$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

"X is n time faster than Y"

$$\text{Performance}_X / \text{Performance}_Y$$

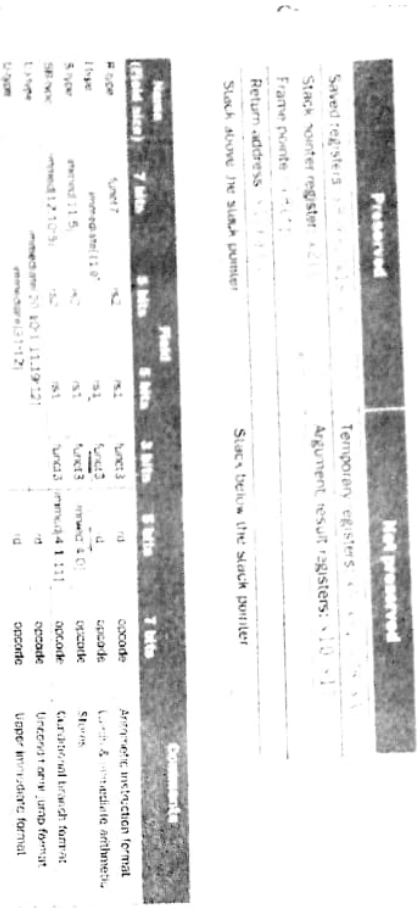
$$= \text{Execution time}_Y / \text{Execution time}_X = n$$

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

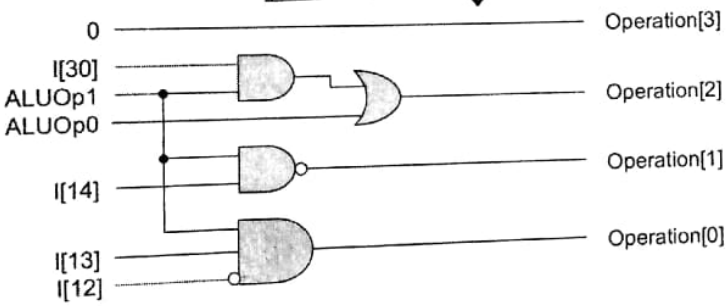
$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \underbrace{\frac{\text{Instruction Count}_i}{\text{Instruction Count}}}_{\text{Relative frequency}} \right)$$

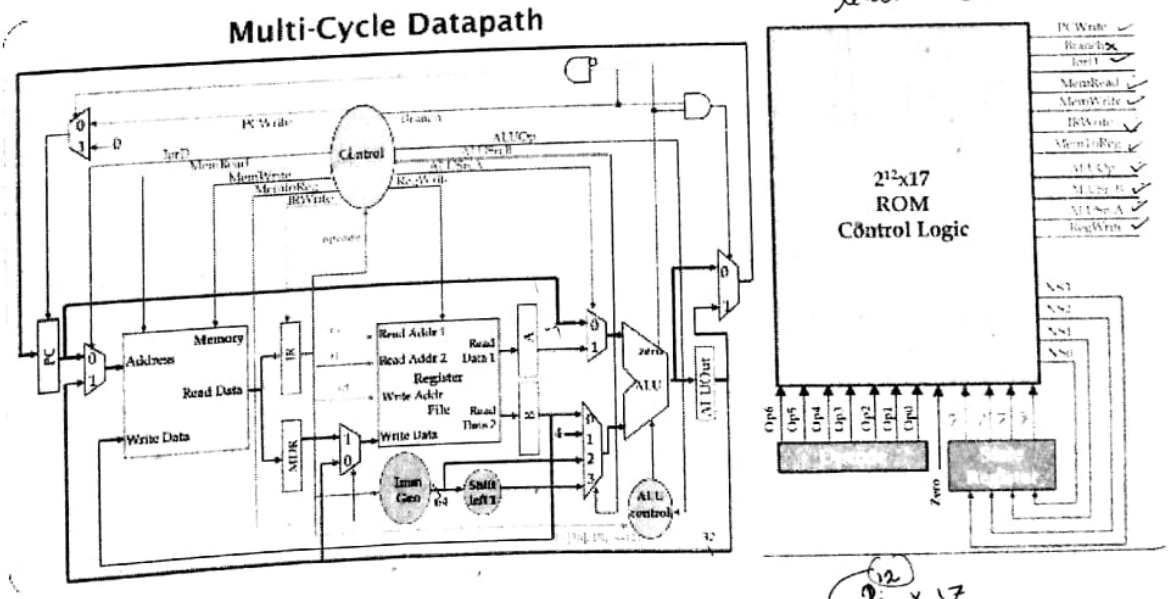
Instruction Class	Clock Cycles Required
Load	5
Store	4
Branch	3 or 4
Arithmetic-logical	4



ALU ctrl → for sc, Mc



Multi-cycle ctrl



a = A[2] → load

A[3] = b → store

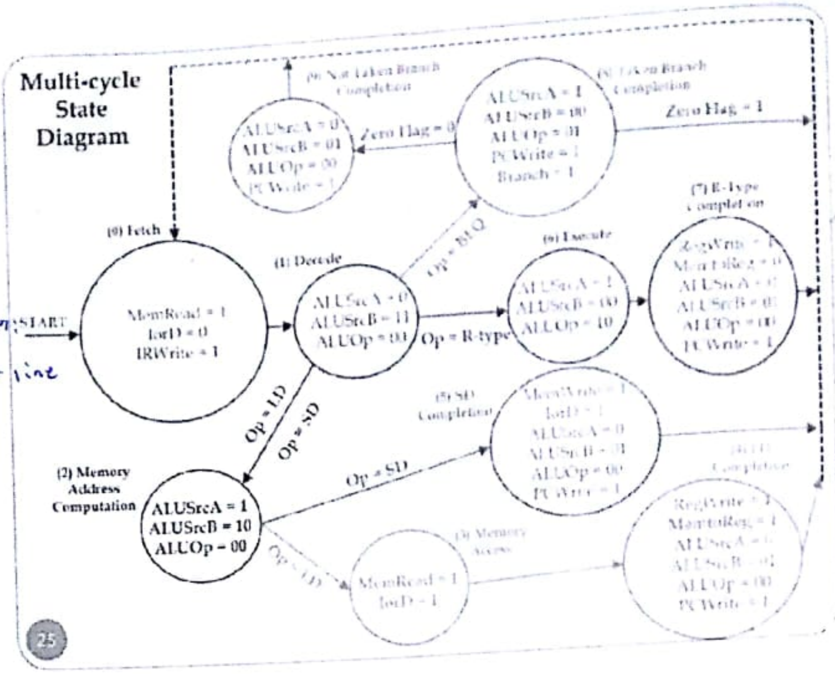
$2^{12} \times 17$

ROM Need Address

Address: 12 bit

= 7 bit opcode + 4 bit current state + 1 bit zero flag.

Signals not mentioned in state diagram:
1) x → Mux select line + ALUOP
2) 0 → other ctrl



NSL Truth Table

Current State ($S_3S_2S_1S_0$)	Input (Opcode + Zero Flag)	Next State ($NS_3NS_2NS_1NS_0$)
0000 (State0)	Op[6:0] = xxxxxxx	0001
0001 (State1)	Op[6:0] = 0000011 (LD)	0010
0001 (State1)	Op[6:0] = 0100011 (SD)	0010
0001 (State1)	Op[6:0] = 0110011 (R-format)	0110
0001 (State1)	Op[6:0] = 1100011 (BEQ)	1000
0010 (State2)	Op[6:0] = 0000011 (LD)	0011
0010 (State2)	Op[6:0] = 0100011 (SD)	0101
0011 (State3)	Op[6:0] = xxxxxxx	0100
0100 (State4)	Op[6:0] = xxxxxxx	0000
0101 (State5)	Op[6:0] = xxxxxxx	0000
0110 (State6)	Op[6:0] = xxxxxxx	0111
0111 (State7)	Op[6:0] = xxxxxxx	0000
1000 (State8)	(Op[6:0] = xxxxxxx) && (Zero Flag = 1)	0000
1000 (State8)	(Op[6:0] = xxxxxxx) && (Zero Flag = 0)	1001
1001 (State9)	Op[6:0] = xxxxxxx	0000

- $NS_0 = State0 + State2\,((Op = LD) + (Op = SD)) + State6 + State8\,(Zero\ Flag = 0)$
- $NS_0 = \bar{S_3}\bar{S_2}\bar{S_1}\bar{S_0} + \bar{S_3}\bar{S_2}S_1\bar{S_0}\,(\overline{Op_6}\,\overline{Op_5}\,\overline{Op_4}\,\overline{Op_3}\,\overline{Op_2}Op_1Op_0 + \overline{Op_6}Op_5\overline{Op_4}\,\overline{Op_3}\,\overline{Op_2}Op_1Op_0) + \bar{S_3}S_2S_1\bar{S_0} + S_3\bar{S_2}\bar{S_1}\bar{S_0}\bar{Z}$
- Using the same approach write the equations for NS_0, NS_1, and NS_2.

27

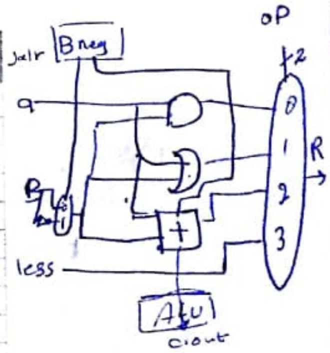
OFL Truth Table

Current State ($S_3S_2S_1S_0$)	PCWrite	Branch	IorD	MemRead	MemWrite	IRWrite	MemToReg	ALUOp<1>	ALUOp<0>	ALUSrcB<1>	ALUSrcB<0>	ALUSrcA	RegWrite
0000 (State0)	0	0	0	1	0	1	X	X	X	X	X	X	0
0001 (State1)	0	0	X	0	0	0	X	0	0	1	1	0	0
0010 (State2)	0	0	X	0	0	0	X	0	0	1	0	1	0
0011 (State3)	0	0	1	1	0	0	X	X	X	X	X	X	0
0100 (State4)	1	0	X	0	0	0	1	0	0	0	1	0	1
0101 (State5)	1	0	1	0	1	0	X	0	0	0	1	0	0
0110 (State6)	0	0	X	0	0	0	X	1	0	0	0	1	0
0111 (State7)	1	0	X	0	0	0	0	0	0	0	1	0	1
1000 (State8)	1	1	X	0	0	0	X	0	1	0	0	1	0
1001 (State9)	1	0	X	0	0	0	X	0	0	0	1	0	0

- $PCWrite = State_4 + State_5 + State_7 + State_8 + State_9 = \bar{S_3}S_2\bar{S_1}\bar{S_0} + \bar{S_3}S_2\bar{S_1}S_0 + \bar{S_3}S_2S_1S_0 + S_3\bar{S_2}\bar{S_1}\bar{S_0} + S_3\bar{S_2}\bar{S_1}S_0$
- $IorD = State_3 + State_5 = \bar{S_3}\bar{S_2}S_1S_0 + \bar{S_3}S_2\bar{S_1}S_0$
- Using the same approach write the equations for the remaining control signals.

28

Input or output	Signal name	R-format	ld	sd	beq
Inputs	I[6]	0	0	0	1
	I[5]	1	0	1	1
	I[4]	1	0	0	0
	I[3]	0	0	0	0
	I[2]	0	0	0	0
	I[1]	1	1	1	1
	I[0]	1	1	1	1
Outputs	1 ALUSrc	0	1	1	0
	0 MemtoReg	0	1	X	X
	1 RegWrite	1	1	0	0
	0 MemRead	0	1	0	0
	0 MemWrite	0	0	1	0
	0 Branch	0	0	0	1
	0 ALUOp1	1	0	0	0
	0 ALUOp0	0	0	0	1
	jalr	0	0		



→ for SC, MC ALU control

func	Ainvert	Bnegate	operation 2bit
and	0	0	00
or	0	0	01
add	0	0	10
sub	0	1	10

- ALUOp[1:0], 00 - Addition, 01 - Subtraction, 10 - R-type
- ALU Control lines, 0000 - AND, 0001 - OR, 0010 - Add, 0110 - Subtract
- PCSource = Branch & Zero
- Slowest instruction in SC-CPU is ld (load)
  - 2 Memory accesses
  - 2 Register accesses
  - 1 ALU Operation

- In each cycle, only one major unit is allowed to be used in multi-cycle CPU (ALU/Mem/Register File).
- The values for the signals that are not mentioned in a state are either
  - Don't Care for MUX select signals and ALUOp OR
  - Deasserted (0) for all other control signals

Multi-Cycle Control

- **Implementing the Next State Function with a Sequencer**

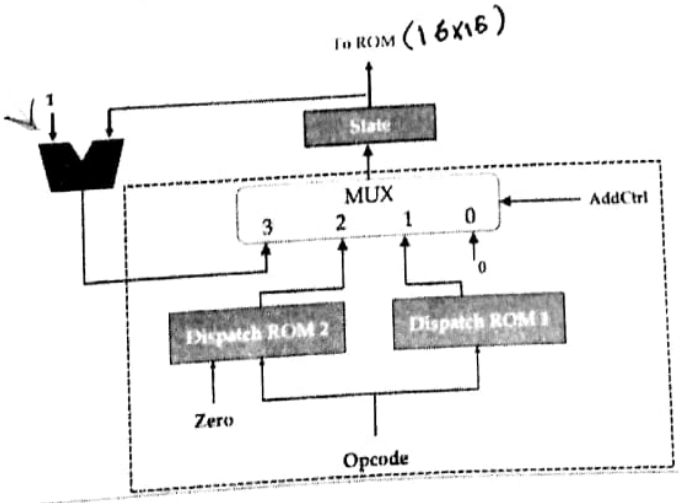
Dispatch ROM1		
Opcode Field	Opcode Name	Value
0110011	R-Format	0110
1100011	beq	1000
0000011	ld	0010
0100011	sd	0010

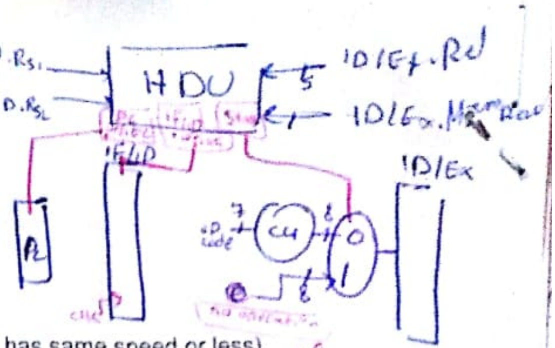
Dispatch ROM2			
Zero	Opcode Field	Opcode Name	Value
0	1100011	beq	1001
1	1100011	beq	0000
0	0000011	ld	0011
1	0000011	ld	0011
0	0100011	sd	0101
1	0100011	sd	0101

State number	Address-control action	Value of AddrCtl
0	Use incremented state	3
1	Use dispatch ROM 1	1
2	Use dispatch ROM 2	2
3	Use incremented state	3
4	Replace state number by 0	0
5	Replace state number by 0	0
6	Use incremented state	3
7	Replace state number by 0	0
8	Replace state number by 0	0
9	Replace state number by 0	0

AddrCtl value	Action
0	Set state to 0
1	Dispatch with ROM 1
2	Dispatch with ROM 2
3	Use the incremented state

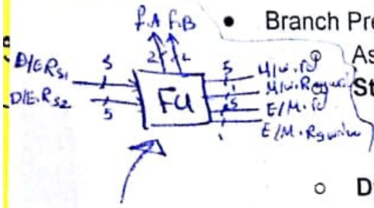
- Time between instructions$_{pipelined}$
  = Time between instructions$_{nonpipelined}$ / Number of stages





- 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 increases throughput not latency (single instruction has same speed or less).
- Each instruction has the same latency.
- Hazard Types:
  - Structural: A required resource is busy.
  - Data: Need to wait for previous instruction to complete its data read/write
  - Control: Deciding on control action depends on previous instruction.
- Branch Prediction:
  - Assume it's always taken or not (not realistic).
  - **Static branch prediction**, Based on typical branch behavior:
    - Predict backward branches taken.
    - Predict forward branches not taken.
  - **Dynamic branch prediction**, Hardware measures actual branch behavior.

if stall = 1
IF/ID write & PC write = 1



EX hazard M to E

- if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs1)) ForwardA = 10
- if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs2)) ForwardB = 10

MEM hazard W to E

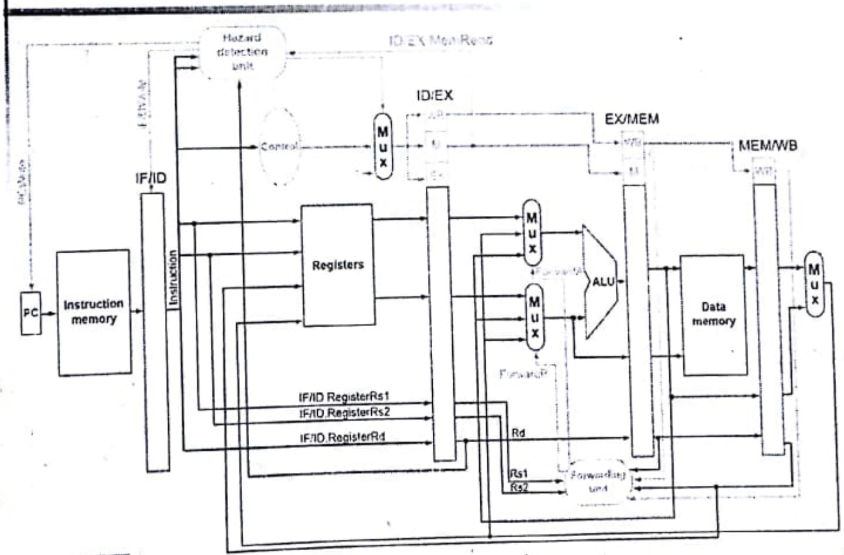
- if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and (MEM/WB.RegisterRd = ID/EX.RegisterRs1)) ForwardA = 01
- if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and (MEM/WB.RegisterRd = ID/EX.RegisterRs2)) ForwardB = 01

Load-use hazard when

- ID/EX.MemRead and (ID/EX.RegisterRd ≠ 0) and ((ID/EX.RegisterRd = IF/ID.RegisterRs1) or (ID/EX.RegisterRd = IF/ID.RegisterRs2))

ld x3 — F D E M W
add —, x3, — — F D D E M W
sub — F F D - - -

Datapath with Hazard Detection



in this Design so
3 Regwrite
2 Mem write
3 MemtoReg
2 Branch
1 [ALU src, ALU op

static via prediction
1) predict backward (do while) → Taken
2) predict forward (while) Not Taken

Data Hazard
R-type → M to E / W to E
ld inst → stall + W to E if we have dependency.

ctrl Hazard (Branch
① + comparator in Decode stage D
→ + 1 cycle stall
OR ② + (via prediction)
Assume Always (Taken / not Taken)
beq F D E M W
PC+4 F D D D D