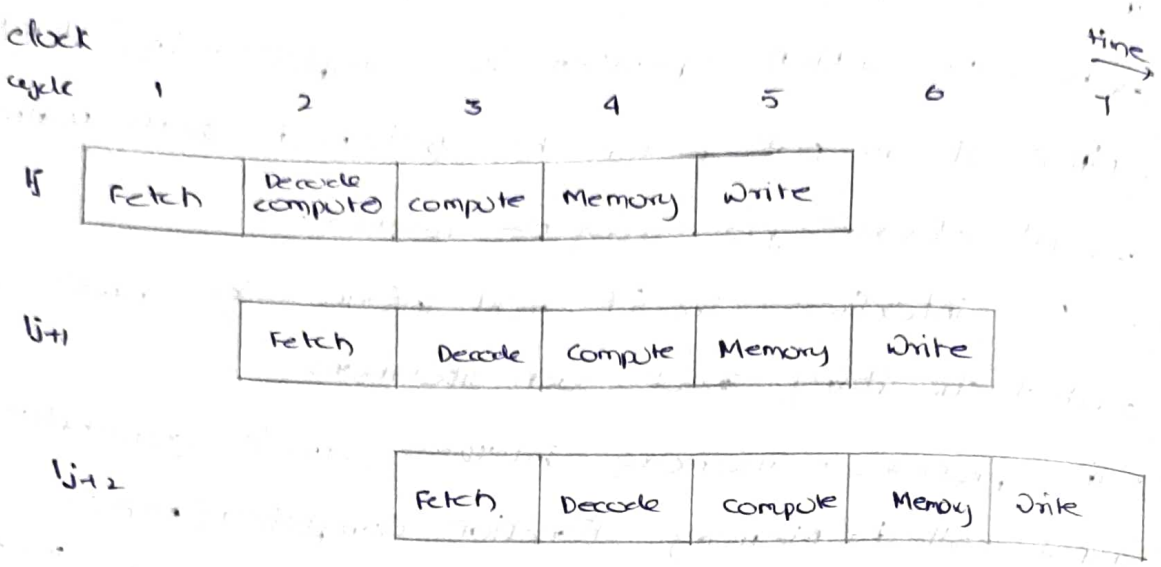
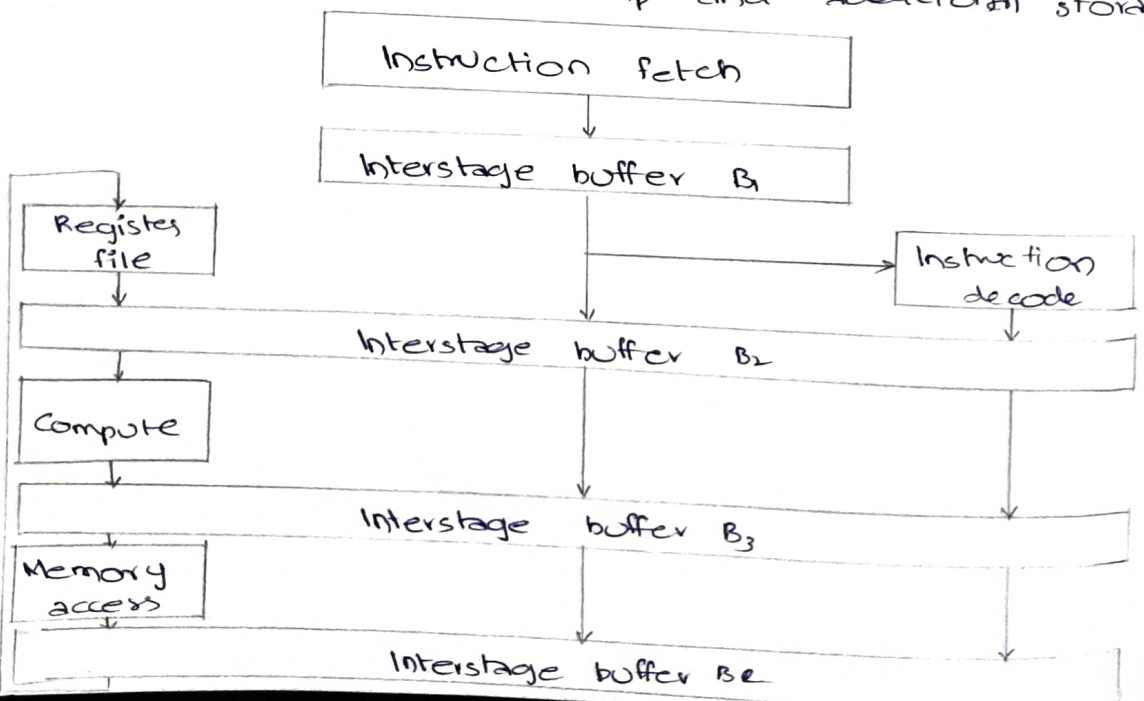
Pipelining :-

• Pipelining is a particularly effective way of organizing concurrent activity in a computer system.



Pipeline organization :-

In the first stage, PC is used to fetch a new instruction. As other instructions are fetched, execution proceeds through successive stages. Information such as registers addresses, immediate data, etc must be carried through the pipeline from one stage to next. This information is held in interstage buffers. These include R_A, R_B, R_M, R_Y, R_2, IR, PC-temp and additional storage.



Pipelining issues:-

- There are times when it is not possible to have a new instruction enter the pipeline in every cycle.
- Any condition that causes pipeline to stall is called a hazard.

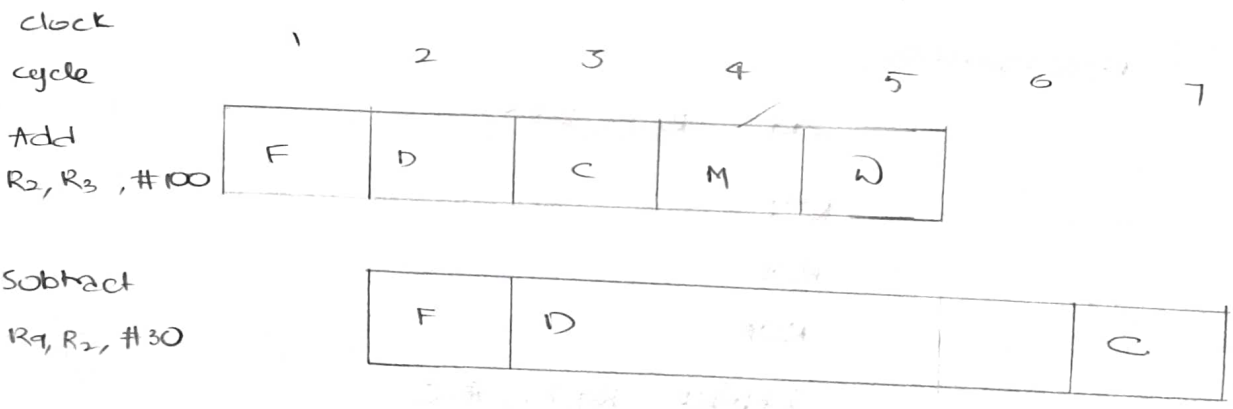
Data dependencies:-

Consider

Add R2, R3, #100

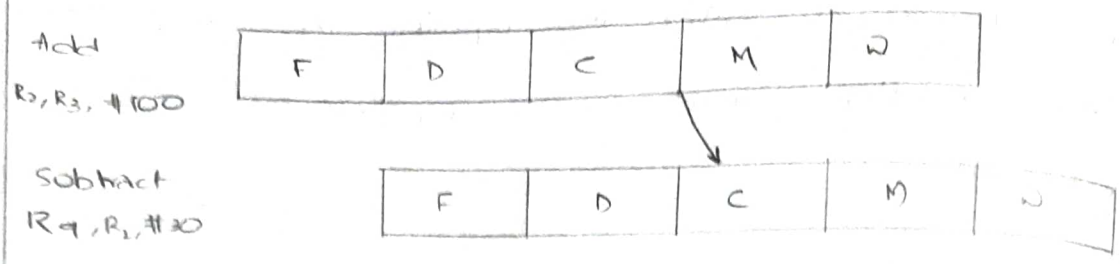
Subtract R9, R2, #30

The destination R_2 for Add is a source for subtract. There is a data dependency between these two instructions the subtract instruction is stalled for three cycles to delay reading register R_2 until cycle 6 when the new value becomes available.



Operand forwarding:-

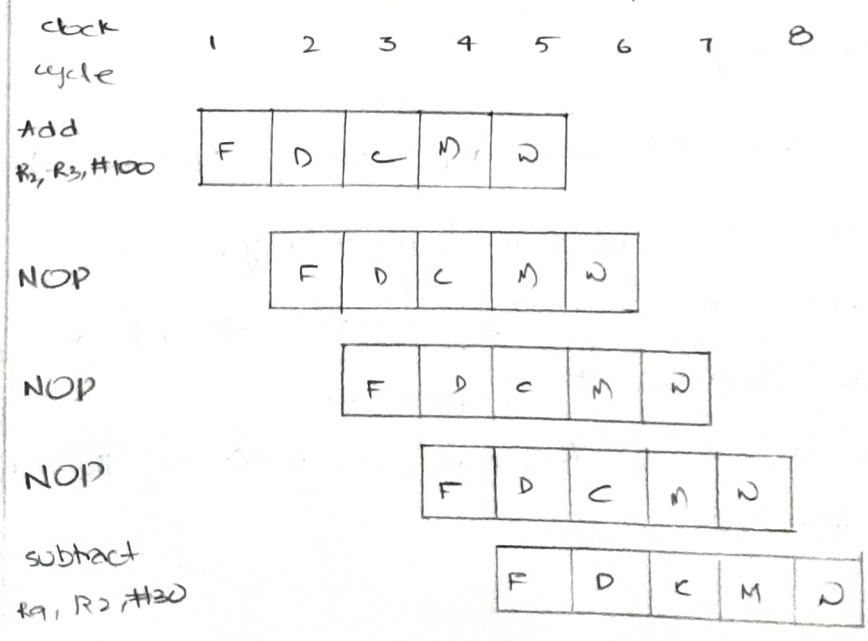
- Pipeline stalls due to data dependency can be alleviated through the use of operand forwarding.
- The desired value is actually available at end of cycle 3, when the ALU completes the operation
- The value is stored in R_2, part of buffer B_3, the hardware can forward the value



Handling data dependencies in software:-

- An alternate approach is to leave the task of detecting data dependencies and dealing with them to compiler.
- When the compiler identifies a data dependency between two successive instructions, it can insert explicit NOP (NO-operation) instructions between them.
- However the code size increases and the execution time is not reduced
- The compiler attempt to optimize code to improve performance

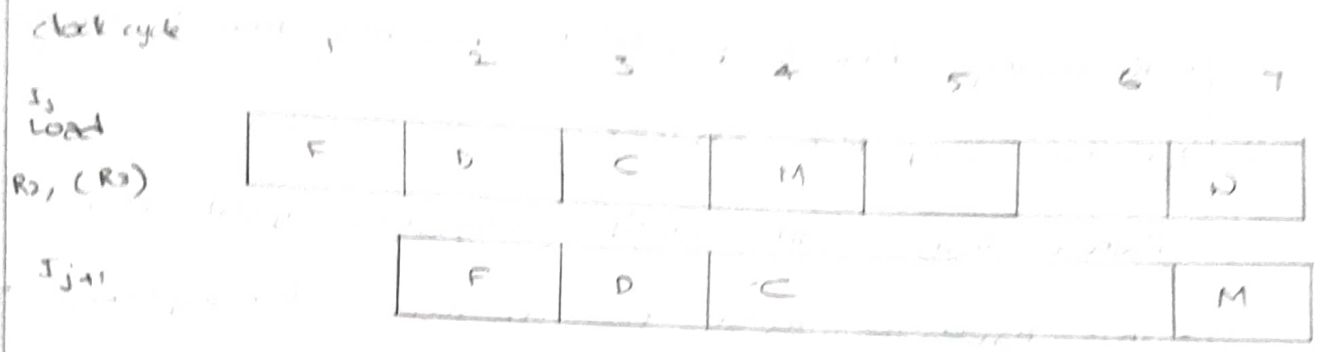
```
Add     R2, R3, #100
NOP
NOP
NOP
Subtract  R9, R2, #30
```



Memory delay:

- Delays arising from memory accesses are another cause of pipeline stalls.
- A load instruction may require more than one clock cycle to obtain its operand from memory.
- Because, the requested instruction or data are not found in the cache, resulting in cache miss.

```
Load    R2, (R3)
Subtract R1, R2, #30
```

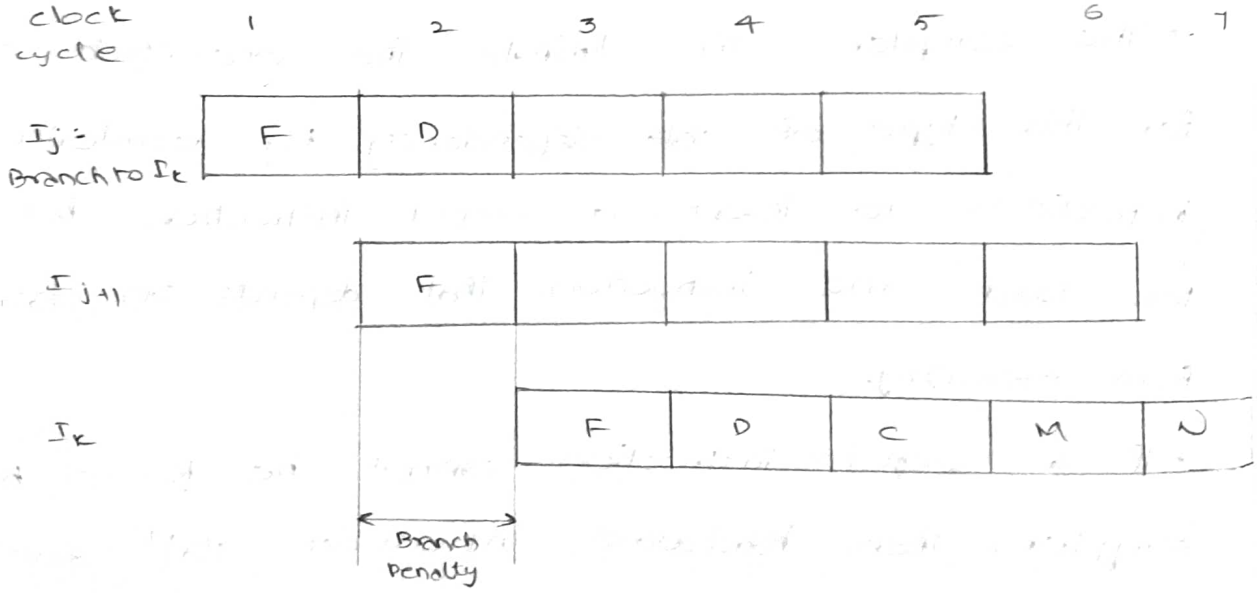


- The compiler can eliminate the one-cycle stall for this type of data dependency by reordering instructions to insert a useful instruction between the load and instruction that depends on data read from memory.
- If a useful instruction cannot be found by the compiler, then hardware introduces stall automatically.
- If hardware doesnot deal with dependencies, then the compiler must insert an explicit NOP instruction

Branch delays:-

Unconditional branches:-

- The branch instruction is fetched in cycle 1, deco in cycle 2 and target address is computed in cycle 3 Hence, target is fetched in cycle 4 after the PC has been updated with target address.
- In pipelined execution, I_{j+1} & I_{j+2} are fetched in 2 & before the branch instruction is decoded
- The resulting delay is branch penalty
- Reducing branch penalty requires branch target to be computed earlier.
- Rather than wait until compute, update PC in decode
- The hardware must be modified to implement this



- When the instruction decoder determines that the instruction is indeed a branch instruction, the computed target address will be available before the end of the cycle.
- It can then be used to fetch target instruction in next cycle

conditional branches:-

```
        Add          R7, R8, R9
    Branch_if_[R3]=0  TARGET
        Ij+1
        :
TARGET: Ik
```

- The location that follows branch instruction is called branch delay slot
- The compiler attempts to find a suitable instruction to occupy the delay slot, one that needs to be executed even when the branch is taken.
- If a useful instruction is found, then there will be no branch penalty.
- If no useful instruction, a NOP must be placed instead.

```
    Branch_if_[R3]=0    TARGET
        Add             R7, R8, R9
        Ij+1
        :
TARGET: Ik
```

- In the above example, Add can be safely moved into branch delay slot
- Add is always fetched and executed even if the branch is taken

Branch prediction:-

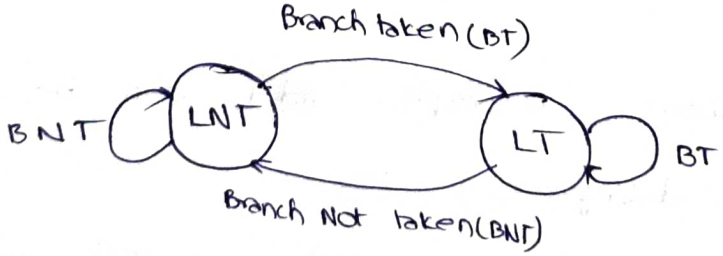
- To reduce the branch penalty further, the processor needs to anticipate that the instruction being fetched is a branch instruction and predict its outcome to determine which instruction should be fetched in cycle 2

Static branch prediction:-

- The simple approach is a form of static prediction
- The same choice is used for every time a conditional branch is encountered.
- The processor can determine the static prediction as taken or not-taken by checking sign of the branch offset.

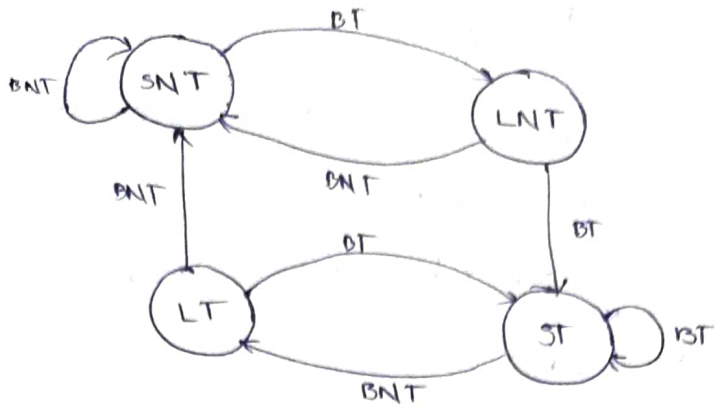
Dynamic branch prediction:-

- The processor hardware assesses the likelihood of a given branch being taken by keeping track of branch decisions every time that a branch instruction is executed.



LT- branch is likely to be taken

LNT - branch is likely not to be taken



ST- strongly likely to be taken

LT - likely to be taken

LNT - likely not to be taken

SNT - strongly likely not to be taken

30/03/2021

Resource limitations:-

- The pipeline stalls when there are insufficient hardware resources to permit all actions to proceed concurrently.
- If two instructions need to access the same resource in the same clock cycle, one instruction must be stalled to allow other instruction to use the resource
- This can be prevented by using additional hardware
- Single cache - Fetch and memory access.
- If both the fetch and memory stages of the pipeline are connected to the cache, then it is not possible for activity in both stages to proceed simultaneously
- One access at a time/cycle, not simultaneously
- Use separate caches for data and instructions

Performance evaluation:-

Non-pipelined system:-

In non-pipelined system, the execution time T of a program with N instructions is given by

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

where S = avg no of clock cycles to execute one instruction

R = clock rate in cycles per second

Instruction throughput:-

It is defined as no of instructions executed per second

$$P_{np} = \frac{R}{S}$$

Pipelined system:-

- If there is no cache miss, S is 5
- Thus, in absence of stalls, S is 1, and throughput with pipelining is

$$P_p = R \quad (S=1)$$

- An n-stage pipeline has the potential to increase the throughput by n times
- Real measure of performance is real execution time of a program

Effects and stalls of penalties:-

- Consider a processor with operand forwarding in hardware, so there are no penalties due to data dependencies, except in case of load instructions.
- Ideal pipelined execution has S=1, stalls due to such load instructions have the effect of increasing S by amount S_{stall}
- Assume that load instructions constitute 25% of the dynamic instruction, and 40% of these are dependent

- A one-cycle stall is needed in such cases,

$$S_{stall} = 0.25 \times 0.40 \times 1 = 0.10$$

- Execution time T is increased by 10%, throughput is reduced

$$P_p = \frac{R}{1 + S_{stall}} = \frac{R}{1.1} = 0.91R$$

- Consider penalties due to mispredicting branches during program execution.

Assume that branches constitute 20% of dynamic count of a program, and average prediction accuracy is 90%, so 10% are executed incur a one-cycle penalty.

$$S_{branch\text{-}penalty} = 0.20 \times 0.10 \times 1 = 0.02$$

- The time to access the slower main memory is a penalty that stalls the pipeline for P_m cycles every time there is a cache miss. A fraction m_i of all instructions that are fetched incur a cache miss. A fraction d of all instructions are load or store, and m_d are instructions that incur a cache miss.

$$S_{miss} = (m_i + d \times m_d) \times P_m$$

When all factors are combined, S is increased from the ideal value of 1 to $1 + S_{stall} + S_{branch\text{-}penalty} + S_{miss}$.

Arithmetic:-

Addition/subtraction of signed numbers:-

x_i	y_i	carry-in c_i	sum s_i	carry-out c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

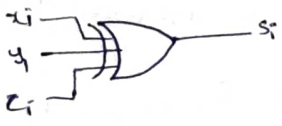
$S_i = \bar{x_i}\bar{y_i}c_i + \bar{x_i}y_i\bar{c_i} + x_i\bar{y_i}\bar{c_i} + x_iy_ic_i = x_i \oplus y_i \oplus c$

$c_{i+1} = \bar{x_i}y_ic_i + x_i\bar{y_i}c_i + x_iy_i\bar{c_i} + x_iy_ic_i$

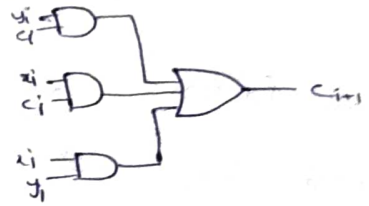
$= y_ic_i(\bar{x_i} + x_i) + x_i(\bar{y_i}c_i + y_i\bar{c_i})$

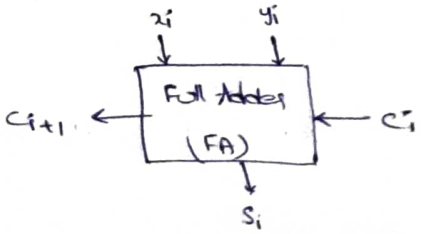
$= y_ic_i + x_ic_i + x_iy_i$

Sum:-



Carry:-

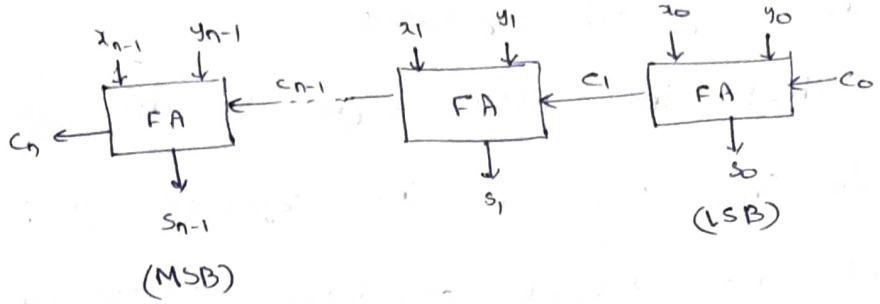




n-bit adder:-

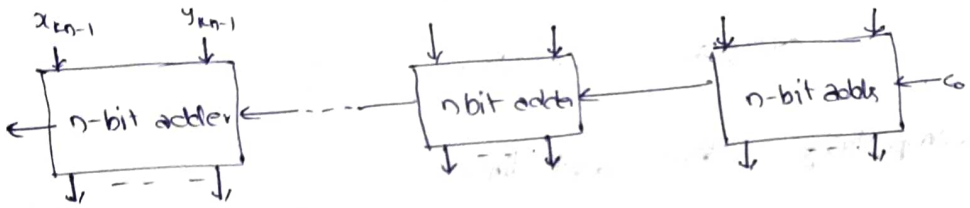
. cascade n Full adder (FA) blocks to form a n-bit adder.

. carries propagate or ripple through this cascade, n-bit ripple adder



k n-bit adder:-

k n-bit numbers can be added by cascading k n-bit adders

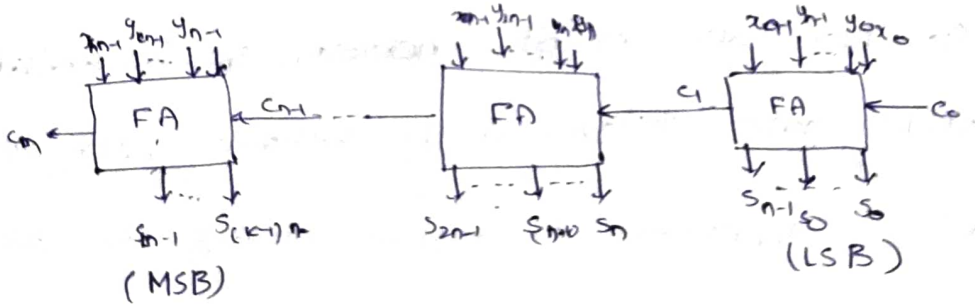


n-bit subtractor:-

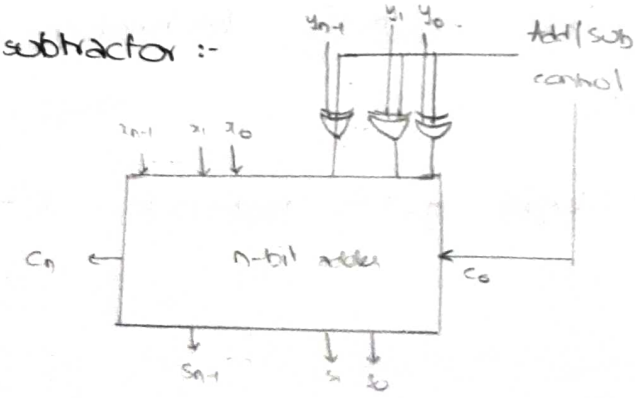
. x-y is equivalent to adding 2's c of y to x

. 2's c is equivalent to is c +1

. $x-y = x + \bar{y} + 1$



n-bit adder/subtractor :-



add/sub control:-

=0, addition

=1, subtraction

02/04/24

Detecting overflows:-

- Overflows can only occur when the sign of two operands is the same
- Overflow occurs if the sign of result is different from the sign of operands
- Recall that the MSB represent the sign.

x_{n-1}, y_{n-1}, s_{n-1} represent the sign of x, y and s

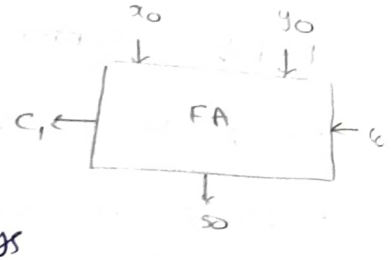
- Circuit to detect overflow can be implemented by the following logic expression:

$$\text{overflow} = x_{n-1} y_{n-1} \bar{s}_{n-1} + \bar{x}_{n-1} \bar{y}_{n-1} s_{n-1}$$

$$\text{overflow} = c_n \oplus c_{n-1}$$

computing the add time:-

consider 0^{th} stage:



- c_1 is avialable after 2 gate delays
- s_0 is available after 1 gate delay
- c_2 is available after 4 gate delays
- s_1 is available after 3 gate delays

So - For an n-bit adder, s_{n-1} is available after $2n-1$ gate delays, c_n is available after $2n$ gate delays

* Two approaches to reduce delays in addition

- Fastest possible electronic technology in ripple carry adders
- Augmented logic gate network structure

Fast addition:-

$$S_i = x_i \oplus y_i \oplus c_i$$

$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

$$c_{i+1} = x_i y_i + c_i (x_i + y_i)$$

$$c_{i+1} = G_i + P_i c_i$$

where $G_i = x_i y_i$

$P_i = x_i + y_i$

- G_i is called generate function and P_i is called propagate function.
- G_i, P_i are computed only from x_i & y_i and not from c_i, thus they can be computed in one gate delay after x and y are applied to the inputs of an n-bit adder

Carry-look ahead adder:-

$$c_{i+1} = G_i + P_i c_i$$

$$c_1 = G_{i-1} + P_{i-1} c_{i-1}$$

$$c_{i+1} = G_{i+1} + P_{i+1} (G_{i-1} + P_{i-1} c_{i-1})$$

$$c_{i+1} = G_i + P_i (G_{i-1} + P_{i-1} (G_{i-2} + P_{i-2} c_{i-2}))$$

$$\vdots$$

$$c_{i+1} = G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} + \cdots + P_i P_{i-1} \cdots P_0 c_0$$

- All carries can be obtained 3 gate delays after X, Y & C_0 are applied
  - → one gate delay for P_i and G_i
  - → Two gate delays in AND-OR circuit for c_{i+1}
- All sums can be obtained 1 gate delay after the carries are computed
- Independent of n, n-bit addition requires only 4 gate delays

• This is called carry-look ahead adder

4-bit carry lookahead adder:-

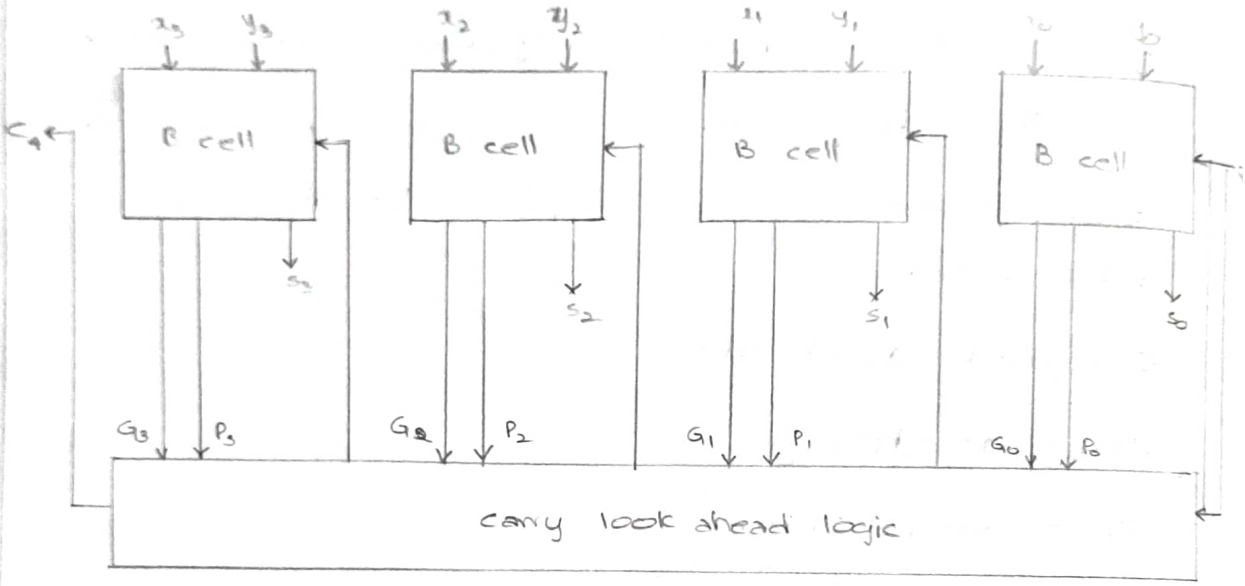
The carries can be implemented as

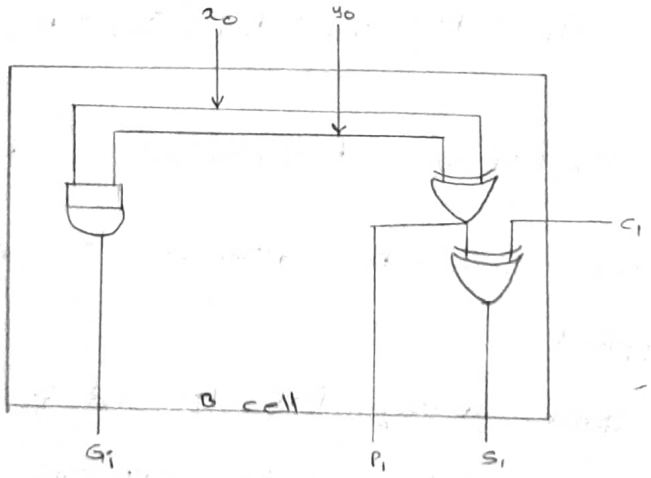
$C_1 = G_0 + P_0C_0$

$C_2 = G_1 + P_1C_1 + P_1P_0C_0$

$C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$

$C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$





Multiplication:-

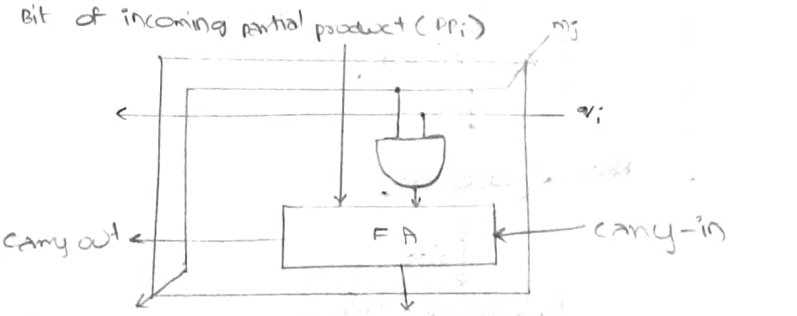
Multiplication of unsigned numbers:-

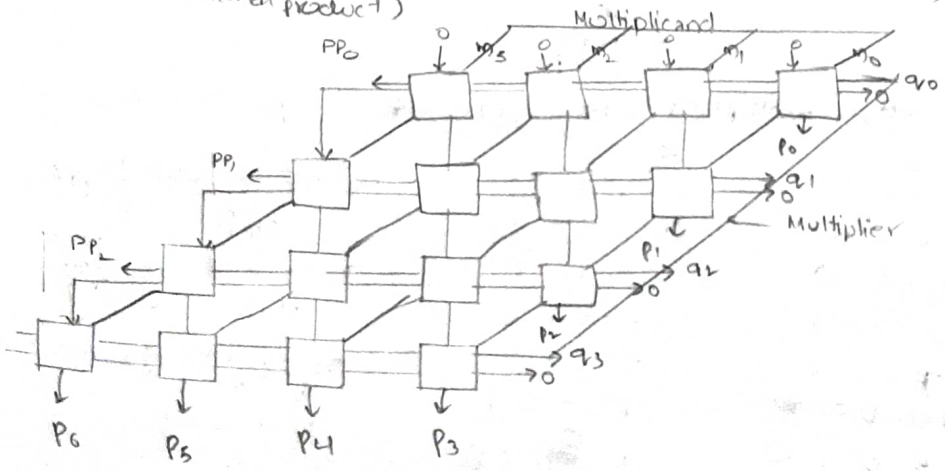
- We add the partial products at end
→ alternative would be to add the partial products at each stage

- Rules to implement multiplication are:-
→ If the ith bit of the multiplier is 1, shift the multiplicand and add the shifted multiplicand to the current value of the partial product.
→ Hand over the partial product to the next stage
→ Value of the partial product at the start stage is 0

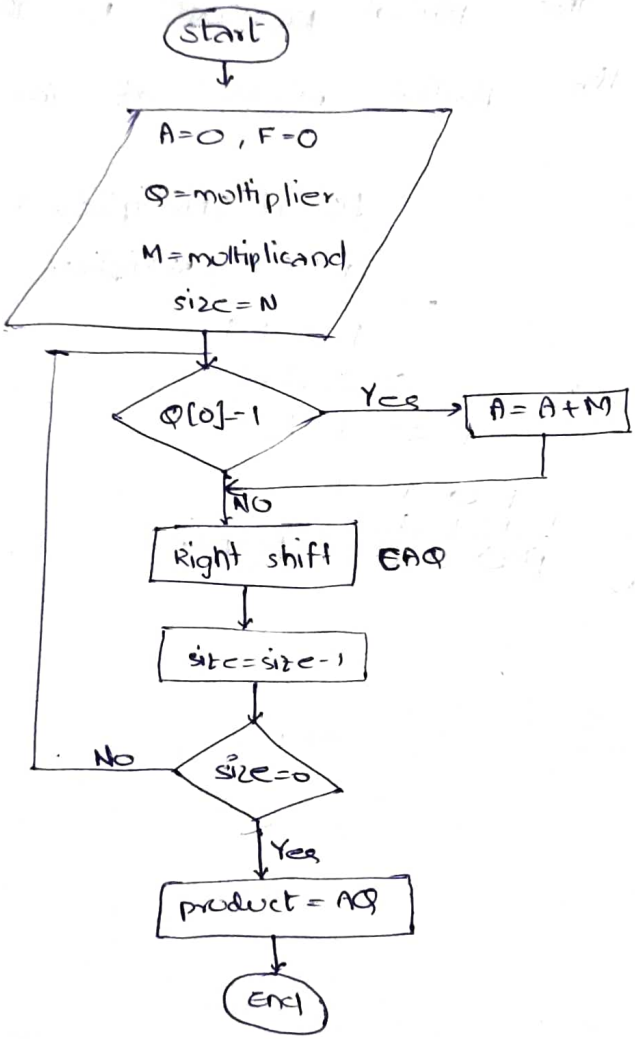
```
  1011   (Multiplicand M)
  1101   (Multiplier Q)
 ------
  1011
 0000
1011
1011
-------
1000 1111
```

combinatorial array multiplier:-





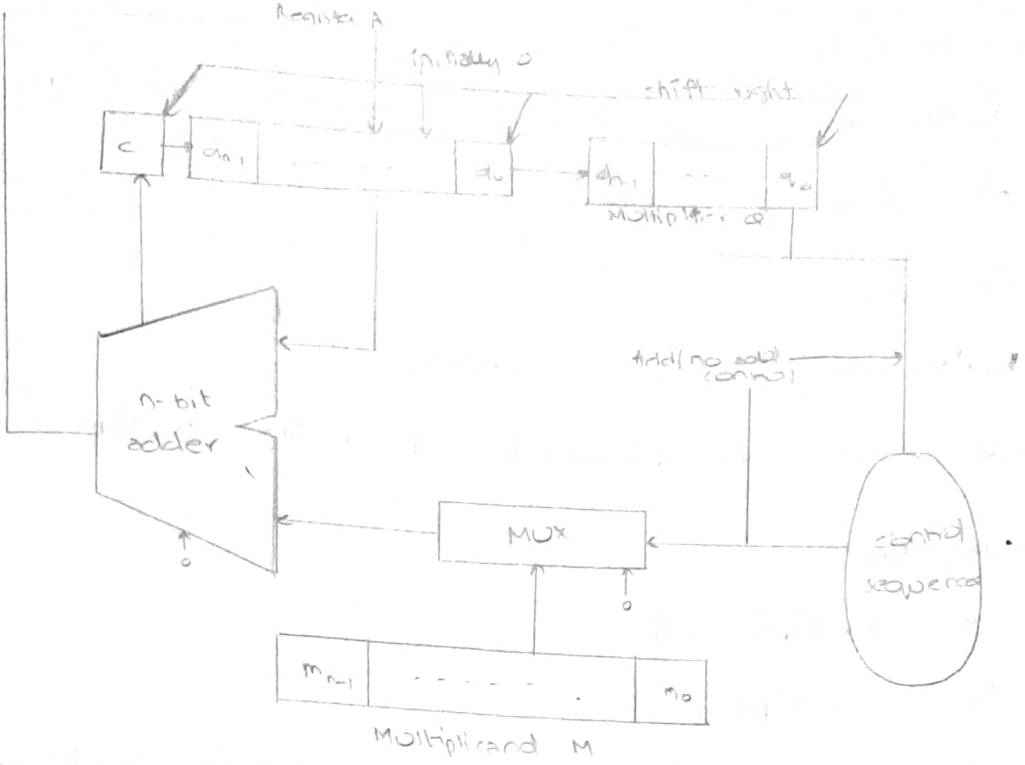
Sequential multiplication method:-

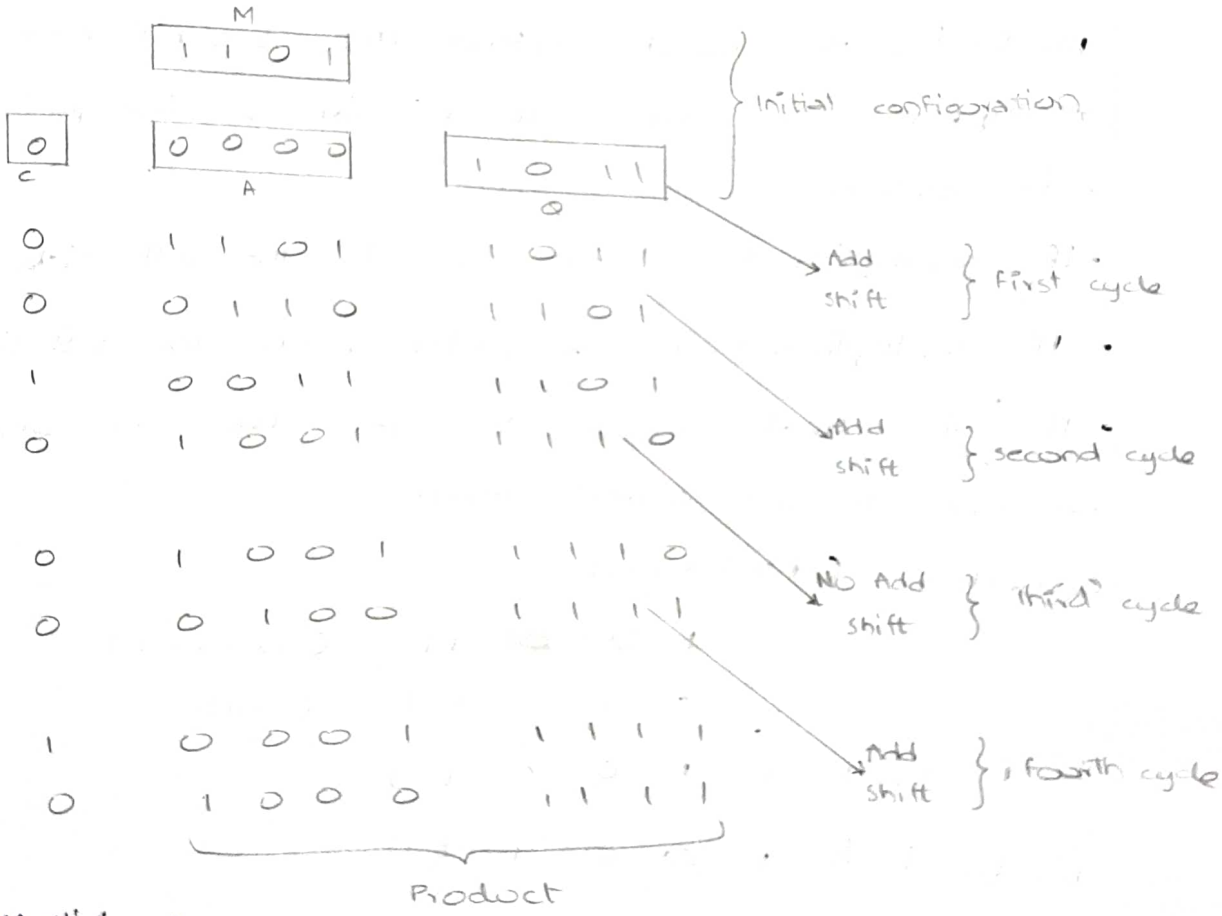


Multiplicand = +6, multiplier = 14, size = 4

C	A	Q	
0	0000	1110	q_0=0, no add } 1st
0	0000	0111	shift
0	0110	0111	q_0=1, add } 2nd
0	0011	0011	shift
0	1001	0011	q_0=1, add } 3rd
0	0100	1001	shift
0	0010	1100	q_0=0, add } 4th
			shift

Sequential circuit multiplier:-





Multiplicand = 15 Multiplier = 10

Signed Multiplication:-

case-(i):-

Both are non negative numbers (+ve) multiplication of unsigned numbers

case-(ii):-

Both are -ve numbers

Take 2's complement of both multiplier and multip

case-(iii):-

-ve multiplicand

+ve multiplier

- When we add a negative multiplicand to a partial product, we must extend the sign-bit value of the multiplicand to the left as far as the product will extend
- If multiplicand is +ve, '0' to the left side
- If multiplicand is -ve, add '1' to the left side
- If the final result is -ve take 2's complement of that to get correct result

consider (-13) x (+11)

```
                1 0 0 1 1   (-13 - 2's c)
              0 1 0 1 1     (+11)
            ---------
  1 1 1 1 1 1 0 0 1 1
  1 1 1 1 1 0 0 1 1
  0 0 0 0 0 0 0 0
  1 1 1 0 0 1 1
0 0 0 0 0 0
1 1 1 1 1 1 1 1
-------------------
0 0 0 1 1 1 0 0 0 1   (-143)
```

case-(iv):-

For a negative multiplier, a straightforward solution is to form the 2's complement of both multiplier and the multiplicand and proceed as in the case of a positive multiplier

- This is possible because complementation of both operands does not change the value or the sign of the product
- A technique that works equally well for both negative and positive multipliers - Booth algorithm

The Booth's algorithm:-

- It handles both the +ve and -ve numbers equally
- This algorithm generates a 2n bit product
- It treats both the +ve and -ve 2's complement operands uniformly

Multiplier Bit (i)	Bit (i-1)	version of multiplicand selected by bit i
0	0	0 × M
0	1	+1 × M
1	0	-1 × M
1	1	0 × M

(+13) × (-6) ⇒

0 1 1 0 1

0 -1 +1 -1 0

0 0 0 0 0 0 0 0 0 0

1 1 1 1 1 0 0 1 1

0 0 0 0 1 1 0 1

1 1 1 0 0 1 1

0 0 0 0 0 0 1

1 1 1 0 1 1 0 0 1 0 (-78)

0 0 1 1 0

2's 1 0 0 1

(-6) = 1 1 0 1 0 (2)

0 -1 1 -1 0

05-04-

- Best case - a long string of 1s
- Worst case - 0's and 1's are alternating

Worst: 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

+1 -1 +1 -1 +1 -1 +1 -1 +1 -1 +1 -1 +1 -1 +1 -1

Ordinary: 1 1 0 0 0 1 0 1 1 0 1 1 1 1 0 0

0 -1 0 0 +1 -1 +1 0 -1 +1 0 0 0 -1 0 0

Good: 0 0 0 0 1 1 1 1 1 0 0 0 0 -1 1 1

0 0 0 +1 0 0 0 0 0 -1 0 0 +1 0 0 -1

Fast multiplication:-

1. Bit-pair recoding of multipliers
2. Carry save Addition of Summands
3. Summand addition tree using 3-2 reducers.

Bit-pair recoding of multipliers:-

Multiplier bit-pair		Multiplier bit on right	Multiplicand
i+1	i	i-1	i
0	0	0	0×M
0	0	1	+1×M
0	1	0	+1×M
0	1	1	+2×M
1	0	0	-2×M
1	0	1	-1×M
1	1	0	-1×M
1	1	1	0×M

Ex:-

1 0 1

-1 1

2^1 2^0

$= -1 \times 2^1 + 1 \times 2^0 = -2+1$

$= -1$

0 0 1

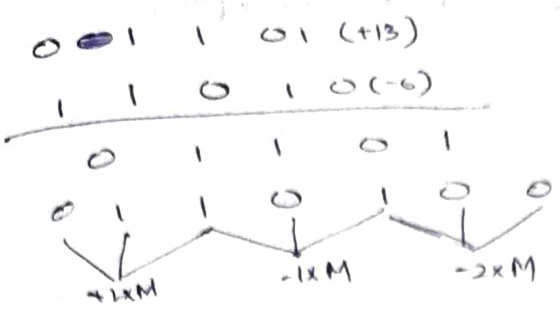
0 +1

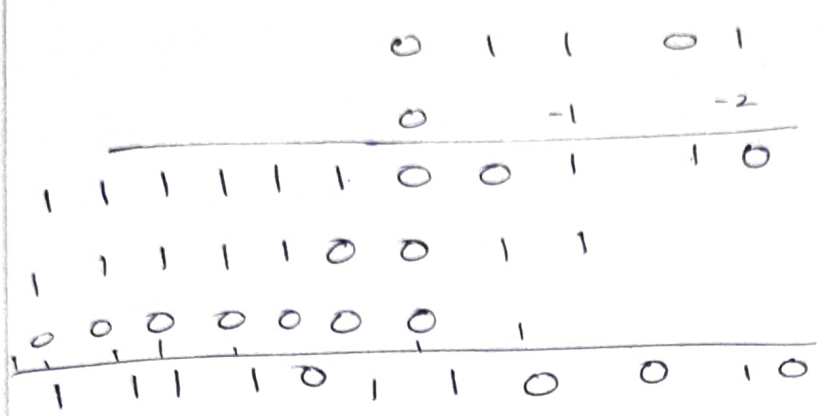
2^1 2^0

$= 2^1 \times 0 + 1 \times 2^0$

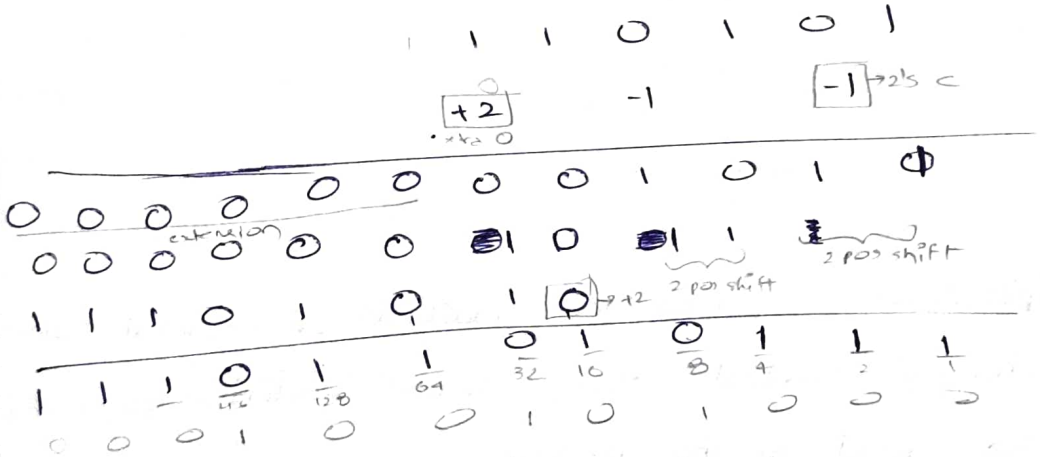
$= 0+1 = 1$

Ex: (+13) × (-6)





M = -11 Q = +27

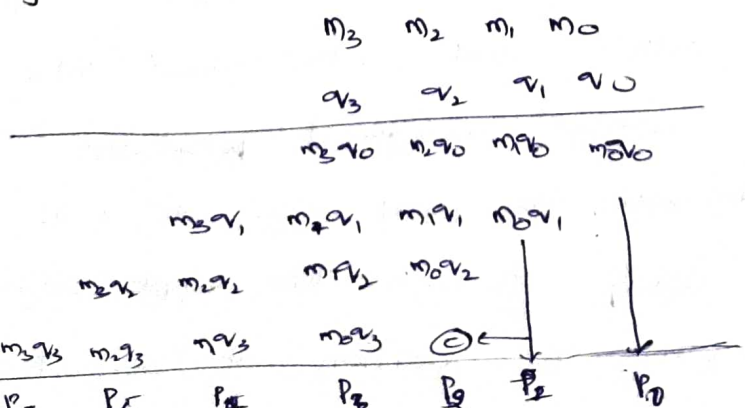


(-1) = 2's c
1 1 0 1 0 1
0 0 1 0 1 0
1

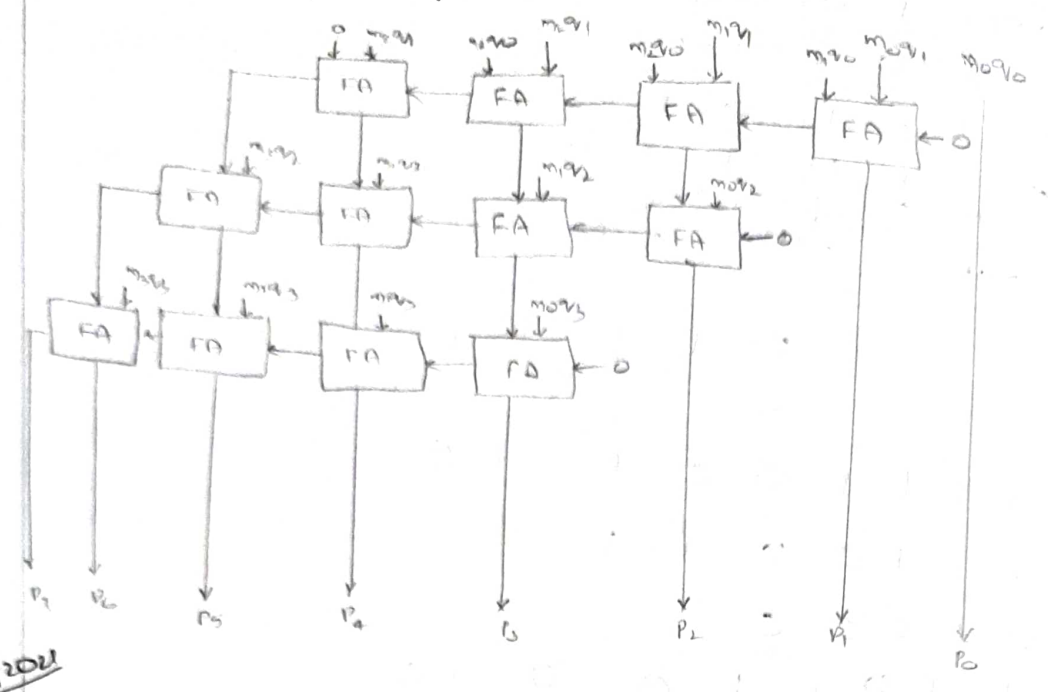
(+2) → extra 0 on right

256
32
8
2
2=7

Carry-Save addition of Summands :-

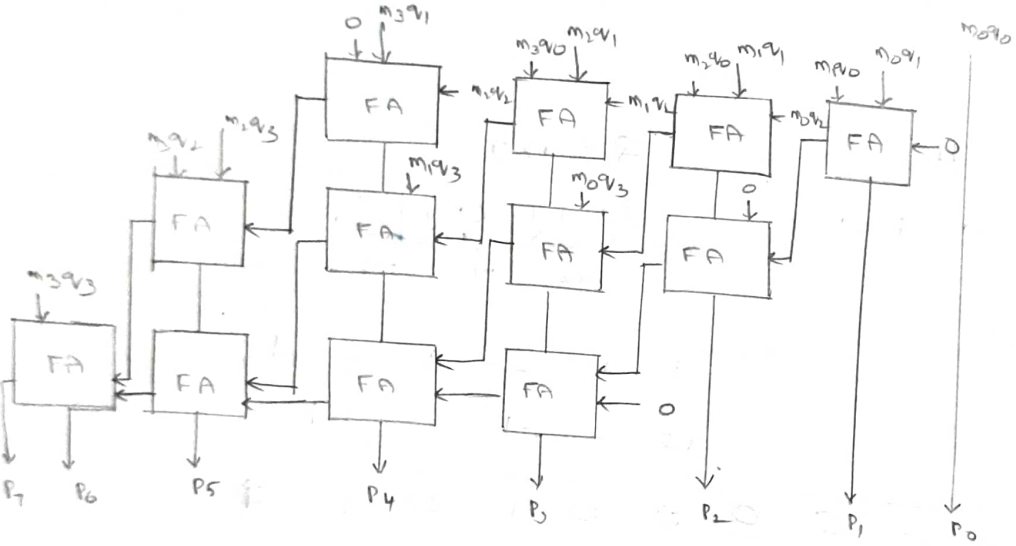


Ripple carry array for 4x4 multiplier:-



07/09/2021

Carry save array by 4x4 Multiplier:-



- Multiplication requires the addition of several summands
- A technique called carry save addition (CSA) can be used to speed up the process
- Instead of letting the carries ripple along the rows, they can be 'saved' and introduced into the next row at the correct weighted positions
- The delay through the carry-save array is somewhat less than the delay through the ripple carry array
- Multiplication requires the addition of several summands

→ Consider the addition of many commands, we can

• Group the commands in threes and perform carry-save addition on each of these groups is parallel to generate a set of S and C vectors in one full adder delay.

• Group all of the S and C vectors into threes, and perform carry-save addition on them, generating a further set of S and C vectors in one more full adder delay.

• Continue with this process until there are only two vectors remaining

• They can be added in a RCA or CLA to produce the desired product.

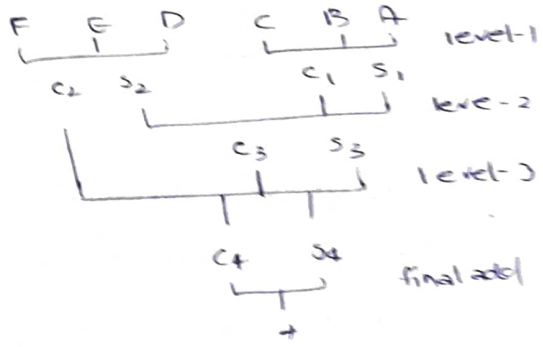
Command addition tree using 3-2 reducers:-

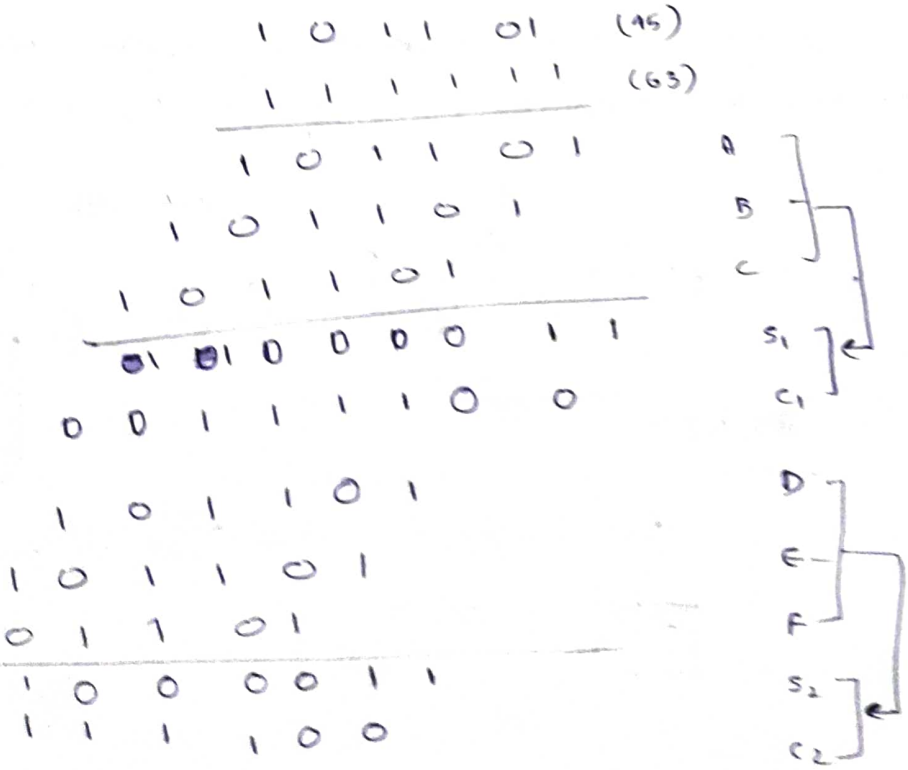
• Gate delay

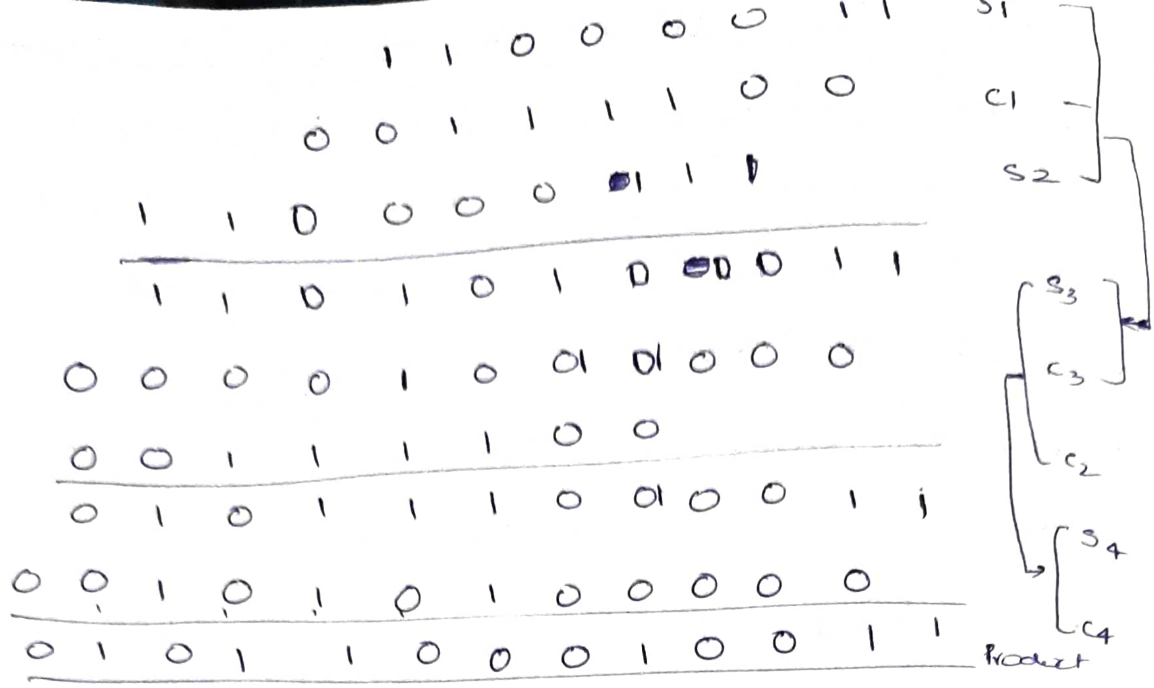
→ By Array multiplier:

$6(n-1)-1 = 29$

→ By CSA = 15 (14G+8)







Division:-

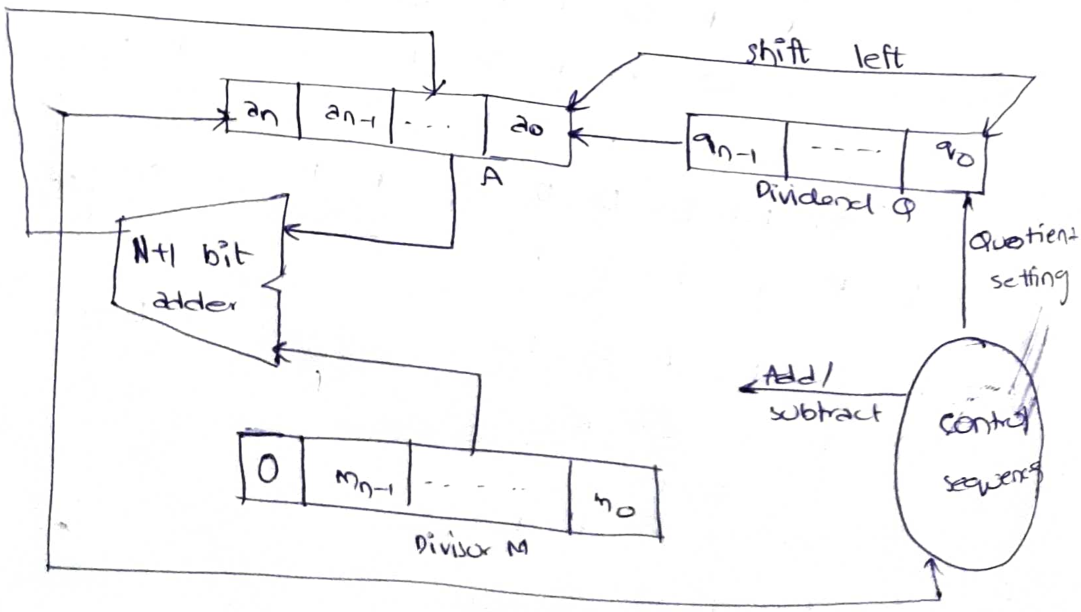
Manual division:-

```
13) 274 ( 21
    26
    ---
     14
     13
    ---
     (1)
```

```
                       dividend
                          ↑
1101) 1000 10010 ( 10101
  ↓    01101
divisor ------
       0010000
           1101
          ------
           1110
           1101
           -----
            (1)
```

Longhand division:-

- Position the divisor appropriately with respect to dividend and perform a subtraction.
- If the remainder is zero or positive, a quotient bit of 1 is determined, the remainder is extended by another bit



of the dividend, the divisor is repositioned, and another subtraction is performed

- If the remainder is negative, the quotient bit is 0 is determined, the dividend is restored by adding back the divisor, and the divisor is repositioned for another subtraction

Restoring division:-

- This method uses three n-bit registers A, M, Q for dividing two n-bit numbers.
- The register M is used to hold the divisor
- Initially A contains zero and M holds n-bit divisor

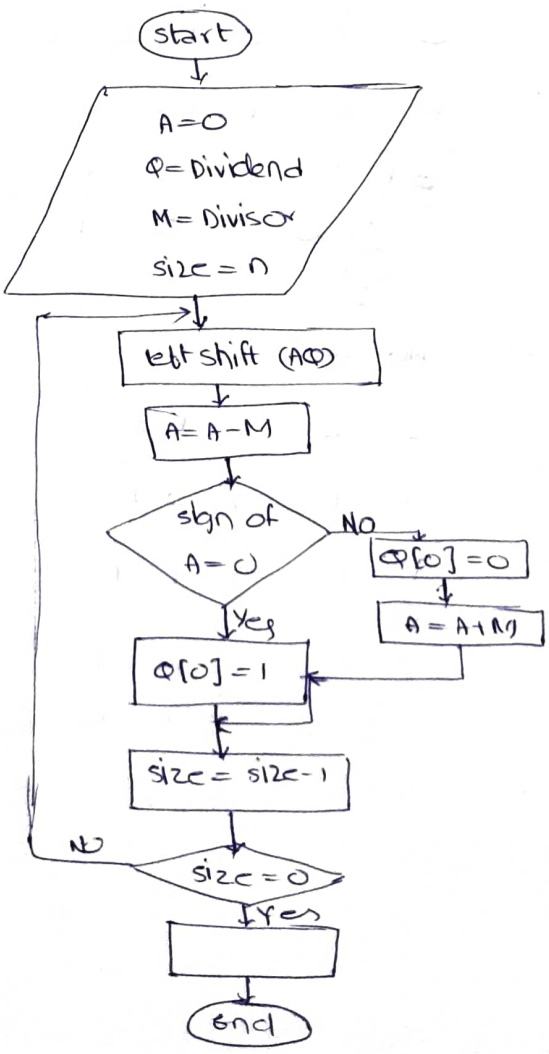
In each iteration

→ Shift A and Q left one binary position

→ Subtract M from A, and place the answer back in A

→ If the sign of A is 1, set q_0 to 0 and add M back to A (restore A); otherwise, set q_0 to 1

→ Repeat these steps n times



Example:

$$\begin{array}{r} 1000 \;(10 \\ 11 \\ \hline 10 \end{array}$$

A = Q

Initial

A: 0 0 0 0 0 — Q: 1 0 0 0

M: 0 0 0 1 1

shift : 0 0 0 0 1 — 0 0 0 □

subtract M: 2's c: 1 1 1 0 1

set q_0 : 1 1 1 1 0

-ve, set q_0 = 0

restore : 0 0 0 1 1 — 0 0 0 0

size = 3

0 0 0 0 1 — 0 0 0 0

shift : 0 0 0 1 0 — 0 0 0 0

subtract : 1 1 1 0 1

set q_0 : 1 1 1 1 1

-ve, set q_0 = 0

Q: 5

restore : 0 0 0 1 1 — 0 0 0 0

size = 4

0 0 0 1 0 — 0 0 0 0

shift : 0 0 1 0 0 — 0 0 0 0

subtract : 1 1 1 0 1

set q_0 : 0 0 0 0 1

q_0 = 1

~~restore~~ = No restoration

0 0 0 1

shift : 0 0 0 1 0 — 0 0 1 0

subtract : 1 1 1 0 1

set q_0 : 1 1 1 1 1

set q_0

restore : 0 0 0 1 1 — 0 0 1 0

size = 0

0 0 0 1 0

remainder — quotient

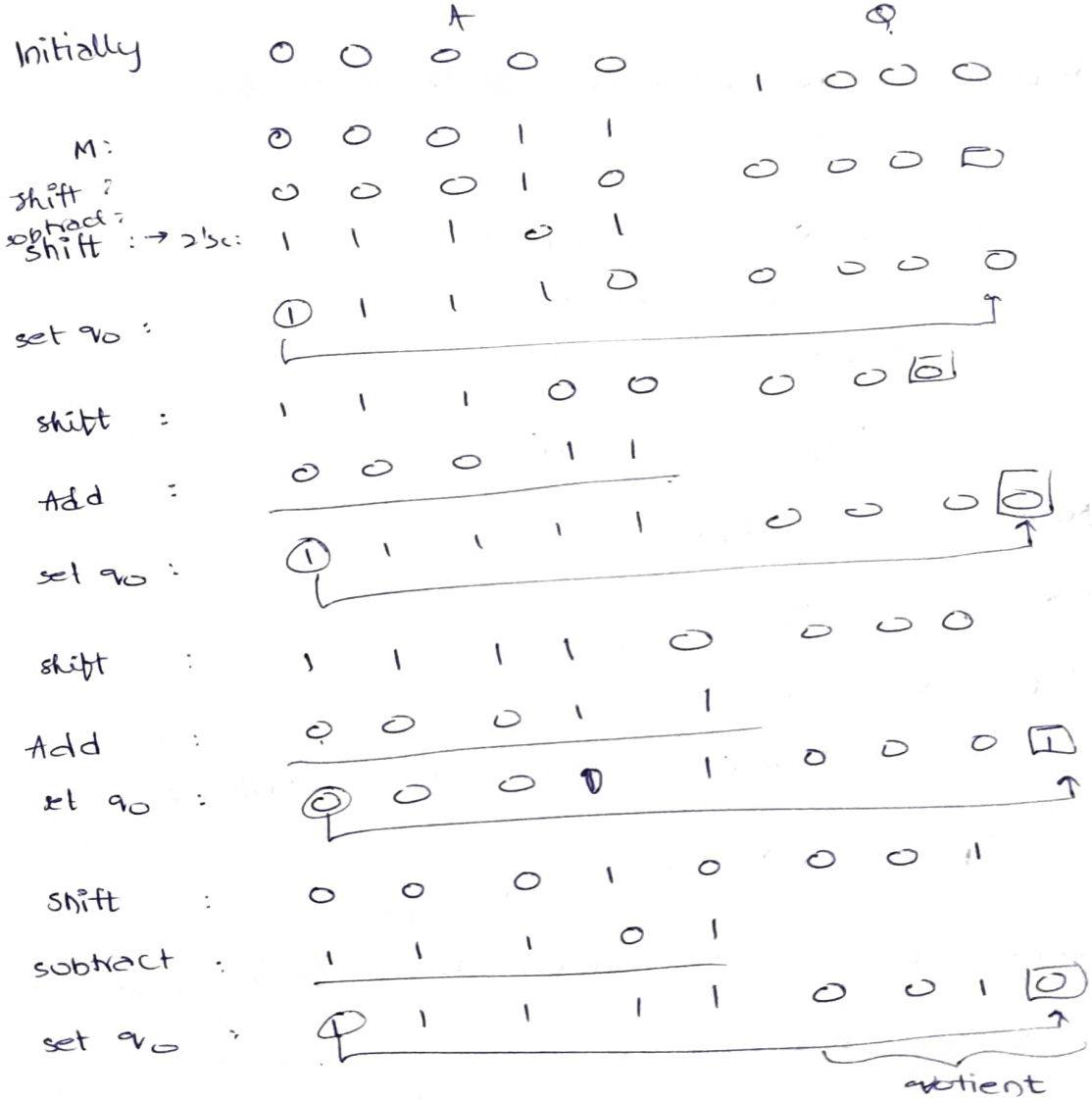
10/1/2021

Non restoring division:-

- Avoid the need for restoring A after an unsuccessful subtraction.
- step-1: (Repeat n times)

If the sign of A is 0, shift A and Q left one bit position and subtract M from A, otherwise shift A & Q and add M to A, If the sign of A is 0, set q_0 to 1, or 0

step-2: If sign of A is 1, add M to A



- Represent the binary positive number 1101011 in the IEEE single precision format

$1101011 = +1.101011 \times 2^6$

23-bit mantissa is

10101100000000000000000

Exponent = 6 = 0110

- Represent the decimal -0.75 into IEEE - single precision

$-0.75 \times 2 = 1.5 \rightarrow 1$

$0.5 \times 2 = 1.0 \rightarrow 1$

$0.0 \times 2 = 0.0 \rightarrow 0$

$0.0 \times 2 = 0.0 \rightarrow 0$

$(-0.75)_{10} = (-0.11)_2$

Normalize $= -1.1 \times 2^{-1}$

mantissa $= 10000\underbrace{\ldots}_{23 \text{ bits}}$

Exponent $= -1$

$e' = E + 127 = -1 + 127 = 126 = 01111110$

1	01111110	1000...

- Represent -9.5 in 64-bit IEEE floating point representation

Decimal Number = -9.50

$(9)_{10} = (1001)_2$

$(0.5) \times 2 = 1.0 \rightarrow 1$

$0.0 \times 2 = 0.0 \rightarrow 0$

$(-9.5)_{10} = (-1001.1)_2$

$M = -1.0011 \times 2^3$

$M = .0011000\ldots$

$E = E + (1023) = 3 + 1023 = 1026$, sign = 1

1	10000000010	0001000...

$1460.125 = (10110110100.001)_2$

Normalization:-

$1.0110110100001 \times 2^{10}$

Mantissa

sign = 0, $E = 10 = 127 + 10 = 137 = 10001001$

0	10001001	01101101000001

11/04/20

Floating point arithmetic : ADD / SUB rule :-

- choose the number with the smaller exponent
- shift its mantissa right until the exponents of both the numbers are equal
- Add or subtract the mantissas
- Determine the sign of the result
- Normalize the result if unecessary and truncate/ round to the number of mantissa bits.

Mul rule:-

- Add the exponents
- Subtract the bias
- Multiply the mantissas and determine the sign of the result.
- Normalize the result
- Truncate/ round the mantissa of the result

DIV rule:-

- Subtract the exponents
- Add the bias
- Divide the mantissas and determine the sign of result

- Normalize the result (if necessary)
- Truncate / round the mantissa of the result

Special values:-

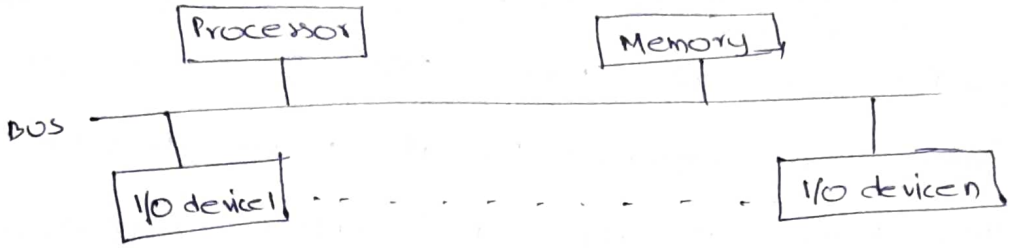
- The end values 0 and 255 of excess -127 exponent e' are used to represent special values
- when e' is 0 and $M=0$, the exact 0 value is represented
- when $e'=255$ and $M=0$, the value ∞ is represent
- when $e'=0$ & $M \neq 0$ denormal numbers are represented their value is $\pm 0.M \times 2^{-126}$

01/20

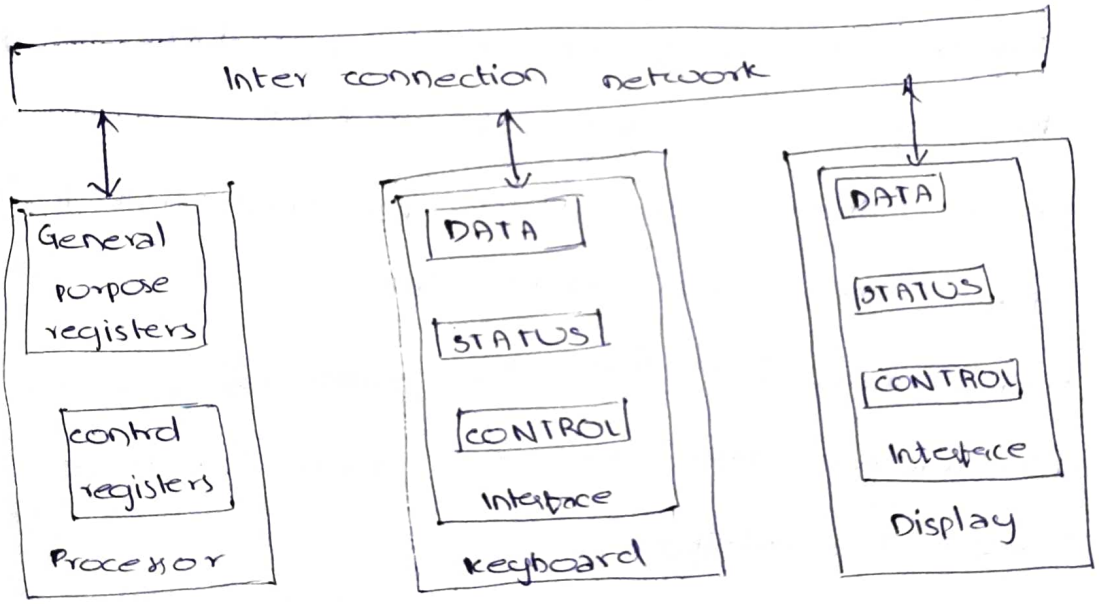
Input / output organization:-

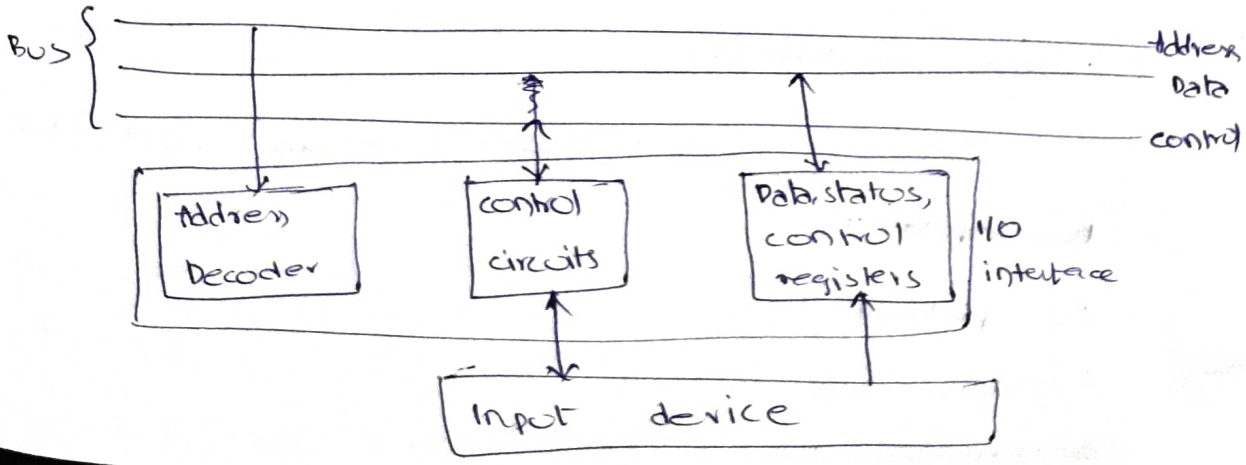
Bus structure:-

- The interconnection network is used to transfer the data among the processor, memory and I/O devices
- The commonly used interconnection network is called bus.



- Only one source / destination pair of units can use this bus to transfer data at any one time
- Bus consits of three sets of lines used to carry address, data, control signals





- Processor, main memory, I/O devices are interconnected by means of a bus
- Each I/O device is assigned a unique set of addresses for the registers in its interface
- When a processor places a particular address on the address lines, it is examined by the address decoders of all devices on the bus
- Address is observed by all device decoders.
- The device recognizes the address, responds to the commands issued on the control lines
- The processor uses the control lines to request either a Read/Write operation, requested data is transferred over data lines

Memory mapped I/O:-

- It is a way to exchange data and instructions between CPU & I/O attached to it
- It is the one where, processor and I/O devices share the same address space.
- Any instruction that can access memory can be used to transfer data to and from I/O device
- Load R2, DATAIN

  DATAIN is address of a register in an input device
- Store R2, DATAOUT

  DATAOUT is a register in an output device

Bus operation:-

Bus protocol

Synchronous bus

Asynchronous bus

Bus protocol:

- The set of rules that govern the behaviour of various devices connected to the bus, as to when to place information on the bus, when to assert control signals et
- Theser rules are implemented by control signals that indicate what and when actions are to be taken
- one control line, usually labelled R/W specifies whether a Read or write operation is to be performed

Read - 1, write - 0

- control signals specify:

→ whether it is a read or a write operation

→ when several operand sizes (byte, word, long) required, size is indicated by other control lines

→ Timing information to indicate when the processor and I/O devices may place data or recieve data from the bus.

- Two types of data transfer depending on the mechanism of timing data

→ Synchronous

→ Asynchronous

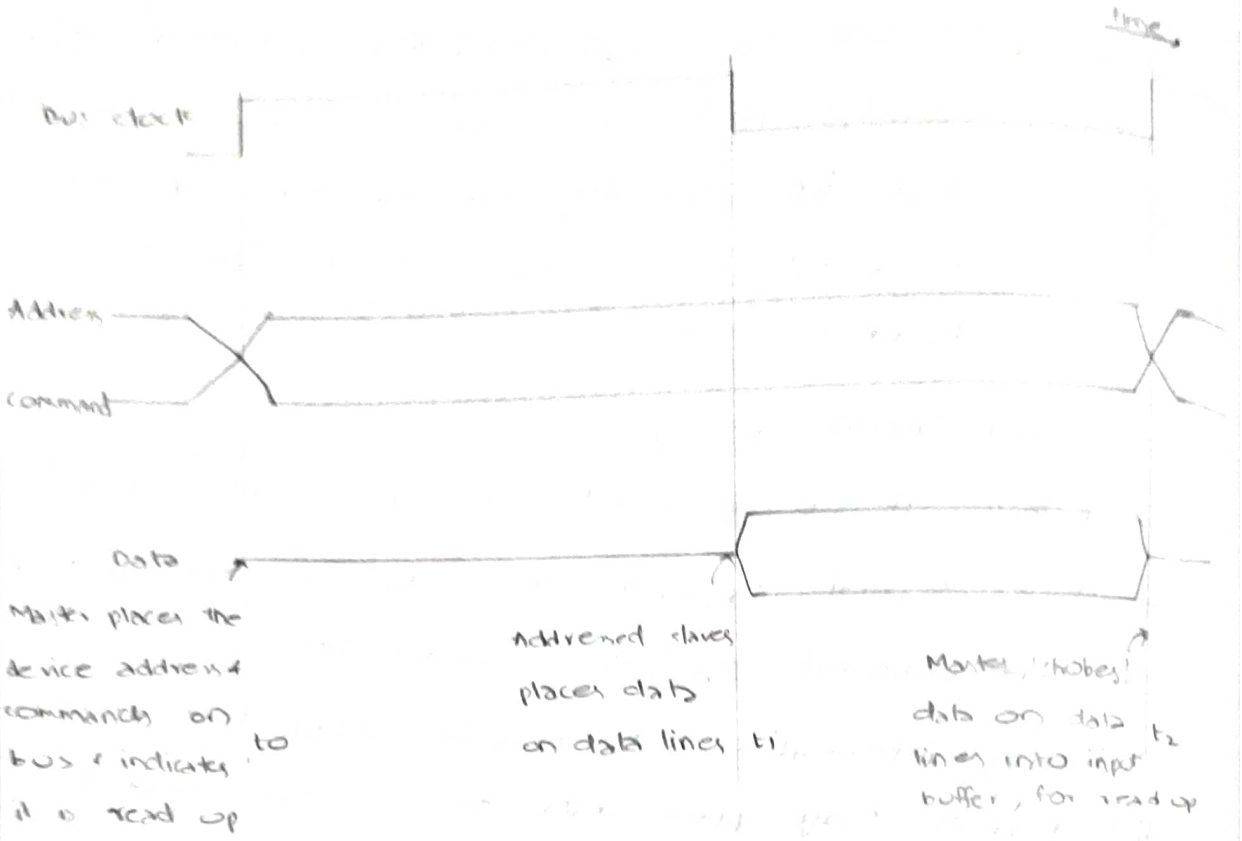
- In any data transfer one device plays the role of a master
- This is a device that indicates initiates data transfer by issuing read/ write command on the bus
- Normally CPU behaves like a master

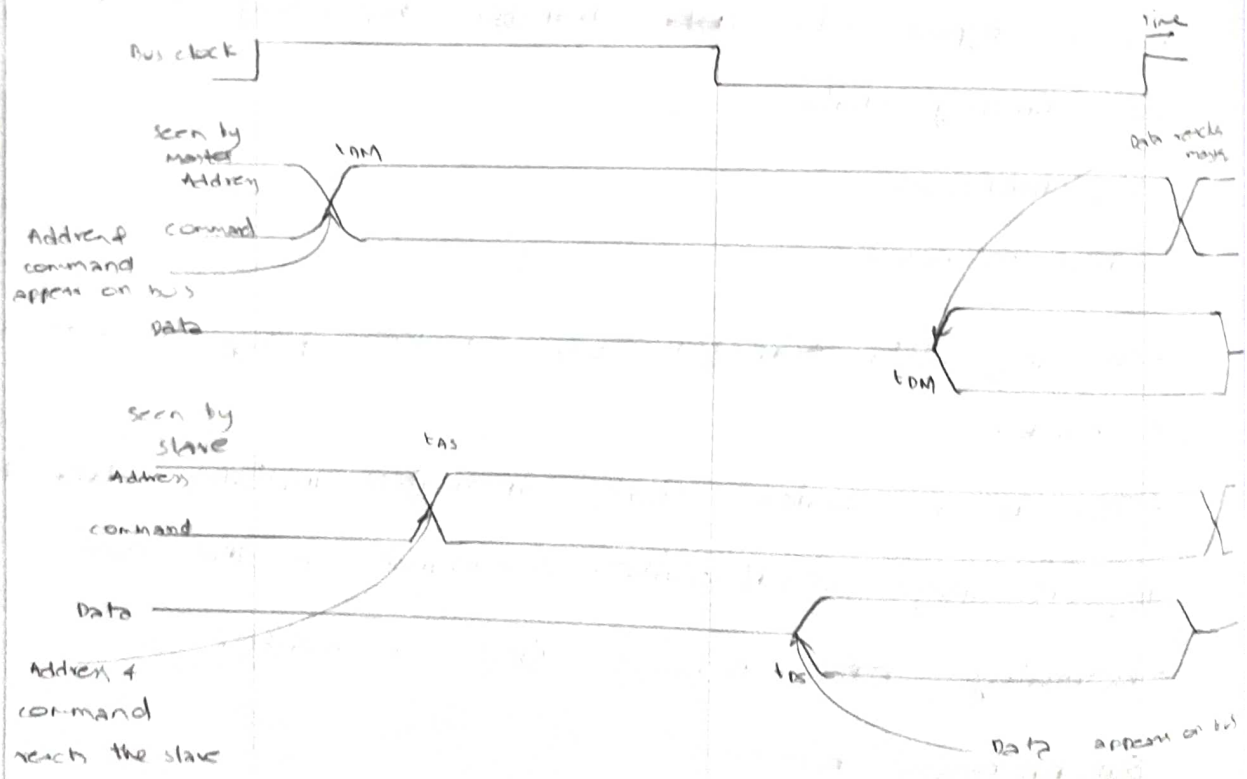
Synchronous bus:-

- All devices derive timing information from a common clock line - Bus clock
- The signal on this line has two phases

A high level followed by low level

- The clock line has equally spaced pulses which define equal time intervals (bus cycles)
- One data transfer can take place during one bus cycle





- Once the master places the device address and command on the bus, it takes time for this information to propagate to the devices

• Also all the devices have to be given enough time to decode the address and control signals, so that addressed slave can place data on bus

• Width of the pulse t_1 to depends on

→ Maximum propagation delay between two devices connected to bus

→ Time taken by all the devices to decode the address and control signals, so that the addressed slave can respond at t_1

• At the end of the clock cycle, at time t_2, the master strobes the data on data lines into its buffer if its a read operation (strobe means capture the values and store in buffer)

• When data are to be loaded into a storage buffer register, the data should be available for a period longer than the setup time of the device.

• Width of pulse $t_2 - t_1$ should be longer than:

→ Max propagation time of bus

→ set up time of input buffer register of the master.

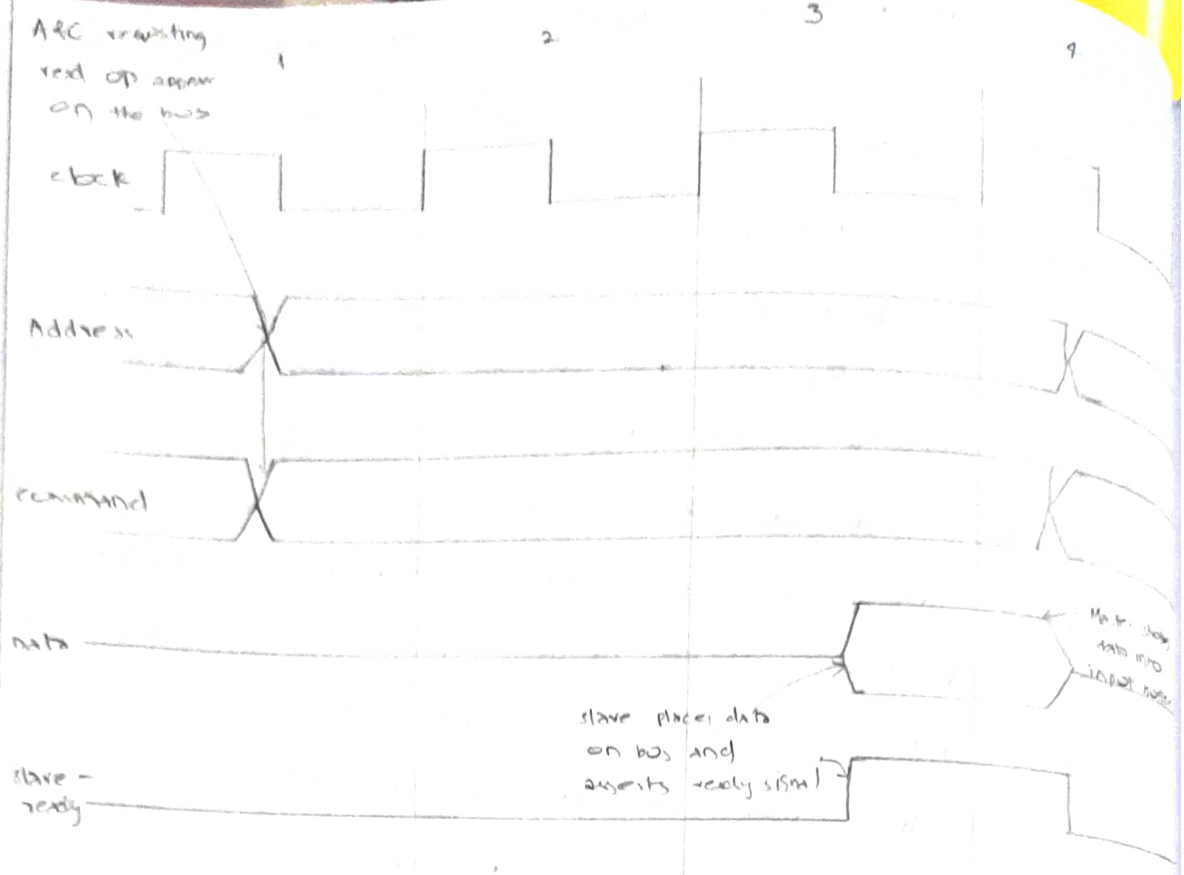
Limitations:-

• Data transfer has to be completed within one clock cycle

→ clock period $t_2 - t_0$ must be such that the longest propagation delay on the bus and the slowest device interface must be accomodated

→ forces all the devices to operate at the speed of slowest device.

• Processor just assumes that the data are available at t_2 in case of a Read operation, or are read by the device in case of a write operation.



11/04/2021

Asynchronous bus:-

- Data transfers on the bus is controlled by a handshake between the master and the slave
- common clock in the synchronous bus case is replaced by two timing control lines.

Master ready, slave ready.

- Master-ready signal is asserted by the master to indicate to the slave that it is ready to participate a data transfer.
- slave-ready signal is asserted by the slave in response to the master-ready from the master, and it indicates to the master that the slave is ready to transfer the data.

* Data transfer using the handshake protocol:-

→ Master places the address & command information on bus

→ Asserts the Master-ready signal to indicate to the slaves that

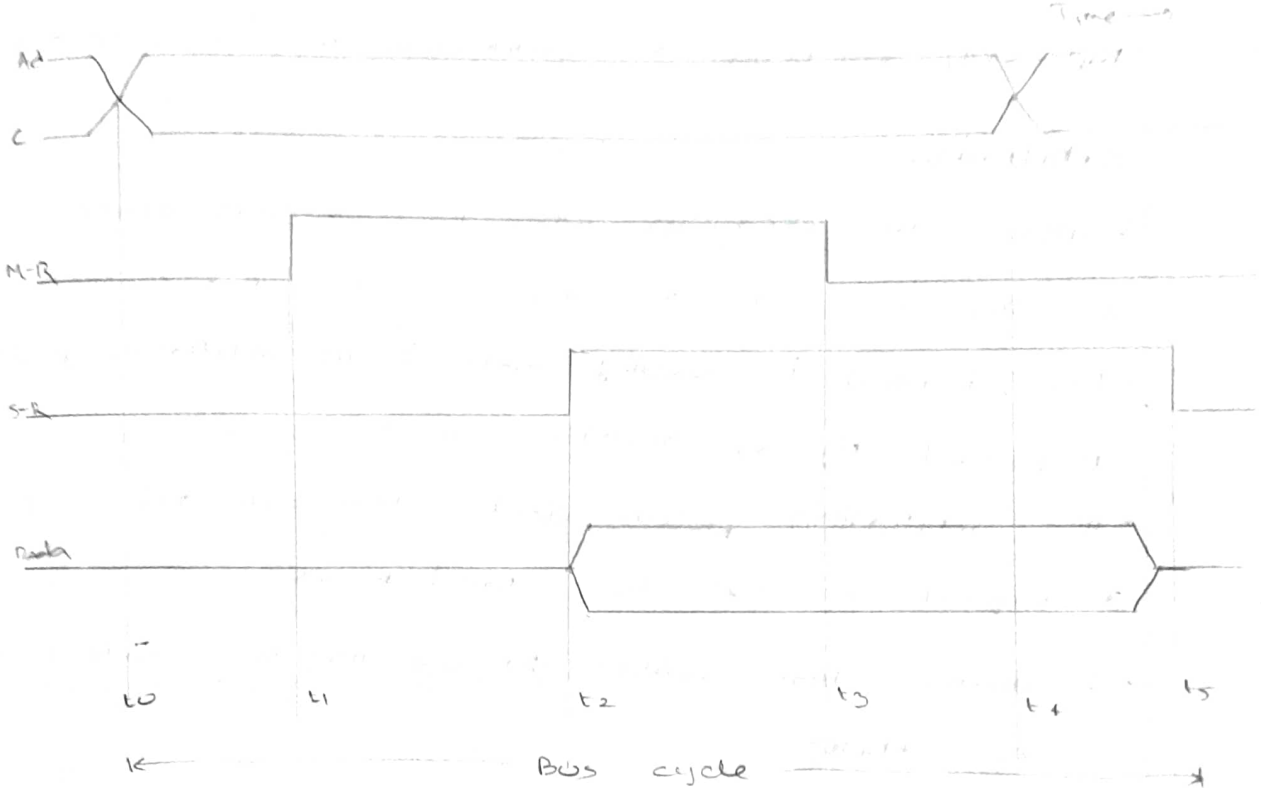
the address and command information has been placed on bus

→ All devices on the bus decode the address

→ Addressed slave performs the required operation and informs the processor it has done so by asserting the slave-ready signal

→ Master removes all the signal from the bus, once slave-ready is asserted

→ If the operation is read, master also strobes the data into the input buffer.



At t_0 - master places the address and command info on bus

t_1 - master asserts the M-r signal. M-r is asserted at t_1 instead of t_0

t_2 - Addressed slave places the data on bus and asserts s-r signal

t_3 - s-r signal arrives at the master

t_4 = Master removes the address & command info

t_5 - Slaves recieves the transition

Advantages:-

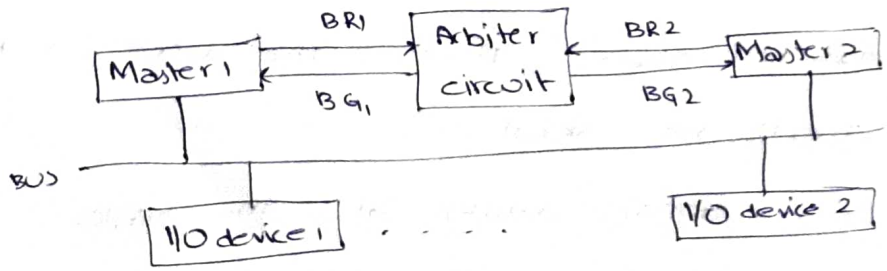
- Eliminates the need for synchronization between the sender and the reciever
- can accomodate varying delays automatically, using slave ready signal

Disadvantages:-

- Data transfer rate with full handshake is limited by two round trip delays
- Data transfers using a synchronous bus involves only one round trip delay, and hence a synchronous bus can achieve faster rate

Arbitration:-

- There are occasions when two or more entities contend for the use of a single resource in a computer system.
- This decision is usually made in an arbitration process performed by an abitration circuit
- The arbitration process starts when each device sending a request to use the shared resource.
- A device that wishes to use the bus sends a request to the arbiter
- when multiple requests arrive at the same time, the arbiter selects one request and grants the bus to the corresponding device.



Granting use of bus based on priorities

Interface circuits:-

- I/O interface circuits consist of the circuitry required to connect an I/O device to a computer bus
- Side of the interface which connects to the computer has bus signals for:

Address, Data, Control.

- Side of the interface which connects to the I/O device has:

→ Datapath and associated controls to transfer data between the interface and the I/O device.

→ This side is called as a "port"

- Ports can be classified into two:

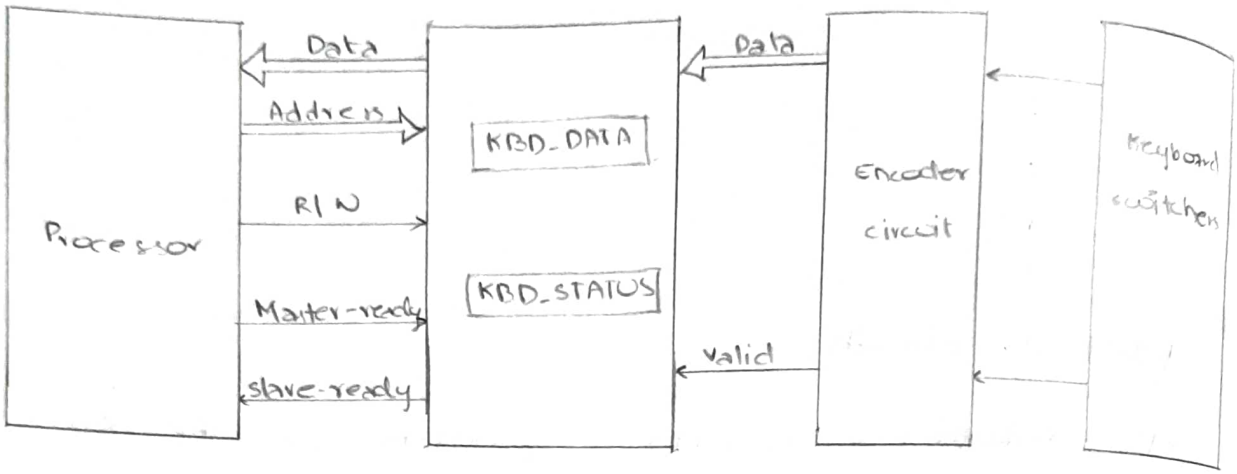
→ Parallel port

→ Serial port

- Parallel port transfers data in the form of number of bits, normally 8 or 16 to or from the device
- Serial port transfers and receives data one bit at a time

• Processor communicates with the I/O in the same way, whether it is parallel port or a serial port

→ conversion from the parallel to serial and vice versa takes place inside the interface circuit

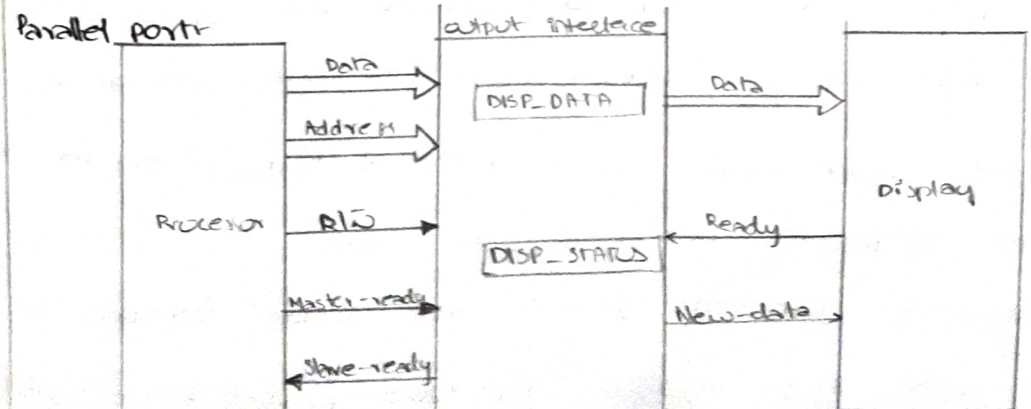


20/04/2021

Functions of the I/O interface:-

1. Provides a register for temporary storage of data.
2. Includes a status register containing status information that can be accessed by the processor
3. Includes a control register that holds the information governing the behaviour of the interface
4. Contains address-decoding circuitry to determine when it is being addressed by the processor
5. Generates the required timing signals.
6. Performs any format conversion that may be necessary to transfer data between the processor and the I/O device, such as parallel-to-serial conversion in the case of serial port.

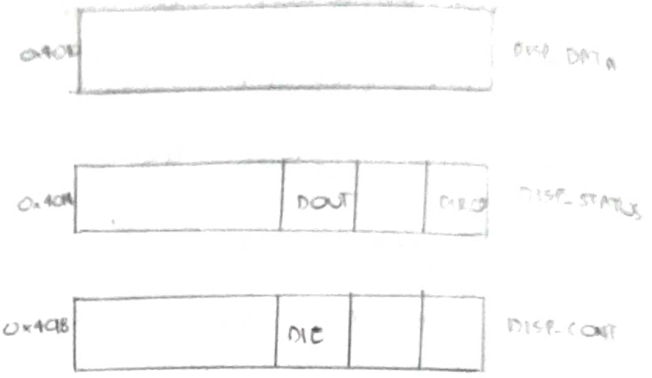
- Keyboard is connected to a processor using a parallel port.
- Processor is 32-bits and uses memory mapped I/O and the asynchronous bus protocol
- On the processor side of the interface :-
  - Datalines
  - Address lines
  - control or R/W line
  - Master-ready signal
  - slave ready signal
- On the keyboard side of the interface:
  - encoder circuit which generates a code for key pressed
  - Datalines contain the code for the key
  - Valid line changes from 0 to 1 when the key is pressed
  - This causes the code to be loaded into DATAIN and SIN to be set to 1
- circuit can be used to connect a keyboard to a processor
- There are only two registers:
  - A data register (KBD_DATA)
  - status register (KBD_STATUS)
- The latter contains the keyboard status flag, KIN
- When key pressed (KIN-1, KBD_DATA - Data)
- Output of encoder : one byte of data representing encoded characters and one valid bit



- On the printer side:
  - Hle signal line which the printer asserts when it is ready to accept the character
  - This causes the SOUT flag to be set to 1
  - Processor places a new character into a DATAOUT register
  - Valid signal, asserted by the interface circuit when it places a new character on data lines

22/07/2021

- On the processor side
  - Data lines
  - Address lines
  - control or R/W lines
  - Master-ready signal
  - slave-ready signal

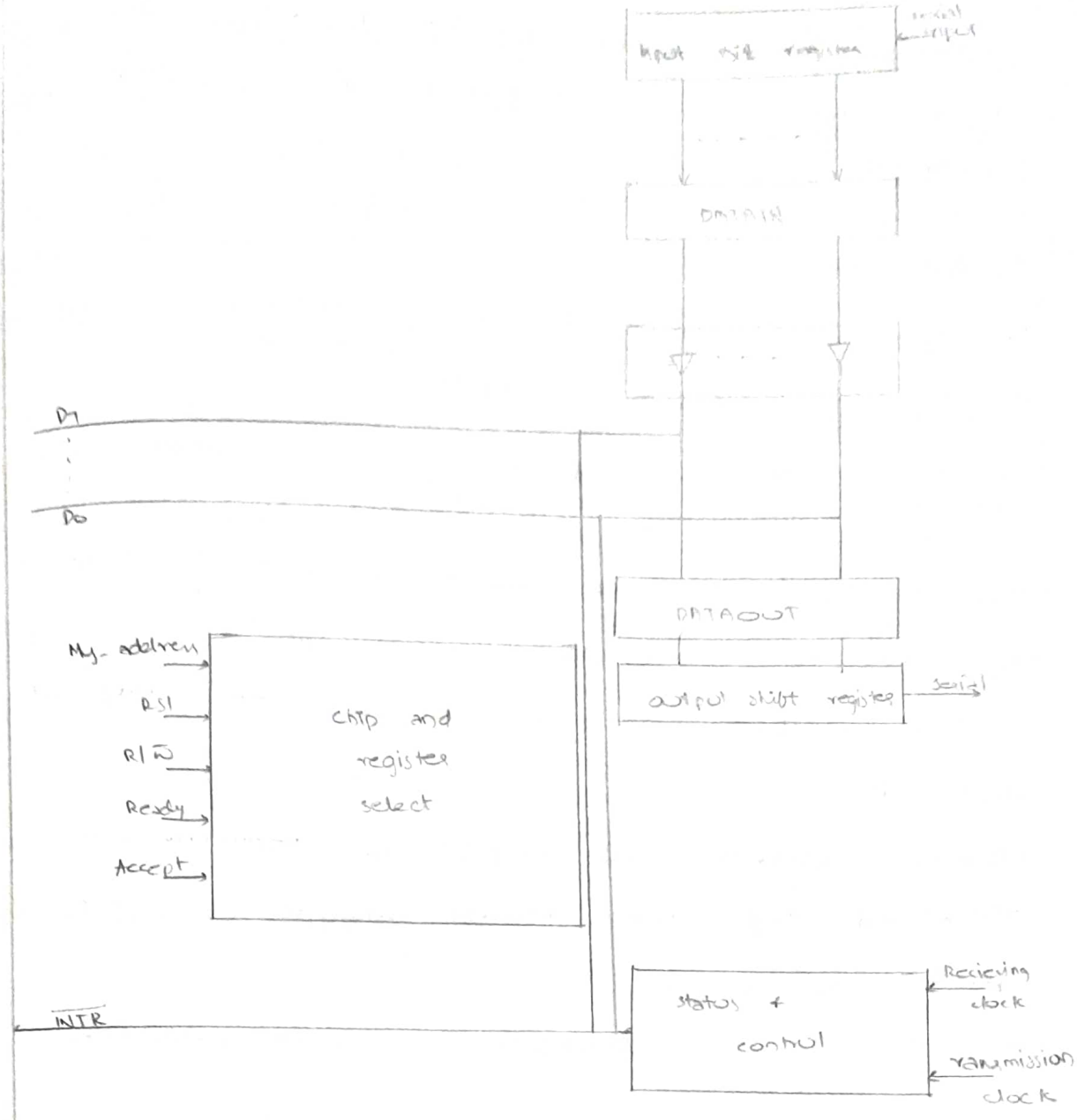


- When the I/O routine checks DOUT and finds out it equal to 1, it sends a character to DISP_DATA
- This clears the DOUT flag to 0 and sets New-data signal to 1
- In response, the display returns ready to 0 accepts and displays the character in DISP_DATA
- When it is ready to receive another character it sets ready again and the cycle repeats

Serial port:-

- Serial port is used to connect the processor to I/O devices that require transmission of data one bit at a time
- Serial port communicates in a bit-serial fashion on the device side and bit parallel on the bus side.

→ Transformation between parallel and serial formats is achieved with shift registers that have parallel access capability.



- Input shift register accepts input one at a time from the I/O device
- Once all the 8 bits are recieved, the contents of the input shift register are loaded in parallel into DATAIN register
- Output data in the DATAOUT register are loaded into the output shift register
- Bits are shifted out of the output shift register and sentout to the I/O device bit at a time.
- As soon as data from the input shift are loaded into DATAIN, it can start accepting another 8 bits of data.
- Input shift register and DATAIN registers are both used at input so that the input shift register can start recieving

another set of 8 bits from the input device after loading the contents to DATAIN, before the processor reads the contents of DATAIN. This is called as double buffering.

23/04/2021

- Serial interface require fewer wires and hence serial transmission is convenient for connecting devices that are physically distant from the computer.
- Speed of transmission of the data over a serial interface is known as the "bit rate".
→ Bitrate depends on the nature of the devices connected
- In order to accomodate devices with a range of speeds, a serial interface must be able to use a range of clock speeds.
- Several standard serial interfaces have been developed
→ Universal Asynchronous Receiver Transmitter (UART) for low-speed serial devices
→ RS-232-C for connection to communication links

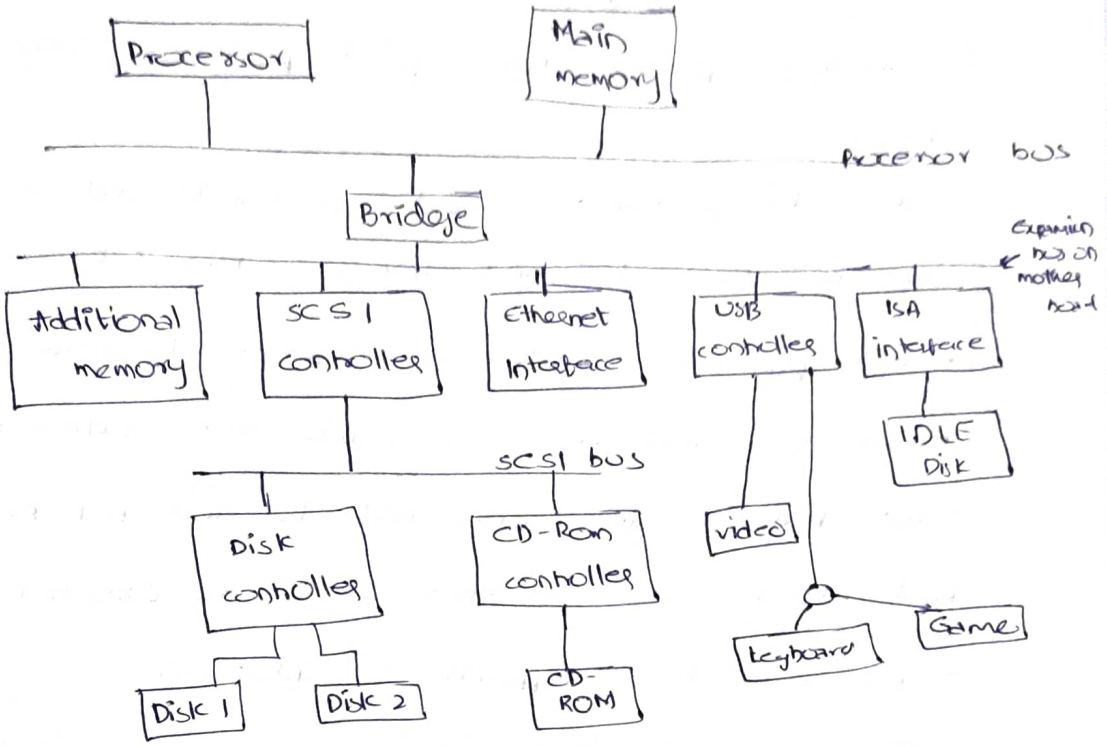
PCI Bus:-

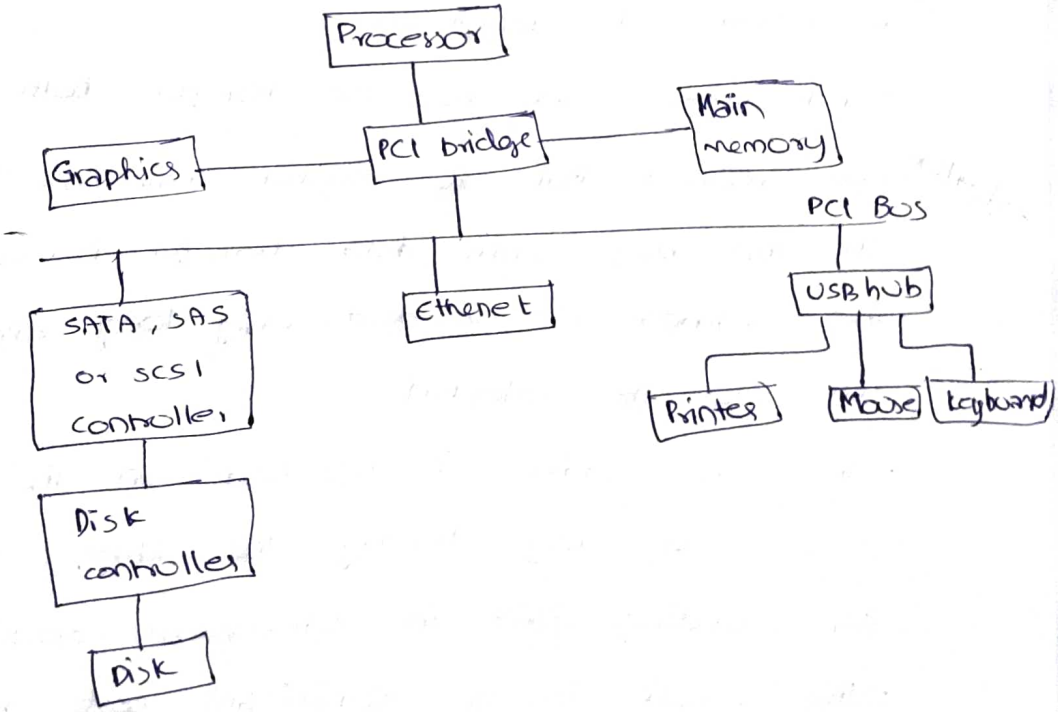
- A no. of standards have been developed for the expansion bus
→ some have evolved by default
→ IBM's Industry standard Architecture
- Three widely used bus standards:-
→ PCI (Peripheral component Interconnect)
→ SCSI (small computer system interface)
→ USB (Universal serial Bus)

PCI :-

- Peripheral component interconnect
- Introduced in 1992

- Low-cost and Processor independent bus
- It is housed on the motherboard of a computer and used to connect I/O interfaces for a wide variety of devices
- Plug and Play capability





Bus structures:-

- The PCI bus is connected to the processor bus via a controller called a bridge
- The bridge has a serial port for connecting the computer memory
- Another special high speed port for connecting graphic devices
- The bridge translates and commands and responses from one bus to other and transfers data between them
- The processor sends a request to an I/O device, the bridge forwards the commands and address lines on bus
- I/O devices are connected to the PCI bus through port that use standards such as Ethernet, USB, SATA, SCSI
- PCI bus has supports 3 interfaces
- R/W operation that involves a single word is treated as burst of length one.
- PCI uses same bus to transfer both address and data

28/01/20

- We assumed that the master maintains the address info on the bus until data transfer is completed. But, the address is needed only long enough for the slave to be selected
- Thus, the address is needed on the bus for one clock cycle only, freeing the address lines to be used for sending data in subsequent clock cycles
- The result is a significant cost reduction

Data transfer:-

- Bus master, which is the device that initiates data transfer by issuing read and write commands, is called initiator in PCI terminology

. the addressed device that responds to these commands is called a target

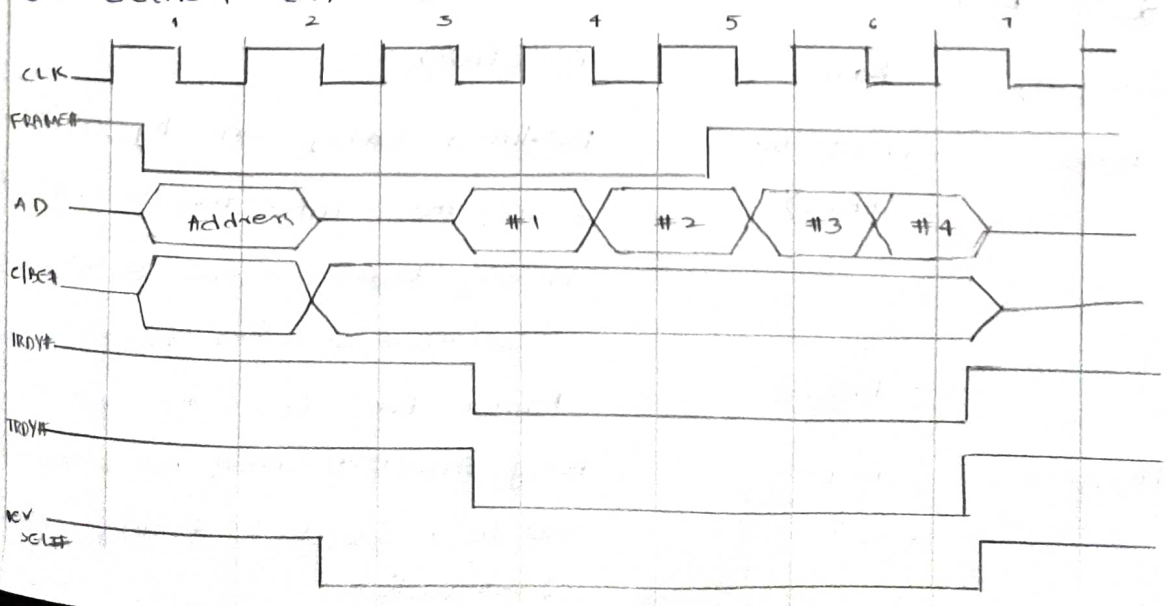
. the target ready, TRDY # = slave-ready

. Initiator ready signal, IRDY #, to support burst transfers

signals on PCI bus:-

Name	Function
CLK	A 33 MHz or 66 MHz clock
FRAME#	sent by initiator to indicate duration of a transaction
AD	32 address/data lines, which may be optionally increased to 64
C/BE #	Command/byte-enable lines
IRDY#, TRDY#	Initiator-ready, target-ready signals
DEVSEL#	A respons from the device indicating that it has recognized its address and is ready for a data transfer transaction
ID SEL#	Initialization Device select

. A complete transfer operation on the PCI BUS involving an address and a burst of data is called transaction.



- Clock cycle #1: Bus master asserts frame# to indicate the beginning of a transaction
- It sends address on address lines and read command on C/BE # lines
- Clock cycle #2: The initiator removes the address and disconnects its drivers from AD lines and asserts IRDY #
- IRDY # - it is ready to recieve data
- Target asserts DEVSEL# signal
- Clock cycle #3:

Target asserts TRDY # and begins to send the data

- It maintains DEVSEL # is asserted state until the end of the transaction.
- Initiator uses frame # to indicate duration of the burst
- This deactivates at cycle #5, during this the initiator recieves 3rd word
- In respons to that it recieves next word and stops
- After sending 4th word decativates TRDY # and DEVSEL# and disconnects its drivers on the AD lines

SCSI Bus:-

Category	Name	Function
Data	- DB(0) to DB(7)	Datalines carry one by of info during the info transfer phase & identify device during arbitration selection & reselection phases
	- DB(P)	Parity bit for the data bus
Phase	- BSY	Busy asserted when bus is not free
	- SEL	selection asserted during selection & reselection

Info type	- C/D	control / Data asserted during transfer of control info (command status or message)
	- MSG	msg indicates the info being transferred is a message
Handshake	- REQ	request asserted by a target or request data transfer cycle
	- ACK	Acknowledge: asserted by the initiator when it has completed ad transfer operation
Direction of transfer	- I/O	Input/Output asserted to indicate input operation
Other	- ATN	Attention: Asserted by an initiator when it wishes to send a message
	- RST	reset: causes a device controls to disconnect from the bus and assume the start-update