Medha Gowraya
Asst. Prof, CSE, RNSIT

140+20=160

COMPUTER ORGANIZATION

Course objectives: This course will enable students to

- Understand the basics of computer organization: structure and operation of computers and their peripherals.
- Understand the concepts of programs as sequences or machine instructions.
- Expose different ways of communicating with I/O devices and standard I/O interfaces.
- Describe hierarchical memory systems including cache memories and virtual memory.
- Describe arithmetic and logical operations with integer and floating-point operands.
- Understand basic processing unit and organization of simple processor, concept of pipelining and other large computing systems.

Module -1	Teaching Hours	RBT Levels
Basic Structure of Computers: Basic Operational Concepts, Bus Structures, Performance – Processor Clock, Basic Performance Equation, Clock Rate, Performance Measurement. Machine Instructions and Programs: Memory Location and Addresses, Memory Operations, Instructions and Instruction Sequencing, Addressing Modes, Assembly Language, Basic Input and Output Operations, Stacks and Queues, Subroutines, Additional Instructions, Encoding of Machine Instructions	10Hours	L1, L2
Module -2		
Input/Output Organization: Accessing I/O Devices, Interrupts – Interrupt Hardware, Enabling and Disabling Interrupts, Handling Multiple Devices, Controlling Device Requests, Exceptions, Direct Memory Access, Buses Interface Circuits, Standard I/O Interfaces – PCI Bus, SCSI Bus, USB.	10 Hours	L1, L2
Module - 3		
Memory System: Basic Concepts, Semiconductor RAM Memories, Read Only Memories, Speed, Size, and Cost, Cache Memories – Mapping Functions, Replacement Algorithms, Performance Considerations, Virtual Memories, Secondary Storage.	10 Hours	L1, L2, L3
Module-4		
Arithmetic: Numbers, Arithmetic Operations and Characters, Addition and Subtraction of Signed Numbers, Design of Fast Adders, Multiplication of Positive Numbers, Signed Operand Multiplication, Fast Multiplication, Integer Division, Floating-point Numbers and Operations.	10 Hours	L1, L2, L3, L4
Module-5		

Basic Processing Unit: Some Fundamental Concepts, Execution of a Complete Instruction, Multiple Bus Organization, Hard-wired Control, Micro programmed Control. Pipelining, Embedded Systems and Large Computer Systems: Basic Concepts of pipelining, Examples of Embedded Systems, Processor chips for embedded applications, Simple Microcontroller, Forms of parallel processing, Array Processors, The structure of General-Purpose Multiprocessors.	10 Hours	L1, L2, L4, L6

Course outcomes:

After studying this course, students will be able to:

Acquire knowledge of

- The basic structure of computers & machine instructions and programs, Addressing Modes, Assembly Language, Stacks, Queues and Subroutines.
- Input/output Organization such as accessing I/O Devices, Interrupts.
- Memory system basic Concepts, Semiconductor RAM Memories, Static memories, Asynchronous DRAMS, Read Only Memories, Cache Memories and Virtual Memories.
- Some Fundamental Concepts of Basic Processing Unit, Execution of a Complete Instruction, Multiple Bus Organization, Hardwired Control and Micro programmed Control.
- Pipelining, embedded and large computing system architecture.

Analyse and design arithmetic and logical units.

Apply the knowledge gained in the design of Computer.

Design and evaluate performance of memory systems

Understand the importance of life-long learning

Graduate Attributes (as per NBA)

1. Engineering Knowledge
2. Problem Analysis
3. Life-Long Learning

Question paper pattern:

The question paper will have ten questions.
There will be 2 questions from each module.
Each question will have questions covering all the topics under a module.
The students will have to answer 5 full questions, selecting one full question from each module.

Text Books:

1. Carl Hamacher, ZvonkoVranesic, SafwatZaky: Computer Organization, 5th Edition, Tata McGraw Hill, 2002. (Listed topics only from Chapters 1, 2, 4, 5, 6, 7, 8, 9 and12)

Reference Books:

1. William Stallings: Computer Organization & Architecture, 7th Edition, **PHI, 2006.**

Madhu Gowrayya
Asst. Prof
CSE, RNSIT

III rd Sem
Computer Science.
[Module Scheme]
2016

Computer Architecture

It is a set of rules and methods that describes the functionality and implementation of computer systems. It involves instruction set architecture (ISA) logic design and implementation.

Computer Organization

It refers to operational units and their interconnection that realize architecture specification.

Computer

Computer is a fast electronic calculating machine that accepts digitized input information, processes it according to a list of internally stored instructions, and produces the resulting output information.

→ The list of instruction is called a computer program and the internal storage is called computer memory.

Types of computer

1) Desktop computers

→ These are the computers which are widely used at home, schools, colleges etc. They are also known as Personal computers.

→ It compraises of processing and storage units, video and audio output units, keyboard and mouse as input units, storage includes hard disk, CD-ROM's and diskettes.

→ The most compact form of personal computer is Laptops, where all the components are packaged into single unit.

2) Klorkstations

Klorkstations have high computational power than personal computers.

→ They have high resolution graphics capabilities.

→ The application that requires moderate amount of computing power and high resolution graphics, uses

workstations like animation, simulation, Engg applications like CAD, CAM.
→ Workstations are used for intractive design work.

3) Enterprise systems also known as Mainframes
→ These systems have large computational power and storage capabilities.
→ These are used by large organization that requires bulk amount of data processing like transactions, census and

4) Servers
→ Servers work at back-end. They are capable of holding huge amounts of data. It has large database. Large amount of data is accessed from the server via internet.

5) Supercomputers
used in the field of complex computation. ex weather forecasting, aircraft design, military applications etc. It comes with great speed and memory.

Generations of Computer
Computer mechanical devices were designed using gears, pulleys, levers etc.
→ These were used to perform arithmetic operations.
→ Instructions were fed into punch cards and output were printed on punch card or papers.

First Generation 1945 - 1955 (Vacuum Tubes)
→ Vacuum Tubes were used to perform arithmetic and logical operations. It was 100 to 1000 times faster than electro mechanical systems (gears, pulleys, levers)

→ The concept of stored program was introduced by Jhon Von Neumann.
→ Magnetic drums were used as memory. Program and data were stored in the same memory.
→ Assembly language was used to code the program which was converted to machine language for execution.
→ Magnetic storage devices were also developed.
→ Drawback of this generation was only one problem could be solved at a time (NO multitasking).

Second Generation 1955 to 1965 (Transistors)

→ Beginning of this generation transistors were used which replaced vacuum tubes.
→ Programs were stored in the memory, which moved the generation from magnetic drum to magnetic core technology
→ High level languages like FORTRAN and COBOL were developed.
→ Since transistors were used the size of computers reduced and computers were faster, cheaper, energy effecient and reliable compared to first generation computers.
→ System programs like compilers and assemblers were developed.

Third Generation 1965 to 1975 (Integrated circuits)

→ Integrated circuits were developed.
→ Multiple transistors were fabricated on a single chip called as Integrated circuits.
→ This increased the speed and efficiency of computer.
→ Keyboards and monitors replaced punch cards and printouts.
→ Operating systems were developed which allowed more than one application to run at a time (Multi-tasking)

→ Cache and virtual memories were developed.

Fourth Generation 1975 onwards (Microprocessor)

→ Very Large scale Integration (VLSI) came into existance which led the complete processor being designed on a single chip.

→ Tens of thousands of transistors could be placed on a single chip.

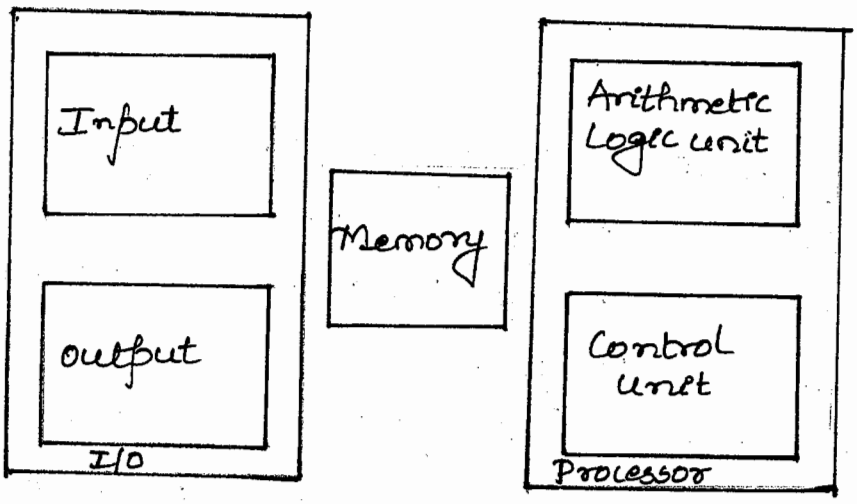
→ High performance computing started with the concepts like concurrency, pipelining, caches & virtual memories.

Beyond Fourth Generation (Artifical Intelligence)

→ Parallel machines, distributed systems are current trends Desktop computers and widespread use of internet is driving the growth of computer industry.

Functional Units of Computer

→ Computer consists of five functional units : input, output memory, arithmetic logic unit and control unit.



Basic functional units of a computer.

Input unit

→ It accepts the digitized information from user by an electromechanical device such as keyboard.

→ 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 memory or processor.

Information handled by the computer are of 2 types

1) Instructions or Machine Instructions (Program)

→ These instructions specify the arithmetic and logical operation.

→ Instructions also govern transfer of information within the functional units of computer i,e i/p, o/p, memory, processor.

2) Data

→ Data are the operands on which operations are performed

→ Numbers and the encoded characters are the operands.

→ Data handled by a computer is encoded as series of 0's and 1's called as bits.

→ Numbers are represented as Binary coded decimal (BCD)

→ Alphanumerics are represented as 7-bit ASCII code or 8-bit EBCDIC (Extended binary coded decimal Interchange code)

Memory unit

→ Before the execution of the program begins it has to be loaded onto memory.

→ Memory stores both program (instructions) and data.

→ There are two types of memory

1) Primary memory

2) Secondary memory

Primary memory (RAM)

→ It is a fast memory where the program and data are stored when they are being executed.

→ It is made up of large number of semiconductor

storage cells, where each cell is capable of storing one bit of information.

→ Memory is identified by distinct address. Since it is inefficient to access one bit at a time every byte has given a address.

→ The fixed no of bits that can be read or written in one read or write operation is known as word. The number of bits tells the wordlength.

→ Wordlength is the maximum number of bits that can be read or written in one operation.

→ The length of the word determines the type of computer. i;e 16-bit computer, 32-bit computer, 64-bit computer.

→ For example in a 16-bit computer in one read operation the maximum no of bytes that can be read 2 bytes (16 bits)

→ For a 32-bit computer in one read cycle maximum no of bytes that can be read is 4 (32-bits)

→ Note:- Size of integer will be always equal to wordlength of the machine.

→ RAM is a memory which can be reached in short amount of time once the address is specified.

→ The time taken to access the memory is known as memory access time which is few nanoseconds to 100ns.

Cache

→ Small fast RAM is known as cache.

→ They are coupled to processor and are part of same processor chip.

→ Access time to cache are much faster than main memory since cache is present inside processor.

Secondary Memory

→ Secondary memory can store huge amount of data that are not used frequently.
ex:- magnetic disk, tapes, CD-ROMs

→ Primary memory is costly so secondary memory can be used to keep data.

→ Secondary memory is slow compared to primary memory

Arithmetic and Logic unit

→ Any arithmetic or logical operations are carried out by ALU
→ operations like addition, subtraction, Multiplication & Division; comparision of numbers etc are performed in ALU
→ If two numbers are to be added which are stored in memory, are first fetched into processor and addition is carried out by ALU. Results are stored in memory or in register for immediate use.
→ when operands are brought into the memory they are stored in high speed storage elements called as Registers.
→ Each register can store one word of data.

output unit

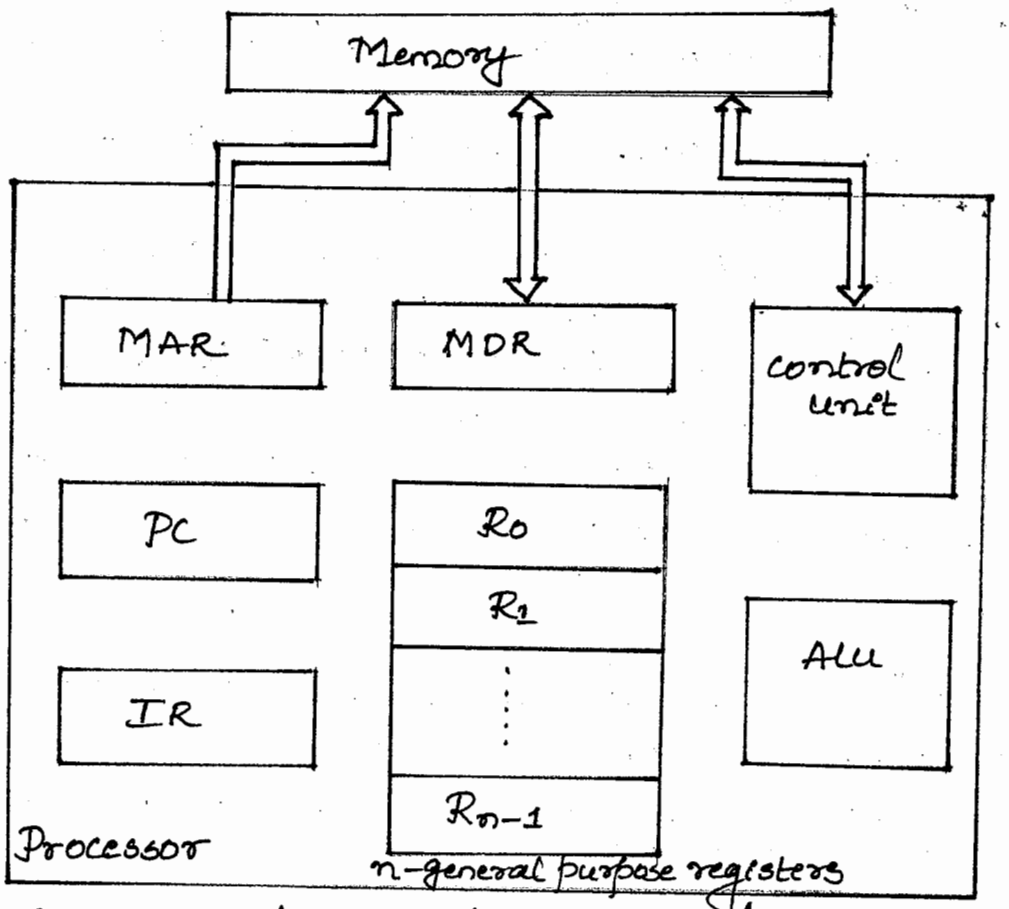
→ output unit sends the processed result to outside world.
→ Ex:- monitors, printers.
→ speed of input and output devices are very less compared to the speed of processor.

control unit

→ Memory, ALU, I/O devices are controlled and co-ordinated by control unit by sending timing signals.
→ Timing signals are the signals that determine when a given action has to take place.
→ Data transfer between processor and memory are also controlled by timing signals.
→ control bus carries the signals used for timing and synchronization of events in all units.

Basic operational concepts

→ To perform a task, program or set of instructions and data (operands) are stored in the main memory.
→ Every instructions as well as data from memory is fetched into processor and the desired operations are carried out.
→ Execution of every instruction happens in four phases.
1) Instruction Fetch
2) Instruction Decode
3) Instruction Execute
4) Write Back



Connection between processor and memory

→ Processor consist of ALU, control unit, General purpose registers (GPR), MAR, MDR, PC, IC

→ GPR (General purpose register)

→ Processor consist of n general purpose registers $R_0 - R_{n-1}$.

→ These registers are used during computation and program execution.

Instruction Register (IR)

→ This register holds the current instruction being executed.

→ The output of IR is available to control unit to generate control information.

Program counter (PC)

→ Program counter will have the address of the next instruction in the memory to be executed.

→ It is used to keep track of the program.

Memory Address Register (MAR)

It hold the address of an instruction (location in memory) or operand which has to be accessed.

Memory Data Register (MDR)

It will have the data which is read from memory location or which has to be written to some memory location.

Instruction Fetch Phase

Note :- Instruction and data are in main memory

→ Execution of program begins when Pc holds the address of first instruction in the memory.
→ Contents of Pc (address) is transferred to MAR.
→ Processor sends the read signal to the address present in MAR.
→ The first instruction is fetched (Read out of memory) into MDR.
→ Contents of MDR is transferred to IR.

Instruction Decode Phase

Note :- In fetch phase only the instruction is fetched into processor.
In decode phase data which is required for the operation is read from memory to processor.
In decode phase the operation to be performed, address of the operands in the memory will be found out so that data can be read for the operation.

→ When instruction comes to IR it is ready to execute.
→ If the data (operands) resides in the memory then it has to read (fetched) by sending the address of the memory to MAR.
→ processor initiates a read signal and data is read into MDR
→ From MDR data comes to ALU.
→ If more than one operand resides in memory the same procedure is repeated.

Instruction Execution

→ once all the operands are read from memory ALU performs the desired operation.

→ Result will be stored in MDR.

Write Back

→ If at all the result has to be stored in memory then address of memory is sent to MAR

→ Write cycle is initiated

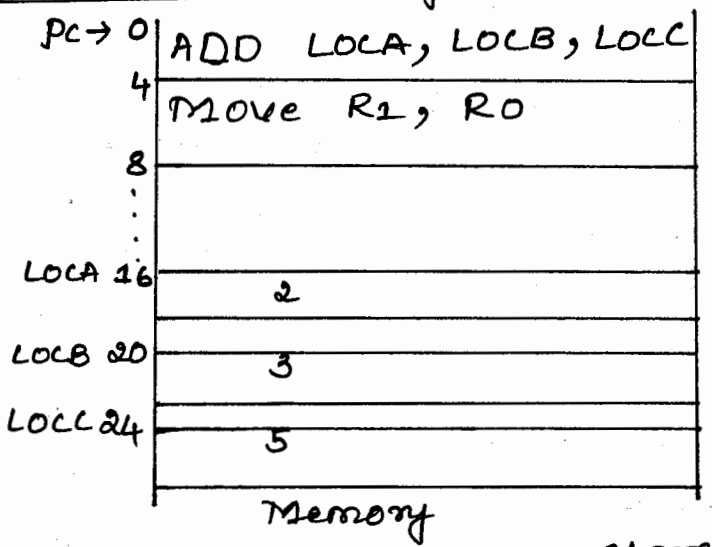
→ contents from MDR are written to the address specified in MAR.

→ During these phases Pc will be incremented to the address of next instruction in memory.

MAR, MDR, PC, IR are the special registers.

→ After the completion of one instruction next instruction will be fetched and executed in the similar manner.

Consider an example (Inst & data in memory)



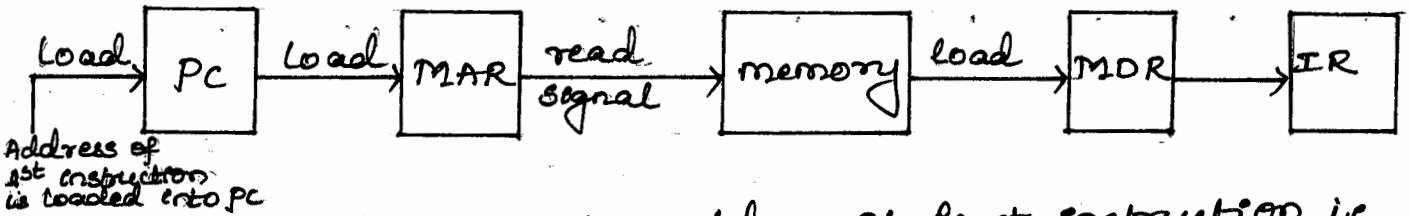
→ Two instructions are stored in memory. Even data are stored in memory.

→ First instruction adds the contents of LOCA with LOCB and stores in LOCC

→ Second instruction copies the content of register R1 to R0

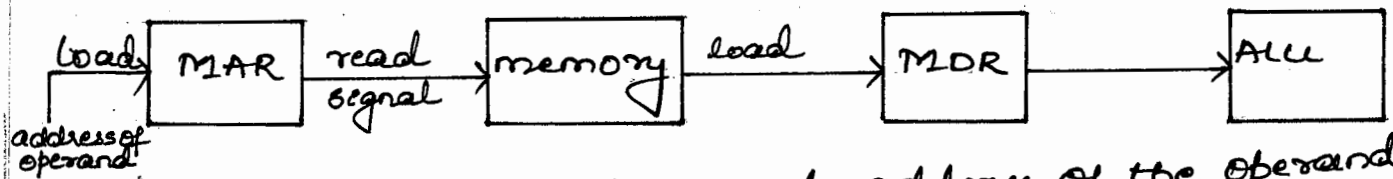
The four different phase of these instructions are as follows

Instruction Fetch



→ Pc will be pointing to address of first instruction i,e 0 in this example.
→ This address is loaded into MAR. MAR has the address 0
→ A read signal is initiated to the memorylocation.
→ Instruction is fetched into MDR
→ From MDR instruction i,e ADD LOCA, LOCB, LOCC moves to IR.

Instruction Decode



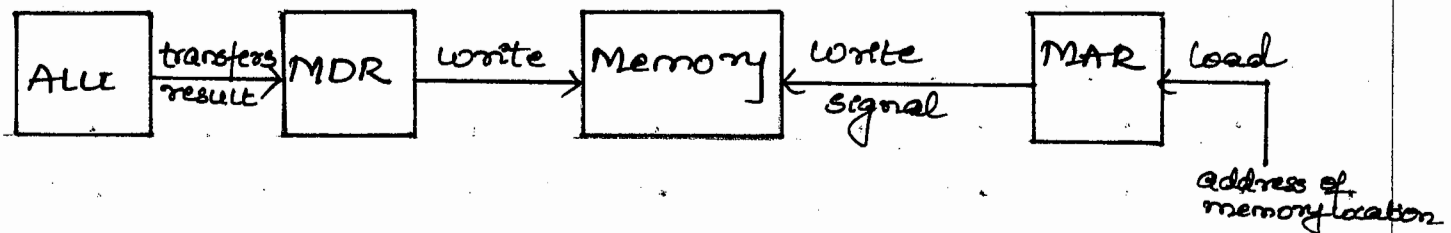
→ Identifies the operation and address of the operand in the memory.
→ address of the first operand i,e LOCA (i,e (16)) is sent to MAR and read signal is initiated.
→ data is read from memory to MDR.
→ from MDR data is transferred to ALU.
→ Similarly data from LOCB (20) is also read into ALU
Instruction E

Instruction Execution

→ once all the operands are in ALU, it performs the operation
→ Result is stored in MDR

Write Back

→ Address of LOCC is sent to MAR i,e (24)
→ Processor sends a write signal to the address* present in MAR
→ contents (data) from MDR is written to the address in MAR.



→ Meanwhile PC is incremented to next location i,e 4 in this example.

→ Next instruction MOVE R1, R0 is fetched and executed in the similar manner.

Bus Structure

→ The different operational units of computer i,e Input output, memory and processor are connected together so that they can communicate.

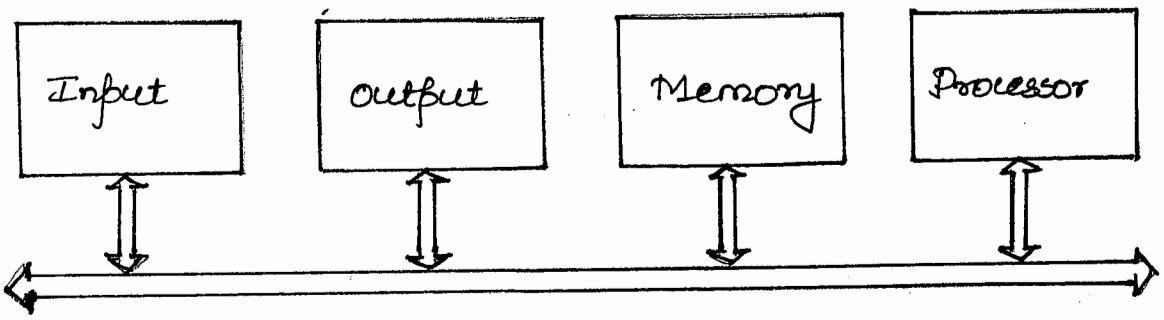
→ All the operational units are connected by parallel wires or lines called as bus.

→ Each line or wire can carry one bit of information.

→ Depending on the architecture bus can have 16, 32, 64 lines.

→ Buses can carry data, address and signals. There are three types of bus
  → Data bus
  → Address bus
  → Control bus

Single bus Architecture



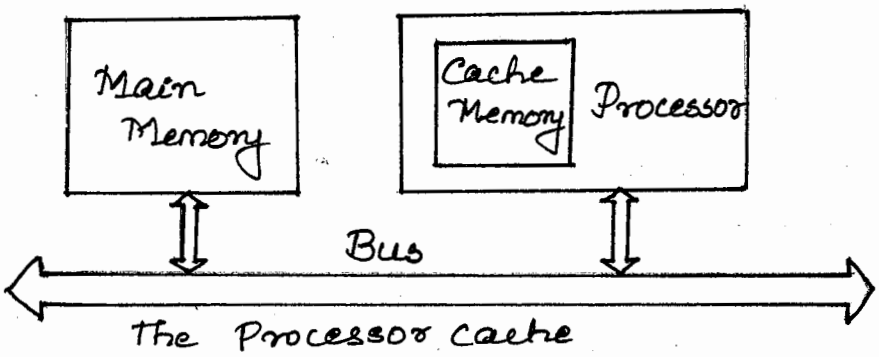
→ In a single bus architecture all the devices are connected to a single bus.
→ Since all the devices are connected to the same bus only two devices can communicate with each other at a time.
→ Advantage of single bus is low cost and flexibility of attaching all the peripheral devices to the bus.
→ Devices attached to the bus operate at their own speed.
→ Input and output devices (keyboard, printer) are electromechnical devices and are very slow.
→ Memory and Processor unit operates at electronic speed.
→ To coordinate the activities of all these devices on the same bus in a faster manner, effecient mechanism should be used.
→ For example consider a scenario where Processor and Printer are communicating with each other. Hence no other device can communicate.
→ Task is to print some characters on the printer so processor is sending the characters one after the other to printer.
→ Since printer is a slow device it recieves the character and prints the character taking its own time.
→ Meanwhile processor has to wait until printer complete its task of printer.
→ That means processor is blocked and no othe device can communicate until printing task is completed.
→ To synchronize the timing issue and processor being blocked buffer registers are used for the slow devices.
→ Register are faster than the electromechanical devices
→ Processor sends all the character one after the other to the buffer register.
→ when all the characters are transferred to buffer printer starts printing the characters from buffer taking its own time.
→ So the processor is free and it can perform some other task.

Performance Issue

Performance of a computer is measured by how quickly it executes the program.

→ speed of a computer is affected by design of its hardware and its machine language instructions.

→ One method of increasing the speed of the computer is to use cache memory.



The Processor Cache

→ When execution of program is above to begin, program is loaded into main memory.

→ At a time one instruction is fetched from main memory to processor via bus, and one copy of that instruction is stored in cache

→ Even when data are read into processor one copy is placed in cache.

→ So if at all the same instruction or data is required then it is fetched from cache instead of main memory.

→ Access time to cache is much faster than main memory since cache is placed inside the processor.

→ Hence cache speeds up the program execution.

Processor clock

→ Processor has its own timing circuits called as clock

→ Processor defines a regular time interval called as clock cycle.

→ Length of the clock cycle is denoted by P

→ Inverse of clock cycle is clockrate(R)i.e $R = 1/P$

→ clock rate is measured in cycles per second

→ SI unit for cycles per second is Hertz

Example

1) If the length of the clock cycle (P) = 2ns then calculate the clock rate

$R = 1/P$

$R = 1/2 \times 10^{-9}$

$R = 0.5 \times 10^{9}$

$R = 500\ MHz$ i,e R is 500 million cycles per second

2) If the length of the clock cycle (P) is 0.8ns then calculate the clock Rate.

$R = 1/P$

$R = 1/0.8 \times 10^{-9}$

$R = 1.25 \times 10^{9}$

$R = 1.25\ GHz$ i,e 1.25 billion cycles per second.

The speed of the processor.

Basic Performance Equation

→ Let T = Processor time required to execute the complete program written in high level language

N = Actual number of machine instruction executed to complete the execution of program

S = Average number of basic steps required to execute one machine instruction, and every basic step is completed in one clock cycle.

R = Clock Rate (cycles per second)

Program execution time is given as

$$T = \frac{N * S}{R}$$

Note :- Effeciency will be more if N and S are less and R is more.

→ i,e N will be less if program can be converted to fewer machine instruction.

→ S can be less if instruction can be executed in fewer Basic steps.

→ If a high frequency clock is used R will be high.

Clock Rate (R)

Can be increased in two ways

→ Improve the Integrated Technology to make logic circuits faster and hence time needed to execute basic step is reduced.

→ Reduce the amount of processing done in one basic step

Performance Measurement

→ Performance is measured using benchmark programs.

→ Benchmark programs are some standard programs.

→ Benchmark program have been developed for wide range of applications like Gaming, compilers, Database Application, astrophysics, quantum chemistry and so on.

→ Benchmark programs were developed by an non-profitable organization called SPEC (System Performance Evaluation corporation)

→ These Benchmark programs are first run on reference computer (ex:- SPEC95 uses SUN SPARC station 10/40)

→ SPEC rating is computed as

$$\text{SPEC rating} = \frac{\text{Running time on reference computer}}{\text{Running time on computer under test}}$$

→ So if a ratio of 50 is got then it means test device is 50 times faster than reference device.

→ Test is carried out for all the programs (ex:-gaming database etc) and final rating is given as

$$\text{SPEC rating} = \left(\prod_{i=1}^{n} \text{SPEC}i \right)^{1/n}$$

where n is the number of programs run.

Problems

1) List the steps needed to execute the machine instruction "Add LOCA, R0" in terms of transfers between the components shown in the figure (connection between processor and memory) and some simple control commands. Assume that the instruction itself is stored in memory at location INSTR and this address is initially in register PC.

Solution:-

→ Transfer the contents of register PC to MAR
→ Issue a read command to memory.
→ Instruction is fetched into MDR
→ Transfer the instruction from MDR to IR and decode it.
→ Transfer the address of LOCA to MDR
→ Read command is sent to memory
→ Read the data from memory LOCA to MDR
→ Transfer the contents of MDR to ALU.
→ Transfer contents of R0 to ALU
→ ALU performs the desired operation.
→ Result is transferred from ALU to R0.
→ PC is incremented to INSTR +1.

2) Repeat Problem 1 for the machine instruction "Add R_1, R_2, R_3"

Solution:-

→ First four steps are same as problem 1.
→ Transfer the contents of R_1 and R_2 to ALU.
→ Perform addition and store the result in R_3.
→ PC is incremented to INSTR +1.

3) How the performance of a computer is measured? Assuming that the reference computer is ultra SPARC10 workstation with 300MHz ultra SPARC -IIi processor. A company has to purchase 500 new computers, hence ordered testing of a new computer with SPEC2000(run on reference as well as new computer). Following observation were made

Programs	Runtime on reference computer	Runtime on new computer
1	50 mins	5 mins
2	75 mins	4 mins
3	60 mins	6 mins
4	30 mins	3 mins

A company's system manager will place the orders for purchasing new computers only of the overall SPEC rating is atleast 12. After the said test, will the system manager place order for the purchase of new computers?

Solution:-

$$\text{SPEC rating} = \left(\prod_{i=1}^{4} \text{SPEC}_i\right)^{1/4}$$

i,e geometric mean

SPECi = Runtime of reference computer / Runtime of computer under test.

Hence, $\text{SPEC}_1 = 50/5 = 10$

$\text{SPEC}_2 = 75/4 = 18.75$

$\text{SPEC}_3 = 60/6 = 10$

$\text{SPEC}_4 = 30/3 = 10$

Overall SPEC rating $= [10 * 18.75 * 10 * 10]$

$= 11.70$

Since the rating is 11.70 which is less than the benchmark rating 12, purchase orders will not be placed.

Memory Locations and Addresses

→ Instructions and data (numbers, characters) are stored in a memory.

→ Memory consist of several storage cells, where each cell is capable of holding (storing) 1 bit of information. That bit can be 0 or 1.

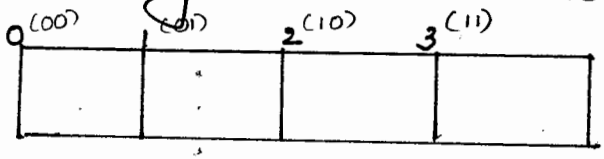
→ Memory is accessed by its address. Giving address to every bit is unnecessary and tedious because we hardly deal with individual bits.

→ So every byte has given an address. Smallest memory that can be accessed is byte.

→ Since every byte has been given an address it is known as byte-addressable memory.

If there are four memory location then two bits are sufficient to give the address.

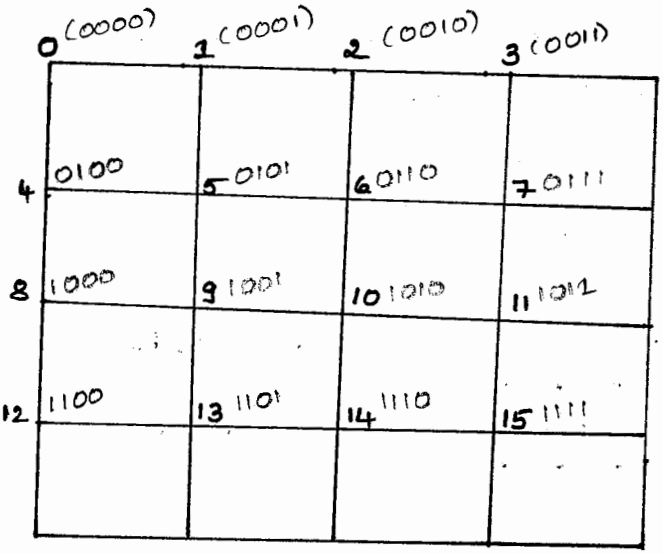
0 → 00
1 → 01
2 → 10
3 → 11



using 2 bit 4 locations can be given an address.

If there are 16 memory location then four bits are enough to give the address.

0 → 0000
1 → 0001
2 → 0010
3 → 0011
4 → 0100
5 → 0101
6 → 0110
7 → 0111
8 → 1000
9 → 1001
10 → 1010
11 → 1011
12 → 1100
13 → 1101
14 → 1110
15 → 1111

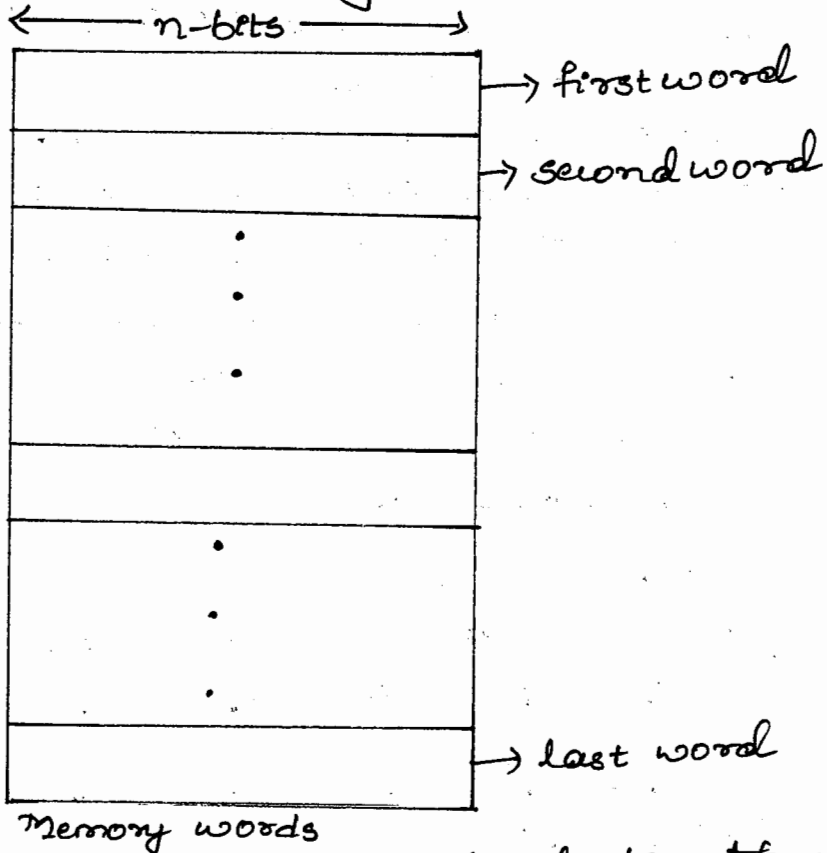


which means that using 4 bit 16 locations can be given an address. The fourmal is

0 to $2^n - 1$ = 0 to $2^4 - 1$ = 0 to 15 (16 location)

→ using 10 bits 1024 locations can be addressed.
→ using 32 bits 4G locations can be addressed.

Note:- One location is one byte.

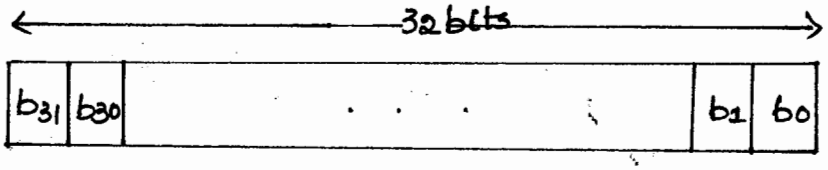


Consider, if we are using 16 bit computer then length of one word is 2 bytes. So in one read cycle max of 16 bits i,e 2 bytes can be read.

If a 32 bit computer is used then length of one word is 4 bytes. So in one read cycle max of 32 bits i,e 4 bytes can be read.

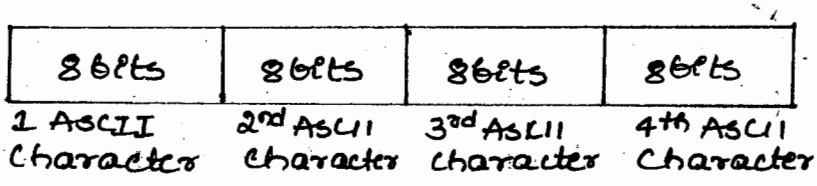
Assuming machine is 32-bit

For integer



For character

4 characters can be stored in one word



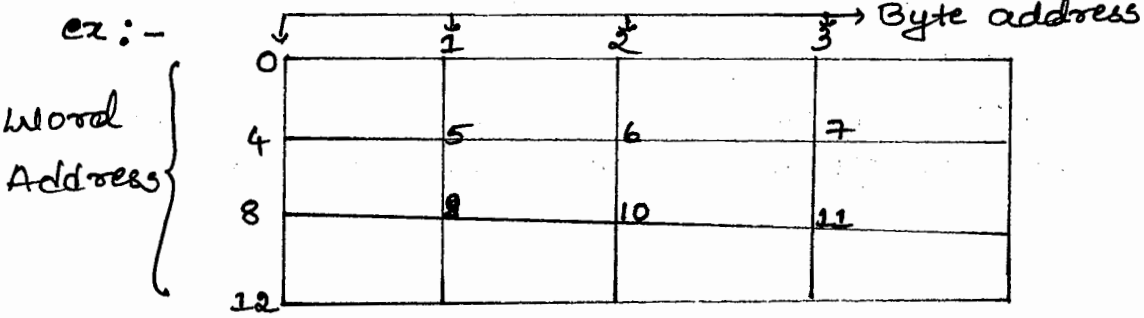
Byte Addressability

There are 3 terms we have come across till now

1) Bit
2) Byte → 8 bits make a byte
3) word → multiple bytes make a word (ex:- 32 bit word has 4 bytes & 16 bit word has 2)

→ Memory can be addressed as bit level, byte level & word level.
→ Bit level address is seldom done.
→ Most practical way of assigning address is byte levels which is known as Byte Addressable Memory.
→ Byte locations have address 0, 1, 2, 3, 4, ...
→ word is a set of multiple bytes, in a 32-bit machine word address will be 0, 4, 8, 12, 16, ...

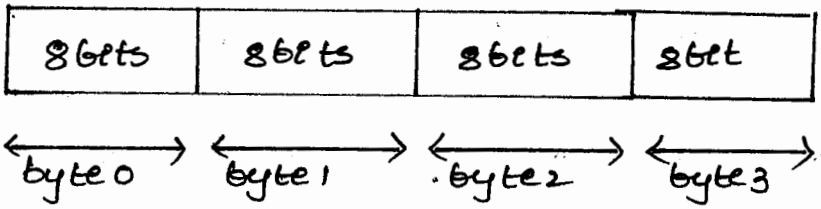
ex:-



Big- Endian AND Little-Endian Assignment

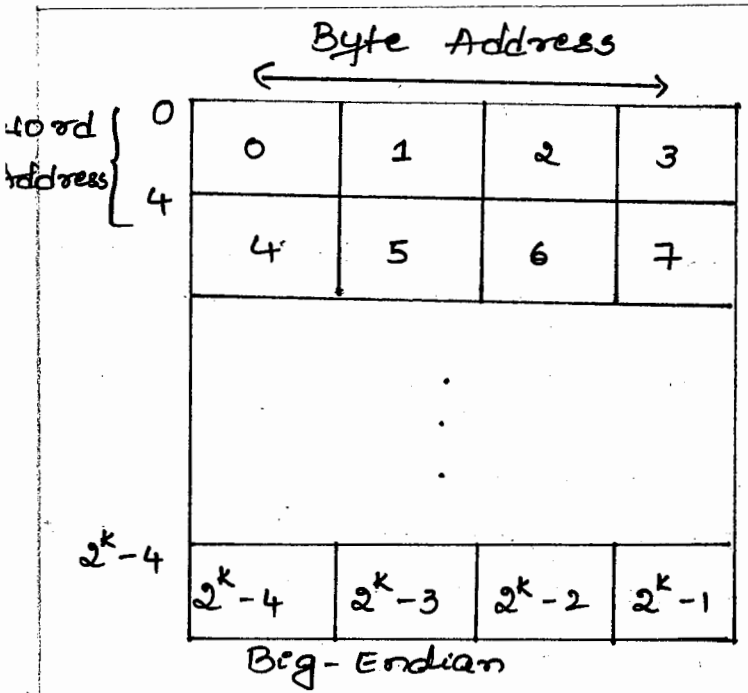
→ These assignment tells in which order data is stored in the memory. i)e Big Endian or Little Endian
→ Computer follows Big Endian or Little Endian is an Architectural issue.

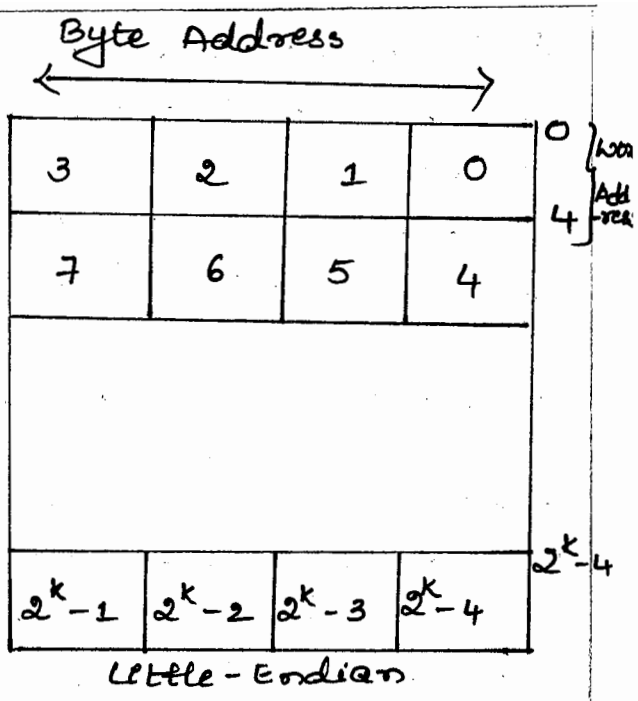
Conside a 32-bit word



Big - Endian Assignment

→ In this system the lower byte of the memory stores the higher order byte (MSB). Or
→ Big Endian is used when lower byte address are used for the Most significant bytes of the word.





Little-Endian Assignment

→ In this system, lower byte of memory stores the lower order byte. Or

→ In Little-Endian assignment lower byte address are used for less significant bytes of the word.

Problem.

1) Consider a 32-bit integer : ABC0EF12 in hex. Represent this number in Big-Endian and Little-Endian

Solution :- Assume one word is 32-bit so 4 bytes.

One byte = 8 bits

Every digit is treated as hex. i.e ~~word~~ stored in binary coded decimal.

So every digit requires 4 bits. Since one byte is 8 bits, two digits can be stored in one byte (location)

Big-Endian

	0	1	2	3
0	AB	CD	EF	12
4				

Little Endian

	0	1	2	3
0	12	EF	CD	AB
4				

2) Represent the number 90AB12CD14 in Big-Endian & Little Endian Assignments

Big - Endian

0	1	2	3
90	AB	12	CD
14			

Little - Endian

0	1	2	3
CD	12	AB	90
			14

3) Represent the number 81234561 in Big-Endian and Little-Endian memory organization.

Big - Endian

0	1	2	3
81	23	45	61

Little - Endian

0	1	2	3
61	45	23	81

4) Consider a computer that has a byte-addressable memory organized in 32-bit words according to the big-Endian and little-Endian scheme. A program reads ASCII characters entered at a keyboard and stores them in successive byte locations, starting at location 1000. Show the contents of the two memory words at location 1000 and 1004 after the name "Johnson" has been entered.

Big - Endian

1000	1001	1002	1003
J	o	h	n
s	o	n	

Little - Endian

1000	1001	1002	1003
n	h	o	J
	n	o	s

Write the hexadecimal values for each

1000	1001	1002	1003
4A	6F	68	6E
73	6F	6E	

1000	1001	1002	1003
6E	68	6F	4A
	6E	6F	73

Word Alignment

If a word is 32-bit long and if word boundaries occur at address 0, 4, 8, 12, 16, ... then we say that words have aligned address. ex:- word1 is 0–3 bytes, word2 is 4–7 bytes and so on.

Unaligned address

If a word (32 bit word) does not align at word boundaries 0, 4, 8 .... then such words are called unaligned words.
ex:- word1 is 2–5 bytes, word2 is 6–9 bytes.

Memory Operations

Memory contains instructions as well as data operands
For example

```
int main()
{
    int a=5, b=10, c;
    c = a+b;
}
```

→ The instruction c=a+b is stored in the memory.
→ The variables store some value in the memory.
→ Even though we have written the program in high level language during compilation it gets converted to assembly level language.
→ Then this assembly level language is converted to machine level language.
→ This instruction c=a+b will be converted into assembly language using three-address instruction, two-address, one address or zero-address instruction
→ which type of instruction is used it depends on the architecture.

To manage the instructions and data, two basic operation involving memory are needed.

1) Load (Fetch) (Read)

2) Store (write)

1) **Load**

Load operation transfers a copy of instructions or data from memory to the processor. The contents of memory will not chang when load operation is performed.

2) **Store**

Store operation transfers the data(result) from processor to specific memory location. The previous contents of memory is overwritten.

Note:- An information that is transferred between a memory and processor could be either a word or a byte.

Syntax for Load is :-	Load memory, Registor
Syntax for Store is :-	Store Register, memory

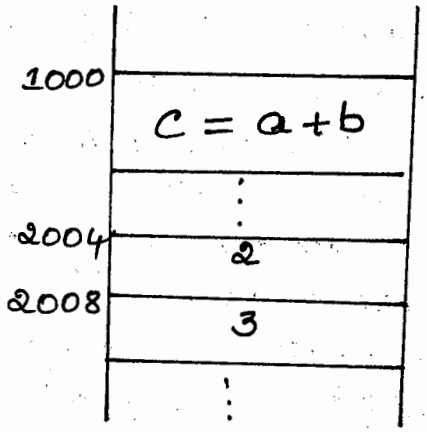
Problems

1) Give a short sequence of machine instructions for the task.

Add the contents of memory location A to those of location B, and place the result in location C. Instruction of the type Load Loc, Ri and Store Ri, Loc are the only instructions available to transfer data between memory and GPR Ri. Do not destroy the contents of location A and B

Solution :-

```
Load  A, R0
Load  B, R1
Add   R1, R0
Store R0, C
```



2) Suppose Move and Add instructions are available with the format Move Location1, Location2 and Add Location1, Location2. These instructions add or move a copy of the operand at first location to the second location, overwriting the original operand at the second location.

Location1 and Location2 can be memory or processor register. Is it possible to give fewer instruction compared to previous problem? If yes then give the sequence of machine instructions.

Solution

Yes.

Move B, C

Add A, C

Instruction and Instruction Sequencing

A computer performs four types of operation

→ Data or instruction transfer between memory and processor

→ Arithmetic and logical operations.

→ Program sequencing and control

→ Input output transfers.

To understand this some notations are to be defined

1) Register Transfer Notation

→ Data can be transferred between memory location, processor register and special registers in memory register of Input output i.e buffer registers.

→ Memory locations can be identified by a symbolic name like LOC, VAR, PLACE, A and so on

→ Processor registers are identified as R0, R1 ... Rn-1.

→ I/O registers (buffer registers) are identified as DATAIN, OUTSTATUS etc.

→ In Register transfer notation the contents of memory or identified by placing them in square braces.

ex:-

1) $R_1 \leftarrow [LOC]$ The contents of memory Location LOC is copied to R_1.

2) $R_3 \leftarrow [R_1] + [R_2]$ The contents of register R_1 is added with contents of register R_2 and result is stored in R_3.

2) Assembly language Notation

→ To represent machine instruction another notation is assembly language

→ A high level language is first converted to assembly language

RTN	Assembly language
$R_2 \leftarrow [LOC]$	Move Loc, R_1
$R_3 \leftarrow [R_1] + [R_2]$	Add R_1, R_2, R_3

Basic Instruction Types

1) Three address instruction
2) Two address instruction
3) One address instruction
4) Zero address instruction

Three address instruction.

→ Consider a C statement $C = A + B$, where A and B are added and stored in C.

→ RTN can be represented as $C \leftarrow [A] + [B]$

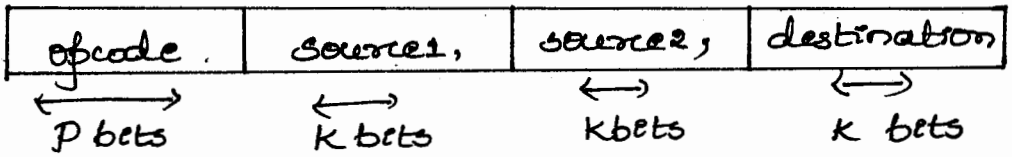
→ Assembly language code for the above RTN is Add A, B, C

→ This is known as three address instruction where Add is the opcode, A, B & C are the operands.

→ Syntax can be written as

opcode source1, source2, destination

→ Suppose p bits are required to represent opcode in binary, and k bits for source1, source2 and destination then this requires $p + 3k$ bits.



→ 3 address instruction will be large to store in one word.

→ In that case multiple word is used to store the instruction

Two-address instruction

→ An instruction that works with two operands is called as two address instruction.

Syntax :- | opcode source, destination. |

ex :- Add A , B

C = A+B can be represented in two address instruction as follows

	RTN
Move B, C	C ← [B]
Add A, C	C ← [A] + [C]

One Address Instruction

→ In one address instruction only one operand is specified

→ In case of one address instruction other operand is the processor register called as Accumulator.

Syntax :- | opcode Source/destination |

→ Operand is source or destination depends on the type of operation (opcode)

→ Load A :- Contents of memory location A is fetched into accmulator. A is the source and accumulator is the destination.

→ Store B :- Contents of accmulator are written to memory location B. B is the destination. Accmulator is the source.

To perform c=a+b using one address instruction.

```
Load A    // contents of A are fetched into accmulator
Add B     // contents of B are added to accmulator and
          stored in accmulator
Store C   // contents from accmulator are written to C
```

write an assembly level code to evaluate the expression A*B + C*D in a single accmulator processor. Assume that the processor has load, store, Multiply and Add instructions, and all the values fits onto accmulator.

Solu:-

```
Load A
Multiply B
store Result
```

Load C
Multiply D
Add Result
Store Result

→ In some processors operations can be performed only on Registers. In such processor the content from memory is first moved to register. C = A+B can be written as

	RTN for the same instruction
Move A, R0	R0 ← [A]
Move B, R1	R1 ← [B]
Add R0, R1	R1 ← [R0] + [R1]
Move R1, C	C ← [R1]

→ In some processor if one operand can be memory location then the previous instruction can be written as

	RTN for the same instruction
Move A, R0	R0 ← [A]
Add B, R1	R1 ← [B] + [R1]
Move R1, C	C ← [R1]

Note:- without spoiling the original contents of memory location.

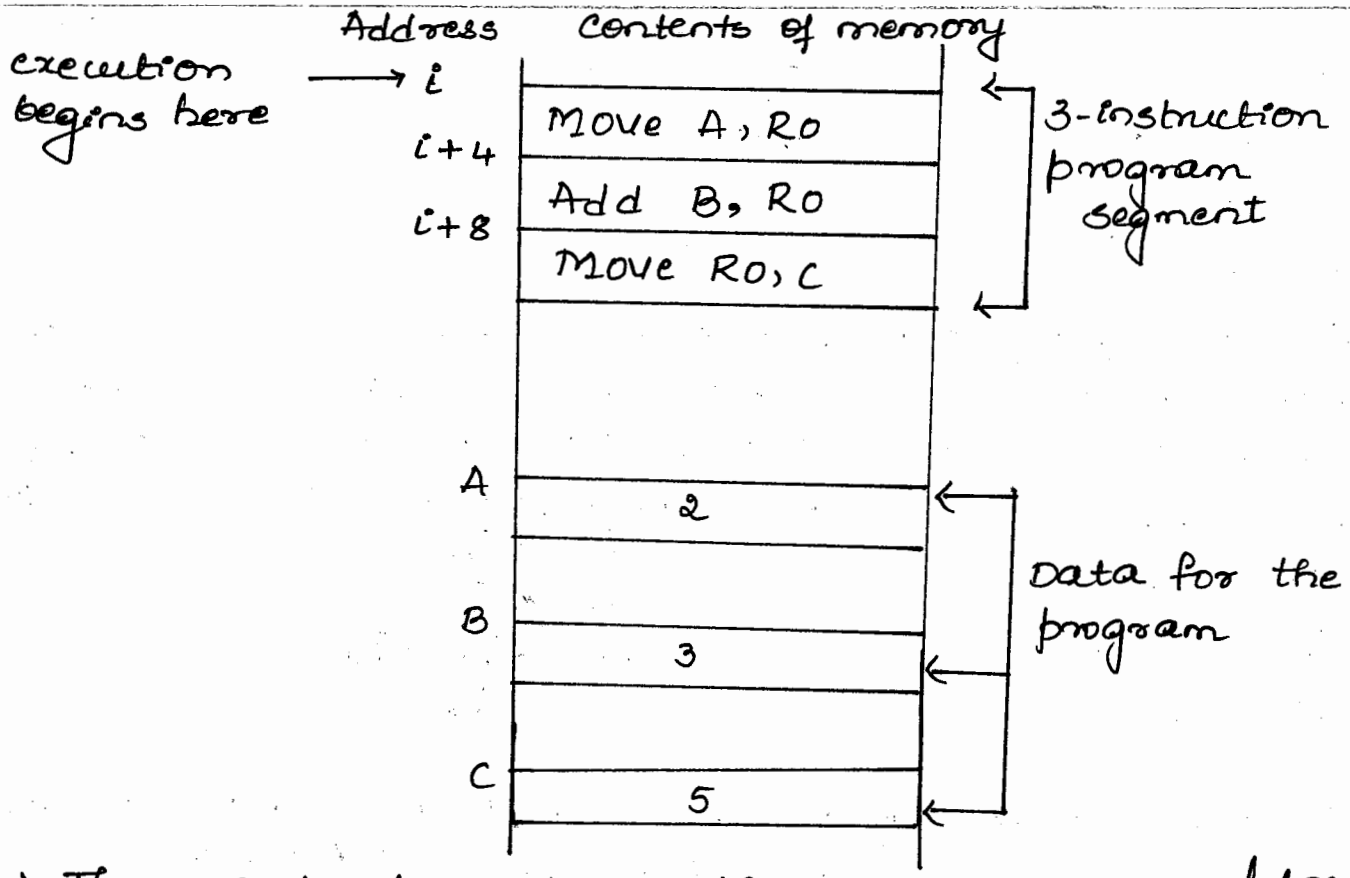
Instruction Execution and straight line sequencing

→ Here will see how an Instruction Execution takes place
→ And meaning of straight line sequencing.

Consider an example C ← [A] + [B]. Two address instruction for this instruction can be written as

Move A, R0
Add B, R0
Move R0, C

Before these instructions are executed they are stored in memory.



→ These instructions takes three contiguous word location called as program segment.

→ Even the data are stored in the memory, so three memory location named A, B and C are reserved.

→ we assume that each instruction takes 4 bytes of memory that is 32 bits. So if the first instruction is stored in the address i, then next instruction will be in the address i+4, i+8 and so on.

→ Execution of the instruction happens in two phases.

1) Instruction Fetch

→ Program counter points to the address of first instruction

→ contents of PC is sent to MAR and instruction is fetched into MDR.

→ From MDR instruction is transferred to IR

2) Instruction Decode

→ In this phase, operation to be performed is identified.

→ The required operands are fetched for example the value from memory location A and B.

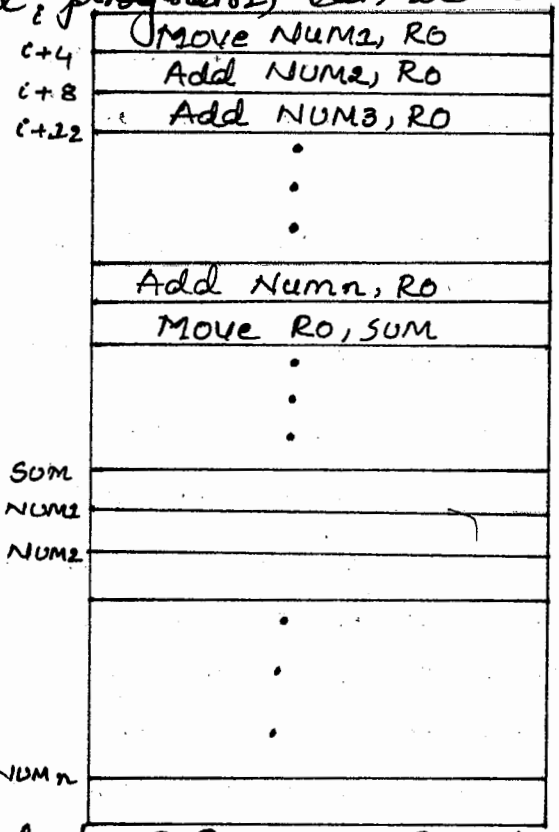
→ when these instructions are executed PC will move to the address of instruction i,e i+4.

→ Every instructions gets executed one after the other in the

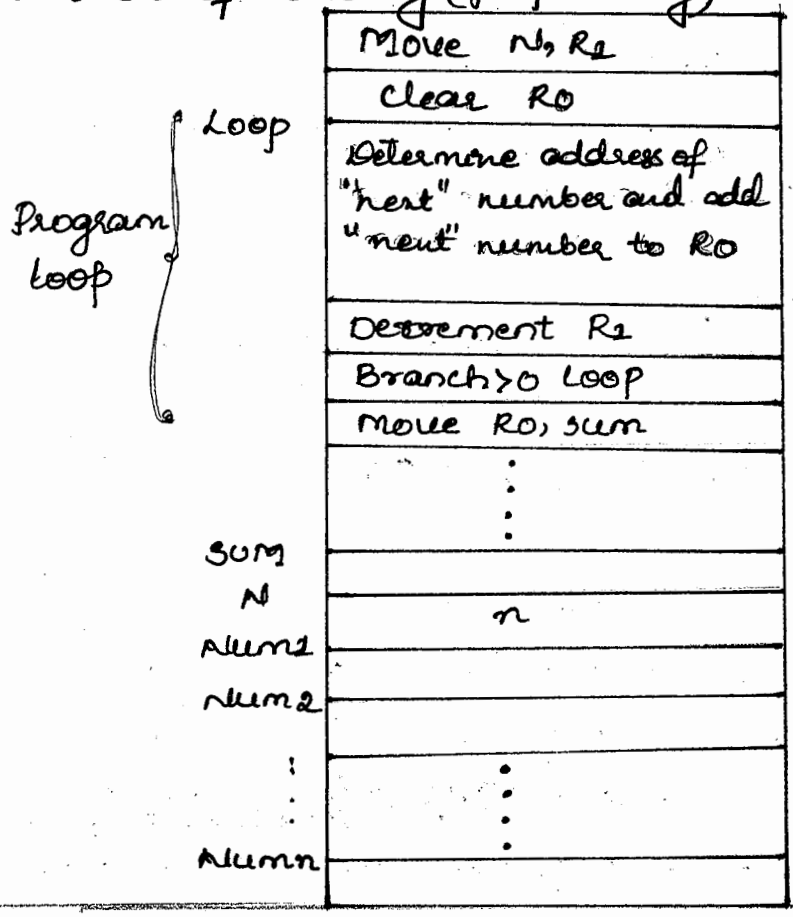
increasing order of their word address which is known as straight line sequencing.

Branching

Consider a task of adding 'n' numbers Num1 to Numn and store the result on a variable SUM. Straight line program (sequential program) can be written as follows.



Instead of sequential program we can use branching, which reduces the task of writing (performing) same instruction repeatedly



→ In the above program segment we used branching to perform the task of adding n numbers.

Steps are as follows.

→ Variable N stores the number of times SUM operation needs to be performed. N is moved to R2. So R2 is the counter.

→ R0 is cleared to zero because it stores the intermediate result as it keeps getting added. Final result is stored in SUM.

→ Looping block is shown in the brace '{'. In the looping block next 'Num' is fetched from the memory using addressing modes and added to R0.

→ Register R2 is decrement by 1.

→ Condition should be checked. A special instruction called 'Branch >0 Loop' is used. This means that if the result of the instruction above this i,e (Decrement R2) is greater than zero then go back or jump to the branch target address called as Loop. If condition is false then execute the next instruction i,e Move R0,SUM

→ If condition is true PC is loaded with branch target address, if condition is false PC is loaded with the address of next instruction.

→ When condition becomes false next instruction executes, so R0 is moved to SUM.

Conditional Codes

→ Processor has a special register called as status register or conditional codes register.

→ This register is a set of flags i,e every bit represents one flag.

→ Four out of these flags are

1)	N (Negative)	set to 1 if result is -ve, else cleared to 0
2)	Z (zero)	set to 1 if result is 0, else cleared to 0
3)	V (overflow)	set to 1 if overflow occurs, else cleared to 0 for arithmetic operation
4)	C (Carry)	set to 1 if carry out results, else cleared to 0

→ Consider an example which includes branch instruction.
→ Processor wants to keep track of the results of these instruction based on which it can decide to continue with the loop or with the next instruction.
→ Whenever the result of Branch instruction or Arithmetic operations is false (overflow & carry not Arithmetic operation) the status register flags are set.
→ By seeing the flags of status register processor decides condition is true or false.

1) Negative flag

This flag will be set if the result of arithmetic operations or logical operation results in negative value.

Example :- Consider the memory named Loc has a value 0

```
        Move  Loc, R1
Loop    Decrement  R1
        Branch >0  Loop
        <next instruction>
```

→ -ve flag will be set since R1 becomes -ve when decremented. Zero flag will be set. Hence PC will be loaded with next instruction address.

2) Zero flag.

This flag will be set if the result of arithmetic or Branch instruction results in zero.

example :- Consider the memory named Loc has a value 2

```
        Move  Loc, R0
Loop    Decrement  R0
        Branch >0  Loop
        <next instruction>
```

→ Second time when R0 gets decremented it becomes 0. So Zero flag will be set. There wont be jump instead PC is loaded with address of next instruction.

3) Overflow flag :- An arithmetic overflow occurs when the MSB of two numbers are same and MSB of answer is different.

ex:- 0 1 0 1 (+5)

0 0 1 1 (+3)
1 1 1

1 0 0 0

→ MSB of 5 and 3 are 0 whereas the MSB of answer is 1. this is an overflow.

4) Carry flag

If there is no space for last bit then it is known as carry. so if at all carry occurs then carry flag of status flag will be set.

example :- 1001 (-7)

1111 (-1)

11000

Problem :- write a program that can evaluate the expression A x B + C x D in a single accumulator processor. Assume that the processor has Load, store, Multiply and Add instructions, and that all values fit in the accumulator.

Solution

```
Load A
Multiply B
Store RESULT
Load C
MULTIPLY D
Add RESULT
Store RESULT
```

Machine Instructions and Programs

Addressing Modes.

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

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

Note :- Address of the operands are specified in the instruction itself which is known as addressing modes.

→ Addressing modes comes into existance when during Instruction decode phase.

→ In that phase address of the operand is found out to fetch the data.

1) Register Addressing mode

→ In this mode the operands are processor register itself.

→ The name of the register (address) is given in the instruction.

Ex:- Move R1, R2

→ Effective address is address of register itself, where the data can be found.

2) Absolute mode (Direct mode)

→ In this mode operand is the memory location.

→ The address of the memory location is explicitly specified in the instruction. (name of the memory specifies address)

ex:- Move LOC, R1

Or

Add A, R1

3) Immediate Addressing mode

In this mode value of the memory (operand) is explicitly given in the instruction.

Immediate mode is used to specify the value of source operand and represent by # symbol or uppercase I.

Example

Move #500, R0 can also be written as

MOVEI 500, R0 .

In both the cases value 500 is moved to the register R0.

Consider a literal constant in C language i,e C=A+2. In assembly language using Immediate addressing mode it can be written as follows.

Move A, R0

Add #2, R0

Move R0, C

4) Indirect addressing mode (Indirection and Pointers)

→ In this mode address of the operand or the value is not given in the instruction.

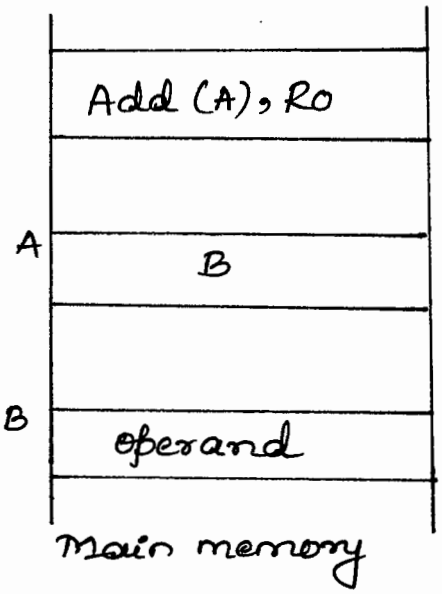
→ Instead it provides an information from which the memory address of an operand can be determined. This is known as Effective address (EA).

→ Effective address of an operand is the contents of a register or memory location whose address appears in the instruction.

→ Effective address is specified by placing a register or memory inside paranthese.

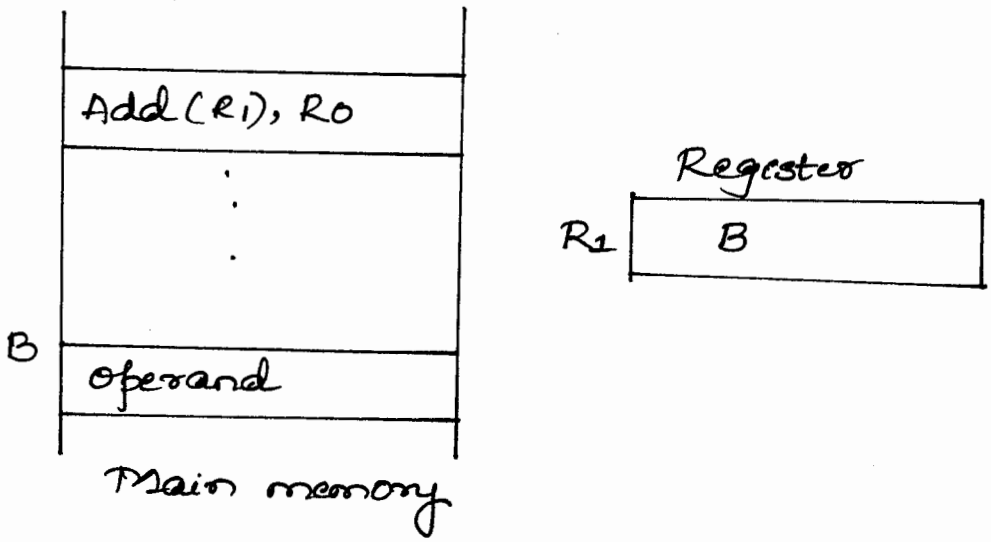
Example 1

→ A is the memory location which is having the address of memory location B.
→ In the memory location B, operand is present.
→ which means that A is having the effective address and hence it is a pointer.
→ Processor first reads the contents of memory location A.
→ A is having the address of memory location B.
→ It then requests a second read operation to memory location B where the data resides.
→ Register or the memory that has an address is known as pointer.



Example 2

→ Register R1 is having the effective address
→ Processor sends a read operation and reads the content of register R1. R1 has the address of memory location B.
→ Processor sends the second read operation to memory loc B where data resides.



Consider an example

→ A is the memory location, which is having the address of memory location B.

→ In the memory location B, operand is present.

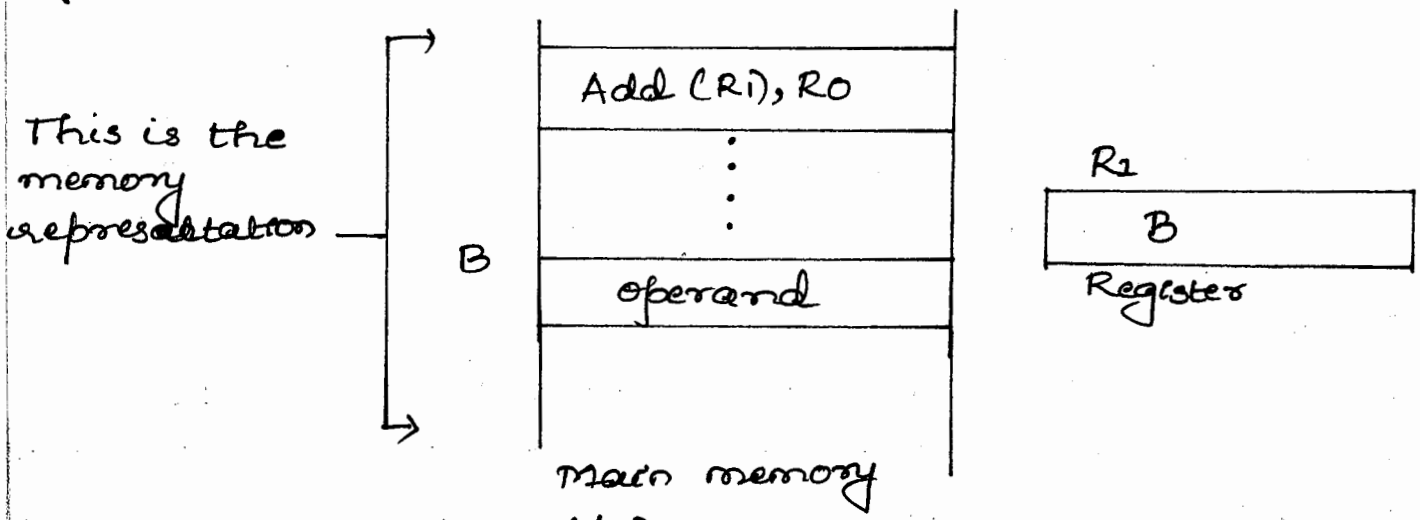
→ which means that A is having the Effective address and hence it is a pointer.

→ The processor first reads the contents of memory location A.

→ A is having the address of B.

→ It then requests a second read operation to memory location B where the data resides.

→ Register or the memory that has an address is known as pointer.



Consider another example2.

→ Register R1 is having the effective address.

→ Processor sends a read operation and reads the contents of memory ~~location~~ Register R1 which has the address of memory location B.

→ Processor sends the second read operation to memory B where the data resides.

→ This is how operands are fetched using Indirect mode.

Consider a program which performs addition of n numbers using indirect addressing mode. (using pointer)

```
        Move    N, R1
        Move    #NUM1, R2
        Clear   R0
Loop    Add     (R2), R0
        Add     #4, R2
        Decrement R1
        Branch>0  Loop
        Move    R0, SUM
```

→ Register R2 is used as a pointer to the numbers on the list or array, and the operands are accessed indirectly through R2.

Steps

1) Value of N is copied to R1. R1 is counter which tells how many times sum operation should be performed.
2) Immediate addressing mode is used to place the address of memory location NUM1 into R2. R2 will have the address.
3) R0 is used to hold the intermediate sum. So R0 is cleared to zero.
4) R2 contains the address of NUM1, the data in memory location NUM1 is fetched and added to R0.
5) Considering the wordlength of machine is 32bits(4 bytes) a constant value 4 is added to R2. R2 points to the next word address i.e NUM2 on this example.
6) R1 is decremented
7) A special branch instruction is executed to check whether the condition is true or false. condition is true or false depends on the result of previous instruction (i.e decrement R1).

→ If the result of previous instruction is -ve or zero then the respective flags will be set in status Register based on which processor decides whether the condition is true or false

→ If any flags are set then condition is false else it is true

when counter becomes zero condition is false so PC is loaded with the address of next instruction.

→ R0 is moved to the memory location sum.

5) Index Mode

The effective address of the operand is generated by adding a constant value to the contents of a register.

→ The register used may be general purpose register or special register.

→ Register used for index mode is known as index Register.

→ It is represented as X(Ri), where x is the constant and Ri is the register.

→ Effective address is calculated by adding constant X with register Ri.

→ X is known as offset or displacement.

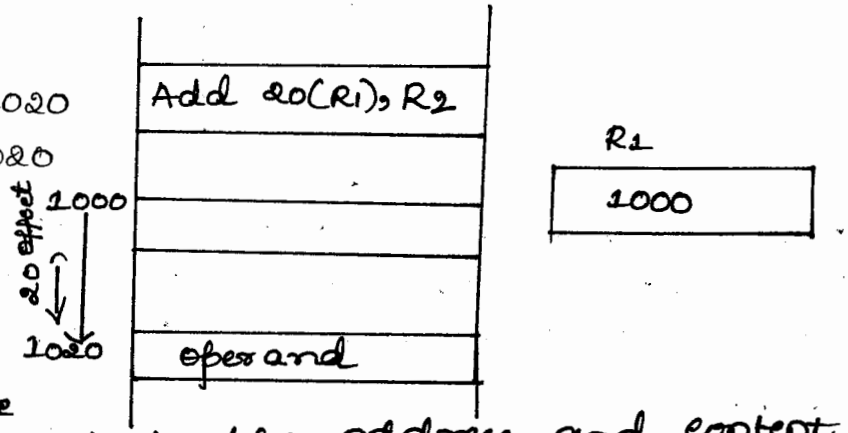
Consider an example

→ The index register R1 contains the address of the memory location. Effective address is found out by adding the constant 20 to R1. (R2 contains address 1000)

→ i.e 20 + 1000. = 1020 is the address where the operand is present.

Steps involved

1) 20 is added to 1000 to get 1020
2) data stored in location 1020 is picked.
3) that value is added to register R2.



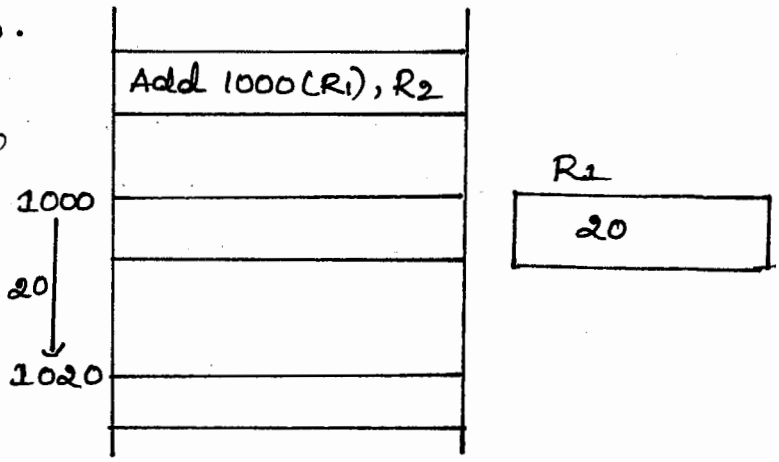
Consider another example

→ Here constant X corresponds to the address and content of register gives the offset.

→ 1000 + 20 = 1020 is the EA.

Steps

1) 1000 is added to 20 to get 1020
2) data is picked from location 1020
3) that value is added to register R2.



Example 3

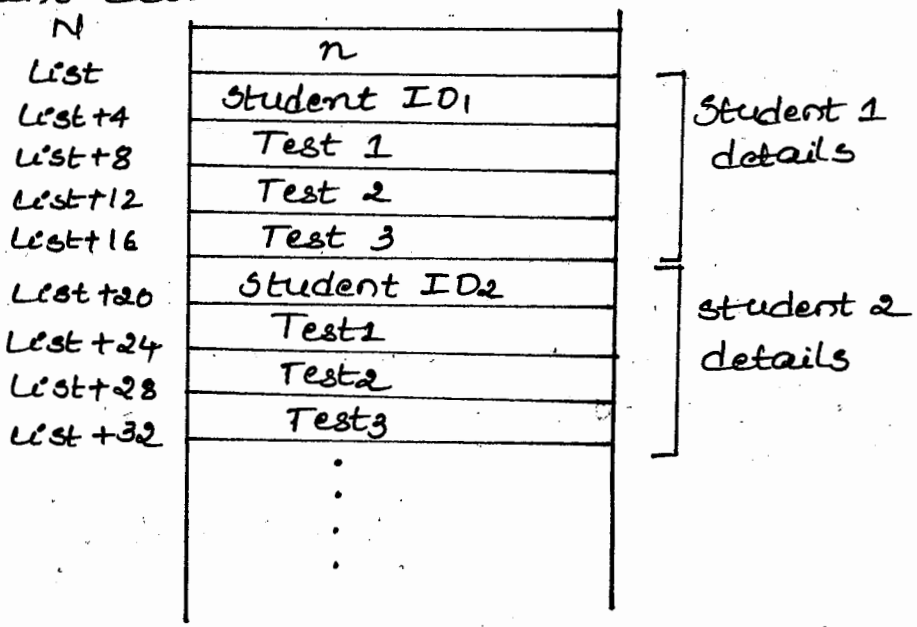
Consider a student database with following fields.
Student ID, test1 marks, test2 marks, and test3 marks.

Student ID	TEST 1	TEST 2	TEST 3

Calculate the total of test1, test2 and test3 score of n students separately.

Considering there are n students, every student details requires 4 words of memory. 4×4 bytes = 16 bytes.

→ First student details takes the memory LIST to LIST+15, second student detail takes LIST+16 to LIST +31 & so on.



consider LIST as the base address of the student as LIST, then test1 details are obtained by 4 to LIST, test2 is obtained by adding 8 to LIST and 3rd test detail by adding 12 to LIST.

TO jump from 1st student to 2nd student detail we need to add 16 to base address.

Consider the program below

```
Move    #LIST, R0
Move    N, R4
Clear   R1
Clear   R2
Clear   R3
```

```
Loop    Add     4(R0), R1
        Add     8(R0), R2
        Add     12(R0), R3
        Add     #16, R0
        Decrement   R4
        Branch>0    Loop
        Move    R1, sum1
        Move    R2, sum2
        Move    R3, sum3
```

Steps involved

1) Move #LIST, R0. This instruction makes the base address of the student array i,e LIST to be loaded into R0. R0 is a pointer since it has the address.
2) Clear R1, R2, R3 makes R1, R2 and R3 to 0. R1 is used to store test1 marks of all students, R2 is used to store test2 marks of all students and R3 is used to store test3 marks of all students.
3) Move N, R4 makes the no of students to be stored on R4
4) Add 4(R0), R1 :- To the base address (0) add 4 to read the test1 score of 1st student, then add it to R1 and store in R2.
5) Add 8(R0), R2 :- R0 = 0 x+(R0) i,e 8(0) = 8. In the address 8 test2 score of 1st student is present which is fetched added with R2 and stored in R2.
6) Add 12(R0), R2 :- R0 = 0 12+0 = 12. In the address 12 test3 score of 1st student is present which is fetched and added with R3 and stored in R3.
7) Add #16, R0 :- R0 was pointing to 1st students base address when 16 is added to it it jumps to 2nd student's base address.
8) Decrement R4
9) If the condition is true then 2nd students detail will be added in the same manner.
10) Else R1 to copied to sum2, R2 to sum2 and R3 to sum3
11) R1 will have test1 score of all students
12) R2 will have test2 scord of all students similary R3

will have test3 score of all students.

Another form of index mode.

Tell now index mode was used in the following manner

Add 5(R0), R1

→ Constant can also be stored in register.

For example

1) → Add (R0, R2), R1

→ To get the EA, R0 & R2 are added then the data is fetched from that address and added to R1.

2) Add 20(R0, R5), R1

→ In this EA is got by adding R0 with R5 and constant 20. i,e 20+R0+R5 = EA.

6) Relative Addressing mode.

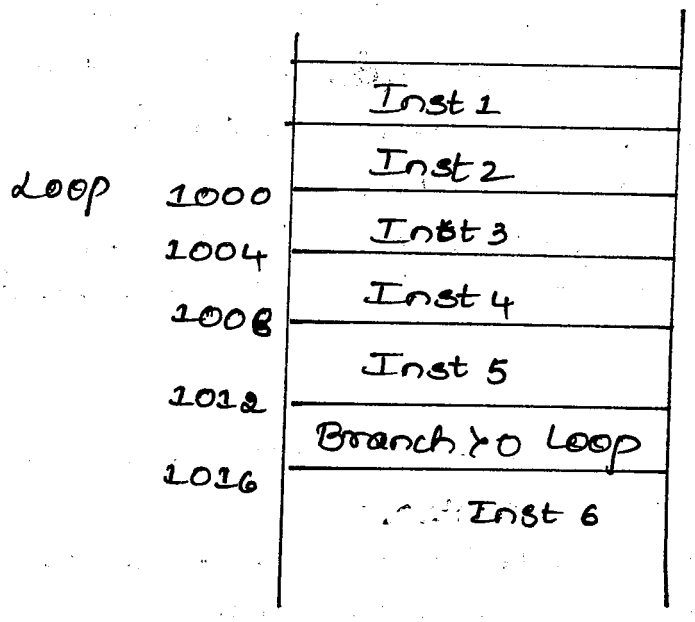
→ In relative addressing mode Program counter is used as the index register.

→ symbolically represented as X(PC).

→ Relative mode is defined as the address of the memory location that is X bytes away from the location (address) pointed by PC.

→ Hence the address is known to be relative to PC.

For example consider an instruction takes one word of memory. Assume the address as 1000, 1004, 1008 and so on



→ when branch instruction executes the contents of PC is 1016.
→ processor executes the branch instruction and checks the condition is true or false by checking status register flags.
→ If condition is true then PC should be loaded with the branch target address. so it calculates the address using relative mode.
→ First Offset is determined i;e $X = (1016 - 100) = 16$
→ [PC] - x will give the address of branch target.

1016 - 16 = 1000.

→ PC is loaded with 1000, and ~~only~~ once again looping Stmts gets executed.

Note;- when Branch instruction is used assembler calculates the address of programcounter using relative mode.

→ If condition is false then PC will be loaded with the address of next instruction.

7) Auto-Increment mode

→ Symbollicaly represented as $(R_i)+$
→ it is a post-increment.
→ First the contents of R_i (address) is fetched then ~~the~~ R_i gets incremented.
→ Program written to add n numbers using indirect addressing mode can be replaced by auto-increment mode, which eliminates the second add instruction.

```
        Move   N, R1
        Move   #Num1, R2
        Clear  R0
Loop    Add    (R2)+, R0      // Autoincrement mode.
        Decrement R1
        Branch >0 Loop
        Move   R0, Sum
```

→ R_2 will have the address of Num2 (ex: 1000) and in Num2 data is present.
→ using R_2 data is fetched then ~~the~~ R_2 gets increment to hold the next word address. Now R_2 will have the address of Num2 (ex:- 1004).

Auto-decrement mode

→ Symbollically represented as $-(R_i)$.
→ It is a pre decrement
→ In Auto-decrement mode first the address in R_i gets decremented then the data in that address is fetched.

Problems

1) Register R_1 and R_2 of a computer contain decimal values 1200 and 4600. what is the effective address of the memory operated on each of the following instructions?

1) Load 20(R_1), R5
Solun:- 20 + 1200 = 1220

2) Move #3000, R5
Solun:- Part of the instruction. or 3000

3) Store R5, 30(R_1 + R_2)
Solun:- 30 + 1200 + 4600 = 5830

4) Add $-(R_2)$, R5
Solun:- Predecrement so first R_2 gets decrement
i,e 4596 (

5) subtract $(R_1)+$, R5
Solun:- post increment so first R_2 is fetched then gets incremented.
i,e R_1 is 1200.

2) Write an ALP program to copy N numbers from array A to array B using indirect addressing. Assume A and B are starting of memory location of an array

Solution :-

```
    Move  N, R0
    Move  #A, R1
    Move  #B, R2
Loop Move  (R1), R3
    Move  R3, (R2)
    Add   #4, R1
    Add   #4, R2
    Decrement  R0
    Branch > 0  Loop
```

SHIRDI SAI ENGG COLLEGE

USN | | | | | | | | | | |

10CS46

Fourth Semester B.E. Degree Examination, June 2012
Computer Organization

Time: 3 hrs. Max. Marks:100

Note: ***Answer FIVE full questions, selecting atleast TWO questions from each part.***

PART – A

1 a. What is performance measurement? Explain the overall SPEC rating for the computer in a program suite. **(04 Marks)**

b. List and explain the technological features and devices improvement made during different generations of computers. **(08 Marks)**

c. Mention four types of operations to be performed by instructions in a computer. Explain with basic types of instruction formats to carry out C ← [A] + [B]. **(08 Marks)**

2 a. Define an addressing mode. Explain the following addressing modes, with example : immediate, indirect, index, relative and auto increment. **(10 Marks)**

b. What is a stack frame? Explain a commonly used layout for information in a subroutine stack frame.

c. Explain shift and rotate operations, with example. **(10 Marks)**

3 a. In a situation where multiple devices capable of initiating interrupts are connected to processor, explain the implementation of interrupt priority, using individual $\overline{\text{INTER}}$ and INTA and a common $\overline{\text{INTR}}$ line to all devices. **(10 Marks)**

b. Define the terms 'cycle stealing' and 'block mode'. **(02 Marks)**

c. What is bus arbitration? Explain the different approaches to bus arbitration. **(08 Marks)**

4 a. Explain with a neat block diagram, the hardware components needed for connecting a keyboard to a processor. **(08 Marks)**

b. Briefly discuss the main phases involved in the operation of SCSI bus. **(06 Marks)**

c. Explain the tree structure of USB with split bus operation. **(06 Marks)**

PART – B

5 a. Explain the internal organization of a 16 megabit DRAM chip, configured as 2 M × 8 cells. **(08 Marks)**

b. With a block diagram, explain the direct and set associative mapping between cache and main memory. **(06 Marks)**

c. Describe the principles of magnetic disk. **(06 Marks)**

6 a. Explain with figure the design and working of a 16-bit carry – look – ahead adder built form 4-bit adders. **(06 Marks)**

b. Explain Booth algorithm. Apply Booth algorithm to multiply the signed numbers +13 and – 6. **(10 Marks)**

c. Differentiate between restoring and non-restoring division. **(04 Marks)**

7 a. List out the actions needed to execute the instruction add(R_3), R_1. Write and explain sequence of control steps for the execution of the same. **(10 Marks)**

b. With a neat block diagram, explain hardwired control unit. Show the generation Z_{in} and End control signals. **(10 Marks)**

8 a. State Amdahl's law. Suppose a program runs in 100 sec on a computer with multiply operation responsible for 80 sec of this time, how much it requires to improve the speed of multiplication, if the program has to run 5 times faster? Justify your answer **(06 Marks)**

b. Explain the classic organization of a shared – memory multiprocessor. **(06 Marks)**

c. What is hardware multithreading. Explain the different approaches to hardware multithreading. **(08 Marks)**

* * * * *

SHIRDI SAI ENGG COLLEGE

USN | | | | | | | | | | |

10CS46

Fourth Semester B.E. Degree Examination, December 2012
Computer Organization

Time: 3 hrs. Max. Marks:100

Note: ***Answer FIVE full questions, selecting atleast TWO questions from each part.***

PART – A

1 a. Explain the different functional units of a digital computer. (05 Marks)

b. Draw and explain the connection between memory and processor with the respective registers. (05 Marks)

c. Explain clearly SPEC rating and its significance. Assuming that the reference computer is ultra SPARCIO work station with 300 MHz ultra SPARC processor. A company has to purchase 1000 new computers hence ordered testing of new computer with SPEC 2000. Following observation were made.

Programs	Runtime on reference computer	Runtime in new computer
1	50 minutes	5 Minutes
2	75 Minutes	4 Minutes
3	60 Minutes	6 Minutes
4	30 Minutes	3 Minutes

The company system manger will place the order for purchasing new computers only if the overall SPEC rating is atleast 12. After the said test will the system manger place order for purchase of new computer. (10 Marks)

2 a. What is little endian and big endian memory? Represent the number 64243848H in 32 bits big endian and little endian memory. (06 Marks)

b. What is addressing mode? Explain immediate, direct and indirect addressing mode by an example. (06 Marks)

c. Explain logical shift and rotate instructions, with examples. (08 Marks)

3 a. Define memory mapped I/O and IO mapped I/O, with examples. (05 Marks)

b. Explain how interrupt requests from several IO devices can be communicated to a processor through a single INTR line. (10 Marks)

c. What are the different methods of DMA? Explain them in brief. (05 Marks)

4 a. With a block diagram, explain how the keyboard is connected to processor. (06 Marks)

b. Explain the serial port and serial interface. (06 Marks)

c. Explain architecture and protocols, with respect to USB. (08 Marks)

PART – B

5 a. Draw a diagram and explain the working of 16 Mega bits DRAM chip configured as 2M × 8. Also explain as at how it can be made to work in fast page mode. (10 Marks)

b. Briefly explain any four non-voltile memory concepts. (05 Marks)

c. With figure analyse the memory hierarchy interms of speed cost and size. (05 Marks)

6 a. Explain the design of a four bits carry – look ahead adder circuit. **(10 Marks)**

b. Gives Booth's algorithm to multiply two binary numbers. Explain the working of algorithm by taking an example. **(10 Marks)**

7 a. Write and explain the control sequence for execution of an unconditional branch instruction. **(10 Marks)**

b. Draw and explain multiple bus organization. Explain its advantages. **(10 Marks)**

8 a. Write short note on power wall **(06 Marks)**

b. What you mean by shared memory multiprocessors. **(06 Marks)**

c. Explain the different approaches used in multithreading. **(08 Marks)**

* * * * *

USN | | | | | | | | | | |

10CS46

Fourth Semester B.E. Degree Examination, June / July 2013
Computer Organization

Time: 3 hrs. Max. Marks:100

Note: *Answer any FIVE full questions, selecting atleast TWO question from each part.*

PART - A

1 a. With a neat block diagram, discuss the basic operational concepts of a computer. (06 Marks)

b. List the different systems used to represent a signed number and give one example for each. Specify which number representation system is preferred in a computer and why? (04 Marks)

c. Perform the following operations on the 5 – bit signed numbers using 2's complement representation system. Also indicate whether overflow has occurred.
i) (-10) + (-13) ii) (-10) – (+4) iii) (-3) + (-8) iv) (-10) – (+7). (10 Marks)

2 a. Define addressing mode. Explain the following addressing modes with an example for each: i) Index addressing mode ii) Indirect addressing mode iii) Relative addressing mode iv) Auto decrement addressing mode. (10 Marks)

b. With a neat block diagram, describe the input and output operations. (05 Marks)

c. Discuss briefly encoding of machine instructions. (05 Marks)

3 a. With neat sketches, explain various methods for handling multiple interrupt requests. (12 Marks)

b. Define bus arbitration. Explain in detail any one approach of bus arbitration. (08 Marks)

4 a. With a neat diagram, explain in detail the input interface circuit. (10 Marks)

b. List out the functions of an I/O interface. (03 Marks)

c. Discuss briefly the protocols of universal serial bus. (07 Marks)

PART - B

5 a. Briefly explain any two cache mapping functions. (06 Marks)

b. With a neat diagram, explain the translation of a virtual address to a physical address. (08 Marks)

c. Discuss in detail any one feature of memory design that leads to improved performance of computer. (06 Marks)

6 a. Perform signed multiplication of numbers (-12) and (-11) using Booth's algorithm. (08 Marks)

b. Given A = 10101 and B = 00100 perform A/B using restoring division algorithm. (08 Marks)

c. Design a logic circuit to perform addition / subtraction of two 'n' bit numbers X and Y. (04 Marks)

7 a. Write down the control sequence for the instruction Add R_4, R_5, R_6 for three – Bus organization. (04 Marks)

b. With a neat sketch, explain the organization of a micro programmed control unit. (08 Marks)

c. With an example, explain the field coded microinstructions. (08 Marks)

8 a. Describe the working of message passing multicomputer (MPM) architecture. (08 Marks)

b. Briefly explain any two parallel computer architecture. (08 Marks)

c. List out any four differences between shared memory multiprocessor and cluster. (04 Marks)

*****

USN | | | | | | | | | | |

10CS46

Fourth Semester B.E. Degree Examination, June 2012
Computer Organization

Time: 3 hrs. Max. Marks:100

Note: *Answer FIVE full questions, selecting atleast TWO questions from each part.*

PART – A

1 a. What is performance measurement? Explain the overall SPEC rating for the computer in a program suite. (04 Marks)

b. List and explain the technological features and devices improvement made during different generations of computers. (08 Marks)

c. Mention four types of operations to be performed by instructions in a computer. Explain with basic types of instruction formats to carry out C ← [A] + [B]. (08 Marks)

2 a. Define an addressing mode. Explain the following addressing modes, with example : immediate, indirect, index, relative and auto increment. (10 Marks)

b. What is a stack frame? Explain a commonly used layout for information in a subroutine stack frame.

c. Explain shift and rotate operations, with example. (10 Marks)

3 a. In a situation where multiple devices capable of initiating interrupts are connected to processor, explain the implementation of interrupt priority, using individual $\overline{INTER}$ and INTA and a common $\overline{INTR}$ line to all devices. (10 Marks)

b. Define the terms 'cycle stealing' and 'block mode'. (02 Marks)

c. What is bus arbitration? Explain the different approaches to bus arbitration. (08 Marks)

4 a. Explain with a neat block diagram, the hardware components needed for connecting a keyboard to a processor. (08 Marks)

b. Briefly discuss the main phases involved in the operation of SCSI bus. (06 Marks)

c. Explain the tree structure of USB with split bus operation. (06 Marks)

PART – B

5 a. Explain the internal organization of a 16 megabit DRAM chip, configured as 2 M × 8 cells. (08 Marks)

b. With a block diagram, explain the direct and set associative mapping between cache and main memory. (06 Marks)

c. Describe the principles of magnetic disk. (06 Marks)

6 a. Explain with figure the design and working of a 16-bit carry – look – ahead adder built form 4-bit adders. (06 Marks)

b. Explain Booth algorithm. Apply Booth algorithm to multiply the signed numbers +13 and −6. (10 Marks)

c. Differentiate between restoring and non-restoring division. (04 Marks)

SHIRDI SAI ENGINEERING COLLEGE

USN | 1 | S | B | 1 | 1 | C | S | 0 | 2 | S |

10CS46

Fourth Semester B.E. Degree Examination, Dec.2013/Jan.2014
Computer Organization

Time: 3 hrs. Max. Marks:100

Note: ***Answer FIVE full questions, selecting at least TWO questions from each part.***

PART – A

1 a. Explain with necessary block diagram the basic functional unit of a computer. (10 Marks)
b. Explain in detail the generation of computers. (10 Marks)

2 a. Big-Endian and Little-Endian assignments, explain with necessary figure. (05 Marks)
b. List the name, assembler syntax and addressing functions for the different addressing modes. (10 Marks)
c. Explain logical and arithmetic shift instructions with an example. (05 Marks)

3 a. Draw the arrangement of a single bus structure and brief about memory mapped I/O. (05 Marks)
b. Explain: i) Interrupt enabling; ii) Interrupt disabling; iii) Edge triggering, with respect interrupts. (10 Marks)
c. Draw the arrangement for bus arbitration using a daisy chain and explain in brief. (05 Marks)

4 a. Draw the hardware components needed for connecting a keyboard to a process and explain in brief. (10 Marks)
b. Explain the use of PCI bas in a computer system with necessary figure. (05 Marks)
c. List the SCSI bus signals with their functionalities. (05 Marks)

PART – B

5 a. Draw for 1K × 1 memory chip with neat figure. (10 Marks)
b. Show with diagram the memory hierarchy with respect to speed, size and cost. (05 Marks)
c. With figure explain about direct mapping cache memory. (05 Marks)

6 a. Explain with figure the design of a 4-bit carry look ahead adder. (10 Marks)
b. With figure explain circuit arrangements for binary division. (05 Marks)
c. IEEE standard for floating point numbers, explain. (05 Marks)

7 a. Draw and explain the single-bus organization of the data path inside a processor. (10 Marks)
b. Write the control sequence for an un-conditional branch instruction. (05 Marks)
c. Draw the block diagram of the control unit organization and describe in brief. (05 Marks)

8 a. Explain in brief about multiprocessor systems. (05 Marks)
b. Explain about memory organization in multiprocessors. (10 Marks)
c. What are the classifications of parallel processing? (05 Marks)

* * * * *

SHIRDI SAI ENGINEERING COLLEGE

USN | | | | | | | | | | |

10CS46

Fourth Semester B.E. Degree Examination, June/July 2014
Computer Organization

Time: 3 hrs. Max. Marks:100

Note: ***Answer FIVE full questions, selecting atleast TWO questions from each part.***

PART – A

1 a. Draw the connection between processor and memory and mention the functions of each component in the connection. **(08 Marks)**

b. Write the difference between RISC and CISC processors. **(04 Marks)**

c. A program contain 1000 instructions. Out of that 25% instructions requires 4 clock cycles, 40% instructions requires 5 clock cycles and remaining requires 3 clock cycles for execution. Find the total time required to execute the program running in a 1 GHz machine. **(05 Marks)**

d. Add +5 and –9 using 2's compliment method. **(03 Marks)**

2 a. Explain immediate, indirect and indexed addressing modes. **(08 Marks)**

b. Explain different rotate instructions. **(06 Marks)**

c. Write ALP program to copy 'N' numbers from array 'A' to array 'B' using indirect addresses. (Assume A and B are the starting memory location of a array). **(06 Marks)**

3 a. Explain the following terms :
i) Interrupt service routine ii) interrupt latency iii) interrupt disabling. **(06 Marks)**

b. With a diagram, explain daisy chaining technique. **(06 Marks)**

c. What you mean by bus arbitration? Briefly explain different bus arbitration techniques. **(08 Marks)**

4 a. With a block diagram, explain how the printer is interfaced to processor. **(08 Marks)**

b. Explain the architecture and addressing scheme of USB. **(08 Marks)**

c. Define two types of SCSI controller. **(04 Marks)**

PART – B

5 a. Explain direct memory mapping technique. **(06 Marks)**

b. What is virtual memory? With a diagram, explain how virtual memory address is translated. **(08 Marks)**

c. Explain the working of 16 megabyte DRAM chip configured as 1 M × 16 memory chip. **(06 Marks)**

6 a. Design 4 bit carry look ahead logic and explain how it is faster them 4 bit ripple adder. **(08 Marks)**

b. Multiply 14 × –8 using Booth's algorithm. **(06 Marks)**

c. Explain normalization, excess – exponent and special values with respect to IEEE floating point representation. **(06 Marks)**

7 a. With a diagram, explain typical single bus processor data path. **(08 Marks)**

b. Explain with neat diagram, the basic organization if a microprogrammed control unit. **(08 Marks)**

c. Differentials hardwired and microprogrammed control unit. **(04 Marks)**

8 a. Define and discuss Amdahl's law. **(06 Marks)**

b. With a diagram, explain a shared memory multiprocessor architecture. **(08 Marks)**

c. What is hardware multithreading? Explain different approaches in hardware multithreading. **(06 Marks)**

* * * * *

USN | | | | | | | | | | |

10CS46

Fourth Semester B.E. Degree Examination, Dec.2014/Jan. 2015
Computer Organization

Time: 3 hrs. Max. Marks:100

Note: ***Answer FIVE full questions, selecting atleast TWO questions from each part.***

PART – A

1 a. Explain the basic operational concepts between the processor and the memory. **(08 Marks)**
b. How to measure the performance of the computer? Explain. **(06 Marks)**
c. Write a note on byte addressability, big-endian and little – endian assignment. **(06 Marks)**

2 a. What is an addressing mode? Explain any four types of addressing modes, with example. **(08 Marks)**
b. With example, explain subroutine stack frame. **(06 Marks)**
c. Explain how to encode the instructions into 32 – bit words. **(06 Marks)**

3 a. What is an interrupt? With example illustrate the concept of interrupts. **(06 Marks)**
b. Explain in detail, the situations where a number of devices capable of initiating interrupts are connected to the processor? How to resolve the problems? **(08 Marks)**
c. Explain the two approaches for bus arbitration. **(06 Marks)**

4 a. Describe how a read operation is performed on a PCI bus. **(10 Marks)**
b. List the sequence of events that takes place when a processor sends a commands to the SCSI controller. **(10 Marks)**

PART – B

5 a. Discuss the internal organization of a 2M × 8 asynchronous DRAM chip. **(10 Marks)**
b. Describe the different mapping functions in cache. **(10 Marks)**

6 a. Write the logic diagram of 4 – bit carry look ahead adder. Explain the operation. **(06 Marks)**
b. Perform multiplication for –13 and + 9 using Booth's algorithm. **(06 Marks)**
c. Write the circuit arrangement for binary division. Perform the restoring division for the given binary numbers 1000 ÷ 11, show all the cycles. **(08 Marks)**

7 a. Explain the three – bus organization of the processor. **(08 Marks)**
b. Discuss the organization of hardwired control unit. **(08 Marks)**
c. Write the micro-routine for the instruction Add – (Rsrc), Rdst. **(04 Marks)**

8 a. With a neat diagram, explain the organization of a shared memory multiprocessor. **(08 Marks)**
b. What is hardware multithreading? Explain the three approaches to hardware multithreading. **(08 Marks)**
c. Explain : i) SISD ii) MIMD iii) SIMD and iv) SPMD. **(04 Marks)**

* * * * *

7 a. List out the actions needed to execute the instruction add(R_3), R_1. Write and explain sequence of control steps for the execution of the same. (10 Marks)

b. With a neat block diagram, explain hardwired control unit. Show the generation Z_{in} and End control signals. (10 Marks)

8 a. State Amdahl's law. Suppose a program runs in 100 sec on a computer with multiply operation responsible for 80 sec of this time, how much it requires to improve the speed of multiplication, if the program has to run 5 times faster? Justify your answer (06 Marks)

b. Explain the classic organization of a shared – memory multiprocessor. (06 Marks)

c. What is hardware multithreading. Explain the different approaches to hardware multithreading. (08 Marks)

* * * * *

USN | | | | | | | | | | |

06CS46

Fourth Semester B.E. Degree Examination, Dec.2014/Jan.2015
Computer Organization

Time: 3 hrs. Max. Marks:100

Note: ***Answer FIVE full questions, selecting at least TWO questions from each part.***

PART – A

1 a. List the steps needed to execute the machine instructions

i) Add LOC A, R0 ii) Add R1, R2, R3

in terms of transfers with the help of suitable diagram and some simple control commands. Assume that the instruction itself stored in the memory location INSTR and that this address is initially in register PC. The first two steps might be expressed as

* Transfer the contents of Register PC to register MAR
* Issue a Read command to the memory and then wait until it has transferred the requested word into register MDR.

Remember to include the steps needed to update the contents of PC from INSTR to INSTR+1. So that the next instruction can be fetched. **(10 Marks)**

b. Registers R1 and R2 of a computer contain the decimal values 1200 and 4600. What is the effective address of the memory operated in each of the following instructions:

i) Load 20(R1), R5 ii) MOVE #3000, R5 iii) Store R5, 30(R1, R2)

iv) Add – (R2), R5 v) Subtract (R1)+, R5 **(05 Marks)**

c. Bring out the differences between RISC and CISC. **(05 Marks)**

2 a. "Having a large number of processor registers makes it possible to reduce the number of memory accesses needed to perform complextasks." Device a simple computational task to show the validity of this statement for a processor that has four registers compared to another that has only two registers. **(10 Marks)**

b. Register R5 is used in a program to point to the top of a stack. Write a sequence of instructions using the index, Auto increment and auto decrement addressing modes to perform each of the following tasks:

i) Pop the top two items off the stack, add them and the push the results onto the stack.
ii) Copy the fifth item from the top into register R3.
iii) Remove the top ten items from the stack. **(06 Marks)**

c. Write a program that can evaluate the expression

$$A \times B + C \times D$$

in a single accumulator processor. Assume that the processor has load, store, multiply and add instructions and that all values fit in the accumulator. **(04 Marks)**

3 a. What is the difference between a subroutine and an interrupt service routine? **(04 Marks)**

b. Three devices A, B and C are connected to the bus of a computer. I/O transfers for all three devices use interrupt control. Interrupt nesting for devices A and B is not allowed, but interrupt requests from C may be accepted while either A or B is being serviced. Suggest different ways in which this can be accomplished in each of the following cases:

i) The computer has one interrupt request line.
ii) Two interrupt request lines, INTR1 and INTR2 are available, with INTR1 having higher priority.

Specify when and how interrupts are enabled and disabled in each case. **(06 Marks)**

c. The address bus of a computer has 16 address lines A_{15-0}. If the address assigned to one device is $7CA4_{16}$ and the address decoder for that device ignores lines A_8 ad A_9, what are all the addresses to which this device will respond? **(04 Marks)**

d. Consider a computer in which several devices are connected to a common interrupt request line as shown below. Refer Fig.Q3(d).

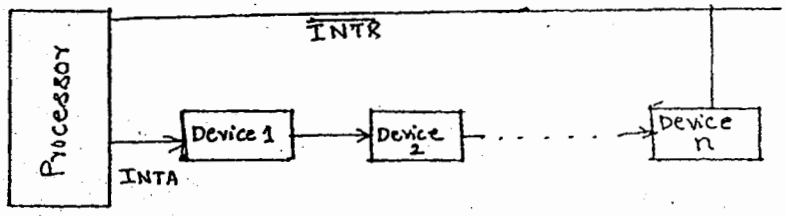


Fig.Q3(d)

Explain how you would arrange for interrupts from device J to be accepted before the execution of the interrupt-service routine for device I is completed. Comment in particular on the times at which interrupts must be enables and disabled at various points in the system. **(06 Marks)**

4 a. Consider a write operation on a bus that uses multiple cycle scheme. Assume that the processor can send both address and data in the first clock cycle of a bus transaction. But the memory requires two clock cycles after that to store the data.
   i) Can the bus be used for another transactions during that period?
   ii) Can we do away with the memory's response in this case? **(10 Marks)**

b. What is isochronous data stream? What is the main requirement for isochronous traffic? How universal serial bus (USB) support the isochronous data traffic? Explain briefly. **(10 Marks)**

PART – B

5 a. Give the block diagram for a 8M×32 memory using 512×8 memory chips. **(08 Marks)**

b. Bring out the differences between the following :
   i) RAM and ROM ii) SRAM and DRAM **(08 Marks)**

c. In many computers the cache block size is in the range of 32×128 bytes. What would be the main advantages and disadvantages of making the size of cache blocks larger or smaller? **(04 Marks)**

6 a. A disk unit has 24 recording surfaces. It has a total of 14,000 cylinders. There is an average of 400 sectors per track. Each sector contains 512 bytes of data.
   i) What is the maximum number of bytes that can be stored in this unit?
   ii) What is the data transfer rate in bytes per second at a rotational speed of 7200 rpm?
   iii) Using a 32 bit word, suggest a suitable scheme for specifying the disk address assuming that there are 512 bytes per sector. **(10 Marks)**

b. A half adder is a combinational logic circuit that has two inputs x and y and two outputs s and c, that are the sum and carry-output respectively, resulting from the binary addition of x and y.
   i) Design a half adder as a two level AND-OR circuit.
   ii) Show how to implement a full adder by using two half adders and external logic gates as necessary. **(10 Marks)**

RNS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING.

QUESTION BANK -1 (Module –I & 2)

Subject: Computer Organization **Code: 15CS34**
Semester: 3 CSE-A & B **Date: 01/09/2016**
Staff: Prof. H.R. Shashidhar and Prof. Medha Gourayya

1. With a neat block diagram explain the functions of the following processor registers
   (i)PC (ii) MAR (iii) IR (iv) MDR
2. Explain the block diagram of connections between the processor and the memory and explain how the Following typical instructions can be executed
   i) Add LOCA, R0
   ii) Load LOCA, R1
   iii) Add R1, R0
   iii) Move (R3), R2
3. What is a BUS? Explain single bus structure used to interconnect functional units in computer system?
4. Define a Digital Computer. Explain different functional units of a digital computer.
5. Explain how the performance of a computer can be measured. What are the measures to improve the performance of computers?
6. Explain: (i) Byte addressability
   (ii) Big-endian assignment
   (iii) Little-endian assignment
7. Mention four types of operations required to be performed by instruction in a computer. What are the basic types of instruction formats? Give an example for each?
8. Briefly explain the history of computer development from first generation to fourth generation computers?
9. For a simple example of I/O operations involving a keyboard and a display device, write an assembly language program that reads one line from the keyboard, stores it in memory buffer and echoes it back to the display?
10. What are condition code flags? Explain the use of them?
11. Write a straight-line program for adding **n** numbers using indirect addressing mode and immediate addressing mode.
12. What are assembler directives? Explain the functions following assembler directives:
    (i). EQU (ii). ORIGIN (iii). DATAWORD (iv). RESERVE (v). RETURN
    (vi). START (vii). END
13. Write a program that can evaluate the expression
    1) $S = (A * B) + (C * D)$ 2) $Y = (A + B) * (C + D)$
    Using One Address, Two Address, Three Address and Zero Address formats.
14. Registers R1 and R2 of a computer contain the decimal values 1200 and 4600.
    What is the effective address of the memory operand in each of the following instructions?
    (a). Load 20(R1), R5
    (b). Move #3000, R5
    (c). Store R5, 30(R1,R2)
    (d). Add -(R2), R5
    (e). Subtract (R1)+, R5

15. Explain clearly SPEC rating and its significance?
16. What is an Addressing Mode? Explain the different types of addressing modes with an example.
17. What is subroutine linkage ? explain with an example, subroutine linkage using linkage register ?
18. What is stack frame? Illustrate the use of stack frame in the mechanism for implementing subroutines.
19. Write the basic performance equation .Explain the role of each of the parameters in the equation on the performance of a computer ?
20. Show how the operation C = A + B can be implemented in a single accumulator computer by (i) Three –address instruction (ii) Two -address instruction (iii) One -address instruction
21. What is Stack? Explain its role in subroutine nesting.

22. Represent the number 81234561 in 32 bit Big-endian and Little-endian memory organization.
23. Explain the **LOGICAL, SHIFT,** and **ROTATE** instructions with examples.
24. Explain the concept of stack frames when subroutines are nested .
25. List and explain three important differences between how a stack and a queue are organized.
26. With the help of suitable examples, illustrate encoding of machine instructions.

27. What is word alignment of a machine (microprocessor based system)? Explain. What are the consecutive addresses of aligned words for 16, 32, and 64 bit word lengths of machines? Give two consecutive addresses for each case.
28. Bring out the differences between subroutine and interrupt service routine.
29. Explain the concept of parameter passing with respect to subroutine with Assembly code.
30. Explain the important functions of an I/O interface for an input device with a neat diagram.
31. Explain Program-controlled I/O with a program that reads one line from the Key board , stores it in Memory Buffer, and Echoes it back to display.
32. What is an Interrupt ? Explain with Compute and Print routines.

****

Note : Consider this Question Bank as Assignment-1 and Submit the solutions On or Before : 15--09-2016

****

Assembly Language.

The programming language written using Mnemonics like MOV, ADD etc with some rules is known as assembly language.

→ Assembly language is converted to machine language by Assembler.

→ Assembly language is not case sensitive.

→ In Assembly language instructions has two parts : opcode and operands. Ex:- MOVE R0, R1.

Assembler Directives.

→ Assembler directives are the non executable statements. They are not the instructions like MOV, ADD and so on.

Some of the Assembler Directives are

1) EQU
2) ORIGIN
3) DATAWORD
4) RESERVE
5) END
6) RETURN

→ Assembler directives instructs the assembler to perform specific operations.

1) EQU Directive (equate)

→ EQU directive is similar to define directive of high level language.

Ex:- #define PI 3.14

This directive replaces PI with 3.14 wherever PI is written in the program.

→ SUM EQU 2000

In this EQU directive instructs the assembler to replace SUM with 2000 wherever it finds SUM.

2) DATAWORD Directive

→ DATAWORD directive instructs an assembler to place a specific value in the memory location (just like initilization)

N DATAWORD 200

→ 200 is placed in the memory location named N.

3) ORIGIN Directive.

→ ORIGIN directive instructs the assembler to move to a particular location. Then the instruction following ORIGIN directive will be executed.

Ex:- ORIGIN 204

The assembler moves to the memory location 204. Then any instruction below this directive will be executed.

Ex:- ORIGIN 204

```
MOVE N, R1
 .
 .
 .
```

4) RESERVE Directive

→ RESERVE directive informs the assembler to reserve a chunk of memory.

Ex:- NUM1 RESERVE 400

This reserves 400 words in the memory and the name of this memory chunk is NUM1.

5) END Directive

→ END START

This directive implies that the source code ends here. Now go and start executing the code from the label "START".

6) RETURN Directive.

→ RETURN directive means that execution should stop when assembler encounters this directive.

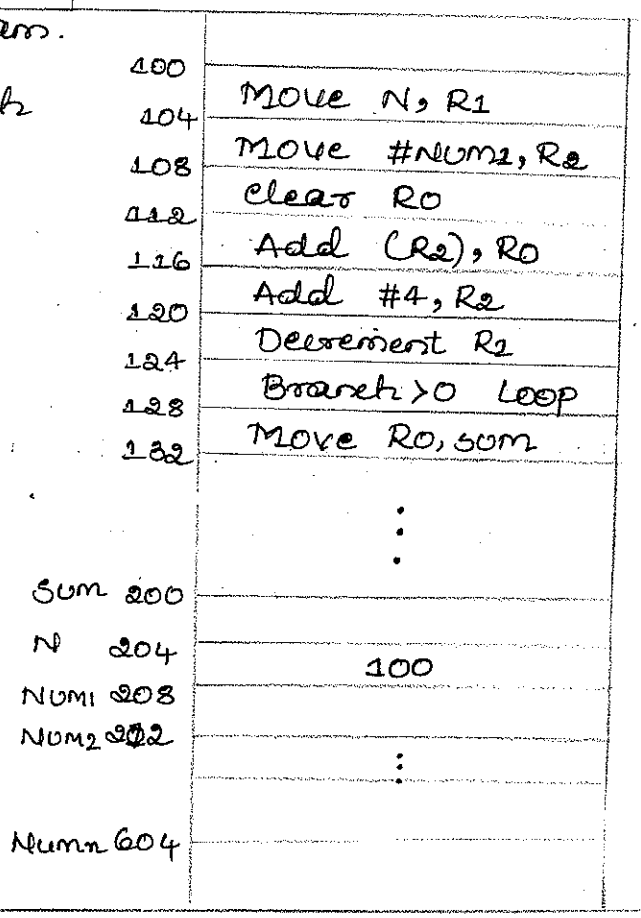
Consider the program below which uses these assembler Directives.

Source code to add all numbers Num2 to Numn.

	Memory address label	operation	Addressing or data information
Assembler directives	SUM	EQU	50
		ORIGIN	204
	N	DATAWORD	100
	NUM1	RESERVE	400
		ORIGIN	100
Statements that generate machine instructions	START	MOVE	N, R1
		MOVE	#NUM1, R2
		CLR	R0
	LOOP	ADD	(R2), R0
		ADD	#4, R2
		DEC	R1
		BGTZ	LOOP
		MOVE	R0, SUM
Assembler directives		RETURN	
		END	START

Memory layout for the program.

NOte;- BGTZ stands for Branch greater than zero.



Syntax for assembly language instruction is as follows.

Label	operation	operands	comment

Label : Is a memory address.
operation : Is the opcode for the instruction.
operands : Source and destination on which operation is performed.
comment : Is an optional field used to describe instruction

Assembly and execution of the program.

→ Assembler directives instructs the assembler to perform some specific task. (Ex DATAWORD, ORIGIN, EQU)
→ Assembler maintains specific numbers for these names.
→ These names and numbers are maintained in the symbol table.
→ Assembler goes through the code in two-passes. In first pass it collects the names in the symbol table and in the second pass, it maps the values to these names. Such an assembler is known as Two-pass Assembler.

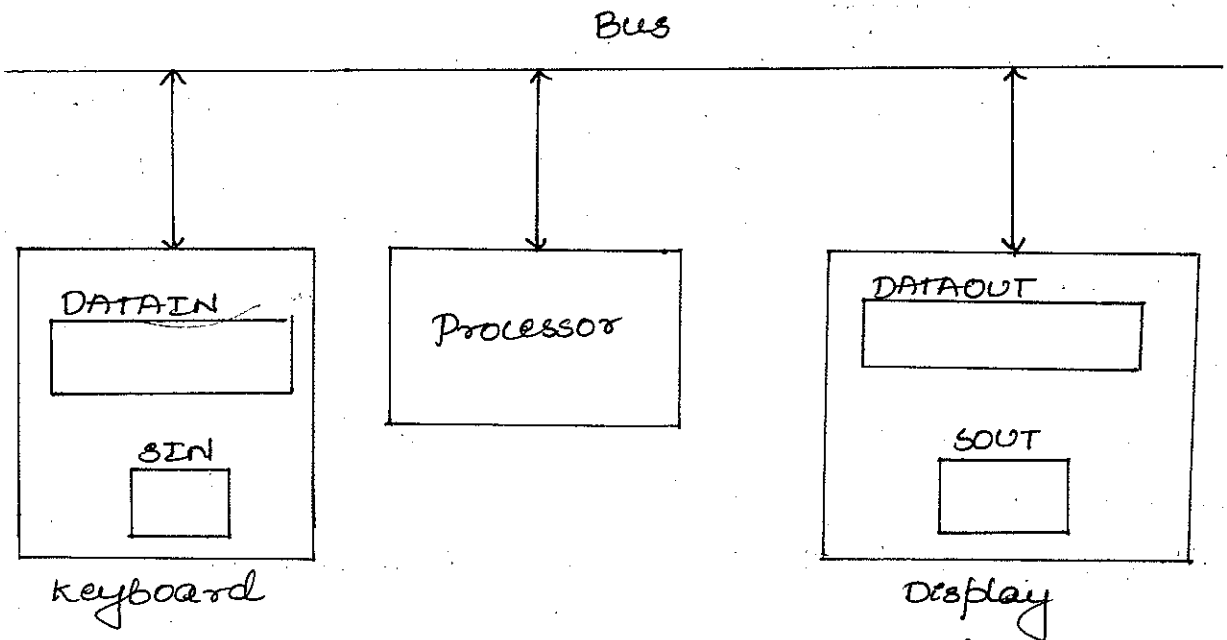
Number Notation.

Assembler accepts the numbers in three formats.

1) Decimal number
2) Binary number
3) Hexadecimal number

1) Decimal number :- Add instruction adds 24 to register R2
2) Binary number :- '%' informs assembler that the number following it is a binary number.
   Ex:- Add #%00111001, R2
3) Hexadecimal number :- '$' informs assembler that the number following it is a hexadecimal number.

Basic Input output operation



→ In the previous examples we considered that data was read from memory and result was stored on memory.
→ Usually an input is given or read from keyboard and output is displayed on monitor.
→ When a key is pressed or typed on keyboard, generally it seems to appear instantaneously on the monitor making us belive that the character was directly typed on monitor, which is not true.
→ Every keypress goes via processor.
→ Consider the above diagram where all the devices are connected via a single bus.
→ Compared to the processor, speed of the input output device is very slow.
→ To synchronize the timing difference, buffer registers are used for slow devices (keyboard & display).
→ Buffer register placed at keyboard is named as DATAIN, which is a 8 bit register.
→ Buffer register placed at display is named as DATAOUT.
→ In the status register two bits are used called as SIN bit and SOUT bit.

Consider the steps of how the character are read from keyboard and instantly displayed on the screen.

In the Input device.

→ Initally SIN flag will be zero.
→ When a key is pressed corresponding character is moved to DATAIN.
→ Now SIN flag is set to 1.
→ Processor will check the status regester. Since it is 1 it reads the character.
→ Once the processor reads the character SIN flag is set to 0.

Note :-

1) If SIN flag is zero means there is no character in DATAIN and processor is waiting for the character.
2) If SIN flag is 1 it means DATAIN has a valid character and processor will read the character.

In the output device.

→ Initally Sout flag is set to 1.
→ DATAOUT is empty, it is ready to accept the data from processor.
→ When processor sends a character to DATAOUT Sout flag will be set to zero.
→ Once the character has been printed Sout flag will be set to 1, indicating it can accept second character.

Note:-

1) If Sout is 1 means there is no character in DATAOUT to print. It is ready to accept the character from processor
2) If Sout is 0 means there is a character in DATAOUT and display device is busy in printing or displaying it.

Pseudo-code (Instruction) for reading the data from keyboard.

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

which means if sin = 0 then processor has to wait for user to press a key. As soon as SIN becomes 1 (user presses a key) the character from DATAIN is moved to register R0.

Pseudo-instruction for displaying the data on screen.

```
WRITEWAIT   Branch to WRITEWAIT if SOUT = 0
            Output from R2 to DATAOUT
```

which means as long as SOUT is zero display is busy on displaying the character on screen so processor has to wait. Once SOUT becomes 1, processor sends the character from R_1 to DATAOUT.

SIN and SOUT are the part of status registers "INSTATUS" and "OUTSTATUS". we'll consider the 3rd bit of INSTATUS and OUTSTATUS is reserved for SIN and SOUT.
The code to read and write can be written as follows

```
READWAIT    Testbit   #3, INSTATUS
            Branch =0 READWAIT
            MoveByte  DATAIN, R1
```

→ 3rd bit of INSTATUS register is checked, if it is zero then processor has to wait until a character is entered by user.
→ As soon as 3rd bit of INSTATUS register becomes 1, the character is moved from DATAIN to R1.

```
WRITEWAIT   Testbit   #3, OUTSTATUS
            Branch =0 WRITEWAIT
            MoveByte  R1, DATAOUT
```

→ 3rd bit of OUTSTATUS register is checked, if it is 0, then display is busy in displaying the character so processor has to wait.
→ Once the 3rd bit of OUTSTATUS register becomes 1, processor sends the character from register R1 to DATAOUT.

Note :- MoveByte opcode is used because we are considering buffer register as 8 bit registers. whenever a byte information has to be transferred MoveByte opcode is used.

Example Program.

→ Reads one character displays on screen till a newline is entered.

```
        Move    #LOC, R0        Initalize pointer register R0 to point to the address of first location in memory where the characters are to be stored.

READ    Testbit #3, INSTATUS    wait for character to be entered in the keyboard buffer DATAIN.
        Branch =0  READ
        MoveByte DATAIN, (R0)   Move the character from DATAIN to memory. (this clears SIN to 0)

WRITE   Testbit #3, OUTSTATUS   wait for display to become ready.
        Branch =0  WRITE
        MoveByte (R0), DATAOUT  Move the character just read to display. (this clears SOUT to 0)

        Compare #CR, (R0)+      Check of the character just read is carriage return. If it is not CR, then branch back and read another character. Also, increment the pointer to store next character

        Branch ≠0  READ
```

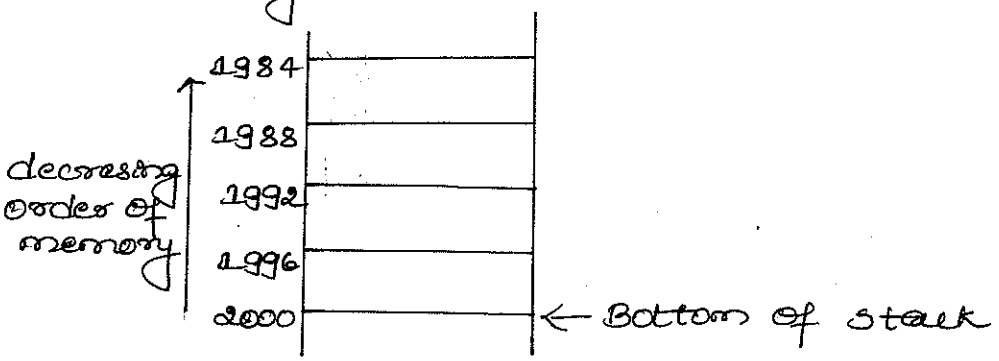
Stacks and Queues.

Stack is a chunck of memory in which data elements are pushed and popped out in last-in-First-out manner.

Defination.

Stack is a collection of data elements (usually words or bytes) with the accessing restriction that elements can be added or removed from one end only. Also known as Pushdown stack.

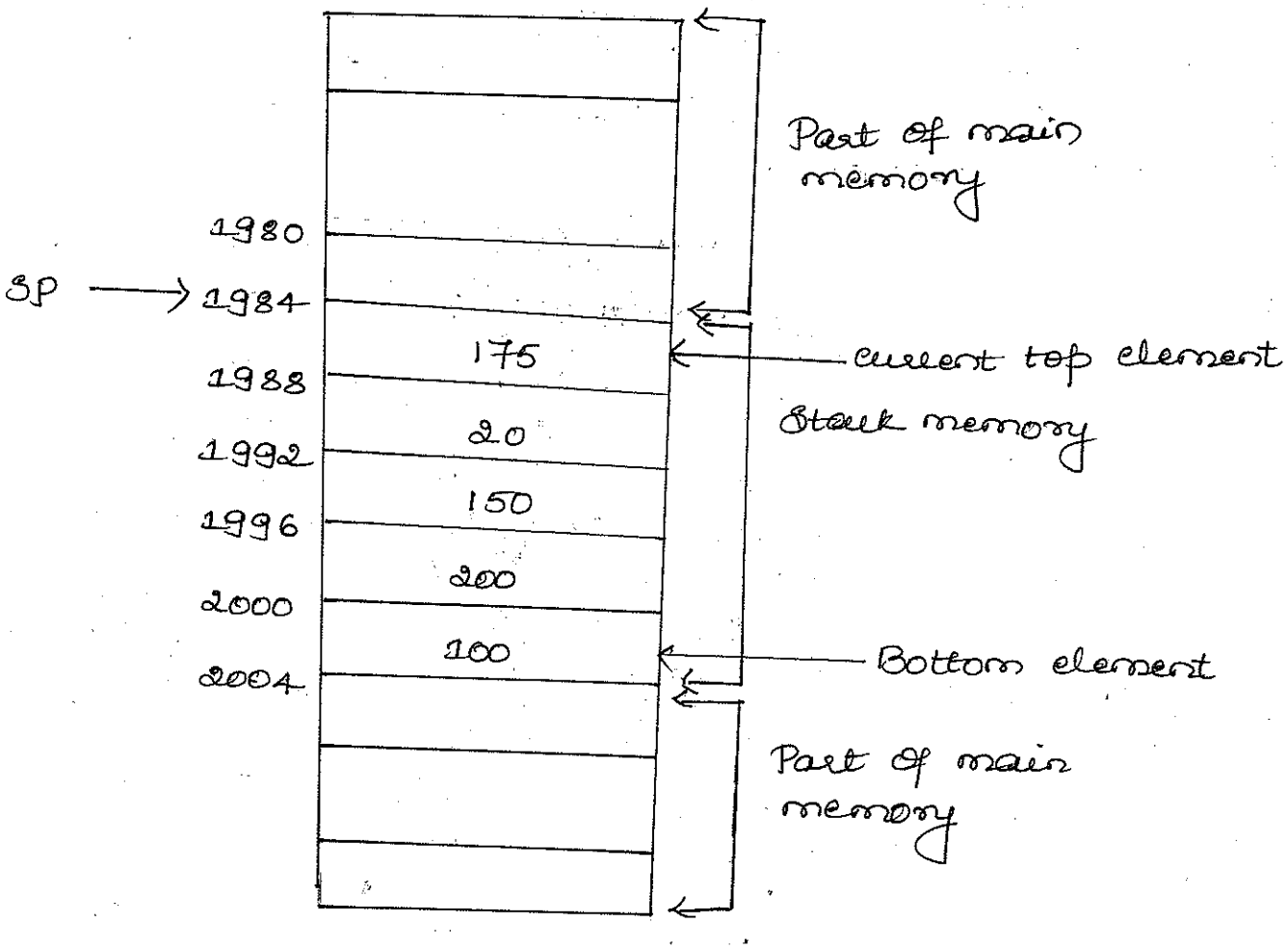
→ First element is placed at the bottom of the stack, rest of the elements are pushed on to the stack in decreasing order of their memory address.



→ In order to keep track of top most element of the stack, processor maintains a register called as Stack Pointer.

→ Stack Pointer will always have the address of topmost element in the stack.

Consider a stack.



Push operation.

To perform push operation first stack pointer should be decremented and then place newelement in that address

i.e Subtract #4, SP (SP becomes 1980)

MOVE NEWELEMENT, (SP) (In the address SP place the value from the memory NEWELEMENT)

Suppose NEWELEMENT is 80 then it is placed in the address 1980.

Pop operation

→ To perform pop operation first the element to be poped is placed on some memory location and then stack pointer is incremented.

i,e

```
Move (SP), Item
Add #4, SP
```

Instead of these two instructions pop can be written with one instruction using Autoincrement mode.

i,e Move (SP)+, Item

→ Push operation can be performed using Autodecrement mode

i,e Move Newelement, — (SP)

Before stack is used for our purpose the size of stack is fixed. In our example stack size is ranging from the address 2000 to 1984. Above 1984 and below 2000 is not a stack memory. So whenever pop and push operations are performed some condetions has to be taken care of.

→ To perform push operation stack overflow condition has to be checked. If stack is full in our example if SP is pointing to 1984 stack is full.

A Pseudo-instruction to check for overflow condition and perform push operation.

```
SAFEPUSH   Compare #1980, SP      check if sp contains a value
           Branch ≤0 Fullerror    less than or equal to 1980 If
           Move Newelement, -(SP) true then stack is full. Else
                                  decrement sp and move
                                  newelement into the
                                  address pointed by sp.
```

Pseudo-instruction to check for underflow condition and perform pop operation.

```
SAFEPOP    Compare #2000, SP      check if sp contains a value
           Branch >0 EMPTYERROR   greater than 2000 if yes then
           Move (SP)+, item       stack is empty. Else place
                                  the element into memory item
                                  and increment sp.
```

Note:- Compare instruction i,e compare src, dst performs the operation [dst] − [src]. It dosent change the value of any operand but it effects the conditional code flags.

Queue

A queue is a First-in Last-out datastructure. If queue is assumed to grow in higher address, then elements are added to higher and elements are removed from lower address.

→ Stack can be managed by only one pointer.
→ Queue requires two pointers one at the beginning and one at the end.
→ Circular buffer can be used for a queue.

Subroutines (functions)

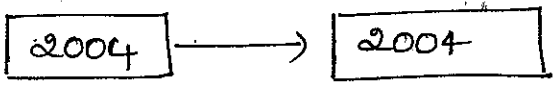
→ Subroutine is a set of instructions which performs a task.
→ Subroutine can be called repetatedly.
→ The instruction that performs the branch is called as Call instruction. (Subroutine call)

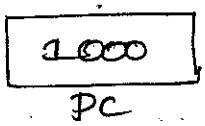
Memory location	calling program	Memory location	Subroutine SUB
	:		
2000	call SUB	1000	first instruction
2004	next instruction		:
			Return.

Note:- After a subroutine has been executed, calling program must resume execution, continuing immediately after the instruction that called subroutine

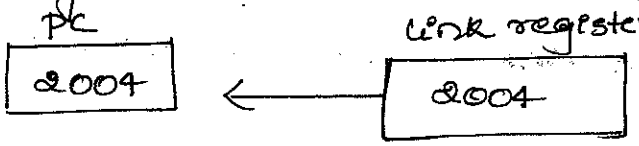
→ Execution begins from main.
→ Once the call instruction has been encountered the contents of PC i.e 2004 is stored in a special register called as link register.
→ Then PC is loaded with the new address of subroutine in the above example PC is loaded with 1000.
→ After the execution of subroutine because of return instruction control transfers to calling program.
→ Contents of link register i.e 2004 is loaded onto PC.
→ Now the instruction from address 2004 will be executed.

During call instruction.





During return instruction



Subroutine Nesting and Processor Stack.

Ex:-

```
main()
{
2000    add();
2004    next instruction;
        return;
}
```

```
add()
{
3000 sub();
3004 next instruction;
     return;
}
```

```
sub()
{
4000 instructions;
     return;
}
```

→ main() function calls the subroutine add and the return address i.e 2004 will be stored in link register.

→ This is the address where the control should return on execution of add() subroutine.

→ Subroutine add() calls the subroutine sub() and the return address i.e 3004 will be stored in the link register overwriting the previous contents of link register.

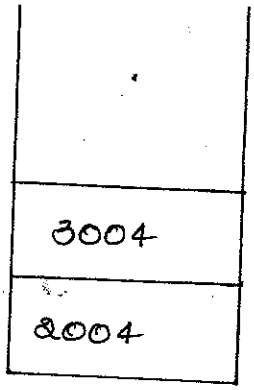
→ Hence when the subroutine completes its execution it returns to the address 3004.

→ But from the add() subroutine it is not possible to return to main() since processor has no information where to return.

→ To overcome this problem, processor stack is used to store the return address.

→ Whenever a subroutine is called return address is pushed on to the stack.

→ When return stmt executes on subroutine the return address from stack is poped.



Processor stack

→ When a subroutine is called contents of PC is pushed on to the stack. PC gets loaded with the address of subroutine

→ Return instruction will pop the contents from the stack and loads it into PC.

Parameter passing to subroutine

1) Passing parameter via registers or memory locations
2) Passing parameter through processor stack.

1) Parameter passing via registers.

Consider a program of adding 'n' numbers

Calling program

```
Move   N, R1          R1 is a counter
Move   #Nums, R2      R2 points to the array named Nums.
Call   ADDLIST        Subroutine call
Move   R0, SUM        Save result.
  :
```

Subroutine

```
ADDLIST  Clear  R0
LOOP     Add    (R2)+, R0     Add the no's from array
         Decrement R1
         Branch >0 LOOP
         Return               Returning to calling program
```

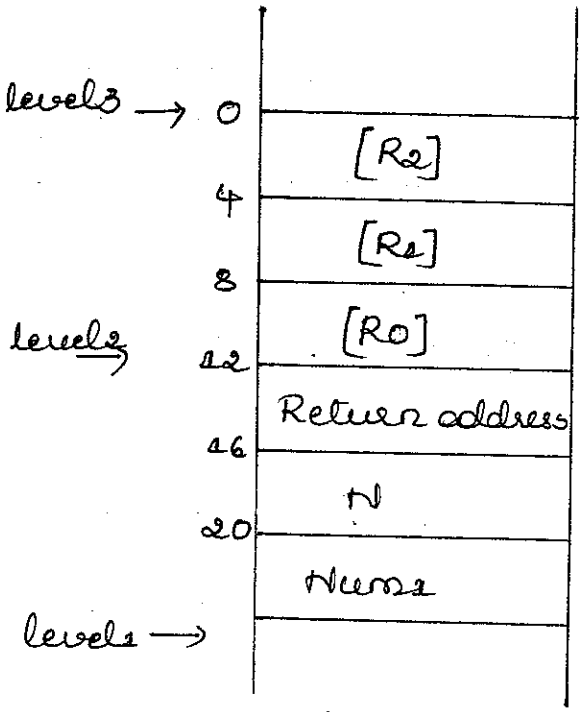
→ Subroutine ADDLIST adds the number Num2 to Num1.
→ Before the subroutine is called, value of N is stored on register R_1 and address of array (Num_1) stored on R_2.
→ In the subroutine ADDLIST all the numbers are added to R0.
→ On return to calling program the result is copied to sum.
→ In this program R_1 and R_2 are the parameters.

Disadvantage of this method is

1) Since general purpose registers are used for parameter passing any subroutine can access them. (They are like global variables which can be accessed by any function).
→ If calling program has some data on registers then any routine can modify that value (no security for data).
→ The number of general purpose registers are limited. So better option is to save the data on to the processor stack.

2) Parameter passing by using processor stack.

→ All the information needed by subroutine and the data that should not be altered by any of the subroutine will be placed on stack.
→ will consider a stack which is initially empty which means stack is at level1.
→ - the main program first the address of array and N is pushed on to the stack.
→ when call instruction executes return address is pushed on to the stack.
→ now we say that Stack is at level2.
→ The subroutine then pushes the contents of R0, R1 and R2 on the stack because it may be used by calling program. If the contents are not saved on stack then it can be modified by subroutine.
→ Now the stack is at level3.



Stack representation for the below program.

```
Move   #Num1, -(SP)     Push address of Num1 on to stack
Move   N, -(SP)         Push no of elements on to stack
Call   ADDLIST          call subroutine
Move   4(SP), SUM       Save result
Add    #8, SP           Restore top of stack (Level1)

ADDLIST  MoveMultiple  R0-R2, -(SP)   Save registers R0, R1, R2
         Move  16(SP), R1             Initialize R1 with N
         Move  20(SP), R2             Initialize R2 with address of
                                      Num1.
         Clear R0
         Add   (R2)+, R0
         Decrement  R1
         Branch>0 loop
         Move  R0, 20(SP)             Put result on to the stack
         MoveMultiple (SP)+, R0-R2    Restore registers.
         Return                       Return to calling
                                      program.
```

Stack Frame

→ when a subroutine is called parameters, return value, and local variables are pushed on to the stack.

→ Stack will contain the complete information of a subroutine, this is known as stack frame.

→ Consider a stackframe for the subroutine.

Initially stack will be at level 1. Parameters are pushed on to the stack. Next return address is pushed on to the stack.

→ Above the return address contents of Frame pointer is pushed on.

→ Frame pointer is one of the general purpose register.

→ General purpose register might be used by calling routine which will have some valid contents in that.

→ Since we dont want the contents to be modified by subroutine, save the contents of frame pointer on the stack and later on restore the contents of framepointer.

→ stack pointer will always point to the top of stack.

→ By using Frame pointer it is convenient to access the parameters passed to the subroutine and also the local variables used by subroutine.

→ Above the saved contents of FP local variables are pushed on to the stack.

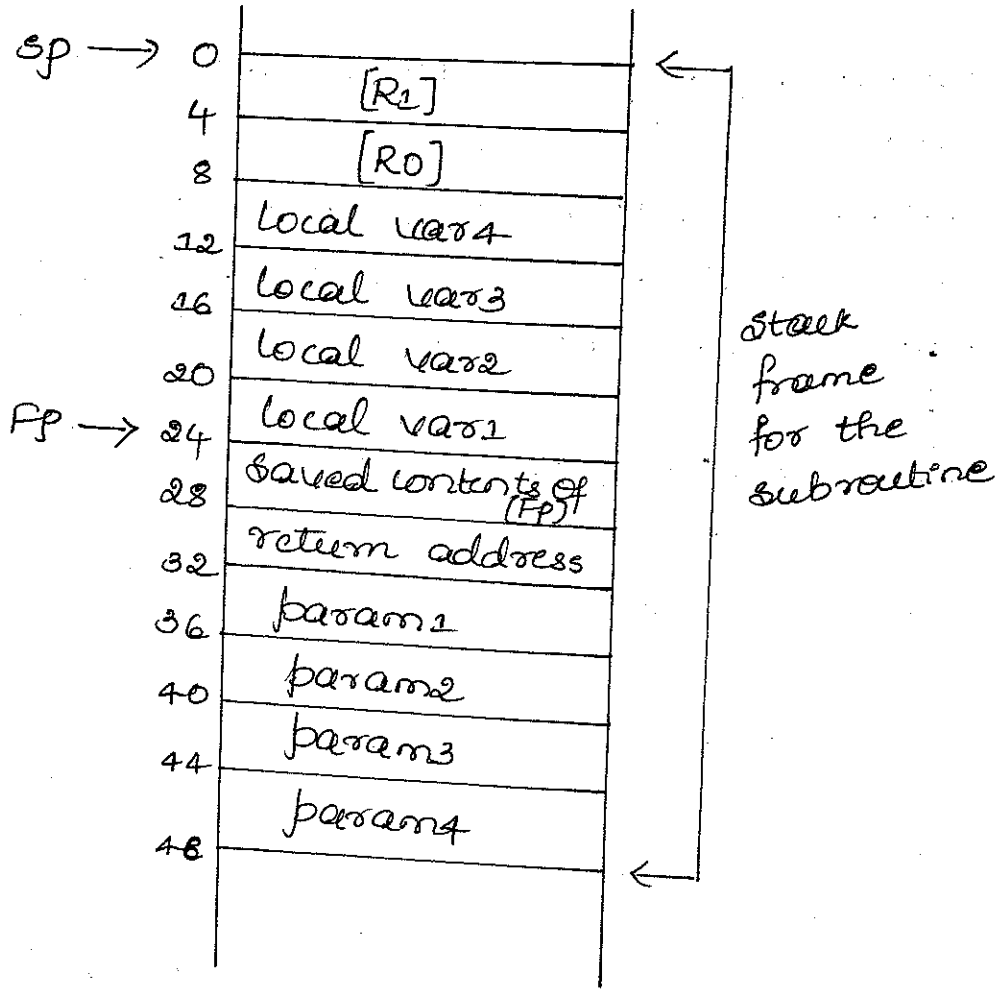
→ Even the contents of registers are stored on the stack, which will be used by calling program.

→ By using Frame pointer we can access the parameters using index addressing mode.

i,e 8(FP) = 32 . param1 is stored in the address 36
12(FP) = 36 param2 is stored in the address 40

→ local variables can be accessed in the following manner.

i,e −8(FP) = −8 + 24 = 16 localvar2 is accessed.



→ Contents of Fp are stored on the stack by the instruction

Move Fp, -(sp) → decrement sp & store the contents of Fp to sp

Move sp, Fp → contents of sp is moved to sp.

Consider an example of nested subroutine

Main program

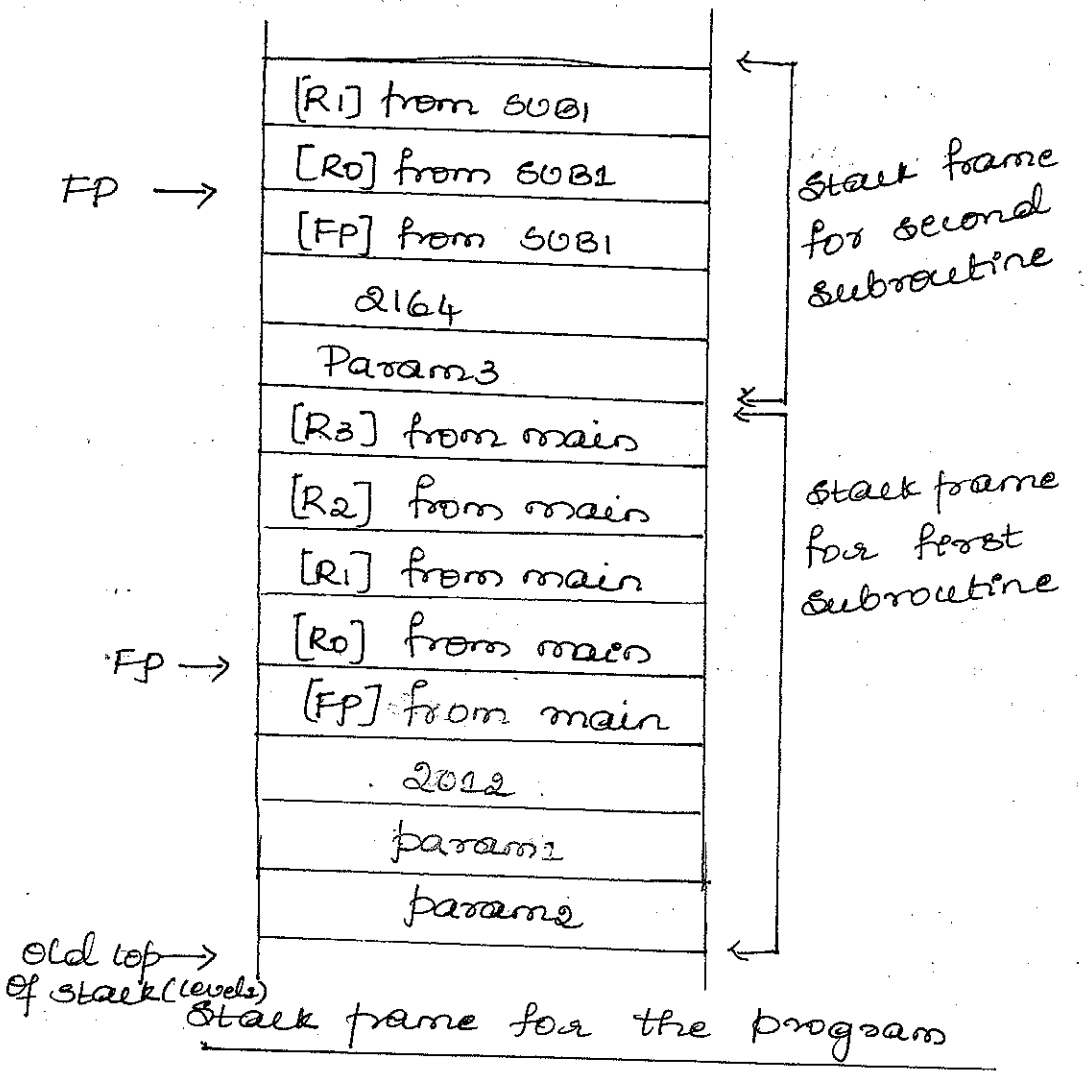
2000	Move	Param2, -(sp)	Place parameters on stack
2004	Move	Param2, -(sp)	
2008	call	Sub1	
2012	Move	(sp), Result	Store result
2016	Add	#8, sp	Restore stack level
2020	next	instruction	
	⋮		

First Subroutine

```
2100  sub1  Move  FP, -(SP)        Save contents of FP on stack
            Move  SP, FP           Load the address of SP to FP
2104        MoveMultiple R0-R3,-(SP)  Save the registers
2108        Move  8(FP), R0        get the first parameter
2112        Move  12(FP), R1       get the second parameter
              :
            Move  Param3, -(SP)    Place the parameter on stack
2160        Call  Sub2
2164        Move  (SP)+, R2        Pop sub2 result into R2
              :
            Move  R3, 8(FP)        Place answer on stack
            MoveMultiple (SP)+, R0-R3  Restore registers
            Move  (SP)+, FP        Restore the contents of
                                   frame pointer
            Return                 return to main program
```

Second Subroutine

```
3000  sub2  Move  FP, -(SP)        Save contents of FP on stack
            Move  SP, FP           load address of SP to FP
            MoveMultiple R0-R1, -(SP)  Save the register
            Move  8(FP), R0        get the parameters
              :
            Move  R2, 8(FP)        Place sub2 result on stack
            MoveMultiple (SP)+, R0-R1  Restore registers R0-R1
            Move  (SP)+, FP        Restore frame pointer
            Return                 Return to subroutine1
```



Additional Instructions

→ Along with the instructions Move, Branch, add and so on logical instructions are also available.

Logical Instructions

→ AND, OR and NOT are applied to individual bits.

1) **NOT:**

Not instruction complements the contents of an operand by making all zero's to one's and one's to zero's.

Ex:- NOT R0

Suppose R0 = 0011

Not R0 gives the value of R0 as 1100

Not operator is similar to 1's complement of a number

→ 2's complement of a number is obtained from 1's

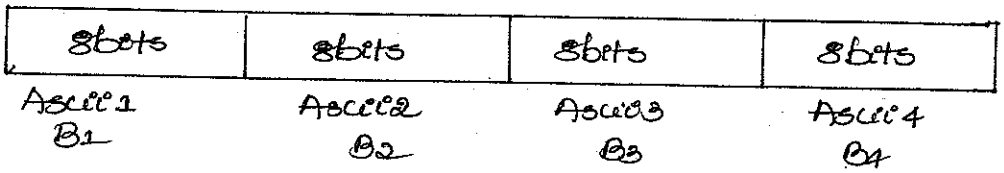
complement by adding 1 to it. This can be achived as follows.

```
NOT R0
ADD #1, R0
```

AND:

AND instruction is used to clear some bits.

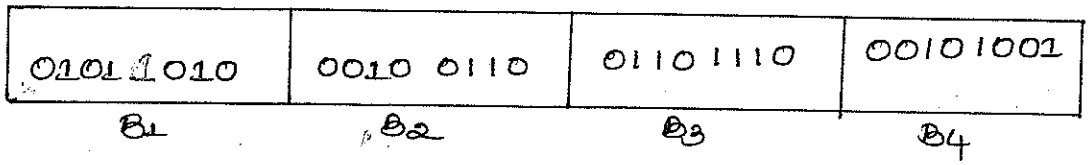
consider an example where R0 is a register representing 32 bit word. Since character occupies 1byte, 4 characters can be stored in 32bit word.



How to find out of the Byte B1 contains the character "Z"?

solution:- Hexa representation of Z is 5A and in BCD it is 0101 1010.

→ If register R0 contains Z then register R0 will look like



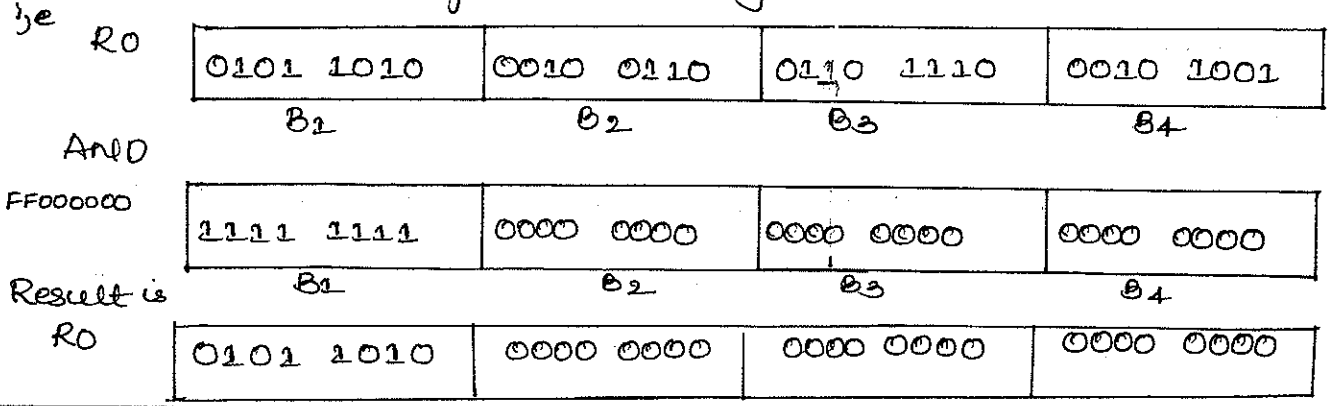
Byte B2, B3 and B4 can have any other characters.

→ TO test B1 we have to remove the existing content of by Byte B2, B3 and B4.

→ This is achived by using AND instruction.

```
AND #FF000000, R0
```

which will wipe off the bytes B2, B3 and B4 to zero



→ Next step is to compare R0 with the ascii value of Z
i,e compare #$5A000000 , R0

If the difference is zero then B1 byte of R0 has the character Z.

Code is as follows:- AND #$FF000000 , R0
Compare #$5A000000, R0
Branch =0 yes

Shift Instruction

→ There are two types of shifts
1) Logical shift.
   a) logical left shift
   b) logical right shift
2) Arithmetic shift
   a) Arithmetic right shift.

→ Instead of multiplication logical left shift can be used Because left shift is always multiply by 2.
→ Instead of division logical right shift can be used since right shift is divide by 2.

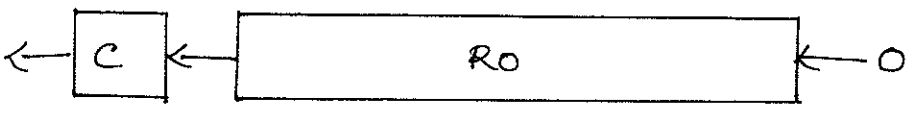
1) Logical left shift.

→ It is represented by the instruction:

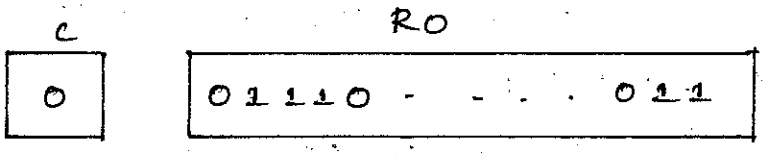
LshiftL count, destination.

→ LshiftL means logical shift left
→ count means the number of times leftshift has to be performed.
→ destination is the number to be shifted.

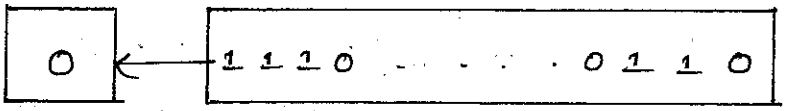
ex:- LshiftL #2, R0



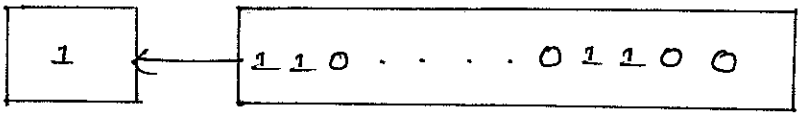
→ when LshiftL is done zero is padded to LSB and the bit at MSB is stored on carry flag and eventually dropped.



Lshiftl #2, R0



when shifted for 1'st time



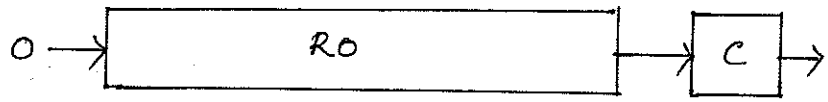
shifted for 2nd time.

Logical shift right

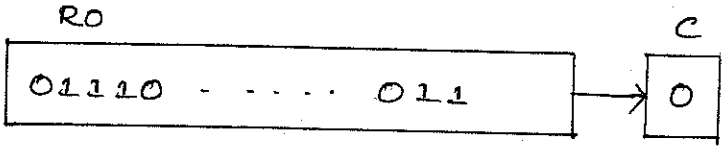
→ It is represented by the instruction

Lshiftr count, destination.

ex:- Lshiftr #2, R0

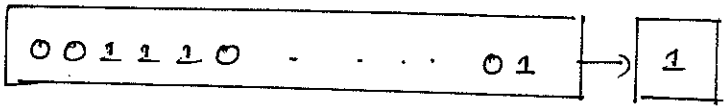


when logical right shift is done the bit at the lsb is stored in carry and eventually dropped and zero is padded to MSB

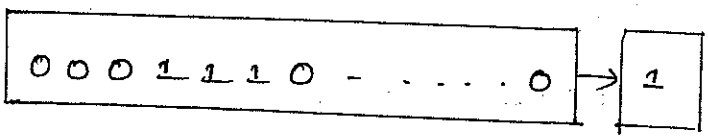


Lshiftr #2, R0

Shifted for the first time



shifted for second time



Digit packing

BCD stands for binary coded decimal.

There are two types of BCD

→ unpacked Bcd

→ packed BCD

unpacked Bcd

→ In upacked Bcd every decimal digit takes 8 bits

ex:- 4 when stored in memory looks like

8bits	8bits	8bits	0000 0100

2 when stored in memory looks like

8bits	8bits	8bits	0000 0010

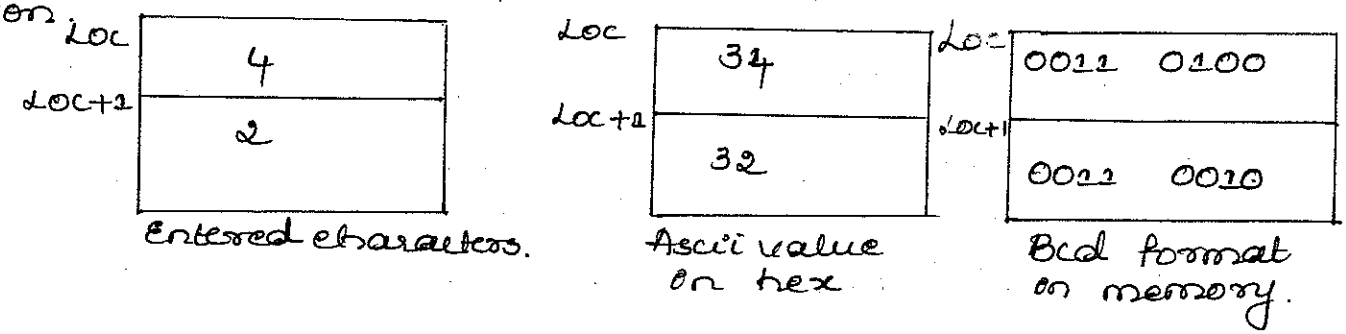
Packed Bcd

→ In Packed Bcd every decimal digit takes 4 bits.

Note :- Since 8 bits are required to represent one decimal digit in unpacked bcd and packed bcd requires only four bits to represent one decimal digit, Using packed bcd we can represent two digits using 8 bits.

Consider an example where two numbers are represented in upacked bcd stored at the memory Loc and Loc+1. we want to pack them onto a single memory location

Solution,



→ Two no's 4 and 2 are entered by the user. The ascii of these no's in hex is 34 and 32, which is stored in the memory as Bcd format.

In order to pack them into single memory location follow the steps.

1) Take the first no and left shift of four times.

i,e when we leftshift 0011 0100 four times we get

0100 0000

2) Take the second no and perform AND operation to clear higher order bits.

i,e 0011 0010

AND 0000 1111

0000 0010

3) Now perform OR operation with the result of step1 and step2

i,e

0100 0000

OR 0000 0010

0100 0010

4 2

This is how we packed two digits in 8 bits.

Assembly language code for digit packing

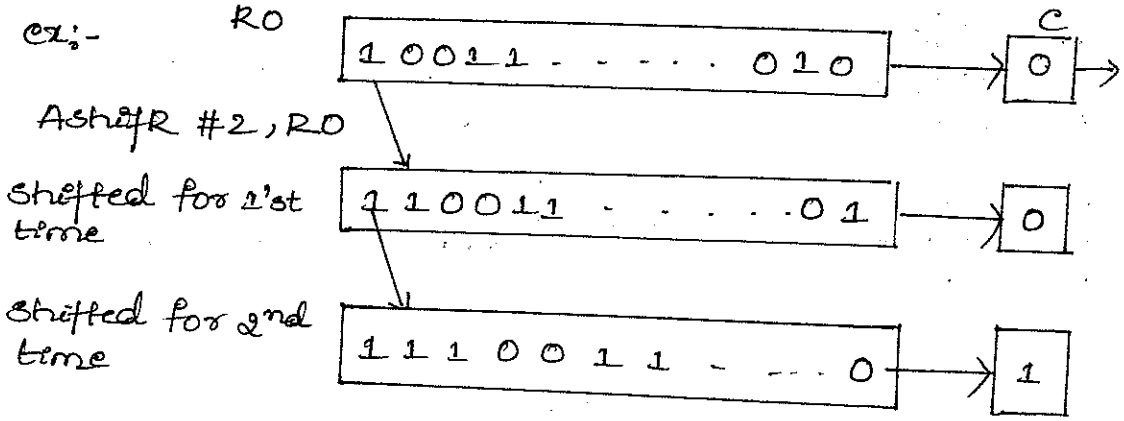
Move	#LOC, R0	R0 points to the memory where first no is stored.
Movebyte	(R0)+, R1	Load firstbyte (8bits) into R1
LshiftL	#4, R1	Leftshift R1 by fourtimes
Movebyte	(R0), R2	Load firstbyte (8bits) of second no into R2
AND	#0F, R2	Remove higher order bits of R2
OR	R1, R2	Concatenate BCD digits
Movebyte	R2, PACKED	Store the result into memory location named PACKED.

Arithmetic Right shift.

It ensures that sign of the number is maintained. It is same as logical right shift in case of a +ve no. As the bits are shifted zero is padded to MSB.

For a Negative number.

→ As we right shift the number n times the vacant places at MSB is filled with 1.



Rotate Instruction

→ There are two types of Rotate

1) Rotate without carry
   a) Rotate left
   b) Rotate Right
2) Rotate with carry
   a) Rotate left
   b) Rotate right

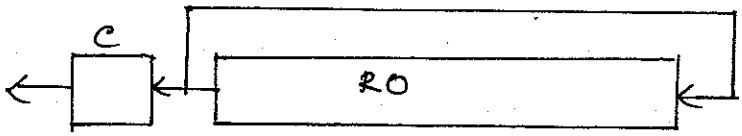
In rotate without carry, carry bit will not participate. The bit that will rotated will be stored on carry flag and eventually dropped.

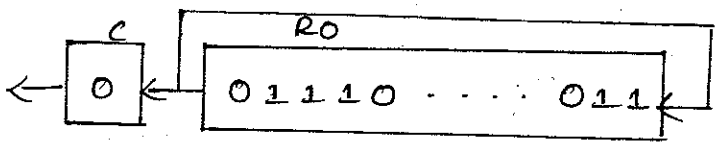
1) Rotate left without carry.

Instruction is as follows:-

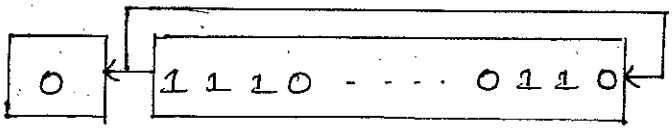
RotateL count, destination.

Ex:- RotateL , #2, R0

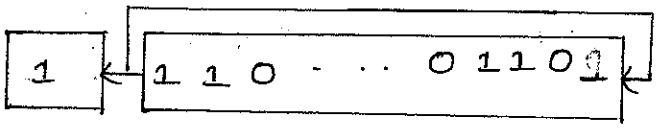




RotateL #2, R0



1st Rotation

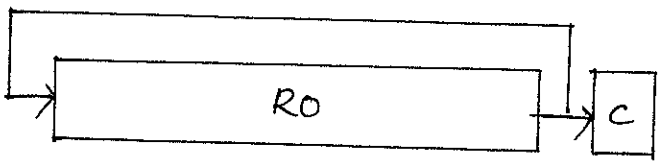


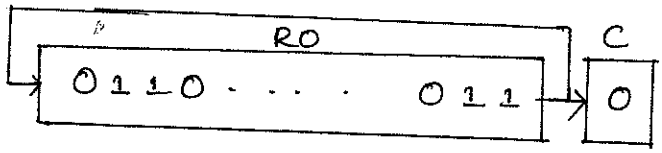
2nd Rotation.

2) Rotate right without carry.

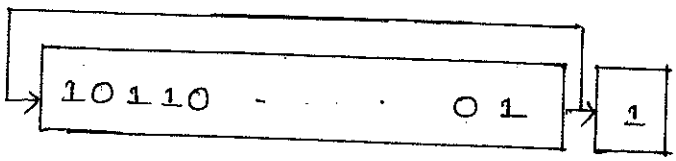
Instruction:- RotateR count, destination

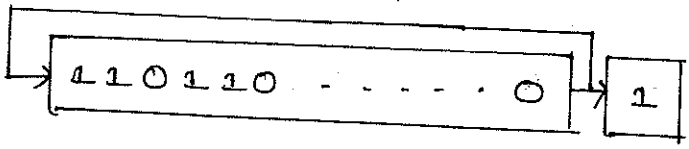
Example:- RotateR #2, R0





RotateR #2, R0



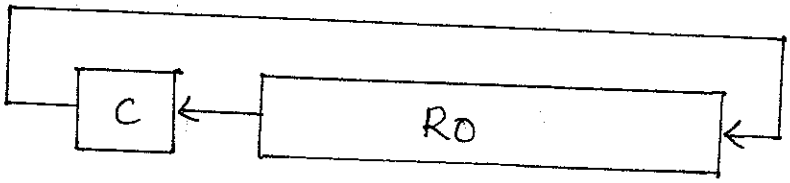


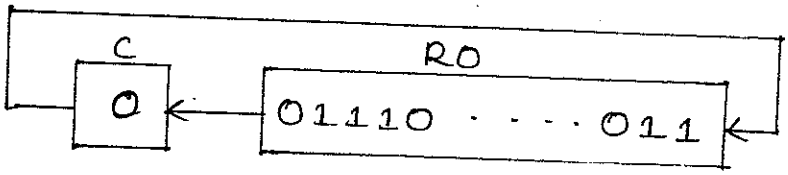
Right Rotate with carry.

→ Even carry bit will participate in rotation.

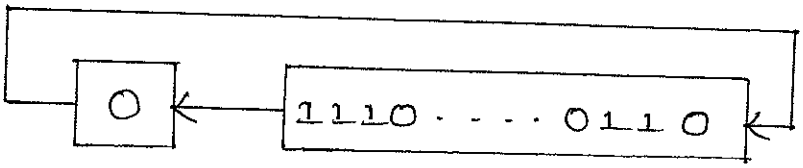
Instruction :- RotateLC count, destination

RotateLC → Rotateleft with carry

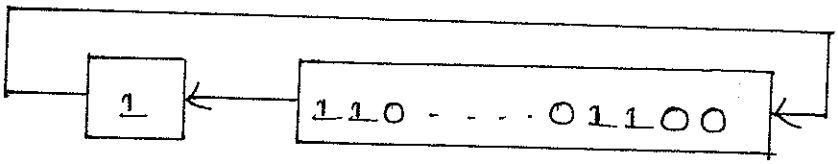




RotateLC #2, R0



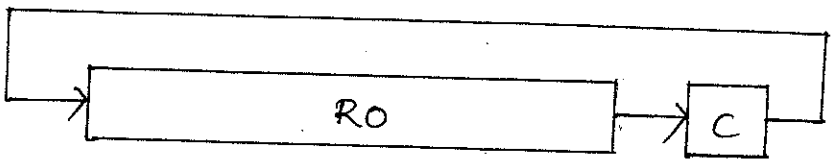
1st rotation

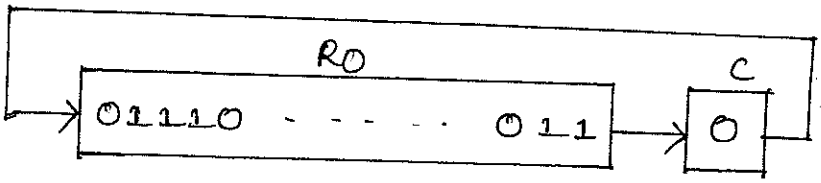


2nd rotation.

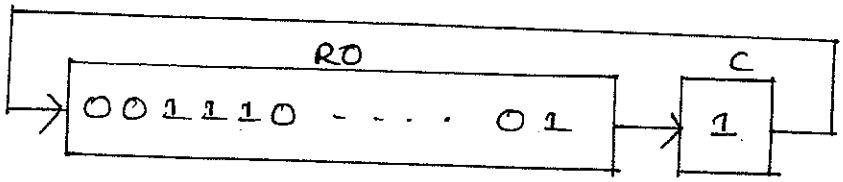
2) Rotateright with carry.

Instruction :- RotateRC count, destination

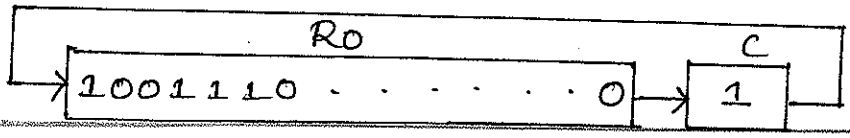




RotateRC #2, R0



1st rotation



2nd rotation

Encoding of Machine Instructions

Binary pattern of Assembly language instruction which will be stored on memory and gets executed is known as machine Instruction.

An instruction is made up of opcode, source, destination, and addressing modes of source and destination.

unless these opcodes, source, destination & addressing modes are converted to binary computer cannot understand the instruction.

specific identification will be given to all of these on...

→ suppose one bit is used then we can represent two opcodes.

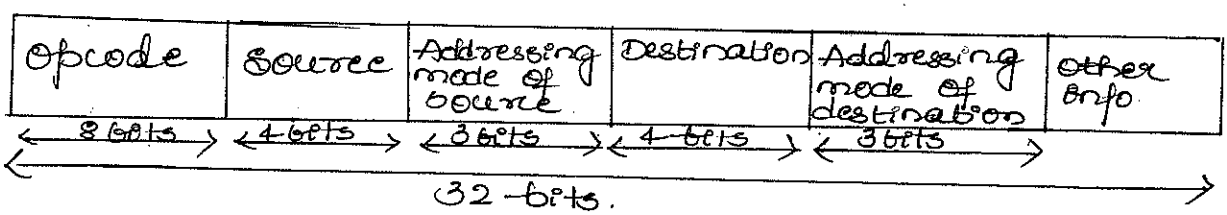
i,e 0 → move
1 → Add } example

→ If two bits are used then four opcodes can be represented.

00 → move
01 → add
10 → Branch
11 → call

→ To represent an opcode 8 bits are used.
i,e $2^8 = 256$ different opcodes can be represent using 8 bits.

For a 32-bit Architecture



Example 1>

Add R_1, R_2

→ Opcode Add takes 8 bits
→ To specify register R_1 (source) needs 4 bits
→ Addressing mode of R_1 needs 3 bits
→ To specify register R_2 (destination) needs 4 bits
→ Addressing mode of R_2 needs 3 bits
→ This complete instruction can fit on the memory.

→ Since there are 16 registers four bits are enough to specify the registers. Every register is represented using four bits.

Ex:- 0000 → R0
0001 → R1

→ 3 bits are used to specify addressing modes. using 3 bits 8 addressing modes can be represented.

ex:- 000 → Register
001 → direct

Example 2) can the instruction Move 24(R1), R5 be fit in the 32-bit word?

1) Opcode Move takes 8 bits
2) Register R0 takes 4 bits
3) Addressing mode of R0 takes 3 bits
4) Register R5 takes 4 bits
5) Addressing mode of R5 takes 3 bits

Bits assigned till now = 32 - 8 - 4 - 3 - 4 - 3 = 10 bits
the index value can be represented using 10 bits
Hence the instruction Move 24(R1), R5 can fit in 32-bit word.

Note:- Other information field is used for index and immediate values.

Example 3) can the instruction fit into 32-bit word?

a) LShiftR #2, R0

b) Move #$3A, R1

a) LshiftR takes 8 bits
R0 takes 4 bits
Addressing mode of R0 takes 3 bits
Addressing mode of #2 takes 3 bits

Remaining bits are 32 - 8 - 4 - 3 - 3 = 14 bits.
Immediate value i.e 2 can be represented using 14 bits.

b) Move takes 8 bits
R1 takes 4 bits
Addressing mode of R2 takes 3 bits
Addressing mode of #$3A takes 3 bits

Remaining bits are 32 - 8 - 4 - 3 - 3 = 14 bits
Immediate value 3A can be represented using 14 bits

Hence the instruction can fit on 32-bit word.

Example 4) Branch >0 Loop

Opcode Branch>0 takes 8 bits.
Offset Address ("Loop") needs how many bits?
32 - 8 = 24 bits remaining
∴ Offset is found by relative addressing mode by assembler itself. Represented as -x+[PC].
where x is the offset.
Since offset is always negative one bit is used to represent sign and remaing bits are used to represent offset.
So using 23 bits offset can be represented.
Instruction Branch>0 fits on 32-bit word.

Example 5) Move R0, Loc

Opcode Move takes 8 bits

R0 takes 4 bits

Addressing mode of R0 takes 3 bits

Addressing mode of Loc takes 3 bits.

Remaining bits are 32 - 8 - 4 - 3 - 3 = 14 bits

14 bits are not sufficient to represent a memory location.
32 bits are required to represent a memory location.
Hence the instruction requires two words of memory.

Opcode	Source	Addressing mode of source	Destination Addressing mode	other info
destination (memory) address				

Example 6) Add #$FF000000, R2

Add takes 8 bits

Addressing mode of #$FF000000 takes 3 bits

R2 takes 4 bits

Addressing mode of R2 takes 3 bits

remaining bits are 32 - 8 - 4 - 3 = 14 bits

FF000000 cannot fit in 14 bits so next word of memory is use.

This is an two word instruction.

Example 7) Move Loc1, Loc2.

Loc1 needs 32 bits

Loc2 needs 32 bits

Opcode move needs 8 bits

Addressing mode of Loc1 and Loc2 needs 3 + 3 = 6 bits

This is an example of three word instruction.

Problems

1) Register R5 is used to point the top of stack. Write a sequence of instructions using index, Auto increment and auto decrement addressing modes to perform each of the following task.

a) Pop the two top items off the stack, add them and push the results on to the stack.

b) Copy the fifth item from the top into register R3

c) Remove the top ten items from the stack.

Solution:-

a) Move (R5)+, R0
Add (R5)+, R0
Move R0, -(R5)

b) Move 16(R5), R3

c) Add #40, R5

2) Both the following statements causes the value 300 to be stored in location 1000, but at different times

ORIGIN 1000
DATAWORD 300

and

Move #300, 1000

write the difference.

→ The assembler directive ORIGIN and DATAWORD causes the assembler to place the value 300 in the location 1000 at the time program is loaded into memory before execution.

→ Move instruction places 300 into memory address 1000 when the instructions are executed as a part of program.

3) A program contains 1000 instructions. out of that 25% of instructions requires 4 clock cycles, 40% instructions requires 5 clock cycles and remaining requires 3 clock cycles for execution. Find the total time required to execute the program running in a 1GHz machine.

Instruction Type	Instruction Count	Cycles Required
Type 1 (25% of 1000)	250	250 * 4 = 1000
Type 2 (40% of 1000)	400	400 * 5 = 2000
Type 3 (Remaining 35% of 1000)	350	350 * 3 = 1050
	Total clock cycles =	4050

Time required to execute the program, $T = \frac{N * S}{R}$

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

Note:- Every basic step is computed in 1 clock cycle.

Hence Average number of clock cycles to execute one instruction $S = 4050 / 1000 = 4.05$ clock cycles.

$$T = \frac{1000 * 4.05}{1 * 10^9}$$

$$T = 4.05 \mu sec$$

4) write three-address, two-address and one address instructions for the following expressions.

a) $Z = A + BCx + D$

b) $Y = Ax^2 + Bx + C$

a) $Z = A + BCx + D$

One-Address	Two-address	Three Address
Load B	Mult B, C	Mult B, #x, Y
Mult #x	Mult #x, C	Mult C, Y, C
Mult C	Add C, D	Add C, D, E
Add D	Add D, A	Add E, A, Z
ADD A	Move A, Z	
Store Z		

b) $Y = Ax^2 + Bx + C$

One-Address	Two-address	Three-address
Load A	Mult #x, B	Mult #x, A, A
Mult #x	Mult #x, A	Mult #x, A, A
Mult #x	Mult #x, A	Mult #x, B, B
Store Z	Add A, B	Add A, B, E
Load B	Add B, C	Add C, E, Y
Mult #x	Move C, Y	
Add Z		
Add C		
Store Y		

Difference between RISC and CISC Architecture

CISC	RISC
1) CISC stands for complex instruction set computer	RISC stands for Reduced instruction set computer
2) It has large amount of different & complex instructions	It has few instructions
3) Slow compared to RISC	fast compared to CISC
4) CISC machines execute an instruction in 2-10 machine cycles	RISC machines executes an instruction in 1 machine cycle
5) used in computers that has wider range of instructions to execute	used in faster processing like games.
6) Ex:- 8086 processor	Ex:- Arm processor

Medha Gourayya
Asst Prof, CSE
RNSIT

Input output organization

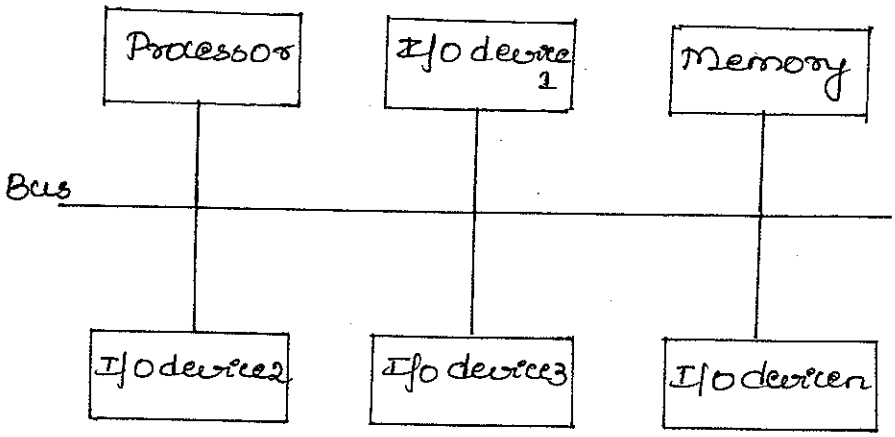
II Module

Accessing Input/output devices

Computer is connected to input output devices by set of lines or wires called as Bus.

Bus enables the exchange of information between computer and input output devices.

The figure shows single bus architecture where input output devices are connected to a computer.



A bus is made up of three sets of lines or wires. They are;-

1) Address lines :- (Unidirectional)
   Address lines carries the address of memory or input output devices from processor.

2) Control lines :- (Bidirectional)
   Control lines carries the signals to memory or input output devices from processor.
   It also carries signals from input output devices to processor in case of interrupts.

3) Data lines :- (Bidirectional)
   Data lines carries data from processor to memory or input/output devices, or from memory or input/output device to processor.

→ Every input output device has a unique address.

→ Whenever processor wants to communicate with any of the device, it places the address of device on address bus.

→ The device which recognizes it address, identifies the signal on control line (read or write signal)

→ Based on the signal on the control line the requested data is transferred on the data lines.

Memory mapped Input output

→ It is the arrangment where the main memory and input / output devices (buffer registers) share the same address space.

→ i.e chunk of main memory is assigned or mapped to keyboard and display.

→ when data is read or written to this memory it is as good as reading from keyboard and writing to display.

Ex:- Move DATAIN, R0

Since DATAIN is mapped to keyboard, we say that reading from keyboard.

Ex:- Move R0, DATAOUT

DATAOUT is mapped to display, we say that writing on to display.

I/O mapped Input output

→ It is used in some computers

→ Has seperate memory for input and output.

→ special instructions like IN and OUT are used to perfor input output operations.

Input / output Interface circuit.

The hardware required to connect input ofutput device to bus is known as input/output interface circuit

→ It consists of address decoder, control circuit, data and status register.

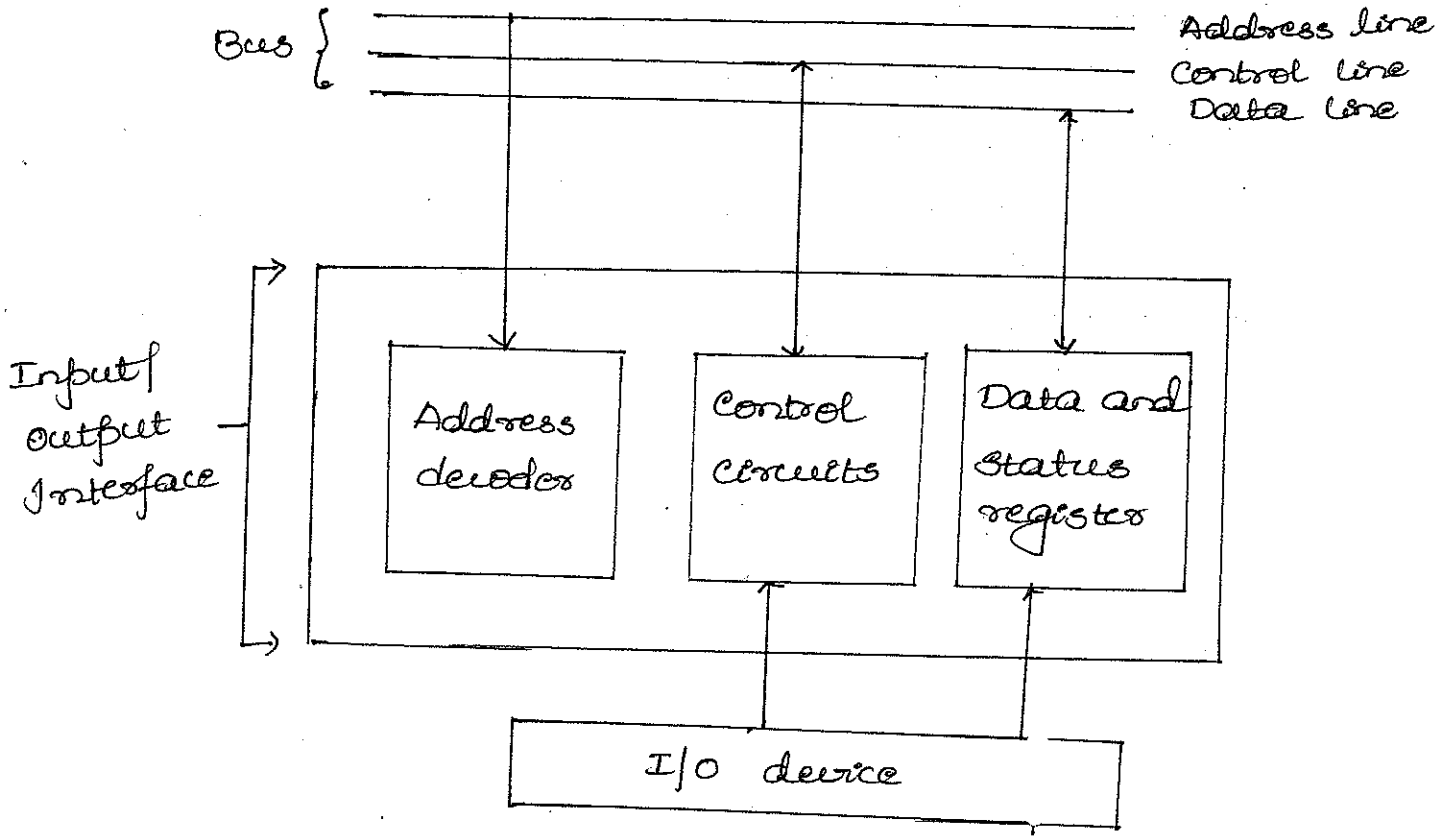
→ Every Iput or output device has an Io interface circuit.

→ The address decoder decodes the address on the address line which is placed by processor. If the address matches with the address of the device connected to it then the addressed device responds to the processor request.

→ Data register holds the data being transferred to or from the processor.

→ Status register contains the information which helps the input output device to perform data transfer operation.

→ Control circuit senses the signal read or write operation.



The speed of the processor is very high compared to the speed of input output device. So there are some methods to synchronize the timing difference between processor and input/output device.

There are three methods for this purpose

1) Program controlled Input output

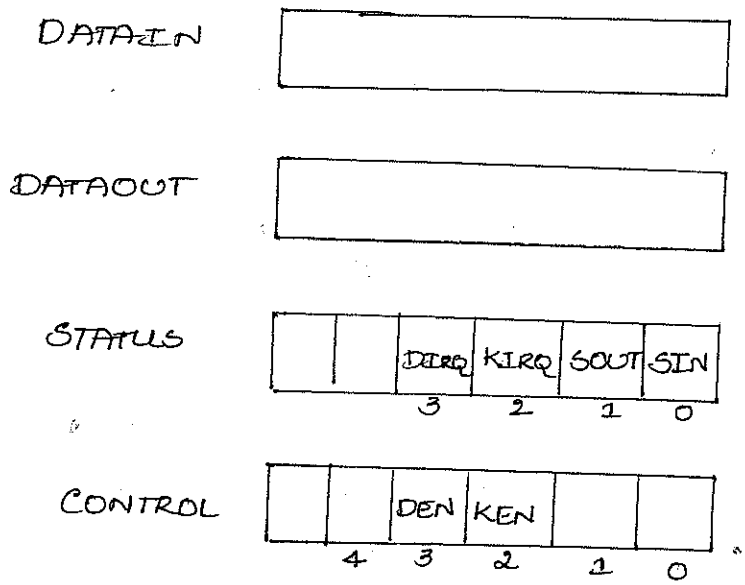
2) Interrupt

3) Direct memory Access (DMA)

Program controlled Input/output

Consider an example program where the characters are read from keyboard until a newline is encountered and echoed on the screen.

	Move #Line, R0	Initialize memory pointer
WAITK	Testbit #0, STATUS	Test SIN bit of STATUS register
	Branch =0 WAITK	wait for character to be entered
	Move DATAIN, R1	move the character to R1
WAITO	Testbit #1, STATUS	Test SOUT bit of STATUS register
	Branch =0 WAITO	wait for display to become ready
	Move R1, DATAOUT	Send the character to display
	Move R1, (R0)+	store the character in memory and increment the pointer to point next address
	Compare #$0D, R1	check of carriage return
	Branch ≠0 WAITK	If not then read another character
	Move #$0A, DATAOUT	else send line feed (ln) to DATAOUT
	Call PROCESS	Call the subroutine Process



Registers in keyboard and display interfaces.

→ In the above program processor will be waiting for a valid character to be entered by user.

→ In that case sin flag will be zero.

→ When user enters a character sin becomes 1. Now processor will read that character into processor register.

→ In the same way processor will wait until device device becomes free. i,e sout flag becomes 1.

→ When sout is 1 processor sends the character to DATAOUT buffer.

→ In both the cases processor wastes its valid time in checking sin and sout bit of status register.
→ This method is known as program controlled Input output.

2) Interrupts.

In program controlled input output processor wastes its valid time. So next mechanism can be used where processor will not wait for SIN and SOUT bit to set. Instead when the Input output device becomes ready it informs the processor that it is ready for data transfer by sending the hardware signal on the control line.

This signal sent by Input output device to processor is known as Interrupts.

Interrupt is sent using control bus. One line in control bus is dedicated for this purpose called as Interrupt request line.

Consider an example with two routines COMPUTE and PRINT
→ COMPUTE routine performs some computations and produces n lines to be printed by PRINT routine.
→ COMPUTE will repeatedly performs computation and produces n lines to be printed by PRINT routine.

Scenario in program controlled Input output.

→ When COMPUTE produces n lines it sends to PRINT routine.
→ PRINT routine sends one line at a time to printer.
→ Until printer finishes printing processor has to wait for printer to become free that it can send second line to printer.
→ When all the lines are printed control is transferred to compute routine and it starts executing.
→ Hence processor time is wasted in waiting for printer to become ready.

Interrupt Method

→ Compute routine sends n lines to PRINT routine.

→ PRINT routine sends one line to Printer. Instead of processor waiting for Printer to finish printing processor suspends PRINT routine and control is transferred to COMPUTE.

→ COMPUTE will resume its activity of computation.

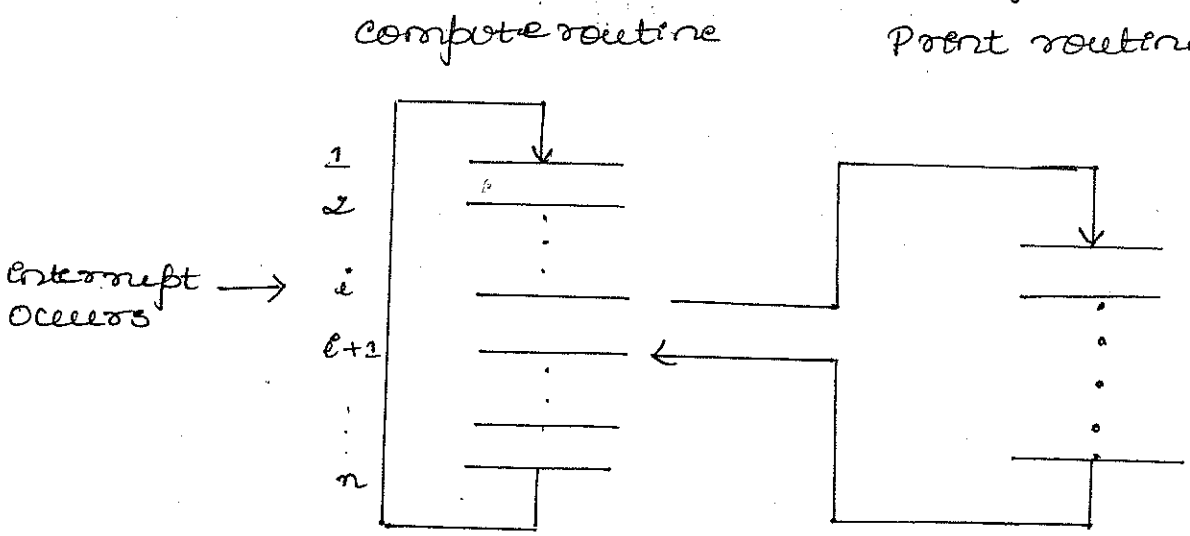
→ Once Printer finishes printing it informs processor saying that it is ready for data transfer by sending an interrupt.

→ Now the processor suspends COMPUTE and moves to PRINT routine and sends second line to Printer.

→ Once again PRINT routine is suspended.

→ These steps are repeated until all the lines computed by COMPUTE routine gets printed.

→ Routine that executes when an interrupt request is made is known as Interrupt Service Routine (ISR). In the above example PRINT is the Interrupt Service Routine.



control transfer when an interrupt occurs.

→ Assume processor is executing an instruction i when an interrupt occurs.

→ Processor completes the execution of instruction at address i, saves the contents of PC (i+1) in stack.

→ PC is loaded with the address of Interrupt service routine.

→ Now control is transferred to Interrupt service routine.

→ On return from Print routine, return is poped from stack loaded on to PC and execution of compute resumes.

Difference between Interrupt Service Routine and Subroutine

→ ISR and Subroutine call looks similar i,e PC is pushed on stack and ISR is placed into PC, on return from ISR, stored PC value is popped back onto PC.

→ Difference is that when a subroutine is called all the register values are saved before calling the subroutine.

→ When an interrupt service routine is called, then only two register values are saved.

1) PC

2) Status register.

The reason why processor does not store the contents of all registers is that, it takes some time to store these values meanwhile ISR should wait. This delay to serve Interrupt service routine is known as Interrupt latency.

Solution to avoid this is either the ISR, programmer can write code to save all register values and just before exiting the ISR restore all register values.

Different methods of handling latency on different Processor Architecture.

1) In some processor architecture the number of registers on the system is very less. In this case processor will store the values befor ISR.

2) Some architecture provides two special instruction. One to save the register another to restore the register.

3) Processor supports two sets of registers. One for normal processing and another duplicate set for Interrupt handling. So for the interrupts duplicate set of registers are used. Hence there is no need to store and restore registers.

Interrupt Hardware

Multiple Input/output devices can be connected to the computer.

→ All the input/output devices are connected to computer via switches to ground.

→ There is a common interrupt line which serves 'n' I/O devices connected to computer.

Interrupt handling works in the following manner.

1) When none of the I/O devices raises an interrupt all the switches are open. This means entire Vdd(1) passes on the line. When it reaches the processor because of NOT gate, processor gets 0. Processor understants that there is no interrupt from any device.

2) As soon as an I/O device raises an interrupt (to indicate key has been typed on keyboard or display is ready to recieve next character), the switch associated with I/O device gets closed.

→ As a result 0 voltage is passed along the line. When it reaches the processor because of a NOT gate processor recieves 1. Hence processor understands that an interrupt has occured.

→ The value of INTR is a logical OR of the request from individual devices.

$$INTR = INTR1 + INTR2 + \cdots + INTRN$$

→ Interrupt is represented in complementary form i.e $\overline{INTR}$.

→ This is because when there is no interrupt line carries 1 (Vdd) and when there is interrupt line carries 0.

→ INTR :- Interrupt Request Line

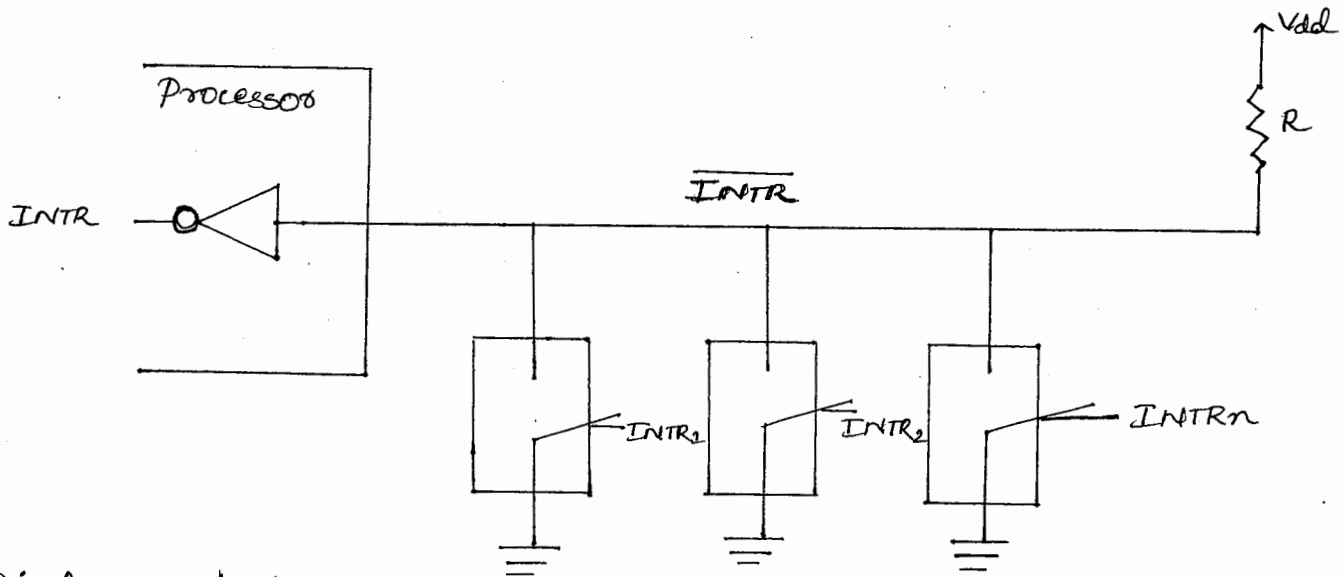


Fig :- An equivalent circuit for an open-drain bus used to implement a common interrupt request line.

When an interrupt occurs, the processor suspends the execution of current program and begins the execution Interrupt Service Routine (ISR).

→ Since Interrupts may arrive at any time some problems may occur.

1) ISR may change some of the data required by the processor for later use i,e after returning from ISR.

2) Second problem is it may also happen that, processor is not ready to serve the requested device.

Another major problem is how long the interrupt request signal must be active?

For this consider a case of a single request from one device.

→ When a device raises an interrupt by activating interrupt request line, it keeps the line active until it recieves an acknowledgment from the processor saying that your request has been accepted.

→ This means that the line line is active during the execution of ISR, Because of which processor thinks that once again an interrupt has occured and this continues. i,e Interrupt request will make successive interruption, causing the processor to execute the same ISR and thus making the system to enter an infinite loop. This is known as Recursive Interrupt Problem.

To overcome this Recursive Interrupt Problem, there are three methods to enable and disable interrupts.

Method 1 :- The processor hardware must ignore the interrupt request line until the execution of the first instruction of ISR has been completed.

→ The first instruction in ISR must be an interrupt disable instruction. Hence no further interrupts are allowed.

→ Once the ISR completes its execution the last instruction should be Interrupt enable instruction.

→ So after completion of ISR further interrupts may occur.

→ next instruction after enable instruction will be return instruction which causes the processor is resume the suspended program.

Method 2:- Processor automatically disables the interrupt before starting the execution of ISR.

→ For this processor has a register known as processor status (PS) register.

→ One bit on PS register is called as Interrupt enable bit i,e IE bit which is used to indicate whether interrupt is enabled or disabled.

→ When IE bit is set to 1, processor is ready to allept an interrupt (Interrupt enable).

→ When IE bit is set to 0, processor is busy serving another device.

In this method before the execution of ISR begins, processor sets the IE bit to 0, pushes the return address and IE bit on to the stack. Thus disabling further interrupts.

→ When a return from ISR is executed the processor restores the content of PS and PC from the stack and sets IE bit to 1. Hence interrupts are enabled.

Method 3:- The processor has a interrupt request line for which the interrupt handling circuit responds only to the leading edge of the signal. Such an interrupt is called as edge triggered.

→ In this case, processor recieves only one request when the interrupt line is going high.

→ Hence processor recieves one request regardless of how long the line is active.

→ So there wont be multiple interrupts and there is no need to explicitly enable and disable interrupts.

Note:- In method1 and method2 interrupts are enabled and disabled using IE bit of processor status register.

Handling Multiple Devices.

→ When multiple devices are connected to a single interrupt request line, many issues arises since all the devices can raise an interrupt at any random time.

The issues are:-

1) How will the processor identify the device raising an interrupt?

2) When the processor is serving device A, should another device be allowed to raise an interrupt?

3) How should the processor handle multiple request which has seperate INTR and INTA lines?

4) How to handle simultaneous Request?

Solution to the above issues is as follows:-

1) How will processor identify the device raising an interrupt?
Ans:- 1) Polling 2) Interrupt Vector.

Polling:- All the I/O device will have status register.
→ One bit in the status register is Interrupt request bit ie IRQ bit.
→ KIRQ is keyboard interrupt request bit.
→ DIRQ is display interrupt request bit.
→ All the I/O device raises an interrupt by setting the IRQ bit.
→ In polling method, processor has a special interrupt service routine which polls (checks) all the I/O device status register.
→ The first device encountred with IRQ bit set will be served (Thinking that particular device has raised interrupt).

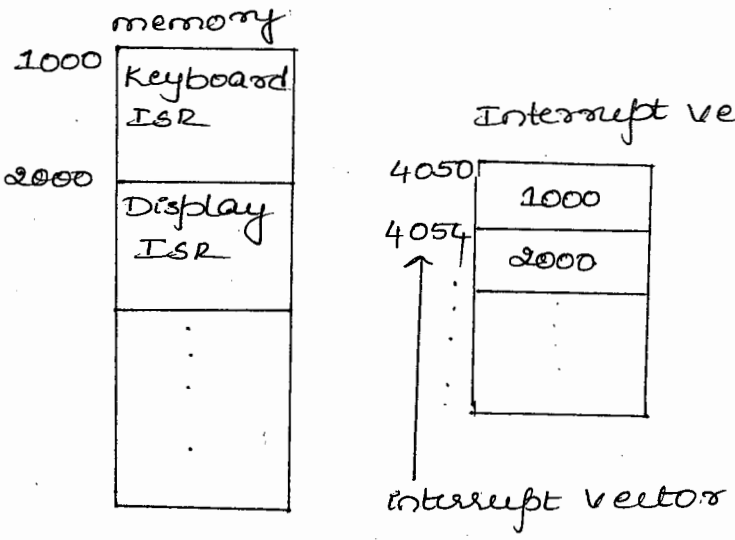
Advantage
→ Polling method is simpler and easy to implement

Disadvantage
→ Processor spends lot of time in checking all the device status register.

Interrupt Vector
To overcome the limitation of polling scheme, Interrupt Vector is used where processor need not check whether the device an interrupt or not. Instead the device which raises an interrupt will identify itself to the processor by sending some identification code to the processor.

→ This identification code is an 4 bit to 8 bit number.
→ The code contains an address through which the staring address of ISR can be obtained (Like an Indirect Addressing mode)

→ The ISR for I/O devices are stored in the memory.
→ Code sent by the I/O device is known as interrupt vector.



For example the code sent by I/O device is 4050, this is known as interrupt vector.
→ This interrupt vector causes the appropriate address to be loaded into PC and ISR to be executed.
→ Since 4050 was sent by the device, the value at 4050 i.e 1000 gets loaded into PC and keyboard ISR gets executed.
→ So Interrupt vector will give you the address through which we can find where the appropriate ISR is stored in the memory.

2) When the processor is serving device A, should another device be allowed to interrupt?
Ans:- yes. Interrupt Nesting.
→ During the execution of ISR, the interrupts are disabled till the processor completes its execution. Hence till the device which has interrupted first is served, no other devices are allowed to request interrupt.
→ Many times it may happen that the requesting device must be served immediately without making them to wait until processor becomes free.
→ Errors will occur if some devices are made to wait.
For example, Computer keeps track of the time of the day using a real time clock.
→ This device sends the interrupt to processor at regular time interval to increment the seconds, minuts, hrs.
→ When processor recives the interrupt it executes an ISR to increment a set of counters in the memory that keep track of time in seconds, minuts and so on.

→ To ensure that the clock is displaying proper time to the user, the real time clock's request must be accepted by the processor during the execution of the ISR for another device.

→ This means that instead of disabling the interrupts, all the I/O devices can be given priority and served accordingly.

→ So an interrupt request from high priority device must be accepted while processor is serving another device of low priority.

3) How should the processor handle multiple request?

Ans:- Multilevel priority Implementation.

→ In this method all the devices are assigned some priority.

→ All the devices has got individual Interrupt request line (IRQR) and Interrupt acknowledgment line (INTA)

→ Interrupt requests received on the lines is sent to the priority arbitration circuit in the processor.

→ A request is accepted only if it has a higher priority than that currently assigned to the processor.

→ The priority of the processor is nothing but the priority of the device that is served.

→ When a request comes from a device, the arbitration circuit in the processor compares the priority of processor with that of priority of requesting device.

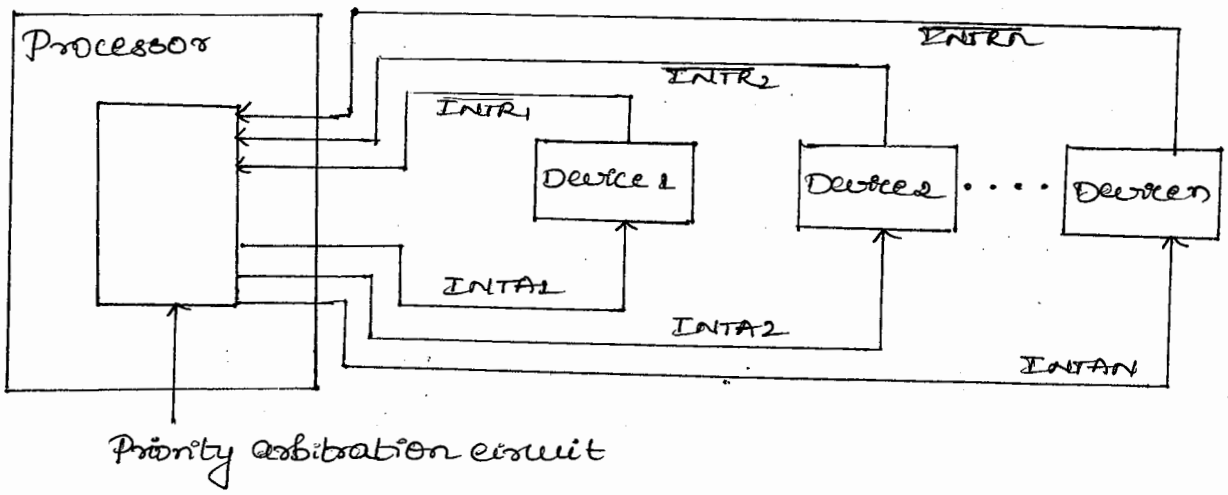
→ If the priority level of processor is high then processor continues its execution.

→ Suppose priority level of requesting device is higher than the processor then processor suspends the current program and starts the execution of ISR of requesting device.

Disadvantage:-

→ The device which has the lowest priority may not be served at all.

Fig:- Multilevel priority implementation.



4) How to handle Simultaneous Request?

Ans:- Daisy chain.

If multiple devices are connected to single interrupt request line then Simultaneous request are handled using daisy chain.

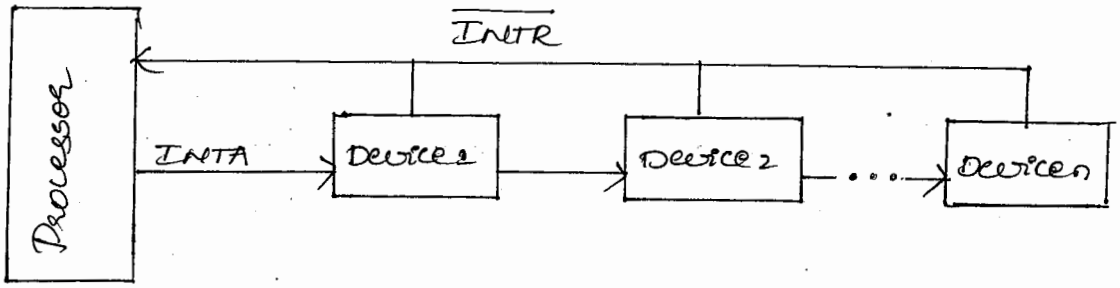


Fig:- Daisy chain arrangement

→ In daisy chain arrangement all the devices are connected to common Interrupt request line.

→ Interrupt acknowledgment line (INTA) is connected to the devices in a daisy-chain fashion such that the INTA line propagates serially through the devices.

→ When one or more devices raises an interrupt, $\overline{INTR}$ is activated.

→ The processor responds to these request by sending an interrupt acknowledge signal to the devices through INTA.

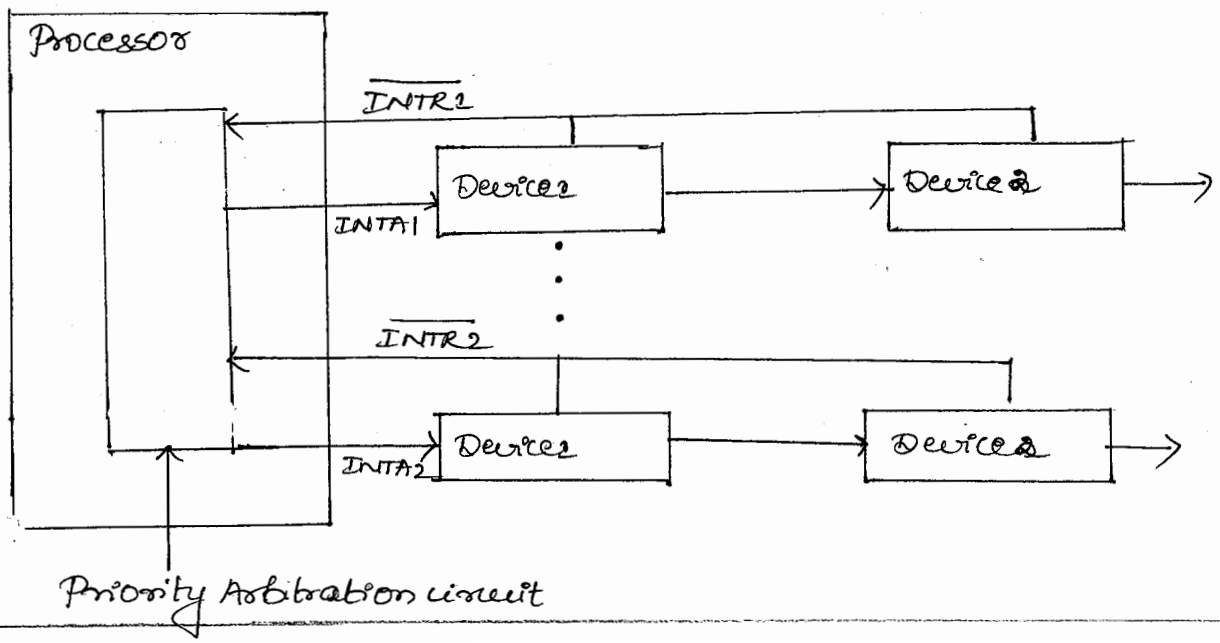
→ Acknowledgment signal starts propagating from device1 to devicen.

→ First device1 recieves INTA signal. Device1 passes this signal to next device i.e device2 only if it doesnt require an service from processor (i.e if device1 has not raised an interrupt)

→ If devices has raised an interrupt then it blocks the INTA signal. Hence INTA signal dosent propagate further.
→ After blocking the signal device puts its identification code on the lines so that processor understands which device has blocked the signal.

Disadvantage
→ The device which is far from the processor may not be served at all.

The limitation of Daisy chain and Multilevel priority implementation can be overcome to a certain extent by priority groups.
→ This is a combination of both Daisy chain and Multilevel priority implementation method.
→ In this method devices are grouped and every group is assigned with a priority.
→ All the devices in a group are arranged in daisy chain fashion.
→ When simultaneous request comes, arbitration circuit checks the priority of the group.
→ If the priority level of the group is higher than the priority of the device getting served then processor suspends the execution of current program and an appropriate ISR is executed.
→ Now to identity which device has raised an interrupt in the group, daisy chain methodology is used.



Exceptions

There are two types of interrupt. Interrupt raised by I/O devices is known as Hardware interrupt. Interrupt occured during the execution of a program is known as Software interrupt.

→ Exception is an software interrupt which alters the normal flow of program execution.

→ When an exception occurs the execution of current program is suspended and ISR begins executing.

Categories of exception.

1) Recovering from errors
2) Debugging
3) Privileged Exception

1) Recovery from errors

→ An error checking code is included in the main memory, which allows the detection of errors in the stored data.

→ If an error occurs the control hardware detects and informs the processor by raising an interrupt.

Ex:- if an attempt is made to divide a no by zero or if the opcode field is incorrect.

2) Debugging

→ System software provides a program called as debugger to find the errors in the program.

→ To debug the program, debugger provides two important facilities.

1) trace
2) Breakpoints

→ When processor is running in trace mode after the execution of every instruction an exception occurs because of which an exception service routine will be called. (Debbugging program is the Exception service routine).

→ Exception service routine enables the user to examine contents of memory, registers, accumulator and so on.

→ On return from the exception service routine, next instruction in the debugging program gets executed.

Breakpoints

→ Breakpoints is also same as trace, except that the program is interrupted at specific points selected by user.
→ For this there is a speial instruction called as trap.
→ While debugging a program user wishes to interrupt a program after the execution of instruction i.
→ i+1 will be saved on stack and PC is loaded with the exception service routine.
→ This enables the user to view the contents of memory, register etc.
→ On return from exception service routine contents from stack are restored and next instruction gets executed.

Privileged Exception

→ Computer works in two modes 1) user mode
2) System mode.
→ All the application programs run under user mode.
→ All the operating system routines run under system mode.
→ When system is in user mode, if any attempt is made to run OS routines will cause an exception known as privileged exception.
→ For example, trying to change the priority of processor, or trying to access computer memory will lead to privileged exception, which causes the processor to switch to the supervisor mode (system mode) and begin executing an appropriate routine in the operating system.

Direct Memory Access

→ We have seen how data transfer takes place between I/O devices and processor. An example for such a data transfer is MOVE DATAIN, R0
→ For such a datatransfer processor checks the status flag of all the devices. (i.e it checks SIN & SOUT flags) and then performs data transfer.
→ In this method processor wastes its time in waiting for I/O device to become ready for data transfer.

→ To overcome this interrupts are used. The device when it becomes ready for data transfer informs the processor by raising an interrupt.

→ When interrupts are used there is an additional overhead of storing and restoring the program counter and state information.

→ Hence Polling and interrupts are better when ample of data has to be read or written.

→ So when a chunk of data has to be read or written to or from memory a seperate mechanism is used called as Direct Memory Access.

→ Direct Memory Access is a mechanism where data transfer takes place without continous involvment of processor (unlike polling and interrupt method where processor is required for every character transfer).

→ Hence processor is needed only at the start of data transfer, which informs the DMA controller how many words to be transfered, starting address of memory location and the direction of data transfer (R/W; i.e read or write)

→ Working of DMA controller is as follows

1) User program issues an instruction to read/write data from I/O device to/from memory.

2) OS realizes that this is a DMA operation and blocks the user program to start another program.

3) To initiate data transfer, processor sends starting address of memory from/to where data needs to be read/written. Processor also specify word count (no of words to be read or written) and direction of data transfer (read or write)

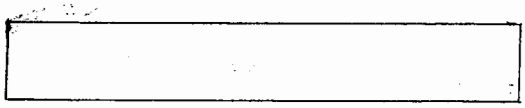
4) On recieving this information, DMA initiates the operation. Now the processor is free to do other task as DMA has taken over read/write operation.

5) Once DMA completes its operation it informs processor by raising an interrupt.

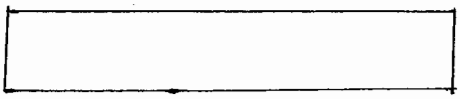
6) On recieving an interrupt OS realizes that DMA operation is complete and it removes the user program from blocked state to runnable state

I/O interface has a circuit called as DMA controller. DMA controller or DMA interface has three registers.

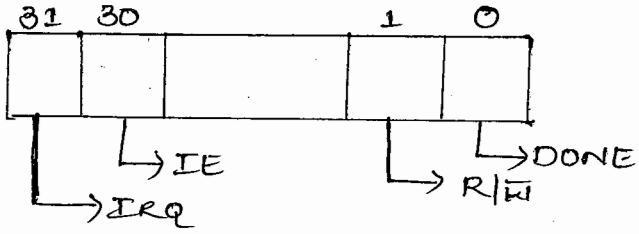
1) Starting address register :- holds the starting address of the block of data to be transferred.



2) Word count register :- Holds the number of words present in the block to be transferred.



3) Status and control register :-



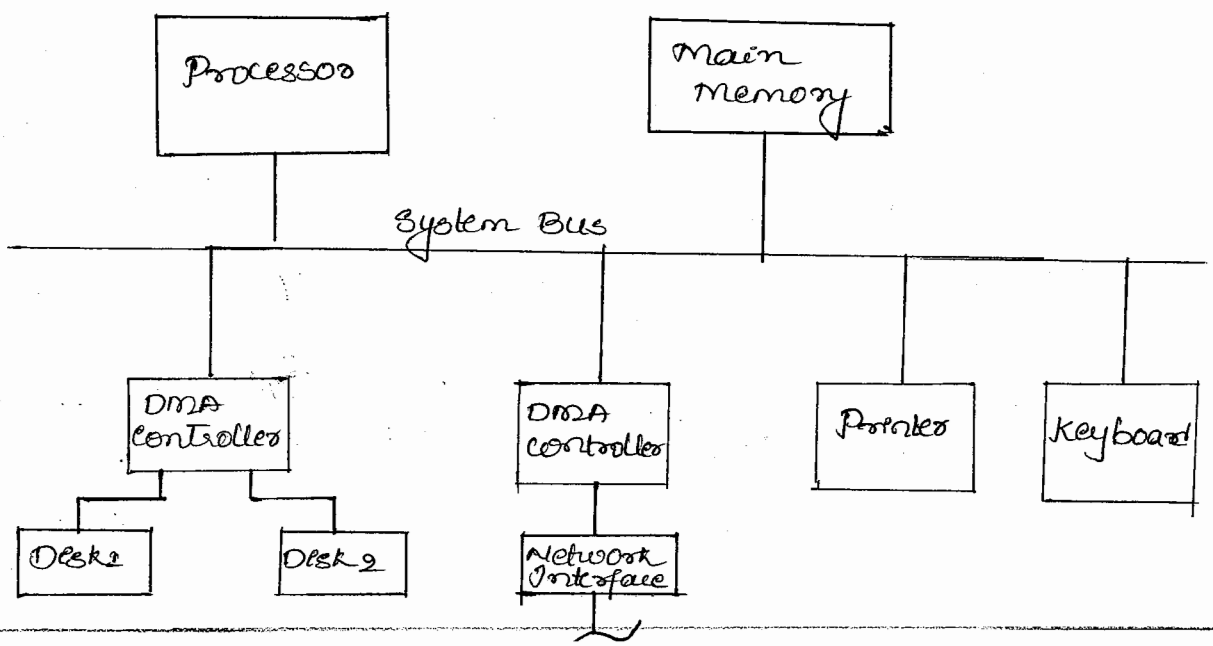
→ Bit 0 is DONE flag, which will be set when DMA has completed data transfer.

→ Bit 1 is R/$\overline{W}$ flag. If Bit 1 = 1 then it specifies read operation. If Bit 1 = 0 then its write operation.

→ Bit 30 is Interrupt Enable. If this flag is set to 1 then DMA can raise an interrupt.

→ Bit 31 is Interrupt request bit :- After raising an interrupt DMA sets this flag to 1.

DMA controller along with other components in the computer system.



→ In the above figure there are multiple DMA controllers. One DMA controller connects network interface to computer bus. Another DMA controller controls two disks.

→ DMA controller provides two channels (DMA channel) for each disk. which means separate registers i,e starting address, word count & status and data registers for both the disks.

When data needs to move from main memory to one of the disk, then following steps are performed.

1) Processor writes the starting address, word count and direction of transfer into the registers of one of the disks.

2) DMA controller works independently and performs required operation.

3) Once DMA transfer is completed following steps takes place
   a) Done flag is set to 1 indicating DMA transfer is completed.
   b) IE bit is set to 1, indicating DMA channel can raise an interrupt
   c) IRQ bit is set to indicate DMA has raised an interrupt.

Burst mode or Block mode transfer.

DMA controller is given exclusive transfer to access main memory to transfer a block of data without any interruption. Processor is required during the initation of data transfer. This is known as Burst mode transfer.

Cycle stealing.

Processor has the highest priority in the system to access Bus But when DMA comes into existence DMA controller gets the high priority than processor for data transfer. So when DMA performs data transfer we say that it is stealing the memory cycles of processor to perform data transfer.

Bus Arbitration

If two DMA controller tries to access the system bus at the same time then a conflict arises. i,e which DMA controller has to be given access to transfer data?

→ To select DMA controller arbitration method is used.

Bus Master :- The device which is allowed to transfer data (access system bus) at given point of time is known as Bus Master.

Arbitration is a method of selecting next bus master. (Means presently some other device is using the bus)

There are two types of arbitration

1) Centralized Arbitration
2) Distrubted Arbitration

1) Centralized Arbitration

→ In this method Professor is the one who selects the next master.
→ Normally when no DMA is involved in data transfer processor is the bus master.
→ Consider the fig where we assume that DMA3 is the bus master.
→ Since it is the bus master DMA3 would have activated $\overline{BBSY}$ (Bus Busy) signal to inform all the device that it is using the bus.
→ In meanwhile DMA controller1 and DMA2 wants to become the bus master.
→ Hence it requests the processor by activating $\overline{BR}$ (Bus request) signal.
→ Bus Request line is common to all devices.
→ On reciving this signal processor will the send BG (Bus Grant signal) on Bus Grant Lines.
→ DMA controllers are connected in Daisy chain fashion.
→ So DMA close to the processor recives Bus grant signal.

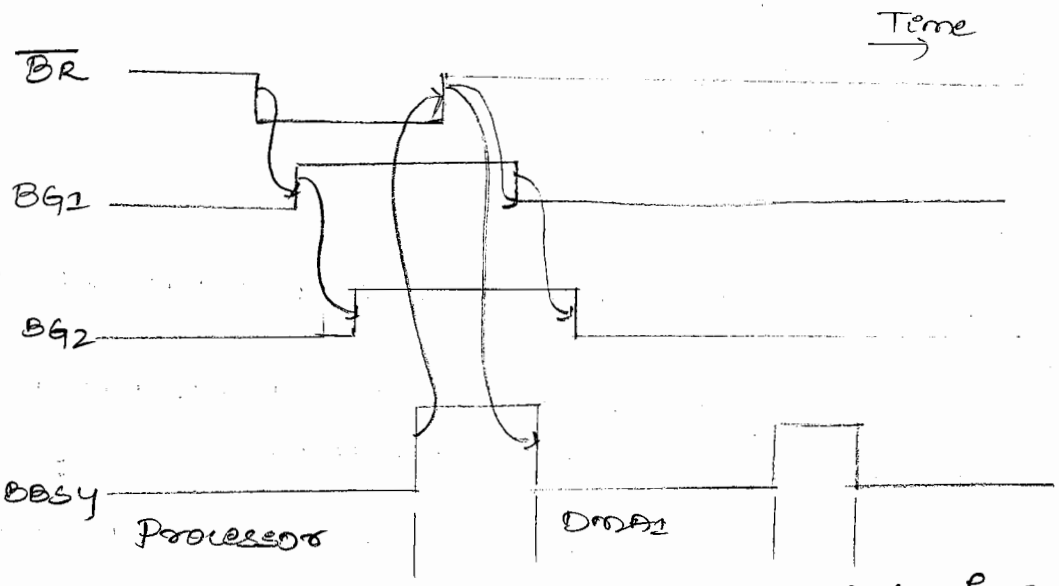
→ If DMA has request for bus then it blocks it else it forwards the signal to next DMA.

→ In this example we have assumed DMA1 and DMA2 has requested bus.

→ so DMA1 blocks BG signal and puts its identification code to the processor.

→ Once DMA3 completes the data transfer it inactivates $\overline{BBSY}$ signal.

→ DMA1 becomes the master and activates $\overline{BBSY}$ to inform all the device that bus is busy.



Signal transfer during Bus mastership for the devices.

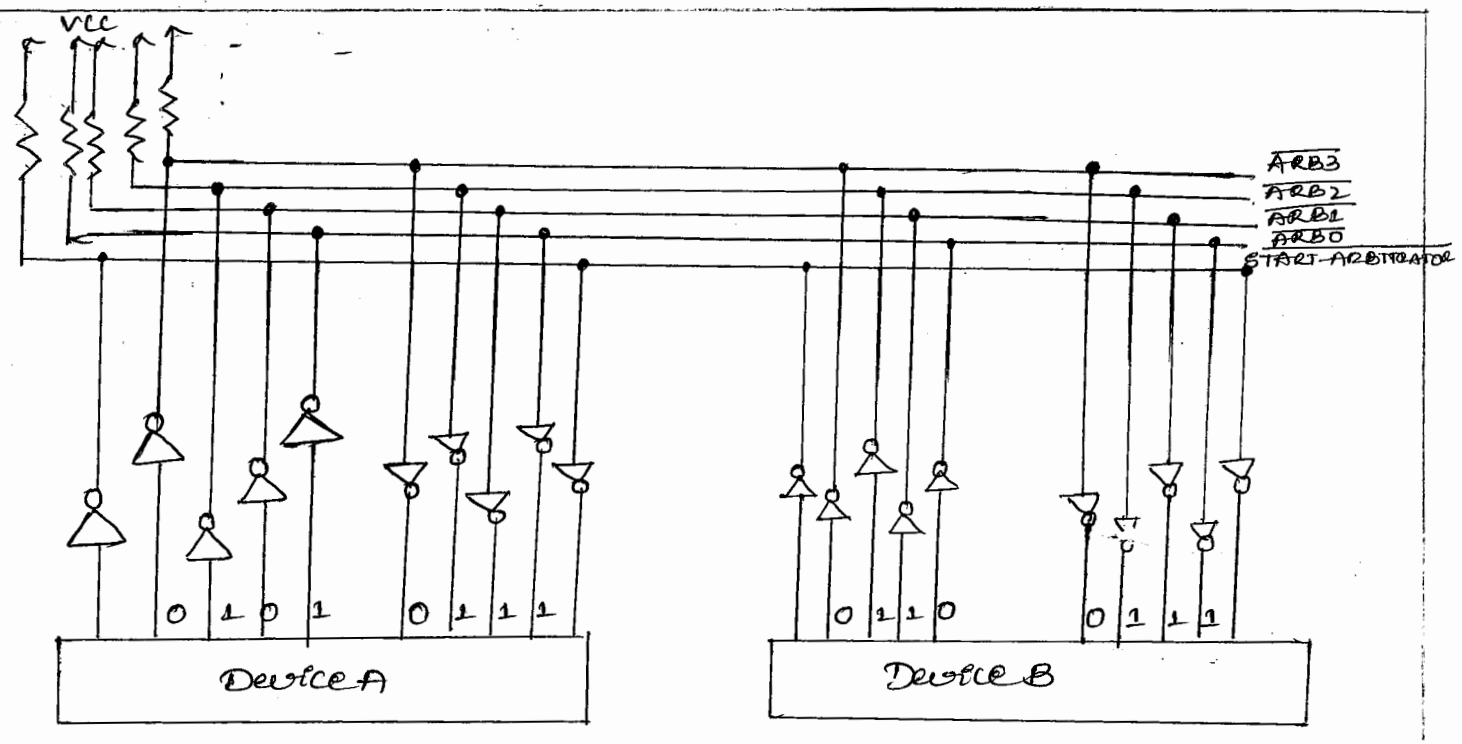
Distributed Arbitration

→ In a distributed Arbitration all the device can participate in becoming next bus master. There is no central arbiter.

→ Each device on the bus is assigned a unique 4 bit ID.

→ When device wants to participate for next bus master, it asserts start-arbitration and their ID's are put on the lines $\overline{ARB0}$ to $\overline{ARB3}$.

→ A winner is chosen based on the interaction among signals transmitted on these lines.



1 → Let device A transmits it ID 0101 (4) on $\overline{ARB0}$ to $\overline{ARB3}$.

2 → let device B transmits it ID 0110 (5) on $\overline{ARB0}$ to $\overline{ARB3}$

3 → An OR operation is performed on both the ID's.

4 → Result of this is transmitted back on lines $\overline{ARB0}$ - $\overline{ARB3}$.

5 → Now, device A checks its ID with return ID from the MSB. If any bit dosent matth then that bit line is disabled by plaing a zero.

6) If the answer matches with the unique ID then that device has won the arbitration process.

Step3) 0101
0110

0111 → return ID

3 2 1 0
Step5) 0101 → Original ID of A
0111 → Return ID

0100

At bit 1 there is a difference so from that position onwards 0 is plaied. compare the ans ie 0100 with orginal ID 0101, there is no match hence Device A has lost.

0110 → ID of B
0111 → Return ID

0110

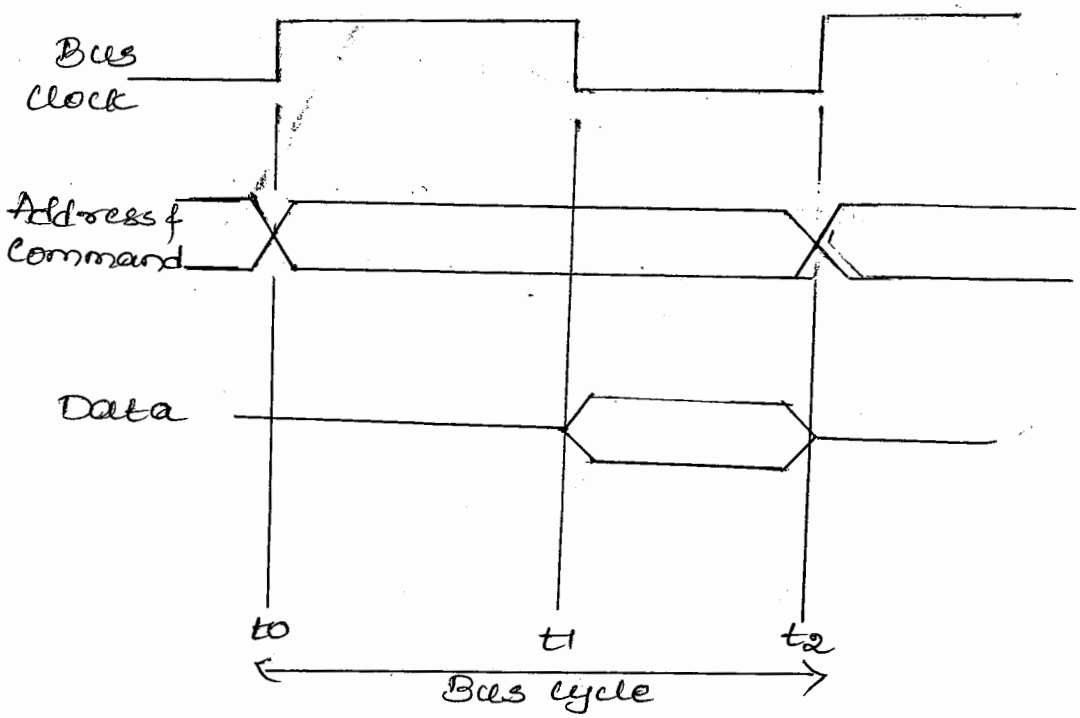
At bit 0 there is a mismatch so 0 has been placed. Compare ans 0110 with original ID 0110 both are same. hence device B has won the arbitration.

Buses

Bus provides communication path between processor, memory and I/O devices.
Buses are grouped into three types, data, address and control lines.
Control lines along with read write signal also carries timing information.
Timing information specifies at what time processor, I/O devices places data on bus and recieve data from bus.
Based on the timing of data transfer two schemes are developed
1) Synchronous scheme
2) Asynchronous scheme

Synchronous Bus

A bus is called as synchronous bus if all the device attached to the bus obtain their timing information from a common clock.
→ This clock generates equal time intervals, which is known as bus cycle, during which one data transfer takes place.
→ i,e master may place address on address bus and slave may place data on databus.



1) At time t0, master places the address on address line and command on command line.

2) This information travels along the bus. The time duration t0-t1 should be long enough for the address to reach slave.

3) At time t1 slave places the data on data bus.

4) Data is held on the bus for long enough for data to travel over the bus and read by master.

5) At time t2 master stores the data on to the buffer register.

This is an ideal scenario which never happens. In reality timing diagram looks as follows.

1) At time t0 master places address and command on the respective bus.

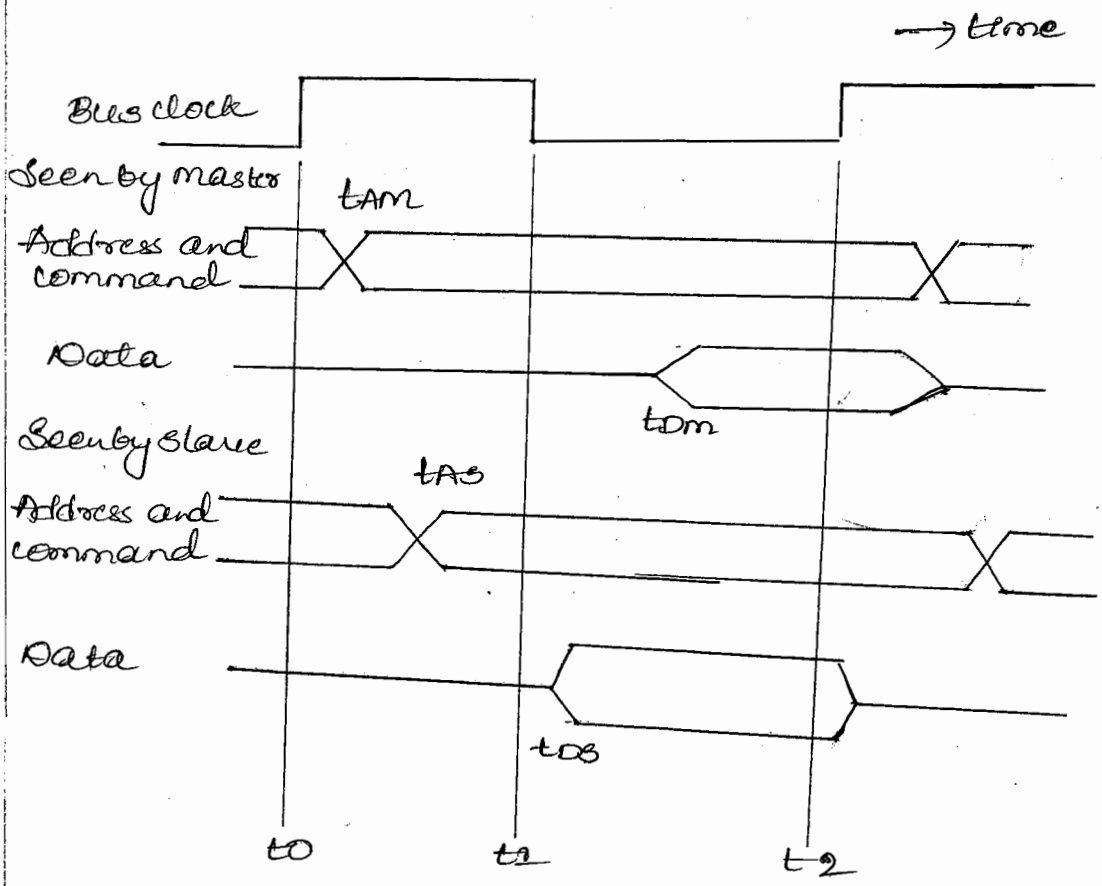
2) This doesn't appear on the lines immediately. Address and command appears on the line at time t_{AM} at the master.

3) This information travels along the bus reaches slave at time t_{AS}.

4> At time t_1 slave puts the data on data lines, but data appears on the data line at time t_{DS}.

5> Data travels along the bus and reaches master at time t_{DM}.

6> At time t_2, master saves the data into the buffer.



Multiple cycle transfer.

→ In the above method at time t_2, master assumes that a valid data is present on data line.

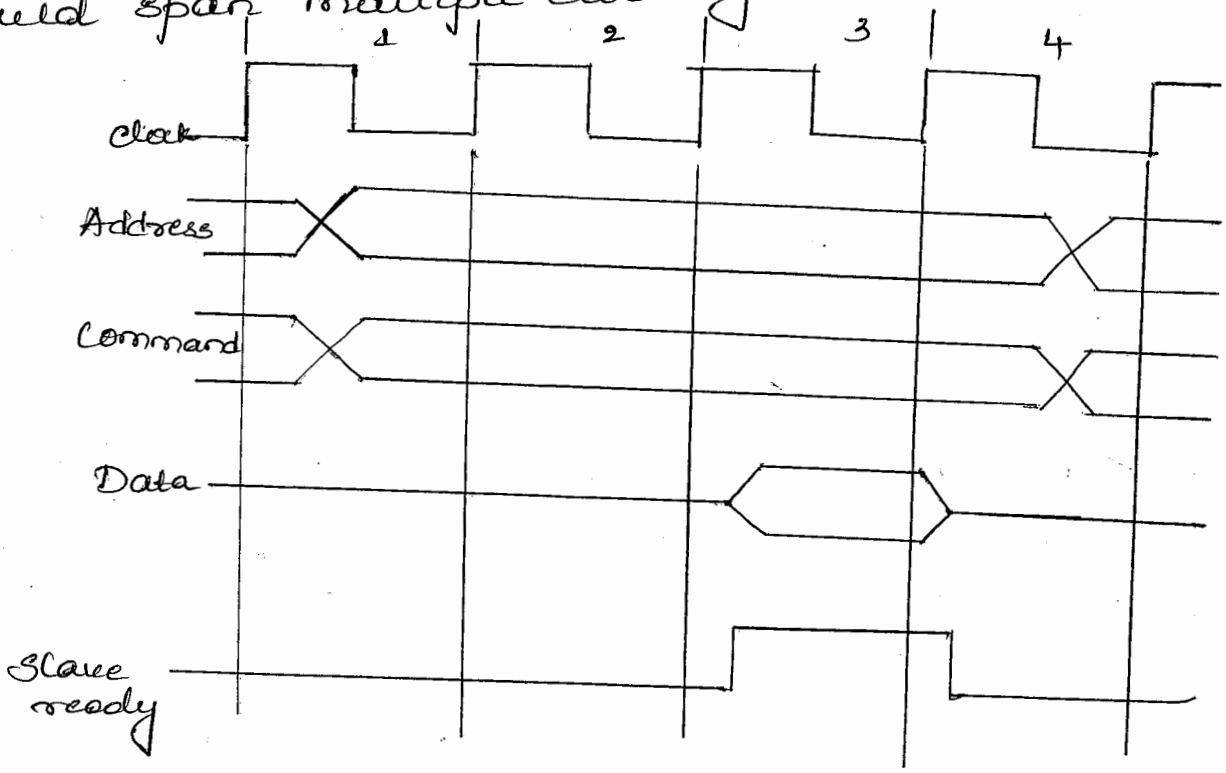
→ This may not be true because of malfunction of the device.

→ So there should be a mechanism through which master stores valid data into the buffer.

→ To fulfill this a slave ready signal is asserted when slave places data on datalines.

→ On recieving this signal master understands a valid data has been sent and it saves the data in buffer.

A high frequency clock is used, so that data transfer could span multiple clock cycles.



step1) During first clock cycle, master places address and command on address and command line respectively.

2) Slave receives this and decodes it. At clock cycle 2, Slave decides to respond.

3) At clock cycle 3, slave places the data on data lines and asserts a slave ready signal.

4) master receives slave ready signal (which is an acknowledgment from slave) and at clock cycle 4 master saves the data into buffer.

Asynchronous Bus

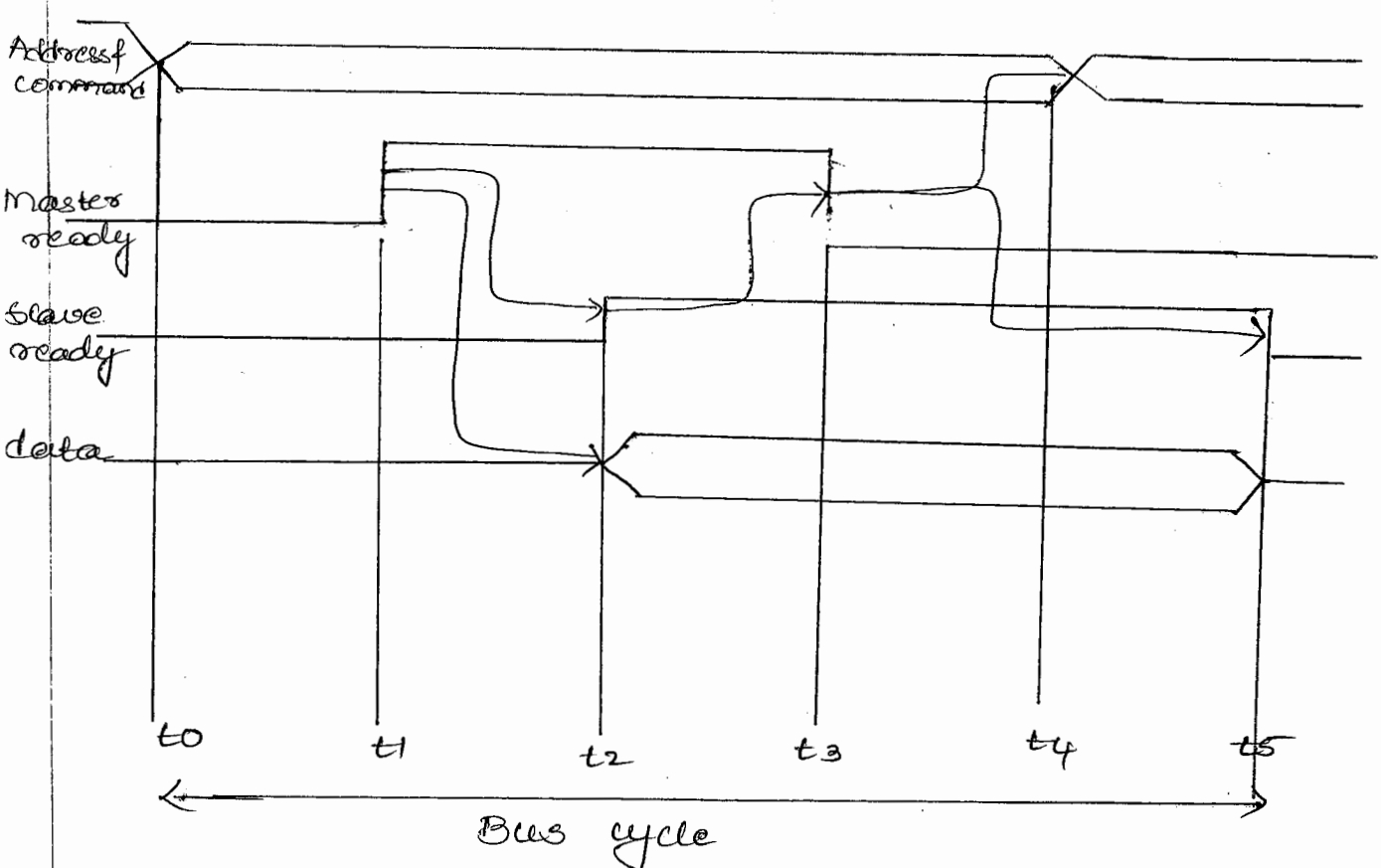
→ In asynchronous bus there exists communication between master and slave.

→ Hence data transfer takes place via handshake between the two.

→ There are two control line used for this purpose
1) Master ready to indicate it is ready for transaction
2) Slave ready is a response from slave.

Data transfer takes place as follows.

→ Master places the address and command information on bus.
→ Then it indicates all the devices by placing (asserting) Master ready signal.
→ All devices will decode the address. Selected slave performs the operation and informs the processor by placing (asserting) slave ready signal.
→ On recieving acknowledgment from slave master places the data on buffer and removes master ready signal.



Handshake control of data transfer

t_0 → At time t_0, master places the address and command on address and command lines respectively.

t_1 → At time t_1, master sets Master ready signal to 1 to indicate all devices the availability of address and command. Slave decodes the address.

time $t_1 - t_0$ is a delay. This is the time taken for address to reach all the devices. This delay is known as Skew.

t_2 → At time t_2 Selected slave will place the data on data lines and activates slave ready signal.

t_3 → At time t_3, slave ready signal reaches master, master places the data into buffer and deactivates master ready signal.

t_4 → At time t_4, master removes the address and command from address and command lines.

t_5 → At time t_5 slave ready signal goes low and data is removed from the data line.

Above timing diagram is for read operation.

Medha Gourayya
CSE, RNSIT

Computer organization
Module -3
Memory systems

The maximum size of memory that can be used in any computer is determined by the addressing scheme. By using 2 bits we can address 4 memory locations.

i,e $2^2 = 4$.

$2^3 = 8$ (using 3 bits 8 memory locations can be addressed)

$2^{10} = 1024$ or 1k (Kilo locations). $2^{19} = 512k$

$2^{20} = 1M$ (mega locations) $2^{21} = 2M$

$2^{30} = 1G$ (Giga locations)

$2^{40} = 1T$ (Tera locations)

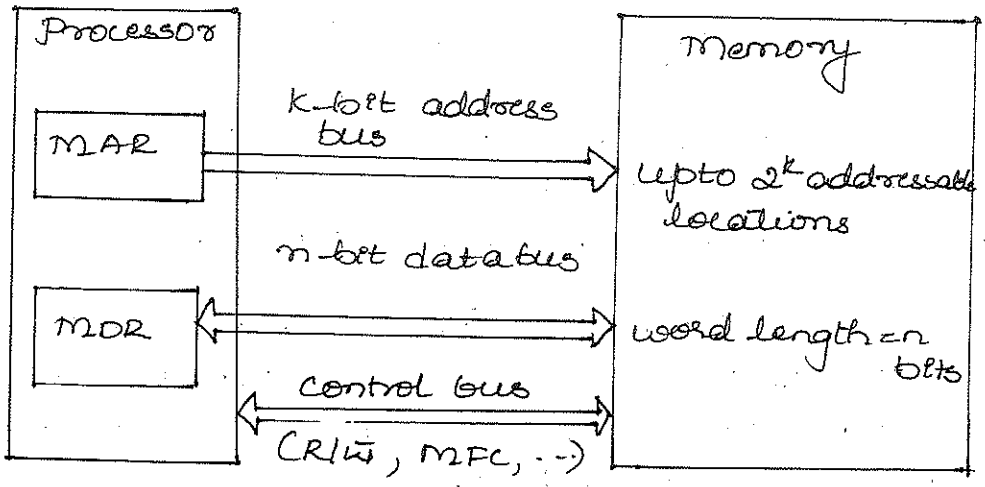
The number of bits that can be stored or retrived in one memory access is known as wordlength of a computer.

Connection of the memory to the processor

→ Data transfer between memory and processor takes place using the two processor registers, usually called MAR and MDR.

→ MAR holds the address which is of k-byte long and MDR holds the data of n-bit long.

→ The memory is made up of 2^k addressable locations.



→ Control line includes read/write (R/W) and Memory Function complete signals (MFC)

Read operation

→ Processors reads the data from the memory by loading the address of the required memory location into MAR register, and setting R/$\overline{W}$ signal to 1 (read operation).

→ Memory responds to the processor by placing the data from the addressed location onto the datalines.

→ It informs the processor that data has been placed on datalines by activating MFC signal.

→ When processor recieves this MFC signal, it loads the data from datalines to MDR register.

Write operation

→ Processor loads the address of memory location to which data has to be written into MAR register.

→ Processor loads the data into MDR register, and also sets the R/$\overline{W}$ signal to 0 (write operation).

→ Data is written to the addressed location.

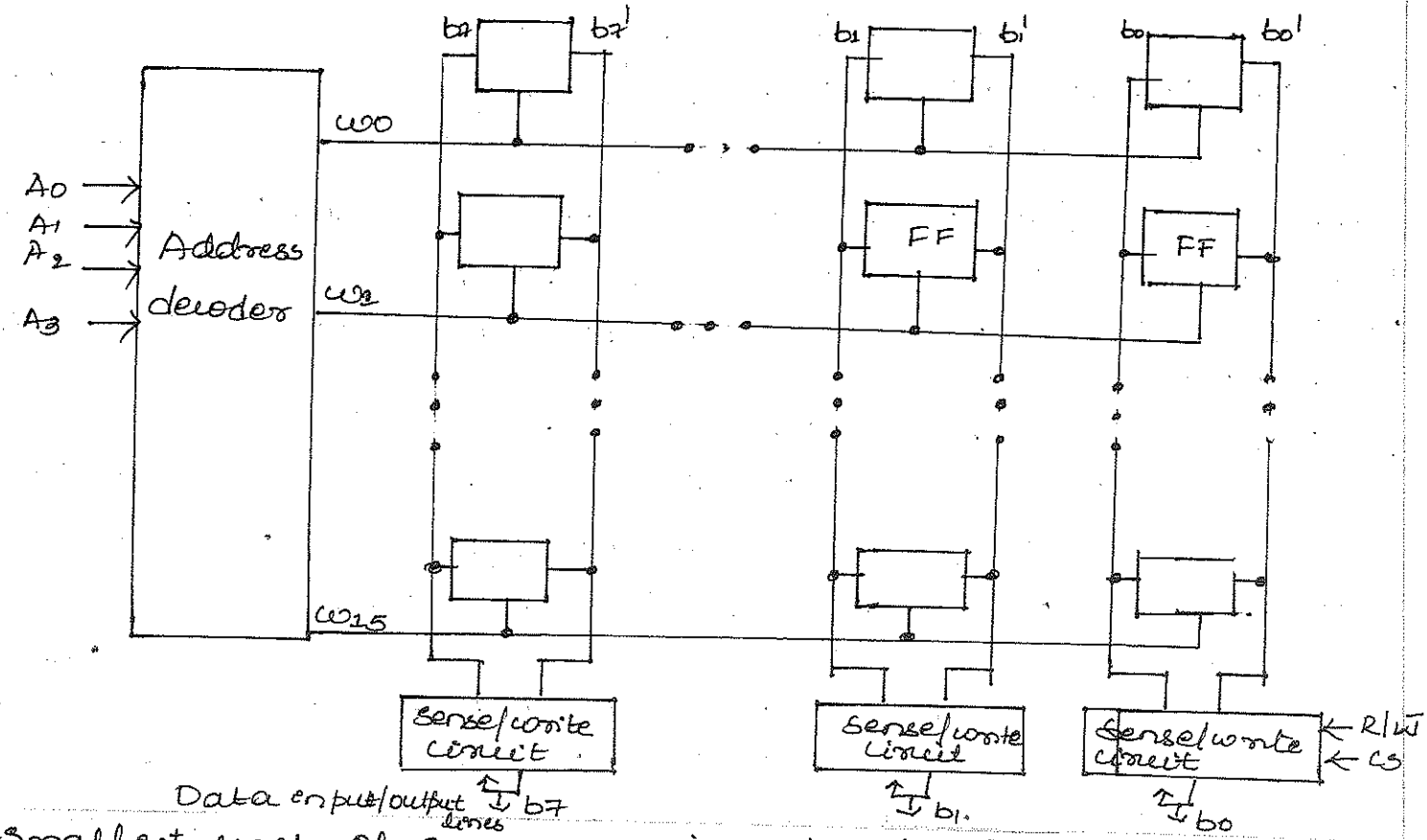
Note:- If read or write operation involves consecutive address locations in the main memory, then a "block transfer" operation can be performed in which the only address sent to the memory is the one that identifies the first location.

Memory Access time :- The time that elapses between the initiation of an operation and completion of that operation. i.e time between Read and MFC signals.

Memory cycle time :- The minimum time delay required between the initiation of two successive memory operations. i.e:- time between two read operations, or write operations or read-write operations.

Random Access Memory :- The memory which can be accessed in a fixed amount of time. Memory is made up of semiconductor cell. speed of RAM ranges from 10 ns to 100 ns.

Internal organization of Memory chips (16 × 8)



→ Smallest unit of a memory is a bit (cell).

→ Memory cells are organized in the form of array, and all the cells are capable of holding one bit of information. (0 or 1).

→ Each row makes a word. All the cells in the row are connected to word line.

→ These word lines are driven (connected) to the address decoder. (i,e when processor sends the address, these

→ address decoder helps to select a particular word in the memory).

→ The cells in the column are connected to sense/write (read/write) circuits using two bit lines.

→ Sense/write circuits are connected to data input/output line.

Read operation ...

→ Address decoder will identify a particular word in the memory.

→ Sense circuits will read the data from the cells of a selected word.

→ The read data is sent to output lines.

Write operation

→ Address decoder identifies or selects a particular word in the memory (as requested by processor).

→ write circuit will recieve the data from data input lines.

→ These data is stored in the cells of selected word.

The above figure shows the memory organization of 16 x 8

1) There are 16 words (rows) of 8 bits.

2) Since there are 16 words, 4 bits are needed to identify 16 rows (words)

3) 8 data lines are required to access individual cell of a selected row, which are inturn connected to data bus.

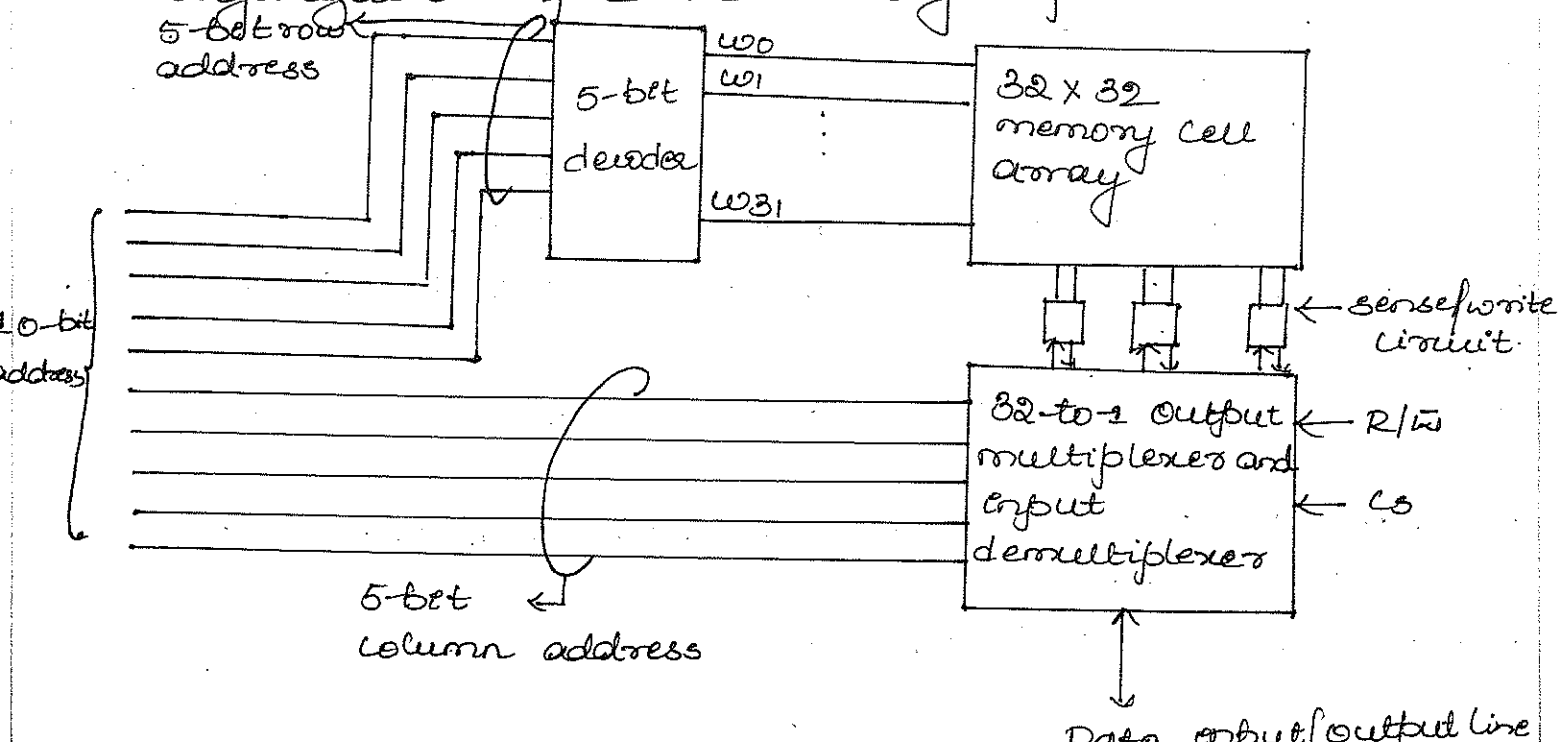
4) Two control lines are needed i,e R/$\bar{W}$ and CS (Chip select).
   → R/$\bar{W}$ specifies read or write operation.
   → Cs is used to select one of the memory chips in a RAM. There can be multiple chips (ex 16 x 8, 512K x 8) connected in parallel. Cs is used to select one such chip.

5) In 16 x 8 there are 16 words, 8 columns and totally 128 bits.

6) Thus there are 14 external connections from this memory, namely, 4 address lines, 8 datalines, R/$\bar{W}$ & CS,

7) Two additional lines are also needed one for power another for ground.

Organization of 1K * 1 memory chip.



→ 1K*1 (1024 cells) can be organized as 128 × 8 i.e 128 words(rows) each of 8 bits.

→ In this configuration 7 lines are needed for address, 8 lines for data, 1 line for R/$\bar{W}$, 1 line for CS and 2 lines for power and ground. Totally 19 external connections.

→ Another way to organize 1K(1024) memory is 1K*1 format. which means that there will be 1024 rows and one cell per row.

→ So we can read or write only one bit at a time. (only one data input/output line is supplied).

→ Since 1024 rows are present, 10 bits are required to uniquely identify each row.

→ Above figure shows that 10-bit address is grouped into, two five bits each.

→ first five bit is known as row address, and next five bits for column address.

→ Since 5-bits are used for row address there can be 32 rows, and 5-bits are used for column address there can be 32 columns (but only one cell can be read at a time)

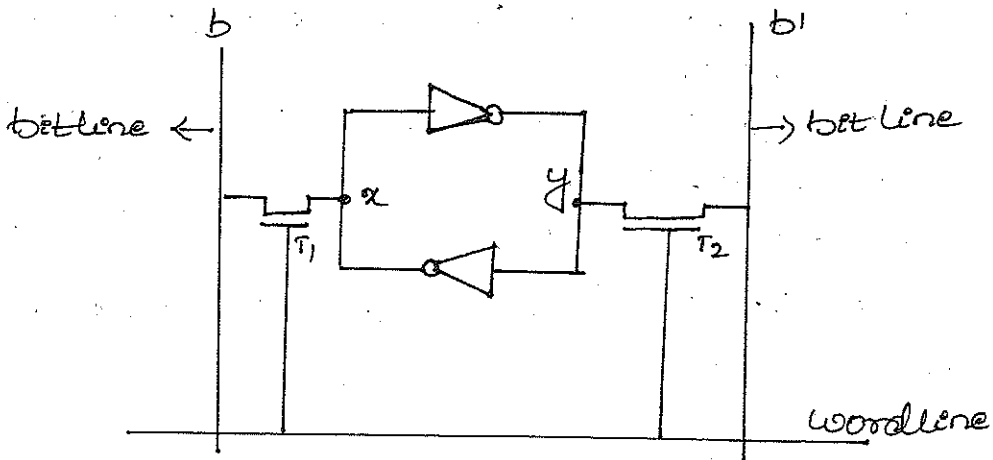
Working of circuit

1) 5-bit row address will fetch a row of 32-bits from 32 × 32 memory.

2) Since one row is having 32 bits we need to access only one bit at a time. so these 32 bits(cells) are passed to the column selector (lower circuit). using column address one of the 32 bits are selected and read or write operation is done on that cell.

Static Memories (Static RAM or SRAM)

Memories which are capable of retaining their state as long as power is applied are known as static memories.



A static RAM cell

1) Two inverters are cross connected to form a latch.
2) Latch is connected to, two bit lines using transistor T_1 and T_2.
3) Transistors work like a switch, that can be opened (grounded) or closed under the control of word line.
4) If word line is active then transistors are closed else transistors are open.

Read operation

→ Word line is activated to +5V. This causes the transistors T_1 and T_2 to close.
Consider a cell state is 0 (0 is stored in a cell). In this case voltage drop at x is 0 and at y is +5V.
→ Voltage drop at x is sensed at bit line b and y is sensed at b^1.
→ Sense circuit will sense the bit line b (voltage is zero) and passes it to system bus.
If cell state is 1 (1 is stored in a cell). In this case voltage drop at x is +5V and at y is 0V.
→ Voltage drop at bit line b is sensed and passed on to System bus.

Write operation.

→ Word line is activated.
→ Transistors are turned on.
→ Data to be written (0 or +5V) is sent on the bit line b and complement on b^1.
→ x gets the voltage as that of bit line b and y gets the voltage as that of b^1

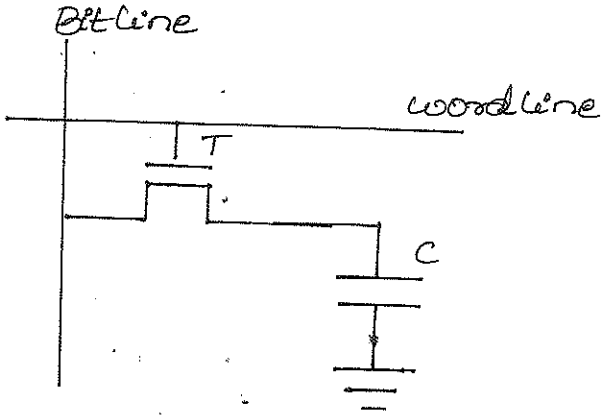
→ Once the value is sensed word line is pulled to ground. which means the sensed value is latched.

Asynchronous DRAM (Dynamic Ram)

→ Static RAM's are fast but its expensive so it is implemented using a capacitor.

→ Since only one transistor and one capacitor per cell implementation of DRAM cells are less expensive compared to SRAM.

Storing element is the capacitor. If capacitor has charge then cell state is 1. If capacitor charge is lesser than threshold then cell state is 0.



Storing information.

→ To store data in a cell, word line is activated which will turn on the transistor.

→ voltage is applied across bit lines because of which capacitor is charged.

→ when capacitor is charged that means 1 is stored in the cell. (cell state is 1)

But the property of capacitor is that it can hold the charge for few miliseconds, then it starts discharging. So to retain the data capacitor has to be periodically refreshed

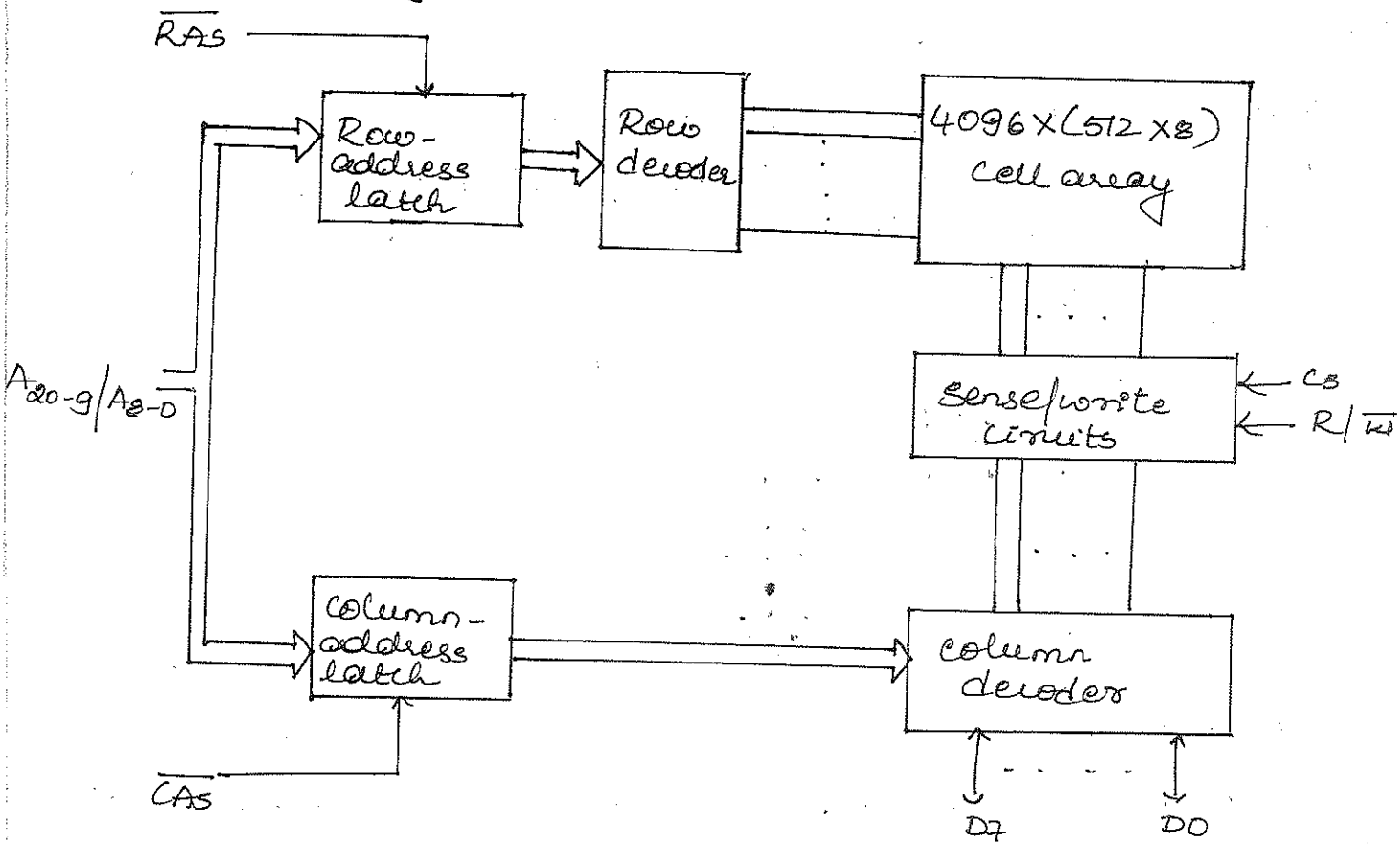
Read operation.

→ word line is activated, transistor is turned on.

→ Sense amplifier connected to bit lines detects if charge on the capacitor is more than the threshold. If yes then

cell state is 1. And also capacitor charge is made full.
→ bit line is pulled high.
→ If sense amplifier detects that charge is below threshold then it drains the capacitor to zero. i,e bit line will be pulled to ground.

Internal organization of 16-megabit DRAM organized as 2M * 8 memory chip



→ This organization is configured to read 8-bits at a time.
→ The cells are organized as 4K X 4K array. which means
→ there are 4K rows or words (4K = 4096 rows)
→ Each row is made up of 4096 cells.
→ Cells in each row is [illegible] into 512 groups of 8 bits each.
→ One row can have 512 bytes of data.
Note :- 2M x 8 means only 8 bits can be read or written at a time

→ total no of address lines required = 12 lines for 4096 rows and 9 line for 512 columns. 21 address lines are needed

Read operation

→ Row address are applied first.
→ when $\overline{RAS}$ (Row address strobe, active low signal) becomes low, row address will be latched (stored in latch)
→ Row address will be decoded by the row decoder and that particular row gets selected. Once it is selected all the cells in the row gets refreshed.
→ Column address is applied.
→ when $\overline{CAS}$ (column address strobe, active low signal) becomes low, column address will be latched.
→ Column address gets decoded by column decoder and one column gets selected (1 group among 512 groups)
→ If there is a read signal i;e $R/\overline{W} = 1$ then the sense write circuit will read the data from the cells and will place on data output lines. These lines are connected to system bus.
→ If there is a write signal ;e $R/\overline{W} = 0$ then sense/write circuit will store the data on input lines to selected cells.

Only one byte is read or written hence there are only 8 data lines.

Fast page mode

→ In the above method we read 8 bits from the entire row even though the complete row has been selected.
→ If we want to read entire row ;e all 512 columns following steps has to be repeated.

1) Apply row address. Row address gets activated.
2) Apply column address. Column gets activated & 8 bits are read
3) Apply row address to read the same row
4) Apply column address to read next 8 bits.

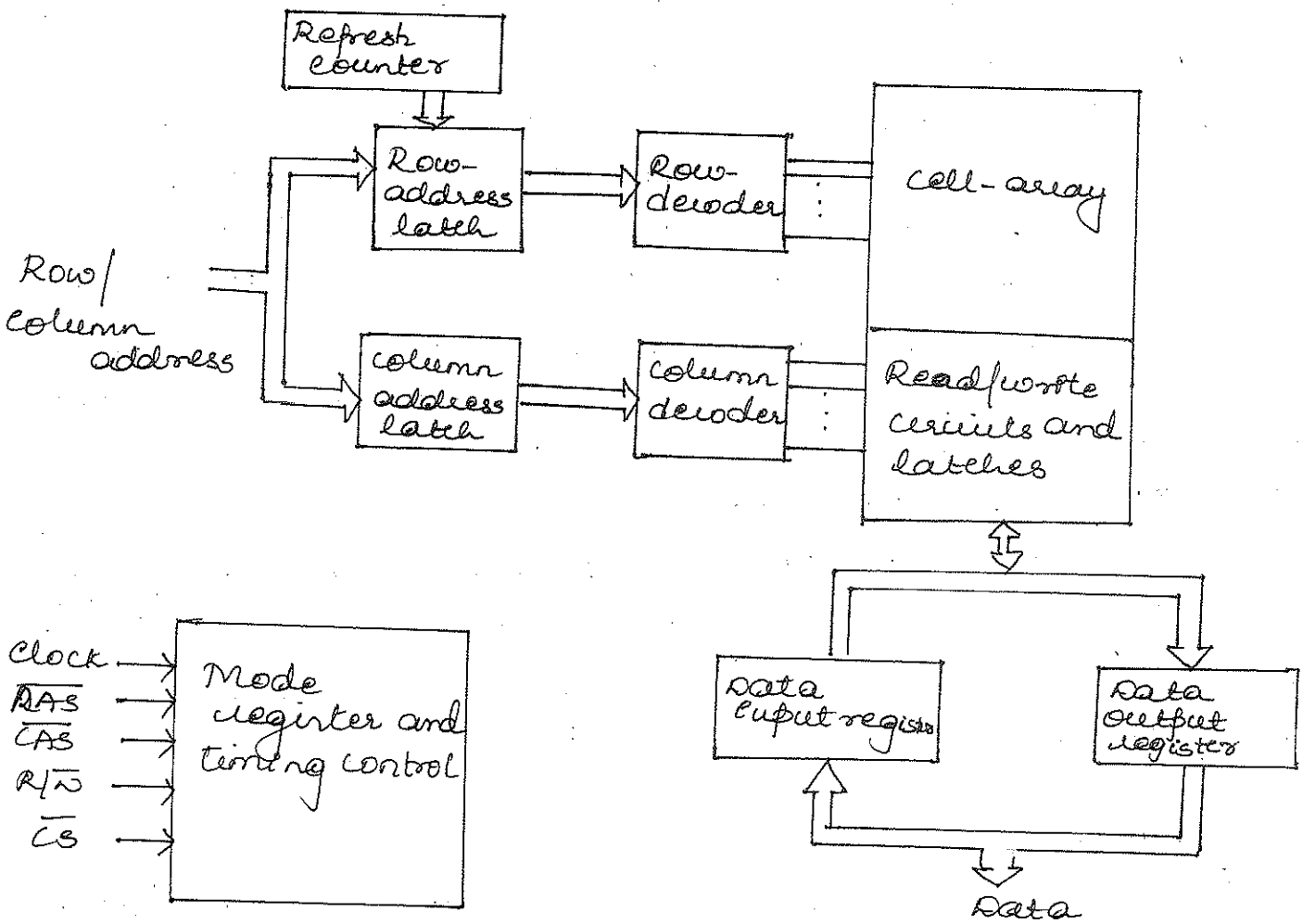
So this procedure has to be repeated 512 times to read the entire row.

→ TO avoid applying row address 512 times (multiple times) a latch is provided to each column.

→ on applying row address for the first time it gets saved into these latches.
→ so anymore to read the columns from same row no need to apply row address only column address are sufficient.

This is known as Fast page mode which makes the access fast.

Synchronous DRAM's



Asynchronous DRAMs have gone through some improvements In synchronous DRAM's operations are synchronized with a common clock.

All the arrangements are same as that of Asyne DRAM's except a special Refresh counter is added. For input and output special buffers are used.

→ Along with sense/write circuits latches are used.

→ When a row address it applied it gets stored in latch and gets decoded. On decoding that particular row gets selected.

→ When row is selected, the contents of entire row is stored in latches (512 latches of 8 bits).

→ On applying column address, data stored in the latch will be accessed.

→ Every column will have 8 bit latch which can store 8 bits of data.

→ SDRAM's has different modes of operation.

What type of operation is selected by writing into the mode register by processor.

If it is a burst transfer then each time there is no need to apply column address.

→ Internally there is a column counter which gets increment for every clock tick or clock cycle.

→ When column counter is incremented the data (8 bits) will be accessed and placed on the lines

Timing diagram for burst read of length 4 in SDRAM

→ In the first clock cycle row address is placed on the address line when $\overline{RAS}$ goes low.

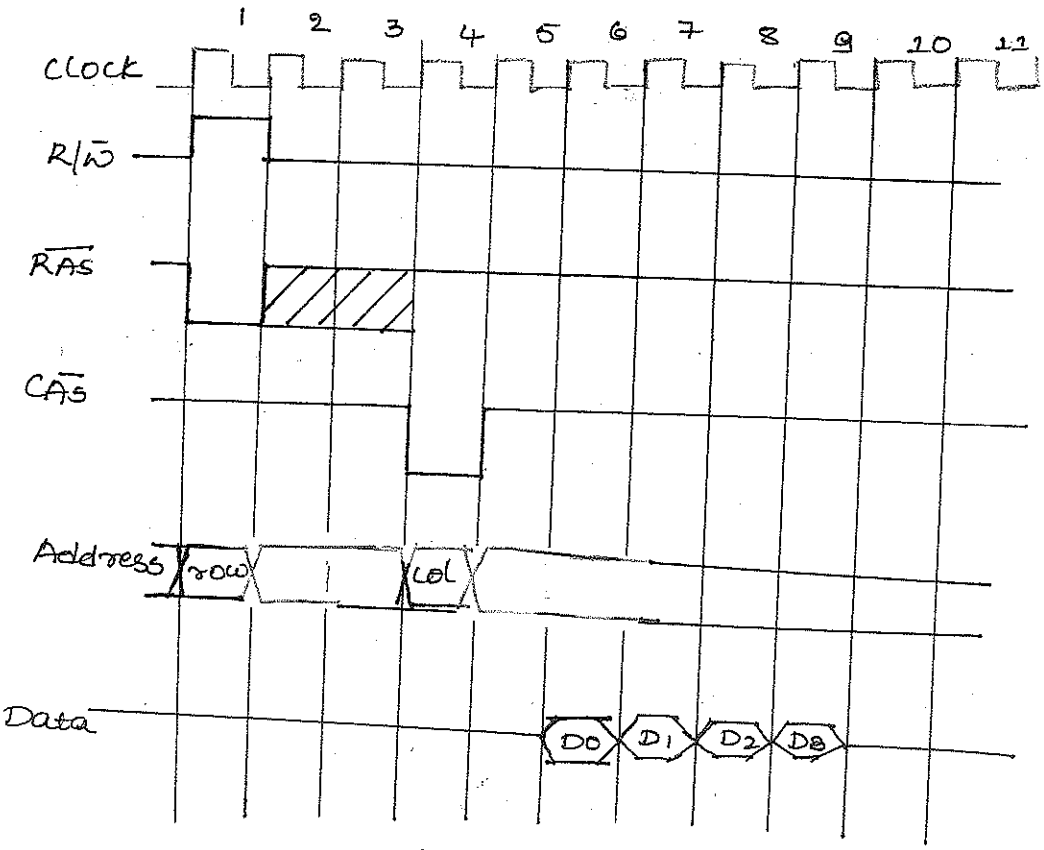
→ It takes 2 or 3 clock cycles to decode the address and to select a particular row. Once selected the contents are placed into the latches.

→ Then column address is applied which takes one clock cycle to decode the column.

→ At 6th clock cycle the first 8 bits are placed on the data line. Because of internal counter, next clock cycle counter gets incremented and data is placed on the data lines.

→ So in 6th, 7th, 8th & 9th clock cycle four bytes of data are placed on the data line.

→ There is no need to apply column address explicitly except for the first time.



Burst read of length 4 in a SDRAM.

Latency:- Is to refer the amount of time it takes to transfer a word of data to or from memory. That is time taken to transfer first word of it is a block of data. In the above diagram the time taken to access first word is 5 clock cycles. If this SDRAM had a clock rate of 100 MHz that what is the memory latency?

$R = 1/P$

$P = 1/R$

$R = 1/100MHz$

$P = 10ns.$

memory latency $= 5 \times 10ns = 50ns$

time taken for read is 50ns.

Bandwidth:- Number of bits or bytes that can be transfered in one second.

It also depends on speed of access to stored data, number of bits that can be accessed on parallel.

Structure of Larger Memories.

Consider a memory of 2M(2,097,152) words of 32 bits each. Suppose we need to design this memory using 512K x 8 static memory. Then following are the requirments.

1) How many chips are required?

$$\frac{2M \times 32}{512K \times 8} = \frac{2 \times 2^{20} \times 32}{512 \times 2^{20} \times 8} = 16 \text{ chips}$$

2) How many row are needed?

$$\frac{2M}{512K} = \frac{2 \times 2^{20}}{512 \times 2^{10}} = \frac{2^{21}}{2^{19}} = 2^{2} = 4 \text{ rows}$$

3) How many columns are needed?

chips = No of rows x no of columns

no of columns = 64/4 = 4 columns

4) total address lines?

$2M = 2 \times 2^{20} = 2^{21}$

21 address lines.

5) every chip has 512k(524288 rows).

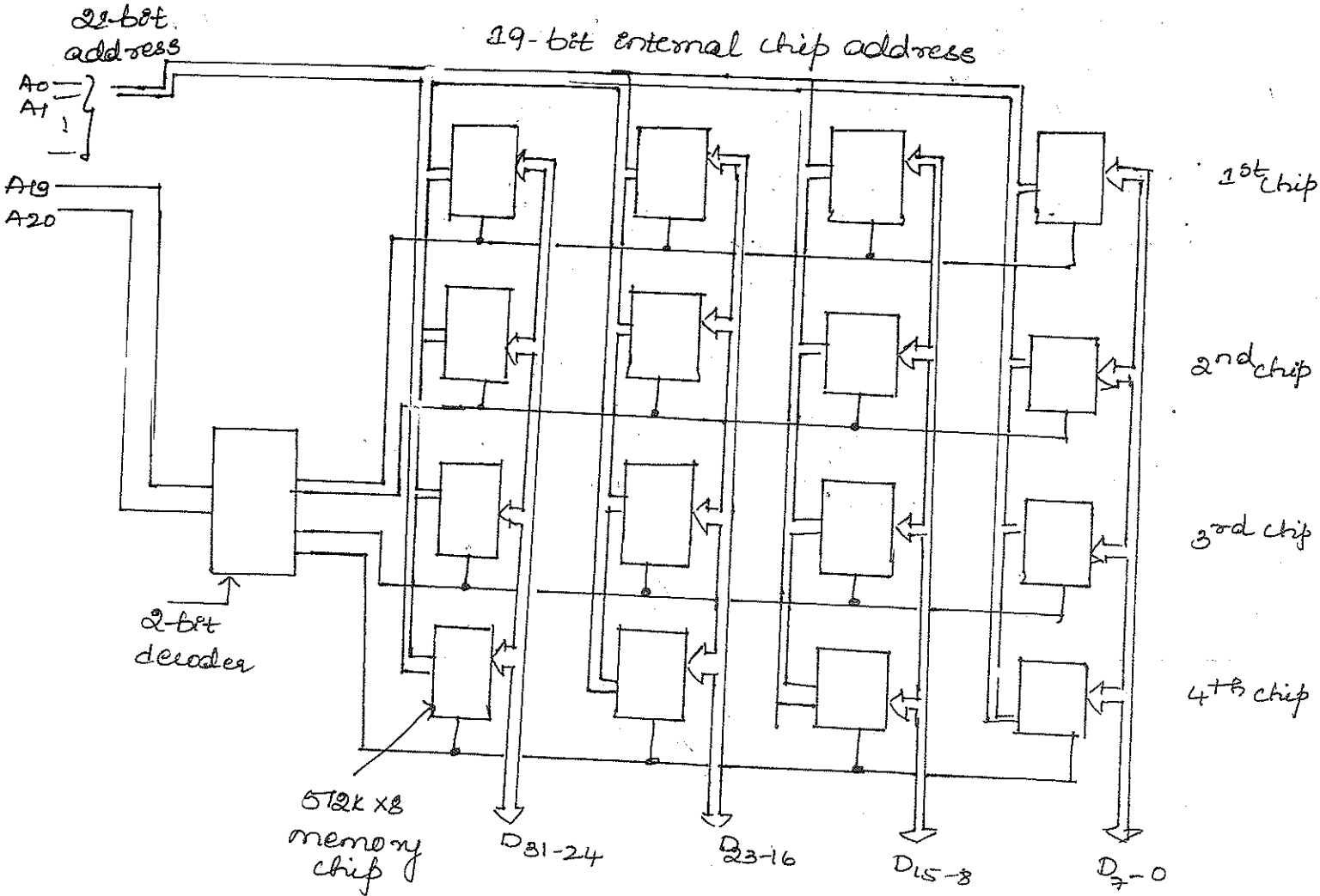
512×2^{10} i;e $2^{9} \times 2^{10} = 2^{19}$. 19 bits are needed to identify 524288 rows.

$2^{21} - 2^{19} = 2^{2}$.

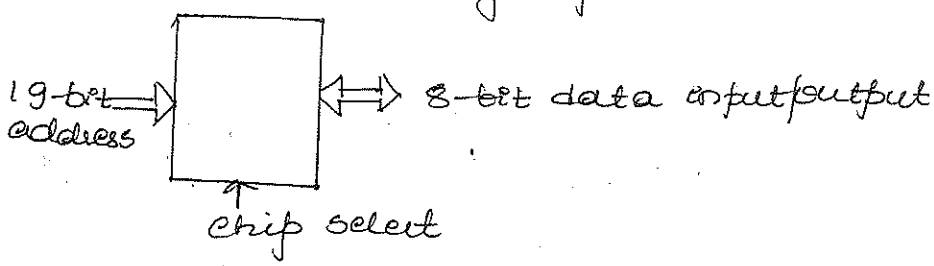
2 bits are need to identify 4 chips (512K*8)

2) Find out the rows, colums, chips, f address lines for 8M x 32 using 512K * 8

Organization of 2M x 32 using 512K x 8 static memory chip.



512K x 8 memory chip



→ First column gives 1'st byte of 2M memory. similarly 2nd, 3rd & 4th column gives 2nd, 3rd & 4th byte of 2M memory.

→ 4 bits are used for chip select.

→ 19 bits are used to fetch specific byte from each chip of selected row.

Memory Controller

Memory controller is placed inbetween processor and main memory.

When processor wants to read or write from a memory it issues the address and also sends read or write signal.

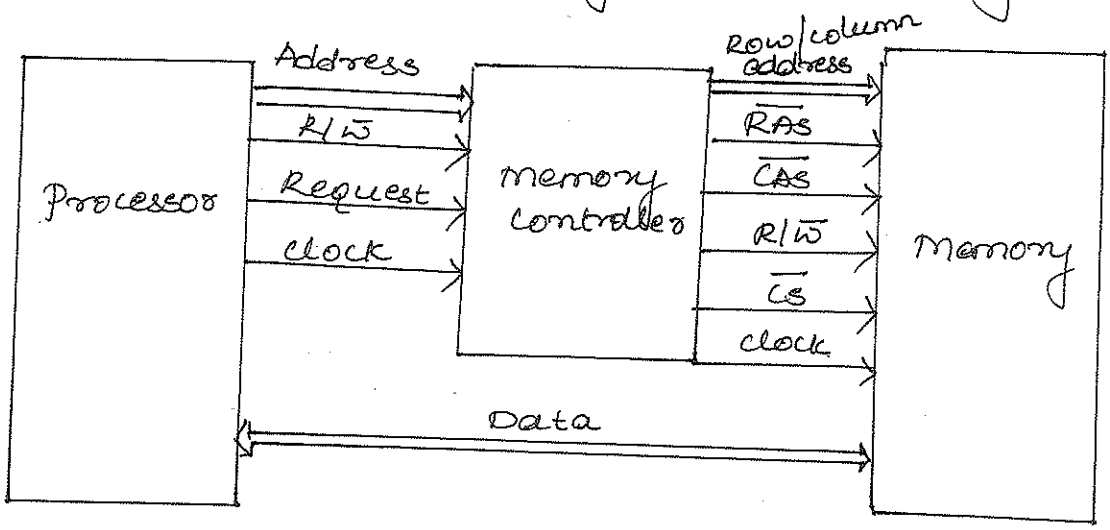
Memory controller then divides the address to row address and column address by using multiplexing mechanism.

Along with that it also generates $\overline{RAS}$ and $\overline{CAS}$.

Memory controller also performs refreshing of cells.

At 64ms cells of DRAM has to be refreshed.

It also sends R/$\overline{W}$ and cs signals to memory.



→ The controller accepts the complete address from processor and R/$\overline{W}$ signal from processor.

→ It also sends request signal. On recieving request signal Memory controller understands that memory operation is required.

→ It then divides the address into row address and column address and generates $\overline{RAS}$ and $\overline{CAS}$ signals

→ It also sends (forwards) R/$\overline{W}$ and cs to memory.

In SDRAM each row will be refreshed at every 64ms. Consider a SDRAM having 8k rows and clockrate is 133 MHz. Calculate the time needed to refresh all rows. Assume 4 clock cycle is needed to access each row. Also calculate refreshing overhead.

1 row access = 4 clk cycle

8k row access = $8 \times 1024 \times 4 = 32768$ cycles. (to refresh 8 rows)

Rate = cycles/time

time = cycles/rate = $32768 / 133 \times 10^6$

$= 246 \times 10^{-6}$

$= 0.246$ ms

$$\text{Refreshing overhead} = \frac{\text{time required to refresh}}{\text{Refresh interval}}$$

$$= \frac{0.246}{0.64}$$

$$= 0.0038$$

Difference between SRAM and DRAM

SRAM	DRAM
→ stands for Static Random Access memory	stands for Dynamic Random Access memory
→ No need to refresh as the transistor held inside would continue to hold the data as long as power supply is not cut off	Needs to be refreshed periodically.
→ Faster access time	Slow because of refreshing
→ High cost	Low cost
→ used for cache memory	used for main memory.

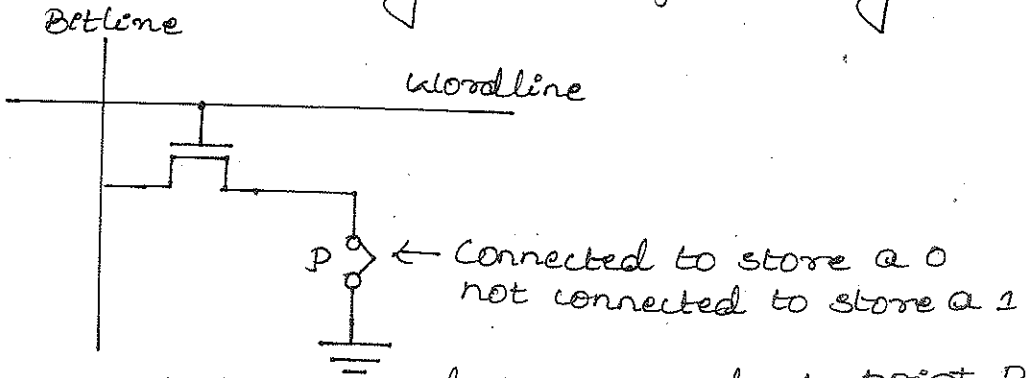
Read-only Memories (Non Volatile Memories)

SRAM and DRAM are volatile memories. When we switch on the computer boot program gets executed which loads the operating system into main memory. So a small amount of non-volatile memory is required which loads the data or instructions. On execution of these instructions booting takes place.

Booting is a process of loading os to main memory from hard disk.

1) ROM

→ Data is written during its manfucturing.



→ If transistor is connected to ground at point P, cell state is 0.

→ If transistor is not connected to ground at point P, cell state is 1.

→ Voltage is applied across bit line.

→ word line is activated, transistors will be turned on.

→ If transistor is connected to ground at point p then bit line will be pulled to 0V.

→ If transistor is not connected to ground at point p then bit line will be high.

→ Sense circuit connected to bit line will detect the voltage across bit line. If bit line is 0, cell state is 0. If bit line is 1, cell state is 1.

2) PROM (Programmable ROM)

Some ROM's allow the user to enter some data into it. Hence it known as programmable ROM.

Initially all the cells will be zero. There will be a fuse at point P.

User can place 1 (data) wherever he wishes by burning the fuses at point p using high current pulses. Hence ROM is programmable. But they are programmed only once.

3) EPROM (Erasable, reprogrammable ROM)

Some ROM's allows erasing of stored information and reprogramming of ROM called as EPROM.

To erase the content entire chip is exposed to uv rays. When it is subjected to uv rays the content get erased by dissipating the charge.

cell structure is same as that of ROM cell. A special transistor is used which acts as normal transistor or disabled transistor (always open).

charges are injected to transistor, once injected it gets trapped inside it.

If transistor contains charge, cell state is 1 else zero.

4) EEPROM (Electrically Erasable, reprogrammable ROM)

→ The disadvantage of EPROM is that to erase the data chip has to be removed and subjected to uv rays. All the cells in the chip will loose its data.

→ Another version of EPROM that can be programmed and erased electrically is available called EEPROM.

→ EEPROM do not need to be removed to be erased.

→ Also selective cells can be erased using high voltages.

→ Only disadvantage is that different voltages are required for erasing, writing and reading the stored data.

5) Flash Memory

→ Like EEPROM, flash memory cell, cell state will be 1 by trapping the charge inside a transistor.

Difference between Flash and EEPROM

→ In EEPROM data is written one bit (cell) at a time where as in case of Flash data is written one block at a time.

→ Flash devices has higher density which leads to higher capacity and low cost.

→ They require a single power supply voltage and consume less power in operation.

Because of low power consumption they are used in devices like hand-held computer, mobile phones, digital camera's, MP3 players and so on.

Two forms of flash module are:-

1) Flash cards

→ Flash memory is in the form of a card which can be inserted in devices like MP3 players. They come in various sizes like 8, 32 and 64 MB (now its available in GB)
One minute of music takes one MB space for MP3 format. So, 64 MB card holds about an hour of music.

2) Flash Drives

→ Flash drives that emulate hard disks have been created. Their storage capacity is limited compared to hard disk.
→ Flash drives are solid state drives without movable parts hence they have shorter seek time and access time that results in faster response.
→ They have low power consumption hence can be used in battery operated device.
→ They are insensitive to vibration.
→ Limitation is that they become unusable after about million read or write.

Speed Size and Cost (Memory Hierarchy)

An ideal memory would be fast, large and inexpensive.
→ SRAMs are fast but expensive because of transistors/inverters in one cell. So it is used to construct memory which is smaller in size. ex cache memory.
→ DRAMs are simple and less cost compared to SRAM, but slow because of periodic refreshing of cells. It is used to construct main memory.
→ Secondary storage can store bulk amount of data, it is inexpensive but very slow. Ex Magnetic disk.

Memory hierarchy is as follows.

1) Registers :- Fastest access to data is via registers. They are in top in terms of speed of access.

2) Processor cache :-

Small memory which is implemented on processor chip itself is known as cache. It holds the copies of instruction and data that are held by main memory.

There are two types of cache.

a) Primary cache :- This is placed on processor chip, also called as level 1 cache or L1 cache.

b) Secondary cache :- This is placed in between between processor and main memory. It is known as level 2 or L2 cache.

3) Main Memory

These are larger than cache but slower than cache. They are made up of DRAMs

4) Secondary Memories

These are huge and inexpensive storage device. But they are very slow compared to main memory.
(Memory Hierarch diagram is on the last page)

Cache Memories

When a program has to be executed it is loaded into main memory.
Processor will fetch the instruction and data from main memory.
Since processor executes at very high speed and main memory is slow, to fetch the information from main memory processor has to wait.
So to increase the speed of execution or to reduce the waiting time of processor, a small fast memory is used inbetween processor and main memory.
Cache is used to increase the speed of execution. Hence effectiveness of cache depends on "locality of reference".
In a program there will be loops, nested loops, recursive code which executes repeatedly. Such codes are brought into cache. This is known as locality of reference.
It has two aspects.

1) temporal aspects :- "A recently executed instructions is likly to execute again very soon." When processor fetches an instruction from main memory it is brought into cache thinking that processor needs the same instruction again.
So next time when requests for same instruction it is fetched from cache instead of main memory.

→ If the address sent by the processor is not in cache then write miss occurs.

To perform write operation two protocols are used:

1) Write through protocol.

→ On a write hit, the cache and main memory are updated simultaneously.

→ On a write miss, processor directly writes to main memory.

2) Write back protocol

→ On a write hit, only cache gets updated. which means the data is written only on cache. Hence updatation of main memory is pending. To identify - main memory updation is pending, the updated block in cache is marked as dirty or modified bit.

→ Main memory gets updated when dirty block of cache has to be replaced by a newer block of main memory when cache is full.

→ On a write miss, the word containing the block from main memory - brought into cache and then cache gets updated.

Since cache memory is smaller in size all the blocks of main memory cannot fit in cache. Hence when cache is full, the blocks of cache are replaced by the newer block of main memory. To identify which blocks of cache has to be replaced, Replacement algorithms are used.

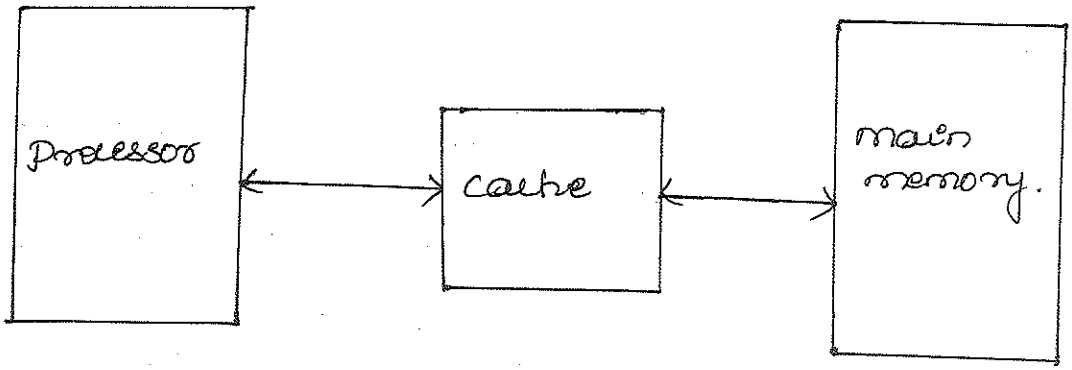
Processor doesn't know the existance of cache. Hence there should be some correspondance between cache and main memory.

This correspondance is achived by mapping functions.

There are three types of mapping functions.

1) Direct
2) Associative
3) Set-Associative.

spatial aspect :- " Instruction in close proximity to a recently executed instruction is also likely to be executed soon. Not only one instruction, but the instructions nearing to the address sent by the processor (block of code ex loops--) will be placed in cache, also the data is placed in cache Hence when processor wants to read a word from a block it reads from instead of main memory.



Use of a cache memory.

Processor issues a read request by sending the address of the word in main memory. The cache controller circuit determines whether the request word is available in cache. If the word is available the read hit has said to be occured.

If the requested word in not present in cache then we say that Read miss has occured.

On a read miss the block of code containing the requested word is brought to cache and from cache it is sent to processor.

Because of this process (coping from main memory to cache & from cache to processor) processor has to wait for longer time, hence on a read miss the requested word is directly transfered to processor from main memory. This process is known as load-through or early restart.

Processor issues a write request by sending the address of the main where data has to be written (stored). If the address sent by processor is present on cache then write hit has occured.

→ tag bits are used to identity the superblock. i,e the block belongs to which superblock. Since there are 32 superblocks five bits are sufficient to uniquely identity them.

→ block bits are used to identify the blocks within a superblock. Since there are 128 blocks in a group, seven bits are enough to uniquely identity them.

→ word bits are used to identify the word in a block. Since every block has 16 words, 4 bits are used to identify them.

→ When processor issues a read or write request by sending an address, the first five bit of address is compared with the tag bits of cache. If there is a match only then 7 bits are compared to identify the block and then 4 bits are compared to identify the word.

→ If tag bits doesn't match then there is no need to compare block bits and word bits.

→ If none of the tag bits matches then it is a read miss. So the block has to be read from main memory.

The limation of direct mapping is, even though cache is not full, contention arises when block 0 and block 128 of main memory needs to be placed in cache. Because both the blocks occupies same place in cache. Hence Block 0 of cache is replaced by block 128 of main memory. So once again if processor wants to read block 0 it has to be fetched from main memory.

Direct Mapping

Consider a main memory having 64k words of 16 bits.
i,e $64 * 1024 = 65536$ words.
These words are grouped into 4096 blocks. which means each block will have $64k / 4096 = 16$ words.

Consider a cache having 2k words of 16 bits.

$2 * 1024 = 2048$ words.

These words are grouped into 128 blocks.

$2k / 128 = 16$. 16 words in each block.

→ Direct mapping states that block j of main memory maps to "j modulo 128" block in cache.

$0 \bmod 128 = 0$ $128 \bmod 128 = 0$ $256 \bmod 128 = 0$
$1 \bmod 128 = 1$ $129 \bmod 128 = 1$
$127 \bmod 127 = 127$ $255 \bmod 128 = 127$

→ Block 0, 128, 256, 512, 1024 .... of main memory always maps to Block 0 in cache.

→ Block 1, 129, 257, 513, ... of main memory always maps to Block 1 in cache.

.....

To implement such a cache system, the main memory address sent by the processor is divided into three fields.

Main memory can be thought as are divided into

1) 32 groups (superblocks)
2) Each superblock will have 128 blocks.
3) Each block will have 16 words.

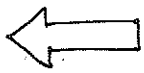
Hence 16 bit address sent by processor is divided into 5 bit tag field, 7 bit block field and 4 bit word field.

Main memory address

5	7	4
tag	Block	word

Cache

tag	Block 0
tag	Block 2
tag	Block 127



Main memory

Block 0
Block 1
//
Block 127
Block 128
//
Block 255
Block 256
//
Block 4095

Direct mapped cache.

Associative mapping

Main memory address

12	4
tag	word

Cache

tag	Block 0
tag	Block 2
tag	Block 127

Main memory

Block 0
Block 1
//
Block i
//
Block 4095

Associative Mapping

→ In this mapping, blocks of main memory can be placed anywhere in cache. which solves the contention problem of direct mapping.

→ Address of main memory sent by processor is divided into two fields.

1) 12 bit tag field :- since there are 4096 blocks, 12 bits are required to identify 4096 blocks.

2) 4 bit word field :- every block has 16 words, so 4 bits are needed to identify 16 words in a block.

→ tag bits of the address received by processor is compared with tag bits in cache, to check if block is present in cache or not. This is known as Associative mapping.

→ An existing block is replaced to make a way for new block only if cache is full.

Set-Associative Mapping

→ It is a combination of Direct and Associative mapping.

→ Main memory is divided into 64 superblocks.

→ Each superblock has 64 blocks

→ Each block has 16 words.

Even cache memory is grouped into sets. Each set can have 2 blocks in it. So there are 64 sets in cache.

→ Block 0, 64, 128 ..... of main memory is always placed in set 0 of cache. (Direct mapping)
In set 0 it can be placed as Block0 or Block1 (Associative mapping)

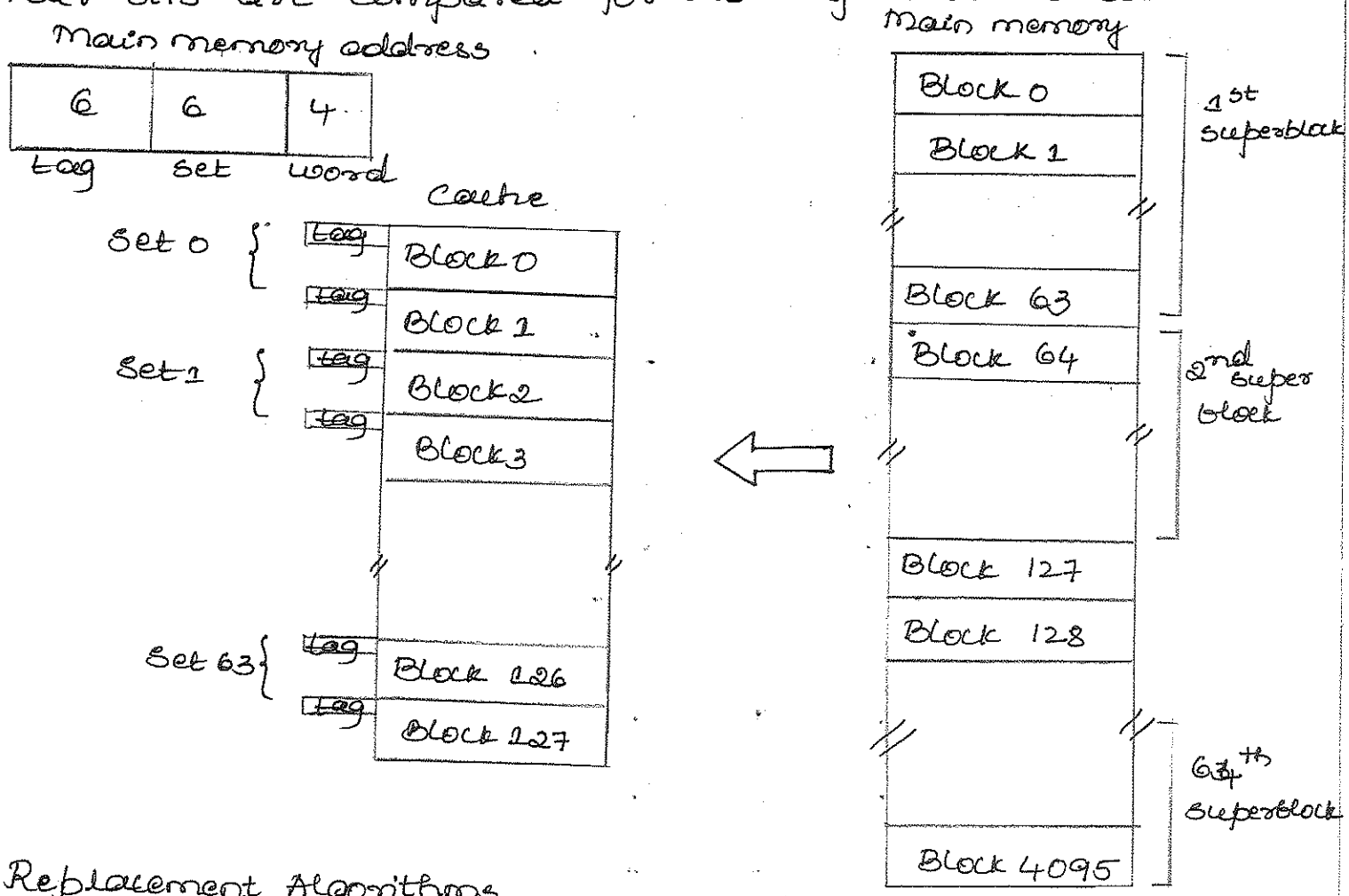
→ Block 1, 65, 129 ... of main memory is alway placed in set1 of cache.

→ Address is divided into three fields.

→ 6 set bits are used to identify 64 superblocks.

6 tag bits are used to identity 64 blocks in a superblock.
4 word bits are used to identify 16 words in a block.

When processor sends the address set bits of the address is compared with set bits of cache, if they match then tag bits are compared to identify the block if it matches four bits are compared for the respective word.



Replacement Algorithms

In case of Associative mapping and Set-Associative mapping blocks from the cache has to be replaced by a new block when cache is full. To determine which block to be replaced Replacement algorithms are used.

1) Least Recently used (LRU) algorithm.

When a block needs to be replaced choose a block that has not been referred for a long time known as LRU algorithm

Every blocks will have a counter. counter is set to zero when the block has be accessed and all the counters of other block is incremented.

During the time of replacement the block whose counter

has highest value is replaced.

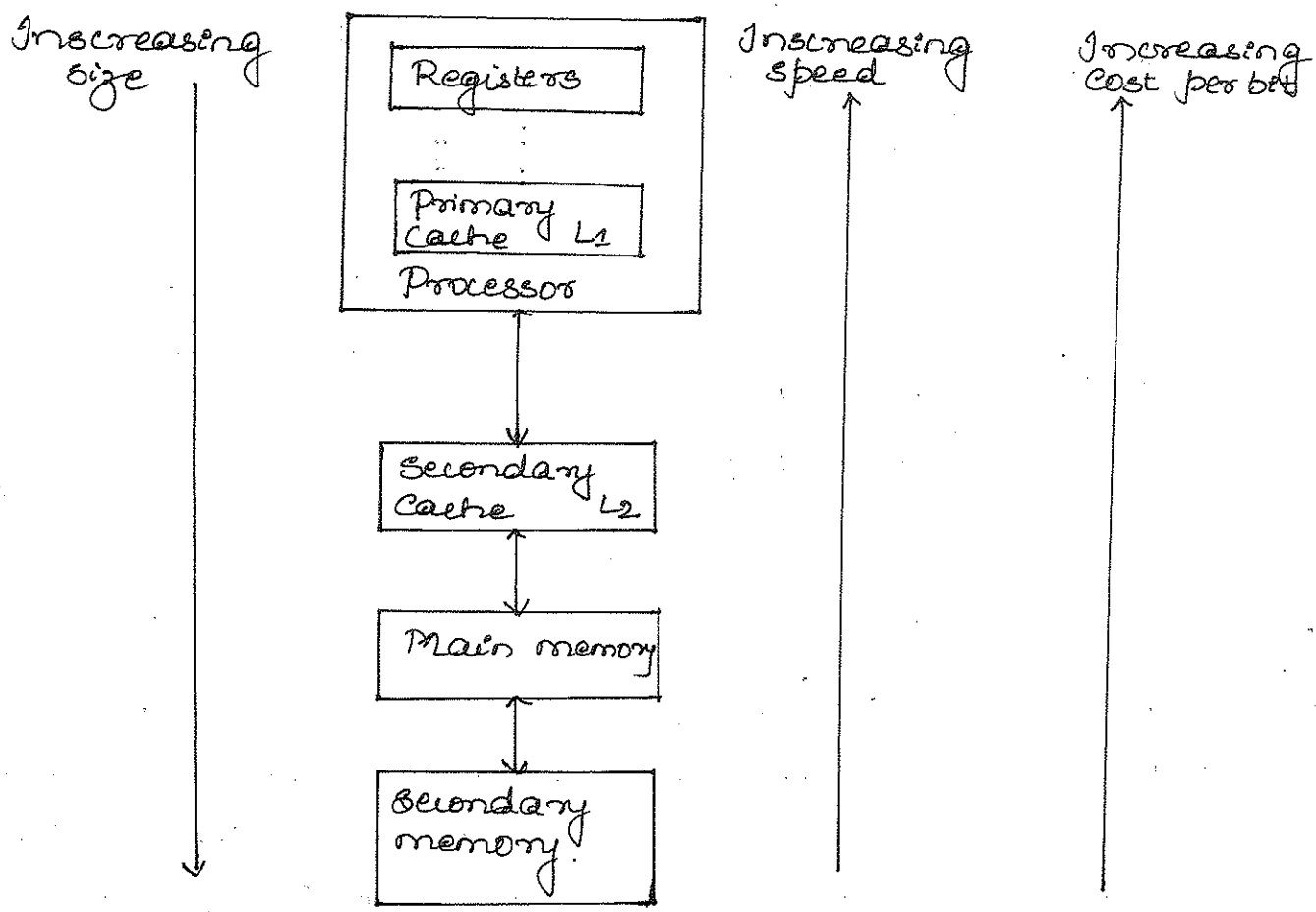
2) Oldest block first algorithm

In this method the oldest block that was fetched is removed to make a way for new block.

3) Random chosen block remove algorithm

In this method a block is randomly chosen in the cache to be replaced for the newblock.

Memory Hierarchy



Medha Gowrayya
Asst. Prof, CSE
RNSIT

Assignment Questions Of Module2 and Module3

1) Explain how interrupt requests from several I/O devices can be communicated to a processor through a single INTR line.
Ans :- Interrupt hardware

2) Explain the different methods of enabling and disabling Interrupts.
Ans :- Three methods.

3) Explain in detail, the situations where a number of devices capable of initiating interrupts are connected to the processor? How to resolve the problems?
Ans :- Processor identifies the device raising the interrupt using two methods.
1) Polling 2) Interrupt vector

4) In a situation where multiple devices capable of initiating interrupts are connected to processor, explain the implementation of interrupt priority, using individual $\overline{\text{INTR}}$ and INTA to all devices.
Ans :- Multilevel priority implementation.

5) How to handle simultaneous request from the device? (when only one INTR is available for all devices)
Ans :- Daisy chain.

6) Explain various methods of handling multiple request?
Ans :- 1) Multilevel priority implementation
2) Grouped priority implementation
3) Interrupt vector
4) Daisy chain

7) When processor is serving Device A, whether device B is allowed to raise an interrupt? If yes then justify the answer with example.

Ans:- Yes. Interrupt nesting (Priority assignment) example of real time clock.

8) Draw the arrangement for bus arbitration using a daisy chain and explain in bref

Ans:- Centralized bus arbitration

9) Define Bus Arbitration. Explain two approaches of bus arbitration.

Ans:- 1) Centralized arbitration
2) Distributed arbitration.

10) What are the different methods of DMA? explain

Ans:- Block mode and cycle stealing.

11) Explain Direct memory access in detail. OR Explain the working of Direct Memory Access.

12) What is an exception? List and explain the different types of exception.

13) With a timing diagram explain Handshake control of data transfer during an input operation.

Ans:- Asynchronous Bus.

14) Explain synchronous Bus.

Ans:- Timing diagram for input transfer as seen by Master and slave.

15) With a timing diagram explain an input transfer using multiple clock cycle.

Ans:- Synchronous bus, input transfer using multiple clock cycle (high frequency clock) and slave ready signal.

1) With a neat sketche explain the internal organization of memory chips.

2) With a neat diagram explain 1K*1 memory chip.

3) Discuss the structure and operation of SRAM and DRAM cell.

4) Draw a diagram and explain the working of 16 mega bit DRAM chip configured as 2M*8. Also explain how it can be made to work in fast page mode.

5) Explain Synchronous DRAMs and timing diagram with respect to burst transfers.

6) Define Latency, Bandwith, read hit, write miss, locality of reference, readmiss and write hit, DDR SDRAMs

7) Briefly explain any four non-volatile memories

8) With a neat figure analyze the memory hierarchy interms of speed cost and size.

9) Give the block diagram for 8M*32 memory chip

10) With a block diagram, explain direct, Associative and set associative mapping between Cache and main memory.

11) Compare and contrast a) SRAM and DRAM
b) Direct mapping, Associative mapping & set Associative mapping.

12) What is memory Interleaving? Explain with an example.

Medha Gowraiyya
Asst Prof, CSE
RNSIT

CO: 4th Module
Arithmetic

There are three ways to represent a positive and negative number.

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

In all the three systems, leftmost bit is 0 for positive no and 1 for negative number.

Positive numbers have identical representation in all the three systems. whereas negative no has three different representation in three system.

In sign and magnitude system negative number is got by changing the MSB from 0 to 1.

In 1's complement system negative numbers are obtained by complementing each bit i,e 0 is changed to 1 and 1 to 0.

In 2's complement system negative number is got by adding 1 to 1's complement.

b_3	b_2	b_1	b_0	sign and magnitude	1's complement	2's complement
0	1	1	1	+7	+7	+7
0	1	1	0	+6	+6	+6
0	1	0	1	+5	+5	+5
0	1	0	0	+4	+4	+4
0	0	1	1	+3	+3	+3
0	0	1	0	+2	+2	+2
0	0	0	1	+1	+1	+1
0	0	0	0	+0	+0	+0
1	0	0	0	-0	-7	-8
1	0	0	1	-1	-6	-7
1	0	1	0	-2	-5	-6
1	0	1	1	-3	-4	-5
1	1	0	0	-4	-3	-4
1	1	0	1	-5	-2	-3
1	1	1	0	-6	-1	-2
1	1	1	1	-7	-0	-1

Binary signed integer representation

Addition & Subtraction of signed numbers

2's complement is the most effecient method for performing addition and subtraction operations. For this consider the graphical method for adding, mod N of positive integers is a circle with N values, 0 through N-1 marked along its parameter. Consider N = 16 (There are 16 no's represented for ~~addition~~ positive no's.

1) Add 7 and 4.

Solun:- (7+4) mod 16 = 11

Locate 7 on circle move 4 steps in clockwise direction we get -5. unsigned representation of -5 is 11.

2) Add (9 and 14

Solun:- (9+14) mod 16 = 7

Locate 9 on circle (unsigned representation of 9 is 1001) which is -7. Move 14 steps in clockwise direction we get 7

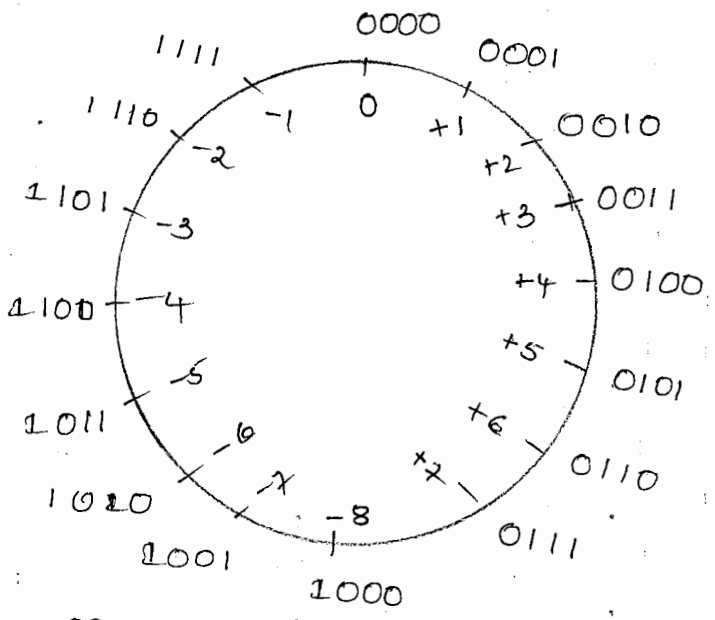
For -ve no

1) Add (+7) and (-3)

Solun:- +7 + (-3) = +4

Locate +7 move move 13 steps in clockwise direction we get +4.

why 13 steps? Because -3 is 1101 in 2's complement whose unsigned representation is +13.



Mod 16 for 2's complement numbers.

Modular system and 2's complement system

Addition and Subtraction

When any arithmetic operation is performed the result should be within the range -2^{n-1} to $+2^{n-1}-1$.

Addition.

1) $(+2)+(+3)=+5$

```
  0010
  0011
------
  0101
```

2) $(+4)+(-6)=-2$

```
  0100
+ 1010
------
  1110
```

3) $(-5)+(-2)=-7$

```
  1011
  1110
------
  1001
```

Subtraction:- (take 2's complement of 2nd no & add it to first no)

1) $(-7)-(-5)=-2$

```
  1001
  0101
------
  1110
```

2) $(-7)-(+1)=-8$

```
  1001
  1111
------
  1000
```

Arithmetic overflow

For a n bit no the result of arithmetic operation should be within the range -2^{n-1} to $+2^{n-1}-1$.

An arithmetic overflow is said to occur when the sign of operands is same and sign of result differs.

Consider five bits for the following examples. For a five bit no the range should be within -16 to $+15$.

1) $(+7)+(+13)=+20$ (overflow out of range)

```
7  = 00111
13 = 01101
-----------
  10100
```

Sign of two operands is 0 (+ve) and result is 1 (-ve) Hence overflow.

2) $(-14) + (+11) = -3$

```
  10010
  01011
 -------
  11101
```

no overflow coz ans is within the range and sign of result matches with sign of first operand.

3) $(-10) + (-13) = -23$ (out of range)

$-10 = 10110$

$-13 = 10011$

$\overline{\quad\quad\quad}$

$\boxed{1}\ 01001$

overflow since sign of result is different from operands.

Multiplication of positive (unsigned) Numbers.

```
        1101
      x 1011
     -------
        1101
       1101
      0000
     1101
    ---------
    10001111
```

manual multiplication algorithm.

→ This algorithm applies to unsigned no's and positive signed no's.

→ The product of two n-digit no's will be 2n digits.

Binary multiplication of positive operands can be implemented in a combinational two dimensional array. The main component in each cell is a full adder FA. The AND gate in each cell determines whether the multiplicand bit m_j, is added to incoming partial-product bit based on the value of multiplier.

Sequential circuit binary multiplier.

Previous diagram shows the hardware arrangment which consist of three registers M, Q and A. A flipflop C to hold carry. An n-bit adder and multiplexer and a control sequencer to generate signals.

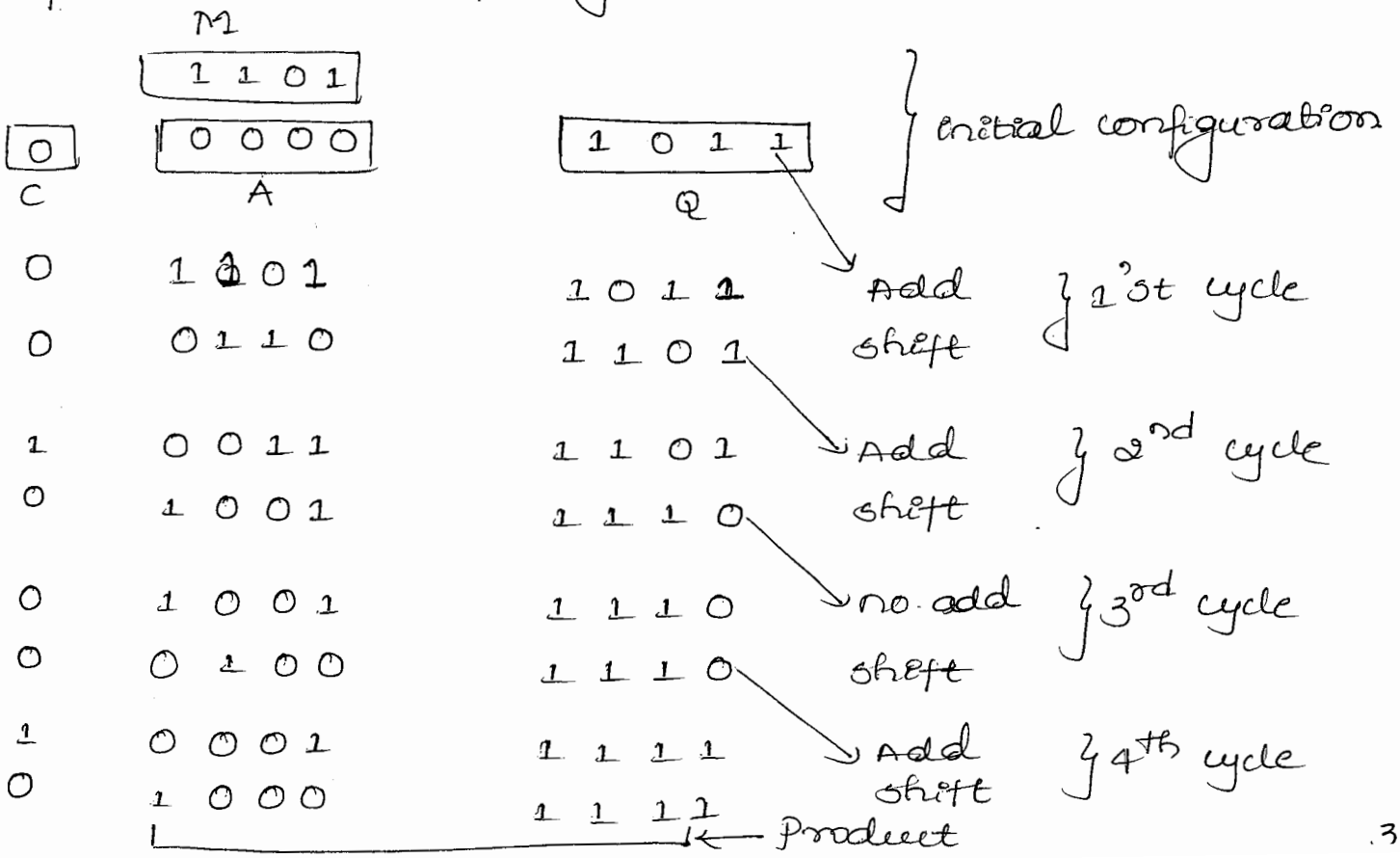
Register Q holds the Multiplier. Register M holds the Multiplicand. Register A holds the partial product which is zero initially.

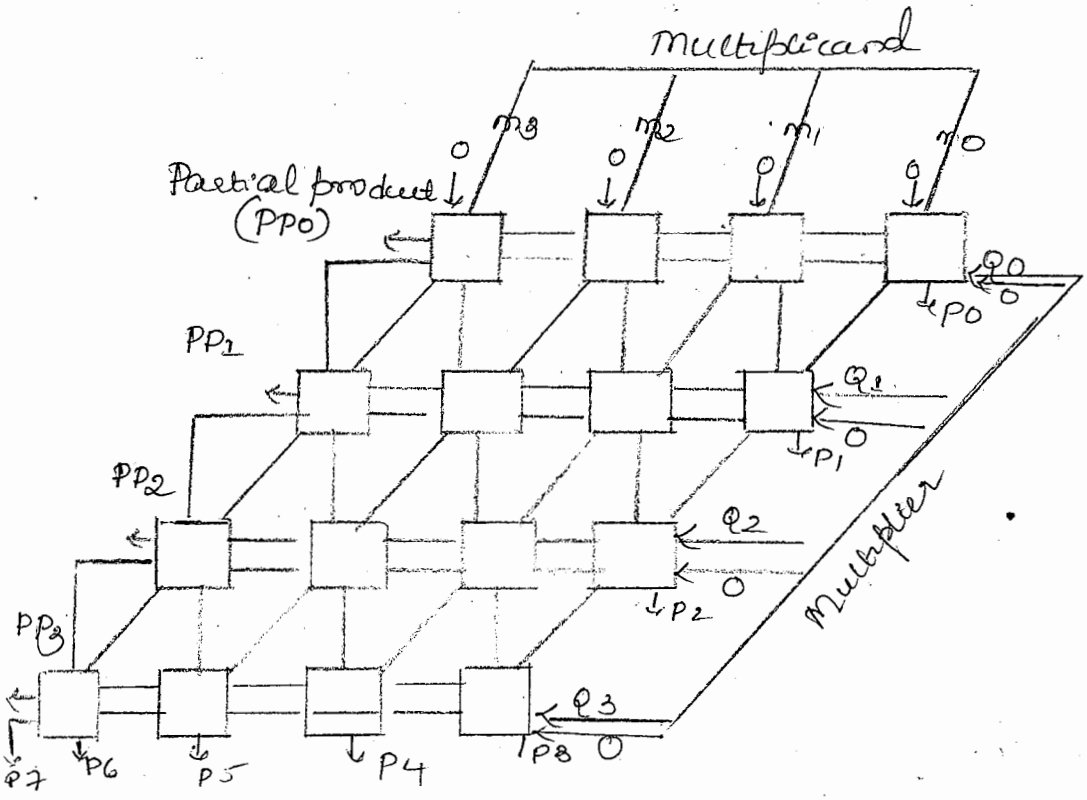
Steps

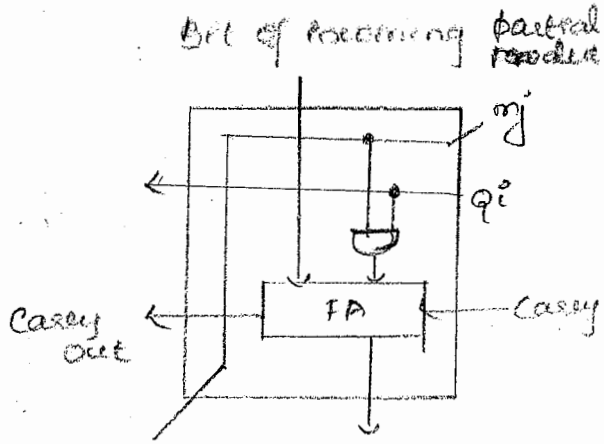
1) Bit 0 of multiplier i.e Register Q is checked.
2) If bit 0 is 1 then multiplicand and partial product are added and all the bits of C, A & Q are shifted to right one bit position.
3) If bit 0 is 0 then no addition is performed, only shift operation is carried out
4) Repeat these steps until it is equal to no of bits.

control sequencer generates add/no add signal based on the bit Q0. It it is 1 then add signal gets generated so the n bit adder adds register M & A stores it in A then performs right shift.

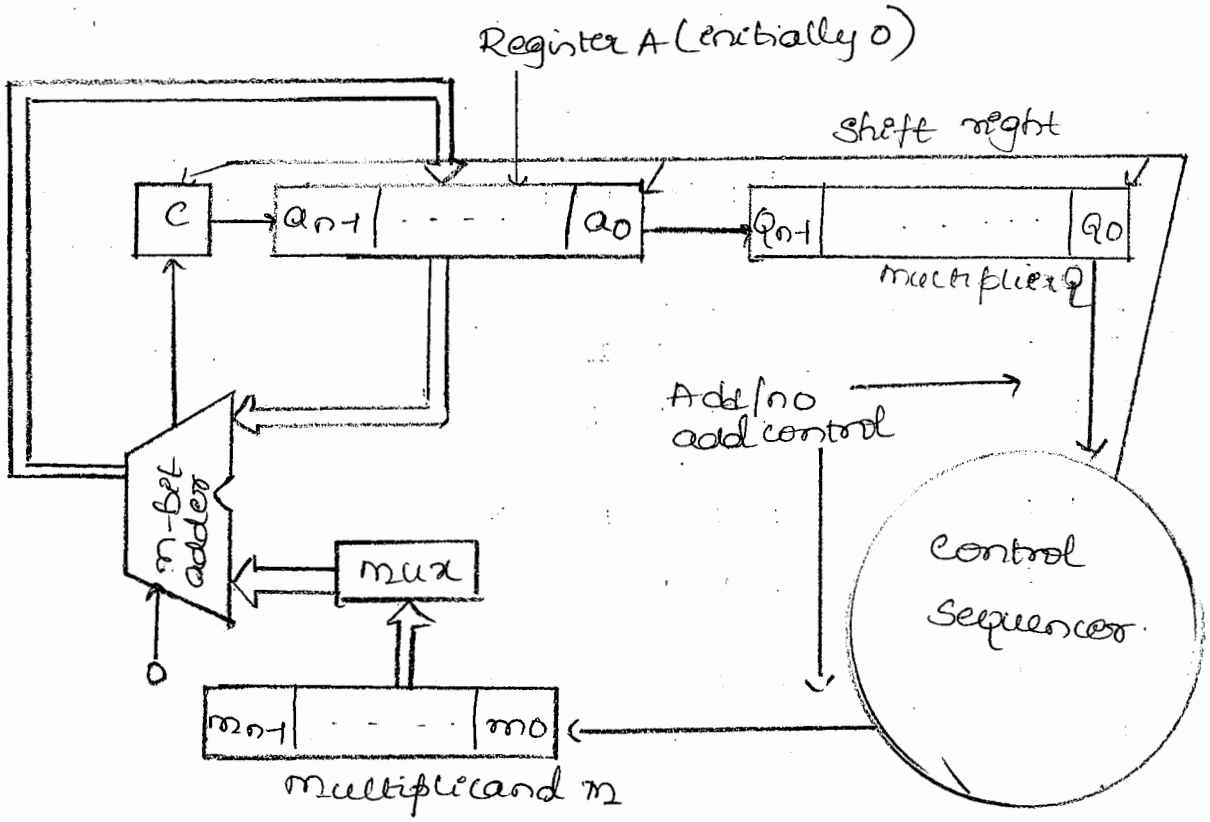
If Q0 is 0 then no add signal is generated which just shifts the content of register A, Q & C to right.







Sequential circuit binary multiplier



Signed operand Multiplication (Booth's Algorithm)

Booth's algorithm generates an product and treats both +ve and -ve no equally.

Booths multiplier recoding table.

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

1) $(+13) \times (-6)$

$+13 = 01101$ $\qquad +6 = 00110$

2's comp $= 10011$ $\qquad -6 = 11010$

For multiplier apply Booth recoded multipler from table

i.e $\underset{\text{Booth's bit}}{1\ 1\ 0\ 1\ 0\ 0}$

$0\ -1\ +1\ -1\ 0$

$01101 \times 0\ -1\ +1\ -1\ 0$

0	0	0	0	0	0	0	0	0	0	
1	1	1	1	1	0	0	1	1	+	(2's complement of multiplicand)
0	0	0	0	1	1	0	1	+		
1	1	1	0	0	1	1	+			
0	0	0	0	0	0	+				
1	1	1	0	1	1	0	0	1	0	(-78)

2) (+14) x (-5)

+14 = 01110
-14 = 10010

+5 = 00101
-5 = 11011

0 -1 +1 0 -1

10010 x 0 -1 +1 0 -1

```
  1 1 1 1 1 1 0 0 1 0
  0 0 0 0 0 0 0 0 0 0 +
  0 0 0 0 1 1 1 1 0 +
  1 1 1 0 0 1 0 +
  0 0 0 0 0 0 +
-------------------------
  1 1 1 0 1 1 1 0 1 0      (-70)
```

Fast Multiplication.

1) Bit-Pair Recoding of multipliers
2) Carry save Addition of summands.

1) Bit-Pair Recoding.

This technique halves the maximum number of summands The no of partial product that gets generated is half the no of bits in multiplier.

c_{i+1}	i	$i-1$	b_i	b_{i-1}	multiplicand selected.
0	0	0	0	0	0 x m
0	0	1	0	+1	+1 x m
0	1	0	+1	-1	+1 x m
0	1	1	+1	0	+2 x m
1	0	0	-1	0	-2 x m
1	0	1	-1	+1	-1 x m
1	1	0	0	-1	-1 x m
1	1	1	0	0	0 x m

i	$i-1$	b_{i-1}
0	0	0
0	1	+1

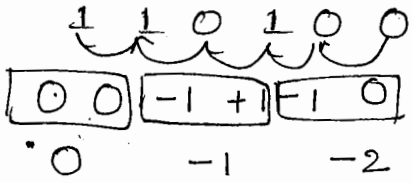
c_{i+1}	i	b_i
0	0	0
0	0	0
0	1	+1

1) (+13) × (−6)

+13 = 0 1 1 0 1 — 6 = 0 0 1 1 0

2's = 1 0 0 1 1 — −6 = 1 1 0 1 0

Booth's Recoded pair



0 1 1 0 1 × 0 −1 −2

1 1 1 1 1 0 0 1 1 0 (2's complement of multiplicand × 2)

1 1 1 1 0 0 1 1 + +

0 0 0 0 0 0 + +

1 1 1 0 1 1 0 0 1 0 (−78)

1 0 0 1 1 × 1 0

0 0 0 0 0

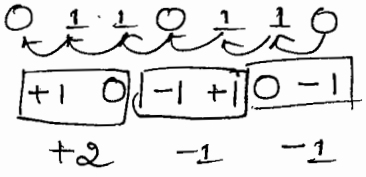
1 0 0 1 1 +

1 0 0 1 1 0

2) (−11) × (+27)

−11 = 1 1 0 1 0 1

2's = 0 0 1 0 1 1 — 27 = 0 1 1 0 1 1



1 1 0 1 0 1 × +2 −1 −1

0 0 0 0 0 0 0 0 1 0 1 1

0 0 0 0 0 0 1 0 1 1 + +

1 1 1 0 1 0 1 0 + + (multiplicand ×2)

1 1 1 0 1 1 0 1 0 1 1 1 (−297)

1 1 0 1 0 1 × 1 0

0 0 0 0 0 0

1 1 0 1 0 1 +

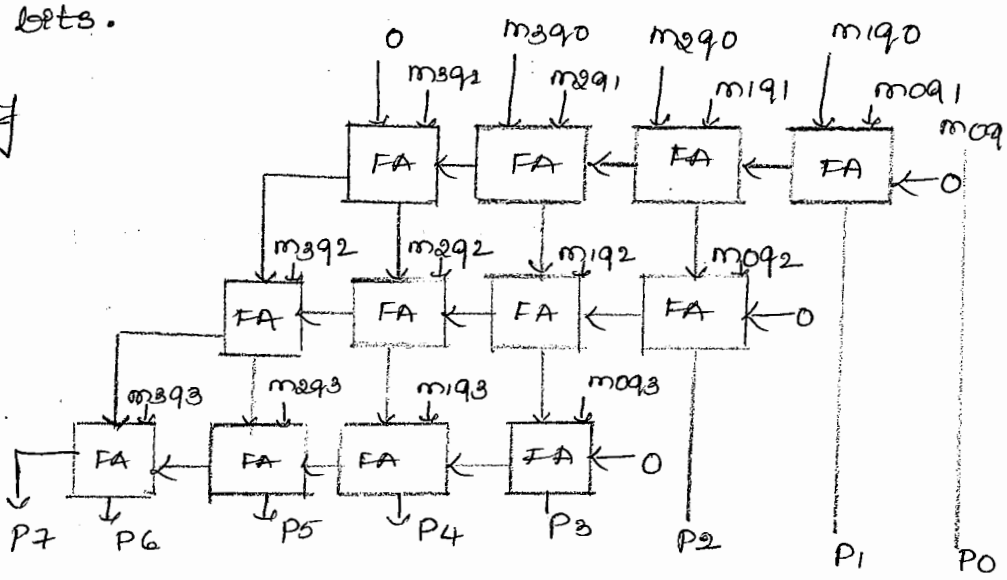
1 1 0 1 0 1 0

Carry Save addition of Summands

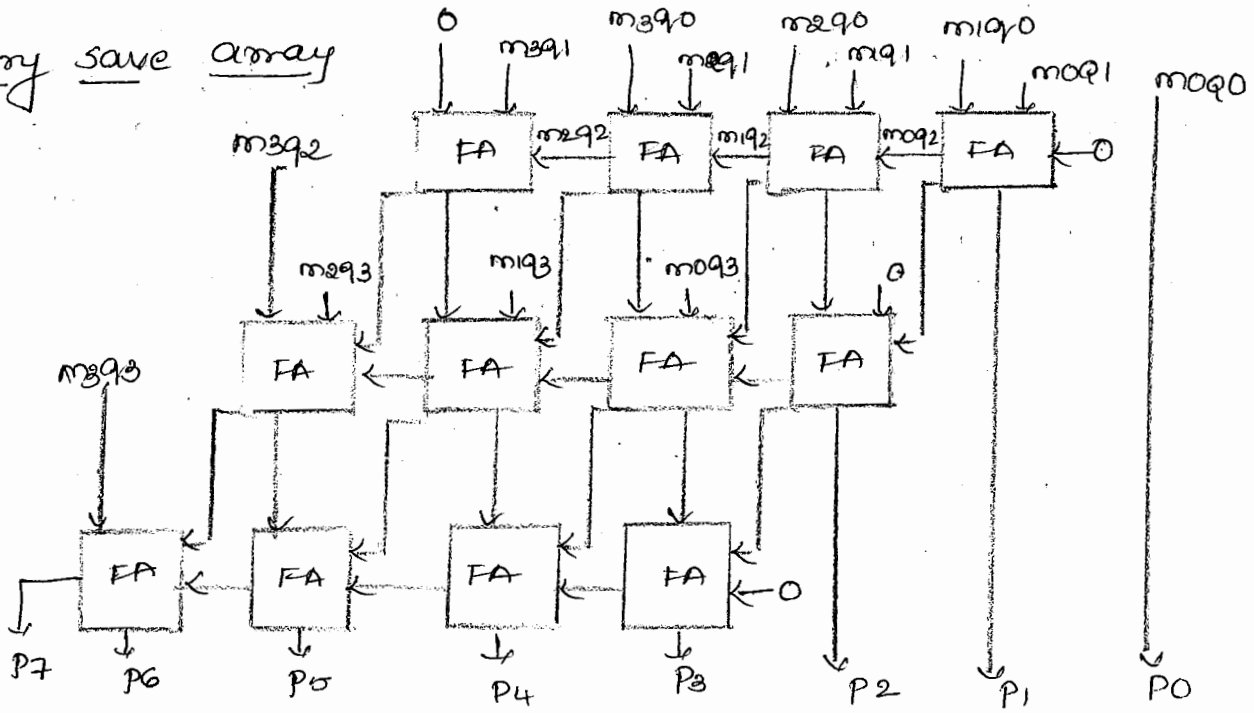
Carry Save addition speeds up the addition process. Instead of letting carries ripple along the rows they can be saved and introduced into the next row in the correct weighted positions. This frees up an input to three full adders in the first row. These inputs are used to introduce the third Summand bit products m_{292}, m_{191} & m_{092}. Now two inputs of each full adder in second row are fed by sum and carry outputs from first row. The third i/p is used to introduce the bits products m_{293}, m_{193} of the fourth Summand.

Higher order bit products m_{392} and m_{393} of 3rd & 4th Summands are introduced to the remaining free inputs at the left end in the 2nd and 3rd rows. The saved carry bits and Sum bits from 2nd row are now added in the third row to produce final product bits.

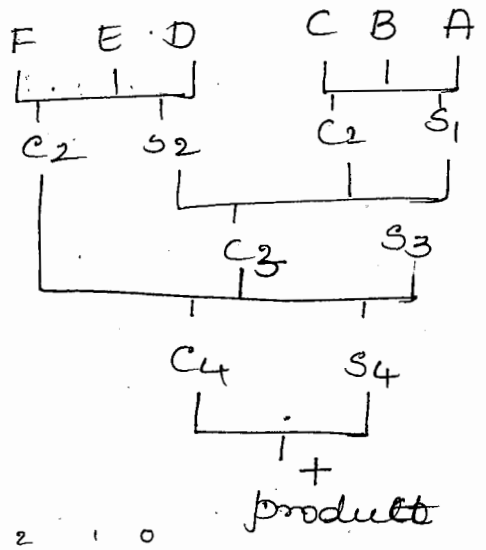
Ripple carry array



Carry save array



Schematic representation of carry-save addition operations



5 4 3 2 1 0

1 0 1 1 0 1 × 1 1 1 1 1 1

							1	0	1	1	0	1		
						1	0	1	1	0	1			
						1	0	1	1	0	1			
						1	1	0	0	0	0	1	1	S_1
					0	0	1	1	1	1	0	0		C_1
				1	0	1	1	0	1					
			1	0	1	1	0	1						
		1	0	1	1	0	1							
		1	1	0	0	0	0	1	1					S_2
	0	0	1	1	1	1	0	0						C_2
					1	1	0	0	0	0	1	1		S_1
				0	0	1	1	1	1	0	0			C_1
			1	1	0	0	0	0	1	1				S_2
		1	1	0	1	0	1	0	0	0	1	1		S_3
	0	0	0	0	1	0	1	1	0	0	0			C_3
	0	0	1	1	1	1	0	0						C_2
	0	1	0	1	1	1	0	1	0	0	1	1		S_4
0	0	1	0	1	0	1	0	0	0	0	0			C_4
1	0	1	1	0	0	0	1	0	0	1	1			product

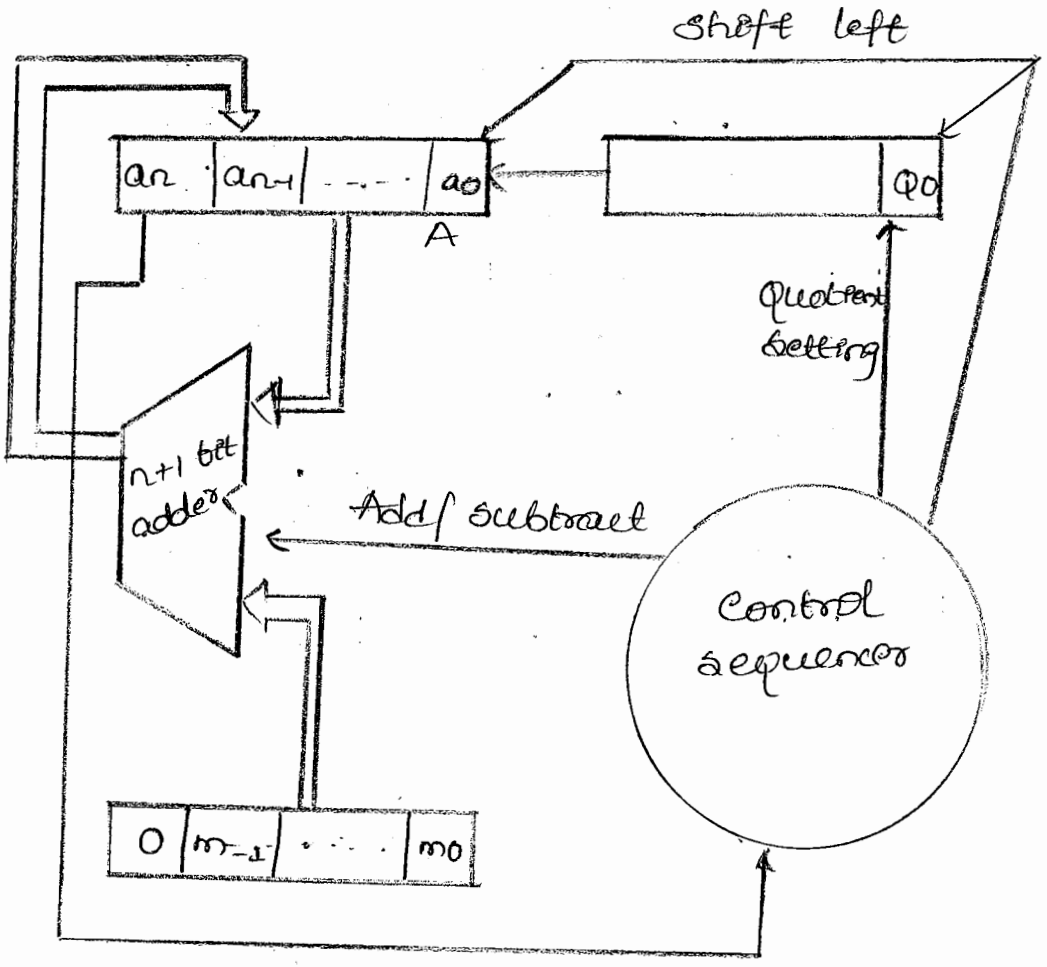
Integer Division.

The circuit arrangment for binary division (restoring divisio consist of register Q, A & M. Register Q holds the Dividend Register M holds the Divisor, A hold the remainder but initially zero. Quotient is stored in Q at the end. An n+1 bit adder is used.

Algorithm for Restoring Division.

1) Shift A and Q left one bit position.
2) Subtract M (Divisor) from A and place the answer in A
3) If the sign of A is 1, then set Q0 to 0, ~~else Q0 to 1~~ and add A & M.
4) If the sign of A is 0, then set Q0 to 1.

circuit arrangment for binary division.



1) Dividend = 1 0 1 0 (Q)

Divisor = 0 0 0 1 1 (m) $-m =$ 1 1 1 0 1

	A register	Q register
Initially	0 0 0 0 0	1 0 1 0
Shift	0 0 0 0 1	0 1 0 _
Subtract m	1 1 1 0 1	
Set Q0	(1) 1 1 1 0	
Restore (A+m)	0 0 0 1 1	0 1 0 [0]
	0 0 0 0 1	
Shift	0 0 0 1 0	1 0 0 _
Subtract	1 1 1 0 1	
Set Q0	(1) 1 1 1 1	1 0 0 [0]
Restore (A+m)	0 0 0 1 1	
	0 0 0 1 0	
Shift	0 0 1 0 1	0 0 0 _
Subtract	1 1 1 0 1	
Set Q0	(0) 0 0 1 0	0 0 0 [1]
Shift	0 0 1 0 0	0 0 1 _
Subtract	1 1 1 0 1	
Set Q0	(0) 0 0 0 1	0 0 1 1

remainder — Quotient

2) Dividend = 1000 Q

Division = 00011 M . 2's complement = 11101

	A	Q
Initially	00000	1000
Shift	00001	000 —
Subtract	11101	
Set q0	(1) 1110	000[0]
Restore	00011	
	00001	
Shift	00010	000 —
Subtract	11101	
Set q0	(1) 1111	000[0]
restore	00011	
	00010	
Shift	00100	000 —
Subtract	11101	
set q0	(0) 0001	000[1]
shift	00010	001 —
Subtract	11101	
Set q0	(1) 1111	0010
Restore	00011	
	00010	quotient

remainder

1 Non-Restoring Division

Algorithm

1) If the sign of A is 0, shift A & Q left one bit position and subtract M from A. Otherwise add M to A
2) Now if sign of A is 0, set Q_0 to 1 else set Q_0 to 0
3) If the sign of A is 1, add M to A

1) Dividend = 1010 (Q)
Divisor = 00011 (M) 2's complement = 11101

	A register	Q register
Initially	0 0000	1 0 1 0
Shift	0 0 0 0 1	0 1 0 –
Subtract	1 1 1 0 1	
Set Q_0	(1) 1 1 1 0	0 1 0 [0]
Shift	1 1 1 0 0	1 0 0 –
Add	0 0 0 1 1	
Set Q_0	(1) 1 1 1 1	1 0 0 [0]
Shift	1 1 1 1 1	0 0 0 –
Add	0 0 0 1 1	
Set Q_0	(0) 0 0 1 0	0 0 0 [1]
Shift	0 0 1 0 0	0 0 1 –
Subtract	1 1 1 0 1	
Set Q_0	0 0 0 0 1	0 0 1 1
	remainder	Quotient

2) Dividend = 1011 (Q)

Divisor = 00101 (M) 2's complement = 11011

	A	Q	
Initially	00000	1011	
Shift	00001	011 —	} 1st cycle
Subtract	11011		
Set Q0	(1) 1100	0110	
Shift	11000	110 —	} 2nd cycle
Add	00101		
Set Q0	(1) 0101	1100	
Shift	11011	100 —	} 3rd cycle
Add	00101		
Set Q0	(0) 0000	1001	
Shift	00001	001 —	} 4th cycle
Subtract	11011		
Set Q0	(1) 1100	0010	
	remainder	Quotient	

```
A    11100
M    00101
-----------
     00001
     -----
     Remainder
```

3) Solve :- Dividend = 1000

Divisor = 00101

Carry - Look Ahead Addition.

A fast adder circuit must speed up the generation of carry signals. The expression for sum (s_i) and carry (c_{i+1}) of stage i are :-

$$s_i = x \oplus y \oplus c_i$$

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

Factoring the second equation into

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

we can write $c_{i+1} = G_i + P_i c_i$

$$G_i = x_i y_i \quad \text{and} \quad P_i = x_i + y_i$$

The expressions G_i and P_i are called Generate & Propagate functions for stage i.

Generate function for stage i is equal to 1, then $c_{i+1} = 1$. This occurs when both x_i and y_i are 1.

Propage function means input carry will produce an output carry when $x_i = 1$ or $y_i = 1$.

All G_i and P_i can be formed independently and in parallel in one logic gate delay after x & y are applied as input to n bit adder.

Each bit stage contains AND gate to form G_i, an OR gate to form P_i, and a three-input XOR gate to form s_i.

Consider the design for 4 bit adder. Carries can be implemented as follows.

$$C_1 = G_0 + P_0 c_0$$

$$C_2 = G_1 + P_1 G_0 + P_1 P_0 C_{in} \quad (\text{or } C_2 = G_1 + P_1 C_1)$$

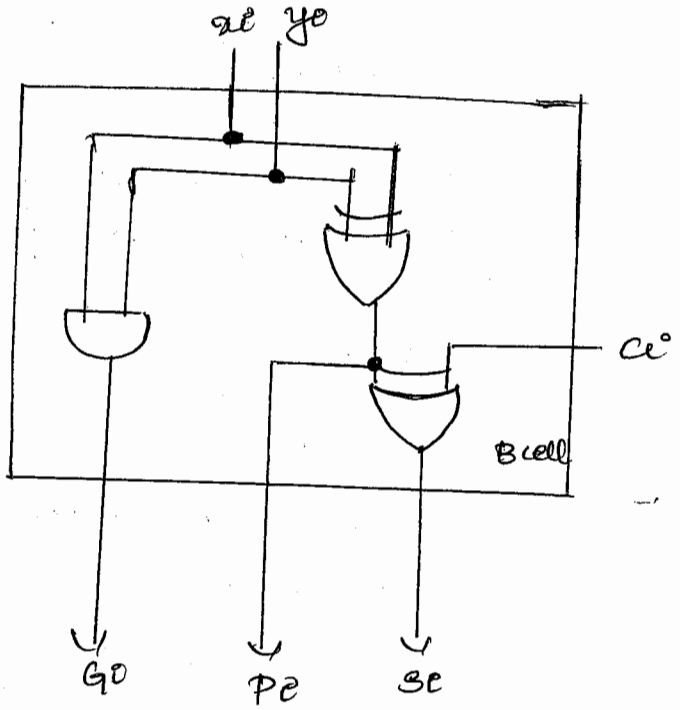
$$C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 c_0$$

$$C_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$$

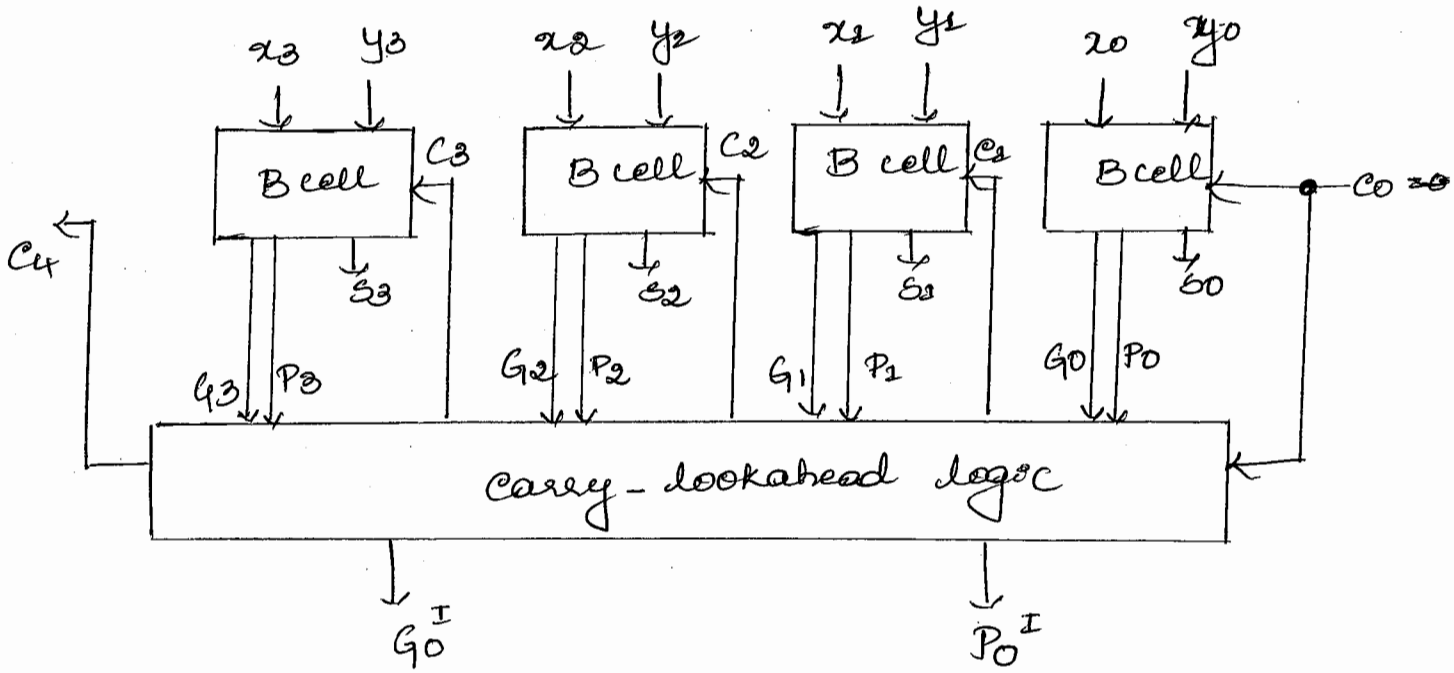
Consider the diagram. Carries are implemented in the block labled carry lookahead logic. An adder implemented in this form is known as carry look ahead adder.

Delay through the adder is 3 gate delays for all carry bits and 4 gate delays for all sum bits.

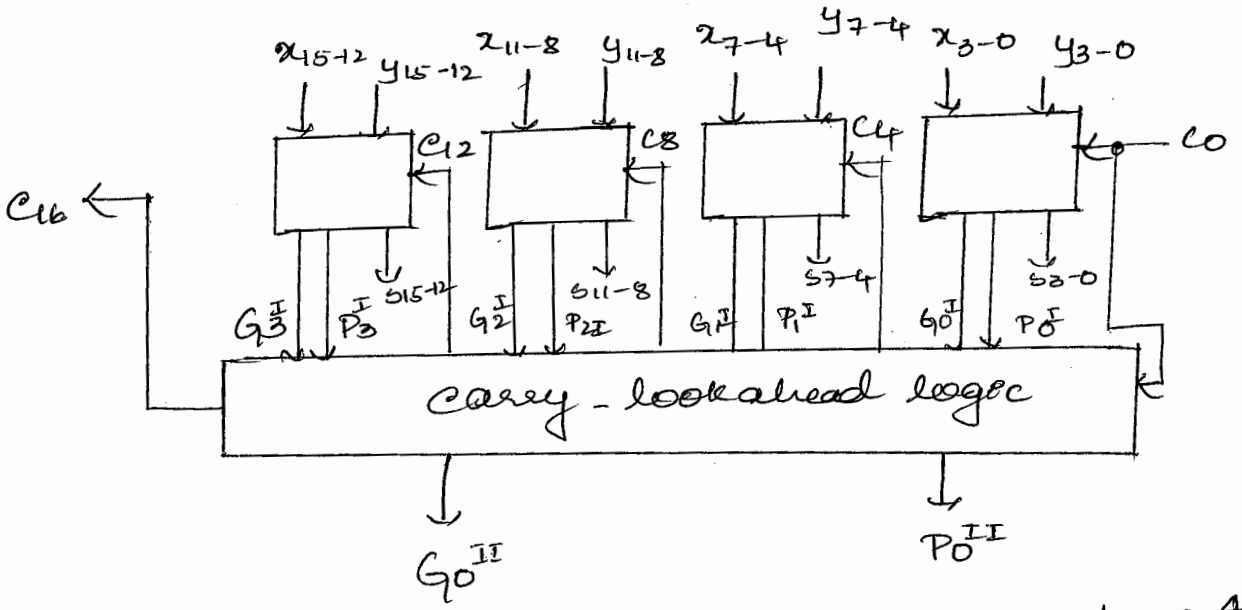
A 4 bit ripple carry adder requires 7 gate delays for s_3 and 8 delay for c_4.



Bit-stage cell



4-bit adder



16 bit Carry-look ahead adder built from 4 bit adder.

This circuit is built from 4 bit adder blocks. These blocks provide new output functions defined as G_k^I and P_k^I where $k=0$ for the first four bit block, $k=1$ for 2nd four bit block and so on.

$$P_0^I = P_3 P_2 P_1 P_0$$

and $$G_0^I = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0$$

$$C_{16} = G_0^I + P_3^I G_2^I + P_3^I P_2^I G_1^I + P_3^I P_2^I P_1^I G_0^I + P_3^I P_2^I P_1^I P_0^I C$$

The first level G_i & P_i functions determine whether bit stage i generates or propagates a carry and second level G_k^I and P_k^I functions determine whether block k generates or propagates a carry

Medha Gourayya
Asst Prof, CSE
RNSIT

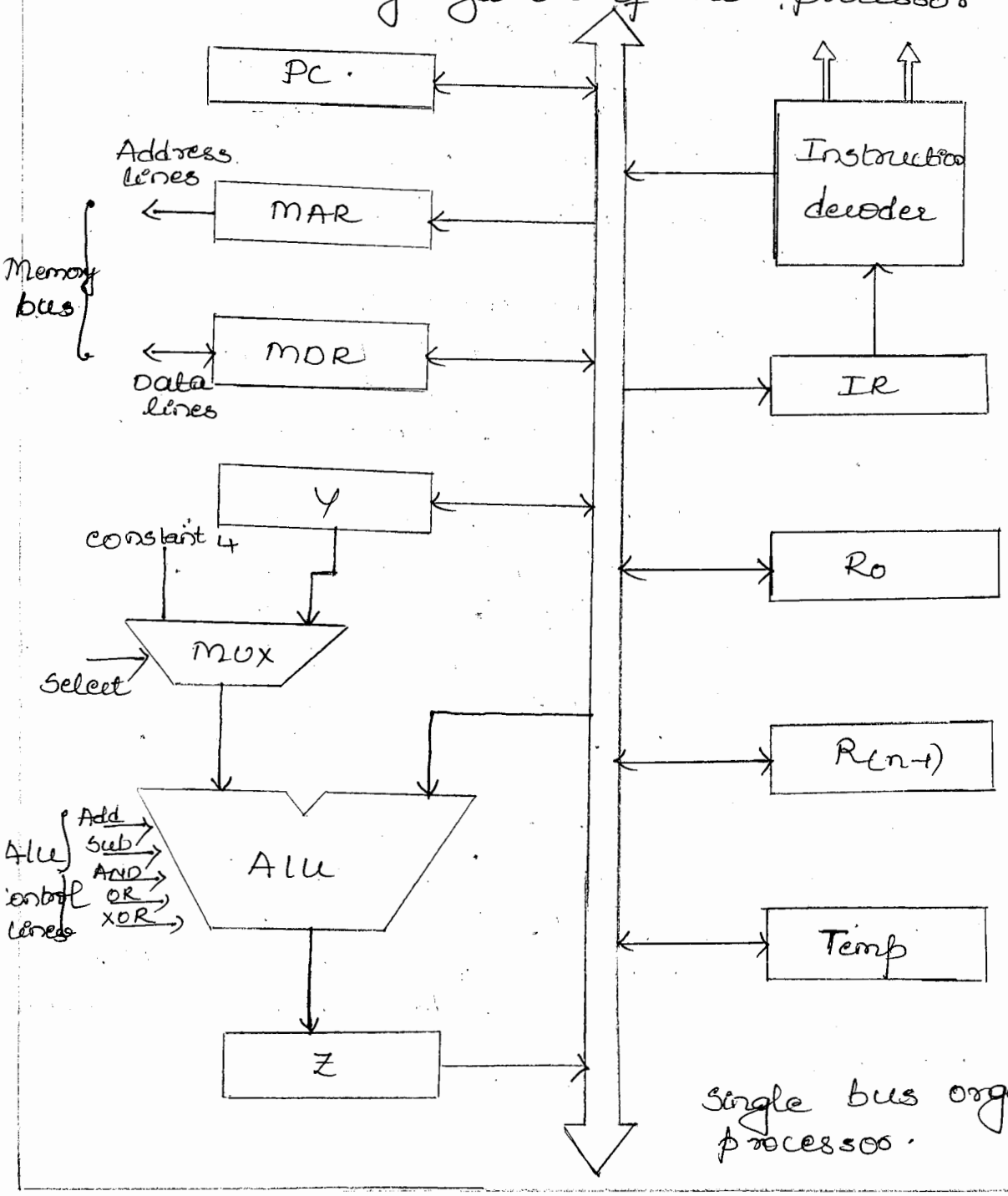
CO : 5th Module

Basic Processing unit

Some Fundamental Concepts

To execute a program processor fetches an instruction one after the other from memory sequentially unless a branch or jump instruction is encountered. To execute a instruction there are four stages, which are Fetch, Decode, Execute (Alu or logical operation), write Back. when the instruction is fetched from memory it gets decoded, to find the operands, meanwhile the content of Pc gets incremented to point to next instruction in memory. Once the operands are read it gets executed using Alu and the result is stored in the destination (write Back).

Internal organization of the processor



Single bus organization of processor.

In this organization all the registers, ALU and control unit is connected via a single bus. (This bus is the internal bus of the processor).

→ The external bus i.e data bus and address bus are connected to the processor bus via MAR and MDR registers.

→ MDR register is bidirectional, It has two inputs and two outputs. So data may be loaded into MDR by processor internal bus and by External memory bus & Data stored in MDR can be given to external memory bus and internal processor bus.

→ Register MAR has one input and one output. It gets the input from processor bus (address) and it sends the address to external address bus.

→ All the general purpose registers R0 to Rn-1 has one input and one output.

→ The three registers X, Y and Z are used by processor for temporary storage of data during execution. Register X recieves the data from bus but it cannot give the data to bus. Register Z gives the data to bus.

→ ALU has two inputs A & B. A is recieved from multiplexer, B is directly from bus.

→ It is a two input multiplexer. One input to a multiplexer is constant4 another input is output of register Y. There are two select lines, Select4 and Select Y. Select4 is selected when content of PC has to be incremented. Select Y is selected during execution of instruction.

→ Output of ALU is directly gated to the register Z.

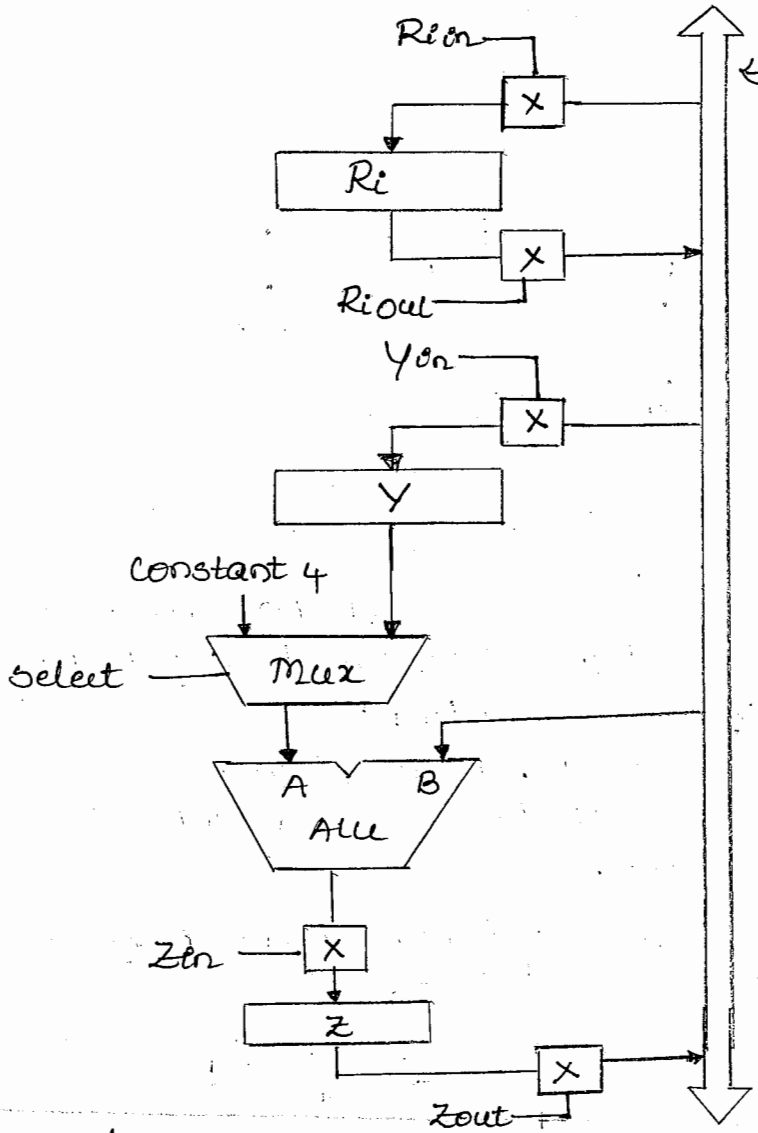
→ Instruction decoder and control logic unit is responsible for decoding the instruction which is present in the register IR. Hence the output of IR is directly connected to this block.

→ Decoder generates the control signals needed to select the registers involved and direct the transfer of data.

Register Transfers

An instruction execution involves a series of steps where the data is transferred from one to another.

Every register has two control signals to place the content of register on to the bus or load the content from bus to register.



→ The input and output of register are connected to bus via switches. The two signals for input and output are Rin and Rout.

→ When Rin is set to 1, the data on the bus is loaded into R.

→ When Rout is set to 1, data from register R is loaded on to bus.

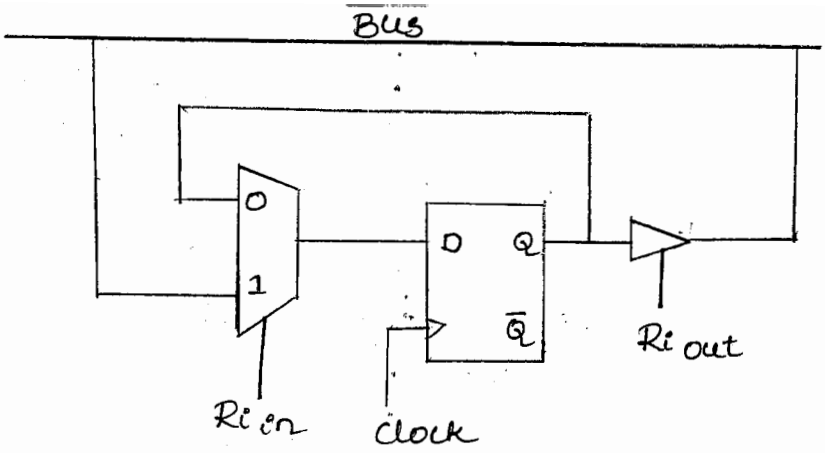
For example transfer the content of Register R_2 to R_4.

1) R_1 out is set to 1. This places the contents of R_1 onto bus.

2) R_4 in is set to 1. This loads the content from bus to R_4

All the operations and data transfers take place within the time periods defined by the processor clock. Registers are made of edge triggered flip flops.

Consider an implementation for one bit of the Register Ri.



Input and output for one register bit

A two input multiplexer is used to select the the data applied to the i/p of edge triggered D flip flop. The two select lines are Riino and Riin1.

→ When Riin is equal to 1, multiplexer selects the data on the bus. This data will be loaded into flip-flop at the rising edge of the clock.

→ When Ri in is zero, multiplexer feeds back the value currently stored in the flip flop.

→ Output of the flip-flop is connected to the bus via a tri-state gate.

→ When Riout is equal to zero, its an open switch, the gate's output is in the high impedance state.

→ When Riout=1 the value (0 or 1) is placed on the bus. i,e the gate drives the bus to 0 or 1 depending on the value of Q.

Performing an Arithmetic operation.

eg:- $R_3 = R_1 + R_2$ (we assume that instruction has been fetched)

→ Alu is a combinational circuit which has no internal storge
→ It performs logical and arithmetic operations on two operands applied to it A and B i/ps.
→ One of the operand is from o/p of multiplexer and other operand is directly obtained from bus.
→ Result of Alu operation is directly gated to the register Z.

The sequence for the instruction $R_3 = R_1 + R_2$ is as follows.

1) R_2 out, Y on
2) R_2 out, select Y, Add, Z en
3) Z out, R_3 in

These are the signals that gets generated. 3 steps occurs in 3 clock cycles.

In the first step content of R_2 is placed on the bus and it is loaded on to the register Y.

In the second step content of R_2 is placed on the bus and it is directly given to ALU as second input. Multiplexer selects the output of Y and gives it to ALU as first i/p. ALU add signal gets generated. o/p of ALU is directly stored on register Z.

In third step content of Z (result) will be placed on the bus and loaded onto the register R_3.

Fetching a word from Memory

Note:- It is not fetching an instruction. Instead it is to fetch an operand after decoding.

→ To fetch (read) a word from memory processor sends the address of memory location where the information is stored and then raises a Read request.

→ Address is sent to MAR register and control signal read gets generated. When the time period is complete data is stored in register MDR. From where it can be transfered to other registers inside the processor.

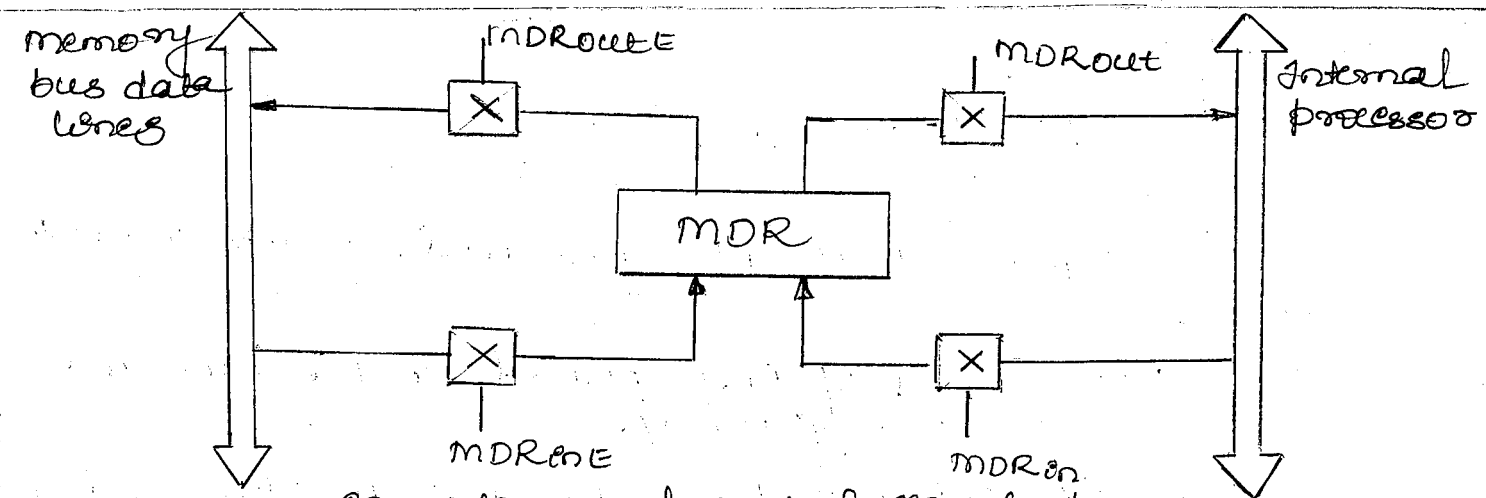
MDR has four control signals. MDRin and MDRout control the connection to internal bus. $MDRin_E$ and $MDROut_E$ control the connection to external bus.

MDRin = 1. Data from internal processor bus is loaded into MDR

MDRout = 1 Data from MDR is loaded onto internal processor bus.

$MDRin_E$ = 1 Data from external bus is loaded onto MDR

$MDROut_E$ = 1 Data from MDR is loaded onto external memory bus.



connection and control signals for register MDR

Example: Move (R1), R2

The actions to be performed

1) Content of R1 (address) should be given to MAR. i,e MAR ← [R1]
2) Initiate a read operation.
3) wait for MFC (memory function complete) signal from memory (Addressed device always sends MFC signal). The generation of MFC signal means the data is present on MDR register.
4) Load MDR from
5) R2 ← [MDR]

Sequence of signals are as follows.

1) R1out, MARin, Read (MAR out is always enabled)
2) MDRinE, WMFC
3) MDRout, R2in

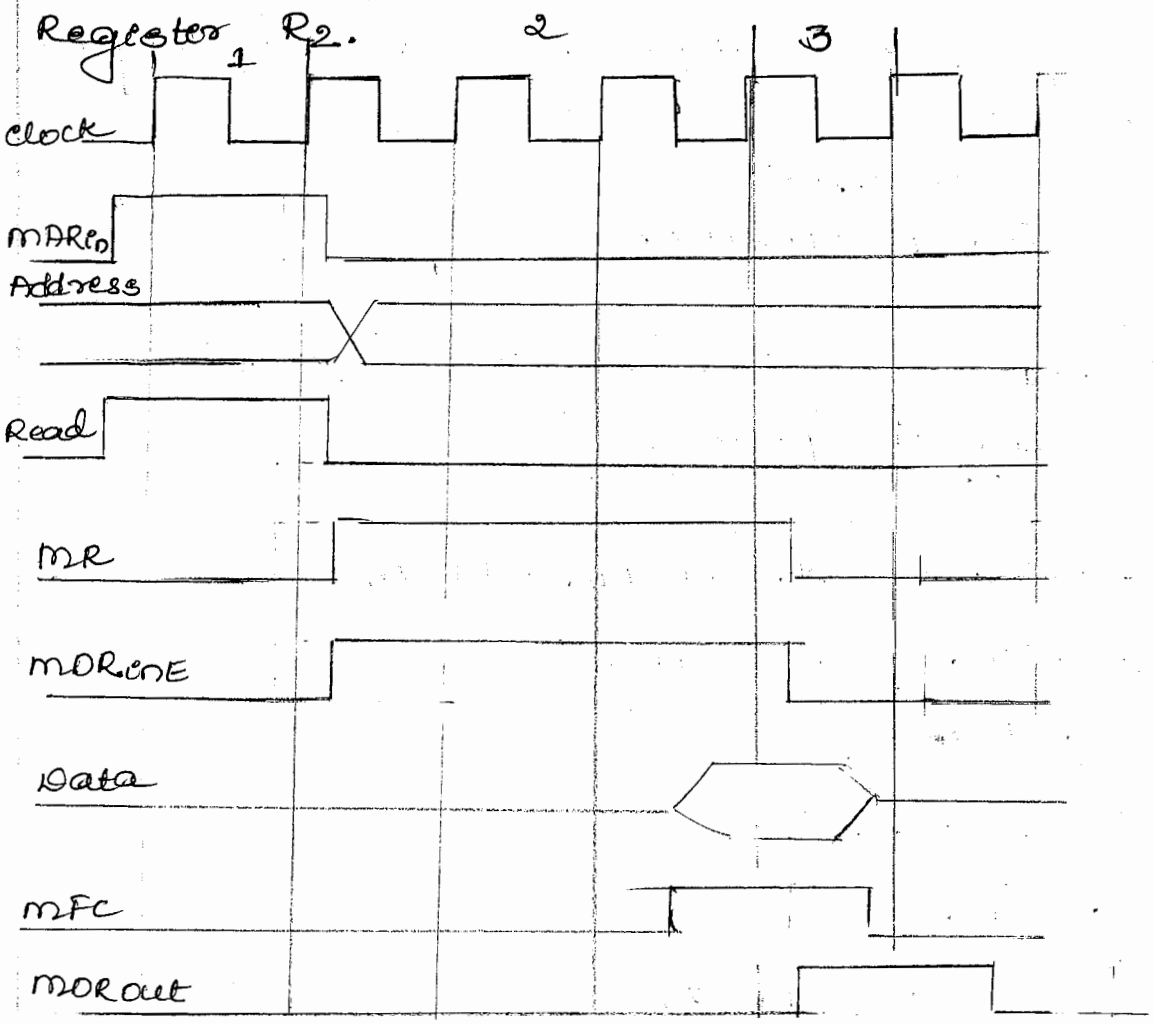
→ Content of R1 is placed on the bus which is received by MAR. Since MARout is always enabled ~~of~~ of MAR is received by external memory address bus. A read control signal is activated at the same time MAR is loaded.

→ This Read signal causes the bus interface circuit to generate an external command called as Memory Read i,e MR on the bus.

→ In the second step data is received from external bus into MDR and a wait MFC signal is activated

→ In third step, only after WMFC signal is generated (which means a valid data is present on MDR) content of MDR

is loaded on to internal processor bus. which is recieved by Register R_2.



Storing a word in Memory

Move $(R_2), (R_1)$

1) R_1out, MARin
2) R_2out, MDRin, Write
3) MDRoutE, WMFC

→ Content of R_1 (address) is loaded in MAR. The data to be written is loaded into MDR and write signal is initiated. In the third step the content from MDR is written to memory and processor waits until it recieves WMFC from memory (addressed device).

Execution of a complete Instruction

To execute an instruction 4 steps are required. They are

1) Fetch the instruction
2) Decode (read the operands)
3) Perform ALU or Logical operation
4) store the result.

Ex:- Add (R3), R1

Sequence of steps are as follows.

Step	Action
1)	PCout, MARin, Read, Select4, Add, Zin
2)	Zout, PCin, Yin, WMFC
3)	MDRout, IRin
4)	R3out, MARin, Read
5)	R1out, Yin, WMFC
6)	MDRout, Selecty, Add, Zin
7)	Zout, R1in, End

Note:- First three steps are for fetch phase which is common to all the instructions. Remaining steps are for Decoding, Execution and write back. One step occurs in one clock cycle.

1) In the first step, contents of PC (address of instruction in memory) are placed on bus, which goes onto MAR and Read signal gets activated. To increment content of PC, constant 4 is selected by mux, Add signal is generated. ALU performs addition on the I/Ps. one I/P is constant 4 and another I/P is content of PC which is available on Bus. Result is stored in Register Z.

2) In 2nd clock cycle updated value of PC which was in Z is placed on bus and recieved by PC and register Y. (It is saved in Y coz, this value is needed during Branch instruction) WMFC signal gets activated.

3) In third clock cycle content of MDR is placed in register IR

4) In fourth cycle content of R_3 (address) is placed on MAR and read signal is activated. (Decoding step)

5) In fifth cycle content of R_1 is placed on the bus, which is stored in Y and waits for MFC

6) In sixth cycle content of MDR is put on bus which becomes the 2nd i/p to ALU. First i/p is selected by mux using Select Y. Addition is performed & stored in Z. (Execution step)

7) In seventh cycle result is placed on bus, and stored in R. End signal causes a next instruction to be fetched.

Branch Instruction.

Branch instruction replaces the contents of Pc with branch target address. The address is got by adding the offset to updated value of Pc. $(-x+[PC])$

Instruction fetch phase is same for all instructions. In fetch phase the updated contents of Pc is also loaded into register Y. (check step 2). Before the instruction is fetched Pc would have got the updated value. Then in the third step instruction will be loaded into IR.

If the instruction which was loaded into IR was a branch instruction then Pc should get the branch target address not the recently updated value. In this case the offset should be added with content of Pc. Content of Pc is on Y. First i/p to ALU is from multiplexer, it selects, Select Y and second i/p is from bus which comes directly to ALU. Offset is calculated by instruction decoder circuit and loaded on to bus.

ALU performs addition places the result on Z. Lastly content of Z is placed on the bus and received by Z.

1) Pcout, MARin, Read, Select4, Add, Zin

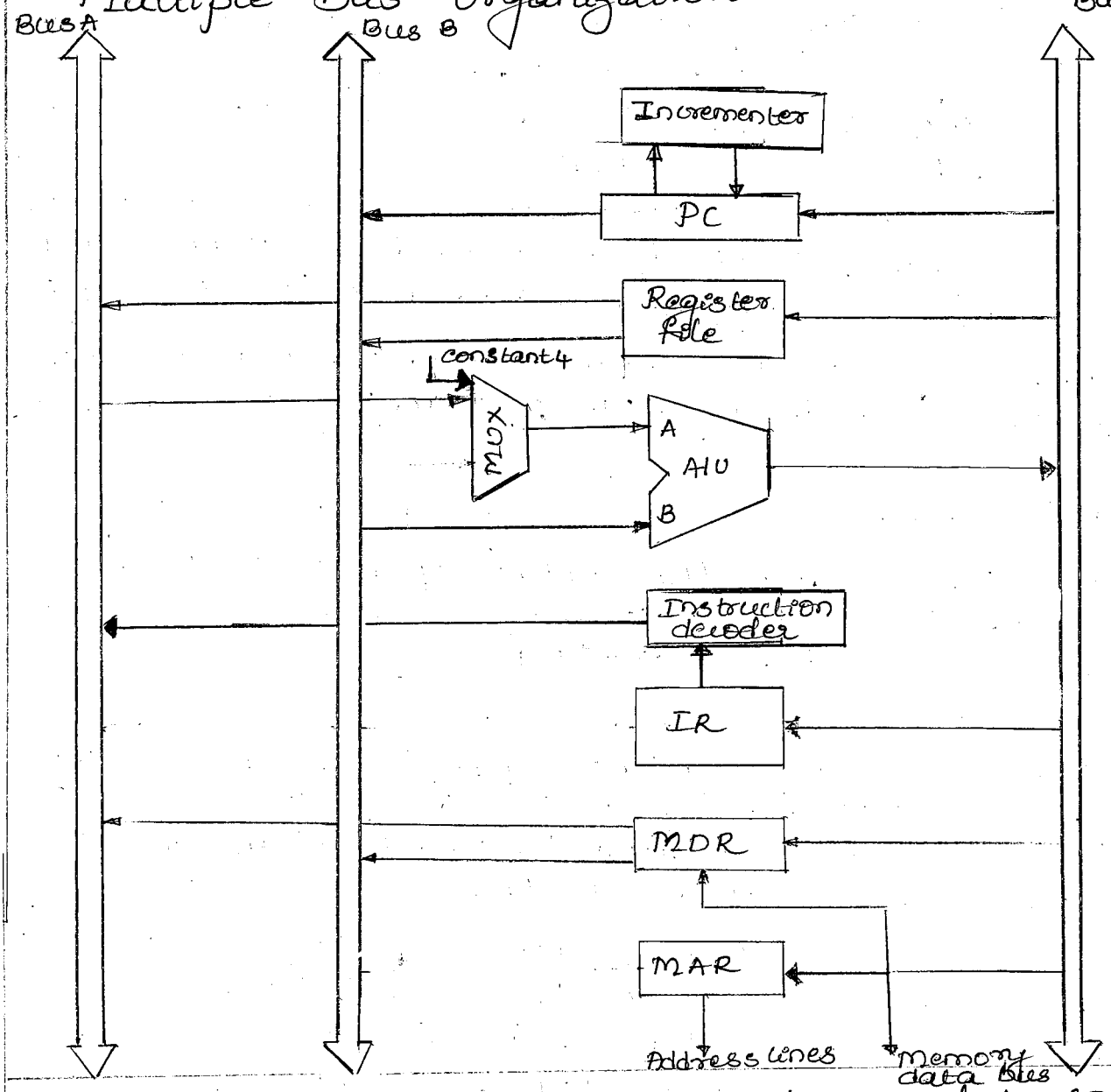
2) Zout, Pcin, Yin, WMFC

3) MDRout, IRin

4) Offset-field-of IRout, Add, Zin

5) Zout, Pcin, End

Multiple Bus Organization



In a single bus architecture only one data can be placed on a bus in one clock cycle which is time consuming. In modern processors a three bus architecture is used where more than one data can be transferred in 1 clock cycle. All the registers are grouped into a single register file. This file has two o/p port and one i/p port. Which means content of 2 registers can be placed on 2 bus (A and B). Third port allows the content from bus to be placed in regist Bus A and B are used to transfer source operands to A & B i/p of ALU. Result is transferred to destination using bus C.

Example:- Add R4, R5, R6

Steps	Action
1)	PCout, R=B, MARin, Read, IncPc
2)	WMFC
3)	MDRoutB, R=B, IRin
5)	R4outA, R5outB, selectA, Add, R6in, End

Note:- R=B means content on Bus B is given to Bus C
R=A means content on Bus A is given to Bus C

In this architecture an Incrementer unit is used which increments the contents of PC by 4. This unit eliminates the need to add 4 to PC by ALU. Constant 4 at ALU I/P mux is still useful to increment other address such as memory address on LoadMultiple and Store Multiple instructions.

In the Step1 to fetch an instruction Add R4, R5, R6, the content of PC is placed on Bus B. The signal R=B causes the content on B (which is content of PC) to be placed on Bus C. From Bus C, content is placed into MAR register. MAR is connected to external address bus. PC gets incremented.

In step 2 WMFC gets activated.

In step 3 content of MDR is placed on Bus B. This content is sent to Bus C by activating the signal R=B. From Bus C, content is placed into register IR.

In step 4 R4 content is placed on Bus A & R5 content placed on Bus B. R4 is given to ALU by mux, R5 is directly given to ALU. Result is stored on R6

Hardwired Control

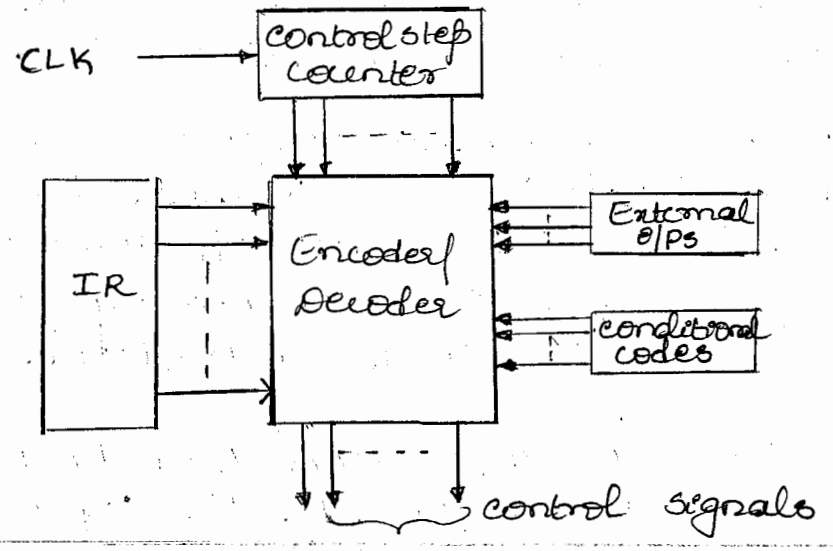
To execute instructions, processor must have some means of generating signals on a proper sequence. There are two approaches. 1) Hardwired control 2) Microprogrammed control.

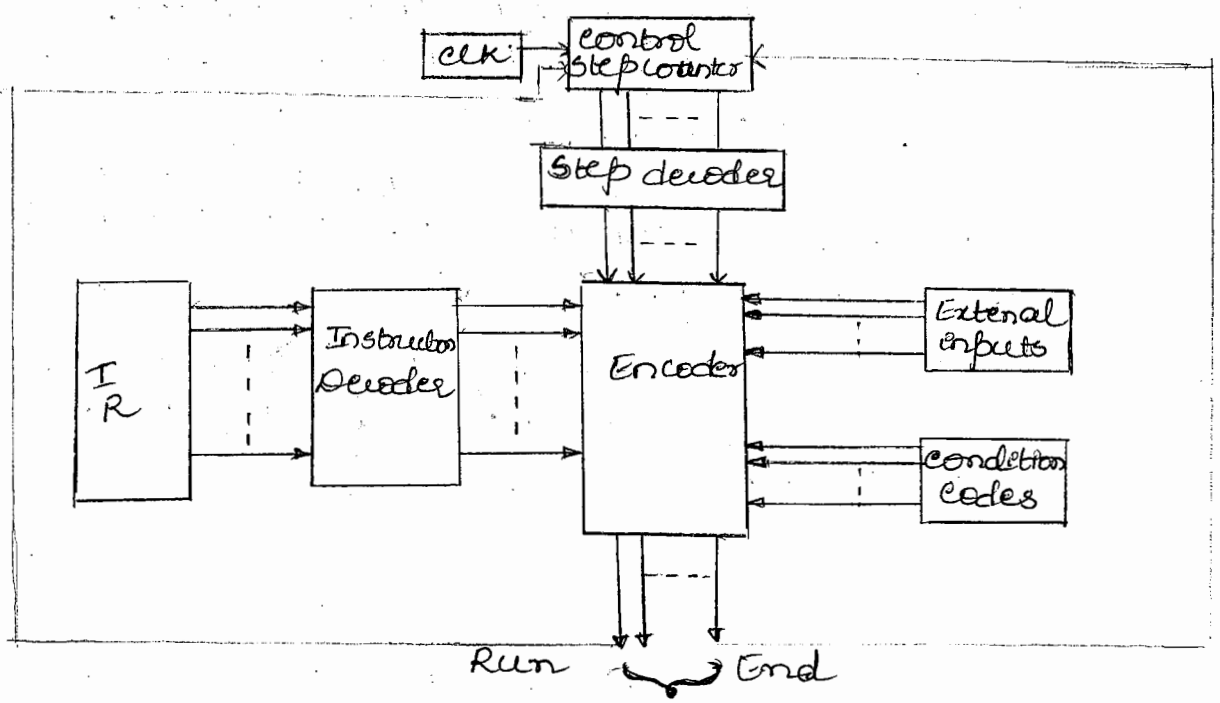
Control sequence consider the control sequence for the instruction Add (R3), R1 and Branch instruction.

Every step in the sequence is completed in one clock period.

The required control signals are determined by

1) contents of control step counter
2) contents of instruction register
3) contents of conditional code flags
4) External I/P signals, such as MFC and interrupt request.





The decoder/encoder block is a combinational circuits that generates the required control outputs depending on the state of all its inputs. Counter is to keep track of steps.

Step decoder provides a seperate signal line for each step, or time slot on the control sequence. i,e at time T_1, step1 gets generated, T_2 step2 gets generated and so on.

IR will have different instruction but one at a time. (Add, XOR, Move ...) Instruction decoder will generate seperate signal for each instruction. So if we assume in instruction set architecture there are 256 instructions then 256 different signals can be generated. (only one will be active at a time)

Encoder circuits combines all the inputs and generate a individual control signal ;e Yin, PCout ... .

For the control sequence Add (R3), R and Branch instruction when decoder generates Zin signal.

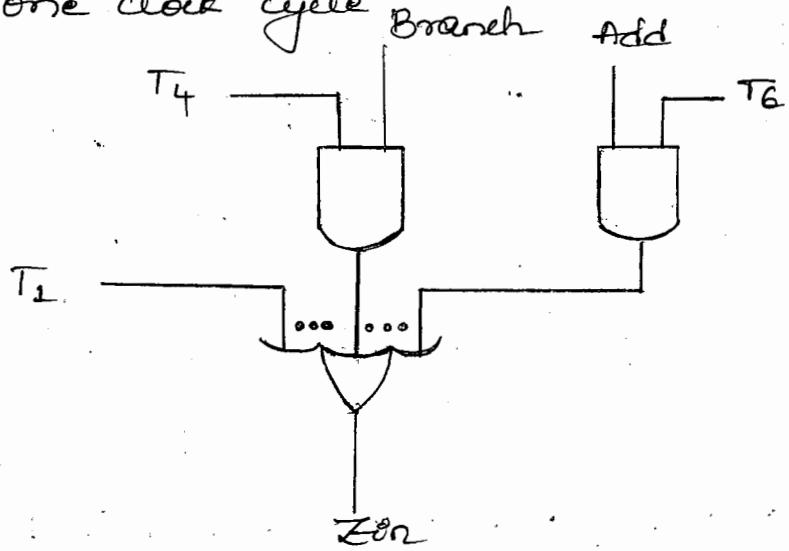
$$Zin = T_1 + T_6 \cdot Add + T_4 \cdot BR + \cdots$$

Zin is generated in time slot T_1 for all the instruction since fetch phase is same for all instruction. During T_6, Zin is generated for Add and during T_4, Zin is generated for Branch instruction.

$$End = T_7 \cdot Add + T_5 \cdot BR + \cdots$$

End signal resets the counter so that next instruction will be fetched.

Run signal is set to 1 causes the counter to be incremented at every clock cycle. When Run is 0 counter stops counting which is required when MFC signal is issued, which causes the processor to wait for the reply from memory for more than one clock cycle



Microprogrammed control

In hardwired, control signals are generated by encoder decoder circuit.

In microprogramed control, signals are generated and stored in memory. As and when required they are read from memory.

Control word is a word whose individual bits represents various control signals.

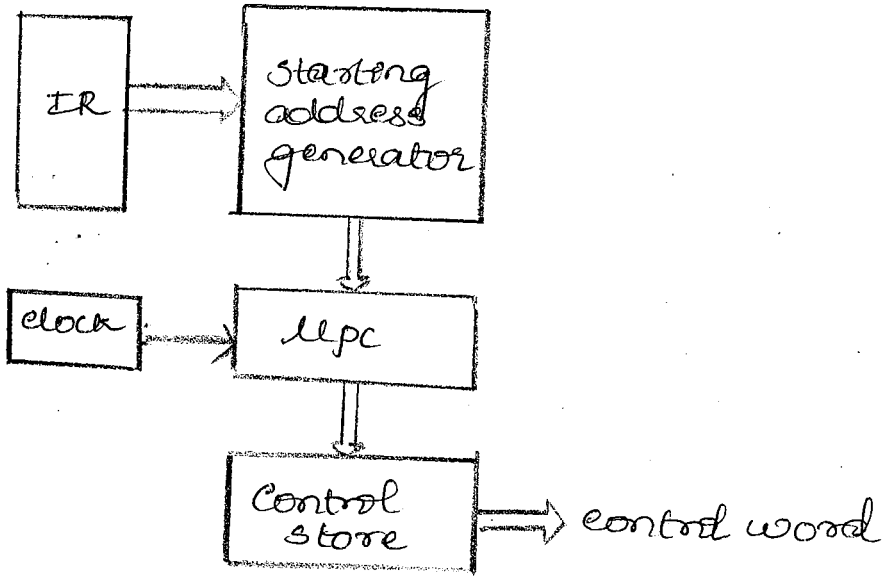
Consider an example of control sequence written for Add (R3), R1.

Seven steps represents seven control word. A sequence of control words which is generated for a individual instruction is known as microroutine for that instruction.

Individual control word in a microroutine represents microinstruction.

An example of microinstructions for the sequence generated for Add (R3), R1

Micro-instructions	..	PCin	PCout	MARin	Read	MDRout	IRin	Yin	Select	Add	Zin	Zout	R1out	R1in	R3out	WMFC	End	..
1		0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	
2		1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	
3		0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	
4		0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	
5		0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	
6		0	0	0	0	1	0	0	1	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	



Basic organization of microprogrammed control unit

The microroutines for all instructions are stored in a special memory called as control store. The control unit can generate the control signals by sequentially reading the control word of the corresponding microroutine from the control store.

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

When a new instruction is loaded into IR, starting address generator, generates the starting address of the microroutine in control store.

The starting address is stored in μpc. on application of clock i,e every clock cycle the μpc gets incremented and control words are read sequentially from control store.

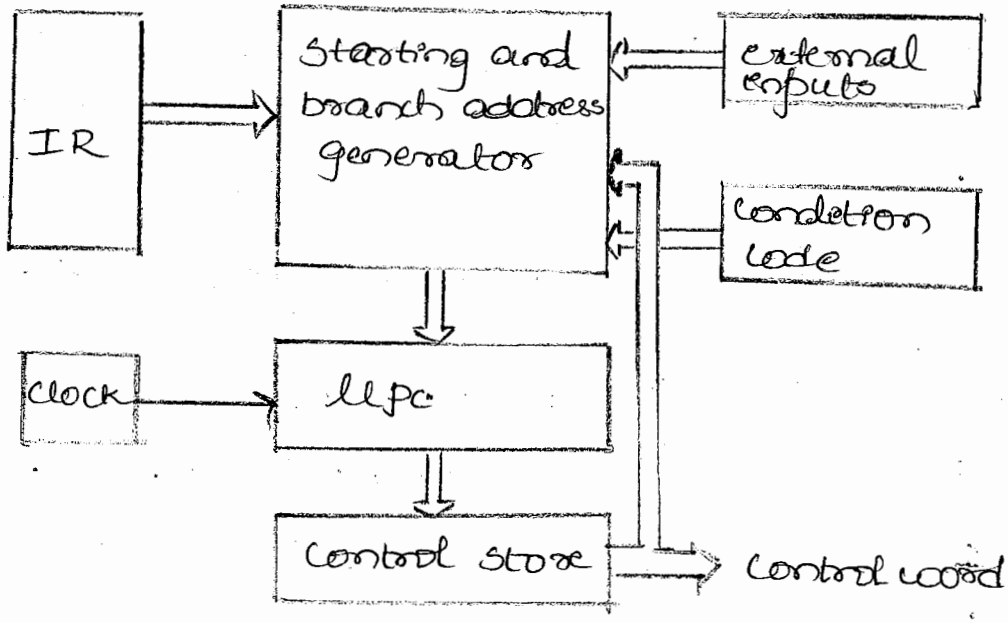
Microroutine for instruction Branch <0

Address	Microinstruction
0	PC_{out}, MARin, Read, Select4, Add, Zin
1	Zout, PCin, Yin, WMFC
2	MDRout, IRin
3	Branch to starting address of appropriate microroutine.
. . .	. . . . .
25	If N=0, then branch to microinstruction 0
26	Offset-field-of IRout, Select4, Add, Zin
27	Zout, PCin, End.

when the instruction Branch<0 is loaded into IR, in the above example we assumed that the appropriate routine for Branch<0 is in the address 25 of control store.
Since it is conditional branch, if N=0 (i,e result of arithmetic operation is -ve then N=1 else N=0) then branch to microroutine 0. (Fetch the next instruction).
If not (N=1) then Branch target address should be ~~address~~ loaded into PC. So to calculate the branch target address by adding content of PC to offset & store back in PC. (This happens in the address 26,27).

---

organization of the control unit to allow conditional branching in microprogram.



To support microprogram branching the control unit is modified. Starting address generator block becomes Starting and branch address generator.
This block loads a new address into μpc when the microinstruction instructs the starting address & branch address generator to do so.
To allow conditional branch, inputs to the block consist of external inputs and conditional codes and also Instruction register.

Microinstructions

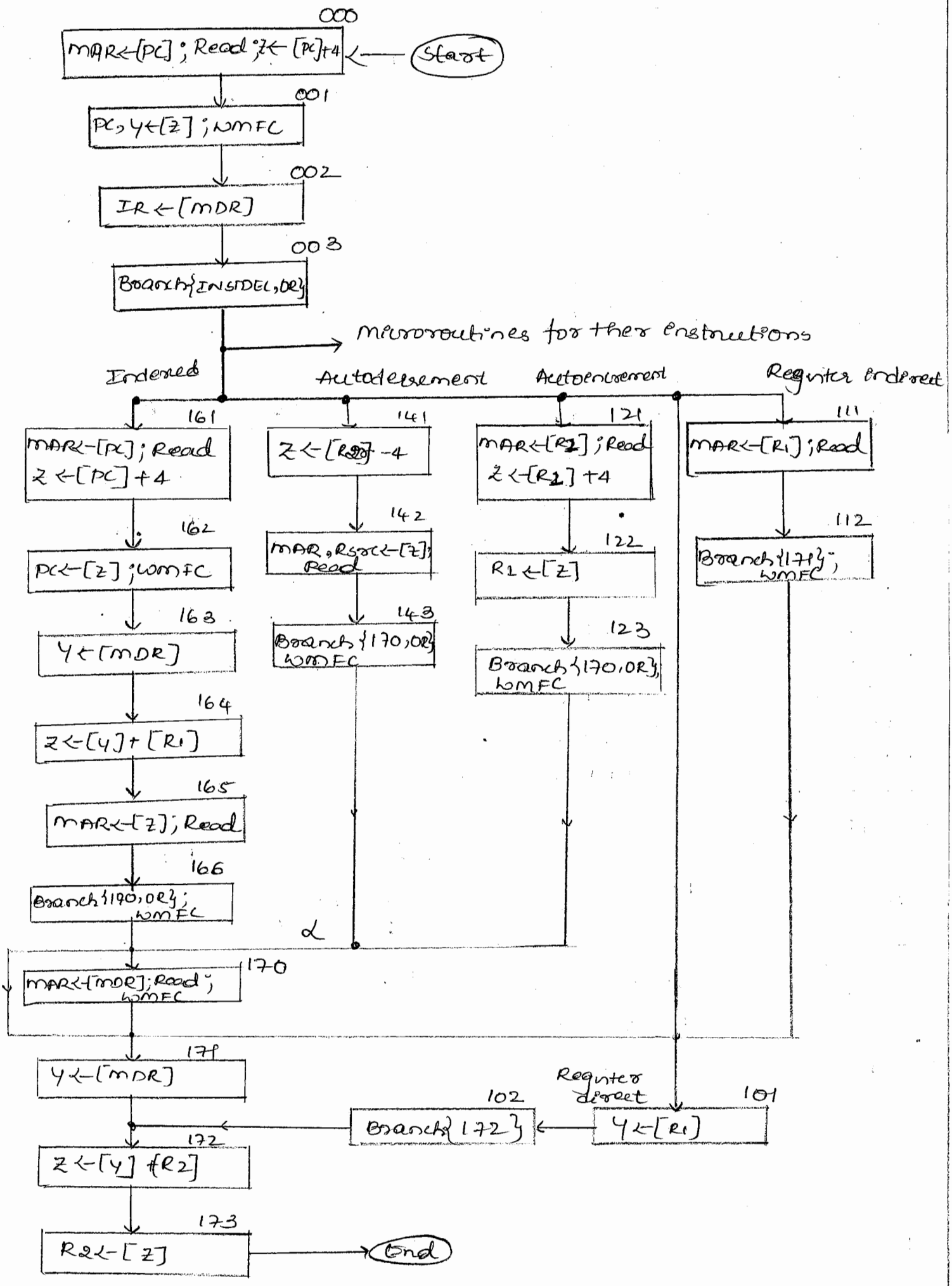
A simple way to structure microinstruction is to assign one bit position to each control signal.
Assigning individual bits to each control signal signal results in long microinstruction. Mor
To avoid this the signals are grouped into different fields. Signals are grouped in such a way that the signals are mutually exclusive i,e only one signal in a field can be active at a time.

Microinstruction

F1	F2	F3
(4 bits)	(3 bits)	(3 bits)
0000: No transfer	000: No transfer	000: No transfer
0001: PCout	001: PCin	001: MARin
0010: MDRout	010: IRin	010: MDRin
0011: Zout	011: Zin	011: TEMPin
0100: R0out	100: R0in	100: Yin
0101 R1out	101: R1in	
0110: R2out	110: R2in	
0111: R3out	111: R3in	
1010: TEMPout		
1011: Offset out		

F4	F5	F6	F7 (1 bit)
(4 bits)	(2 bits)	(1 bit)	0: No action
0000: Add	00: No action	0: Select y	1: WMFC
0001: Sub	01: Read	1: Select4	
.	10: write		
.			
1111: XOR			

F8 (1 bit)
0: Continue
1: End



Flow chart for microprogram Add R1, R2 instruction

The flow chart is for add instruction, which represents the microroutines for different addressing mode of add instruction.

We will consider the microroutine for add instruction in autoincrement mode.

i,e Add (R1)+, R2

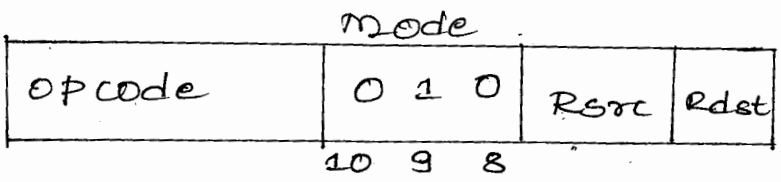
The address 000, 001, 002 are for fetch, which is same for all the instruction.

The address 003 is a branch instruction. which decides the jump to the appropriate routine.

Instruction Decoder determines the address of the appropriate microroutine in control store. For the above example, since it is a Add instruction a jump is taken to the address 101.

Now there is a need to determine the microroutine for the autoincrement addressing mode. So for that 10th bit and 9th bit of IR register is added to 5th bit and 4th bit of μPC.

Contents of IR



In IR register three bits i,e 10, 9 & 8 are used to specify modes.

10th	9th bit	
1	1	→ this indicates indexed addressing mode
1	0	→ autodecrement addressing mode
0	1	→ autoincrement addressing mode
0	0	→ register addressing mode.

If 8th bit is 1 then it is indirect form of the corresponding addressing mode.

Ex:- 111, 101, 011, 001 → indirect indexed, autodecrement autoincrement and register mode.

Ex:- 110, 100, 010, 000 → direct indexed, autodecrement autoincrement and register mode.

· $\mu pc = 101$ (octal)

8	7	6	5	4	3	2	1	0	
0	0	1	0	0	0	0	0	1	← μpc (101)
						1	0		← (10th & 9th bit of IR)

BitOR

0	0	1	1 0 0 0 0 1 ← μpc

Set the 3rd bit of μpc by 8th bit of IR i.e

$\mu pc_3 \leftarrow [\overline{IR_{10}}] \cdot [\overline{IR_9}] \cdot [IR_8]$

$0 \cdot 1 \cdot 0 = 0$

$\mu pc = 001\ 100\ 001$

$1\quad 4\quad 1$

Take a branch to 141 address which is the routine for autoincrement addressing mode.

→ At the address 143, once again there is a branch instruction to decide whether to execute the instruction at address 170 are to skip.

→ Micro instruction at address 170 gets executed only if it is indirect form of addressing mode. In our example it is direct form of autoincrement mode hence 170 has to be bypassed.

μpc contents = 170 i.e 001 111 000 ← μpc (170)

$\mu PC_0 = [\overline{IR_8}]$ = bit OR 1

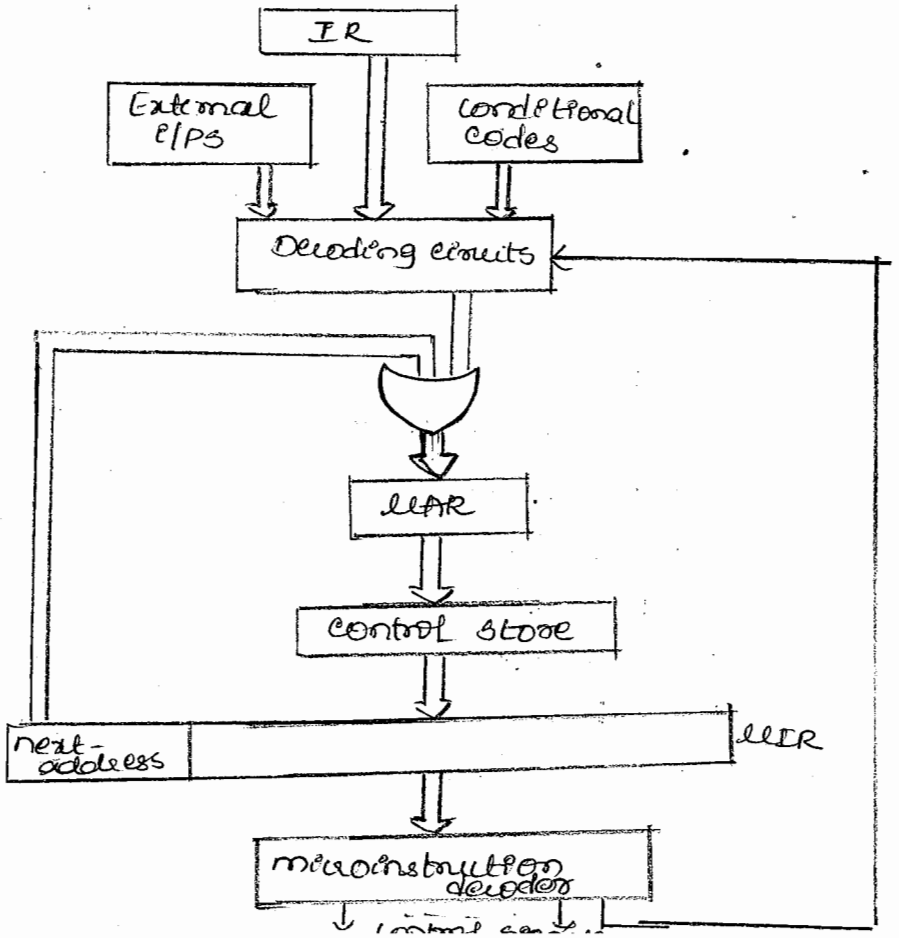
$\overline{IR_8} = 1$

001 111 001 (171)

(Octal) Address	Microinstruction
000	PCout, MARin, Read, Select4, Add, Zin
001	Zout, PCin, Yin, WMFC
002	MDRout, IRin
003	uBranch { uPC ← 101 (from Instruction decoder); $uPC_{5,4} \leftarrow [IR_{10,9}]$; $uPC_3 \leftarrow [\overline{IR_{10}}] \cdot [\overline{IR_9}] \cdot [IR_8]$
121	R1out, MARin, Read, Select4, Add, Zin
122	Zout, R1in
123	uBranch { uPC ← 170; $uPC_0 \leftarrow [\overline{IR_8}]$ }, WMFC
170	MDRout, MARin, Read, WMFC
171	MDRout, Yin
172	R2out, SelectY, Add, Zin
173	Zout R2in, End

For our example 170 will not execute.

Microinstructions with Next-Address Field



In the previous flowchart there are several branch microinstructions.
These microinstructions are needed just to find the address of next microinstructions. So to avoid this extra branch routine an alternate approach is used where the address field is included as a part of every microinstruction. This address specifies the from where the next microinstruction has to be fetched. (so every instruction becomes a branch instruction also). Only problem in this approach is extra bits are needed to specify address.

Since every microinstruction holds the address of next instruction μpc is not needed to hold the starting address (sequential) of microinstructions.
μpc is replaced by micro address register which is loaded from next address field of every microinstruct

Next-address is fed into μAR by OR-gate, so that address can be modified based on external I/P, conditional codes.
Decoding circuits will generate the starting address of microroutine based on opcode in IR.

For this approach several extra signals are needed.
Ex:- OR mode to set of bit-ORing is used.
OR indsrc → is set to 1 if indirect addressing mode is used.

Microinstruction

F0 (8 bits)	F1 (3 bits)	F2 (3 bits)
Address of next instruction	000 : NO transfer 001 : PCout 010 : MDRout 011 : Zout 100 : Rsrcout 101 : Rdstout 110 : Temp out	000 : NO transfer 001 : PCin 010 : IRin 011 : Zin 100 : Rsrcin 101 : Rdstin

F3 (3 bits)	F4 (4 bits)	F5 (2 bits)
000 : No transfer	0000 : Add	00 : No action
001 : MARin	0001 : Sub	01 : Read
010 : MDRin	:	10 : write
011 : Tempin	:	
100 : Yin	1111 : XOR	

F6 (1 bit)	F7 (1 bit)	F8 (1 bit)
0 : Select Y	0 : No action	0 : Next Addr
1 : Select 4	1 : wmFC	1 : Instdec

F9 1bit	F10 1bit
0 : No action	0 : No action
1 : OR mode	1 : OR indsrc.

Implementation of microroutine for sequence Add (R1)+, R2 using encoded signals

octal address	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
000	00000001	001	011	001	0000	01	1	0	0	0	0
001	00000010	011	001	100	0000	00	0	1	0	0	0
002	00000011	010	010	000	0000	00	0	0	0	0	0
003	00000000	000	000	000	0000	00	0	0	1	1	0
121	01010010	100	011	001	0000	01	1	0	0	0	0
122	01111000	011	100	000	0000	00	0	1	0	0	1
170	01111001	010	000	001	0000	01	0	1	0	0	0
171	01111010	010	000	100	0000	00	0	0	0	0	0
172	01111011	101	011	000	0000	00	0	0	0	0	0
173	00000000	011	100	000	0000	00	0	0	0	0	0

QUESTION BANK - III (Module – 4 & 5)

1. Convert the following pairs of decimal numbers to 5 bit, signed, 2's complement binary numbers and add them. State whether or not overflow occurs in each case.

i) 7 and 13
ii) –5 and 7
iii) –10 and –13
iv) –14 and 11
v) -3 and -8

2. Draw a figure to illustrate and explain a 16-bit Carry-Look Ahead Adder using 4-bit adder blocks.
   Show that the carry and sum are generated in 5 and 8 gate delay respectively.

3. Using a block diagram which shows the register configuration, perform sequential circuit binary multiplication of Multiplicand=1010 and Multiplier = 1101.

4. Explain Booth's algorithm. Multiply 01110 (+14) and 11011 (-5) using Booth's multiplication, and explain.

5. Explain the design of 4-bit Ripple Carry Adder with a neat diagram.

6. Explain the design of a 4-bit Carry-Look Ahead Adder , with a neat diagram.

7. Show the multiplication of (+13) and (-6) using bit-pair recoding technique.

8. Explain Binary addition-subtraction logic network to perform the subtraction operation X-Y on 2's complement numbers X and Y.

9. Differentiate between restoring and non-restoring division.

10. Illustrate the steps for non restoring division algorithm on the following data:
    Dividend=1011, divisor=0101.

11. Explain Two methods of Fast Multiplication.

12. Explain Carry-Save Addition of Summands.

13. Write IEEE standard floating–point formats for 32-bit representation and explain

14. Illustrate with an example the algorithm for Restoring binary division with block diagram.

15. Illustrate with an example the algorithm for Non-Restoring binary division with block diagram.

16. Explain 4 bit carry–look ahead adder and use it to build a 12-bit carry–look ahead adder.

17. Perform 56-78 using 1's complement and 2's complement methods.

18. Using Booth algorithm multiply (-13) and (+ 107).

19. Draw a circuit diagram for binary division and explain its operation.

20. Explain how Booths algorithm is suitable for signed numbered multiplication in comparison of conventional shift and add method.
21. Write a short note on look ahead carry generator.

22. How do you design FAST ADDERS? Explain a 4-bit carry look ahead adder?

23. Explain the sequential binary multiplier with the use of a block diagram.

24. Explain the computational details of multiplying two 4 bit numbers 1 0 1 1 and 0 1 0 1 using Booths algorithm. Verify the result obtained.

25. With a neat diagram explain the floating-point addition-subtraction unit.

26. Multiply 10011 and 01001 using Booth's algorithm.

27. Let Multiplicand A = 110101 and Multiplier B= 011011. Multiply the given signed 2's complement numbers using Booth's algorithm. Verify the result using bit-pairing of multiplier.

28. Draw a neat sketch of single bus organization of the data path inside a processor , explain the three steps to be performed by the processor to execute an instruction.

29. Write and explain the control sequences for execution of following instruction with respect to single bus organization: Add R2, (R4) .

30. Write and explain the control sequences for execution of following instruction with respect to single bus organization: Add (R3), R1 .

31. List the actions needed to execute the instruction Add R1, (R3). Write the sequence of control steps to perform the actions for a single bus structure. Explain steps.

32. With a neat Block diagram , explain three bus organization and write control sequence for the instruction: Add R1, R2, R3 .

33. Compare hardwired control unit with micro-programmed control unit.

34. Write the control sequences for the instruction Move (R1), R2.

35. With a neat block diagram , explain hardwired control unit ,which shows separation of the decoding and encoding function.

36. Explain with a neat block diagram , the basic organization of a micro-programmed control unit.

37. Write and explain the control sequences for execution of an unconditional branch instruction.

38. Write and explain the control sequences for the execution of following instruction: Add (R3), R1

39. What are the modifications required in the basic organization of a micro programmed control unit to support conditional branching in the micro program.

40. Explain micro instruction sequencing with next address field.

41. Give the Micro instruction for Branch < 0.

42. Draw a block diagram of a complete processor and identify the units