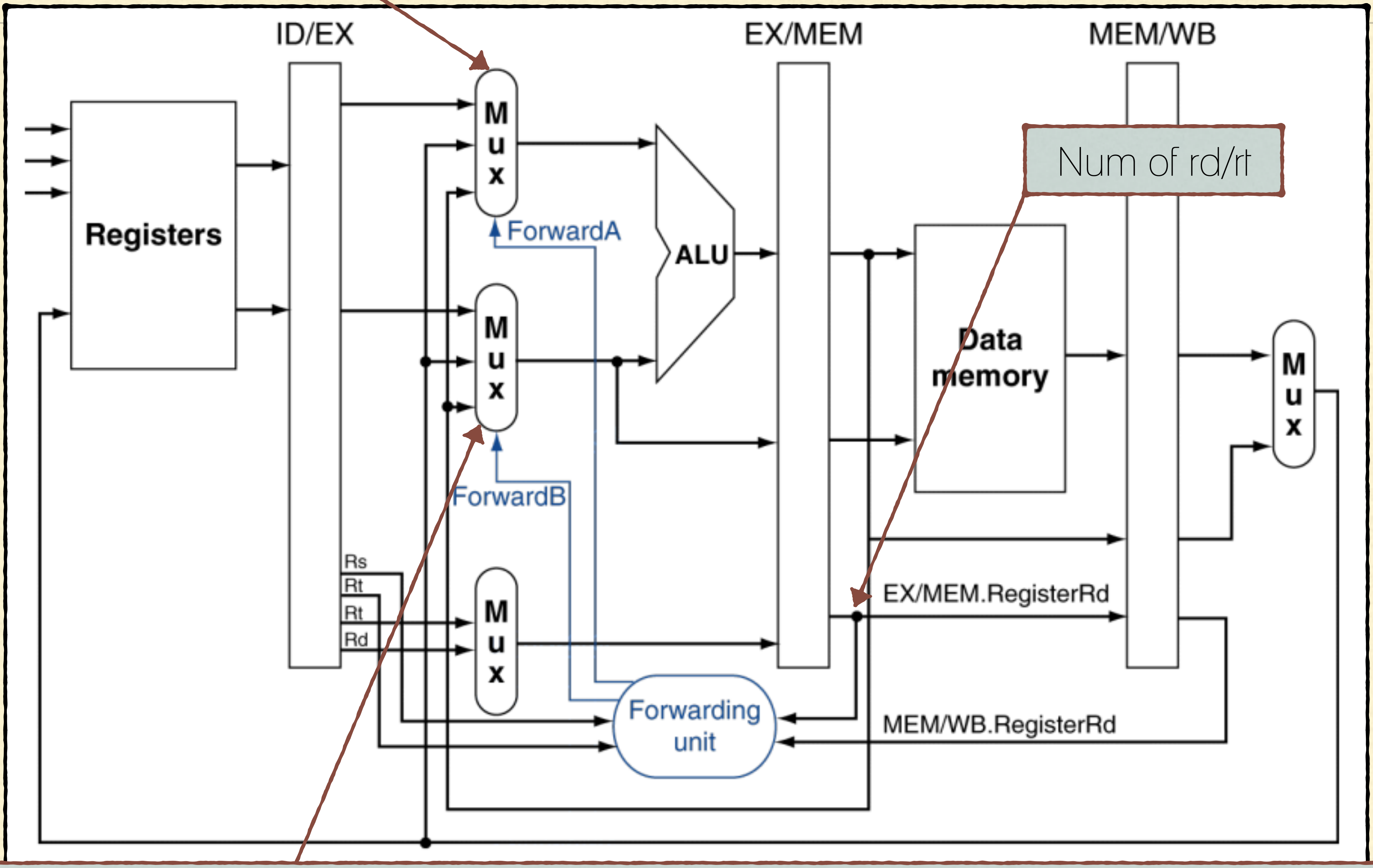
PIPELINING HAZARDS PART II

Ayman Hajja, PhD

FORWARDING UNIT

Three different input possibilities: rs, output of ALU, & output of accessing memory



Num of rd/rt

Three different input possibilities: rt, output of ALU, & output of accessing memory

DATA HAZARD (LOADS)

- Can't solve all cases with forwarding
    - 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)

EXAMPLE OF DATA HAZARD

For $t1, No problem here!

Can be solved by forwarding!

	Stages								
lw $t1, 0($t0)	IF	ID	EX	MEM	WB				
lw $t2, 4($t0)		IF	ID	EX	MEM	WB			
add $t3, $t1, $t2			IF	ID	EX	MEM	WB		
...				IF	ID	EX	MEM	WB	
...					IF	ID	EX	MEM	WB

EXAMPLE OF DATA HAZARD

For $t2, we have a BIG PROBLEM

CAN'T GO BACK IN TIME

	Stages								
lw $t1, 0($t0)	IF	ID	EX	MEM	WB				
lw $t2, 4($t0)		IF	ID	EX	MEM	WB			
add $t3, $t1, $t2			IF	ID	EX	MEM	WB		
. . .				IF	ID	EX	MEM	WB	
. . .					IF	ID	EX	MEM	WB

EXAMPLE OF DATA HAZARD

BIG PROBLEM — CAN'T GO BACK IN TIME

	Stages								
lw $t1, 0($t0)	IF	ID	EX	MEM	WB				
lw $t2, 4($t0)		IF	ID	EX	MEM	WB			
add $t3, $t1, $t2			IF	ID	EX	MEM	WB		
. . .				IF	ID	EX	MEM	WB	
. . .					IF	ID	EX	MEM	WB

- Can't solve all cases with forwarding
    - Must stall instructions dependent on load

DATA HAZARD

- Hardware stalls pipeline
    - Called “hardware interlock”

	Stages									
lw $t1, 0($t0)	IF	ID	EX	MEM	WB					
lw $t2, 4($t0)		IF	ID	EX	MEM	WB				
add $t3, $t1, $t2			IF	ID	🟥	EX	MEM	WB		
…				IF	🟥	ID	EX	MEM	WB	
…					🟥	IF	ID	EX	ME	WB

- Creates a “bubble” in the pipeline

DATA HAZARD

- Slot after load is called a load delay slot
    - If that instruction uses the result of the load, then the hardware interlock will stall it for one cycle
- What does MIPS stand for?

DATA HAZARD

- Slot after load is called a load delay slot
  - If that instruction uses the result of the load, then the hardware interlock will stall it for one cycle

- What does MIPS stand for?
  - Microprocessor <u>without</u> Interlocked Pipeline Stages

DATA HAZARD

- One solution would be to explicitly add a no-op (or nop) in the delay slot

DATA HAZARD

- Can’t solve all cases with forwarding
    - lw $t1, 0($t0)
    - lw $t2, 4($t0)
        - no-op (add $0, $0, $0)
    - add $t3, $t1, $t2
    - sw $t3, 12($t0)
    - lw $t4, 8($t0)
        - no-op (add $0, $0, $0)
    - add $t5, $t1, $t4
    - sw $t5, 16($t0)

DATA HAZARD

- Better idea than "nop"; Let the compiler put an unrelated instruction in that slot — No stall

CODE SCHEDULING TO AVOID STALLS

- MIPS code for A[3] = A[0] + A[1]; A[4] = A[0]+ A[2];
  - 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)

Address of A is stored in $t0

Will there be stall, and where?

CODE SCHEDULING TO AVOID STALLS

- MIPS code for A[3] = A[0] + A[1]; A[4] = A[0]+ A[2];
    - lw $t1, 0($t0)
    - lw $t2, 4($t0)
        - Here
    - add $t3, $t1, $t2
    - sw $t3, 12($t0)
    - lw $t4, 8($t0)
        - Here
    - add $t5, $t1, $t4
    - sw $t5, 16($t0)

How many CPU cycles will this need to finish executing?

CODE SCHEDULING TO AVOID STALLS

- MIPS code for A[3] = A[0] + A[1]; A[4] = A[0]+ A[2];
    - lw $t1, 0($t0)
    - lw $t2, 4($t0)
        - Here
    - add $t3, $t1, $t2
    - sw $t3, 12($t0)
    - lw $t4, 8($t0)
        - Here
    - add $t5, $t1, $t4
    - sw $t5, 16($t0)

How many CPU cycles will this need to finish executing?

13

CODE SCHEDULING TO AVOID STALLS

- MIPS code for A[3] = A[0] + A[1]; A[4] = A[0]+ A[2];
  - lw $t1, 0($t0)
  - lw $t2, 4($t0)
    - Here
  - add $t3, $t1, $t2
  - sw $t3, 12($t0)
  - lw $t4, 8($t0)
    - Here
  - add $t5, $t1, $t4
  - sw $t5, 16($t0)

CODE SCHEDULING TO AVOID STALLS

- Reorder code to avoid use of load result in the next instruction!
- MIPS code for A[3] = A[0] + A[1]; A[4] = A[0]+ A[2];
    - lw $t1, 0($t0)
    - lw $t2, 4($t0)
        - Here
    - add $t3, $t1, $t2
    - sw $t3, 12($t0)
    - lw $t4, 8($t0)
        - Here
    - add $t5, $t1, $t4
    - sw $t5, 16($t0)

Yes. Here's one solution

Method 2:

```
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)
```

CODE SCHEDULING TO AVOID STALLS

- Reorder code to avoid use of load result in the next instruction!
- MIPS code for A[3] = A[0] + A[1]; A[4] = A[0]+ A[2];
    - lw $t1, 0($t0)
    - lw $t2, 4($t0)
        - Here
    - add $t3, $t1, $t2
    - sw $t3, 12($t0)
    - lw $t4, 8($t0)
        - Here
    - add $t5, $t1, $t4
    - sw $t5, 16($t0)

No Stall — How many cycles?

Method 2:

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)

CODE SCHEDULING TO AVOID STALLS

- Reorder code to avoid use of load result in the next instruction!
- MIPS code for A[3] = A[0] + A[1]; A[4] = A[0]+ A[2];
    - lw $t1, 0($t0)
    - lw $t2, 4($t0)
        - Here
    - add $t3, $t1, $t2
    - sw $t3, 12($t0)
    - lw $t4, 8($t0)
        - Here
    - add $t5, $t1, $t4
    - sw $t5, 16($t0)

No Stall — How many cycles?

11

Method 2:

lw $t1, 0($t0)

lw $t2, 4($t0)

lw $t4, 8($t0)

add $t3, $t1, $t2

sw $t3, 12($t0

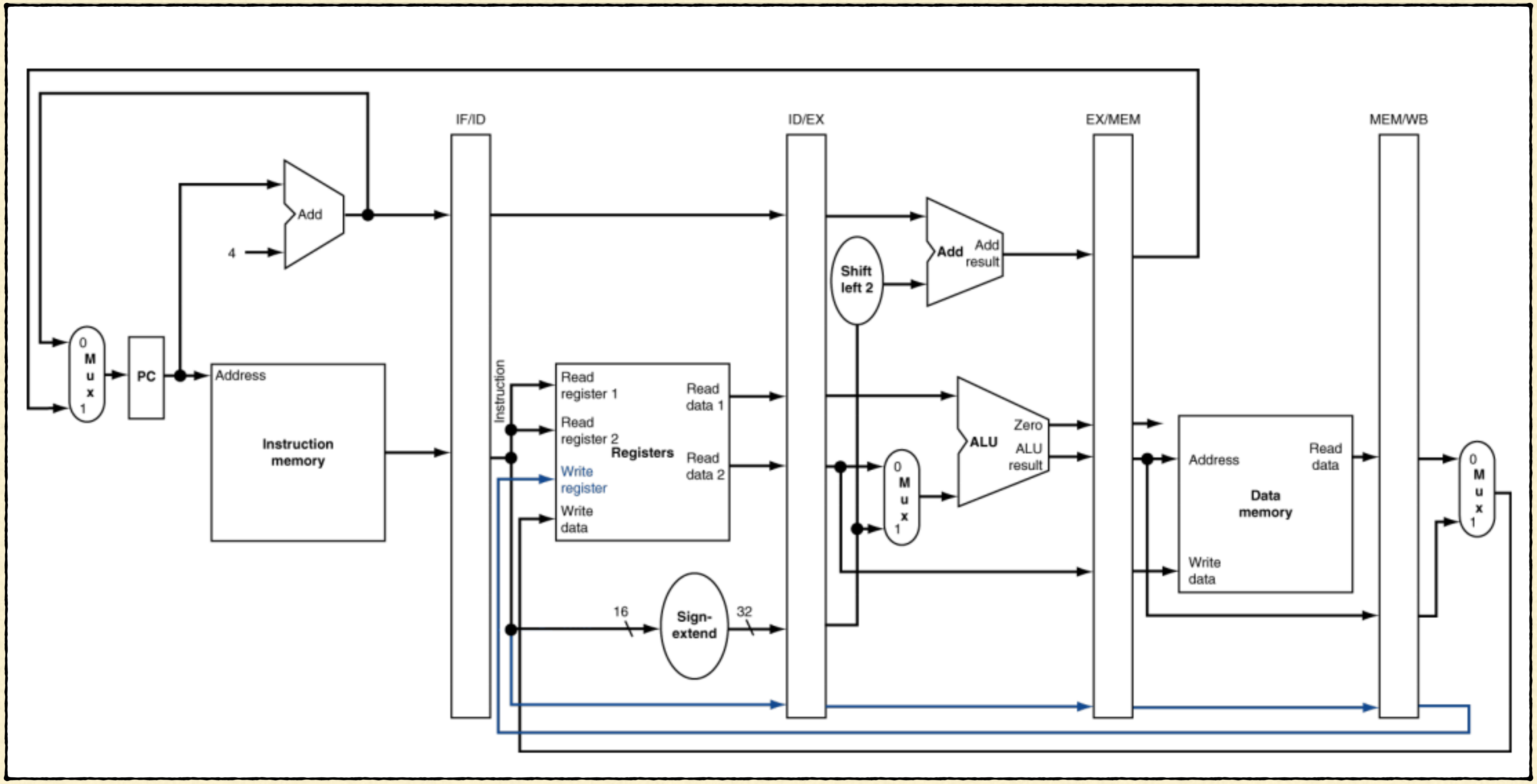
add $t5, $t1, $t4

sw $t5, 16($t0)

CONTROL HAZARD

- Branch determined flow of control:
    - Fetching next instruction depends on branch outcome
    - Pipeline can't always fetch correct instruction
        - Still working on ID stage of branch
- Simple solution Option 1: Stall on every branch until branch condition resolved:
    - Would add 2 bubbles/clock cycles for every branch

RECALL DATAPATH OF BRANCH



CONTROL HAZARD

	Stages								
beq	IF	ID	EX	MEM	WB				
Instruction 2		IF	ID	EX	MEM	WB			
Instruction 3			IF	ID	EX	MEM	WB		
Instruction 4				IF	ID	EX	MEM	WB	
...					IF	ID	EX	MEM	WB

- Where (in which stage) do we compare for the branch?

CONTROL HAZARD

By the time we branch, there are two instruction already in the pipeline

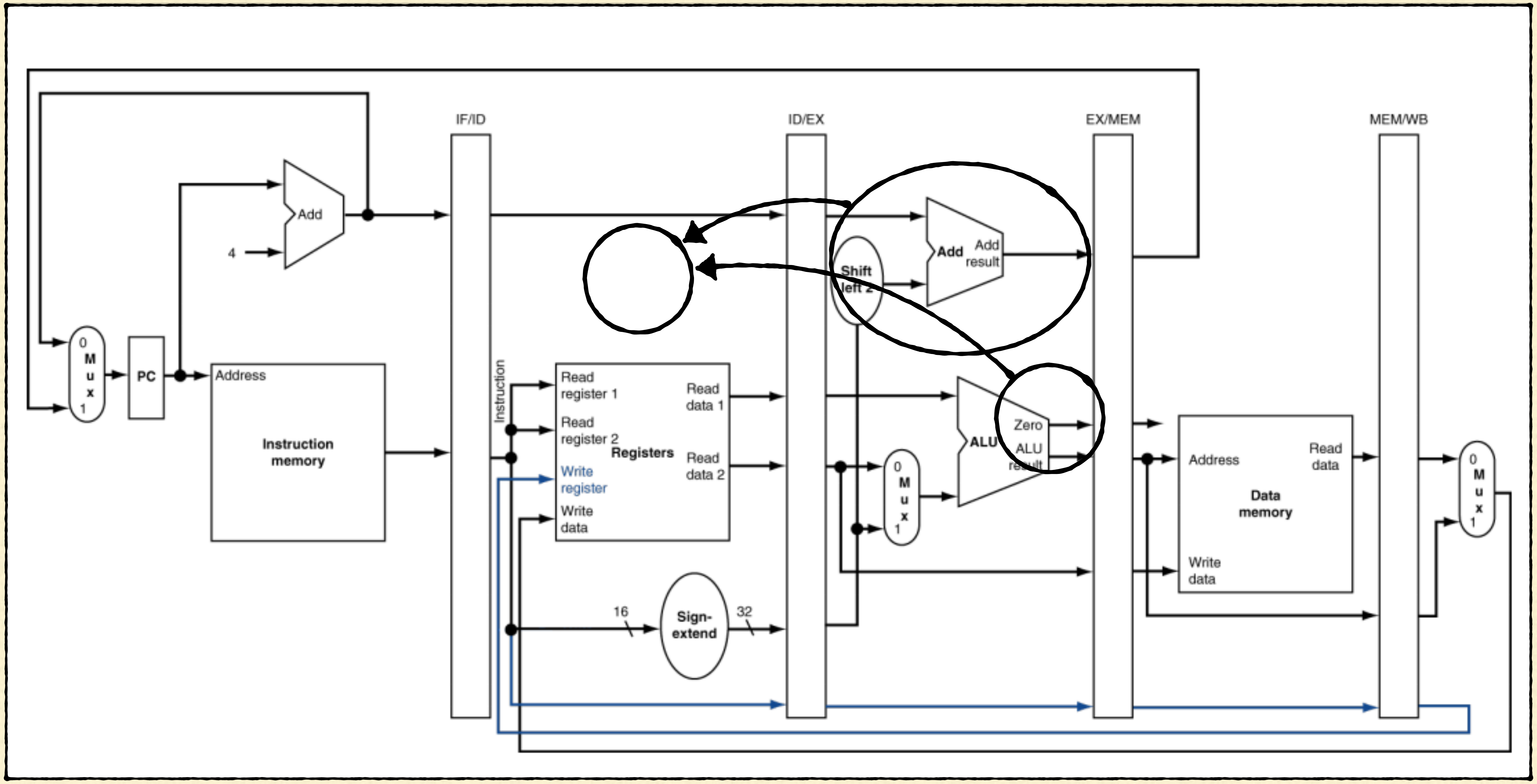
	Stages								
beq	IF	ID	EX	MEM	WB				
Instruction 2		IF	ID	EX	MEM	WB			
Instruction 3			IF	ID	EX	MEM	WB		
Instruction 4				IF	ID	EX	MEM	WB	
...					IF	ID	EX	MEM	WB

- Where (in which stage) do we compare for the branch?
    - ALU

CONTROL HAZARD

- Optimization idea:
    - Insert special branch comparator in Stage 2
    - As soon as instruction is decoded (controller identifies it's a branch), we compare 'rs' with 'rt' and we immediately make a decision and set the new value of the PC
    - Benefit: since branch is completed in Stage 2, only one unnecessary instruction is fetched

RECALL DATAPATH OF BRANCH



CONTROL HAZARD

After moving the comparator to the decode stage:
By the time we branch, there is only one instruction already in the pipeline

	Stages								
beq	IF	ID	EX	MEM	WB				
Instruction 2		IF	ID	EX	MEM	WB			
Instruction 3			IF	ID	EX	MEM	WB		
Instruction 4				IF	ID	EX	MEM	WB	
…					IF	ID	EX	MEM	WB

- Branch comparator moved to Decode Stage

CONTROL HAZARD

- Option 1: Predict outcome of a branch, fix up if goes wrong
  - Must cancel all instructions in pipeline that depends on guess that was wrong
  - This is called “flushing” the pipeline

CONTROL HAZARD

- Option 2: Redefine branches
  - Old definition: if we take the branch, none of the instructions after the branch get executed by accident
  - New definition: whether or not we take the branch, the single instruction immediately following the branch gets executed (the branch-delay slot)
- Delayed Branch means we always execute instruction after branch
- This optimization is used with MIPS (not MARS simulator)

EXAMPLE

- Non-delayed Branch:
    - or $8, $9, $10
    - add $1, $2, $3
    - sub $4, $5, $6
    - beq $1, $4, Exit
    - xor $10, $1, $11
    - Exit:

EXAMPLE

- Non-delayed Branch:
    - or $8, $9, $10
    - add $1, $2, $3
    - sub $4, $5, $6
    - beq $1, $4, Exit
    - xor $10, $1, $11
    - Exit:

- Delayed Branch:
    - add $1, $2, $3
    - sub $4, $5, $6
    - beq $1, $4, Exit
    - or $8, $9, $10
    - xor $10, $1, $11
    - Exit:

CONTROL HAZARD

- Notes on Branch-Delay Slot
  - Worst-case: put a no-op (nop) in the branch-delay slop
  - Better-case: place some instruction preceding the branch in the branch-delay slot — as long as the change doesn't affect the logic of program
    - Re-ordering instructions is common way to speed up programs

IN CONCLUSION

- Pipe stages should be balanced for highest clock rate
- Three types of pipeline hazard:
  - Structural (conflict for use of a resource)
  - Data (use interlocks or bypassing to resolve)
  - Control (reduce impact with branch prediction or branch delay slots)