23A04504A- Computer Architecture & Organization

III B.Tech I Sem (E.C.E) **L – T – P – C**

3 – 0 – 0 – 3

COMPUTER ARCHITECTURE & ORGANIZATION

Course Objectives:

1. To learn the design of various functional units of digital computers and performance issues of computer systems.
2. To understand the basic processing unit and their connections.
3. To get familiar with different types of Data representation and Computer Arithmetic operations.
4. To know about different types of memory and their interconnections.
5. To learn the basics of parallel computing and pipelining.

Course Outcomes:

At the end of this course, the students will be able to

1. Learn the design of various functional units of digital computers and performance issues of computer systems.
2. Understand the basic processing unit and their connections.
3. Know about different types of Data representation and Computer Arithmetic operations.
4. Learn about different types of memory and their interconnections.
5. Understand the basics of parallel computing and pipelining.

UNIT I

Digital Computers: Introduction, Block diagram of Digital Computer, Definition of Computer Organization, Computer Design and Computer Architecture.

Register Transfer Language and Micro operations: Register Transfer language, Register Transfer, Bus and memory transfers, Arithmetic Micro operations, logic micro operations, shift micro operations, Arithmetic logic shift unit.

Basic Computer Organization and Design: Instruction codes, Computer Registers Computer instructions, Timing and Control, Instruction cycle, Memory Reference Instructions, Input – Output and Interrupt.

UNIT II

Micro programmed Control: Control memory, Address sequencing, micro program example, design of control unit.

Central Processing Unit: General Register Organization, Instruction Formats, Addressing modes, Data Transfer and Manipulation, Program Control.

UNIT III

Data Representation: Data types, Complements, Fixed Point Representation, Floating Point Representation.

Computer Arithmetic: Addition and subtraction, multiplication Algorithms, Division Algorithms, Floating – point Arithmetic operations. Decimal Arithmetic unit, Decimal Arithmetic operations.

UNIT IV

Input-Output Organization: Input-Output Interface, Asynchronous data transfer, Modes of Transfer, Priority Interrupt Direct memory Access.

Memory Organization: Memory Hierarchy, Main Memory, Auxiliary memory, Associate Memory, Cache Memory.

UNIT V

Reduced Instruction Set Computer: CISC Characteristics, RISC Characteristics. Pipeline and Vector Processing: Parallel Processing, Pipelining, Arithmetic Pipeline, Instruction Pipeline, RISC Pipeline, Vector Processing, Array Processor. Multi Processors: Characteristics of Multiprocessors, Interconnection Structures, Inter-processor arbitration, Inter-processor communication and synchronization, Cache Coherence.

Textbook:

1. Computer System Architecture – M. Moris Mano, Third Edition, Pearson/PHI.

References:

1. Computer Organization – Car Hamacher, ZvonksVranesic, SafeaZaky, Vth Edition, McGraw Hill.
2. Computer Organization and Architecture – William Stallings Sixth Edition, Pearson/PHI.

Structured Computer Organization – Andrew S. Tanenbaum, 4th Edition, PHI

UNIT I

Digital Computers: Introduction, Block diagram of Digital Computer, Definition of Computer Organization, Computer Design and Computer Architecture.

Register Transfer Language and Micro operations: Register Transfer language, Register Transfer, Bus and memory transfers, Arithmetic Micro operations, logic micro operations, shift micro operations, Arithmetic logic shift unit.

Basic Computer Organization and Design: Instruction codes, Computer Registers Computer instructions, Timing and Control, Instruction cycle, Memory Reference Instructions, Input – Output and Interrupt.

INTRODUCTION TO DIGITAL COMPUTERS

Digital Computers:

- Introduction,
- Block diagram of Digital Computer,
- Definition of Computer Organization.

Digital Computers:

It is a digital system that performs various computational tasks.

First electronic digital computers introduced in the year 1940's were primarily used for the numerical computations.

Digital computer uses the binary number system, which has two digits, 0

& 1. A Binary digit is called a bit.

In computers, information is represented in **'Group of bits'.**

Computer System:

A computer system is subdivided into 2 functional units:

1. Hardware and

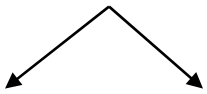
2. Software

1. Hardware: Consists of electronic components and electromechanical devices that comprise the physical entity of the system.

2. Software: Consists of instructions and data that the computer manipulates to perform various data processing tasks.

Program: It is a sequence of instructions for the computer

Software



Application software System software

System Software:

Consists of collection of programs whose purpose is to make more effective use of computer.

The programs included in the system software are referred to as **operating system.** The system software is an indispensable part of a computer.

Application Software:

It is software **that performs specific tasks for an end-user**.

For example, A High level language program written by user to solve particular data processing needs is an **Application program.**

A compiler that is used to translate high level language to machine language is a **System program.**

Block Diagram of a Digital Computer:

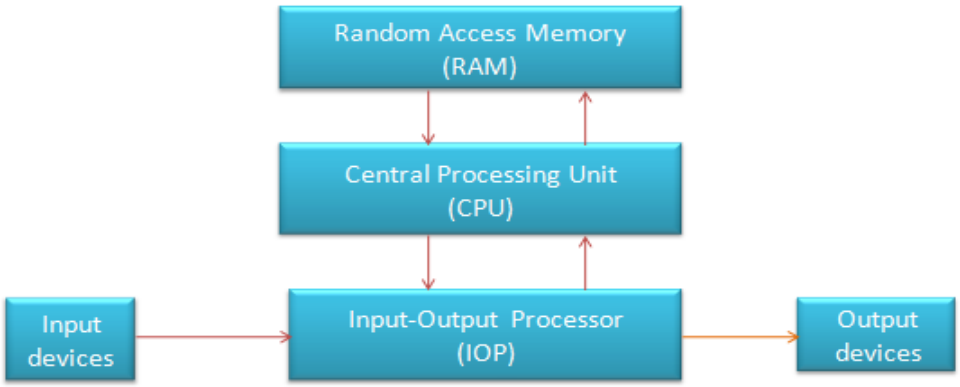


Fig1.2: Block Diagram of a Digital Computer

A digital computer consists of five functionally independent parts.

1. **CPU:** Contains an Arithmetic and logical unit for manipulating data, a number of registers for storing data, and control circuit for fetching and executing instructions.
2. **RAM:** Contains storage for instructions and data .Here, the CPU can access any location at random and retrieve the binary information within the fixed interval of time.
3. **IOP:** Contains electronic circuits for communicating and controlling the transfer of information between the computer and the outside the world.
4. **Input Devices:** Computers accept coded information through input units, which reads the data. Ex: Keyboard, Mouse, joy sticks.
5. **Output Devices:** Used to produce output through output devices. Ex: Printer, Plotter, Micro film Output, Voice Output, speakers.

Definition of Computer Organization

Computer Organization is how operational parts of a computer system are linked together. It implements the provided computer architecture. Computer organization deals with 'How to do?'

Computer Architecture and Organization

Computer Architecture and Organization is the study of internal working, structuring, and implementation of a computer system. Architecture in the computer system, same as anywhere else, refers to the externally visual attributes of the system

- Computer architecture explains **what a computer should do**.
- Computer organization explains **how a computer works**.

Computer Architecture

Computer Architecture is a blueprint for design and implementation of a computer system. It provides the functional details and behavior of a computer system and comes before computer organization. Computer architecture deals with 'What to do?'

Computer Organization

Computer Organization is how operational parts of a computer system are linked together. It implements the provided computer architecture. Computer organization deals with 'How to do?'

Following are some of the important differences between Computer Architecture and Computer Organization:

Sr. No.	Key	Computer Architecture	Computer Organization
1	Purpose	Computer architecture explains what a computer should do.	Computer organization explains how a computer works.
2	Target	Computer architecture provides functional behavior of computer system.	Computer organization provides structural relationships between parts of computer system.
3	Design	Computer architecture deals with high level design.	Computer organization deals with low level design.
4	Actors	Actors in Computer architecture are hardware parts.	Actor in computer organizaton is performance.
5	Order	Computer architecture is designed first.	Computer organization is started after finalizing computer architecture.

WHY STUDY COMPUTER ORGANIZATION?

It gives an insight of how a computer executes programs internally and can help programmer to write more effective programs.

For system programmers, a good knowledge of Computer Organization is essential because they need to program the bare hardware without the support of an operating system.

Relation between Computer Architecture, Organization

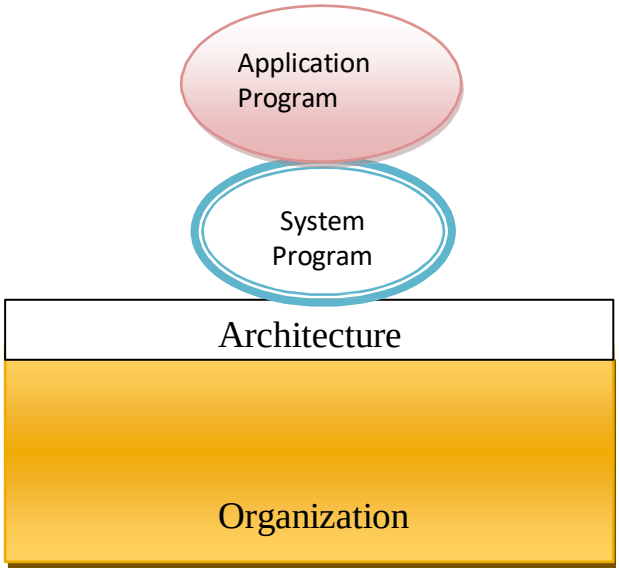


Fig1.1 Organization Implements Architecture

Computer Architecture:

It gives the external view of the computer. It is concerned with the **structure and behavior** of the computer.

An Assembly level programmer needs to be aware of ***Specific Instruction supported by the processor, the instruction formats, the specific registers, and their roles, the way to perform input or output data.***

Computer Organization:

CO is concerned with the way the hardware components operate and the way they are connected together to form the computer system.

The various components are assumed to be in place and the task is to investigate the organizational structure to verify that computer parts operate as intended.

CO gives an internal view of a computer and the roles that internal components play during execution of a program.

In other words, CO deals with how different parts of the computer such as the processor, memory, and peripheral devices are interconnected and the roles that internal components play during program execution.

Figure 1.1 shows that System program (operating system) directly interacts with the Computer Hardware.

The Application program invokes the services offered by the System programs.

Application programs are independent of the Architecture (High level Language) and are converted to machine dependent programs through a system program.

The internal organization of a basic computer is defined by its **internal registers, the timing and the control structure, & the set of instructions that it uses.**

The internal organization of a digital system is defined by sequence of micro operations it performs on data stored in registers.

Register Transfer Language and Micro operations: Register Transfer language, Register Transfer, Bus and memory transfers, Arithmetic Micro operations, logic micro operations, shift micro operations, Arithmetic logic shift unit.

Register Transfer Language

Digital systems are composed of modules that are constructed from digital components, such as registers, decoders, arithmetic elements, and control logic

The modules are interconnected with common data and control paths to form a digital computer system

The operations executed on data stored in registers are called micro operations • A micro operation is an elementary operation performed on the information stored in one or more registers

Examples are shift, count, clear, and load

Some of the digital components from before are registers that implement micro operations

The internal hardware organization of a digital computer is best defined by specifying

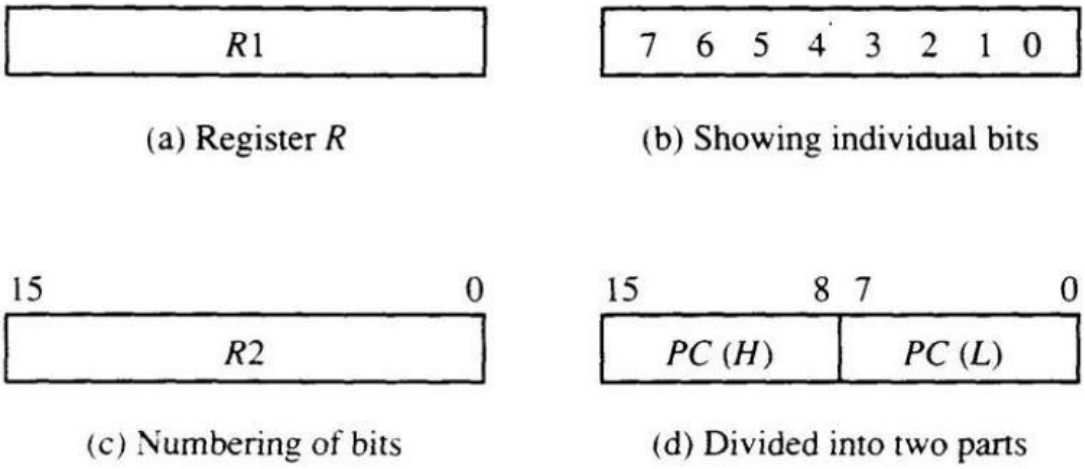
- The set of registers it contains and their functions
- The sequence of micro operations performed on the binary information stored
- The control that initiates the sequence of micro operations

Use symbols, rather than words, to specify the sequence of micro operations

- The symbolic notation used is called a register transfer language
  - A programming language is a procedure for writing symbols to specify a given computational process
  - Define symbols for various types of micro operations and describe associated hardware that can implement the micro operations

Register Transfer

- Designate computer registers by capital letters to denote its function
- The register that holds an address for the memory unit is called MAR
- The program counter register is called PC
- IR is the instruction register and R1 is a processor register
- The individual flip-flops in an n-bit register are numbered in sequence from 0 to n-1
- Refer to Figure 4.1 for the different representations of a register



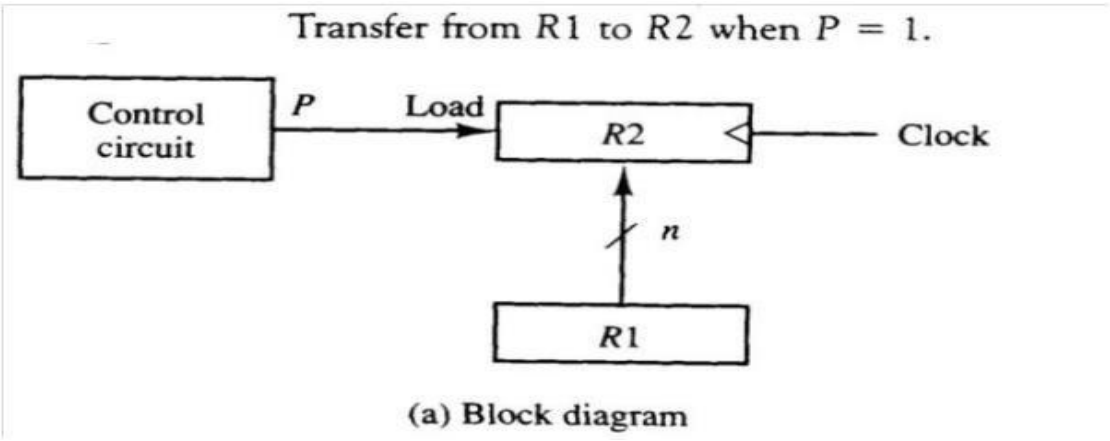
- Designate information transfer from one register to another by R2 ← R1
- This statement implies that the hardware is available
  - The outputs of the source must have a path to the inputs of the destination
  - The destination register has a parallel load capability
- If the transfer is to occur only under a predetermined control condition, designate it by

If (P = 1) *then* (R2 ← R1)

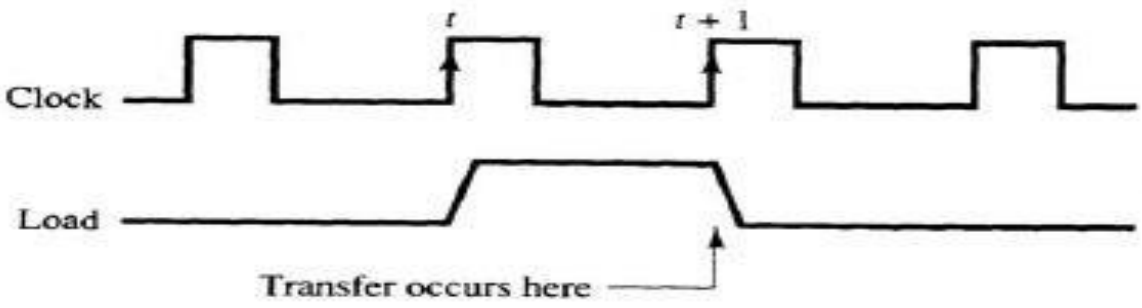
or, P: R2← R1,

Where P is a control function that can be either 0 or 1

- Every statement written in register transfer notation implies the presence of the required hardware construction



(a) Block diagram



(b) Timing diagram

• It is assumed that all transfers occur during a clock edge transition
• All micro operations written on a single line are to be executed at the same time T: R2 ← R1, R1 ← R2

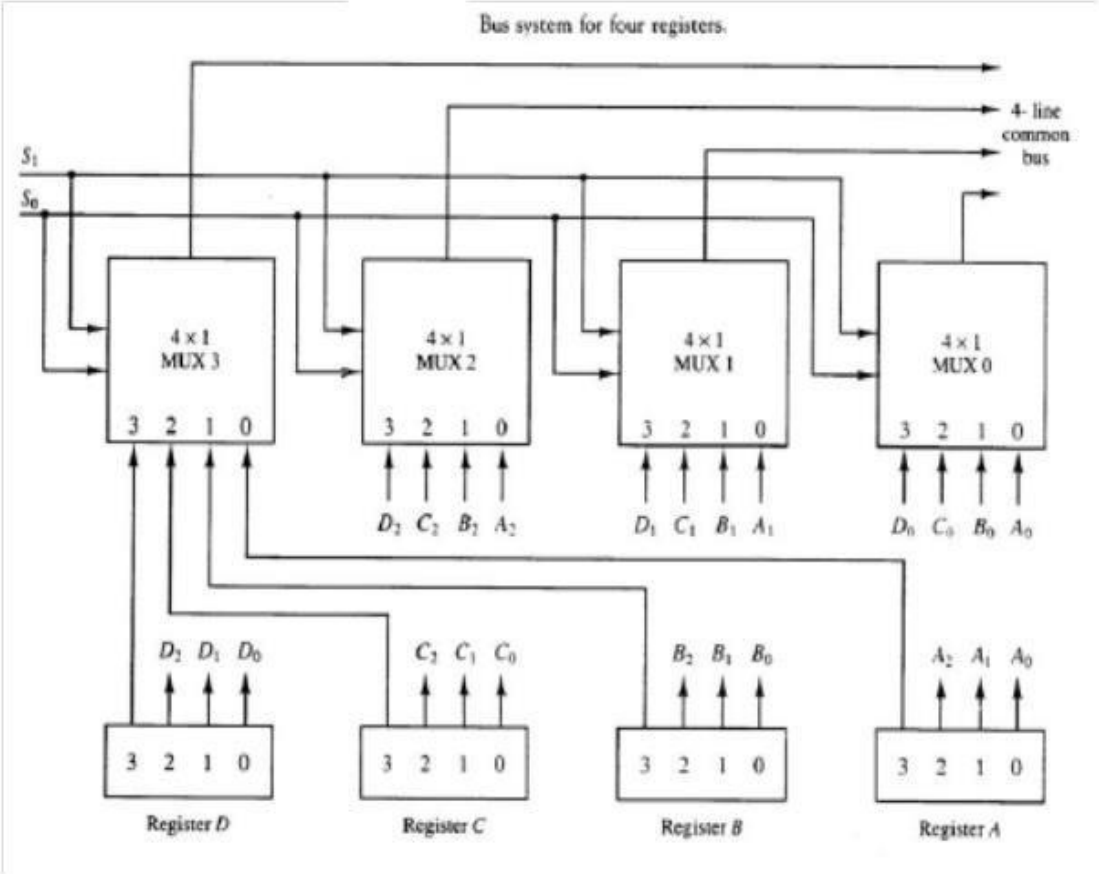
Basic Symbols for Register Transfers

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	*MAR*, *R2*
Parentheses ( )	Denotes a part of a register	*R2*(0–7), *R2*(*L*)
Arrow ←	Denotes transfer of information	*R2* ← *R1*
Comma ,	Separates two microoperations	*R2* ← *R1*, *R1* ← *R2*

Bus and Memory Transfers

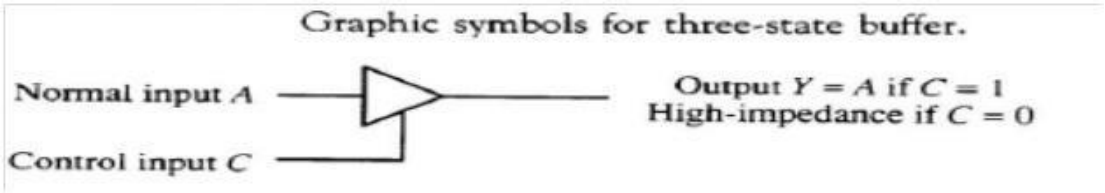
• Rather than connecting wires between all registers, a common bus is used
• A bus structure consists of a set of common lines, one for each bit of a register
• Control signals determine which register is selected by the bus during each transfer
• Multiplexers can be used to construct a common bus
• Multiplexers select the source register whose binary information is then placed on the bus

• The select lines are connected to the selection inputs of the multiplexers



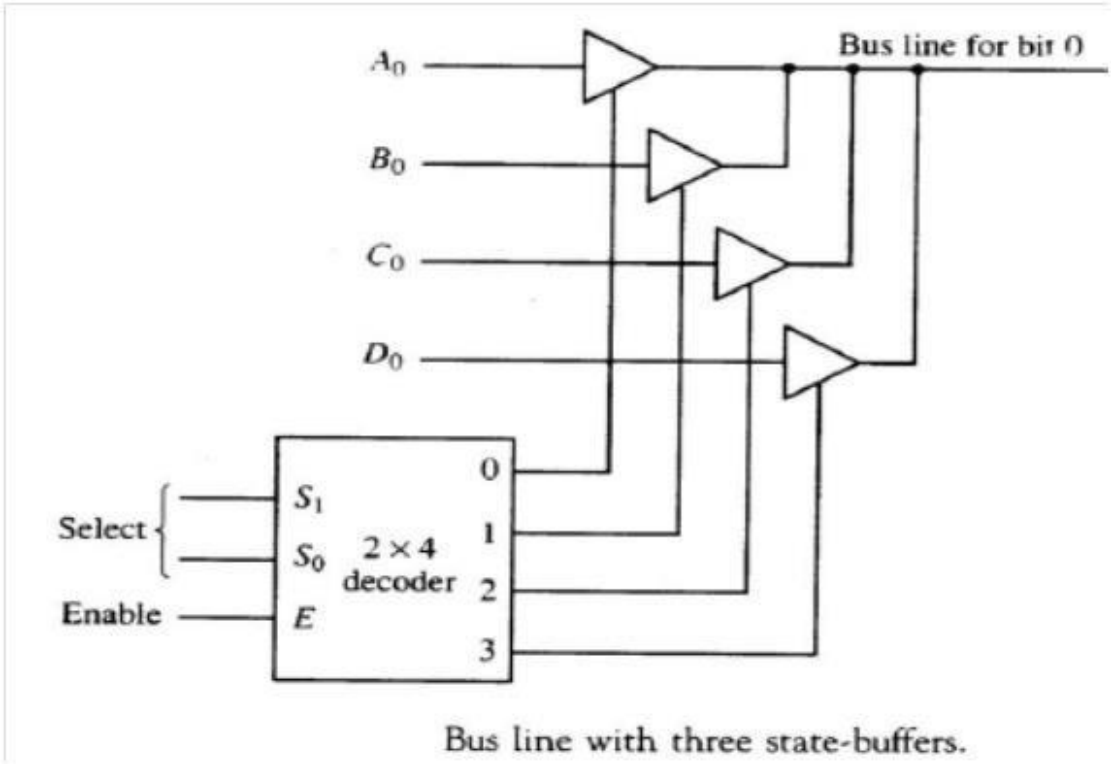
and choose the bits of one register .

- In general, a system will multiplex *k* registers of *n* bits each to produce an *n*-line common bus
- This requires *n* multiplexers – one for each bit
- The size of each multiplexer must be *k* x 1
- The number of select lines required is *log k* • 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
- Rather than listing each step as BUS ← C, R1 ← BUS, use R1 ←C, since the bus is implied
- Instead of using multiplexers, *three-state gates* can be used to construct the bus system
- A three-state gate is a digital circuit that exhibits three states
- Two of the states are signals equivalent to logic 1 and 0
- The third state is a *high-impedance* state – this behaves like an open circuit, which means the output is disconnected and does not have a logic significance .



- The three-state buffer gate has a normal input and a control input which determines the output state

• With control 1, the output equals the normal input
• With control 0, the gate goes to a high-impedance state
• This enables a large number of three-state gate outputs to be connected with wires to form a common bus line without endangering loading effects.



Bus line with three state-buffers.

• Decoders are used to ensure that no more than one control input is active at any given time
• This circuit can replace the multiplexer in Figure 4.3
• To construct a common bus for four registers of *n* bits each using three-state buffers, we need *n* circuits with four buffers in each
• Only one decoder is necessary to select between the four registers
• Designate a memory word by the letter M
• It is necessary to specify the address of M when writing memory transfer operations
• Designate the address register by AR and the data register by DR

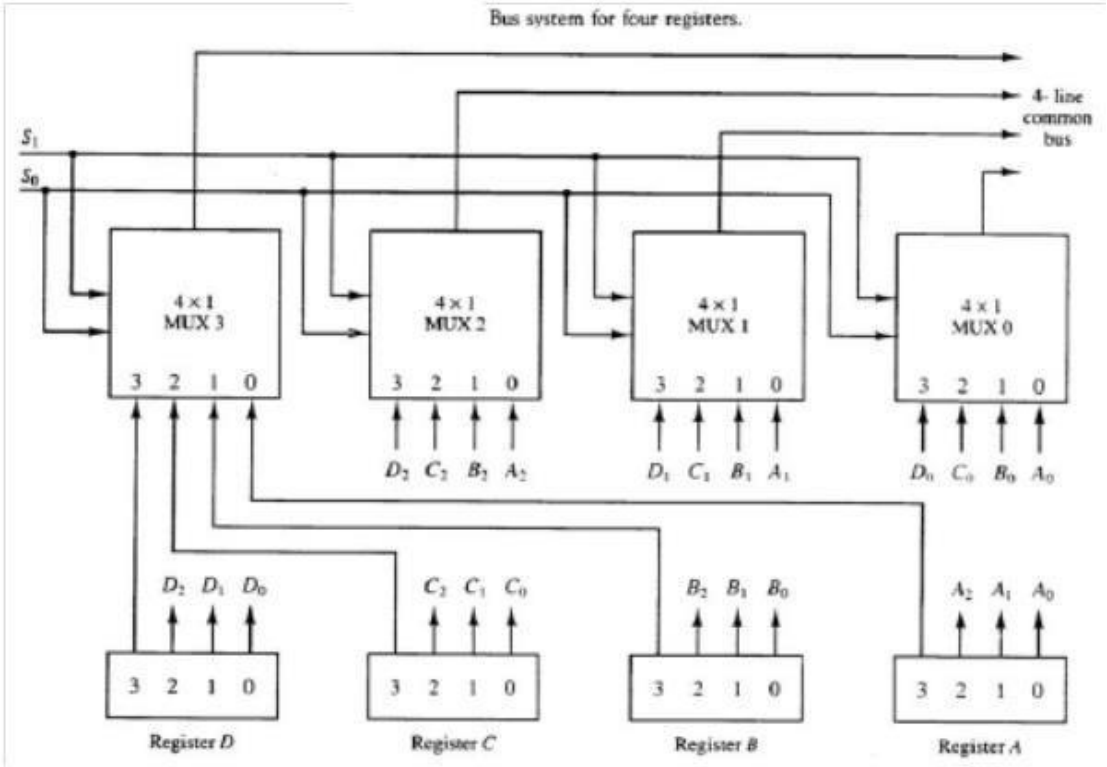
• The read operation can be stated as:
Read: DR ← M[AR]
• The write operation can be stated as:
Write: M[AR] ← R1

Arithmetic Micro operations

• There are four categories of the most common micro operations:
- Register transfer: transfer binary information from one register to another
- Arithmetic: perform arithmetic operations on numeric data stored in registers
- Logic: perform bit manipulation operations on non-numeric data stored in registers
- Shift: perform shift operations on data stored in registers

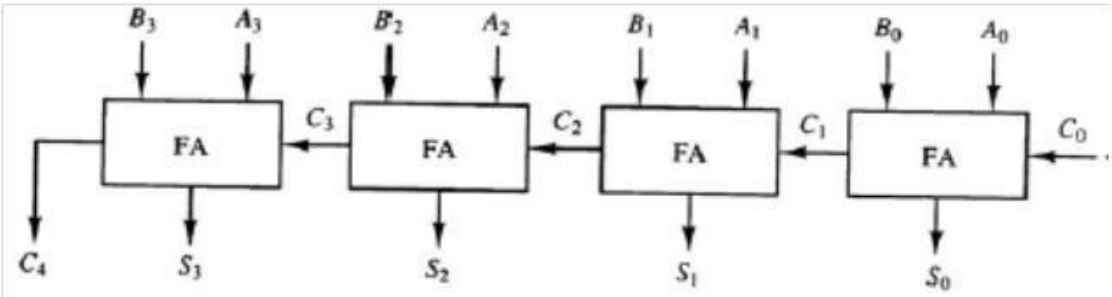
• The basic arithmetic micro operations are addition, subtraction, increment, decrement, and shift

• Example of addition: R3 ← R1 +R2
• Subtraction is most often implemented through complementation and addition
• Example of subtraction: R3 ← R1 + R2 + 1 (strikethrough denotes bar on top – 1's complement of R2)
• Adding 1 to the 1's complement produces the 2's complement .
• Adding the contents of R1 to the 2's complement of R2 is equivalent to subtracting.



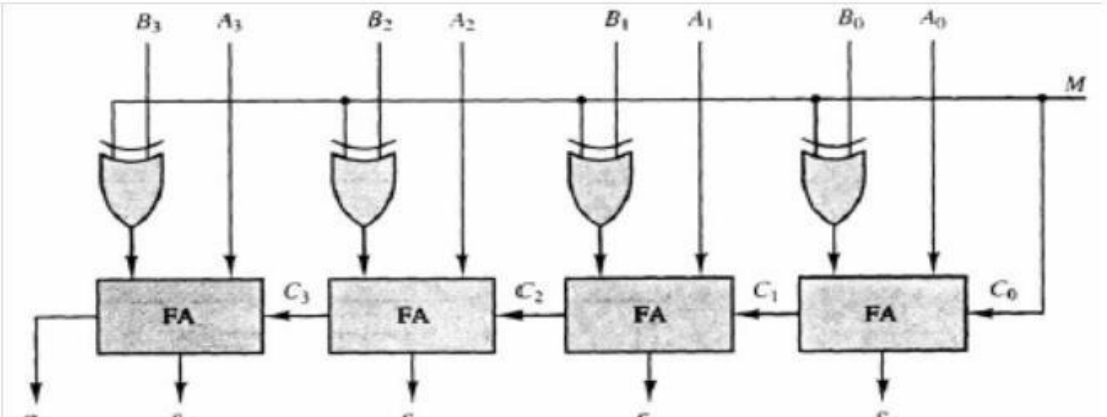
• Multiply and divide are not included as micro operations
• A micro operation is one that can be executed by one clock pulse

• Multiply (divide) is implemented by a sequence of add and shift micro operations (subtract and shift)
• To implement the add micro operation with hardware, we need the registers that hold the data and the digital component that performs the addition
• A full-adder adds two bits and a previous carry.
• A *binary adder* is a digital circuit that generates the arithmetic sum of two binary numbers of any lengths
• A binary added is constructed with full-adder circuits connected in cascade
• An *n*-bit binary adder requires *n* full-adders



4-bit binary adder.

• The subtraction A-B can be carried out by the following steps
  - Take the 1's complement of B (invert each bit)
  - Get the 2's complement by adding 1
  - Add the result to A
• The addition and subtraction operations can be combined into one

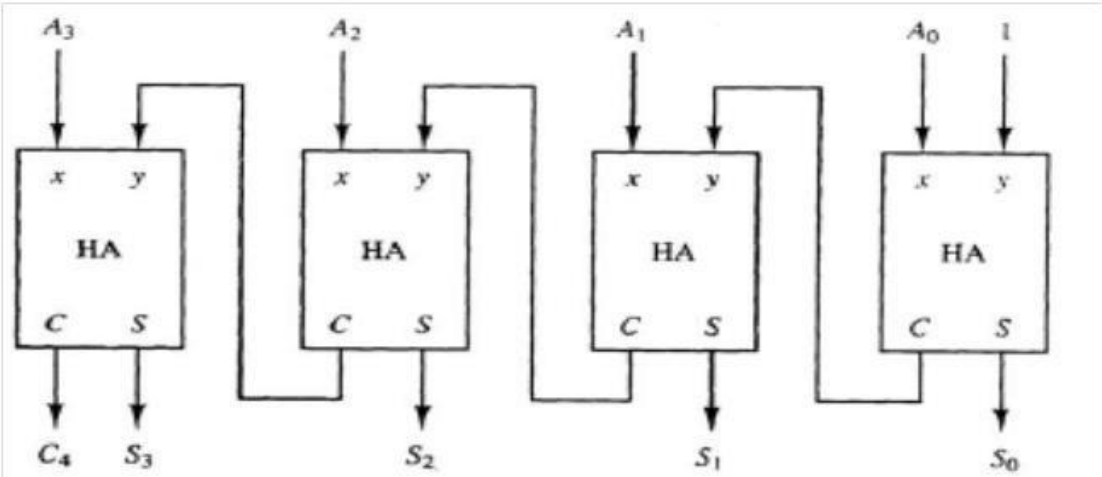


4-bit adder-subtractor.

common circuit by including an XOR gate with each full-adder.

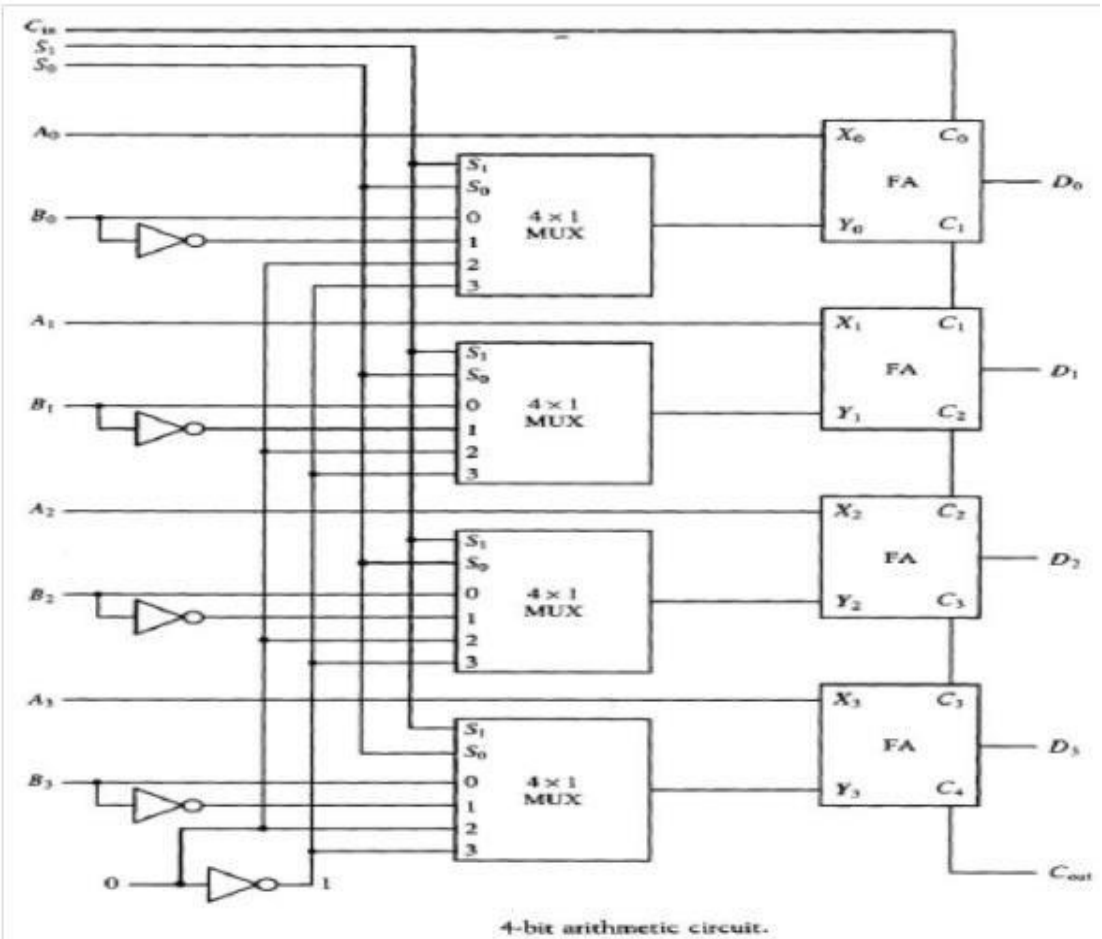
• The increment micro operation adds one to a number in a register
• This can be implemented by using a binary counter – every time the count enable is active, the count is incremented by one
• If the increment is to be performed independent of a particular register, then use half- adders connected in cascade .

• An *n*-bit binary incrementer requires *n* half-adders



4-bit binary incrementer.

• Each of the arithmetic micro operations can be implemented in one composite arithmetic circuit
• The basic component is the parallel adder
• Multiplexers are used to choose between the different operations
• The output of the binary adder is calculated from the following sum: D = A + Y + Cin



4-bit arithmetic circuit.

Arithmetic Circuit Function Table

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

2.2. Logic Micro operations

• Logic operations specify binary operations for strings of bits stored in registers and treat each bit separately

• Example: the XOR of R1 and R2 is symbolized by P: R1 ← R1 ⊕ R2

• Example: R1 = 1010 and R2 =

1100 1010 Content of R1

1100 Content of R2

0110 Content of R1 after P = 1

• Symbols used for logical micro operations:

- OR: ∨
- AND: ∧
- XOR: ⊕

• The + sign has two different meanings: logical OR and summation

• When + is in a micro operation, then summation

• When + is in a control function, then OR

• Example: P + Q: R1 ← R2 + R3, R4 ← R5 ∨ R6

• There are 16 different logic operations that can be performed with two binary variables.

Truth Tables for 16 Functions of Two Variables

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

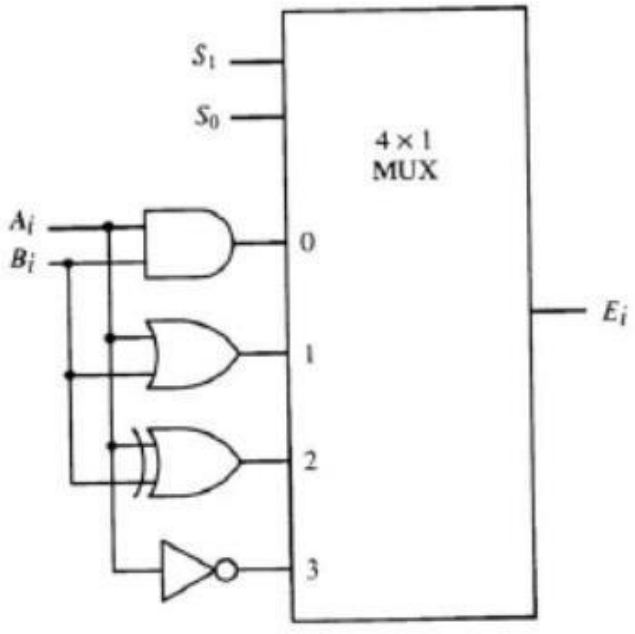
Sixteen Logic Microoperations

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

• The hardware implementation of logic micro operations requires that logic gates be inserted for each bit or pair of bits in the registers.

• All 16 micro operations can be derived from using four logic gates.

One stage of logic circuit.



(a) Logic diagram

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

(b) Function table

• Logic micro operations can be used to change bit values, delete a group of bits, or insert new bit values into a register

• The *selective-set* operation sets to 1 the bits in A where there are corresponding 1's in B

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

A ← A ∨ B

• The *selective-complement* operation complements bits in A where there are corresponding 1's in B

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

A ← A ⊕ B

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

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

A ← A ∧ B

The *mask* operation is similar to the selective-clear operation, except that the bits of A are cleared only where there are corresponding 0's in B

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

A ← A ∧ B

• The *insert* operation inserts a new value into a group of bits

• This is done by first masking the bits to be replaced and then Oring them with the bits to be inserted 0110 1010 A before

0000 1111 B (mask)
0000 1010 A after masking
0000 1010 A before
1001 0000 B (insert)
1001 1010 A after insertion

• The *clear* operation compares the bits in A and B and produces an all 0's result if the two number are equal

1010 A
1010 B
0000 A ← A ⊕ B

2.3. Shift Micro operations

• Shift microoperations are used for serial transfer of data

• They are also used in conjunction with arithmetic, logic, and other data-processing operations

• There are three types of shifts: logical, circular, and arithmetic

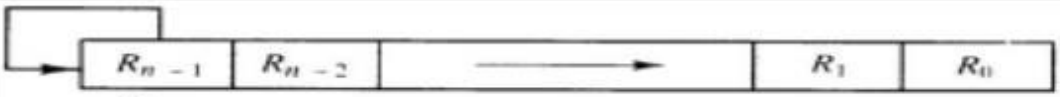
• A *logical shift* is one that transfers 0 through the serial input

• The symbols *shl* and *shr* are for logical shift-left and shift-right by one position R1 ← shl R1
• The *circular shift* (aka rotate) circulates the bits of the register around the two ends without loss of information
• The symbols *cil* and *cir* are for circular shift left and right

Shift Microoperations

Symbolic designation	Description
$R \leftarrow \text{shl } R$	Shift-left register R
$R \leftarrow \text{shr } R$	Shift-right register R
$R \leftarrow \text{cil } R$	Circular shift-left register R
$R \leftarrow \text{cir } R$	Circular shift-right register R
$R \leftarrow \text{ashl } R$	Arithmetic shift-left R
$R \leftarrow \text{ashr } R$	Arithmetic shift-right R

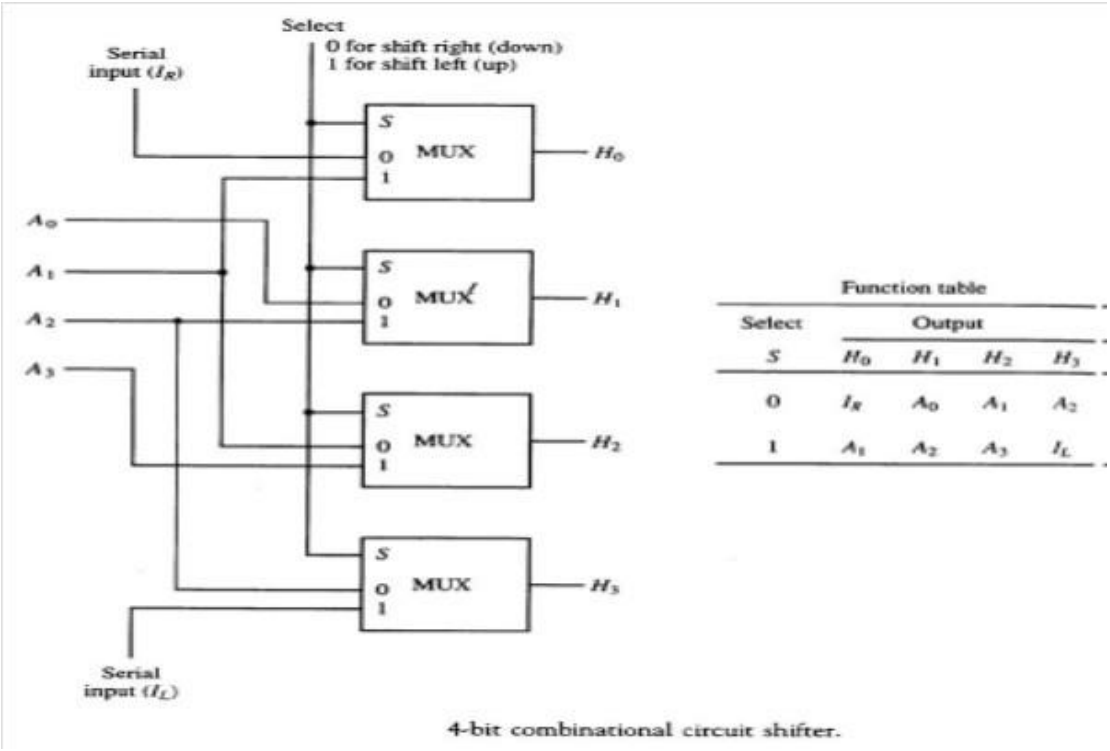
• The *arithmetic shift* shifts a signed binary number to the left or right
• To the left is multiplying by 2, to the right is dividing by 2
• Arithmetic shifts must leave the sign bit unchanged
• A sign reversal occurs if the bit in R_{n-1} changes in value after the shift
• This happens if the multiplication causes an overflow
• An overflow flip-flop V_S can be used to detect the



Arithmetic shift right.

overflow $V_S = R_{n-1} \oplus R_{n-2}$

• A bi-directional shift unit with parallel load could be used to implement this
• Two clock pulses are necessary with this configuration: one to load the value and another to shift
• In a processor unit with many registers it is more efficient to implement the shift operation with a combinational circuit
• The content of a register to be shifted is first placed onto a common bus and the output is connected to the combinational shifter, the shifted number is then loaded back into the register
• This can be constructed with multiplexers



Function table

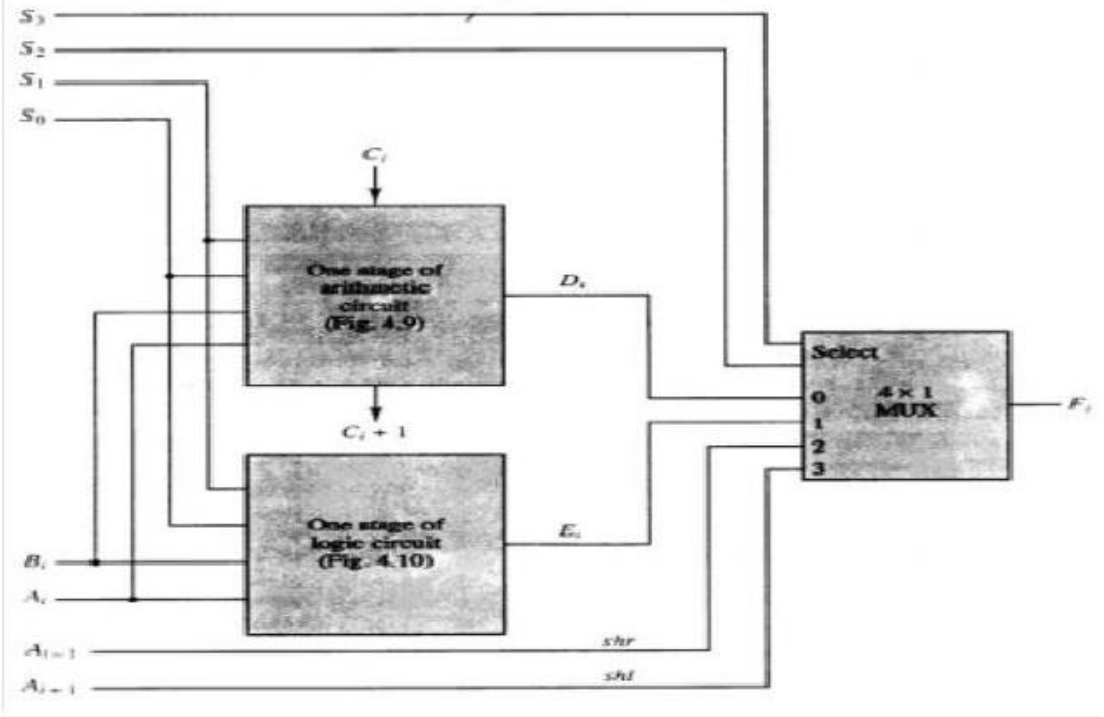
Select	Output			
S	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

4-bit combinational circuit shifter.

Arithmetic Logic Shift Unit

- The *arithmetic logic unit (ALU)* is a common operational unit connected to a number of storage registers
- To perform a microoperation, the contents of specified registers are placed in the inputs of the ALU
- The ALU performs an operation and the result is then transferred to a destination register
- The ALU is a combinational circuit so that the entire register transfer operation from the source registers through the ALU and into the destination register can be performed during one clock pulse period

One stage of arithmetic logic shift unit.



Function Table for Arithmetic Logic Shift Unit

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

Basic Computer Organization & Design

CONTENTS:

- Instruction Codes
- Computer Registers
- Computer Instructions
- Timing and Control
- Instruction Cycle

- Memory Reference Instructions
- Input-Output and Interrupt

Instruction Codes:

The user of a computer can **control the process** by means of a program

A program is a set of instructions that specifies the operations, operands, and the sequence by which processing has to occur.

The instruction code is a **group of bits** that instruct the computer to perform a **specific operation.**

An instruction consists of Opcode and operands.

Instruction Format

Opcode	Operands

Examples:

- ADD A, B
- ADD R1, R2
- MOV CX, 4929h
- MOV AX,BX
- SUB AX, BX
- INC AX

OPCODE:

The most basic part of instruction code is its operation part.

The **operation part** of an instruction code specifies the operation to be performed. The operation code of an instruction is a group of bits that define operations such as ADD, Subtract, multiply, shift and complement.

Examples:

ADD A, B

SUB AX, BX

MUL AX, BX

The number of bits required for an operation code of an instruction depends on the total number of operations available on the computer.

The operation code must consists of at least n bits for a given 2^n distinct operations.

An **operation** is a part of instruction stored in computer memory.

The control unit receives the instruction from the memory and interprets the operation code bits.

It then (Opcode) issues a sequence of control signals to initiates micro operations in internal computer registers.

For every operation code, the control issues a sequence of micro operations needed for the hardware implementation of the specified operation.

OPERANDS:

An instruction code should not only specify the operation but also **the registers or the memory words where the operations are to be found** and also the registers or the memory words where the results are to be stored.

Memory words can be specified by instructions codes by their address.

Processor registers can be specified by assigning to the binary code of k bits that specifies one of 2^k registers.

DIFFERENT MODES OF INSTRUCTION:

Based on Second part of Instruction, We can specify the Different modes of an instruction. They are:

Immediate Mode:

When the second part of an instruction specifies an operand, the instruction is said to have an Immediate Operand

EX: ADD AX, 2387 H

Direct Address:

When the second part of an instruction specifies an address of an operand, The instruction is said to have a Direct Address

EX: MOV AX, [1592H]

Indirect Address:

When the second part of an instruction designates an address of a memory word in which the address of the operand is found, the instruction is said to have a Indirect Address.

EX: Load R1, @500

THE BASIC COMPUTER

The Basic Computer has two components, a Processor Register and Memory. The Memory unit has a capacity of 4096 words. Each word contains 16 bits.

To specify the address of operand, 12 bits are needed; $4096 = 2^{12}$ .So 12 bits of an instruction word are needed to specify Address and 4 more bits are available for the Opcode. (Or 3 bits for opcode & 1 bit to specify Direct or Indirect Address).

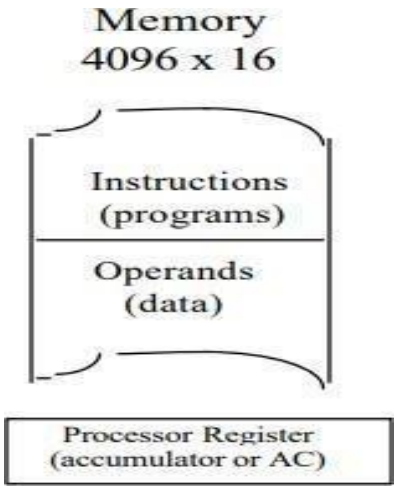


Fig1.3: Basic Computer

Stored program Organization:

The simplest way to organize a computer is to have **one processor register(AC) and a instruction code format** with two parts.

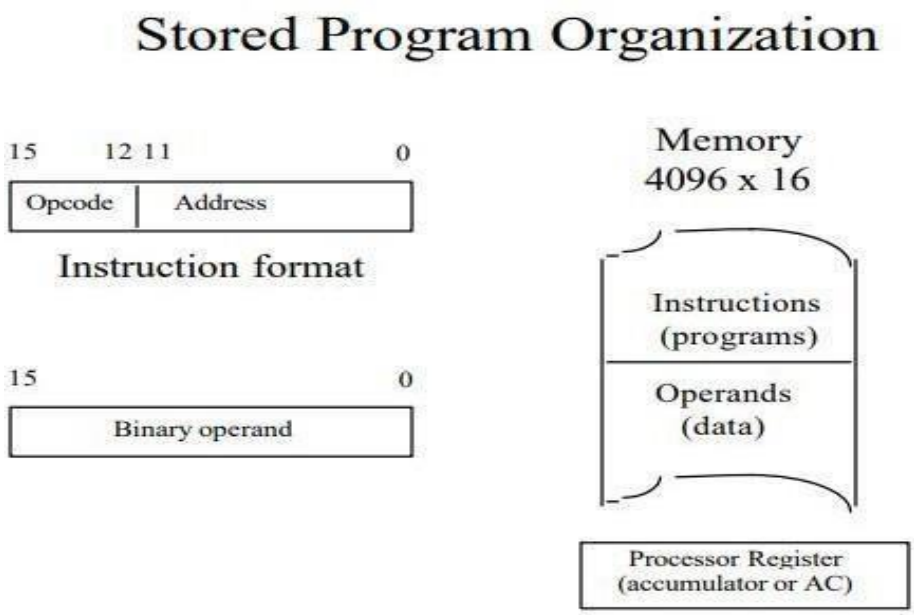
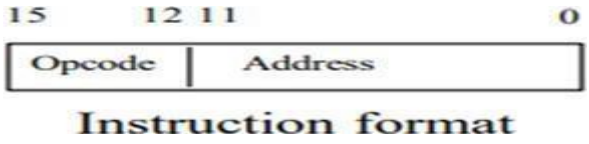


Fig1.4: Stored Program Organization

A computer instruction is often divided into two parts



The first part specifies the operation to be performed. The second part specifies an address.

The memory address tells the control where to find an operand in memory. This operand is read from memory and used as the data to be operated on together with the data stored in the processor register. Fig1.4 depicts the type of Organization.

MEMORY:

Instructions are stored in one section of memory and data in other.

For a memory unit with 4096 words we need 12 bits to specify an address since 2^12=4096.

If we store each instruction code in one 16-bit memory word. There are 4 bits available for the operation code to specify one out of 16 possible operations and 12 bits to specify the address of an operand.

The control read a 16 bit operand form the data portion of the memory. It then executes the operation specified by the operation code.

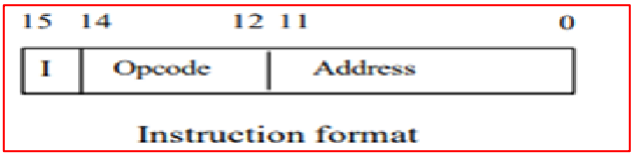
PROCESSOR REGISTER (ACCUMULATOR):

Computers that have a single processor register usually assign to it the name accumulator and label it AC. The operation is performed with the memory operand and content of AC.

If an operation in an instruction code does not need an operand from memory, the rest of the bits in the instruction can be used for other purposes. For example, operations such as clear AC complement AC, and increment AC operate on data stored in AC register. They do not need an operand from memory and they can be used to specify other operations for the computer. They do not need an operand from memory and they can be used to specify other operations for the computer.

Direct Addressing& Indirect Addressing:

Consider the instruction format shown in figure a.



Instruction format

It consists of 3 bit Opcode, a 12 bit address and an indirect address mode designated by I.

How to distinguish between a direct and indirect address?

A. One bit of the instruction code (I bit) can be used to distinguish between a direct and indirect address.

When I bit =0; It Specifies a Direct Address

When I bit =1; It Specifies an Indirect Address

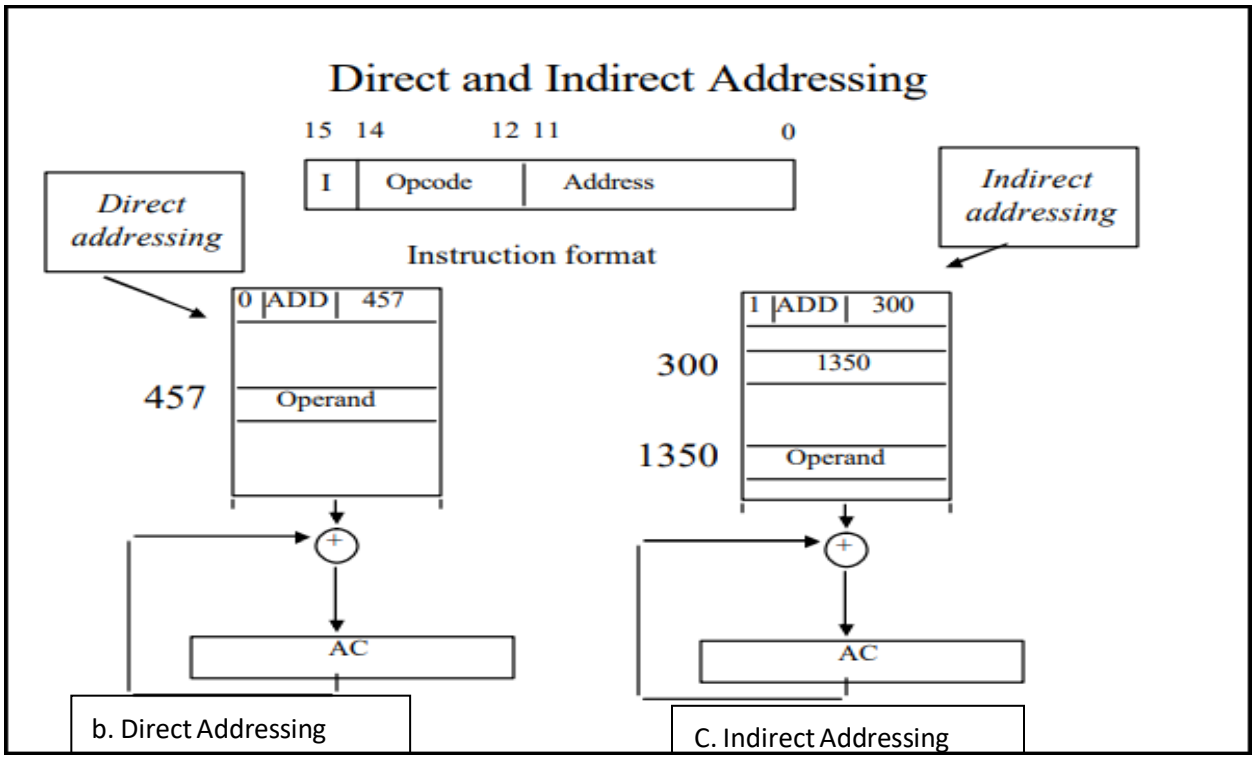


Fig1.5 Pictorial Representation of Direct & Indirect Addressing

Direct addressing

A direct address instruction is shown in figure b.

It is placed in address 22 in memory and the I bit is 0, so the instruction is recognized as a direct address instruction. The opcode specifies an ADD instruction, and the address part is the binary equivalent of 457.The control finds the operand in memory at address 457 & adds it the content of AC.

Indirect addressing

The instruction in address 35 in memory is shown in figure c.

It has a mode bit I =1, therefore it is recognized as an Indirect address instruction. The address part is the binary equivalent of 300.The control goes to address 300 to find the address of the operand. The address of the operand in this case is 1350.The operand found in address 1350 is then added to the content of AC. The indirect address instruction needs two references to memory to fetch an operand.

- The first reference is to needed to read the address of the operand.
- The second reference is for the operand itself.

Effective Address:

It is defined as the address of an operand. Thus the Effective Address in the instruction of figure a is 457 and figure b is1350.

Computer Registers:

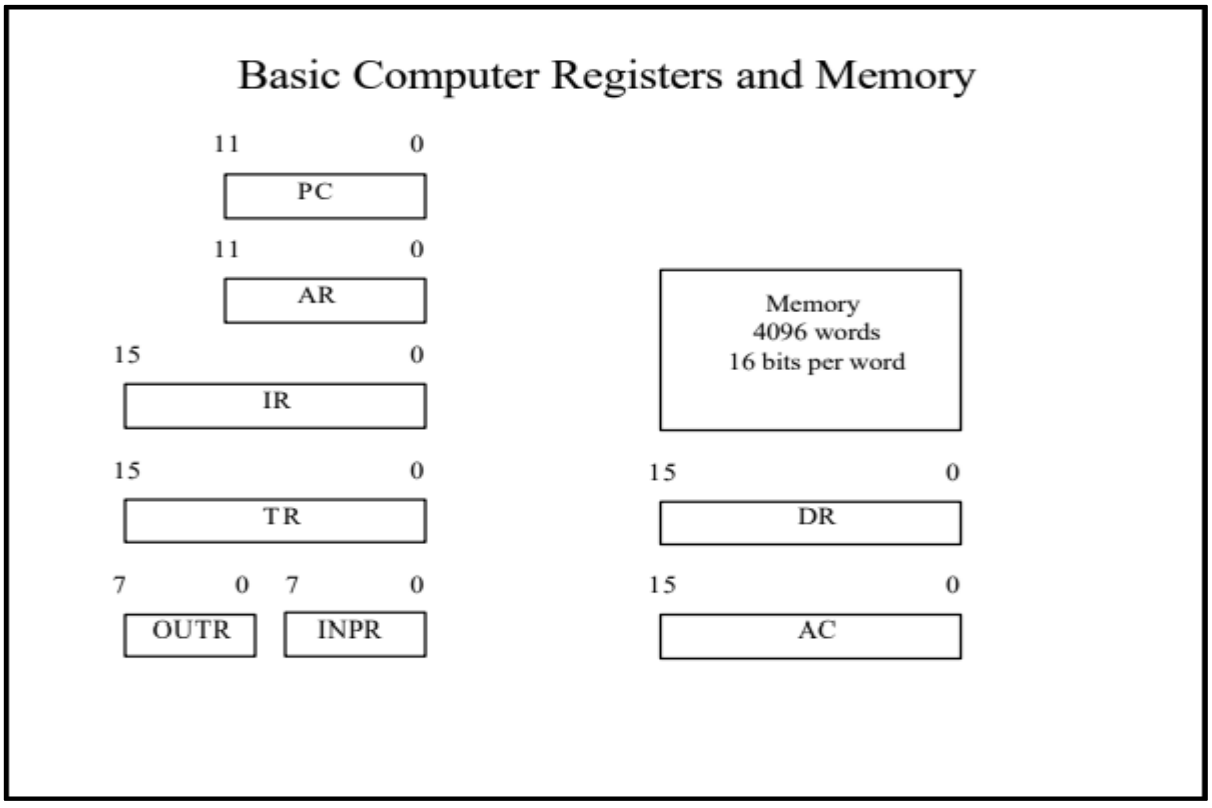


Fig1.6 Basic Computer Registers and Memory

A processor has many registers to hold instructions, addresses, data, etc

The processor has a register, the *Program Counter* (PC) **that holds the memory address of the next instruction .**

- Since the memory in the Basic Computer only has 4096 locations, the PC only needs 12 bits

The memory unit has a capacity of 4096 words and each word contains 16 bits.12 bits of an instruction word are needed to specify the address of an operand. This leaves 3 bits for the operation part of the instruction and a bit (I) to specify a direct or indirect address. In a Direct or indirect addressing, **the processor needs to keep track of what locations in memory it is addressing:**

The *Address Register* (AR) is used for this

- The AR is a 12 bit register in the Basic Computer

When an operand is found, using either direct or indirect addressing, it is placed in the *Data Register* (DR). The **data register (DR) holds the operand read from memory.** The processor then uses this value as data for its operation

The accumulator (AC) register is a **general purpose processing register.** The significance of a general purpose register is that it can be referred to in instructions

- e.g. load AC with the contents of a specific memory location(LDA); store the contents of AC (STA)into a specified memory location

The instruction read form memory is placed in the instruction register (IR).

Often a processor will need a scratch register to store intermediate results or other temporary data; in the Basic Computer this is the *Temporary Register* (TR). **The temporary register (TR) is used for holding the temporary data during the processing.**

The Basic Computer uses a very simple model of input/output (I/O) operations. Input devices are considered to send 8 bits of character data to the processor. The processor can send 8 bits of character data to output devices

The *Input Register* (INPR) holds an **8 bit character received from an input device**

The *Output Register* (OUTR) holds **an 8 bit character to be send to an output device**

- The registers are also listed in table together with a brief description of their function and the number of bits that they contain..

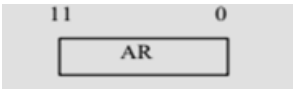
List of Registers for the Basic Computer

Register Symbol	# of Bits	Register Name	Function
DR	16	Data Register	Holds memory operand
AR	12	Address Register	Holds mem. address
AC	16	Accumulator	Processor Reg.
IR	16	Instruction Register	Holds instruction code
PC	12	Program Counter	Holds instruction address
TR	16	Temporary Register	Holds temporary data
INPR	8	Input Register	Holds input character
OUTR	8	Output Register	Holds output character

Table1.1 List of registers for Basic computers

The memory address register has 12 bits since this is the width of a memory address.

The program counter also has 12 bits and it holds the next instruction to be read from memory



after the current instruction is executed.

The PC goes through a counting sequence and causes the computer to read sequential instructions previously stored in memory. Instruction words are read andexecuted in sequence unless a branch instruction is encountered.

A branch instruction calls for a transfer to a nonconsecutive instruction in the program. The address part of a branch instruction is transferred to PC to become the address of the next instruction.

Common Bus System:

A basic computer has 8 registers, memory unit and a control unit. Paths must be provided to transfer information from one register to another and between memory and registers. . A more efficient scheme for transferring information in a system with many registers is to use a common bus. **To avoid excessive wiring,** memory and all the register are connected via **a common bus.** The connection of the registers and memory of the basic computer to a common bus system is shown in Fig.1.6. The **outputs of seven registers and memory are connected to the common bus.** The specific output that is selected for the bus lines at any given time is determined from the binary value of the selection variables $S_2S_1S_0$**.** The register who's **LD (Load)** is enabled receives the data from the bus. Registers can be incremented by setting the **INR** control input and can be cleared by setting the **CLR** control input.

The Accumulator's input must come via the Adder & Logic Circuit. This allows the Accumulator and Data Register to swap data simultaneously.

The address of any memory location being accessed must be loaded in the Address Register (AR).

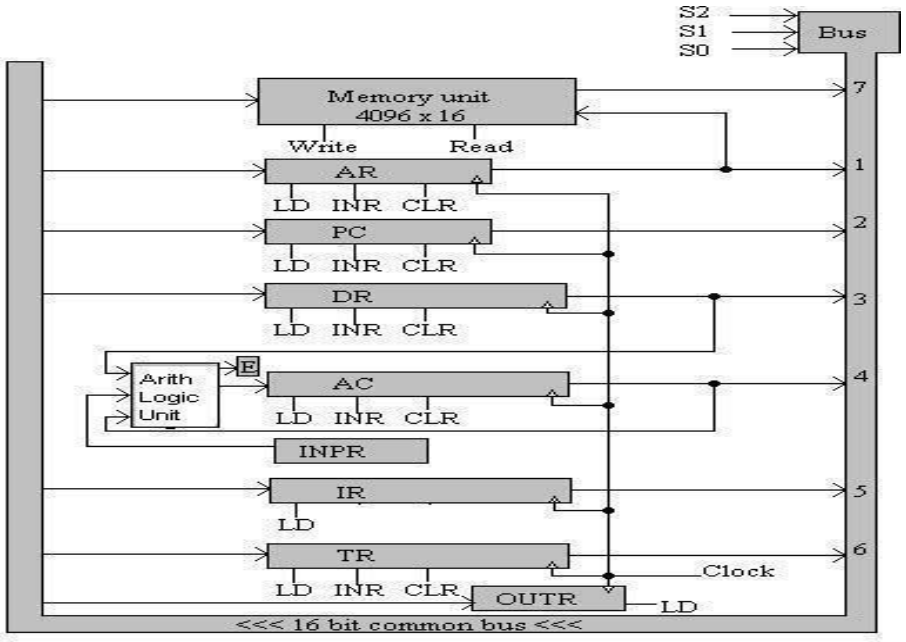


Fig1.6 Basic registers connected via a common Bus

SUMMARY

1. The organization of the computer is defined by its internal registers, the timing and control structure, and the set of instructions that it uses.
2. The user of a computer can control the process by means of a program. A program is a set of instructions that specify the operations, operations operands, and the sequence by which processing has to occur.
3. The general-purpose digital computer is capable of executing various micro operations and, in addition, can be instructed as to what specific sequence of operations it must perform.
4. An operation is part of an instruction stored in computer memory. It is a binary code tells the computer to perform a specific operation.
5. The operation part of an instruction code specifies the operation to be performed.
6. Instruction code formats are conceived computer designers who specify the architecture of the computer.

7. The simplest way to organize a computer is to have one processor register and instruction code format with two parts. The first part specifies the operation to be performed and the second specifies an address.

8. Computer instructions are normally stored in consecutive memory locations and are executed sequentially one at a time.

9. The direct and indirect addressing modes are used in the computer.

10. The memory word that holds the address of the operand in an indirect address instruction is used as a pointer to an array of data. The pointer could be placed in processor register instead of memory as done in commercial computers.

11. The basic computer has eight registers, a memory unit, and a control unit. Paths should be provided to transfer information from one register to another and between memory and registers.

12. The output of seven registers and memory are connected to the common bus.

13. The lines from the common bus are connected to the inputs of each register and the data input of each register and the data inputs of the memory.

14. The 16 lines of the common bus receive information from sex registers and the memory unit. The bus lines are connected to the inputs of six registers and the memory.

15. The content of any register can be applied onto the bus and an operation can be performed in the adder and logic circuit during the same clock cycle.

16. The input data and output data of the memory are connected to the common bus, but the memory address is connected to AR.

17. Content of any register can be applied onto the bus and an operation can be performed in the adder and logic circuit during the same clock cycle.

18. The clock transition at the end of the cycle transfers the content of the bus into the designated destination register and the output of the adder and logic circuit into AC.

Computer Instructions

The basic computer has 3 instruction code formats as shown in the figure below:

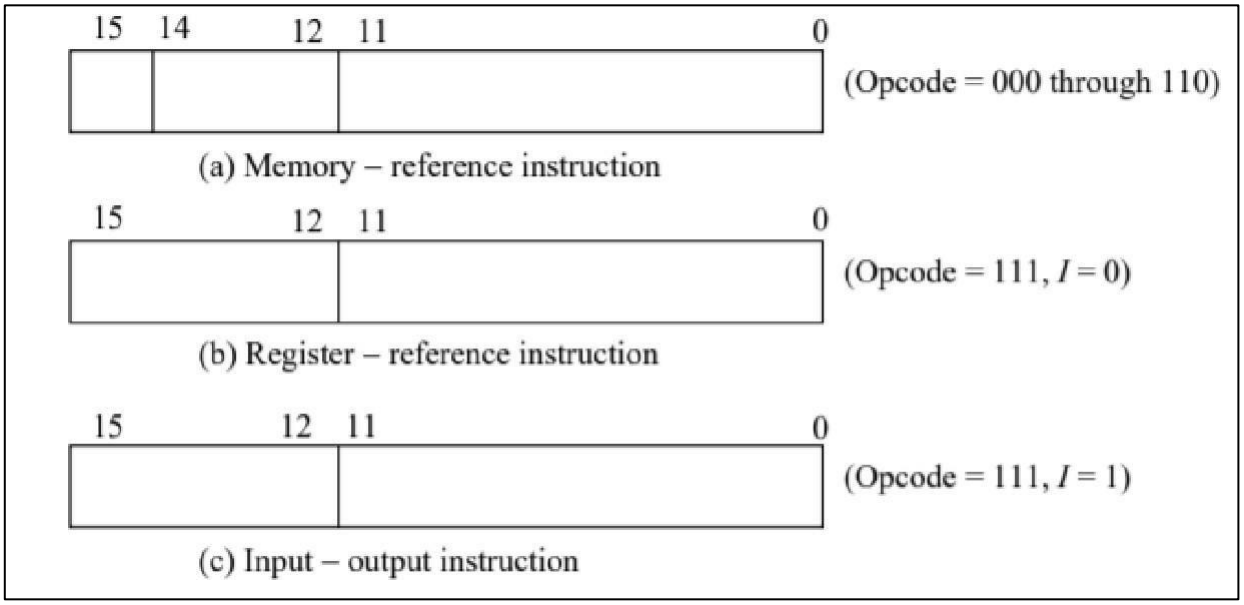


Fig1.7 Basic computer Instruction format

1. In Memory-reference instruction, 12 bits of memory is used to specify an address, 3bits for opcode and one bit to specify the addressing mode I.

When I=0; represents direct Addressing Mode

I=1; represents Indirect Addressing Mode

2. The Register-reference instructions are represented by the **Opcode 111** with a **0 in the leftmost bit (bit 15) of the instruction.** A Register-reference instruction specifies an operation on or a test of the AC (Accumulator) register.Here the operand from memory is not needed, therefore the other 12 bits are used to specify the operation or test to be executed.

3. An Input-Output instruction does not need a reference to memory and is recognized by the **operation code 111** with a **1 in the leftmost bit of the instruction.** The remaining 12 bits are used to specify the type of the input-output operation or test performed.

The three operation code bits in positions 12 through 14 should be equal to 111. Otherwise, the instruction is a memory-reference type. When the three operation code bits are equal to 111, control unit inspects the bit in position 15. If

the bit (15) is 0, the instruction is a register-reference type. Otherwise, the instruction is an input-output type having bit 1 at position 15.

The instructions for the computer are listed in Table 1.2. The symbol designation is a three letter word and represents an abbreviation intended for programmers and users.

Symbol	Hex Code I = 0	Hex Code I = 1	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load AC from memory
STA	3xxx	Bxxx	Store content of AC into memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instr. if AC is positive
SNA	7008		Skip next instr. if AC is negative
SZA	7004		Skip next instr. if AC is zero
SZE	7002		Skip next instr. if E is zero
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

Table1.2 Basic computer Instructions

The hexadecimal code is equal to the equivalent hexadecimal number of the binary code used for the instruction.

By using the hexadecimal equivalent we reduced the **16 bits of an instruction code to four digits with each hexadecimal digit being equivalent to four bits.**

Instruction Set Completeness:

A computer should have a set of instructions so that the user can construct machine language programs to evaluate any function.

A set of instructions is said to be complete if the computer includes a sufficient number of instructions in each of the following categories:

- Arithmetic, logical and shift instructions
- A set of instructions for moving information to and from memory and processor registers.
- Program control Instructions together with instructions that check status conditions.
- Input and Output instructions

Arithmetic, logic and shift instructions provide computational capabilities for processing the type of data the user may wish to employ.

Transfer of Information: A huge amount of binary information is stored in the memory unit, but all computations are done in processor registers. Therefore, one must possess the **capability of moving information between these two units**.

Program control instructions such as branch instructions are used to change the sequence in which the program is executed.

Input and Output instructions act as an interface between the computer and the user. Programs and data must be transferred into memory, and the results ofcomputations must be transferred back to the user.

Instruction Types

Functional Instructions
- Arithmetic, logic, and shift instructions
- ADD, CMA, INC, CIR, CIL, AND, CMA,CLA

Transfer Instructions: Data transfers between the main memory and the processoregisters
- LDA, STA

Control Instructions
- Program sequencing and control
- BUN, BSA, ISZ

Input/Output Instructions
- Input and output
- INP, OUT

Timing and Control

The timings for all the registers in the basic computer is controlled by a master clock generator. Its clock pulses are applied to all flip-flops and register in the system & to flip-flops and registers in the control unit.

The clock pulses do not change the state of a register, unless the register is enabled by a control signal.

The control signals are generated in the control unit and provide control inputs for the bus's multiplexers and for the processor registers and provides micro operations for the accumulator.

CONTROL ORGANIZATION:

The Control Organization is classified into two major categories:

- Hardwired Control
- Micro programmed Control

Hardwired Control

The Hardwired Control organization involves the control logic to be implemented with gates, flip-flops, decoders, and other digital circuits.

The main advantage of Hardwired Control is its fast mode of operation.

If the design has to be modified or changed, it requires changes in the wiring among the various components.

Micro-programmed Control

The Micro programmed Control organization is implemented by using the programming approach.

The control information is stored in control memory.

The control memory is programmed to initiate requires sequence of micro operations.

Any required changes or modifications can be done by updating the micro program in control memory.

Control unit of a basic computer (Hardwired Control organization):

The following image shows the block diagram of a Hardwired Control organization.

Instruction Register

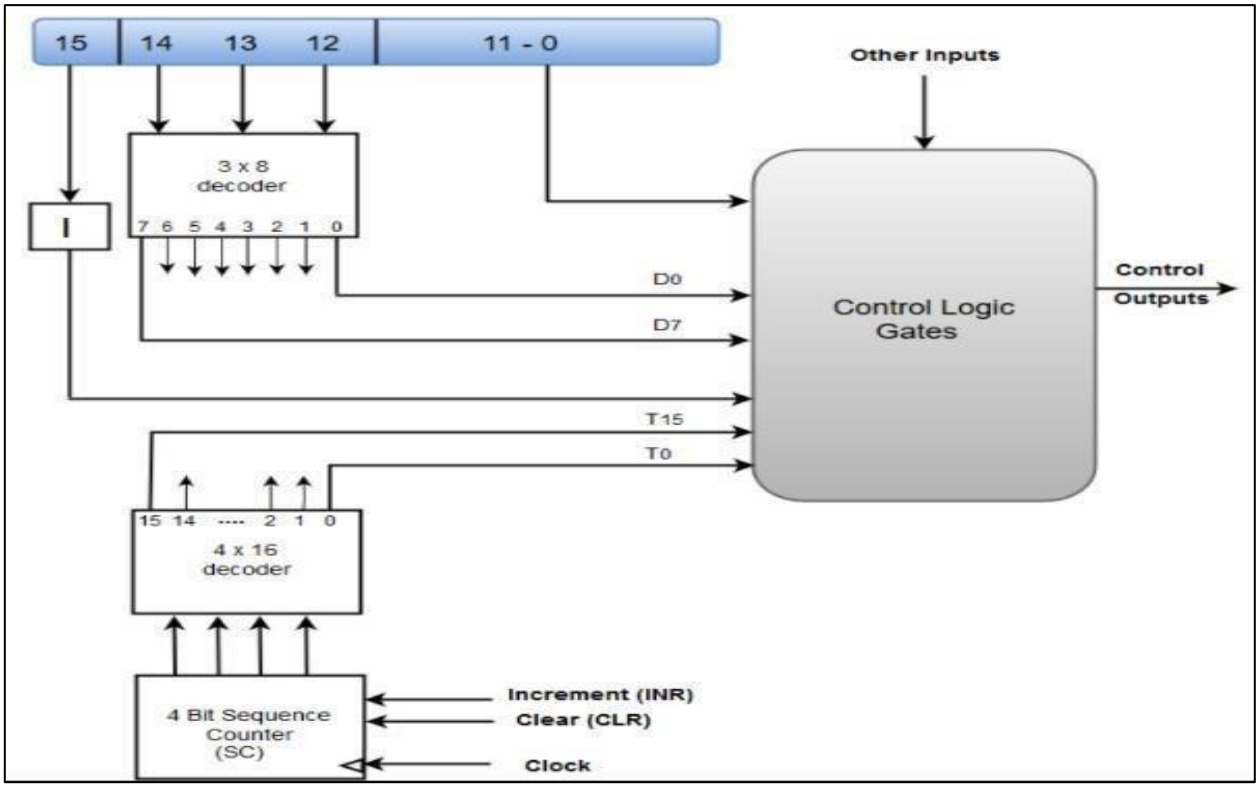


Fig1.8 Control Unit of Basic computer

A Hard-wired Control consists of **two decoders, a sequence counter, and a number of logic gates.**

An instruction fetched from the memory unit is placed in the instruction register (IR). The component of an instruction register includes: I bit, the operation code, and bits 0 through 11.

The operation code in bits 12 through 14 are decoded with a 3 x 8 decoder. The outputs of the decoder are designated by the symbols D0 through D7. The operation code at bit 15 is transferred to a flip-flop designated by the symbol I. The operation codes from Bits 0 through 11 are applied to the control logic gates.

The Sequence counter (SC) can count in binary numbers from 0 through 15.The outputs of the counter are decoded into 16 timing signals T_0 through T_{15}.The sequence counter SC can be incremented or cleared synchronously. Most of the time, the counter is incremented to provide the sequence of timing signals out of the 4 x 16 decoder. Once in a while, the counter is clearedto 0, causing the next active timing signal to be T_0.

As an example, consider the case where SC is incremented to provide timing signals T_0, T_1, T_2, T_3, and T_4 in sequence. At time T_4, SC is cleared to 0 if decoder output D_3 is active. This is expressed symbolically by the statement.

$$\mathbf{D_3T_4: SC \leftarrow 0}$$

Timing Signals

The timing diagram of Fig.1.9 shows the time relationship of the control signals.

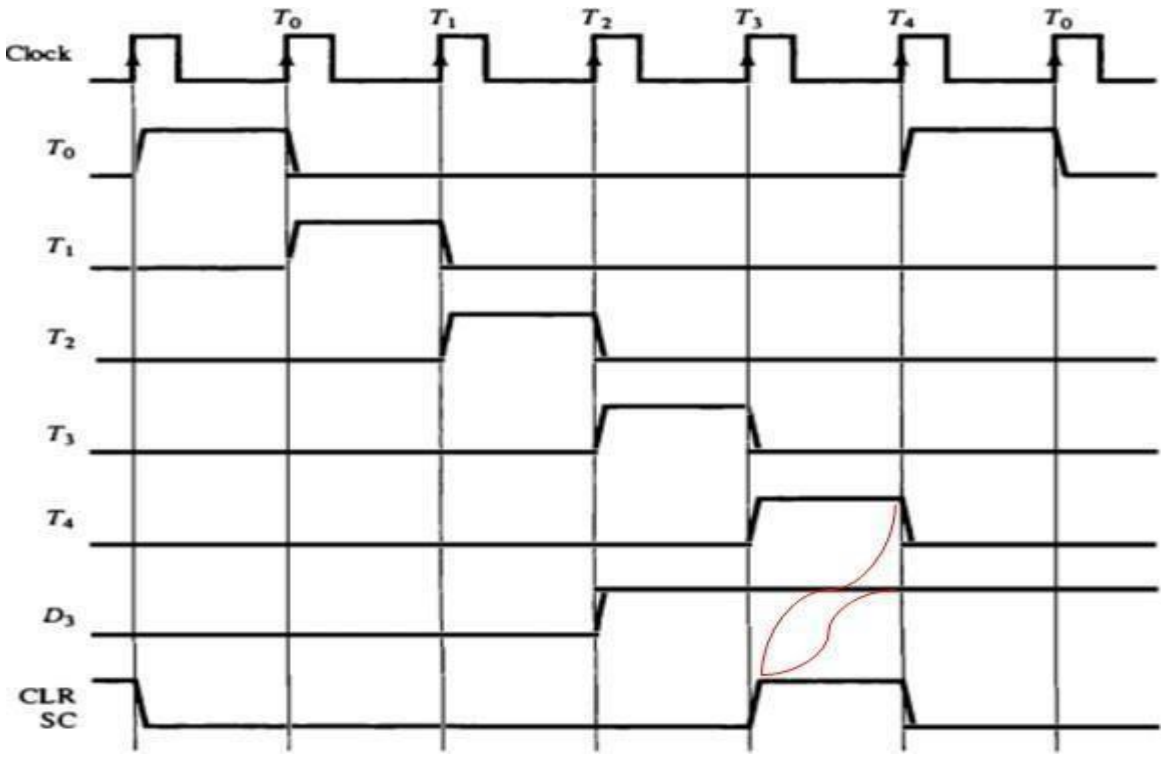


Fig1.9 Example of control Timing Signals

The sequence counter SC responds to the positive transition of the clock. Initially, the CLR input of SC is active. The first positive transition of the clock clears SC to 0, which in turn activates the timing signal T_0 out of the decoder.

T_0 is active during one clock cycle. The positive clock transition labeled T_0 in the diagram will trigger only those registers whose control inputs are connected to timing signal T_0.

SC is incremented with every positive clock transition unless its CLR input is active. This produces the sequence of timing signals T_0, T_1, T_2, T_3 ,T_4 and so on, asshown in the diagram.

If SC is not cleared, the timing signals will continue with T5, T_6 up to T_{15} and back to T_0.The last three waveforms in Fig. show how SC is cleared when D_3T_4 =

1. Output D_3 from the operation decoder becomes active at the end of timing signal T_2.

When timing signal T_4 becomes active, the output of the AND gate that implements the control function D_3T_4 becomes active. This signal is applied to the CLR input of SC. On the next positive clock transition (the one marked T_4 in the diagram) the counter is cleared to 0. This causes the timing signal T_0 to become active instead of T_5 that would have been active if SC were incremented instead of cleared.

- *Reference*

A memory read or writes cycle will be initiated with the rising edge of a timing signal. It will be assumed that a memory cycle time is less than the clock cycle time.

According to this assumption, a memory read or write cycle initiated by a timing signal will be completed by the time the next clock goes through its positive transition. The clock transition will then be used to load the memory word into a register. This timing relationship is not valid in many computers because the memory cycle time is usually longer than the processor clock cycle.

In such a case it is necessary to provide wait cycles in the processor until the memory word is available. To facilitate the presentation, we will assume that await period is not necessary in the basic computer.

To fully comprehend the operation of the computer, it is crucial that one understands the timing relationship between the clock transition and the timing signals. For example, the register transfer statement

$$T_0: AR \leftarrow PC$$

Specifies a transfer of the content of PC into AR if timing signal T_0 is active. T_0 is active during an entire clock cycle interval during this time the content of PC is placed onto the bus (with $S_2S_1S_0 = 010$) and the LD (load) input of AR is enabled.

The actual transfer does not occur until the end of the clock cycle when the clock goes through a positive transition. This same positive clock transition increments the sequence counter SC from 0000 to 0001. The next clock cycle has T_1 active and T_0 inactive.

INSTRUCTION CYCLE

A program residing in the memory unit of the computer consists of a sequence of Instructions. In the basic computer, each instruction cycle consists of the following phases:

1. Fetch an instruction from memory
2. Decode the instruction
3. Read the effective address from memory if the instruction has an indirect address
4. Execute the instruction

Upon the completion of step 4, the control goes back to step1 to fetch, decode and execute the next instruction. This process continues indefinitely unless a HALT instruction is encountered.

Fetch and Decode:

Initially, the PC is loaded with the address of the first instruction in the program.

The sequence counter SC is cleared to 0, provided a decoding timing signal T0

After each clock pulse, the SC is incremented by one, so that the timing signals go through the sequence **T0, T1, T2**, etc.

The micro operations for the **fetch and decode phases** can be specified by the following register transfer statements:

To: AR ← PC

T1: IR ← M [AR], PC ← PC+1

T2:D0,….,D7 ← Decode IR(12-14), AR ← IR(0-11), I ← IR(15)

FETCH PHASE:

Since only **AR is connected to the address inputs of memory**, it is necessary to transfer the address from PC to AR during the clock transition associated with timing signal T0.

The instruction read from memory is then placed in the instruction register IR with the clock transition associated with timing signal T1.

At the same time, PC is incremented by one to prepare it for the address of the next instruction in the program.

DECODE PHASE:

At time T2, the operation code in IR is decoded, the indirect bit is transferred to flip- flop I, and the address part of the instruction is transferred to AR.

Note that SC is incremented after each clock pulse to produce the sequence T0, T1, and T2

Determine the Type of Instruction:

The timing signal that is active after the decoding is T_3. During time T_3, the control unit determines the type of instruction that was just read from memory.

The flowchart of Fig. below presents an initial configuration for the instruction cycle and shows how the control determines the instruction type after the decoding.

The three possible instruction types available in the basic computer are specified in Fig. on basic computer formats.

1. **Memory Reference instructions**
2. **Register Reference instructions**
3. **I/O Reference instructions**

Decoder output D_7 is equal to 1 if the operation code is equal to binary 111. From Fig. on basic computer formats we determine that if $D_7 = 1$, the instruction must be **a register reference or I/O reference**.

I**f $D_7 = 0$,** the operation code must be one of the other seven values 000 through 110, specifying a memory-reference instruction.

Control then inspects the value of the first bit of the instruction, which is now available in flip-flop I. If D7 = 0 and I = 1, we have a memory reference instruction with an indirect address.

It is then necessary to read the effective address from memory. The micro operation for the indirect address condition can be symbolized by the register transfer statement :

$$AR \leftarrow M[AR]$$

Initially, AR holds the address part of the instruction. This address is used during the memory read operation.

The word at the address given by AR is read from memory and placed on the common bus.

Flow chart for Instruction Cycle

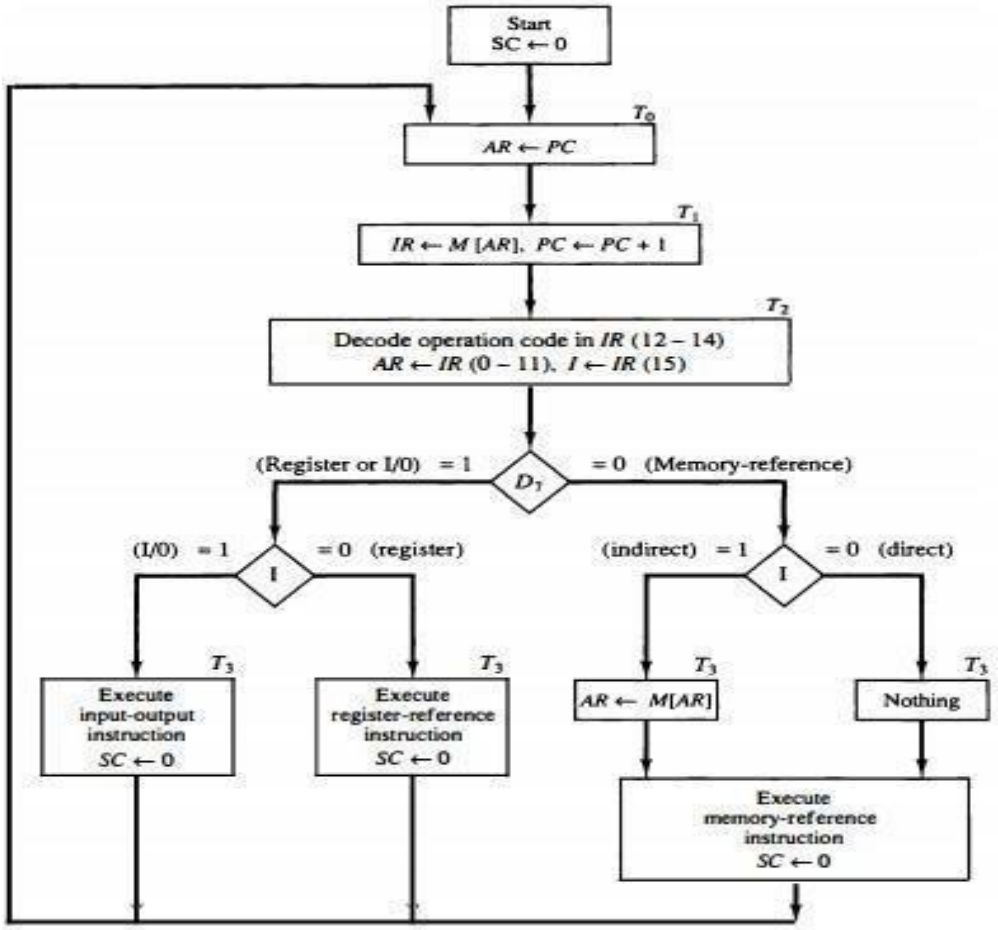


Fig1.10 Flow chart for Instruction cycle

The LD input of AR is then enabled to receive the indirect address that resided in the 12 least significant bits of the memory word.

The three instruction types are subdivided into four separate paths. The selected operation is activated with the clock transition associated with timing signal T_3. This can be symbolized as follows:

$D_7 I'T_3$: Execute a register-reference instruction

$D_7 IT_3$: Execute an input-output instruction

$D'_7 I'T_3$: Nothing

$D'_7 IT_3$: AR ← M [AR]

When a memory-reference instruction with I = 0 is encountered, it is not necessary to do anything since the effective address is already in AR.

However, the sequence counter SC must be incremented when $D'_7T_3 = 1$, so that the execution of the memory-reference instruction can be continued with timing variable T_4

A register-reference or input-output instruction can be executed with the clock associated with timing signal T_3. After the instruction is executed, SC is cleared to 0 and control returns to the fetch phase with $T_0 = 1$.

Note that the sequence counter SC is either incremented or cleared to 0 with every positive clock transition.

We will adopt the convention that if SC is incremented, we will not write the statement SC ← SC + 1, but it will be implied that the control goes to the next timing signal in sequence.

When SC is to be cleared, we will include the statement SC ← 0.

Register-Reference Instructions($D_7I'T_3$):

Register-reference instructions are recognized by the control when $D_7 = 1$ and I = 0. **These instructions** use bits 0 through 11 of the instruction code to specify one of 12 instructions.

These 12 bits are available in IR(0-11). They were also transferred to AR during time T_2.**The control functions** and micro operations for the register-reference instructions are listed in Table below.

These instructions are executed with the clock transition associated with timing variable T_3.

Each control function needs the Boolean relation $D_7I'T_3$, which we designate for convenience by the symbol **"r"**. **The control function** is distinguished by one of the bits in IR(0-11).

By assigning the symbol B_i to bit i of IR, all control functions can be simply denoted by rB_i.

TABLE Execution of Register-Reference Instructions

$D_7I'T_3 = r$ (common to all register-reference instructions)
$IR(i) = B_i$ [bit in $IR(0–11)$ that specifies the operation]

	r:	$SC \leftarrow 0$	Clear SC
CLA	rB_{11}:	$AC \leftarrow 0$	Clear AC
CLE	rB_{10}:	$E \leftarrow 0$	Clear E
CMA	rB_9:	$AC \leftarrow \overline{AC}$	Complement AC
CME	rB_8:	$E \leftarrow \overline{E}$	Complement E
CIR	rB_7:	$AC \leftarrow \text{shr } AC,\ AC(15) \leftarrow E,\ E \leftarrow AC(0)$	Circulate right
CIL	rB_6:	$AC \leftarrow \text{shl } AC,\ AC(0) \leftarrow E,\ E \leftarrow AC(15)$	Circulate left
INC	rB_5:	$AC \leftarrow AC + 1$	Increment AC
SPA	rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	Skip if positive
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if negative
SZA	rB_2:	If $(AC = 0)$ then $PC \leftarrow PC + 1$	Skip if AC zero
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if E zero
HLT	rB_0:	$S \leftarrow 0$ (S is a start–stop flip-flop)	Halt computer

Example of Register reference Instruction:

For example, the instruction CLA has the hexadecimal code 7800, which gives the binary equivalent 0111 1000 0000 0000.

The first bit is a zero and is equivalent to I'. **The next three bits** constitute the operation code and are recognized from decoder output D_7. **Bit 11 in IR is 1** and is recognized from B_{11}.

The control function that initiates the micro operation for this instruction is $D_7I'T_3B_{11} = rB_{11}$. **The execution** of a register-reference instruction is completed at time T_3. **The sequence counter SC** is cleared to 0 and the control goes back to fetch the next instruction with timing signal T_0.

The first seven register-reference instructions perform clear, complement, circular shift, and increment micro operations on the AC or E registers.

The next four instructions cause a skip of the next instruction in sequence when a stated condition is satisfied. The skipping of the instruction is achieved by incrementing PC once again (in addition, it is being incremented during the fetch phase at time T_1).

The condition control statements must be recognized as part of the control conditions.

- **The AC is positive when the sign bit** in AC (15) = 0; **it is negative** when AC (15) = 1.
- The content of AC is zero (AC = 0) if all the flip-flops of the register are zero.

The **HLT instruction clears a start-stop flip-flop S** and stops the sequencecounter from counting. To restore the operation of the computer, the start-stop flip-flop must be set manually.

Memory-Reference Instructions

In order to specify the micro operations needed for the execution of each instruction, it is necessary that the function that they are intended to perform be defined precisely.

We will now show that the function of the **memory-reference instructions can be defined precisely by means of register transfer notation.**

Table below lists the seven memory-reference instructions. The decoded outputD_i for i = 0, 1, 2, 3, 4, 5, and 6 from the operation decoder that belongs to each instruction is included in the table.

The effective address of the instruction is in the address register AR and was placed there during timing signal T_2 when I = 0, or during timing signal T_3 when I = 1.

The execution of the memory-reference instructions starts with timing signal T_4. The symbolic description of each instruction is specified in the table in terms of register transfer notation.

TABLE Memory-Reference Instructions

Symbol	Operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR], \quad E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC, \quad PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1,$ If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

AND to AC:

This is an instruction that performs the AND logic operation on pairs of bits in AC and the memory word specified by the effective address.

The result of the operation is transferred to AC. The micro operations that execute this instruction are:

D_0T_4: DR ← M[AR]
D_0T_5: AC ← AC A DR, SC ← 0

The control function for this instruction uses the operation decoder D_0 since this output of the decoder is active when the instruction has an AND operation whose binary code value is 000.

Two timing signals are needed to execute the instruction. The clock transition associated with timing signal T_4 transfers the operand from memory into DR.

The clock transition associated with the next timing signal T_5 transfers to AC the result of the AND logic operation between the contents of DR and AC.

ADD to AC:

This instruction adds the content of the memory word specified by the effective address to the value of AC.

The sum is transferred into AC and the output carry C_{out} is transferred to the E (extended accumulator) flip-flop.

The micro operations needed to execute this instruction are D_1T_4: DR ←

M[AR]

D_1T_5: AC ← AC + DR, E ← C_{out}, SC ← 0

The same two timing signals, T_4 and T_5, are used again but with operation decoder D_1 instead of D_0, which was used for the AND instruction.

After the instruction is fetched from memory and decoded, only one output of the operation decoder will be active, and that output determines the sequence of rnicro operations that the control follows during the execution of a memory-reference instruction.

LDA: Load to AC

This instruction transfers the memory word specified by the effective address to AC. The micro

operations needed to execute this instruction are

D_2T_4: DR ← M [AR] D_2T_5:

AC ← DR, SC ← 0

Note that there is no direct path from the bus into AC (see figure under Common Bus System).

The adder and logic circuit receive information from DR which can be transferred into AC.

Therefore, it is necessary to read the memory word into DR first and then transfer the content of DR into AC.

The reason for not connecting the bus to the inputs of AC is the delay encountered in the adder and logic circuit.

It is assumed that the time it takes to read from memory and transfer the word through the bus as well as the adder and logic circuit is more than the time of one clock cycle.

By not connecting the bus to the inputs of AC we can maintain one clock cycle per micro operation.

STA: Store AC

This instruction stores the content of AC into the memory word specified by the effective address.

Since the output of AC is applied to the bus and the data input of memory is connected to the bus, we can execute this instruction with one micro operation:

$$D_3T_4: M[AR] \leftarrow AC, SC \leftarrow 0$$

BUN: Branch Unconditionally

This instruction transfers the program to the **instruction specified by the effective address.**

Remember that PC holds the address of the instruction to be read from memory in the next instruction cycle.

PC is incremented at time T_1 to prepare it for the address of the next instruction in the program sequence.

The BUN instruction allows the programmer to specify an instruction out of sequence and we say that the program branches (or jumps) unconditionally.

The instruction is executed with one micro operation:

$$D_4T_4: PC \leftarrow AR, SC \leftarrow 0$$

The effective address from AR is transferred through the common bus to PC.

Resetting SC to 0 transfers control to T_0. The next instruction is then fetched and executed from the memory address given by the new value in PC.

BSA: Branch and Save Return Address

This instruction is useful for branching to a portion of the program called a subroutine or procedure.

When executed, the BSA instruction stores the address of the next instruction in sequence (which is available in PC) into a memory location specified by the effective address.

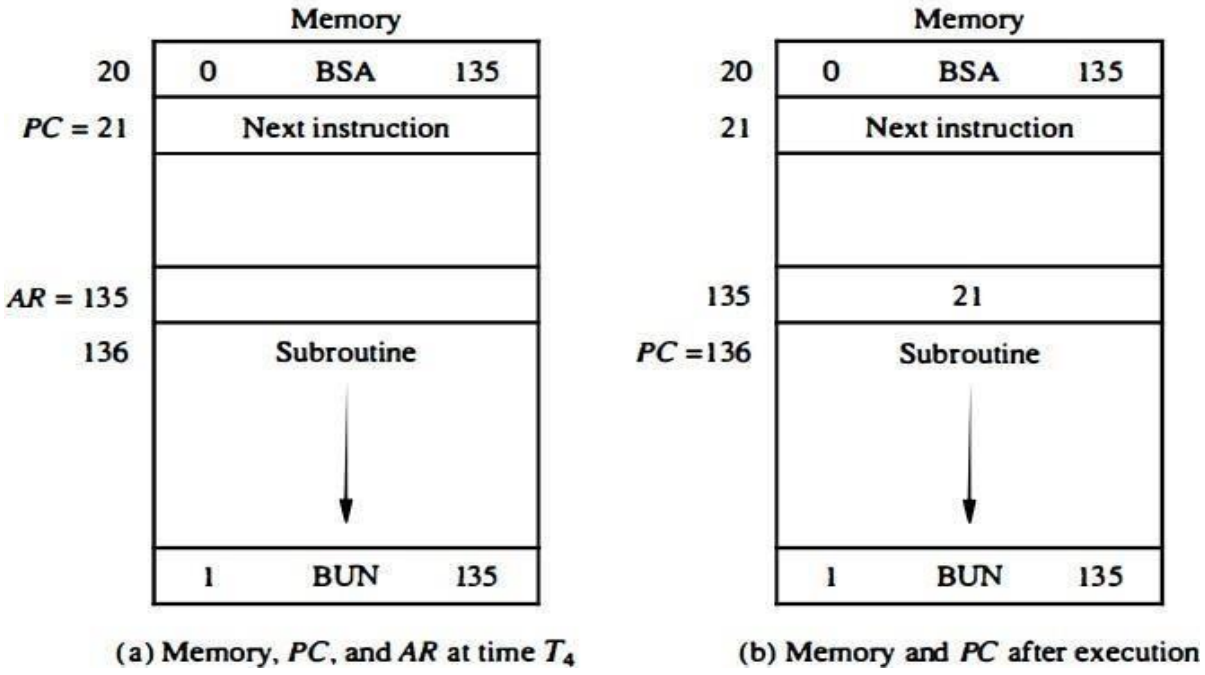
The effective address plus one is then transferred to PC to serve as the address of the first instruction in the subroutine.

This operation was specified in Table above (see Memory-Reference Instructions) with the following register transfer:

$$M[AR] \leftarrow PC, PC \leftarrow AR + 1$$

A numerical example that demonstrates how this instruction is used with a subroutine is shown in Fig. below.

Figure Example of BSA instruction execution.



The BSA instruction is assumed to be in memory at address 20. Th I bit is 0 and the address part of the instruction has the binary equivalent of 135.

After the fetch and decode phases, PC contains 21, which is the address of the next instruction in the program (referred to as the return address). AR holds the effective address 135.

This is shown in part (a) of the figure. The BSA instruction performs the following numerical operation:

$$M[135] \leftarrow 21, PC \leftarrow 135 + 1 = 136$$

The result of this operation is shown in part (b) of the figure. The return address 21 is stored in memory location 135 and control continues with the subroutine program starting from address 136.

The return to the original program (at address 21) is accomplished by means of an indirect BUN instruction placed at the end of the subroutine.

When this instruction is executed, control goes to the indirect phase to read the effective address at location 135, where it finds the previously saved address 21.

When the BUN instruction is executed, the effective address 21 is transferred to PC.

The next instruction cycle finds PC with the value 21, so control continues to execute the instruction at the return address.

ISZ: Increment and Skip if Zero

This instruction increment the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1.

The programmer usually stores a negative number (in 2's complement) in the memory word.

As this negative number is repeatedly incremented by one, it eventually reaches the value of zero.

At that time PC is incremented by one in order to skip the next instruction in the program.

Since it is not possible to increment a word inside the memory, it is necessary to read the word into DR, increment DR, and store the word back into memory.

This is done with the following sequence of micro operations:

D_6T_4: DR ← M [AR] D_6T_5:
DR ← DR + 1
D_6T_6: M[AR] ← DR, if (DR = 0) then (PC ← PC + 1), SC ← 0

Control Flowchart

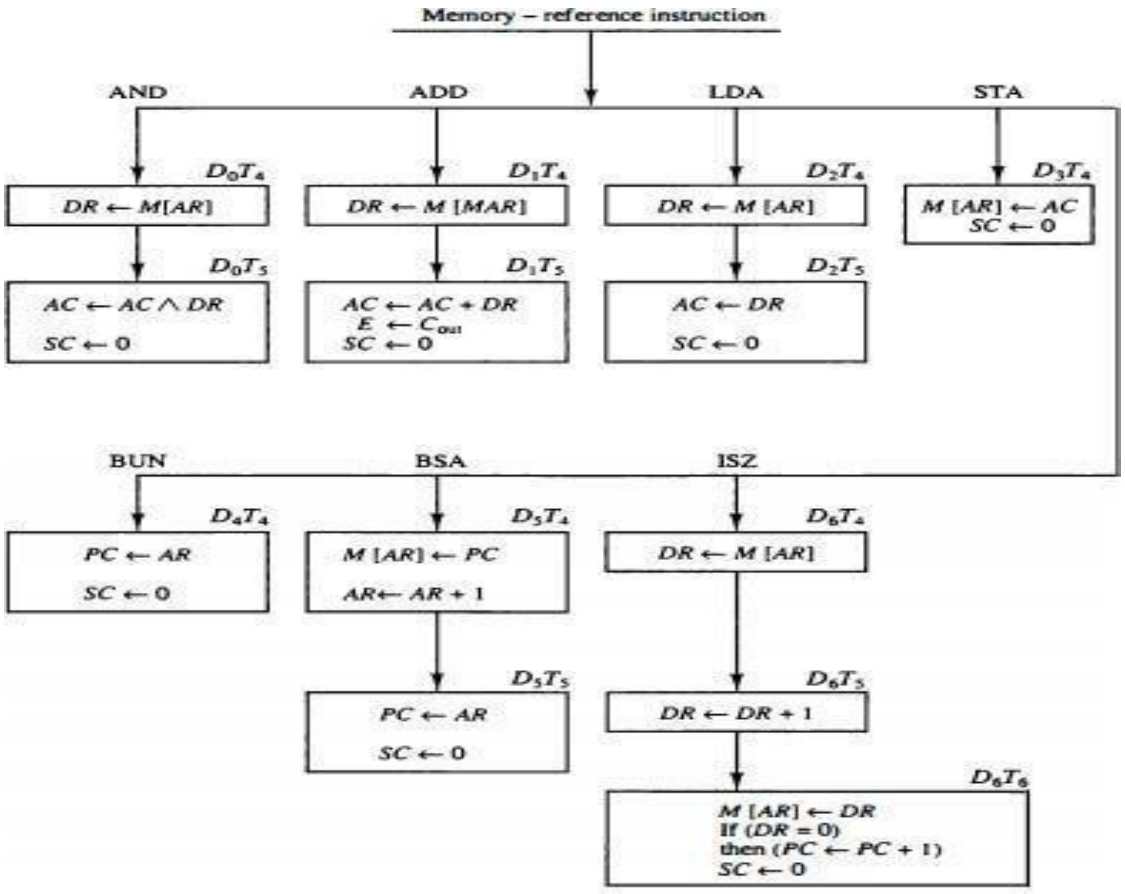


Figure Flowchart for memory-reference instructions.

The control functions are indicated on top of each box.

The micro operations that are performed during time T_4, T_5, or T_6 depend on the operation code value.

This is indicated in the flowchart by six different paths, one of which the control takes after the instruction is decoded.

The sequence counter SC is cleared to 0 with the last timing signal in each case.

This causes a transfer of control to timing signal T_0 to start the next instruction cycle.

Note that we need only seven timing signals to execute the longest instruction (ISZ).

Input-Output

A computer can serve no useful purpose unless it communicates with the external environment.

To demonstrate the most basic requirements for input and output communication, we will use as an illustration a terminal unit with a keyboard and printer.

Input-Output Configuration

The terminal sends and receives serial information. Each quantity of information has **eight bits of an alphanumeric code.**

The serial information from the keyboard is shifted into the input register INPR. The serial information for the printer is stored in the output register OUTR.

These two registers communicate with a communication interface serially and with the AC in parallel.

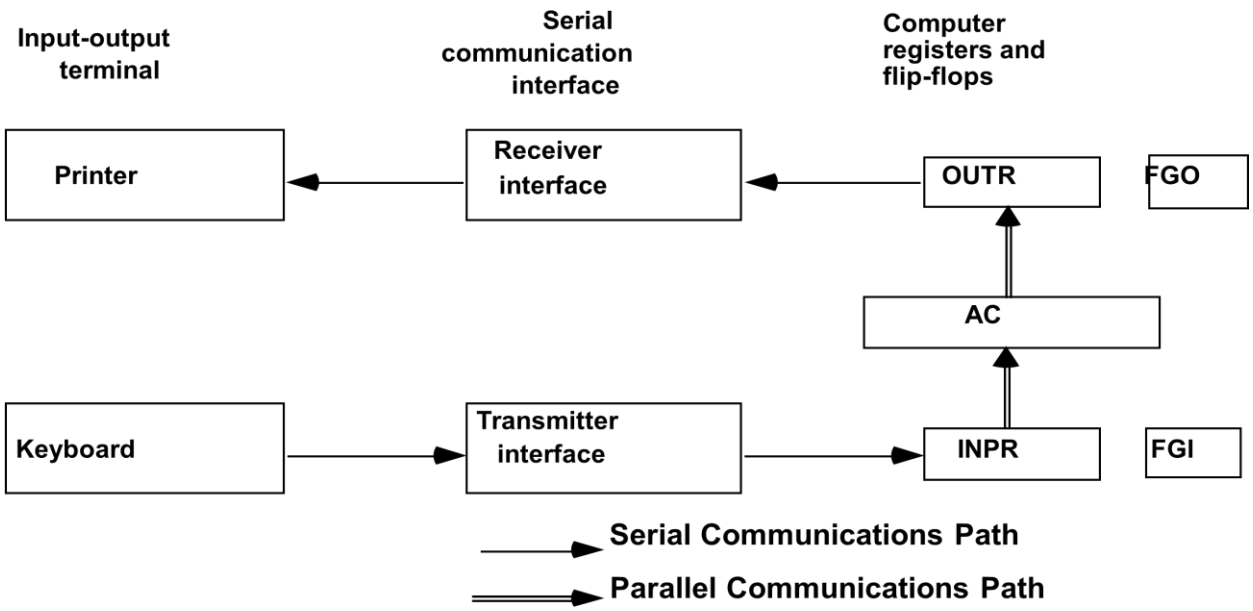
- The input-output configuration is shown in Fig. below.
- The transmitter interface receives serial information from the keyboard and transmits it to INPR.
- The receiver interface receives information from OUTR and sends it to the printer serially.
- The input register INPR consists of eight bits and holds alphanumeric input information.

The 1-bit input flag FGI is a control flip-flop.

The flag bit is set to 1 when new information is available in the input device and is cleared to 0 when the information is accepted by the computer.

The flag is needed to synchronize the timing rate difference between the input device and the computer.

Input –output Configuration



INPR **Input register - 8 bits**
OUTR **Output register - 8 bits**
FGI **Input flag - 1 bit**
FGO **Output flag - 1 bit**
IEN **Interrupt enable - 1 bit**

The process of information transfer is as follows. Initially, the input flag FGI is cleared to 0.

When a key is struck in the keyboard, an 8-bit alphanumeric code is shifted into INPR and the input flag FGI is set to 1.

As long as the flag is set, the information in INPR cannot be changed by striking another key.

The computer checks the flag bit FGI; if FGI= 1, the information from INPR is transferred in parallel into AC and FGI is cleared to 0.

Once the flag is cleared, new information can be shifted into INPR by striking another key.

The output register OUTR works similarly but the direction of information flow is reversed. Initially, the output flag FGO is set to 1.

The computer checks the flag bit FGO; if FGO=1, the information from AC is transferred in parallel to OUTR and FGO is cleared to 0.

The output device accepts the coded information, prints the corresponding character, and when the operation is completed, it sets FGO to 1.

The computer does not load a new character into OUTR when FGO is 0 because this condition indicates that the output device is in the process of printing the character.

Input-Output Instructions (D_7IT_3):

Input and output instructions are needed for **transferring information to and from AC register, for checking the flag bits, and for controlling the interrupt facility.**

Input-output instructions have an operation code 1111 and are recognized by the control when $D_7 = 1$ and I = 1. The remaining bits of the instruction specify the particular operation.

The control functions and micro operations for the input-output instructions are listed in Table below.

These instructions are executed with the clock transition associated with timing signal T_3.

Each control function needs a Boolean relation D_7IT_3, which we designate for convenience by the symbol **"p".**

The control function is distinguished by one of the bits **in IR (6-11).**

By assigning the symbol B_i to bit i of IR, all control functions can be denoted by **"pB$_i$ "**for i = 6 though 11.

The sequence counter SC is cleared to 0 when $p = D_7IT_3 = 1$.

Input-output Instructions

$D_7IT_3 = p$
$IR(i) = B_i,\ i = 6, \ldots, 11$

	p:	SC ← 0	Clear SC
INP	pB_{11}:	AC(0-7) ← INPR, FGI ← 0	Input char. to AC
OUT	pB_{10}:	OUTR ← AC(0-7), FGO ← 0	Output char. from AC
SKI	pB_9:	if(FGI = 1) then (PC ← PC + 1)	Skip on input flag
SKO	pB_8:	if(FGO = 1) then (PC ← PC + 1)	Skip on output flag
ION	pB_7:	IEN ← 1	Interrupt enable on
IOF	pB_6:	IEN ← 0	Interrupt enable off

The INP instruction transfers the input information from lNPR into the eight low- order bits of AC and also clears the input flag to 0.

The OUT instruction transfers the eight least significant bits of AC into the output registers OUTR and clears the output flag to 0.

The next two instructions in Table above **check the status of the flags** and cause a skip of the next instruction if the flag is 1.

The instruction that is skipped will normally be a branch instruction to return and check the flag again.

The branch instruction is not skipped if the flag is 0. If the flag is 1, the branch instruction is skipped and an input or output instruction is executed.

The last two instructions set and clear an interrupt enable flip flop IEN. The purpose of IEN is explained in conjunction with the interrupt operation

Interrupt cycle

The interrupt cycle is an hardware implementation of a branch and save return address.

The return address is available in PC is stored in a specific location where it can be found later when the program returns to the instruction at which it was interrupted.

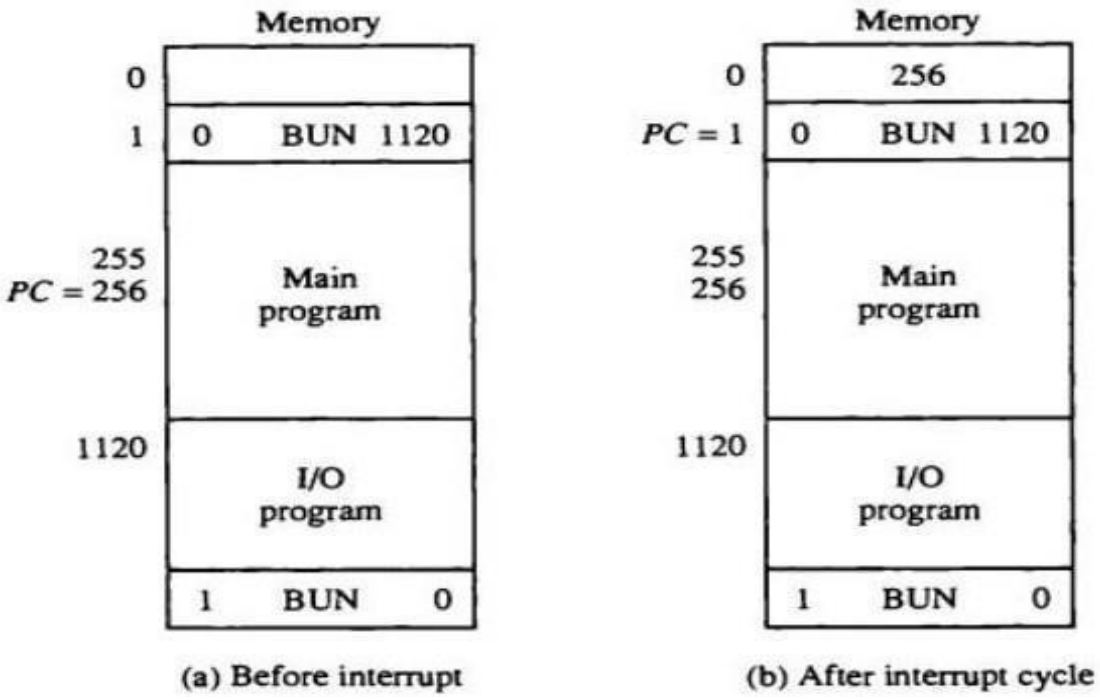
This location can be a processor register or a memory stack or a specific memory location.

Here we choose the **memory location to be 0 as the place for storing the return address;**

Control then inserts address 1 into PC and clears IEN and R so that no more interrupts can occur until the interrupt request from the flag has been received.

An example that shows what happens during the interrupt cycle is shown below

Demonstration of the interrupt cycle



Suppose if an interrupt has occurred then R is set to 1 while the control is executing the instruction at address 255.

At this time, the return address **256** is in PC.

The programme has previously placed an input-output service program in memory starting from address 1120 and a BUN 1120 instruction at address 1.

When control reaches timing signal T0 and finds that R=1, it proceeds with the interrupt cycle.

The content of PC (256) is stored in memory location 0, PC is set to 1, and R is cleared to 0.

At the beginning of the next instruction cycle, the instruction that is read from memory is in address 1, since this is the content of PC.

The **branch instruction at address1** causes the **program to transfer to the input –output service program at address 1120.**

This program checks the flags, determines which flag is set, and then transfers the required input or output information.

Once this is done, the instruction ION is executed to set IEN to 1 & the program returns to the location where it was interrupted.

The instruction that returns the computer to the original place in the main program is a **branch indirect instruction with an address part of 0**.

This instruction is placed at the I/O service program.

After this instruction is read from memory during the fetch phase, control goes to the indirect phase to read the effective address.

The effective address is in location 0 and is the return address that was stored there during the previous interrupt cycle.

The execution of the indirect BUN instruction results in placing into PC the return address from location 0.

Flow chart for Interrupt cycle

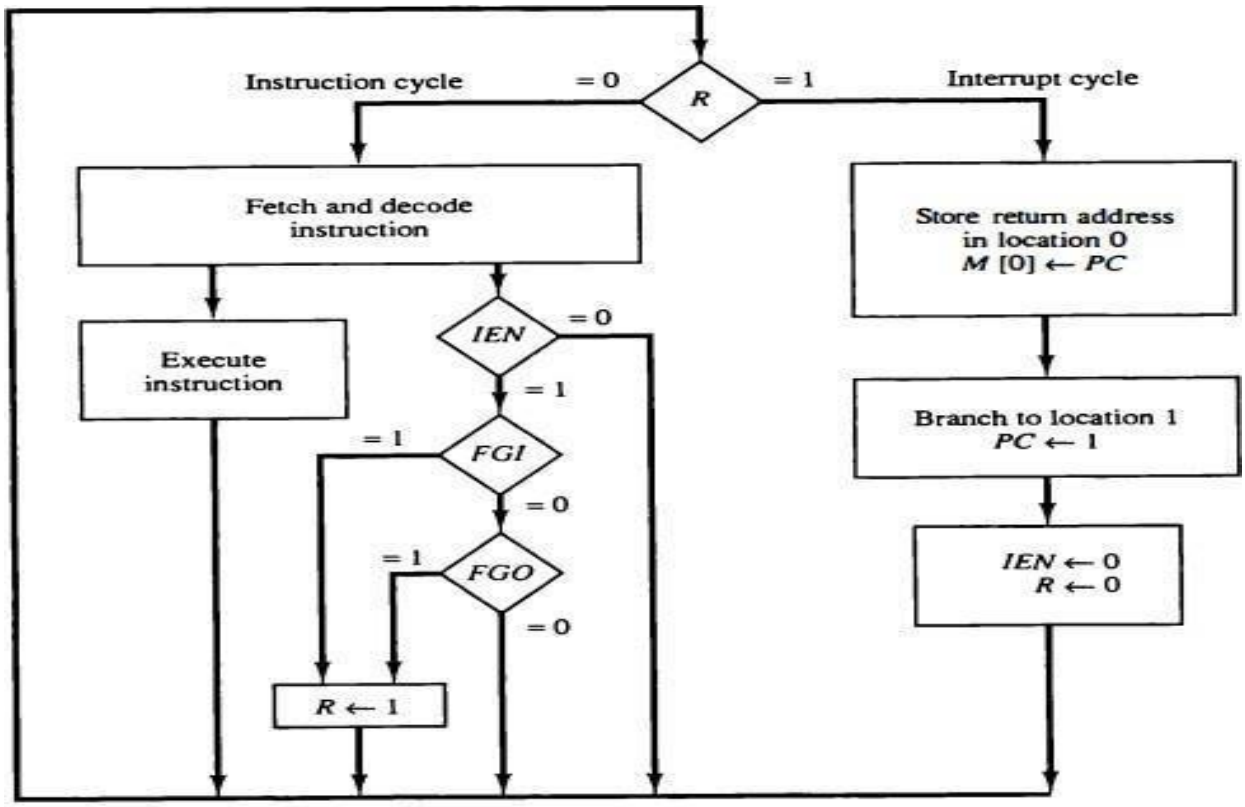


Figure Flowchart for interrupt cycle.

The way the interrupt is handled by the computer can be explained by means of the flow chart.

An **interrupt flip flop R** is included in the computer. When **R=0**; computer goes through an instruction cycle.

During the execution phase of the instruction cycle, **IEN** is checked by the control.

If it is 0 (IEN=0); it indicates the programmer does not want to use the interrupt. So control continues with the next instruction cycle.

If IEN=1; the control checks the flag bits (FGI &FGO) .

If both flags indicate 0 **(FGI=0 &FGO=0)**; it indicates that a neither the input nor the output registers are ready for transfer of information

In this case, control continues with the next instruction cycle.

If either flag (FGI or FGO) is set to1 while IEN=1; flipflop R is set to 1.

At the end of the execution phase, control checks the value of R,& if is equal to 1 **(R=1)** it goes to an interrupt cycle.

Interrupt Cycle (Register Transfer Notation)

The list of Register transfer operations in interrupt cycle is given below

The interrupt cycle is initiated after the last execute phase if the interrupt flip-flop R is equal to 1.

This flip-flop (R) is set to 1 if IEN=1 and either the FGI or FGO are equal to 1. This can happen with any clock transition except when timing signals T0, T1, T2 are active.

The condition for setting flipflop R to 1 can be expressed with the following register transfer statement

$$T0'T1'T2' (IEN) (FGI + FGO): R \leftarrow 1$$

The symbol + between FGI and FGO in the control function designate a logic OR operation. This is ANDed with IEN and T0'T1'T2'.

Modified fetch phase

The fetch and decode phases of the instruction cycle must be modified: Replace T0, T1, T2 with **R'T0, R'T1, R'T2.**

The reason for this is that after the instruction is executed and SC is cleared to 0, the control will go through a fetch phase only if R=0.

If R=1, the control will go through interrupt cycle.

The interrupt cycle stores the return address (PC) into memory location 0, branches to memory location 1, clears IEN,R and SC to 0.

This can be done with following sequence of micro operations:

RT0: AR ← 0, TR ←

PC RT1: M[AR] ←

TR, PC ← 0

RT2: PC ← PC + 1, IEN ← 0, R ← 0, SC ← 0

During the first timing signal AR is cleared to 0, the content of PC is transferred to the temporary register TR.

With the second timing signal, the return address is stored in memory at location 0 and PC is cleared to 0.

The third timing signal increments PC to1, clears IEN and R and control goes back to T0 by clearing SC to 0.

The beginning of the next instruction cycle has the condition RT0 and content of PC=1.

The control then goes through an instruction cycle that fetches and executes the BUN instruction in location 1.