(14) (+15) × (−13)

13 → 01101

−13 → 10010

−13 → 10011

BR = 01111

$\overline{BR}+1 = 10001$ 10000

QR = 10011

$Q_n Q_{n+1}$		AC	QR	Q_{n+1}	SC
	Initial	00000	10011	0	101
10	Subtract BR	10001			
		10001			
	ashr				
	~~ash~~ (AC & BR)	11000	11001	1	100
11	ashr	11100	01100	1	011
01	Add BR	01111			
		+ 01011			
	ashr	00101	10110	0	010
00	ashr	00010	11011	0	001
10	Subtract BR	10001			
		10011			
	ashr	11001	11101	1	000

00110 00010

00110000011

12864 32 16 8 4 2

128 64 8 4 2 1

195

$(j-1)$, k-bit adders to produce a product of $j+k$ bits.

Consider $j = 3$, $k = 4$ → Multiplicand bits.
↳ Multiplier bits

$j \times k = 12$ AND gates.

$j - 1 = 2$ four bit adders

$k = 4$, $j = 3$ 4 bit by 3 bit multiplier

↳

$b_3\ b_2\ b_1\ b_0$

$a_2\ a_1\ a_0$

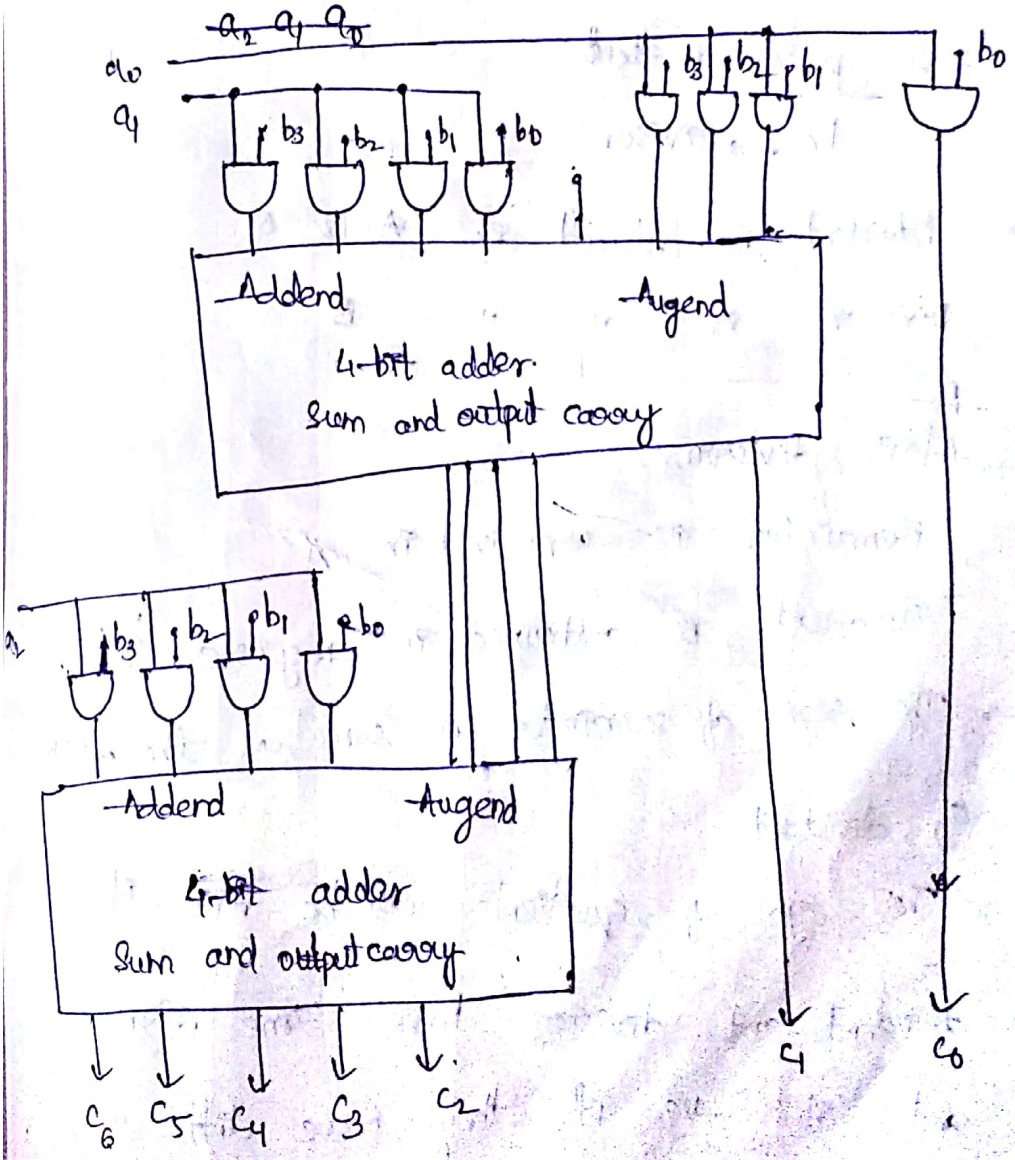


Fig: 4-bit by 3 bit array multiplier.

~~2.1~~

→ ~~It is a~~ 4x3 multiplier in which, 2(j-1) 4-bit adders to produce a product of 4+3=7 bits ~~output~~.

* Division Algorithms:

→ Algorithm for division of signed magnitude numbers

Nr → Dividend
Dr → Divisor

→ Dividend is stored in A & Q
Divisor is " " B

~~Re~~

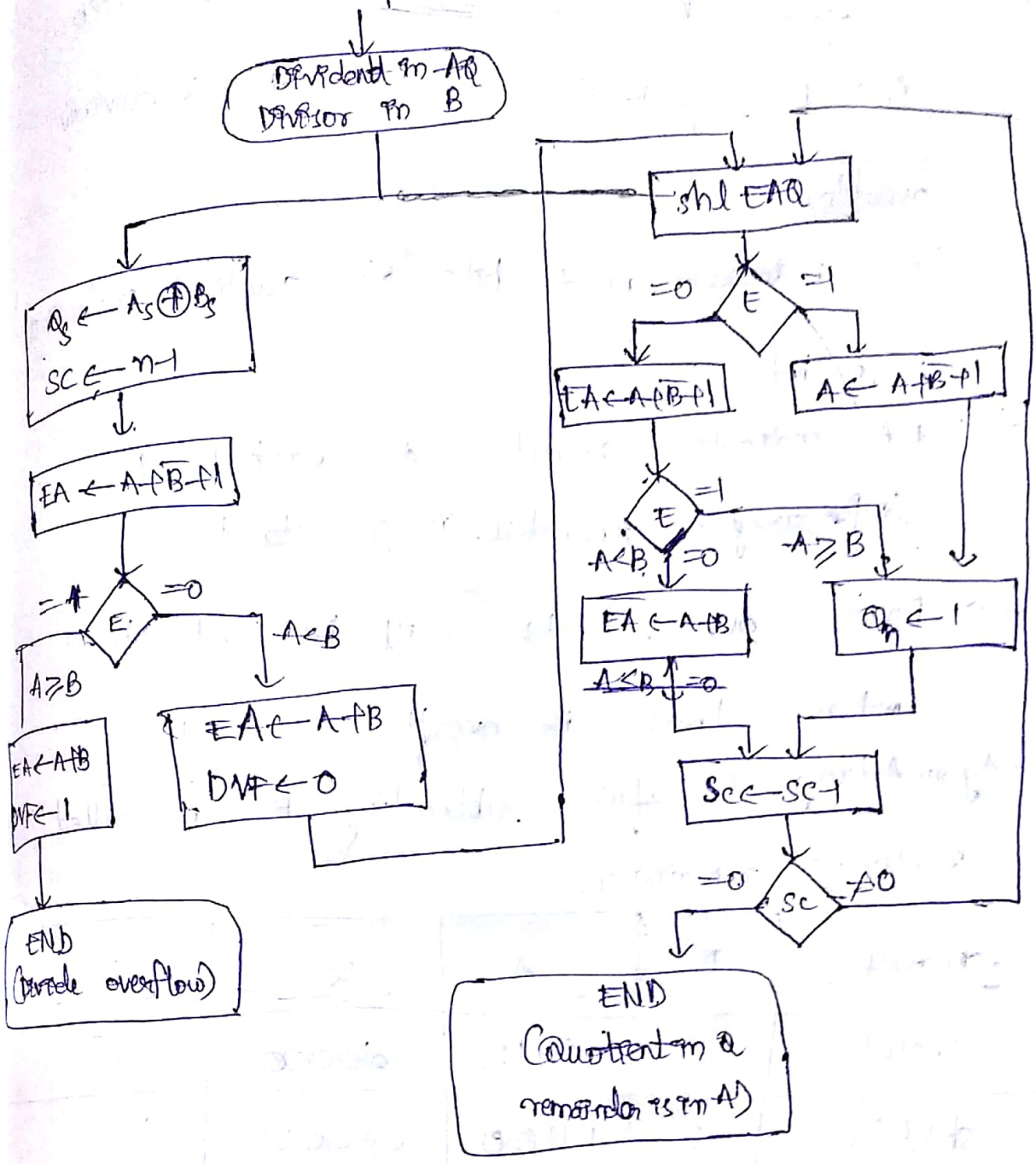
→ After division,
Remainder is contained in A
Quotient is contained in register Q

→ The sign of remainder is same as the sign of dividend

→ The sign of quotient will be +ve if dividend and division have same sign and it is -ve if they have diff "
sign of divisor is in B_s
" " dividend " " A_s

Flowchart

Divide operation



Eg: 44 8

B = 17

B = 10001

$\overline{B}+1$ = 01111

→ The divide quotient overflow occurs when $A \geq B$. quotient " indicates divide overflow.

→ 'n' indicates no. of bits in quotient including sign bit.

→ EA indicates register A with E.

(If carry is generated, it goes to E)

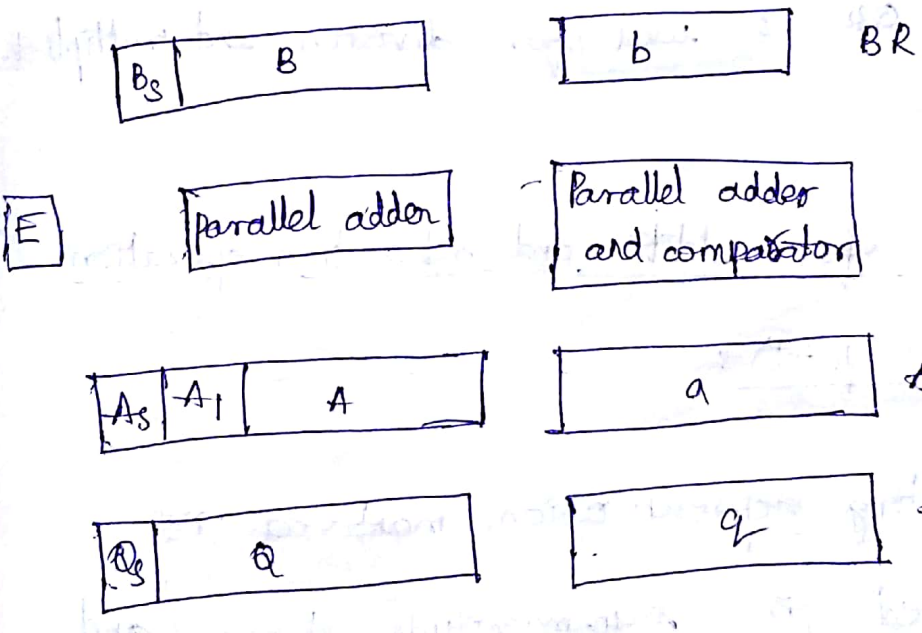
→ Divide overflow FF will be set to 1 when there is overflow.

Again Adding B after subtracting B is called restoring remainder.

Initial	E	A	Q	SC
Initial		01110	00000	5
shl EAQ	0	11100	00000	
add B+1		00111		
E=1	1	01011	0000 1	
Set Q_n=1	1	01011	0000 1	4
shl EAQ	0	10110	0001 0	
add B+1		01111		
E=1	1	00101		
Set Q_n=1	1	00101	0001 1	3
shl EAQ	0	01010	00110	
add B+1		01111		
E=0, leave Q_n=0	0	11001		
Add B		10001		
Restoring remainder	1	01010		2

	E	A	Q	SC
	0	10100	01100	
shl EAQ		01111		
Add B+1	1	00011		
E=1	1	00011	01101	1
set Q_n=1				
shl EAQ	0	00110	11010	
Add B+1		01101		
E=0 leave Q_n=0	0	10001		
Add B.	1	00110	11010	0
		6	26	
		↓ remainder	↓ quotient	

Floating-Point Arithmetic Operations:



Reg. Register for floating-point arithmetic operations

B_S – sign of mantissa

B – magnitude of mantissa

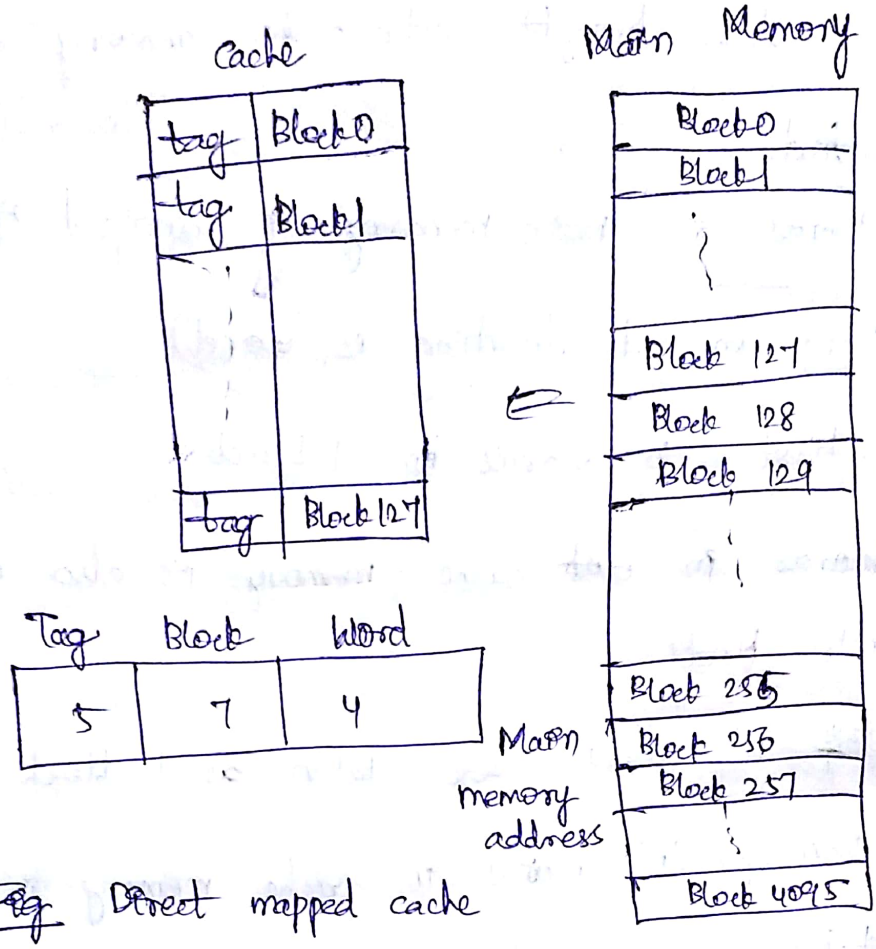
b – exponent

Together it is called BR

Similarly A_S, A_1, A and 'a' indicates AC

Mapping functions:

Direct Mapping:



Direct mapped cache

Here, each block contains 16 words.

Total no. of words = 4096 × 16

= 64 K locations

= 64 × 1024

= $2^6 \times 2^{10}$.

∴ 16 address bits are required to access 64K locations.

Cache memory contains 128 blocks

∴ No. of words = 128 × 16

= 2048 words

= 2K blocks

Block 1 of main memory is placed in block '1 modulo 128' of cache memory

Block 0, 128, 256.. of main memory is placed in block 0 of cache

ly, block 1, ~~128~~ 129, 257. ... in block 1.

→ 16 bits in the figure indicates address of word of main memory.

→ 7 bits contains the block of cache memory where the word may be present.

→ Tag bit indicates which block of main memory is contained in block of cache (0, 128, 256)

→ If the tag bit in main memory matches with the " " " in cache memory, then it indicates that the required word is contained in cache memory (~~It~~ is brought into cache if it is not present.)

Associative mapping:

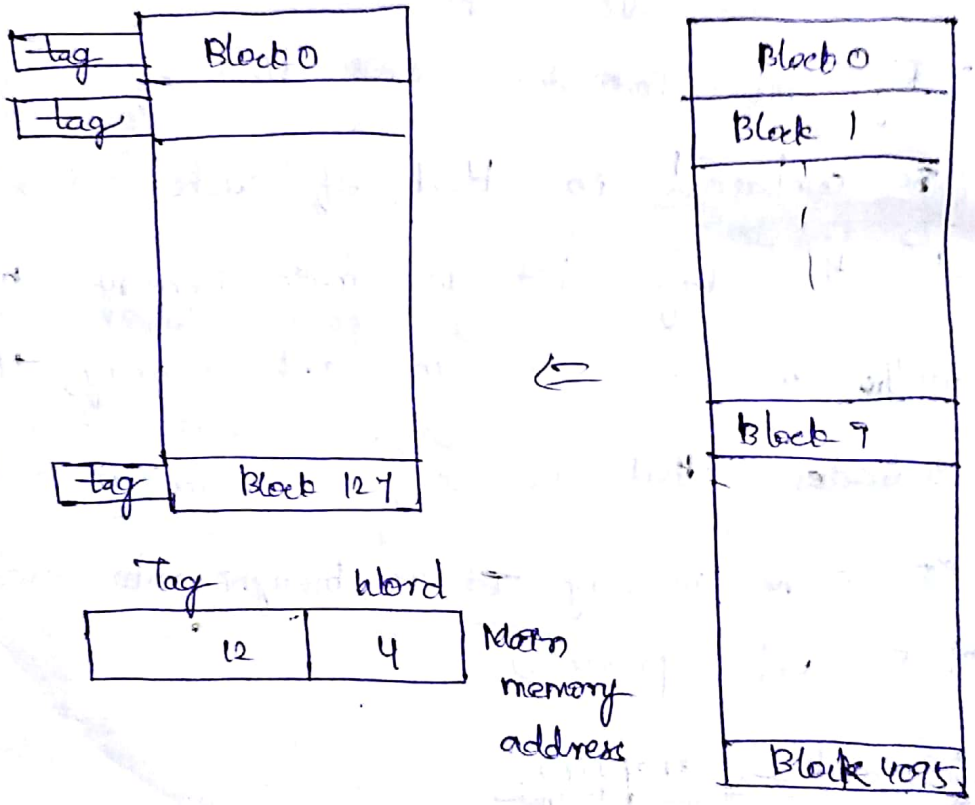
Any block of main memory can be placed in " " " cache " .

Main memory address indicates the word of " " which is to be accessed.

Word is the data stored in a location.

→ Tag bits of block of cache memory indicate which block of main memory is contained in that block.

→ If the tag bits of main memory matches with the tag bits of cache memory, it indicates that the word is present in cache memory.



Eg. Associative mapped cache.

→ When new block is to be brought into cache memory (from main memory), one block of cache memory should be removed when the

cache is full.

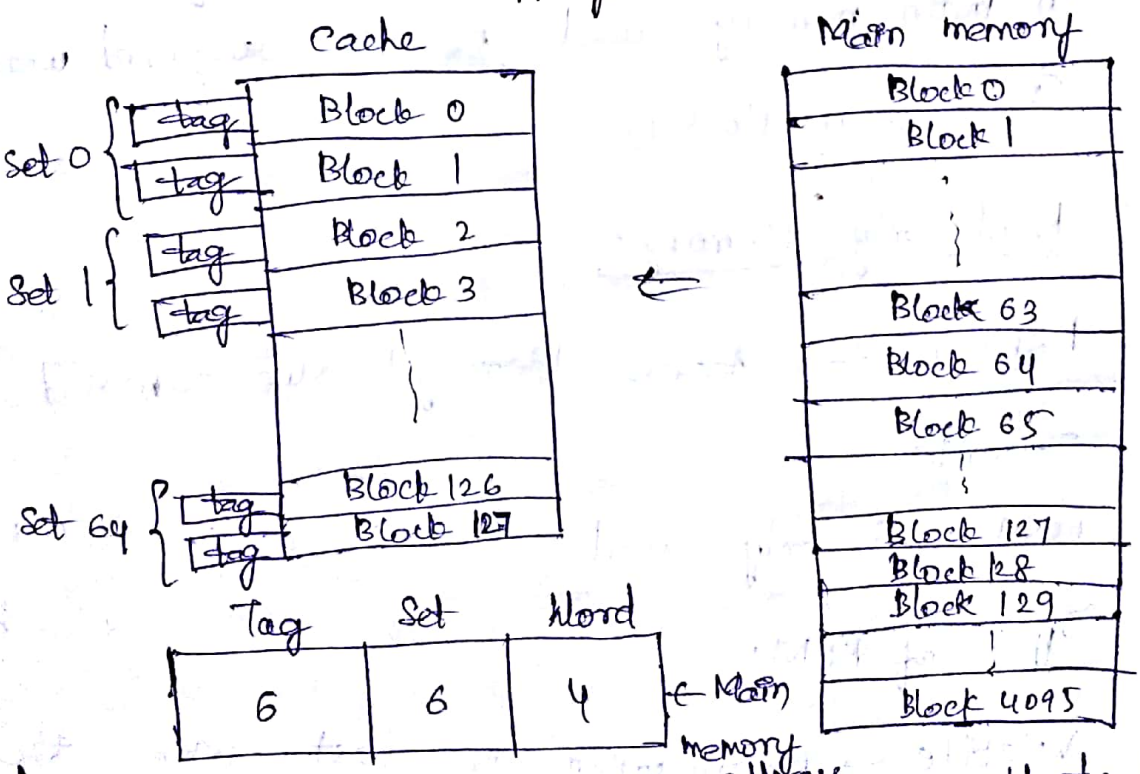
→ Algorithms which decides which block of cache memory is to be replaced when the cache is full are called replacement algorithms.

→ One of the algorithm is LRU (Least Recently Used block) (which is accessed long back.)

→ Another algorithm is oldest algorithm (where the oldest block is removed.)

→ 4 bits in main memory address indicates which word in the block (16 words in a block) is to be accessed.

Set-Associative Mapping:



Eg: Set associative ~~mapping~~ mapped cache memory with two blocks per set.

→ In this mapping, in cache memory, some no. of blocks are taken as a set.

Here, each set contains 2 blocks

∴ 128 blocks ⇒ 64 sets

Block 0, 64, 128 ---- 4032 are placed in set 0 of cache memory (Block 0 or block 1)

→ 16 bit main memory address is divided into 3 (parts) fields

6 bits ($2^6 = 64$) are used to indicate set no's set field.

→ In the corresponding set (eg. set 0) the tag bits which matches with the tag bits in main memory add. is the required word (block 0 (or) block 1)

Read only Memory:

RAM (Random Access Memory) → we can read or write.

ROM can only read.

Types of ROM:

Volatile → information is lost when the power is off

Virtual Memories:

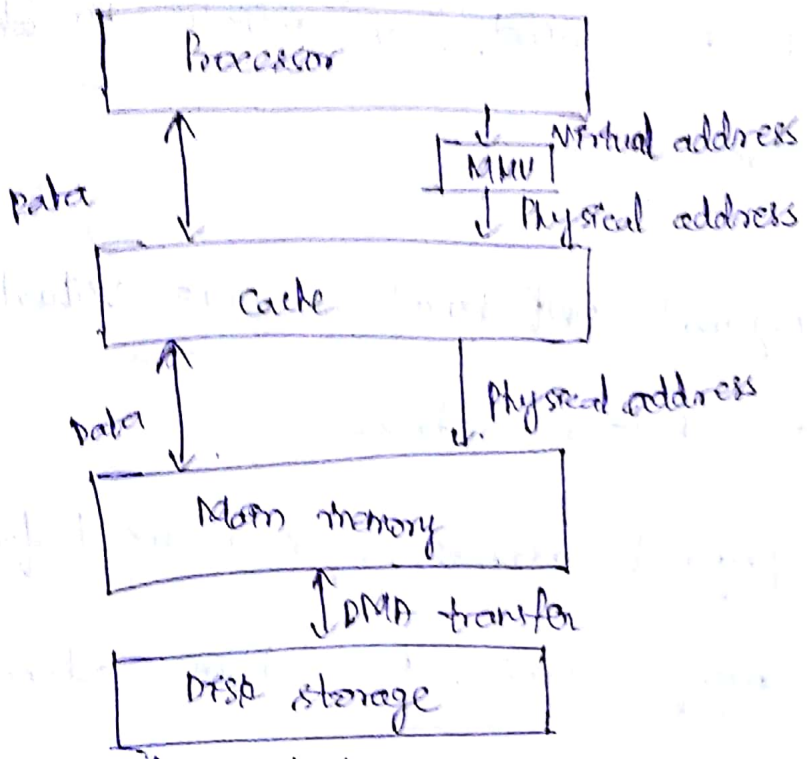


Fig Virtual memory organisation

→ Initially, files will be stored in disk.

→ To execute that file, it should be brought into main memory. For faster access, cache memory is used i.e. data from main memory is transferred to cache.

→ Transfer b/w disk and main memory is done in terms of pages where each page contains 2K - 16K words.

→ Initially, the address generated by processor is called virtual address. It contains page num

and offset. Page num indicates in which page required word is contained. offset indicates in what location " " "

→ Memory Management Unit (MMU) converts virtual address into Physical address

→ Using physical address, required word from main memory address. It contains address of page in the main memory

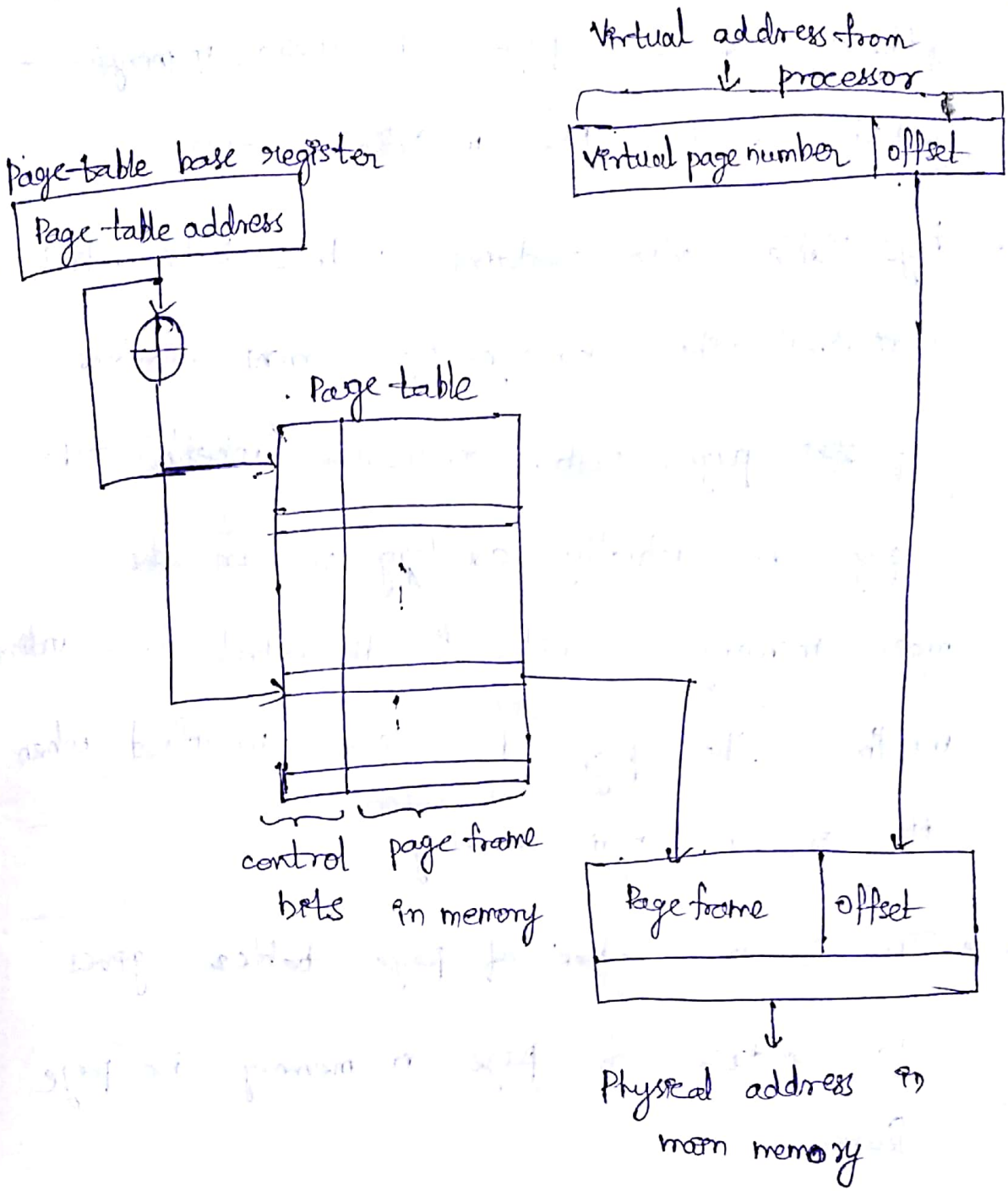
Eg: If virtual address gives page 0

location of page 0 is given by physical address.

→ Physical address contains page frame and offset. Page frame indicates the starting location of page in the main memory and offset indicates the location of word in page

* Address translation:

(It is the process of converting virtual address into physical address)



→ Page table contains the address of each page in main memory

→ Page table base register contains the starting address of the page table.

Eg: If page no. is 2, 2 is added to page table base register value and it gives the

address of the page in main memory (which is in page table)

→ Page Table also contains control bits which indicates the corresponding current status of the page. status indicates whether the page is actually existing in the main memory or not. Ly, control bits tells whether the page has been modified when it is in main memory

→ The location of page tables gives the address of page in memory i.e page frame

→ starting address of page in main memory is called page frame.

Eg: If page frame = 1000, offset = 4

word in box is accessed.

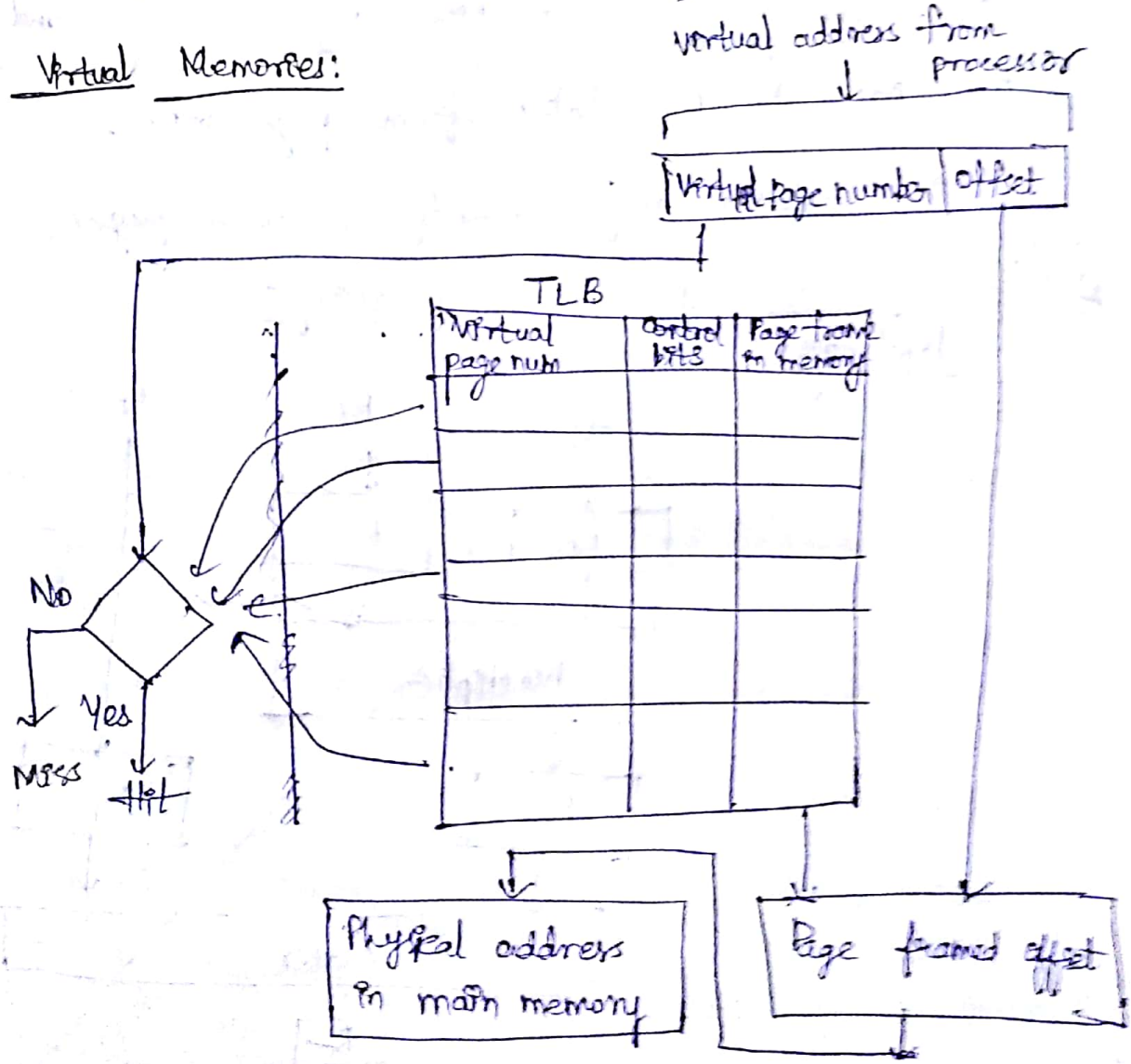
→ Page table is present in main memory.

→ Conversion of virtual to physical address is done by MMU.

→ As page table is contained in main memory, accessing time of converting virtual address to physical address will be more.

→ In order to reduce the time to convert from virtual to physical address, a portion of the page table is placed in RAM in the cache memory. The portion the page table which is contained in RAM is called TLB (Translation Lookaside Buffer)

Virtual Memories:



TLB contains page table entries of recently accessed pages.

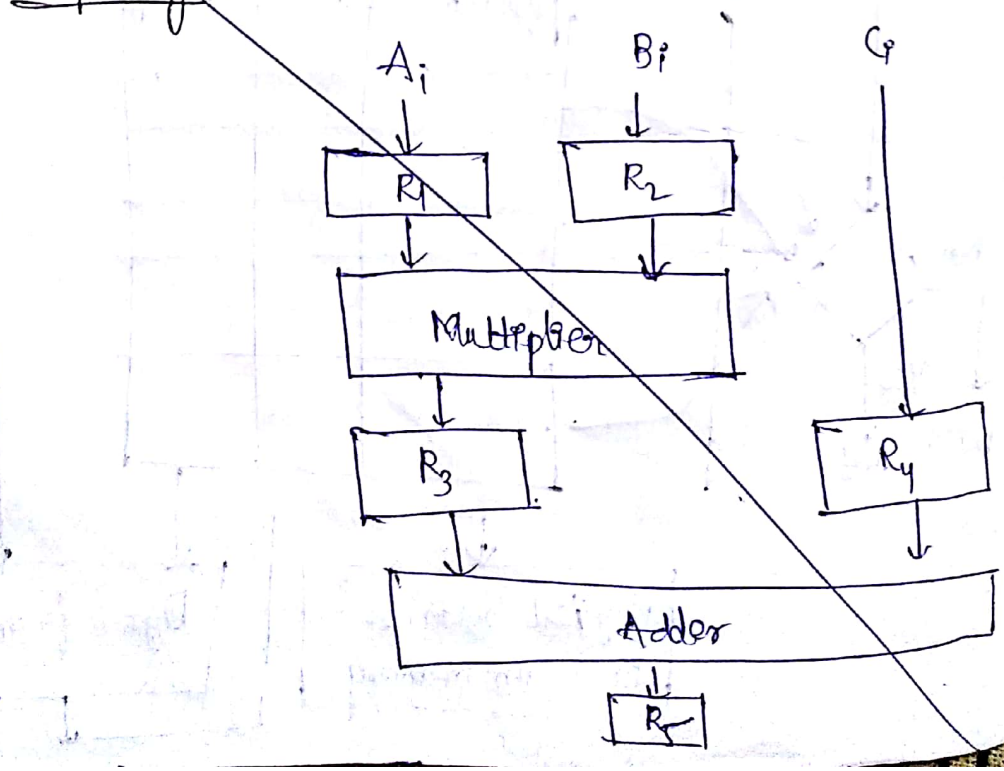
Eg: If TLB contains page 1, 4, 5, then virtual page num will be 1 or 4 or 5.

→ Virtual page num in virtual add. is compared with virtual page num in TLB.

If it matches with virtual " " ,
the required info. is contained in TLB.
Otw, the " " " not " " " and
it has to be taken from page table.
TLB contains only recently accessed pages.

21/9/17 Pipelining and Vector Processing:

Pipelining:



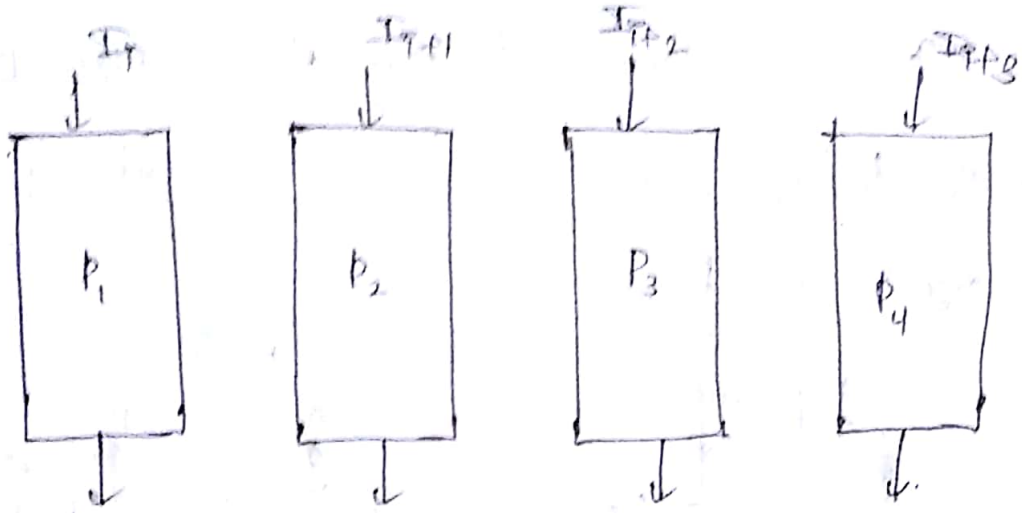


fig Multiple functional units in ALU

Here 4 (P_1, P_2, P_3, P_4) nonpipelining ckts are required to achieve the speed through pipelining

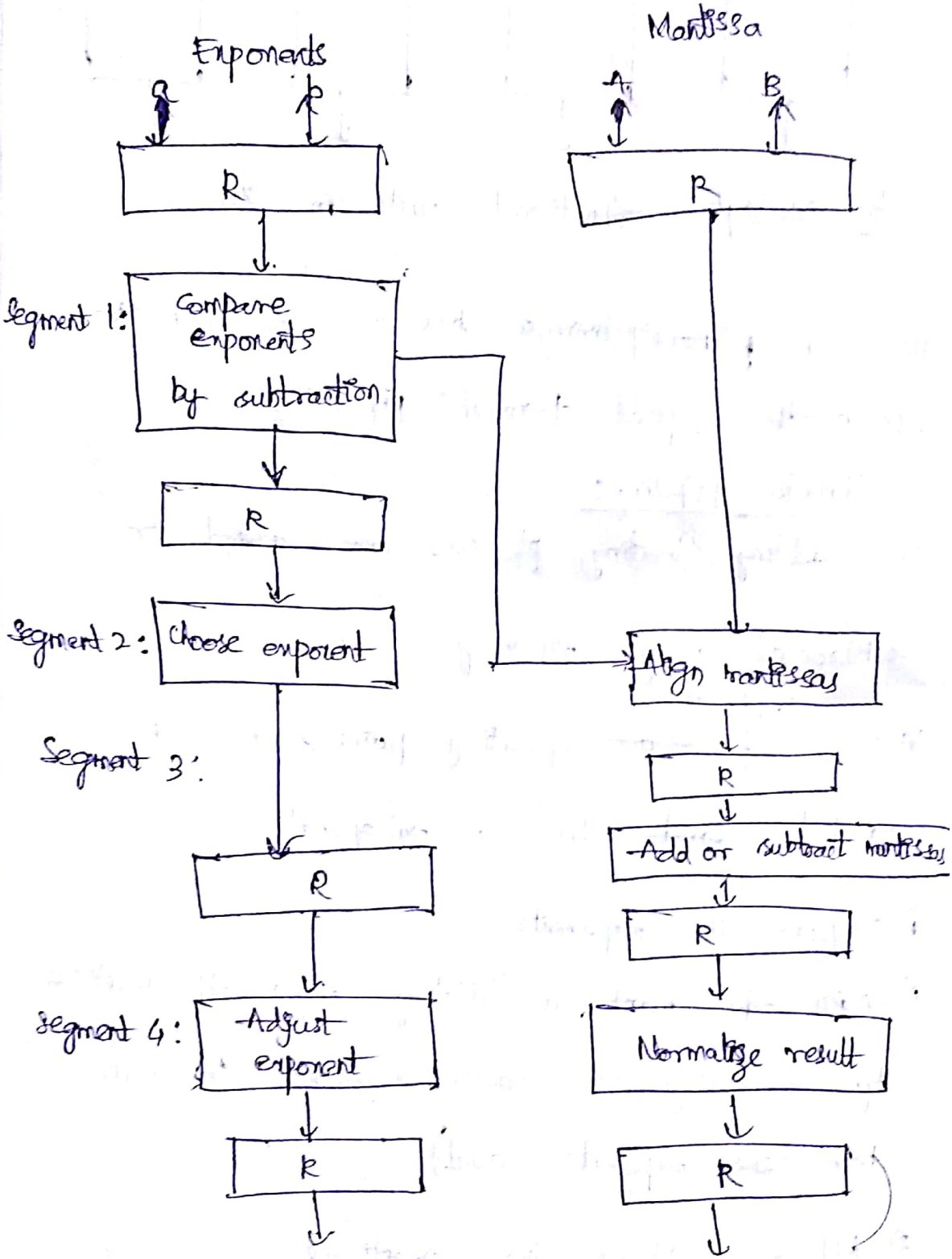
Arithmetic pipeline:

Here, floating pt. no's are added or subtracted using pipelining.

Adding/subtraction of floating point no's is divided into 4 sub operations.

1) Compare the exponents.
2) Align the mantissa (shifting right the mantissa by increasing the lower exponent to make both the exponents equal)
3) Add or subtract the mantissas
4) Normalize the result. (Most significant digit of floating pt is nonzero → normalised)

In case binary floating pt no, MSB bit of binary " " should be equal to 1.



Instruction Pipeline:

It is used to process multiple instructions simultaneously. In order to process the instructions, following steps are to be followed are:

1) Fetch the instruction from memory
2) Decode the instruction
3) Calculate the effective address. (to find the location of operand.)

Eg: In case of branch instruction, eff. address indicates where to branch.

4) Fetch the operands from memory
5) Execute the instruction:

Eg: Add, Mul, data transfer etc

6) Store the result in proper place

Eg: In case of addition, result is stored in memory of register

These steps are divided into 4 segments.

For conditional BR instruction, condition is verified in segment 4. It will verify whether the condition is true. Processing takes place

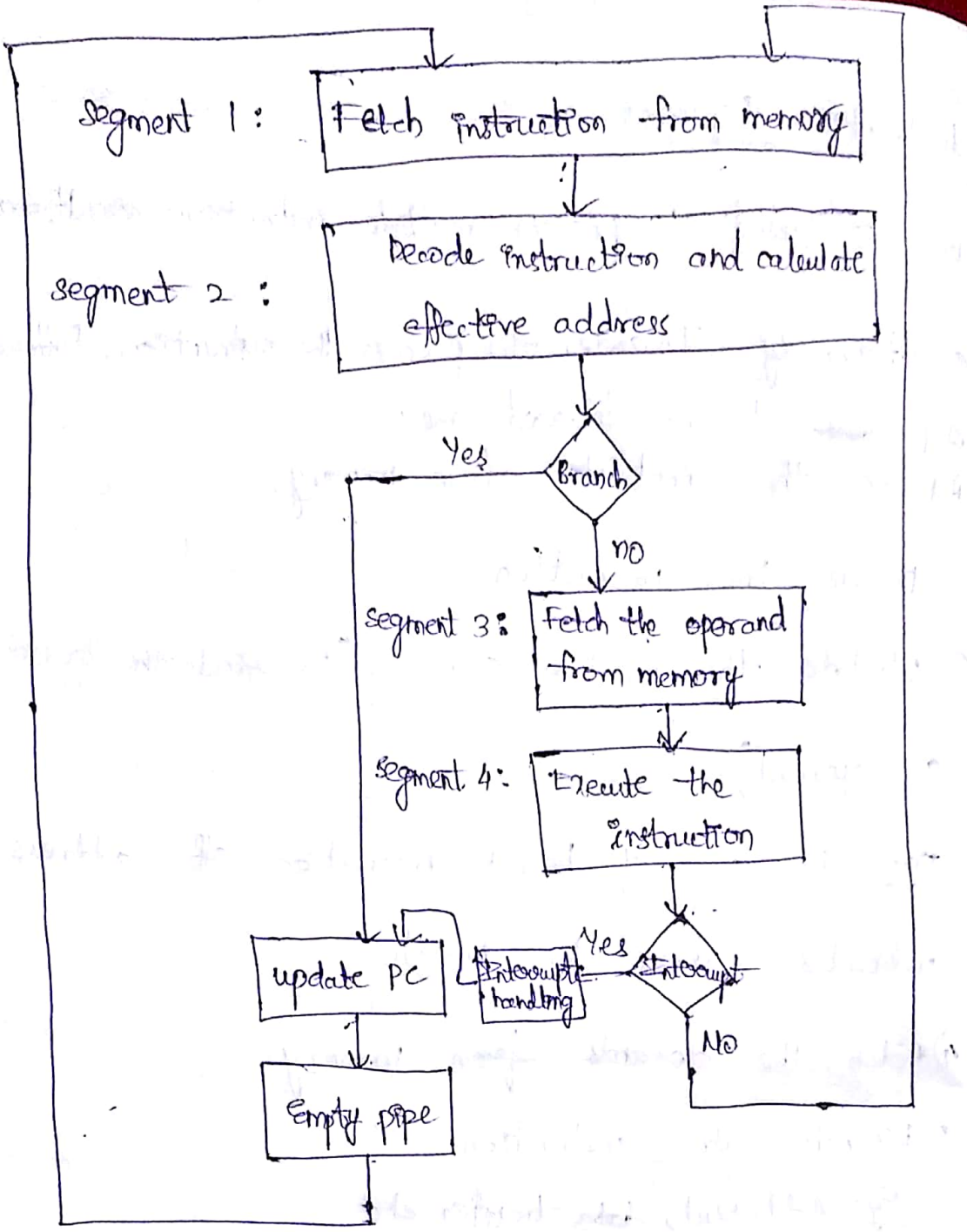


Fig Four-segment CPU pipeline

from branching address. After executing some instruction, before executing next instruction, it will check whether there is any interrupt. and if there is an interrupt ISR (Interrupt service Routine) is executed.

and PC is updated with address of ISR.

→ For unconditional BR, PC is updated with branching address, and pipe is emptied. Branching instruction is prefetched (before) and condition is checked.

→ If the condition is false in case of cond. branching, the next instruction after BR instruct^n is executed

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

Fig. Timing of instruction pipeline

If the decoded instruction is BR instruction, the next instruction is halted till the execution of BR instruction. If the condition is true, the first segment of instruction in branching add. is executed. Otherwise, the the fetched

instruction is decoded. During decoding of first instruction after branching, second instruction after branching is decoded.

Diff

Pipeline conflicts:

Problems in pipeline which arise are called pipeline conflicts

1) Resource conflicts

→ When 2 segments of instruction pipeline try to access the memory at the same time, it is called as resource conflicts.

(depends on instructions in pipeline)

2) Data dependency conflicts:

→ When an instruction depends upon the result of previous instruction which is not yet available, data dependency conflict occurs.

Eg: The instruction is delayed till the result in previous is obtained.

→ Inorder to delay the instruction,

a hardware ckt is used called as hardware interlocks.

→ Another way is to delay the instruct^{h}, is, by using compiler which inserts the No operation instruction to delay the instruct^{h}. This is called called as delayed load.

→ Instead of storing the result of previous instruct^{h} in destination register from which the next instruct^{h} has the access, it is directly forwarded to next instruct^{n}. by bypassing the register the operation is called as operand forwarding.

~~3) Brach~~

3) Branch difficulties:

During pipelining, operation of pipelining is interrupted or halted which is called as branch penalty. It degrades the performance of pipeline In order to minimize the degradation of the performance of pipeline, during branching instruction, we will be prefetching the instruct^{h}'s

completion of the from branching address before the execution of the instruction. It is done through hardware. It is one way to handle Branch difficulties

→ In order to have a continuous flow of pipeline during branching instruction, compiler will be either rearranging the instructions or inserts the No operation instruct^n. This which is done in RISC computer. This technique is called as delayed branch

Examples for delayed load and delayed branch using RISC pipeline:

Instruct^n pipeline corresponding to RISC computer is called RISC pipeline. It is devided into 3 segments

I : Instruction fetch

A : ALU operation → decoding and performing ALU operation.

E : Execute instruction

→ RISC computer consists of 3 types of

instructions.

1) Data manipulation instruction. Ex: Add (+, -, *, /)
2) Data transfer — Ex: load and store
3) Program control — Ex: branch

→ During data manipulation instruction, we perform data manipulation operation in 2nd segment & ALU result is transfered to destination register in 3rd segment.

→ For data transfer instructions (load or store) in the 2nd segment, effective address is calculated for load or store instruction

Ex: M[x] → R

x → eff

eff. Add.

During 3rd segment, effective add. is transfered to the data memory for load or store operation.

→ For program control instruction, during 2nd segment, branch address is calculated by ALU. During 3rd segment, branch add. is

transfered to PC.

Delayed load:

Eg:

① LOAD : $R_1 \leftarrow M[\text{address 1}]$

② LOAD : $R_2 \leftarrow M[\text{address 2}]$

③ ADD : $R_3 \leftarrow R_1 + R_2$

④ STORE : $M[\text{address 3}] \leftarrow R_3$

Pipeline

clock cycles	1	2	3	4	5	6
1. Load R_1	I	A	E			
2. Load R_2		I	A	E		
3. Add R_1+R_2			I	A	E	
4. Store R_3				I	A	E

a) Pipeline timing with data conflict

During 4th clock cycle, the value of R_2 is not ready as R_2 is loaded into destination register and data dependency conflict occurs due to in R_1+R_2 and the 3rd instruction is delayed

until R_2 is loaded by compiler by each inserting NO instruction i/w, 2nd, 3rd and 4th instruction.
∴ In 5th clock cycle, R_1+R_2 is performed.

clock cycle

clock cycle	1	2	3	4	5	6	7
1) Load R_1	I	A	E				
2) Load R_2		I	A	E			
3) No operation			I	A	E		
4) Add R_1+R_2				I	A	E	
5) Store R_3					I	A	E

∴ This technique is called as delayed load.

7/10/17

Vector Processing:

→ Vector is a collection of data items of same or diff type (Array-same type)

→ Vector processing deals with operations on vectors

→ There are two types of processors to perform any operation.

1) Scalar processor:

In scalar processor we can process

only single data item at a time.
(No arithmetic pipeline.) ⇒ : only single data item can be processed.

2) Vector processor

It processes multiple data items at a time

It uses arithmetic pipeline

→ Speed of operation is more in vector processor than scalar processor.

Eg: Vector instruction for adding vectors.

C(1:100) = A(1:100) + B(1:100)

A is vector containing 100 data items

B " " " " " "

C " " " " " "

→ Vector processor uses vector instructions.

Scalar " " scalar "

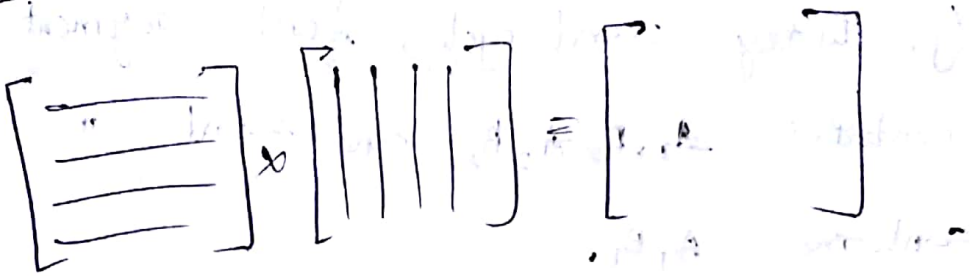
The above exp. indicates vector "

→ same operation can also be performed using scalar instruction with more than one instruction using loops.

→ In scalar instruction processor, single instruction will be operated on single data item at a time.

→ In vector processor, single instruction will be operated on multiple data items at a time. due to arithmetic pipeline.

Eg: Matrix Multiplication:



To get inner product (: element in product matrix), row vector and column vector of matrices are multiplied.

$$C = A_1B_1 + A_2B_2 + A_3B_3 + \cdots + A_kB_k$$

$A_1, A_2, \ldots A_k$ are row vectors

$B_1, B_2, \ldots B_k$ are column vectors

C - inner product.

k - can be any value

↔ This inner product is obtained by vector processor which uses pipelining.

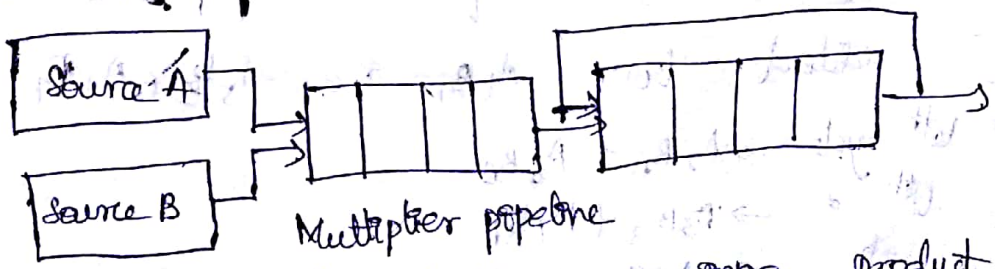


Fig. Pipeline for calculating an inner product

→ There 4 segments in multiplier pipeline and adder pipeline

→ During first cycle, first segment of multiplier pipeline contains A_1B_1,

→ IIly, during second cycle, first segment contains A_2B_1 A_2B_2 and second contains A_1B_1.

→ During 4th cycle, the segments of multiplier pipeline contains $A_4B_4, A_3B_3, A_2B_2, A_1B_1$,

→ During 5th cycle, initially adder ckt consists of 0's and 0's added to A_1B_1 which is transferred to Adder pipeline

→ After 8 cycles, the

→ During 9th cycle,

first segment of adder pipeline is added to A_1B_1 i.e. $A_5B_5 + A_1B_1$

→ 10th cycle → $A_2B_2 + A_6B_6$

11th " → $A_3B_3 + A_7B_7$

12th " → $A_4B_4 + A_8B_8$

→ Finally, partial products are obtained

$$C = A_1B_1 + A_5B_5 + A_9B_9 + A_{13}B_{13} + \ldots \quad \text{— (1)}$$

$$+ A_2B_2 + A_6B_6 + A_{10}B_{10} + A_{14}B_{14} + \ldots \quad \text{— (2)}$$

$$+ A_3B_3 + A_7B_7 + A_{11}B_{11} + A_{15}B_{15} + \ldots$$

$$+ A_4B_4 + A_8B_8 + A_{12}B_{12} + A_{16}B_{16}$$

finally, 4 partial products in 4 segments of adder pipeline are added.

Super scalar processor:

contains multiple functional units or processor units such that multiple instructions are operated on multiple (seperate) data items at the same time.

Super computer:

A commercial computer with vector instructions and pipelined floating point arithmetic operations is called a super computer.

→ In vector processor, data items are represented using floating point.

Array processors

→ They are specially used to perform operations on arrays. (collection of data items of same type)

→ There are 2 types of array processors.

1) Attached array processor

2) SIMD (Single instruction Multiple data) array processor.

Attached Array Processor:

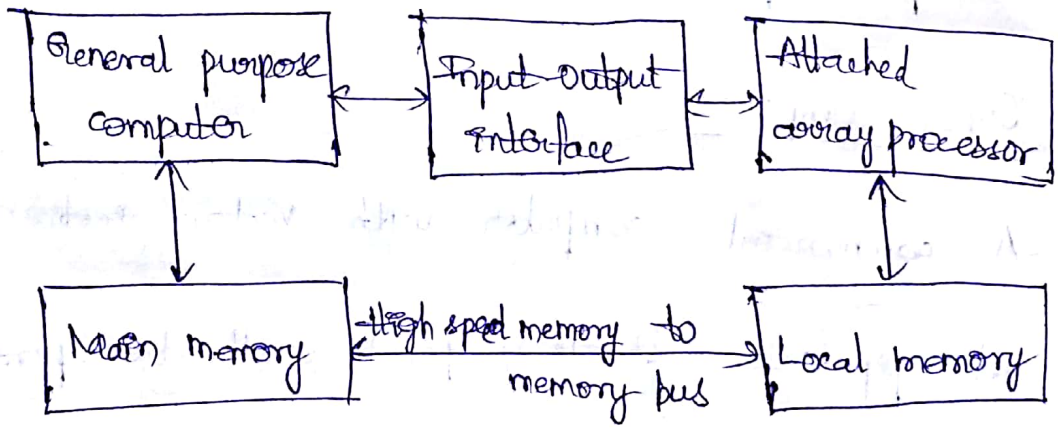


Fig Attached array processor with host computer.

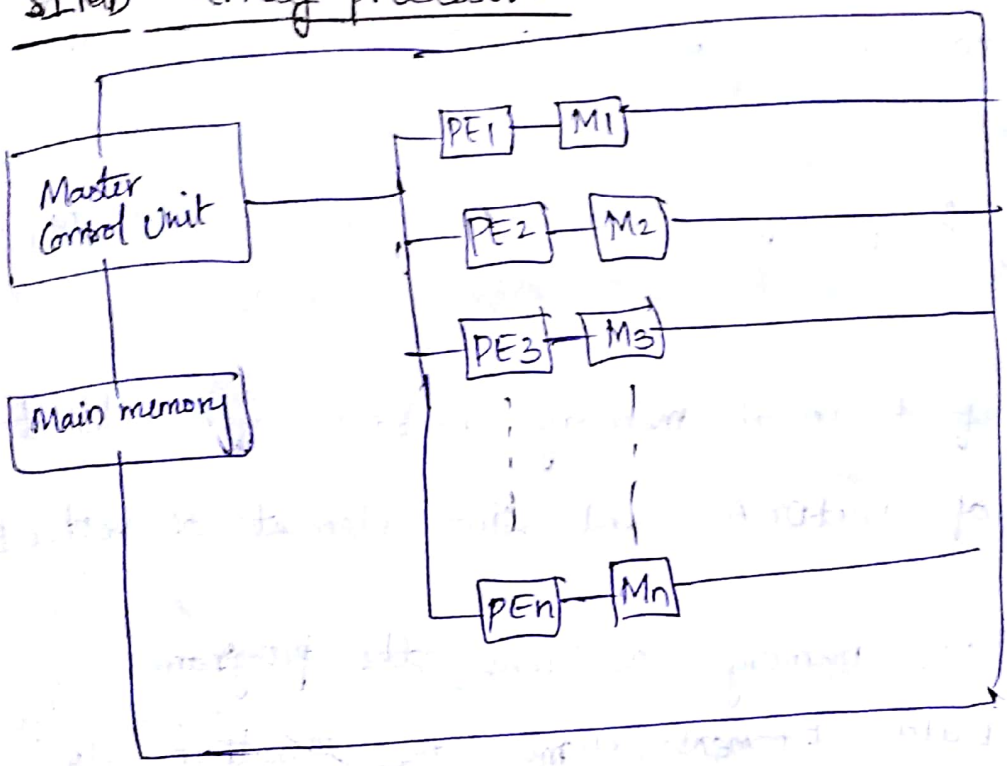
→ It is attached to the general purpose computer. It acts as I/O device to the general purpose computer.

→ It will the access the data from local memory.

→ The transfer of data from main memory and local memory is done using high speed bus

→ It contains one or more pipeline floating pt. adders and multipliers.

SIMD Array processor:



→ In this processor, it contains multiple processing elements (PE) which contains ALU and registers. (Floating Arithmetic unit)

→ $M_1, M_2 \ldots M_n$ indicates local memories

→ Master control unit decodes the instruction and

controls the operation of processing elements.

→ Scalar instructions and program control instructions are executing using master control unit.

Vector instructions are executed by processing elements.

→ In case of vector instruction mentioned before
→ First element of vector A is given to M_1

Second " " " " " M_2

:

My first " " B " " M_1

→ ∴ first local memory contains first element of vector A and first element of vector B

→ Main memory contains the program
∴ Data ~~Elements~~ items are simultaneously processed.

Multiprocessors:

→ a system contains multiprocessors then it is called multiprocessor system.

→ In a microprocessor system different jobs are performed simultaneously (or) single job is partitioned into multiple tasks which are performed simultaneously.

→ There are 2 types of multiprocessor systems
1) Shared memory (or) tightly coupled multiprocessor system.
2) distributed memory (or) loosely coupled multiprocessor system.

In this each processor uses its own local private memory.

→ We have several structures for interconnecting the components of the multiprocessor system

Interconnection structures.

1) Time shared common bus

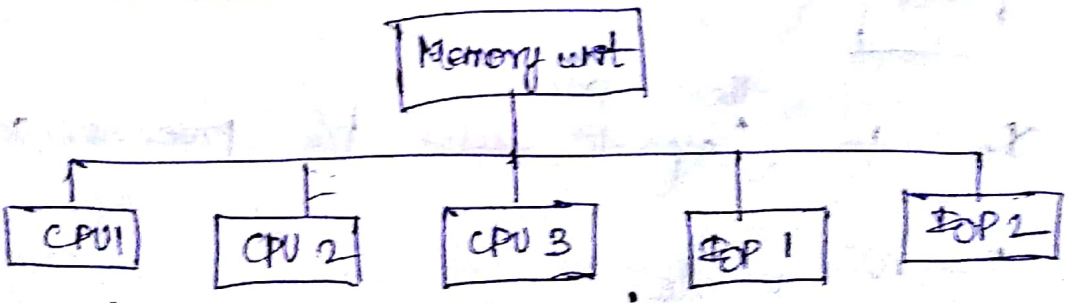


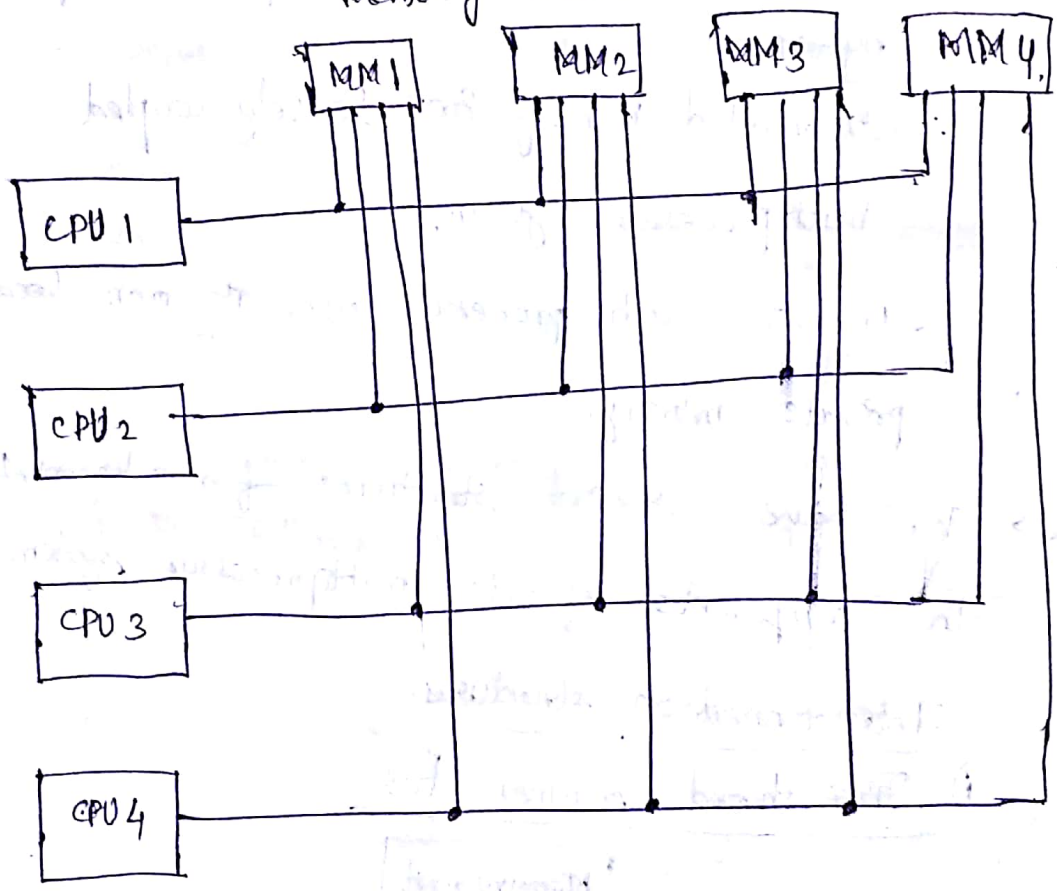
Fig: Time-shared common bus organization

→ It indicates single bus (common) b/w the processors and memory

→ When one processor uses memory, the other has to wait

IOP → I/p o/p processor which is used to perform i/p, o/p transfer

2) Multiport memory:

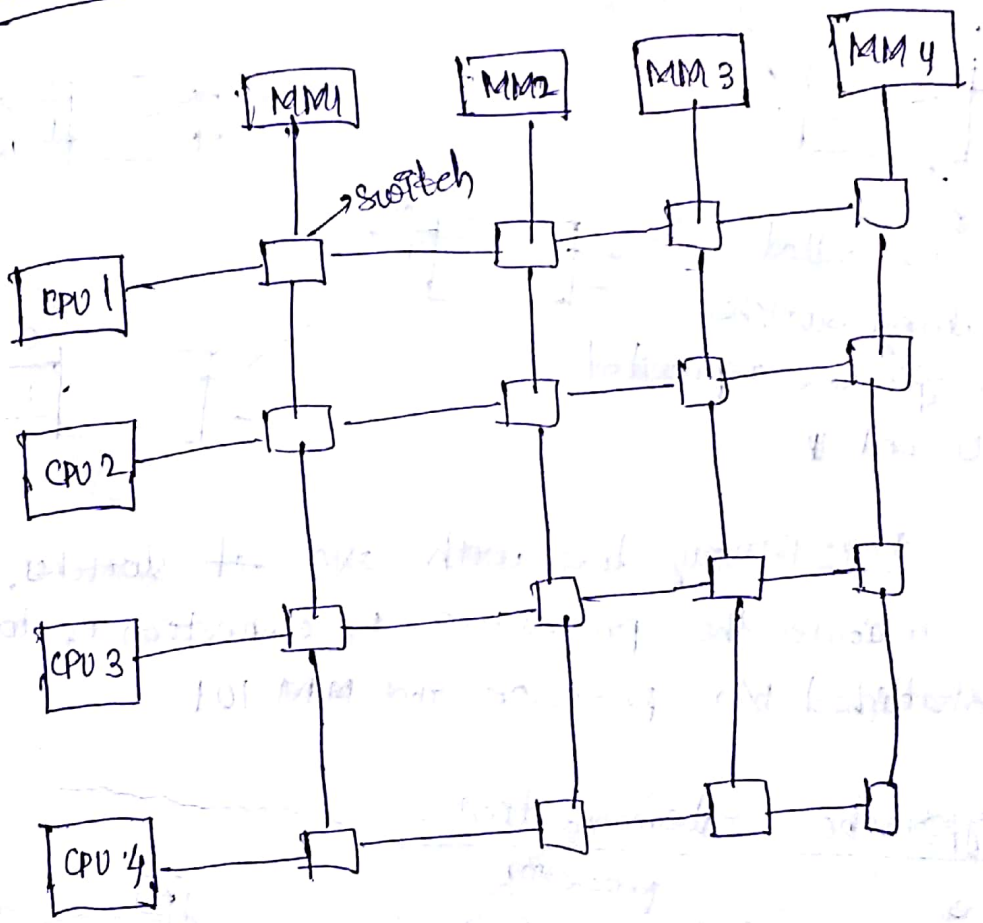


In this, there are separate buses b/w processor and memory module.

→ Memory modules are parts of memory unit.

→ Memory module contains 4 ports. Each port is connected to 1 CPU. As it contains multiple ports, it is called as multiport memory. It comes under shared memory (MM1, MM2 ... are connected to CPU1, CPU2 ...)

3) Crossbar switch:



Here, single port is used for memory module. Here, switches are used to establish the path b/w CPU and Memory module.

So All the switches along the path should be path on for the transfer to take place.

4) Multistage Switching Networks:

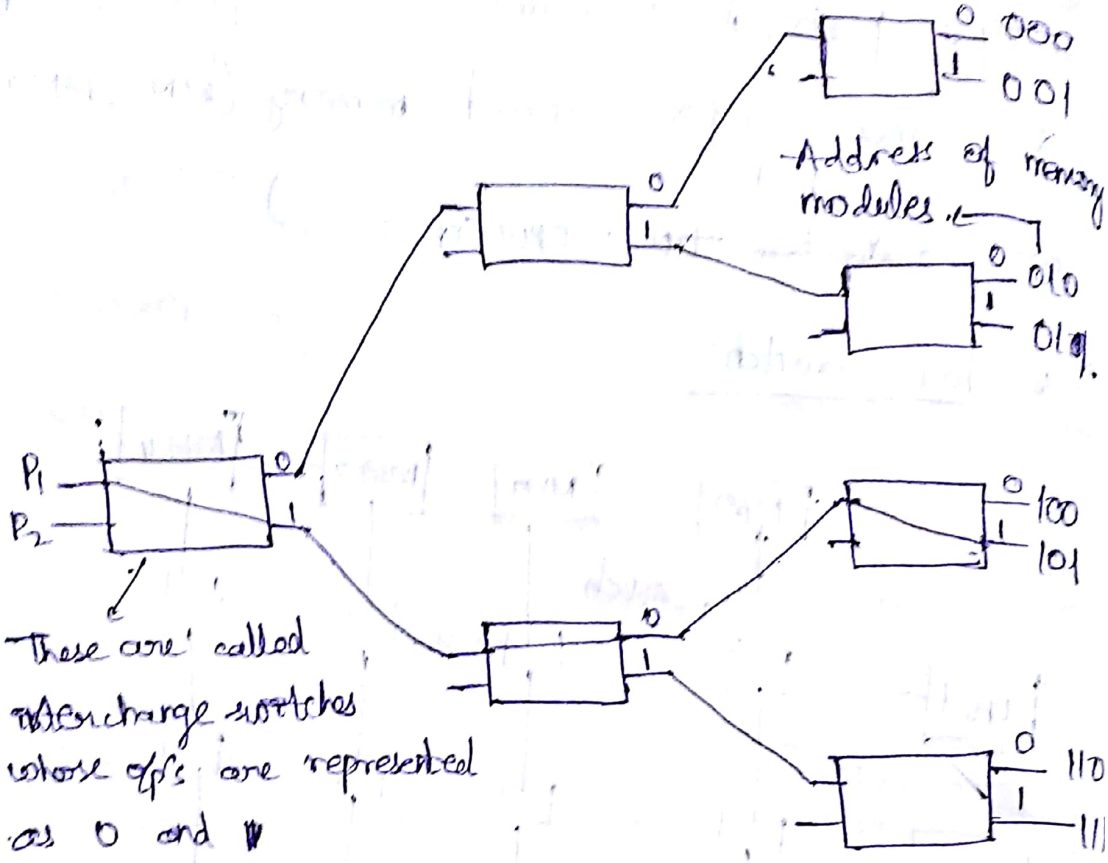
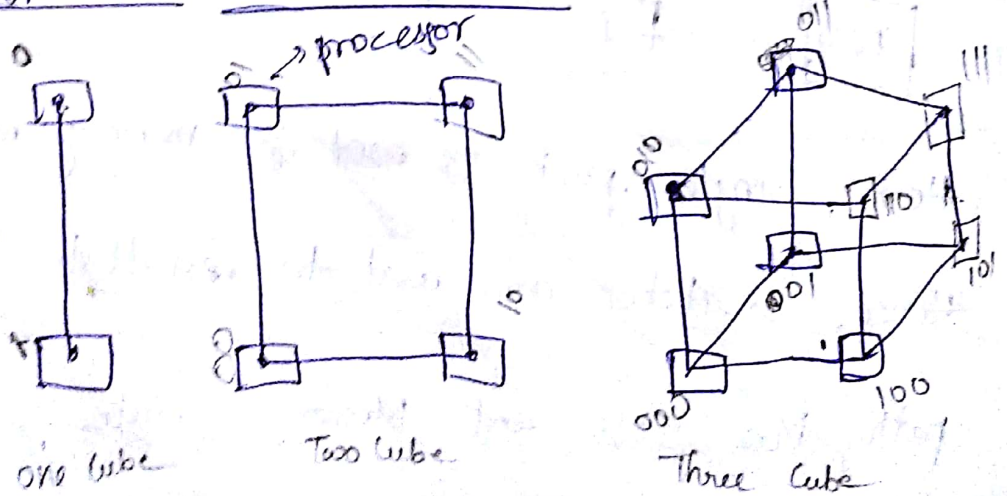


Fig: Binary tree with 2x2 switches. P_1, P_2 indicates the processors. If connection is to be established b/w processor and MM 101

5) Hypercube interconnection



indicates loosely coupled or distributed memory multiprocessor system

For 1 cube, 2^1 nodes.

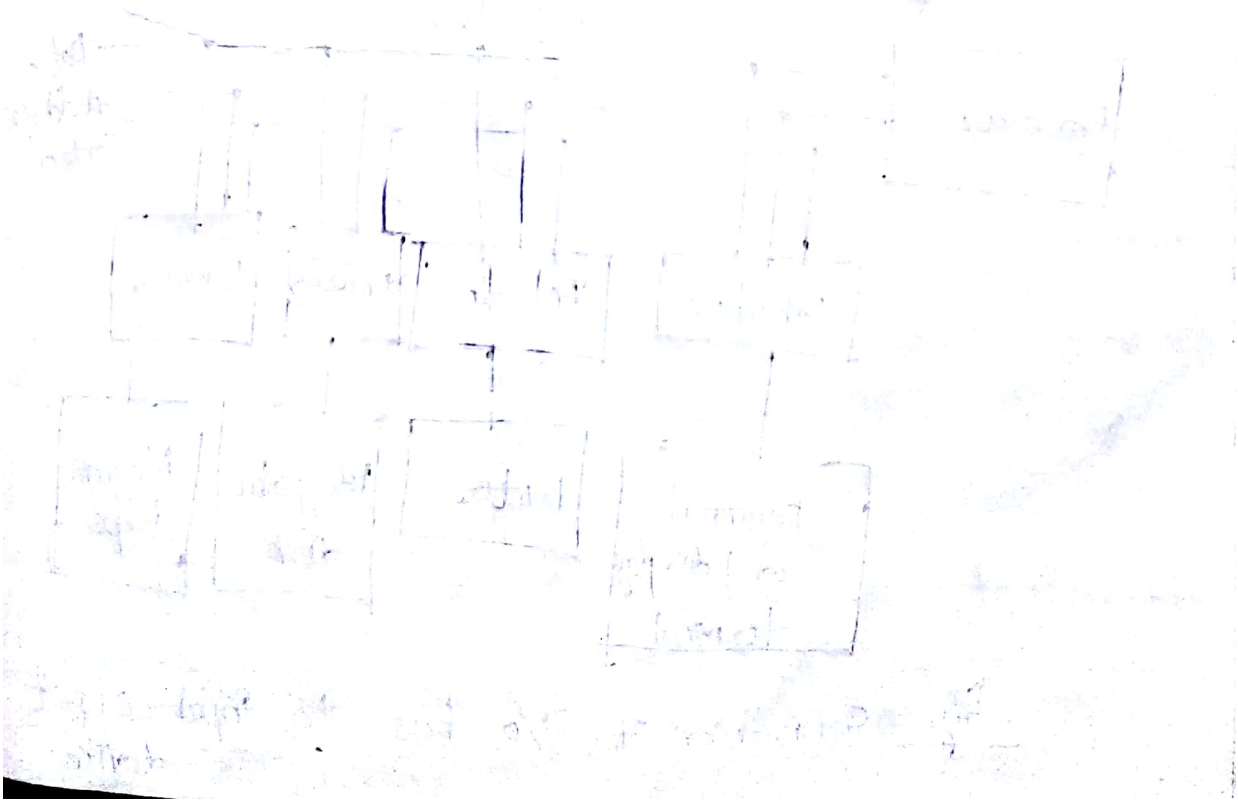
For n cube, 2^n nodes

Each node indicates 1 processor

For n cube, we are interconnecting 2^n processors.

00, 01, 10, 11 indicates address of nodes

Add. of successive nodes differ by only 1 bit)



4) Data Input command: It is used to transfer the data on the interface register into the data lines.

→ The I/O bus which is used b/w processor and I/O devices is called I/O bus. which contains data address and control signals. I/O bus b/w processor and memory → memory bus

→ There are 3 ways of using buses.

1) Using seperate bus for I/O and memory
2) Using one common bus " " " " with seperate control lines which is called as isolated I/O.
3) Using one common bus for I/O and memory with common control lines. ~~with common control~~ which is called as memory mapped I/O.

Modes of Transfer

Either data is transfered from input device to memory or from memory to o/p device. This can be done ~~though~~ with or without CPU.

→ To transfer data from i/p device to memory, it is transferred from i/p ~~u~~ u processor and from processor to memory.

My to transfer data from memory to o/p device, it is transferred from memory to processor and processor to o/p device

It is done ~~u~~ throug CPU.

It is one way to transfer

→ Second way is directly transfering data memory and I/O device which is called as direct memory access.

3 Modes of transfer:

1) Programmed I/O
2) Interrupt initiated I/O } through CPU

3) Direct Memory access (DMA) → without CPU

1) In programmed I/O, as CPU is fast and I/O is slow, CPU has to continuously check whether the device is ready to perform data transfer. By this, CPU time is wasted, which means that, we are not efficiently using CPU. 2) To overcome this interrupt initiated I/O is used.

In which we are using interrupt facility where the interface will be interrupting the CPU whenever the device is ready to perform data transfer. Meanwhile, CPU can be used to execute another program. In this CPU need not continuously check whether the device is ready or not.

2) DMA — We are directly transferring data blw memory and I/O.

12/10/17

Priority Interrupt:

To perform the data

1) Programmed
2) ~~Programmed~~
3) ~~Interrupted in~~ 2) ~~Interrupt initiated~~
2) Interrupt initiated I/O

→ In programmed I/O, CPU has to check whether I/O device is ready to transfer data or not. This is the drawback.

This can be ~~com~~ overcome.

The ~~time~~ delay can be reduced.

Direct Memory Access:

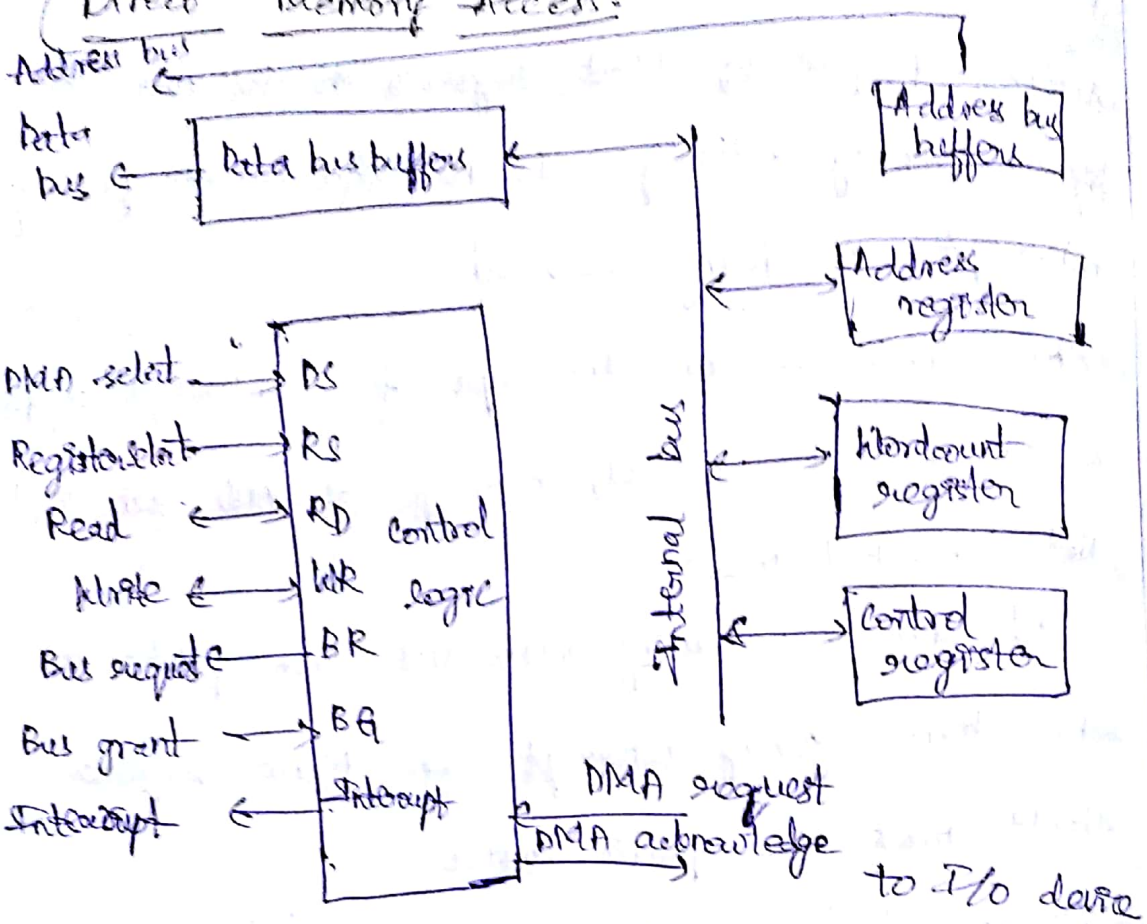


Fig Block diagram of DMA controller

It is one of modes of memory transfer b/w memory and I/O.

→ In DMA, without the intervention of CPU, directly data is transferred b/w I/O and memory. For DMA directly direct access, DMA controller is used.

→ There are 3 registers, which are initially (before performing DMA transfer) initialized by CPU.

1) Address register

→ To read (from memory → o/p) and write (i/p memory)

→ It indicates the starting location of memory from which we have to read or write

2) Word count registers

→ It indicates no. of words that have to be read/store into memory during read/write operation.

3) Control register:

→ It specifies the mode of transfer i.e. whether read or write. and also to start the DMA transfer.

→ When there is a request from I/O device to perform DMA transfer, (Initially memory buses are under control of CPU).

DMA controller should get the control of memory buses by sending bus request signal to the CPU.

→ When bus request signal is sent, CPU terminates the execution of instruction (at that instant) and keeps the memory buses in high impedance state, which means that the memory buses are disconnected from CPU.

→ Then, bus grant signal is sent by CPU (which means memory buses are under control DMA controller)

→ During initialisation of the 3 registers DMA select pin and register select are enabled

↔ Read and write pins are bidirectional i.e. they act as i/p and o/p

These pins act as i/p pins to CPU and o/p pins to memory

→ ~~When data is to be loaded into~~ Inorder to load data into 3 registers. By CPU, Read/write pins are enabled.

→ After each word transfer, wordcount becomes zero

→ When all words are transferred from wordcount register to I/O device & memory, interrupt is enabled and wordcount will be zero.

→ In order to read the wordcount register, Read is enabled i.e CPU sends the read signal.

→ DMA acknowledge will be generated to I/O device by DMA controller

When I/O device receives DMA acknowledge, if it is an o/p device, it will place the data on the data bus. i.e, if it is a o/p device, it will take the data from data bus and word count register will be decremented. If the wordcount value ≠ 0, I/O device will send a DMA request and DMA controller check whether there is a DMA request → If the I/O device is fast it will send a DMA request as soon as previous word is transfered and vice versa.

→ In case of slow I/O devices, DMA controller disables BR signal. and bus grant signal is disabled by CPU. i.e. memory buses are under control of CPU. and the CPU executes the program where it has stopped. This is called as cycle stealing.

handing over the memory buses to CPU after each word transfer in case of slow I/O device is called cycle stealing.

It does not occur in case of fast I/O devices. DMA controller will handover the control of memory buses only after transfering all the words. This is called burst transfer

→ The 3 registers are initialised by CPU through data bus. ex, word count register value is read by CPU through data bus. ✱

After completion of all words, Interrupt is sent to CPU by DMA Controller. CPU reads value of word count register => all words are transferred.

* Input-output Processor:

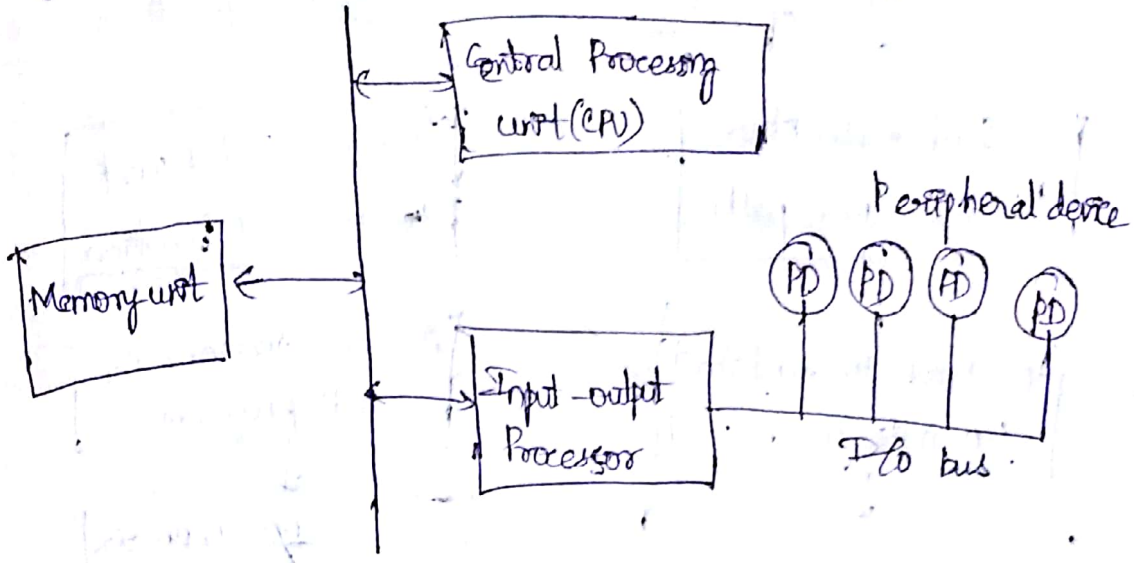


fig: Block diagram of a computer with I/O processor

→ Purpose of Input-output processor is to perform data transfer b/w Memory and peripheral devices (I/O devices) i.e. Memory → IOP, IOP → I/O (or) I/O → IOP, IOP → Memory unit

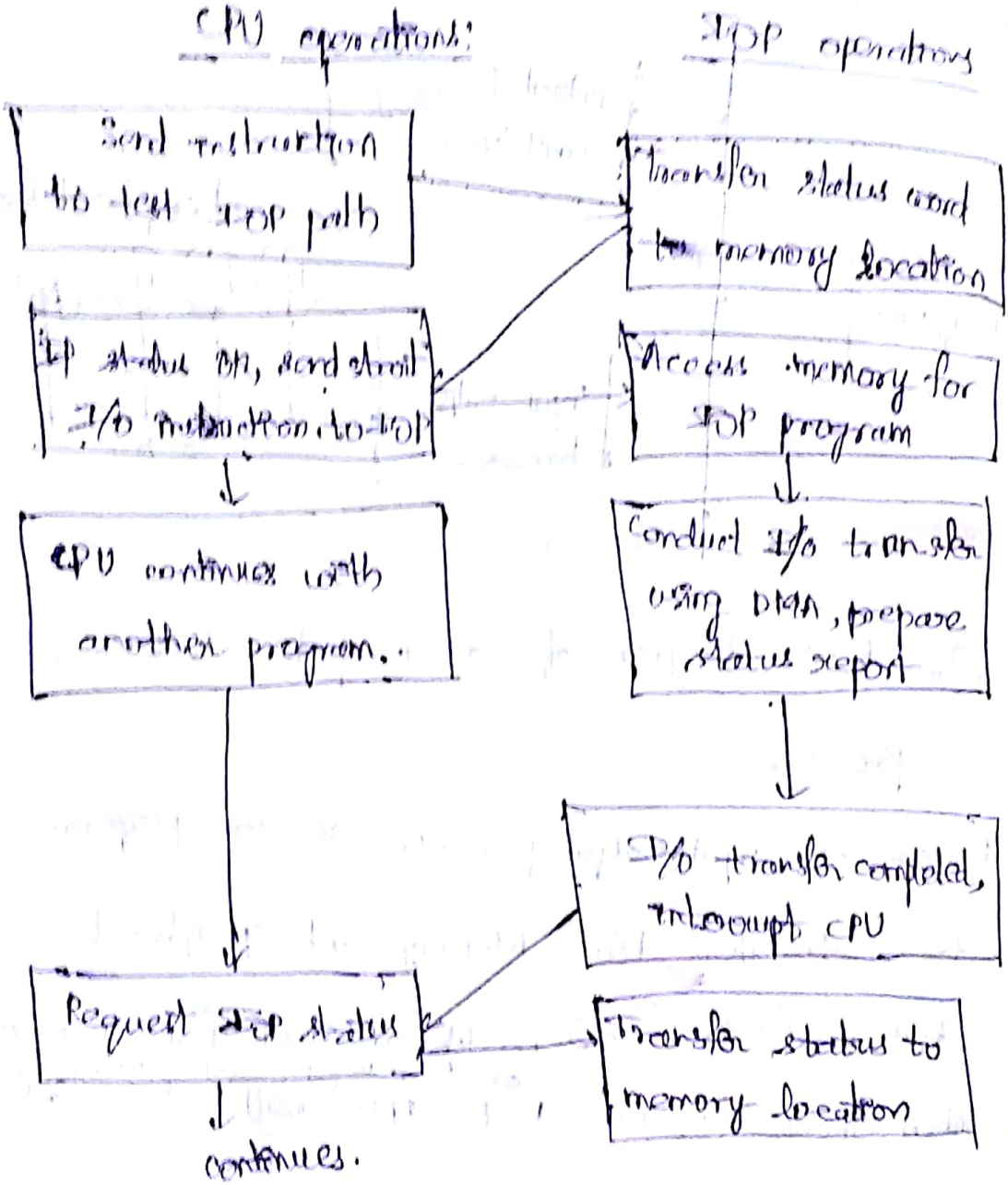
→ When there is no IOP, CPU will transfer data from memory to I/O device.

→ When IOP is performing data transfer, CPU can be used to execute another program

→ CPU sends information to I/O. IOP to know whether the I/O device

is ready or not.

CPU-IOP communications



→ CPU sends info instruction to IOP to know whether the I/O device is ready to perform data transfer or not. In response to instruction sent by CPU, IOP sends status words to some location in memory

which indicates whether the I/O device is ready to perform data transfer.

→ CPU reads the status words in memory and knows whether the device is ready to perform data transfer

→ If status word is ok → I/O device is ready

→ Then CPU sends start I/O instruction to IOP

→ When IOP receives start I/O instruction, IOP accesses IOP program from memory.

→ While IOP performs IOP program, CPU can be used to execute another program.

→ Status report indicates whether there is any error are occured during data transfer

→ After the completion of data transfer, IOP interrupts CPU. The purpose of interrupting CPU is to inform CPU that data transfer is completed. Then CPU requests the IOP status report from IOP and that status report is sent to memory location. From memory location, CPU reads the status word (report)

→ After reading status words, CPU knows whether any error has occured or not.

Interrupt initiated I/O

→ In this I/O, I/O interrupts the CPU when the I/O device is ready to perform data transfer.

→ When CPU is interrupted, it executes interrupt source routine (ISR) which is nothing but I/O program.

→ To know the address of ISR, there are 2 ways.

1) Vectored interrupt
2) Nonvectored interrupt.

→ In [illegible], the branch address (the address to which we have to jump in order to execute ISR) is contained in some fixed location in the memory. ∴ CPU has to take the branch address from memory.

→ In vectored interrupt, the branch address is provided by interrupt source. (one which is producing interrupt i.e. I/p or O/p device).

This branch ~~interrupt~~ information is called as interrupt vector.

∴ This interrupt vector indicates the ~~starting~~ address of the interrupt service routine i.e. I/O program which is used to perform I/p or O/p operation. (or) we get the address of ~~address~~ starting address of ISR in that address (direct or indirect)