Computer Engineering Department

CC 311- Computer Architecture

The Processor:

Multicycle

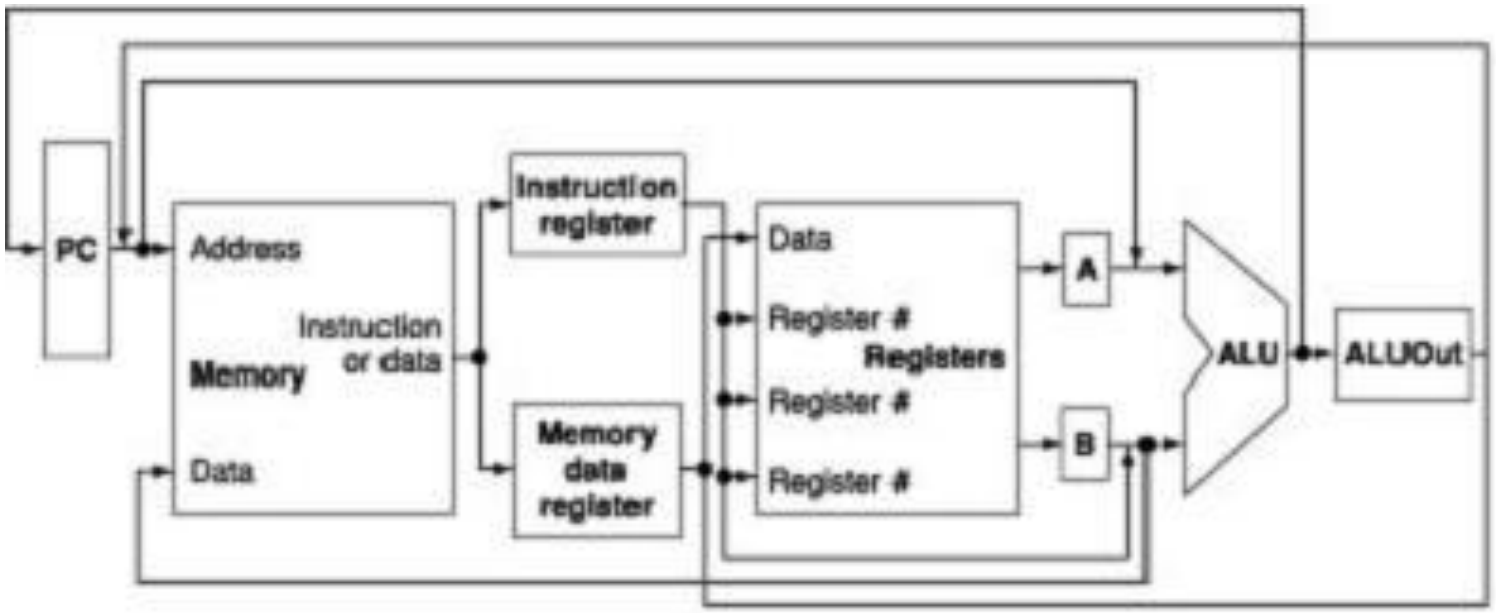
Datapath and Control

Single Cycle Problems

- Single cycle instruction is not used in modern computers
- **Disadvantages:**
  - Inefficient in both performance and HW cost
  - Clock cycle must have the same length for every instruction
  - Clock cycle determined by the longest possible path
  - For complicated instruction like floating point, it will be harder
  - Functional units must be duplicated, since each can be used only once per cycle

Single-cycle to Multi-cycle

– Use a “smaller” cycle time
– Instructions can take different numbers of cycles
– “Multi-cycle” datapath



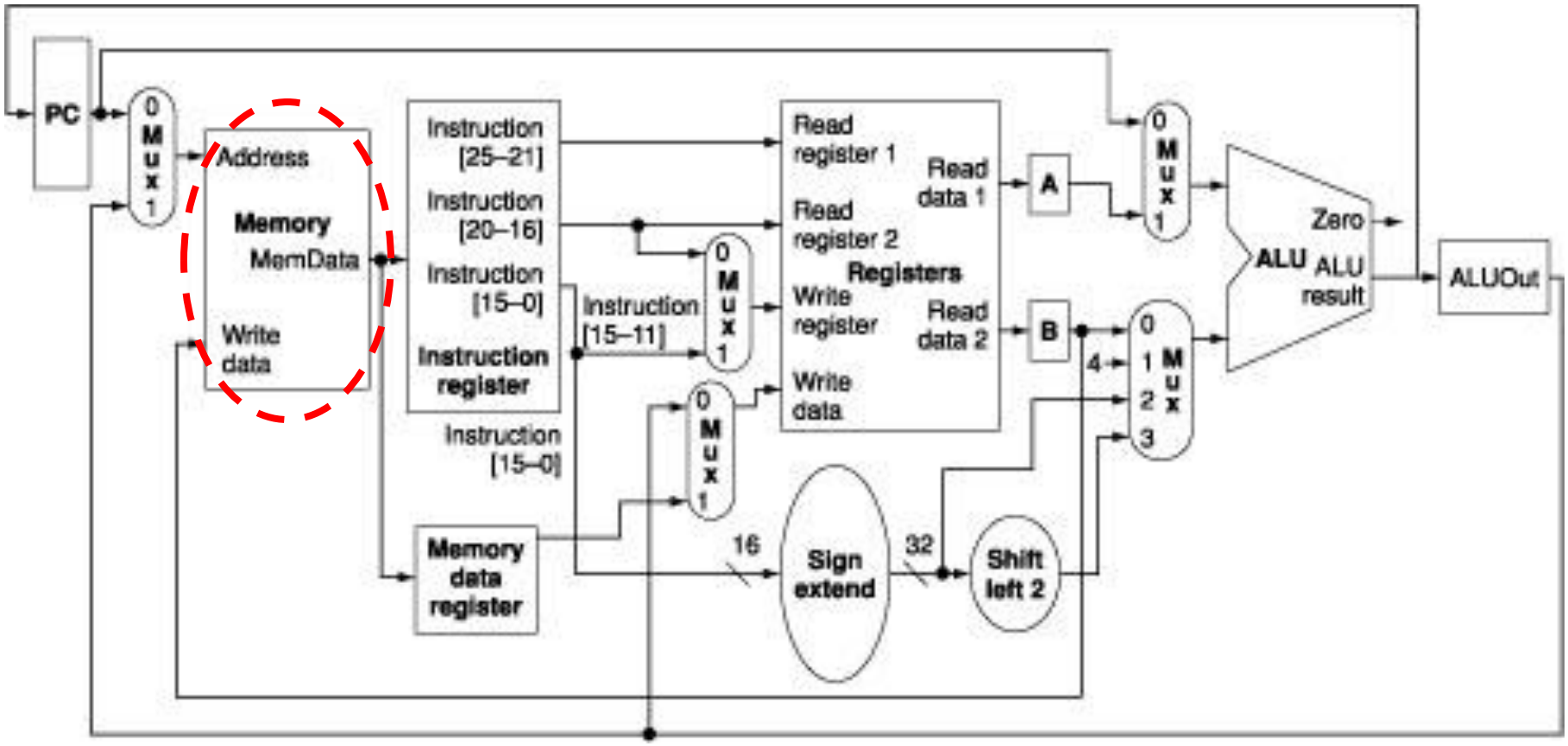
Multicycle Implementation

- Each instruction is broken into a series of steps
- Each step executes in one clock cycle
- Advantages:
  - Allows functional units to be used more than once
  - Help reduce HW cost

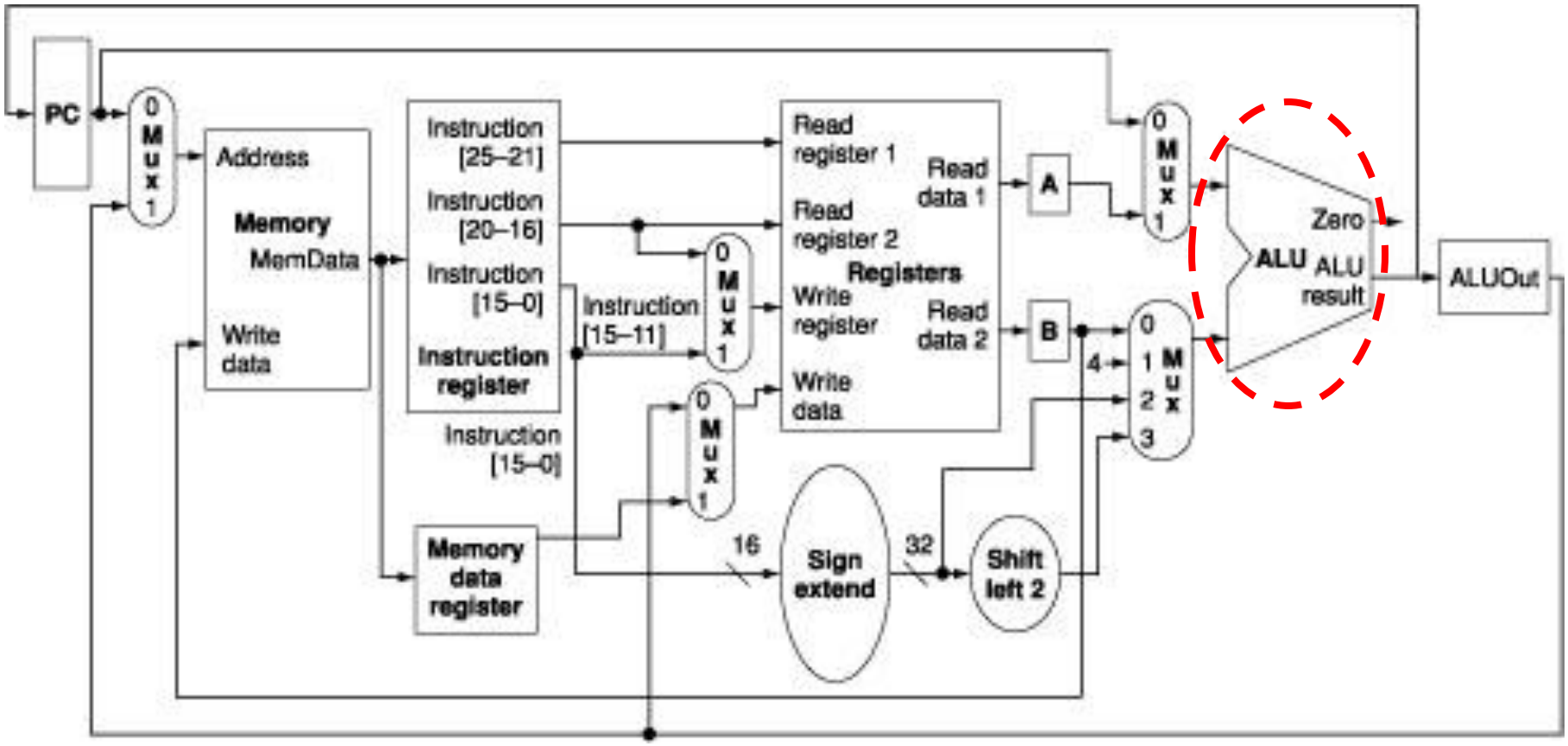
Multicycle Implementation

- Major difference from single cycle datapath:
  - Single memory unit used for both instruction and data
  - Single ALU instead of three
  - One or more registers are added after every major functional unit to hold the output of that unit until the value is used in next clock cycle
- Added registers
  - **Instruction register (IR)**
    - Saves output from memory for instruction read
  - **Memory data register (MDR)**
    - Saves output from memory for data read
  - **A and B registers**
    - Hold the register operands read from register file
  - **ALUOut register**
    - Holds the output from ALU

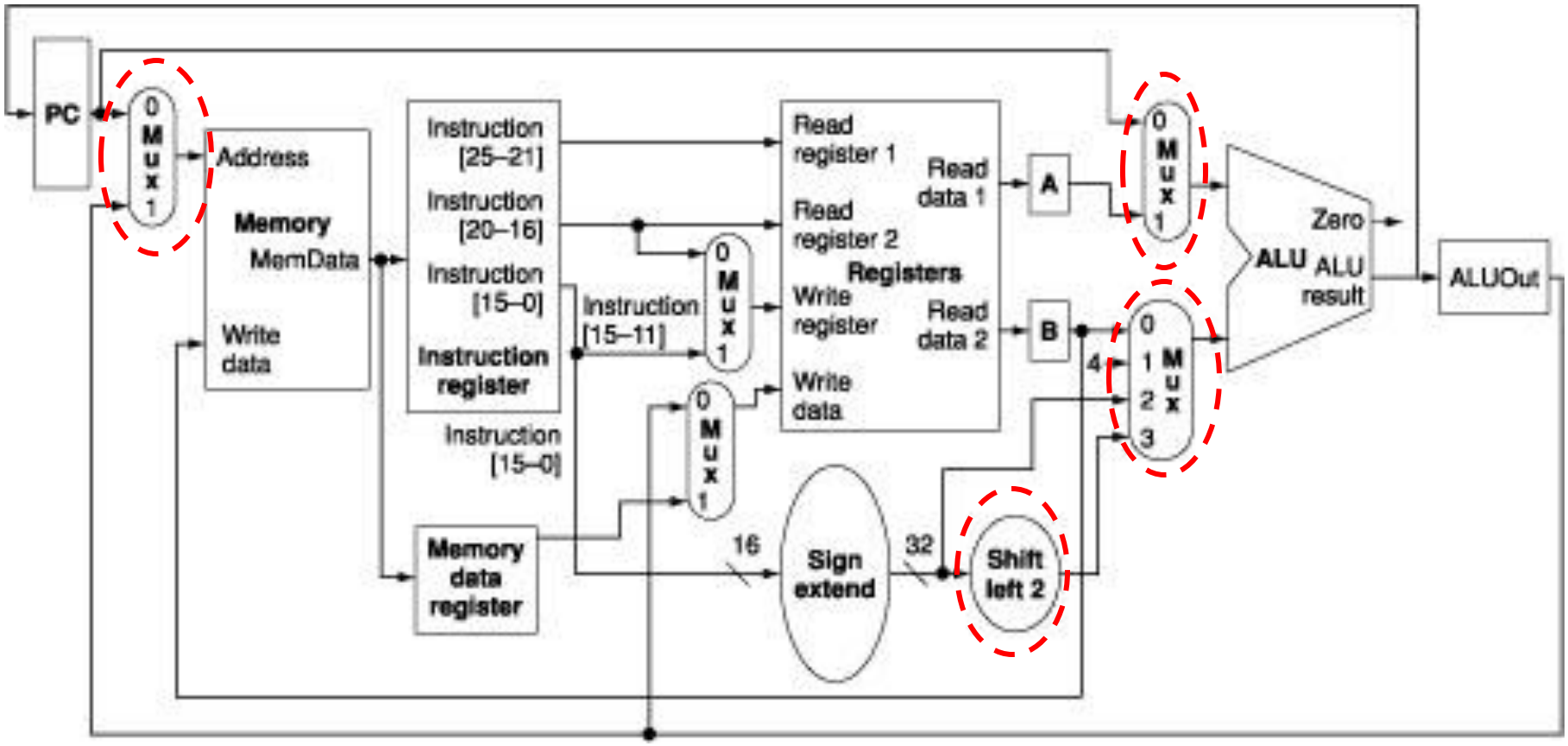
Multicycle Implementation



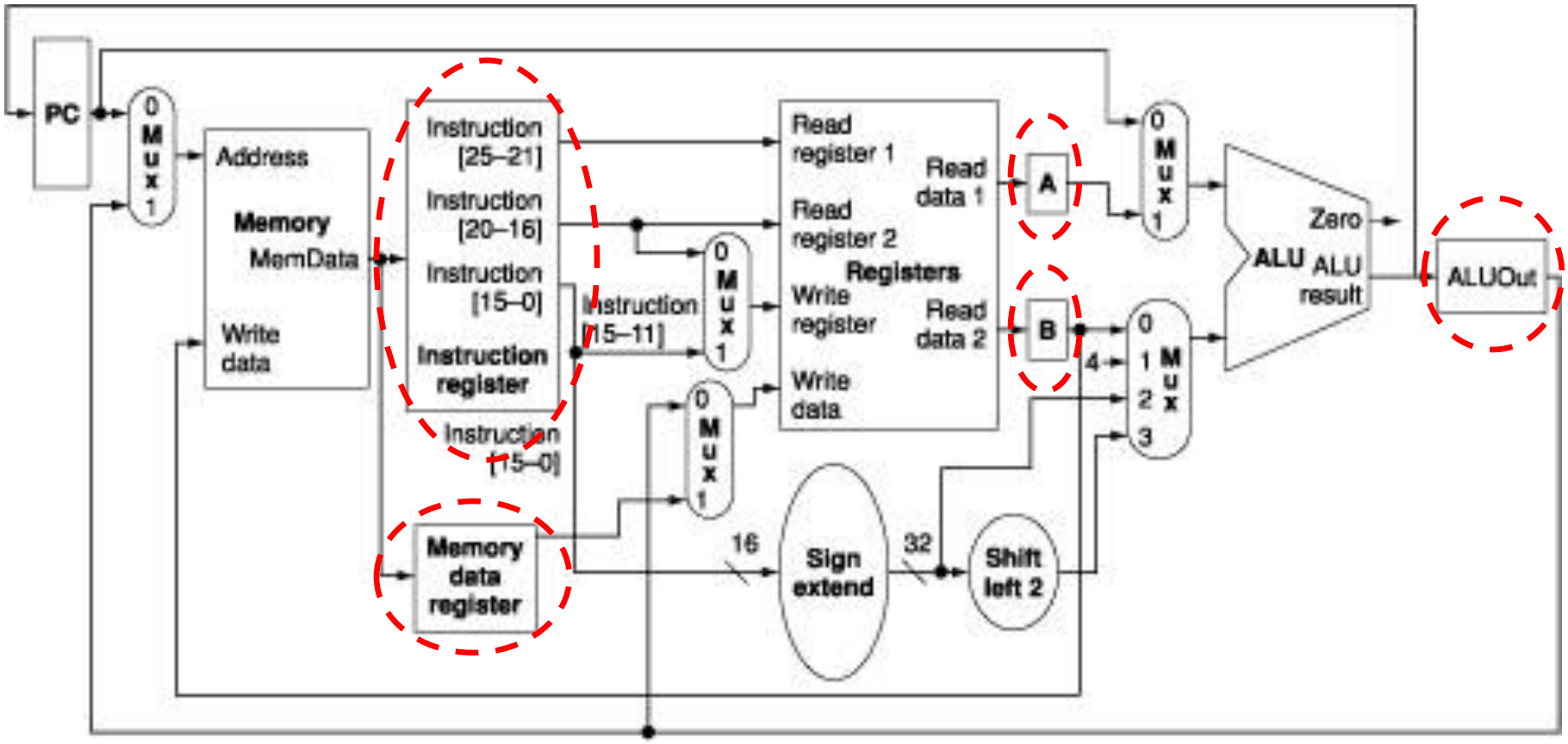
Multicycle Implementation



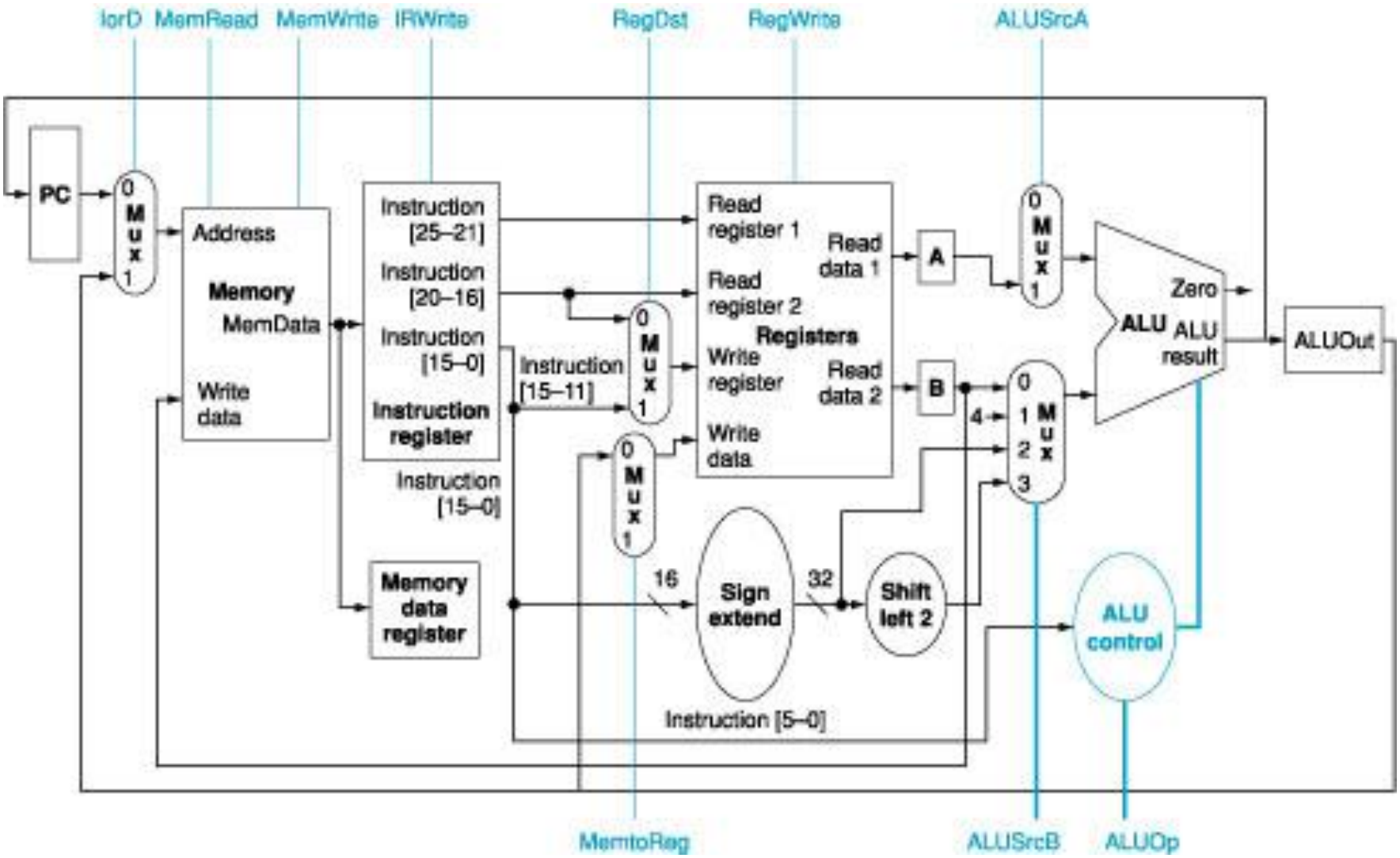
Multicycle Implementation



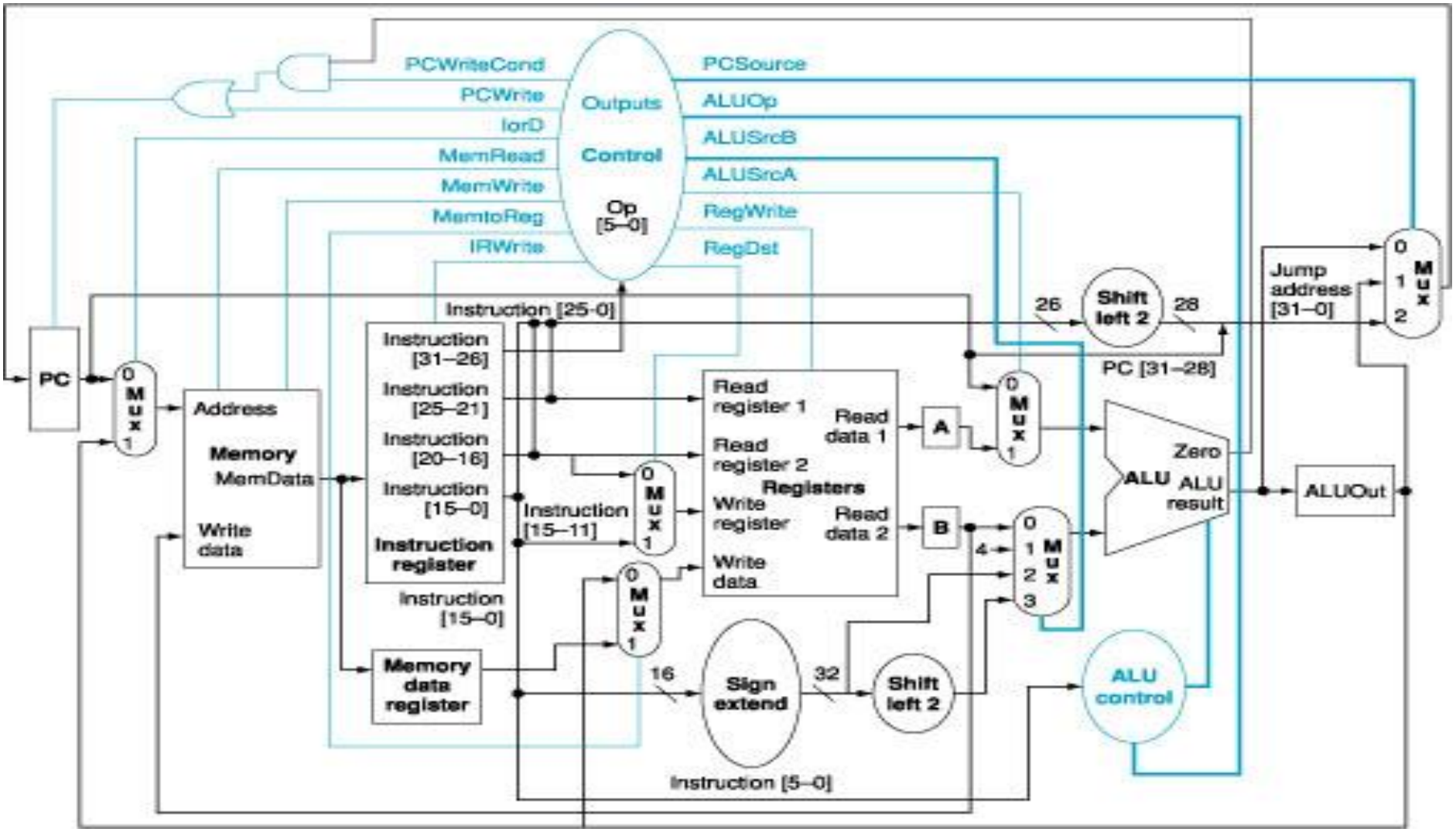
Multicycle Implementation



Multicycle Implementation (322)



Multicycle Implementation (323)



Breaking Instructions

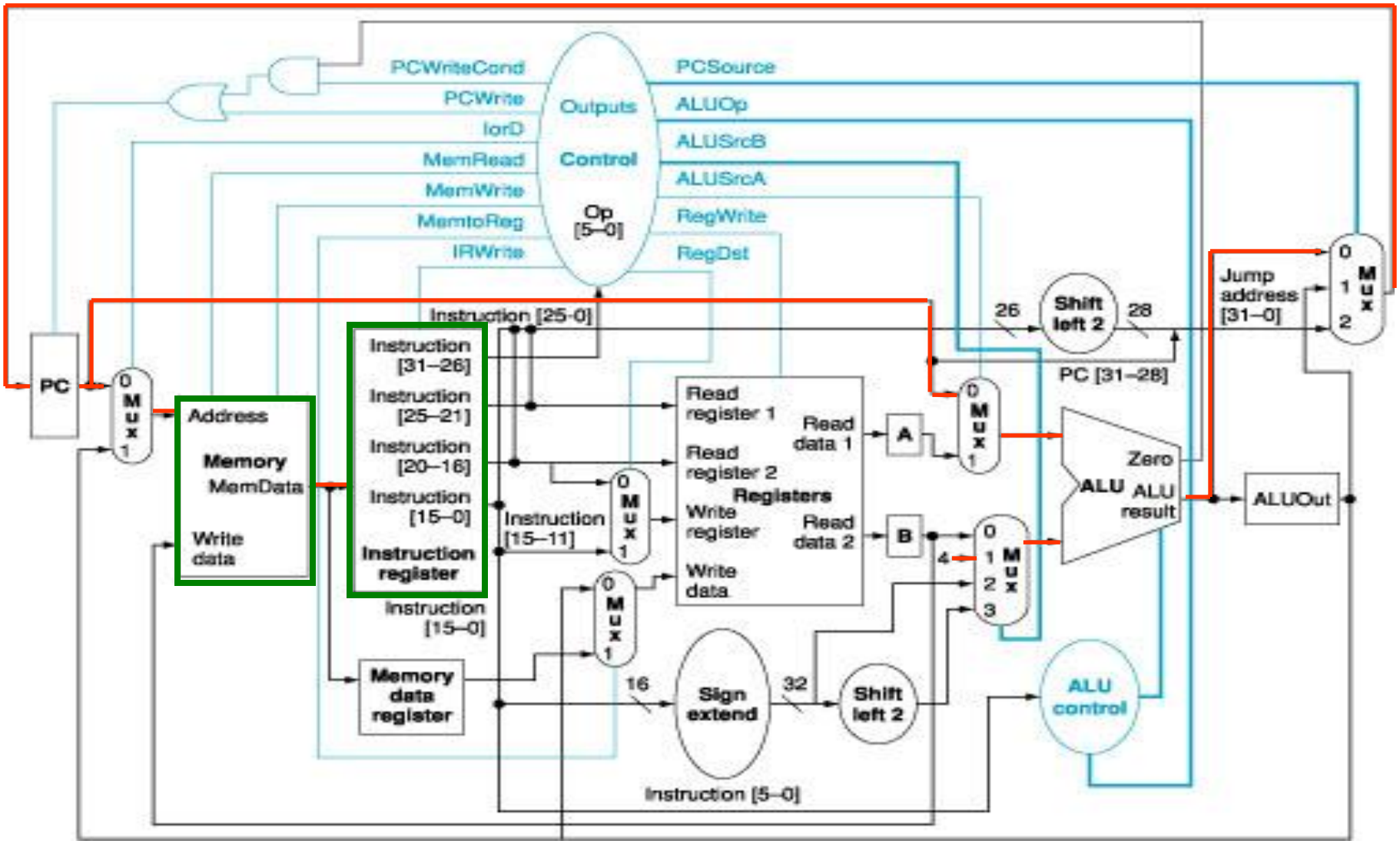
- Each step will contain at most
  - One ALU operation
  - One register file access
  - One memory access
- At the end of each cycle, data values needed for next cycle must be stored into a register
- All operations listed in one step occur in parallel
- Successive steps occur in a series

Breaking Instructions

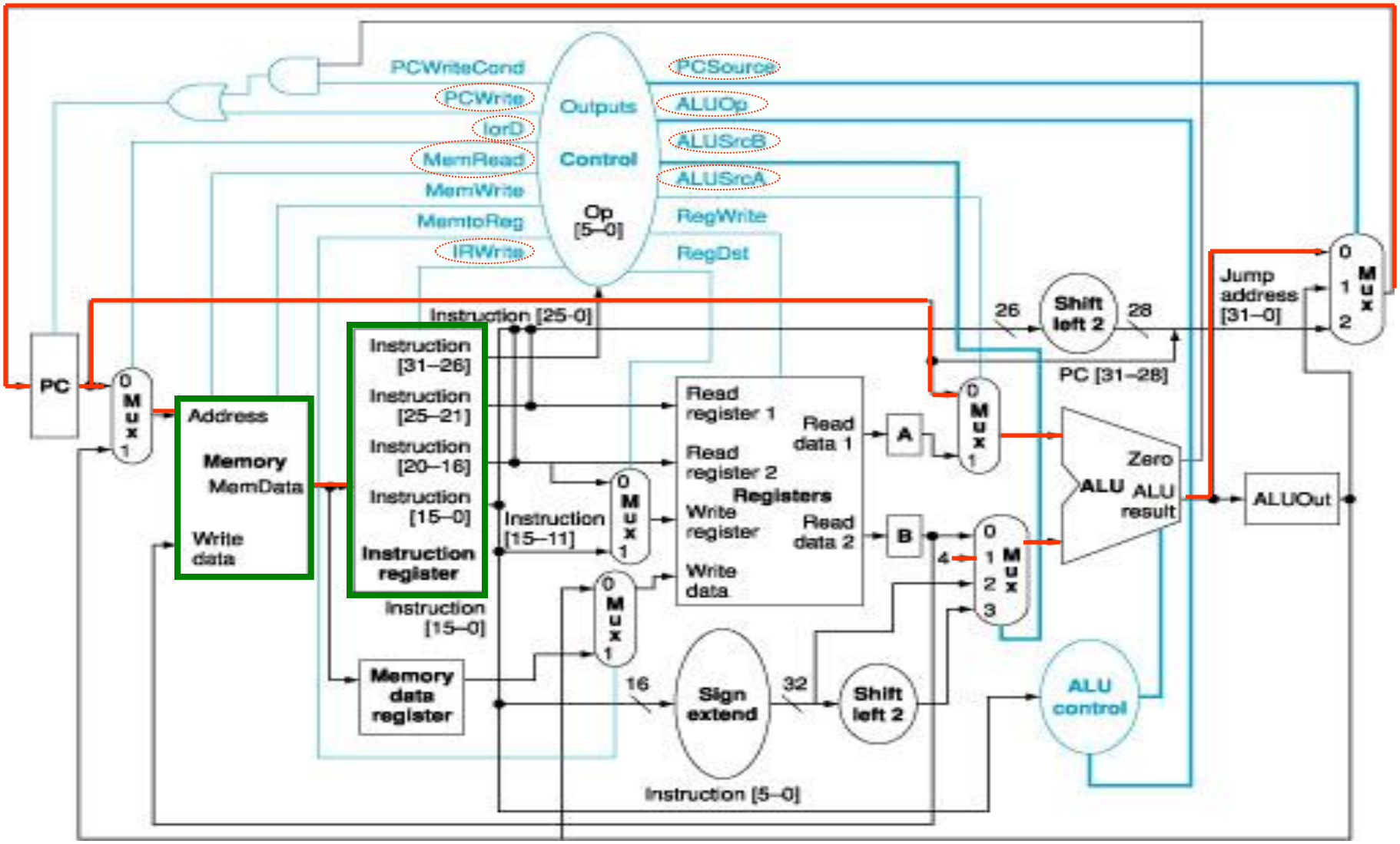
- Steps:
  1. Instruction fetch
  2. Instruction decode & register fetch
  3. Execution, memory address computation, or branch completion
  4. Memory access or R-type instruction completion
  5. Memory read completion

See Fig 5.30, p. 329

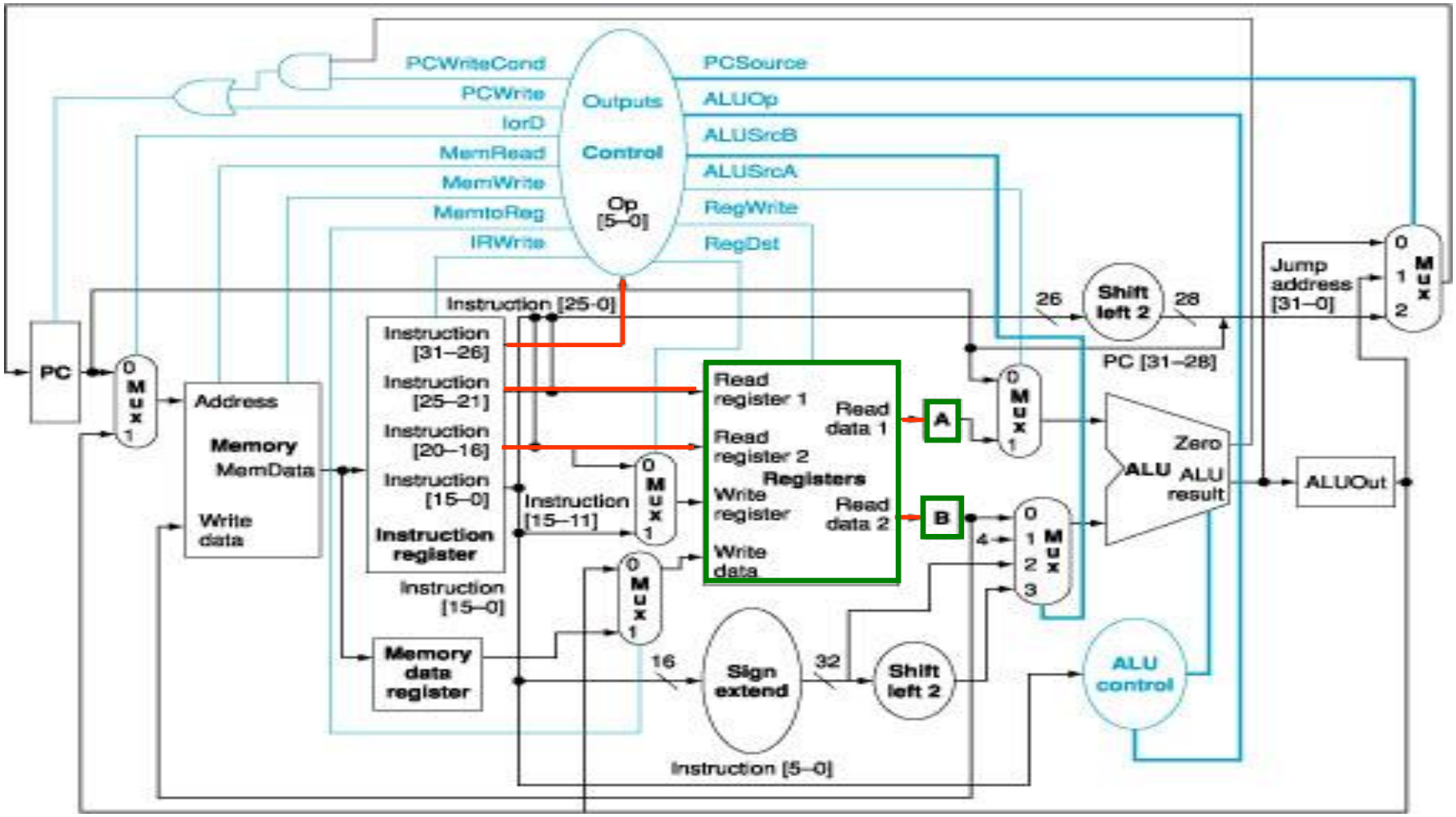
Instruction fetch (R-Type)



Instruction fetch (R-Type)

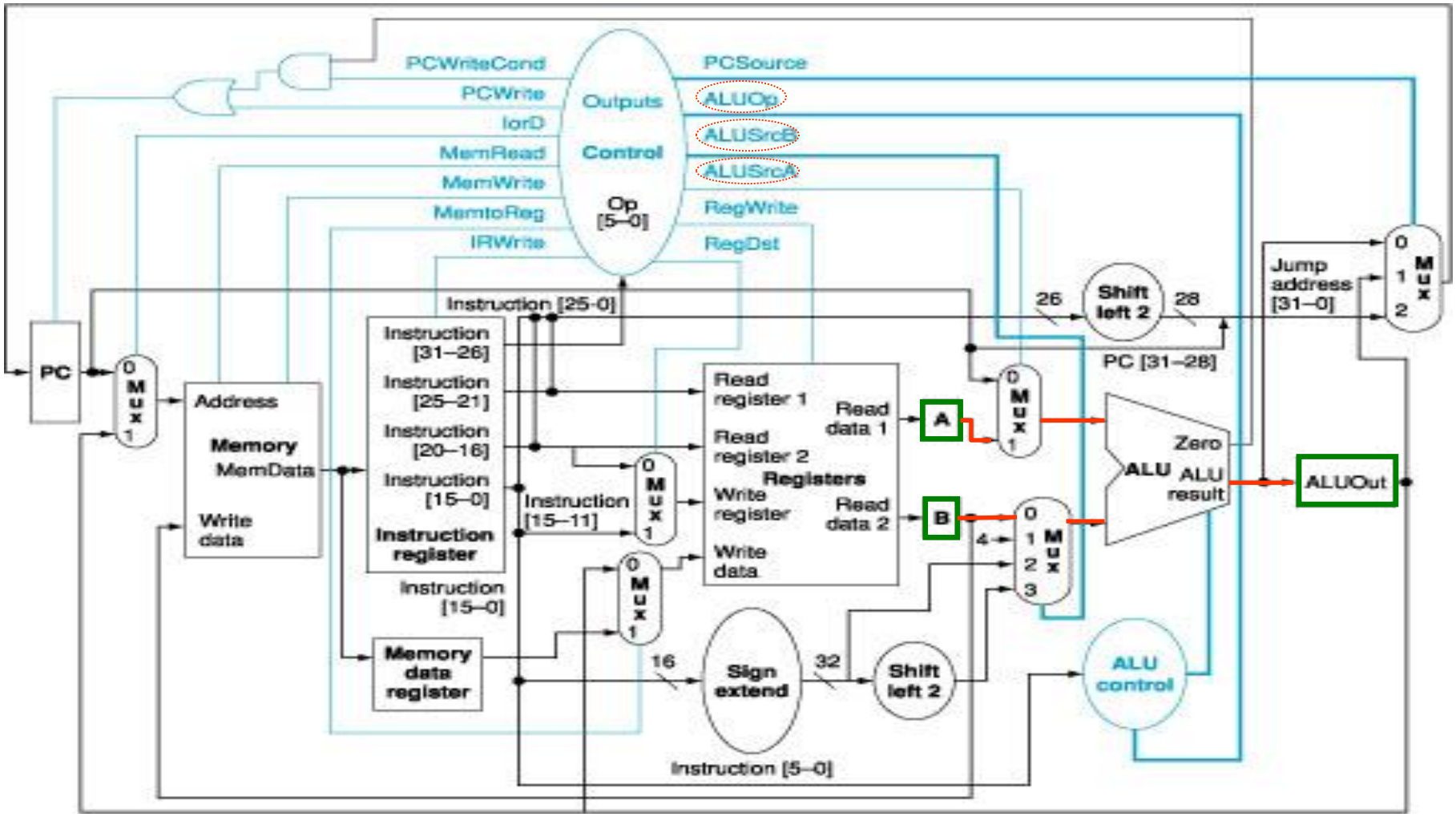


Instruction decode & register fetch



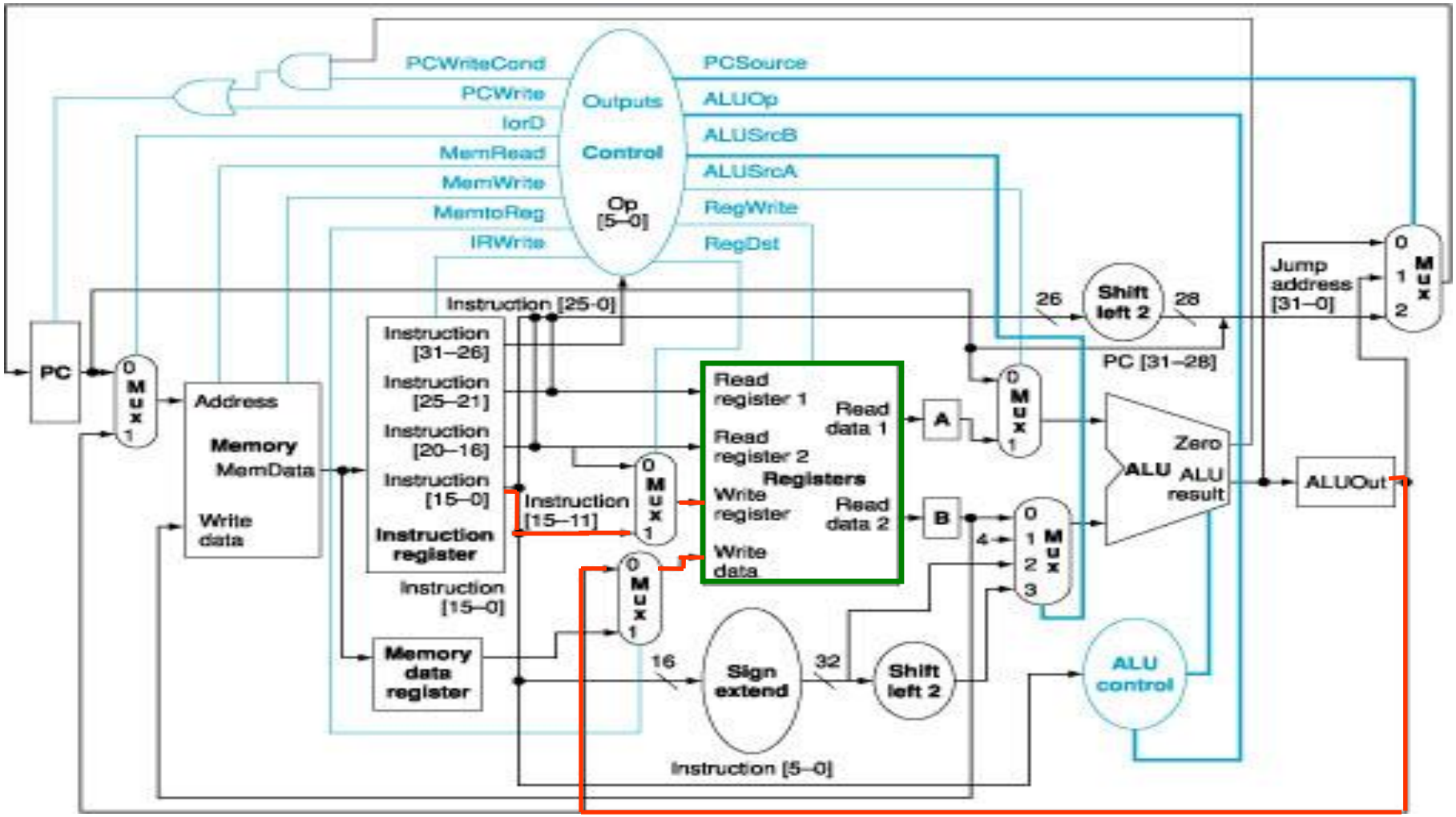
R-Type

Execution



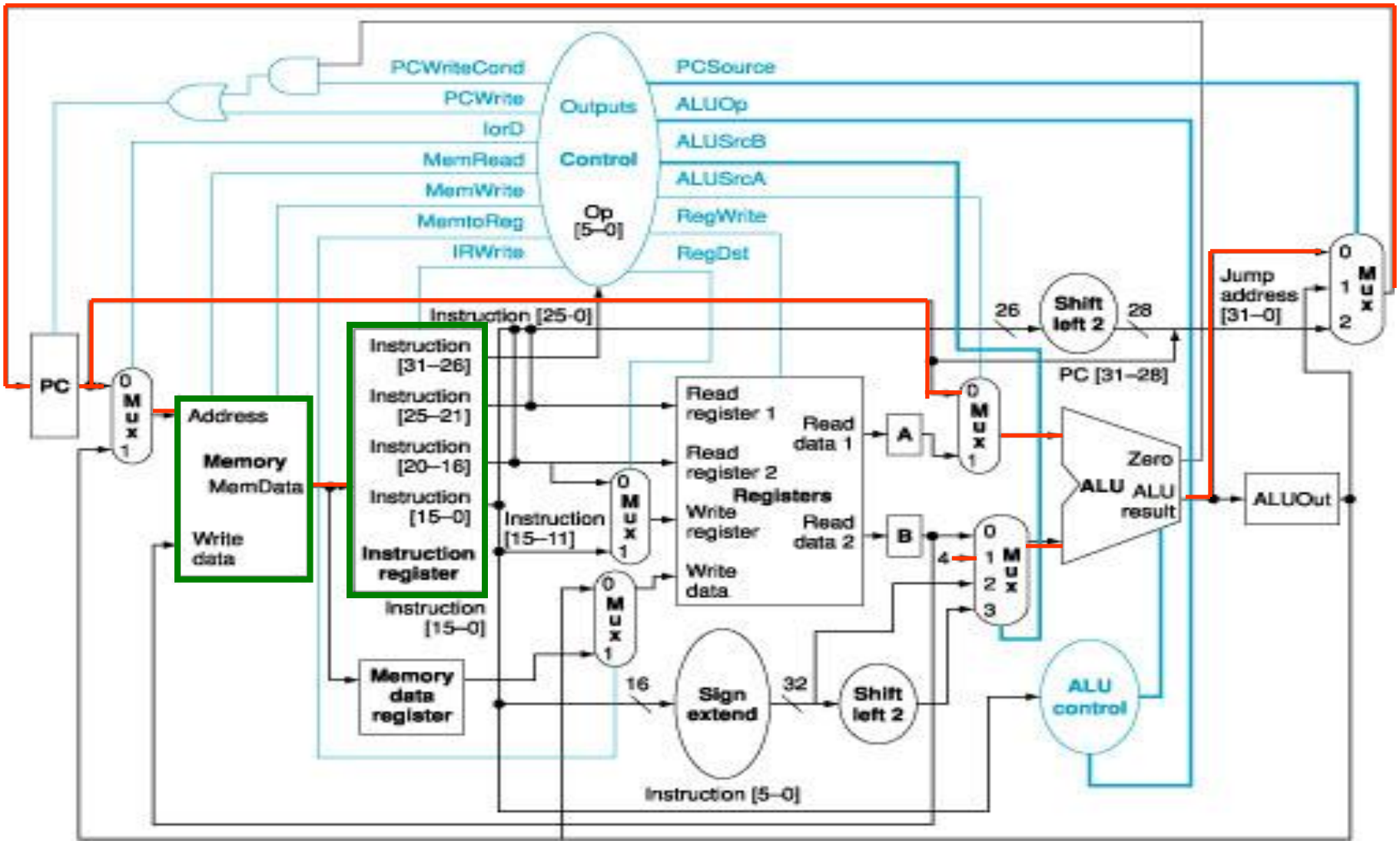
R-Type

Write Result in Register File

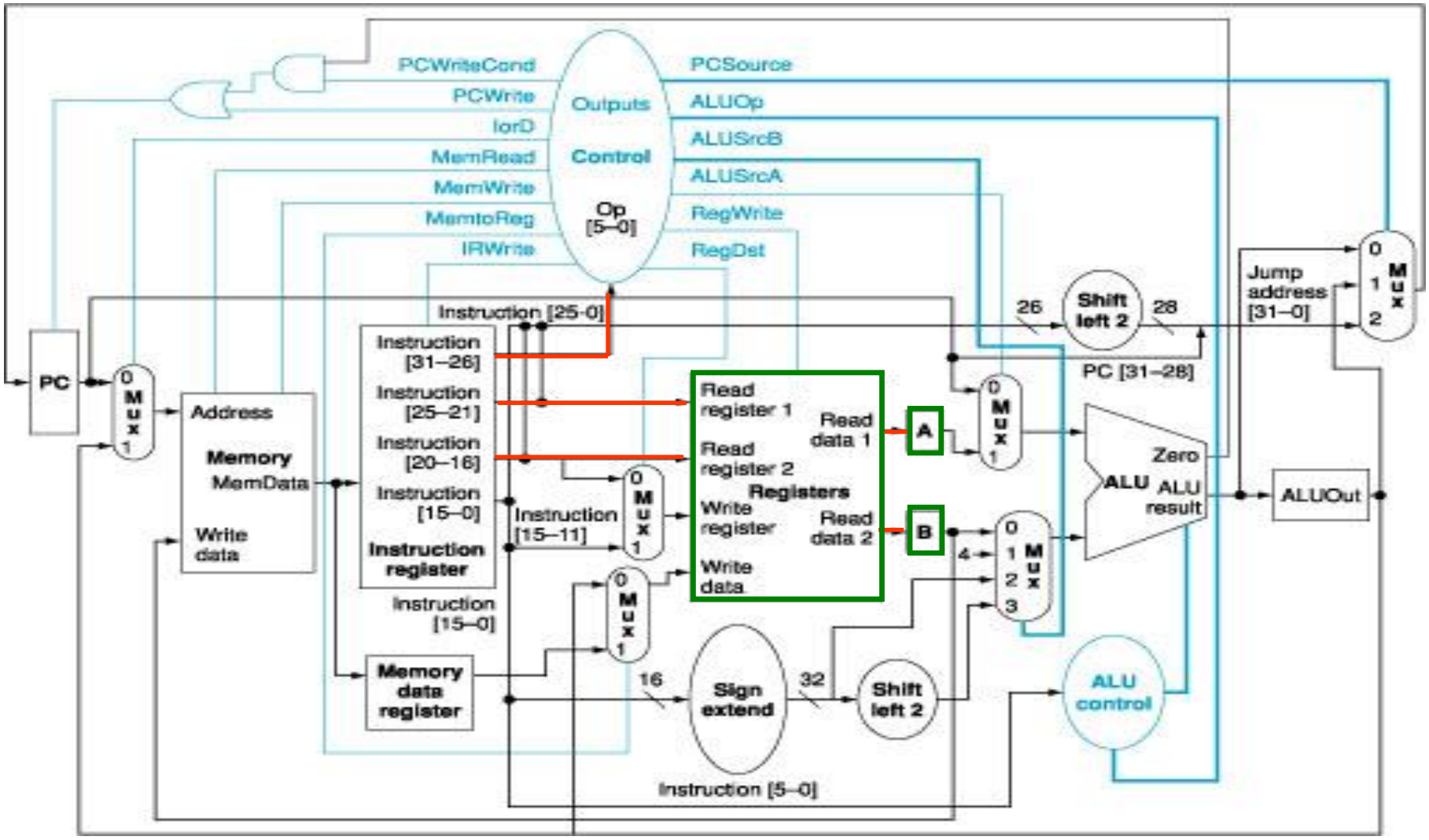


R-Type

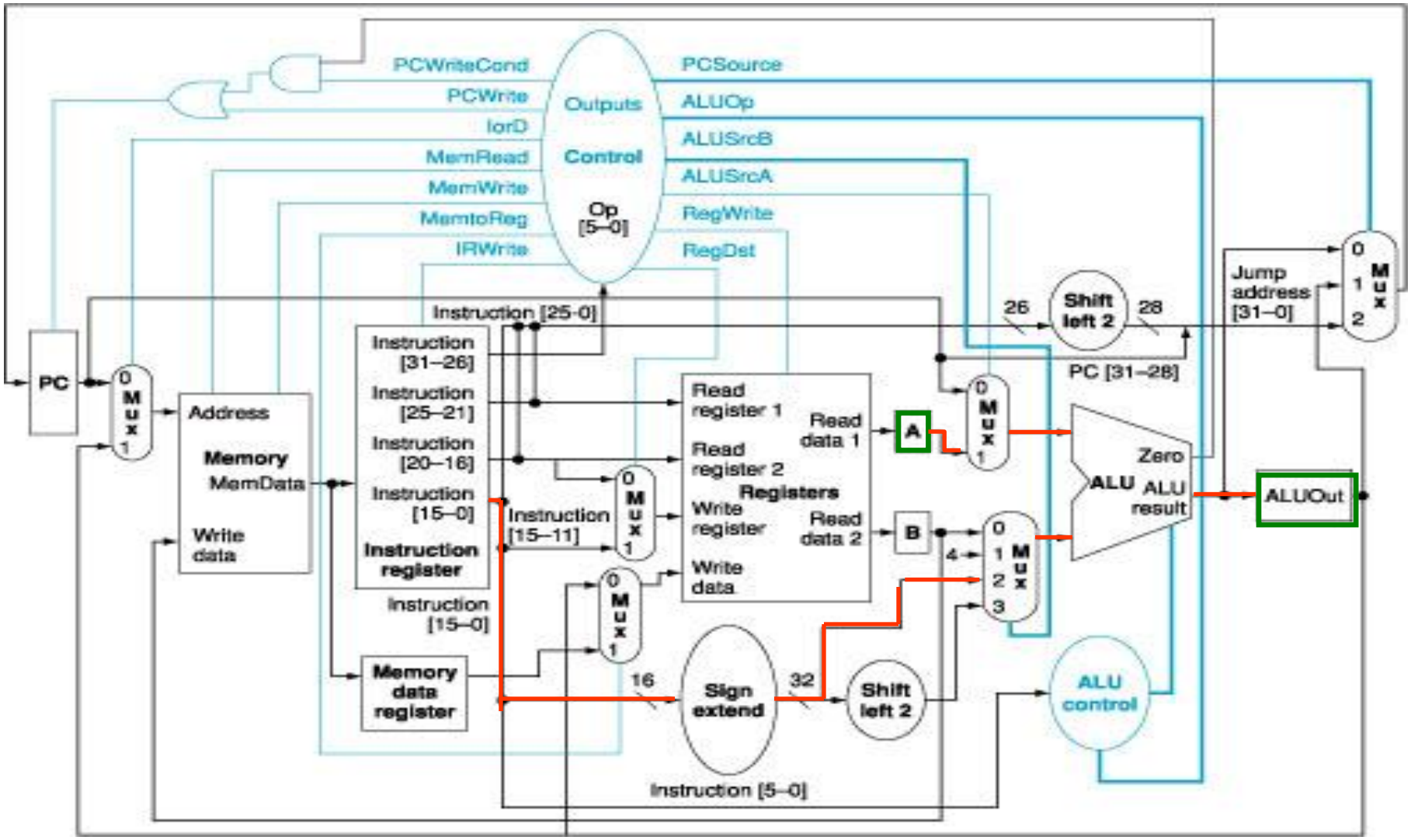
Instruction fetch (Sw)



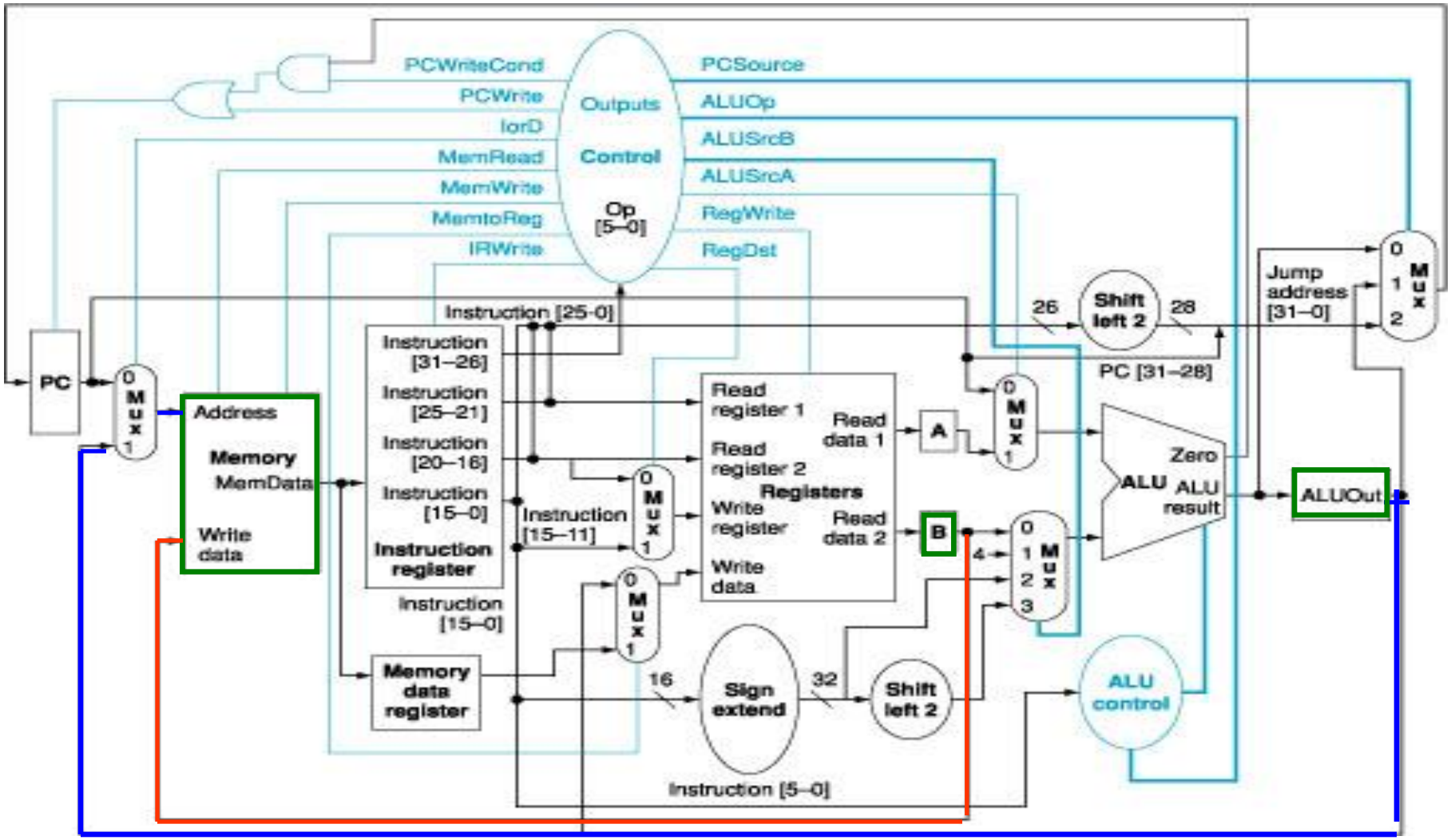
Instruction decode & register fetch (Sw)



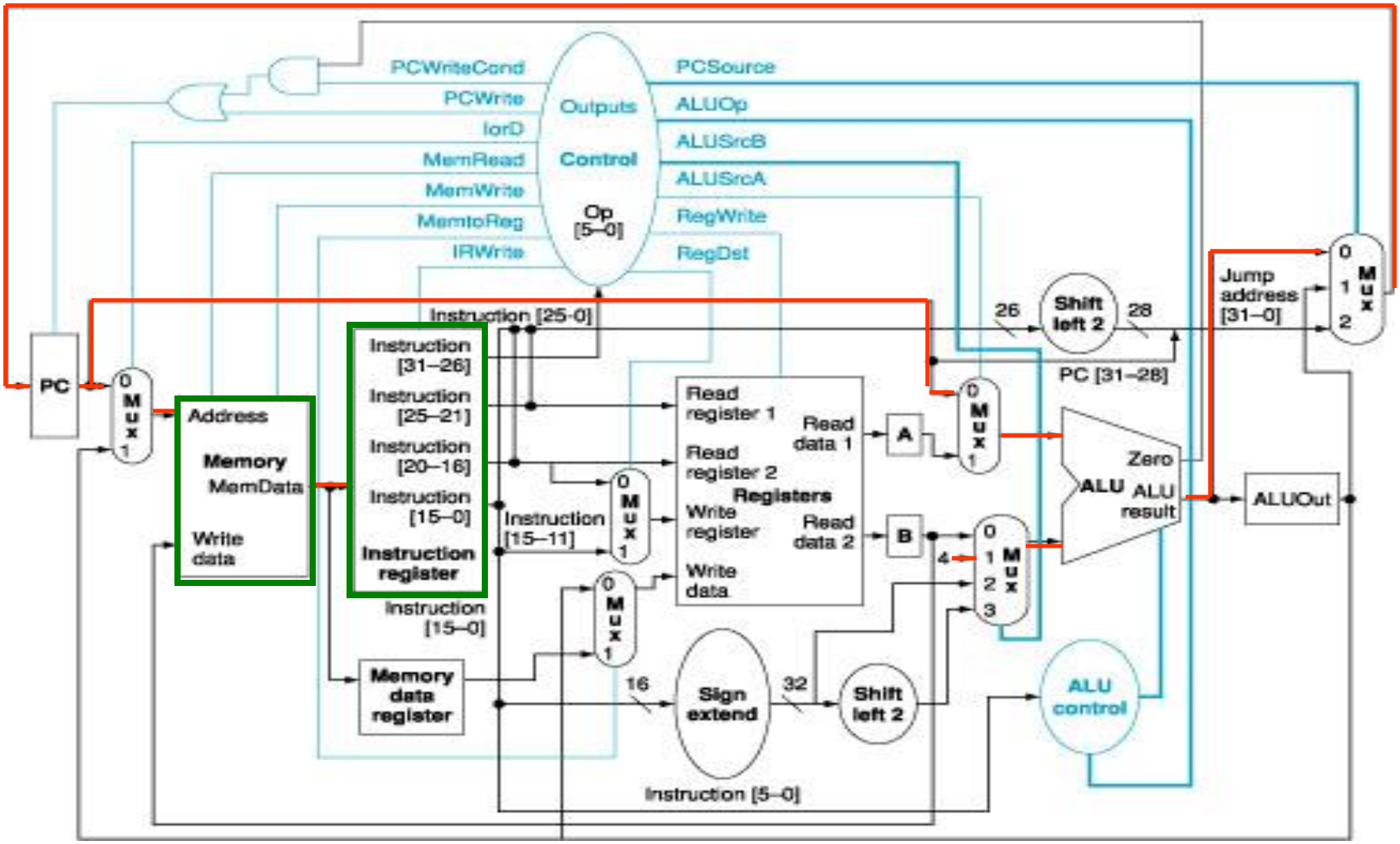
Memory address computation



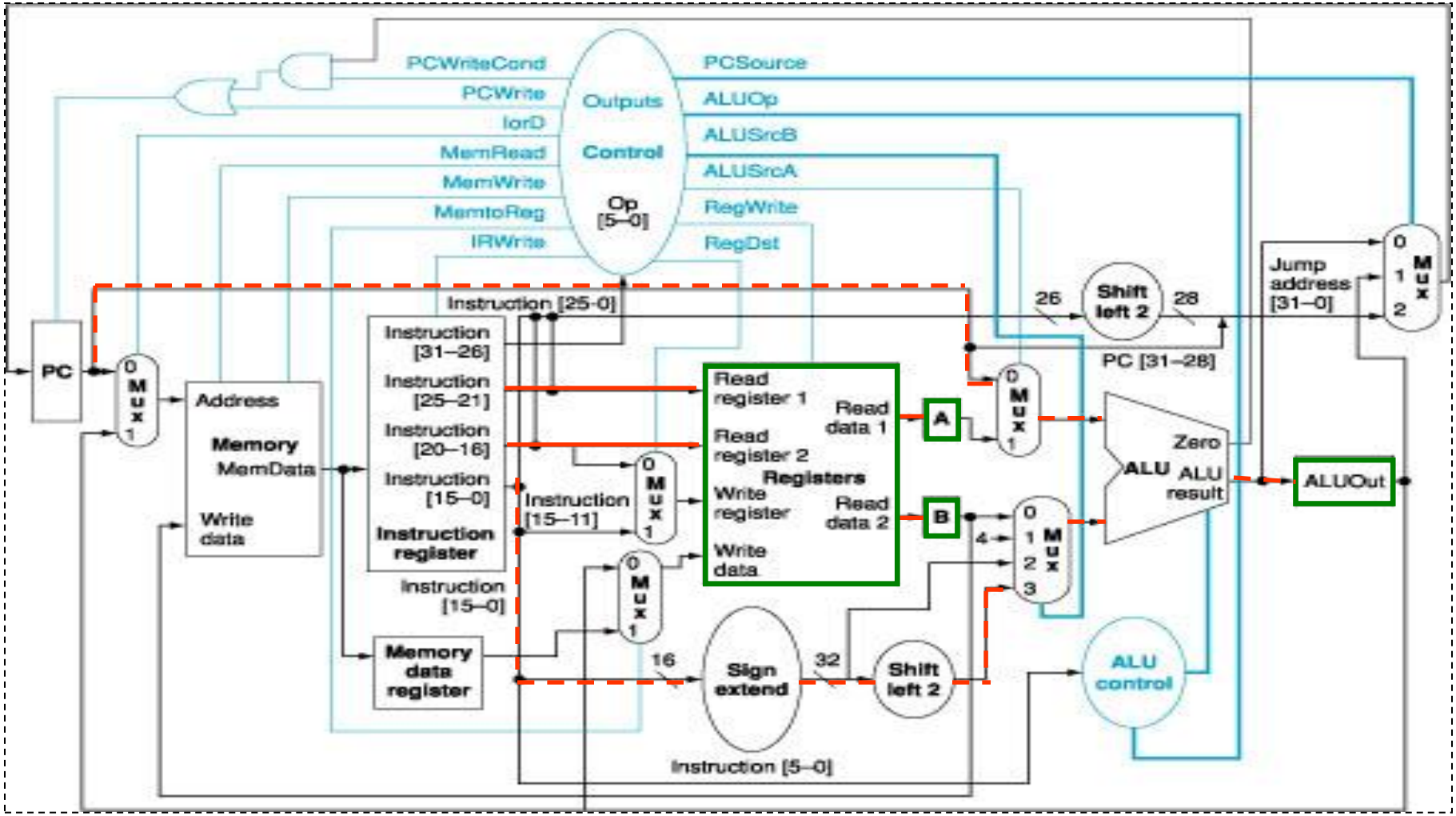
Memory Write completion



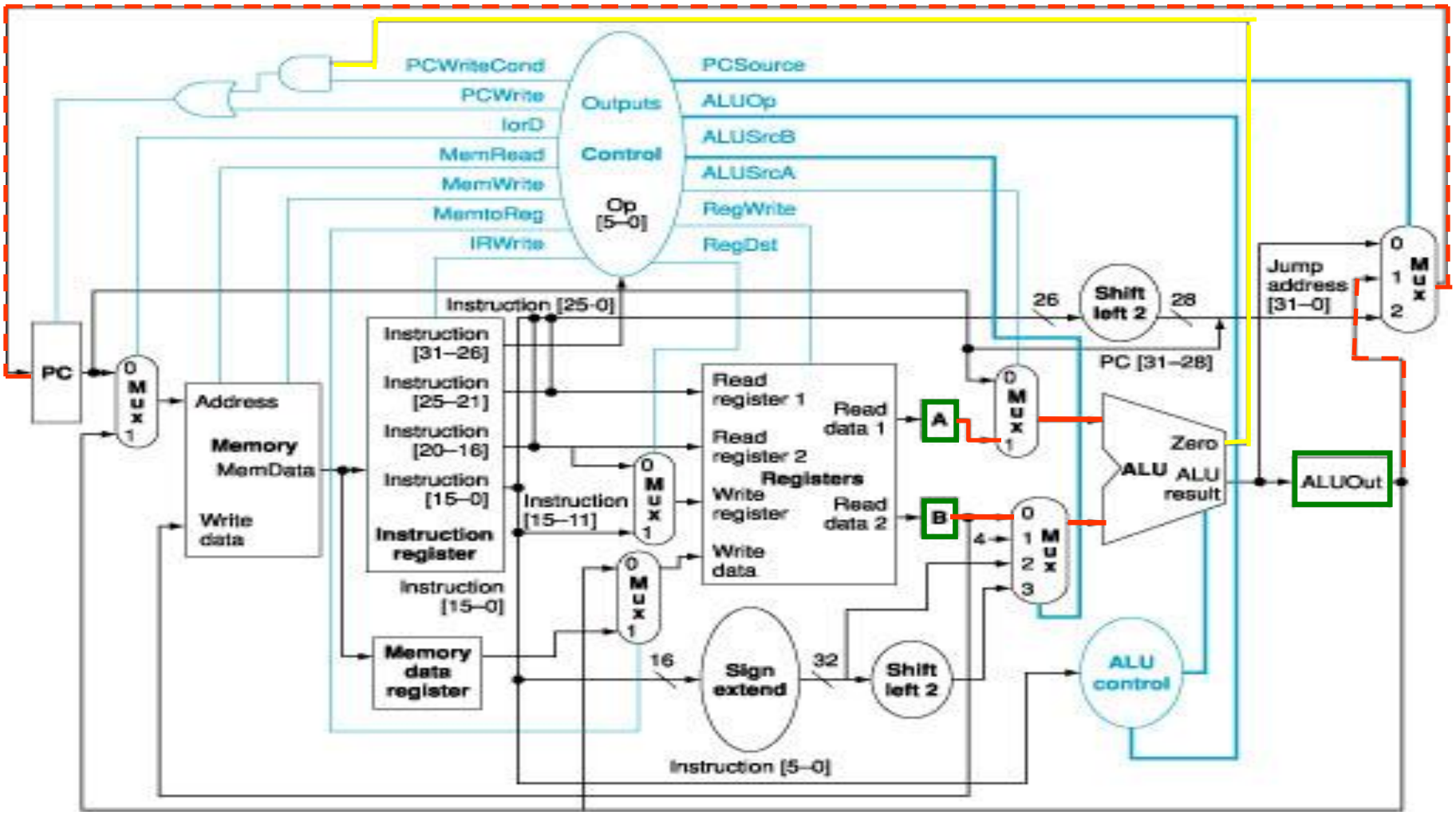
Instruction fetch (Beq)



Instruction decode & register fetch And - Branch address computation



Branch condition computation

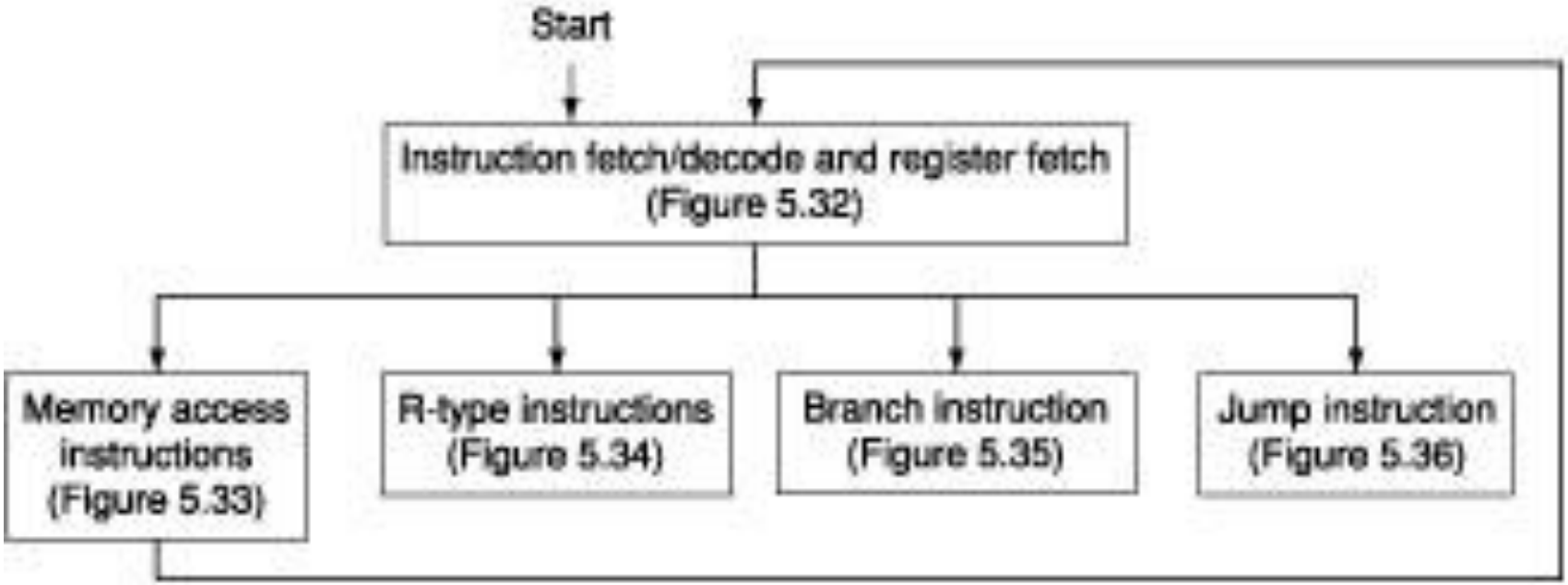


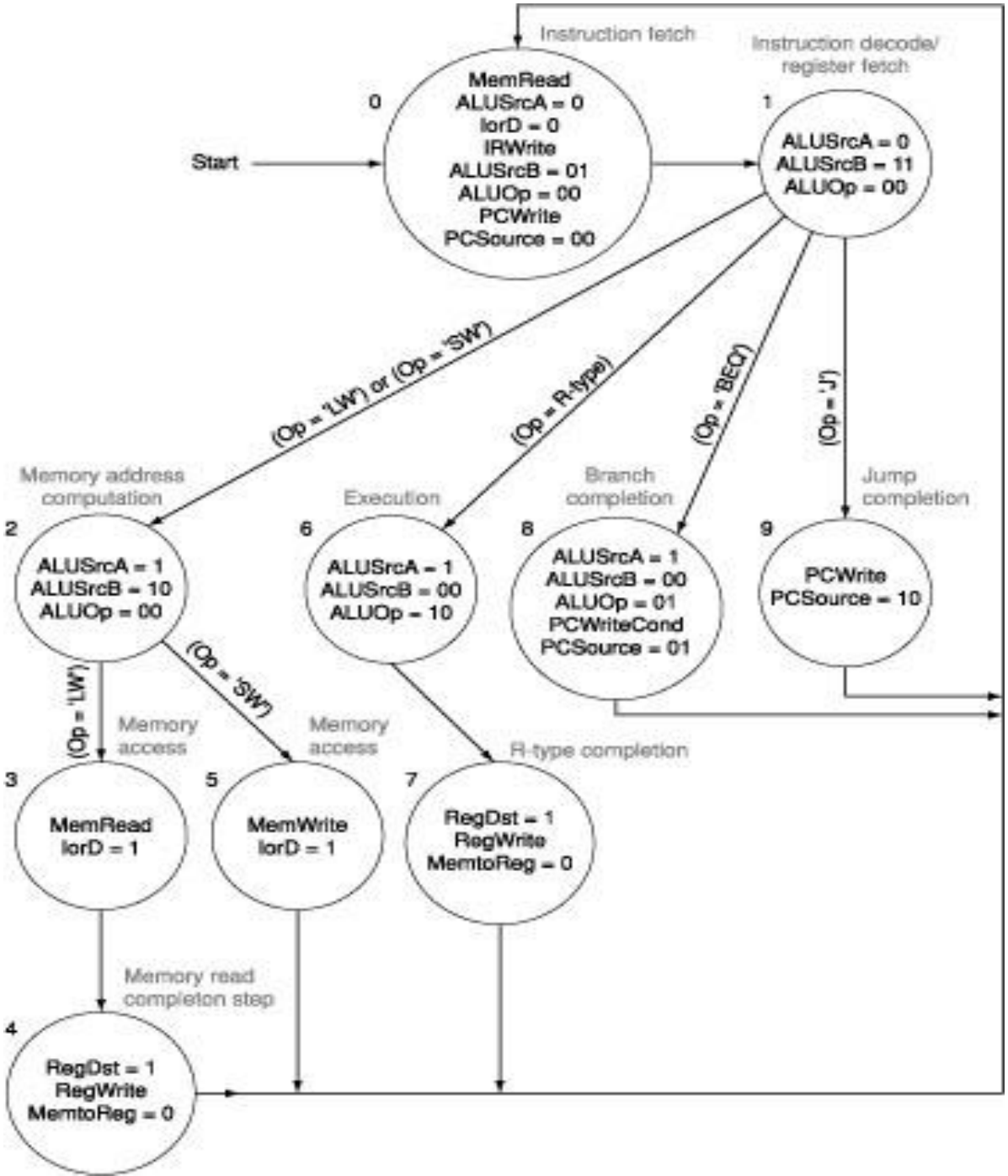
Finite State Machine

- Can be used to specify multi-cycle control
- A sequential logic function consisting of
  - A set of inputs
  - A set of outputs
  - A next state function that maps the current state and the input to a new state
  - An output function that maps the current state and the inputs to a set of asserted outputs

Finite State Machine

- The high-level view of the finite state machine control




Instruction fetch
Instruction decode/
register fetch
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Start
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
Memory address
computation
Execution
Branch
completion
Jump
completion
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
6
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
9
PCWrite
PCSource = 10
(Op = 'LW')
(Op = 'SW')
Memory
access
Memory
access
R-type completion
3
MemRead
IorD = 1
5
MemWrite
IorD = 1
7
RegDst = 1
RegWrite
MemtoReg = 0
Memory read
completon step
4
RegDst = 1
RegWrite
MemtoReg = 0

Step	R-type	Mem Ref	Branch	Jump
Instr fetch	IR = Memory[PC]; PC = PC + 4;			
Decode	A = Reg[IR[25-21]]; B = Reg[IR[20-16]]; ALUOut = PC +(sign-extend(IR[15-0])<< 2);			
Execute	ALUOut = A op B;	ALUOut = A + sign-extend (IR[15-0]);	if (A==B) PC = ALUOut;	PC = PC[31-28] ||(IR[25-0] << 2);
Memory access	Reg[IR[15-11]] = ALUOut;	MDR = Memory[ALUOut]; or Memory[ALUOut] = B;		
Write-back		Reg[IR[20-16]] = MDR;		

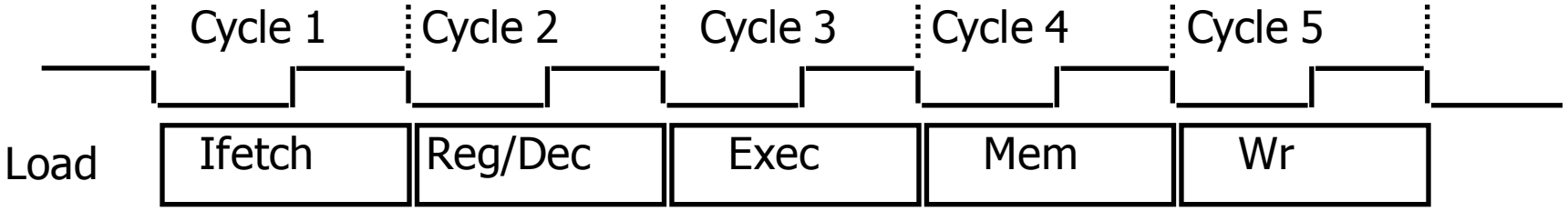
RTL : Register Transfer Language

- R-Type
  - T_0 : IR ← M[PC], PC ← PC+4
  - T_1 : A ← Reg[IR[25-21]], B ← Reg[IR[20-16]]
  - T_2 : ALUout ← A op B
  - T_3 : Reg[IR[15-11]] ← ALUout
- LW
- SW
- Beq
- J

Actions of 1- Bit Control Signals

Signal name	Effect when deasserted	Effect when asserted
RegSDt	The register file destination number for the Write register comes from the rt field	The register file destination number for the Write register comes from the rd file
RegWrite	None	The general purpose register selected by the Write register number is written with the value of the Write data input
ALUSrcA	The first ALU operand is the PC	The first ALU operand comes from the A register
MemRead	None	Content of memory at location specified by the Address input is replaced by value on Write data input
MemWrite	None	Memory contents at the location specified by the Address input is replaced by value on Write data input
MemToReg	The value fed to the register file Write data input comes from ALUOut	The value fed to the register file Write data input comes from MDR
IorD	The PC is used to supply the address to the memory unit	ALUOut is used to supply the address to the memory unit
IRWrite	None	The output of the memory is written into IR
PCWrite	None	The PC is written; the source is controlled by PCSource
PCWriteCond	None	The PC is written if the Zero output from the ALU is also active

Multicycle Disadvantages



Multicycle To . . . . .

Next Design