

Topic : Pipelining

Prepared by Mrs . Kalpana Pawase

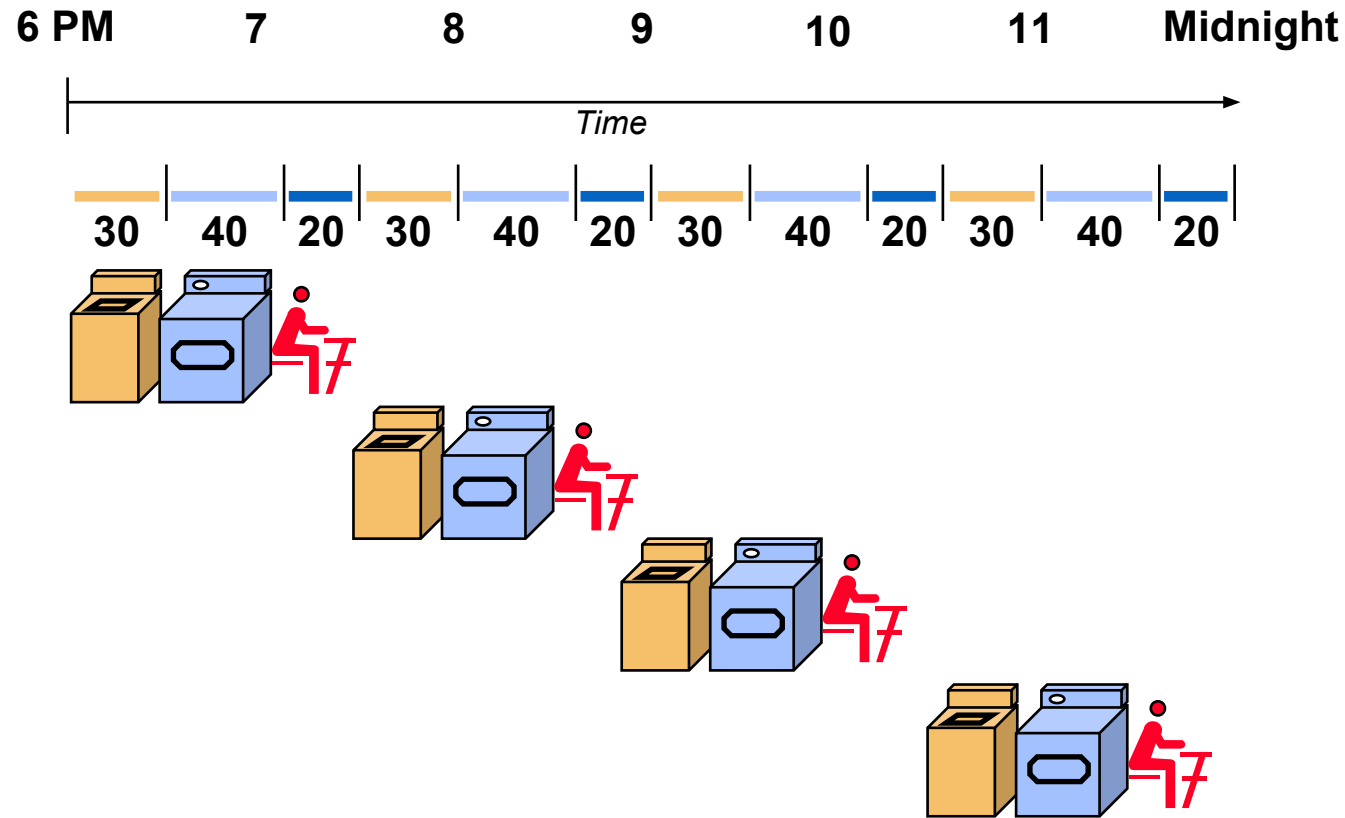
Definition

- A Pipelining is a series of stages, where some work is done at **each stage in parallel**.
- The stages are connected one to the next to form a pipe-instructions enter at one end, progress through the stages , and exit at the other end.

Pipelining Case : Laundry

- 4 loads of laundry that need to be washed, dried and folded
 - 30 minutes to wash, 40 minutes to dry, and 20 min to fold
 - We have 1 washer, 1 dryer and 1 folding station.
- What's the most efficient way to get the 4 loads of laundry done?

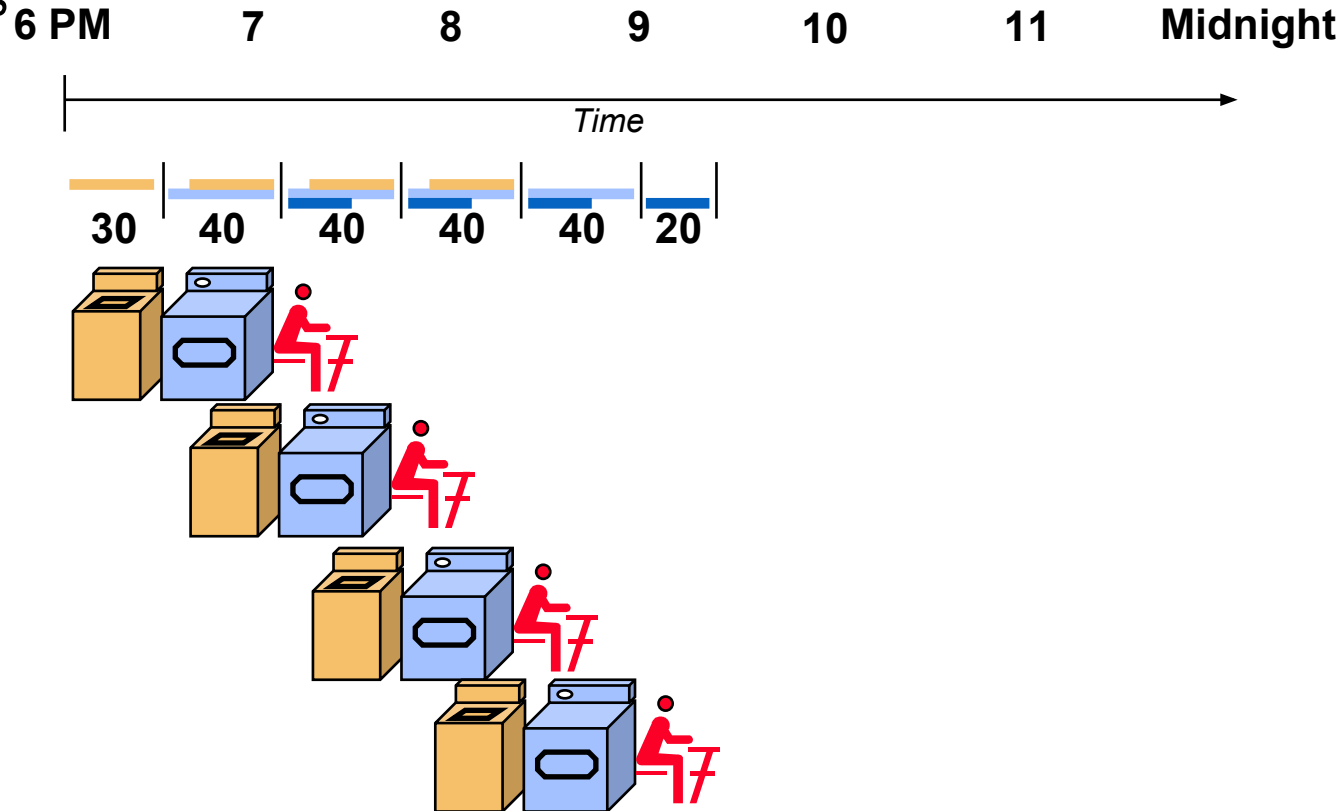
The slow way



- If each load is done sequentially it takes 6 hours

Laundry Pipelining

- Start each load as soon as possible
 - Overlap loads



- Pipelined laundry takes 3.5 hours

Definition :

Pipelining is a speed up technique where multiple instructions are overlapped in execution on a processor.

Pipelining: Processors

Computers like laundry , typically perform the exact same steps for every instruction.

- Fetch an instruction from memory
- Decode the instruction
- Execute the instruction
- Read memory to get input
- Write the result back to memory

Instruction Fetch

- The IF stage is responsible for obtaining the requested instruction from memory. The instruction and the program counter are stored in the register as temporary storage.

Decode Instruction

The DI stage is responsible for decoding the instruction and sending out the various control lines to the other parts of the processor.

Calculate Operands

The CO stage is where any calculations are performed

The main component in this stage is the ALU is made up of arithmetic ,logic capabilities

Fetch Operands and Execute Instruction

The FO and EI stages are responsible for storing and loading values to and from memory . They also responsible for input and output from the processor respectively.

Write Operands

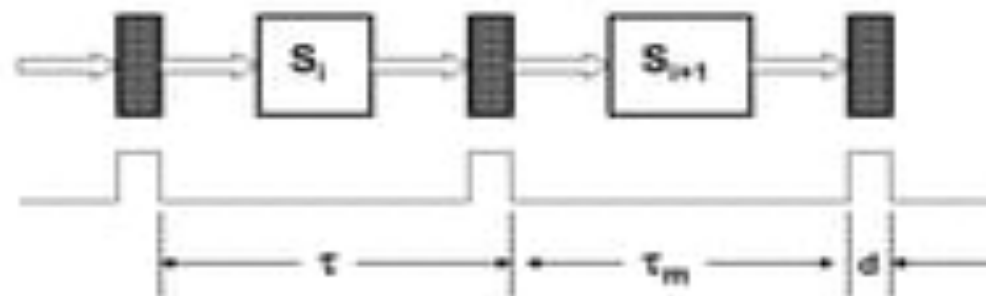
The WO stage is responsible for writing the result of a calculation , memory access or input into the register file.

Timing Diagram for Instruction Pipeline Operation

Time →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

Pipeline Performance: Clock & Timing



Clock cycle of the pipeline : τ

Latch delay : d

$$\tau = \max \{ \tau_m \} + d$$

Pipeline frequency : f

$$f = 1 / \tau$$

Advantages:

- Pipelining makes efficient use of resources.
- Quicker time of execution of large number of instructions.
- The parallelism is invisible to the programmer.

Instruction execution review

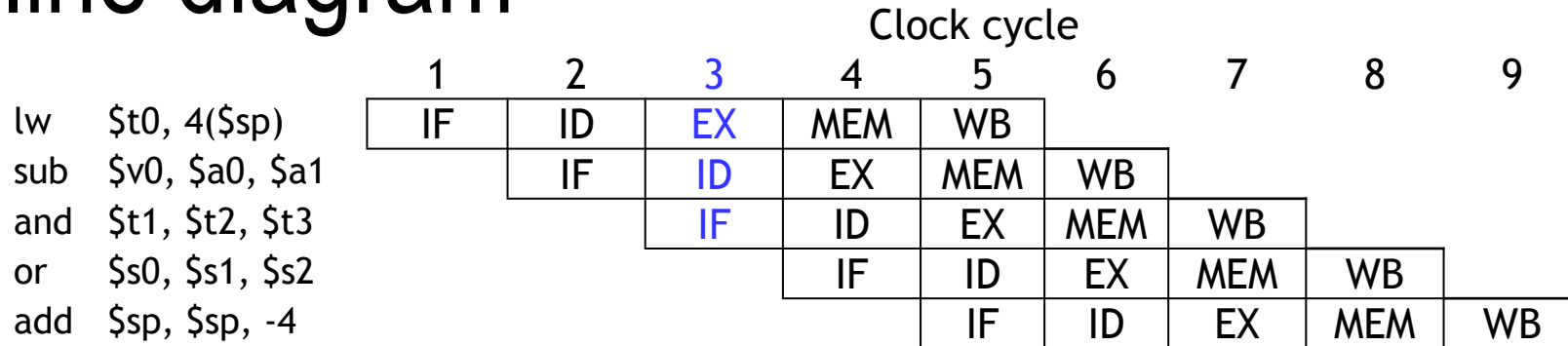
- Executing a MIPS instruction can take up to five steps.

Step	Name	Description
Instruction Fetch	IF	Read an instruction from memory.
Instruction Decode	ID	Read source registers and generate control signals.
Execute	EX	Compute an R-type result or a branch outcome.
Memory	MEM	Read or write the data memory.
Writeback	WB	Store a result in the destination register.

Instruction	Steps required				
beq	IF	ID	EX		
R-type	IF	ID	EX		WB
sw	IF	ID	EX	MEM	
lw	IF	ID	EX	MEM	WB

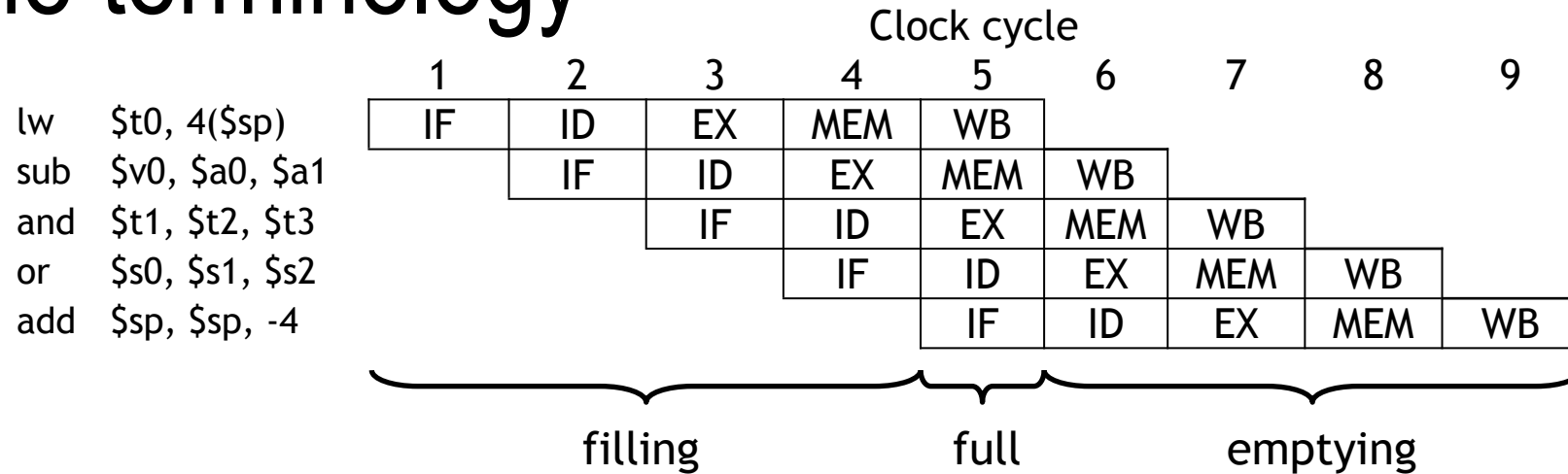
- However, as we saw, not all instructions need all five steps.

A pipeline diagram



- A **pipeline diagram** shows the execution of a series of instructions.
 - The instruction sequence is shown vertically, from top to bottom.
 - Clock cycles are shown horizontally, from left to right.
 - Each instruction is divided into its component stages. (We show five stages for every instruction, which will make the control unit easier.)
- This clearly indicates the overlapping of instructions. For example, there are three instructions active in the third cycle above.
 - The “lw” instruction is in its Execute stage.
 - Simultaneously, the “sub” is in its Instruction Decode stage.
 - Also, the “and” instruction is just being fetched.

Pipeline terminology



- The **pipeline depth** is the number of stages—in this case, five.
- In the first four cycles here, the pipeline is **filling**, since there are unused functional units.
- In cycle 5, the pipeline is **full**. Five instructions are being executed simultaneously, so all hardware units are in use.
- In cycles 6-9, the pipeline is **emptying**.

Pipeline Hazards

Limits to pipelining: Hazards prevent next instruction from executing during its designated clock cycle

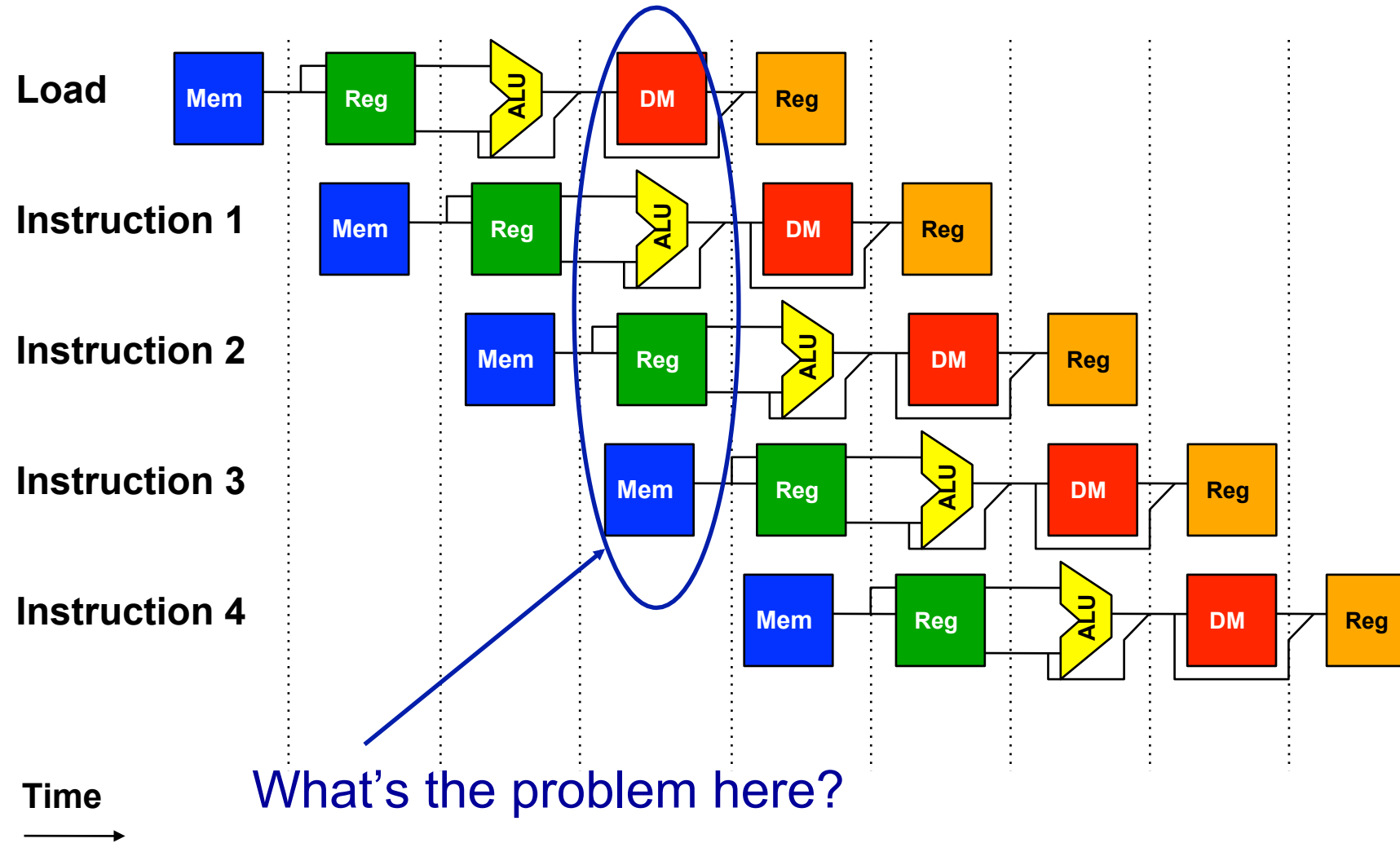
- **Structural hazards**: HW cannot support this combination of instructions (single person to fold and put clothes away)
- **Data hazards**: Instruction depends on result of prior instruction still in the pipeline (missing sock)
- **Control hazards**: Pipelining of branches & other instructions that change the PC
- Common solution is to **stall** the pipeline until the hazard is resolved, inserting one or more “**bubbles**” in the pipeline

STRUCTURAL HAZARDS

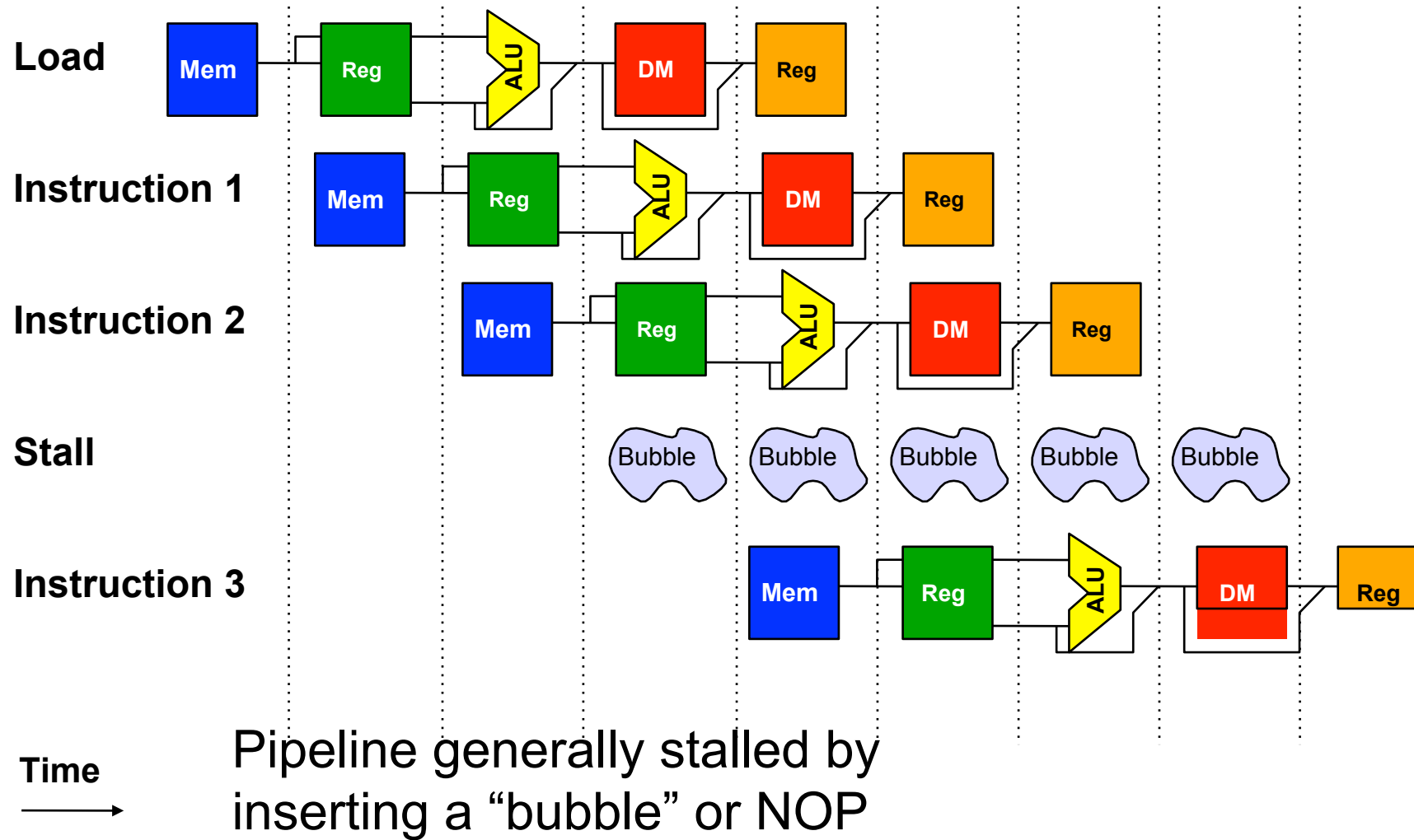
Structural hazards

- Avoid structural hazards by duplicating resources
 - e.g. an ALU to perform an arithmetic operation and an adder to increment PC
- If not all possible combinations of instructions can be executed, structural hazards occur
- Pipelines stall result of hazards, CPI increased from the usual “1”

An example of a structural hazard



How is it resolved?



Or alternatively...

	Clock Number									
Inst. #	1	2	3	4	5	6	7	8	9	10
LOAD	IF	ID	EX	MEM	WB					
Inst. i+1		IF	ID	EX	MEM	WB				
Inst. i+2			IF	ID	EX	MEM	WB			
Inst. i+3				stall	IF	ID	EX	MEM	WB	
Inst. i+4						IF	ID	EX	MEM	WB
Inst. i+5							IF	ID	EX	MEM
Inst. i+6								IF	ID	EX

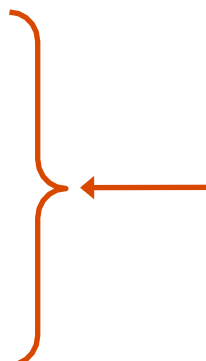
- LOAD instruction “steals” an instruction fetch cycle which will cause the pipeline to stall.
- Thus, no instruction completes on clock cycle 8

What's the realistic solution?

- **Answer: Add more hardware.**
 - (especially for the memory access example – i.e. the common case)
 - CPI degrades quickly from our ideal '1' for even the simplest of cases...

DATA HAZARDS

Data hazards

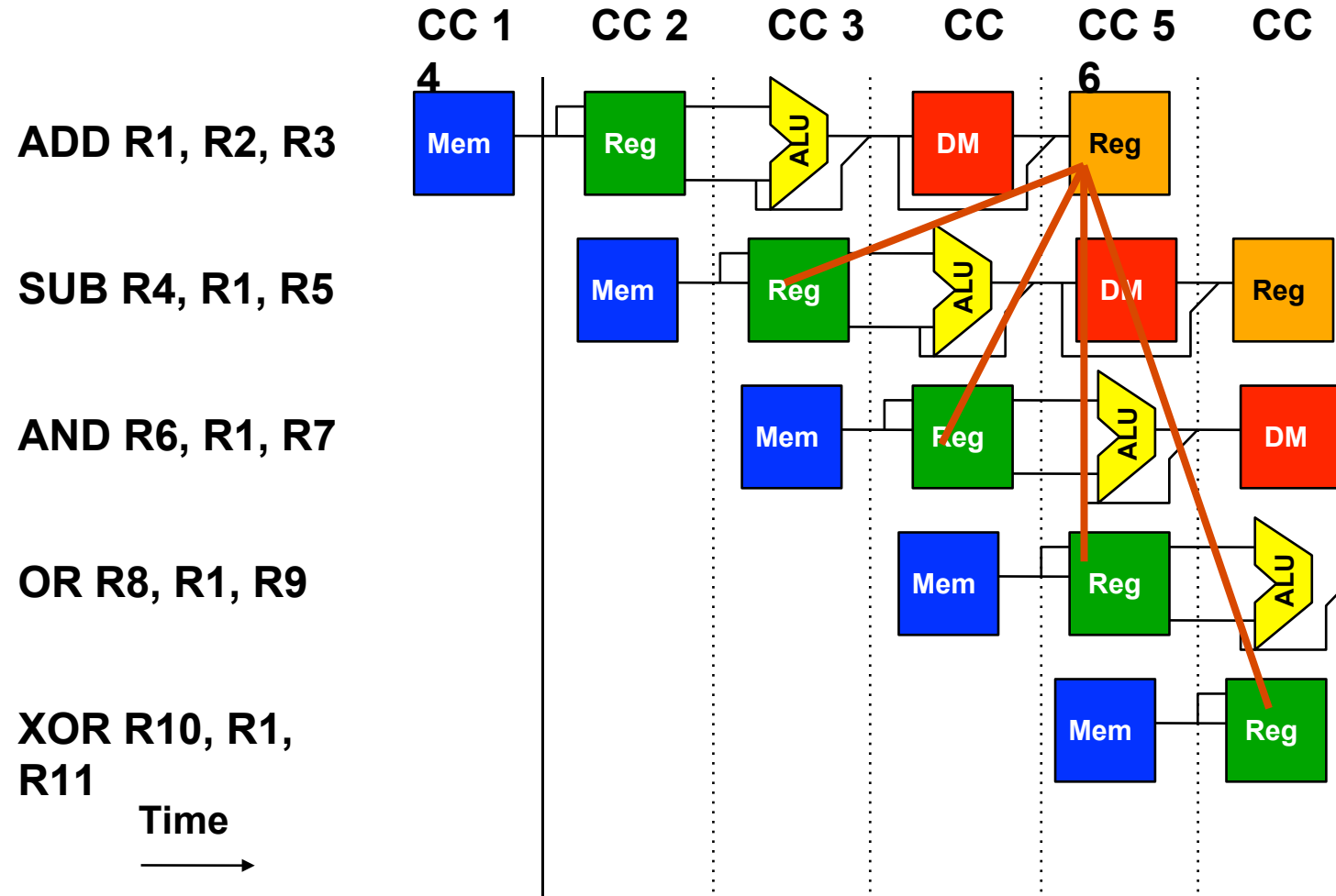
- These exist because of pipelining
- Why do they exist???
 - Pipelining changes when data operands are read, written
 - Order differs from order seen by sequentially executing instructions on un-pipelined machine
- Consider this example:
 - ADD R1, R2, R3
 - SUB R4, R1, R5
 - AND R6, R1, R7
 - OR R8, R1, R9
 - XOR R10, R1, R11

All instructions after ADD use result of ADD

ADD writes the register in WB but SUB needs it in ID.

This is a data hazard

Illustrating a data hazard

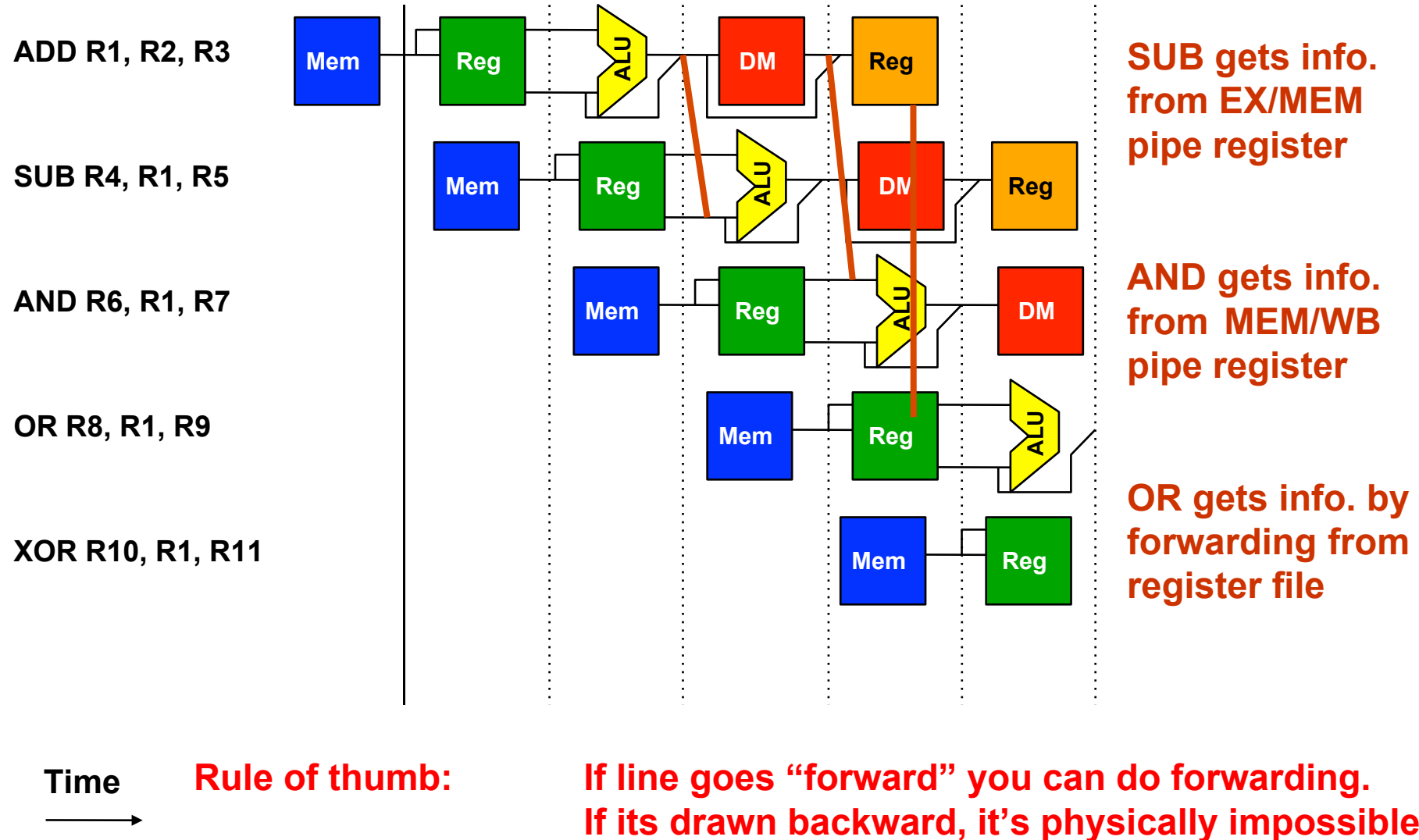


**ADD instruction causes a hazard in next 3 instructions
b/c register not written until after those 3 read it.**

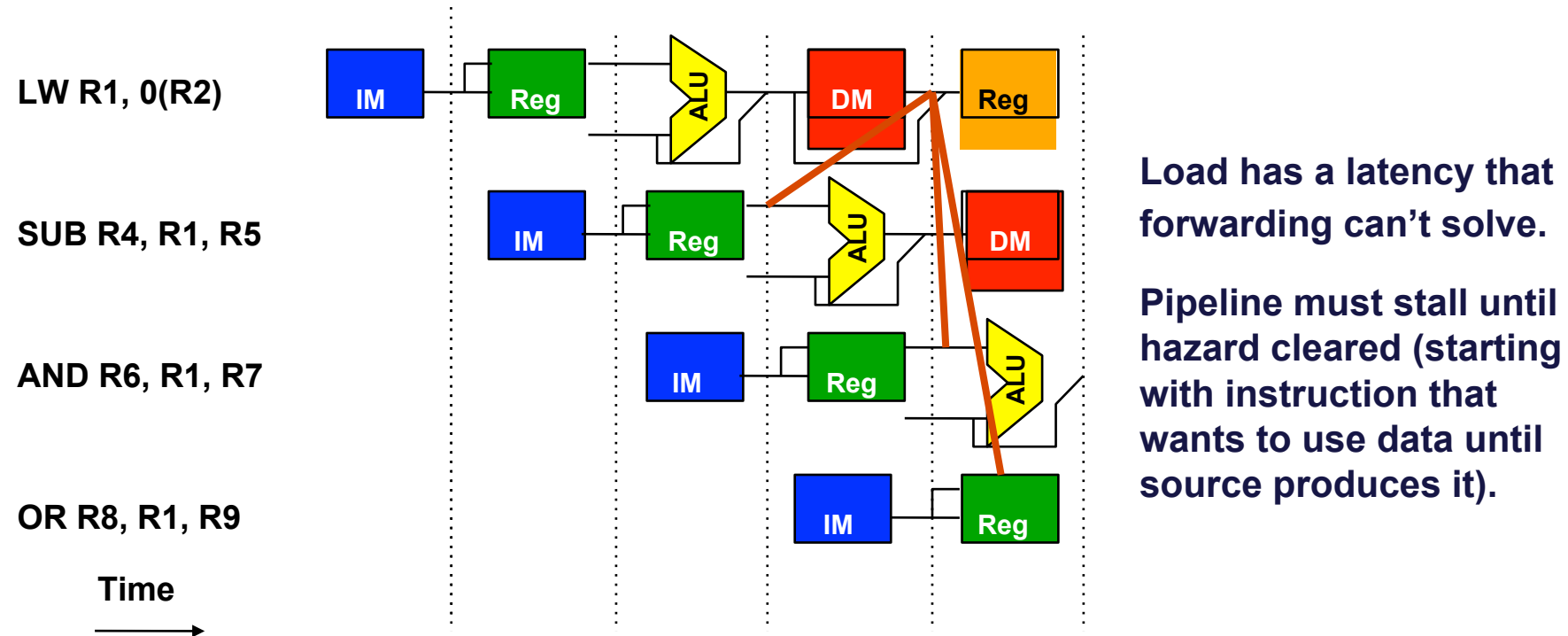
Forwarding

- Problem illustrated on previous slide can actually be solved relatively easily – **with forwarding**
- Can we move the result from EX/MEM register to the beginning of ALU (where SUB needs it)?
 - **Yes!**
- Generally speaking:
 - **Forwarding occurs when a result is passed directly to functional unit that requires it.**
 - **Result goes from output of one unit to input of another**

When can we forward?



Forwarding doesn't always work



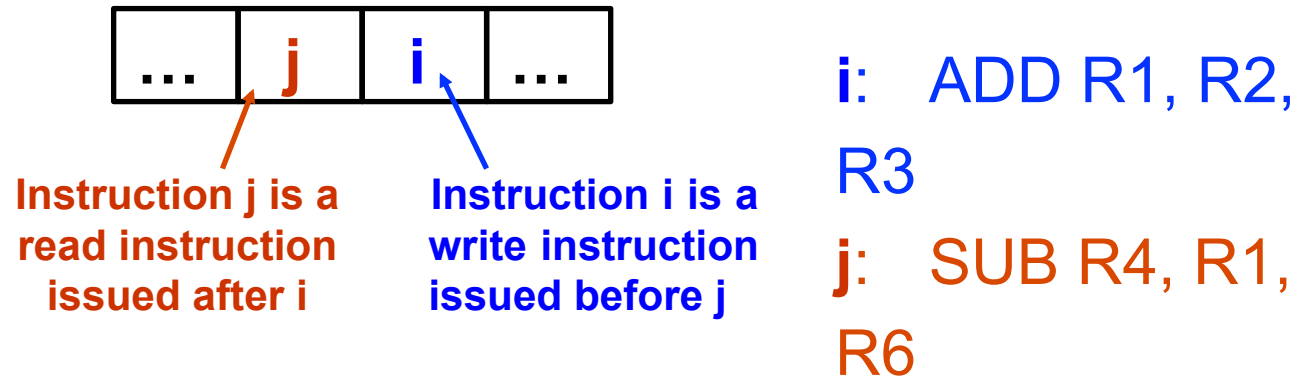
Can't get data to subtract b/c result needed at beginning of CC #4, but not produced until end of CC #4.

Data hazard specifics

- **There are actually 3 different kinds of data hazards:**
 - **Read After Write (RAW)**
 - **Write After Write (WAW)**
 - **Write After Read (WAR)**
- **With an in-order issue/in-order completion machine, we're not as concerned with WAW, WAR**

Read after write (RAW) hazards

- With RAW hazard, instruction j tries to read a source operand before instruction i writes it.
- Thus, j would incorrectly receive old or incorrect value
- Graphically/Example:



- Can use stalling or forwarding to resolve this hazard

Write after write (WAW) hazards

- I1 R3 R1*R2
- I2 R3 R4+R5
- Due to some delay if I1 take more time or parallel execution

I1 IF ID EX WB

I2 IF ID EX WB

Domain (I1) R1 R2

Domain (I2) R4 R5

Range(I1) R3

Range(I2) R3

Range(I1) \cap Range(I2) $\neq \emptyset$ **Can use stalling or forwarding to resolve this hazard**

R3

Write after Read (WAR) hazards

- I1 R1 R2+R3
- I2 R2 R4+R5
- Due to some delay if I1 take more time or parallel execution

I1 IF ID EX WB

I2 IF ID EX WB

Domain (I1) R2 R3

Domain (I2) R4 R5

Range(I1) R1

Range(I2) ~~R2~~ R2

Domain(I1) \cap Range(I2) $\neq 0$

R2 R3

Data hazards and the compiler

- **Compiler should be able to help eliminate some stalls caused by data hazards**
- **i.e. compiler could not generate a LOAD instruction that is immediately followed by instruction that uses result of LOAD's destination register.**

What about control logic?

- For MIPS integer pipeline, all data hazards can be checked during ID phase of pipeline
- If data hazard, instruction stalled before its issued
- Whether forwarding is needed can also be determined at this stage, controls signals set
- If hazard detected, control unit of pipeline must stall pipeline and prevent instructions in IF, ID from advancing

Some example situations

Situation	Example	Action
No Dependence	LW R1, 45(R2) ADD R5, R6, R7 SUB R8, R6, R7 OR R9, R6, R7	No hazard possible because no dependence exists on R1 in the immediately following three instructions.
Dependence requiring stall	LW R1, 45(R2) ADD R5, R1, R7 SUB R8, R6, R7 OR R9, R6, R7	Comparators detect the use of R1 in the ADD and stall the ADD (and SUB and OR) before the ADD begins EX
Dependence overcome by forwarding	LW R1, 45(R2) ADD R5, R6, R7 SUB R8, R1, R7 OR R9, R6, R7	Comparators detect the use of R1 in SUB and forward the result of LOAD to the ALU in time for SUB to begin with EX
Dependence with accesses in order	LW R1, 45(R2) ADD R5, R6, R7 SUB R8, R6, R7 OR R9, R1, R7	No action is required because the read of R1 by OR occurs in the second half of the ID phase, while the write of the loaded data occurred in the first half.

CONTROL HAZARDS

Branch / Control Hazards

- Also need to consider hazards involving branches:

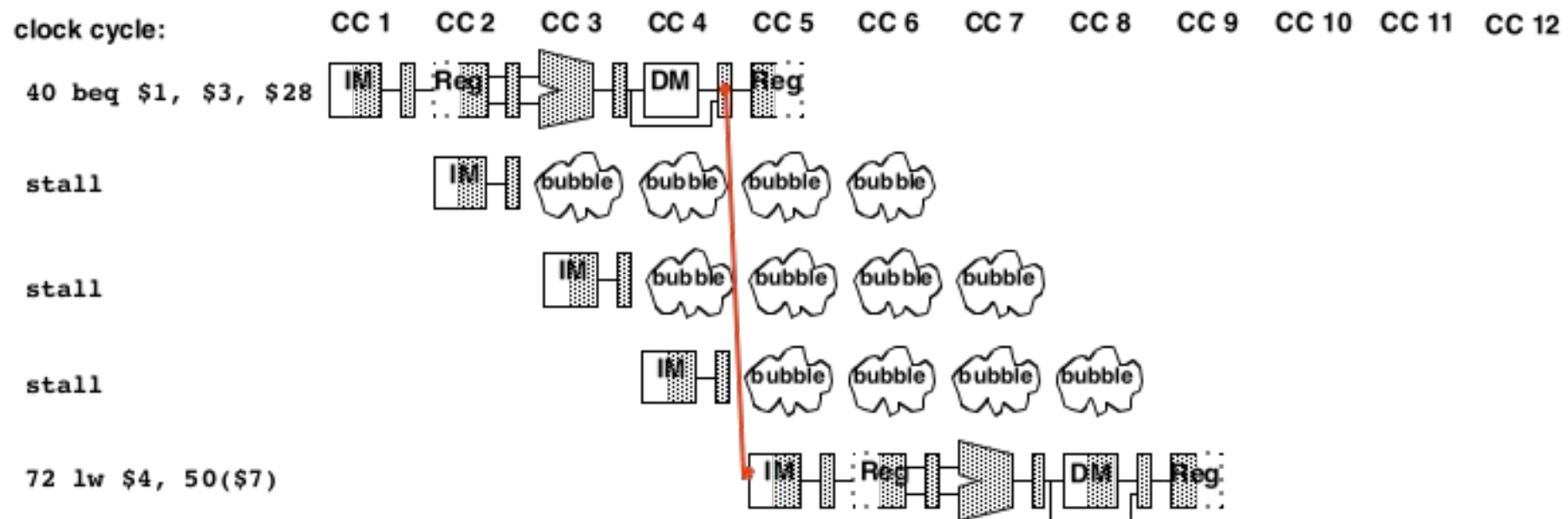
- **Example:**

- 40: beq \$1, \$3, 28 # (28 leads to address 72)
- 44: and \$12, \$2, \$5
- 48: or \$13, \$6, \$2
- 52: add \$14, \$2, \$2
- 72: lw \$4, 50(\$7)

- How long will it take before the branch decision takes effect?
 - What happens in the meantime?

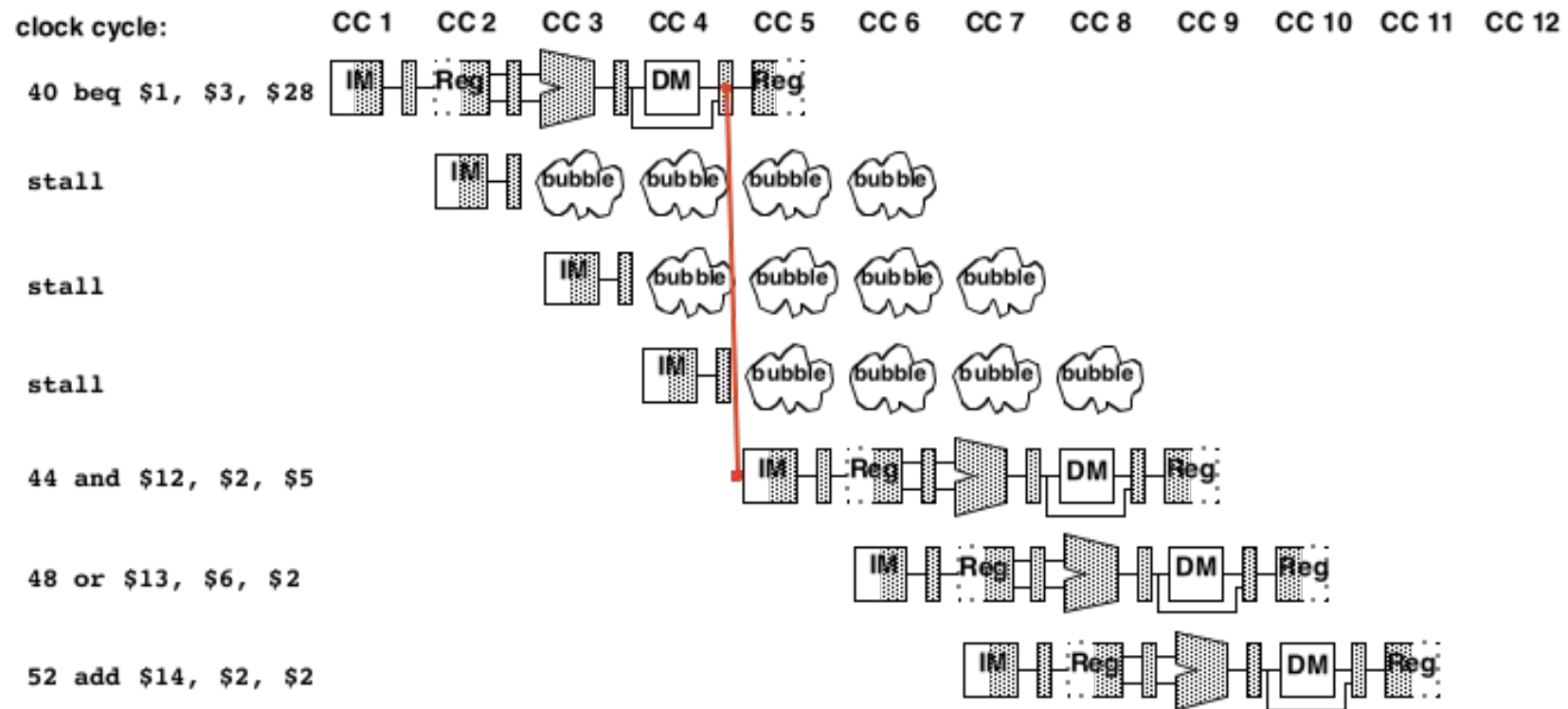
Dealing w/branch hazards: always stall

- **Branch taken**
 - **Wait 3 cycles**
 - **No proper instructions in the pipeline**
 - **Same delay as without stalls (no time lost)**



Dealing w/branch hazards: always stall

- **Branch not taken**
 - Still must wait 3 cycles
 - Time lost
 - Could have spent CCs fetching, decoding next instructions

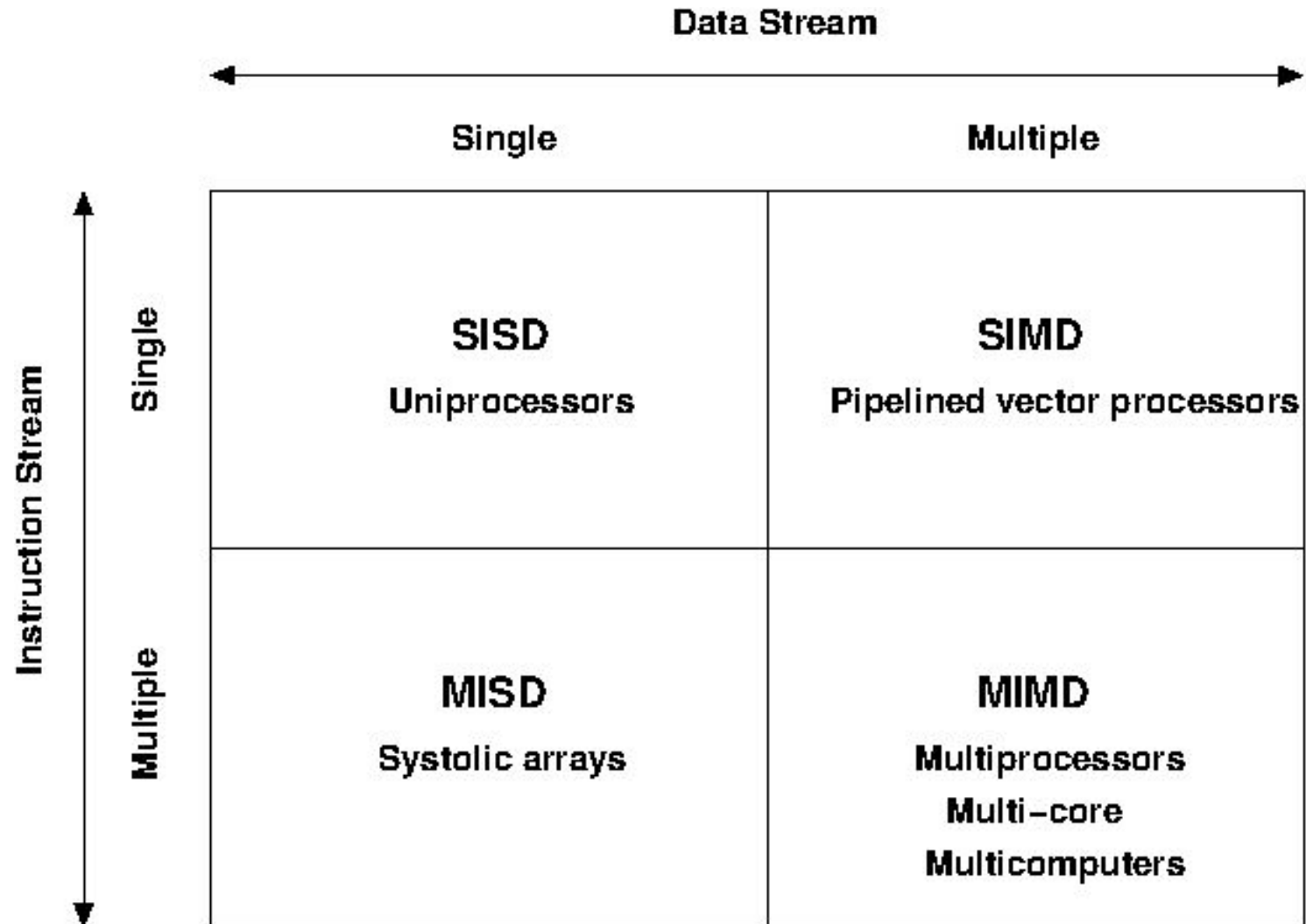


Processor Architecture/Presentation on Flynn's Classification

Flynn's Taxonomy

- Flynn's Taxonomy uses two basic concepts: Parallelism in instruction stream, and parallelism in data stream.
- A n CPU system has n program counter, so there are n "instruction stream" that can execute in parallel.
- A data stream can be used as a sequence of data, and there exist 4 possible combinations.

Flynn's Classification

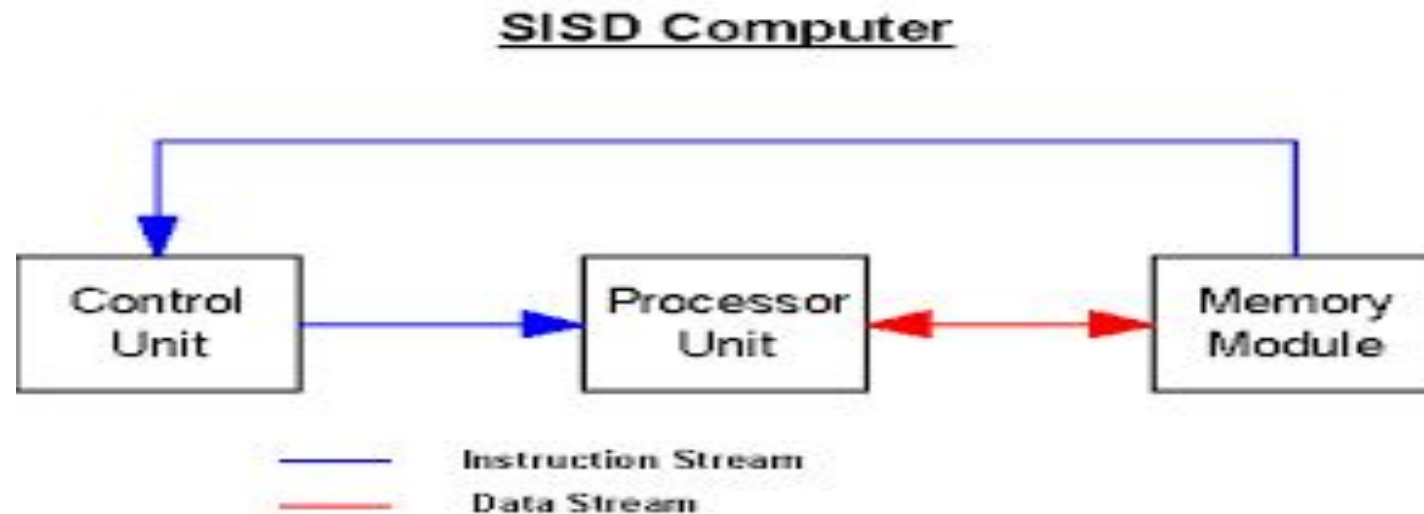


Flynn's Classification

Insrtuction Set	Data Streams	Name	Examples
1	1	SISD	Von Neumann Machine
1	Multiple	SIMD	Vector Super Computer
Multiple	1	MISD	Arguably None
Multiple	Multiple	MIMD	Multiprocessor, Multicomputer

SISD(Single Instruction Single Data)

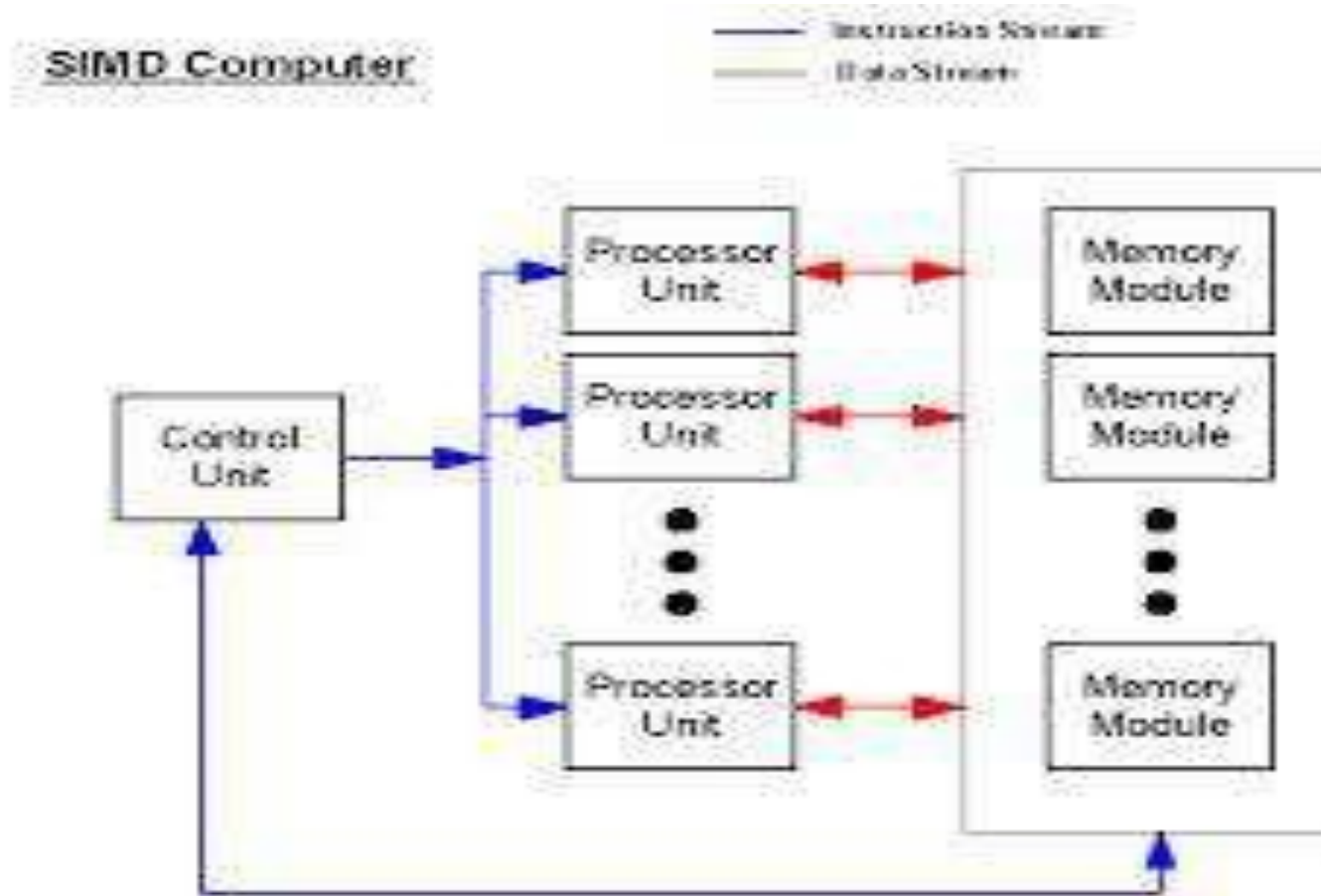
- A processor that can only do one job at a time from start to finish.



SIMD(Single Instruction Multiple Data)

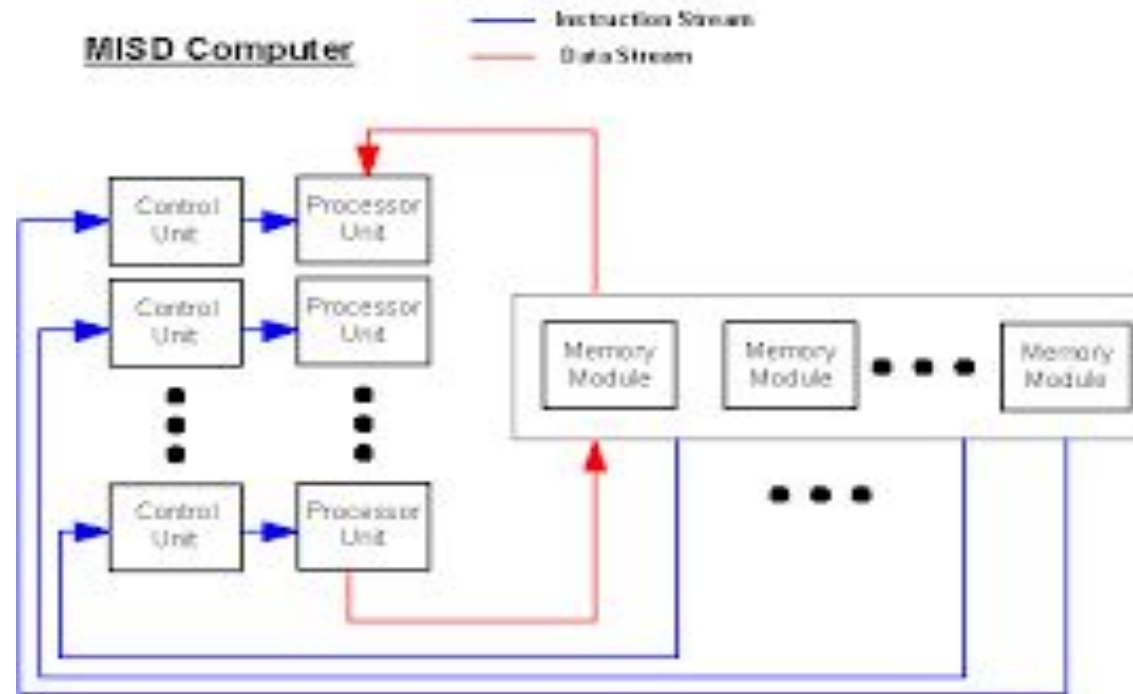
- Single CU and multiple PEs
- CU fetches an instruction from memory and after decoding, broadcasts control signals to all PEs.
- That is, at any given time, all PEs are Synchronously executing the same.
- Instruction but on different sets of data; hence the name SIMD

SIMD(Single Instruction Multiple Data)



MISD(Multiple instructions single data)

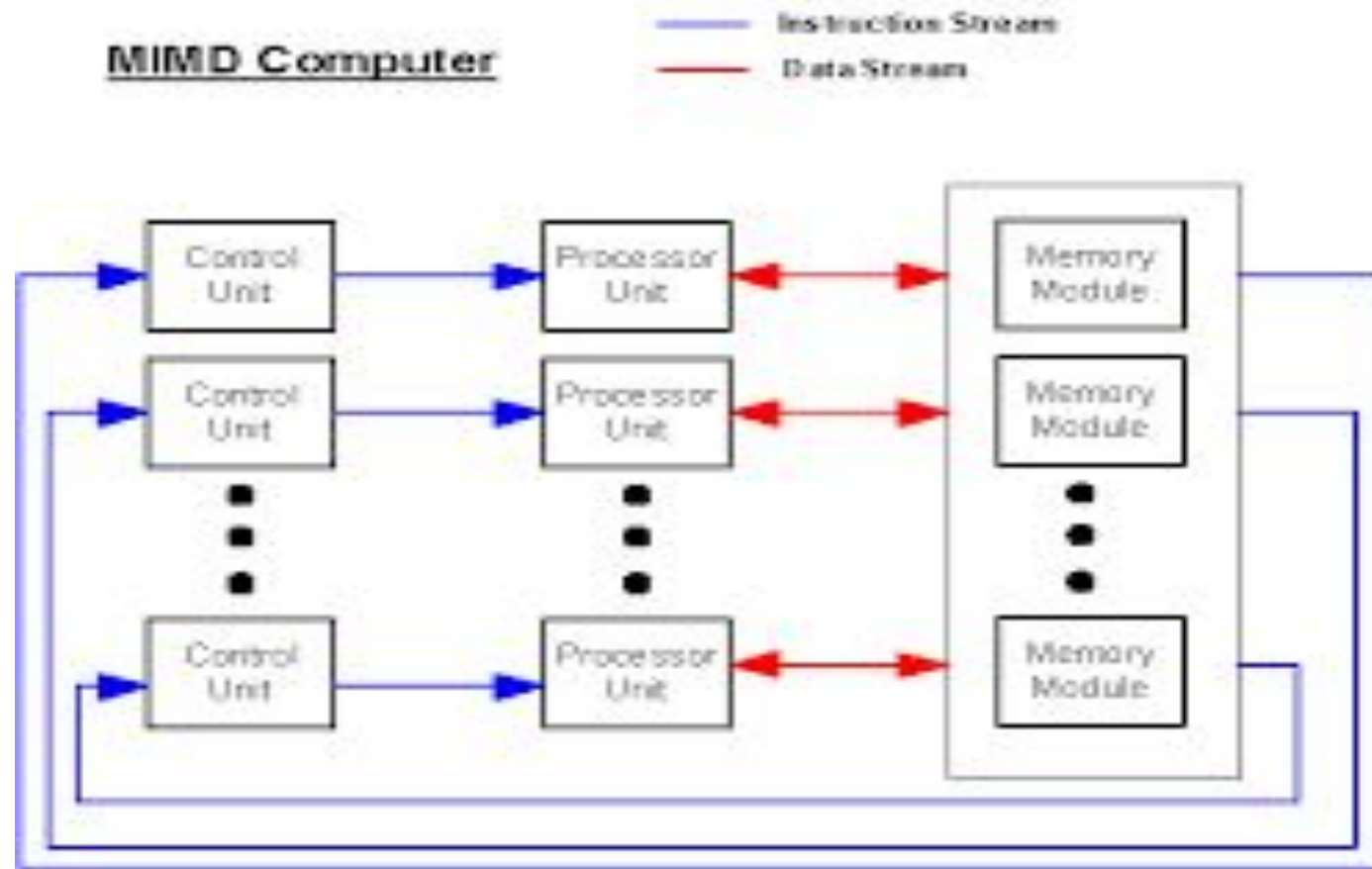
- A rare type, since data throughput is limited.



MIMD(Multiple instruction Multiple Data)

- A MIMD is a true multiprocessor
- In contrast to SIMD, a MIMD is a general-purpose machine.
- When all the processor in MIMD are running the same program, we call it Single Program Multiple Data(SPMD) computation.
- The SPMD model is widely used by many parallel platforms.

MIMD(Multiple instruction Multiple Data)



Thank you