

Chapter-4

(How instructions are being executed inside processor?)

$$\text{CPU time} = \underbrace{\text{Instruction count}}_{\substack{\text{Determined by the} \\ \text{language or platform} \\ \text{you are using.}}} \times \underbrace{\text{CPI} \times \text{Clock Cycle Time}}_{\text{Determined by CPU hardware.}}$$

→ How many Risc-V instructions we need to complete the task/code?

* Two types of Risc-V implementations -

✓ (i) A simplified version (Slow, too much time wasted)

✓ (ii) A more realistic pipelined version (Optimized)

* We will only look into subset of instructions divided into 3 categories -

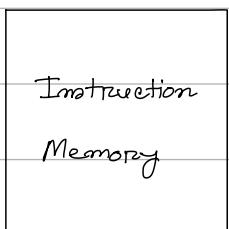
(i) Memory Reference : ld, sd → (main memory associated)
 (l) (s)

✓ (ii) Arithmetic / Logical : add, sub, and, orz (P)

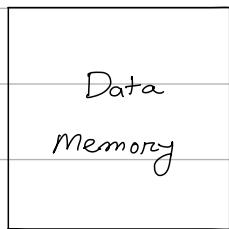
(iii) Control Transfer : beq ✓

 → (Branching)

Instruction Execution :



(holds the instructions only)



(Refers to the main-memory)

e.g. while using load/store instruction,

this memory is accessed.

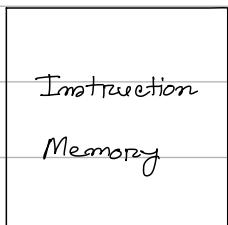
Instruction Execution

- PC → instruction memory, fetch instruction
- Register numbers → register file, read registers
- Depending on instruction class
 - Use ALU to calculate
 - Arithmetic result
 - Memory address for load/store
 - Branch comparison
 - Access data memory for load/store
 - $PC \leftarrow \text{target address or } PC + 4$

#8000

PC contains the address of the instruction in the program to be executed.

Now, go to that address in the



and fetch that instruction

#8000 → add x₂₀, x₂₁, x₀

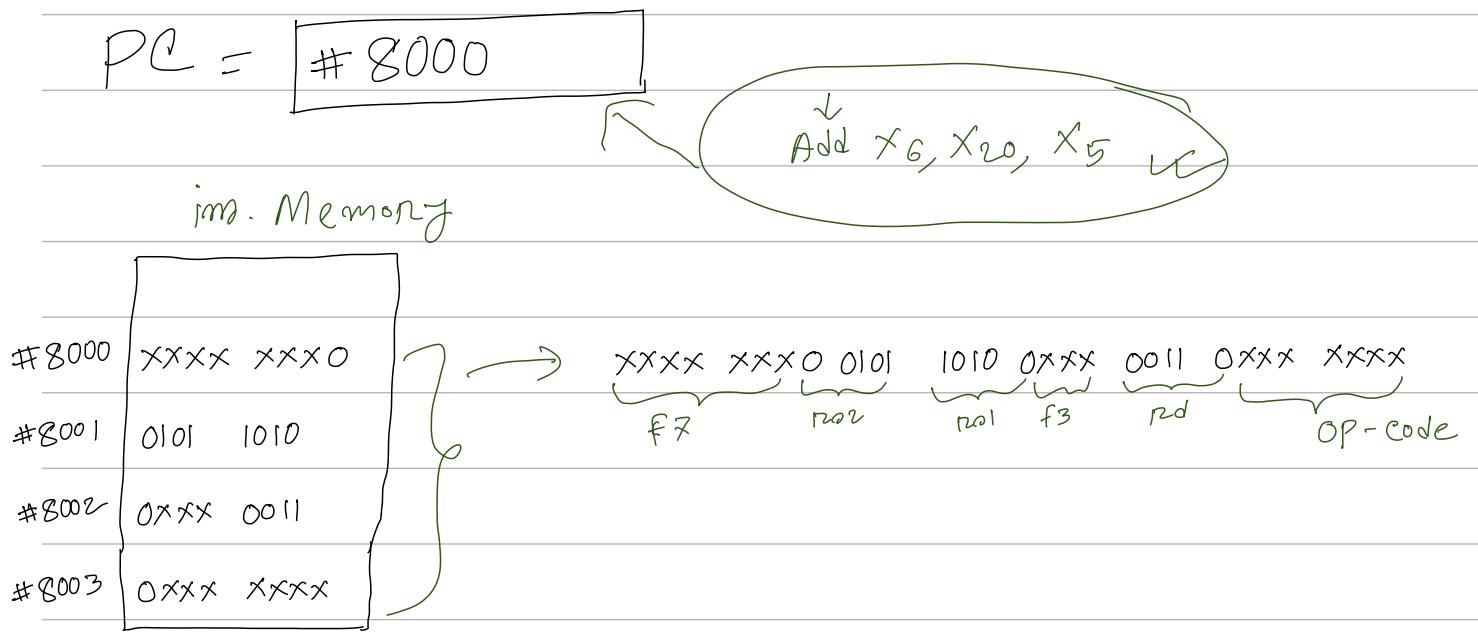
the fetched instruction will now be decoded.

↳ the registers associated with that decoded instruction

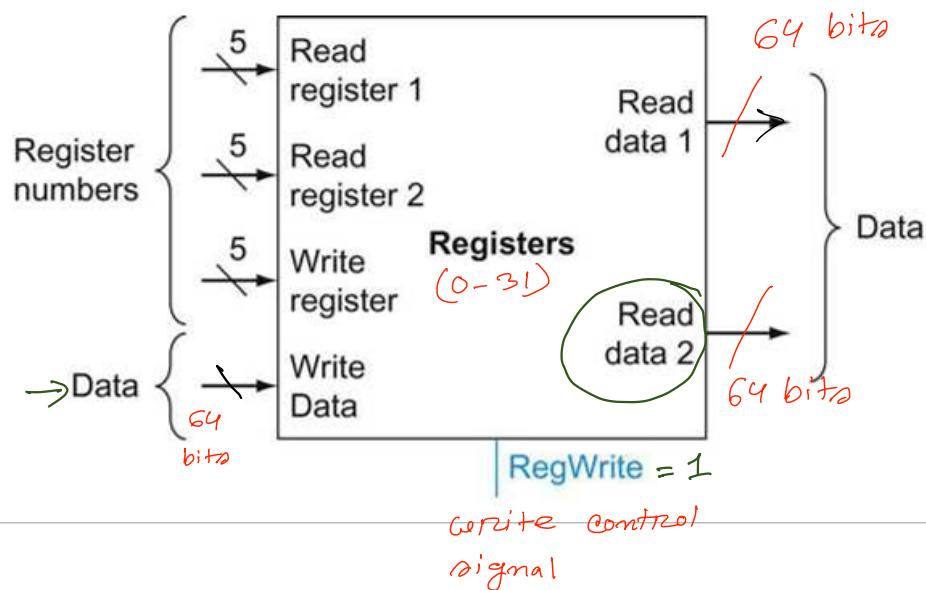
will be activated to read data.

Sample Simulation of the ins. exe.

Steps



Register File



(i) Read \Rightarrow Provide the register number that you want to read through the **Read register bus**.

\Rightarrow Data output will come from **Read data bus**.

(ii) Write \Rightarrow Provide the register number that you want to write through the **Write register bus**.

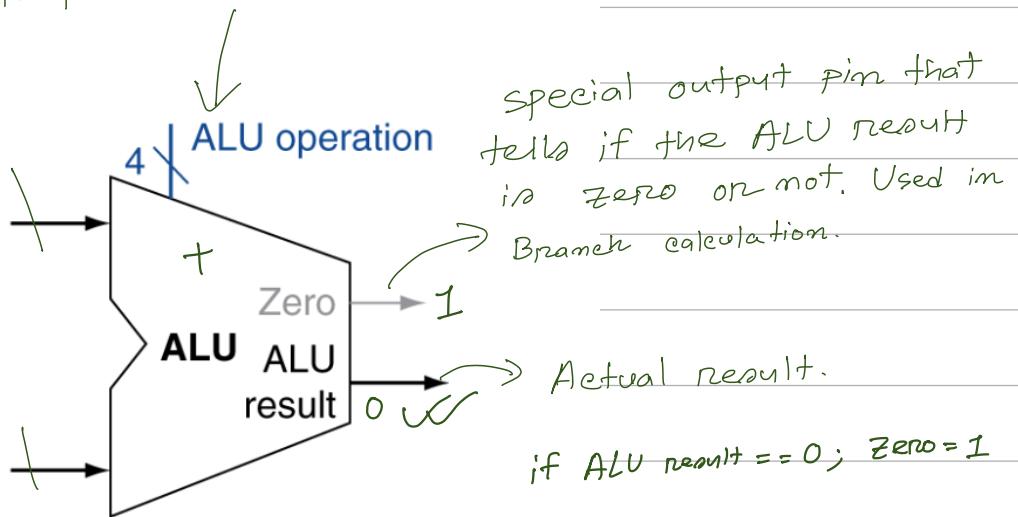
\Rightarrow Provide the data to be written via the **Write data bus**.

\Rightarrow **Write control signal** must be asserted.

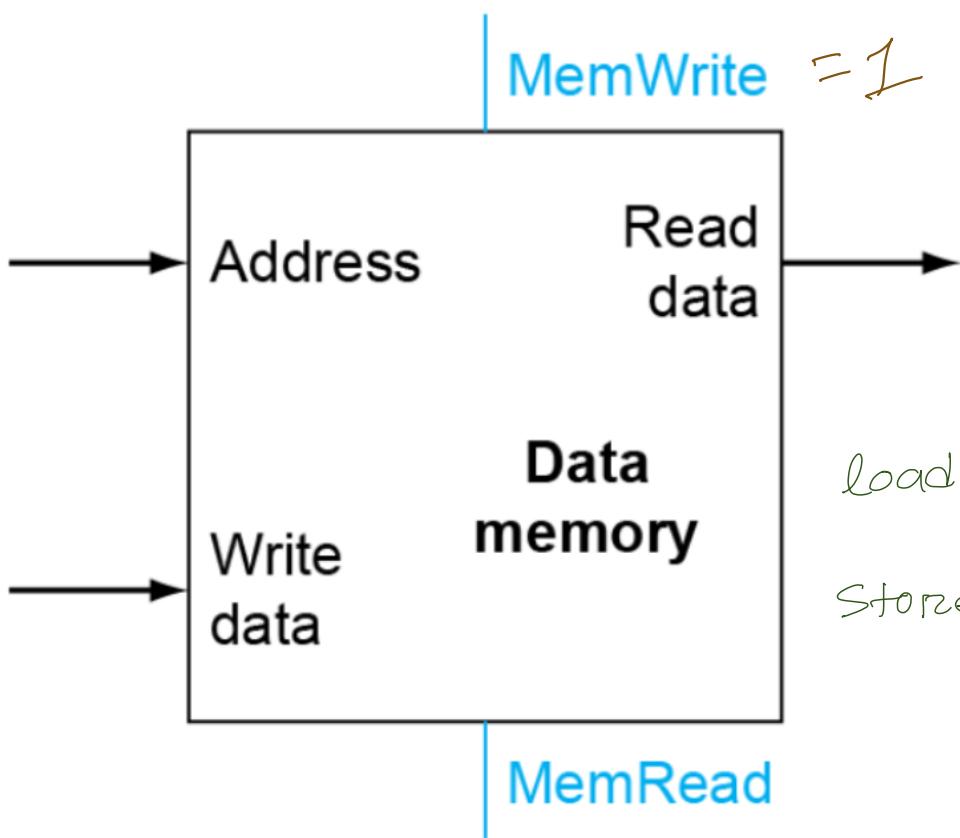
ALU (Arithmetic Logic Unit)

tells ALU which operation to perform on the inputs.

size of both inputs must be same



Data Memory

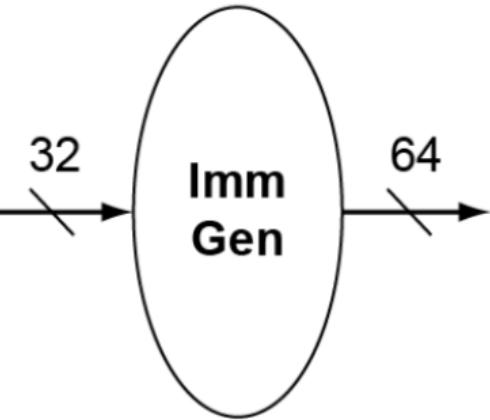


load \Rightarrow read data

Store \Rightarrow write data

Immediate Generation unit:

↳ Expands 12 bit immediate to 64 bit value.



⇒ $(d \times 2^1), 10 (x_{22})$

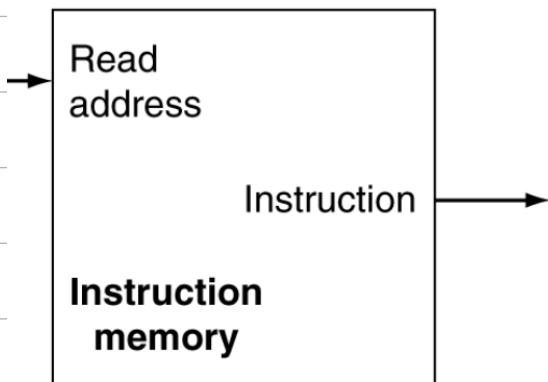
Size of immediate in I type instruction
is 12 bits.

⇒ Size of the complete instruction is 32 bits.

⇒ which is entered
as input in the
immediate generation
unit.

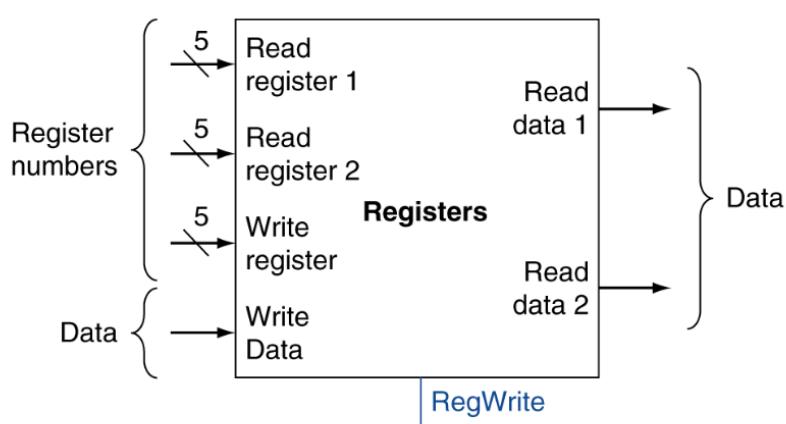
⇒ This unit then
extracts the 12 bit
immediate and expands
it into 64 bit.

Instruction Memory

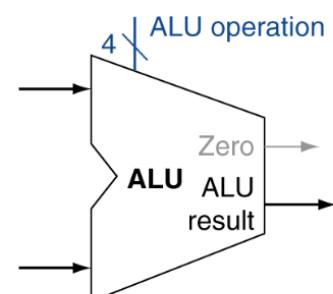


R-Format Instructions

- Read two register operands
- Perform arithmetic/logical operation
- Write register result



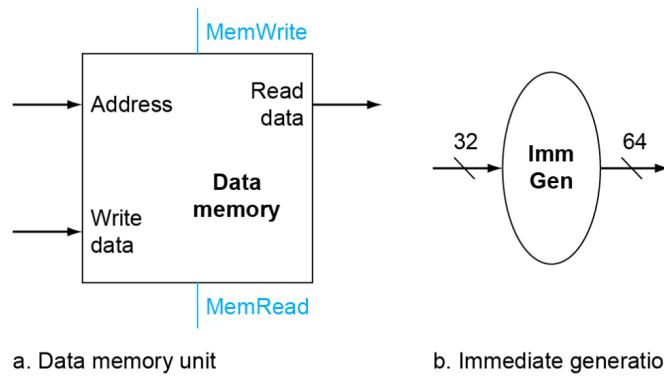
a. Registers



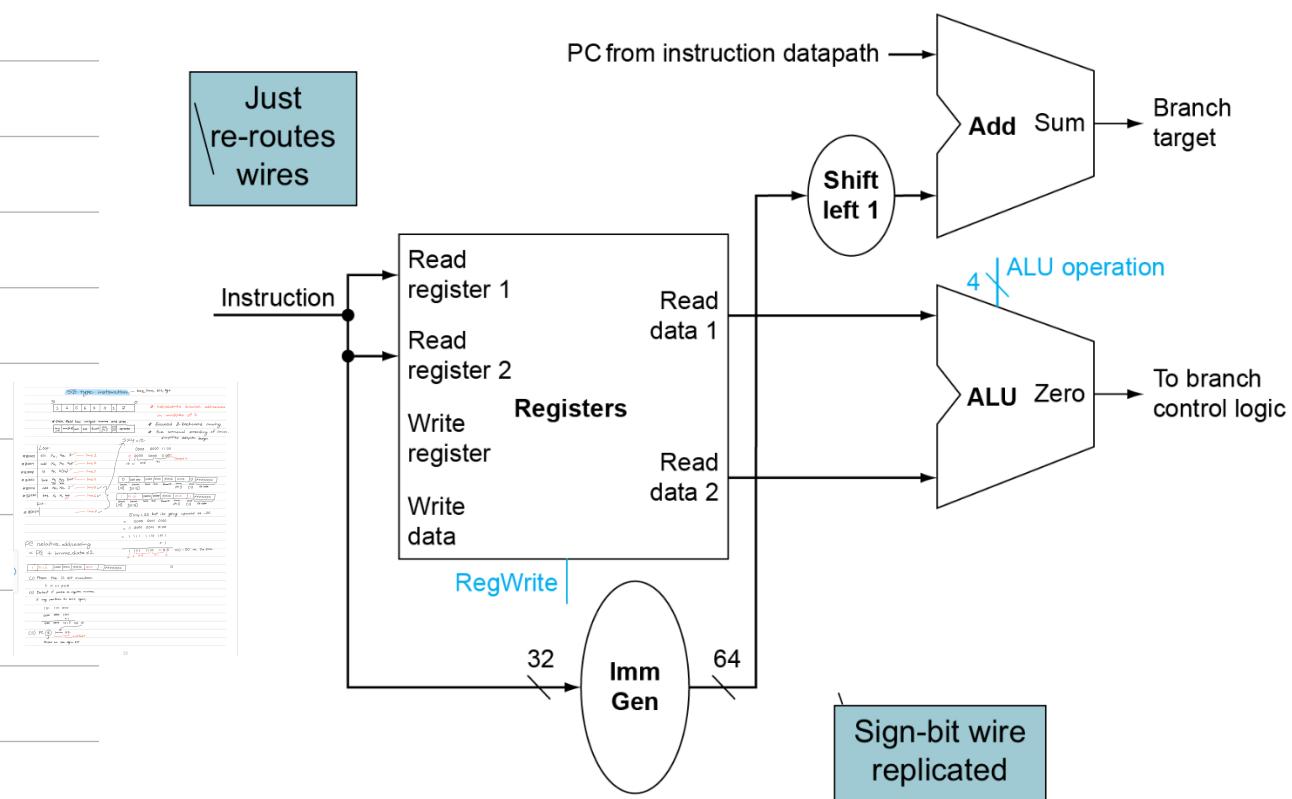
b. ALU

Load/Store Instructions

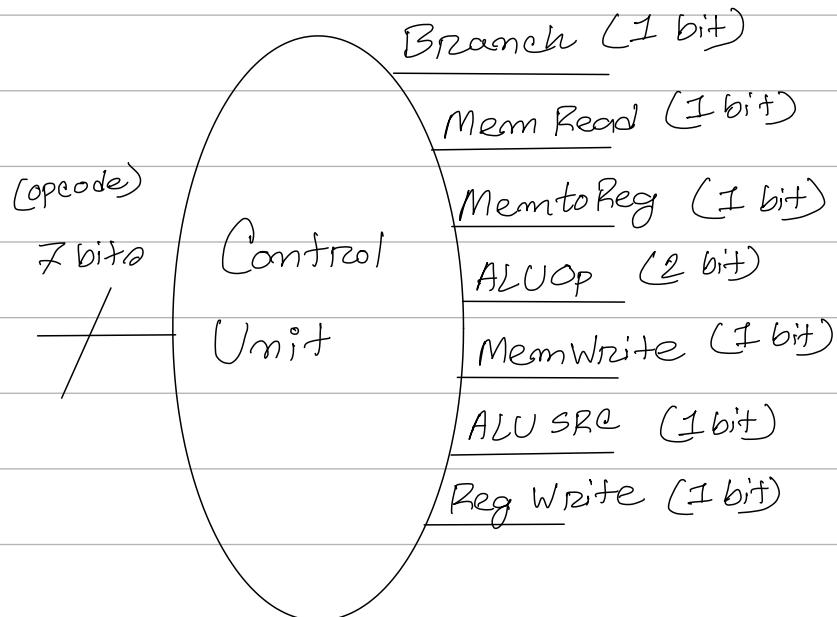
- Read register operands
- Calculate address using 12-bit offset
 - Use ALU, but sign-extend offset
- Load: Read memory and update register
- Store: Write register value to memory



Branch Instructions



Control Unit



ALU OP \Rightarrow 00 \rightarrow load / Store / Addi

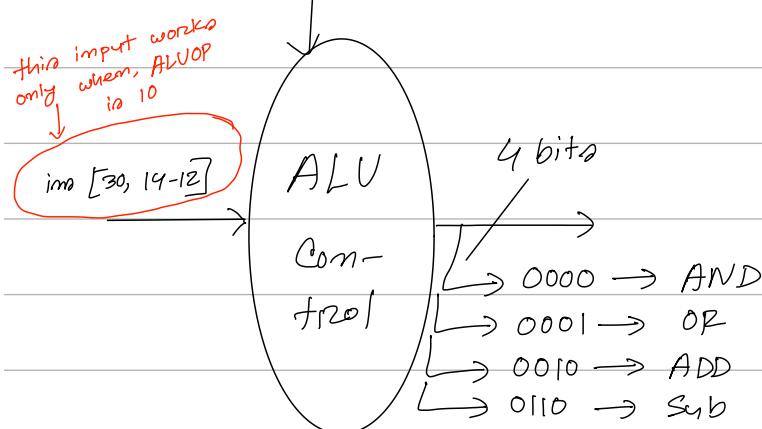
01 \rightarrow Beq

10 \rightarrow R type

ALU OP

ALU Control

it controls ALU's operations.



opcode	ALUOp	Operation	ALU function	ALU control
ld	00	load register	add	0010
sd	00	store register	add	0010
beq	01	branch on equal	subtract	0110
R-type	10	add	add	0010
		subtract	subtract	0110
		AND	AND	0000
		OR	OR	0001

Performance Issues :

⇒ Longest delay determines clock period.

↳ We execute one instruction per clock cycle.

⇒ Typically Load instruction takes the longest time to execute

$$\left. \begin{array}{l} \text{Sub} = 8 \text{ ps} \\ \text{Mul} = 10 \text{ ps} \\ \text{lw} = 15 \text{ ps} \\ \text{sw} = 13 \text{ ps} \end{array} \right\} \begin{array}{l} \text{Clock period} = 15 \text{ ps} \\ \therefore \text{Total time to execute this sequence} \\ = 15 + 15 + 15 + 15 = 60 \text{ ps} \end{array}$$

∴

$$\therefore \text{Actual time to execute this sequence} \\ = 8 + 10 + 15 + 13 = 46 \text{ ps}$$

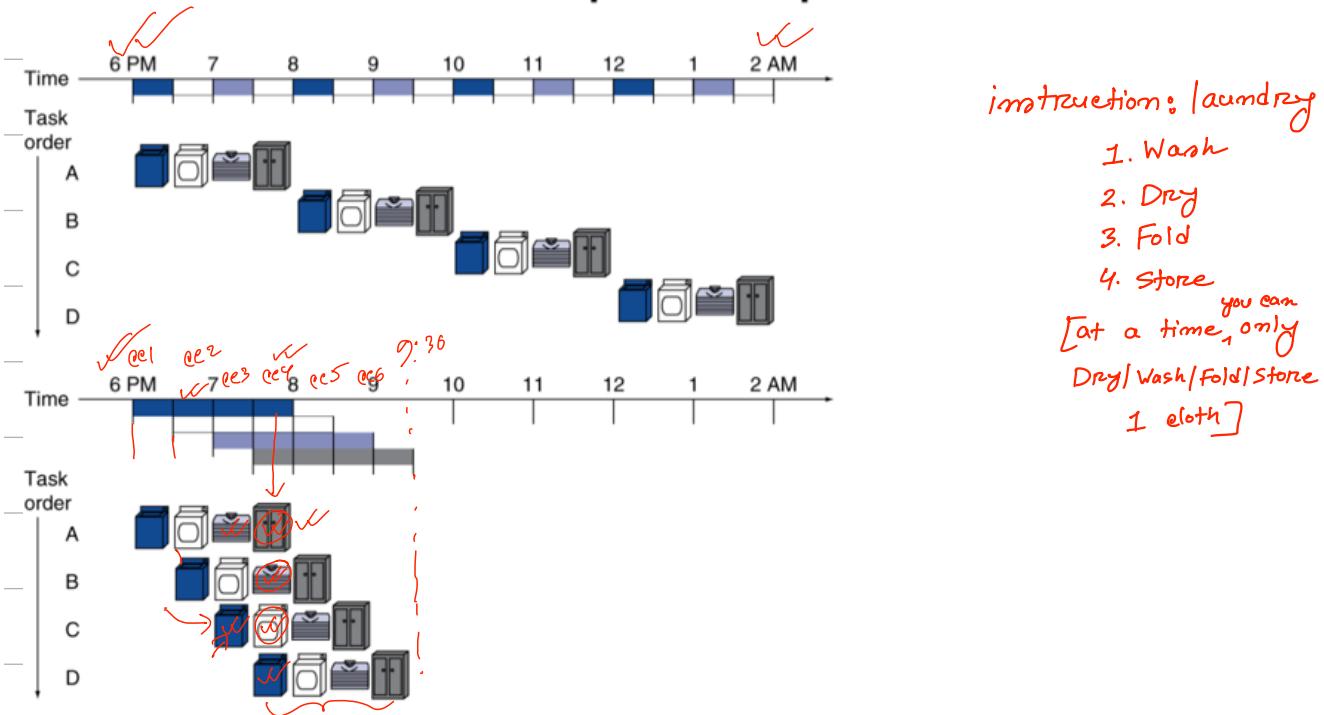
$$\text{Time Wastage} = (60 - 46) \text{ ps} = 14 \text{ ps}$$

to overcome this wastage we move on to
pipelining.

(parallel execution of instruction)

Pipelining Analogy

- Pipelined laundry: overlapping execution
 - Parallelism improves performance



RISC-V Pipeline

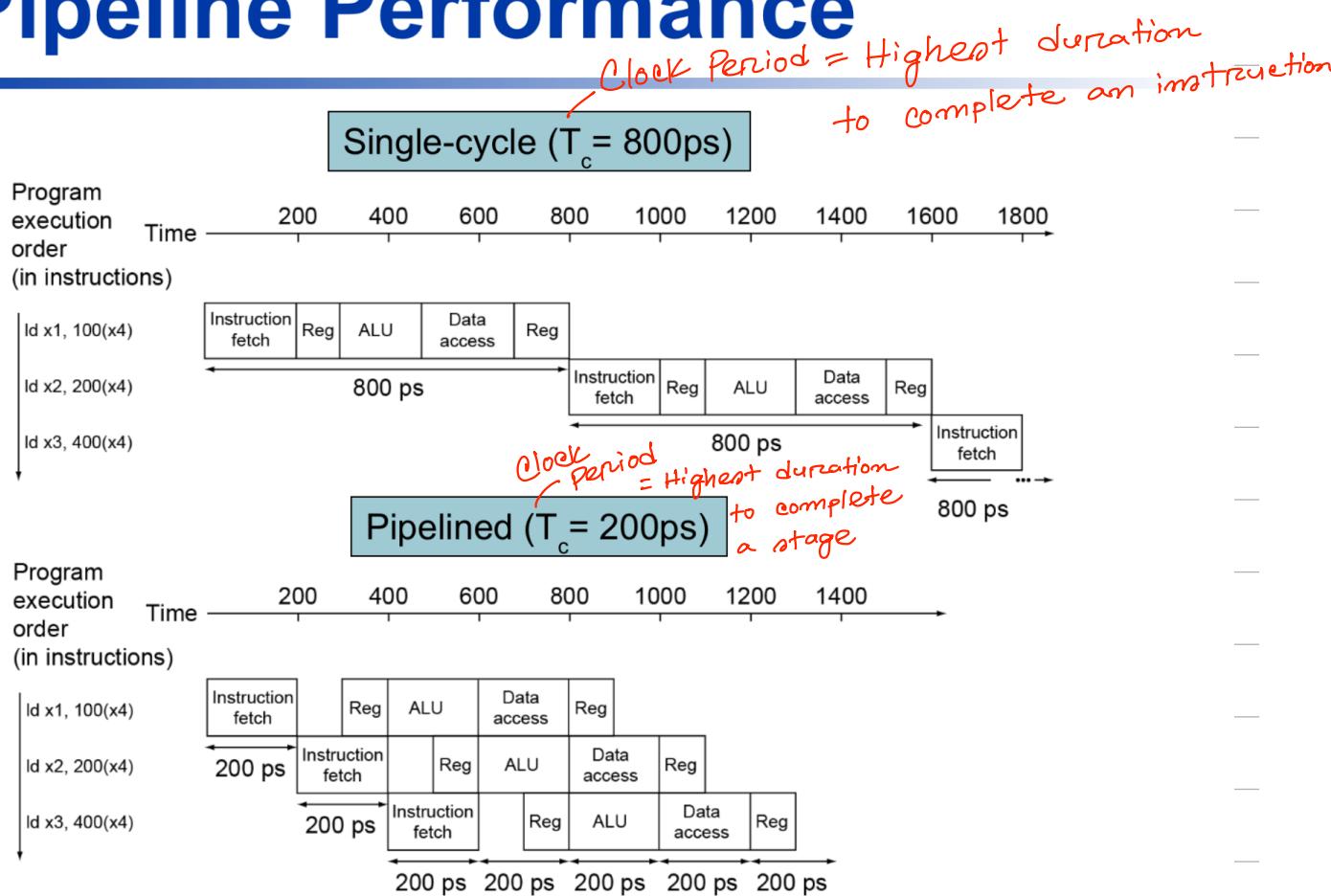
- Five stages, one step per stage
 1. IF: Instruction fetch from memory
 2. ID: Instruction decode & register read
 3. EX: Execute operation or calculate address
 4. MEM: Access memory operand
 5. WB: Write result back to register

Pipeline Performance

- Assume time for stages is
 - 100ps for register read or write
 - 200ps for other stages
- Compare pipelined datapath with single-cycle datapath

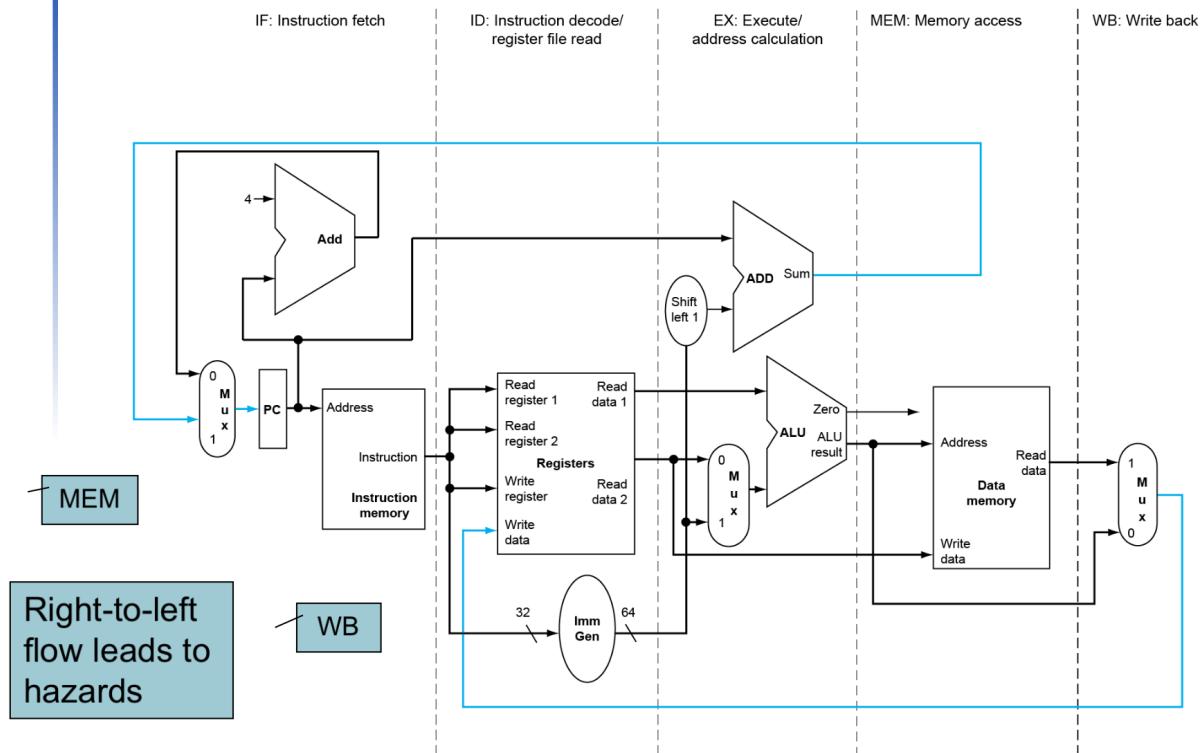
Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
ld	200ps	100 ps	200ps	200ps	100 ps	800ps
sd	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

Pipeline Performance



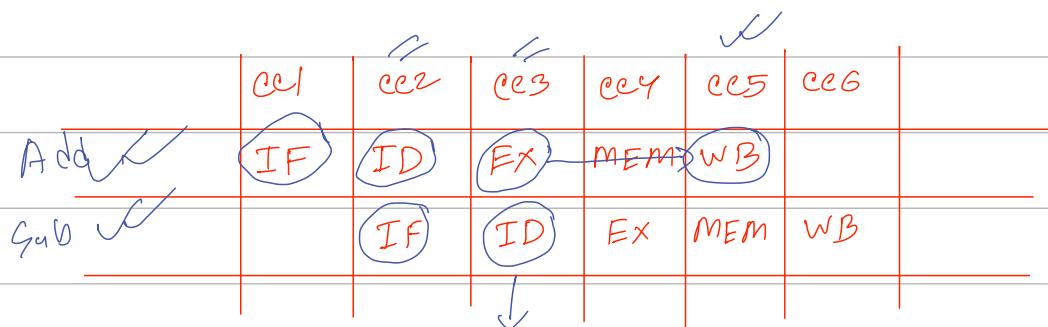
Pipelined Datapath

RISC-V Pipelined Datapath



Data Hazard

Add x_{21}, x_{22}, x_{23} ✓
 Sub x_{24}, x_{21}, x_{20} ↗



x_{21}, x_{20}

In Mem] Merged
Data Mem] together

Structural Hazard

✓	✓	✓	✓	✓						
ee1	ee2	ee3	ee4	ee5	ee6	ee7	ee8	ee9	ee10	
IF	ID	EX	MEM	WB						
IF	ID	EX	MEM	WB						
FF	ID	EX	MEM	WB						
IF	ID	EX	MEM	WB						
IF	ID	EX	MEM	WB						