the memory and the processor.

Transfer started by sending the address of the memory location to be accessed to the memory unit and issuing the appropriate control signals.

Scanned with CamScanner

Registers in the processor:

i) Instruction register (IR)
- holds the instructions that is currently being executed.
- output is available to the control circuits,

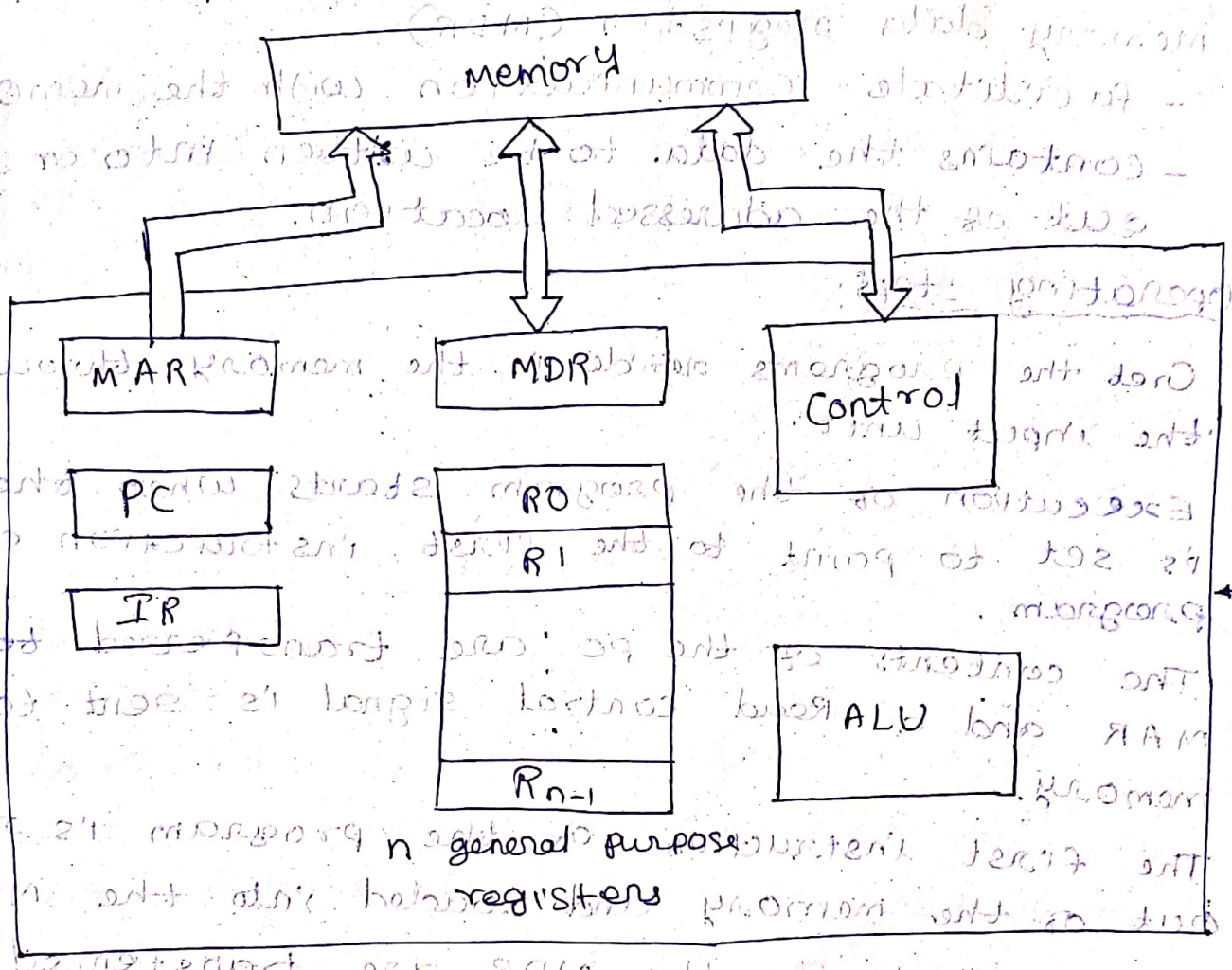


Figure 1.2 connections between the processor and the memory.

ii) Program counter (PC)
- specialised register
- keeps track of the execution of a program
- contains the memory address of the next instruction to be fetched and executed.

Scanned with CamScanner

- During the execution of an instruction, the contents of the PC are updated.

iii) Memory Address register (MAR)
- Registers facilitate communication with the memory
- holds the address of the location to be accessed.

iv) Memory data register (MDR)
- facilitate communication with the memory.
- contains the data to be written into or read out of the addressed location.

Operating steps:

1. Get the programs residing in the memory through the input unit.
2. Execution of the program starts when the PC is set to point to the first instruction of the program.
3. The contents of the PC are transferred to the MAR and a Read control signal is sent to the memory.
4. The first instruction of the program is read out of the memory and loaded into the MDR.
5. The contents of the MDR are transferred to the IR.
6. The instruction is ready to be decoded and executed.
7. If the instruction involves an operation to be performed by the ALU,
   a) Obtain the required operands.

Scanned with CamScanner

i) If operand resides in the memory then fetch it by sending its address to the MAR and initiating a Read cycle.

ii) When the operand has been read from the memory into the MDR

iii) It is transferred from the MDR to the ALU.

b) After one or more operands are fetched, the ALU can perform the desired operation.

c) If the result of this operation is to be stored in the memory, then the result is sent to the MDR.

d) The address of the location where the result is to be stored is sent to the MAR, and a Write cycle is initiated.

8. As soon as the execution of the current instruction is completed, a new instruction fetch may be started.

9. Normal execution of programs may be preempted if some device requires urgent servicing. To do this, the device raises an interrupt signal.

(i) An interrupt is a request from an I/O device for service by the processor.

(ii) The processor provides the requested service by executing an appropriate interrupt-service routine.

10. When the interrupt-service routine is completed, the state of the processor is restored so that the interrupted program may continue.

Scanned with CamScanner

BUS STRUCTURES

1. To achieve a reasonable speed of operation, a computer must be organized so that all its units can handle one full word of data at a given time.
2. To form an operational system all the functions of individual parts of a computer must be connected in some organized way.
3. When a word of data is transferred between units, all its bits are transferred in parallel, that is, the bits are transferred simultaneously over many wires, or lines, one bit per line.
4. A group of lines that serves as a connecting path for several devices is called a bus.
5. In addition to the lines that carry the data, the bus must have lines for address and control purposes.
6. The simplest way to interconnect functional units is to use a single bus, as shown in Fig 1.3.
   i) All units are connected to this bus.
   ii) Only two units can actively use the bus at any given time because the bus can be used for only one transfer at a time.
7. Bus control lines are used to arbitrate multiple requests for use of the bus.
8. Single-bus structure is
   - low cost,
   - flexibility for attaching peripheral devices.

Scanned with CamScanner

9. Multiple buses
- achieve more concurrency in operations by allowing two or more transfers to be carried out at the same time.
- better performance.
- an increased cost.

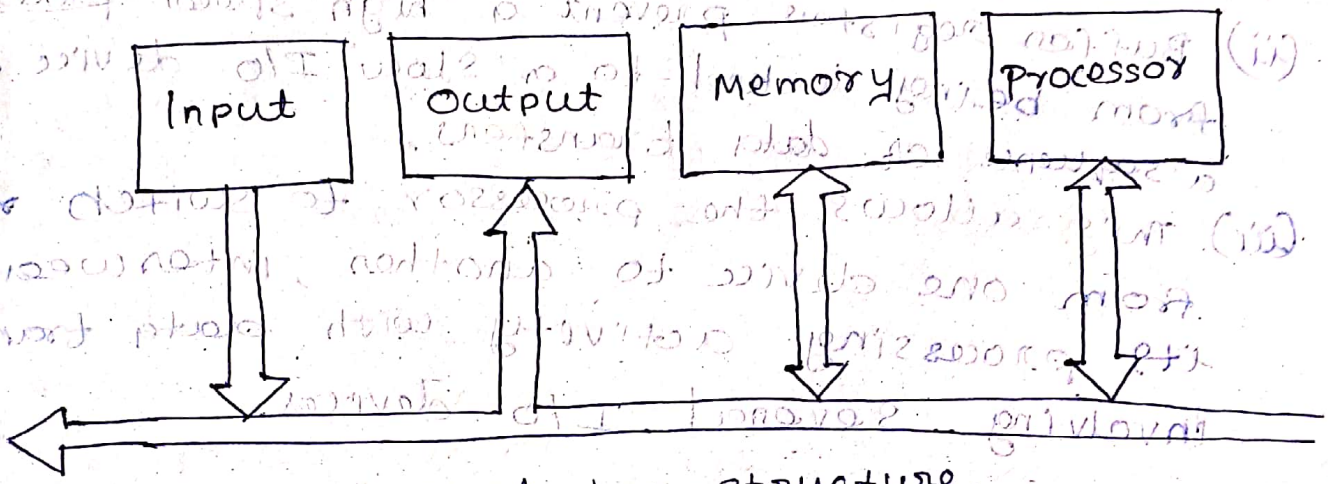


Figure 1.3 Single-bus structure.

10. The devices connected to a bus vary widely in their speed of operation.
a) - electromechanical devices are relatively slow, eg: keyboard and printer
b) - Magnetic or optical disks are considerably faster.
c) - memory and processor units operate at electronic speeds.
d) - All these devices must communicate with each other over a bus.
e) - An efficient transfer mechanism that is not constrained by the slow devices and that can be used to smooth out the differences in timing among processors, memories and external devices is

Scanned with CamScanner

necessary for this there is a common approach known as buffer registers with the devices to hold the information during transfers.

(i) Buffer registers smooth out timing differences among processors, memories, and I/O devices.

(ii) Buffer registers prevent a high-speed processor from being locked to a slow I/O device during a sequence of data transfers.

(iii) This allows the processor to switch rapidly from one device to another, interweaving its processing activity with data transfers involving several I/O devices.

SOFTWARE

In order for a user to enter and run an application program, the computer must already contain some system software in its memory.

System software

System software is a collection of programs that are executed as needed to perform functions such as

i) Receiving and interpreting user commands

ii) Entering and editing application programs and storing them as files in secondary storage devices

iii) Managing the storage and retrieval of files in secondary storage devices

iv) Running standard application programs such as

Scanned with CamScanner

word processors, spreadsheets, or games, with data supplied by the user.

v) Controlling I/O units to receive input information and produce output results.

vi) Translating programs from source form prepared by the user into object form consisting of machine instructions.

vii) Linking and running user-written application programs with existing standard library routines, such as numerical computation packages.

Application programs

i) Application programs are usually written in a high-level programming language, such as C, C++, Java, or Fortran, in which the programmer specifies mathematical or text-processing operations.

ii) A programmer using a high-level language need not know the details of machine program instructions.

text editor

i) Text editor is another important system program that all programmers use.

ii) It is used for entering and editing application programs.

iii) The user of this program interactively executes commands that allow statements of a source program entered at a keyboard to be accumulated in a file.

i) A file is simply a sequence of alphanumeric characters or binary data that is stored in memory or in secondary storage.

Scanned with CamScanner

Operating System (OS)

i) OS is a large program or a collection of routines.
ii) OS is used to control the sharing of and interaction among various computer units as they execute application programs.
iii) The OS routines perform the tasks required to assign computer resources to individual application programs.
iv) Tasks of OS include as
   a) assigning memory and magnetic disk space to program and data files,
   b) moving data between memory and disk units and
   c) handling I/O operations.

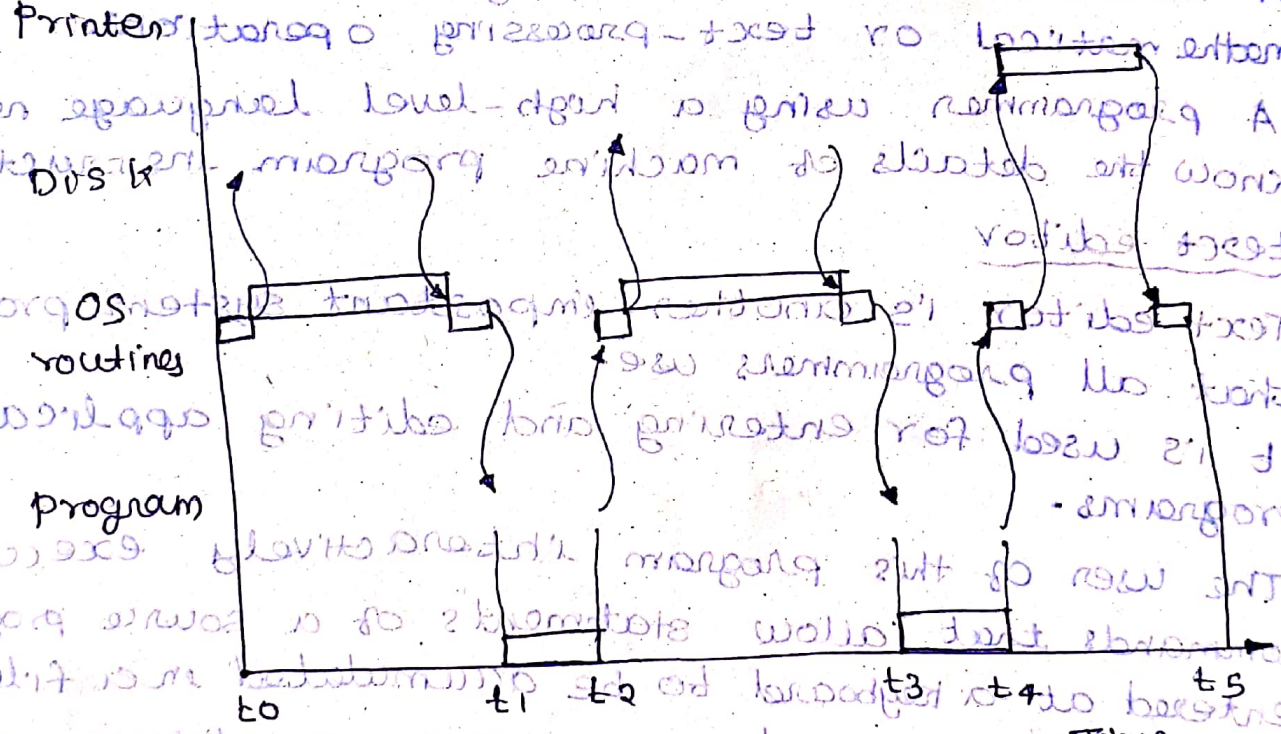


Figure 1.4 User program and OS routine sharing of the processor.

Scanned with CamScanner

Steps involved in running one application program.

1. Application program has been compiled from a high-level language form into a machine language form and stored on the disk.
2. Transfer this file into the memory. When the transfer is complete, execution of the program is started.
3. If the part of the program's task involves reading a data file from the disk into the memory, performing some computation on the data, and printing the results.
4. When execution of the program reaches the point where the data file is needed, the program requests the operating system to transfer the data file from the disk to the memory. This is done by OS and the OS passes execution control back to the application program, which then proceeds to perform the required computation.
5. When the computation is completed and the results are ready to be printed, the application program again sends a request to the operating system.
6. An OS routine is then executed to cause the printer to print the results, as shown in Figure 1.4.

i) t_0 to t_1 — an OS routine initiating
t_2 to t_3 loading the application program from
t_4 to t_5 disk to memory, waits until the transfer is completed, and then passes execution control to the application program.

Scanned with CamScanner

7. Computer resources can be used more efficiently if several application programs are to be processed.

i) the disk and the processor are idle during most of the time period t_4 to t_5.

— The operating system can load the next program to be executed into the memory from the disk while the printer is operating.

ii) during t_0 to t_1, the operating system can arrange to print the previous program's results while the current program is being loaded from the disk.

8. The operating system manages the concurrent execution of several application programs to make the best possible use of computer resources. This pattern of concurrent execution is called multi-programming or multitasking.

MEMORY LOCATIONS AND ADDRESSES

i) How the memory is organized?

1. The memory consists of many millions of storage cells.
2. Each cell can store a bit of information having the value 0 or 1.
3. A single bit represents a very small amount of information.
4. Bits are seldom handled individually because of this, the memory is organized so that a group of n bits can be stored or retrieved in a

Scanned with CamScanner

single, basic operation.

5. Each group of n bits is referred to as a word of information, and n is called the word length.

6. The memory of a computer can be schematically represented as a collection of words as shown in Figure 1.5.

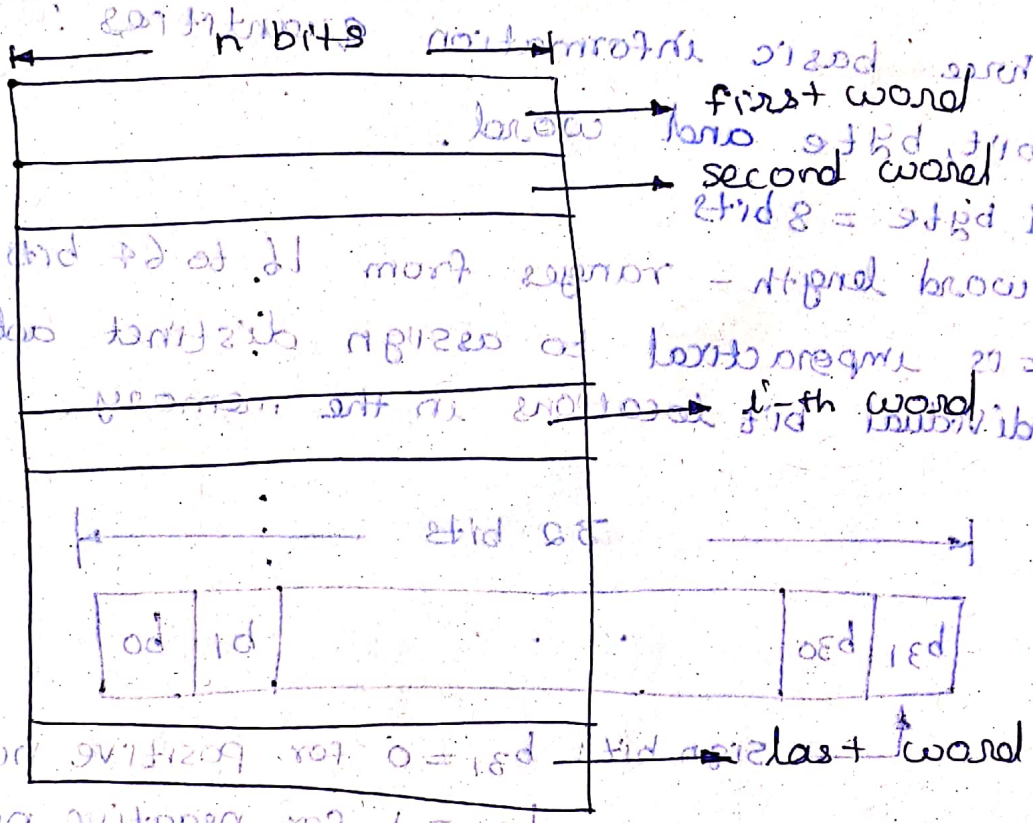


Figure 1.5 Memory words.

7. Modern computers have word lengths that typically range from 16 to 64 bits.

8. If the word length of a computer is 32 bits, a single word can store a 32-bit 2's complement number or four ASCII characters, as shown in Figure 1.6.

9. A unit of 8 bits is called a byte.

Scanned with CamScanner

2^{10} = 1 Kilo — 10 bit address

2^{20} = 1 Mega — 20 bit address

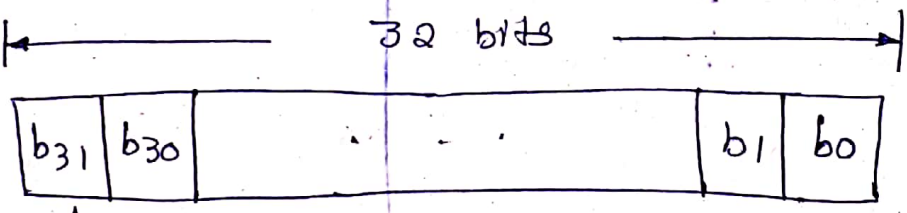
2^{30} = 1 Giga — 30 bit address

2^{40} = 1 Tera — 40 bit address.

1. BYTE ADDRESSABILITY

i) Three basic information quantities:
bit, byte and word.
1 byte = 8 bits
word length - ranges from 16 to 64 bits.

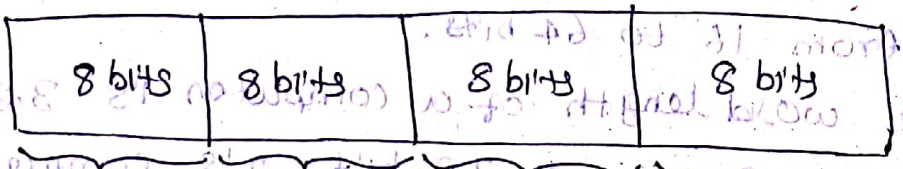
ii) It is impractical to assign distinct addresses to individual bit locations in the memory.



$b_{31} = 0$ for positive numbers

$b_{31} = 1$ for negative numbers

(a) A signed integer



(b) Four characters

Figure 1.6 Examples of encoded information in a 32-bit word.

Scanned with CamScanner

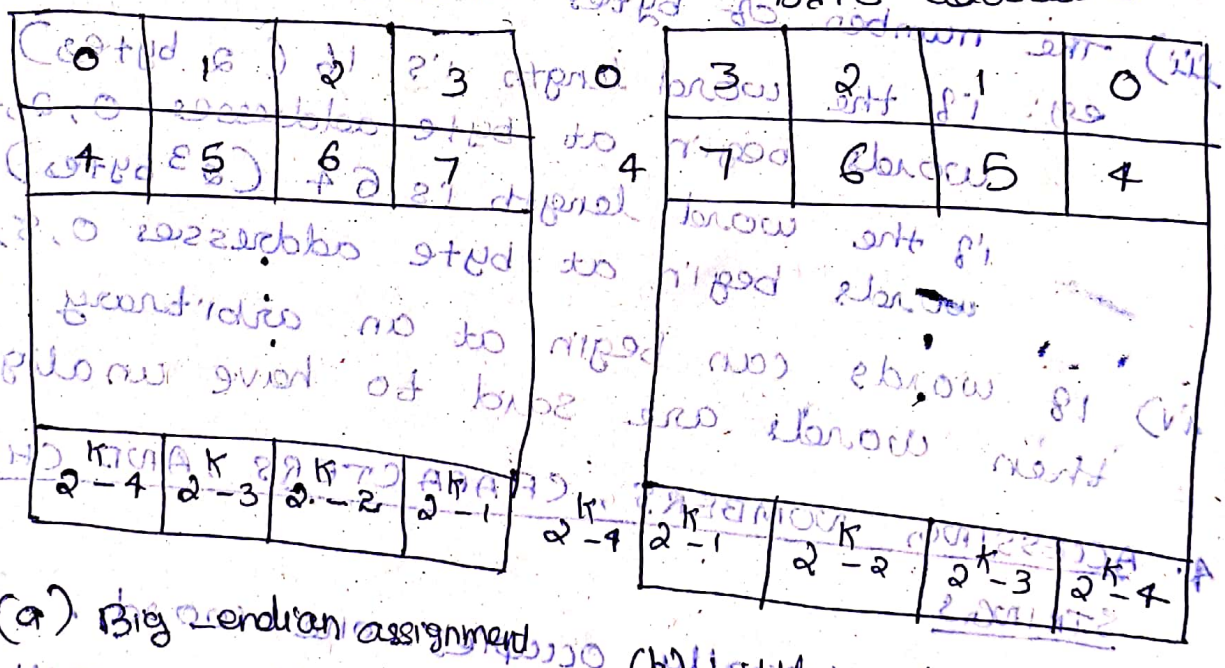
iii) The most practical assignment is to have successive addresses refer to successive byte locations in the memory.

iv) The term byte-addressable memory is used for this assignment.

v) Byte locations have addresses 0, 1, 2, ... Thus, if the word length of the machine is 32 bits, successive words are located at addresses 0, 4, 8, ... with each word consisting of four bytes.

2. BIG-ENDIAN AND LITTLE-ENDIAN ASSIGNMENTS

There are two ways that byte addresses can be assigned across words



(a) Big-endian assignment (b) Little-endian assignment

Figure 1-7 Byte and word addressing.

Scanned with CamScanner

i) The name big-endian is used when lower byte addresses are used for the more significant bytes of the word.

ii) The name little-endian is used when the lower byte addresses are used for the less significant bytes (the rightmost bytes) of the word.

iii) Both little-endian and big-endian assignments are used in commercial machines.

3. WORD ALIGNMENT

i) In the case of a 32-bit word length, natural word boundaries occur at addresses 0, 4, 8, ... then the word locations have aligned addresses.

ii) words are said to be aligned in memory if they begin at a byte address that is a multiples of the number of bytes in a word.

iii) The number of bytes in a word is a power of 2.
eg: if the word length is 16 (2 bytes), aligned words begin at byte addresses 0, 2, 4, ...
if the word length is 64 (2^3 bytes), aligned words begin at byte addresses 0, 8, 16, ...

iv) If words can begin at an arbitrary byte address then words are said to have unaligned addresses.

4. ACCESSING NUMBERS, CHARACTERS AND CHARACTER STRINGS

i) A number usually occupies one word.

ii) A number can be accessed in the memory by specifying its word address.

Scanned with CamScanner

iii) Individual characters can be accessed by their byte address.

a) The beginning of the string is indicated by giving the address of the byte containing its first character.

b) Successive byte locations contain successive characters of the string.

c) Two ways to indicate the length of the string

1. A special control character with the meaning "end of string" can be used as the last character in the string, or
2. A separate memory word location or processor register can contain a number indicating the length of the string in bytes.

MEMORY OPERATIONS

i) Both program instructions and data operands are stored in the memory.

ii) To execute an instruction,

a) the processor control circuits must cause the word (or words) containing the instruction to be transferred from the memory to the processor.

b) operands and results must also be moved between the memory and the processor.

c) Two basic operations involving the memory are needed, namely, Load (or Read or Fetch) and Store (or Write).

Scanned with CamScanner

iii) The Load operation:

a) transfers a copy of the contents of a specific memory location to the processor.

b) The memory contents remain unchanged.

c) To start a load operation, the processor sends the address of the desired location to the memory and requests that its contents be read.

d) The memory reads the data stored at that address and sends them to the processor.

iv) The store operation:

a) transfers an item of information from the processor to a specific memory location, destroying the former contents of that location.

b) The processor sends the address of the desired location to the memory, together with the data to be written into that location.

v) An information item of either one word or one byte can be transferred between the processor and the memory in a single operation.

a) the processor contains a small number of registers, each capable of holding a word.

b) these registers are either the source or the destination of a transfer to or from the memory.

c) When a byte is transferred, it is usually located in the low-order (rightmost) byte position of the register.

(24)

(33)

Scanned with CamScanner

INSTRUCTIONS AND INSTRUCTION SEQUENCING

A computer must have instructions capable of performing four types of operations:

1. Data transfers between the memory and the processor registers
2. Arithmetic and logic operations on data
3. Program sequencing and control
4. I/O transfers.

Notations used in instructions:

1. Register Transfer Notations

i) The contents of a location are denoted by placing square brackets around the name of the location. The expression $R1 \leftarrow [LOC]$ means that the contents of memory location LOC are transferred into processor register R1.

ii) Register Transfer Notation (RTN)

a) The right-hand side of an RTN expression always denotes a value, and the left-hand side is the name of a location where the value is to be placed, overwriting the old contents of that location.

eg: $R3 \leftarrow [R1] + [R2]$

adds the contents of register R1 and R2, and then places their sum into register R3.

2. ASSEMBLY LANGUAGE NOTATION

To represent machine instructions and programs for this assembly language format is used.

Scanned with CamScanner

eg:

Move LOC, R1

Causes the transfer from memory location LOC to processor register R1. The contents of LOC are unchanged by the execution of this instruction, but the old contents of register R1 are overwritten.

eg: Add R1, R2, R3

Adding two numbers contained in processor registers R1 and R2 and placing their sum in R3.

BASIC INTSRUCTION TYPES

1. Three-address instruction.

Add A, B, C

operands A and B are called the source operands, C is called the destination operand, and Add is the operation to be performed on the operands.

general instruction:

operation source1, source2, Destination.

Disadvantage!

If k bits are needed to specify the memory address of each each operand, the encoded form of the above instruction must contain 3k bits for addressing purposes in addition to the bits needed to denote the Add operation.

- a 3-address instruction is too large to fit in one word for a reasonable word length.

Scanned with CamScanner

2. Two-address instructions

operation source, Destination

Add A, B

which performs the operation B ← [A] + [B].
The operand B is both a source and a destination.

Move B, C , C ← [B]

Move B, C
Add A, C } C ← [A] + [B]

Disadvantage:
even two-address instructions will not normally fit into one word for usual word lengths and address sizes.

3. One-address instructions

Machine instructions that specify only one memory operand. When a second operand is needed it is assumed understood implicitly to be in a unique location. Usually the accumulator may be used for this purpose.

Add A

Means, Add the contents of memory location A to the contents of the accumulator register and place the sum back into the accumulator.

Load A

The Load instruction copies the contents of memory location A into the accumulator.

Store A

The Store instruction copies the contents of the

Scanned with CamScanner

accumulator into memory location A. Using only one-address instructions, the operation $C \leftarrow [A] + [B]$ can be performed by executing the sequence of instructions

```
Load A
Add B
Store C
```

Note that the operand specified in the instruction may be a source or a destination, depending on the instruction.

eg: Load instruction, address A specifies the source operand, and the destination location, the accumulator, is implied.

C denotes the destination location in the Store instruction, whereas the source, the accumulator, is implied.

1. some early computers were designed around a single accumulator structure.
2. Most modern computers have a number of general-purpose processor registers - 8 to 32
3. Access to data in these registers is much faster than to data stored in memory locations because the registers are inside the processor.
4. Only a few bits are needed to specify which register takes part in operation. eg: 5 bits are needed for 32 registers.

Eg: Let R_i represent a general-purpose register.

The instructions Load A, R_i

Store R_i, A

Scanned with CamScanner

and Add A, R

are generalizations of the Load, Store and Add instructions for the single-accumulator case, in which register R_i performs the function of the accumulator.

5. Many instructions involve only operands that are in the registers.

Eg: Add R_i, R_j or Add R_i, R_j, R_k

6. To transfer data between different locations.

Move Source, Destination

which places a copy of the contents of Source into Destination.

Move A, R_i is the same as Load A, R_i

and Move R_j, A is the same as Store R_j, A

7. In processors where arithmetic operations are allowed only on operands that are in processor registers, the C = A + B task can be performed by the instruction sequence

```
Move  A, Ri
Move  B, Rj
Add   Ri, Rj
Move  Rj, C
```

8. In processors where one operand may be in the memory but the other must be in a register, an instruction sequence for the required task would be

```
Move  A, Ri
Add   B, Ri
Move  Ri, C
```

9. The speed with which a given task is carried out depends on the time it takes to transfer instructions from memory into the processor and to access the

Scanned with CamScanner

the operands referenced by these instructions.

4. Zero-address instructions

Instructions in which the locations of all operands are defined implicitly (Stack operands in a structure called a pushdown stack.

INSTRUCTION EXECUTION AND STRAIGHT-LINE SEQUENCING

Assume that

1. the computer allows one memory operand per instruction
2. has a number of processor registers.
3. the word length is 32 bits
4. the memory is byte addressable.
5. the three instructions of the program are in successive word locations, starting at location i.
6. a full memory address can be directly specified in a single-word instruction.

Program counter (PC): holds the address of the instruction to be executed next.

Straight-line sequencing: the processor control circuits use the information in the PC to fetch and execute instructions, one at a time, in the order of increasing addresses, this is called straight-line sequencing.

i) During the execution of each instruction, the PC is incremented by 4 to point to the next instruction.

Two-phase procedure for executing a given instruction

First phase (instruction fetch)

i) the instruction is fetched from the memory location whose address is in the PC.

Scanned with CamScanner

ii) the fetched instruction is placed in the instruction register (IR) in the processor.

Second phase (instruction execute)

i) the instruction in IR is examined to determine which operation is to be performed.

ii) the specified operation is then performed by the processor, this often involves

a) fetching operands from the memory or from processor registers

b) performing an arithmetic or logic operation and

c) storing the result in the destination location.

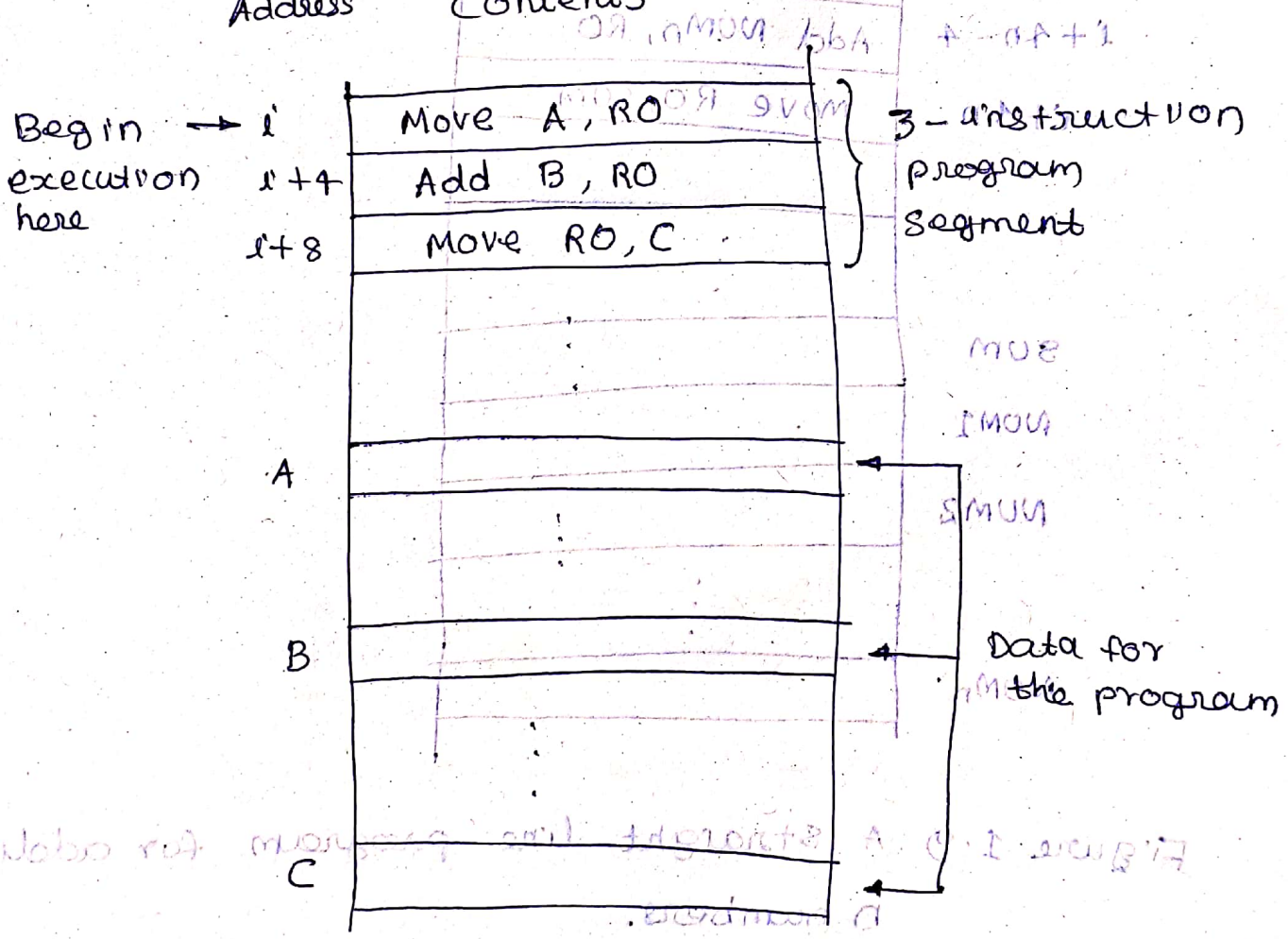


Figure 1.8 A program for C ← [A] + [B].

Scanned with CamScanner

BRANCHING

Consider the task of adding a list of n numbers. Instead of using a long list of Add instructions, it is possible to place a single Add instruction in a program loop as shown in Figure 1.10.

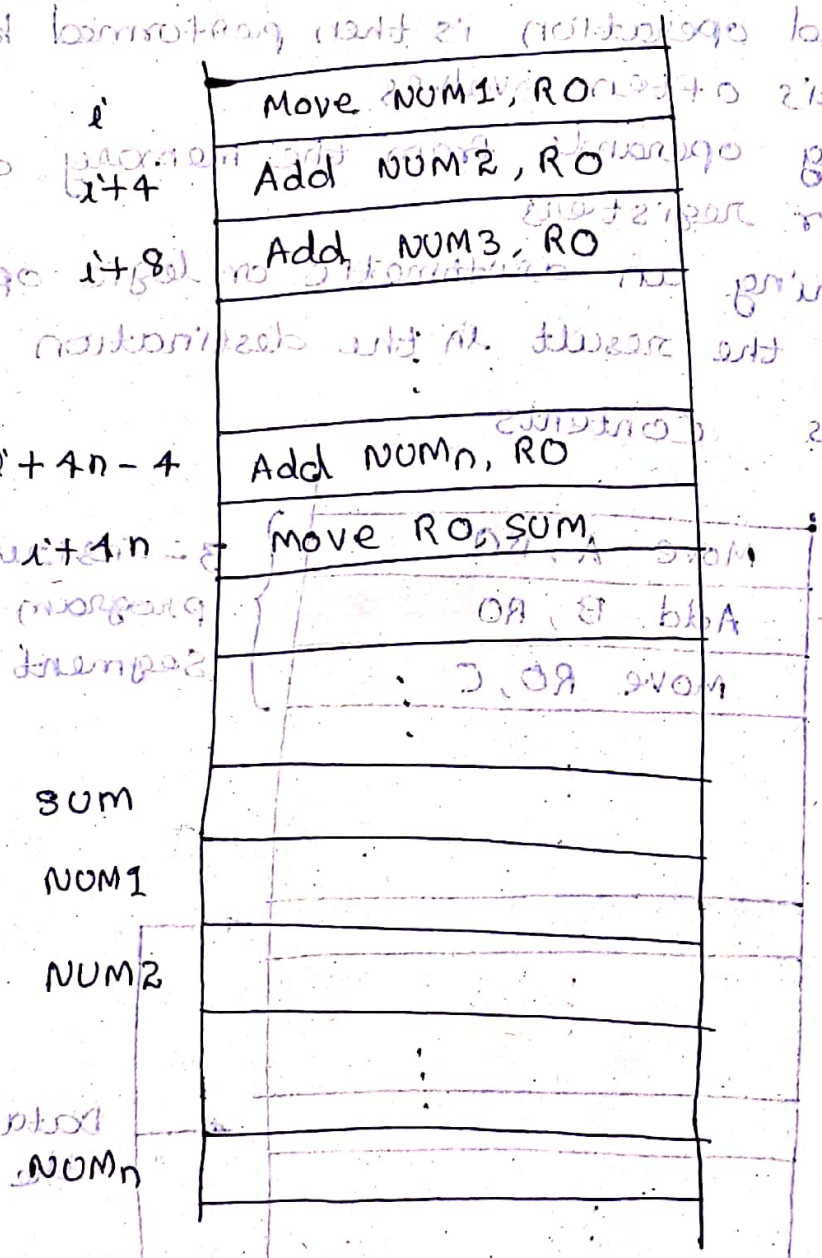


Figure 1.9 A straight-line program for adding n numbers.

Scanned with CamScanner

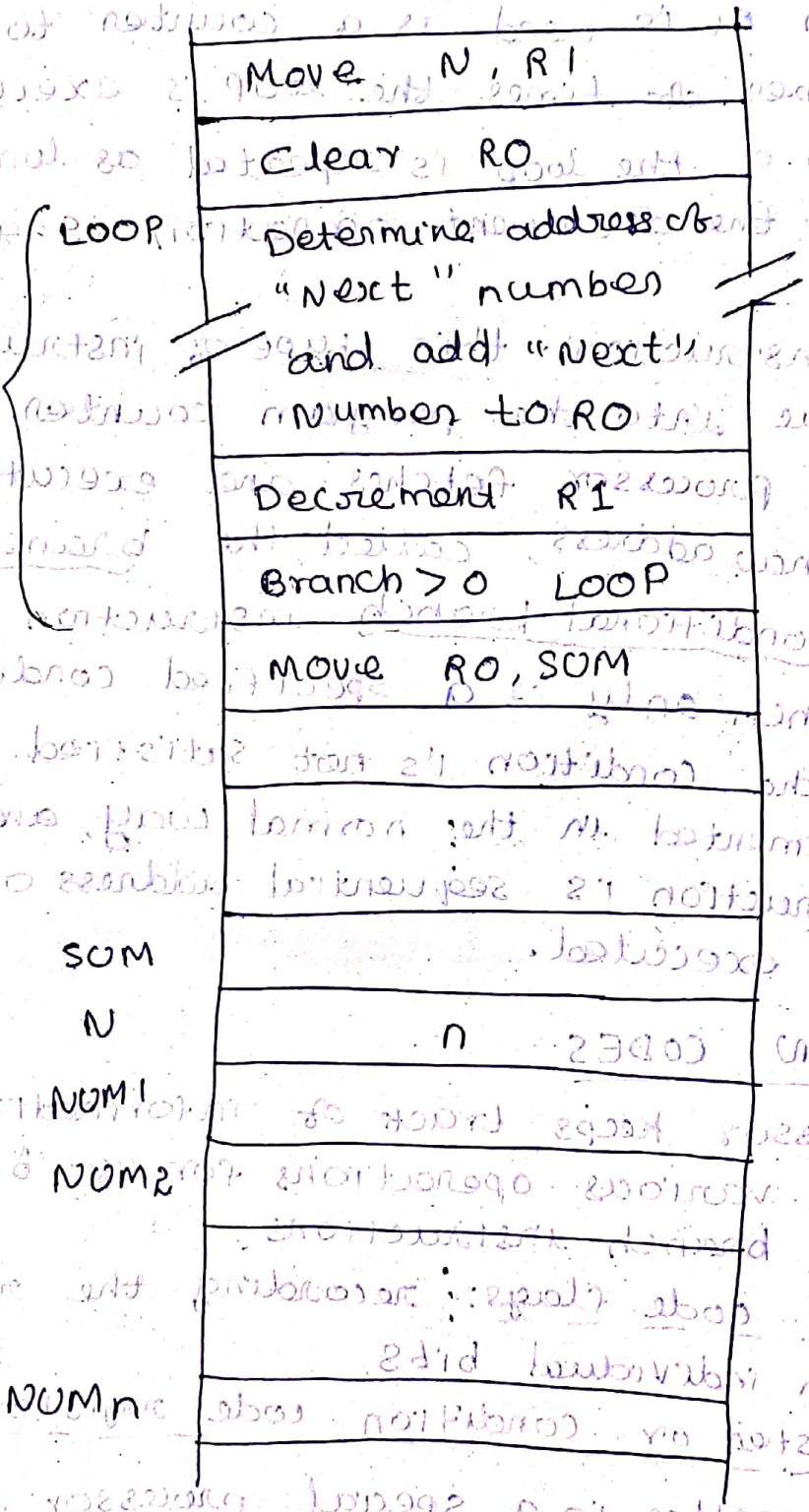


Figure 1.10 Using a loop to add n numbers.

i) The loop is a straight-line sequence of instructions executed as many times as needed.

ii) It starts at location LOOP and ends at the instruction Branch > 0.

Scanned with CamScanner

iii) Register R1 is used as a counter to determine the number of times the loop is executed.

iv) Execution of the loop is repeated as long as the result of the decrement operation is greater than zero.

v) Branch instruction:- this type of instruction loads a new value into the program counter.
   a) the processor fetches and executes the instruction at new address, called the branch target
   b) a conditional branch instruction causes a branch only if a specified condition is satisfied.
   c) If the condition is not satisfied, the PC is incremented in the normal way, and the next instruction in sequential address order is fetched and executed.

CONDITION CODES

The processor keeps track of information about the results of various operations for use by subsequent conditional branch instructions.

conditional code flags: recording the required information in individual bits.

status register or condition code register: Flags are grouped together in a special processor register.

i) Individual condition code flags are set to 1 or cleared to 0, depending on the outcome of the operation performed.

ii) commonly used flags are:

Scanned with CamScanner

a) N (negative) set to 1 if the result is negative; otherwise, cleared to 0

b) Z (zero) set to 1 if the result is 0; otherwise, cleared to 0

c) V (overflow) set to 1 if arithmetic overflow occurs; otherwise, cleared to 0

d) C (carry) set to 1 if a carry-out results from the operation; otherwise, cleared to 0.

GENERATING MEMORY ADDRESSES

How are the addresses to be specified? The memory operand address cannot be given directly in a single Add instruction in the loop. Otherwise, it would need to be modified on each pass through the loop.

As one possibility, suppose that a processor register, Ri, is used to hold the memory address of an operand. If it is initially loaded with the address NUM1 before the loop is entered and is then incremented by 4 on each pass through the loop, it can provide the needed capability.

Addressing modes: are the methods used for specifying flexibly the address of an operand. The details differ from one computer to another, the underlying concepts are the same.

Scanned with CamScanner

ADDRESSING MODES

The different ways in which the location of an operand is specified in an instruction are referred to as addressing modes. When translating a high-level language program into assembly language, the compiler must be able to implement constants, local and global variables, pointers and arrays using the facilities provided in the instruction set of the computer in which the program will be run.

1. Implementation of Variables and constants
2. Indirection and Pointers
3. Indexing and Arrays
4. Relative Addressing
5. Additional Modes

1. IMPLEMENTATION OF VARIABLES AND CONSTANTS

i) In assembly language, a variable is represented by allocating a register or a memory location to hold its value, the value can be changed as needed using appropriate instructions.

Two addressing modes to access variables

a) Register mode — The operand is the contents of a processor register; the name (address) of the register is given in the instruction.

b) Absolute mode (Direct) — The operand is in a memory location; the address of this location is given explicitly in the instruction.

Scanned with CamScanner

Table 1.1 Generic addressing modes

Name	Assembler syntax	Addressing function
Immediate	#value	Operand = value
Register	R_i	$EA = R_i$
Absolute (Direct)	LOC	EA = LOC
Indirect	(R_i)	$EA = [R_i]$
	(LOC)	EA = [LOC]
Index	$X(R_i)$	$EA = [R_i] + X$
Base with index	(R_i, R_j)	$EA = [R_i] + [R_j]$
Base with index and offset	$X(R_i, R_j)$	$EA = [R_i] + [R_j] + X$
Relative	X(PC)	EA = [PC] + X
Autoincrement	$(R_i)+$	$EA = [R_i]$; Increment R_i
Autodecrement	$-(R_i)$	Decrement R_i; $EA = [R_i]$

EA = effective address

Value = a signed number

eg: The instruction Move LOC, R_2 uses Register mode and Absolute mode.

A declaration such as Integer A, B; in a high-level language program will cause the compiler to allocate a memory location to each of the variables A and B. Whenever they are referenced later in the program, the compiler can generate assembly language instructions that use the Absolute mode to access these variables.

Scanned with CamScanner

c) Immediate mode : The operand is given explicitly in the instruction.

eg: Move 200$_{immediate}$, R0

places the value 200 in register R0

i) the immediate mode is only used to specify the value of a source operand.

ii) A common convention to denote the immediate mode is to use the sharp sign (#) in front of value to indicate that this value is to be used as an immediate operand.

eg: Move #200, R0

Assuming that A & B have been declared earlier as variables and may be accessed using the Absolute mode,

```
Move  B, R1
Add   #6, R1
Move  R1, A
```

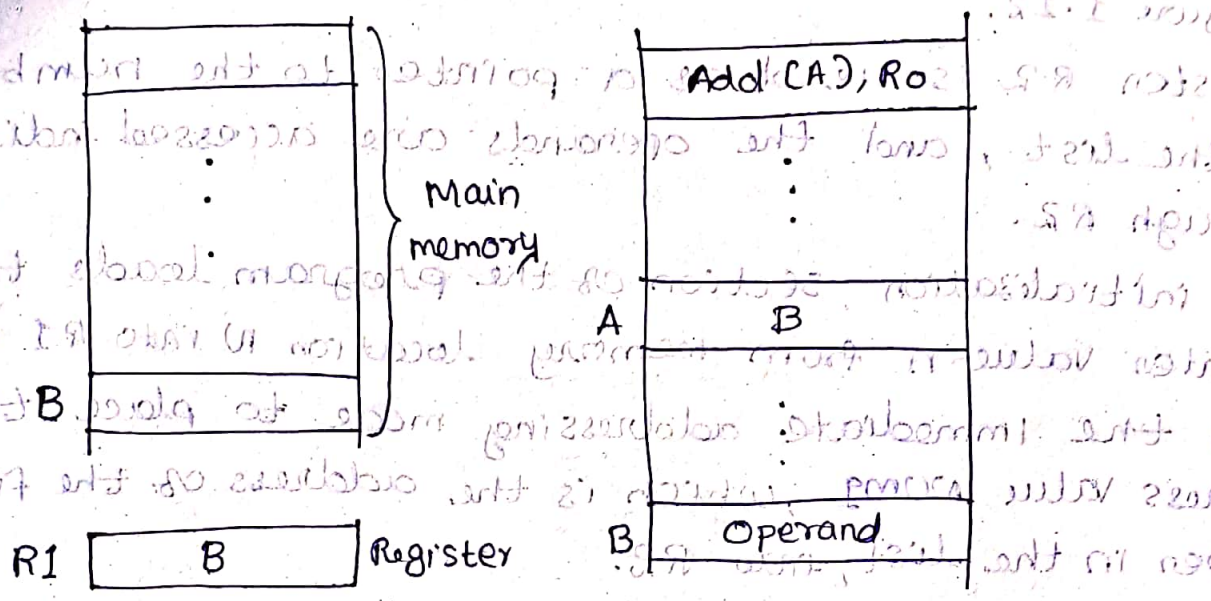
2. INDIRECTION AND POINTERS

Instructions provide information from which the memory address of the operand can be determined. This address is known as the effective address (A) of the operand.

d) Indirect mode - The effective address of the operand is the contents of a register or memory location whose address appears in the instruction.

i) pointer: The register or memory location that contains the address of an operand is called a pointer.

Scanned with CamScanner



(a) Through a general-purpose register (b) Through a memory location.

Figure 1.11 Indirect addressing.

Address	Contents		
	Move	N, R1	Initialization
	Move	#NUM1, R2	
	Clear	R0	
LOOP	Add	(R2), R0	
	Add	#4, R2	
	Decrement	R1	
	Branch>0	LOOP	
	Move	R0, SUM	

Figure 1.12 Use of indirect addressing in the program of Figure 1.10.

i) Indirect addressing can be used to access successive numbers in the list, resulting in the program shown

Scanned with CamScanner

in Figure 1.12.

ii) Register R2 is used as a pointer to the numbers in the list, and the operands are accessed indirectly through R2.

iii) The initialization section of the program loads the counter value n from memory location N into R1 and uses the Immediate addressing mode to place the address value NUM1, which is the address of the first number in the list, into R2.

iv) Clears R0 to 0

Consider the C language statement

$$A = *B;$$

where B is a pointer variable. This statement may be compiled into

```
Move   B, R1
move   (R1), A
```

Using indirect addressing through memory, the same action can be achieved with

```
move (B), A.
```

a) Indirect addressing through memory has proven to be of limited usefulness as an addressing mode.

b) Indirect addressing through registers is used extensively.

c) When absolute addressing is not available, indirect addressing through registers makes it possible to access global variables by first loading

Scanned with CamScanner

the operand's address in a register.

3. INDEXING AND ARRAYS

Different kind of flexibility for accessing operands. It is useful in dealing with lists and arrays.

e) Index mode - The effective address of the operand is generated by adding a constant value to the contents of a register.

The register used may be either a special register provided for this purpose or any one of a set of general-purpose registers in the processor, and it is known as an index register.

eg: $X(R_i)$

$EA = X + [R_i]$

where X is a signed integer

Two ways of using the index mode:

1. the index register, R1, contains the address of a memory location, and the value X defines an offset (also called a displacement) from this address to the location where the operand is found.
2. the constant X corresponds to a memory address and the contents of the index register define the offset to the operand.

In either case, the effective address is the sum of two values; one is given explicitly in the instruction, and the other is stored in a register

Scanned with CamScanner

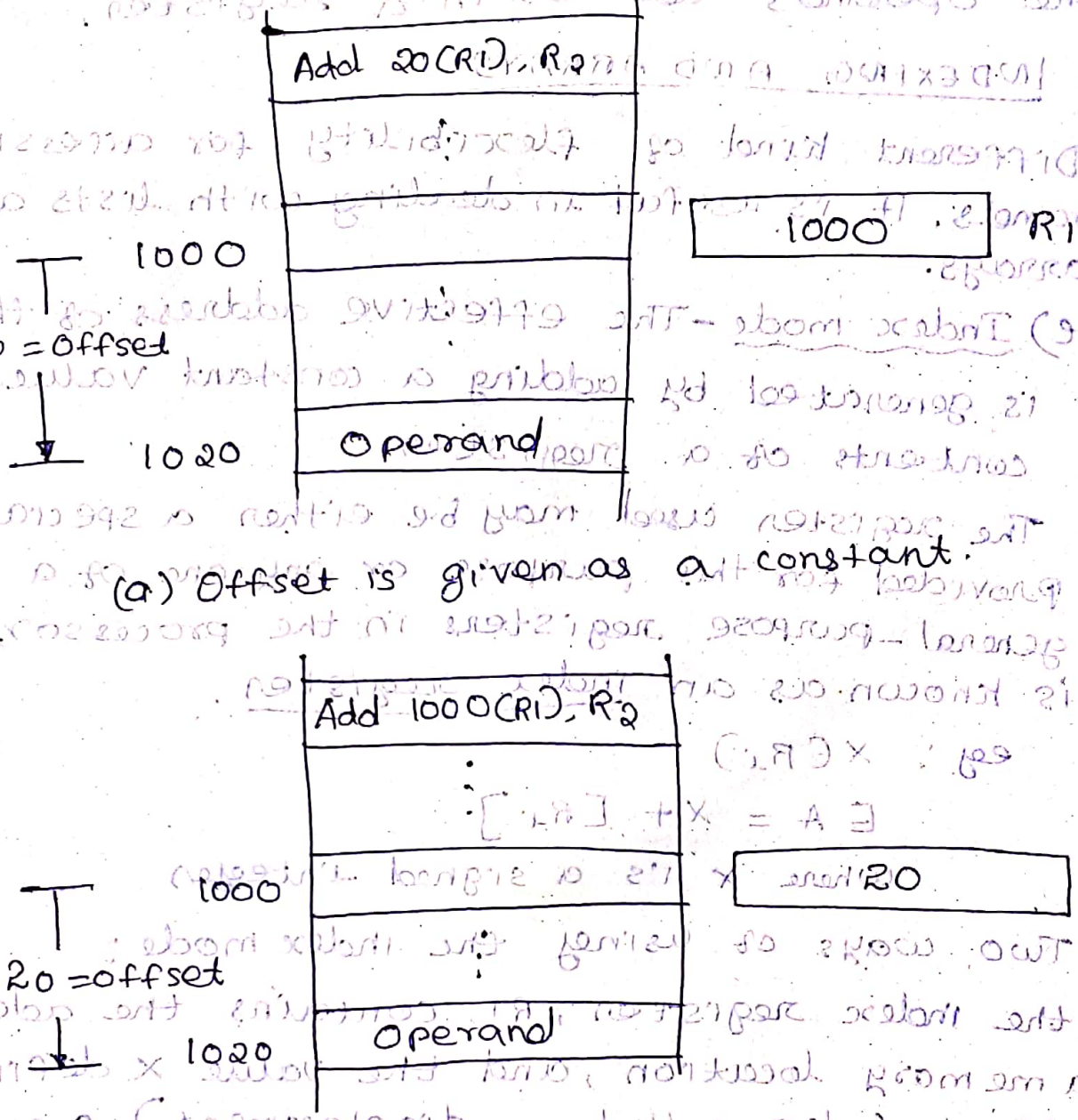


Figure 1.13 Indexed addressing.

Application of indexed addressing:

A list of test scores for students taking a given course is structured in Figure 1.14

Scanned with CamScanner

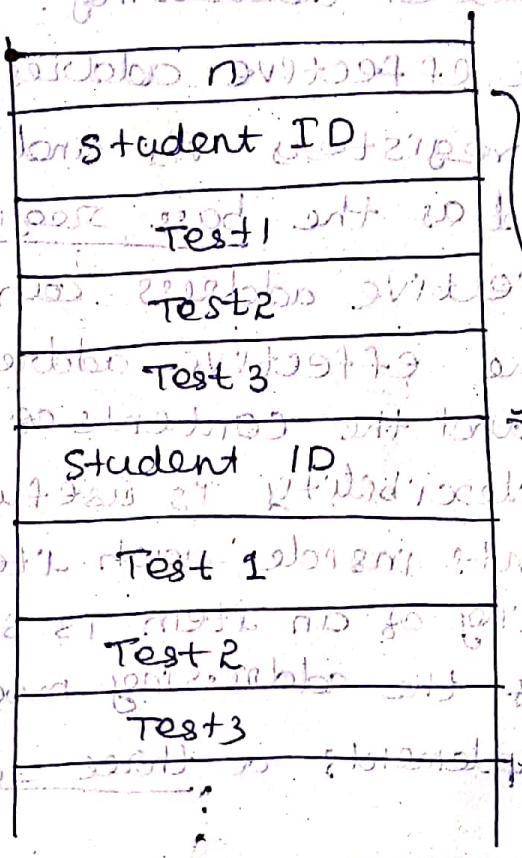


Figure 1.14 A list of students marks.

```
        Move    #LIST, R0
        Clear   R1
        Clear   R2
        Clear   R3
        Move    N, R4
LOOP    Add     1(R0), R1
        Add     8(R0), R2
        Add     12(R0), R3
        Add     #16, R0
        Decrement R4
        Branch>0 LOOP
        Move    R1, SUM1
        Move    R2, SUM2
        Move    R3, SUM3
```

Figure 1.15 Indexed addressing used in accessing test scores in the list in Figure 1.14

Scanned with CamScanner

Variations of indexed addressing:

1. (R_i, R_j) : The effective address is the sum of the contents of registers R_i and R_j. The second register is called as the base register. Both components of the effective address can be changed.

2. $X(R_i, R_j)$: The effective address is the sum of the constant X and the contents of registers R_i and R_j. The added flexibility is useful in accessing multiple components inside each item in a record, where the beginning of an item is specified by the (R_i, R_j) part of the addressing mode. This mode implements a three-dimensional array.

4. RELATIVE ADDRESSING

i) Index mode uses program counter instead of a general-purpose register.
$X(PC) \rightarrow$ X bytes away from the location presently pointed to by the PC.

f) Relative mode:- The effective address is determined by the Index mode using the program counter in place of the general-purpose register R_i. Most common use is to specify the target address in branch instructions.

5. ADDITIONAL MODES

g) Autoincrement mode:- The effective address of the operand is the contents of a register specified in the instruction. After accessing the operand,

Scanned with CamScanner

the contents of this register are automatically incremented to point to the next item in a list.

(Ri)+

Computers that have the Autoincrement mode automatically increment the contents of the register by a value that corresponds to the size of the accessed operand.

b) Autodecrement mode - The contents of a register specified in the instruction are first automatically decremented and are then used as the effective address of the operand.

-(Ri)

Operands are accessed in descending address order.

THE ARM EXAMPLE

1) Advanced RISC Machines (ARM) Limited has designed a family of microprocessors and it licenses the design to other companies for chip fabrication and use in computer products and embedded systems.

2) The main use for ARM microprocessors is in low-power and low-cost embedded applications such as mobile telephones, communication modems, automotive engine management systems, and hand-held digital assistants.

3) All ARM processors share the same basic machine instruction set.

In the ARM architecture:

i) Memory is byte addressable

ii) Using 32-bit addresses and the processor registers

Scanned with CamScanner

are 32 bits long.

iii) Two operand lengths : bytes (8 bits) and words (32 bits).

iv) word addresses must be aligned, i.e multiple of 4.

v) Both little-endian and big-endian memory addressing is supported.

vi) When a byte is transferred then it is always located in the low-order byte position of the register.

vii) Memory is accessed only by Load and Store instructions.

viii) All arithmetic and logic instructions operate only on data in processor registers.

REGISTER STRUCTURE

1) There are 16 32-bit registers (R0 - R15)

R0 - R14 - General purpose registers

R15 - The program counter (PC) register.

2) Current Program Status Register (CPSR) or simply the status register, holds the condition code flags (N, Z, C, V), interrupt disable flags and processor mode bits.

3) There are 15 additional general-purpose registers called the banked registers. (Duplicates of some of the R0 to R14 registers. Used when supervisor or interrupt modes of operation.

4) saved copies of the status register are also available in the nonUser modes.

Scanned with CamScanner

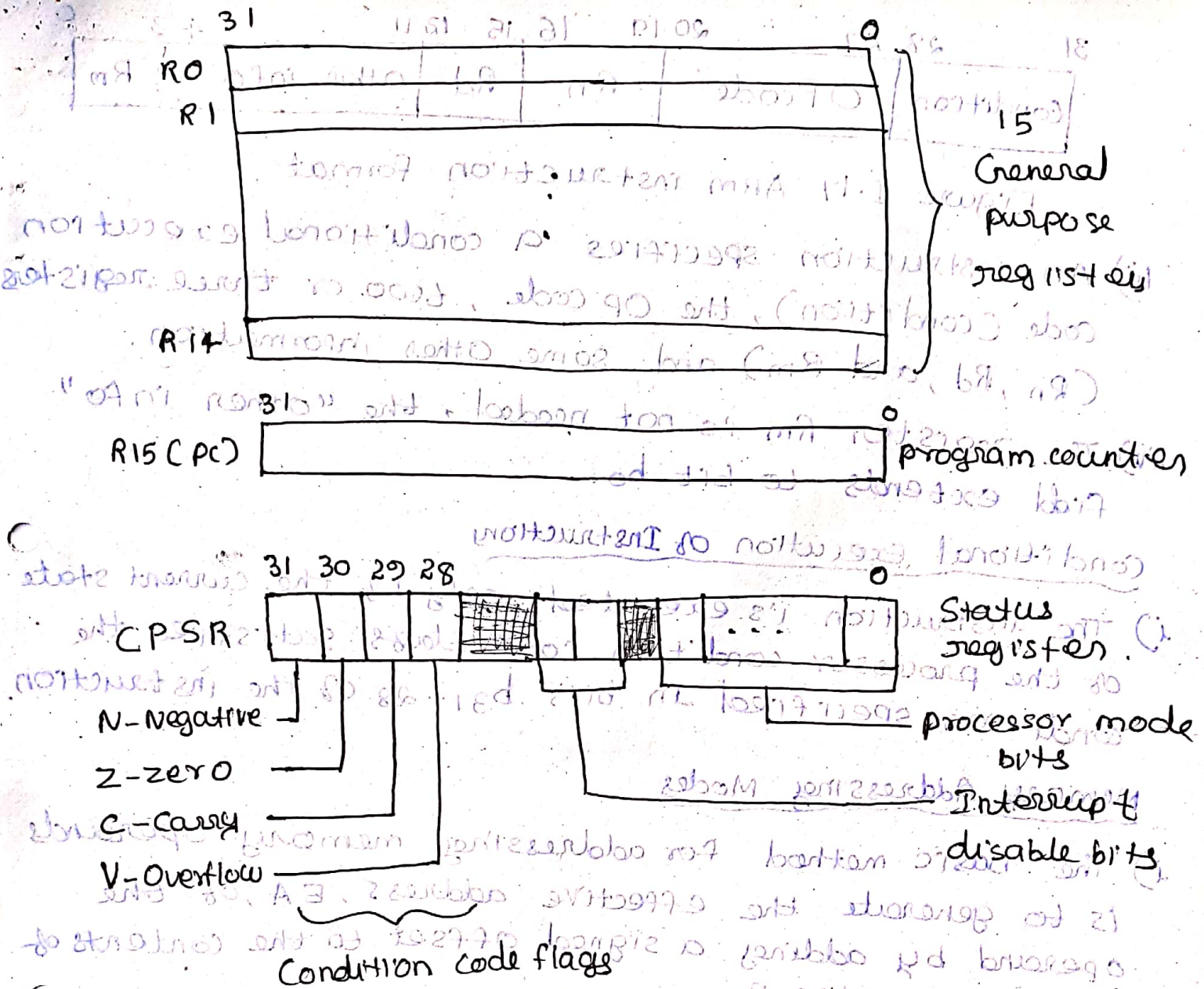


Figure: 1.16 ARM register structure.

MEMORY ACCESS INSTRUCTIONS AND ADDRESSING MODES

i) Each instruction in the ARM architecture is encoded into a 32-bit word in a reasonably uniform way.

ii) The basic encoding format for the ARM instructions is as shown in Figure 1.17.

Scanned with CamScanner

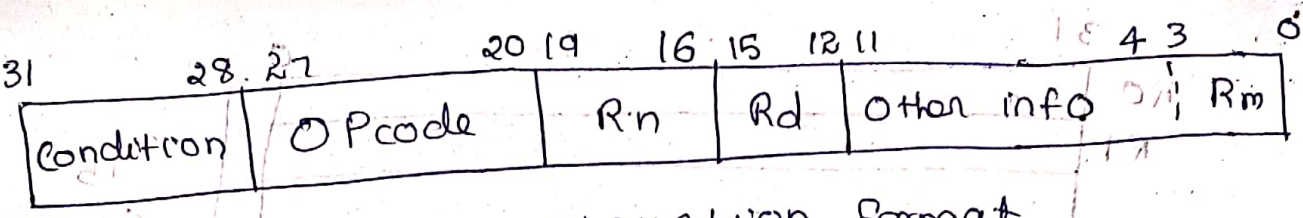


Figure I.17 ARM instruction format.

iii) An instruction specifies a conditional execution code (condition), the OP code, two or three registers (Rn, Rd, and Rm) and some other information. The register Rm is not needed, the "other info" field extends to bit bo.

iv)

Conditional Execution of Instructions

i) The instruction is executed only if the current state of the processor condition code flags satisfies the condition specified in bits $b_{31}-28$ of the instruction.

Memory Addressing Modes

i) The basic method for addressing memory operands is to generate the effective address, EA, of the operand by adding a signed offset to the contents of a base register Rn.

ii) The magnitude of the offset is either an immediate value, contained in the low-order 12 bits of the instruction, or offset is the contents of a third register, Rm, named by the lower-order four bits b_3-0.

iii) The sign (direction) of the offset is contained in the OP-code field.

iv) Access to memory is provided only by Load and Store

Scanned with CamScanner

Instructions: LOAD

LDR Rd, [Rn, #offset] : Rd ← [[Rn] + offset]

LDR Rd, [Rn, Rm] : Rd ← [[Rn] + [Rm]]

LDR Rd, [Rn] : offset of zero, Rd ← [[Rn]]

LDR → specifies that a 32-bit word is loaded from the memory into a register.

LDRB → A byte operand can be loaded into the low-order byte position of a register by using the mnemonic LDRB. Higher order bits are filled with zeros.

STORE

STR Rd, [Rn] : [Rn] ← [Rd]

transferring a word operand into the memory.

The STRB instruction transfers the byte contained in the low-order end of Rd.

Addressing modes

1. Pre-indexed mode - The effective address of the operand is the sum of the contents of the base register Rn and an offset value.
2. Pre-indexed with writeback mode - The effective address of the operand is generated in the same way as in the pre-indexed mode, and then the effective address is written back into Rn.
3. Post-indexed mode - The effective address of the operand is the contents of Rn. The offset is then added to this

Scanned with CamScanner

address and the result is written back into Rn.

i) The exclamation mark signifies writeback in the pre-indexed addressing mode.

ii) The post-indexed mode always involves write back,

iii) post-indexing is the generalized form of the AutoInc ~ rement addressing mode.

iv) pre-indexing with write back is a generalization of the autoincrement addressing mode.

v) In all three addressing modes, the offset may be given as an immediate value in the range ± 4095.

LDR R0, [R1, -R2]! R0 ← [[R1] - [R2]]

The effective address of the operand [R1] - [R2] is then loaded into R1 because write back is specified by the exclamation mark.

vi) When the offset is given in a register, it may be scaled by a power of 2 by shifting to the right or to the left. The amount of the shift is specified by an immediate value in the range 0 to 31.

LDR R0, [R1, -R2, LSL #4]!

R0 ← [[R1] - 16 X [R2]]

and will then load the effective address into R1.

vii) The program Counter, PC, may be used as the Base Register Rn. The assembler determines the immediate offset as the signed distance between the address of the operand and the contents of the updated PC.

Scanned with CamScanner

Table 1.1 ARM indexed addressing modes.

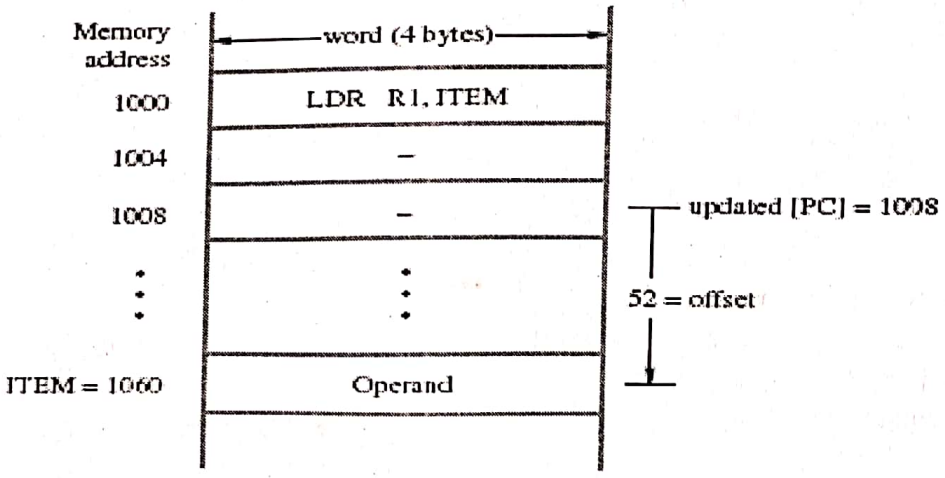
Name	Assembler syntax	Addressing function
With immediate offset:		
Pre-indexed	[R*n*, #offset]	EA = [R*n*] + offset
Pre-indexed with writeback	[R*n*, #offset]!	EA = [R*n*] + offset; R*n* ← [R*n*] + offset
Post-indexed	[R*n*], #offset	EA = [R*n*]; R*n* ← [R*n*] + offset
With offset magnitude in R*m*:		
Pre-indexed	[R*n*, ± R*m*, shift]	EA = [R*n*] ± [R*m*] shifted
Pre-indexed with writeback	[R*n*, ± R*m*, shift]!	EA = [R*n*] ± [R*m*] shifted; R*n* ← [R*n*] ± [R*m*] shifted
Post-indexed	[R*n*], ± R*m*, shift	EA = [R*n*]; R*n* ← [R*n*] ± [R*m*] shifted
Relative (Pre-indexed with immediate offset)	Location	EA = Location = [PC] + offset

EA = effective address
offset = a signed number contained in the instruction
shift = direction #integer
where direction is LSL for left shift or LSR for right shift; and
integer is a 5-bit unsigned number specifying the shift amount
±R*n* = the offset magnitude in register R*m* can be added to or subtracted from the contents of base register R*n*

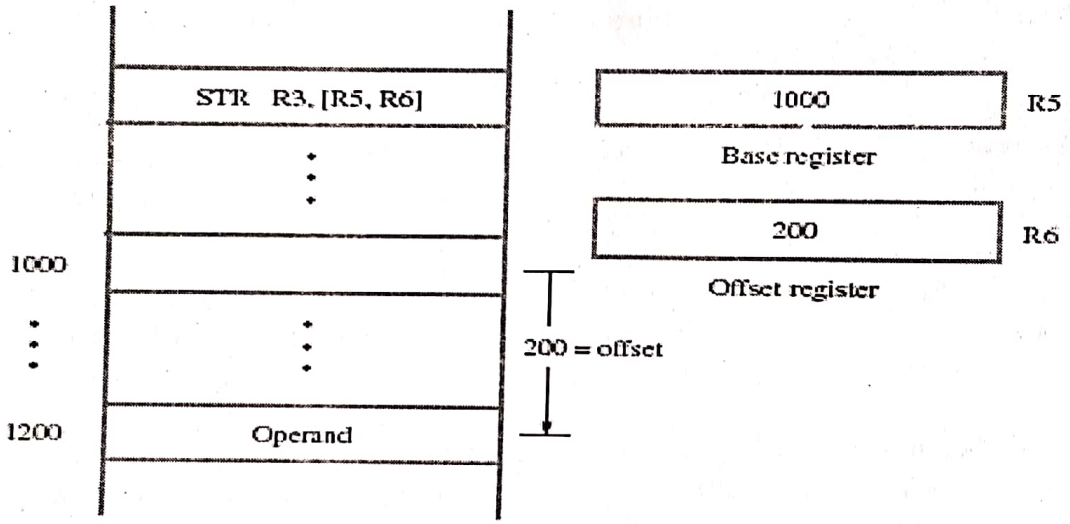
An example of the Relative mode is shown in Figure 1.17. The address of the operand, given symbolically in the instruction as ITEM, is 1060. There is no Absolute addressing mode available in the ARM architecture, other than the limited form described in the previous section. Therefore, when the address of a memory location is specified by placing an address label in the operand field, the assembler uses the Relative addressing mode. This is implemented by the Pre-indexed mode with an immediate offset, using PC as the base register. As shown in the figure, the offset calculated by the assembler is 52, because the updated PC will contain 1008 when the offset is added to it during program execution. The effective address generated by this instruction is 1060 = 1008 + 52. The operand must be within a distance of 4095 bytes forward or backward from the updated PC. If the operand address is outside this range, an error is indicated by the assembler and a different addressing mode must be used to access the operand.

Scanned with CamScanner

Figure 1.17b shows an example of the Pre-indexed mode with the offset contained in register R6 and the base value contained in R5. The Store instruction (STR) stores the contents of R3 into the word at memorylocation 1200.



(a) Relative addressing mode



(b) Pre-indexed addressing mode

Figure 1.17 Examples of memory addressing modes.

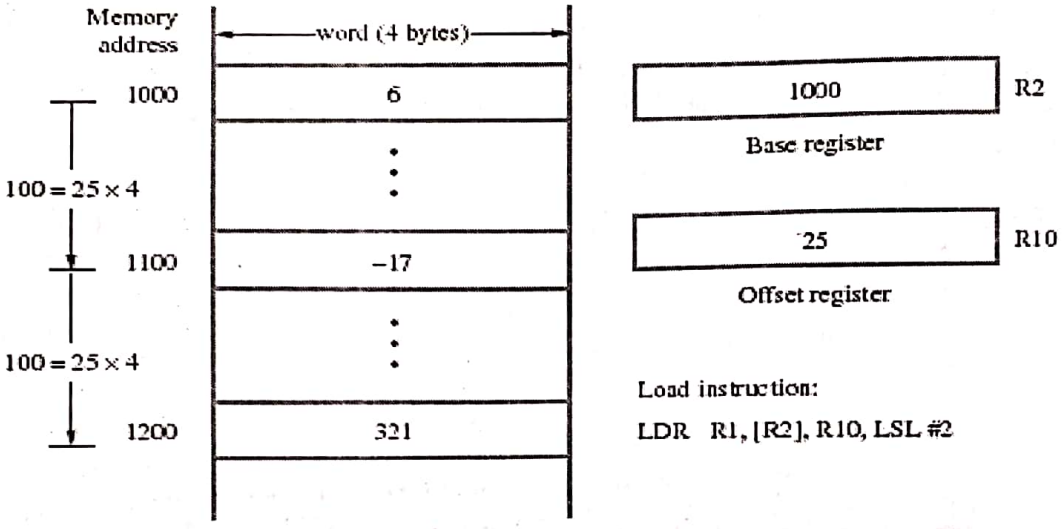
The examples shown in Figure 1.17 illustrate the usefulness of the writeback feature in the Post-indexed and Pre-indexed addressing modes. Figure 1.17*a* shows the first three numbers of a list of 25 numbers, starting at memory address 1000 and spaced 25 words apart. They comprise the first row of a 25 × 25 matrix of numbers that is stored in column order. Memory locations 1000, 1004, 1008, . . . , 1096 contain the first column of the matrix. The first number of the first row of the matrix is stored in word location 1000, and the numbers at addresses 1100, 1200, . . . , 3400 are the successive numbers of the first row. The numbers in the first row of the matrix can be accessed conveniently in a program loop by using the Post-indexed addressing mode, with the offset contained in a register. Suppose that R2 is

Scanned with CamScanner

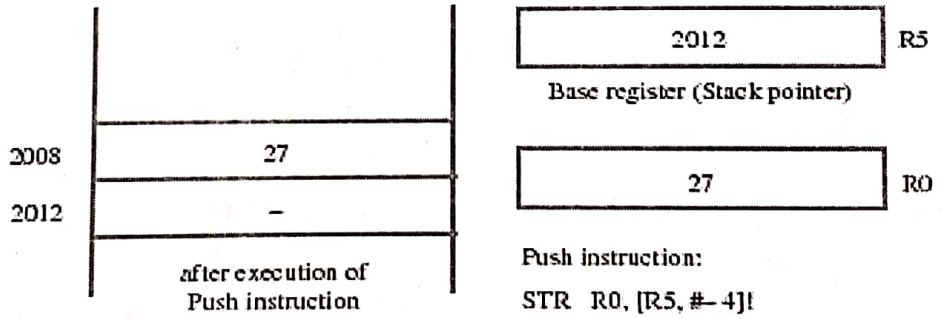
used as the base register and that it contains the initial address value 1000. Suppose also that register R10 is used to hold the offset, and that it is loaded with the value 25. The instruction

LDR R1, [R2], R10, LSL #2

can be used in the body of a program loop to load register R1 with successive elements of the first row of the matrix in successive passes through the loop.



(a) Post-indexed addressing



(b) Pre-indexed addressing with writeback

Figure 1.18 Memory addressing modes involving writeback.

ARM **Instructions**

1. Load and Store Instructions
2. Arithmetic Instructions
3. Move Instructions
4. Logic and Test Instructions
5. Compare Instructions

Scanned with CamScanner

6. Setting Condition Code Flags
7. Branch Instructions
8. Subroutine Linkage Instructions

1. Load and Store Instructions

Byte and half-word values can also be transferred between memory and registers. If the operand is a byte, it is located in the low-order byte position of the register. If the operand is a half word, it is located in the low-order half of the register. For Load instructions, byte and half-word values are zero-extended to the 32-bit register length by using the instruction mnemonics LDRB and LDRH or are sign-extended by using LDRSB and LDRSH. The byte and half-word Store instructions have the mnemonics STRB and STRH.

Loading and Storing Multiple Operands

There are two instructions for loading and storing multiple operands. They are called Block Transfer instructions. Any subset of the general-purpose registers can be loaded or stored. Only word operands are allowed. The OP codes used are LDM (Load Multiple) and STM (Store Multiple). The memory operands must be in successive word locations. As an example, assume that register R10 is the base register and that it contains the value 1000 initially. The instruction

LDMIA R10!, {R0, R1, R6, R7}

transfers the words from locations 1000, 1004, 1008, and 1012 into registers R0, R1, R6, and R7, leaving the address value 1016 in R10 after the last transfer, because writeback is indicated by the exclamation character. The suffix IA in the OP code indicates "Increment After," corresponding to post-indexing.

2. Arithmetic Instructions

The basic assembly-language format for arithmetic instructions is

OP Rd, Rn, Rm

Addition and Subtraction

ADD R0, R2, R4	performs the operation	R0←[R2] + [R4]
SUB R0, R6, R5	performs the operation	R0←[R6] − [R5]
ADD R0, R3, #17	performs the operation	R0←[R3] + 17
ADD R0, R3, #−17	is equivalent to	SUB R0, R3, #17

Shifting or Rotation of the Second Source Operand

The instruction

ADD R0, R1, R5, LSL #4

is executed as follows. The second source operand, which is contained in register R5, is shifted left 4 bit positions (equivalent to [R5] × 16), then added to the contents of register R1. The sum is placed in register R0. The shift or rotation amount can also be specified as the contents of a fourth register.

Multiple-Word Operands

Scanned with CamScanner

The carry flag, C, can be used to facilitate addition and subtraction operations that involve multiple-word numbers. Separate instructions are available for this purpose. Their assembly language mnemonics are ADC (Add with carry) and SBC (Subtract with carry).

64-bit operands can be added by using the instruction

ADDS R6, R2, R4

followed by the instruction

ADC R7, R3, R5

producing the 64-bit sum in the register pair R7, R6. The carry output from the ADDS operation is used as a carry input in the ADC operation to execute the 64-bit addition. The S suffix on the ADD instruction is needed to set the C flag.

Multiplication

The instruction MUL R0, R1, R2 performs the operation R0←[R1] × [R2]
The instruction MLA R0, R1, R2, R3 performs the operation R0←([R1] × [R2]) + [R3]
This is called a Multiply-Accumulate operation. It is often used in signal-processing applications.

3. Move Instructions

The Move instruction MOV R*d*, R*m* copies the contents of register R*m* into register R*d*. The instruction MOV R*d*, #value loads an 8-bit immediate value into the destination register.

A second version of the Move instruction, called Move Negative, with the OP-code mnemonic MVN, forms the bit-complement of the source operand before it is placed in the destination register. Recall that an 8-bit immediate value is an unsigned number in the range 0 to 255. TheMVNinstruction can be used to load negative values in 2's-complement representation as follows. Suppose we wish to load −5 into register R0. The instruction

MVN R0, #4

achieves the desired result because the bit-complement of 4 is the 2's-complement representation for −5.

4. Logic and Test Instructions

The logic operations AND, OR, XOR, and Bit-Clear are implemented by instructions with the OP codes AND, ORR, EOR, and BIC, respectively. They have the same format as arithmetic instructions. The instruction

AND Rd, Rn, Rm

performs a bitwise logicalAND of the operands in registers Rn and Rm and places the result in register Rd. For example, if register R0 contains the hexadecimal pattern 02FA62CA and R1 contains the pattern 0000FFFF, then the instruction

AND R0, R0, R1

will result in the pattern 000062CA being placed in register R0.

Scanned with CamScanner

The Bit Clear instruction, BIC, is closely related to the AND instruction. It complements each bit in operand Rm before ANDing them with the bits in register Rn. Using the same R0 and R1 bit patterns as in the above example, the instruction

BIC R0, R0, R1

results in the pattern 02FA0000 being placed in R0.

Instructions called Test (TST) and Test Equivalence (TEQ) perform the logical AND and XOR operations, respectively, on their word operands, then set condition code flags based on the result. They do not store the result in a register. These instructions can be used to test how an unknown bit pattern matches up against a known bit pattern, and can then be followed by a Branch instruction that is conditioned on the result.
For example, the Test instruction

TST R*n*, #1

performs anAND operation to test whether the low-order bit of register R*n* is equal to 1. If the result of the test is positive, that is, if the low-order bit of the contents of register R*n* is equal to 1, then the result of theAND operation is 1, and the Z bit is cleared to zero. Status bits in I/O device registers can be checked with this type of instruction.
The Test Equivalence instruction

TEQ R*n*, #5

performs an XOR operation to test whether register R*n* contains the value 5. If it does, then the result of the bit-by-bit XOR operation will be zero, and the Z bit will be set to 1.

5. Compare Instructions

The Compare instruction CMP Rn, Rm performs the operation [Rn] − [Rm] and sets the condition code flags based on the result of the subtraction operation. The result itself is discarded. The Compare Negative instruction CMN Rn, Rm performs the operation [Rn] + [Rm] and sets the condition code flags based on the result of the operation. The result of the operation is discarded.

6. Setting Condition Code Flags

The Compare and Test instructions always update the condition code flags. The arithmetic, logic, and Move instructions affect the condition code flags only if explicitly specified to do so by a bit in the OP-code field. This is indicated by appending the suffix S to the assembly language OP-code mnemonic. For example, the instruction

ADDS R0, R1, R2

sets the condition code flags, but

ADD R0, R1, R2

does not.

7. Branch Instructions

Conditional branch instructions contain a 24-bit 2's-complement value that is used to generate a branch offset as follows. When the instruction is executed, the value in the instruction is shifted left two bit positions (because all branch target addresses are word-aligned), then sign-extended to 32 bits to generate the offset. This offset is added to the updated contents

Scanned with CamScanner

of the Program Counter to generate the branch target address.

8. Subroutine Linkage Instructions

The Branch and Link (BL) instruction is used to call a subroutine. It operates in the same way as other branch instructions, with one added step. The return address, which is the address of the next instruction after the BL instruction, is loaded into register R14, which acts as the link register. Since subroutines may be nested, the contents of the link register must be saved on the processor stack before a nested call to another subroutine is made. Register R13 is used as the processor stack pointer.

Basic Input/Output

The components of a computer system communicate with each other through an interconnection network, as shown in Figure 3.1. The interconnection network consists of circuits needed to transfer information between the processor, the memory unit, and a number of I/O devices.

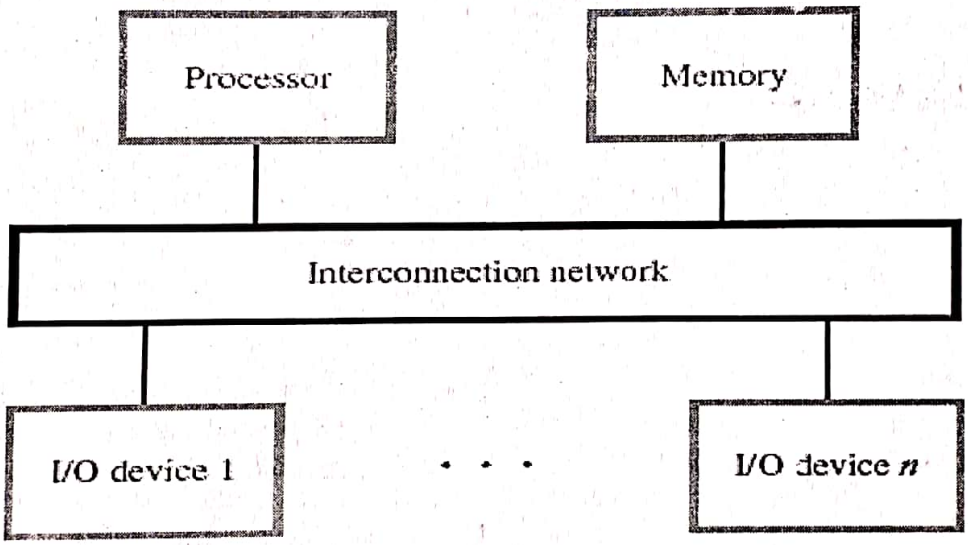


Figure 1.19 A computer system.

Scanned with CamScanner

BASIC INPUT/OUTPUT OPERATIONS

Input/output (I/O) operations are essential to transfer data between the memory of a computer and the outside world. The different basic input/output operations include

1. program-controlled I/O
2. Memory-mapped I/O
3. Interrupt Driven
4. Direct Memory Access

1. Program-controlled I/O.

i) keyboard to a computer – limited speed (characters per second)

ii) computer to display – much higher. (several thousand characters per second).

iii) Processor executes in the speed of many millions of instructions per second.

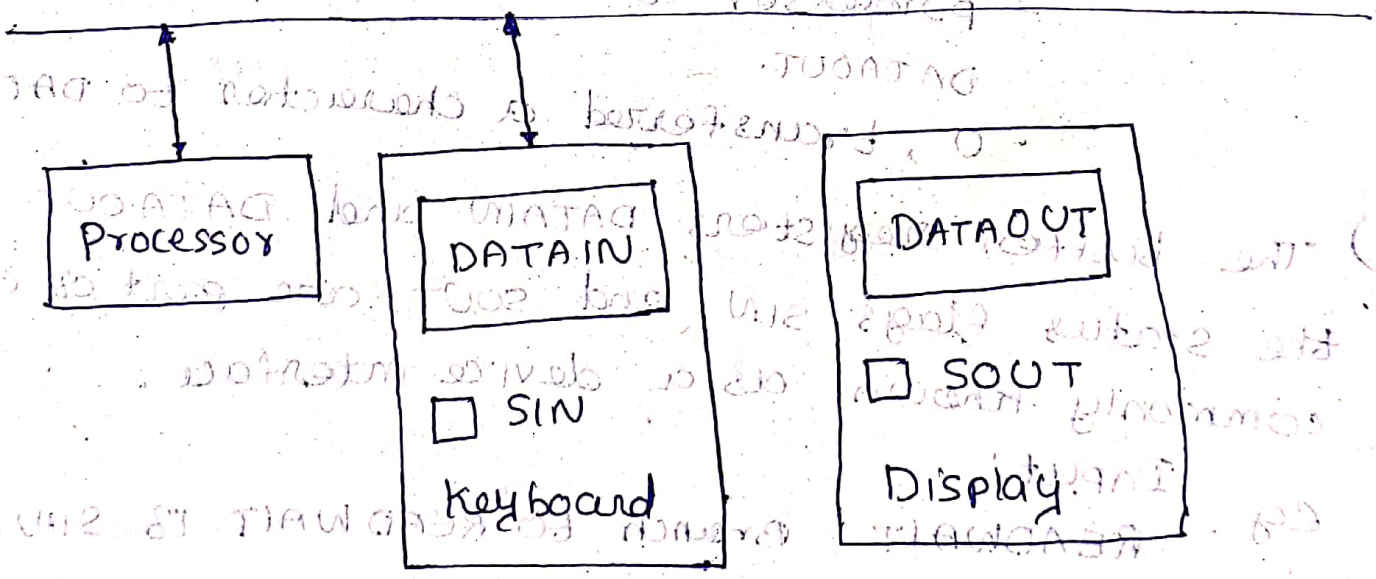


Figure 1.20 BUS connection for processor, keyboard, and display.

Scanned with CamScanner

iv) Programs can be used to synchronise the speed between the processor and I/O devices.

v) One strike of a key on the keyboard causes one block of instructions in the I/O programs transfers the character into the processor, and another associated block of instructions causes the character to be displayed.

DATAIN – 8-bit buffer register associated with the keyboard.

SIN – a status control flag
= 1 indicates DATAIN contains a valid character,
0 otherwise.

DATAOUT – A buffer register in the display

SOUT – 1, display is ready to receive a character.
– processor transfers a character code to DATAOUT.
– 0, transferred a character to DATAOUT

vi) The buffer registers DATAIN and DATAOUT and the status flags SIN and SOUT are part of circuitry commonly known as a device interface.

Eg:
```
Input:
READWAIT   Branch to READWAIT if SIN=0
           Input from DATAIN to R1.
```

The first instruction tests the status flag and the second performs the branch.

Scanned with CamScanner

Output

Eg: WRITEWAIT Branch to WRITEWAIT if SOUT=0
Output from R1 to DATAOUT.

1. CPU has direct control over I/O
   - Sensing status
   - Read/write comands
   - Transferring data.
2. CPU waits for I/O module to complete operation.
3. Wastes CPU time.

The process of periodically checking status bits to see if it is time for the next I/O operation, is called Polling.

If SIN and SOUT can be stored in device status registers in the 3rd bit b3 then Figure 1.21 shows a program to read a line of characters and displays it.

	Move	#LOC, R0	R0 ← LOC (First location)
READ	TestBit	#3, INSTATUS	wait for a character to be entered in the key board buffer DATAIN.
	Branch=0	READ	
	MoveByte	DATAIN, (R0)	R Transfer from DATAIN to memory
ECHO	TestBIT	#3, OUTSTATUS	wait for the display
	Branch=0	ECHO	
	MoveByte	(R0), DATAOUT	character to display buffer register
	Compare	#CR, (R0)+	If carriage return, end of line
	Branch≠0	READ	Read another character.

Figure 1.21 A program that reads a line of characters and displays it.

Scanned with CamScanner

2. Memory-mapped I/O

i) Some memory address values are used to refer to peripheral device buffer register such as DATAIN and DATAOUT.

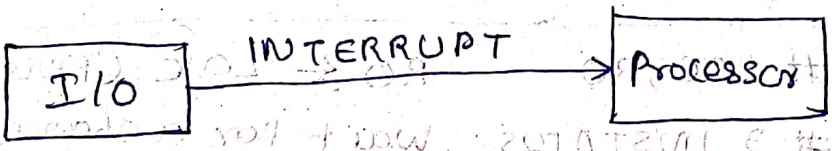
ii) I/O devices and the memory share the same address space, the arrangement is called memory-mapped I/O.

iii) In memory-mapped I/O portions of address space are assigned to I/O devices and reads and writes to these addresses are interpreted as commands to the I/O device.

iv) "DATAIN" is the address of the input buffer associated with the keyboard.

- Move DATAIN, R0
- Move R0, DATAOUT

3. Interrupt-Driven I/O



i) When I/O device is ready, it sends the INTERRUPT signal to processor via a dedicated controller line.

ii) Using interrupt we are ideally eliminating WAIT period.

iii) In response to the interrupt, the processor executes the interrupt service Routine (ISR).

iv) All the registers, flags, program counter values

Scanned with CamScanner

are saved by the processor before running ISR.

v) The time required to save status & restore contribute to execution overhead → "Interrupt Latency"

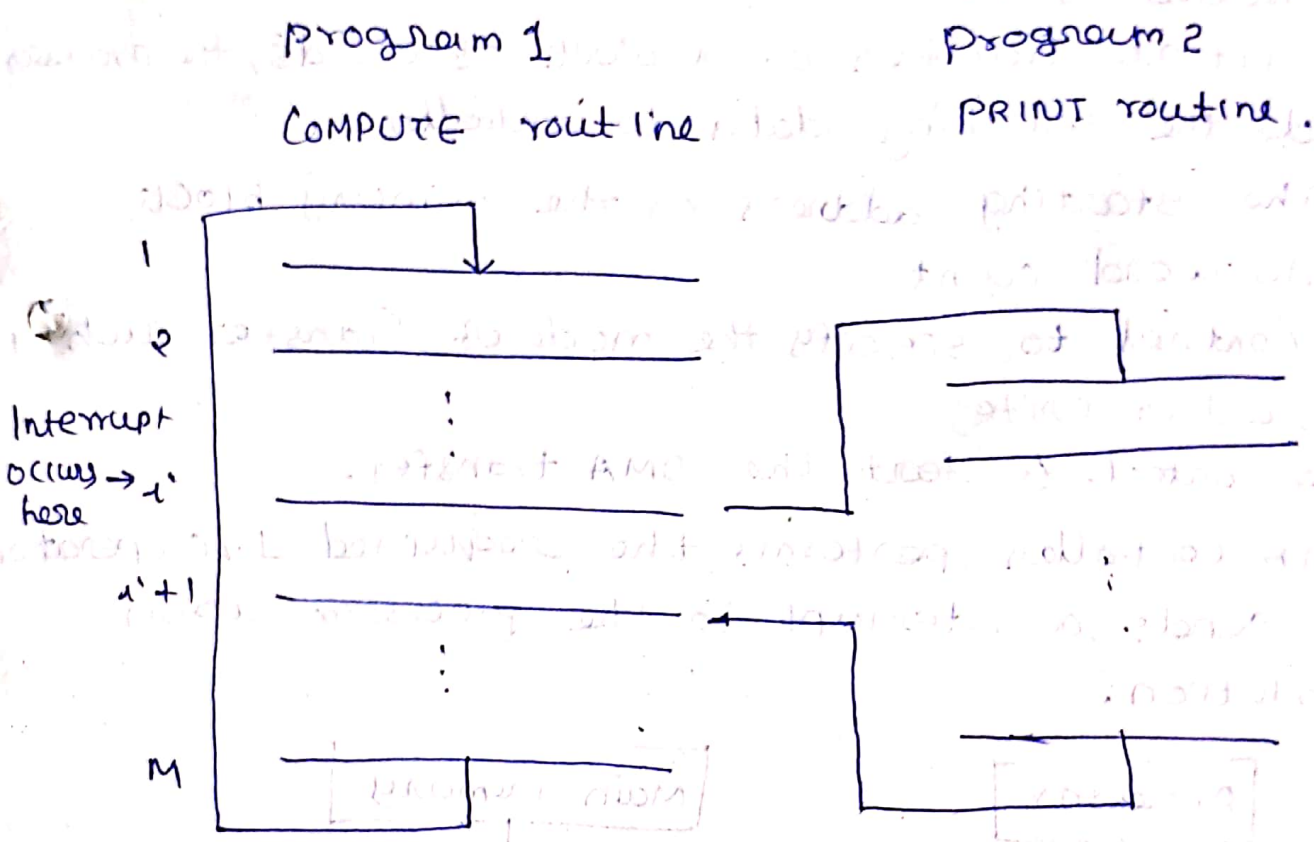


Figure 1: 2R Interrupt

vi) ISR is a sub-routine - may belong to a different user than the one being executed and then halted.

vii) The condition code flags and the contents of any registers used by both the interrupted program and the interrupt-service routine are saved and restored.

Scanned with CamScanner

4. Direct Memory Access (DMA)

i) For huge I/O transfer is required, DMA is used.

ii) DMA controller acts as a processor, but it is controlled by CPU.

iii) To initiate transfer of a block of words, the processor sends the following data to controller.

1. The starting address of the memory block
2. The word count
3. Control to specify the mode of transfer such as read or write
4. A control to start the DMA transfer.

iv) DMA controller performs the required I/O operation and sends a interrupt to the processor upon completion.

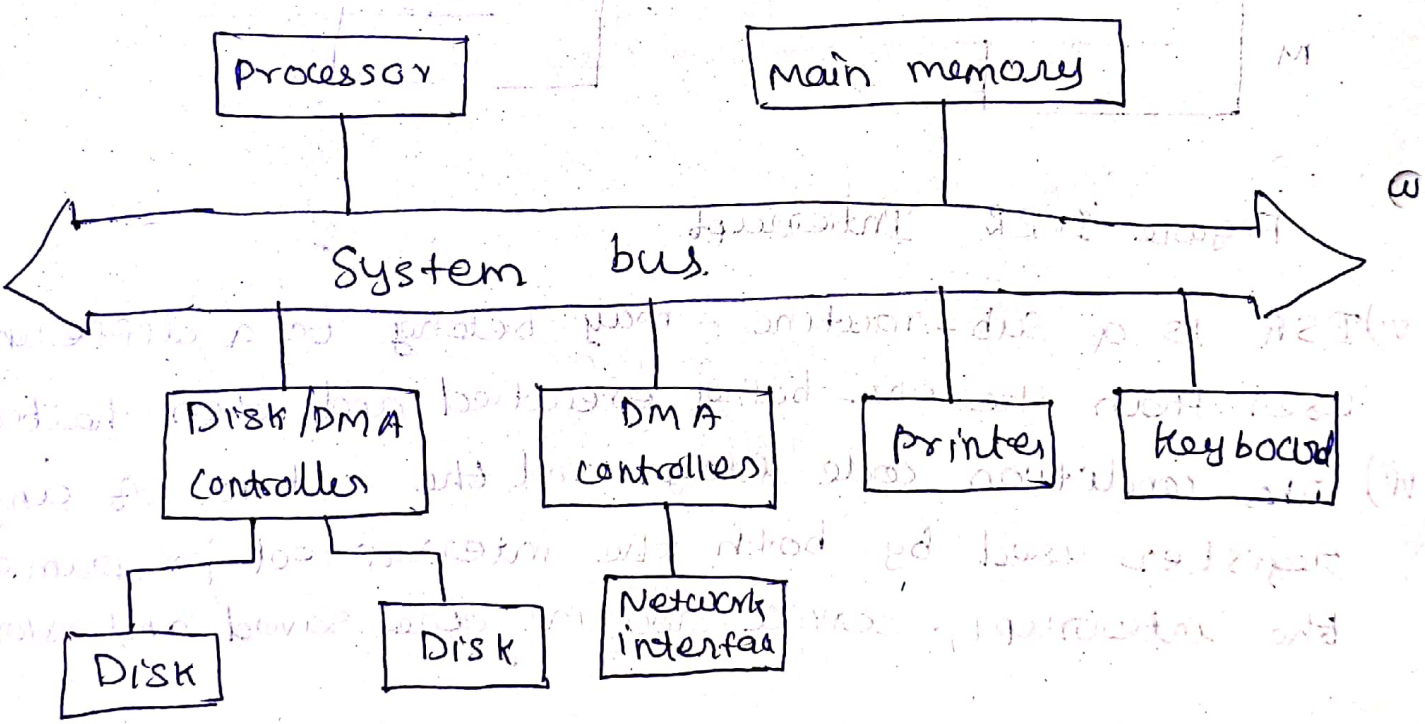


Figure 1.23 Use of DMA Controller in a computer system.

Scanned with CamScanner

v) Memory accesses by the processor and DMA Controller are interwoven.

vi) DMA devices have higher priority than processer over BUS control.

vii) The DMA controller may given exclusive access to the main memory to transfer a block of data without interruption is Block or Burst mode.

STACK

i) Inorder to organize the control and information linkage between the main program and the subroutine, a data structure called a stack is used.

ii) A stack is a list of data elements, words or bytes.

iii) Elements can be added or removed at one end of the list only which is called the top of the stack.

iv) The structure is referred to as a pushdown stack.

v) A processor register is used to keep track of the address of the element of the stack that is at the top at any given time. This register is called the stack pointer (SP).

Scanned with CamScanner

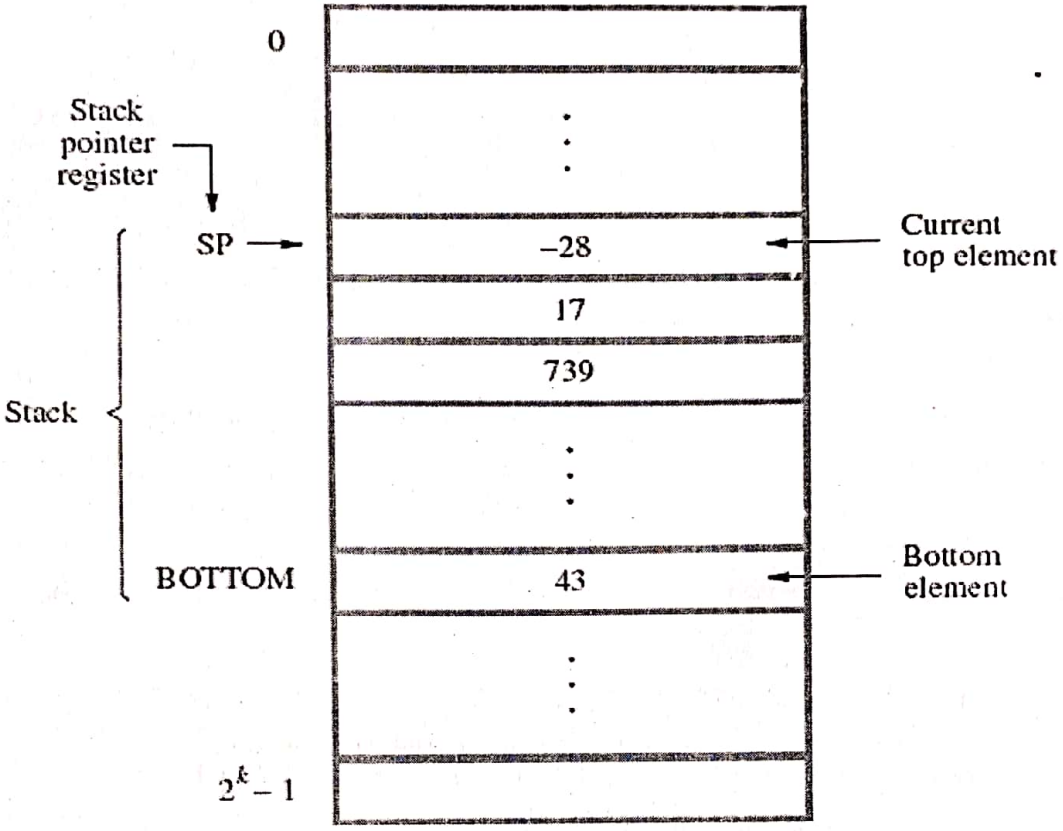


Figure 1.24 A stack of words in the memory

SUBROUTINES

In a given program, it is often necessary to perform a particular task many times on different data values. It is prudent to implement this task as a block of instructions that is executed each time the task has to be performed. Such a block of instructions is usually called a *subroutine*.

When a program branches to a subroutine we say that it is calling the subroutine. The instruction that performs this branch operation is named a Call instruction.

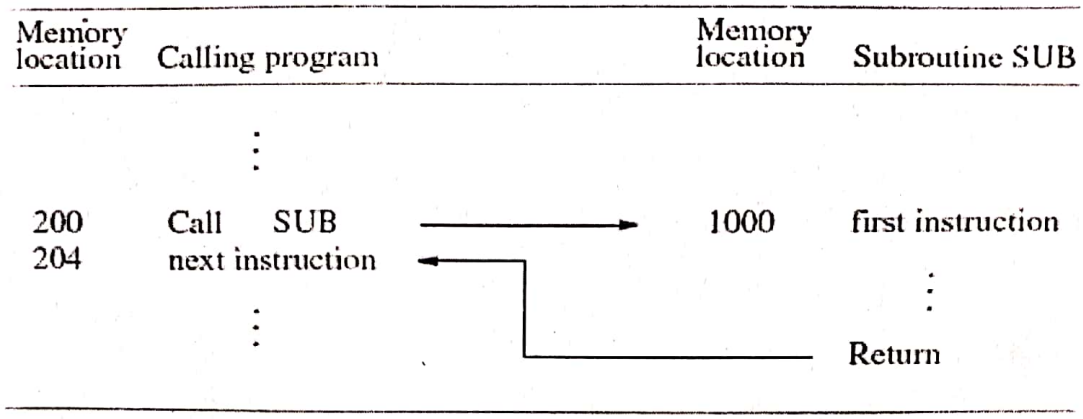
After a subroutine has been executed, the calling program must resume execution, continuing immediately after the instruction that called the subroutine. The subroutine is said to return to the program that called it.

The contents of the PC must be saved by the Call instruction to enable correct return to the calling program.

The way in which a computer makes it possible to call and return from subroutines is referred to as its subroutine linkage method. The simplest subroutine linkage method is to save the return address in a

Scanned with CamScanner

specific location, which may be a register dedicated to this function. Such a register is called the link register.



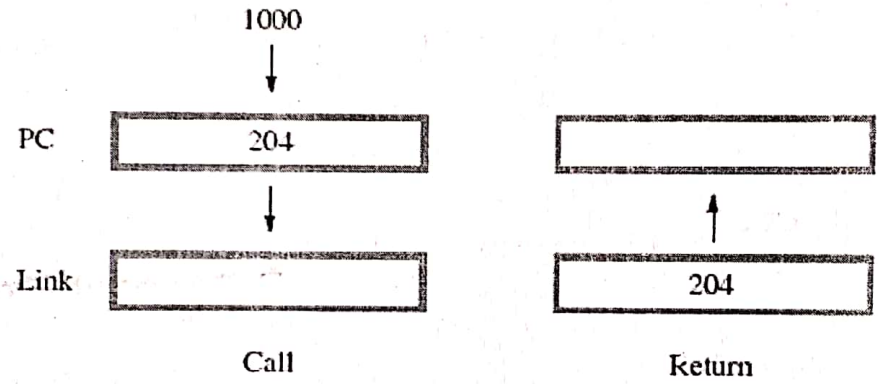


Figure 1.25 Subroutine linkage using a link register

The Call instruction is just a special branch instruction that performs the following operations:

- Store the contents of the PC in the link register
- Branch to the target address specified by the Call instruction

The Return instruction is a special branch instruction that performs the operation

- Branch to the address contained in the link register.

Subroutine Nesting and the Processor Stack

Acommon programming practice, called subroutine nesting, is to have one subroutine call another. In this case, the return address of the second call is also stored in the link register, overwriting its previous contents. Hence, it is essential to save the contents of the link register in some other location before calling another subroutine. Otherwise, the return address of the first subroutine will be lost.

The return addresses associated with subroutine calls should be pushed onto **the processor stack.**

Scanned with CamScanner

Calling program

```
                Load        R2, N          Parameter 1 is list size.
                Move        R4, #NUM1      Parameter 2 is list location.
                Call        LISTADD        Call subroutine.
                Store       R3, SUM        Save result.
                ⋮
```

Subroutine

```
LISTADD:        Subtract    SP, SP, #4     Save the contents of
                Store       R5, (SP)         R5 on the stack.
                Clear       R3             Initialize sum to 0.
LOOP:           Load        R5, (R4)       Get the next number.
                Add         R3, R3, R5     Add this number to sum.
                Add         R4, R4, #4     Increment the pointer by 4.
                Subtract    R2, R2, #1     Decrement the counter.
                Branch_if_[R2]>0 LOOP
                Load        R5, (SP)       Restore the contents of R5.
                Add         SP, SP, #4
                Return                     Return to calling program.
```

Figure 1.26 Program written as a subroutine; parameters passed through registers.

Parameter Passing

The exchange of information between a calling program and a subroutine is referred to as *parameter passing.*
The parameters may be placed in registers or in memory locations, where they can be accessed by the subroutine. Alternatively, the parameters may be placed on the processor stack.

Special instructions for loading and storing multiple registers

StoreMultiple R2–R5, –(SP)

The source registers are specified by the range R2–R5. The notation –(SP) specifies that the stack pointer must be adjusted accordingly. The minus sign in front indicates that SP must be decremented (by 4) before the contents of each register are placed on the stack.
Similarly, the instruction

LoadMultiple R2–R5, (SP)+

Scanned with CamScanner

will load registers R2, R3, R4, and R5, in reverse order, with the values that were saved on the stack. The notation (SP)+ indicates that the stack pointer must be incremented (by 4) after each value has been loaded into the corresponding register.

Parameter Passing by Value and by Reference

Note the nature of the two parameters, NUM1 and *n*, passed to the subroutines in Figures 1.26. The purpose of the subroutines is to add a list of numbers. Instead of passing the actual list entries, the calling program passes the address of the first number in the list. This technique is called passing by reference. The second parameter is passed by value, that is, the actual number of entries, n, is passed to the subroutine.

THE STACK FRAME

During execution of the subroutine, some locations at the top of the stack contain entries that are needed by the subroutine. These locations constitute a private work space for the subroutine, allocated at the time the subroutine is entered and deallocated when the subroutine returns control to the calling program. Such space is called a ***stack frame.***

In addition to the stack pointer SP, it is useful to have another pointer register, called the ***frame pointer*** (FP), for convenient access to the parameters passed to the subroutine and to the local memory variables used by the subroutine.

With the FP register pointing to the location just above the stored parameters, as shown in Figure 1.27, we can easily access the parameters and the local variables by using the Index addressing mode. The parameters can be accessed by using addresses 4(FP), 8(FP), ... The local variables can be accessed by using addresses −4(FP), −8(FP), ... The contents of FP remain fixed throughout the execution of the subroutine, unlike the stack pointer SP, which must always point to the current top element in the stack.

SP points to the old top-of-stack (TOS) element in Figure 1.27. Before the subroutine is called, the calling program pushes the four parameters onto the stack. Then the Call instruction is executed. At this time, SP points to the last parameter that was pushed on the stack. If the subroutine is to use the frame pointer, it should first save the contents of FP by pushing them on the stack, because FP is usually a general-purpose register and it may contain information of use to the calling program. Then, the contents of SP, which now points to the saved value of FP, are copied into FP.

Scanned with CamScanner

pointer, it should first save the contents of FP by pushing them on the stack, because FP is usually a general-purpose register and it may contain information of use to the calling program. Then, the contents of SP, which now points to the saved value of FP, are copied into FP.

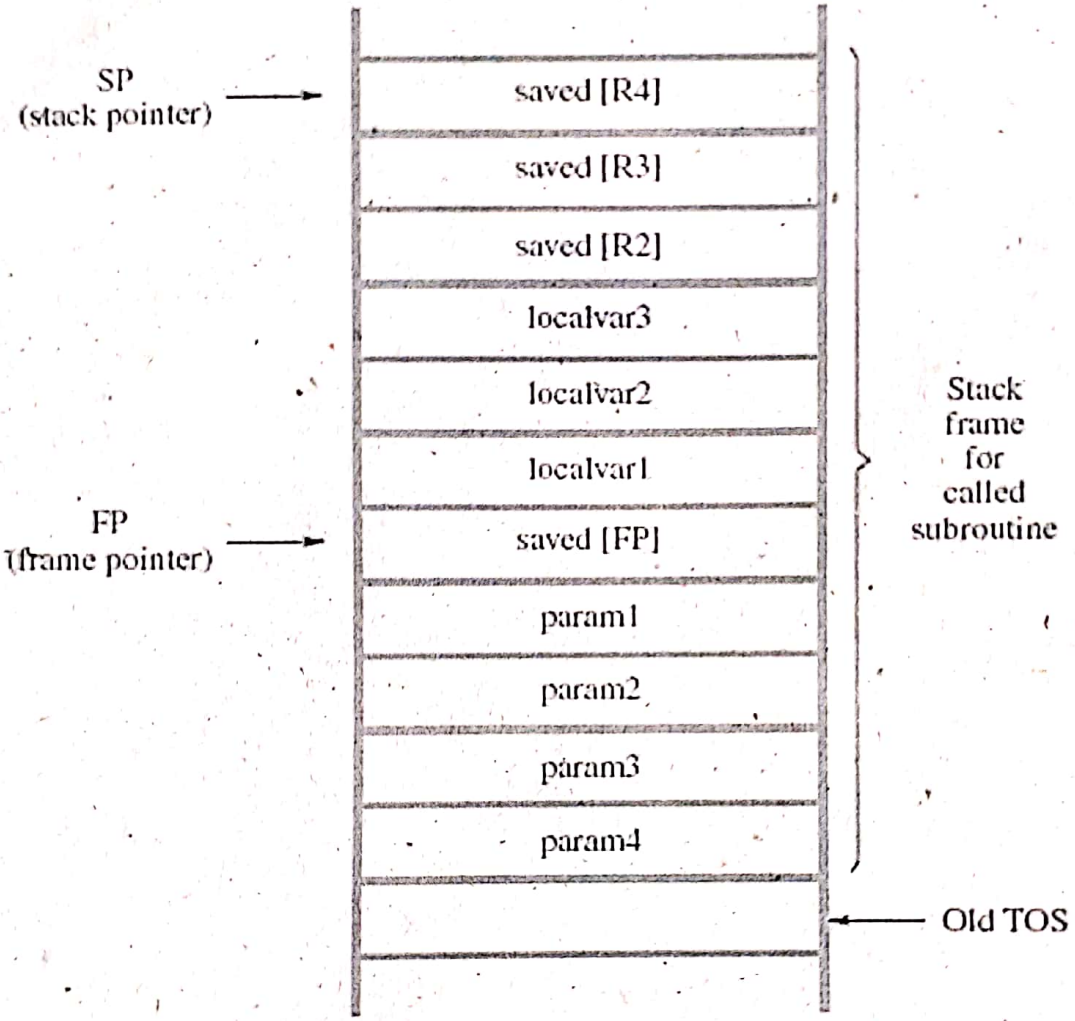


Figure 1.27 A subroutine stack frame example.

Scanned with CamScanner

Module II

BASIC PROCESSING UNIT

Processing unit executes machine instructions and co-ordinates the activities of other units is called the Instruction set processor or the processor.

i) Pipelined execution

The execution of one instruction is started before the execution of the preceding instruction is completed.

ii) Superscalar operations:

several instructions are fetched and executed at the same time.

Some Fundamental Concepts

To execute an instruction, the processor has to perform the following steps

Fetch Phase:

1. Fetch the contents of the memory location pointed to by the PC. $IR \leftarrow [[PC]]$
2. Assuming that the memory is byte addressable increment the contents of the PC by 4. that is

$$PC \leftarrow [PC] + 4.$$

Execution Phase:

3. Carry out the actions specified by the instruction in the IR.

Scanned with CamScanner

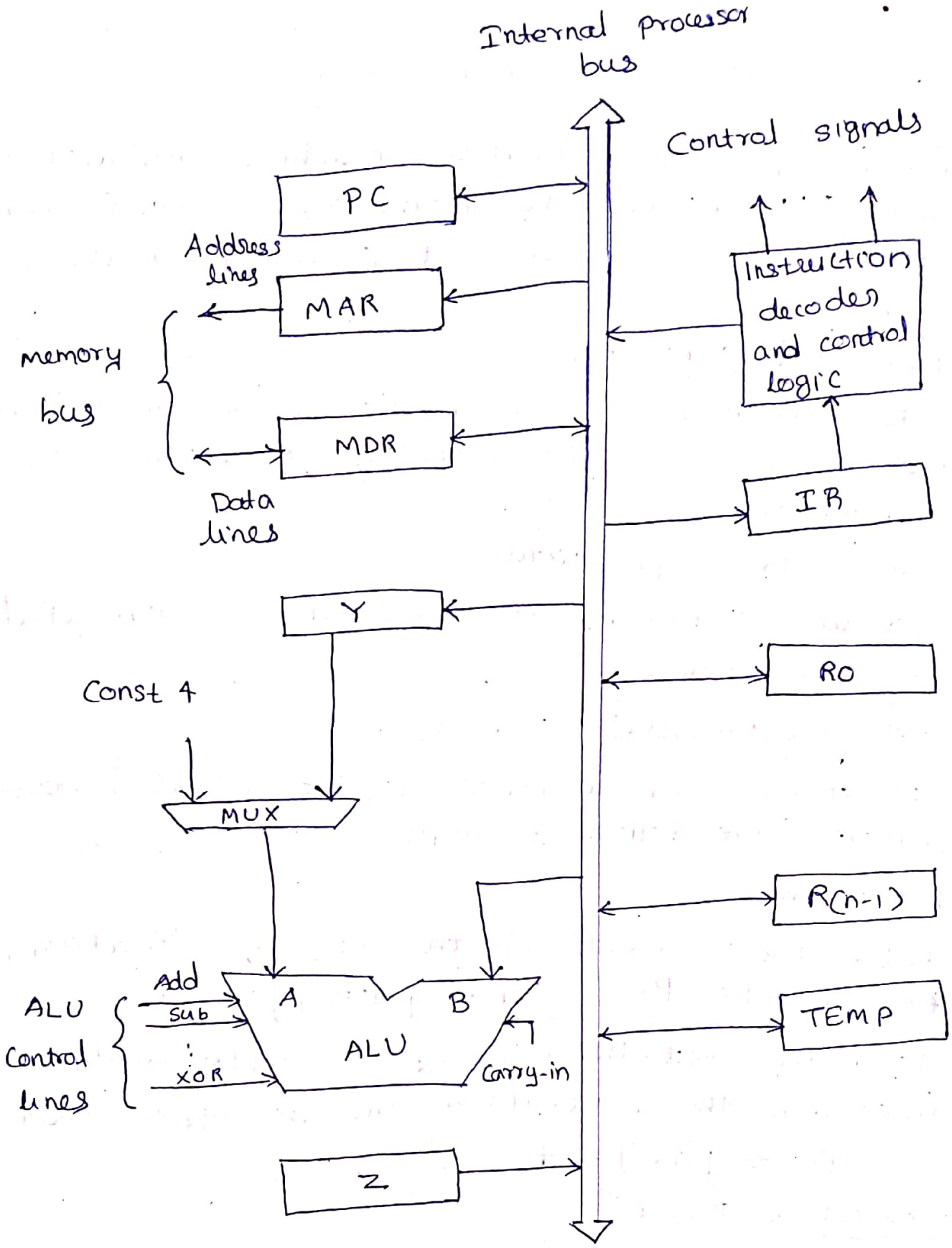


Figure 2.1 single bus organization of the datapath inside a processor.

Scanned with CamScanner

1) Arithmetic and logic unit (ALU) and all the registers are interconnected via a single common bus. Internal bus

2) The bus which connects the processor to the memory and I/O devices.

3) The data line of the external memory are connected to the internal processor bus via the memory data register MDR.

4) The address lines of external memory are connected to the internal processor bus via the memory Address register (MAR).

5) Data may be loaded into MDR either from the memory bus or from the internal processor bus.

6) The data stored in MDR may be placed on either bus.

7) The input of MAR is connected to the internal bus and its output is connected to the external bus.

8) The control lines of the memory bus are connected to the instruction decoder and control logic blocks.

9) processor registers R0 through R(n-1)

10) special-purpose registers eg: Index registers or stack pointers.

11) Y, Z & TEMP are transparent to the programmer used by the processor for temporary storage during execution of some instructions.

12) The registers, the ALU and the interconnecting bus are collectively referred to as the data path

Scanned with CamScanner

Instruction Execution.

1. REGISTER TRANSFERS

Transfer a word of data from one processor register to another or to the ALU.

i) processor clock:
All operations and data transfers within the processor take place within time periods defined by the processor clock.

ii) The registers consists of edge-triggered flip-flops.

iii) multiphase clocking.
When edge-triggered flip-flops are not used, two or more clock signals may be needed to guarantee proper transfer of data.

iv) The input and output of register R_i are connected to the bus via switches controlled by the signals $R_i in$ and $R_i out$, respectively.
$R_i in = 1$, the data on the bus are loaded into R_i.
$R_i out = 1$, the contents of register R_i are placed on the bus.

2. PERFORMING AN ARITHMETIC OR LOGIC OPERATION

i) ALU: it's a combinational circuit (no internal storage)

ii) A sequence of operations to add the contents of register R1 those of register R2 and store the result in register R3 is

1. R1 out, Y in
2. R2 out, Select Y, Add, Z in
3. Z out, R3 in

Scanned with CamScanner

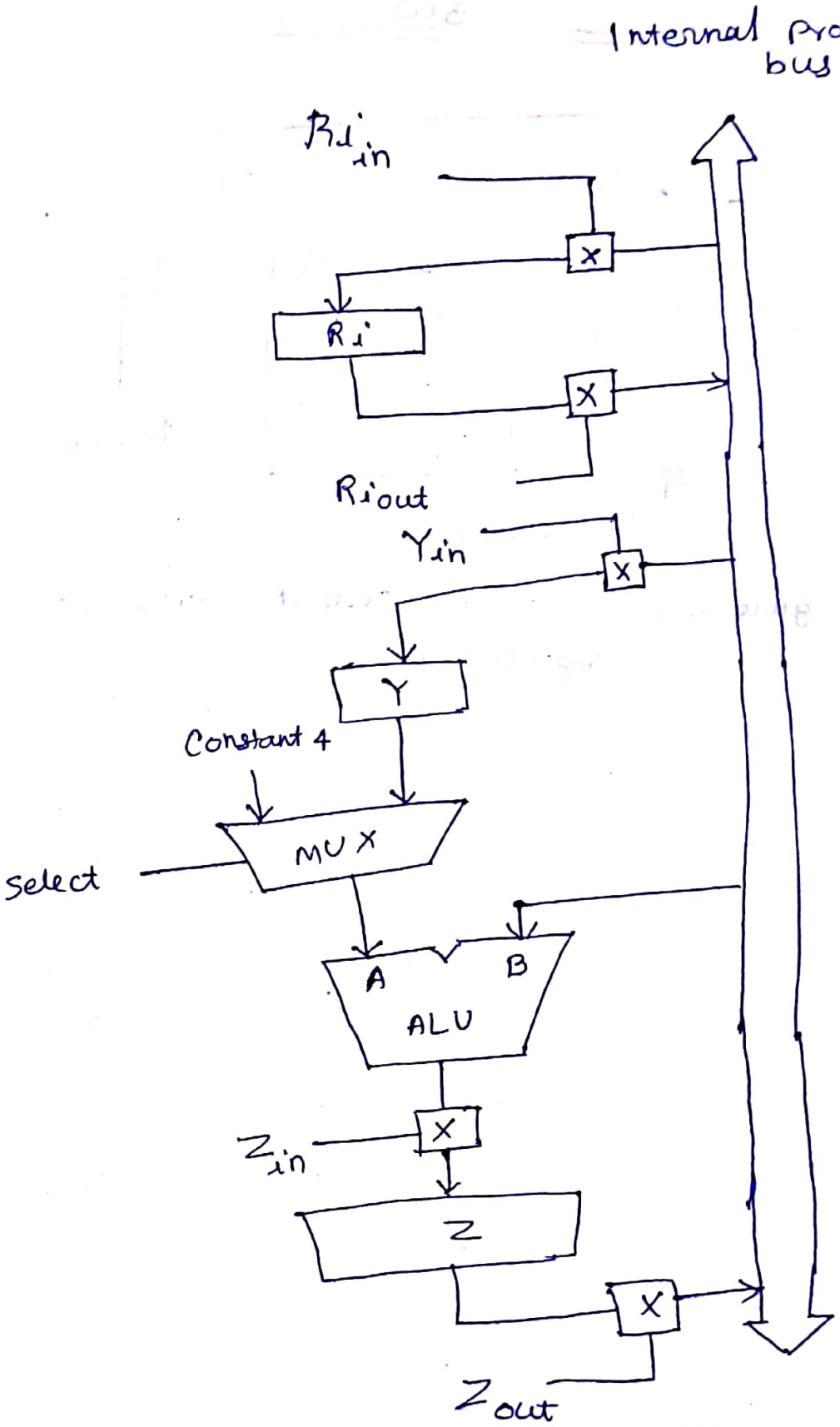


Figure 2.2 Input and output gating for the registers

Scanned with CamScanner

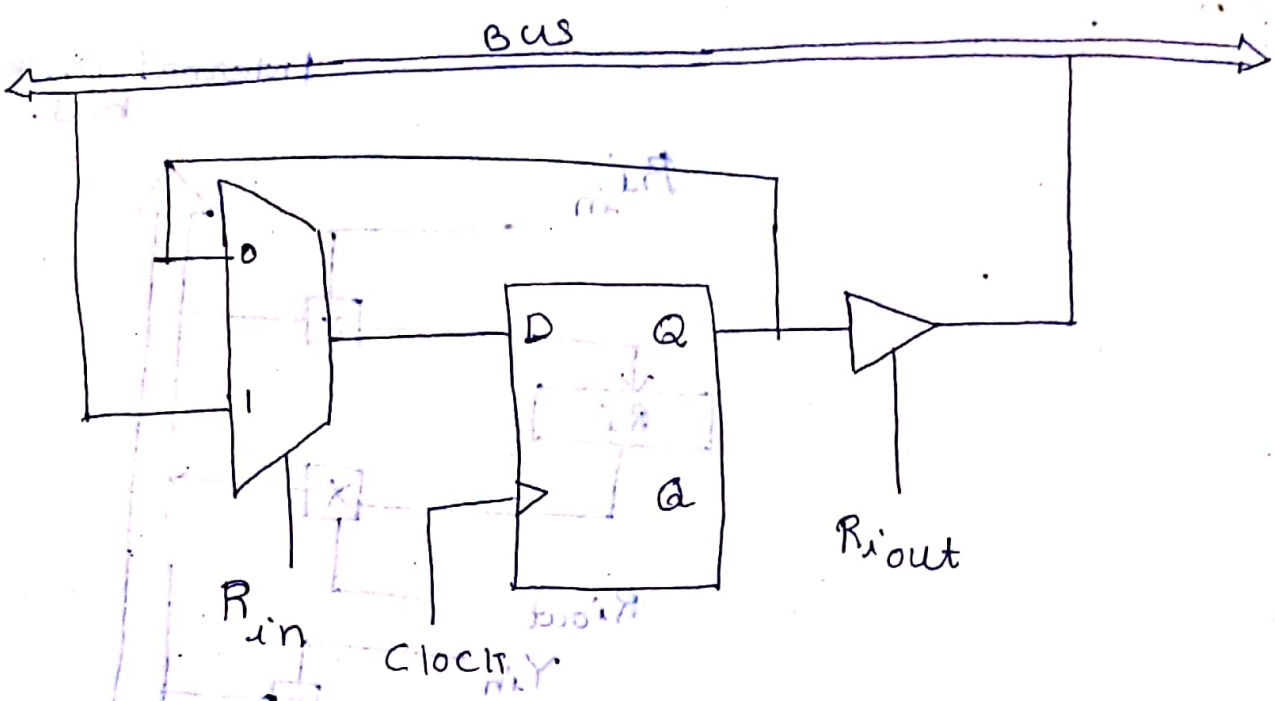


Figure 2.3 Input and output gating for one register bit.

Scanned with CamScanner

FETCHING A WORD FROM MEMORY

1. To fetch a word of information from memory, the processor has to specify the address of the memory location where this information is stored and request a Read operation.
2. When the requested data are received from the memory they are stored in register MDR, from where they can be transferred to other registers in the processor.
3. The processor completes one internal data transfer in one clock cycle.
4. The speed of operation of the addressed device, on the other hand, varies with the device.

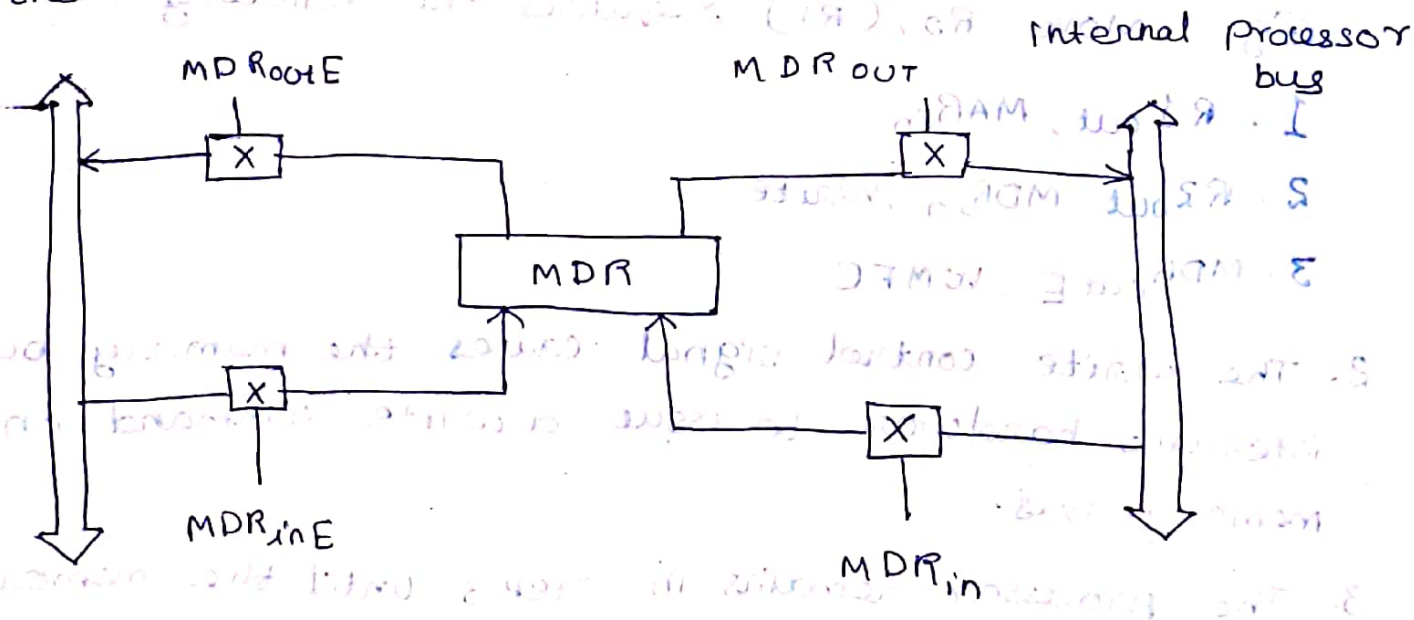


Figure 2.4 Connection and control signals for register MDR.

5. A control signal called Memory-Function-Completed (MFC) is used to accommodate the variability in response time, the processor waits until it receives an indication that the requested Read operation has been completed.

Scanned with CamScanner

Eg: Move (R1), R2

The memory read operation requires three steps, which can be described by the signals being activated as follows:

1. $R1_{out}$, MAR_{in}, Read
2. MDR_{inE}, WMFC
3. MDR_{out}, $R2_{in}$

STORING A WORD IN MEMORY

1. Writing a word into a memory follows,
   a) The desired address is loaded into MAR
   b) the data to be written are loaded into MDR,
   c) A write command is issued.

Eg: Move R2, (R1) requires the following sequence:

1. $R1_{out}$, MAR_{in}
2. $R2_{out}$, MDR_{in}, Write
3. MDR_{outE}, WMFC

2. The write control signal causes the memory bus interface hardware to issue a write command on the memory bus.
3. The processor remains in step 3 until the memory operation is completed and an MFC response is received.

EXECUTION OF A COMPLETE INSTRUCTION

The sequence of elementary operations required to execute one instruction.

Scanned with CamScanner

Eg: Add (R3), R1

The contents of a memory location pointed to by R3 to register R1. Executing this instruction requires the following actions:

1. Fetch the instruction.
2. Fetch the first operand (the contents of the memory location pointed to by R3).
3. Perform the addition.
4. Load the result into R1.

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R3_{out}$, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End

Figure 2.5 Control sequence for execution of the instruction Add (R3), R1.

Figure 2.5 gives the sequence of control steps required to perform the above 4 operations for the single-bus architecture of Figure 2.1.

Scanned with CamScanner

Step 1:
a) the instruction fetch operation is initiated by loading the contents of the PC into the MAR and sending a Read request to the memory.
b) the select signal is set to Select 4, which causes the multiplexer MUX to select the constant 4.
c) this value is added to the operand input B, which is the contents of the PC, and
d) the result is stored in register Z.

Step 2:
a) The updated value is moved from register Z back into the PC, while waiting for the memory to respond.

Step 3:
the word fetched from the memory is loaded into the IR.
Steps 1 through 3 constitute the instruction fetch phase, which is the same for all instructions.

Step 4:
a) The instruction decoding circuit interprets the contents of the IR at the beginning of step 4.
b) this enables the control circuitry to activate the control signals for steps 4 through 7, which constitute the execution phase.
c) The contents of register R3 are transferred to the MAR.
d) and a memory read operation is initiated.

Scanned with CamScanner

Step5:
a) the contents of R1 are transferred to register Y, to prepare for the addition operation.

Step6:
a) When the Read operation is completed, the memory operand is available in register MDR, and the addition operation is performed.
b) The contents of MDR are gated to the bus, and thus also to the B input of the ALU and
c) register Y is selected as the second input to the ALU by choosing Select Y.
d) The sum is stored in register Z.

Step7:
a) The sum is transferred to R1.
b) The End signal causes a new instruction fetch cycle to begin by returning to step1.

BRANCH INSTRUCTIONS

A branch instruction replaces the contents of the PC with the branch target address. This address is usually obtained by adding an offset X, which is given in the branch instruction, to the updated value of the PC.

Figure 7.6 gives a control sequence that implements an unconditional branch instruction.

Step 1 through 3 loads the instruction into the IR.

Step4:
a) The offset value is extracted from the IR by the instruction decoding circuit.

Scanned with CamScanner

b) Since the value of the updated PC is already available in register Y, the offset X is gated onto the bus in step 4, and an addition operation is performed.

step 5:
the branch target address is loaded into the PC.

offset X = branch target address − address immediately following the branch instruction.

Conditional branch, for a Branch-on-negative (Branch < 0) instruction, step 4 in Figure 2.6 is replaced with

Offset-field-of-IRout, Add, Zin, If N=0 then End.

If N=0 the processor returns to step 1 immediately after step 4. If N=1, step 5 is performed to load a new value into the PC, thus performing the branch operation.

MULTIPLE-BUS ORGANIZATION

i) In single-bus structure only one data item can be transferred over the bus in a clock cycle.

ii) Multiple-bus organization provide multiple internal paths that enable several transfers to take place in parallel.

Figure 7.7 depicts a three-bus structure used to connect the registers and the ALU of a processor.

a) All general-purpose registers are combined into a single block called the register file.
- array of memory cells.

Scanned with CamScanner

Step	Action
1	PCout, MARin, Read, Select4, Add, Zin
2	Zout, PCin, Yin, WMFC
3.	MDRout, IRin
4.	Offset-field-of-IRout, Add, Zin
5.	Zout, PCin, End.

Figure 2.6 Control sequence for an unconditional Branch instruction.

Scanned with CamScanner

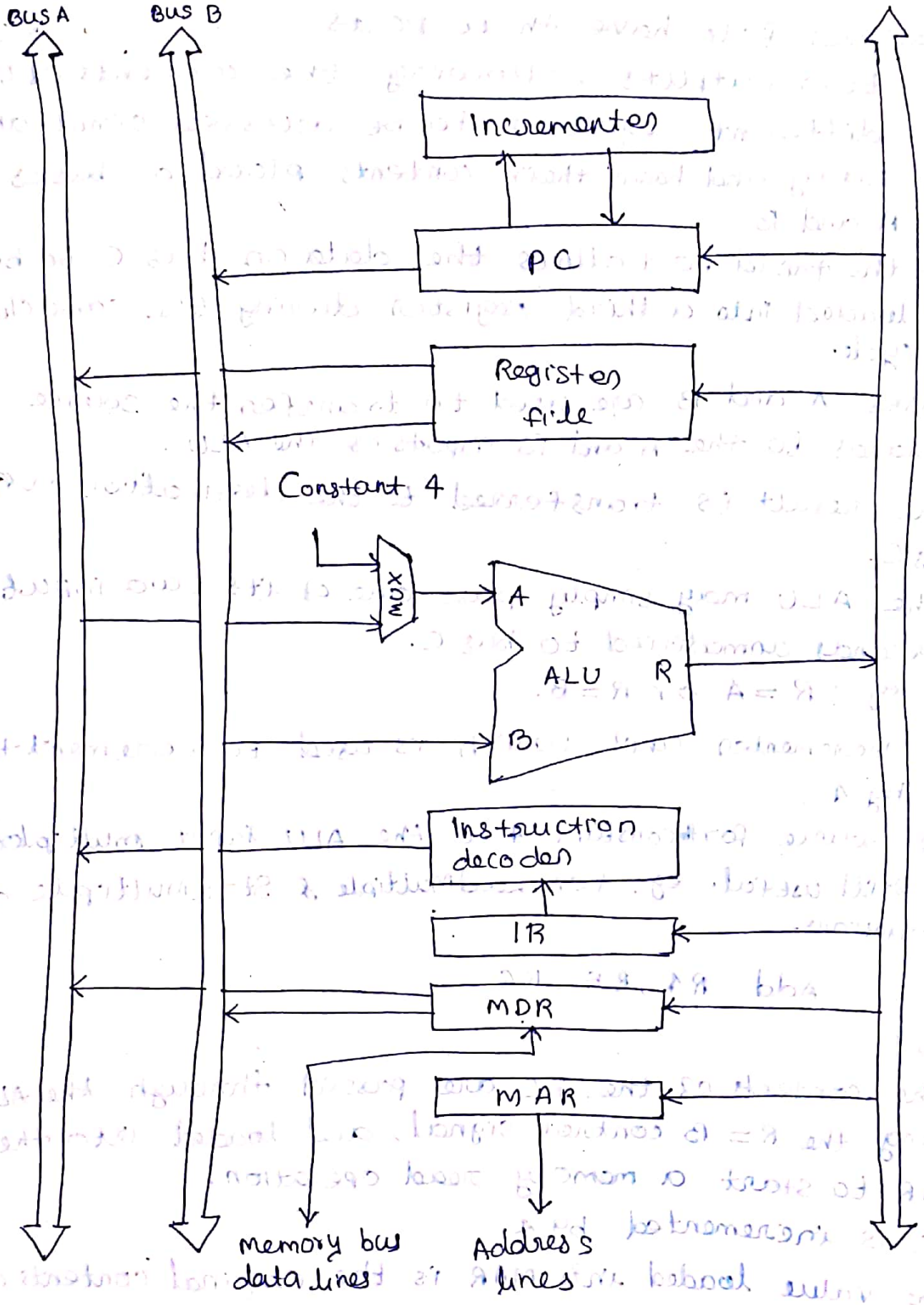


Figure 2.7 Three-bus organization of the datapath.

Scanned with CamScanner

b) Register file have three ports
- two outputs, allowing the contents of two different registers to be accessed simultaneously and have their contents placed on buses A and B.
- the third port allows the data on bus C to be loaded into a third register during the same clock cycle.

c) Buses A and B are used to transfer the source operands to the A and B inputs of the ALU.

d) The result is transferred to the destination over bus C.

e) the ALU may simply pass one of its two input operands unmodified to bus C.
eg: R = A or R = B.

f) The incrementer unit, which is used to increment the PC by 4.

g) The source for the constant 4 at the ALU input multiplexer is still useful. eg: for LoadMultiple & StoreMultiple instructions.

Eg: Add R1, R5, R6

Step 1:

a) the contents of the PC are passed through the ALU using the R = B control signal, and loaded into the MAR to start a memory read operation.

b) PC is incremented by 4.

c) The value loaded into MAR is the original contents of the PC.

Scanned with CamScanner

d) The incremented value is loaded into the PC at the end of the clock cycle and will not affect the contents of MAR.

step2:
The processor waits for MFC and loads the data received into MDR, then transfers them to IR.

step3:
Transfers into IR.

step4:
The execution phase of the instruction requires only one control step to complete.

SEQUENCING OF CONTROL SIGNALS

Generating the control signals in the proper sequence can be done in two ways.
1. Hardwired control and
2. Microprogrammed control.

I. Hardwired control

i) Each step in the sequence is completed in one clock period.
ii) A counter may be used to keep track of the control steps.
iii) Control signals are determined by the following information:
a) Contents of the control step counter.
b) Contents of the instruction register.
c) Contents of the condition code flags.

Scanned with CamScanner

d) External input signals, such as MFC and interrupt requests.

Step	Action
1	PC_{out}, R=B, MAR_{in}, Read, IncPC
2	WMFC
3	MDR_{outB}, R=B, IR_{in}
4	$R4_{outA}$, $R5_{outB}$, Select A, Add, RG_{in}, End

Figure 7.8 Control sequence for the instruction Add R4, R5, R6 for the three-bus organization in Figure 7.7.

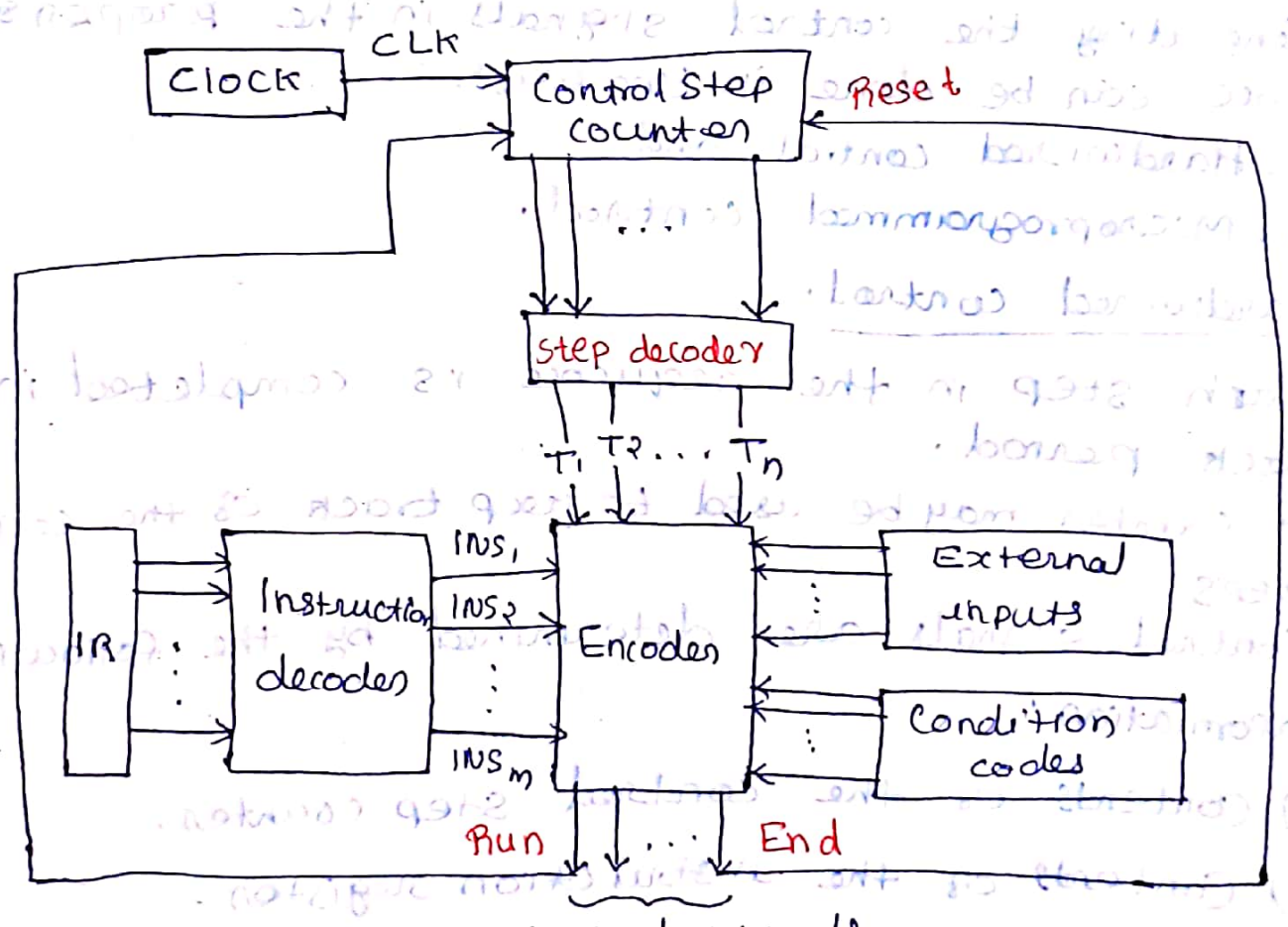


Figure 7.9 Control unit organization

Scanned with CamScanner

e) The step decoder provides a separate signal line for each step or time slot, in the control sequence.

f) For any instruction loaded in IR, one of the output lines INS1 through INSm is set to 1, and all other lines are set to 0.

g) eg: the encoder generates the Z_{in} control signal for the processor organization in figure 2.1 is

$$Z_{in} = T_1 + T_6 \cdot ADD + T_4 \cdot BR + \cdots$$

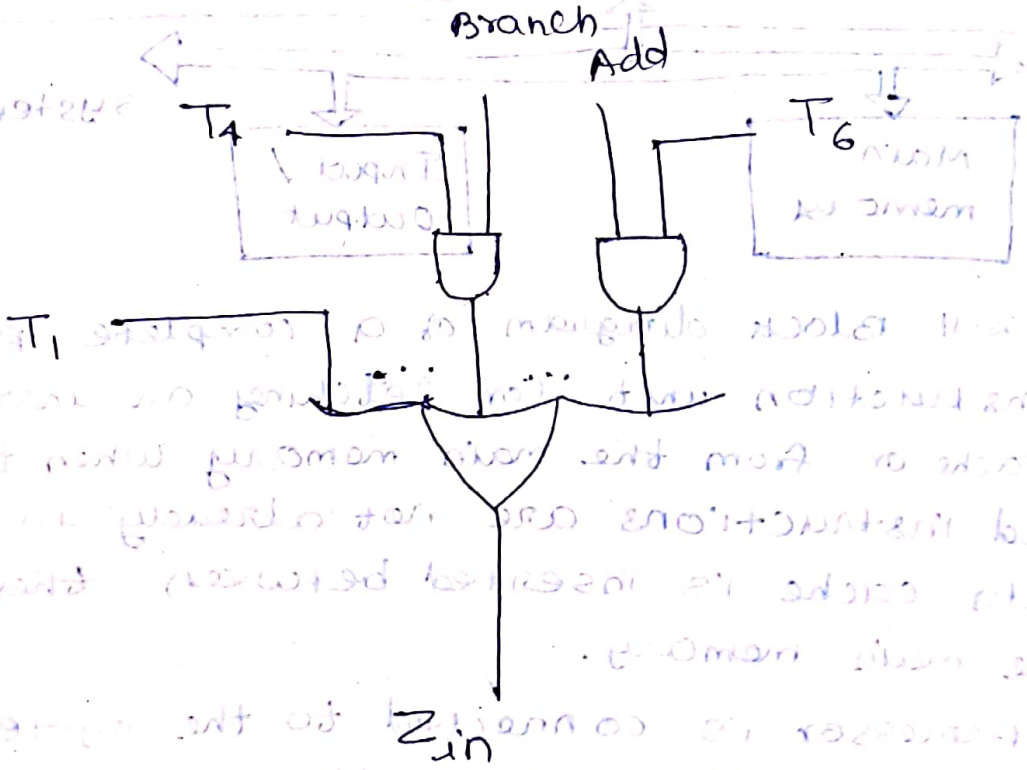


Figure 2.10 Generation of the Z_{in} control signal for the processor in Figure 2.1.

Scanned with CamScanner

A COMPLETE PROCESSOR

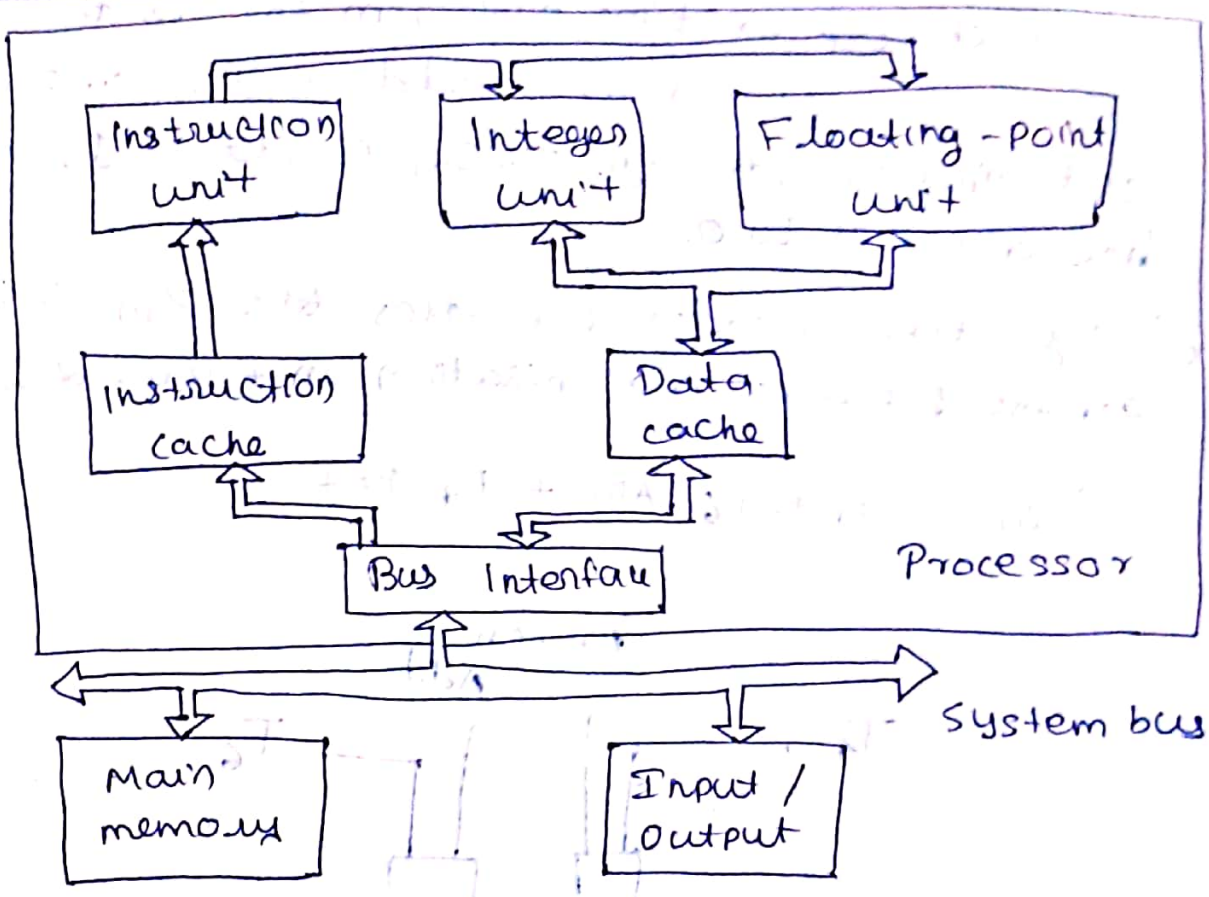


Figure 2.11 Block diagram of a complete processor.

i) An instruction unit for fetching an instruction from cache or from the main memory when the desired instructions are not already in the cache.

ii) A data cache is inserted between these units and the main memory.

iii) The processor is connected to the system bus and hence, to the rest of the computer, by means of a bus interface.

II. MICROPROGRAMMED CONTROL

i) Control signals are generated by a program similar to machine language programs.

Scanned with CamScanner

micro instruction	PCin	PCout	MARin	Read	MDRout	IRin	Yin	Select	Add	Zin	Zout	R1out	R1in	R3out	WMFC	End
1	0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0
2	1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0
3	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
4	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0
5	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0
6	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	1		1			1

Figure 2.12 An example of microinstructions for Figure 2.5

Control word

A control word (CW) is a word whose individual bits represent the various control signals in Figure 2.9

Microroutine

A sequence of CWs corresponding to the control sequence of a machine instruction constitutes the microroutine for that instruction.

Microinstructions

The individual control words in the microroutine are referred to as microinstructions.

Control Store.

The microroutines for all instructions in the instruction set of a computer are stored in a special memory called the control store.

Scanned with CamScanner

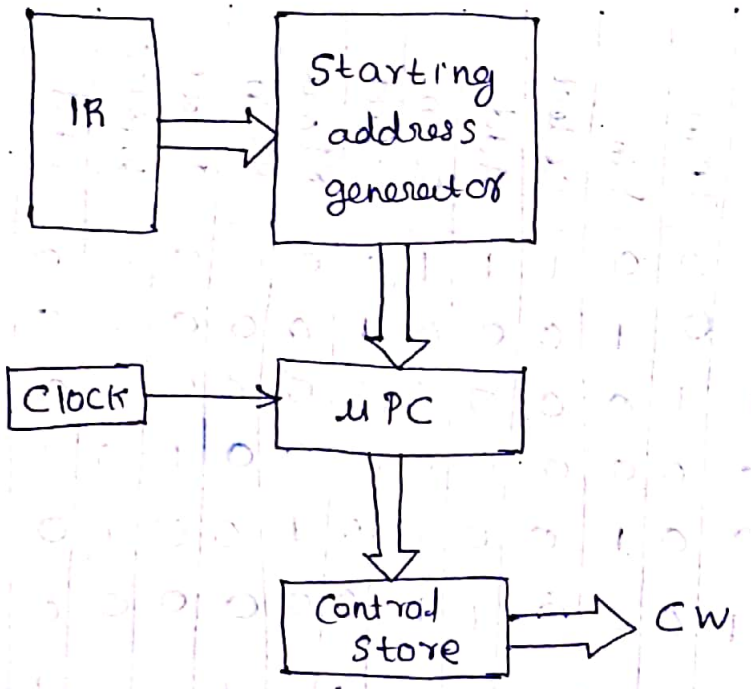


Figure 2.13 Basic organization of a microprogrammed control unit

micro program counter (μPC)

To read the control words sequentially from the control store, a micro program counter (μPC) is used.

i) Every time a new instruction is loaded into the IR, the output of the block labeled "starting address generator" is loaded into the μPC.

ii) The μPC is then automatically incremented by the clock, causing successive microinstructions to be read from the control store.

iii) Hence, the control signals are delivered to various parts of the processor in the correct sequence.

The branch instruction Branch <0 may now be implemented by a microroutine such as that shown in Figure 2.14.

Scanned with CamScanner

Address	Microinstruction.
0	PC_{out}, MAR_{in}, Read, Select 4, Add, Z_{in}
1	Z_{out}, PC_{in}, Y_{in}, WMFC
2	MDR_{out}, IR_{in}
3	Branch to starting address of appropriate microroutine.
...	...
25	If N=0, then branch to microinstruction 0
26	Offset-field-of-IRout, Select Y, Add, Z_{in}
27	Z_{out}, PC_{in}, End

Figure 2.14 Microroutine for the instruction Branch<0.

i) After loading the instruction into IR, a branch microinstruction transfers control to the corresponding microroutine, which is assumed to start at location 25 in the control store.

ii) This address is the output of the starting address generator block in Figure 2.13.

iii) The microinstruction at location 25 tests the N bit of the condition codes. If this bit is equal to 0, a branch takes place to location 0 to fetch a new machine instruction. Otherwise, the microinstruction at location 26 is executed to put the branch target address into register Z. The microinstruction in location 27 loads this address into the PC.

Scanned with CamScanner

ARITHMETIC ALGORITHMS

INTRODUCTION

NUMBER REPRESENTATION

1. Numbers can be represented in 3 formats:

    1) Sign and magnitude
    2) 1's complement
    3) 2's complement

2. In all three formats, MSB=0 for +ve numbers & MSB=1 for -ve numbers.

3. In **sign-and-magnitude system**,
    negative value is obtained by changing the MSB from 0 to 1 of the corresponding positive value.
        For ex, +5 is represented by 0101 &
            -5 is represented by 1101.

4. In **1's complement system**,
    negative values are obtained by complementing each bit of the corresponding positive number.
        For ex, -5 is obtained by complementing each bit in 0101 to yield 1010.
    (In other words, the operation of forming the 1's complement of a given number is equivalent to subtracting that number from 2^n-1).

5. In **2's complement system**,
    forming the 2's complement of a number is done by subtracting that number from 2^n.
        For ex, -5 is obtained by complementing each bit in 0101 & then adding 1 to yield 1011. (In other words, the 2's complement of a number is obtained by adding 1 to the 1's complement of that number).

6. 2's complement system yields the most efficient way to carry out addition/subtraction operations.

ADDITION OF POSITIVE NUMBERS

1. Consider adding two 1-bit numbers.
2. The sum of 1 & 1 requires the 2-bit vector 10 to represent the value 2. We say that sum is 0 and the carry-out is 1.

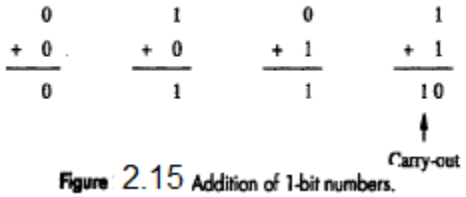


Figure 2.15 Addition of 1-bit numbers.

.

ADDITION & SUBTRACTION OF SIGNED NUMBERS

Following are the two rules for addition and subtraction of n-bit signed numbers using the 2's complement representation system (Figure 2.16).

Rule 1:

- **To Add** two numbers, add their n-bits and ignore the carry-out signal from the MSB position.
- Result will be algebraically correct, if it lies in the range -2^{n-1} to $+2^{n-1}-1$.

Rule 2:

- **To Subtract** two numbers X and Y (that is to perform X-Y), take the 2's complement of Y and then add it to X as in rule 1.
- Result will be algebraically correct, if it lies in the range (2^{n-1}) to $+(2^{n-1}-1)$.

1. When the result of an arithmetic operation is outside the representable-range, an arithmetic overflow is said to occur.
2. To represent a signed in 2's complement form using a larger number of bits, repeat the sign bit as many times as needed to the left. This operation is called **sign extension**.
3. In 1's complement representation, the result obtained after an addition operation is not always correct. The carry-out(c_n) cannot be ignored. If c_n=0, the result obtained is correct. If c_n=1, then a 1 must be added to the result to make it correct.

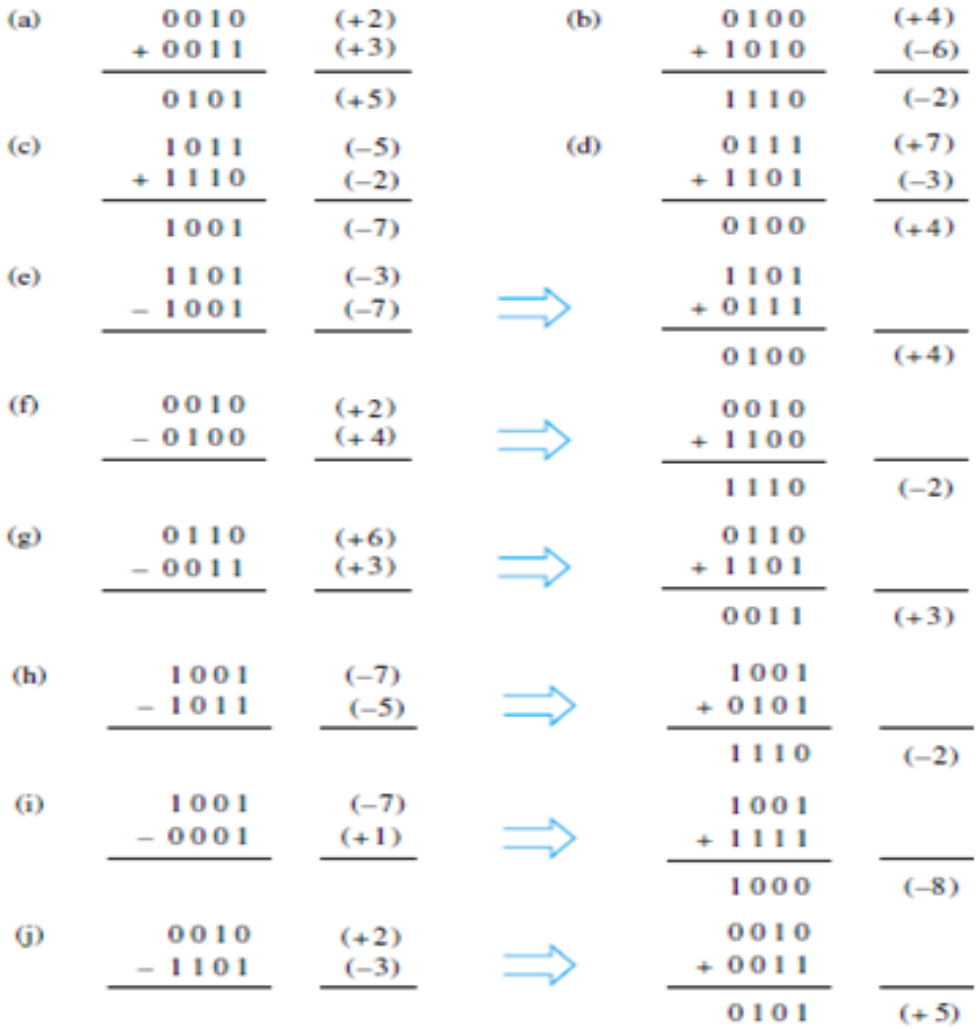


Figure 2.16 2's-complement Add and Subtract operations.

OVERFLOW IN INTEGER ARITHMETIC

1. When result of an arithmetic operation is outside the representable-range, an **arithmetic overflow** is said to occur.
2. For example: If we add two numbers +7 and +4, then the output sum S is 1011(□0111+0100), which is the code for -5, an incorrect result.
3. An overflow occurs in following 2 cases
    1) Overflow can occur only when adding two numbers that have the same sign.
    2) The carry-out signal from the sign-bit position is not a sufficient indicator of overflow when adding signed numbers.

ADDITION & SUBTRACTION OF SIGNED NUMBERS n-BIT RIPPLE CARRY ADDER

1. A cascaded connection of n full-adder blocks can be used to add 2-bit numbers.
2. Since carries must propagate (or ripple) through cascade, the configuration is called an n-bit ripple carry adder (Figure 2.17).

x_i	y_i	Carry-in c_i	Sum s_i	Carry-out c_{i+1}
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

$$s_i = \overline{x}_i\overline{y}_i c_i + \overline{x}_i y_i \overline{c}_i + x_i\overline{y}_i\overline{c}_i + x_i y_i c_i = x_i \oplus y_i \oplus c_i$$

$$c_{i+1} = y_i c_i + x_i c_i + x_i y_i$$

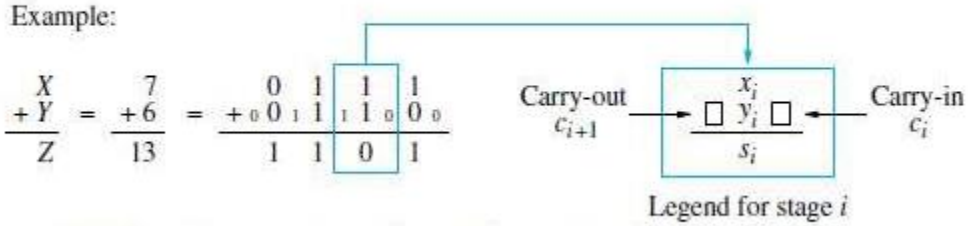
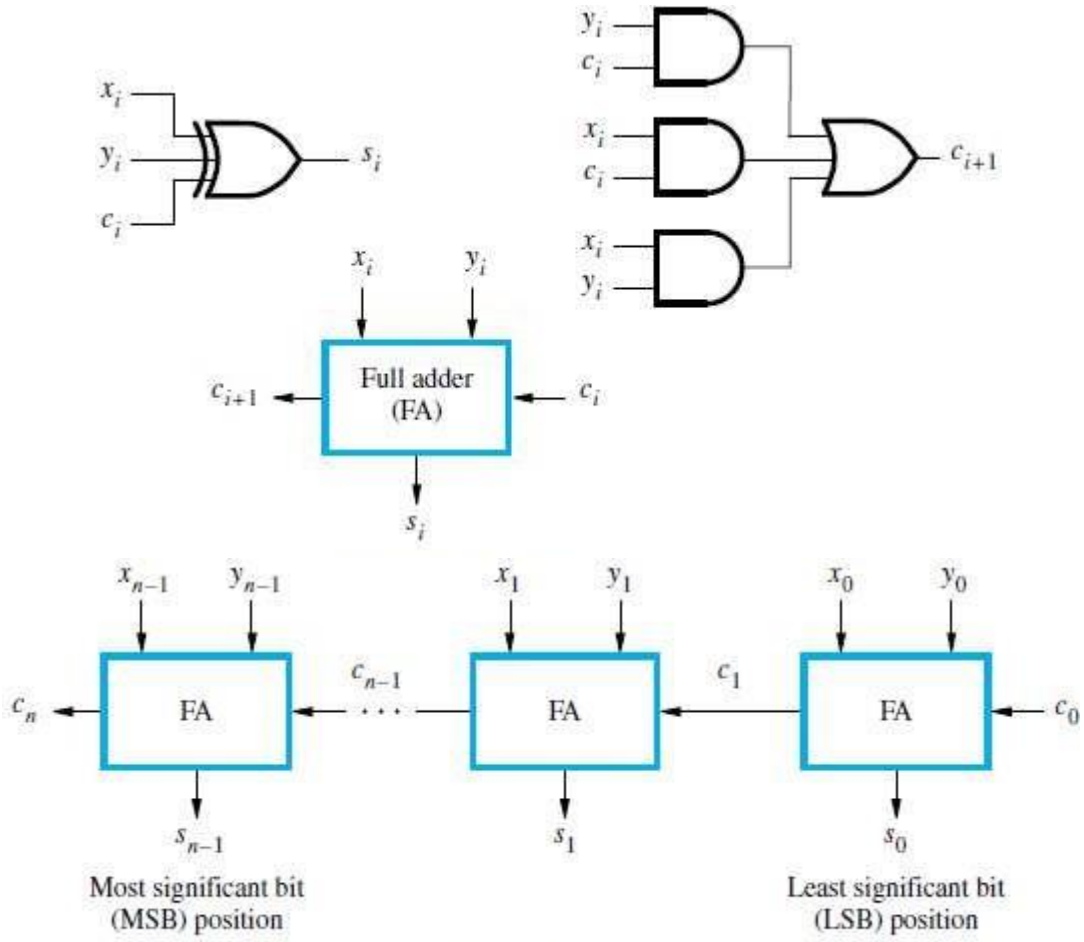
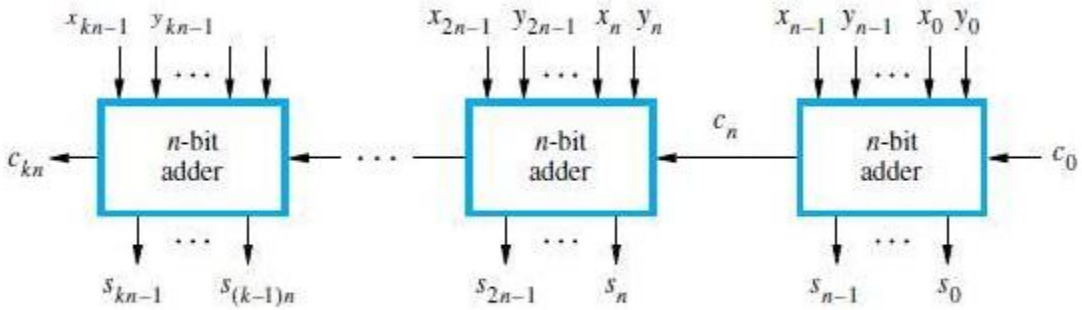


Figure 2.17 Logic specification for a stage of binary addition.



(b) An *n*-bit ripple-carry adder



(c) Cascade of *k* *n*-bit adders

Figure 2.18 Logic for addition of binary numbers.

ADDITION/SUBTRACTION LOGIC UNIT

1) The n-bit adder can be used to add 2's complement numbers X and Y .
2) **Overflow** can only occur when the signs of the 2 operands are the same.
3) In order to perform the subtraction operation X-Y on 2's complement numbers X and

Y; we form the 2's complement of Y and add it to X.

1) Addition or subtraction operation is done based on value applied to the Add/Sub input control-line.
2) Control-line=0 for addition, applying the Y vector unchanged to one of the adder inputs. Control-line=1 for subtraction, the Y vector is 2's complemented.

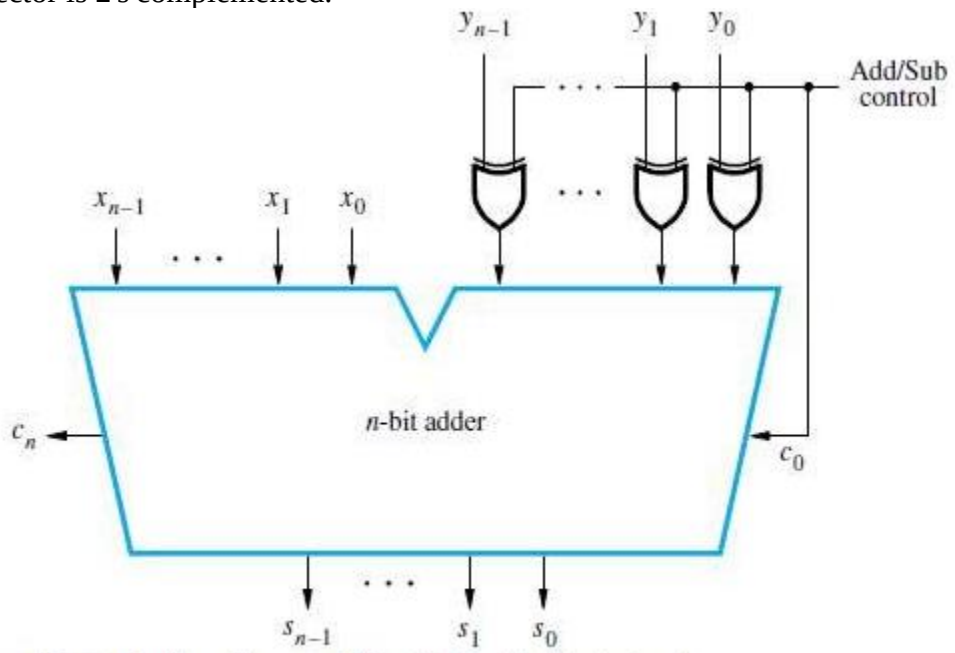


Figure 2.19 Binary addition/subtraction logic circuit.

MULTIPLICATION OF UNSIGNED NUMBERS

The product of two, unsigned, *n*-digit numbers can be accommodated in 2*n* digits. In the binary system, multiplication of the multiplicand by one bit of the multiplier is easy. If the multiplier bit is 1, the multiplicand is entered in the appropriate shifted position. If the multiplier bit is 0, then 0s are entered.

```
                1  1  0  1      (13) Multiplicand M
          ×     1  0  1  1      (11) Multiplier Q
          ------------------
                1  1  0  1
             1  1  0  1
          0  0  0  0
       1  1  0  1
    ----------------------
1  0  0  0  1  1  1  1      (143) Product P
```

Figure 2.20 Manual Multiplication algorithm

There are two types of binary multiplier organization.

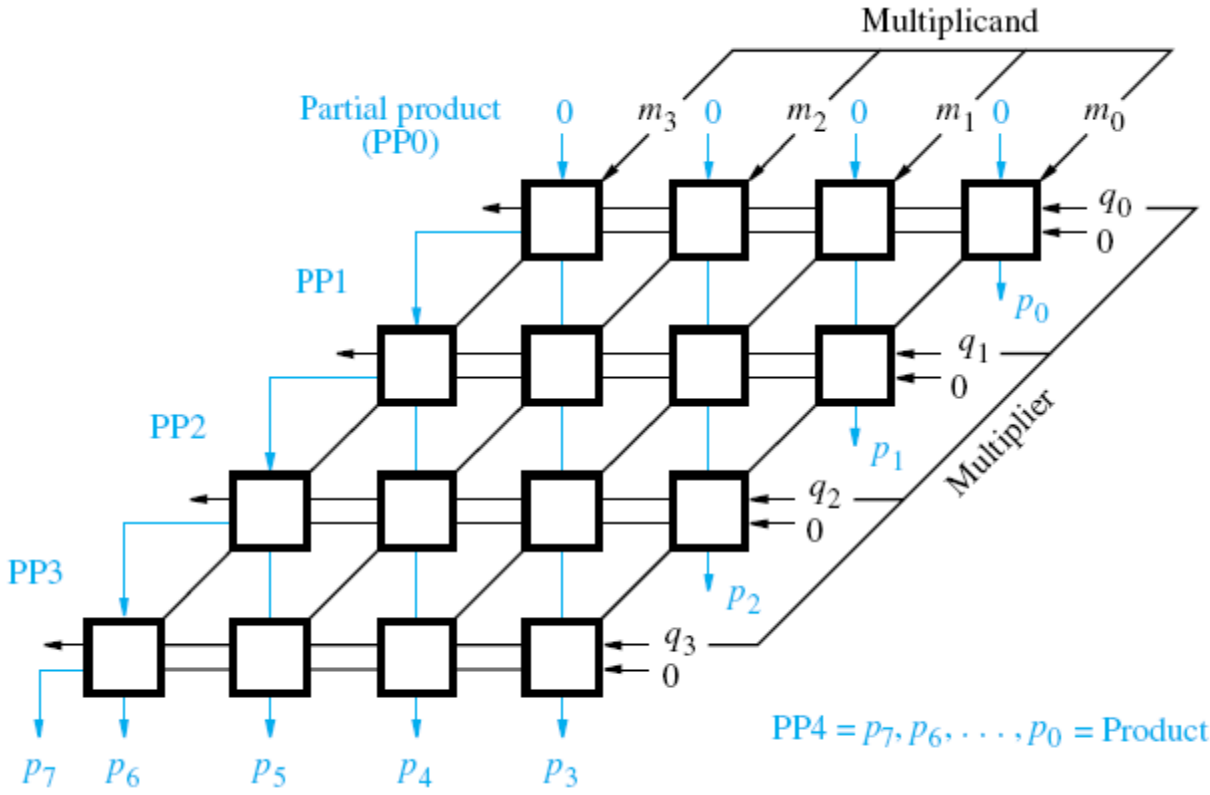
1. Combinational array multiplier
2. Sequential Circuit Multiplier

1. Combinational Array Multiplier

Binary multiplication of unsigned operands can be implemented in a combinational, twodimensional, logic array, as shown in Figure 2.16 for the 4-bit operand case.
The main component in each cell is a full adder, FA. The AND gate in each cell determines whether a multiplicand bit, mj , is added to the incoming partial-product bit, based on the value of the multiplier bit, qi . Each row i, where $0 \leq i \leq 3$, adds the multiplicand (appropriately shifted) to the incoming partial product, PPi, to generate the outgoing partial product, PP(i + 1), if qi = 1. If qi = 0, PPi is passed vertically downward unchanged. PP0 is all 0s, and PP4 is the desired product. The multiplicand is shifted left one position per row by the diagonal signal path.

2. Sequential Circuit Multiplier

The combinational array multiplier just described uses a large number of logic gates for multiplying numbers of practical size, such as 32- or 64-bit numbers. Multiplication of two *n*-bit numbers can also be performed in a sequential circuit that uses a single *n*-bit adder.
The block diagram in Figure 2.18a shows the hardware arrangement for sequential multiplication. This circuit performs multiplication by using a single *n*-bit adder *n* times to implement the spatial addition performed by the *n* rows of ripple-carry adders.



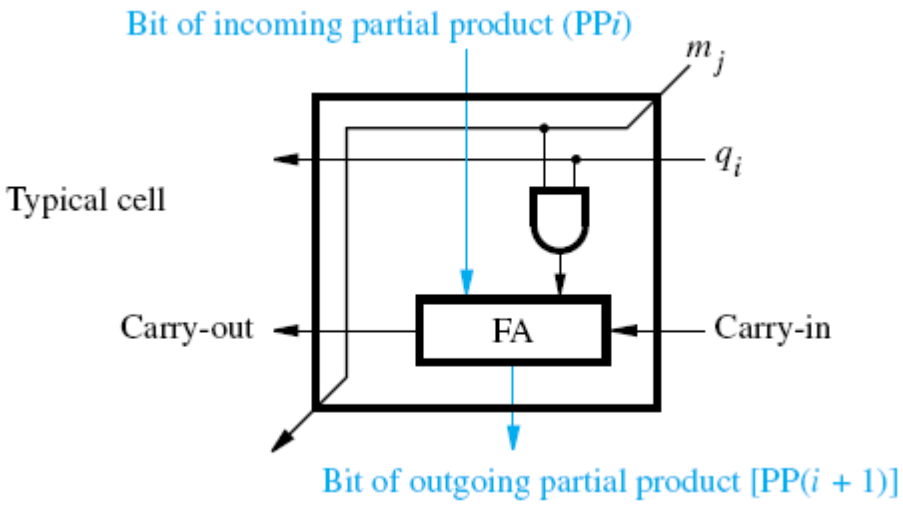


Figure 2.21 Array multiplication of unsigned binary operands using combinational array multiplier.

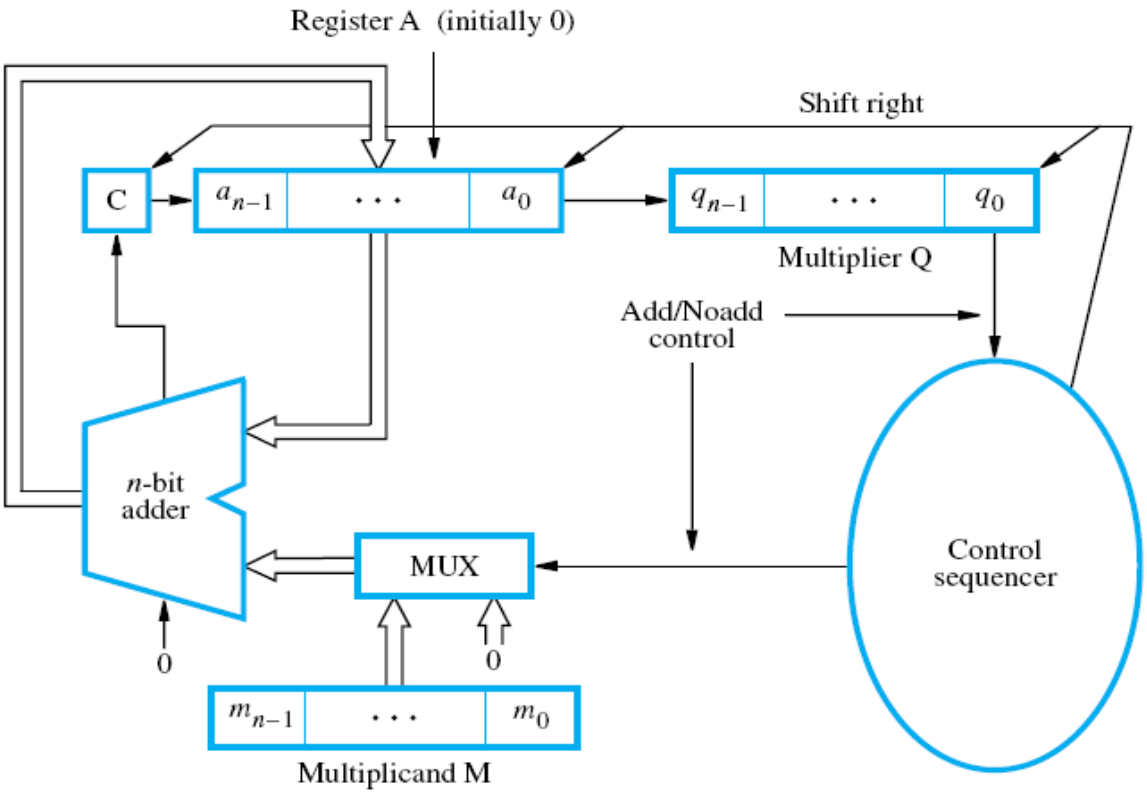
Sequential Circuit Multiplier works as follows:
1. Registers A and Q are shift registers, concatenated as shown. Together, they hold partial product PPi. Store multiplicand in register M, Store multiplier in register Q. Store 0 in register A and carry flip-flop.
2. for i=0 to n-1
If qi = 1, the multiplexer MUX select multiplicand M, to be added to PPi to generate PP(i + 1). add the content of M and A, Store the result in A and the carry in C, Shift C,A, and Q one bit right.
If qi = 0, the multiplexer MUX select 0, Shift C,A, and Q one bit right.

ARRAY MULTIPLICATION

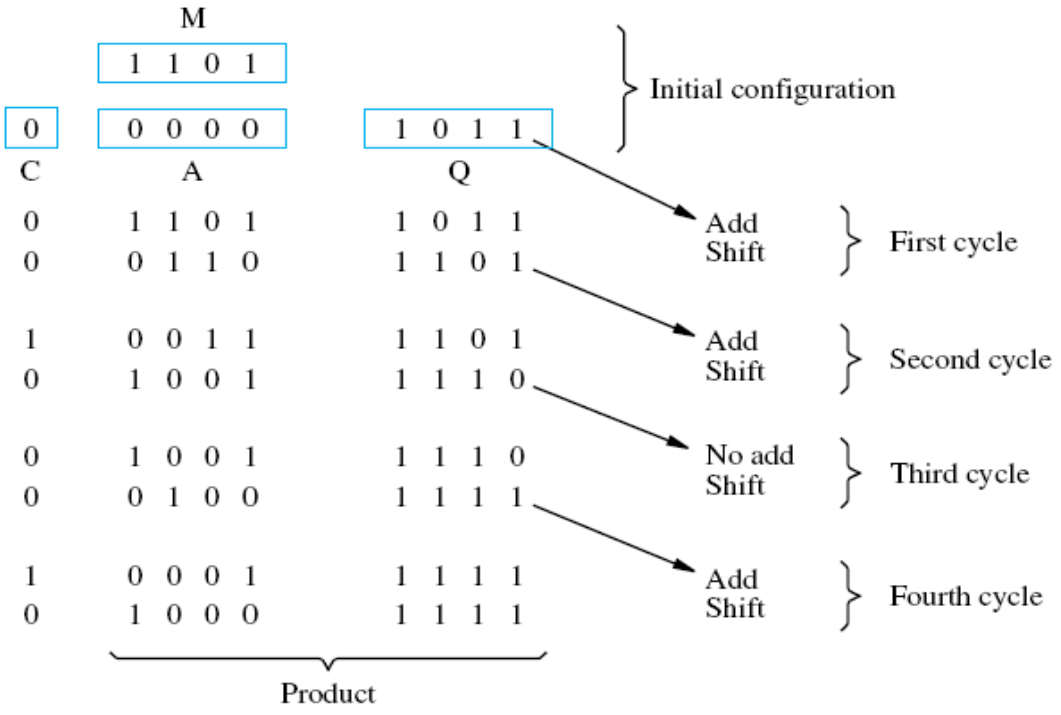
1) The main component in each cell is a full adder(FA)..
2)The AND gate in each cell determines whether a multiplicand bit m_j, is added to the incoming partial- product bit, based on the value of the multiplier bit q_i (Figure 2.21).

SEQUENTIAL CIRCUIT BINARY MULTIPLIER

1) Registers A and Q combined hold PP_i(partial product)
while the multiplier bit q_i generates the signal Add/Noadd.
2) The carry-out from the adder is stored in flip-flop C (Figure 9.7).
3) Procedure for multiplication:
   1) Multiplier is loaded into register Q, Multiplicand is loaded into register M and C & A are cleared to 0.
   2) If q_0=1, add M to A and store sum in A. Then C, A and Q are shifted right one bit-position. If q_0=0, no addition performed and C, A & Q are shifted right one bit-position.
   3) After n cycles, the high-order half of the product is held in register A and the low-order half is held in register Q.



(a) Register configuration



(b) Multiplication example

Figure 2.22 Sequential circuit binary multiplier.

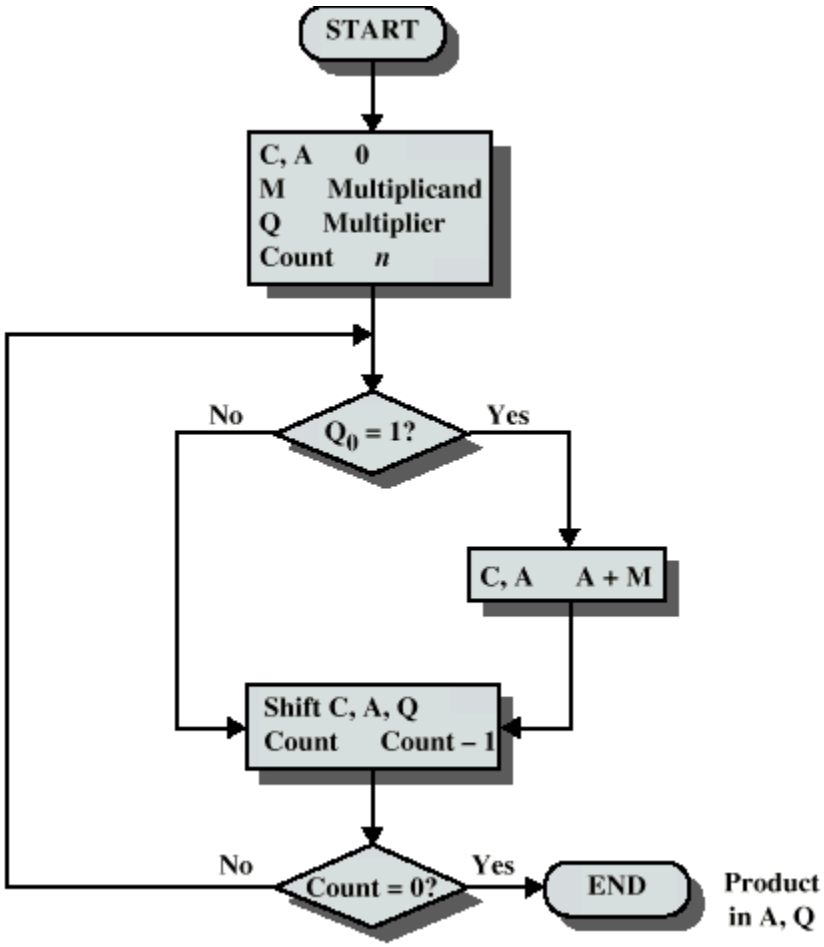


Figure 2.23 Flowchart for unsigned binary multiplication

MULTIPLICATION OF SIGNED NUMBERS

For a negative multiplier, a straightforward solution is to form the 2's-complement of both the multiplier and the multiplicand and proceed as in the case of a positive multiplier. This is possible because complementation of both operands does not change the value or the sign of the product. A technique that works equally well for both negative and positive multipliers, called the Booth algorithm, is described next.

The Booth Algorithm

The Booth algorithm generates a 2*n*-bit product and treats both positive and negative 2'scomplement *n*-bit operands uniformly. The Booth algorithm has two attractive features. First, it handles both positive and negative multipliers uniformly. Second, it achieves some efficiency in the number of additions required when the multiplier has a few large blocks of 1s.

Booth multiplier recoding table.

Multiplier		Version of multiplicand selected by bit i
Bit i	Bit $i-1$	
0	0	$0 \times M$
0	1	$+1 \times M$
1	0	$-1 \times M$
1	1	$0 \times M$

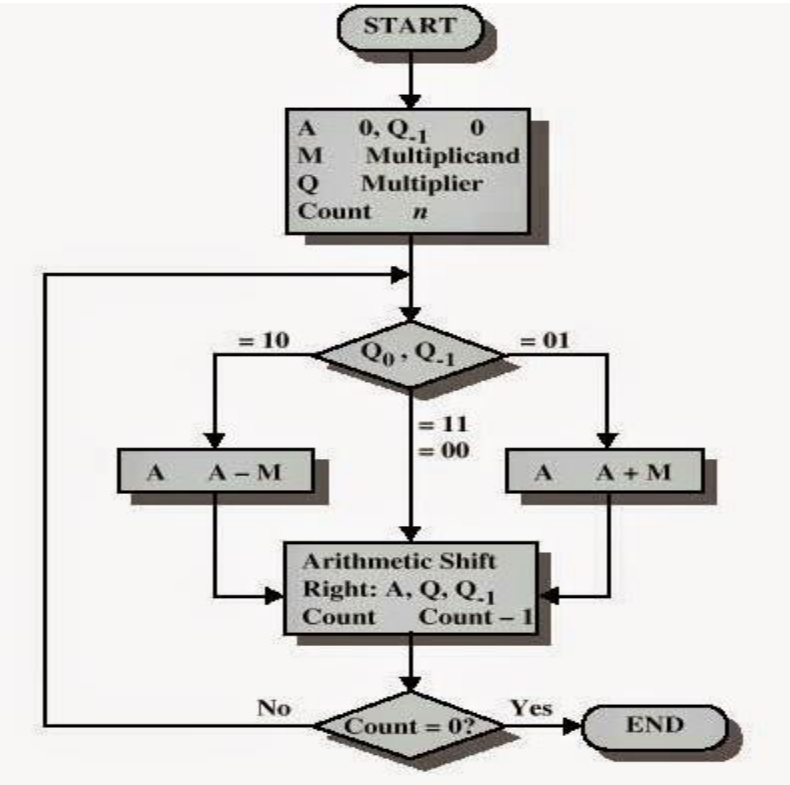


Figure 2.24 Booth's Algorithm for twos complement multiplication.

A	Q	Q_{-1}	M	comment
0000	0011	0	0111	Initial value
1001	0011	0	0111	$Q_0,Q_{-1}=10$ A<- A-M

1100	1001	1	0111	Shift right A,Q,Q_{-1}
1110	0100	1	0111	Q_0,Q_{-1}=11 Shift right A,Q,Q_{-1}
0101	0100	1	0111	Q_0,Q_{-1}=01 A<- A+M
0010	1010	0	0111	Shift right A,Q,Q_{-1}
0001	0101	0	0111	Q_0,Q_{-1}=00 Shift right A,Q,Q_{-1}

Answer(AQ)=00010101

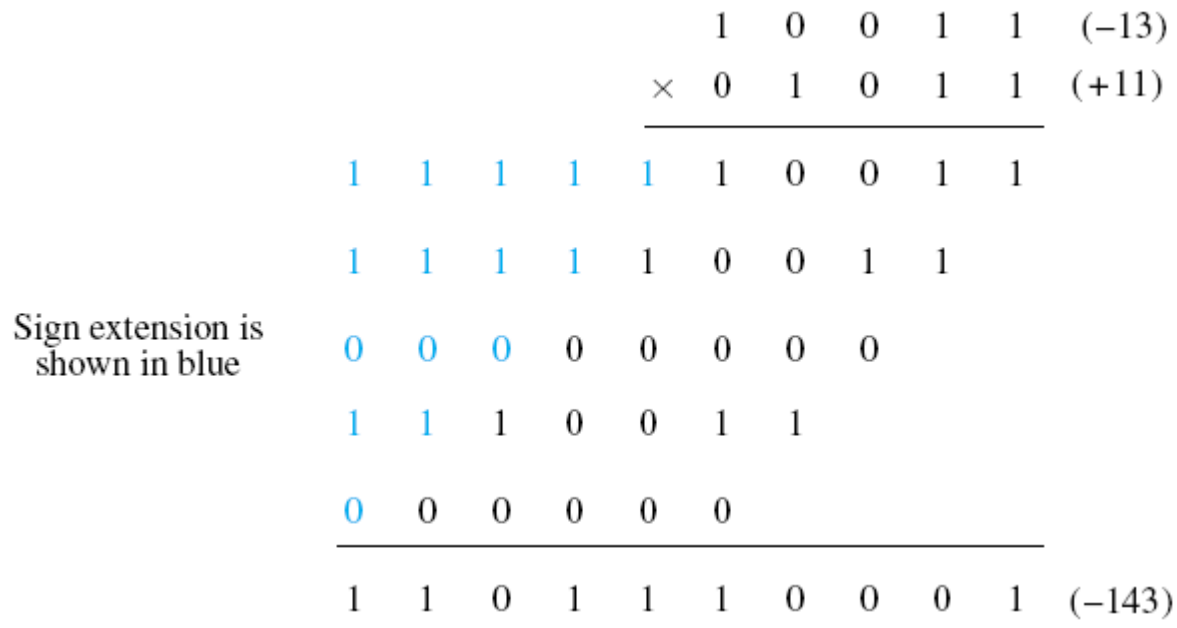
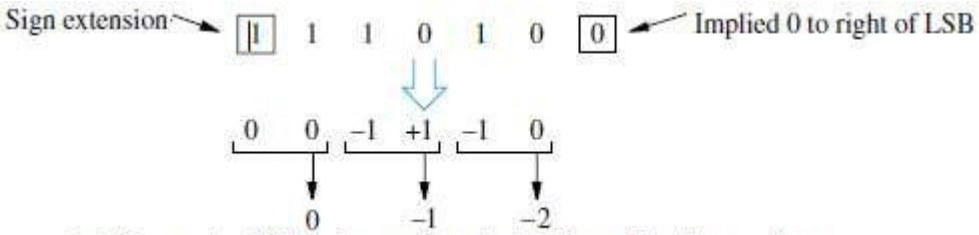


Figure 2.25 Sign extension of negative multiplicand.

FAST MULTIPLICATION
BIT-PAIR RECODING OF MULTIPLIERS

1) This method
→ derived from the booth algorithm
→ reduces the number of summands by a factor of 2
2) Group the Booth-recoded multiplier bits in pairs. (Figure 9.14 & 9.15).
3) The pair (+1 -1) is equivalent to the pair (0 +1).



(a) Example of bit-pair recoding derived from Booth recoding

Multiplier bit-pair		Multiplier bit on the right	Multiplicand selected at position i
$i+1$	i	$i-1$	
0	0	0	$0 \times M$
0	0	1	$+1 \times M$
0	1	0	$+1 \times M$
0	1	1	$+2 \times M$
1	0	0	$-2 \times M$
1	0	1	$-1 \times M$
1	1	0	$-1 \times M$
1	1	1	$0 \times M$

(b) Table of multiplicand selection decisions

Figure 2.26 Multiplier bit-pair recoding.

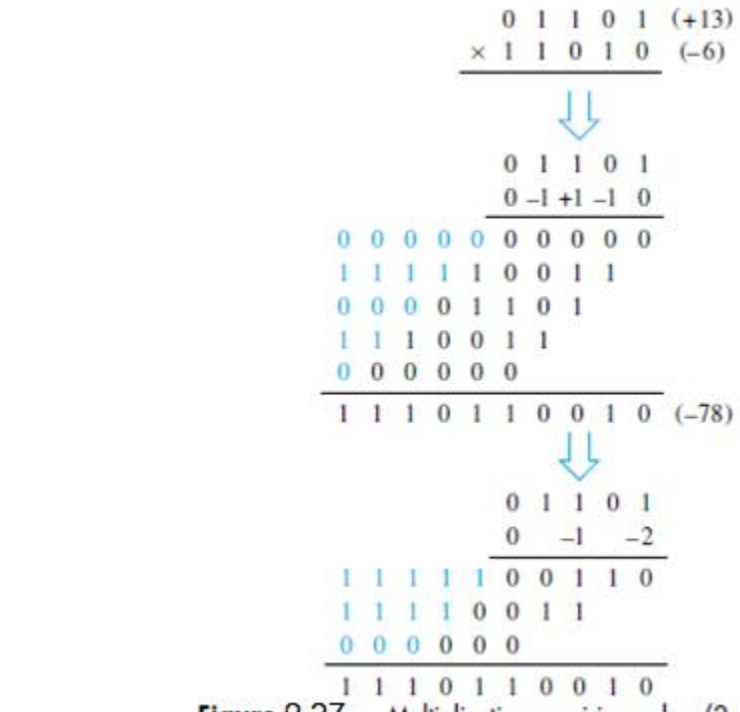


Figure 2.27 Multiplication requiring only $n/2$ summands.

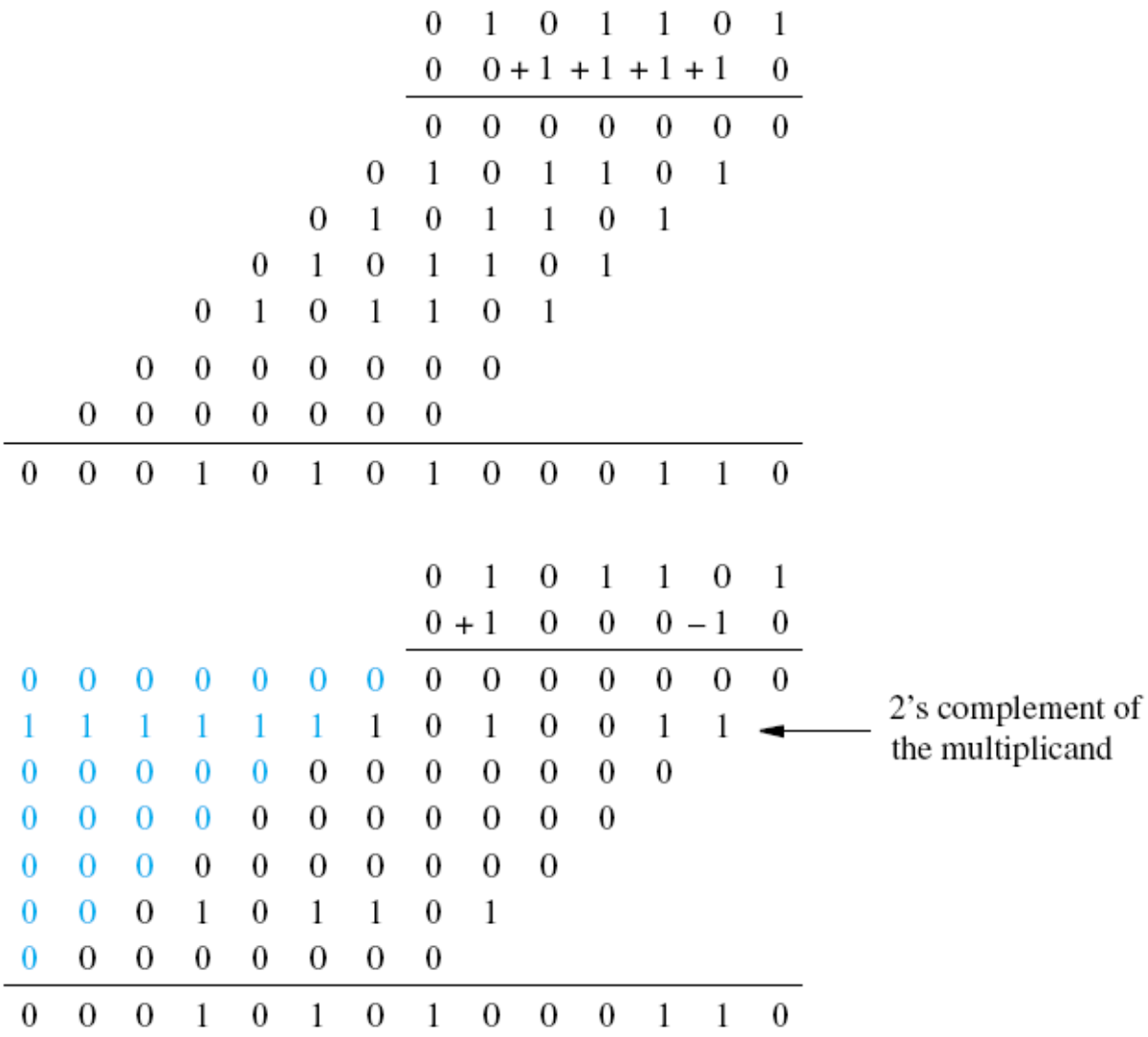


Figure 2.28 Normal and Booth multiplication schemes

Multiplication using Booth's Algorithm (10x3)

1. Start
2. Multiplicant =10
3. Multiplier=3
4. Binary of 10

   1 0 1 0
5. Binary of 3

   0 0 1 1
6. Two's complement of 3

   One's complement + 1

   1 1 0 0 +

   1

   = 1 1 0 1

7. Two's complement of 10

	1	1	1	1	0	1	0	1 +
								1
=	1	1	1	1	0	1	1	0

8. Multiplicant 1 0 1 0

Multiplier 1 1 0 1

9. Booth code for multiplier using

0	0	0 x M
0	1	+1 x M
1	0	-1 x M
1	1	0 x M

1	1	0	1	0
0	-1	+1	-1	

10. Multiply

-1 => Two's complement of Multiplicant

+1 => Multiplicant

					1	0	1	0	
					0	-1	+1	-1	
		1	1	1	1	0	1	1	0
		0	0	0	1	0	1	0	
		1	1	0	1	1	0		
		0	0	0	0	0			
	1	1	1	1	0	0	0	1	0

11. Two's complement of the answer

		1	1	1	1	0	0	0	1	0
=>		0	0	0	0	1	1	1	0	1 +
										1
		0	0	0	0	1	1	1	1	0

12. Decimal equivalent of Two's complement

$2^4 + 2^3 + 2^2 + 2^1$

16 + 8 + 4 + 2

10 x 3 = 30

13. Stop

Multiplication using Booth's Algorithm 10x-3

1. Start
2. Multiplicant =10
3. Multiplier=-3
4. Binary of 10

0 1 0 1 0

5. Binary of -3

1 0 0 1 1

6. Two's complement of 3

One's complement + 1

1 1 0 0 +

1

= 1 1 0 1

1 1 1 0 1

7. Two's complement of 10

0 0 0 0 1 0 1 0

1 1 1 1 0 1 0 1 +

1

= 1 1 1 1 0 1 1 0

8. Multiplicant 0 1 0 1 0

Multiplier 1 1 1 0 1

9. Booth code for multiplier using

0 0 0 x M

0 1 +1 x M

1 0 -1 x M

1 1 0 x M

1 1 1 0 1 0

0 0 -1 +1 -1

10. Multiply

-1 => Two's complement of Multiplicant

+1 => Multiplicant

						0	1	0	1	0
						0	0	-1	+1	-1
	1	1	1	1	1	1	0	1	1	0
	0	0	0	0	0	1	0	1	0	
	1	1	1	1	0	1	1	0		
	0	0	0	0	0	0	0			
	0	0	0	0	0	0				
1	1	1	1	1	1	0	0	0	1	0

11. Two's complement of the answer

	1	1	1	1	1	0	0	0	1	0	
=>	1	0	0	0	0	1	1	1	0	1	+
										1	
	1	0	0	0	0	1	1	1	1	0	

12. Decimal equivalent of Two's complement

$$- \quad 2^4 + 2^3 + 2^2 + 2^1$$

$$16 + 8 + 4 + 2$$

10 x 3 = -30

13. Stop

RESTORING AND NON-RESTORING DIVISION

Figure 2.29 shows examples of decimal division and binary division of the same values. A circuit that implements division by this longhand method operates as follows: It positions the divisor appropriately with respect to the dividend and performs a subtraction. If the remainder is zero or positive, a quotient bit of 1 is determined, the remainder is extended by another bit of the

dividend, the divisor is repositioned, and another subtraction is performed. If the remainder is negative, a quotient bit of 0 is determined, the dividend is restored by adding back the divisor, and the divisor is repositioned for another subtraction. This is called the *restoring division* algorithm.

```
      21                    10101
13 ) 274          1101 ) 100010010
     26                    1101
     14                    10000
     13                     1101
      1                     1110
                            1101
                               1
```

Figure 2.29 Longhand division examples.

Restoring Division

Figure 2.30 shows a logic circuit arrangement that implements the restoring division algorithm

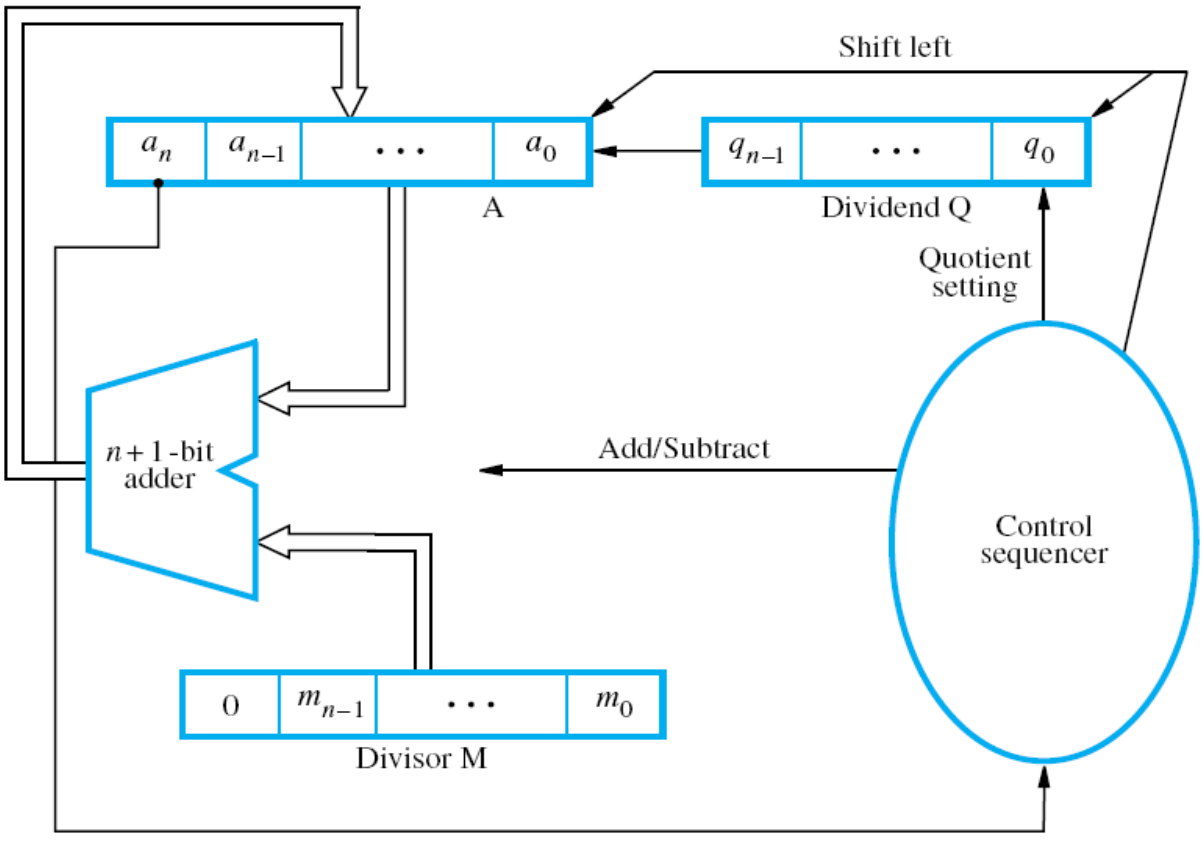


Figure 2.30 Circuit arrangement for binary division.

An n-bit positive divisor is loaded into registerMand an n-bit positive dividend is loaded into register Q at the start of the operation. Register A is set to 0. After the division is complete, the n-bit quotient is in register Q and the remainder is in register A. The required subtractions are facilitated by using 2's-complement arithmetic. The extra bit position at the left end of both A and M accommodates the sign bit during subtractions. The following algorithm performs restoring division.

Do the following three steps n times:

1. Shift A and Q left one bit position.

2. Subtract M from A, and place the answer back in A.

3. If the sign of A is 1, set q0 to 0 and add M back to A (that is, restore A); otherwise, set

q0 to 1.

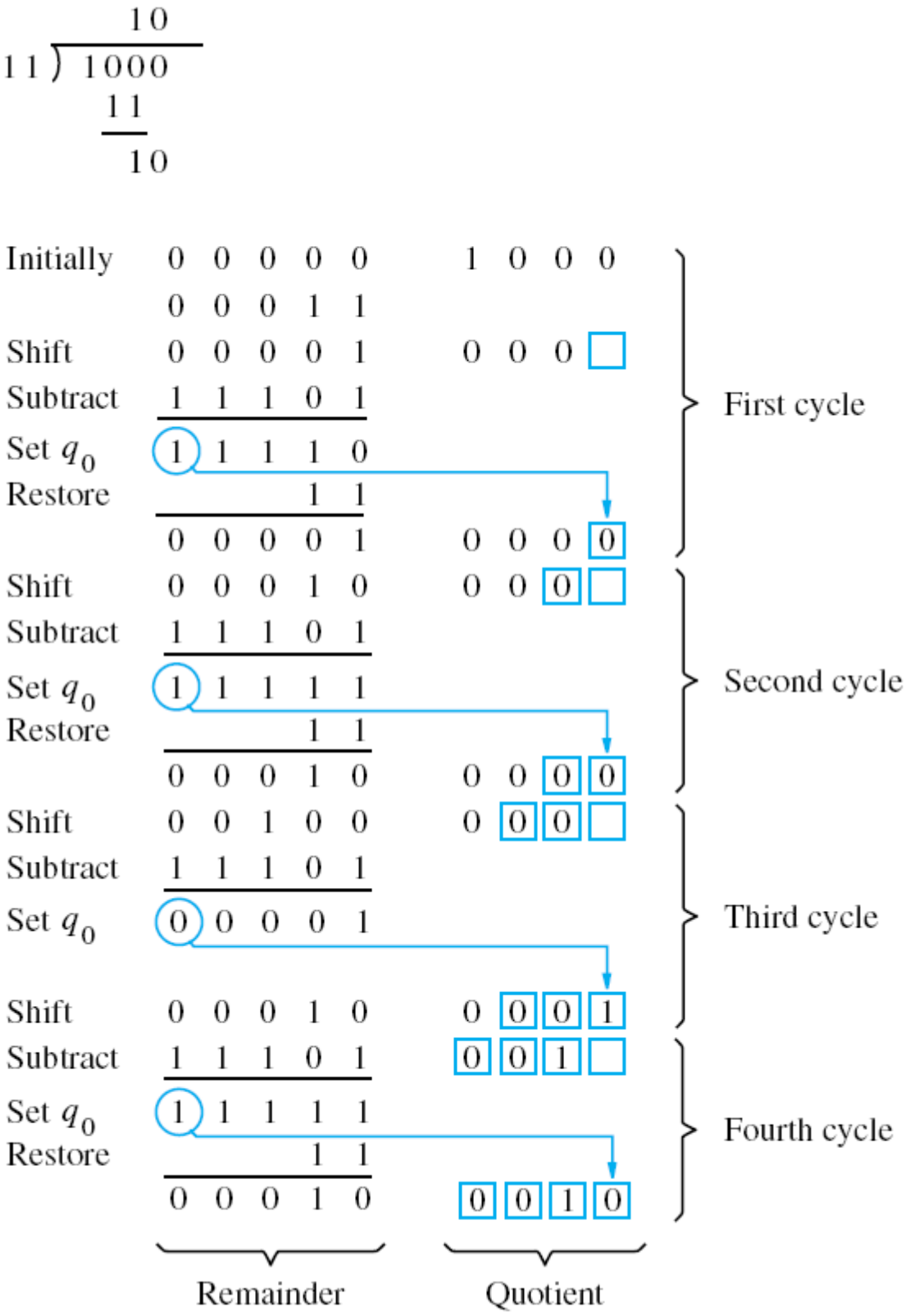


Figure 2.31 A restoring division example

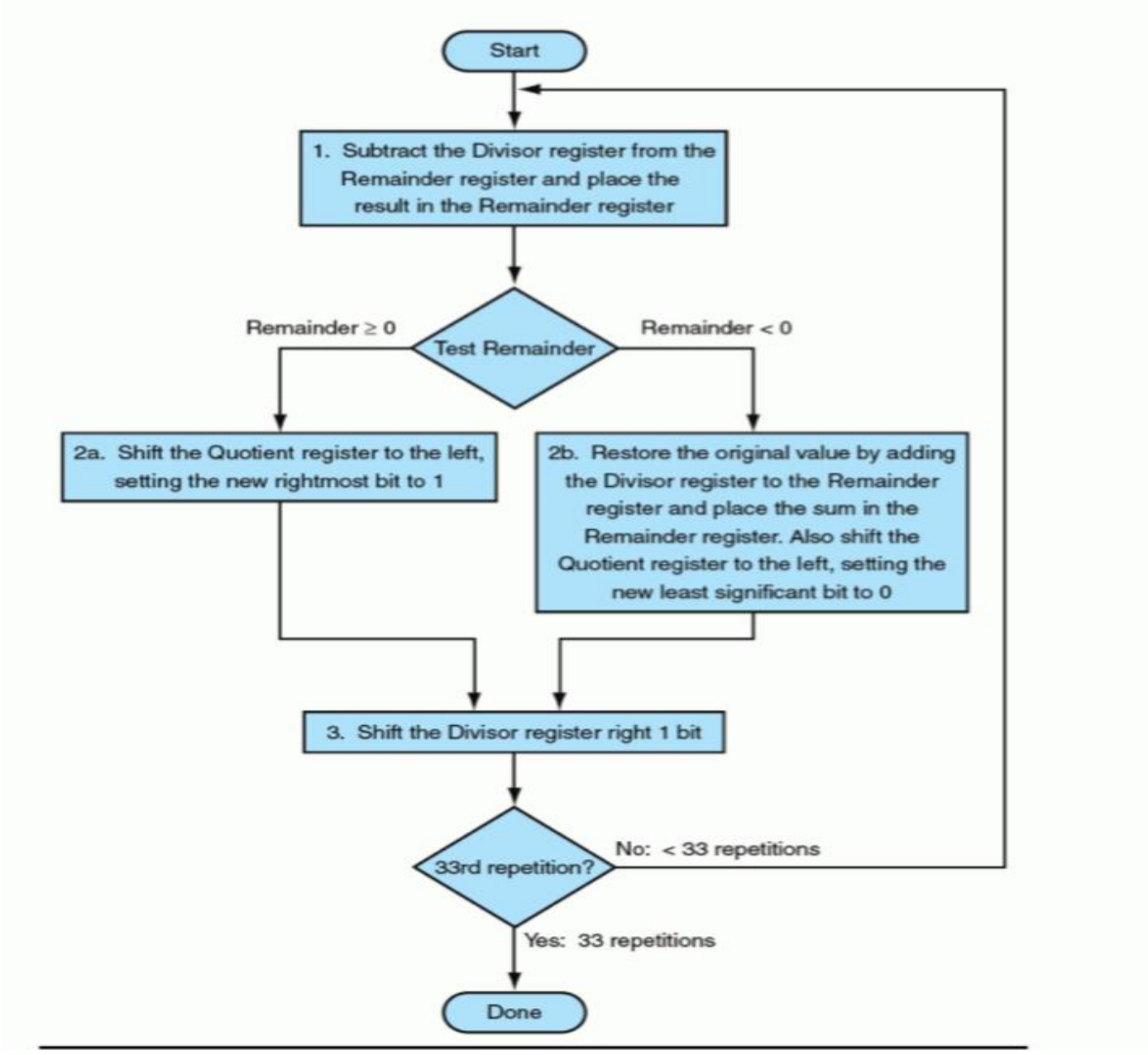


Figure 2.32 Flowchart for restoring division algorithm.

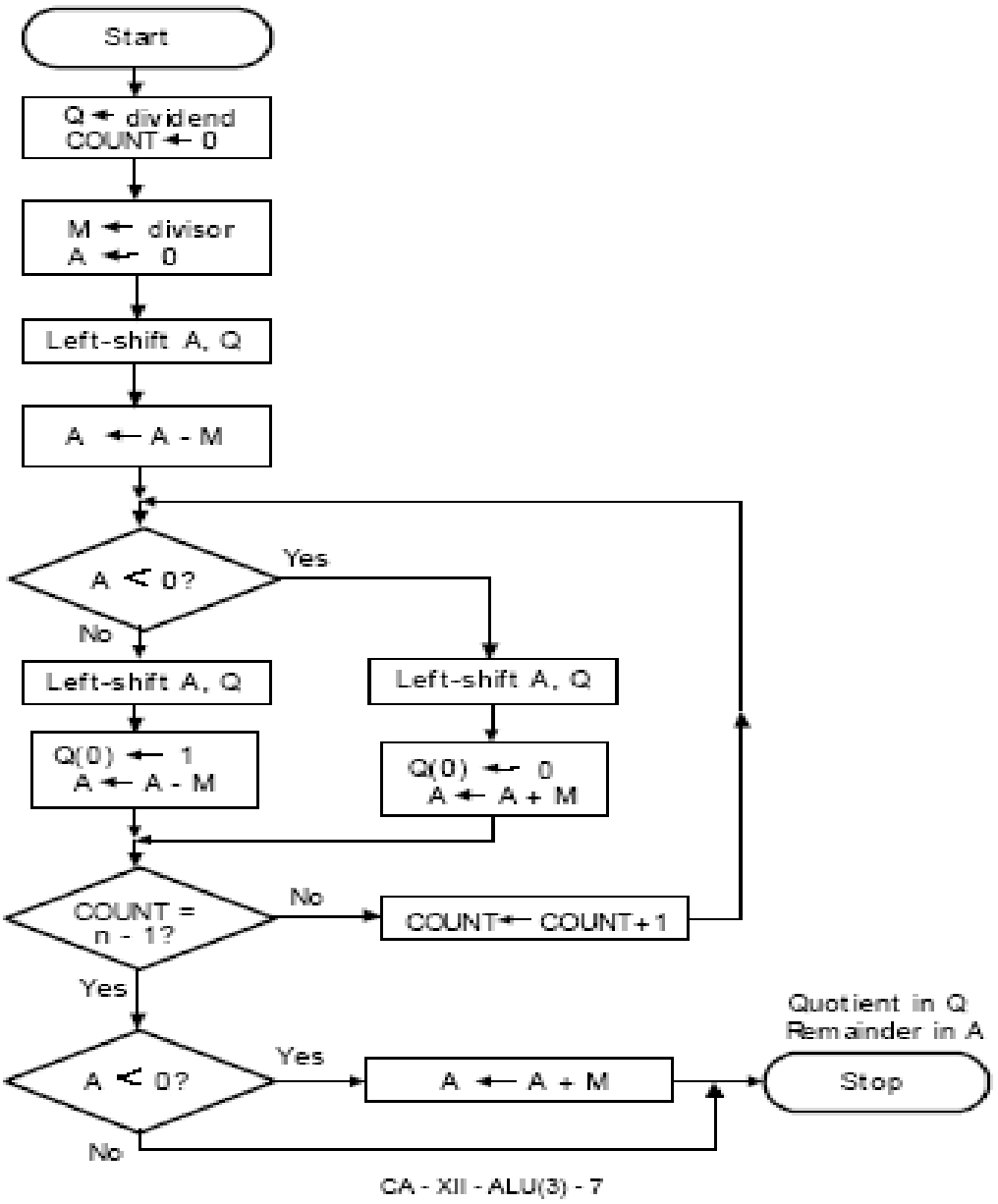


Figure 2.33 Flow chart for restoring division.

Non-Restoring Division

The restoring division algorithm can be improved by avoiding the need for restoring A after an unsuccessful subtraction. Subtraction is said to be unsuccessful if the result is negative. Consider the sequence of operations that takes place after the subtraction operation in the preceding algorithm. If A is positive, we shift left and subtract M, that is, we perform 2A− M. If A is negative, we restore it by performing A+ M, and then we shift it left and subtract M. This is equivalent to performing 2A+ M. The q0 bit is appropriately set to 0 or 1 after the correct operation has been performed. We can summarize this in the following algorithm for *non-restoring division*.

Stage 1: Do the following two steps n times:
1. If the sign of A is 0, shift A and Q left one bit position and subtract M from A; otherwise, shift A and Q left and add M to A.
2. Now, if the sign of A is 0, set q0 to 1; otherwise, set q0 to 0.
Stage 2: If the sign of A is 1, add M to A.
Stage 2 is needed to leave the proper positive remainder in A after the n cycles of Stage 1.
The logic circuitry in Figure 2.25 can also be used to perform this algorithm, except that the Restore operations are no longer needed. One Add or Subtract operation is performed in each of the n cycles of stage 1, plus a possible final addition in Stage 2.

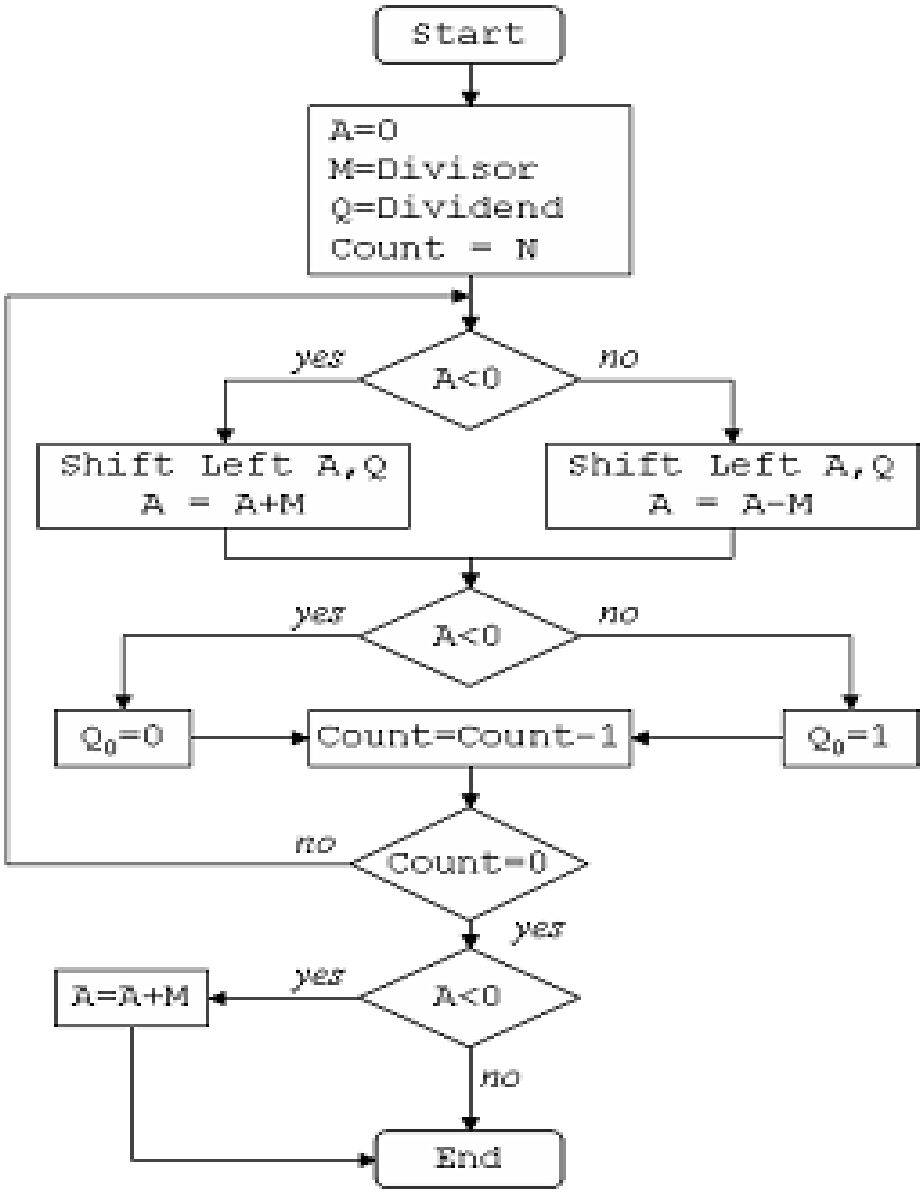


Figure 2.34 Flow chart for Non-Restoring division.

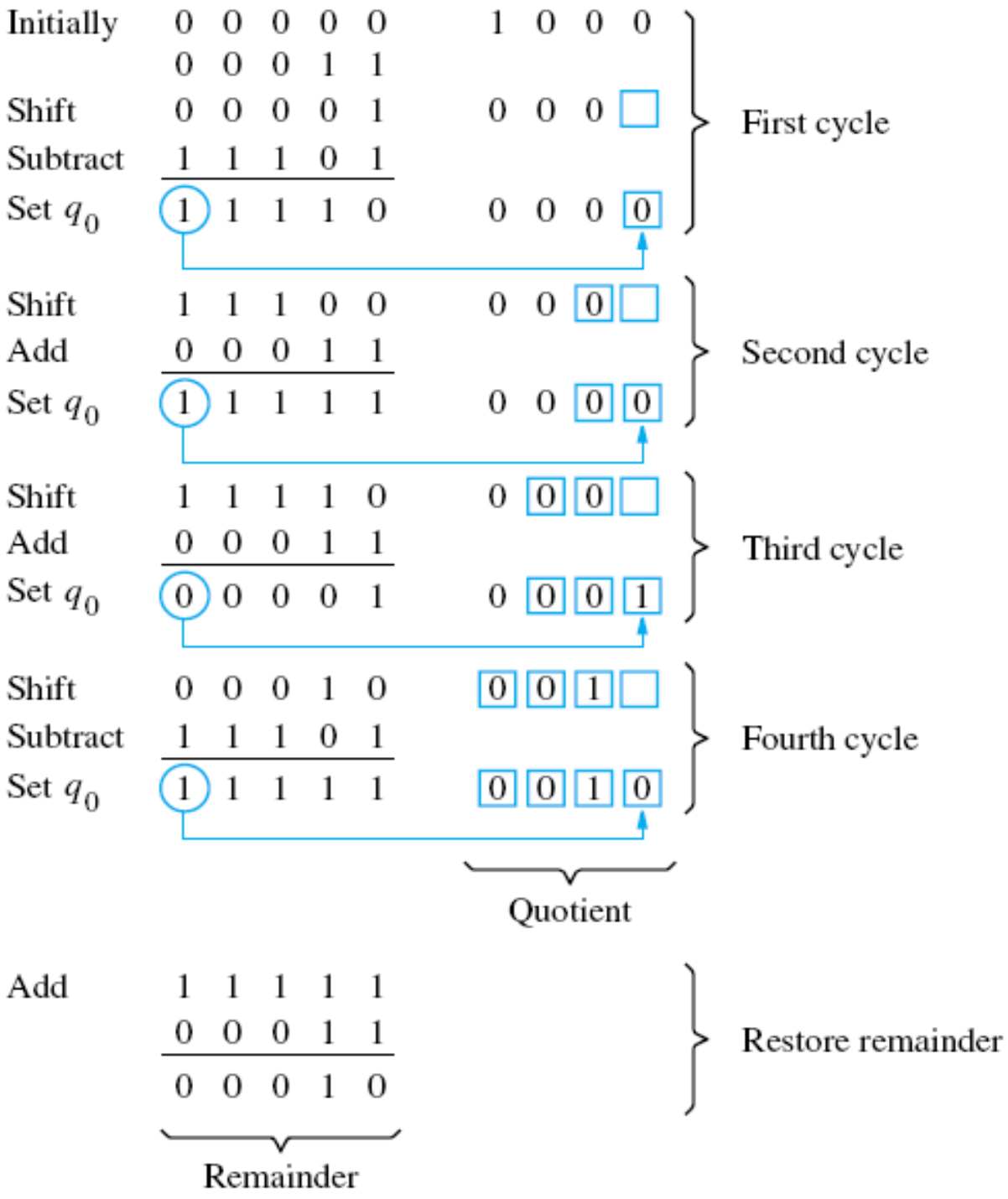
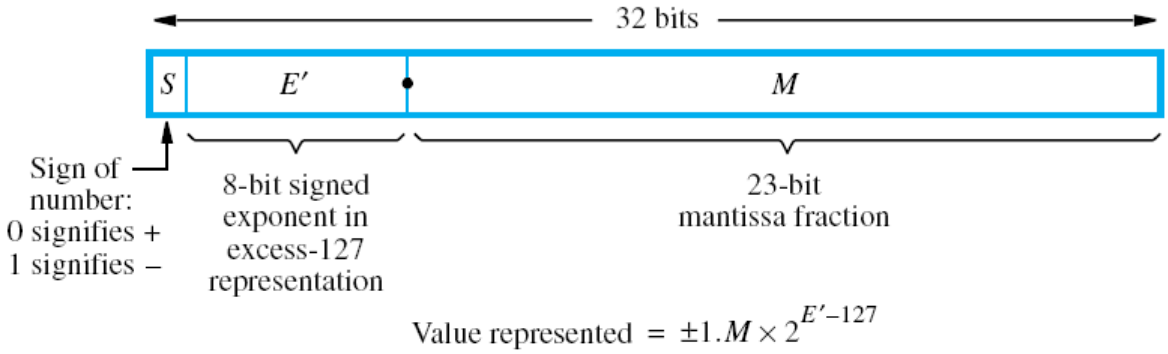


Figure 2.35 A non-restoring division example.

FLOATING-POINT NUMBERS AND OPERATIONS

A binary floating-point number can be represented by
• A sign for the number
• Some significant bits
• A signed scale factor exponent for an implied base of 2



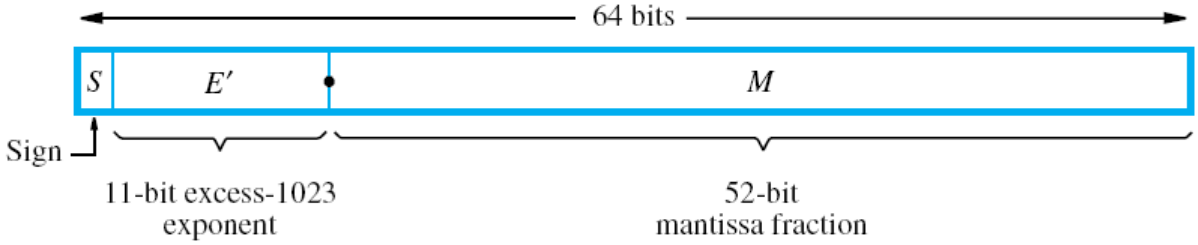
Value represented $= \pm 1.M \times 2^{E'-127}$

(a) Single precision



Value represented $= 1.001010\ldots0 \times 2^{-87}$

(b) Example of a single-precision number



Value represented $= \pm 1.M \times 2^{E'-1023}$

(c) Double precision

Figure 2.36 IEEE standard floating-point formats.

The basic IEEE format is a 32-bit representation, shown in Figure 2.36a. The leftmost bit represents the sign, S, for the number. The next 8 bits, E , represent the signed exponent of the scale factor (with an implied base of 2), and the remaining 23 bits, M, are the fractional part of

the significant bits. The full 24-bit string, *B*, of significant bits, called the *mantissa*, always has a leading 1, with the binary point immediately to its right. Therefore, the mantissa

$$B = 1.M = 1.b{-}1b{-}2 \ldots b{-}23$$

has the value

$$V(B) = 1 + b{-}1 \times 2{-}1 + b{-}2 \times 2{-}2 + \cdot \cdot \cdot + b{-}23 \times 2{-}23$$

By convention, when the binary point is placed to the right of the first significant bit, the number is said to be *normalized*. Note that the base, 2, of the scale factor and the leading 1 of the mantissa are both fixed. They do not need to appear explicitly in the representation. Instead of the actual signed exponent, *E*, the value stored in the exponent field is an unsigned integer $E = E + 127$. This is called the *excess*-127 format. Thus, *E* is in the range $0 \leq E \leq 255$. The end values of this range, 0 and 255, are used to represent special values, as described later. Therefore, the range of *E* for normal values is $1 \leq E \leq 254$. This means that the actual exponent, *E*, is in the range $-126 \leq E \leq 127$. The use of the excess-127 representation for exponents simplifies comparison of the relative sizes of two floating-point numbers.

The 32-bit standard representation in Figure 2.27a is called a single-precision representation because it occupies a single 32-bit word. The scale factor has a range of 2^{-126} to 2^{+127}, which is approximately equal to $10^{\pm 38}$. The 24-bit mantissa provides approximately the same precision as a 7-digit decimal value. An example of a single-precision floating-point number is shown in Figure 2.36b.

To provide more precision and range for floating-point numbers, the IEEE standard also specifies a double-precision format, as shown in Figure 2.36c. The double-precision format has increased exponent and mantissa ranges. The 11-bit excess-1023 exponent E has the range $1 \leq E \leq 2046$ for normal values, with 0 and 2047 used to indicate special values, as before. Thus, the actual exponent E is in the range $-1022 \leq E \leq 1023$, providing scale factors of 2^{-1022} to 2^{1023} (approximately $10^{\pm 308}$). The 53-bit mantissa provides a precision equivalent to about 16 decimal digits.

ARITHMETIC OPERATIONS ON FLOATING-POINT NUMBERS

The general procedures for addition, subtraction, multiplication, and division of floating-point numbers. The rules given below apply to the single-precision IEEE standard format. These rules specify only the major steps needed to perform the four operations; for example, the possibility that overflow or underflow might occur is not discussed. Furthermore, intermediate results for both mantissas and exponents might require more than 24 and 8 bits, respectively. These and other aspects of the operations must be carefully considered in designing an arithmetic unit that meets the standard.

When adding or subtracting floating-point numbers, their mantissas must be shifted with respect to each other if their exponents differ. Consider a decimal example in which we wish to add 2.9400×10^2 to 4.3100×10^4. We rewrite 2.9400×10^2 as 0.0294×10^4 and then perform addition of the mantissas to get 4.3394×10^4. The rule for addition and subtraction can be stated as follows:

Add/Subtract Rule

1. Choose the number with the smaller exponent and shift its mantissa right a number of steps equal to the difference in exponents.

2. Set the exponent of the result equal to the larger exponent.
3. Perform addition/subtraction on the mantissas and determine the sign of the result.
4. Normalize the resulting value, if necessary.
Multiplication and division are somewhat easier than addition and subtraction, in that no alignment of mantissas is needed.

Multiply Rule
1. Add the exponents and subtract 127 to maintain the excess-127 representation.
2. Multiply the mantissas and determine the sign of the result.
3. Normalize the resulting value, if necessary.

Divide Rule
1. Subtract the exponents and add 127 to maintain the excess-127 representation.
2. Divide the mantissas and determine the sign of the result.
3. Normalize the resulting value, if necessary.

Table 2.1 summarizes the basic operations for floating-point arithmetic. For addition and subtraction, it is necessary to ensure that both operands have the same exponent value. This may require shifting the radix point on one of the operands to achieve alignment. Multiplication and division are more straightforward.
A floating-point operation may produce one of these conditions:
• **Exponent overflow**: A positive exponent exceeds the maximum possible exponent value. In some systems, this may be designated as or
• **Exponent underflow**: A negative exponent is less than the minimum possible exponent value (e.g., is less than). This means that the number is too small to be represented, and it may be reported as 0.
• **Significand underflow**: In the process of aligning significands, digits may flow off the right end of the significand. As we shall discuss, some form of rounding is required.
• **Significand overflow**: The addition of two significands of the same sign may result in a carry out of the most significant bit.This can be fixed by realignment, as we shall explain.

Table 2.1 Floating-Point Numbers and Arithmetic Operations

Floating Point Numbers	Arithmetic Operations
$X = X_S \times B^{X_E}$ $Y = Y_S \times B^{Y_E}$	$X + Y = (X_S \times B^{X_E - Y_E} + Y_S) \times B^{Y_E}$ $X - Y = (X_S \times B^{X_E - Y_E} - Y_S) \times B^{Y_E}$ } $X_E \leq Y_E$ $X \times Y = (X_S \times Y_S) \times B^{X_E + Y_E}$ $\frac{X}{Y} = \left(\frac{X_S}{Y_S}\right) \times B^{X_E - Y_E}$

Examples:

$$X = 0.3 \times 10^2 = 30$$
$$Y = 0.2 \times 10^3 = 200$$

$$X + Y = (0.3 \times 10^{2-3} + 0.2) \times 10^3 = 0.23 \times 10^3 = 230$$
$$X - Y = (0.3 \times 10^{2-3} - 0.2) \times 10^3 = (-0.17) \times 10^3 = -170$$
$$X \times Y = (0.3 \times 0.2) \times 10^{2+3} = 0.06 \times 10^5 = 6000$$
$$X \div Y = (0.3 \div 0.2) \times 10^{2-3} = 1.5 \times 10^{-1} = 0.15$$

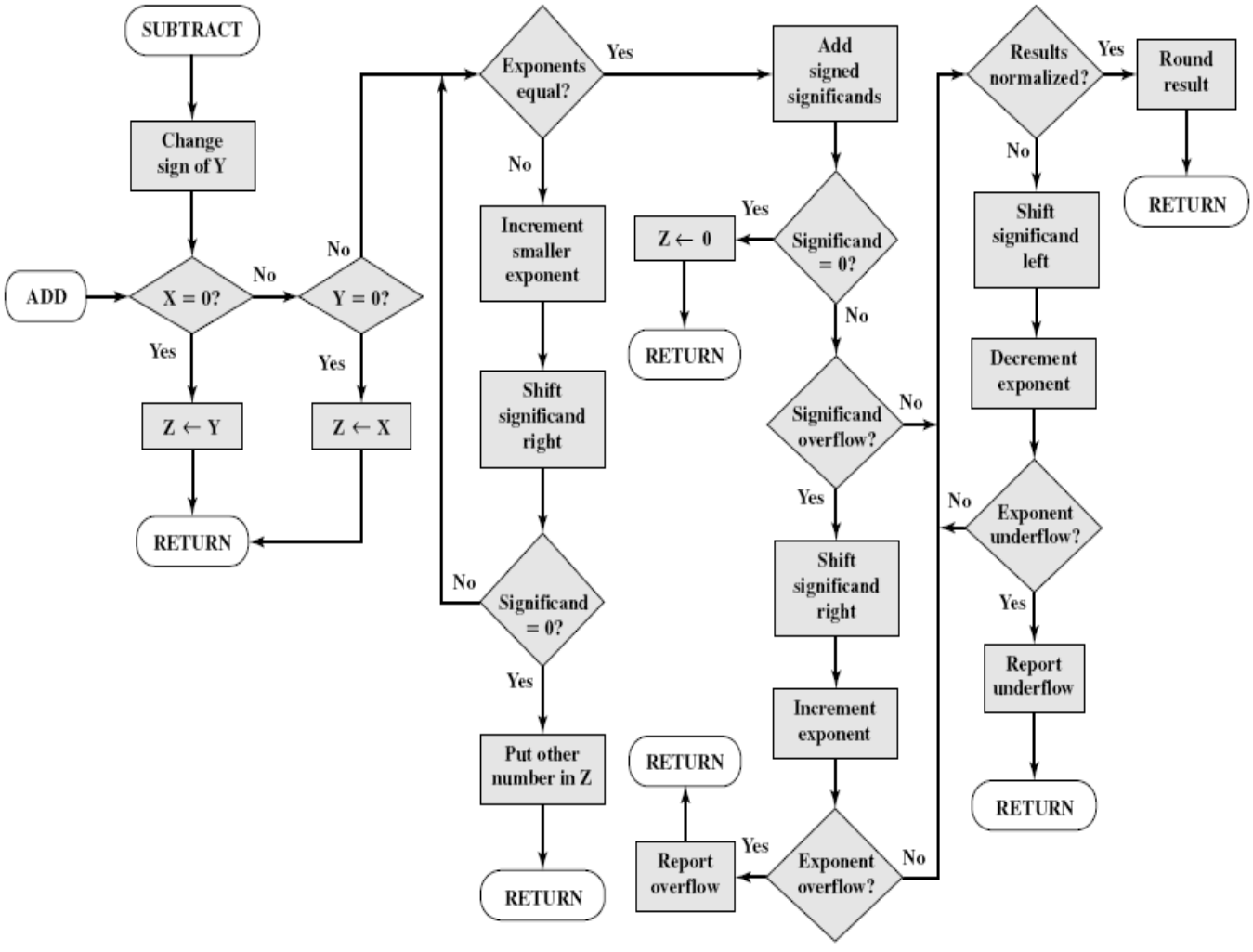


Figure 2.37 Floating-Point Addition and Subtraction $(Z \leftarrow Z \pm Y)$

Addition and Subtraction

In floating-point arithmetic, addition and subtraction are more complex than multiplication and division. This is because of the need for alignment. There are four basic phases of the algorithm for addition and subtraction:

1. Check for zeros.
2. Align the significands.
3. Add or subtract the significands.
4. Normalize the result.

For the addition or subtraction operation, the two operands must be transferred to registers that will be used by the ALU. If the floating-point format includes an implicit significand bit, that bit must be made explicit for the operation.

Phase 1: Zero check. Because addition and subtraction are identical except for a sign change, the process begins by changing the sign of the subtrahend if it is a subtract operation. Next, if either operand is 0, the other is reported as the result.

Phase 2: Significand alignment. The next phase is to manipulate the numbers so that the two exponents are equal.

To see the need for aligning exponents, consider the following decimal addition:

$$(123 \times 10^0) + (456 \times 10^{-2})$$

Clearly, we cannot just add the significands. The digits must first be set into equivalent positions, that is, the 4 of the second number must be aligned with the 3 of the first. Under these conditions, the two exponents will be equal, which is the mathematical condition under which two numbers in this form can be added. Thus,

$$(123 \times 10^0) + (456 \times 10^{-2}) = (123 \times 10^0) + (4.56 \times 10^0) = 127.56 \times 10^0$$

Alignment may be achieved by shifting either the smaller number to the right (increasing its exponent) or shifting the larger number to the left. Because either operation may result in the loss of digits, it is the smaller number that is shifted; any digits that are lost are therefore of relatively small significance. The alignment is achieved by repeatedly shifting the magnitude portion of the significand right 1 digit and incrementing the exponent until the two exponents are equal. (Note that if the implied base is 16, a shift of 1 digit is a shift of 4 bits.) If this process results in a 0 value for the significand, then the other number is reported as the result. Thus, if two numbers have exponents that differ significantly, the lesser number is lost.

Phase 3: Addition. Next, the two significands are added together, taking into account their signs. Because the signs may differ, the result may be 0. There is also the possibility of significand overflow by 1 digit. If so, the significand of the result is shifted right and the exponent is incremented. An exponent overflow could occur as a result; this would be reported and the operation halted.

Phase 4: Normalization. The final phase normalizes the result. Normalization consists of shifting significand digits left until the most significant digit (bit, or 4 bits for base-16 exponent) is nonzero. Each shift causes a decrement of the exponent and thus could cause an exponent underflow. Finally, the result must be rounded off and then reported. We defer a discussion of rounding until after a discussion of multiplication and division.

Multiplication and Division

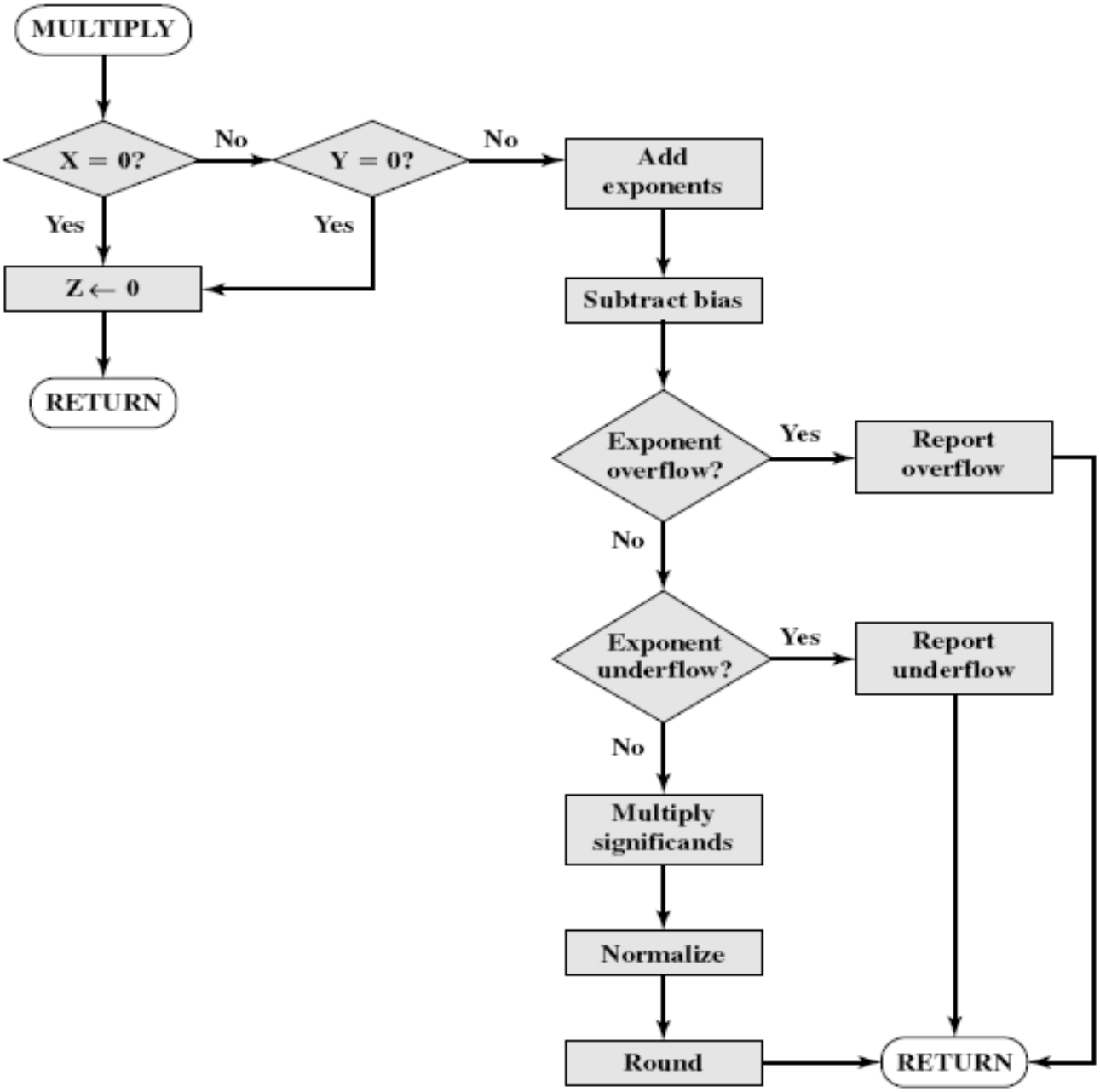


Figure 2.38 Floating-Point Multiplication $(Z \leftarrow X \times Y)$

Floating-point multiplication and division are much simpler processes than addition and subtraction, as the following discussion indicates.

In Figure 2.38. First, if either operand is 0, 0 is reported as the result. The next step is to add the exponents. If the exponents are stored in biased form, the exponent sum would have doubled the bias. Thus, the bias value must be subtracted from the sum. The result could be either an exponent overflow or underflow, which would be reported, ending the algorithm. If the exponent

of the product is within the proper range, the next step is to multiply the significands, taking into account their signs. The multiplication is performed in the same way as for integers. In this case, we are dealing with a sign–magnitude representation, but the details are similar to those for twos complement representation. The product will be double the length of the multiplier and multiplicand. The extra bits will be lost during rounding.

After the product is calculated, the result is then normalized and rounded, as was done for addition and subtraction. Note that normalization could result in exponent underflow.

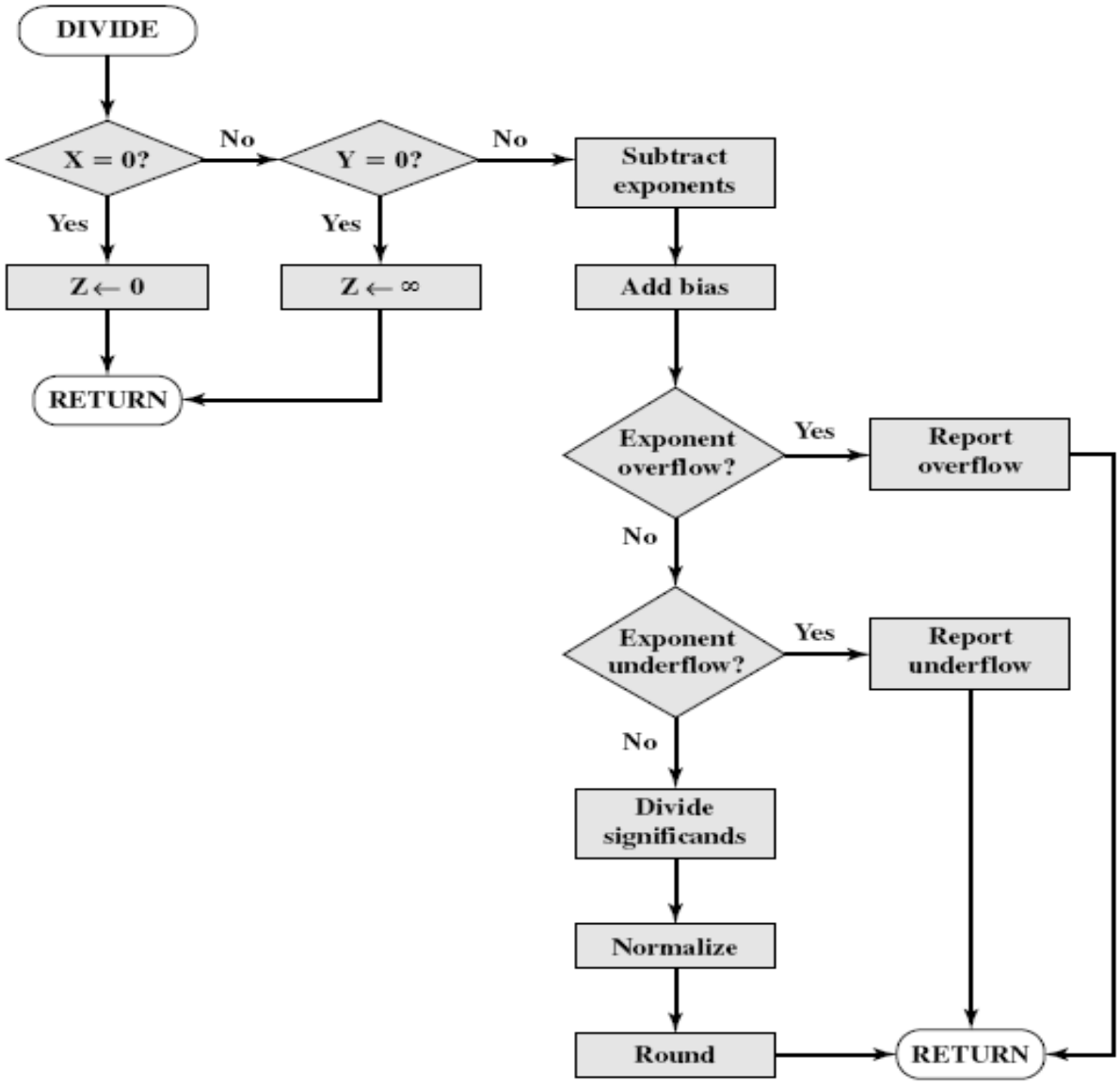


Figure 2.39 Floating-Point Division $(Z \leftarrow X/Y)$

Consider the flowchart for division depicted in Figure 2.39. Again, the first step is testing for 0. If the divisor is 0, an error report is issued, or the result is set to infinity, depending on the

implementation. A dividend of 0 results in 0. Next, the divisor exponent is subtracted from the dividend exponent. This removes the bias, which must be added back in. Tests are then made for exponent underflow or overflow. The next step is to divide the significands. This is followed with the usual normalization and rounding.

Module III

INPUT/OUTPUT ORGANIZATION

Computers is able to exchange data with other devices. Computers can be used to communicate with other computers over the Internet and access information around the globe. A general-purpose computer should have the ability to exchange information with a wide range of devices in varying environments. In different ways I/O operations can be performed.

Accessing I/O Devices

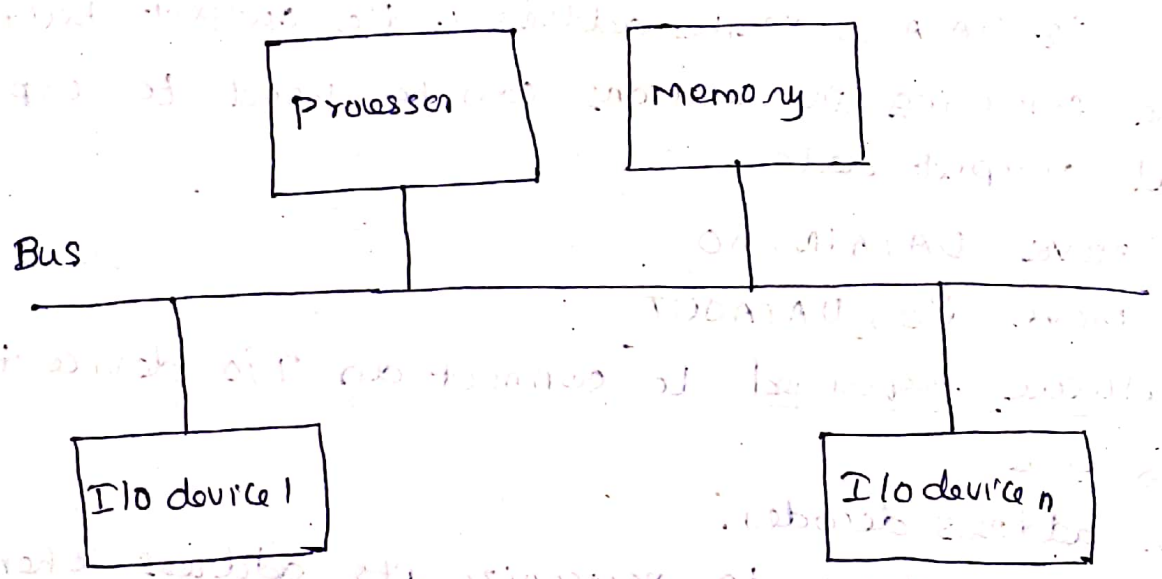


Figure 3.1 A single-bus architecture

1. Bus - exchange information among the connected devices.
   1. consists of three sets of lines used to carry
      a) Address, b) Data and c) control signals.
2. Each I/O device is assigned a unique set of addresses.
3. When the processor places a particular address on the address lines.

Scanned with CamScanner

a) The device that recognizes this address responds to the commands issued on the control lines.

b) The processor requests either a read or a write operation and the requested data are transferred over the data lines.

c) When I/O devices and the memory share the same address space, the arrangement is called memory-mapped I/O.

d) With memory-mapped I/O, any machine instruction that can access memory can be used to transfer data to or from an I/O device.

eg: If DATAIN is the address of the input buffer & If DATAOUT is the address of the output buffer

The following instructions can be used to input and output data.

```
Move  DATAIN, R0
Move  R0, DATAOUT
```

Hardware required to connect an I/O device to the bus are

1. The address decoder.
   Enables the device to recognize its address when this address appears on the address lines.
2. The data register:
   Holds the data being transferred to or from the processor.
3. The status register:
   Contains information relevant to the operation of the I/O device.

Scanned with CamScanner

Both the data and status registers are connected to the data bus and assigned unique addresses.

Interface circuit:

The address decoder, the data and status registers, and the control circuitry required to coordinate I/O transfers constitute the device's Interface circuit.

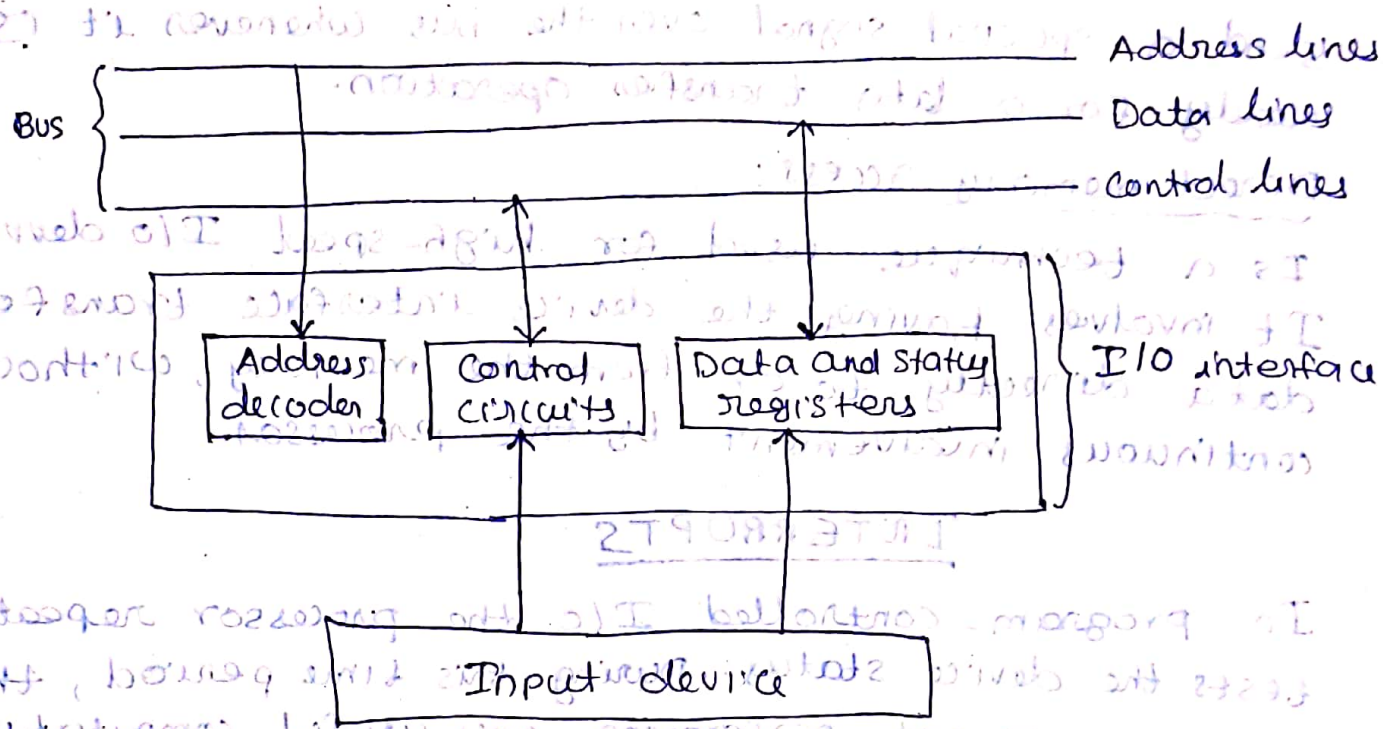


Figure 3.2 I/O interface for an input device.

1. I/O devices operate at speeds that are vastly different from that of the processor.
2. Program-controlled I/O, in which the processor repeatedly checks a status flag to achieve the required synchronization between the processor and an input or output device.
   eg: SIN, SOUT.

The processor polls the device using program-controlled I/O.

Scanned with CamScanner

Other methods or mechanisms for implementing I/O operations are

1. Interrupts and
2. Direct memory access.

Interrupts:

synchronization is achieved by having the I/O device send a special signal over the bus whenever it is ready for a data transfer operation.

Direct memory access:

Is a technique used for high-speed I/O devices. It involves having the device interface transfer data directly to or from the memory, without continuous involvement by the processor.

INTERRUPTS

In program-controlled I/O the processor repeatedly tests the device status. During this time period, the processor is not performing any useful computation. Other operations can be performed while waiting for an I/O device to become ready.

The I/O device can alert the processor when it becomes ready by sending a hardware signal called an interrupt to the processor.

Interrupt-request line of the bus control line is dedicated for interrupt.

Interrupt-service routine: The routine executed in response to an interrupt request.

Scanned with CamScanner

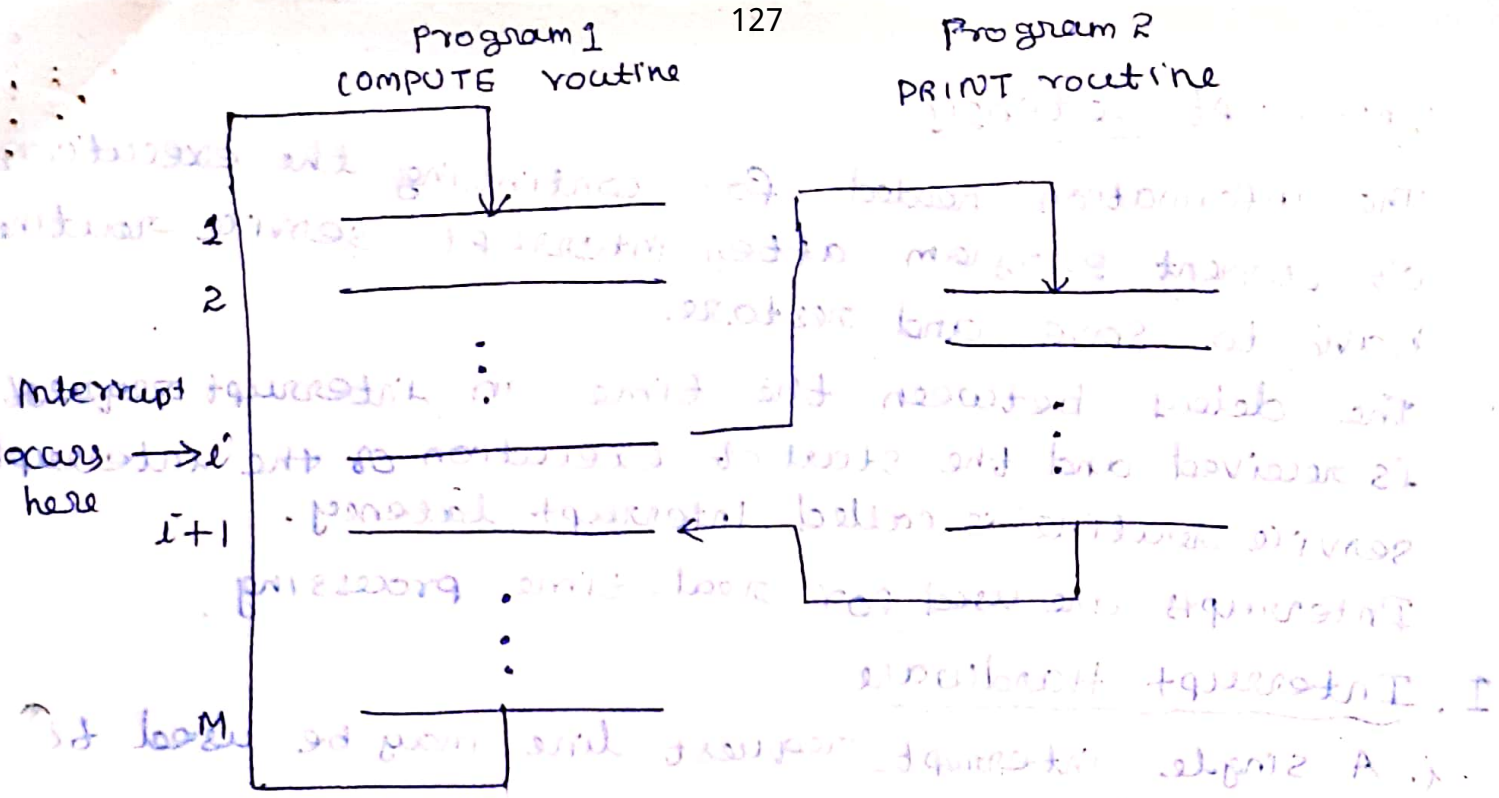


Figure 3.3 Transfer of control through the use of interrupts.

1. Assume that an interrupt request arrives during execution of instruction i in Figure 3.3.
   a) The processor first completes execution of instruction i.
   b) It loads the program counter with the address of the first instruction of the interrupt-service routine (ISR).
   c) The current content of the PC must be put in temporary storage in a known location before loading ISR's first instruction's address. The known location is normally a stack.

Interrupt-acknowledge signal

The processor inform the device that its request has been recognized so that it may remove its interrupt-request signal by using interrupt-acknowledge signal.

Scanned with CamScanner

Interrupt latency:

The information needed for continuing the execution of current program after interrupt service routine have to save and restore.

The delay between the time an interrupt request is received and the start of execution of the interrupt-service routine is called Interrupt latency.

Interrupts are used for real-time processing.

I. Interrupt Hardware

i. A single interrupt-request line may be used to serve n devices.

ii. All devices are connected to the line via switches to ground.

iii. To request an interrupt, a device closes its associated switch.

iv. If all interrupt-request signals INTR1 to INTRn are inactive, that is, if all switches are open, the voltage on the interrupt-request line will be equal to Vdd. This is inactive state of the line.

v. When a device requests an interrupt by closing its switch, the voltage on the line drops to 0, causing

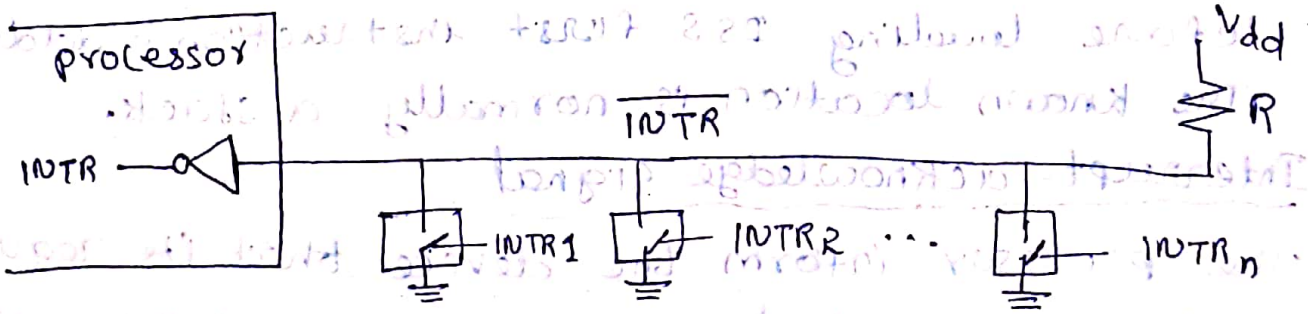


Figure 3.4 An equivalent circuit for an open-drain bus used to implement a common interrupt-request line.

Scanned with CamScanner

the interrupt-request signal, INTR, received by the processor to go to 1.

vi. The closing of one or more switches will cause the line voltage to drop to 0.

$INTR = INTR_1 + \cdots + INTR_n$

(+ : logical OR)

II ENABLING AND DISABLING INTERRUPTS

The programmer or the processor can enable or disable in the following three ways.

1. The processor hardware ignore the interrupt-request line until the execution of the first instruction of the interrupt-service routine has been completed.
   Interrupt disable - the first instruction of ISR
   Interrupt enable - the last " "
2. A simple processor with only one interrupt request line is to have the Processor automatically disable interrupts before starting the execution of the interrupt-service routine.
   processor status register (PS) contains one bit for interrupt-enable.
3. The processor has a special interrupt-request line for which the interrupt-handling circuit responds only to the leading edge of the signal - edge-triggered.

When the interrupts are enabled, the following happens:

Scanned with CamScanner

1. The device raises an interrupt request.
2. The processor interrupts the program currently being executed.
3. Interrupts are disabled by changing the control bits in the PS (except in the case of edge-triggered interrupts).
4. The device is informed that its request has been recognized, and in response, it deactivates the interrupt-request signal.
5. The action requested by the interrupt is performed by the interrupt-service routine.
6. Interrupts are enabled and execution of the interrupted program is resumed.

III HANDLING MULTIPLE DEVICES

The devices connected to the processor are operationally independent, there is no definite order in which they will generate interrupts. We should find solution for the following:

1. How can the processor recognize the device requesting an interrupt?
2. Given that different devices are likely to require different interrupt-service routines, how can the processor obtain the starting address of the appropriate routine in each case?
3. Should a device be allowed to interrupt the processor while another interrupt is being serviced?
4. How should two or more simultaneous interrupt requests be handled?

Scanned with CamScanner

1. polling
2. Vectored Interrupts
3. Interrupt Nesting.
4. Simultaneous Requests.

Polling

When a device raises an interrupt request, it sets to 1 one of the bits in its status register, which we will call the IRQ bit.

To identify the interrupting device is to have the interrupt service routine poll all the I/O devices connected to the bus. The first device encountered with its IRQ bit set is the device that should be serviced.

Disadvantage of polling scheme is the time spent interrogating the IRQ bits of all the devices that may not be requesting any service.

Vectored Interrupts

1. A device requesting an interrupt can identify itself by sending a special code to the processor over the bus. This type of interrupt-handling is known as Vectored interrupts.
2. The code supplied by the device may represent the starting address of the interrupt-service routine for that device.
3. The location pointed to by the interrupting device is used to store the starting address of the interrupt-service routine. This address is known as interrupt vector.
4. In most computers, I/O devices send the interrupt-vector code over the data bus, using the bus control signals to ensure that devices do not interfere with each other.

Scanned with CamScanner

Interrupt Nesting

1. A long delay in responding to an interrupt request may lead to erroneous operation in real-time application.
2. I/O devices should be organized in a priority structure.
3. An interrupt request from a high-priority device should be accepted while the processor is servicing another request from a lower-priority device.
4. The processor's priority is usually encoded in a few bits of the processor status word. It can be changed

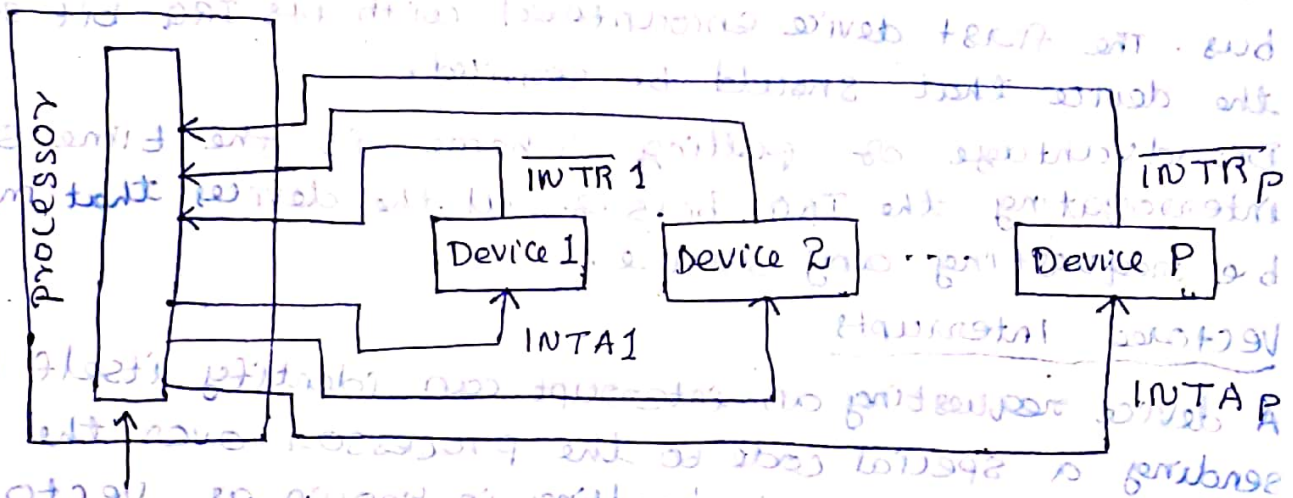


Figure 3.5 Implementation of interrupt priority using individual interrupt-request and acknowledge lines

by program instructions that write into the PS. These are privileged instructions, which can be executed only while the processor is running in the supervisor mode.

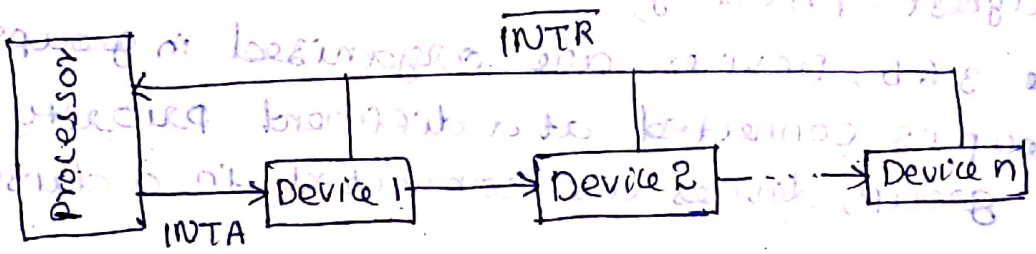
5. An attempt to execute a privileged instruction while in the user mode leads to a special type of interrupt called a privilege exception.

Scanned with CamScanner

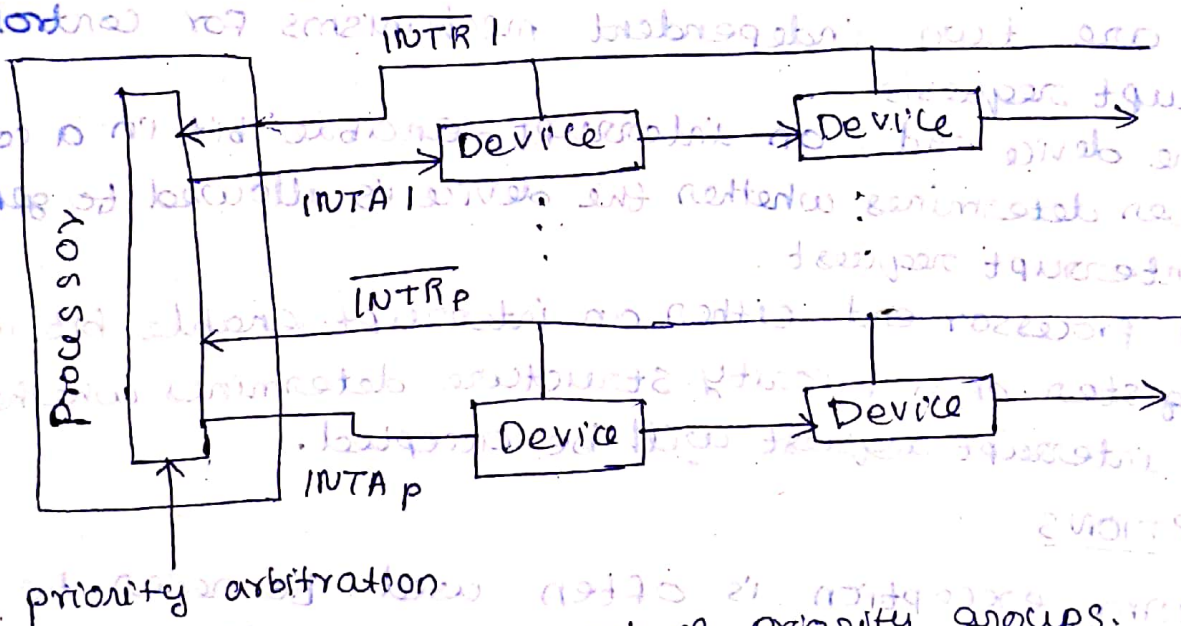
6. In Figure 3.5 each of the interrupt-request lines is assigned a different priority level. A request is accepted only if it has a higher priority level than that currently assigned to the processor.

Simultaneous Requests.

The processor must have some means of deciding which request to service first when there are simultaneous arrivals of interrupt requests from two or more devices. Solution: The processor simply accepts the request having the highest priority.



(a) Daisy Chain



(b) Arrangement of priority groups.

Figure 3.6 Interrupt priority schemes.

Scanned with CamScanner

In Figure 3.6a, when several devices raise an interrupt. request and the INTR line is activated, the processor responds by setting the INTA line to 1. This signal is received by device 1. Device 1 passes the signal onto device 2 only if it does not require any service. If device 1 has a pending request for interrupt, it blocks the INTA signal and proceeds to put its identifying code on the data lines.

Therefore, in the daisy-chain arrangement, the device that is electrically closest to the processor has the highest priority. The second device along the chain has second highest priority, and so on.

In Figure 3.6b, Devices are organised in groups, and each group is connected at a different priority level. Within a group, devices are connected in a daisy chain.

IV CONTROLLING DEVICE REQUESTS

There are two independent mechanisms for controlling interrupt requests.

At the device end, an interrupt-enable bit in a control register determines whether the device is allowed to generate an interrupt request.

At the processor end, either an interrupt enable bit in the PS register or a priority structure determines whether a given interrupt request will be accepted.

V EXCEPTIONS

The term exception is often used to refer to any event that causes an interruption.

Kinds of Exceptions

1. Recovery from Errors

Scanned with CamScanner

2. Debugging.
3. Privilege Exception.

1. Recovery from Errors.

The processor may also interrupt a program if it detects an error or an unusual condition while executing the instructions of the program. For example, the OP_code field of an instruction may not correspond to any legal instruction, or an arithmetic instruction may attempt a division by zero.

2. Debugging.

System software usually includes a program called a debugger, which helps the programmer find errors in a program. The debugger uses exceptions to provide two important facilities called trace and breakpoints. When a processor is operating in the trace mode, an exception occurs after execution of every instruction using the debugging program as the exception-service routine.

In Breakpoints, the program being debugged is interrupted only at specific points selected by the user.

3. Privilege Exception

To protect the operating system of a computer from being corrupted by user programs, certain instructions can be executed only while the processor is in the supervisor mode. These are called privileged instructions.

VI. USE OF INTERRUPTS IN OPERATING SYSTEMS

The operating system incorporates the interrupt-service routines for all devices connected to a computer. Application programs do not perform I/O operations themselves. When an application program needs an input or an

Scanned with CamScanner

output operation, it points to the data to be transferred and asks the OS to perform the operation. The OS suspends the execution of that program temporarily and performs the requested I/O operation. When the operation is completed, the OS transfers control back to the application program. The OS and the application program pass control back and forth using software interrupts.

DIRECT MEMORY ACCESS

To transfer large blocks of data at high speed, a special control unit is provided to allow transfer of a block of data directly between an external device and the main memory, without continuous intervention by the processor. This approach is called direct memory access, or DMA.

DMA controller – control circuit that is part of the I/O device interface.
- perform the functions for accessing the main memory.
- operations are controlled by the processor.

① To initiate the transfer of a block of words, the processor sends the starting address, the number of words in the block, and the direction of the transfer.
② On receiving this information, the DMA controller proceeds to perform the requested operation.
③ When the entire block has been transferred, the controller informs the processor by raising an interrupt signal.
④ While the DMA transfer is taking place, the program

Scanned with CamScanner

that requested the transfer cannot continue, and the processor can be used to execute another program.

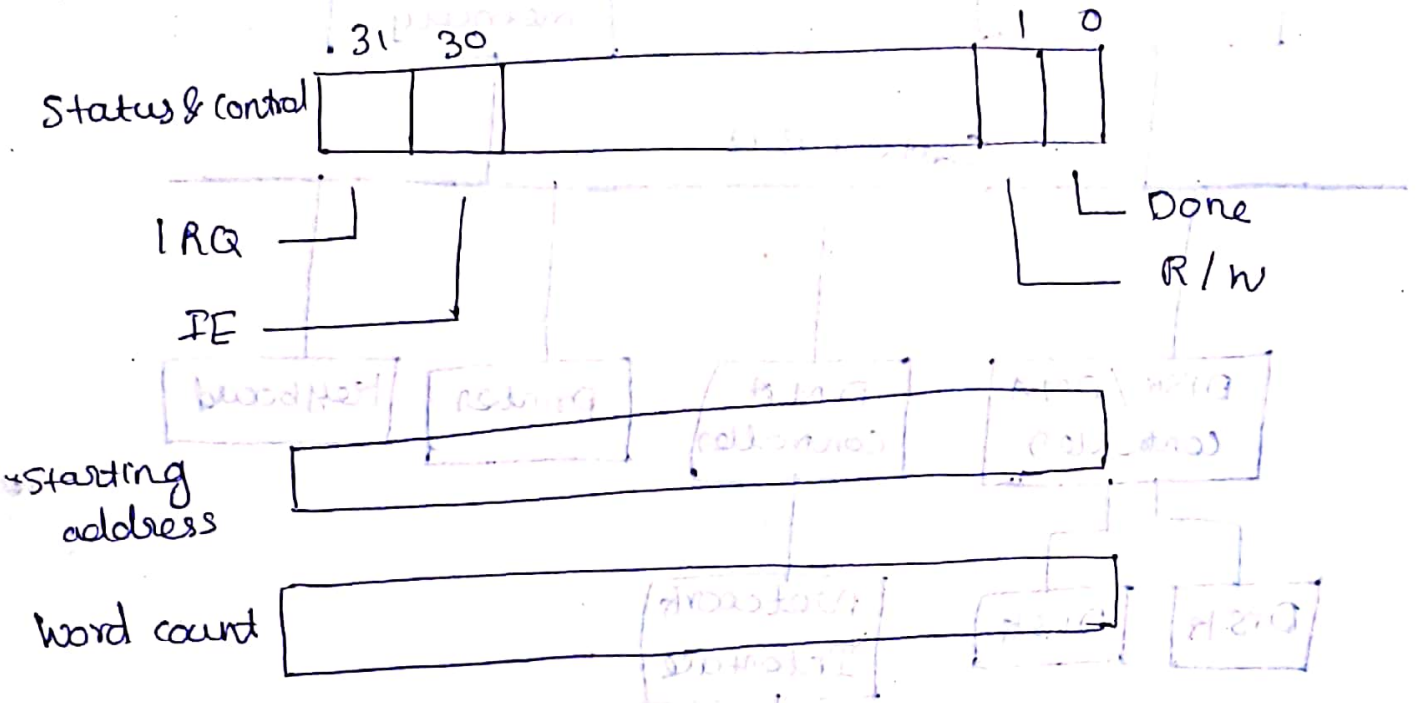


Figure 3.7 Registers in a DMA interface.

Figure 3.7 shows an example of the DMA controller registers that are accessed by the processor to initiate transfer operations.

1) Two registers are used for storing the starting address and the word count.
2) The third register contains status and control flags.
   - The R/$\overline{W}$ bit determines the direction of the transfer. 1 - read, 0 - write
     (1 - read: from memory to the I/O device; 0 - write: I/O device to memory)
3) IE - Interrupt Enabling, IRQ - Interrupt request.

Figure 3.8 shows a computer system with DMA.

1. A DMA controller connects a high-speed network to the computer bus.

Scanned with CamScanner

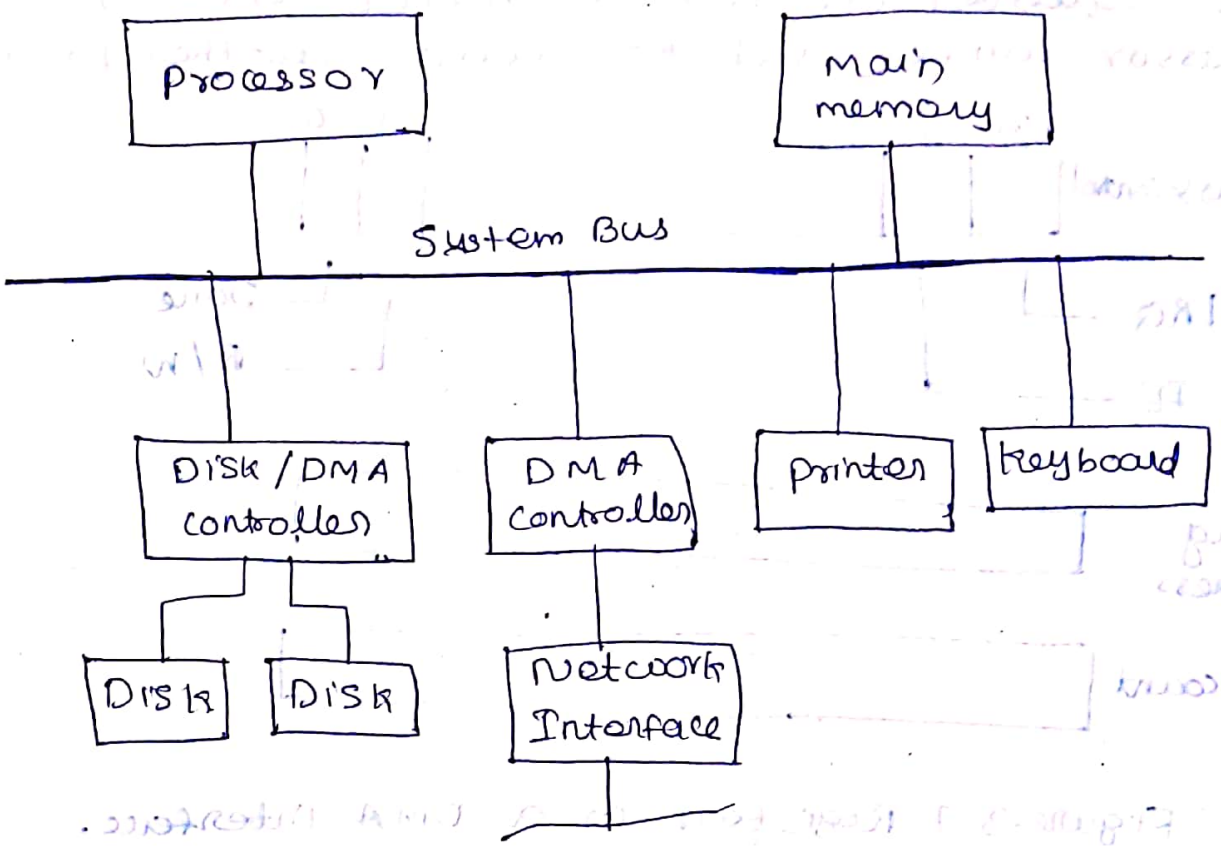


Figure 3.8 Use of DMA controllers in a computer system.

2. ~~A DMA controller connects a high-speed network to the comp~~

2. The disk controller, which controls two disks, also has DMA capability and provides two DMA channels.

To transfer a block of data from the main memory to one of the disks

i. A program writes the address and word count information into the registers of the corresponding channel of the disk controller.

ii) The DMA controller proceeds independently to implement the specified operation.

iii) When the DMA transfer is completed, this fact is

Scanned with CamScanner

recorded in the status and control register of the DMA channel by setting the Done bit.
- at the same time, if the IE bit is set, the controller sends an interrupt request to the processor and sets the IRQ bit.

iv) Requests by DMA devices for using the bus are always given higher priority than processor requests.

v) Cycle stealing: Since the processor originates most memory access cycles, the DMA controller can be said to "steal" memory cycles from the processor. Hence, this interweaving technique is usually called cycle stealing.

vi) Block or burst mode: The DMA controller may be given exclusive access to the main memory to transfer a block of data without interruption. This is known as block or burst mode.

vii) A conflict may arise if both the processor and a DMA controller or two DMA controllers try to use the bus at the same time to access the main memory. To resolve these conflicts, an arbitration procedure is implemented on the bus to coordinate the activities of all devices requesting memory transfers.

BUS ARBITRATION

Bus master:
The device that is allowed to initiate data transfers on the bus at any given time is called the bus master.

Scanned with CamScanner

Bus arbitration is the process by which the next device to become the bus master is selected and bus mastership is transferred to it.

Two types

1. Centralized Arbitration
2. Distributed Arbitration.

Centralized Arbitration

A single bus arbiter performs the required arbitration.

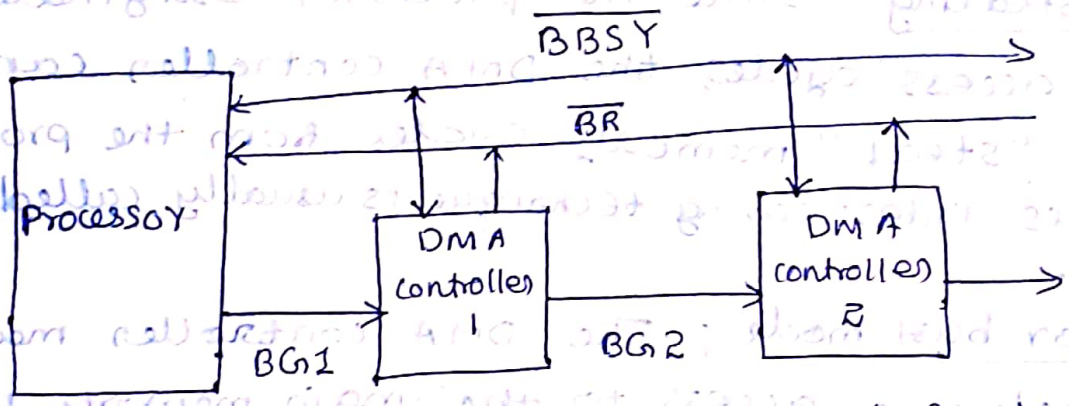


Figure 3.9 A simple arrangement for bus arbitration using a daisy chain.

1) The processor is normally the bus master unless it grants bus mastership to one of the DMA controllers.
2) A DMA controller indicates that it needs to become the bus master by activating the Bus-Request line, $\overline{BR}$.
3) When Bus-Request is activated, the processor activates the Bus-Grant signal, BG1, indicating to the DMA controllers that they may use the bus when it becomes free.

Scanned with CamScanner

4) This signal is connected to all DMA controllers using daisy-chain arrangement.

5) Thus, if DMA controller 1 is requesting the bus, it blocks the propagation of the grant signal to other devices. Otherwise, it passes the grant downstream by asserting BG2.

6). The current bus master indicates to all devices that it is using the bus by activating another line called Bus-Busy, $\overline{BBSY}$.

Distributed Arbitration

Distributed arbitration means that all devices waiting to use the bus have equal responsibility in carrying out the arbitration process, without using a central arbiter.

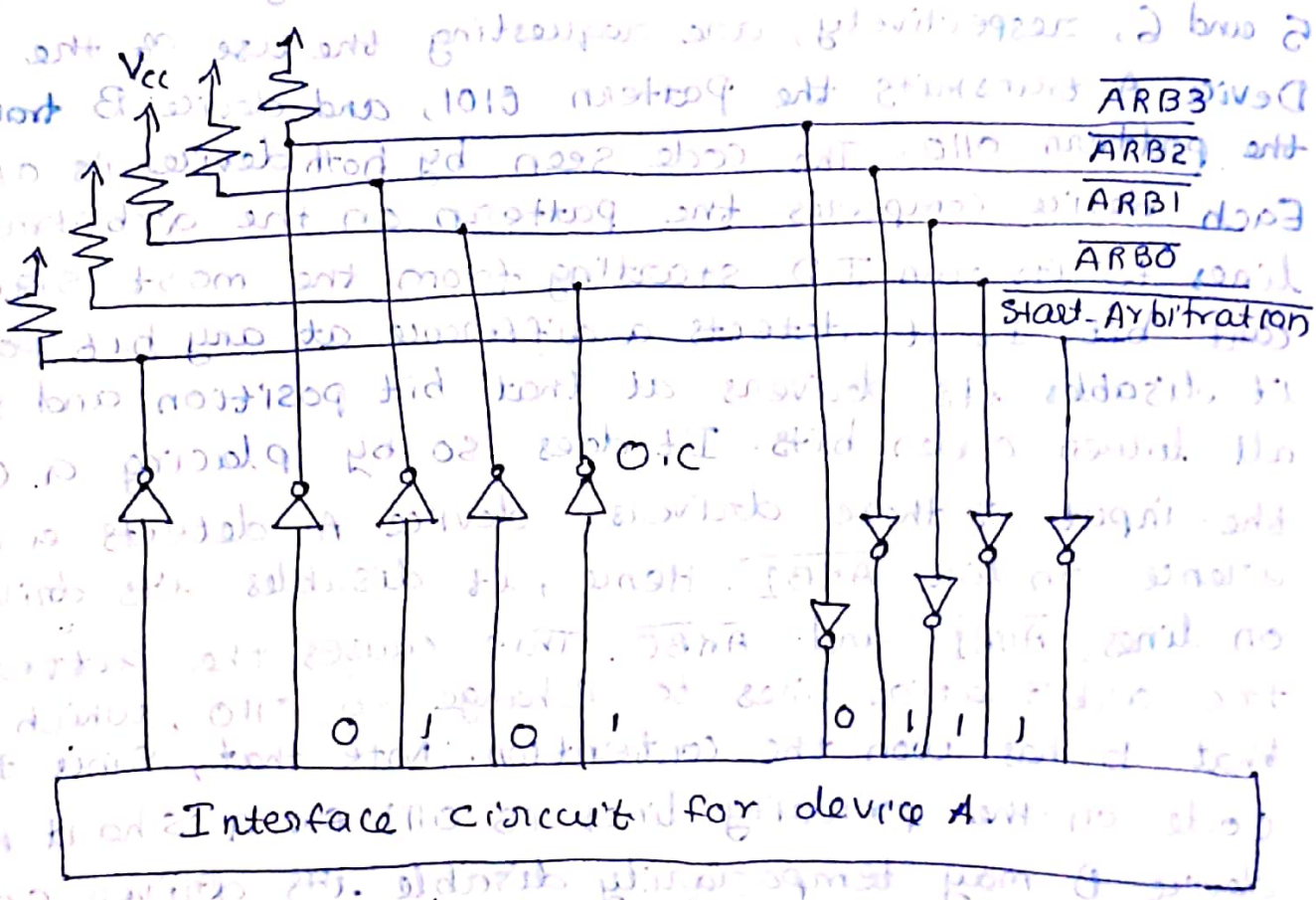


Figure 3.10 A distributed arbitration scheme.

Scanned with CamScanner

① Each device on the bus is assigned a 4-bit identification number.

② When one or more devices request the bus, they assert the Start-Arbitration signal and place their 4-bit ID numbers on four open-collector lines $\overline{ARB0}$ through $\overline{ARB3}$.

③ A winner is selected as a result of the interaction among the signals transmitted over these lines by all contenders.

④ Decentralized arbitration has the advantage of offering higher reliability, because operation of the bus is not dependent on any single device.

Assume that two devices, A and B, having ID number 5 and 6, respectively, are requesting the use of the bus. Device A transmits the pattern 0101, and device B transmits the pattern 0110. The code seen by both devices is 0111. Each device compares the pattern on the arbitration lines to its own ID, starting from the most significant bit. If it detects a difference at any bit position, it disables its drivers at that bit position and for all lower-order bits. It does so by placing a 0 at the input of these drivers. Device A detects a difference on line $\overline{ARB1}$. Hence, it disables its drivers on lines $\overline{ARB1}$ and $\overline{ARB0}$. This causes the pattern on the arbitration lines to change to 0110, which means that B has won the contention. Note that, since the code on the priority lines is 0111 for a short period, device B may temporarily disable its driver on line $\overline{ARB0}$. However, it will enable this driver again once it sees a 0 on line $\overline{ARB1}$ resulting from the action by device A.

Scanned with CamScanner

BUS

A bus requires a set of rules, often called a *bus protocol*, that govern how the bus is used by various devices. The bus protocol determines when a device may place information on the bus, when it may load the data on the bus into one of its registers, and so on. These rules are implemented by control signals that indicate what and when actions are to be taken.

One control line, usually labelled R/W, specifies whether a Read or a Write operation is to be performed. As the label suggests, it specifies Read when set to 1 and Write when set to 0. When several data sizes are possible, such as byte, half word, or word, the required size is indicated by other control lines. The bus control lines also carry timing information. They specify the times at which the processor and the I/O devices may place data on or receive data from the data lines. A variety of schemes have been devised for the timing of data transfers over a bus. These can be broadly classified as either synchronous or asynchronous schemes.

In any data transfer operation, one device plays the role of a *master*. This is the device that initiates data transfers by issuing Read or Write commands on the bus. Normally, the processor acts as the master, but other devices may also become masters.

1. SYNCHRONOUS BUS

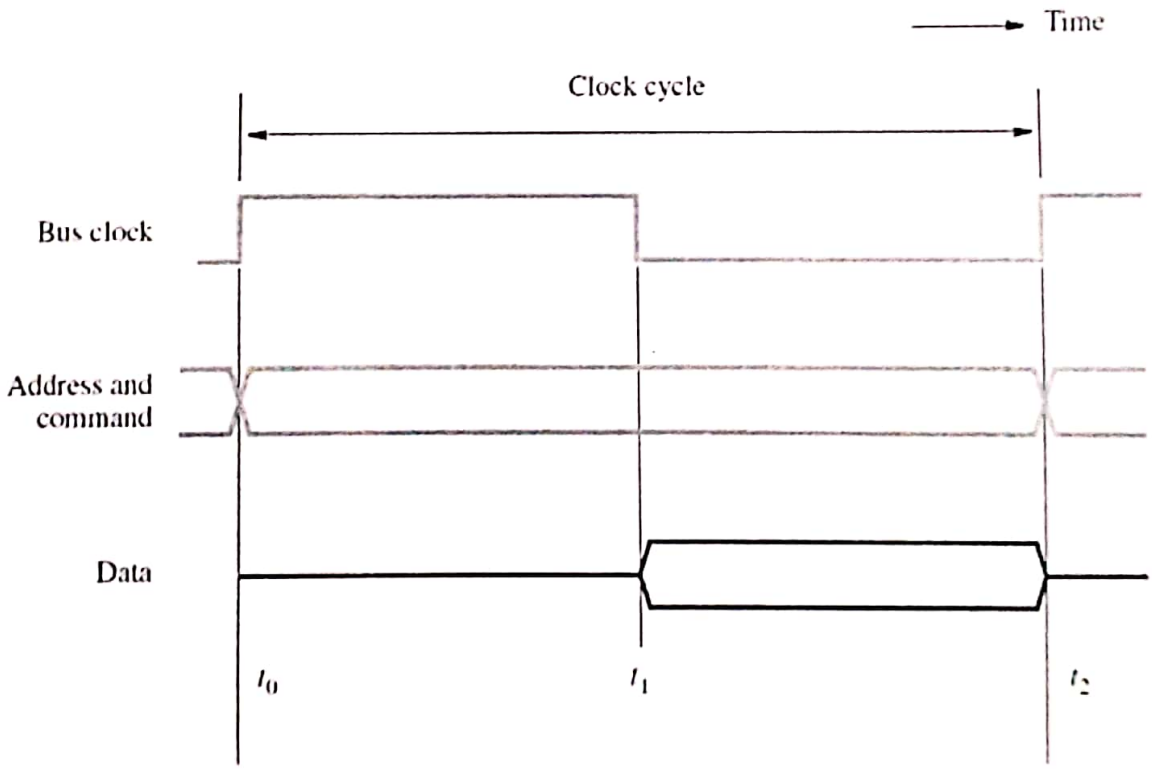


Figure 3.11 Timing of an input transfer on a synchronous bus.

Scanned with CamScanner

On a synchronous bus, all devices derive timing information from a control line called the bus clock, shown at the top of Figure 3.11. The signal on this line has two phases: a high level followed by a low level. The two phases constitute a clock cycle. The first half of the cycle between the low-to-high and high-to-low transitions is often referred to as a clock pulse.

The address and data lines in Figure 3.11 are shown as if they are carrying both high and low signal levels at the same time. This is a common convention for indicating that some lines are high and some low, depending on the particular address or data values being transmitted. The crossing points indicate the times at which these patterns change. A signal line at a level half-way between the low and high signal levels indicates periods during which the signal is unreliable, and must be ignored by all devices.

Let us consider the sequence of signal events during an input (Read) operation. At time t_0, the master places the device address on the address lines and sends a command on the control lines indicating a Read operation. The command may also specify the length of the operand to be read. Information travels over the bus at a speed determined by its physical and electrical characteristics. The clock pulse width, $t_1 - t_0$, must be longer than the maximum propagation delay over the bus. Also, it must be long enough to allow all devices to decode the address and control signals, so that the addressed device (the slave) can respond at time t_1 by placing the requested input data on the data lines. At the end of the clock cycle, at time t_2, the master loads the data on the data lines into one of its registers. To be loaded correctly into a register, data must be available for a period greater than the setup time of the register. Hence, the period $t_2 - t_1$ must be greater than the maximum propagation time on the bus plus the setup time of the master's register.

The timing diagram in Figure 3.11 is an idealized representation of the actions that take place on the bus lines. Because signals take time to travel from one device to another, a given signal transition is seen by different devices at different times. The exact times at which signals change state are somewhat different from those shown, because of propagation delays on bus wires and in the circuits of the devices. Figure 3.12 gives a more realistic picture of what actually happens.

The master sends the address and command signals on the rising edge of the clock at the beginning of the clock cycle (at t_0). However, these signals do not actually appear on the bus until t_{AM}, largely due to the delay in the electronic circuit output from the master to the bus lines. A short while later, at t_{AS}, the signals reach the slave. The slave decodes the address, and at t_1 sends the requested data. Here again, the data signals do not appear on the bus until t_{DS}. They travel toward the master and arrive at t_{DM}. At t_2, the master loads the data into its register. Hence the period $t_2 - t_{DM}$ must be greater than the setup time of that register. The data must continue to be valid after t_2 for a period equal to the hold time requirement of the register.

Limitations:
Because a transfer has to be completed within one clock cycle, the clock period, $t_2 - t_0$, must be chosen to accommodate the longest delays on the bus and the slowest device interface. This forces all devices to operate at the speed of the slowest device.

Scanned with CamScanner

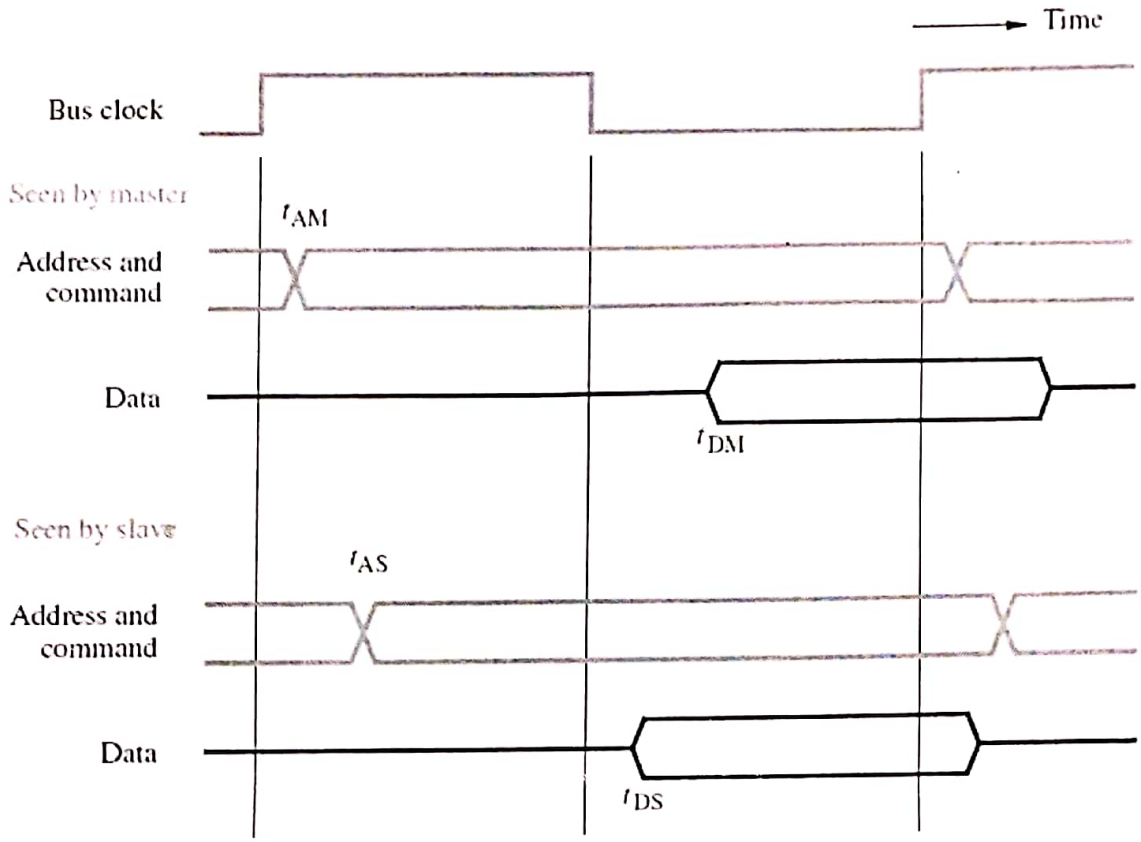


Figure 3.12 A detailed timing diagram for the input transfer of Figure 3.11

Also, the processor has no way of determining whether the addressed device has actually responded. At t_2, it simply assumes that the input data are available on the data lines in a Read operation, or that the output data have been received by the I/O device in a Write operation. If, because of a malfunction, a device does not operate correctly, the error will not be detected.

Solution:
To overcome these limitations, most buses incorporate control signals that represent a response from the device. These signals inform the master that the slave has recognized its address and that it is ready to participate in a data transfer operation. They also make it possible to adjust the duration of the data transfer period to match the response speeds of different devices. This is often accomplished by allowing a complete data transfer operation to span several clock cycles. Then, the number of clock cycles involved can vary from one device to another.

MULTIPLE-CYCLE DATA TRANSFER

In Figure 3.13 during clock cycle 1, the master sends address and command information on the bus, requesting a Read operation. The slave receives this information and decodes it. It begins to access the requested data on the active edge of the clock at the beginning of clock cycle 2. We have assumed that due to the delay involved in getting the data, the slave cannot respond

Scanned with CamScanner

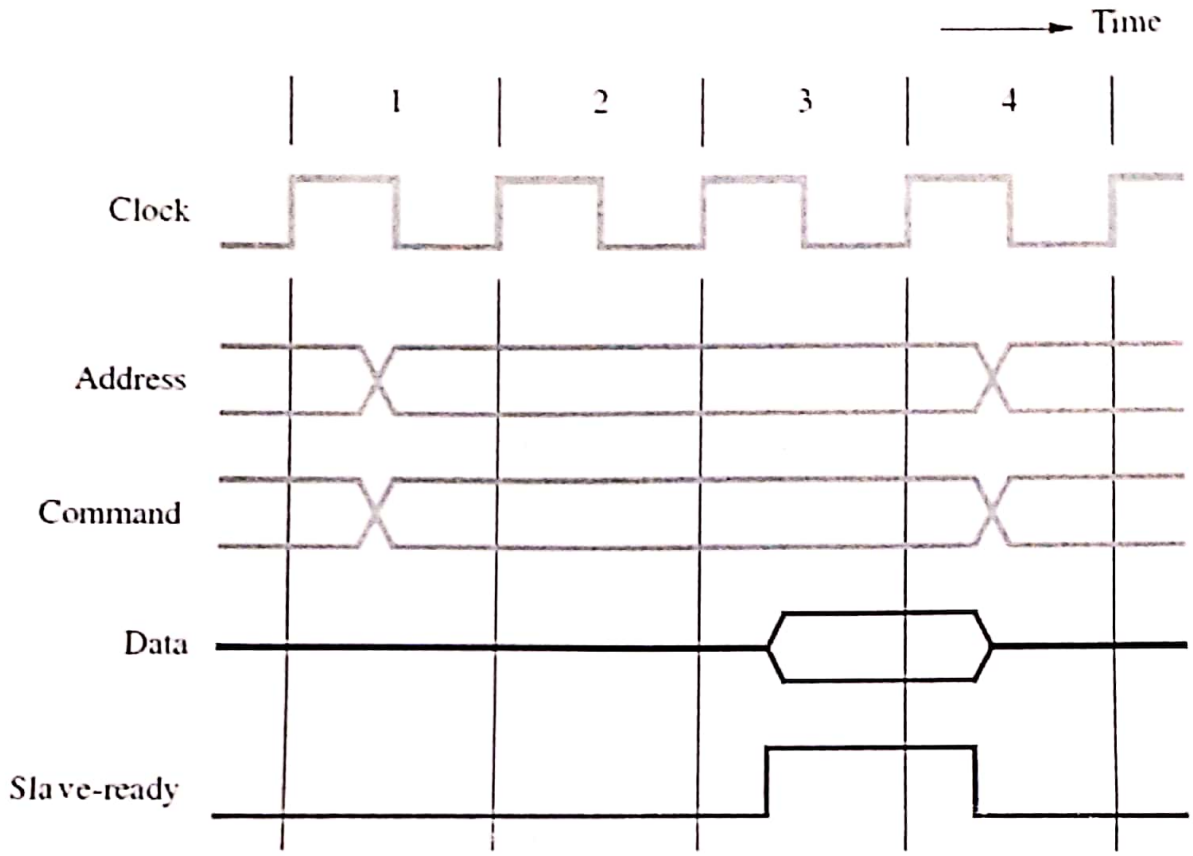


Figure 3.13 An input transfer using multiple clock cycles.

immediately. The data become ready and are placed on the bus during clock cycle 3. The slave asserts a control signal called Slave-ready at the same time. The master, which has been waiting for this signal, loads the data into its register at the end of the clock cycle. The slave removes its data signals from the bus and returns its Slave-ready signal to the low level at the end of cycle 3. The bus transfer operation is now complete, and the master may send new address and command signals to start a new transfer in clock cycle 4.

2. ASYNCHRONOUS BUS

An alternative scheme for controlling data transfers on a bus is based on the use of a *handshake* protocol between the master and the slave. A handshake is an exchange of command and response signals between the master and the slave. A control line called Master-ready is asserted by the master to indicate that it is ready to start a data transfer. The Slave responds by asserting Slave-ready.

Example:

Figure 3.14 depicts the following sequence of events:

t_0— The master places the address and command information on the bus, and all devices on the bus decode this information.

Scanned with CamScanner

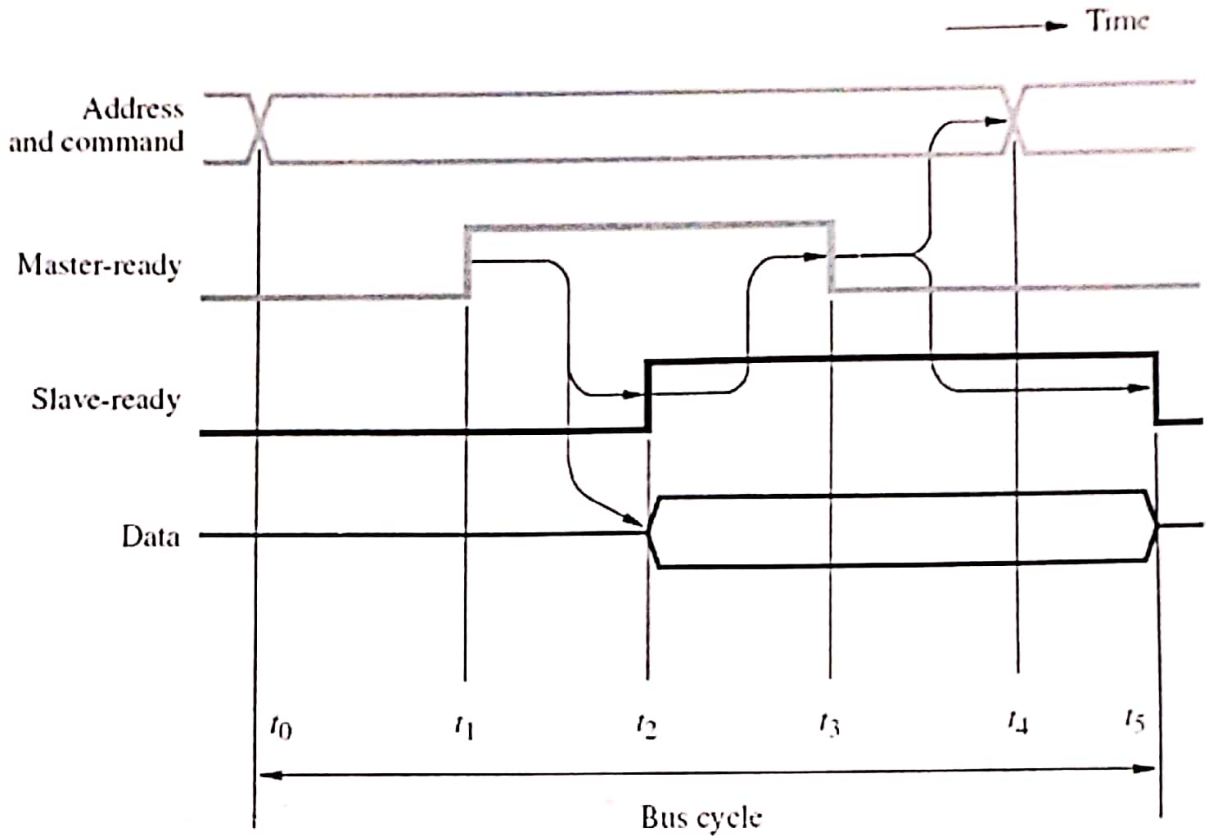


Figure 3.14 Handshake control of data transfer during an input operation.

t_1— The master sets the Master-ready line to 1 to inform the devices that the address and command information is ready. The delay $t_1 - t_0$ is intended to allow for any *skew* that may occur on the bus. Skew occurs when two signals transmitted simultaneously from one source arrive at the destination at different times. This happens because different lines of the bus may have different propagation speeds. Thus, to guarantee that the Master-ready signal does not arrive at any device ahead of the address and command information, the delay $t_1 - t_0$ should be longer than the maximum possible bus skew.

t_2— The selected slave, having decoded the address and command information, performs the required input operation by placing its data on the data lines. At the same time, it sets the Slave-ready signal to 1. If extra delays are introduced by the interface circuitry before it places the data on the bus, the slave must delay the Slave-ready signal accordingly. The period $t_2 - t_1$ depends on the distance between the master and the slave and on the delays introduced by the slave's circuitry.

t_3— The Slave-ready signal arrives at the master, indicating that the input data are available on the bus. The master must allow for bus skew. It must also allow for the setup time needed by its register. After a delay equivalent to the maximum bus skew and the minimum setup time, the master loads the data into its register. Then, it drops the Master-ready signal, indicating that it has received the data.

Scanned with CamScanner

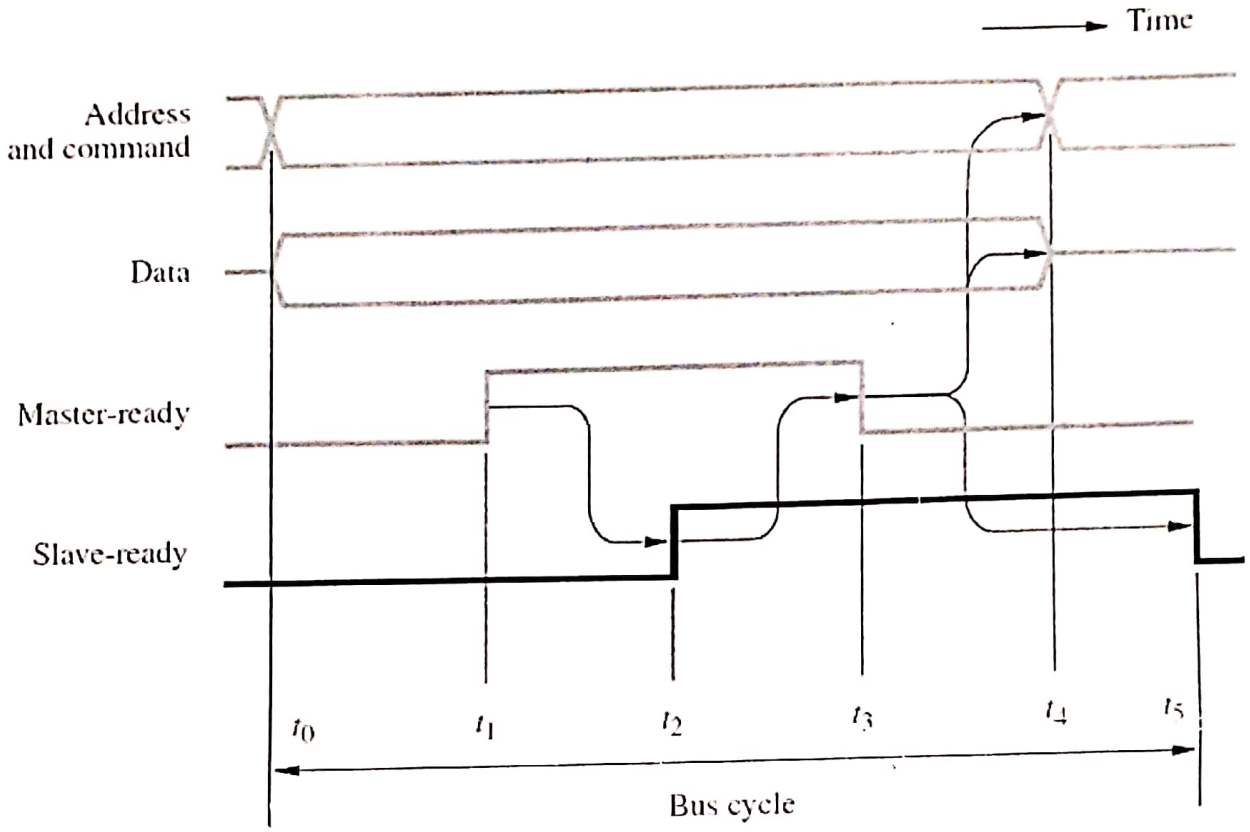


Figure 3.15 Handshake control of data transfer during an output operation.

t_4— The master removes the address and command information from the bus. The delay between t_3 and t_4 is again intended to allow for bus skew. Erroneous addressing may take place if the address, as seen by some device on the bus, starts to change while the Master-ready signal is still equal to 1.

t_5— When the device interface receives the 1-to-0 transition of the Master-ready signal, it removes the data and the Slave-ready signal from the bus. This completes the input transfer.

The handshake signals in Figures 3.14 and 3.15 are said to be fully interlocked, because a change in one signal is always in response to a change in the other. Hence, this scheme is known as a full handshake. It provides the highest degree of flexibility and reliability.

Scanned with CamScanner

INTERFACE CIRCUITS

i. An I/O interface consists of the circuitry for
   1. On one side the bus signals for address, data, and control to a computer bus.
   2. On the other side a data path with its associated control signals to transfer data between the interface and the I/O device. This side is called a port.

ii. Port can be classified as
   1. parallel port (8 or 16 bits at a time) - multiple pin
   2. serial Port (one bit at a time) - long cable

iii. Communication with the bus is same for both ports.

iv. An I/O interface does the following:
   1. Provides a storage buffer for at least one word of data (or one byte, in the case of byte-oriented devices)
   2. contains status flags that can be accessed by the processor to determine whether the buffer is full (for input) or empty (for output).
   3. contains address-decoding circuitry to determine when it is being addressed by the processor.
   4. Generates the appropriate timing signals required by the bus control scheme.
   5. performs any format conversion that may be necessary to transfer data between the bus and the I/O device, such as parallel-serial conversion in the case of a serial port.

Scanned with CamScanner

PARALLEL PORT

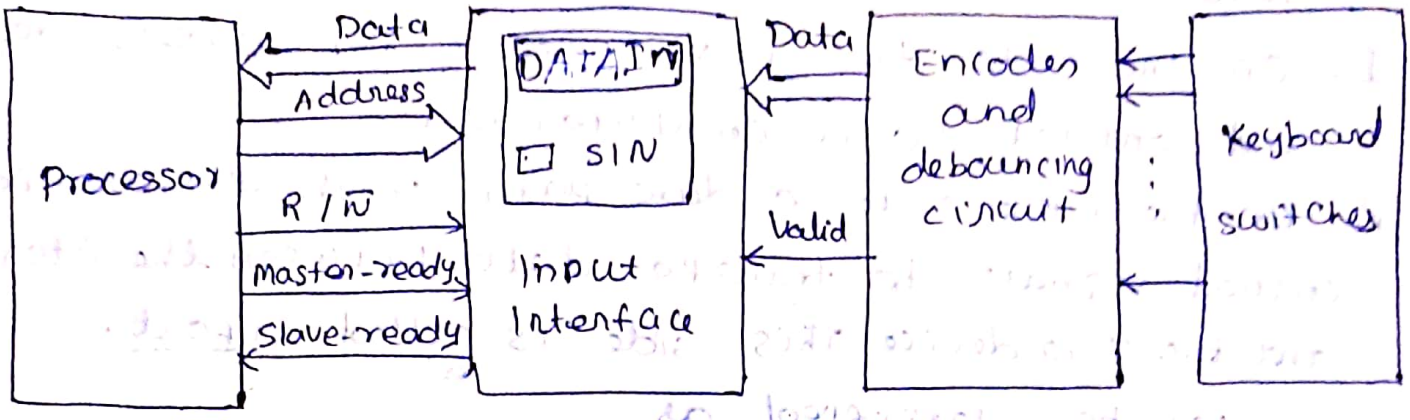


Figure 3.16 Keyboard to processor connection.

Assume that
1. Interface circuit is connected to a 32-bit processor
2. Processor uses memory-mapped I/O
3. Uses the asynchronous bus protocol.

Figure 3.16 shows the hardware components needed for connecting a keyboard to a processor.
1. Keyboard – consists of mechanical switches.
2. Encoder – Generate ASCII for the signal received.
3. Debouncing – to eliminate the effect of bouncing.
4. Interface circuit
   - Data register – DATAIN – data from the encoder
   - Status flag – SIN – Valid signal – key pressed = 1

Figure 3.17 shows a suitable circuit for an input interface.
1. The output lines of the DATAIN register are connected to the data lines of the bus by means of three-state drivers, which are turned on when the processor issues a read instruction with the address that selects this register.

Scanned with CamScanner

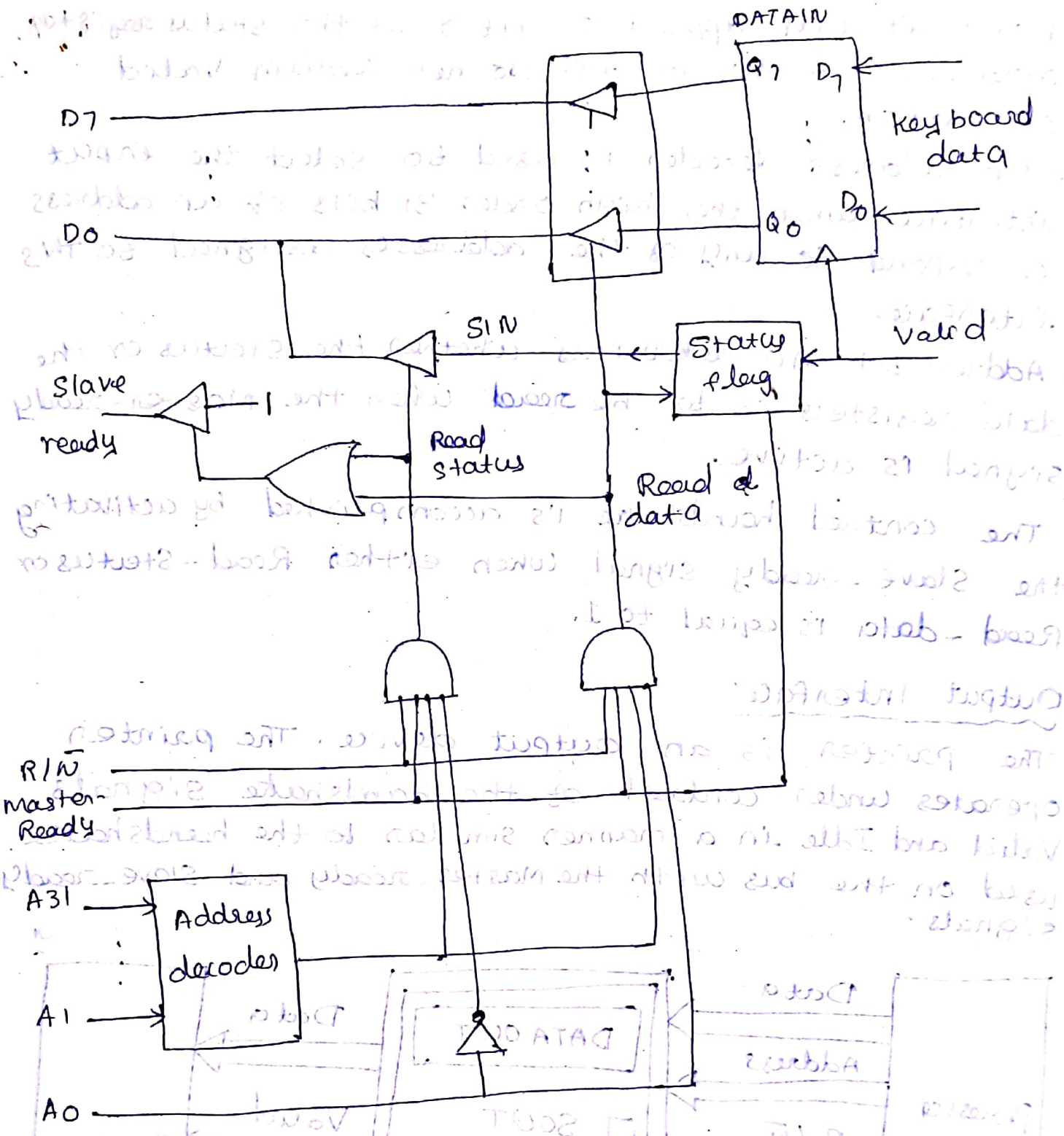


Figure 3.17 Input interface circuit.

2. The SIN signal is generated by a status flag circuit. This signal is also sent to the bus through a three-state driver. It is connected to bit D0, which

Scanned with CamScanner

means it will appear as bit 0 of the status register. Other bits of this register do not contain valid information.

3. An address decoder is used to select the input interface when the high-order 31 bits of an address correspond to any of the addresses assigned to this interface.
4. Address bit A0 determines whether the status or the data registers is to be read when the Master-ready signal is active.
5. The control handshake is accomplished by activating the Slave_ready signal when either Read-Status or Read_data is equal to 1.

Output Interface:

The printer is an output device. The printer operates under control of the handshake signals Valid and Idle in a manner similar to the handshake used on the bus with the Master_ready and Slave_ready signals.

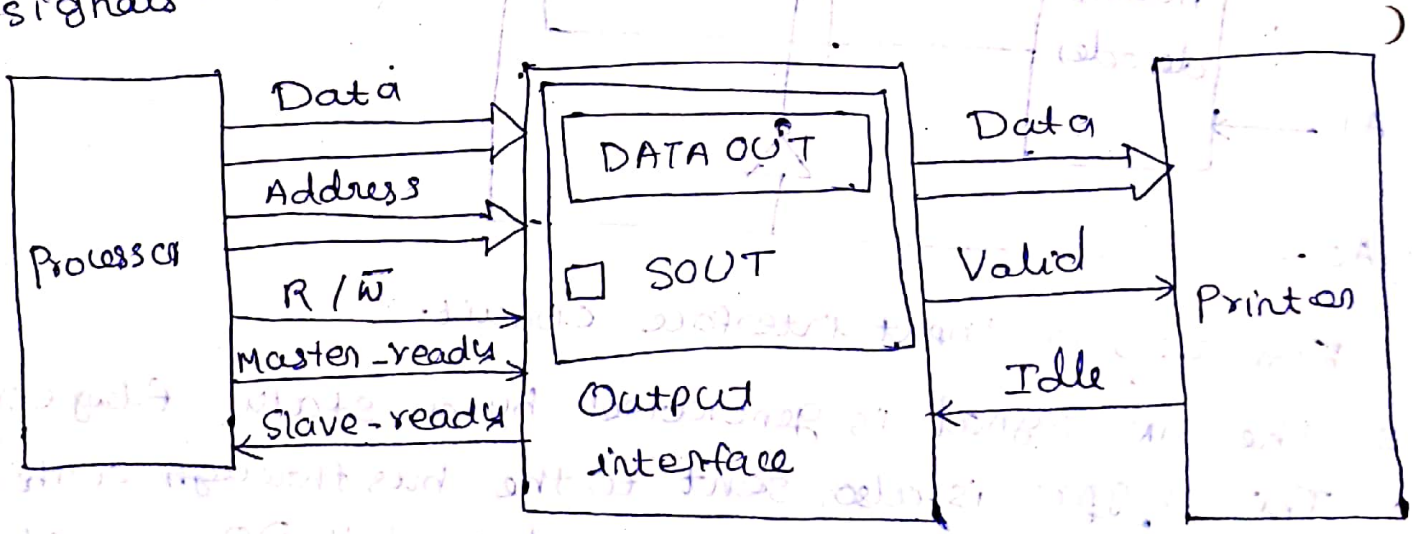


Figure 3.18 printer to processor connection.

Scanned with CamScanner

i. When the printer is ready to accept a character, the printer asserts its Idle signal.

ii. The interface circuit can then place a new character on the data lines and activate the valid signal.

iii. In response, the printer starts printing the new character negates the Idle signal, which in turn causes the interface to deactivate the valid signal.

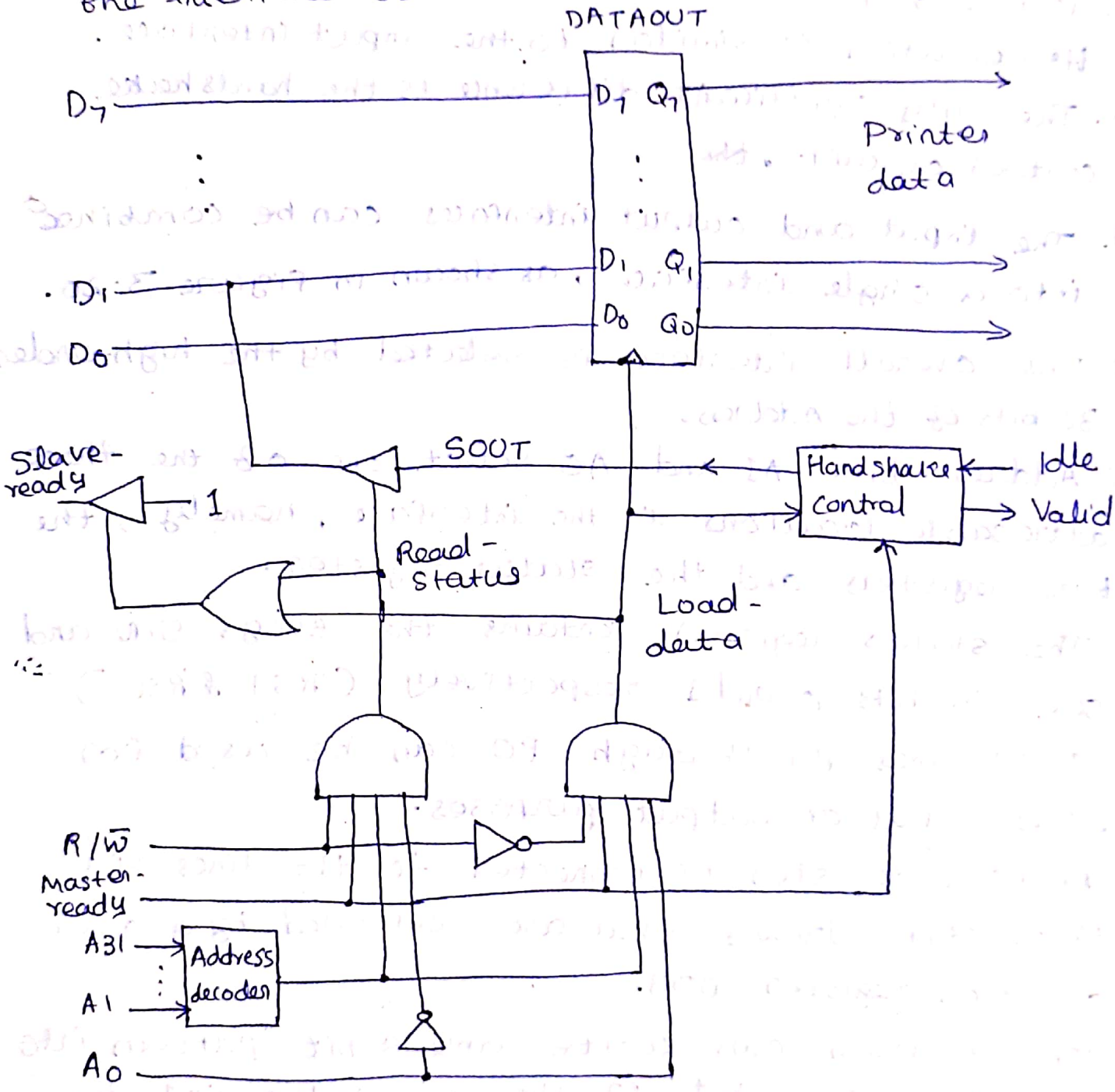


Figure 3.19 Output interface circuit.

Scanned with CamScanner

iv. The interface contains a data register, DATAOUT, and a status flag, SOUT.

v. The SOUT flag is set to 1 when the printer is ready to accept another character, and it is cleared to 0 when a new character is loaded into DATAOUT by the processor.

vi. Figure 3.19 shows an implementation of the interface. Its operation is similar to the input interface.

vii. The only significant difference is the handshake control circuit.

viii. The input and output interfaces can be combined into a single interface, as shown in Figure 3.20.

a) The overall interface is selected by the high-order 30 bits of the Address.

b) Address bits A1 and A0 select one of the three addressable locations in the interface, namely, the two registers and the status register.

c) The status register contains the flags SIN and SOUT in bits 0 and 1, respectively. (RS1 & RS0)

d) Data lines P7 through P0 can be used for either input or output purposes.

e) DATAOUT register is connected to the lines via three-state drivers that are controlled by a data direction register, DDR.

f) The processor can write any 8-bit pattern into DDR. For a given bit, if the DDR value is 1, the

Scanned with CamScanner

: Corresponding data line acts as an output line; Otherwise, it acts as an input line.

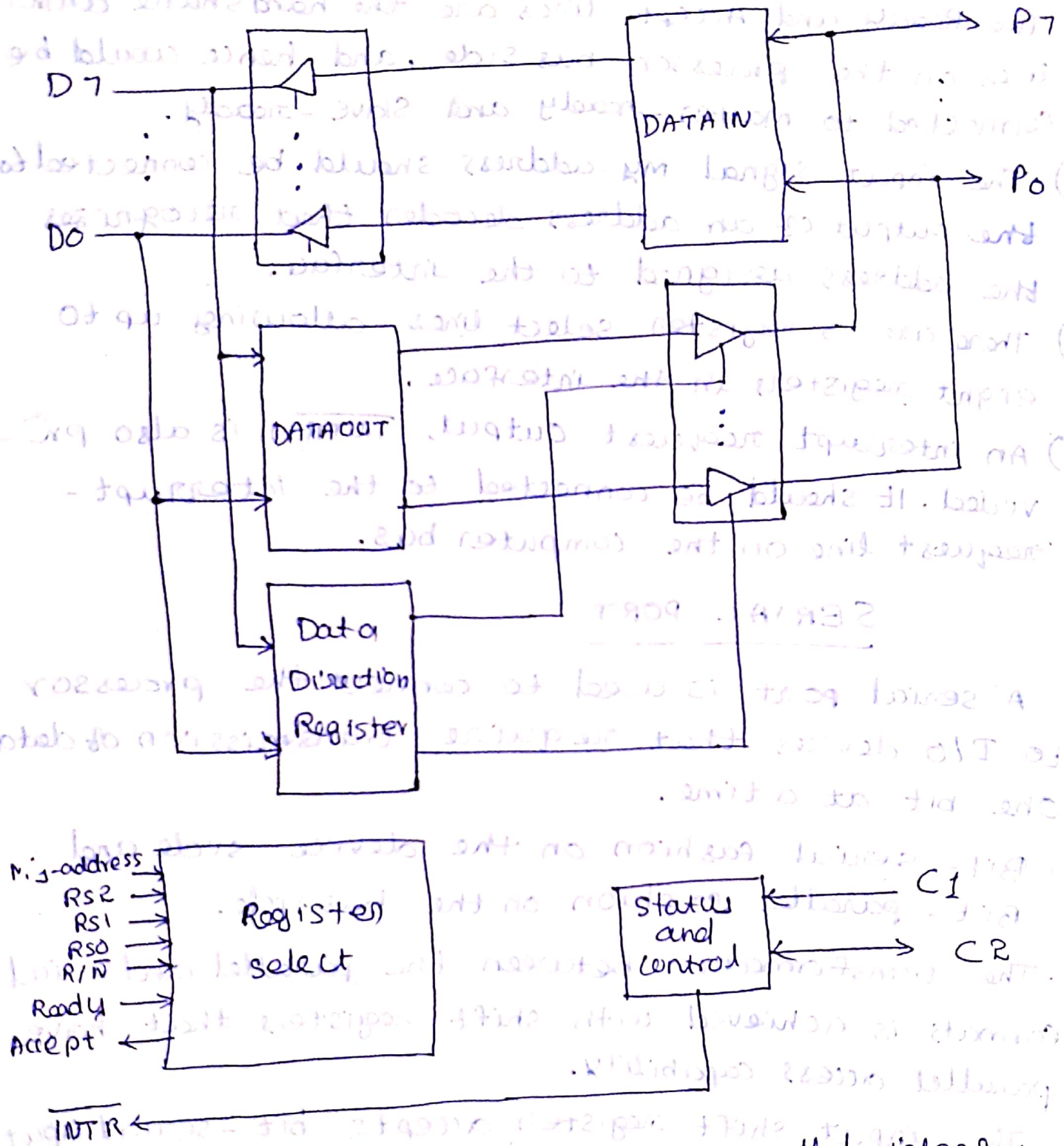


Figure 3.20- A general 8-bit parallel interface.

Scanned with CamScanner

g) Two lines, C1 and C2 - control signals, C2 - hand shake.

h) The Ready and Accept lines are the hand shake control lines on the processor bus side, and hence would be connected to master-ready and slave-ready.

i) The input signal My-address should be connected to the output of an address decoder that recognizes the address assigned to the interface.

j) There are 3 register select lines, allowing up to eight registers in the interface.

k) An interrupt request output, $\overline{INTR}$, is also provided. It should be connected to the interrupt-request line on the computer bus.

SERIAL PORT

1. A serial port is used to connect the processor to I/O devices that require transmission of data one bit at a time.
2. Bit-serial fashion on the device side and Bit-parallel fashion on the bus side.
3. The transformation between the parallel and serial formats is achieved with shift registers that have parallel access capability.
4. The input shift register accepts bit-serial input from the I/O device. When all 8 bits of data have been received, the contents of this shift register are loaded in parallel into the DATAIN register.

Scanned with CamScanner

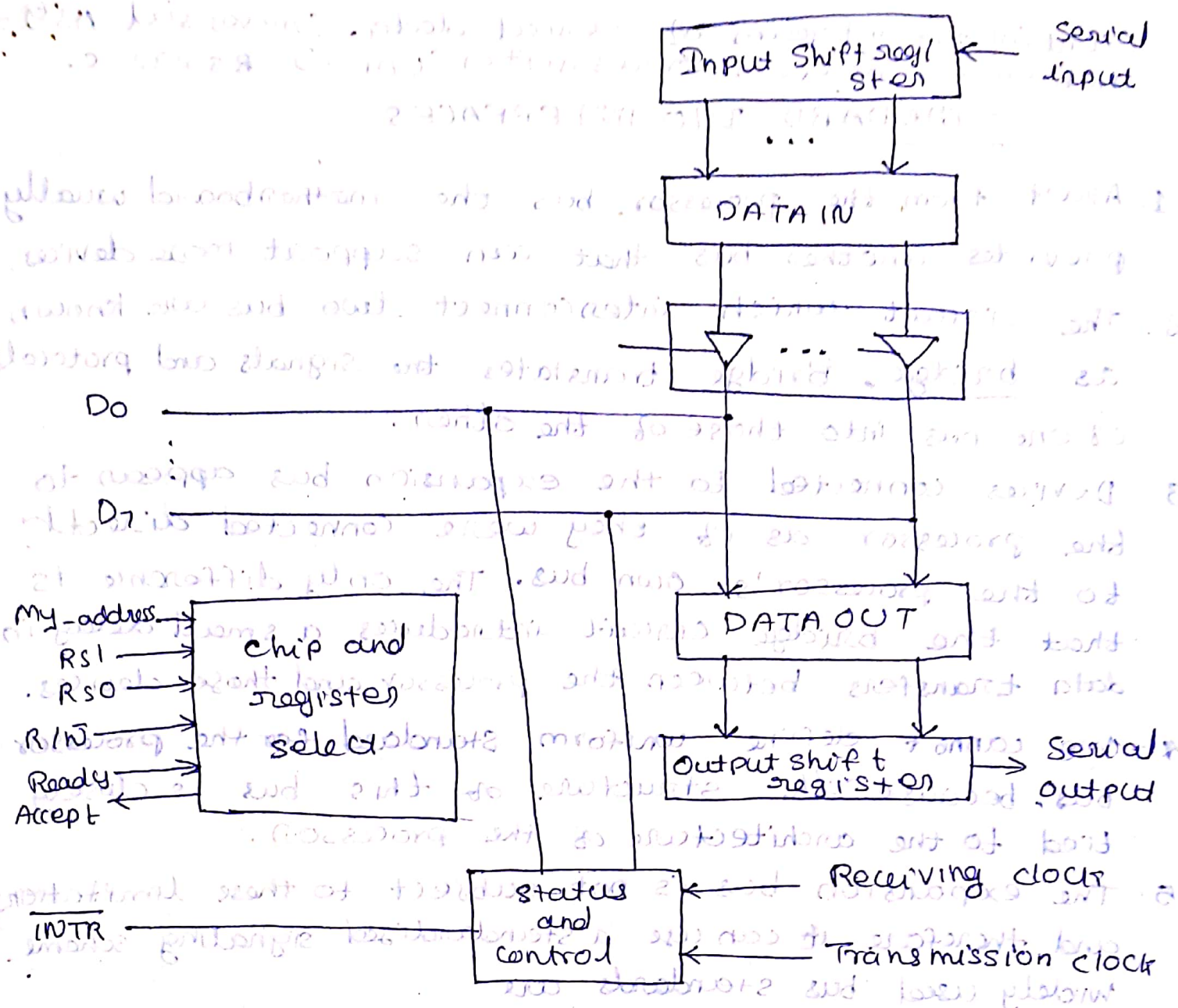


Figure 3.21 A serial interface.

5. With the double buffer, the transfer of the second character can begin as soon as the first character is loaded from the shift register into the DATAIN register.
6. Thus, provided the processor reads the contents of DATAIN before the serial transfer of the second character is completed, the interface can receive a

Scanned with CamScanner

continuous stream of serial data. Universal Asynchronous Receiver Transmitter (UART), RS-232-C.

STANDARD I/O INTERFACES

1. Apart from the processor bus the motherboard usually provides another bus that can support more devices.
2. The circuit which interconnect two bus are known as bridge. Bridge translates the signals and protocols of one bus into those of the other.
3. Devices connected to the expansion bus appear to the processor as if they were connected directly to the processor's own bus. The only difference is that the bridge circuit introduces a small delay in data transfers between the processor and those devices.
4. We cannot define uniform standard for the processor bus, because the structure of this bus is closely tied to the architecture of the processor.
5. The expansion bus is not subject to these limitations and therefore it can use a standardized signaling scheme.

Widely used bus standards are

1. PCI (peripheral Component Interconnect)
2. SCSI (Small Computer System Interface) and
3. USB (Universal Serial Bus).

The way these standards are used in a typical computer system is illustrated in Figure 3.22. The PCI standard defines an expansion bus on the motherboard. SCSI and USB are used for connecting additional devices, both inside and outside the computer box.

Scanned with CamScanner

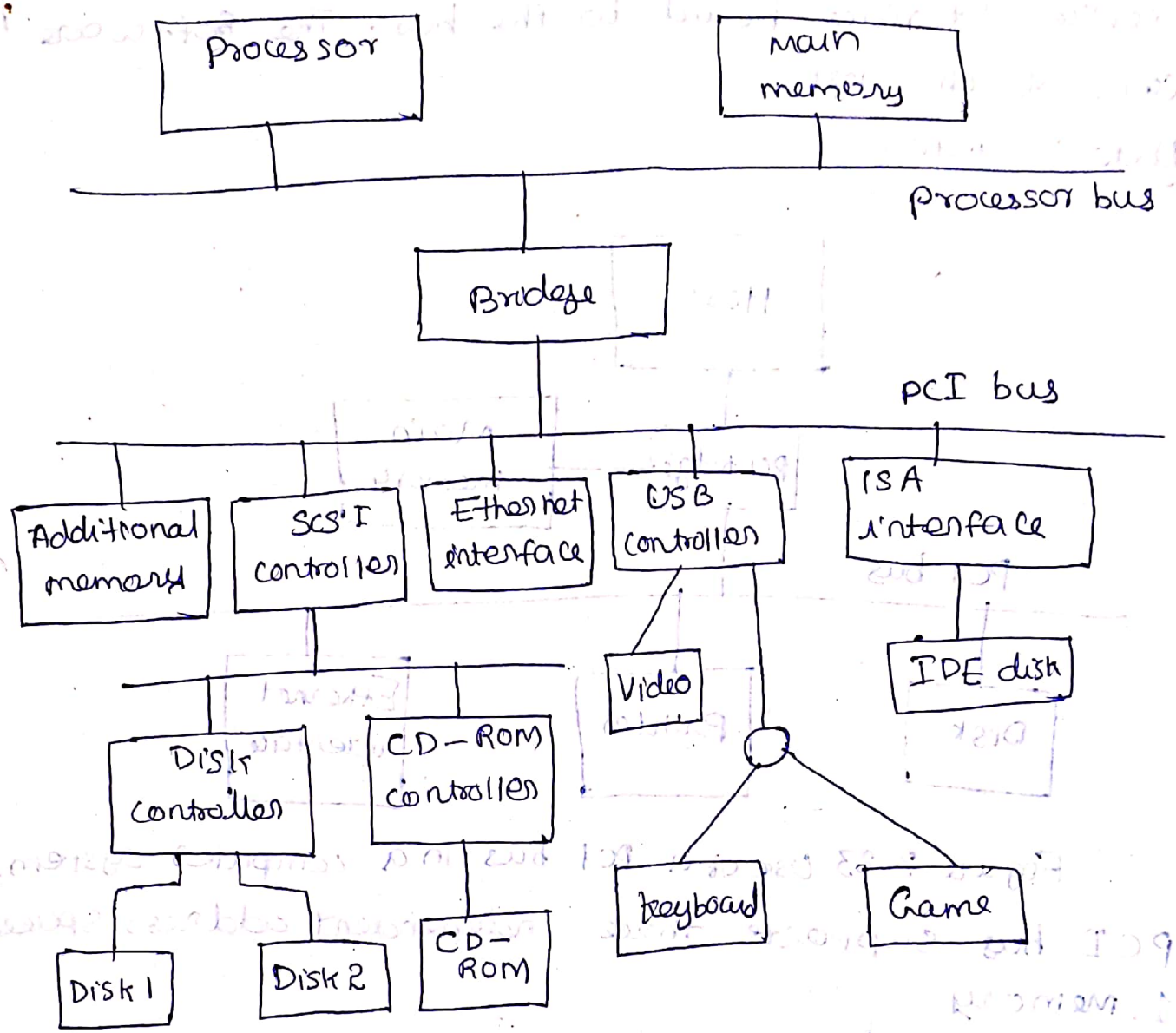


Figure 3.22 An example of a computer system using different interface standards.

IBM developed a bus they called ISA (Industry Standard Architecture).

1. PERIPHERAL COMPONENT INTERCONNECT (PCI) BUS

i. The PCI follows a sequence of bus standards that were used primarily in IBM PCs.

ii. An important feature is a plug-and-play. To connect a new device, the user simply connects the

Scanned with CamScanner

device interface board to the bus. The Software takes care of the rest.

Data Transfer

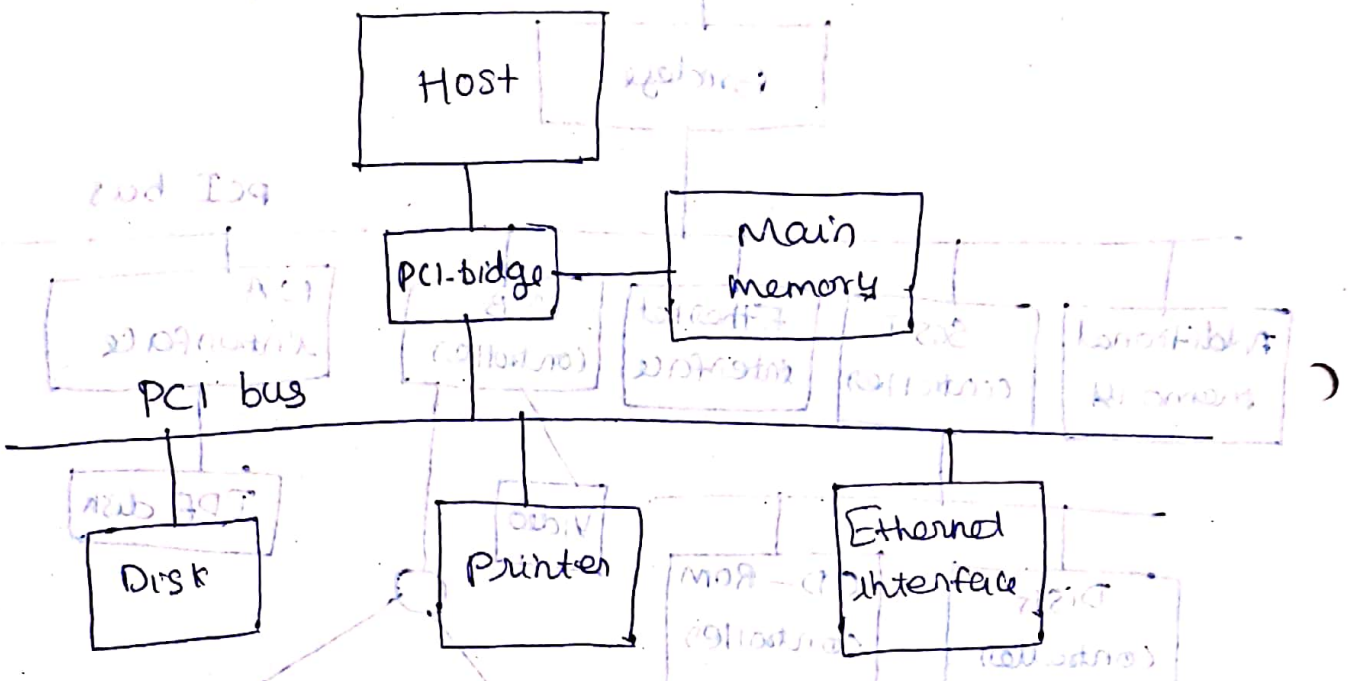


Figure 3-23 Use of a PCI bus in a computer system.

1. PCI bus supports three independent address spaces:
   1. Memory
   2. I/O and
   3. configuration.
2. The configuration space is intended to give the PCI its plug-and-play capability.
3. A 4-bit command that accompanies the address identifies which of the three spaces it is being used in a given data transfer operation.
4. The PCI bridge provides a seperate physical connection for the main memory.

5/12/19

Scanned with CamScanner

The main bus signals used for transferring data are:

Table 3.1 Data transfer signals on the PCI bus

Name	Function
CLK	A 33-MHz or 66 MHz clock
FRAME#	Sent by the initiator to indicate the duration of a transaction.
AD	32 address / data lines, which may be optionally increased to 64.
C/BE#	4 command / byte-enable lines (8 for a 64-bit bus).
IRDY#, TRDY#	Initiator-ready and Target-ready signals.
DEVSEL#	A response from the device indicating that it has recognised its address and is ready for a data transfer transaction.
IDSEL#	Initialization Device Select.

The approach used in the PCI bus:
The address in the address bus is needed only long enough for the slave to be selected. The slave can store the address in its internal buffer. Thus, the address is needed on the bus for one clock cycle only, freeing the address lines to be used for sending data in subsequent clock cycles.

6. Initiator:
A master is called an initiator in PCI terminology

Scanned with CamScanner

7. target:

The addressed device that responds to read and write commands is called a target.

8. Signals whose name ends with the symbol # are asserted when in the low-voltage state.

9. Transaction:

A complete transfer operation on the bus, involving an address and a burst of data, is called a transaction.

10. phases:

Individual word transfers within a transaction are called phases.

Example:

consider a bus transaction in which the processor reads four 32-bit words from the memory.

initiator - processor target - memory.

The sequence of events on the bus is shown in Figure 3-24;

1. A clock signal provides the timing reference used to coordinate different phases of a transaction.

clock cycle 1:

a) processor asserts FRAME# to indicate the beginning of a transaction.

b) sends the address on the AD lines and a command on the C/BE# lines.

c) In this example, the command will indicate that a read operation is requested and that the memory address space is being used.

Scanned with CamScanner

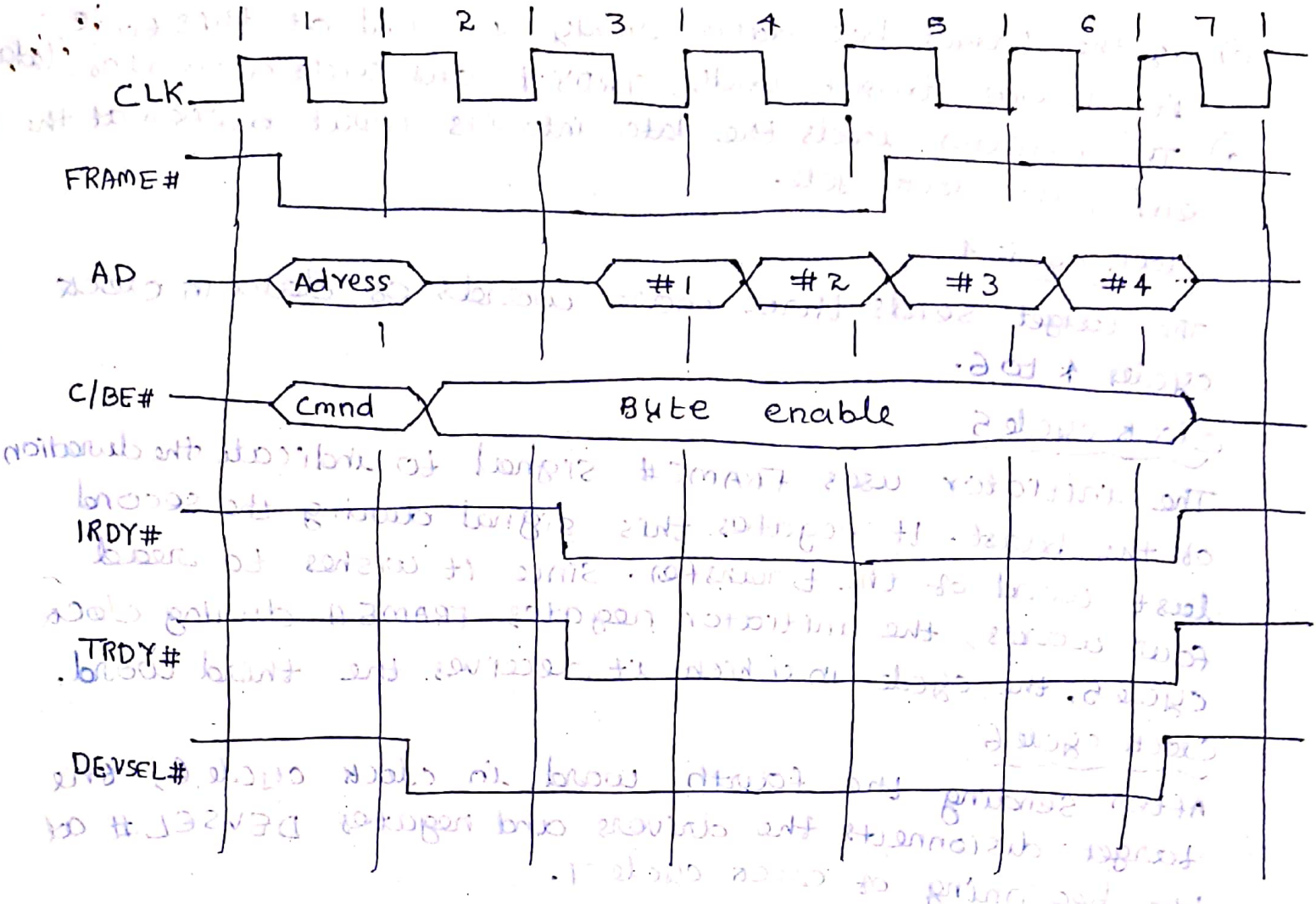


Figure 3.24 A read operation on the PCI bus.

Clock cycle 2.

a) The processor removes the address and disconnects its drivers from the AD lines.

b) The C/BE# lines, which were used to send a bus command in clock cycle 1, are used for a different purpose during the rest of the transaction.

c) Asserts DEVSEL# and maintains it in the asserted state until the end of the transaction.

Clock cycle 3

a) The selected target enables its drivers on the AD lines, and fetches the requested data to be placed on the bus.

b) The initiator asserts the initiator ready signal, IRDY#, to indicate that it is ready to receive data.

Scanned with CamScanner

c) If the target has data ready to send at this time it asserts target ready, TRDY#, and sends a word of data.

d) The initiator loads the data into its input buffer at the end of the clock cycle.

Clock cycle 4:

The target sends three more words of data in clock cycles 4 to 6.

Clock cycle 5

The initiator uses FRAME# signal to indicate the duration of the burst. It negates this signal during the second last word of the transfer. Since it wishes to read four words, the initiator negates FRAME# during clock cycle 5, the cycle in which it receives the third word.

Clock cycle 6

After sending the fourth word in clock cycle 6, the target disconnects the drivers and negates DEVSEL# at the beginning of clock cycle 7.

Device Configuration.

1. When an I/O device is connected to a computer, several actions are needed to configure both the device and the software that communicates with it.
2. The PCI simplifies the configuration by incorporating in each I/O device interface a small configuration ROM memory that stores information about that device.
3. The configuration ROMs of all devices are accessible in the configuration address space.
4. The PCI initialization software reads these ROMs whenever the system is powered up or reset.

Scanned with CamScanner

5. Devices are assigned addresses during the initialization process.
6. Each device has an input signal called Initialization Device Select, IDSEL#. During the bus configuration operation, it is this signal, rather than the address applied to the AD inputs of the device, that causes the device to be selected.
7. The motherboard in which device connectors are plugged typically has the IDSEL# pin of each device connected to one of the upper 21 address lines AD11 to AD31.
8. A device can be selected for a configuration operation by issuing a configuration command and an address in which the corresponding AD line is set to 1 and the remaining 20 lines set to 0.
9. The lower address lines, AD10 to AD00, are used to specify the type of operation and to access the contents of the device configuration ROM.
10. This arrangement limits the number of I/O devices to 21.

The configuration software:

1. The configuration software scans all 21 locations in the configuration address space to identify which devices are present.
2. Each device request an address in the I/O space or in the memory space.
3. The device is then assigned an address by writing that address into the appropriate device register.
4. The configuration software also sets such parameters as the interrupt priority. PCI has 4 interrupt request lines.

Scanned with CamScanner

Electrical Characteristics.

1. 5 or 3.3V power supply.

SCSI Bus

Small Computer System Interface (SCSI) defined by American National Standard Institute (ANSI)

A controller connected to a SCSI bus is one of two types - an initater or a target. An initiator has the ability to select a particular target and to send commands specifying the operations to be performed.

Disk read operation:

The processor sends a command to SCSI controller, which causes the following sequence of events to take place.

1. The SCSI controller, acting as an initiator, contends for control of the bus.
2. When the initiator wins the arbitration process, it selects the target controller and hands over control of the bus to it.
3. The target starts an output operation (from initiator to target); in response to this, the initiator sends a command specifying the read operation.
4. The target, realizing that it first needs to perform a disk seek operation, sends a message to the initiator indicating that it will temporarily suspend the connection between them. Then it releases the bus.
5. The target controller sends a command to the disk

Scanned with CamScanner

drive to move the read head to the first sector involved in the requested read operation. Then, it reads the data stored in that sector and stores them in a data buffer. When it is ready to begin transferring data to the initiator, the target requests control of the bus. After it wins arbitration, it reselects the initiator controller, thus restoring the suspended connection.

6. The target transfers the contents of the data transfer buffer to the initiator and then suspends the connection again. Data are transferred either 8, or 16 bits parallel, depending on the width of the bus.
7. The target controller sends a command to the disk drive to perform another seek operation. Then, it transfers the contents of the second disk sector to the initiator, as before. At the end of this transfer, the logical connection between the two controllers is terminated.
8. As the initiator controller receives the data, it stores them into the main memory using the DMA approach.
9. The SCSI controller sends an interrupt to the processor to inform it that the requested operation has been completed.

Bus signals

i. All signal names are preceded by a minus sign. This indicates that the signals are active, or that a data line is equal to 1, when they are in the low-voltage state.

ii. The bus has no address lines. Instead, the data lines are used to identify the bus controller involved

Scanned with CamScanner

during the selection or reselection process and during bus arbitration.

Table 3.2 The SCSI bus signals

Category	Name	Function
Data	-DB(0) to -DB(7)	Data lines: Carry one byte of information during the information transfer phase and identify device during arbitration, selection and reselection phases.
	-DB(P)	Parity bit for the data bus
Phase	-BSY	Busy: Asserted when the bus is not free.
	-SEL	Selection: Asserted during selection and reselection
Information type	-C/D	Control/Data: Asserted during transfer of control information (command, status or message)
	-MSG	Message: Indicates that the information being transferred is a message
Handshake	-REQ	Request: Asserted by a target to request a data transfer cycle
	-ACK	Acknowledge: Asserted by the initiator when it has completed a data transfer operation
Direction of transfer	-I/O	Input/output: Asserted to indicate an input operation (relative to the initiator)
Other	-ATN	Attention: Asserted by an initiator when it wishes to send a message to a target
	-RST	Reset: Causes all device controls to disconnect from the bus and assume their start-up state

Scanned with CamScanner

Phases of operation

1. Arbitration
2. Selection
3. Information Transfer
4. Reselection.

Arbitration

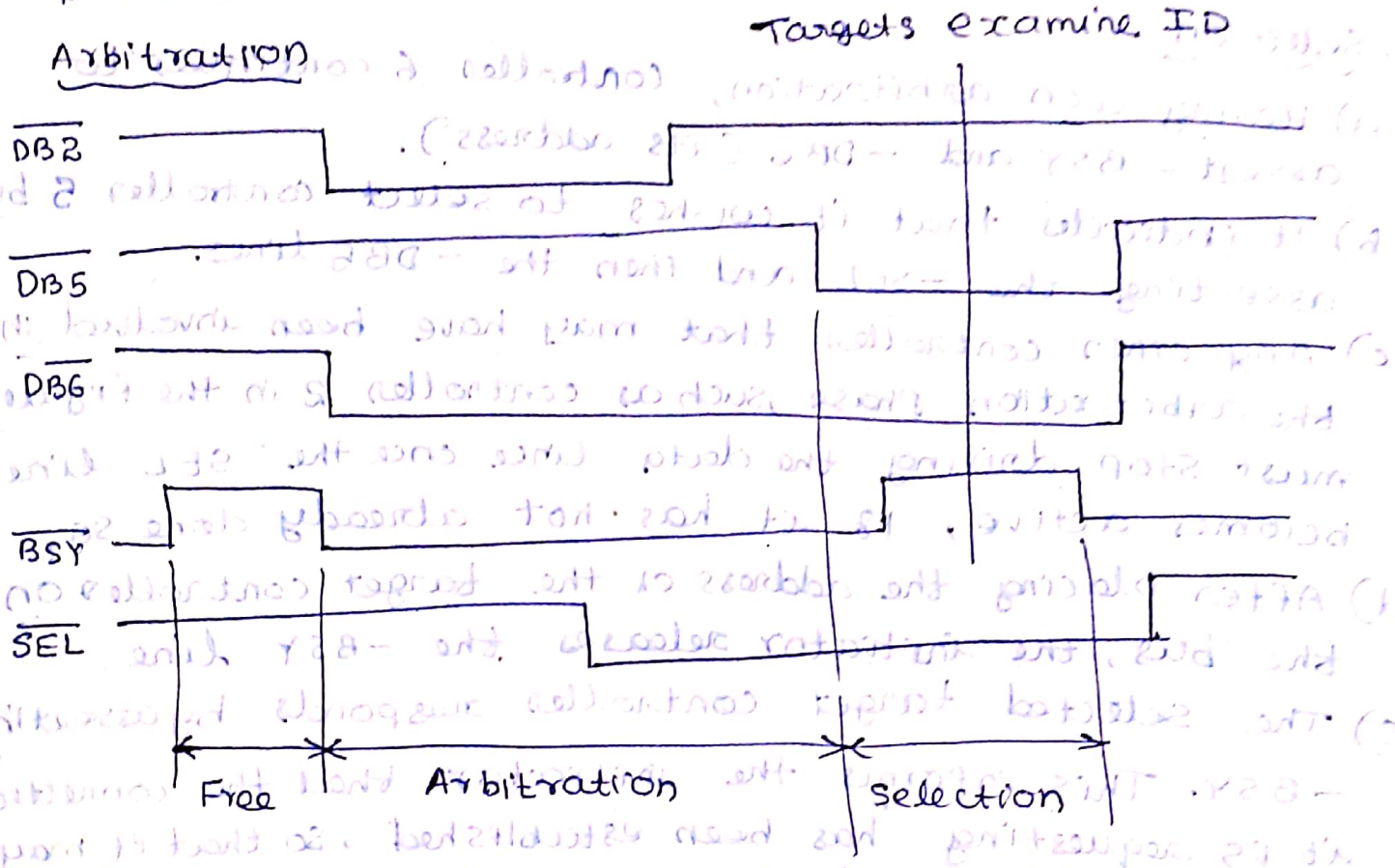


Figure 3.25 Arbitration and selection on the SCSI bus. Device 6 wins arbitration and selects device 2.

a) The bus is free when the -BSY signal is in the inactive (high voltage) state.
b) Any controller can request the use of the bus while it is in this state.
c) Since two or more controllers may generate such a request at the same time, an arbitration scheme must be implemented.
d) The SCSI bus uses a simple distributed arbitration

Scanned with CamScanner

scheme.

In Figure 3.25 controllers 2 and 6 request the use of the bus simultaneously. After winning arbitration, controller 6 proceeds to the selection phase, in which it identifies the target.

2. Selection

a) Having won arbitration, controller 6 continues to assert −BSY and −DB6 (its address).

b) It indicates that it wishes to select controller 5 by asserting the −SEL and then the −DB5 lines.

c) Any other controller that may have been involved in the arbitration phase, such as controller 2 in the figure, must stop driving the data lines once the −SEL line becomes active, if it has not already done so.

d) After placing the address of the target controller on the bus, the initiator releases the −BSY line.

e) The selected target controller responds by asserting −BSY. This informs the initiator that the connection it is requesting has been established, so that it may remove the address information from the data lines.

f) The selection process is now complete, and the target controller (controller 5) is asserting −BSY. From this point on, controller 5 has control of the bus as required for the information transfer phase.

3. Information Transfer

Handshake signaling is used to control information transfers. In the scheme The −REQ and −ACK signals replace the Master_ready and Slave_ready signals. The target asserts −I/O during an input operation

Scanned with CamScanner

(target to initiator), and it asserts -C/D to indicate that the information being transferred is either a command or a status response rather than data.

At the end of the transfer, the target controller releases the -BSY signal, thus freeing the bus for use by other devices.

4. Reselection

a) When a logical connection is suspended and the target is ready to restore it, the target must first gain control of the bus.

b) It starts an arbitration cycle, and after winning arbitration, it selects the initiator ~~or~~ taken controller.

c) With the roles of the target and initiator reversed, the initiator is now asserting -BSY.

d) Before data transfer can begin, the initiator must hand control over to the target. This is achieved by having the target controller assert -BSY after selecting the initiator.
meanwhile, the initiator waits for a short period after being selected to make sure that the target has asserted -BSY, and then it releases the -BSY line.

e) The connection between the two controllers has now been reestablished, with the target in control of the bus as required for data transfer to proceed.

Scanned with CamScanner

UNIVERSAL SERIAL BUS (USB)

USB is an industry standard developed through a collaborative effort of several computer and communications companies, including Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, Nortel Networks, and Philips.

USB supports 3 speeds

1. low-speed (1.5 megabits/s)
2. full-speed (12 megabits/s)
3. high-speed (480 megabits/s).

Objectives:

1. provide a simple, low-cost, and easy to use interconnection system that overcomes the difficulties due to the limited number of I/O ports available on a computer.
2. Accommodate a wide range of data transfer characteristics for I/O devices, including telephone and Internet connections.
3. Enhance user convenience through a "plug-and-play" mode of operation.

Port Limitation

To make it possible to add many devices to a computer system at any time, without opening the computer box.

Device Characteristics

1. The kinds of devices that may be connected to a computer cover a wide range of functionality. The speed, volume, and timing constraints associated with data transfers to and from such devices

Scanned with CamScanner

vary significantly.

2. Since the event of pressing a key is not synchronized to any other event in a computer system, the data generated by the keyboard are called asynchronous.

3. A variety of simple devices that may be attached to a computer generate data of a similar nature - low speed and asynchronous. (Microphone)

4. The sampling process yields a continuous stream of digitized samples that arrive at a regular intervals, synchronized with the sampling clock. Such a data stream is called isochronous, meaning that successive events are separated by equal periods of time.

5. A signal must be sampled quickly enough to track its highest-frequency components. In general, if the sampling rate is S samples per second, the maximum frequency component that will be captured by the sampling process is S/2.

Plug-and-Play

The plug-and play feature means that a new device, such as an additional speaker can be connected at any time while the system is operating. The system should detect the existence of this new device automatically, identify the appropriate device-driver software and any other facilities needed to service that device, and establish the appropriate addresses and logical connections to enable them to communicate.

USB Architecture

To accommodate a large number of devices that can be added or removed at any time, the USB has the

Scanned with CamScanner

tree structure shown in Figure 3.26.

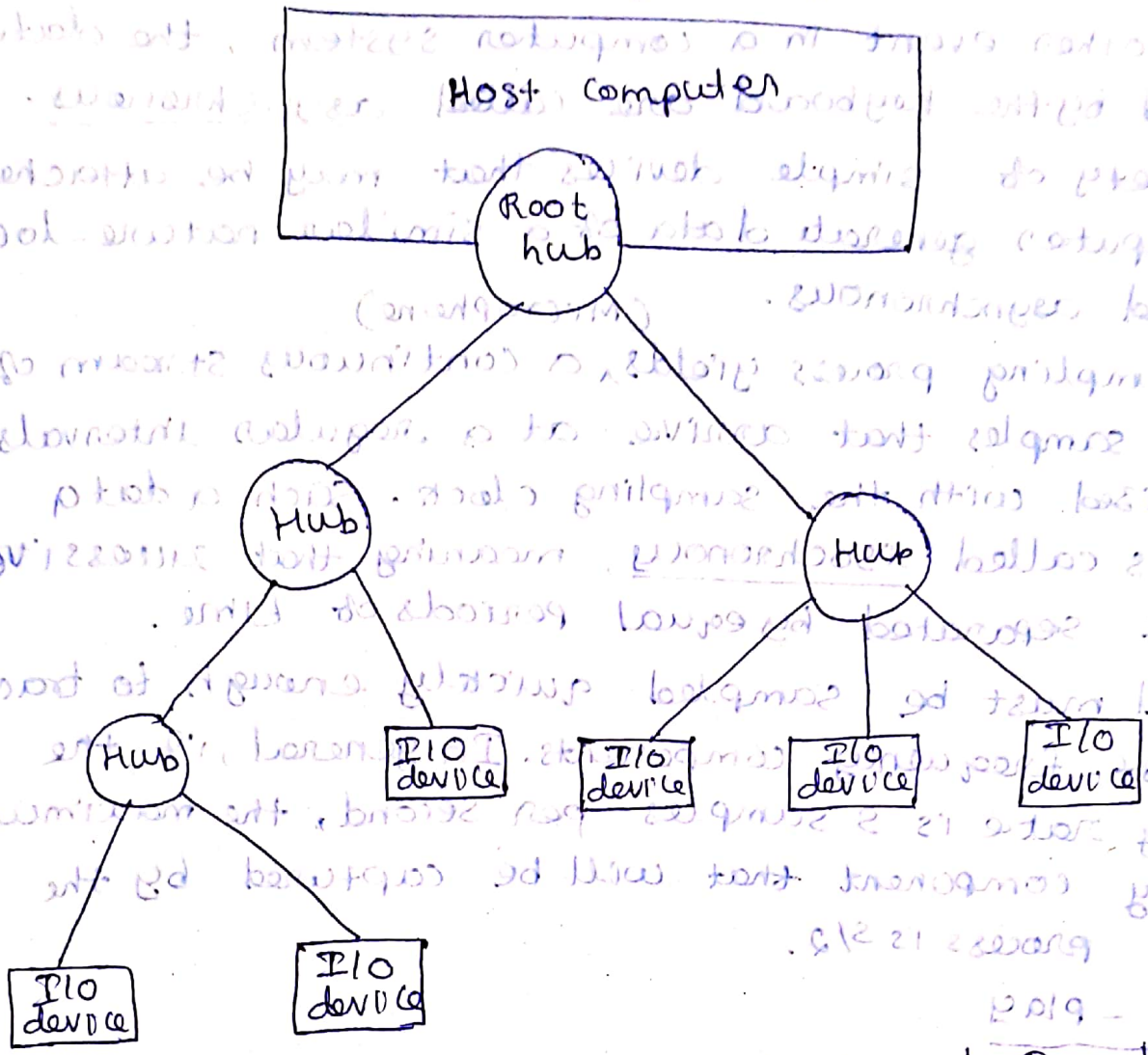


Figure 3.26 Universal Serial Bus tree structure.

1. Each node of the tree has a device called a hub, which acts as an intermediate control point between the host and the I/O devices.
2. At the root of the tree, a root hub connects the entire tree to the host computer.
3. The leaves of the tree are the I/O devices being served (for example, keyboard, Internet connection, speakers or digital TV), which are called functions in USB terminology.

Scanned with CamScanner

Working:

1. A hub copies a message that it receives from its upstream connection to all its downstream ports.
2. A message sent by the host computer is broadcast to all I/O devices, but only the addressed device will respond to that message.
3. The tree makes it possible to connect a large number of devices to a computer through a few ports (the root hub).
4. The USB operates strictly on the basis of polling. A device may send a message only in response to a poll message from the host.
5. Speed:

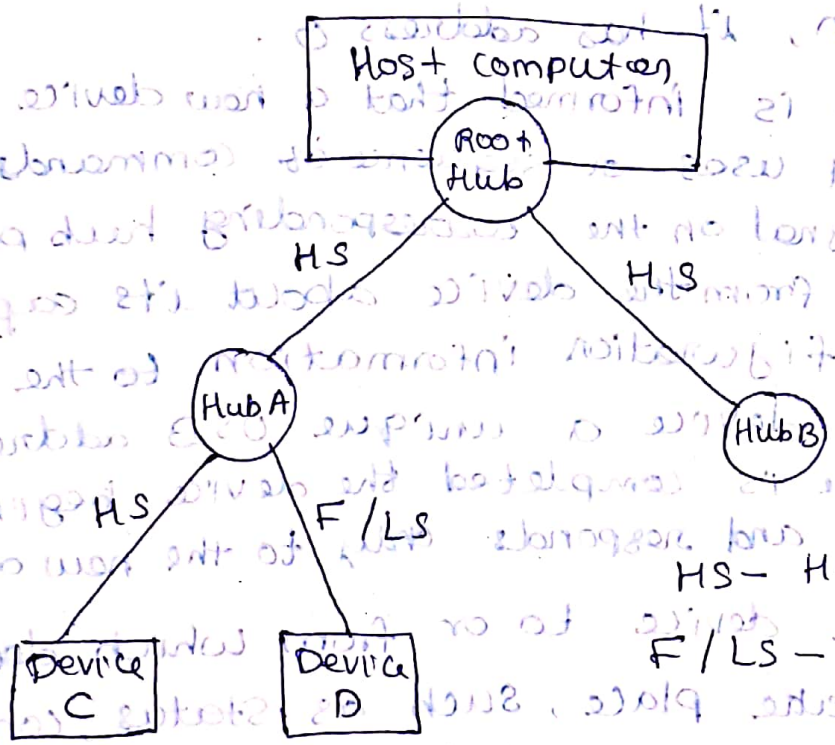


Figure 3-27 Split bus operation

6. The purpose of the USB software is to provide bidirectional communication links between application

Scanned with CamScanner

software and I/O devices. These links are called pipes. Any data entering at one end of a pipe is delivered at the other end.

Addressing

1. When a USB is connected to a host computer, its root hub is attached to the processor bus, where it appears as a single device.
2. Each device on the USB, whether it is a hub or an I/O device, is assigned a 7-bit address. This address is local to the USB tree and is not related in any way to the addresses used on the processor bus.
3. Addresses are assigned arbitrarily in the tree.
4. When a device is first connected to a hub, or when it is powered on, it has address 0.
5. When the host is informed that a new device has been connected, it uses a sequence of commands to send a reset signal on the corresponding hub port, read information from the device about its capabilities, send configuration information to the device, and assign the device a unique USB address. Once this sequence is completed the device begins normal operation and responds only to the new address.
6. Locations in the device to or from which data transfer can take place, such as status, control and data registers, are called endpoints. They are identified by a 4-bit number. Each 4-bit value identifies a pair of endpoints, one for input and one for output.

Scanned with CamScanner

USB protocols

1. All information transferred over the USB is organized in packets.

Packet - one or more bytes
- Control packet - addressing, acknowledge
- Data packet - input and output data.

Packet formats:

First field: packet identifier, PID - identifies the type of that packet - 4-bits - transmitted twice, first - true values, second - each bit complemented

PID_0	PID_1	PID_2	PID_3	PID_0	PID_1	PID_2	PID_3

(a) Packet identifier field

Bits	8	7	4	5
PID	ADR	ENDP	CRC16	

(b) Token packet, IN or OUT

Bits	8	0 to 8192	16
PID	DATA	CRC16	

(c) Data packet

Figure 3.28 USB Packet formats.

i) The four PID bits identify one of 16 different packet types.

ii) Control packets used for controlling data transfer operations are called token packets.

Figure 3.28(b) shows token packet format.

Scanned with CamScanner

PID - One of two PID - distinguish between input and output transfers, respectively.

7-bit address of a device

4-bit end point number within that device.

5-bits for error checking - cyclic redundancy check (CRC)

CRC - calculated based on address & endpoints field.

Data packets format is shown in Figure 3.28(c)

3 different PID, data packets do not carry a device address or an endpoint number. This information is included in the IN or OUT token packet that initiates the transfer.

Figure 3.29 shows an example for output operation.

1. The host computer sends a token packet of type OUT to the hub, followed by a data packet containing the output data.
2. The PID field of the data packet identifies it as data packet number 0.
3. The hub verifies that the transmission has been error free by checking the error control bits, then sends an acknowledgement packet (ACK) back to the host.
4. The hub forwards the token and data packets downstream.
5. All I/O devices receive this sequence of packets, but only the device that recognizes its address in the token packet accepts the data in the packet that follows.
6. After verifying that transmission has been error free, it sends an ACK packet to the hub.

Scanned with CamScanner

7. Successive data packets on a full-speed or less-speed pipe carry the numbers 0 and 1 alternately.

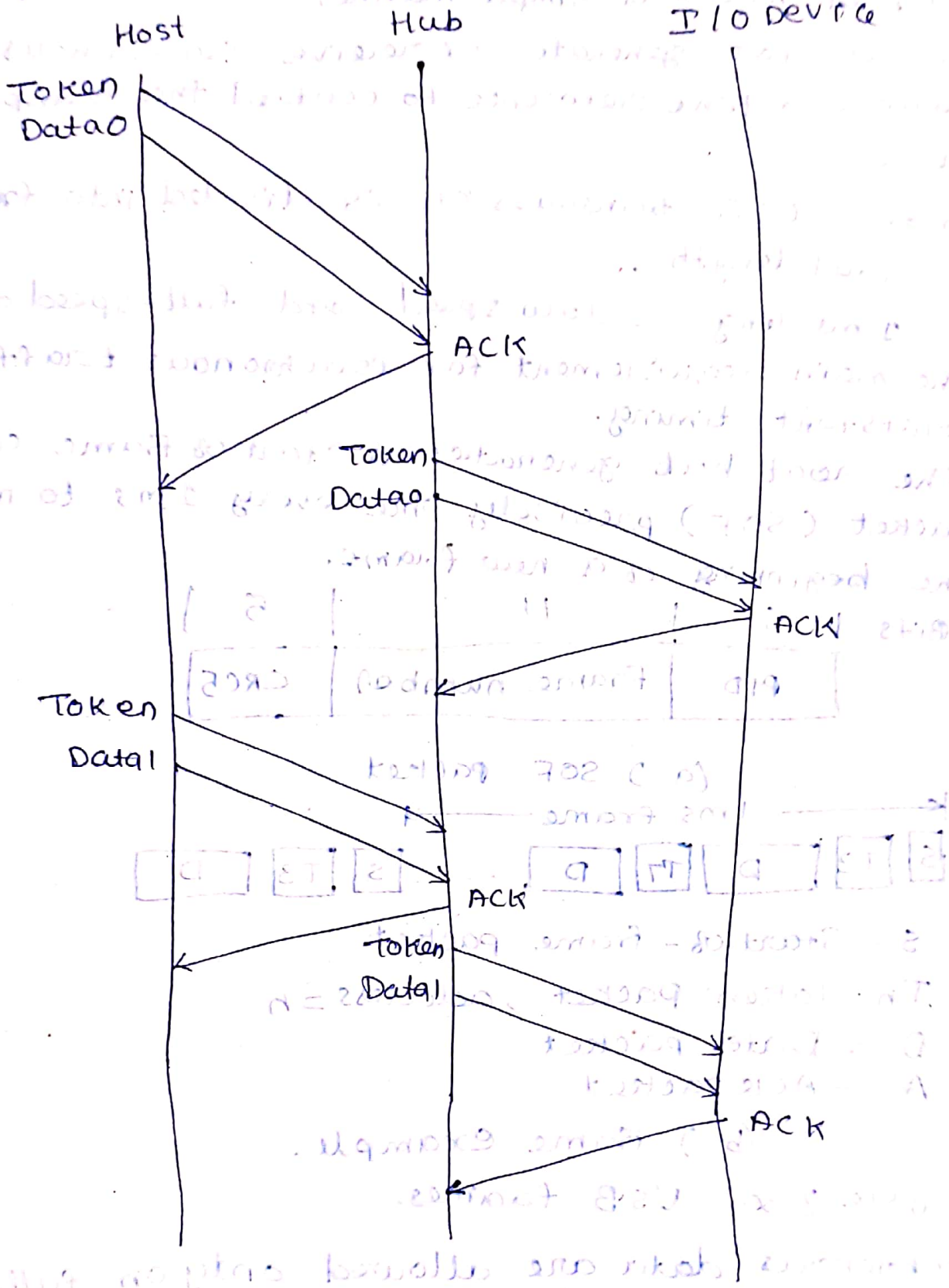


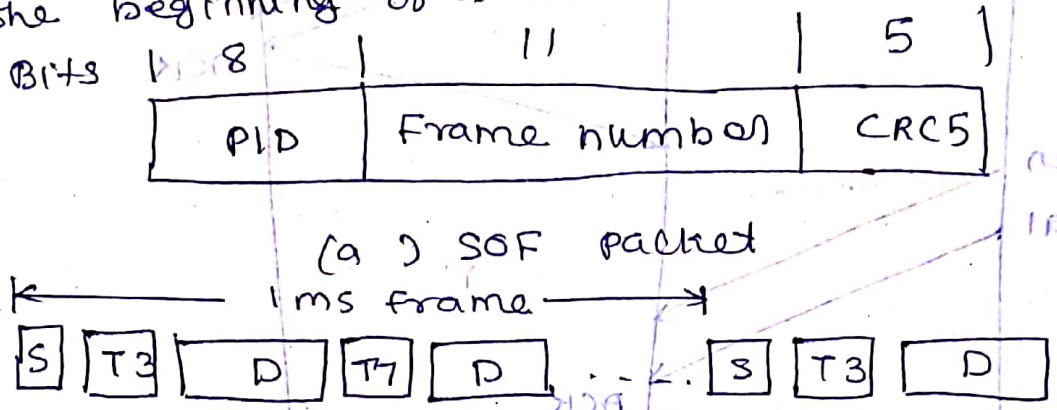
Figure 3.29 An output transfer.

Scanned with CamScanner

Isochronous Traffic on USB

Sampled voice in a simple manner

1. Devices that generate or receive isochronous data require a time reference to control the sampling proces.
2. For this USB transmission is divided into frames of equal length.
   1 ms long – low speed and full-speed data.
3. The main requirement for isochronous traffic is consistent timing.
4. The root hub generates a start of Frame control packet (SOF) precisely once every 1 ms to mark the beginning of a new frame.



(a) SOF packet

S – Start-of-frame packet.
Tn – Token packet, address = n
D = Data packet
A – ACK packet

(b) Frame example.

Figure 3.30 USB frames.

Isochronous data are allowed only on full-speed and high-speed links. For high-speed links, the SOF packet is repeated eight times at equal intervals.

Scanned with CamScanner

MODULE IV

MEMORY SYSTEM

BASIC CONCEPTS

1. Maximum size of memory that can be used in any computer is determined by addressing mode.

Address	Memory Locations
16 Bit	2^{16} = 64 K
32 Bit	2^{32} = 4G (Giga)
40 Bit	2^{40} = IT (Tera)

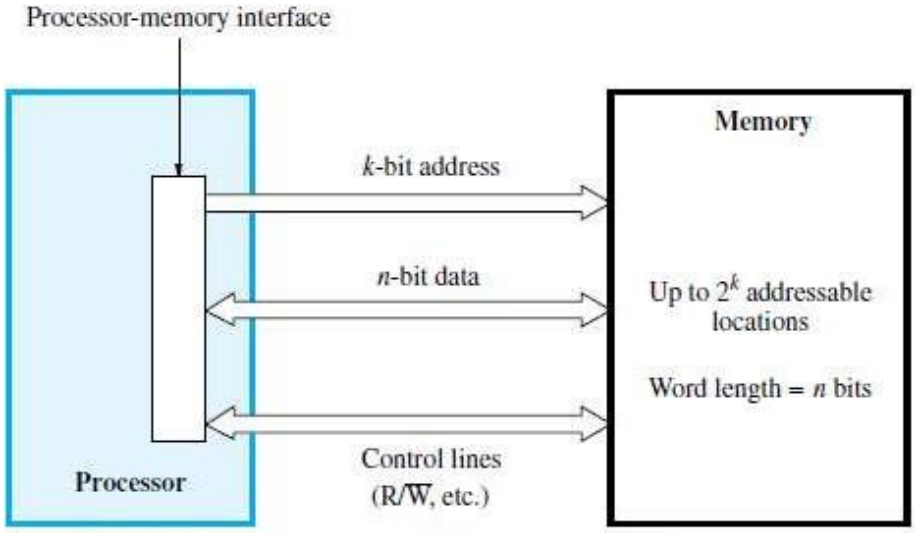


Figure 8.1 Connection of the memory to the processor.

2.If MAR is k-bits long then
→ memory may contain upto 2^K addressable-locations

3.If MDR is n-bits long, then
→ n-bits of data are transferred between the memory and processor.

4. The data-transfer takes place over the processor-bus (Figure 8.1).
5. The processor-bus has
   1) Address-Line
   2) Data-line &
   3) Control-Line (R/W‟, MFC – Memory Function Completed).
6. The Control-Line is used for coordinating data-transfer.
7. The processor reads the data from the memory by
   → loading the address of the required memory-location into MAR and
   → setting the R/W‟ line to 1.
8. The memory responds by
   → placing the data from the addressed-location onto the data-lines and
   → confirms this action by asserting MFC signal.
9. Upon receipt of MFC signal, the processor loads the data from the data-lines into MDR.
10. The processor writes the data into the memory-location by
    → loading the address of this location into MAR &
    → setting the R/W‟ line to 0.
11. **Memory Access Time:** It is the time that elapses between

→ initiation of an operation &
→ completion of that operation.

12. **Memory Cycle Time:** It is the minimum time delay that required between the initiation of the two successive memory-operations.

RAM (Random Access Memory)

1. In RAM, any location can be accessed for a Read/Write-operation in fixed amount of time,

Cache Memory

1. It is a small, fast memory that is inserted between
   → larger slower main-memory and
   → processor.
2. It holds the currently active segments of a program and their data.

Virtual Memory

1. The address generated by the processor is referred to as a **virtual/logical address**.
2. The virtual-address-space is mapped onto the physical-memory where data are actually stored.
3. The mapping-function is implemented by MMU. (MMU = memory management unit).
4. Only the active portion of the address-space is mapped into locations in the physical-memory.
5. The remaining virtual-addresses are mapped onto the bulk storage devices such as magnetic disk.
6. As the active portion of the virtual-address-space changes during program execution, the MMU
   → changes the mapping-function &
   → transfers the data between disk and memory.
7. During every memory-cycle, MMU determines whether the addressed-page is in the memory. If the page is in the memory.
   Then, the proper word is accessed and execution proceeds.
   Otherwise, a page containing desired word is transferred from disk to memory.

Memory can be classified as follows:

1) RAM which can be further classified as follows:
   i) Static RAM
   ii) Dynamic RAM (DRAM) which can be further classified as synchronous & asynchronous DRAM.
2) ROM which can be further classified as follows:
   i) PROM
   ii) EPROM
   iii) EEPROM &
   iv) Flash Memory which can be further classified as Flash Cards & Flash Drives.

SEMI CONDUCTOR RAM MEMORIES

INTERNAL ORGANIZATION OF MEMORY-CHIPS

1. Memory-cells are organized in the form of array (Figure 8.2).
2. Each cell is capable of storing 1-bit of information.
3. Each row of cells forms a memory-word.
4. All cells of a row are connected to a common line called as **Word-Line**.
5. The cells in each column are connected to **Sense/Write** circuit by 2-bit-lines.
6. The Sense/Write circuits are connected to data-input or output lines of the chip.
7. During a write-operation, the sense/write circuit
   → receive input information &
   → store input info in the cells of the selected word.

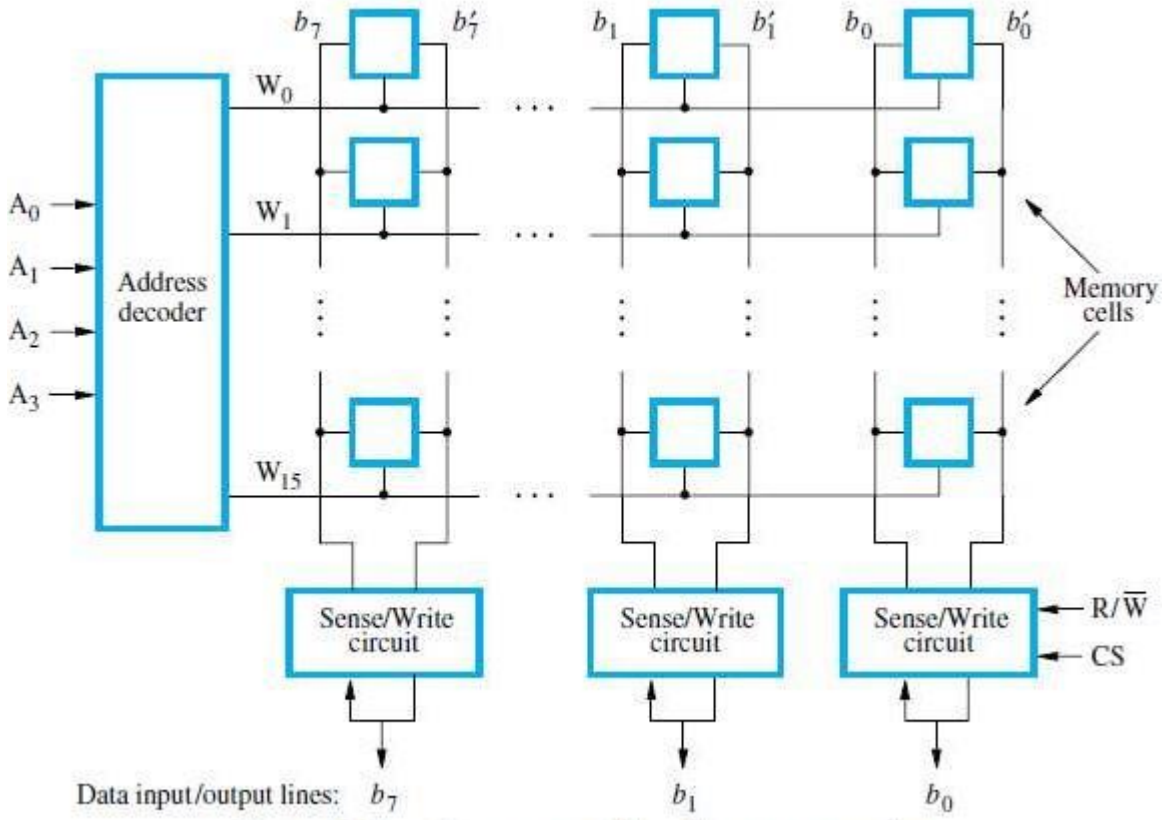


Figure 8.2 Organization of bit cells in a memory chip.

1. The data-input and data-output of each Sense/Write circuit are connected to a single bidirectional data-line.
2. Data-line can be connected to a data-bus of the computer.
3. Following 2 control lines are also used:
   1) R/W' □ Specifies the required operation.
   2) CS' □ Chip Select input selects a given chip in the multi-chip memory-system.

Bit Organization	Requirement of external connection for address, data and control lines
128 (16x8)	14
(1024) 128x8(1k)	19

STATIC RAM (OR MEMORY)

1. Memories consist of circuits capable of retaining their state as long as power is applied are known**.**

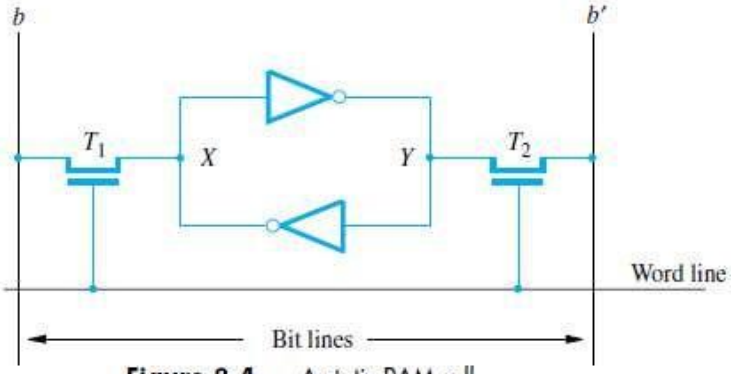


Figure 8.4 A static RAM cell.

2. Two inverters are cross connected to form a latch (Figure 8.4).
3. The latch is connected to 2-bit-lines by transistors T_1 and T_2.
4. The transistors act as switches that can be opened/closed under the control of the word-line.
5. When the word-line is at ground level, the transistors are turned off and the latch retain its state.

Read Operation

1. To read the state of the cell, the word-line is activated to close switches T_1 and T_2.
2. If the cell is in state 1, the signal on bit-line b is high and the signal on the bit-line b‟ is low.
3. Thus, b and b‟ are complement of each other.
4. Sense/Write circuit
   → monitors the state of b & b‟ and
   → sets the output accordingly.

Write Operation

1. The state of the cell is set by
   → placing the appropriate value on bit-line b and its complement on b‟ and
   → then activating the word-line. This forces the cell into the corresponding state.
2. The required signal on the bit-lines is generated by Sense/Write circuit.

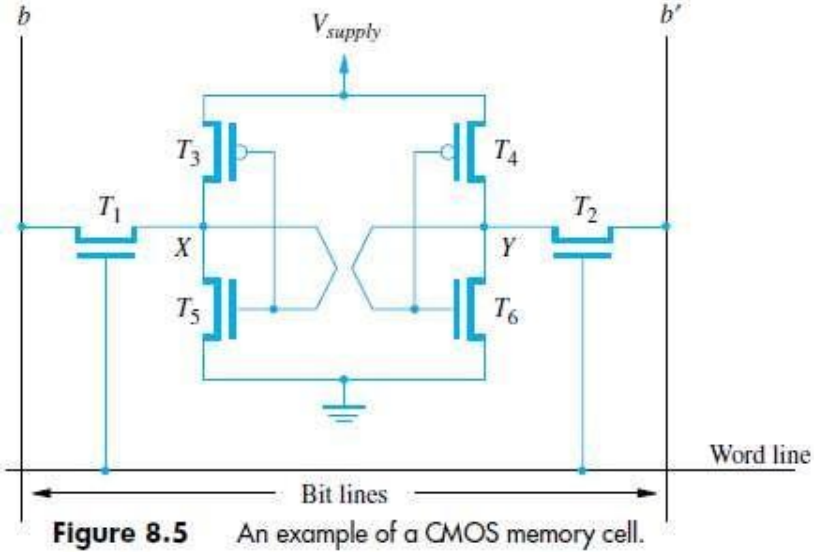


Figure 8.5 An example of a CMOS memory cell.

CMOS Cell

1. Transistor pairs (T_3, T_5) and (T_4, T_6) form the inverters in the latch (Figure 8.5).
2. In state 1, the voltage at point X is high by having T_5, T_6 ON and T_4, T_5 are OFF.
3. Thus, T_1 and T_2 returned ON (Closed), bit-line b and b‟ will have high and low signals respectively.
4. **Advantages:**
   1) It has low power consumption „.“ the current flows in the cell only when the cell is active.
   2) Static RAM‟s can be accessed quickly. It access time is few nanoseconds.

5. **Disadvantage:** SRAMs are said to be volatile memories „.“ their contents are lost when power is interrupted.

ASYNCHRONOUS DRAM

1. Less expensive RAMs can be implemented if simple cells are used.
2. Such cells cannot retain their state indefinitely. Hence they are called **Dynamic RAM (DRAM).**
3. The information stored in a dynamic memory-cell in the form of a charge on a capacitor.
4. This charge can be maintained only for tens of milliseconds.
5. The contents must be periodically refreshed by restoring this capacitor charge to its full value.

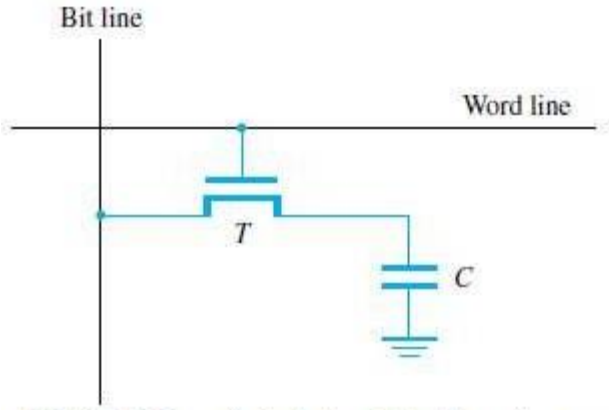


Figure 8.6 A single-transistor dynamic memory cell.

6. In order to store information in the cell, the transistor T is turned „ON“ (Figure 8.6).
7. The appropriate voltage is applied to the bit-line which charges the capacitor.
8. After the transistor is turned off, the capacitor begins to discharge.
9. Hence, info. stored in cell can be retrieved correctly before threshold value of capacitor drops down.
10. During a read-operation,
    → transistor is turned „ON“
    → a sense amplifier detects whether the charge on the capacitor is above the threshold value.
        1. If (charge on capacitor) > (threshold value) □ Bit-line will have logic value „1“.
        2. If (charge on capacitor) < (threshold value) □ Bit-line will set to logic value „0“.

ASYNCHRONOUS DRAM DESCRIPTION

1. The 4 bit cells in each row are divided into 512 groups of 8 (Figure 5.7).
2. 21 bit address is needed to access a byte in the memory. 21 bit is divided as follows:
    1) 12 address bits are needed to select a row.
        i.e. A_{8-0} → specifies row-address of a byte.
    2) 9 bits are needed to specify a group of 8 bits in the selected row.
        i.e. A_{20-9} → specifies column-address of a byte.

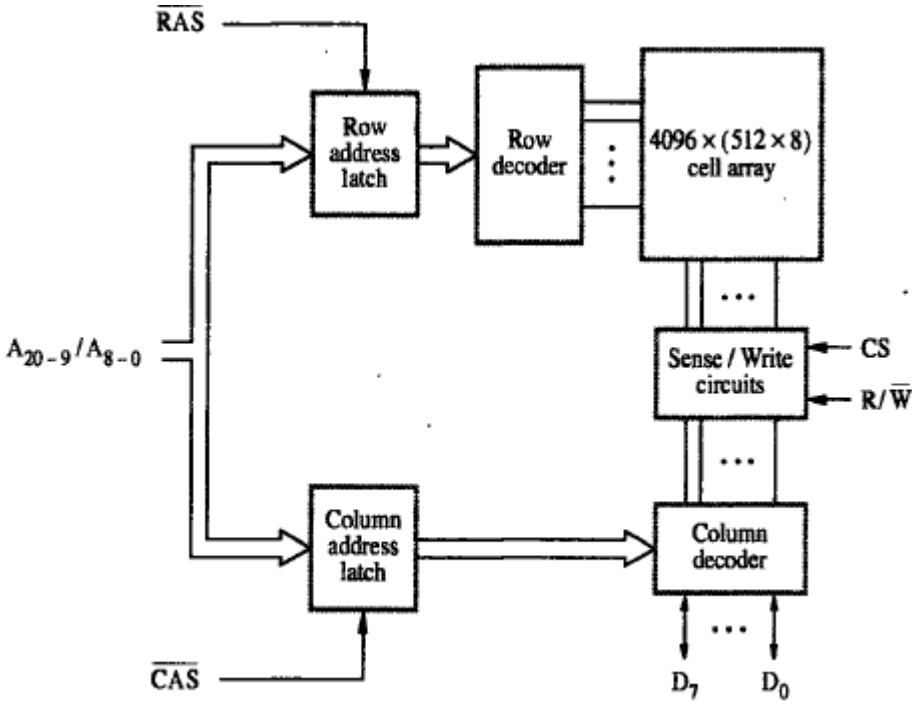


Figure 5.7 Internal organization of a 2M × 8 dynamic memory chip

3. During Read/Write-operation,
   → row-address is applied first.
   → row-address is loaded into row-latch in response to a signal pulse on **RAS'** input of chip.
   (RAS = Row-address Strobe CAS = Column-address Strobe)
4. When a Read-operation is initiated, all cells on the selected row are read and refreshed.
5. Shortly after the row-address is loaded, the column-address is
   → applied to the address pins &
   → loaded into **CAS'.**
6. The information in the latch is decoded.
7. The appropriate group of 8 Sense/Write circuits is selected.
   R/W'=1(read-operation) Output values of selected circuits are transferred to data-lines D_0-D_7.
   R/W'=0(write-operation) Information on D_0-D_7 are transferred to the selected circuits.
8. RAS‟ & CAS‟ are active-low so that they cause latching of address when they change from high to low.
9. To ensure that the contents of DRAMs are maintained, each row of cells is accessed periodically.
10. A special memory-circuit provides the necessary control signals RAS‟ & CAS‟ that govern the timing.
11. The processor must take into account the delay in the response of the memory.

Fast Page Mode

1. Transferring the bytes in sequential order is achieved by applying the consecutive sequence of column-address under the control of successive CAS‟ signals.
2. This scheme allows transferring a block of data at a faster rate.
3. The block of transfer capability is called as *fast page mode***.**

SYNCHRONOUS DRAM

1. The operations are directly synchronized with clock signal (Figure 8.8).
2. The address and data connections are buffered by means of registers.

3. The output of each sense amplifier is connected to a latch.
4. A Read-operation causes the contents of all cells in the selected row to be loaded in these latches.
5. Data held in latches that correspond to selected columns are transferred into data-output register.
6. Thus, data becoming available on the data-output pins.

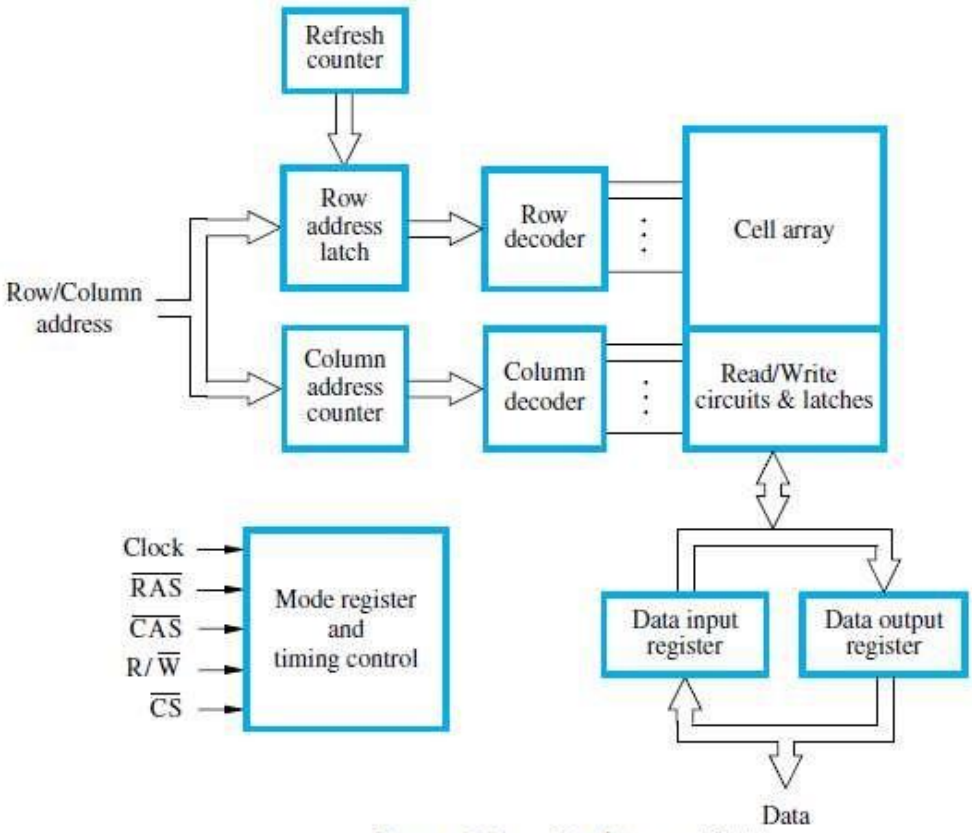


Figure 8.8 Synchronous DRAM.

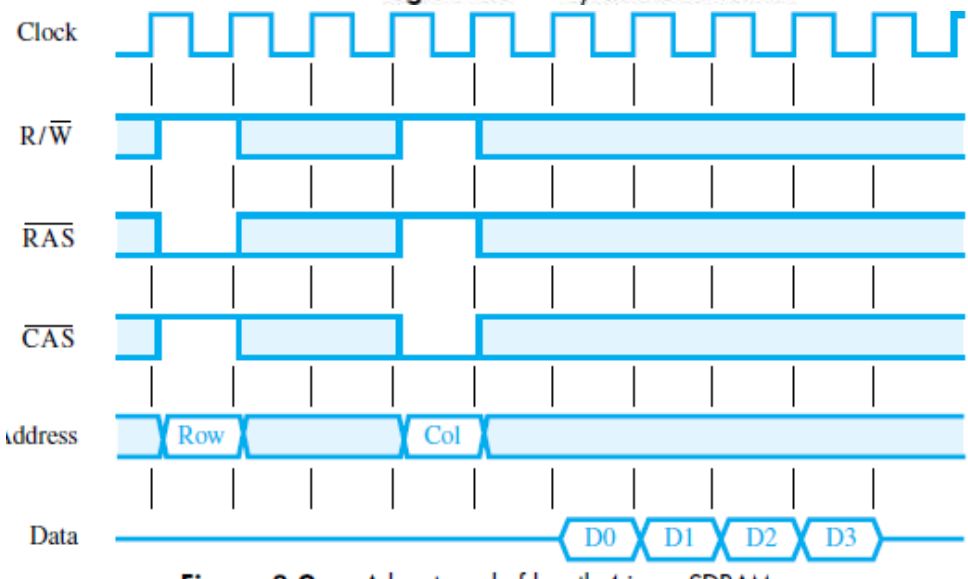


Figure 8.9 A burst read of length 4 in an SDRAM.

1. First, the row-address is latched under control of RAS‟ signal (Figure 8.9).
2. The memory typically takes 2 or 3 clock cycles to activate the selected row.

3. Then, the column-address is latched under the control of CAS‟ signal.
4. After a delay of one clock cycle, the first set of data bits is placed on the data-lines.
5. SDRAM automatically increments column-address to access next 3 sets of bits in the selected row.

LATENCY & BANDWIDTH

1. A good indication of performance is given by 2 parameters: 1) Latency 2) Bandwidth.

Latency

1. It refers to the amount of time it takes to transfer a word of data to or from the memory.
2. For a transfer of single word, the latency provides the complete indication of memory performance.
3. For a block transfer, the latency denotes the time it takes to transfer the first word of data.

Bandwidth

1. It is defined as the number of bits or bytes that can be transferred in one second.
2. Bandwidth mainly depends on
   1) The speed of access to the stored data &
   2) The number of bits that can be accessed in parallel.

DOUBLE DATA RATE SDRAM (DDR-SDRAM)

1. The standard SDRAM performs all actions on the rising edge of the clock signal.
2. The DDR-SDRAM transfer data on both the edges (loading edge, trailing edge).
3. The Bandwidth of DDR-SDRAM is doubled for long burst transfer.
4. To make it possible to access the data at high rate, the cell array is organized into two banks.
5. Each bank can be accessed separately.
6. Consecutive words of a given block are stored in different banks.
7. Such interleaving of words allows simultaneous access to two words.
8. The two words are transferred on successive edge of the clock.

STRUCTURE OF LARGER MEMORIES

Dynamic Memory System

1. The physical implementation is done in the form of memory-modules.
2. If a large memory is built by placing DRAM chips directly on the Motherboard, then it will occupy large amount of space on the board.
3. These packaging consideration have led to the development of larger memory units known as SIMM‟s & DIMM‟s.
   1) SIMM Single Inline memory-module
   2) DIMM Dual Inline memory-module
4. SIMM/DIMM consists of many memory-chips on small board that plugs into a socket on motherboard.

MEMORY-SYSTEM CONSIDERATION MEMORY CONTROLLER

1. To reduce the number of pins, the dynamic memory-chips use multiplexed-address inputs.
2. The address is divided into 2 parts:

1) High Order Address Bit

1. Select a row in cell array.
2. It is provided first and latched into memory-chips under the control of RAS‟ signal.

2) Low Order Address Bit

3. Selects a column.
4. They are provided on same address pins and latched using CAS‟ signals.

3. The Multiplexing of address bit is usually done by **Memory Controller Circuit** (Figure 5.11).

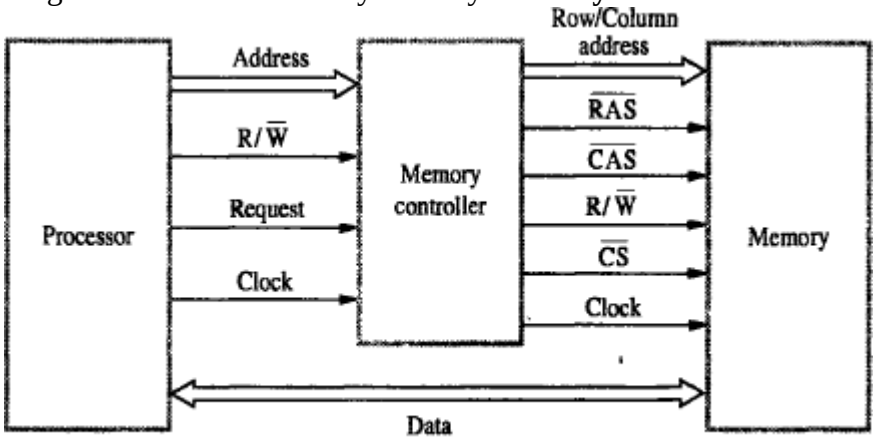


Figure 5.11 Use of a memory controller.

4. The Controller accepts a complete address & R/W‟ signal from the processor.
5. A Request signal indicates a memory access operation is needed.
6. Then, the Controller
   → forwards the row & column portions of the address to the memory.
   → generates RAS‟ & CAS‟ signals &
   → sends R/W‟ & CS‟ signals to the memory.

RAMBUS MEMORY

1. The usage of wide bus is expensive.
2. Rambus developed the implementation of narrow bus.
3. Rambus technology is a fast signaling method used to transfer information between chips.
4. The signals consist of much smaller voltage swings around a reference voltage V_{ref}.
5. The reference voltage is about 2V.
6. The two logical values are represented by 0.3V swings above and below V_{ref}.
7. This type of signaling is generally is known as **Differential Signalling**.
8. Rambus provides a complete specification for design of communication called as **Rambus Channel**.
9. Rambus memory has a clock frequency of 400 MHz.
10. The data are transmitted on both the edges of clock so that effective data-transfer rate is 800MHZ.
11. Circuitry needed to interface to Rambus channel is included on chip. Such chips are called **RDRAM.** (RDRAM = Rambus DRAMs).
12. Rambus channel has:
    1) 9 Data-lines (1^{st}-8^{th} line ->Transfer the data, 9^{th} line->Parity checking).
    2) Control-Line &
    3) Power line.
13. A two channel rambus has 18 data-lines which has no separate Address-Lines.
14. Communication between processor and RDRAM modules is carried out by means of packets transmitted on the data-lines.
15. There are 3 types of packets:
    1) Request
    2) Acknowledge &
    3) Data.

READ ONLY MEMORY (ROM)

1. Both SRAM and DRAM chips are volatile, i.e. They lose the stored information if power is turned off.
2. Many application requires non-volatile memory which retains the stored information if power is turned off.
3. For ex:

OS software has to be loaded from disk to memory i.e. it requires non-volatile memory.

4. Non-volatile memory is used in embedded system.
5. Since the normal operation involves only reading of stored data, a memory of this type is called ROM.
   1. **At Logic value '0'** □ Transistor(T) is connected to the ground point (P). Transistor switch is closed & voltage on bit-line nearly drops to zero (Figure 8.11).
   2. **At Logic value '1'** □ Transistor switch is open. The bit-line remains at high voltage.

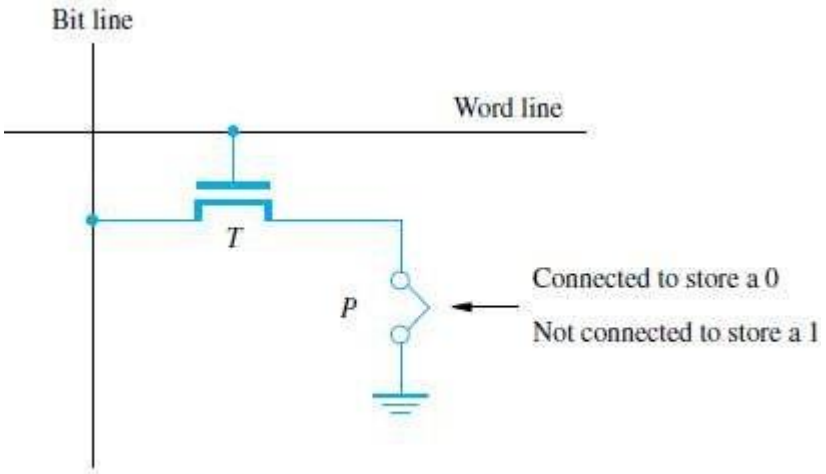


Figure 8.11 A ROM cell.

6. To read the state of the cell, the word-line is activated.
7. A Sense circuit at the end of the bit-line generates the proper output value.

TYPES OF ROM

1. Different types of non-volatile memory are
   1) PROM
   2) EPROM
   3) EEPROM &
   4) Flash Memory (Flash Cards & Flash Drives)

PROM (PROGRAMMABLE ROM)

1. PROM allows the data to be loaded by the user.
2. Programmability is achieved by inserting a „fuse" at point P in a ROM cell.
3. Before PROM is programmed, the memory contains all 0"s.
4. User can insert 1"s at required location by burning-out fuse using high current-pulse.
5. This process is irreversible.
6. **Advantages:**
   1) It provides flexibility.
   2) It is faster.
   3) It is less expensive because they can be programmed directly by the user.

Both SRAM and DRAM chips are volatile, i.e. They lose the stored information if power is turned off.

7. Many application requires non-volatile memory which retains the stored information if power is turned off.
8. For ex:

OS software has to be loaded from disk to memory i.e. it requires non-volatile memory.

9. Non-volatile memory is used in embedded system.
10. Since the normal operation involves only reading of stored data, a memory of this type is called ROM.
    1. **At Logic value '0'** □ Transistor(T) is connected to the ground point (P).
       Transistor switch is closed & voltage on bit-line nearly drops to zero (Figure 8.11).
    2. **At Logic value '1'** □ Transistor switch is open. The bit-line remains at high voltage.

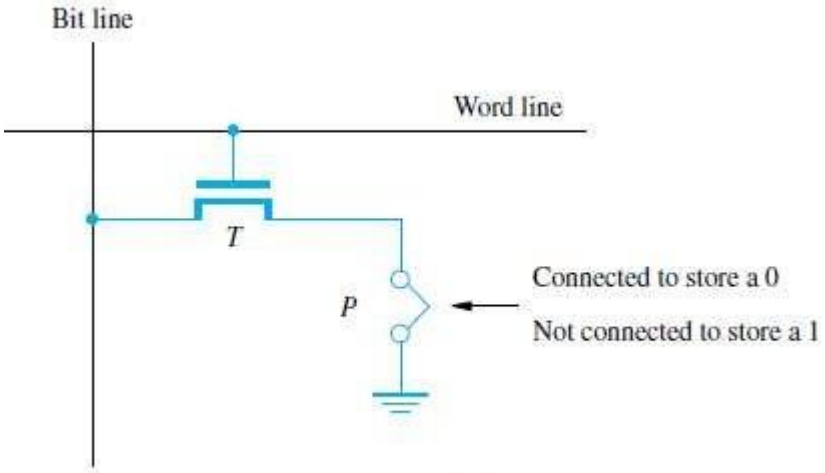


Figure 8.11 A ROM cell.

11. To read the state of the cell, the word-line is activated.
12. A Sense circuit at the end of the bit-line generates the proper output value.

TYPES OF ROM

1. Different types of non-volatile memory are
   1) PROM
   2) EPROM
   3) EEPROM &
   4) Flash Memory (Flash Cards & Flash Drives)

PROM (PROGRAMMABLE ROM)

1. PROM allows the data to be loaded by the user.
2. Programmability is achieved by inserting a „fuse" at point P in a ROM cell.
3. Before PROM is programmed, the memory contains all 0"s.
4. User can insert 1"s at required location by burning-out fuse using high current-pulse.
5. This process is irreversible.
6. **Advantages:**
   1) It provides flexibility.
   2) It is faster.
   3) It is less expensive because they can be programmed directly by the user.

EPROM (ERASABLE REPROGRAMMABLE ROM)

1. EPROM allows
   → stored data to be erased and
   → new data to be loaded.

2.In cell, a connection to ground is always made at „P" and a special transistor is used.

3. The transistor has the ability to function as
   → a normal transistor or
   → a disabled transistor that is always turned „off".
4. Transistor can be programmed to behave as a permanently open switch, by injecting charge into it.
5. Erasure requires dissipating the charges trapped in the transistor of memory-cells. This can be done by exposing the chip to ultra-violet light.
6. **Advantages:**
   1) It provides flexibility during the development-phase of digital-system.
   2) It is capable of retaining the stored information for a long time.
7. **Disadvantages:**
   1) The chip must be physically removed from the circuit for reprogramming.
   2) The entire contents need to be erased by UV light.

EEPROM (ELECTRICALLY ERASABLE ROM)

1. **Advantages:**
   1) It can be both programmed and erased electrically.
   2) It allows the erasing of all cell contents selectively.
2. **Disadvantage:** It requires different voltage for erasing, writing and reading the stored data.

FLASH MEMORY

1. In EEPROM, it is possible to read & write the contents of a single cell.
2. In Flash device, it is possible to read contents of a single cell & write entire contents of a block.
3. Prior to writing, the previous contents of the block are erased.
   Eg. In MP3 player, the flash memory stores the data that represents sound.
4. Single flash chips cannot provide sufficient storage capacity for embedded-system.
5. **Advantages:**
   1) Flash drives have greater density which leads to higher capacity & low cost per bit.
   2) It requires single power supply voltage & consumes less power.
6. There are 2 methods for implementing larger memory: 1) Flash Cards & 2) Flash Drives

1) Flash Cards

1. One way of constructing larger module is to mount flash-chips on a small card.
2. Such flash-card have standard interface.
3. The card is simply plugged into a conveniently accessible slot.
4. Memory-size of the card can be 8, 32 or 64MB.
5. Eg: A minute of music can be stored in 1MB of memory. Hence 64MB flash cards can store an hour of music.

2) Flash Drives

6. Larger flash memory can be developed by replacing the hard disk-drive.
7. The flash drives are designed to fully emulate the hard disk.
8. The flash drives are solid state electronic devices that have no movable parts.

Advantages:

1) They have shorter seek & access time which results in faster response.
2) They have low power consumption. .". they are attractive for battery driven application.
3) They are insensitive to vibration.

Disadvantages:

1) The capacity of flash drive (<1GB) is less than hard disk (>1GB).
2) It leads to higher cost per bit.
3) Flash memory will weaken after it has been written a number of times (typically at least 1 million times).

SPEED, SIZE COST

Characteristics	SRAM	DRAM	Magnetis Disk
Speed	Very Fast	Slower	Much slower than DRAM
Size	Large	Small	Small
Cost	Expensive	Less Expensive	Low price

Memory	Speed	Size	Cost
Registers	Very high	Lower	Very Lower
Primary cache	High	Lower	Low
Secondary cache	Low	Low	Low
Main memory	Lower than Seconadry cache	High	High
Secondary Memory	Very low	Very High	Very High

13. The main-memory can be built with DRAM (Figure 8.14)
14. Thus, SRAM‟s are used in smaller units where speed is of essence.
15. The Cache-memory is of 2 types:
    1) Primary/Processor Cache (Level1 or L1 cache)
    1. It is always located on the processor-chip.
    2) Secondary Cache (Level2 or L2 cache)
    2. It is placed between the primary-cache and the rest of the memory.
16. The memory is implemented using the dynamic components (SIMM, RIMM, DIMM).
17. The access time for main-memory is about 10 times longer than the access time for L1 cache.

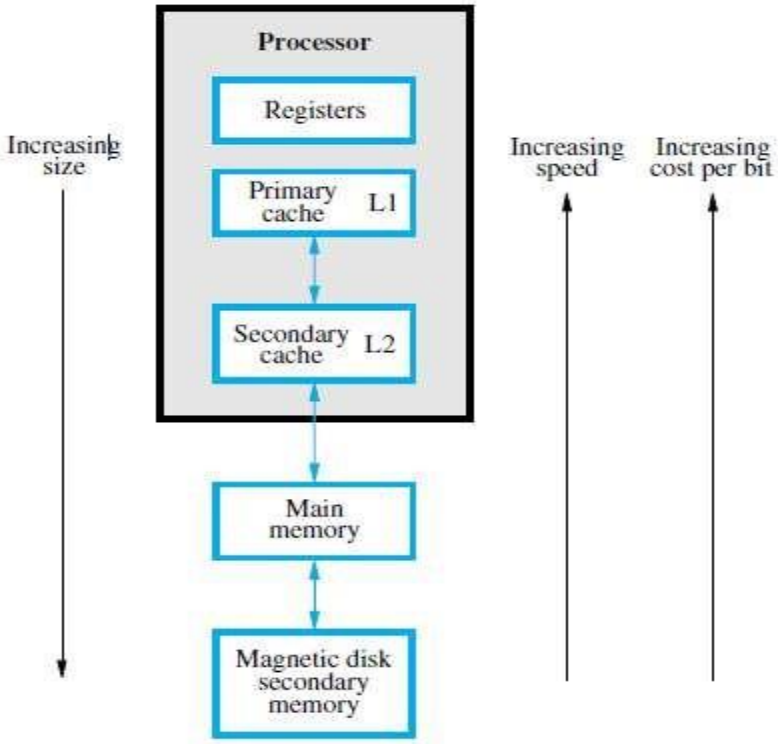


Figure 8.14 Memory hierarchy.

CACHE MEMORIES

1. The effectiveness of cache mechanism is based on the property of „**Locality of Reference'.**

Locality of Reference

1. Many instructions in the localized areas of program are executed repeatedly during some time period
2. Remainder of the program is accessed relatively infrequently (Figure 8.15).
3. There are 2 types:

1) Temporal

1. The recently executed instructions are likely to be executed again very soon.

2) Spatial

2. Instructions in close proximity to recently executed instruction are also likely to be executed soon.

4.If active segment of program is placed in cache-memory, then total execution time can be reduced.

5. **Block** refers to the set of contiguous address locations of some size.
6. The cache-line is used to refer to the cache-block.

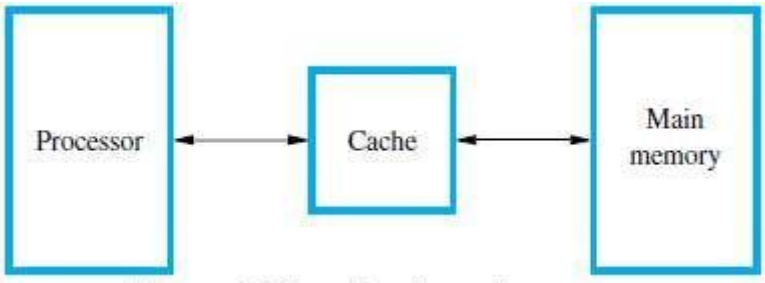


Figure 8.15 Use of a cache memory.

1. The Cache-memory stores a reasonable number of blocks at a given time.

2. This number of blocks is small compared to the total number of blocks available in main-memory.
3. Correspondence b/w main-memory-block & cache-memory-block is specified by mapping-function.
4. Cache control hardware decides which block should be removed to create space for the new block.
5. The collection of rule for making this decision is called the **Replacement Algorithm.**
6. The cache control-circuit determines whether the requested-word currently exists in the cache.
7. The write-operation is done in 2 ways: 1) Write-through protocol & 2) Write-back protocol.

Write-Through Protocol

1. Here the cache-location and the main-memory-locations are updated simultaneously.

Write-Back Protocol

2. This technique is to
   → update only the cache-location &
   → mark the cache-location with associated flag bit called **Dirty/Modified Bit.**
3. The word in memory will be updated later, when the marked-block is removed from cache.

During Read-operation

1. If the requested-word currently not exists in the cache, then **read-miss** will occur.
2. To overcome the read miss, *Load–through/Early restart protocol* is used.

Load–Through Protocol

1. The block of words that contains the requested-word is copied from the memory into cache.
2. After entire block is loaded into cache, the requested-word is forwarded to processor.

During Write-operation

1. If the requested-word not exists in the cache, then **write-miss** will occur.
   1) If **Write Through Protocol** is used, the information is written directly into main-memory.
   2) If **Write Back Protocol** is used,
      → then block containing the addressed word is first brought into the cache &
      → then the desired word in the cache is over-written with the new information.

MAPPING-FUNCTION

1. Here we discuss about 3 different mapping-function:
   1) Direct Mapping
   2) Associative Mapping
   3) Set-Associative Mapping

DIRECT MAPPING

1. The block-j of the main-memory maps onto block-j modulo-128 of the cache (Figure 8.16).
2. When the memory-blocks 0, 128, & 256 are loaded into cache, the block is stored in cache-block 0. Similarly, memory-blocks 1, 129, 257 are stored in cache-block 1.
3. The contention may arise when
   1) When the cache is full.
   2) When more than one memory-block is mapped onto a given cache-block position.
4. The contention is resolved by
   allowing the new blocks to overwrite the currently resident-block.

5. Memory-address determines placement of block in the cache.

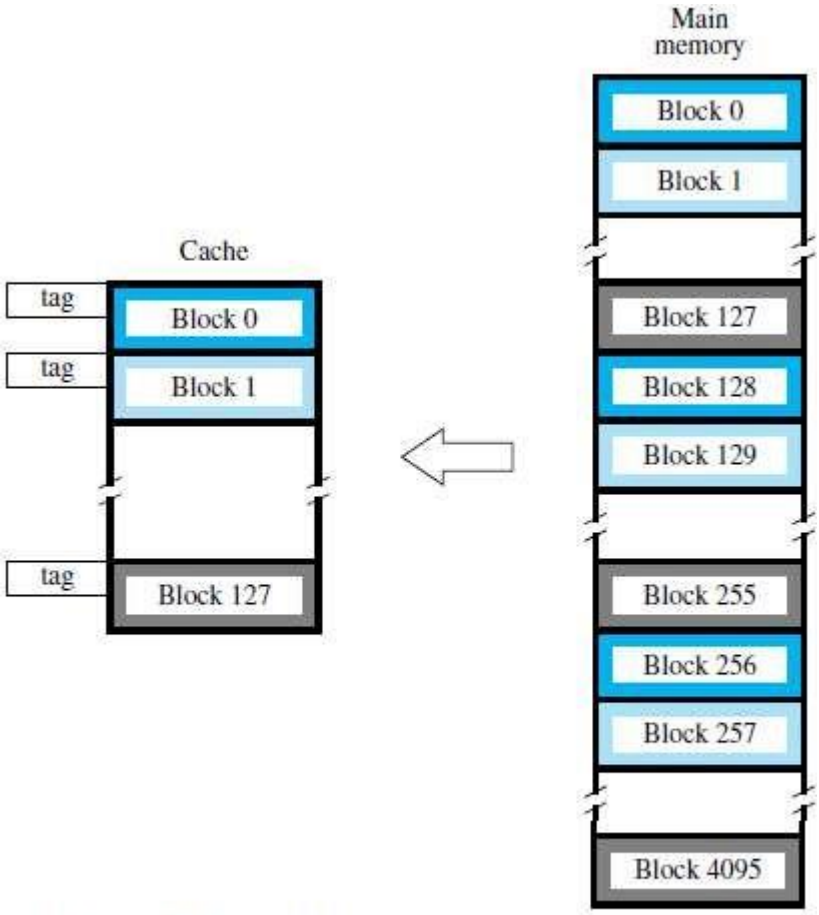


Figure 8.16 Direct-mapped cache.

1. The memory-address is divided into 3 fields:
   1) Low Order 4 bit field
   1. Selects one of 16 words in a block.
   2) 7 bit cache-block field
   2. 7-bits determine the cache-position in which new block must be stored.
   3) 5 bit Tag field
   3. 5-bits memory-address of block is stored in 5 tag-bits associated with cache-location.
2. As execution proceeds,
   5-bit tag field of memory-address is compared with tag-bits associated with cache-location.
   If they match, then the desired word is in that block of the cache.
   Otherwise, the block containing required word must be first read from the memory.
   And then the word must be loaded into the cache.

ASSOCIATIVE MAPPING

1. The memory-block can be placed into any cache-block position. (Figure 8.17).
2. 12 tag-bits will identify a memory-block when it is resolved in the cache.
3. Tag-bits of an address received from processor are compared to the tag-bits of each block of cache.
4. This comparison is done to see if the desired block is present.

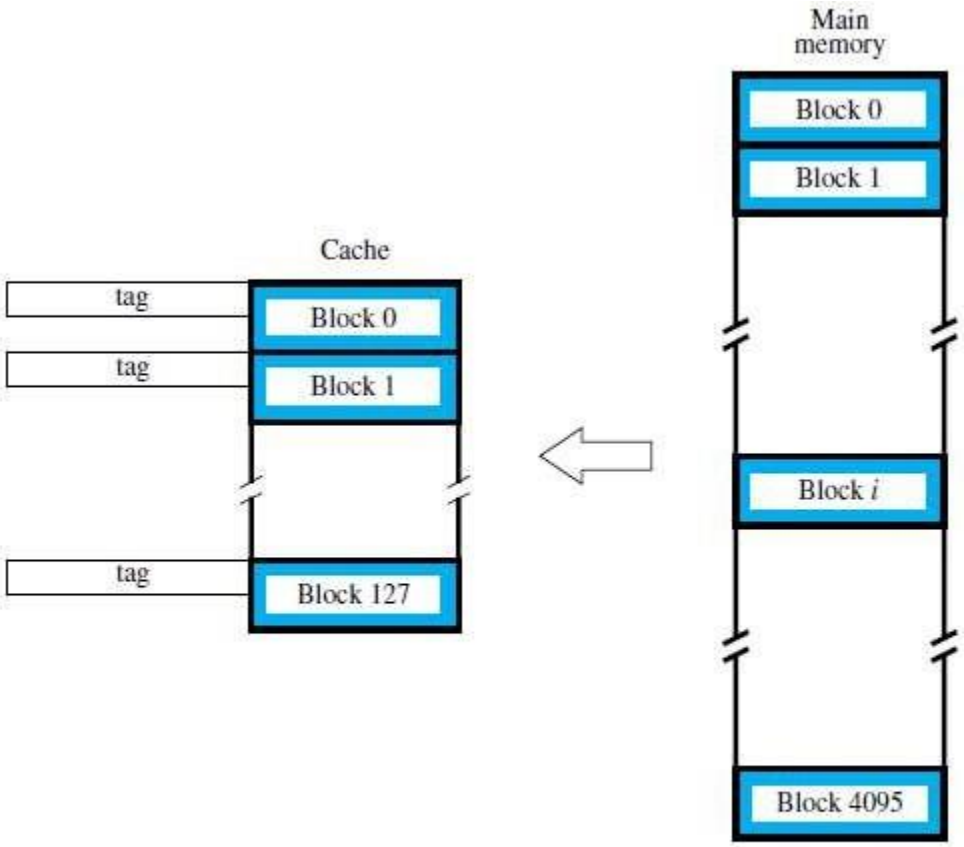


Figure 8.17 Associative-mapped cache.

1. It gives complete freedom in choosing the cache-location.
2. A new block that has to be brought into the cache has to replace an existing block if the cache is full.
3. The memory has to determine whether a given block is in the cache.
4. **Advantage:** It is more flexible than direct mapping technique.
5. **Disadvantage:** Its cost is high.

SET-ASSOCIATIVE MAPPING

1. It is the combination of direct and associative mapping. (Figure 8.18).
2. The blocks of the cache are grouped into sets.
3. The mapping allows a block of the main-memory to reside in any block of the specified set.
4. The cache has 2 blocks per set, so the memory-blocks 0, 64, 128… 4032 maps into cache set „0".
5. The cache can occupy either of the two block position within the set.

6 bit set field

1. Determines which set of cache contains the desired block.

6 bit tag field

2. The tag field of the address is compared to the tags of the two blocks of the set.
3. This comparison is done to check if the desired block is present.

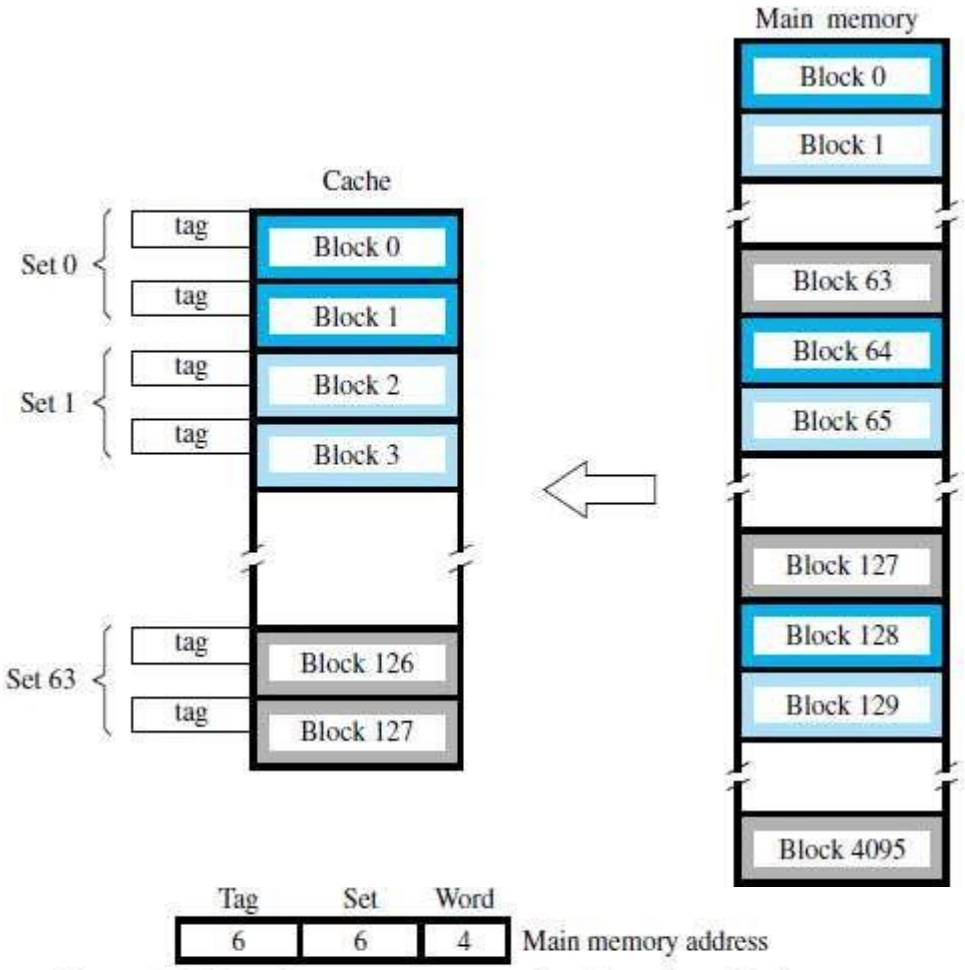


Figure 8.18 Set-associative-mapped cache with two blocks per set.

1. The cache which contains 1 block per set is called **direct mapping.**

2. A cache that has „k" blocks per set is called as "**k-way set associative cache**".
3. Each block contains a control-bit called a **valid-bit**.
4. The Valid-bit indicates that whether the block contains valid-data.
5. The dirty bit indicates that whether the block has been modified during its cache residency.
   Valid-bit=0 □ When power is initially applied to system.
   Valid-bit=1 □ When the block is loaded from main-memory at first time.
6. If the main-memory-block is updated by a source & if the block in the source is already exists in the cache, then the valid-bit will be cleared to "0".
7. If Processor & DMA uses the same copies of data then it is called as **Cache Coherence Problem**.
8. **Advantages:**
   1) Contention problem of direct mapping is solved by having few choices for block placement.
   2) The hardware cost is decreased by reducing the size of associative search.

REPLACEMENT ALGORITHM

1. In direct mapping method,

the position of each block is pre-determined and there is no need of replacement strategy.

2.In associative & set associative method,

The block position is not pre-determined.

If the cache is full and if new blocks are brought into the cache,

then the cache-controller must decide which of the old blocks has to be replaced.

3. When a block is to be overwritten, the block with longest time w/o being referenced is over-written.
4. This block is called **Least recently Used (LRU) block** & the technique is called **LRU algorithm**.
5. The cache-controller tracks the references to all blocks with the help of block-counter.
6. **Advantage:** Performance of LRU is improved by randomness in deciding which block is to be over- written.

Eg:

Consider 4 blocks/set in set associative cache.

1. 2 bit counter can be used for each block.
2. When a **'hit'** occurs, then block counter=0; The counter with values originally lower than the referenced one are incremented by 1 & all others remain unchanged.

When a **'miss'** occurs & if the set is full, the blocks with the counter value 3 is removed, the new block is put in its place & its counter is set to "0" and other block counters are incremented by 1.

Module V

Processor Logic Design.

The operation of a digital system is described by specifying:

1. The set of registers in the system and their function.
2. The binary-coded information stored in the registers.
3. The operations performed on the information stored in the registers.
4. The control functions that initiate the sequence of operations.

The binary information stored in registers may be

1. binary numbers,
2. binary-coded decimal numbers
3. alphanumeric characters,
4. control information, or
5. any other binary-coded information.

The operations performed on the data stored in registers are called microoperations. A micro-operation is an elementary operation that can be performed in parallel during one clock pulse period.

Types of microoperations

1. Inter register transfer: microoperation do not change the information content when the binary information moves from one register to another.

Scanned with CamScanner

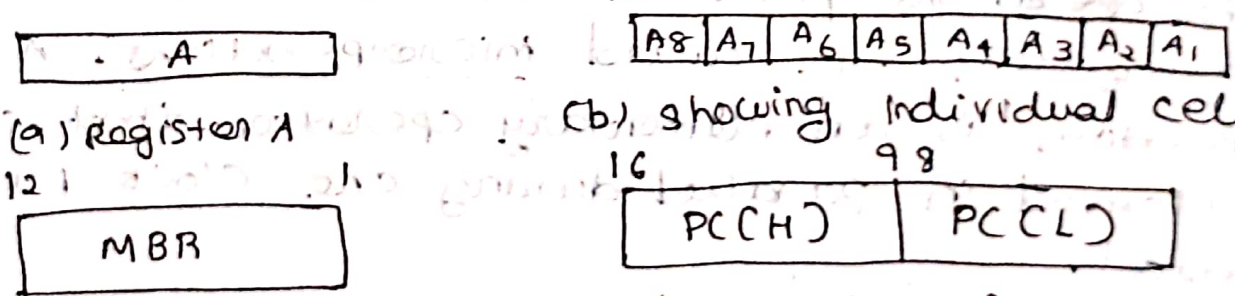
2. Arithmetic: microoperations perform arithmetic on numbers stored in registers.
3. Logic: microoperations perform operations such as AND and OR on individual pairs of bits stored in registers.
4. Shift: microoperations specify operations for shift registers.

The binary information commonly found in registers of digital computers can be classified into three categories:

1. Numerical data such as binary numbers or binary coded decimal numbers used in arithmetic computations.
2. Nonnumerical data such as alphanumeric characters or other binary-coded symbols used for special applications.
3. Instruction codes, addresses, and other control information used to specify the data-processing requirements in the system.

REGISTER-TRANSFER LOGIC

Registers can be represented in the following way



(a) Register A (b) showing individual cells

(c) Numbering of cells. (d) Portions of a register

Figure 5.1 Block diagram of registers.

The registers in a digital system are designated by capital letters to denote the function of the register.

Scanned with CamScanner

INTERREGISTER TRANSFER

Registers can be specified in a register-transfer language with a declaration statement. For example, the registers of Figure 5.1 can be defined with declaration statement such as:

DECLARE REGISTER A(8), MBR(12), PC(16)

DECLARE SUBREGISTER PC(L) = PC(1-8), PC(H) = PC(9-16)

← : replacement operator.

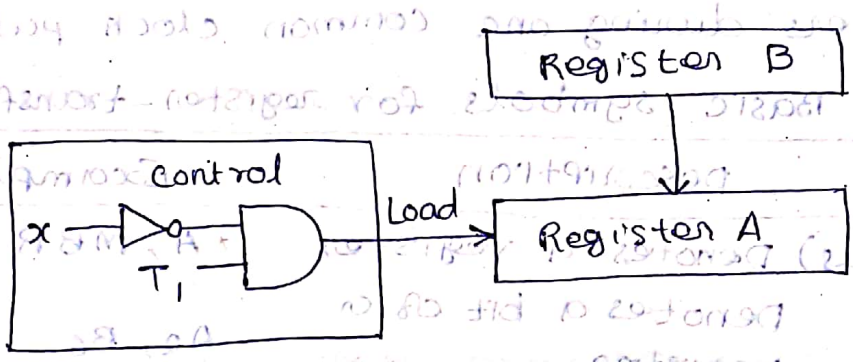


Figure 5.2 Hardware implementation of the statement

$x'T_1 : A \leftarrow B$

Information transfer from one register to another is designated in symbolic form by means of the replacement operator. The statement

$A \leftarrow B$

denotes the transfer of the contents of register B into register A.

The condition that determines when the transfer is to occur is called a control function. A control function is a Boolean function that can be equal to 1 or 0. The control function is included with the statement as follows: $x'T_1 : A \leftarrow B$.

Scanned with CamScanner

The control function is terminated with a colon.

1. Every statement written in a register-transfer language implies a hardware construction for implementing the transfer.
2. A colon terminates a control function, and the comma is used to separate two or more operations that are executed at the same time. The statement:

T_3: $A \leftarrow B$, $B \leftarrow A$

denotes an exchange operation that swaps the contents of two registers during one common clock pulse.

TABLE 5-1 Basic symbols for register-transfer logic

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	A, MBR, R2
Subscript	Denotes a bit of a register	A_2, B_6
Parantheses ( )	Denotes a portion of a register	PC(H), MBR(OP)
Arrow $\leftarrow$	Denotes transfer of information	$A \leftarrow B$
Colon :	Terminates a control function	$x'T_0$:
Comma ,	Separates two micro-operations	$A \leftarrow B$, $B \leftarrow A$
Square brackets [ ]	Specifies an address for memory transfer	$MBR \leftarrow M[MAR]$

Scanned with CamScanner

There are occasions when a destination register receives information from two sources, but evidently not at the same time.

Eg : T1 : C ← A.

T5 : C ← B

The first line states that the contents of register A are to be transferred to register C when timing variable Ti occurs.

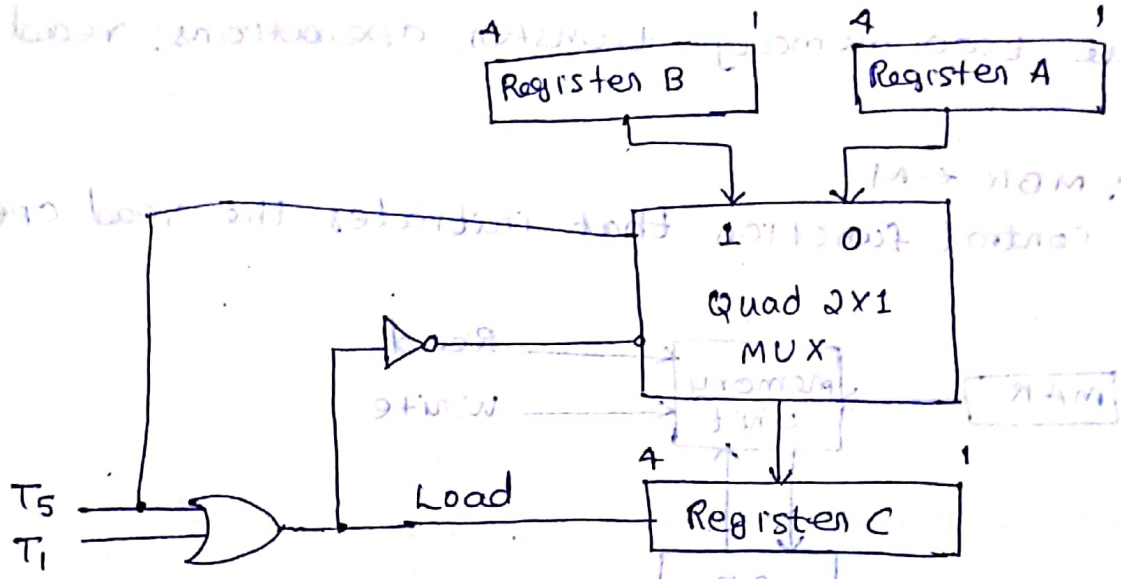


Figure 5.3 Use of a multiplexer to transfer information from two sources into a single destination.

1. Bus Transfer.

A group of wires through which binary information is transferred one at a time among registers is called a bus. For parallel transfer, the number of lines in the bus is equal to the number of bits in the registers.

A common-bus system can be constructed with multiplexers, and a destination register for the bus transfer can be selected by means of a decoder. The multiplexers select one source register for the bus, and the decoder selects one destination register to transfer the information from the bus.

Scanned with CamScanner

Memory Transfer

A memory register or word is symbolized by the letter M. If the letter M stands by itself in a statement, it will always designate a memory register selected by the address presently in MAR. Otherwise, the register that specifies the address (or the address itself) will be enclosed within square brackets after the symbol M.

There are two memory-transfer operations: read and write.

R : MBR ← M.

R is the control function that initiates the read operation

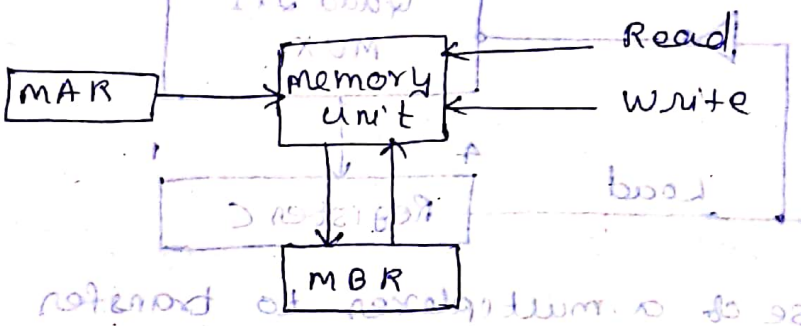


Figure 5.4 Memory unit that communicates with two external registers.

This causes a transfer of information into MBR from the selected memory register M specified by the address in MAR.

The write operation is a transfer from MBR to the selected memory register M. This is designated by the statement:

W : M ← MBR

W is the control function that initiates the write operation. This causes a transfer of information from MBR into the memory register M selected by the address

Scanned with CamScanner

presently in MAR.

The transfer of information from register B2 to a memory word selected by the address in register A1 is symbolized by the statement:

$$W: M[A1] \leftarrow B2$$

This is a write operation, with register A1 specifying the address. The square brackets after the letter M give the address register used for selecting the memory register M. The read operation in a memory with buses can be specified

$$R: B0 \leftarrow M[A3]$$

symbolizes a read operation from a memory register whose address is given by A3. The binary information coming out of memory is transferred to register B0.

ARITHMETIC, LOGIC, AND SHIFT MICROOPERATIONS

The interregister-transfer microoperations do not change the information content when the binary information moves from the source register to the destination register. All other microoperations change the information content during the transfer.

Arithmetic Microoperations.

The basic arithmetic microoperations are

1. add,
2. subtract,
3. complement and
4. shift.

add operation

$$F \leftarrow A + B$$

It states that the contents of register A are to be added to the contents of register B, and the sum transferred

Scanned with CamScanner

to register F.

To implement, require

1. 3 registers A, B & F
2. parallel adder.

Subtraction is most often implemented through complementation and addition, i.e

$$F \leftarrow A + \bar{B} + 1$$

Adding A to the 2's complement of B produces A minus B.

TABLE 5.2 Arithmetic microoperations

Symbolic designation	Description
$F \leftarrow A+B$	Contents of A plus B transferred to F
$F \leftarrow A-B$	Contents of A minus B transferred to F
$B \leftarrow \bar{B}$	complement register B (1's complement)
$B \leftarrow \bar{B}+1$	Form the 2's complement of the contents of register B
$F \leftarrow A+\bar{B}+1$	A plus the 2's complement of B transferred to F
$A \leftarrow A+1$	Increment the contents of A by 1 (count up)
$A \leftarrow A-1$	Decrement the contents of A by 1 (count down)

Consider the two statements:

$T_2 : A \leftarrow A+B$

$T_5 : A \leftarrow A+1$

Timing variable T_2 initiates an operation to add the contents of register B to the present contents of A. Timing variable T_5 ~~im~~ increments register A. The incrementing can be easily done with a counter, and the sum of two binary numbers can be generated with a parallel adder.

Scanned with CamScanner

The transfer of the sum from the parallel adder into register A can be activated with a load input in the register. This dictates that the register be a counter with parallel-load capability.

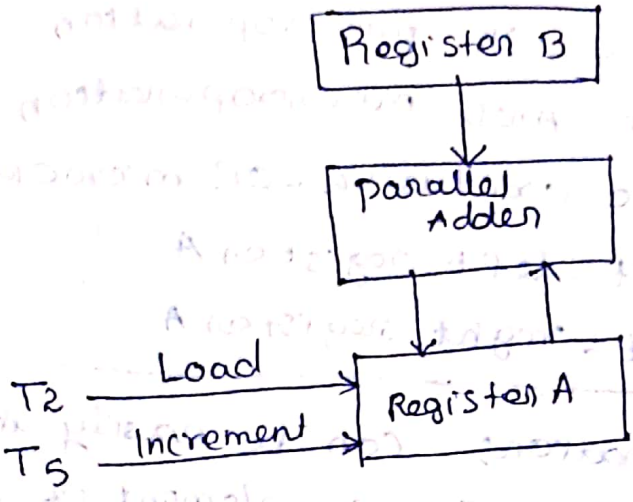


Figure 5.5 Implementation for the add and increment microoperations.

Logic Microoperations

Logic microoperations specify binary operations for a string of bits stored in registers. These operations consider each bit in the registers seperately and treat it as a binary variable.

Consider the statement

$F \leftarrow A \oplus B$

If the content of register A is 1010 and that of register B 1100, the information transferred to register F is 0110:

```
1010    content of A
1100    content of B
----
0110    content of F ← A ⊕ B.
```


Scanned with CamScanner

TABLE 5-3 Logic and Shift microoperations

Symbolic designation	Description
$A \leftarrow \bar{A}$	Complement all bits of register A
$F \leftarrow A \vee B$	Logic OR microoperation
$F \leftarrow A \wedge B$	Logic AND microoperation
$F \leftarrow A \oplus B$	Logic exclusive-OR microoperation
$A \leftarrow shl\ A$	Shift-left register A
$A \leftarrow shr\ A$	Shift-right register A

The logic microoperations can be easily implemented with a group of gates. The complement of a register of n bits is obtained from n inverter gates. The AND microoperation is obtained from a group of AND gates, each of which receives a pair of bits from the two source registers.

Consider the statement:

$T_1 + T_2 : A \leftarrow A + B,\ C \leftarrow D \vee F$

the + between T_1 and T_2 is an OR operation between two timing variables of a control function. The + between A and B specifies an add microoperation. The OR microoperation is designated by the symbol $\vee$ between registers D and F.

Shift Microoperations

1. Used in serial transfer between registers.
2. Registers can be shifted to the left or to the right.

Scanned with CamScanner

3. $A \leftarrow shl\ A$, $B \leftarrow shr\ B$ are two microoperations that specify a 1-bit shift to the left of register A and 1-bit shift to the right of register B.

4. While the bits of registers are shifted, the extreme flip-flops receive information from the serial input.

5. The extreme flip-flop is in the leftmost position of the register during a shift-right operation and in the rightmost position during a shift-left operation.

6. A shift microoperation statement must be accompanied with another microoperation that specifies the value of the serial input for the bit transfer into the extreme flip-flop.

eg: $A \leftarrow shl\ A$, $A_1 \leftarrow A_n$

is a circular shift that transfers the leftmost bit from A_n into the rightmost flip-flop A_1. Similarly:

$A \leftarrow shr\ A$, $A_n \leftarrow E$

is a shift-right operation with the leftmost flip-flop A_n receiving the value of the 1-bit register E.

$A \leftarrow shr\ A$, $A_n \leftarrow A_1$

is a circular shift that transfers the rightmost bit from A_1 into the leftmost flip-flop A_n.

Scanned with CamScanner

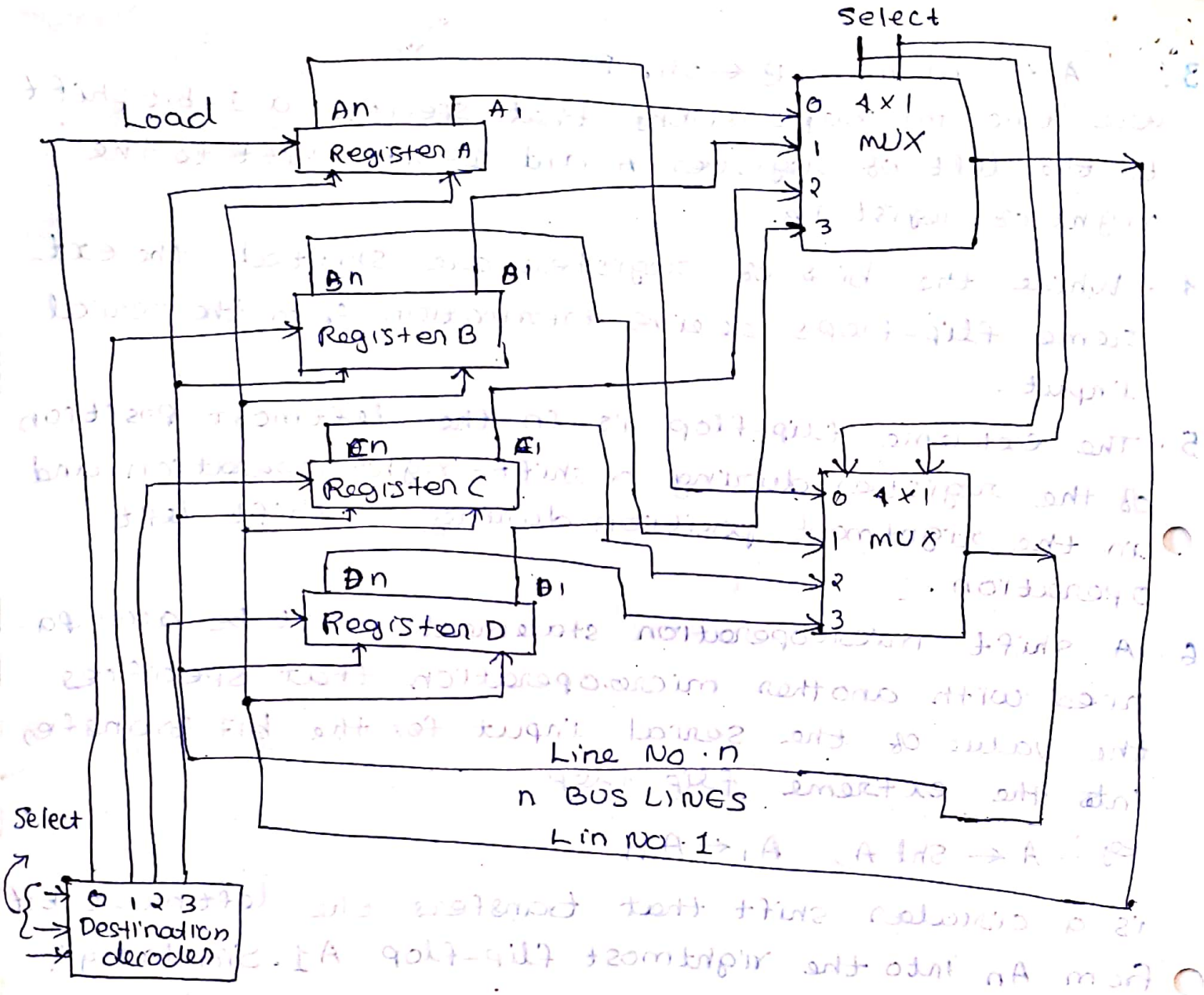


Figure 5.6 Bus system for four registers.

The construction of a bus system for four registers is depicted in Figure 5.6. The four bits in the same significant position in the registers go through a 4-to-1 line multiplexer to form one low-order significant bits and one for the high-order significant bits. For registers of n bits, n multiplexers are needed to produce an n-line bus.

Scanned with CamScanner

Bus Structure

1. Rather than connecting wires between all registers a common bus is used
2. A bus structure consists of a set of common lines one for each bit of a register.
3. Control signals determine which register is selected by the bus during each transfer.
4. Multiplexers can be used to construct a common bus
5. Multiplexers select the source register whose binary information is then placed on the bus.
6. The select lines are connected to the selection inputs of the multiplexers and choose the bits of one register.

In general

1. A bus system will multiplex k registers of n bits each to produce an n-line common bus.
2. This requires n multiplexers - one for each bit
3. The size of each multiplexer must be k x 1
4. The number of select lines required is log k
5. To transfer information from the bus to a register the bus lines are connected to the inputs of all destination registers and the corresponding load control line must be activated

CONDITIONAL CONTROL STATEMENTS

1. A control condition by a conditional statement rather than a Boolean control function.
2. A conditional control statement is symbolized by

Scanned with CamScanner

an if-then-else statement in the following manner:

P : If (condition) then [microoperation (s)] else [microoperation (s)]

3. Consider the conditional control statement:

T_2 : If (C=0) then (F ← 1) else (F ← 0)

If the time variable T_2 (control function) occured. then, F is assumed to be a 1-bit register (flip-flop) that can be set or cleared. If register C is a 1-bit register, the statement is equivalent to the following two statements:

$C' T_2$: F ← 1

$C T_2$: F ← 0

4. Note that the same timing variable can occur in two seperate control functions. The variable C can be either 0 or 1; therefore, only one of the microoperations will be executed during T_2, depending on the value of C.

5. When writing conditional control statements, one must realize that the condition stated after the word if is part of the control function and not part of a microoperation statement. The condition must be clearly stated and must be implementable with a combinational circuit.

PROCESSOR ORGANIZATION

Introduction.

1. A processor is part of a digital computer.
2. Processor is comprised of a number of registers and the digital functions that implement arithmetic,

Scanned with CamScanner

logic, shift and transfer microoperations.

3. The processor unit, when combined with a control unit that supervises the sequence of microoperations, is called a central processor unit or CPU.

4. The digital function that implements the microopera-tions on the information stored in processor registers is commonly called an arithmetic logic unit or ALU.

i) The ALU receives the information from the registers and performs a given operation as specified by the control. The result of the operation is then transferred to a destination register.

5. Accumulator is the general-purpose processor register.

6. Data path

The processor part of a computer CPU is some-times referred to as the data path of the CPU because the processor forms the paths for the data transfers between the registers in the unit.

7. Various paths are controlled by means of gates.

8. In a well-organized processor unit, the data paths are formed by means of buses and other com-mon lines.

9. Alternatives for organizing a general-purpose pro-cessor unit employ a common ALU and shifter, and different organization of the registers and their common path to the ALU.

Scanned with CamScanner

Bus Organization

A bus organization for four processor registers is shown in Figure 5.7

1. Each register is connected to two multiplexers (MUX) to form input buses A and B.
2. The selection lines of each multiplexer select one register for the particular bus.

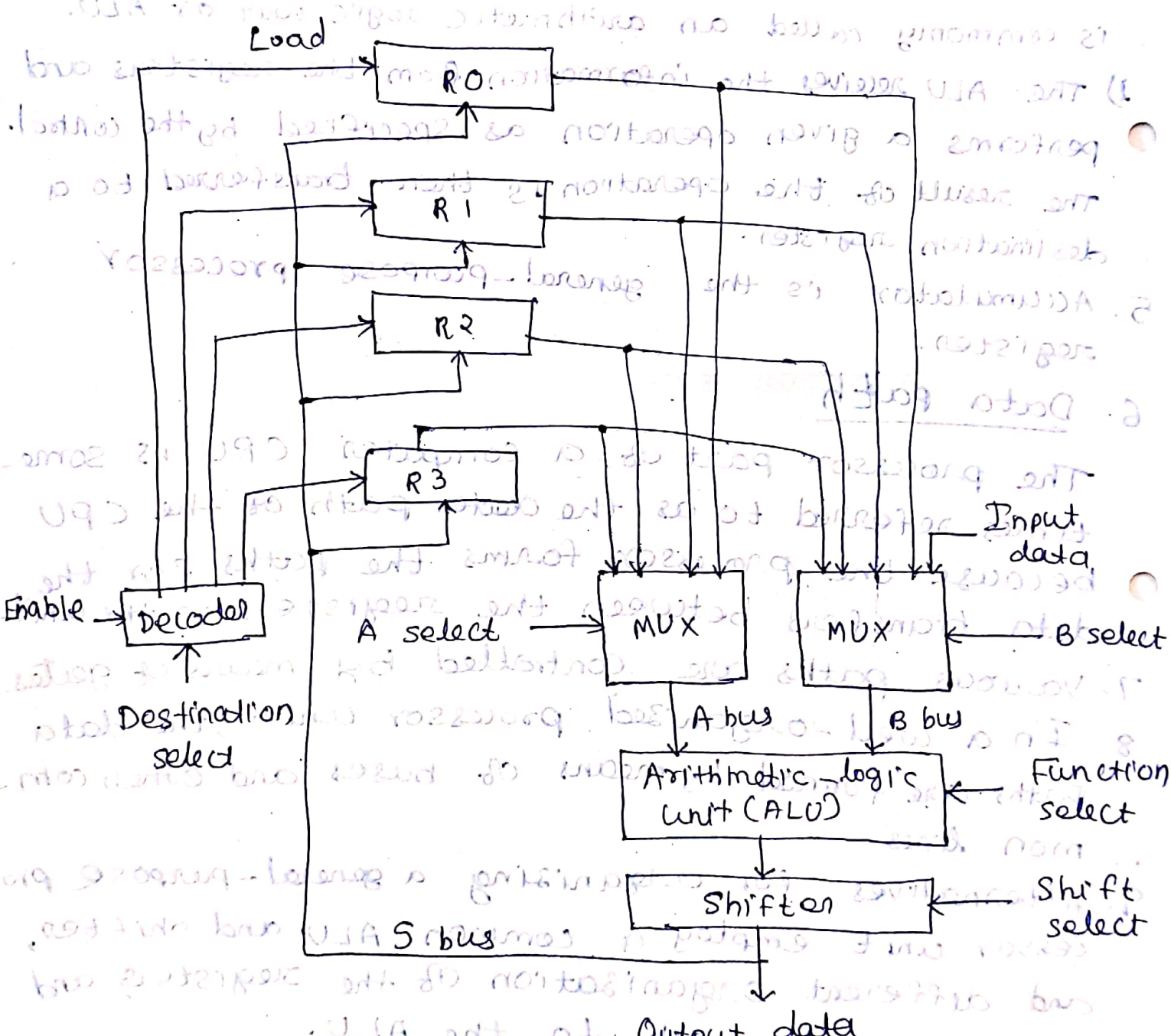


Figure 5.7 Processor registers and ALU connected through common buses.

Scanned with CamScanner

3. The A and B buses are applied to a common arithmetic logic unit.
4. The function selected in the ALU determines the particular operation that is to be performed.
5. The shift micro operations are implemented in the shifter.
6. The result of the microoperation goes through the output bus S into the inputs of all registers.
7. The destination register that receives the information from the output bus is selected by a decoder.
8. When enabled, this decoder activates one of the register load inputs to provide a transfer path between the data on the S bus and the inputs of the selected destination register.

The control unit that supervises the processor bus system directs the information flow through the ALU by selecting the various components in the unit.

Example:

To perform the microoperation:

$$R1 \leftarrow R2 + R3$$

The control must provide binary selection variables to the following selector inputs:

1. MUX A selector: to place the contents of R2 onto bus A.
2. MUX B selector: to place the contents of R3 onto bus B.
3. ALU function selector: to provide the arithmetic operation A + B.
4. Shift selector: for direct transfer from

Scanned with CamScanner

the output of the ALU onto output bus S. (no shift).

5. Decoder destination selector: to transfer the contents of bus S into RI.

The five control selection variables must be generated simultaneously and must be available during one common clock pulse interval.

When enclosed in an IC package, a processor unit is sometimes called a register and arithmetic logic unit or RALU. It is also called by some vendors a bit-slice microprocessor.

The prefix micro refers to the small physical size of the integrated circuit in which the processor is enclosed.

Bit-slice refers to the fact that the processor can be expanded to a processor unit with a longer number of bits by using a number of ICs.

Scratchpad Memory

The registers in a processor unit are enclosed within a small memory unit which is known as scratchpad memory.

1. The use of a small memory is a cheaper alternative to connecting processor registers through a bus system.
2. Connection buses inside the processor can be done in the following 2 ways.
   1. Bus system,
   2. Scratchpad memory.
3. In a bus system, the information transfer is selected by the multiplexers that form the buses.

Scanned with CamScanner

On the other hand, a single register in a group of registers organized as a small memory must be selected by means of an address to the memory unit.

4. The information stored in the scratchpad memory would normally come from the main memory by means of instructions in the program.

Example

Consider a processor unit that employs eight registers of 16 bits each. The registers can be enclosed within a small memory of eight words of 16 bits each, or an 8 x 16 RAM. The eight memory words can be designated R0 through R7, corresponding to address 0 through 7, and constitute the registers for the processor.

5. A processor unit that uses a scratchpad memory is shown in Figure 5.9.
   1. A source register is selected from memory and loaded into register A.
   2. A second source register is selected from memory and loaded into register B.
   3. The selection is done by specifying the corresponding word address and activating the memory read input.
   4. The information in A and B is manipulated in the ALU and shifter.
   5. The result of the operation is transferred to a memory register by specifying its word address and activating the memory-write input control.
   6. The multiplexer in the input of the memory can select input data from an external source.

Scanned with CamScanner

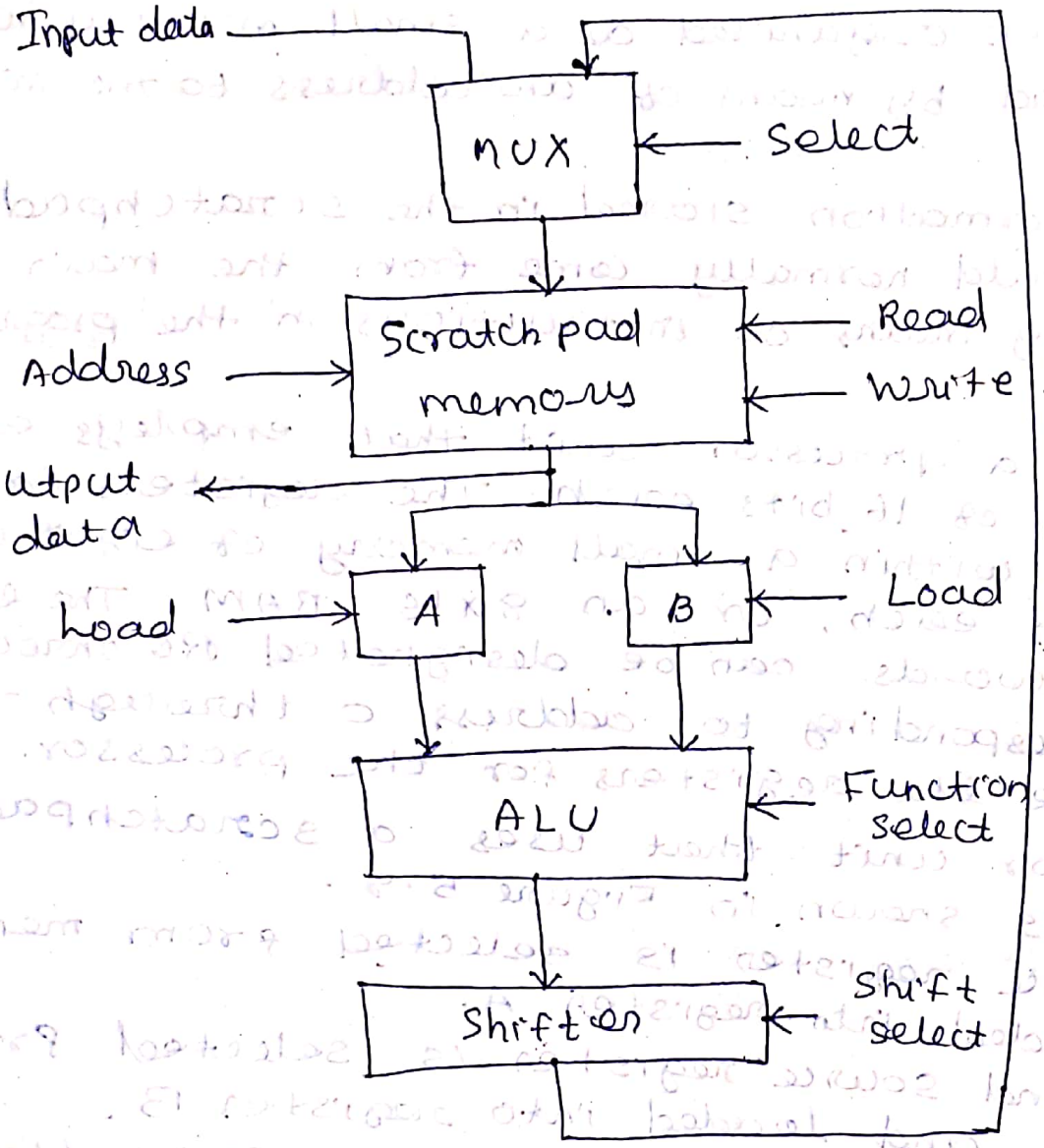


Figure 5.8 Processor unit employing a scratchpad memory.

Example:

Assume that the memory has eight words, so that an address must be specified with three bits. To perform the operation:

$$R1 \leftarrow R2 + R3$$

The control must provide binary selection variables to perform the following sequence of three micro-operations:

Scanned with CamScanner

T_1 : $A \leftarrow M[010]$ read R2 into register A

T_2 : $B \leftarrow M[011]$ read R3 into register B

T_3 : $M[001] \leftarrow A + B$ perform operation in ALU and transfer result to R1.

Control function T_1 must supply an address of 010 to the memory and activate the read and load A inputs. Control function T_2 must supply an address 011 to the memory and activate the read and load B inputs. control function T_3 must supply the function code to the ALU and shifter to perform an add operation (with no shift), apply an address 001 to the memory, select the output of the shifter for the MUX, and activate the memory-write input.

Limitation of memory unit

1. Since the memory unit has only one set of address terminals but two source registers are to be accessed, two accesses to memory are needed to read the source information.
2. The reason for a sequence of three microoperations instead of just one as in a bus-organised processor, is due to the limitation of the memory unit.
3. If the destination register is the same as the second source register, the control could activate the read input to extract the second-source information followed by a write signal to activate the destination transfer, without having to change the address value.

Scanned with CamScanner

2-port memory (solution to the limitation)

Some processors employ a 2-port memory in order to overcome the delay caused when reading two source registers.

1. Has two separate address lines to select two words of memory simultaneously.
2. Two registers (source) can be read at the same time.
3. If the destination register is the same as one of the source registers, then the entire micro-operation can be done within one clock pulse period.

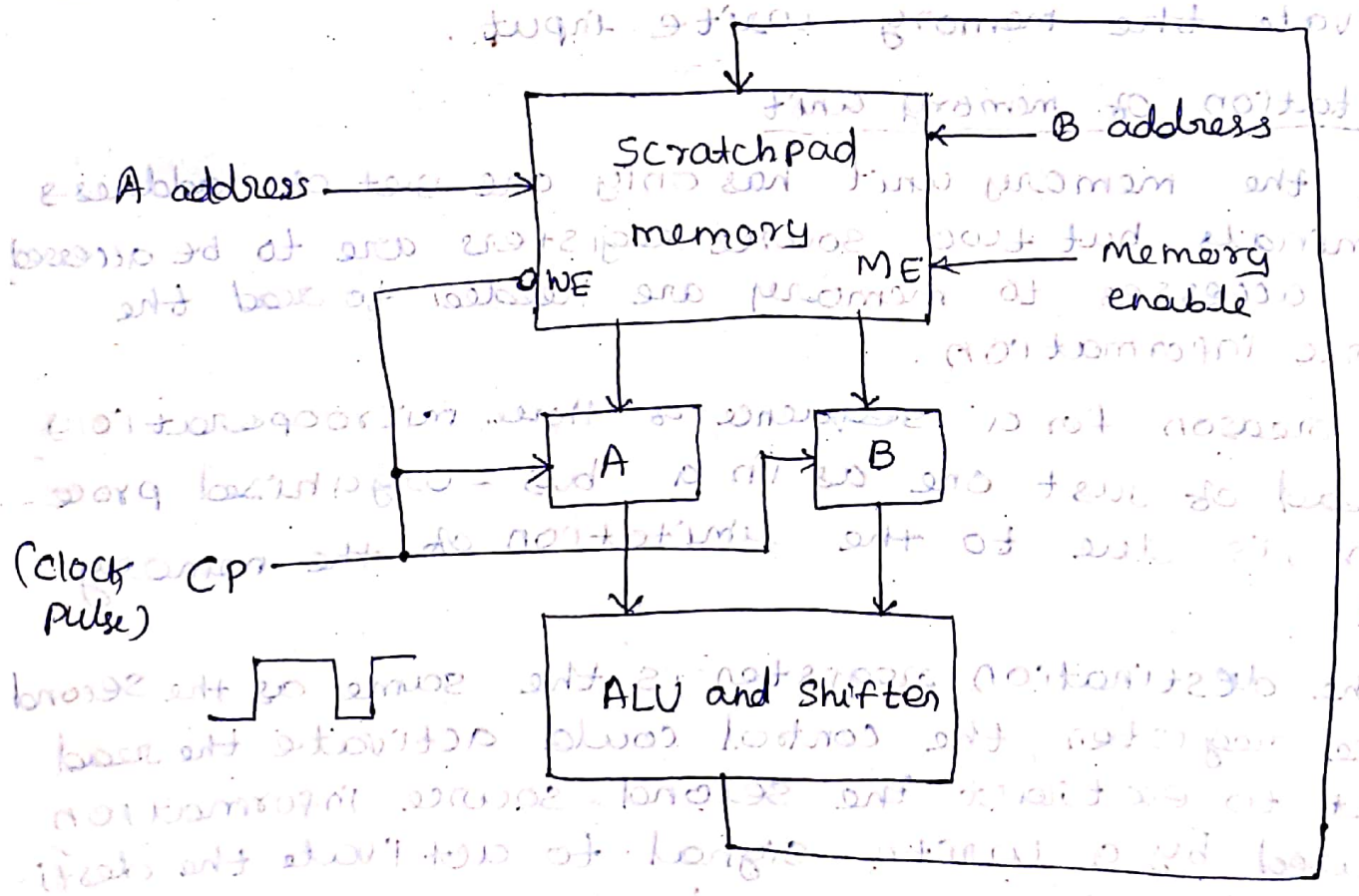


Figure 5-9 Processor unit with a 2-port memory.

Scanned with CamScanner

The organization of a processor unit with a 2-port scratchpad memory is shown in Figure 5.9.

1. The memory has two sets of addresses, one for port A and the other for port B.
2. Data from any word in memory are read into the register A by specifying an A address.
3. Data from any word in memory are read into the register B by specifying a B address.
4. A and B addresses specify two source registers simultaneously, and the B address always specifies the destination register.
5. When the clock pulse (CP) is in the 1-state, the A and B registers are, in effect, latches that accept new information
6. When CP goes to 0, the latches are disabled, and they hold the information that was stored when CP was a 1.
7. The clock input controls the memory read and write operations through the write enable (WE) input.
8. The clock also controls the transfer into the A and B latches.

Accumulator Register

1. Some processor units seperate one register from all others and call it an accumulator register, abbreviated AC or A register.
2. The name of this register is derived from the arithmetic addition process encountered in digital computers.

Scanned with CamScanner

3. The register "accumulates" the sum in a step-by-step manner by performing sequential additions between a new number and the previously accumulated sum.

4. The gates associated with an accumulator register provide all the digital functions found in an ALU.

5. Figure 5.10 shows the block diagram of a processor unit that employs an accumulator register.

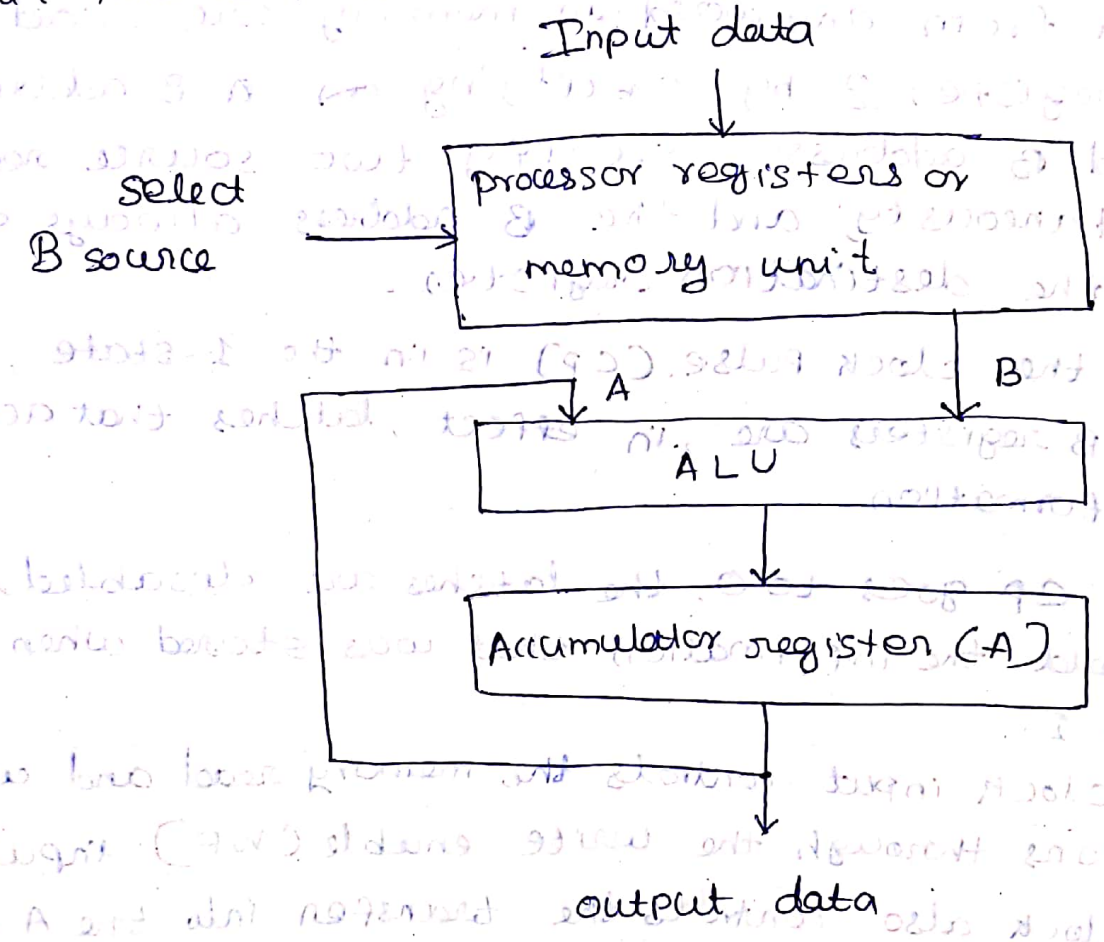


Figure 5.10 Processor with an accumulator register

6. The register itself can function as a shift register to provide the shift microoperations.

To form the sum of two numbers stored in processor registers, it is necessary to add them in the A register using the following sequence of microoperations:

Scanned with CamScanner

T_1 : $A \leftarrow 0$ clear A

T_2 : $A \leftarrow A + R1$ transfer R1 to A

T_3 : $A \leftarrow A + R2$ add R2 to A

Register A is first cleared. The first number in R1 is transferred into the A register by adding it to the present zero content of A. The second number in R2 is then added to the present value of A. The sum formed in A may be used for other computations or may be transferred to a required destination.

ARITHMETIC LOGIC UNIT

An arithmetic logic unit (ALU) is a multioperation, combinational-logic digital function. It can perform a set of basic arithmetic operations and a set of logic operations. The ALU has a number of selection lines to select a particular operation in the unit. The selection lines are decoded within the ALU so that k selection variables can specify up to 2^k distinct operations.

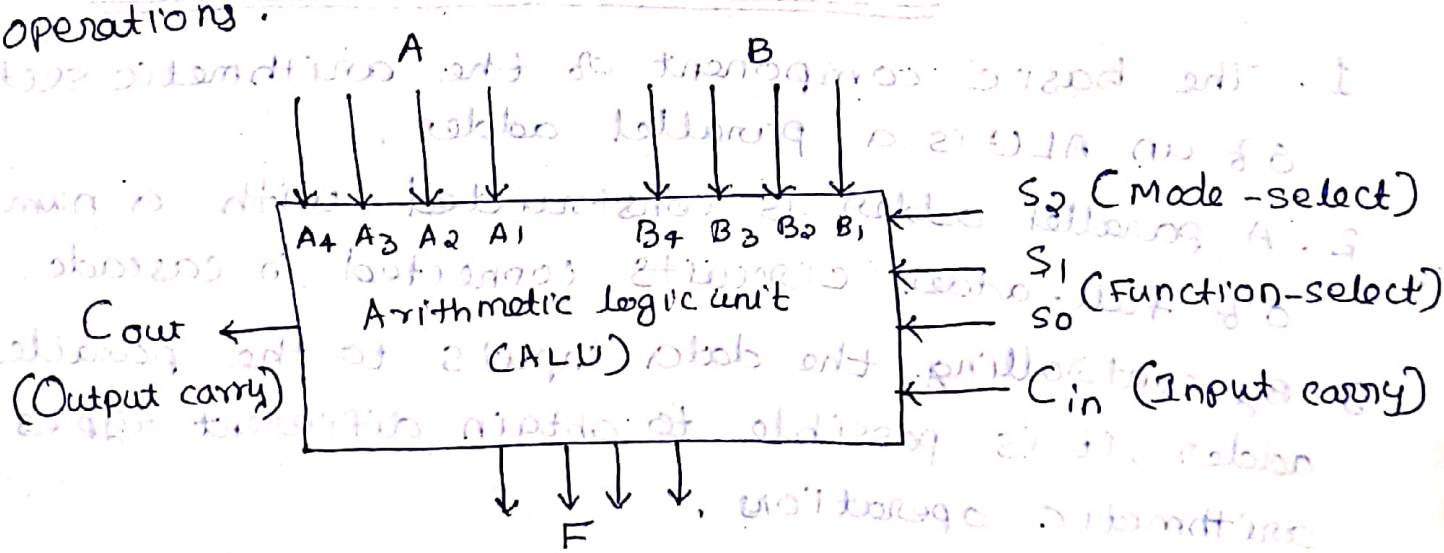


Figure 5.11 Block diagram of a 4-bit ALU

Scanned with CamScanner

1. The four data inputs from A are combined with the four inputs from B to generate an operation at the F outputs.
2. The mode-select input S2 distinguishes between arithmetic and logic operations.
3. With three selection variables, it is possible to specify four arithmetic operations (with S2 in one state) and four logic operations (with S2 in the other state).
4. The input and output carries have meaning only during an arithmetic operation.

The design of a typical ALU will be carried out in three stages:

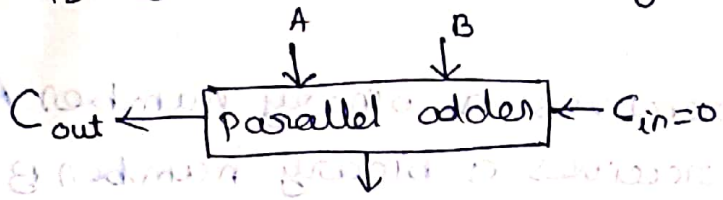
1. The design of the arithmetic section will be undertaken.
2. The design of the logic section will be considered.
3. The arithmetic section will be modified so that it can perform both arithmetic and logic operations.

DESIGN OF ARITHMETIC CIRCUIT

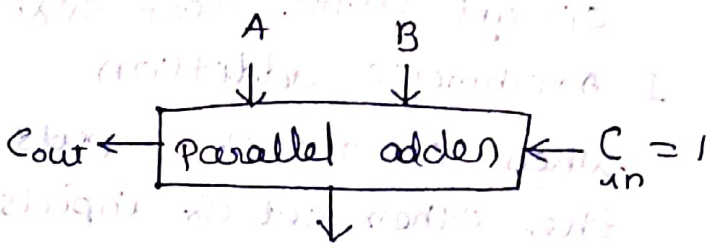
1. The basic component of the arithmetic section of an ALU is a parallel adder.
2. A parallel adder is constructed with a number of full-adder circuits connected in cascade.
3. By controlling the data inputs to the parallel adder, it is possible to obtain different types of arithmetic operations.
4. Figure 5.12 demonstrates the arithmetic operations

Scanned with CamScanner

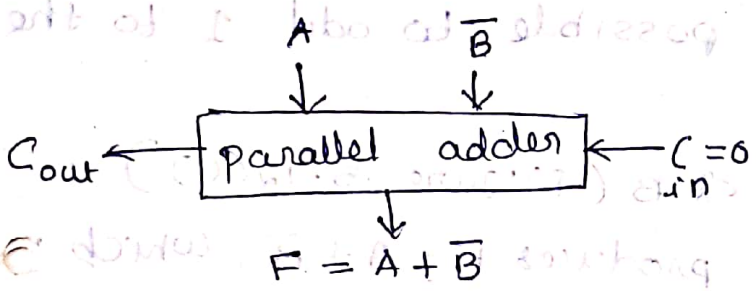
obtained when one set of inputs to a parallel adder is controlled externally.



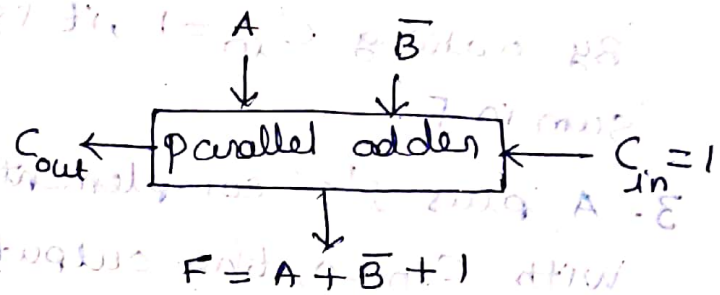
(a) addition



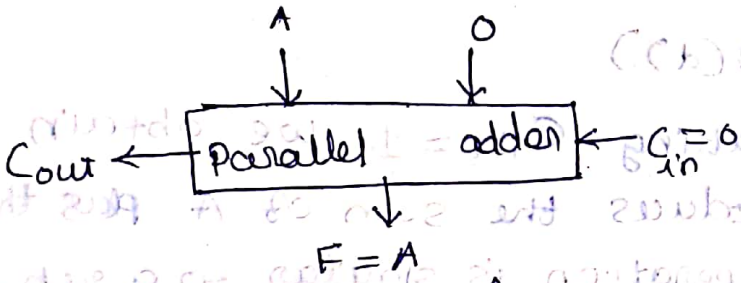
(b) Addition with carry



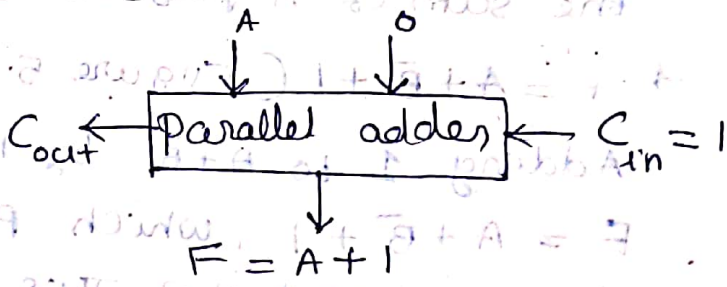
(c) A plus 1's complement of B



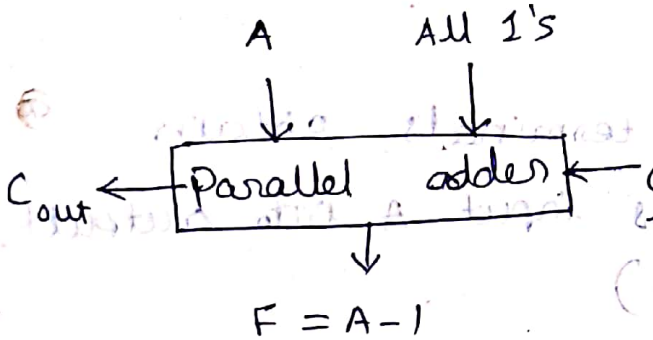
(d) Subtraction



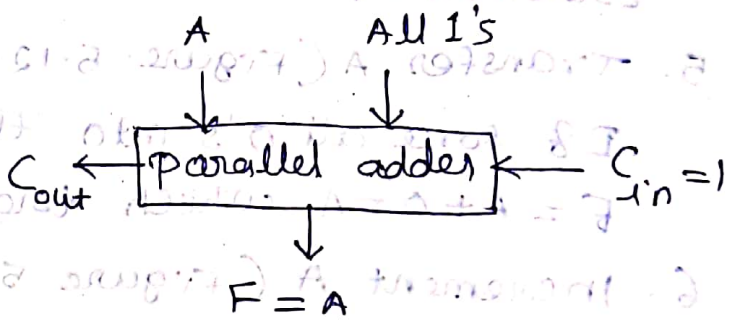
(e) Transfer A



(f) Increment A



(g) Decrement A



(h) Transfer A

Figure 5.12 Operations obtained by controlling one set of inputs to a parallel adder.

Scanned with CamScanner

a) The input carry Cout comes from the full-adder circuit in the most significant bit position.

1. Arithmetic addition
When one set of inputs receives a binary number A, the other set of inputs receives a binary number B, and the input carry maintained at 0. Figure 5.12(a)

2. Addition with carry. (Figure 5.12(b))
By making $C_{in} = 1$, it is possible to add 1 to the sum in F.

3. A plus 1's complement of B (Figure 5.12(c))
With $C_{in} = 0$, the output produces $F = A + \bar{B}$, which is the sum of A plus the 1's complement of B.

4. $F = A + \bar{B} + 1$ (Figure 5.12(d))
Adding 1 to $A + \bar{B}$ by making $C_{in} = 1$, we obtain $F = A + \bar{B} + 1$, which produces the sum of A plus the 2's complement of B. This operation is similar to a subtraction operation if the output carry is discarded.

5. Transfer A (Figure 5.12(e))
If force all 0's into the B terminals, obtain $F = A + 0 = A$, which transfers input A into output F.

6. Increment A (Figure 5.12(f))
Adding 1 through C_{in}, we obtain $F = A + 1$, which is the increment operation.

7. Decrement A (Figure 5.12(g))
Inserts all 1's into the B terminals. This produces the decrement operation $F = A - 1$.

Scanned with CamScanner

$A = 0000\ 1001 = (9)_{16}$ let $n=8$, $A=9$

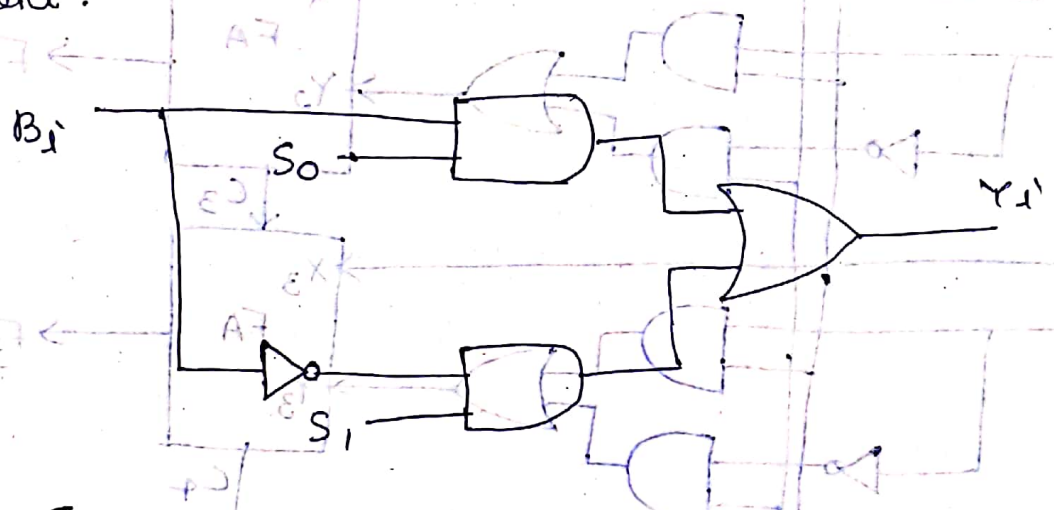
$2^n = 1\ 0000\ 0000 = (256)_{10}$

$2^n - 1 = 1111\ 1111 = (255)_{10}$

$A + 2^n - 1 = 1\ 0000\ 1000 = (256+8)_{10}$

Removing the output carry $2^n = 256$, we obtain $8 = 9 - 1$. Thus, we have decremented A by 1, by adding to it a binary number with all 1's.

The circuit that controls input B to provide the functions in figure 5.12 is called a true/complement, one/zero element.



S_1	S_0	Y_i
0	0	0
0	1	B_i
1	0	B_i'
1	1	1

Figure 5.13 True/complement, one/zero circuit.

A 4-bit arithmetic circuit that performs eight arithmetic operations is shown in Figure 5.14

1. The four full-adder (FA) circuits constitute the parallel adder.
2. The carry into the first stage is the input carry.
3. The carry out of the fourth stage is the output carry.
4. All other carries are connected internally from one stage to the next.

Scanned with CamScanner

5. The selection variables are S_1, S_0, and C_{in}.
6. Variables S_1 and S_0 control all of the B inputs to the full-adder circuits.
7. The A inputs go directly to the other inputs of the full adders.

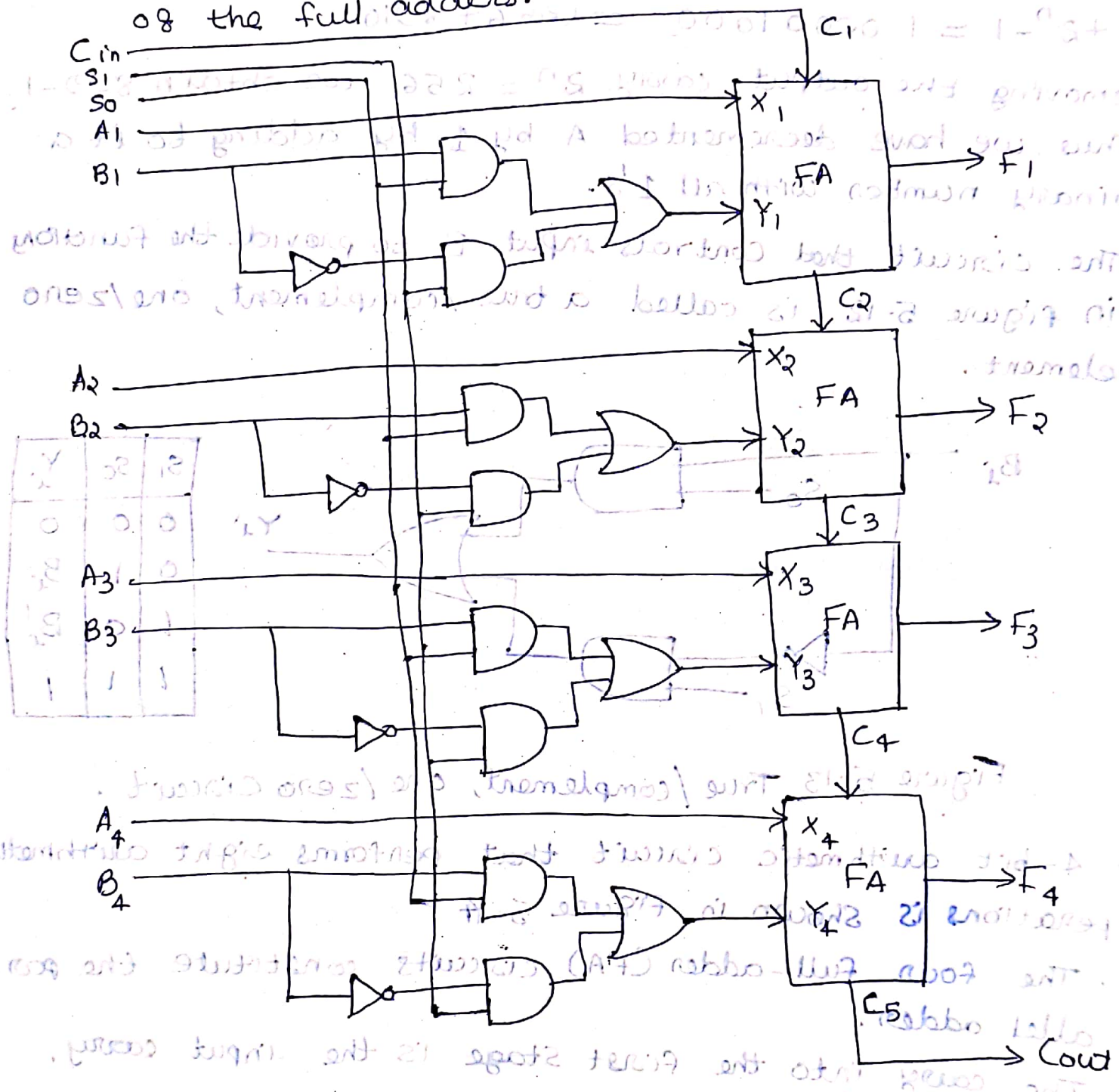


Figure 5.14 Logic diagram of arithmetic circuit

Scanned with CamScanner

TABLE 5-1 Function table for the arithmetic circuit of Figure 5.17

Function select			Y equals	Output equals	Function
S_1	S_0	C_{in}			
0	0	0	0	$F = A$	Transfer A
0	0	1	0	$F = A + 1$	Increment A
0	1	0	B	$F = A + B$	Add B to A
0	1	1	B	$F = A + B + 1$	Add B to A plus 1
1	0	0	$\overline{B}$	$F = A + \overline{B}$	Add 1's complement of B to A
1	0	1	$\overline{B}$	$F = A + \overline{B} + 1$	Add 2's complement of B to A
1	1	0	All 1's	$F = A - 1$	Decrement A
1	1	1	All 1's	$F = A$	Transfer A

The arithmetic circuit of Figure 5.14 needs a combinational circuit in each stage specified by the Boolean functions:

$$X_i = A_i$$

$$Y_i = B_i S_0 + B_i' S_1 \qquad i = 1, 2, \cdots, n$$

where n is the number of bits in the arithmetic circuit. In each stage i, we use the same common selection variables S_1 and S_0.

Scanned with CamScanner

Effect of Output Carry

The output carry of an arithmetic circuit or ALU has special significance, especially after a subtraction operation.

To investigate the effect of the output carry, we expand the arithmetic circuit of Figure 5.4 to n bits so that Cout = 1 when the output of the circuit is equal to or greater than 2^n.

Table 5.2 Effect of output carry in the arithmetic circuit of Figure 5.14.

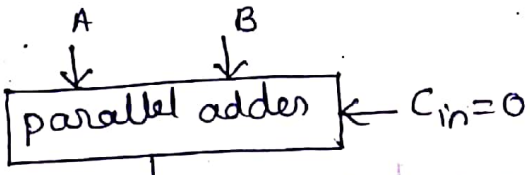
Function select			Arithmetic function	Cout = 1 if	Comments
S_1	S_0	C_{in}			
0	0	0	$F = A$		Cout is always 0
0	0	1	$F = A+1$	$A = 2^n - 1$	Cout = 1 and F = 0 if $A = 2^n - 1$
0	1	0	$F = A+B$	$(A+B) \geq 2^n$	overflow occurs if $C_{out} = 1$
0	1	1	$F = A+B+1$	$(A+B) \geq (2^n - 1)$	Overflow occurs if $C_{out} = 1$
1	0	0	$F = A-B-1$	$A > B$	If Cout = 0, then $A \leq B$ and F = 1's complement of (B − A)
1	0	1	$F = A-B$	$A \geq B$	If Cout = 0, then A < B and F = 2's complement of (B − A)
1	1	0	$F = A-1$	$A \neq 0$	Cout = 1, except when A = 0
1	1	1	$F = A$		Cout is always 1.

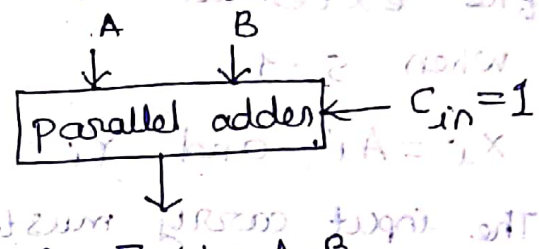
Scanned with CamScanner

Example 5-1:

Design an adder / subtractor circuit with one selection variable S and two inputs A and B. When S = 0 the circuit performs A + B. When S = 1 the circuit performs A − B by taking the 2's complement of B.

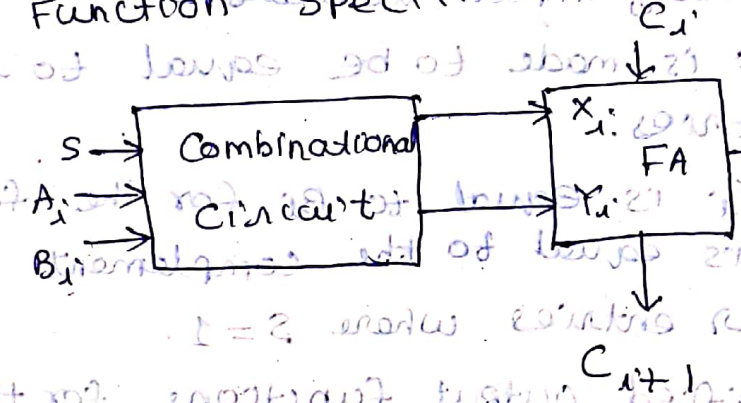
Derivation of the arithmetic circuit:





(a) Function specification

S	x_i	Y_i	C_{in}
0	A_i	B_i	0
1	A_i	B_i'	1



(b) Specifying combinational circuit

S	A_i	B_i	x_i	Y_i
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	0
1	1	0	1	1
1	1	1	1	0

$x_i = A_i$

$Y_i = B_i \oplus S$

$C_{in} = S$

(c) Truth table and simplified equations

Figure 5-15 Derivation of an adder/subtractor circuit

Scanned with CamScanner

i) For the addition part, we need $C_{in} = 0$.

ii) For the subtraction part, we need the complement of B and $C_{in} = 1$.

iii) When $S = 0$,

X_i and Y_i of each full adder must be equal to the external inputs A_i and B_i, respectively.

iv) When $S = 1$,

$X_i = A_i'$ and $Y_i = B_i'$

v) The input carry must be equal to the value of S.

vi) The truth table is obtained by listing the eight values of the binary input variables.

a) Output X_i is made to be equal to input A_i in all eight entries.

b) Output Y_i is equal to B_i for the four entries when $S = 0$. It is equal to the complement of B_i for the last four entries where $S = 1$.

c) The simplified output functions for the combinational circuit are:

$$X_i = A_i$$

$$Y_i = B_i \oplus S$$

vii) The diagram of the 4-bit adder/subtractor circuit is shown in Figure 5.16.

a) Each input B_i requires an exclusive-OR gate.

b) The selection variable s goes to one input of each gate and also to the input carry of the parallel adder.

c) The 4-bit adder/subtractor can be constructed

Scanned with CamScanner

with two ICs. One IC is the 4-bit parallel adder and the other is a quadruple exclusive-OR gates.

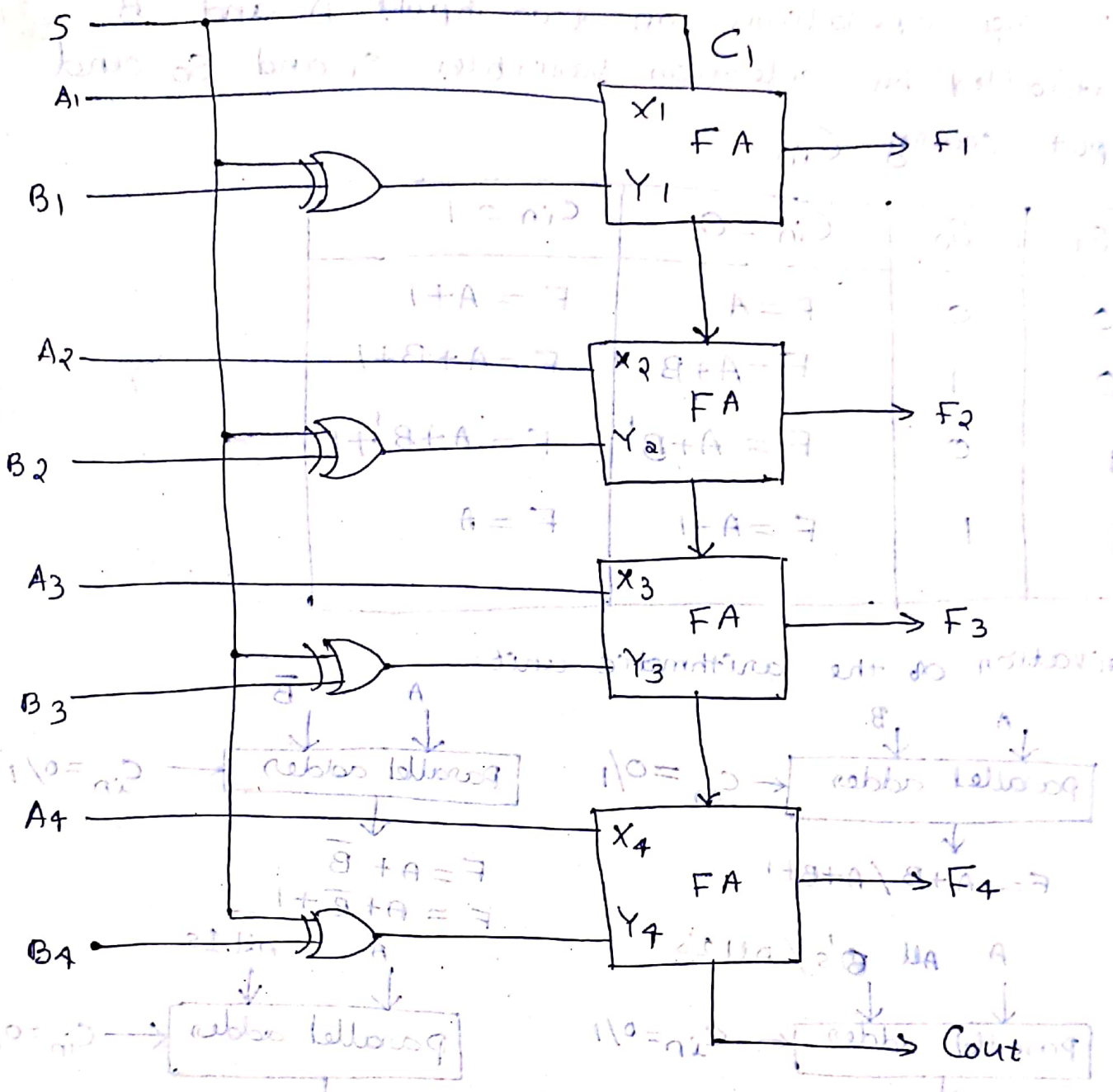


Figure 5.16 4-bit adder/subtractor circuit.

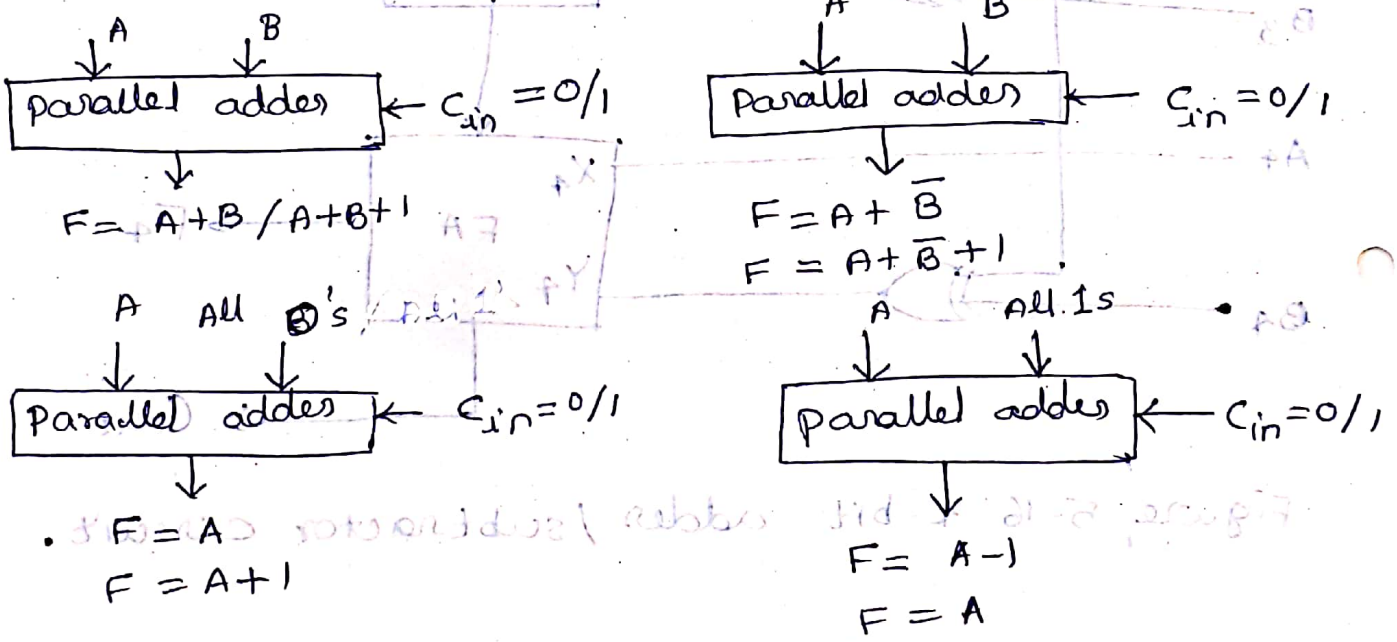
Scanned with CamScanner

Question

Design a 4-bit Arithmetic unit which performs the following operations on two inputs A and B, controlled by selection variables S_1 and S_0 and input carry C_{in}:

S_1	S_0	$C_{in} = 0$	$C_{in} = 1$
0	0	$F = A$	$F = A + 1$
0	1	$F = A + B$	$F = A + B + 1$
1	0	$F = A + B'$	$F = A + B' + 1$
1	1	$F = A - 1$	$F = A$

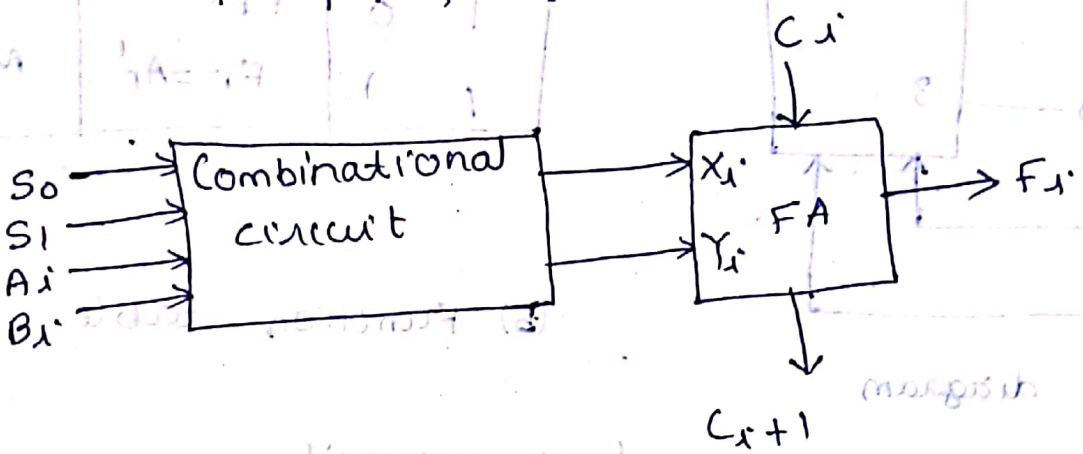
Derivation of the arithmetic unit



(a) Function Specification

Scanned with CamScanner

S_1	S_0	X_i	Y_i	C_{in}
0	0	A_i	0	0
0	1	A_i	B_i	0
1	0	A_i	$\overline{B_i}$	0
1	1	A_i	1	0
0	0	A_i	0	1
0	1	A_i	B_i	1
1	0	A_i	$\overline{B_i}$	1
1	1	A_i	1	1



b) specifying combinational circuit

$X_i = A_i$

$Y_i = B_i S_0 + B_i' S_1 \quad i = 1, 2, 3, 4$

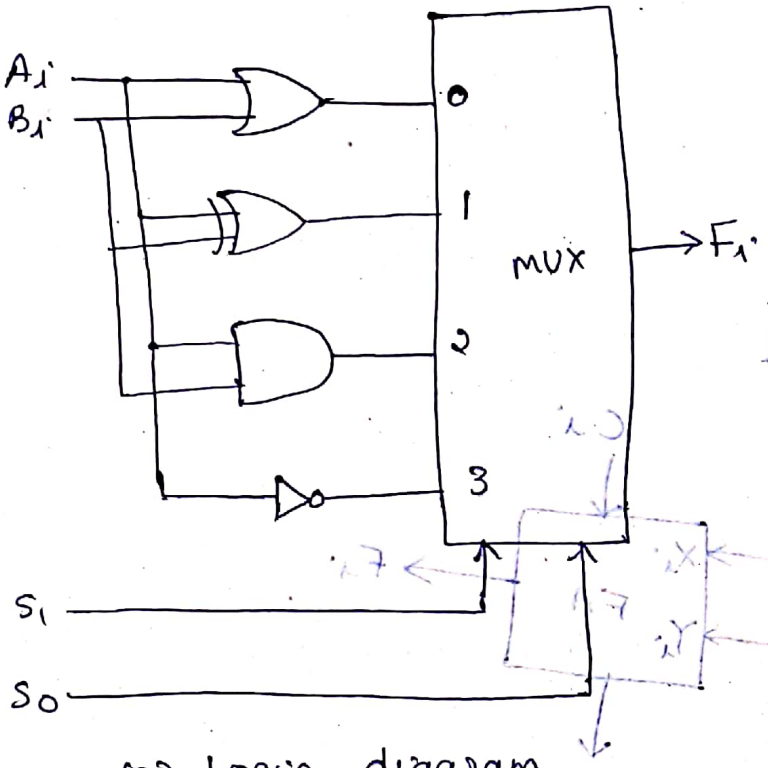
c) simplified equation

Figure 5.14 shows the 4-bit arithmetic unit for the given functions.

Scanned with CamScanner

DESIGN OF LOGIC CIRCUIT

The logic microoperations manipulate the bits of the operands seperately and treat each bit as a binary variable.



(a) Logic diagram

S_1	S_0	Output	Operation
0	0	$F_i = A_i + B_i$	OR
0	1	$F_i = A_i \oplus B_i$	XOR
1	0	$F_i = A_i B_i$	AND
1	1	$F_i = A_i'$	NOT

(b) Function table

Figure 5.17 One stage of logic circuit.

The four gates generate the four logic operations OR, XOR, AND and NOT. The two selection variables in the multiplexer select one of the gates for the output.

The logic circuit can be combined with the arithmetic logic unit. selection variables S_1 and S_0 can be made common to both sections, provided we use a third selection variable, S_2, to differentiate between the two.

Scanned with CamScanner

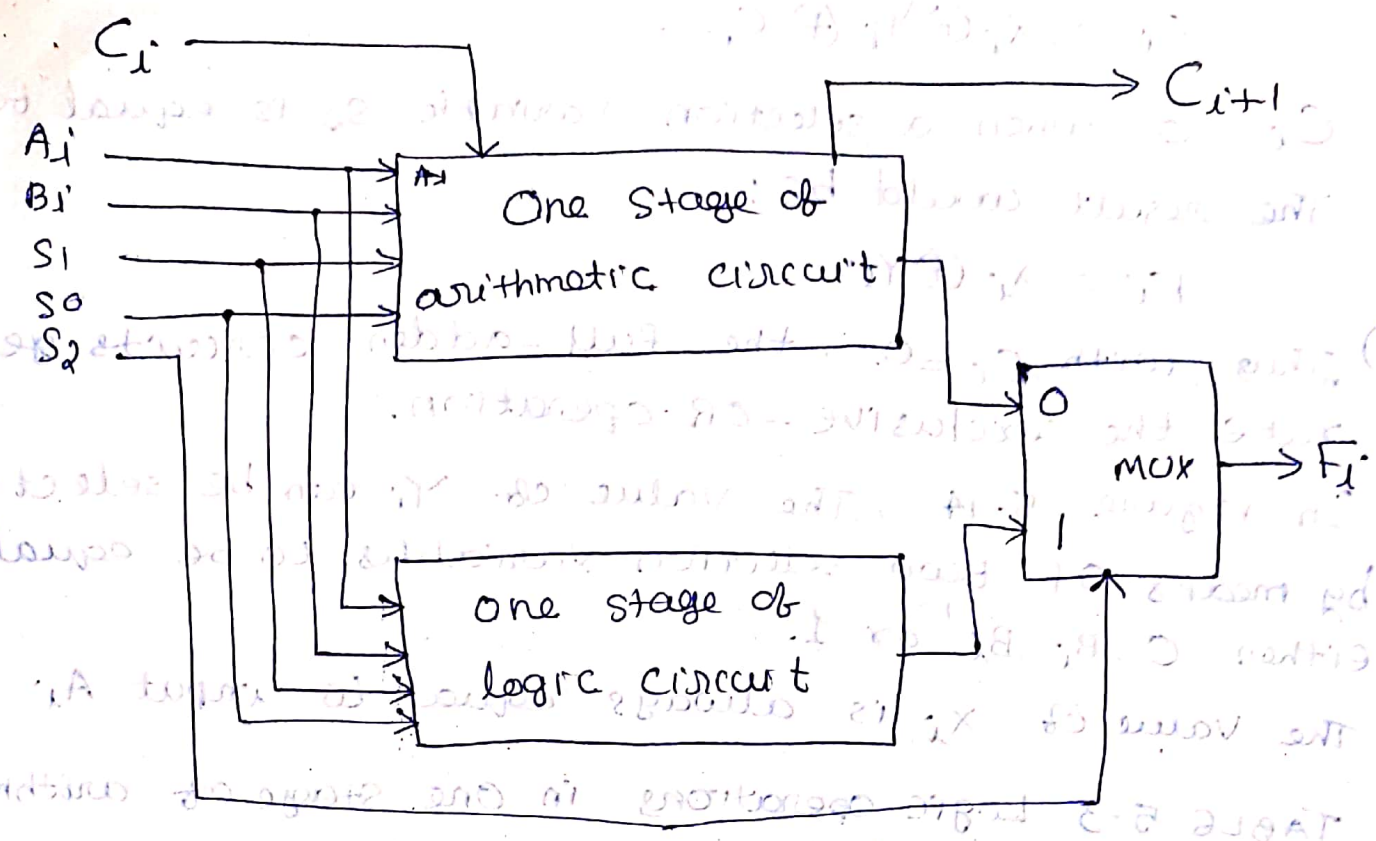


Figure 5.18 Combining logic and arithmetic circuits.

DESIGN OF ARITHMETIC LOGIC UNIT

The steps involved in the design of an ALU are as follows:

1. Design the arithmetic section independent of the logic section.

   Figure 5.14 shows the arithmetic section

2. Determine the logic operations obtained from the arithmetic circuit in step 1, assuming that the input carries to all stages are 0.

i) A more efficient ALU can be obtained if we investigate the possibility of generating logic operations in an already available arithmetic circuit.

ii) Consider the Boolean function that generates the output sum in a full-adder circuit.

Scanned with CamScanner

$$F_i = X_i \oplus Y_i \oplus C_i$$

$C_i = 0$ when a selection variable S_2 is equal to 1.

The result would be:

$$F_i = X_i \oplus Y_i$$

iii) Thus, with $C_i = 0$, the full-adder circuits generate the exclusive-OR operation.

iv) In Figure 5-14, The value of Y_i can be selected by means of two selection variables to be equal to either 0, B_i, B_i' or 1.

The value of X_i is always equal to input A_i

TABLE 5-3 Logic operations in one stage of arithmetic circuit

S_2	S_1	S_0	X_i	Y_i	C_i	$F_i = X_i \oplus Y_i$	Operation	Required operation
1	0	0	A_i	0	0	$F_i = A_i$	Transfer A	OR
1	0	1	A_i	B_i	0	$F_i = A_i \oplus B_i$	XOR	XOR
1	1	0	A_i	B_i'	0	$F_i = A_i \odot B_i$	Equivalence	AND
1	1	1	A_i	1	0	$F_i = A_i'$	NOT	NOT

$A_i \oplus B_i' = A_i B_i + A_i' B_i' = A_i \odot B_i$ ⇒ Third entry.

$A_i \oplus 1 = A_i'$ ⇒ last entry.

3. Modify the arithmetic circuit to obtain the required logic operations.

i) From Table 5.3

Scanned with CamScanner

When $S_2 = 1$

$C_i = 0$ in each stage,

$S_1 S_0 = 00$, each stage generates the function

$F_i = A_i$

ii) To change the output ~~not~~ to an OR operation, change the input to each full-adder circuit from A_i to $A_i + B_i$ and $Y_i = 0$.

iii) The other selection variable that give an undesirable output occur when $S_2 S_1 S_0 = 110$.

Consider $X_i = (A_i + B_i')$

so $X_i \oplus Y_i = (A_i + B_i') \oplus B_i'$

$= A_i B_i + B_i' B_i + A_i B_i B_i'$

$= \underline{\underline{A_i B_i}}$

So when $A_i = A_i + B_i'$ and $B_i = B_i'$, $F_i = A_i B_i$

The conclusion is that, if A_i is ORed with B_i' when $S_2 S_1 S_0 = 110$, the output will generate the AND operation.

iv) $X_i = A_i + S_2 S_1' S_0' B_i + S_2 S_1 S_0' B_i'$

$Y_i = S_0 B_i + S_1 B_i'$

$Z_i = S_2' C_i$

} Boolean functions for the inputs to each full-adder circuit

v) when $S_2 = 0$, Arithmetic circuit Boolean function.

$X_i = A_i$

$Y_i = S_0 B_i + S_1 B_i'$

$Z_i = S_2' C_i$

Scanned with CamScanner

vi) when $S_2 = 1$ - logic operations Boolean functions

$X_i = A_i$

$Y_i = S_0 B_i + S_1 B_i'$

$C_i = 0$.

Table 5-4 Function table for the ALU of Figure 5.19.

selection				Output	Function
S_2	S_1	S_0	C_{in}		
0	0	0	0	$F = A$	Transfer A
0	0	0	1	$F = A + 1$	Increment A
0	0	1	0	$F = A + B$	Addition
0	0	1	1	$F = A + B + 1$	Add with carry
0	1	0	0	$F = A - B - 1$	Subtract with borrow
0	1	0	1	$F = A - B$	Subtraction
0	1	1	0	$F = A - 1$	Decrement A
0	1	1	1	$F = A$	Transfer A
1	0	0	X	$F = A \vee B$	OR
1	0	1	X	$F = A \oplus B$	XOR
1	1	0	X	$F = A \wedge B$	AND
1	1	1	X	$F = \bar{A}$	Complement A

The 12 operations generated in the ALU are summarized in Table 5-4. The particular function is selected through S_2, S_1, S_0 and C_{in}. The value of C_{in} for the four logic functions has no effect on the operation

Scanned with CamScanner

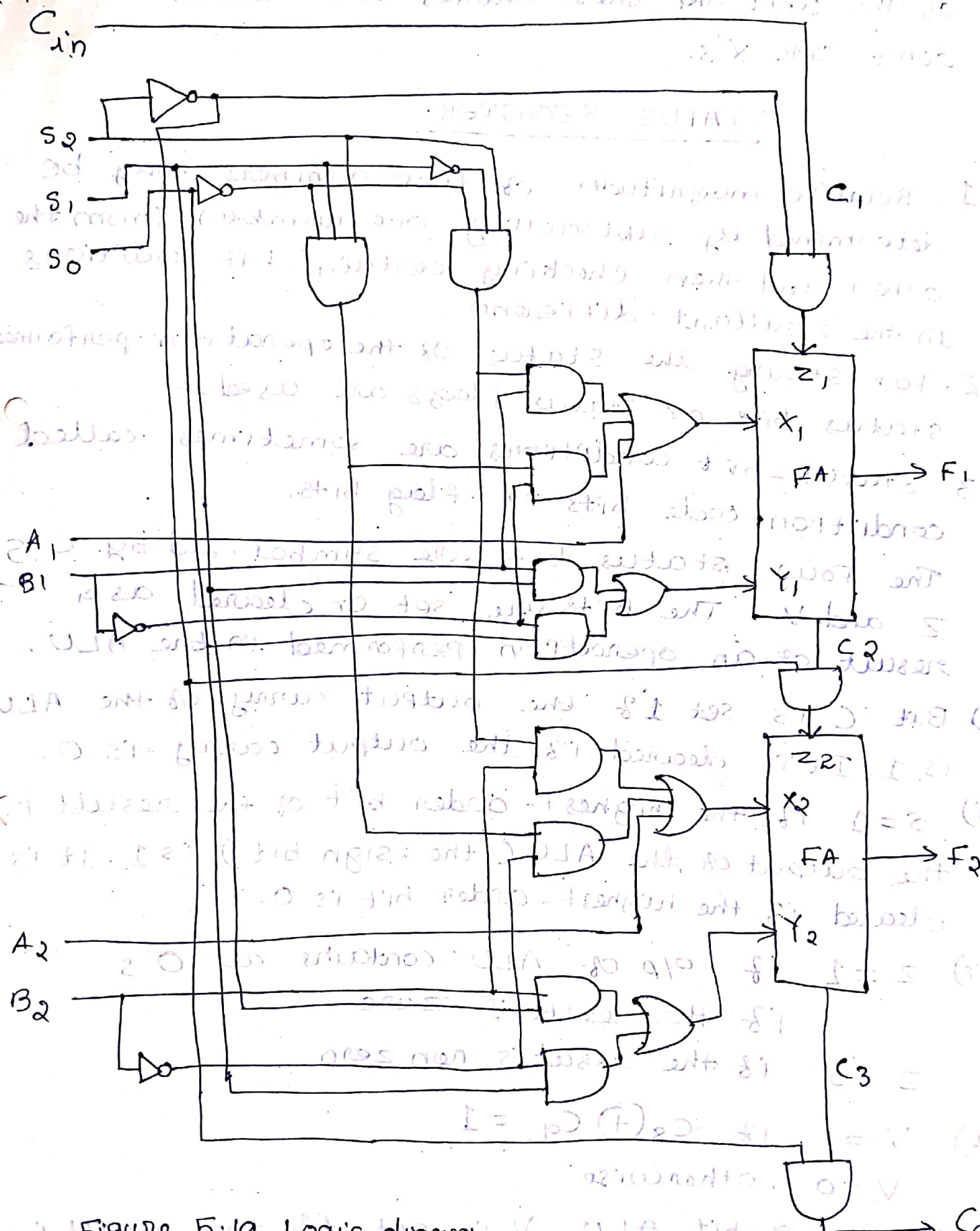


Figure 5.19 Logic diagram of arithmetic logic unit (ALU)

Scanned with CamScanner

of the unit and those entries are marked with don't-care X's.

STATUS REGISTER

1. Relative magnitudes of two numbers may be determined by subtracting one number from the other and then checking certain bit conditions in the resultant difference.
2. For storing the status of the operation performed status bits or status flags are used.
3. Status-bit conditions are sometimes called condition-code bits or flag bits.

The four status bits are symbolized by C, S, Z and V. The bits are set or cleared as a result of an operation performed in the ALU.

i) Bit C is set if the output carry of the ALU is 1. It is cleared if the output carry is 0.

ii) S = 1 if the highest-order bit of the result i.e. the output of the ALU (the sign bit) is 1. It is cleared if the highest-order bit is 0.

iii) Z = 1 if o/p of ALU contains all 0's
if the result is zero

Z = 0 if the result is non zero.

iv) V = 1, if $C_8 \oplus C_9 = 1$

V = 0, otherwise

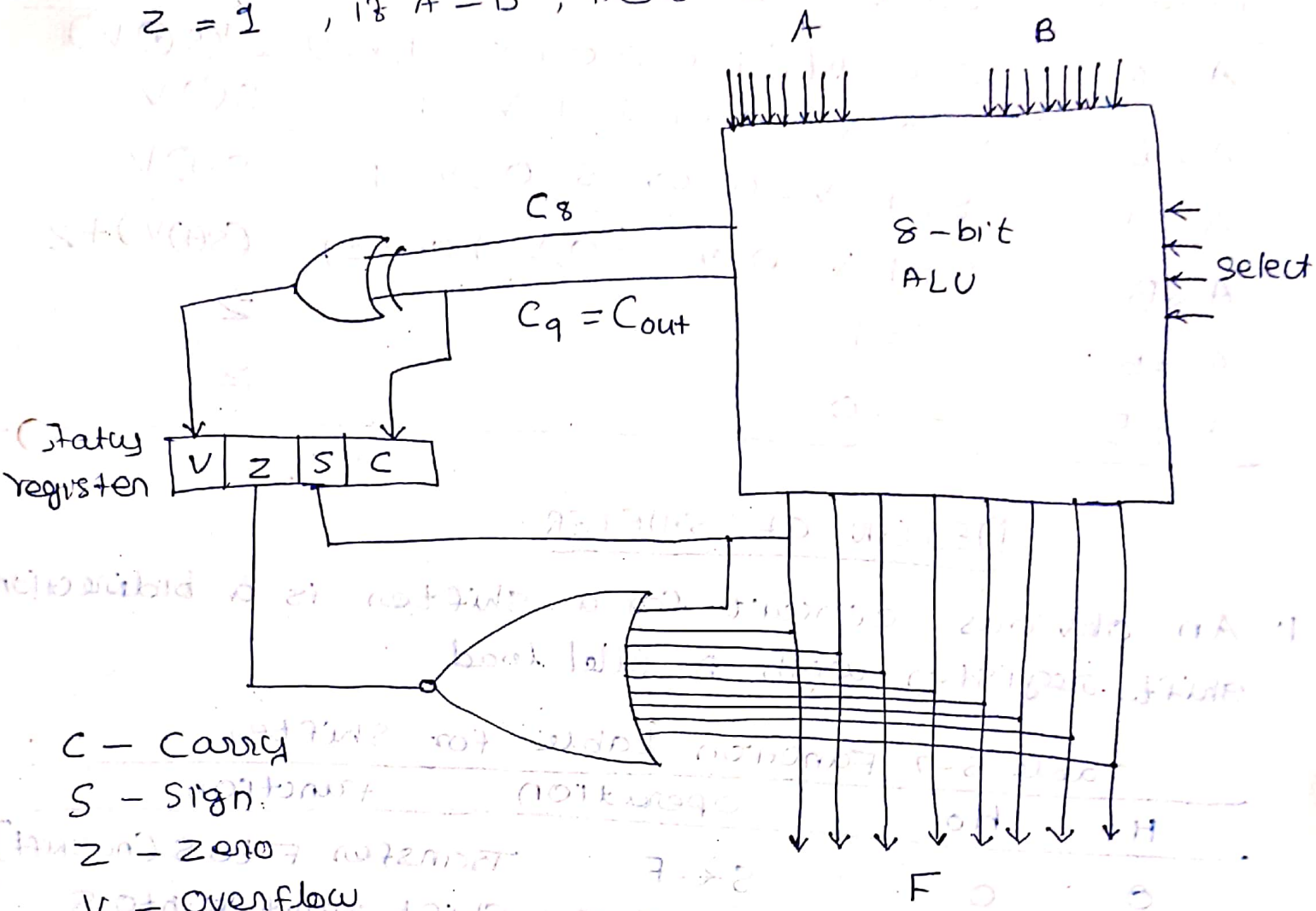
For the 8-bit ALU, V is set if the result is

Scanned with CamScanner

greater than 127 or less than −128.

V = 1 , after the addition of two signed numbers.

Z = 1 , if A = B , $A \oplus B = 0$



C – Carry
S – Sign
Z – Zero
V – Overflow

Figure 5.20 Setting bits in a status register

Table 5.5 Status bits after the subtraction of unsigned numbers (A − B)

Relation	Condition of status bits	Boolean function
A > B	C = 1 and Z = 0	CZ'
A ≥ B	C = 1	C
A < B	C = 0	C'
A ≤ B	C = 0 or Z = 1	$C' + Z$
A = B	Z = 1	Z
A ≠ B	Z = 0	Z'

Scanned with CamScanner

Table 5-6 Status bits after the subtraction of sign-2's complement numbers (A - B)

Relation	Condition of status bits	Boolean function
$A > B$	$Z=0$ and $(S=0, V=0$ or $S=1, V=1)$	$Z'(S \oplus V)'$
$A \geq B$	$S=0, V=0$ or $S=1, V=1$	$S \odot V$
$A < B$	$S=1, V=0$ or $S=0, V=1$	$S \oplus V$
$A \leq B$	$S=1, V=0$ or $S=0, V=1$ or $Z=1$	$(S \oplus V)+Z$
$A = B$	$Z=1$	Z
$A \neq B$	$Z=0$	Z'

DESIGN OF SHIFTER

1. An obvious circuit for a shifter is a bidirectional shift register with parallel load.

Table 5-7 Function table for shifter

H_1	H_0	operation	Function
0	0	$S \leftarrow F$	Transfer F to S (no shift)
0	1	$S \leftarrow shrF$	Shift-right F into S
1	0	$S \leftarrow shlF$	Shift-left F into S
1	1	$S \leftarrow 0$	Transfer 0's into S

2. The transfer from a source register to a destination register can be done with one clock pulse if the shifter is implemented with a combinational circuit.
3. Figure 5.21 shows a combinational-logic shifter constructed with multiplexers.

Scanned with CamScanner

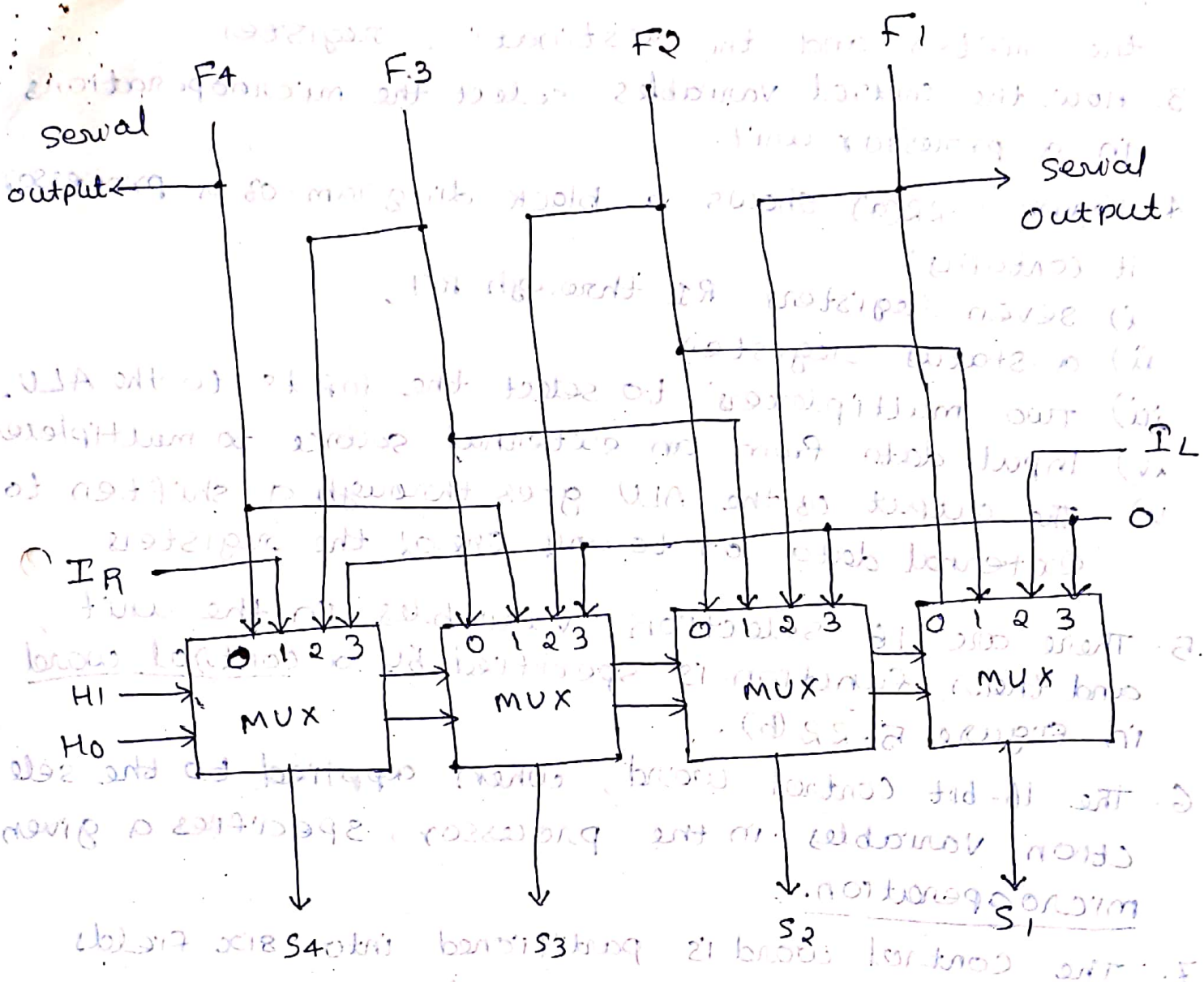


Figure 5.21 4-bit combinational logic shifter.

4. Inputs I_R and I_L serve as serial inputs for the least and first stages during a shift-right or shift left respectively.

PROCESSOR UNIT

1. The selection variables in a processor unit control the microoperations executed within the processor during any given clock pulse.
2. The selection variables control the buses, the ALU,

Scanned with CamScanner

the shifter and the destination register.

3. How the control variables select the microoperations in a processor unit.

4. Figure 5.22(a) shows a block diagram of a processor. It contains:
   i) seven registers R1 through R7,
   ii) a status register
   iii) two multiplexers to select the inputs to the ALU.
   iv) Input data from an external source to multiplexers
   v) The output of the ALU goes through a shifter to external data or to any one of the registers.

5. There are 16 selection variables in the unit, and their function is specified by a control word in Figure 5.22(b).

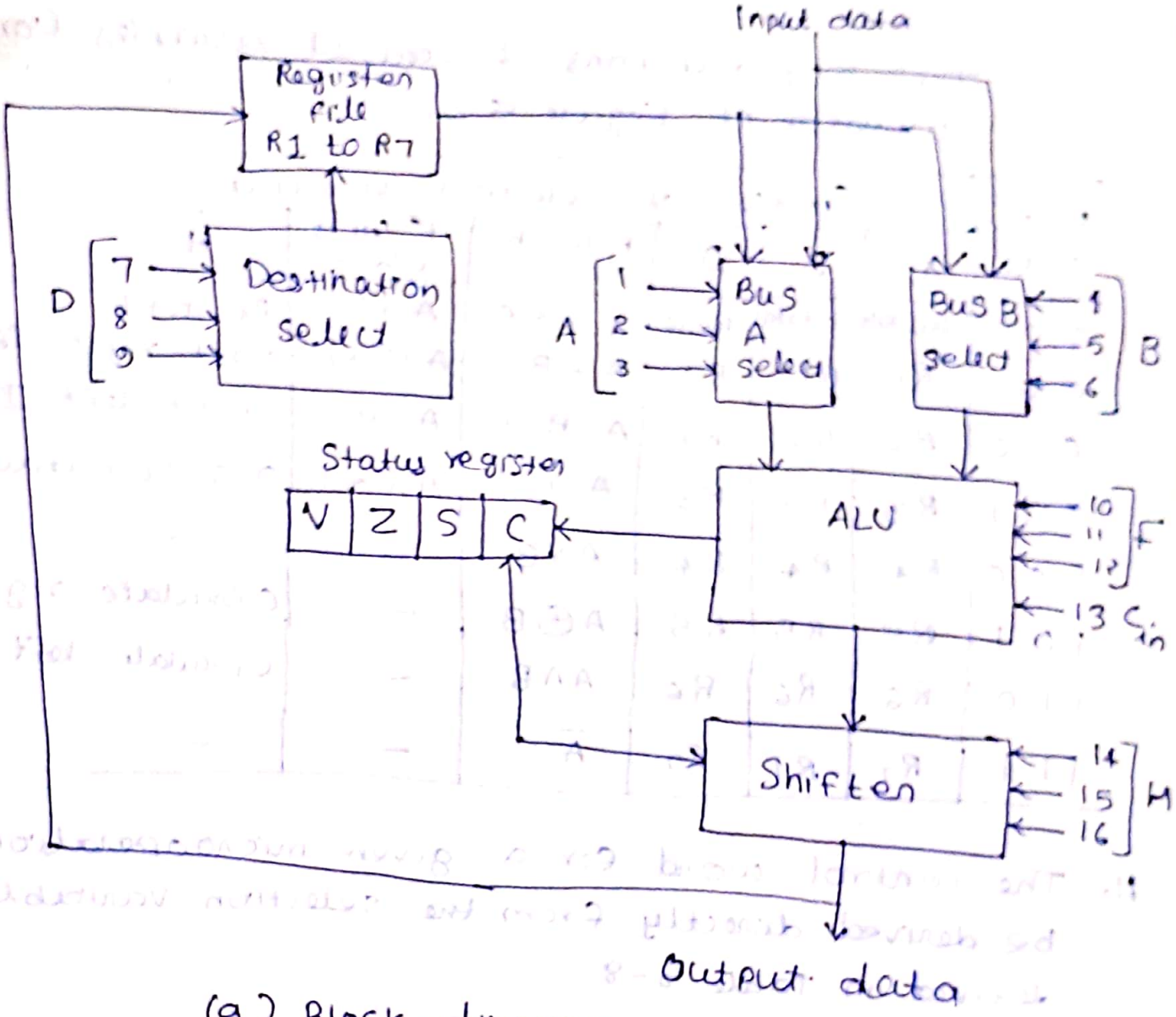
6. The 16-bit control word, when applied to the selection variables in the processor, specifies a given microoperation.

7. The control word is partitioned into six fields
   A - select a source register, left
   B - select a source register, Right
   D - select a destination register
   F, C_{in} - selects a function for the ALU
   H - selects the type of shift in the shifter unit.

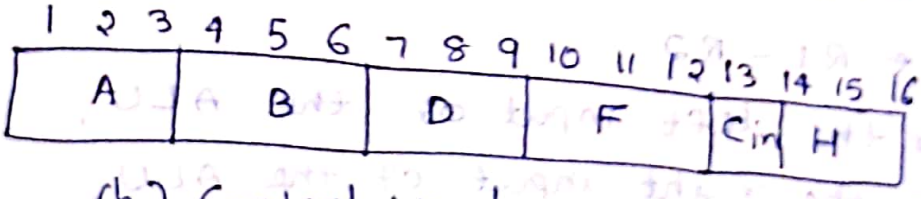
8. The three bits in the F field, together with the input carry C_{in}, provide the 12 operations of the ALU as specified in Table 5-4.

9. The first four entries for the code in the H field in Table 5.8 specify the shift operation of Table 5.7.

Scanned with CamScanner



(a) Block diagram



(b) Control word

Figure 5.22 Processor unit with control variables.

10. A third selection variable is used to specify either a 0 for the serial input I_R and I_L or a circular shift with the carry bit C.

$R \leftarrow$ cir C R is an abbreviation for

$R \leftarrow$ shr R, $R_n \leftarrow C$, $C \leftarrow R_1$.

Scanned with CamScanner

Table 5-8 Functions of control variables for the processor of Figure 5.2.2.

Binary code	Function of selection variables					
	A	B	D	F with Cin = 0	F with Cin = 1	H
0 0 0	Input data	I/p data	None	A, C ← 0	A + 1	No shift
0 0 1	R1	R1	R1	A + B	A + B + 1	shift-right, $I_R = 0$
0 1 0	R2	R2	R2	A − B − 1	A − B	Shift-left, $I_L = 0$
0 1 1	R3	R3	R3	A − 1	A, C ← 1	0's to output bus
1 0 0	R4	R4	R4	A ∨ B	—	—
1 0 1	R5	R5	R5	A ⊕ B	—	Circulate-right with C
1 1 0	R6	R6	R6	A ∧ B	—	Circulate-left with C
1 1 1	R7	R7	R7	$\bar{A}$	—	—

11. The control word for a given microoperation can be derived directly from the selection variables defined in Table 5-8.

Eg The subtract microoperation:

R1 ← R1 − R2

i) R1 for the left input of the ALU,
ii) R2 for the right input of the ALU,
iii) A − B for the ALU operation,
iv) no shift for the shifter, and
v) R1 for the destination register.

From Table 5-8, derive the control word for this operation to be 001010001010000:

Scanned with CamScanner

A	B	D	F	C_{in}	H
001	010	001	010	1	000

TABLE-5-9 Examples of microoperations for processor

Microoperation	Control word A	B	D	F	C_{in}	H	Function
R1 ← R1 − R2	001	010	001	010	1	000	subtract R2 from R1
R3 − R4	011	100	000	010	1	000	Compare R3 and R4
R5 ← R4	100	000	101	000	0	000	Transfer R4 to R5
R6 ← Input	000	000	110	000	0	000	Input data to R6
Output ← R7	111	000	000	000	0	000	output data from R7
R1 ← R1, C ← 0	001	000	001	000	0	000	Clear carry bit C
R3 ← shl R3	011	011	011	100	0	010	shift-left R3 with I_L=0
R1 ← crc R1	001	001	001	100	0	101	Circulate-right R1 with carry
R2 ← 0	000	000	010	000	0	011	Clear R2

DESIGN OF ACCUMULATOR

1. Some processor units distinguish one register from all others and call it an accumulator register.

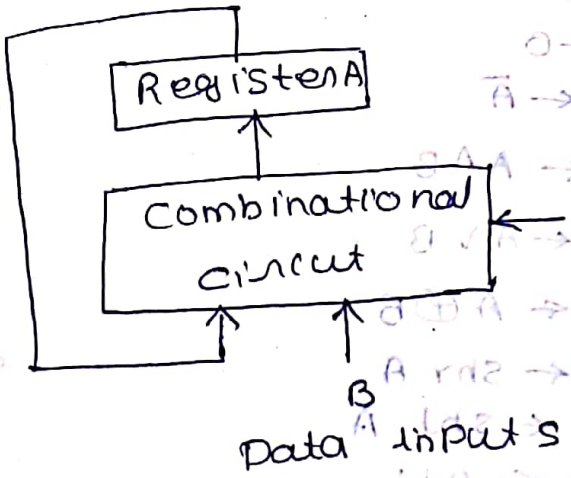


Figure 5-22 Block diagram of accumulator

Scanned with CamScanner

2. The organization of a processor unit with an accumulator register is shown in Figure 5.20.
3. Because of the feedback connection from the output of the register to one of the inputs in the ALU, the accumulator register and its associated logic, when taken as one unit, constitute a sequential circuit.
4. The block diagram of an accumulator that forms a sequential circuit is shown in Figure 5.22. The A register and the associated combinational circuit constitute a sequential circuit.
5. An accumulator is a multi-function register that by itself can be made to perform all of the micro-operations in a processor unit.
6. The set of microoperations for the accumulator is given in Table 5.10

TABLE 5-10 List of microoperations for an accumulator

Control variable	Microoperation	Name
P1	$A \leftarrow A + B$	Add
P2	$A \leftarrow 0$	clear
P3	$A \leftarrow \bar{A}$	complement
P4	$A \leftarrow A \wedge B$	AND
P5	$A \leftarrow A \vee B$	OR
P6	$A \leftarrow A \oplus B$	Exclusive-OR
P7	$A \leftarrow shr\ A$	shift-right
P8	$A \leftarrow shl\ A$	shift-left
P9	$A \leftarrow A + 1$	increment
	If (A=0) then (Z=1)	check for zero

Scanned with CamScanner

7. Control variables P_1 through P_3 are generated by control logic circuits and should be considered as control functions that initiate the corresponding register-transfer operations.

Design Procedure

1. The accumulator consists of n stages and n flip-flops, $A_1, A_2, \ldots, A_n$, numbered consecutively starting from the rightmost position.
2. In the design procedure, we consider only one typical stage i with the understanding that an n-bit accumulator consists of n stages for $i = 1, 2, \ldots, n$.
3. Each stage A_i is interconnected with the neighboring stage A_{i-1} on its right and stage A_{i+1} on its left.
4. The register will be designed using JK-type flip-flops.

1) Add B to A (P_1):

a) Add is initiated when $P_1 = 1$

b) Add part of accumulator use a parallel adder composed of full-adder circuits.

c) The full-adder accept as inputs
- the present state of A_i,
- the data input B_i and
- a previous carry bit C_i.

d) The sum bit generated in the full-adder must be transferred to flip-flop A_i, and the output carry C_{i+1} must be applied to the input carry of the next stage.

Scanned with CamScanner

e) The state table of a full-adder, when considered as a sequential circuit is shown in Figure 5.23.

Present state	Inputs		Next state	Flip-Flop inputs		Output
A_i	B_i	C_i	A_i	JA_i	KA_i	C_{i+1}
0	0	0	0	0	x	0
0	0	1	1	1	x	0
0	1	0	1	1	x	0
0	1	1	0	0	x	0
1	0	0	1	x	0	0
1	0	1	0	x	1	1
1	1	0	0	x	1	1
1	1	1	1	x	0	1

Figure 5-23 Excitation table for add microoperation

$$JA_i = B_i C_i' + B_i' C_i$$

$$KA_i = B_i C_i' + B_i' C_i$$

$$C_{i+1} = A_i C_i + A_i B_i + B_i C_i$$

- The value of flip-flop A_i before a clock pulse specifies the present state in the sequential circuit.
- The value of A_i after the application of a clock pulse specifies the next state.
- The next state of A_i is a function of its present state and inputs B_i and C_i.

f) The present state and inputs in the state table correspond to the inputs of a full-adder.

Scanned with CamScanner

9) Excitation table for the JK flip-flop

1) When both present state and next state are 0, the J input must remain at 0 and the K input can be either 0 or 1.

2) When both present state and next state are 1, the K input must remain at 0 while the J input can be 0 or 1.

3) If the flip-flop is to have a transition from the 0-state to the 1-state, J must be equal to 1 since the J input sets the flip-flop.

4) For a transition from the 1-state to the 0-state, we must have K = 1, since the K input clears the flip-flop.

b) The equations should effect the flip-flop only when P_1 is enabled; therefore, they should be ANDed with control variable P_1.

i) The part of the combinational circuit associated with the add microoperation can be expressed with three Boolean functions:

$$JA_i = B_i C_i' P_1 + B_i' C_i P_1$$
$$KA_i = B_i C_i' P_1 + B_i' C_i P_1$$
$$C_{i+1} = A_i B_i + A_i C_i + B_i C_i$$

The first two equations are identical and they specify a condition for complementing A_i.

if K = 1 and J = 1, the flip-flop is complemented and goes from the 0-state to the 1-state as required.

2. Clear (P_2)

a) Control variable P_2 clears all flip-flops in register A.

b) To cause this transition in a JK flip-flop, we need only apply control variable P_2 to the K input of the flip-flop.

Scanned with CamScanner

The input functions for the clear microoperation are:

$JA_i = 0$

$KA_i = P_2$

3. Complement (P_3)

a) Control variable P_3 complements the state of register A.

b) To cause this transition in a JK-flip-flop, we need to apply P_3 to both the J and K inputs:

$JA_i = P_3$

$KA_i = P_3$

4. AND (P_4):

a) The AND microoperation is initiated with control variable P_4.

b) This operation forms the logic AND operation between A_i and B_i and transfers the result to A_i.

Present state	Input	Next state	Flip-flop inputs	
A_i	B_i	A_i	JA_i	KA_i
0	0	0	0	X
0	1	0	0	X
1	0	0	X	1
1	1	1	X	0

Figure 5.24 Excitation table for AND operation

$JA_i = 0$

$KA_i = B_i' P_4$

5. OR (P_5):

a) Control variable P_5 initiates the logic OR operation between A_i and B_i, with the result transferred to A_i.

Scanned with CamScanner

Present state	Input	Next state	Flip-flop inputs	
A_i	B_i	A_i	JA_i	KA_i
0	0	0	0	X
0	1	1	1	X
1	0	1	X	0
1	1	1	X	0

Figure 5.25 Excitation table for OR operation

The input functions for the OR microoperation are:

$$JA_i = B_i \cdot P5$$

$$KA_i = 0$$

6. Exclusive-OR (P6):

a) Control signal P6 initiates ~~EOR~~ Exclusive-OR

b) This operation forms the logic exclusive-OR between A_i and B_i and transfers the result to A_i.

Present state	Input	Next state	Flip-flops inputs	
A_i	B_i	A_i	JA_i	KA_i
0	0	0	0	X
0	1	1	1	X
1	0	1	X	0
1	1	0	X	1

Figure 5.26 Excitation table for XOR operation.

$$JA_i = B_i \cdot P6$$

$$KA_i = B_i \cdot P6$$

Scanned with CamScanner

7. Shift - right (P7):

a) This operation shifts the contents of the A register one position to the right.

$$JA_i = A_{i+1} P_7$$

$$KA_i = A'_{i+1} P_7$$

8. Shift - left (P8):

a) This operation shifts the A register one position to the left.

b) The value of A_{i-1}, which is one position to the right of stage i, must be transferred to A_i.

$$JA_i = A_{i-1} P_8$$

$$KA_i = A'_{i-1} P_8$$

9. Increment (PD):

a) This operation increments the contents of the A register by one.

b) The register behaves like a synchronous binary counter with PD enabling the count. The Boolean functions for a typical stage can be expressed as follows:

$$JA_i = E_i$$

$$KA_i = E_i$$

$$E_{i+1} = E_i A_i \quad i = 1, 2, \ldots, n$$

$$E_1 = PD.$$

10. Check for zero (Z)

a) Variable Z is an output from the accumulator

Scanned with CamScanner

used to indicate a zero content in the A register.

b) This output is equal to binary 1 when all the flip-flops are cleared.

c) When a flip-flop is cleared, its complement output, Q', is equal to 1.

The Boolean functions for a typical stage can be expressed as follows:

$$z_{i+1} = z_i \cdot A_i' \qquad i = 1, 2, \ldots, n$$

$$z_1 = 1$$

$$z_{n+1} = z$$

Variable z becomes 1 if the output signal from the last stage z_{n+1}, is 1.

One Stage of Accumulator

1. A typical accumulator stage consists of all the circuits that were derived for the individual microoperations.
2. Combining all the input functions for the J and the K inputs of flip-flop A_i produces a composite set of input Boolean functions for a typical stage:

$$JA_i = B_i C_i' P_1 + B_i' C_i P_1 + P_3 + B_i P_5 + B_i P_6 + A_{i+1} P_7 + A_{i-1} P_8 + E_i$$

$$KA_i = B_i C_i' P_1 + B_i' C_i P_1 + P_2 + P_3 + B_i' P_4 + B_i P_6 + A_{i+1}' P_7 + A_{i-1}' P_8 + E_i$$

3. Each stage in the accumulator must also generate the carries for the next stage:

$$C_{i+1} = A_i B_i + A_i C_i + B_i C_i$$

$$E_{i+1} = E_i A_i'$$

$$Z_{i+1} = Z_i A_i'$$

Scanned with CamScanner

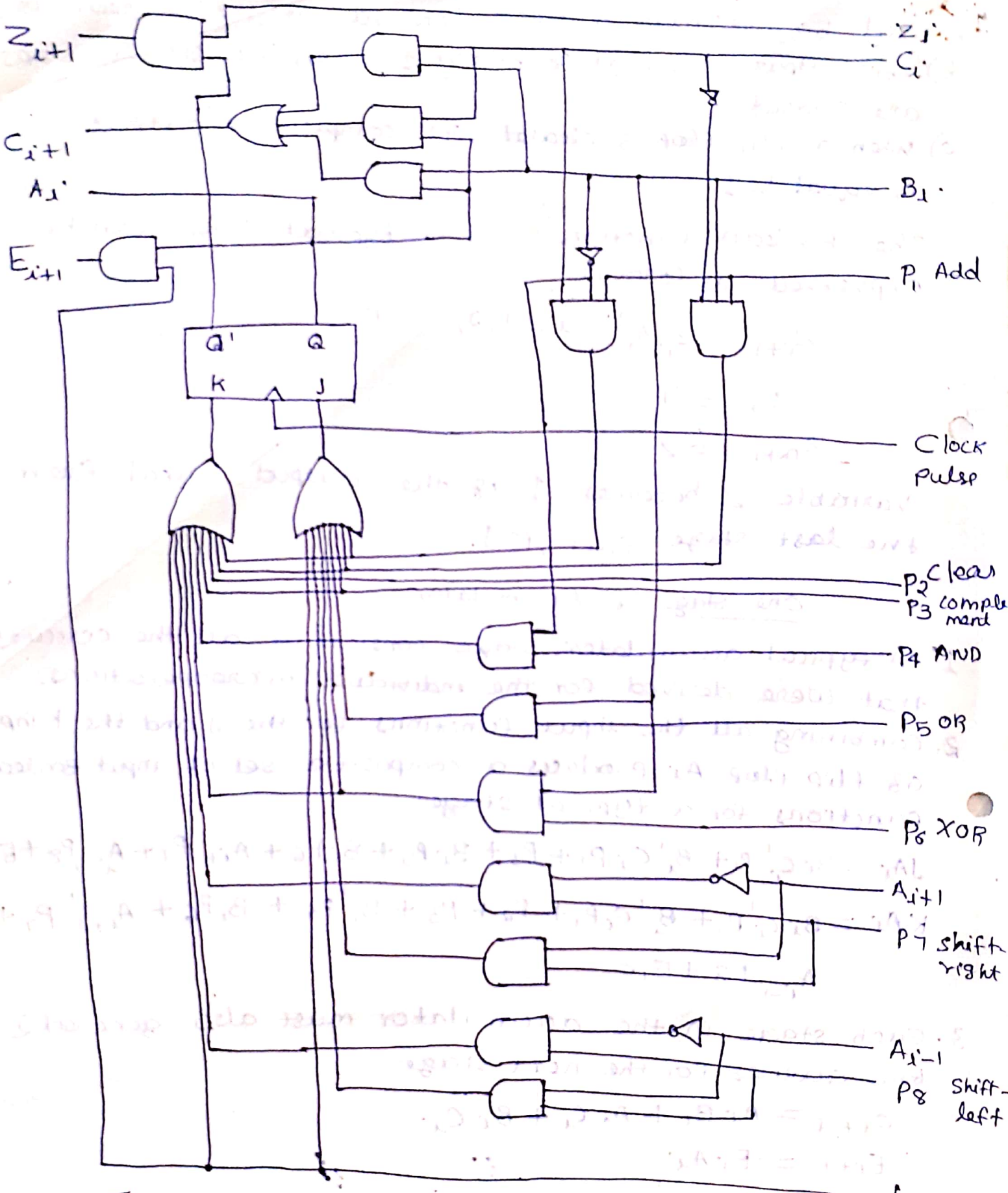


Figure 5-27 One typical stage of the accumulator.

(60)

Scanned with CamScanner

Complete Accumulator.

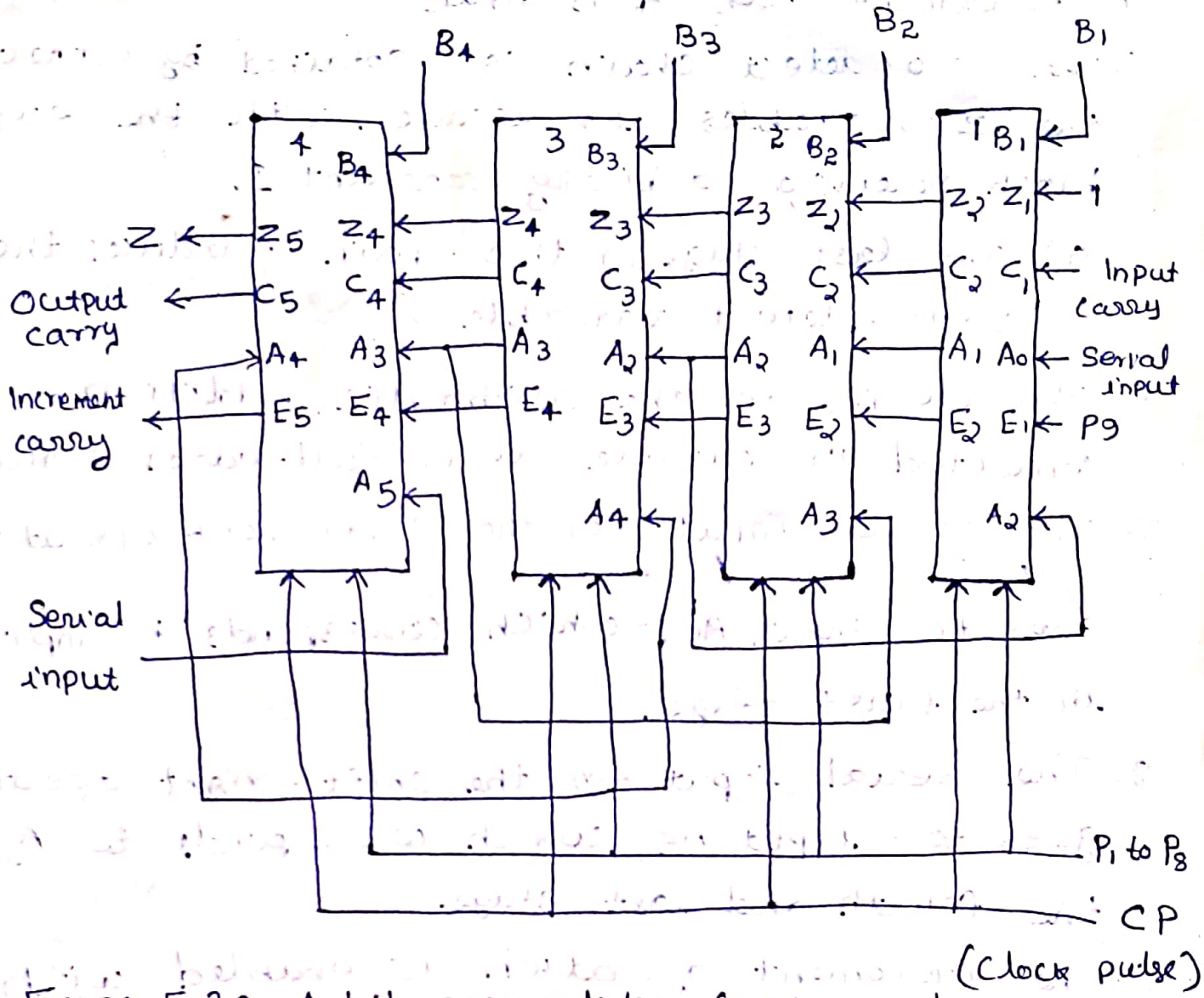


Figure 5·28 4-bit accumulator constructed with four stages.

1. The number on top of each block represents the bit position in the accumulator.
2. All blocks receive eight control variables P_1 through P_8 and clock pulse from CP.
3. The other six inputs and four outputs in each block are identical to those of a typical stage, except that subscript i is now replaced by the particular number in each block.

Scanned with CamScanner

4. The circuit has A B inputs.
5. The zero-detect chain is obtained by connecting the z variables in cascade, with the first block receiving a binary constant 1.
   a) The last stage in this chain produces the zero-detect variable z.
6. The carries for the arithmetic addition are connected in cascade as in full-adder circuits.
7. The serial input for the shift-left operation goes to input A_0, which corresponds to input A_1 in the first stage.
8. The serial input for the shift-right operation goes to input A_5, which corresponds to A_{i+1} in the fourth and last stage.
9. The increment operation is enabled with control variable P_0 in the first stage. The other blocks receive the increment carry from the previous stage.

Scanned with CamScanner

Module VI

Control Logic Design

Control Organisation

1. Design methods that use state and excitation tables can be used in theory, but in practice they are difficult to manage.
2. Control circuit obtained by state and excitation table method usually requires an excessive number of flip-flops and gates, which implies the use of small scale integrated circuits.
3. Major goal of control logic design should be the development of a circuit that implements the desired control sequence in a logical and straight forward manner.
4. Methods for control logic design
   1. One Flip-Flop per state method.
   2. Sequence register and decoder method.
   3. PLA control
   4. Microprogram control

   } LSI (Large scale integrated)

   Hard-wired control, SSI & MSI (Medium Scale Integrated.

1. One Flip-Flop per State Method
   i) only one flip-flop is set at any particular time, all others are cleared.
   ii) Uses maximum number of flip-flops.
   iii) A sequential circuit with 12 states requires a minimum of four flip-flops because $2^3 < 12 < 2^4$. By this method, the control circuit uses

Scanned with CamScanner

12 flip-flops, one for each state.

Advantages

1. Simplicity in design
2. Controller can be designed by inspecting state diagram.
3. savings in design effort
4. Increased operational simplicity.
5. Potential decrease in the combinational circuits required to implement the complete sequential circuit.

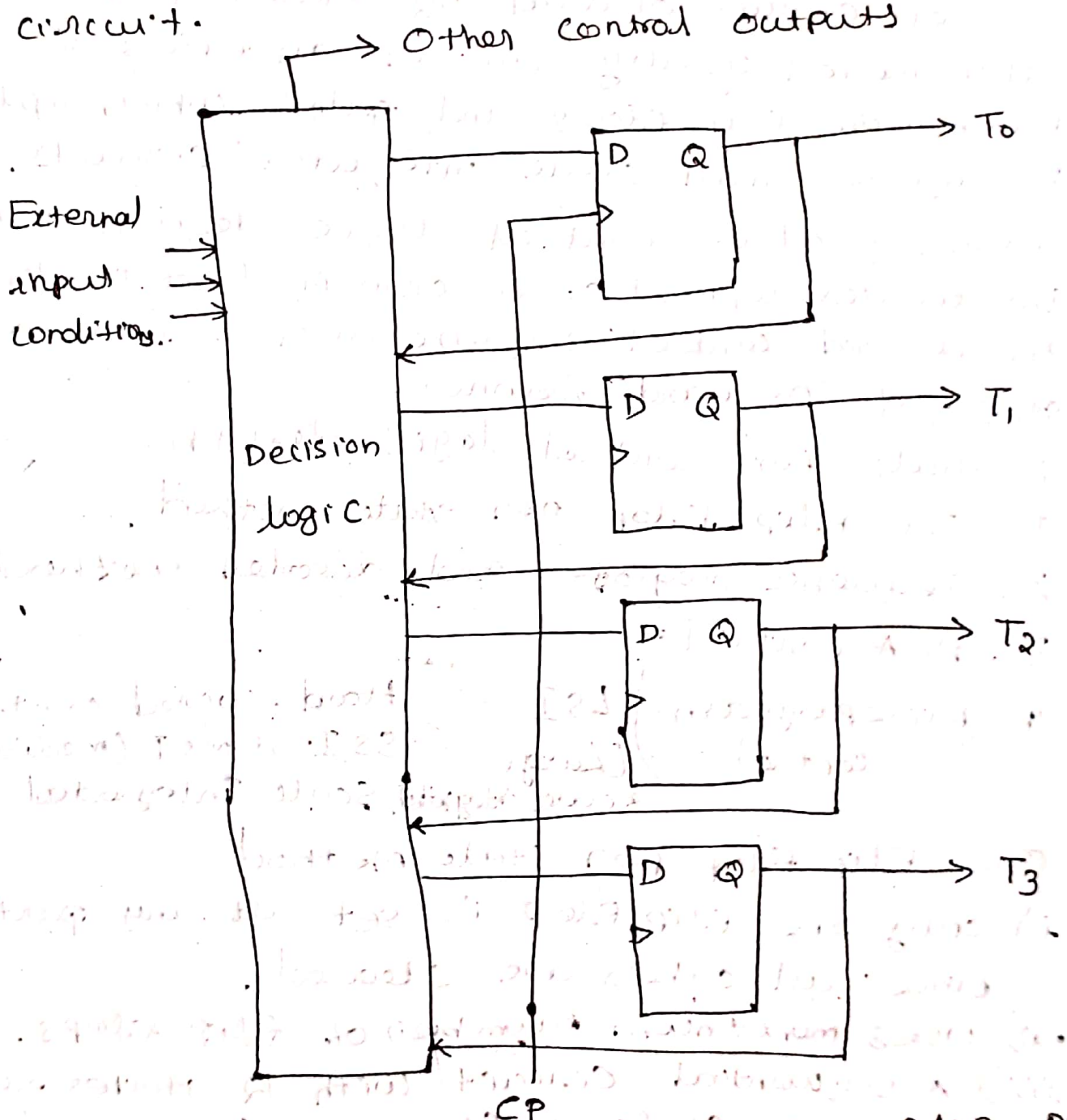


Figure 6.1 Control logic with one flip-flop per state.

Scanned with CamScanner

Figure 6-1 shows the configuration of a four-state sequential control logic that uses four D-type flip-flops: one flip-flop per state T_i, $i = 0, 1, 2, 3$.

1. Each of the flip-flop outputs is connected to the data-processing section of the digital system to initiate certain microoperations.
2. One flip-flop per state method is sometimes called a ring-counter controller.

If the control circuit does not need external inputs for its sequencing, the circuit reduces to a straight shift register with a single bit shifted from one position to the next.

If the control sequence must be repeated over and over again, the control reduces to a ring counter. A ring counter is a shift register with the output of the last flip-flop connected to the input of the first flip-flop.

2. Sequence Register and Decoder Method

1. Uses a register to sequence the control states.
2. The register is decoded to provide one output for each state.
3. For n flip-flops in the sequence register, the circuit will have 2^n states and the decoder will have 2^n outputs.
4. Both the sequence register and decoder are MSI devices.
5. Figure 6-2 shows the configuration of a four-state sequential control logic

Scanned with CamScanner

6. The transition to the next state in the sequence register is a function of the present state and the external input conditions.
7. If the control circuit does not need external inputs, the sequence register reduces to a counter that continuously sequences through the four states. For this reason, this method is sometimes called a counter-decoder method.

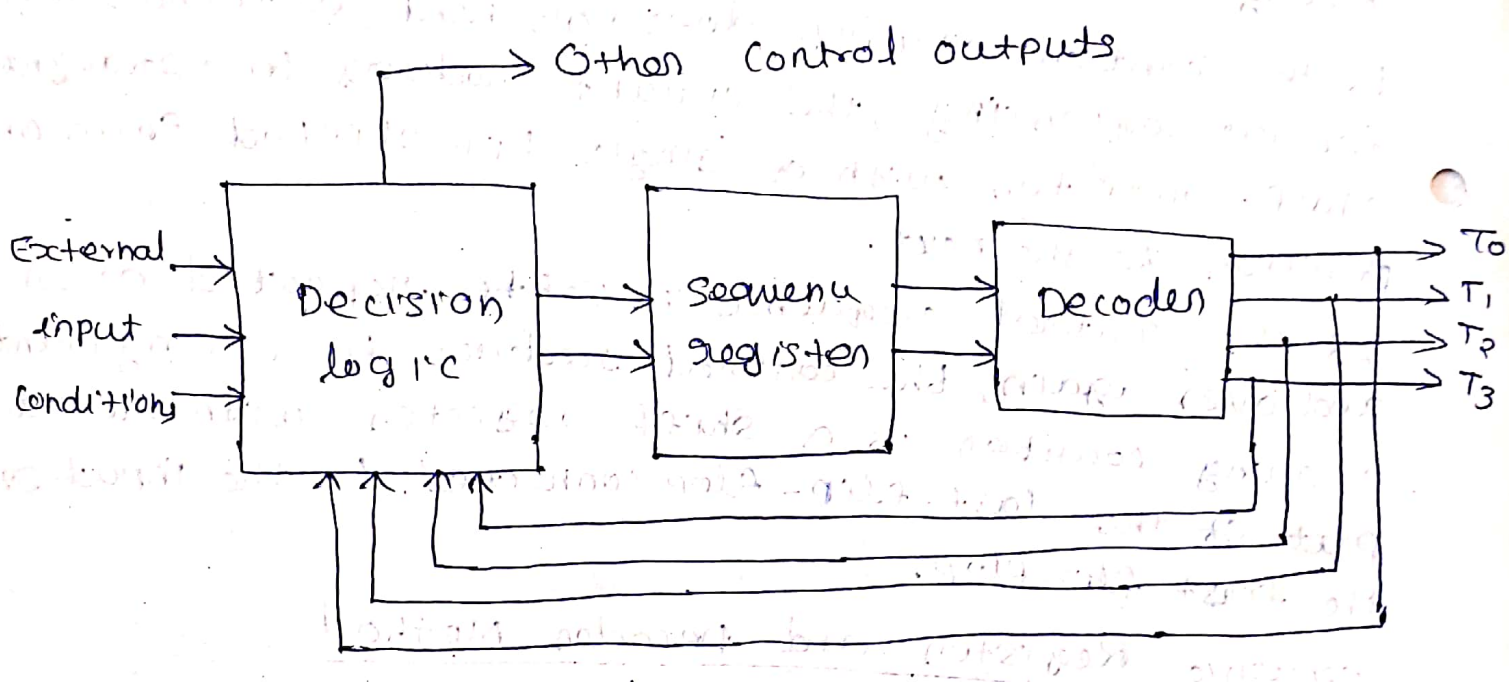


Figure 6-2 Control logic with sequence register and decoder.

3. PLA Control

1. All combinational circuits are implemented with a PLA.
2. By using a PLA for the combinational circuit, it is possible to reduce the number of ICs and the number of interconnection wires.
3. Figure 6-3 shows the configuration of a PLA controller. An external sequence register establishes

Scanned with CamScanner

the present state of the control circuit.

4. The PLA outputs determine which microoperations should be initiated, depending on external input conditions and the present state of the sequence register.

5. At the same time, other PLA outputs determine the next state of the sequence register.

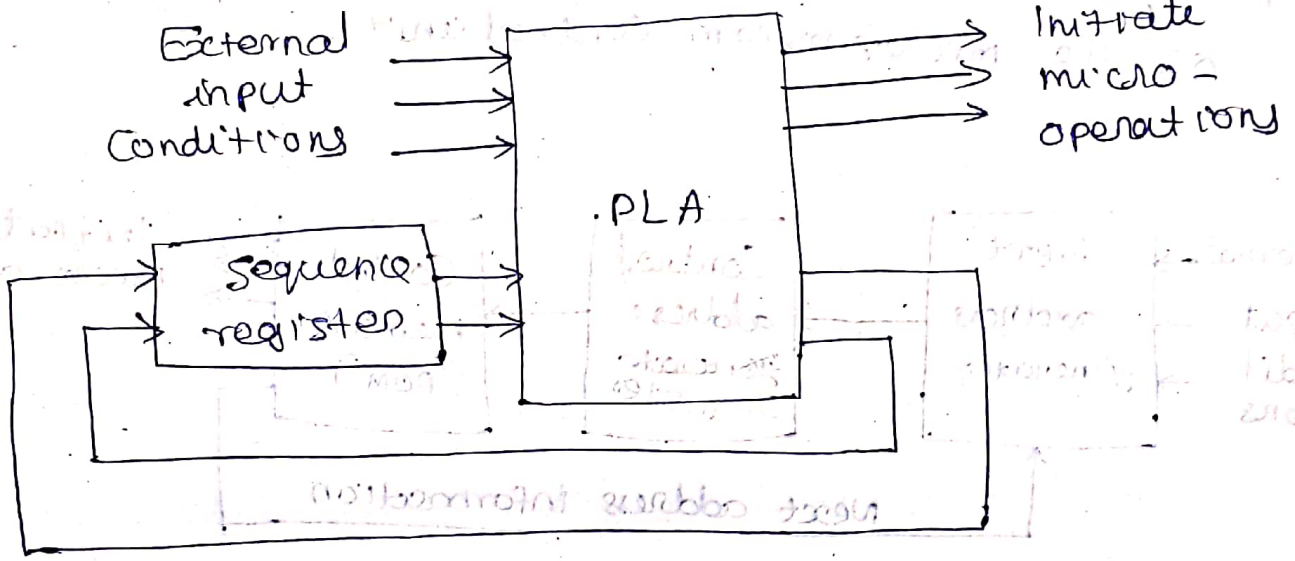


Figure 6.3 PLA control logic

4. Microprogram Control

1. The purpose of the control unit is to initiate a series of sequential steps of microoperations.
2. During any given time, certain operations are to be initiated while all others remain idle.
3. The control variables at any given time can be represented by a string of 1's and 0's called a control word.
4. A control unit whose control variables are stored

Scanned with CamScanner

in a memory is called a microprogrammed control unit.

5. Each control word of memory is called a micro-instruction, and a sequence of microinstructions is called a microprogram.

6. The control memory can be a ROM, PLA or WCM (writable control memory) - magnetic disk storage.

7. Figure 6.4 illustrates the general configuration of the microprogram control unit.

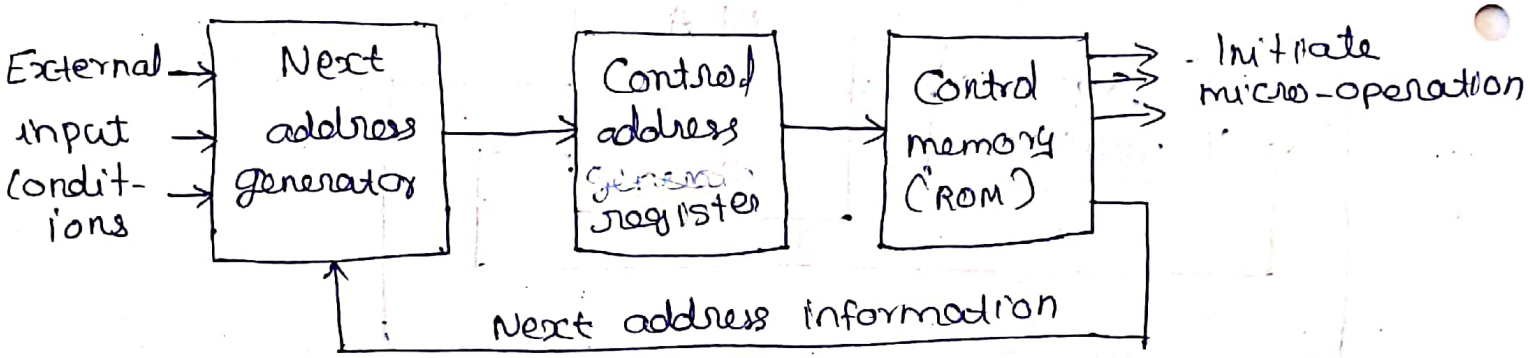


Figure 6.4 Microprogram control logic.

8. The control memory is ROM within which all control information is permanently stored.

9. The control memory address register specifies the control word read from control memory.

10. ROM operates as a combinational circuit, with the address value as the input and the corresponding word as the output.

11. The content of the specified word remains on the output wires as long as the address value remains in the address register.

Scanned with CamScanner

12. The location of the next microinstruction may be the next one in sequence, or it may be located somewhere else in the control memory. For this reason, it is necessary to use some bits of the microinstruction to control the generation of the address for next microinstruction. The next address may also be a function of external input conditions.

DESIGN OF HARD-WIRED CONTROL

The design is carried out in five consecutive steps.

1. The problem is stated.
2. An initial equipment configuration is assumed.
3. An algorithm is formulated.
4. The data-processor part is specified.
5. The control logic is designed.

1. Statement of the problem

The problem here is to implement with hardware the addition and subtraction of two fixed-point binary numbers represented in sign-magnitude form. Complement arithmetic may be used, provided the final result is in sign-magnitude form.

2. Equipment configuration

Two binary numbers of n bits.
magnitude, $k = n-1$ & stored in registers A and B.
sign bits, in flipflops As & Bs.
ALU performs arithmetic operations
1-bit register E serves as overflow-flip-flop.
o/p carry is transferred to E.

Scanned with CamScanner

two control inputs q_a (add) and q_s (subtract) & x to indicate the end of operation.

Figure 6-5 shows the registers and associated equipment.

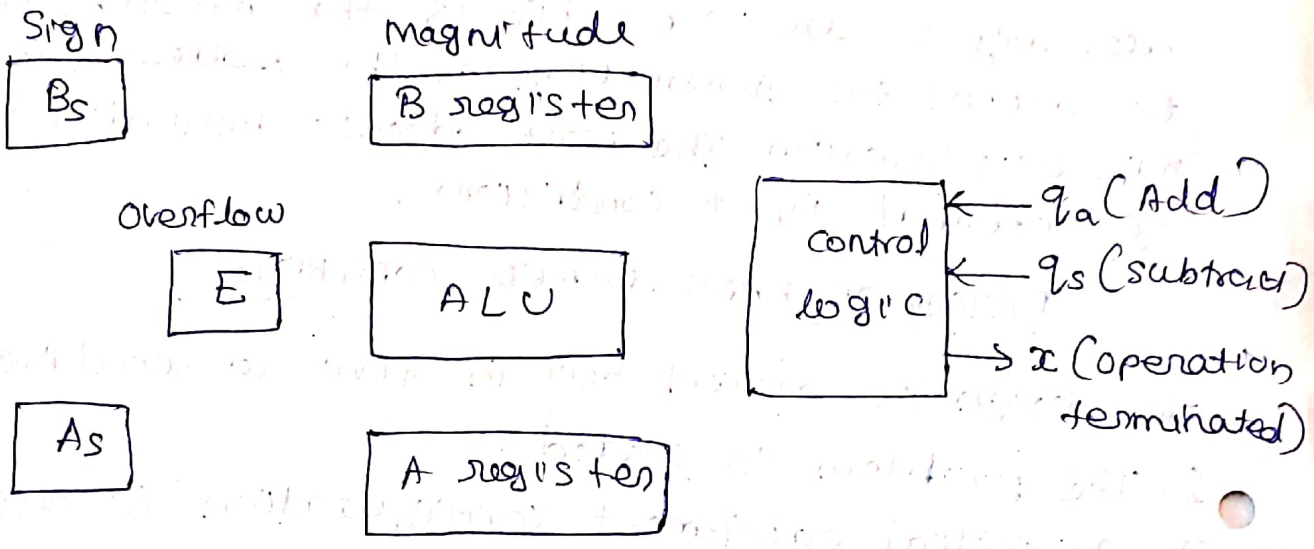


Figure 6-5 Register configuration for the adder-subtracter.

3. Derivation of the Algorithm

When the numbers are subtracted or added algebraically, we find that there are eight different conditions to consider, depending on the sign of the numbers and the operation performed.

$$(\pm A) \pm (\pm B)$$

For subtraction we change the sign of B and add.

$$(\pm A) - (+B) = (\pm A) + (-B)$$

$$(\pm A) - (-B) = (\pm A) + (+B)$$

This reduces to 4 possible conditions.

$$(\pm A) + (\pm B)$$

i) when signs of A & B are same.

- add A + B, sign is the common sign.

Scanned with CamScanner

ii) when the signs of A & B are not the same
- Subtract the smaller number from larger
- sign is the sign of the larger

	if $A \geq B$	if $A < B$
a) $(+A) + (+B) = +(A+B)$		
b) $(+A) + (-B) =$	$+(A-B)$	$-(B-A)$
c) $(-A) + (+B) =$	$-(A-B)$	$+(B-A)$
d) $(-A) + (-B) = -(A+B)$		

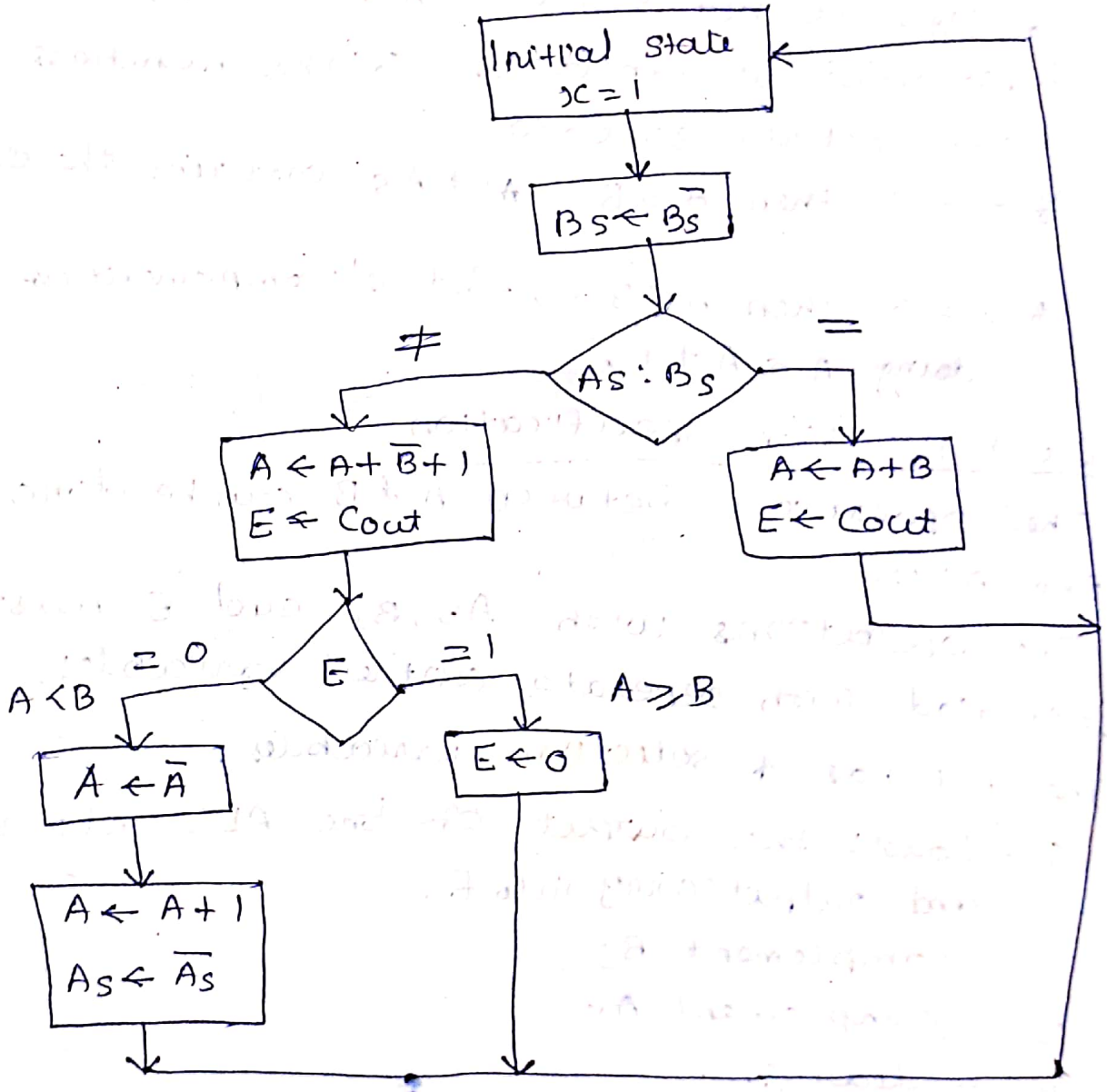


Figure 6-6 Flowchart for sign-magnitude addition and subtraction.

Scanned with CamScanner

1) The Flowchart of Figure 6-6 shows how we can implement sign-magnitude addition and subtraction with equipment of Figure 6-5.

2) Operation initiated by either q_s or q_a
If $q_s = 1$, $B_s \leftarrow \overline{B_s}$, If $q_a = 1$, B left unchanged.

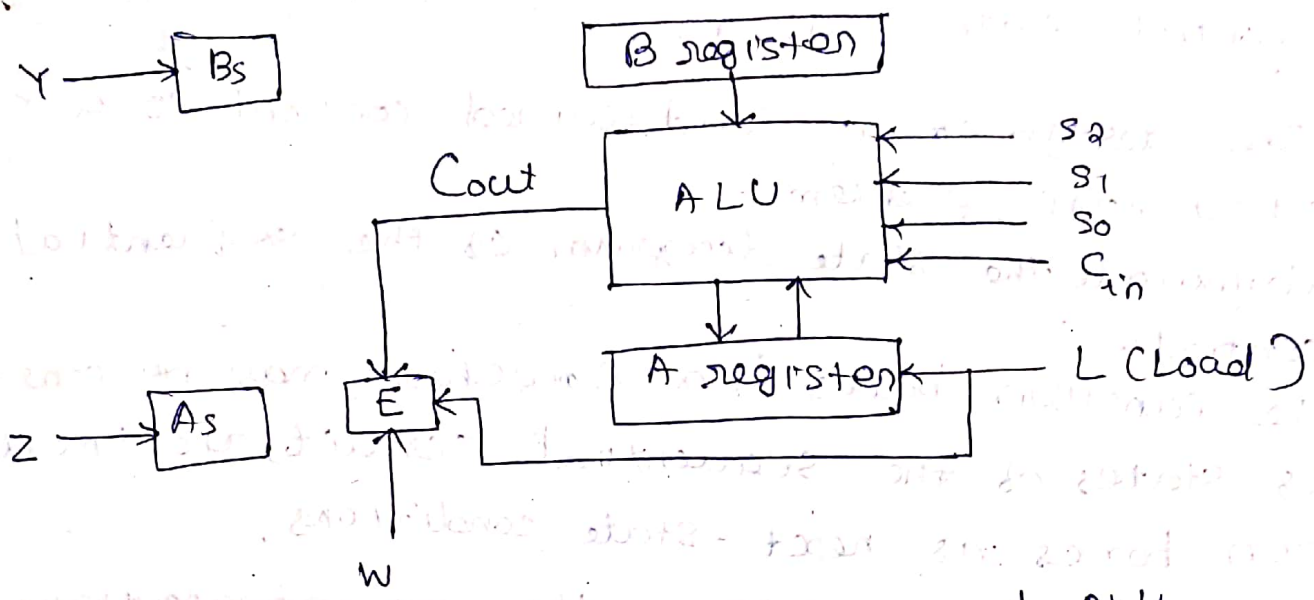
3) compare two signs.
If signs are equal do $A \leftarrow A + B$, $E \leftarrow Cout$. go to initial state.
sign is same as A_s.
If signs are not same $A \leftarrow A + \overline{B} + 1$, $E \leftarrow Cout$
- no overflow can occur If the numbers are subtracted. so $E \leftarrow 0$.
If $E = 1$ then $A \geq B$ and A_s contains the original value.
If $E = 0$ then $A < B$, find 2's complement of A by doing $A \leftarrow \overline{A} + 1$.

4. Data processor specification

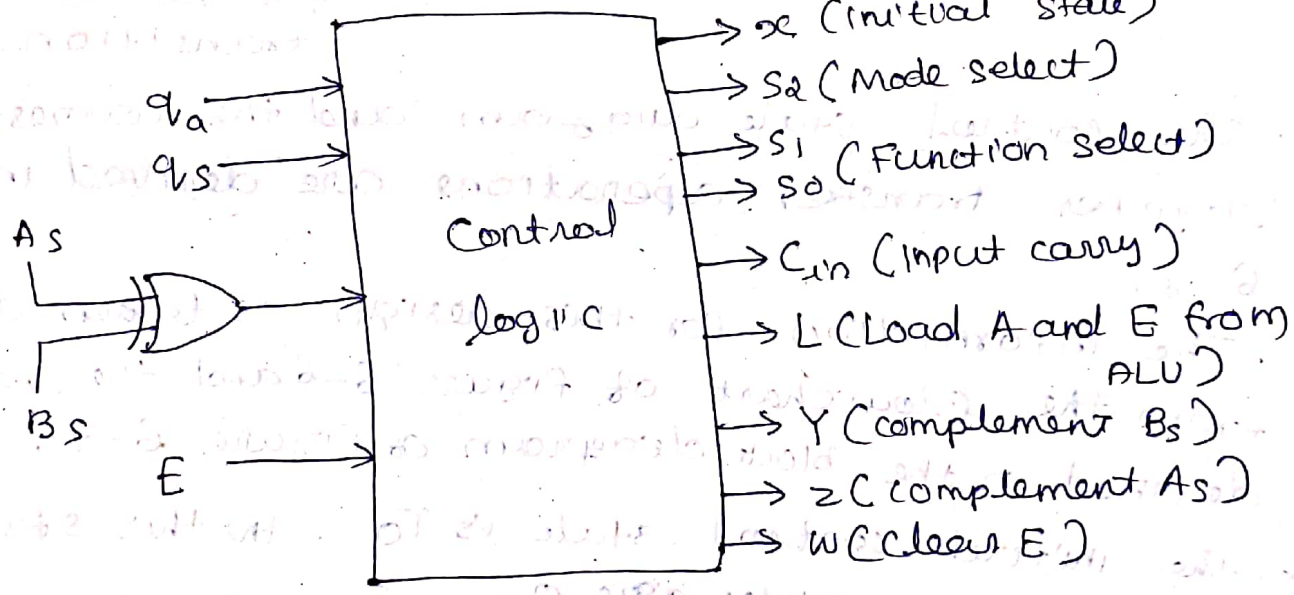
1- The operations between A & B can be done with the ALU.
2. The operations with A_s, B_s and E must be initiated with seperate control variables.
3. The ALU has 4 selection variables.
L - Loads the output of the ALU into register A and output carry into E.
Y - complement B_s
Z - complement A_s
W - clear E.
4. Figure 6-7 shows the data processor with the

Scanned with CamScanner

required control variables.



(a) Data processor registers and ALU



(b) Control block diagram

Figure 6-7 System block diagram.

Scanned with CamScanner

$S = A_s \oplus B_s$ to compare b/w the two sign bits.

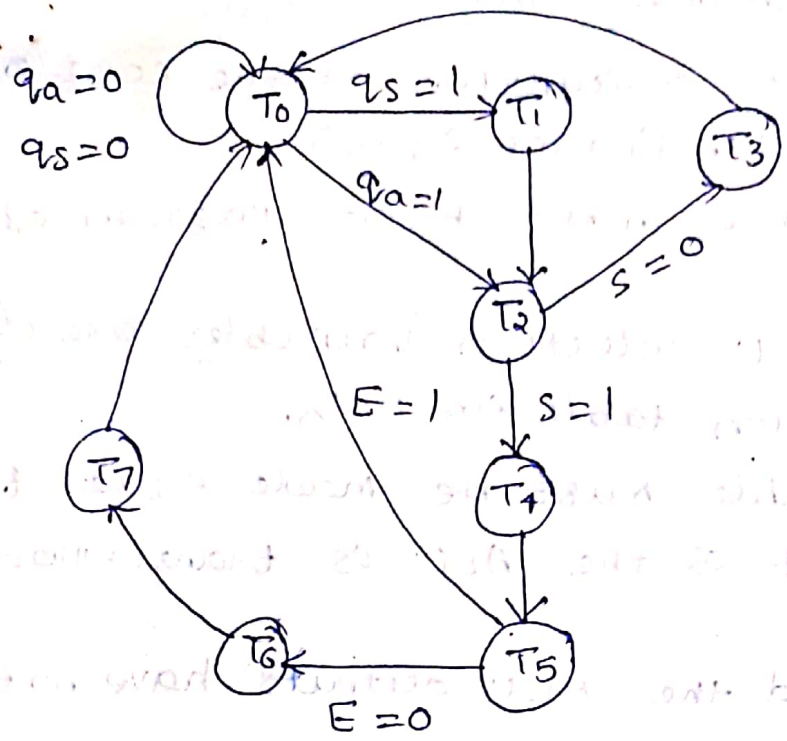
5. Control State Diagram

1. The design of a hard-wired control is a sequential-logic problem.
2. Formulate the state diagram of the sequential control.
3. The function boxes in a flowchart may be considered as states of the sequential circuit, and the decision boxes as next-state conditions.

function box — states — the microoperations
decision box — directed lines — the conditions for the next state transition.

4. The control state diagram and the corresponding register-transfer operations are derived in Figure 6-8.
5. The information for this design is taken directly from the flowchart of Figure 6-6 and the variables defined in the block diagram of Figure 6-7.
6. The initial control state is T_0. In this state $x=1$ all other variables are 0.
7. As long as q_a and q_s are 0, the control stays in its initial state.
8. If $q_s = 1$ goto T_1 else if $q_a = 1$ goto T_2.
9. The next state after T depends the value of S
   If $S=0$, goto T_3, if $S=1$ goto T_4.
10. From T_4 directly goes to T_5 where E is evaluated.

Scanned with CamScanner



q_a Add
q_s Subtract
$S=0$ signs alike
$S=1$ signs unlike
E Output carry

(a) State diagram.

T_0: Initial state $x=1$
T_1: $B_S \leftarrow \bar{B}_S$
T_2: nothing
T_3: $A \leftarrow A+B$, $E \leftarrow C_{out}$
T_4: $A \leftarrow A+\bar{B}+1$, $E \leftarrow C_{out}$
T_5: $E \leftarrow 0$
T_6: $A \leftarrow \bar{A}$
T_7: $A \leftarrow A+1$, $A_S \leftarrow \bar{A}_S$

	Control outputs							
x	S_2	S_1	S_0	C_{in}	L	Y	Z	W
1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0
0	0	1	0	1	1	0	0	0
0	0	0	0	0	0	0	0	1
0	1	1	1	0	1	0	0	0
0	0	0	0	1	1	0	1	0

(b) Sequence of register transfer.

(13) Figure 6-8 Control state diagram and sequence of microoperations.

Scanned with CamScanner

Design of Hard-wired control.

1. The control outputs are a function of the control states and are listed in Figure 6-8(b).
2. The outputs are defined in the block diagram of Figure 6-7(b).
3. The values for the ALU selection variables are determined from function table for ALU.
4. The L (load A) variable must be made equal to 1 every time the output of the ALU is transferred to register A.
   Otherwise, L is 0 and the ALU outputs have no effect on the register.
5. The control can be designed using the classical sequential-logic procedure. This procedure requires a state table with eight states, four inputs, and nine outputs.
   The sequential circuit to be derived from such a state table will not be easy to obtain because of the large number of variables.
   If the control is designed by the one flip-flop per state method the difficulty can be removed.
6. A control organization that uses one flip-flop per state has the convenient characteristic that the circuit can be derived directly from the state diagram by inspection.
7. No state or excitation tables are needed if D flip-flops are employed.

Scanned with CamScanner

8. The next state of a D flip-flop is a function of the D input and is independent of the present state.

9. Since the method requires one flip-flop for each state, we choose 8 D flip-flops and label their outputs $T_0, T_1, T_2, \ldots, T_7$.

10. The condition for setting a given flip-flop is specified in the state diagram.

eg: flip-flop T_2 is set with the next clock pulse if $T_1 = 1$ or if $T_0 = 1$ and $q_a = 1$. This condition can be defined with the Boolean function:

$$DT_2 = q_a T_0 + T_1$$

TABLE 6-1 Boolean functions for control

Flip-flop input functions	Boolean functions for output control
$DT_0 = q_a' q_s' T_0 + T_3 + E T_5 + T_7$	$x = T_0$
$DT_1 = q_s T_0$	$S_2 = T_6$
$DT_2 = q_a T_0 + T_1$	$S_1 = T_4 + T_6$
$DT_3 = S' T_2$	$S_0 = T_3 + T_6$
$DT_4 = S T_2$	$C_{in} = T_4 + T_7$
$DT_5 = T_4$	$L = T_3 + T_4 + T_6 + T_7$
$DT_6 = E' T_5$	$y = T_1$
$DT_7 = T_6$	$z = T_7$
	$w = T_5$

11. The flip-flop input functions are mutually exclusive and only one flip-flop can be set at any given time, all others are cleared.

Scanned with CamScanner

12. To specify the control outputs as a function of flip-flop states. This is done with the Boolean functions given in Table 6-1.
eg $L = 1$ during T_3, T_4, T_6 or T_7.

MICROPROGRAM CONTROL

1. The hard-wired control designed for the sign-magnitude adder-subtractor will be replaced by a micro program control to be designed subsequently.
2. The microprogram control organization is more efficient in large, complicated systems.
3. A state in control memory is represented by the address of a microinstruction.
4. Since there are eight states in the control, we choose a control memory with eight words having addresses 0 through 7.
5. The address of the control memory corresponds to the subscript number under the T's in the state diagram.
6. Inspection of the state diagram reveals that the address sequencing in the microprogram control must have the following capabilities.
   a) Provision for loading an external address as a result of the occurrence of external signals q_A and q_S.
   b) Provision for sequencing consecutive addresses.
   c) Provision for choosing between two addresses as a function of present values of the status variables S and E.

Scanned with CamScanner

7. Each microinstruction must contain a number of bits to specify the way that the next address is to be selected.

Hardware Configuration

The organisation of the microprogram control unit is shown in Figure 6-9.

1. The control memory is an 8-word by 14-bit ROM.
   - 1st 9-bits of microinstruction word — control variable
   - last five bits — information for selecting the next address.
2. The control address Register (CAR) holds the address for the control memory.
   - receives an input when its load control is enabled; otherwise, incremented by 1.
3. Microinstruction
   - Bit 1 — Initial state, $x = 1$
   - Bits 2–9 — control variables $S_2, S_1, S_0, C_{in}, L, y, z, w$
   - Bits 10–12 — an address for CAR.
   - Bits 13–14 — select an input for a multiplexer.
4. Multiplexer (MUX)
   - Four inputs — selected with Bits 13 & 14 of the microinstruction.

ROM bits		MUX select function
13	14	
0	0	Increment CAR
0	1	Load input to CAR if $S=0$
1	0	Load input to CAR if $S=1$, increment CAR if $E=0$
1	1	Load inputs to CAR if $E=1$,

Scanned with CamScanner

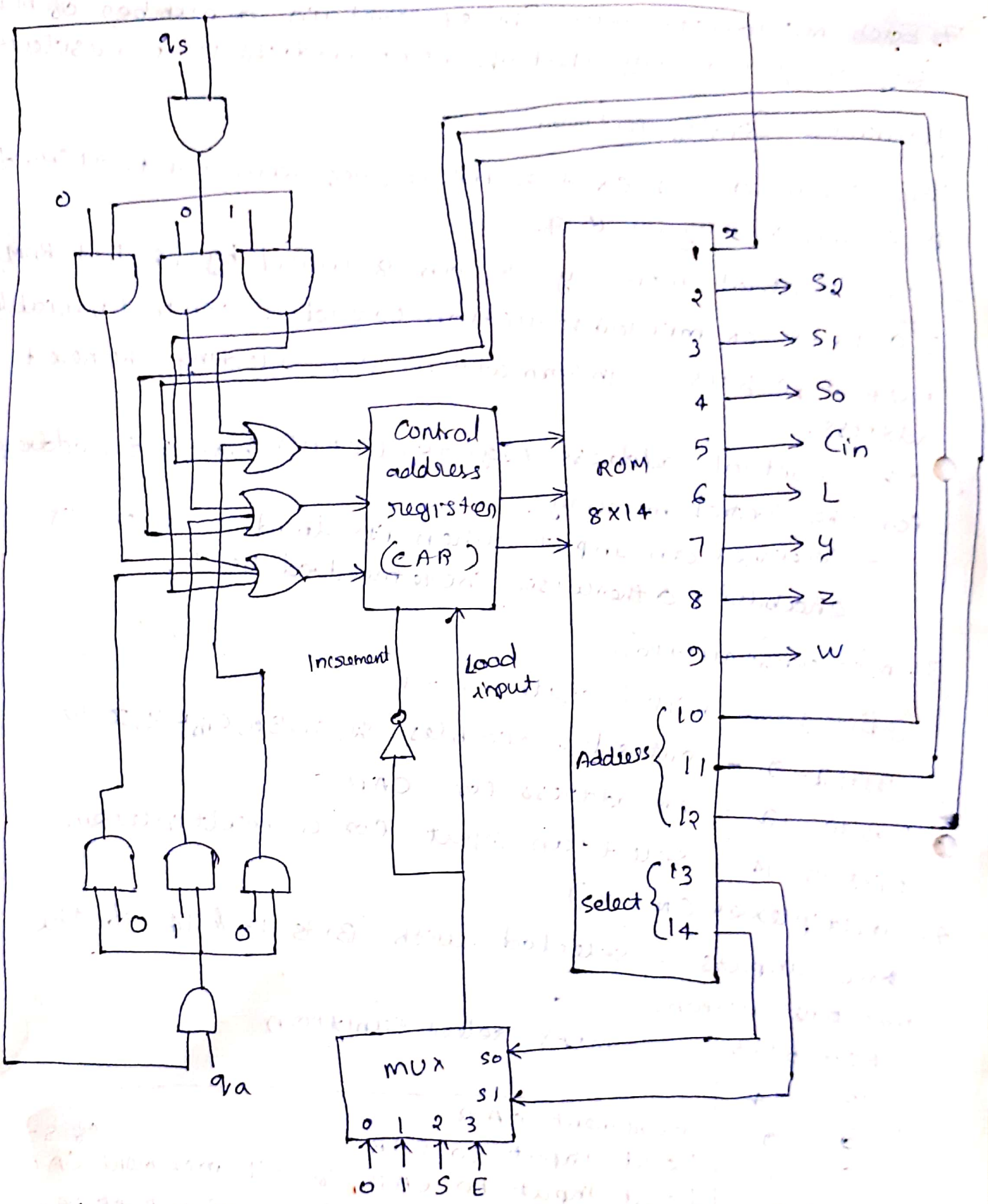


Figure 6-9 Micro program control block diagram.

Scanned with CamScanner

The Microprogram

1. Derive the microprogram for the adder-subtractor. The control memory has eight words and each word contains 14 bits.
2. To microprogram the control memory, we must determine the bit values of each of the eight words.
3. The register-transfer method can be adopted for developing a microprogram.
4. The microoperation sequence can be specified with register-transfer statements.
5. No need for control functions, the control variables are the control words stored in control memory. Instead of control function, an address with each register-transfer is specified.
6. The address associated with each symbolic statement corresponds to the address where the microinstruction is to be stored in memory.
7. The sequencing from one address to the next can be indicated by means of conditional control statements.
8. Conditional control statement can specify address to which control goes, depending on status conditions.
9. The microprogram in symbolic form is given in Table 6-2. The eight addresses of the ROM are listed in the first column. In the second column, the microinstruction that must be stored

Scanned with CamScanner

at each address is given in symbolic form.

TABLE 6-2 Symbolic microprogram for control memory

ROM address	Microinstruction	comments
0	$x=1$, if $(q_s=1)$ then (goto 1), if $(q_a=1)$ then (go to 2), if $(q_s \wedge q_a)=0$ then (go to 0)	Load 0 or external address.
1	$B_s \leftarrow \overline{B_s}$	$q_s=1$, start subtraction
2	If $(S=1)$ then (go to 4)	$q_a=1$, start addition
3	$A \leftarrow A+B$, $E \leftarrow C_{out}$, go to 0	Add magnitudes and return
4	$A \leftarrow A+\overline{B}+1$, $E \leftarrow C_{out}$,	subtract magnitudes
5	If $(E=1)$ then (go to 0) $E \leftarrow 0$	operation terminated if $E=1$
6	$A \leftarrow \overline{A}$	$E=0$, complement A
7	$A \leftarrow A+1$, $A_s \leftarrow \overline{A_s}$, go to 0	Done, return to address 0

10. The conditional control statements in the other microinstructions use the status variables S and E.

11. If there is no goto statement in the microinstruction, it implies that the next microinstruction is taken from the next address in sequence.

12. The equivalent binary form of the microprogram is given in Table 6-3.

i) the addresses for the ROM control memory

Scanned with CamScanner

are listed in binary.

ii) The first nine bits in each ROM word give the control word that initiates the specified micro-operations. From Figure 6-8(b).

iii) The last five bits in each ROM word are derived from the conditional control statements in the symbolic program.

iv) At address 000, we have 01 for the select field. This allows an external address to be loaded into CAR if q_3 or q_4 is equal to 1.

TABLE 6-3 Binary microprogram for control memory

ROM address	ROM outputs													
	x	S_2	S_1	S_0	C_{in}	L	Y	Z	W	Address			Select	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
000	1	0	0	0	0	0	0	0	0	0	0	0	0	1
001	0	0	0	0	0	0	1	0	0	0	0	0	0	1
010	0	0	0	0	0	0	0	0	0	1	0	0	1	0
011	0	0	0	1	0	1	0	0	0	0	0	0	0	1
100	0	0	1	0	1	1	0	0	0	1	0	1	0	1
101	0	0	0	0	0	0	0	0	1	0	0	0	1	1
110	0	1	1	1	0	1	0	0	0	1	1	1	0	1
111	0	0	0	0	1	1	0	1	0	0	0	0	0	1

Scanned with CamScanner

CONTROL OF PROCESSOR UNIT

The hardware configuration is expanded to include the control of an entire processor unit.

1. A general-purpose processor unit has seven registers, an ALU, a shifter, and a status register.
2. A microoperation is selected with a control word of 16 bits.
3. A microprogram organization for controlling the processor unit is shown in figure 6-10. It has

i) control memory - 64 words with 26 bits per word.

To select 64 words — need an address of 6 bits.
To select 8 status bits — 3 selection lines for the multiplexer.

To select external or the address field of the microinstruction — 1 bit

To select the microoperation in the processor — 16 bits

Total = 26 bits

ii) The processor unit is included in the diagram to show its connection to the microprogram control unit.

iii) The status bits from the processor are applied to the inputs of a multiplexer. Both the normal and complement values are used, except for the overflow bit V.

Scanned with CamScanner

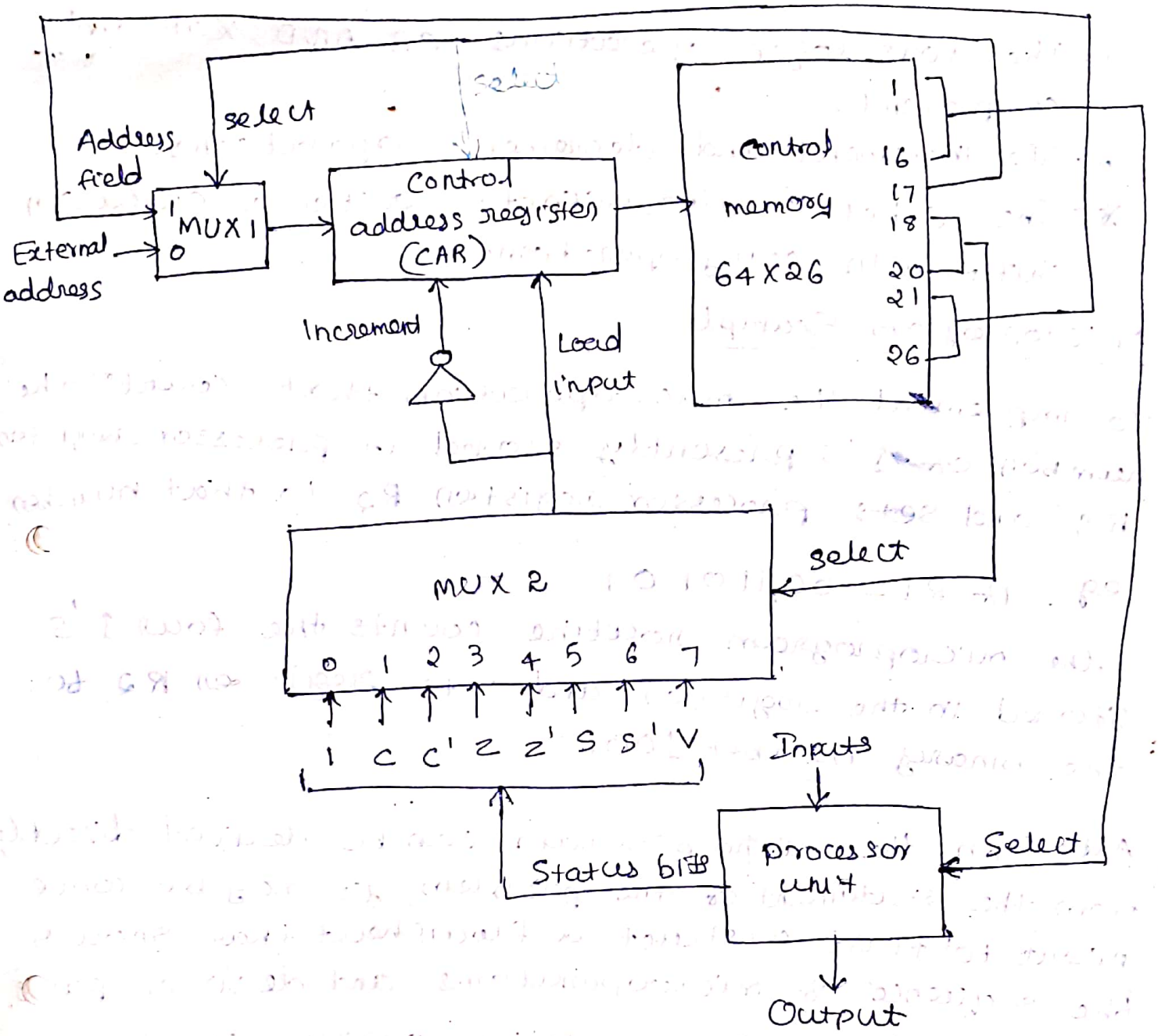


Figure 6.10 Microprogram control for processor unit

To construct correct microprograms it is necessary to specify exactly how the status bits are affected by each microoperation in the processor.

The S (sign) and Z (zero) bits are affected by all operations.

The C (carry) and V (overflow) bits do not change after the following ALU operations:

Scanned with CamScanner

1. The four logic operations OR, AND, XOR and complement.
2. The increment and decrement operations.
3. The C bit is also affected after a circular shift with carry operation.

Microprogram Example.

1) To implement the macrooperation that counts the number of 1's presently stored in processor register R1 and sets processor register R2 to that number.

eg: If R1 = 00110101

the microprogram routine counts the four 1's stored in the register and sets register R2 to the binary number 100.

2) Although the micro program can be derived directly from the statement of the problem, it may be convenient to first construct a flowchart that shows the sequence of microoperations and decision paths.

3) The flowchart for the microprogram is shown in Figure 6-11.

4) We assume that the microprogram routine starts at address 8. Register R2 and the C (carry) bit are first set to 0.

The content of R1 is then examined. If it is 0, it signifies that there are no 1's stored in it; so the microprogram routine terminates with R_2 equal to 0.

Scanned with CamScanner

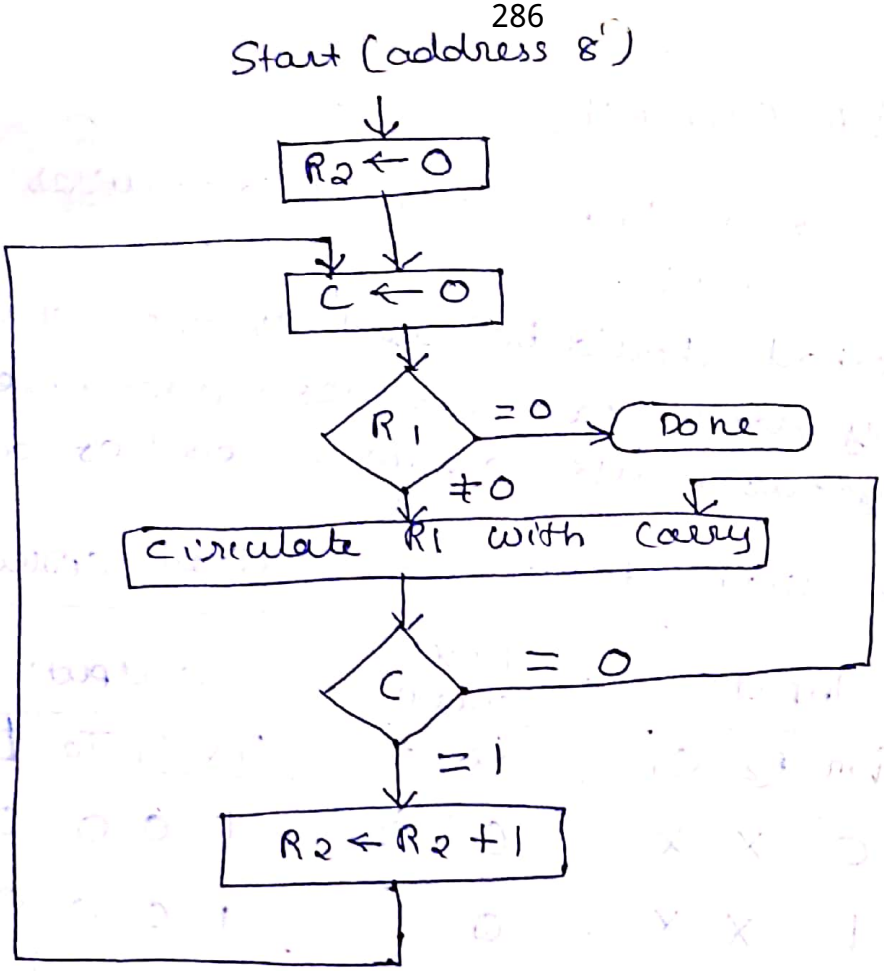


Figure 6-11 Flowchart for counting the number of 1's in register R1.

TABLE 6-4 Symbolic microprogram to count the number of 1's in R1

ROM address	Microinstruction	Comments
8	R2 ← 0	clear R2 counter
9	R1 ← R1, C ← 0	clear C, set status bits
10	If (Z=1) then (go to external address)	Done if R1 = 0
11	R1 ← crc R1	Circular R1 right with carry
12	If (C=0) then (go to 11)	Circular again if C = 0
13	R2 ← R2+1, go to 9	carry = 1, increment R2

Scanned with CamScanner

PLA CONTROL

1. The design of a PLA control requires the state table for the circuit.
2. The PLA method should be used if the state table contains many don't care entries; otherwise, it may be advantageous to use a ROM instead of a PLA.

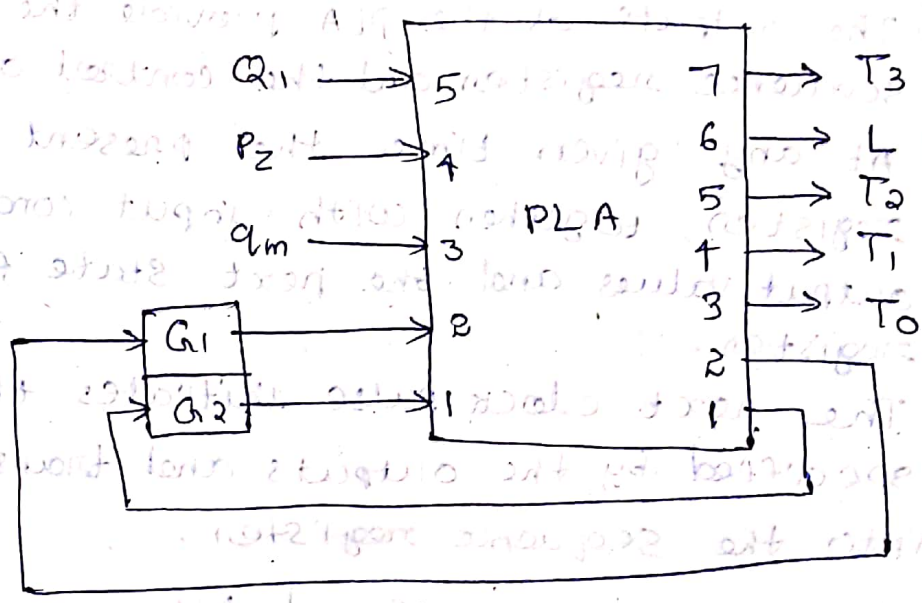
TABLE 6-5 State table for control circuit

Present state		Inputs			Next state		Outputs				
G_2	G_1	q_m	P_2	Q_1	G_2	G_1	T_0	T_1	T_2	E	T_3
0	0	0	X	X	0	0	1	0	0	0	0
0	0	1	X	X	0	1	1	0	0	0	0
0	1	X	X	X	1	0	0	1	0	0	0
1	0	X	X	0	1	1	0	0	1	0	0
1	0	X	X	1	1	1	0	0	1	1	0
1	1	X	0	X	1	0	0	0	0	0	1
1	1	X	1	X	0	0	0	0	0	0	1

3. The present state is determined from flip-flops G_1 and G_2.
4. The input variables for the control circuit are q_m, P_2, and Q_1.
5. The next state of G_1 and G_2 may be a function of one of the inputs or it may be independent of any inputs.

Scanned with CamScanner

6. If an input variable does not influence the next state, we mark it with a don't care X.
7. If the next state is a function of a particular input, the present state is repeated in the table but the next states are assigned different binary values.



a (Block diagram)

Product term	Inputs 1	2	3	4	5	Outputs 1	2	3	4	5	6	7	Comments
1	0	0	0	—	—	—	—	1	—	—	—	—	$T_0 = 1, q_m = 0$
2	0	0	1	—	—	—	1	1	—	—	—	—	$T_0 = 1, q_m = 0$
3	0	1	—	—	—	1	—	—	1	—	—	—	$T_1 = 1$
4	1	0	—	—	0	1	1	—	—	1	—	—	$T_2 = 1, Q_1 = 0$
5	1	0	—	—	1	1	1	—	—	1	1	—	$T_2 = 1, L = 1, Q_1 = 1$
6	1	1	—	0	—	1	—	—	—	—	—	1	$T_3 = 1, P_Z = 0$
7	1	1	—	1	—	—	—	—	—	—	—	1	$T_3 = 1, P_Z = 1$

(b) PLA program table

Figure 6.12 PLA control for binary multiplier.

Scanned with CamScanner

8. The block diagram of the PLA control is shown in Figure 6.12.

i) The PLA is connected to a sequence register with two flip-flops G_1 and G_2.

ii) The input to the PLA are the values of the present state of the sequence register and the three external inputs.

iii) The outputs of the PLA provide the next state for the sequence register and the control output variables.

iv) At any given time, the present state of the sequence register, together with input conditions, determines the output values and the next state for the sequence register.

v) The next clock pulse initiates the microoperations specified by the outputs and transfers the next state into the sequence register.

9. A PLA is specified by
- the number of inputs,
- the number of product terms and
- the number of outputs.

The number of product terms is a function of the circuit we wish to implement.

10. The PLA program table can be obtained directly from the state table without the need for simplification procedures.

11. The translation from the state table to a PLA program table is very simple: The x's in the input columns and the 0's in the output columns are changed to dashes, and all other entries remain the same.

Scanned with CamScanner

The procedure for designing control logic with a PLA.

1. Obtain a state table for the controller.
2. The number of states determines the number of flip-flops for the sequence register.
3. The PLA is then connected to the sequence register and to the input and output variables.
4. The PLA program table is obtained directly from the state table.

MICROPROGRAM SEQUENCER

A microprogram control unit has two parts:

1. The control memory
   - stores the micro instructions.
2. Associated circuits that control the generation of the next address.

The address-generation part is the microprogram sequencer.

A typical sequencer provides the following address-sequencing capabilities:

1. Increments the present address for control memory.
2. Branches to an address as specified by the address field of the micro-instruction.
3. Branches to a given address if a specified status bit is equal to 1.
4. Transfers control to a new address as specified

Scanned with CamScanner

by an external source.

5. Has a facility for subroutine calls and returns.

I The sequencer transfers a new address for control memory to start executing a new macrooperation. The external address transfers control to the first microinstruction in a microprogram routine that executes the specified macrooperation.

II Subroutines are programs used by other routines to accomplish a particular task.

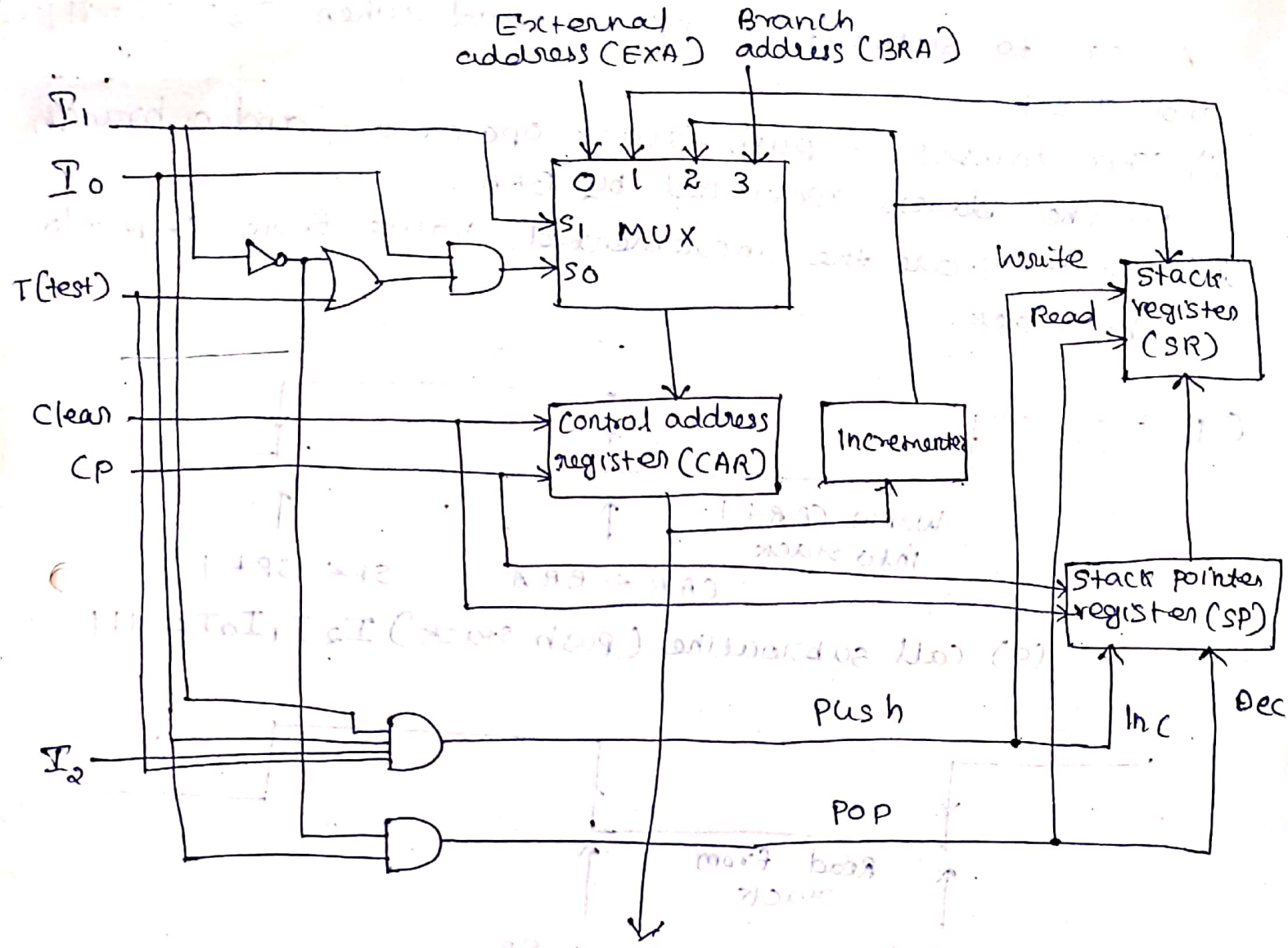
III Return address can place into a special register and then branch.

IV Organisation of register files can be done with Stack.

V Figure 6.13 shows the block diagram of a microprogram sequencer.

1. It consists of a multiplexer that selects an address from four sources and routes it into a control address register.
2. Inputs I_0, I_1 and I_2 specify the operation for the sequencer.
3. T is a test point for a status bit.
4. The control address registers can be cleared to zero to initialize the system and the clock pulses synchronize the loading into the registers.
5. The function table listed in the diagram specifies the operation of the sequencer.

Scanned with CamScanner



Function table

I_2	I_1	I_0	T	S1	S0	Operation	Comments
X	0	0	X	0	0	CAR ← EXA	Transfer external address
X	0	1	X	0	1	CAR ← SR	Transfer from register stack
X	1	0	X	1	0	CAR ← CAR + 1	Increment address
0	1	1	0	1	0	CAR ← CAR + 1	Increment address
0	1	1	1	1	1	CAR ← BRA	Transfer branch address
1	1	1	0	1	0	CAR ← CAR + 1	Increment address
1	1	1	1	1	1	CAR ← BRA, SR ← CAR+1	Branch to subroutine

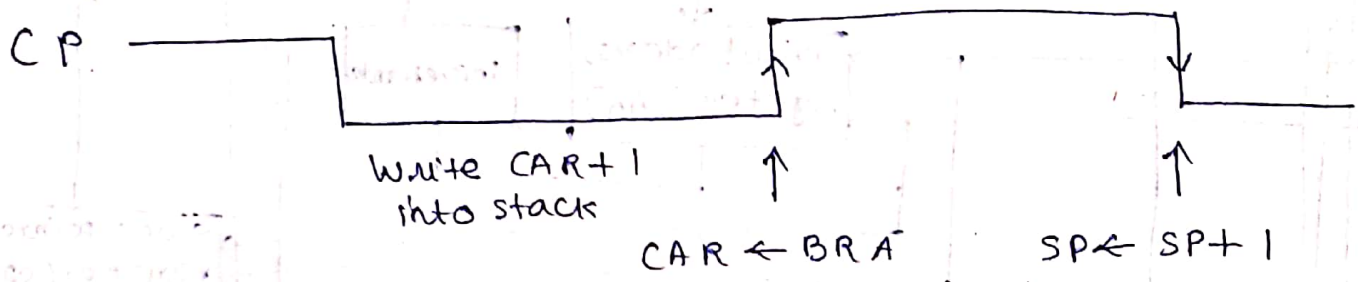
Figure 6.13 Typical microprogram sequencer organization.

Scanned with CamScanner

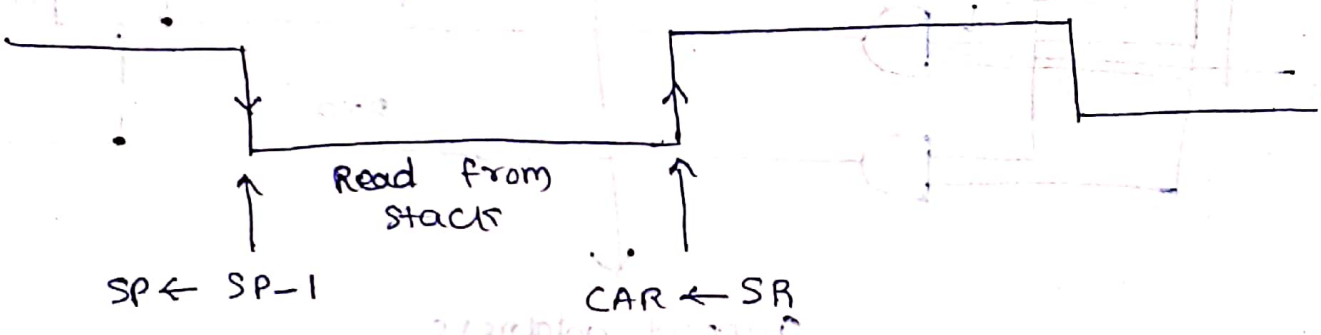
A call to subroutine is executed when $I_2 I_1 I_0 = 111$ and $T = 1$.

i) This causes a push-stack operation and a branch to the address specified by BRA.

ii) first store the incremented value from CAR into the stack.



(a) call subroutine (push stack) $I_2 I_1 I_0 T = 1111$



(b) Return from subroutine (POP stack) $I_1 I_0 = 01$

Figure 6.14 Stack operations in microprogram sequencer.

iii) When clock pulse CP goes through a positive-edge transition, the BRA address is transferred to CAR and the write input to the stack is inhibited.

iv) The stack-pointer register is incremented later when CP goes through its negative-edge transition.

Scanned with CamScanner

The return from subroutine is executed when $I, I_0 = 01$.

i) This causes a pop-stack operation and a branch to the address stored on top of the stack.

ii) This is implemented by first decrementing the stack-pointer register on the negative-edge transition of CP.

iii) The value in the stack, given by the address presently available in the stack pointer, is then read and transferred to CAR on the positive-edge transition of CP.

iv) The stack pointer is incremented after the transfer into CAR and decremented prior to the transfer into CAR.

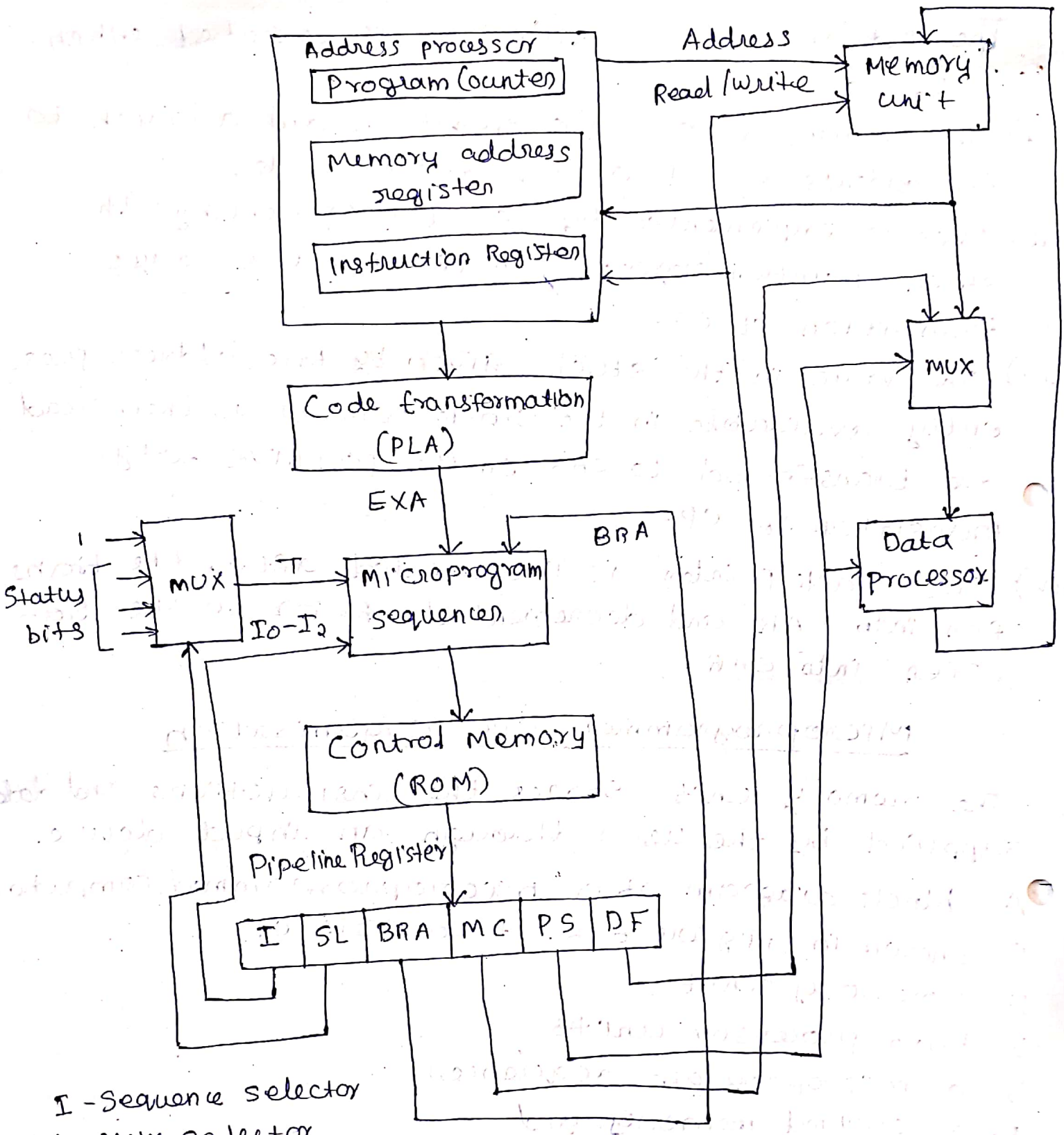
Microprogrammed CPU Organization

The memory unit stores the instructions and data supplied by the user through an input device.

A block diagram of a microprogrammed computer is shown in Figure 6.15. It consists of

1. a memory unit,
2. two processor units,
3. a microprogram sequencer,
4. a control memory and
5. few other digital functions.

Scanned with CamScanner



I - Sequence selector
SL - MUX selector
BRA - Branch address
MC - Memory control
PS - Processor selector
DF - Data Field

Figure 6.15 Microprammed computer organization.

Scanned with CamScanner

CPU = processor section + microprogram sequencer.

1. The data processor manipulates the data, and the address processor manipulates the address information received from memory.
2. An instruction extracted from memory during the fetch cycle goes into the instruction register.
3. The instruction-code bits in the instruction register specify a macrooperation for control memory.
4. A code transformation is sometimes needed to convert the operation-code bits of an instruction into a starting address for the control memory.
5. The address generated in the code transformation mapping function is applied to the external address (EXA) input of the sequencer.

The microprogram control unit consists of the
1. sequencer
2. a control memory (Store microinstructions)
3. a multiplexer
4. A pipeline register.

A possible microinstruction format for the control memory is illustrated within the pipeline register.

- I - 3 bits - input information for the sequencer
- SL - Status bit selects for the multiplexer.
- BRA - supplies a branch address.

→ determine the next address for control memory.

Scanned with CamScanner

The other three fields in the microinstruction are for controlling the microoperations in the processor and memory units.

MC - Memory control - controls the address processor and read write operations in the memory unit.

PS - processor select - controls the operations in the data processor unit.

DF - Data field used to introduce constants into the processor.

The construction of a CPU from LSI components as shown in Figure 6.15 provides the freedom to define the instruction set for a computer system. It must be realized, however, that integrated circuits are available that contain a complete CPU within

Microinstructions

Microoperations

The operations performed on the data stored in registers are called microoperations.

Control unit

The purpose of the control unit is to initiate a series of sequential steps of microoperations. During any given time, certain operations are to be initiated while all others remain idle.

Control word

The control variables at any given time can be represented by a string of 1's and 0's called a control word.

Scanned with CamScanner

Microprogrammed control unit

A control unit whose control variables are stored in a memory is called a microprogrammed control unit.

Microinstruction

Each control word of memories is called a microinstruction.

Microprogram

A sequence of microinstructions is called a microprogram.

Since alteration of the microprogram is seldom needed, the control memory can be a ROM.

Implementation of micro-programed control.

1. All the control unit does is generate a set of control signals.
2. Each control signal is on or off
3. Represent each control signal by a bit
4. Have a control word for each micro-operation
5. Have a sequence of control words for each machine code instruction.
6. Add an address to specify the next microinstruction depending on conditions
7. Today's large microprocessor have many instructions and associated register-level hardware and many control points to be manipulated. So need a control memory that contains a large number of words.

Scanned with CamScanner

Micro-instruction Types

There are two types

1. Horizontal microinstructions,
2. Vertical microinstructions.

Horizontal Microinstruction

Each micro-instruction specifies many different micro-operations to be performed in parallel

1. Wide memory word
2. High degree of parallel operations possible
3. Little encoding of control information.

Format:

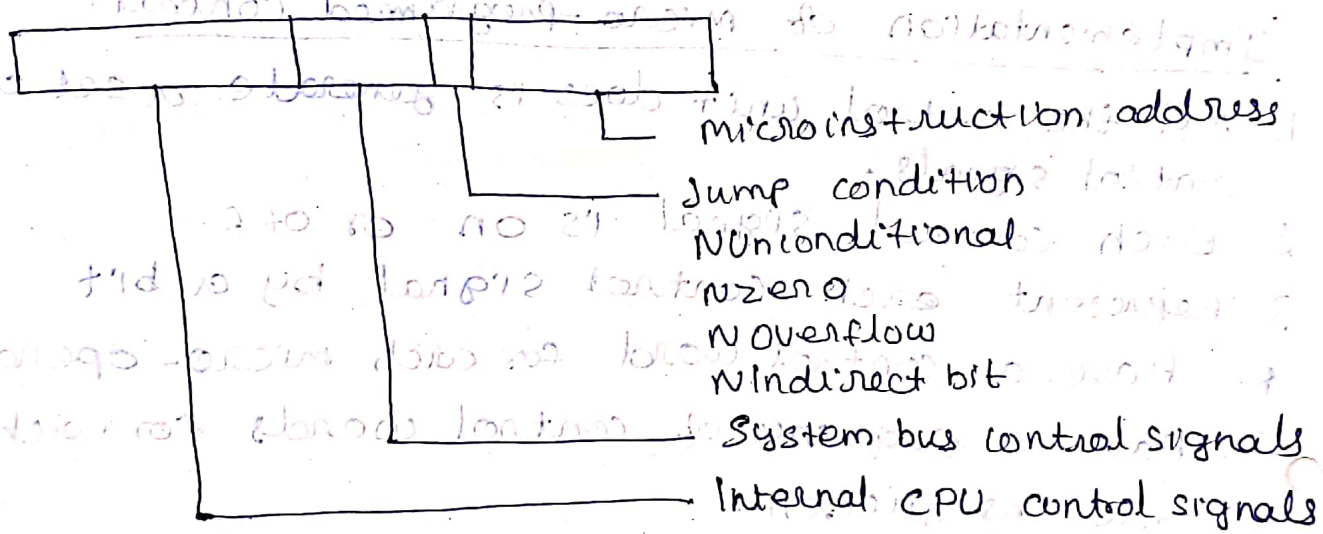


Figure 6.16 Horizontal microinstruction

Vertical Microinstruction

Each microinstruction specifies single (or few) microoperations to be performed.

1. Width of control word is narrow
2. n control signals encoded into $\log_2 n$ bits.
3. Limited ability to express parallelism.

Scanned with CamScanner

4. Considerable encoding of control information requires external memory word decoder to identify the exact control line ~~bre~~ being manipulated.

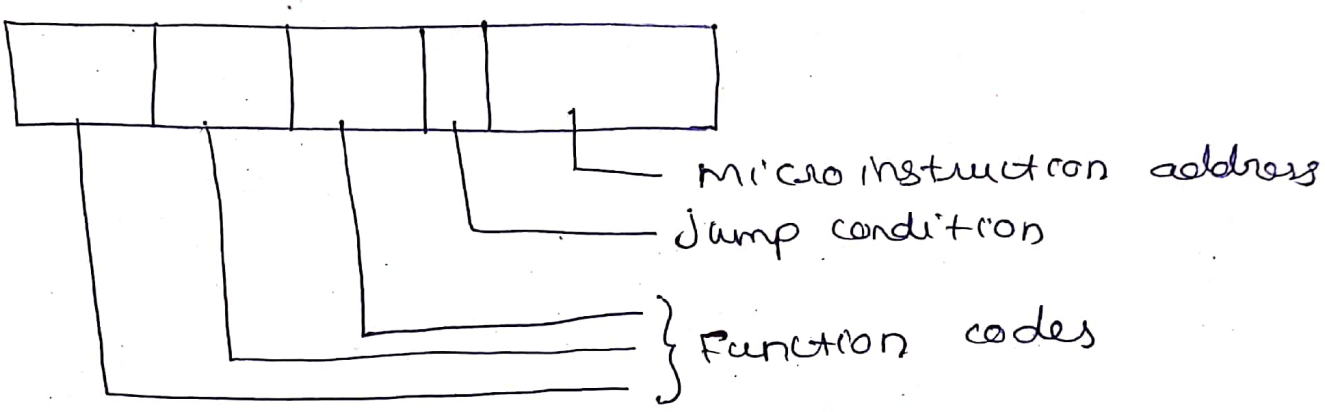


Figure 6.17 Vertical microinstruction.

Scanned with CamScanner

APPENDIX 1

CONTENT BEYOND THE SYLLABUS

1. Graphene Processors

Graphene is a material made of strongly bonded lattices of carbon atoms (similar to carbon nanotubes). It has a number of remarkable properties, including immense physical strength and near-superconductivity. There are dozens of potential applications for graphene, from space elevators to body armor to better batteries, but the one that's relevant to this article is their potential role in computer architectures.

Another way of making computers faster, rather than shrinking transistor size, is to simply make those transistors run faster. Unfortunately, because silicon isn't a very good conductor, a significant amount of the power sent through the processor winds up converted to heat. If you try to clock silicon processors up much above nine gigahertz, the heat interferes with the operation of the processor. The 9 gigahertz requires extraordinary cooling efforts (in some cases involving liquid nitrogen). Most consumer chips run much more slowly.

Graphene, in contrast, is an excellent conductor. A graphene transistor can, in theory, run up to 500 GHz without any heat problems to speak of — and, you can etch it the same way you etch silicon. IBM has engraved simple analog graphene chips already, using traditional chip lithography techniques. Until recently, the issue has been two fold: first, that it's very difficult to manufacture graphene in large quantities, and, second, that we do not have a good way to create graphene transistors that entirely block the flow of current in their 'off' state.

The first problem was solved when electronics giant Samsung announced that its research arm had discovered a way to mass produce whole graphene crystals with high purity. The second problem is more complicated. The issue is that, while graphene's extreme conductivity makes it attractive from a heat perspective, it's also annoying when you want to make transistors – devices that are intended to stop conducting billions of times a second. Graphene, unlike silicon, lacks a 'band gap' — a rate of current flow that is so low that it causes the material to drop to zero conductivity. Luckily, it looks like there are a few options on that front.

Samsung has developed a transistor that uses the properties of a silicon-graphene interface in order to produce the desired properties, and built a number of basic logic circuits with it. While not a pure graphene computer, this scheme would preserve many of the beneficial effects of graphene. Another option may be the use of 'negative resistance' to build a different kind of transistor that could be used to construct logic gates that operate at higher power, but with fewer elements.

Of the technologies discussed in this article, graphene is the farthest away from commercial reality. It could take up to a decade for the technology to be mature enough to really replace silicon entirely. However, in the long term, it's very likely that graphene (or a variant of the material) will be the backbone of the computing platform of the future.

2. Memristor Storage

A different approach to solving the memory problem is designing computer memory that has the advantage of more than one kind of memory. Generally, the tradeoffs with memory boil down to cost, access speed, and volatility (volatility is the property of needing a constant supply of power to keep data stored). Hard drives are very slow, but cheap and non-volatile.

Ram is volatile, but fast and cheap. Cache and registers are volatile and very expensive, but also very fast. The best-of-both-worlds technology is one that's non-volatile, fast to access, and cheap to create. In theory, memristors offer a way to do that.

Memristors are similar to resistors (devices that reduce the flow of current through a circuit), with the catch that they have memory. Run current through them one way, and their resistance increases. Run current through the other way, and their resistance decreases. The result is that you can build inexpensive, high-speed RAM-style memory cells that are nonvolatile, and can be manufactured cheaply.

This raises the possibility of RAM blocks as large as hard drives that store the entire OS and file system of the computer (like a huge, non-volatile RAM disk), all of which can be accessed at the speed of RAM. No more hard drive. No more walking to the moon.

HP has designed a computer using memristor technology and specialized core design, which uses photonics (light based communication) to speed up networking between computational elements. This device (called "The Machine") is capable of doing complex processing on hundreds of terrabytes of data in a fraction of a second. The memristor memory is 64-128 times denser than conventional RAM, which means that the physical footprint of the device is very small — and, the entire shebang uses far less

power than the server rooms it would be replacing. HP hopes to bring computers based on The Machine to market in the next two to three years.

3. Micron Hybrid Memory Cube

One of the principle bottlenecks for current computer design is the time it takes to fetch the data from memory that the processor needs to work on. The time needed to talk to the ultra-fast registers inside a processor is considerably shorter than the time needed to fetch data from RAM, which is in turn vastly faster than fetching data from the ponderous, plodding hard drive.

The result is that, frequently, the processor is left simply waiting for long stretches of time for data to arrive so it can do the next round of computations. Processor cache memory is about ten times faster than RAM, and RAM is about one hundred thousand times faster than the hard drive. Put another way, if talking to the processor cache is like walking to the neighbor's house to get some information, then talking to the RAM is like walking a couple of miles to the store for the same information — getting it from the hard drive is like walking to the moon.

Micron Technology may break the industry from the regular progression of conventional DDR memory technology, replacing it with their own technology, which stacks RAM modules into cubes and uses higher-bandwidth cables to make it faster to talk to those cubes. The cubes are built directly onto the motherboard next to the processor (rather than inserted into slots like convention ram). The hybrid memory cube architecture offers five times more bandwidth to the processor than the DDR4 ram coming out this year, and uses 70% less power. The technology is expected to hit the supercomputer market early next year, and the consumer market a few years later.

4. Neuromorphic Chips

As the electronics market moves toward smarter technologies that adapt to users and automate more intellectual grunt work, many of the problems that computers need to solve are centered around machine learning and optimization. One powerful technology used to solve such problems are 'neural networks.'

Neural networks reflect the structure of the brain: they have nodes that represent neurons, and weighted connections between those nodes that represent synapses. Information flows through the network, manipulated by the weights, in order to solve problems. Simple rules dictate how the weights between neurons change, and these changes can be exploited to produce learning and intelligent behavior. This sort of learning is computationally expensive when simulated by a conventional computer.

Neuromorphic chips attempt to address this by using dedicated hardware specifically designed to simulate the behavior and training of neurons. In this way, an enormous speedup can be achieved, while using neurons that behave more like the real neurons in the brain.

IEEE

"The Computer Society's predictions, based on an in-depth analysis by a team of leading technology experts, identify top technologies that have substantial potential to disrupt the market in the year 2019," said Hironori Kasahara, IEEE Computer Society President. "The technical community depends on the Computer Society as the source of technology IP, trends, and information. IEEE-CS predictions represent our commitment to keeping our community prepared for the technological landscape of the future."

Dejan Milojicic, Hewlett Packard Enterprise Distinguished Technologist and IEEE Computer Society past president (2014), said, "In 2019 we expect to see ever-increasing adoption of deep learning accelerators in the areas of transportation, advanced security, and technologies for humanity. Fueled by advanced materials, adoption of virtual reality and the Internet of Bodies will stretch the future to new unknowns. We are excited about our predictions and the bets we have made for 2019's technology trends."

5. The top 10 technology trends

1. **Deep learning accelerators** such as GPUs, FPGAs, and more recently TPUs. More companies have been announcing plans to design their own accelerators, which are widely used in data centers. There is also an opportunity to deploy them at the edge, initially for inference and for limited training over time. This also includes accelerators for very low power devices. The development of these technologies will allow machine learning (or smart devices) to be used in many IoT devices and appliances.

2. **Assisted transportation**. While the vision of fully autonomous, self-driving vehicles might still be a few years away, increasingly automated assistance is taking place in both personal and municipal (dedicated) vehicles. Assisted transportation is already very useful in terms of wide recognition and is paving the way for fully

autonomous vehicles. This technology is highly dependent on deep learning accelerators (see #1) for video recognition.

3. **The Internet of Bodies (IoB)**. IoT and self-monitoring technologies are moving closer to and even inside the human body. Consumers are comfortable with self-tracking using external devices (such as fitness trackers and smart glasses) and with playing games using augmented reality devices. Digital pills are entering mainstream medicine, and body-attached, implantable, and embedded IoB devices are also beginning to interact with sensors in the environment. These devices yield richer data that enable more interesting and useful applications, but also raise concerns about security, privacy, physical harm, and abuse.

4. **Social credit algorithms**. These algorithms use facial recognition and other advanced biometrics to identify a person and retrieve data about that person from social media and other digital profiles for the purpose of approval or denial of access to consumer products or social services. In our increasingly networked world, the combination of biometrics and blended social data streams can turn a brief observation into a judgment of whether a person is a good or bad risk or worthy of public social sanction. Some countries are reportedly already using social credit algorithms to assess loyalty to the state.

5. **Advanced (smart) materials and devices**. We believe novel and advanced materials and devices for sensors, actuators, and wireless communications, such as tunable glass, smart paper, and ingestible transmitters, will create an explosion of exciting applications in healthcare, packaging, appliances, and more. These technologies will also advance pervasive, ubiquitous, and immersive computing, such as the recent announcement of a cellular phone with a foldable screen. The use of such technologies will have a large impact in the way we perceive IoT devices and will lead to new usage models.

6. **Active security protection**. The traditional method of protecting computer systems involves the deployment of prevention mechanisms, such as anti-virus software. As attackers become more sophisticated, the effectiveness of protection mechanisms decreases as the cost increases. However, a new generation of security mechanisms is emerging that uses an active approach, such as hooks that can be activated when new types of attacks are exposed and machine-learning mechanisms to identify sophisticated attacks. Attacking the attacker is a technological possibility as well, but is almost always illegal.

7. **Virtual reality (VR) and augmented reality (AR)**. These related technologies have been hitting the mainstream in some respects for a number of years. For a well-

known example, Pokemon Go is a game that uses the camera of a smartphone to interpose fictional objects in real-world surroundings. Gaming is clearly a driver of these technologies, with other consumer devices becoming affordable and commonplace. VR and AR technologies are also useful for education, engineering, and other fields. However, there has been a Catch-22 in that there is a lack of applications resulting from the high cost of entry, yet the cost has stayed high due to a lack of applications. With advertisements for VR headsets appearing during prime-time television programs, we may have finally reached a tipping point.

8. **Chatbots.** These artificial intelligence (AI) programs simulate interactive human conversation using key pre-calculated user phrases and auditory or text-based signals. Chatbots have recently started to use self-created sentences in lieu of pre-calculated user phrases, providing better results. Chatbots are frequently used for basic customer service on social networking hubs and are often included in operating systems as intelligent virtual assistants. We have recently witnessed the use of chatbots as personal assistants capable of machine-to-machine communications as well. In fact, chatbots mimic humans so well that some countries are considering requiring chatbots to disclose that they are not human. Industry is looking to expand chatbot applications to interaction with cognitive-impaired children as a way to provide therapeutic support.

9. **Automated voice spam (robocall) prevention**. Spam phone calls are an ongoing problem of increasing sophistication, such as spoofing the caller ID number of the victim's family and business associates. This is leading people to regularly ignore phone calls, creating risks such as true emergency calls going unanswered. However, emerging technology can now block spoofed caller ID and intercept questionable calls so the computer can ask questions of the caller to assess whether he or she is legitimate.

10. **Technology for humanity (specifically machine learning)**. We are approaching the point where technology can help resolve societal issues. We predict that large-scale use of machine learning, robots, and drones will help improve agriculture, ease drought, ensure supply of food, and improve health in remote areas. Some of these activities have already started, but we predict an increase in adoption rate and the reporting of success stories in the next year. "Sensors everywhere" and advances in IoT and edge computing are major factors contributing to the adoption of this technology. Recent events, such as major fires and bridge collapses, are further accelerating the urgency to adopt monitoring technologies in fields like forests and smart roads.