The execution of instructions define the instruction cycle. This is the thorough methodology computer processors use for executing a given instruction. Many times processors can be compared to combustion engines. Both follow a process continuously being carried out to fetch the desired outcome. Every processor shows a three-step instruction cycle. These three steps of the instruction execution cycle are,

### 1.Fetch: The processor copies the instruction data captured from the RAM.

### 2. Decode: Decoded captured data is transferred to the unit for execution.

### 3. Execute:Instruction is finally executed. The result is then registered in the processor or RAM (memory address).

## First step: Fetch (instruction cycle)

According to the execution instruction definition, the instruction cycle’s first step is to capture or fetch the instruction. This instruction in the fetch stage is captured from RAM. This memory is assigned to the processor through various units and registers; they are:

## Second step: Decode (instruction cycle)

## There are various instructions, and we can never be sure which instruction belongs to which execution unit. Decoding sorts this out. A decoder is responsible for taking in the instruction and decoding it to assign the respective execution unit to complete the execution instruction cycle.

## Third step: Execute (instruction cycle)

The last stage of the execute instruction definition is to execute. It involves executing the given instruction that was fetched at the first stage. No two instructions ever get resolved in the same manner because their ways of utilising the hardware depend on their functions. There are four types of instructions that are generally present,

2. **For the expression X=(A+B)\*(C+D) when evaluated on stack machine how many** **number of machine instructions are required?**

Zero Address Instructions –

A stack-based computer does not use the address field in the instruction. To evaluate an expression first it is converted to reverse Polish Notation i.e. Postfix Notation.

2. One Address Instructions –

This uses an implied ACCUMULATOR register for data manipulation. One operand is in the accumulator and the other is in the register or memory location. Implied means that the CPU already knows that one operand is in the accumulator so there is no need to specify it.

3. Two Address Instructions –

This is common in commercial computers. Here two addresses can be specified in the instruction. Unlike earlier in one address instruction, the result was stored in the accumulator, here the result can be stored at different locations rather than just accumulators, but require more number of bit to represent address. Here destination address can also contain operand.

4. Three Address Instructions –

This has three address field to specify a register or a memory location. Program created are much short in size but number of bits per instruction increase. These instructions make creation of program much easier but it does not mean that program will run much faster because now instruction only contain more information but each micro operation (changing content of register, loading address in address bus etc.) will be performed in one cycle only.

*6*operations would be performed it can be calculated using stack polish notation

Question 6

1. **Explain instruction formats with examples.**

Zero Address Instructions –

A stack-based computer does not use the address field in the instruction. To evaluate an expression first it is converted to reverse Polish Notation i.e. Postfix Notation.

2 .One Address Instructions –

This uses an implied ACCUMULATOR register for data manipulation. One operand is in the accumulator and the other is in the register or memory location. Implied means that the CPU already knows that one operand is in the accumulator so there is no need to specify it.

3. Two Address Instructions –

This is common in commercial computers. Here two addresses can be specified in the instruction. Unlike earlier in one address instruction, the result was stored in the accumulator, here the result can be stored at different locations rather than just accumulators, but require more number of bit to represent address. Here destination address can also contain operand.

4. Three Address Instructions –

This has three address field to specify a register or a memory location. Program created are much short in size but number of bits per instruction increase. These instructions make creation of program much easier but it does not mean that program will run much faster because now instruction only contain more information but each micro operation (changing content of register, loading address in address bus etc.) will be performed in one cycle only.

**Question 7**

1. **Differentiate between hardwired control and micro programmed control. Is it Possible to have a hardwired control associated with a control memory?**

### **Hardwired Control Unit**

With the help of generating control signals, the hardwired control unit is able to execute the instructions at a correct time and proper sequence. As compared to the micro-programmed, the hardwired CU is generally faster. In this CU, the control signals are generated with the help of PLA circuit and state counter. Here the Central processing unit requires all these control signals. With the help of hardware, the hardwired control signals are generated, and it basically uses the circuitry approach.

The image of a hardwired control unit is described as follows, which contains various components in the form of circuitry. We will discuss them one by one so that we can properly understand the generation of control signals So, on the basis of the input obtained by the conditional signals, step counter, external inputs, and instruction register, the control signals will be generated with the help of Control signal Generator.

### **Micro-programmed Control Unit**

A micro-programmed control unit can be described as a simple logic circuit. We can use it in two ways, i.e., it is able to execute each instruction with the help of generating control signals, and it is also able to do sequencing through microinstructions. It will generate the control signals with the help of programs. At the time of evolution of CISC architecture in the past, this approach was very famous. The program which is used to create the control signals is known as the "Micro-program". The micro-program is placed on the processor chip, which is a type of fast memory. This memory is also known as the control store or control memory.

A micro-program is used to contain a set of microinstructions. Each microinstruction or control word contains different bit patterns. The n bit words are contained by each microinstruction. On the basis of the bit pattern of a control word, every control signals differ from each other.

Like the above, the instruction execution in a micro-programmed control unit is also performed in steps. So for each step, the micro-program contains a control word/ microinstruction. If we want to execute a particular instruction, we need a sequence of microinstructions. This process is known as the micro-routine. The image of a micro-programmed control unit is described as follows. Here, we will learn the organization of micro-program, micro-routine, and control word/ microinstruction.

2. **Explain various memory based addressing modes in detail with the help of suitable diagram.**

## **1. Implied Addressing Mode-**

In this addressing mode,

The definition of the instruction itself specify the operands implicitly.

It is also called as implicit addressing mode.

## Examples-

The instruction “Complement Accumulator” is an implied mode instruction.

In a stack organized computer, Zero Address Instructions are implied mode instructions.

(since operands are always implied to be present on the top of the stack)

## **2. Stack Addressing Mode-**

In this addressing mode, The operand is contained at the top of the stack.

## Example-

ADD

This instruction simply pops out two symbols contained at the top of the stack.

The addition of those two operands is performed.

The result so obtained after addition is pushed again at the top of the stack.

## **3. Immediate Addressing Mode-**

In this addressing mode,

The operand is specified in the instruction explicitly.

Instead of address field, an operand field is present that contains the operand

## Examples-

ADD 10 will increment the value stored in the accumulator by 10.

MOV R #20 initializes register R to a constant value 20.

## **4. Direct Addressing Mode-**

In this addressing mode,

The address field of the instruction contains the effective address of the operand.

Only one reference to memory is required to fetch the operand.

It is also called as absolute addressing mode

## Example-

ADD X will increment the value stored in the accumulator by the value stored at memory location X.

AC ← AC + [X]

## **5. Indirect Addressing Mode-**

In this addressing mode,

The address field of the instruction specifies the address of memory location that contains the effective address of the operand.

Two references to memory are required to fetch the operand.

Example-

ADD X will increment the value stored in the accumulator by the value stored at memory location specified by X.

AC ← AC + [[X]]

## **6. Register Direct Addressing Mode-**

In this addressing mode,

The operand is contained in a register set.

The address field of the instruction refers to a CPU register that contains the operand.

No reference to memory is required to fetch the operand.

## Example-

ADD R will increment the value stored in the accumulator by the content of register R.

AC ← AC + [R]

## NOTE-

It is interesting to note-

This addressing mode is similar to direct addressing mode.

The only difference is address field of the instruction refers to a CPU register instead of main memory.

## **7. Register Indirect Addressing Mode-**

In this addressing mode,

The address field of the instruction refers to a CPU register that contains the effective address of the operand.

Only one reference to memory is required to fetch the operand.

Example-

ADD R will increment the value stored in the accumulator by the content of memory location specified in register R.

AC ← AC + [[R]]

## NOTE-

It is interesting to note-

This addressing mode is similar to indirect addressing mode.

The only difference is address field of the instruction refers to a CPU register.

## **8. Relative Addressing Mode-**

In this addressing mode,

Effective address of the operand is obtained by adding the content of program counter with the address part of the instruction.

|  |
| --- |
| Effective Address  = Content of Program Counter + Address part of the instruction |

## **9. Indexed Addressing Mode-**

In this addressing mode,

Effective address of the operand is obtained by adding the content of index register with the address part of the instruction.

|  |
| --- |
| Effective Address  = Content of Index Register + Address part of the instruction |

## **10. Base Register Addressing Mode-**

In this addressing mode,

Effective address of the operand is obtained by adding the content of base register with the address part of the instruction.

|  |
| --- |
| Effective Address  = Content of Base Register + Address part of the instruction |

## **11. Auto-Increment Addressing Mode-**

This addressing mode is a special case of Register Indirect Addressing Mode where-

In this addressing mode,

After accessing the operand, the content of the register is automatically incremented by step size ‘d’.

Step size ‘d’ depends on the size of operand accessed.

Only one reference to memory is required to fetch the operand.

## **12. Auto-Decrement Addressing Mode-**

This addressing mode is again a special case of Register Indirect Addressing Mode where

In this addressing mode,

First, the content of the register is decremented by step size ‘d’.

Step size ‘d’ depends on the size of operand accessed.

After decrementing, the operand is read.

Only one reference to memory is required to fetch the operand.

## NOTE-

In auto-decrement addressing mode,

First, the instruction register RAUTO value is decremented by step size ‘d’.

Then, the operand value is fetched.

Question 8.

1. **Discuss booth algorithm and perform -5\*2 using this.**

* 1. Set the Multiplicand and Multiplier binary bits as M and Q, respectively.
  2. Initially, we set the AC and Qn + 1 registers value to 0.
  3. SC represents the number of Multiplier bits (Q), and it is a sequence counter that is continuously decremented till equal to the number of bits (n) or reached to 0.
  4. A Qn represents the last bit of the Q, and the Qn+1 shows the incremented bit of Qn by 1.
  5. On each cycle of the booth algorithm, Qn and Qn + 1 bits will be checked on the following parameters as follows:
     1. When two bits Qn and Qn + 1 are 00 or 11, we simply perform the arithmetic shift right operation (ashr) to the partial product AC. And the bits of Qn and Qn + 1 is incremented by 1 bit.
     2. If the bits of Qn and Qn + 1 is shows to 01, the multiplicand bits (M) will be added to the AC (Accumulator register). After that, we perform the right shift operation to the AC and QR bits by 1.
     3. If the bits of Qn and Qn + 1 is shows to 10, the multiplicand bits (M) will be subtracted from the AC (Accumulator register). After that, we perform the right shift operation to the AC and QR bits by 1.
  6. The operation continuously works till we reached n - 1 bit in the booth algorithm.
  7. Results of the Multiplication binary bits will be stored in the AC and QR registers.

2**. Discuss division algorithm (restoration method) and solve 10/3 using this.**

A division algorithm provides a quotient and a remainder when we divide two number. They are generally of two type slow algorithm and fast algorithm. Slow division algorithm are restoring, non-restoring, non-performing restoring, SRT algorithm and under fast comes Newton–Raphson and Goldschmidt.

Let’s pick the step involved:

Step-1: First the registers are initialized with corresponding values (Q = Dividend, M = Divisor, A = 0, n = number of bits in dividend)

Step-2: Then the content of register A and Q is shifted left as if they are a single unit

Step-3: Then content of register M is subtracted from A and result is stored in A

Step-4: Then the most significant bit of the A is checked if it is 0 the least significant bit of Q is set to 1 otherwise if it is 1 the least significant bit of Q is set to 0 and value of register A is restored i.e the value of A before the subtraction with M

Step-5: The value of counter n is decremented

Step-6: If the value of n becomes zero we get of the loop otherwise we repeat from step 2

Step-7: Finally, the register Q contain the quotient and A contain remainder

Question 9

1. **What do you mean by Memory Hierarchy and Cache Organization ? Also explain different mapping in cache (memory interfacing).**

In the Computer System Design, Memory Hierarchy is an enhancement to organize the memory such that it can minimize the access time. The Memory Hierarchy was developed based on a program behavior known as locality of references.The figure below clearly demonstrates the different levels of memory hierarchy :

This Memory Hierarchy Design is divided into 2 main types:

External Memory or Secondary Memory –

Comprising of Magnetic Disk, Optical Disk, Magnetic Tape i.e. peripheral storage devices which are accessible by the processor via I/O Module.

Internal Memory or Primary Memory –

Comprising of Main Memory, Cache Memory & CPU registers. This is directly accessible by the processor.

Cache is close to CPU and faster than main memory. But at the same time is smaller than main memory. The cache organization is about mapping data inweb memory to a location in cache. A Simple Solution: One way to go about this mapping is to consider last few bits of long memory address to find small cache address, and place them at the found address.

Cache Mapping: There are three different types of mapping used for the purpose of cache memory which is as follows: Direct mapping, Associative mapping, and Set-Associative mapping. These are explained below.

A. Direct Mapping

The simplest technique, known as direct mapping, maps each block of main memory into only one possible cache line. or In Direct mapping, assign each memory block to a specific line in the cache. If a line is previously taken up by a memory block when a new block needs to be loaded, the old block is trashed. An address space is split into two parts index field and a tag field. The cache is used to store the tag field whereas the rest is stored in the main memory. Direct mapping`s performance is directly proportional to the Hit ratio.

B. Associative Mapping

In this type of mapping, the associative memory is used to store content and addresses of the memory word. Any block can go into any line of the cache. This means that the word id bits are used to identify which word in the block is needed, but the tag becomes all of the remaining bits. This enables the placement of any word at any place in the cache memory. It is considered to be the fastest and the most flexible mapping form. In associative mapping the index bits are zero.

C. Set-associative Mapping

This form of mapping is an enhanced form of direct mapping where the drawbacks of direct mapping are removed. Set associative addresses the problem of possible thrashing in the direct mapping method. It does this by saying that instead of having exactly one line that a block can map to in the cache, we will group a few lines together creating a set. Then a block in memory can map to any one of the lines of a specific set. Set-associative mapping allows that each word that is present in the cache can have two or more words in the main memory for the same index address. Set associative cache mapping combines the best of direct and associative cache mapping techniques.

2. **Explain pipelining in detail i.e. performance, advantages, implementation in Control Unit etc. Also explain pipeline Hazard and ways to overcome them.**

Pipelining is a process of arrangement of hardware elements of the CPU such that its overall performance is increased. Simultaneous execution of more than one instruction takes place in a pipelined processor. Let us see a real-life example that works on the concept of pipelined operation. Consider a water bottle packaging plant. Let there be 3 stages that a bottle should pass through, Inserting the bottle(I), Filling water in the bottle(F), and Sealing the bottle(S). Let us consider these stages as stage 1, stage 2, and stage 3 respectively. Let each stage take 1 minute to complete its operation. Now, in a non-pipelined operation, a bottle is first inserted in the plant, after 1 minute it is moved to stage 2 where water is filled. Now, in stage 1 nothing is happening. Similarly, when the bottle moves to stage 3, both stage 1 and stage 2 are idle. But in pipelined operation, when the bottle is in stage 2, another bottle can be loaded at stage 1. Similarly, when the bottle is in stage 3, there can be one bottle each in stage 1 and stage 2. So, after each minute, we get a new bottle at the end of stage 3. Hence, the average time taken to manufacture 1 bottle is:

Without pipelining = 9/3 minutes = 3m

With pipelining = 5/3 minutes = 1.67m

Overlapped execution:

Total time = 5 Cycle Pipeline Stages RISC processor has 5 stage instruction pipeline to execute all the instructions in the RISC instruction set. Following are the 5 stages of the RISC pipeline with their respective operations:

Stage 1 (Instruction Fetch) 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) 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 (Instruction Execute) In this stage, ALU operations are performed.

Stage 4 (Memory Access) In this stage, memory operands are read and written from/to the memory that is present in the instruction.

Stage 5 (Write Back) In this stage, computed/fetched value is written back to the register present in the instructions.

Dependencies in a pipelined processor

There are mainly three types of dependencies possible in a pipelined processor. These are :

* + 1. Structural Dependency
    2. Control Dependency
    3. Data Dependency

These dependencies may introduce stalls in the pipeline.

Stall : A stall is a cycle in the pipeline without new input.

**Structural dependency**

This dependency arises due to the resource conflict in the pipeline. A resource conflict is a situation when more than one instruction tries to access the same resource in the same cycle. A resource can be a register, memory, or ALU.

**Control Dependency (Branch Hazards)**

This type of dependency occurs during the transfer of control instructions such as BRANCH, CALL, JMP, etc. On many instruction architectures, the processor will not know the target address of these instructions when it needs to insert the new instruction into the pipeline. Due to this, unwanted instructions are fed to the pipeline.

**Data Dependency (Data Hazard)**

A position in which an instruction is dependent on a result from a sequentially earlier instruction before it can be done its execution. In high-performance processors operating pipeline or superscalar techniques, a data dependency will learn an interruption in the flowing services of a processor pipeline or prevent the parallel issue of instructions in a superscalar processor.

Consider two instructions ik and ii of the same program, where ik precedes ii. If ik and ii have a common register or memory operand, they are data-dependent on each other, except when the common operand is used in both instructions as a source operand.Data dependency can appear either in ‘straight-line code’ between subsequent instructions or in a loop between instructions belonging to subsequent iterations of a loop

Data dependencies in straight-line code