Chapter-4

(How instructions are being executed inside processor?)

CPU time = Instruction count × CPI × Clock Cycle Time

Instruction count: Determined by the language or platform you are using. → ISA, Compiler
→ How many Risc-V instructions we need to complete the task/code?

CPI × Clock Cycle Time: Determined by CPU hardware.

* Two types of Risc-V implementations –

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

(ii) A more realistic pipelined version (Optimized)

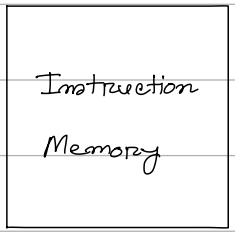
* We will only look into subset of instructions divided into 3 categories —

(i) Memory Reference: ld, sd (I) (S) → (main memory associated)

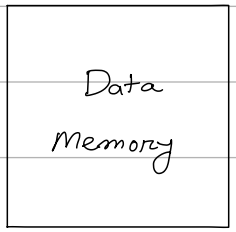
(ii) Arithmetic / Logical: add, sub, and, or (R)

(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.

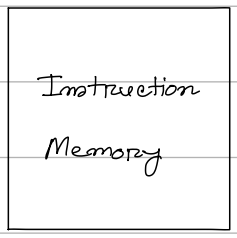
Instruction Execution

- 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 ← target address or PC + 4

#8000

PC contains the address of the instruction in the program being executed.

Now, go to that address in the



and fetch that instruction

#8000 → add X_{20}, X_{21}, X_0

the fetched instruction will now be decoded.

↳ the registers associated with that decoded instruction will be activated to read data.

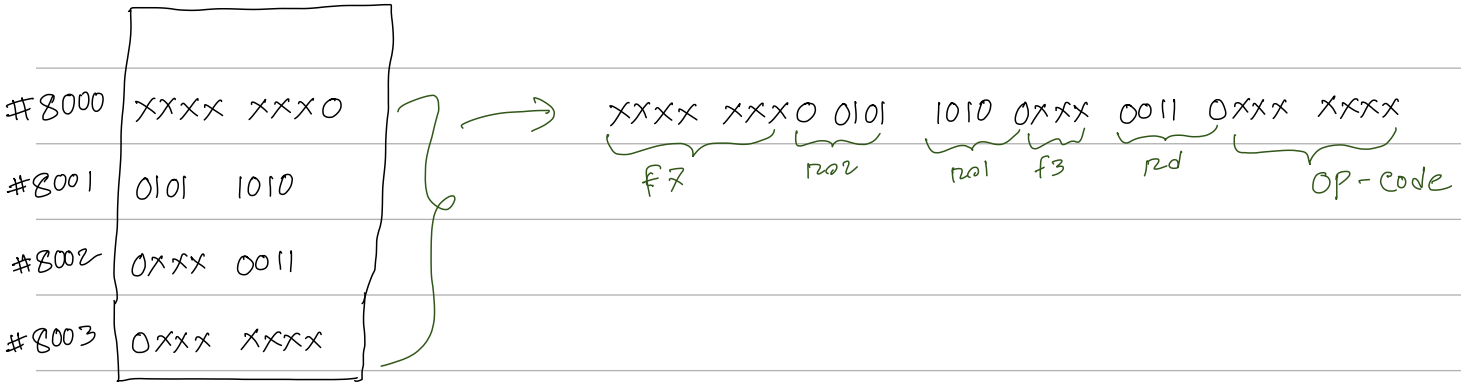
Sample Simulation of the ins. exe.

Steps

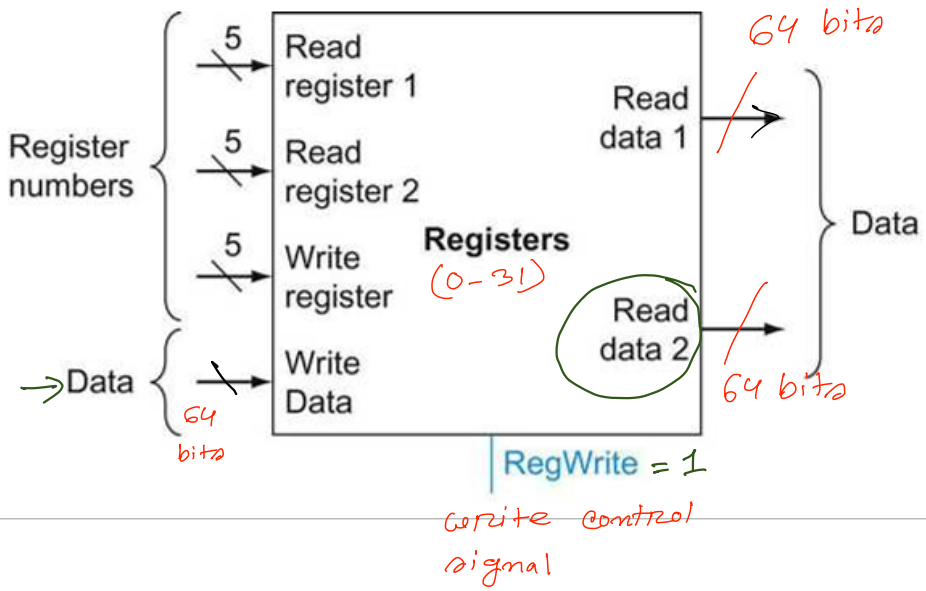
PC = #8000

Add x6, x20, x5

ins. Memory



Register File



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

⇒ Data output will come from **Read data** bus.

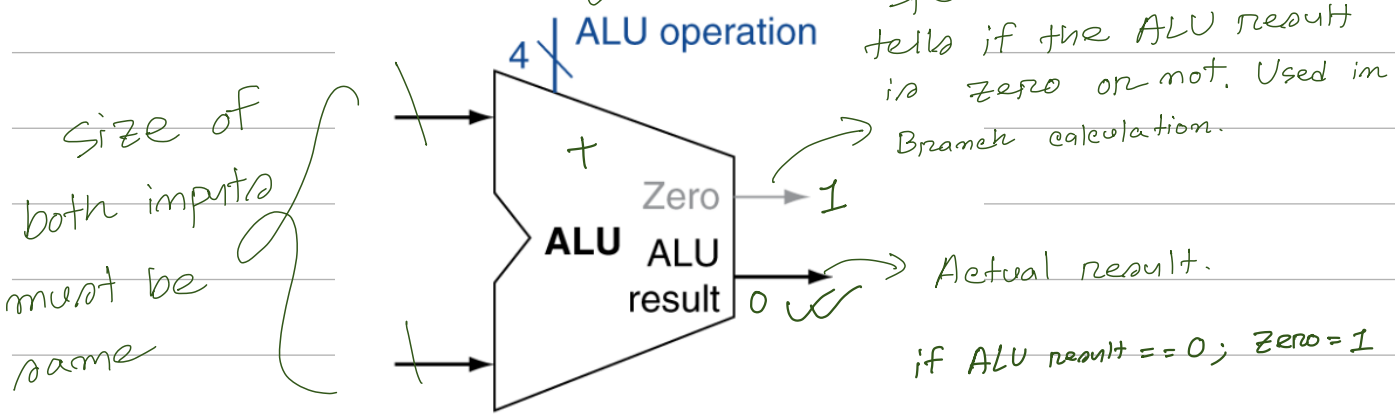
(ii) Write ⇒ Provide the register number that you want to write through the **Write register** bus.

⇒ Provide the data to be written via the **Write data** bus.

⇒ **Write control signal** must be asserted.

ALU (Arithmetic Logic Unit)

tells ALU which operation to perform on the inputs.



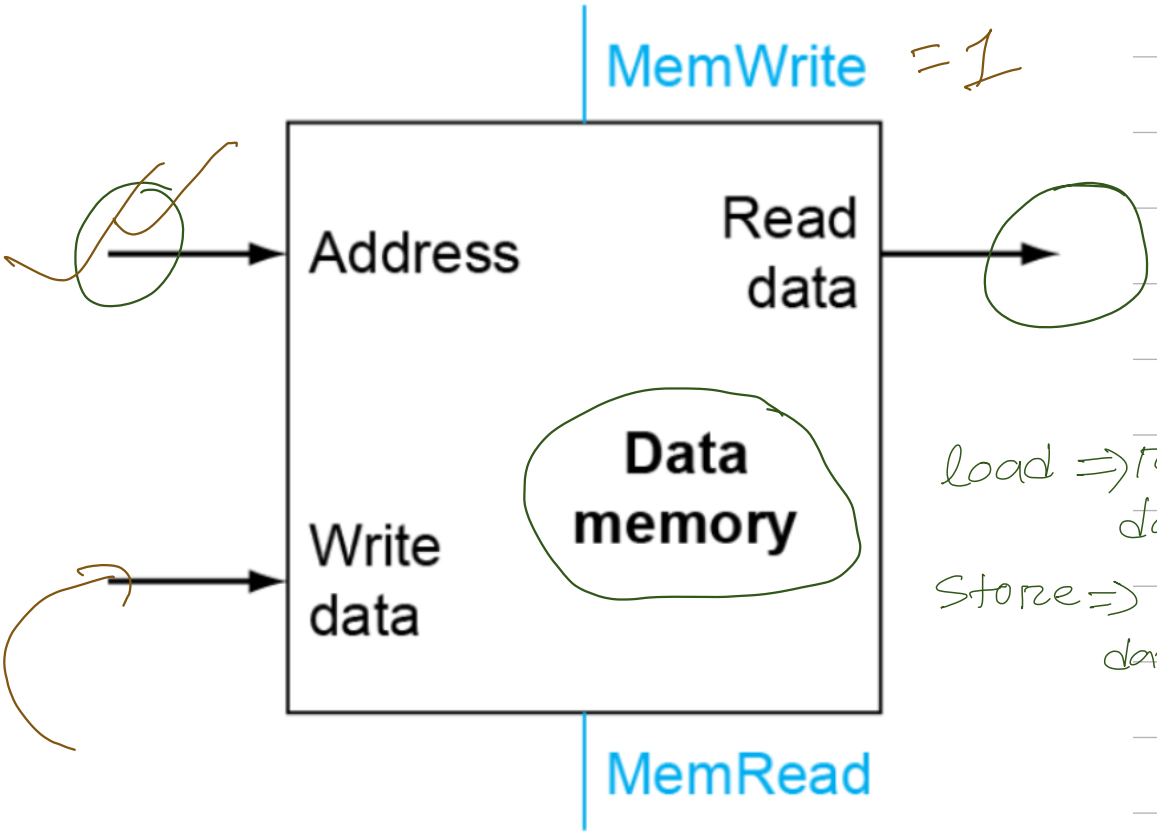
size of both inputs must be same

special output pin that tells if the ALU result is zero or not. Used in Branch calculation.

Actual result.

if ALU result == 0; zero = 1

Data Memory

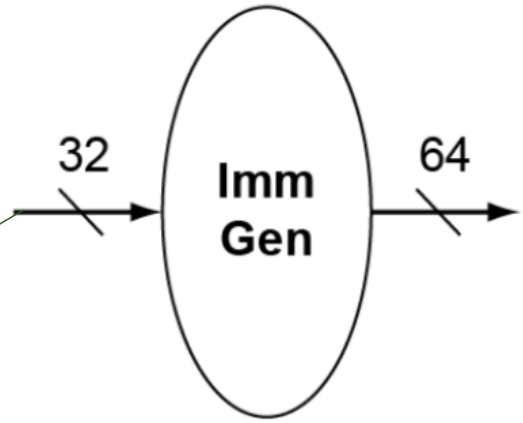


load ⇒ read data

Store ⇒ write data

Immediate Generation unit:

↳ Expands 12 bit immediate to 64 bit value.



⇒ ld X21, 10 (X22)

Size of immediate in I type instruction is 12 bits.

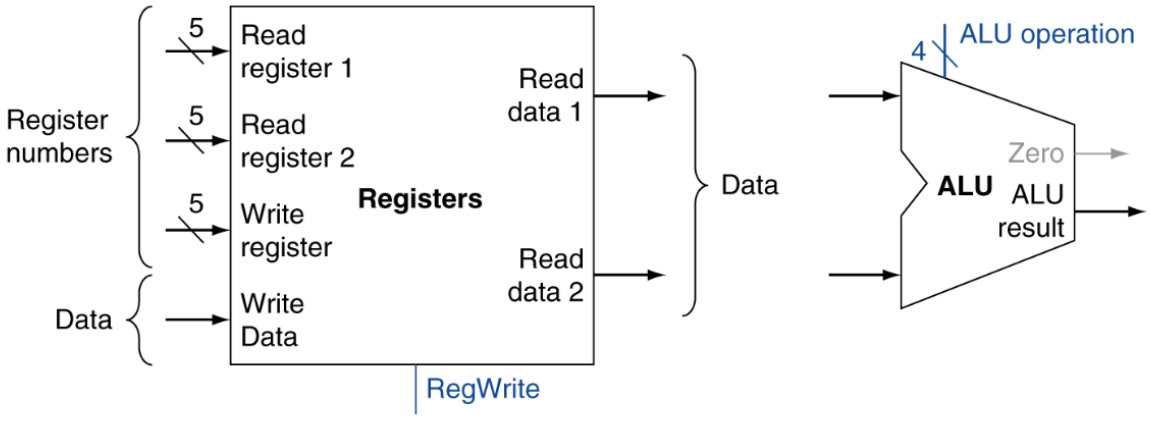
⇒ Size of the complete instruction is 32 bits.

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

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

R-Format Instructions

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



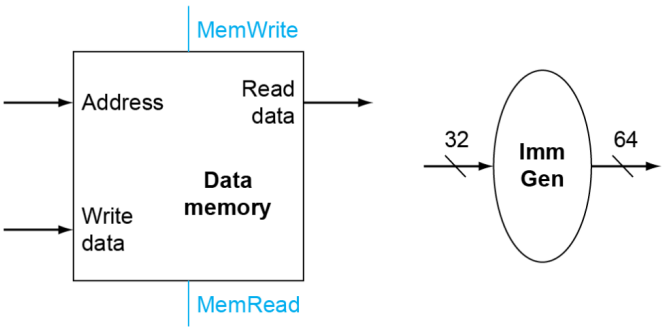
a. Registers

b. ALU

ld x22, 16(x21)

Load/Store Instructions

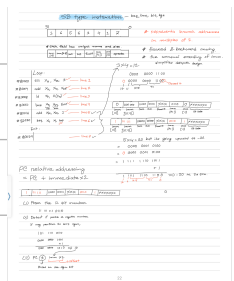
- Read register operands
- Calculate address using 12-bit offset
  - Use ALU, but sign-extend offset
- Load: Read memory and update register
- Store: Write register value to memory

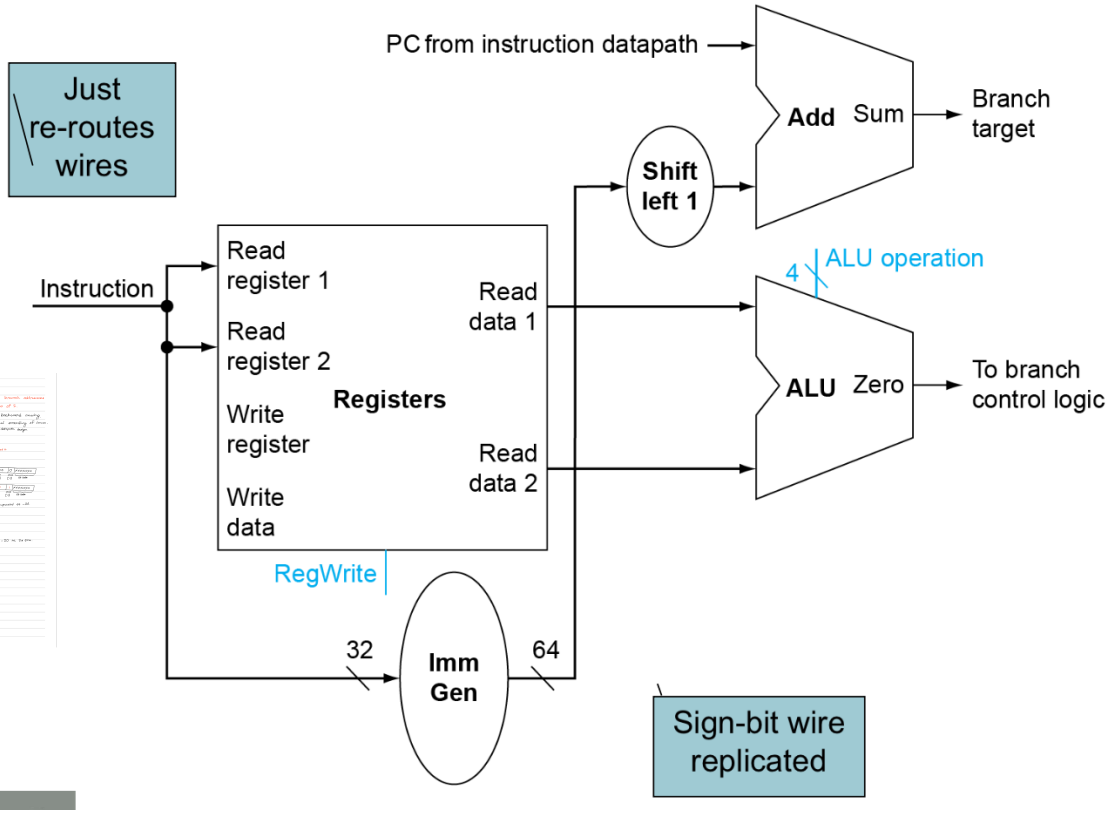


a. Data memory unit

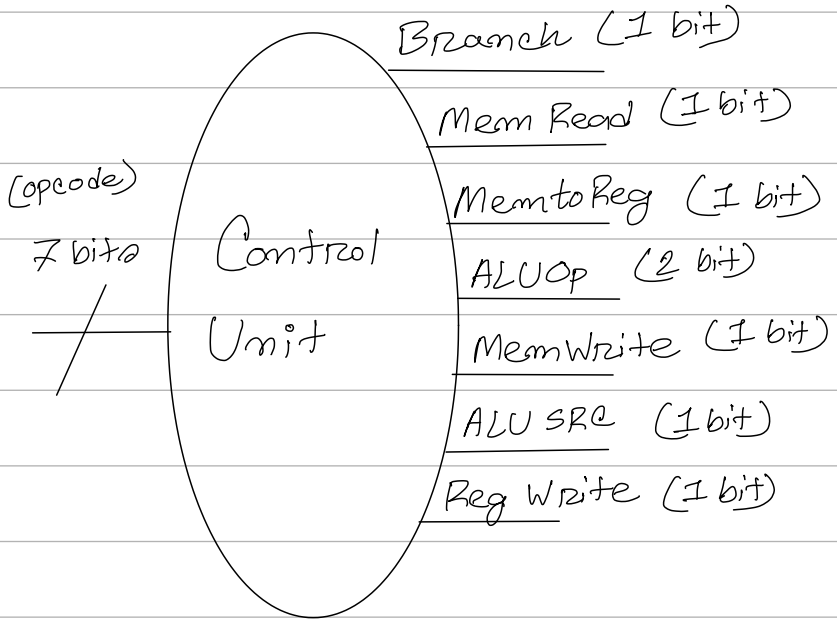
b. Immediate generation unit

Branch Instructions





Control Unit



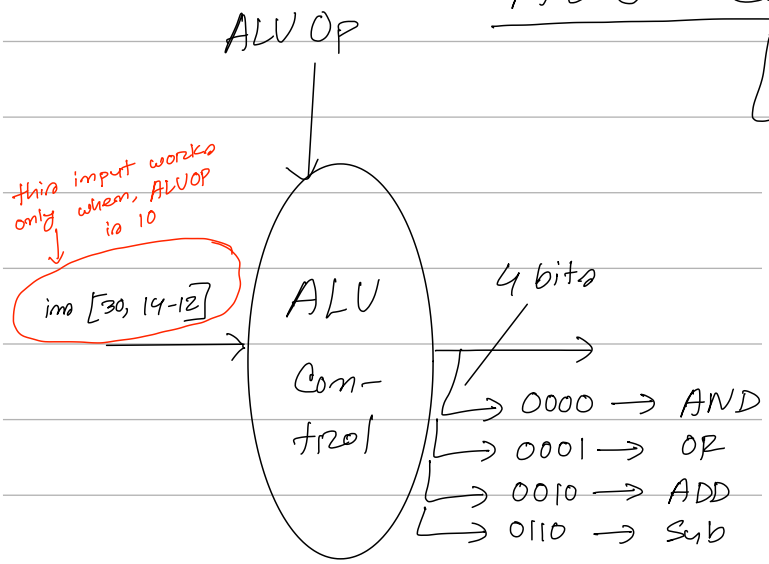
ALU Op => 00 → load / Store / Addi

01 → Beq

10 → R type

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

# Sub = 8 ps
Mul = 10 ps
lw = 15 ps
sw = 13 ps

Clock period = 15 ps

∴ Total time to execute this sequence
= 15+15+15+15 = 60 ps

Vs

∴ Actual time to execute this sequence
= 8+10+15+13 = 46 ps

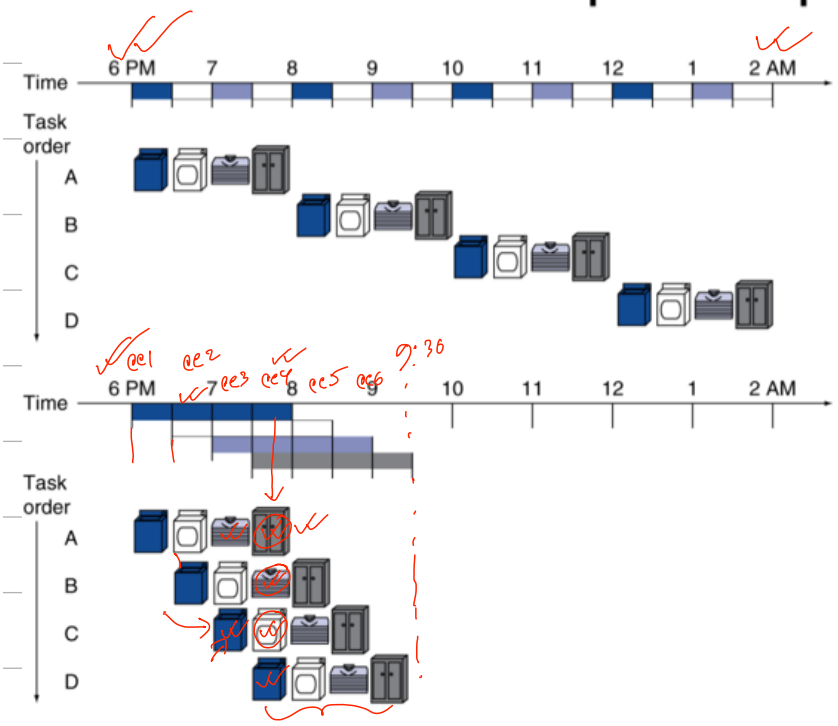
Time Wastage = (60 - 46) ps = 14 ps

to overcome this wastage we move on to pipelining.

(parallel execution of instruction)

Pipelining Analogy

- Pipelined laundry: overlapping execution
  - Parallelism improves performance



instruction: laundry
1. Wash
2. Dry
3. Fold
4. Store
[at a time, you can only Dry/Wash/Fold/Store 1 cloth]

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

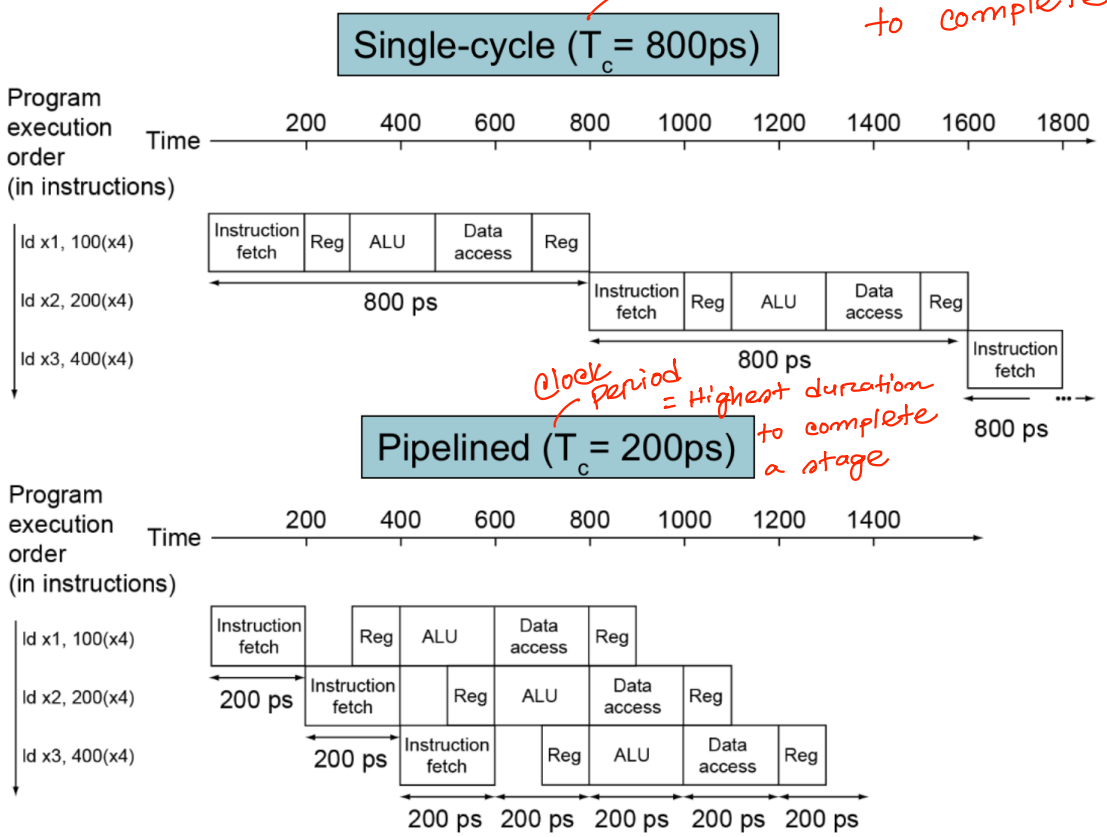
Pipeline Performance

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

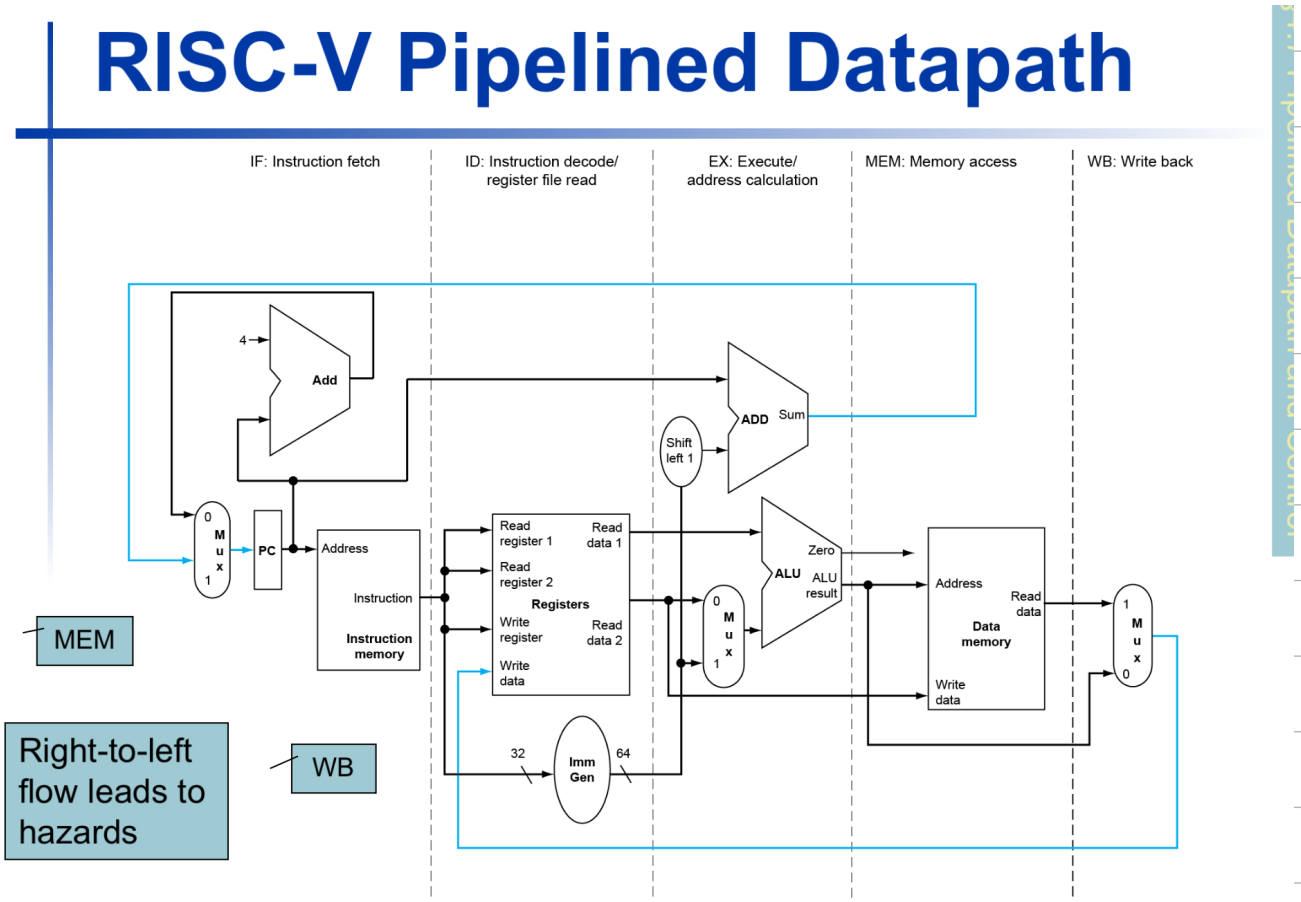
Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
ld	200ps	100 ps	200ps	200ps	100 ps	800ps
sd	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

Pipeline Performance

Clock Period = Highest duration to complete an instruction



Pipelined Datapath



Data Hazard

Add x21, x22, x23

Sub x24, x21, x20

	cc1	cc2	cc3	cc4	cc5	cc6
Add	IF	ID	EX	MEM	WB	
Sub		IF	ID	EX	MEM	WB

x21, x20

Structural Hazard