10/01/18

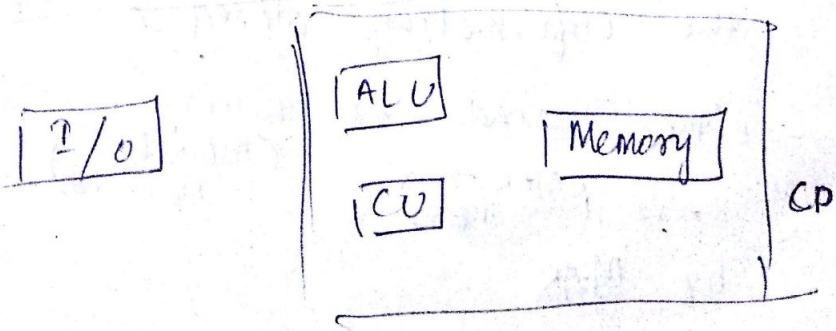
Q.1 From switch on to get window screen what are the steps ? (booting
beep sound
LED glow.
Loader (Boot strap)

2. Where the prog will save ?
compile - who is doing that
run - how
result - where

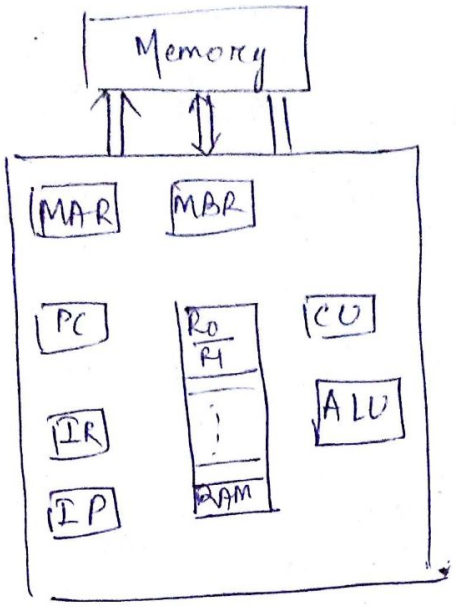
Assignment

* Generation of computer - vacuum tube
1st gen.
2nd gen - transistor.
3rd gen - IC
(Integrated ckt)
4th gen - Microprocessor
5th gen - AI, VLSI
VLSI
(very large scale integration)

* Micro computer, Mini Computer, Mainframe, Super computer



ALU + CU + with some register = processor



MAR - Memory Address Register.

PC - Prog. counter

IR - Instruction Register

IP - Instruct Pointer

MBR - Memory buffer ~~being~~ data register

Prog — Sequence of instruct^n

⊙ what to do - upcode

⊙ upon what the oper^n will be performed - data / operand part (you want to do)

Starting address given by Prog. counter.

⊛ funct of PC ⇒ gives the address of next instruction to be executed (give to MAR)

MAR - locate the adderess from or to the information stored or retrieved

MBR - either read or write located (MAR to MBR) (MBR to memory) by MAR.

of upcode, store in IR.

IR - stores the instruct^n which to be decoded / executed.

Address of that instruction that will be addressed by the IP.

IP - stores address of the instruction which is under decoding.

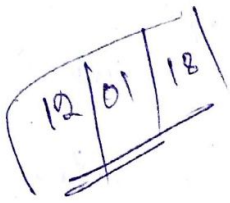
Prog. counter = Prog. Counter +

If it is data, it will go directly to ALU

→ Accumulator - ~~stored result~~ stores result.

- important for operation.

R_{15} - act as general register & accumulator



BUS - collectn of wires which transfers inform^n to diff. parts.

↳ require to address a memory locth

3 type of BUS

1) Data Bus
2) Address Bus
3) Control Bus

Type of Data Bus -

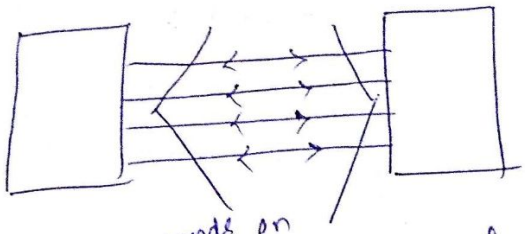
whether the data will transfered in

width of DB { or serial fashion
parallel fashion - word length.

word length - at a time how many bits are tranfered to the memory or from the memory.

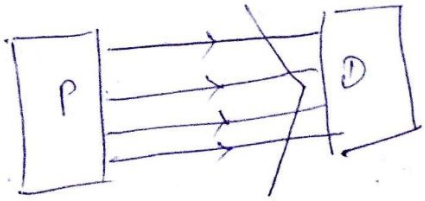
DBus which transfer the ~~data~~ from device to memory. and vice versa (bi directional) or (sender to reciever.).

B.



Address Bus (depends on max capacity of memory)
require to add. the loc^n of memory.

only processor can address any device.



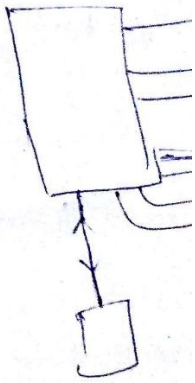
from processor to device
device can't address ~~address~~ processor
(thus unidirect^nal)

(*) width of Add. Bus - depends on max. capacity of memory.

e.g. for 16 location - 4 bit required
32 loc^n = 5 " 4

Control Bus

Contr



not all

* there is a concept of

2 types o

1) Single
2) Multi

disadvantag

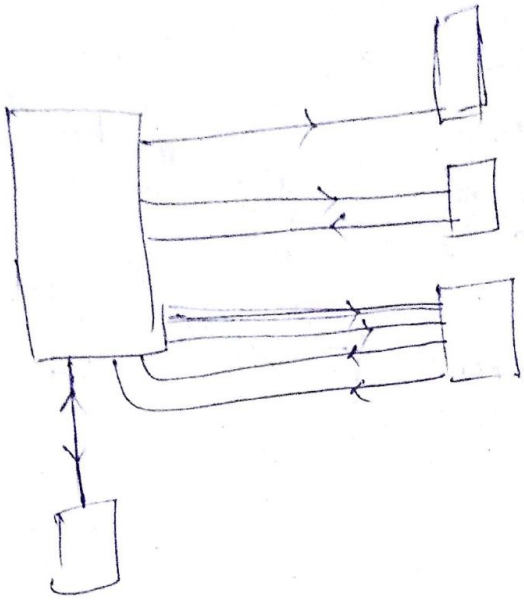
at

u

flexi

Control Bus

Control inform" will transfer

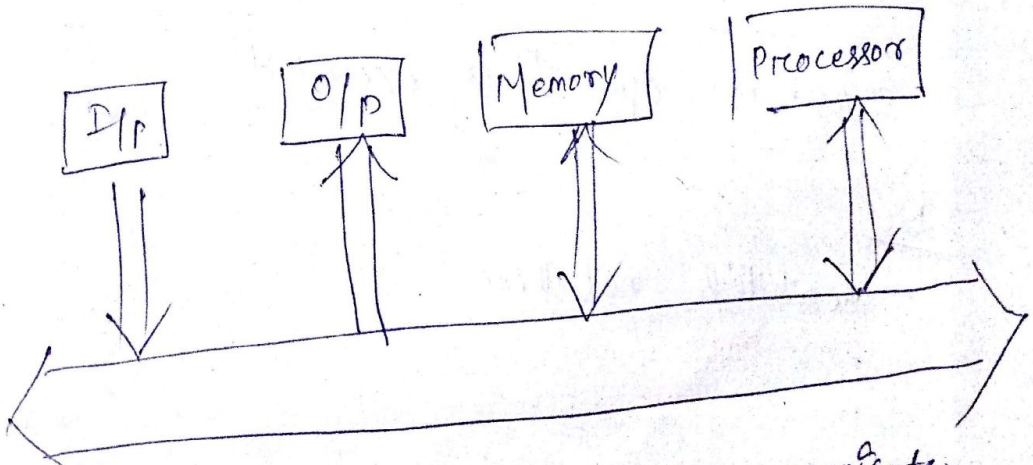


not always collect" of wires.

* there is a concept of direct" control wires.

2 types of BUS-Structure

1) Single Bus-structure.
2) Multi "

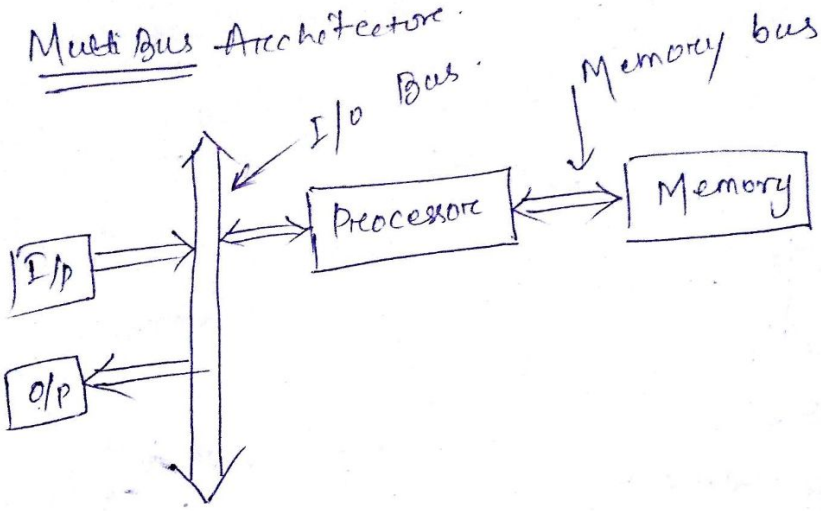


disadvantage -
at a time 2 units can communicate with each other.

flexibility -

Bus arbitration technology - decides which unit will communicate next

Multi Bus Architecture.



Some work required CPU bound
" " " IO bound
(much time

initiates the starting.

Speed of operation is executed.

dis

flexibility not there

Some control circuitary inbuilt in I/o devices called as I/o channels or I/o processor.

17/01/18

Design ele

what

the

dedicated

8 b

16 b

Multiplexin

ALE-

BUS

Cen

Di

central

17/01/18

④ Address = 16 bit
Data = 8 bit

Design element of BUS -

what are the element upon which the BUS should be design

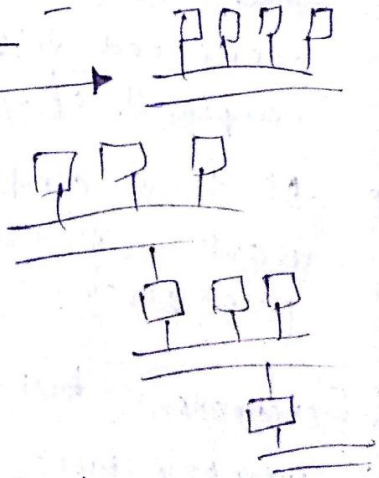
dedicated BUS
8 bit - transfer of data - data bus
16 bit - " " address - address bus

Multiplexing of BUS
↳ to reduce the hardware complexity.

ALE - Address Latch enable
↳ when the processor starts work,
when ALE - is high - 16 bit BUS used as address bus
ALE - low - from 16 bit BUS the lower 8 bit BUS will act as data bus.

* BUS Arbiteration Method -

Centralised Bus
Distributed Bus



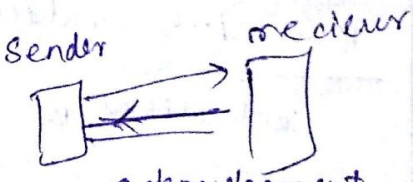
Centralised
there is a arbitor, which pair of units at this time will communicate

Distributed

divided into diff. module,
each module having one
access control logic.
↳ a hardware unit with mechanism of controlling bus.

one will - master,
others - slave.

* Timing

Synchronous Bus	Asynchronous Bus
* occurence of event depend on time	* occurence of event depends on occurence of prev. event doesn't depends on time.
* On this Bus str, with Bus bit, another clock line is there. That clk line also connected diff. component of system.	* if prev. event is not occur then next process can't work.
* All these component work with clock & processor.	* eg - I/O Bus
* Synchronous Bus is memory Bus - Bus connected processor with memory.	

* Types of Data Transfer -

whether 1 line is req. for read

or

1 line is for both read & write

i.e line { high - read
low - write

serial data transfer

parallel data transfer.

read after write/- only one address

write after read

* block signal

* Band width -

bits / sec.

→ Von Neuman concept & IAS Architecture.

both instruction & data will be in same area

will be write)

(instruction followed by data)

IAS - Institute of Advance Study

18/01/18

IAS computer

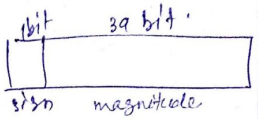
storage loc^n - 1000

coord length - 40 bits

In storage part we can store both instruct^n & data.

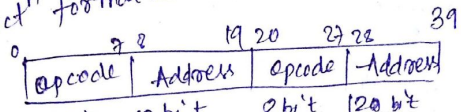
1 bit = sign

39 bit = magn.



each instruct^n = 20 bits

instruct^n format —



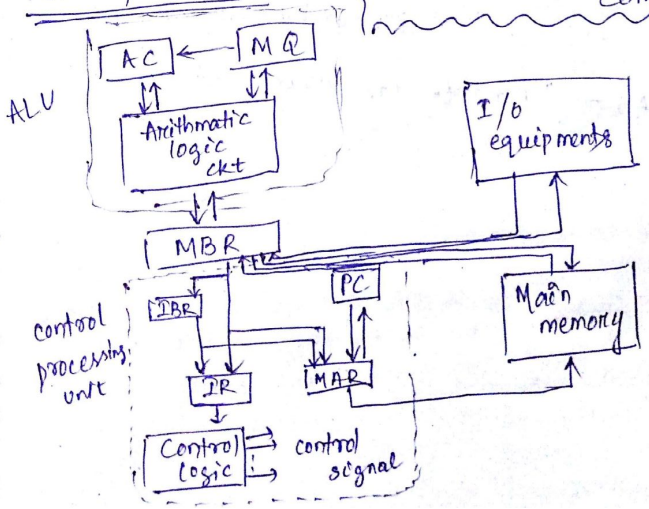
left instruct^n part

right instruct^n part

in Address - max. loc^n = 2^{12}

For operation —

BLOCK STR. OF IAS COMPUTER



MBR -

MQ - Multiply quotient

MAR -

PC -

IR -

DBR - Instruct^n Buffer Register. (right instruct^n)
- Keeps the right most instruct^n

AC - (Accumulator)
- special purposed register which takes part in processor
- atleast one operand should be in accumulator
- ~~of~~ result ~~for~~ will automatically store here if not specified

* If we get result > 40 bit then LSB 40 bit will be in "AC" and next 40 bit will be in "MQ"

* If sequence address - that will communicated by prog. counter.

if not sequential - that address will communicated by MAR

If data → then direct go to ALU.

instruc^n → then decoded.

PTO →

1) Data transfer Instruct^n ★
2) Arithmatic "
3) Unconditional branch "
4) Conditional Branch. "
5) Address modifier

STOR - AC to memory
LOAD - memory to AC

1) Data Transfer Instruction

i) Load MQ → The content of MQ register will copied to AC.

ii) LOAD MQ, M[x] - content of 'x' loc^n that will be loaded to MQ.

iii) STOR M[x] - content of AC will be stored in the 'x' loc^n. of memory.

iv) LOAD M[-x] - content of '-x' will be loaded to AC.

2) Arithmatic Instruct^n

i) ADD M[x] = content of AC will be added content of 'x' loc^n. - result will be in AC

ii) LSH - content of AC will be shifted left by 1 unit. (multiply by 2)

iii) RSH -

3) Unconditional Branch

(left instruct^n) JUMP M[x, 0:19] - in sequential prog.
- 'x' jump to
- the instruct^n 0-19
(left half)

(right instruct^n)
JUMP M[x, 20:39] -

4) Conditional Branch -

JUMP +M[x, 0:19] - if no. is non-negative
* then only jump otherwise
go sequentially.

5) Address modifier -

STOR M[x, 8:19] - (Address given)
- the add. field of M[x]
- replace x locn of 8:19, by 12 bit value.
of Accumulator

(7) Assignment -
Instruction flow chart of IAS computer

19/01/18

	+0	−0
sign magn	0 000 0000	1 000 0000
1's compl.	0 000 0000	1 111 1111
2's "	0 000 0000	none

- for general sign $\mp (2^{n-1} - 1)$
- for 1's complement $\mp (2^{n-1} - 1)$
- for 2's complement $-(2^{n-1})$ to $+(2^{n-1} - 1)$ (+ve 0)

(no negative zero)

floating pointer no. –

$m \times r^e$

m = mantisha part
e = exponent part
r = radix (eg - for binary 2)

2 register require
1 is for mantisha
&
1 is for exponent.

Q. 563.52

$= 0.56352 \times 10^3$

$m = 0.56352$

$e = 3$

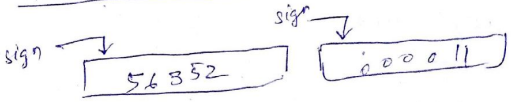
$r = 10$

Q. 11101.001

$= 0.11101001 \times 2^5$

$= 0.11101001 \times 10^{0101}$

General Representation –



Mantissa. Exponent

* In invisible way radix pt. – that is why taken to extreme length.

① Addⁿ –

always makes the exponent part equal.

$5.323 = 0.5323 \times 10^1$

$603.23 = 0.603233 \times 10^3 = 60.3233 \times 10^1$

$(A \pm B) r^e = (60.8553) \times (10)^1$

② Multiplicⁿ – $(A \times B) r^{e_1+e_2}$

③ Divsⁿ – $(A/B) r^{e_1-e_2}$

(4) Assignment

(a). What do you mean by normilization of floating pt. no. ? Represent with example

(a) IEEE- standardise IEEE representation of floating pt. no. (using 1 register.

30/01/18 Multiplication

```
  10111
  10011
  -----
  10111
 10111
10111
00000
00000
10111
-------
1101101
```

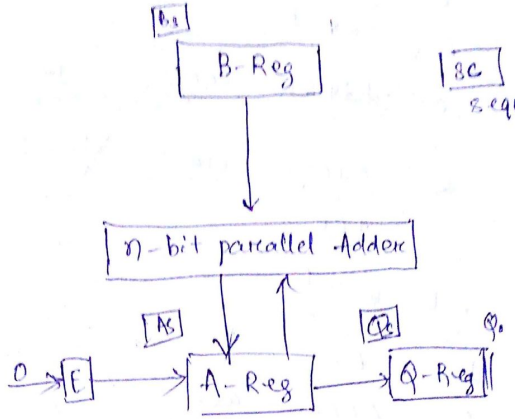
A = $B_s \oplus Q_s$

General
- Right shift the result by one bit.

When 1 : shift

Multiplicand in B-reg

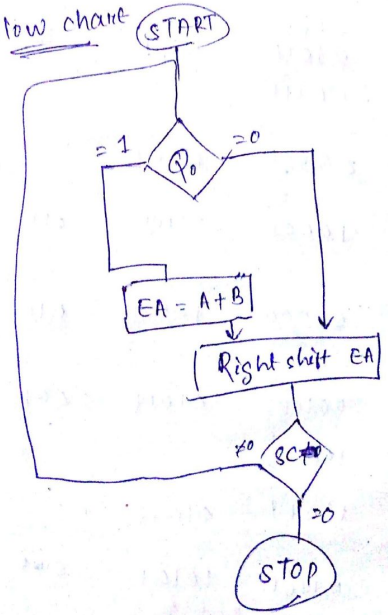
Multiplier in Q-reg

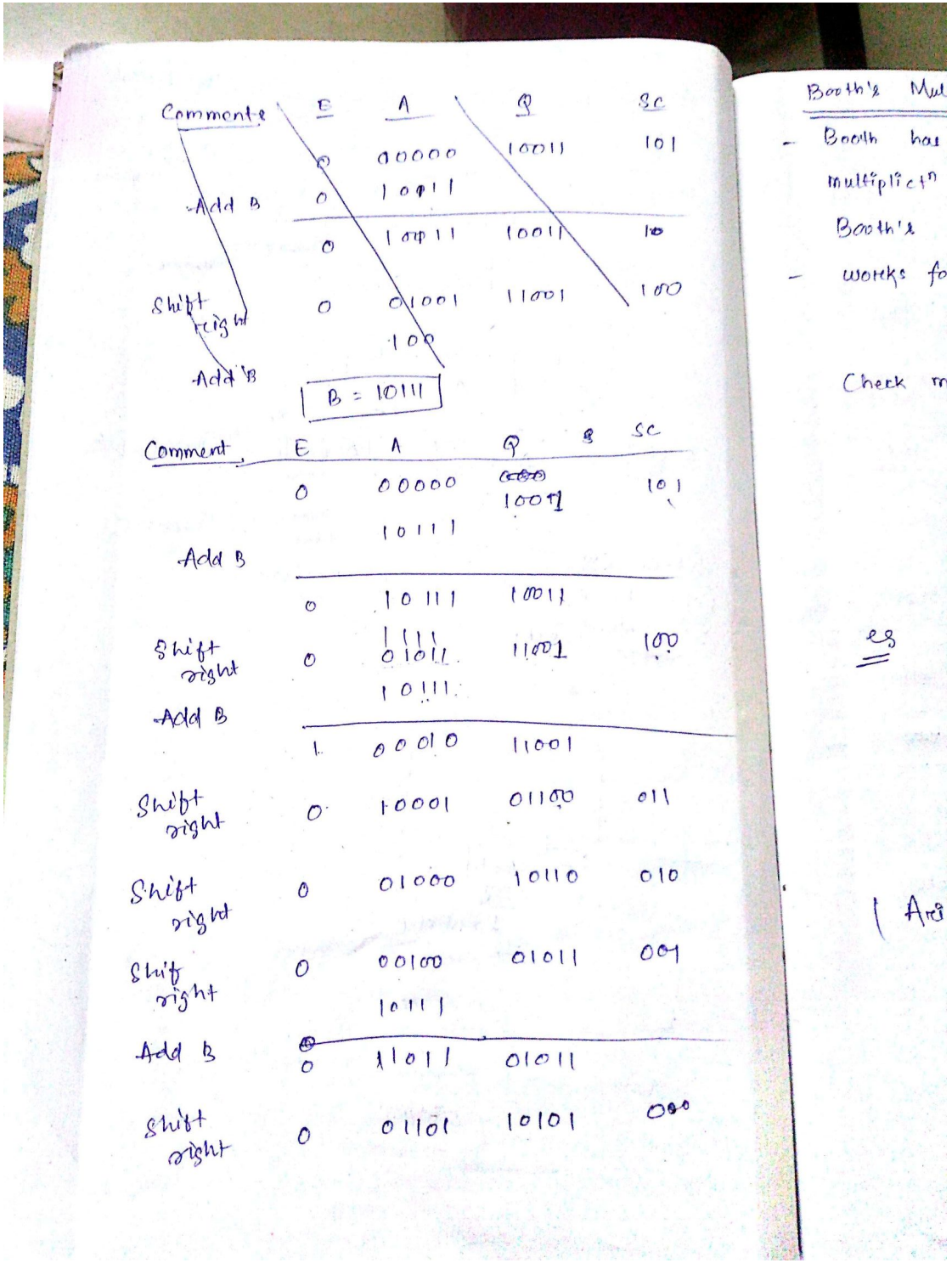


$A_S = Q_S \oplus B_S$

* When sequence counter = 0, end of product
* Final Result will be in AQ-Register.

Flow chart





Comments	E	A	Q	SC
	0	00000	10011	101
Add B	0	10111		
	0	10111	10011	100
Shift right	0	01001	11001	011
Add B		10111		
	1	00000	11001	010
Shift right	0	10000	01100	001
Shift right	0	01000	00110	000
Shift right	0	00100	00011	

B = 10111

Booth's Mul

- Booth has
multiplicn
Booth's
- works fo

Check m

eg

Ans

Booth's Multiplication Algorithm

- Booth has developed another type of multiplicⁿ algoᵐ which is known as Booth's Multiplicⁿ Algoᵐ.
- works for signed 2's complement.

Check multiplier,

eg

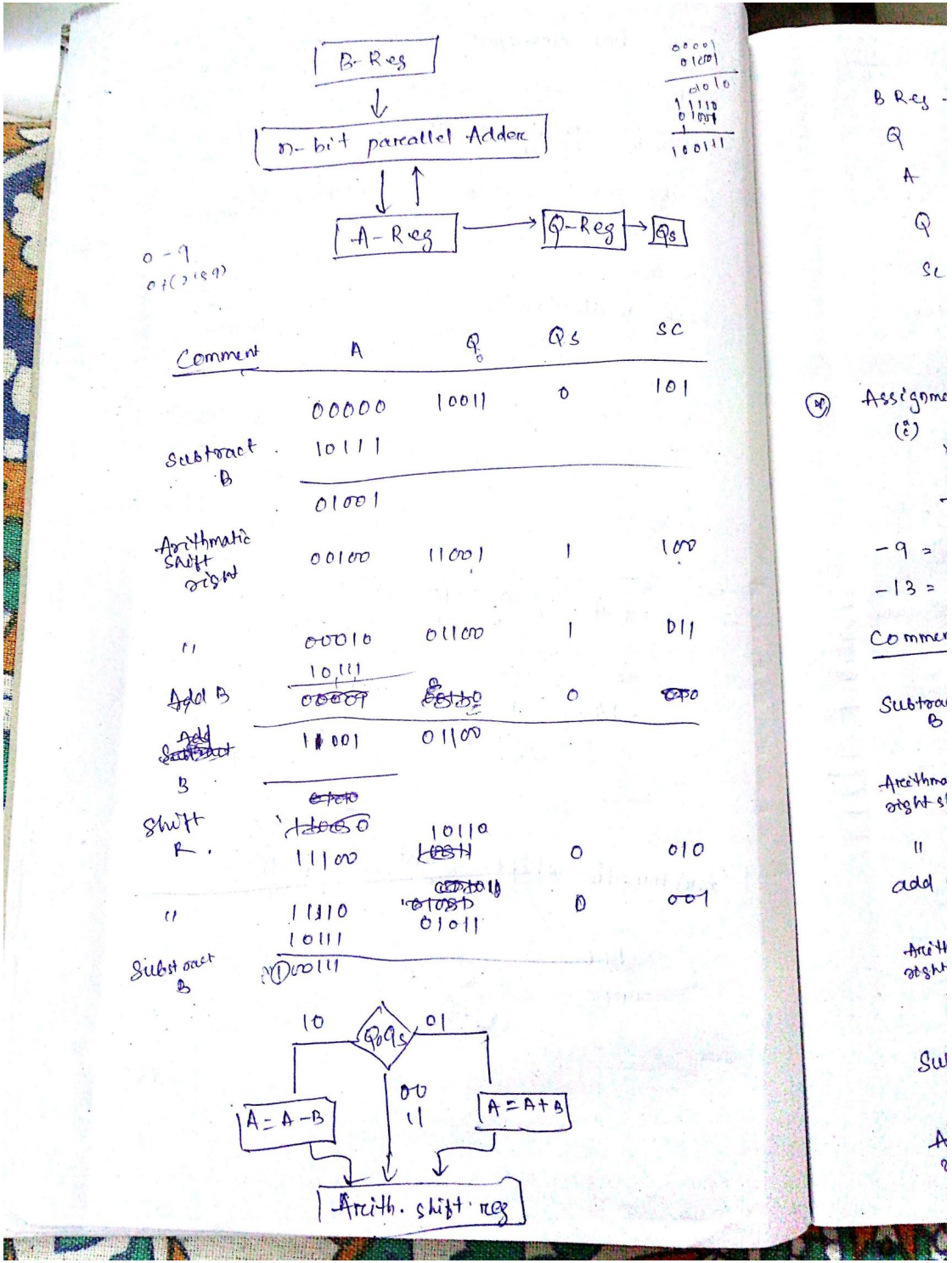
+14 0 1110
$+2^4$ -2^1 $= 16-2 = +14$

-14 1 0010
-2^4 $+2^2-2^1$ $= -16+4-2$
$= -14$

Arithmatic shift right ⇒ 1100
1110

(1)10
111

PTO→



B Reg - Multiplier

Q - Multiplicand

A - 0's

Q

Sc - bit in Multiplier

01000
01001

11110
01001

111

11110
01001

00111

(*) Assignment

(i) x -9 (ii) x -9

-13 +13

8421
9 = 1001
0110
1011

1101
0010
0011

01000
01001

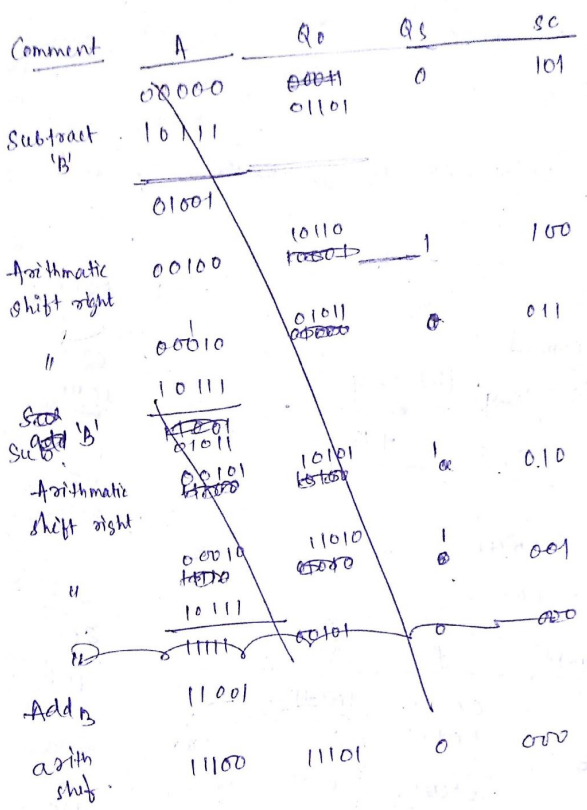
-9 = 10111

-13 = 10011

Comment	A	Q0	Qs	Sc
	00000	10011	0	101
Subtract B	10111			
	01001	10011	0	
Arithmatic right shift	00100	11001	1	100
"	00010	01100	1	011
add B	10111			
	11001	01100		
Arithmatic right shift	11100	10110	0	010
"	11110	01011	0	001
Subtract B'	10111			
	00111	01011		
Arithmatic right shift	00011	10101	1	000

(ii) -9 = 10111

+13 = 00011 01101



(20/22)

$-9 = 10111$

$+13 = 01101$

$00001 + 01001 = 01010$

$11101 + 01001 = 00110$

Comment	A	Q_0	Q_s	Sc
	00000	01101	0	101
Subtract B	10111			
	01001			
Ar. Shift right	00100	10110	1	100
add B	10111			
	11011			
A.S. right	11101	11011	0	011
Sub. B	10111			
	00110			
A.S. right	00011	01101	1	010
"	00001	10110	1	001
Sub. B	10111			
	01010			
A.S. right	00101	01011	0	000

Ans. (00101 01011)

Ans.

$1101010100 + 1 = 1101010101$

Array Multi

4 bit
3 bit
m x n

Applications

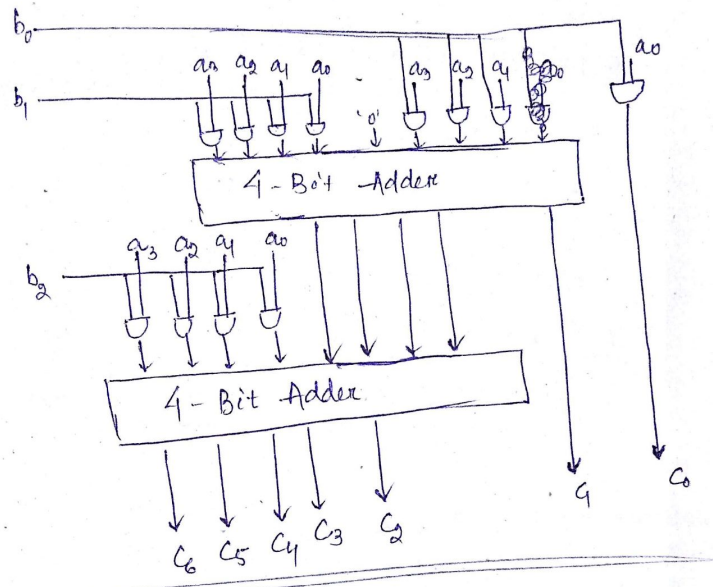
BCD - add

Decimal-

(Block- diagram)

(Cell-diagram)

Array Multiplier :-



4 bit multiplicand – m
3 bit multiplier – q
m x q = no. of AND gate
(q-1) = no. of adder (n-bit)

Assignment

(#) BCD - add^n
Subtraction.

Decimal - add^n
Subtract^n
9's complement
10's complement

(*) Division Algorithm

Auxiliary carry - generated
then add 6.

01/02/18

Instruction—

* binary no. which is already designed within the processor.

* Total no. of instruction = instruct^n set of that processor.

opcode	operand/Address

Macro instruction —

- This instruction is decoded by CU.
- CU then generates control signal.
- These are communicated to the required divices of the system.

* Each small operat^n are called as micro instruction. → with in CU

* Some micro instruct^n are to be followed to complete macro instruction.

eg

PC ← initialised
MAR ← PC
MBR ← m[MAR]
if instruct^n.
IR ← MBR.

Flag - Register

Memory location Register is required for operation

PTO→

Flag Register –

X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AC	X	PF	X

CF = Carry flag
PF = Parity flag.
AC = Auxiliary carry
ZF = Zero flag – result will '0', this will set.
SF = Siagn flag – to check abt –ve & +ve no.
TF = Trap flag – non-maskable Interrupt

IF = Interrupt flag – General.
DF = Direct^{n} flow flag – low memory loc^{n} to high – set
reverse – reset
OF = Owerflow flag – register overflow
underflow, stack overflow — set
underflow — reset

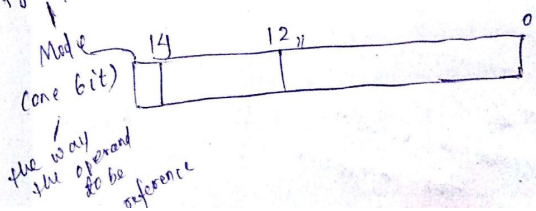
Interrupt

2 types – 1) maskable (vc with SIR).
2) non maskable (vc SIR with pcon).
↳ can't be mask.

How to design an Instruction ?

3 types of instruct^{n} – 1) memory reference instruct^{n}
2) register " "
3) Input output instruct^{n}

for finding of operand
1) memory has to be reference.
2) register has to be reference
3) I/o devices should be reference

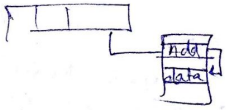


④ if mode bit = 0 ⇒ direct memory reference instruct^n

mode bit = 1 ⇒ indirect memory reference instruct^n

Direct :— The address given in add. part is memory reference. the add. of data.

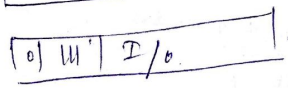
Indirect :— address of add. part gives the memory reference add. of data.



7 instruct^n
⇓
So, 000 to 110.

for 111 ?—

— 2^{12} no. of register I can reference



Generally, instruct^n depending upon function of instruction.

1) Data Transfer instruct^n — MOV Rd, Rs — from source to destin
2) Arithmatic instruct^n — ADD Rd, Rs
3) Logical " — AND, OR Rd, Rs
4) Branch control " — JUMP one, on carry
5) Machine control " — HLT (halt)
6) Data manipulation instruct^n
   - within Arithmatic
   - INR a (increment), DCR b (decrem

Rd = Rd + Rs

Rd = Rd · Rs bit wise AND with Rs, Rd.

Depending upon funct^n of instruct^n

(TYPES OF INSTRUCT^n PRESENT IN MICROPROCESSOR)

02/02/16

(*) Generally CPU may follow either ~~for~~ Org^n or more than one Org^n.

1) Single Accumulator Organisation
2) General Register Organisation
3) Stack Organisation.

~~OR~~ - Single Ac org^n -

~~ADD~~ ADD X

AC ← Ac + M[x]

STOR Y M[Y] ← AC

General register -

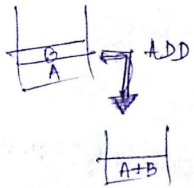
ADD R_1, R_2, R_3

ADD R_1, R_2 ($R_1 \leftarrow R_1 + R_2$)

ADD R_1, X ($R_1 \leftarrow R_1 + M[x]$)

Stack Organisation

PUSH A
PUSH B
ADD



(*) Various types of instruct^n format depending upon no. of add. field in instruction. —

i) Three Address Instruct^n - 3 add. field
ii) Two " " - 2 " "
iii) One Address Instruct^n - 1 " "
iv) Zero " " - 0 " "

Q1 - Y = (A+B) * (C+D)

1) Three Address Instruction —

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

2) Two Address Instruction —

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

* from loc. to reg] load
from Reg - loc storing

3) One Address Instruction —

LOAD	A	$AC \leftarrow M[A]$
ADD	B	$AC \leftarrow AC + M[B]$
STOR	X	$M[X] \leftarrow AC$
LOAD	C	$AC \leftarrow M[C]$
ADD	D	$AC \leftarrow AC + M[D]$
MUL	X	$AC \leftarrow AC * M[X]$
STOR	Y	$M[Y] \leftarrow AC$

4) Zero Address Instruction

PUSH A
PUSH B
ADD
PUSH C
PUSH D
ADD
MUL
POP Y