4.7 INTERRUPTS: CONCEPT AND TYPES

The mechanism, by which the execution control of the CPU temporarily gets transferred from one module to another module or it can be said as the CPU temporarily gets jumped from one program routine execution to another one program routine execution, is known as Interrupt. Now, for happening so, there must have to be one input event or signal that will trigger the CPU for getting its control transferred, and that input event is known as Interrupt Request Signal and the signal issued in response to the interrupt request to let inform the source of the interrupt request that interrupt request is granted, is known as Interrupt Acknowledge Signal. Again, the routine executed in response to an interrupt request is known as Interrupt Service Routine. Interrupt Service Routine or ISR is similar to sub-routine of a program, but may not have any relationship with the program.

Now, let us take an example scenario for transferring the control of the CPU execution from one program segment to another one program segment or routine, through interrupt, and let's have a look to observe that how to make it possible.

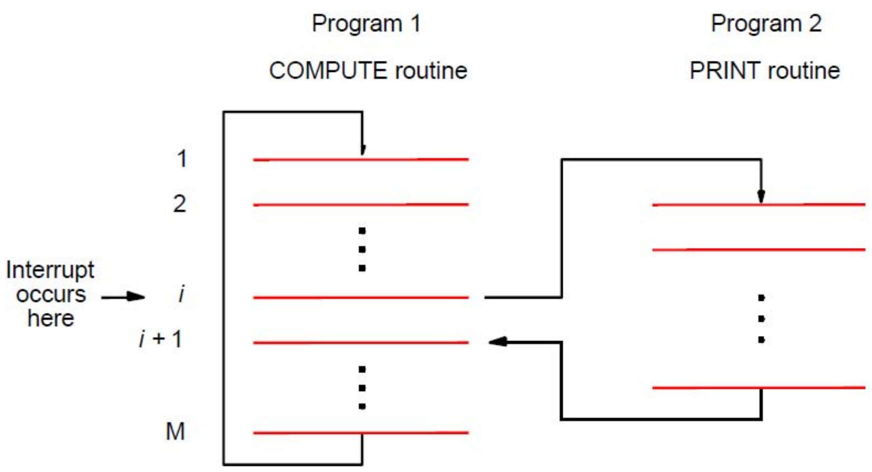

Figure: 4.4 Execution of an Interrupt request

In the above **figure 4.4**, it has been shown that the interrupt request arrives during execution of the ith instruction of the program 1 (i.e. COMPUTE routine) and then the CPU suspends the execution of the current program being executed, i.e. COMPUTE Routine and saves the address of the next instruction i.e. (i+1)th instruction (or you can say the current content of the program counter) and saves any other relevant information to the processor current activity in the temporary storage. Then, the CPU loads the program counter with the base address/ address of the first instruction of the interrupt service routine, i.e. PRINT Routine and executes the PRINT routine. After finishing the execution of the interrupt service routine, the CPU will load the program counter with address of the i+1th instruction of COMPUTE Routine that is already saved in temporary storage and get back for the execution of the COMPUTE routine, starting from the i+1th instruction.

Types of the interrupts:

Basically, there are two types of the interrupts- Hardware Interrupt and Software Interrupt. Look at the following for some classes of interrupts belonging to either hardware interrupt or software interrupt-

Program: Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, or reference outside a user's allowed memory space.

Timer: Generated by the timer within the processor. This allows the operating system to perform certain functions on a regular basis.

I/O: Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.

Hardware Failure: Generated by a failure such as power failure or memory parity error.

Handling Multiple Interrupts:

At this point, we are very much familiar with what an interrupt is, what an interrupt request signal is, what an interrupt acknowledgement signal is, what an interrupt service routine is, types of interrupt and how to handle an interrupt. So, it is the time to handle multiple interrupt as it is possible to occur at the same time, because it is possible for occurring other interrupts while one interrupt service routine is being processed. So, there should have a module that can handle the interrupt requests from various devices and allow one by one to the processor and is known as Interrupt Controller. Basically, there are two methods for handling multiple interrupts and they are as follows-

- **Disabling the interrupts:** Here, question comes to mind that what a disabled interrupt is. A disabled interrupt means that the CPU can and will ignore that interrupt request signal. If an interrupt occurs during this time, it generally remains pending and will be checked by the CPU after the CPU has enabled the interrupts. So, when a user program is executing and interrupt occurs, interrupts are disabled immediately, and after the completion of the interrupt service routine, interrupts are enabled before resuming the user program, and the processor checks to see if additional interrupts have occurred.

  The main disadvantage of this approach is that it does not take into account time-critical needs.

- **Defining priorities to the interrupts:** Here, question comes to mind that what a priority interrupts is. A priority interrupt is an interrupt that establishes a priority over the various sources to determine which condition is to be serviced first when two or more requests arrive simultaneously.

  In this method, priorities will be defined for the interrupts and will be allowed a higher priority interrupt to cause a lower priority interrupt service routine to be itself interrupted. When the interrupt service routine for the higher priority interrupt will be completed, then the interrupt service routine of the lower priority will be resumed first and finally control comes to the user program. So, it's a nested kind of nature of the transfer of the control of the CPU.

4.8 INSTRUCTION EXECUTION CYCLE AND INTERRUPT

We know that the computer is for executing the instructions of the program stored in the memory and the processing required for executing a single instruction is known as Instruction Cycle and one important thing is that one instruction cycle consists of two sub-cycles or steps-

1. Read the instruction from the memory to Instruction register, called Fetch cycle.

2. Execute the instruction from the instruction register, called Execute Cycle.

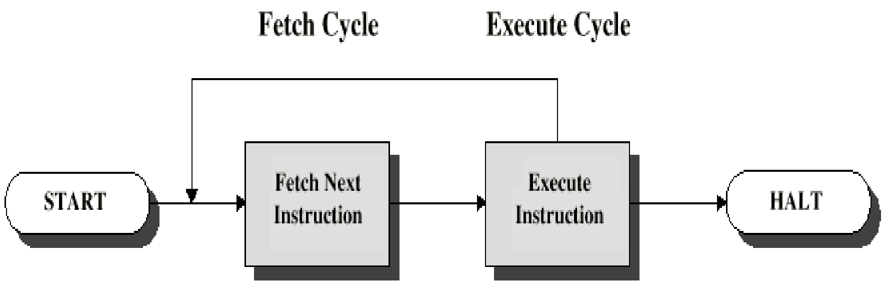

Figure-4.5 An instruction cycle

In the fetch cycle, CPU loads the instruction in the instruction register from the memory location address which is holding by the program counter and then program counter will be incremented by one to hold the address of the next instruction to be executed.

In the execute cycle, interprets the opcode of the instruction and perform the indicated operation.

So, the basic instruction cycle states are as follow-

1. Determine the address of the next instruction.
2. Read the instruction from the memory to instruction register.
3. Interprets the instructions for the operation and about the operand.
4. If operation involves a reference to an operand in memory or in I/O, then calculate the address of the operand and read the operand form memory or I/O.
5. Perform the operation indicated in the instruction.
6. Store the result back into the memory or out through the I/O

So, the important thing is that one more sub-cycle may be introduced within the execute cycle, called indirect cycle. Because, after an instruction is fetch, it is examined to determine if any indirect addressing is involved and if so, the required operand is fetch using indirect addressing. Therefore, if the execution of an instruction involves one or more operands in memory, each of which requires a memory access and it is known as indirect cycle.

Now, what will happen if we consider the occurrence of interrupt in the execute cycle. Suppose, an interrupt occurs when the CPU is executing an instruction, then the CPU will not transferred its control to the interrupt service routine without completing the execution of current instruction. So finally, the instruction cycle can be represented as follows-

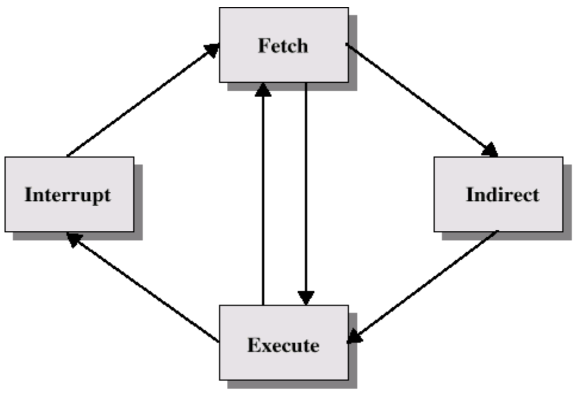

Figure- 4.6 A complete instruction cycle