Pipelining

Pipelining

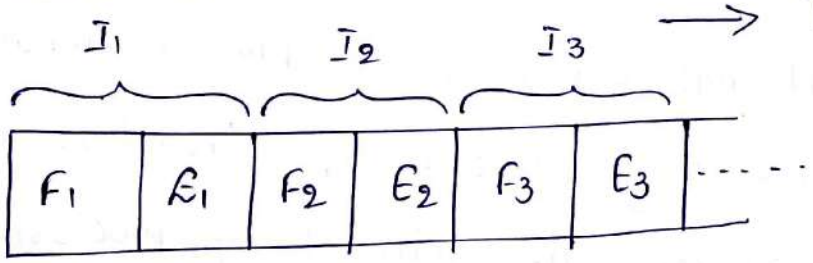
Basic Concepts, Data Hazards. Instruction Hazards, Influence on instructionsets

Basic Concepts:-

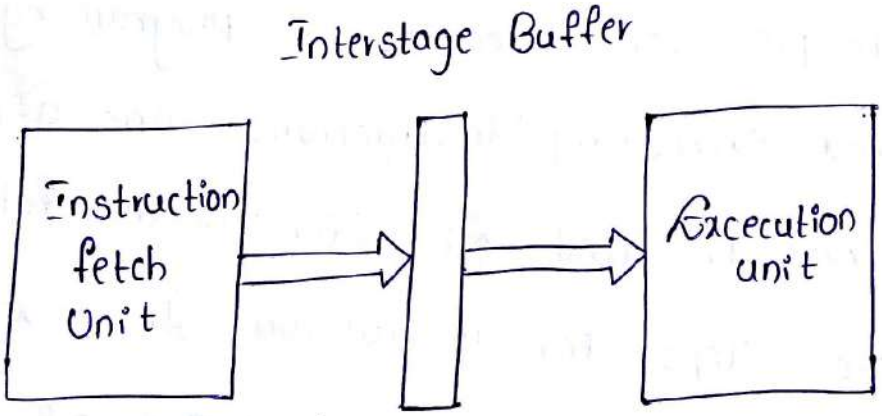
The speed of execution of programs in influenced by many factors. One way to improve performance is to use faster circuit technology to build the processor and the main memory. Another possibility is to arrange the hardware so that more than one operation can be performed at the same time. DMA devices transfers and Computational activities to proceed Simultaneously possible because they perform I/O transfers independently Once these transfers are initiated by processor.

Pipelining is Commonly known as assembly-line Operation. 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 instructions Ii. Execution of a program Consists of a Sequence of fetch and execute steps.

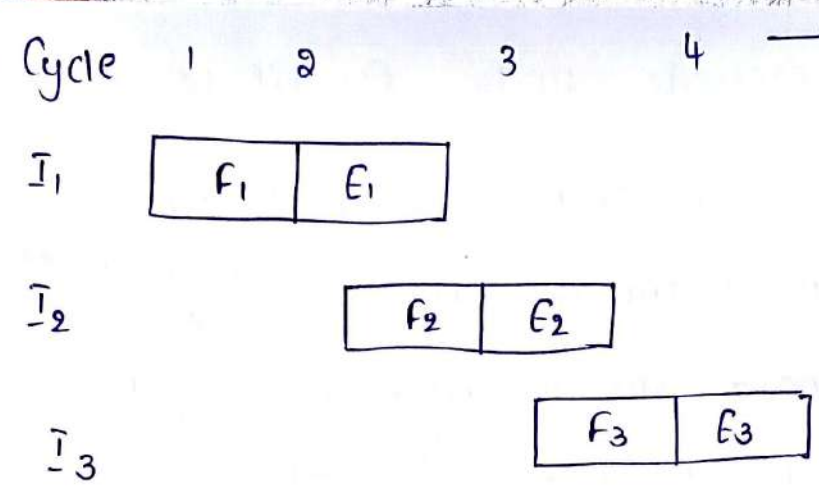
The Computer that has two seperate hardware units, One for fetching instructions and another for excecting them. The instruction fetched by the fetch unit is deposited in an intermediate storage Buffer Bi. This buffer is needed to enable the excecution unit to excecute the instruction while the fetch unit is fetching the next instruction. The results of excecution are deposited in the destination location Specified by the instruction. The both source and destination of the data Operated on by the instruction are inside the block labeled " Excecution Unit".



(a) Sequential Excecution



(b) Hardware Organization.



(c) Pipelined Execution.

Fig: Basic idea of instruction pipelining.

The Computer is Controlled by a clock whose period is Such that fetch and excecute steps of any instruction can each be Completed in One cycle. In the first clock cycle, the fetch unit fetches an instruction I_1 [step F_1] and stores it in buffer B_1 at the end of the clock cycle. In the second clock cycle, the instruction fetch unit proceeds with the fetch Operation for instruction I_2 [step F_2]. The excecution unit performed the Operation Specifie by instruction I_1 which is available to in in buffer B_1 [Step E_1]. By the end of the second clock cycle, the excecution of instruction I_1 is Completed and instruction I_2 is available.

Fetch and execute units constitute a two-stage pipeline in which each stage performs one step in processing an instruction. An interstage buffer B_1 is needed to hold the information being passed from one stage to the next. New information is loaded into this buffer at the end of each clock cycle.

The processing of an information need not be divided into only two steps. For eg: a pipelined processor may process each instruction in 4 steps as follows:

F : Fetch :- read the instruction from memory

D : Decode - decode the instruction and fetch the Source Operand(s)

E : Execute :- perform the Operation Specified by the instruction

W : write :- Store the result in the destination location.

This means that four distinct hardwares units are needed. These units may must be capable of performing their task simultaneously and without interfering with one another.

An instruction progresses through the pipeline, all the information needed by the stages downstream must be passed along.

For Example:

During Clock cycle 4, the information in the buffer is as follows

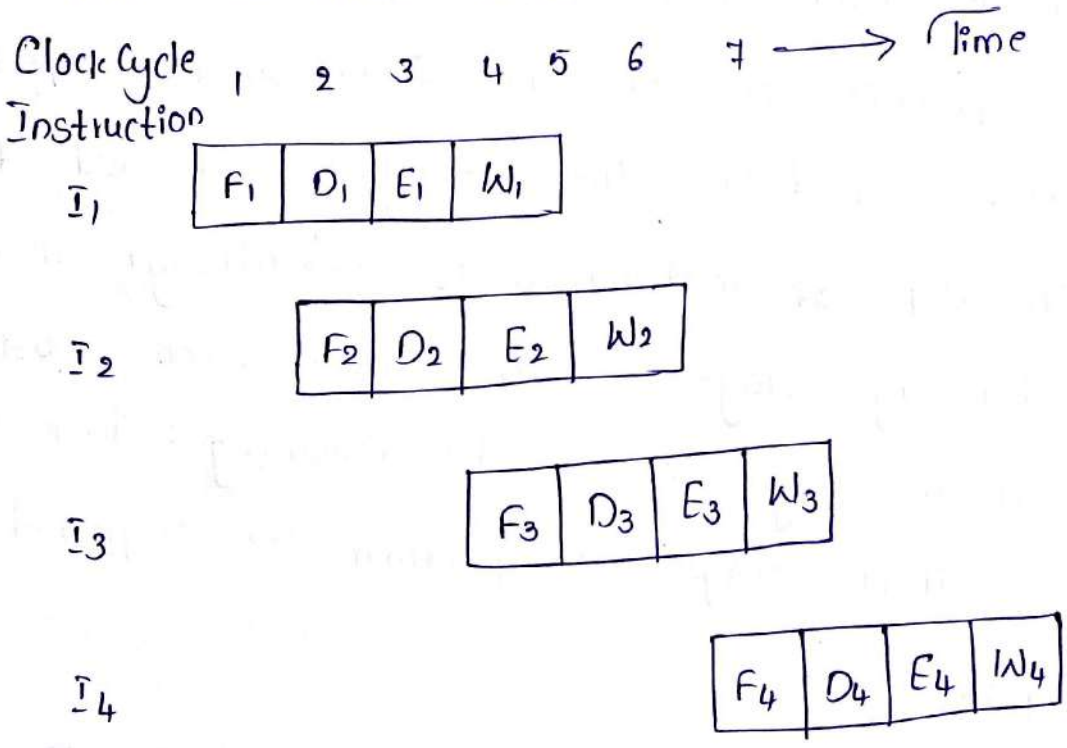
(i) Buffer B1 holds instruction I_3 which was fetched in cycle 3 and is being decoded by the instruction decoding unit.

(ii) Buffer B2 holds both Source Operands for instruction I_2 and Specification of the Operation to be performed The buffer also holds the information needed for the write step of instruction I_2. Eventhough it is not needed by Stage E, this information must be passed on to Stage W in the following Clock Cycle to enable that Stage to perform the required write Operation.

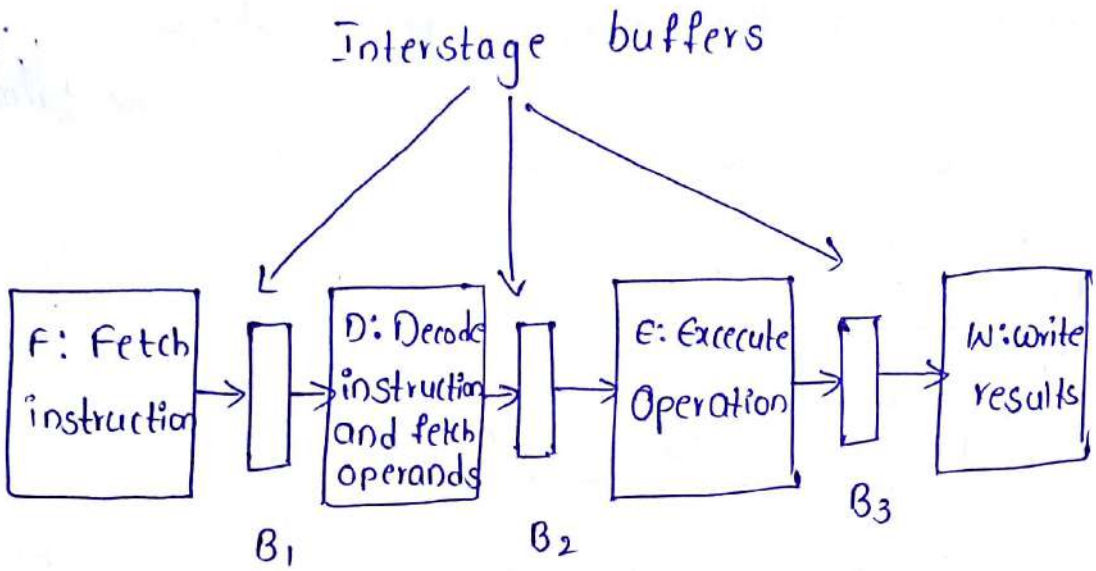
(iii) Buffer B_3 holds the results produced by the execution unit and the destination information for instruction I_1.

Role of Cache Memory:-

Each Stage in a pipeline is expected to Complete its Operation in one Clock cycle. Hence, the Clock period should be Sufficiently long to Complete the task being performed in any stage. 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 task being performed in different stages require about the Same amount of time.



(a) Instruction excecution divided into four steps.



(b) Hardware Organization

fig: A 4-stage Pipeline.

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.

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 no. of pipeline stages.

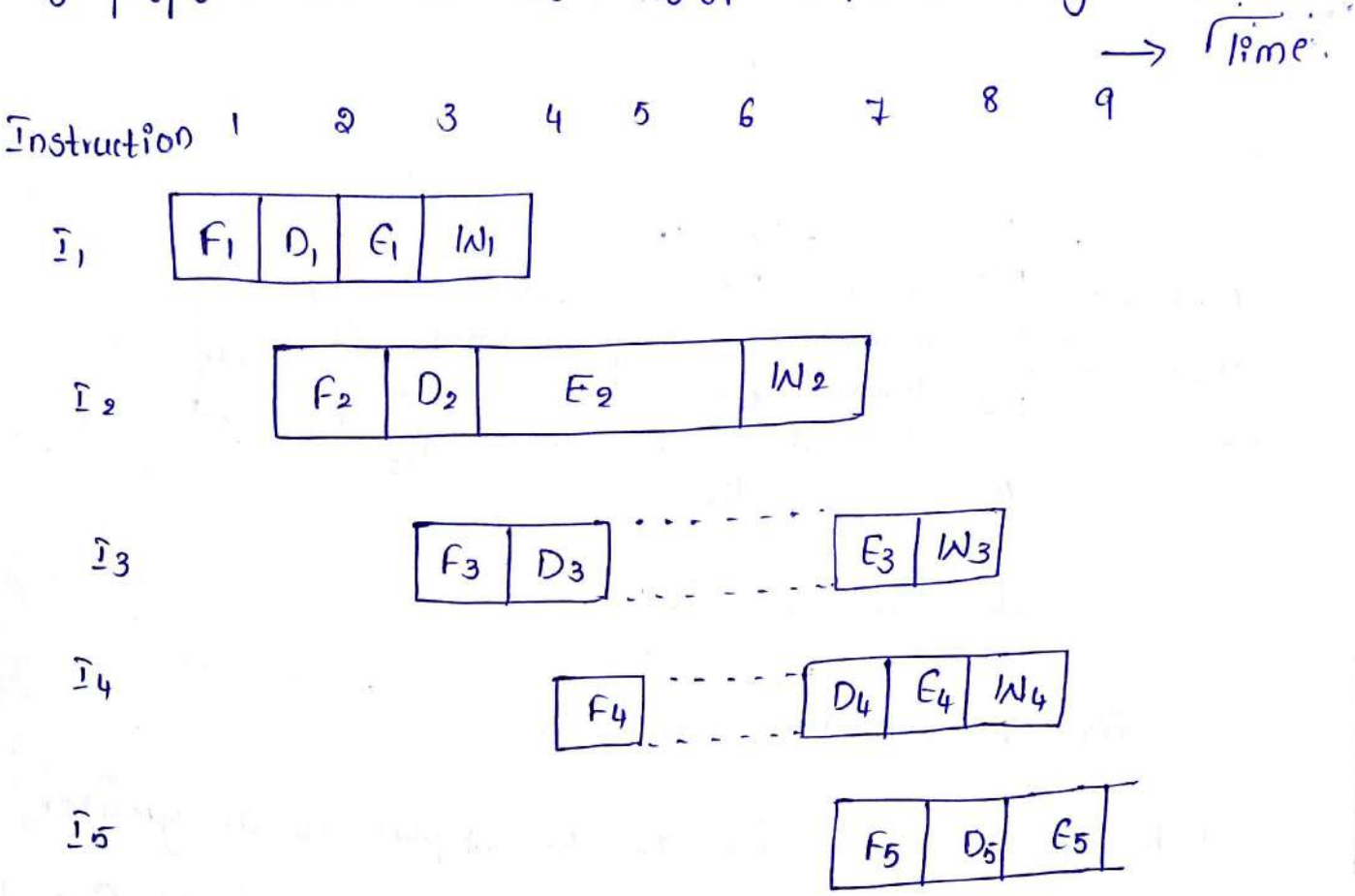
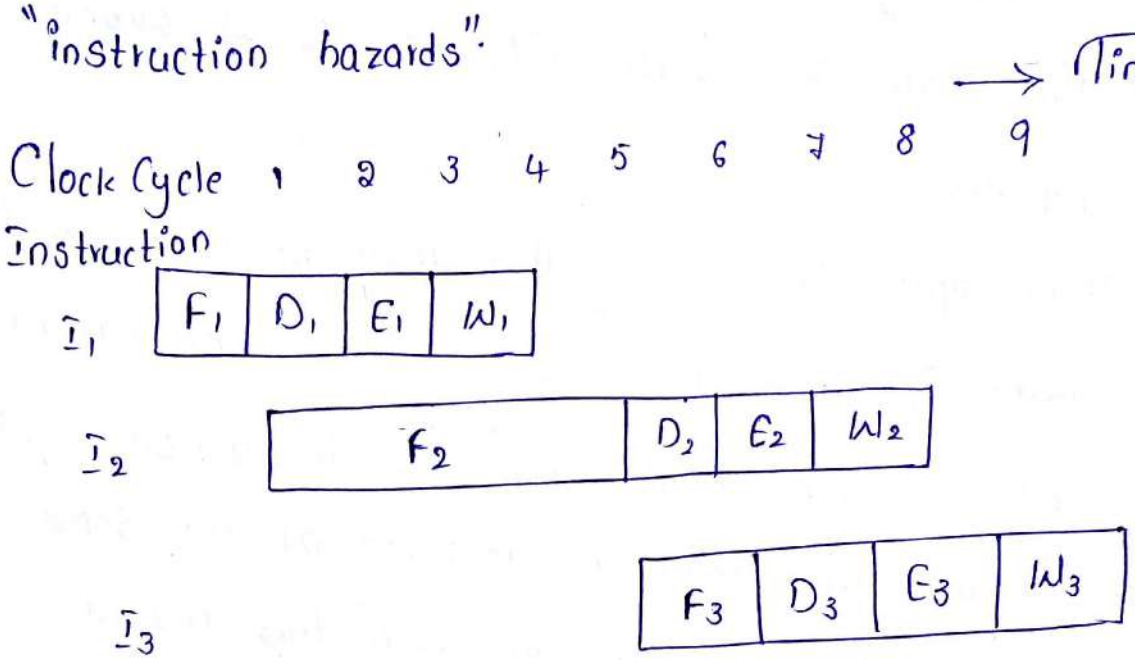


fig: Effect of an execution operation taking more than one Clock cycle.

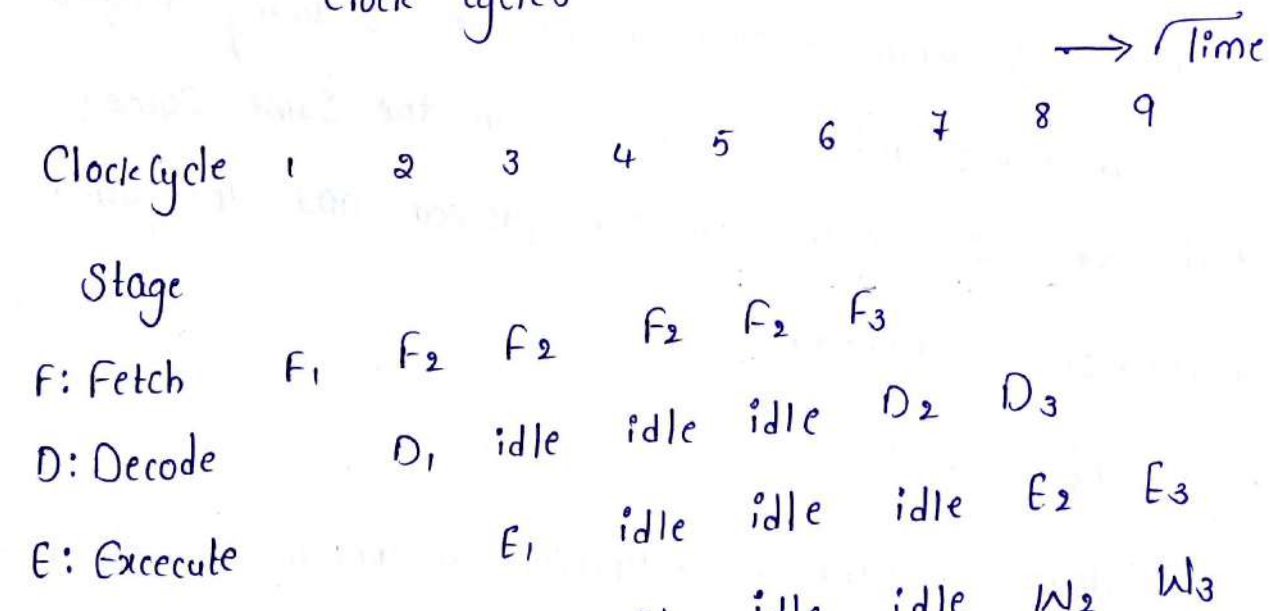
Pipelined Operation [in above fig] is said to have been stalled for two Clock cycles. Any Condition that Causes the pipeline to stall is Called "hazard". A data hazard is any Condition in which either the source or the destination Operands of an instruction are not available at the time expected in the pipe line.

The pipeline may also be stalled because of a

delay in the availability of an instruction. For example, this may be 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".



(a) Instruction execution steps in Successive Clock cycles



(b) Function performed by each processor stage in Successive Clock cycles

An alternative representation of the operation of a pipeline in the case of a cache miss. The figure gives the function performed by each pipeline stage in each clock cycle. The idle periods are called "stalls". They are also often reffered to as "bubbles" in the pipeline.

A third type of hazard that may be encountered in pipelined operation is known as "Structural hazard". This is the situation occurs when two instruction requires the use of a given hardware resource at the same time. The most common case in which this hazard may arise in access to memory. One instruction may need to access memory as part of the Excecute 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.

Data Hazards :-

A data hazard is situation in which the pipeline is stalled because the data to be operated on are delayed for some reasons.

Consider a program that contains two instructions I_1 followed by I_2. When this program is executed in a pipeline, the execution of I_2 can begin before the execution of I_1 is Completed. This means that the results generated by I_1 may not be available for use by I_2.

Assume that A=5 and Consider the following two Operations:

$$A \leftarrow 3 + A$$
$$B \leftarrow 4 \times A$$

when these operations are performed in order given, the result is B=32. If these two operations are performed by instructions in a program, then the instructions must be executed one after the Other, because the data used in the Second instruction depend on the result of the first instruction.

The Other two operations

$$A \leftarrow 5 \times C$$
$$B \leftarrow 20 + C$$

Can be performed Concurrently, because these Operations are independent.

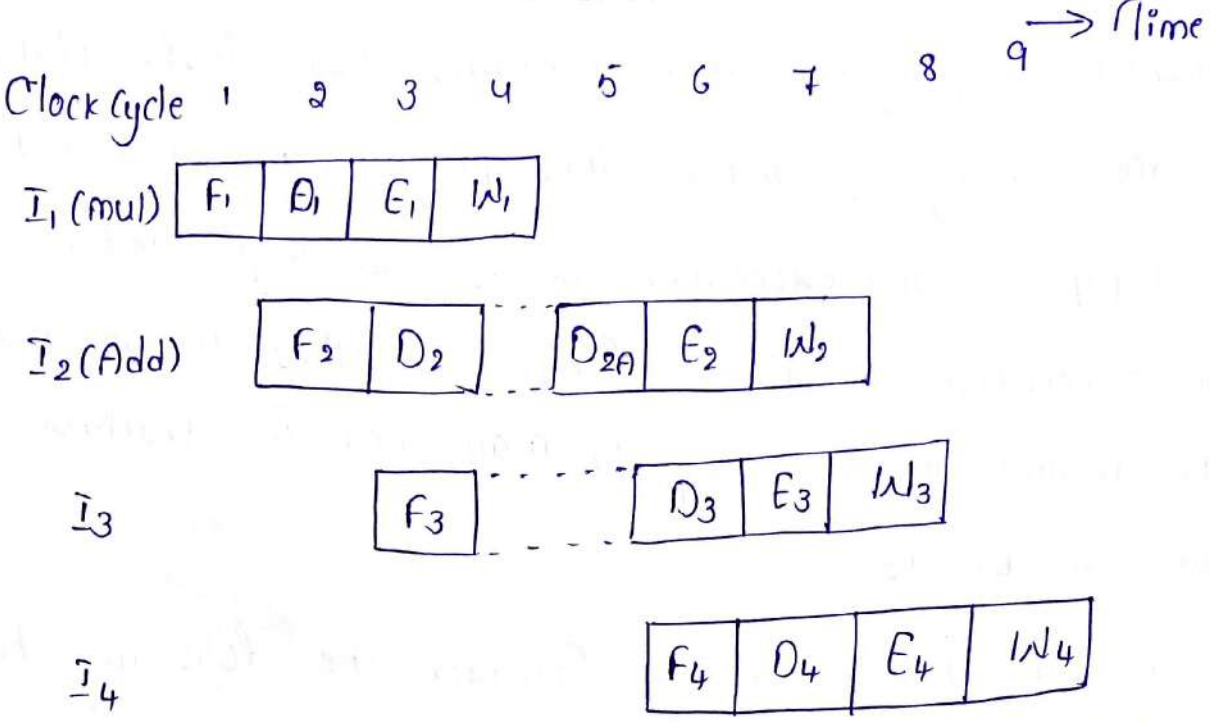


Fig: Pipeline stalled by data depency b/w D_2 & W_1

The two instructions are

Mul R_2, R_3, R_4

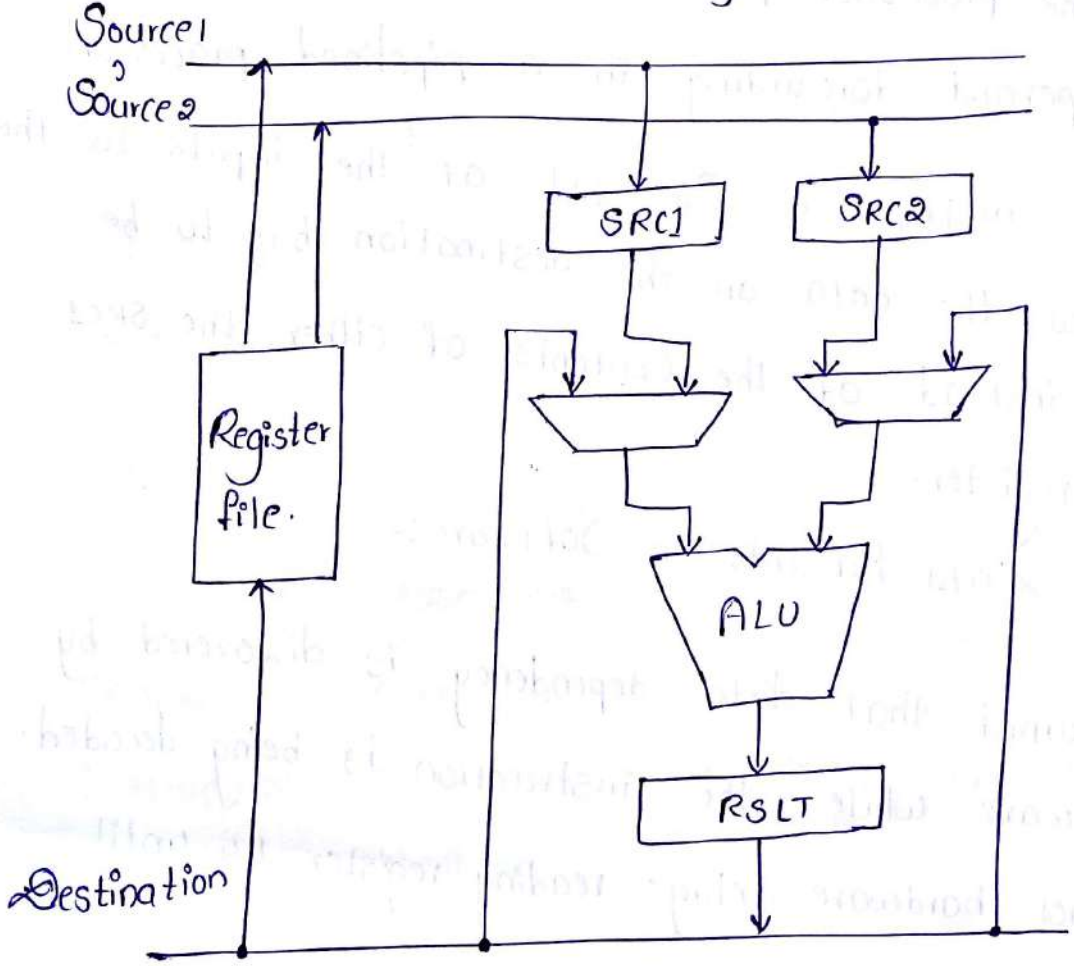
Add R_5, R_4, R_6

gives rise to data dependency. The result of multiply instruction is placed into register R_4 which in turn is one of the two source operands of the Add instruction. Assuming that the multiply operation takes one clock cycle to complete, As decode unit decodes the Add instruction in cycle 3, it realizes that R_4 is used as 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 D_2 must be delayed at clock cycle 5 i.e D_{2A}. Instruction

I_3 is fetched in Cycle 3 but its decoding must be delayed because Step D_3 cannot precede D_2. Hence, pipelined execution is stalled for two cycles.

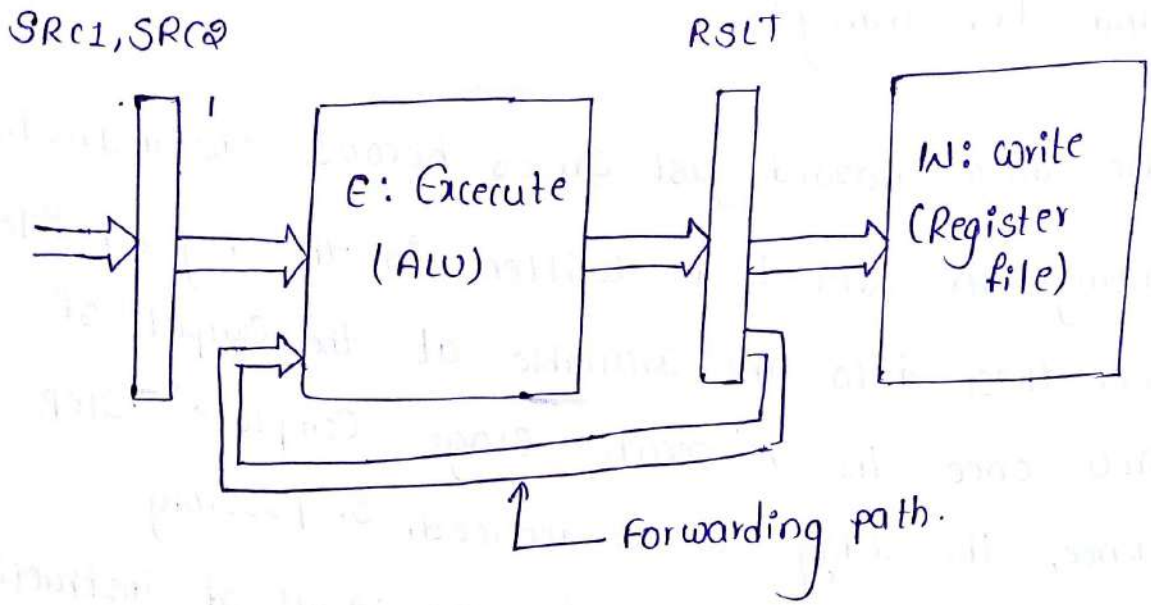
Operand Forwarding :-

The data hazard just arises because one instruction 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 E_1. Hence, the delay can be reduced, or possibly eliminated, if we arrange for the result of instruction I_1 to be forwarded directly for use in step E_2.



(a) Data path.

fig: Shows a part of the processor datapath involving the ALU and the register file. This arrangement is Similar to the three-bus Structure.



(b) position of the Source and result registers in the processor pipeline.

fig: Operand forwarding in a pipelined processor.

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.

Handling Data Hazards in Software:-

We assumed that data dependency is discovered by the hardware while the instruction is being decoded. The Control hardware delays reading register R4 until

Cycle-5, thus introducing a 2-cycle stall unless operand forwarding is used. 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 b/w instructions I_1 and I_2 by inserting NOP (No-Operation) instructions as,

```
I1: Mul   R2, R3, R4
    Nop
    Nop
I2: Add   R5, R4, R6
```

This possibility illustrates the link b/w the Compiler and the hardware. A particular feature can be either implemented in hardware or left to the Compiler. The insertion of NOP instruction leads to larger Code size.

Instruction Hazards:-

The purpose of the instruction fetch unit is to Supply the execution units with a steady stream of instructions. A branch instruction may also cause the pipeline to stall.

Unconditional Branches :-

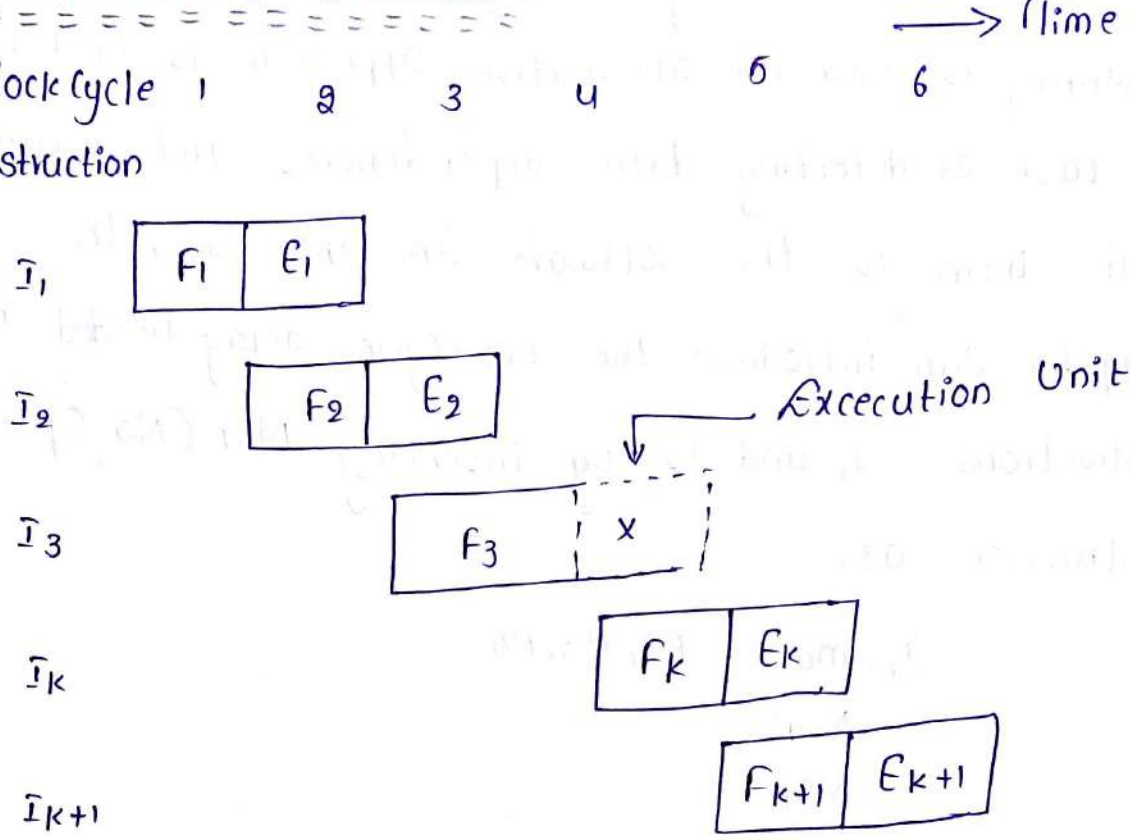
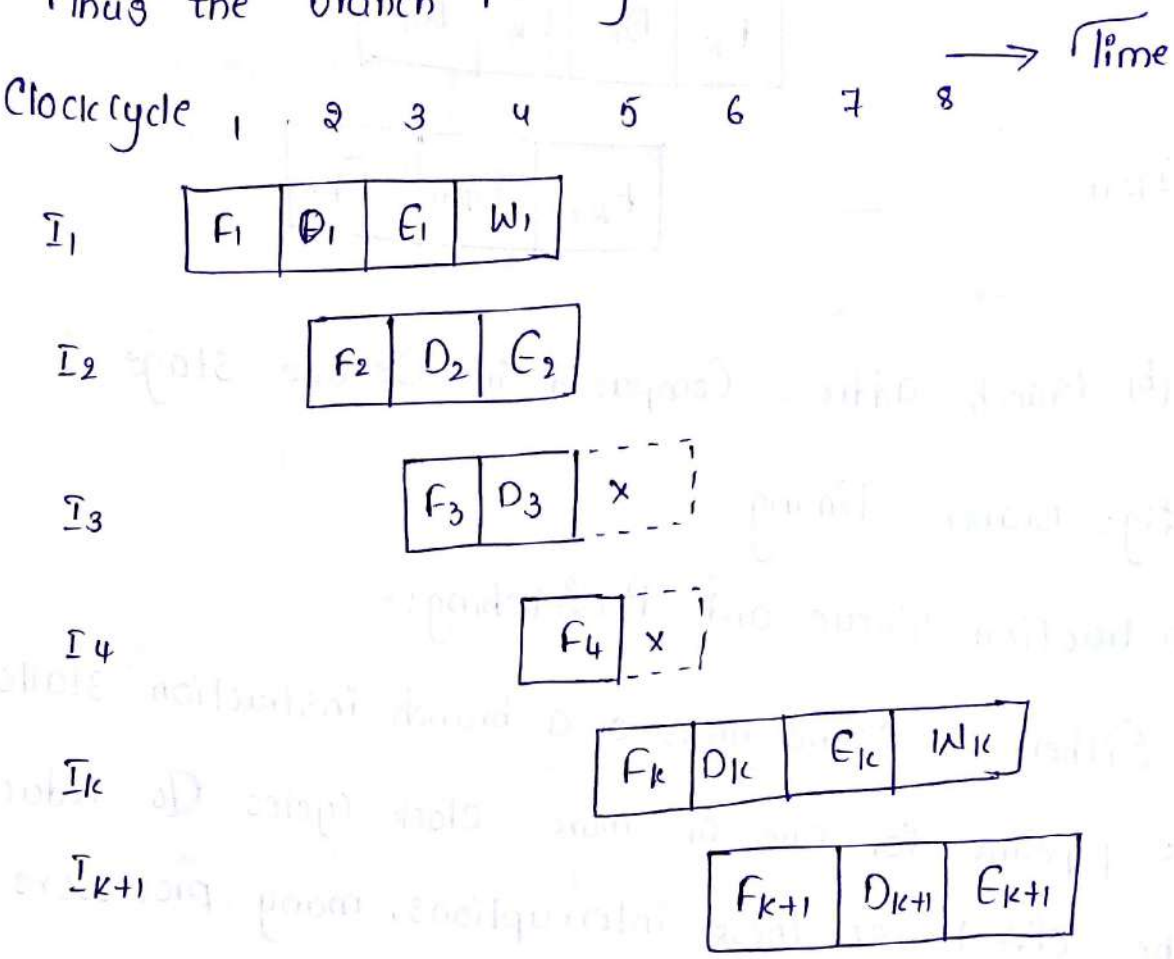


fig: An idle caused by a branch instruction.

Instructions I_1 to I_3 are stored at Succesive memory addresses, and I_2 is a branch instruction. Let the branch target be instruction I_k. In clock cycle 3 the fetch operation for instruction I_3 is in progress at the same time that the branch instruction is being decoded. In the mean time the hardware unit responsible for the Excecute (E) step must be told to do nothing during the clock period. Thus, the pipeline is stalled for one clock cycle.

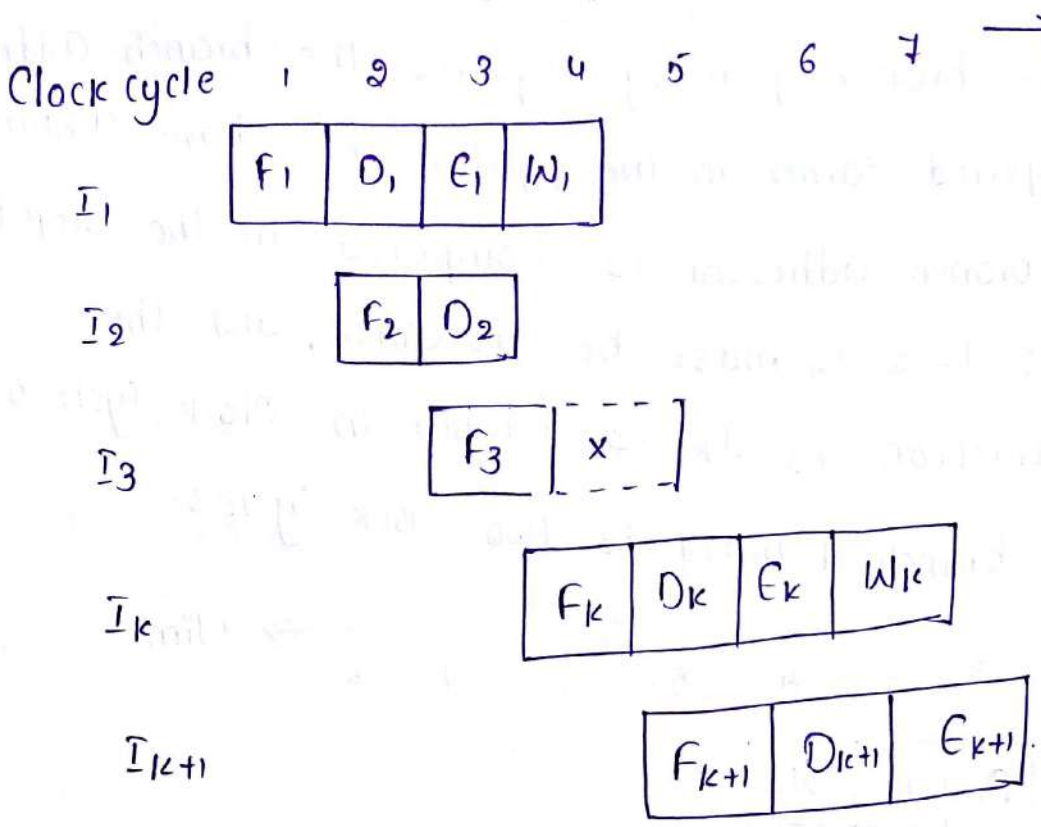
The time lost as a result of a branch instruction is often referred to as the "branch penality".

Reducing the branch penalty requires the branch address to be computed earlier in the pipeline. We have assumed that the branch addressed is computed in the step E_2. Instructions I_3 & I_4 must be discarded, and the target instruction is I_k is fetched in clock cycle 5. Thus the branch penalty is two clock cycles.



(a) Branch address Computed in Execute stage.

For a longer pipeline the branch penalty may be higher. (a) shows the effect of a branch instruction on a four-stage pipeline.



(b) Branch address Computed in Decode stage

fig: Branch timing

Instruction Queue and Prefetching:-

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 fetch units that can fetch instructions before they are needed and put them in a queue.

A seperate unit, which we call the dispatch unit, takes instructions from the front of the queue and sends them to excecution unit.

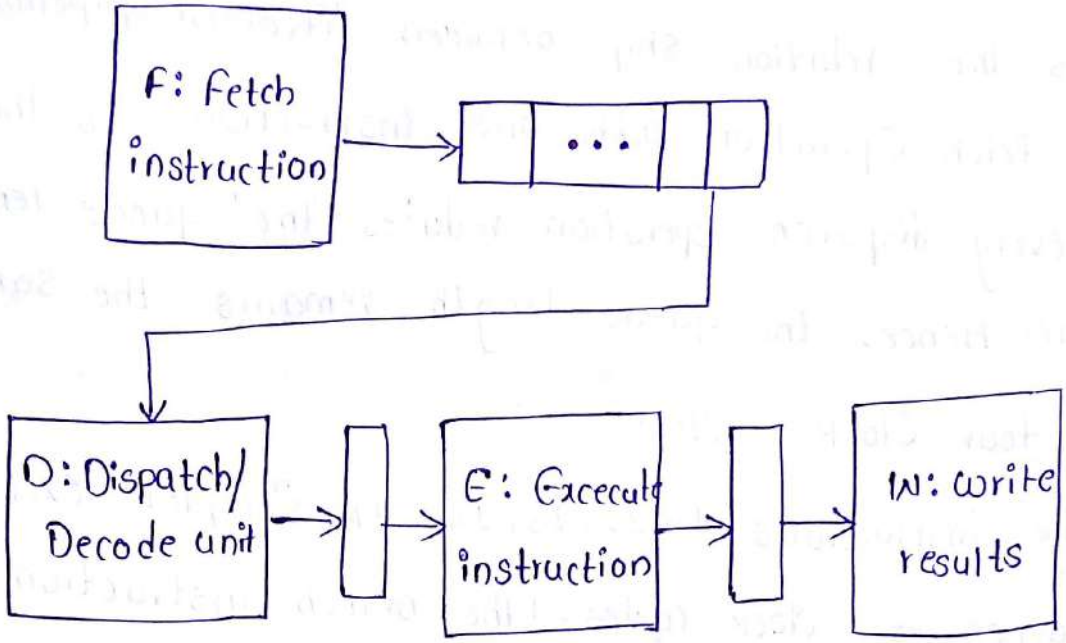


fig: Use of an instruction queue in the hardware Organization

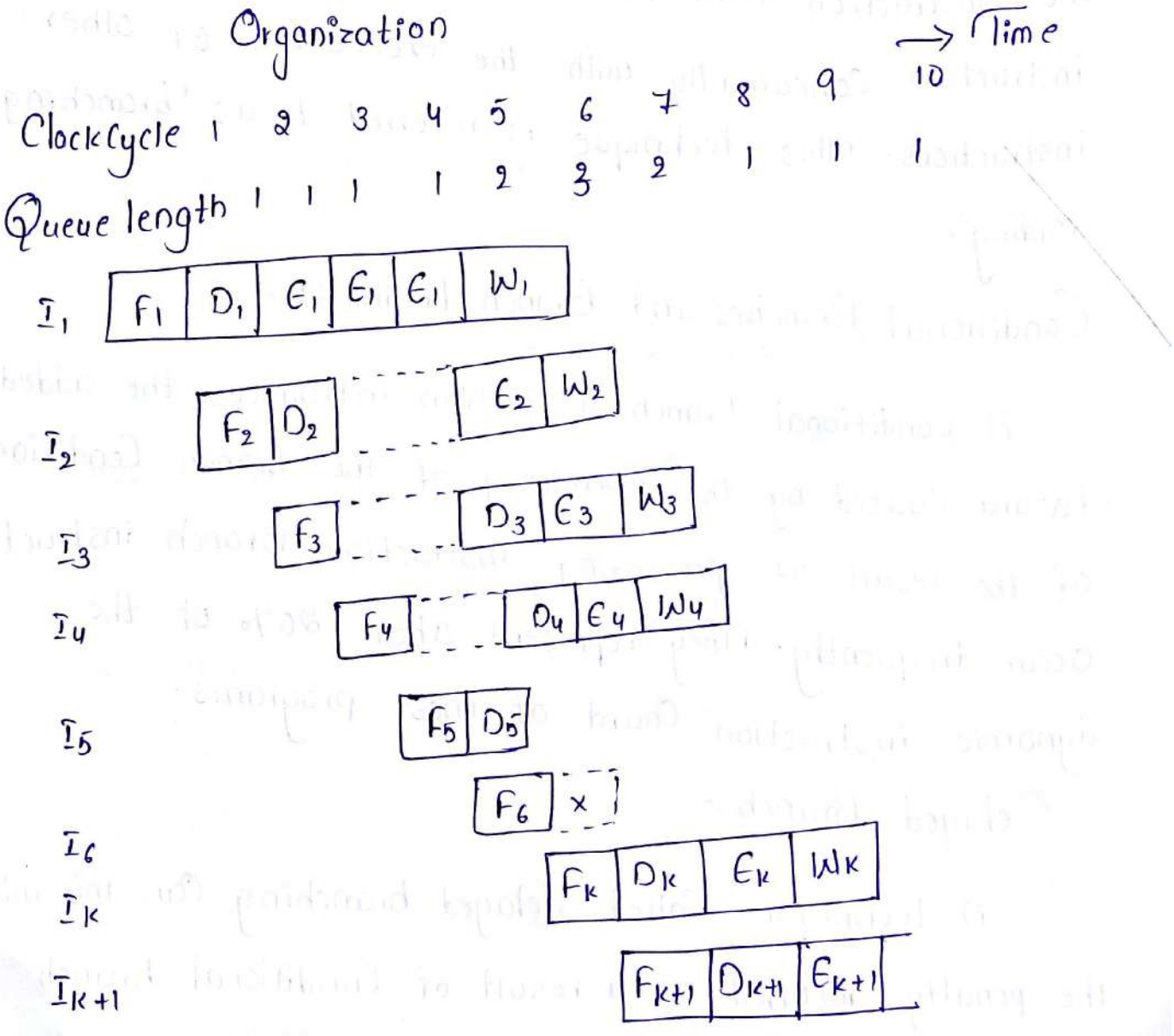


fig explains how the queue length changes and how it affects the relation ship between different pipeline stages. Every fetch Operation adds one instruction to the queue and every dispatch operation reduces the queue length by one. Hence, the queue length remains the same for first four Clock cycles.

The instructions I_1, I_2, I_3, I_4 & I_k Complete excecution in successive clock cycles. The branch instruction doesnot increase the Overall excecution time. This is because the instruction fetch unit has excecuted the branch instruction Concurrently with the excecution of other instructions. This technique is referred to as "branching folding".

Conditional Branches and Branch Prediction:-

A Conditional branch instruction introduces the added hazard caused by the dependency of the branch Condition of the result of preceeding instruction. Branch instruction occur frequently. They represent about 20% of the dynamic instruction Count of most programs.

Delayed Branch:-

A technique Called delayed branching Can minimize the penalty incurred as a result of Conditional branch

instructions. The instruction in the delay slots are always fetched.

```
LOOP    Shift_left    R1
        Decrement     R2
        Branch=0      LOOP
NEXT    Add           R1, R3
```

(a) Original program loop

```
LOOP    Decrement     R2
        Branch=0      LOOP
        Shift_left    R1
NEXT    Add           R1, R3
```

(b) Reoordered instructions

fig: Reordering of instructions for a delayed branch

Register R2 is used as a Counter to determine the no. of times the contents of register R1 are shifted left. The shift instruction is fetched while the branch instruction is being excecuted. After evaluating the branch instruction, the processor fetches the instruction at LOOP or at NEXT, depending on whether

the branch Condition is true or false.

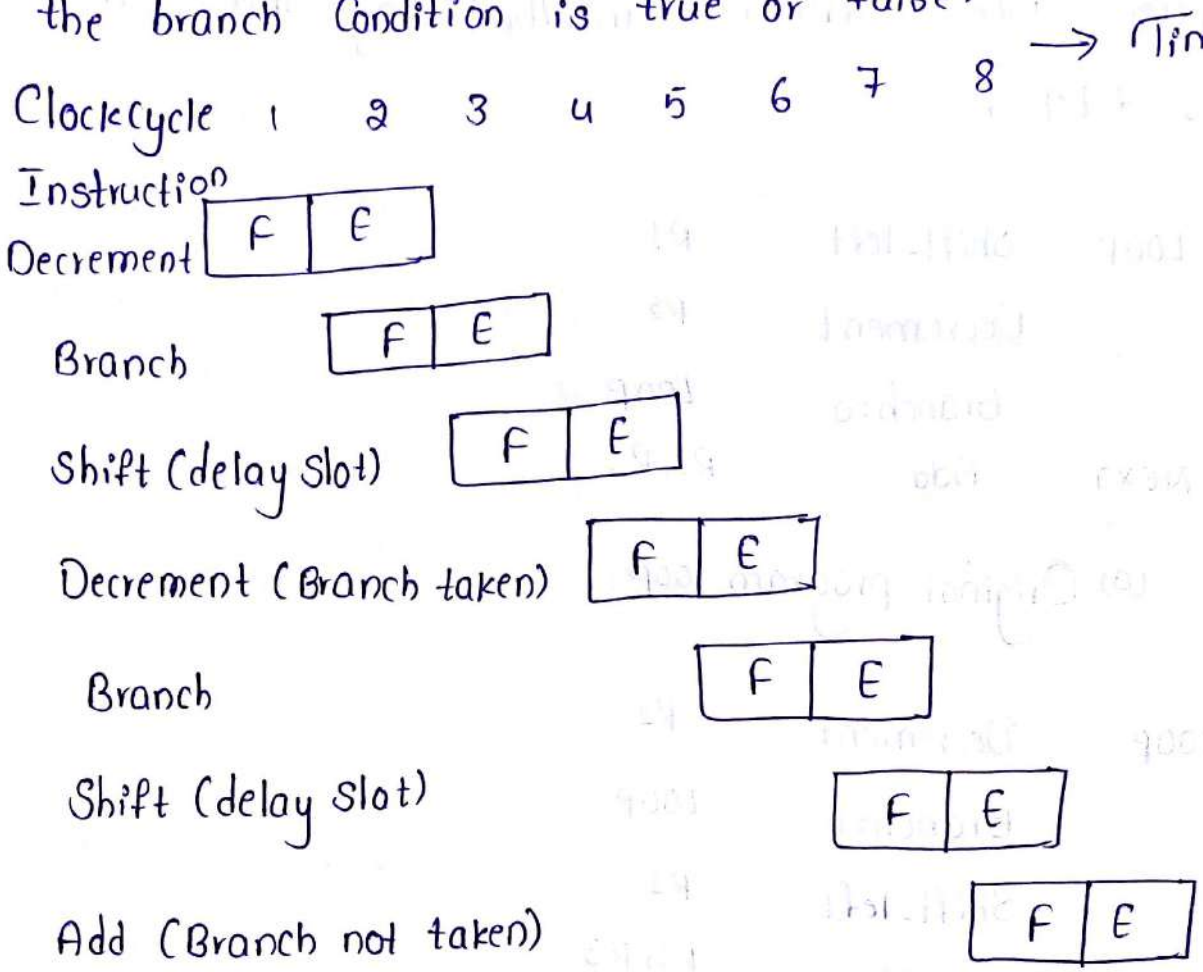


fig: Execution timing showing the delay slot being filled during the last two passes through the loop

Pipelined operation is not interrupted at any time and there are no idle cycles. Logically the program is excecuted as if the branch instruction were placed after the shift instruction. Branching takes place one instruction later than cohere the branch instruction appears in the instruction Sequence in the memory hence the name "delayed branch".

Branch Prediction:-

Speculative excecution means that instructions are excecuted before the processor is certain that they are in correct excecution sequence.

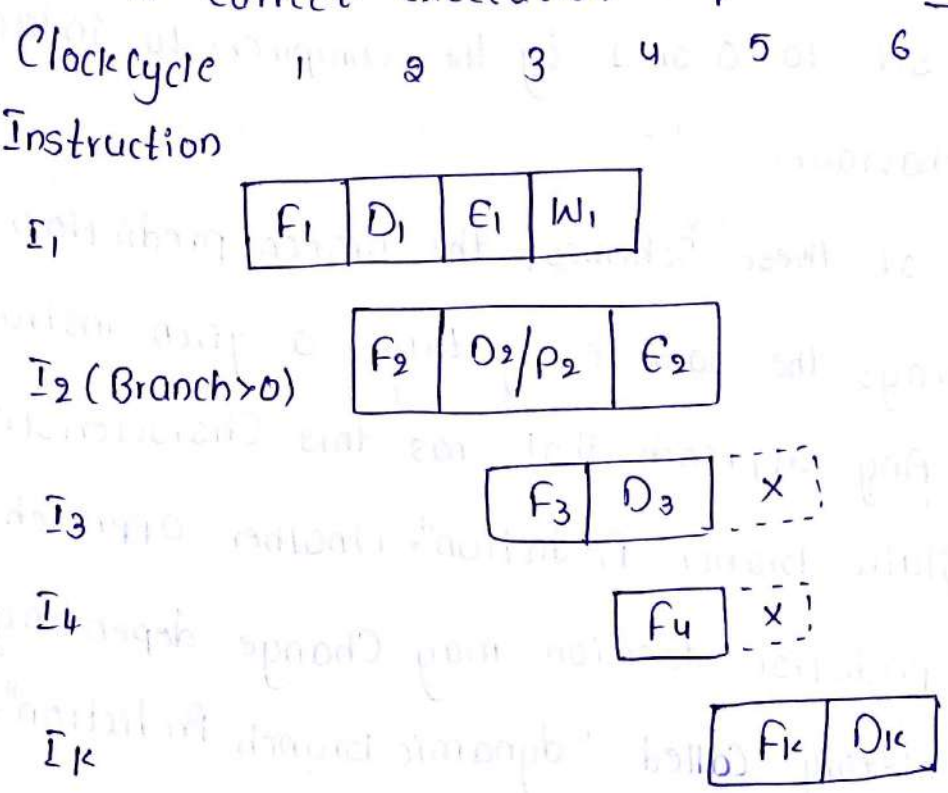


fig: Timing when a branch decision has been incorrectly predicted as not taken.

An incorrectly predicted branch is for a four-stage pipeline. Compare instruction followed by a Branch>0 instruction. Branch prediction takes place in Cycle 3. while instruction I_3 is being fetched. The fetch unit predicts that the branch will not be taken, and it Continues to fetch instruction I_4 as I_3 enters the Decode stage.

A more flexible approach is to have the Compiler decide whether a given branch instruction should be predicated taken or not taken. The branch instructions of some processors, such as SPARC, include a branch prediction bit, which is set to 0 or 1 by the Compiler to indicate the desired behaviour.

With either of these schemes, the branch prediction decision is always the same every time a given instruction is excecuted. Any approach that has this Characteristic is called "Static Branch Prediction". Another approach in which the prediction decision may Change depending on excecution history called "dynamic Branch Prediction".

Dynamic Branch Prediction:-

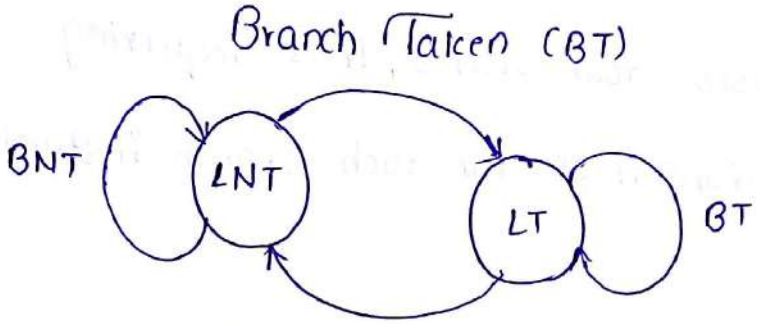
The Objective of branch prediction algorithms is to reduce the probability of making a wrong decision, to avoid fetching instructions that eventually have to be discarded.

In dynamic branch prediction schemes, the processor hardware assesses the likelihood of a given branch being taken by keeping track of branch decisions every time that instruction is excecuted.

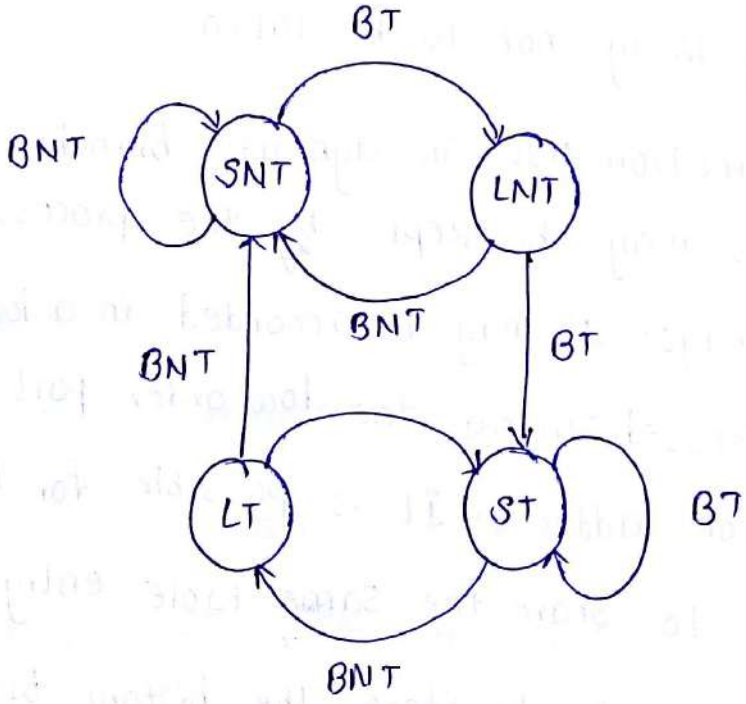
The algorithm may be described by the two-state machine in the two states are:

LT: Branch is likely to be taken

LNT: Branch is likely not to be taken.



(a) A 2-state algorithm



(b) A 4-state algorithm.

fig: State machine representation of branch-prediction algorithms

Suppose that the algorithm is started in state LNT.

When the branch instruction is excecuted and if the branch is taken, the machine moves to state LT. Otherwis it remains in state LNT, the branch is predicated as if the Corresponding state machine is in state LT.

An algorithm that uses four states, thus requiring two bits of history information for each branch instruction. The four states are:

ST: Strongly likely to be taken
LT: Likely to be taken
LNT: Likely not to be taken
SNT: Strongly likely not to be taken

The state information used in dynamic branch predication algorithms may be kept by the processor in a variety of ways. It may be recorded in a look-up table, which is accessed using the low-order part of the branch instruction address. It is possible for two branch instructions to share the same table entry. An alternative approach is to store the history bits as a tag associated with branch instructions in the instruction Cache.

Influence on Instruction Set:-

Addressing Modes:-

Addressing modes should provide the means for accessing a variety of data structures simply and efficiently. Usefull addressing modes include index, indirect, autoincrement, and autodecrement.

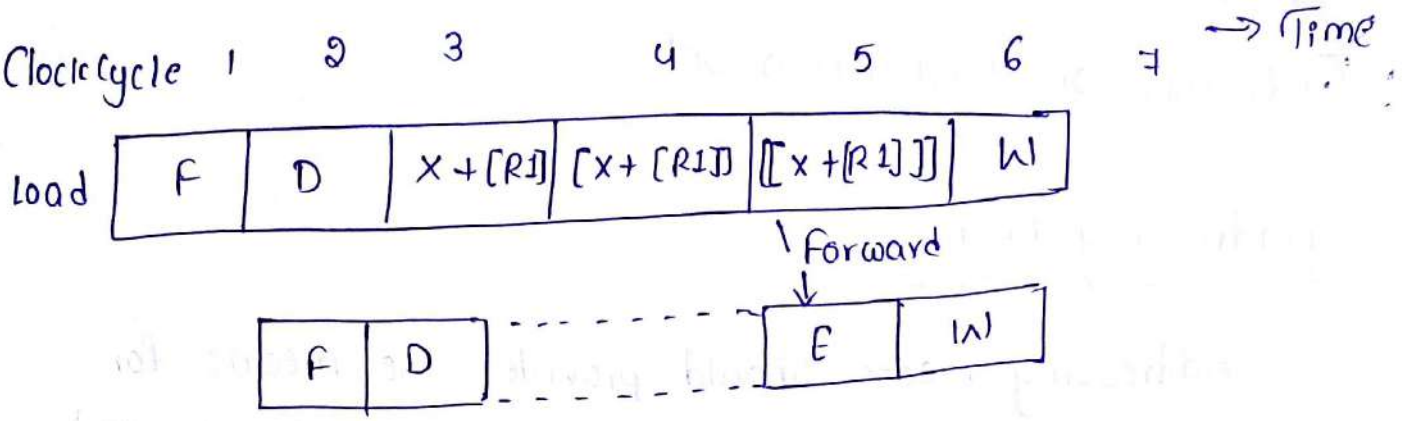
To Compare various approaches, we assume a simple model for accessing operands in the memory. The load instruction Load X(R1), R2 takes five cycles to Complete execution.

Load (R1), R2

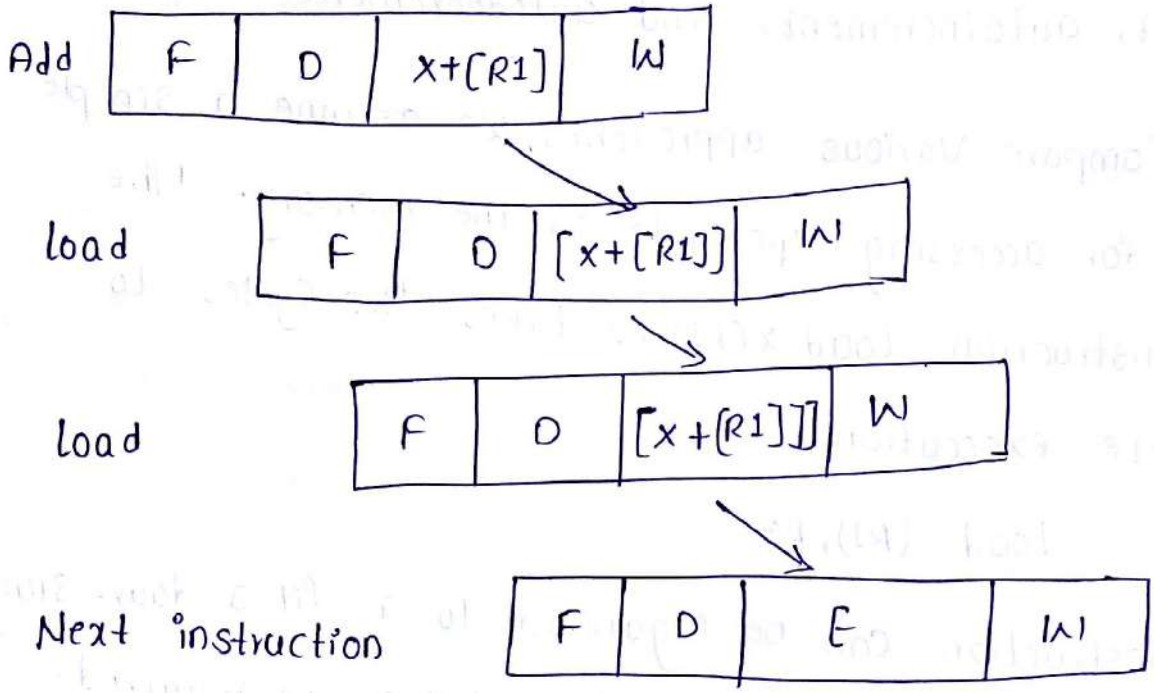
The instruction can be organized to a fit a four-stage pipeline because no address Computation is required.

Load (X(R1)), R2

X is a given in the instruction word. After Computing the address in cycle 3, the processor needs to access memory twice- first to read location X+[R1] in Clock cycle 4 and then to read location [X+[R1]] in Cycle-5 which can be reduced to two cycles with Operand forwarding.



(a) Complex addressing mode



(b) Simple addressing mode.

fig:- Equivalent Operations using Complex and Simple addressing modes

To implement the same load Operation using only simple addressing modes requires several instructions. for example,

Add #X, R1, R2

Load (R2), R2

Load (R2), R2

The Add instruction performs the Operation R2 ← X + [R1].

The Load instructions fetch the address and then the operand from the memory.

The addressing modes used in modern processors often have the following features.

* Access to an operand does not require more than one access to the memory
* Only load and store instructions access memory operands
* The addressing modes used do not have side effects.

Condition Codes :-

The Condition codes flags are stored in the processor status register. They are either set or cleared by many instructions. An Optimizing Compiler for a pipelined processor attempts to reorder instructions to avoid stalling the pipeline when branches or instructions or data dependencies b/w successive instruction occur.

```
Add        R1, R2
Compare    R3, R4
Branch=0   ...
```

a) A program fragment

```
Compare    R3,R4
Add        R1,R2
Branch=0   ...
```

(b) Instructions reordered

fig: Instruction reordering.

Assume that the excecution of the Compare and Branch=0 instruction proceeds as the branch decision takes place in step E_2 rather than O_2 because it must await the result of the Compare instruction. The excecution time of the branch instruction can be reduced by interchanging the Add and Compare instructions, This will delay the branch instruction by one or cycle relative to 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 a program the Condition Codes are affected and in which they are not. The SPARC and ARM architectures provide this flexibility.