3/17/2020

PIPELINING

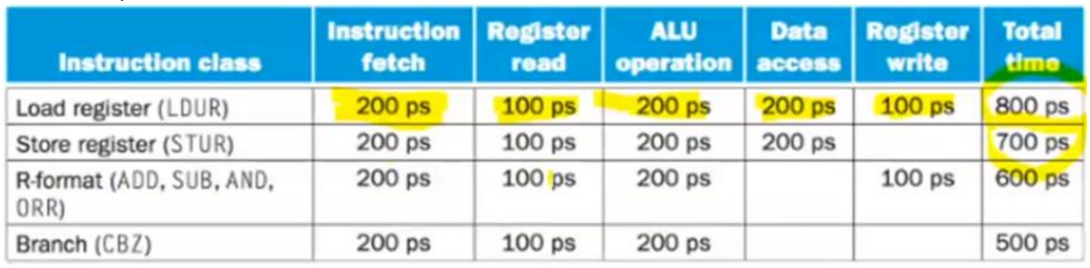
5 STAGES: IF / ID / EX / MEM / WB

Single Data Path vs Pipelining Instructions (Simultaneously)

3 HAZARDS: Structural(Hardware) / Data(with ALU & Mem) /Control

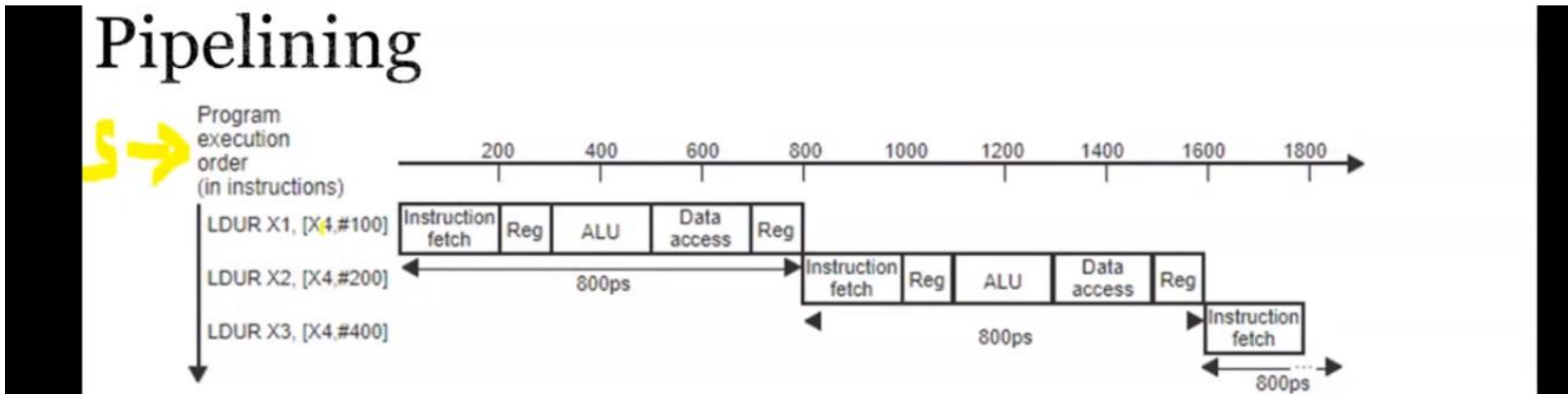
Abbrev.		
IF	Instruction Fetch	Fetching Instruction from Memory
ID	Instruction Decode	Read Registers & Decode instruction
EX	Execute	ALU/Execute Operation || Calculate Address
MEM	Memory Access	(If neccessary) Access an Operand (NOT Used in ADD)
WB	Write Back	Write Result in Register

EXAMPLE.)

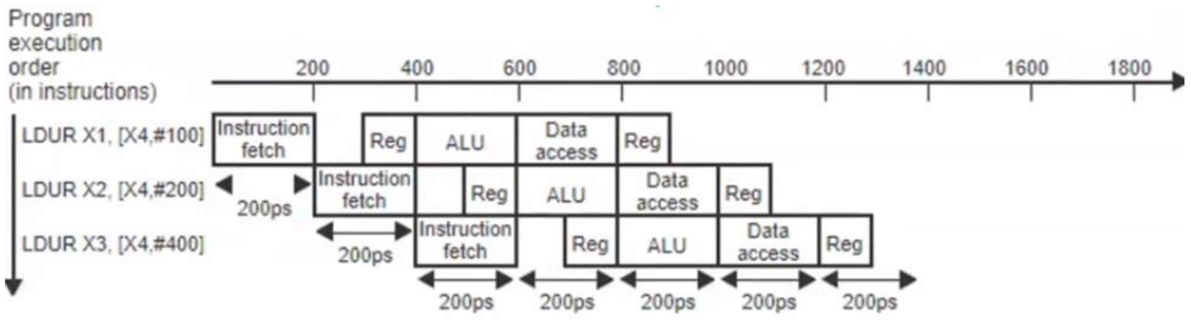


Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load register (LDUR)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store register (STUR)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (ADD, SUB, AND, ORR)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (CBZ)	200 ps	100 ps	200 ps			500 ps

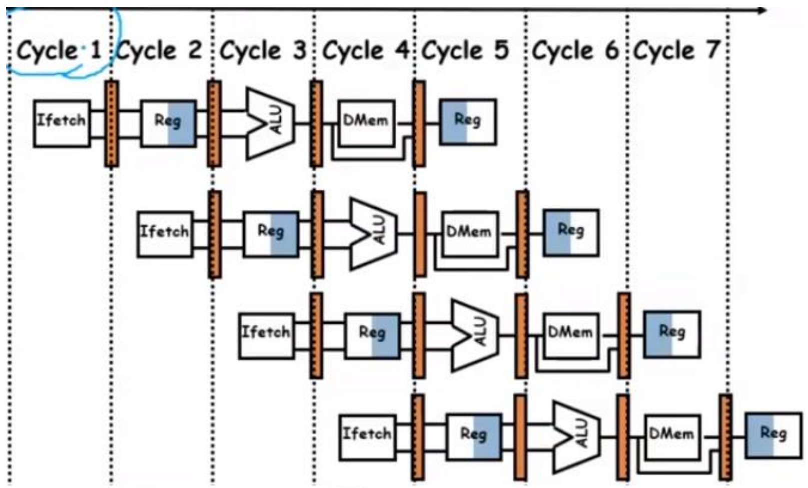
SINGLE DATA PATH



PIPELINED PATH



$$\text{Time between instructions}_{\text{Pipelined}} = \frac{\textit{Time between instructions}_{\textit{NonPipelined}}}{\textit{No.of pipe stages}}$$

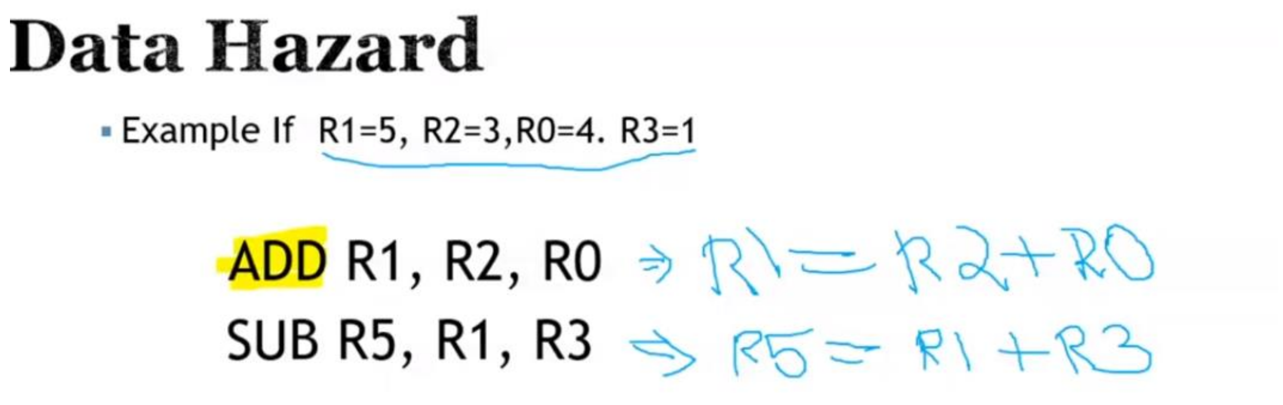


Pipeline Hazards

- There are situations in pipelining when the next instruction cannot execute in the following clock cycle
- Structural hazard
- Data hazards
- Control Hazards

Pipeline hazards

- **Structural Hazards**
  - The hardware cannot support the combination of instructions that we want to execute in the same clock cycle.
- **Data Hazards**
  - When a planned instruction cannot execute in the proper clock cycle because data that is needed to execute the instruction are not yet available.
    - ***Forwarding /Bypassing***. A method of resolving a data hazard by retrieving the missing data element from internal buffers rather than waiting for it to arrive from programmer-visible registers or memory.
    - **Stall** : inserting one or more **"bubbles"** in the pipeline

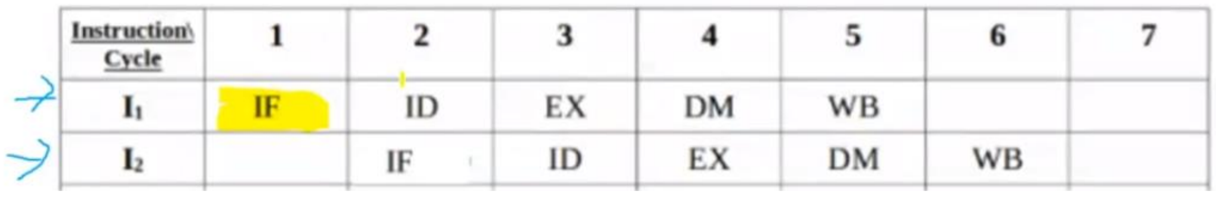


Data Hazard

- Example If R1=5, R2=3,R0=4. R3=1

ADD R1, R2, R0 ⇒ R1 = R2+R0

SUB R5, R1, R3 ⇒ R5 = R1 + R3



Instruction\Cycle	1	2	3	4	5	6	7
I_1	IF	ID	EX	DM	WB		
I_2		IF	ID	EX	DM	WB	

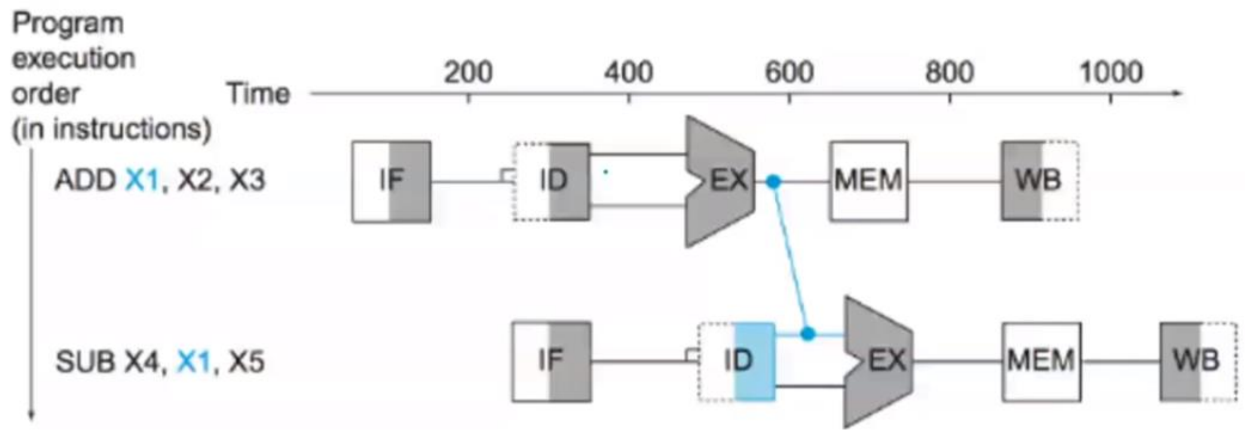
<u>For ADD R1, R2, R0:::</u>

IF	-	Read ADD opcode	
ID	-	Read Register Values	(R2=3, R0=4)
EX	-	3 + 4 = 7 ALU Execution	
MEM	-	N/a	
WB	-	Stores 7 in R1	

HAZARD: R1 is not updated in MEM before it is needed in the SUB command.
R1 is 5 before updated, 7 after updated.

THE SOLUTION:

Bypassing OR *Forwarding*



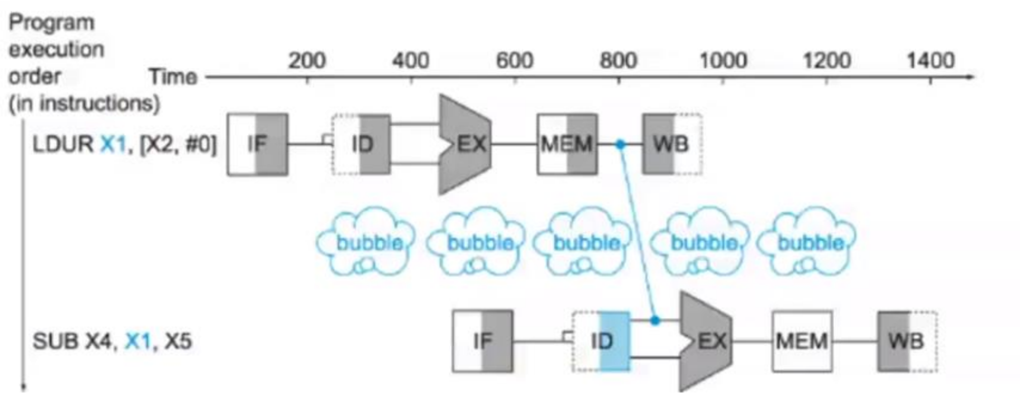
Passing values into **INTERMEDIATE REGISTERS** before the 5 stages are done.

-BYPASSING || FORWARDING – Uses **INTERMEDIATE REGISTERS** to pass information before the 5 stages are done.
-**STALLING** – Using the **NOP** instruction to prevent Data hazards.

Data Hazards

- Common solution is to **stall** the pipeline until the hazard is resolved, inserting one or more "**bubbles**" in the pipeline. Programmers use NOP instruction

A



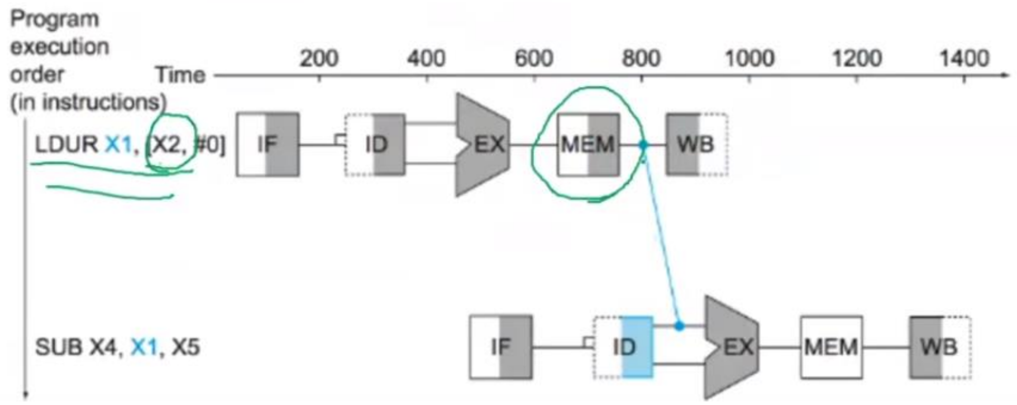
ADDING
NOP
instruction

(Like a Delay)

NOP is used if the programmer is responsible for avoiding the data hazards.

Data Hazards *Example*

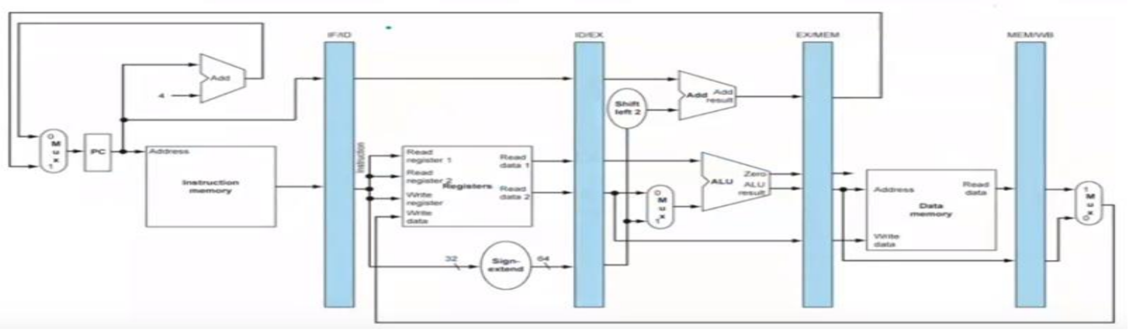
- ***Load-use data hazard***: A specific form of data hazard in which the data being loaded by a load instruction has not yet become available when it is needed by another instruction.



INTERMEDIATE REGISTERS:

Pipeline registers

- Need registers between stages
  - To hold information produced in previous cycle



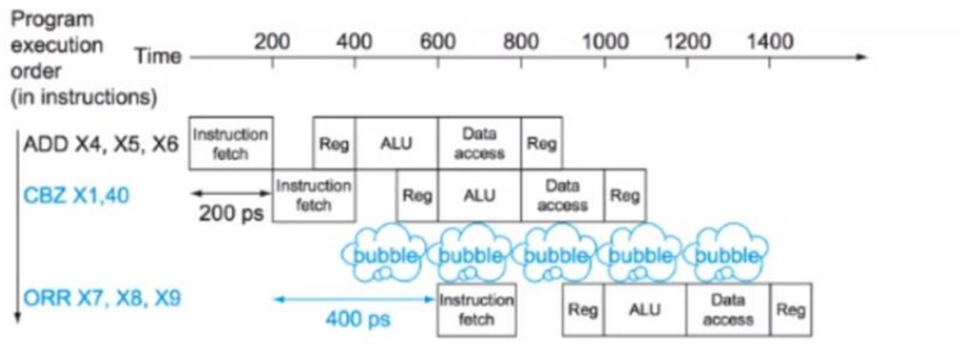
3/31/2020

CONTROL HAZARDS- Branch hazards

-Branches are determined @ ALU(Execution) stage..When the flags are being compared.

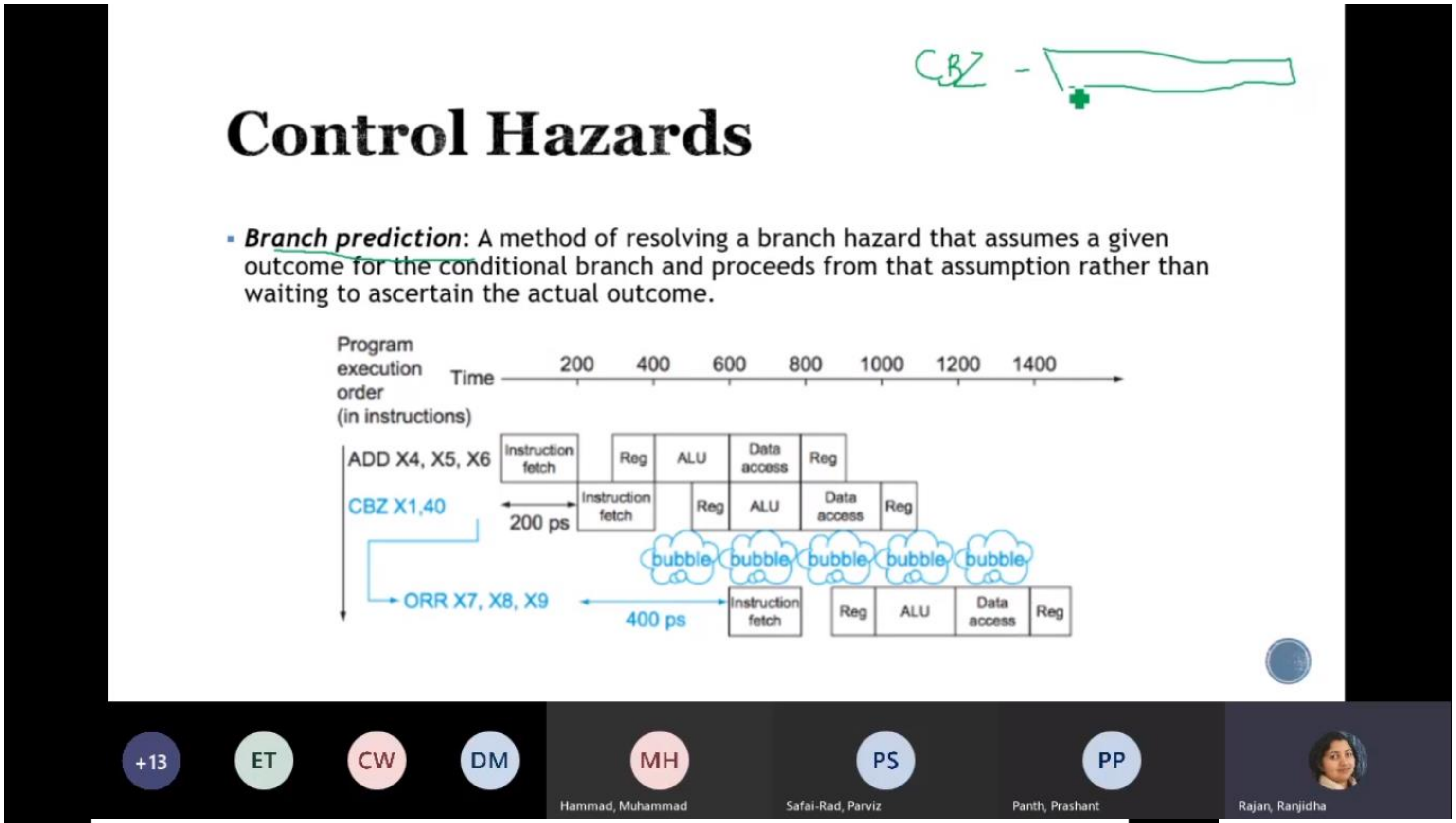
Control Hazards

- Control Hazards /Branch hazard
  - Arising from the need to make a decision based on the results of one instruction while others are executing.



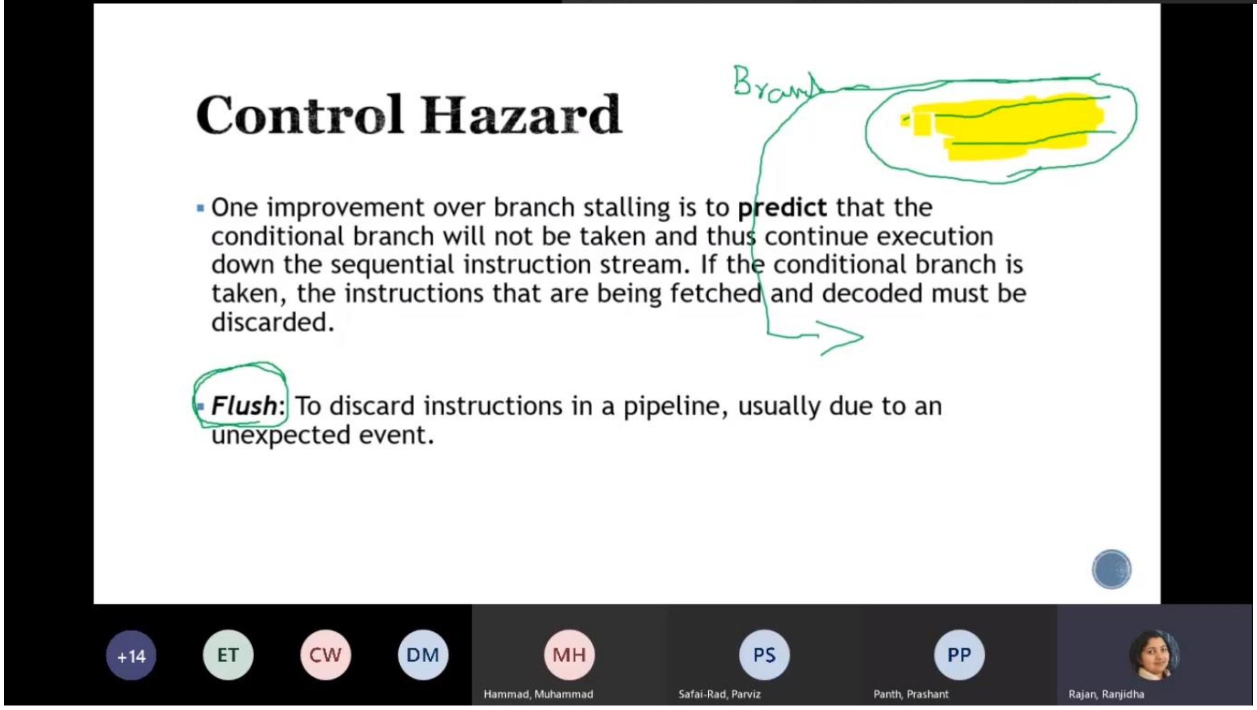
--An instruction that come after a Branching instruction must wait until the ALU executes, 3rd stage, before the next instruction can fetch instructions, IF, 1st stage.

--The Assumption is that the instructions will not branch to the next set of instructions.



Control Hazards

- ***Branch prediction***: A method of resolving a branch hazard that assumes a given outcome for the conditional branch and proceeds from that assumption rather than waiting to ascertain the actual outcome.

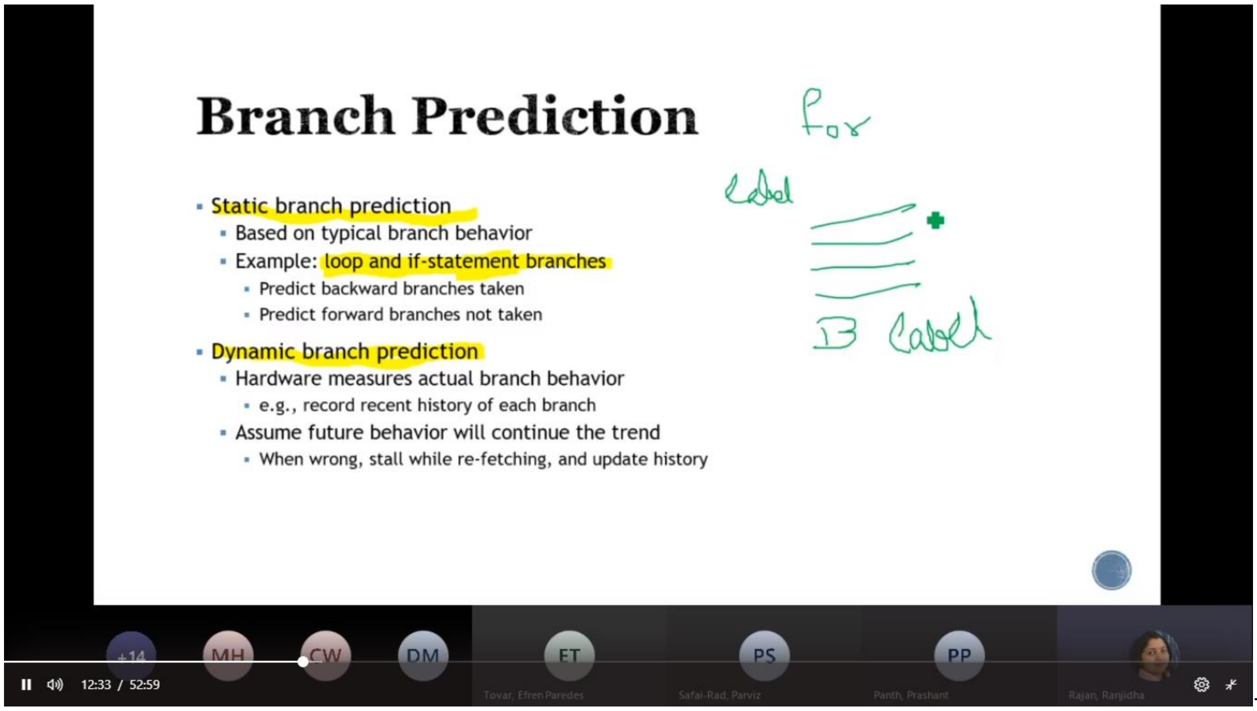


Control Hazard

- One improvement over branch stalling is to **predict** that the conditional branch will not be taken and thus continue execution down the sequential instruction stream. If the conditional branch is taken, the instructions that are being fetched and decoded must be discarded.

- ***Flush***: To discard instructions in a pipeline, usually due to an unexpected event.

FLUSH – gets rid of the instructions that come before the location to branch.

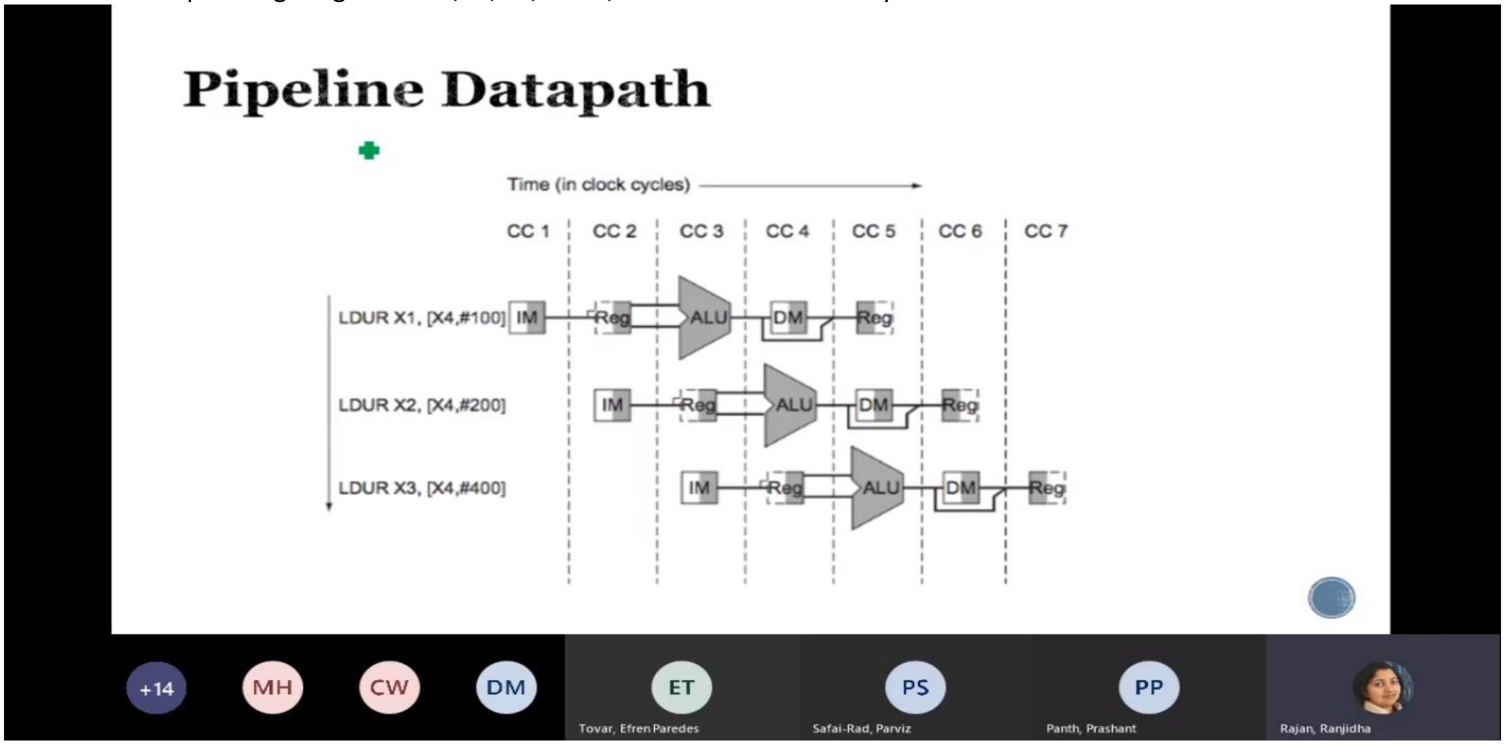


Branch Prediction

- Static branch prediction
  - Based on typical branch behavior
  - Example: loop and if-statement branches
    - Predict backward branches taken
    - Predict forward branches not taken
- Dynamic branch prediction
  - Hardware measures actual branch behavior
    - e.g., record recent history of each branch
  - Assume future behavior will continue the trend
    - When wrong, stall while re-fetching, and update history

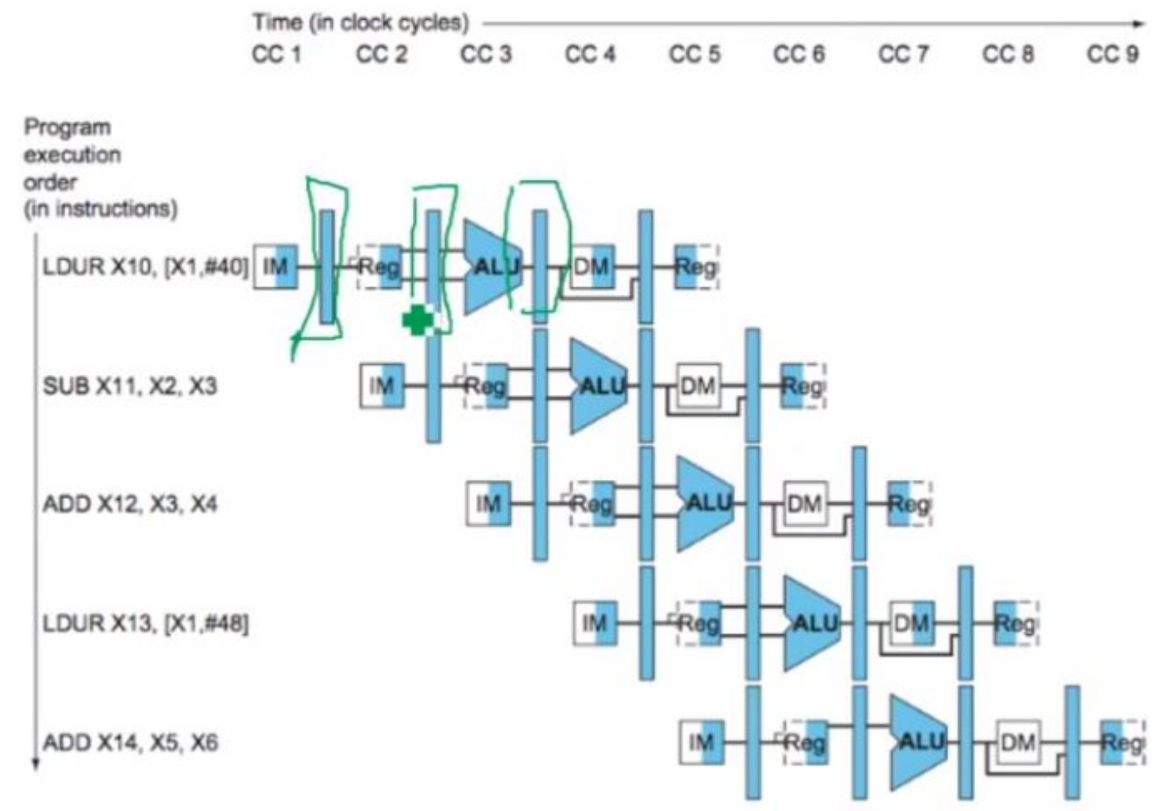
-**Static Branching prediction**: Loop || if-statement. It is wrong on the last iteration.
-**Dynamic Branching prediction**: Based on the history of each branch. Implemented w/ Hardware
-

How to draw PipeLining Diagrams = IF/ID/EX/MEM/WB & Number Clock Cycles

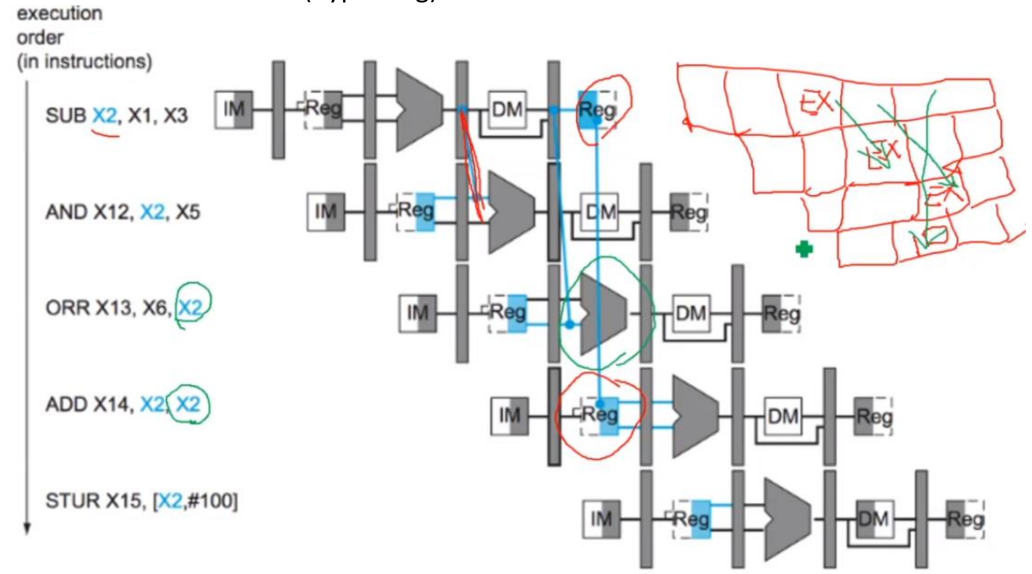


Pipeline Datapath

With Intermediate Registers.



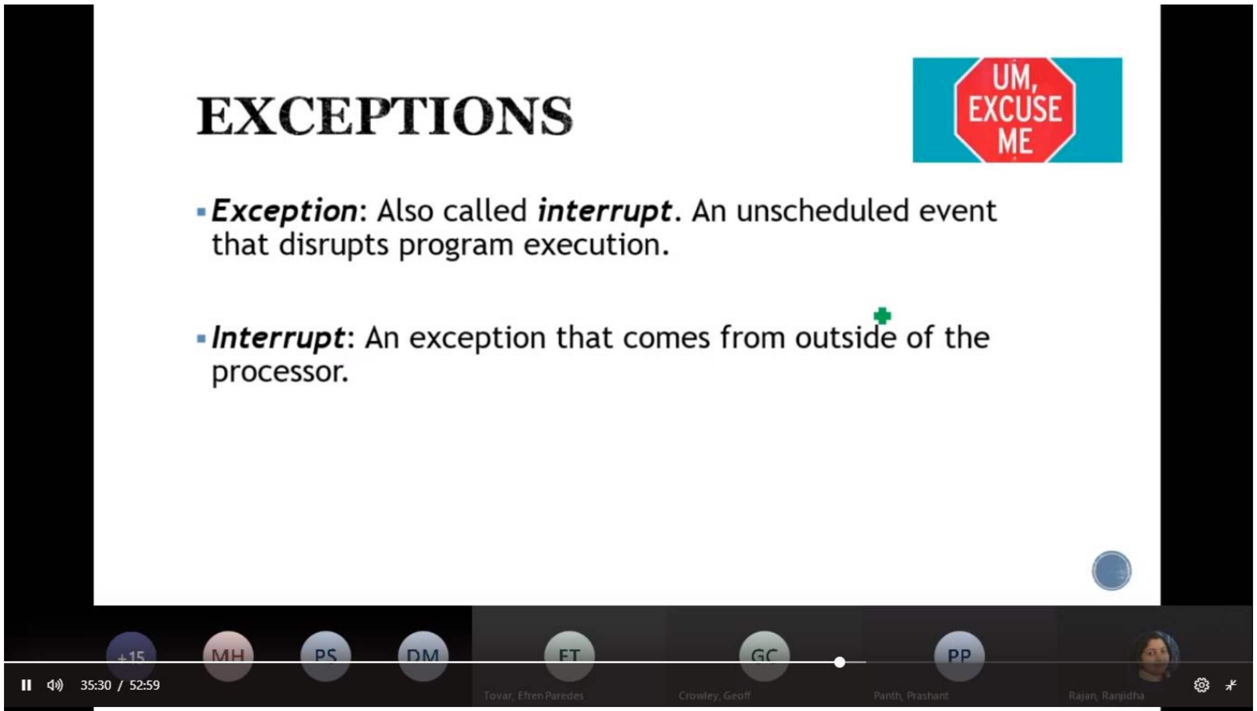
If Line 1 has the Value being updated to use in other lines, there must be 2 lines in between the lines.
Line 2 must wait 3 cycles for Line 1 to finish with the ALU.
This shows FORWARDING(Bypassing)



--Bubbling or NOP adds gap in between instructions to let some process happen first.
--ALWAYS: All Arrows for forwarding from intermediate registers must go to the right.

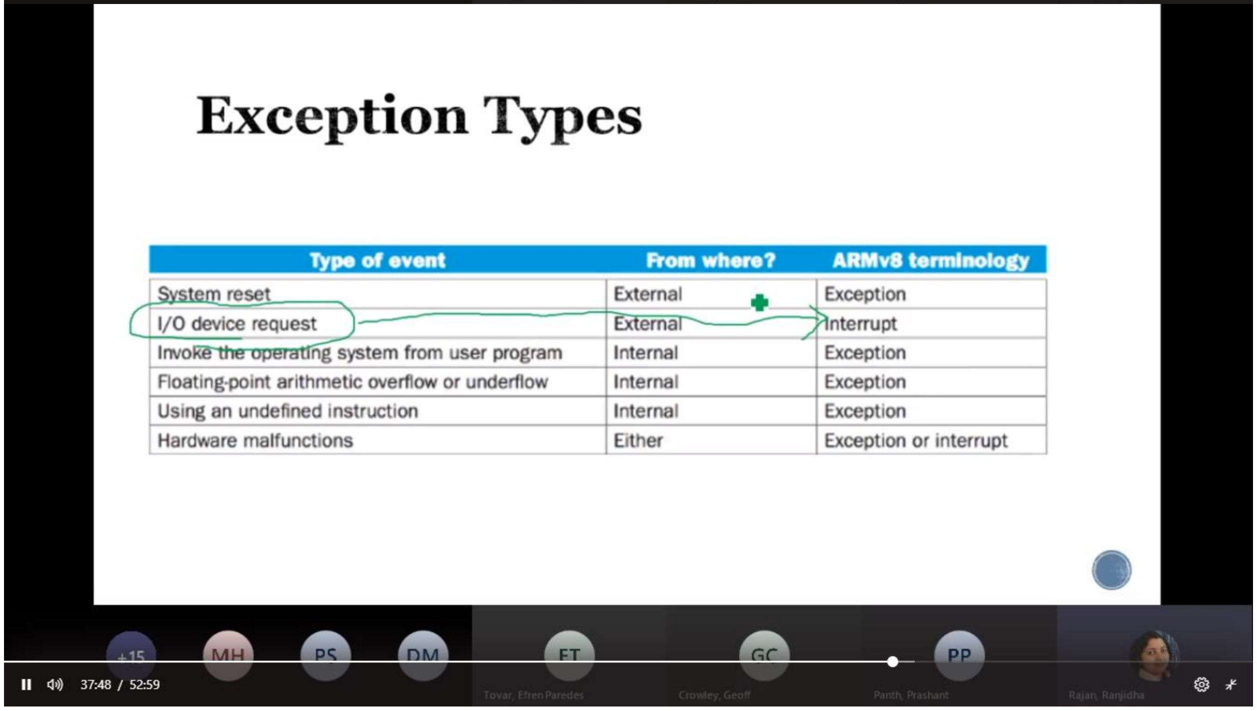
--Latency- Time taken to execute 1, 5-stage, pipeline.
--Hazard Detection Unit / Forwarding Unit- the Hardware Helps with Forwarding.

--Exceptions/Interrupts – Caused by Errors / Power Failure.



EXCEPTIONS

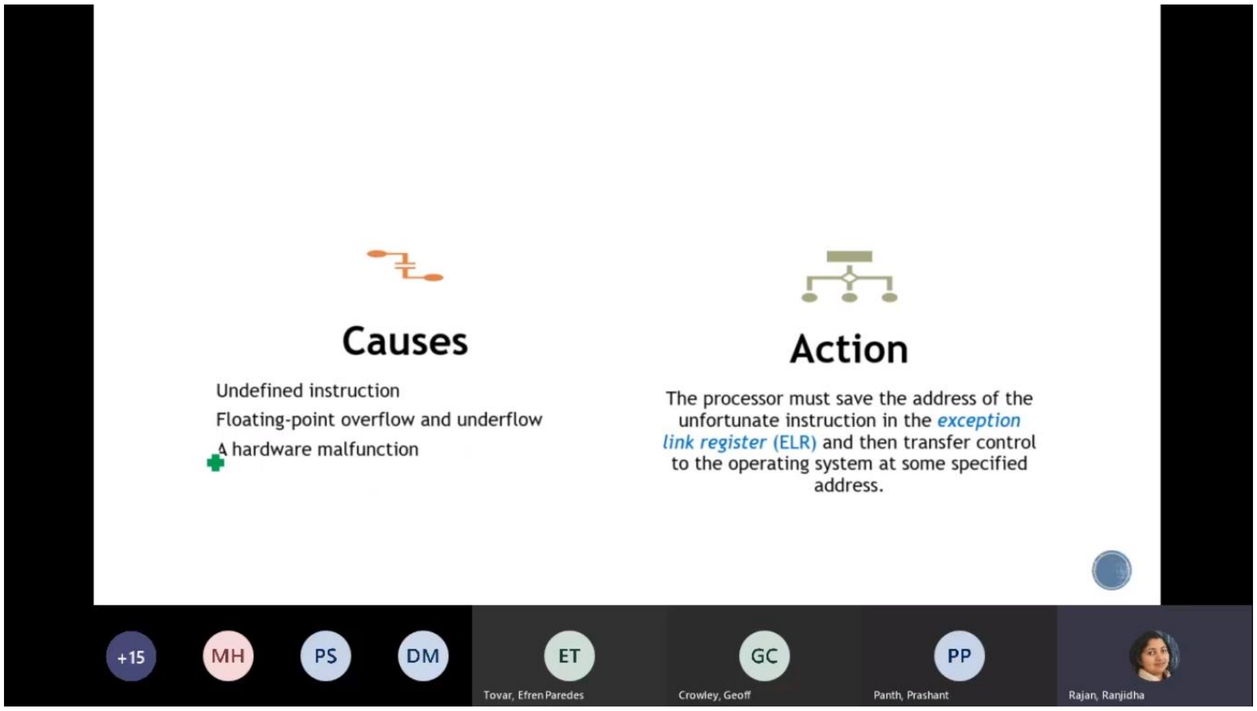
- ***Exception***: Also called ***interrupt***. An unscheduled event that disrupts program execution.
- ***Interrupt***: An exception that comes from outside of the processor.



Exception Types

Type of event	From where?	ARMv8 terminology
System reset	External	Exception
I/O device request	External	Interrupt
Invoke the operating system from user program	Internal	Exception
Floating-point arithmetic overflow or underflow	Internal	Exception
Using an undefined instruction	Internal	Exception
Hardware malfunctions	Either	Exception or interrupt

--All Error events are **Exceptions**, except I/O Device Request is an **Interrupt**.



Causes

Undefined instruction

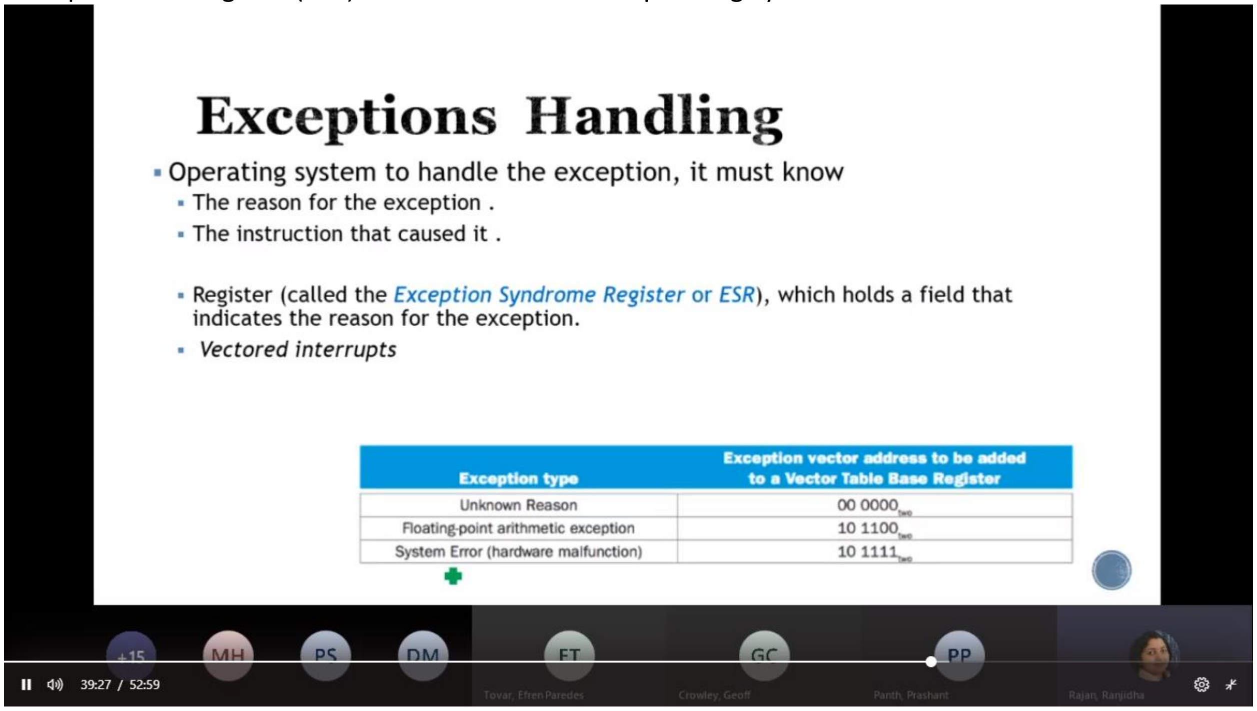
Floating-point overflow and underflow

A hardware malfunction

Action

The processor must save the address of the unfortunate instruction in the *exception link register* (ELR) and then transfer control to the operating system at some specified address.

-Exception Link Register (ELR) – Sends a code to the Operating System to handle the Error.



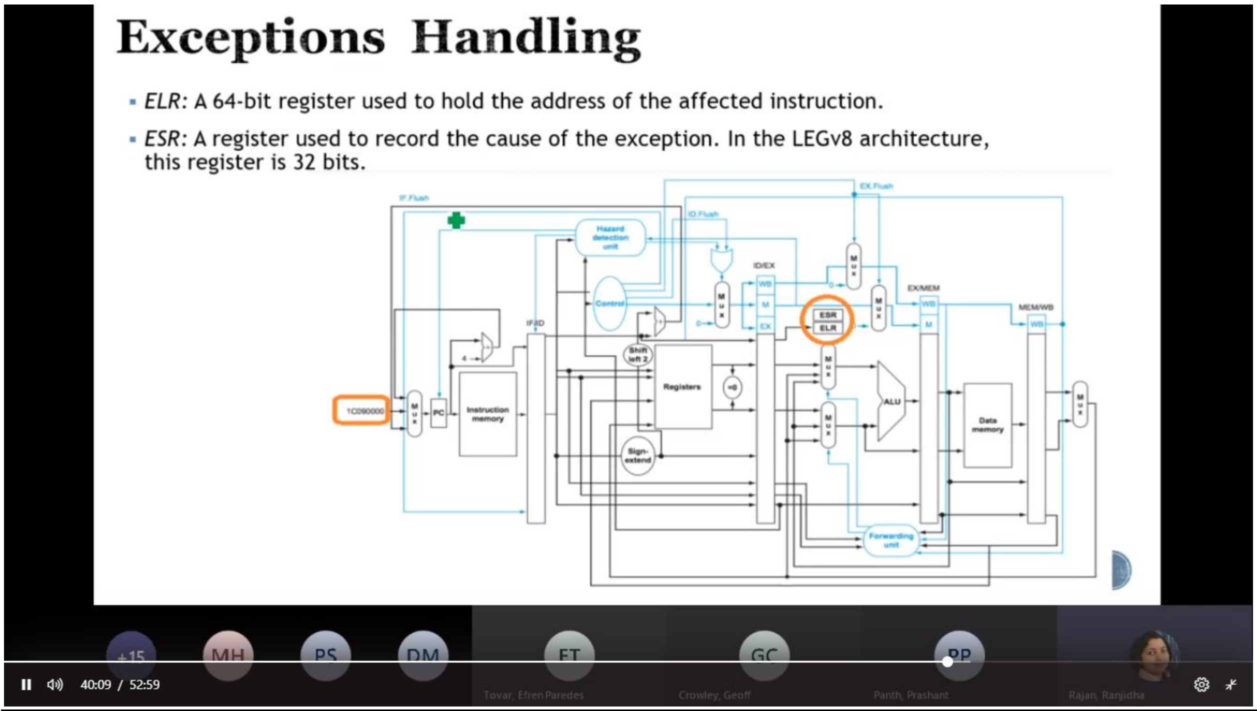
Exceptions Handling

- Operating system to handle the exception, it must know
  - The reason for the exception .
  - The instruction that caused it .
  - Register (called the *Exception Syndrome Register* or *ESR*), which holds a field that indicates the reason for the exception.
  - *Vectored interrupts*

Exception type	Exception vector address to be added to a Vector Table Base Register
Unknown Reason	00 0000$_{two}$
Floating-point arithmetic exception	10 1100$_{two}$
System Error (hardware malfunction)	10 1111$_{two}$

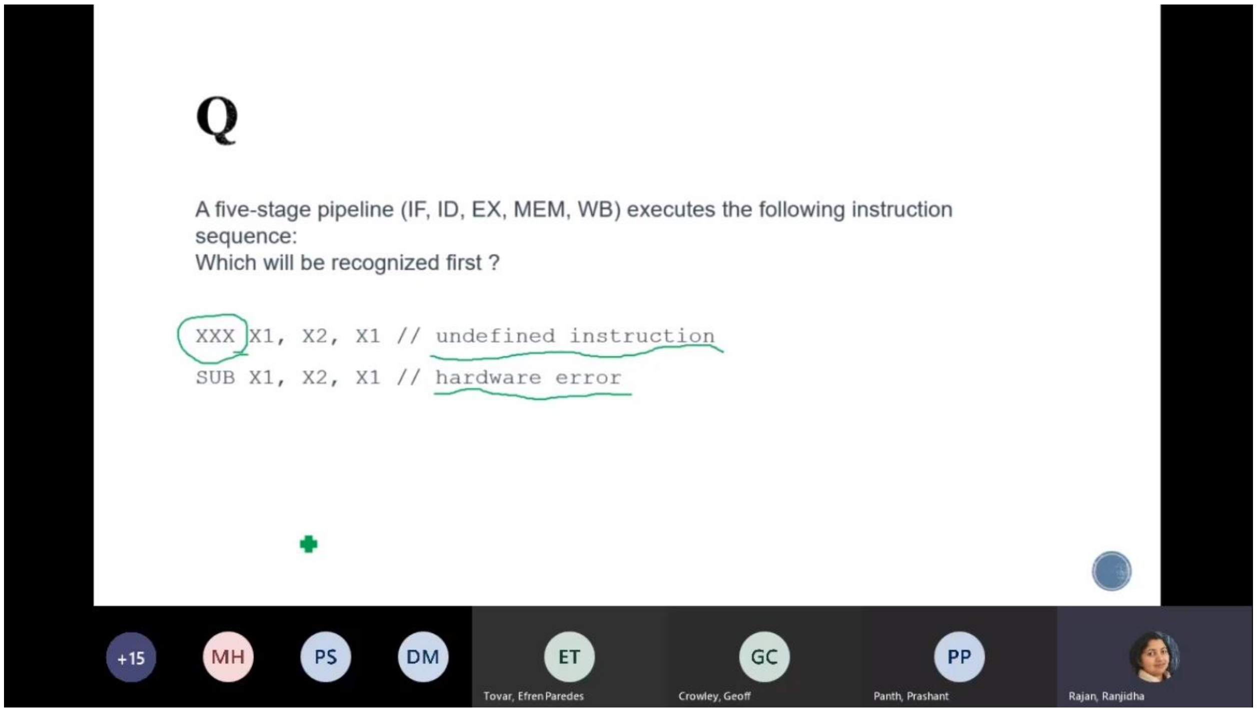
--Vector Interrupts- the code sent.

--**Exception Syndrome Register** – Holds the field that indicates the error.



Exceptions Handling

- *ELR*: A 64-bit register used to hold the address of the affected instruction.
- *ESR*: A register used to record the cause of the exception. In the LEGv8 architecture, this register is 32 bits.

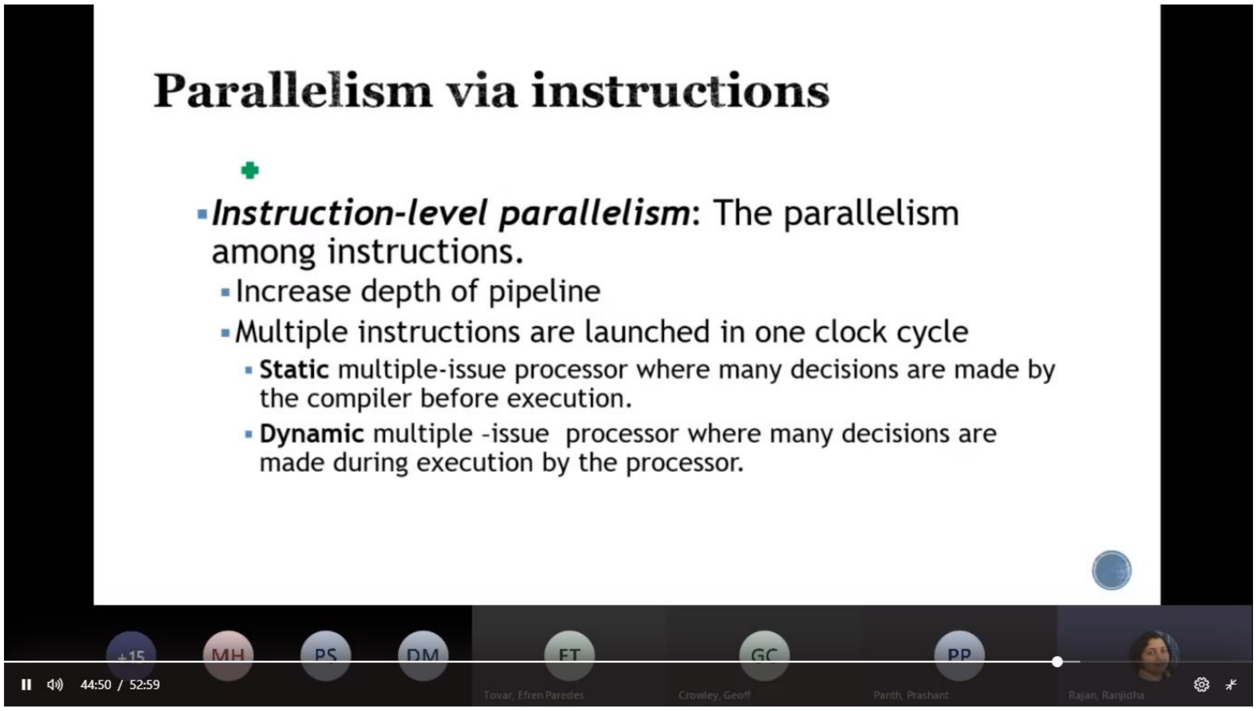


Q

A five-stage pipeline (IF, ID, EX, MEM, WB) executes the following instruction sequence:
Which will be recognized first ?

```
XXX X1, X2, X1 // undefined instruction
SUB X1, X2, X1 // hardware error
```

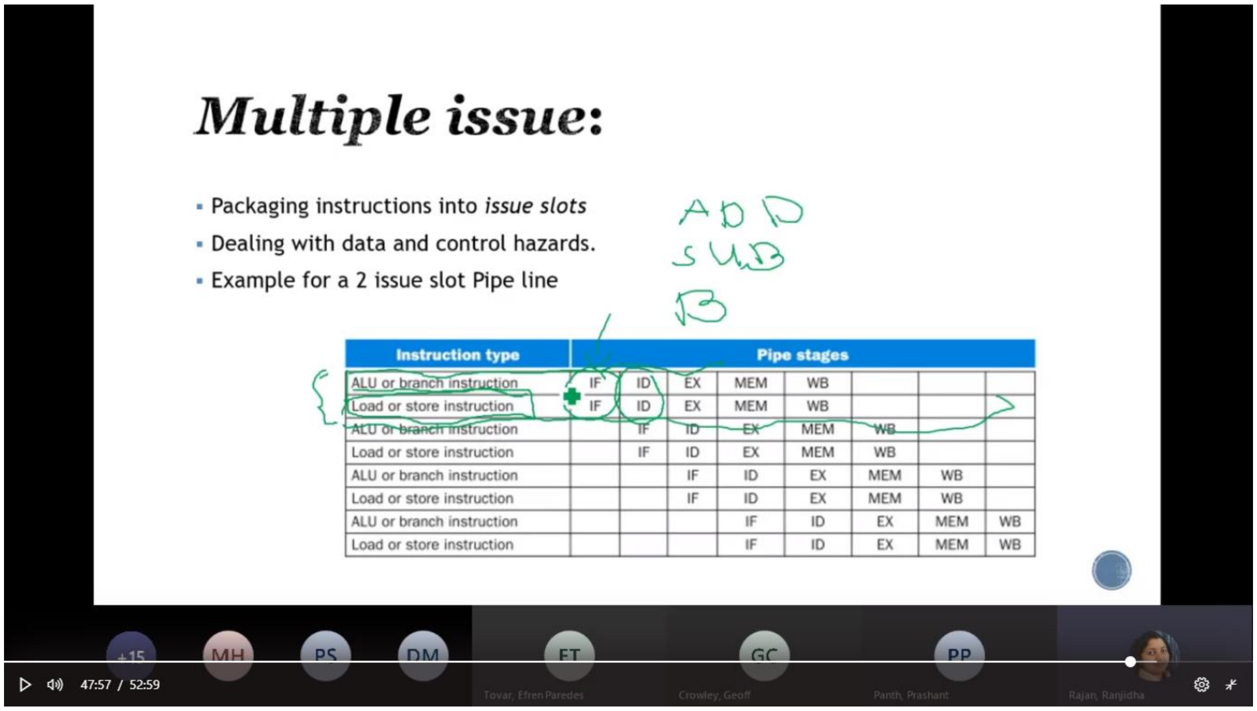
Undefined Instructions- It is missing an opcode, which is the first process of the pipeline.



Parallelism via instructions

- *Instruction-level parallelism*: The parallelism among instructions.
  - Increase depth of pipeline
  - Multiple instructions are launched in one clock cycle
    - **Static** multiple-issue processor where many decisions are made by the compiler before execution.
    - **Dynamic** multiple -issue processor where many decisions are made during execution by the processor.

--Parallelism- Multiple Instructions in 1 clock cycle.
--**Static** Para.- Decisions are made before execution by the Compiler.
--**Dynamic** Para. – Decisions are made during execution by the processor.



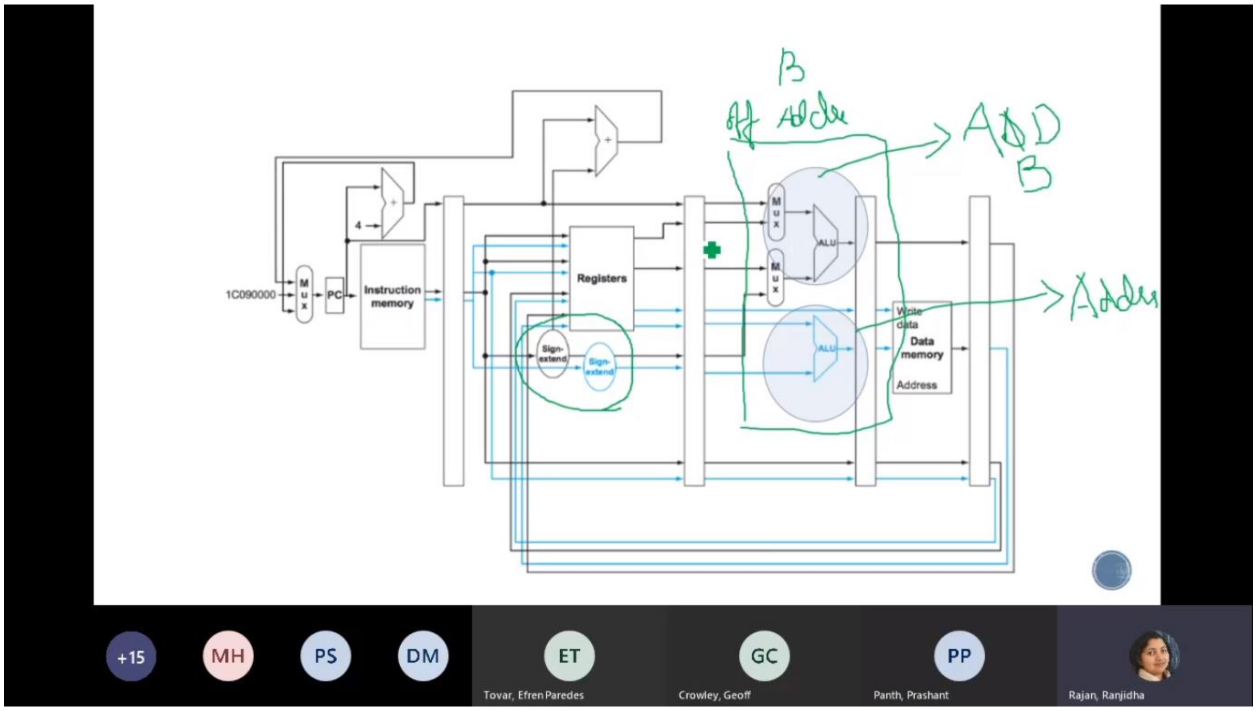
Multiple issue:

- Packaging instructions into *issue slots*
- Dealing with data and control hazards.
- Example for a 2 issue slot Pipe line

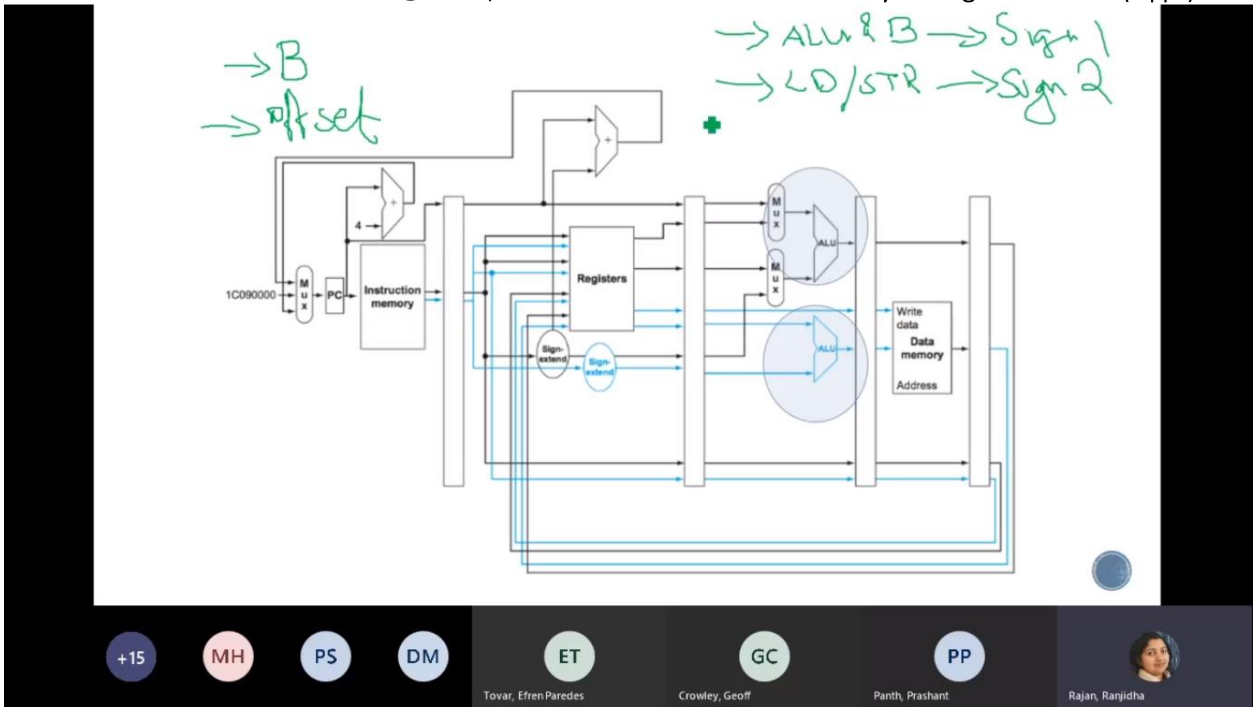
ADD
SUB
B

Instruction type	Pipe stages							
ALU or branch instruction	IF	ID	EX	MEM	WB			
Load or store instruction	IF	ID	EX	MEM	WB			
ALU or branch instruction		IF	ID	EX	MEM	WB		
Load or store instruction		IF	ID	EX	MEM	WB		
ALU or branch instruction			IF	ID	EX	MEM	WB	
Load or store instruction			IF	ID	EX	MEM	WB	
ALU or branch instruction				IF	ID	EX	MEM	WB
Load or store instruction				IF	ID	EX	MEM	WB

--Running many processes at once can cause a Multiple issue.



--Need 2 ALU's for 2 Instructions @ once, Instructions are discriminated by its' Sign Extension.(1||2)



04/02/2020 <=> Online EXAM 2 & Programming EXAM w/Arrays & Stack **CHAPTER 4.1-4.9**

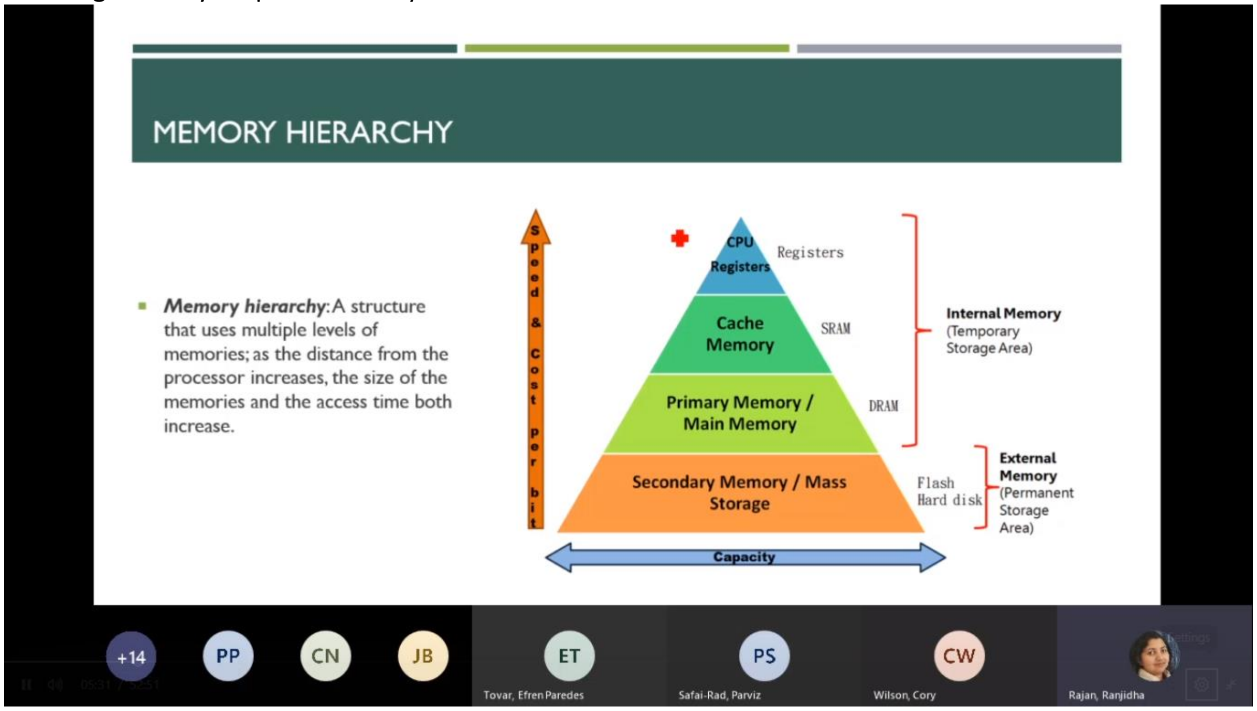
4/9/2020 CHAPTER 05 **MEMORY**

Locality of Reference- Spatial & Temporal Locality.

Spatial - Based on Location of the data, groups data together.

Temporal - Based on History, recent data will probably be reused.

Iterating an array is Spatial Locality.

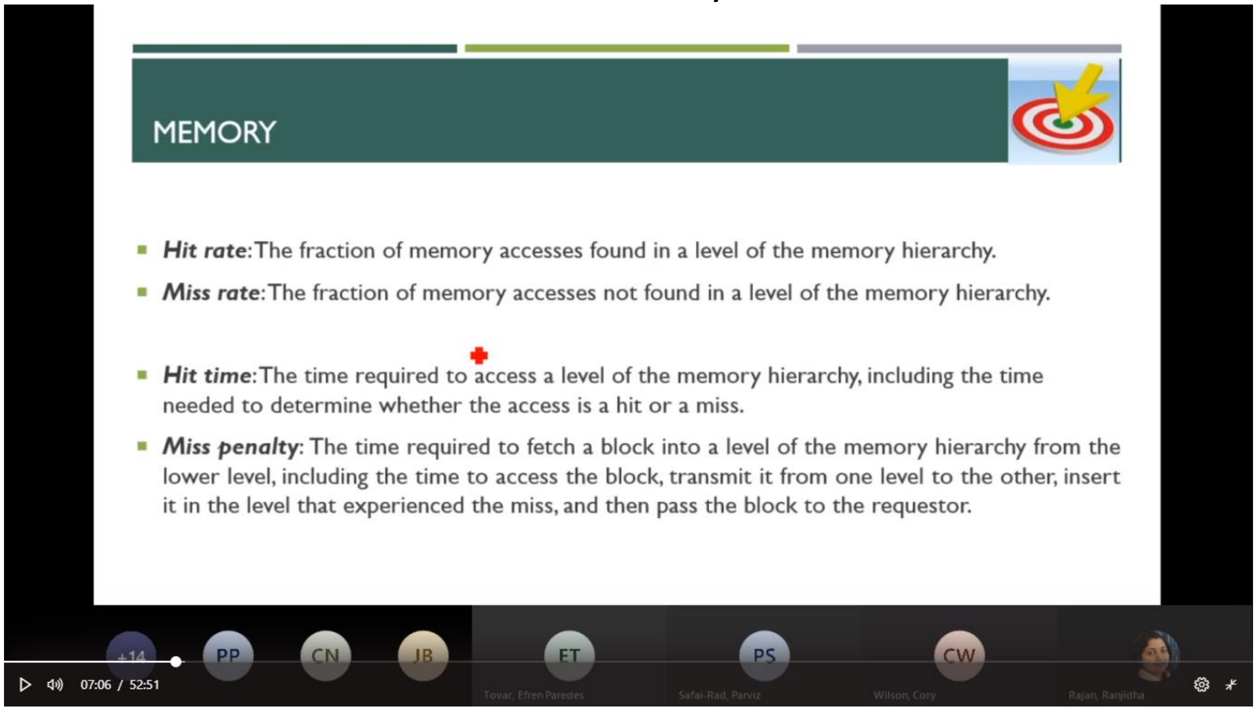


MEMORY HIERARCHY

- *Memory hierarchy*: A structure that uses multiple levels of memories; as the distance from the processor increases, the size of the memories and the access time both increase.

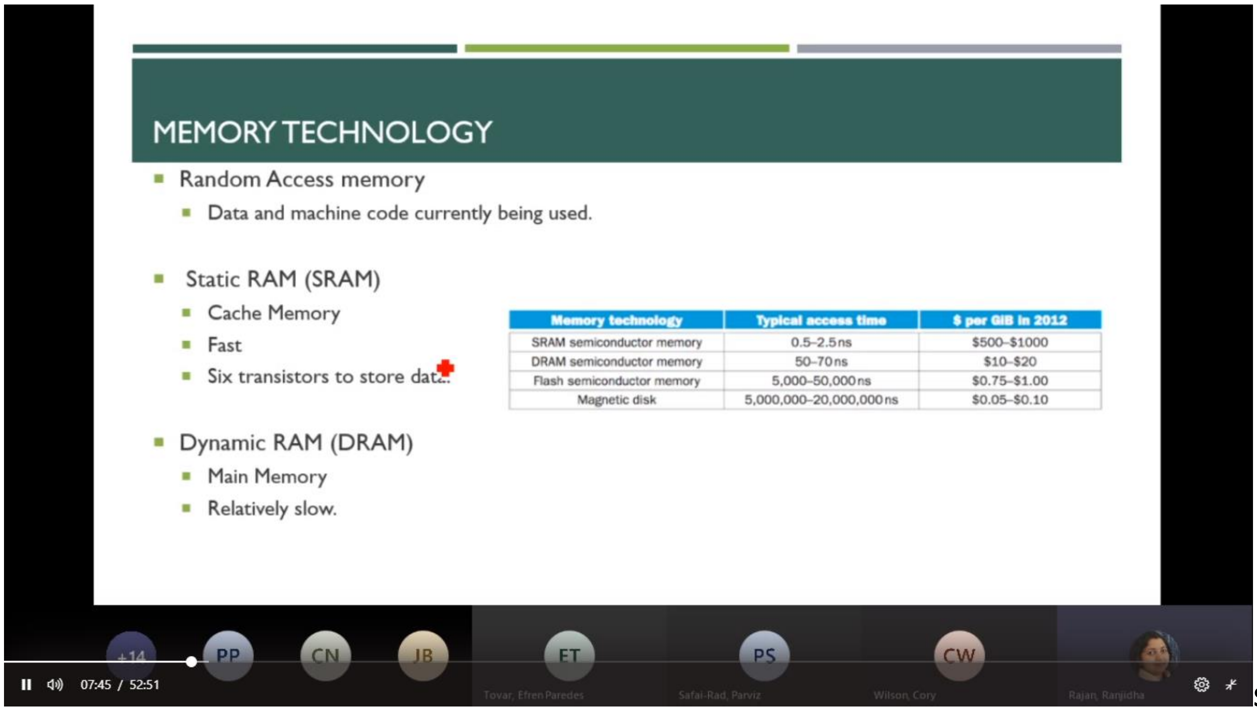
MEMORY HIERACHY- Bigger & Slower || Smaller & Faster

Hit Rate- **Miss Rate**- **Hit Time**- **Miss Penalty**-



MEMORY

- *Hit rate*: The fraction of memory accesses found in a level of the memory hierarchy.
- *Miss rate*: The fraction of memory accesses not found in a level of the memory hierarchy.
- *Hit time*: The time required to access a level of the memory hierarchy, including the time needed to determine whether the access is a hit or a miss.
- *Miss penalty*: The time required to fetch a block into a level of the memory hierarchy from the lower level, including the time to access the block, transmit it from one level to the other, insert it in the level that experienced the miss, and then pass the block to the requestor.

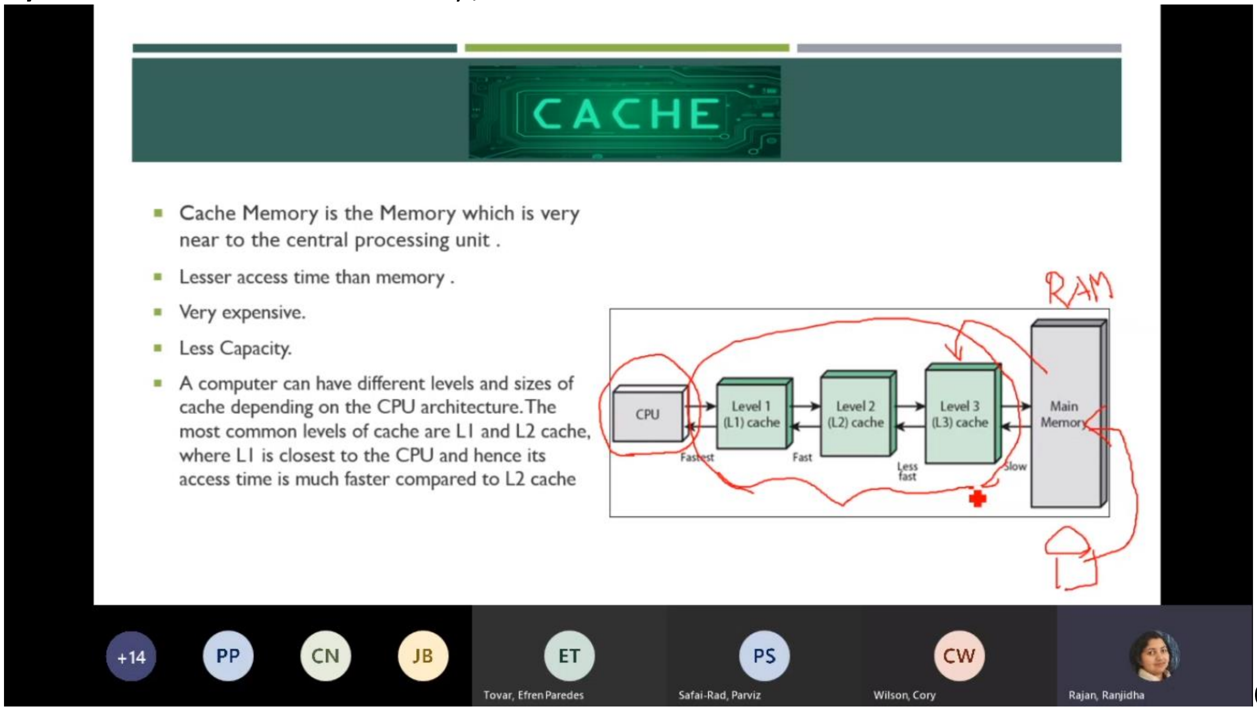


MEMORY TECHNOLOGY

- Random Access memory
  - Data and machine code currently being used.
- Static RAM (SRAM)
  - Cache Memory
  - Fast
  - Six transistors to store data.
- Dynamic RAM (DRAM)
  - Main Memory
  - Relatively slow.

Memory technology	Typical access time	$ per GiB in 2012
SRAM semiconductor memory	0.5–2.5 ns	$500–$1000
DRAM semiconductor memory	50–70 ns	$10–$20
Flash semiconductor memory	5,000–50,000 ns	$0.75–$1.00
Magnetic disk	5,000,000–20,000,000 ns	$0.05–$0.10

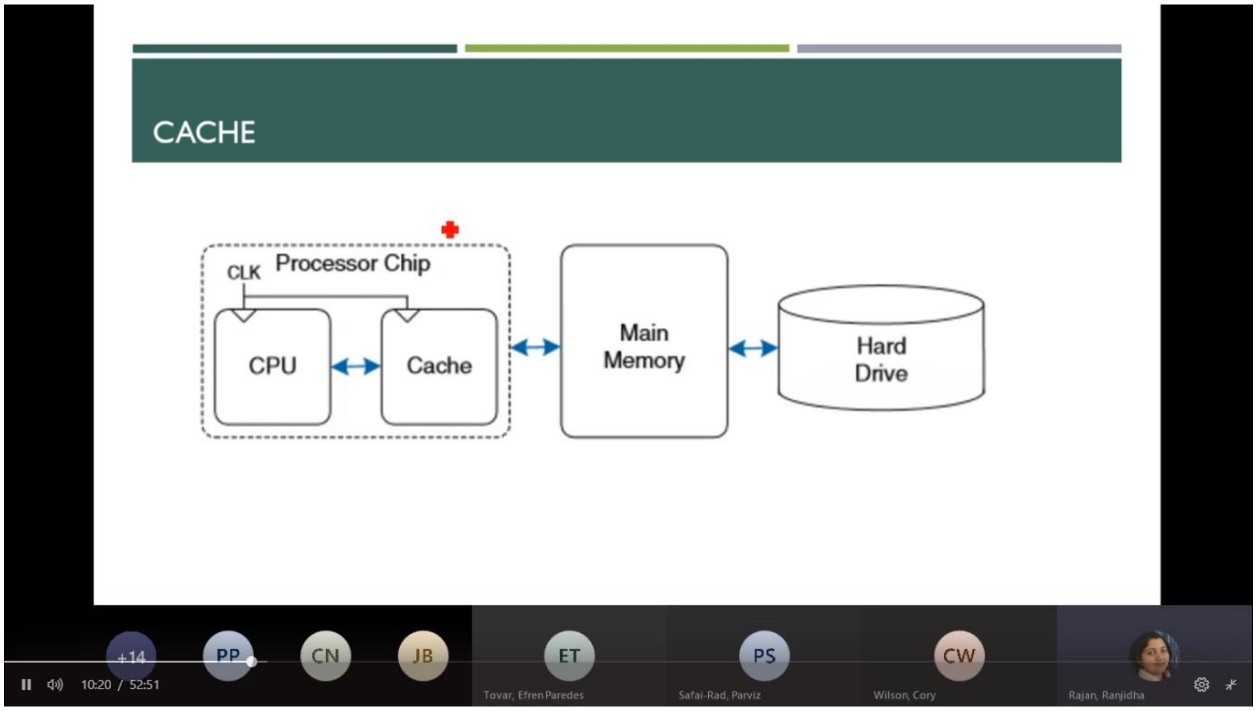
Static RAM=SRAM =Cache Memory / Faster
Dynamic RAM – DRAM- Main Memory / Slower



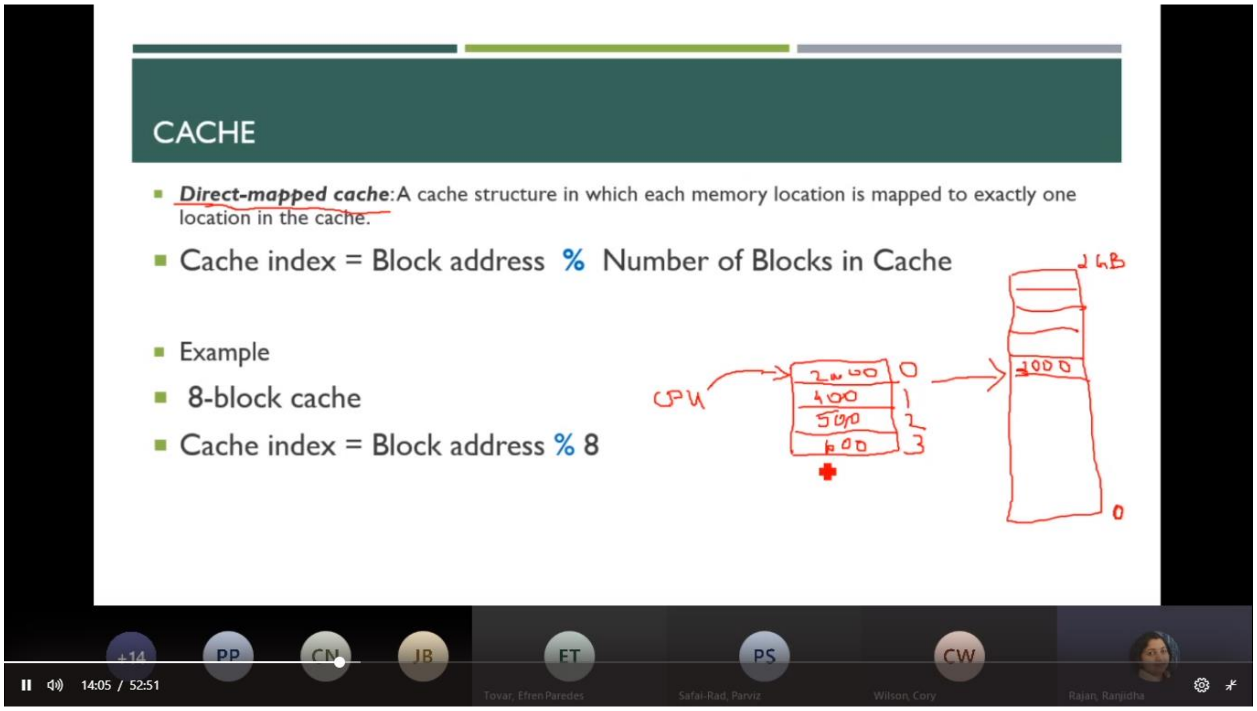
CACHE

- Cache Memory is the Memory which is very near to the central processing unit .
- Lesser access time than memory .
- Very expensive.
- Less Capacity.
- A computer can have different levels and sizes of cache depending on the CPU architecture. The most common levels of cache are L1 and L2 cache, where L1 is closest to the CPU and hence its access time is much faster compared to L2 cache

CACHE- Nearest Memory to the CPU, besides Registers in the CPU. Level of Cache: L1,L2,L3


CACHE

Direct Mapped Cache


CACHE

- *Direct-mapped cache*: A cache structure in which each memory location is mapped to exactly one location in the cache.
- Cache index = Block address % Number of Blocks in Cache
- Example
- 8-block cache
- Cache index = Block address % 8

^ 4-Lined Cache ^ Address %= 4;

-Address with an 8 Lined Cache, needs an Index of: 2^3 so, **3 Index Bits.**

-TERMINOLOGY= Byte = Word

Cache		**Main Mem**		**Process**
Lines	=	Frame/Blocks	=	Pages

-**# of Blocks** = Cache Size / Block Size **EX)** Cache Size = 128 Bytes / Block Size = 8 Bytes = 16 Blocks

-**# of Sets** = # of Blocks / Set Size **EX)** 16 Blocks / Set Size = 2 Lines = 8 Sets (of 2 Lines)

-**Cache Index** = Block Address / # of Blocks EX) 1001 = 9 9 % 16 =9

-

-

@ 33:59 / 44:05 1st VIDEO 04/16/20

4/16/2020

4/16/2020 Video2