EECS 322 Computer Architecture

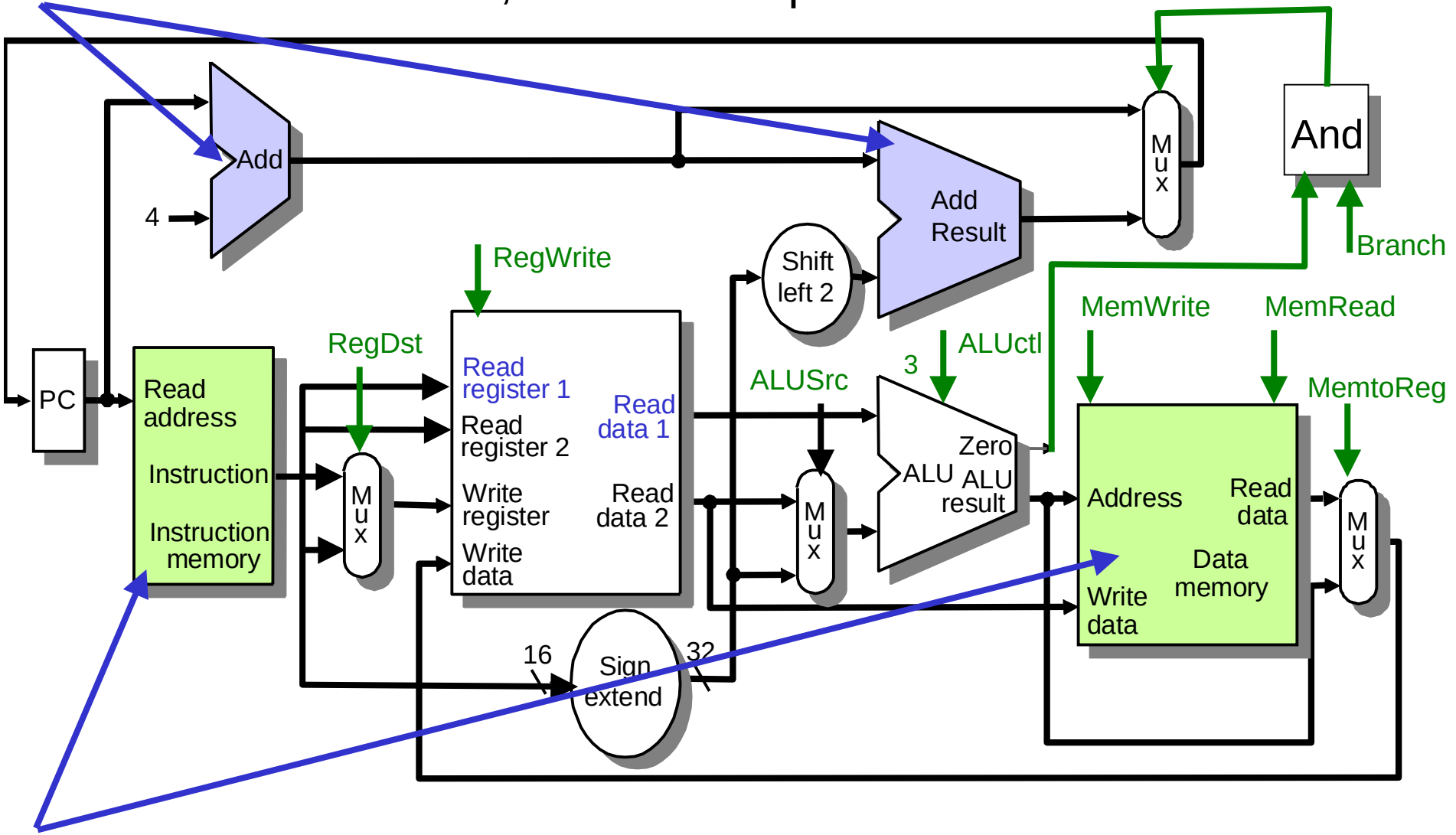
Finite State Machines

Instructor: Francis G. Wolff
wolff@eecs.cwru.edu
Case Western Reserve University

This presentation uses powerpoint animation: please viewshow

Review: Single-Cycle Datapath

2 adders: PC+4 adder, Branch/Jump offset adder

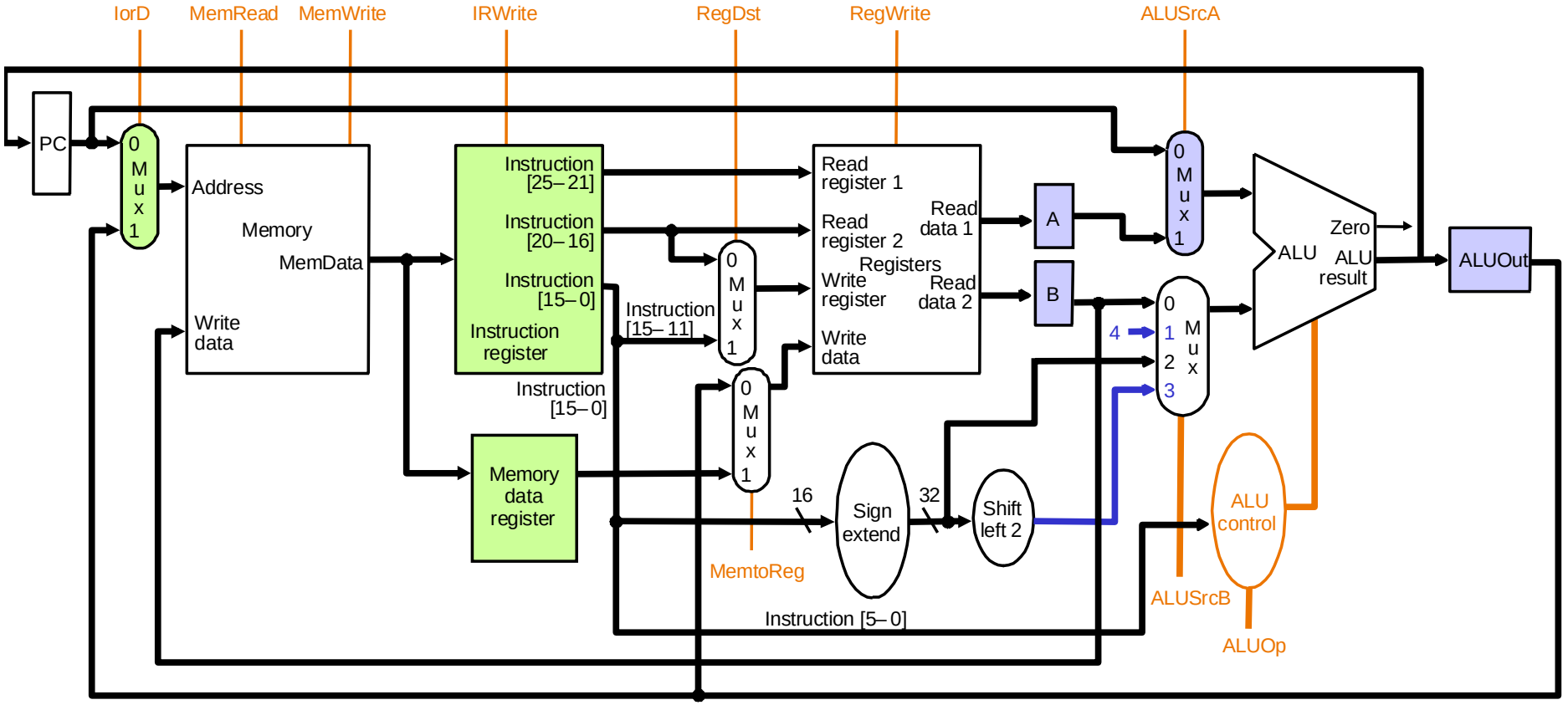


Harvard Architecture: Separate instruction and data memory

Multi-cycle Processor Datapath

Combine adders: add 1½ Mux & 3 temp. registers, A, B, ALUOut

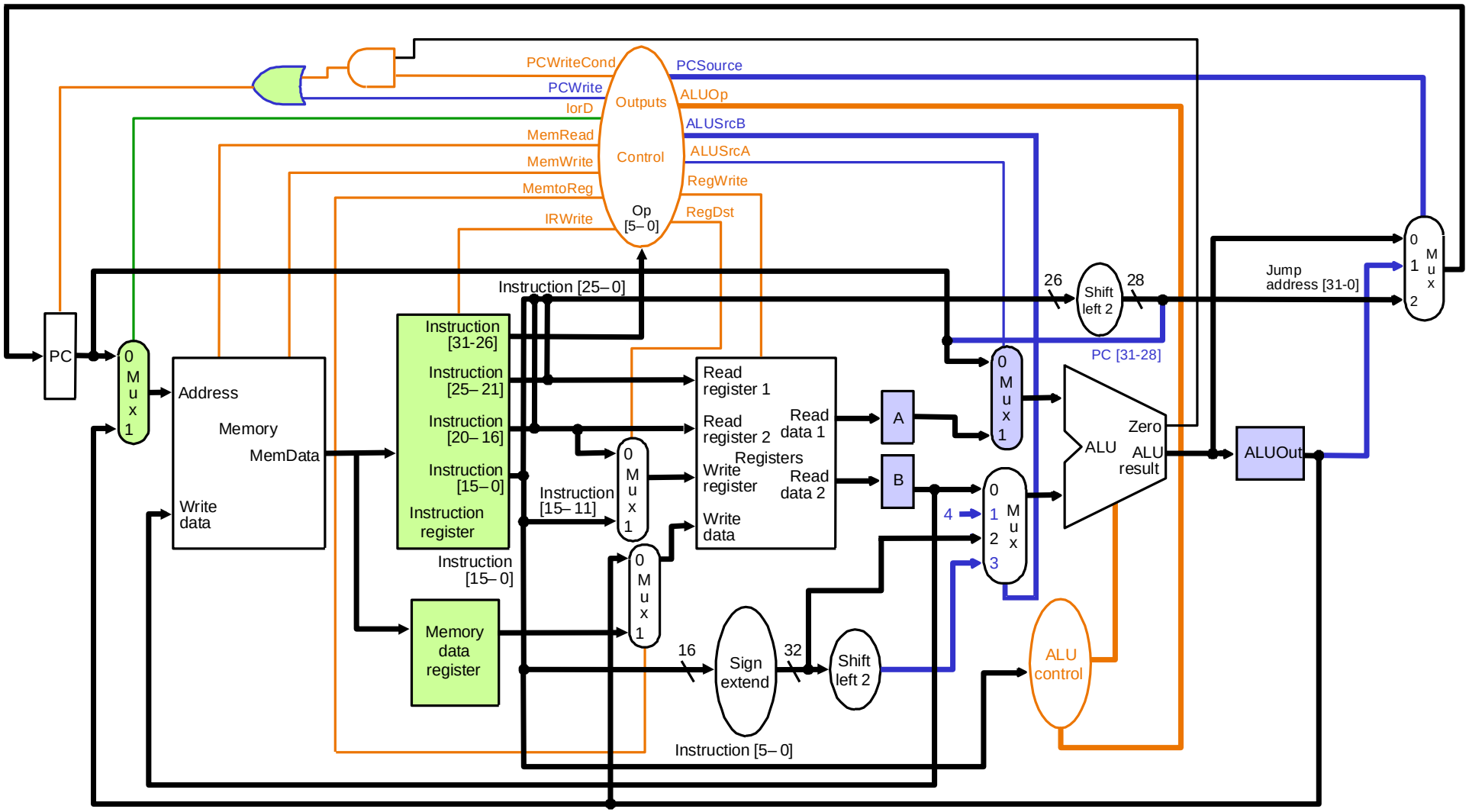
Combine Memory: add 1 Mux & 2 temp. registers, IR, MDR



Single-cycle= 1 ALU + 2 Mem + 4 Muxes + 2 adders + OpcodeDecoders

Multi-cycle = 1 ALU + 1 Mem + 5½ Muxes + 5 Reg (IR,A,B,MDR,ALUOut) + FSM

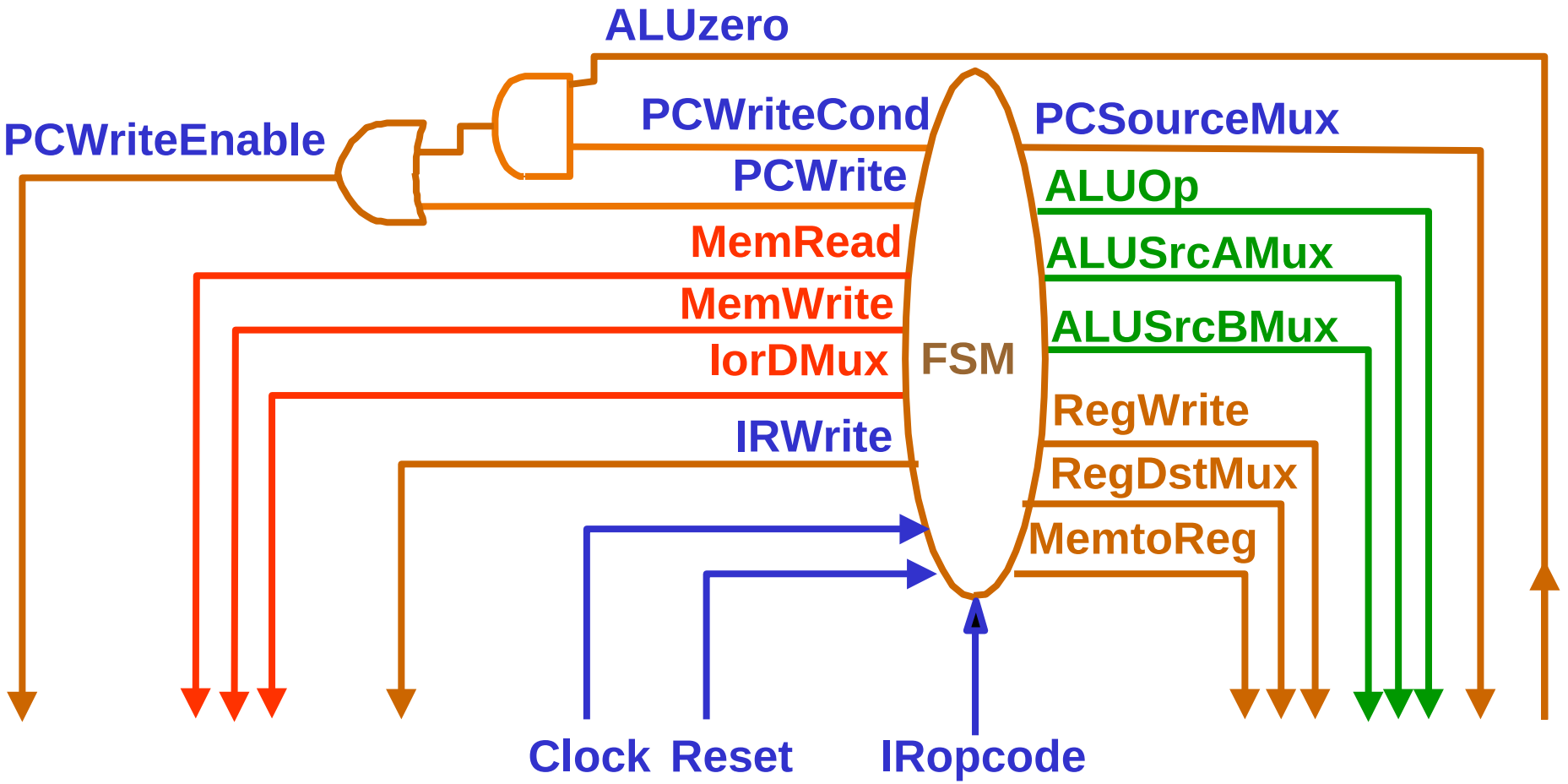
Multi-cycle Datapath: with FSM controller



New datapath controls: IorD, ALUSrcA, PCWrite

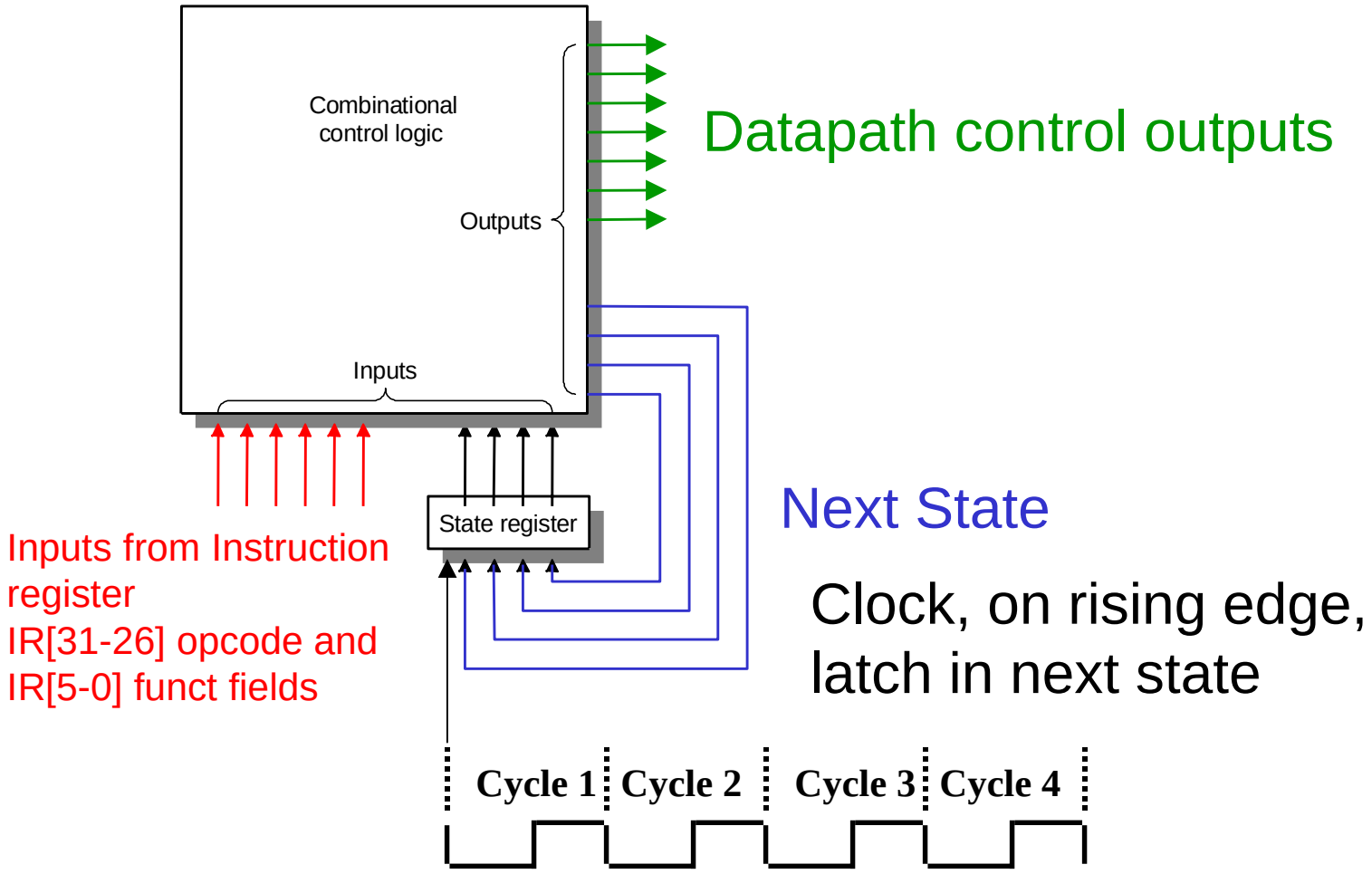
Changed: ALUSrcB(was ALUsrc), PCWriteCond(Branch), PCSource

CPU controller: Finite State Machine

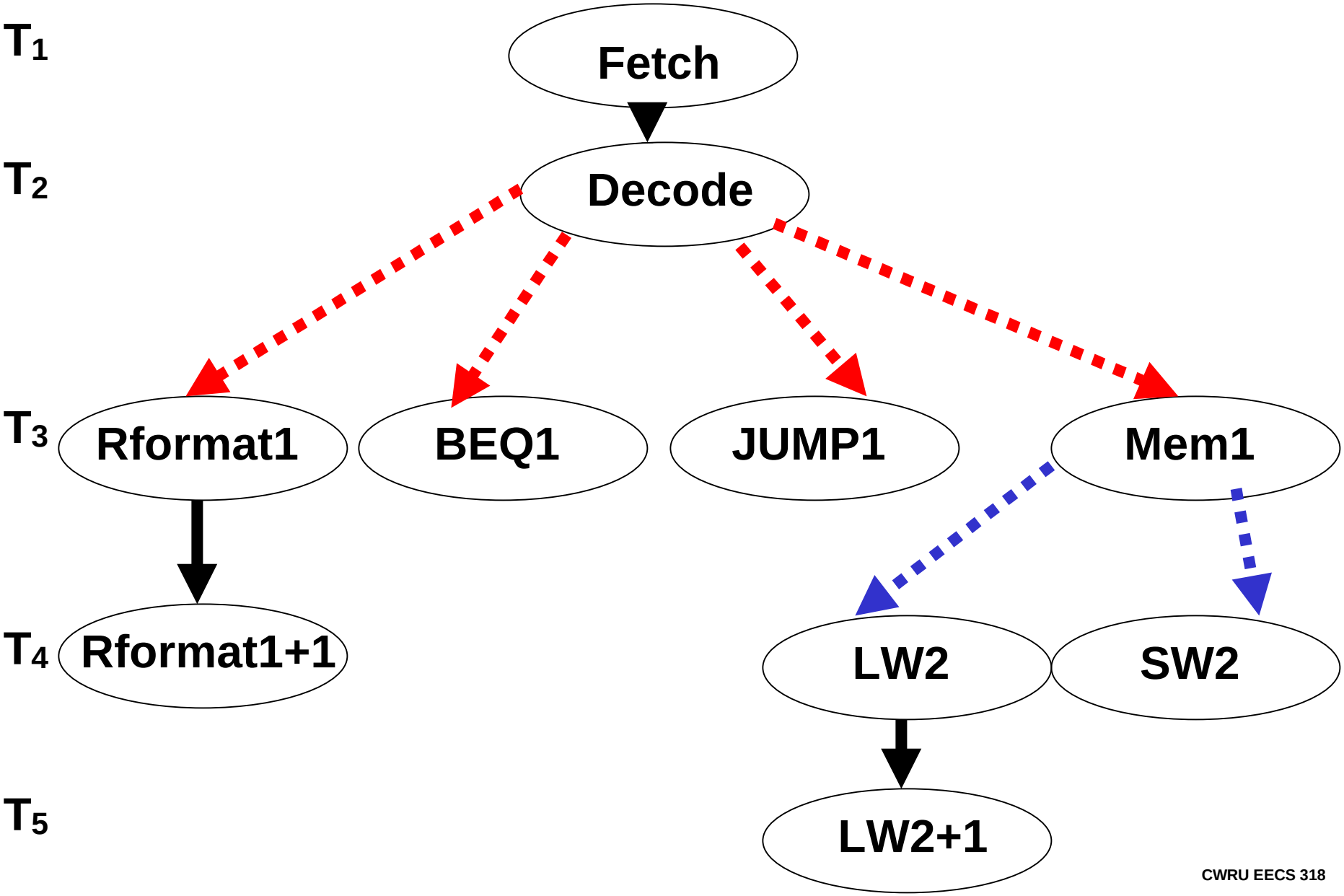


Multi-cycle using Finite State Machine

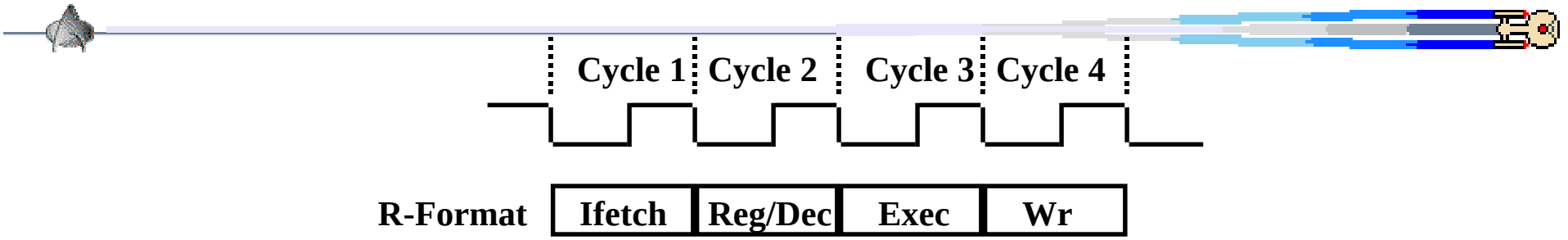
Finite State Machine
(*hardwired control*)



Finite State Machine: program overview

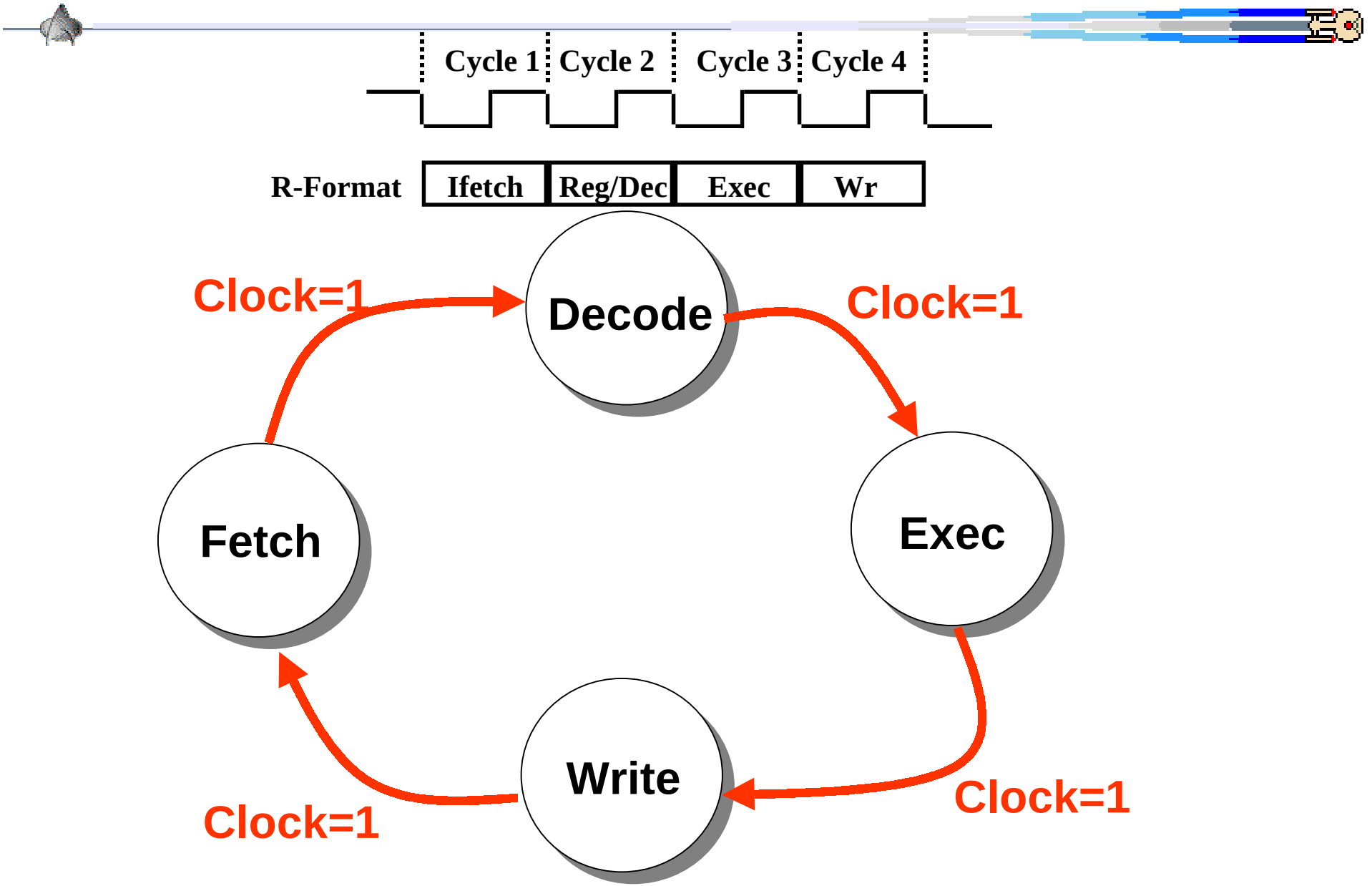


The Four Stages of R-Format

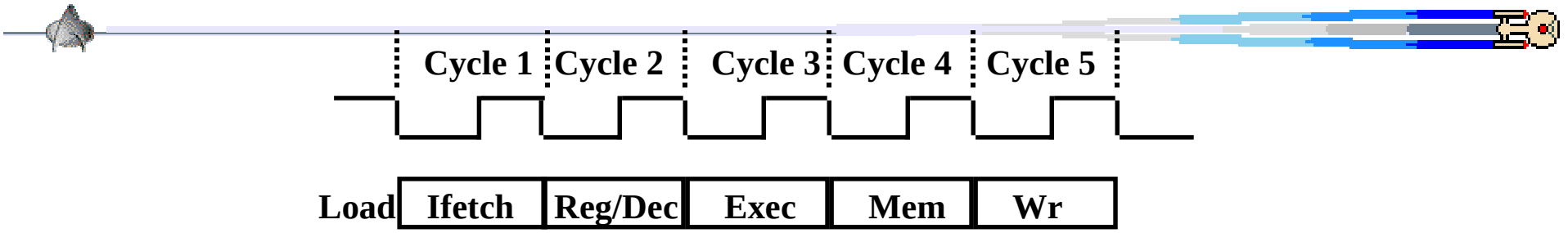


- **Fetch**: IR=MEM[$pc]; $pc=$pc+4;
  - Fetch the instruction from the Instruction Memory
- **Decode**: A=reg[$rs]; B=reg[$rt]; Datapath=(IR[31-26,5-0])
  - Registers Fetch and Instruction Decode for datapath
- **Exec**: ALUOut = A Funct B
  - ALU operates on the two register operands
  - Update PC
- **Write**: Reg[$rd]=ALUOut
  - Write the ALU output back to the register file

R-Format State Machine

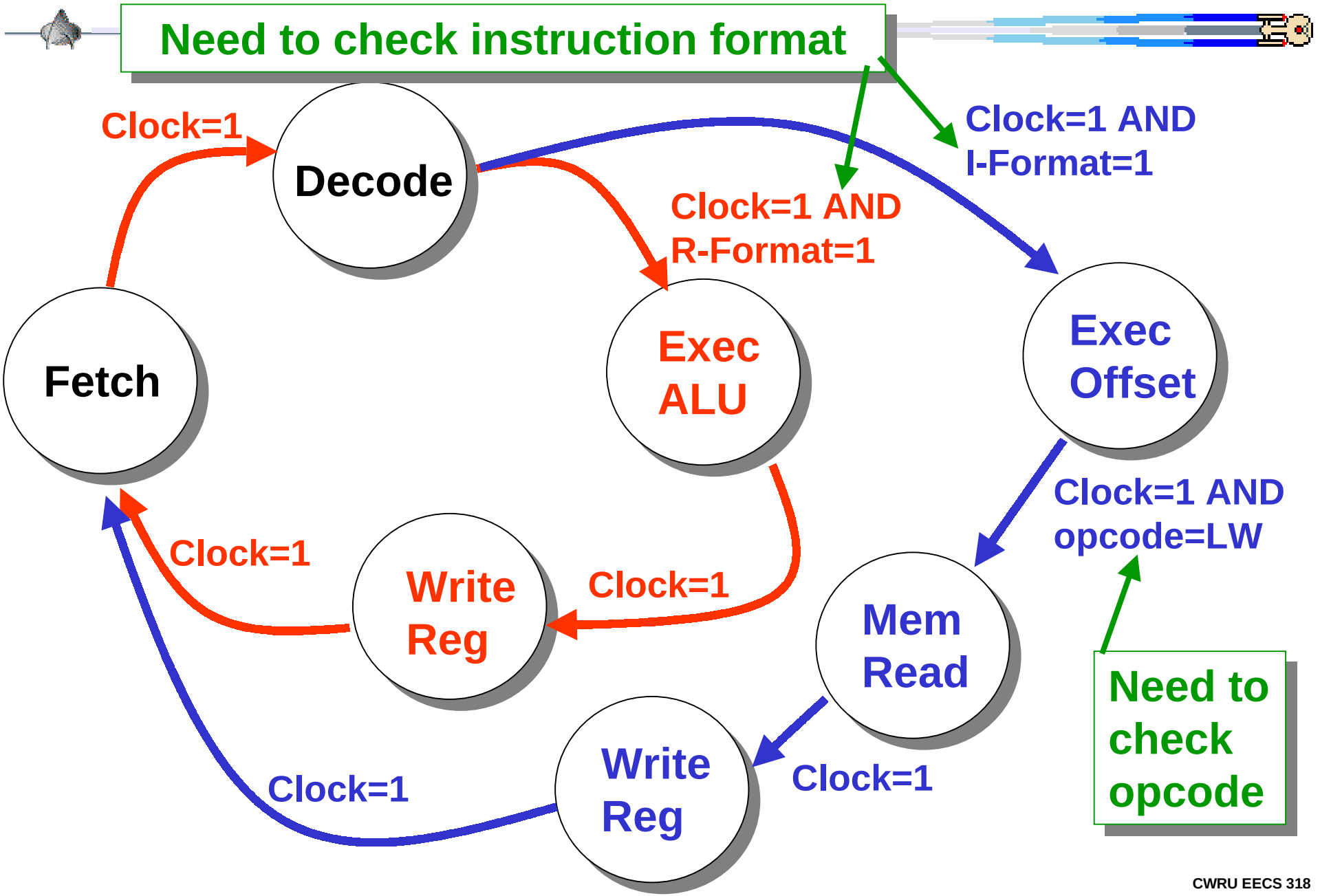


The Five Stages of Load Instruction

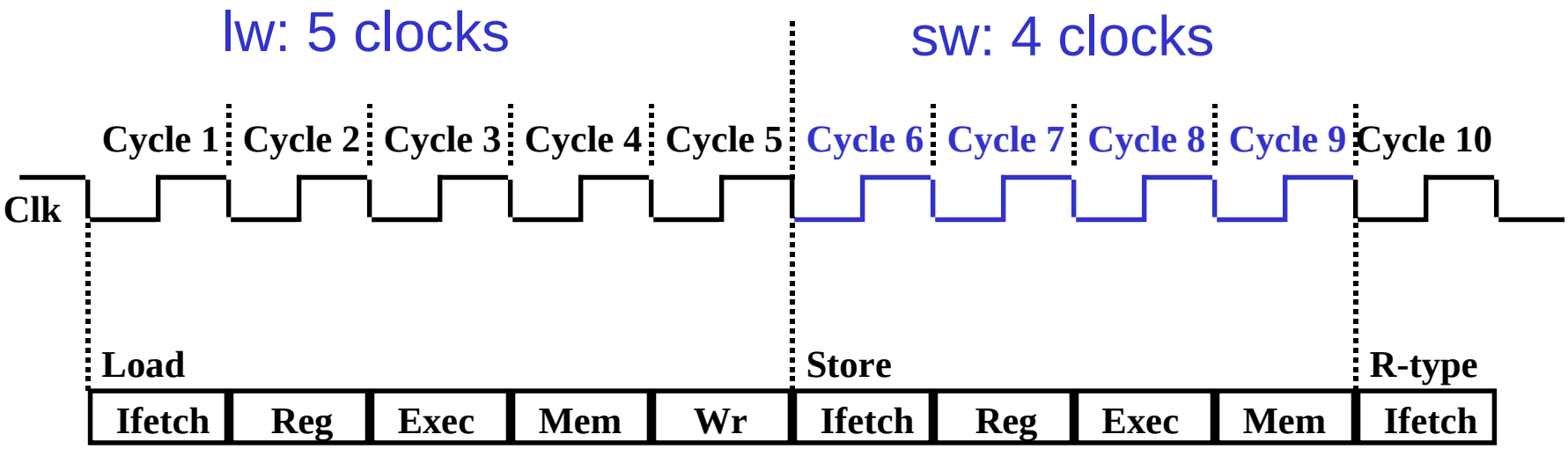


- **Fetch**: IR=MEM[$pc]; $pc=$pc+4;
  - Fetch the instruction from the Instruction Memory
- **Decode**: A=reg[$rs]; Datapath=(IR[31-26])
  - Registers Fetch and Instruction Decode for datapath
- **Exec**: ALUout = sign_extend(offset) + A
  - Calculate the memory address
- **Mem**: MDR = Mem[ALUOut]
  - Read the data from the Data Memory
- **Wr**: Reg[$rt]=MDR
  - Write the data back to the register file

R-Format & I-Format State Machine

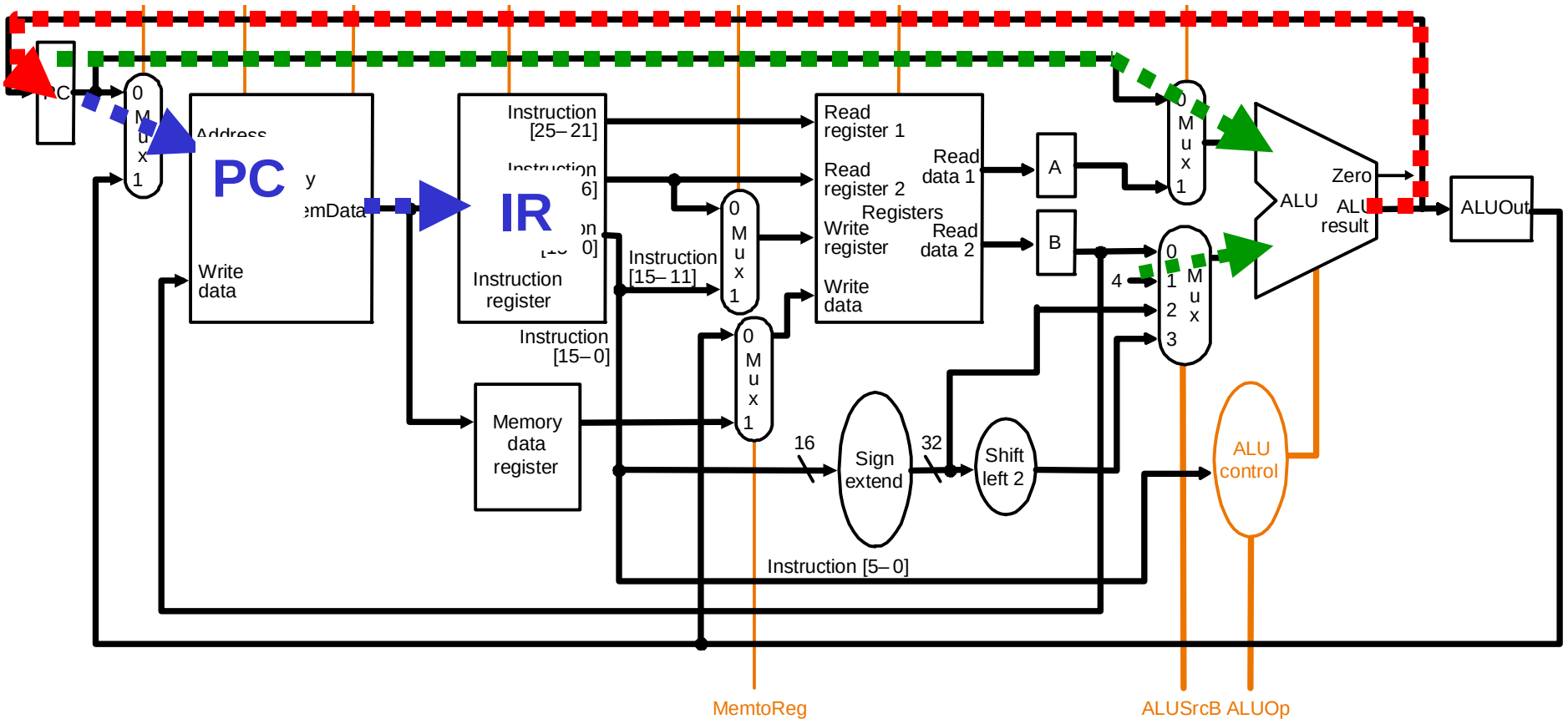


Multi-Instruction sequence

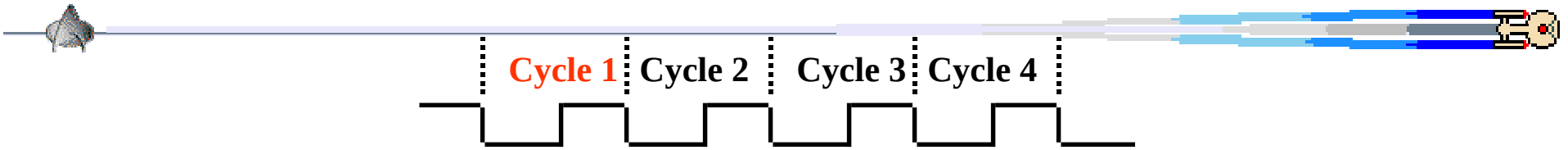


State machine stepping: T_1 Fetch

(Done in parallel) IR←MEMORY[PC] & PC ← PC + 4



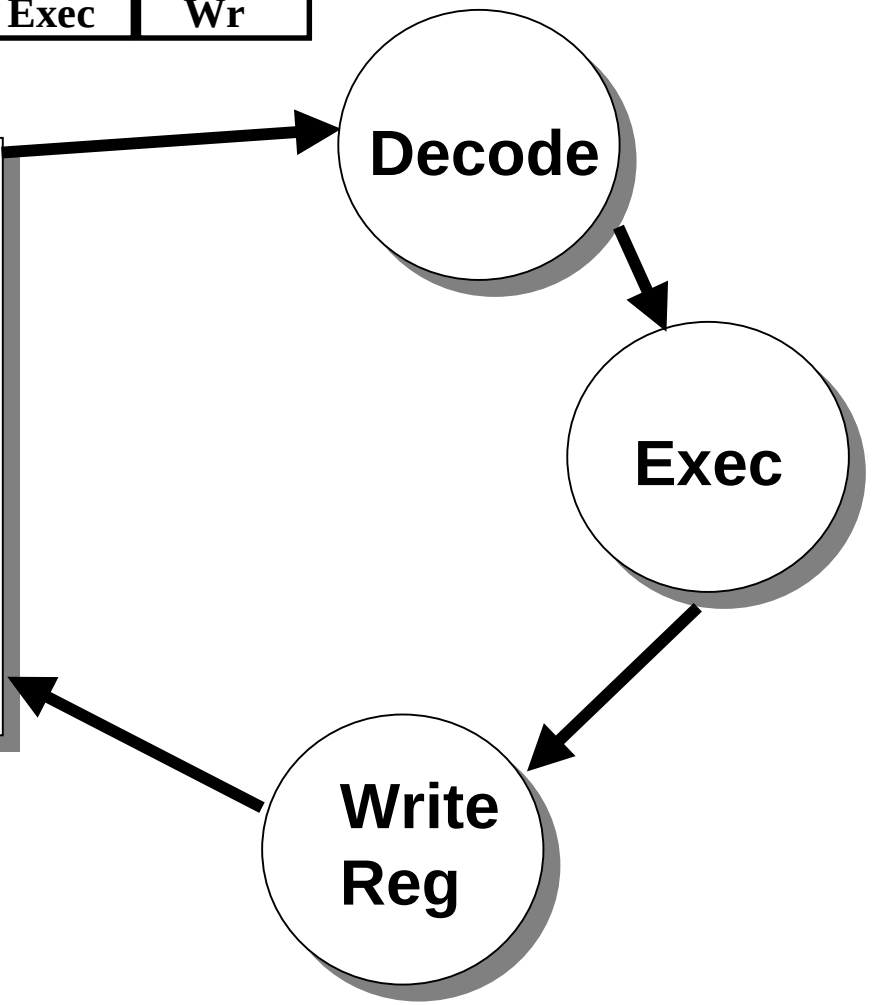
T_1 Fetch: State machine



Start

MemRead=1, MemWrite=0
IorD=1 (MemAddr←PC)
IRWrite=1 (IR←Mem[PC])
ALUSrcA=0 (=PC)
ALUSrcB=1 (=4)
ALUOP=ADD (PC←4+PC)
PCWrite=1, PCSource=1 (=ALU)
RegWrite=0,
MemtoReg=X, RegDst=X

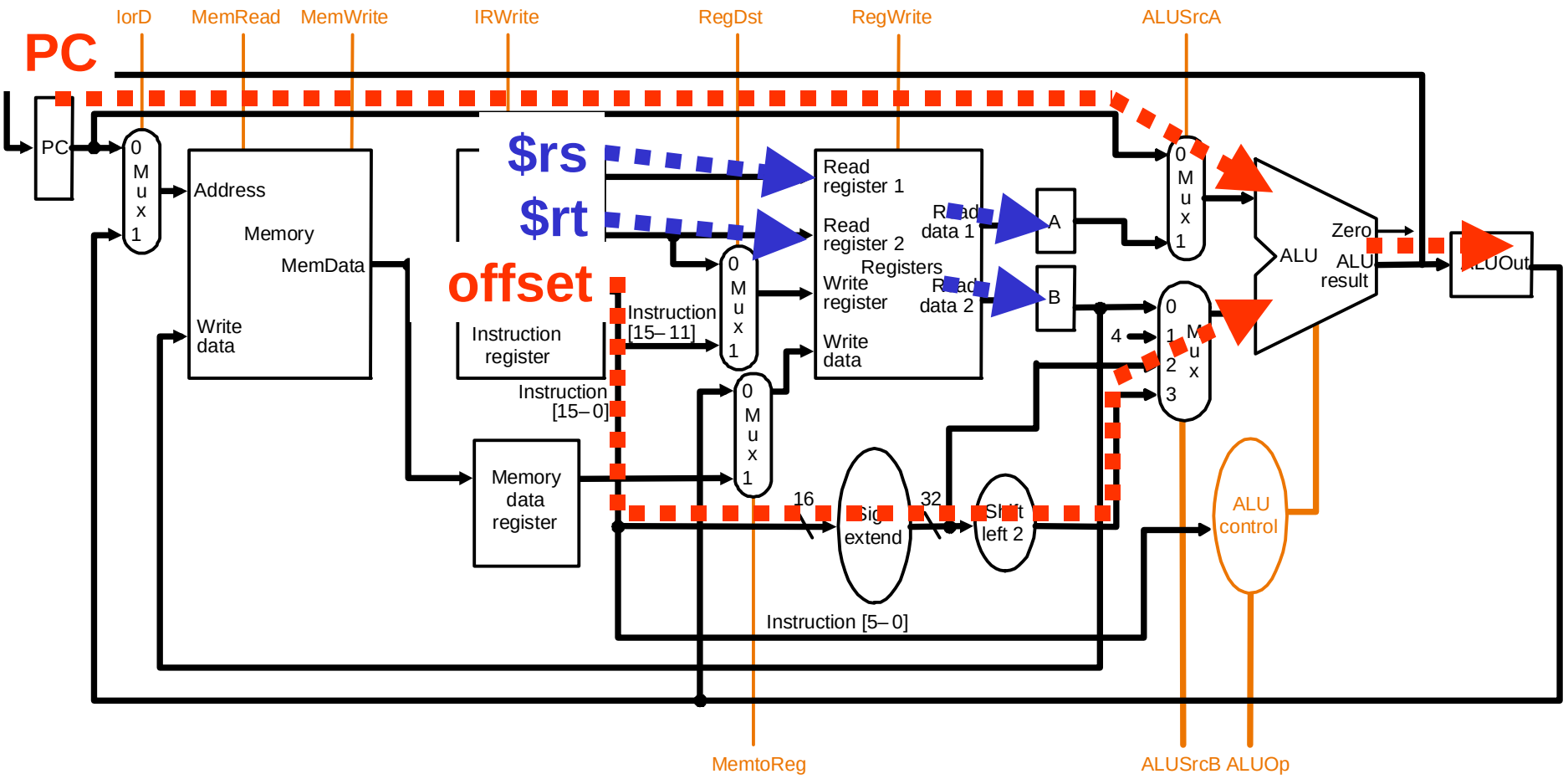
Instruction Fetch



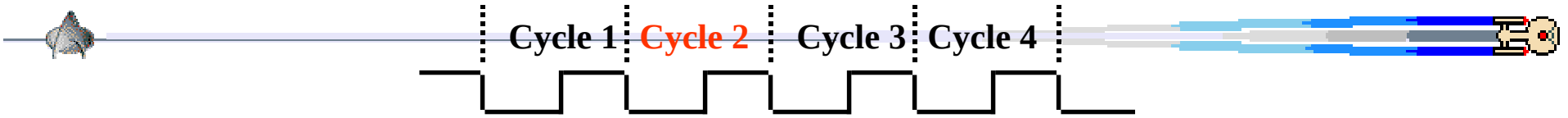
T_2 Decode (read $rs and $rt and offset+pc)

A←Reg[IR[25-21]] & B←Reg[IR[20-16]]

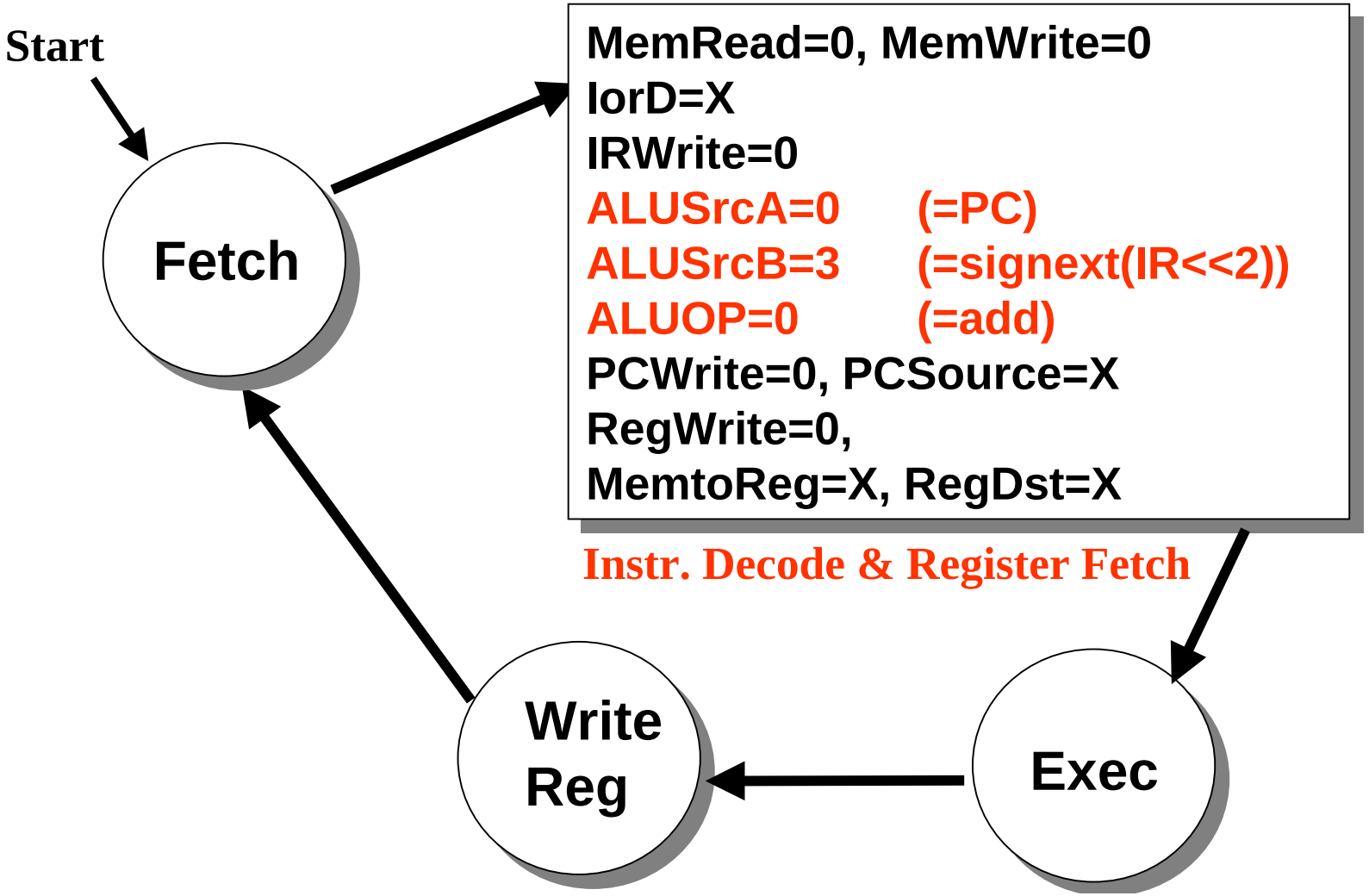
& ALUOut←PC+signext(IR[15-0]) <<2



T_2 Decode State machine

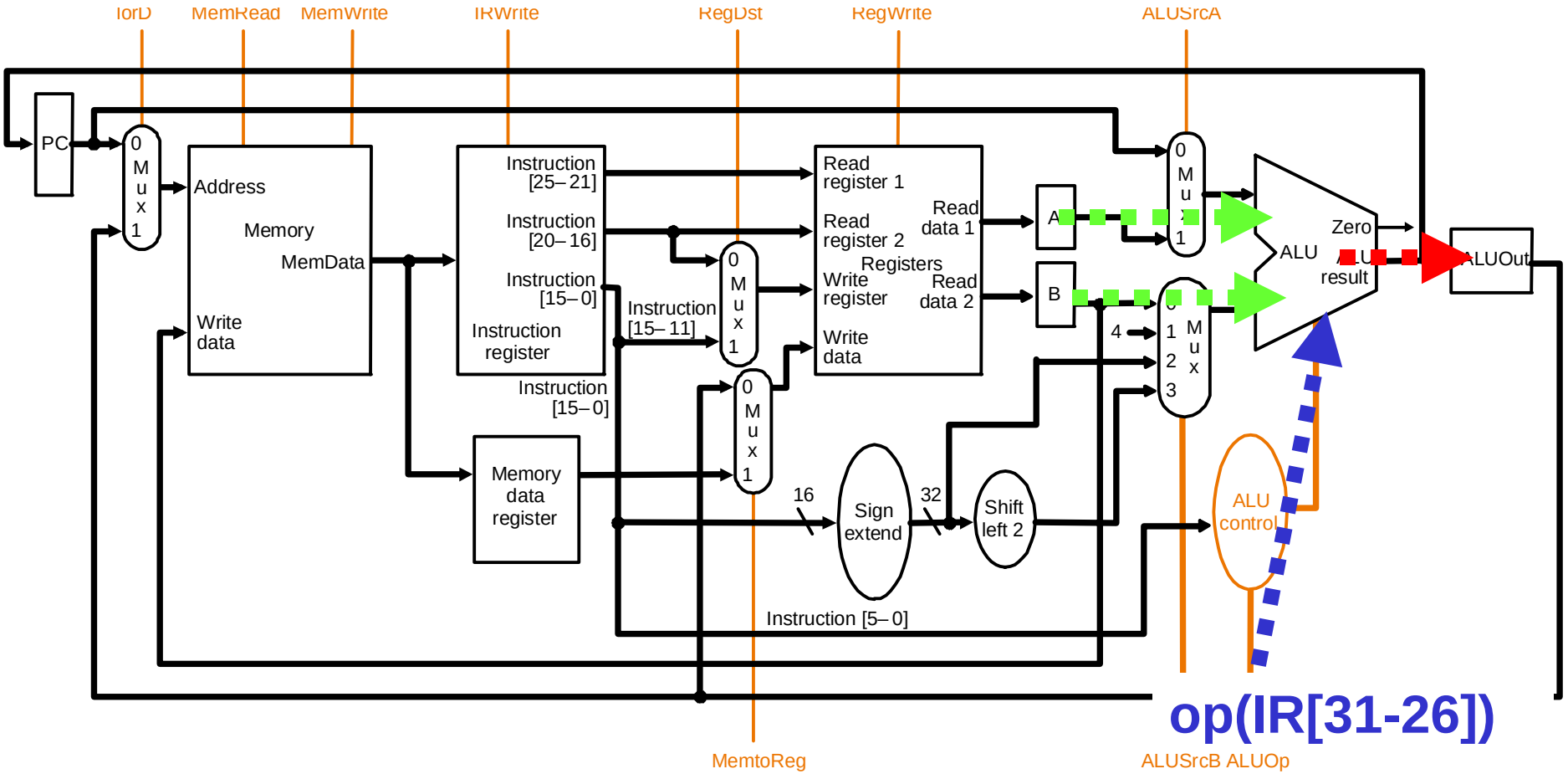


R-Format	Ifetch	Reg/Dec	Exec	Wr

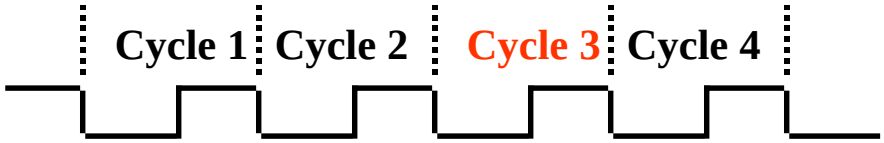


T_3 ExecALU (ALU instruction)

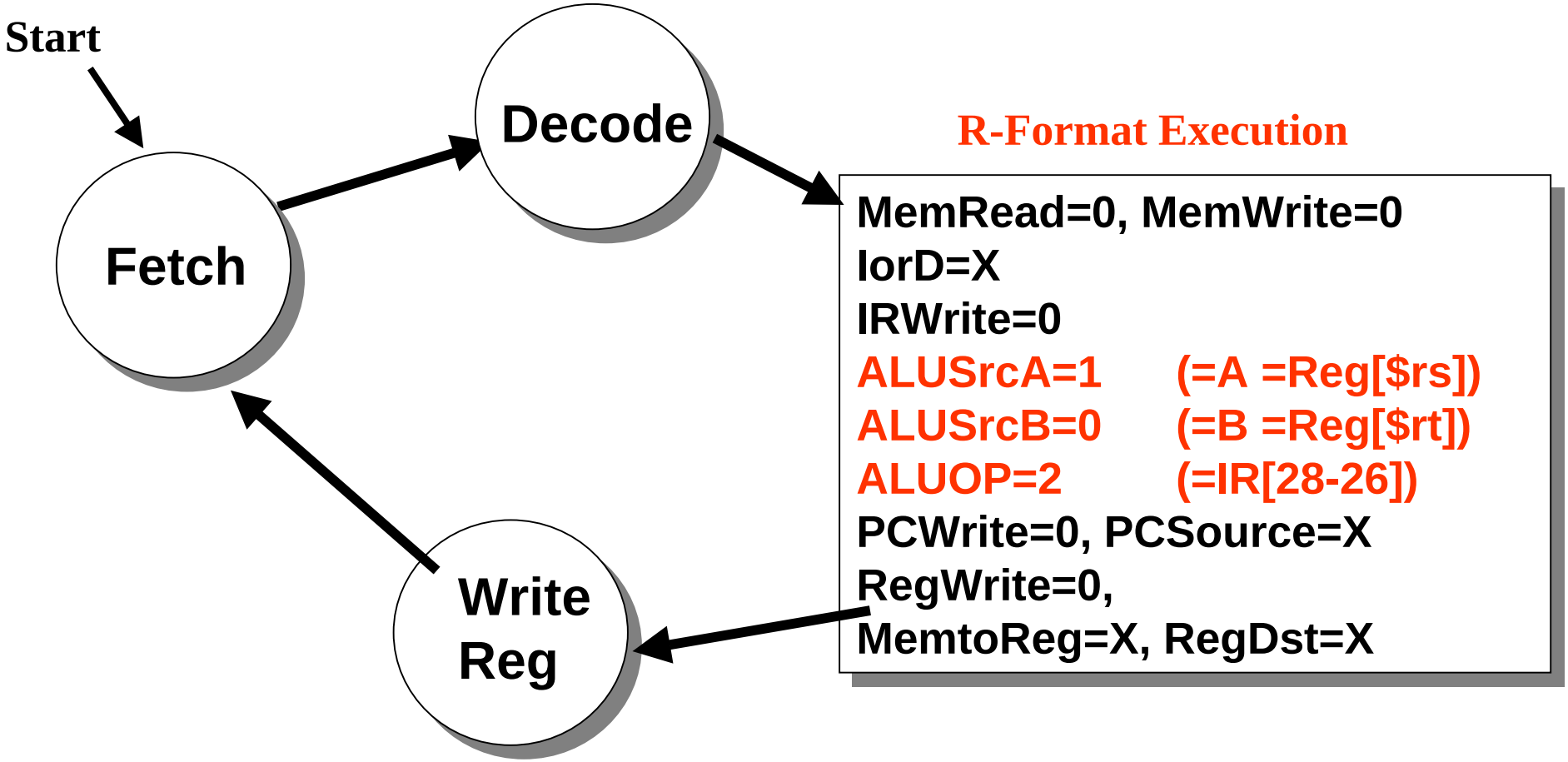
ALUOut ← A op(IR[31-26]) B



T_3 ExecALU State machine



R-Format	Ifetch	Reg/Dec	Exec	Wr

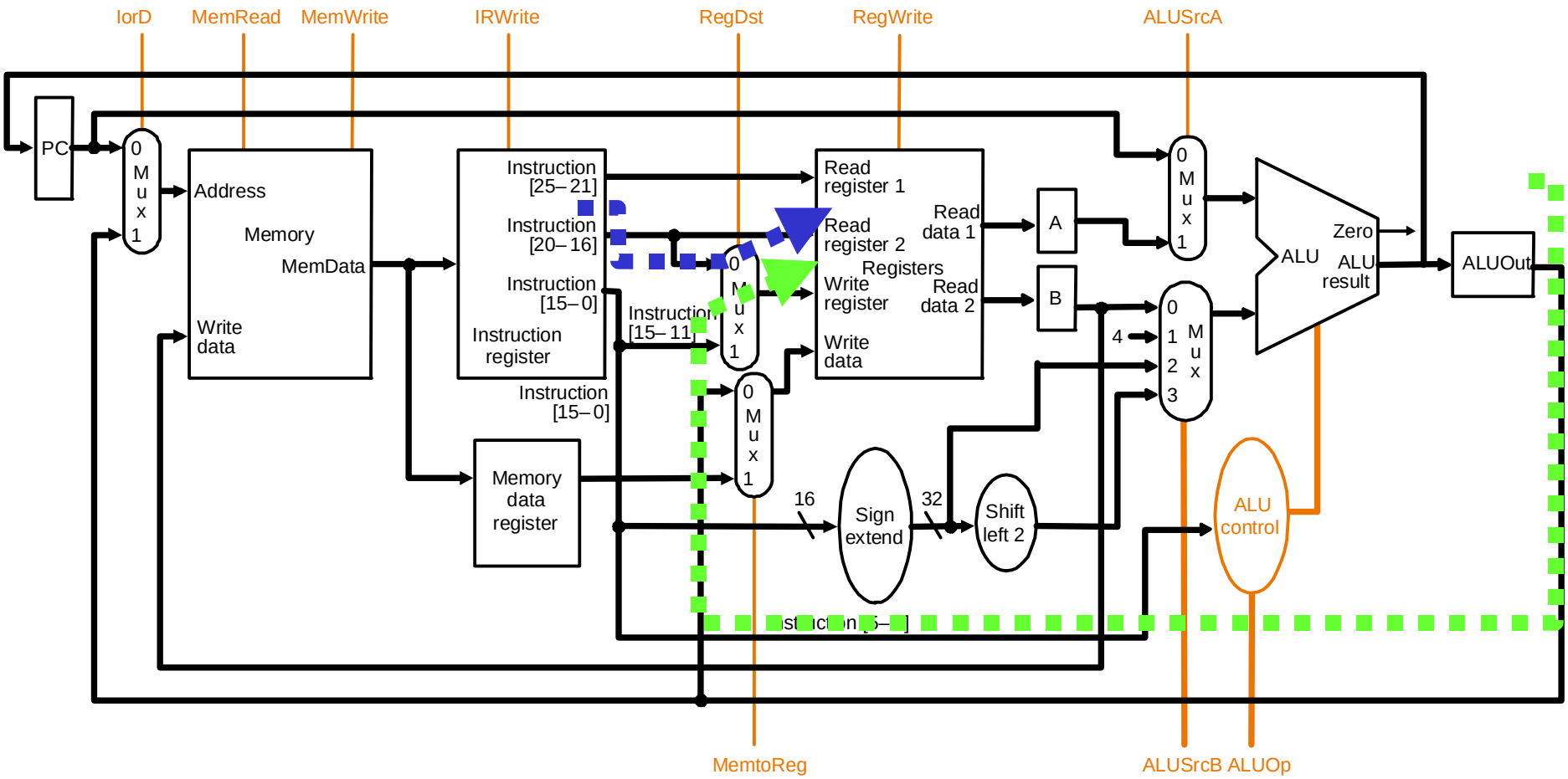


R-Format Execution

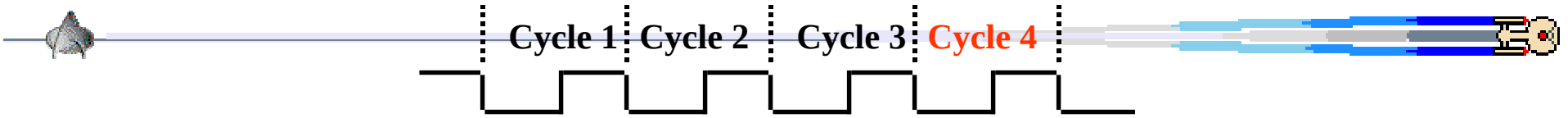
MemRead=0, MemWrite=0
IorD=X
IRWrite=0
ALUSrcA=1 (=A =Reg[$rs])
ALUSrcB=0 (=B =Reg[$rt])
ALUOP=2 (=IR[28-26])
PCWrite=0, PCSource=X
RegWrite=0,
MemtoReg=X, RegDst=X

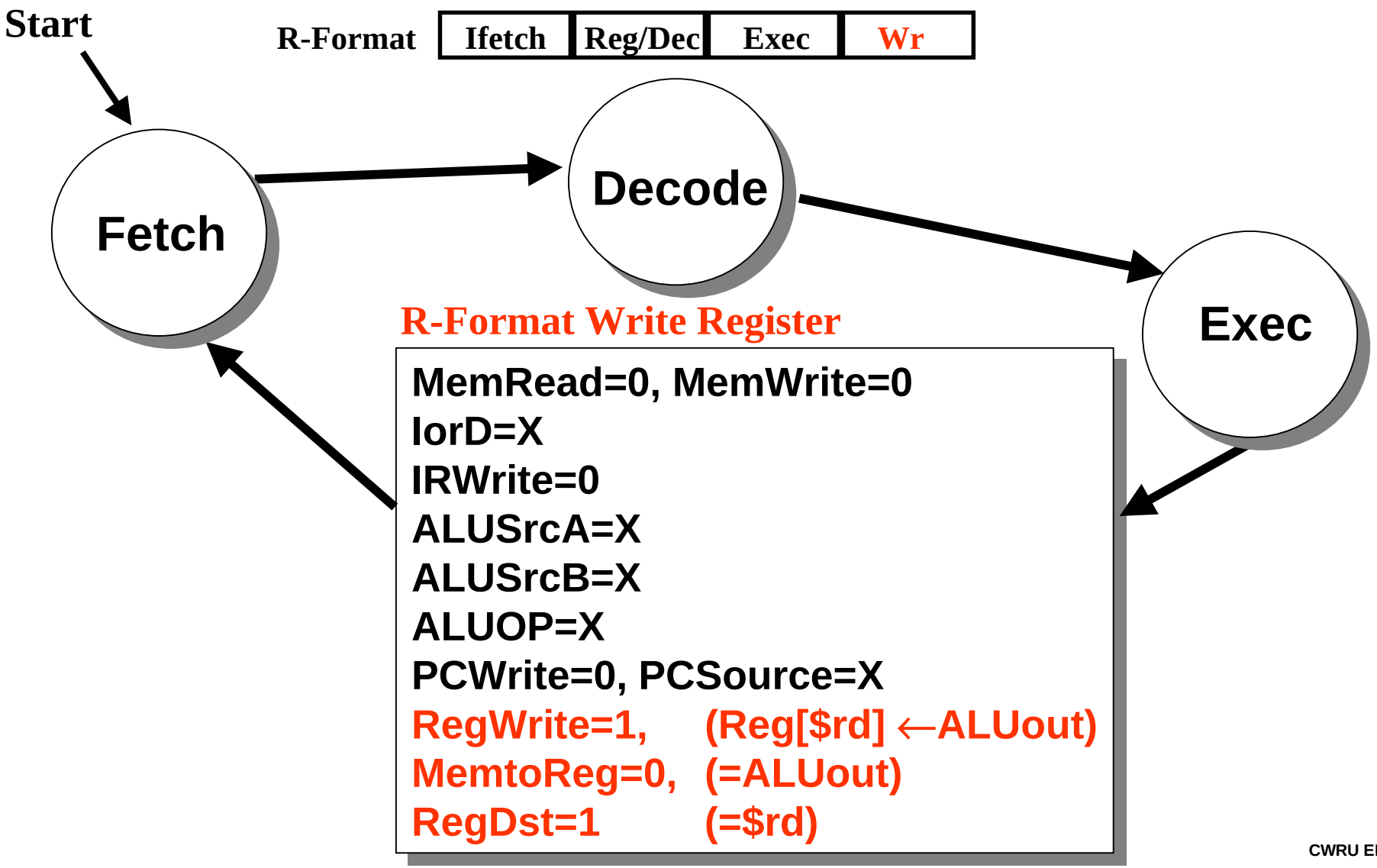
T_4 WrReg (ALU instruction)

Reg[IR[15-11]] ← ALUOut



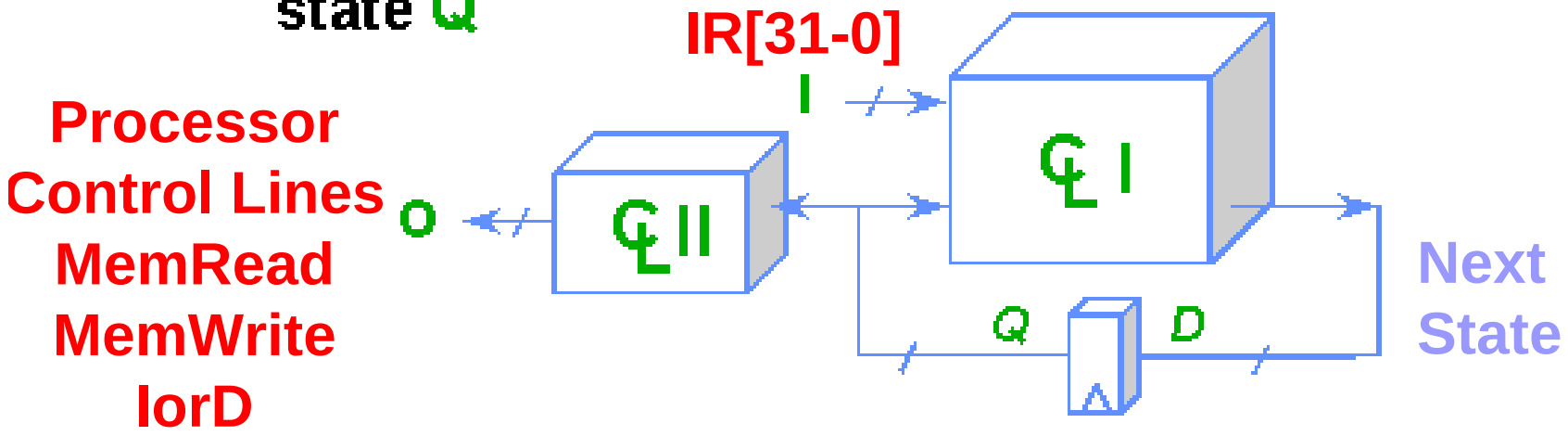
T_4 WrReg State machine





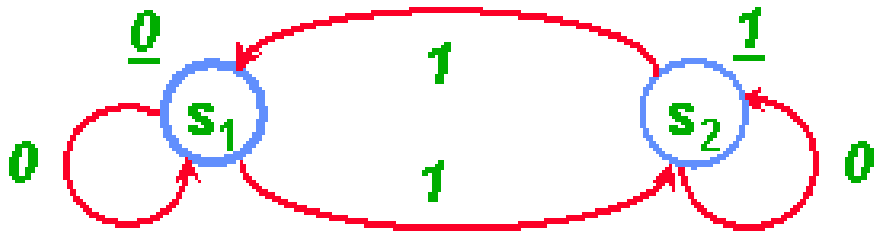
Moore Machines

- So far we considered Moore machines where the output O is a function of only the current state Q



- Moore FSM State Transition Graph

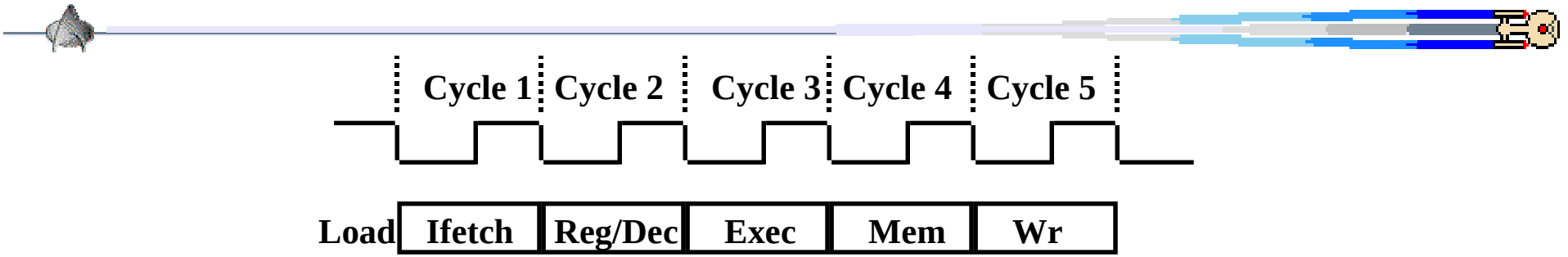
...



Moore Output State Tables: O(State)

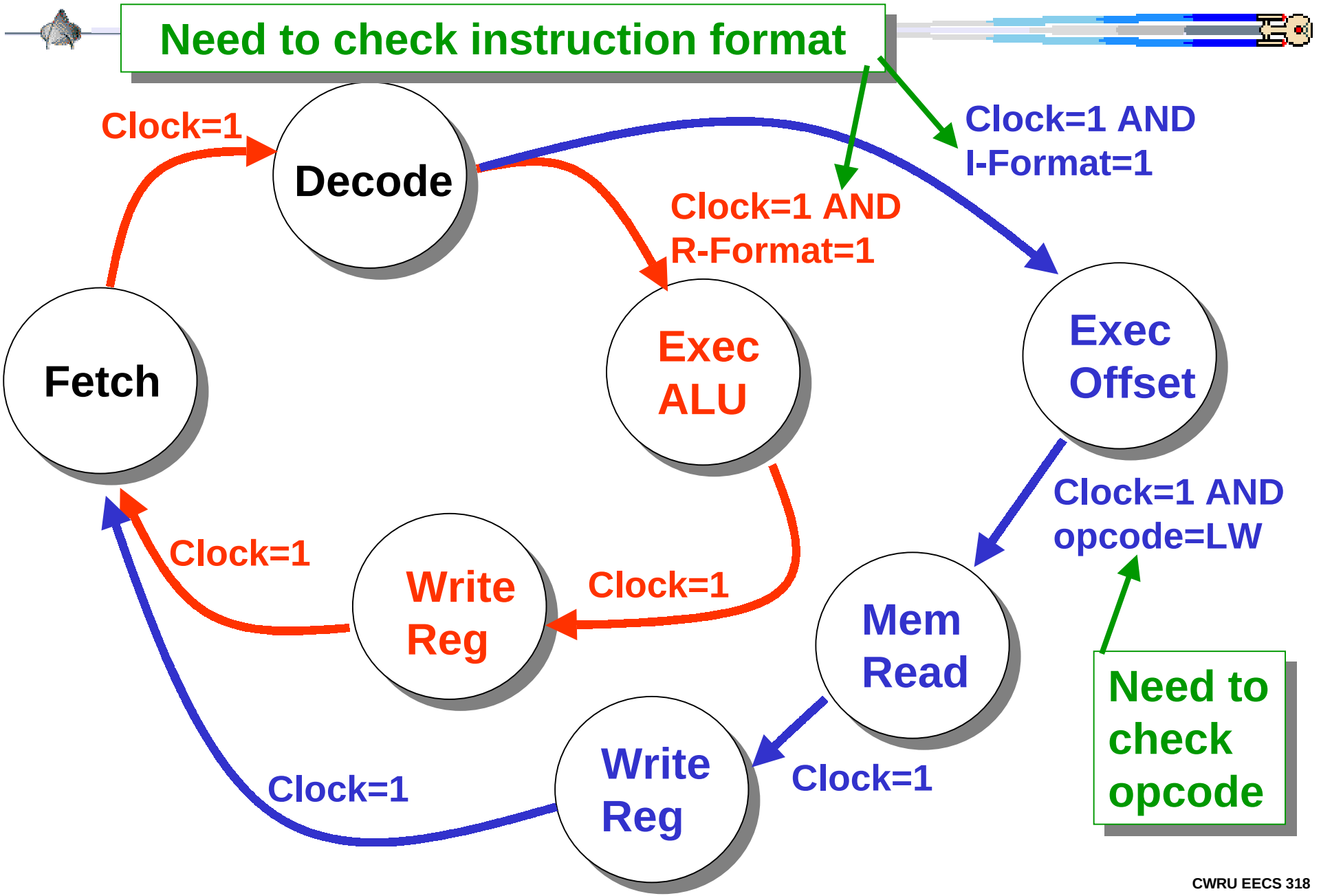
State	T_1	T_2	T_3-R	T_4-R
MemRead	1	0	0	0
MemWrite	0	0	0	0
MUX IorD	0 =PC	X	X	X
IRWrite	1	0	0	0
ALUOP	0 +	0 +	2 =op	X
MUX ALUSrcA	0 =PC	0 =PC	1 =A =$rs	X
MUX ALUSrcB	1 =4	3 =offset	0 =B =$rt	X
PCWrite	1	0	0	0
MUX PCSource	0 =ALU	X	X	X
RegWrite	0	0	0	1
MUX MemtoReg	X	X	X	0 =ALUOut
MUX RegDst	X	X	X	1 =$rd

Review: The Five Stages of Load Instruction



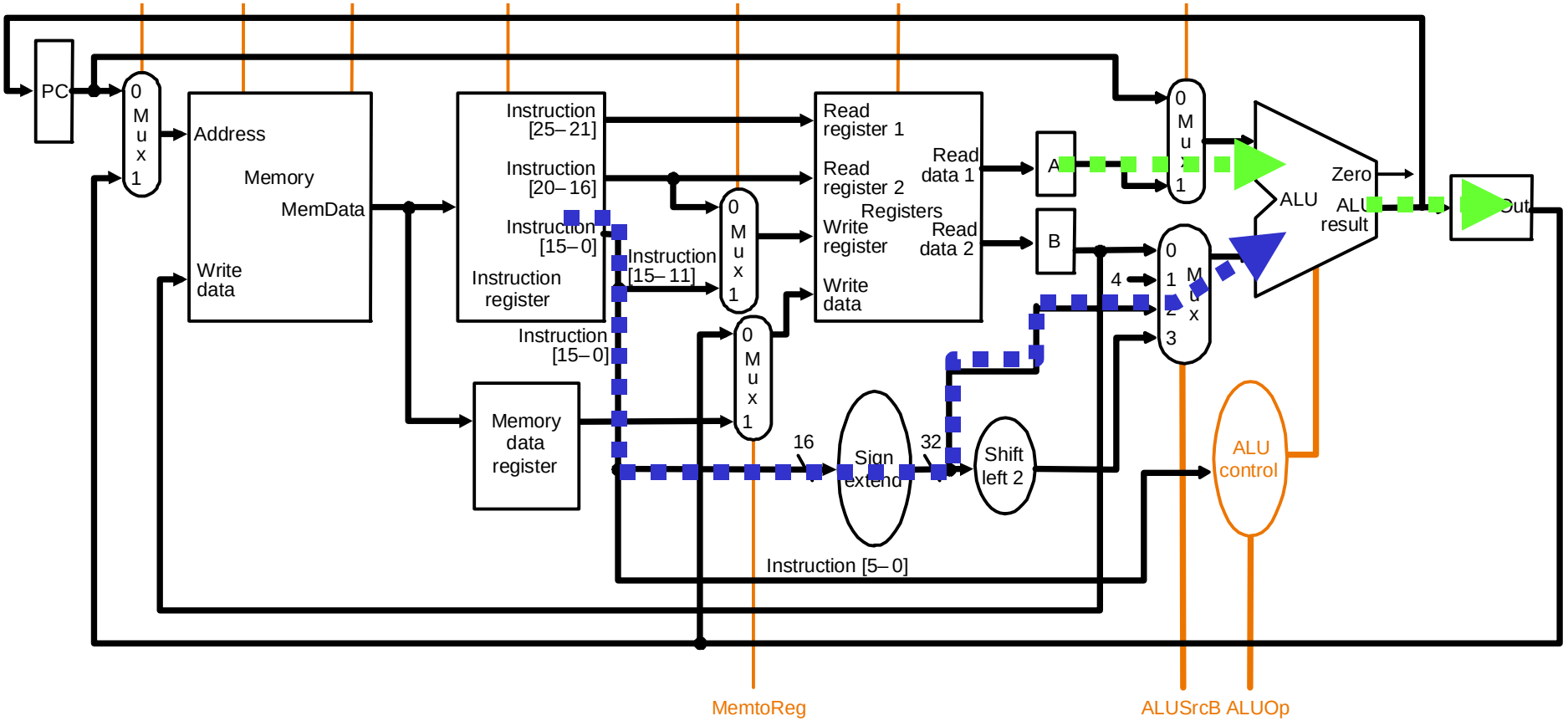
- **Fetch**:
  - Fetch the instruction from the Instruction Memory
- **Decode**:
  - Registers Fetch and Instruction Decode
- **Exec**: **Offset**
  - Calculate the memory offset
- **Mem**:
  - Read the data from the Data Memory
- **Wr**:
  - Write the data back to the register file

Review: R-Format & I-Format State Machine

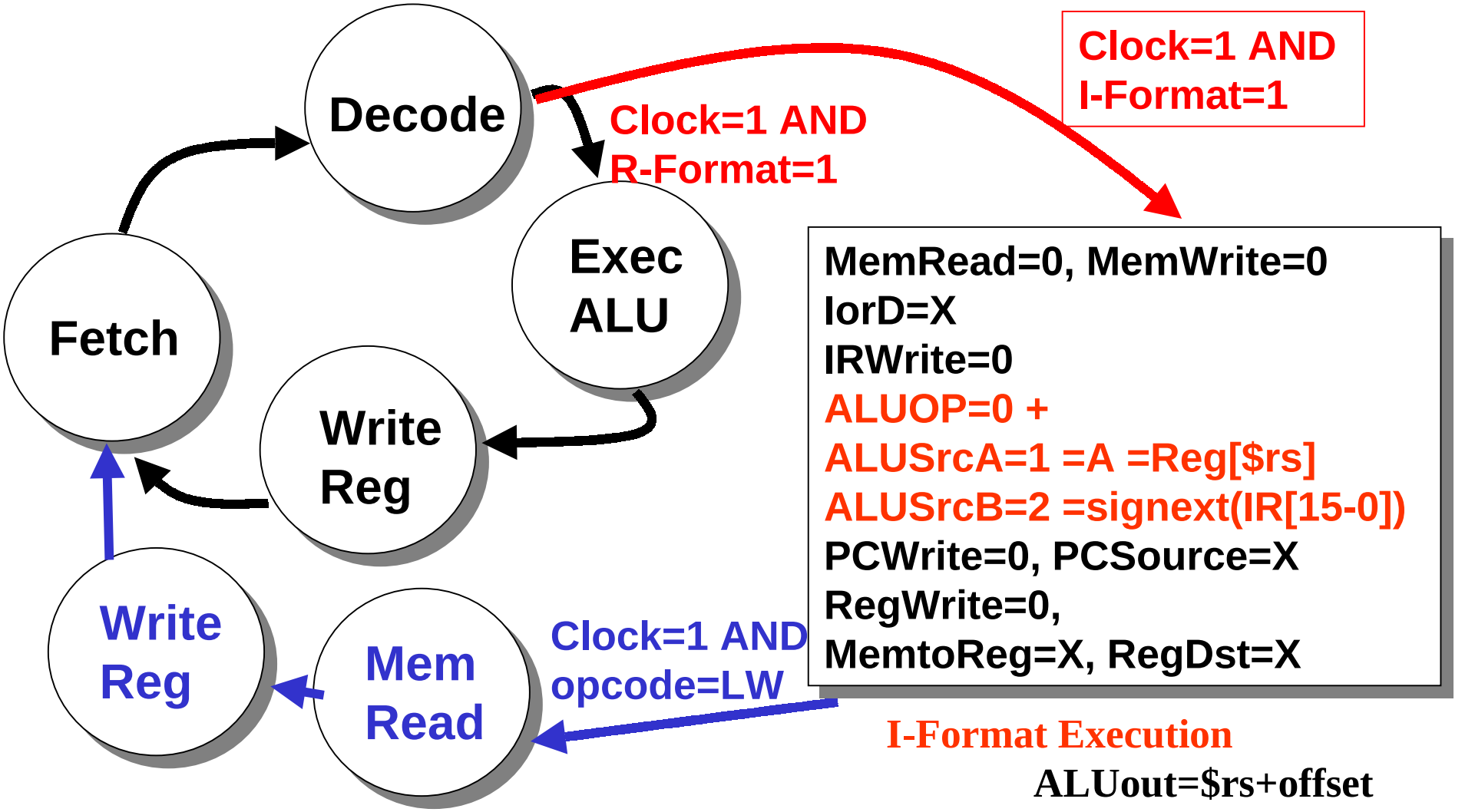


T_3–I Mem1 (common to both load & store)

ALUOut ← A + sign_extend(IR[15-0])

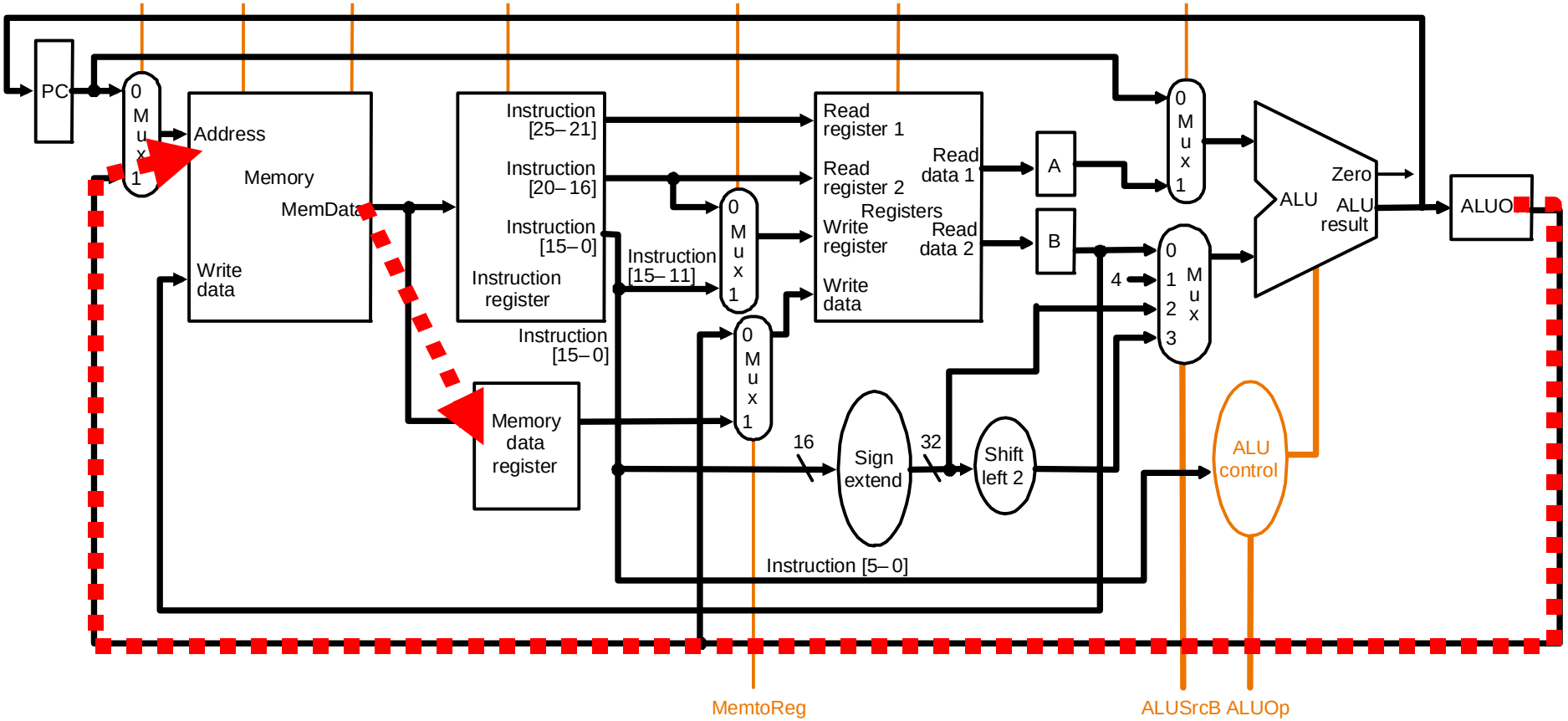


T_3 Mem1 I-Format State Machine =$rs + offset

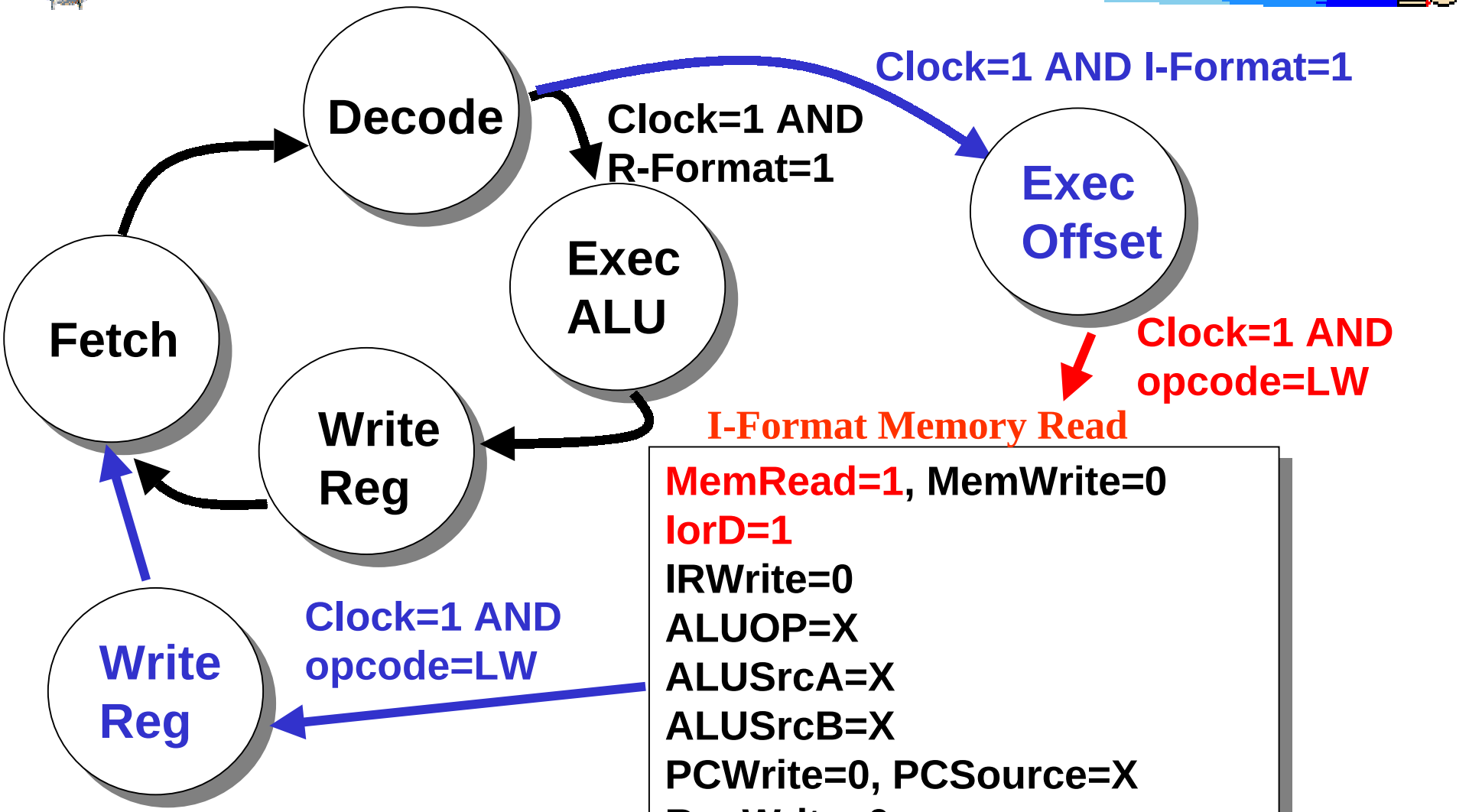


T_4 –LW1 : load instruction, read memory

MDR ← Memory[ALUOut]

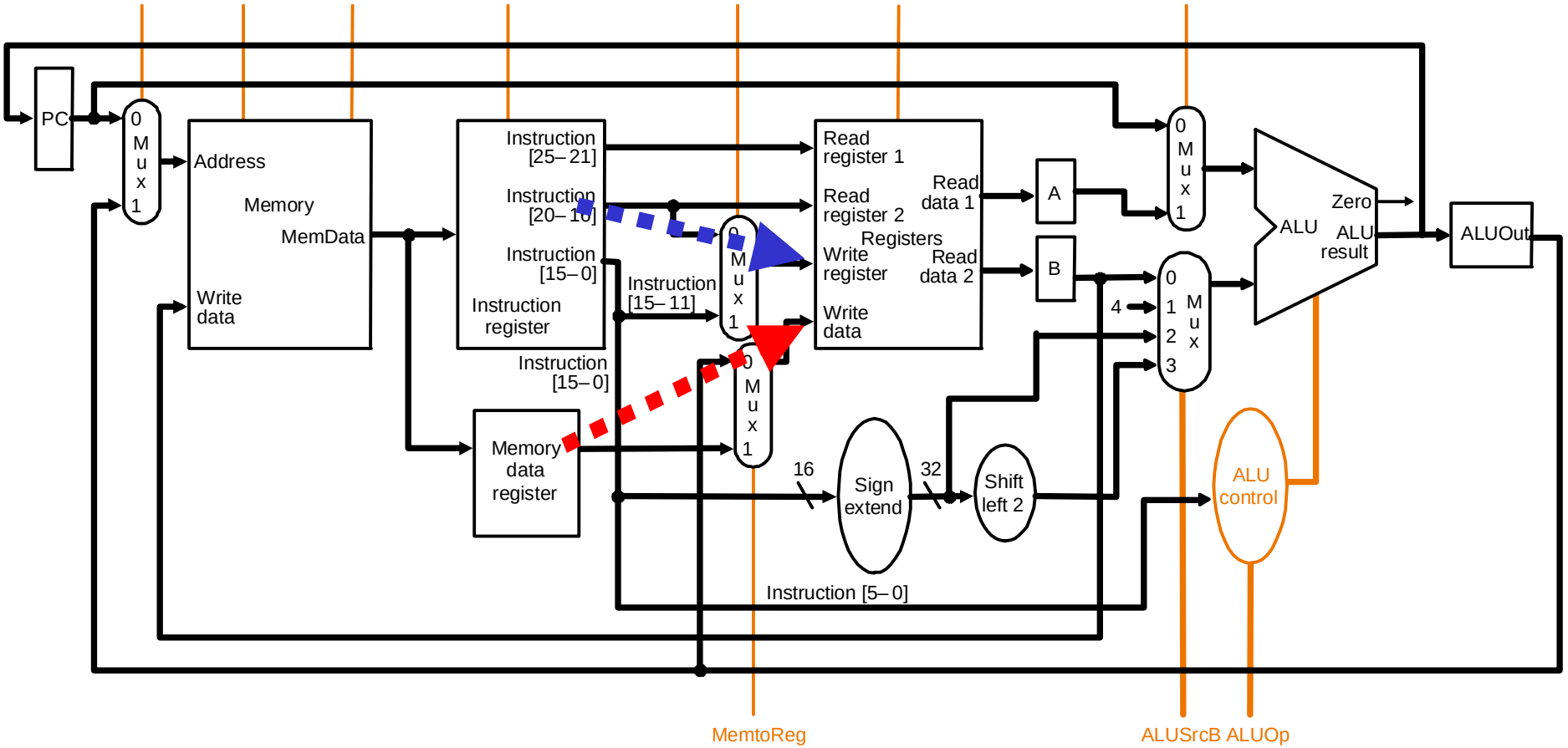


T_4 LW2 I-Format State Machine =Mem[ALU]

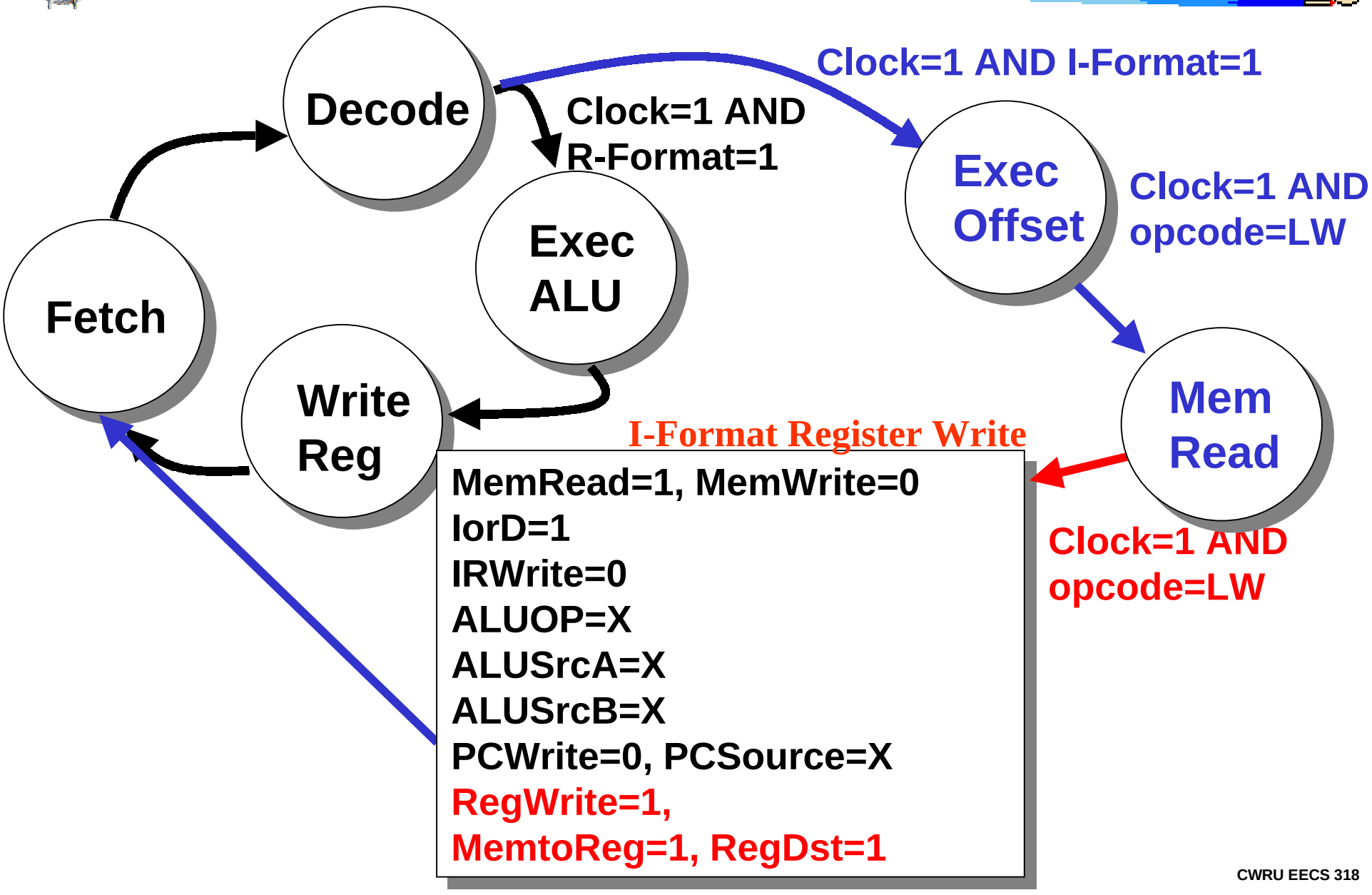


T_5 –LW2 Load instruction, write to register

Reg[IR[20-16]] ← MDR

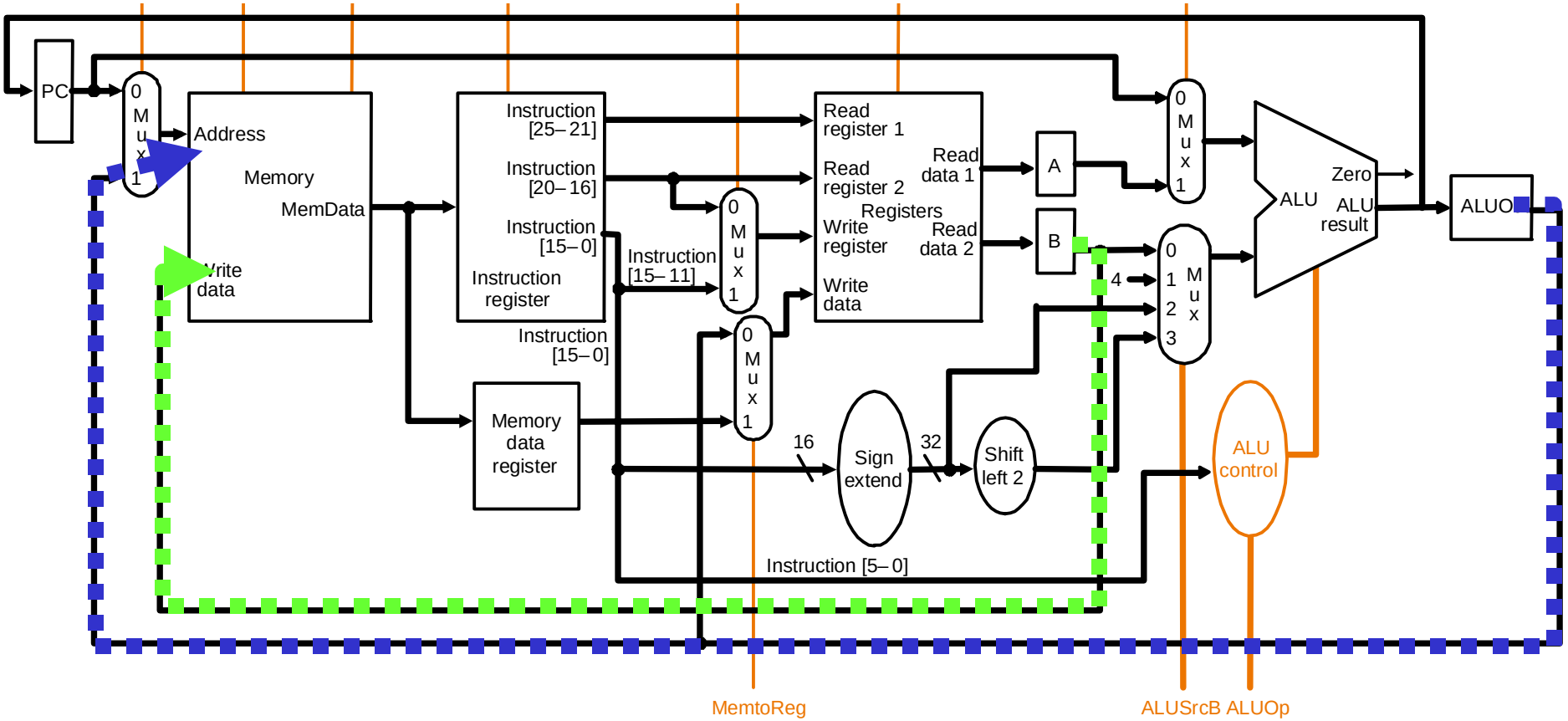


T_5 LW2 I-Format State Machine $rt=MDR

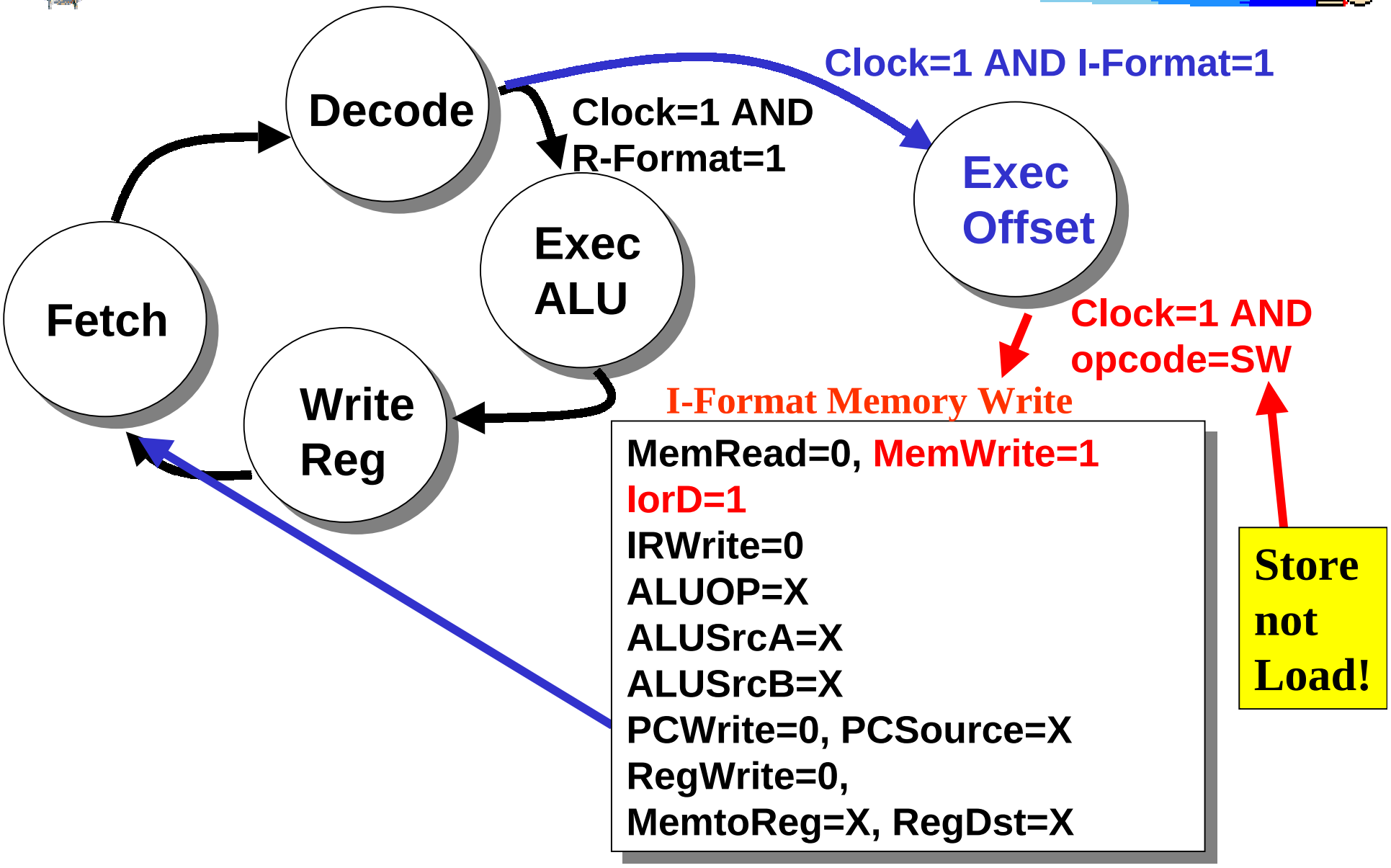


T_4–SW2 Store instruction, write to memory

Memory[ALUOut] ← B

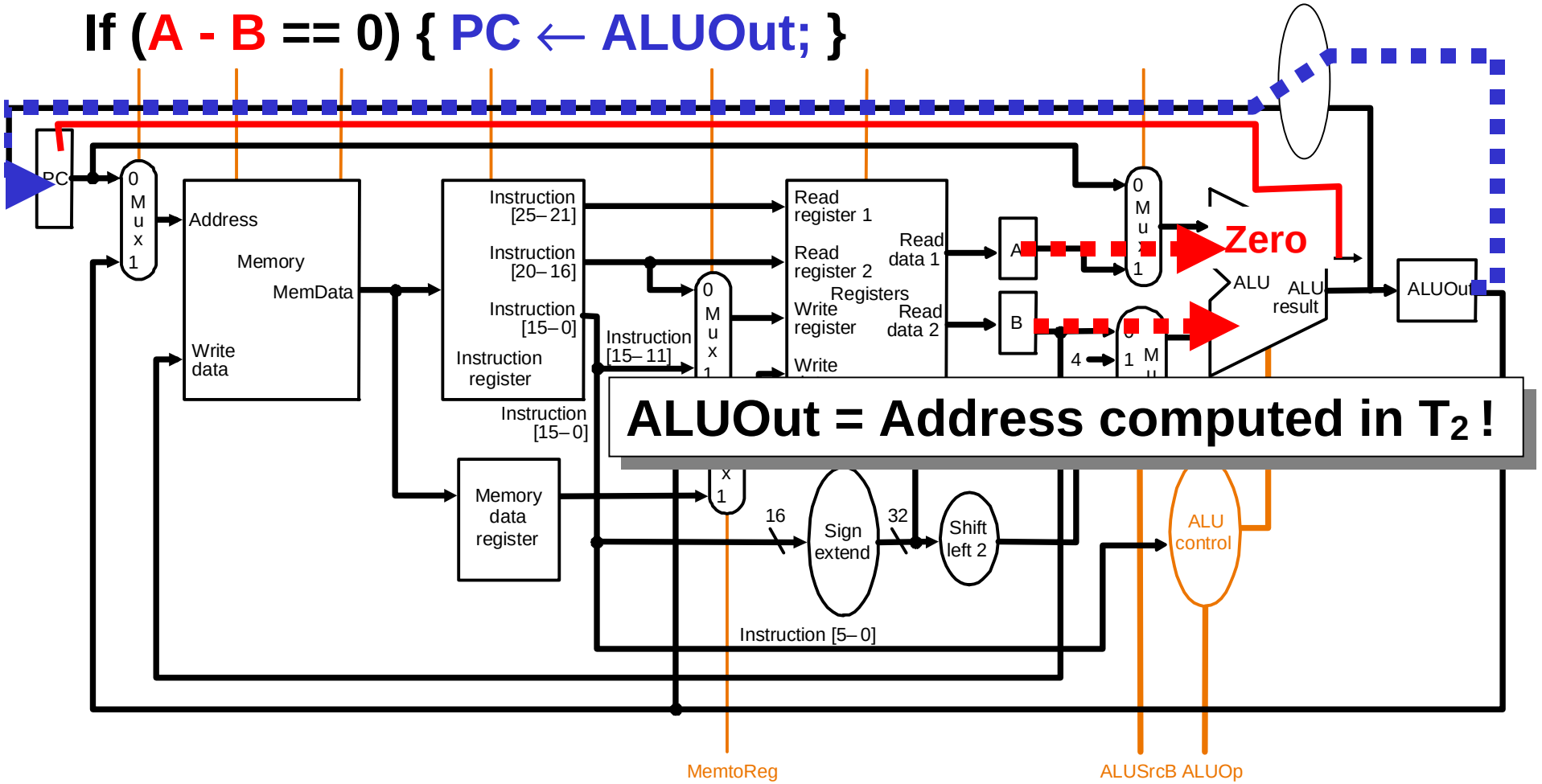


T_4 SW2 I-Format State Machine Mem[ALU]=



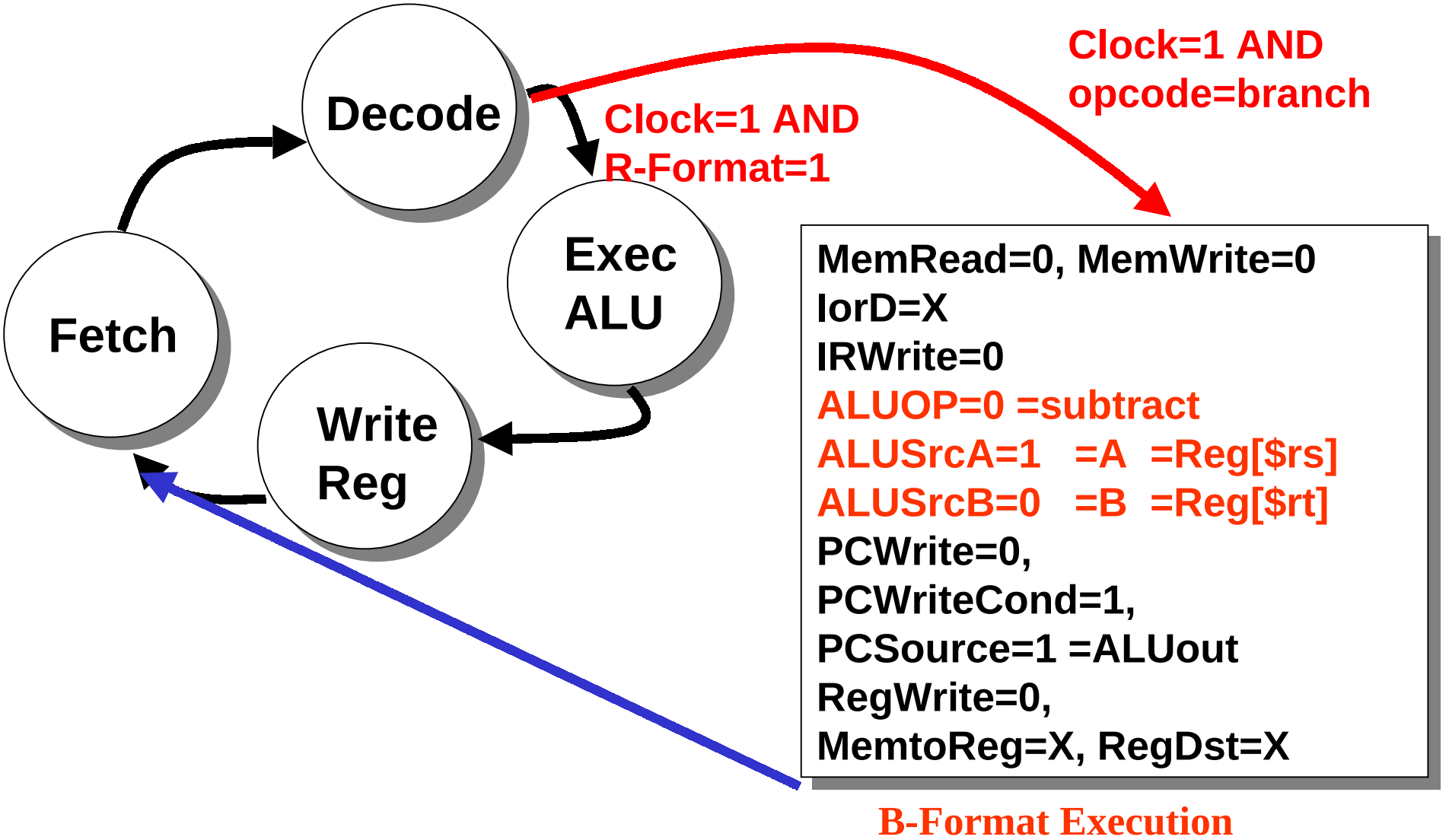
T_3 BEQ1 (Conditional branch instruction)

If (A - B == 0) { PC ← ALUOut; }



ALUOut = Address computed in T_2 !

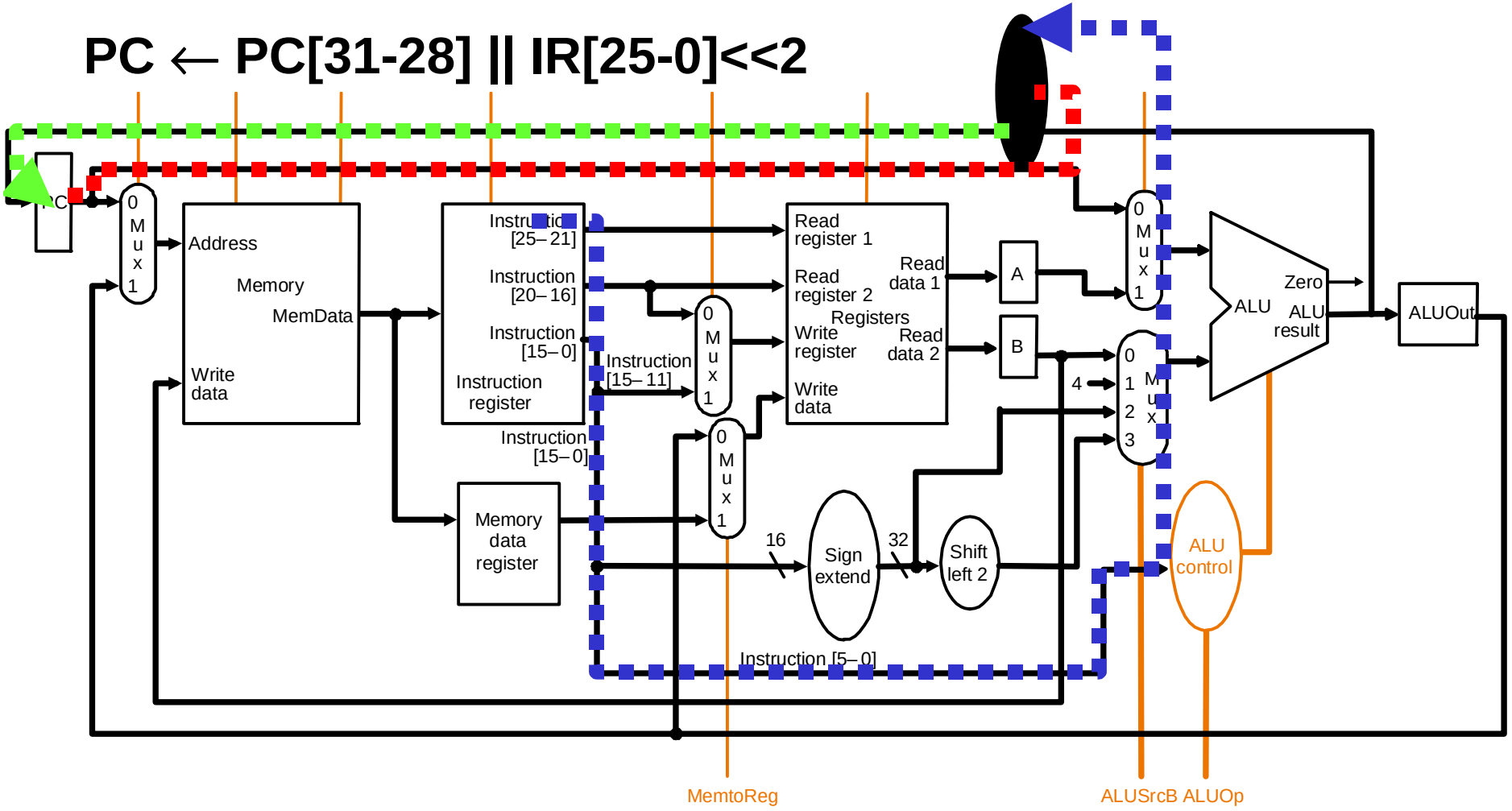
T_3 BEQ1 I-Format State Machine =$rs + offset



B-Format Execution

T_3 Jump1 (Jump Address)

PC ← PC[31-28] || IR[25-0]<<2



Moore Output State Tables: O(State)

State	T_1	T_2	T_3-R	T_4-R	T_3-I	T_4-SW	T_4-LW	T_5-LW
MemRead	1	0	0	0	0	0	1	0
MemWrite	0	0	0	0	0	1	0	0
MUX IorD	0=PC	X	X	X	X	1=ALU	1=ALU	X
IRWrite	1	0	0	0	0	0	0	0
ALUOP	0=+	0	2=op	X	0=add	X	X	X
MUX ALUSrcA	0=PC	0	1=A=$rs	X	1=A=$rs	X	X	X
MUX ALUSrcB	1=4	3	0=B=$rt	X	2=sign	X	X	X
PCWrite	1	0	0	0	0	0	0	0
MUX PCSource	0=AL	X	X	X	X	X	X	X
RegWrite	0	0	0	1	0	0	0	1
MUX MemtoReg	X	X	X	0=ALU	X	X	X	1=MDR
MUX RegDst	X	X	X	1=$rd	X	X	X	1=$rt

Multi-cycle: 5 execution steps

- T_1 (a,lw,sw,beq,j) Instruction Fetch
- T_2 (a,lw,sw,beq,j) Instruction Decode and Register Fetch
- T_3 (a,lw,sw,beq,j) Execution, Memory Address Calculation, or Branch Completion
- T_4 (a,lw,sw) Memory Access or R-type instruction completion
- T_5 (lw) Write-back step

INSTRUCTIONS TAKE FROM 3 - 5 CYCLES!

Multi-cycle Approach

All operations in each clock cycle T_i are done in parallel not sequential!

For example, T_1, IR = Memory[PC] and PC=PC+4 are done simultaneously!

	Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
T_1	Instruction fetch	IR = Memory[PC] PC = PC + 4			
T_2	Instruction decode/register fetch	A = Reg [IR[25-21]] B = Reg [IR[20-16]] ALUOut = PC + (sign-extend (IR[15-0]) << 2)			
T_3	Execution, address computation, branch/ jump completion	ALUOut = A op B	ALUOut = A + sign-extend (IR[15-0])	if (A ==B) then PC = ALUOut	PC = PC [31-28] II (IR[25-0]<<2)
T_4	Memory access or R-type completion	Reg [IR[15-11]] = ALUOut	Load: MDR = Memory[ALUOut] or Store: Memory [ALUOut] = B		
T_5	Memory read completion		Load: Reg[IR[20-16]] = MDR		

Between Clock T_2 and T_3 the microcode sequencer will do a dispatch 1