

1000  
*Multi-Cycle CPU*

ENGR 3410 - Computer Architecture

Mark L. Chang

Fall 2007

# Performance of Single-Cycle Machine

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

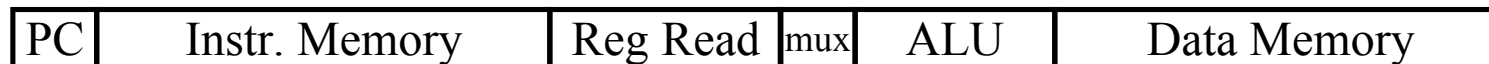
Arithmetic & Logic



Load



Store



Branch

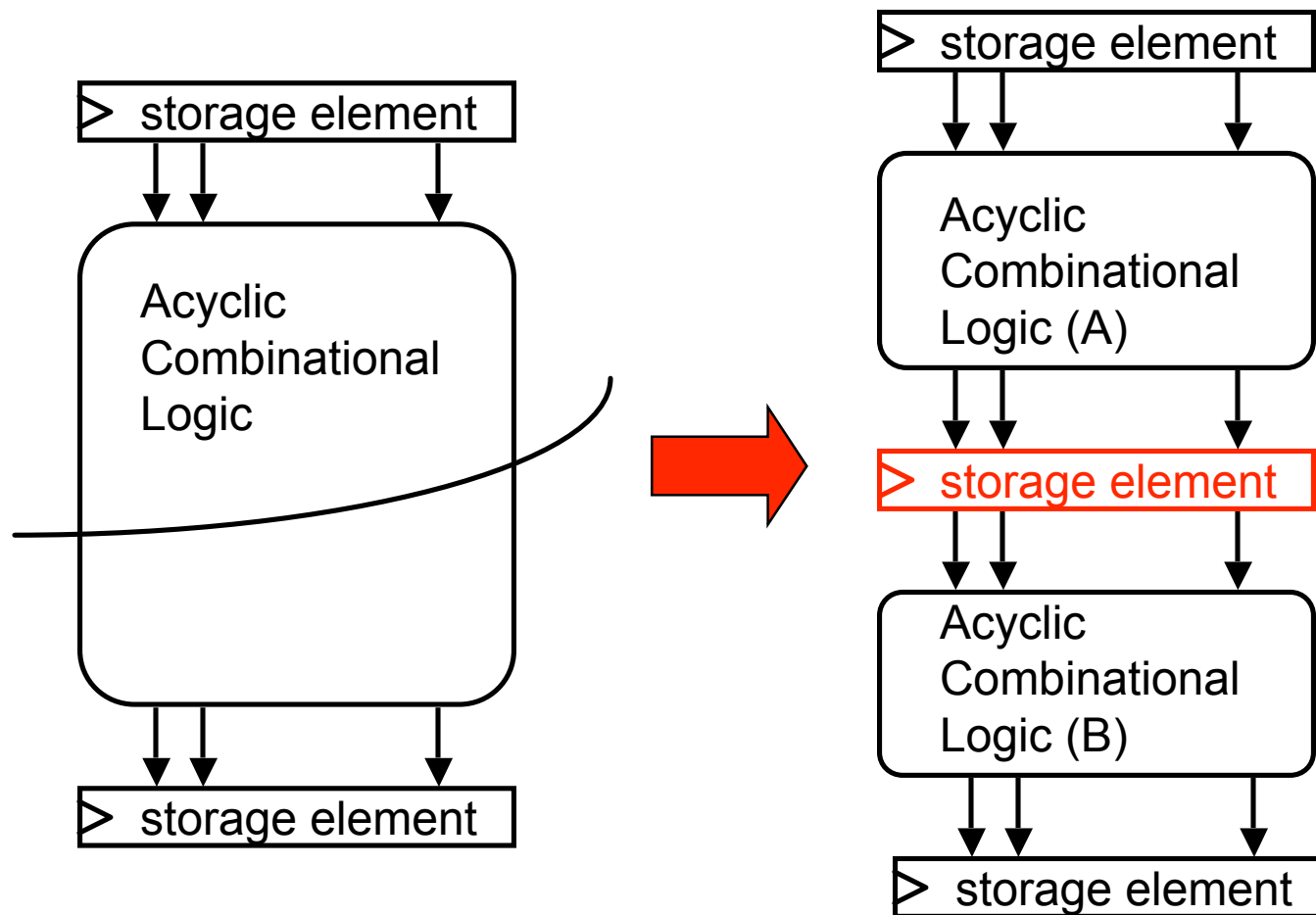


Jump



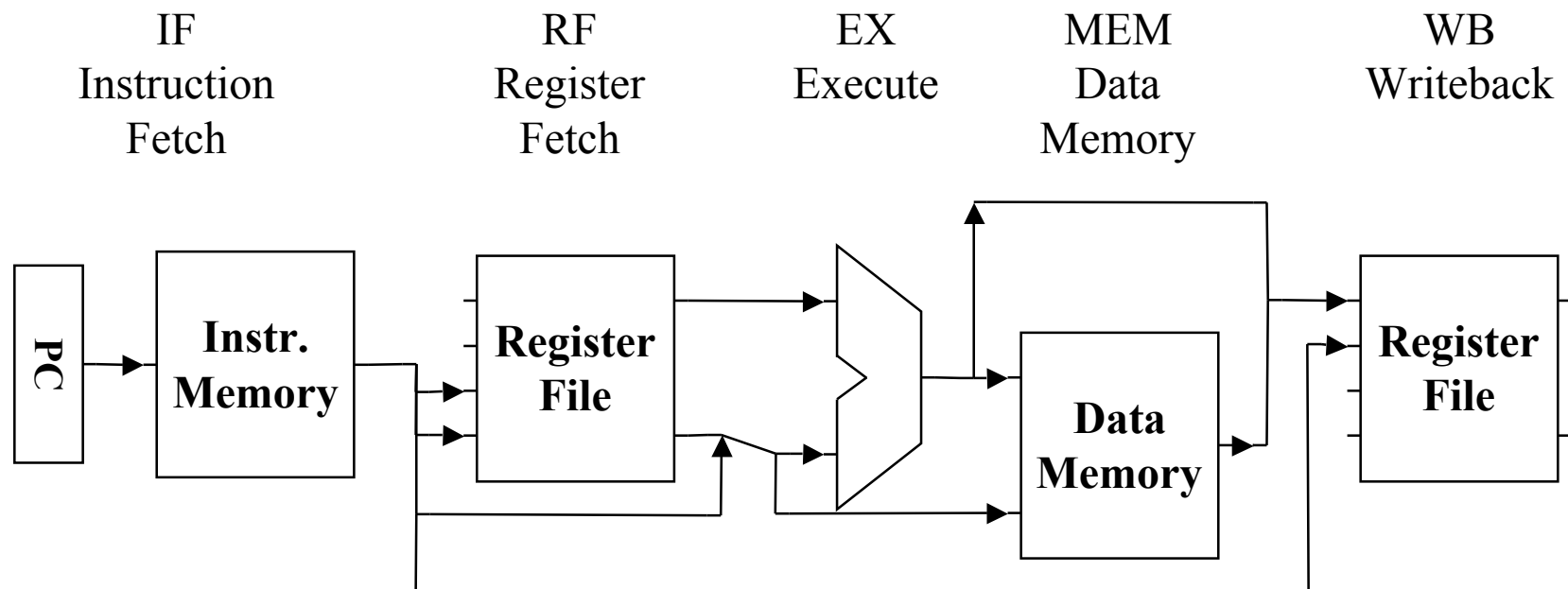
## Reducing Cycle Time

- 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)

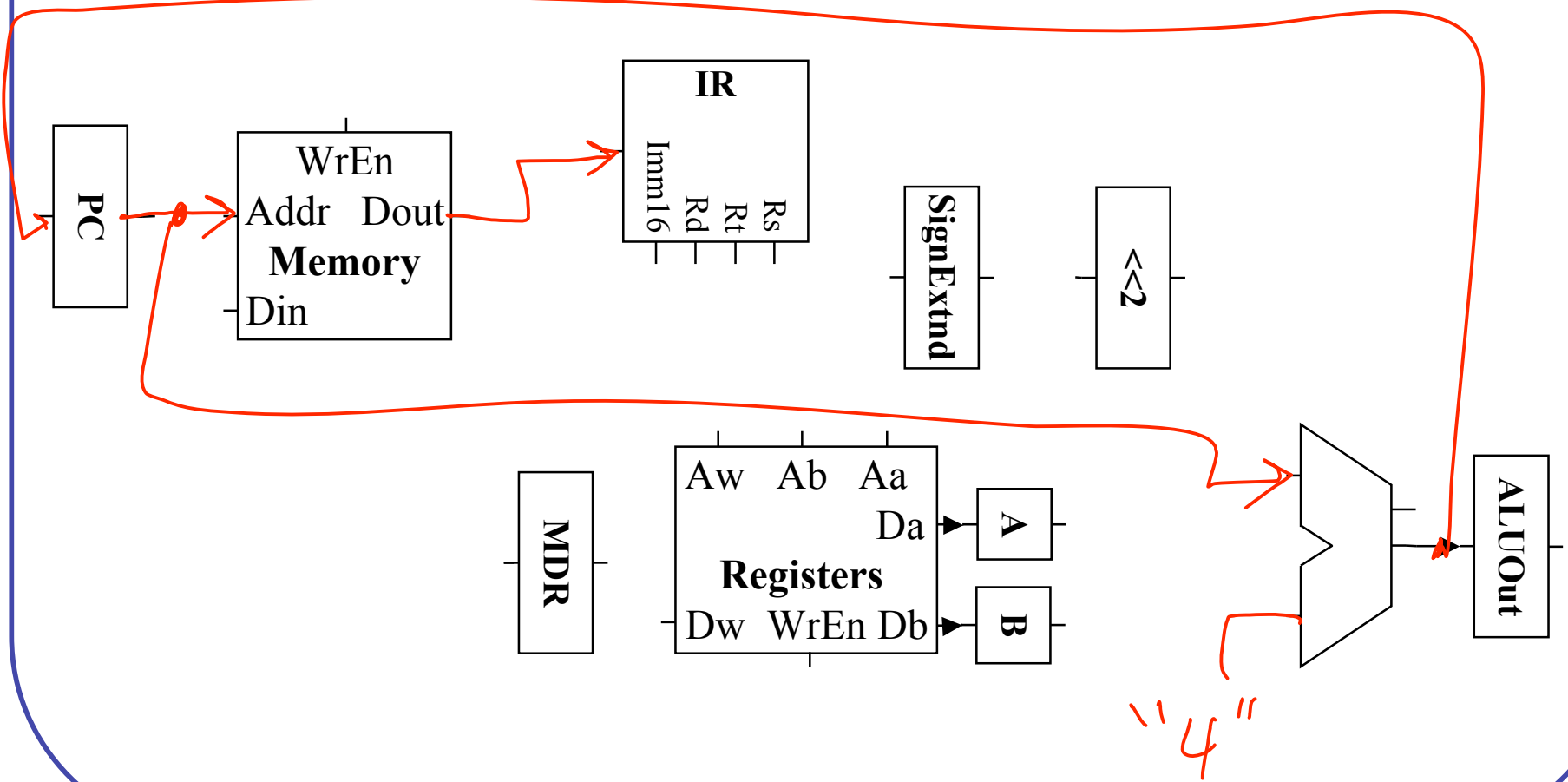
RTL:  $IR \leftarrow Mem[PC]$

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

RTL:  $PC \leftarrow PC + 4$

# Cycle 1 Datapath

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



## Cycle 2: Instruction Decode, Register Fetch

---

- Store the two GPR operands into registers A and B

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

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

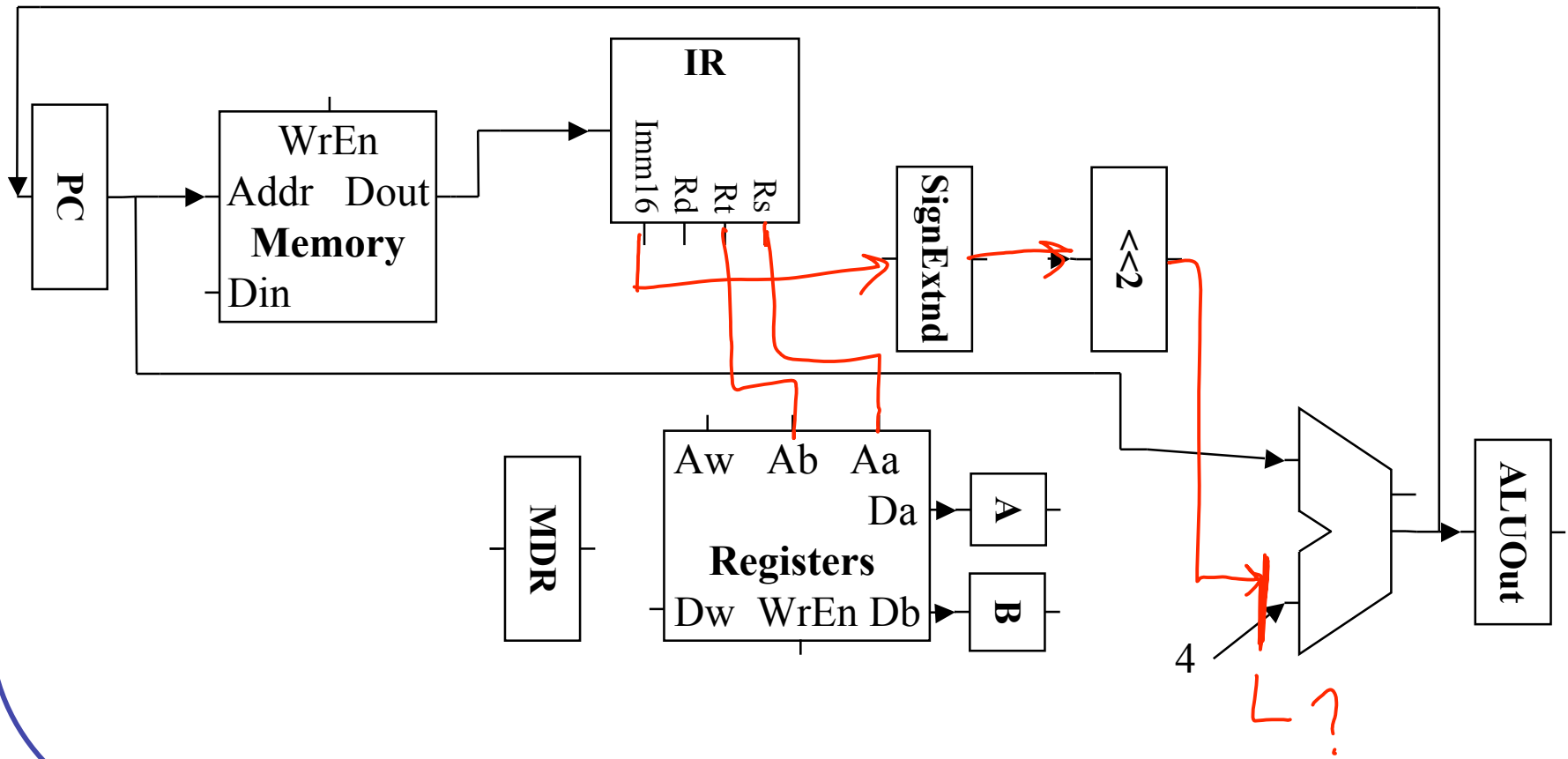
- Compute Branch Target (in case it's a branch operation, won't have time later)

RTL:  $ALUOut = PC + (SE(imm16) \ll 2)$



# Cycle 1+2 Datapath

$A = \text{Reg}(Rs)$   $B = \text{Reg}(Rd)$   $ALUOut = PC + SE(Imm16) \ll 2$



## Cycle 3 (Branch)

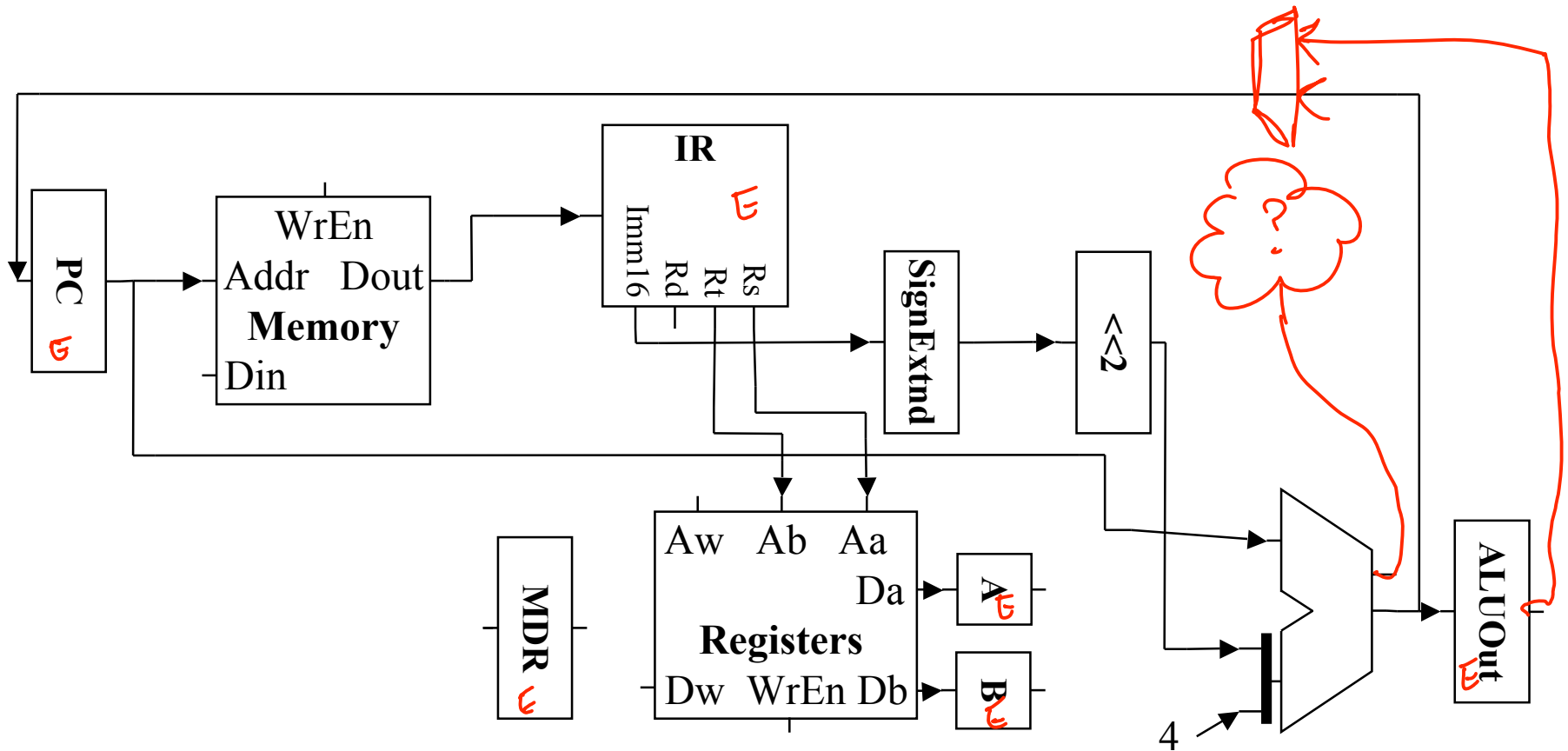
---

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

RTL:  $Zero = (A - B)$   
if (Zero is true)  
     $PC = ALUOUT$

# Branch Datapath

$Z = A - B$ ,  $if(Z) PC = ALUOut$



## Cycle 3-4 (Add, Subtract)

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

RTL:

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

" + "  
" - "

- Cycle 4: Write value from ALUout to destination register

RTL:

$$\text{Reg}[\text{rd}] = \text{ALUout}$$

(rd)

d = MPEWR

if (s != true)

    a = b + c

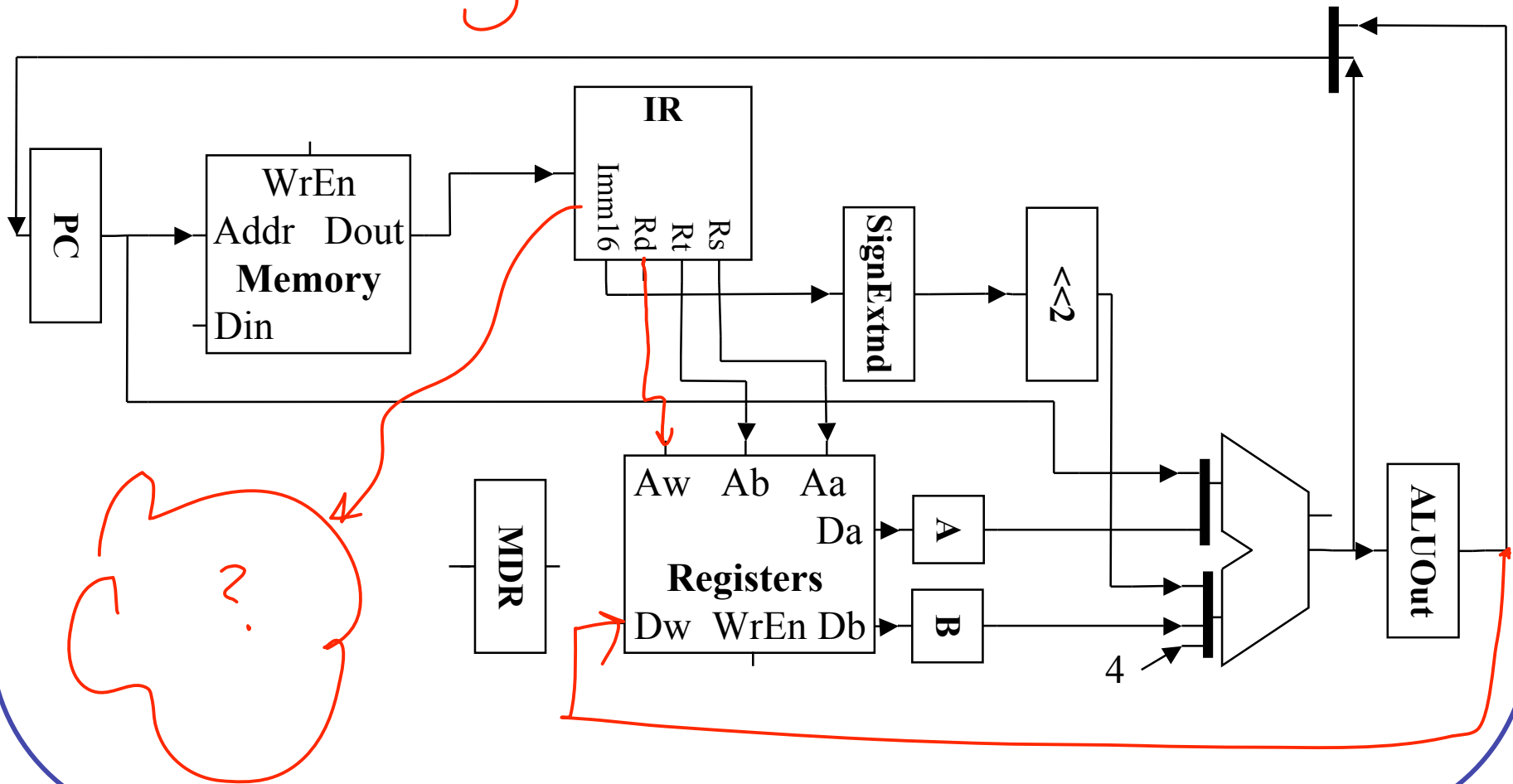
    a = f \* c

has fadd a(b, c, d)

# Branch, R-Type Datapath

3:  $ALUOut = A \neq B$

4:  $Reg[Rd] = ALUOut$



## Cycle 3-4 (Store)

---

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

RTL:  $ALU_{out} = A + SE(im16)$

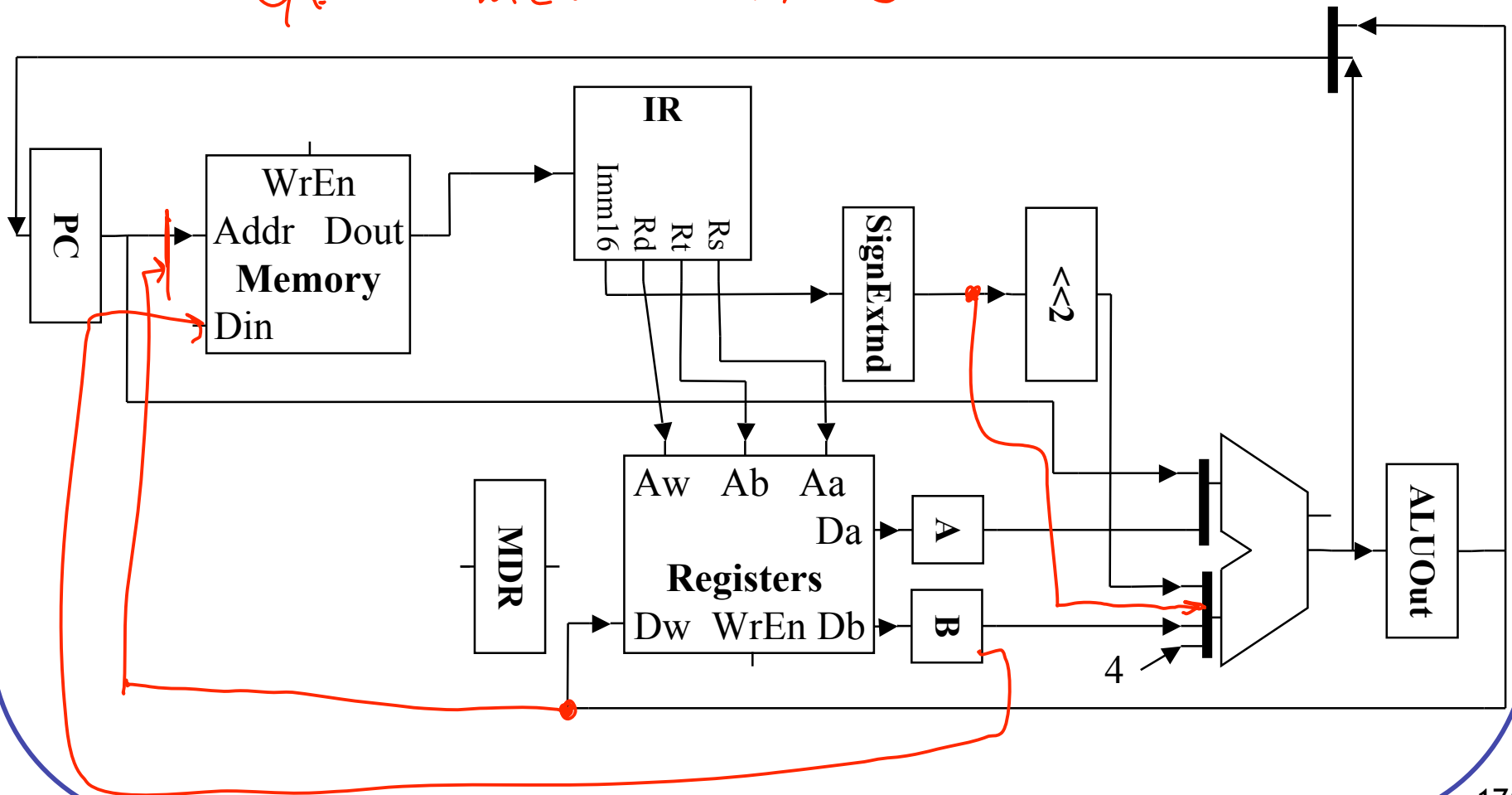
- Cycle 4: Store value from operand B to address specified in ALUout

RTL:  $Mem[ALU_{out}] = B$

# 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

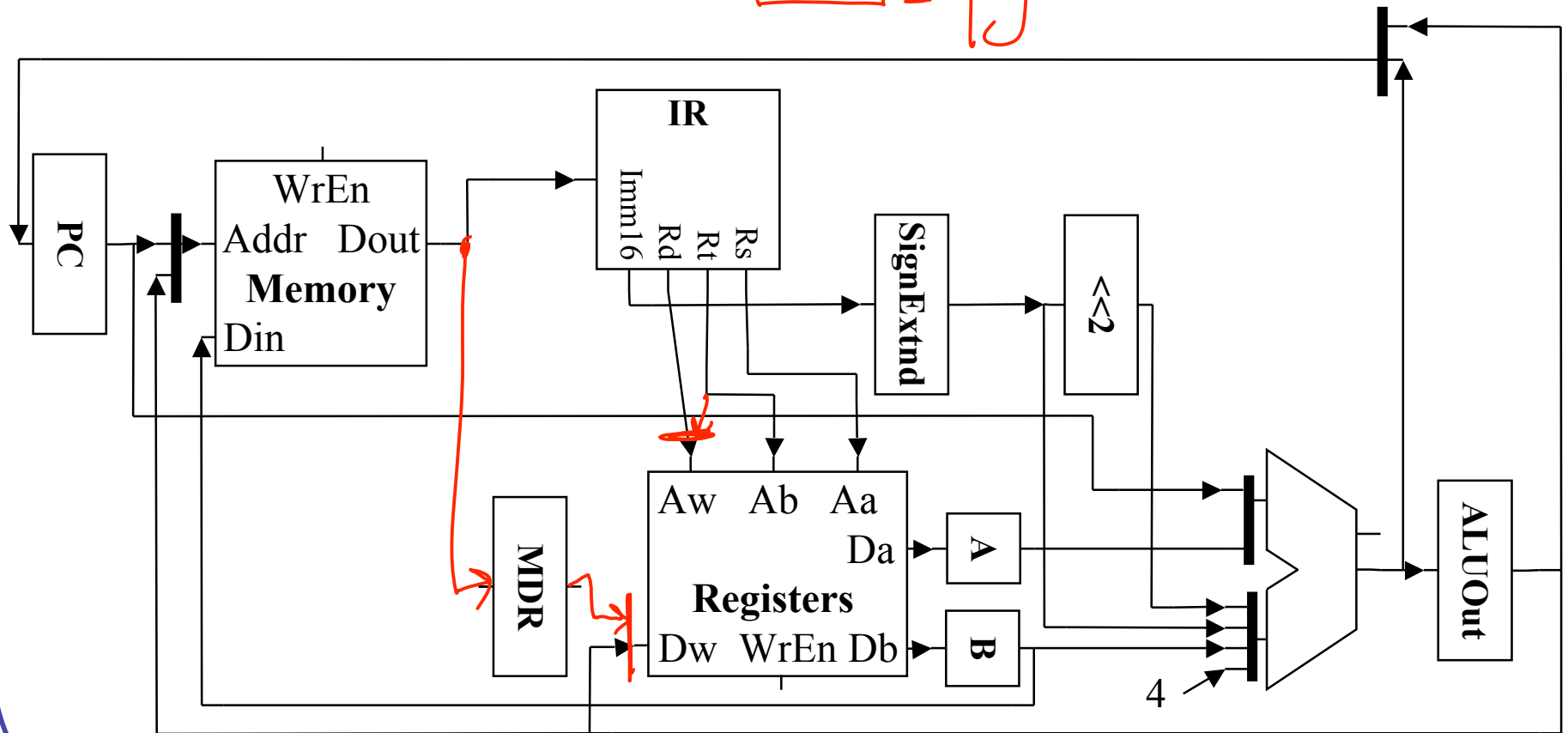
RTL:  $MDR = Mem[ALUout]$

- Cycle 5: Write value from MDR to destination register

RTL:  $Reg[R+] = MDR$



# 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

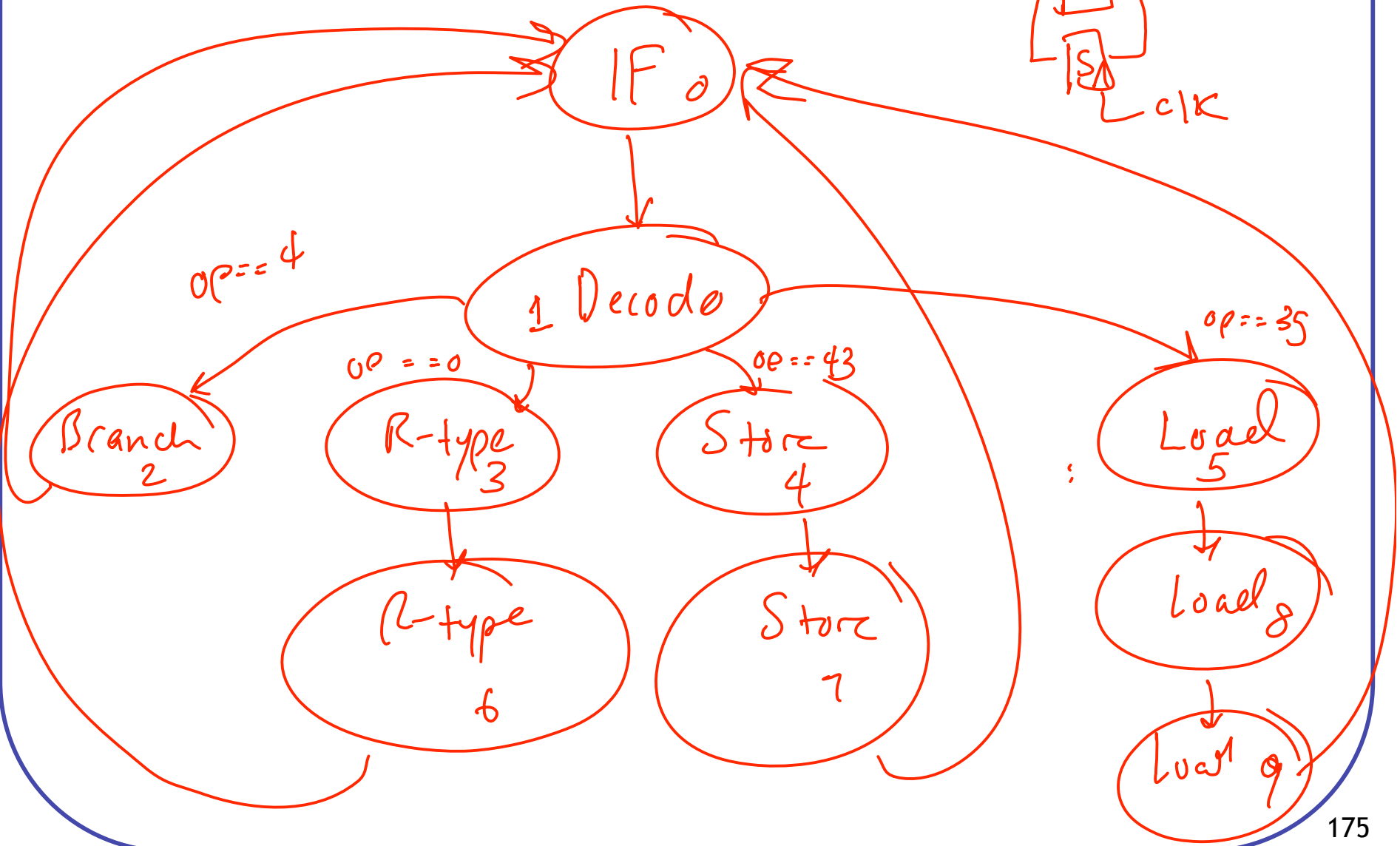
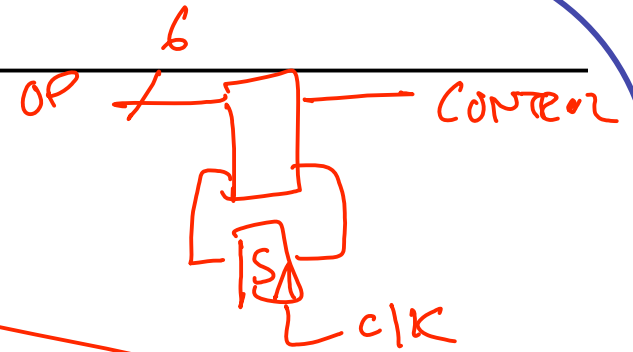
Cycle 3 Load: ALUout = A + sign-extend(Imm16)

Cycle 4 Load: MDR = Mem[ALUout]

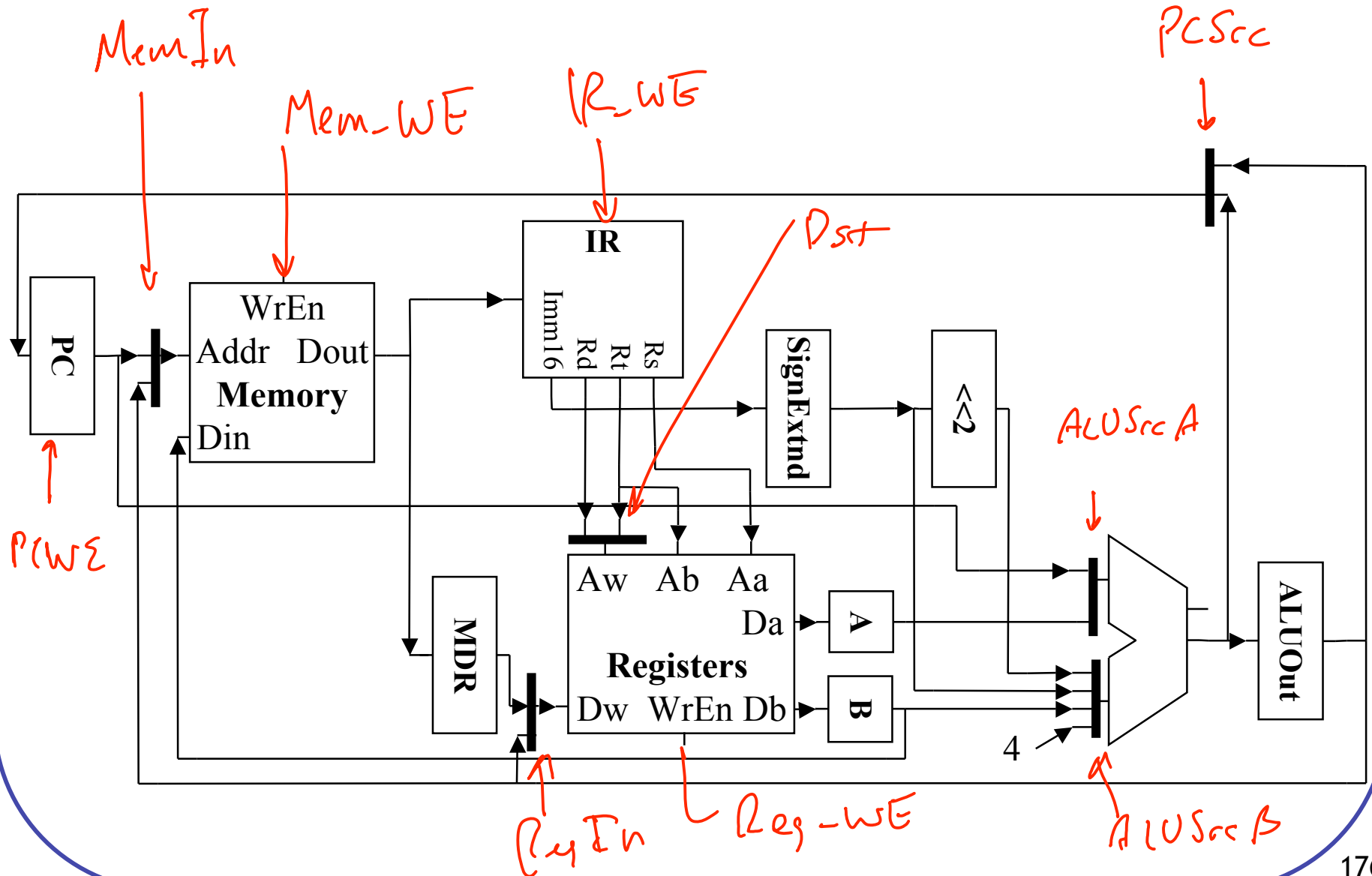
Cycle 5 Load: Reg[Rt] = MDR

# Control FSM

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



# Datapath Control Signals



# Multicycle Datapath Control

Stat	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:  
 $A = Reg[Rs]$   
 $B = Reg[Rt]$   
 $ALUOut = PC + (SE(imm16) \ll 2)$

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

3Rt:  
 $ALUout = A \text{ op } B$

4Rt:  
 $Reg[Rd] = ALUout$

3St:  
 $ALUout = A + SE(Imm16)$

4St:  
 $Mem[ALUout] = B$

3Lo:  
 $ALUout = A + SE(Imm16)$

4Lo:  
 $MDR = Mem[ALUout]$

5Lo:  
 $Reg[Rt] = MDR$

# Multicycle Datapath Control

e	WE				ALU			Dest	MemIn	RegIn	PCSrc
	PC	Mem	Reg	IR	SrcA	SrcB	Op				
	1	0	0	1	PC	4	+	X	PC	X	ALU
	0	0	0	0	PC	<<2	+	X	X	X	X
r	(zero)	0	0	0	A	B	-	X	X	X	ALUout
t	0	0	0	0	A	B	(IR)	X	X	X	X
t	0	0	1	0	X	X	X	Rd	X	ALUout	X
t	0	0	0	0	A	SE	+	X	X	X	X
t	0	1	0	0	X	X	X	X	ALUout	X	X
o	0	0	0	0	A	SE	+	X	X	X	X
o	0	0	0	0	X	X	X	X	ALUout	X	X
o	0	0	1	0	X	X	X	Rt	X	MDR	X
					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:  
 $A = Reg[Rs]$   
 $B = Reg[Rt]$   
 $ALUOut = PC$   
 $+ (SE(imn16) \ll 2)$

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

3Rt:  
 $ALUout = A \text{ op } B$

4Rt:  
 $Reg[Rd] = ALUout$

3St:  
 $ALUout = A + SE(Imm16)$

4St:  
 $Mem[ALUout] = B$

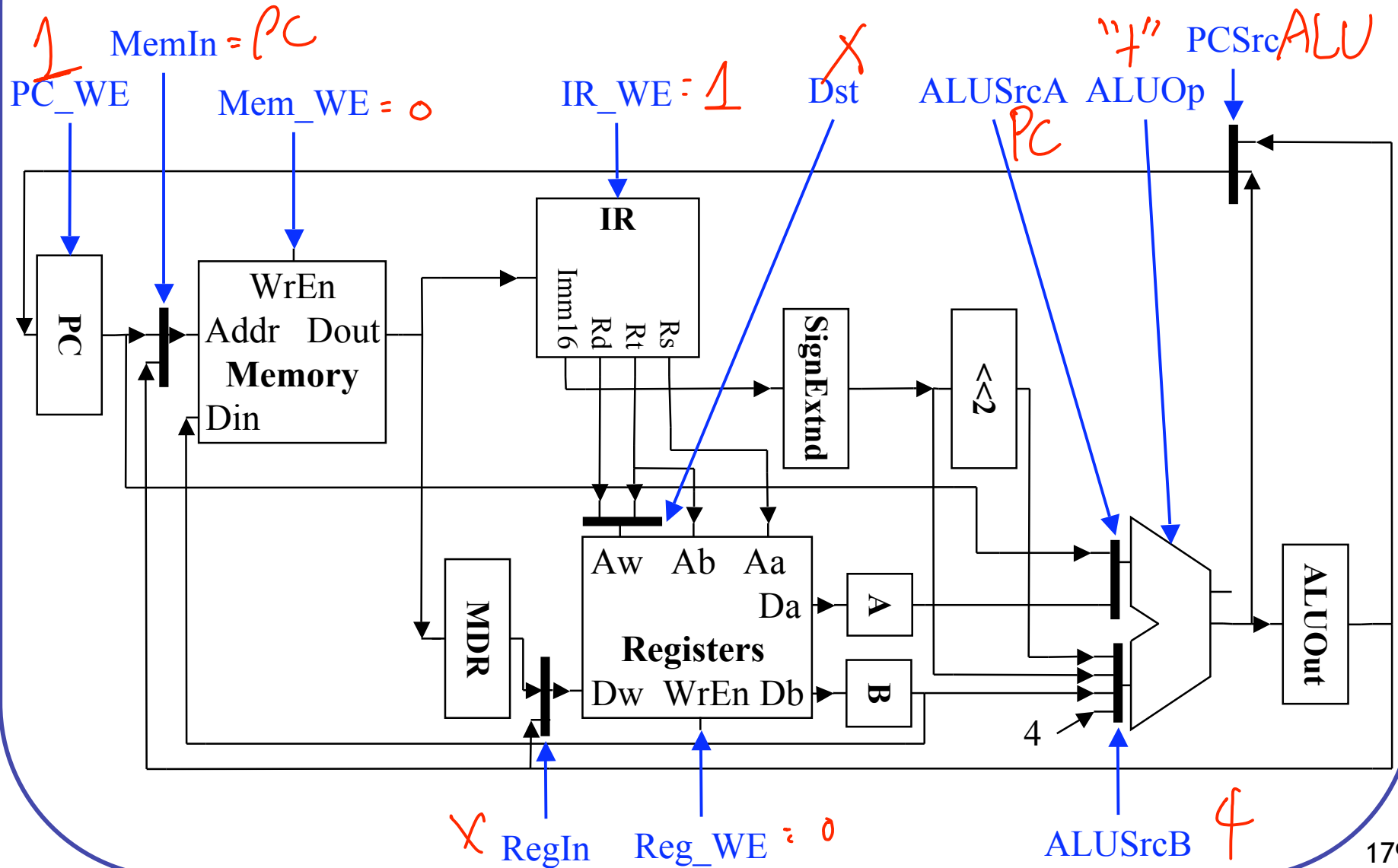
3Lo:  
 $ALUout = A + SE(Imm16)$

4Lo:  
 $MDR = Mem[ALUout]$

5Lo:  
 $Reg[Rt] = MDR$

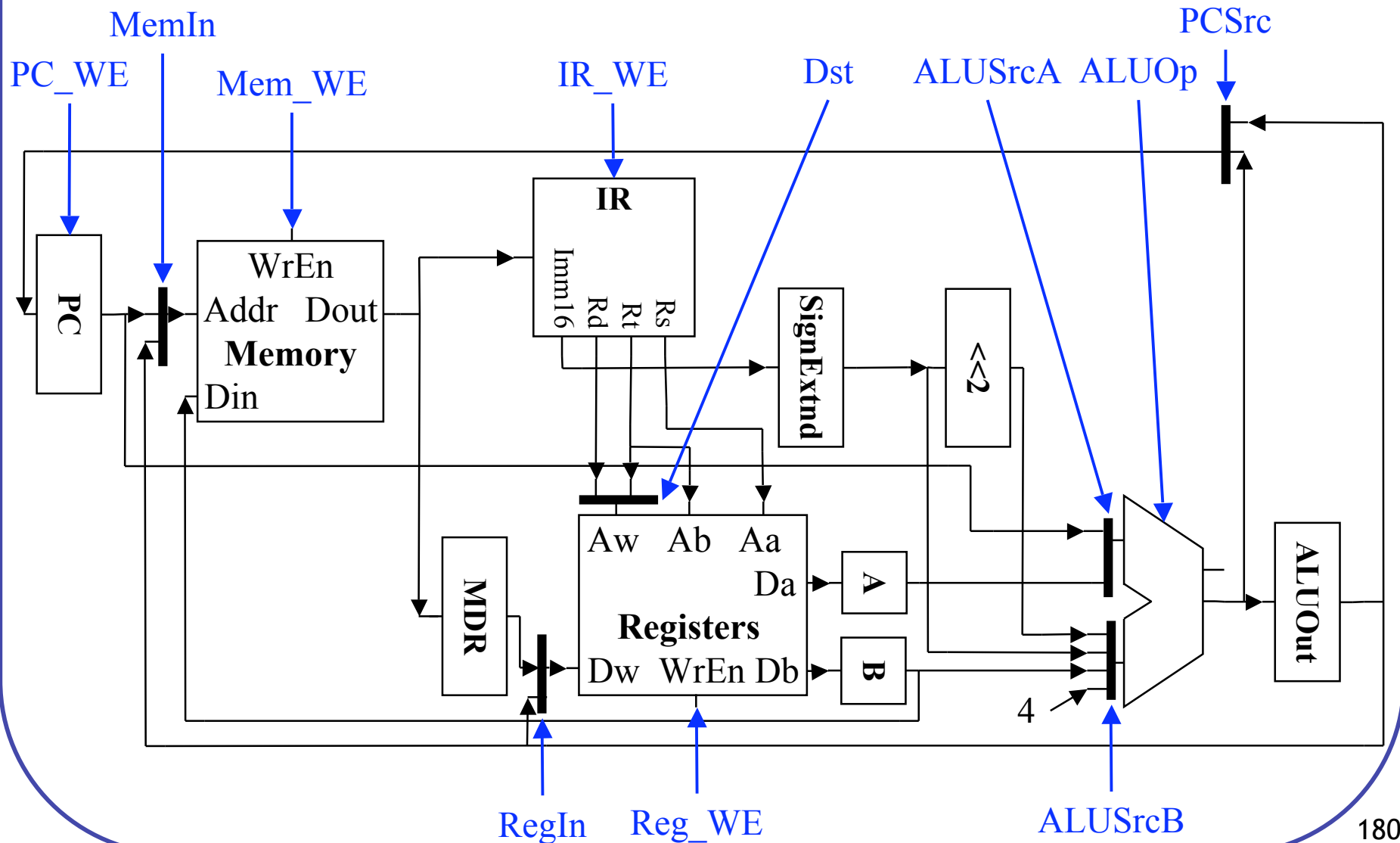
# Cycle 1 Control

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



# Cycle 2 Control

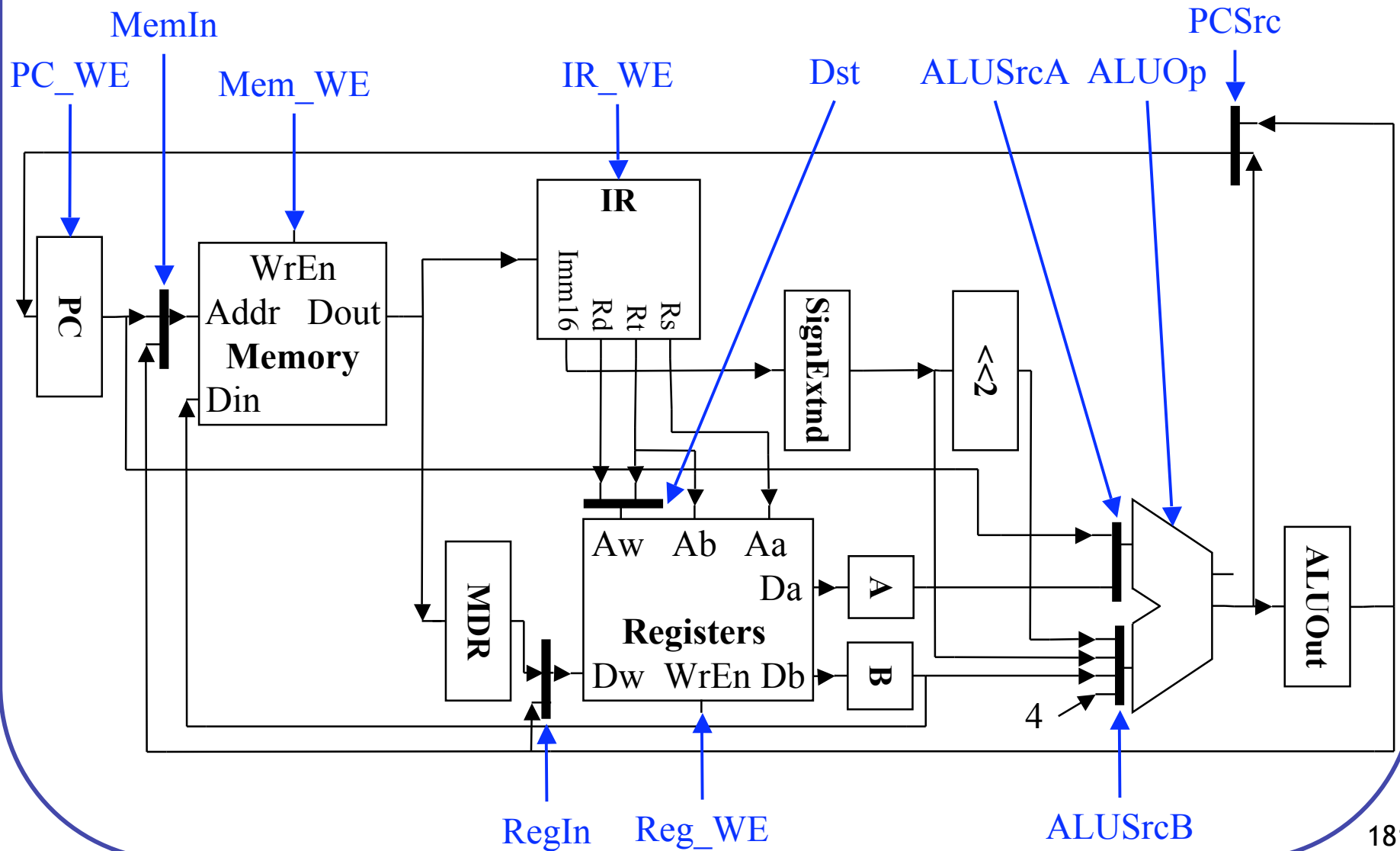
- $A = \text{Reg}[Rs]; B = \text{Reg}[Rt]; \text{ALUOut} = \text{PC} + (\text{Sign-extend}(\text{Imm16}) \ll 2)$





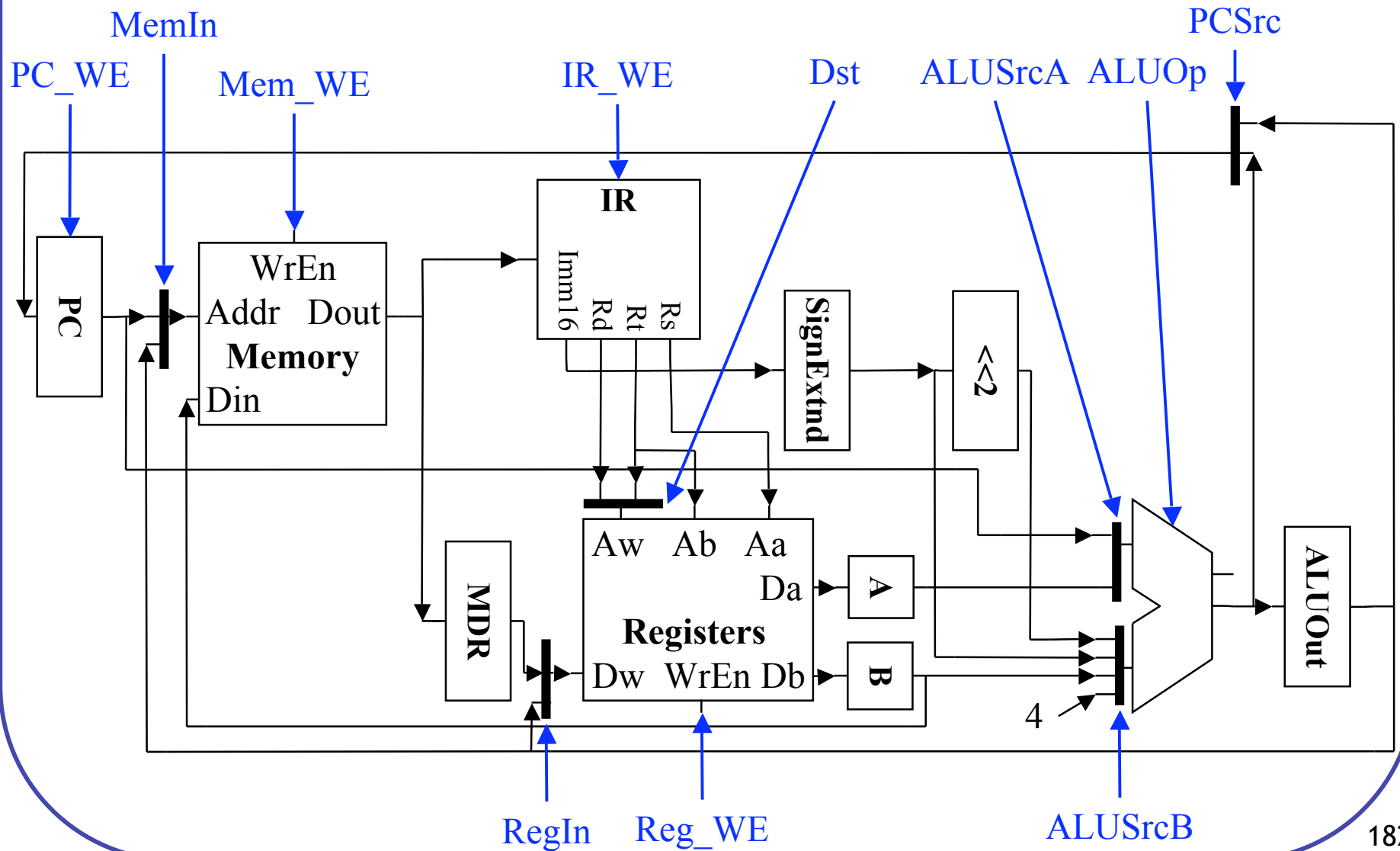
# Cycle 3 (Branch) Control

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



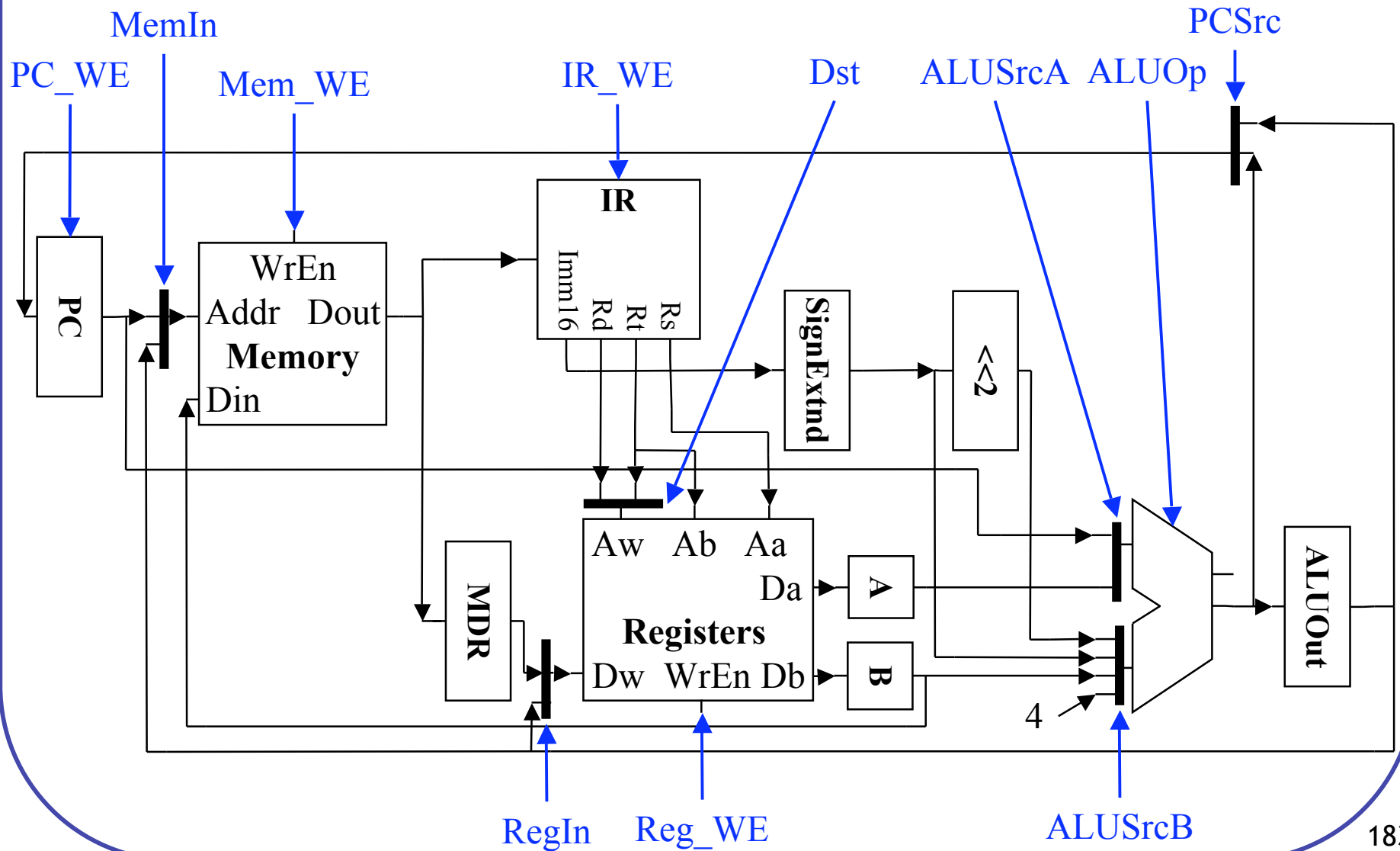
# Cycle 3 (R-Type) Control

- $ALU_{out} = A \text{ op } B$



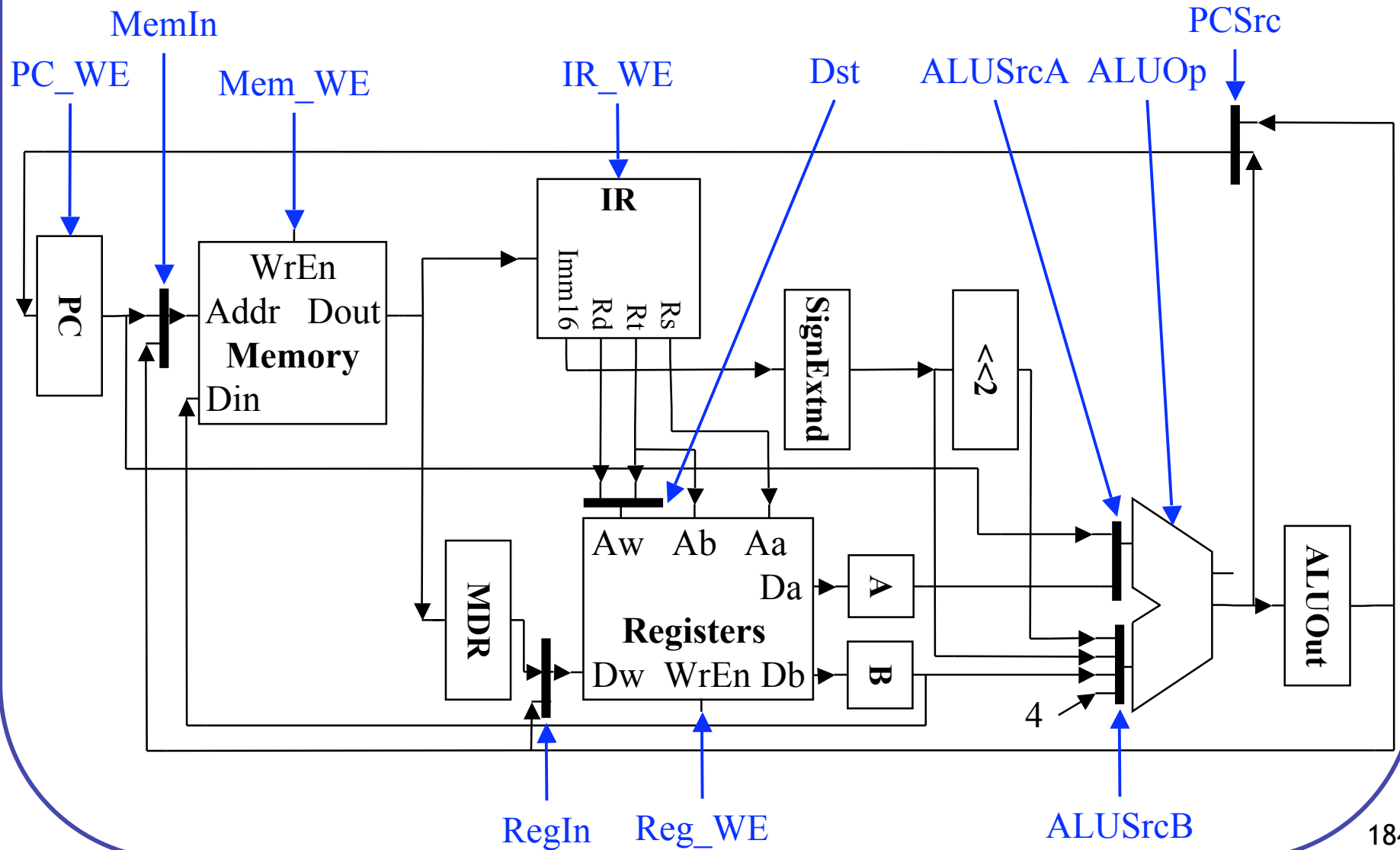
# Cycle 4 (R-Type) Control

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



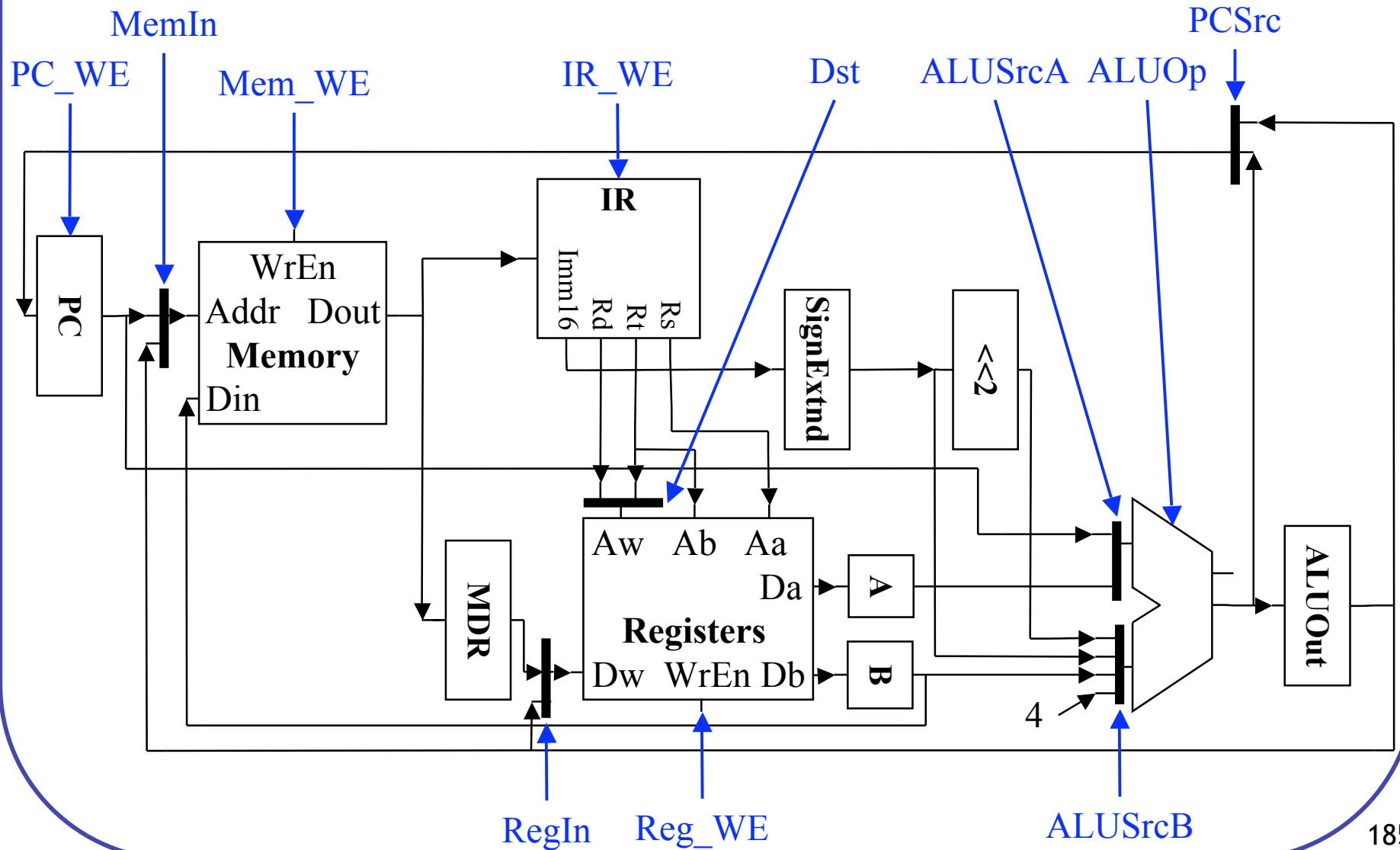
# Cycle 3 (Store) Control

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



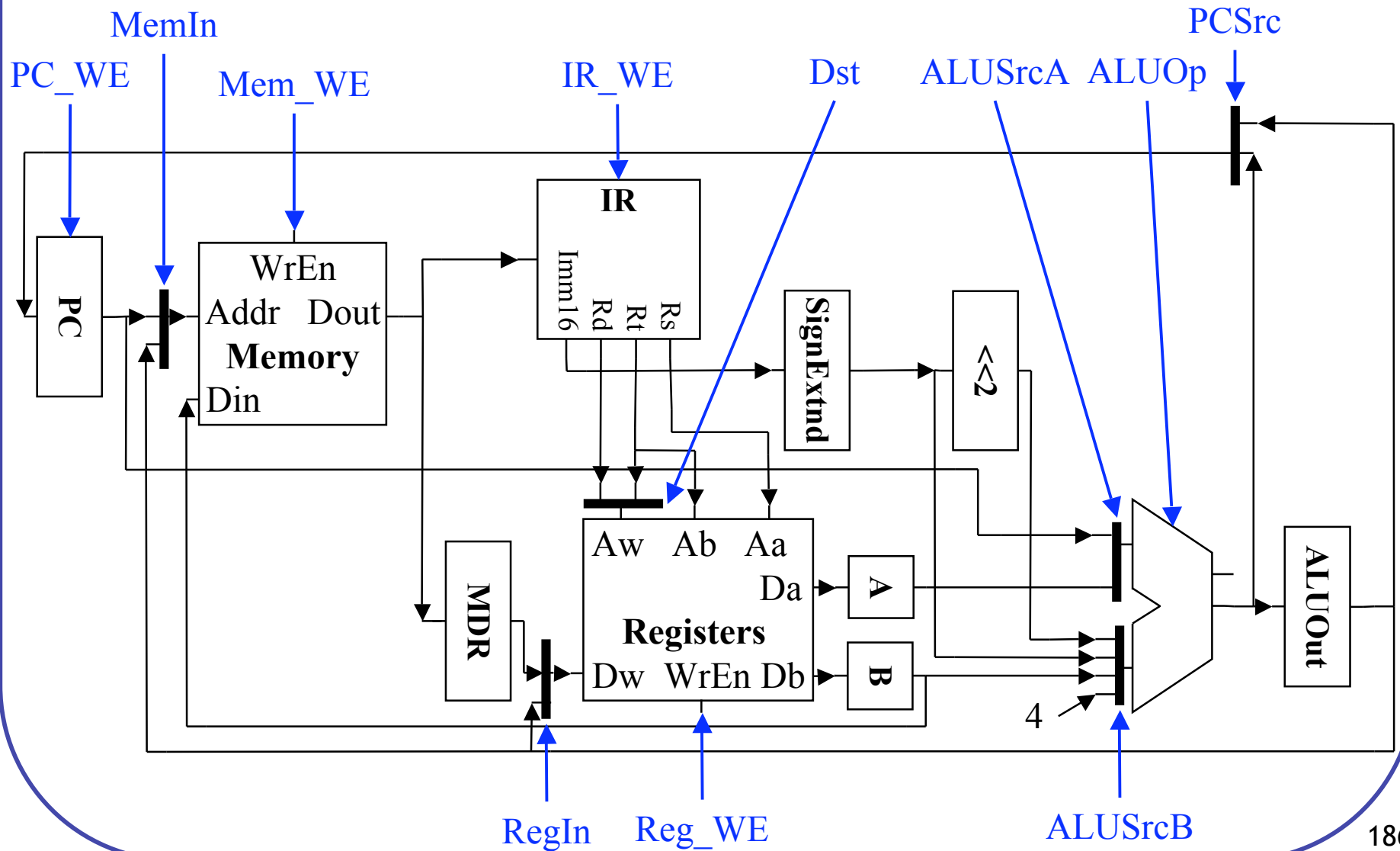
# Cycle 4 (Store) Control

- $\text{Mem}[\text{ALUout}] = \text{B}$



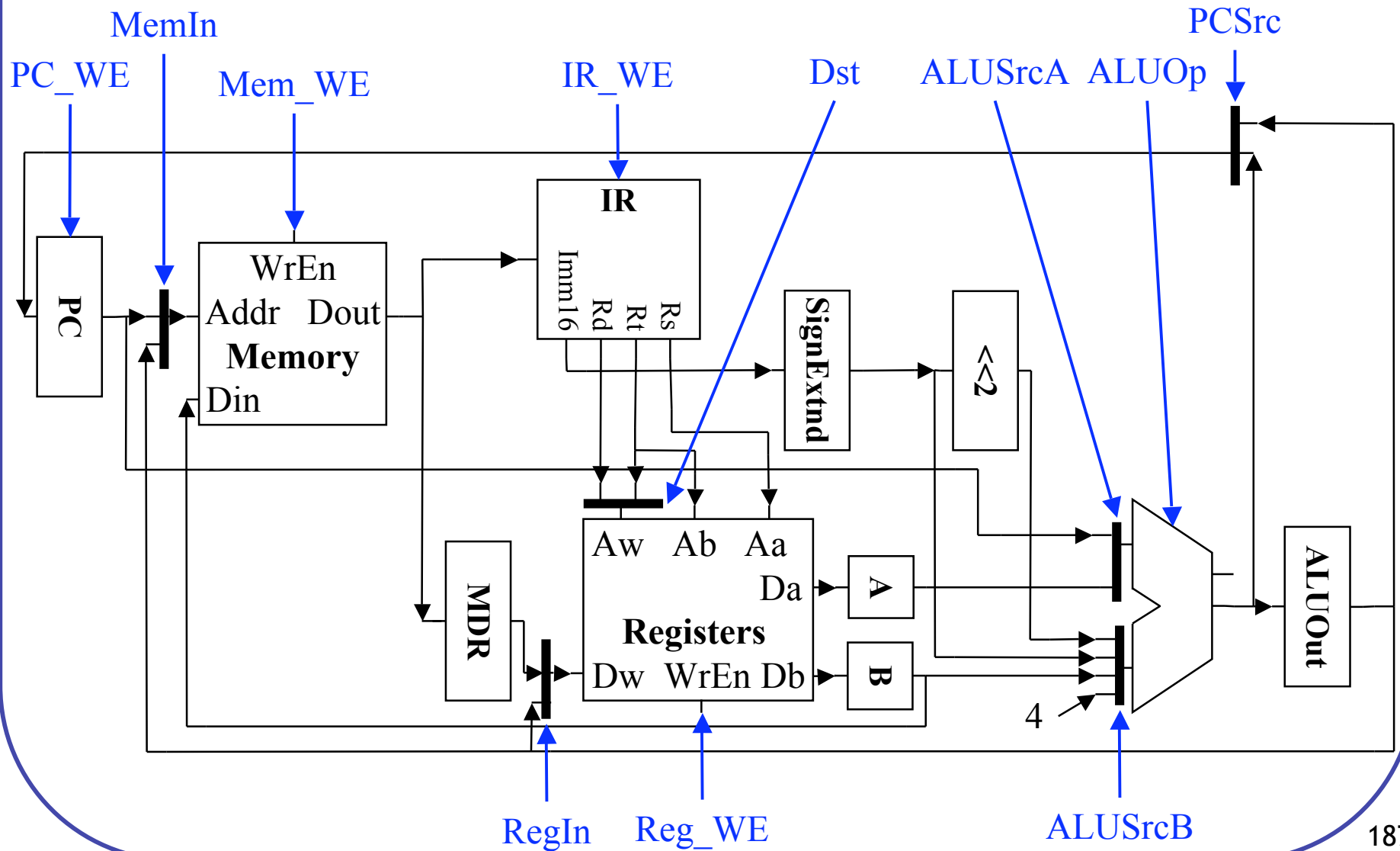
# Cycle 3 (Load) Control

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



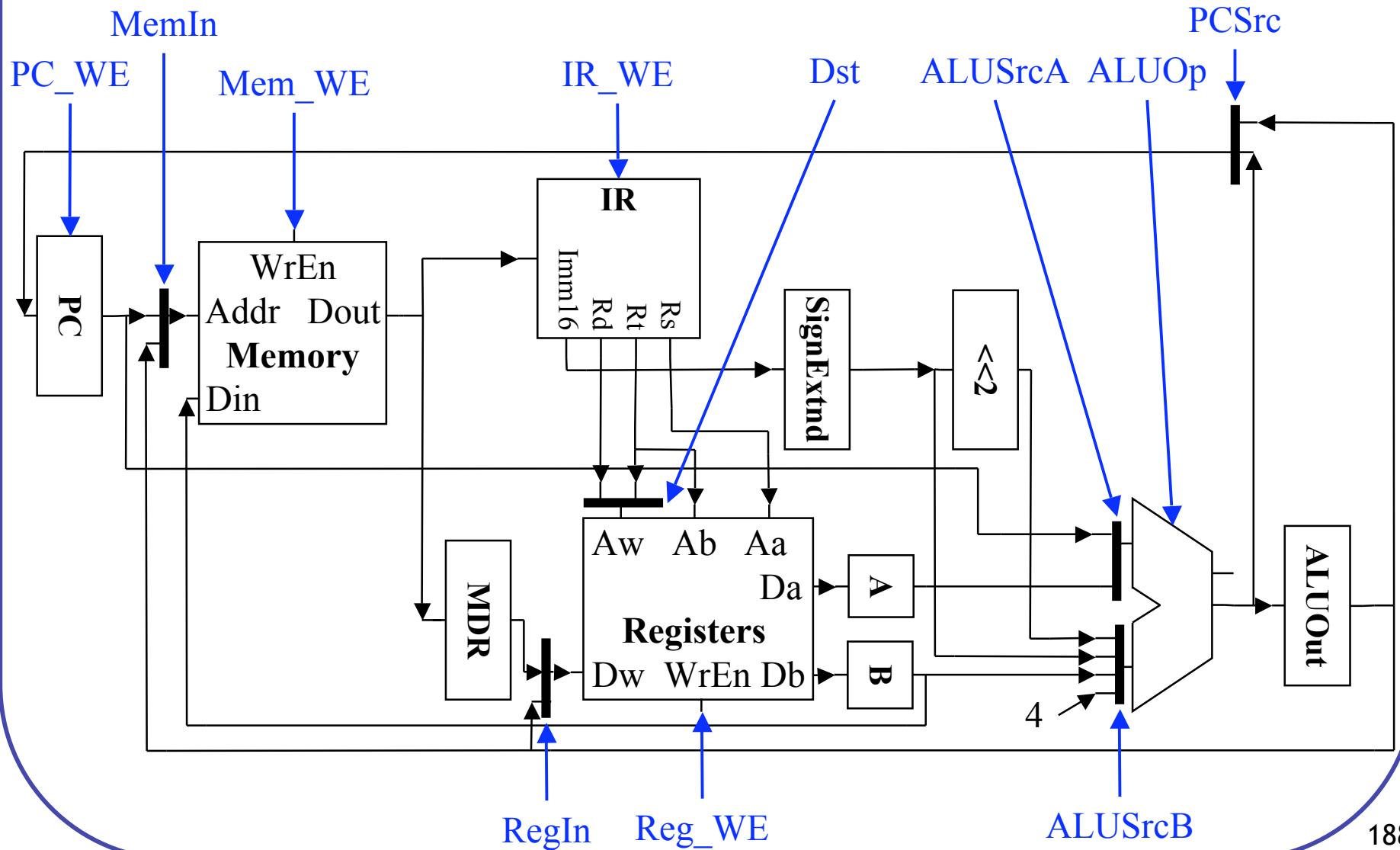
# Cycle 4 (Load) Control

- 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	4	50%	2.0
Load	5	20%	1.0
Store	4	10%	0.4
Branch	3	20%	0.6
CPI:			4.0

## Multicycle Summary

---

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

- Shorter ops = less time
- clock freq  $\uparrow$  ( $\rightarrow \times 4$  hopefully)
- More complex CPU
- Control = more complex
- More registers