**UNIT - VI**

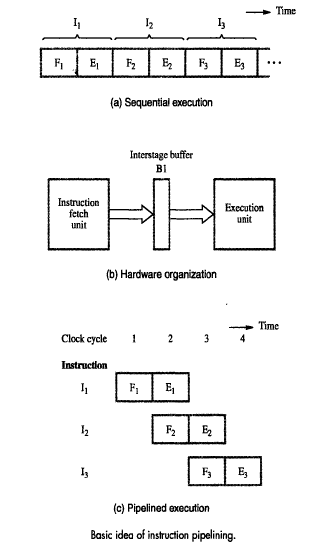
**PIPELINING**

**BASIC CONCEPT OF PIPELINING:**

The speed of execution of programs is influenced by many factors. One way to improve performance is to use faster circuit technology to build the processor and the main memory. Another possibility is to arrange the hardware so that more than one operation can be performed at the same time that is called Pipelining. Pipelining is particularly effective way of organizing concurrent activity in a computer system.

**Two Stage Pipelining:**

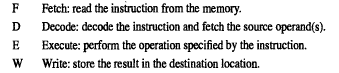
* The Processor executes a program by fetching and executing instructions, one after the other.
* Let Fi and Ei refer to the fetch and execute steps for instruction Ii. Executions of a program consist of a sequence of fetch and execute steps.
* Now consider a computer that has two separate hardware units, one for fetching instructions and another for executing them.
* The instruction fetched by the fetch unit is deposited in an intermediate storage buffer B1. This buffer is needed to enable the execution unit to execute the instruction while the fetch unit is fetching the next instruction.
* The results of execution are deposited in the destination location specified by the instruction. For the purposes of both the source and the destination of the data operated on by the instructions are inside the block labeled Execution unit.

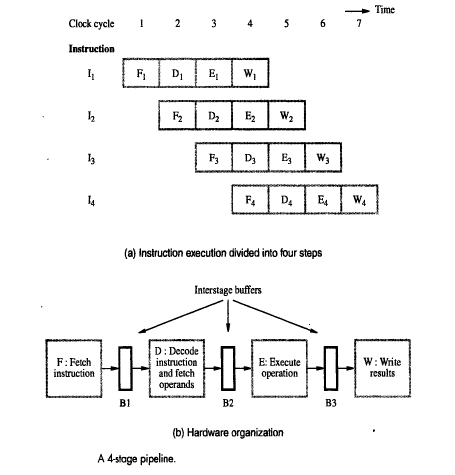


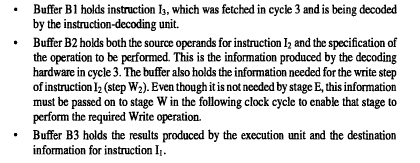
* The computer is controlled by a clock whose period is such that the fetch and execute steps of any instruction can each be completed in one clock cycle.
* In the first clock cycle, the fetch unit fetches an instruction I1 and stores it in buffer B1 at the end of the clock cycle.
* In the second clock cycle, the instructions fetch unit proceeds with the fetch operation for instruction I2. The execution unit performs the operation specified by instruction I1, which is available to it in buffer B1.
* By the end of the second clock cycle, the execution of instruction I1 is completed and instruction I2 is available. Instruction I2 is stored in B1, replacing I1, which is no longer needed.
* Step E2 is performed by the execution unit during the third clock cycle, while instruction I3 is being fetched by the fetch unit. In this manner, both the fetch and execute units are kept busy all the time.

**Four Stage Pipelining:**

The Processing of an instruction need not be divided into only two steps. For example, a pipelined processor may process each instruction in four steps, as follows:

****

****

****

**Role of Cache Memory:**

* Each stage in a pipeline is expected to complete its operation in one clock cycle.
* If different units require different amounts of time, the clock period must allow the longest task to be completed.
* A unit that completes its task early is idle for the remainder of the clock period. Hence, pipelining is most effective in improving performance if the tasks being performed in different stages require about the same amount of time.
* The clock cycle has to be equal to or greater than the time needed to complete a fetch operation.
* However, the access time of the main memory may be as much as ten times greater than the time needed to perform basic pipeline stage operations inside the processor.
* Thus, if each instruction fetches required access to the main memory, pipelining would be of little value.
* The use of cache memories solves the memory access problem. In particular, when a cache is included on the same chip as the processor, access time to the cache is usually the same as the time needed to perform other basic operations inside the processor.

**Pipeline Performance:**

* The pipelined processor completes the processing of one instruction in each clock cycle, which means that the rate of instruction processing is four times that of sequential operation.
* The potential increase in performance resulting from pipelining is proportional to the number of pipeline stages.
* One of the pipeline stages may not be able to complete its processing task for a given instruction in the time allotted.

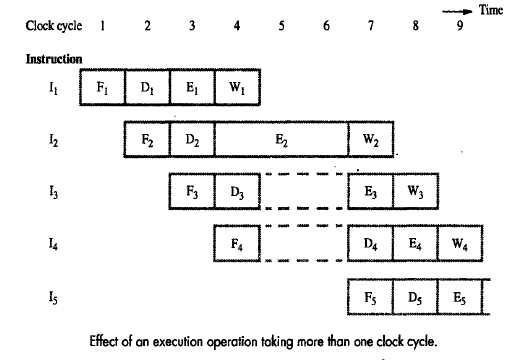
**What is Hazard?**

Any condition that causes the pipeline to stall is called a Hazard. Hazards are three types.

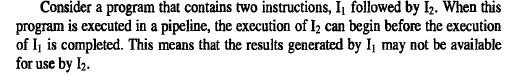
1. Data Hazards
2. Instruction Hazards
3. Structural Hazards

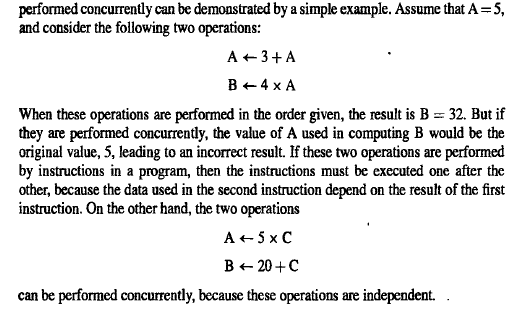
**Data Hazards:**

A Data Hazard is any condition in which either the source or the 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.



* For example, stage E in the four-stage pipeline of responsible for arithmetic and logic operations and one clock cycle is assigned for this task.
* The operation specified instruction I2 requires three cycle to complete, from cycle 4 through cycle 6. Thus, in cycle 5 and 6, the write stage must be told to do nothing, because it has no data to work with. The information in buffer B2 must remain intact until the execute stage has completed its operation.
* This means that stage 2 and in turn, stage 1 are blocked from accepting new instructions because the information in B1 cannot be overwritten. Thus, steps D4 and F5 must be postponed.





Consider the pipeline data dependency just described arises when the destination of one instruction is used as a source in the next instruction. For example, the two instructions give rise to a data dependency.

Mul R2, R3, R4

Add R5, R4, R6

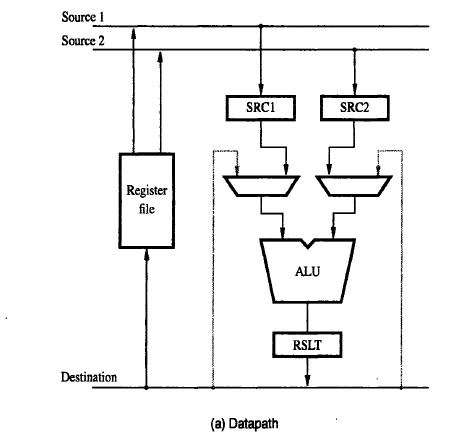
The result of the multiply instruction is placed into register R4, which in turn is one of the two source operands of the Add instruction.

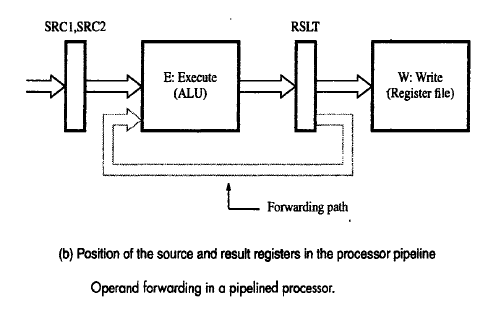
As the Decode unit decodes the Add instruction in cycle3, it realizes that R4 is used as a source operand. Hence, the D step of that instruction cannot be completed until the W step of the multiply instruction has been completed. Completion of step D2 must be delayed to clock cycle 5, and is shown step D2A. Instruction I3 is fetched in cycle3, but its decoding must be delayed because step D3 cannot precede D2. Hence, pipelined execution is stalled for two cycles.

**Operand Forwarding:**

The data hazard just described arises because one instruction, Instruction I2 is waiting for data to be written in the register file. However, these data are available at the output of the ALU once the Execute stage completes step E1. Hence, the delay can be reduced, or possibly eliminated, if we arrange for the result of instruction I1 to be forwarded directly for use in step E2.

It shows a part of the processor data path involving the ALU and the register file. This arrangement is similar to the three bus structure, except that registers SRC1, SRC2 and RSLT have been added. These registers constitute the interstage buffers needed for pipelined operation. With reference registers SRC1 and SRC2 are part of buffer B2 and RSLT is part of B3. The two multiplexers connected at the inputs to the ALU allow the data on the destination bus to be selected instead of the contents of either the SRC1 or SRC2 register.

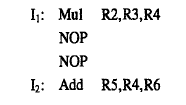


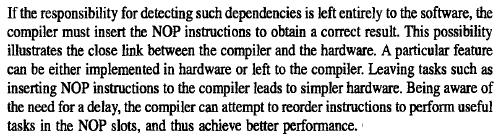


The operations performed in each clock cycle are as follows. After decoding instruction I2 and detecting the data dependency, a decision is made to use data forwarding. The operand not involved in the dependency, register R2 is read and loaded in register SRC1 in clock cycle3. In the next clock cycle, the product produced by instruction I1 is available in register RSLT and because of the forwarding connection; it can be used in step E2. Hence, execution of I2 proceeds without interruption.

**Handling Data Hazards in Software:**

An alternative approach is to leave the task of detecting data dependencies and dealing with them to the software. In this case, the compiler can introduce the two cycle delay needed between instruction I1 and I2 by inserting NOP(No-operation) instructions

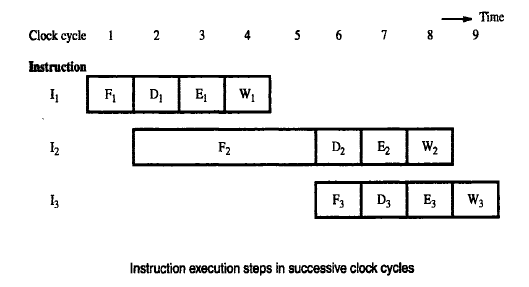




**INSTRUCTION HAZARDS:**

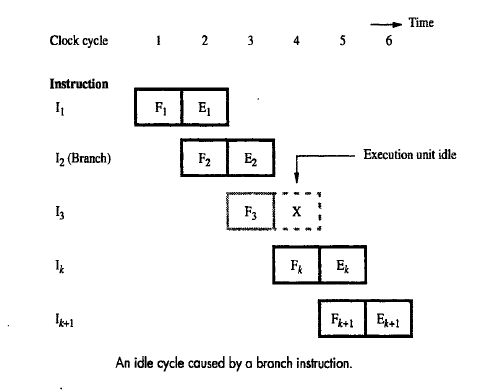
The pipeline may also be stalled because of a delay in the availability of an instruction. In this instruction may be a result of a miss in the cache, requiring the instruction to be fetched from the main memory. Such hazards are often called Control Hazards or Instruction Hazards.

* The effect of a cache miss on pipelined operation is illustrated in the figure.
* Instruction I1 is fetched from the cache in cycle1, and its execution proceeds normally. However, the fetch operation for instruction I2, which is started in cycle2, results in a cache miss.
* The instruction fetch unit must now suspend any further fetch requests and wait for I2 to arrive.
* Instruction I2 received and loaded into buffer B1 at the end of cycle5. The pipeline resumes its normal operation at that point.



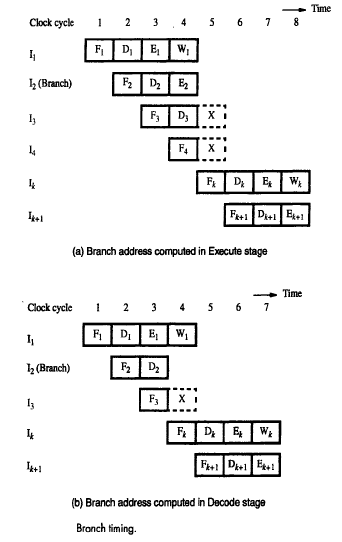
**UNCONDITIONAL BRANCH:**

* It shows a sequence of instructions being executed in a two-stage pipeline.
* Instructions I1 to I3 are stored at successive memory addresses, and I2 is a branch instruction. Let the branch target be instruction Ik.
* In clock cycle3, the fetch operation for instruction I3 is in progress at the same time that the branch instruction is being decoded and the target address computed.
* In clock cycle4, the processor must discard I3, which has been incorrectly fetched, and fetch instruction Ik.
* In the meantime, the hardware unit responsible for the Execute (E) step must be told to do nothing during that clock period. Thus, the pipeline is stalled for one clock cycle.
* The time lost as a result of a branch instruction is often referred to as the branch penalty. The branch penalty is one clock cycle.
* For a longer pipeline, the branch penalty may be higher.



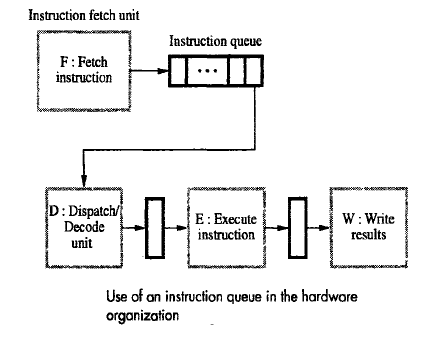
**Four stage pipeline:**

* The effect of a branch instruction on a four stage pipeline, the branch address is computed in step E2.
* Instruction I3 and I4 must be discarded, and the target instruction Ik is fetched in clock cycle5. Thus, the branch penalty is two clock cycles.
* Reducing the branch penalty requires the branch address to be computed earlier in the pipeline.
* Typically, the instruction fetch unit has dedicated hardware to identify a branch instruction and compute the branch target address as quickly as possible after an instruction is fetched.
* With this additional hardware, both of these tasks can be performed in step D2. In this case, the branch penalty is only one clock cycle.



**Instruction Queue and Pre fetching:**

* Either a Cache miss or a branch instruction stalls the pipeline for one or more clock cycles.
* To reduce the effect of these interruptions, many processors employ sophisticated fetch units that can fetch instructions before they are needed and put them in a queue.
* Typically, the instruction queue can store several instructions.
* A separate unit is dispatch unit, takes instructions from the front of the queue and sends them to the execution unit. The dispatch unit also performs the decoding function.



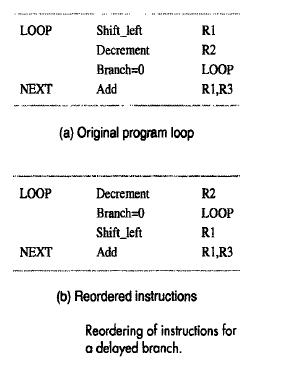
* The fetch unit must have sufficient decoding and processing capability to recognize and execute branch instructions.
* It attempts to keep the instruction queue filled at all times to reduce the impact of occasional delays when fetching instructions.
* However, the fetch unit continues to fetch instructions because of a branch or a cache miss, the dispatch unit continues to issue instructions from the instruction queue.

**CONDITIONAL BRANCHS AND BRANCH PRIDECTION:**

* A conditional branch instruction introduces the added hazard caused by the dependency of the branch condition on the result of a preceding instruction.
* Because of the branch penalty, this large percentage would reduce the gain in performance expected from pipelining.
* Fortunately, branch instructions can be handled in several ways to reduce their negative impact on the rate of execution of instructions.

Delayed Branch:

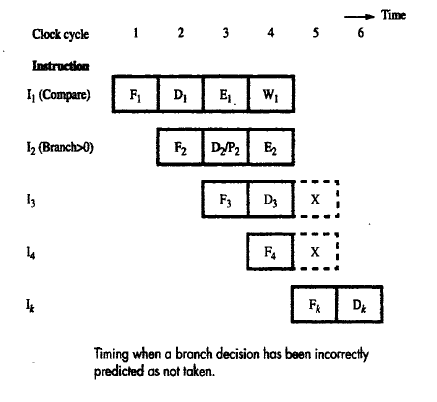
* The processor fetches instruction I3 before it determines whether the current instruction I2 is a branch instruction.
* When execution of I2 is completed and a branch is to be made, the processor must discard I3 and fetch the instruction at the branch target.
* The location following a branch instruction is called a branch delay slot. There may be more than one branch delay slot, depending on the time it takes to execute a branch instruction.
* Consider the instruction sequence given; Register R2 is used as a counter to determine the number of times the contents of register R1 are shifted left.
* For a processor with one delay slot, the instructions can be reordered.



* The shift instruction is fetched while the branch instruction is being executed.
* After evaluating the branch condition, the processor fetches the instruction at LOOP or at NEXT, depending on whether the branch condition is true or false, respectively.
* Pipelined operation is not interrupted at any time, and there are no idle cycles.
* Logically, the program is executed as if the branch instruction were placed after the shift instruction.

**Branch Prediction:**

* Another technique for reducing the branch penalty associated with conditional branches is to attempt to predict whether or not a particular branch will be taken.
* Until the branch condition is evaluated, instruction execution along the predicted path must be done on a speculative basis.
* Speculative execution means that instructions are executed before the processor is certain that they are in the correct execution sequence.
* For a four stage pipeline compare instruction followed by a Branch > 0 instruction.



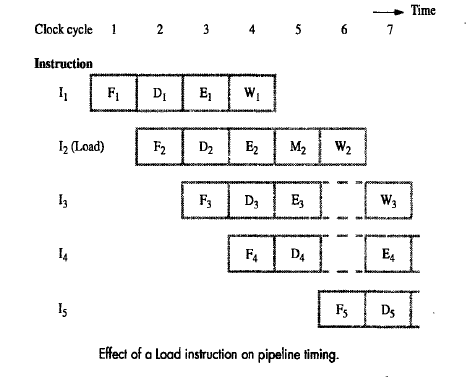
* Branch prediction takes place in cycle3, while instruction I3 is being fetched. The fetch unit predicts that the branch will not be taken, and it continues to fetch instruction I4 as I3 enters the decode stage.
* The results of the compare operation are available at the end of cycle3
* Assuming that they are forwarded immediately to the instruction fetch unit, the branch condition is evaluated in cycle4.
* At this point, the instruction fetch unit realizes that the prediction was incorrect, and the two instructions in the execution pipe are purged.

**STRUCTURAL HAZARDS:**

The Pipelined operation is the situation when two instructions require the use of a given hardware resource at the same time is known as Structural Hazard. The most common case in which this hazard may arise is in access to memory.

* One instruction may need to access memory as part of the Execute or Write stage while another instruction is being fetched.
* If instructions and data reside in the same cache unit, only one instruction can proceed and the other instruction is delayed.
* Many processors use separate instruction and data caches to avoid this delay.
* An example of a structural hazard is

Load X(R1), R2



* In can be accommodated in our example 4 stage pipeline, the memory address X + [R1] is computed in step E2 in cycle4, and then memory access takes place in cycle5.
* The operand read from memory is written into register R2 in cycle6. This means that the execution step of this instruction takes two clock cycles( cycles 4 and 5).
* It causes the pipeline to stall for one cycle, because both instructions I2 and I3 require access to the register file in cycle6.
* Even though the instructions and their data are all available, the pipeline is stalled because one hardware resource, the register file cannot handle two operations at once.
* If register file had two input ports, that is it allowed two simultaneous write operations, the pipeline would not be stalled.
* In general, structural hazards are avoided by providing sufficient hardware resources on the processor chip.

**INFLUENCE ON INSTRUCTION SETS:**

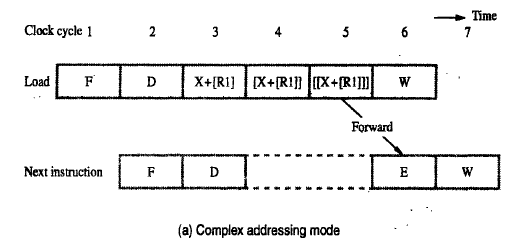
Some instructions are much better suited to pipeline execution. For example, instruction side effects can lead to undesirable data dependencies. In this section, the relationship between pipelined execution and machine instruction features. It is discuss two key aspects of machine instructions are Addressing modes and Condition code flags.

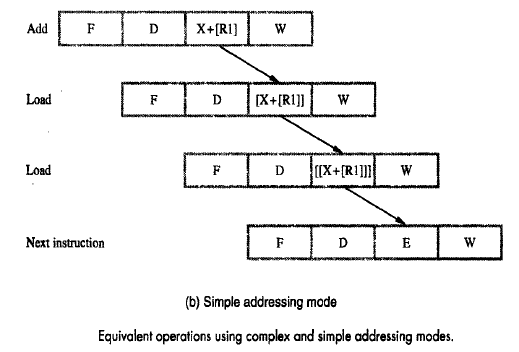
**Addressing Modes:**

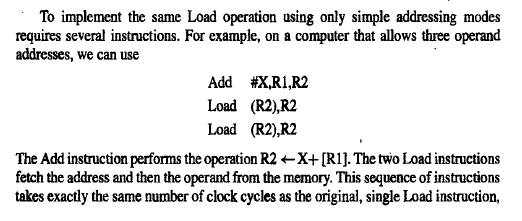
* Addressing modes should provide the means for accessing a variety of data structures simply and efficiently.
* Useful addressing modes include index, indirect, auto increment, and auto decrement.
* Many processors provide various combinations of these modes to increase the flexibility of their instruction sets.
* Complex addressing modes, such as those involving double indexing
* The addressing modes to be implemented in a pipelined processor, the effect of each addressing mode in instruction flow in the pipeline.
* Two important considerations in this regard are the side effects of modes such as auto increment and auto decrement which complex addressing modes cause the pipeline to stall.
* Another important factor is whether a given mode is likely to be used by compilers.
* The Load instruction Load X(R1), R2 takes five cycles to complete execution.
* The Instruction Load (R1), R2 can be organized to fit a four-stage pipeline because no address computation is required. Access to memory can take place in stage E.
* A more complex addressing mode may require several accesses to the memory to reach the named operand.

Load (X(R1)),R2

* The instruction may be executed assuming that the index offset, X is given in the instruction word.
* After computing the address in cycle3, the processor needs to access memory twice – first to read location X + [R1] in clock cycle4 and then to read location [ X + [R1]] in cycle 5. If R2 is a source operand in the next instruction, that instruction would be stalled for three cycles, which can be reduced to two cycles with operand forwarding.

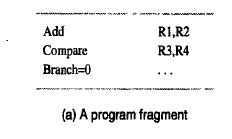




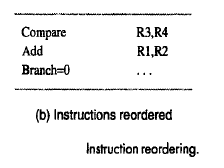


**Addressing Modes:**

* The condition code flags are stored in the processor status register.
* They are either set or cleared by many instructions, so that they can be tested by subsequent conditional branch instructions to change the flow of program execution.
* An optimizing compiler for a pipelined processor attempts to reorder instructions to avoid stalling the pipeline when branches or data dependencies b/w successive instructions occur.
* The dependency introduced by the condition-code flags reduces the flexibility available for the compiler to reorder instructions.



* The sequence of instructions and assume that the execution of the Compare and Branch=0 instructions proceeds.
* The branch decision takes place in step E2 rather tan D2 because it must await the result of the Compare instruction.



* These observations lead to two important conclusions about the way condition codes should be handled.
* First to provide flexibility in reordering instructions, the condition-code flags should be affected by as few instructions as possible.
* Second the compiler should be able to specify in which instructions of program the condition codes are affected.
* The instructions reordered assuming that the condition code flags are affected only when this is explicitly stated as part of the instruction OP code.