# MODULE 3 (University questions)

**1.**

**List the limitations of repeated addition method for multiplication.**

a. Least sophisticated method
b. Just use adder over and over again
c. If the multiplier is n bits, can have as many as
d. 2n iterations of addition -- O(2n)
e. Not used in an ALU

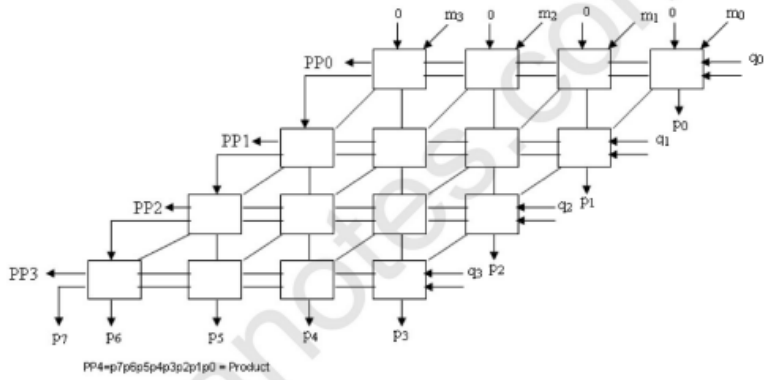
**2. Design a 4x4 array multiplier and explain the steps involved in multiplying two binary numbers using this multiplier.**

**What are the disadvantages of array multiplier?**

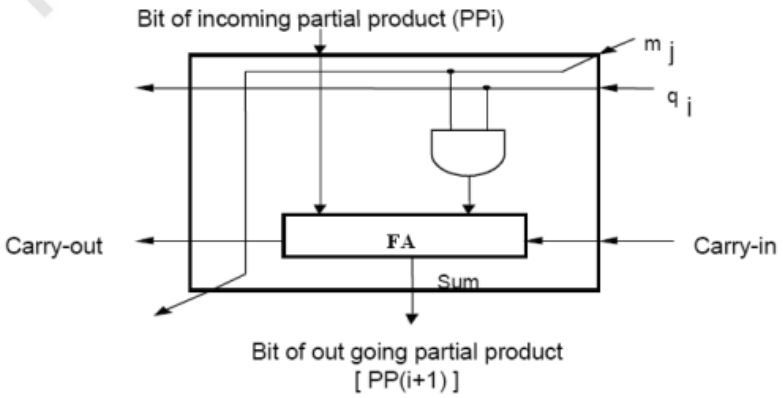
**OR**

**Describe array processor with a neat diagram.**

Binary multiplication of positive operands can be implemented in a combinational two dimensional array.




The AND gate in each cell determines whether a multiplicand bit **mj** is added to the incoming partial product bit, based on the value of the multiplier bit **qj**. Each raw i adds the multiplicand to the incoming partial product **ppi** to generate pp(i+1) if qi=1. If qi=0 ppi is passed downward unchanged.



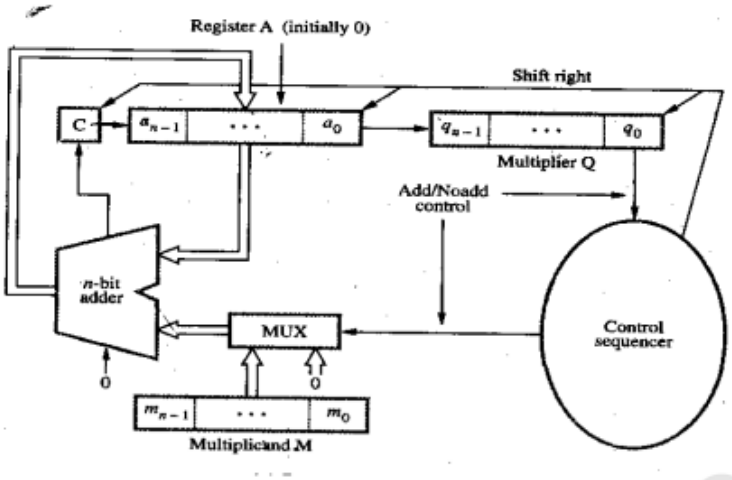
**Disadvantages**:

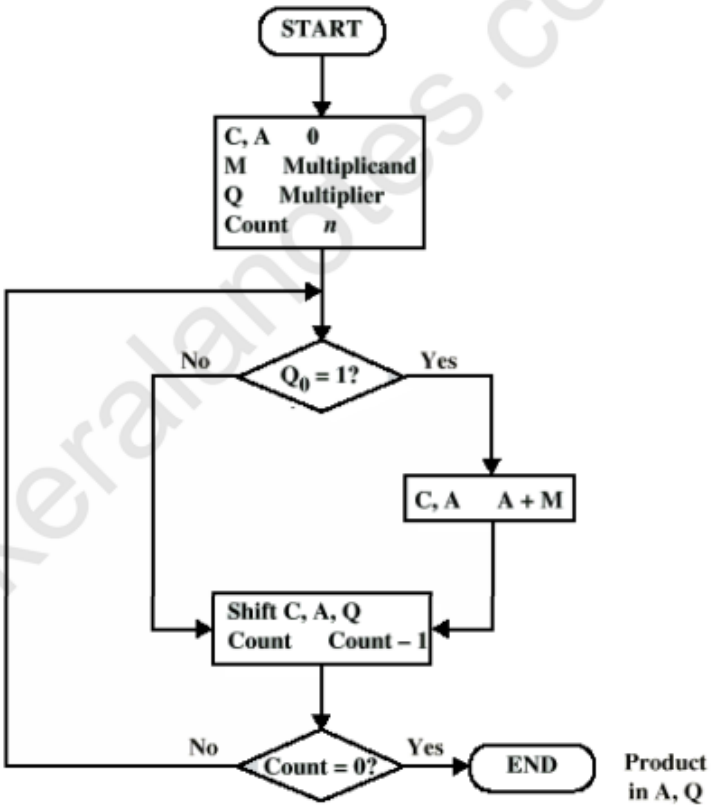
(1) An n bit by n bit array multiplier requires $n^2$ AND gates and n(n-2) full adders and n half adders.(Half aders are used if there are 2 inputs and full adder used if there are 3 inputs).

(2) The longest part of input to output through n adders in top row, n -1 adders in the bottom row and n-3 adders in middle row. The longest in a circuit is called **critical path.**

**3.**

**Expalin sequential multiplier with an example.**



| C | A | Q | M | | |
|---|---|---|---|---|---|
| 0 | 0000 | 1101 | 1011 | Initial Values | |
| 0 | 1011 | 1101 | 1011 | Add | First Cycle |
| 0 | 0101 | 1110 | 1011 | Shift | |
| 0 | 0010 | 1111 | 1011 | Shift | Second Cycle |
| 0 | 1101 | 1111 | 1011 | Add | Third Cycle |
| 0 | 0110 | 1111 | 1011 | Shift | |
| 1 | 0001 | 1111 | 1011 | Add | Fourth Cycle |
| 0 | 1000 | 1111 | 1011 | Shift | |

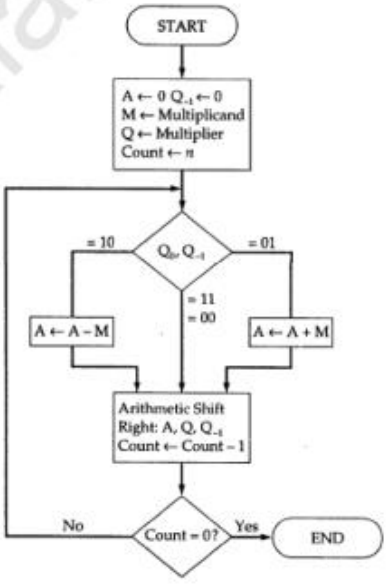
**4.**

**Give the flow chart for Booth's Multiplication.**

**OR**

**Illustrate Booth algorithm with a suitable example.**

When moving from 0 to 1 then -1 is selected and moving from 1 to 0 then +1 is selected as the multiplier scanning from left to right. This algorithm clearly extends to any number of blocks of 1s in a multiplier, including the situation in which a single 1 is considered as a block. If the first bit is 1 then consider the previous bit is 0.




The Multiplicand is placed in M register and multiplier is loaded into register Q. Registers A and $Q_{-1}$ is cleared initially. A bit of multiplier is examined together with the bit in $Q_{-1}$. If

these bits are same (0-0,1-1) then all bits of A,Q,$Q_{-1}$ are shifted right 1 bit. If two bits are differ then the multiplicand is added to or subtracted from A depending on 0-1 or 1-0 then right shift occurs. In either case the right shift $A_{n-1}$ to $A_{n-2}$ occurs and $A_{n-1}$ maintains the same bit for maintaining the sign. This is called Arithmetic shift.

| A | Q | $Q_{-1}$ | M | | |
|---|---|---|---|---|---|
| 0000 | 0011 | 0 | 0111 | Initial Values | |
| 1001 | 0011 | 0 | 0111 | A A - M | First Cycle |
| 1100 | 1001 | 1 | 0111 | Shift | |
| 1110 | 0100 | 1 | 0111 | Shift | Second Cycle |
| 0101 | 0100 | 1 | 0111 | A A + M | Third Cycle |
| 0010 | 1010 | 0 | 0111 | Shift | |
| 0001 | 0101 | 0 | 0111 | Shift | Fourth Cycle |

**5.**

**What are the advantages of Booth Algorithm**

- It handles both positive and negative multiplier uniformly.
- it reduce the number of partial product,
- The speed gained by skipping 1's depends on the data.

## 6. Explain how nested subroutines are processed internally.

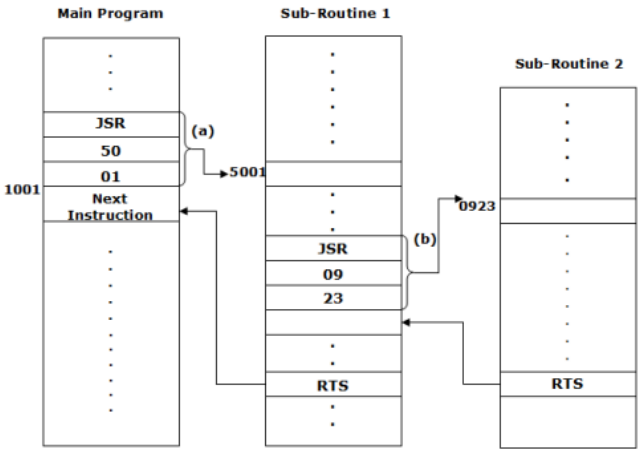
subroutine nesting, is to have one subroutine call another. In this case, the return address of the second call is also stored in the link register, destroying its previous contents. Hence, it is essential to save the contents of the link register in some other location before calling another subroutine. Otherwise, the return address of the first subroutine will be lost. Subroutine nesting can be carried out to any depth. Eventually, the last subroutine called completes its computations and returns to the subroutine that called it. The return address needed for this first return is the last one generated in the nested call Memory location Subroutine SUB 1000 first instruction …. …. Return 204 Return sequence. That is, return addresses are generated and used in a last-in-first-out order. This suggests that the return

addresses associated with subroutine calls should be pushed onto a stack. A particular register is designated as the stack pointer, SP, to be used in this operation. The stack pointer points to a stack called the processor stack. The Call instruction pushes the contents of the PC onto the processor stack and loads the subroutine address into the PC. The Return instruction pops the return address from the processor stack into the PC.

Since PC saving and recovery is automatic by the use of stack, it is possible to execute nested subroutines as shown in figure 2. In other words, it is possible for a subroutine to call another subroutine.

As in figure 2, first the subroutine 1 is called from part 'a' of main program. At this point, the returning address i.e. 1001 should pe pushed on the top of the return stack. From this subroutine, subrotuine2 is called and jump is made to address 0923 i.e. jump 2. At this point, the returning address in subroutine 1 should be pushed onto stack. Thus the return stack should contain the returning addresses in the reverse order of the calling order. First 'a' is called, and then 'b' is called; so returning address of 'b' should be on top of stack and below it should be the returning address of 'a'.

Figure 2: Nested Subroutines referred via main program




**7. Highlight the key steps involved in the restoring method for binary division.**

**OR**

**Divide (1000) 8 by (11) 3 using restoring division method.**

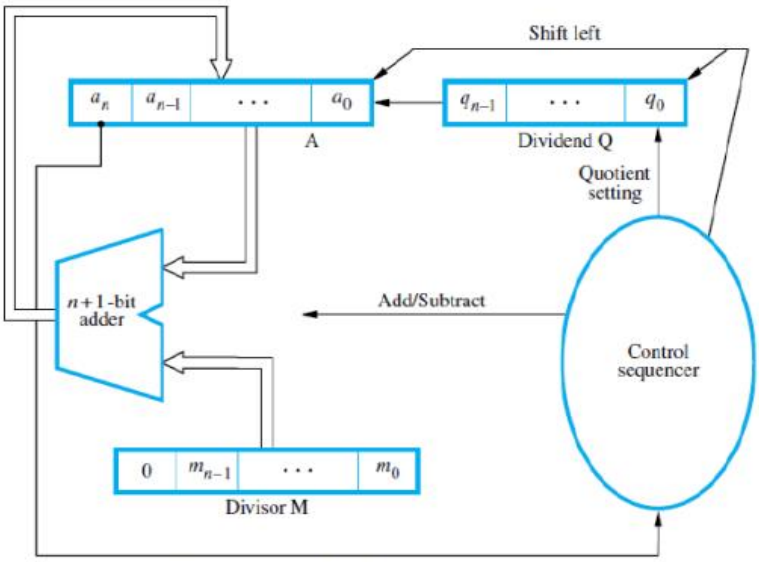
**OR**

**Illustrate restoring division algorithm with an example.**

Restoring Division: Register A is initially loaded with 0 and it consists of n+1 bits, where n is the number of bits in the dividend. Dividend is loaded in register Q and Register M is loaded with the divisor. After division is complete n bit quotient is in register Q and remainder is in register A. extra bit on A and M accommodates the sign bit during subtractions.

If the remainder is negative, a quotient bit of 0 is determined, the dividend is restored by adding back the divisor, and the divisor is repositioned for another subtraction. This is called the **restoring division algorithm.**

***Restoring Division***




Circuit arrangement for binary division.

Do the following three steps n times:

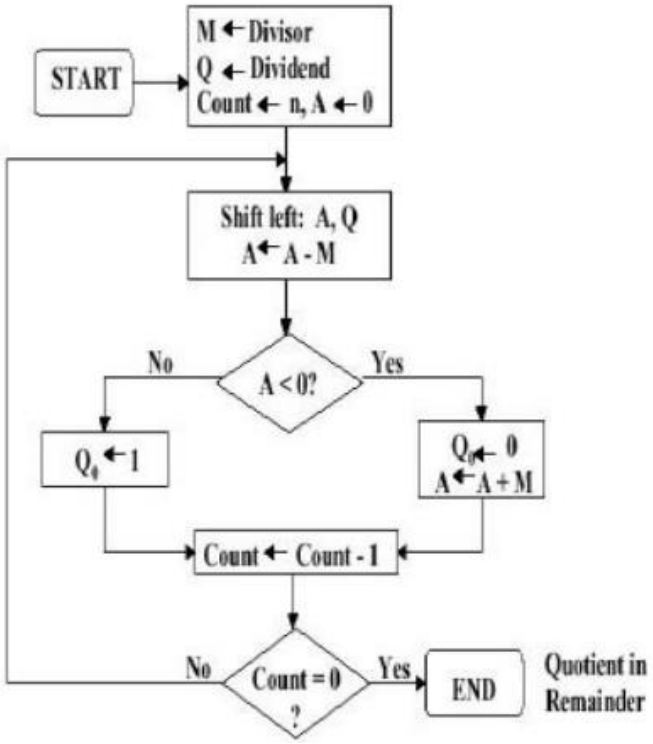
1. Shift A and Q left one bit position.
2. Subtract M from A, ie; (A-M) and place the answer back in A.
3. If the sign of A is 1, set $q_0$ to 0 and add M back to A (that is, restore A); otherwise, set $q_0$ to 1.

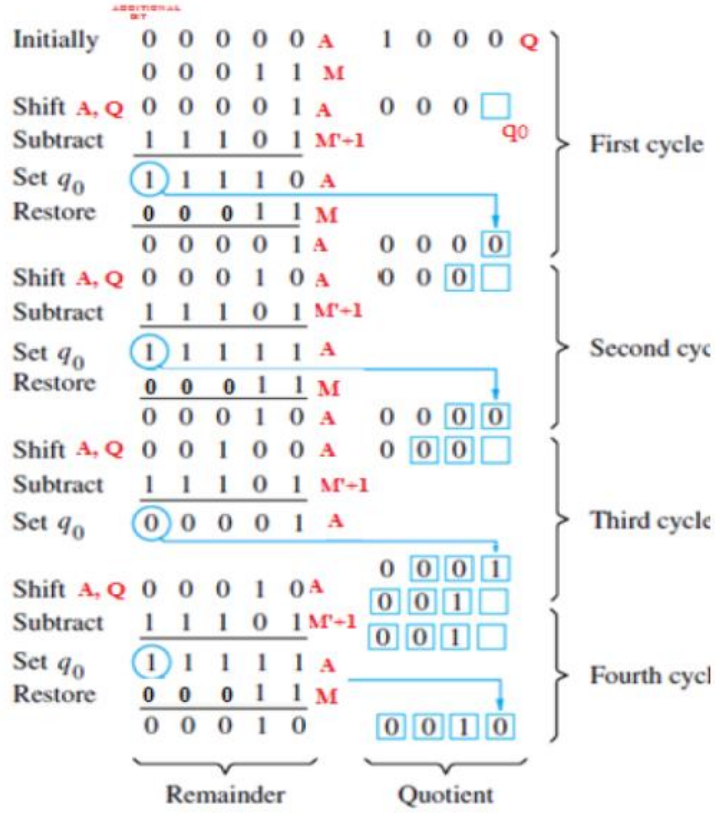
```
        10
   ---------
11 ) 1000   Q
M    11
     --
     10
```

M = 00011

M'+1 = 11100+1

= 11101






A restoring division example.

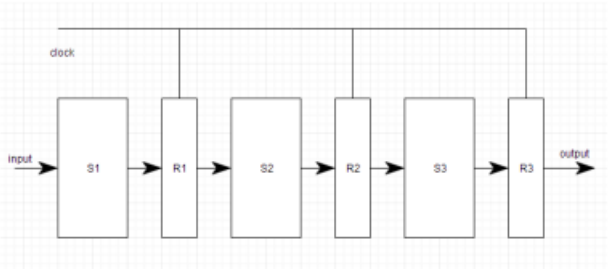
Dividend=8 [1000], Divisor=3 [11]

Quotient=2 [0010], Remainder=2 [00010]

## 8.

**What is Pipelining?**

Pipelining is a technique where multiple instructions are overlapped during execution. Pipeline is divided into stages and these stages are connected with one another to form a pipe like structure. Instructions enter from one end and exit from another end. In pipeline system, each segment consists of an input register followed by a combinational circuit. The register is used to hold data and combinational circuit performs operations on it. The output of combinational circuit is applied to the input register of the next segment.




## 9.

**List the advantages of pipeline.**

- Multiple tasks operating simultaneously using different resources
- Pipelining increases throughput

## 10. What are the different types of pipelines?

1. Arithmetic pipelining
2. Instruction pipelining
3. Processor pipelining

## 11. Explain Instruction pipelining

**Instruction Pipeline:**

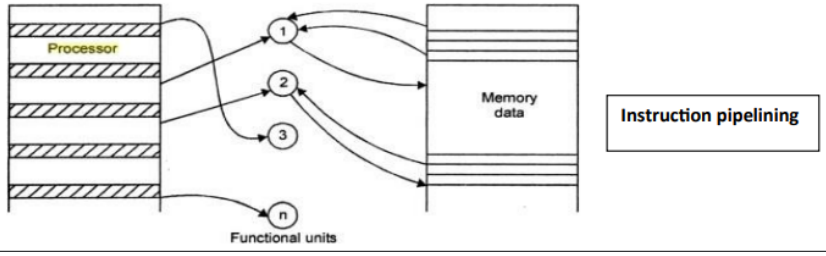
An instruction cycle may consist of many operations like, fetch opcode, decode opcode, compute operand addresses, fetch operands, and execute instructions.

These operations of the instruction execution cycle can be realized through the pipelining concept. Each of these operations forms one stage of a pipeline.

The overlapping of execution of the operations through the pipeline provides a speedup over the normal execution.

Thus, the pipelines used for instruction cycle operations are known as instruction pipelines.

The execution of a stream of instructions can be pipelined **by overlapping** the execution of the current instruction with the fetch, decode and operand fetch of subsequent instructions. This technique is also known as **instruction lookahead.**




Instruction pipelining

## 12.

**What are the various stages in instruction pipeline?**

Stage 1 :**Instruction Fetch (FI)-**In this stage the CPU reads instructions from the address in the memory whose value is present in the program counter.

Stage 2 **:Instruction Decode (DI)-**In this stage, instruction is decoded and the register file is accessed to get the values from the registers used in the instruction.

Stage 3: **Calculate Operands(CO)**-In this stage, effective address of the operands are calculated.

Stage 4: **Fetch Operands (FO)-**memory operands are read from memory

Stage 5 :**Instruction Execute (EI)-**In this stage, ALU operations are performed.

Stage 6: **Write Back (WO)-**In this stage, computed/fetched value is written back to the register present in the instructions.

## 13.

**How pipelining speeding up the performance of a computer?**

Speedup (S) of the pipelined processor over non-pipelined processor, when 'n' tasks are executed on the same processor is:

S= Performance of pipelined processor/ Performance of non pipelined processor

Consider a 'k' segment pipeline with clock cycle time as 'Tp'. Let there be 'n' tasks to be completed in the pipelined processor. Now, the first instruction is going to take 'k' cycles to come out of the pipeline but the other 'n – 1' instructions will take only '1' cycle each, i.e, a total of 'n – 1' cycles. So, time taken to execute 'n' instructions in a pipelined processor.

$ET_{pipeline}$=k+n-1 cycles= (k+n-1)Tp

In the same case, for a non-pipelined processor, execution time of 'n' instructions will be:

$ET_{non\ pipeline}$= n*k*Tp

As the performance of a processor is inversely proportional to the execution time, we have,

$S= ET_{nonipeline}/ET_{pipeline}$= n*k*Tp/ (k+n-1)Tp

= (n*k)/(k+n-1)



## 14. With proper example describe about arithmetic pipelining?

## OR

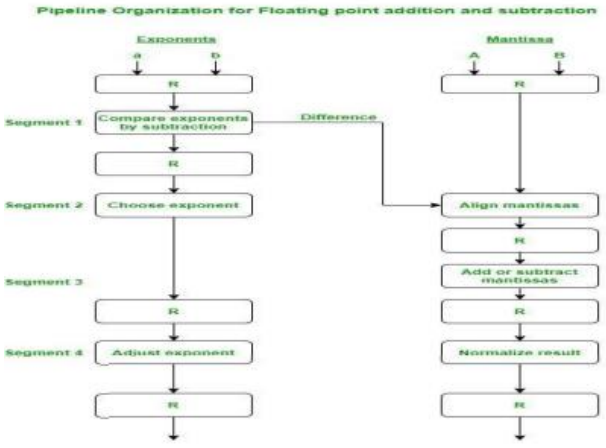
**Explain Arithmetic Pipeline with an example.**

An arithmetic pipeline divides an arithmetic problem into various sub problems for execution in various pipeline segments. It is used for floating point operations, multiplication and various other computations.

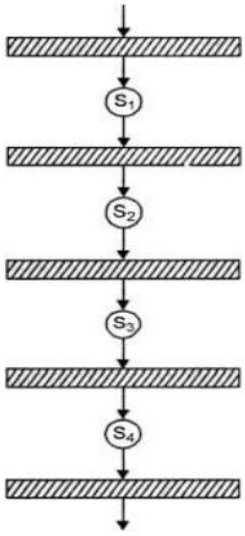
**Example:** Floating point addition using arithmetic pipeline

The following sub operations are performed:

- Compare the exponents.
- Align the mantissas.
- Add or subtract the mantissas.
- Normalize the result



The complex arithmetic operations like multiplication, and floating point operations consume much of the time of the ALU. These operations can also be pipelined **by segmenting** the operations of the ALU and as a consequence, high speed performance may be achieved.




Arithmetic pipelining

## 15.

**What is a pipeline hazards?**

Pipeline hazards are situations that prevent the next instruction in the instruction stream from executing during its designated clock cycles. Any condition that causes a stall in the pipeline operations can be called a hazard.

## 16.

**Define structural, data, and control hazard.**

**Structural hazard**

a. two different instructions use same h/w in same cycle – They arise from the resource conflict
b. HW cannot support all combination of instructions in overlapped fashion (single person to fold and put clothes away)

**Data hazard**

c. They arise when the instruction depends on the result of previous instruction.
d. must appear as if the instructions execute in correct order

**Control hazard**

e. one instruction affects which instruction is next
f. They arise when the pipelining of branches & other instructions that change the PC

**17.**

**List the techniques used for overcoming hazard.**

Structural hazard: Stall, Use of additional hardware

Data Hazard: Stall, Data Forwarding, Code reordering

Control Hazard: Stall, branch predict, delayed branch

**18.**

**Give examples of data hazard.**

Read after Write (RAW): $Instr_J$ tries to read operand before $Instr_I$ writes it

Write after Read (WAR):$Instr_J$ tries to write operand *before* $Instr_I$ reads -Gets wrong operand

Write after Write (WAW): $Instr_J$ tries to write operand *before* $Instr_I$ writes it

- Leaves wrong result ( $Instr_I$ not $Instr_J$ )

Read after Read (RAR).: It occurs when the instruction both read from the same register. No hazard occurs

**19. Explain the various method available to get rid of data hazards inside the system.**

**OR**

**Describe in detail about data hazards and resolution techniques?**

**OR**

**Explain the methods for dealing with data hazard**

- **Answer :**

  Data Hazard –

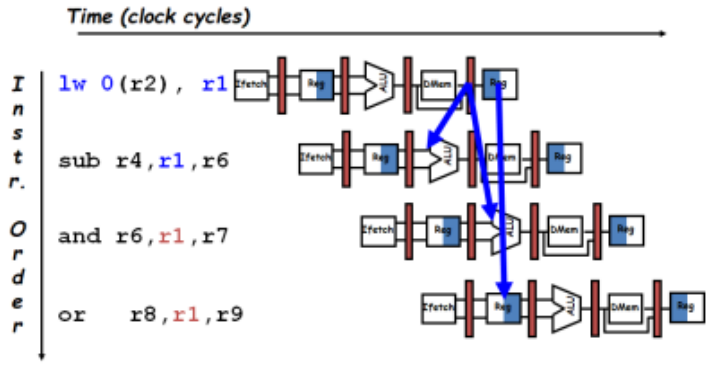
  Can occur when an instruction depends on the result of a previous instruction still being processed in the pipeline.

  Any condition in which either the source or destination operands of an instruction are not available at the time expected in the pipeline.

  As a result some operation has to be delayed and the pipeline stalls.

Resolution techniques:

**Forwarding**: It adds special circuitry to the pipeline. This method works because it takes less time for the required values to travel through a wire than it does for a pipeline segment to compute its result.




**Code reordeing**: We need a special type of software to reorder code. We call this type of software a hardware-dependent compiler.

**Stall insertion**: it inserts one or more installs (no-op instructions) into the pipeline, which delays the execution of the current instruction until the required operand is written to the register file, but this method decreases pipeline efficiency and throughput.

## 20.

**What is branch hazard? Describe the method for dealing with the branch hazard?**

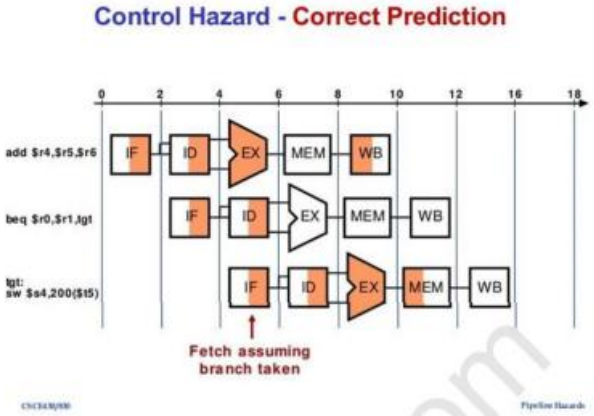
A control/branch hazard is when we need to find the destination of a branch, and can't fetch any new instructions until we know that destination.


**Solutions**

**Stall:** Stop loading instruction until result is available. Stalling for each branch is not practical

**Predict:** Assume an outcome and continue fetching (undo if prediction is wrong)

- Loses cycles only on mis prediction




**Delayed Branch**: Specify in architecture that the instruction immediately following branch is always executed.


**21.**

**Write the example of WAW hazard.**

If instruction X tries to modify some data before it is written by instruction (X-1)

**22. Explain the various pipeline structures available inside a computer**

## Classification of Pipeline processors:

- Various types of pipelining can be applied in computer operations depending on the following factors:
  - **Level of Processing**
    1. Instruction pipelining
    2. Arithmetic pipelining
    3. Processor pipelining

## ○ Pipeline configuration

1. Uni function and Multi function pipelining

Unifunction vs. multifunction pipelines:

A pipeline unit with a fixed and dedicated function, such as the floating point adder is called **unifunctional.**

A **multifunction** pipe may perform different functions, either at different times or at the same time, by interconnecting different subsets of stages in the pipeline.

2. Static and Dynamic pipelining

A static pipeline may assume only one functional configuration at a time.

**Static pipelines** can be either unifunctional or multi functional. Pipelining is made possible in static pipes only if instructions of the same type are to be executed continuously.

A **dynamic pipeline** processor permits several functional configurations to

exist simultaneously. In this sense, a dynamic pipeline must be multifunctional. On the other hand, a unifunctional pipe must be static.

The dynamic configuration needs much more elaborate control and sequencing mechanisms than those for static pipelines.

Most existing computers are equipped with static pipes, either unifunctional or multifunctional.

## ○ Instruction and data type

1. Scalar and Vector pipelining

A **scalar pipeline** processes a sequence of scalar operands under the control of a DO loop. Instructions in a small DO loop are often prefetched into the instruction buffer. The required scalar operands for repeated scalar instructions are moved into a data cache in order to continuously supply the pipeline with operands.

**Vector pipelines** are specially designed to handle vector instructions over vector operands. Computers having vector instructions are often called vector processors. The design of a vector pipeline is expanded from that of a scalar pipeline.

The handling of vector operands in vector pipelines is under firmware and hard ware controls (rather than under software control as in scalar pipelines).

**23. Identify the various types of hazards occurring during the execution of the following program in a pipelined system. Where the pipeline consist of five stages, opcode fetch , instruction decode, operand fetch, execution, store the result. All stages take equal time duration**

**MOV [R1],[R2]**

**MOV R3,[R1]**

**SUB R2,R3**

**ADD R1,R3**

**CALL 5000**

**MOV R2,R3**

- **Answer:**
  - Draw time space diagram

Between $I_1$ & $I_3$ structural

Between $I_1$ & $I_2$ data(RAW)

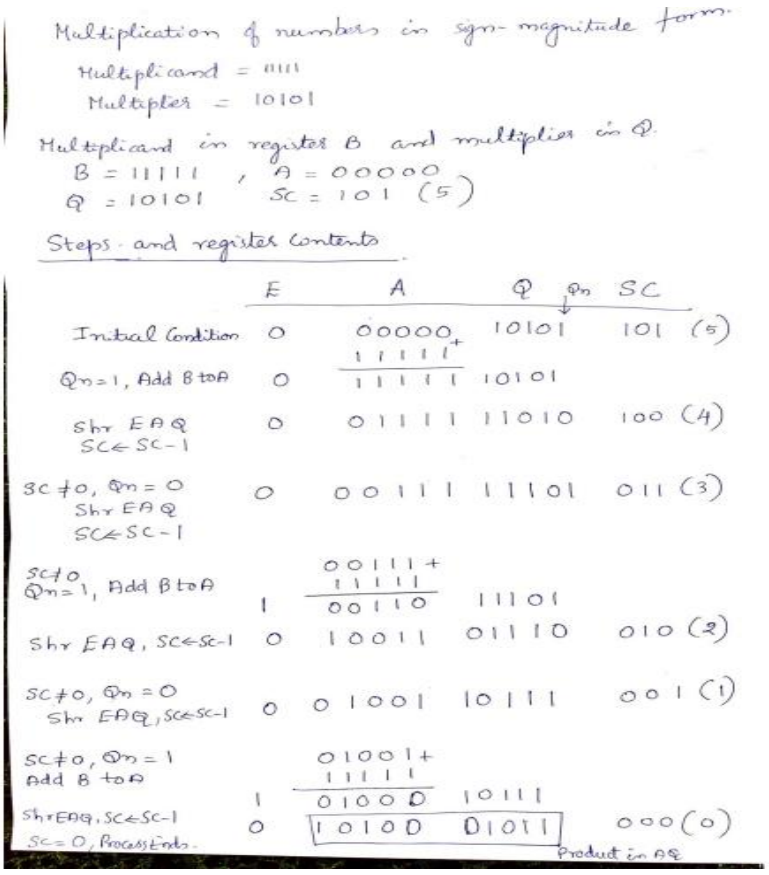
Between $I_2$ & $I_3$ data(RAW)

Between $I_1$ & $I_5$ structural

$I_4$ Control

**24.**

Outline the hardware requirement for multiplying two binary numbers in sign magnitude format and specify a flowchart for same. Illustrate the algorithm, showing the contents of registers, for the multiplication of **11111** by **10101**




**25. Design and draw the block diagram for a 4 by 3 array multiplier.**

(Sol : Draw diagram accordingly, labelling all inputs and outputs)

( Since k=4 and j=3, we need 4x3 = 12 AND gates and two 4 bit adders to produce a product of 7 bits.)

**26. Draw a 3X2 array multiplier?**

(Sol : Draw diagram accordingly, labelling all inputs and outputs)

**27. Differentiate between instruction and arithmetic pipelines.**

**(**Sol : Refer notes and answer accordingly)

**28. Explain in detail about pipeline processors?**

**(**Sol : Refer notes and answer accordingly)

**29. Multiply following using booth's multiplication algorithm: -7 and -3**

(Sol : Solve problem accordingly, labelling all values in table)

**30. Discuss about pipeline hazards?**

**(**Sol : Refer notes and answer accordingly)

**31. Draw the flowchart of Booth's multiplication algorithm and multiply -5 X -4 using booths algorithm?**

(Sol : Solve problem accordingly, labelling all values in table)

**32. Check the correctness of the following statements and justify your results**

**i)All unifunctional pipeline are static. But all static pipeline are not unifunctional.**

**ii)All dynamic pipelines are multifunctional. But all multifunctional pipeline are not dynamic.**

***Answer :**

i) All unifunctional pipelines are static . But all static pipeline are not unifunctional.
→ TRUE

A pipeline unit with fixed and dedicated function is called unifunctional.

Static pipelines can be either unifunctional or multi functional. Pipelining is made possible in static pipes only if instructions of the same type are to be executed continuously

ii)All dynamic pipelines are multifunctional. But all multifunctional pipeline are not dynamic. → TRUE

A multifunction pipe may perform different functions, either at different times or at the same time, by interconnecting different subsets of stages in the pipeline.

A dynamic pipeline processor permits several functional configurations to exist simultaneously. In this sense, a dynamic pipeline must be multifunctional.

**33. Draw the flow chart of Booth's multiplication algorithm. Multiply (+24) and (-21) using Booth's multiplication algorithm.**

(Sol : Solve problem accordingly, labelling all values in table)

**34. Illustrate Read After Write (RAW) hazard with an example**

**(**Sol : Refer notes and answer accordingly)

**35. Analyze the hazards associated with pipeline processors and propose solutions for overcoming data hazards in pipeline architecture.**

**(**Sol : Refer notes and answer accordingly)

**36. What do you mean by array multiplier, design 3x3 array multiplier and list out its disadvantages?**

(Sol : Draw diagram accordingly, labelling all inputs and outputs)

**37. Explain the principle of pipelining and discuss how it enhances the overall performance of a processor. Provide examples to support your explanation.**

**38. Describe in detail about instruction hazards and their solution?**