CS161 – Design and Architecture of Computer Systems

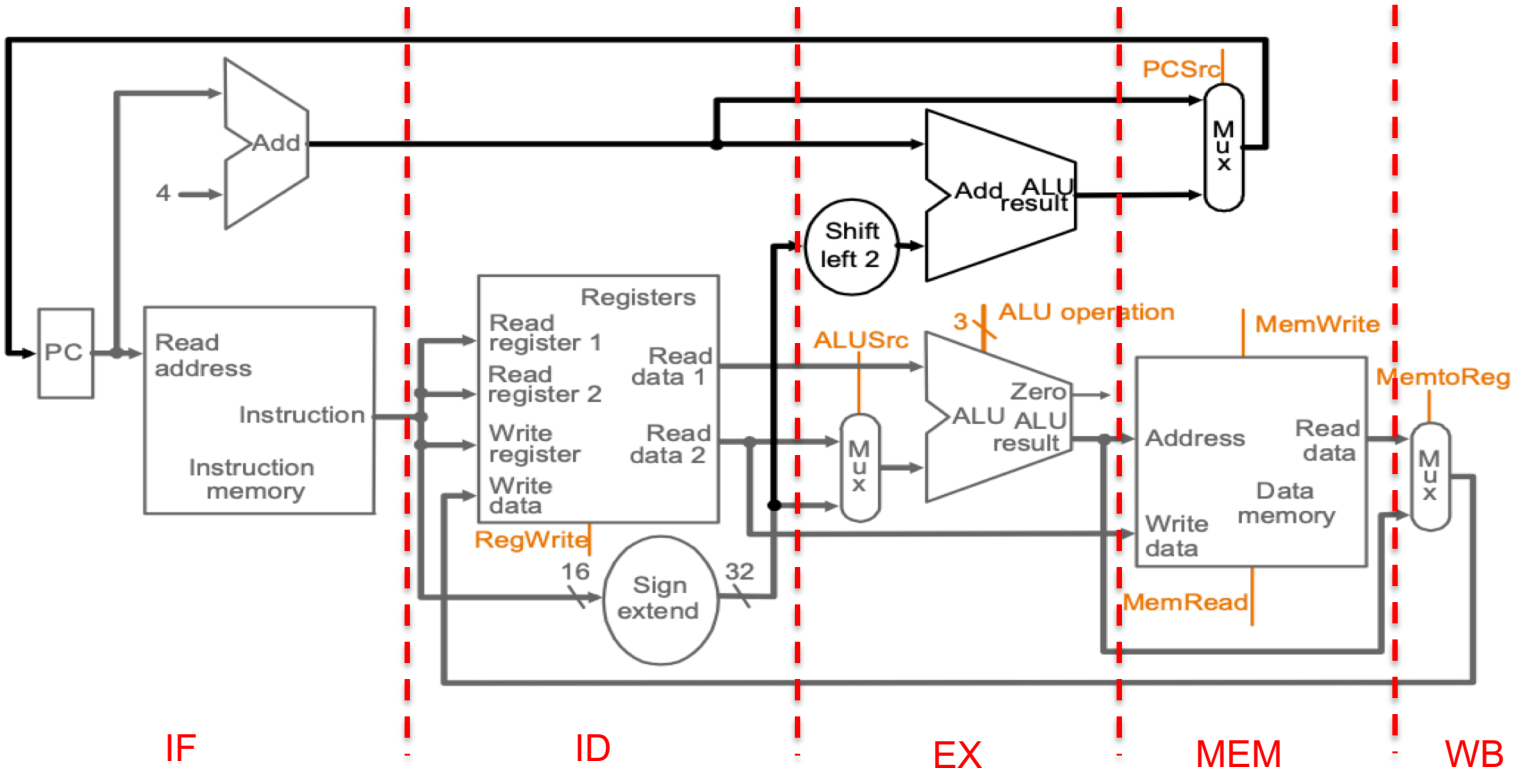
Week 3 – Discussion

*some slides adapted from:
Prof Daniel Wong UCR – EE/CS)

UNIVERSITY OF CALIFORNIA, RIVERSIDE

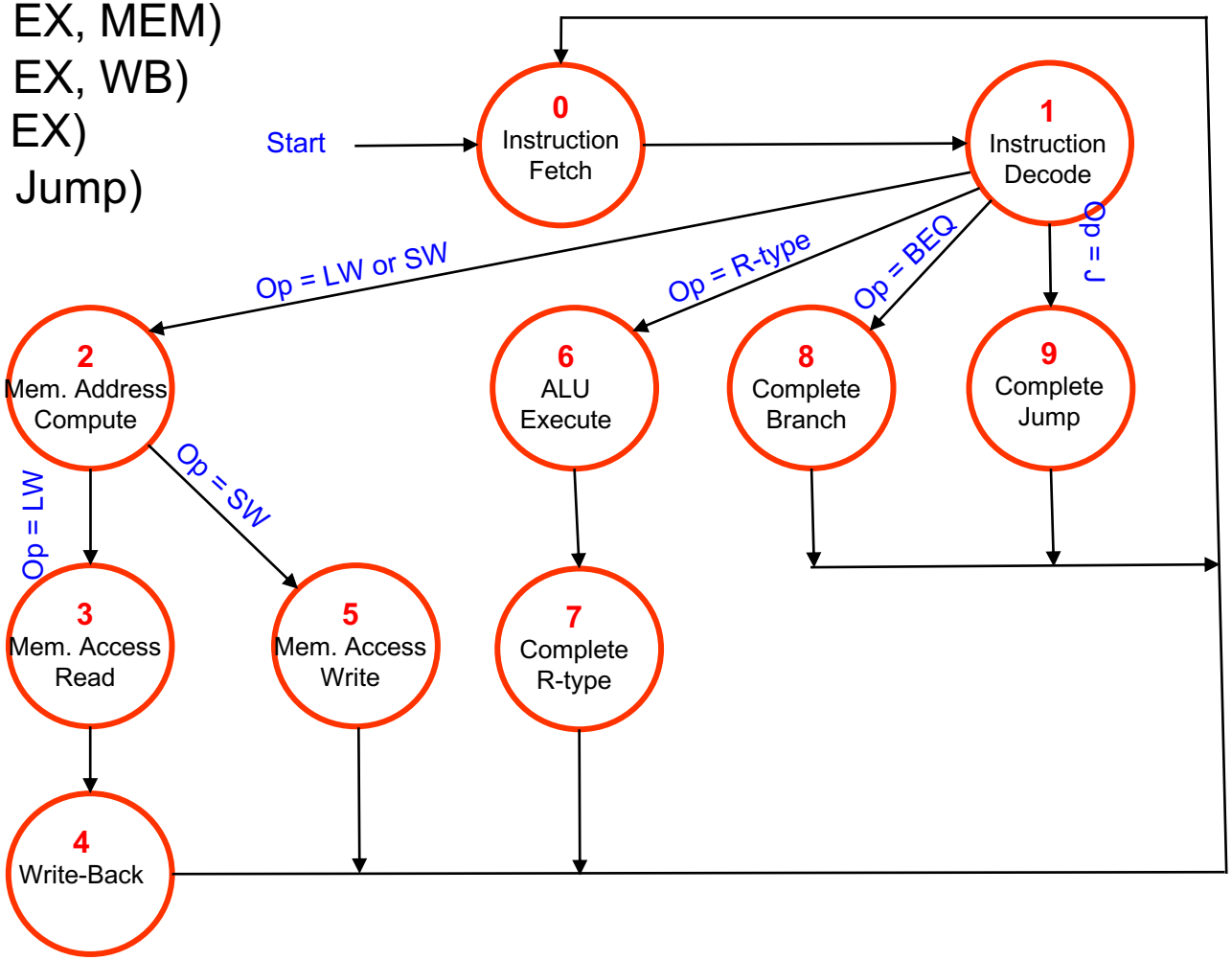
Multicycle

- Divides operations into steps
- One cycle per step
- Critical Path: longest functional unit per step (i.e., PC+4 vs I-Mem)

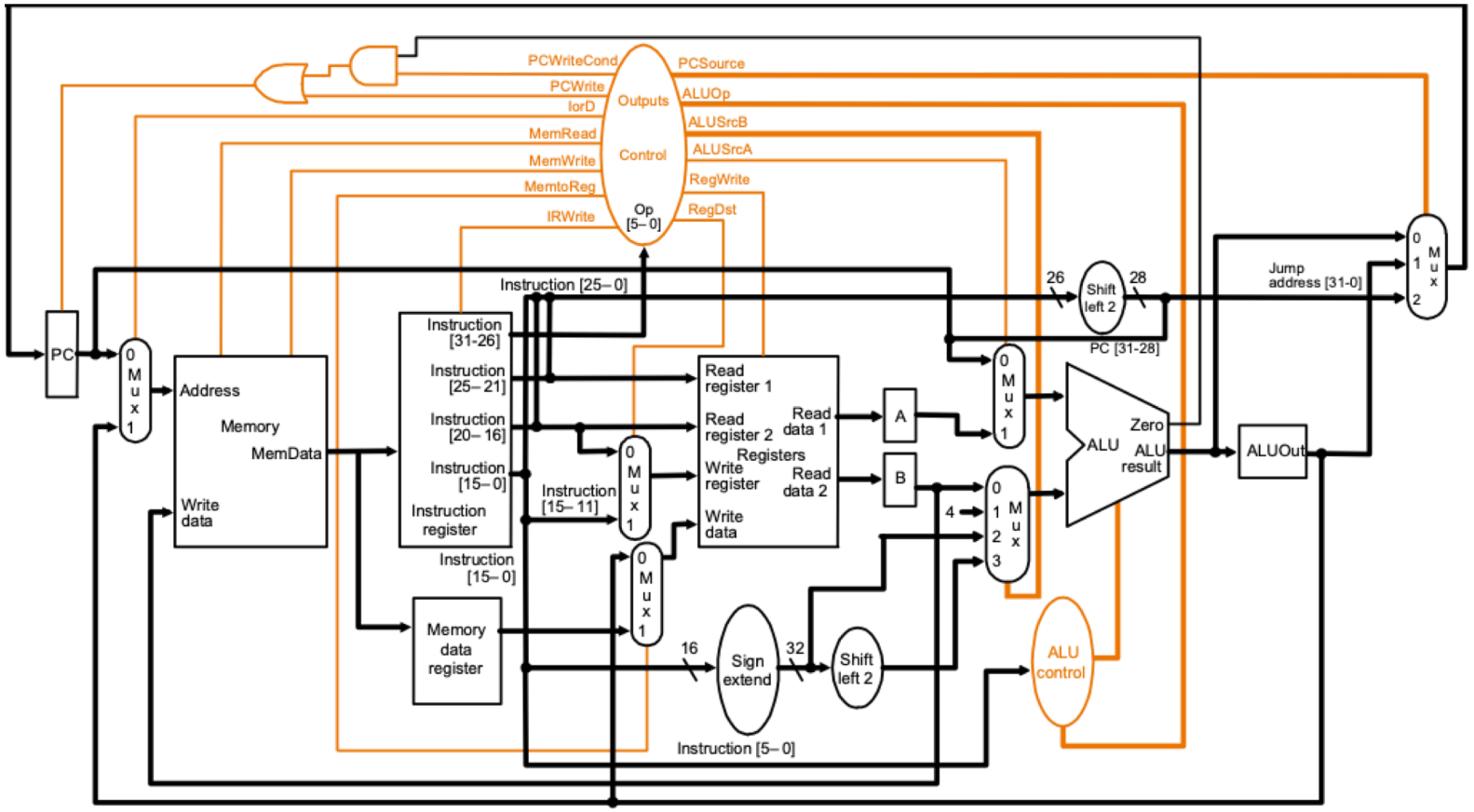


Multicycle

Load = 5 cycles (IF, ID, EX, MEM, WB)
Store = 4 cycles (IF, ID, EX, MEM)
R-type = 4 cycles (IF, ID, EX, WB)
Branch = 3 cycles (IF, ID, EX)
Jump = 3 cycles (IF, ID, Jump)



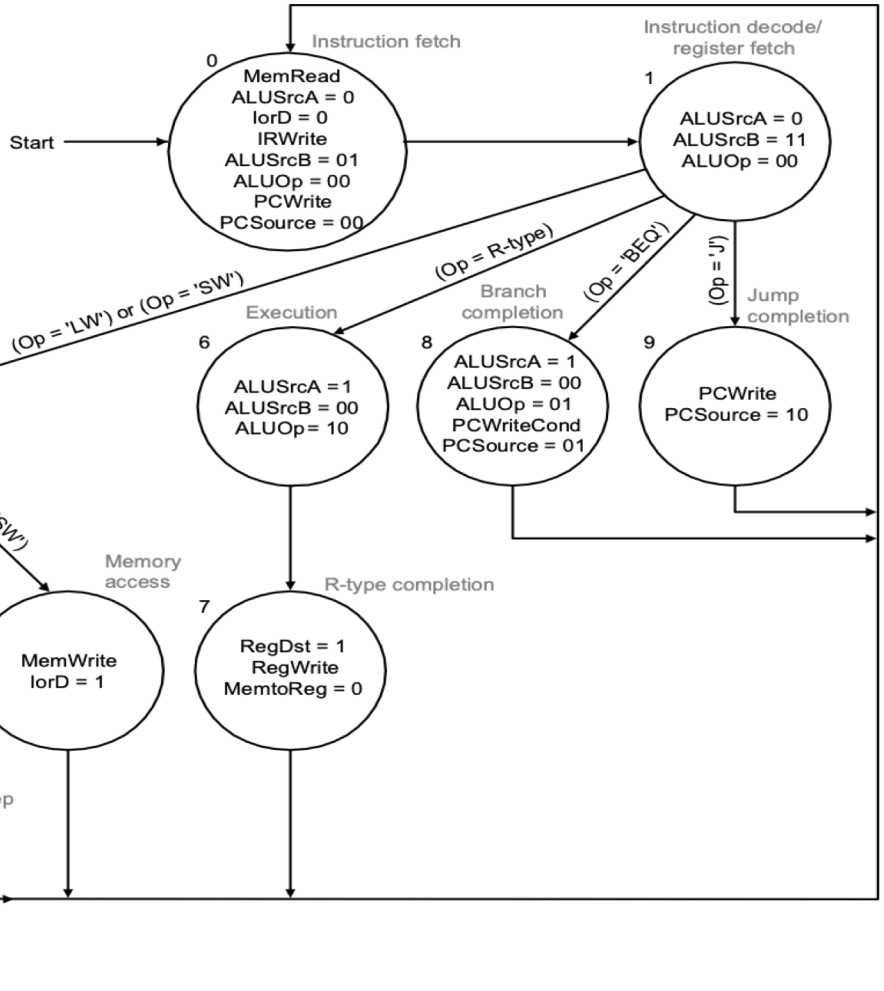
Control Signals



R-Type	LW	SW	BEQ	J	Jump	Branch	Reg Dst	ALU Src	Memto-Reg	Reg Write	Mem Read	Mem Write	ALU Op[1]	ALU Op[0]
1	0	0	0	0	0	0	1	0	0	1	0	0	1	0
0	1	0	0	0	0	0	0	1	1	1	1	0	0	0
0	0	1	0	0	0	0	X	1	X	0	0	1	0	0
0	0	0	1	0	0	1	X	0	X	0	0	0	0	1
0	0	0	0	1	1	0	X	X	X	0	0	0	X	X

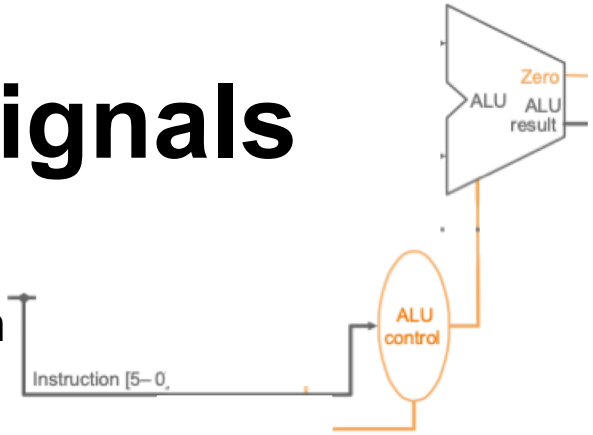
Multicycle

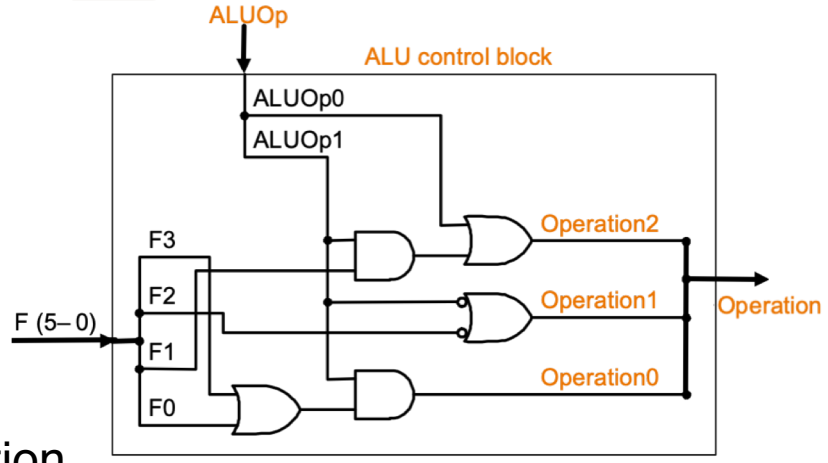
Load = 5 cycles (IF, ID, EX, MEM, WB)
Store = 4 cycles (IF, ID, EX, MEM)
R-type = 4 cycles (IF, ID, EX, WB)
Branch = 3 cycles (IF, ID, EX)
Jump = 3 cycles (IF, ID, Jump)



ALU Control Signals

- ALU Control input
  - function field of the instruction
  - 2-bit control field (ALUOp)
- ALUOp indicates operation
  - (00) add - for loads and stores
  - (01) subtract - for beq
  - (10) determined by funct field
- ALU Control output:
  - 4-bit ALU control signal lines
  - Note: Operation3 used only for NOR function





ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

Control Signal Example: SWAP

SWAP

Instruction: Swap Rs, Rt

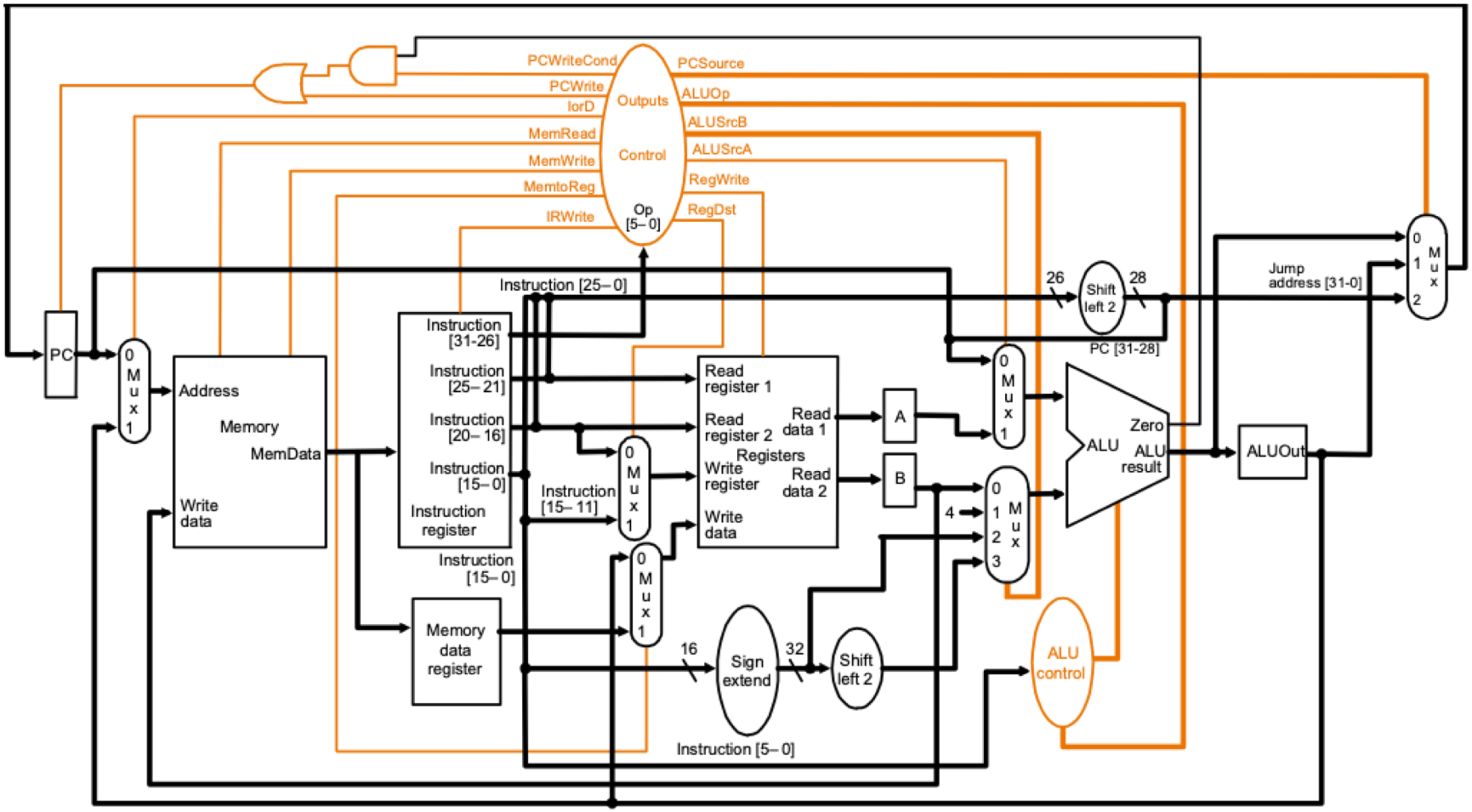
Assume r-type instruction where Rs is in Rd field

Swap Rs, Rs, Rt

Interpretation: Reg[Rs] = Reg[Rt]

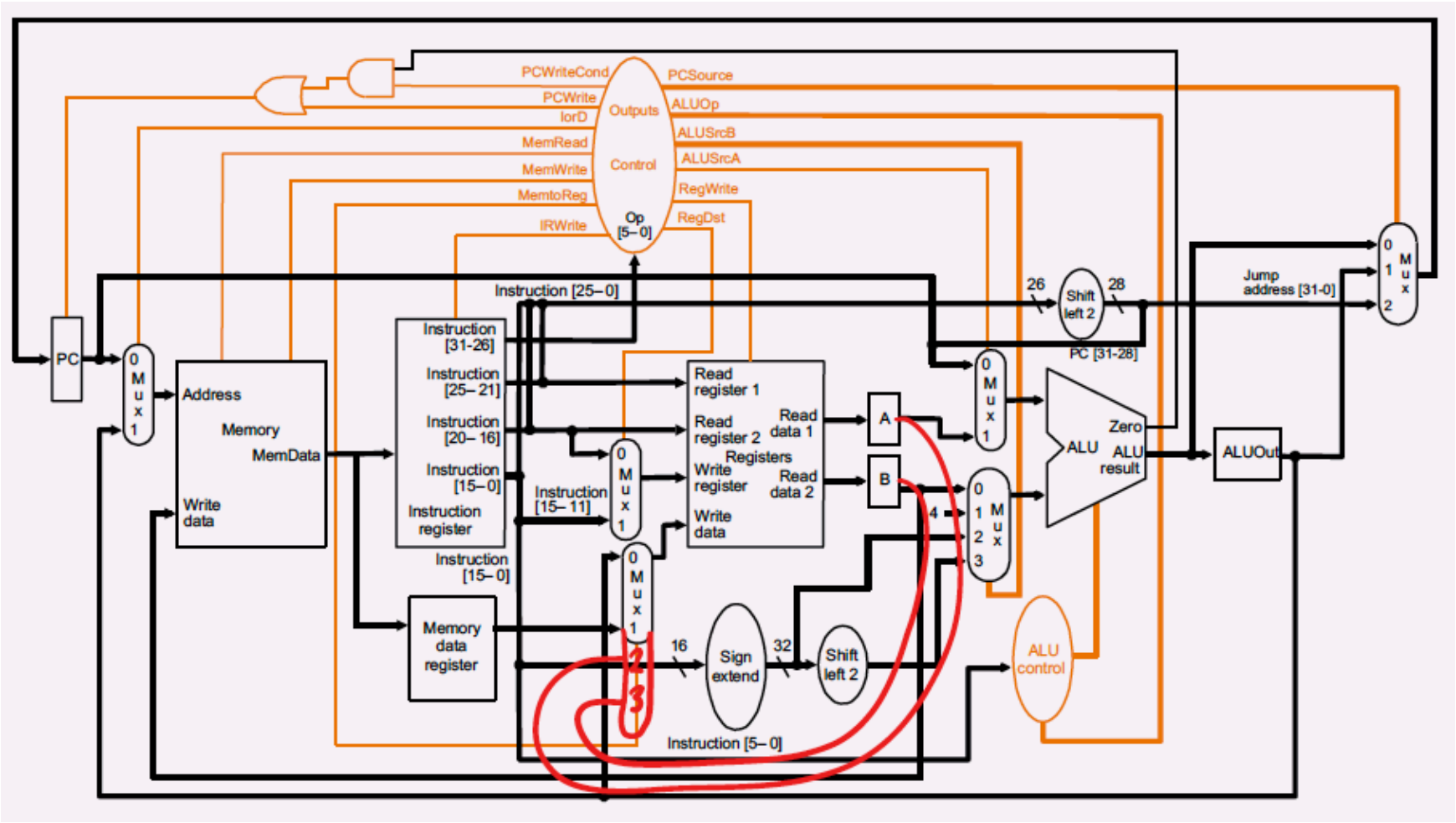
Reg[Rt] = Reg[Rs]

Control Signal Example: SWAP

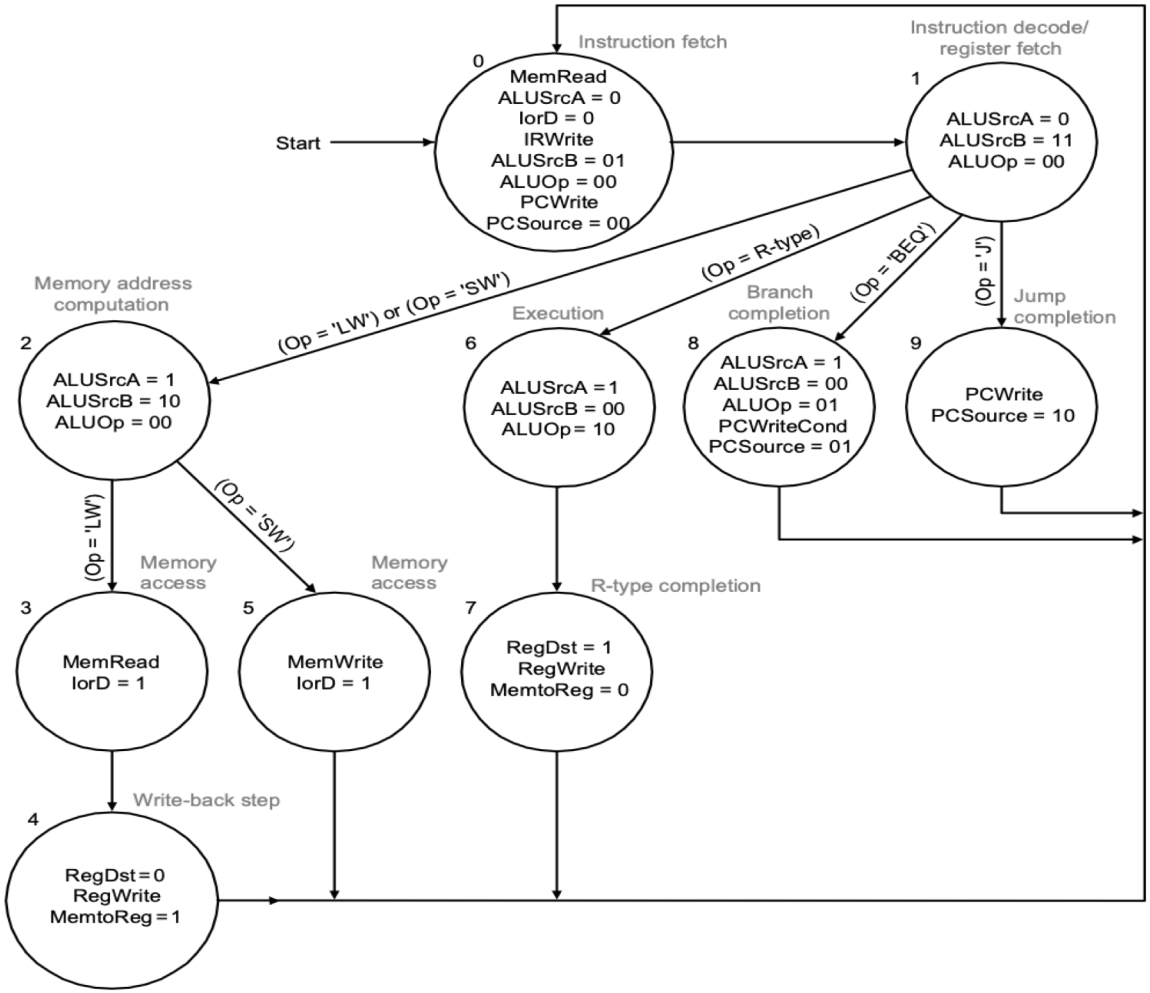


- Add any necessary changes to datapath

Control Signal Example: SWAP

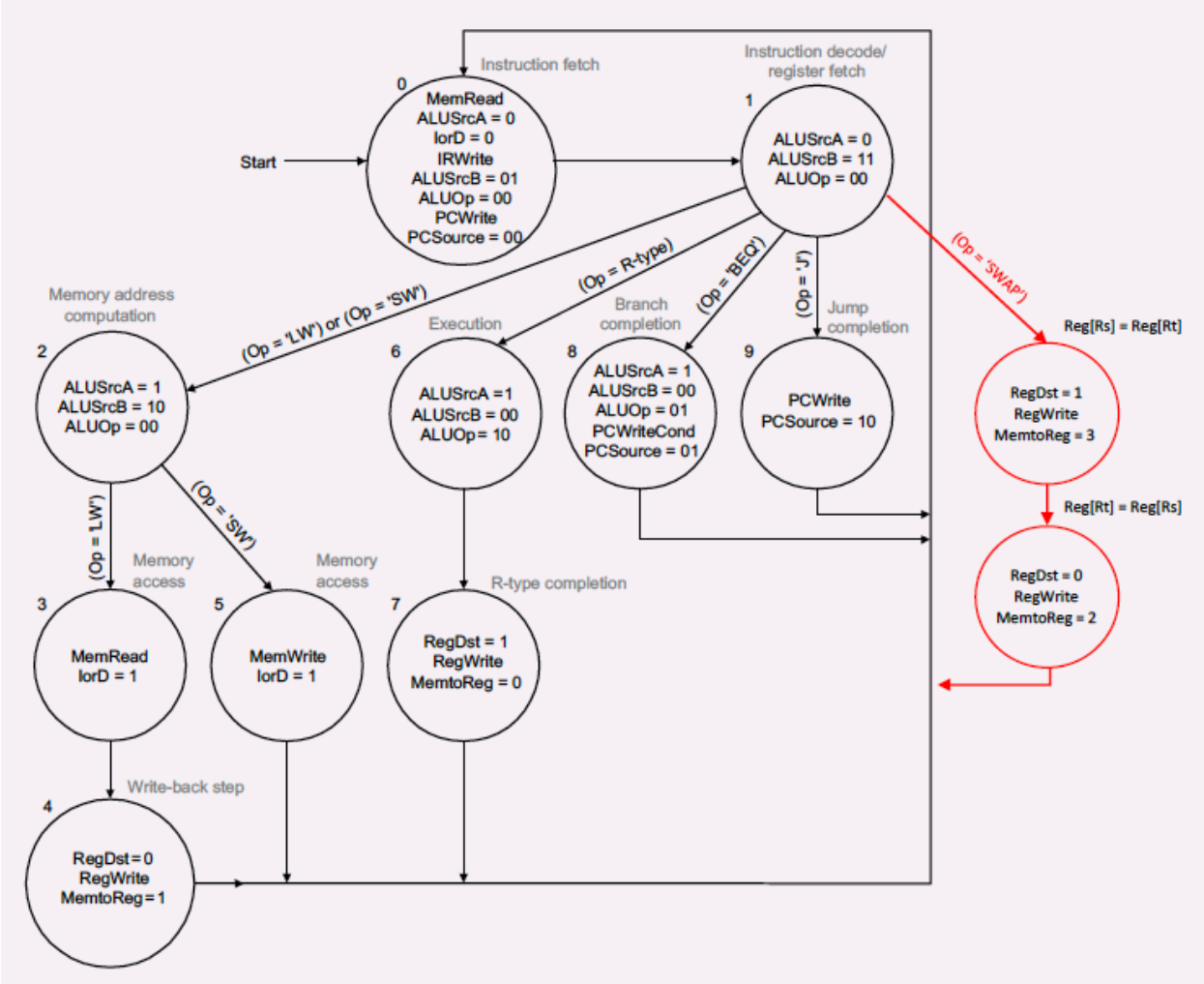


Control Signal Example: SWAP

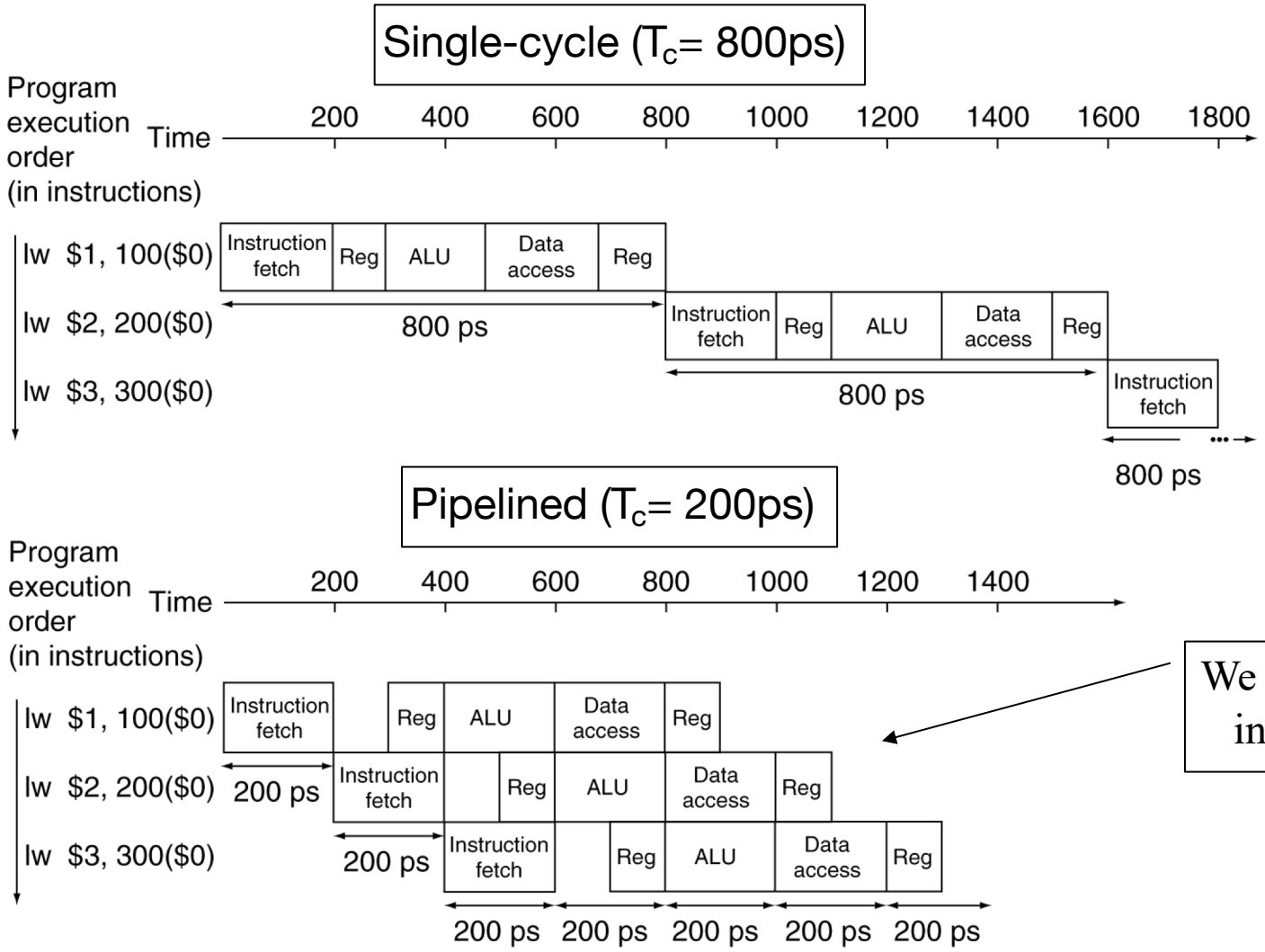


- Add any necessary changes to control signals

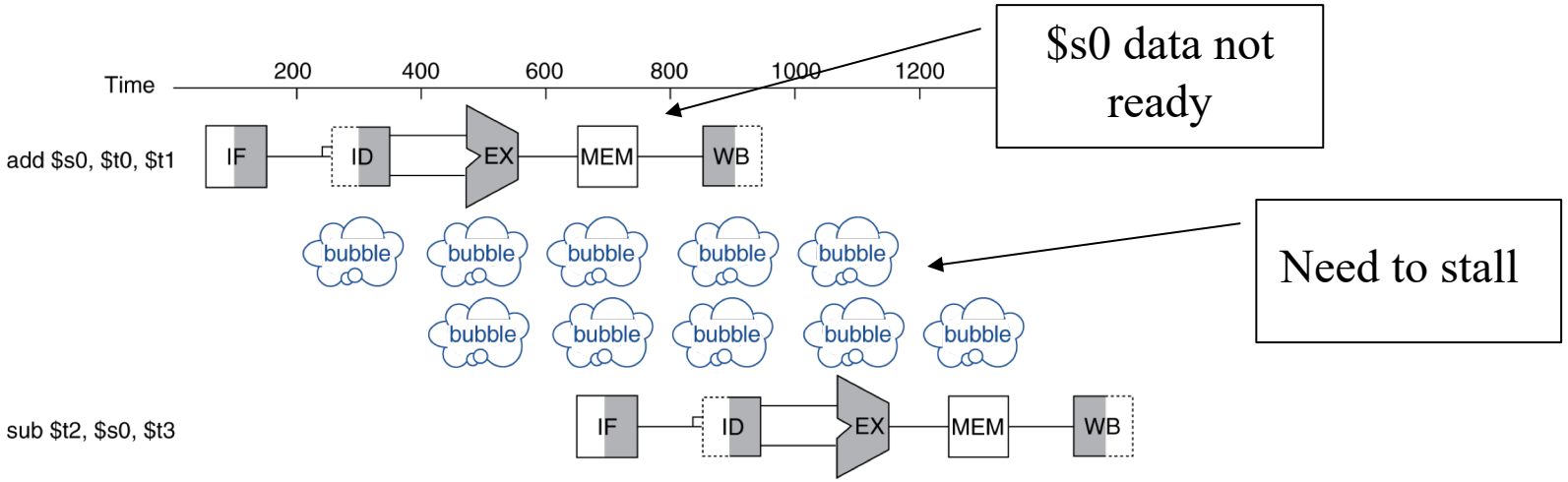
Control Signal Example: SWAP



Multicycle → Pipeline

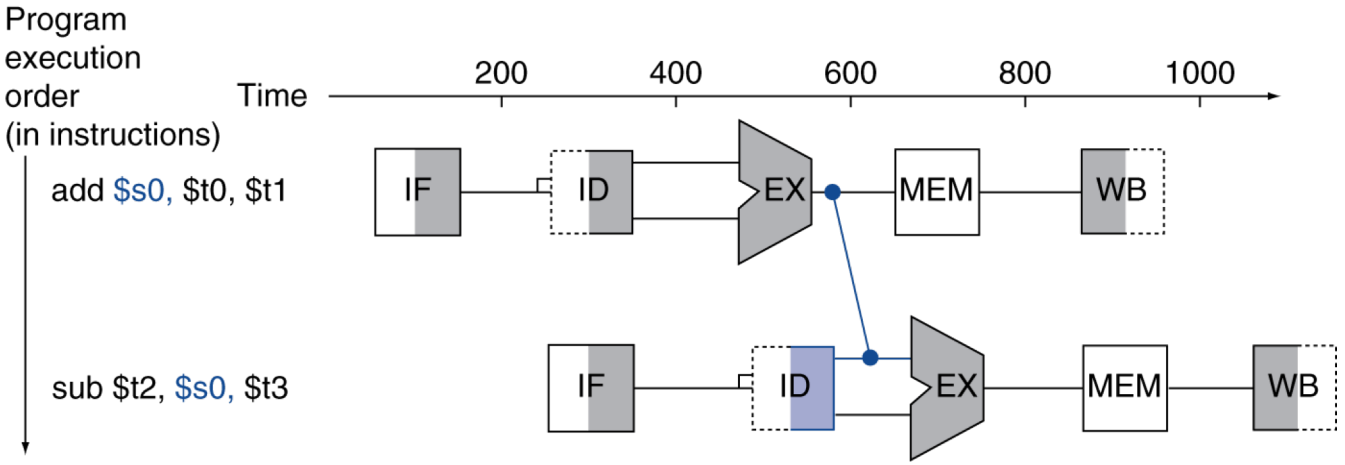


Problem: Account for Delays



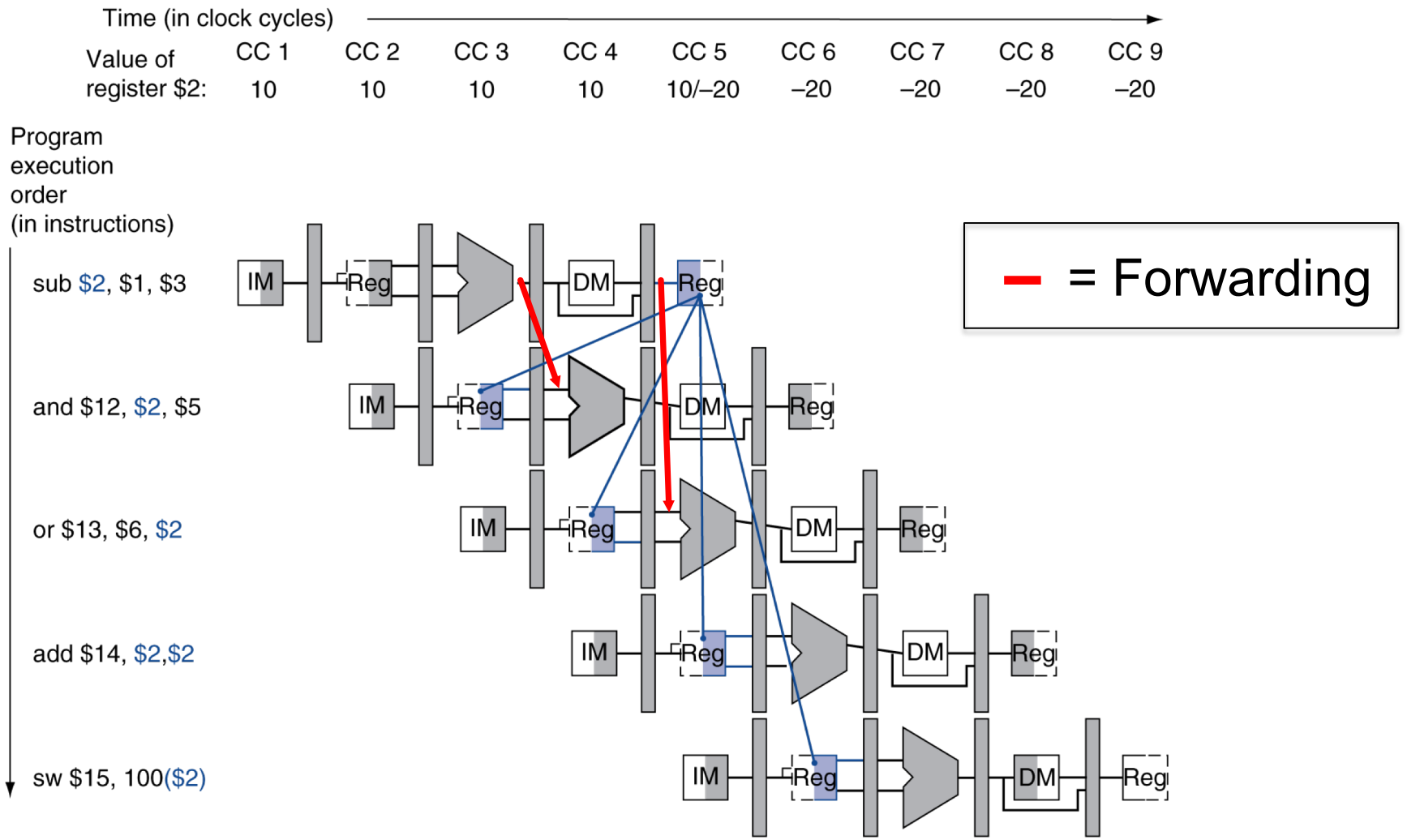
Solution: Forwarding

- Forwarding



  - Forward data as soon as it's available

When to Forward



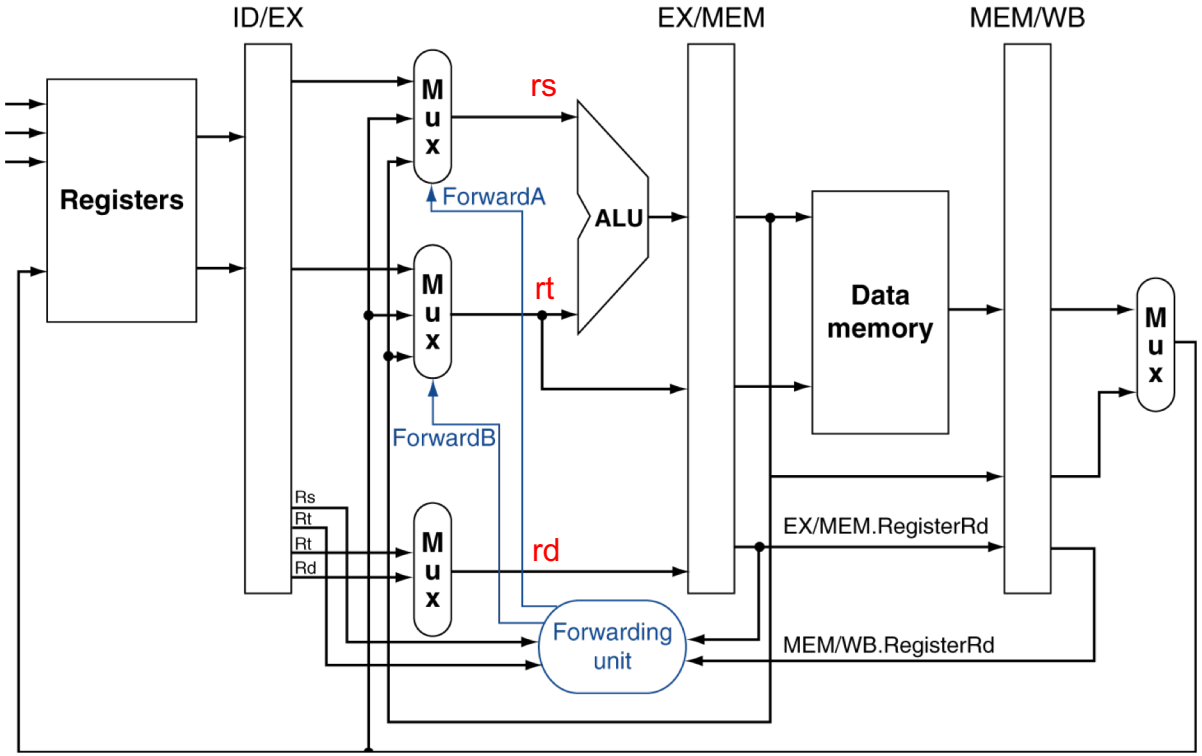
When to Forward

1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
1b. EX/MEM.RegisterRd = ID/EX.RegisterRt

If dest == either src register
Fwd from EX/MEM pipeline reg

2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

If dest == src register
Fwd from MEM/WB pipeline reg



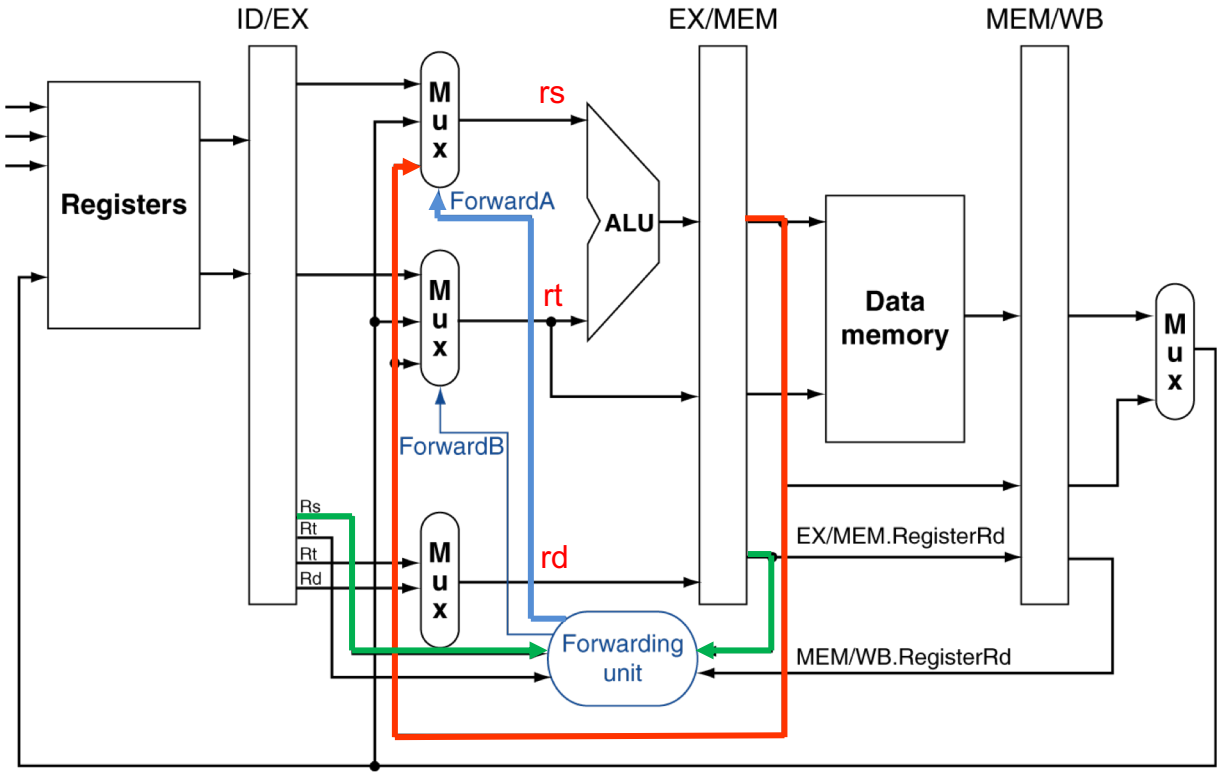
- But only if forwarding instruction will write to register
  - EX/MEM.RegWrite
  - MEM/WB.RegWrite

With Forwarding

- EX hazard
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)

    and (EX/MEM.RegisterRd = ID/EX.RegisterRs))

    ForwardA = 10 (where b10 = #2)

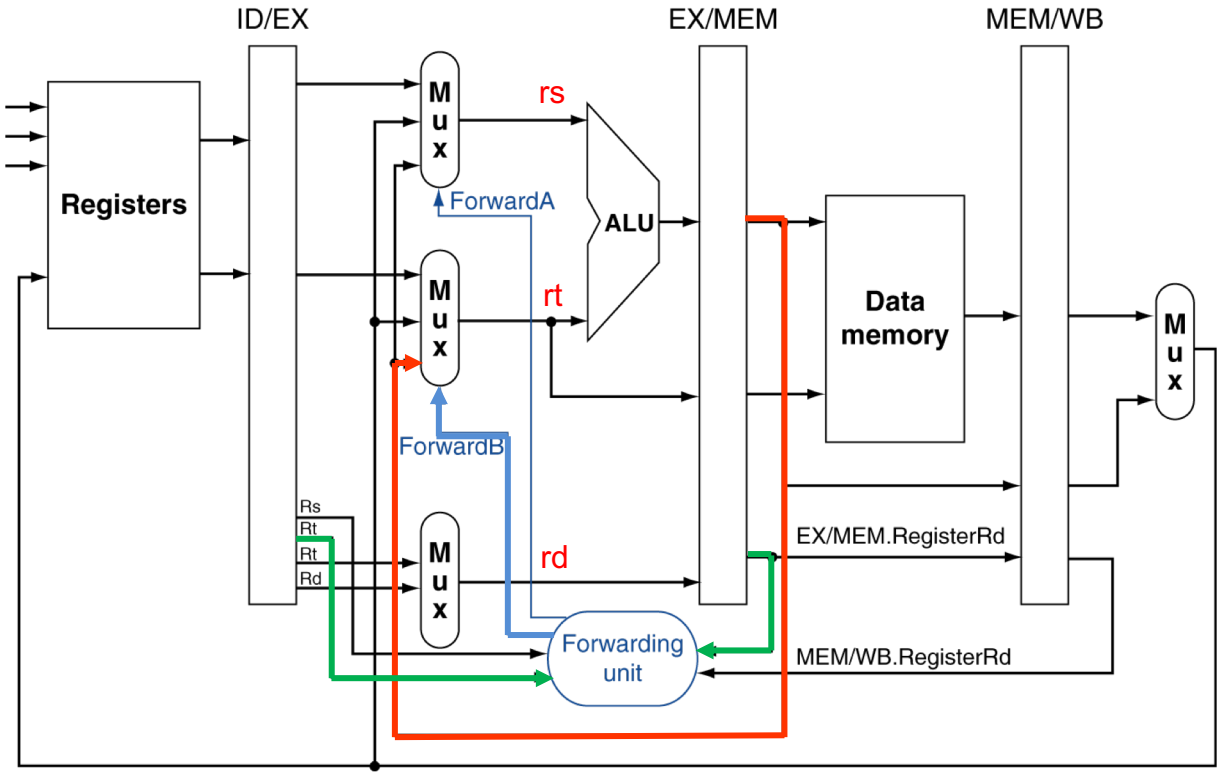


With Forwarding

- EX hazard
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)

    and (EX/MEM.RegisterRd = ID/EX.RegisterRt))

    ForwardB = 10 (where b10 = #2)

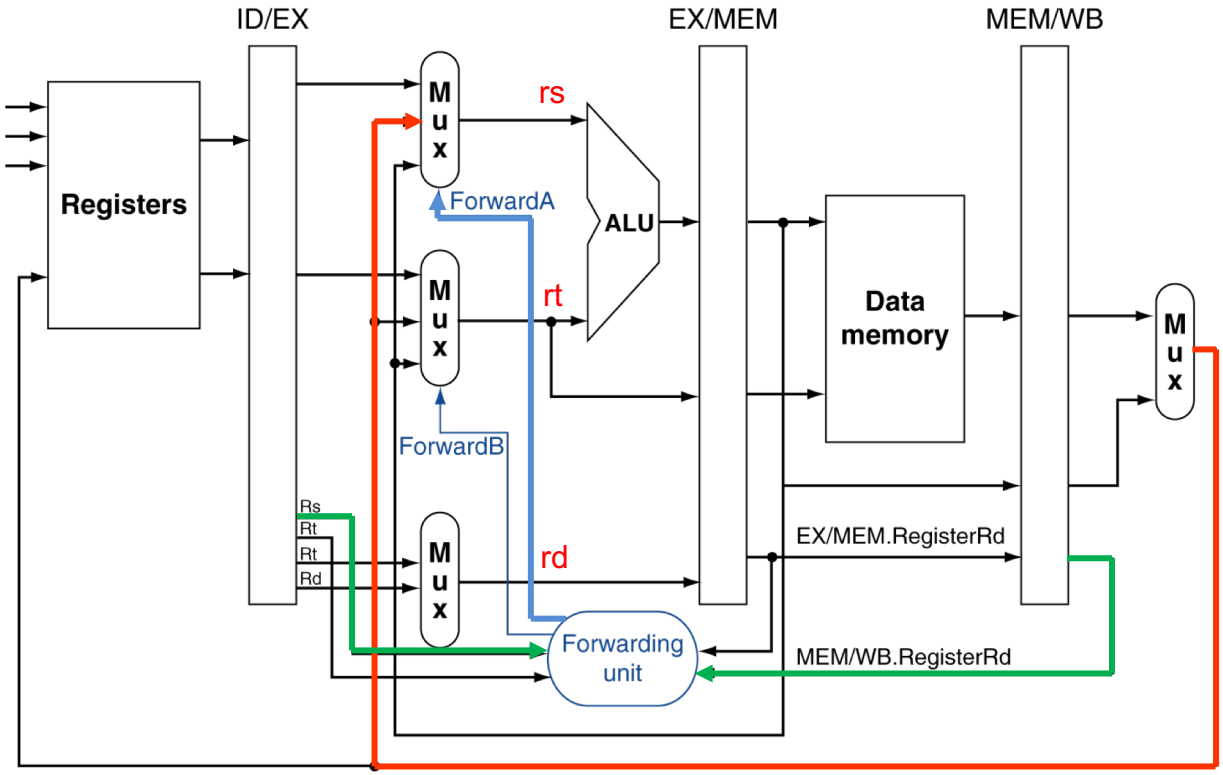


With Forwarding

- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)

    and (MEM/WB.RegisterRd = ID/EX.RegisterRs))

    ForwardA = 01 (where b01 = #1)

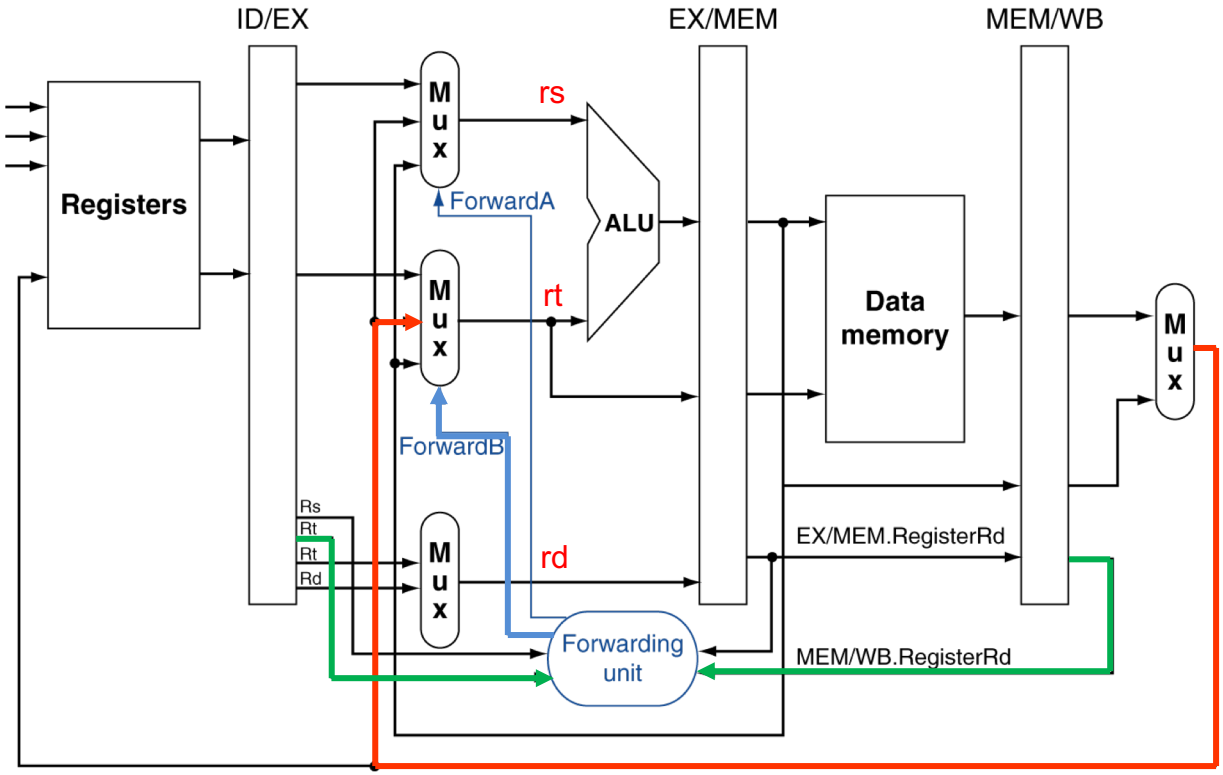


With Forwarding

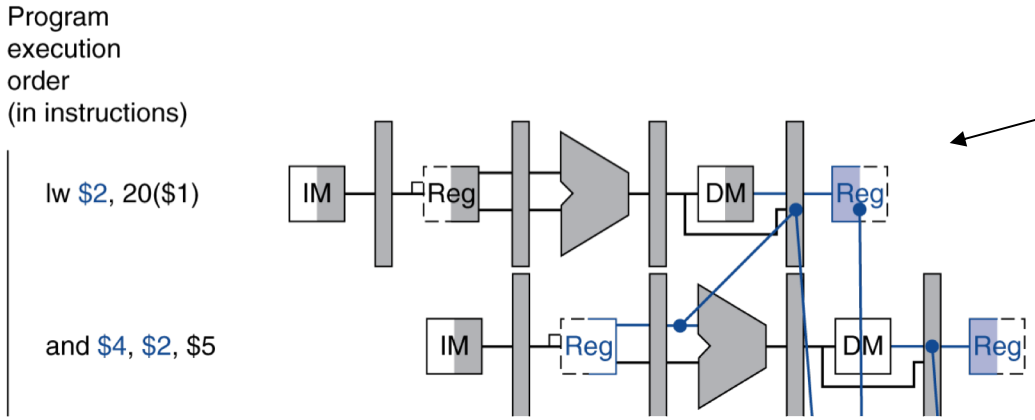
- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)

    and (MEM/WB.RegisterRd = ID/EX.RegisterRt))

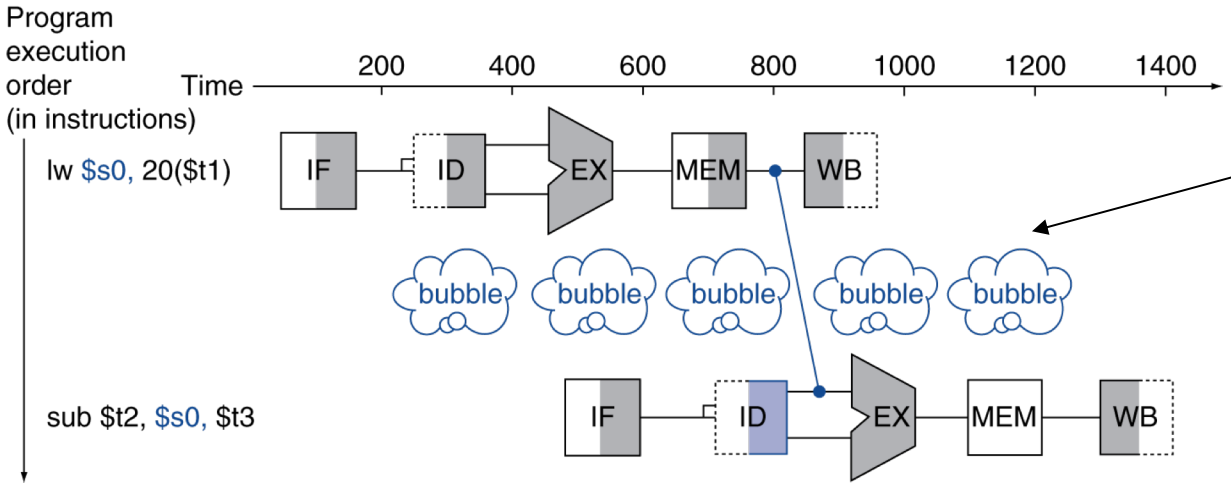
    ForwardB = 01 (where b01 = #1)



Still Can't Avoid All Stalls



Can't "forward" data backwards

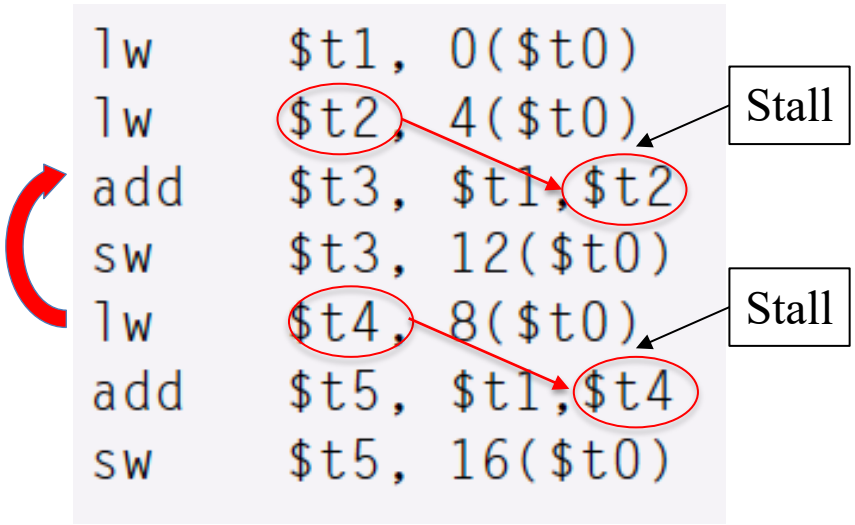


Still need to stall for one cycle

Solution: LW Stalls

- Instruction reordering to fill stalls

```
a = b + e;
c = b + f;
```




```
lw   $t1, 0($t0)
lw   $t2, 4($t0)
lw   $t4, 8($t0)
add  $t3, $t1,$t2
sw   $t3, 12($t0)
add  $t5, $t1,$t4
sw   $t5, 16($t0)
```

LW Forwarding Example 1

With no forwarding enabled

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	Comments
`lw $t1, 0($t0)`																				
`lw $t2, 4($t0)`																				
`add $t3, $t1, $t2`																				
`sw $t3, 12($t0)`																				
`lw $t4, 8($t0)`																				
`add $t5, $t1, $t4`																				
`sw $t5, 16($t0)`																				

LW Forwarding Example 1

With no forwarding enabled (dependencies)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	Comments
lw $t1, 0($t0)	F	D	E	M	W															
lw $t2, 4($t0)		F	D	E	M	W														
add $t3, $t1, $t2			F	-	-	D	E	M	W											Wait on $t2, stall 2 cycles
sw $t3, 12($t0)						F	-	-	D	E	M	W								Wait on $t3, stall 2 cycles
lw $t4, 8($t0)									F	D	E	M	W							
add $t5, $t1, $t4										F	-	-	D	E	M	W				Wait on $t4, stall 2 cycles
sw $t5, 16($t0)													F	-	-	D	E	M	W	Wait on $t5, stall 2 cycles

LW Forwarding Example 2

With forwarding enabled

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	Comments
`lw $t1, 0($t0)`																				
`lw $t2, 4($t0)`																				
`add $t3, $t1, $t2`																				
`sw $t3, 12($t0)`																				
`lw $t4, 8($t0)`																				
`add $t5, $t1, $t4`																				
`sw $t5, 16($t0)`																				

LW Forwarding Example 2

With forwarding enabled (dependencies – forwarding)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	Comments
`lw $t1, 0($t0)`	F	D	E	M	W															
`lw $t2, 4($t0)`		F	D	E	M	W														
`add $t3, $t1, $t2`			F	-	D	E	M	W												Forward $t2 from Mem/WB to Exec, Stall 1 cycle
`sw $t3, 12($t0)`					F	D	E	M	W											Forward $t3 from Exec/Mem to Exec.
`lw $t4, 8($t0)`						F	D	E	M	W										
`add $t5, $t1, $t4`							F	-	D	E	M	W								Forward $t4 from Mem/WB to Exec, Stall 1 cycle
`sw $t5, 16($t0)`									F	D	E	M	W							Forward $t5 from Exec/Mem to Exec.

LW Forwarding Example 3

With forwarding and reordering enabled

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	Comments
`lw $t1, 0($t0)`																				
`lw $t2, 4($t0)`																				
`add $t3, $t1, $t2`																				
`sw $t3, 12($t0)`																				
`lw $t4, 8($t0)`																				
`add $t5, $t1, $t4`																				
`sw $t5, 16($t0)`																				

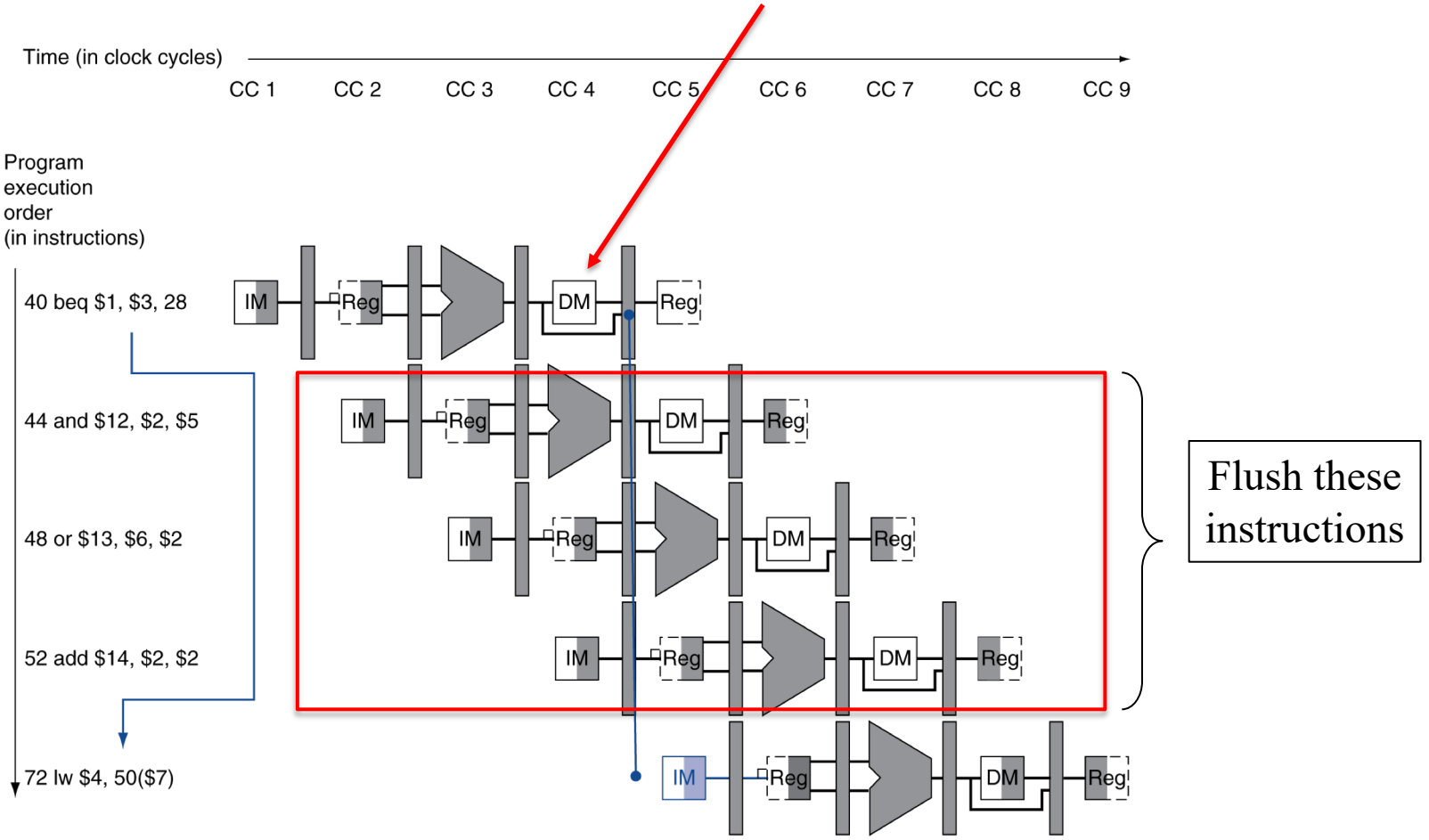
LW Forwarding Example 3

With forwarding and reordering enabled (dependencies – forwarding)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	Comments
`lw $t1, 0($t0)`	F	D	E	M	W															
`lw $t2, 4($t0)`		F	D	E	M	W														
`lw $t4, 8($t0)`			F	D	E	M	W													
`add $t3, $t1, $t2`				F	D	E	M	W												Forward $t2 from Mem/WB to Exec.
`sw $t3, 12($t0)`					F	D	E	M	W											Forward $t3 from Exec/Mem to Exec.
`add $t5, $t1, $t4`						F	D	E	M	W										
`sw $t5, 16($t0)`							F	D	E	M	W									Forward $t5 from Exec/Mem to Exec.

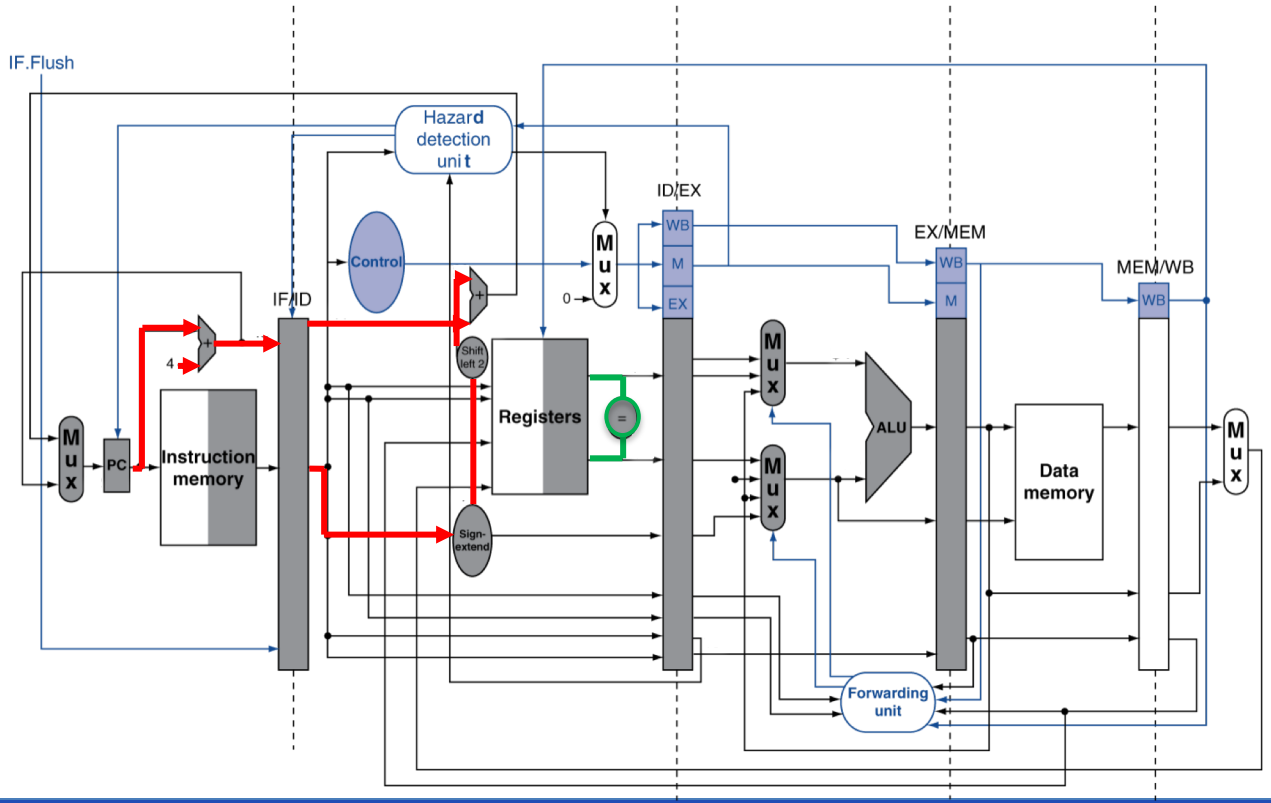
Problem: Branch Hazard

- If branch determined in MEM



Solution: Branch Hazard

- Move hardware determine outcome in ID stage
  - Target address adder
  - Register comparator
- Use branch prediction (history of branches) & instruction reordering



Branch Forwarding Example 1

Branches resolve in MEM, with forwarding enabled. Assume **stall on branch (i.e., no branch predictor)**

Branch Taken case

	1	2	3	4	5	6	7	8	9	10	11	12	13	Comments
`lw $t1, 8($t2)`														
`add $t4, $t5, $t6`														
`beq $t1, $t4, tgt`														
`add $t1, $t3, $t2`														
`add $t5, $t1, $t4`														
`add $t3, $t2, $t4`														
`tgt:` `lw $t1, 8($t2)`														

Branch Forwarding Example 1

Branches resolve in MEM, with forwarding enabled. Assume **stall on branch (i.e., no branch predictor)**

Branch Taken case

	1	2	3	4	5	6	7	8	9	10	11	12	13	Comments
`lw $t1, 8($t2)`	F	D	E	M	W									
`add $t4, $t5, $t6`		F	D	E	M	W								
`beq $t1, $t4, tgt`			F	D	E	M	W							Cycle 5, forward $t1 and $t4
`add $t1, $t3, $t2`				-	-									(Stall 2 cycles on branch)
`add $t5, $t1, $t4`														
`add $t3, $t2, $t4`														
`tgt:` `lw $t1, 8($t2)`						F	D	E	M	W				Fetch target in cycle 6 because branch is now resolved

Branch Forwarding Example 2

Branches resolve in MEM with forwarding enabled. Assume no stall on branch, so next instruction can be fetched right after branch **(i.e., branch predictor)**. Assume **always not-taken predictor.**

Branch Taken case

	1	2	3	4	5	6	7	8	9	10	11	12	13	Comments
`lw $t1, 8($t2)`														
`add $t4, $t5, $t6`														
`beq $t1, $t4, tgt`														
`add $t1, $t3, $t2`														
`add $t5, $t1, $t4`														
`add $t3, $t2, $t4`														
`tgt:` `lw $t1, 8($t2)`														

Branch Forwarding Example 2

Branches resolve in MEM with forwarding enabled. Assume no stall on branch, so next instruction can be fetched right after branch **(i.e., branch predictor)**. Assume **always not-taken predictor.**

Branch Taken case

	1	2	3	4	5	6	7	8	9	10	11	12	13	Comments
`lw $t1, 8($t2)`	F	D	E	M	W									
`add $t4, $t5, $t6`		F	D	E	M	W								
`beq $t1, $t4, tgt`			F	D	E	M	W							Cycle 5, forward $t1 and $t4
`add $t1, $t3, $t2`				F	D									Branch predicted as not-taken
`add $t5, $t1, $t4`					F									
`add $t3, $t2, $t4`														
`tgt:` `lw $t1, 8($t2)`						F	D	E	M	W				Fetch target in cycle 6 because branch is now resolved

Branch Forwarding Example 3

Branches resolve in MEM, with forwarding enabled. Assume **stall on branch (i.e., no branch predictor)**

Branch Not Taken case

	1	2	3	4	5	6	7	8	9	10	11	12	13	Comments
`lw $t1, 8($t2)`														
`add $t4, $t5, $t6`														
`beq $t1, $t4, tgt`														
`add $t1, $t3, $t2`														
`add $t5, $t1, $t4`														
`add $t3, $t2, $t4`														
`tgt:` `lw $t1, 8($t2)`														

Branch Forwarding Example 3

Branches resolve in MEM, with forwarding enabled. Assume **stall on branch (i.e., no branch predictor)**

Branch Not Taken case

	1	2	3	4	5	6	7	8	9	10	11	12	13	Comments
`lw $t1, 8($t2)`	F	D	E	M	W									
`add $t4, $t5, $t6`		F	D	E	M	W								
`beq $t1, $t4, tgt`			F	D	E	M	W							Cycle 5, forward $t1 and $t4
`add $t1, $t3, $t2`				-	-	F	D	E	M	W				Branch resolves in cycle 6 (Stall 2 cycles on branch)
`add $t5, $t1, $t4`							F	D	E	M	W			
`add $t3, $t2, $t4`								F	D	E	M	W		
`tgt:` `lw $t1, 8($t2)`									F	D	E	M	W	

Branch Forwarding Example 4

Branches resolve in MEM, with forwarding enabled. Assume no stall on branch, so next instruction can be fetched right after branch **(i.e., branch predictor)**. Assume **always taken predictor.**

Branch Not Taken case

	1	2	3	4	5	6	7	8	9	10	11	12	13	Comments
`lw $t1, 8($t2)`														
`add $t4, $t5, $t6`														
`beq $t1, $t4, tgt`														
`add $t1, $t3, $t2`														
`add $t5, $t1, $t4`														
`add $t3, $t2, $t4`														
`tgt:` `lw $t1, 8($t2)`														

Branch Forwarding Example 4

Branches resolve in MEM, with forwarding enabled. Assume no stall on branch, so next instruction can be fetched right after branch **(i.e., branch predictor)**. Assume **always taken predictor.**

Branch Not Taken case

	1	2	3	4	5	6	7	8	9	10	11	12	13	Comments
`lw $t1, 8($t2)`	F	D	E	M	W									
`add $t4, $t5, $t6`		F	D	E	M	W								
`beq $t1, $t4, tgt`			F	D	E	M	W							Cycle 5, forward $t1 and $t4
`add $t1, $t3, $t2`						F	D	E	M	W				Branch resolved in cycle 6. Misprediction!
`add $t5, $t1, $t4`							F	D	E	M	W			
`add $t3, $t2, $t4`								F	D	E	M	W		
`tgt:` `lw $t1, 8($t2)`				F	D									(assumes branch taken)

Practice Midterm

1. (20 points total) We have the following timing data (in ns) on the MIPS data-path. Please write your answer in the table below.

Inst type	*Inst. Mem*	*Reg File Read*	*ALU*	*Data Mem*	*Reg File Write*	*Total (ns)*	*Inst %*
R-type	2	1	2	0	1	6	40
Load	2	1	2	2	1	8	20
Store	2	1	2	2	0	7	10
Branch	2	1	2	0	0	5	20
Jump	2	0	0	0	0	2	10

a. (4 points) What is the clock cycle time if this is a single cycle implementation? What is the CPI?

Practice Midterm

1. (20 points total) We have the following timing data (in ns) on the MIPS data-path. Please write your answer in the table below.

Inst type	*Inst. Mem*	*Reg File Read*	*ALU*	*Data Mem*	*Reg File Write*	*Total (ns)*	*Inst %*
R-type	2	1	2	0	1	6	40
Load	2	1	2	2	1	8	20
Store	2	1	2	2	0	7	10
Branch	2	1	2	0	0	5	20
Jump	2	0	0	0	0	2	10

a. (4 points) What is the clock cycle time if this is a single cycle implementation? What is the CPI?

Clock cycle time = 8ns

CPI = 1

Practice Midterm

1. (20 points total) We have the following timing data (in ns) on the MIPS data-path. Please write your answer in the table below.

Inst type	*Inst. Mem*	*Reg File Read*	*ALU*	*Data Mem*	*Reg File Write*	*Total (ns)*	*Inst %*
R-type	2	1	2	0	1	6	40
Load	2	1	2	2	1	8	20
Store	2	1	2	2	0	7	10
Branch	2	1	2	0	0	5	20
Jump	2	0	0	0	0	2	10

b. (5 points) Compute the CPI if this is a multi-cycle implementation.

Practice Midterm

1. (20 points total) We have the following timing data (in ns) on the MIPS data-path. Please write your answer in the table below.

Inst type	Inst. Mem	Reg File Read	ALU	Data Mem	Reg File Write	Total (ns)	Inst %
R-type	2	1	2	0	1	6	40
Load	2	1	2	2	1	8	20
Store	2	1	2	2	0	7	10
Branch	2	1	2	0	0	5	20
Jump	2	0	0	0	0	2	10

b. (5 points) Compute the CPI if this is a multi-cycle implementation.

R-type = 4 cycles (IF, ID, EX, WB) → 4*.4
Load = 5 cycles (IF, ID, EX, MEM, WB) → 5*.2
Store = 4 cycles (IF, ID, EX, MEM) → 4*.1
Branch = 3 cycles (IF, ID, EX) → 3*.2
Jump = 3 cycles (IF, ID, Jump) → 3*.1

CPI = 4*.4+5*.2+4*.1+3*.2+3*.1 = 3.9

Practice Midterm

1. (20 points total) We have the following timing data (in ns) on the MIPS data-path. Please write your answer in the table below.

Inst type	*Inst. Mem*	*Reg File Read*	*ALU*	*Data Mem*	*Reg File Write*	*Total (ns)*	*Inst %*
R-type	2	1	2	0	1	6	40
Load	2	1	2	2	1	8	20
Store	2	1	2	2	0	7	10
Branch	2	1	2	0	0	5	20
Jump	2	0	0	0	0	2	10

c. (5 points) What is the clock cycle time if the multi-cycle implementation has a 500 MHz clock frequency? Assume the program has 100 instructions, what is the CPU time?

Practice Midterm

1. (20 points total) We have the following timing data (in ns) on the MIPS data-path. Please write your answer in the table below.

Inst type	Inst. Mem	Reg File Read	ALU	Data Mem	Reg File Write	Total (ns)	Inst %
R-type	2	1	2	0	1	6	40
Load	2	1	2	2	1	8	20
Store	2	1	2	2	0	7	10
Branch	2	1	2	0	0	5	20
Jump	2	0	0	0	0	2	10

c. (5 points) What is the clock cycle time if the multi-cycle implementation has a 500 MHz clock frequency? Assume the program has 100 instructions, what is the CPU time?

$$clock\ cycle = \frac{1}{clock\ rate} = \frac{1}{500\ MHz} = \frac{1}{500 * 10^6} = \frac{1}{0.5 * 10^9} = 2 * 10^{-9} = 2\ ns$$

$$CPU\ time = clock\ cycle\ * CPI\ * I = 2\ ns\ * 3.9\ * 100 = 780 ns$$

Practice Midterm

1. (20 points total) We have the following timing data (in ns) on the MIPS data-path. Please write your answer in the table below.

Inst type	*Inst. Mem*	*Reg File Read*	*ALU*	*Data Mem*	*Reg File Write*	*Total (ns)*	*Inst %*
R-type	2	1	2	0	1	6	40
Load	2	1	2	2	1	8	20
Store	2	1	2	2	0	7	10
Branch	2	1	2	0	0	5	20
Jump	2	0	0	0	0	2	10

d. (6 points) A modified multi-cycle micro-architecture implementation reduces the ALU time to 1 ns, and by doubling the number of registers can reduce the number of Load instructions by half. However, the clock cycle time must be increased by 10%. Compute the new CPU time

Practice Midterm

1. (20 points total) We have the following timing data (in ns) on the MIPS data-path. Please write your answer in the table below.

Inst type	Inst. Mem	Reg File Read	ALU	Data Mem	Reg File Write	Total (ns)	Inst %
R-type	2	1	2	0	1	6	40
Load	2	1	2	2	1	8	20
Store	2	1	2	2	0	7	10
Branch	2	1	2	0	0	5	20
Jump	2	0	0	0	0	2	10

d. (6 points) A modified multi-cycle micro-architecture implementation reduces the ALU time to 1 ns, and by doubling the number of registers can reduce the number of Load instructions by half. However, the clock cycle time must be increased by 10%. Compute the new CPU time

$$new\ CPI = 4(.4) + 5(.1) + 4(.1) + 3(.2) + 3(.1) = 3.4$$

$$new\ clock\ cyce = 2\ ns\ * 1.10$$

$$CPU\ time = clock\ cycle\ * CPI\ * I = 2\ ns\ *\ 1.10\ * 3.4\ * 100 = 748 ns$$

Practice Midterm

2. (10 points) The following equation describes the relationship between CPU time and Instruction Count (IC), Clocks per Instruction (CPI), and Clock Cycle Time (CCT):

   CPU time = IC * CPI * CCT

   Each of these is influenced by a number of factors. Indicate this influence in the table below by putting a checkmark in the appropriate column.

Factor	IC	CPI	CCT
Algorithm			
Programming language			
Compiler			
ISA			
Micro-architecture			
Technology (feature size)			

Practice Midterm

2. (10 points) The following equation describes the relationship between CPU time and Instruction Count (IC), Clocks per Instruction (CPI), and Clock Cycle Time (CCT):

   CPU time = IC * CPI * CCT

   Each of these is influenced by a number of factors. Indicate this influence in the table below by putting a checkmark in the appropriate column.

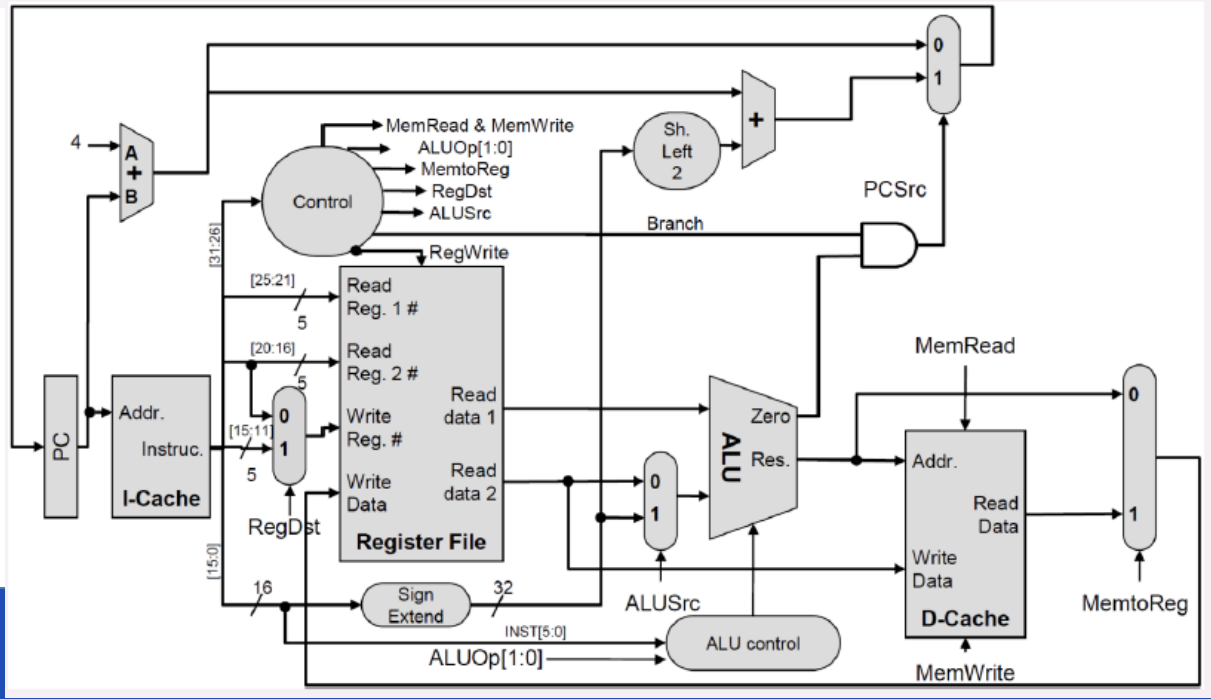
Factor	IC	CPI	CCT
Algorithm	X	X	
Programming language	X	X	
Compiler	X	X	
ISA	X	X	X
Micro-architecture		X	X
Technology (feature size)			X

Practice Midterm

4. (20 points) The figure below shows a single cycle data path implementation along with the truth table to generate the control signals. This problem focuses on the LW (load word) instruction. An example of this instruction is: LW R2, 8(R1)

   a. (10 points) The control signals for the LW instruction is omitted. Please fill it in.

R-Type	LW	SW	BEQ	J	Jump	Branch	Reg Dst	ALU Src	Memto-Reg	Reg Write	Mem Read	Mem Write	ALU Op[1]	ALU Op[0]
1	0	0	0	0	0	0	1	0	0	1	0	0	1	0
0	1	0	0	0										
0	0	1	0	0	0	0	X	1	X	0	0	1	0	0
0	0	0	1	0	0	1	X	0	X	0	0	0	0	1
0	0	0	0	1	1	0	X	X	X	0	0	0	X	X

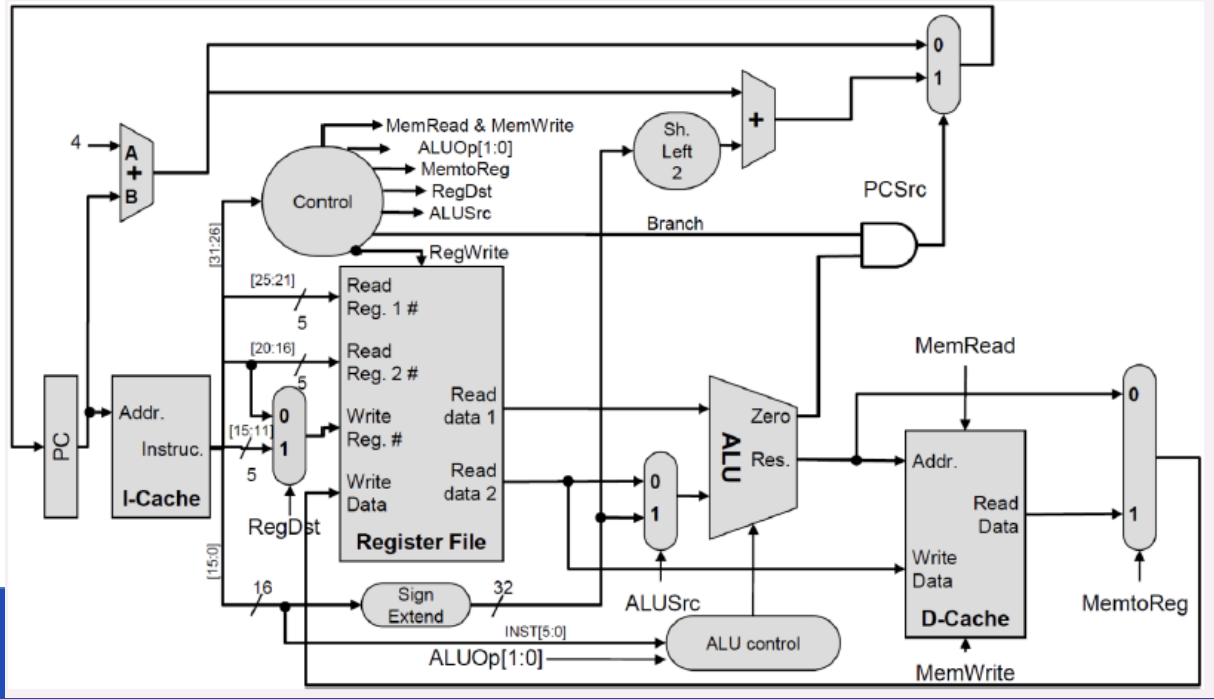


Practice Midterm

4. (20 points) The figure below shows a single cycle data path implementation along with the truth table to generate the control signals. This problem focuses on the LW (load word) instruction. An example of this instruction is: LW R2, 8(R1)

   a. (10 points) The control signals for the LW instruction is omitted. Please fill it in.

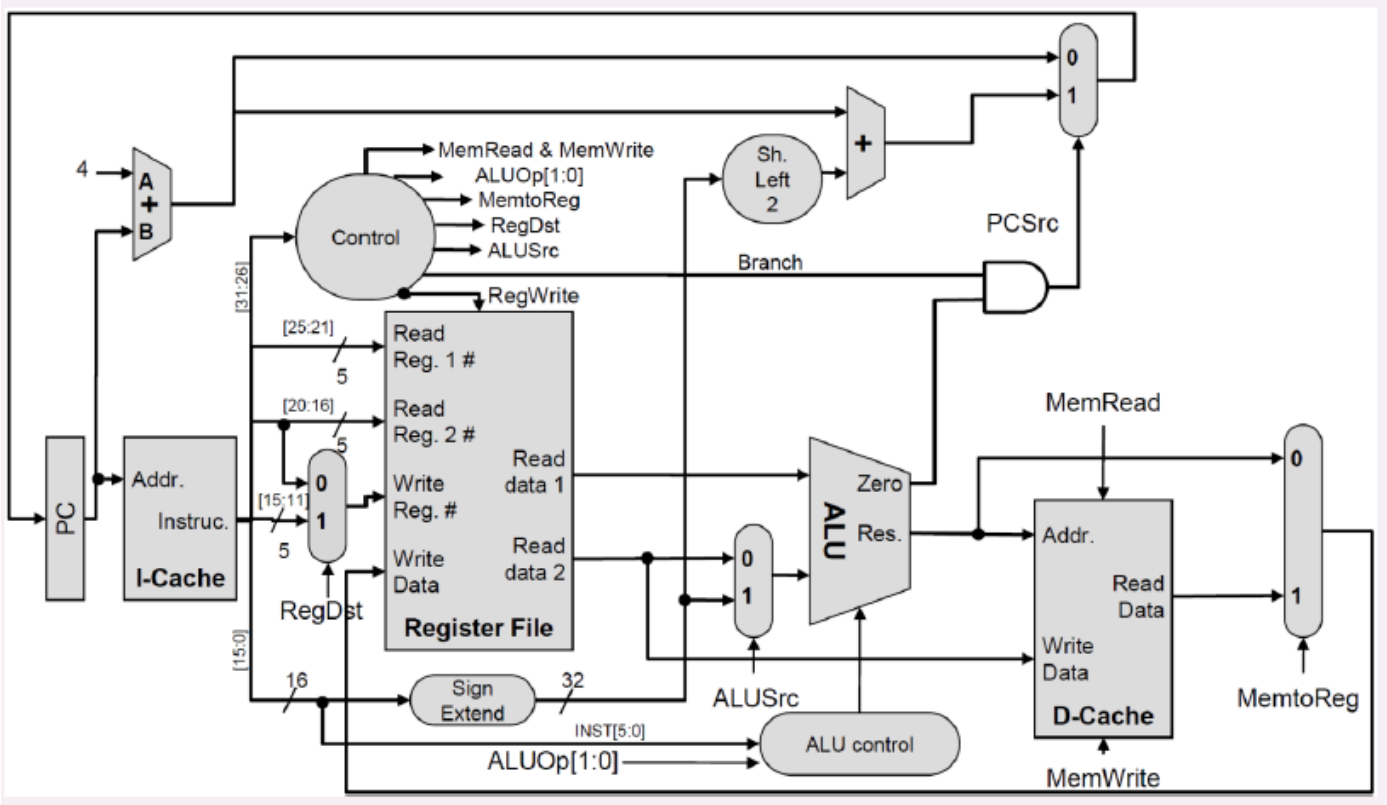
R-Type	LW	SW	BEQ	J	Jump	Branch	Reg Dst	ALU Src	Memto-Reg	Reg Write	Mem Read	Mem Write	ALU Op[1]	ALU Op[0]
1	0	0	0	0	0	0	1	0	0	1	0	0	1	0
0	1	0	0	0	0	0	0	1	1	1	1	0	0	0
0	0	1	0	0	0	0	X	1	X	0	0	1	0	0
0	0	0	1	0	0	1	X	0	X	0	0	0	0	1
0	0	0	0	1	1	0	X	X	X	0	0	0	X	X



Practice Midterm

4. (20 points) The figure below shows a single cycle data path implementation along with the truth table to generate the control signals. This problem focuses on the LW (load word) instruction. An example of this instruction is: LW R2, 8(R1)

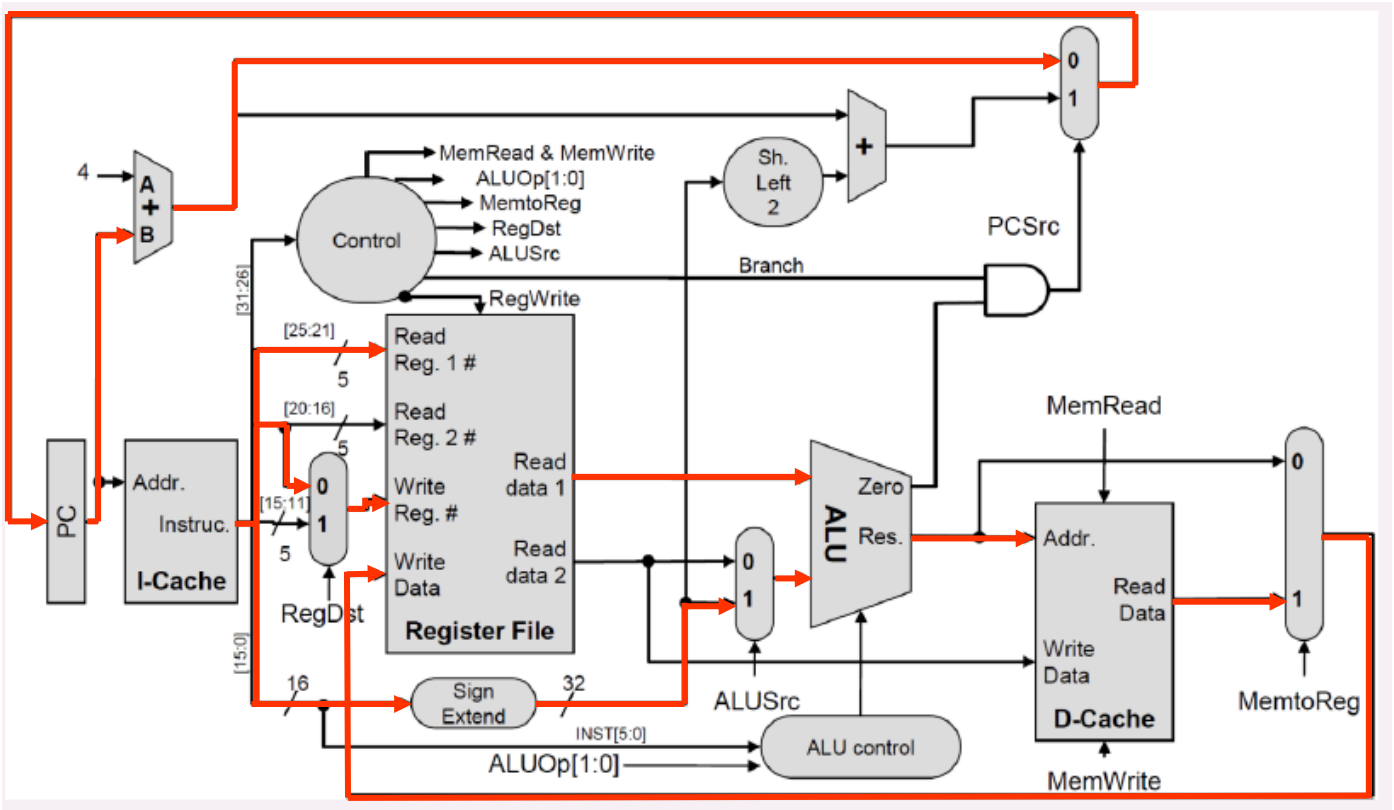
b. (10 points) Draw out the data paths that are utilized for the LW instruction



Practice Midterm

4. (20 points) The figure below shows a single cycle data path implementation along with the truth table to generate the control signals. This problem focuses on the LW (load word) instruction. An example of this instruction is: LW R2, 8(R1)

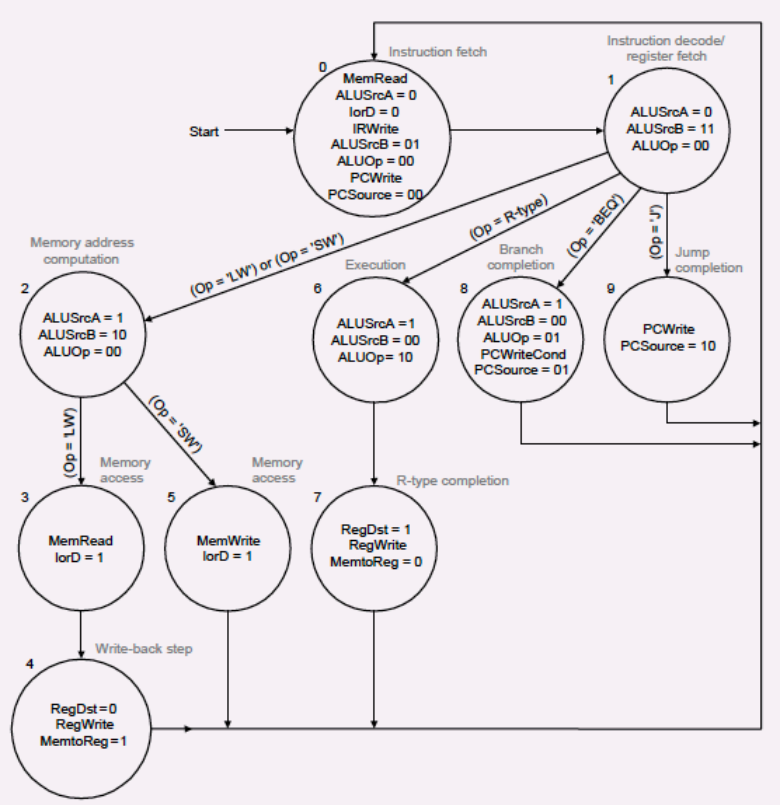
b. (10 points) Draw out the data paths that are utilized for the LW instruction



Practice Midterm

5. (30 points) The figure below, and on the next page, shows the control state machine and data path for a multi-cycle cpu implementation. We would like to modify this implementation to include a move instruction which moves the values of one register into another register.
Example: MOV Rt, Rs which performs the following: Reg[Rt] = Reg[Rs]. Assume this instruction is an I-type instruction.

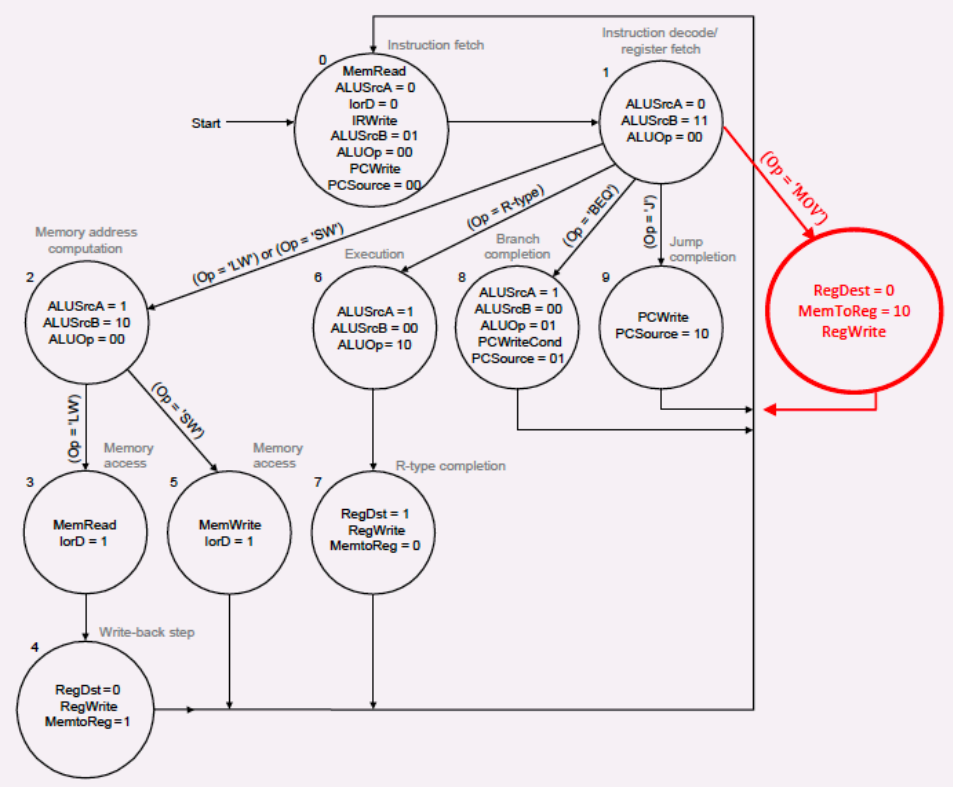
   a. (12 points) Modify the state machine to support the MOV instruction



Practice Midterm

5. (30 points) The figure below, and on the next page, shows the control state machine and data path for a multi-cycle cpu implementation. We would like to modify this implementation to include a move instruction which moves the values of one register into another register.
Example: MOV Rt, Rs which performs the following: Reg[Rt] = Reg[Rs]. Assume this instruction is an I-type instruction.

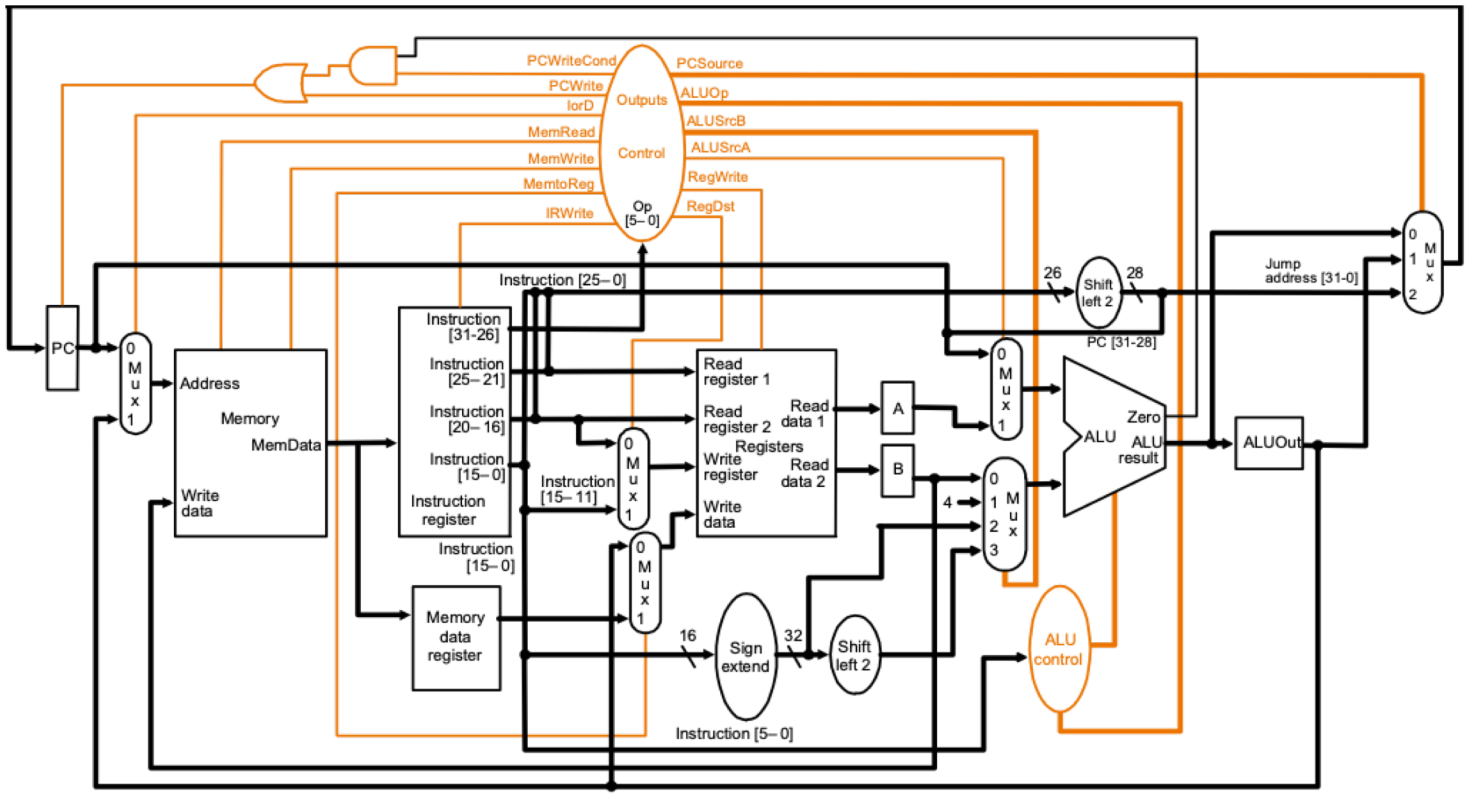
   a. (12 points) Modify the state machine to support the MOV instruction



Practice Midterm

5. (30 points) The figure below, and on the next page, shows the control state machine and data path for a multi-cycle cpu implementation. We would like to modify this implementation to include a move instruction which moves the values of one register into another register.
Example: MOV Rt, Rs which performs the following: Reg[Rt] = Reg[Rs]. Assume this instruction is an I-type instruction.

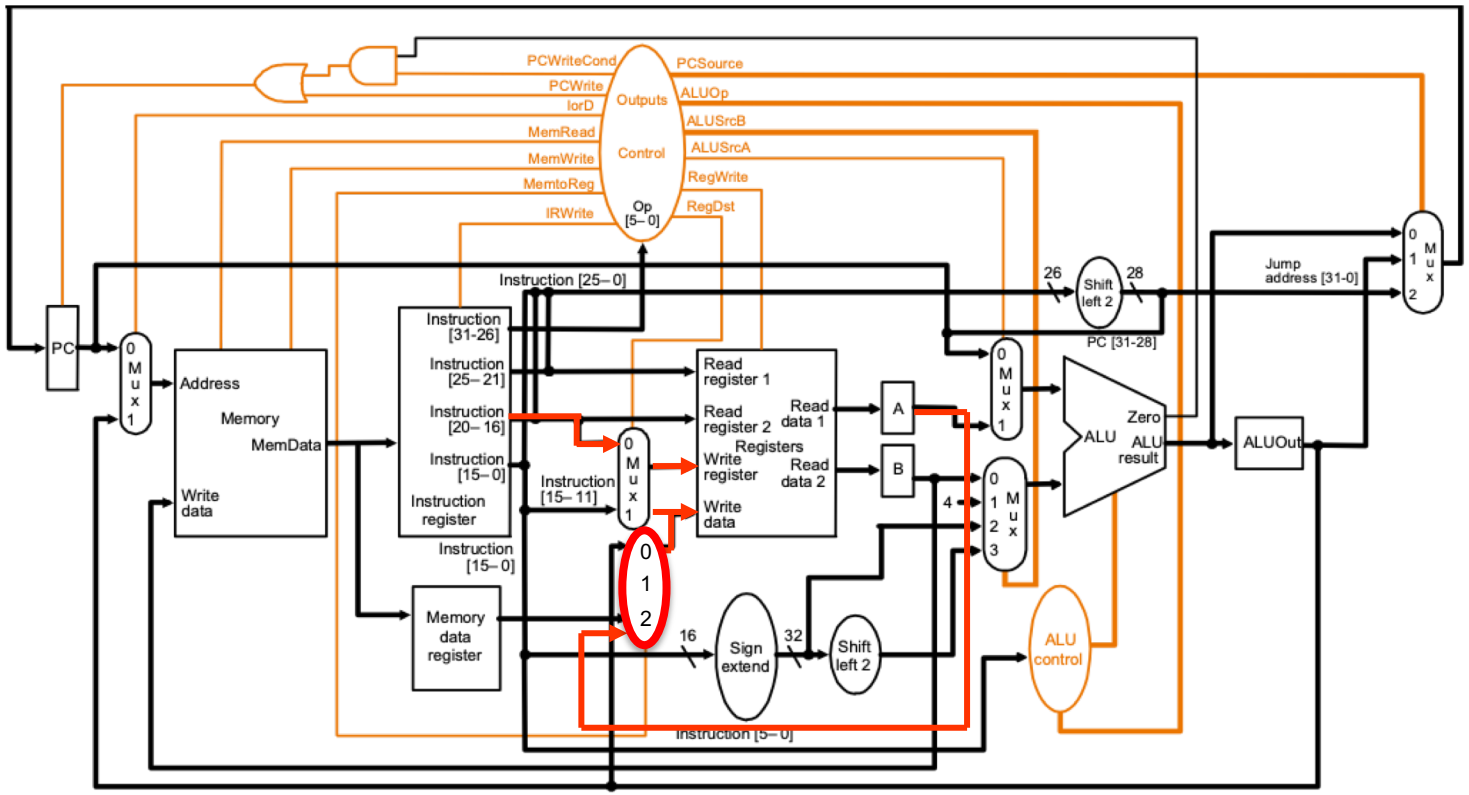
   b. (6 points) On the next page, draw the data path modifications required to support the MOV instruction



Practice Midterm

5. (30 points) The figure below, and on the next page, shows the control state machine and data path for a multi-cycle cpu implementation. We would like to modify this implementation to include a move instruction which moves the values of one register into another register.
Example: MOV Rt, Rs which performs the following: Reg[Rt] = Reg[Rs]. Assume this instruction is an I-type instruction.

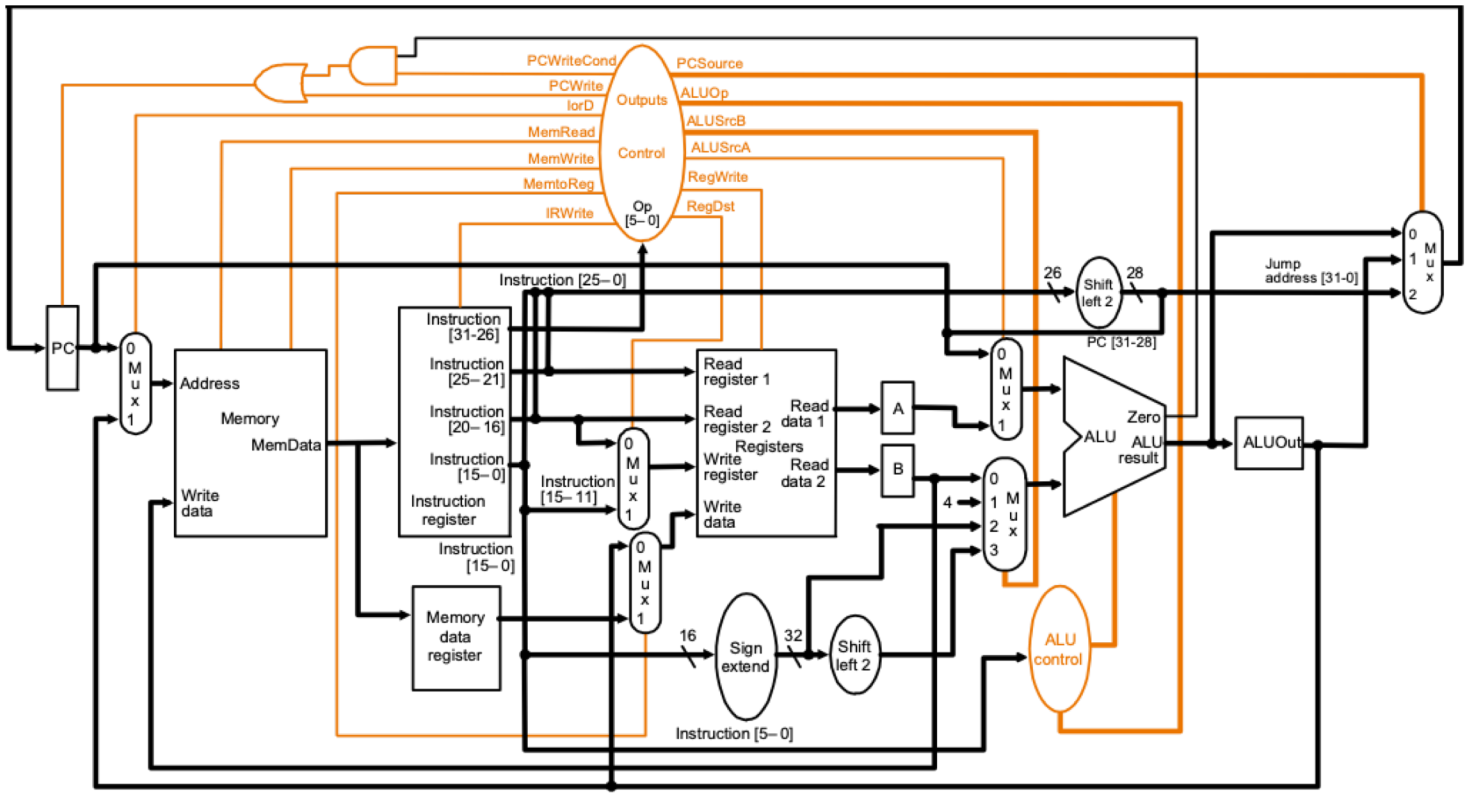
   b. (6 points) On the next page, draw the data path modifications required to support the MOV instruction



Practice Midterm

5. (30 points) The figure below, and on the next page, shows the control state machine and data path for a multi-cycle cpu implementation. We would like to modify this implementation to include a move instruction which moves the values of one register into another register.
Example: MOV Rt, Rs which performs the following: Reg[Rt] = Reg[Rs]. Assume this instruction is an I-type instruction.

c. (12 points) On the next page, draw out the data paths that are utilized during the fetch and execute state



Practice Midterm

5. (30 points) The figure below, and on the next page, shows the control state machine and data path for a multi-cycle cpu implementation. We would like to modify this implementation to include a move instruction which moves the values of one register into another register.
Example: MOV Rt, Rs which performs the following: Reg[Rt] = Reg[Rs]. Assume this instruction is an I-type instruction.

c. (12 points) On the next page, draw out the data paths that are utilized during the fetch and execute state