6. PIPELINING

⇒ pipelining is the process (or) technique which improves the performance of system (or) processor in terms of throughput.

⇒ pipelining is widely used in modern processors.

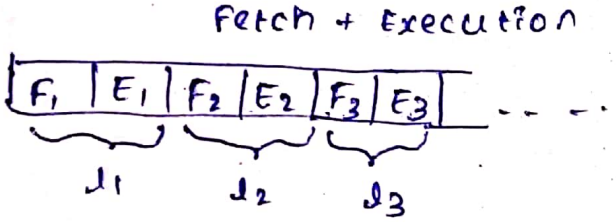
⇒ pipelined organization requires sophisticated compilation tasks.

Making the execution of programs faster (or) factors that improves the system performance

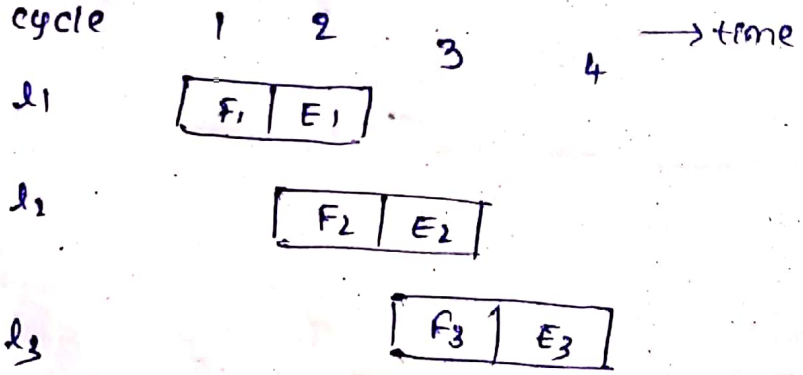
⇒ The effective design of the circuit is one of the factor of system performance improvement.

⇒ The arrangement of hardware components

Use the data of pipeling in a computer.

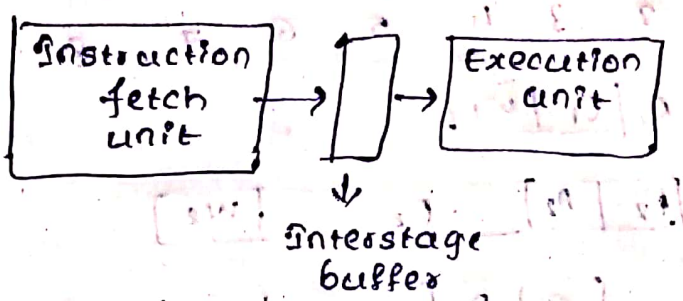


a) Sequential execution



(b) pipelined execution.

Scanned with CamScanner



(b) Hardware Organization

Fetch + Decode + Execution + Write

⇒ Here we use 3 buffers in order to perform these tasks.

Role of cache memory :-

⇒ Each pipeline stage is expected to complete in one cycle.

⇒ The clock period should be long enough to let the slowest pipeline stage to complete.

⇒ Faster stages can only wait for the slowest one to complete.

⇒ Since main memory is very slow compared to the execution, if each instruction needs to be fetched from main memory, pipeline is almost useless

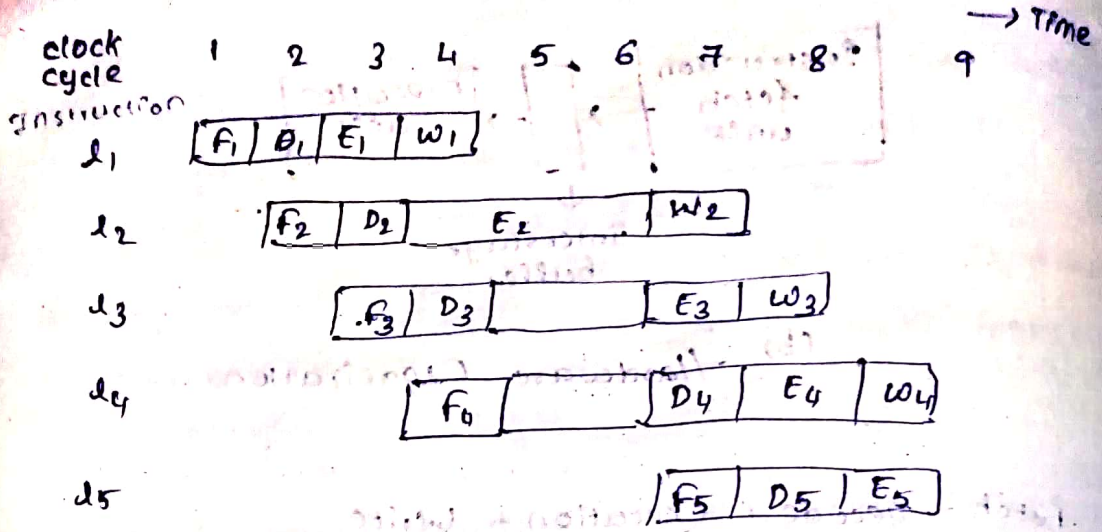
⇒ So, we have cache.

Pipelines performance :-

⇒ Pipeline performance ∝ Pipeline stages number

⇒ Some pipeline stages couldn't complete within a clock pulse due to some interruption.

Scanned with CamScanner



⇒ Generally division takes long time to complete so, E_2 is large.
This time delay is effected to remaining instructions.

⇒ Hazard is a condition, at which the pipeline stage stops its progression due to unavailability of data from other stage.

Hazard is of 3 types.

1. Data Hazard :-
   If the data is not available, the particular stage of pipeline stops

2. Instruction Hazard :-
   If the condition occurs due to non-availability of instruction.

   Ex:- F_4 ..... D_4

   If the instruction is not available in cache then also this condition occurs.
   (cache-miss)

3. Structural H
   If 2
   hard
   This
   of h

   I_1 F1
   I_2 (load)
   I_3
   I_4
   I_5

Data

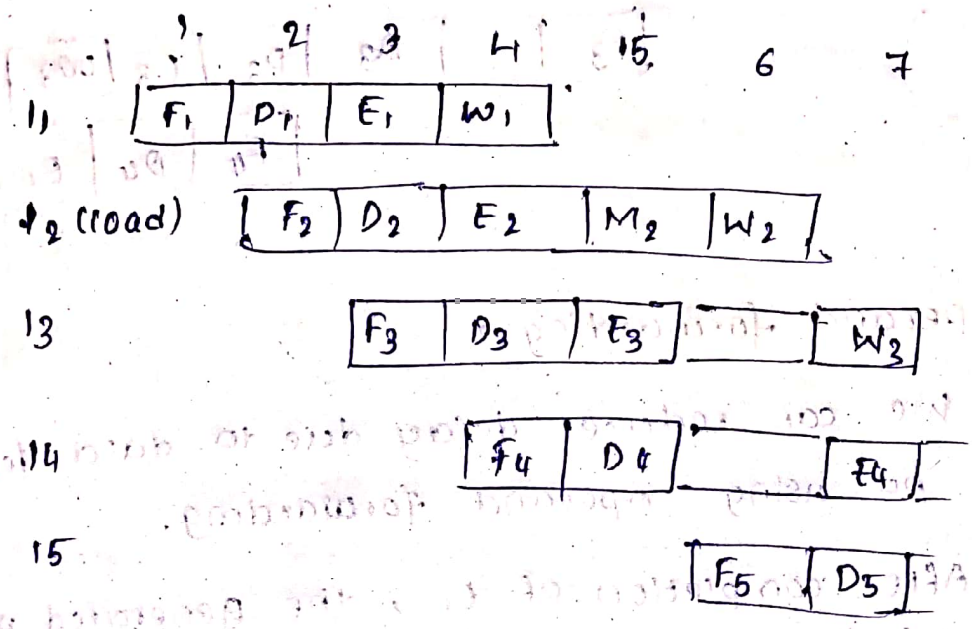
A
E

⇒ Haza
co
con

Scanned with CamScanner

3. Structural hazard :-

If 2 instructions want to use the same hardware resources at a same time. It This hazard occurs due to unavailability of hardware.



Effect of a load instruction on pipeline timing

load X(R1), R2

Data Hazard

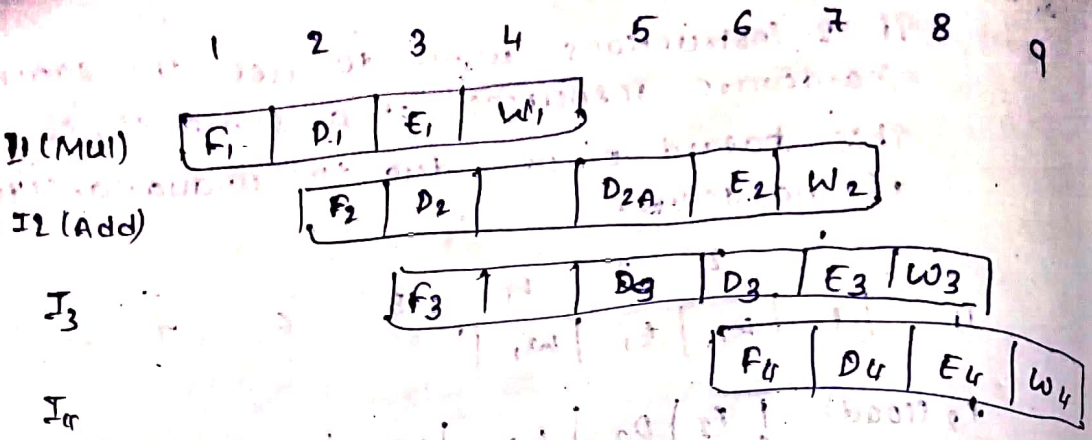
A ← B + A
B ← 4 x A
} Hazard occurs

A ← 5 x C
B ← 20 + C
} No Hazard

⇒ Hazard occurs only when we execute concurrently rather than sequentially concurrent means executing all instructions at a time.

Mul R2 R3 R4
Add R5 R4 R6

Scanned with CamScanner

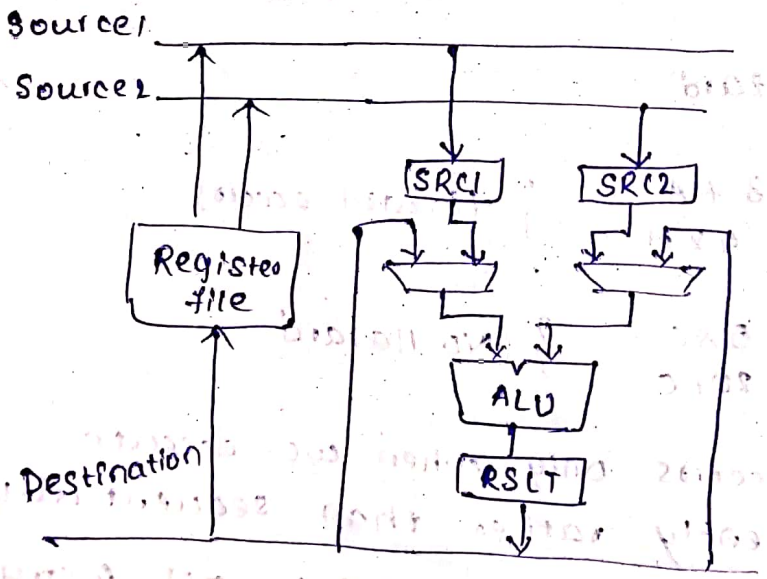


Operand forwarding :

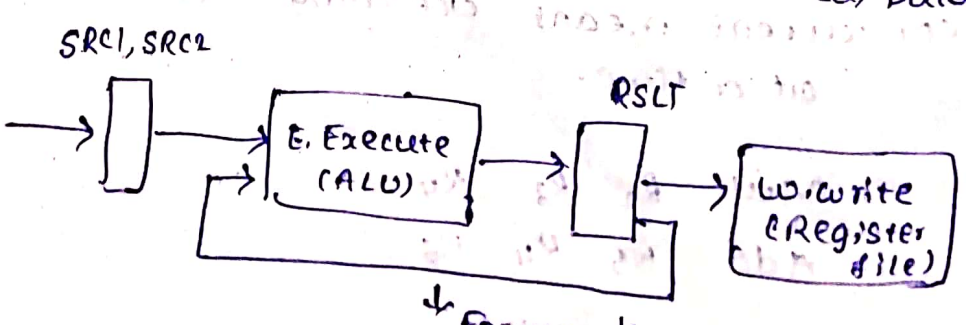
We can reduce delay due to data dependency by using operand forwarding.

After completion of E_1, the generated product is in ALU.

So, if we forward this operand, we can reduce the delay.



(a) Datapath



(b) Position of the source & result registers in processor pipeline

⇒ SRC1, SRC2
places
These
⇒ RSLT
Handlin
let th
I1
I
⇒ NOP =
As this
NOP
So, a
be
this
Datapa
⇒ Separa
⇒ PC is
⇒ DMAR
⇒ Separ
⇒ Buffe
⇒ Instr
⇒ Instr
⇒ Rea
⇒ Inc
⇒ Dec
⇒ Rec
⇒ Rec
⇒ Rec
⇒ wr
⇒

Scanned with CamScanner

⇒ SRC1, SRC2 registers stores the 2 operands and places in Execute phase (E_1)
These are in b_2 register.

⇒ RSLT value is forwarded to E_2 phase.

Handling datahazards in software:

let the compiler detect & handle the hazard.

I1 : Mul R_2, R_3, R_4
NOP
NOP
I2 : Add R_5, R_4, R_6

⇒ NOP = No operation instruction.

As this delay is 2 cycles, we placed 2 NOP instructions

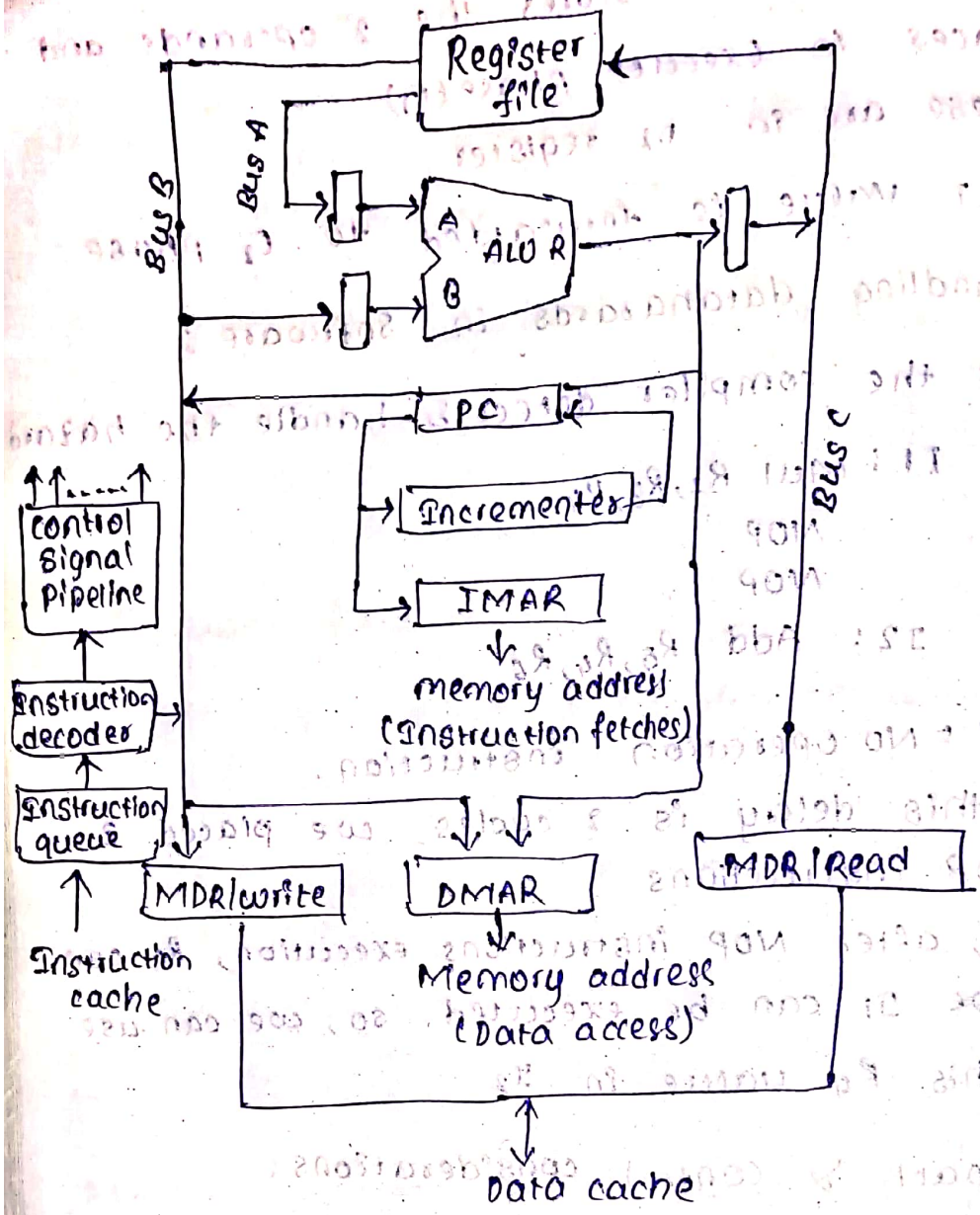
So, after NOP instructions execution, R_4 can be I1 can be executed. so, we can use this R_4 value in I_2

Datapath & control considerations:

⇒ Separate Instruction and data cache
⇒ PC is connected to IMAR
⇒ DMAR
⇒ Separate MDR
⇒ Buffers for ALU
⇒ Instruction queue
⇒ Instruction decoder o/p.

⇒ Reading an instruction from the Instruction cache
⇒ Incrementing the PC
⇒ Decoding an Instruction
⇒ Reading from or writing into the data cache
⇒ Reading the contents of up to 2 regs
⇒ writing into one register in the reg file
⇒ Performing an ALU operation.

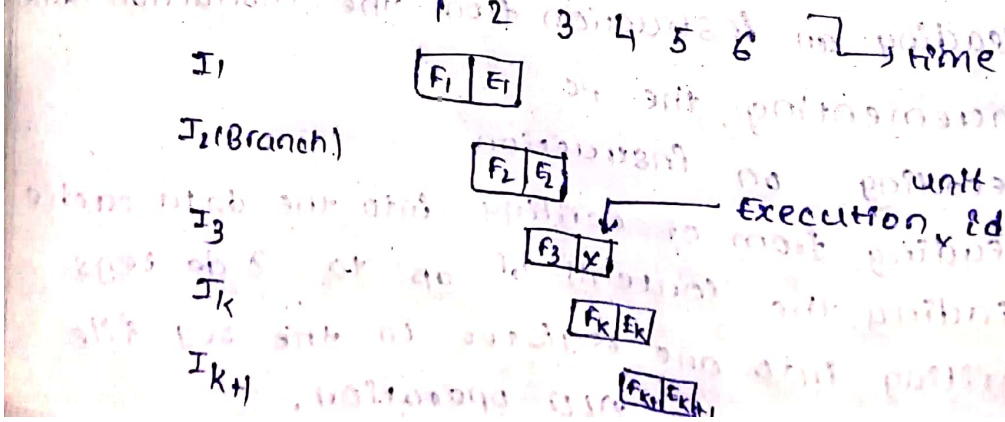
Scanned with CamScanner



Instruction Hazard

Whenever the stream of instructions supplied by the instruction fetch unit is interrupted, the pipeline stalls/stops.

→ cache miss
→ Branch

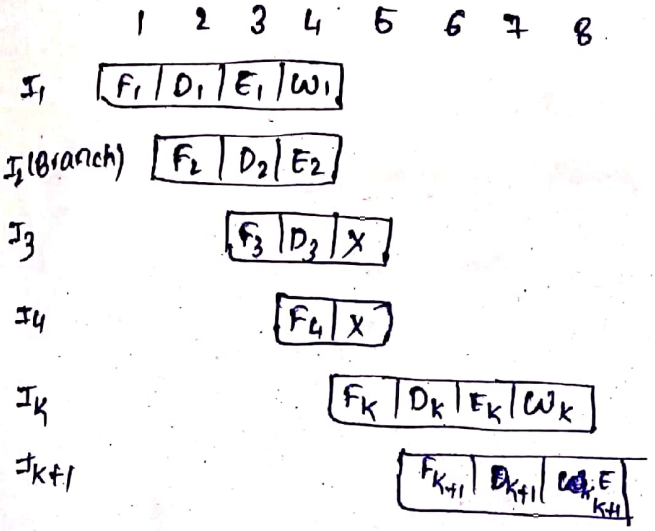


Scanned with CamScanner

Branch Penalty :-

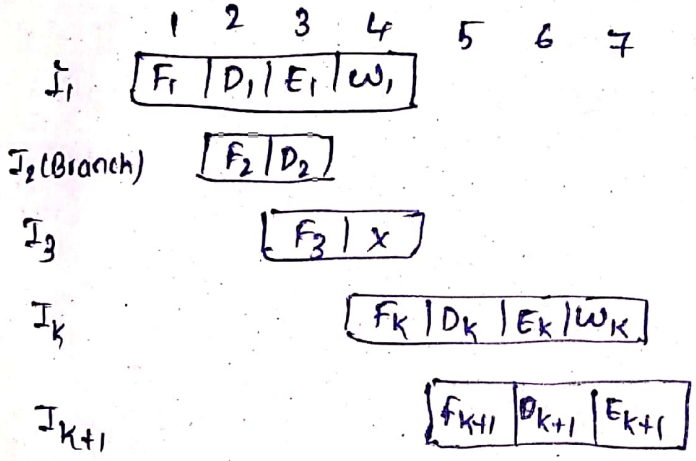
loss of a clock cycle due to branching.

⇒ Branch penalty ∝ no. of clock cycles



(loss of 2 clock cycles)

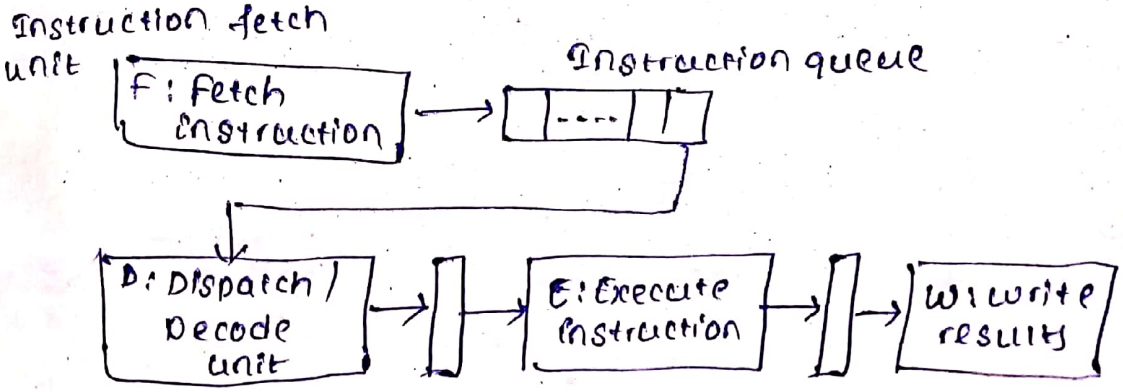
(a) Branch address computed in execute stage



(loss of 1 clock cycle)

(b) Branch address computed in decode stage

Instruction queue & prefetching :-



(Dispatch unit)

Use of an instruction queue in hardware Organisation

Scanned with CamScanner