JAWAHARLAL COLLEGE OF ENGINEERING AND TECHNOLOGY

(Approved by AICTE, Affiliated to APJ Abdul Kalam Technological University, Kerala)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(NBA Accredited)

COURSE MATERIAL

CST 202 COMPUTER ORGANIZATIONS AND ARCHITECTURE

VISION OF THE INSTITUTION

Emerge as a centre of excellence for professional education to produce high quality engineers and entrepreneurs for the development of the region and the Nation.

MISSION OF THE INSTITUTION

- To become an ultimate destination for acquiring latest and advanced knowledge in the multidisciplinary domains.
- To provide high quality education in engineering and technology through innovative teaching-learning practices, research and consultancy, embedded with professional ethics.
- To promote intellectual curiosity and thirst for acquiring knowledge through outcome based education.

- To have partnership with industry and reputed institutions to enhance the employability skills of the students and pedagogical pursuits.
- To leverage technologies to solve the real life societal problems through community services.

ABOUT THE DEPARTMENT

- Established in: 2008
- Courses offered: B.Tech in Computer Science and Engineering
- Affiliated to the A P J Abdul Kalam Technological University.

DEPARTMENT VISION

To produce competent professionals with research and innovative skills, by providing them with the most conducive environment for quality academic and research oriented undergraduate education along with moral values committed to build a vibrant nation.

DEPARTMENT MISSION

- Provide a learning environment to develop creativity and problem solving skills in a professional manner.
- Expose to latest technologies and tools used in the field of computer science.
- Provide a platform to explore the industries to understand the work culture and expectation of an organization.
- Enhance Industry Institute Interaction program to develop the entrepreneurship skills.
- Develop research interest among students which will impart a better life for the society and the nation.

PROGRAMME EDUCATIONAL OBJECTIVES

Graduates will be able to

- Provide high-quality knowledge in computer science and engineering required for a computer professional to identify and solve problems in various application domains.
- Persist with the ability in innovative ideas in computer support systems and transmit the knowledge and skills for research and advanced learning.
- Manifest the motivational capabilities, and turn on a social and economic commitment to community services.

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.

COURSE OUTCOMES

SUBJECT CODE: C210	
COURSE OUTCOMES	
C210.1	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
C210.2	To design processing unit using the concepts of ALU and control logic design.
C210.3	To select appropriate interfacing standards for I/O devices.
C210.4	To identify the pros and cons of different types of Memory systems and understand mapping functions.
C210.5	To select appropriate interfacing standards for I/O devices.
C210.6	To identify the roles of various functional units of a computer in instruction execution. And analyze the types of control logic design in processors.

PROGRAM SPECIFIC OUTCOMES (PSO)

The students will be able to

- Use fundamental knowledge of mathematics to solve problems using suitable analysis methods, data structure and algorithms.
- Interpret the basic concepts and methods of computer systems and technical specifications to provide accurate solutions.
- Apply theoretical and practical proficiency with a wide area of programming knowledge, design new ideas and innovations towards research.

CO PO MAPPING

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

CO'S	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
C210.1	3	2	2	-		-	-	-	-	-	-	2
C210.2	3	3	-	-	2	-	-	-	-	-	-	2
C210.3	3	2	-	-	2	-	-	-	-	-	-	2
C210.4	3	2	2	-		-	-	-	-	-	-	2
C210.5	3	3	3	-	-	-	-	-	-	-	-	-
C210.6	3	2	-	-	2	-	-	-	-	-	-	2
C210	3	2.3	2.3		2							2

CO PSO MAPPING

CO'S	PSO1	PSO2	PSO3
C210.1	3	2	2
C210.2	3	2	2
C210.3	3	2	2
C210.4	2	2	2
C210.5	2	3	2
C210.6	2	3	2
C210	2.5	2.3	2

S:NO	TOPIC
1	**Von Neumann architecture**
2	**Computer Organization and Architecture | Pipelining**

Reference Materials

MODULE - I

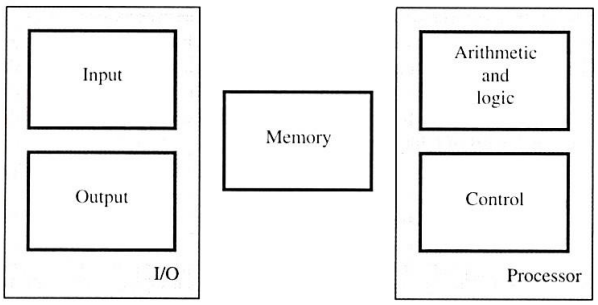
Basic Structure of computers – functional units - basic operational concepts - bus structures. Memory locations and addresses - memory operations, Instructions and instruction sequencing, addressing modes. Basic processing unit – fundamental concepts – instruction cycle – execution of a complete instruction - single bus and multiple bus organization

Computer Organization: It refers to the operational units and their interconnections that realize the architectural specifications.
The components from which computers are built, i.e., computer organization.
It describes the function of and design of the various units of digital computer that store and process information.

Computer Architecture: It is concerned with the structure and behaviour of the computer.
It includes the information formats, the instruction set and techniques for addressing memory.
Computer architecture is the science of integrating those components to achieve a level of functionality and performance.

Functional units of a Computer
A computer consists of 5 main parts.

1. Input
2. Memory
3. Arithmetic and logic
4. Output
5. Control Units



Basic functional units of a computer.

Input Unit:

- Computers accept coded information through input units, which read the data.
- Whenever a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted over a cable to either the memory or the processor.
- Some input devices are Keyboard, Joysticks Trackballs Mouse Microphones

Output Unit:

- Its function is to send the processed results to the outside world. eg.Printer
- Printers are capable of printing 10000 lines per minute but its speed is comparatively slower than the processor.

Memory Unit:

- It stores the programs and data.
- There are 2 types of storage classes
  Primary
  Secondary

Primary Storage:

- It is a fast memory that operates at electronic speeds.
- Programs must be stored in the memory while they are being executed.
- The memory contains large no of semiconductor storage cells.
- Each cell carries 1 bit of information.
- The Cells are processed in a group of fixed size called Words.
- To provide easy access to any word in a memory, a distinct address is associated with each word location.
- Addresses are numbers that identify successive locations.
- The number of bits in each word is called the word length.
- The word length ranges from 16 to 64 bits.

ALU:

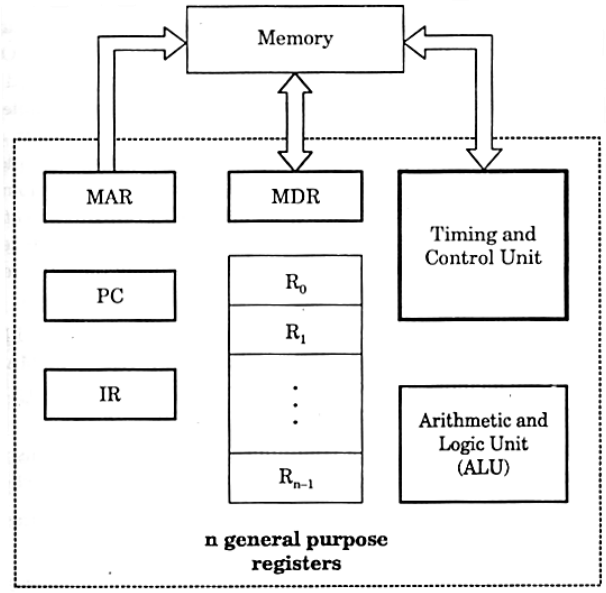
- Most computer operations are executed in ALU.
- Consider a example, Suppose 2 numbers located in memory are to be added. They are brought into the processor and the actual addition is carried out by the ALU. The sum may then be stored in the memory or retained in the processor for immediate use.
- Access time to registers is faster than access time to the fastest cache unit in memory.

Control Unit:

- The operations of Input unit, output unit, ALU are co-ordinate by the control unit.
- The control unit is the Nerve centre that sends control signals to other units and senses their states.
- Data transfers between the processor and the memory are also controlled by the control unit through timing signals.
- The operation of computers are,
  - The computer accepts information in the form of programs and data through an input unit and stores it in the memory.
  - Information stored in the memory is fetched, under program control into an arithmetic and logic unit, where it is processed.
  - Processed information leaves the computer through an output unit.
  - All activities inside the machine are directed by the control unit.

2. Basic operational concepts

To execute a given task as per the appropriate program,
Program consists of list of instructions stored in memory



The individual instructions are brought from the memory to the processor, which executes the specified operation.
In order to execute an operation in a processor the required instructions have to be brought out from the memory to the processor.
Transfers between memory and processor are started by sending the address of the memory location to be accessed to the memory unit and issuing the appropriate control signals.
Then the data are transferred to or from the memory.

Registers are fast stand-alone storage locations that hold data temporarily. Multiple registers are needed to facilitate the operation of the CPU. Some of these registers are

- Instruction register (IR) : Hold the Instructions that is currently being executed
- Program Counter (PC): It contains the memory address of the next instruction to be fetched for execution.
- **Memory Address Register (MAR)** and **Memory Data Register (MDR)**: These are the two registers which are facilitating the communication between the processor and memory.

  Read Operation : In order to read an information from memory **the address** of the memory location in which the information is residing have to be **put in MAR** and a **Read control signal** will be sent to memory unit. When the memory unit sees the address in address line of the bus and read signal in control line of the bus memory will start the read operation from the concerned address and the **result will be sent to the MDR**.

  Write operation : Initially the **data** to be written into the memory has to be **placed in MDR** and the **address** in which the desired information have to be kept in memory will be **placed in MAR** and a **Write control signal** will be sent to the memory unit. When the memory unit sees the address, data and writes signal in the external memory bus it will start the corresponding write operation.

Execution of instructions:

- The contents of PC transferred to MAR
- Select a particular memory location
- Issues RD control signals
- Reads instruction present in memory and loaded into MDR will be placed in IR (Contents transferred from MDR to IR)
- Instruction present in IR will be decoded by which processor understand what operation it has to perform
- Increments the contents of PC by 1, so that it points to the next instruction address
- If data required for operation is available in register, it performs the operation

Note : Normal execution of the program may be pre-empted if some device requires urgent servicing. In order to deal with the situation immediately, the normal execution of the current program may be interrupted.
The processor provides the requested service called the **Interrupt Service Routine (ISR).**
Contents of PC, general registers, and some control information are stored in memory.
When ISR is completed, the state of the processor is restored and the interrupted program may continue its execution.

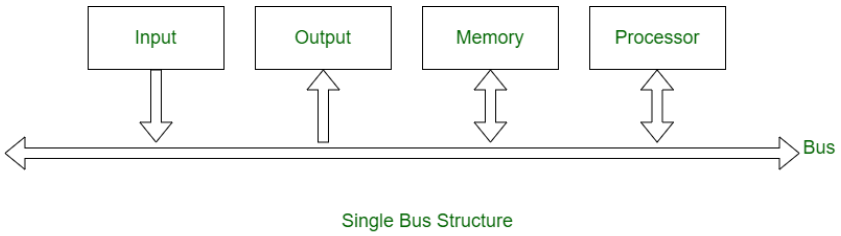
3. Bus Structure

A group of lines that serves as the connection path to several devices is called a Bus.

There are 2 types of Bus structures. They are

- Single Bus Structure
- Multiple Bus Structure

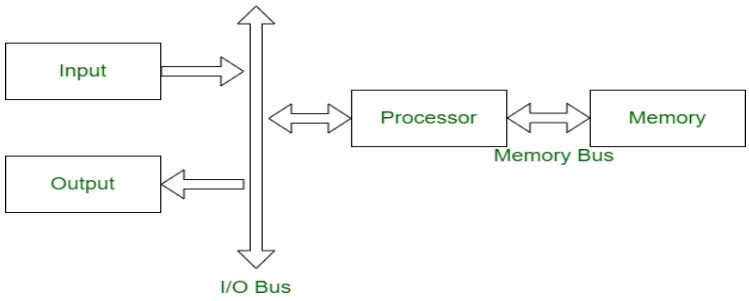
Single Bus Structure



- Common bus used to communicate between peripherals and microprocessor
- It allows only one transfer at a time.
- It costs low.
- It is flexible for attaching peripheral devices.
- Its Performance is low.

Multiple Bus Structure

It is to overcome the bottleneck of single bus structure.



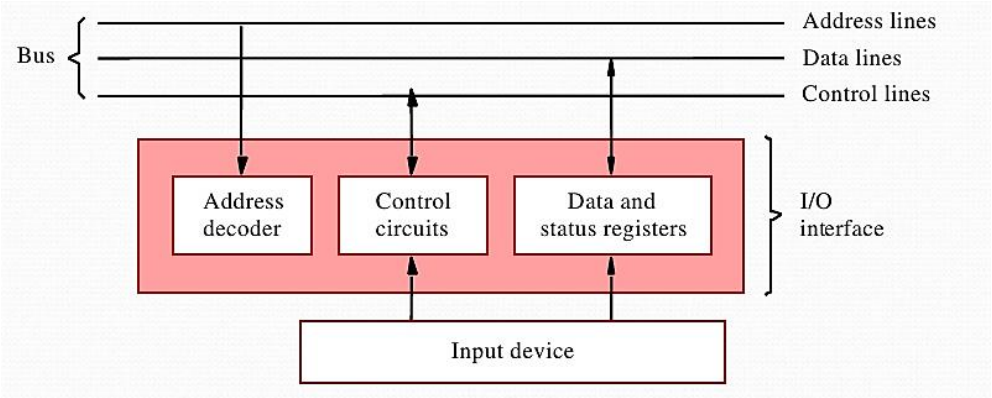
Double Bus Structure

- To improve performance multibus structure can be used
- It allows two or more transfer at a time.
- It costs high.
- It provides concurrency in operation.
- Its Performance is high.
- In two – bus structure: One bus can be used to fetch instruction other can be used to fetch data, required for execution, thus improving the performance, but cost increases.

Single Bus Structure	Double Bus Structure
One common bus is used for communication between peripherals and processor.	Two buses are used, one for communication from peripherals and other for processor.
Instructions and data both are transferred in same bus.	Instructions and data both are transferred in different buses.
Its performance is low.	Its performance is high.
Cost of single bus structure is low.	Cost of double bus structure is high.
Number of cycles for execution is more.	Number of cycles for execution is less.
Execution of process is slow.	Execution of process is fast.
Number of registers associated is less.	Number of registers associated is more.
At a time single operand can be read from bus.	At a time two operands can be read.

The bus consists of three set of lines used to carry address, data and control signals known as *data bus*, *address bus* and *control bus*.

- **Address bus:** *unidirectional:* group of wires which carries *address information bits* form processor to peripherals.
- **Data bus:** *bidirectional* : group of wires which carries *data information bit* form processor to peripherals and vice – versa
- **Control bus:** *bidirectional:* group of wires which carries *control signals* form processor to peripherals and vice – versa



- Above figure shows the input device interface to the bus.
- Each I/O device is assigned a unique set of addresses for the registers in its interface.
- When the processor places a particular address on the address lines, it is examined by the address decoders of all devices on the bus.
- The device that recognizes this address responds to the commands issued on the control lines.
- The processor uses the control lines to request either a Read or a Write operation, and the requested data are transferred over the data lines.

4. Memory locations and addresses

- **Main memory** is the second major subsystem in a computer. It consists of a collection of storage locations, each with a unique identifier, called an **address**.
- Data is transferred to and from memory in groups of bits called **words**. A word can be a group of 8 bits, 16 bits, 32 bits or 64 bits (and growing).

Memory units

Unit	*Exact Number of Bytes*	*Approximation*
kilobyte	2^{10} (1024) bytes	10^3 bytes
megabyte	2^{20} (1,048,576) bytes	10^6 bytes
gigabyte	2^{30} (1,073,741,824) bytes	10^9 bytes
terabyte	2^{40} bytes	10^{12} bytes

- A computer has 32 MB (megabytes) of memory. How many bits are needed to address any single byte in memory?
  - The memory address space is 32 MB, or 2^{25} ($2^5 \times 2^{20}$). This means that we need $\log_2 2^{25}$, or **25 bits**, to address each byte.
- A computer has 128 MB of memory. Each word in this computer is eight bytes. How many bits are needed to address any single word in memory?
  - The memory address space is 128 MB, which means 2^{27}. However, each word is eight (2^3) bytes, which means that we have 2^{24} words. This means that we need $\log_2 2^{24}$, or **24 bits**, to address each word.

Assignment of byte addresses:

- Little Endian: low order byte stored at lowest address
  - Eg: CD12AB90 (32 bit data)

H BYTE ← L BYTE

Address	Value
1000	90
1001	AB
1002	12
1003	CD

  - e.g., e.g., in DEC, Intel
- Big Endian: high order byte stored at lowest address
  - Eg: CD12AB90 (32 bit data)

Address	Value
1000	CD
1001	12
1002	AB
1003	90

5. Memory operations

- Both program instructions and data operands are stored in the memory.
- To execute an instruction, the instruction to be transferred from the memory to the processor.
- Operands and results must also be moved between the memory and the processor.
- Two basic operations requires in memory access
  - Load operation (Read or Fetch):
    - The Load operation transfers a copy of the contents of a specific memory-location to the processor.
    - The memory contents remain unchanged.
    - Steps for Load operation:

      1) Processor sends the address of the desired location to the memory.

      2) Processor issues „read" signal to memory to fetch the data.

      3) Memory reads the data stored at that address.

      4) Memory sends the read data to the processor.

- Store operation (Write):
  - The Store operation transfers the information from the register to the specified memory-location.
  - This will destroy the original contents of that memory-location.
  - Steps for Store operation are:

    1) Processor sends the address of the memory-location where it wants to store data.

    2) Processor issues „write" signal to memory to store the data.

    3) Content of register (MDR) is written into the specified memory-location.

6. Instructions and instruction sequencing

- A computer must have instruction capable of performing the following operations. They are
  - Data transfer between memory and processor register.
  - Arithmetic and logical operations on data.
  - Program sequencing and control.
  - I/O transfer.
- Complete instruction set of the processor is called instruction set architecture.
- The possible locations in which transfer of information occurs are,
  - Memory Location (LOC,PLACE,MEM are the address of memory location)
  - Processor register (R1 , R2,… are processor registers)
  - Registers in I/O sub-system (DATA_IN, DATA_OUT are I/O registers)
- Contents of location is indicated by using square brackets [ ]
- Assembly Language Notation
  - Use Mnemonics
    - Eg:

      MOVE LOC, R1 - Transfers the contents of memory location to the processor register R1

      ADD R1,R2,R3 - Add the contents of register R1 & R2 and places their sum into register R3.
- **Type of Instructions**
  - *Three address instruction*
    - Syntax: Operation source 1, source 2, destination
    - Eg: ADD D,E,F where D,E,F are memory location
    - Advantage: Single instruction can perform the complete operation
    - Disadvantage : Instruction code will be too large to fit in one word location in memory

- *Two Address Instruction*
  - Syntax : Operation source, destination
    - Eg: MOVE E,F
      ADD D, F
  - Perform ADD A,B,C using 2instructions

    MOVE B, C

    ADD A, C

    Disadvantage: Single instruction is not sufficient to perform the entire operation.
- *One Address Instruction*
  - Syntax- Operation source/destination
  - In this type either a source or destination operand is mentioned in the instruction
  - Other operand is implied to be a processor register called **Accumulator**
  - Eg: ADD B
  - Perform ADD D,E,F

    Load D
    ADD E
    STORE F
- Zero address instruction
  - Syntax : Operation source
  - Location of all operands are defined implicitly
  - Ex: CLA is a 0-address instruction, CLA stands for clear accumulator.

- Branch instruction is those which change the normal sequence of execution.
  - We have conditional branch instructions and unconditional branch instruction.
  - Unconditional branch instruction changes the sequence of execution irrespective of condition of the results.
    - Example : JMP 2000 Jumps to the address 2000
  - Conditional branch instruction changes the sequence only when certain conditions are met.
    - Example :

      JNC 3000 Jumps to the address if carry flag is 0
      JZ 2000 Jumps to the address if zero flag is 1
  - **Conditional Code Flags**: The processor keeps track of information about the results of various operations for use by subsequent conditional branch instructions

    - N – Negative 1 if results are Negative
      0 if results are Positive
    - Z – Zero 1 if results are Zero
      0 if results are Non zero
    - V – Overflow 1 if arithmetic overflow occurs
      0 non overflow occurs
    - C – Carry 1 if carry and from MSB bit
      0 if there is no carry from MSB bit

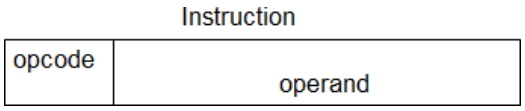
7. Addressing Modes

The different ways in which the location of the operand is specified in an instruction are referred to as addressing modes

1. Immediate Addressing
2. Direct Addressing
3. Indirect Addressing
4. Register Addressing
5. Register Indirect Addressing
6. Indexed Addressing
7. Relative Addressing

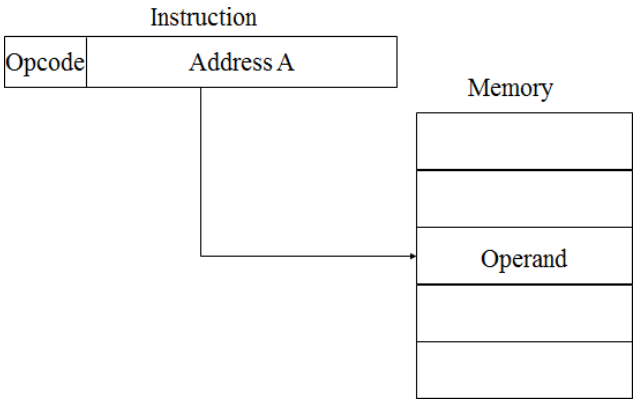
Immediate Addressing

- Operand is given explicitly in the instruction
- Operand = Value
- e.g. ADD 5
  - Add 5 to contents of accumulator
  - 5 is operand
- No memory reference to fetch data
- Fast
- Limited range



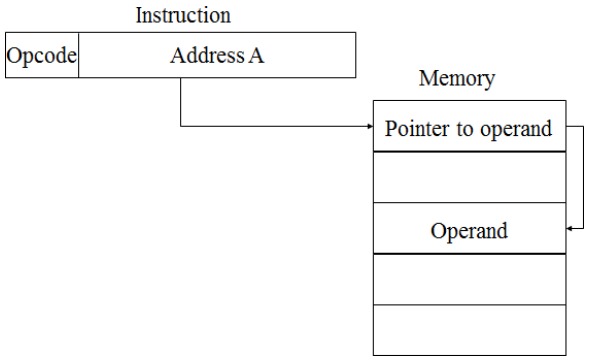
Direct Addressing

- Address field contains address of operand
- Effective address (EA) = address field (A)
- e.g. ADD A
  - Add contents of cell A to accumulator
  - Look in memory at address A for operand
- Single memory reference to access data
- No additional calculations to work out effective address
- Limited address space



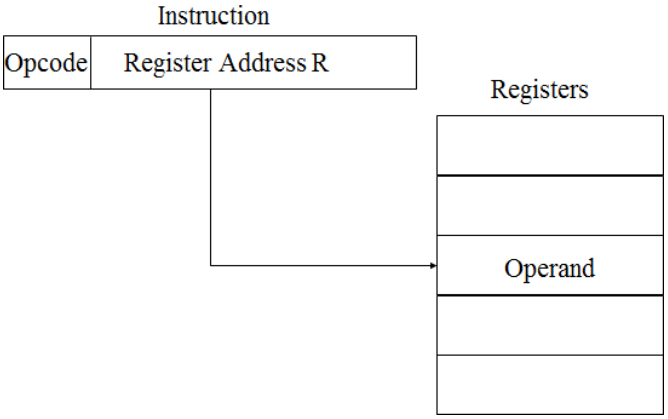
Indirect Addressing

- Memory cell pointed to by address field contains the address of (pointer to) the operand
- EA = [A]
  - Look in A, find address (A) and look there for operand
- e.g. ADD (A)
  - Add contents of cell pointed to by contents of A to accumulator
- Large address space
- 2^n where n = word length
- May be nested, multilevel, cascaded
  - e.g. EA = (((A)))
    - Draw the diagram yourself
- Multiple memory accesses to find operand
- Hence slower



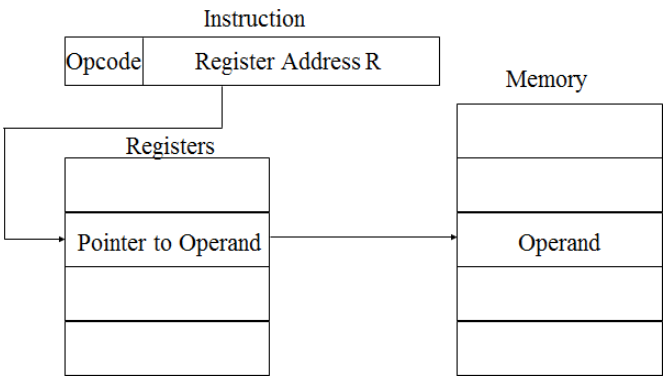
Register Addressing

- Operand is held in register named in address field
- EA = R
- Limited number of registers
- Very small address field needed
  - Shorter instructions
  - Faster instruction fetch
- No memory access
- Very fast execution
- Very limited address space
- Multiple registers helps performance
- Requires good assembly programming or compiler writing



Register Indirect Addressing

- C.f. indirect addressing
- EA = [R]
- Operand is in memory cell pointed to by contents of register R
- Large address space (2^n)
- One fewer memory access than indirect addressing
-

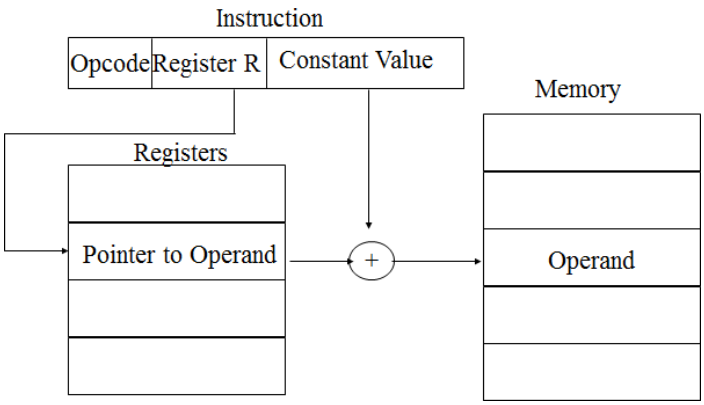


Indexed Addressing

- EA = X + [R]
- Address field hold two values
  - X = constant value (offset)
  - R = register that holds address of memory locations
  - or vice versa

  (Offset given as constant or in the index register)

  Add 20(R1),R2 or Add 1000(R1),R2



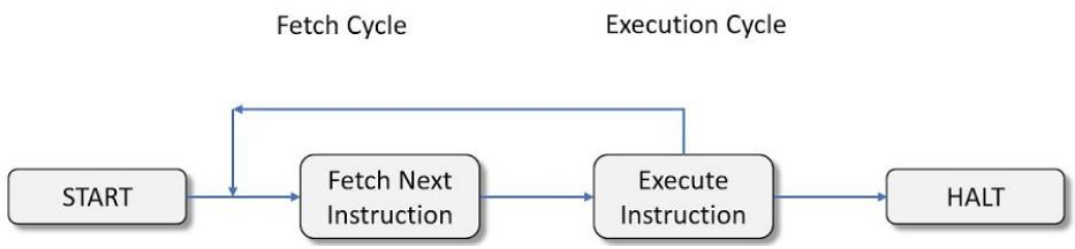
Relative Addressing

- A version of displacement addressing
- R = Program counter, PC
- EA = X + (PC)
- i.e. get operand from X bytes away from current location pointed to by PC.

8. Instruction cycle

- The basic function performed by a computer is execution of a program, which consists of a set of instructions stored in memory.
- The processor does the actual work by executing instructions specified in the program.
- In the simplest form, instruction processing consists of two steps: the processor reads (fetches) instructions from memory one at a time and executes each instruction.
- The processing required for a single instruction is called an *instruction cycle.*

  An instruction cycle is shown below:



- Program execution halts only if the machine is turned off, some sort of unrecoverable error occurs, or a program instruction that halts the computer is encountered.
- Instruction Fetch and Execute
  - The processor fetches an instruction from memory – program counter (PC) register holds the address of the instruction to be fetched next.
  - The processor increments the PC after each instruction fetch so that it will fetch the next instruction in the sequence.
  - The fetched instruction is loaded into the instruction register (IR) in the processor – the instruction contains bits that specify the action the processor will take.

9. Single Bus Organization

- In this section we are dealing with how the instructions are processed within the processing unit.
- The ALU registers and the internal processor bus together known as datapath.

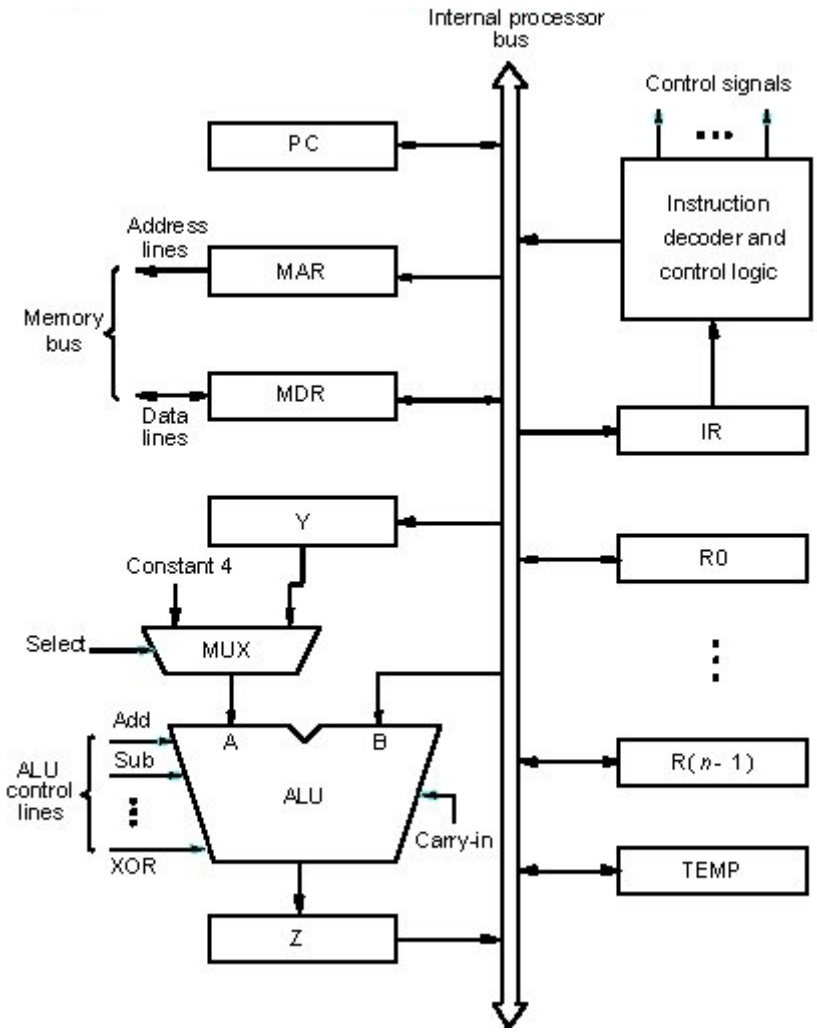


Figure: Single-bus organization of the datapath inside a processor

Normally in the instruction execution following operations may be happen in different sequences:

- Register transfer
- Performing an arithmetic and logic operation
- Fetching a word from memory
- Storing a word in memory

Register transfer :

Example : Transfer the contents of register R1 to R4.

- Set R_{1out} to 1. This places the contents of R1 to the bus.
- set R_{4in} to 1. This loads data from the processor bus to register R4

Performing an arithmetic and logic operation

Example : SUB R4,R5,R6 Subtract the contents of R4 from R5 and store the result in R6.

- R_{4out}, Yin
- R_{5out} ,Select Y ,sub, Z_{in}
- Zout , R6in

Fetching a word from memory

Example: MOVE [R1] , R2

R1out, MARin, READ

MDR_{inE}, WMFC

MDRout, R2in

Storing a word in memory

Example: MOVE R2, (R1)

R1out,MARin

R2out, MDRin, WRITE

MDRout, WMFC

Execution of a complete instruction

Example : Add R2, R1

- Fetch the instruction
- Fetch the operands
- Perform the addition
- Load the result into R1

Step	Action
1	PC_{out} , MAR_{in} , Read, Select4,Add, Z_{in}
2	Z_{out} , PC_{in} , Y_{in} , WMF C
3	MDR_{out} , IR_{in}
4	$R1_{out}$, Y_{in}
5	$R2_{out}$, SelectY, Add, Z_{in}
6	Z_{out} , $R1_{in}$, End

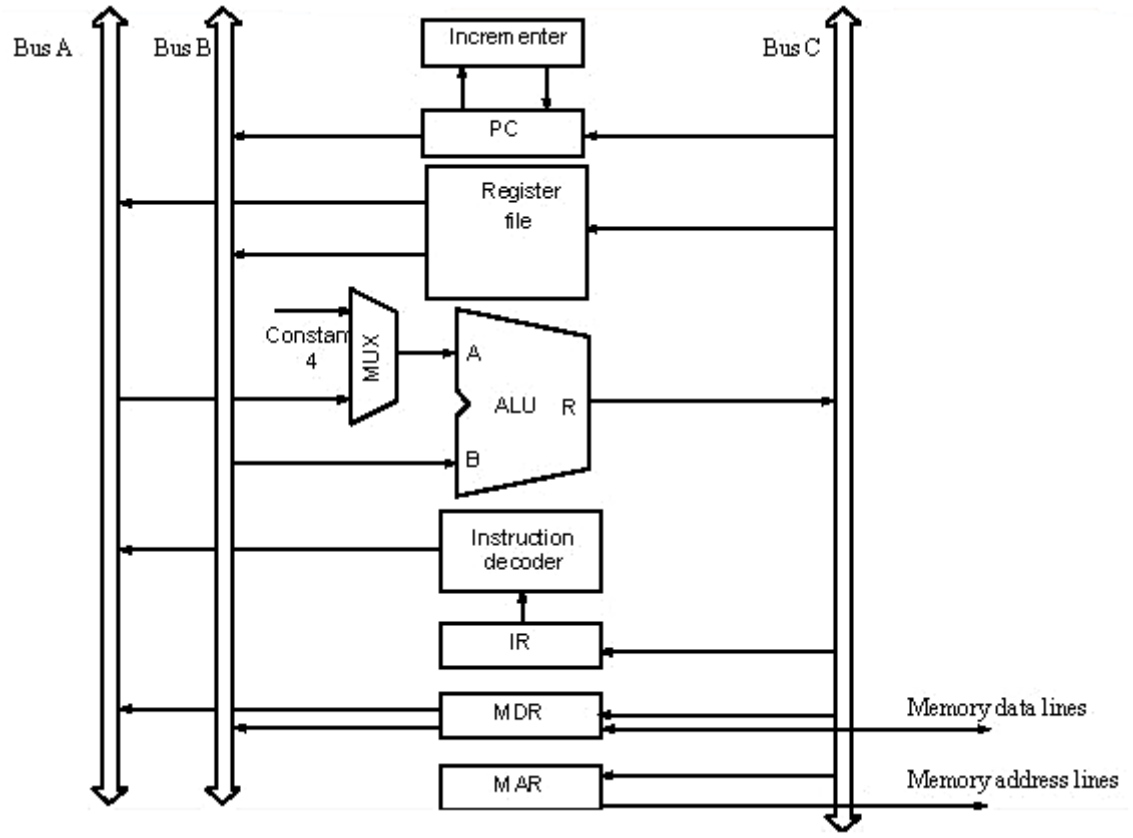
Example : Add (R3), R1

- Fetch the instruction
- Fetch the operands
- Perform the addition
- Load the result into R1

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

10. Multiple Bus Organization

- Multiple bus more than one data word can be transferred over the bus in a clock cycle. This increases the steps required to complete the execution of the instruction
- To reduce the number of steps needed to execute instructions, most commercial process provide multiple internal paths that enable several transfer to take place in parallel



- 3 buses are used to connect registers and the ALU of the processor.
- All general purpose registers are shown by a single block called register file.
- There are 3 ports, one input port and two output ports. So it is possible to access data of three register in one clock cycle, the value can be loaded in one register from bus C and data from two register can be accessed to bus A and bus B.

- **Execution of a complete instruction**

Example : Add R4, R5, R6

Step	Action
1	PC_{out}, R=B, MAR_{in}, Read, IncPC
2	WMFC
3	MDR_{outB}, R=B, IR_{in}
4	$R4_{outA}$, $R5_{outB}$, SelectA, Add, $R6_{in}$, End

MODULE – II

Register transfer logic: *inter register transfer – arithmetic, logic and shift micro operations.*
Processor logic design: - *processor organization – Arithmetic logic unit - design of arithmetic circuit - design of logic circuit - Design of arithmetic logic unit - status register – design of shifter - processor unit – design of accumulator.*

REGISTER TRANSFER LOGIC:

- Digital system is a collection of digital hardware modules.
- A digital system is a sequential logic system constructed with flip flops and gates. The sequential circuit can be specified by means of a state table.
- Specifying a large digital system with a state table would be very difficult, since the number of states would be very large.
- To overcome this difficulty, digital systems are designed using a modular approach, where each modular subsystem performs some functional task.
- The modules are constructed from such digital functions as registers, counters, decoders, multiplexers, arithmetic elements and control logic.
- Various modules are interconnected with data and control path.
- The interconnection of digital functions cannot be described by means of combinational or sequential logic techniques.
- The information flow and the processing task among the data stored in the registers can be described by means of **register transfer logic**.
- The registers are selected as primitive components of the system. Register transfer logic uses a set of expressions and statements which compare the statements used in programming language.
- It provides the necessary tool for specifying the interconnection between various digital functions.

➢ Components of Register Transfer Logic

1. ***The set of registers in the system and their functions:*** A register also encompasses all type of registers including shift registers, counters and memory units.
2. ***The binary-coded information stored in the registers:*** The binary information stored in registers may be binary numbers, binary coded decimal numbers, alphanumeric characters, control information or any other binary coded information.
3. ***The operations performed on the information stored in the registers:*** The operations performed on data stored in registers are called micro operations. Examples are shift, count, add, clear and load
4. ***The control functions that initiate the sequence of operations:*** The control functions that initiate the sequence of operations consists of timing signals that sequence the operations one at a time.

Register transfer language (Computer hardware description language)

- Symbolic notation used for registers, for specifying operations on the contents of registers and specifying control functions.
- A statement in a register transfer language consists of control function and a list of microoperations.
- **Micro-Operation:** Operations performed in data stored in registers. Elementary operation that can be performed parallel during one clock pulse period.
- The result of operation may replace the previous binary information of a register or may be transferred to another register. Example: Shift, count, clear, add & load
- A micro-operation requires one clock pulse for the execution if the operation done in parallel. In serial computer a microoperation requires a number of clock pulses equal to the word time in the system.

Types of Micro-Operations in digital system

1. **Inter register transfer micro-operation:** Do not change the information content when the binary information moves from one register to another
2. **Arithmetic operation:** Perform arithmetic on numbers stored in registers.
3. **Logic microoperation:** Perform operations such as AND and OR on individual pairs of bits stored in registers.
4. **Shift microoperation:** Specify operations for shift registers.

Inter Register Transfer

- Registers are the primitive component.
- Computer registers are designated by capital letters (sometimes followed by numerals) to denote the function of the register.

  Example:

  R1 - Processor Register,
  MAR - Memory Address Register
  PC - Program Counter,
  IR - Instruction Register,
  SR: Status Register
- The cells or flip-flops of n-bit register are numbered in sequence from1 to n (from 0 to n-1) starting either from left or from right.

The register can be represented in 4 ways:

- Rectangular box with name of the register inside
- The individual cells is assigned a letter with a subscript number
- The numbering of cells from right to left can be marked on top of the box as the 12 bit register Memory Buffer Register (MBR).
- 16 bit register is partitioned into 2 parts , bits 1 to 8 are assigned the letter L(for low) and bits 9 to 16 are assigned the letter H(for high)

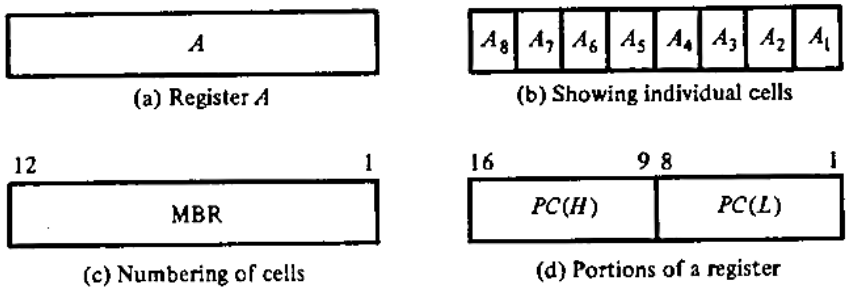


Fig: Register Representations

– Information transfer from one register to another is described by a **replacement operator**:

$$A \leftarrow B.$$

– This statement denotes a transfer of the content of register B into register A and this transfer happens in one clock cycle.

– After the operation, the content of the B (source) does not change. The content of the A (destination) will be lost and replaced by the new data transferred from B.

➢ *Conditional transfer occurs only under a control condition:*

– The condition that determines when the transfer is to occurs called a **control function.**

– A control function is a Boolean function that can be equal to 1 or 0.

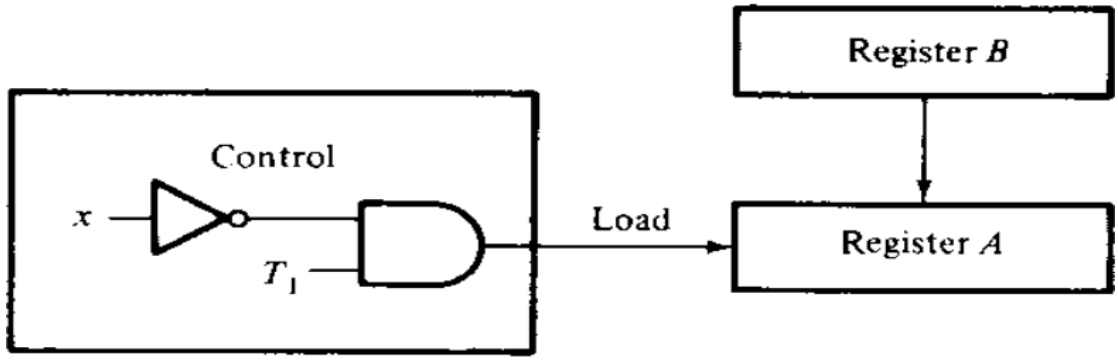
– The control function is included with the statement as follows

$$\mathbf{x' T_1: A \leftarrow B}$$

– The control function is terminated with a colon.

– It symbolizes the requirement that the transfer operation be executed by the hardware only when the Boolean function $x'T_1 = 1$. ie; when variable x = 0 and timing variable $T_1 = 1$.

- Hardware implementation of a controlled transfer: ***x'T₁: A ← B*** is as follows



- The outputs of register B are connected to the input of register A and the number of lines in this connection is equal to the number of bits in the registers.
- Register A must have a load control input so that it can be enabled when the control function is 1.
- It is assumed that register A has an additional input that accepts continuous synchronized clock pulses.
- The control function is generated by means of an inverter and an AND gate.
- It is also assumed that the control unit that generates the timing variable T_1 i synchronized with the same clock pulses that are applied to register A.
- The control function stays on during one clock pulse period (when the timing variable is equal to 1) and the transfer occurs during the next transaction of a clock pulse.
- Destination register receives information from two sources but not at the same time.

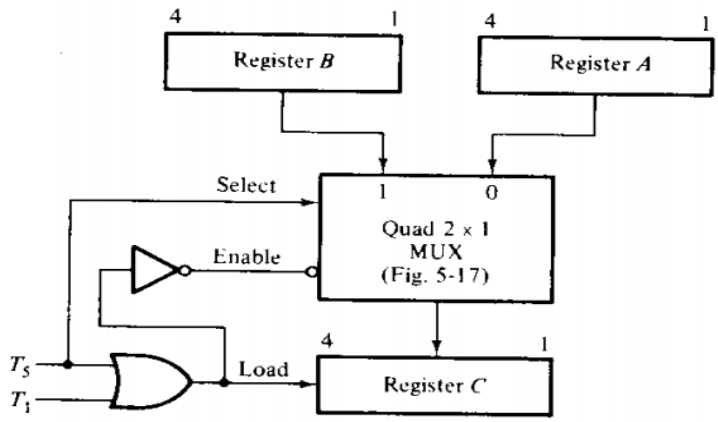
- Consider an example

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 T_1 occurs.
- The second statement uses the same destination register C as the first but with a different source register and a different timing variable.
- The connections of two source registers to the same destination register cannot be done directly but requires a multiplexer circuit to select between the two possible paths.

– The block diagram of the circuit that implements the two statements is shown in the figure.



– For registers with four bits each, we need a quadruple 2 to 1 line multiplexer in order to select either A or B. When T_5 =1, register B is selected but when T_1=1, register A is selected (because T_5 must be 0 when T_1 is 1).

– The multiplexer and the load input of register C are enabled every time T_1 and T_5 occur.

– This causes a transfer of information from the selected source register to destination register.

The basic symbols of Register Transfer Logic are

Symbol	Description	Examples
Letter (and Numerals)	Denotes a Register	A, MDR, R2
Subscript	Denotes a bit of a Register	A_2, B_6
Parenthesis ( )	Denotes a portion of Register	PC(H), MBR (OP)
Arrow ←	Denotes transfer of information	A ← B
Colon :	Terminates a control function	$X'T_0$:
Comma	Separates two micro-operations	A ← B, B ← A
Square Brackets [ ]	Specifies an address for memory transfer	MBR ← M [MAR]

➢ Conditional control statement

– Specify a control condition by a conditional statement rather than a Boolean control function

– Symbolised by if-then-else statement

P: If (condition) then [microoperation(s)] else [microoperation(s)]

– As an example, consider the conditional control statement:

T2: If(C=0) then (F←1) else (F←0)

F is assumed to be 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 statements

$$C'T_2 : F \leftarrow 1$$
$$CT_2 : F \leftarrow 0$$

➢ Arithmetic Micro-Operations

The basic arithmetic micro-operations are:

- Addition
- Subtraction
- Increment
- Decrement
- Arithmetic shift

– The increment and decrement micro-operations are implemented with a combinational circuit or with a binary up-down counter as these micro-operations use plus-one and minus-one operation respectively.

– The arithmetic add microoperations are defined by the statement

$$F \leftarrow A + B$$

– It states that the contents of register A are to be added to the contents of register B and the sum is transferred to register F

– To implement this statement require 3 registers A, B and F and a digital function that performs the addition operation such as parallel adder.

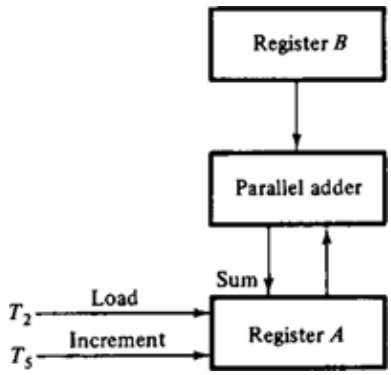
Symbolic designation	Description
$F \leftarrow A + B$	Contents of A plus B transferred to F
$F \leftarrow A - B$	Contents of A minus B transferred to F
$B \leftarrow \overline{B}$	Complement register B (1's complement)
$B \leftarrow \overline{B} + 1$	Form the 2's complement of the contents of register B
$F \leftarrow A + \overline{B} + 1$	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)

– There must be a direct relationship between the statements written in a register transfer language and the registers and digital functions which are required for the implementation

– Consider the statements

T2: A ← A + B

T5: A ← A + 1



– Timing variable T2 initiates an operation to add the contents of register B to the present contents of A with a parallel adder.
– Timing variable T5 increments register A with a counter.
– The transfer of the sum from parallel adder into register A can be activated with a load input in the register.
– Register be a counter with parallel load capability.
– The parallel adder receives input information from registers A and B.
– The sum bits from the parallel adder are applied to the inputs of A and timing variable T2 loads the sum into register A.
– Timing variable T5 increments there by enabling increment input register.
– Two basic arithmetic operations (multiplication and divide) are not included in the basic set of micro-operations.
– They are implemented by means of a combinational circuit.
– In general, the multiplication micro-operation is implemented with a sequence of add and shift micro- operations.
– Division is implemented with a sequence of subtract and shift micro-operations.

➢ **Logic Micro-Operations**

– Logic micro-operations specify binary operations for strings of bits stored in registers.
– These operations consider each bit of the register separately and treat them as binary variables. For example, the exclusive-OR micro-operation with the contents of two registers A and B is symbolized by the statement

$$F \leftarrow A \oplus B$$

- It specifies a logic micro-operation that considers each pair of bits in the registers as a binary variable.
- Let the content of register A is 1010 and the content of register B is 1100. The exclusive-OR micro-operation stated above symbolizes the following logic computation:

$$\begin{array}{ll} 1010 & \text{Content of A} \\ \underline{1100} & \text{Content of B} \\ 0110 & \text{Content of } F \leftarrow A \oplus B \end{array}$$

- The content of F, after the execution of the micro-operation, is equal to the bit-by-bit exclusive-OR operation on pairs of bits in B and values of A.
- The logic micro-operations are seldom used in scientific computations, but they are very useful for bit manipulation of binary data and for making logical decisions.

Table: 16 different logic operations with 2 binary variables

Boolean Functions	Operator symbol	Name	Comments
$F_0=0$		Null	Binary constant 0
$F_1=xy$	x.y	AND	x and y
$F_2=xy'$	x/y	Inhibition	x but not y
$F_3=x$		Transfer	X
$F_4=x'y$	y/x	Inhibition	y but not x
$F_5=y$		Transfer	Y
$F_6=xy'+x'y$	$x \oplus y$	Exclusive-OR	x or y but not both
$F_7= x + y$	$x + y$	OR	x or y
$F_8= (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9= xy +x'y'$	x Oy	Equivalence	x equals y
$F_{10}=y'$	y'	Complement	Not y
$F_{11} = x + y'$	$x \subset y$	Implication	If y then x
$F_{12}= x'$	x'	Complement	Not x
$F_{13} = x' + y$	$x \supset y$	Implication	If x then y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1

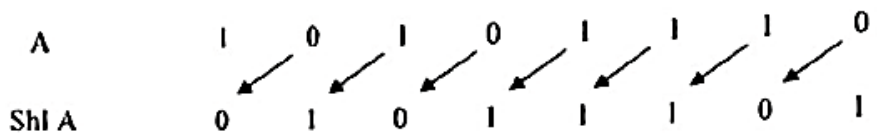
➢ Shift Micro-Operations

- Shift micro-operations shift the contents of a register either left or right.
- These micro- operations are generally used for serial transfer of data.
- They are also used along with arithmetic, logic, and other data-processing operations.
- No conventional symbol for shift operation. Here adopt symbols shl or shr
    - shl - shift left
    - shr - shift right

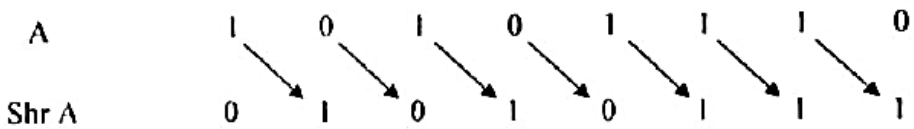
Example:

A ← shl A 1-bit shift to the left of register A

B ← shr B 1-bit shift to the right of register B



$A \leftarrow shl\ A, \quad A_1 \leftarrow A_n$



$A \leftarrow shr\ A, \quad A_n \leftarrow E$

PROCESSOR LOGIC DESIGN

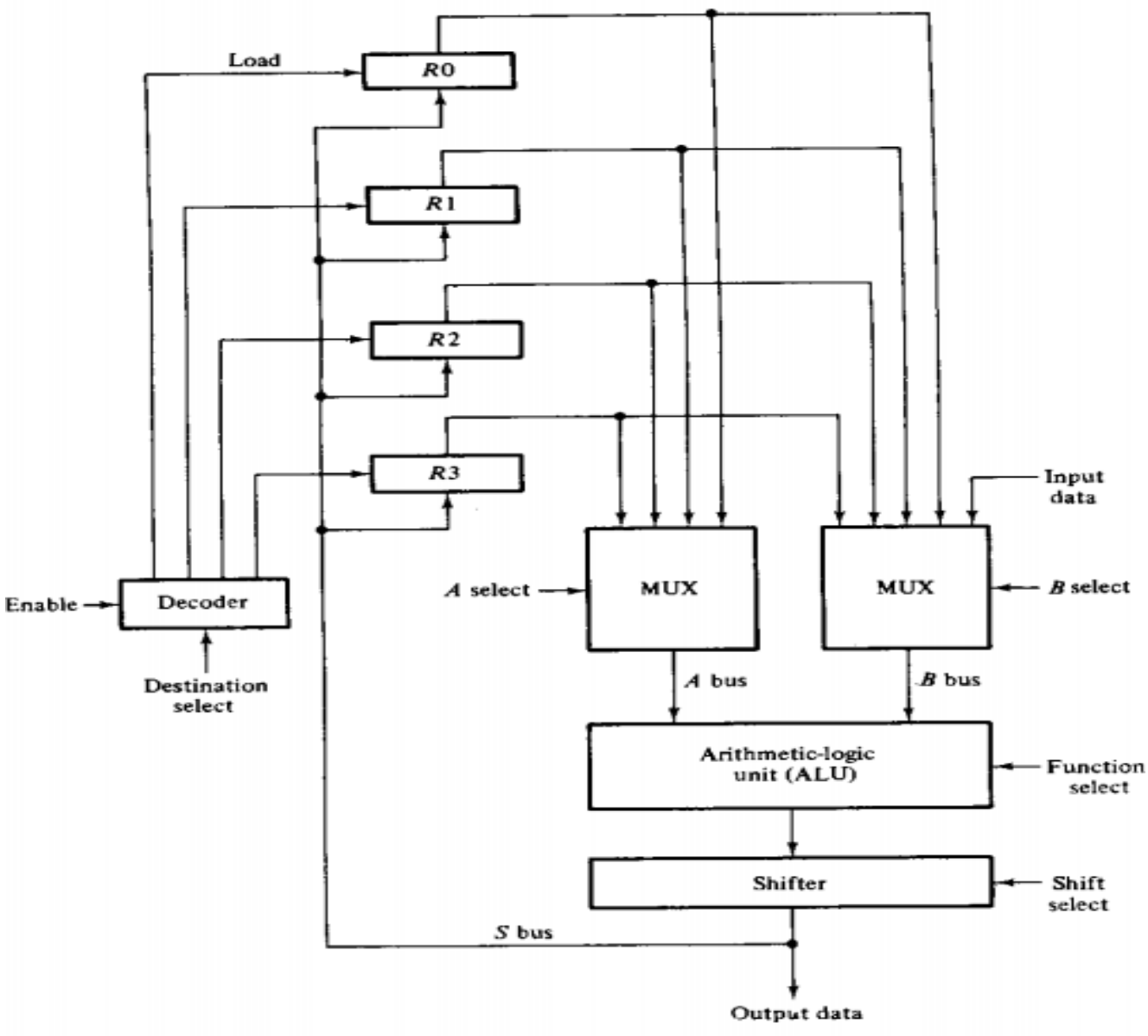
➢ Processor Organization

– The processor part of a computer CPU is sometimes 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.

– The various paths are said to be controlled by means of gates that open the required path and close all others.

– A processor unit can be designed to fulfill the requirements of a set of data paths for a specific application.

- In a processor unit, the data paths are formed by means of buses and other common lines.
- The control gates that formulate the given path are essentially multiplexers and decoders whose selection lines specify the required path.
- The processing of information is done by one common digital function whose data path can be specified with a set of common selection variables.
- A bus organization for four processor registers shown in below fig



- A bus organization for four processor registers is shown in Figure. Each register is connected to two multiplexers (MUX) to form input buses A and B.
- The selection lines of each multiplexer select one register for the particular bus.

- The A and B buses are applied to a common arithmetic logic unit.
- The function selected in the ALU determines the particular operation that is to be performed.
- The shift micro- operations are implemented in the shifter
- The result of the micro-operation goes through the output bus S into the inputs of all registers.
- The destination register that receives the information from the output bus is selected by a decoder.
- 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 output bus S provides the terminals for transferring data to an external destination.
- One input of multiplexer A or B can receive data from the outside.
- The control unit that supervises the processor bus system directs the information flow through the ALU by selecting the various components in the unit.

  For example, to perform the microoperation:

$$R1 \leftarrow R2 + R3$$

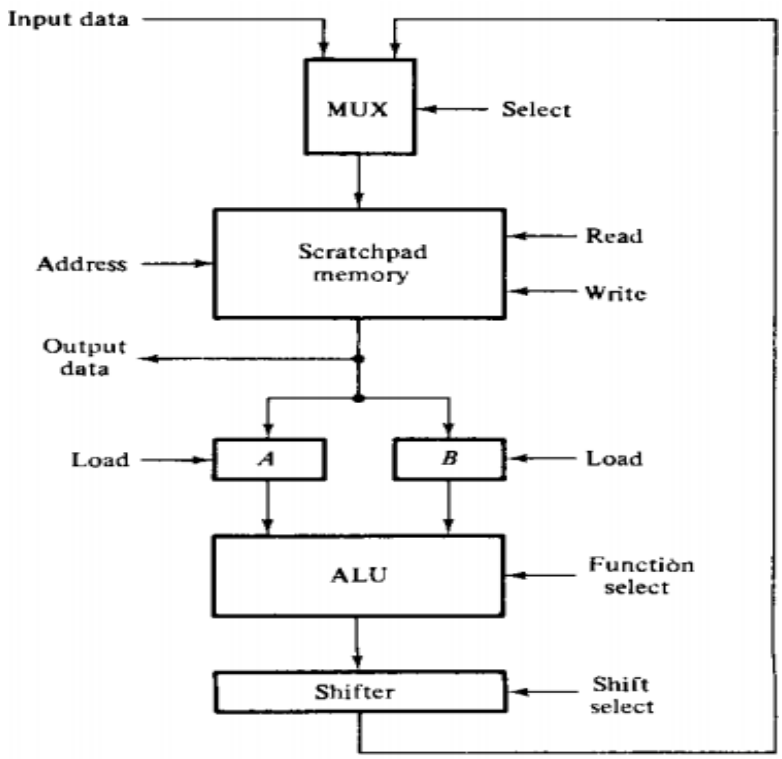
The control must provide binary selection variables to the following selector inputs:

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 the output of the ALU onto output bus S (no shift).
5. ***Decoder destination selector***: to transfer the contents of bus S into R 1.

➢ Scratchpad memory

- The register in a processor unit can be enclosed in a small memory unit.
- When included in a processor unit, a small memory is sometime called a scratchpad memory.
- The use of a small memory is a cheaper alternative to collecting processor registers through a bus system.
- The difference between the two system is the manner in which information is selected for transfer into the ALU.
- In a bus system, the information transfer is selected by the multiplexer that form the buses.
- Processor unit that uses scratchpad memory is shown in figure. Resource register is selected from memory and loaded into register A.

- A Second source register is selected from memory and loaded into register B.
- The information in A and B is manipulated in the ALU and shifter.



- Result of the operation is transferred to a memory register specifying its word address and activating the memory-write input control.
- Assume that the memory has eight words, so that an address must be specified with three bits.
- To perform the operation R1 ← R2 + R3
- The control must provide binary selection variable to perform the following sequence of micro-operations

T_1: A ← M[010]	read R2 to register A
T_2: B ← M[011]	read R3 to register B
T_3: M[001] ← A + B	perform operation in ALU and transfer result to R1

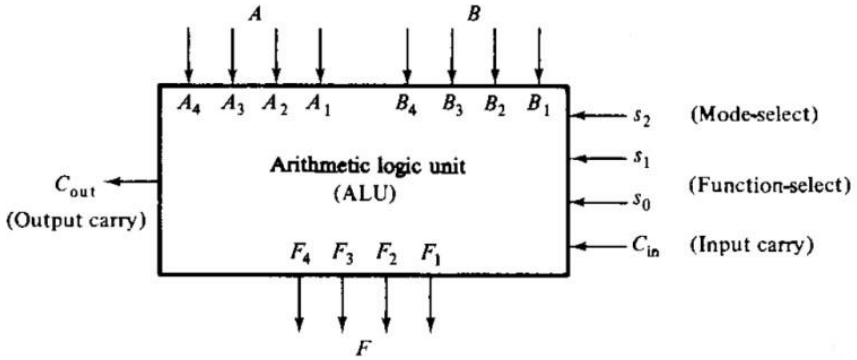
- Control function T_1 must supply an address of 010 to the memory and activate the read and load A inputs.
- Control function T_2 must supply an address 011 to the memory and activate the read and load B inputs.

- Control function T_3 must supply the function code to the ALU and shifter to perform an add operation, apply an address 001 to the memory, select the output of the shifter for the MUX and activate the memory write input.
- Some processor employ a 2 port memory in order to overcome the delay caused when reading two source registers.

➢ Arithmetic Logic Unit

- An arithmetic logic unit (ALU) is a multi-operation, 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.

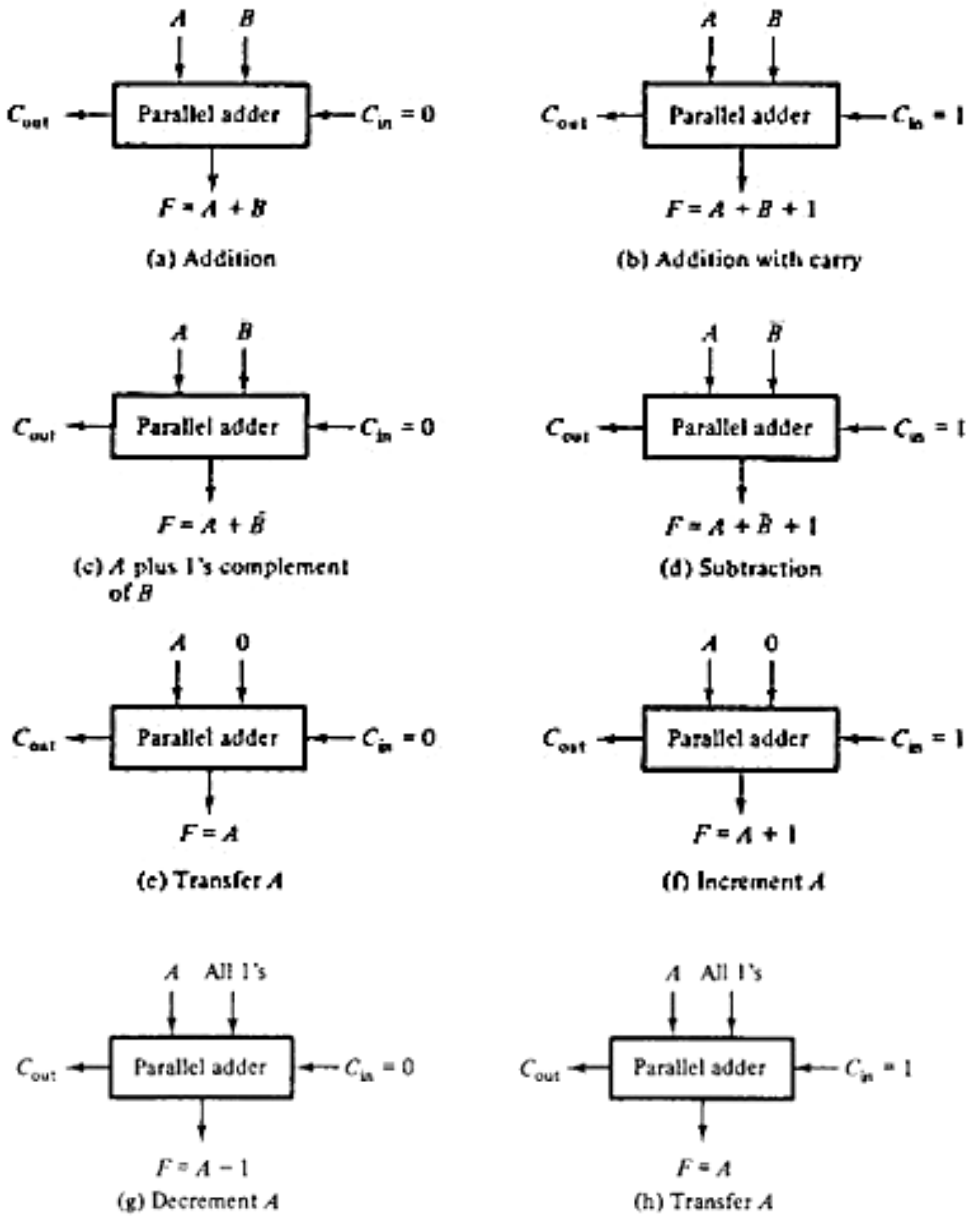
The figure shows the block diagram of 4 bit ALU.



- The 4 data input from A is combined with the input B to generate the output F.
- The Mode selection variable S2 is used to select either arithmetic operation or logic operation.
- The two function select input $S1$ and $S0$ specify the particular arithmetic or logic operation to be generated.
- The design of a typical ALU will be carried out in three stages.
- First, the design of the arithmetic section will be undertaken.
- Second, the design of the logic section will be considered.
- Finally, the arithmetic section will be modified so that it can perform both arithmetic and logic operations.

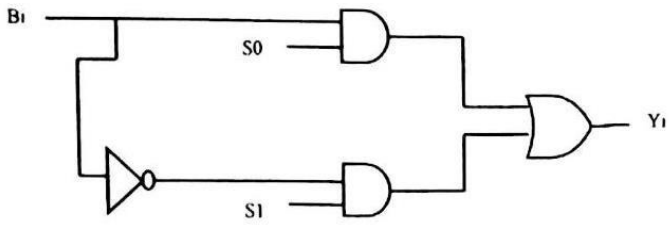
➢ Design of Arithmetic Circuit

- The basic component of the arithmetic section of an ALU is a parallel adder.
- A parallel adder is constructed with a number of full-adder circuits connected in cascade.
- By controlling the data inputs to the parallel adder, it is possible to obtain different types of arithmetic operations.



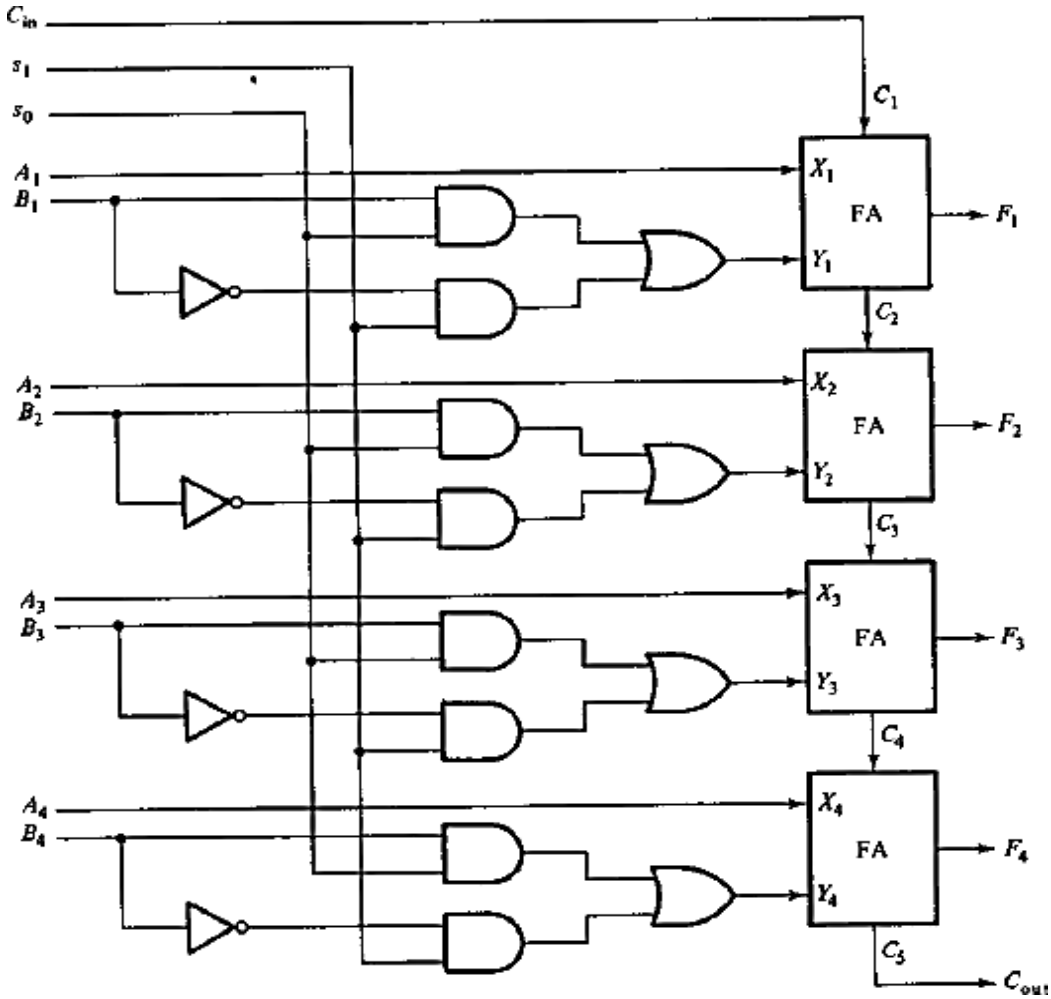
- The above figure demonstrates the arithmetic operations obtained when one set of inputs to a parallel adder is controlled externally.

- The design of 4 bit arithmetic circuit that performs 8 arithmetic operations is shown in the figure.
- The number of bits in the parallel adder may be of any value.
- The input carry C_{in} goes to the full-adder circuit in the least significant bit position.
- The output carry C_{out} comes from the full-adder circuit in the most significant bit position.
- From the above Figure, by changing the B input and C_{in} we get 8 operations. So the input B is applied in four different forms by using following circuit.



S1	S0	Yi
0	0	0
0	1	Bi
1	0	$\overline{Bi}$
1	1	1

- The input A is applied directly to the 4-bit parallel adder and the input B is modified.
- The resultant arithmetic circuit is shown in below figure.



– The function table for the arithmetic circuit is given below.

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

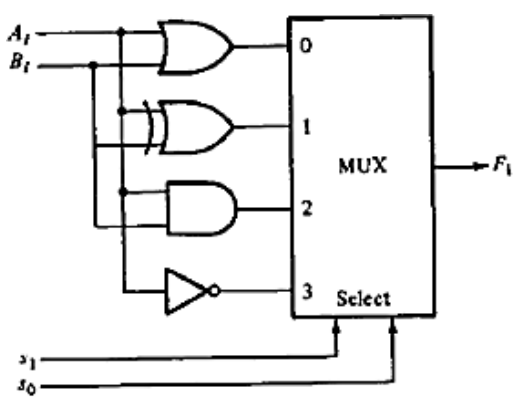
➢ **Design of Logic Circuit**

– The logic microoperations manipulate the bits of the operands separately and treat each bit as a binary variable.

– The 16 logic operations can be generated in one circuit and selected by means of four selection lines. Since all logic operations can be obtained by means of AND, OR, and NOT (complement) operations, it may be more convenient to employ a logic circuit with just these operations.

– For three operations, we need two selection variables. But two selection lines can select among four logic operations, so we choose also the exclusive-OR (XOR) function for the logic circuit to be designed in this and the next section.

– The simplest and most straight forward way to design a logic circuit is shown in figure given below.



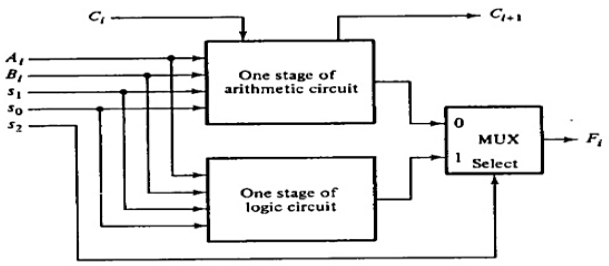
(a) Logic diagram

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

– The diagram shows one typical stage designated by subscript i.

- The circuit must be repeated n times for an n-bit logic circuit.
- The four gate generate four logic operations OR, XOR, AND, and NOT.
- The two selection variables in the multiplexer select one of the gates for the output.
- The function table lists the output logic generated as a function of the two selection variables.
- The logic circuit can be combined with the arithmetic circuit to produce one arithmetic logic unit.
- This is illustrated in the below figure



- We use a third selection variable, S_2, to differentiate between arithmetic logic circuit.
- When $S_2 = 0$, the arithmetic output is selected, when $S_2 = 1$, the logic output is selected.
- Although the two circuits can be combined in this manner, this is not the best way to design an ALU.
- The logic circuit can be combined with the arithmetic circuit to produce one arithmetic logic unit.

Design steps

1. Design the arithmetic section independent of the logic section.
2. Determine the logic operations obtained from the arithmetic circuit in step 1, assuming that the input carries to all stages are 0.
3. Modify the arithmetic circuit to obtain the required logic operations.

– The inputs to each full-adder circuit are specified by the Boolean functions:

$$X_i = A_i + S_2 S_1' S_0' B_i + S_2 S_1 S_0' B_i$$

$$Y_i = S_0 B_i + S_1 B_i'$$

$$Z_i = S_2' C_i$$

– When S2 = 0, the three functions reduce to:

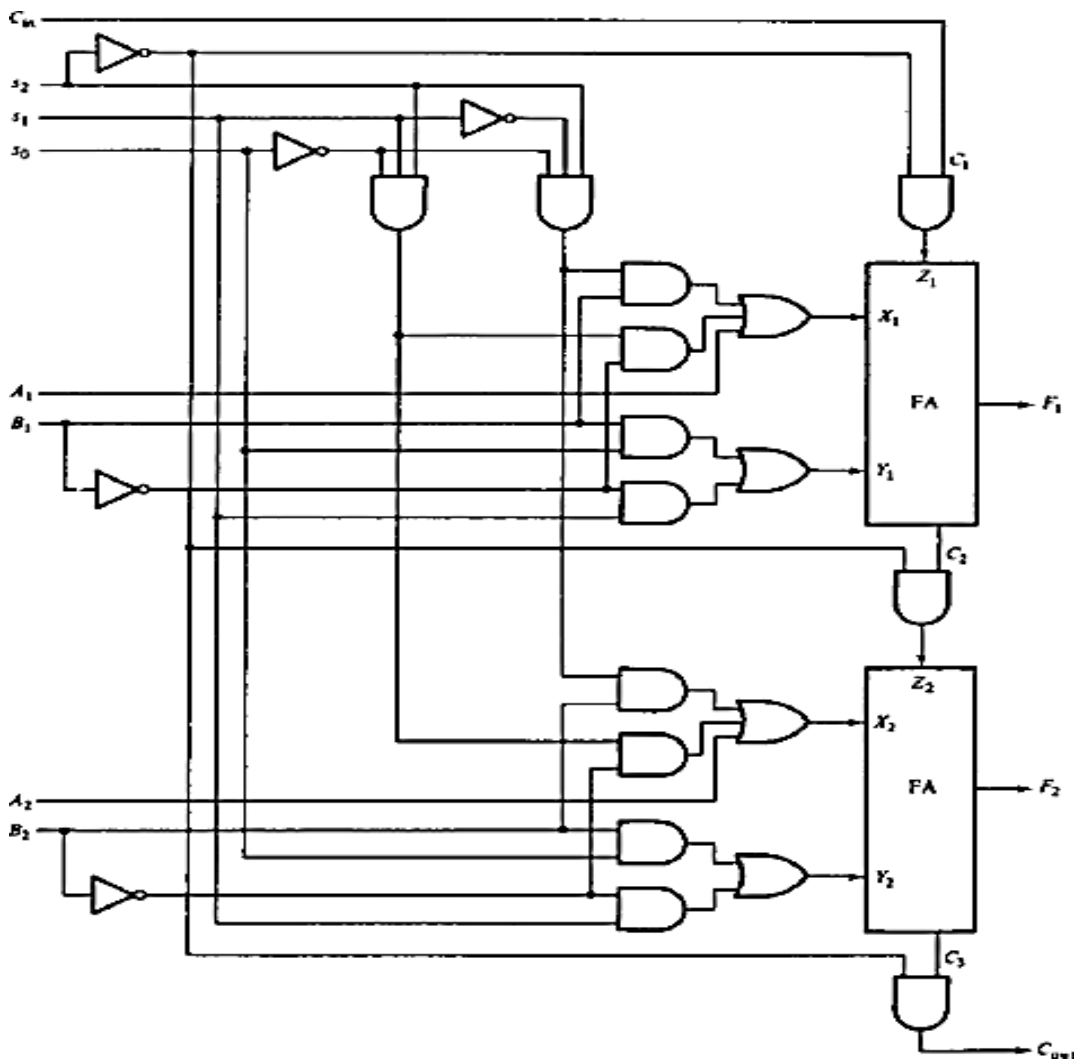
$$X_i = A_i$$

$$Y_i = S_0 B_i + S_1 B_i'$$

$$Z_i = C_i$$

– Which are the functions for the arithmetic circuit.

– The logical operations are generated when S2 = 1



– The function table for the Arithmetic and Logic Unit is shown below

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

- **Status Registers**

– The status register is the hardware register that contains information about the state of the processor.

– The relative magnitude of two numbers may be determined by subtracting one number from the other and then checking certain bit conditions in the resultant difference.

– This status bit conditions (often called condition-code bits or flag bits) are stored in a status register.

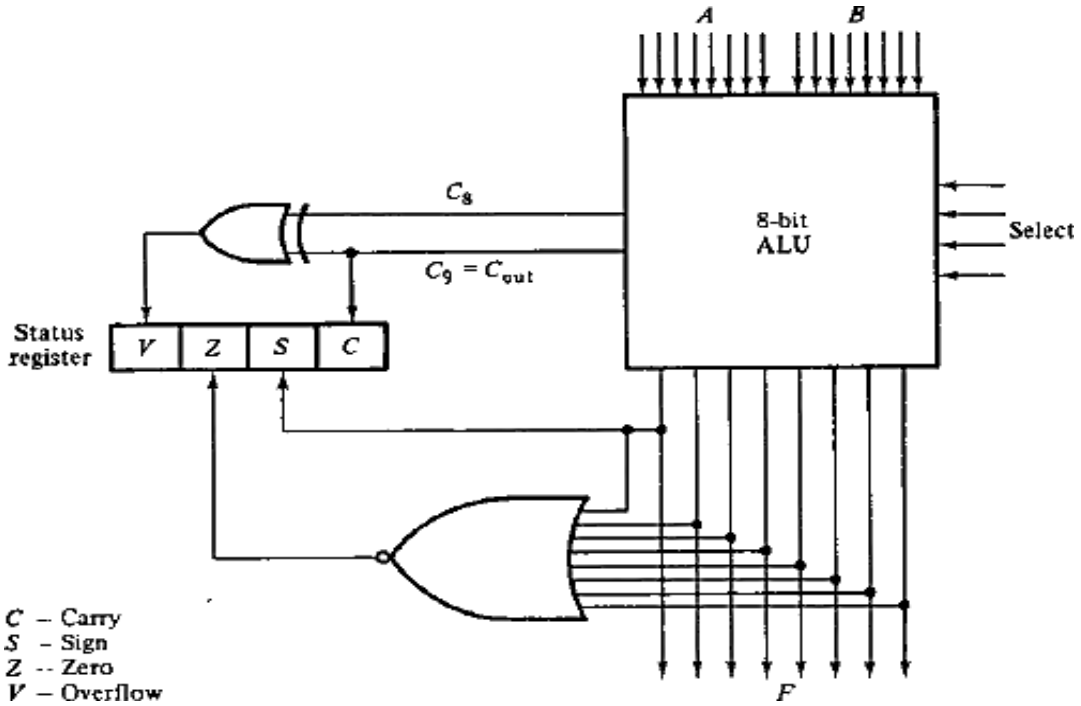
– Status register is a 4 bit register.

– The four bits are C (carry), Z (zero),S (sign) and V (overflow).

– These bits are set or cleared as a result of an operation performed in the ALU.

- Bit C is set if the output carry of an ALU is 1.
- Bit S is set to 1 if the highest order bit of the result in the output of the ALU is 1.
- Bit Z is set to 1 if the output of the ALU contains all O's.
- Bit V is set if the exclusive —OR of carries C8 and C9 is 1, and cleared otherwise. This is the condition for overflow when the numbers are in signed 2's complement representation. For an 8 bit ALU, V is set if the result is greater than 127 or less than -128.

– After an ALU operation, status bits can be checked to determine the relationship that exist between the values of A and B.



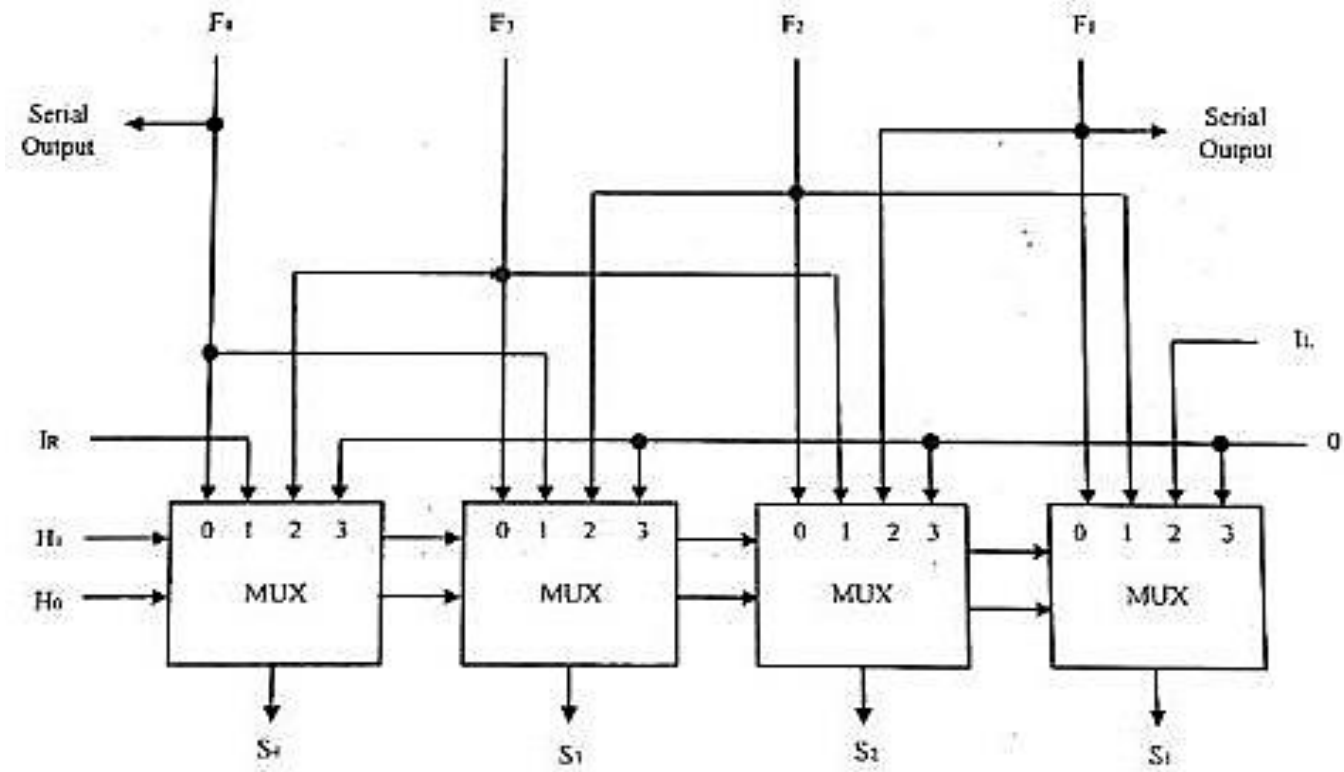
- If bit V is set after the addition two signed numbers, it indicates an overflow condition.
- C-carry: Bit C is set if the output carry of the ALU is 1. It is cleared if the output carry is 0.
- S-sign bit: Bit S is set if the highest order of the result in the output of the ALU is 1. It is cleared if the highest order bit is 0.
- Z-zero: Bit Z is set if the output of the ALU contains 0's and cleared otherwise. Z=1 if the result is zero and Z=0 if result is nonzero
- V-overflow: set if there is any overflow, for a 8 bit ALU, V is set if the result is greater than 127 and less than -128.
- Relative magnitudes of A and B can be checked by compare operation.
- If A-B is performed for two unsigned binary numbers, relative magnitudes of A and B can be determined from the values transferred to the C and Z bits. If Z=1,we knows that A=B, since A-B=0. If Z=0, then we know that A is not equal to B. Similarly C=1 if A>=B and C=0 if A<B. The following table lists the various conditions

Relation	Condition of status bits	Boolean function
$A > B$	$C = 1$ and $Z = 0$	CZ'
$A \geq B$	$C = 1$	C
$A < B$	$C = 0$	C'
$A \leq B$	$C = 0$ or $Z = 1$	$C' + Z$
$A = B$	$Z = 1$	Z
$A \neq B$	$Z = 0$	Z'

- **Design of Combinational Logic Shifter**

- The shift unit attached to the processor transfers the output of the ALU onto the output bus.
- Shifter may function in four different ways.
  1. The shifter may transfer the information directly without a shift.
  2. The shifter may shift the information to the right.
  3. The shifter may shift the information to the left.
  4. In some cases no transfer is made from ALU to the output bus.
- A shifter is a bi-directional shift-register with parallel load.

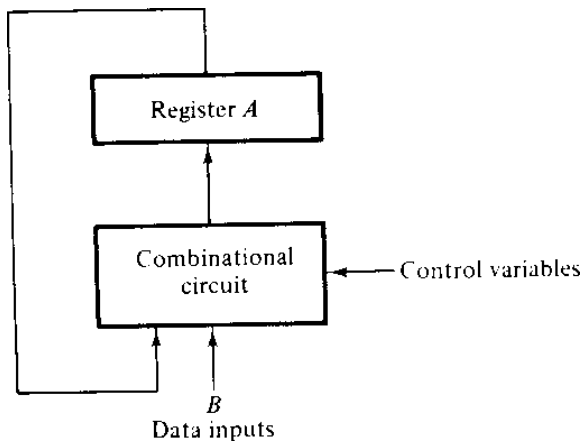
- The information from ALU can be transferred to the register in parallel and then shifted to the right or left.
- In this configuration, a clock pulse is needed for the transfer to the shift register, and another pulse is needed for the shift.
- Another clock pulse may also in need of when information is passed from shift register to destination register



- The number of clock pulses may reduce if the shifter is implemented with a combinational circuit.
- A combinational—logic shifter can be constructed with multiplexers. The above figure will show the same.
- Shifter operation can be selected by two variables H_1 H_0
  - If H_1 H_0 = 0 0 No shift is executed and the signal from F go directly to S lines
  - If H_1 H_0 = 0 1 Shift Right is executed
  - If H_1 H_0 = 1 0 Shift Left is executed
  - If H_1 H_0 = 1 1 No operations

Design of Accumulator

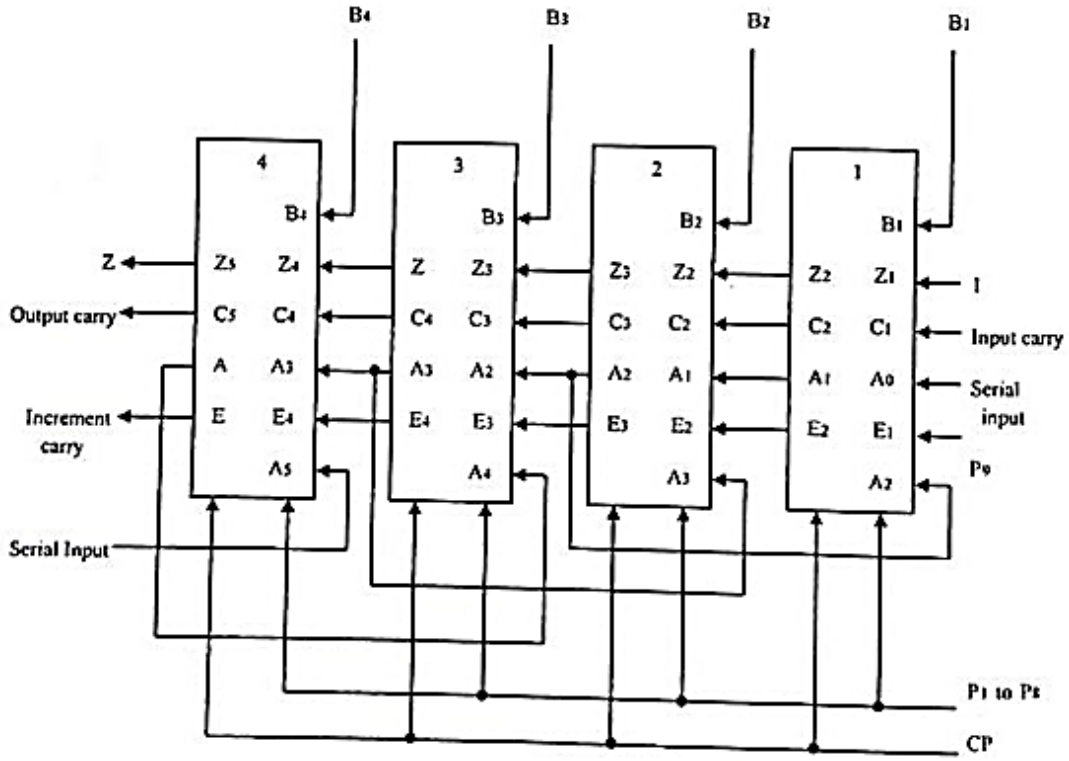
- Some processor units distinguish one register from all others and call it an accumulator register.
- An accumulator is a register for short-term, intermediate storage of arithmetic and logic data in a computer's CPU
- The block diagram of an accumulator that forms a sequential circuit is shown in figure below.



- The A register and the associated combinational circuit constitutes a sequential circuit.
- The combinational circuit replaces the ALU but cannot be separated from the register, since it is only the combinational-circuit part of a sequential circuit.
- The A register is referred to as the accumulator register and is sometimes denoted by the symbol AC.
- Here, accumulator refers to both the A register and its associated combinational circuit.
- The external inputs to the accumulator are the data inputs from B and the control variables that determine the micro operations for the register.
- The next state of register A is a function of its present state and of the external inputs.
- An Accumulator can performs Parallel load, Shift operations, Counting, and Data-processing operations

Binary code	Function of selection variables					
	A	B	D	F with $C_{in} = 0$	F with $C_{in} = 1$	H
0 0 0	Input data	Input data	None	$A, C \leftarrow 0$	$A + 1$	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	$R3$	$R3$	$R3$	$A - 1$	$A, C \leftarrow 1$	0's to output bus
1 0 0	$R4$	$R4$	$R4$	$A \vee B$	—	—
1 0 1	$R5$	$R5$	$R5$	$A \oplus B$	—	Circulate-right with C
1 1 0	$R6$	$R6$	$R6$	$A \wedge B$	—	Circulate-left with C
1 1 1	$R7$	$R7$	$R7$	$\bar{A}$	—	—

- The micro operations included in an accumulator depend on the operations that must be included in the particular processor.
- The set of micro operations for the accumulator is given in table
- In all listed microoperations A is the source register. B register is used as the second source register.
- The destination register is also accumulator register itself. For a complete accumulator there will be n stages.



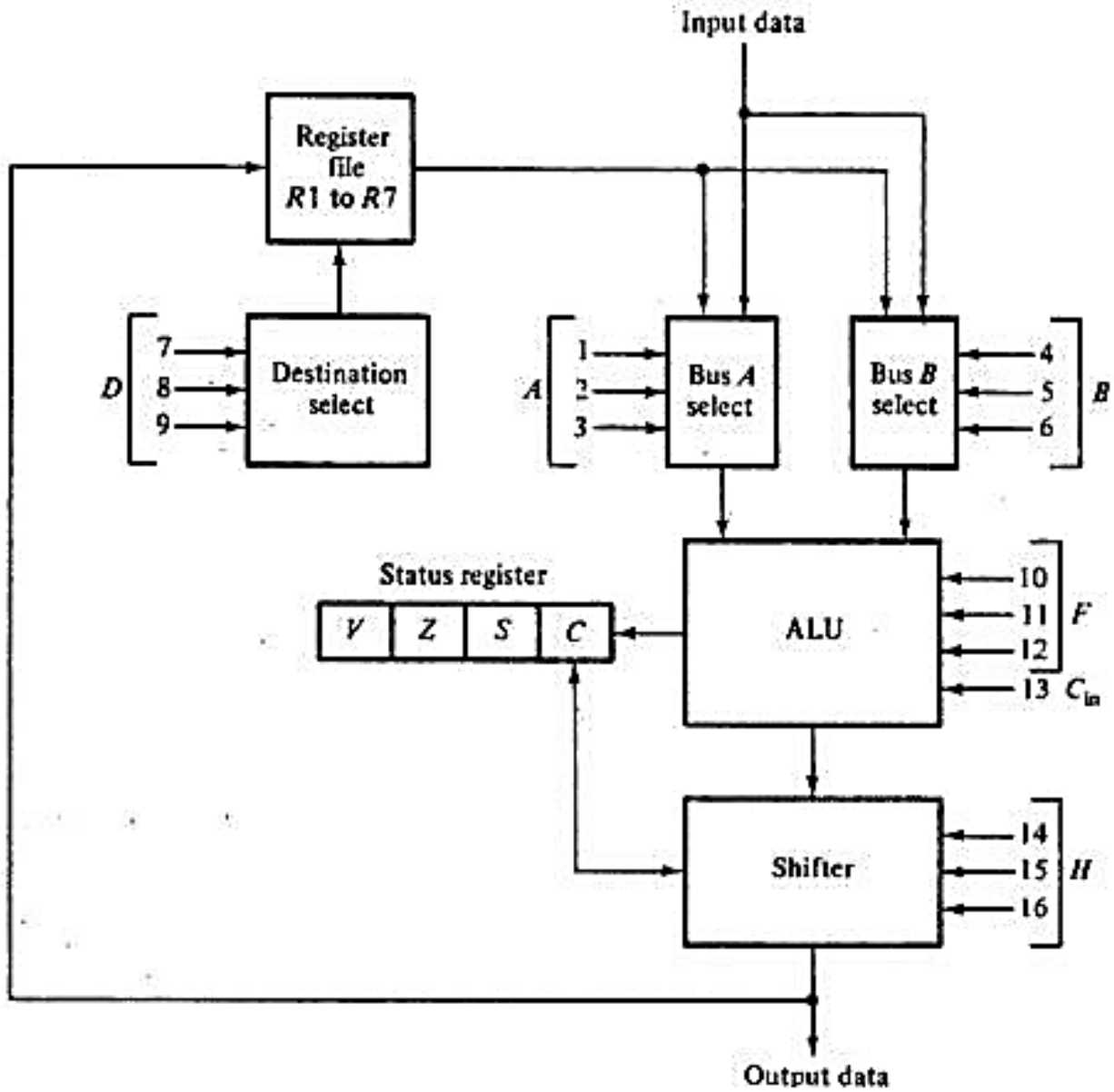
- The inputs and outputs of each stage can be connected in cascade to form a complete accumulator.
- Here we are discussing the design of a 4 bit accumulator.
- The number on top of each block represents the bit position.
- All blocks receive 8 control variables P1 to P8 and the clock pulses from CP.
- The other six inputs and four outputs are same as with the typical stage.
- The zero detect chain is obtained by connecting the z variables in cascade, with the first block receiving a binary constant I.
- The last stage produces the zero detect variable Z.
- Total number of terminals in the 4 bit accumulator is 25, including terminals for the A outputs.

- Incorporating two more terminals for power supply, the circuit can be enclosed within one IC package having 27 or 28 pins.
- The number of terminals for the control variable can be reduced from 9 to 4 if a decoder is inserted in the IC.
- In such cases, IC pin count is also reduced to 22 and the accumulator can be extended to 16 microoperations without adding external pins (That is, with 4 bits we can identify 16 operations).

➢ Processor Unit

- A block diagram of a processor unit is shown in figure.
- It consists of seven registers R1 through R7 and a status register.
- The outputs of the seven registers go through two multiplexers to select the inputs to the ALU.
- Input data from an external source are also selected by the same multiplexers. The output of the ALU goes through a shifter and then to a set of external output terminals.
- The functions of all selection variables are specified in table below.
- The 3-bit binary code listed in the table specifies the code for each of the five fields A, B, D, input data, F, and H.
- The register selected by A, B, and D is the one whose decimal number is equivalent to the binary number in the code.
- When the A or B field is 000, the corresponding multiplexer selects the input data. When D = 000, no destination register is selected.

Control variable	Microoperation	Name
p_1	$A \leftarrow A + B$	Add
p_2	$A \leftarrow 0$	Clear
p_3	$A \leftarrow \overline{A}$	Complement
p_4	$A \leftarrow A \wedge B$	AND
p_5	$A \leftarrow A \vee B$	OR
p_6	$A \leftarrow A \oplus B$	Exclusive-OR
p_7	$A \leftarrow \text{shr } A$	Shift-right
p_8	$A \leftarrow \text{shl } A$	Shift-left
p_9	$A \leftarrow A + 1$	Increment
	If $(A = 0)$ then $(Z = 1)$	Check for zero



(a) Block Diagram

1 2 3 4	5 6 7	8 9	10 11 12	13	14 15 16
A	B	D	F	C_{in}	H

(b) Control word

- The three bits in the F field, together with the input carry C_{in}, provide the 12 operations of the ALU as specified in above table.
- Note that there are two possibilities for F = A. In one case the carry bit C is cleared, and in the other case it is set to 1.

MODULE – III

Arithmetic algorithms: *Algorithms for multiplication and division (restoring method) of binary numbers. Array multiplier, Booth's multiplication algorithm.*

Pipelining: *Basic principles, classification of pipeline processors, instruction and arithmetic pipelines (Design examples not required), hazard detection and resolution.*

ARITHMETIC ALGORITHMS

1. Multiplication of binary numbers

- 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, as in the third row of the example.
- The product is computed one bit at a time by adding the bit columns from right to left and propagating carry values between columns.

```
                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
```

Array Multiplier

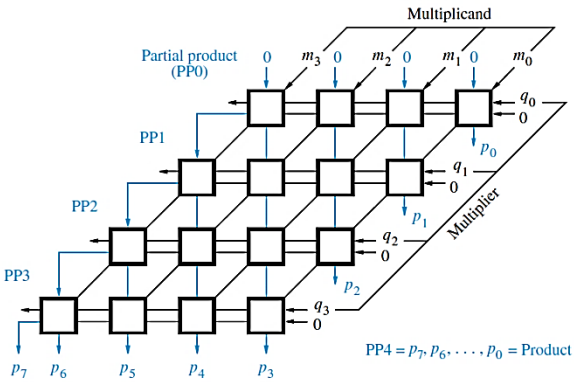
- Binary multiplication of unsigned operands can be implemented in a combinational, two dimensional logic array, as shown in Figure for the 4-bit operand case.
- The main component in each cell is a full adder, FA. The AND gate in each cell determines whether a multiplicand bit, m_j, is added to the incoming partial-product bit, based on the value of the multiplier bit, q_i.

– Each row i, where 0 ≤ i ≤ 3, adds the multiplicand (appropriately shifted)to the incoming partial product, PPi, to generate the outgoing partial product, PP(i + 1), ifqi = 1. If qi = 0, PPi is passed vertically downward unchanged.

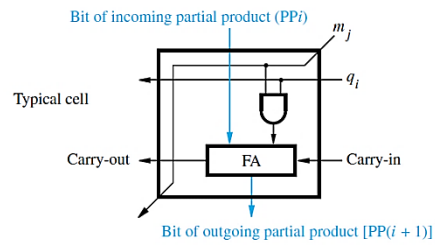
– PP0 is all 0s, and PP4 is the desired product.

– The multiplicand is shifted left one position per row by the diagonal signal path.

– We note that the row-by-row addition done in the array circuit differs from the usual hand addition described previously, which is done column-by-column.

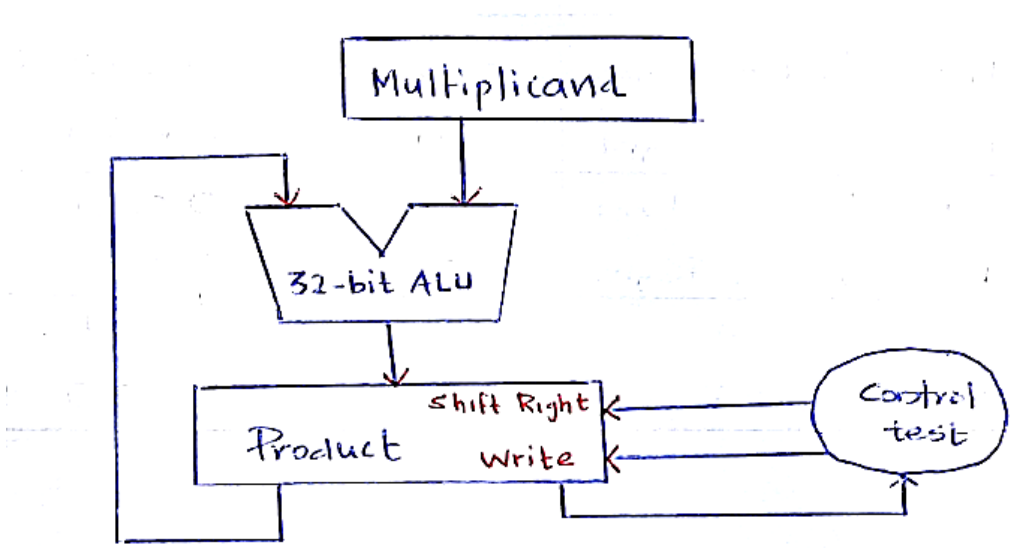


(a)Array multiplication of positive binary operands

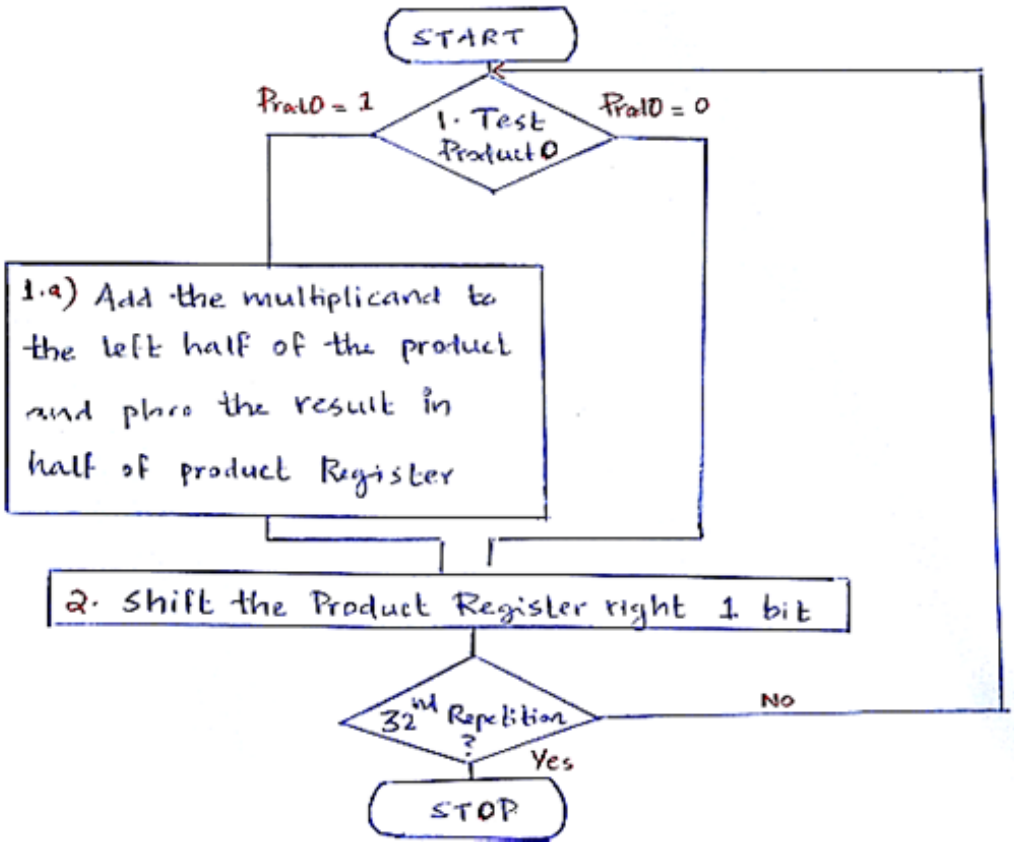


(b) Multiplier cell

Hardware:



Flowchart



Example:

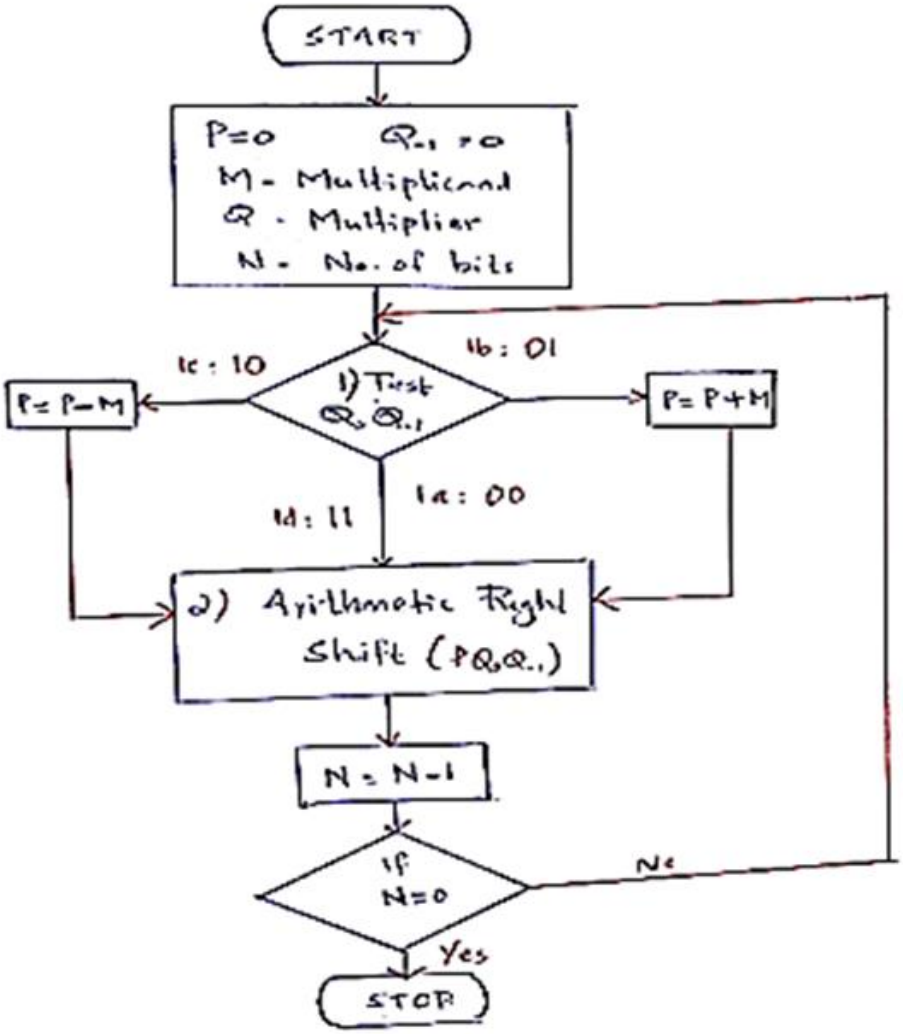
Example : Multiply 2×6 using final version of multiplication

Iteration	Steps	Multiplicand	Product
0	Initial values	0010	0000 0110
1	1a : no operation 2 : shift Product Right	0010 0010	0000 0110 0000 0011
2	1a : Prod = Prod + Mcand 2 : shift Product Right	0010 0010	0010 0011 0001 0001
3	1a : Prod = Prod + Mcand 2 : Shift Product Right	0010 0010	0011 0001 0001 1000
4	1a : no operation 2 : Shift Product Right	0010 0010	0001 1000 0000 1100

Booth's Multiplication Algorithm

- Booth's algorithm is a powerful direct algorithm to perform signed number multiplication.
- The algorithm is based on the fact that any binary number can be represented by the sum and difference of other binary numbers.
- It operates on the fact that strings of 0's in the multiplier require no addition but just shifting, and a string of 1's in the multiplier from bit weight 2^k to weight 2^m can be treated as $2^{k+1} - 2^m$
- It handles both + ve & - ve numbers uniformly.
- It achieves some efficiency in the number of additions required, when the multiplier has a few large blocks of 1's.
- Booth algorithm Speed up the multiplication process.

Flowchart:



Example:

Multiply 2 × -3 using Booth's Multiplication
Multiplicand (M): 2 -> 0010
Multiplier (Q): -3 -> 1101

3 : 0011
(-3) 2's complement of 3 : 1100
1
1101

Iteration	Steps	Multiplicand	Product	Q_0	Q_{-1}
0	Initial values	0010	0000	1101	0
1	1c: 10 ⇒ P = P−M	0010	1110	1101	0
	2: Shift Product Right	0010	1111	0110	1
2	1b: 01 ⇒ P = P+M	0010	0001	0110	1
	2: Shift Product Right	0010	0000	1011	0
3	1c: 10 ⇒ P = P−M	0010	1110	1011	0
	2: Shift Product Right	0010	1111	0101	1
4	1d: 11 ⇒ No operation	0010	1111	0101	1
	2: Shift Product Right	0010	1111	1010	1

Example: Multiply "15 x -13" using Booth's Algorithm

Ans: 15 → 01111
13 → 01101
-13 → 10011

15x-13 = -195

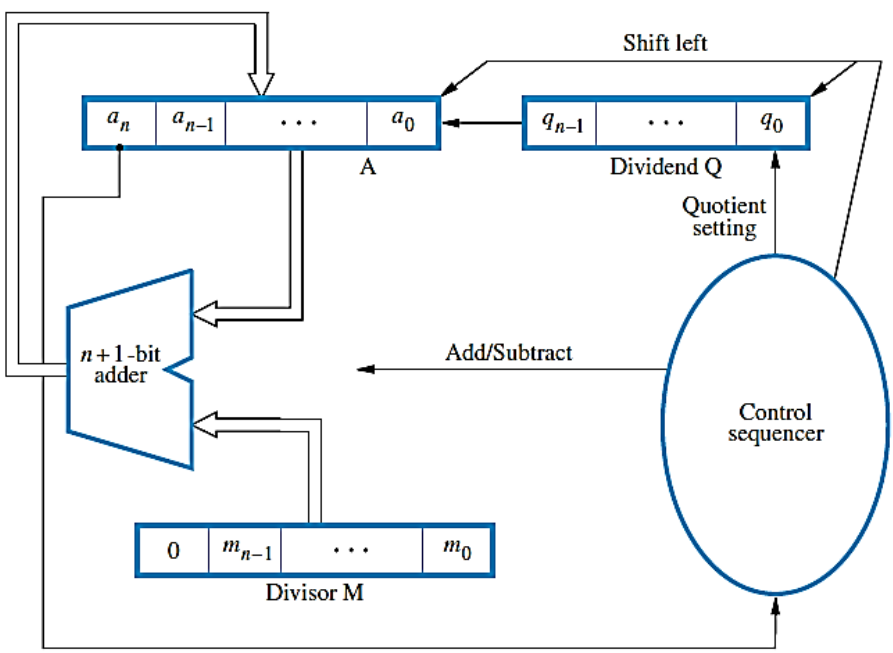
195 → 00110 00011
-195 → 11001 11101

	Steps	Multiplicand	Product	Q_0	Q_{-1}
0	Initial Value	01111	00000	10011	0
1	1c: 10 ⇒ P = P−M	01111	10001	10011	0
	2: Shift Product Right	01111	11000	11001	1
2	1d: 11 ⇒ No operation	01111	11000	11001	1
	2: Shift Product Right	01111	11100	01100	1
3	1b: 01 ⇒ P = P+M	01111	01011	01100	1
	2: Shift Product Right	01111	00101	10110	0
4	1a: 00 ⇒ No operation	01111	00101	10110	0
	2: Shift Product Right	01111	00010	11011	0
5	1c: 10 ⇒ P = P−M	01111	10011	11011	0
	2: Shift Product Right	01111	11001	11101	1

-195

Integer Division

- Figure shows a logic circuit arrangement that implements the restoring division algorithm just discussed.
- An n-bit positive divisor is loaded into register M and 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.



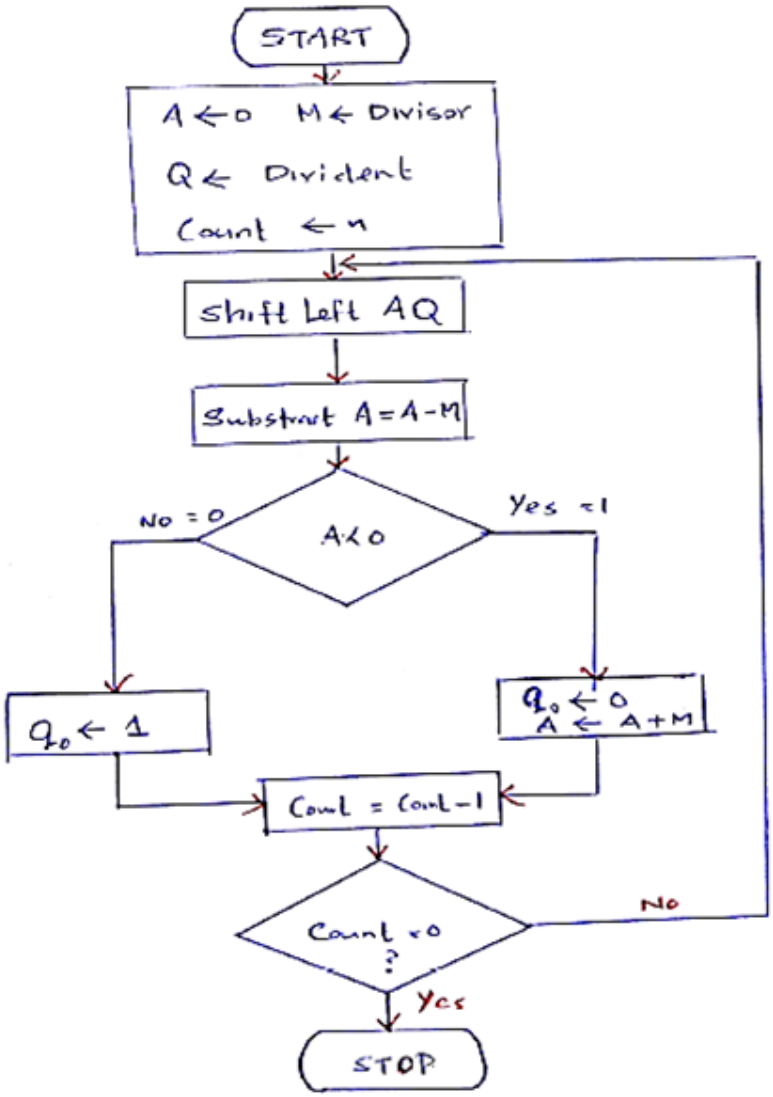
Circuit arrangement for binary division.

- 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 Q_0 to 0 and add M back to A (that is, restore A); otherwise, set Q_0 to 1

Flowchart:



Example: Divide "8 by 3" using Restoring method.

8 → 1000 (Divident)

3 → 0011 (Divisor

```
        0010 (Quotient)
0011 | 1000
       (Divident
       ----
       0010
       (Remainder)
```

```
      2
3 | 8
    6
    ---
    2
```

Iteration	Steps	Divisor M	Register A	Dividend Q
0	Initial Values	00011	00000	1000
1	1: Shift Left AQ	00011	00001	000 □
	2: Substract A = A−M	00011	11110	000 □
	3: Set q_0, Restore A = A+M	00011	00001	0000
2	1: Shift Left AQ	00011	00010	000 □
	2: Substract A = A−M	00011	11111	000 □
	3: Set $q_0 \to 0$, Restore A = A+M	00011	00010	0000
3	1: Shift Left AQ	00011	00100	000 □
	2: Substract A = A−M	00011	00001	000 □
	3: Set $q_0 \leftarrow 1$	00011	00001	0001
4	1: Shift Left AQ	00011	00010	001 □
	2: Substract A = A−M	00011	11111	001 □
	3: Set $q_0 \to 0$, Restore A = A+M	00011	00010	0010
			Remainder	Quotient

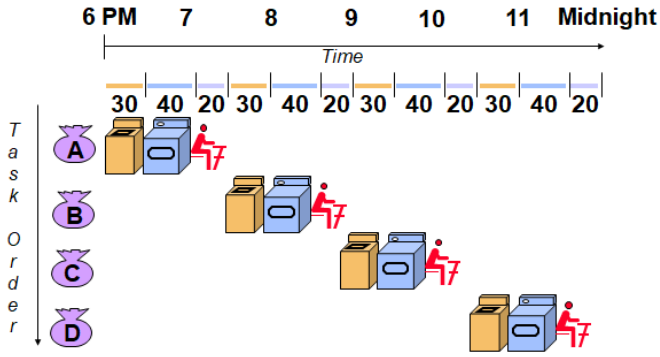
❖ PIPELINING

- Pipelining is a technique of decomposing a sequential process into sub operations, with each **sub process** being executed in a special dedicated **segment** that **operates concurrently** with all other segments.
- Laundry Example :Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold

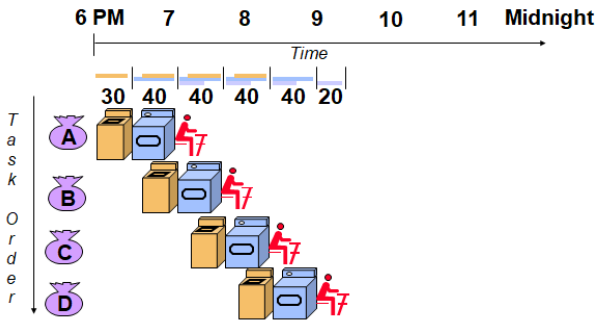
Washer takes 30 minutes 

Dryer takes 40 minutes 

Folder takes 20 minutes 



Sequential laundry takes ***6 hours for 4 loads***



Pipelined laundry takes ***3.5 hours for 4 loads***

- Pipelining doesn't help **latency** of single task, it helps **throughput** of entire workload
- Pipeline rate limited by **slowest** pipeline stage
- **Multiple** tasks operating simultaneously
- Potential speedup = **Number pipe stages**
- A pipeline can be visualized as a collection of processing segments through which binary information flows. Each segment performs partial processing dictated by the way the task is partitioned.
- The result obtained from the computation in each segment is transferred to the next segment in the pipeline. The final result is obtained after the data have passed through all segments.

Advantages of Pipelining

1. The cycle time of the processor is reduced.
2. It increases the throughput of the system
3. It makes the system reliable.

Disadvantages of Pipelining

1. The design of pipelined processor is complex and costly to manufacture.
2. The instruction latency is more.

Pipeline Organization

- The simplest way of viewing the pipeline structure is to imagine that each segment consists of an input register followed by a combinational circuit.
- The register holds the data and the combinational circuit performs the sub operation in the particular segment.

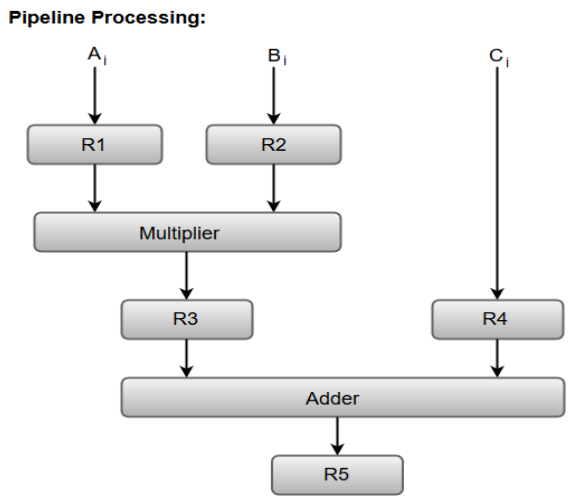
- The output of the combinational circuit is applied to the input register of the next segment.
- A clock is applied to all registers after enough time has elapsed to perform all segment activity.
- In this way the information flows through the pipeline one step at a time.

Example demonstrating the pipeline organization

- Suppose we want to perform the combined multiply and add operations with a stream of numbers.

  Ai*Bi + Ci for i=1, 2, 3 7

- Each sub operation is to implemented in a segment within a pipeline. Each segment has one or two registers and a combinational circuit as shown in fig.



- The sub operationsperformed in each segment of the pipeline are as follows:

R1<- Ai	R2<-Bi	Input Ai and Bi
R3<-R1*R2	R4<-Ci	multiply and input Ci
R5<-R3+R4		add Ci to product

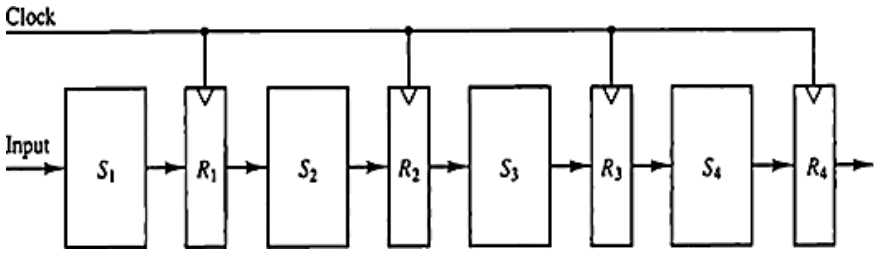
- The five registers are loaded with new data every clock pulse.

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A_1	B_1	—	—	—
2	A_2	B_2	$A_1 * B_1$	C_1	—
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
6	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
7	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
8	—	—	$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
9	—	—	—	—	$A_7 * B_7 + C_7$

- The first clock pulse transfers A1 and B1 into R1 and R2.
- The second clock pulse transfers the product of R1 and R2 into R3 and C1 into R4.
- The same clock pulse transfers A2 and B2 into R1 and R2.
- The third clock pulse operates on all three segments simultaneously.
- It places A3 and B3 into R1 and R2, transfers the product of R1 and R2 into R3, transfers C2 into R4, and places the sum of R3 and R4 into R5.
- It takes three clock pulses to fill up the pipe and retrieve the first output from R5.
- From there on, each clock produces a new output and moves the data one step down the pipeline.
- This happens as long as new input data flow into the system.

Four Segment Pipeline

- The general structure of four segment pipeline is shown in fig.



- The operands are passed through all four segments in affixed sequence.
- Each segment consists of a combinational circuit Si that performs a sub operation over the data stream flowing through the pipe.
- The segments are separated by registers Ri that hold the intermediate results between the stages.
- Information flows between adjacent stages under the control of a common clock applied to all the registers simultaneously.

Space time diagram:

- The behavior of a pipeline can be illustrated with a space time diagram.
- This isa diagram that shows the segment utilization as a function of time.

Segment:	1	2	3	4	5	6	7	8	9
1	T_1	T_2	T_3	T_4	T_5	T_6			
2		T_1	T_2	T_3	T_4	T_5	T_6		
3			T_1	T_2	T_3	T_4	T_5	T_6	
4				T_1	T_2	T_3	T_4	T_5	T_6

→ Clock cycles

- Fig The horizontal axis displays the time in clock cycles and the vertical axis gives the segment number.
- The diagram shows six tasks T1 through T6 executed in four segments.
- Initially, task T1 is handled by segment 1.
- After the first clock, segment 2 is busy with T1, while segment 1 is busy with task T2. Continuing in this manner, the first task T1 is completed after fourth clock cycle.
- From then on, the pipe completes a task every clock cycle.
- Consider the case where a k-segment pipeline with a clock cycle time tp is used to execute n tasks.
- The first task T1 requires a time equal to ktp to complete its operation since there are k segments in a pipe.
- The remaining n-1 tasks emerge from the pipe at the rate of one task per clock cycle and they will be completed after a time equal to (n-1) tp.
- Therefore, to complete n tasks using a k segment pipeline requires

  k+ (n-1) clock cycles.
- Consider a non-pipeline unit that performs the same operation and takes a time equal to tn to complete each task.
- The total time required for n tasks is n tn.
- The speedup of a pipeline processing over an equivalent non pipeline processing is defined by the ratio

  S=ntn / (k+n-1)tp
- As the number of tasks increases, n becomes much larger than k-1, and k+n-1 approaches the value of n, under this condition the speed up ratio becomes

  S=tn/tp
- If we assume that the time it takes to process a task is the same in the pipeline and non-pipeline circuits, we will have

$$tn=ktp.$$

- Including this assumption speed up ratio reduces to

$$S=ktp/tp=k$$

Classification of Pipeline Processors

In general, the pipeline organization is applicable for two areas of computer design which includes:

1. Arithmetic Pipeline
2. Instruction Pipeline

1. Arithmetic Pipelines:

- An arithmetic pipeline divides an arithmetic operation into sub operations for execution in the pipeline segments.
- So in arithmetic pipeline, an arithmetic operation like multiplication, addition, etc. can be divided into series of steps that can be executed one by one in stages in Arithmetic Logic Unit (ALU).
- Pipeline arithmetic units are usually found in very high speed computers.
- They are used to implement floating point operations, multiplication of fixed point numbers, and similar computations encountered in scientific problems.
- To understand the concepts of arithmetic pipeline in a more convenient way, let us consider an example of a pipeline unit for floating-point addition and subtraction.

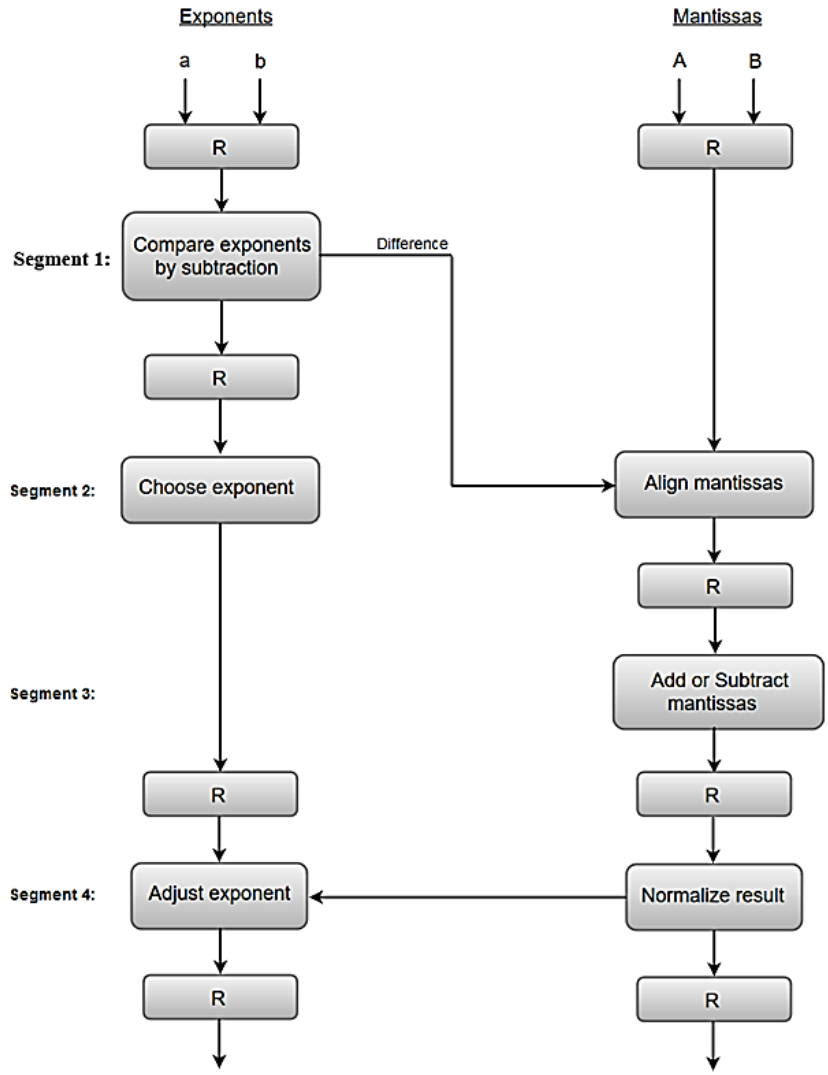
$$X = A * 10^a = 0.9504 * 10^3$$

$$Y = B * 10^b = 0.8200 * 10^2$$

$$Z = X + Y = 0.1324 * 10^4$$

- Here A and B are mantissas (significant digit of floating point numbers), while a and b are exponents.
- The floating point addition and subtraction is done in 4 parts:
  1. Compare the exponents.
  2. Align the mantissas.
  3. Add or subtract mantissas
  4. Produce the result.
- Registers are used for storing the intermediate results between the above operations.
- The following block diagram represents the sub operations performed in each segment of the pipeline.

Pipeline organization for floating point addition and subtraction:



1. Compare exponents by subtraction:
   - The exponents are compared by subtracting them to determine their difference. The larger exponent is chosen as the exponent of the result.
   - The difference of the exponents, i.e., 3 - 2 = 1 determines how many times the mantissa associated with the smaller exponent must be shifted to the right.
2. Align the mantissas:
   - The mantissa associated with the smaller exponent is shifted according to the difference of exponents determined in segment one.

$$X = 0.9504 * 10^3 \qquad Y = 0.08200 * 10^3$$

3. Add mantissas:
   - The two mantissas are added in segment three.

$$Z = X + Y = 1.0324 * 10^3$$

4. Normalize the result:

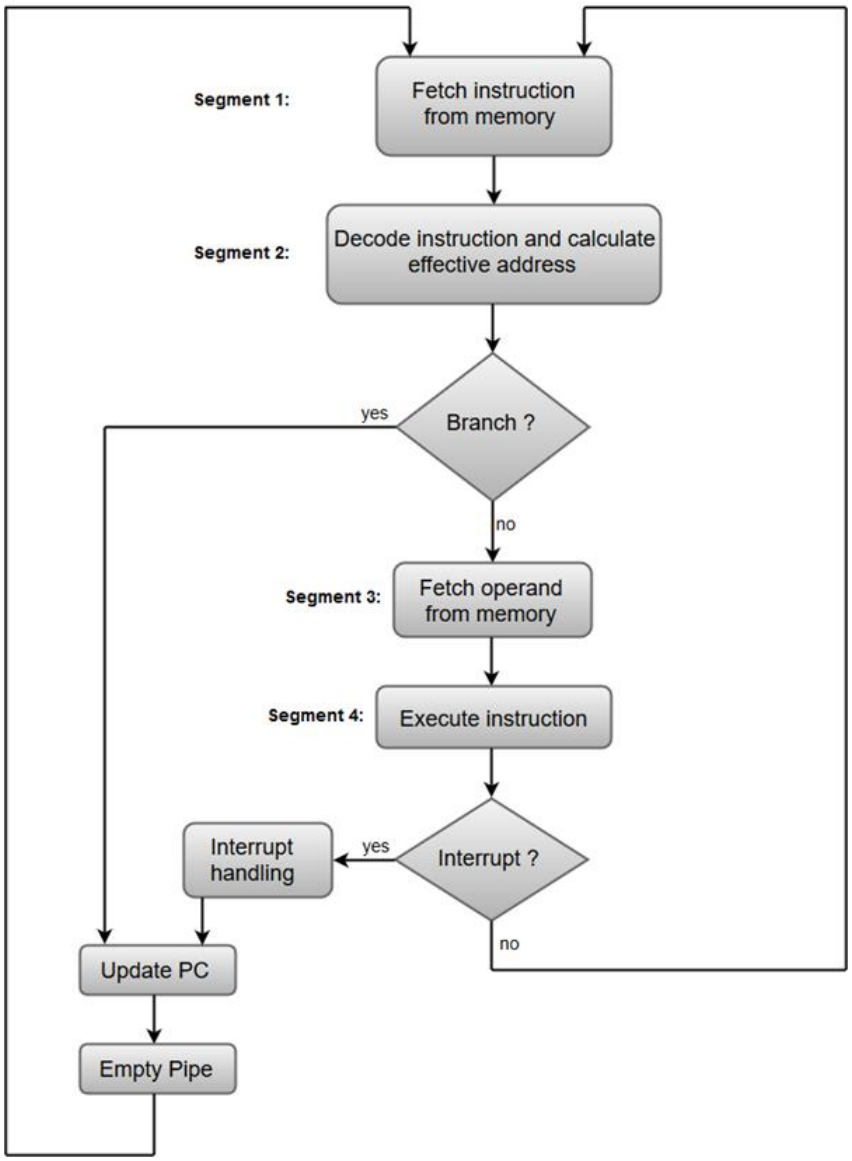
After normalization, the result is written as:

$$Z = 0.1324 * 10^4$$

2. Instruction Pipeline

- Pipeline processing can occur not only in the data stream but in the instruction stream as well.
- Most of the digital computers with complex instructions require instruction pipeline to carry out operations like fetch, decode and execute instructions.
- In general, the computer needs to process each instruction with the following sequence of steps.
    1. Fetch instruction from memory.
    2. Decode the instruction.
    3. Calculate the effective address.
    4. Fetch the operands from memory.
    5. Execute the instruction.
    6. Store the result in the proper place.

- Each step is executed in a particular segment, and there are times when different segments may take different times to operate on the incoming information.
- Moreover, there are times when two or more segments may require memory access at the same time, causing one segment to wait until another is finished with the memory.
- The organization of an instruction pipeline will be more efficient if the instruction cycle is divided into segments of equal duration.
- One of the most common examples of this type of organization is a **Four-segment instruction pipeline.**
- A four-segment instruction pipeline combines two or more different segments and makes it as a single one.
- For instance, the decoding of the instruction can be combined with the calculation of the effective address into one segment.
- The following block diagram shows a typical example of a four-segment instruction pipeline. The instruction cycle is completed in four segments.



Segment 1: FI

The instruction fetch segment can be implemented using first in, first out (FIFO) buffer.

Segment 2:DA

The instruction fetched from memory is decoded in the second segment, and eventually, the effective address is calculated in a separate arithmetic circuit.

Segment 3: FO

An operand from memory is fetched in the third segment.

Segment 4:EX

The instructions are finally executed in the last segment of the pipeline organization.

– Following shows the operation of the instruction pipeline. The time in the horizontal axis is divided into steps of equal duration. The four segments are represented in the diagram with an abbreviated symbol.

Step:		1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction:	1	FI	DA	FO	EX									
	2		FI	DA	FO	EX								
(Branch)	3			FI	DA	FO	EX							
	4				FI	-	-	FI	DA	FO	EX			
	5					-	-	-	FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

– It is assumed that the processor has separate instruction and data memories so that the operation in FI and FO can proceed at the same time.

– In the absence of a branch instruction, each segment operates on different instructions.

– Thus, in step 4, instruction 1 is being executed in segment EX; the operand for instruction 2 is being fetched into segment FO; instruction 3 is being decoded in segment DA; and instruction 4 is being fetched from memory in segment FI.

● Pipeline Conflicts

There are some factors that cause the pipeline to deviate its normal performance. Some of these factors are given below:

1. Timing Variations

All stages cannot take same amount of time. This problem generally occurs in instruction processing where different instructions have different operand requirements and thus different processing time.

2. Data Hazards

When several instructions are in partial execution, and if they reference same data then the problem arises. We must ensure that next instruction does not attempt to access data before the current instruction, because this will lead to incorrect results.

3. Branching

In order to fetch and execute the next instruction, we must know what that instruction is. If the present instruction is a conditional branch, and its result will lead us to the next instruction, then the next instruction may not be known until the current one is processed.

4. Interrupts

Interrupts set unwanted instruction into the instruction stream. Interrupts effect the execution of instruction.

5. Data Dependency

It arises when an instruction depends upon the result of a previous instruction but this result is not yet available.

Pipeline Hazards Detection and Resolution

- The problems that occur in the pipeline are called hazards.
- Hazards that arise in the pipeline prevent the next instruction from executing during its designated clock cycle.
- There are three types of hazards:
  1. **Data hazards:** Instruction depends on result of prior instruction still in the pipeline
  2. **Structural hazards:** Hardware cannot support certain combinations of instructions (two instructions in the pipeline require the same resource).
  3. **Control hazards:** Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jumps).

1. Data hazards

- Data hazards occur when instructions that exhibit **data dependence** modify data in different stages of a pipeline.
- Ignoring potential data hazards can result in race conditions (also termed race hazards).
- There are three situations in which a data hazard can occur:

  a) Read After Write (RAW), a true dependency
  b) Write After Read (WAR), an anti-dependency
  c) Write After Write (WAW), an output dependency

Note : Read after read (RAR) is not a hazard case.

- Consider two instructions i1 and i2, with i1 occurring before i2 in program order.

Read after write (RAW): *(i2 tries to read a source before i1 writes to it)*

- A read after write (RAW) data hazard refers to a situation where an instruction refers to a result that has not yet been calculated or retrieved.
- This can occur because even though an instruction is executed after a prior instruction, the prior instruction has been processed only partly through the pipeline.

For example:

i1: R2 <- R5 + R3

i2: R4 <- R2 + R3

- The first instruction is calculating a value to be saved in register R2, and the second is going to use this value to compute a result for register R4.
- However, in a pipeline, when operands are fetched for the 2nd operation, the results from the first have not yet been saved, and hence a data dependency occurs.
- A data dependency occurs with instruction i2, as it is dependent on the completion of instruction i1.

Write after read (WAR): *(i2 tries to write a destination before it is read by i1)*

- A write after read (WAR) data hazard represents a problem with concurrent execution.

For example:

i1. R4 <- R1 + R5

i2. R5 <- R1 + R2

- In any situation with a chance that i2 may finish before i1 (i.e., with concurrent execution), it must be ensured that the result of register R5 is not stored before i1 has had a chance to fetch the operands.

Write after write (WAW): *(i2 tries to write an operand before it is written by i1)*

- A write after write (WAW) data hazard may occur in a concurrent execution environment.

For example:

i1. R2 <- R4 + R7

i2. R2 <- R1 + R3

- The write back (WB) of i2 must be delayed until i1 finishes executing.

Solutions for Data Hazards

a) Stalling

b) Forwarding

c) Reordering

a) Stalling:

Consider the following example

add $1, $2, $3

sub $4, $5, $1

Earlier instruction produces a value used by a later instruction.

Cycle: Instruction	1	2	3	4	5	6	7	8	9	10
add	F	D	X	M	W					
sub		F	D	X	M	W				

Cycle: Instruction	1	2	3	4	5	6	7	8	9	10
add	F	D	X	M	W					
sub						F	D	X	M	W

b) ***Forwarding***: Connect new value directly to next stage

Data Hazard the root cause is the data dependency

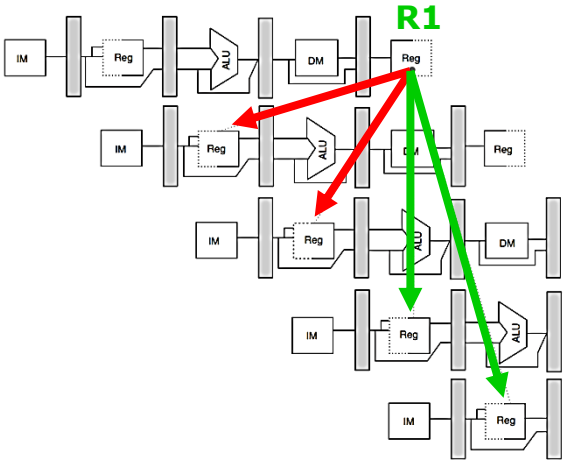
ADD R2, R3,**R1**

SUB R4, **R1**, R5

AND R6, **R1**, R7

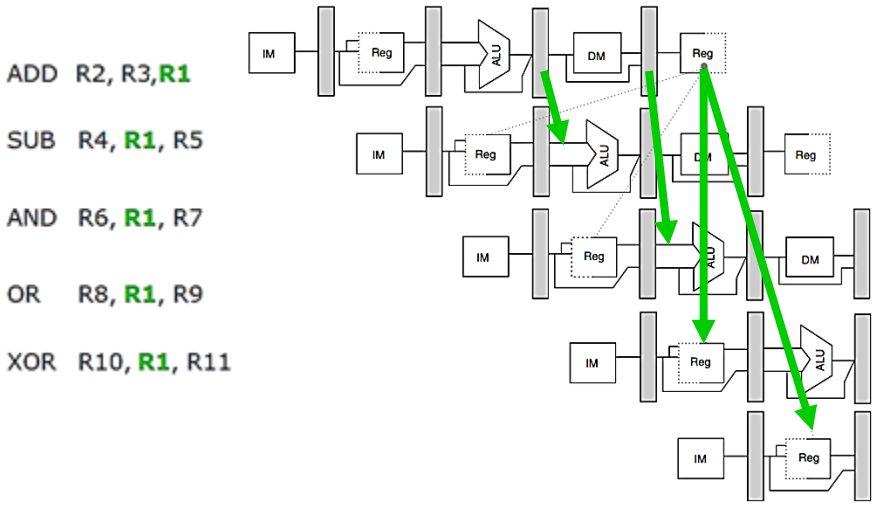
OR R8, **R1**, R9

XOR R10, **R1**, R11



To minimize data dependency stalls in the pipeline, **operand forwarding** is used.

Operand Forwarding: In operand forwarding, we use the interface registers present between the stages to hold intermediate output so that dependent instruction can access new value from the interface register directly.



2. Structural hazards

- This dependency arises due to the resource conflict in the pipeline.
- A structural hazard occurs when two (or more) instructions that are already in pipeline need the same resource.
- The result is that instruction must be executed in series rather than parallel for a portion of pipeline.
- Structural hazards are sometime referred to as resource hazards.

Example:

- A situation in which multiple instructions are ready to enter the execute instruction phase and there is a single ALU (Arithmetic Logic Unit).
- One solution to such resource hazard is to increase available resources, such as having multiple ports into main memory and multiple ALU (Arithmetic Logic Unit) units.

Another Example:

Instruction / Cycle	1	2	3	4	5
I_1	IF(Mem)	ID	EX	Mem	
I_2		IF(Mem)	ID	EX	
I_3			IF(Mem)	ID	EX
I_4				IF(Mem)	ID

– In the above scenario, in cycle 4, instructions I1 and I4 are trying to access same resource (Memory) which introduces a resource conflict.

Dealing with Structural Hazards

a) **Stall**

- low cost, simple
- Increases CPI
- use for rare case since stalling has performance effect

- To avoid this problem, we have to keep the instruction on wait until the required resource (memory in our case) becomes available. This wait will introduce stalls in the pipeline as shown below:

Cycle	1	2	3	4	5	6	7	8
I_1	IF(Mem)	ID	EX	Mem	WB			
I_2		IF(Mem)	ID	EX	Mem	WB		
I_3			IF(Mem)	ID	EX	Mem	WB	
I_4				–	–	–	IF(Mem)	

b) **Pipeline hardware resource**

- useful for multi-cycle resources
- good performance
- sometimes complex e.g., RAM

c) **Replicate resource**

- good performance
- increases cost (+ maybe interconnect delay)
- useful for cheap or divisible resources

Structural hazards are reduced with these rules:

- Each instruction uses a resource at most once
- Always use the resource in the same pipeline stage
- Use the resource for one cycle only

3. Control hazards (branch hazards or instruction hazards)

- This type of dependency occurs during the transfer of control instructions such as BRANCH, CALL, JMP, etc.
- A control hazard is when we need to find the destination of a branch, and can't fetch any new instructions until we know that destination.
- Control hazard occurs when the pipeline makes wrong decisions on branch prediction and therefore brings instructions into the pipeline that must subsequently be discarded.
- The term branch hazard also refers to a control hazard.
- Consider the following sequence of instructions in the program:

100: I1
101: I2 (JMP 250)
102: I3
.
.
250: BI1

- Expected output: I1 -> I2 -> BI1
- NOTE: Generally, the target address of the JMP instruction is known after ID stage only.

Instruction/ Cycle	1	2	3	4	5	6
I_1	IF	ID	EX	MEM	WB	
I_2		IF	ID (PC:250)	EX	Mem	WB
I_3			IF	ID	EX	Mem
BI_1				IF	ID	EX

- Output Sequence: I1 -> I2 -> I3 -> BI1
- So, the output sequence is not equal to the expected output that means the pipeline is not implemented correctly.

Dealing with Control Hazards

a) **Stall :** Stop fetching instruction until result is available

- To correct the problem we need to stop the Instruction fetch until we get target address of branch instruction. This can be implemented by introducing delay slot until we get the target

address.

Instruction/ Cycle	1	2	3	4	5	6
I_1	IF	ID	EX	MEM	WB	
I_2		IF	ID (PC:250)	EX	Mem	WB
Delay	-	-	-	-	-	-
BI_1				IF	ID	EX

- Output Sequence: I1 -> I2 -> Delay (Stall) -> BI1
- As the delay slot performs no operation, this output sequence is equal to the expected output sequence. But this slot introduces stall in the pipeline.
- Another example

Untaken branch instruction	IF	ID	EX	MEM	WB				
Instruction $i + 1$		IF	ID	EX	MEM	WB			
Instruction $i + 2$			IF	ID	EX	MEM	WB		
Instruction $i + 3$				IF	ID	EX	MEM	WB	
Instruction $i + 4$					IF	ID	EX	MEM	WB

Taken branch instruction	IF	ID	EX	MEM	WB				
Instruction $i + 1$		IF	**idle**	**idle**	**idle**	**idle**			
Branch target			IF	ID	EX	MEM	WB		
Branch target + 1				IF	ID	EX	MEM	WB	
Branch target + 2					IF	ID	EX	MEM	WB

b) Predict: Assume an outcome and continue fetching (undo if prediction is wrong)

- Solution for Control dependency Branch Prediction is the method through which stalls due to control dependency can be eliminated.
- Continue fetching as if we won't take the branch, but then invalidate the instructions if we do take the branch
- Simply treat every branch as untaken; when the branch is untaken, pipelining as if

Untaken branch instruction	IF	ID	EX	MEM	WB				
Instruction $i + 1$		IF	ID	EX	MEM	WB			
Instruction $i + 2$			IF	ID	EX	MEM	WB		
Instruction $i + 3$				IF	ID	EX	MEM	WB	
Instruction $i + 4$					IF	ID	EX	MEM	WB

- But if the branch is taken, turn fetched instruction into a no-op (idle) and restarts the IF at the branch target address.

Taken branch instruction	IF	ID	EX	MEM	WB				
Instruction $i + 1$		IF	**idle**	**idle**	**idle**	**idle**			
Branch target			IF	ID	EX	MEM	WB		
Branch target + 1				IF	ID	EX	MEM	WB	
Branch target + 2					IF	ID	EX	MEM	WB

c) Delayed branch

- Specify in architecture that the instruction immediately following branch is always executed.
- Always execute instructions following a branch regardless of whether or not we take it
- The instructions in the delay slots are always fetched. Therefore, we would like to arrange for them to be fully executed whether or not the branch is taken.
- The objective is to place useful instructions in these slots.
- The effectiveness of the delayed branch approach depends on how often it is possible to reorder instructions.

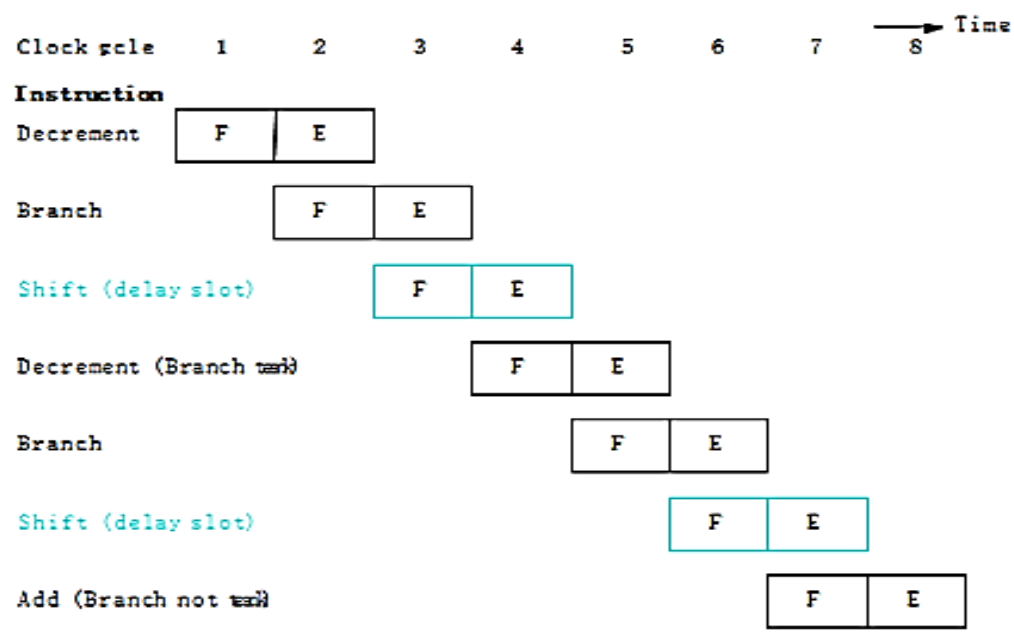
```
LOOP    Shift_left    R1
        Decrement     R2
        Branch=0      LOOP
NEXT    Add           R1,R3
```

(a) Original program loop

```
LOOP    Decrement     R2
        Branch=0      LOOP
        Shift_left    R1
NEXT    Add           R1,R3
```

(b) Reordered instructions

Reordering of instructions for a delayed branch.



Execution timing showing the delay slot being filled during the last two passes through the loop

MODULE – IV

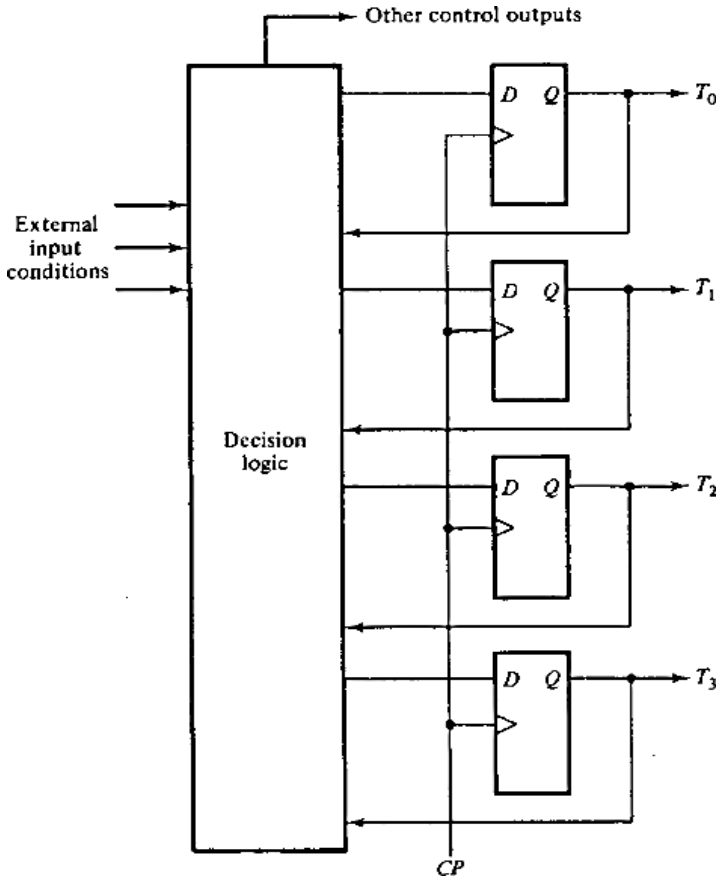
Control Logic Design: *Control organization – Hardwired control-microprogram control –control of processor unit - Microprogram sequencer, microprogrammed CPU organization -horizontal and vertical micro instructions.*

1. Control Unit organization

- The main responsibility of control unit is the generation of timing and control signals.
- The control unit is the circuitry that controls the flow of information through the processor, and coordinates the activities of the other units within it.
- The control unit directs the entire computer system to carry out stored program instructions.
- The control unit must communicate with both the arithmetic logic unit (ALU) and main memory.
- The control unit instructs the arithmetic logic unit that which logical or arithmetic operation is to be performed.
- The control unit co-ordinates the activities of the all other units as well as all peripherals and auxiliary storage devices linked to the computer.
- Goal of control logic design should be development of a circuit that implements the desired control sequence in a logical and straight forward manner.
- Designers used specialized methods for control logic design which is considered as the extension of the classical sequential logic method combined with register transfer method.
- The four methods of control organization are:
    1. **One flip-flop per state method**
    2. **Sequence register and decoder method**
    3. **PLA Control**
    4. **Microprogram control**
- The first two methods result in a circuit that must use SSI and MSI circuits for the implementation.
- A control unit implemented with SSI and MSI devices is said to be a hard-wired control.
- If any alterations or modifications are needed, the circuits must be rewired to fulfill the new requirements.
- The PLA or micro-program control which uses an LSI device such as a programmable logic array or a read-only memory.
- Any alterations or modifications in a micro-program control can be easily achieved without wiring changes by removing the ROM from its socket and inserting another ROM programmed to fulfill the new specifications.

1.1 One flip flop per state method

- One flip flop per state in the control sequential circuit
- One flip flop is set at any particular time; all others are cleared
- A single bit is made to propagate from one flip flop to other under the control of a decision logic
- Each flip flop represents a state and is activated only when the control bit is transferred to it
- Uses maximum number of flip flops
- Eg:- A sequential circuit with 12 states requires a minimum of 4 flip flop because $2^3 < 12 < 2^4$. Control circuit uses 12 flip flops one for each state



- 4 stage sequential control logic that uses 4 D-type flip flops: one flip flop per state Ti , i=0, 1, 2, 3
- At any given time interval between 2 clock pulses only one flip flop is equal to 1 all others are 0
- The transition from the present state to next state is a function of the present Ti that is a 1 and certain input conditions
- The next state is manifested when the previous flip flop is cleared and a new one is set
- Each of the flip flop is connected to the data processing section of the digital system to initiate certain microoperations
- The control outputs are a function of the T's and external inputs
- These outputs may also initiate microoperations

- If control circuit does not need external inputs for its sequencing, the circuit reduces to straight shift register with a single bit shifted from one position to the next
- If control sequence must repeated over and over again the control reduces to ring counter
- Ring counter is a shift register with the output of last flip flop connected to the input of the first flip flop.
- Also called ring counter controller. In a ring counter single bit continuously shifts from one position to the next in a circular manner.

Advantages

• Simple to design by inspecting the state diagram that describes the control sequence

• Offers a savings in design effort.

• An increase in operational simplicity.

• Decrease in the combinational circuits required to implement the complete sequential circuit.

Disadvantage

• Increase system cost since more flip flops are used

1.2 Sequence Register and Decoder Method

- Uses register to sequence the control states
- The register is decoded to provide one output for each state
- For n flip flops in the sequence register the circuit will have 2n states and decoder will have 2n outputs. Eg :- A 4 bit register can be in any one of 16 states
- A 4X16 decoder will have 16 outputs one for each state of the register

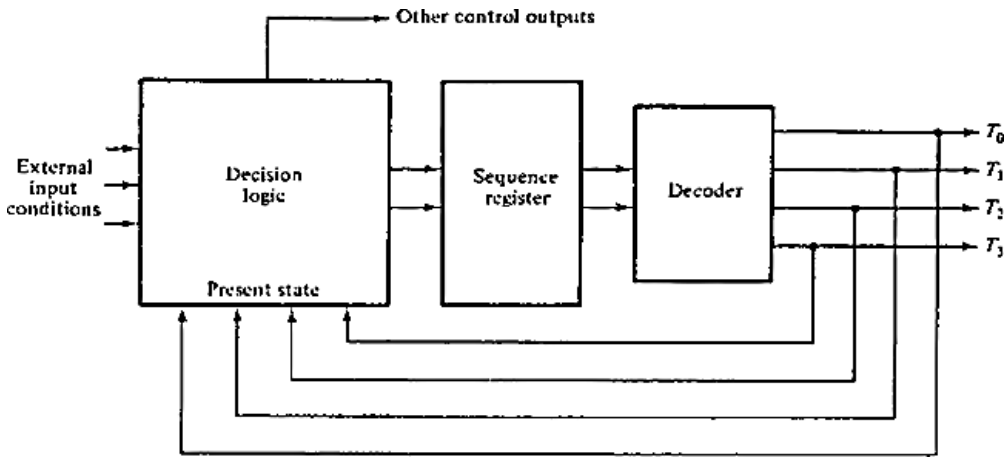


Figure: Control logic with sequence register and decoder

- 4 state sequential control logic
- The sequence register has 2 flip flops and the decoder establishes separate outputs for each state in the register

- The transition to the next state in the sequence register is a function of the present state and the external input conditions
- If the control circuit does not need external inputs the sequence register reduces to a counter that continuously sequence through the four states so called counter decoder method

1.3 PLA Control

- The programmable logic array (PLA) control is essentially similar to the sequence register and decoder method except that all combinational circuit are implemented with the PLA, including the decoder and decision logic.
- By using PLA for combinational circuit reduce the number of ICs and the number of interconnection wires.

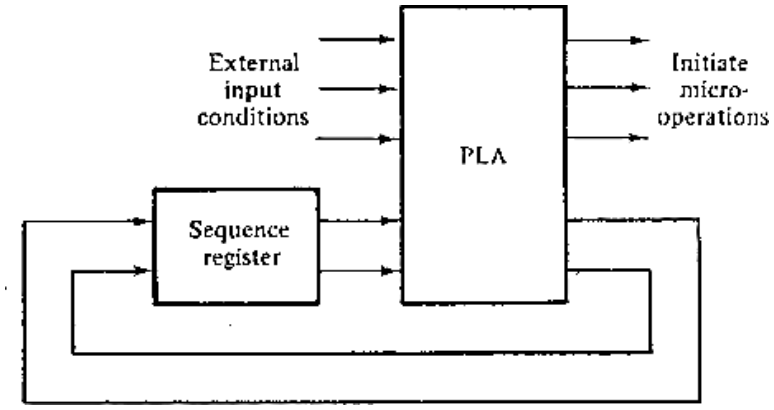


Figure: PLA control logic

- The external sequence register establishes the present state of the control circuit
- The PLA outputs determine which microoperations should be initiated depending on the external input conditions and the present state of the sequence register
- At the same time other PLA outputs determine the next state of the sequence register
- The sequence register is external to the PLA if the unit implements only combinational circuits

1.4 Microprogram Control

- Purpose of control unit is to initiate a series of sequential steps of microoperations
- Any given time certain operations are to be initiated while all others remain idle
- Control variable at any given time can be represented by a string of 1's and 0's called control word
- Control words can be programmed to initiate the various components in the system in an organized manner
- A control unit whose control variables are stored in a memory called a Microprogrammed control unit

- Microinstruction :- Each control word of memory
- Microprogram :- Sequence of microinstructions

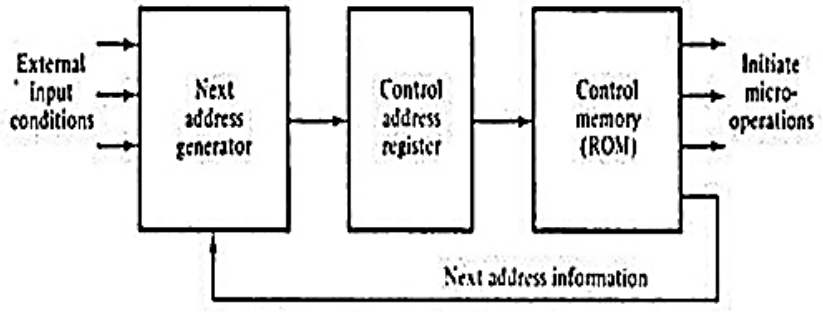


Figure: Microprogram control logic

- Control memory is usually ROM since alterations of Microprogram is seldom needed
- Use of Microprogram involves placing all control variables in words of the ROM for use by the control unit through successive read operations.
- The content of the word in the ROM at a given address specifies the microoperations for the system.
- A ROM, PLA or WCM when used in a control unit is referred as a control memory.
- Control memory address register specifies the control word read from control memory
- ROM operates as a combinational circuit with address value as the input and the corresponding word as the output
- The content of the specified word remains on the output wires as long as the address value remains in the address register
- The word out of the ROM should be transferred to a buffer register if the address register changes while the ROM word is still in use
- If the change in address and ROM word can occur simultaneously no buffer register is needed
- The word read from memory represents a microinstruction.
- The microinstruction specifies one or more microoperations for the components of the system
- Once these operations are executed the control unit must determine its next address
- The location of the next microinstruction may be next one in the sequence or it may locate somewhere else in the control memory
- Some bits of the microinstruction to control the generation of the address for the next microinstruction
- The next address may be function of external input conditions
- The next address is computed in the next address generator circuit and then transferred into the control address register to read the next microinstruction.

2. Design of hardwired control

- The main objective of control unit is to generate the control signal in proper sequence.
- The control hardware can be viewed as a state machine that changes from one state to another in every clock cycle, depending on the contents of the instruction register, the condition codes and the external inputs.
- The outputs of the state machine are the control signals.
- The sequence of the operation carried out by this machine is determined by the wiring of the logic elements and hence named as "hardwired".
- In this hardwired control techniques, the control signals are generated by means of hardwired circuit.
- The control signals depend on the instruction, i.e. the contents of the instruction register.
- The execution of some of the instructions depends on the contents of condition code or status flag register.
- The status flags represent the various state of the CPU and various control lines connected to it, such as MFC status signal.
- The structure of control unit can be represented in a simplified view by putting it in block diagram.

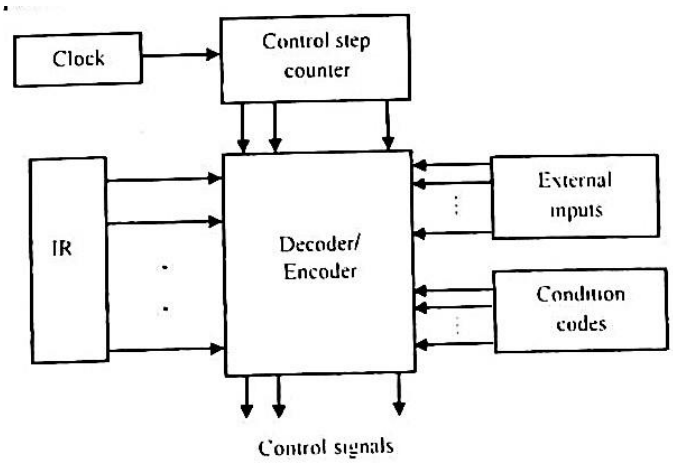


Fig : Control Unit Organization

- The decoder/encoder block is simply a combinational circuit that generates the required control outputs depending on the state of all its input.
- Control step counter keeps tracks of the count of control steps.
- It is required to generate many control signals by the control unit. These are basically coming out from the encoder circuit of the control signal generator. The control signals are lie PCin PCout, ZIn, Zout , MARin , Add , End etc.

3. Control of Processor Unit

- General purpose Microprogram control unit must have control memory large enough to store many microinstructions
- Include all possible control variables in the system - controlling ALU, Multiplexer, status bits
- Provision must be available to accept an external address to initiate many operations
- Advantage of Microprogram control
  - Once the hardware configuration is established there should be no need for further hardware or wiring changed
  - Only change Microprogram reside in the control memory for different operations

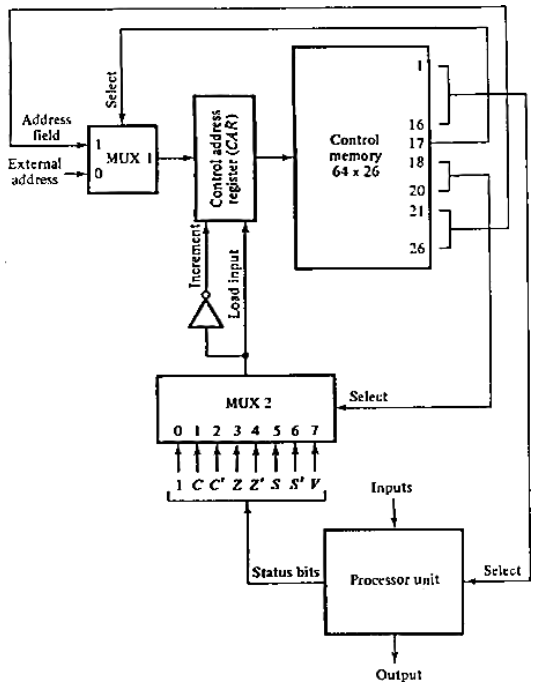


Figure: Microprogram control for processing unit

- Control memory of 64 words with 26 bits
- To select 64 words require an address of 6 bits
- To select 8 status bits 3 selection lines for multiplexer
- 1 bit of the microinstruction selects between an external address and address field of the microinstruction
- First 16 bits of the microinstruction select the microoperation for the processor
- The other 10 bits select the next address for the control address register
- The status bits from the processor are applied to the inputs of the multiplexer
- Both normal and complement values are used for status bits except for overflow bit V
- Input 0 of the of MUX2 is connected to a binary constant which is always 1

- The load of CAR is enabled when the input is selected by bits 18,19 and 20 in the microinstruction causes a transfer of information from output of MUX1 into CAR
- The input into CAR is a function of bit 17 in the microinstruction Bit 17 = 1: CAR receives the address field of microinstruction Bit 17 = 0: External address is loaded into CAR
- External address is for the purpose of initiating a new sequence of microinstructions which can be specified by the external environment

4. Micro-programmed Control

- Each micro operation is associated with a specific set of control lines which, when activated, causes that micro operation to take place.
- Since the number of instructions and control lines is often in the hundreds, the complexity of hardwired control unit is very high.
- The hardwired control unit is relatively inflexible because it is difficult to change the design, if one wishes to correct design error or modify the instruction set.

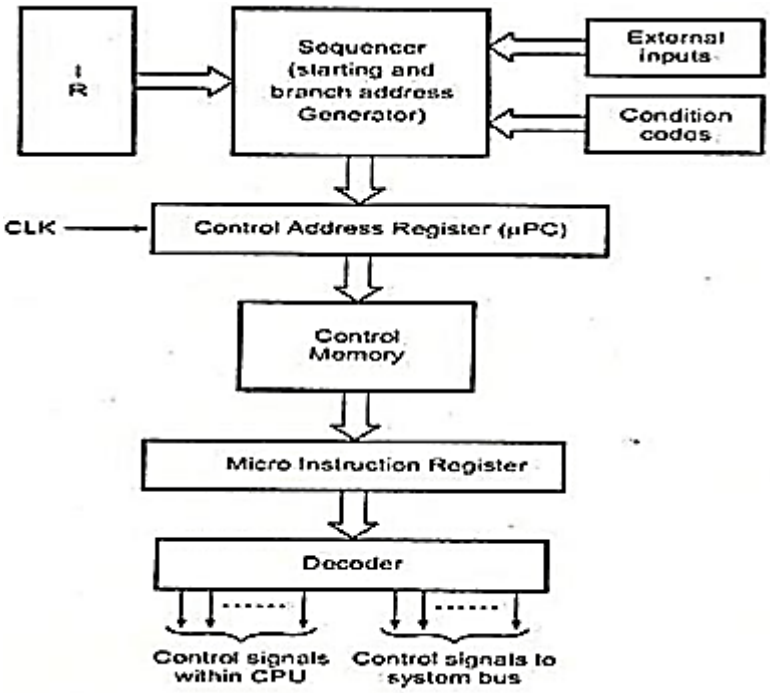


Fig. Microprogrammed control unit

- The micro programmed control unit consists of
  - *Control memory :*
    - In micro programmed control, the micro programs for all instructions are stored in the control memory (CM)
    - The control signals to be activated at any time are specified by a microinstruction, which is fetched from Control memory (CM).

- *Control address register*
  - The control address register holds the address of the next microinstruction to be read.
  - When address is available in control address register, the sequencer issues READ command to the control memory.
- *Microinstruction register*
  - After issue of READ command, the word from the addressed location is read into the microinstruction register.
  - Now the content of the micro instruction register generates control signals and next address information for the sequencer.
- *Microprogram sequencer*
  - A sequence of one or more micro operations designed to control specific operation, such as addition, multiplication is called a micro program.
  - The sequencer loads a new address into the control address register based on the next address information

Advantages of Micro programmed control

- It simplifies the design of control unit. Thus it is both, cheaper and less error prone implement.
- Control functions are implemented in software rather than hardware.
- The design process is orderly and systematic
- More flexible, can be changed to accommodate new system specifications or to correct thedesign errors quickly and cheaply.
- Complex function such as floating point arithmetic can be realized efficiently.

Disadvantages

- A micro programmed control unit is somewhat slower than the hardwired control unit,because time is required to access the microinstructions from CM
- The flexibility is achieved at some extra hardware cost due to the control memory and itsaccess circuitry.

5. Microinstructions

Each Microprogram is the sequence of microinstructions. And these microinstructions are executed in sequence. The execution sequence is maintained by Microprogram counter.

Example: Control Sequence for instruction Add (R3), R1:

1. PCout, MARin, Read, Select4, Add, Zin
2. Zout, PCin, Wait for the MFC
3. MDRout, IRin

4. R3out, MARin, Read
5. R1out, Yin, Wait for MFC
6. MDRout, Select Y, Add, Zin
7. Zout, R1in, End

Techniques of grouping of Control Signals

- The grouping of control signal can be done either by using technique called
  a) Horizontal organization
  b) Vertical organization

a) Horizontal Micro-Instructions

- The scheme of micro-instruction by assigning one bit position to each control signal is called horizontal micro-instructions.

Micro - instruction	..	PC_{in}	PC_{out}	MAR_{in}	Read	MDR_{out}	IR_{in}	Y_{in}	Select	Add	Z_{in}	Z_{out}	$R1_{out}$	$R1_{in}$	$R3_{out}$	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	0	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	0	1	0	0	1	

- Each microinstruction is nothing but the combination of 0's and 1's which is known as control word.
- Each position of control word specifies a particular control signal. On the control word means that a low signal value is generated for that control signal at that particular instant of time, similarly 1 indicates a high signal.
  Example: 0111000111000000

Advantage: It is suitable when operating speed of computer is a critical factor

b) Vertical Micro-Instructions

- We can reduce the length of the horizontal micro-instruction so easily by implementing another method known as vertical micro-instructions.
- In this case, Most signals are not needed simultaneously and many others are mutually exclusive.

F1 (4 bits)	F2 (3 bits)	F3 (3 bits)	AS3 AS2 AS1	Read	Write	Carry-in	WMFC	End
0000: No action	000: No action	000: No action	000: ADD					
0001: PCout	001: PCin	001: MARin	001: INC					
0010: MDRout	010: IRin	010: MDRin	010: SUB					
0011: Zout	011: Zin	011: TEMPin	011: DEC					
0100: R0out	100: R0in	100: Yin	100: AND					
0101: R1out	101: R1in		101: OR					
0110: R2out	110: R2in		110: XOR					
0111: R3out	111: R3in		111: NOT					
1000: IRout								
1001: TEMPout								

Advantage

• Fewer bits are required in the microinstruction.

Disadvantage

• Vertical approach results in slower operations speed.

Comparison between Horizontal and Vertical Organization

Sl.No	Horizontal	Vertical
1.	Long formats.	Short formats
2.	Ability to express a high degree of parallelism.	Limited ability to express parallel micro operations.
3.	Little encoding of the control information.	Considerable encoding of the control information
4.	Useful when higher operating speed is desired.	Slower operating speeds.

6. Microprogram Sequencer

• Micro program sequencer is a control unit which does the tasks of Microprogram sequencing.

• There are two important factors must be considered while designing the micro program sequencer.

o The size of the microinstruction

o The address generation time

• Microprogram sequencer is attached to the control memory.

• It inspects certain bits in the microinstruction to determine the next address for control memory.

• A typical sequencer has the following address sequencing capabilities.

1. Increments the present address of control memory
2. Branches to an address which will be specified in the bits of microinstruction
3. Branches to a given address if a specified status bit is equal to 1.
4. Transfers control to a new address as specified by an external source
5. Has a facility for subroutines calls and returns.

• The block diagram of Microprogram Sequencer is shown in the below figure.

• It consists of a multiplexer that selects an address from four sources and routes it into a control address register.

• The O/P from CAR provides the address for control memory.

• The contents of CAR are incremented and applied to the multiplexer and to the stack register file.

• The register selected in the stack is determined by stack pointer. Inputs (I_0-I_2) specify the operation for the sequencer and input T is the test point for a status bit.

• Initially the address register is cleared to zero and clock pulse synchronizes the loading into registers.

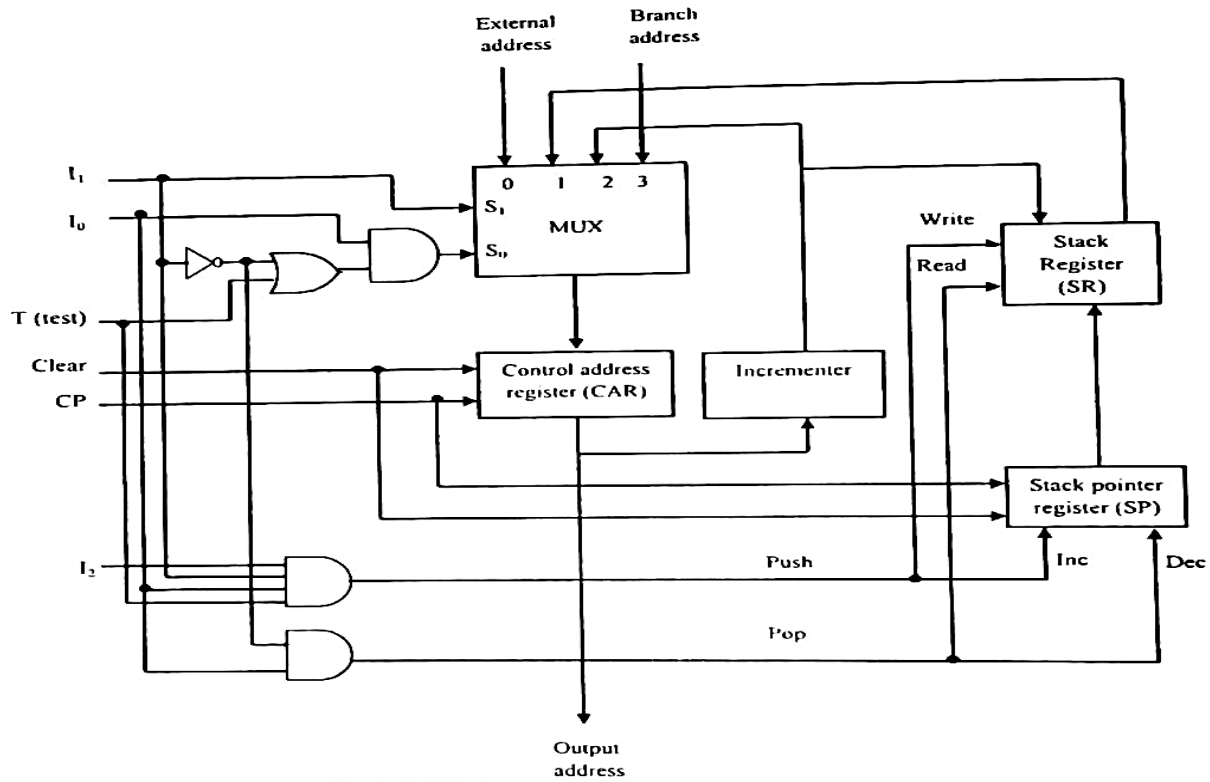


Fig: Block diagram of Microprogram Sequencer

- The following function diagram illustrates the operation of a sequencer.

Function table

I_2	I_1	I_0	T	s_1	s_0	Operation	Comments
X	0	0	X	0	0	$CAR \leftarrow EXA$	Transfer external address
X	0	1	X	0	1	$CAR \leftarrow SR$	Transfer from register stack
X	1	0	X	1	0	$CAR \leftarrow CAR + 1$	Increment address
0	1	1	0	1	0	$CAR \leftarrow CAR + 1$	Increment address
0	1	1	1	1	1	$CAR \leftarrow BRA$	Transfer branch address
1	1	1	0	1	0	$CAR \leftarrow CAR + 1$	Increment address
1	1	1	1	1	1	$CAR \leftarrow BRA, SR \leftarrow CAR + 1$	Branch to subroutine

7. Microprogrammed CPU Organization

Digital computer consists of

1. Central Processing Unit (CPU)
2. Memory unit
3. Input-output devices

• CPU can be divides into 2 distinct and interactive sections

1. Processing section
   - Useful device for constructing the processor section of a CPU
2. Control section
   - Controlling the entire units of computer
   - Microprogram sequencer - constructing a Microprogram control of CPU

• **Microprogrammed computer**

A computer CPU use the Microprogram sequencer

• Microprogrammed computer consists of

1. Memory unit
   - Stores instructions and data supplied by the user through an input device
2. Two processor unit
   - Data processor :- Manipulates data
   - Address processor :- Manipulates the address information received from memory
3. A Microprogram sequencer
4. A control memory
5. Other digital functions

• An instruction extracted from memory unit during fetch cycle goes into instruction register.

• Code transformation constitutes a mapping function that is needed to convert the operation code bits of an instruction into a starting address for the control memory and is implemented with ROM or PLA

• Mapping concept provides flexibility for adding instructions or microoperations for control memory as need arises.

• The address generated in code transformation mapping function is applied to the external address (EXA) input of the sequencing

Microprogram control unit consists of

1. The sequencer
   - Generates next address
2. A control memory
   - Reads the next macroinstruction while present macroinstruction are being executed in the other units of the CPU
   - For storing microinstructions

3. A multiplexer
   - Selects one of the many status bits and applies to the T(test) input of the sequencer
   - One of the input of the multiplexer is always I to provide an unconditional branch operation
4. A pipeline register
   - Speed up the control operation
   - Allows next address to be generated and the output of control memory to change while the control word in pipeline register initiates the micropertaions given by present microinstruction
   - Not always necessary
   - The output of control memory can go directly to the control inputs of the various units in the CPU

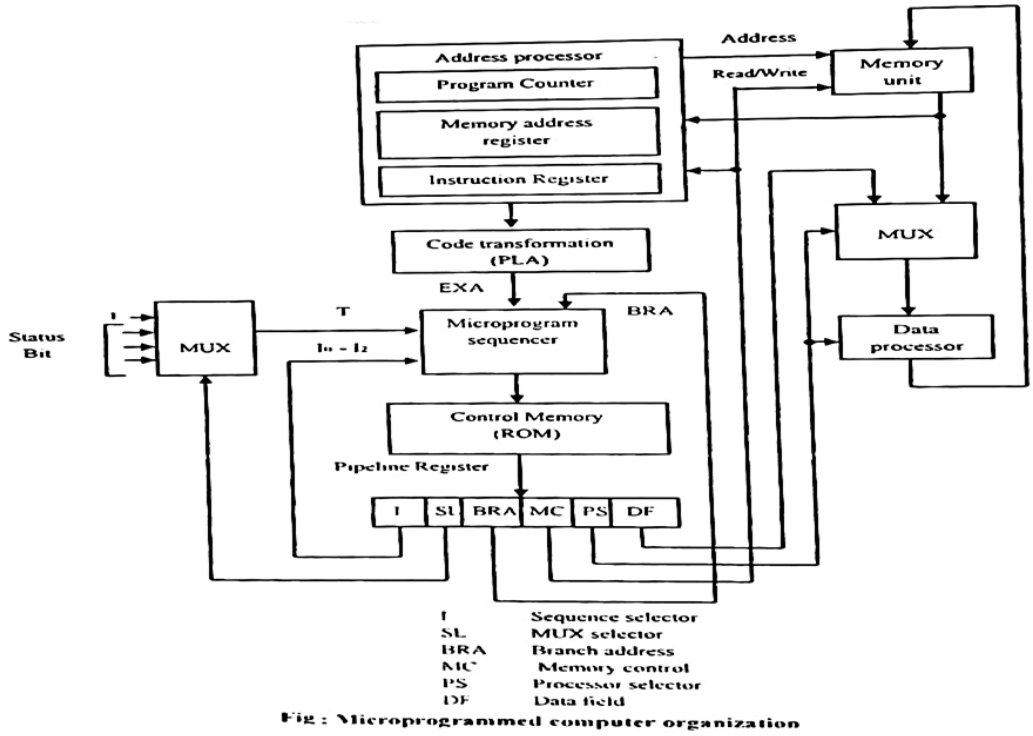


Fig : Microprogrammed computer organization

• **Microinstruction format:-**

- Contains 6 fields
- First 3 fields(I,SL, BRA) provide information to the sequencer to determine the next address for control memory
  - I field (3 bits) :- Supplies input information for the sequencer
  - SL field :- Selects a status bit for the multiplexer
  - BRA field :- Address field of microinstruction and supplies a branch address(BRA) to the sequencer

- The next 3 fields(MC, PS, DF) are for controlling microoperations in the processor and the memory units
    - Memory control (MC) field :- Controls the address processor and the read and write operations in the memory unit
    - The processor select(PS) field :- Controls the operations in the data processor unit
    - The data field(DF) :- Used to introduce constants into the processor
- Output from data field may be used to set up control registers and introduce data in processor registers

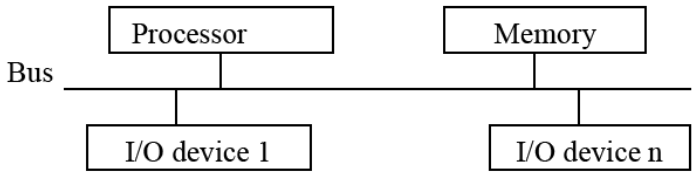
MODULE – V

***I/O organization**: accessing of I/O devices – interrupts, interrupt hardware -Direct memory access.* ***Memory system:** basic concepts – semiconductor RAMs. memory system considerations – ROMs, Content addressable memory, cache memories - mapping functions.*

1. Accessing I/O Devices

- Most modern computers use single bus arrangement for connecting I/O devices to CPU & Memory
- The bus enables all the devices connected to it to exchange information

Single Bus Structure



- Bus consists of 3 set of lines : Address, Data & Control Lines
- Processor places a particular address on address lines
- Device which recognizes this address responds to the commands issued on the Control lines
- Processor requests for either Read / Write
- The data will be placed on Data lines

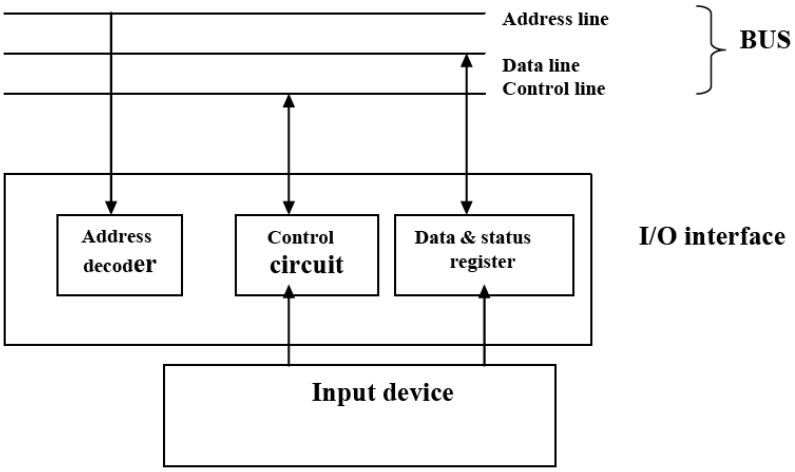


Fig: I/O Interface for an Input Device

Address Decoder:

- It enables the device to recognize its address when the address appears on address lines.

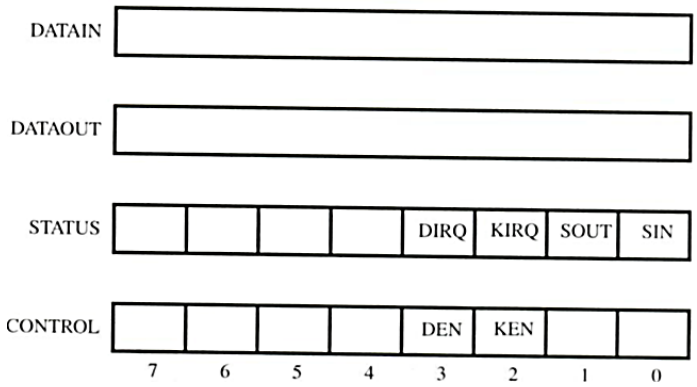
Data register

- It holds the data being transferred to or from the processor.

Status register

- It contains information relevant to the operation of the I/O devices.
- The address decoder, data & status registers and the control circuitry required to co-ordinate I/O transfers constitute the device‟s I/O interface circuit.

- For an input device, SIN status flag is used SIN = 1, when a character is entered at the keyboard, SIN = 0, once the char is read by processor.
- For an output device, SOUT status flag is used.



DIRQ - Interrupt Request for display.
KIRQ - Interrupt Request for keyboard.
KEN - Keyboard enable.
DEN - Display Enable.
SIN, SOUT - Status flags.

- The data from the keyboard are made available in the DATAIN register & the data sent to the display are stored in DATAOUT register.

2. Modes of Data Transfer

2.1 Programmed I/O Mode

- In programmed I/O mode data are exchanged between the processor and the I/O module.
- When a processor is executing a program and encounters an instruction relating to I/O, it executes that instruction by issuing a command to that appropriate I/O module.
- The I/O commands issued by the processor to the I/O module are Test, Control, Read, Write.

2.2 Memory Mapped I/O

- There is a single address space for memory location and I/O devices.(the address space is shared)
- With memory mapped I/O a single read line a single write line are needed on the bus. The bus may be equipped with memory read and write plus Input and output command lines.
- Now the command line specifies whether the address refers to memory location or an I/O device.
- Most CPU uses memory mapped I/O.
- It deals with fewer address lines.

2.3 Interrupt Driven I/O

- There are many situations where tasks can be performed while waiting for an I/O device to be ready, to allow this the I/O device should alert the processor when it becomes ready. It can be done by sending a hardware signal called an interrupt.
- In this method the program issues an I/O command and then continues to execute untill it is interrupted by the I/O hardware to signal the end of I/O operation.

- The routine executed in response to an interrupt request is called **Interrupt Service Routine(ISR)**.
- The processor first completes execution of instruction then it loads the program counter(pc) with the address of 1st instruction of ISR.

2.4 Direct Memory Access

- In this method the input and output devices read/write information from the main memory without interference of the CPU.
- By DMA approach, large blocks of data at high speed can be sent between external device and main memory.
- DMA Controller allows the data transfer between I/O device and memory.
- DMA controllers act as a processor but it is controlled by the CPU.
- To initiate the transfer of a block of words, the processor sends *starting address of memory block,* the word count, Control to specify the mode of transfer such as read or writes etc. to the DMA controller.
- The DMA controller performs the required I/O operation and sends an interrupt to the processor upon completion.

3. Interrupts

- To avoid processor waiting time, we can arrange the I/O device to alert the processor when it becomes ready.
- It can do so by sending a hardware signal called an ***interrupt*** to the processor, so that the processor doesn't have to continuously check the device status.
- One of the bus control lines named as interrupt request line is dedicated for this purpose.
- The routine executed in response to an interrupt request is called Interrupt Service Routine (ISR).

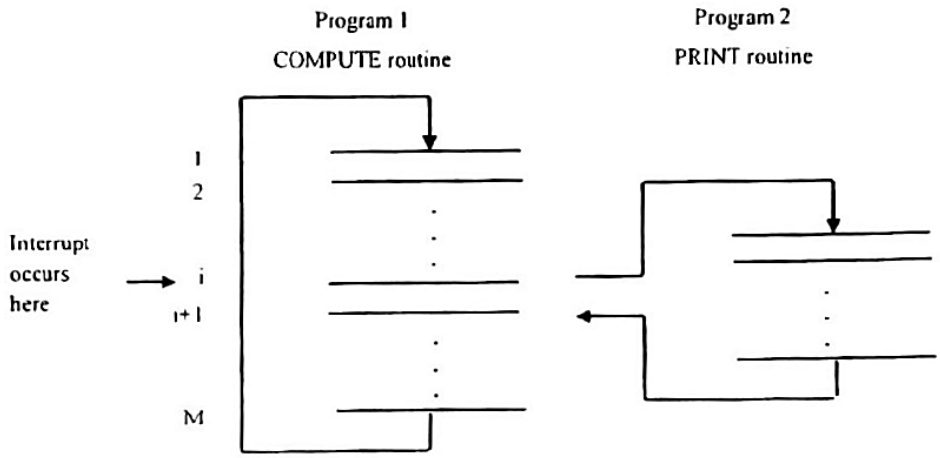


Figure Transfer of control through the use of interrupts.

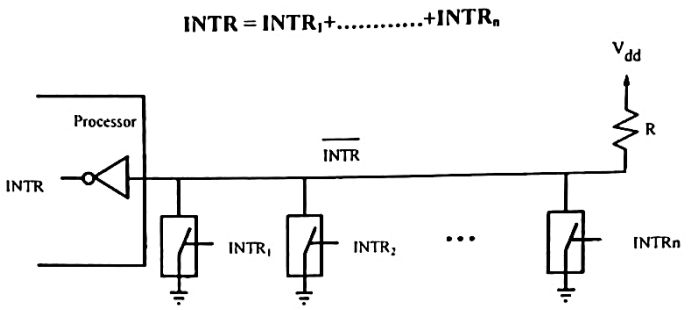
Sequence of events involved in handling an Interrupt

- Assume that an interrupt requests arrives during the execution of instruction i.
- The processor first completes the execution of instruction i.
- Then it loads the PC (Program Counter) with the address of the first instruction of the ISR.
- After the execution of ISR, the processor has to come back to instruction i + 1.

- Therefore, when an interrupt occurs, the current contents of PC which point to i+1 is put in temporary storage in a known location.
- The processor saves only the contents of program counter & status register.
- The task of saving and restoring the information can be done automatically by the processor.
- Saving registers also increases the delay between the time an interrupt request is received and the start of the execution of the ISR. This delay is called the Interrupt Latency.
- When the processor is handling the interrupts, it must inform the device that its request has been recognized so that it removes its interrupt requests signal.
- This may be accomplished by a special control signal called the interrupt acknowledge signal.

3.1 Hardware interrupts

- A single interrupt request line may be used to serve "n" devices. All devices are connected to the line via switches to ground.



- To request an interrupt, a device closes its associated switch, the voltage on INTR line drops to 0(zero).
- If all the interrupt request signals (INTR1 to INTRn) are inactive, all switches are open and the voltage on INTR line is equal to Vdd.
- When a device requests an interrupts, the value of INTR is the logical OR of the requests from individual devices.

  (ie) INTR = INTR1+……+INTRn
- Resistor "R" is called a pull-up resistor because it pulls the line voltage upto the high voltage state when the switches are open.

3.2 Enabling and Disabling Interrupts

- We have to ensure that active interrupt request signal does not lead to successive interruptions.

First Method:

- Processor hardware ignores the IRQ (Interrupt request) line until the execution of first instruction of ISR has been completed.
- Dl (Disable interrupts) instruction will be used as first instruction in ISR (Interrupt Service Routine).
- Last instruction in ISR is El, before RET instruction.

Second Method

- Processor automatically disables interrupts before starting the execution of ISR.
- For this, one bit in Processor Status Register called Interrupt Enable is used to indicate whether interrupts are enabled or not. The interrupt request will be received only when this bit is one.

Third Method

- Processor has a special IRQ line for which the interrupt-handling circuit responds only to the leading edge of the signal.
- Such a line is said to be edge-triggered. In this case, the processor will receive only one request, regardless of how long the line is activated.
- There is no chance for multiple interruptions and hence no need for explicit disable interrupts.

3.3 Handling Multiple Devices

- When more than one device is connected to a single interrupt Request line, additional information is needed to identify the device that activates the IRQ line.

Polling

- With the help of one bit in status register, it is able to identify the device who initiates the interrupt request.
- The simplest way to identify the interrupting device is poll all the I/O devices connected to the bus.
- The first device encountered with IRQ bit set is the device that should be serviced.
- Advantage is It is easy to implement.
- Disadvantage So much time is wasted by interrogating the IRQ bit of all devices that may not be requesting any service.

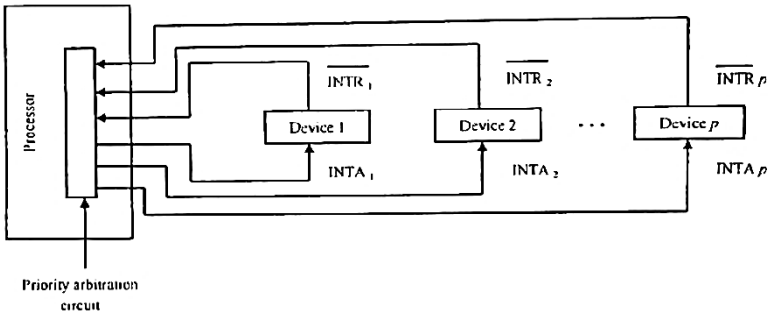
Vectored Interrupt

- Here the device requesting an interrupt may identify itself to the processor by sending a special code over the bus& then the processor start executing the ISR.
- The code supplied by the processor indicates the starting address of the ISR for the device. The code length ranges from 4 to 8 bits.
- The location pointed to by the interrupting device is used to store the staring address to ISR.
- The processor reads this address, called the interrupt vector& loads into PC.
- The interrupt vector also includes a new value for the Processor Status Register.
- When the processor is ready to receive the interrupt vector code, it activate the interrupt acknowledge (INTA) line.

Interrupt Nesting

- I/O devices organized in a priority structure.
- Interrupt request from a higher priority device be accepted while the processor is servicing another request from a lower priority device.
- A long delay in responding to an interrupt request may lead to erroneous situation for some devices that type of interrupt request has to be handled in top priority fashion.
- Priority level of the processor is the priority of the program currently being executed.

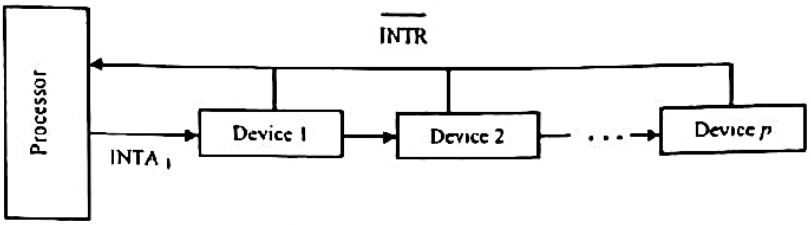
- Processor accepts interrupts only from devices that have priorities higher than the program currently being executed.
- Processor is in supervised mode only when executing OS routines. It switches to user mode before executing application programs.
- A multiple priority scheme can be implemented easily by using separate interrupt request and interrupt acknowledge lines for each device. This is shown in the following figure.



- Each of the interrupt request line is assigned a different priority level.
- Interrupt requests received over these lines are cent to a priority arbitration circuit in the processor.
- A request is accepted only if it has higher priority level than that currently assigned to the processor.

Simultaneous Requests

- If simultaneous request arrives from 2 or more devices, then processor must have some means of deciding which request to service first.
- A widely used scheme is to connect the devices to form a daisy chain, as in below figure.

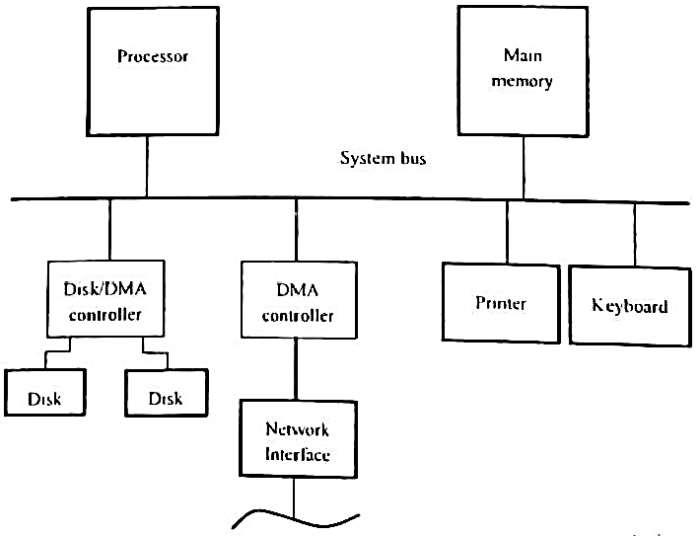


- Here all interrupt devices are serially connected. Higher priority device places first closed to the processor. Second device along the chain have second priority, third device along the chain have third priority and so on.
- Interrupt request line (INTR) is common to all devices.
- CPU responds to the interrupt via Interrupt Acknowledge line (INTA)
- INTA signal propagates serially through the devices. When several devices raise INTR, processor responds by setting INTA line to 1. It is received by device 1. It passes to device 2 only if device does not require any service.

4. Direct Memory Access

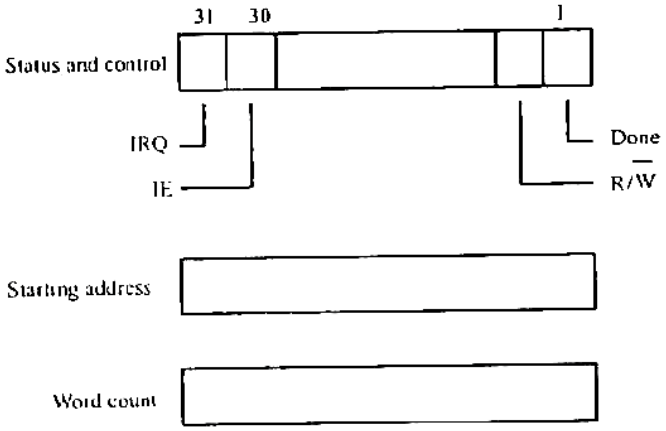
- Direct Memory Access (DMA) is the mechanism to allow the transfer of large block of data at high speed directly between the external device and main memory, without continuous intervention by the processor.
- DMA transfers are performed by a control circuit called the DMA Controller.

- To initiate the transfer of a block of words , the processor sends Starting address, Number of words in the block, Direction of transfer etc to the DMAC
- When a block of data is transferred , the DMA controller increment the memory address for successive words and keep track of number of words and it also informs the processor by raising an interrupt signal.
- When the entire block has been transferred, the controller informs the processor by raising an interrupt signal.



4.1 Registers in a DMA Interface

- Two registers are used for storing the starting address and the word count.
- Third register contains the status and control flags.



- R/W determines the direction of transfer.
- Done flag will be set to one when the controller has completed the whole data transfer and is ready to receive another command.
- IRQ bit is set to 1 when it has requested an interrupt.
- Interrupt Enable (IE) flag will be set to 1 means that it causes the controller to raise an interrupt after it has completed transferring a block of data.

4.2 Cycle stealing

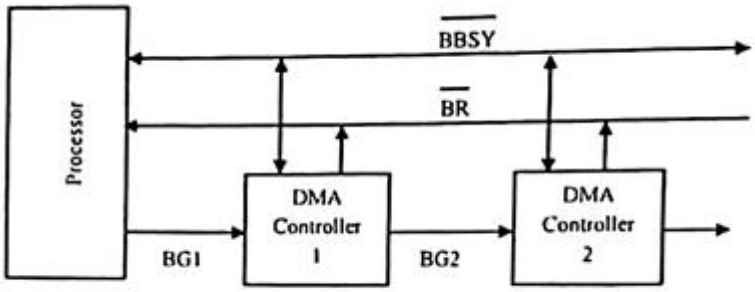
- Requests by DMA devices for using the bus (for memory access) are always given higher priority than processor requests.
- Among different DMA devices, top priority is given to high-speed peripherals (disks. high-speed network interface, and graphics display device). Since the processor originates most memory access cycles, it is often stated that DMA steals memory cycles from the processor ***(cycle stealing).***
- If DMA controller is given exclusive access to the main memory to transfer a block of data without interruption, this is called ***block or burst mode.***

4.3 Bus Arbitration

- Bus Arbitration is needed to resolve conflicts with *more than one device* (2 DMA Controllers and processor. etc...) *try to use the bus* to access main memory.
- The device that is allowed to initiate bus transfers on the bus at any given time is called bus master.
- The selection of next device to become bus master and bus mastership is transferred to it is known as bus arbitration.
- There are two approaches to bus arbitration.
  - Centralized arbitration (A single bus arbiter performs arbitration)
  - Distributed arbitration (all devices participate in the selection of next bus master).

4.3.1 Centralized Arbitration

- A single arbiter performs the arbitration.
- Bus arbiter may be processor or a separate unit connected to the bus.

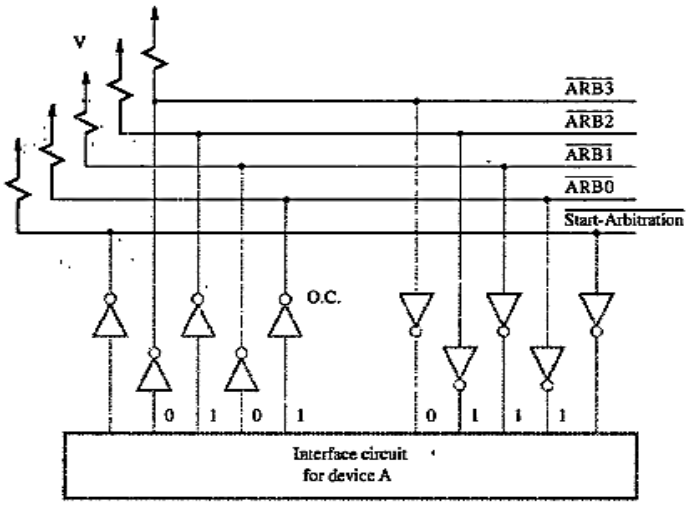


A Simple arrangement for bus arbitration using a daisy chain

- A DMA controller indicates that it needs to become the bus master by activating the Bus Request line (BR).
- When BR is activated the processor activates the Bus Grant Signal (BGI) and indicated the DMA controller that they may use the bus when it becomes free.
- This signal is connected to all devices using a daisy chain arrangement.
- If DMA requests the bus, it blocks the propagation of Grant Signal to other devices and it indicates to all devices that it is using the bus by activating Bus Busy (BBSY) line.

4.3.2 Distributed Arbitration.

- It means that all devices waiting to use the bus have equal responsibility in carrying out the arbitration process without using a central arbiter.
- Here all devices participate in the selection of the next bus master.
- No central arbiter is used.
- Each device on bus is assigned a 4-bit identification number.
- When one or more devices request the bus, they assert the Start-Arbitration signal and place their 4-bit ID number on ARB_0 to ARB_3
- A winner is selected as a result of the interaction among the signals transmitted over these lines.



- The net outcome is that the code on the four lines represents the request that has the highest ID number.
- The drivers are of open collector type. Hence, if the input to one driver is equal to 1, the input to another driver connected to the same bus line is equal to 0, then bus will be in low-voltage state.
- Advantage is that it offers higher reliability (operation of the bus is not dependent on any one device).
- SCSI bus is an example of distributed (decentralized) arbitration.

Example

- Assume two devices A & B have their ID 5 (0101), 6(0110). They are requesting the use of bus.
- Device A transmits the pattern 0101 and B transmits 0110. The code seen by both devices is 0111.
- Each device compares the pattern on the arbitration line to its own ID starting from MSB.
- If it detects a difference at any bit position, it disables the drivers at that bit position and for all lower order bits.
- It does this by placing 0 at the input of these drivers.
- In our example. A detects a difference in line ARB1; hence it disables the drivers on lines ARB1 & ARB0.

- This causes the pattern on the arbitration line to change to 0110 which means that B has won the contention.
- Note that since the code on the priority line is 0111 for a shorter period, device B may be temporarily disabling its driver on line ARB0. However, it will enable this driver once it sees a 0 on line ARB1 resulting from the action by device A.

5. Memory System

- Programs and data they operate on are resided in the memory of the computer.
- The execution speed of the program is dependent on how fast the transfer of data and instructions in-between memory and processor.
- There are three major types of memory available: Cache, Primary and Secondary Memories.
- A good memory would be fast, large and inexpensive. Unfortunately, it is impossible to meet all three of these requirements simultaneously. Increased speed and size are achieved at increased cost.
- A memory unit is considered as a collection of cells, in which each cell is capable of storing a bit of information.
- It stores information in group of bits called byte or word.
- The maximum size of the memory that can be used in any computer is determined by the addressing scheme.

Address	Memory Locations
16 Bit	2^{16} = 64 K
32 Bit	2^{32} = 4G (Giga)
40 Bit	2^{40} = IT (Tera)

- **Word length** is the number of bits that can be transferred to or from the memory, it can be determined from the width of data bus, if the data bus has a width of n bits, it means word length of that computer system is n bits.
- **Memory access time** is the time elapses between the initiation of an operation and the completion of that operation.
- **Memory cycle time** is the minimum time delay required between the initiations of two successive memory locations.

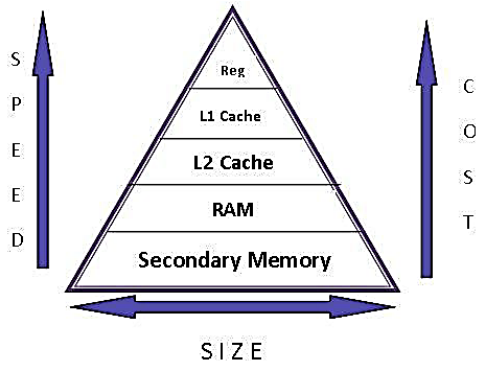
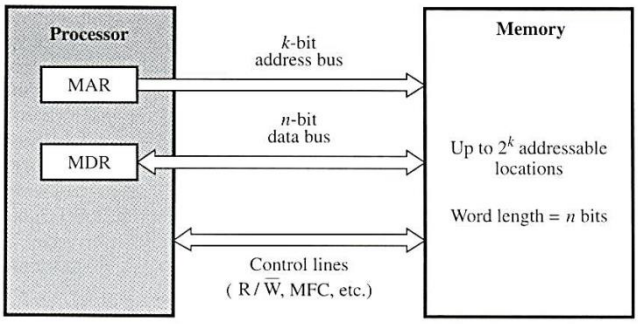


Fig : memory hierarchy

- In the memory hierarchy, speed will decrease and size will increase from top to bottom level.
- An important design issue is to provide a computer system with as large and fast a memory as possible, within a given cost target.
- Compared to processor, the main memory unit is very slow. So in order to transfer something between memory and processor takes a long time. The processor has to wait a lot.
- To avoid this speed gap between memory and processor a new memory called cache memory is placed in between main memory and processor.
- Random Access Memory (RAM) is a memory system in which any location can be accessed for a Read or Write operation in some fixed amount of time that is independent of the location address.
- Several techniques to increase the effective size and speed of the memory: Cache memory (to increase the effective speed) &Virtual memory (to increase the effective size)

5.1 Connection of a memory to a processor

- The processor reads data from the memory by loading the address of the location into the MAR register and setting the R/W line to 1.
- This transfer takes place over the processor bus.
- The processor bus has,
    - Address Line
    - Data Line
    - Control Line



- If MAR is k bits long and MDR is n bits long, then the memory may contain upto2K addressable locations and the n-bits of data are transferred between the memory and processor.
- 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.
- The memory responds by placing the data from the addressed location onto the data lines and confirms this action by asserting MFC signal.
- Upon receipt of MFC signal, the processor loads the data onto the data lines into MDR register.
- The processor writes the data into the memory location by loading the address of this location into MAR and loading the data into MDR sets the R/W line to 0.
- The control line is used for co-ordinating data transfer.
- The time that elapses between the initiation of an operation and the completion of that operation is called Memory Access Time.
- The minimum time delay that required between the initiations of the two successive memory operations is called Memory Cycle Time.

- The amount of time it takes to transfer a word of data to or from the memory is called Latency.
- For a transfer of single word, the latency provides the complete indication of memory performance. For a block transfer, the latency denotes the time it takes to transfer the first word of data.
- The number of bits or bytes that can be transferred in one second is called Bandwidth.
- Bandwidth mainly depends upon the speed of access to the stored data & on the number of bits that can be accessed in parallel.

5.2 Semi-Conductor RAM Memories

- Semi-Conductor memories are available is a wide range of speeds.
- Their cycle time ranges from 100ns to 10ns

5.2.1 Internal Organization of Memory Chips

- Memory cells are usually organized in the form of array, in which each cell is capable of storing one bit of information.
- Each row of cells constitutes a memory word and all cells of a row are connected to a common line called as word line.

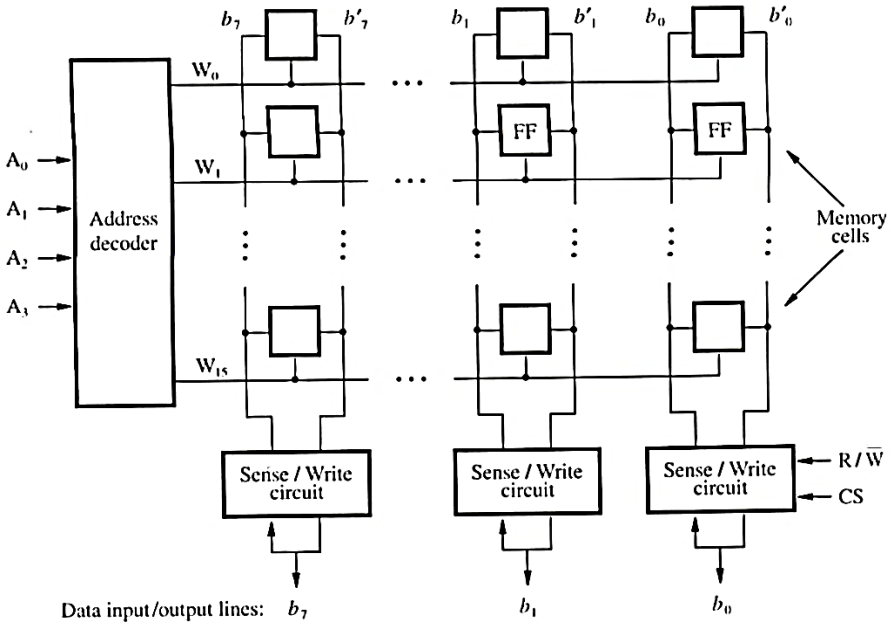
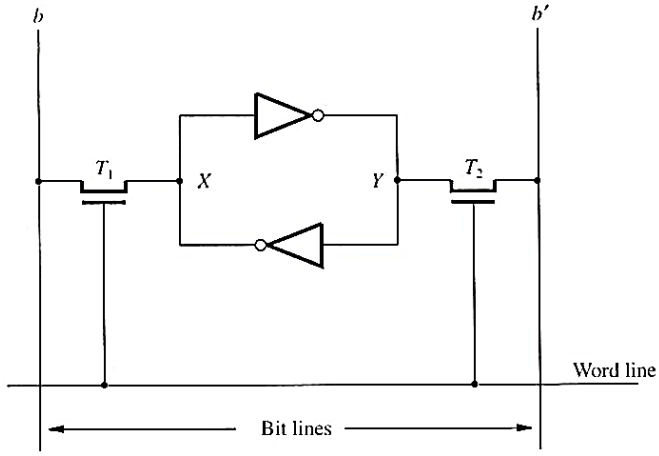


Figure: Organization of bit cells in a memory chip

- The cells in each column are connected to Sense / Write circuit by two bit lines.
- The Sense / Write circuits are connected to data input or output lines of the chip.
- During a write operation, the sense / write circuit receive input information and store it in the cells of the selected word.
- The data input and data output of each senses / write circuit are connected to a single bidirectional data line that can be connected to a data bus of the computer.
- R / W Specifies the required operation.
- CS (Chip Select) input selects a given chip in the multi-chip memory system

5.2.2 Static RAM

- Memories that consist of circuits capable of retaining their state as long as power is applied are known as static memory.
- Two inverters are cross connected to form a latch
- The latch is connected to two bit lines by transistors T1 and T2.



- These transistors act as switches that can be opened / closed under the control of the word line.
- When the wordline is at ground level, the transistors are turned off and the latch retain its state.

Read Operation:

- In order to read the state of the SRAM cell, the word line is activated to close switches T1 and T2.
- 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. Thus b and b' are complement of each other.
- Sense / write circuit at the end of the bit line monitors the state of b and b' and set the output accordingly.

Write Operation:

- 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.
- The required signal on the bit lines are generated by Sense / Write circuit.

Merit:

- It has low power consumption because the current flows in the cell only when the cell is being activated accessed.
- Static RAM‟s can be accessed quickly. It access time is few nanoseconds.

Demerit:

- SRAMs are said to be volatile memories because their contents are lost when the power is interrupted.

5.2.3 Dynamic RAM

- Less expensive RAM‟s can be implemented if simplex calls are used such cells cannot retain their state indefinitely. Hence they are called Dynamic RAM (DRAM).
- The information stored in a dynamic memory cell in the form of a charge on a capacitor and this charge can be maintained only for tens of Milliseconds.

- The contents must be periodically refreshed by restoring by restoring this capacitor charge to its full value.

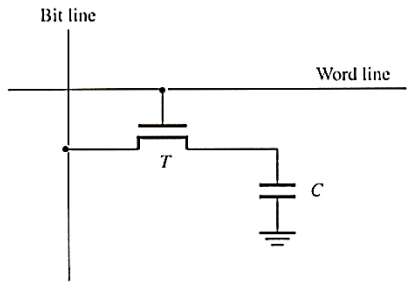


Figure: A single transistor dynamic Memory cell

- In order to store information in the cell, the transistor T is turned "on" & the appropriate voltage is applied to the bit line, which charges the capacitor.
- After the transistor is turned off, the capacitor begins to discharge which is caused by the capacitor's own leakage resistance.
- Hence the information stored in the cell can be retrieved correctly before the threshold value of the capacitor drops down.
- During a read operation, the transistor is turned "on" & a sense amplifier connected to the bit line detects whether the charge on the capacitor is above the threshold value.
- If charge on capacitor > threshold value -> Bit line will have logic value '1'.
- If charge on capacitor < threshold value -> Bit line will set to logic value '0'.

Static vs Dynamic RAM

Static RAM	Dynamic RAM
• Bits stored as on/off switches via flip-flops	• Bits stored as charge in semiconductor capacitors
• No charges to leak	• capacitor charge leaks
• No refreshing needed when powered	• Need refreshing even when powered
• More complex construction	• Simpler construction
• Larger per bit	• Smaller per bit
• More expensive	• Less expensive
• Does not need refresh circuits	• Need refresh circuits (every few milliseconds)
• Faster	• Slower
• Cache	• Main memory

Asynchronous DRAMS

- The processor must take into account the delay in the response of the memory. Such memories are referred to as Asynchronous DRAM's.
- The 4 bit cells in each row are divided into 512 groups of 8.
- 21 bit address is needed to access a byte in the memory (12 bit→To select a row,9 bit→Specify the group of 8 bits in the selected row).

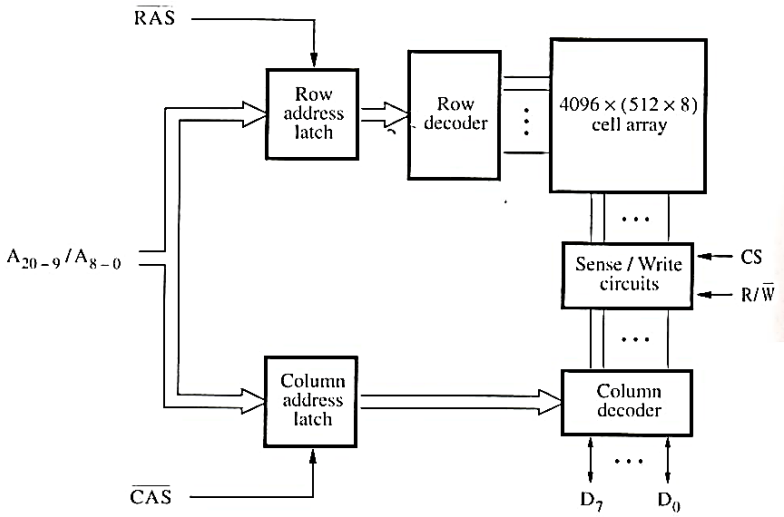
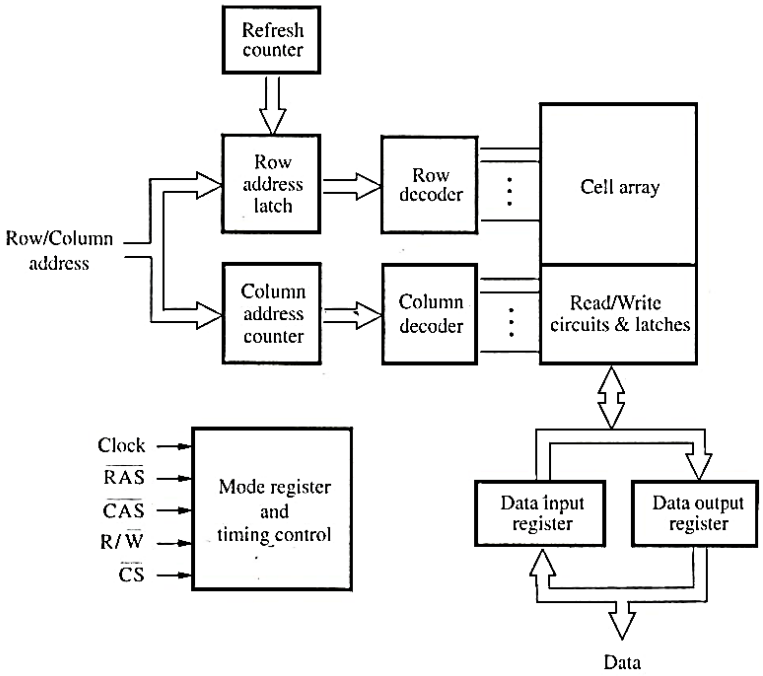


Figure: Internal organization of a 2M X 8 dynamic Memory chip

- A8-0 →Row address of a byte.
  A20-9 →Column address of a byte.
- During Read/ Write operation, the row address is applied first. It is loaded into the row address latch in response to a signal pulse on **Row Address Strobe (RAS)** input of the chip.
- When a Read operation is initiated, all cells on the selected row are read and refreshed.
- Shortly after the row address is loaded,the column address is applied to the address pins & loaded into **Column Address Strobe(CAS).**
- The information in this latch is decoded and the appropriate group of 8 Sense/Write circuits is selected.
- R/W =1(read operation) →The output values of the selected circuits are transferred to the data lines D0 - D7.
- R/W =0(write operation) →The information on D0 - D7 are transferred to the selected circuits.
- RAS and CAS are active low so that they cause the latching of address when they change from high to low. This is because they are indicated by RAS & CAS.
- To ensure that the contents of a DRAM „s are maintained, each row of cells must be accessed periodically.
- Refresh operation usually perform this function automatically.
- A specialized memory controller circuit provides the necessary control signals
- RAS & CAS govern the timing.
- The processor must take into account the delay in the response of the memory.
- Such memories are referred to as Asynchronous DRAM's.

Synchronous DRAM

- Here the operations e directly synchronized with clock signal.
- The address and data connections are buffered by means of registers.
- The output of each sense amplifier is connected to a latch.
- A Read operation causes the contents of all cells in the selected row to be loaded in these latches.



- Data held in the latches that correspond to the selected columns are transferred into the data output register, thus becoming available on the data output pins.

Double Data Rate SDRAM (DDR-SDRAM)

- The standard SDRAM performs all actions on the rising edge of the clock signal.
- The double data rate SDRAM transfer data on both the edges (loading edge, trailing edge).
- The Bandwidth of DDR-SDRAM is doubled for long burst transfer.
- To make it possible to access the data at high rate, the cell array is organized into two banks.
- Each bank can be accessed separately.
- Consecutive words of a given block are stored in different banks.
- Such interleaving of words allows simultaneous access to two words that are transferred on successive edge of the clock.

Rambus DRAM (RDRAM)

- Rambus technology is a fast signaling method used to transfer information between chips.
- Instead of using signals that have voltage levels of either 0 or V_{supply} to represent the logical values, the signals consists of much smaller voltage swings around a reference voltage V_{ref}.
- This type of signaling is generally is known as Differential Signalling.
- Rambus provides a complete specification for the design of communication links(Special
- Interface circuits) called as Rambus Channel.
- The circuitry needed to interface to the Rambus channel is included on the chip. Such chips are known as Rambus DRAM(RDRAM).
- Rambus memory has a clock frequency of 400MHZ.
- The data are transmitted on both the edges of the clock so that the effective data transfer rate is 800MHZ.
- A two channel rambus has 18 data lines which has no separate address lines. It is also called as
- Direct RDRAM's.

5.3 Read Only Memory (ROM)

- Both SRAM and DRAM chips are volatile, which means that they lose the stored information if power is turned off. Many applications require Non-volatile memory (which retains the stored information if power is turned off).
- Eg: Operating System software has to be loaded from disk to memory which requires the program that boots the Operating System. It requires non-volatile memory.
- Non-volatile memory is used in embedded system.
- Since the normal operation involves only reading of stored data, a memory of this type is called ROM.

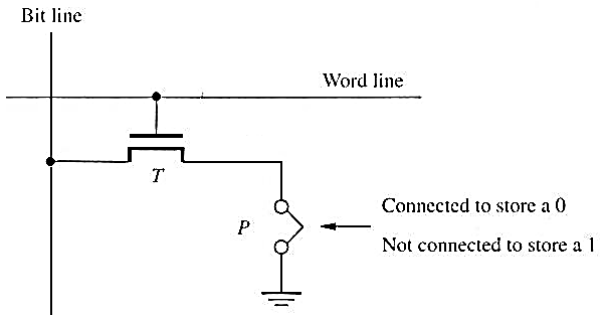


Fig:ROM cell

- 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.
- At Logic value '1' -> Transistor switch is open. The bit line remains at high voltage.
- To read the state of the cell, the word line is activated.
- A Sense circuit at the end of the bitline generates the proper output value.

5.3.1 Types of ROM

- Different types of non-volatile memory are,
  - PROM
  - EPROM
  - EEPROM
  - Flash Memory

Programmable ROM (PROM)

- PROM allows the data to be loaded by the user.
- Programmability is achieved by inserting a "fuse" at point P in a ROM cell.
- Before it is programmed, the memory contains all 0's
- The user can insert 1's at the required location by burning out the fuse at these locations using high-current pulse.
- This process is irreversible.

Merit:

- It provides flexibility.
- It is faster.
- It is less expensive because they can be programmed directly by the user.

Erasable Programmable ROM (EPROM)

- EPROM allows the stored data to be erased and new data to be loaded.

- In an EPROM cell, a connection to ground is always made at "P" and a special transistor is used, which has the ability to function either as a normal transistor or as a disabled transistor that is always turned "off".
- This transistor can be programmed to behave as a permanently open switch, by injecting charge into it that becomes trapped inside.
- Erasure requires dissipating the charges trapped in the transistor of memory cells.
- This can be done by exposing the chip to ultra-violet light, so that EPROM chips are mounted in packages that have transparent windows.

Merits:

- It provides flexibility during the development phase of digital system.
- It is capable of retaining the stored information for a long time.

Demerits:

- The chip must be physically removed from the circuit for reprogramming and its entire contents are erased by UV light.

Electrically Erasable PROM (EEPROM)

- EEPROM is programmed and erased electrically. It can be erased and reprogrammed about ten thousand times.
- Both erasing and programming take about 4 to 10 ms (millisecond).
- In EEPROM, any location can be selectively erased and programmed. EEPROMs can be erased one byte at a time, rather than erasing the entire chip. Hence, the process of reprogramming is flexible but slow.

Merits:

- It can be both programmed and erased electrically.
- It allows the erasing of all cell contents selectively.

Demerits:

- It requires different voltage for erasing, writing and reading the stored data.

Flash Memory

- In EEPROM, it is possible to read & write the contents of a single cell. In Flash device, it is possible to read the contents of a single cell but it is only possible to write the entire contents of a block.
- Prior to writing, the previous contents of the block are erased.
- Eg. In MP3 player, the flash memory stores the data that represents sound.

Merits:

- Flash drives have greater density which leads to higher capacity & low cost per bit.
- It requires single power supply voltage & consumes less power in their operation.
- Single flash chips cannot provide sufficient storage capacity for embedded system application.
- There are 2 methods for implementing larger memory modules consisting of number of chips.
- They are Flash Cards and Flash Drives.
    - Flash Cards:
        - One way of constructing larger module is to mount flash chips on a small card.
        - Such flash card have standard interface. The card is simply plugged into a conveniently accessible slot.

- Its memory size are of 8,32,64MB.
- Eg: A minute of music can be stored in 1MB of memory. Hence 64MB flash cards can store an hour of music.

- Flash Drives:
  - Larger flash memory module can be developed by replacing the hard disk drive.
  - The flash drives are designed to fully emulate the hard disk.
  - The flash drives are solid state electronic devices that have no movable parts.

  Merits:
  - They have shorter seek and access time which results in faster response.
  - They have low power consumption which makes them attractive for battery driven application.
  - They are insensitive to vibration.

  Demerit:
  - The capacity of flash drive (<1GB) is less than hard disk (>1GB).
  - It leads to higher cost per bit.
  - Flash memory will deteriorate after it has been written a number of times.

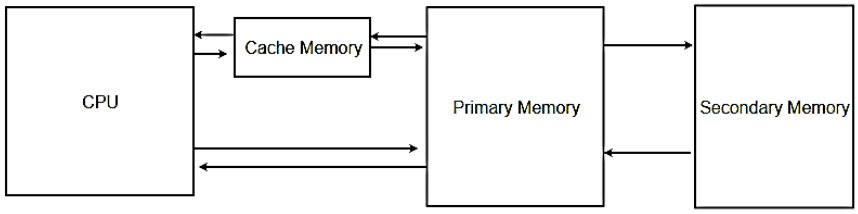
5.4 Cache Memories

- Cache memory is extremely fast memory that is built into a computer's central processing unit (CPU), or located next to it on a separate chip.
- The CPU uses cache memory to store instructions that are repeatedly required to run programs, improving overall system speed.
- Cache built into the CPU itself is referred to as Level l (LI) cache.
- Cache that resides on a separate chip next to the CPU is called Level 2 (L2) cache.
- Some CPUs have both LI and L2 cache built-in and designate the separate cache chip as Level 3 (L3) cache.
- The effectiveness of the cache mechanism is based on a property of called locality of reference.
- Locality of reference can be defined as follows: Many instructions in localized areas of the program are executed repeatedly during some period & remainder of the program is accessed relatively infrequently.

- Two types of locality of reference are there.
  - Temporal
    - Recently executed instruction is likely to be executed again very soon. The temporal aspect of the locality of reference suggests that whenever an information item is first needed, this item should be brought into the cache where it will hopefully remain until it is needed again.
  - Spatial
    - Instructions in close proximity to a recently executed instruction are also likely to be executed soon. Spatial aspects suggest that instead of fetching just one item from the main memory to the cache, it is useful to fetch several items that reside at adjacent addresses as well (That is read a block).

5.4.1 Working of cache Memory

- Cache is placed in between main memory and processor.
- When a read request is received from the processor, the contents of a block of memory words containing the location specified are transferred into the cache one word at a time.
- Subsequently when the program references any of the locations in this block, the desired contents are read directly from the cache.



Read operation

- The following things can happen:
  1. Cache hit: cache block is valid and contains proper address, so read desired word
  2. Cache miss: desired word is not present in cache, so fetch the word from main memory
  3. Cache miss, block replacement: required data not in cache; some other data in the space; fetch desired data from memory and replace

Write operation

- For a write operation system can proceed in two ways:
  1. Write Through: the cache and the main memory locations are updated simultaneously.
  2. Write Back: cache location updated during a write operation is marked with a dirty or modified bit. The main memory location is updated later when the block is to be removed from the cache. The dirty bit is helping to identify the modified block. Write through protocol is simpler, but it results in unnecessary write operations in the main memory when a given cache word is updated several times during its cache residency.

- Write back protocol also results in unnecessary write operations because when a cache block is written back to the memory all words of the block are written back even if only a single word has been changed while the block was in the cache.
- During a write operation, if the addressed word is not in the cache, a write miss occurs. Then, if the write through protocol is used, the information is written directly into the main memory. In the case of write back, the block containing the addressed word is first brought into the cache, and then desired word in the cache is overwritten with the new information.

5.4.2 Cache Memory mapping function

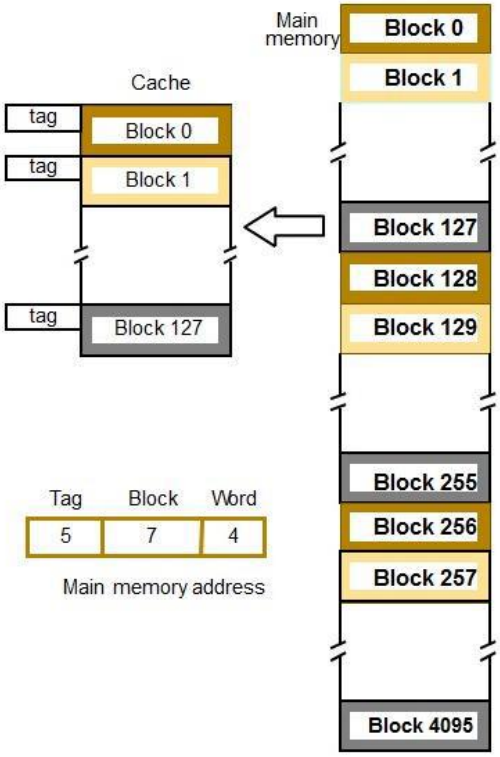
- The mapping functions are used to map a particular block of main memory to a particular block of cache.
- This mapping function is used to transfer the block from main memory to cache memory.
- Mapping functions determine how memory blocks are placed in the cache.
- Three mapping functions:
  1. Direct mapping.
  2. Associative mapping.

3. Set-associative mapping.

- To explain the mapping procedures, we consider a 2K cache consisting of 128 blocks of 16 words each, and a 64K main memory addressable by a 16-bit address, 4096 blocks of 16 words each.

Direct Mapping

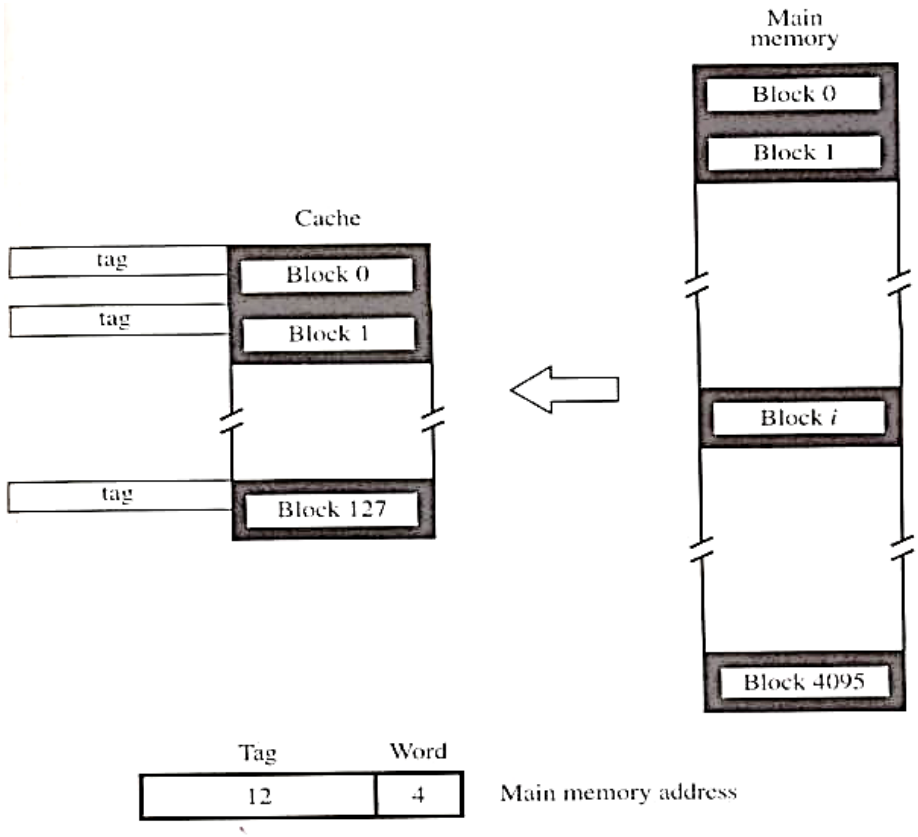
- A particular block of main memory can be brought to a particular block of cache memory. So, it is not flexible.
- The simplest way of associating main memory blocks with cache block is the direct mapping technique. In this technique, block k of main memory maps into block k modulo m of the cache, where m is the total number of blocks in cache.
- According to this mapping function block j of the main memory is mapped to block j modulo 128 of cache. (block 0 is mapped to block 0 modulo 128 of cache that is block 0 is mapped to block 0 of cache)



- Memory address is divided into three fields: Low order 4 bits determine one of the 16 words in a block.
- When a new block is brought into the cache, the next 7 bits determine which cache block this new block is placed in. High order 5 bits determine which of the possible32 blocks is currently present in the cache. These are tag bits.
- No any replacement algorithm is needed, but the existing block is replaced with the incoming block even though the cache is free.
- This mapping methodology is simple to implement but not very flexible.

Associative Mapping

- In this method, main memory block can be placed into any cache block position.
- So, the space in the cache can be used more efficiently.
- In this case, the main memory address is divided into two groups, a low-order bit identifies the location of a word within a block and a high-order bit identifies the block.
- The 12 tag bits identify a memory block residing in the cache and the lower-order 4 bits select one of 16 words in a block.
- The tag bits of an address received from the processor are compared to the tag bits of each block of the cache to see if the desired block is present.
- It gives complete freedom in choosing the cache location.
- A new block that has to be brought into the cache has to replace an existing block if the cache is full.

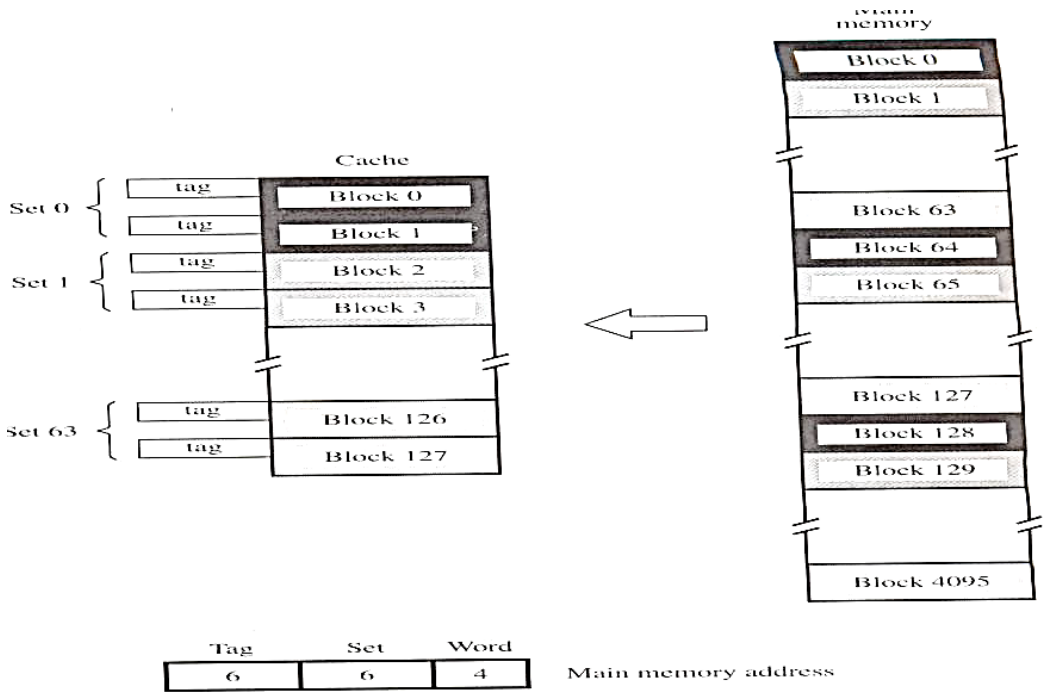


- The cost of an associative cache is higher than the cost of a direct mapped cache because of the need to search all 128 tag patterns to determine whether given cache block is in the cache.
- A search of this kind is called an associative Search.

Set-Associative Mapping

- It is the combination of direct and associative mapping.
- The blocks of the cache are grouped into sets and the mapping allows a block of the main memory to reside in any block of the specified set.
- In this case, the cache has two blocks per set, so the memory blocks 0, 64,128……..4032 maps into cache set "0" and they can occupy either of the two block position within the set.

- 6 bit **set field** determines which set of cache contains the desired block.
- 6 bit **tag field** is the tag field of the address is compared to the tags of the two blocks of the set to clock if the desired block is present.



- The cache which contains 1 block per set is called direct Mapping.
- A cache that has "k" blocks per set is called as "k-way set associative cache".
- Each block contains a control bit called a valid bit.
- The Valid bit indicates that whether the block contains valid data. 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".
- The dirty bit indicates that whether the block has been modified during its cache residency.
- If Processor & DMA uses the same copies of data then it is called as the Cache Coherence Problem.
- The Contention problem of direct mapping is solved by having few choices for block placement.
- The hardware cost is decreased by reducing the size of associative search.

Note :

<u>Replacement Algorithm</u>

- When the cache is full, there is a need for replacement algorithm for replacing the cache block with a new block. For achieving the high-speed such types of the algorithm is implemented in hardware.
- In the cache memory, there are three types of replacement algorithm are used that are:
    1. Random replacement policy.
    2. First in first Out (FIFO) replacement policy

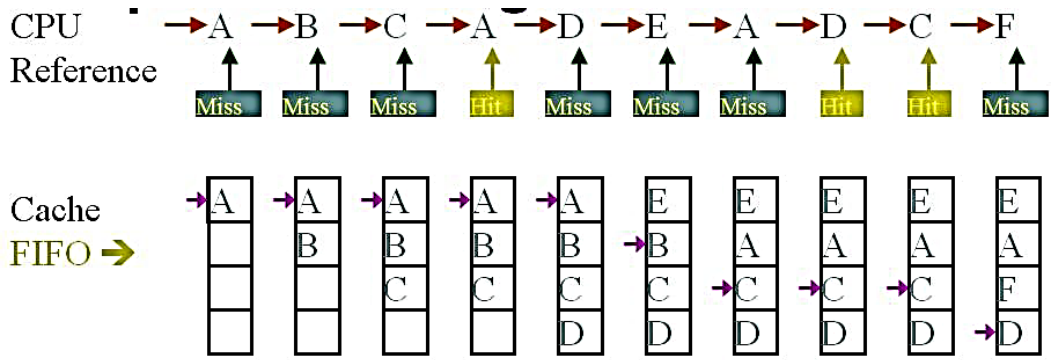
3. Least recently used (LRU) replacement policy.

Random replacement policy

- This is a very simple algorithm which used to choose the block to be overwritten at random.
- In this algorithm replace any cache line by using random selection.
- It is an algorithm which is simple and has been found to be very effective in practice.

First in first out (FIFO)

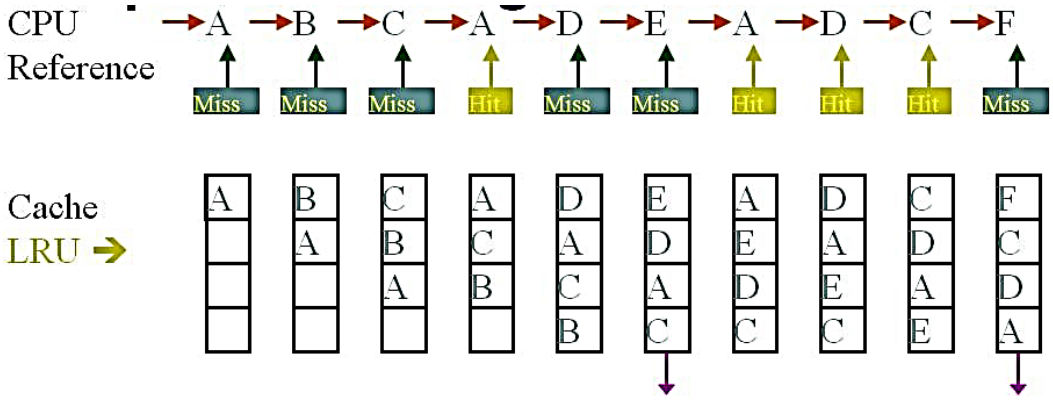
- In this algorithm replace the cache block which is having the longest time stamp.
- While using this technique there is no need of updating when a hit occurs but when there is a miss occur then the block is put into an empty block and the counter values are incremented by one.



Hit Ratio = 3 / 10 = 0.3

Least recently used (LRU)

- In the LRU, replace the cache block which is having the less reference with the longest time stamp.
- In this case also when a hit occurs when the counter value will be set to 0 but when the miss occurs there will be arising of two possibilities in which one possibility is that counter value is set as 0 and in another possibility, the counter value can be incremented as 1.



Hit Ratio = 4 / 10 = 0.4