UNIT – I

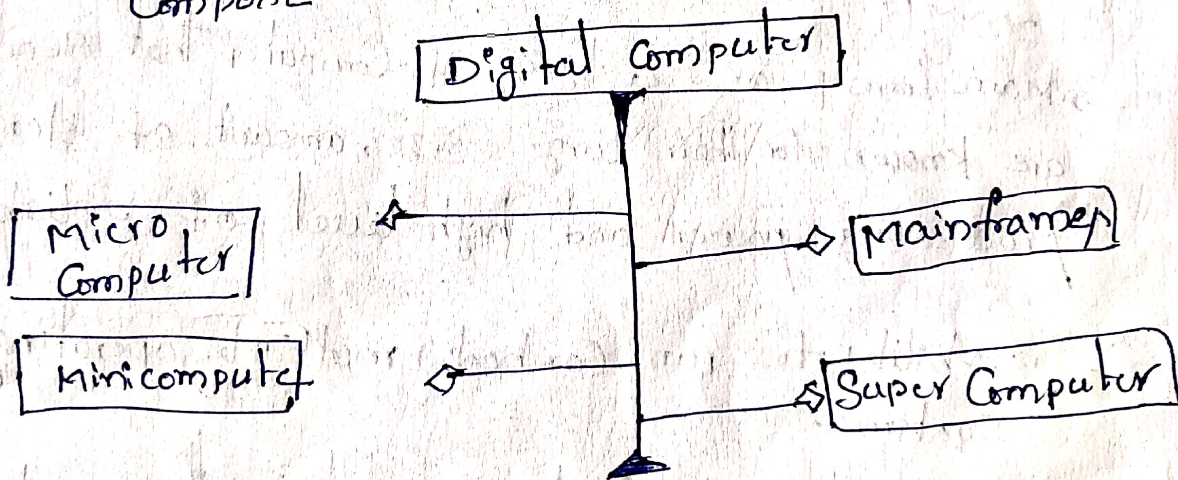
Digital Computers

→ Introduction

→ It is the most Commonly used type of Computer and is used to process information with quantities using digits, usually using the binary number system.

* Digital Computers are programmable machines that use electronic technology to generate, store and process Data

* Two terms, positive "1" and nonpositive "0", Compose the data into a string.



Micro-Computers

→ It is a small, relatively inexpensive computer with a Micro processor as its CPU. It includes a Microprocessor, memory, and I/O devices

→ Also known as "personal Computer"

→ Includes workstations, desktops, servers, laptop and notebook.

Minicomputer :-

→ MiniComputers emerged in the Mid-1960s and were first developed by IBM Corporation

→ This May also be called a mid-range Computer

→ Mini Computers may Contain one or more processors, support multiprocessing and tasking

Mainframes Computer :-

→ Main frames are a type of Computer that generally are known for Their Large Size, amount of storage, processing power and high-level of reliability.

→ Ability to run (or host) multiple operating Systems.

→ Mainframes first appeared in the early 1940's

Super Computer :-

-> Supercomputers Consists of tens of thousands of processors that are able to perform billions and trillions of calculations or computations per second.

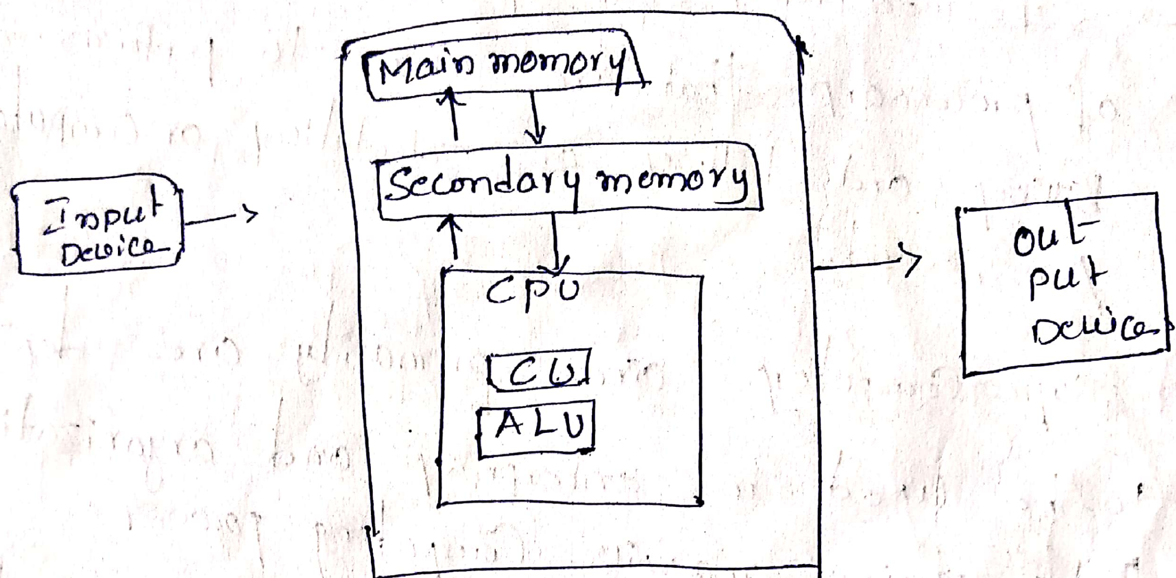
-> Super Computers are primarily are designed to be used in enterprises and organizations that require massive Computing power

-> It has more Than 98,000 processors That allow it to process at a speed of 16,000 trillion Caculations per second.

-> A large and very powerful mainframe Computer is Called a SuperComputer.

-> SuperComperrs are applied to the solution of very Complex and sophisticated scientific problem and for national security purposes of some advance nations

Block diagram of Digital Computer



-> A Computer Can process data, pictures, sound and graphics. they Can solve highly Complicated problems quickly and accurately.

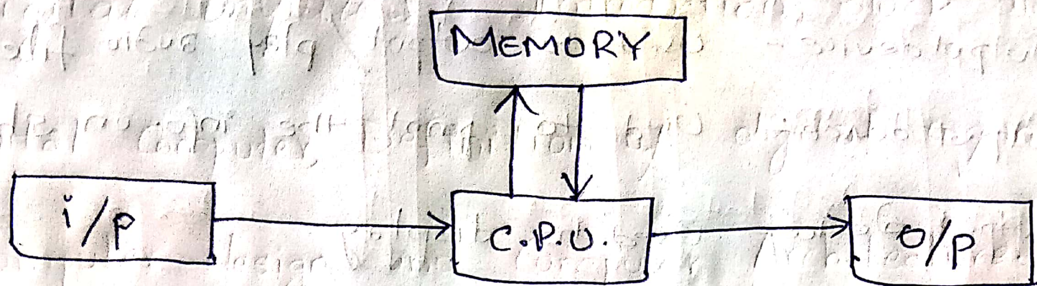
Input Device : This is the process of entering data and programs in to the Computer System

-> Computer need to receive data and instruction in order to solve any problem. therefore we need to input the data and instructions into the Computers

-> the input unit Consists of one or more input devices

-> input devices are
* Keyboard
* Mouse
* Floppy drive
* magnetic tape

Block Diagram of Computers:



Input devices: The devices which are used by the computer to take the input from a user. Ex: Mouse, Keyboard, Scanner etc.

Output devices: The devices which are used by the computer to give output to the user.
Ex: Printer, moniter, speaker etc.

C.P.U.: Central Processing Unit consists of ALU & CU.

Arithematic & Logical Unit is responsible for performing arithematic & logical operations.

Control Unit Controls the flow of data inside the computer.

Memory Unit: Stores the received information and retreives when needed. Divided into Primary & Secondary. Ex: Primary → RAM, ROM, Cache etc.
Secondary → USB, CD, floppy etc.

Mouse: Input device - used to point/move the cursor on the screen.

Keyboard: Input device - used to enter keys/characters using keys

Microphone: Input device - used to record/receive audio input.

Speaker: Output device - Used to output/play audio files.

Printer: Output device - Used to print the info. on sheets.

Monitor: Output device - Used to display

Primary Memory is volatile i.e. temporary.

Secondary Memory is permanent.

Primary Memory directly interacts with the processing unit.

Secondary Memory must be loaded on to primary memory to be processed.

COMPUTER DESIGN & ARCHITECTURE:

→ Designing a computer is a challenging task. It involves software & hardware. functional Organization, logic design and implementation.

→ Following are the parameters : * Cost
* Performance
* Usage

Micro computers
Mini computers
Mainframe
Super Computer

Architecture:

It is a graphical representation.

→ Computer architecture is concerned with the way in which hardware components are connected together to form a computer system.

→ Computer Architecture acts as the interface between hardware & software.

→ It helps to understand the functionalities of a system.

→ Architecture is a set of rules and methods that describe the functionality, organization and implementation of Computer systems.

→ An instruction set Archi (ISA) is the interface between the Computer's s/w and H/w and also can be viewed as the programmer's view of the Machine

→ Computer do not understand high-level programming languages such as Java, C++ or most programming languages used.

→ A processor only understands instructions Encoded in some numerical fashion, usually as binary numbers

⇒ Computer organization helps plan the selection of a processor for a particular project.

Implementation

→ once an instruction set and Micro Archi- are designed, a practical Machine must be developed. This design process is called implementation

1

=> Computer Architecture Vs Computer Organization

-> Computer Archi is Concerned with the way H/w Components are Connected together to form a Computer system.

-> It acts as the interface between H/w and S/w

-> It helps us to undersand The functionalities of a system

-> A programmer Can view archi. in terms of instructions, Addressing Modes and registers.

-> It deals with high level Design issues

-> Architecture involves logic (sets, addressing Modes

-> It is Concerned with the structure and behaviour of a computer system.

-> It deals with the Components of a Connection in a system.

-> It tells us how exactly all the units in the system are arranged and interConnected

-> Organization Expresses The realisation of architecture.

-> This deals with low level

-> It involves physical Components

23/01/14 UNIT

(6) Register Transfer Language AND Microoperations

⇒ Register Transfer Language :

→ To solve any problem with the computer we need a procedure That Can be coded in any high level language for our understanding and it should be translated into Machine language if we want to Execute the procedure or program using Computer.

→ the high level languages have its own Syntax and data types.

→ Similarly if we want to learn about the data transfer between two registers, it is defecult for us to read the instruction codes Each time and analyze the operation.

→ the language or the symbolic notation which describe the data transfer operation between the registers is called "the register transfer language"

→ this language also specifies the arithmetic, logic and shift operations performed on the data stored in the register. these operations are Called "Microprogram."

2/7/16

Memory Transfers

Q

-> there are various Types of memory used in Computer.

A memory is a collection of storage cells. Each Cell store 1-bit information.

-> the memory stores binary information in groups of bits called words.

-> the Transfer of information from a memory word to the outside Environment is called a read operation

-> the Transfer of new information to be stored into the memory is called a write operation.

-> the number of words in the memory decides the size of the memory and the number of address bits

-> For Example :-, 8 bit address Can access upto $2^8 = 256$ different words.

-> The number of information bits Can be read or written at a time is decided by the word length (numbers of bits in one word) of the memory.

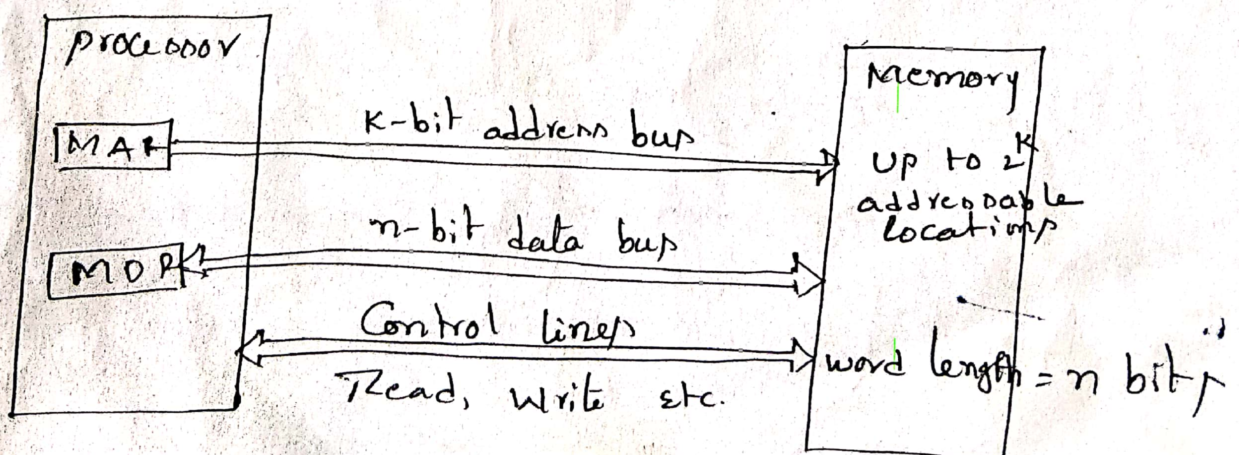
-: As shown in fig. the Control lines from the processor decides the memory ope

. In case of read operation Read signal is activated. It is used to enable the active low o/p Enable signal of the memory.

-> In case of write operation write signal is activated to indicate the write operation.

-> The data transfer between the memory and processor takes place through the use of two processor registers, usually called MAR or MAR and MDR.

-> If MAR is k-bit long and MDR is n bit long, it is possible to access upto 2^k memory locations, and during one memory cycle it is possible to Transfer n-bit data.



In read operation, the address of the memory word is specified by address register, AR, and the data word read from the memory is loaded into the data register, DR. This read operation can be represented as

Read : DR ← M [AR]

→ As shown above letter M represents memory word whose address is specified by address Register; AR.

→ The Control signal Read indicates that the operation is performed only when Read signal is active. once the data is available in the DR register it can be Transferred to any other register within the CPU. For Example, it can be Transferred to register, R_2 by following operation.

R_2 ← DR

→ It is important to note that for above operation the activation of Read signal is not necessary; it is necessary only when data is read from Memory unit.

→ the Write operation Transfers the Content of a data register to a memory word M selected by the address in the address register, AR.

→ the write operation Can be represented as

Write : $M[AR] \leftarrow DR$

→ the Control signal <u>write</u> indicates That the operation is performed only when Write signal is active when it is necessary to transfer data from another CPU register to the memory we have to transfer the data from That register to the data register, DR before write operation.

→ for Example, if we want to transfer data from register R_1 to the memory whose address is in AR then This operation Can be represented as

$DR \leftarrow R_1$

Write: $M[AR] \leftarrow DR$

→ BUS Transfer

→ A digital Computer has many registers holding different types of information

→ In performing various logical, arithmetic, data transfer and other operations, data has to be transferred from one register to another register.

→ In this approach, where interconnections are made between every pair of registers, the number of Interconnections becomes too large.

→ An alternate approach is to create a bus structure which is more Convenient and efficient for the purpose of data transfer.

→ In the approach to bus structure, the Information from registers is transferd using multiplexers.

→ the information from registers is Connected to Multiplexers and the selection input of the Multiplexers is chosen to select the i/p of the Multiplexer from the appropriate i/p register.

→ The o/p of the multiplexer is one of the i/p's of the Multiplexers, which in turn is the o/p of one of the registers, is selected by the Control lines of multiplexers and is treated as the o/p.

→ A four line bus structure, which selects one of eight registers is shown in fig.

In this diagram, the o/p's of the four bit register are called A0, A1, A2, A3

→ The four o/p's of register B are called B0, B1, B2 and B3. and similarly all the four bit O/p's of Each of the registers are called by the four suffixes 0, 1, 2 and 3 with the register names C, D, E, F, G, and H.

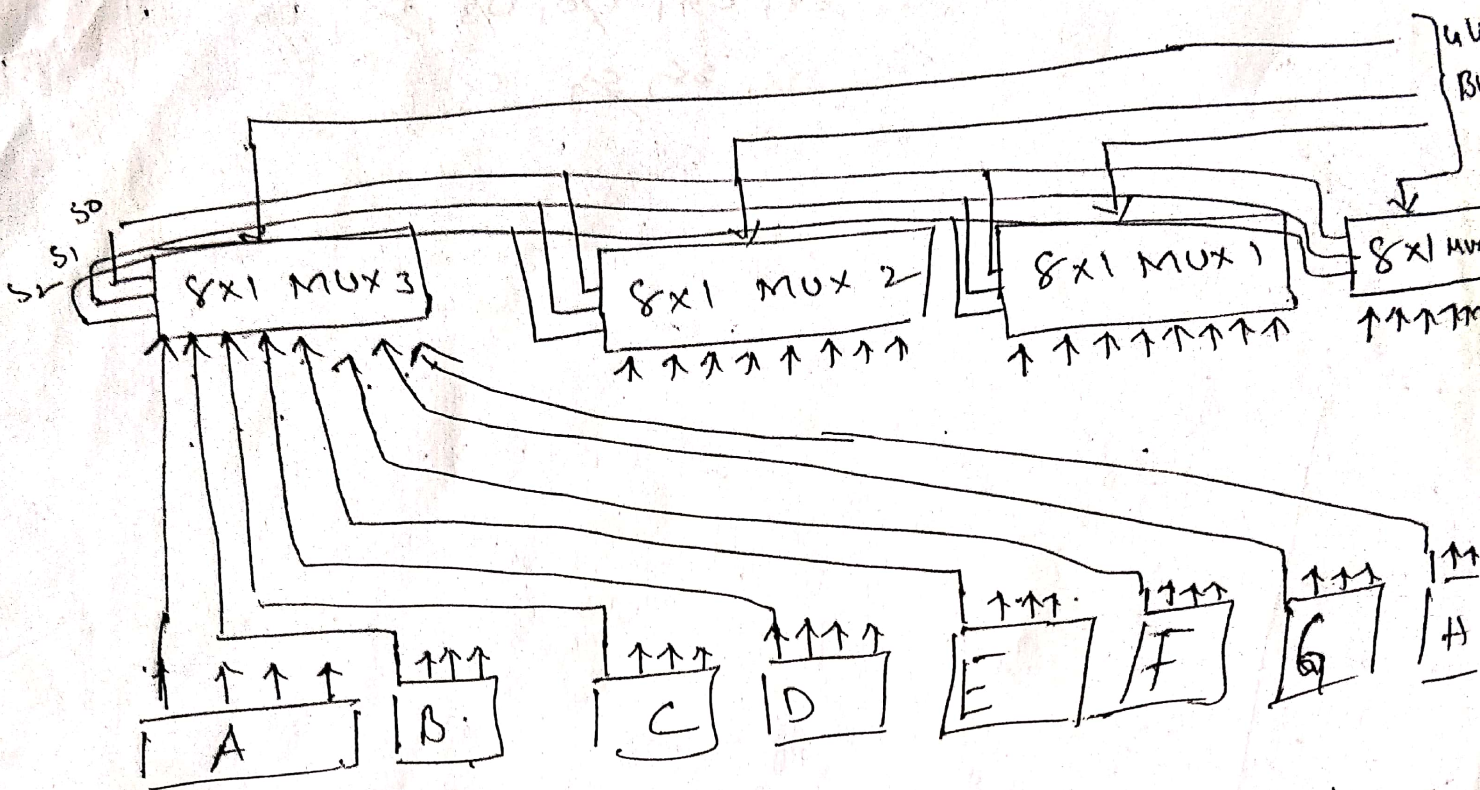
→ The Eight i/p one output multiplexer is connected such that the 0th o/p of each register is connected to multiplexer 0, 1 o/p of each register is connected multiplexer

→ the Multiplexer has three selection Lines S0, S1, S2 and depending on the values of selection i/p's.

→ one of the Eight registers is selected on the bus as per table

		S_2	S_1	S_0	Bus o/p
0	→	0	0	0	Register A
2	→	0	0	1	" B
3	→	0	1	1	" C
					" D
4	→	1	0	0	" E
5	→	1	0	1	" F
6	→	1	1	0	" G
7	→	1	1	1	" H

2



→ In general, a bus structure will multiplex N registers each with M bits to produce a M line common bus.

→ The number of multiplexers needed to construct the bus structure is M, the number of bits in each register.

BUS ← R

T ← BUS

61,

8

=> arithmetic micro operations

-> there are number of micro operations performed on the data stored in registers of digital computers.

-> A micro operation is a basic operation performed on the data stored in registers of digital computers

-> these micro operations are broadly classified into four types.

1. Register transfer operations which perform the data transfer from one Register to another registers.

2. arithmetic Micro operations which perform different arithmetic operation on the data stored in the computer registers.

3. Logical Micro operations perform logical operations on the data stored in the registers.

4. shift micro operations shift the data stored in computer registers

S.no	Arithmetic operation	Description	Register transfer Language
1	Addition ->	Add the contents of two registers and transfer the results to a third register	$D \leftarrow A + B$
2	subtraction ->	subtract the contents of two registers and store the result in third register	$D \leftarrow A - B$
3.	Two's Complement ->	Two's Complement the contents of the register	$R_1 \rightarrow \bar{R}_1 + 1$
4.	one's Complement ->	Complement the contents of a register	$A \leftarrow \bar{A}$
5.	Increment ->	Increment the contents of a register	$A \leftarrow$ +
	Decrement ->		

Addition:

→ To implement an add micro operation, with logic circuits, we require registers to hold the two operands and the logic circuitry that performs addi[tion]

fig: Bit Binary adder

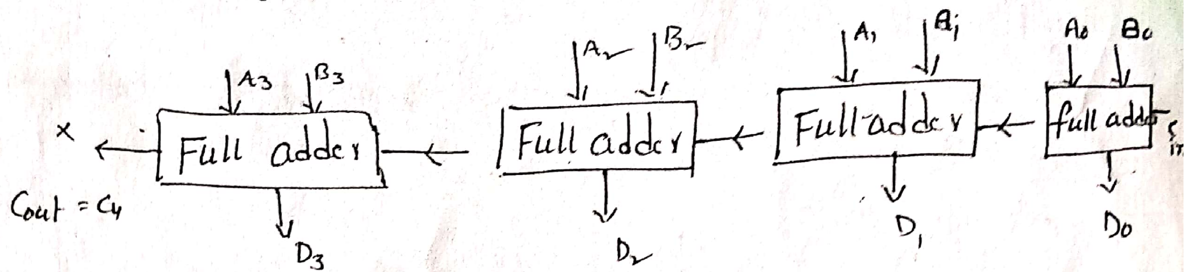


Table : Truth table of full Adder and Boolean Expressi[on]

	A	B	Cin	output D	output Cout
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	0
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	1	1

count → 1, 1 + 1 = 0 result

count → 1, 1 + 1 + 1 = 1 result

Subtraction:

→ the subtraction of two numbers is equivalent to adding the complement of the subtract and performing addition.

→ for Example (A−B) is equivalent to addi[ng] the two's complement of the contents of regist[er] B to the contents of register A and then adding 1 to the least significant bi[t]

⟹ logic Micro operations

28/01/14 ①

→ logic micro operations specify binary operations on the contents of two registers and the output of the logic operation is stored either in a third register or in one of the two registers.

→ logic micro operations consider every bit of the register separately and hence every bit is treated as a separate binary variable.

→ An AND operation with two contents of two registers A and B can be expressed as a register transfer language statement.

This statement has a meaning that the contents of registers A and B are ANDed together and the result o/p is stored in register D,

$C : D \leftarrow A \wedge B$

provided the variable $C = 1$.

→ A four bit numerical example is shown

1011	Contents of register A
1100	Contents of " B
1000	Contents of register D if C

→ However, logic operations are rarely used in scientific calculations but they are useful for bit manipulation of binary data, for making logical decisions.

fig: Symbol of logic Micro operations.

Logic operation	Symbol
OR	$A \vee B$
AND	$A \wedge B$
ones complement	$\bar{A}$
Exclusive OR	$A \oplus B$

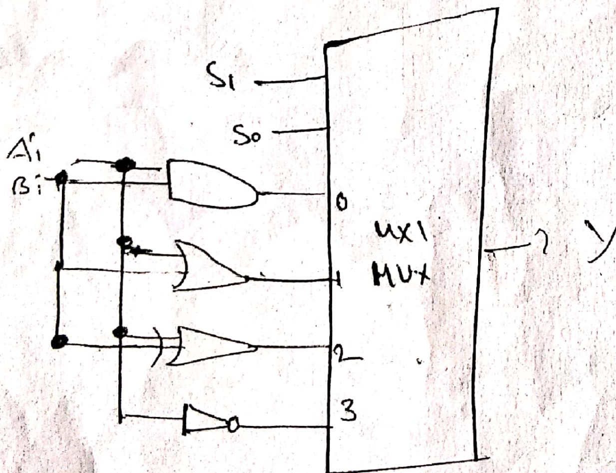
$\oplus$ { 0 0 → A, 1 0 → B; Ex: 1 1 / 0 0

→ same binary no → 0, different → 1

→ the "+" symbol is used for addition as well as logical OR operation.

→ When the symbol "+" occurs in a Micro operations, it represents an arithmetic addition when it occurs in a Control function, it denotes an OR operation.

→ fig: one stage of logic circuit

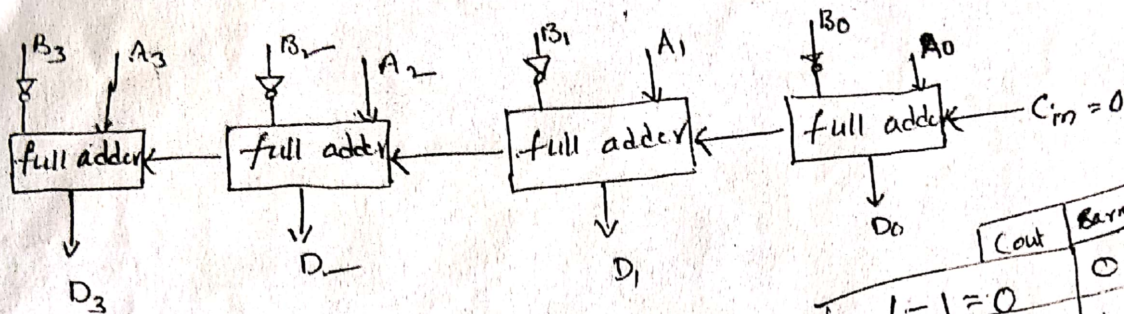


S_1	S_0	o/p	operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \bar{A}$	Complement

functional table.

27/01/14 ②

fig: Block Diagram of Binary subtractor



Cout	Borrow
	0
1 - 1 = 0	1
0 - 1 = 1	1
1 - 0 = 1	

(-)

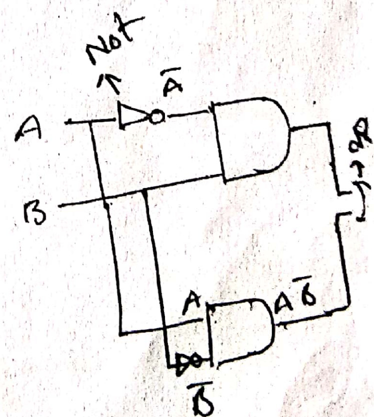
x	y	x+y	x-y
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

(+)

→ A digital circuit that performs the addition of two binary bits is called a half adder whereas a full adder performs the arithmetic sum of two bits and a previous carry to generate an output sum and output carry.

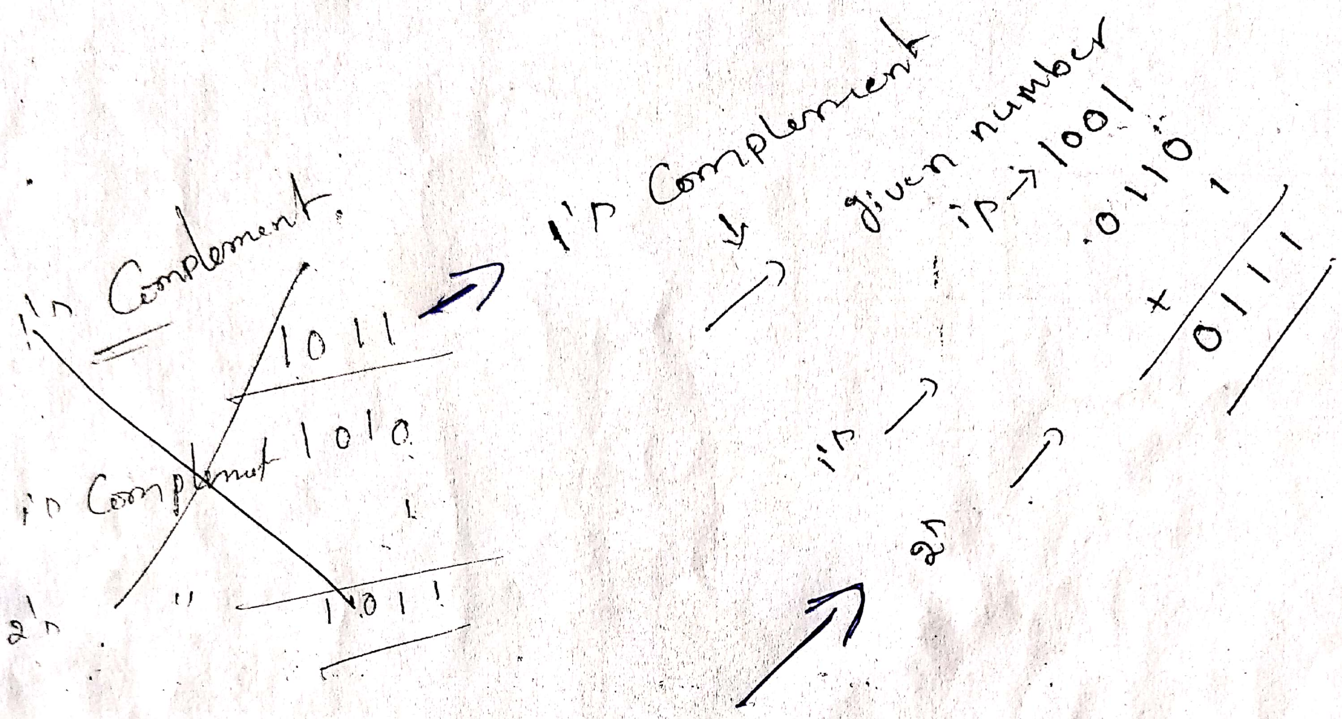
H·A Add

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Sum = $\bar{A}B + A\bar{B}$

$\bar{A}B + A\bar{B}$



Incrementer :- the increment micro operation adds one to the Contents of the binary number in the register.

→ If the Contents of a register are 1110 incrementation will make the Contents of register equal to 1111. If initial Contents of the register are 1111, the Circu[illegible] of the register after incrementation will be 0000,

Decrementer :- the decrement micro operation subtract one from the Contents of a given register.

Logic Micro operations

→ Logic Micro operations performs logic operations such as AND, OR, Complement and XOR on the string of bits stored in registers.

→ Special symbols are used for the Logic Microoperations OR, AND and Complement to distinguish them from the Corresponding symbols used to Express Boolean functions.

→ The list of Micro operations and their symbols

Logic Microoperation	Symbol
OR	$\vee$
AND	$\wedge$
Complement	Bar ($\bar{}$) — one's Complement
Ex-OR	$\oplus$

AND : the Micro operation logically ANDs the bits of one register with the bits of another register having same word length.

→ If the Whenever all the i/ps are high then the o/p is high (or) any one of the i/p is low o/p is low

Bn of Register	Bn of another Register	B[illegible]
0	0	0
0	1	0
1	0	0
1	1	1

fig: truth Table of AND micro operation

for example,

$$T: R_1 \leftarrow R_1 \wedge R_2$$

→ statement represents the AND micro operation with the Contents of two registers R_1 and R_2. the result of the micro operation is strored in the Register R_1. In the statement, T is a Control input and when it is logic 1 the Micro operation is performed.

OR micro operation:

→ This microoperation logically ORs the bits of one register with the bits of another register having same word length.

Bn of Register	Bn of another register	Bn of Result
0	0	0
0	1	1
1	0	1
1	1	1

Note: If any (one of the) i/p is high o/p is high (or) whenever all the i/p's are zero then the o/p is zero

XOR :- This Micro operation logically XOR, the bits of register with the bits of another register having same word length

Bn of register	Bn of another register	Bn of Result
0	0	0
0	1	1
1	0	1
1	1	0

Note :- Whenever all the i/p's are same o/p is zero (0) otherwise 1

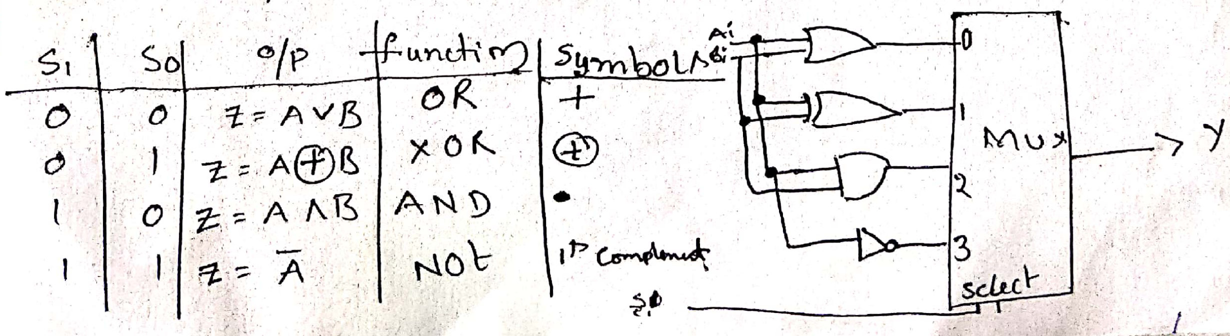
for Example :-

$$T: R_1 \leftarrow R_1 \oplus R_2$$

→ statement represents the XOR Micro operation with the Contents of two registers R_1 and R_2. The result of Micro operation is stored in the register R_1. in the statement T is a Control Input and when it is logic 1 the micro operation is performed

Hardware Implementation :- the design of logic Circuit is Comparatively simple than the design of arithmetic Circuit

S_1	S_0	o/p	function	Symbols
0	0	$Z = A \vee B$	OR	+
0	1	$Z = A \oplus B$	XOR	$\oplus$
1	0	$Z = A \wedge B$	AND	•
1	1	$Z = \bar{A}$	NOT	1's complement



for Example,

$T: R_1 \leftarrow R_1 \vee R_2$

→ statement represent the OR Micro operation w[ith] the Contents of two registers R_1 and R_2. the result of Micro operation is stored in the register R_1. In the statement T_1 is a Control input and when it is logic 1 the microoperation is performed.

Complement Microoperation :

→ this Microoperation Complement all bits in the register.

for Example:

$T_2: \overline{R_1}$

→ statement represents the Complement micro operation with the Contents of register R_1. In the statement T_2 is a Control input and when it is logic 1 the Microoperation is performed.

SHIFT Micro operations

9/07/16

→ shift micro operations are used for transfering data serially.

→ shift micro operations are used to shift the Contents of a register to the left or the right.

→ the Contents of a register Can be shifted to the right or left and the corresponding micro operations are called Left shift and right shift micro operations respectively.

→ As the bits are shifted left in a Left shift micro operation, serial input transfers a bit into the right most position and the left most bit is lost in the left shift operation.

→ In a right shift micro operation, a serial i/p is Entered into the left most position and right most bit is lost

→ there are three types of shift operations.

1. Logical shift
2. Arithmetic shift
3. Circular shift or rotate

Logic shift :

→ In logical shift operation the serial i/p is the vacant position created within the register due shift operation is filled with zero.

→ fig: there are two logical shift microoperations logical shift left (LshL) and logical shift right (LshR). For Example.

$R_1 \leftarrow LshL\ R_1$

$R_2 \leftarrow LshR\ R_2$

are two micro operations that specify a 1-bit shift to the left of the content of register R_1 and a 1-bit shift to the right of the contents of Register R_2.

→ Source and destination registers are same,

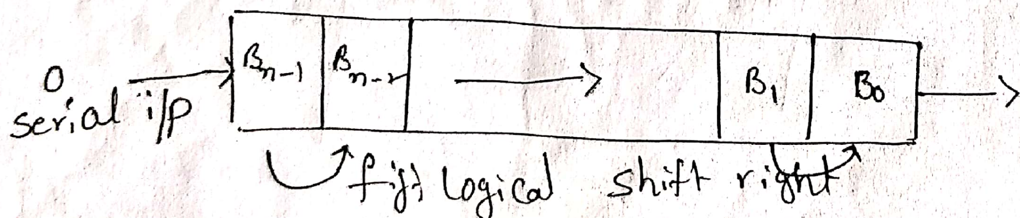


fig: Logical shift right

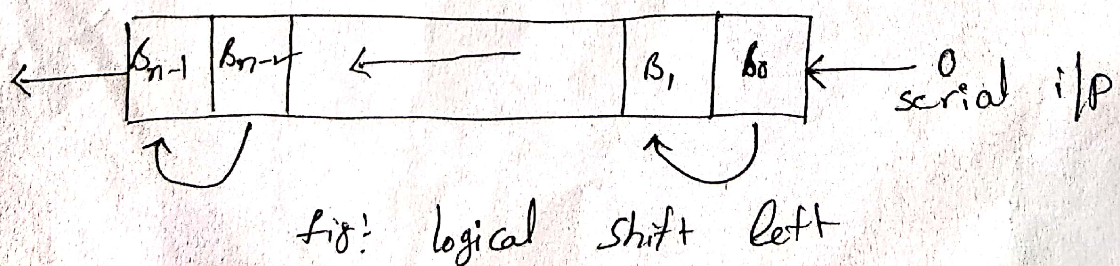


fig: logical shift left

Rotate or Circular shift Micro operations

→ Circular shift circulates the bits of the register around the ends without loss of any information. In the case of logical shift, one of the end bits is lost.

→ Circular shift or rotate operation is performed by connecting the least significant bit to the most significant bit position. The symbolic representation is shown in fig:

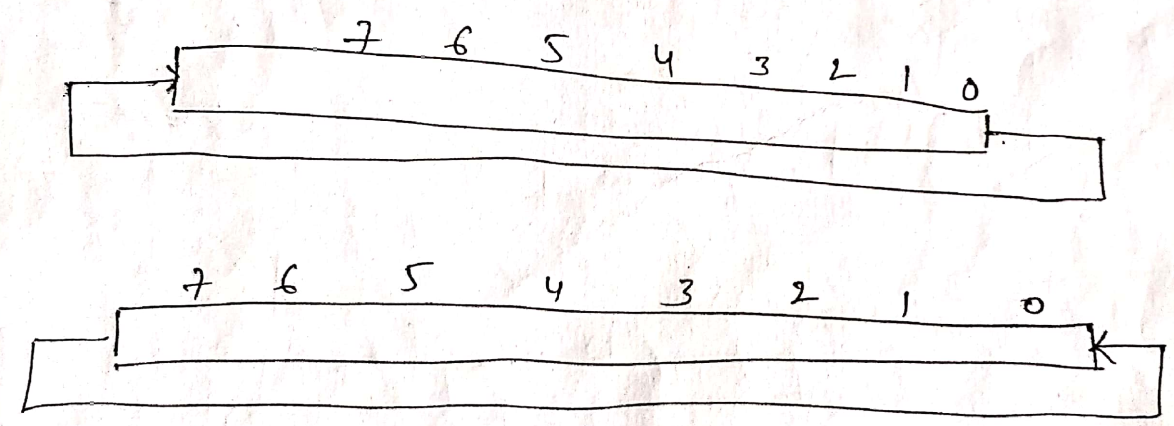


fig: Rotate right and rotate left operations.

fig: Shift Type micro operations

Type of shift	Description
R ← shl R	shift left the contents of register R
R ← " r R	shift right the contents of register R
R ← cil R	circular shift left of the contents of register R
R ← cir R	" right "
R ← ashl R	arithmetic shift left of the contents of register R
R ← ashr R	" right " "

Arithmetic shift :

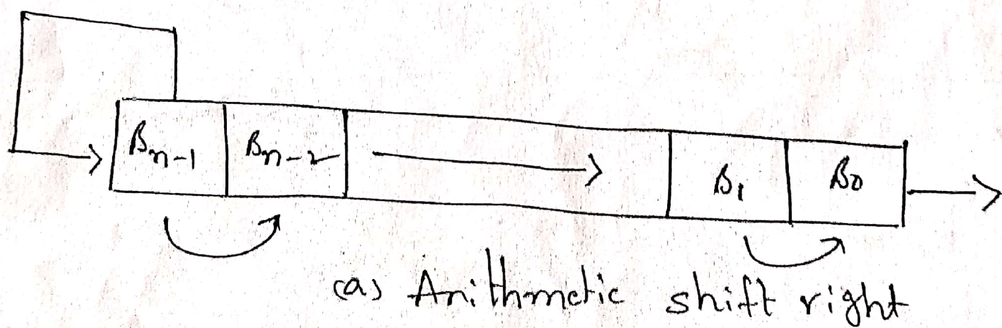
→ In logical shift operation we have seen that the vacant positions created within the register due to shift operation are filled with zeroes.

→ In arithmetic right shift it is necessary to repeat the sign bit as the fill-in bit for the vacated position.

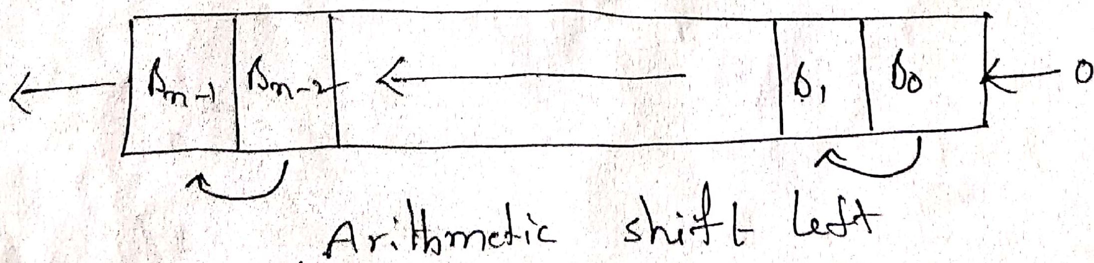
→ This requirement on right shifting distinguishes arithmetic shifts from logical shifts. Otherwise two shift operations are very similar.

→ fig: shows the arithmetic right shift (AShR) and the arithmetic left shift (AShL).

→ The arithmetic shift left is same as logic shift left.



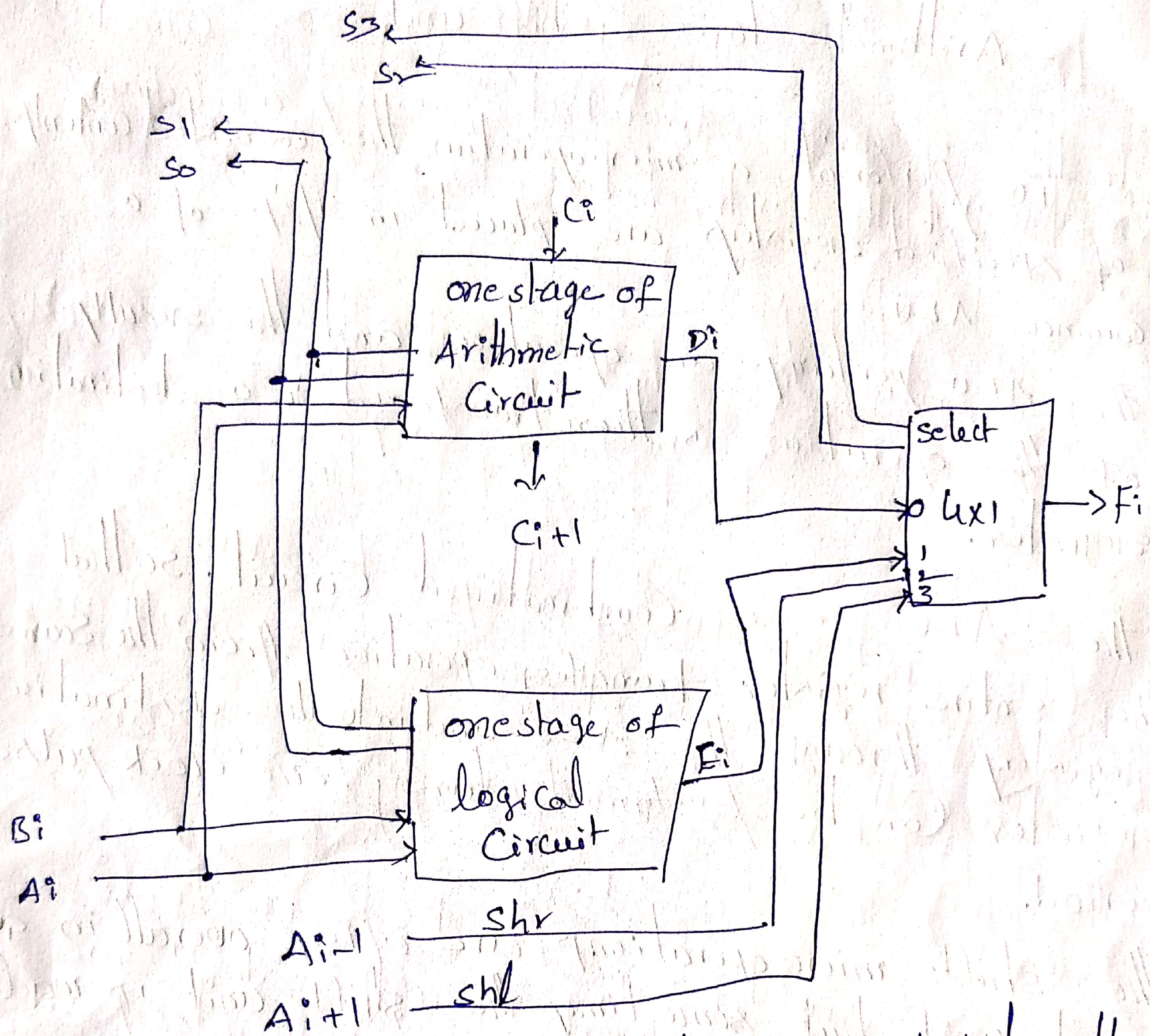
(a) Arithmetic shift right



Arithmetic shift left

Arithmetic logic shift unit

- To perform a microoperation, the connected contents of specified registers are placed in i/p's of a common ALU
- the ALU performs an operation and the results of the operation is then transferred to a destination register.
- the ALU is a combinational circuit. so that the entire register transfer operation from the source registers through the ALU and into the destination register can be performed during one clock pulse period.
- the shift microoperations are often overall in a separate unit, but some times the shift unit is made part of the overall ALU.
- the arithmetic, logic, and shift circuits can be combined into one ALU with common selection variables.
- one stage of an arithmetic logic shift unit is shown in fig.
- inputs A_i and B_i are applied to both the arithmetic and logic units.



→ A particular Micro-operation is selected with i/p's S_1 and S_0. A 4x1 Multiplexer at the o/p chooses between an arithmetic o/p in E_i and logic o/p in H_i.

→ The data in the Multiplexer receive i/ps A_{i-1} for the shift right operation and A_{i+1} ~~for the~~ for the shift left operation.

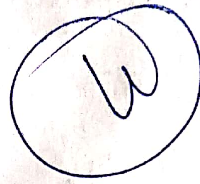
→ The o/p Carry C_{i+1} of a given arithmetic stage must be connected to the i/p Carry C_i of the next stage in sequence.

- the i/p Carry to the first stage must be connected to the i/p Carry Cin, which provides a selection variable for the arithmetic operations.
- Eight arithmetic operation, four logic operations and shift two operations.
- Each operation is selected with the five variables S_3, S_2, S_1, S_0 and Cin. the i/p Carry Cin is used for selecting an arithmetic operation only.

- the table in this 14 operations of the ALU.
- the first 8 are arithmetic operations and are selected with $S_3 S_2 = 00$.
- the next four are logic operations and are selected with $S_3 S_2 = 01$.
- the i/p Carry has no effect during the logic operations and is marked with don't-care x's.
- the last two operations are shift operations and are selected with $S_3 S_2 = 10$ and 11.
- the other three selection i/ps have no effect on the shift

operation select					operation	Function
S_3	S_2	S_1	S_0	C_{in}		
0	0	0	0	0	$F = A$	Transfer A
0	0	0	0	1	$F = A + 1$	Increment A
0	0	0	1	0	$F = A + B$	Add
0	0	0	1	1	$F = A + B + 1$	Add with Carry
0	0	1	0	0	$F = A + \overline{B}$	subtract with borrow
0	0	1	0	1	$F = A + \overline{B} + 1$	subtraction
0	0	1	1	0	$F = A - 1$	Decrement A
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	X	$F = A \wedge B$	AND
0	1	0	1	X	$F = A \vee B$	OR
0	1	1	0	X	$F = A \oplus B$	XOR
0	1	1	1	X	$F = \overline{A}$	Complement A
1	0	X	X	X	$F = shr\ A$	shift right A into F
1	1	X	X	X	$F = shl\ A$	shift left A into F

1. Instruction codes.



①

→ Every Computer System has a Central processing unit the operation of the Central processing unit and the Computer System is determined by the instruction Executed by Central processing unit. These instructions are known as machine instructions or Computer instructions.

→ Machine instructions are in the form of binary Codes known as instruction codes.

→ Each instruction of the CPU Contain specific information fields, which are required to Execute it. These information fields of instructions are called Elements of instruction.

there are :

1. operation code :

→ The operation Code field in the instruction specifies the operation to be performed.

→ The operation is specified by binary code.

2. Source / Destination operand :

The
→ Source /destination operand field directly specifies The Source /destination operand for the instruction.

→ In 8085, the instruction MOV A, B has B register as a source operand and A Register as a destination operand, because this instruc-tion Copies the Contents of register B to register A.

3. Source operand Address :

→ We know that the operation specified by the instruction may require one or more operands

→ the Source operand may be in the CPU register or in the Memory.

→ Many times the instruction specifies the address of the source operand so that operand (s) Can be accessed and operated by the CPU according to the instruction.

4. Destination operand address:-

-> The operation executed by the CPU may produce result. Most of the times the result is stored in one of the operand. Such operand is known as destination operand.

-> The instruction which produce result specifies the destination operand address.

5. Next instruction address:-

-> The next instruction address tells the CPU from where to fetch the next instruction after completion of execution of current instruction.

-> For JUMP and BRANCH instructions that address of the next instruction is specified with in the instruction.

-> However, for other instructions, the next instruction to be fetched immediately follows the current instruction.

Types of operands :

→ We have seen that each instruction in a program specifies operation to be performed and data to be processed.

→ For this reason, an instruction is divided into two parts: its operation code (opcode) and its operands.

→ The operand is an another name for data. It may appear in different forms:

* Address : the addresses are in fact a form of data
* Numbers.
* characters.
* logical Data

→ In many situations, some calculation must be performed on the operand reference in an instruction to determine physical address

Numbers : All computer supports numeric data types.

→ The common numeric data types are

* integer or Fixed point
* Decimal

characters : For documentation a common form of data is text or character strings. Today, most of the computers use ASCII (American standard code for information Interchange) code for character represented by a unique 7-bit pattern.

logical Data : logical data is used to store an array of binary data items and with logical data we can manipulate the bits of data items

CO (1)

Instruction format

(5)

→ Each instruction is represented by a sequence of bits with in the Computer. the instruction is divided into group of bits Called field. the way instruction is Expressed is Known as instruction format.

→ ~~the way instruction is~~ It is usually represented in the form of rectangular box.

Ex: (A+B)

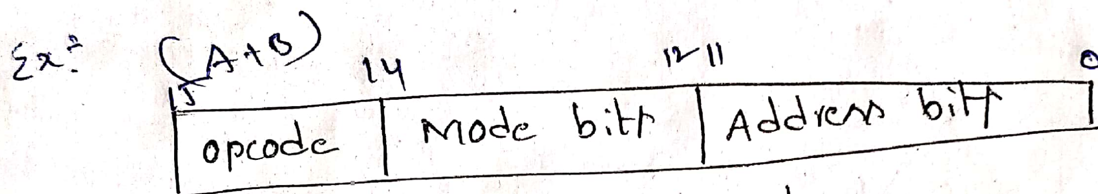


fig: Instruction Format

→ the bits of the instruction are divided into a number of fields. the number of fields are not same in all the instructions, However all the instructions will have an opcode field, an address field and a mode field.

→ Some instructions may have a register field and some instruct. have more than one, sometimes even zero, address field.

→ A shift-type instruction may have one extra field which Specifies the number of shift to be executed for the particular shift instruction.

* opcode field: this field specifies the operation to be performed.

→ It is a group of bits and the group of bits define number of process operations like add, subtract, Multiply, Complem. shift, more, etc

Mode field :-

→ The mode field specifies how the effective address of the operand is to be obtained.

→ The bits that are used in mode field specify various alternatives for choosing the operands from a given address.

Address field :-

→ The address field designates either a memory address (or) a processor register. If it specifies a processor register, the contents of the processor, interpreted with the Mode, decides the address of the operand.

→ If the address field specifies a memory address, it is treated as operand address i.e. the memory contents are calculated to find the operand address.

→ The number of address fields in a given instruction format depends on the internal organization of registers.

→ Most of the Computers fall into any one of three types of CPU organizations.

1. Single accumulator organization
2. general register organization
3. stack organization.

Single Accumulator Organization :

→ The instruction format has a provision to accommodate only one address field. Such organization may also use zero address instructions like increment, which implies the contents of accumulator to be incremented by one.

→ In the Single Accumulator organization type, instruction format:

ADD Y --- AC ← AC + M (DX)

→ Means add the contents of accumulator to the contents of memory at location Y and store the result in accumulator. Since the instruction contains at most one address field, the length of the instruction format is not too long.

* General Register organization :

→ The instruction format in this type of organisation requires three address fields.

→ It is possible out of the three address fields, all the addresses may be registers or one of the addresses may be memory location.

→ In such a Computer, an instruction for arithmetic operation of addition may be written as

ADD R_1, R_2, R_3

To specify the operation $R_1 \leftarrow R_2 + R_3$, the num[ber] of address fields can be reduced to two, if one of the source operand registers is treated as destination.

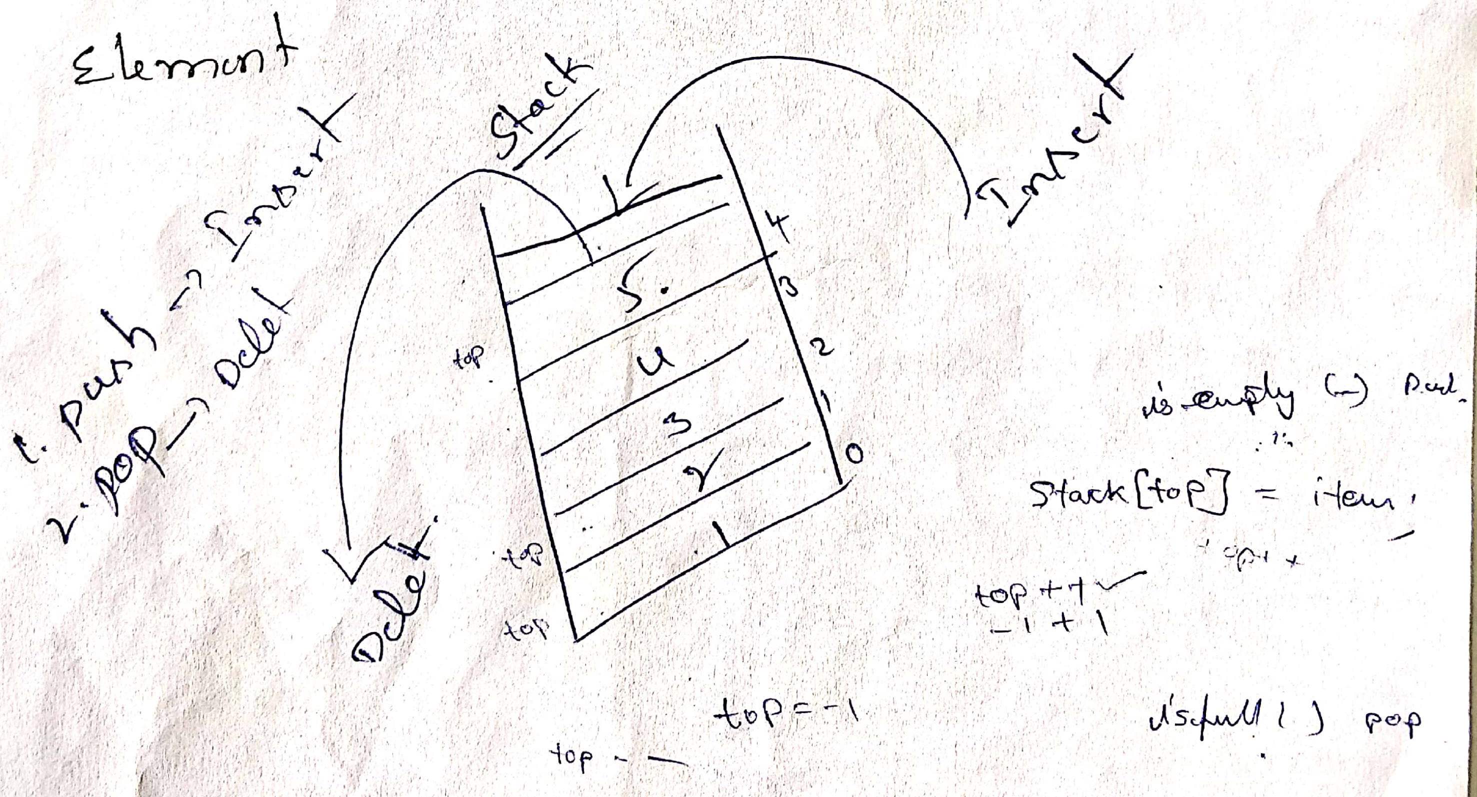
(8)

* stack organization :

-> Computer using stack organization use PUSH and pop instructions which require an address field.

-> PUSH P implies the contents of memory location P have to be transferred to the top of stack and the top of stack is automatically updated

-> pop instructions is used to Delete the Element



9

→ The number of addresses on computer program, we will evaluate the arithmetic statement.

$$X = (A+B) * (C+D)$$

→ using zero, one, two, or three address instructions.

→ we will use the symbols ADD, SUB, MUL and DIV for the four arithmetic operations

→ MOV for the Transfers type operation and LOAD and store for Transfers to and memory and AC register.

→ we will assume that the operands are in memory address A, B, C and D and the result must be stored in memory at address X.

* three Address Instruction:

ADD	R_1, A, B	$R_1 \leftarrow M[A] + M[B]$
ADD	R_2, C, D	$R_2 \leftarrow M[C] + M[D]$
MUL	X, R_1, R_2	$M[X] \leftarrow R_1 * R_2$

* Two Address Instructions: $X = (A+B)*(C+D)$

(10)

MOV	R_1, A	$R_1 \leftarrow M[A]$
ADD	R_1, B	$R_1 \leftarrow R_1 + M[B]$
MOV	R_2, C	$R_2 \leftarrow M[C]$
ADD	R_2, D	$R_2 \leftarrow R_2 + M[D]$
MUL	R_1, R_2	$R_1 \leftarrow R_1 \times R_2$
MOV	X, R_1	$M[X] \leftarrow R_1$

* One Address Instructions:

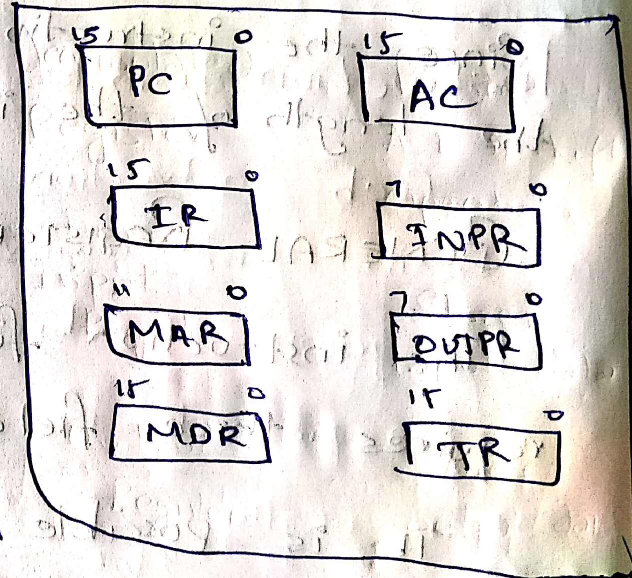
LOAD	A	$AC \leftarrow M[A]$
ADD	B	$AC \leftarrow AC + M(B)$
STORE	T	$M[T] \leftarrow AC$
LOAD	C	$AC \leftarrow M[C]$
ADD	D	$AC \leftarrow AC + M[D]$
MUL	T	$AC \leftarrow AC * M[T]$
STORE	X	$M[X] \leftarrow AC$

* Zero Address Instructions:

PUSH	A	$TOS \leftarrow A$
PUSH	B	$TOS \leftarrow B$
ADD		$TOS \leftarrow (A+B)$
PUSH	C	$TOS \leftarrow C$
PUSH	D	$TOS \leftarrow D$
ADD		$TOS \leftarrow (C+D)$
MUL		$TOS \leftarrow (C+D) * (A+B)$
POP	X	$M(X) \leftarrow TOS$

COMPUTER REGISTERS:

A computer requires no. of registers to hold data. Each register has a specified word length.



PC: Program Counter consists of 12 bits and length 0 to 11.

MAR: Memory address register. It stores (or) transfers the address word length is 0 to 11 and 12 bits.

MDR : Memory Data Register it stores the data, of register word length 0 to 15 and size is 16 bits.

IR : Instruction Register, it stores current instruction Word length is 0 to 15 size is 16 bits.

AC : Accumulated Register performs Arithematic, Logical and shift operations ; word length 0 to 15 bits. Size → 16 bits.

INPR : Input register stores input data word length 0 to 7 bits & size is 8 bits.

OUTR : Output register stores output data. Word length 0 to 7 & size is 8 bits.

TR : Temporary Register. stores temporary data word length 0 to 15. Size is 16 bits.

Register	Word length	Register Name	Function
PC	0-11	Program Counter	Stores Current informatio[n]
MAR	0-11	Memory Address Register	Stores Addresses
MDR	0-15	Memory data register	Stores data
IR	0-15	Instruction Register	Stores Instrution
AC	0-15	Accomulated Register	Performs A, L & Shift Operations
INPR	0-7	Input Register	Stores input data
OUTR	0-7	Output Register	Stores output data
TR	0-15	Temporary Register	Stores tempora[ry] data or address

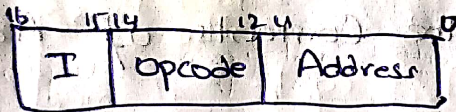
COMPUTER INSTRUCTIONS:

→ Instruction is a group of bits.

→ Computer instructions can be classified as memory reference instruction, register reference instruction and input and output instruction.

(i) Memory Reference Instruction:

A memory reference instruction reply? that are operand is required to be obtained from memory.



Ex:- **ADD**: Adds constants of memory to the constants of accomolate register.

AND: Perform and operation between contents of memory.

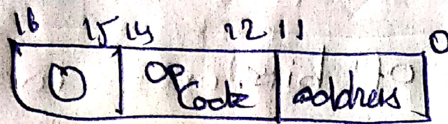
LOAD: Load the Contents of memory location in accomolator.

STORE: Stores the contents of accumolator in memory.

BRANCH: Branch unconditional to the address.

OR: Perform OR operation b/w the contents of memory location.

Register Reference Instruction:



It is basic instruction format, it consists of three fields.

i) Mode bit ii) Op Code iii) Address.

Mode bit is zero, It can transfer or hold direct information.

i) CLAC: Clear the Contents of Accomolator

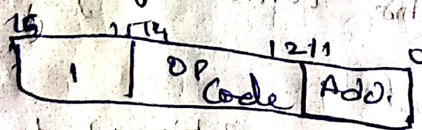
CMAC: Complement the Content of Accumolator.

INC: Increment the content of Accumolator.

DEC: Decrement the content of Accomolator.

SPAC: Skip one instruction contents of Accumolat

Input/Output Reference Instruction:



It is the basic instruction format, it consists of three fields.

i) MODE bit ii) OP Code iii) Address

→ Mode bit is 1. It can transfer or hold indirect information.

I/O instructions are: (i) Input (ii) Output - - -

Input : Input data to accumulator.

Output : Data (or) character in accumulator to be output.

INTON: Interrupt Enable

INTOFF: Interrupt disable

SKIP 1: Skip next instruction if input flag is 1.

SKIP 0: Skip next instruction if input flag is 0.

Timing and control

Clock pulses :-

The timing for all registers in the basic computer is controlled by a master clock generator. The clock pulses are applied to all flipflops and registers in the system, including the flipflops and registers in the control unit. The clock pulse do not change the state of a register unless the register is enable by a control signal. The control signals are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in processor registers and micro Operations for the accumulator

Hardwired control

There are two major types of control Organisation

i) Hardwired control
ii) Micro programmed control

i) Hardwired control

In the hardwired Organization, the control logic is implemented with gates, flipflops, decoders, and other digital circuits. It is the advantage that it can be optimized to Produce a fast mode of Operation.

ii) Micro programmed control

In micro programmed Organization, the control information is stored in a control memory. The control memory is programmed to initiate the required Sequence of Micro-Operations.

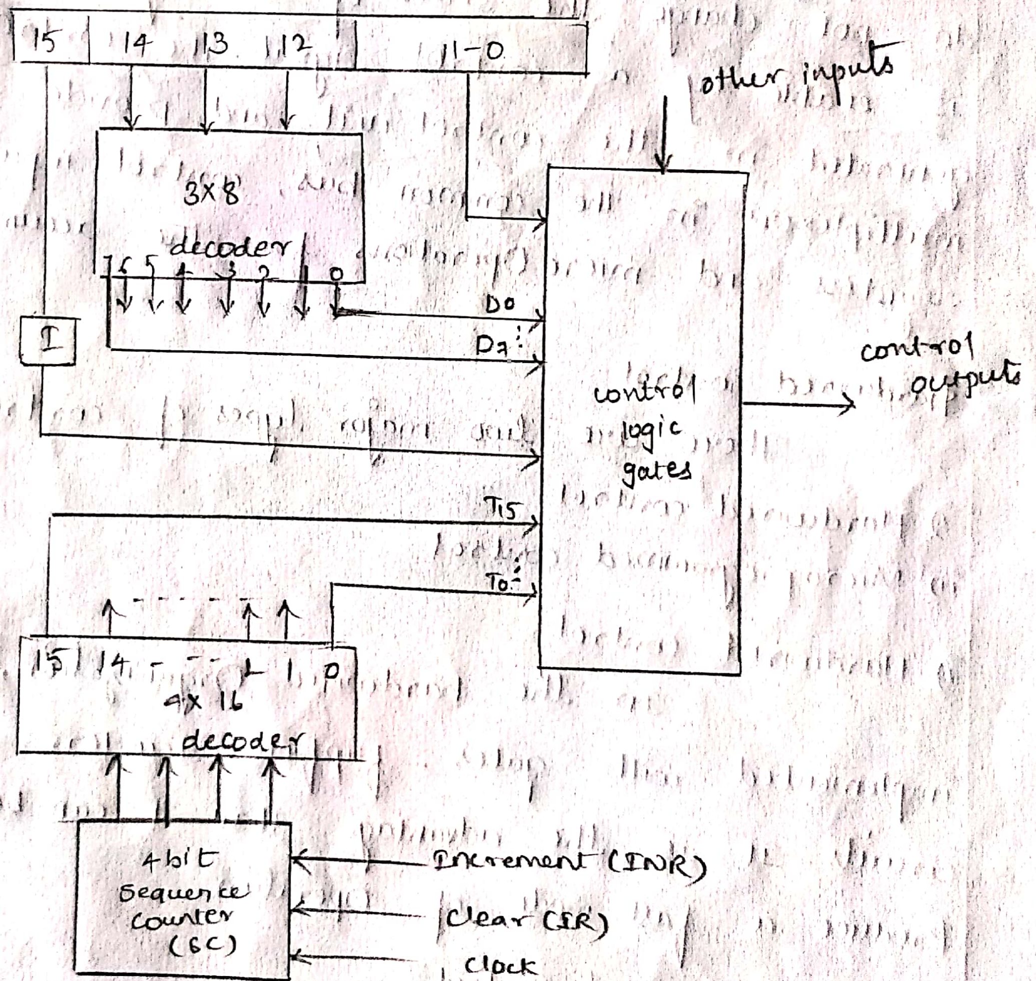
→ A hardwired control as the name implies, required changes in the wiring among the various components if the design has to be modified or changed.

→ In Microprogrammed control, any required changes or modifications can be done by updating the microprogram in control memory.

→ A hardwired control for the basic computer is presented in this section

Control Unit

The block diagram of the control unit.



control unit of Basic computer

It consists of two decoders, a sequence counter, and a number of control logic gates. An instruction read from memory is placed in the instruction register (IR). The position of this register in the common bus system is indicated.

The instruction register which is divided into three parts

i) The I bit
ii) the Operation code
iii) bits 0 through 11

The Operation code in 12 bits through 14 are decoded with a 3x8 decoder. The eight outputs of decoder are designed by the symbols D_0 through D_7. The Subcripted decimal number is Equivalent to the binary value of the corresponding Operation code. Bit 15 of the instruction is transferred to a flip flop designated by the symbol I. Bits 0 through 11 are applied to the control logic gates. The 4 bit Sequence counter can count in binary from 0 through 15. The Outputs of the counter are decoded into 16 timing signals T_0 through T_{15}. The internal logic of the control gates will be derived later when we consider the design of the computer in detail

The Sequence counter SC can be incremented or cleared Synchronously (Refer 4-bit synchronous binary counter). Most of the time, the counter is incremented to provide the Sequence of timing signals out of the 4x16 decoder. Once in a while, the counter is cleared to 0, causing the next active timing signals to be T_0. As an Example, consider the case where SC is incremented to provide timing signals T_0, T_1, T_2, T_3 and T_4 in Sequence. At time T_4, SC is cleared to 0 if decoder output D_3 is active. This is Expressed symbolically by the stmt

$$D_3T_4 : SC \leftarrow 0$$

The timing diagram of Fig

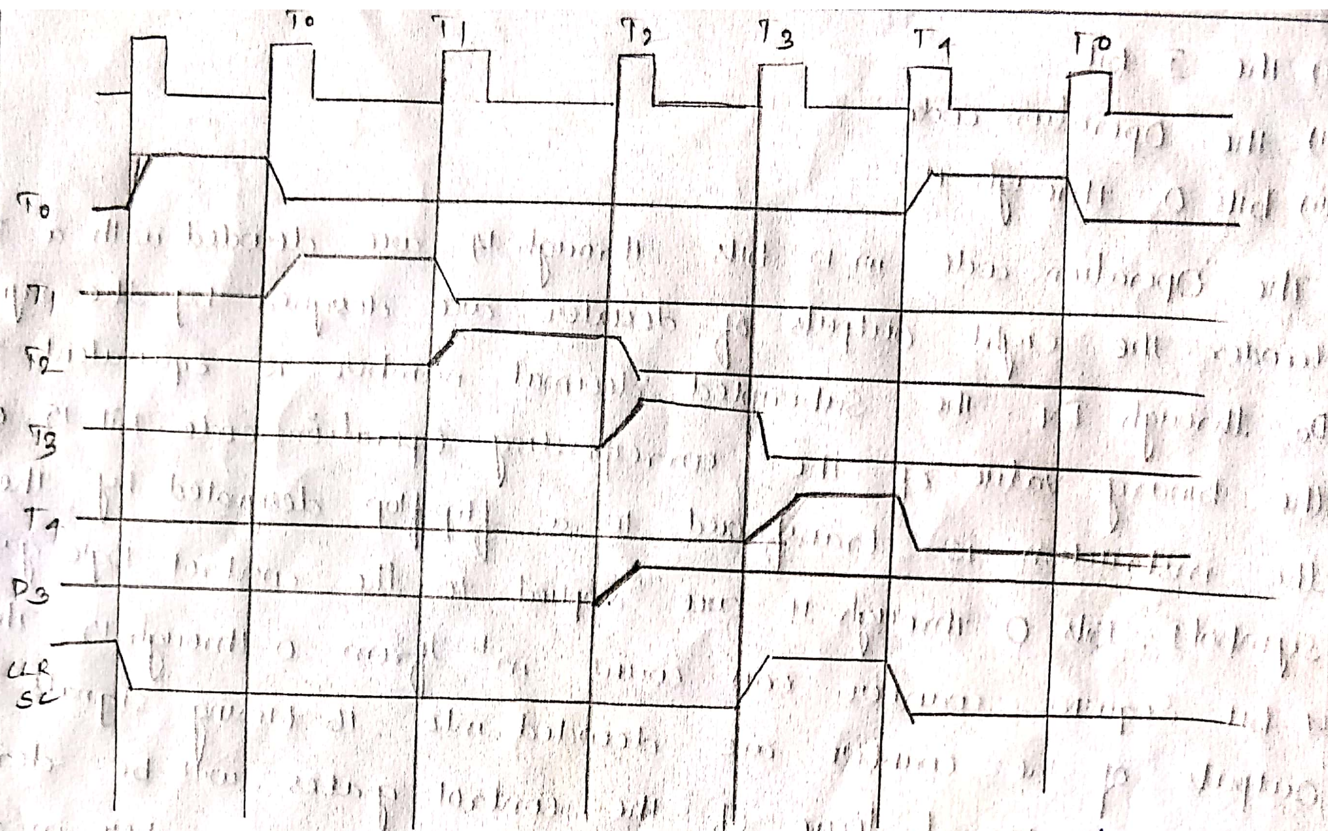


Figure: Example of control timing signals

it show the time relationship of the control signals. The sequence counter SC responds to +ve transistion of clock. Initially, the CLR inputs of SC is active. the first positive transistion of the clock clear SC to 0, which in turn activates the timing signal T0 out of decoder T0 is active during one clock cycle. The +ve clock transistion labeled T0 in diagram will trigger only those registers whose control inputs are connected to timing signals T0. SC is incremented with every +ve clock transistion, unless it CLR input is active. This produces the sequence of timing signals T0, T1, T2, T3 T4 and so on as shown in diag (Note the relationship b/w the Timing signals and its corresponding +ve clock transietion) If SC is not cleared, the Timing signals will continue with T5, T6 upto T15 & Back to T0

The last three waveforms show how SC is cleared when $D_3T_4 = 1$. Output D_3 from operation decoder becomes active at the end of timing signal T_3. When timing signal T_4 become active, the output of the AND gate that implements the control function D_3T_4 becomes active. This signal is applied to the CLR input of SC on the next +ve clock transistion (the one marked T_4 in diag), the counter is cleared to 0. This causes the timing signal T_0 become active instead of T_5 that would have been active if SC were incremented instead of cleared

A Memory read or write cycle will be initiated with the rising edge of a timing signal. It will be assumed that a memory cycle time is less than the clock cycle time. According to this assumption, the memory read or write cycle initiated by a timing signal will be completed by the time the next clock goes through its +ve transition. The clock transistion will then be used to load the memory word into a register. This timing relationship is not valid in many computers because the memory cycle time is usually longer than the processor clock pulse. In such a case it is necessary to provide wait cycles in the processor until the memory word is available. To facilitate the presentation, we will assume that a wait period is not necessary in this basic computer

TO fully comprehend the operation of the computer, it is crucial that one understands the timing relationship between clock transistion & the timing signals. For Ex the register transfer statement.

T_0: AR ← PC

Specifies a transfer of content of PC into AR if timing signal T_0 is active. T_0 is active during an entire clock cycle interval. During this time the content of PC is placed into the bus (with $S_2S_1S_0 = 010$) and the LD (load) input of AR is enabled. The actual transfer does not occur until the end of clock cycle when the clock goes through a +ve transistion. This same +ve clock transistion increments the sequence counter SC from 0000 to 0001. The next clock cycle has T_1 active & T_0 inactive.

→ Instruction cycle also known as fetch, decode and execute cycle ; is the basic operational process of a computer. This process is repeated continuously by C.P.U. from boot up to shut down of computer.

FETCH: Instructions are fetched from the Memory.

→ In this phase, the Sequence Counter is initialized to zero.

→ The address of first Instruction PC is loaded into Address Register during the first clock cycle.

DECODE:

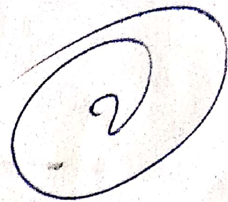
→ The instruction is decoded by the instruction decoder of a processor,

→ All the bits of the instruction under execution stored in IR are analyzed and decode in third clock cycle.

EXECUTE: In the last phase, the processor executes the instruction.

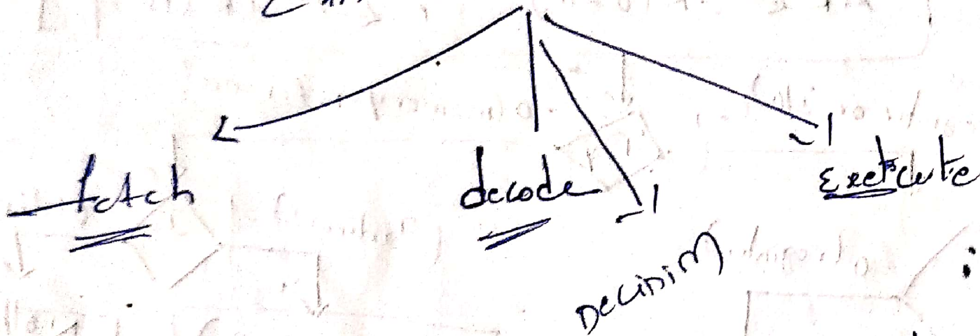
→ This involves setting the contents of the internal register AC to constant value zero.

Instruction Cycle



program :→ Memory unit

→ sequence of instruction
→ Each instruction will be Executed from Instruction cycle



→ step 1 : start SC ← 0 → Next Instruction executed that will be stored in PC

step 2 : T_0 : AR ← PC
step 3 : T_1 : IR ← M[AR] , PC ← PC+1 } fetch from memory

step 4 : T_2 : decode opcode IR (12-14)
AR ← IR (0-11), I ← IR(15) } Decode

step 5 : T_3 : decision { Memory reference Instruction, Register ", I/O " }

→ if it is Indirect Execution will be T_4
if it is derect execution will be T_3

step 6 : T_4

start SC ← 0 — T_0

AR ← PC — T_0

IR ← M(AR), PC ← PC+1 — T_1

(fetch)

Decode opcode in IR (12-14)
AR ← IR (0-11)   I ← IR (15) — T_2

(Decode)

D_7: (Register or I/O) = 1 ; = 0 (memory-reference)

I: (I/O) = 1 ; = 0 (register)

I: (Indirect) = 1 ; = 0 (direct)

(Decision)

T_3: AR ← M(AR)

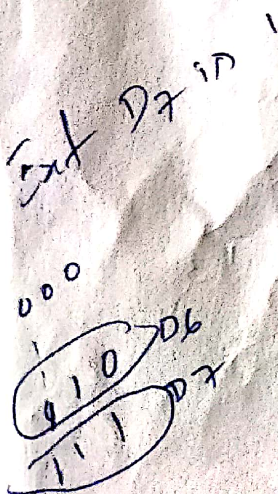
T_3: Nothing

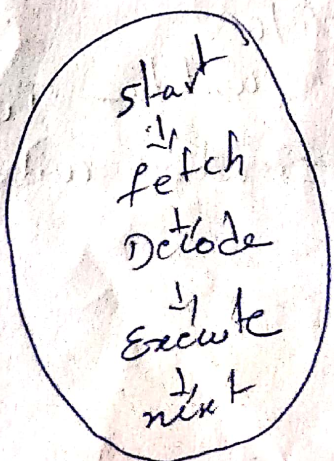
Execute input-output instruction SC ← 0

Execute register-reference instruction SC ← 0

Execute memory-reference instruction SC ← 0 — T_4

(Execute)





→ Encoding is the process of putting a ~~as~~ sequence of characters (letters, number) in to a specialised format for ~~for~~ efficient transmission

→ Decoding is the opposite process — the conversion ~~into a~~ of an encoded ~~formate~~ back into the original sequence of characters

17/07/19

→ When a process is Executed by the CPU and when a user Request for another process then This will create disturbance for the running process. This is called Interrupt

Types of Interrupts and How to Handle Interrupts

Interrupts:

In early years of computing processor ~~has to~~ wait for the signal for processing, so processor has to check each and every hardware and software program in the system if it has any signal to process. This method of checking the signal in the system for p040rocessing is called polling method. In this method the problem processor has to waste number of clock cycles just for checking the signal in the system, by this processor will become busy unnecessarily. If any signal came for the process, processor will take some time to process the signal due to the polling process in action. So system performance also will be degraded and response time of the system will also decrease.

So to over this problem engineers introduced a new mechanism, in this mechanism processor will not check for any signal from hardware or software but instead hardware/software will only send the signal to the processor for processing. The signal from hardware or software should have highest priority because processor should leave the current process and process the signal of hardware or software. This mechanism of processing the signal is called interrupt of the system.

What is an Interrupt?

Interrupt is a signal which has highest priority from hardware or software which processor should process its signal immediately.

Types of Interrupts:

Although interrupts have highest priority than other signals, there are many type of interrupts but basic type of interrupts are

1. **Hardware Interrupts:** If the signal for the processor is from external device or hardware is called hardware interrupts. Example: from keyboard we will press the key to do some action this pressing of key in keyboard will generate a signal which is given to the processor to do action, such interrupts are called hardware interrupts. Hardware interrupts can be classified into two types they are

o **Maskable Interrupt:** The hardware interrupts which can be delayed when a much highest priority interrupt has occurred to the processor.

o **Non Maskable Interrupt:** The hardware which cannot be delayed and should process by the processor immediately.

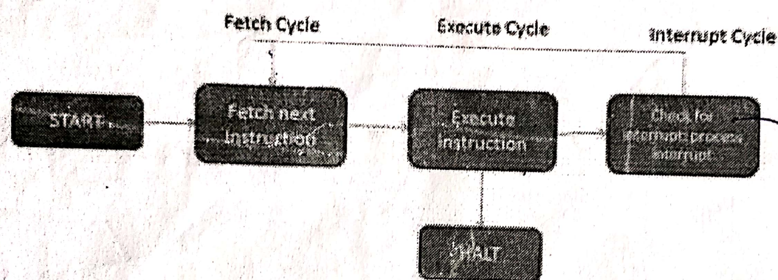
2. **Software Interrupts:** Software interrupt can also divided in to two types. They are

o **Normal Interrupts:** the interrupts which are caused by the software instructions are called software instructions.

o **Exception:** unplanned interrupts while executing a program is called Exception. For example: while executing a program if we got a value which should be divided by zero is called a exception.

Interrupt Cycle:

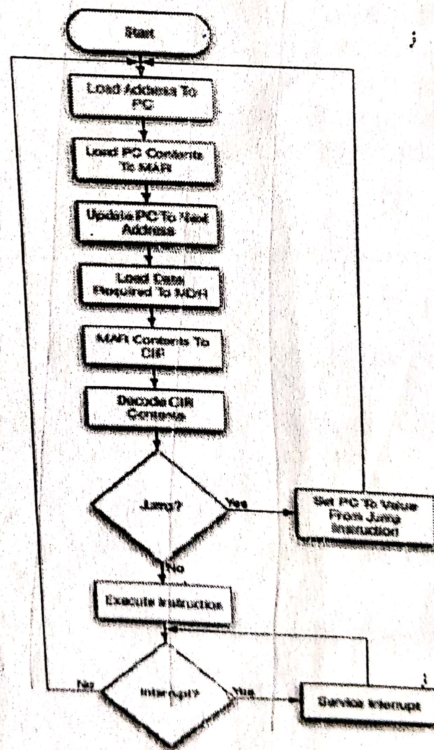
- An instruction cycle (sometimes called fetch-and-execute cycle, fetch-decode-execute cycle, or FDX) is the basic operation cycle of a computer. It is the process by which a computer retrieves a program instruction from its memory, determines what actions the instruction requires, and carries out those actions. This cycle is repeated continuously by the central processing unit (CPU), from bootupto when the computer is shut down.

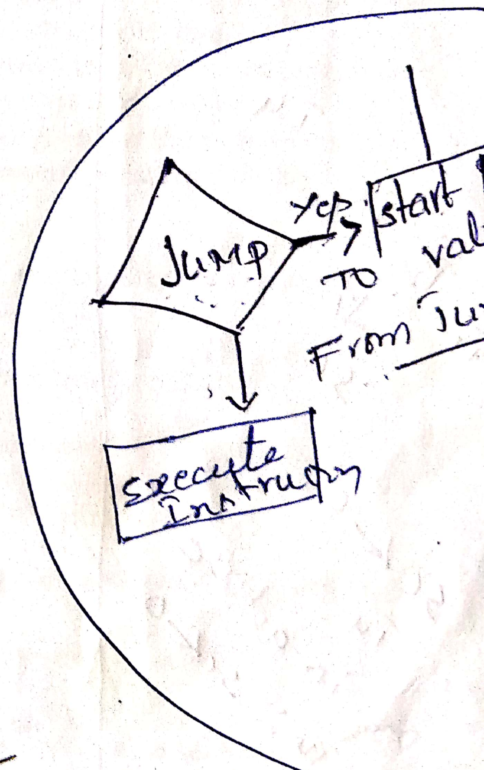


Block diagram of Interrupt Cycle

- After the execute cycle is completed, a test is made to determine if an interrupt was enabled (e.g. so that another process can access the CPU)
- If not, instruction cycle returns to the fetch cycle
- If so, the interrupt cycle might perform the following tasks: (simplified...)
- move the current value of PC into MBR

- move the PC-save-address into MAR
- move the interrupt-routine-address into PC
- move the contents of the address in MBR into indicated memory cell
- continue the instruction cycle within the interrupt routine
- after the interrupt routine finishes, the PC-save-address is used to reset the value of PC and program execution can continue





Microprogrammed CO

-> seauence of micro instructions in microprogramming language.

-> midway b/w H/w & s/w

-> it generates a set of Control lines (signals)

-> It is Easy to design, test & implements

-> It is flexible to Modify

-> Consist of
* ctrl signals
* " variable
* " word
* " memory
* Micro instruction
* Micro program

->* group of bits

->* Binary variable specify Micro operation that is Called Control variable

->* string of 1 & 0 represents Control variable called Control word

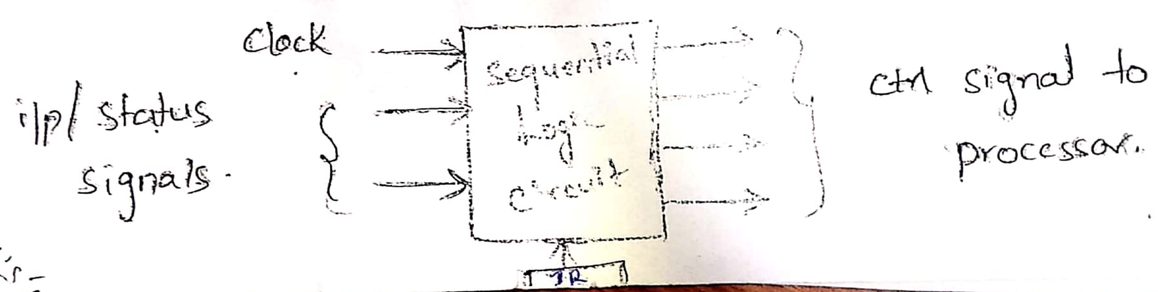
->* Control Memory Contains Control word

->* Containing Control word stored in Control memory it specify ctrl signals for a execution of Micro operation

Scanned by CamScanner

⇒ There are two techniques that implement C.U. They are.

(i) Hard-wired control unit: the design of Hardwired ctrl unit involves the use of fixed instructions, fixed logical blocks and most of the RISC architecture use Hardwired ctrl unit and the hardwired is organized by using multiple wires. Hence Design modifications are done among different component by making changes in wiring.



(2)

Micro Programmed Control Unit: It is second alternative to implement control unit. (most of the CISC architecture use this type of ctrl unit). In this micro operation specific in terms of binary variables and that micro operation is executed - The modification with done by updating/ modifying the Registere. when we adding new micro Instructions

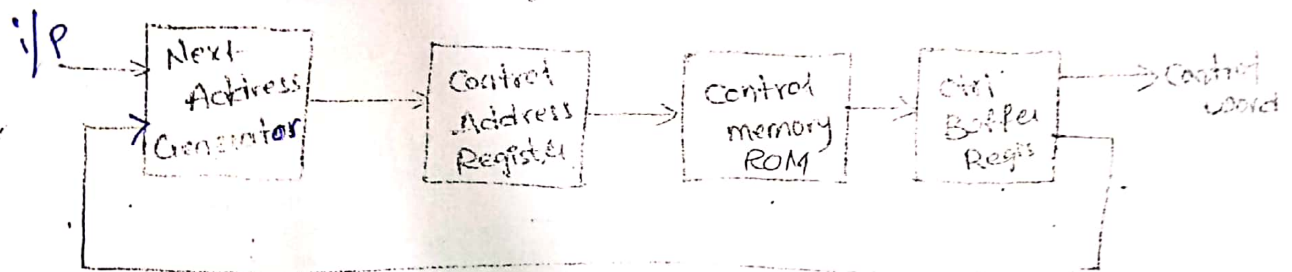


fig Micro programmed ctrl unit

NAG: In this type of CU include NAG that provides sequence of instruction to be read from the control memory

CAR: It is a temparary Register located in b/n NAG & CM it stores the address of certain micro instructions.

Control memory: It is a memory unit like Rom. It can read the instructions temparaly. the writing operations will not done in this because it have only Reading Capability.

CBR: The Reading values will be stored in CBR and it generates control word, which should be executed Next. sometimes, it is called as pipeline Register

Control word: The string of 0's & 1's are called control word which can perform various operations on the system) (4)

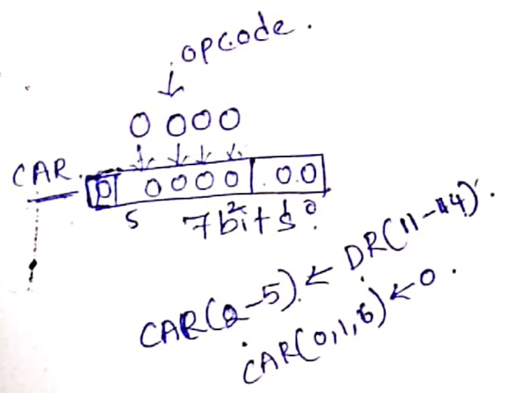
Scanned by CamScanner

Address Sequencing

① → In control memory the micro Instructions are store in the form of groops. The groups are specified by routine (→ regular procedure); each instruction has its own Routine for executing the instructions.

→ The h/w which is capable to control sequence of micro instructions, must have the capable of sequencing of address within routine and able to transfer from one Routine to another routine.

→ the transformation of Data from Main memory to Cache memory is called Mapping.



Instruction code

Mapping Logic

status bit — Branch Logic — Multiplexer — Sub routine BR

Clock → CAR (Control Address Register)

Increment

Control memory

Select a status bit

fig

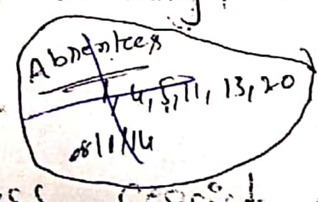
for executing / Arrange the Instruction Sequentially

→ Initially, the Instructions will be stored in the main memory. whenever the system power is on, the fetch (→go for and bring back) routine (→function) is activate. and the address of first instruction of the program will be loaded into control Address Register (CAR)

→ After storing the address in CAR, the control memory read the effective address of the CAR

(b) The effective address reached to the control memory through the branch condition and the reading data will be stored in the sub-routine buffer register (SBR) for feature purpose

Because, the SBR consists of some registers for storing the Return address, the Return address can't be stored in control memory, because the unit has no writability capacity.



→ If the address consist of any branch condition, then it will assign to the branch logic and the branch logic and SBR values will be assign to the Multiplexer to generating the next address

→ After completing one Instruction the SBR will be incremented

Scanned by CamScanner

and the control must return to the fetch routine

this process will be done untill the completion of Instructions, which were arranged in sequentially.

Here, we are used mapping process to transfer the instruction code from main memory to control memory

→ Address Sequencing Capabilities

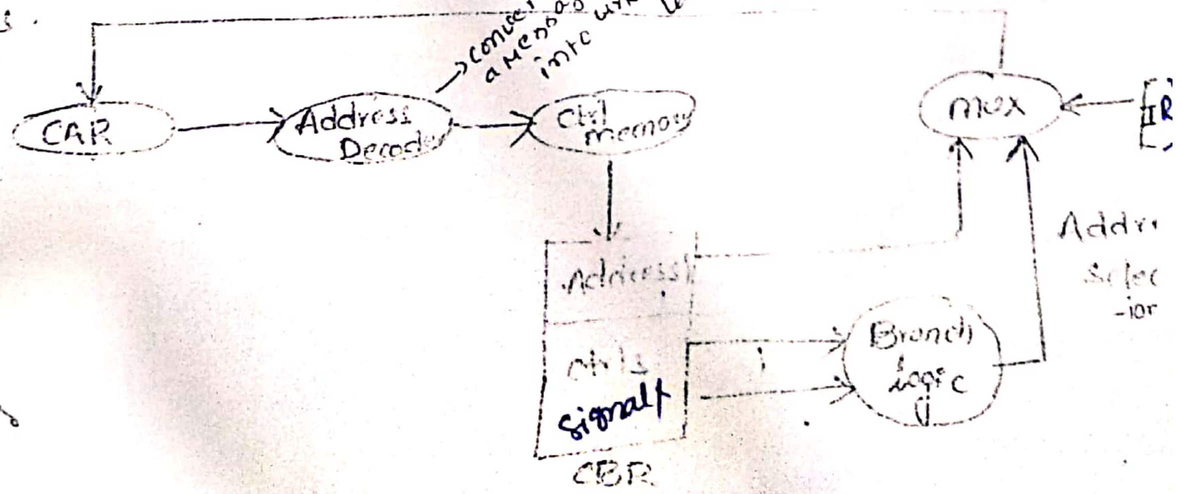
1) Incrementing the CAR, SBR
2) check the branch logic based on status bit.
3) mapping process done from main memory to ctrl memory
4) A facility for call & Return subroutine

Address Sequencing Techniques :

In order to execute the next instruction address derivation, which should be executed. There are 3 Techniques.

1) Single Address Technique :

In this technique: only one field is used for specific address.

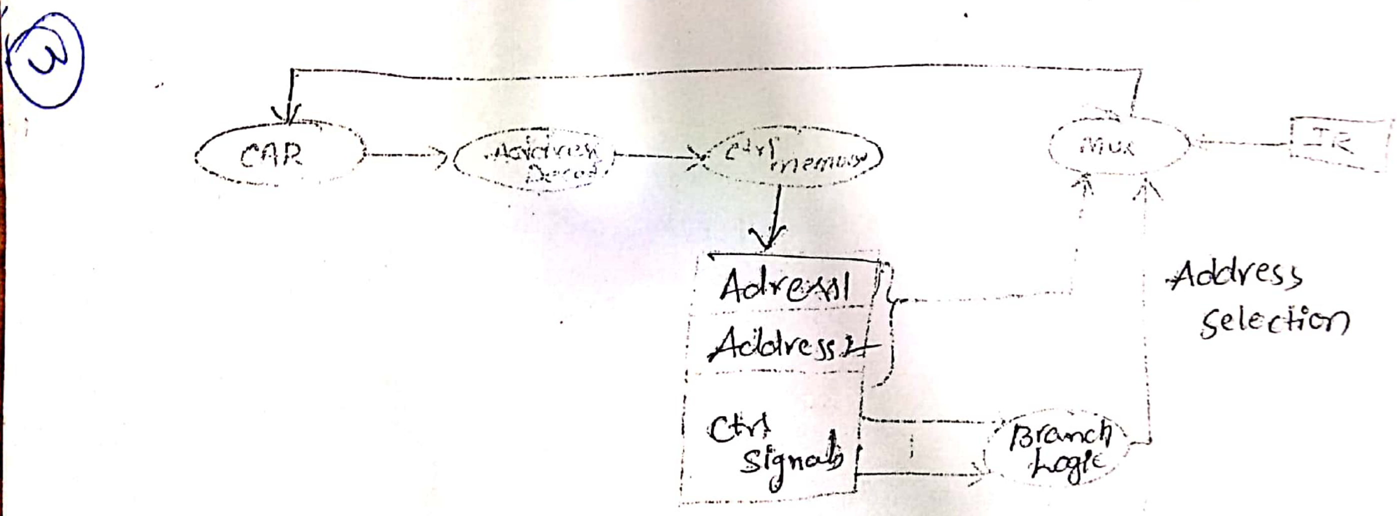


II C.S.E
7/1/14
II C.S.E
08/01/14
not writing

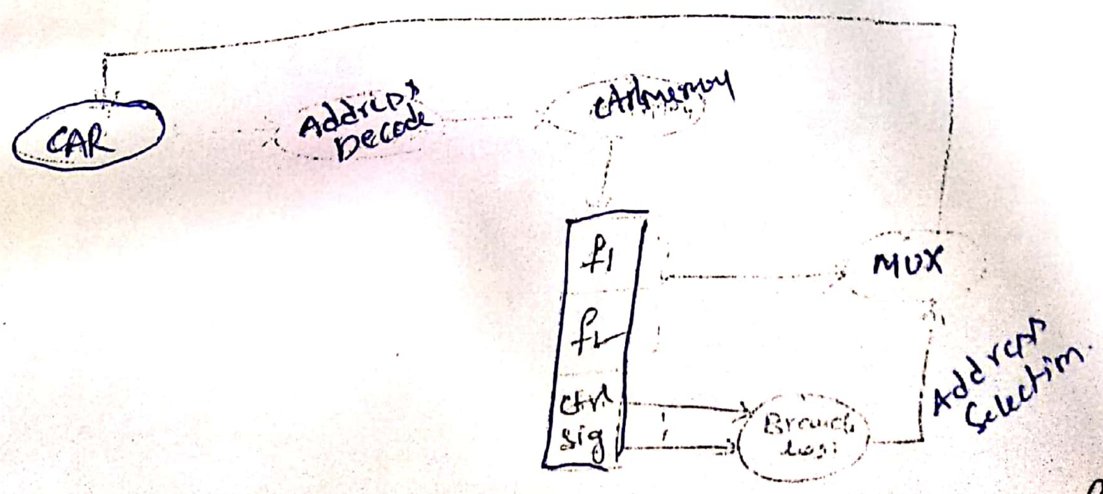
The next address can be calculated from the address field in the IR (or) next sequential address.

→ The Mux selects any one of these address based on the address selected by control logic.

2) Two - Address field : It consists of 2 address fields & the control bits are given to branch logic which produces address selection signal. The content of both address field & IR are given to mux. then it transfer only one address to the CAR. which is then decoded by add decoder



3) Variable Address Field



In this technique uses two micro instructions format.

Scanned by CamScanner

In both F_1 & F_2 the first bit indicates the Type format and the rest of bits are enable the ctrl sig. The next address acquired after executing two Instru...

(a) Mapping of Instructions:

Scanned by CamScanner

Micro program Example

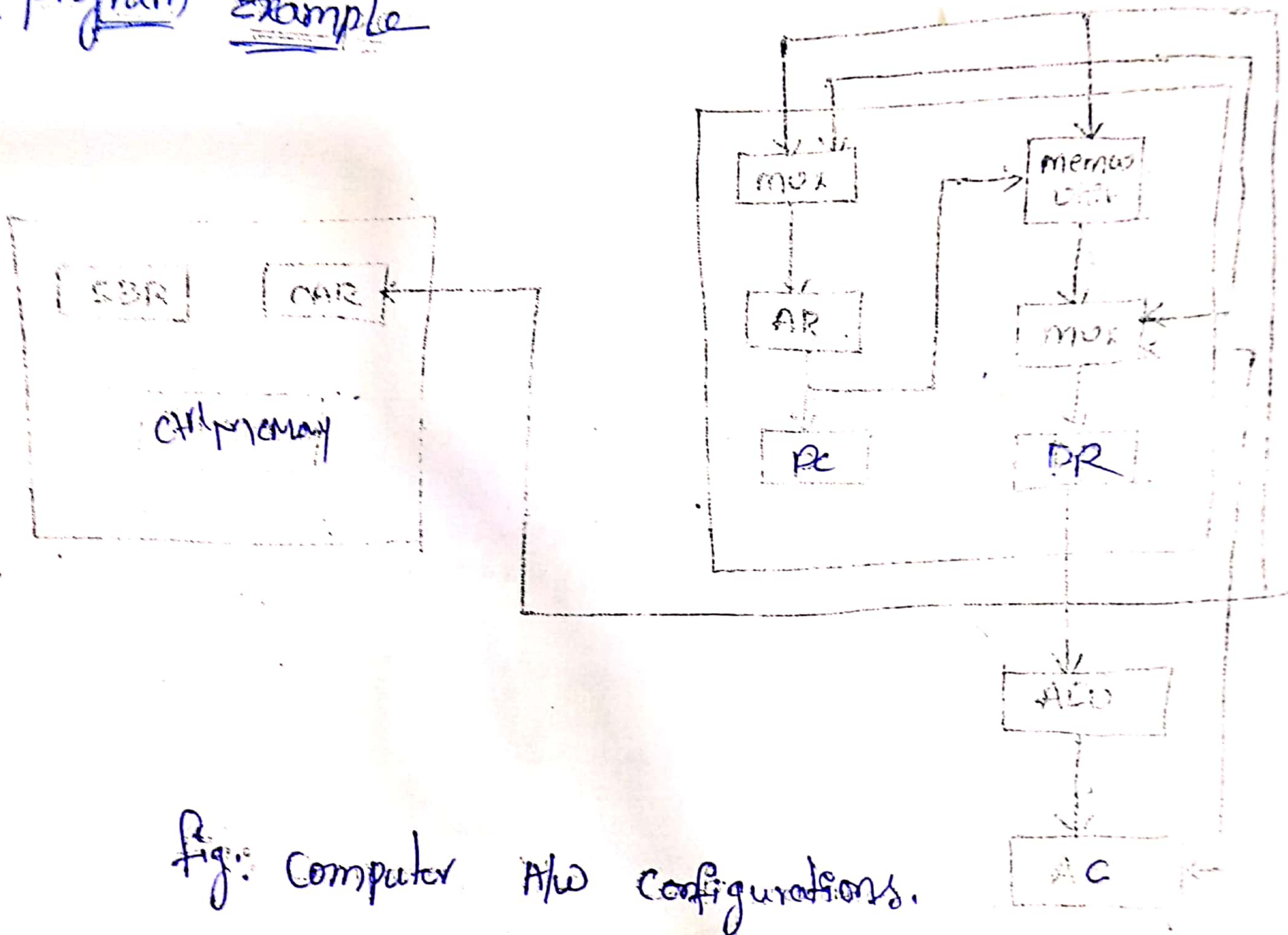


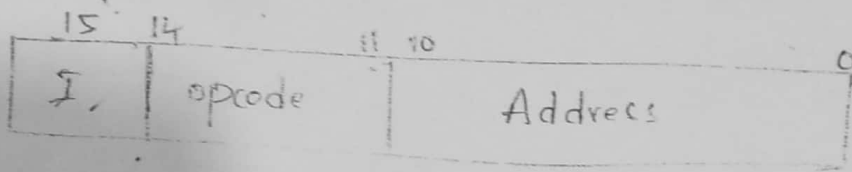
fig: Computer H/w configurations.

Scanned by CamScanner

Instruction format :

(6)

The computer instruction concist of 16 bits in 3 fields. One bit is field for indirect addressing, represented by I and 4-bit field for opcode and 11-bit is used for address.

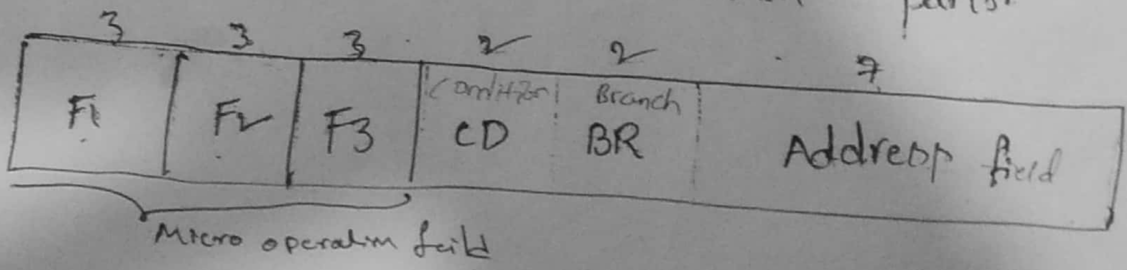


(a) instruction format

In figure the opcode consist of 4 bits i.e. having 16 possible memory location / references. Some of those are

Symbol	opcode	Description
ADD	0000	AC ← AC + M[EA]
BRANCH	0001	if (AC<0) then PC←EA
STORE	0010	M[EA] ← AC
EXCHANGE	0011	AC ← M[EA], M[EA] ← AC

Micro Instruction format : The micro instruction format in the control memory having 20 bits. These 20-bits are divided into four functional parts.



AD → Address field , CD → Condition for Branching
BR → Branch field , f_1, f_2 & f_3 are micro operation field.

- The CD field (consists) selects status bit condition and BR specifies the type of branch to be used. The AD field contains the address. it is seven bi[illegible] since the control memory having 128 words.

- The micro operations are sub-divided into 3-fie[illegible] of three-bits each. the 3-bits in each field are encoded to specify seven distinct operations

Scanned by CamScanner

Symbolic Micro Program:

The Computer Instruction can be defined in 2 w those are symbolic and binary form.

The Symbolic micro program can be written in Natu language. This representation is usefull for writing the micro program in assembly language format. Thi can be easily understandable by user.

The micro Instruction have four fields, so, these be represented by five fields i.e; one field is used for label remaining four for micro Instructn 4 fie

- The label field must be end with colon(:) & the micro operations consist of more than one symbols separated by comma(,)

- The CD field having symbols U, I, S, Z and BR field having JMP, CALL, RET, MAP.

Scanned by CamScanner

→ The AD field consist of symbols like FETCH, NEXT ---

Eg:

Label	Symbol	CD (— condition)	BR	Address
Add:	Nop (→ No operation)	I (→ Indirect)	CALL	INDRCT
	READ	U (→ unconditional)	Jmp	NEXT
	ADD	U	Jmp	FETCH
STORE:	Nop	I	CALL	INDRCT
	ACTDR	U	Jmp	NEXT
	WRITE	U	Jmp	FETCH
FETCH:	PCTAR	U	Jmp	NEXT
	READ, INCPC	U	Jmp	NEXT
	DRTAR	U	Jmp	
EXCHANGE:	Nop	U	Jmp	NEXT
	ACTDR, DRTAC	U	Jmp	NEXT
	WRITE	U	Jmp	FETCH

Examples for symbolic micro Instruction

Binary micro program: Binary micro program is in the format of 0's & 1's. it is machine level language i.e; The symbolic micro program will be translate into binary micro program. This can be easily understandable by system. This program also consist

	f_1	f_2	f_3	CD	BR	AD
Add:	000	000	000	01	01	00000001
	000	100	000	00	00	00000010
	001	000	000	00	00	00000011
STORE:	000	000	000	01	01	0000 0010
	000	101	000	00	00	0000 0011
	111	000	000	00	00	00000100
Fetch:	110	000	000	00	00	00000101
	000	100	101	00	00	0000 0111
	101	000	000	00	00	1000 0000
Exchange:	000	000	000	00	00	1000 0001
	100	101	000	00	00	1000 0010
	111	000	000	00	00	1000 0011

Examples for Binary program Representation

Scanned by CamScanner

Design of Control UNIT 6/8/17

(1)

→ the bits of Microinstruction are usually divided into feilds, with each field defining a distinct separate functions

→ The various fields Encountered in instruction formats provide Control bits to initiate microoperatio in the system, Special bits to specify the way that the next address is to be evaluated, and an address field for branching.

→ the number of Control bits that initiate microoperations Can be reduced by grouping mutually Exclusive variables into fields and Encoding the k bits in each field to provide 2^k microoperations.

→ Each field requires a decoder to produce the Corresponding Control signals.

→ this method reduces the size of the microinstruction bits but requires additional Hardware External to the Control memory.

→ It also increases the delay time of the Control signals because they must propagate through the decoding Circuits.

Scanned by CamScanner

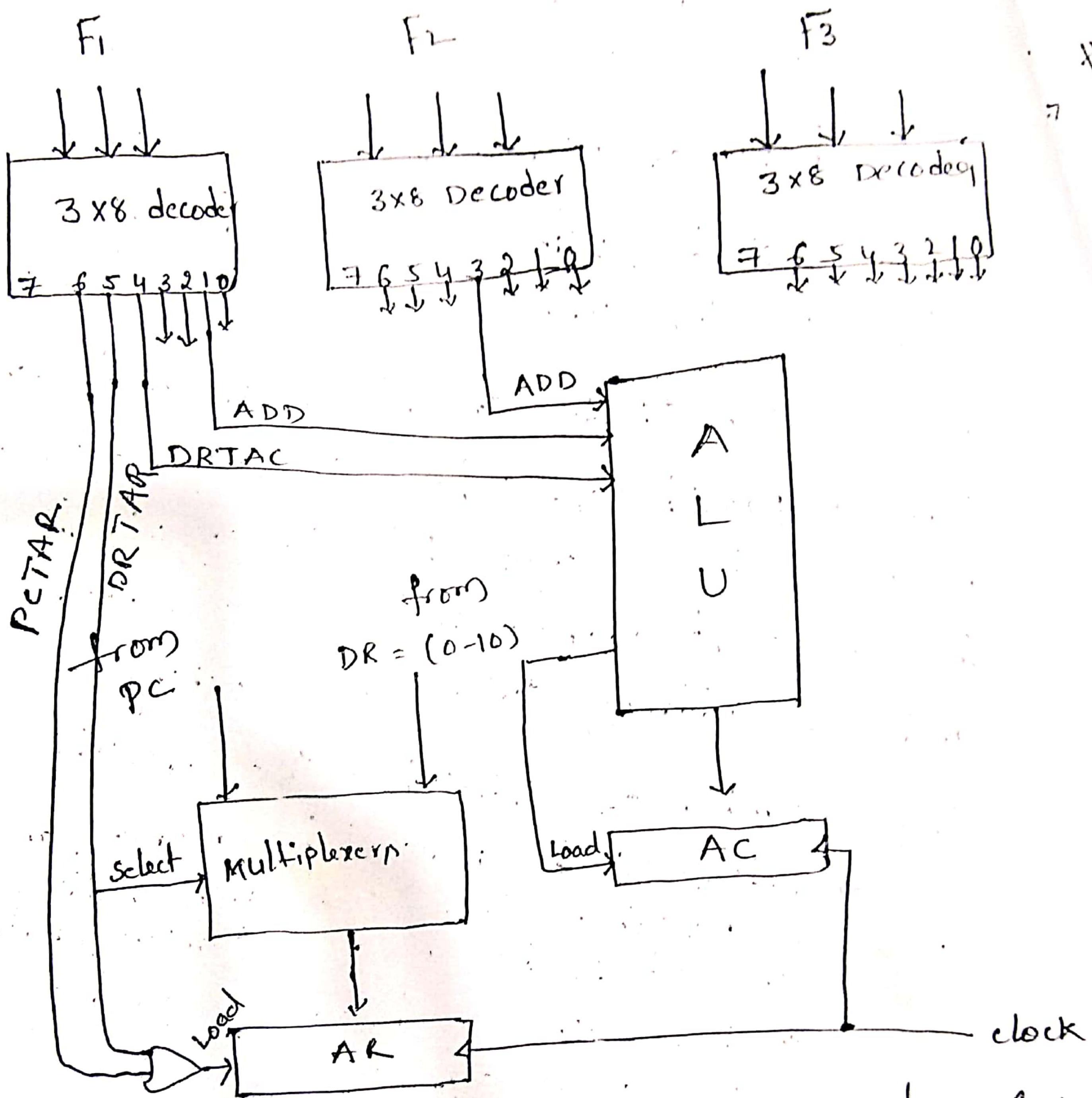


fig: Decoding of Microoperation fields

Scanned by CamScanner

-> the above fig. three decoders and some of the connections that must be made from their o/p's

-> Each of the three fields of the micro instruction presently available in the output of control memory are decoded with a 3×8 decoder to provide eight o/p's.

-> Each of these o/p's must be connected to the proper circuit to initiate the corresponding microoperation.

-> for example, when F1 = 101 (binary 5), the next clock pulse transition transfers the content of DR (0-10) to AR (symbolized by DRTAR.

-> Similarly, when F1 = 110 (binary 6) there is a transfer from PC to AR (symbolized by PCTAR)

-> As shown in fig, outputs 5 and 6 of decoder F1 are connected to the load input of AR so that when either one of these o/p's is active, information from the multiplexers is transferred to AR.

Scanned by CamScanner

→ the multiplexers select the information from DR when o/p's 5 is active and from PC when o/p 5 is inactive

→ the transfer into AR occurs with a clock pulse transition only when output 5 or o/p 6 of the decoder are active.

→ the others o/p's of the decoders that initiate transfers between registers must be connected in a similar fashion.

→ Arithmetic logic unit can be designed. the i/p's this unit came from the o/p's of decoder associate with the symbols AND, ADD, DRTAC, respectively and the i/p of this unit will be loded in AR

→ Here it also use clock pulses to measure the time to decode the address and store data.

Scanned by CamScanner

Addressing Modes

→ Addressing Mode provide different ways for access an address to given data to processor.

→ Addressing modes specify the location of an operand.

→ the term addressing modes refers to the way in which the operand of an instruction is specified

→ the addressing Mode specifies a rule for interpreting (or) Modifying the address field of the instruction before the operand is actually executed.

Scanned by CamScanner

(1) Addressing Modes

→ the different ways in which a source operand is denoted in an Instruction is known as addressing modes,

→ (1) Immediate addressing mode:

→ the addressing mode in which the data operand is a part of the instruction itself is known as Immediate addressing mode

Ex: MOV CX, 4929 H, (Move the data 4929H CX)

ADD AX, 2387 H,

MOV AL, FFH

→ (2) Register addressing mode

→ It means that the register is the source of an operand for an instruction.

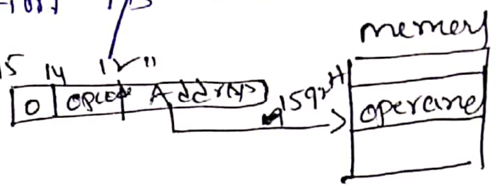
Ex: MOV CX, AX ; Copies the Contents of The 16 bit AX register into the 16-bit CX regi

Scanned by CamScanner

(7)

3) Direct addressing Mode :-

→ the addressing mode in which the effective address of the memory location is written directly in the instruction



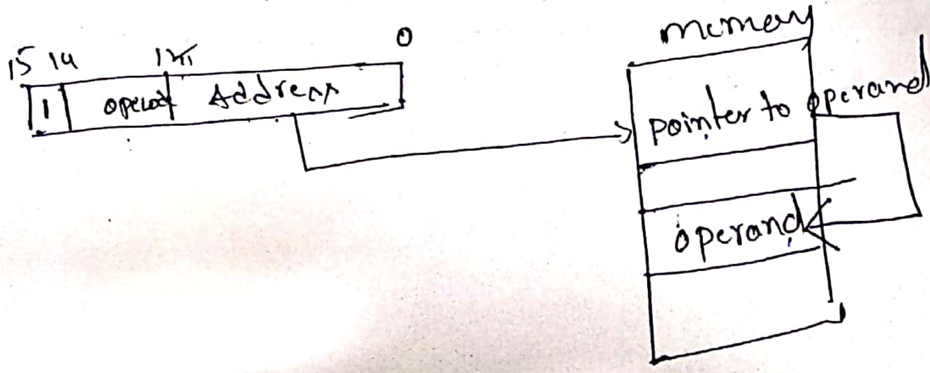
Ex:- MOV AX, [1592H], MOV AL, [0300H]

4) Register indirect addressing Mode :-

→ This addressing mode allows data to be addressed at any memory location through an offset address held in any of the following registers : BP, BX, DI & SI

Ex: MOV AX, [BX];

→ suppose the register BX contains 4895H, then the contents 4895H are moved to AX

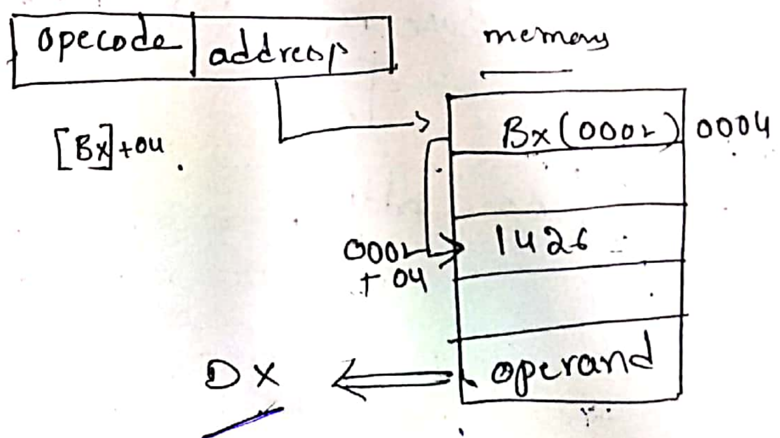


Scanned by CamScanner

5) => Based Addressing Mode :-

→ In this A.M, the offset address of the operand is given by the sum of contents of Bx / Bp Reg & 8/16 bit displacement

Ex:- MOV Dx, [Bx+04] → displacement



6) => Indexed Addressing Mode :-

→ In this AM, the operand offset address is found by adding the contents of SI or DI reg & 8 bit / 16 bit displacement.

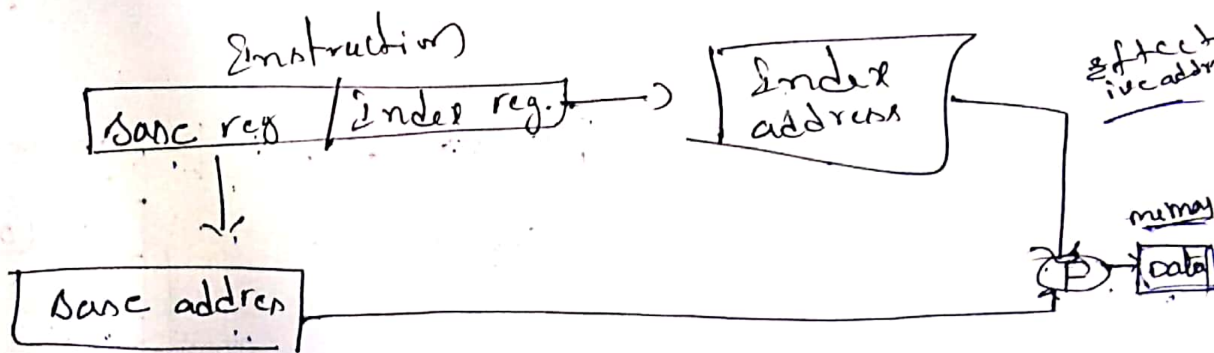
Ex:- MOV Bx, [SI+16] ; → 16 bit displacement value

Scanned by CamScanner

(7) => Based-Indexed Addressing Mode:

→ In this A.M, the offset addressing of operand is computed by summing the base reg. to the content of the indexed reg.

Ex: ADD CX, (AX+SI)

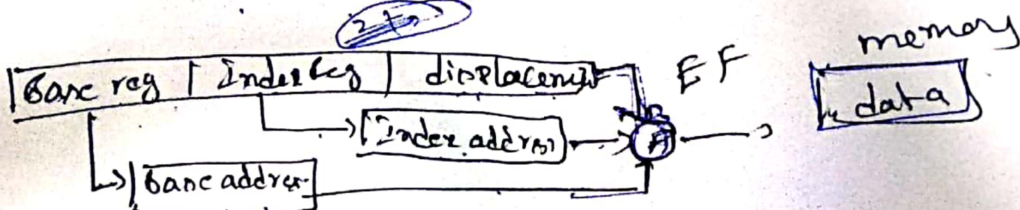


(8) => Based index with displacement A.M:

→ # In this A.M, the operands offsets computed by adding the base reg content of index reg. & 8 or 16 bit displacement

Ex: MOV AX, [BX + DI + 08]

ADD CX, [BX + SI + 16]



Scanned by CamScanner

* ALU
* Flag register
* Timing & Control " Registers

General purpose register (GPRs) :

→ 8086 μp Consists of 4 GP → General purpose Registers 16-bit ⇓ two 8bit registers

AX	16 bit
BX	16 bit
CX	"
DX	"

higher order ← AH, lower order ← AL

AH AL, BH BL, CH CL, DH DL, ⇒ 8bit Registers

AX, BX, CX, DX ⇒ 16 bit

Scanned by CamScanner

General purpose register

There are 8 general purpose registers, i.e., AH, AL, BH, BL, CH, CL, DH, and DL. These registers can be use individually to store 8-bit data and can be used in pairs to store 16bit data. The valid register pairs are AH a AL, BH and BL, CH and CL, and DH and DL. It is referred to the AX, BX, CX, and DX respectively.

- **AX register** – It is also known as accumulator register. It is used to store operands for arithmetic operations.
- **BX register** – It is used as a base register. It is used to store the starting base address of the memory area within the data segment.
- **CX register** – It is referred to as counter. It is used in loop instruction to store the loop counter.
- **DX register** – This register is used to hold I/O port address for I/O instruction.

Scanned by CamScanner

Data Transfer and Manipulation (19)

→ Computers provide an extensive set of instructio to give the user the flexibility to carry out various computational tasks.

→ the actual operations available in the instruction set are not very different from one computer to another.

→ Most computer instructions can be classified into three categories.

1. Data transfer instructions
2. Data manipulation instructions
3. program Control instructions

Data transfer instructions

→ Data transfer instructions cause transfer of data from one location to another without changing the binary information content.

→ the most common transfers are between memory and processor registers, between processor registers and i/o.

fig: Data Transfer Instruction

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Input	IN
output	OUT
push	PUSH
pop	POP

→ the "Load" instruction has been used mostly to designa a transfer from memory to a processor register, usually an accumulator

→ the "store" instruction designates a transfer from a processor register into memory

→ the "Move" instruction has been used in Computers with multiple CPU registers to designate a transfer from one register to another.

→ It has also been used for data transfers between CPU registers and memory or between two memory words.

→ the "Exchange" instruction Swaps information between two registers or a register and a memory word.

Scanned by CamScanner

-> the "input and output" instructions transfer data among processor registers and input or output terminals.

-> the "push and pop" instructions transfer data between processor registers and a memory stack.

=> Data Manipulation Instructions

-> Data Manipulation is the process of changing data in an effort to make it easier to read or more organized.

-> Data manipulation instructions perform operations on data and provide the Computational Capabilities for the Computer.

-> the data manipulation instructions in a typical Computer are usually divided into three basic types.

1. Arithmetic instructions
2. Logical instructions
3. Shift instructions.

Scanned by CamScanner

* Arithmetic Instructions

→ the four basic arithmetic operations are addition, subtraction, multiplication, and division

→ Most Computers provide instructions for all four operations.

→ the four basic arithmetic operations are sufficient for formulating solutions to scientific problems when expressed in terms of numerical analysis methods.

→ the add, subtract, multiply, and divide instructions may be available for different types of data.

→ An arithmetic instruction may specify fixed point or floating point data, binary or decimal data

fig: Typical Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
subtract	SUB
Multiply	MUL
Divide	DIV
Add with Carry	ADDC
subtract with borrow	SUB B

Scanned by CamScanner

(23) (5)

* Logical Instructions

-> Logical instructions perform binary operations on strings of bits stored in registers.

-> Some typical logical and bit manipulation instructions are listed in table:

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement Carry	COMC
Enable interrupt	EI
Disable interrupt	DI

* Shift Instructions

-> Instructions to shift the content of an operand are quite useful and are often provided in several variations.

-> Shifts are operations in which the bits of a word are moved to the left or right.

Scanned by CamScanner

fig: Typical shift instructions

Name	Mnemonic
logical shift right	SHR
logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through Carry	RORC
Rotate left through Carry	ROLC

→ This is a shift right operation with the end bit remaining the same

→ the Arithmetic shift left instruction inserts 0 to the End position and is identical to the logical shift left instruction.

→ the rotate instructions produce a circular shift.

① 162

(2)

UNIT - IV

COMPUTER ARITHMETIC

② Copies.

Addition and Subtraction :-

We can perform addition and subtraction of two binary numbers in three different ways:

1) Signed - Magnitude Representation
2) Signed - 1's Complement "
3) Signed - 2's " "

The Signed - 2's complement representation is used for performing arithmetic operations. The signed - magnitude representation is used for floating - point representation.

Addition and subtraction with signed - Magnitude data :-

Here, we are going to consider the magnitude of any two numbers i.e., A and B. There are eight different operations which are listed in below:

Operation	Add magnitudes	Subtract Magnitudes: when A > B	when A < B	when A = B
(+A) + (+B)	+(A + B)			
(+A) + (-B)		+(A - B)	-(B - A)	+(A - B)
(-A) + (+B)		-(A - B)	+(B - A)	+(A - B)
(-A) + (-B)	-(A + B)			
(+A) - (+B)		+(A - B)	-(B - A)	+(A - B)
(+A) - (-B)	+(A + B)			
(-A) - (+B)	-(A + B)			
(-A) - (-B)		-(A - B)	+(B - A)	+(A - B)

www.jntuworldupdates.org

The columns in the table show the actual operation to be performed with the magnitude of the numbers. The last column is needed to prevent a negative zero.

Algorithm :-

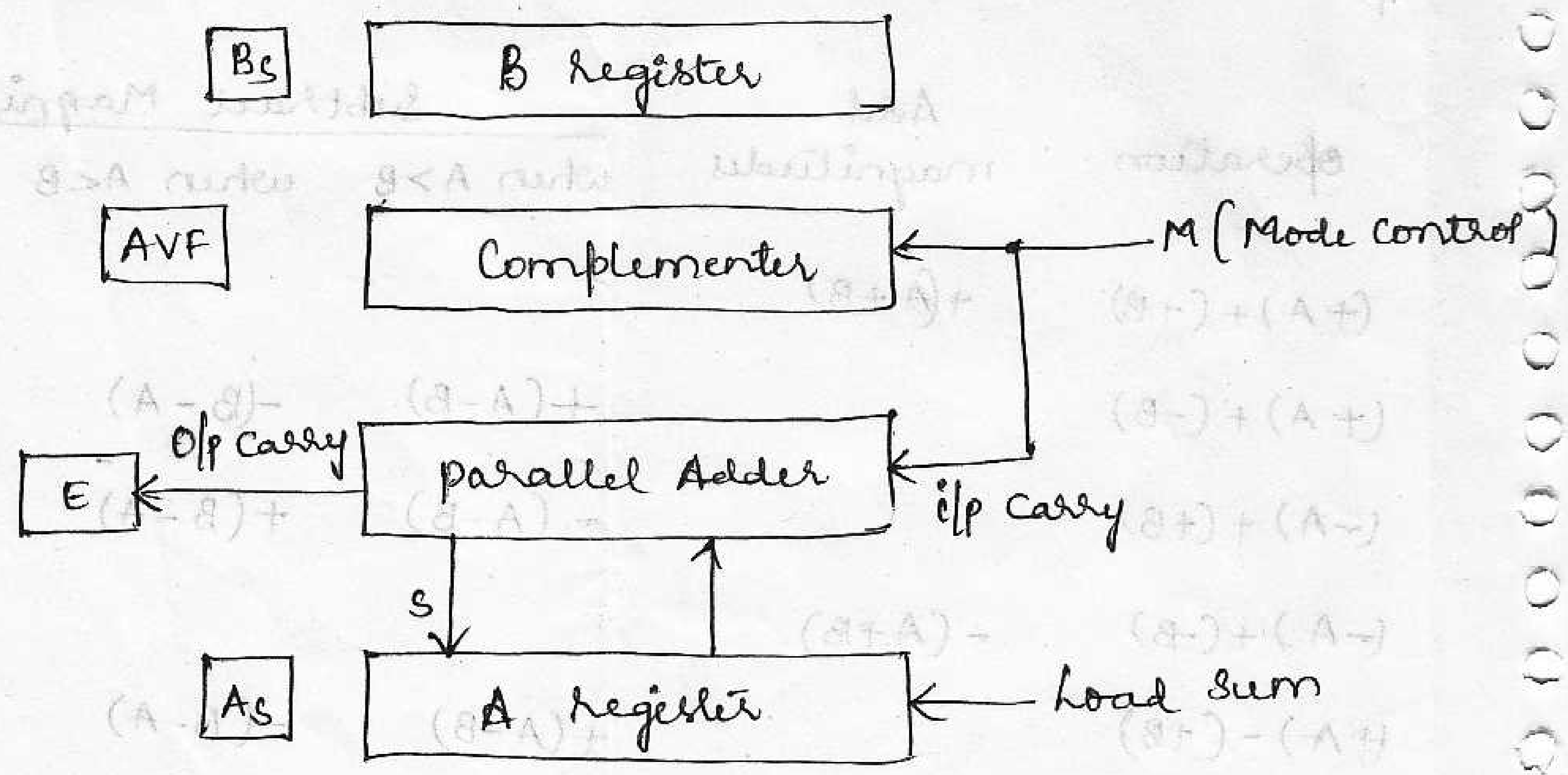
When the signs of A and B are identical, add the two magnitudes and attach the sign of A to the result.

When the signs of A and B are different, compare the magnitudes, subtract the smaller number from the larger.

Hardware implementation :-

To implement the two arithmetic operations with hardware, it is first necessary that the two numbers be stored in registers.

The h/w implementation for addition & subtraction is shown in fig.



Let A and B be two registers that holds the magnitudes of the numbers and As & Bs be two flip-flops that hold the corresponding signs. The result of the

www.specworld.in
www.smartzworld.com

operation may be transferred to a third register or the result is transferred into A or A_S.

First, a parallel-adder is needed to perform micro operation A+B. Second a Comparator circuit is needed to establish if A>B, A=B or A<B. The third, two parallel-subtractor circuits are needed to perform the microoperations, A-B & B-A.

The block diagram consists of registers A & B and sign flip-flops A_S & B_S. Subtraction is done by adding A to the 2's complement of B. The o/p carry is transferred to 'E'. The add overflow flip-flop (AVF) holds the overflow bit when A & B are added.

The addition of A plus B is done through the parallel Adder. The S (Sum) o/p of the adder is transferred to A-register. The M (Mode control) signal is also applied to the i/p carry of the adder. When M=0, the o/p of B is transferred to the adder, the i/p carry is '0' and the o/p of the adder is equal to the sum A+B. When M=1, the 1's complement of B is applied to the adder, the i/p carry is 1, o/p $S = A + \bar{B} + 1$.

<u>Hardware Algorithm</u>:-

The flowchart for the h/w algorithm is shown in fig.

The two signs A_S & B_S are compared by EX-OR gate. If the o/p of the gate is 0, the signs are identical. If the o/p of the gate is 1, the signs are different. For an add operation, the

www.jntuworldupdates.org

Subtract operation → Minuend in A, Subtrahend in B → $A_S \oplus B_S$ (=0 / =1)

Add operation → Augend in A, Addend in B → $A_S \oplus B_S$ (=1 / =0)

$EA \leftarrow A + \bar{B} + 1$, $AVF \leftarrow 0$

$EA \leftarrow A + B$

$AVF \leftarrow E$

E: =0 → $A < B$ → $A \leftarrow \bar{A}$ → $A \leftarrow A + 1$, $A_S \leftarrow \bar{A_S}$

E: =1 → $A \geq B$ → A: ≠0 / =0 → $A_S \leftarrow 0$

END (Result is in A & A_S)

Identical signs dictate that the magnitudes be added. For a subtract operation, different signs dictate that the magnitudes be added. The magnitudes are added with $EA \leftarrow A + B$, where EA is a register that combines E & A.

The two magnitudes are subtracted if the signs are different for an add operation or identical for a subtract operation. No overflow can occur if the numbers are subtracted so AVF is to '0'.

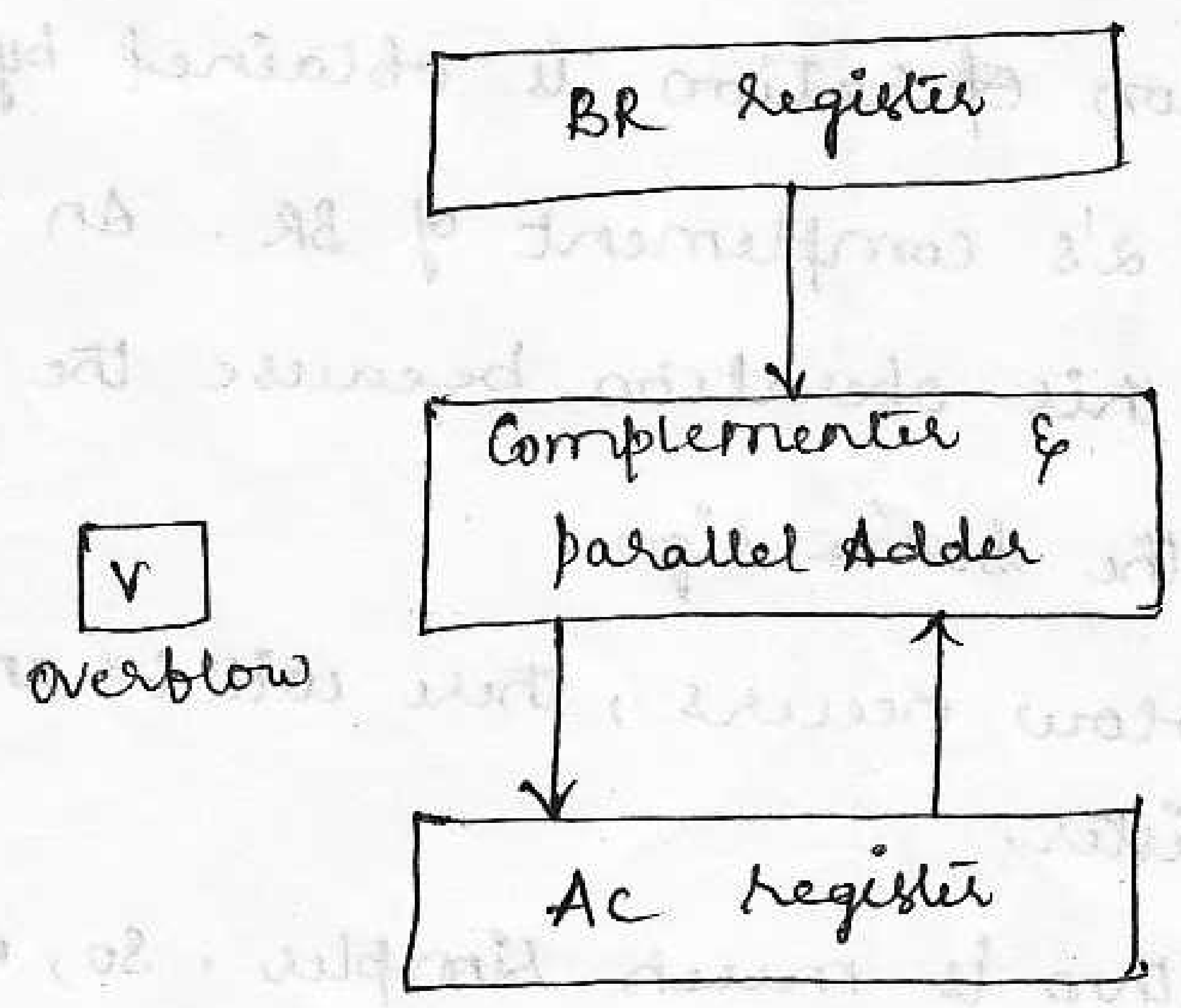
If $E = 1$, then the condition is $A \geq B$, if A is '0' then the result is correct result. If $E = 0$, then the condition i

Condition is A<B. For this, we are going to consider the 2's Complement of the value in A. This operation can be done one microoperation $A \leftarrow \bar{A}+1$. If the sign of the result is as same as the sign of A, so no change in A_s is required. When A<B, the sign of the result is the complement of the original sign of A. The complement of As is required to get the correct sign. The final result is found in register A & its sign in As.

Addition & Subtraction with signed 2's complement data :-

When two numbers of 'n' digits are added and the Sum occupies n+1 digits, than overflow is occurred. An overflow can be detected by inspecting the last two carries of the addition. When the two carries are applied to an ex-OR gate, the overflow is detected when the o/p of the gate is equal to 1.

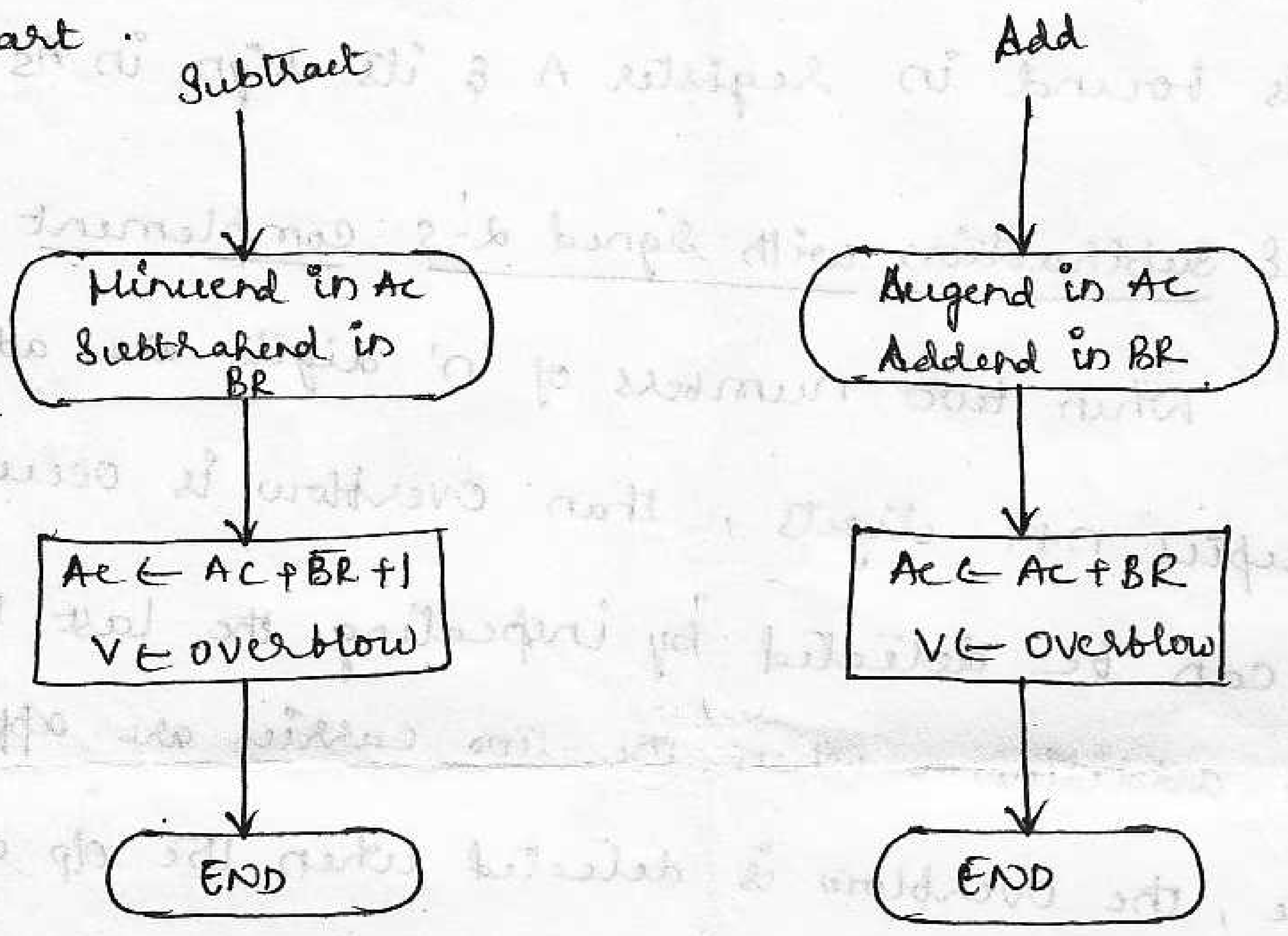
The reg. Configuration for the h/w implementation is shown in fig.



www.jntuworldupdates.org

The leftmost bit in AC & BR represent the sign bits of the number. The two sign bits are added or subtracted together with the other bits in the complementer & parallel Adder. The overflow flip-flop V is set to 1 if there is an overflow.

The algorithm for adding and subtracting two binary numbers in signed - 2's complement representation is shown in flowchart.



The sum is obtained by adding the contents of AC & BR. The overflow bit is set to 1 if the ex-OR of the last two carries is set to 1.

The subtraction operation is obtained by adding the content of AC to the 2's complement of BR. An overflow must be checked during this operation because the two numbers added could have the same sign.

If an overflow occurs, there will be an erroneous result in the AC register.

This algorithm is much simpler. So, most computers uses the signed-Magnitude representation.

www.specworld.in

www.smartzworld.com

www.jntuworldupdates.org

Multiplication Algorithms :-

The sign of the product is determined from the signs of the multiplicand and multiplier. If they are like, the sign of the product is +ve, If they are unlike, the sign of the product is -ve.

H/w implementation for Signed-Magnitude Data :-

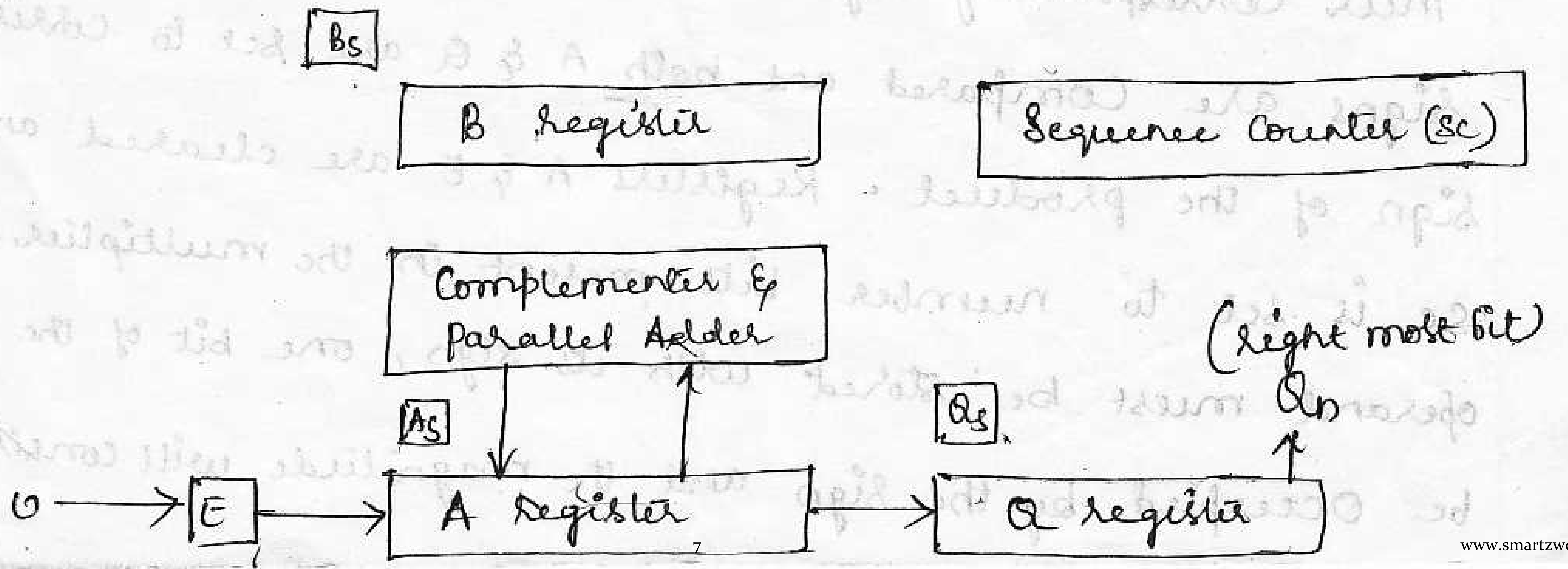
The Multiplication is implemented in a digital computer First, instead of providing registers to store & add, it is convenient to provide an adder for the summation of only two binary numbers and accumulate the partial products in a register.

Second, instead of shifting the multiplicand to the left, the partial product shifted to right.

Third, when the corresponding bit of the multiplier is 0, there is no need to add all zeros to the partial product.

Eg:
```
23     10111    Multiplicand
19   x 10011    Multiplier.
```

The h/w for multiplication consists is shown in fig.



www.specworld.in
www.smartzworld.com

www.jntuworldupdates.org

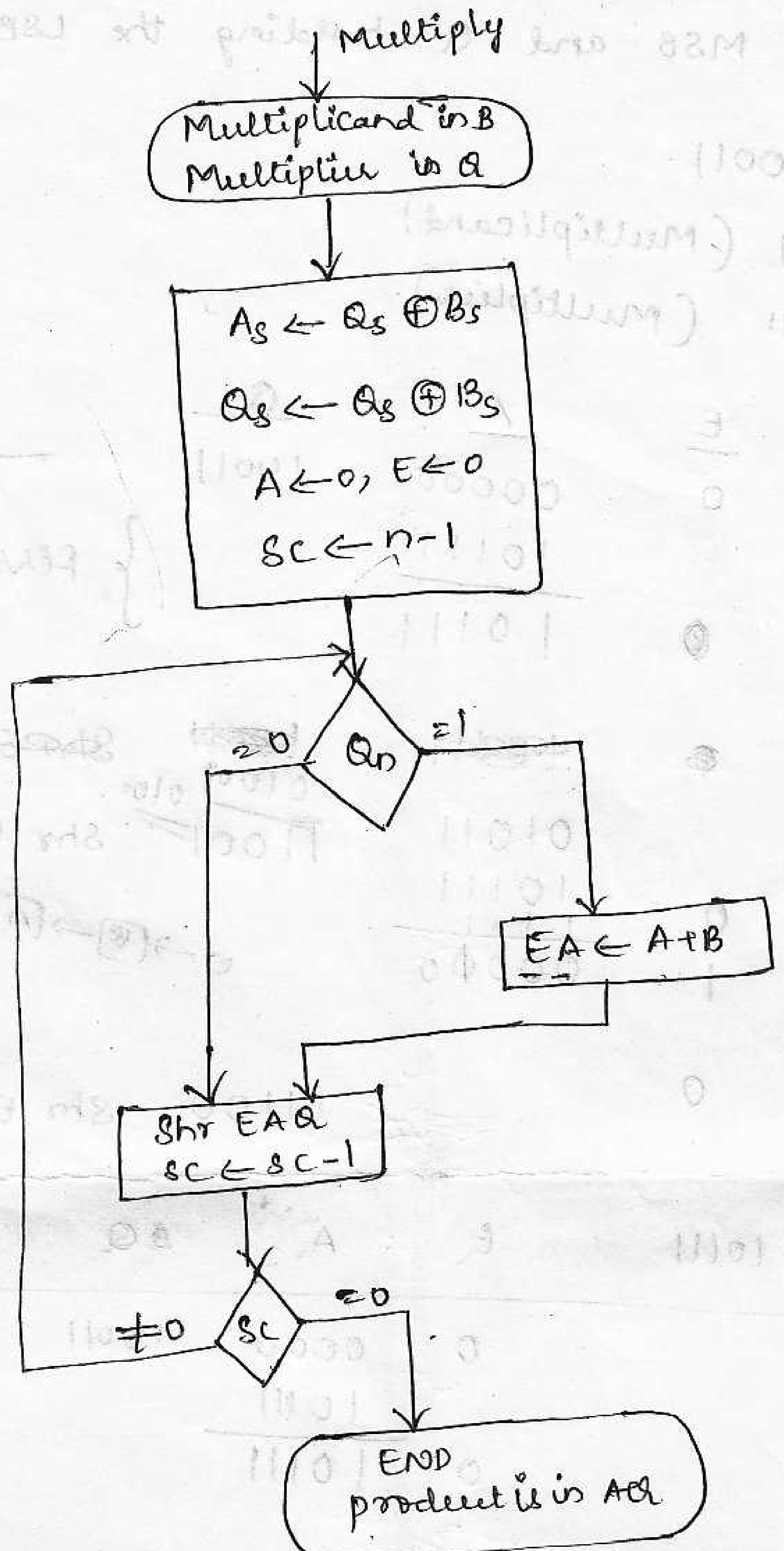
The multiplier stored in the Q register & its sign in Q_s. The Sequence Counter (SC) is initially set to a number equal the number of bits in the multiplier. The counter is decremented by 1 after forming each partial product. The counter is reaches to zero, the product is formed and the process stops.

Initially, the multiplicand is in B reg. and the multiplier is in Q. The sum of A & B forms a partial product which is transferred to the EA register. Both partial product & multiplier are shifted to the right. This shift will be denoted by the statement shr EAQ. The LSB of A is shifted into MSB of Q the bit from E is shifted into the MSB of A and '0' is shifted into E. After the shift, one bit of the partial product is shifted into Q, pushing the multiplier bits one position to the right. The right most flip-flop in register Q, designated by Q_n, will hold the bit of the multiplier.

H/w Algorithm :-

The fig. shows the flowchart of the hardware multiply algorithm.

Initially, the multiplicand is in B and the multiplier in Q. Their corresponding signs are in B_s & Q_s respectively. The signs are compared and both A & Q are set to correspond to the sign of the product. Registers A & E are cleared and sequence SC is set to number bits present in the multiplier. Since an operand must be stored with its sign, one bit of the word will be occupied by the sign and the magnitude will consists of n-1 bits.



After initialization, the low-order bit of the multiplier Q_n is tested. If it is 1, the multiplicand in B is added to the present partial product in A. If it is 0, nothing is done. Register EAQ is then shifted once to the right to form the new partial product. The sequence counter is decremented by 1 and its new value is checked. If it is not equal to zero, the process is repeated and if it is equal to zero, the process stops. The final product is available in both A & Q, with

A holding the MSB and Q holding the LSB.

Eg: 10111 x 10011.

B = 10111 (Multiplicand)

Q = 10011 (Multiplier).

B	SC	E	A	Q	
10111	5	0	00000	10011	← Initial values.
			10111		} First partial product
		0	10111		
	4	0	~~10011~~	~~10011~~ 0100 0100	~~ShrEAQ~~
			01011	11001	Shr EAQ } second partial products
	4	0	10111		0 → [E] → [A] → [Q]
		1	00010		
		0		01100	Shr EAQ] X

Multiplicand B = 10111	E	A	Q	SC	
	0	0000	10011	101	} First partial product.
		10111			
$Q_n=1$ Add	0	10111			
Shr EAQ	0	01011	11001	100	} second partial product.
		10111			
$Q_n=1$, Add	1	00010			
Shr EAQ	0	10001	01100	011	
$Q_n=0$, Shr EAQ	0	01000	10110	010	
$Q_n=0$, Shr EAQ	0	00100	01011	001	
		10111			
$Q_n=1$, Add B.	0	11011	01011	001	
& Shr, EAQ		01101	10101	000	

AQ = 01101 10101

www.jntuworldupdates.org

Booth Multiplication Algorithm :-

Booth algorithm gives a procedure for multiplying binary integers in signed - 2's complement representation. It operates on the fact that strings of 0's in the multiplier requires no addition but just shifting and a string of 1's in the multiplier from bit weight 2^k to weight 2^m can be treated as $2^{k+1} - 2^m$.

Eg: The binary number 001110 has a string of 1's from 2^3 to 2^1. Here k=3, m=1. The number can be represented as $2^{k+1} - 2^m$ $= 2^4 - 2^1 = 14$. Therefore, the multiplication M x 14, where M is the multiplicand and 14 is the multiplier. This can be done as $M \times 2^4 - M \times 2^1$, thus the product can be obtained by shifting the binary Multiplicand M four times to the left and subtracting 'M' shifted left once.

Booth algorithm requires examination of the multiplier bits and shifting the partial product. The following rules are to be required.

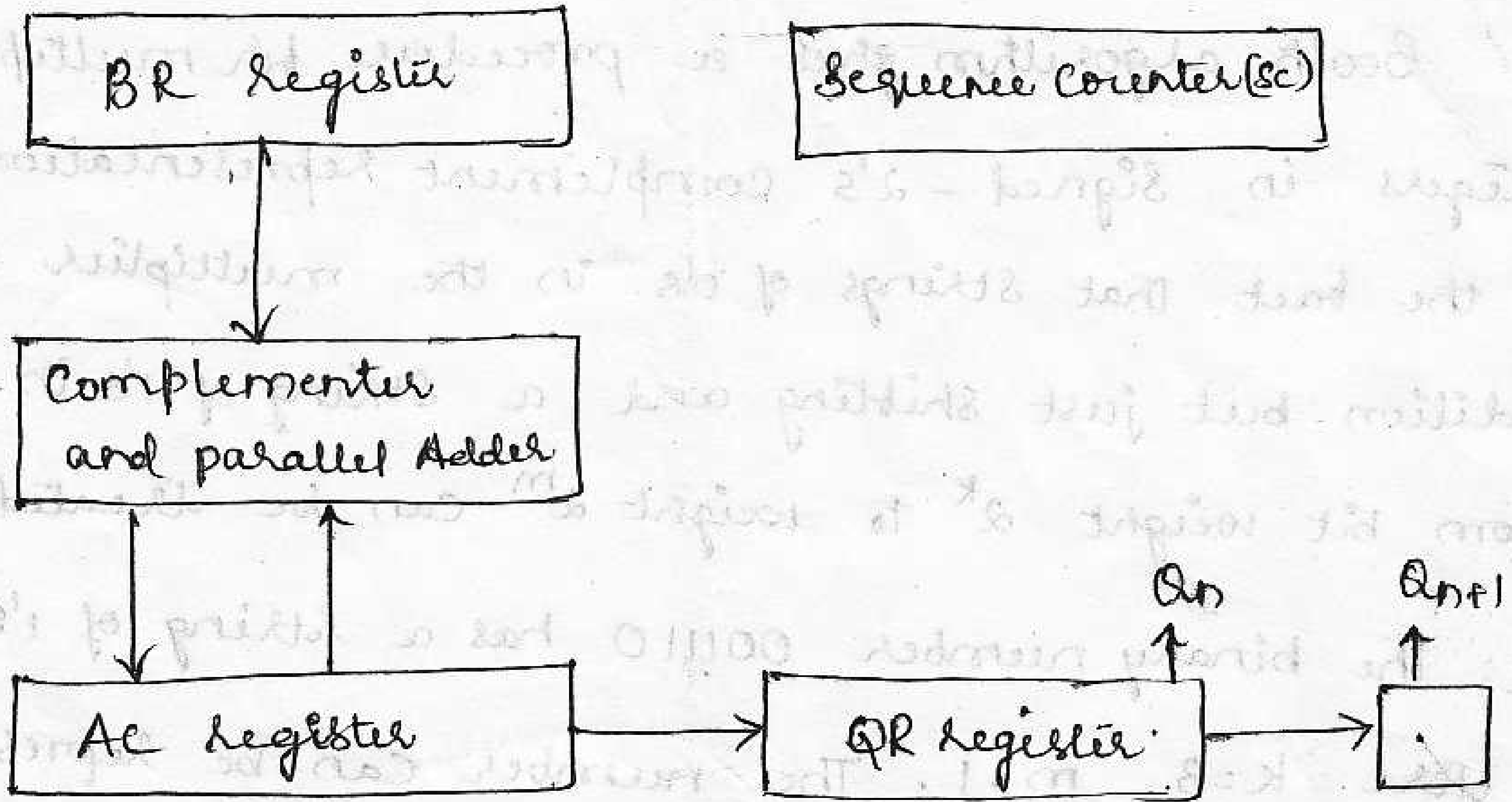
1) The multiplicand is subtracted from the partial product upon encountering the first least significant 1 in a string of 1's in the multiplier.

2) The multiplicand is added to the partial product upon encountering the first 0 in a string of 0's in the multiplier.

3) The partial product doesn't change when the multiplier bit is identical to the previous multiplier bit.

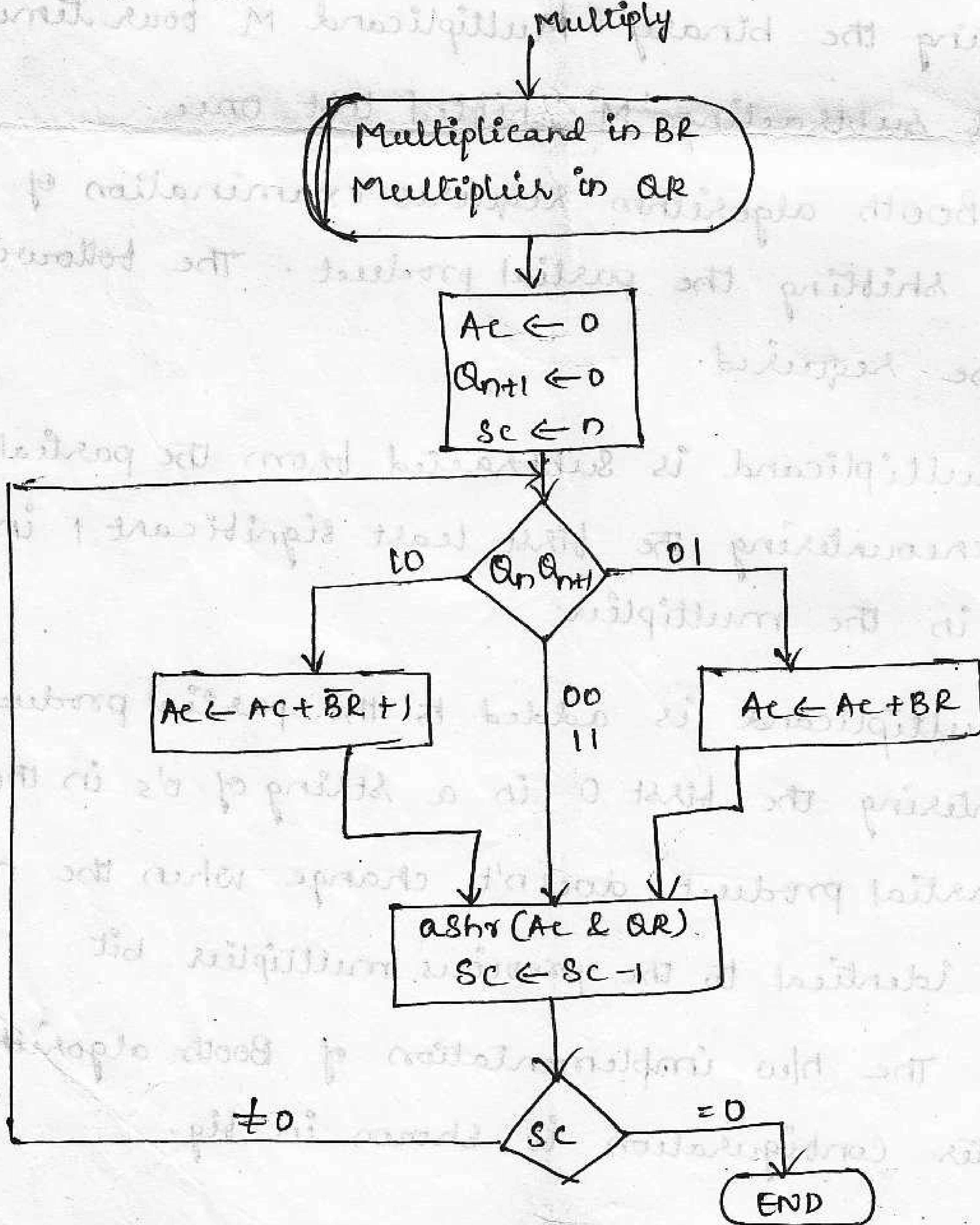
The h/w implementation of Booth algorithm requires the register configuration is shown in fig.

www.jntuworldupdates.org



Here an extra flip-flop Q_{n+1} is appended to QR to facilitate a double bit inspection of the multiplier.

The flow chart for Booth algorithm is shown in fig.



Eg: (−9) × (−13) = +117.

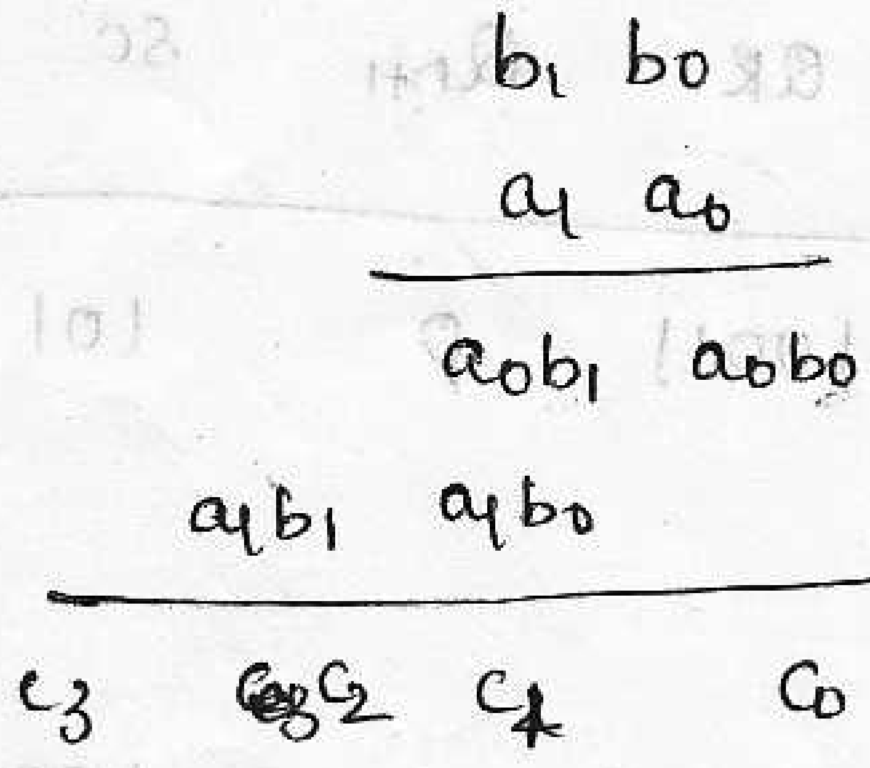
Q_n	Q_{n+1}	BR = 10111, $\overline{BR}$+1 = 01001	AC	QR	Q_{n+1}	SC
1	0	Initial	00000	10011	0	101
		Subtract	01001			
			01001			
		ashr	00100	11001	1	100
1	1	ashr	00010	01100	1	011
0	1	Add BR	10111			
			11001			
		ashr	11100	10110	0	010
0	0	ashr	01110	01011	0	001
1	0	Subtract BR	01001			
			10111			
		ashr	01011	10101	1	000

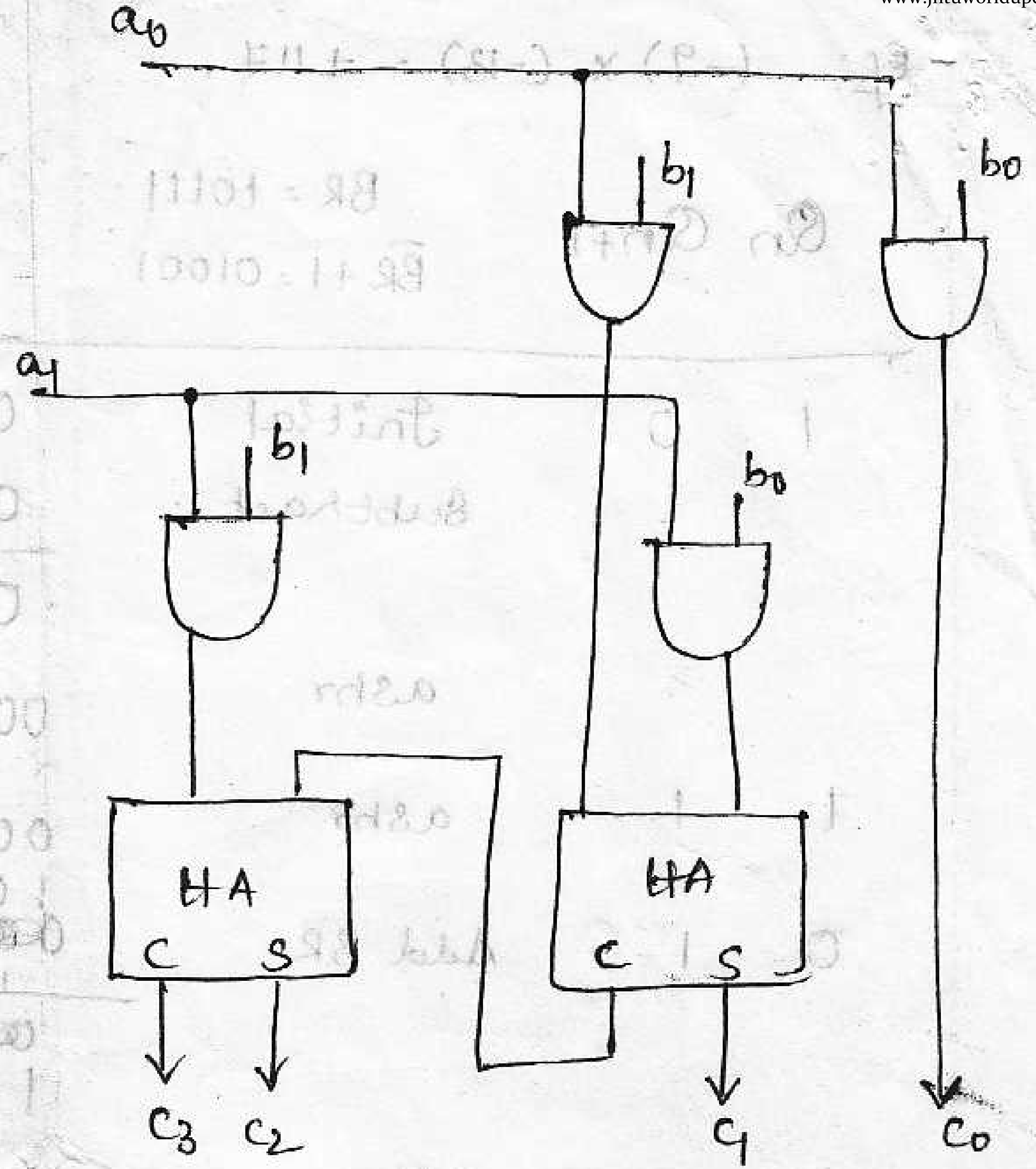
AC QR = 0101110101

Array Multiplier :-

Checking the bits of the multiplier one at a time and forming partial products is a sequential operation that requires a sequence of add and shift micro operations. The multiplication of two binary numbers can be done with one micro operation by means of a combinational circuits that forms the product bit all at once. This is a fast way of multiplying two numbers.

An array multiplier can be implemented with a combinational circuit as shown in fig.





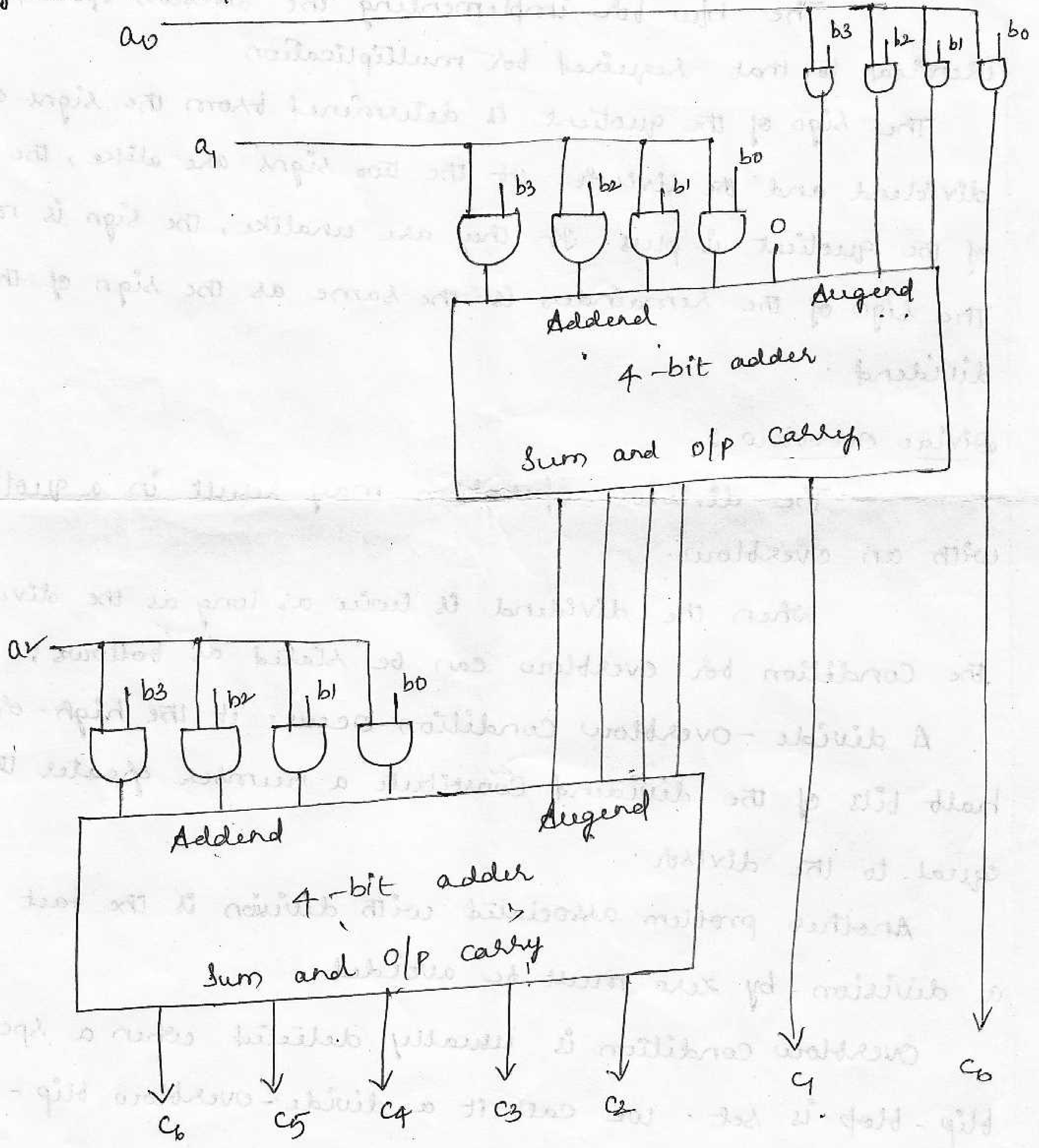
The multiplicand bits are b_1 & b_0, the multiplier bits a_1 & a_0 and the product is $c_3\, c_2\, c_1\, c_0$. The first partial product is formed by multiplying a_0 by $b_1 b_0$. The multiplication of two bits a_0 & b_0 produces '1' if both bits are 1, otherwise, it produces 0. This is identical to 'AND' operation.

The second partial product is formed by multiplying a_1 by $b_1 b_0$ and is shifted one position to the left. The two partial products are added with two half adders (HA) circuits. It is necessary of half adders because to produce sum.

A combinational circuit binary multiplier with more bits can be constructed in a similar fashion. The binary output in each level of AND gates is added parallel with the partial product of the previous level to form a new partial product.

For j multiplier bits & 'k' multiplicand bits we need $j \times k$ AND gates and $(j-1)$ k-bit adders to produce a product of $j+k$ bits.

As a second example, consider a multiplier circuit that multiplies a binary number of four bits with a number of three bits. The multiplicand is represented by $b_3 b_2 b_1 b_0$ and the multiplier by $a_2 a_1 a_0$ since $K = 4$ & $j = 3$. The logic diagram of the multiplier is shown below.



Division Algorithms :-

Division of two fixed - point binary numbers in signed - Magnitude representation.

H/w implementation for signed - magnitude data :-

The h/w for implementing the division operation is identical to that required for multiplication

The sign of the quotient is determined from the signs of the dividend and the divisor. If the two signs are alike, the sign of the quotient is plus. If they are unalike, the sign is minus. The sign of the remainder is the same as the sign of the dividend.

Divide overflow :-

The division operation may result in a quotient with an overflow.

When the dividend is twice as long as the divisor, the condition for overflow can be stated as follows:

A divide - overflow condition occurs if the high - order half bits of the dividend constitute a number greater than or equal to the divisor.

Another problem associated with division is the fact that a division by zero must be avoided.

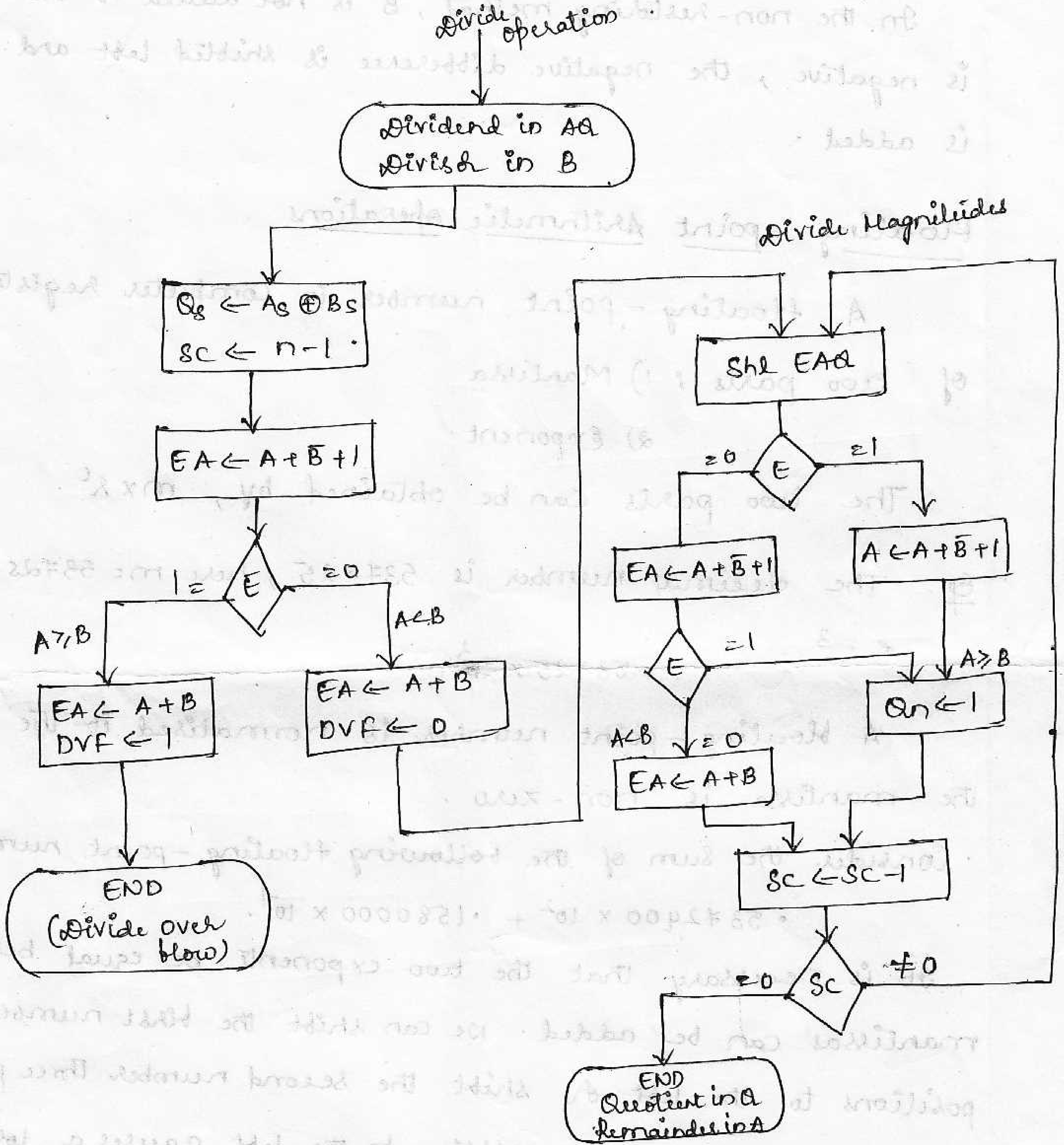
Overflow condition is usually detected when a special flip - flop is set. We call it a divide - overflow flip - flop and label it DVF.

The best way to avoid a divide overflow is to use floating - point data.

www.jntuworldupdates.org

Hardware Algorithm :-

The h/w divide algorithm is shown in the flowchart.



The sign of the result is transferred to Q_s to be part of the quotient.

Other Algorithms

The h/w method which described is called the restoring method. The two other methods are the comparison method and the non restoring method.

www.specworld.in www.smartzworld.com

When two numbers are added, the sum may contain an overflow digit. An overflow can be corrected by shifting the sum once to the right & incrementing the exponent.

We can subtract two numbers in the following way.

$$\begin{array}{r} .56780 \times 10^5 \\ - .56430 \times 10^5 \\ \hline .00350 \times 10^5 \end{array}$$

A floating-point number that has a '0' in the MSB of the Mantissa is said to have an underflow.

Floating-point multiplication and division don't require an alignment of the mantissas. The product can be formed by multiplying the two mantissas & adding the exponents. Division is accomplished by dividing the mantissa and subtracting the exponents.

The exponent may be represented in any one of three representations: Signed-Magnitude, Signed-2's complement, or Signed-1's complement.

A fourth representation is a biased exponent. The sign bit is removed from being a separate entity. The bias is a positive number that is added to each exponent as the floating-point number is formed, so that internally all exponents are positive.

for eg, exponent that ranges from −50 to 49. Internally, it is represented by two digits (without a sign) by adding to it a bias of 50. The exponent register contains the number e+50 where 'e' is the actual exponent.

The advantage of biased exponents is that they contain only positive numbers. Another advantage is that the smallest possible biased exponent contains all zero.

Addition & Subtraction :-

The algorithm can be divided into four parts

1) check for zeros
2) Align the mantissas
3) Add or subtract the mantissas
4) Normalize the result.

Multiplication :-

The multiplication algorithm can be divided into four parts

1) check for zeros
2) Add the exponents
3) multiply the mantissas
4) Normalize the product.

Division :-

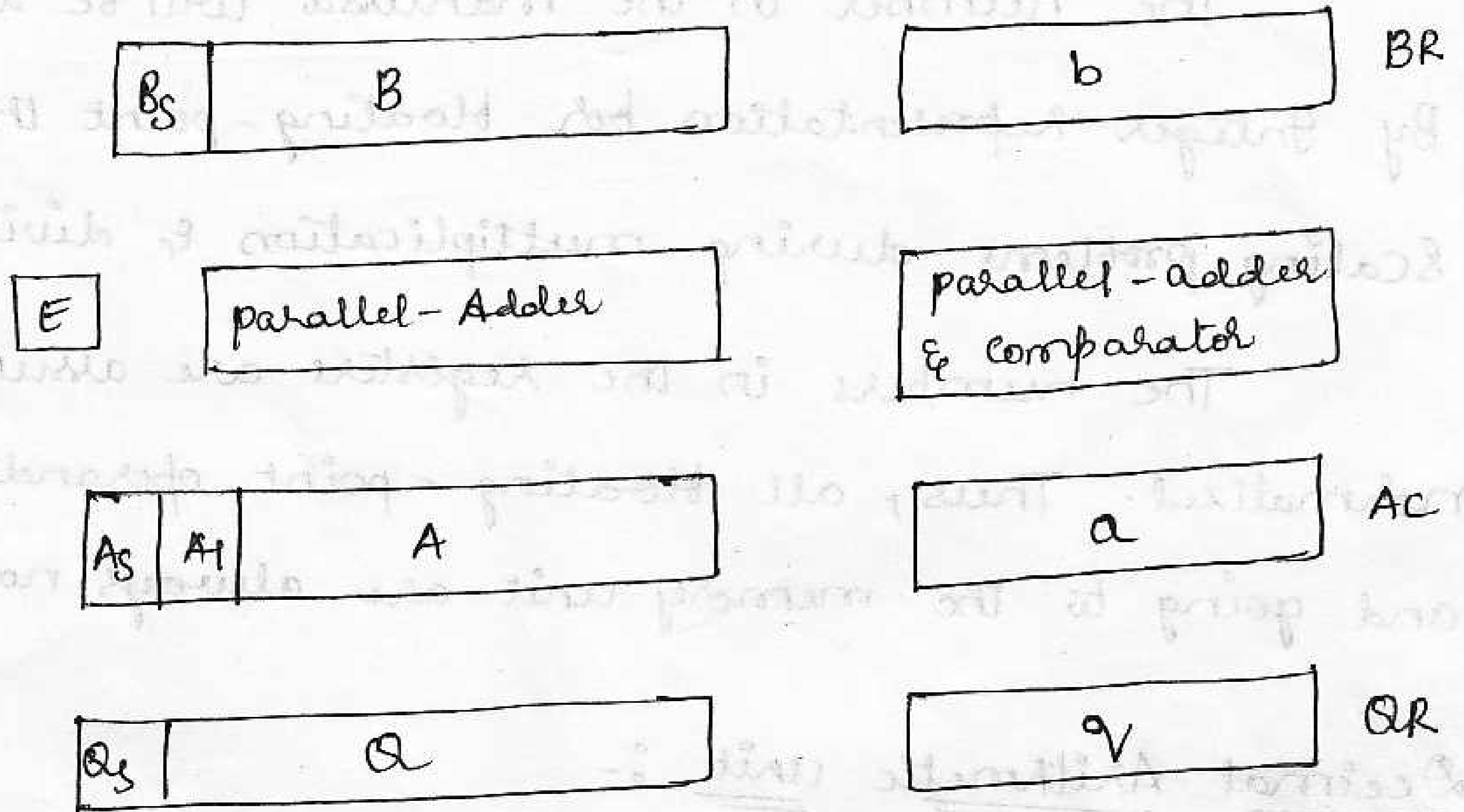
The division algorithm can be divided into five parts:

1) check for zeros
2) Initialize the registers & evaluate the sign.
3) Align the dividend
4) Subtract the exponents.
5) Divide the mantissas.

www.jntuworldupdates.org

Register configuration :-

The register configuration for floating-point operations as shown in fig.



There are three registers BR, AC & QR. Each register is subdivided into two parts. The mantissa part & the exponent part.

The Mantissa part has the same uppercase letter symbols as in fixed-point representation. The exponent part uses the corresponding lower case letter symbol. Each floating-point number has a mantissa in signed-Magnitude representation and a biased exponent. Thus AC has mantissa whose sign is in As and a magnitude is in A. The exponent is represented in 'a'. The diagram shows the MSB of A, is labeled by A1 whose value must be '1' to get normalised. In the same way, Register BR is subdivided into Bs, b & B and QR into Qs, Q, q.

A parallel-adder adds the two mantissas and transfers the sum into A and carry to 'E'. A separate parallel-adder is used for the exponents. If this, they don't have a distinct sign bit, but are represented as a biased positive quantity.

www.specworld.in www.smartzworld.com

www.jntuworldupdates.org

The exponents are also connected to a magnitude compara-tor that provides three binary o/p's to indicate relative magnitude.

The number in the mantissa will be taken as a fraction By Integer representation for floating-point they may cause scaling problems during multiplication & division.

The numbers in the registers are assumed to be initially normalized. Thus, all floating - point operands coming from and going to the memory unit are always normalized.

Decimal Arithmetic unit :-

To perform arithmetic operations with decimal data, it is necessary to convert the i/p decimal numbers to binary, it is ~~necessary~~ perform all calculations with binary numbers and to convert the results into decimal. It is very efficient Method. The decimal numbers are then applied to a decimal arithmetic unit to executing decimal arithmetic microoperations.

Many computers have h/w for arithmetic calculations with both binary & decimal data.

A decimal arithmetic unit is a digital function that performs decimal micro operations. A single-stage decimal arithmetic unit consists of nine binary i/p variables & five binary o/p variables, since a minimum of four ~~bits~~ bits is required to represent each coded decimal digit.

BCD Adder :-

In BCD, each i/p digit doesn't exceed 9, the o/p sum can't

be greater than 9+9+1 = 19, the '1' in the sum being an i/p carry.

For eg, we apply two BCD digits to a 4-bit binary adder. The adder will form the sum in binary & produce a result that may range from 0 to 19. These binary numbers are shown in below table and are labeled by symbols K, Z_8, Z_4, Z_2 & Z_1, K is the carry. The first column in the table lists the binary sums as they appear in the o/p of a 4-bit binary adders. The o/p sum of two decimal numbers must be represented in BCD. In the second column, the values of the first value can be converted to the correct BCD digit.

Binary sum					BCD sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

www.jntuworldupdates.org

www.jntuworldupdates.org

In the table, when the binary sum is equal to or less than 1001, the corresponding BCD number is identical, and there is no conversion is needed. When the binary is greater than 1001, than nonvalid BCD representation is obtained.

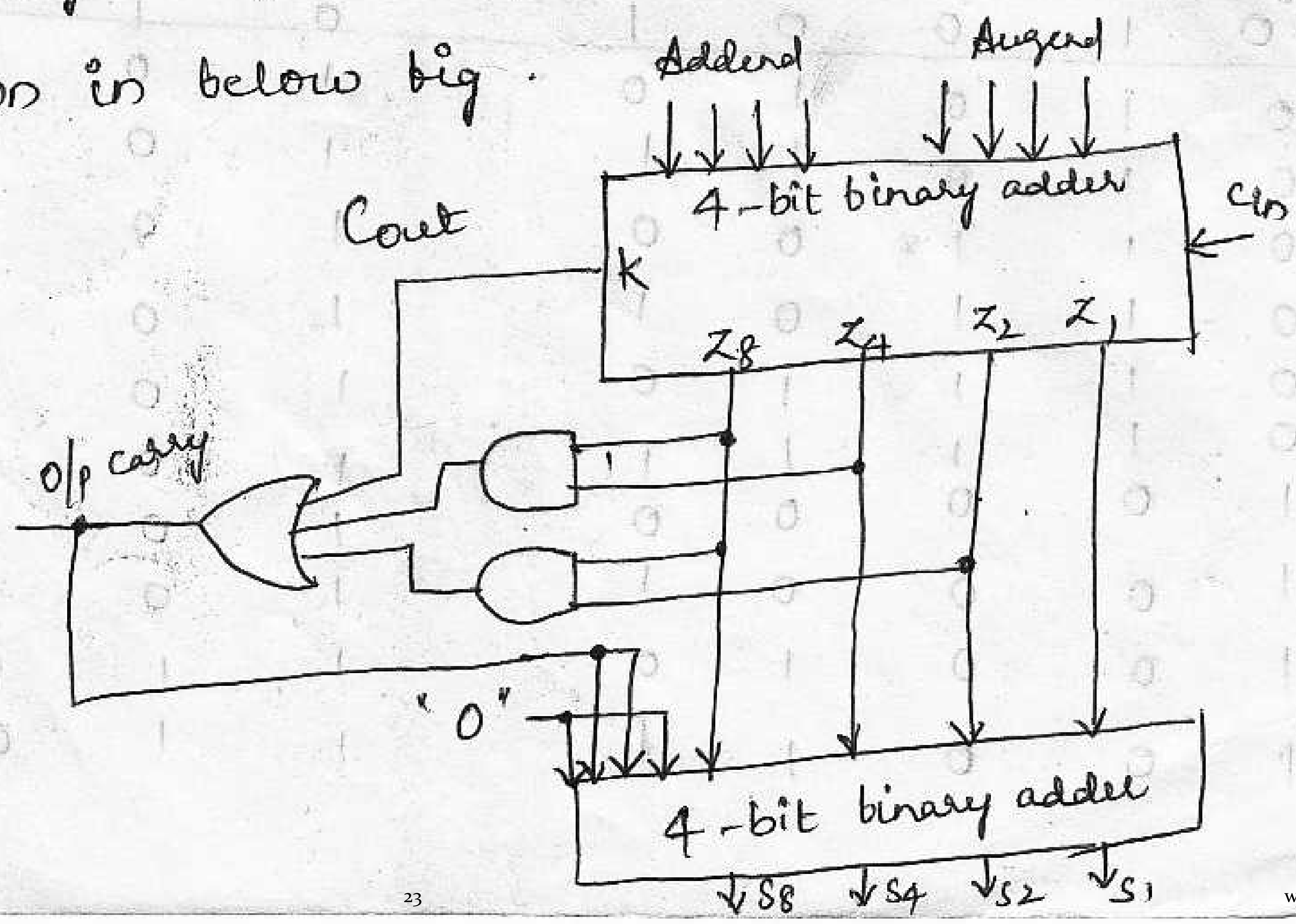
One method of adding decimal numbers in BCD would be to employ one 4 bit binary adder & perform the arithmetic operation one digit at a time. If the result is greater than or equal to 1010, it corrected by adding 0110 to the binary sum. The second operation will automatically produce an o/p carry for the next pair of significant digits. The procedure is repeated until all decimal digits are added.

The condition for a correction and an o/p - carry can be expressed by the Boolean function.

$$C = K + Z_8 Z_4 + Z_8 Z_2 .$$

When C=1, it is necessary to add 0110 to the binary sum and provide an output - carry for the next stage.

A BCD adder is a circuit that adds two BCD digits in parallel & produces a sum digit also in BCD. To add 0110 to the binary sum, we use a second 4-bit binary adder as shown in below fig.



www.specworld.in

www.smartzworld.com

www.jntuworldupdates.org

The two decimal digits, together with the input-carry are first added in the top 4-bit binary adder to produce the binary sum. When it is equal to 1, binary 0110 is added to the binary sum through the bottom 4-bit binary adder. The o/p - carry generated from the bottom binary adder may be ignored, since it supplies information already available in the o/p - carry terminal.

A decimal parallel-adder that adds n decimal digits needs n BCD adder stages with the output-carry from one stage connected to the input-carry of the next-higher order stage.

BCD Subtraction :-

BCD is not a self-complementing code, the 9's complement cannot be obtained by complementing each bit in the code.

The 9's complement of a decimal digit represented in BCD may be obtained by complementing the bits in the coded representation of the digit provided a correction is included.

There are two possible correction methods.

1) binary 1010 (decimal 10) is added to each complemented digit and the carry discarded after each addition
2) binary 0110 (decimal 6) is added before the digit is complemented.

Eg: 1) 0111

Complement

= 1000
1010 (add)
0010

2) 0111

0111 (add)
0110
1101

0010 (complement)

Complementing each bit of a 4-bit binary number N is identical to the subtraction of the number from 1111 (decimal 15)

1) Adding the binary equivalent of decimal 10 gives

$$15 - N + 10 = 9 - N + 16$$

So, here 16 signifies the carry is discarded. So, the result is 9 - N

2) Adding the binary equivalent of decimal 6 gives

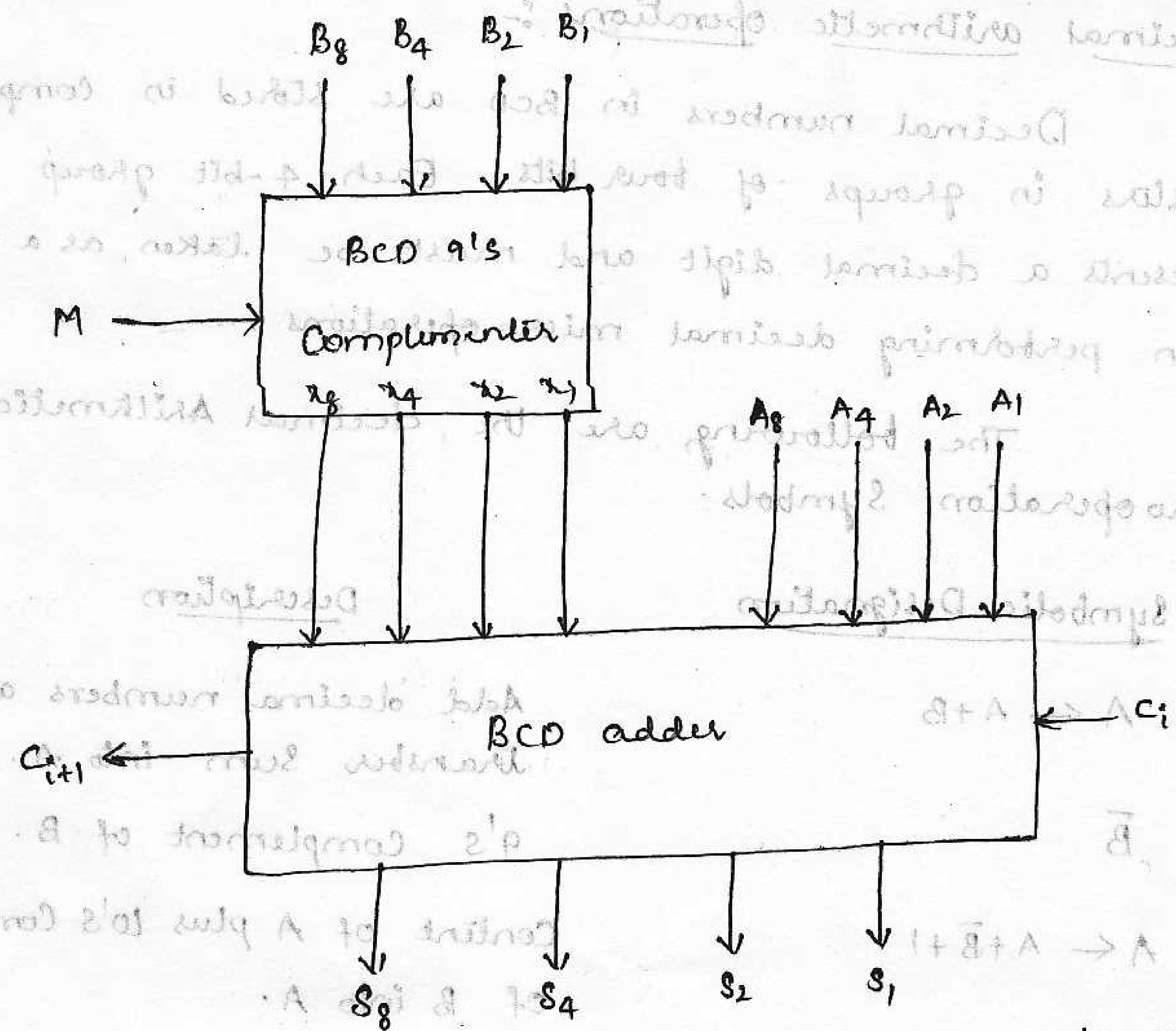
$$15 - (N+6) = 9 - N.$$

The 9's complement of a BCD digit can also be obtained through a combinational circuit. When this circuit is attached to a BCD adder, the result is BCD adder / subtractor.

Let the subtrahend (or addend) digit be denoted by the four binary variables B_8, B_4, B_2, B_1. Let M be a mode bit that controls the add / subtract operation.

Let the binary variables x_8, x_4, x_2, x_1 be the o/p's of the 9's complementer circuit. The below fig shows the one stage of a decimal arithmetic unit.

www.jntuworldupdates.org



It consists of a BCD adder and a 9's Complementer. The mode 'M' controls the operation of the unit. With M=0, the 'S' o/p's form the sum of A and B. With M=1, the 'S' o/p's form the sum of A plus the 9's complement of B.

The o/p carry c_{i+1} from one stage must be connected to the i/p carry c_i of the next higher-order stage.

The way to subtract the two decimal numbers is to let M=1 and apply a '1' to the i/p carry c_1 of the first stage.

The o/p's will form the sum of A plus the 10's Complement of B which is equivalent to a subtraction operation if the carry-out of the last stage is discarded.

www.jntuworldupdates.org

Decimal arithmetic operations :-

Decimal numbers in BCD are stored in computer registers in groups of four bits. Each 4-bit group represents a decimal digit and must be taken as a unit when performing decimal micro operations.

The following are the decimal Arithmetic microoperation Symbols.

Symbolic Designation	Description
$A \leftarrow A+B$	Add decimal numbers and transfer sum into A.
$\bar{B}$	9's complement of B.
$A \leftarrow A+\bar{B}+1$	Content of A plus 10's Complement of B into A.
$Q_L \leftarrow Q_L+1$	Increment BCD number in Q_L
dshr A	Decimal shift-right register A
dshl A	Decimal shift-left register A

Incrementing / decrementing a register is the same for binary and decimal except that the binary counter goes through 16 states from 0000 to 1111, when incremented. The decimal counter goes through 10 states from 0000 to 1001 and back to 0000.

A decimal shift right or left is preceded by the letter 'd' to indicate a shift over the four bits that hold the decimal digits.

Eg: 3 4 2 1

0011 0100 0010 0001

dshr 0000 0011 0100 0010

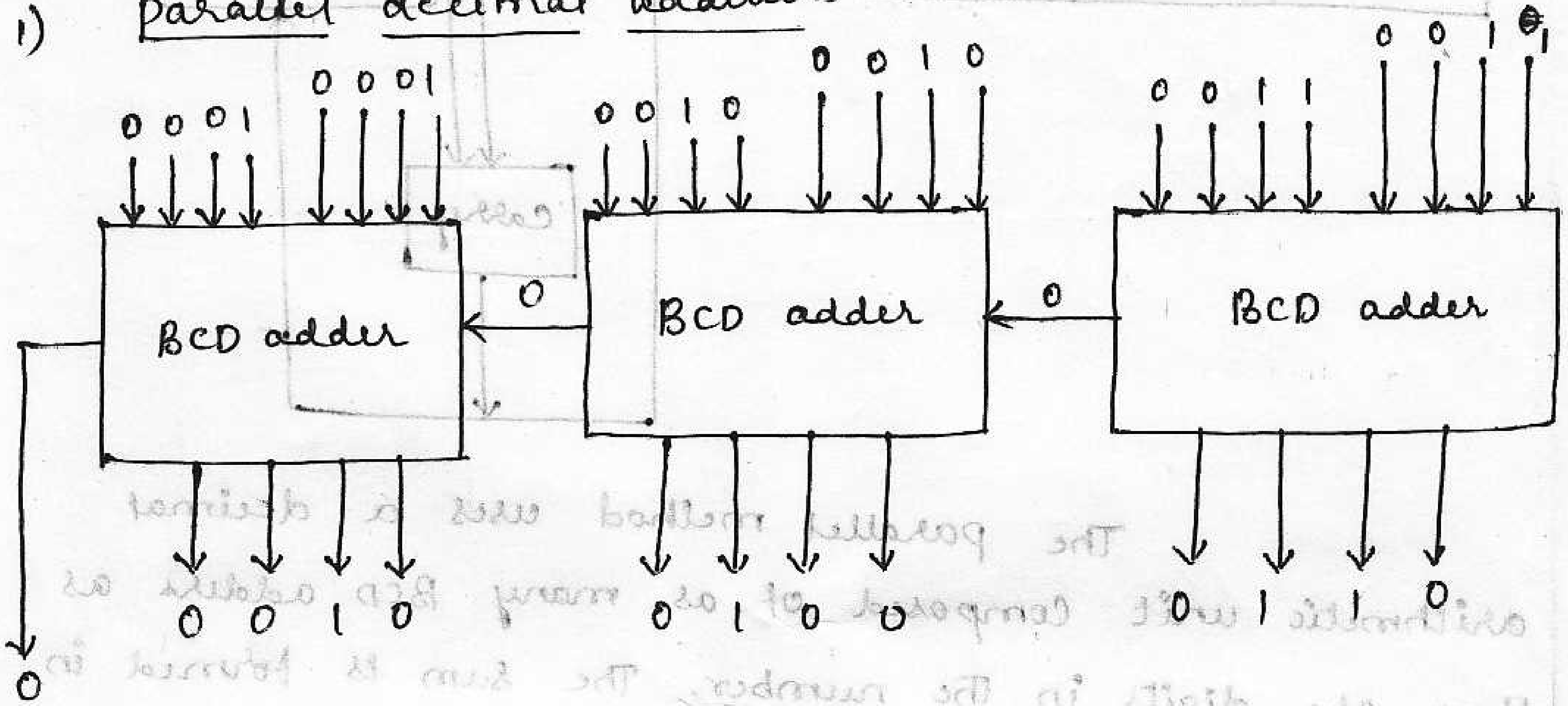
www.jntuworldupdates.org

Addition and Subtraction :-

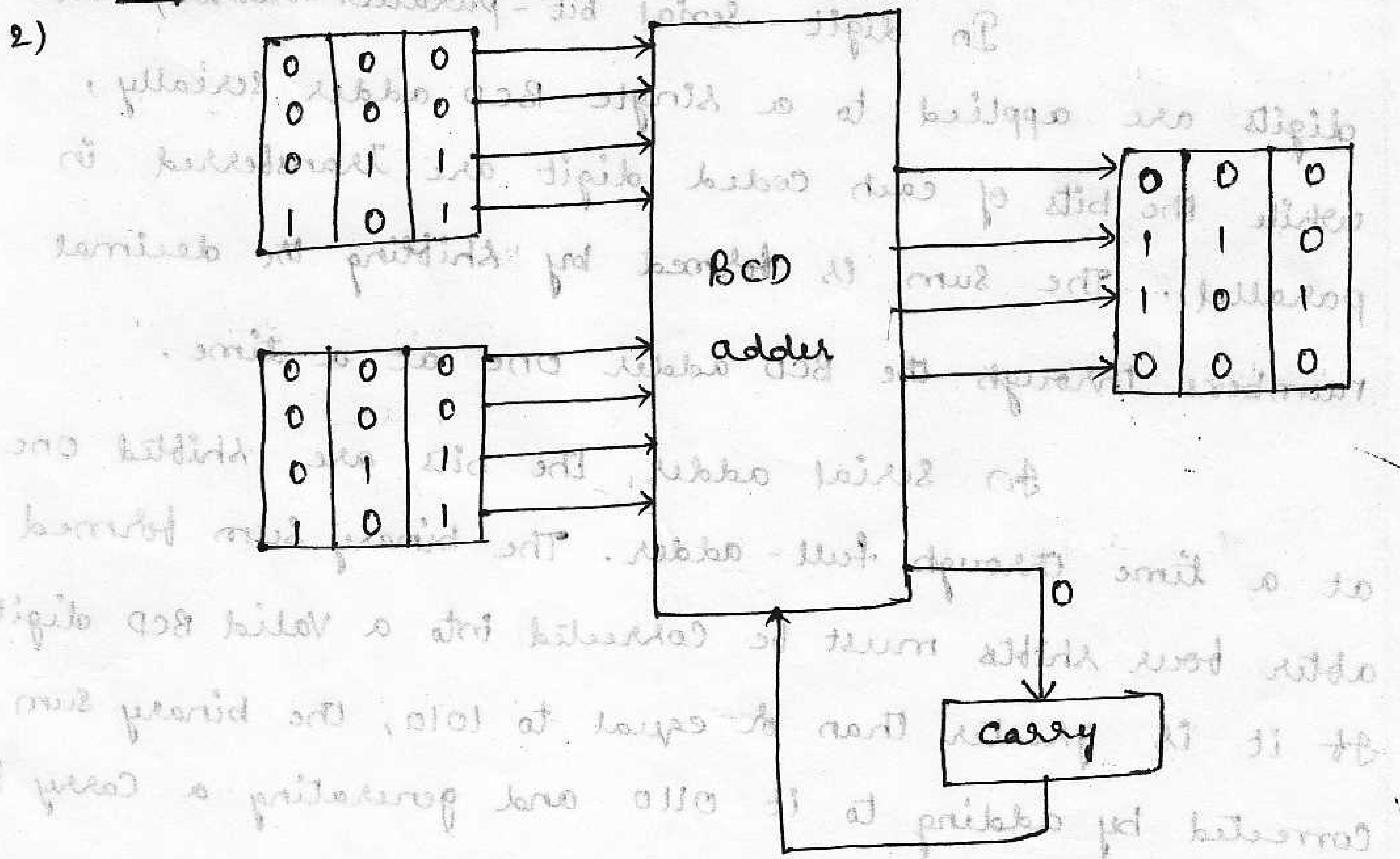
A decimal data can be added in three different ways.

Eg: 123 + 123 = 246

1) parallel decimal addition



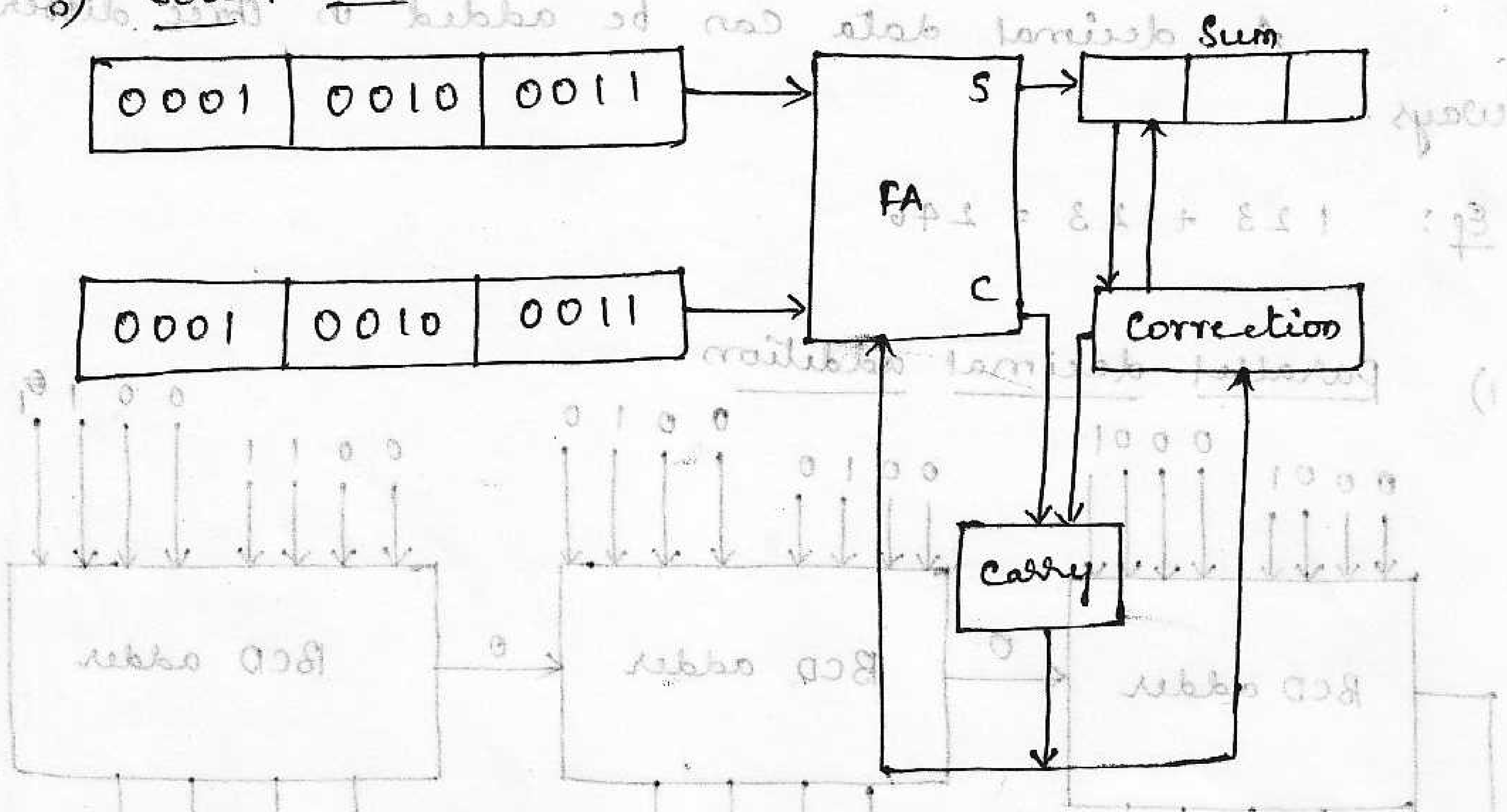
2) Digit - Serial, bit parallel decimal addition



www.specworld.in

www.smartzworld.com

3) Serial adder



The parallel method uses a decimal arithmetic unit composed of as many BCD adders as there are digits in the number. The sum is formed in parallel and requires only one micro operation.

In digit-serial bit-parallel method, the digits are applied to a single BCD adder serially, while the bits of each coded digit are transferred in parallel. The sum is formed by shifting the decimal numbers through the BCD adder one at a time.

In serial adder, the bits are shifted one at a time through full-adder. The binary sum formed after four shifts must be corrected into a valid BCD digit. If it is greater than or equal to 1010, the binary sum is corrected by adding to it 0110 and generating a carry for the next pair of digits.

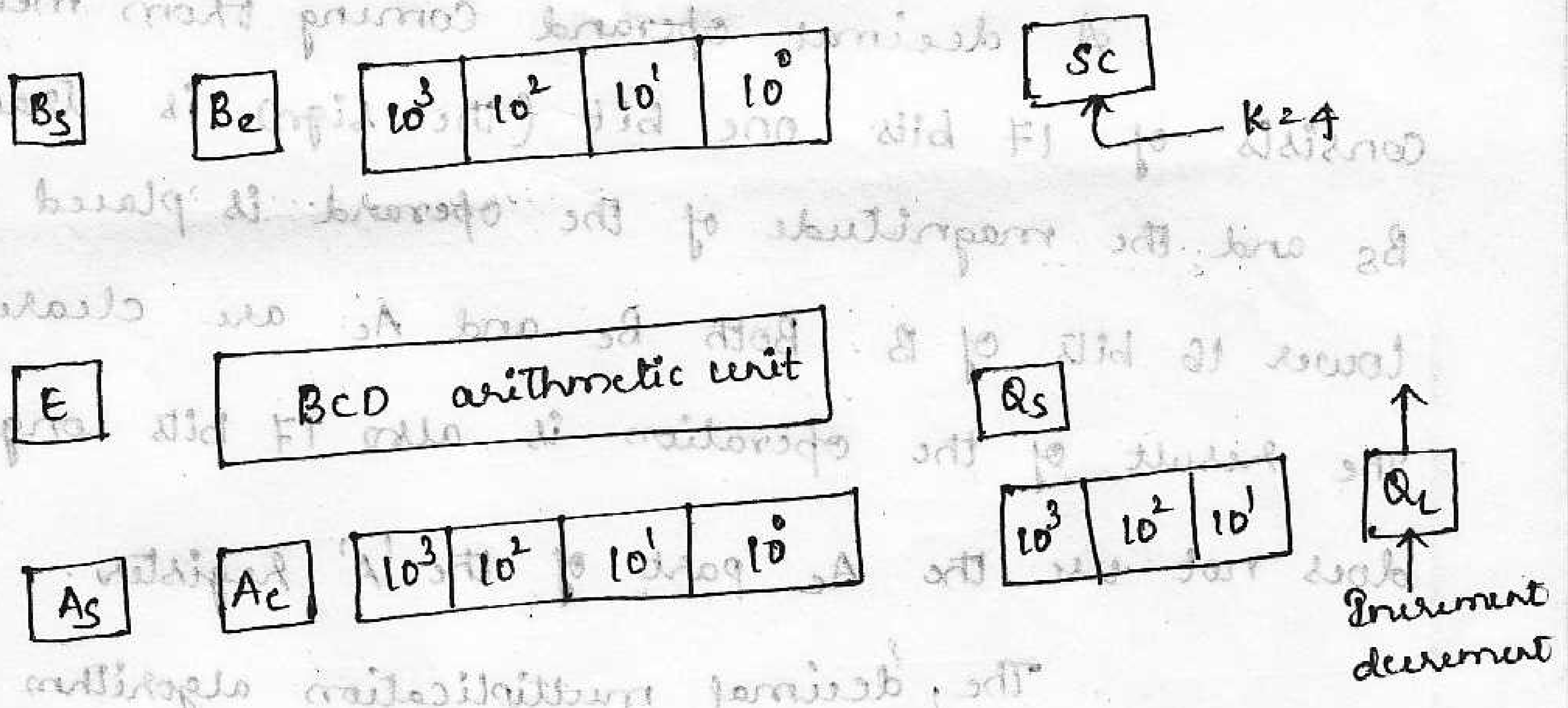
The parallel method is fast but requires a large number of adders.

The digit-serial bit-parallel method requires only one BCD adder, which is shared by all the digits. It is slower than the parallel method because of the time required to shift the digits.

The serial method requires a minimum amount of equipment but is very slow.

Multiplication :-

The registers configuration for the decimal multiplication is shown in fig.



Assuming here four-digit numbers, with each digit occupying four bits, for a total of 16 bits for each number. There are three registers A, B & Q each having a corresponding sign flip-flop A_s, B_s & Q_s.

Registers A and B have four more bits designated by A_e & B_e, that provide an extension of one more digit to the registers.

www.jntuworldupdates.org

The BCD arithmetic unit adds the five digits in parallel and places the sum in the five-digit A register. The end carry goes to the flip flop 'E'.

The purpose of digit A_e is to accommodate an overflow while adding the multiplicand to the partial product during multiplication.

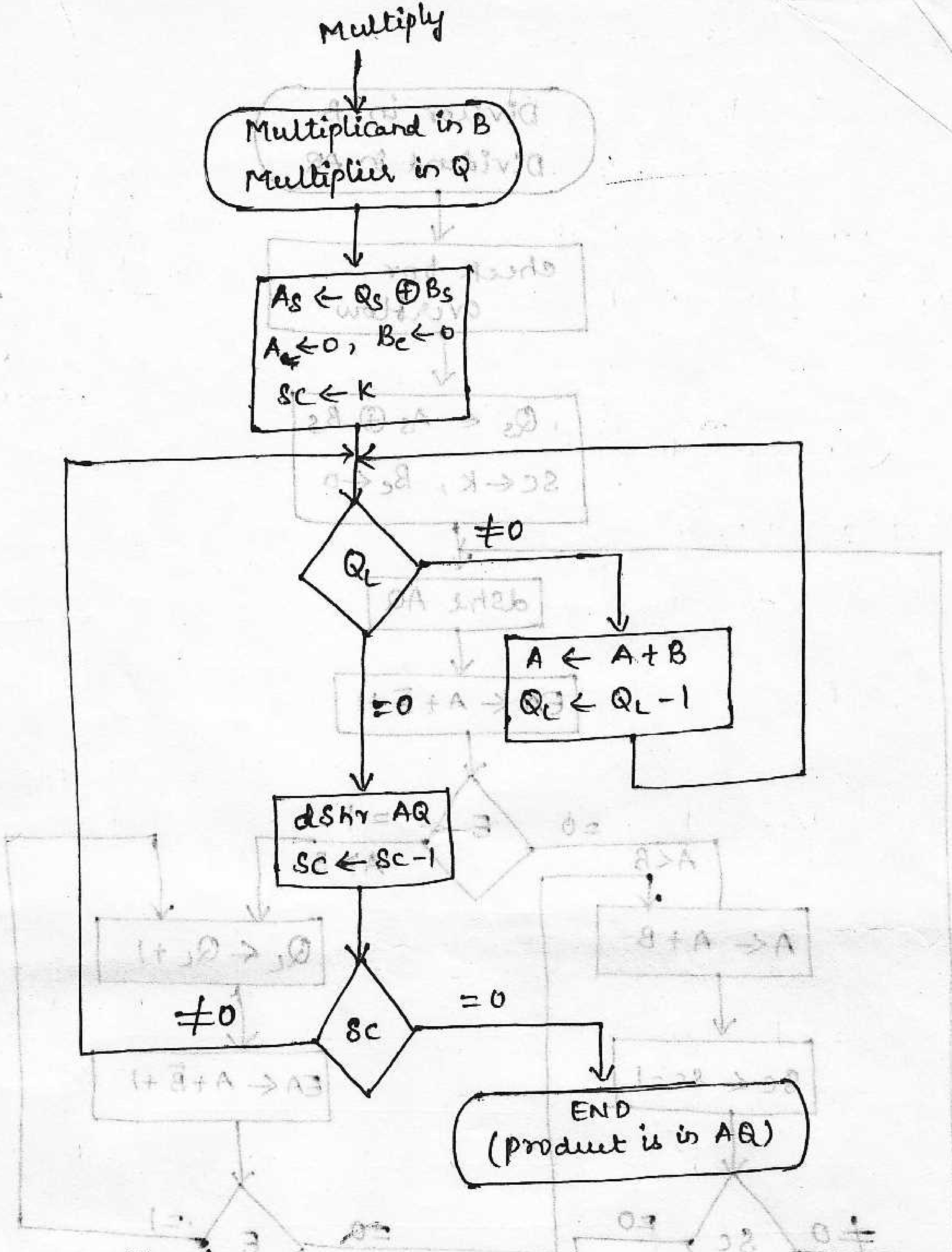
The purpose of digit B_e is to form the 9's complement of the divisor when subtracted from the partial remainder during the division operation.

The least significant digit in register Q is denoted by Q_L. This digit can be incremented or decremented.

A decimal operand coming from memory consists of 17 bits one bit (the sign) is transferred to B_s and the magnitude of the operand is placed in the lower 16 bits of B. Both B_e and A_e are cleared initially. The result of the operation is also 17 bits long and does not use the A_e part of the 'A' register.

The decimal multiplication algorithm is shown below.

www.jntuworldupdates.org



Division algorithm :-

In the restoring division method, the divisior is subtracted from the dividend or partial remainder as many times as necessary until a negative remainder results. The correct remainder is then restored by adding the divisior. The digit in the quotient reflects the no. of subtractions, upto but excluding the one that caused the negative difference.

The decimal division algorithm is shown below.

Divisor in B
Dividend in AQ

check for overflow

$Q_s \leftarrow A_s \oplus B_s$
$SC \leftarrow K$, $B_c \leftarrow 0$

dshl AQ

$EA \leftarrow A + \bar{B} + 1$

E
=0 : A<B
=1 : A≥B

$A \leftarrow A + B$

$SC \leftarrow SC - 1$

SC
≠0
=0

$Q_L \leftarrow Q_L + 1$

$EA \leftarrow A + \bar{B} + 1$

E
=0
=1

END
Quotient is in Q
Remainder is in A

UNIT - VI

Input - Output Organization

peripheral Devices :-

The input-output subsystem of a computer is commonly referred to as I/O which provides an efficient mode of communication between the central system and the outside environment.

Devices that are under the direct control of the computer are said to be connected on-line. These devices are designed to read information into or out of the memory unit upon command from the CPU and are considered to be part of the total computer system.

Input or output devices attached to a computer are also called 'peripherals'. There are three types of peripherals 1) input 2) output 3) Input-output. These peripherals may be analog or digital and serial or parallel.

The common peripherals are keyboards, display units and printers.

peripherals that provide auxiliary storage for the system are magnetic disks & tapes.

Monitor and Keyboard :-

The most commonly used peripheral is 'Video monitors'. They consist of a keyboard as the input device and a display unit as the O/P device.

There are different types of Video monitors, the most popular use is Cathode Ray Tube (CRT). The CRT contains

an electronic gun that sends an electronic beam to a phosphorescent screen in front of the tube. The beam can be deflected horizontally and vertically. To produce a pattern on the screen, a grid inside the CRT receives a variable voltage that causes the beam to hit the screen and make it glow at selected spots.

A characteristic feature of display devices is a cursor that marks the position in the screen where the next character will be inserted. The cursor can be moved to any position in the screen, to a single character, beginning of a word, ending of a word etc.

Edit keys add or delete information based on the cursor position. The display terminal can operate in a single character mode where all characters entered on the screen through the keyboard.

In the block mode, the edited text is first stored in a local memory inside the terminal. The text is transferred to the computer as a block of data.

printer :-

printers provide a permanent record on paper of computer output data or text.

There are three different types of character printers.

1) Daisy wheel printer 2) Dot matrix 3) Laser printer.

The daisy wheel printer contains a wheel with the characters placed along the circumference. To print a character, the wheel rotates to the proper position and a magnet then presses the letter against the ribbon.

The dot matrix printer contains a set of dots along the printing mechanism. Each dot can be printed or not, is depending on the specific characters that are printed on the line.

The laser printer uses a rotating photographic drum that is used to imprint the character images. The pattern is then transferred onto paper in the same manner as a copying machine.

Magnetic tape :-

Magnetic tapes are used mostly for storing files of data.

Magnetic disk :-

Magnetic disks have high-speed rotational surfaces coated with magnetic material.

Input - output Interface :-

I/o interface provides a method for transferring information between internal storage and external I/o devices.

The purpose of the communication link is to resolve the differences that exist between the central computer and each peripheral.

The major differences are

1) peripherals are electromechanical and electromagnetic devices and their manner of operation is different from the operation of the CPU and memory which are electronic devices. so, a conversion of signal values may be required.

2) The data transfer of peripherals is usually slower than the

www.jntuworldupdates.org

transfer rate of the CPU and a synchronization mechanism may be needed.

3) Data Codes and formats in peripherals differ from the word format in the CPU and memory.

4) The operating modes of peripherals are different from each other and each must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

To resolve these differences, computer systems include special h/w components between the CPU and peripherals to supervise and synchronize all i/p & o/p transfers. These components are called 'interface' units.

In general, the word "interface" is the point of contact between two parts of a system.

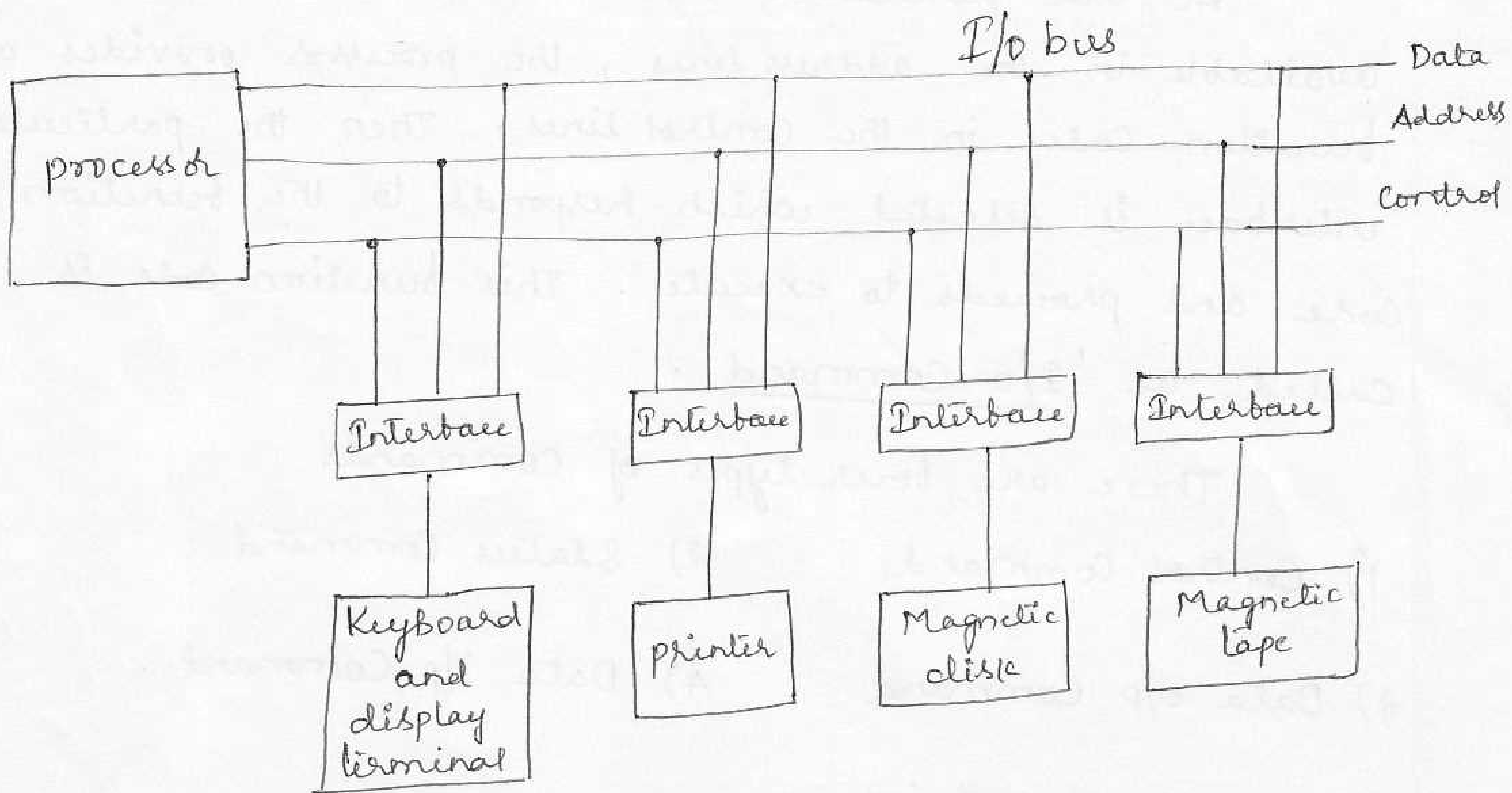
Two main types of interface are 1) CPU interface that corresponds to the system bus.
2) input-output interface that corresponds to the input output device.

I/O Bus and interface Modules :-

The communication link between the processor and several peripherals is shown in fig.

The I/O bus consists of data lines, address lines and control lines. The magnetic disk, printer and terminal are employed in any general-purpose computer.

Each peripheral device has associated with it an interface unit. Each interface decodes the address



and control received from the I/o bus, interprets them for the peripheral and provides signals for the peripheral controller. It also synchronizes the data flow and supervises the transfer between peripheral and processor. Each peripheral has its own controller that operates the particular electro-mechanical device.

The I/o bus from the processor is attached to all peripheral interfaces.

To communicate with a device, the processor places a device address on the address lines. Each interface is attached to the I/o bus contains an Address decoder that monitors the address lines. When the interface detects its own address, it activates the path between the bus lines and the device that it controls, remaining others are disabled by their interface.

www.jntuworldupdates.org

At that particular instance, the address is made available in the address lines, the processor provides a function code in the control lines, Then the particular interface is selected which responds to the function code and proceeds to execute. This function code is called as 'I/o Command'.

There are four types of commands

1) Control Command 2) Status Command

3) Data o/p Command 4) Data i/p Command.

Control Command :-

It is issued to activate the peripheral and to inform it what to do.

Status Command :-

It is used to test various status conditions in the interface and the peripheral.

Data o/p Command :-

It causes the interface to respond by transferring data from the bus into one of its registers.

Data i/p Command :-

The interface which receives an item of data from the peripheral and places it in its buffer register. The processor checks if data are available by means of a status command and then issues a data i/p command. Then the interface places the data on the data lines, where they are accepted by the processor.

www.specworld.in

www.smartzworld.com

www.jntuworldupdates.org

I/O Versus Memory Bus :-

The memory bus contains data lines, address lines and read/write control lines.

There are three ways that computer buses can be used to communicate with memory and I/O.

1) use two separate buses, one for memory and other for I/O.
2) use one common bus for memory and I/O with common control lines.
3) use one common bus for both memory and I/O but have separate control lines.

In method 1, the computer has independent sets of data, address and control buses i.e., one for accessing memory and the other for I/O. This method can be implemented, where we have separate I/O processor in addition to CPU.

The memory communicates with both the CPU and the IOP through a memory bus. The IOP also communicates with the input and the output devices through a separate I/O bus with its own address, data and control lines.

The purpose of the IOP is to provide an independent pathway for transfer of information between external devices and internal devices.

The IOP is sometimes called '<u>data channel</u>'.

www.specworld.in

www.smartzworld.com

www.jntuworldupdates.org

Isolated Versus Memory - Mapped I/O :-

The I/O read and I/O write control lines are enabled during an I/O transfer. The memory read and memory write control lines are enabled during an memory transfer. This configuration isolates all I/O interface addresses from the addresses assigned to memory and is refered to as the 'isolated I/O' for assigning addresses in a common bus.

In the isolated I/O configuration, the CPU has distinct i/p and o/p instructions and each of these instructions is associated with address of an interface register.

When the CPU fetches and decodes the operation code of an i/p or o/p instruction, it places the address associated with the instruction into the common address lines. At mean time, it enables the I/O read or I/O write control line. This informs the external components that are attached to the common bus that the address in the address lines is for an interface register and not for a memory word.

When the CPU is fetching an instruction or an operand from memory, it places the memory address on the address lines and enables the memory read or memory write control line. This informs the external components that the address is for a memory word and not for an I/O interface register.

The Second alternative, has same address space for both memory and I/O. i.e., it employs only one set of read and write signals and not distinguish between memory and I/O addresses. This configuration is referred to as memory-mapped I/O.

The assigned addresses for interface registers cannot be used for memory words, which reduce the memory address range available.

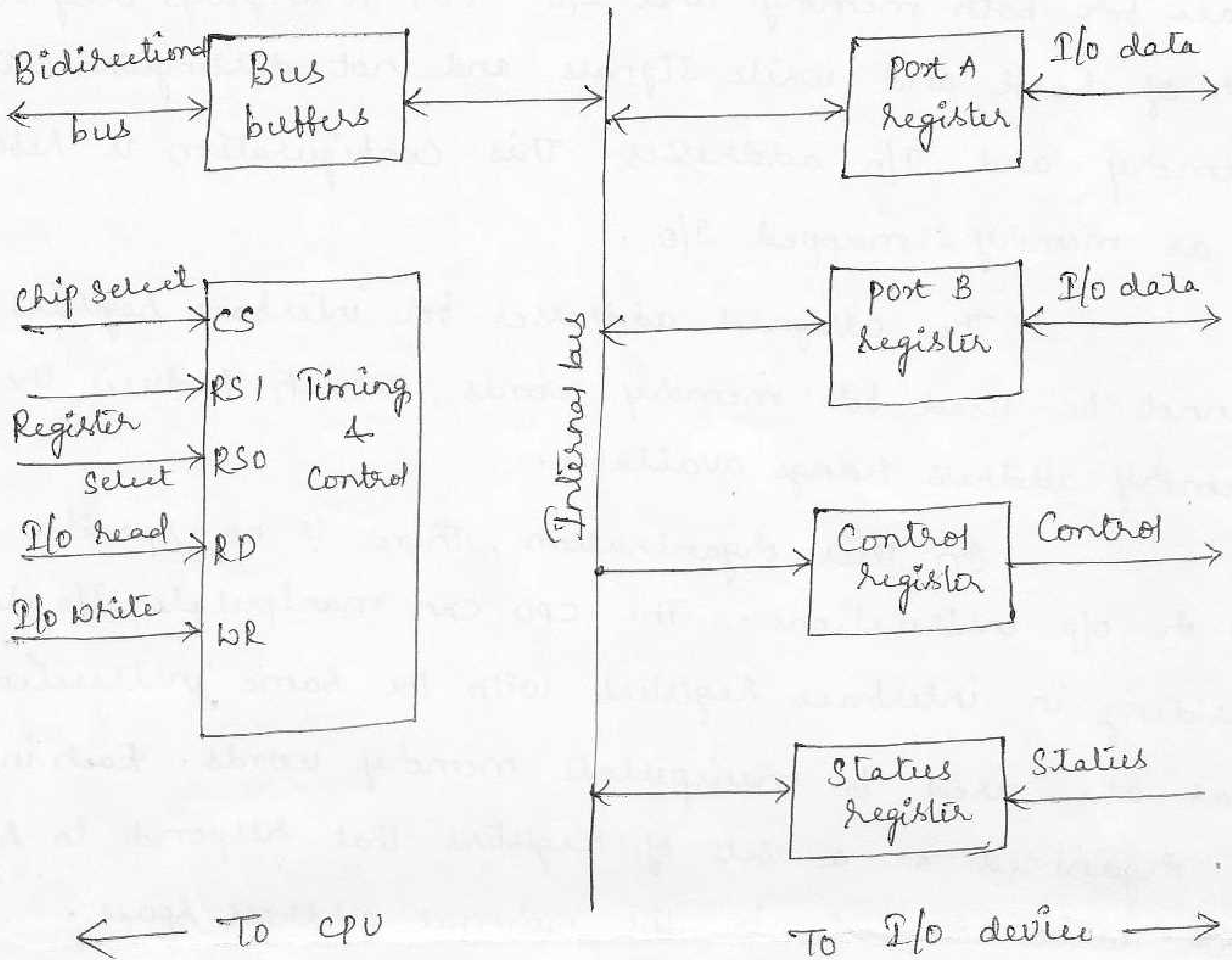
In this organization, there is no specific i/p or o/p instructions. The CPU can manipulate I/O data residing in interface registers with the same instructions that are used to manipulate memory words. Each interface is organized as a set of registers that respond to read and write requests in the normal address space.

The computers who implement this organization can use memory-type instructions to access I/O data. It allows the computer to use the same instructions for either i-o transfers or for memory transfers.

The advantage is that the load and store instructions used for reading and writing from memory, can be used as input and output data from I/O registers.

Example of I/O interface :-

An example of an I/O interface unit is shown in the fig.



CS	RS1	RS0	Register Selected
0	X	X	None : data bus in high impedance
1	0	0	port A register
1	0	1	port B register
1	1	0	Control register
1	1	1	Status register

The example Consists of two data registers called 'ports', a Control register, a Status register, bus buffers and timing and control circuits.

The interface Communicates with the CPU through the databus. The chip Select and register Select inputs determine the address assigned to the interface. The I/o read and write are two control lines that specify an i/p or o/p respectively.

The four registers Communicate directly with the I/o device attached to the interface.

The I/o data to and from the device can be transferred into either port A or port B. The interface may operate with an o/p device or with an i/p device or with a device that requires both i/p and o/p.

If the interface is Connected to a printer, it will only o/p data. But a magnetic disk unit transfers data in both directions, so the interface can use bidirectional lines.

A command is passed to the I/o device by sending a word to the appropriate interface register. In a system, the function code in the I/o bus is not needed because Control is sent to the Control register, Status information is received from the Status register and data are transferred to and from ports A and B registers. Thus the transfer of data, control and Status information is always through the common data bus.

The difference between data, control or

www.jntuworldupdates.org

Status information is determined from the particular interface register with which the CPU communicates.

The control register receives control information from the CPU. By loading appropriate bits into the control register, the interface and the I/O device attached to it.

for eg, port A may be defined as i/p port and port B may be defined as o/p port. A magnetic tape unit may be instructed to rewind the tape or to start the tape moving in the forward direction.

The bits in the status register are used for status conditions and for recording errors that may occur during the data transfer.

for eg, a status bit may indicate that port A has received a new data item from the I/O device. Another bit indicates that, there is an error occured during data transfer.

The interface registers communicate with the CPU through the bidirectional data bus. The address bus selects the interface unit through the chip select and the two register select inputs.

A circuit must be provided externally, to detect the address assigned to the interface registers. This circuit enables the chip select (CS) i/p when the interface is selected by the address bus.

Asynchronous Data Transfer :-

Asynchronous data transfer between two independent units requires that control signals be transmitted between the communicating units to indicate the time at which data is being transmitted.

There are two different methods to achieve this.

1) Strobe 2) HandShaking

Strobe :-

The strobe pulse supplied, one of the units to indicate to the other units when the transfer has to occur.

Hand Shaking :-

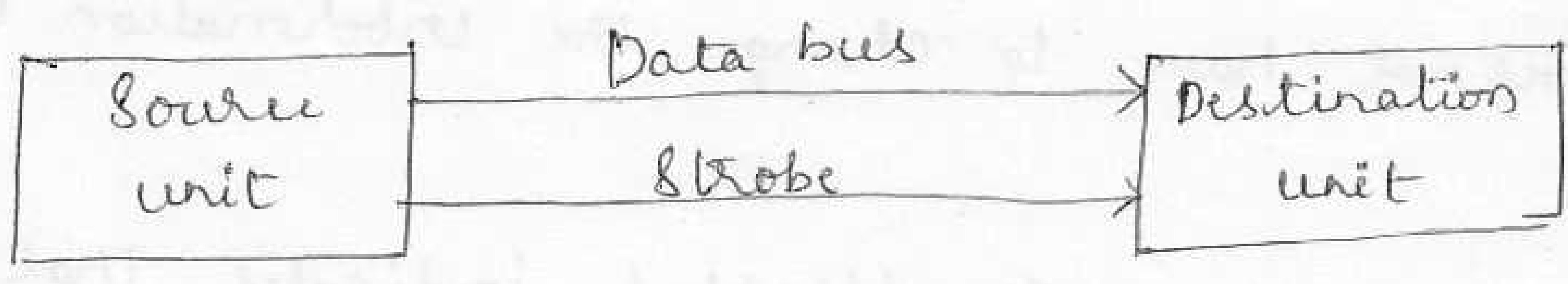
Each data item is being transferred with a control signal that indicates the presence of data in the bus. The unit receiving the data item responds with another control signal to acknowledge receipt of the data. This type of agreement between two independent units called 'Hand Shaking'.

Strobe Control :-

The Strobe control method of asynchronous data transfer employs a single control line to time each transfer. The strobe may be activated by either source or destination.

(i) Source - initiated transfer :-

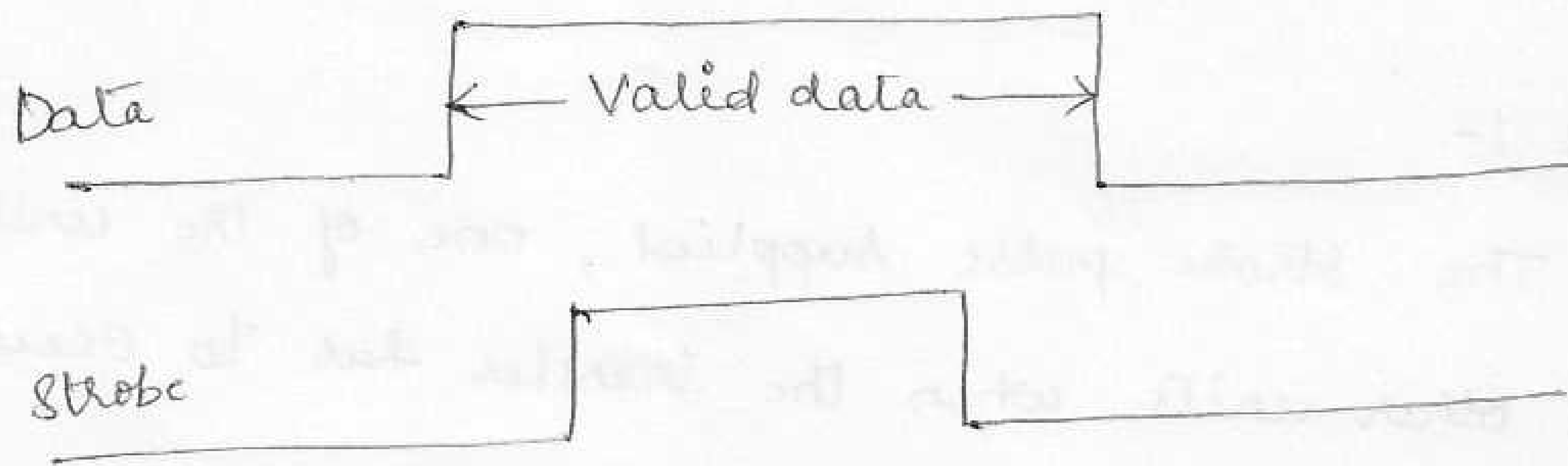
The below fig. shows the source - initiated transfer.



www.jntuworldupdates.org

The databus carries the binary information from source to destination unit. The bus has multiple lines to transfer an entire byte or word. The strobe is a single line that informs the destination when a valid data word is available in the bus.

The below shows the timing diagram.



In this,

The source unit first places data on the data bus.

The data settle to a steady value, the source activates the strobe pulse.

The information on the data bus and the strobe signal remain in the active state for a sufficient time period to allow the destination unit to receive the data item.

The destination unit uses the falling edge of the strobe pulse to transfer the contents of the data bus into one of its internal registers.

The source removes the data from the bus for a period after it disables its strobe pulse. Actually, the source does not have to change the information in the data bus.

The strobe signal is disabled indicates that the databus

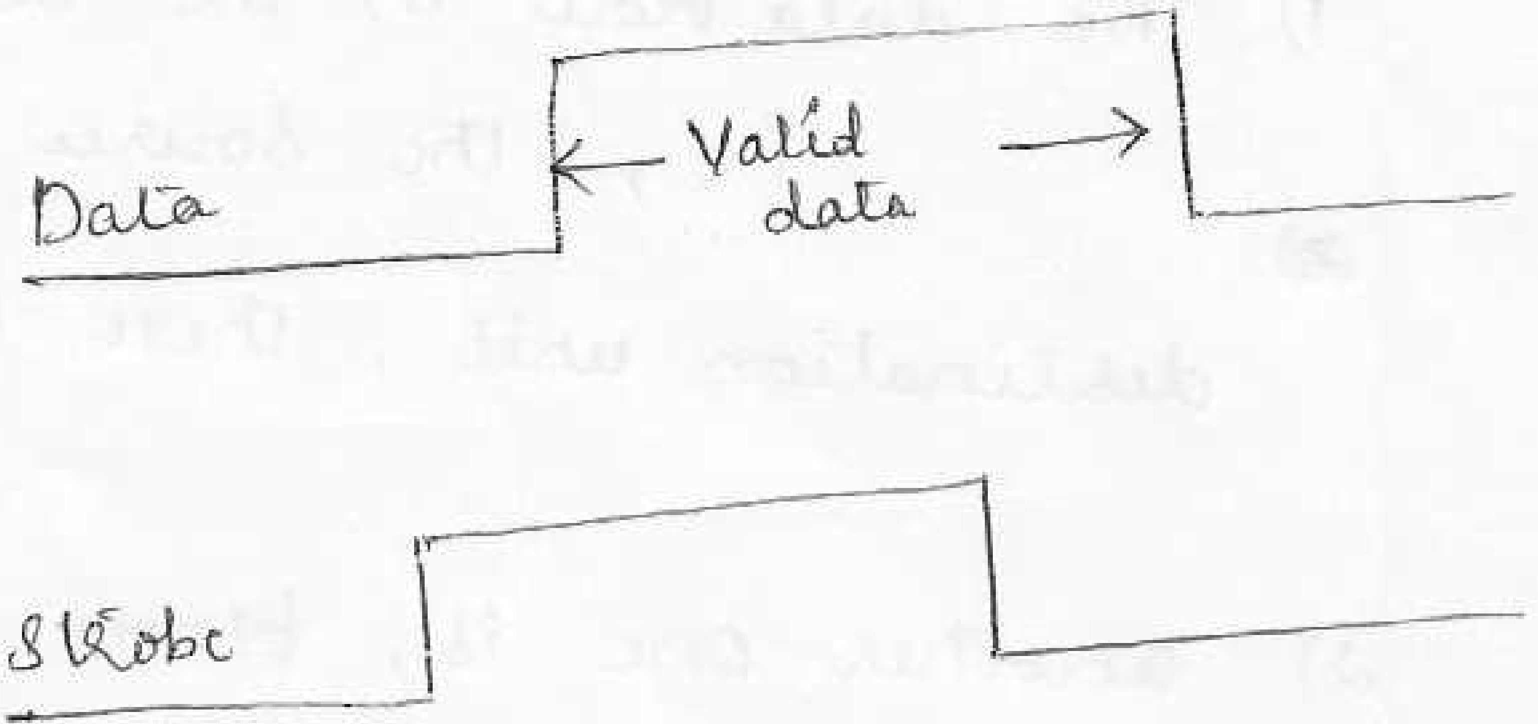
www.jntuworldupdates.org

does not contain valid data.

New valid data will be available only after the strobe is enabled again.

2) Destination - initiated transfer :-

In this, the destination unit activates the strobe pulse, informing the source to provide the data.



The source unit responds by placing the requested binary information on the data bus. The data must be valid and remain in the bus long enough for the destination unit to accept it. The falling edge of the strobe pulse can be used again to trigger a destination register. The destination unit then disables the strobe. The source removes the data from the bus after a predetermined time interval.

Hand shaking :-

In the strobe method, there is no reply from destination unit to source unit, whether it receives a particular data item or not.

The same happens to the destination unit, whether the source unit has actually placed the data on the bus.

www.specworld.in

www.smartzworld.com

www.jntuworldupdates.org

The handshake method solves this problem, by introducing a second control signal that provides a reply to the unit that initiates the transfer.

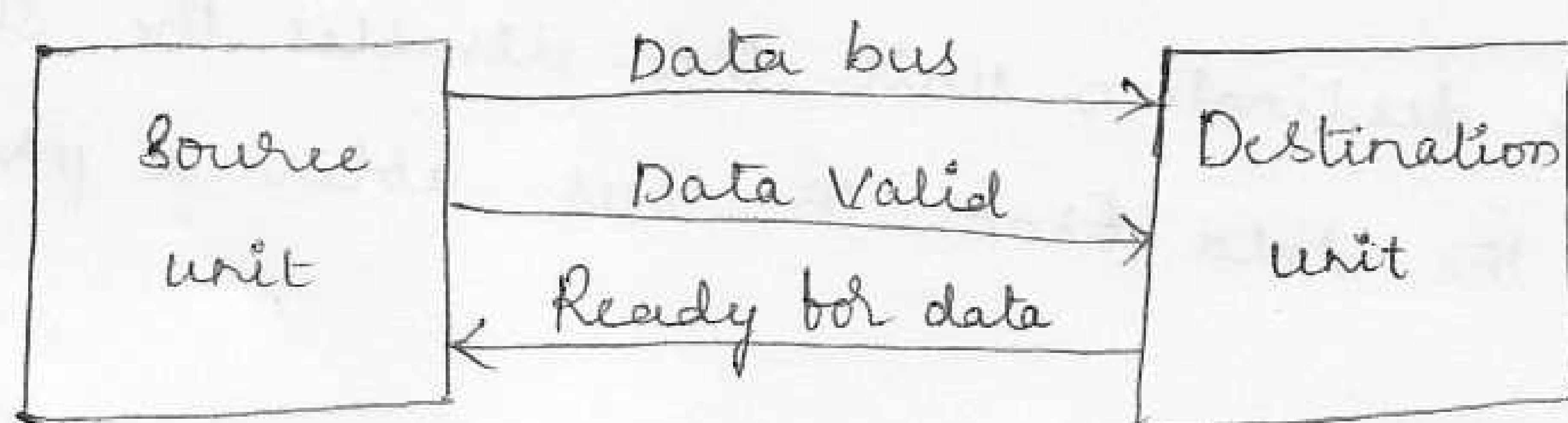
The basic principle is 'two - wire'

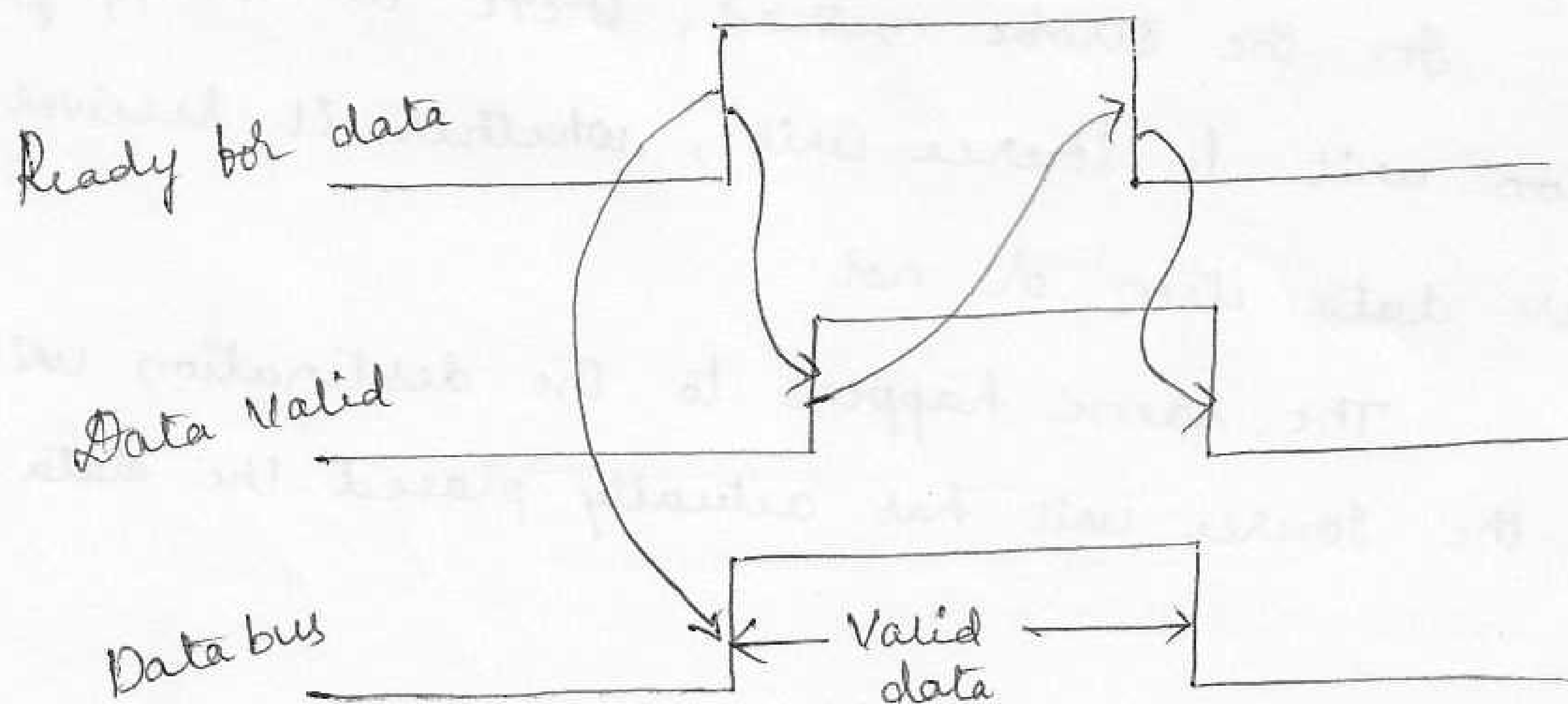
1) The data flow in the bus from source to destination. i.e., the source unit makes information to destination unit, there is a valid data on the bus.

2) another one is, from destination to source. i.e, destination unit informs that, the data is received from the source unit.

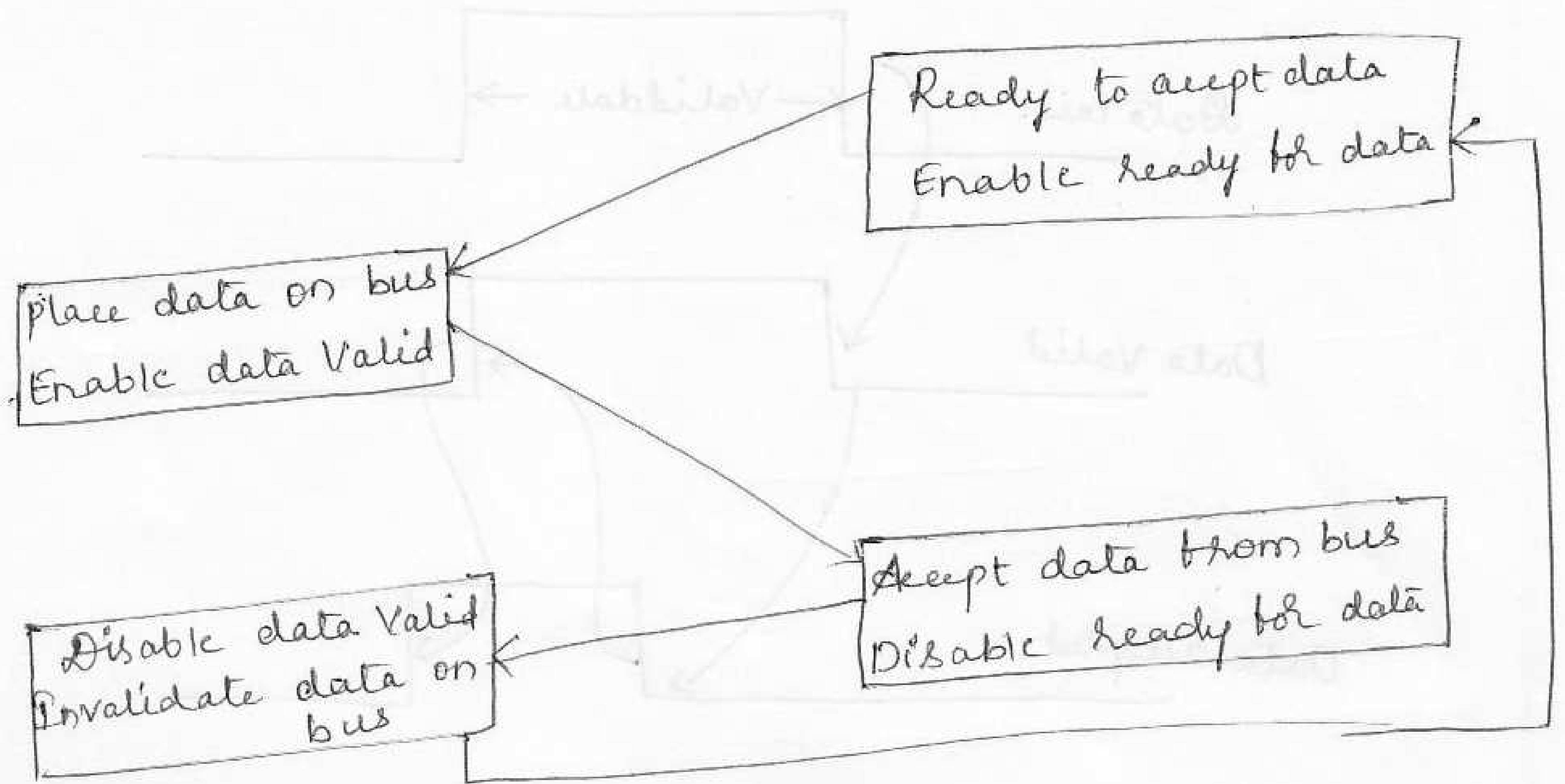
Destination - initiated transfer using handshaking :-

The below fig. shows different procedures how the data transfer exists.





www.jntuworldupdates.org



Here, we are introducing two hand-shaking lines are

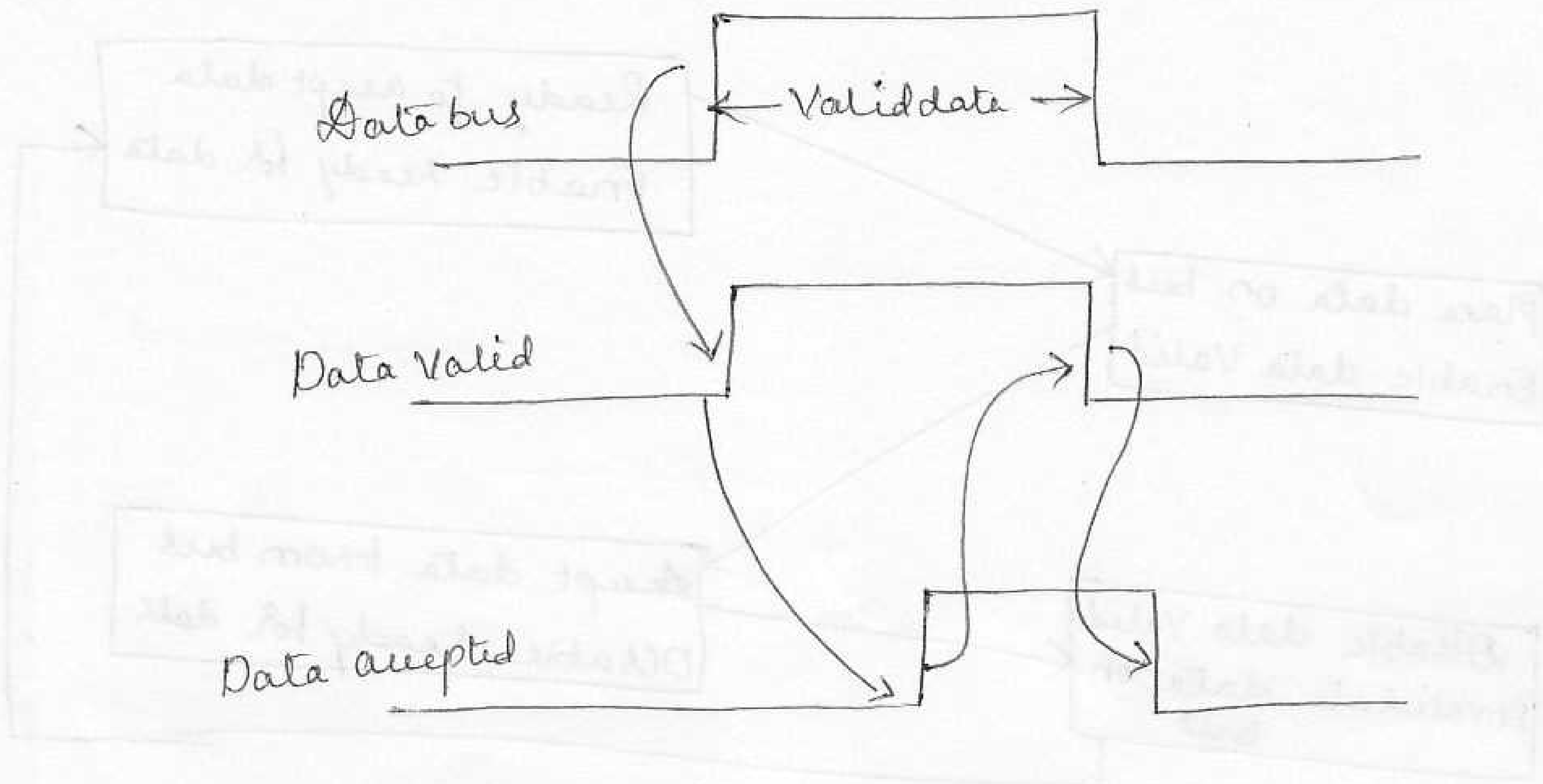
1) data valid, generated by source unit.

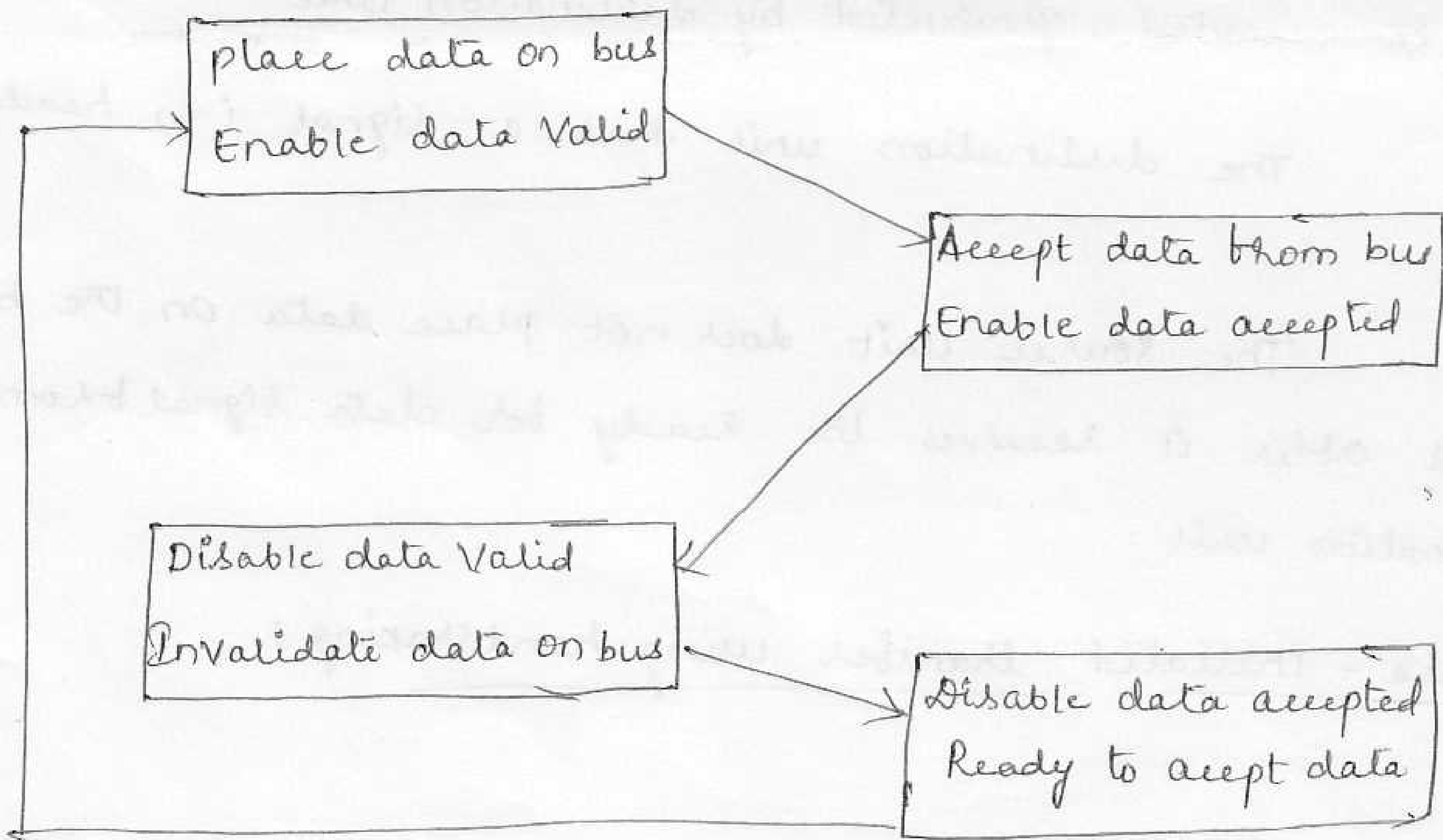
2) data accepted, generated by destination unit.

The destination unit has a signal i.e., ready for data.

The source unit does not place data on the bus until after it receives the ready for data signal from the destination unit.

<u>Source - initiated transfer using handshaking</u> :-





The Source unit initiates the transfer of data by placing the data on the bus and enabling its 'data Valid' signal.

The data accepted signal is activated by

the destination unit after it accepts the data from the bus.

Then source unit disables its data valid signal, which invalidates the data on the bus.

The destination unit then disables its data accepted signal and the system goes into its initial state.

The source does not send the next data item until after the destination unit shows its readiness to accept new data by disabling its 'data accepted signal'.

The handshaking scheme provides a high degree of flexibility and reliability because the successful completion of a data transfer relies on active participation by both units.

If one unit is faulty, the data transfer will not be completed. Such an error can be detected by means of a timeout mechanism, which produces an alarm if the data transfer is not completed within a predetermined time. The time out is implemented by means of an internal clock that starts counting time when the unit enables one of its handshaking control signals.

If the return handshake signal does not respond within a given time period, the unit assumes that an error has occurred. The time out signal can be used to interrupt the processor and hence execute a service routine that takes appropriate error recovery action.

www.jntuworldupdates.org

Asynchronous Serial Transfer :-

The transfer of data between two units may be done in parallel or serial.

In parallel data transmission, each bit of the message has its own path and the total message is transmitted at the same time. In this, n-bit message must be transmitted through 'n' separate conductor paths.

In serial data transmission, each bit in the message is sent in sequence one at a time.

parallel data transmission is faster than the serial data transmission.

Serial transmission can be either synchronous or asynchronous.

In synchronous, the bits are transmitted continuously at the rate of clock pulses. In this, each bit is driven a separate clock of the same frequency.

In asynchronous, binary information is sent only when it is available and the line remains idle when there is no information to be transmitted.

In a asynchronous serial transfer, a special bits are inserted at both ends of the character code. Each character consists of three parts

1) Start bit 2) character bits 3) Stop bits

The first bit called the start bit always starts with '0' and it indicate the beginning of a character.

The last bit called the stop bit is always a '1'.

A transmitted character can be detected by the receiver from knowledge of the transmission rules.

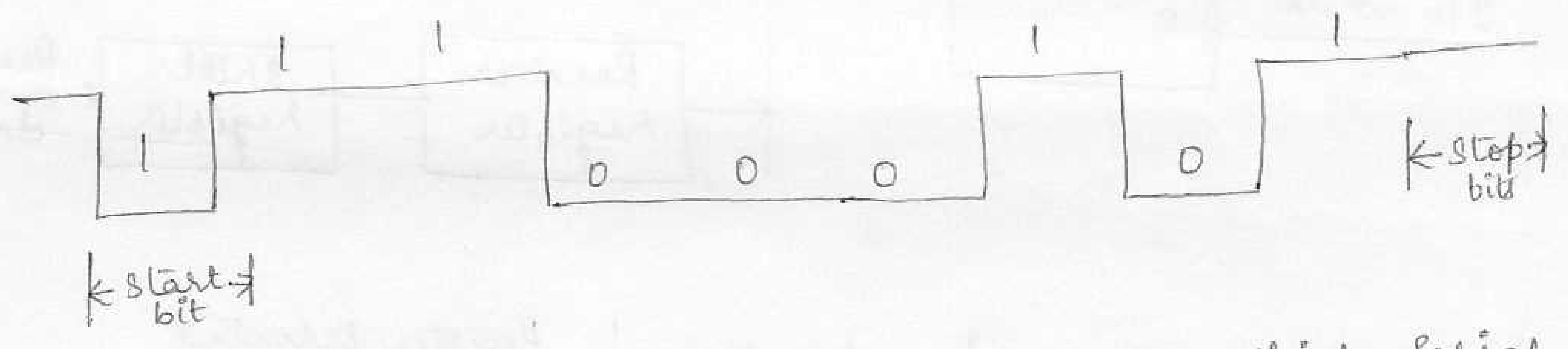
1) When a character is not being sent, the line is kept in the 1-State.

2) The initiation of a character transmission is detected from the start bit, which is always '0'.

3) The character bits always follow the start bit.

4) After the last bit of the character is transmitted, a stop bit is detected when the line returns to the 1-State for atleast one bit time.

Eg: 11000101



The 'baud rate' is defined as the rate at which serial information is transmitted and the data transfer in bits per second.

Integrated circuits are available which are specifically designed to provide the interface between computer and similar interactive terminals. Such a circuit is called an asynchronous communication interface or a universal asynchronous receiver transmitter (UART).

This byte is transferred to a shift register for serial transmission. The receiver portion receives serial information into another shift register, and when a complete data byte is accumulated, it is transferred to the receiver register. The CPU can select the receiver register to read the byte through the databus.

The bits in the status register are used for input and output flags and for recording certain errors that may occur during the transmission.

The chip select is used to select the interface through the address bus. The register select (RS) is associated with the read (RD) and write (WR) controls. Two registers are write-only and two are read-only.

Operation :-

The operation is initialized by the CPU by sending a byte to the control register. The initialization procedure places the interface in a specific mode of operation as it defines certain parameters such as the baud rate to use, how many bits are in each character, whether to generate and check parity and how many stop bits are appended to each character.

In status register, the two bits are used as flags. One bit is used to indicate whether the transmitter register is empty and another bit is used to indicate whether the receiver register is full.

Transmitter :-

The operation of transmitter is as follows :

- The CPU reads the status register and checks the flag to

see if the transmitter register is empty. If it is empty, the CPU transfers a character to the transmitter register and the interface clears the flag to mark the register full.

- The first bit in the transmitter shift register is set to '0' to generate a start bit. This character is transferred in parallel from the transmitter register to shift register and appropriate stop bits are appended in the shift register. The transmitter register is empty.

- The character is transmitted one bit at a time by shifting the data in the shift register at the specified baud rate.

- The CPU can transfer another character to the transmitter register after checking the flag in the status register.

- The interface is double buffered because a new character can be loaded as soon as the previous one starts transmission.

Receiver :-

- The receiver data input is in the 1-state when the line is idle.

- The receiver control monitors the receive-data line for a '0' signal to detect the occurrence of a start bit. Once a start bit has been detected, the incoming bits of the character are shifted into the shift register at the prescribed baud rate.

- After receiving the data bits, the interface checks for the parity and stop bits.

- The character with out the start and stop bits is then

www.jntuworldupdates.org

transferred in parallel from the shift register to the receiver register.

- The flag in the status register is set to indicate that the receiver register is 'full'.

- The CPU reads the status register and checks the flag, and if set, it reads the data from the receiver register.

Errors :-

There are three possible errors, that the interface checks during transmission are

1) parity error 2) framing error 3) overrun error.

A parity error occurs if the number of 1's in received data is not correct parity.

A framing error occurs if the right number of stop bits is not detected at the end of the received character.

An overrun error occurs if the CPU does not read the character from the receiver register before the next one becomes available in the shift register. Overrun error results in a loss of characters in the received data stream.

First - In, First - Out Buffer :-

FIFO buffer is a memory unit that stores information in a manner that the item first in is the item first out. It survive with separate i/p and o/p terminals.

The important feature of this buffer is that, it can i/p data and o/p data at two different rates and the o/p data are always in the same order, at which the data entered the buffer.

The logic diagram of a typical 4x4 FIFO buffer is shown in fig.

It consists of four 4-bit registers RI, I = 1,2,3,4 and a control register with flip-flops F_i, i = 1,2,3,4 one for each register.

The FIFO can store four words of four bits each. The no. of bits per word can be increased by increasing the no. of bits in each register and the no. of words can be increased by increasing the number of registers.

A flip-flop F_i in the control register that is set to '1' indicates that a 4-bit data word is stored in the corresponding register RI.

A '0' in F_i indicates that the corresponding register does not contain valid data.

The control register directs the movement of data through the registers.

Whenever the F_i bit of the control register is set $F_i = 1$ and the F_{i+1} bit is reset ($F'_{i+1} = 1$), a clock is generated causing register R(I+1) to accept the data from register RI.

The clock transition sets F_{i+1} to '1' and resets F_i to '0'. This causes the control flag to move one position to the right together with the data.

Data are inserted into the buffer provided that the input ready signal is enabled. This occurs when the first control flip-flop F_1 is reset, indicating that the register R1 is empty.

Data are loaded from the I/p lines by enabling the clock in R1 through the 'insert' control line. The same clock

www.jntuworldupdates.org

Sets F_1, which disables the i/p ready control, indicating that the fifo is now busy and unable to accept more data.

The ripple - through process begins provided that R2 is empty. The data in R1 are transferred into R2 and F_1 is cleared. This enables the i/p ready line, indicating that the I/p's are now available for another data word.

If the fifo is full, F_1 remains set and the i/p ready line stays in the '0' state.

The data falling through the registers stack up at the o/p end. The o/p ready control line is enabled when the last control flip-flop F_4 is set, indicating that there are valid data in the o/p register R4.

The O/P data from R4 are accepted by a destination unit, which then enables the 'delete' control signal. This resets F_4 causing o/p ready to disable, indicating that the data on the o/p are no longer valid.

www.jntuworldupdates.org

MODES OF TRANSFER

Data transfer between the central computer and I/O devices may be handled in a variety of modes. Data transfer to and from peripherals may be handled in one of three possible modes:

1) Programmed I/O
2) Interrupt-Initiated I/O
3) Direct memory Access

Programmed I/O:

Programmed I/O operations are the result of I/O instructions written in the computer program . Each data item transfer is initiated by an instruction in the program . The transfer is to and from a CPU register and peripheral.Other instructions are needed to transfer the data to and from CPU and memory . Transferring data under program control requires constant monitoring of the peripheral by the CPU.

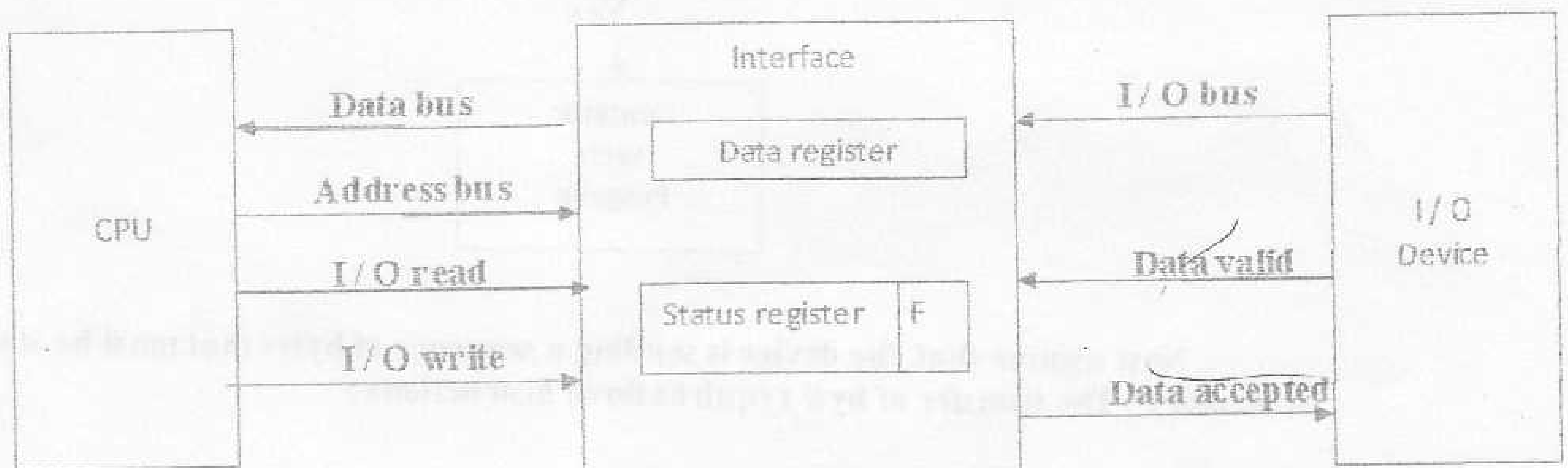
In the Programmed I/O method, the CPU stays in a program loop until the I/O unit indicates that it is ready for data transfer.

This is a time-consuming process since it keeps the processor busy needlessly

Example:

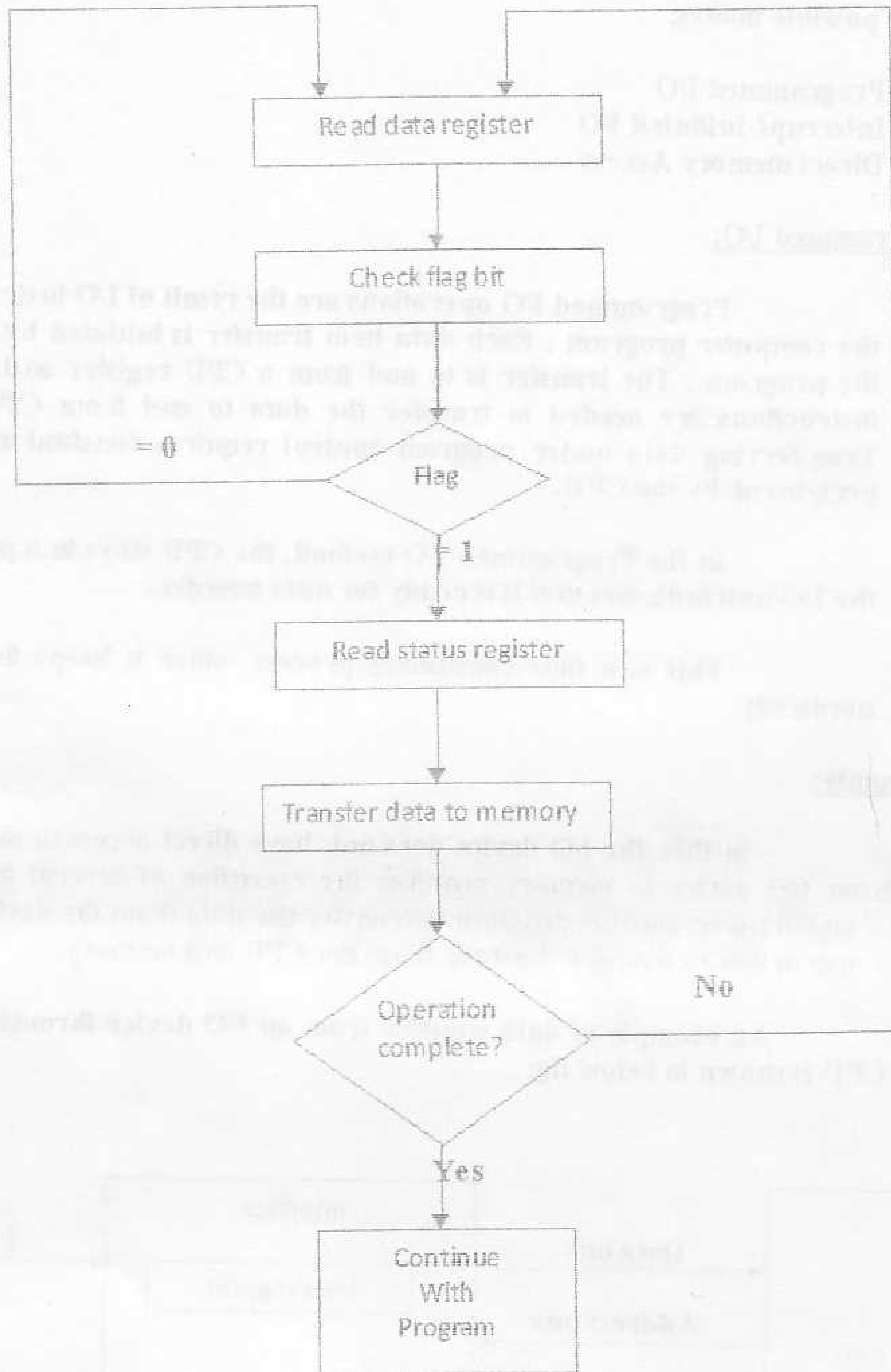
In this, the I/O device does not have direct access to memory. A transfer from an I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from the device to the CPU and store instruction to transfer the data from the CPU and memory.

An example of data transfer from an I/O device through an interface into the CPU is shown in below fig:



The device transfers bytes of information one at a time .When a byte of data is available , the device places it in the I/O bus and enables the data valid line. The interface accepts the byte into its data register and enables the data accepted line . The interface sets a bit in the status register as F bit. The device can disable the data valid line ,but it will not transfer another byte until the data accepted line is disabled by the interface .

The flow chart of the program that must be written for the CPU is shown below:



Now assume that the device is sending a sequence of bytes that must be stored in memory. The transfer of byte requires three instructions:

1) Read the Status register.
2) Check the status of the flag bit and branch to step 1 if not set or to step3 if set.
3) Read the data register.

Each byte is read into a CPU register and then transferred to memory with a store instruction. A common I/O programming task is to transfer a block of words from an I/O device and store them in a memory buffer.

This method is used in small low-speed computers.
Here the CPU is wasting time while checking the flag instead of doing some other useful processing task.

Interrupt-Initiated I/O :

Programmed I/O is an time-consuming process ,so in order to keep CPU in a busy state introduce an interrupt and special commands to inform the interface to issue an interrupt request signal when data are available from the device. In the mean time ,the CPU can proceed to execute another program

When the device is ready for data transfer , it generates an interrupt request to the computer.

When there is an external interrupt signal is generated ,the CPU will stops the execution of the original program and branches to service program to process the I/O transfer and then returns back to the original program.

The CPU responds to the interrupt signal by storing the return addresses from program counter into a memory stack and then control branches to a service routine that processes the required I/O transfer.

The way the processor chooses the branch address of the service routine varies from one unit to another. There are two methods for using this:

1) Vectored interrupt
2) Non Vectored interrupt

In a nonvectored interrupt , the branch address is assigned to fixed location in memory.

In vectored interrupt, the source that interrupts supplies the branch information to the computer. This information is called the interrupt vector.

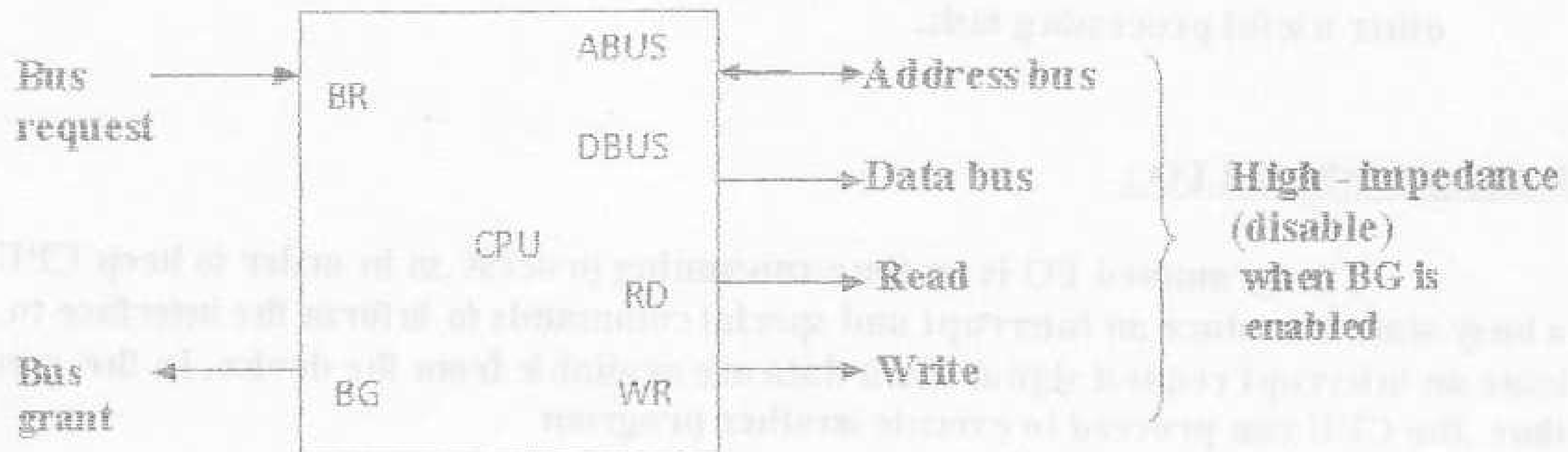
Direct Memory Access (DMA) :

The transfer of data between a fast storage device such as magnetic disk is often limited by the speed of the CPU. Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This transfer technique is called DMA. During DMA transfer,the cpu is idle and has no control of the memory buses . A DMA controller takes over the buses to manage the transfer directly between the I/O device and memory.

The CPU may be in idle state in many ways. One method is used in microprocessors is to disable the buses through special control signals.

The below fig. shows the control signals of the CPU to facilitate the DMA transfer.

www.jntuworldupdates.org



The bus request (BR) input is used by the DMA controller to request the CPU to get control of buses.When this is input is active ,the CPU terminates the execution of the current instruction and places the address bus, data bus and read & write lines into the high-impedance state.

The CPU activates the bus grant(BG) output to inform the external DMA that the buses are in a high-impedance state. Then the DMA will take over the control of the buses with out any processor intervention. When the DMA terminates the transfer,it disables the bus request line.the CPU disables the bus grant,takes control of the buses and returns to its normal operation.

When the DMA takes control of the bus system,it communicates directly with the memory. The transfer can be done in different ways:

1) Burst Transfer: In this , a block sequence consisting of a number of memory words is transferred in a continuous burst while the DMA controller is master of all the buses. This Mode of transfer can be used by fast device such as magnetic disks etc.,

2) Cycle stealing : In this , the DMA controller to transfer one data word at a time ,after which it must return control of the buses to the CPU .The CPU delays its operation ffor one memory cycle to allow the direct memory I/O transfer to steal one memory cycle.

DMA Controller:

The below figure shows the DMA controller. The communicates with the CPU via the data bus and control lines.

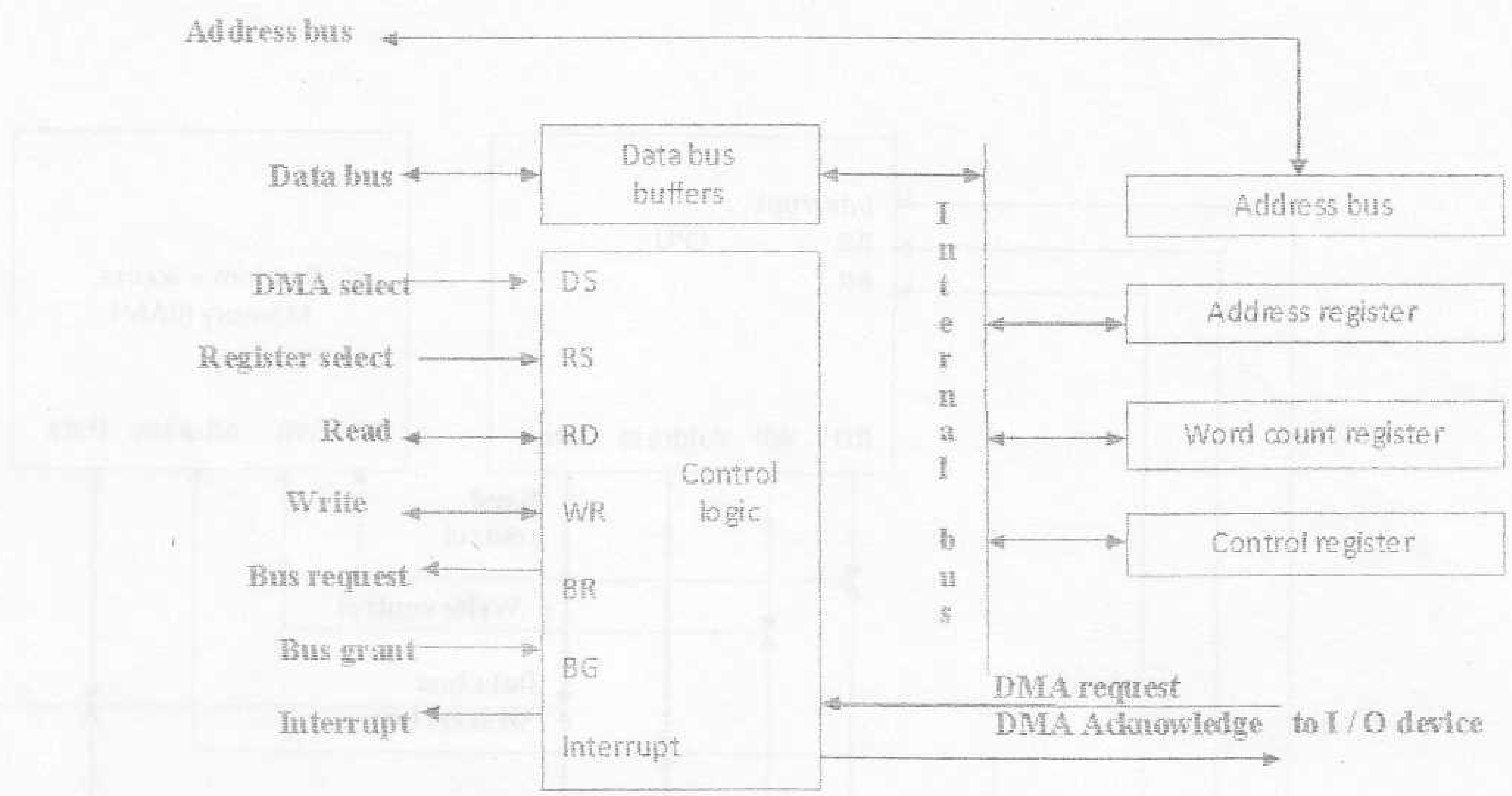
The registers in the DMA are selected by the CPU through the address bus by enabling the DS(DMA select) and RS (register select) inputs . The read and write inputs are bidirectional . When the BG(bus grant) input is 0 , the CPU communicate with the DMA registers through the data bus to read from or write to the DMA registers

When BG=1 , the CPU has the buses and the DMA can communicate directly with the memory by specifying an address in the address bus and activating the RD or WR control.

The DMA communicates with the external peripheral through the request and acknowledge lines.

The DMA controller has three registers:

1) address register
2) word count register
3) control register.



The address register contains an address to specify the desitred location in memory . The address bits go through bus buffers into the address bus. The address register is incremented after each word that is transferred to memory.

The word count register holds the number of words to be transferred. This register is decremented by one after each word is transferred .

The control register specifies the mode of transfer.All the registers in the DMA appear to the CPU as I/O interface . Thus the CPU can read from or write into the DMA registers under program control via databus.

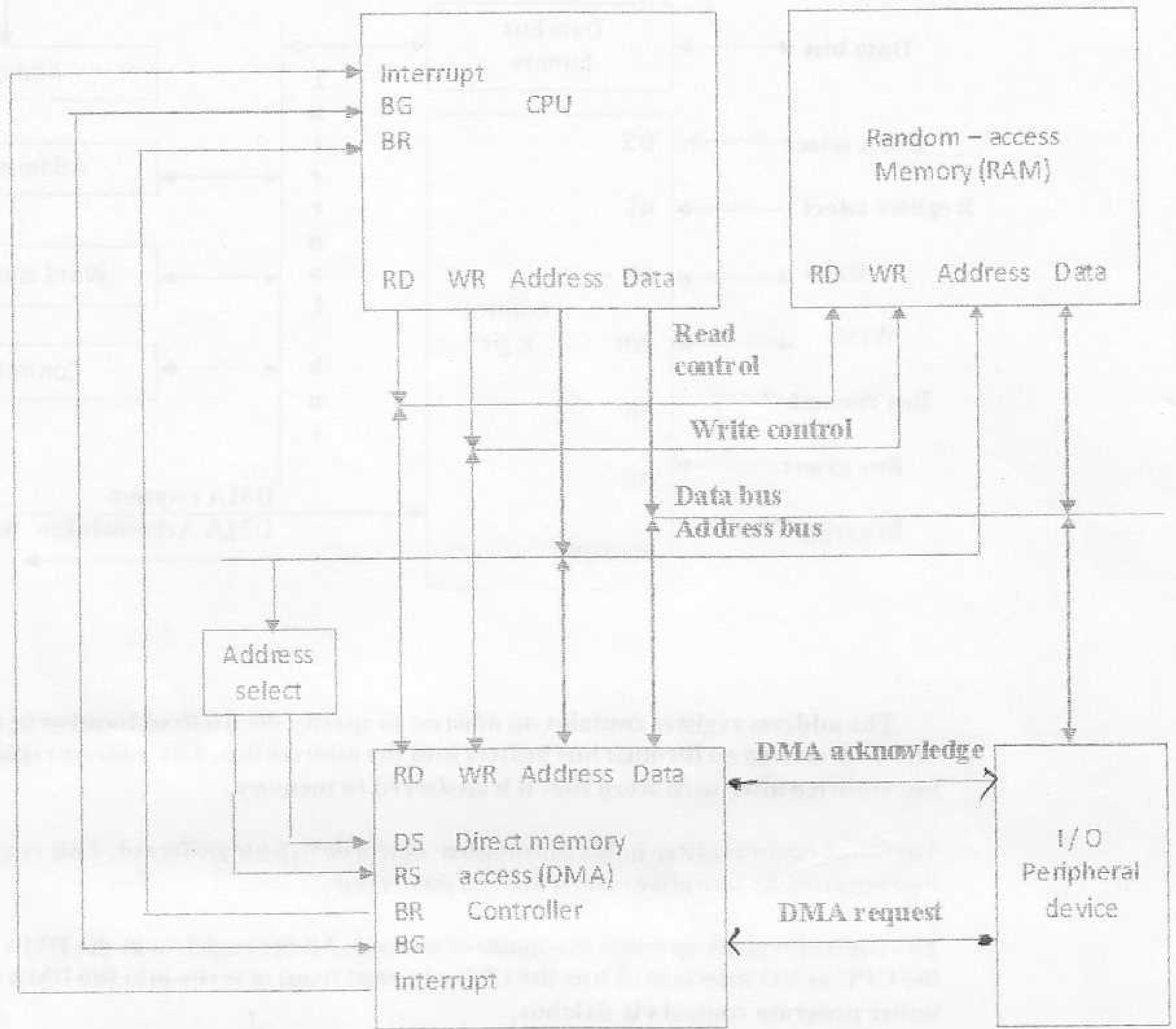
The CPU initializes the DMA by sending the following information through the databus:

1) The starting address of the memory block where data are available(for read) or where data are to be stored(for write)
2) The word count , which is the number ofprds in the memory block.
3) Control to specify the mode of transfer such as read or write.
4) A conrol to start the DMA transfer.

<u>DMA transfer :</u>

The position of the DMA controller among the other components in a system is illustrated is shown in below fig::

www.jntuworldupdates.org



The CPU communicates with the DMA through the data and address buses with any interface unit. The DMA has its own address, which activates the DS and RS lines. The CPU initializes the DMA through the data bus.

Once the DMA receives the start control command, it can start the transfer between the peripheral device and the memory.

When the peripheral devices sends a DMA request, the DMA controller activates the BR line, informing the CPU to relinquish the buses.

X[The CPU initializes the DMA through the data bus. Once the DMA receives the start control command, it can start the transfer between the peripheral device and the memory.]X

~~When the peripheral device sends a DMA request,~~

The CPU responds with its BG line, informing the DMA that its buses are disabled. The DMA then puts the current value of its address register into the address bus, initiates the RD or WR signal, and sends a DMA acknowledge to the peripheral device.

Both RD and WR lines in the DMA controller are bidirectional. The direction of transfer depends on the status of the BG line. When BG=0, the RD and WR are i/p lines allowing the CPU to communicate with the internal DMA registers.

When BG=1, the RD and WR are o/p lines from the DMA controller to the random-access memory to specify the read or write operation for the data.

When the peripheral device receives a DMA acknowledge, it puts a word in the databus (for write) or receives a word from the databus (for read). Thus the DMA controls the read or write operations and supplies the address for the memory.

The peripheral unit can then communicate with memory through the databus for direct transfer between the two units while the CPU is momentarily disabled.

For each word that is transferred, the DMA increments its address register and decrements its word-count register. If the word count does n't reach zero, the DMA checks the request line coming from the peripheral.

For a high speed device, the line will be active as soon as the previous transfer is completed. A second transfer is initiated, and the process continues until the entire block is transferred.

If the peripheral device speed is slow, then the DMA request line will become late. In this case, the DMA disables the bus request line so that the CPU can continue to execute its program. When the peripheral requests a transfer, the DMA requests the buses again.

If the word count register reaches zero, the DMA stops the transaction and removes its bus request.

It also informs the CPU of the termination by means of an interrupt. Now, the CPU responds to the interrupt, it reads the content of the word count register. The zero of this register indicates that all the words were transferred successfully.

If the DMA controller has more than channel. Each channel has a request and acknowledge pair of control signals which are connected to separate peripheral devices.

Each channel also has its own address register and word count register within the DMA controller. A priority among the channels may be established so that the channels with high priority are serviced before channels with lower priority.

It is used for fast transfer of information between magnetic disks and memory.

Input - Output Processor (IOP) :—

A computer may incorporate one or more external processors and assign them the task of communicating directly with all I/O devices, instead of having each interface communicate with the CPU.

An Input-Output Processor (IOP), is a processor with direct memory access capability that communicates with I/O devices. Each IOP takes care of input and output tasks involved in I/O transfers.

A processor that communicates with remote terminals over telephone and other communication media in a serial fashion is called a data Communication Processor.

I/O processing:—

We know that, DMA controller must be entirely set up by the CPU. where as IOP can fetch and execute its own instructions.

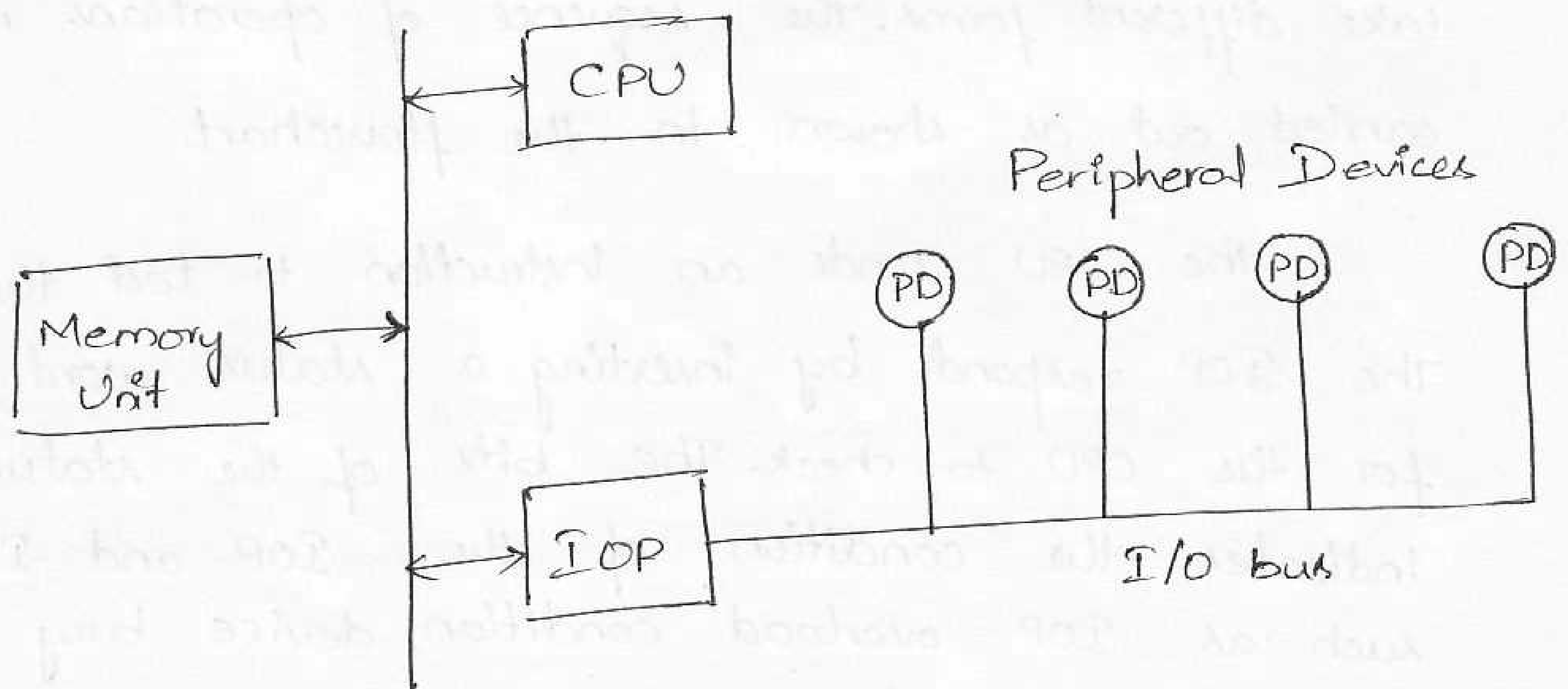


fig:- Block Diagram of a computer with I/O processor

www.jntuworldupdates.org

In the block diagram, the memory unit occupies a central position and can communicate with each processor by means of direct memory access. The CPU is responsible for processing data needed in the solution of computational tasks. The IOP provides a path for transfer of data between various peripheral devices and the memory unit. The CPU is usually assigned the task of initiating the I/O program.

The data formats of peripheral devices differ from memory and CPU data formats. The IOP must structure data words from many different sources. The communication between the IOP and the devices attached to it is similar to the program control method of transfer. Communication with the CPU and IOP depends on the level of sophistication in system & communication with memory is similar to the direct access method.

CPU-IOP Communication:-

The communication between CPU and IOP may take different forms. The sequence of operations may be carried out as shown in the flowchart.

The CPU sends an instruction to test the IOP path. The IOP responds by inserting a status word in memory for the CPU to check. The bits of the status word indicates the condition of the IOP and I/O device, such as IOP overload condition, device busy with another transfer etc.,

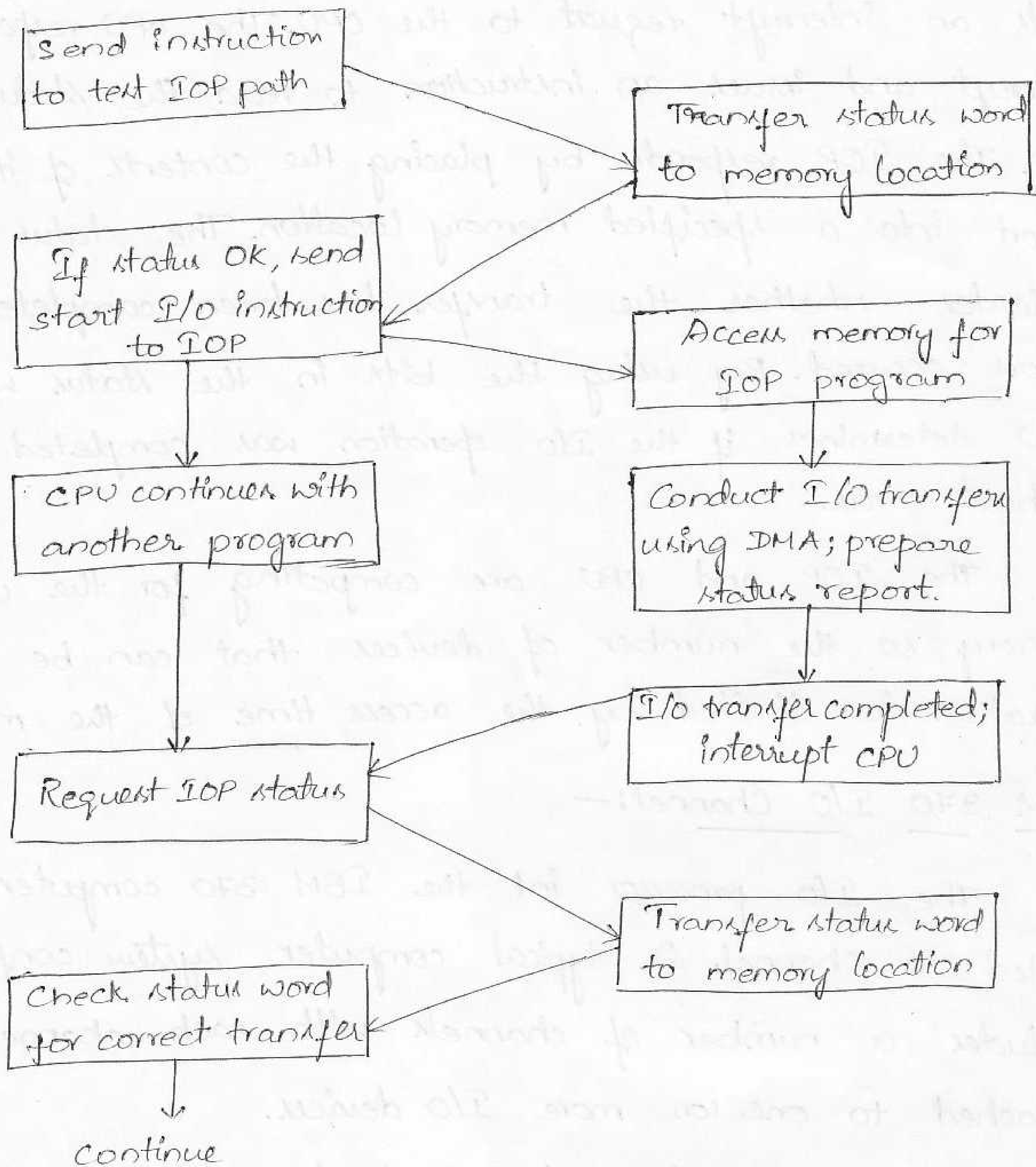


fig: CPU - IOP communication

The CPU refers to the status word in memory to decide the work to be done. Then CPU sends the instruction to start I/O transfer. The memory address received with this instruction refers to the program. Ep CPU can continue with another program while IOP is busy with the I/O program.

www.jntuworldupdates.org

When IOP terminates the execution of its program, it sends an interrupt request to the CPU. The CPU responds to the interrupt and issues an instruction to read the status from IOP. The IOP responds by placing the contents of its status report into a specified memory location. The status word indicates whether the transfer has been completed (or) any errors occured. By using the bits in the status word, the CPU determines if the I/O operation was completed satisfactorily without errors.

The IOP and CPU are competing for the use of memory, so the number of devices that can be in operation is limited by the access time of the memory.

IBM 370 I/O Channel:—

The I/O processor in the IBM 370 computer is called a channel. A typical computer system configuration includes a number of channels with each channel attached to one (or) more I/O devices.

There are three types of channels:

i, Multiplexer
ii, Selector
iii, Block Multiplexer.

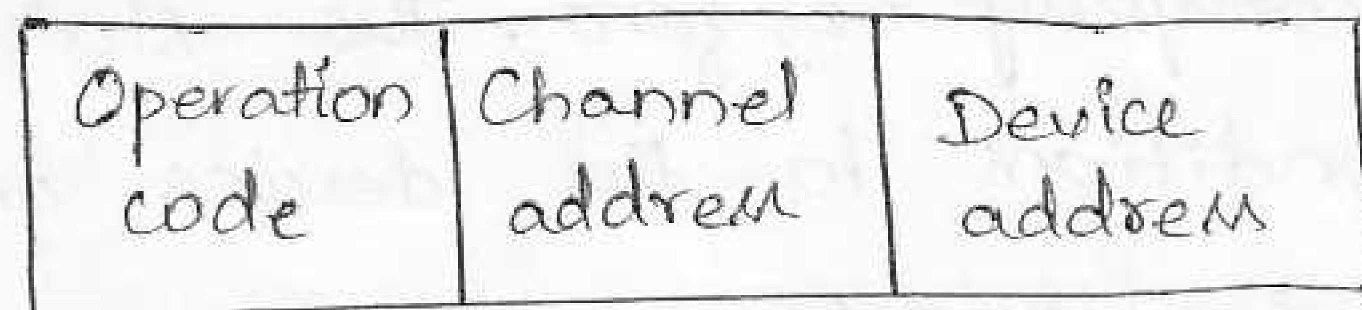
The Multiplexer channel can be connected to a number of slow and medium-speed devices. It is capable of operating with a number of I/O devices simultaneously.

www.specworld.in

www.smartzworld.com

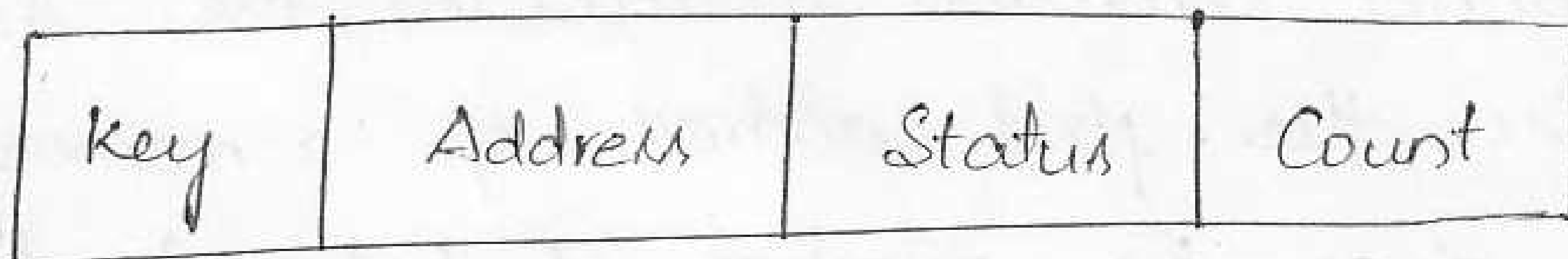
The selector channel is designed to handle one I/O operation at a time. It is normally used to control one high-speed device.

The block-Multiplexer channel combines the features of both the multiplexer and selector channels.



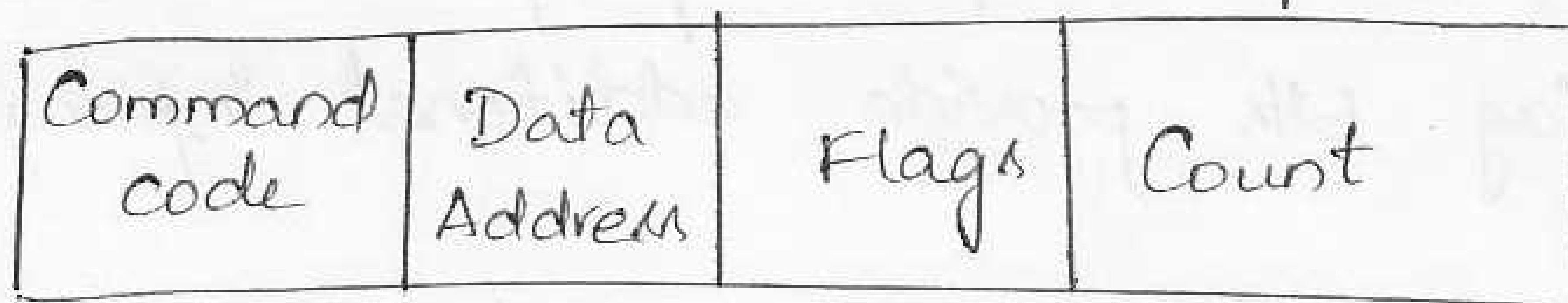
Operation code	Channel address	Device address

(a) I/O instruction format



Key	Address	Status	Count

(b) Channel status word format



Command code	Data Address	Flags	Count

(c) Channel command word format

fig:- IBM 370 I/O related word formats

The I/O instruction format has three fields: operation code, channel address, and device address, which contains the address of channels of computer system. Each channel may be connected to several devices and its address is specified by device address. The operation code specifies one of eight I/O instructions: start I/O, start I/O fast release, test I/O, clear I/O, halt I/O, halt device, test channel & store channel. The CPU can check the condition code in the PSW (program Status Word) to determine the result of the I/O operation

www.jntuworldupdates.org

The channel status word is always stored in location 64 in memory. The key field is a protection mechanism used to prevent un-authorized access by one user information to another user. Address field gives the address of last command word used by the channel. Count field gives the residue count when the transfer was completed successfully i.e., zero. The status field identifies the conditions in the device and the channel & also errors during transfer.

In the Channel Command Word (CCW), the data address field specifies the first address of a memory buffer. The count field gives the number of bytes involved in the transfer. The command field specifies an I/O operation and the flag bits provide additional information for the channel.

The command field corresponds to an operation code that specifies one of the six types of I/O operations:

1. Read: Transfer data from I/o device to memory
2. Write: Transfer data from memory to I/O device
3. Read backwards: Read magnetic tape with tape moving backward
4. Control: Used to initiate an operation not involving transfer of data.
5. Sense: Informs the channel to transfer its channel status word to memory location 64.
6. Transfer in channel:- Used instead of a jump instruction. Here the data address field specifies the address of next command word to be executed by the channel.

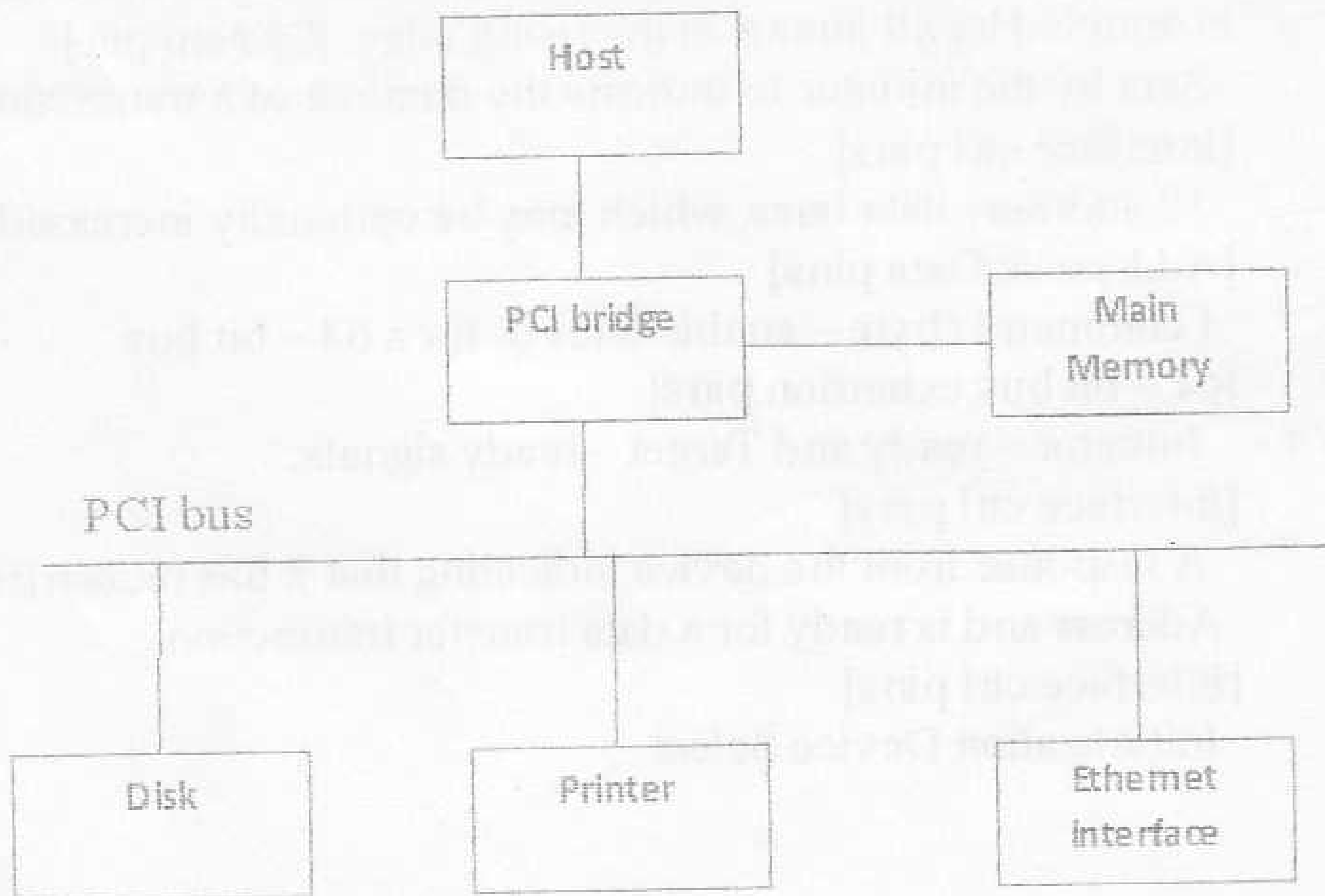
PCI: (Peripheral Component Interconnect). BUS:-

The PCI is a popular high – bandwidth, Processor – independent bus that can function as a mezzanine (or) peripheral bus. Compared with other common bus specifications, PCI delivers better system performance for high – speed I / O subsystems like graphics display adapters, network interface controllers, disk controllers and so on. PCI is specifically designed to meet economically the I / O requirements of modern systems. It requires very few chips to implement and supports other buses attached to the PCI bus.

PCI is designed to support a variety of microprocessor based configurations, including both single – and multiple – processor systems. It Provides a general – Purpose set of functions. It makes use of synchronous timing and a centralized arbitration scheme.

PCI was developed as a low – cost bus that is truly processor independent. An important feature that the PCI is a plug – and – play capability – for connecting I / O devices. To connect a new device, the user simply connects the device interface board to the bus.

Data transfer: - Most memory transfers involve a burst of data rather than one word. Data are transferred between the cache and the main memory in bursts of several words each. The words involved in such a transfer are stored at successive memory locations. When the processor specifies an address and requests a read operation from the main memory, the memory responds by sending a sequence of data words starting at that address. During a write operation the processor sends a memory address followed by a sequence of data words, to be written in successive memory locations starting at that address. The PCI is designed primarily to support this mode of operation.



www.jntuworldupdates.org

The PCI bridge provides a separate physical connection for the main memory. For electrical reasons, the bus may be further divided into segments connected via bridges. The segment device address is mapped into the processor's memory address space.

At any given time, one device is the bus master. It has the right to initiate data transfer y issuing read and write commands. A 'master' is called an "initiator" in PCI terminology. This is either a processor (or) a DMA controller. The addressed device that responds to read & write commands is called a "target".

Consider a bus transaction in which the processor reads four 32 – bit words from the memory. In this case, the initiator is the processor and the target is the memory. A complete transfer operation on the bus, involving an address and a burst of data, is called a "transaction". Individual word transfers within a transaction are called "phases". A clock signal provides the timing reference used to coordinate different phases of transaction. All signal transactions are triggered by the rising edge of the clock.

BUS Structure:- PCI may be configured as a 32 – or 64 – bit bus. PCI contains 49 mandatory signal lines. These are divided into 5 groups.

1.System pins :- Include the clock and reset pins.
2.Address & data pins
3.Interface ctrl pins
4.Arbitration pins
5.Error reporting pins.

And PCI specification defines 51 optional signal lines divided into 4 groups.

1.Interrupt pins
2.cache support pins
3.64 – bit bus extension pins
4.JTAG / boundary scan pins

PCI Commands : -

Name	Function
1.CLK -	A 33 – MHZ clock. Provides timing to all transactions and is sampled by all inputs on the rising edge. [System pin]
2.FRAME# -	Sent by the initiator to indicate the duration of a transaction. [Interface ctrl pins]
3.AD -	32 address / data lines, which may be optionally increased to 64. [Address & Data pins]
4.C/BE# -	4 command / byte – enable lines (8 for a 64 – bit bus. [64 – bit bus extension pins]
5.IRDY#, TRDY# -	Initiator – ready and Target – ready signals. [Interface ctrl pins]
6.DEVSEL# - (Device Select	A response from the device indicating that it has recognized its Address and is ready for a data transfer transaction. [Interface ctrl pins]
7.IDSWL# -	Initialization Device Select

Serial communication standards

Serial communication is a popular means of transmitting data between a computer and a peripheral device such as a programmable instrument or even another computer. Serial communication uses a transmitter to send data, one bit at a time, over a single communication line to a receiver. we can use this method when data transfer rates are low or you must transfer data over long distances. Serial communication is popular because most computers have one or more serial ports, so no extra hardware is needed other than a cable to connect the instsument to the computer or two computers together.

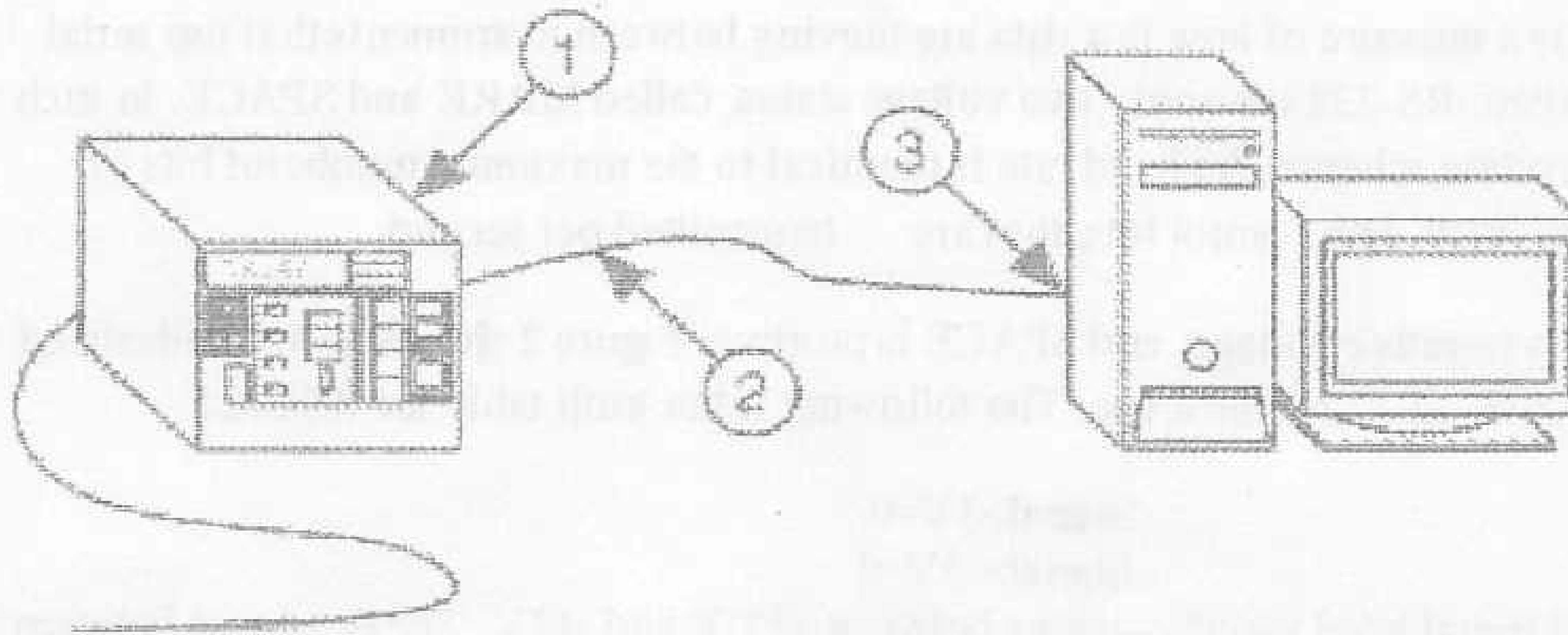


Figure 1:

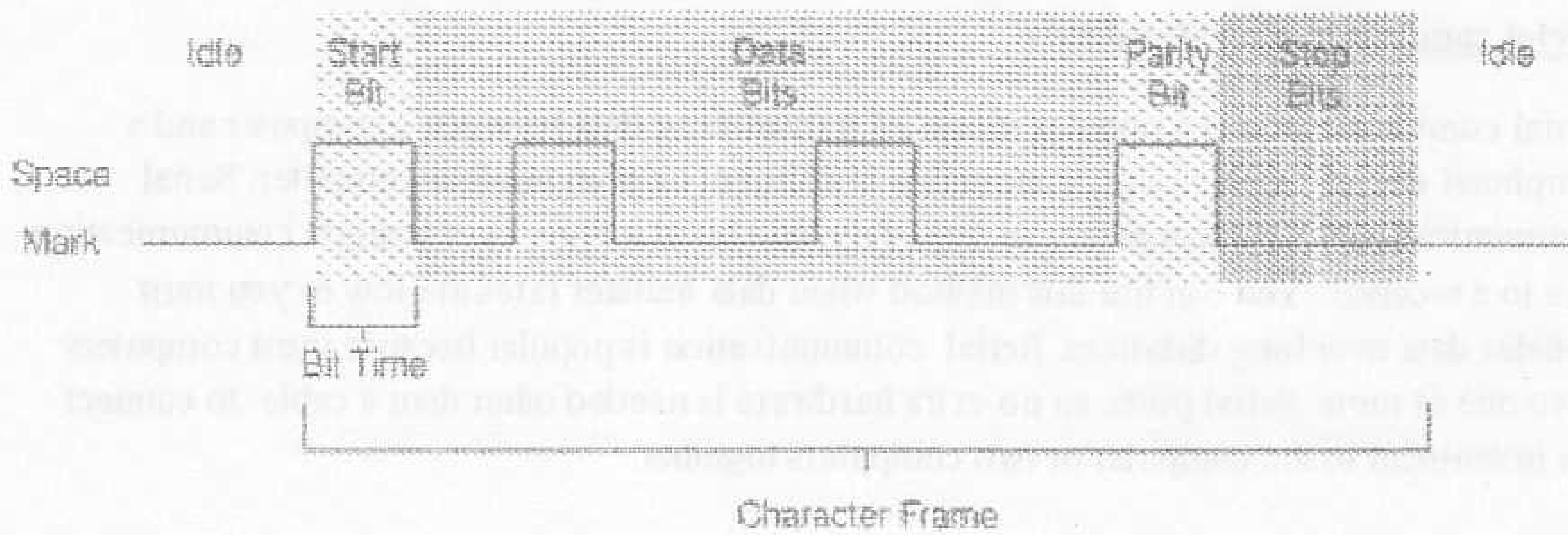
1: RS-232 Instrument,2: RS-232 Cable, 3: Serial Port

Serial communication requires that you specify the following four parameters:

1. The baud rate of the transmission
2. The number of data bits encoding a character
3. The sense of the optional parity bit
4. The number of stop bits

Each transmitted character is packaged in a character frame that consists of a single start bit followed by the data bits, the optional parity bit, and the stop bit or bits. Figure 2 shows a typical character frame encoding the letter m.

www.jntuworldupdates.org



Character Frame

Baud rate is a measure of how fast data are moving between instruments that use serial ommunication. RS-232 uses only two voltage states, called MARK and SPACE. In such a two-state coding scheme, the baud rate is identical to the maximum numberof bits of information, including control bits, that are transmitted per second.

MARK is a negative voltage, and SPACE is positive. Figure 2 shows how the idealized signal looks on an oscilloscope. The following is the truth table for RS-232:

Signal>3 V=0
Signal>-3V=1

The output signal level usually swings between +12 V and -12V. The dead area between +3 V and -3 V is designed to absorb line noise.
A start bit signals the beginning of each character frame. It is a transition from negative (MARK) to positive (SPACE) voltage. Its duration in seconds is the reciprocal of the baud rate. If the instrument is transmitting at 9,600 baud, the duration of the start bit and each subsequent bit is about 0.104 ms. The entire character frame of eleven bits would be transmitted in about 1.146 ms.
Data bits are transmitted upside down and backwards. That is, inverted logic is used, and the order of transmission is from least significant bit (LSB) to most significant bit (MSB). To interpret the data bits in a character frame, you must read from right to left and read 1 for negative voltage and 0 for positive voltage. This yields 1101101 (binary) or 6D (hex). An ASCII conversion table shows that this is the letter m.
An optional parity bit follows the data bits in the character frame. The parity bit, if present, also follows inverted logic,1 for negative voltage and 0 for positive voltage. This bit is included as a simple means of error handling. You specify ahead of time whether the parity of the transmission is to be even or odd. If the parity is chosen to be odd, the transmitter then sets the parity bit in such a way as to make an odd number of ones among the data bits and the parity bit. This transmission uses odd parity. There are five ones among the data bits, already an odd number, so the parity bit is set to 0.

The last part of a character frame consists of 1, 1.5, or 2 stop bits. These bits are always represented by a negative voltage. If no further characters are transmitted, the line stays in the negative (MARK) condition. The transmission of the next character frame, if any, is heralded by a start bit of positive (SPACE) voltage.

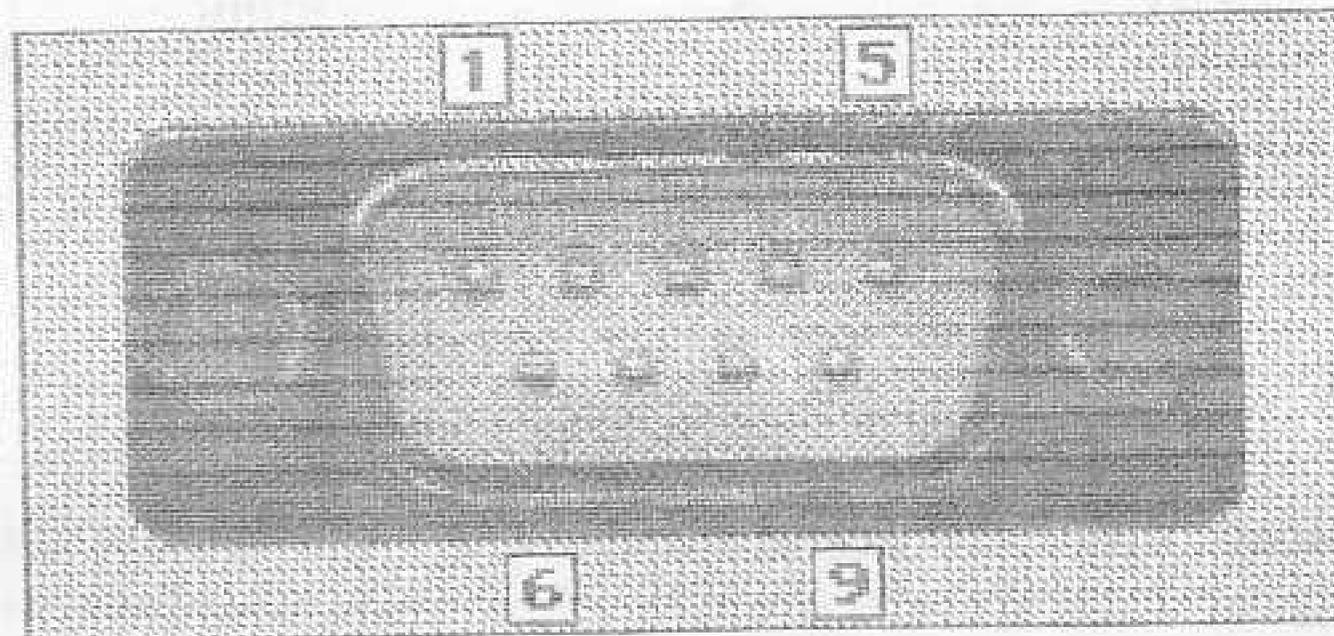
Hardware Overview

There are many different recommended standards of serial port communication, including the following most common types.

The RS-232 is a standard developed by the Electronic Industries Association (EIA) and other interested parties, specifying the serial interface between Data Terminal Equipment (DTE) and Data Communications Equipment (DCE). The RS-232 standard includes electrical signal characteristics (voltage levels), interface mechanical characteristics (connectors), functional description of interchange circuits (the function of each electrical signal), and some recipes for common kinds of terminal-to-modem connections. The most frequently encountered revision of this standard is called RS-232C. Parts of this standard have been adopted (with various degrees of fidelity) for use in serial communications between computers and printers, modems, and other equipment. The serial ports on standard IBM-compatible personal computers follow RS-232.

RS-232 Cabling

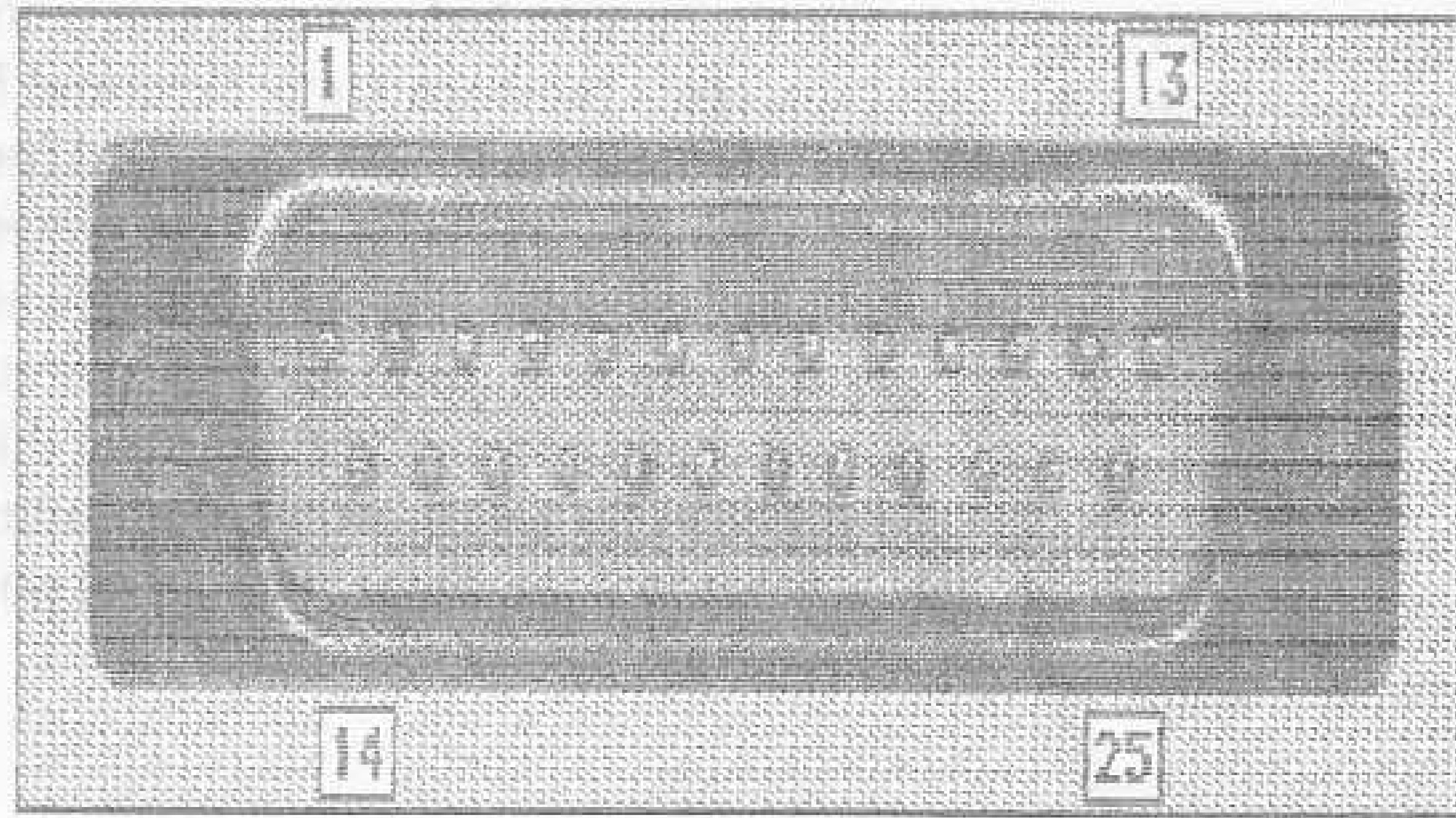
Devices that use serial cables for their communication are split into two categories. These are DCE and DTE. DCE are devices such as a modem, TA adapter, plotter, and so on, while DTE is a computer or terminal. RS-232 serial ports come in two sizes, the D-Type 25-pin connector and the D-Type 9-pin connector. Both of these connectors are male on the back of the PC. Thus, you require a female connector on the device. Table 1 shows the pin connections for the 9-pin and 25-pin D-Type connectors.



Function	Signal	PIN	DTE	DCE
Data	TxD	3	Output	Input
	RxD	2	Input	Output
Handshake	RTS	7	Output	Input
	CTS	8	Input	Output
	DSR	6	Input	Output
	DCD	1	Input	Output
	STR	4	Output	Input
Common	Com	5	--	--
Other	RI	9	Output	Input

The DB-9 connector is occasionally found on smaller RS-232 lab equipment. It is compact, yet has enough pins for the core set of serial pins (with one pin extra). The DB-25 connector is the standard RS-232 connector, with enough pins to cover all the signals specified in the standard. Table 2 shows only the core set of pins that are used for most RS-232 interfaces.

www.jntuworldupdates.org



Function	Signal	PIN	DTE	DCE
Data	TxD	2	Output	Input
	RxD	3	Input	Output
Handshake	RTS	4	Output	Input
	CTS	5	Input	Output
	DSR	6	Input	Output
	DCD	8	Input	Output
	STR	20	Output	Input
Common	Com	7	--	--

Universal Serial Bus

In information technology, Universal Serial Bus (USB) is a serial bus standard to connect devices to a host computer. USB was designed to allow many peripherals to be connected using a single standardized interface socket and to improve plug and play capabilities by allowing hot swapping; that is, by allowing devices to be connected and disconnected without rebooting the computer or turning off the device. Other convenient features include providing power to low-consumption devices, eliminating the need for an external power supply; and allowing many devices to be used without requiring manufacturer-specific device drivers to be installed.

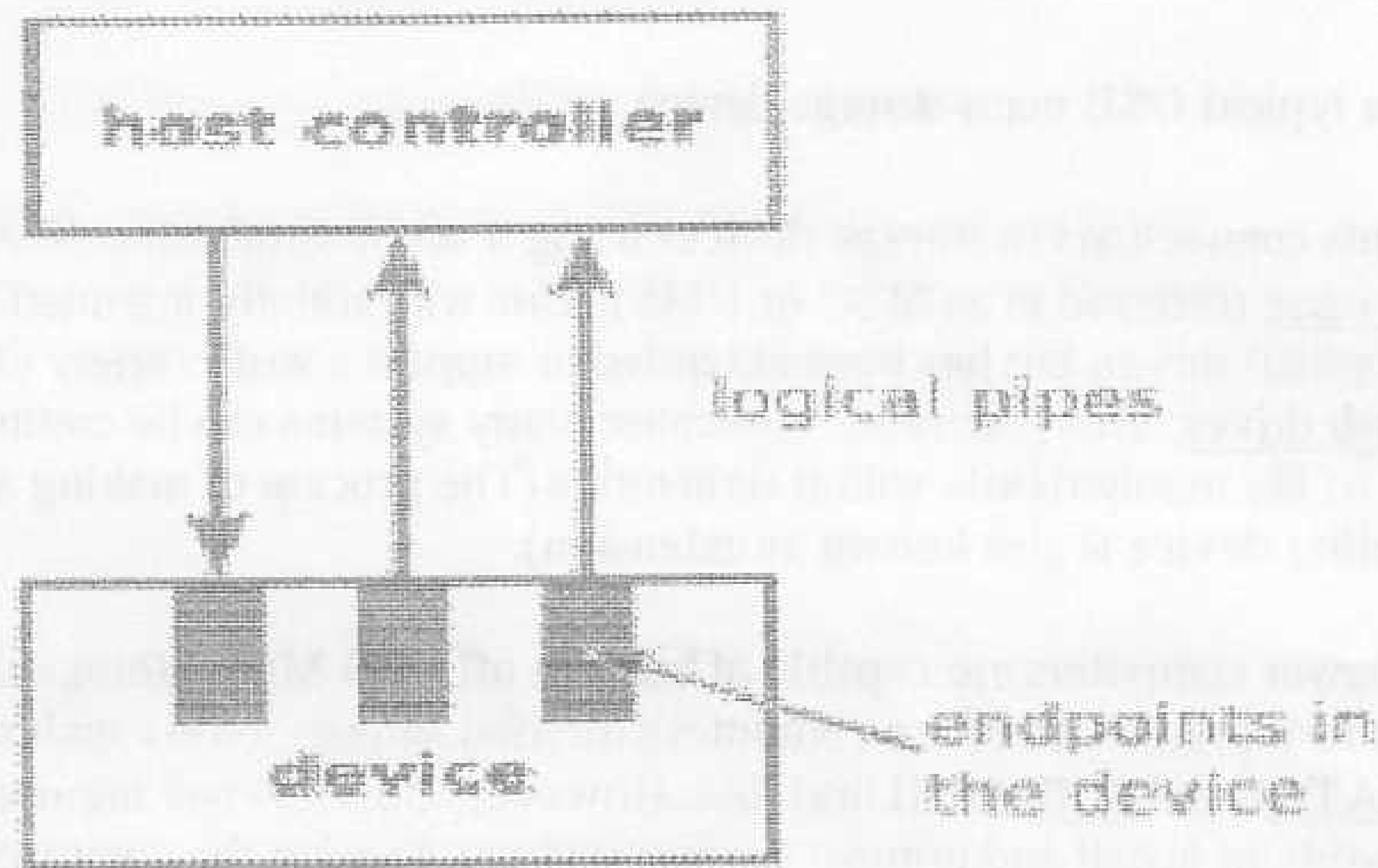
USB is intended to replace many varieties of serial and parallel ports. USB can connect computer peripherals such as mice, keyboards, PDAs, gamepads and joysticks, scanners, digital cameras, printers, personal media players, flash drives, and external hard drives. For many of those devices, USB has become the standard connection method. USB was designed for personal computers, but it has become commonplace on other devices such as PDAs and video game consoles, and as a power cord between a device and an AC adapter plugged into a wall plug for charging. As of 2008, there are about 2 billion USB devices sold per year, and about 6 billion total sold to date.[1]

The design of USB is standardized by the USB Implementers Forum (USB-IF), an industry standards body incorporating leading companies from the computer and electronics industries. Notable members have included Agere (now merged with LSI Corporation), Apple Inc., Hewlett-Packard, Intel, Microsoft and NEC.

A USB system has an asymmetric design, consisting of a host, a multitude of downstream USB ports, and multiple peripheral devices connected in a tiered-star topology. Additional USB hubs may be included in the tiers, allowing branching into a tree structure with up to five tier levels. A USB host may have multiple host controllers and each host controller may provide one or more USB ports. Up to 127 devices, including the hub devices, may be connected to a single host controller.

USB devices are linked in series through *hubs*. There always exists one hub known as the root hub, which is built into the host controller. So-called "sharing hubs", which allow multiple computers to access the same peripheral device(s), also exist and work by switching access between PCs, either automatically or manually. They are popular in small-office environments. In network terms, they converge rather than diverge branches.

A physical USB device may consist of several logical sub-devices that are referred to as *device functions*. A single device may provide several functions, for example, a webcam (video device function) with a built-in microphone (audio device function).



USB device communication is based on *pipes* (logical channels). Pipes are connections from the host controller to a logical entity on the device named an endpoint. The term *endpoint* is occasionally used to incorrectly refer to the pipe. A USB device can have up to 32 active pipes, 16 into the host controller and 16 out of the controller.

Each endpoint can transfer data in one direction only, either into or out of the device, so each pipe is uni-directional. Endpoints are grouped into *interfaces* and each interface is associated with a single device function. An exception to this is endpoint zero, which is used for device configuration and which is not associated with any interface.

When a USB device is first connected to a USB host, the USB device enumeration process is started. The enumeration starts by sending a reset signal to the USB device. The speed of the USB device is determined during the reset signaling. After reset, the USB device's information is read by the host, then the device is assigned a unique 7-bit address. If the device is supported by the host, the device drivers needed for communicating with the device are loaded and the device is set to a configured state. If the USB host is restarted, the enumeration process is repeated for all connected devices.

www.jntuworldupdates.org

implementation costs and a simplified, more adaptable cabling system. The 1394 standard also defines a backplane interface, though this is not as widely used.

IEEE 1394 has been adopted as the High-Definition Audio-Video Network Alliance (HANA) standard connection interface for A/V (audio/visual) component communication and control.[1] FireWire is also available in wireless, fiber optic, and coaxial versions using the isochronous protocols.

ireWire is Apple Inc.'s name for the IEEE 1394 High Speed Serial Bus. It was initiated by Apple (in 1986[2]) and developed by the IEEE P1394 Working Group, largely driven by contributions from Apple, although major contributions were also made by engineers from Texas Instruments, Sony, Digital Equipment Corporation, IBM, and INMOS/SGS Thomson (now STMicroelectronics).

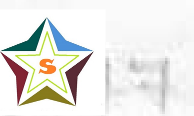
Apple intended FireWire to be a serial replacement for the parallel SCSI (Small Computer System Interface) bus while also providing connectivity for digital audio and video equipment. Apple's development began in the late 1980s, later presented to the IEEE,[3] and was completed in 1995. As of 2007, IEEE 1394 is a composite of four documents: the original IEEE Std. 1394-1995, the IEEE Std. 1394a-2000 amendment, the IEEE Std. 1394b-2002 amendment, and the IEEE Std. 1394c-2006 amendment. On June 12, 2008, all these amendments as well as errata and some technical updates were incorporated into a superseding standard IEEE Std. 1394-2008.[4]

Apple's internal code-name for FireWire was "Greyhound" as of May 11, 1992.

Sony's implementation of the system, known as "i.LINK" used a smaller connector with only the four signal circuits, omitting the two circuits which provide power to the device in favor of a separate power connector. This style was later added into the 1394a amendment.[3] This port is sometimes labeled "S100" or "S400" to indicate speed in Mbit/s.

The system is commonly used for connection of data storage devices and DV (digital video) cameras, but is also popular in industrial systems for machine vision and professional audio systems. It is preferred over the more common USB for its greater effective speed and power distribution capabilities, and because it does not need a computer host. Perhaps more important, FireWire makes full use of all SCSI capabilities and has high sustained data transfer rates, a feature especially important for audio and video editors. Benchmarks show that the sustained data transfer rates are higher for FireWire than for USB 2.0, especially on Apple Mac OS X with more varied results on Microsoft Windows.[5][6]

However, the royalty which Apple Inc. and other patent holders initially demanded from users of FireWire (US$0.25 per end-user system) and the more expensive hardware needed to implement it (US$1–$2), both of which have since been dropped, have prevented FireWire from displacing USB in low-end mass-market computer peripherals, where product cost is a major constraint.[3]


www.jntuworldupdates.org

implementation costs and a simplified, more adaptable cabling system. The 1394 standard also defines a backplane interface, though this is not as widely used.

IEEE 1394 has been adopted as the High-Definition Audio-Video Network Alliance (HANA) standard connection interface for A/V (audio/visual) component communication and control.[1] FireWire is also available in wireless, fiber optic, and coaxial versions using the isochronous protocols.

FireWire is Apple Inc.'s name for the IEEE 1394 High Speed Serial Bus. It was initiated by Apple (in 1986)[3] and developed by the IEEE P1394 Working Group, largely driven by contributions from Apple, although major contributions were also made by engineers from Texas Instruments, Sony, Digital Equipment Corporation, IBM, and INMOS/SGS Thomson (now STMicroelectronics).

Apple intended FireWire to be a serial replacement for the parallel SCSI (Small Computer System Interface) bus while also providing connectivity for digital audio and video equipment. Apple's development began in the late 1980s, later presented to the IEEE,[3] and was completed in 1995. As of 2007, IEEE 1394 is a composite of four documents: the original IEEE Std. 1394-1995, the IEEE Std. 1394a-2000 amendment, the IEEE Std. 1394b-2002 amendment, and the IEEE Std. 1394c-2006 amendment. On June 12, 2008, all these amendments as well as errata and some technical updates were incorporated into a superseding standard IEEE Std. 1394-2008.[4]

Apple's internal code-name for FireWire was "Greyhound" as of May 11, 1992.

Sony's implementation of the system, known as "i.LINK", used a smaller connector with only the four signal circuits, omitting the two circuits which provide power to the device in favor of a separate power connector. This style was later added into the 1394a amendment.[3] This port is sometimes labeled "S100" or "S400" to indicate speed in Mbit/s.

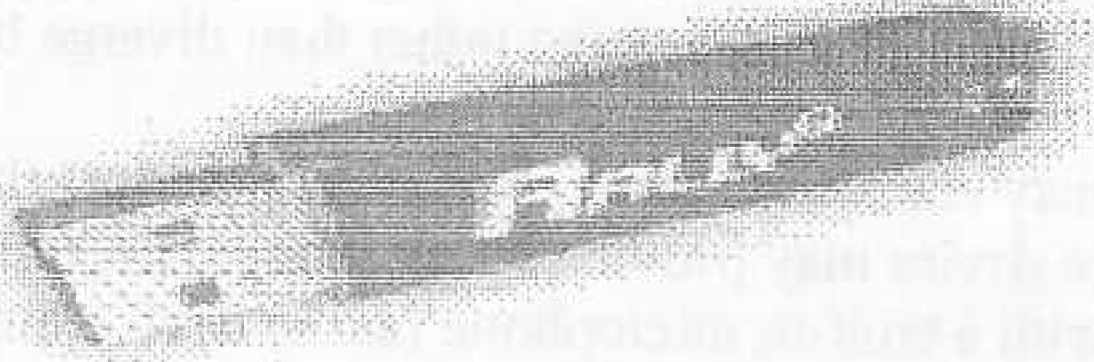
The system is commonly used for connection of data storage devices and DV (digital video) cameras, but is also popular in industrial systems for machine vision and professional audio systems. It is preferred over the more common USB for its greater effective speed and power distribution capabilities, and because it does not need a computer host. Perhaps more important, FireWire makes full use of all SCSI capabilities and has high sustained data transfer rates, a feature especially important for audio and video editors. Benchmarks show that the sustained data transfer rates are higher for FireWire than for USB 2.0, especially on Apple Mac OS X with more varied results on Microsoft Windows.[citation needed]

However, the royalty which Apple Inc. and other patent holders initially demanded from users of FireWire (US$0.25 per end-user system) and the more expensive hardware needed to implement it (US$1–$2), both of which have since been dropped, have prevented FireWire from displacing USB in low-end mass-market computer peripherals, where product cost is a major constraint.[3]

The host controller directs traffic flow to devices, so no USB device can transfer any data on the bus without an explicit request from the host controller. In USB 2.0, host controller polls the bus for traffic, usually in a round-robin fashion. In SuperSpeed USB, connected devices can request service from host.

USB mass-storage

Main article: USB mass storage device class



A flash drive, a typical USB mass-storage device.

USB implements connections to storage devices using a set of standards called the *USB mass storage device class* (referred to as MSC or UMS). This was initially intended for traditional magnetic and optical drives, but has been extended to support a wide variety of devices, particularly flash drives. This generality is because many systems can be controlled with the familiar idiom of file manipulation within directories (The process of making a novel device look like a familiar device is also known as extension).

Though most newer computers are capable of booting off USB Mass Storage devices, USB is not intended to be a primary bus for a computer's internal storage: buses such as ATA (IDE), Serial ATA (SATA), and SCSI fulfill that role. However, USB has one important advantage in that it is possible to install and remove devices without opening the computer case, making it useful for external drives. Originally conceived and still used today for optical storage devices (CD-RW drives, DVD drives, etc.), a number of manufacturers offer external portable USB hard drives, or empty enclosures for drives, that offer performance comparable to internal drives[citation needed]. These external drives usually contain a translating device that interfaces a drive of conventional technology (IDE, ATA, SATA, ATAPI, or even SCSI) to a USB port. Functionally, the drive appears to the user just like an internal drive. Other competing standards that allow for external connectivity are eSATA and FireWire.

Another use for USB Mass Storage devices is the portable execution of software applications without the need of installation on the host computer,[5] such as Web Browsers, VoIP clients,[6] etc.

IEEE 1394

The **IEEE 1394 interface** is a serial bus interface standard for high-speed communications and isochronous real-time data transfer, frequently used by personal computers, as well as in digital audio, digital video, automotive, and aeronautics applications. The interface is also known by the brand names of **FireWire** (Apple Inc.), **i.LINK** (Sony), and **Lynx** (Texas Instruments). IEEE 1394 replaced parallel SCSI in many applications, because of lower

UNIT VII

KK

Parallel - processing :-

parallel - processing is a term used to denote a large class of technique that are used to provide simultaneous data - processing tasks.

A parallel processing system is able to perform concurrent data processing to achieve faster execution time.

For eg, while an instruction is being executed in the ALU, the next instruction can be read from memory. The system may have two or more ALUs & be able to execute two or more instructions at the same time.

The purpose of parallel processing is to speed up the computer processing capability & increase its throughput. The throughput is defined as the amount of processing that can be accomplished during a given interval of time.

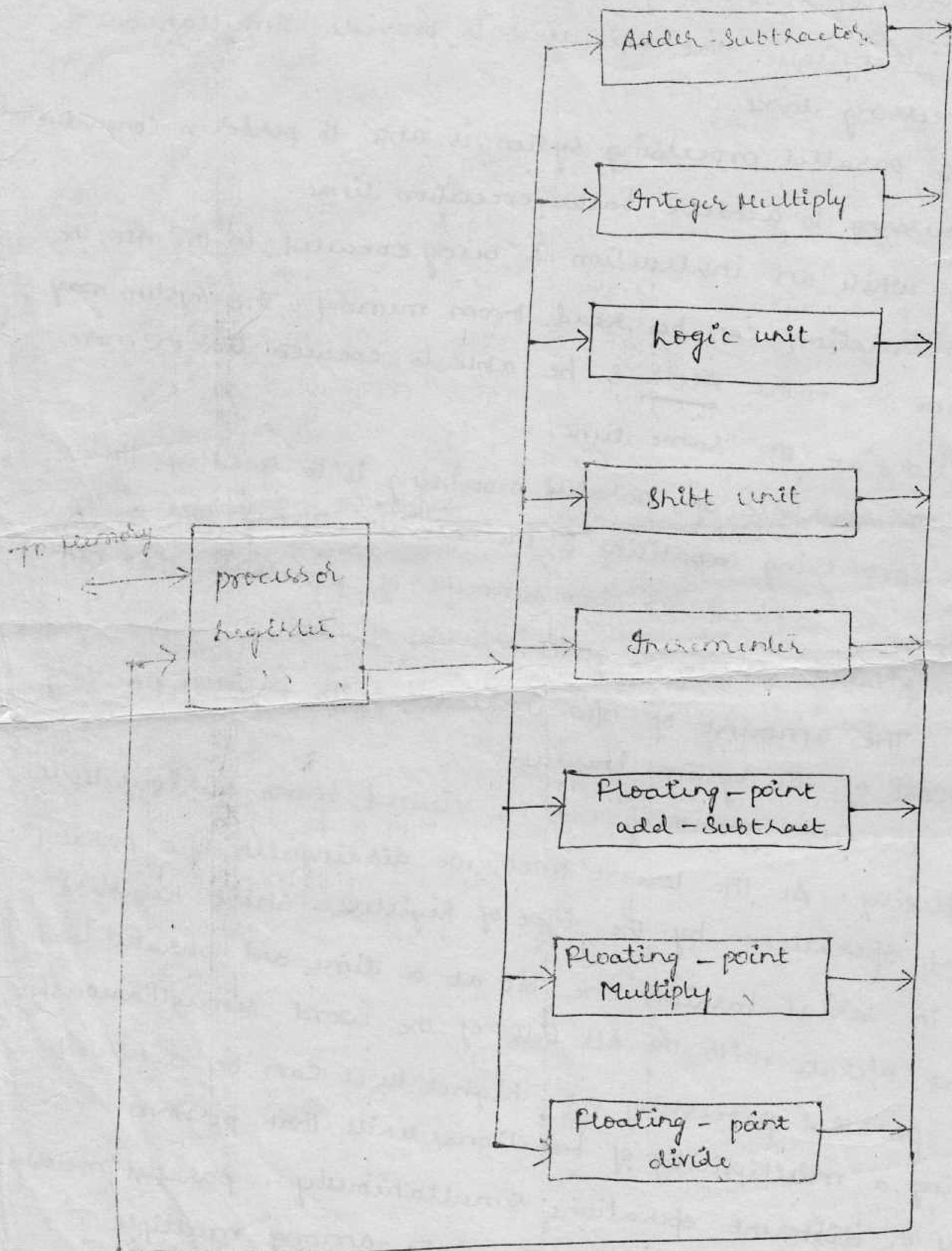
The amount of h/w increases with parallel processing & the cost of the system increases.

parallel processing can be viewed from various levels of complexity. At the lowest level, we distinguish b/w parallel & serial operations by the type of registers. shift registers operate in serial fashion one bit at a time and parallel load registers operate with the all bits of the word simultaneously.

parallel processing at higher level can be achieved by having a multiplicity of functional units that perform identical or different operations simultaneously. parallel processing is established by distributing the data among multiple functional units.

The fig. shows one possible way of separating

execution unit into eight functional units operating in



The operands in the registers are applied to one of the units depending on the operation specified by the instruction association with the operands.

www.jntuworldupdates.org

The adder & integer multiplier perform the arithmetic operations with integer numbers. The floating-point operations are separated into three circuits operating in parallel. The logic, shift & increment operations can be performed concurrently on different data. All units are independent of each other, so one number can be shifted while another number is being incremented. A multifunctional organization is usually associated with a complex control unit to coordinate all the activities among various components.

There are a variety of ways that parallel processing can be classified. It can be considered as internal organization of the processors, from the interconnection structure b/w processors or from the flow of information through the system.

One classification which introduced by M.T. Flynn, considers the organization of a computer systems by the no. of instructions & data items that are manipulated.

The sequence of instruction read from memory constitutes an instruction stream. The operations performed on the data in the processor constitutes a data stream.

Flynn's classification divides computers into four major groups as follows:

SISD (Single instruction stream, single data stream)
SIMD (Single instruction stream, multiple data stream)
MISD (Multiple instruction stream, single data stream)
MIMD (Multiple instruction stream, multiple data stream).

SISD represents the organization of a single computer containing a control unit, a processor unit & a memory unit

Instructions are executed sequentially & the system may / mayn't have internal parallel processing

SIMD represents an organization that includes many processing units under the supervision of a common control unit. All processors receive the same instruction from the control unit but operate on different items of data. The shared memory units must contain multiple modules so that it can with all the processors simultaneously.

MISD structure is only of theoretical interest but there is no practical system.

MIMD organization refers to a computer system capable of processing several programs at the same time.

Pipelining :-

pipelining is a technique of decomposing a sequential process into suboperations, with each subprocess being executed in a special dedicated segment that operates concurrently with all other segment.

Each segment performs partial processing in which the task is partitioned. The result obtained from the computation in each segment is transferred to the next segment in the pipeline. The final result is obtained after the data have passed through all segments.

Its characteristic of p is that several computations can be in progress in distinct segments at the same time. The overlapping is also possible, by associating a register with each segment in the pipeline.

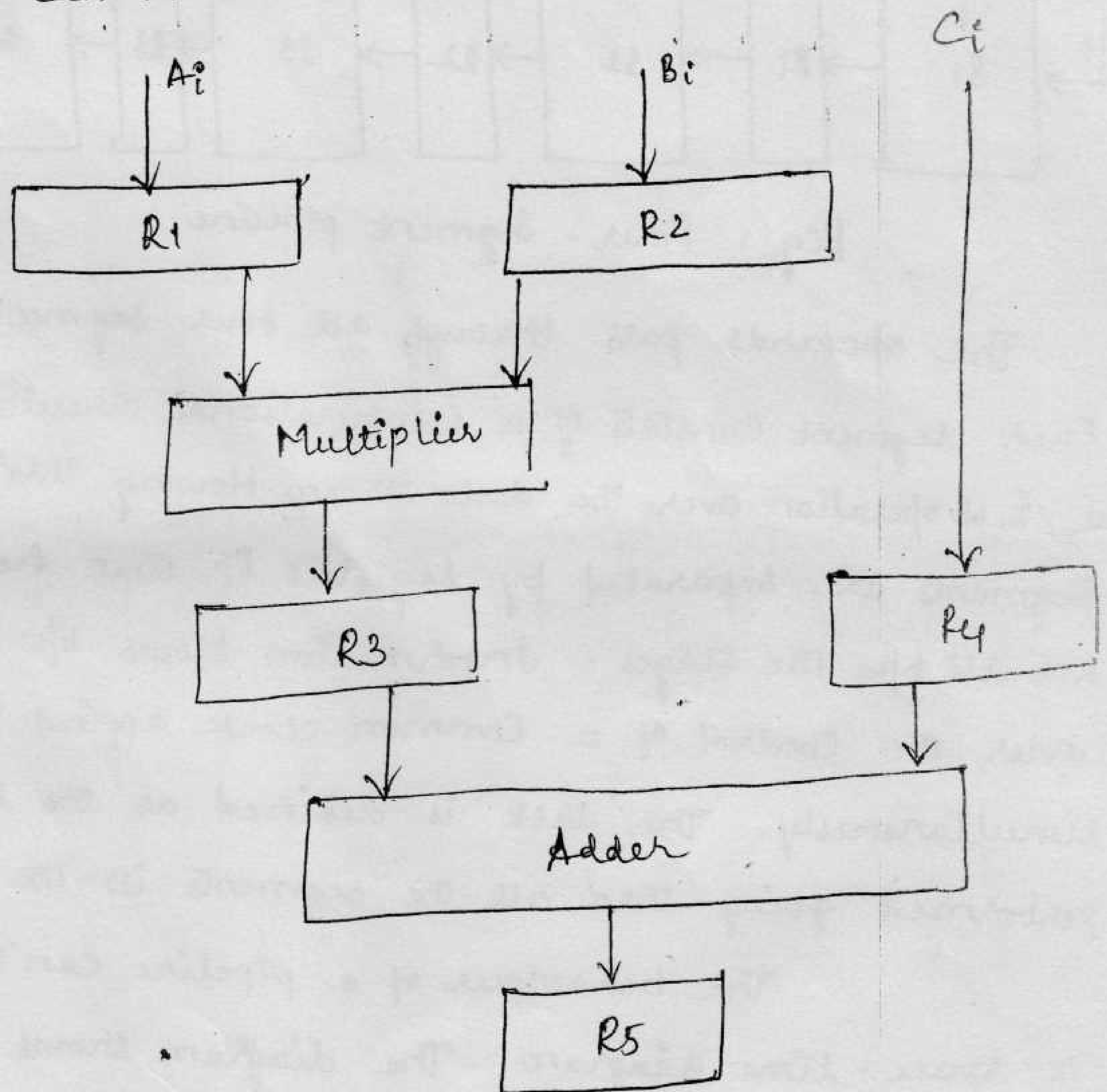
The simplest way is to implement pipeline is to imagine that each segment consists of an i/p register followed by a combinational circuit. The register holds the data & the

in a given segment is applied to the i/p reg. of the next segment. A clock is applied to all registers. In this way, the information flows through the pipeline one step at a time

Eg: Suppose that want to perform the combined multiply & add operations with a stream of numbers.

$$A_i * B_i + C_i \quad \text{for } i = 1 \text{ to } 7.$$

This can be shown below:



The registers R1 through R5 that receive new data with every clock pulse. The multiplier & adder are combinational circuits. The suboperations performed in each segment of the pipeline as follows:

$R1 \leftarrow A_i$, $R2 \leftarrow B_i$	I/p A_i & B_i
$R3 \leftarrow R1 * R2$, $R4 \leftarrow C_i$	Multiply & i/p C_i
$R5 \leftarrow R3 + R4$	Add C_i to product

www.jntuworldupdates.org

~~The~~

General Considerations :-

The general structure of a four-segment pipeline is shown below.

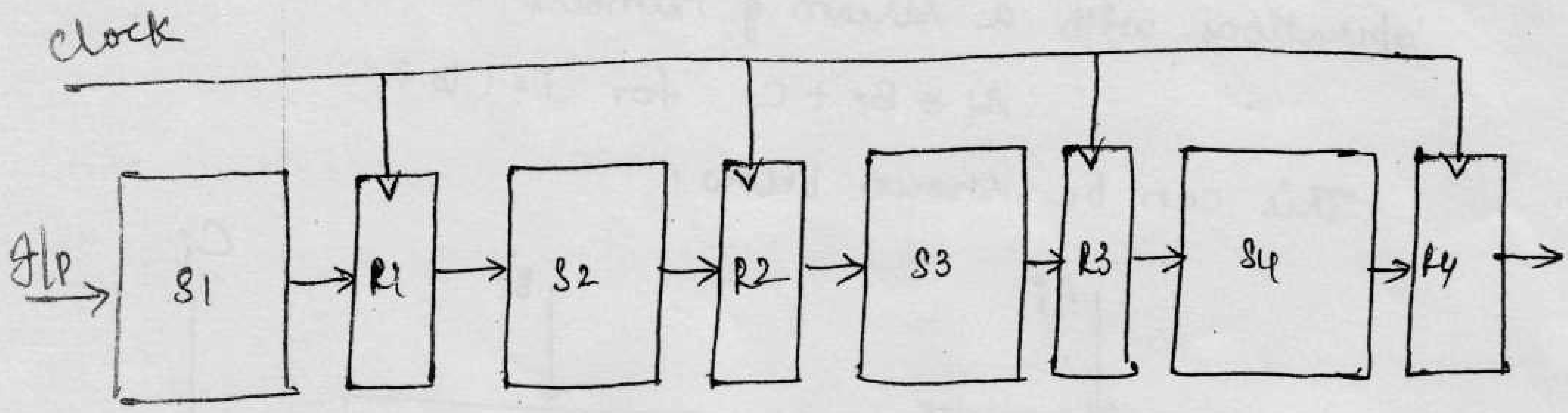


Fig : Four - Segment pipeline .

The operands pass through all four segments in a fixed sequence. Each segment consists of a combinational circuit S_i that performs a suboperation over the data stream flowing thro' the pipe. The segments are separated by registers R_i that holds the intermediate results b/w the stages. Information flows b/w adjacent stages under the control of a common clock applied to all registers simultaneously. The 'task' is defined as the total operation performed going thro' all the segments in the pipeline.

The behaviour of a pipeline can be illustrated with a space - time diagram. The diagram shows the segment utilization as a function of time. This can be shown in below fig.

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

The diagram shows six Tasks T_1 through T_6 executed in

www.jntuworldupdates.org

is four segments. Initially, task T1 is handled by segment 1. After the first clock, segment 2 is busy with T_1, while segment 1 is busy with task T_2. Continuing this manner, the first task T_1 is completed after the fourth clock cycle. Once the pipeline is full, it takes only one clock period to obtain an o/p.

Now consider the case where a K-segment pipeline with a clock cycle time t_p is used to execute 'n' tasks. The first task T_1 requires a time equal to Kt_p to complete its operation. The remaining (n-1) tasks emerge from the pipe at the rate of one task per clock cycle & they will be completed after a time equal to $(n-1)t_p$. Therefore, to complete 'n' tasks using a K-segment pipeline requires $K+(n-1)$ clock cycles.

Consider a Non pipeline unit that performs the same operation & takes a time equal to t_n to complete each task. The total time required for 'n' tasks is nt_n. The speedup of a pipeline processing over an equivalent non pipeline processing is defined by the ratio,

$$S = \frac{nt_n}{(K+n-1)t_p}$$

As the no. of tasks increases, n becomes much larger than K-1, & K+n-1 approaches the value of n. under this condition, the speed up becomes

$$S = \frac{t_n}{t_p}$$

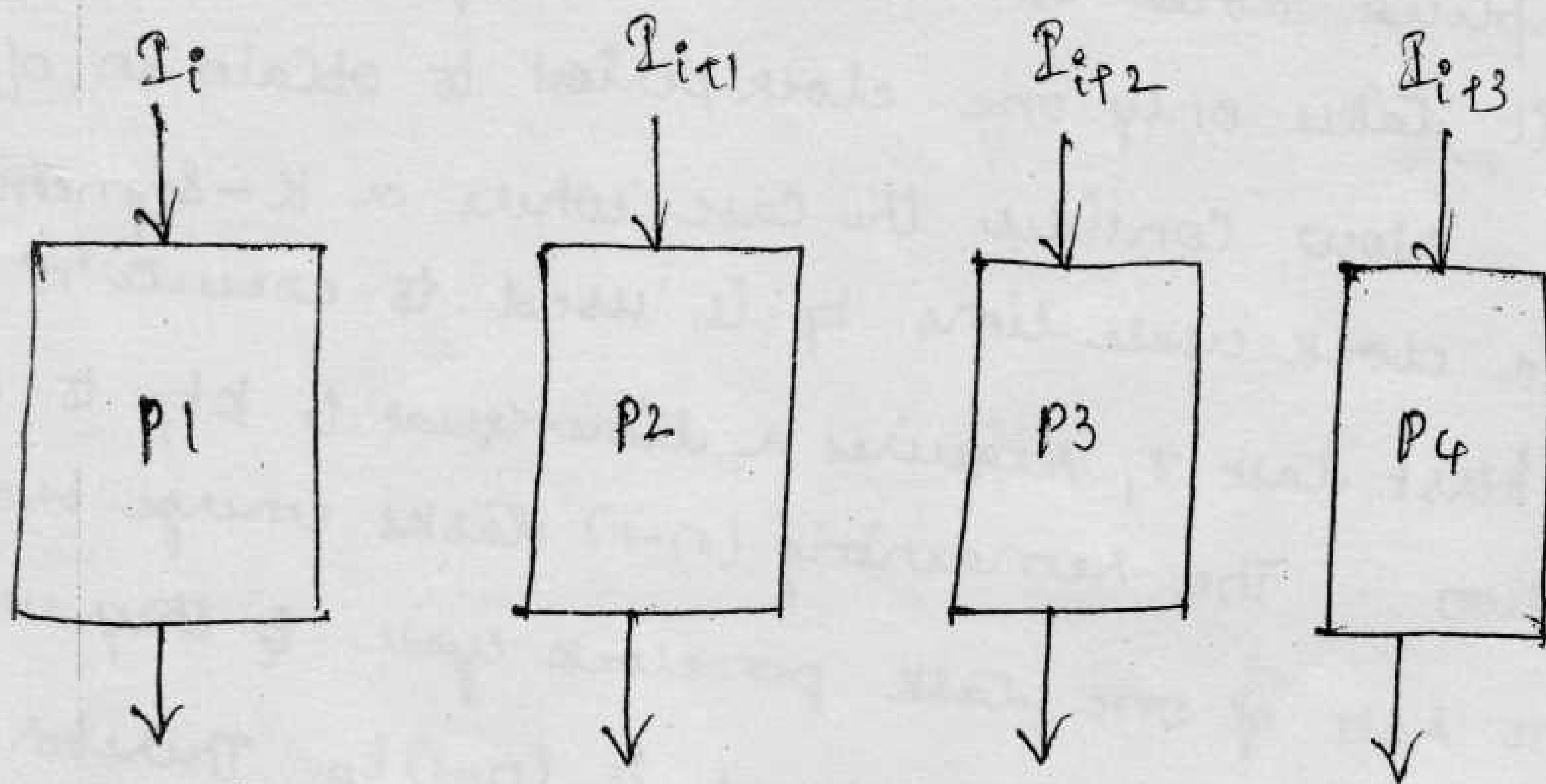
We assume that the time it takes to process a task is the same in the pipeline & non-pipeline circuits, we have $t_n = Kt_p$. The speedup reduces to

$$S = \frac{Kt_p}{t_p} = K.$$

www.specworld.in

www.jntuworldupdates.org

The speed advantage of a pipeline process by means of multiple functional units, that will be operating in parallel. This is illustrated in below fig.



Each 'p' circuit performs the same task of an equivalent pipeline circuit. The parallel i/p's accepts the data simultaneously & perform four tasks at the same time.

The above fig. shows SIMD, in this also same instruction is used to operate on multiple data in parallel.

pipeline can't operate for many reasons. It has maximum theoretical rate we aren't always going to be considered that the nonpipe circuit has the same time delay as that of pipeline circuit. pipeline provides a faster operation over a purely serial sequence.

~~Arithmatic~~ pipelining:-

pipeline organization can be applicable in two areas 1) Arithmetic pipeline and 2) Instruction pipeline

An arithmetic pipeline divides an arithmetic operation into sub operations for execution in the pipeline segments

An instruction pipeline operates on a stream of instructions by overlapping the fetch, decode & execute phases of the instruction cycle.

The pipeline arithmetic units are usually found in very high speed computers. They are used to implement floating-point operations, multiplication of fixed-point numbers.

The floating-point operations are easily decomposed into sub-operations.

The i/p's to the floating-point adder pipeline are two normalized floating-point binary numbers

$$X = A \times 2^a, \quad Y = B \times 2^b.$$

A & B are two fractions that represent the mantissas & a & b are the exponents.

The floating-point addition & subtraction can be performed in four segments which is shown in fig. The registers labeled 'R' are placed b/w the segments to store intermediate results. The suboperations that are performed in four segments are:

1) compare the exponents
2) Align the mantissas
3) Add or Subtract the mantissas
4) Normalize the result.

The exponents are compared by subtracting them to their difference. The larger exponent is chosen as the exponent of the result. The exponent difference determines how many times the mantissa associated with the smaller exponent must be shifted to the right.

The result is normalized in segment 4. when an Overflow occurs, the mantissa of the sum or difference is shifted right & the exponent incremented by one.

If an underflow occurs, the no of leading

www.jntuworldupdates.org

Exponents

a b

R

Segment 1: Compare exponents by Subtraction — Difference

R

Segment 2: Choose exponent

Segment 3:

R

Segment 4: Adjust Exponent

R

Mantissas

A B

R

Align Mantissas

R

Add or Subtract Mantissas

R

Normalize Result

R

mantissa & the number that must be subtracted from the exponent.

Eg: $X = 0.9504 \times 10^3$, $Y = 0.8200 \times 10^2$.

The two exponents are subtracted 3-2 = 1. The larger exponent 3 is chosen as the exponent of the result. The next segment shifts the mantissa of Y to the right to obtain

$X = 0.9504 \times 10^3$, $Y = 0.0820 \times 10^3$.

The exponents are same, then the addition of the two mantissas in segment 3 produces the sum

$Z = 1.0324 \times 10^3$.

The sum is adjusting by normalizing the result

$Z = 0.10324 \times 10^4$.

This is done by the shifting the mantissa once to the right & incrementing the exponent by one to obtain the normalized sum.

Instruction pipeline :-

An instruction pipeline reads consecutive locations from memory while previous instructions are being executed in other segments. This causes the instruction fetch & execute phases to overlap and performs simultaneous operation.

Consider a computer with an instruction fetch unit & an instruction execution unit designed to provide a two-segment pipeline. The instruction fetch segment can be implemented by means of a first-in, FIFO buffer. This unit uses queue. When ever the execution unit is not using memory, the control increments the PC & uses its address value to read consecutive

www.jntuworldupdates.org

instructions from memory. So, that the instructions are executed on the FIFO. Thus an instruction stream can be placed in a queue, waiting for decoding & processing by the execution segment. This means reduce the average access time to memory for reading instructions and it works in a efficient way. when ever there is space in the FIFO buffer, the control unit initiates the next instruction fetch phase. The buffer acts as a queue from which control then extracts the instruction for the execution unit.

Computers with complex instructions require other phases in addition to fetch & execute phases. The following steps are

1) Fetch the instruction from memory.
2) Decode the instruction
3) Calculate the effective address.
4) Fetch the operands from memory.
5) Execute the instruction.
6) Store the result

The instruction pipeline can prevent from difficult at its maximum rate. Different segments may take different times to operate on the incoming information. Some segments are skipped for certain operations.

For eg, A register mode instruction does n't need in effective address but remaining other segments may require it. The instruction must wait until another one is finished. So, Memory access conflicts are resolved by using two memory buses for accessing instructions & data in separate modules

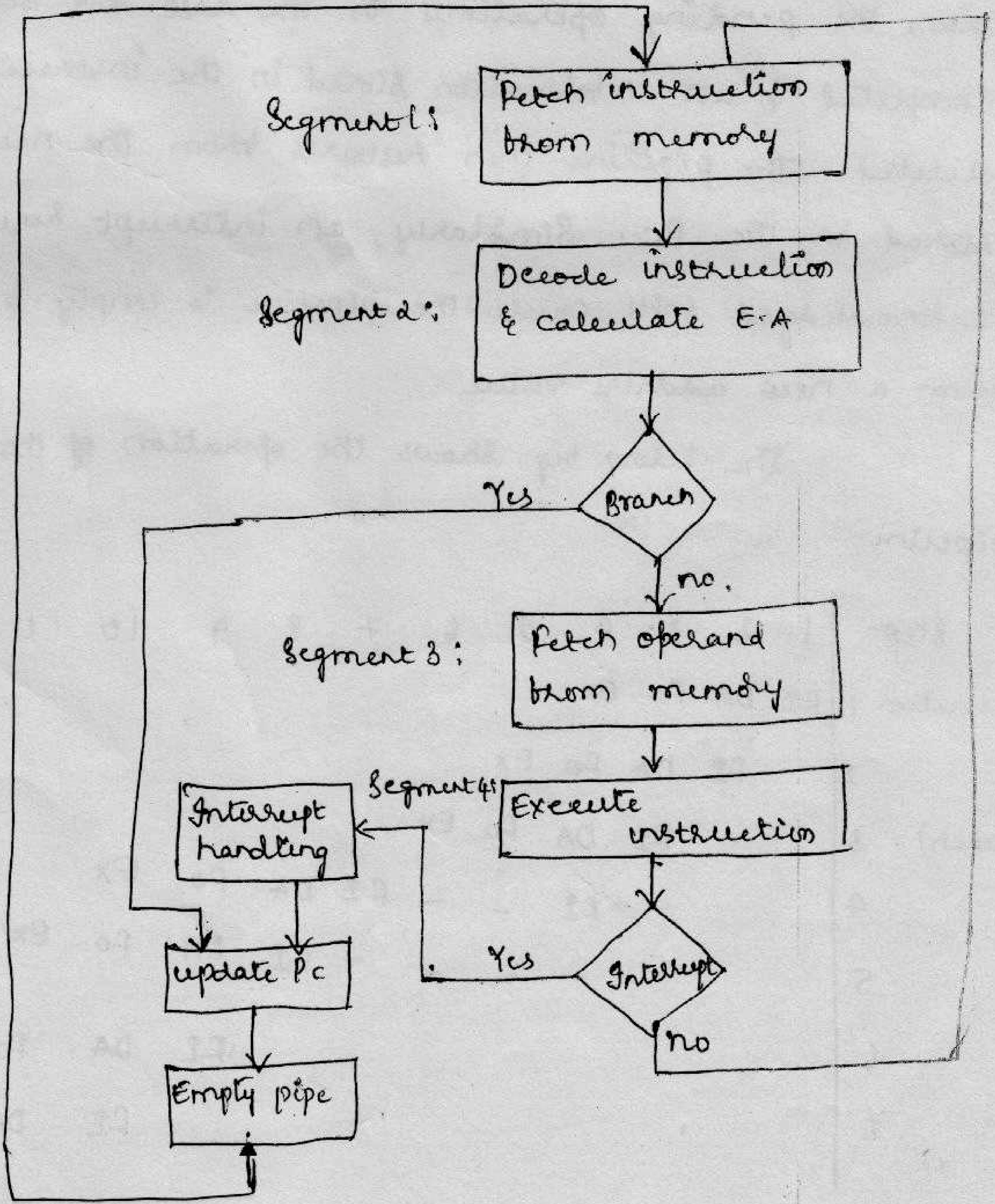
www.jntuworldupdates.org

segments of equal duration. The time that each step takes to fulfill its function depends on the instruction & the way it is executed.

Four-segment Instruction pipeline :-

Consider that the decoding of the instruction can be combined with the calculation of effective address into one segment. The instruction execution & storing of the result can be combined into one segment. This reduces the instruction pipeline into four segments.

The below fig. shows the instruction cycle in the CPU can be proceed with a four-segment pipeline.



www.smartzworld.com

www.jntuworldupdates.org

While an instruction is being executed in segment 4, the next instruction in sequence is busy fetching an operand from memory in segment 3. The effective address may be calculated in a separate arithmetic circuit for the third instruction, and whenever the memory is available, the fourth & all subsequent instructions can be fetched & placed in an instruction in FIFO. four sub operations in the instruction cycle can overlap and upto four different instructions can be in progress of being processed at the same time.

An instruction in the sequence may be a program control type that causes a branch out of normal sequence. In that case, the pending operations in the last two segments are completed & all information stored in the instruction buffer is deleted. The pipeline then restarts from the new address stored in the PC. Similarly, an interrupt request, when acknowledged will cause the pipeline to empty & start again from a new address value.

The below fig. shows the operation of the instruction pipeline.

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

www.jntuworldupdates.org

equal duration. The four segments are represented in the diagram with an abbreviated symbol.

1) FI is the segment that fetches an instruction
2) DA is the segment that decodes the instruction & calculates the effective address.
3) FO is the segment that fetches the operand
4) EX is the segment that Executes the instruction

pipeline conflicts :-

The three major difficulties that cause the instruction pipeline to deviate from its normal operation.

1) Resource conflicts :-
It caused by access to memory by two segments at the same time. Most of these conflicts can resolved by using separate instruction & data memories.

2) Data dependency :-
This conflicts arise when an instruction depends on the result of a previous instruction

3) Branch difficulties :-
This arise from branch & other instructions that change the value of PC.

Data dependency :-

This difficulty may caused a degradation of performance in an instruction pipeline is due to possible collision of data or address. A collision occurs when an instruction can't proceed because previous instructions didn't complete certain operations. A data dependency occurs when an instruction need data that not available.

Similarly, an address dependency may occur when an operand

www.jntuworldupdates.org

address can't be calculated

Hardware interlocks :-

This is a most straight forward method. An interlock is a circuit that detects instructions whose source operands are destinations of instructions.

Detection of this situation causes the instruction whose source isn't available to be delayed by enough clock cycles to resolve the conflict. This approach maintains the program sequence by using h/w to insert the required delay.

operand forwarding :-

This uses a special h/w to detect a conflict and then avoid it by routing the data through special paths b/w pipeline segments.

For eg, Instead of transferring an ALU result into a destination register, the h/w checks the destination operand and if it is need as source in the next instruction, it pass the result directly into the ALU i/p, by passing the register file. This method requires additional h/w paths through multiplexers as well as the circuit that detects the conflict.

Delayed load :-

The responsibility for solving data conflicts problems to the compiler that translates the high-level programming language into a machine language program. The compiler for such computer is designed to detect a data conflict & reorder the instructions as necessary to delay the loading of the conflicting data by inserting no-operations instructions. This method is referred to as 'delayed load'.

www.jntuworldupdates.org

One of the major problems in an instruction pipeline is branching. A branch instruction can be conditional or unconditional. An unconditional branches always the alters the sequential program flow by loading the program counter with the target address.

In an conditional branch, the control selects the target instruction if the condition is satisfied or the next sequential instruction if the condition is n't satisfied.

- prefetch target instructions:-

This is one kind of handling conditional branch. In this, if the branch condition is successful, the pipeline continues from the branch target instruction. An extension of this procedure is to continue fetching instruction from both places until the branch decision is made.

- Branch target Buffer :- (BTB)

Each entry in the BTB consists of the address of a previously executed branch instruction & target instruction for that branch. It also stores the next few instructions after the branch target instruction. When the pipeline decodes a branch instruction, it searches the associative memory BTB for the address of the instruction. If it is in the BTB, the instruction is available directly & prefetch continues from the new path. If it isn't in the BTB, the pipeline shifts to a new instruction stream & stores the target instruction in the BTB.

The advantage of this scheme is that branch instructions that have occured previously are readily available in the pipeline with out interruption.

www.jntuworldupdates.org

Loop buffer :-

A variation of BTB is loop buffer. When a program loop is detected in the program, it is stored in the loop buffer. The program loop can be executed directly with out having to access memory until the loop mode is removed by the final branching out.

Branch prediction :-

A pipeline with branch instruction uses some additional logic to guess the outcome of a conditional branch instruction before it is executed. The pipeline then begins prefetching the instruction stream from the predicted path. A correct prediction eliminates the wasted time caused by branch penalties.

Delayed branch :-

In this procedure, the compiler detects the branch instructions & rearranges the machine language code sequence by inserting useful instructions that keep the pipeline operating with out interruptions.

RISC Pipeline :-

RISC has ability to use an instruction pipeline in an efficient manner. The simplicity of the instruction set can be utilized to implement an instruction pipeline using a small no. of sub operations with each being executed in one clock cycle. The instruction format is of fixed length, so, the decoding of the operation can occur at the same time as the register selection. All data manipulation instructions have register-to-register operations. All operands are in registers, there is no need for

www.specworld.in

www.smartzworld.com

www.jntuworldupdates.org

Calculating E.A. The instruction pipeline can be implemented with two or three segments. One segment fetches the instruction from program memory and the other segment executes the instruction in the ALU. A third segment may be used to store the result of the ALU operation in a destination register.

The data transfer instructions in RISC are limited to load & store instructions. These instructions use register indirect addressing. They usually need three or four stages in the pipeline. To prevent conflicts b/w a memory access to fetch an instruction & to load or store an operand, the RISC provide two separate buses with two memories.

1) For storing the instructions

2) For storing the data

The two memories can some time operate at the same speed as the CPU clock & are refered to as cache memories.

The Major advantages of RISC is, it has ability to execute instructions at the rate of one per clock cycle. The main goal is to achieve a single - cycle instruction execution.

The advantage of RISC over CISC is that RISC can achieve pipeline segments, requiring just one clock cycle, while uses many segments in the pipeline, with the longest segment requiring two or more clock cycles

The RISC support another characteristic, i.e., compiler that translates the HLL into Machine language program

The following examples show how a compiler can optimize the machine language program to compensate for pipeline conflicts

www.jntuworldupdates.org

Three Segment Instruction pipeline :

The data Manipulation instructions operate on data in processor registers. The data transfer instructions are load and store instructions that use an effective address obtained from the addition of two registers or a register and a displacement constant provided in the instruction. The program control instructions use register values & a constant to evaluate the branch address, which is transferred to a register or the program counter PC.

Consider the h/w operation for such a computer. The control fetches the instruction from program memory into an instruction register. The instruction is decoded at the same time that the registers needed for the execution of the instruction are selected. The processor unit consists of a no. of registers & an ALU that performs the necessary arithmetic logic & shift operations. A data memory is used to load or store the data from a selected register in the register file. The instruction cycle can be divided into three suboperations & implemented in three segments.

I : Instruction fetch

A : ALU operation

E : Execute operation

The I segment fetches the instruction from program memory. The instruction is decoded & an ALU operation in the A segment. The ALU is used for three different functions depending on the decoded instruction. It performs an operation for a data manipulation instruction, it evaluates the effective address for a load or store instruction or it calculates the branch

address for a program control instruction. The E segment directs the output of the ALU to one of three destinations depending on the decoded instruction. It transfers the result of the ALU operation into a destination register in the register file, it transfers the effective address to a data memory for loading or storing, it transfers the branch address to the program counter.

Delayed Load :-

Consider the operations of the following four operations

1. LOAD : R1 ← M[address 1]
2. LOAD : R2 ← M[address 2]
3. ADD : R3 ← R1 + R2
4. STORE : M[address 3] ← R3.

If the three-segment pipeline proceeds with out interruptions, there will be a data conflict in instruction 3 because the operand in R2 is not available in the 'A' segment. This can be shown in the timing of the pipeline.

clock cycles	1	2	3	4.	5	6
1. Load R1	I	A	E			
2. Load R2		I	A	E		
3. Add R1+R2			I	A	E	
4. Store R3				I	A	E

The E segment in clock cycle 4 is in a process of placing the memory data into R2. The A segment in clock cycle 4 is using the data from R2, but the value in R2 will not be the correct value because it hasn't been transferred

www.smartzworld.com

www.jntuworldupdates.org

from memory. It is up to the compiler to make sure that the instruction following the load instruction uses the data fetched from memory. If the compiler can't find a useful instruction to put after the load, it inserts a no-operation instruction.

This is a type of instruction that is fetched from memory but has no operation, thus wasting a clock cycle. The concept of delaying the use of the data loaded from memory is referred to as delayed load.

The below shows the program with no-operation instruction inserted after the load to R2 instruction.

clock cycle	1	2	3	4	5	6	7
1. Load R1	I	A	E				
2. Load R2		I	A	E			
3. No-operation			I	A	E		
4. Add R1+R2				I	A	E	
5. Store R3					I	A	E

The data is loaded into R2 in clock cycle 4. The add instruction uses the value of R2 in step 5. Thus the no-op instruction is used to advance one clock cycle inorder to compensate for the data conflict in the pipeline.

The advantage of delayed load approach is that the data dependency is taken care by the compiler rather than h/w. This results in a simpler h/w segment since the segment doesn't have to check if the content of the register being accessed is currently valid or not.

www.jntuworldupdates.org

The Method used in most RISC processors is to rely on the compiler to redefine the branches so that they effect at the proper time in the pipeline. This method is referred to as delayed branch.

The compiler for a processor that uses delayed branches is designed to analyse the instructions before & after the branch & rearrange the program sequence by inserting useful instructions in the delay steps.

An example of delayed branch is shown in fig. The program for this, consists of 5 instructions.

Load from memory to R1

Increment R2.

Add R3 to R4

Subtract R5 from R6.

Branch to address X.

clock cycles :	1	2	3	4	5	6	7	8	9	10
1. Load	I	A	E							
Increment		I	A	E						
Add			I	A	E					
Subtract				I	A	E				
Branch to X					I	A	E			
No-operation						I	A	E		
No-operation							I	A	E	
Instruction in X								I	A	E

Fig. (a) No-operation instructions.

www.smartzworld.com

Clock cycles	1	2	3	4	5	6	7	8
1. Load	I	A	E					
Increment		I	A	E				
Branch to X			I	A	E			
Add				I	A	E		
Subtract					I	A	E	
Instruction in X						I	A	E

Fig.(b) Re-arranging the instructions

Array processors :-

An array processor is a processor that performs computations on large arrays of data.

This array processor is divided into two types

1) Attached Array processor
2) SIMD Array processor.

An attached array processor is an auxiliary processor attached to a general-purpose computer. It is intended to improve the performance of the host computer in specific numerical computation tasks.

An SIMD array processor is a processor that has a single-instruction multiple-data organization. It manipulates vector instructions by means of multiple functional units.

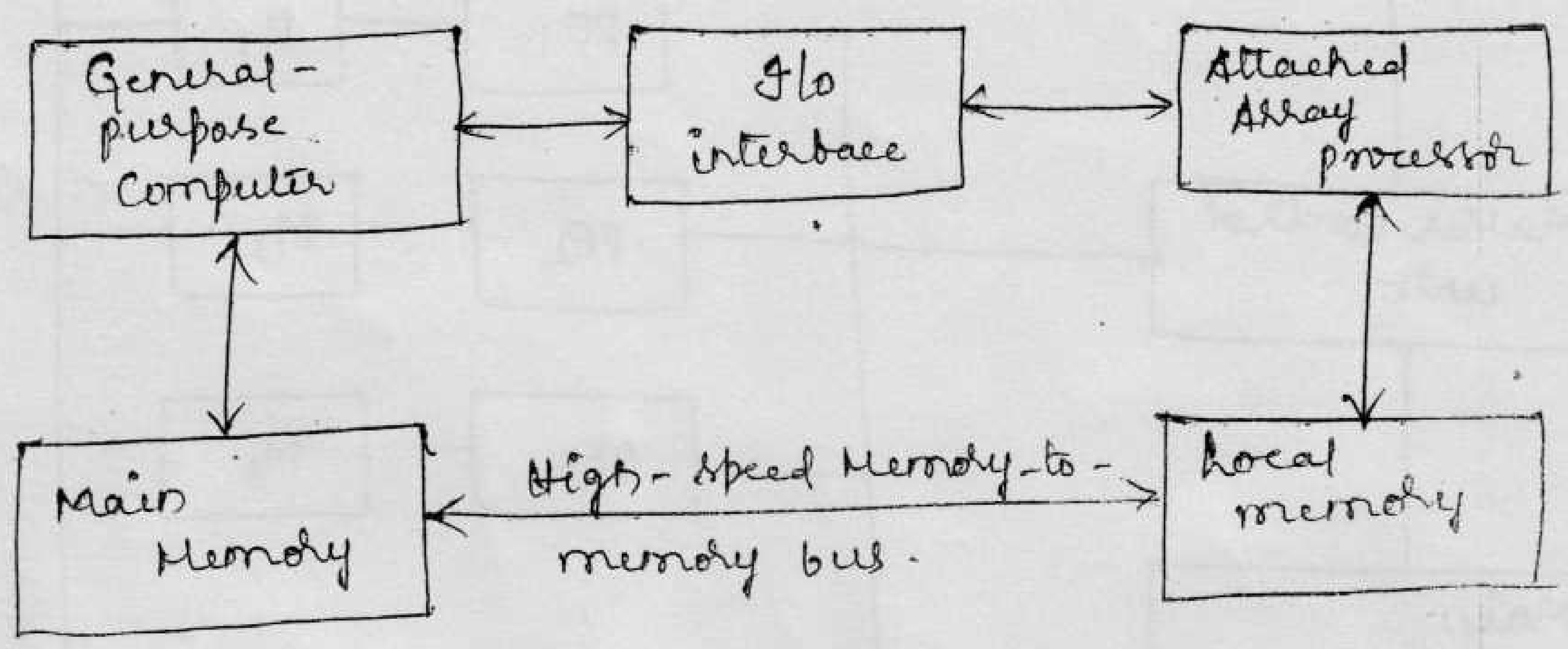
Attached Array processor :-

It is designed for a peripheral for a conventional host computer & its purpose to enhance the performance of the host computer by providing vector processing for complex scientific

www.jntuworldupdates.org

unit containing one or more pipelined floating-point adders & multipliers. The array processor can be programmed by the user to accommodate a variety of complex arithmetic problems.

The below fig. shows the interconnection of an attached array processor to a host computer.



The host computer is a general-purpose commercial computer & the attached processor is a back-end machine driven by the host computer.

The array processor is connected thro' an i/p & o/p controller of the computer & the computer treats it like an external interface. The data for the attached processor are transferred from main memory to a local memory through a high-speed bus. The general purpose computer with out the attached processor serves the user that need conventional data processing. The system with the attached processor satisfies the needs for complex arithmetic applications.

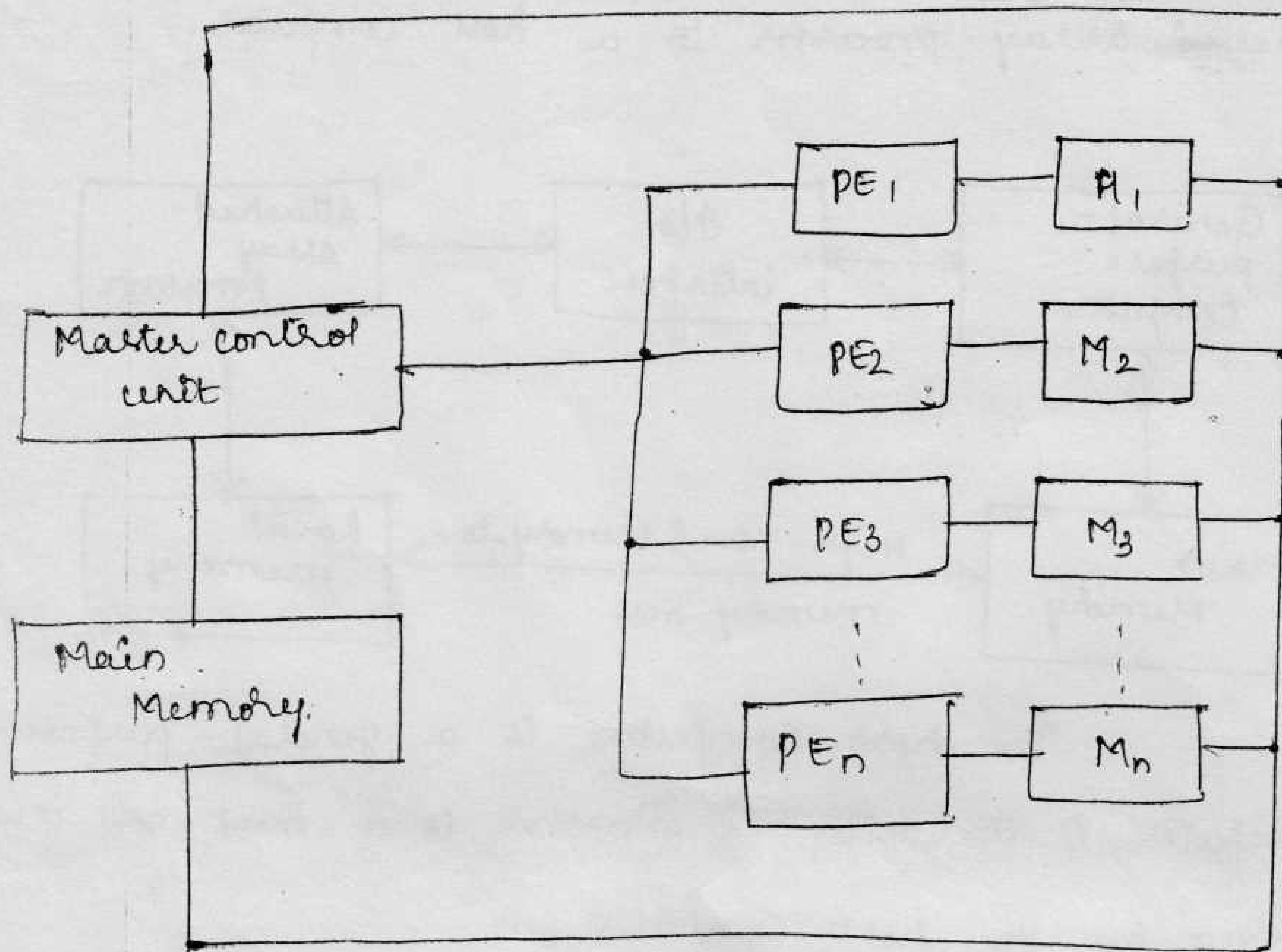
SIMD Array processor :-

An SIMD array processor is a computer with multiple processing units operating in parallel. The processing units are ~~synchronized~~ Synchronized to perform the same operation under the control of

www.specworld.in www.smartzworld.com

a common control unit, thus providing a single instruction stream, multiple data stream (SIMD) organization.

A general block diagram of an array processor is shown in below fig.



The fig. consists of a set of identical processing elements (PE) each having a local Memory M. Each Processor element includes an ALU, a floating-point arithmetic unit & working registers. The master control unit controls the operations in the PE's. The main memory is used for storage of the program. The function of the master control unit ~~controls the operations in the processor elements~~ is to decode the instructions & determine how the instruction is to be executed. Scalar & program control instructions are directly executed with in the Master Control unit. Vector instructions are broad cast to all PE's simultaneously. Each PE uses operands stored in its local memory. Vector operands are distributed to

www.jntuworldupdates.org

PE has a flag that is set when the PE is active & reset when the PE is inactive.

For eg, An array processor contains a set of 64 PE's. If a vector length of less than 64 data items is to be processed, the control unit selects the proper number of PE's to be active. Vector of greater length than 64 must be divided into 64-word portions by the control unit.

Vector processing :-

In many scientific & engineering applications, the problems can be formulated interms of vectors & matrices that lend themselves to vector processing.

Computers with vector processing capabilities are in demand in specialized applications.

The following are representative application areas where vector processing are used.

- Long-range weather forecasting
- petroleum explorations
- Seismic data analysis
- Medical diagnosis.
- Aerodynamics & space flight simulations
- Artificial intelligence & expert systems
- Mapping the human genome
- Image processing.

To achieve the required level of high performance it is necessary to utilize the fastest & most reliable hlw & apply innovative procedures from vector & parallel processing

www.smartzworld.com

www.jntuworldupdates.org

Vector operations :-

Many scientific problems require arithmetic operations on large arrays of numbers. These numbers are usually formulated as vectors & matrices of floating-point numbers.

A vector is an ordered set of a one-dimensional array of data items. A vector 'V' of length n is represented as a row vector by $V = [V_1\ V_2\ V_3\ ---\ V_n]$. It may be represented as column vector. consequently, operations on vectors must be broken down into single computations with subscripted variables. The element V_i of vector V is written as V(I) and the index I refers to a memory address or register where the number is stored.

Consider the following 'Do' loop:

```
DO    20 I = 1, 100
20       C(I) = B(I) + A(I).
```

This a program for adding two vectors A & B of length 100 to produce a vector c.

A possible instruction format for a vector instruction is shown below.

operation Code	Base address Source 1	Base address Source 2	Base address destination	vector length

Matrix Multiplication :-

Matrix Multiplication is one of the most computational intensive operations performed in computers with vector processors. The Multiplication of two n×n matrices consists of n^2 inner products or n^3 multiply-add operations. An n×m matrix of numbers has 'n' rows & 'm' columns and may be considered

www.jntuworldupdates.org

adder. During the next ...

of the adder products entering the adder pipeline. At the end of the eighth cycle, the first products A_1B_1 through A_4B_4 are in the four adder segments, and the next four products A_5B_5 through A_8B_8 are in the multiplier segments. At the beginning of the ninth cycle, the o/p of the adder is A_1B_1 & the o/p of the multiplier is A_5B_5.

Thus the ninth cycle starts the addition $A_1B_1 + A_5B_5$ in the adder pipeline. The tenth cycle starts the addition $A_2B_2 + A_6B_6$ and so on.

This pattern is shown below.

$$C = A_1B_1 + A_5B_5 + A_9B_9 + A_{13}B_{13} + \cdots$$
$$+ A_2B_2 + A_6B_6 + A_{10}B_{10} + A_{14}B_{14} + \cdots$$
$$+ A_3B_3 + A_7B_7 + A_{11}B_{11} + A_{15}B_{15} + \cdots$$
$$+ A_4B_4 + A_8B_8 + A_{12}B_{12} + A_{16}B_{16} + \cdots$$

Memory Interleaving:-

Instead of using two memory buses for simultaneous access, the memory can be partitioned into a no. of modules connected to a common memory address & data buses. A memory module is a memory array together with its own address & data registers.

The below fig. shows a memory unit with four modules. Each memory array has its own address register AR & data register DR. The address registers receive information from a common address bus and the data registers communicate ... directional data bus.

www.specworld.in

for eg, the multiplication of two 3x3 matrices A & B.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

The inner product obtain is,

$$c_{ij} = \sum_{k=1}^{3} a_{ik} \times b_{kj}$$

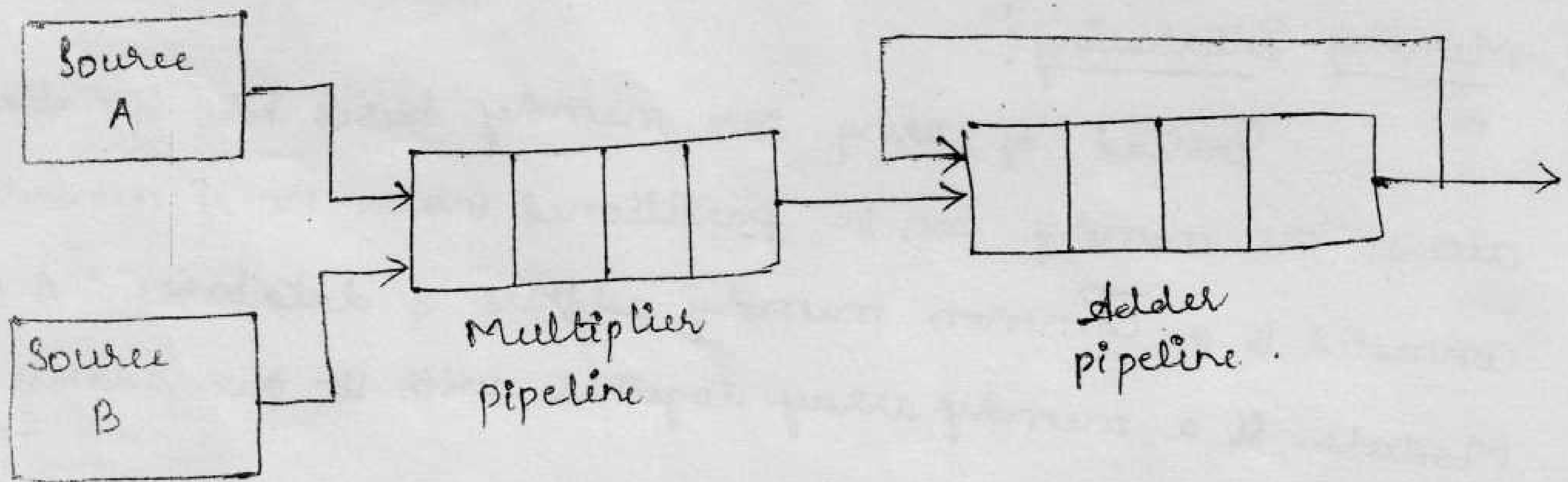
for eg, $i=1, j=1$

$$c_{11} = a_{11} b_{11} + a_{12} b_{21} + a_{13} b_{31}$$

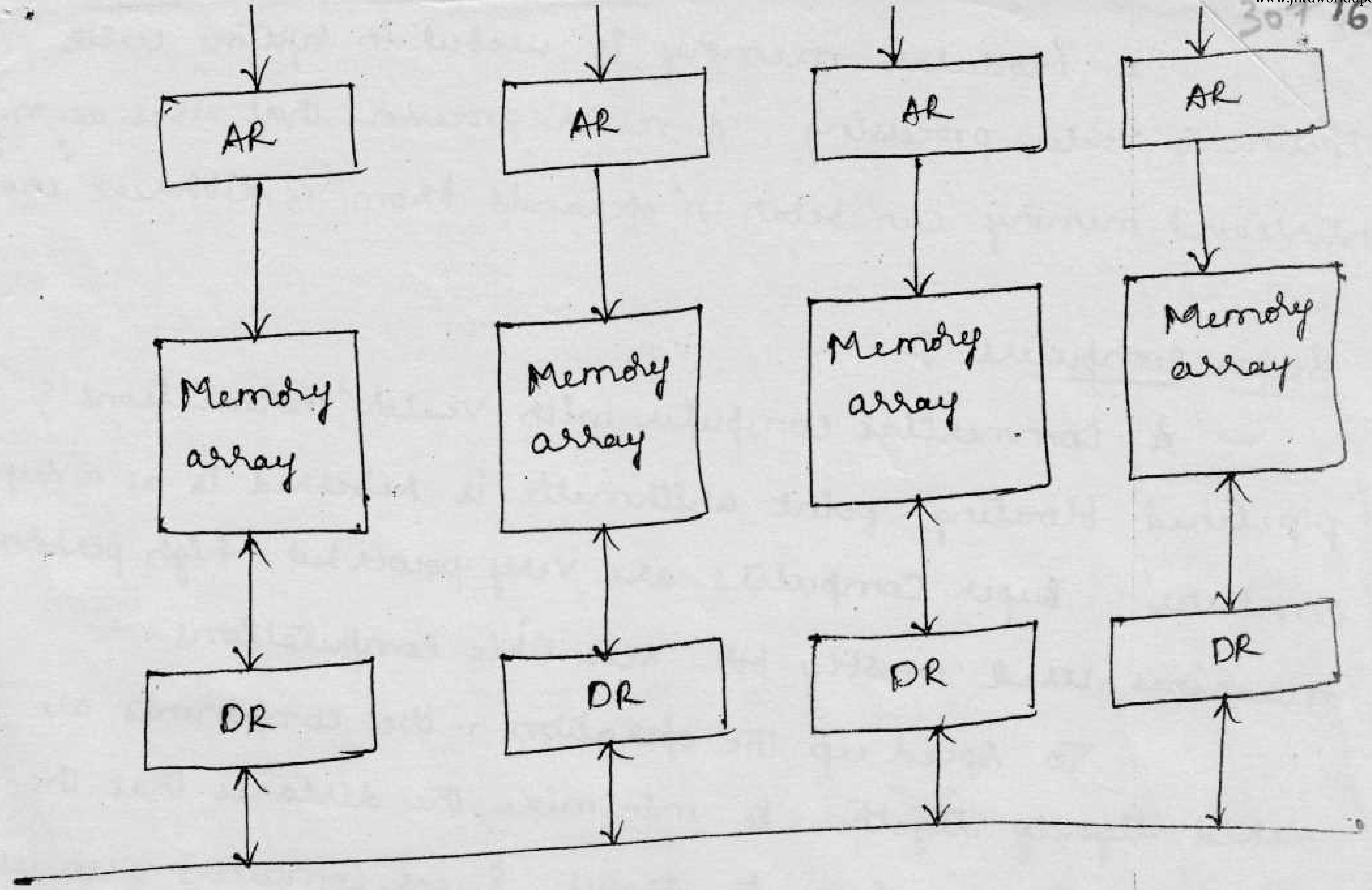
In general, the inner product consists of the sum of 'k' product terms of the form.

$$C = A_1 B_1 + A_2 B_2 + \cdots + A_k B_k .$$

The inner product calculation on a pipeline vector processor is shown in fig.



The values of A & B are either in memory or in processor registers. The floating-point Multiplier pipeline and the floating point adder pipeline are assumed to have four segments each. All segment registers in the multiplier & Adder are initialized to zero. Therefore the o/p of the adder is 0 for the first eight cycle until both pipes are full. Ai & Bi pairs are brought in



Data bus.

The two LSB of the address can be used to distinguish between the four modules. The Modular System permits one module to initiate a memory access while other modules are in the process of reading or writing a word and each module can have a memory request independent of the state of the other modules.

The advantage of a modular memory is that it allows the use of a technique called 'interleaving'. In an interleaved memory, different sets of addresses are assigned to different Memory Modules.

For eg, In two-module memory system, the even addresses may be in one module & the Odd addresses in other. When the no. of modules is of power 2, the LSB of the address select a memory module & the remaining bits designate the specific location to be accessed with in the selected Module.

A Modular memory is useful in systems with pipeline & vector processing. A vector processor that uses an n-way interleaved memory can fetch 'n' operands from 'n' different modules.

Super computers :-

A commercial computer with vector instructions & pipelined floating-point arithmetic is referred to as a 'super computer'. Super computers are very powerful, high performance machines used mostly for scientific computations.

To speed up the operation, the components are packed tightly together to minimize the distance that the electronic signals have to travel. Super computers also use special techniques for removing the heat from circuits to prevent them from burning up because of their closeness.

Super computers aren't suitable for normal everyday processing. They are limited in their use to a no of scientific applications etc. They have limited use & limited market because of their high price.

A measure used to evaluate computers in their ability to perform a given no. of floating-point operations per second is referred to as 'flops'. The term 'megaflops' is used to denote million flops & gigaflops to denote billion flops.