PIPELINING

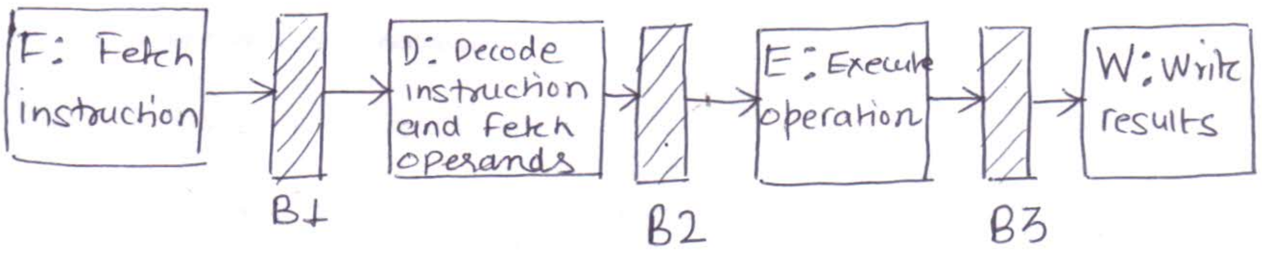
BASIC CONCEPTS

An instruction can be divided into multiple stages, like,
(a) Fetch instruction from main memory | cache
(b) Decode instruction and fetch operands
(c) Execute operation specified in instruction
(d) Write result back to Register / Memory

each of the above stages can be executed independently
The basic definition of pipeline is

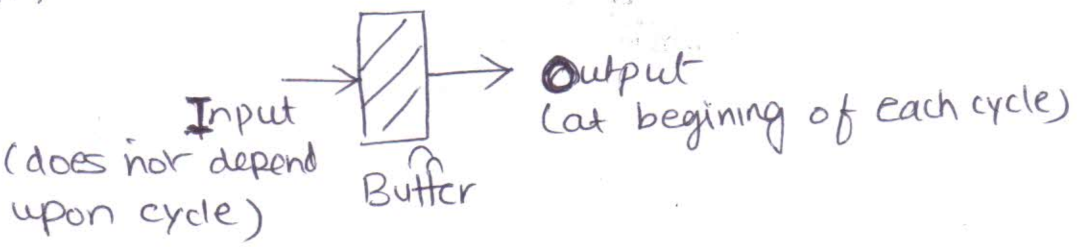
" EXECUTION OF MACHINE INSTRUCTION CONCURRENTLY."

Hardware organization for a 4-stage pipeline

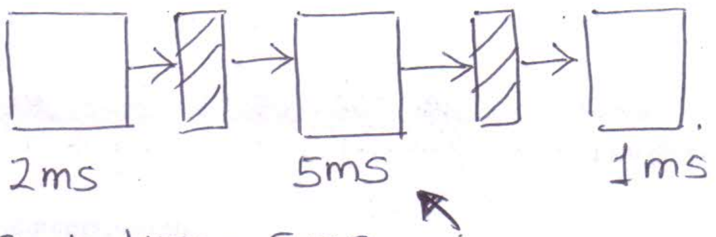


Here, F, D, E, and W are four distinct hardware units, which can perform their operations independently means without interfering with one another.
B1, B2, and B3 are buffers, which holds the intermediate results generated by its predecessor unit, for example B1 holds the results generated by F unit, B2 holds the results generated by D unit and so on.

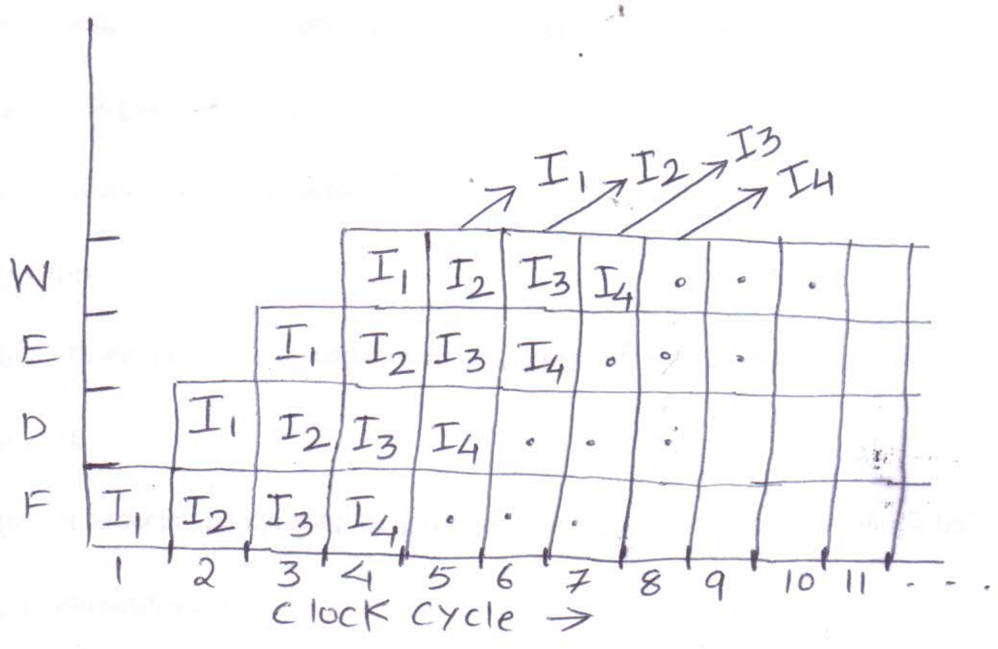
Pipeline operates in CYCLES, means at the begining of each cycle, each hardware unit receives the data from its preceding buffer, or in other words in each cycle each buffer gives its out put to the next unit.



Cycle time is set according to the time taken by the slowest hardware unit



cycle time = 5ms



In a pipeline system four instructions I_1, I_2, I_3, and I_4 take 7 cycles to complete.

whereas in case of NON- PIPELINE system

F1	D1	E1	W1	F2	D2	E2	W2	F3	D3	E3	W3	F4	→
1	2	3	4	5	6	7	8	9	10	11	12	13	

D4	E4	W4	. .
14	15	16	

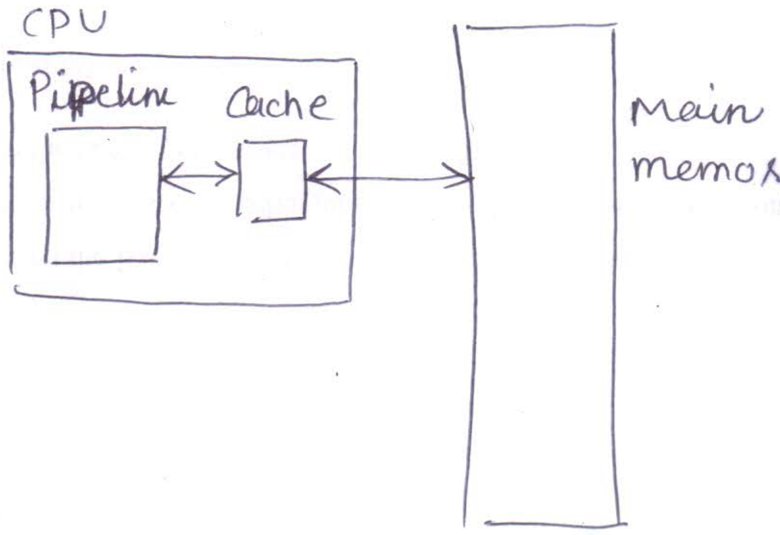
It takes 16 cycles to complete.,

So, the efficiency of a pipeline syste is,

roughly $\frac{7}{16} = 0.4375$, means a pipeline system is 44% more efficient than NON- PIPELINE system.

THE ROLE OF CACHE MEMORY

A cache memory is placed in between CPU and the main memory, generally, a cache memory can supply Instructions and data at the speed which is TEN TIMES faster than main memory. Usually, a cache is used to further reduce the overall execution time of a pipeline system



DATA HAZARDS

It is a problem in a Pipeline system which causes to stall the pipeline for one or more than one cycles. Data Hazards arises when one of the operand or more than one is/are not available for the complete execution of an instruction

For example

$A \leftarrow 3 + A$ — I_1

$B \leftarrow 4 \times A$ — I_2

~~each instruction~~

Here, I_2 instruction can not generate correct result until I_1 is completed, so due to I_1, the execution of I_2 gets delayed. (stalling of pipeline).

whereas, I_3 and I_4 can run concurrently

$A \leftarrow 5 \times C$ — I_3

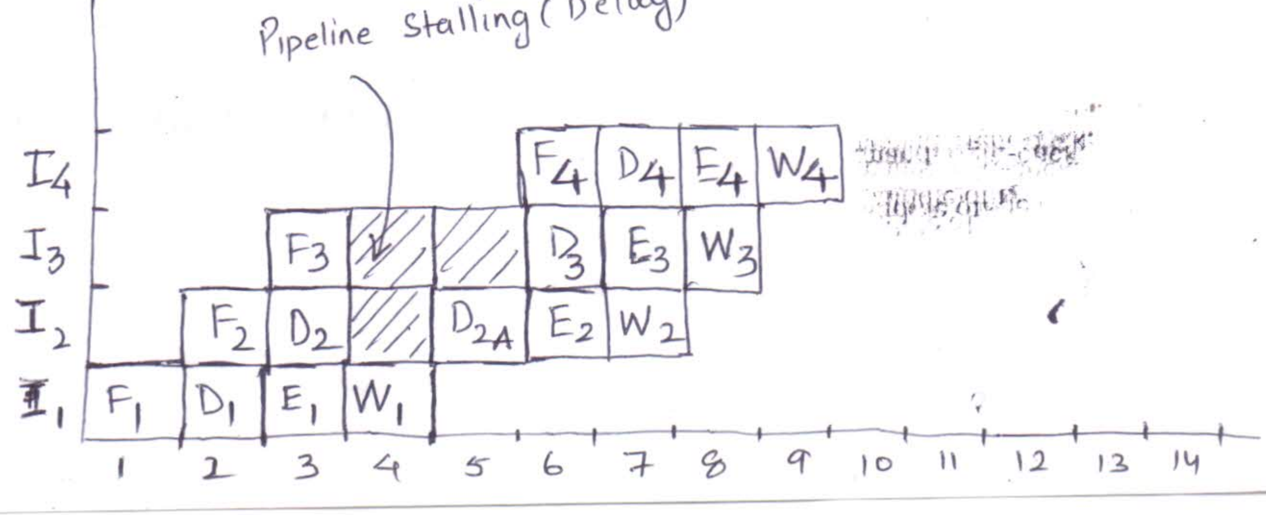
$B \leftarrow 20 + C$ — I_4

Another example of data dependency due to which Data hazards is taken place.

MUl R2, R3, R4 — I_1 { $R4 \leftarrow R2 * R3$

Add R5, R4, R6 — I_2 { $R6 \leftarrow R4 + R5$

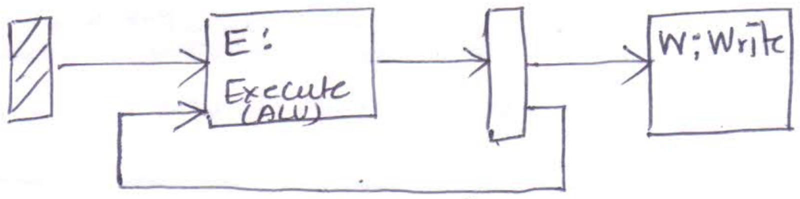
In this example, I_1 has to wait until I_2 gives R4



In the diagram, I_2 has to wait until I_1 writes its result in R4 register, the decoding stage of I_2 is waiting for the completion of writing operation of I_1. In case of I_3 since the decoding stage is busy with I_2, so I_3 has to wait ~~until~~ for its decoding, until the decoding stage is freed by I_2.

OPERAND FORWARDING

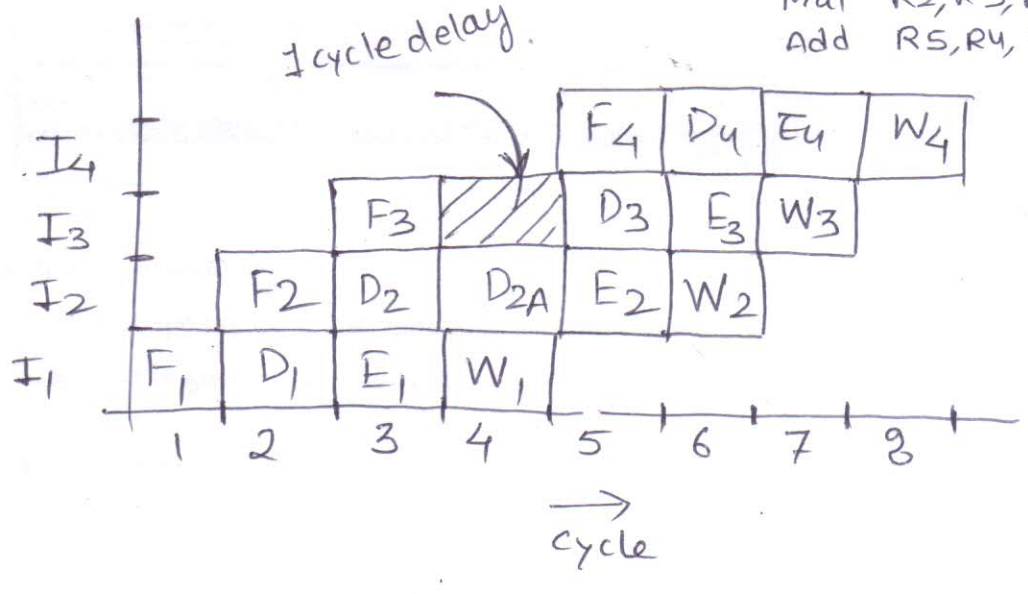
To deal with Data Hazards, operand forwarding can be used, In operand forwarding approach, the results are directly forwarded to the place where they are needed, without waiting to get them stored in destination registers.



Forwarding path

For previous example, implement operand forwarding, by doing so, we can reduce delay by one cycle.

Mul R2, R3, R4
Add R5, R4, R6

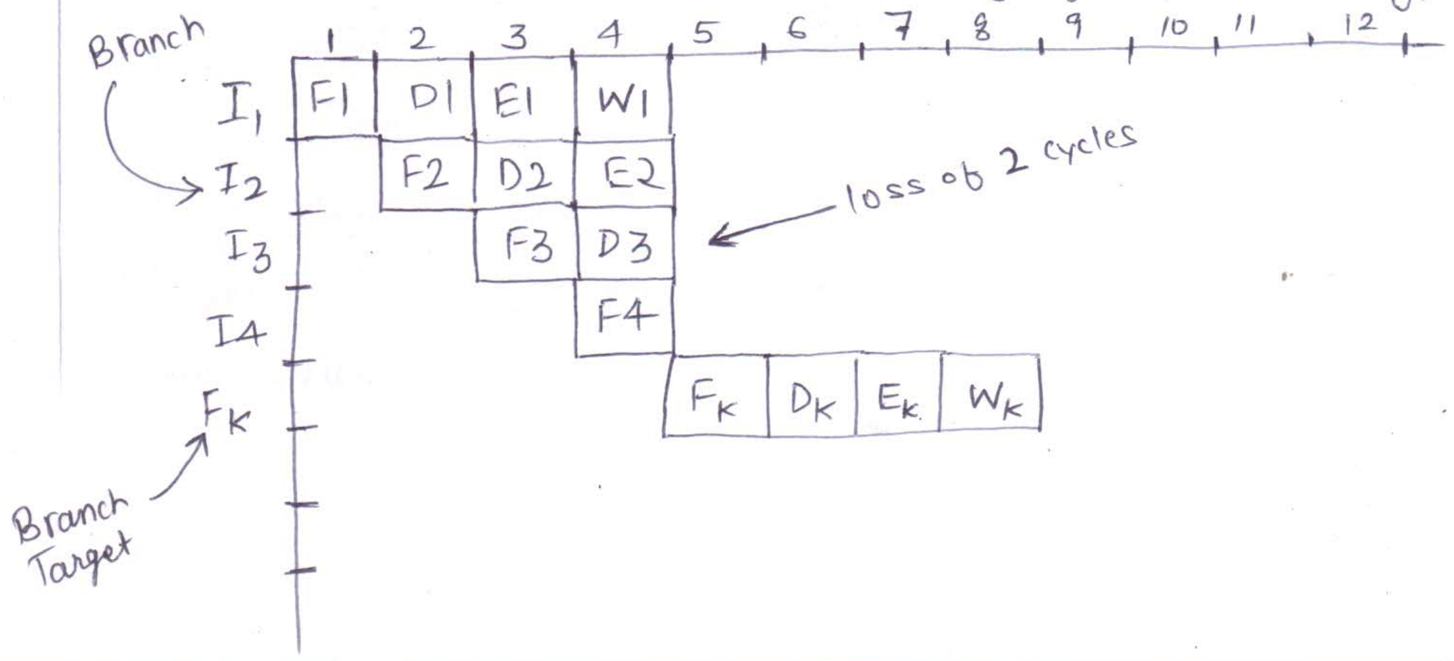


INSTRUCTION HAZARDS

Instruction Fetch unit is suppose to supply instructions to Decode unit but due to CACHE miss or due to BRANCH instruction, there may be some delay. Now, we are going to discuss the effect of BRANCH instructions on a pipeline system. Branch instruction may be of two types
(a) unconditional (b) conditional

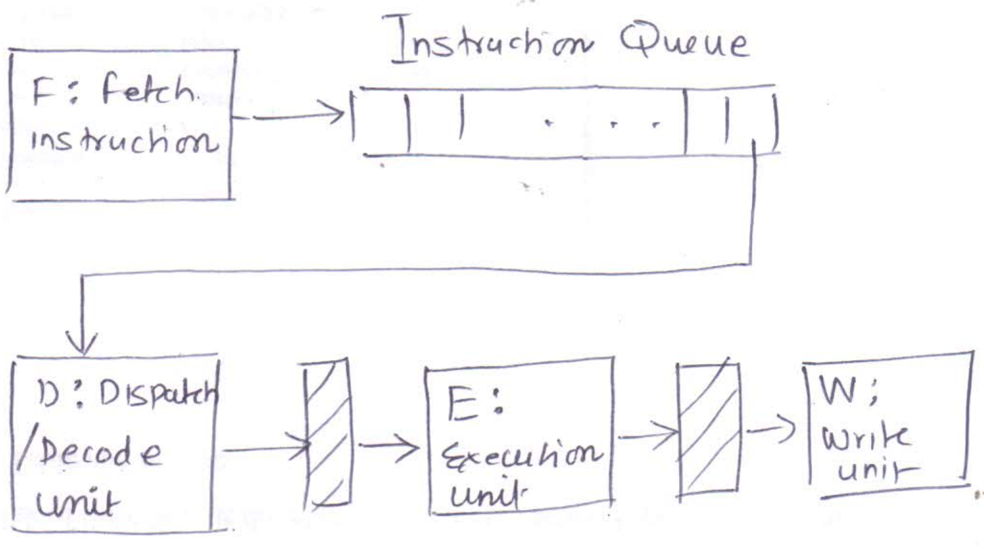
(a) Effects and Solution for unconditional branch instructions

when an unconditional branch instruction is encountered, it becomes essential to decide the next instruction, which is to be executed as BRANCH TARGET. Normally, it is decided by the EXECUTION UNIT. If the branch target instruction decided by the execution unit is not the next instruction, then the instructions that have been entered into the pipeline have to be removed, in order to load branch target instruction, which causes stalling of pipeline (delay).



INSTRUCTION QUEUE AND PREFETCHING

To avoid Pipeline Stalling due to branch instruction the concept of Instruction queue and Prefetching is implemented. Many processor implements a Sophisticated fetch unit, which fetches instructions in advance, and put them into a QUEUE. A DISPATCH unit is used to fetch an instruction from queue and send it to execution unit. Now, if fetch unit faces some delay, the dispatch unit continuously supplies instructions to Execution unit from queue, and in other case, if Execution unit faces some delay due to which dispatch unit has to stop the supply of instruction to Execution unit, Fetch unit can continue the fetching of instructions from memory and storing them into queue. thus, the delay can be minimized.



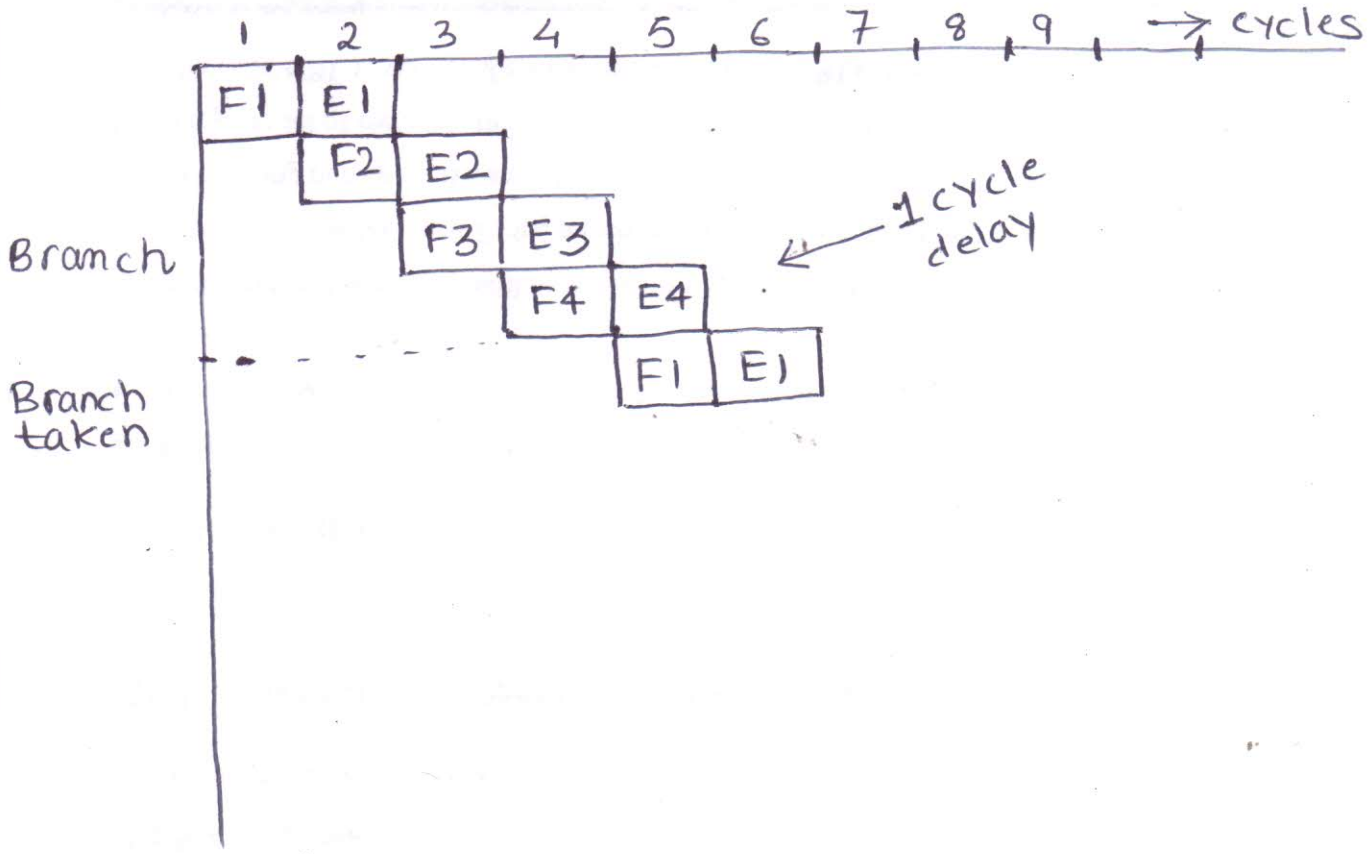
Q: What is branch folding ?
Hint: Page 468, Hamacher

CONDITIONAL BRANCHES AND BRANCH PREDICTION

DELAYED BRANCHING is a technique in which one or more than one instructions are reordered to avoid the delay caused by a branch instruction. Sophisticated compilers are used to implement this technique. Let us discuss an example,

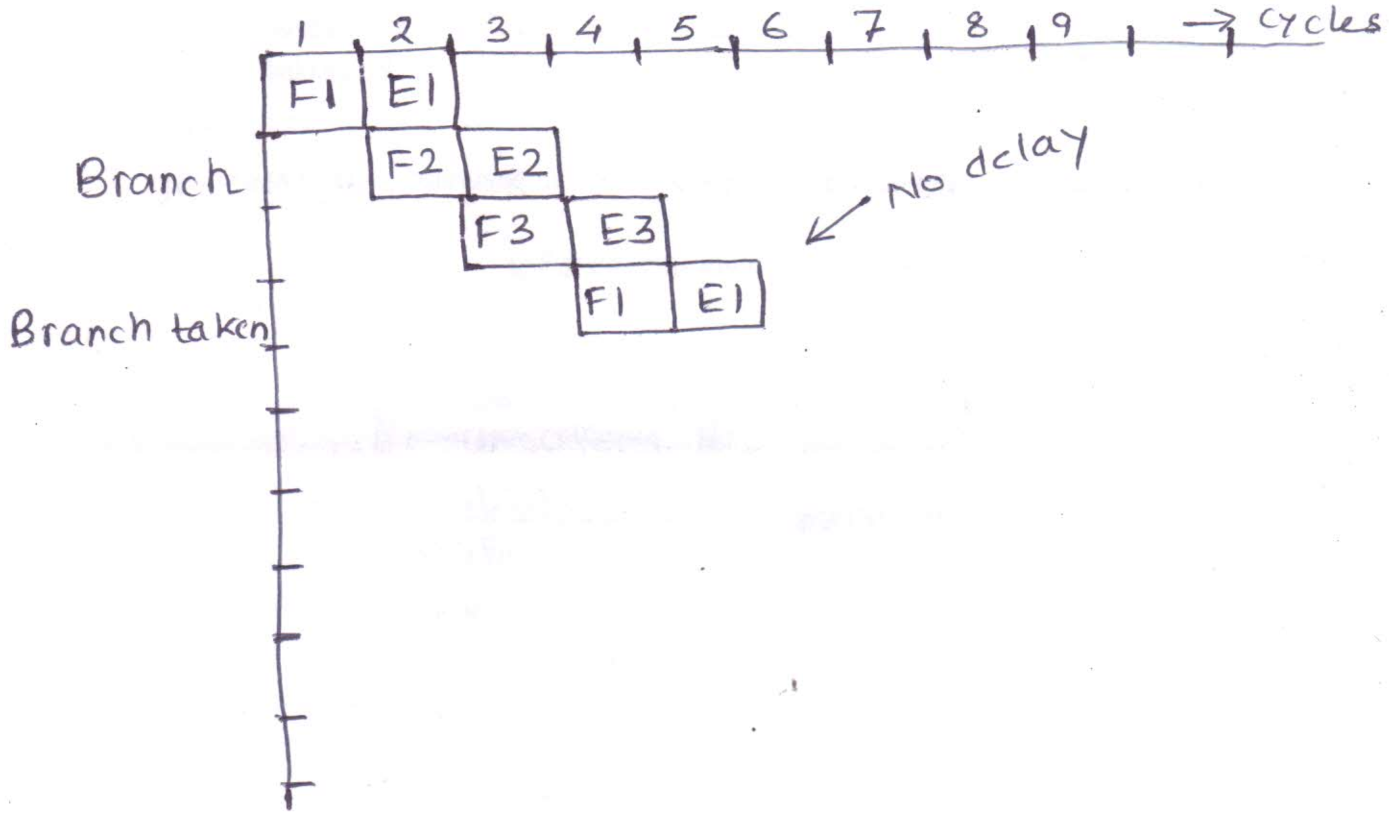
(a) Original program

LOOP	Shift_left	R1	— ①
	Decrement	R2	— ②
	Branch = 0	Loop	— ③
NEXT	Add	R1, R3	— ④



(b) Reordered Program

```
LOOP    Decrement    R2       — ①
        Branch=0     LOOP     — ②
        shift_left   R1       — ③
NEXT    Add          R1, R3   — ④
```



Here, branch instruction is executed one step later, that is why this technique is also known as "DELAYED BRANCH" technique.

BRANCH PREDICTION

To reduce the branch panelty associated with conditional branches, some prediction is used to guess whether a branch ~~is~~ would be taken or NOT. Two types of branch prediction approaches (a) static branch prediction (b) Dynamic branch prediction.

In static branch prediction, a sophisticated compiler is used to predict whether the branch would be taken or not.

Another approach is Dynamic branch prediction, in which decision is made on the basis of execution history.

A dynamic branch prediction algorithm

$$\begin{tikzcd}
SNT \arrow[loop left, "BNT"] \arrow[r, bend left, "BT"] & LNT \arrow[l, bend left, "BNT"] \arrow[d, "BT"] \\
LT \arrow[u, "BNT"] \arrow[r, bend left, "BT"] & ST \arrow[l, bend left, "BNT"] \arrow[loop right, "BT"]
\end{tikzcd}$$

Here

ST	strongly likely to be taken
LT	Likely to be taken
LNT	Likely not to be taken
SNT	strongly likely not to be taken

In case of loop execution, let us assume that processor set initial state to LNT, when loop runs for first time, branch will be taken and state is changed to ST, if loop executed for n time, each time branch is taken and state remains ST, at the end, when loop terminates branch will not be taken, and state sets to LT and so on, In short, if a branch is taken more than once, the state becomes ST, and same with SNT, if branch is not taken more than once, the state becomes SNT.

INFLUENCE OF INSTRUCTION SETS

To discuss the effect of an instruction set on a pipeline, two factors are considered

(A) ADDRESSING MODE

(B) CONDITION CODE FLAGS

(A) ADDRESSING MODE

Addressing modes specify the way/method to calculate effective address of an operand, if an addressing mode requires more than one memory accesses to ~~cal~~ Calculate effective address of an operand, makes pipeline execution slow. Q:

	1	2	3	4	5	6	7	8	9 → cycles
I_1	F1	D1	MA_1	MA_2	MA_3	W1			
I_2		F2	D2	▨	▨	E2	W2		
I_3			F3	D3	▨	▨	E3	W3	

here, MA_1 – first memory access
MA_2 – Second memory access
MA_3 – Third memory access

To avoid such delays in pipeline, The addressing modes used in modern computers have following features

(a) Access to an operand does not require more than one memory access.

(b) only load and store instructions access memory locations

(c) The addressing modes used do not have side effects.

(B) CONDITION CODES

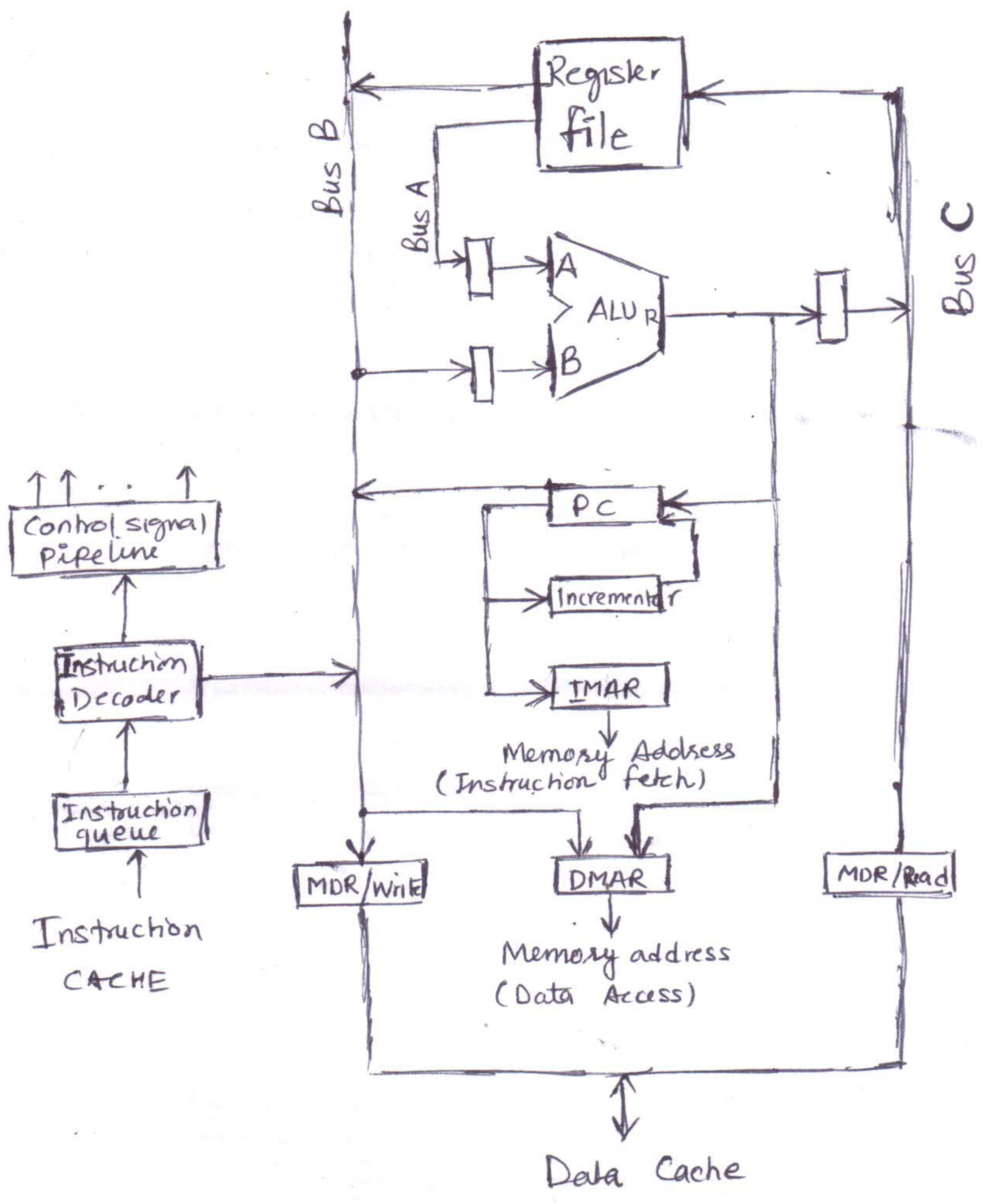
Condition codes / status registers are used to convey the information about the result of the previous instruction to next instruction. Such as if result of a subtraction instruction is zero, the ZERO FLAG is set, which can be read by next instruction, usually, condition codes are used by branch instruction to take decision about branch to be taken, or branch is not to be taken.

The instructions, which uses condition codes, are usually written in pair one after another, when reordering of instruction is done to reduce the delay in pipeline, the pair of instructions, which uses condition codes should be taken into consideration, otherwise, the logical meaning of program would be lost. For example

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

Note:- Interchanging of Add and Compare instruction is done, only when add instruction does not affect condition codes, otherwise, Branch=0 results in ~~[illegible]~~ incorrect result.

DATAPATH AND CONTROL CONSIDE-RATIONS



In this type of organization, the following points should be noted

(a) Separate instruction and data caches are used, that is why, IMAR (Instruction memory Address Register) and DMAR (Data memory Address Register) are there.

(b) PC and IMAR directly connected, and ALU is free to execute its stuff.

(c) In DMAR, the address of data can be directly obtained from Register file or from ALU.

(d) Data can be transferred through MDR without disturbing ALU.

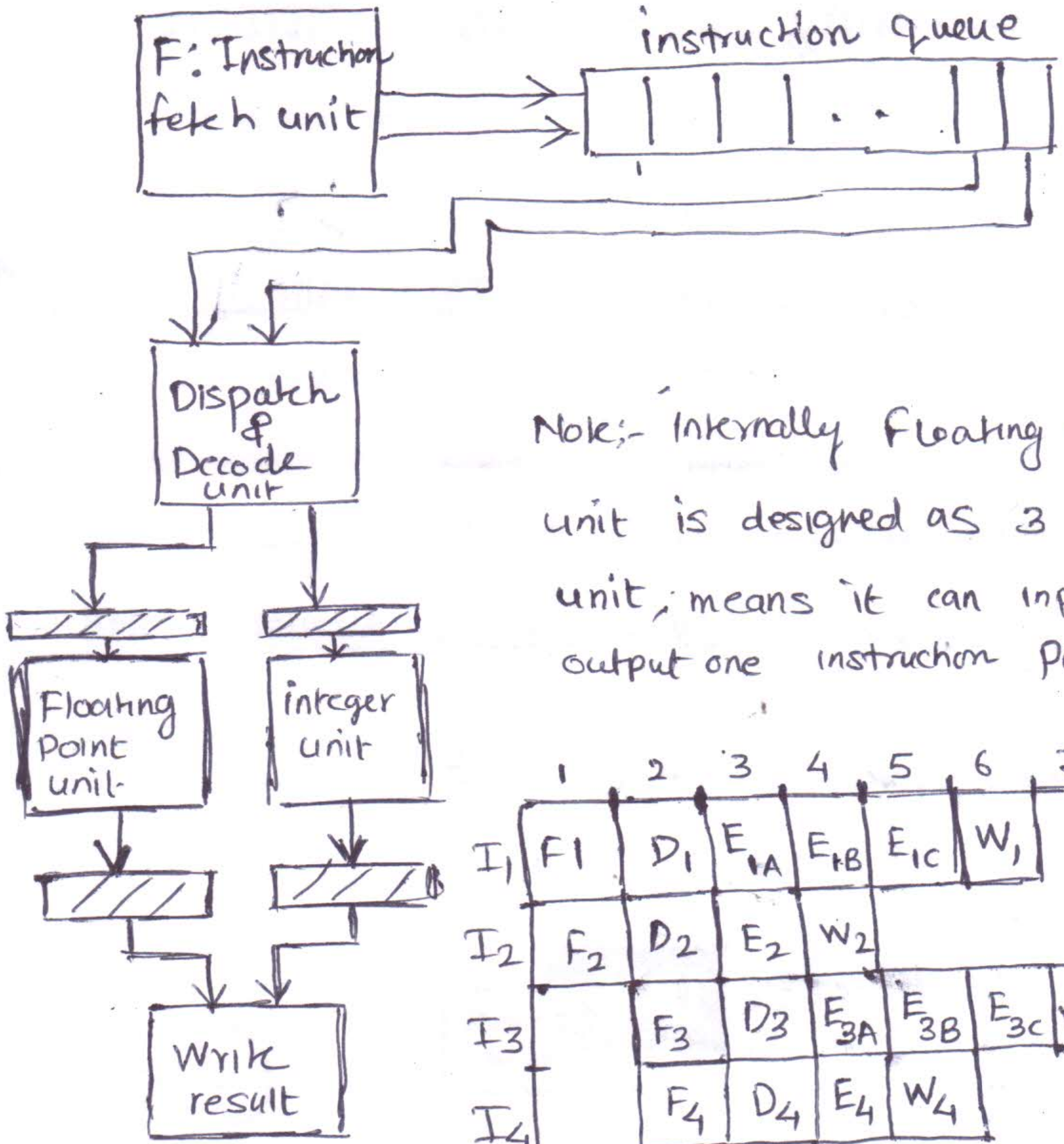
(e) IR (Instruction register) has been replaced with instruction queue.

The following operations can be performed independently

(i) Reading an instruction from instruction cache.
(ii) Incrementing the PC
(iii) Decoding the instruction
(iv) Reading / writing in data cache
(v) Reading two contents from register file
(vi) writing into one register into register file
(vii) Performing an ALU operation.

SUPER SCALAR OPERATION

To improve throughput of a pipeline, in modern computer, multiple copies of a stage is used, for example, two execution units can be implemented so that two instructions can execute simultaneously.



Note:- Internally Floating point unit is designed as 3 stages unit, means it can input and output one instruction per cycle.

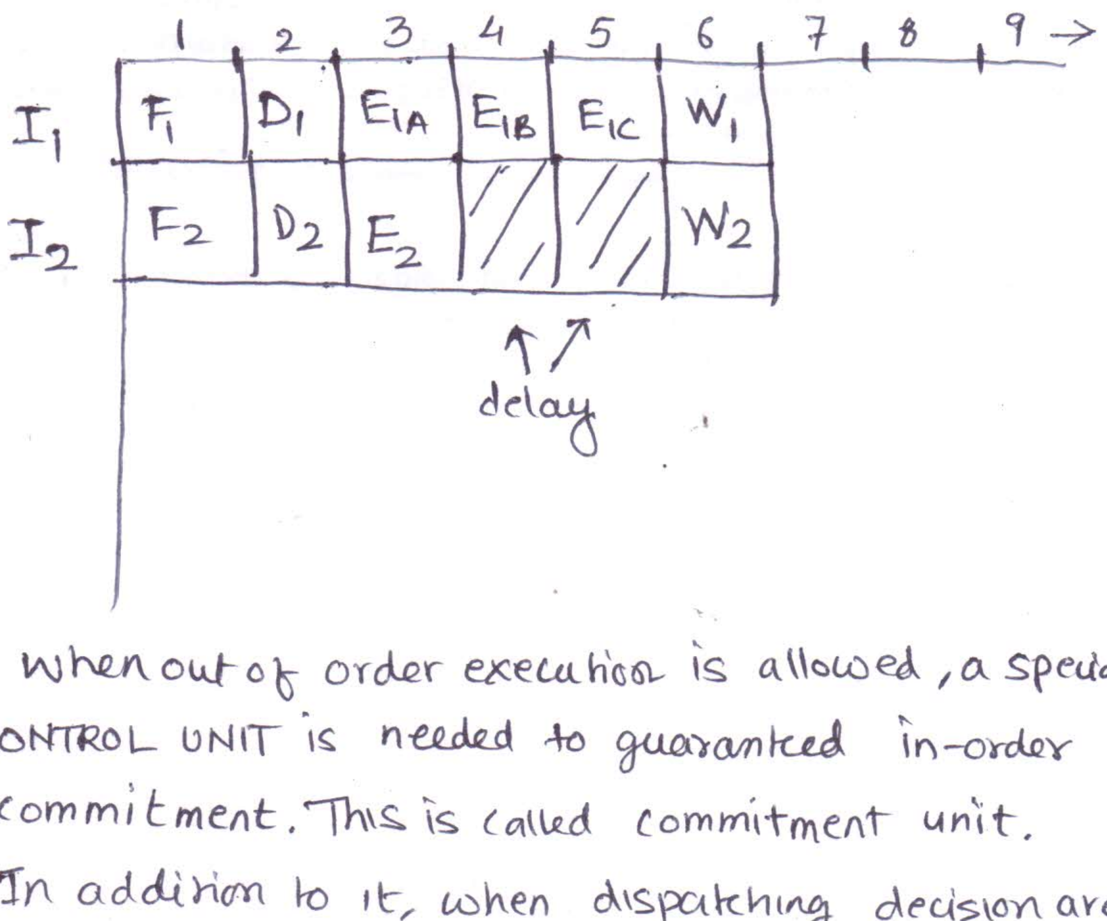
	1	2	3	4	5	6	7	8
I_1	F_1	D_1	E_{1A}	E_{1B}	E_{1C}	W_1		
I_2	F_2	D_2	E_2	W_2				
I_3		F_3	D_3	E_{3A}	E_{3B}	E_{3C}	W_3	
I_4		F_4	D_4	E_4	W_4			

Note:- Fetch and Dispatch/Decode unit can handle two instructions at a time

OUT OF ORDER EXECUTION

Due to out of order execution, means in example I_1 is the first instruction in Program but I_2 has been executed first, so, due to this out of order execution some problems may occur, if instructions are dependent on each other, what, if the result of I_1 is needed in I_2?.

To overcome this problem, the result writing of I_2 must be delayed as



When out of order execution is allowed, a special CONTROL UNIT is needed to guaranteed in-order commitment. This is called commitment unit.
In addition to it, when dispatching decision are made, the dispatch unit must ensure that all the resources needed for the execution of an instruction are available.