

1000  
*Multi-Cycle CPU*



ENGR 3410 - Computer Architecture  
Mark L. Chang  
Fall 2006

# Performance of Single-Cycle Machine

CPI = 1.0, but what about cycle time? Unit reuse (i.e. adder for PC vs. ALU)

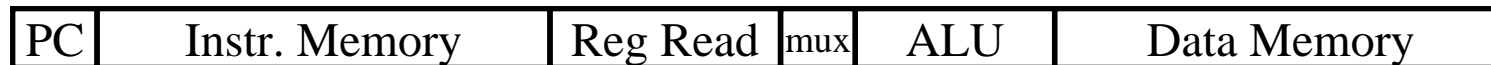
Arithmetic & Logic



Load



Store



Branch

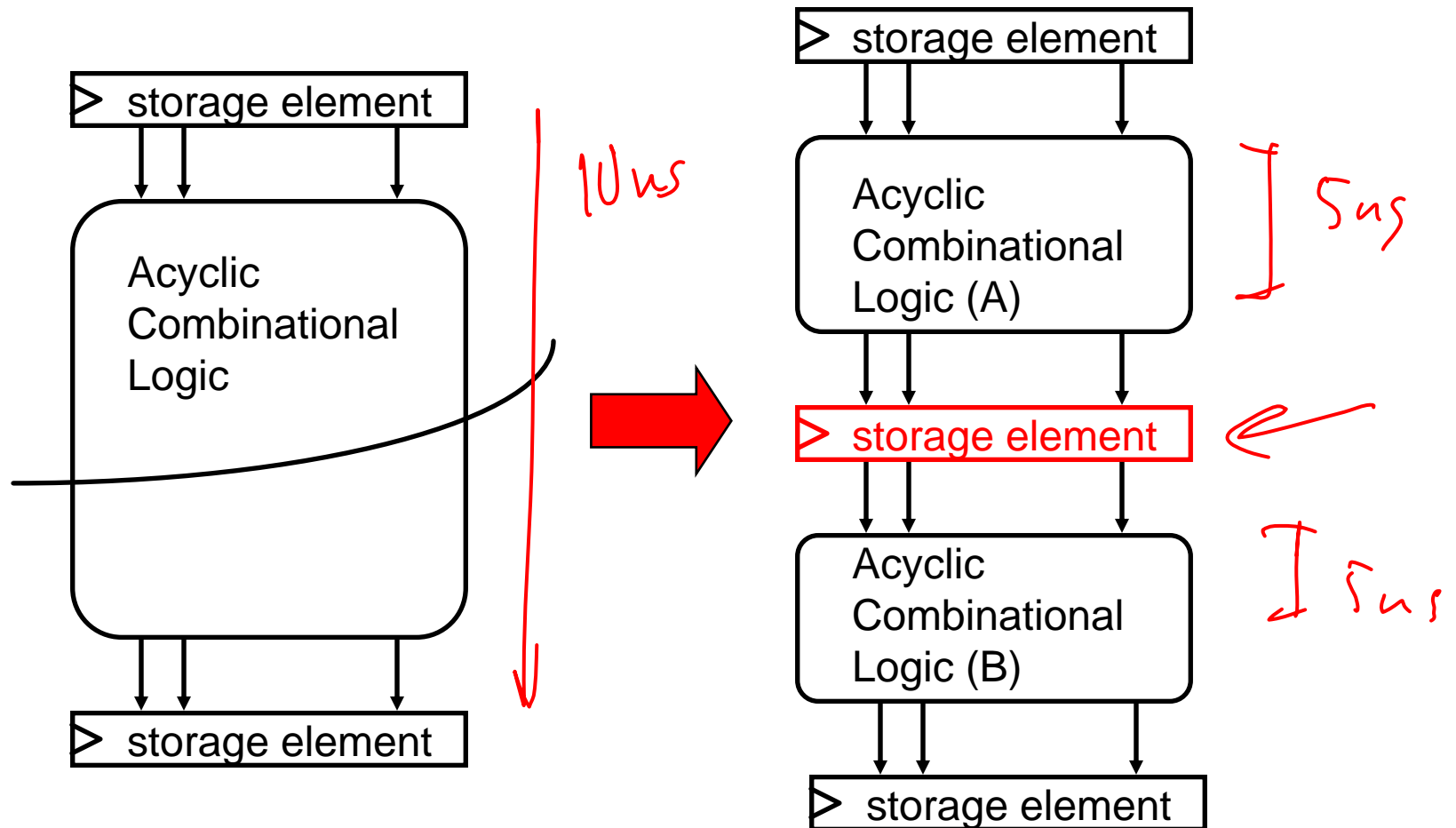


Jump



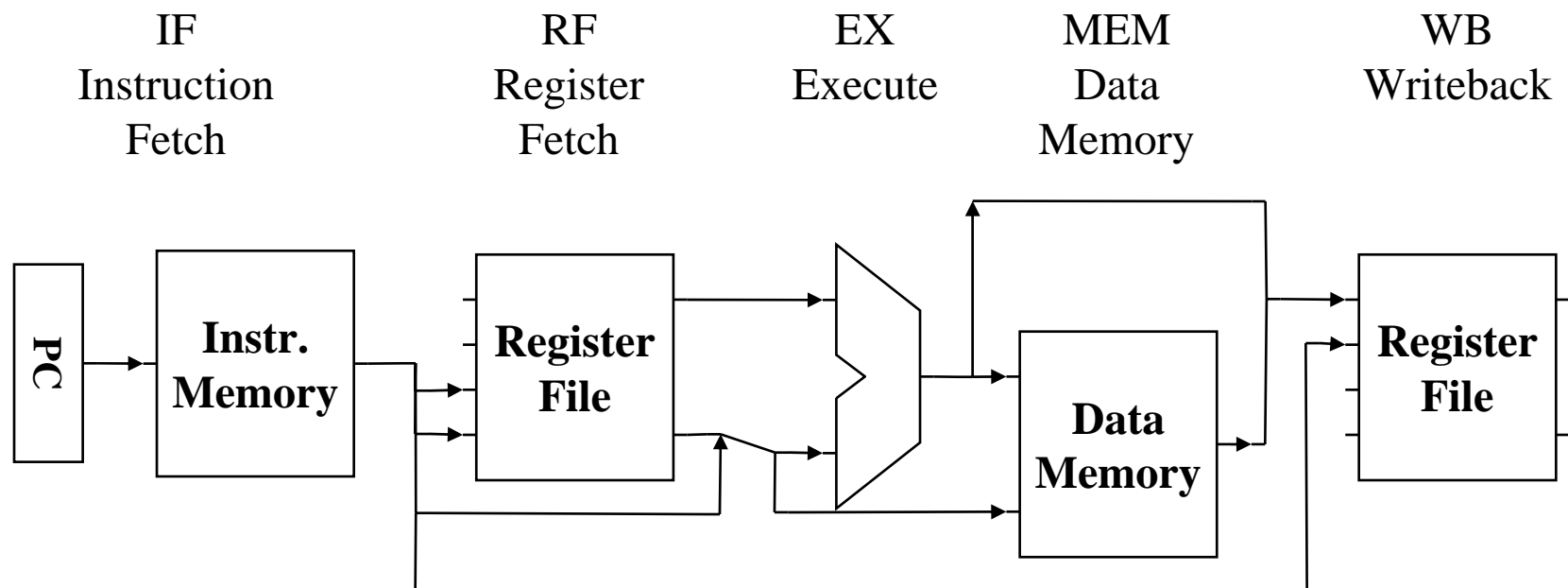
# Reducing Cycle Time

- Cut combinational dependency graph and insert register / latch
- Do same work in two fast cycles, rather than one slow one



# Multicycle Processor Overview

- Divide datapath into multiple cycles



## Multicycle Processor Changes

---

- Only one memory
  - Shared between instructions and data
- Only one ALU/adder
  - Use ALU for instructions & PC computations
- Add registers to datapath
  - IR: instruction register
  - MDR: Memory Data Register
  - A & B: Values read from register file
  - ALUout: Output of ALU

## Cycle 1: Instruction Fetch

---

- Put the instruction to execute into the Instruction Register (IR)

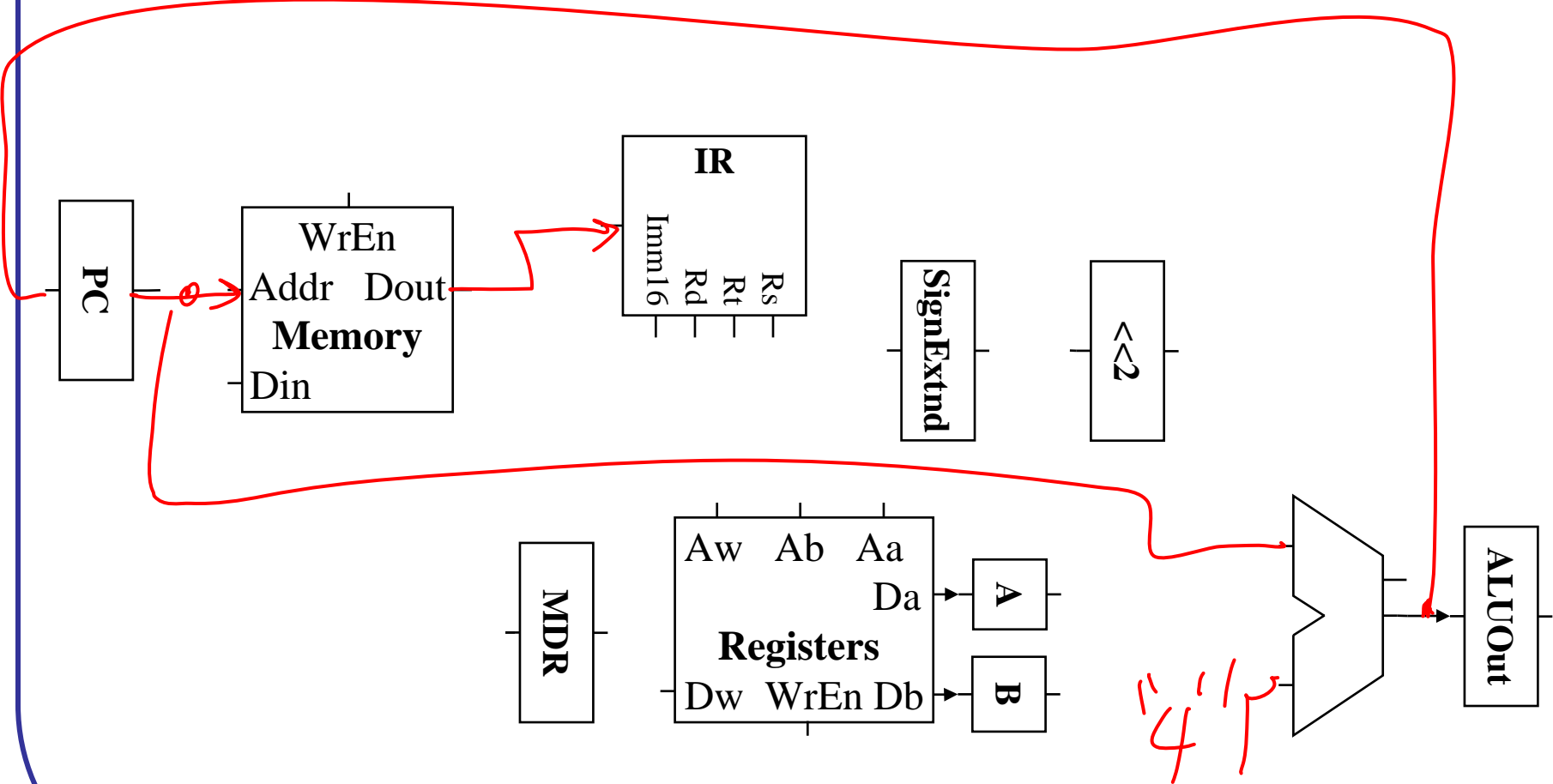
RTL:  $IR \leftarrow Mem[PC]$

- Set the PC to the next instruction (ignore branches)

RTL:  $PC \leftarrow PC + 4$

# Cycle 1 Datapath

$IR = Mem[PC]$   
 $PC \leftarrow PC + 4$



## Cycle 2: Instruction Decode, Register Fetch

---

- Store the two GPR operands into registers A and B

$$A = \text{Reg}[Rs]$$

RTL:

$$B = \text{Reg}[Rt]$$

- Compute Branch Target (in case it's a branch operation, won't have time later)

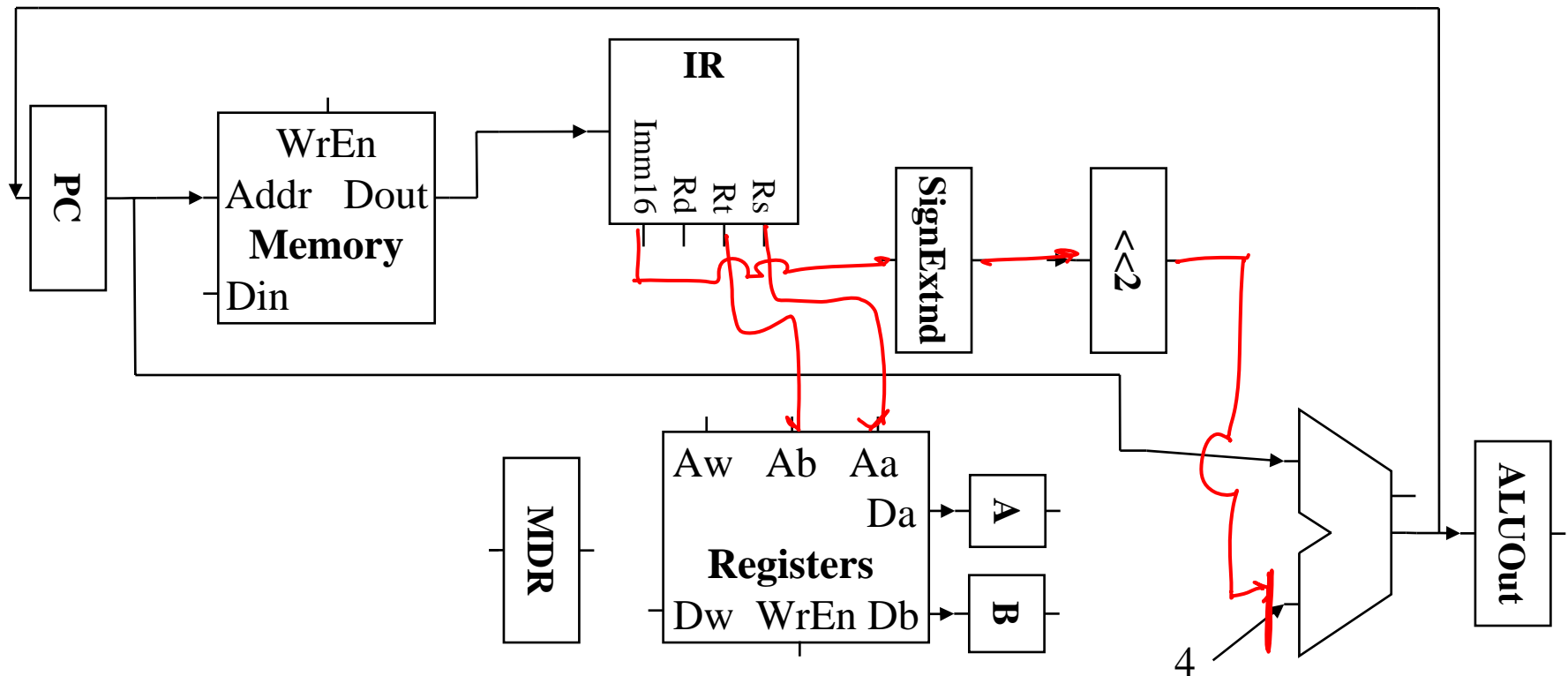
RTL:

$$ALU0_{u4} \leftarrow PC + SE(\text{imm}16) \ll 2$$



# Cycle 1+2 Datapath

*ALUOut = PC + SETimm16 << 2*



## Cycle 3 (Branch)

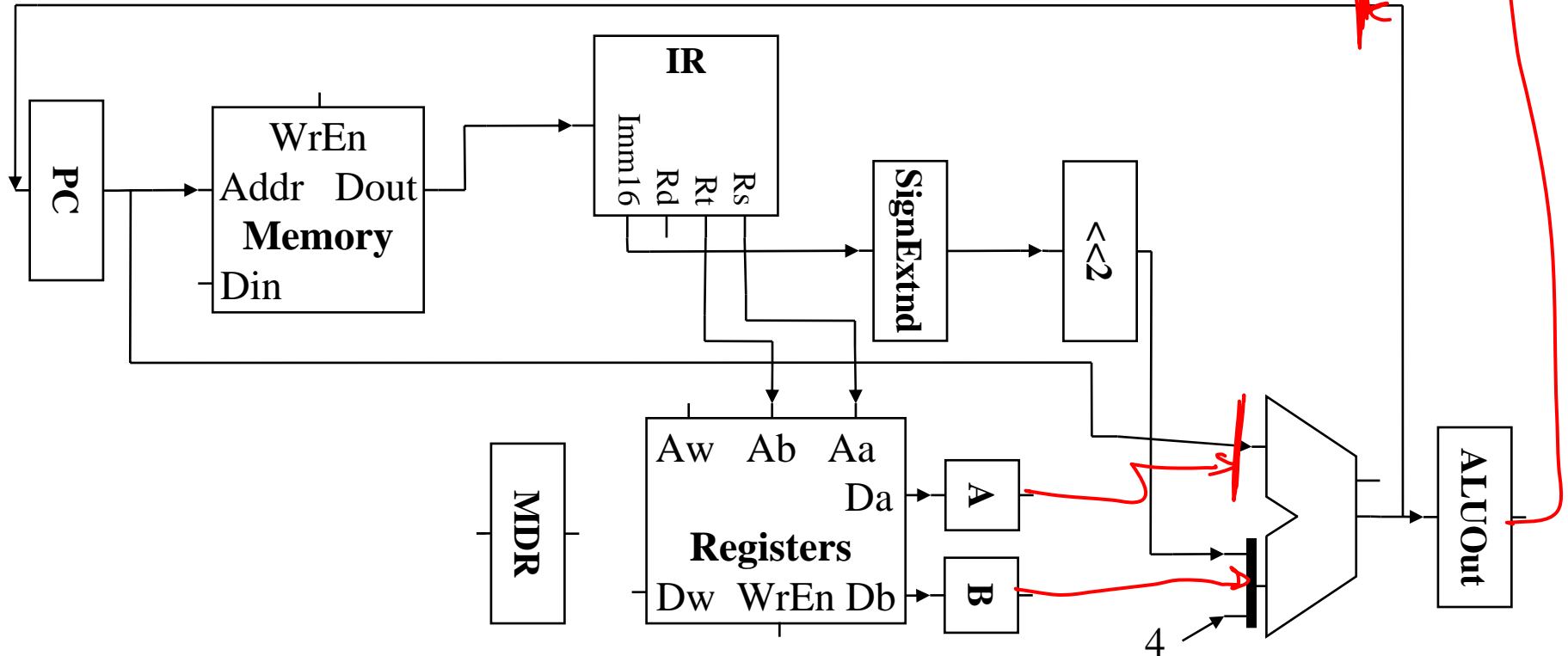
---

- Branch (Beq) - Branch address in ALUout. Set PC to branch address if  $A==B$

RTL:     Zero = (A-B)  
          if (Zero)  
              PC = ALUOut

# Branch Datapath

$Zero = (A - B)$   
if (Zero)  $Pc = ALUOut$



## Cycle 3-4 (Add, Subtract)

---

- Cycle 3: compute function in ALU, operands in A & B. Store in ALUout

RTL:  $ALUout = A \text{ op } B$

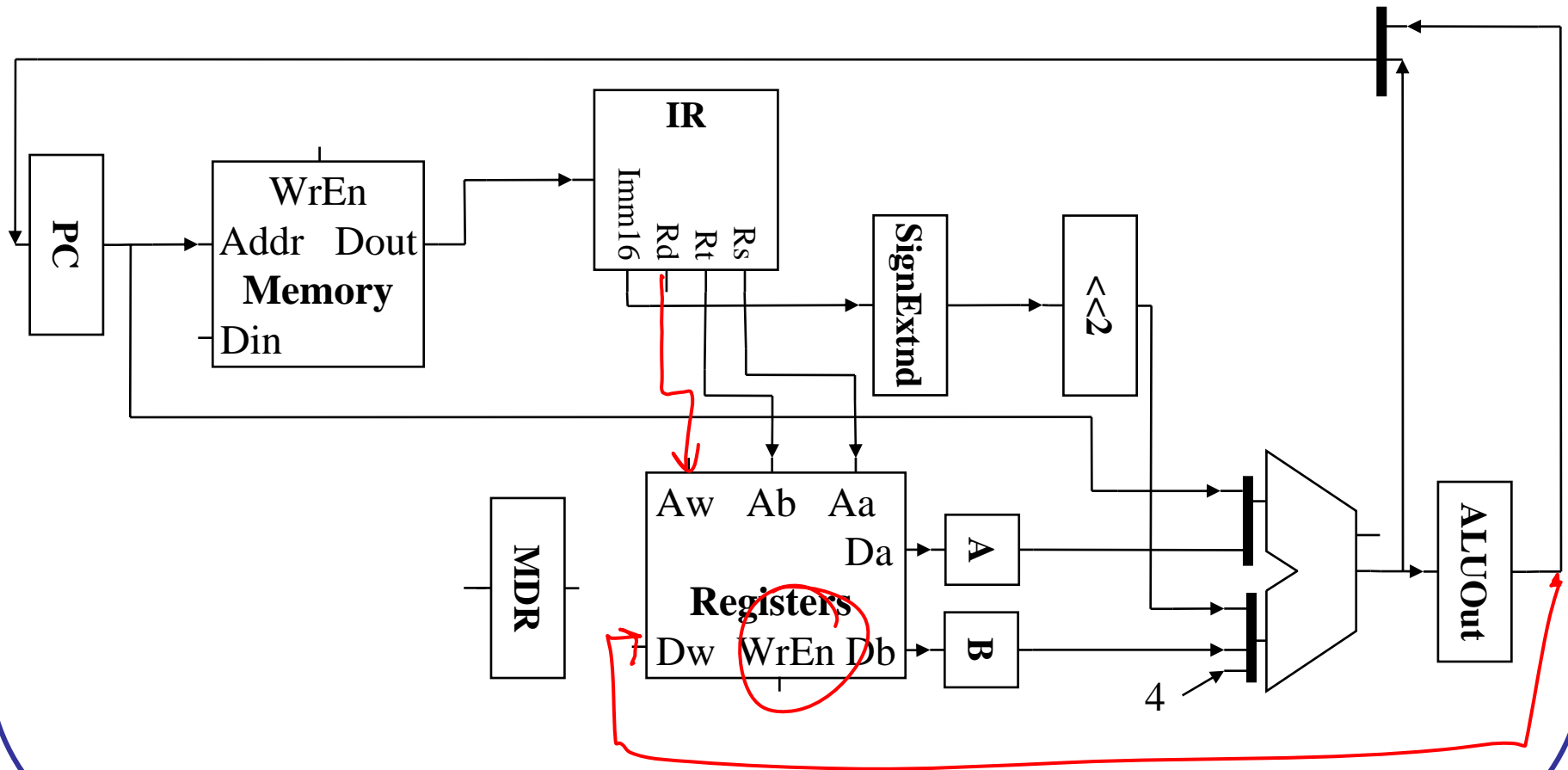
- Cycle 4: Write value from ALUout to destination register

RTL:  $Reg[Rd] = ALUout$

# Branch, R-Type Datapath

3:  $ALUOut = A \oplus B$

4:  $Rg[Rd] = ALUOut$



## Cycle 3-4 (Store)

---

- Cycle 3: compute address from operand A and IR[15-0], put into ALUout

RTL:  $ALUout = A + SE(IR[15:6])$

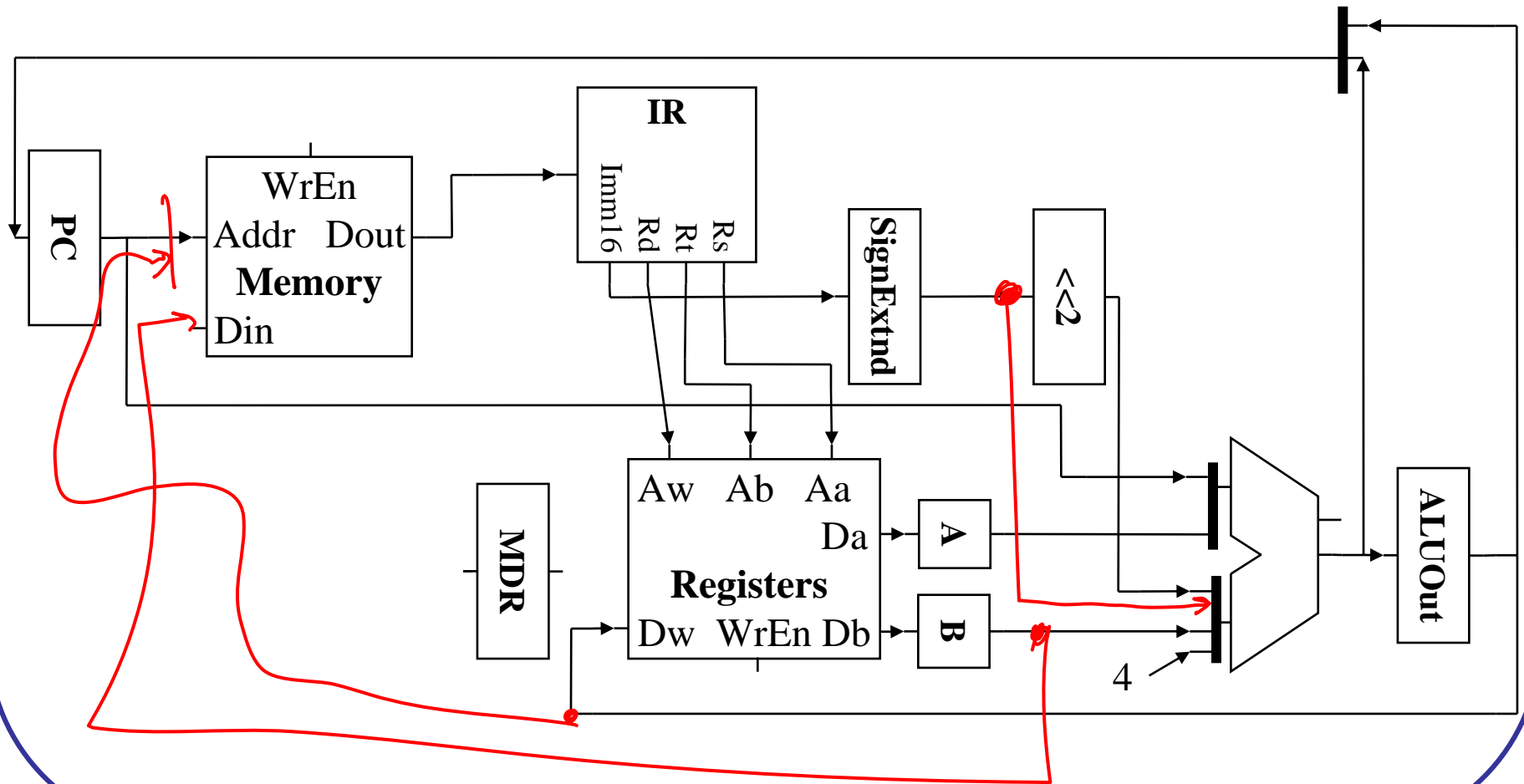
- Cycle 4: Store value from operand B to address specified in ALUout

RTL:  $Mem[ALUout] = B$

# Branch, R-Type, Store Datapath

3:  $ALUOut = A + SE(Imm16)$

4:  $Mem[ALUOut] = B$



## Cycle 3-5 (Load)

---

- Cycle 3: compute address from operand A and IR[15-0], put into ALUout

RTL:

$$ALUout = A + SE(imm16)$$

- Cycle 4: Load value from address specified in ALUout to MDR

RTL:

$$MDR = Mem[ALUout]$$

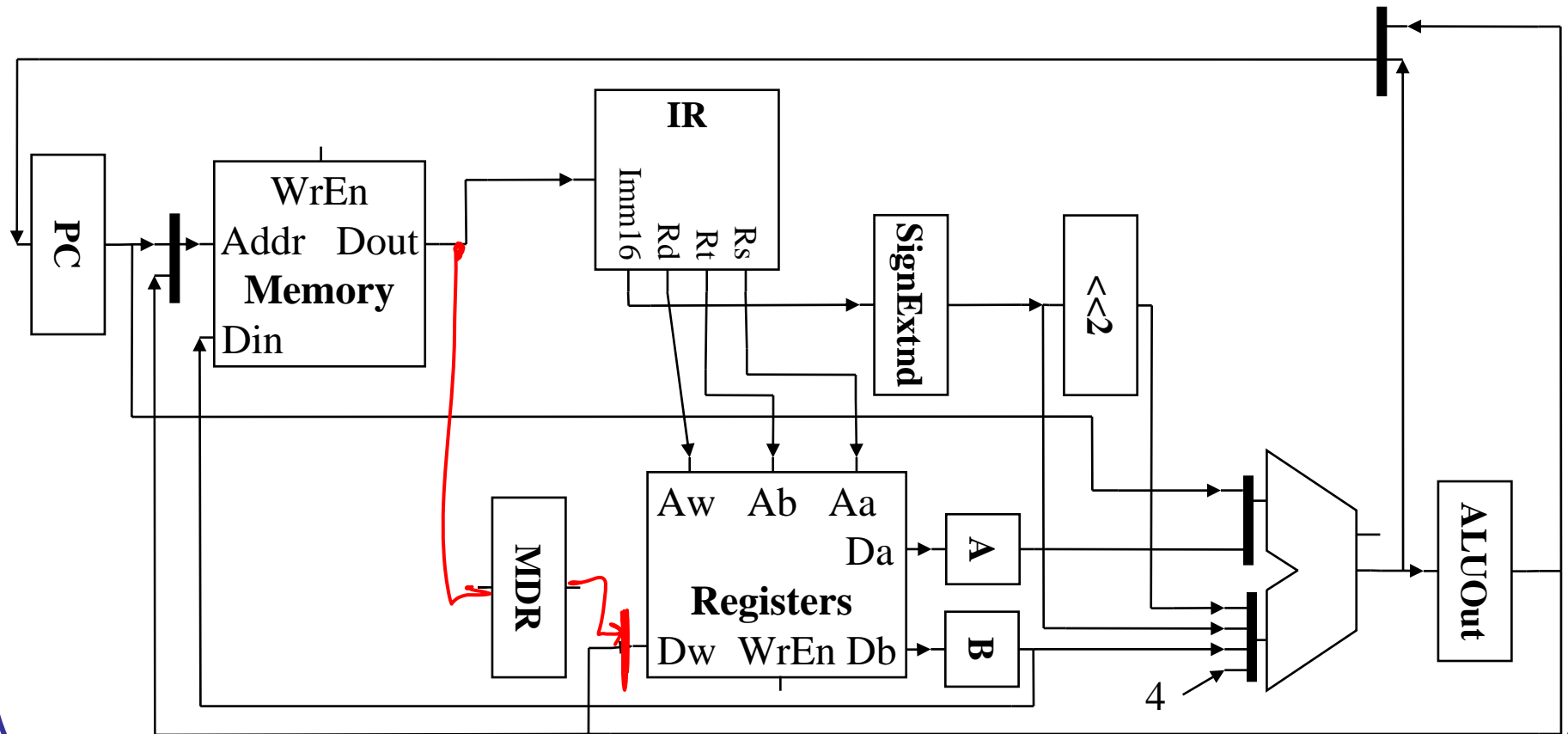
- Cycle 5: Write value from MDR to destination register

RTL:

$$Reg(Rd) = MDR$$



# Multicycle Processor Datapath



# Multicycle Processor Control

- Need to control data path to perform required operations
  - Multiple cycles w/different control values each cycle, so control is an FSM.

Cycle 1:           IR = Mem[PC]; PC = PC + 4

Cycle 2:           A = Reg[RS]; B = Reg[Rt];  
                  ALUOut = PC + (Sign-extend(Imm16)<<2)

~~Cycle 3 Branch: Zero = (A-B); If (zero) PC = ALUOut~~

~~Cycle 3 R-Type: ALUout = A op B~~

~~Cycle 4 R-Type: Reg[Rd] = ALUout~~

~~Cycle 3 Store: ALUout = A + sign-extend(Imm16)~~

~~Cycle 4 Store: Mem[ALUout] = B~~

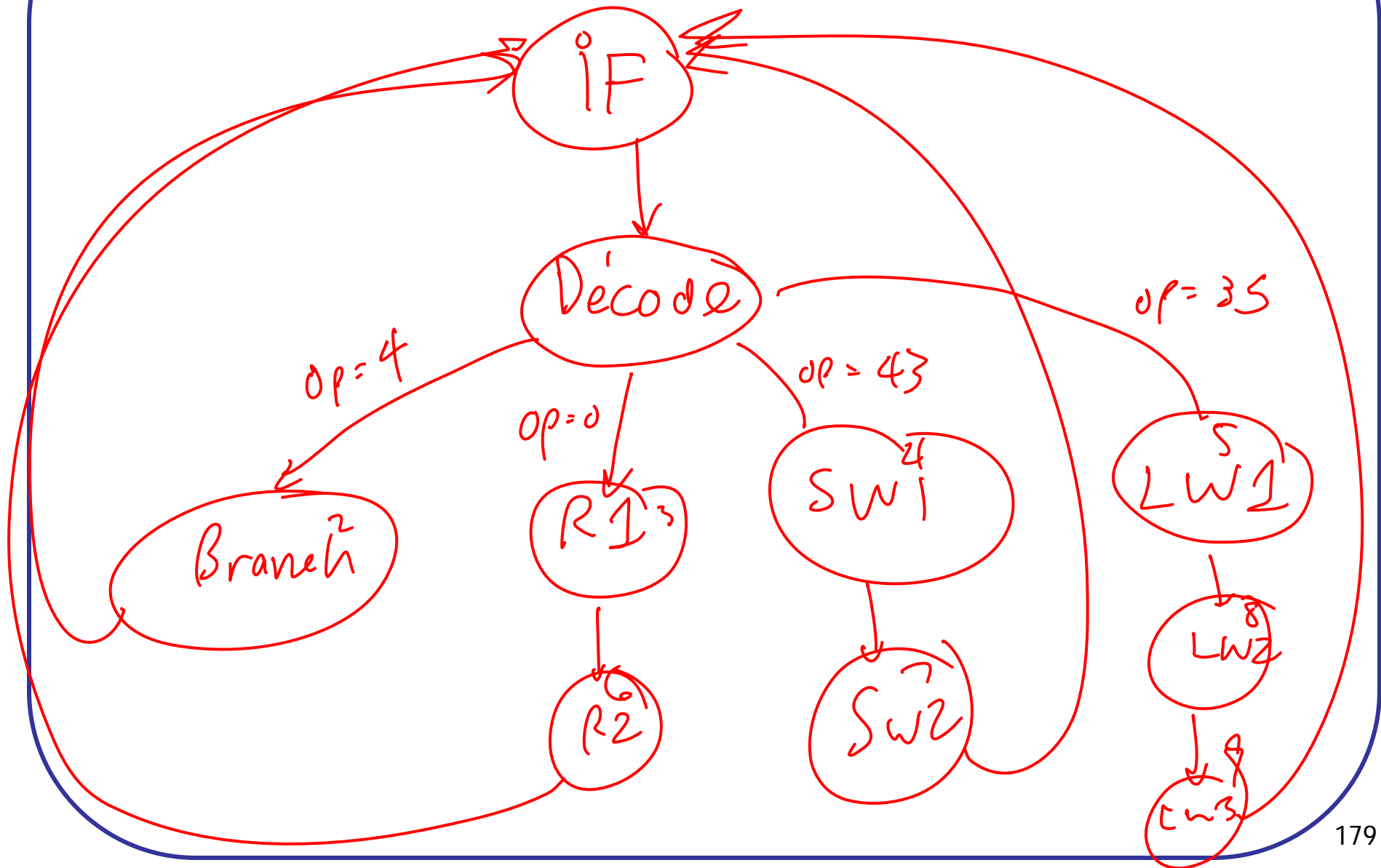
~~Cycle 3 Load: ALUout = A + sign-extend(Imm16)~~

~~Cycle 4 Load: MDR = Mem[ALUout]~~

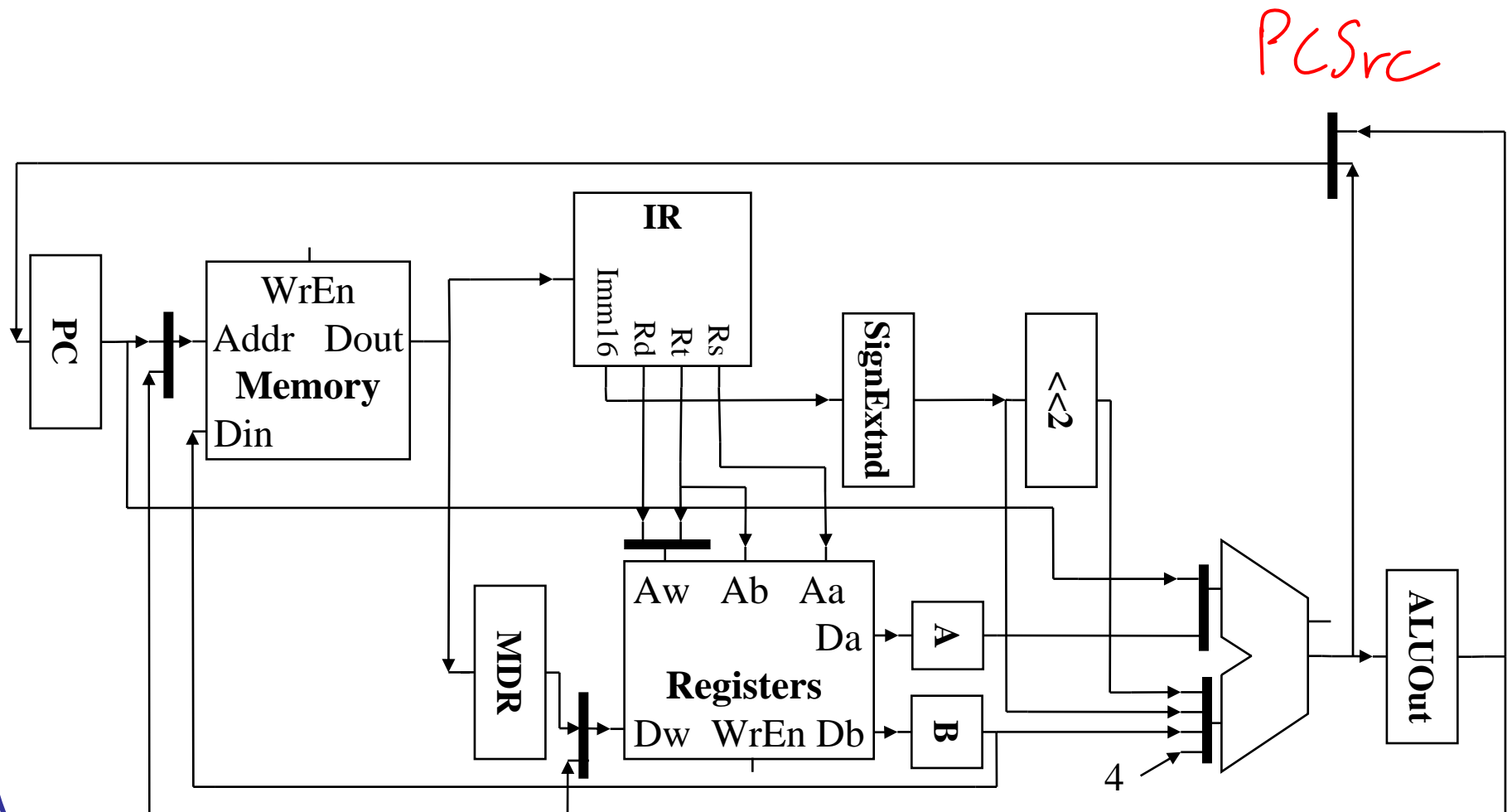
~~Cycle 5 Load: Reg[Rt] = MDR~~

# Control FSM

- Opcodes: LW 35, SW 43, BEQ 4, add/sub 0



# Datapath Control Signals



# Multicycle Datapath Control

State	WE				ALU			Dest	MemIn	RegIn	PCSrc
	PC	Mem	Reg	IR	SrcA	SrcB	Op				
1											
2											
3Br											
3Rt											
4Rt											
3St											
4St											
3Lo											
4Lo											
5Lo											
					A	<<2		Rt[20:16]	PC	MDR	ALU
					PC	SE		Rd[15:11]	ALUout	ALUout	ALUout
						B					
						4					

```

1:
IR = Mem[PC]
PC = PC + 4

2:
A = Reg[Rs]
B = Reg[Rt]
ALUOut = PC
  + (SE(imm16)<<2)

3Br:
Zero = (A-B)
If (zero) PC = ALUout

3Rt:
ALUout = A op B

4Rt:
Reg[Rd] = ALUout

3St:
ALUout = A + SE(Imm16)

4St:
Mem[ALUout] = B

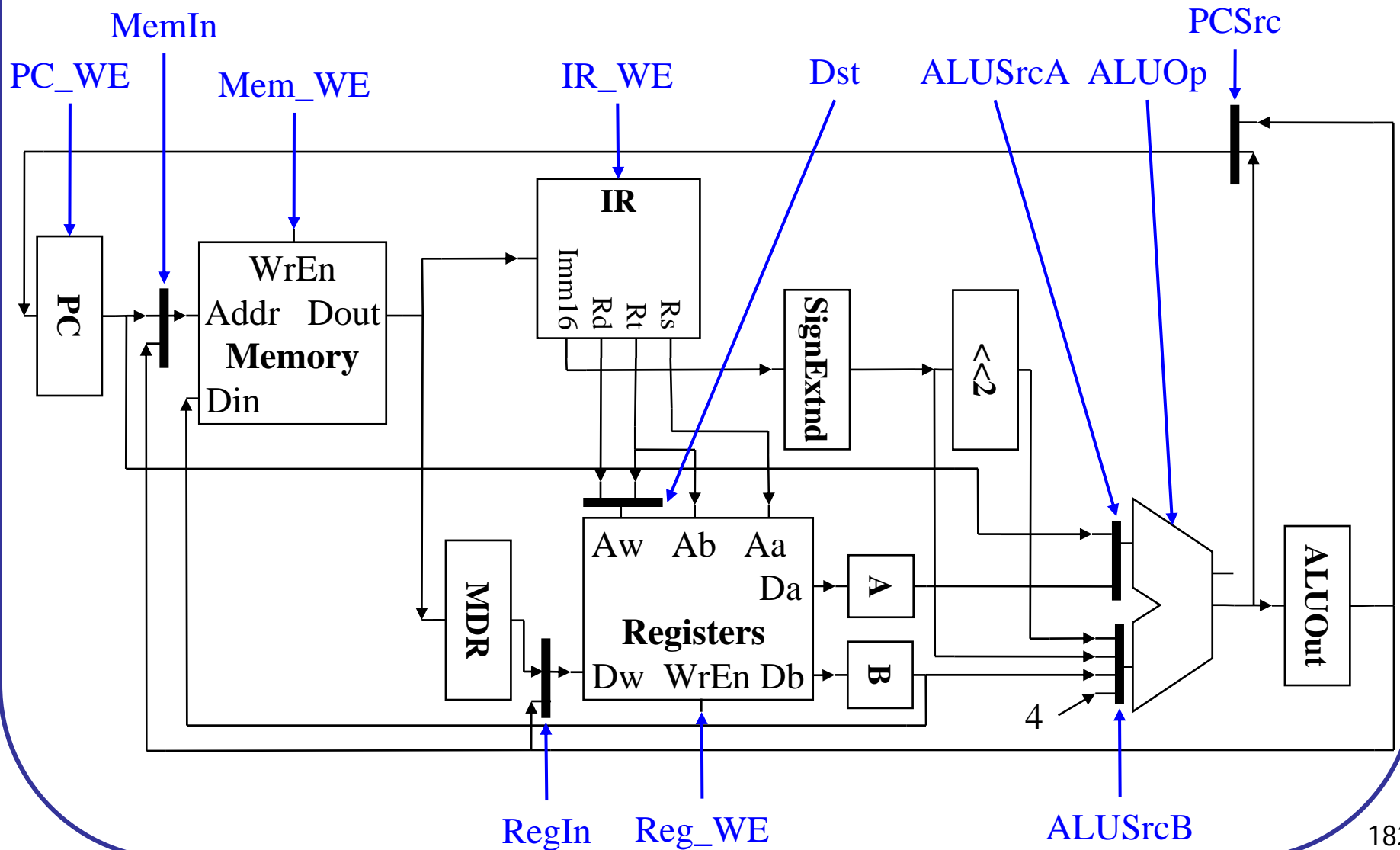
3Lo:
ALUout = A + SE(Imm16)

4Lo:
MDR = Mem[ALUout]

5Lo:
Reg[Rt] = MDR
    
```

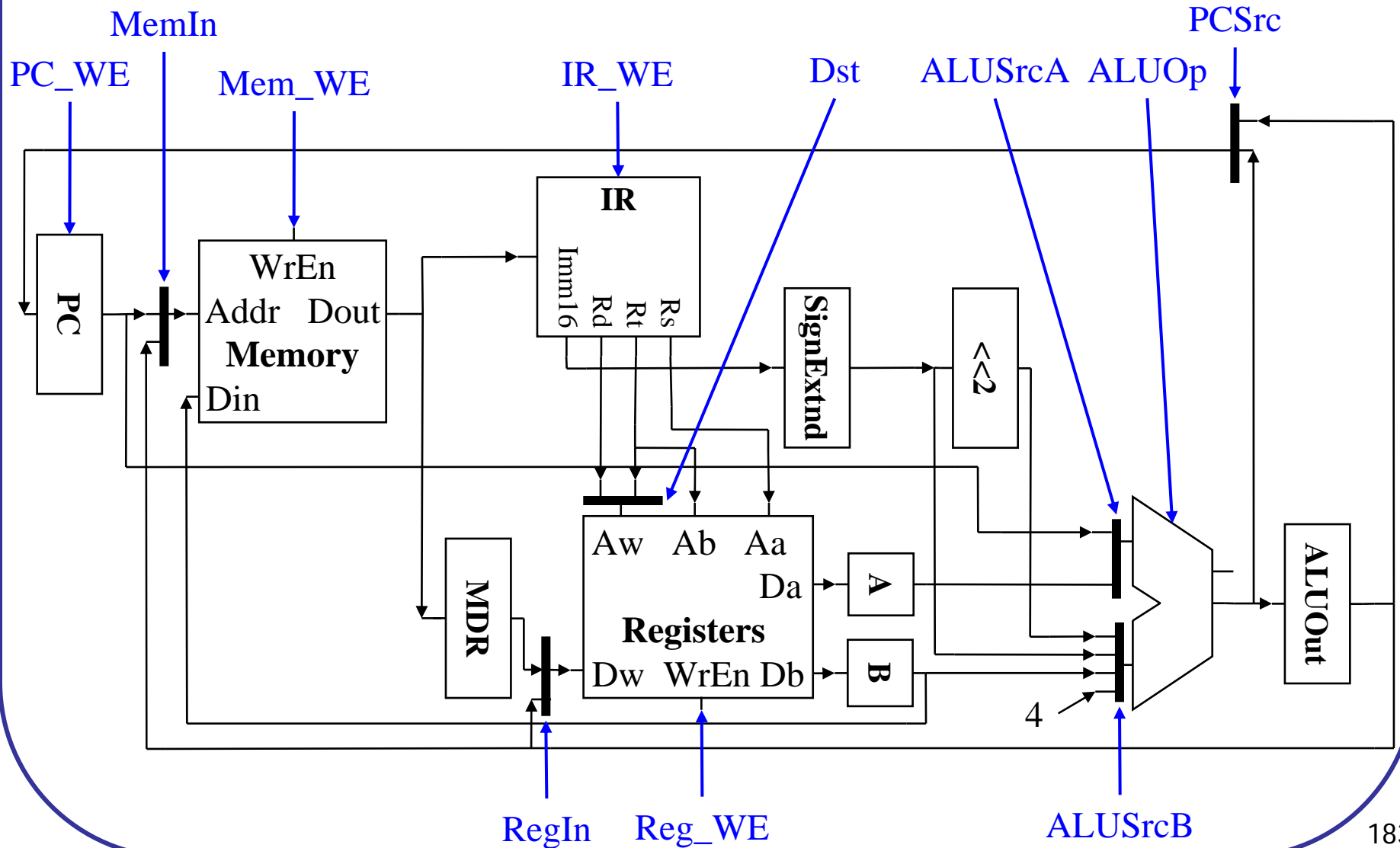
# Cycle 1 Control

- IR = Mem[PC]; PC = PC + 4



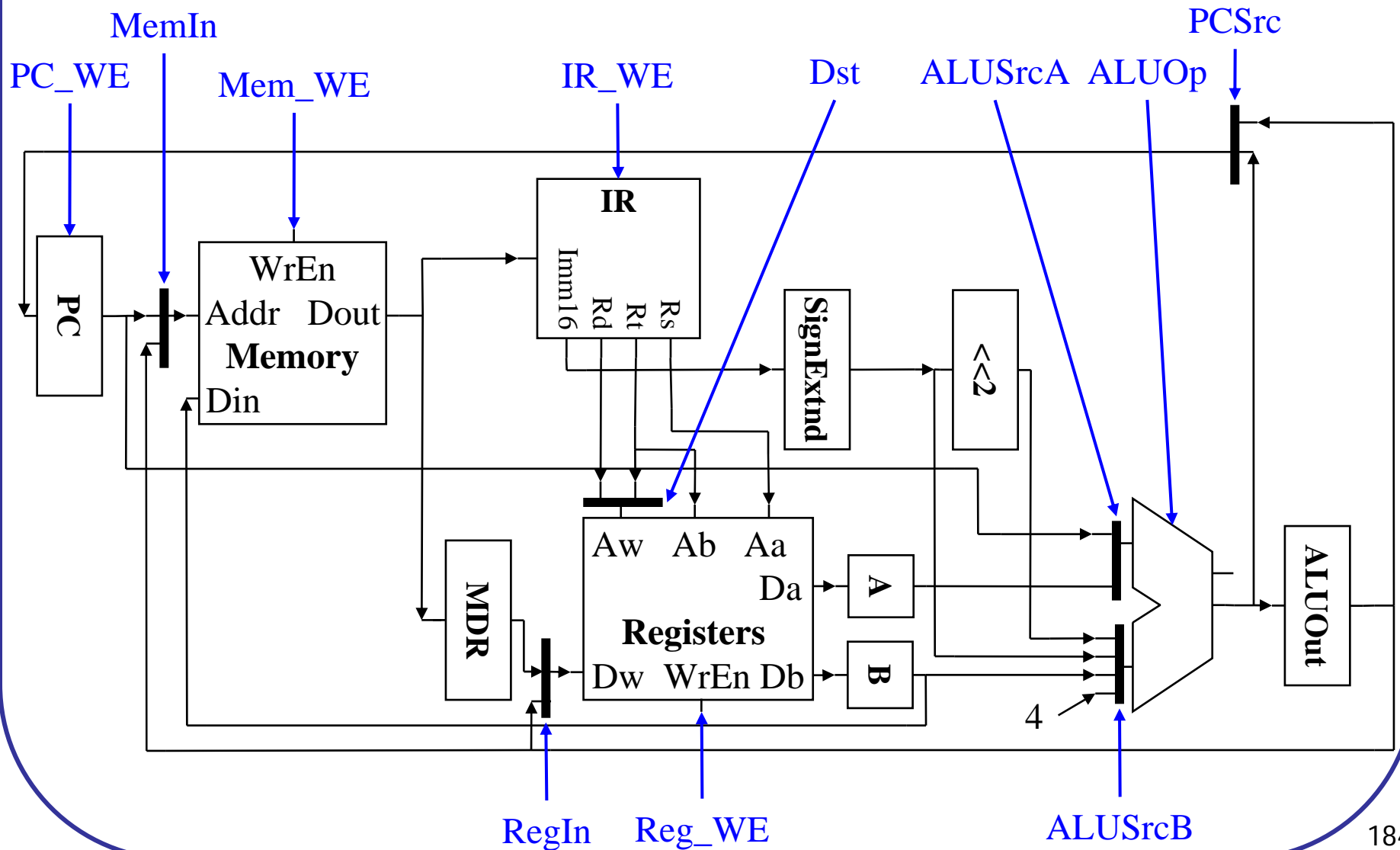
# Cycle 2 Control

- $A = \text{Reg}[Rs]; B = \text{Reg}[Rt]; \text{ALUOut} = \text{PC} + (\text{Sign-extend}(\text{Imm16}) \ll 2)$



# Cycle 3 (Branch) Control

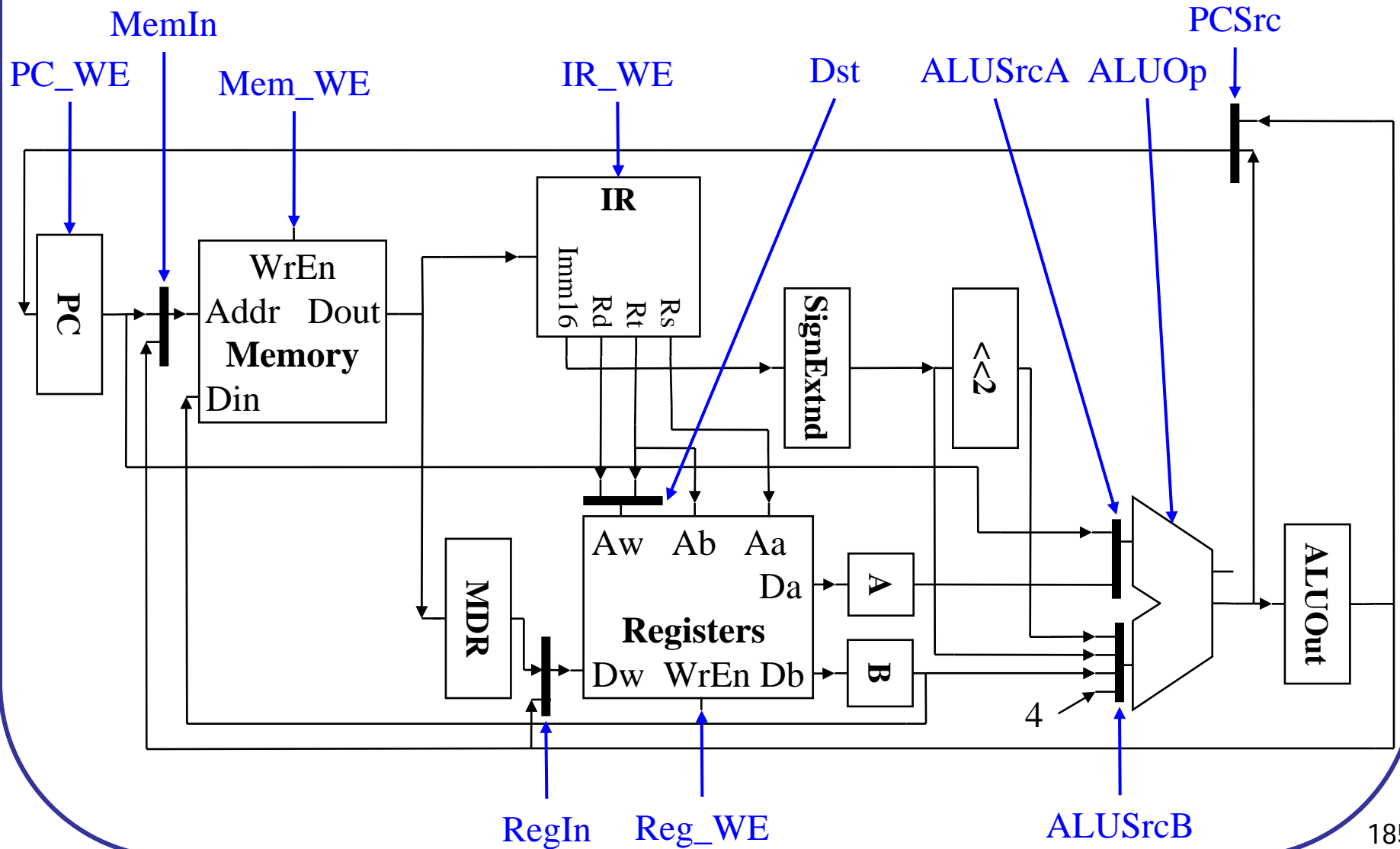
- Zero = (A-B); If (zero) PC = ALUout





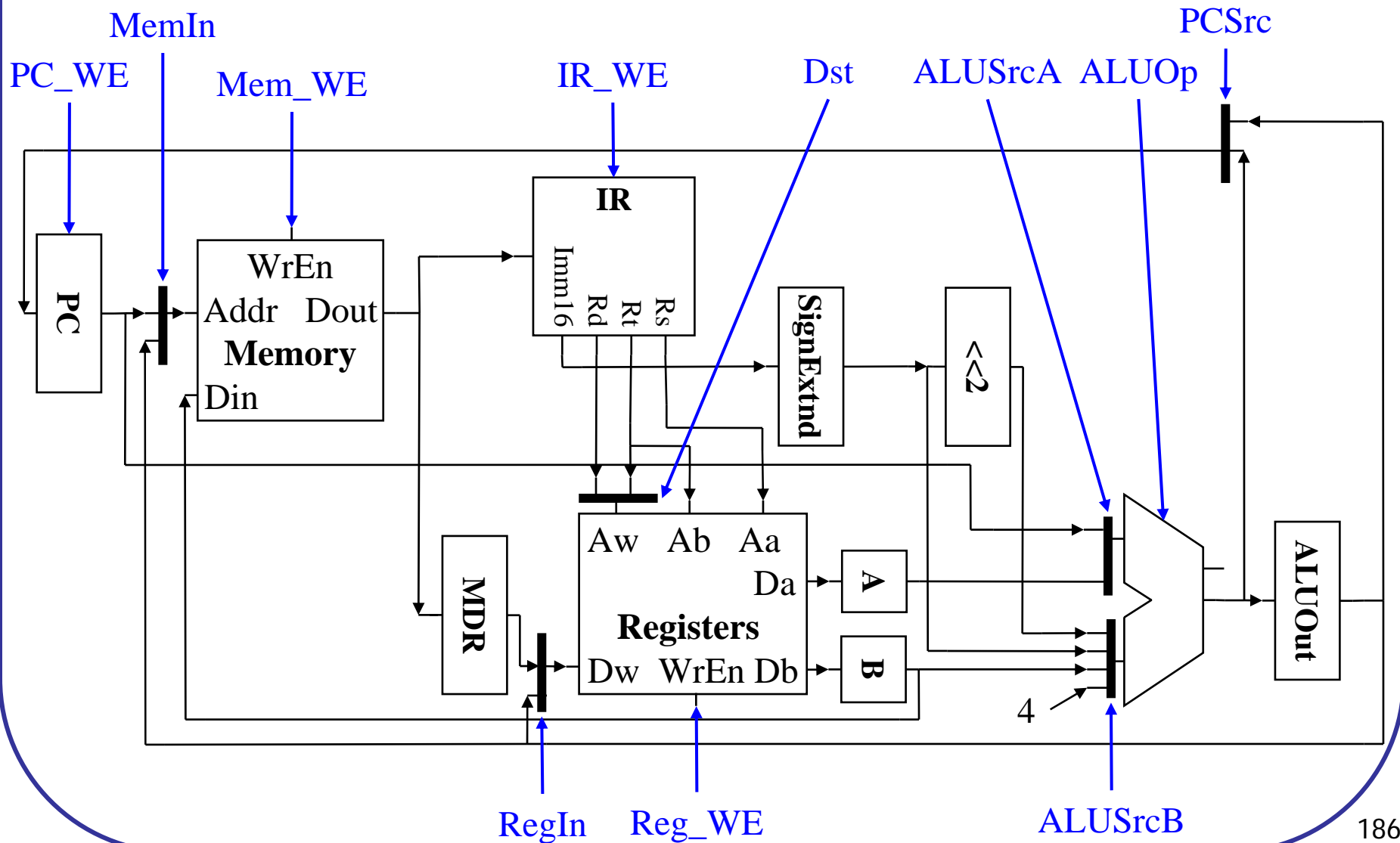
# Cycle 3 (R-Type) Control

- ALUout = A op B



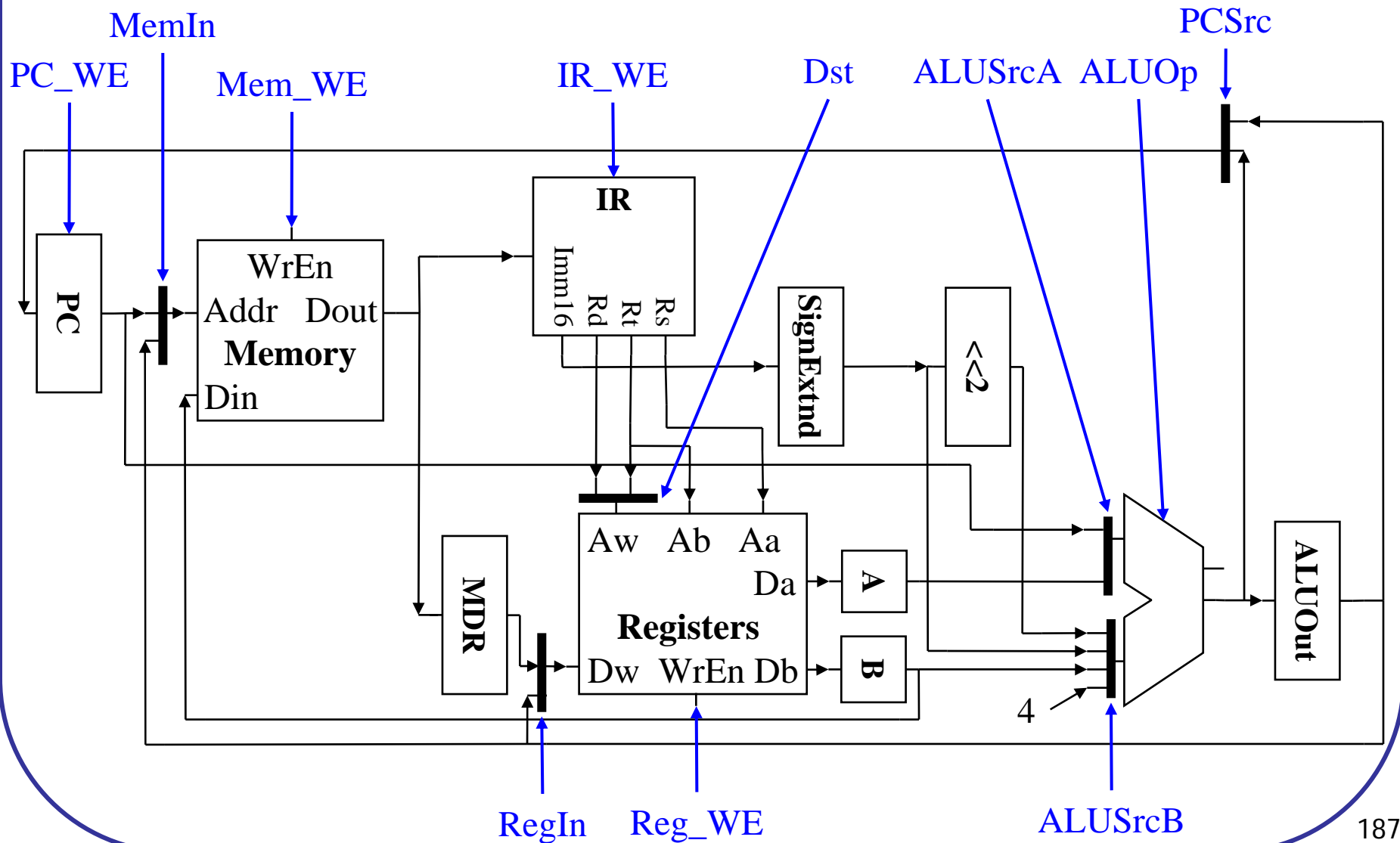
# Cycle 4 (R-Type) Control

- $\text{Reg}[\text{Rd}] = \text{ALUOut}$



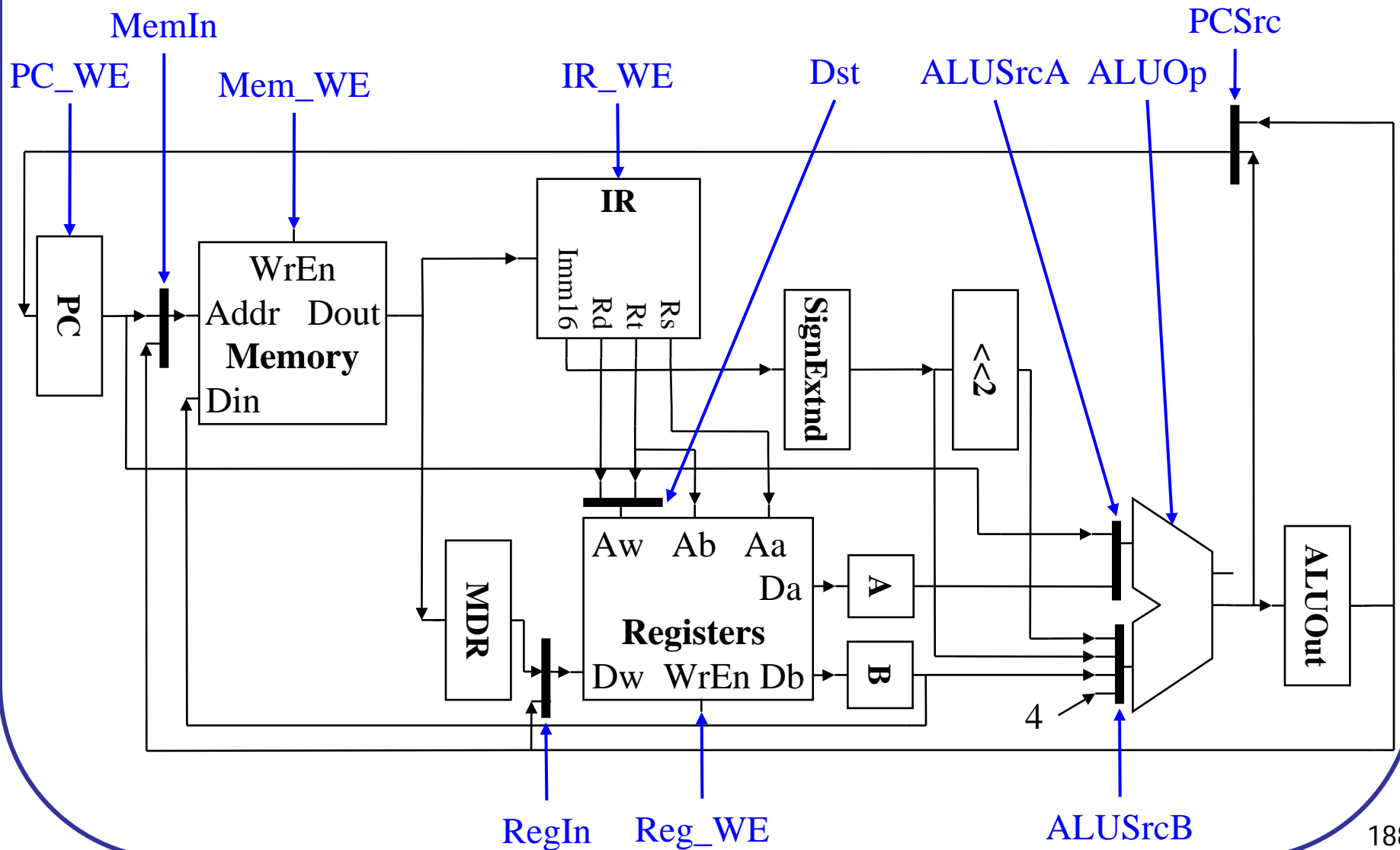
# Cycle 3 (Store) Control

- $ALUout = A + \text{sign-extend}(Imm16)$



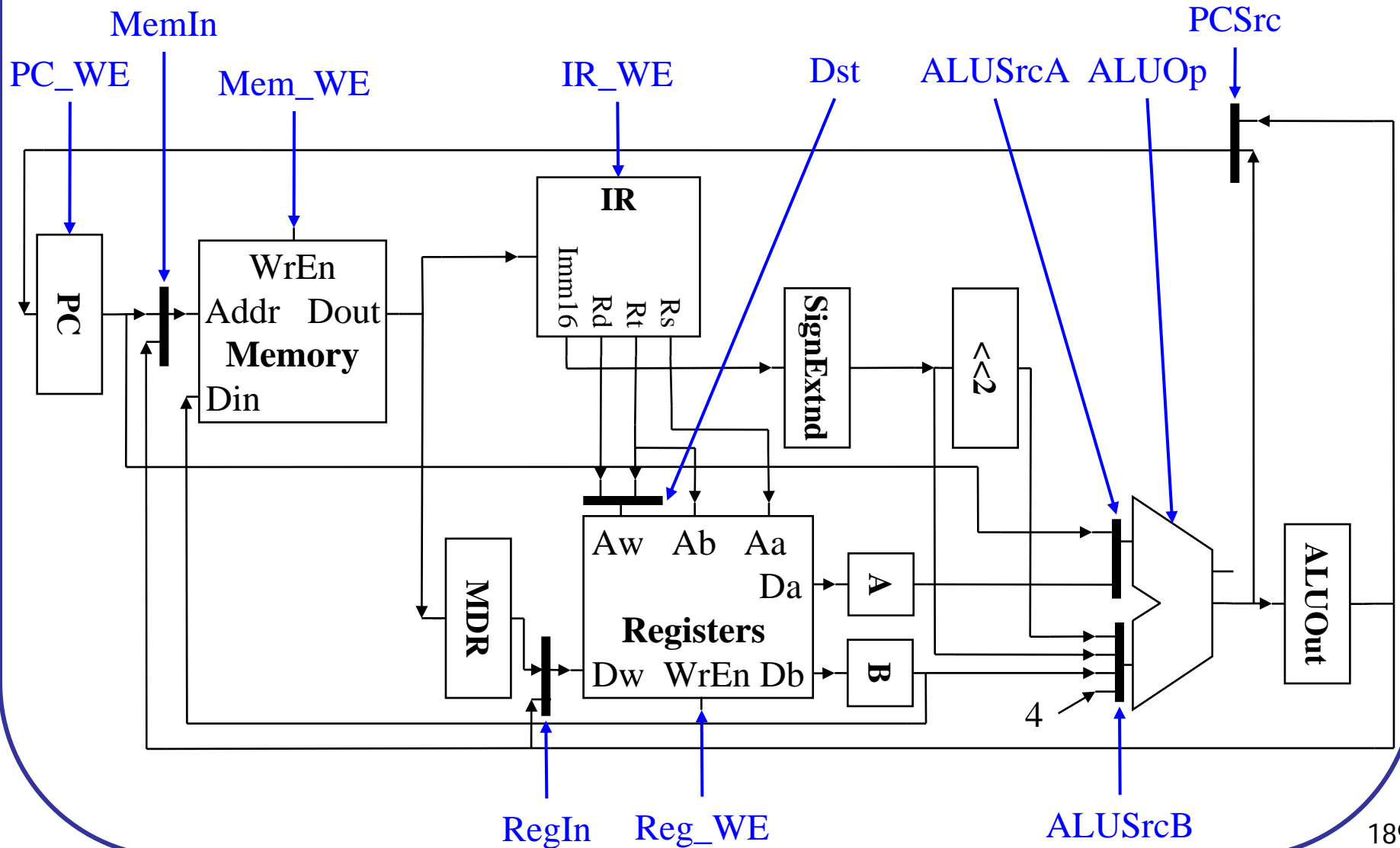
# Cycle 4 (Store) Control

- Mem[ALUout] = B



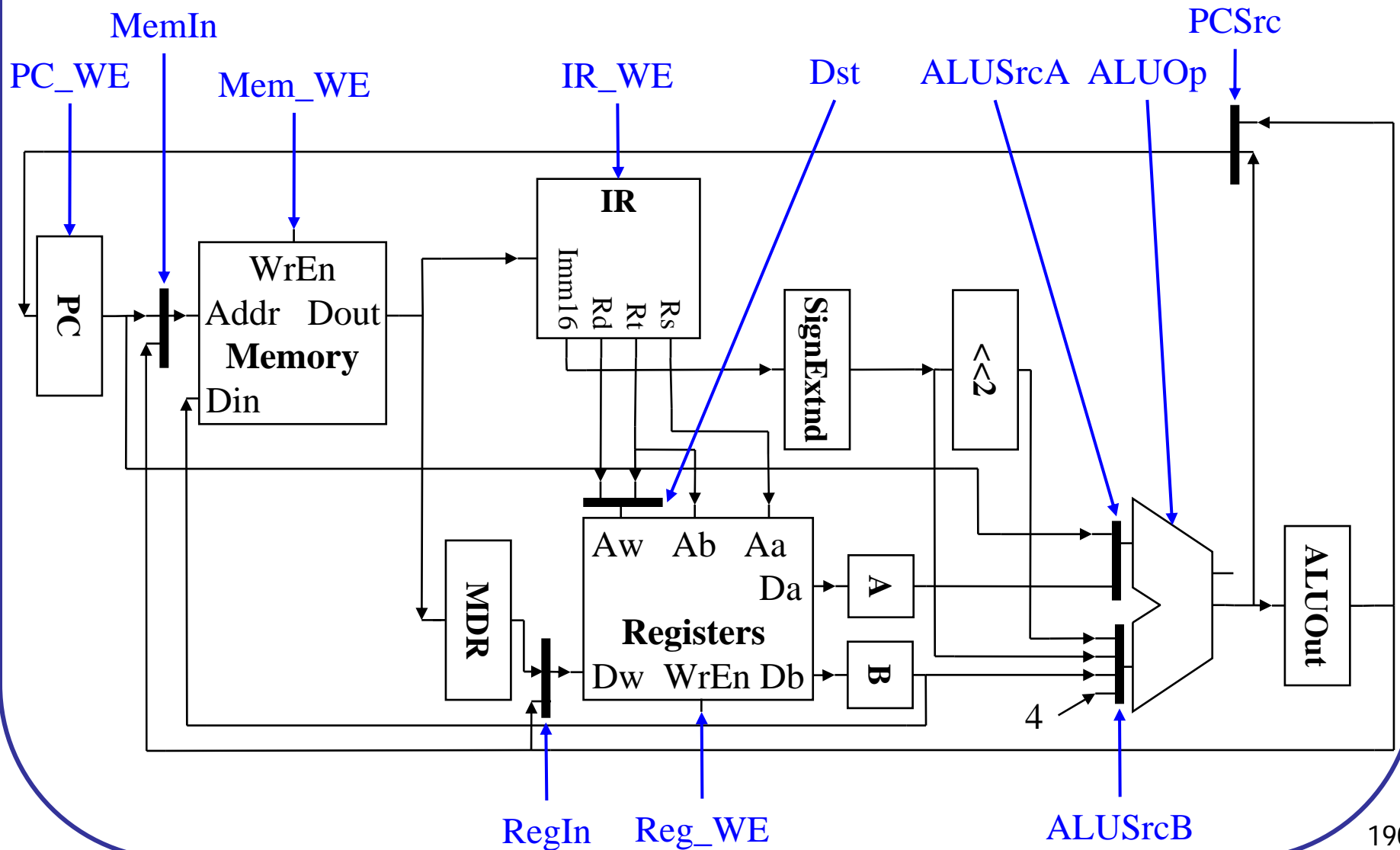
# Cycle 3 (Load) Control

- $ALUout = A + \text{sign-extend}(Imm16)$



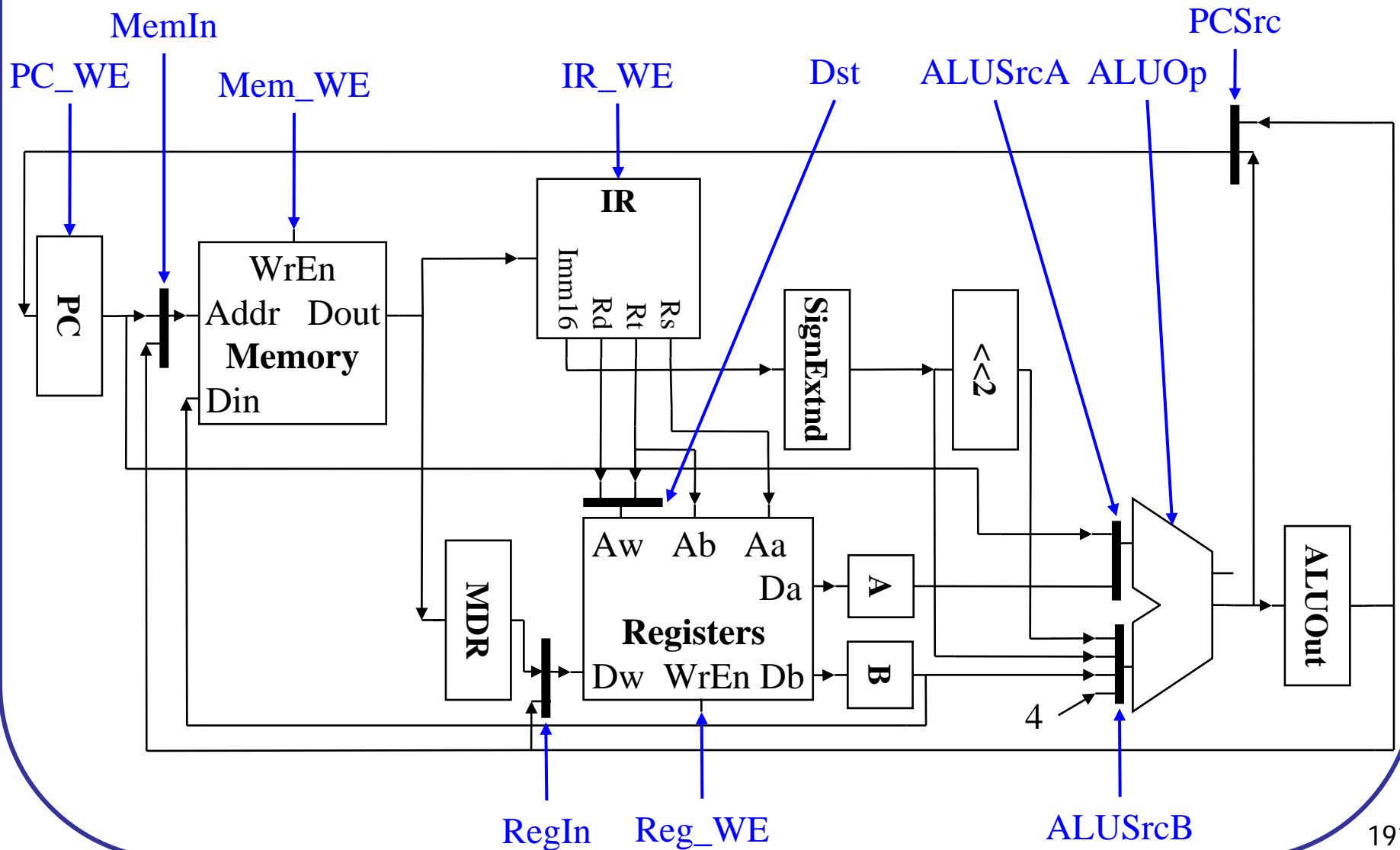
# Cycle 4 (Load) Control

- MDR = Mem[ALUOut]



# Cycle 5 (Load) Control

- $\text{Reg}[\text{Rt}] = \text{MDR}$



## Multicycle CPI

- Compute the CPI of the machine, given the frequencies specified

$$CPI = \sum_{types} (Cycles_{type} * Frequency_{type})$$

Instruction Type	Type Cycles	Type Frequency	Cycles * Freq
ALU		50%	
Load		20%	
Store		10%	
Branch		20%	
CPI:			



## Multicycle Summary

---

- By splitting the single-cycle datapath up we achieve: