28/11/17

UNIT-III

COMPUTER ARITHMETIC

Addition & Subtraction:

Addition & subtraction with signed-Magnitude Data.

Operation	Add Magnitudes	Subtract Magnitudes		
		When A > B	A < B	A = B
(+A) + (+B) ↙ Augend ↓ Addend	+(A+B)			
(+A) + (-B)		+(A-B)	-(B-A)	+(A-B) =0
(-A) + (+B)		-(A-B)	+(B-A)	0 +(A-B)
(-A) + (-B) ~~(-A) - (+B)~~ ~~↙ Minuend ↘ subtrahend~~	-(A+B)			
(+A) - (+B) ← Minuend, → subtrahend		+(A-B)	-(B-A)	+(A-B)
(+A) - (-B)	+(A+B)			
(-A) - (+B)	-(A+B)			
(-A) - (-B)		-(A-B)	+(B-A)	+(A-B)

→ For add operations, we add the magnitudes

when the signs are same.

→ For subtract operations, use add Magnitude when the signs are different

→ When the magnitudes are added, the sign of the result will be same as that of A.

→ For Add, we subtract magnitudes if signs are different.

→ For subtract, we subtract magnitudes if signs are same

→ For A>B, condition, the sign of result is same as sign of A

→ In A<B condition, sign of result is complement of sign of A.

→ In A=B condition, the sign of result is always +ve.

HARDWARE SYSTEM (for Addition & Subtraction)

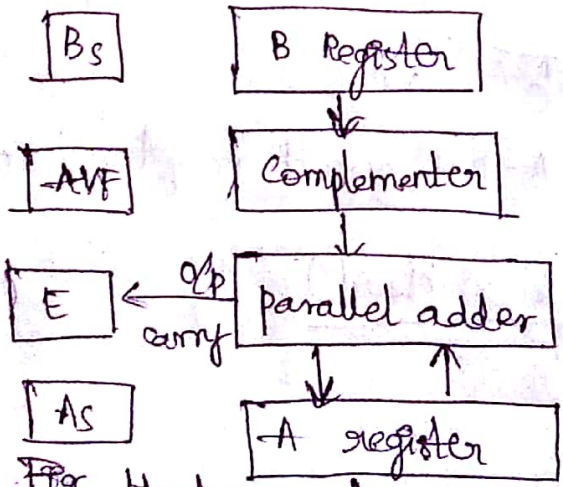


Fig Hardware for signed magnitude addition & subtraction

Hardware system is used for addition and subtraction

→ For addition, A register & B registers contains operands and A_s & B_s flipflops indicates sign.

→ For subtraction, A register - minuend, A_s - sign of minuend, B " - subtrahend, B_s - sign of subtrahend.

→ To perform A+B, M=0

→ To perform A-B, A+$\bar{B}$+1 is performed.

To perform A+$\bar{B}$+1, M=1

→ After adding A & B, carry goes to E-Flipflop and Result is stored in E & A register

If E=1 ⇒ Add overflow (AVF)

↘ o/p carry

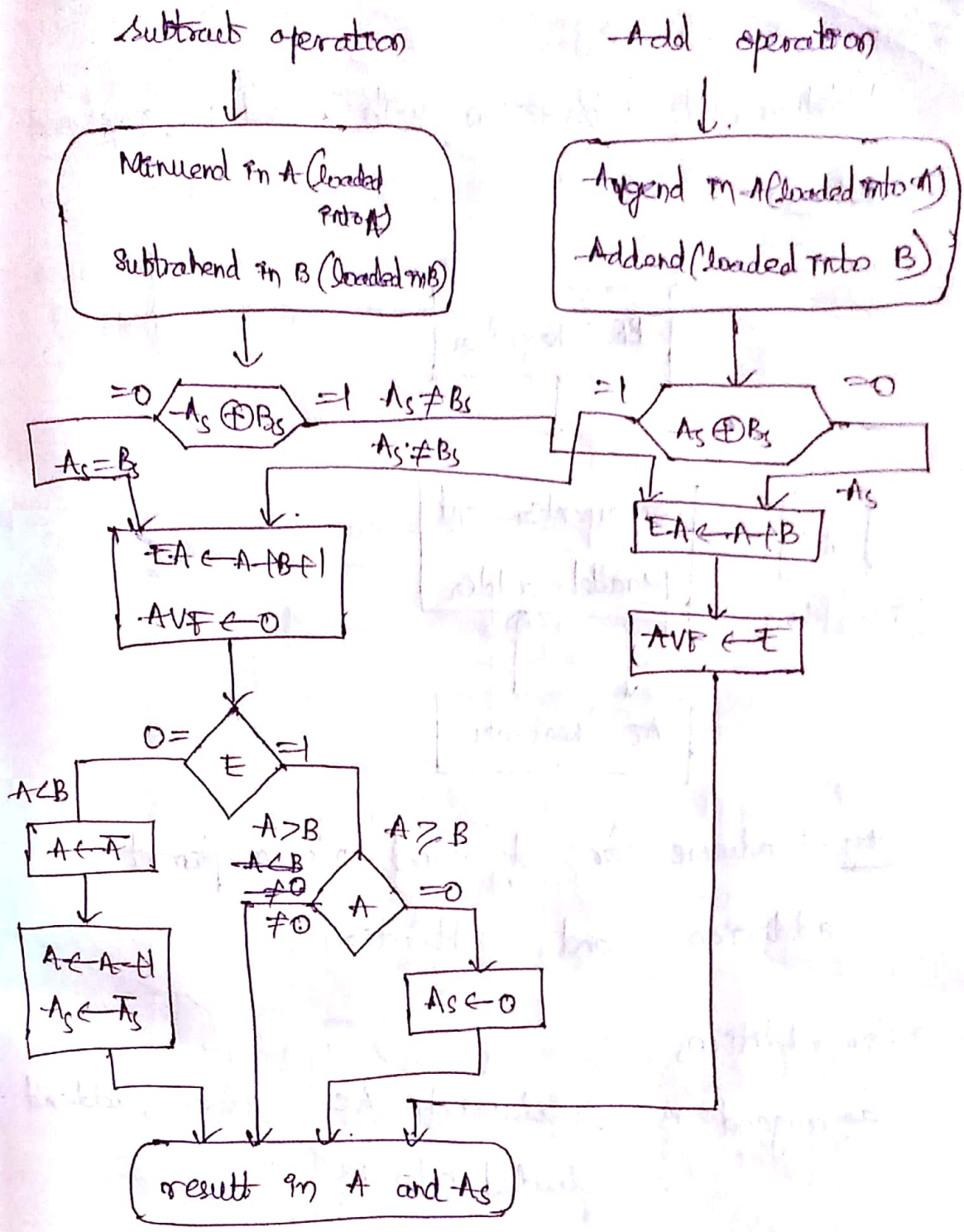
→ To subtract the magnitudes, perform A+$\bar{B}$+1

When M=1 → o/p carry & B is complemented

∴ o/p is A+$\bar{B}$+1 ⇒ A-B

After doing A+$\bar{B}$+1, result is stored in E,

Flow chart (for add & subtract)



30/8/17

Addition and subtraction with signed-2's complement Data :

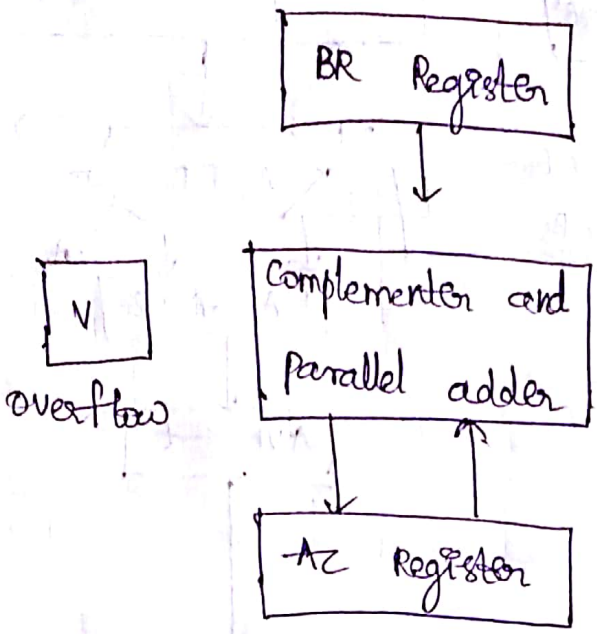


Fig Hardware for signed-2's complement addition and subtraction

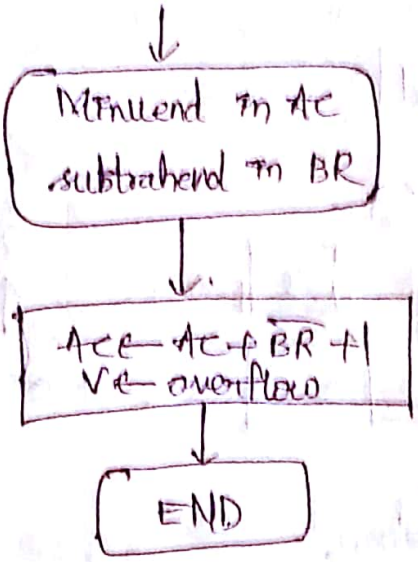
→ For addition, augend is loaded into AC register, addend is loaded into BR.

Then AC and BR are added and the result is stored in AC. If overflow occurs, overflow flip flop V is set.

→ For subtraction, minuend is loaded into AC and subtrahend is loaded into BR.

BR is complemented (2's complement) and it is added to AC. The result AC+$\overline{BR}$+1 (AC−BR) is stored in AC. V will be set if overflow occurs.

Flowchart for common hardware subtract



Add

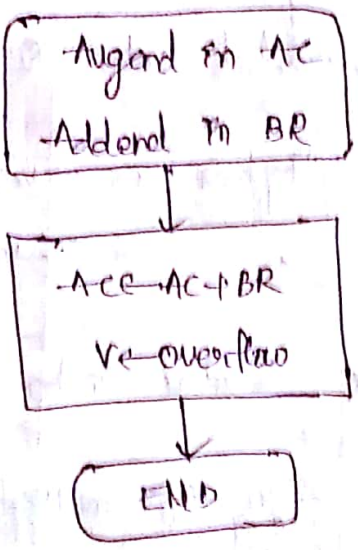
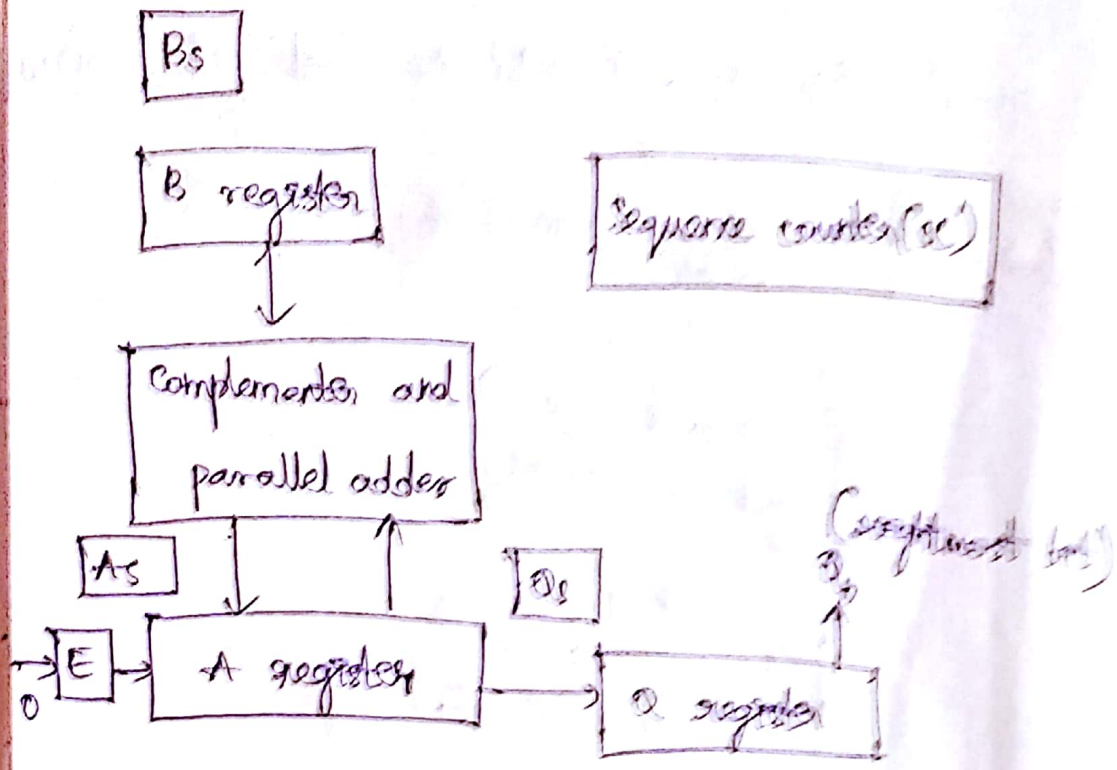


Fig: Algorithm for adding and subtracting numbers in signed 2's complement representation.

Multiplication Algorithm:



In multiplication,

first operand → Multiplicand

second " → Multiplier

→ Multiplier is loaded into Q register

$Q_s = 0$ if multiplier is +ve

$Q_s = 1$ " " " -ve

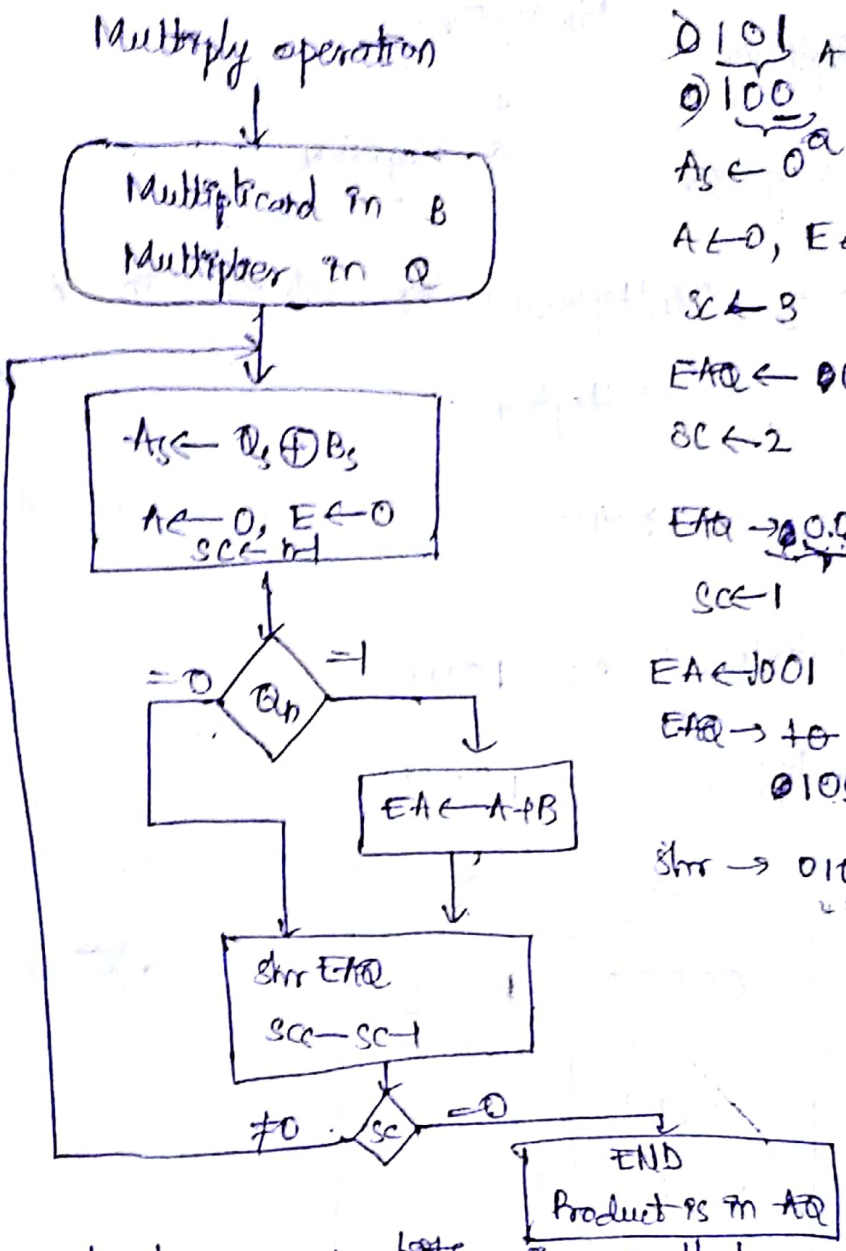
→ Multiplicand is loaded into register B

B_s is the sign of multiplicand

→ Result of Multiplication is stored in A & Q registers, A_s → sign of result

→ SC contains no. of bits in multiplier excluding sign bit.

Algorithm:



0101 A
0100 Q
$A_S \leftarrow 0$
$A \leftarrow 0$, $E \leftarrow 0$
$SC \leftarrow 3$
$EAQ \leftarrow 00000010$
$SC \leftarrow 2$
$EAQ \rightarrow 00000001$
$SC \leftarrow 1$
$EA \leftarrow 1001$
$EAQ \rightarrow$
01001001
shr → 01001001

'n' indicates no. of bits in multiplier including sign bit

After adding A and B, if carry is generated, it is stored in E.

shr → logic right (Ashr → Arithmetic shift right)

shr EAQ → indicates E is loaded into A and LSB of E is loaded into MSB of Q

Eg:

23 × 19 → Multiplier

↓ Multiplicand

↓ B register

Q register

Sign of Multiplicand is stored in A_S B_S

" " Multiplier " " " A_S

Sign of result " " " A_S

Multiplicand B = 10111

Multiplier Q = 10011

	E	A	Q	SC
Initial	0	00000	10011	101
$Q_n = 1$; add B		10111		
First product	0	10111		
shr EAQ	0	01011	11001	100
$Q_n = 1$; add B		10111		
Second partial Product	0	00010		
shr EAQ	0	00001	00111	011
$Q_n = 0$ shr EAQ	0	00100	10011	010
$Q_n = 0$; shr EAQ	0	00010	01011	001
$Q_n = 1$; add B		10111		
shr EAQ	0	11011	01011	
	0	01101	10101	000

$437 = 0110110101$

The value EA before performing shr operation is called partial product.

Booth Multiplication Algorithm:

The algorithm used for performing multiplication of signed 2's complement numbers is called Booth multiplication algorithm

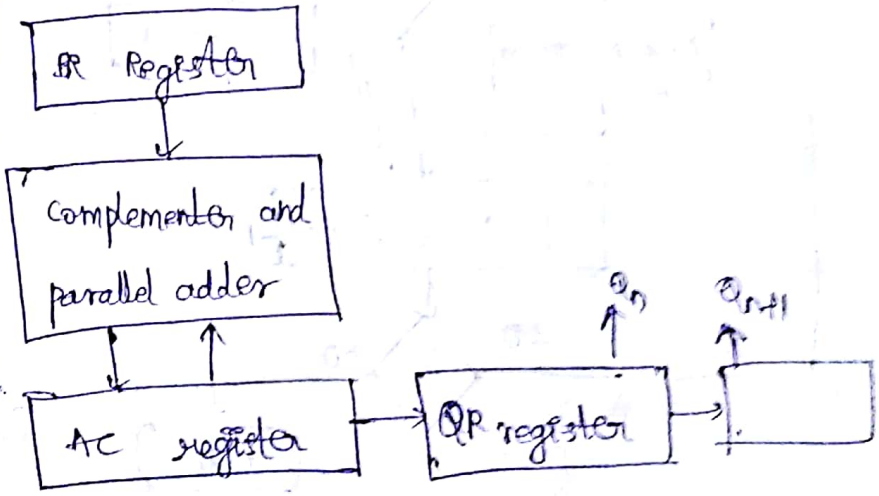


Fig. Hardware for Booth algorithm

QR contains the multiplier in 2's complement form

BR " " multiplicand "

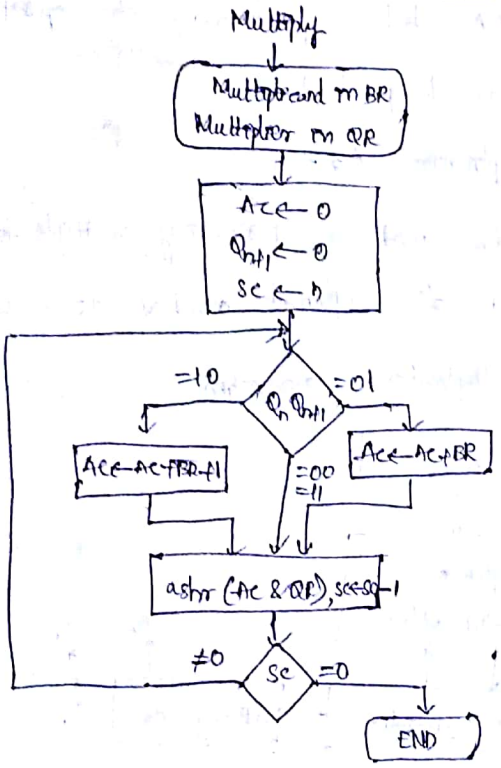
→ SC contains the no. of bits in QR register.

QR & BR contains the sign bits also

→ After multiplication, result is given AC & QR

→ The bit coming out from QR is goes into Q_{n+1}

Flowchart for booth Multiplication:



$Q_n Q_{n+1}$
10
11
01
00
10

Higher Example of multiplication with Booth algorithm:

(-9) x (-13) = 117

Multiplicand ↓ Multiplier →

BR = 10111

BR+1 = 01001

$Q_n Q_{n+1}$ =

-9 → 01001 → -13

10110

10111

-13 → 01101

00010

QR = 10011

$Q_n Q_{n+1}$	AC	QR	Q_{n+1}	SC
Initial	00000	10011	0	101
Subtract BR	01001			
	01001			
ashr	00100	11001	1	100
ashr	00010	01100	1	011
Add BR	10111			
	11001			
ashr	11100	10110	0	010
ashr	11110	01011	0	001
Subtract BR	01001			
	00111			
ashr	00011	10101	1	000

Result is stored in AC & QR

Array Multiplier (Fast multiplier)

Consider 2-bit by 2-bit multiplier

$$\begin{array}{r} b_1 \quad b_0 \\ a_1 \quad a_0 \\ \hline a_0 b_1 \quad a_0 b_0 \\ a_1 b_1 \quad a_1 b_0 \\ \hline C_2 \quad C_1 \quad C_0 \end{array}$$

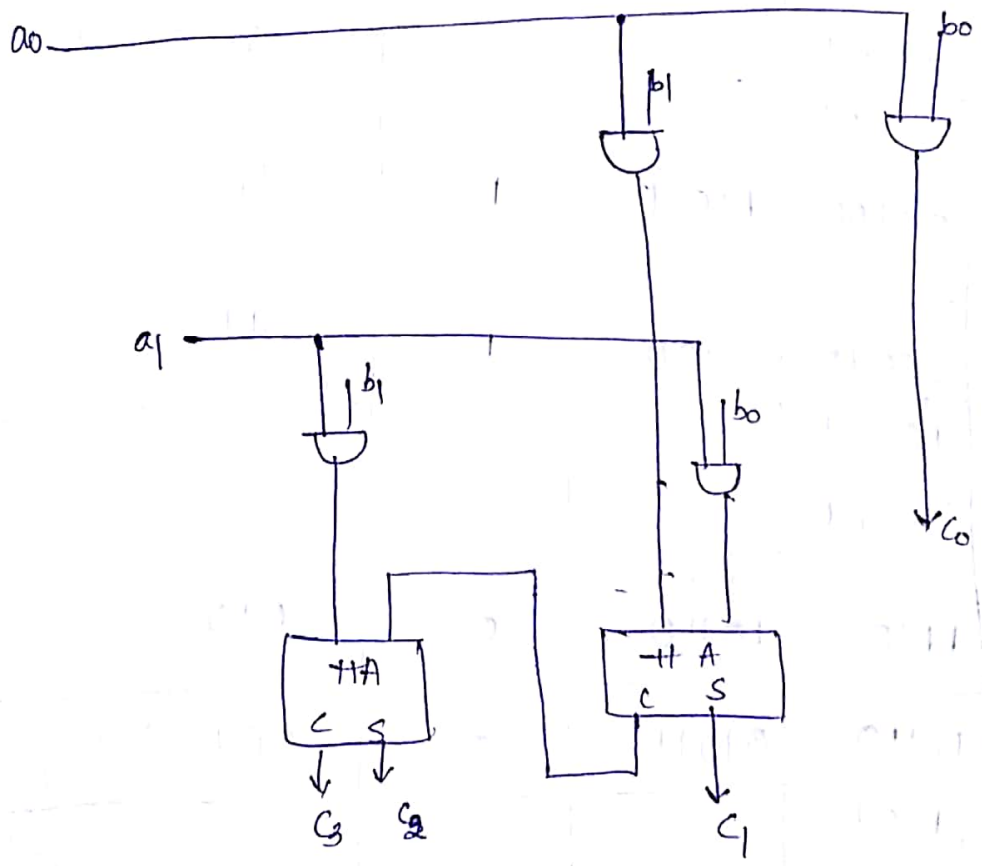


Fig: 2-bit by 2-bit Array Multiplier.

→ It is faster when compared to other multipliers

→ For j-multiplier bits and K-multiplicand bits, we need jxk AND gates and

$j-1$, 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

↳

$b_3\ b_2\ b_1\ b_0$
$a_2\ a_1\ a_0$

~~$b_3\ b_2\ b_1\ b_0$~~

~~$a_2\ a_1\ a_0$~~

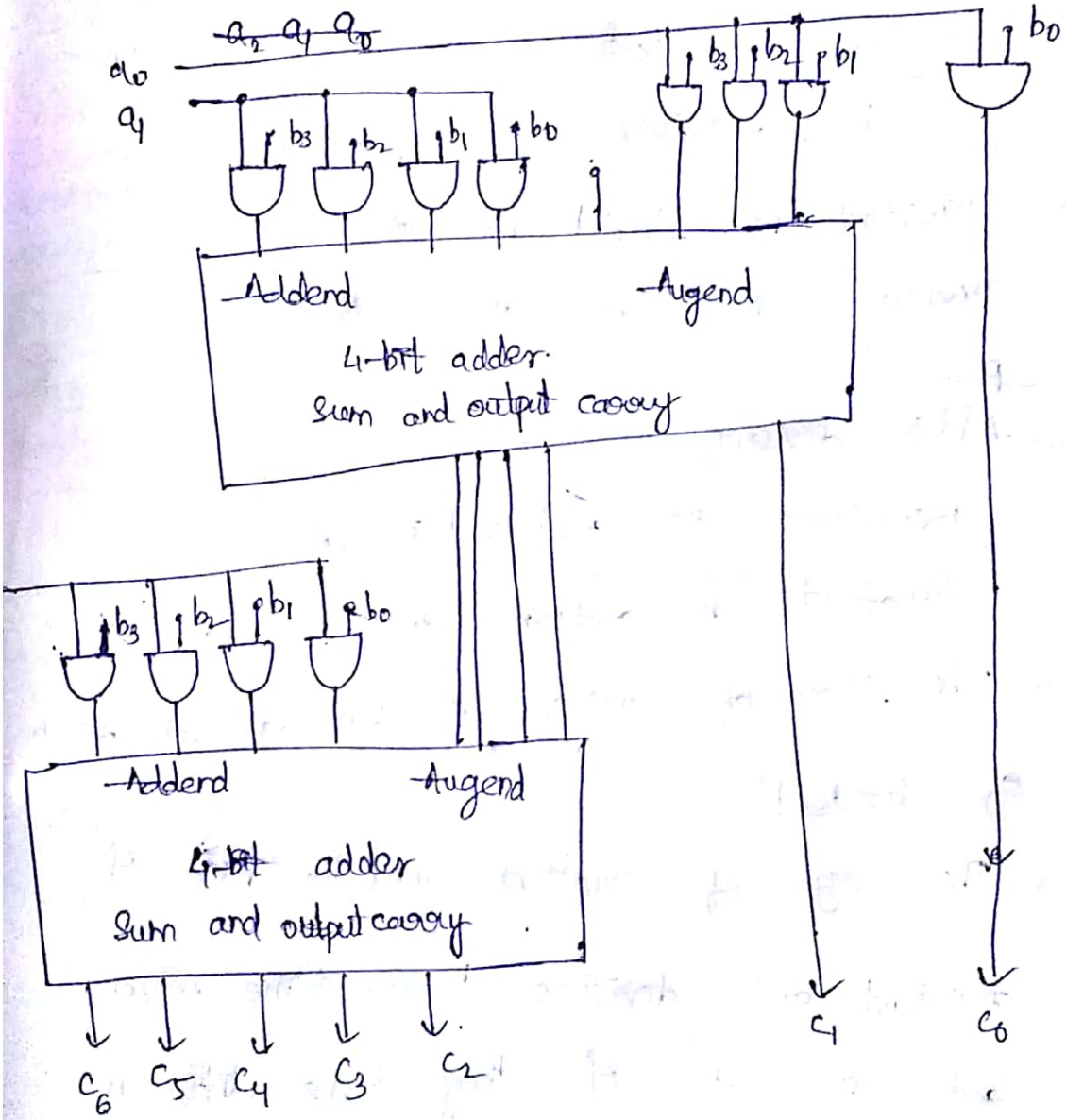


Fig: 4-bit by 3-bit array multiplier.

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

Division Algorithm:

→ Algorithm for division of signed magnitude numbers

Nr → Dividend
Dr → Divisor

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

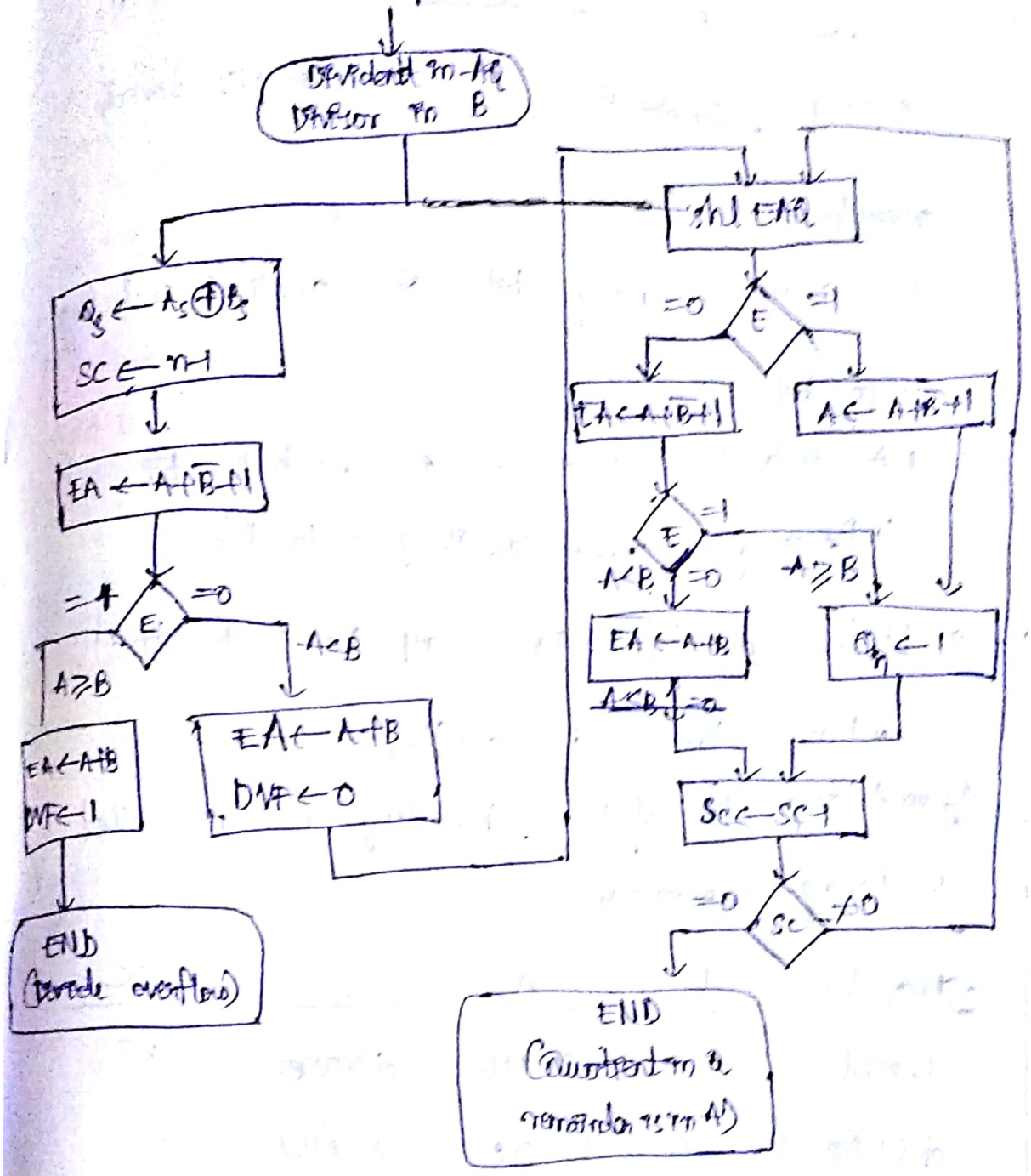
→ After division,
Remainder is contained in A
Quotient is contained in register Q

→ 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 division is in Bs
" " dividend " " As

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- FF (carry is generated, it goes to E)

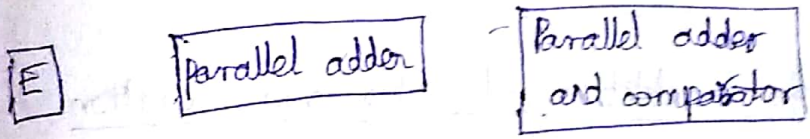
→ Divide overflow FF will be set to 1 when there is overflow.

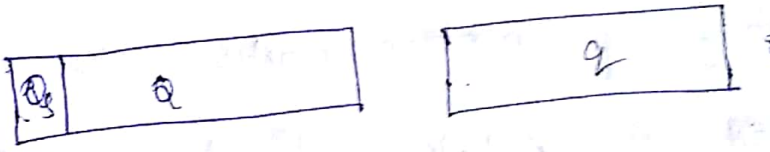
Again Adding B after subtracting B is called restoring remainder.

Initial	E	A	Q	SC
Initial		01110	00000	5
shl EAQ	0	11100	00000	
add B+1		01111		
E=1	1	01011	00001	
Set Q_n=1	1	01011	00001	4
shl EAQ	0	10110	00010	
add B+1		01111		
E=1	1	00101		
Set Q_n=1	1	00101	00011	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

A_q – MSB of the magnitude of mantissa.

B_s, B, q indicate the registers

→ For addition operation

AC contains augend

BR " addend and result is stored in AC

→ For subtract operation

AC contains minuend

BR " subtrahend and result is stored in AC

→ Register QR is used for division and multiplication operation.

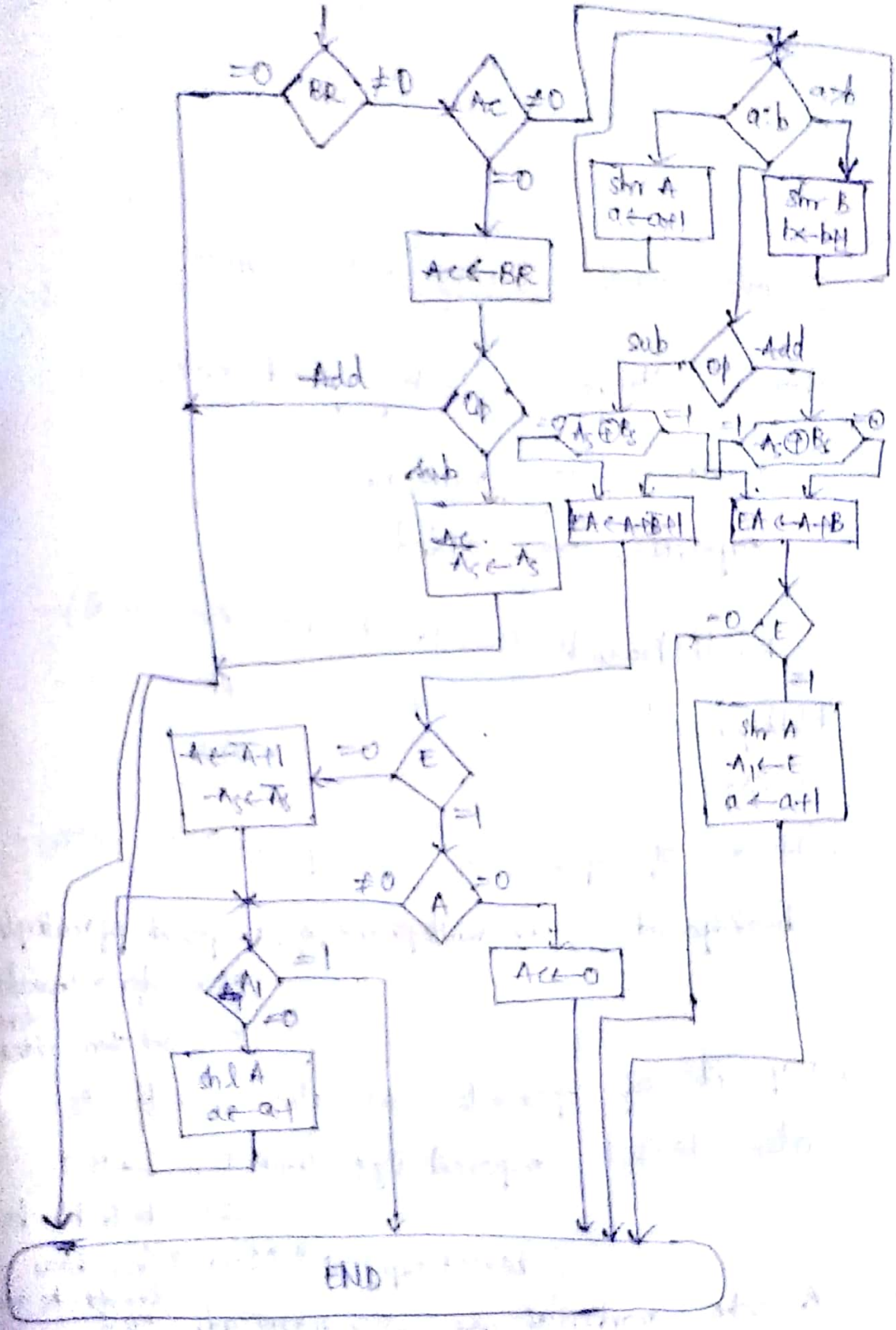
Algorithm for addition and subtraction operation (Floating point):

→ In floating representation, mantissa is represented in sign-magnitude form and exponent is represented in biased form

→ Biased exponent is always a +ve value which is given by adding +ve no. to the actual exponent.

For addition and subtraction operations, exponent values of the operands should be made equal

Add or subtract



If BR = 0, no operation is performed (i.e. AC ← AC, AC ← BR = 0)

When exponent is incremented, mantissa is shifted right

" " decremented " " " left

Note:

Multiplication of floating point numbers:

→ For multiplying floating point nos,
mantissas are multiplied
exponents are added

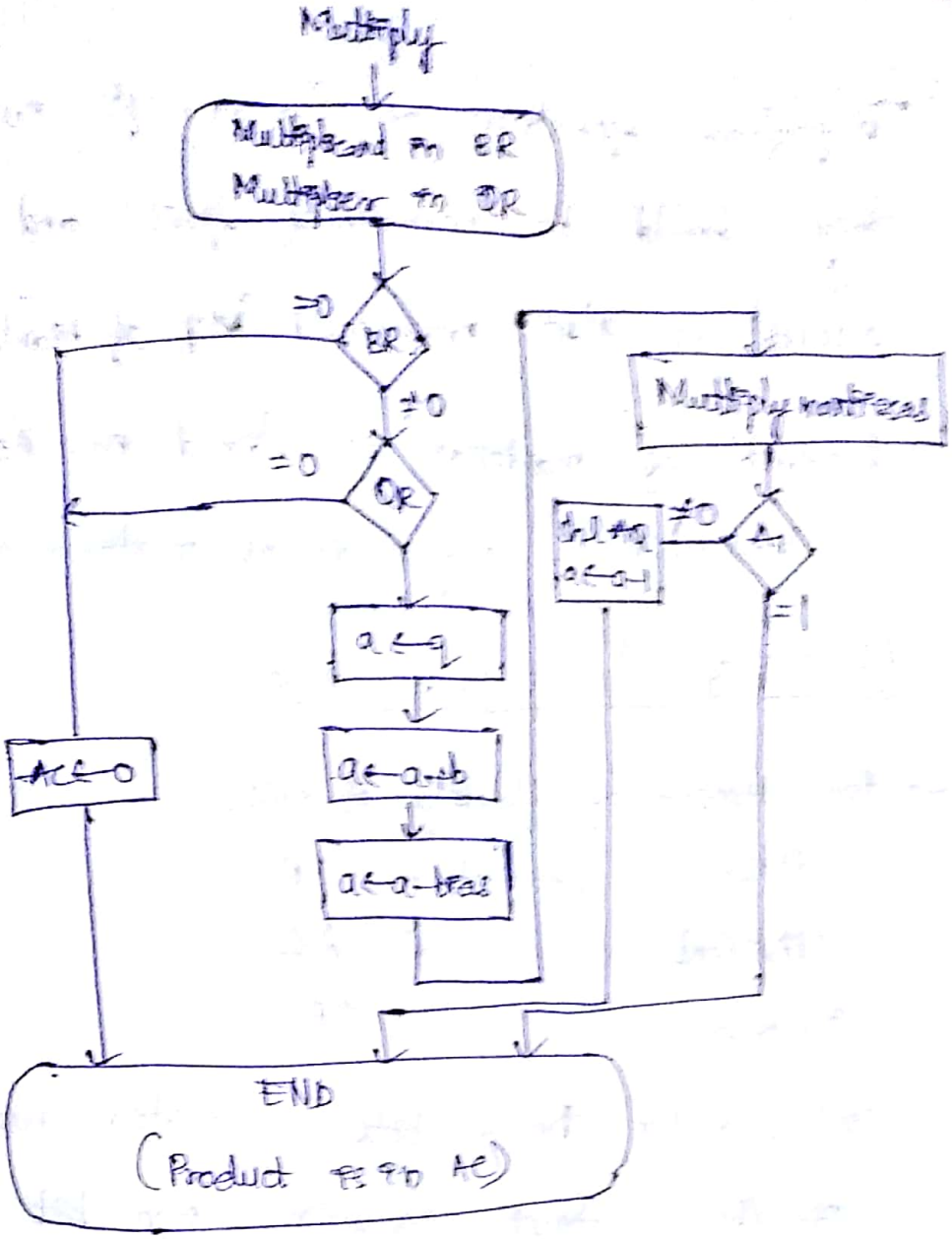
Result (Product) is stored in AC (AC, BR)
Multiplicand " " " BR
Multiplier " " " QR

→ If one of operand is zero, product is zero

→ Exponent of the multiplicand, q → exponent of multiplier
↳ It is first transformed to 'a' and then added.

→ Exponents of operands and also result is also biased exponent. Eg: 4+128, 2+128
261 → to double bias
biased exp → 4+2+128 (one bias should be removed)

→ As the mantissas are sign-magnitude nos, multiplication is same as " "



(It is shifted left only once ∵ Eg: 0.1×0.1
= 0.01)

To perform operation on floating pt. no's, they should be normalised first and result is also normalised (MSB of Mantissa = 1)

Product of mantissas is stored in AQ

AQ → MSB of magnitude of mantissa of result

Division of floating point numbers:

→ For division of floating pt no's,

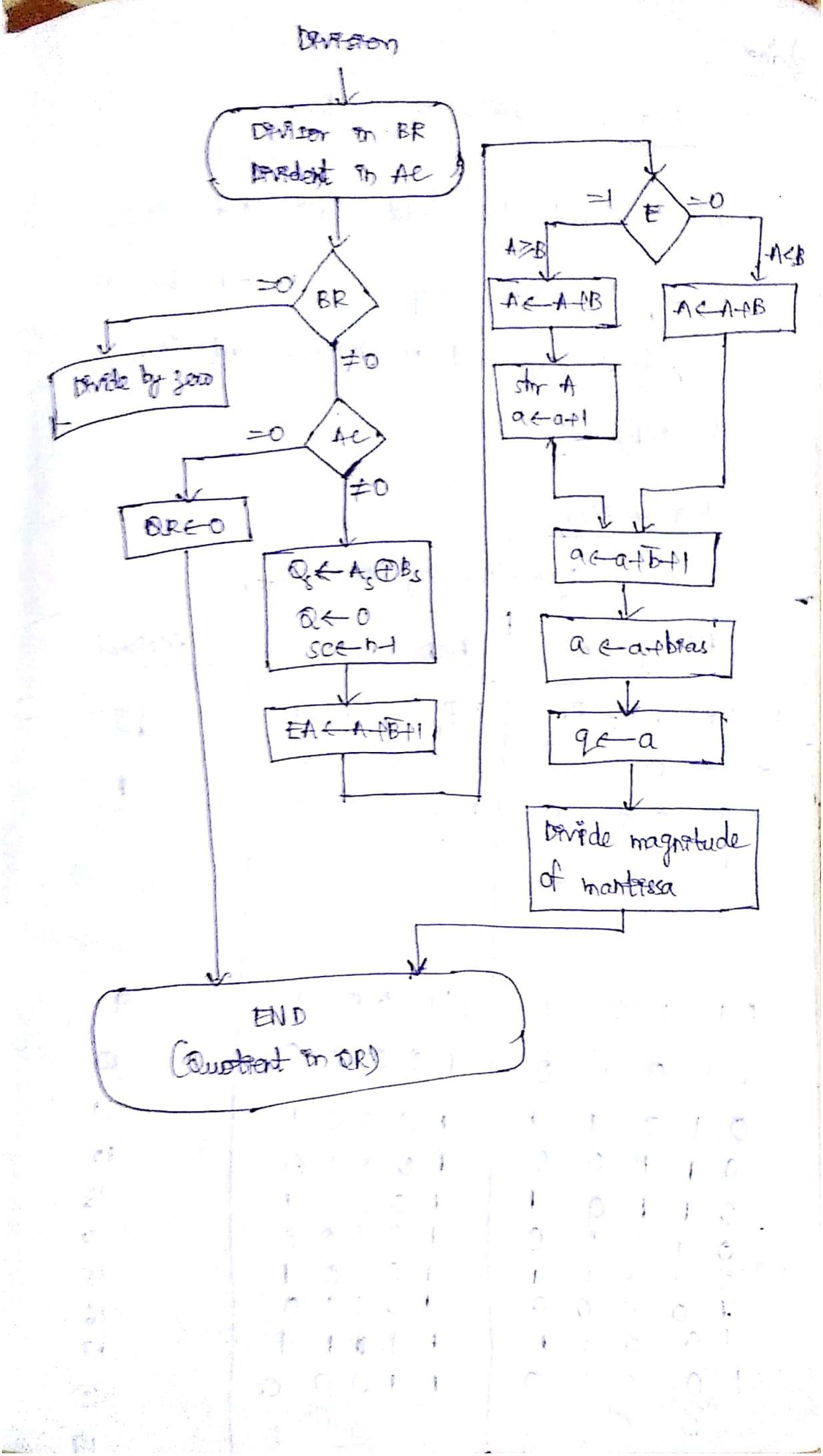
divisor is stored in BR

Dividend " " " AC

Quotient " " " QR

→ 'n' indicates no. of bits used for mantissa of the quotient including sign bit.

Overflow (divide) is avoided in floating point numbers by performing certain operations (i.e. shift right)



8/9/19

Decimal Arithmetic unit

It is used to add two BCD nos.

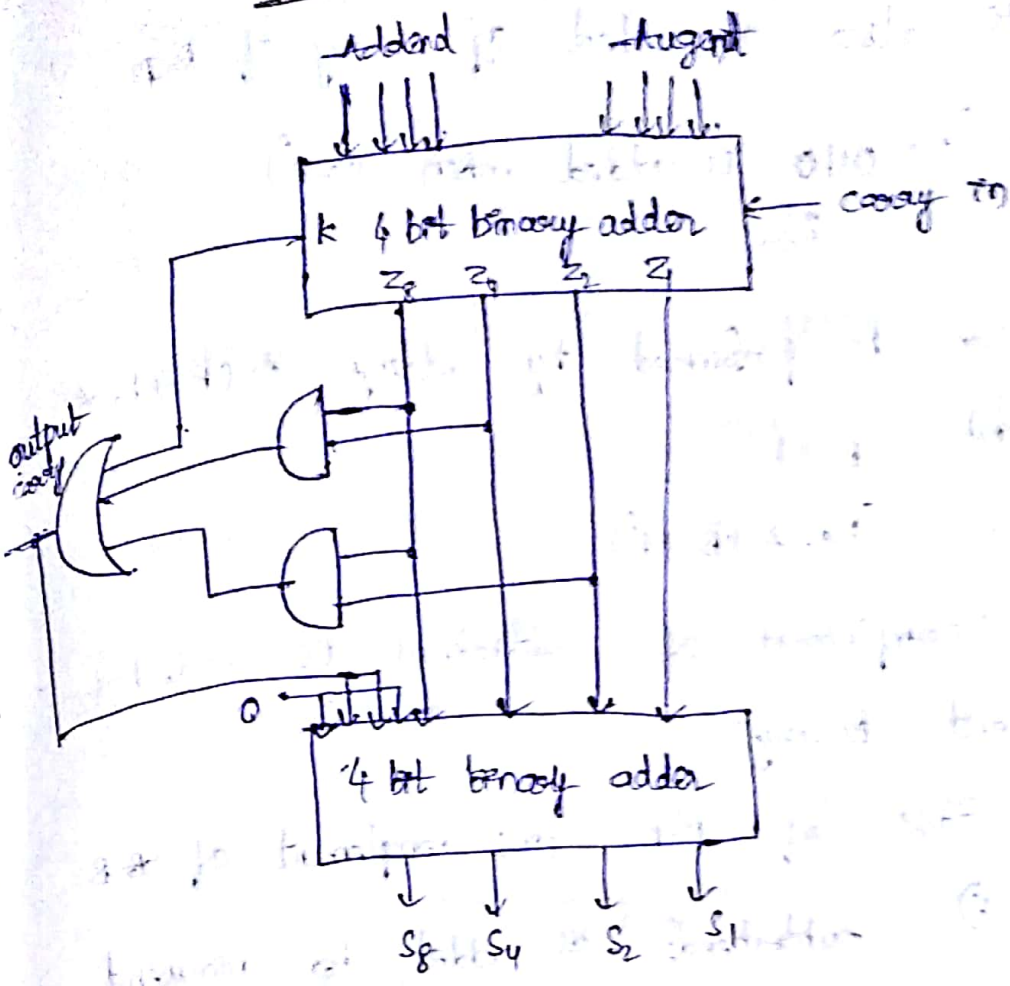
(Result is also expressed in BCD, but it is first expressed in binary and then converted to BCD)

BCD Adder:

Observation of BCD Adder

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	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

BCD Adder:



In addition, first operand is augend, second operand is addend.

Binary and BCD codes are same from 0 to 9

Binary is converted into BCD when decimal > 9

i.e (i) when $K=1$ (or) $\rightarrow$ 1st i/p of OR gate

(ii) $Z_8 Z_4 = 11$ (or) $\rightarrow$ 2nd i/p of OR gate

(iii) $Z_8 Z_2 = 11$ $\rightarrow$ 3rd " " "

Binary is converted into BCD by adding 6 (0110)

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

When $C=1$, 0110 (6) is added to binary

→ C is also called of carry of BCD sum (∵ 0110 is added when C=1)

→ A-B can be performed by adding A + $\bar{B}$ + 1

and $\bar{B}$ + 1

(i.e. A + $\bar{B}$ + 1)

→ Two's complement of subtrahend is added to minuend (Binary)

In case of BCD, 10's complement of B ($\bar{B}$ + 1) (subtrahend) is added to minuend

BCD subtractor:

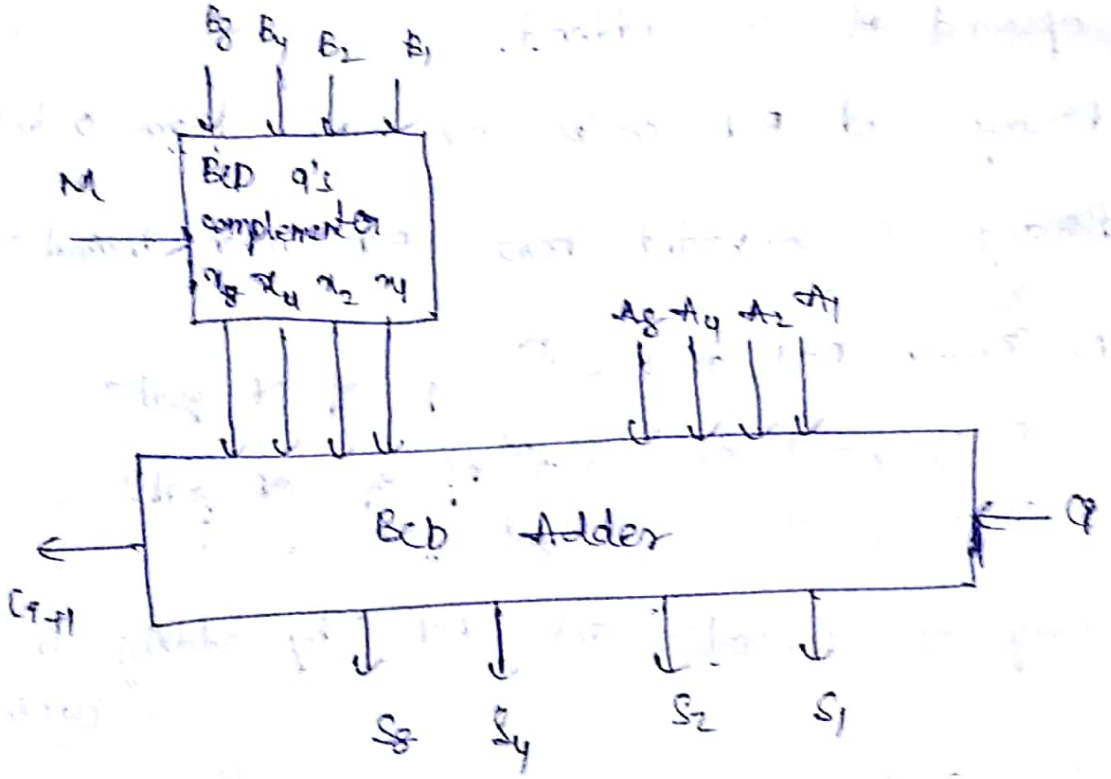


Fig: One stage of a decimal arithmetic unit

'n' stages are required for n-digit decimal no.

M = 0 for BCD addition.

M = 1 " " subtraction $(\overline{B}+1)$ $(\overline{B})$

for addition, A and B are added

for subtraction, A and 9's complement + 1 of B is added.

C_0 is the I/p carry which is always '1' for first stage of BCD subtraction $(\overline{B}+1)$

$C_i = 0$ for first stage of BCD addition

whereas for the other stages, it depends on the O/p carry of previous stages.

$x_1 = B_1 M' + B_1' M$

$x_2 = B_2$

$x_3 = B_4 M' + (B_4' B_2 + B_1 B_2') M$

$x_4 = B_8 M' + B_8' B_4' B_2' M$

$M = 0 \Rightarrow x_1 = B_1$, $x_2 = B_2$, $x_3 = B_3$, $x_4 = B_4$

$M = 1 \Rightarrow x_1 = B_1'$, $x_2 = B_2'$, $x_3 = B_3'$, $x_4 = B_4'$

→ The algorithm applied for sign magnitude binary no's can also be applied for signed BCD no's.

→ Similarly, algorithm for addition and subtraction of 2's complement of binary no's can also be applied for +/− of 10's complement of BCD

BCD Multiplication:

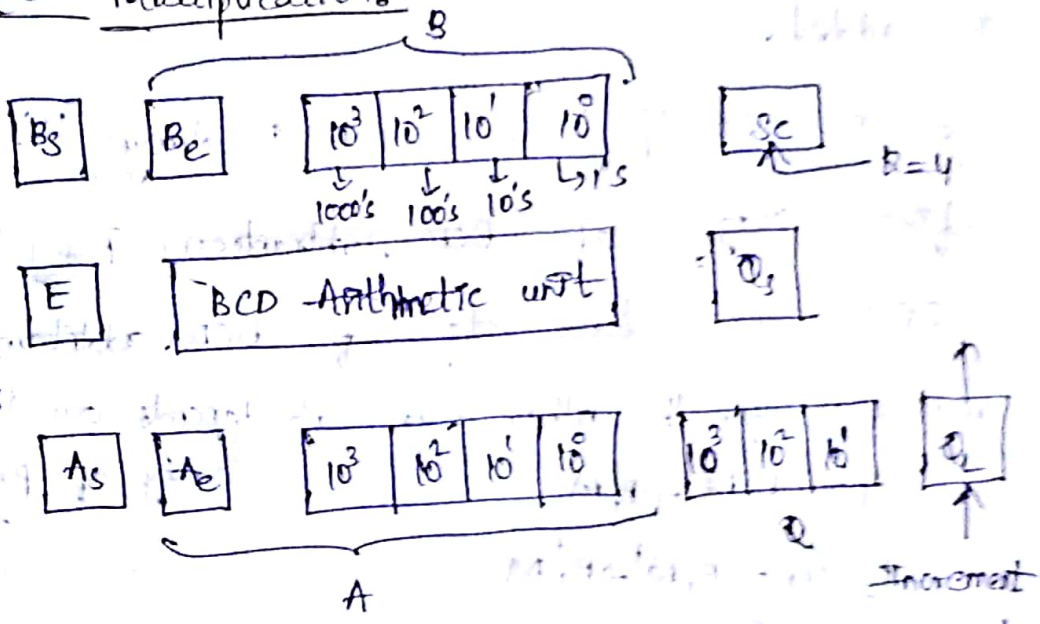


Fig: Register for decimal arithmetic multiplication and division

It is used for signed 4-digit BCD no's (16 bits)

Each digit is represented in 4 bits.

Multiplicand is stored in B

Multiplier " " " Q

A_e, B_e are extension bits.

→ For division, dividend in A & Q

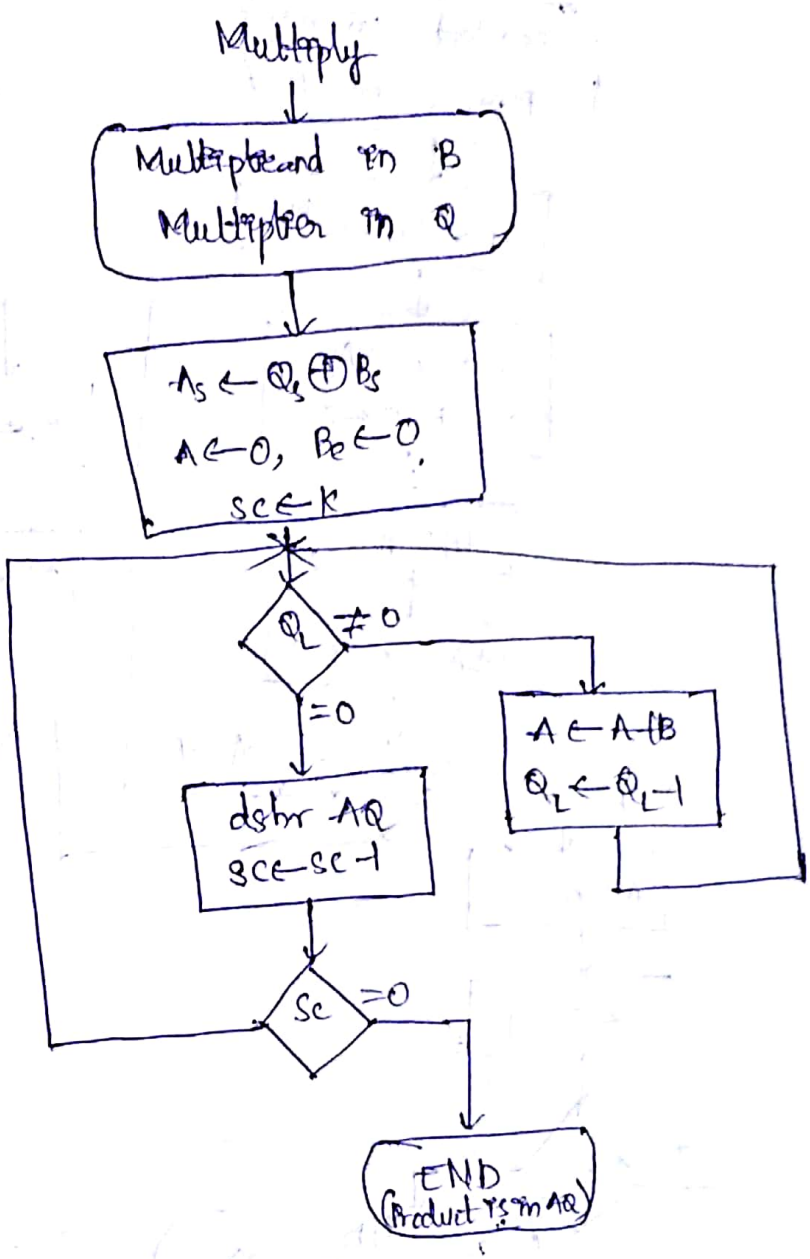
divisor in B

K indicates the no. of digits in multiplier (K=4

For division, k - no of ~~bits~~ digits in quotient

After multiplication, product is stored in AQ

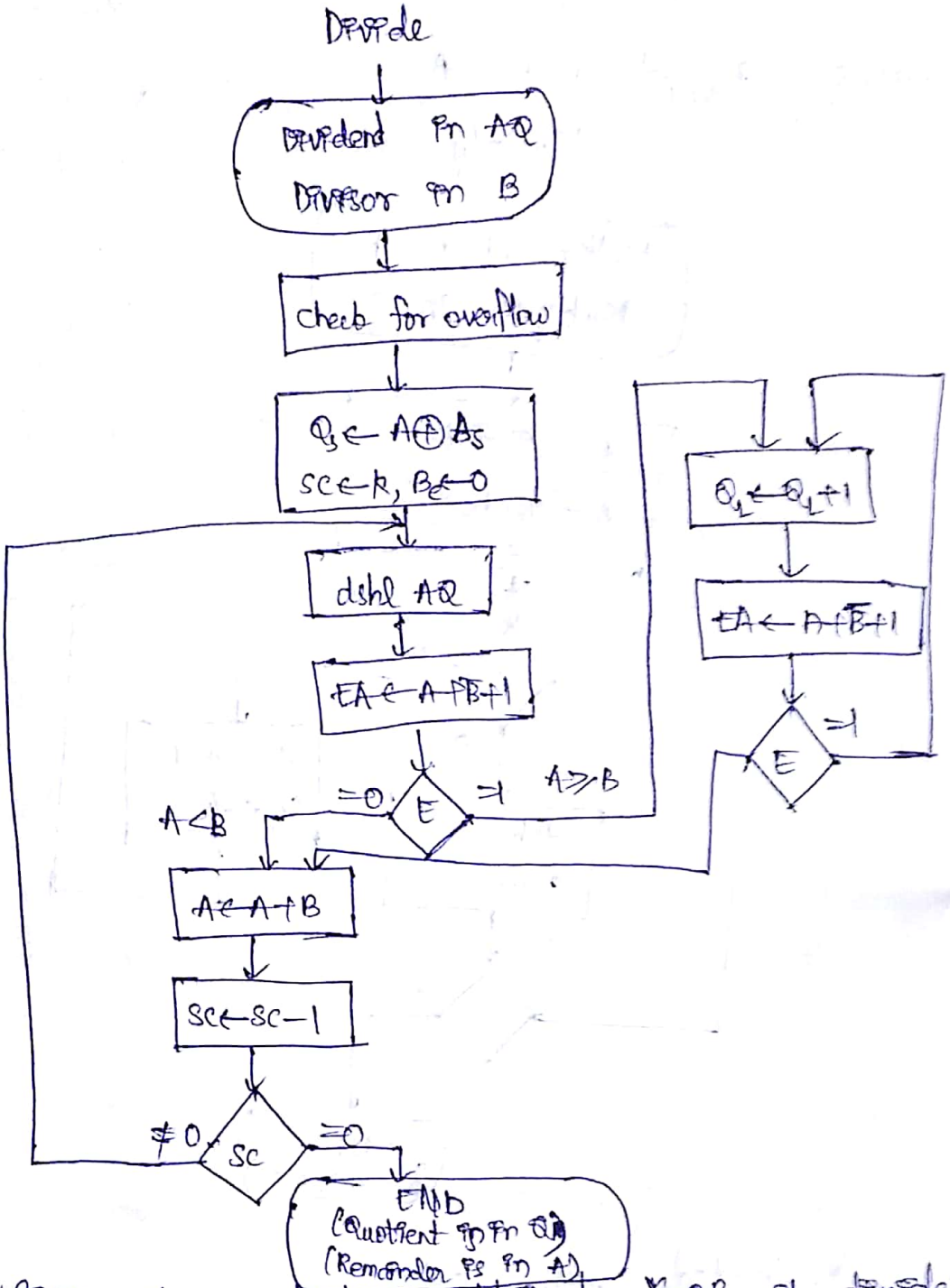
After division, quotient is stored in Q and remainder is stored in A.



SC is loaded with 'k'

Q_L indicates the ~~no of~~ LSB of Q (multiplier or dividend). shr → decimal shift right

Division (signed BCD)



Sign of quotient is obtained X-OR of dividend and divisor and sign of remainder is the sign of dividend.

k indicates no. of digits in ~~divid~~ quotient

E=0 ⇒ ~~A ← B = 0~~ A < B (∵ carry is not generated ⇒ -ve)

E=1 ⇒ A ≥ B

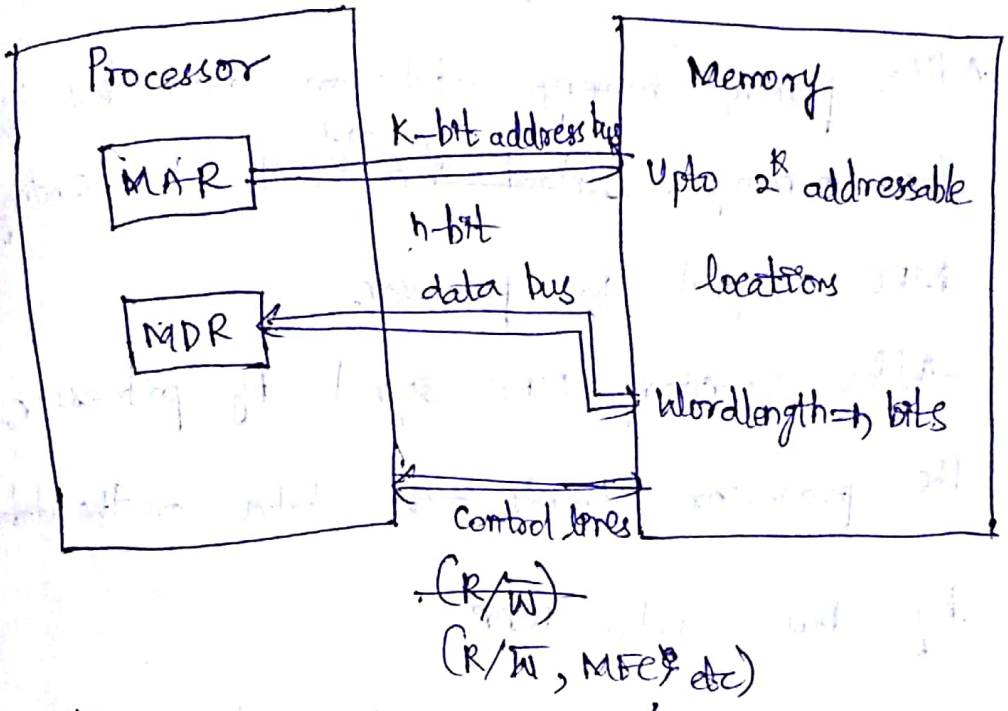
for operations on

Algorithm applied for float pt. binary no's can also be

" " " decimal no's (BCD) (+, −, *, /)

THE MEMORY SYSTEM

Some basic concepts:



~~Fig.~~ Connection of the memory to the processor

MAR (Memory Address Register)

MAR provides address of memory location to be accessed

MDR (Memory Data Register)

It contains the data that is to be read/written into memory.

Address bus is used to transfer address data

" " " data

control lines are used to ~~transfer~~ control signals.

Eg: Write/Read signal ($R/\overline{W}$ ~~$= 0$~~ $= 1 \Rightarrow$ Read

$R/\overline{W} = 1 \Rightarrow$ Write)

MFC (Memory function completed)

~~After~~ placing ~~memory~~ data on data bus by the memory, ~~control lines~~ memory will be sending MFC signal to processor.

~~After~~ receiving MFC signal by processor, the processor loads the data on the data bus into MDR

→ Size of MAR is k bits ⇒ it can provide 2^k no. of addresses from the memory

→ Size of MDR is n-bit, we can transfer n-bits at a time from memory & processor.

Size of MDR = Word length = n-bits

Word length → no. of bits that can be transferred b/w processor & memory at a time.

→ The no. of address lines depends on size of MAR.

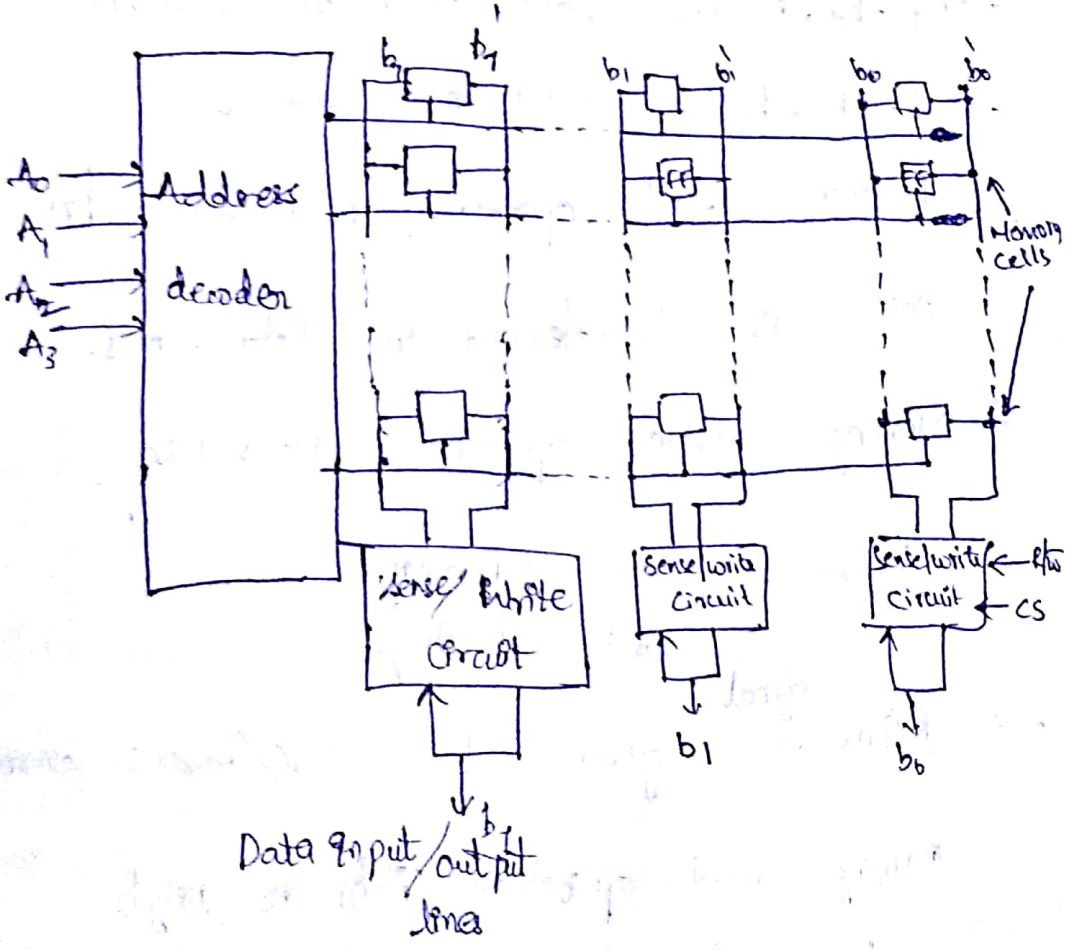
→ The no. of data lines in data bus
= no. of bits in MDR.

Semiconductor RAM Memories:

→ If any location in a memory can be accessed in a fixed amount of time, it is called as Random Access Memory (RAM).

→ If RAM is made up of semiconductor material, it is called as semiconductor RAM.

Internal organisation of memory chips:



→ For storing each bit, 1 memory cell is required

" " 128 bits, 128 " " "

→ 1 memory cell is call also called 1 flip flop

→ 128 bits are stored in 16 locations such that each location contains 8 bits

→ W_0 to W_{15} indicate 16 locations in the memory and each location contains 8 bits

→ For accessing 16 locations, we require 4 bits for address ($2^4 = 16$)

→ After accessing some location, other data is written into location (or) read from location

→ $b_7, b_7', b_1, b_1' \ldots$ are called as bit lines.

→ $b_7, b_1, b_0 \ldots$ → data lines (data bus)

→ During read operation, data from bit lines is transferred to data lines.

During write operation vice versa.

Sense is sense → Reading

Write → Writing

→ R/$\overline{W}$ signal is given to read/write circuit

During read operation, R/$\overline{W}$ is high

" write " , R/$\overline{W}$ is low.

CS indicates chip select. When the memory

contains multiple bits, chip select pins used in order to select a particular chip

Static Memories:

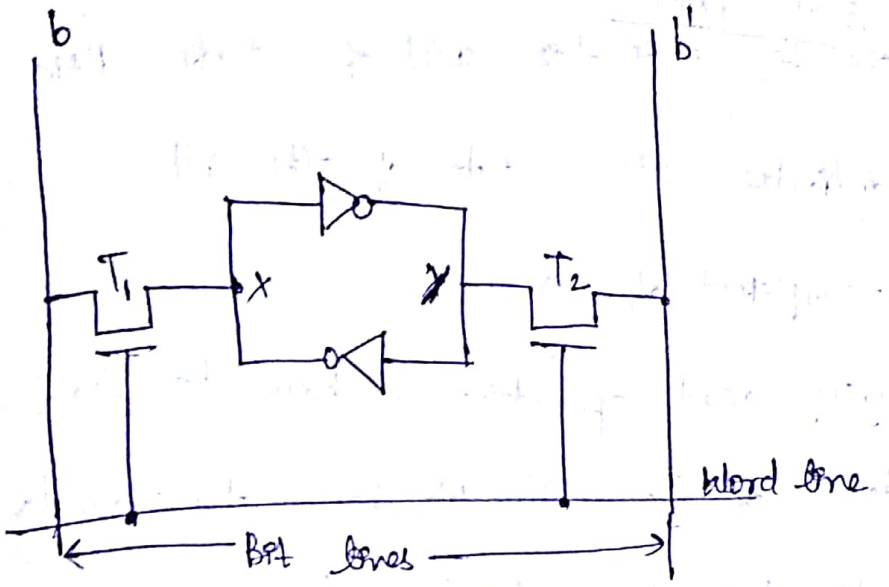


Fig. A static RAM cell.

→ In RAM, we can store information as long as power is on and it is lost when power is off.

→ It is of 2 types.

1) Static RAM
2) Dynamic RAM.

Static RAM:

(Cell is used to store one bit)

In this RAM, once information is stored, it is retained as long as power is on.

~~Dynamic~~
~~Static~~
~~Dynamic~~ PMA:

In the PMA, cells are refreshed periodically (~~Restoring information~~) to restore information.

Static PMA:

→ ~~Above fig~~ indicates cell of static PMA

X indicates the state of the cell.

Y - complement of X

During read operation, we have to read the state of the cell. We have to transfer information to data lines during read operation.

→ Word line is used to activate the word. It indicates the amount of information stored in each ~~bank~~ location ~~here~~

→ If the word contains 2 bits, 2 cells are required.

→ During read operation, word line is activated, transistors T_1 and T_2 will be on and X is transferred to b, and Y is transferred to b'. b & b' are called bit lines which are connected to sense/write ckt.

The purpose of sense/write ckt during read operation is to transfer data from bit lines to data lines and vice versa (during write)

During write operation, data is transfered to bit lines from data lines by activating word line and sense/write ckt and word line is activated and $b \rightarrow x$, and $b' \rightarrow y$.

→ 4 transistors for inverting and 2 for transfering are required. Totally, 6 transistors are required.

Advantage: It is fast

Disadvantage: It is expensive.

If CMOS transistors are used for inverting, then power consumption will be less.

⇒ In dynamic RAM, there are of 2 types

1) Synchronous dynamic RAM
2) Asynchronous "

Diff b/w synchronous and asynchronous dynamic RAM is, clock is given to

synchronous dynamic RAM. In this RAM, all operations are synchronised with clock.

~~Ash~~

Asynchronous Dynamic RAM (DRAM)

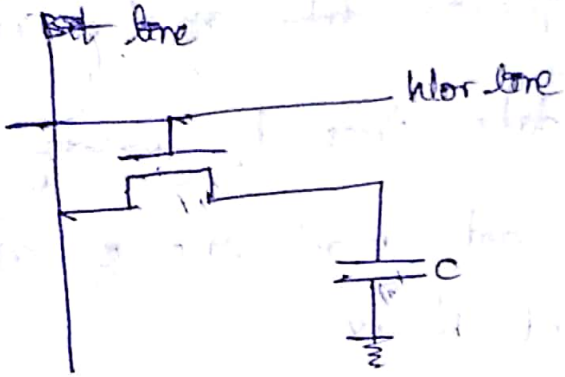


fig. A single transistor dynamic memory cell.

→ It uses only 1 transistor

∴ It is less expensive.

→ When information is to be stored in cell, transistor is switched on and appropriate voltage is applied to bit line and capacitor is charged.

→ When the transistor is off, capacitor discharges. ∴ Again capacitor should be recharged ~~which is called~~ to restore the value (to logic 1) which is called

periodical refreshing.

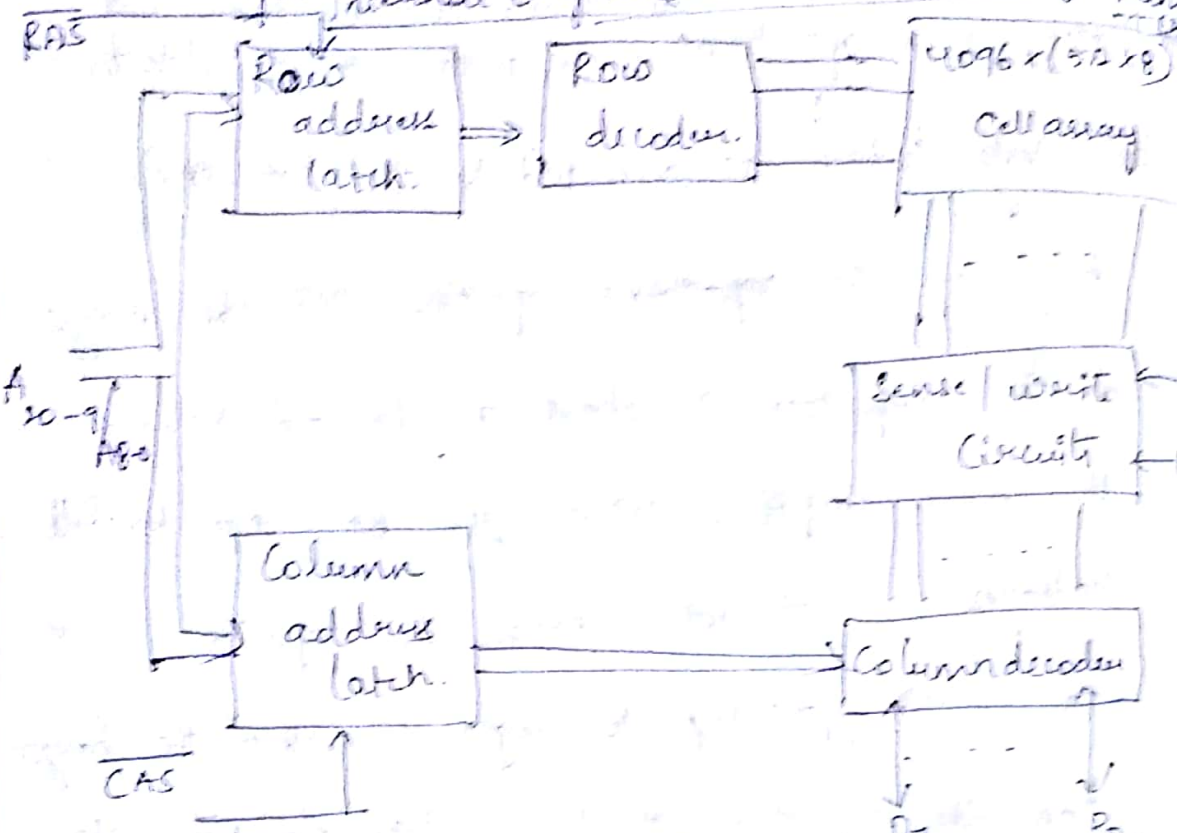
→ During read operation, sense/write circuit detects the value (which is connected to the bit line) of charge on the capacitor. If the charge on the capacitor is above a threshold value, the sense amplifier drives the bit lines to full voltage ∴ it will charge the capacitor " " corresponding to logic 1. When the charge on the capacitor is below threshold value, the bit line is pulled to ground level which indicates no charge on capacitor. ⇒ logic 0 (storing wrong information).

In order to avoid this, cell should be periodically refreshed. It should be refreshed before the charge on capacitor falls below threshold value, inorder to retain the information in the cell.

14/09/19

Asynchronous DRAMS

Fig:- Internal Organisation of a 2M×8 dynamic Memory chip



We have to design 16 Megabit dynamic Memory Array. For this we need 16 Mega cells (2M×8). We are organising 2 Mega location which can contains 8 cells in each location total we get 16 cells. and we need 2 Mega addresses for 2 Mega locations.

[1M = $2^{10} \times 2^{10}$] For 2M = $2 \times 2^{10} \times 2^{10}$

For addressing 2M locations we require

10 + 10 + 1 = 21 address bits (12 – selecting row, 9 – selecting column)

i) Total 16M cells are arranged in a 4096 rows and each row contains 4096 cells.

(4096 × 4096 = 16M)

↓

(512×8)

ii) Each row we have 512 groups with 8 cells

(512 × 8 = 4096 cells)

8 cells – 1 bit
for 512 bits × 8 cells. in each row we have 512 addresses.

w) We have 512 columns in each row for selecting a particular column we need 9 bits (A_{8-0}).

w) For particular bit we should select the row first and then column

Eg:- 2nd bit of 1 row 2nd column.

w) A_{20-12} → indicats 12 address bits.

w) first row address loaded into Row address latch by enabling $\overline{RAS}$ → (Row signal) which is in active low) and row address decoded by the row address decoder.

Eg:- If the Row address indicats address of 2nd row the 2nd o/p of decoder is activated and then particular row is selected.

w) If we are performing read operation · data in the row read by the Sense/write (in read operation data is transfered from Memory to datalines.)

in read operation → acts like sense circuit

in write operation → acts like write circuit

o/p sense/write select 4096 bits and the column decoder as 8 bits ($D_7 - D_0$) it selects 8 bits from 4096 bits (512 × 8)

↳ columns → for selecting column we need 9 bits

w) for selecting column we need 9 bits (A_{8-0}) that is sent to Column address latch depending upon

the column address one of the column address is selected which is connected to column decoder

During write operation data is transfered from datalines to Memory.

8 bits is transfered ($D_7 - D_0$) to 4096 bits i/p to sense/write depending upon the column address

From sense/write circuit it is transfered to one of the row depending on row address.

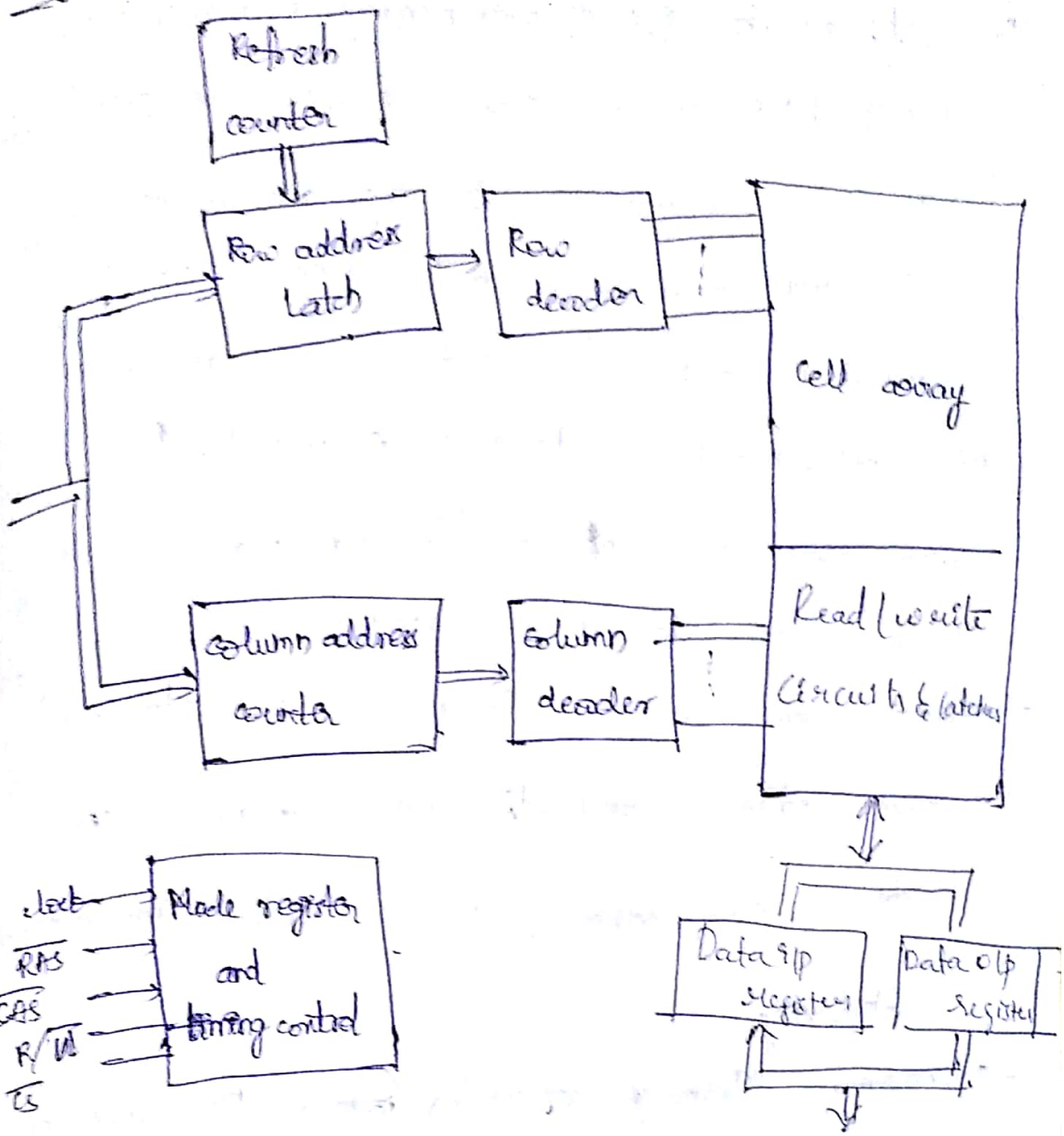
* When we perform block transfer (transfering multiple bytes) we have to transfer multiple bytes which indicates we have to transfer set of consecutive bytes (Consecutive bytes = bits which are in consecutive address) for consecutive address row address same and column address is varied.

The bytes is in particular row indicates consecutive bytes with same row address and column address is varying inorder to perform block transfer we have to transfer consecutive bytes.

For that purpose, we use latches at the o/p of sense/write circuit when we are using latches at the sense/write data is stored into latches

When we are using latches at the o/p of sense/write circuit by providing only column address, we use consecutive clks. This is called as Fast paged Mode.

Synchronous DRAM's:



All operations are synchronized with clock in synchronous DRAM.

2M×8 indicates it contains 2 Mega locations and each location " 8 cells.

Dynamic RAM is used for main memory → less expensive

Static RAM " " " cache " → more expensive (fast)

Speed of DRAM is less as DRAM is used. As the size of main memory is more, more address bits are required. To decrease the no. of " " , we are dividing into two rows and, columns and it contains more than, 1 word in each row. And we require row and column address to access a word. Here, row address is sent and then column address. Here, no. of address bits is reduced.

→ When same address lines are used to send row address and column address one after the other, it is called address multiplexing.

→ Memory controller (memory bus memory and processor) divides 21 address bits in 12-row addresses and 9-column address bits

→ For accessing two consecutive words in a row, column is incremented ∴ column address counter is used (same row) → Mode register is used to select the mode of operation of DRAM.

Block transfer indicates multiple word transfer.

During block transfer, mode register indicates the no. of successive bits to be transfered (during a block transfer)

→ Speed of transfer b/w main memory and processor depends on speed of main memory and speed of bus

Speed of main memory (DRAM) is expressed in terms of latency (time taken to transfer one word from/into memory). → Band width is another quantity used to measure the performance of main memory which is used to measure speed of block transfer.

It indicates no. of bits or bytes transferred per second.

15/9/17

Cache Memories:

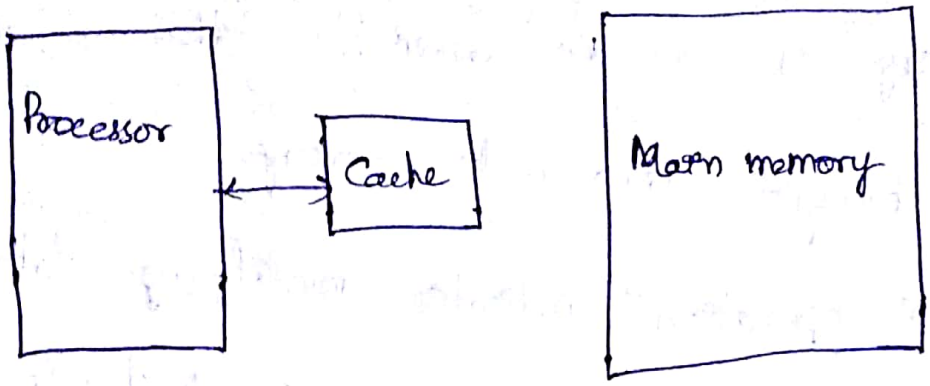


Fig Use of a cache memory.

Main memory and cache indicates RAM (Read or write)

Main memory is DRAM

Cache memory is static RAM.

Main memory contains programs and data. As the the size of main memory is large, to decrease the cost, DRAM is used. It is slow (DRAM) and performance is decreased (depends upon time of execution) To overcome this, first time to read/write a memory, it is brought into cache memory (High speed static RAM) When the word is read second time, it directly read from cache memory (already

available).

During read operation, it is read from cache memory if it is available otherwise, it is brought into cache memory.

While operation updates modifying data in same location. We have 2 protocols for write.

1) Write through protocol.

In this protocol, if the word which is to be modified in main memory is also available in cache memory, it is modified by both cache and main memory simultaneously.

If it is not available, it is modified only in main memory.

2) Write back protocol:

In this protocol, if the word to be modified is available in cache memory, it is modified only in cache memory initially.

→ If the cache is full, word is removed from cache and then modified in main memory.

→ If the word is not present in cache memory, it is first brought into cache memory and then modified.

→ Words in main memory is divided into blocks (data in each location is word)

Eg: First 16 words in 1 block ---

→ Words in cache memory is also divided into blocks.

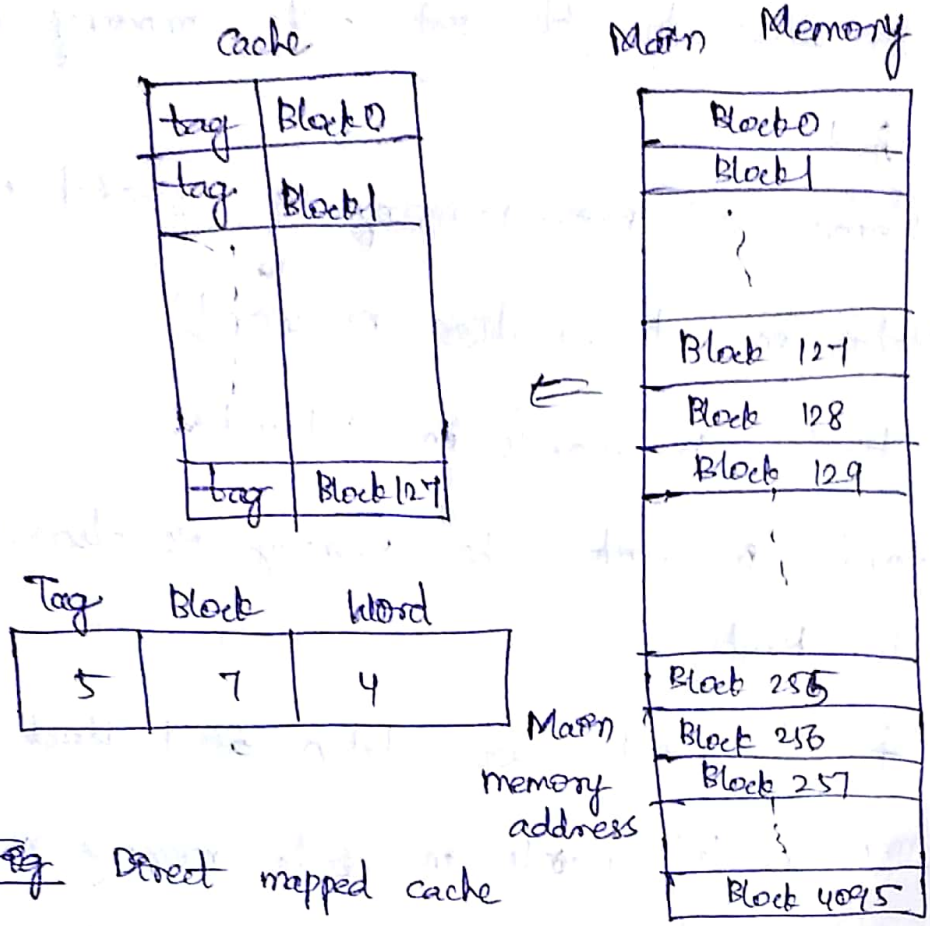
→ If 'n' words are taken as 1 block in main memory, 'n' words in cache memory is also taken as a block.

→ If some word of main memory is to be transferred into some block of cache memory from main memory, the entire block is transferred.

→ Mapping function decides where the block of main memory is to be placed in which block of cache memory.

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 "modulo 8" of cache memory

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

Ay, 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 centered 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 centered in cache memory (~~If~~ it is brought into cache if it is not present.)

Associative mapping:

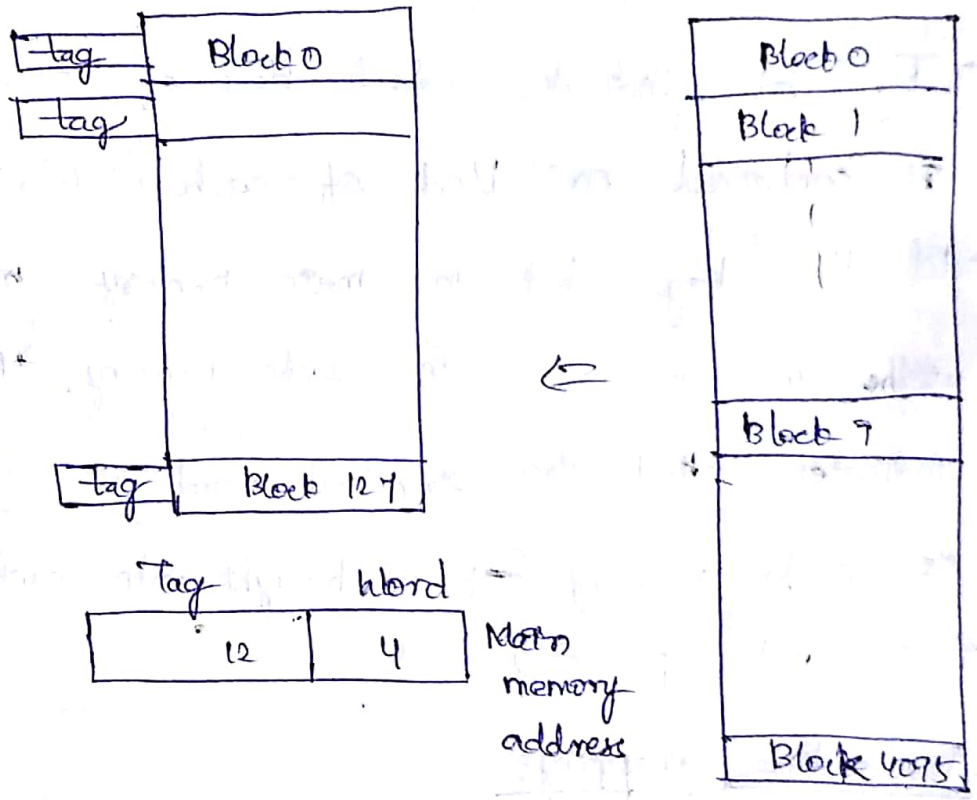
~~Any~~ Any block of main memory can be placed in " " " cache " .

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

Word is the data stored in a location.

→ Tag bits of block of cache memory indicates 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.

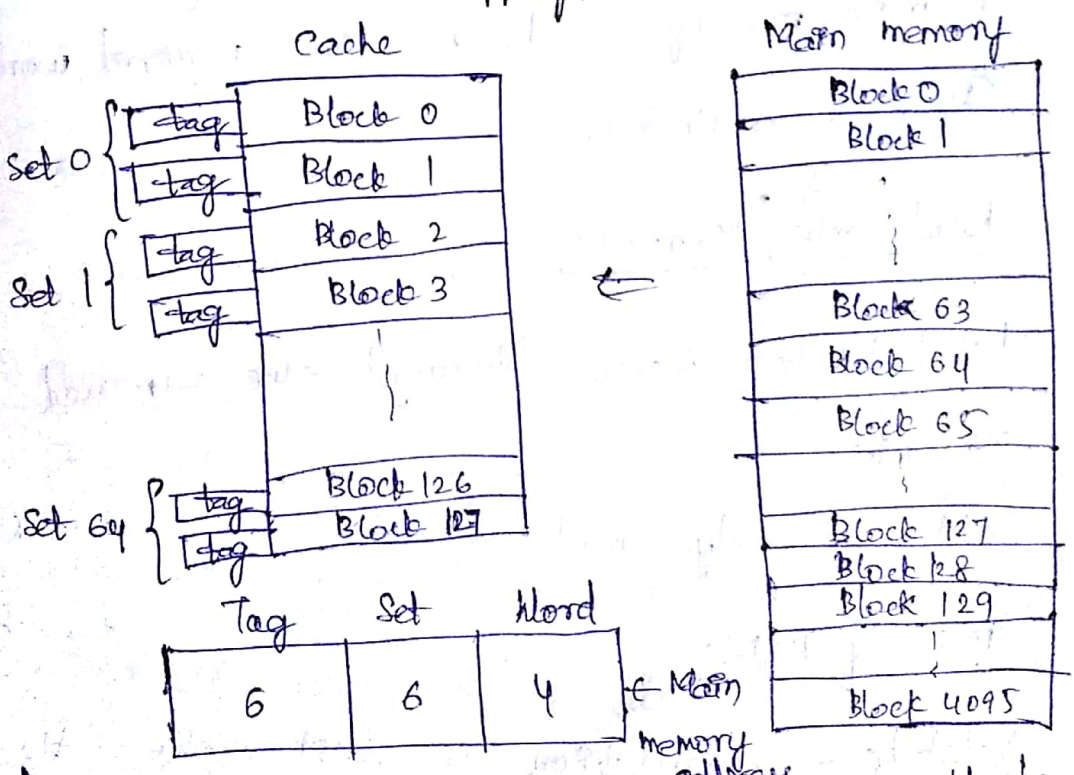
→ Algorithms which decides which of block of cache memory is to be replaced when the cache is full are called replacement algorithms.

→ 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; 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)

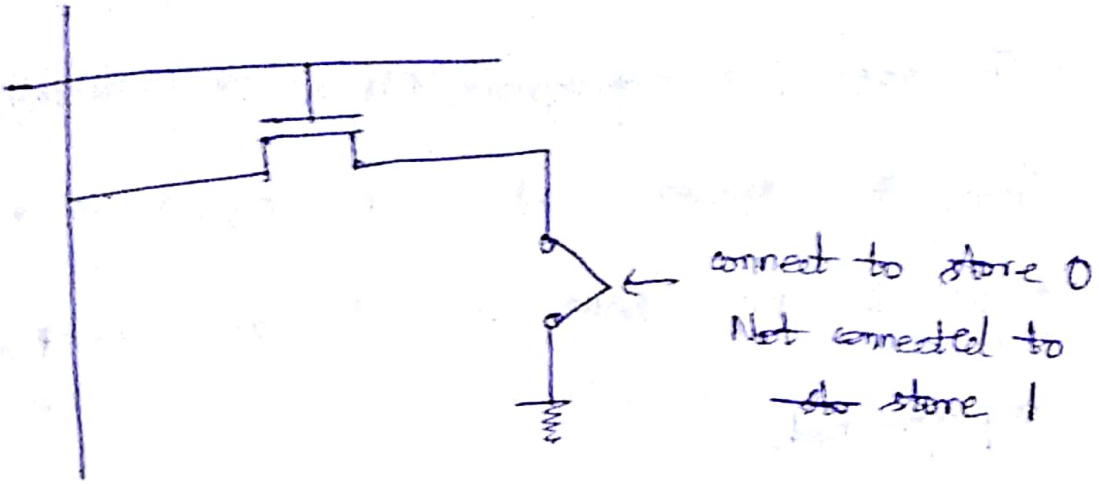
Read only Memory:

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



RAM → volatile

ROM → non-volatile

To store bit 0, terminals at p are connected

" " bit 1, " " " " not connected

1) ROM → It is programmed by the manufactured

2) PROM → " " " " user

It is similar to ROM but all cells contain fuses b/w the terminals at point 'P'. Initially '0' is stored in PROM. To store 1, fuses are burnt at point 'P' by the user by using high current pulse.

3) EROM

In this type of memory, we can erase the cells and reprogram the cell. Whereas, in ROM and PROM, it is not erased once it is programmed.

To erase and reprogram, EPROM is removed from the circuit and it is exposed to UV rays. In this ROM, all cells are erased and reprogrammed.

4) EEPROM (Electrically Erasable PROM)

→ In order to erase and reprogram, it need not be removed from the circuit.

Select cells can be erased and reprogrammed.

Disadvantage: Diff. voltages are used to read, write and erase.

5) Flash memory:

It is similar to EEPROM.

We can read single cell but can write into blocks of cells.

Advantage: Low power consumption, uses only single power supply voltage, high density, (density = no. of cells that can be accomodated in a given area) low cost per bit (cost of each cell is less)

Applications: battery driven applications (due to low power consumption)

Virtual Memories:

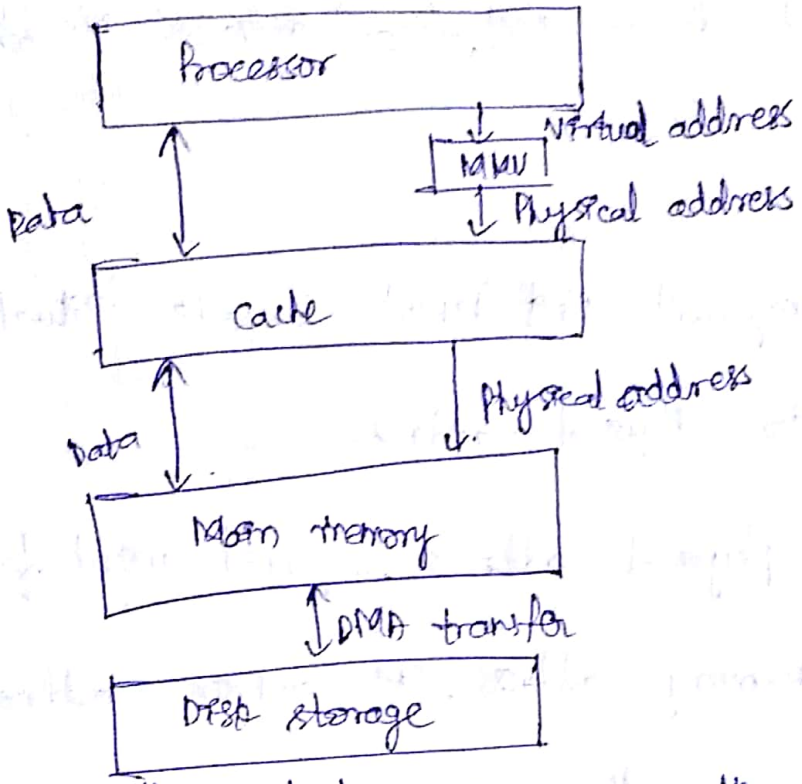


Fig Virtual memory organisation

→ 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 which 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

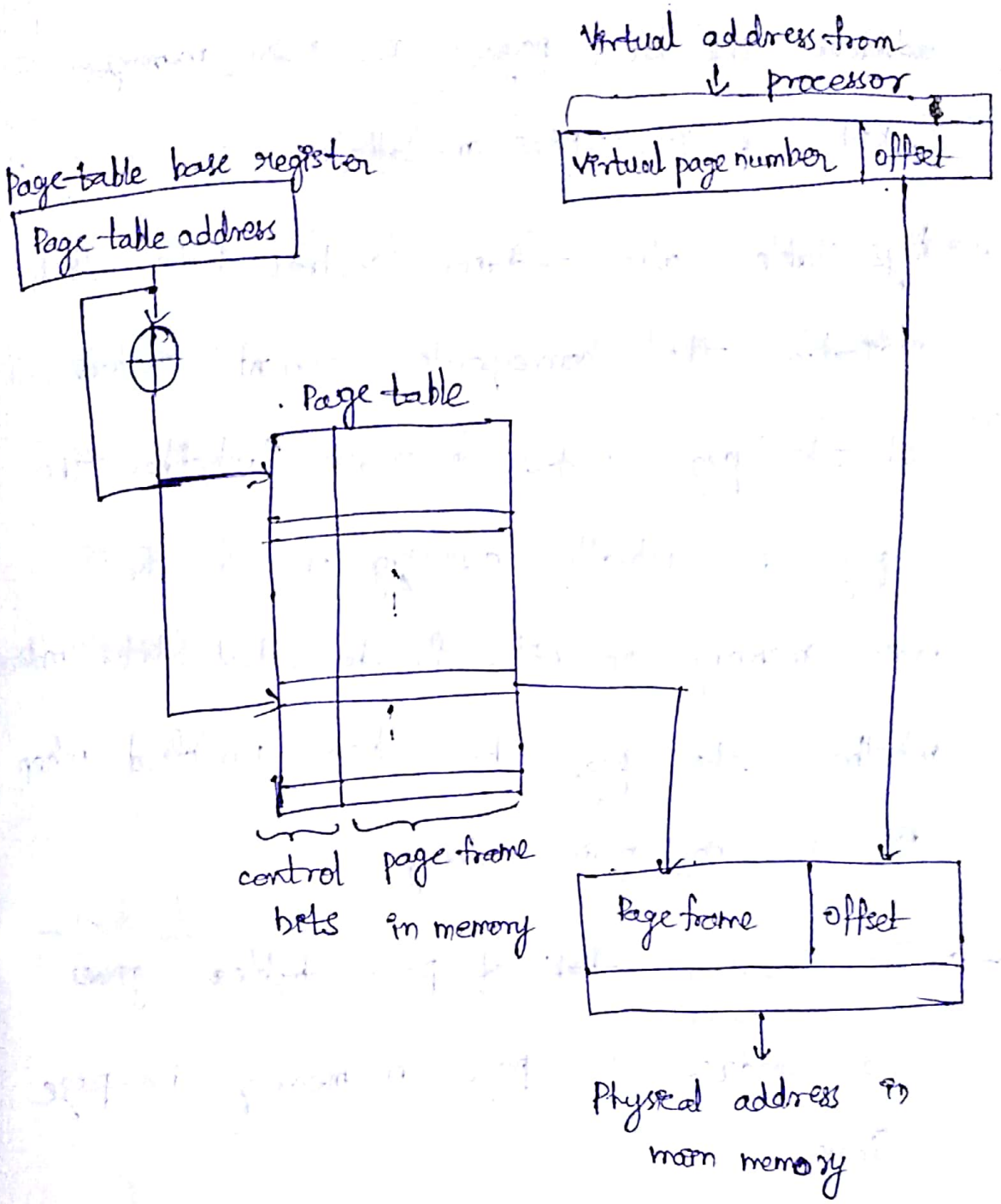
Eg: If virtual address gives page 0

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 indica 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 1004 is accessed.

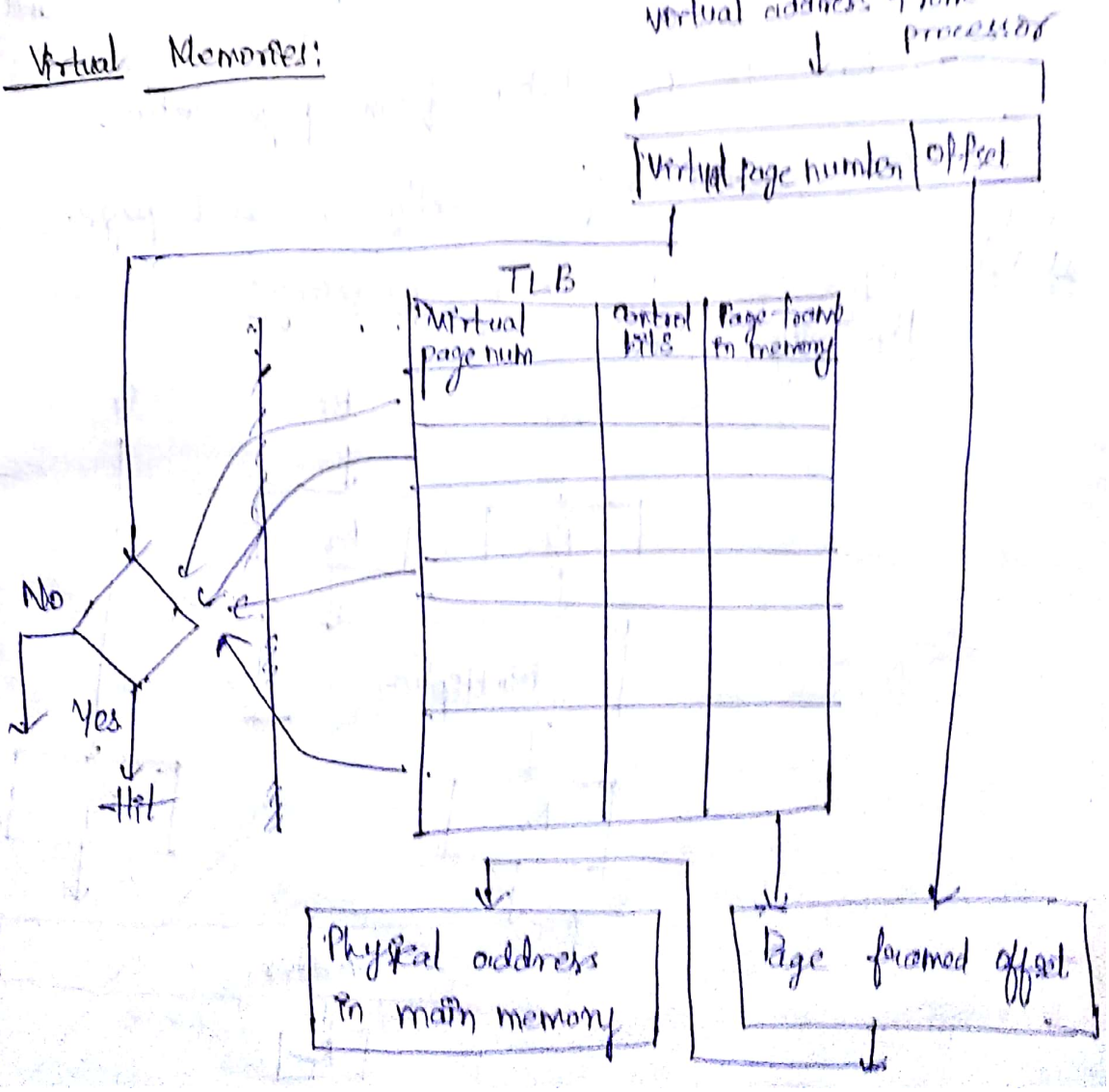
→ Page table is present in main memory.

→ 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.

→ Inorder to reduce the time to convert from virtual to physical address, a portion of the page table is placed in MMU in the cache memory. The portion the page table which is contained in MMU 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.

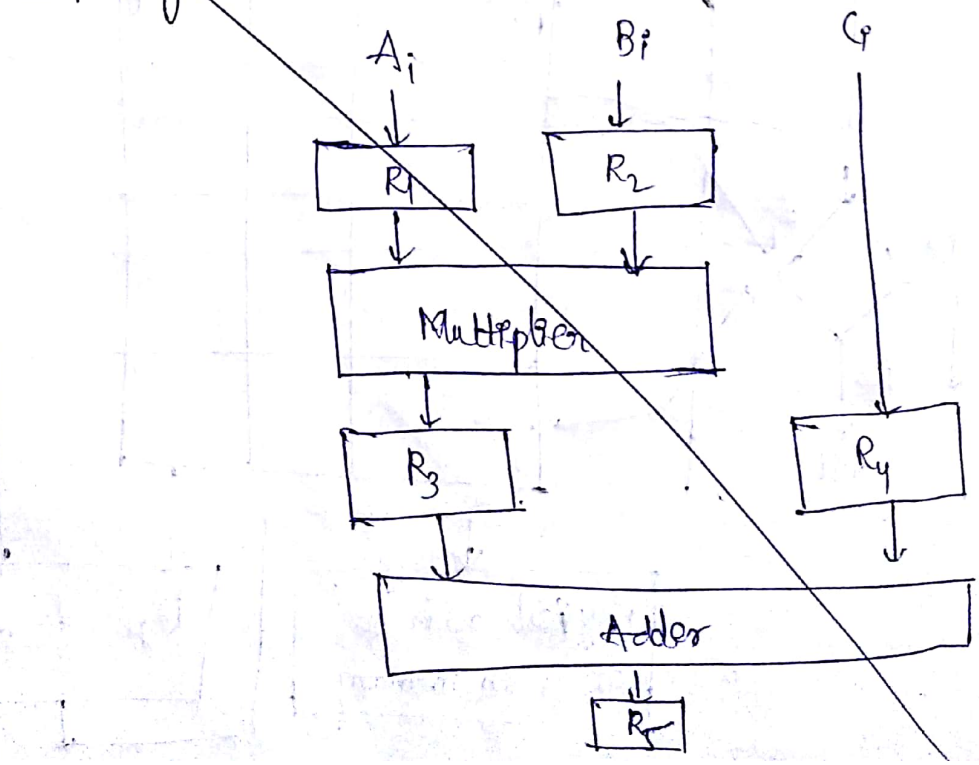
If it matches with virtual " " ", the required info. is contained in TLB.

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:

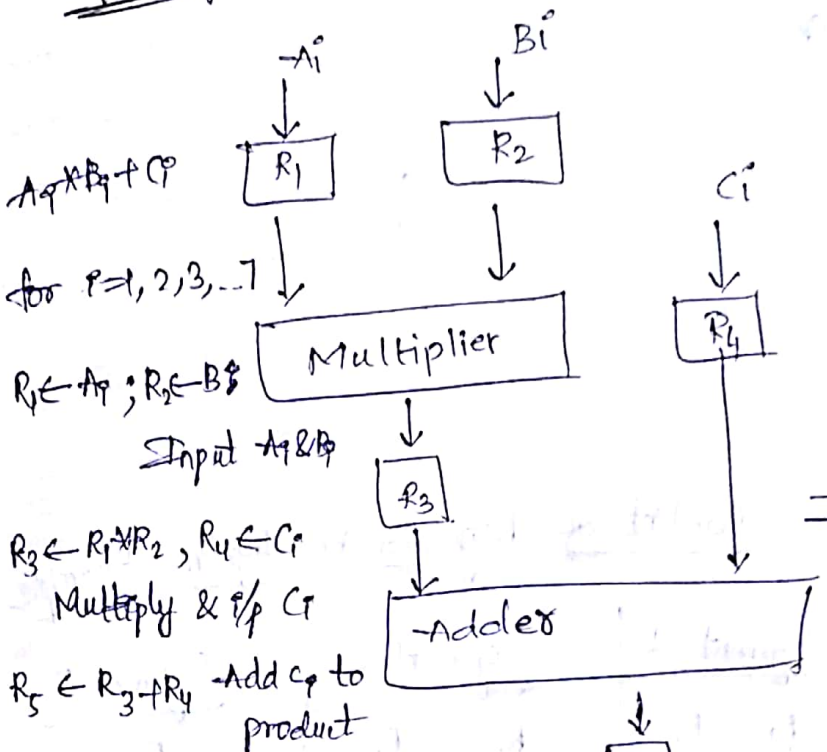
Pipelining:



UNIT-V

Pipelining and Vector Processing.

Pipelining



$A_i * B_i + C_i$ for $i = 1, 2, 3, \ldots 7$

$R_1 \leftarrow A_i$; $R_2 \leftarrow B_i$ Input A_i & B_i

$R_3 \leftarrow R_1 * R_2$, $R_4 \leftarrow C_i$ Multiply & i/p C_i

$R_5 \leftarrow R_3 + R_4$ Add C_i to product

fig. Example of pipeline processing

It performs operation on seven different data.

Difference b/w pipelining & Non-pipelining ckt is non-pipelining ckt will process only one data at a time.

→ After completion of processing of 1st data, processing of 2nd data is performed. In non-pipelining.

→ In pipelining, we can process more than 1 data simultaneously.

→ To perform $A_i * B_i + C_i$, this operation is divided into suboperations or segments

→ Here, it is divided into 3 segments.

→ During the first clock cycle, first suboperation is performed.

$A_1 * B_1 + C_1$

Contents of Register in Pipeline Example

Clock pulse number	Segment 1		Segment 2		Segment 3
	R_1	R_2	R_3	R_4	R_5
1	A_1	B_1	—		—
2	A_2	B_2	$A_1 * B_1$	C_1	—
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4.	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
6.	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
7.	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
8.	—		$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
9.	—			—	$A_7 * B_7 + C_7$

→ In non-pipelining, (after performing $A_1 * B_1 + C_1$, processing of data is performed one after the other (A_1 and B_1 are loaded)

→ Whereas in pipelining, During 2nd clk pulse, 2nd suboperation on 1st data is performed simultaneously 1st suboperation on 2nd data is performed.

General considerations:

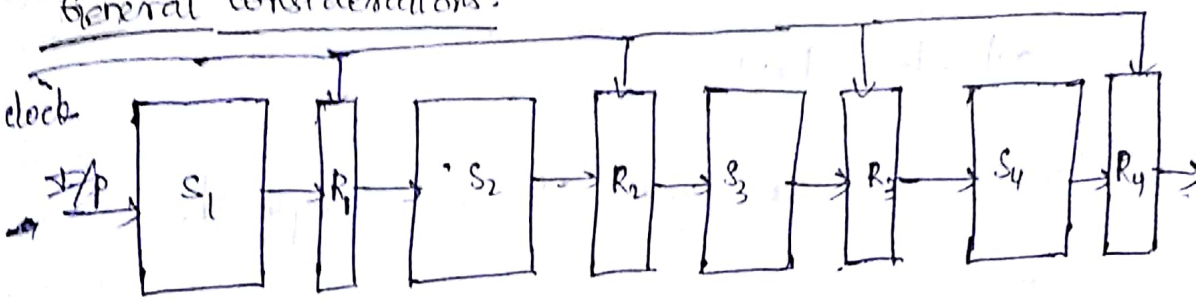


Fig. Four segment pipeline

→ Four segment pipeline indicates 4 suboperations.

→ In the previous eg, n = 7 (7 tasks)

n = no. of tasks.

Here, 6 tasks are performed ⇒ n = 6

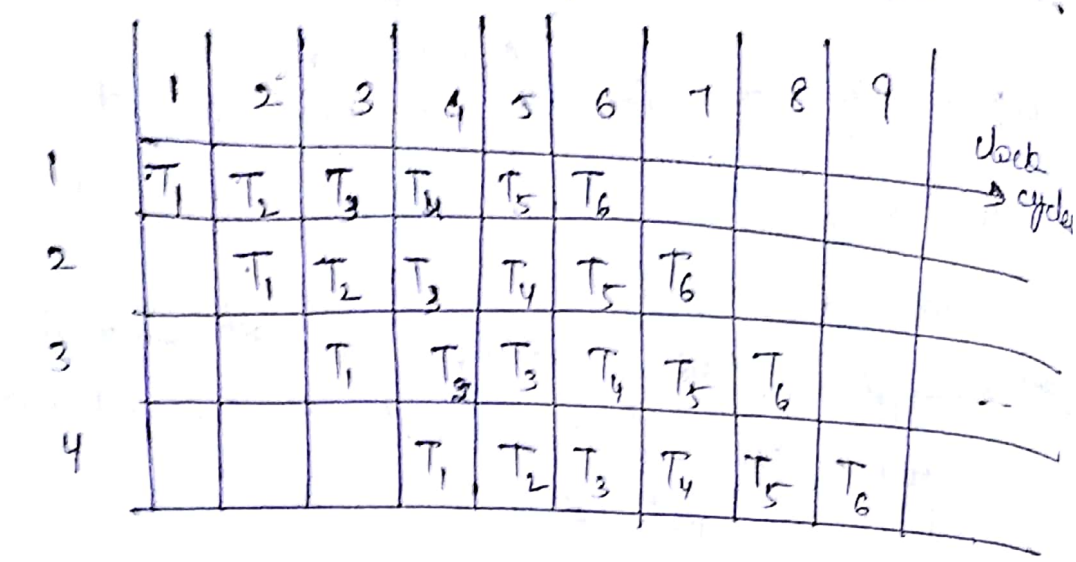
Assume that

→ These tasks indicates the same operation ($A*B+C$) but on diff data.

→ 6 tasks are denoted as $T_1, T_2 \ldots T_6$

→ All these 6 tasks should go through the 4 segments

Space-time diagram for pipeline:



→ Total 9 clock cycles are required to complete all the tasks.

→ After 4 clk cycles, T_1 is completed,

5 " " T_2 " "

9 , T_6 " " "

→ In 4+5 clock cycles, 6 tasks are completed. Here 4 indicates no. of segments

→ 'k+n-1' clock cycles are required to perform 'n' tasks, where k indicates no. of segments.

→ In nonpipelining, t_n of 't_n' indicates the time taken to complete one task,

the time taken to perform 'n' tasks is

nt_n.

'S' indicates the speed up ratio which gives the diff in speed of pipelining and nonpipelining.

$$S = \frac{nt_n}{(k+n-1)t_p}$$

Here t_p indicates clock period.

→ If $S=4$, the time taken by nonpipelining is 4 times the " " by pipelining.

→ If n is large, $n > k-1$

$\therefore k-1$ can be neglected.

$$S \approx \frac{nt_n}{nt_p}$$

$$S = \frac{t_n}{t_p}$$

$t_n = t$

The time taken by pipelining ckt to complete first task) single $t_n = kt_p$

k = no. of segments in pipelining

If we assume the time taken to

complete one task in nonpipelining = time taken to complete one task in pipelining

$t_n = kt_p$

t_n = kt_p

nonpipelining pipelining (single task)

$$S = \frac{kt_p}{t_p} = k$$

→ the maximum theoretical speed that can be achieved using pipelining ckt.

practically $S \leq k$ $(S \leq k)$

where k indicates no. of segments.

→ If k=4, speed of pipelining is 4 times greater than " " nonpipelining

→ $S = k$ when n is large and $t_n = kt_p$

→ n' k nonpipelining ckts are required to achieve the speed of pipelining ckt

$S = 4$ indicates 4-segment pipeline

→ for 'n' tasks, speed of pipelining = k × speed in

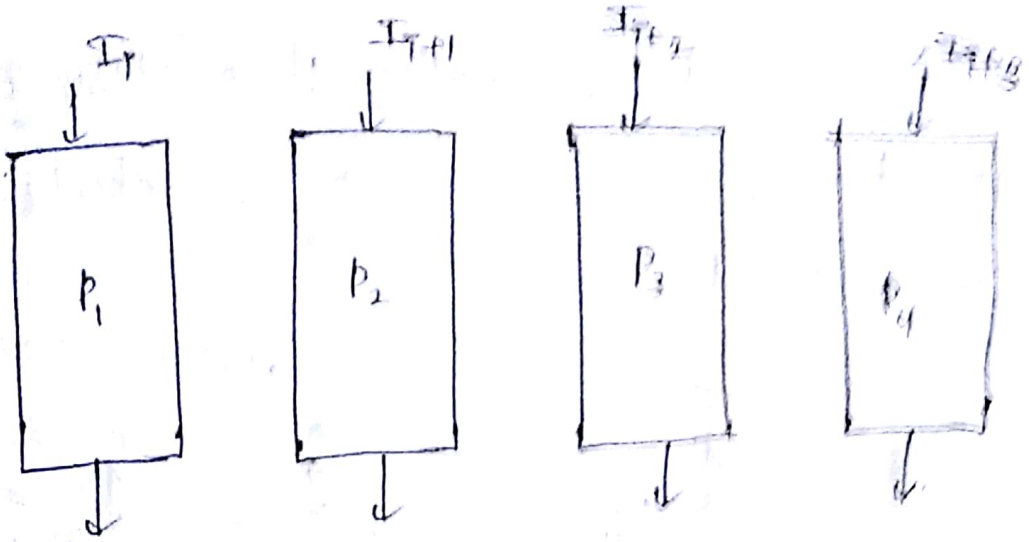


fig Multiple functional units in ALU (P_1, P_2, P_3, P_4)

Here 4 nonpipelining ckts are required to achieve the speed through pipelining

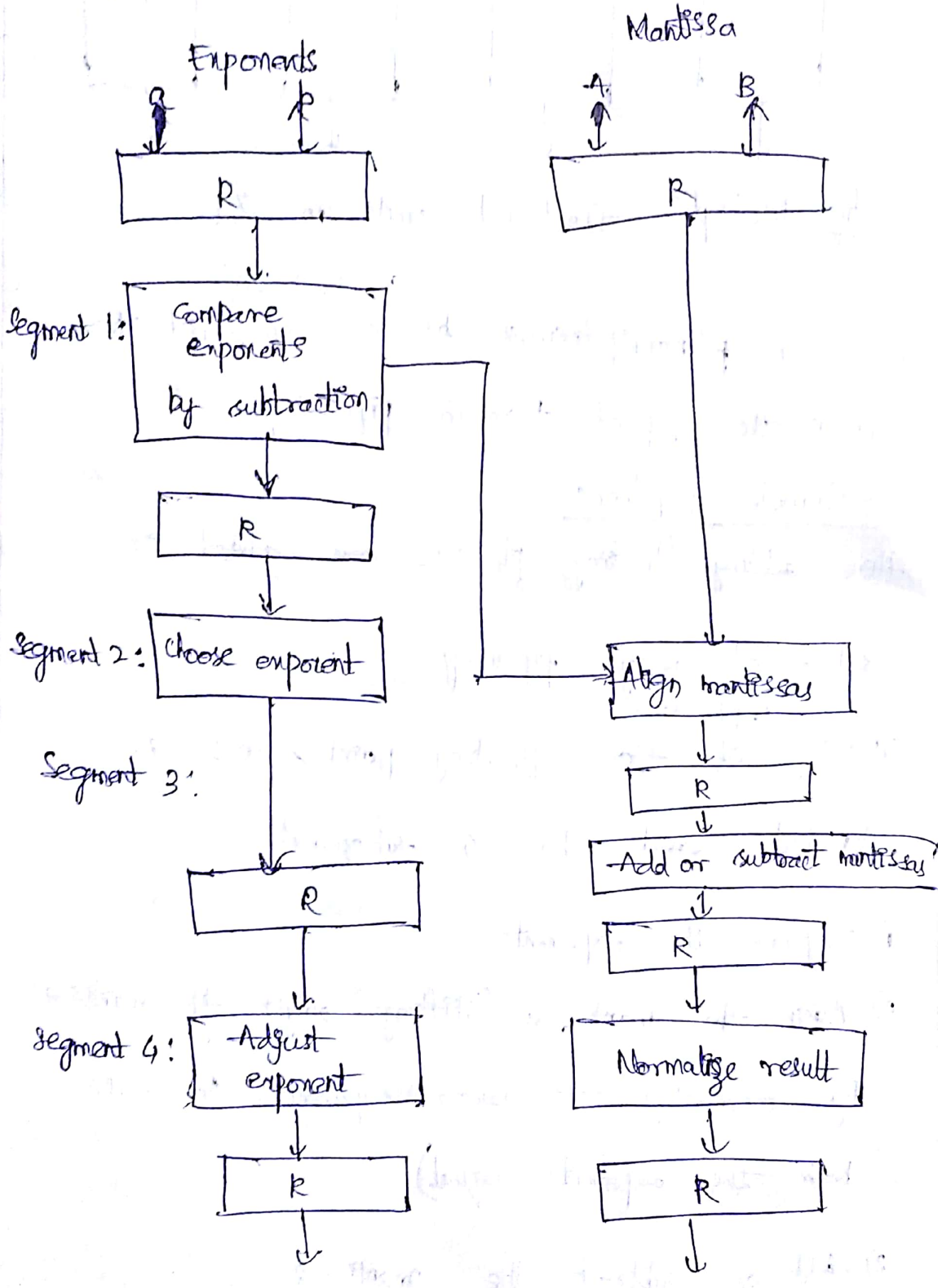
- Arithmetic pipeline:

Here, adding 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

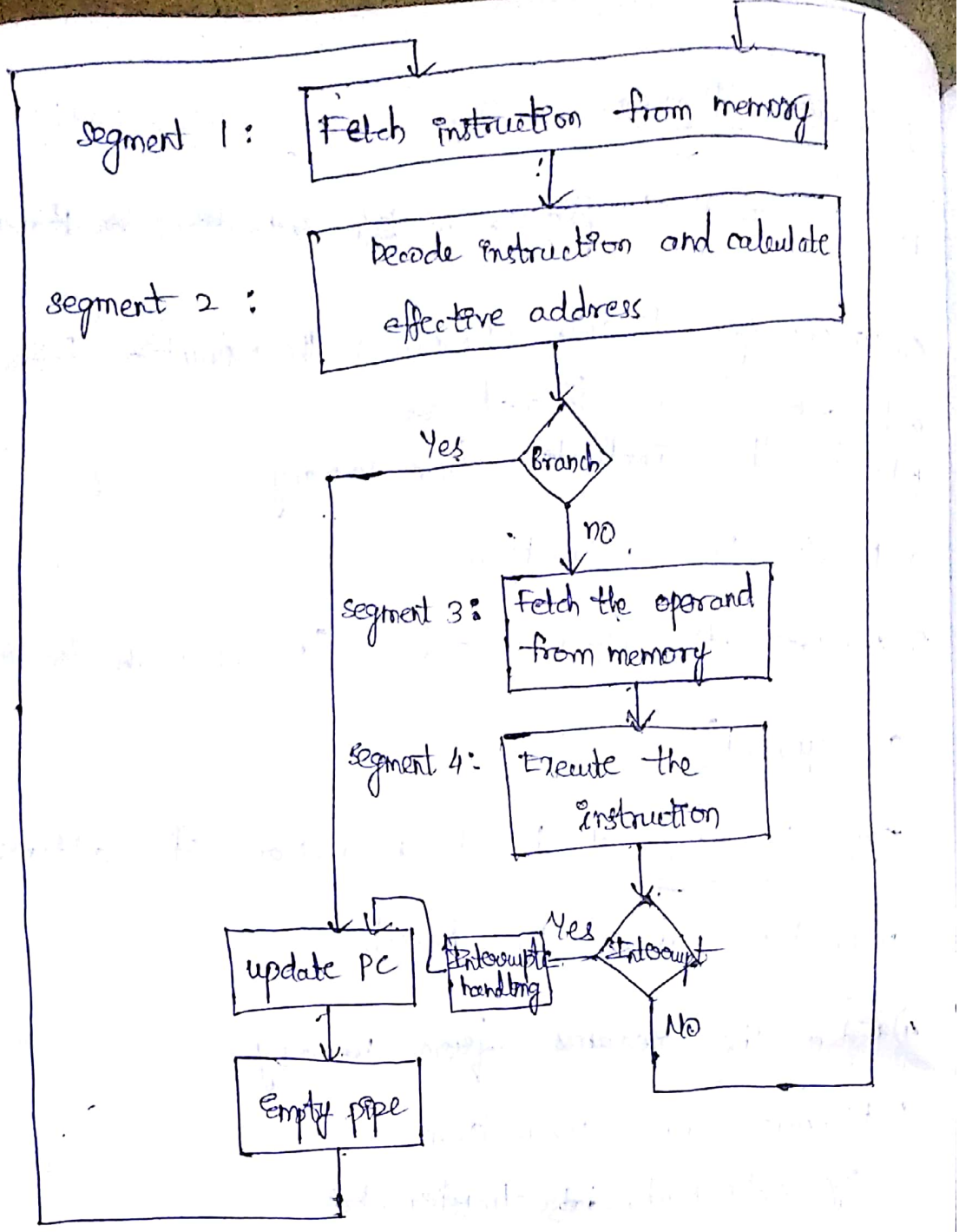


Fig Four-segment CPU pipeline

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 starting address of BR.

→ 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.

6/11/17

Pipeline conflicts:

Problems in pipeline are called pipeline conflicts which arise

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 dot is used called as hardware interlocks.

→ Another way is to delay the instruct^{h}, is, by using compiler which inserts the No operation instruction is 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 reduce minimize the degradation of the performance of pipeline, during branching instruction, we will be prefetching the instruct^{h}'s

from branching address before the completion of 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 instruction. This which is done in RISC computer. The technique is called as delayed branch

Examples for delayed load and delayed branch using RISC pipeline:

Instruction pipeline corresponding to RISC computer is called RISC pipeline. It is divided 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 instructions. 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 and result of ALU is transfered to destination register in 3rd segment. on AL

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

Ex: M[x] → R

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 inserting NO instruction b/w 2nd and 3rd instruction

∴ In 5th clock cycle, R_1+R_2 is performed.

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 "

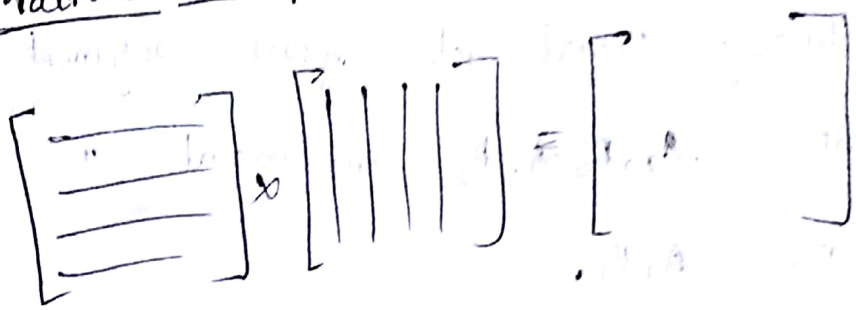
→ Same operation can also be performed using scalar instruction with more than one instruction using loops.

→ 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: Mat~~

Matrix Multiplication:



~~To perfo~~

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

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

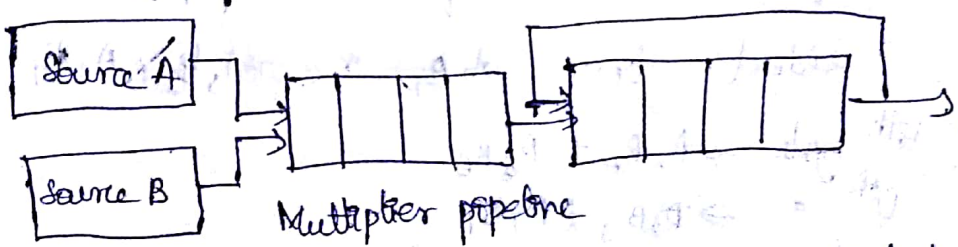
~~Ar~~ $A_1, A_2, \ldots A_k$ are row vectors

$B_1, B_2, \ldots B_k$ 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,

→ By 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

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

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

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

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

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

Super scalar processor:

It 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:

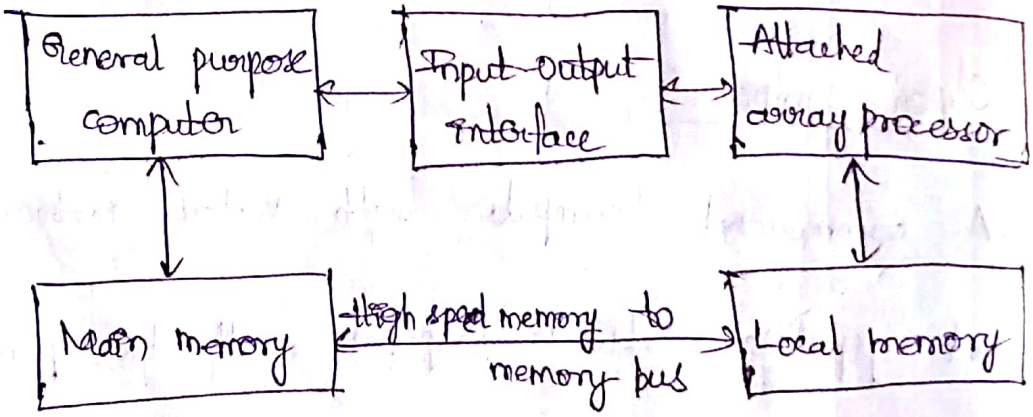


Fig Attached array processor with host computer.

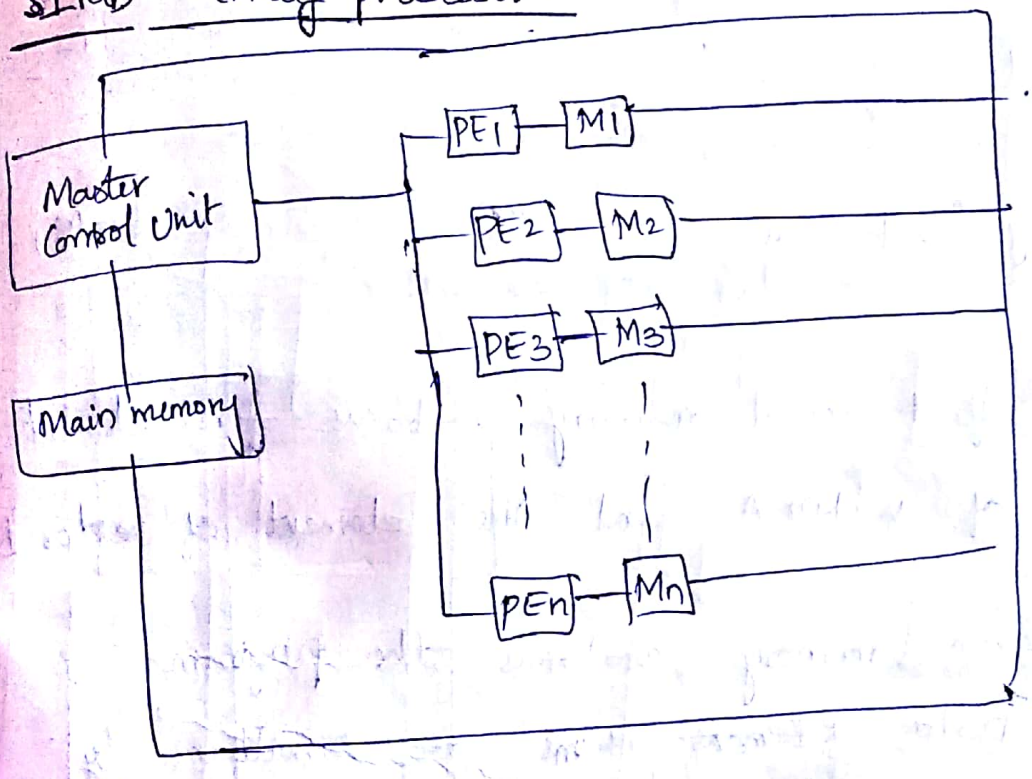
→ It is attached to the general purpose computer. It acts as I/O device to the general purpose computer.

→ 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 instn... 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
⋮
Ny, 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:

→ If 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

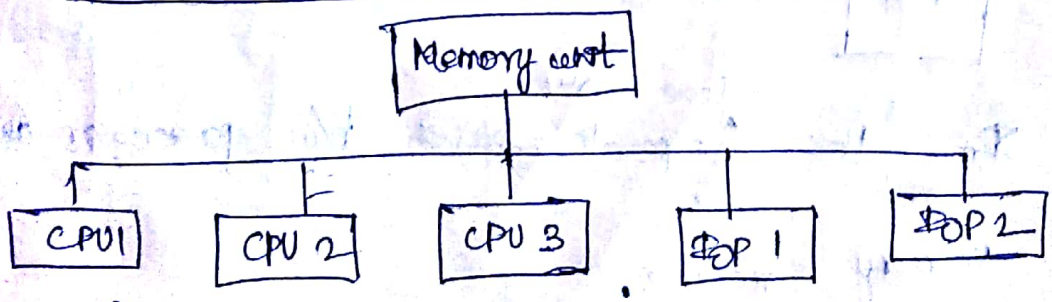


fig: Time-shared common bus organization

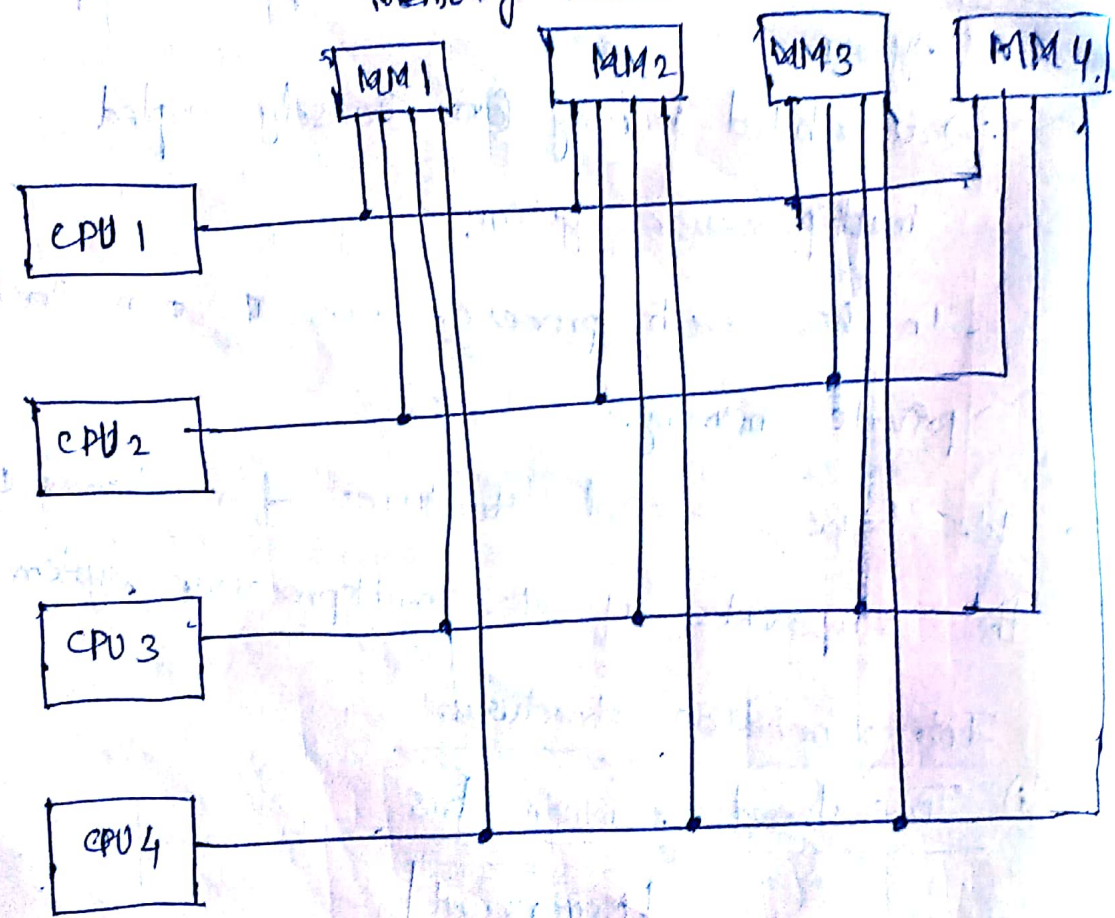
→ It indicates single bus (common) b/w the processors and memory

→ 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:

Memory modules.

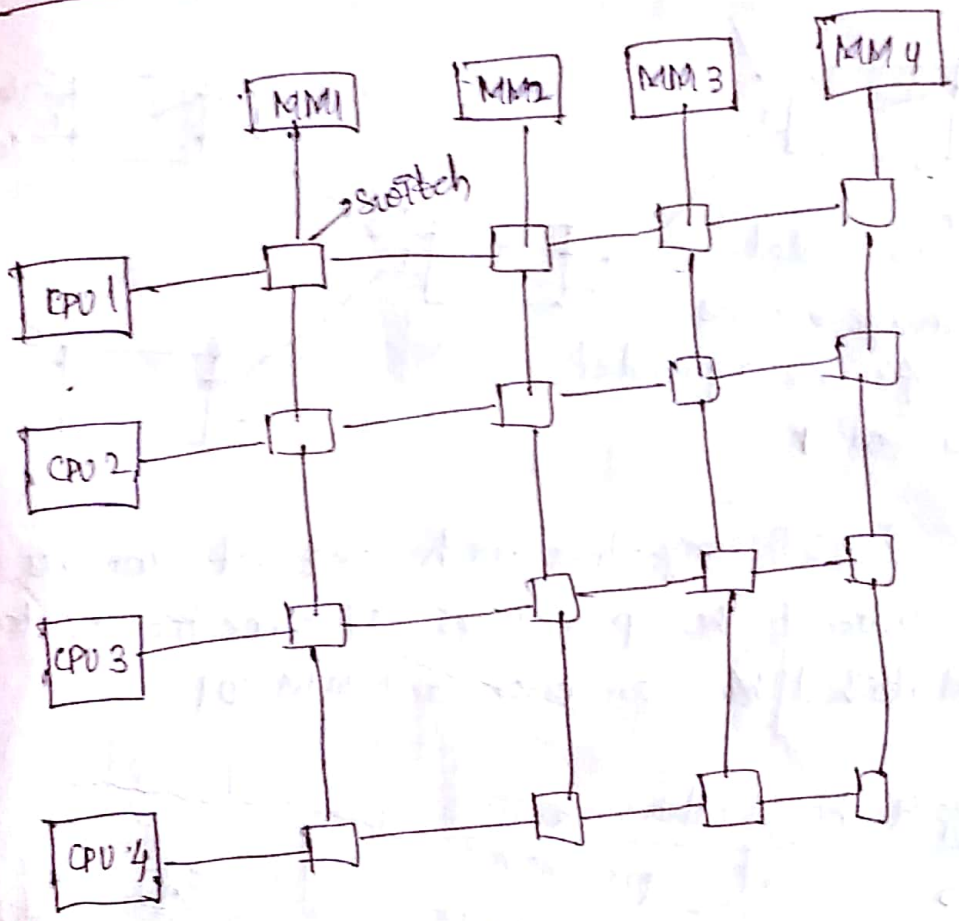


In this, there are separate buses b/w processor and memory module.

→ 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:

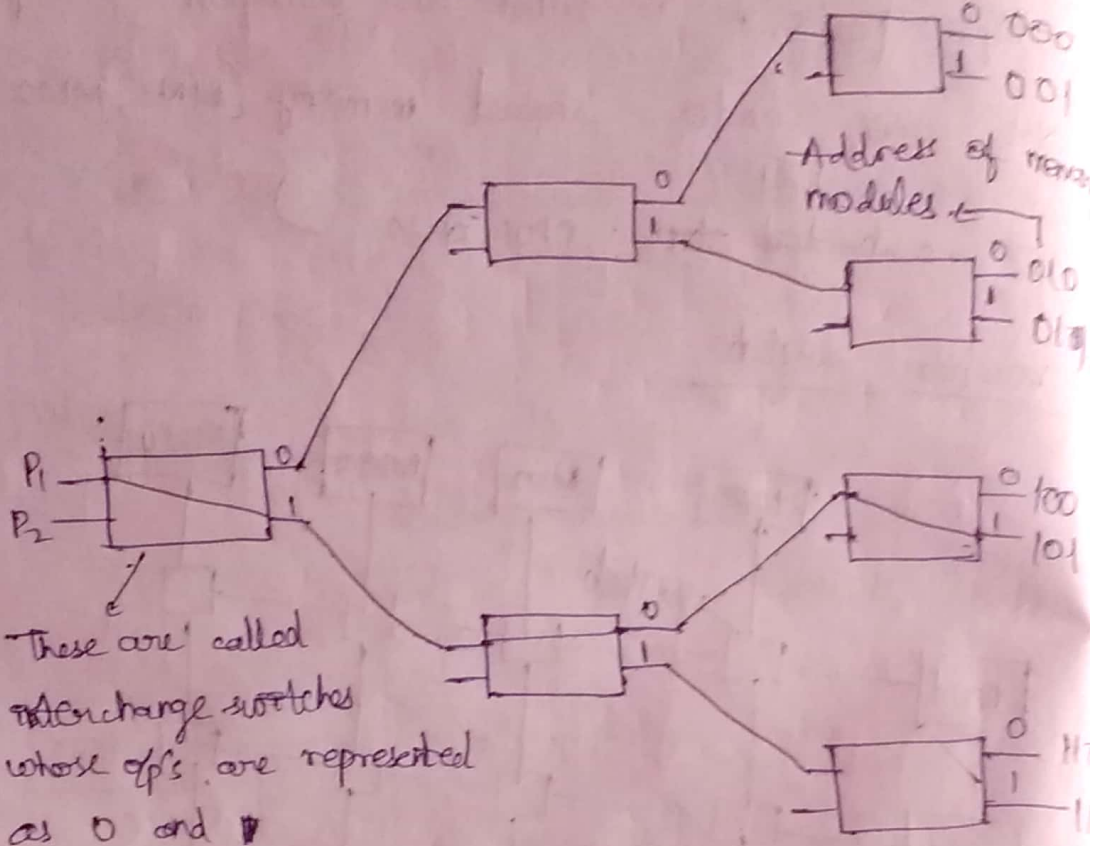
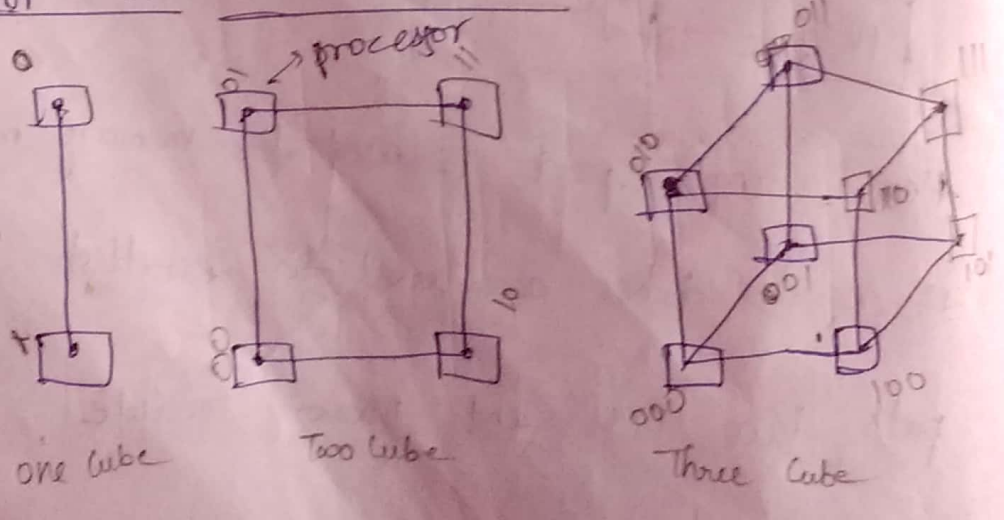


Fig: Memory tree with 2x2 ~~it~~ switches. P_1, P_2 indicates the processors. If connection is to be established b/w processor and MM 101

5) Hypercube interconnection



It 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

UNIT-IV

INPUT-OUTPUT ORGANIZATION:

Input-output Interface:

→ In order to compensate diff's b/w CPU and I/O devices, I/O interfaces are used

Eg: CPU - high speed - electronic devices

I/O - slow - electromechanical or electromagnetic devices.

∴ Signals values should be converted.

→ Another diff is data codes used by CPU and memory is diff from data codes used by I/O devices. (i.e. ASCII or E

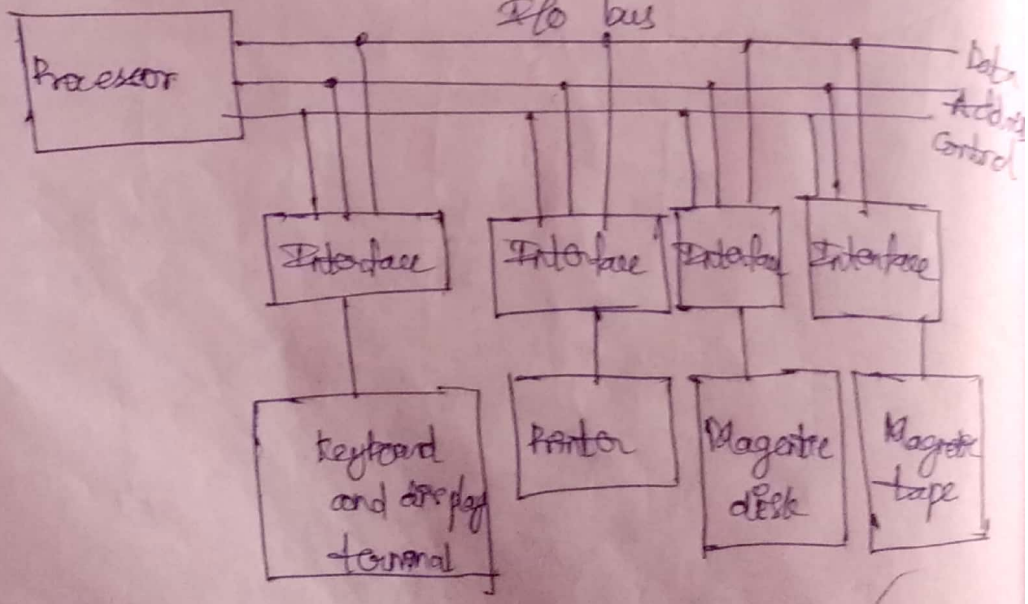


Fig Connection of I/O bus to input-output device

→ Magnetic disks and Magnetic tape are used for storage devices

→ I/O devices are also called peripheral or external devices.

→ Bus contains data, address and control lines.

→ The address sent by the processor on the address lines is used to select the I/O device.

→ Info transferred on the control lines is function code which is referred as I/O command. They are of 4 types.

I/O commands:

1) Control command: It will inform the peripheral what to do.

2) Status command: It is used for diff. purposes eg: to know the status of peripheral i.e. whether it is ready to perform data transfer or not.
ii) whether there are any errors during the data transfer.

3) Data output command: It is used to transfer to data on the data lines into the register in the interface

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

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

Ethere 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 " " processor and from processor to memory.

Ily 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 through CPU.

It is one way to transfer

→ Second way is directly transferring 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 mem. blw memory and I/O.

12/10/17

Priority Interrupt:

To perform the data

1) Programmed
2) ~~Programmed~~
3) ~~Interrupted~~ 3) ~~Interrupt initiated~~
4) 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.

because, CPU will get interrupt from I/O device when I/O is ready to transfer.

The time in which CPU is waiting

→ For I/O transfer we can perform another operations.

→ For processor (CPU), we may have 2 (or) more I/O devices like keyboard, mouse etc. all these are interrupted from I/O.

→ If all interrupts are arriving in the processor, processor responses to only one interrupt based on priority.

→ High speed I/O device - high priority

→ low speed " " - lowest "

→ Processor does sourcing (or) responses to high priority when 2 (or) more devices are given can be done using 1) Hardware
2) Software.

→ The task of sourcing higher priority device by software is called polling

Software → by program (executing)

→ In program, first we will be checking the 1st highest priority device is being interrupted or not

→ If interrupted, it will be sourced first

→ If no first higher priority device, then 2nd

higher priority is selected and checked till continued.

→ The task of sourcing can also be done by 'hardware' i.e. 'Daisy chaining priority'

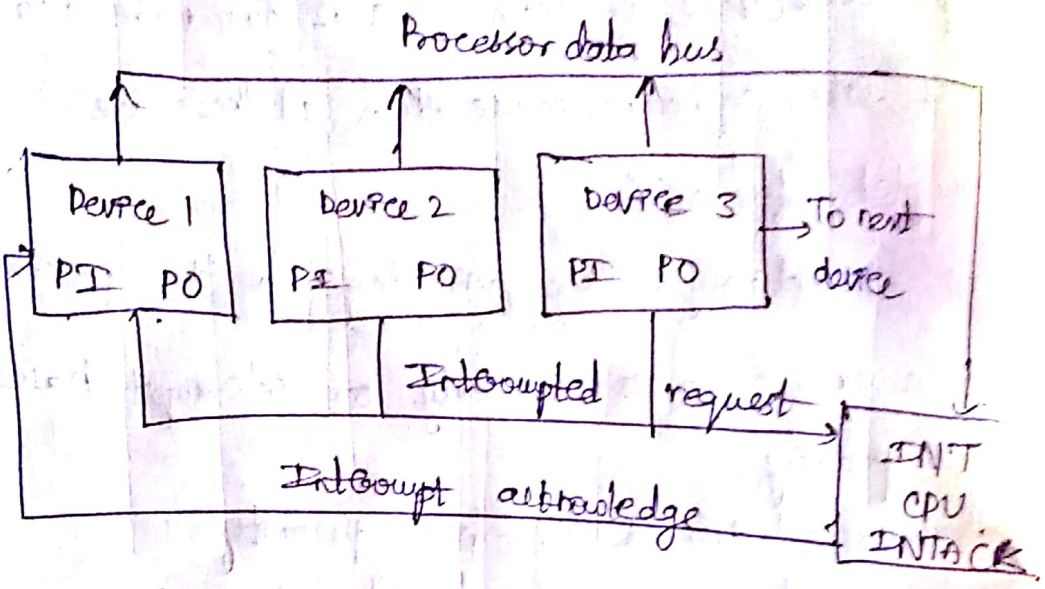


fig Daisy chaining priority Interrupt

→ In these, we are connecting devices serially & which pt device is highest priority device & last device is lowest priority device.

INT → Interrupt Active low signal.

→ One (or more no. of interrupt may be received to CPU.

→ In response of receiving interrupt, CPU gen... INTACK signal which is Active high sign...

INTACK - Interrupt acknowledge signal

→ This interrupt acknowledge signal is given to device with 1st highest priority device that if interrupt is present by device 1, PO will be 0
(o/p of 1)
PO will be 'zero' means, the acknowledgement signal is not passed to 2nd device
→ If in case of 1st device is having no interrupt then PO(o/p of 1st device) will be '1'.
-that means, interrupt acknowledge signal is passed to 2nd device and so no interrupt is there, the acknowledgement signal reaches the correct device.
→ The device which is having PI = 1 & PO=0 is the device to be sequenced by CPU.
→ Securing nothing but transfer data from device to CPU, that transfer is done by vector Address (VAD)
→ Another hardware method is "parallel priority Interrupt".
→ Here devices are connected parallely where as in previous hardware method, they are connected serially.

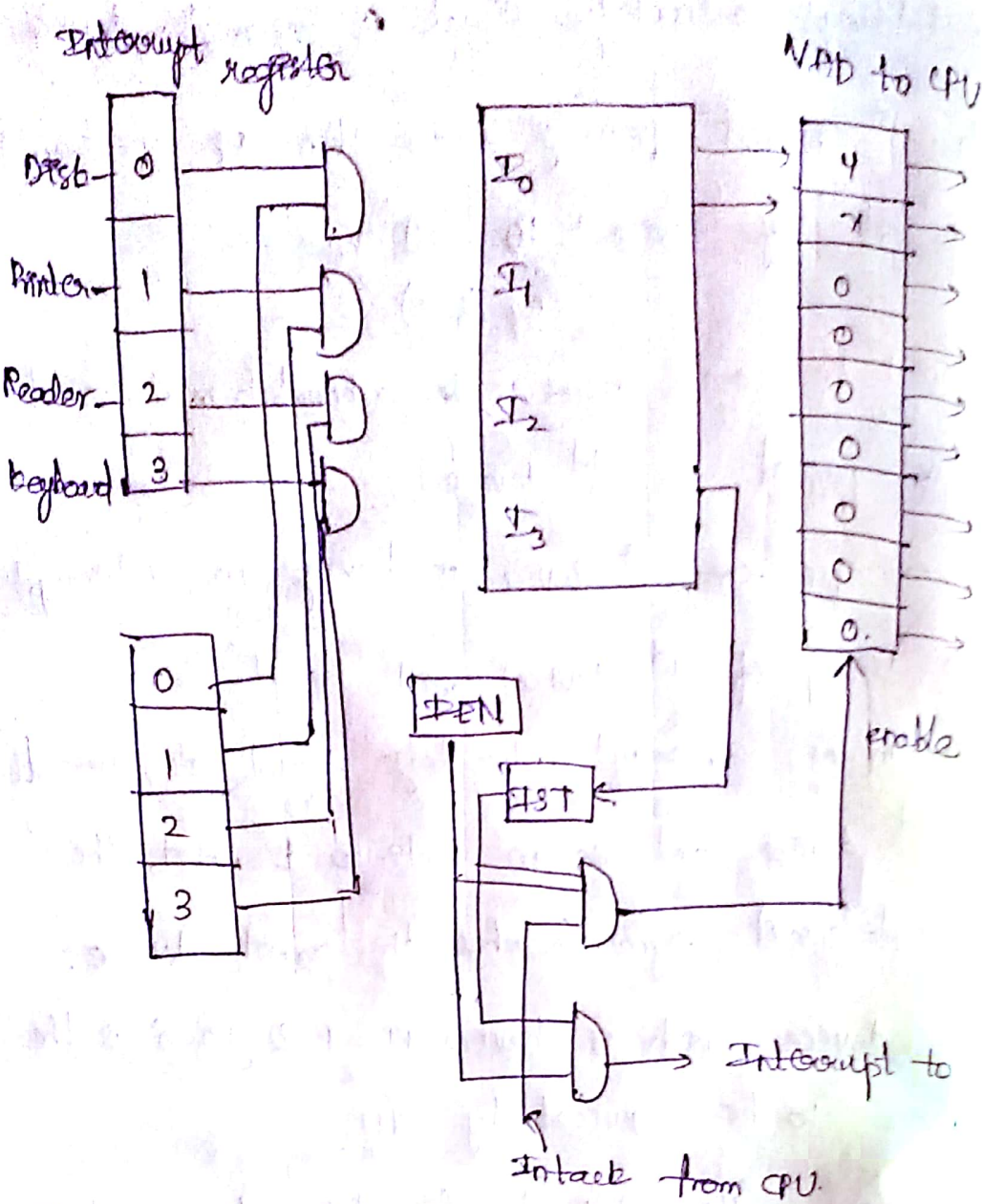


Fig. Priority interrupt hardware.

→ In interrupt register the devices are placed from top to bottom with high priority first and lowest priority device at bottom.

→ The purpose of mask register is not to allow the lowest priority interrupt to CPU when CPU is dealing with highest

priority interrupt.

→ Another purpose of Mask register is to allow the highest priority interrupt to CPU when lowest priority interrupt is being serviced.

→ When there is an Interrupt from Interrupt register & it is not noticed, then o/p of AND gate is '1'

→ Mask register means

If CPU is dealing with disk, then printer at same time giving interrupt, the Mask register should mask the printer device.

→ If any of the o/p of AND gate is '1', IST flip flop is activated.

→ The purpose of IEN - Interrupt enable signal

IEN = 1, the interrupt will be source &

IEN = 0, No " " " taken

→ After getting INTACK from CPU, we are enabling CPU sending VAD (Vector Address)

→ To VAD to CPU (register)

from device 1 → we have 00

" " 2 → " " 01

" " 3 → " " 10

" " 4 → " " 11

IST - Interrupt status flip flop.

Direct Memory Access:

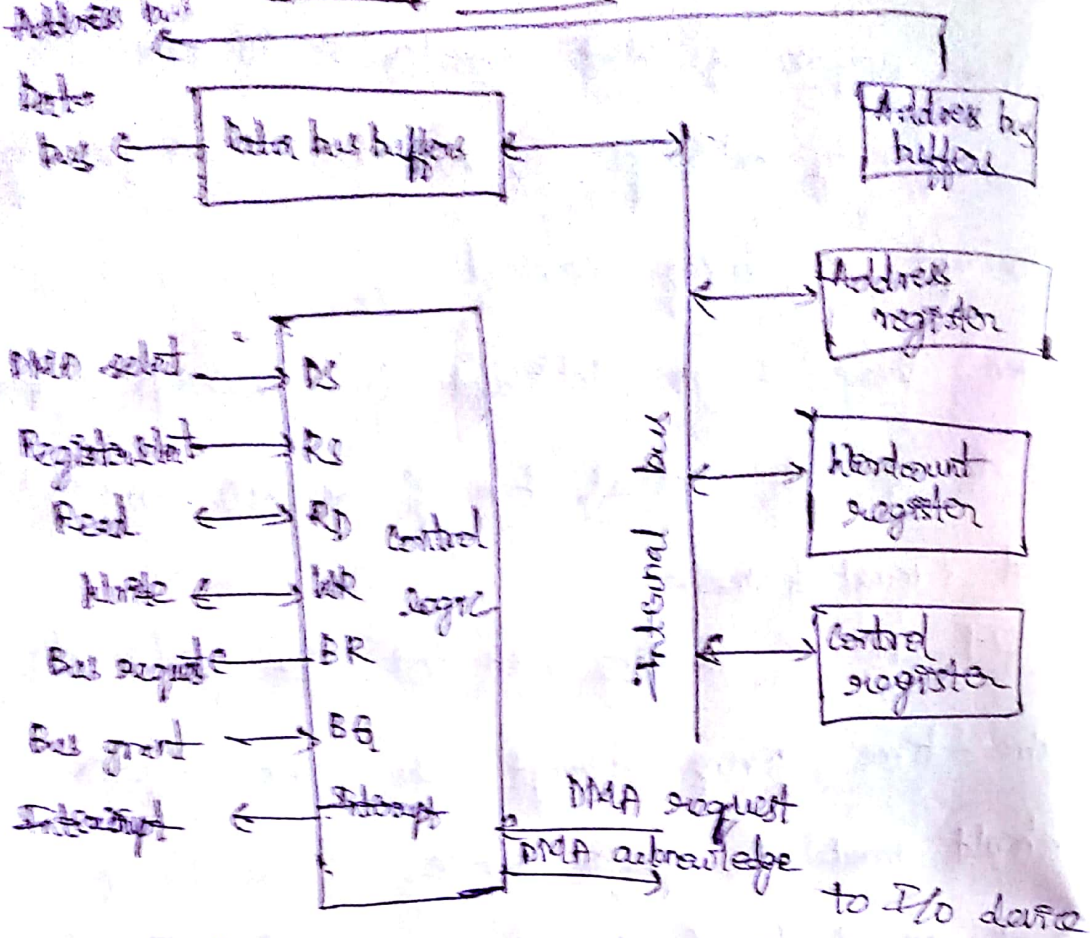


Fig: Block diagram of DMA controller

→ 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 register

→ 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 and bus requesting the control of signal to the CPU.

→ When bus request signal is sent, CPU terminates the execution of instruction (which is at that instant) and keeps the memory buses in high impedance 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 execution of the 3 pins DMA select pin and register select are enabled

↔ Read and write pins are bidirectional. They act as i/p and o/p. These pins act as i/p pins to CPU and o/p pins to memory

In order to load data into memory

→ When data is to be loaded

3 registers by CPU, Read/write pins are enabled.

→ After each word transfer, wordcount becomes 3000

→ When all words are transferred from wordcount register to I/O device & memory, interrupt is enabled and coordount will be 3000.

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

→ DMA acknowledge will be generated to I/O device by cont 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 coord count register will be decremented. If the wordcount value ≠ 0, I/O device will send a DMA request and DMA controller check the whether there is a DMA request → If the I/O device is fast it will send a DMA request is activated as soon as previous word is transfered and viceversa.

→ 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

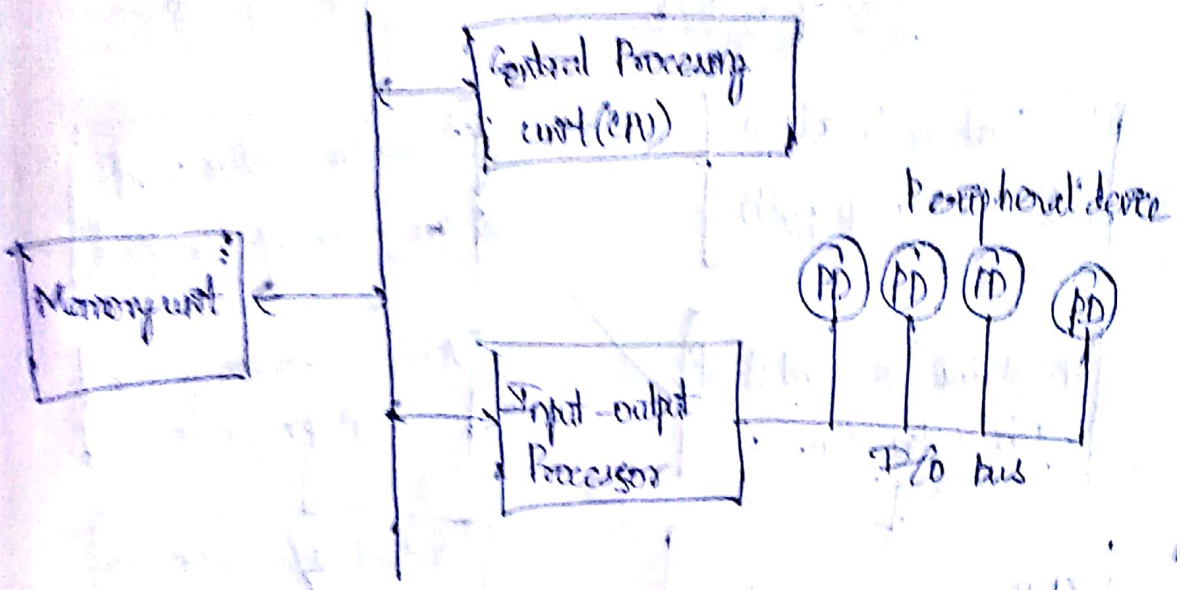


Fig: Block diagram of a computer with I/O processor

→ 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

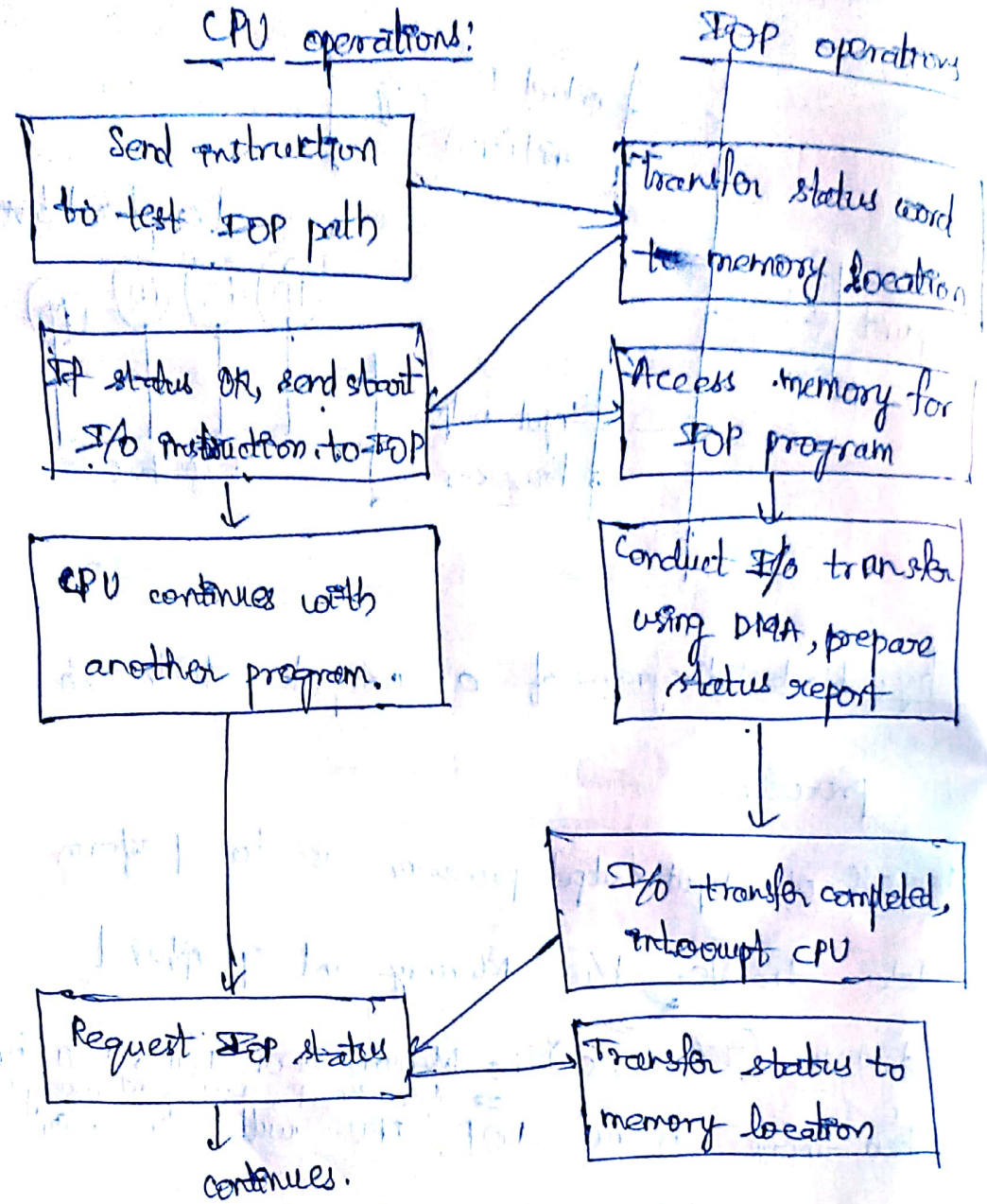
→ When there is no IOP, CPU will transfer data from memory to I/O device.

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

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

PS ready or not.

CPU-IOP communication



→ CPU sends 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 service 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)