CS/ECE 552: Midterm Review

Prof. Matthew D. Sinclair

Lecture notes based in part on slides created by Mikko Lipasti, Mark Hill, Josh San Miguel, and John Shen

Announcements

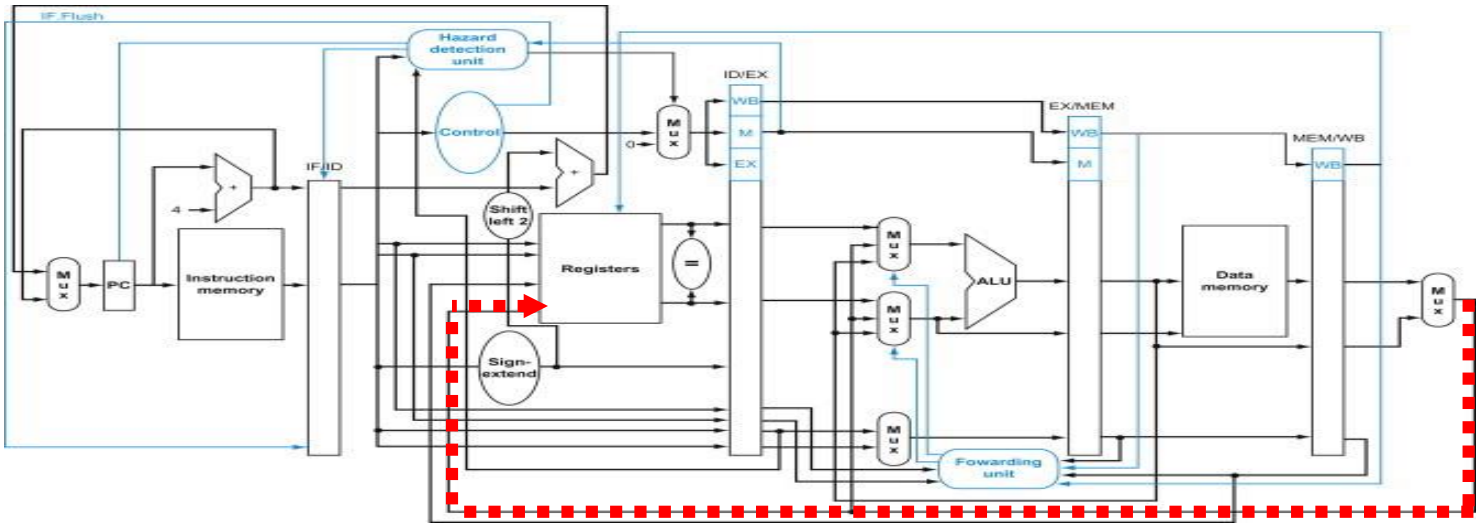
- **Midterm** Thursday (3/5 in class)
  - Closed book, one double-sided hand-written 8.5”x11” cheat sheet
  - Calculators allowed
  - MIPS green cards provided
  - Covers Weeks 1 through 6
  - Posted additional Midterm Details
    - Practice Exams posted on Canvas Week 7
    - Link to Course Website with topics that will covered
- Mid-semester evals (due 3/6) – please fill out!
  - >= 60% → treat for class
- HW3 grading in progress

Register File Bypassing

Producer instruction:
- Writes into a register

Consumer instruction:
- Reads that register

insn\cycle	1	2	3	4	5	6	7	8
add $t0, $t1, $t2	F	D	X	M	W			
…		F	D	X	M	W		
…			F	D	X	M	W	
add $s0, $s0, $t0				F	D	X	M	W



COD Figure 4.65

EX-to-EX Forwarding

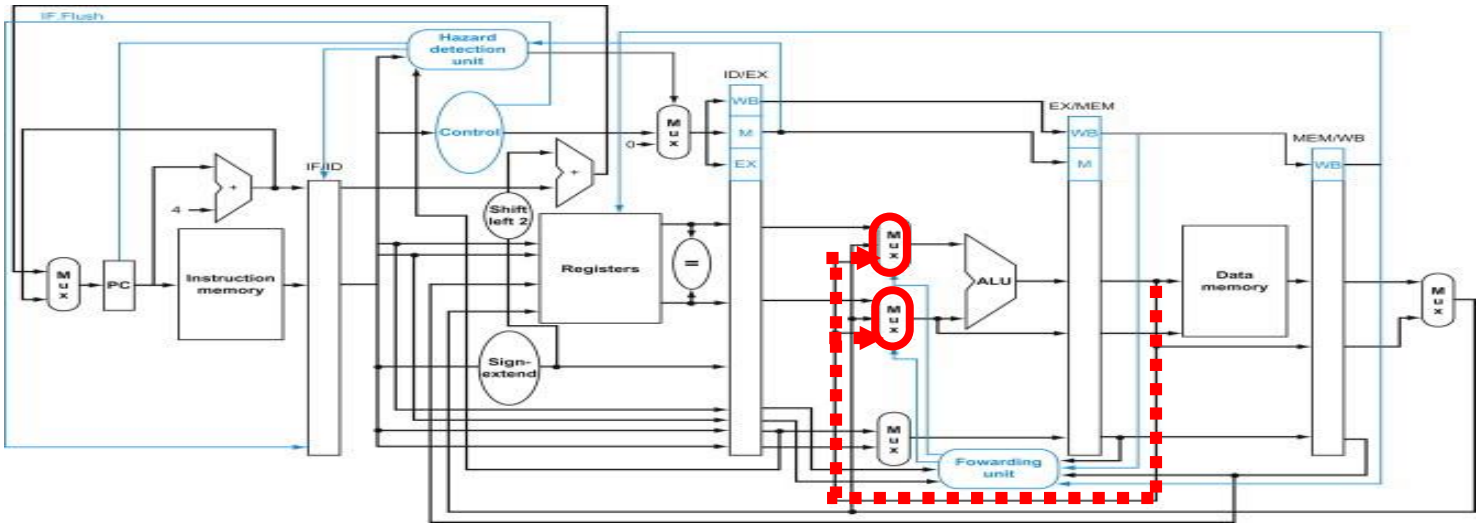
Producer instruction:

- Writes an ALU result into a register

Consumer instruction:

- Reads that register and needs its value for ALU

insn\cycle	1	2	3	4	5	6	7	8
add $t0, $t1, $t2	F	D	X	M	W			
add $s0, $s0, $t0		F	D	X	M	W		



COD Figure 4.65

MEM-to-EX Forwarding

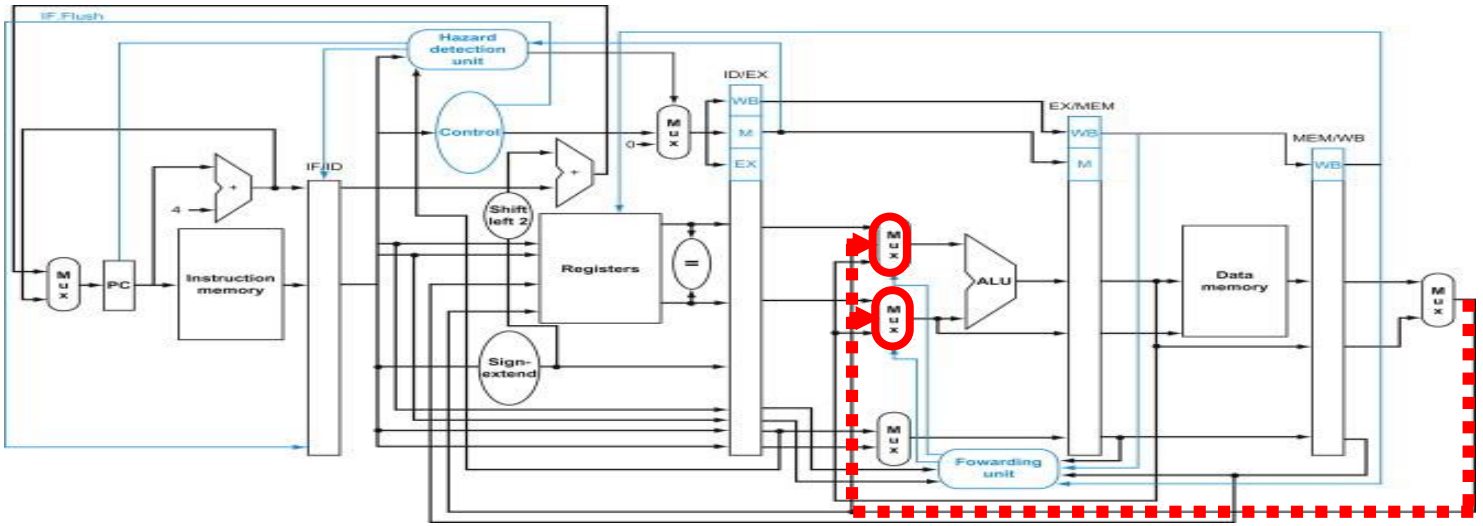
Producer instruction:

- Writes an ALU result into a register

Consumer instruction:

- Reads that register and needs its value for ALU, one cycle later

insn\cycle	1	2	3	4	5	6	7	8
add $t0, $t1, $t2	F	D	X	M	W			
…		F	D	X	M	W		
add $s0, $s0, $t0			F	D	X	M	W	



COD Figure 4.65

MEM-to-EX Forwarding

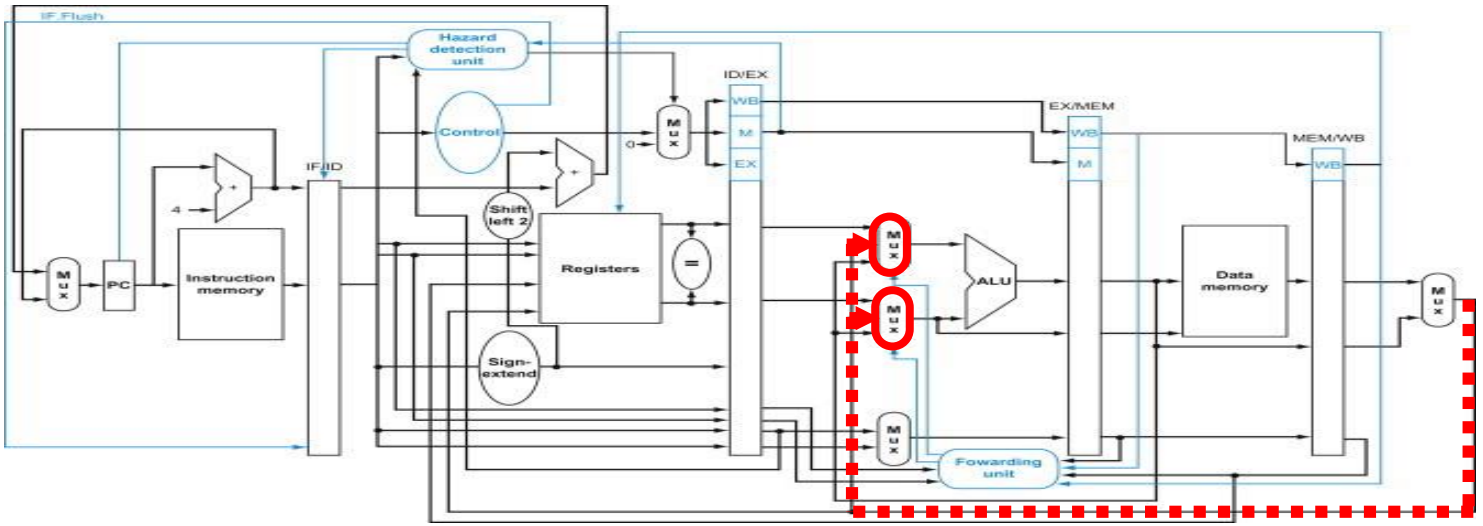
Producer instruction:

- Writes a loaded memory result into a register

Consumer instruction:

- Reads that register and needs its value for ALU

insn\cycle	1	2	3	4	5	6	7	8
lw $t0, 0($t1)	F	D	X	M	W			
sw $s0, 0($t0)		F	D*	D	X	M	W	



COD Figure 4.65

MEM-to-MEM Forwarding

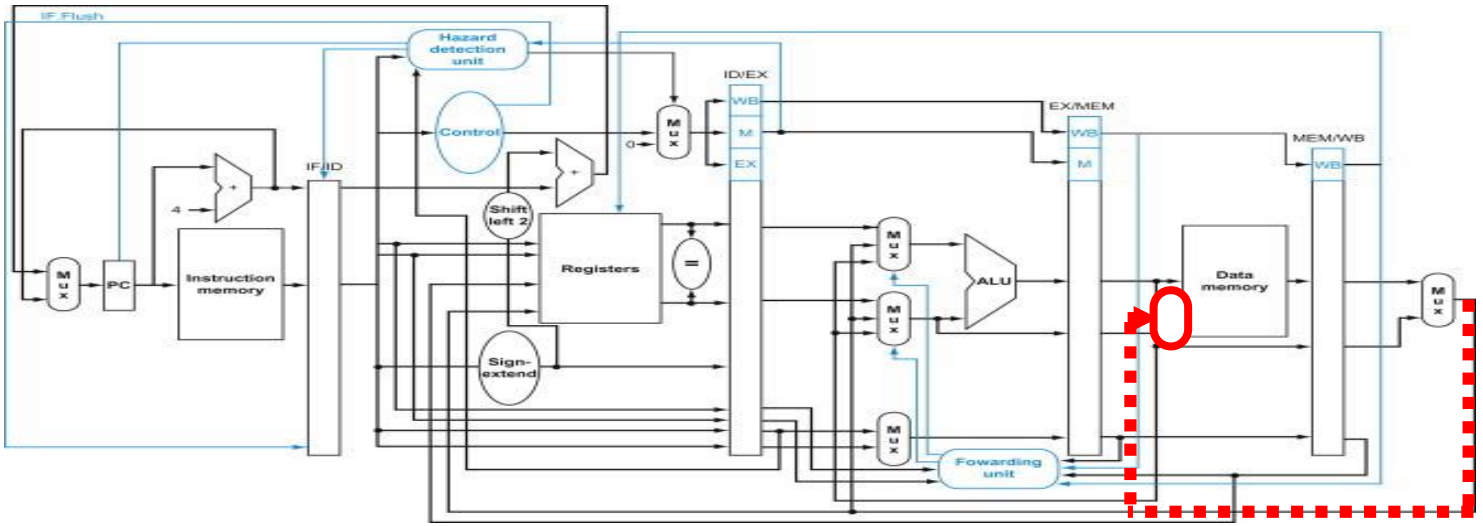
Producer instruction:

- Writes a loaded memory result into a register

Consumer instruction:

- Reads that register and needs its value for storing to memory

insn\cycle	1	2	3	4	5	6	7	8
lw $t0, 0($t1)	F	D	X	M	W			
sw $t0, 0($s0)		F	D	X	M	W		



COD Figure 4.65

EX Forwarding to Branch in ID

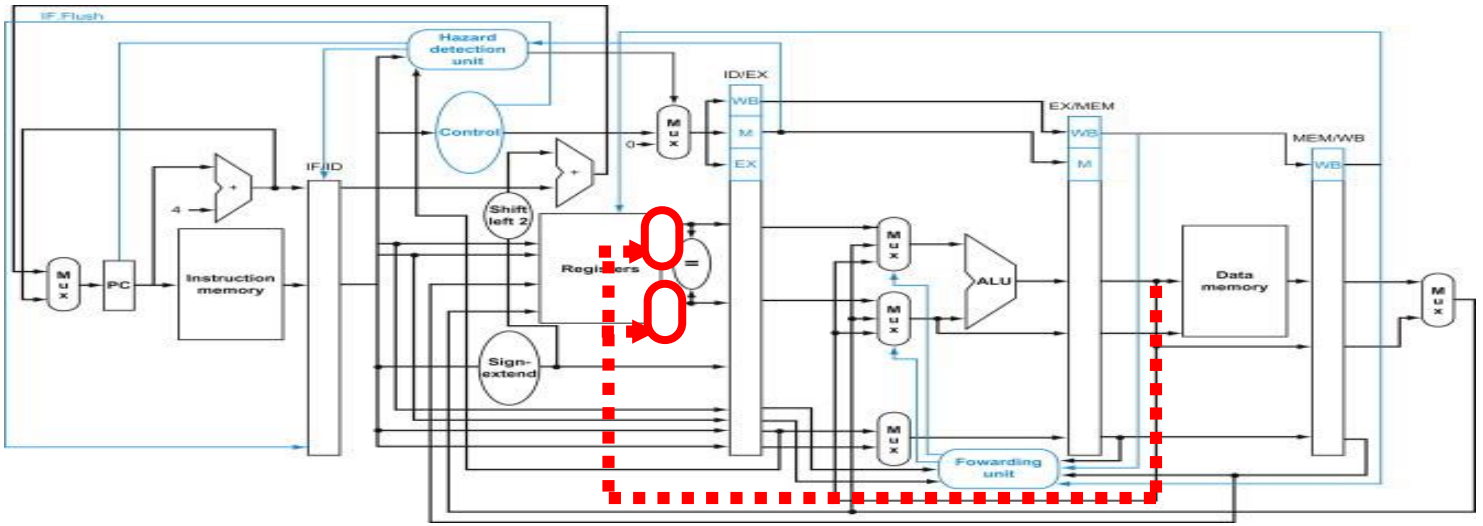
Producer instruction:

- Writes an ALU result into a register

Consumer instruction:

- Reads that register and needs its value for branch decision in ID

insn\cycle	1	2	3	4	5	6	7	8
add $t0, $t1, $t2	F	D	X	M	W			
beq $s0, $t0, DST		F	D*	D	X	M	W	



COD Figure 4.65

Register File Bypassing to Branch in ID

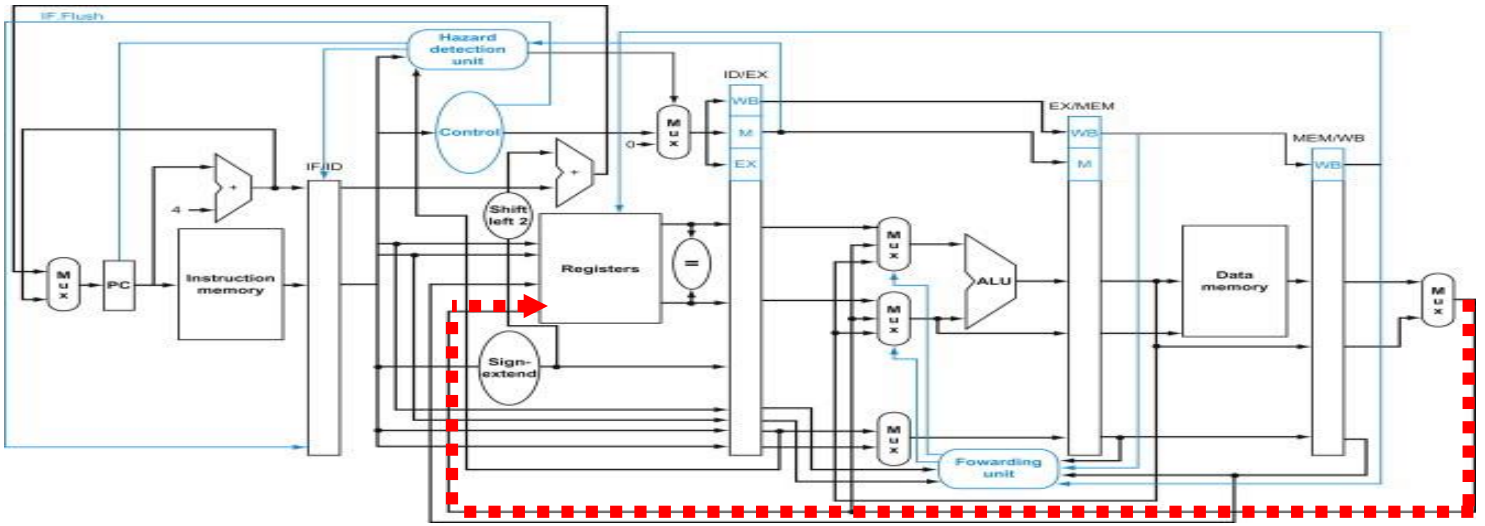
Producer instruction:

- Writes a loaded memory result into a register

Consumer instruction:

- Reads that register and needs its value for branch decision in ID

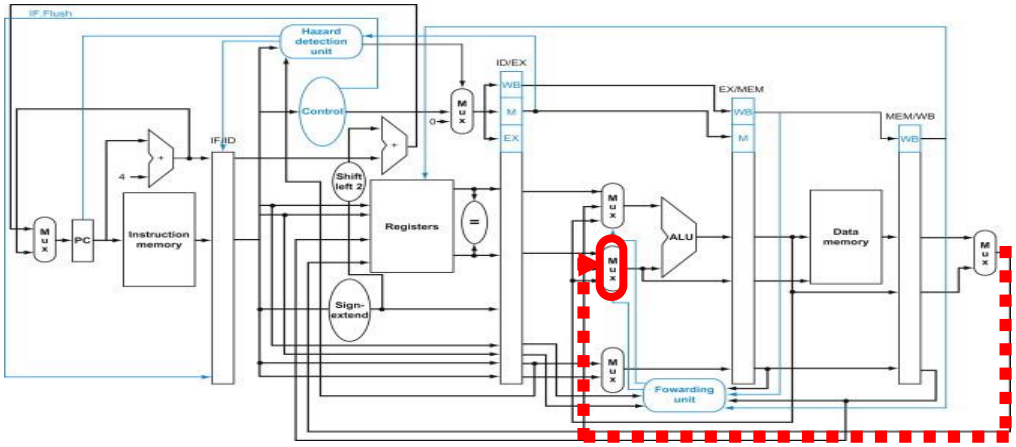
insn\cycle	1	2	3	4	5	6	7	8
lw $t0, 0($t1)	F	D	X	M	W			
beq $s0, $t0, DST		F	D*	D*	D	X	M	W



COD Figure 4.65

Data Forwarding

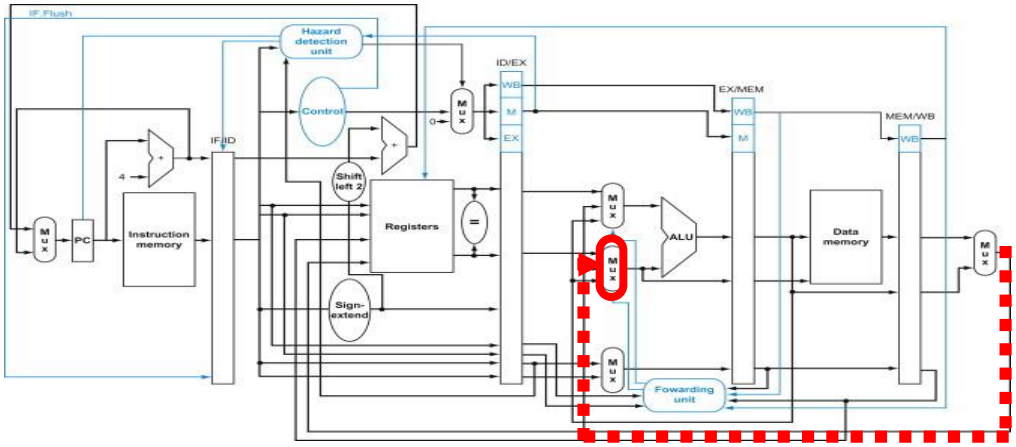
lw $t0, 0($t1)
lw $s1, 4($t0)
add $t2, $t3, $s1



At which cycle(s) is this forwarding path enabled?
Assume full forwarding and bypassing.

Data Forwarding

lw $t0, 0($t1)
lw $s1, 4($t0)
add $t2, $t3, $s1

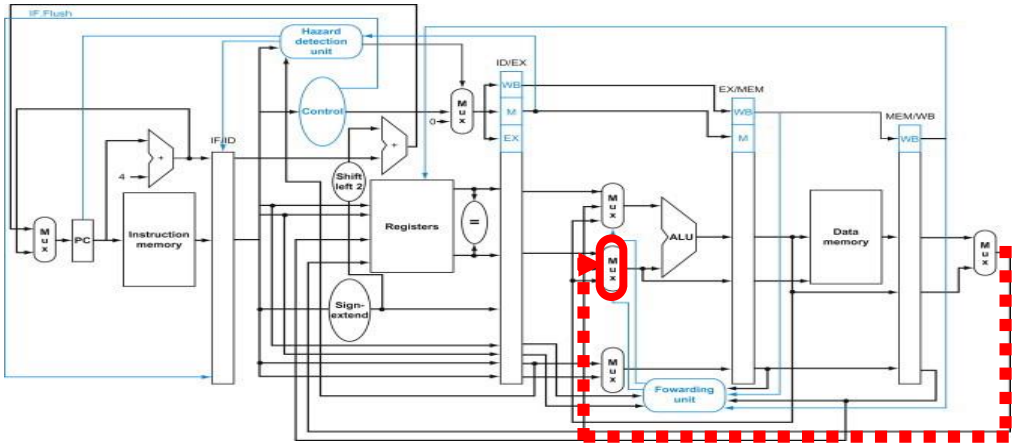


At which cycle(s) is this forwarding path enabled?
Assume full forwarding and bypassing.

insn\cycle	1	2	3	4	5	6	7	8	9	10	11	12	13
lw $t0	F	D	X	M	W								

Data Forwarding

lw $t0, 0($t1)
lw $s1, 4($t0)
add $t2, $t3, $s1

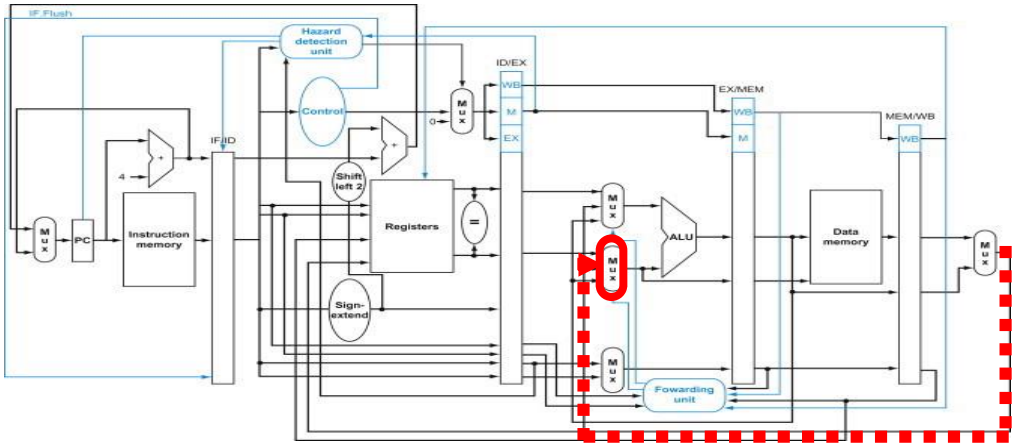


At which cycle(s) is this forwarding path enabled?
Assume full forwarding and bypassing.

insn\cycle	1	2	3	4	5	6	7	8	9	10	11	12	13
lw $t0	F	D	X	M	W								
lw $s1		F	D*	D									

Data Forwarding

lw $t0, 0($t1)
lw $s1, 4($t0)
add $t2, $t3, $s1

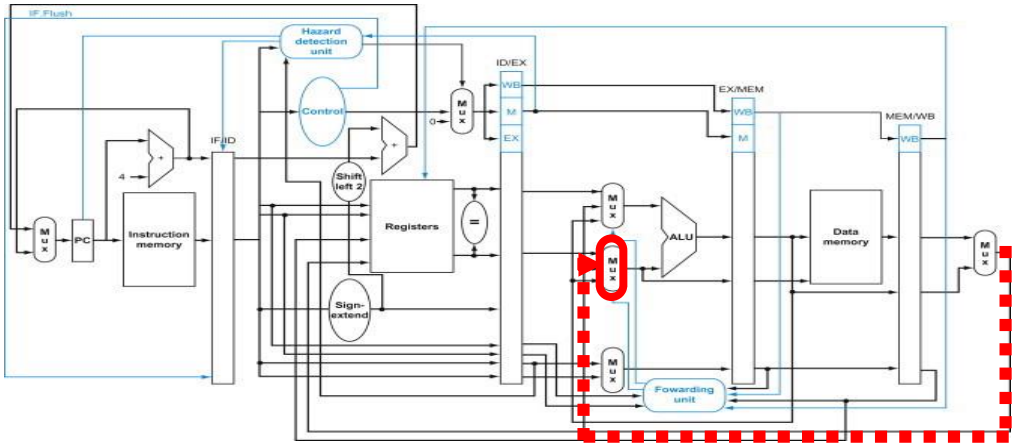


At which cycle(s) is this forwarding path enabled?
Assume full forwarding and bypassing.

insn\cycle	1	2	3	4	5	6	7	8	9	10	11	12	13
lw $t0	F	D	X	M	W								
lw $s1		F	D*	D	X	M	W						

Data Forwarding

lw $t0, 0($t1)
lw $s1, 4($t0)
add $t2, $t3, $s1

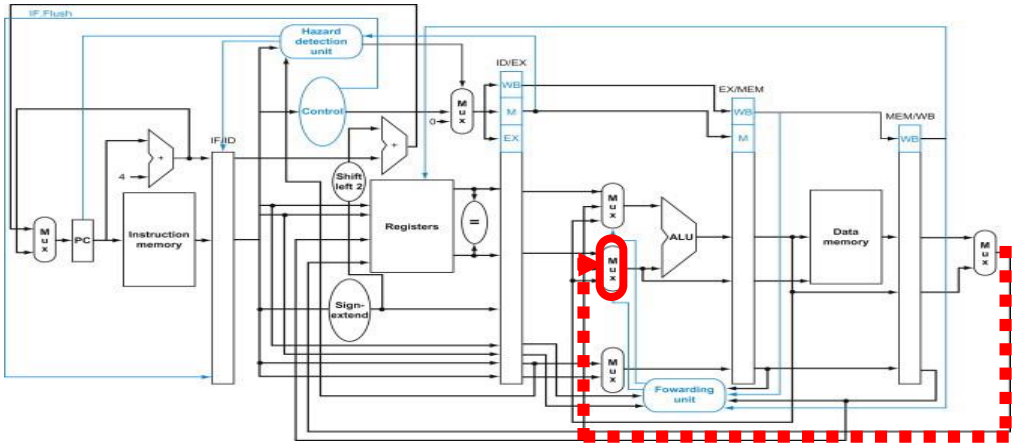


At which cycle(s) is this forwarding path enabled?
Assume full forwarding and bypassing.

insn\cycle	1	2	3	4	5	6	7	8	9	10	11	12	13
lw $t0	F	D	X	M	W								
lw $s1		F	D*	D	X	M	W						
add			F*	F									

Data Forwarding

lw $t0, 0($t1)
lw $s1, 4($t0)
add $t2, $t3, $s1

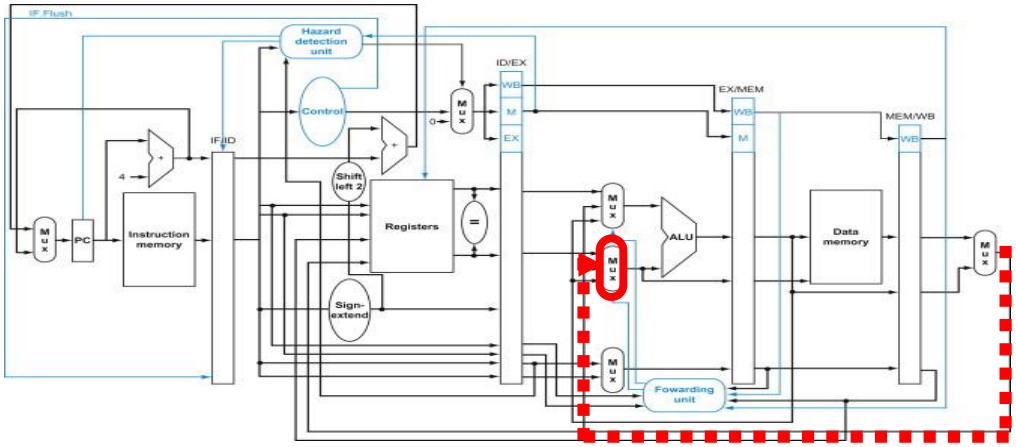


At which cycle(s) is this forwarding path enabled?
Assume full forwarding and bypassing.

insn\cycle	1	2	3	4	5	6	7	8	9	10	11	12	13
lw $t0	F	D	X	M	W								
lw $s1		F	D*	D	X	M	W						
add			F*	F	D*	D							

Data Forwarding

lw $t0, 0($t1)
lw $s1, 4($t0)
add $t2, $t3, $s1

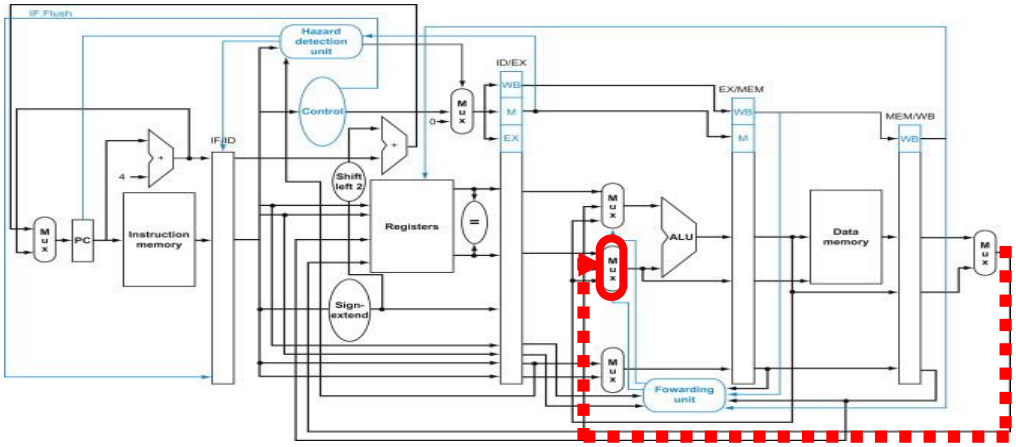


At which cycle(s) is this forwarding path enabled?
Assume full forwarding and bypassing.

insn\cycle	1	2	3	4	5	6	7	8	9	10	11	12	13
lw $t0	F	D	X	M	W								
lw $s1		F	D*	D	X	M	W						
add			F*	F	D*	D	X	M	W				

Data Forwarding

lw $t0, 0($t1)
lw $s1, 4($t0)
add $t2, $t3, $s1



At which cycle(s) is this forwarding path enabled?
Assume full forwarding and bypassing.

insn\cycle	1	2	3	4	5	6	7	8	9	10	11	12	13
lw $t0	F	D	X	M	W								
lw $s1		F	D*	D	X	M	W						
add			F*	F	D*	D	X	M	W				

Therefore, cycle 7.

Five-Stage Pipeline

Assume RF bypassing but no forwarding. Assume all control hazards are perfectly predicted and that you never have to flush. Consider this long-running loop, what is the average CPI?

```
LOOP:   lw $s0, 0($s1)
        sw $s0, 4($s1)
        add $t0, $t0, $s0
        addi $s1, $s1, 8
        bne $s0, $zero, LOOP
```

Five-Stage Pipeline

Assume RF bypassing but no forwarding. Assume all control hazards are perfectly predicted and that you never have to flush. Consider this long-running loop, what is the average CPI?

```
LOOP:   lw $s0, 0($s1)
        sw $s0, 4($s1)
        add $t0, $t0, $s0
        addi $s1, $s1, 8
        bne $s0, $zero, LOOP
```

no control hazard flushes!

Five-Stage Pipeline

Assume RF bypassing but no forwarding. Assume all control hazards are perfectly predicted and that you never have to flush. Consider this long-running loop, what is the average CPI?

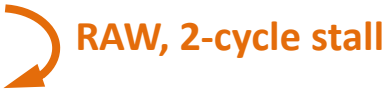
```
LOOP:   lw $s0, 0($s1)
        sw $s0, 4($s1)
        add $t0, $t0, $s0
        addi $s1, $s1, 8
        bne $s0, $zero, LOOP
```

CPI = 1?

Five-Stage Pipeline

Assume RF bypassing but no forwarding. Assume all control hazards are perfectly predicted and that you never have to flush. Consider this long-running loop, what is the average CPI?

```
LOOP:   lw $s0, 0($s1)
        sw $s0, 4($s1)
        add $t0, $t0, $s0
        addi $s1, $s1, 8
        bne $s0, $zero, LOOP
```



CPI = 1 + (2) / 5?

Five-Stage Pipeline

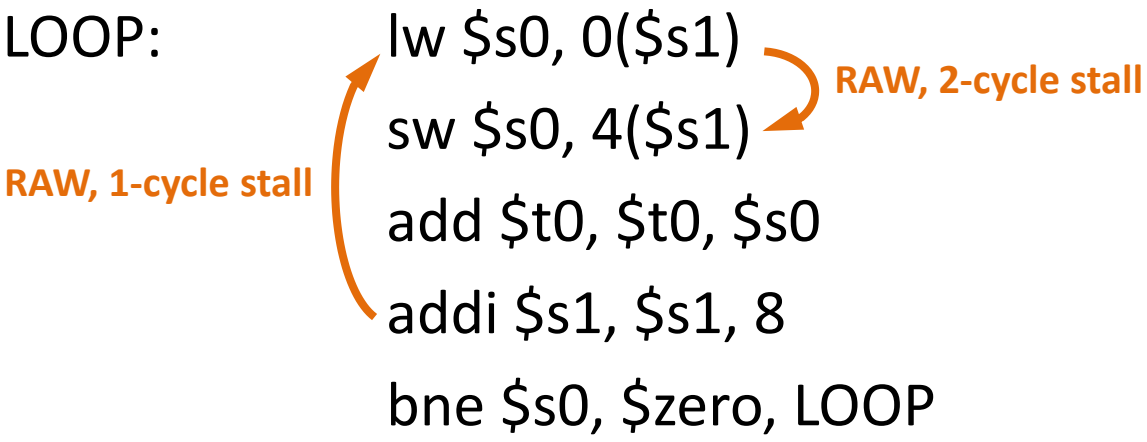
Assume RF bypassing but no forwarding. Assume all control hazards are perfectly predicted and that you never have to flush. Consider this long-running loop, what is the average CPI?

```
LOOP:   lw $s0, 0($s1)
        sw $s0, 4($s1)
        add $t0, $t0, $s0
        addi $s1, $s1, 8
        bne $s0, $zero, LOOP
```

CPI = 1 + (2) / 5?

Five-Stage Pipeline

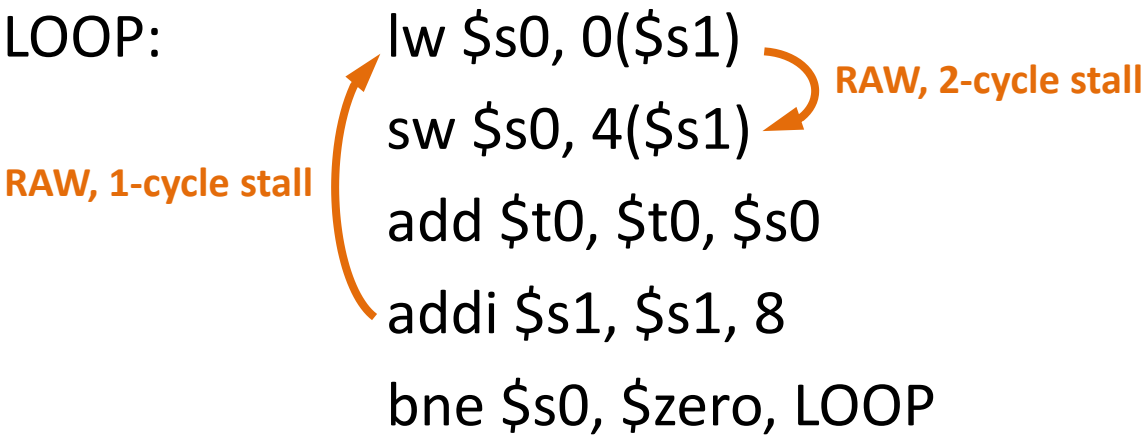
Assume RF bypassing but no forwarding. Assume all control hazards are perfectly predicted and that you never have to flush. Consider this long-running loop, what is the average CPI?



CPI = 1 + (2 + 1) / 5?

Five-Stage Pipeline

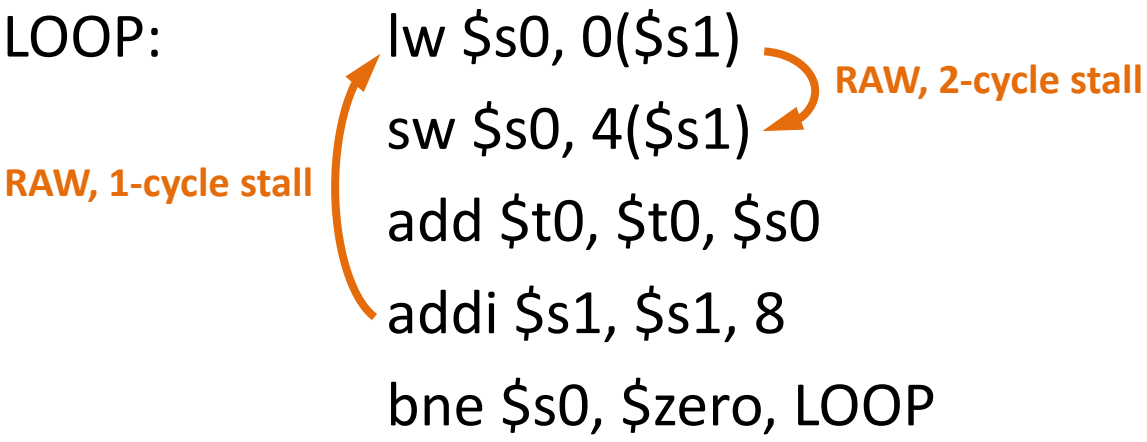
Assume RF bypassing but no forwarding. Assume all control hazards are perfectly predicted and that you never have to flush. Consider this long-running loop, what is the average CPI?



CPI = 1 + (2 + 1) / 5 = 1.6

Five-Stage Pipeline

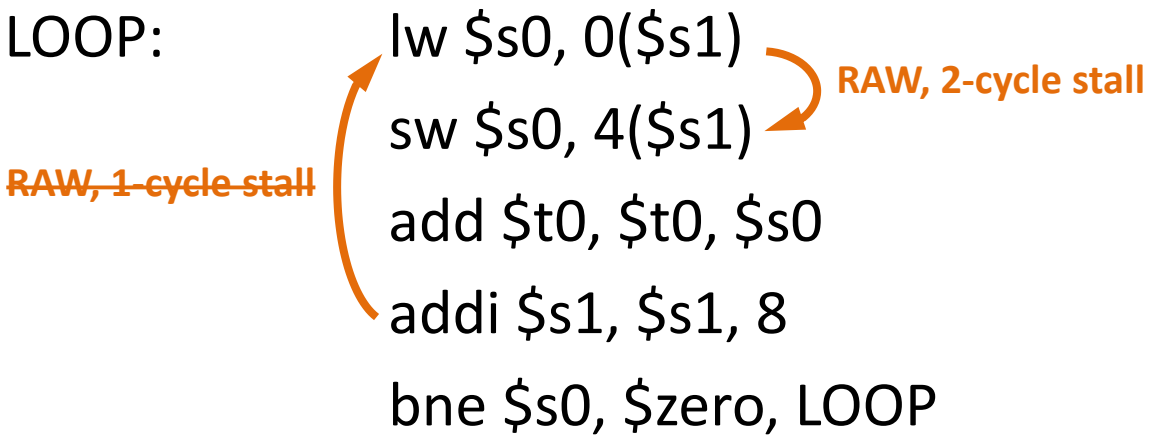
Assume forwarding and bypassing except MEM-to-MEM. Assume all control hazards are perfectly predicted and that you never have to flush. Consider this long-running loop, what is the average CPI?



$CPI_0 = 1 + (2 + 1) / 5 = 1.6$

Five-Stage Pipeline

Assume forwarding and bypassing except MEM-to-MEM. Assume all control hazards are perfectly predicted and that you never have to flush. Consider this long-running loop, what is the average CPI?



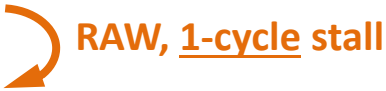
$CPI_0 = 1 + (2 + 1) / 5 = 1.6$

$CPI_1 = 1 + (2) / 5$

Five-Stage Pipeline

Assume forwarding and bypassing except MEM-to-MEM. Assume all control hazards are perfectly predicted and that you never have to flush. Consider this long-running loop, what is the average CPI?

```
LOOP:   lw $s0, 0($s1)
        sw $s0, 4($s1)
        add $t0, $t0, $s0
        addi $s1, $s1, 8
        bne $s0, $zero, LOOP
```



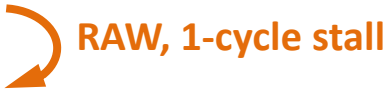
$CPI_0 = 1 + (2 + 1) / 5 = 1.6$

$CPI_1 = 1 + (1) / 5$

Five-Stage Pipeline

Assume forwarding and bypassing except MEM-to-MEM. Assume all control hazards are perfectly predicted and that you never have to flush. Consider this long-running loop, what is the average CPI?

```
LOOP:   lw $s0, 0($s1)
        sw $s0, 4($s1)
        add $t0, $t0, $s0
        addi $s1, $s1, 8
        bne $s0, $zero, LOOP
```



$CPI_0 = 1 + (2 + 1) / 5 = 1.6$

$CPI_1 = 1 + (1) / 5 = 1.2$

Five-Stage Pipeline

Assume forwarding and bypassing **with MEM-to-MEM**. Assume all control hazards are perfectly predicted and that you never have to flush. Consider this long-running loop, what is the average CPI?

```
LOOP:   lw $s0, 0($s1)
        sw $s0, 4($s1)
        add $t0, $t0, $s0
        addi $s1, $s1, 8
        bne $s0, $zero, LOOP
```

RAW, 1-cycle stall

$CPI_0 = 1 + (2 + 1) / 5 = 1.6$

$CPI_1 = 1 + (1) / 5 = 1.2$

Five-Stage Pipeline

Assume forwarding and bypassing with MEM-to-MEM. Assume all control hazards are perfectly predicted and that you never have to flush. Consider this long-running loop, what is the average CPI?

```
LOOP:   lw $s0, 0($s1)
        sw $s0, 4($s1)
        add $t0, $t0, $s0
        addi $s1, $s1, 8
        bne $s0, $zero, LOOP
```

$CPI_0 = 1 + (2 + 1) / 5 = 1.6$

$CPI_1 = 1 + (1) / 5 = 1.2$

$CPI_2 = 1$

Five-Stage Pipeline

Assume this long-running loop executes on an energy-harvesting device. Originally this code made up 75% of total runtime; the rest spent idle charging. What speedup does full forwarding offer?

```
LOOP:   lw $s0, 0($s1)
        sw $s0, 4($s1)
        add $t0, $t0, $s0
        addi $s1, $s1, 8
        bne $s0, $zero, LOOP
```

$CPI_0 = 1 + (2 + 1) / 5 = 1.6$

$CPI_1 = 1 + (1) / 5 = 1.2$

$CPI_2 = 1$

Five-Stage Pipeline

Assume this long-running loop executes on an energy-harvesting device. Originally this code made up 75% of total runtime; the rest spent idle charging. What speedup does full forwarding offer?

```
LOOP:   lw $s0, 0($s1)
        sw $s0, 4($s1)
        add $t0, $t0, $s0
        addi $s1, $s1, 8
        bne $s0, $zero, LOOP
```

$CPI_0 = 1 + (2 + 1) / 5 = 1.6$

$CPI_1 = 1 + (1) / 5 = 1.2$

$CPI_2 = 1$

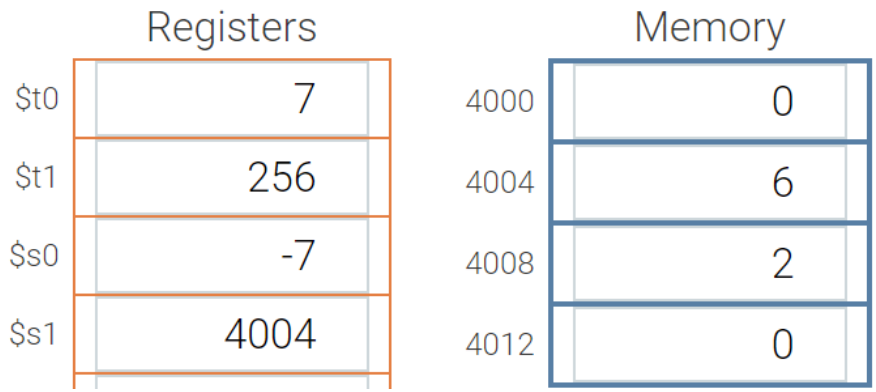
$Speedup_2 = 1 / (0.25 + 0.75/1.6)$

$= 1.39x$

Single-Cycle Datapath

There's a bug where Instruction[2] is always stuck at 1. Assuming the following register and memory contents before execution, what are [$t0], [$t1], [$s0], [$s1] after execution?

```
LOOP:   lw $t0, 0($s1)
        ori $s1, $s1, 8
        add $s0, $t0, $s0
        subu $t1, $t1, $t0
        nor $s0, $s0, $zero
        bne $s0, $zero, LOOP
```



Single-Cycle Datapath

There's a bug where Instruction[2] is always stuck at 1. Assuming the following register and memory contents before execution, what are [$t0], [$t1], [$s0], [$s1] after execution?

```
LOOP:   lw $t0, 0($s1)
        ori $s1, $s1, 8
        add $s0, $t0, $s0
        subu $t1, $t1, $t0
        nor $s0, $s0, $zero
        bne $s0, $zero, LOOP
```

Registers

$t0	7
$t1	256
$s0	-7
$s1	4004

Memory

4000	0
4004	6
4008	2
4012	0

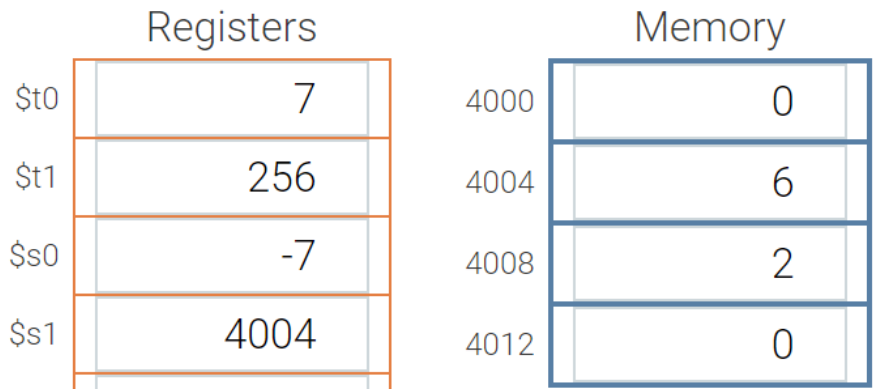
Correct execution:

$t0	6
$t1	250
$s0	0
$s1	4012

Single-Cycle Datapath

There's a bug where Instruction[2] is always stuck at 1. Assuming the following register and memory contents before execution, what are [$t0], [$t1], [$s0], [$s1] after execution?

```
LOOP:   lw $t0, 0($s1)
        ori $s1, $s1, 8
        add $s0, $t0, $s0
        subu $t1, $t1, $t0
        nor $s0, $s0, $zero
        bne $s0, $zero, LOOP
```

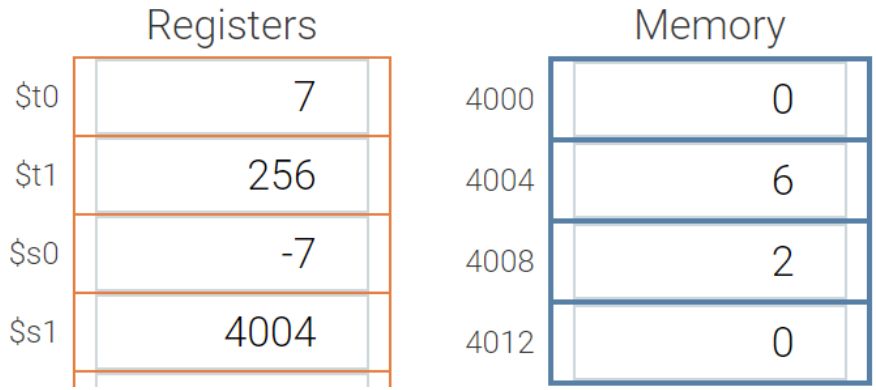


Incorrect execution:

Single-Cycle Datapath

There's a bug where Instruction[2] is always stuck at 1. Assuming the following register and memory contents before execution, what are [$t0], [$t1], [$s0], [$s1] after execution?

```
LOOP:   lw $t0, 4($s1)
        ori $s1, $s1, 8
        add $s0, $t0, $s0
        subu $t1, $t1, $t0
        nor $s0, $s0, $zero
        bne $s0, $zero, LOOP
```

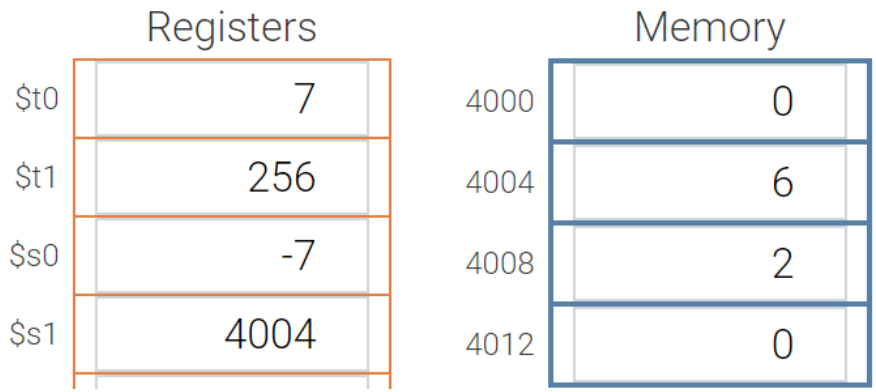


Incorrect execution:

Single-Cycle Datapath

There's a bug where Instruction[2] is always stuck at 1. Assuming the following register and memory contents before execution, what are [$t0], [$t1], [$s0], [$s1] after execution?

```
LOOP:   lw $t0, 4($s1)
        ori $s1, $s1, 12
        add $s0, $t0, $s0
        subu $t1, $t1, $t0
        nor $s0, $s0, $zero
        bne $s0, $zero, LOOP
```

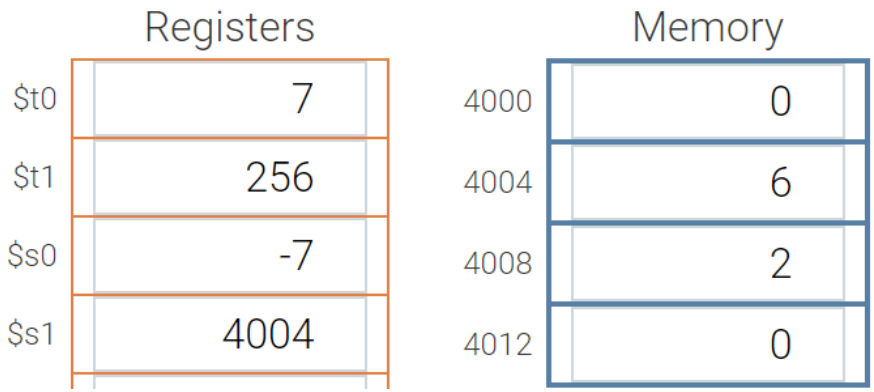


Incorrect execution:

Single-Cycle Datapath

There's a bug where Instruction[2] is always stuck at 1. Assuming the following register and memory contents before execution, what are [$t0], [$t1], [$s0], [$s1] after execution?

```
LOOP:   lw $t0, 4($s1)
        ori $s1, $s1, 12
        and $s0, $t0, $s0
        subu $t1, $t1, $t0
        nor $s0, $s0, $zero
        bne $s0, $zero, LOOP
```

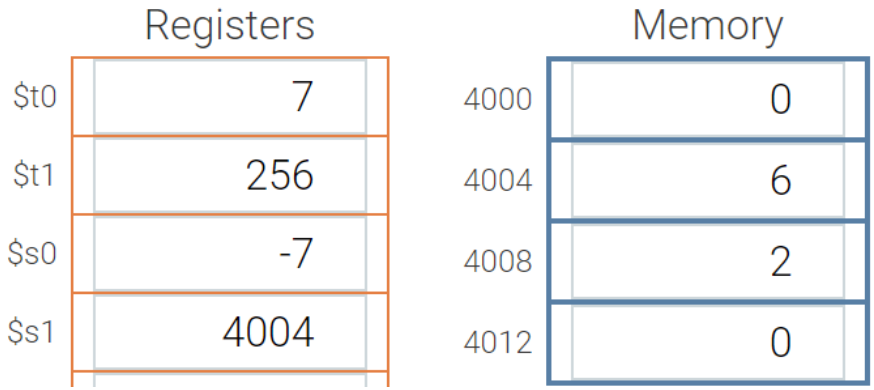


Incorrect execution:

Single-Cycle Datapath

There's a bug where Instruction[2] is always stuck at 1. Assuming the following register and memory contents before execution, what are [$t0], [$t1], [$s0], [$s1] after execution?

```
LOOP:   lw $t0, 4($s1)
        ori $s1, $s1, 12
        and $s0, $t0, $s0
        nor $t1, $t1, $t0
        nor $s0, $s0, $zero
        bne $s0, $zero, LOOP
```

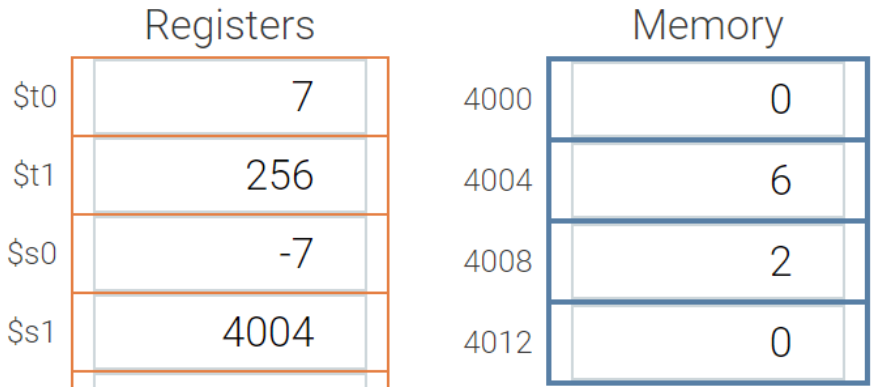


Incorrect execution:

Single-Cycle Datapath

There's a bug where Instruction[2] is always stuck at 1. Assuming the following register and memory contents before execution, what are [$t0], [$t1], [$s0], [$s1] after execution?

```
        lw $t0, 4($s1)
        ori $s1, $s1, 12
        and $s0, $t0, $s0
        nor $t1, $t1, $t0
LOOP:   nor $s0, $s0, $zero
        bne $s0, $zero, LOOP
```



Incorrect execution:

Single-Cycle Datapath

There's a bug where Instruction[2] is always stuck at 1. Assuming the following register and memory contents before execution, what are [$t0], [$t1], [$s0], [$s1] after execution?

```
        lw $t0, 4($s1)
        ori $s1, $s1, 12
        and $s0, $t0, $s0
        nor $t1, $t1, $t0
LOOP:   nor $s0, $s0, $zero
        bne $s0, $zero, LOOP
```

Registers

$t0	7
$t1	256
$s0	-7
$s1	4004

Memory

4000	0
4004	6
4008	2
4012	0

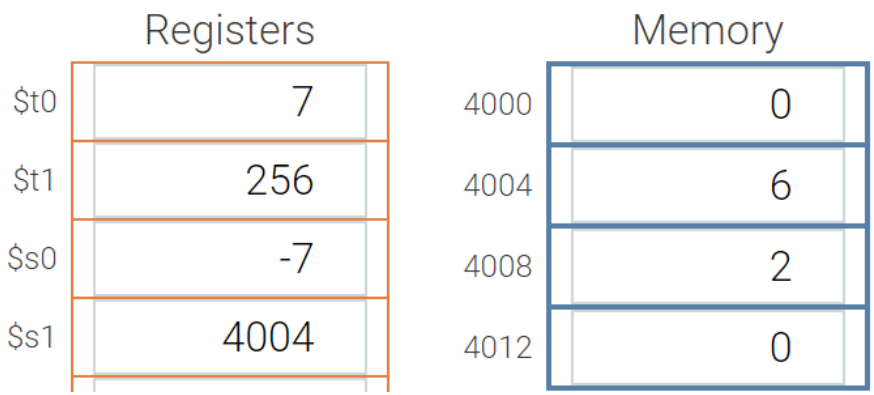
Incorrect execution:

$t0	2
$t1	-259
$s0	0
$s1	4012

Pipeline Data Hazards

You enable RF bypassing but not forwarding/stalling. Assume branches resolved in ID. Given register and memory contents before execution, what are [$t0], [$t1], [$s0], [$s1] after execution?

```
LOOP:   lw $t0, 0($s1)
        ori $s1, $s1, 8
        add $s0, $t0, $s0
        subu $t1, $t1, $t0
        nor $s0, $s0, $zero
        bne $s0, $zero, LOOP
```



Pipeline Data Hazards

You enable RF bypassing but not forwarding/stalling. Assume branches resolved in ID. Given register and memory contents before execution, what are [$t0], [$t1], [$s0], [$s1] after execution?

```
LOOP:   lw $t0, 0($s1)
        ori $s1, $s1, 8
        add $s0, $t0, $s0
        subu $t1, $t1, $t0
        nor $s0, $s0, $zero
        bne $s0, $zero, LOOP
```

Registers

$t0	7
$t1	256
$s0	-7
$s1	4004

Memory

4000	0
4004	6
4008	2
4012	0

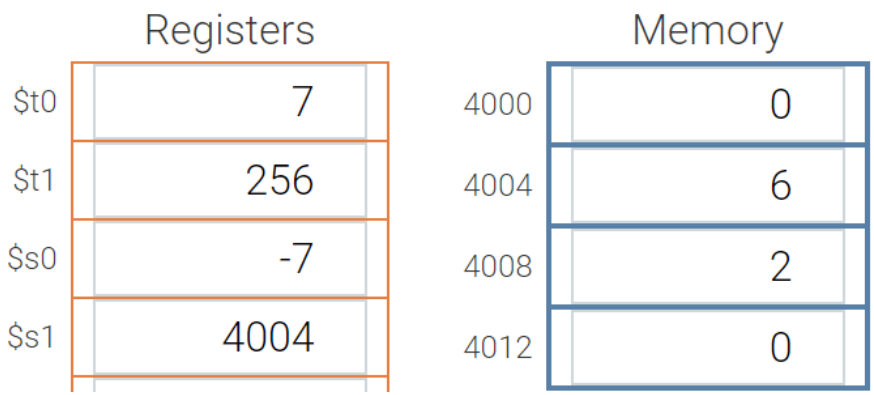
Correct execution:

$t0	6
$t1	250
$s0	0
$s1	4012

Pipeline Data Hazards

You enable RF bypassing but not forwarding/stalling. Assume branches resolved in ID. Given register and memory contents before execution, what are [$t0], [$t1], [$s0], [$s1] after execution?

```
LOOP:   lw $t0, 0($s1)
        ori $s1, $s1, 8
        add $s0, $t0, $s0
        subu $t1, $t1, $t0
        nor $s0, $s0, $zero
        bne $s0, $zero, LOOP
```



Incorrect execution:

Pipeline Data Hazards

You enable RF bypassing but not forwarding/stalling. Assume branches resolved in ID. Given register and memory contents before execution, what are [$t0], [$t1], [$s0], [$s1] after execution?

1: lw $t0, 0($s1)

2: ori $s1, $s1, 8

3: add $s0, $t0, $s0

4: subu $t1, $t1, $t0

5: nor $s0, $s0, $zero

6: bne $s0, $zero, LOOP

0: Registers

$t0	7
$t1	256
$s0	-7
$s1	4004

Memory

4000	0
4004	6
4008	2
4012	0

Incorrect execution:

Pipeline Data Hazards

You enable RF bypassing but not forwarding/stalling. Assume branches resolved in ID. Given register and memory contents before execution, what are [$t0], [$t1], [$s0], [$s1] after execution?

1: lw $t0, 0($s1)

2: ori $s1, $s1, 8

3: add $s0, $t0**.0**, $s0

4: subu $t1, $t1, $t0

5: nor $s0, $s0, $zero

6: bne $s0, $zero, LOOP

0: Registers

$t0	7
$t1	256
$s0	-7
$s1	4004

Memory

4000	0
4004	6
4008	2
4012	0

Incorrect execution:

Pipeline Data Hazards

You enable RF bypassing but not forwarding/stalling. Assume branches resolved in ID. Given register and memory contents before execution, what are [$t0], [$t1], [$s0], [$s1] after execution?

1: lw $t0, 0($s1)

2: ori $s1, $s1, 8

3: add $s0, $t0**.0**, $s0

4: subu $t1, $t1, $t0

5: nor $s0, $s0**.0**, $zero

6: bne $s0, $zero, LOOP

0: Registers

$t0	7
$t1	256
$s0	-7
$s1	4004

Memory

4000	0
4004	6
4008	2
4012	0

Incorrect execution:

Pipeline Data Hazards

You enable RF bypassing but not forwarding/stalling. Assume branches resolved in ID. Given register and memory contents before execution, what are [$t0], [$t1], [$s0], [$s1] after execution?

1: lw $t0, 0($s1)

2: ori $s1, $s1, 8

3: add $s0, $t0**.0**, $s0

4: subu $t1, $t1, $t0

5: nor $s0, $s0**.0**, $zero

6: bne $s0**.3**, $zero, LOOP

0: Registers

$t0	7
$t1	256
$s0	-7
$s1	4004

Memory

4000	0
4004	6
4008	2
4012	0

Incorrect execution:

Pipeline Data Hazards

You enable RF bypassing but not forwarding/stalling. Assume branches resolved in ID. Given register and memory contents before execution, what are [$t0], [$t1], [$s0], [$s1] after execution?

1: lw $t0, 0($s1)

2: ori $s1, $s1, 8

3: add $s0, $t0**.0**, $s0

4: subu $t1, $t1, $t0

5: nor $s0, $s0**.0**, $zero

6: bne $s0**.3**, $zero, LOOP

0: Registers

$t0	7
$t1	256
$s0	-7
$s1	4004

Memory

4000	0
4004	6
4008	2
4012	0

Incorrect execution:

$t0	6
$t1	250
$s0	6
$s1	4012

Pipeline Diagrams

Assume forwarding and bypassing with MEM-to-MEM. Employ predict-not-taken policy and resolve branches in ID (with EX forwarding). Complete the pipeline diagram.

```
        lw $s0, 0($s0)
        sw $s0, 0($t0)
        bne $s0, $zero, SKIP        # taken
        lw $t1, 4($s0)
SKIP:   sw $s1, 0($t1)
```

insn\cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14
lw $s0, 0($s0)	F	D	X	M	W									

Pipeline Diagrams

Assume forwarding and bypassing with MEM-to-MEM. Employ predict-not-taken policy and resolve branches in ID (with EX forwarding). Complete the pipeline diagram.

```
                lw $s0, 0($s0)
                sw $s0, 0($t0)
                bne $s0, $zero, SKIP        # taken
                lw $t1, 4($s0)
SKIP:           sw $s1, 0($t1)
```

insn\cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14
lw $s0, 0($s0)	F	D	X	M	W									
sw $s0, 0($t0)		**F**	**D**	**X**	**M**	**W**								

Pipeline Diagrams

Assume forwarding and bypassing with MEM-to-MEM. Employ predict-not-taken policy and resolve branches in ID (with EX forwarding). Complete the pipeline diagram.

```
                lw $s0, 0($s0)
                sw $s0, 0($t0)
                bne $s0, $zero, SKIP        # taken
                lw $t1, 4($s0)
SKIP:           sw $s1, 0($t1)
```

insn\cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14
lw $s0, 0($s0)	F	D	X	M	W									
sw $s0, 0($t0)		**F**	**D**	**X**	**M**	**W**								
bne $s0, $zero, SKIP			**F**	**D***	**D**	**X**	**M**	**W**						

Pipeline Diagrams

Assume forwarding and bypassing with MEM-to-MEM. Employ predict-not-taken policy and resolve branches in ID (with EX forwarding). Complete the pipeline diagram.

```
                lw $s0, 0($s0)
                sw $s0, 0($t0)
                bne $s0, $zero, SKIP        # taken
                lw $t1, 4($s0)
SKIP:           sw $s1, 0($t1)
```

insn\cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14
lw $s0, 0($s0)	F	D	X	M	W									
sw $s0, 0($t0)		**F**	**D**	**X**	**M**	**W**								
bne $s0, $zero, SKIP			**F**	**D***	**D**	**X**	**M**	**W**						
lw $t1, 4($s0)				**F***	**F**	**=**								

Pipeline Diagrams

Assume forwarding and bypassing with MEM-to-MEM. Employ predict-not-taken policy and resolve branches in ID (with EX forwarding). Complete the pipeline diagram.

```
                lw $s0, 0($s0)
                sw $s0, 0($t0)
                bne $s0, $zero, SKIP        # taken
                lw $t1, 4($s0)
SKIP:           sw $s1, 0($t1)
```

insn\cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14
lw $s0, 0($s0)	F	D	X	M	W									
sw $s0, 0($t0)		**F**	**D**	**X**	**M**	**W**								
bne $s0, $zero, SKIP			**F**	**D***	**D**	**X**	**M**	**W**						
lw $t1, 4($s0)				**F***	**F**	**=**								
sw $s1, 0($t1)						**F**	**D**	**X**	**M**	**W**				

Pipeline Diagrams

Assume forwarding and bypassing with MEM-to-MEM. Employ predict-not-taken policy and resolve branches in ID (with EX forwarding). Complete the pipeline diagram.

```
LOOP:   lw $s0, 0($s0)
        bne $s0, $zero, LOOP  # taken then not taken
        lw $t0, 4($s0)
        sw $s1, 0($t0)
```

insn\cycle	1	2	3	4	5	6	7	8	9	10	11	12
lw $s0, 0($s0)	F	D	X	M	W							

Pipeline Diagrams

Assume forwarding and bypassing with MEM-to-MEM. Employ predict-not-taken policy and resolve branches in ID (with EX forwarding). Complete the pipeline diagram.

```
LOOP:   lw $s0, 0($s0)
        bne $s0, $zero, LOOP  # taken then not taken
        lw $t0, 4($s0)
        sw $s1, 0($t0)
```

insn\cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
lw $s0, 0($s0)	F	D	X	M	W											

Pipeline Diagrams

Assume forwarding and bypassing with MEM-to-MEM. Employ predict-not-taken policy and resolve branches in ID (with EX forwarding). Complete the pipeline diagram.

```
LOOP:   lw $s0, 0($s0)
        bne $s0, $zero, LOOP  # taken then not taken
        lw $t0, 4($s0)
        sw $s1, 0($t0)
```

insn\cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
lw $s0, 0($s0)	F	D	X	M	W											
bne $s0, $zero, LOOP		**F**	**D***	**D***	**D**	**X**	**M**	**W**								

Pipeline Diagrams

Assume forwarding and bypassing with MEM-to-MEM. Employ predict-not-taken policy and resolve branches in ID (with EX forwarding). Complete the pipeline diagram.

```
LOOP:   lw $s0, 0($s0)
        bne $s0, $zero, LOOP  # taken then not taken
        lw $t0, 4($s0)
        sw $s1, 0($t0)
```

insn\cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
lw $s0, 0($s0)	F	D	X	M	W											
bne $s0, $zero, LOOP		**F**	**D***	**D***	**D**	**X**	**M**	**W**								
lw $t0, 4($s0)			**F***	**F***	**F**	**=**										

Pipeline Diagrams

Assume forwarding and bypassing with MEM-to-MEM. Employ predict-not-taken policy and resolve branches in ID (with EX forwarding). Complete the pipeline diagram.

```
LOOP:   lw $s0, 0($s0)
        bne $s0, $zero, LOOP  # taken then not taken
        lw $t0, 4($s0)
        sw $s1, 0($t0)
```

insn\cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
lw $s0, 0($s0)	F	D	X	M	W											
bne $s0, $zero, LOOP		**F**	**D***	**D***	**D**	**X**	**M**	**W**								
lw $t0, 4($s0)			**F***	**F***	**F**	**=**										
lw $s0, 0($s0)						**F**	**D**	**X**	**M**	**W**						

Pipeline Diagrams

Assume forwarding and bypassing with MEM-to-MEM. Employ predict-not-taken policy and resolve branches in ID (with EX forwarding). Complete the pipeline diagram.

```
LOOP:   lw $s0, 0($s0)
        bne $s0, $zero, LOOP  # taken then not taken
        lw $t0, 4($s0)
        sw $s1, 0($t0)
```

insn\cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
lw $s0, 0($s0)	F	D	X	M	W											
bne $s0, $zero, LOOP		F	D*	D*	D	X	M	W								
lw $t0, 4($s0)			F*	F*	F	=										
lw $s0, 0($s0)						F	D	X	M	W						
bne $s0, $zero, LOOP							F	D*	D*	D	X	M	W			

Pipeline Diagrams

Assume forwarding and bypassing with MEM-to-MEM. Employ predict-not-taken policy and resolve branches in ID (with EX forwarding). Complete the pipeline diagram.

```
LOOP:   lw $s0, 0($s0)
        bne $s0, $zero, LOOP  # taken then not taken
        lw $t0, 4($s0)
        sw $s1, 0($t0)
```

insn\cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
lw $s0, 0($s0)	F	D	X	M	W											
bne $s0, $zero, LOOP		F	D*	D*	D	X	M	W								
lw $t0, 4($s0)			F*	F*	F	=										
lw $s0, 0($s0)						F	D	X	M	W						
bne $s0, $zero, LOOP							F	D*	D*	D	X	M	W			
lw $t0, 4($s0)								F*	F*	F	D	X	M	W		

Pipeline Diagrams

Assume forwarding and bypassing with MEM-to-MEM. Employ predict-not-taken policy and resolve branches in ID (with EX forwarding). Complete the pipeline diagram.

```
LOOP:   lw $s0, 0($s0)
        bne $s0, $zero, LOOP  # taken then not taken
        lw $t0, 4($s0)
        sw $s1, 0($t0)
```

insn\cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
lw $s0, 0($s0)	F	D	X	M	W											
bne $s0, $zero, LOOP		F	D*	D*	D	X	M	W								
lw $t0, 4($s0)			F*	F*	F	=										
lw $s0, 0($s0)						F	D	X	M	W						
bne $s0, $zero, LOOP							F	D*	D*	D	X	M	W			
lw $t0, 4($s0)								F*	F*	F	D	X	M	W		
sw $s1, 0($t0)											F	D*	D	X	M	W