Control Hazards

Branches disrupt the flow of instructions in the pipeline

- The branch target address is computed in stage 3
- The branch condition is evaluated in stage 3
- The branch takes effect in stage 4

Instructions behind a taken branch must not complete

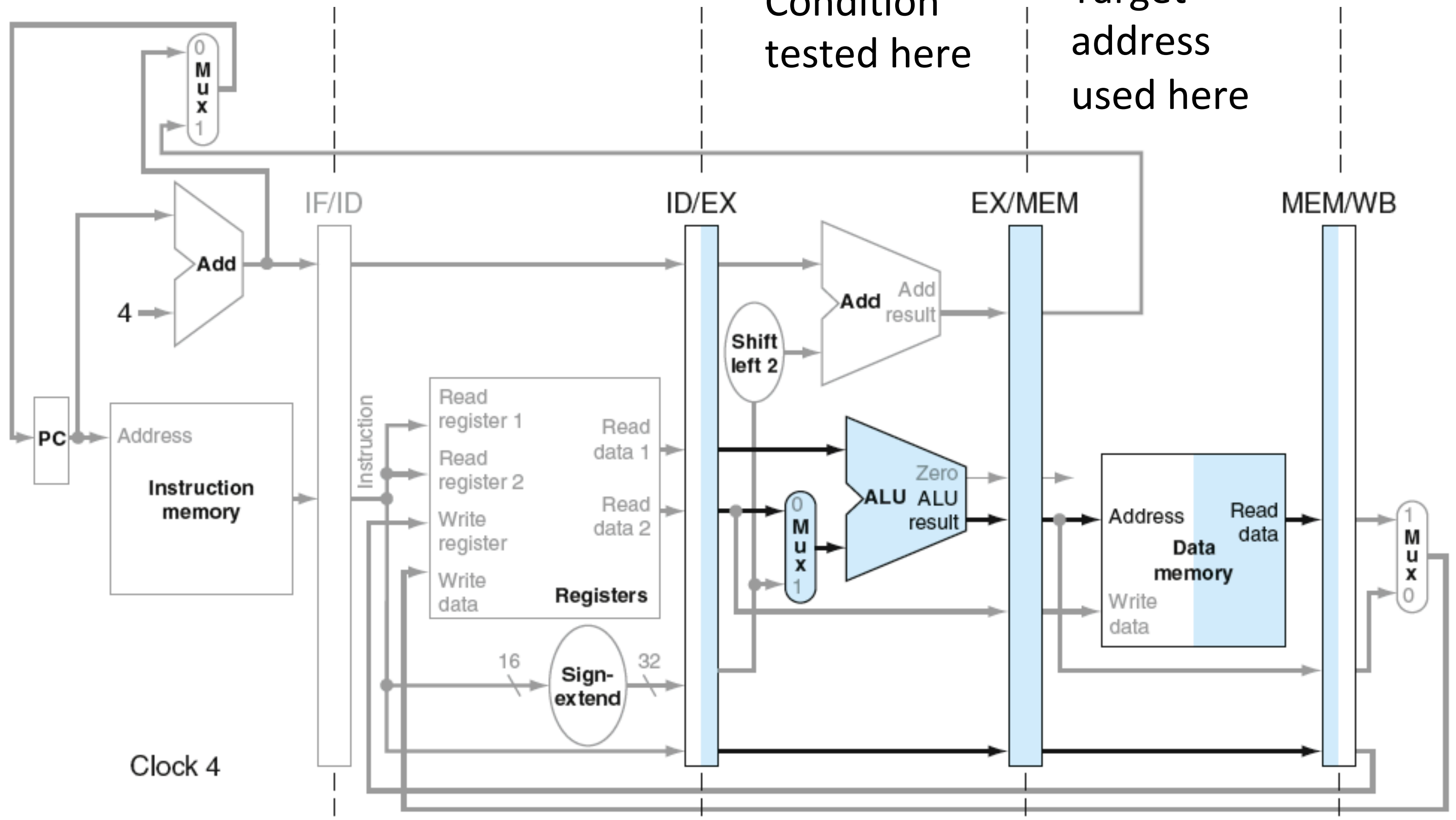
- Stages 1 through 3 must be flushed
- This creates 3 bubbles (a 3-cycle penalty)

The effect is called a control hazard

Control Hazards

Condition tested here

Target address used here



If condition is true, the PC is loaded with the target address when beq is in stage 4

Control Hazard Example

```
        sub  $11,$6,$5

        beq   $11,$0,skip

        add $4,$7,$3

        sw    $9,4($7)

        add   $9,$2,$9
                .
                .
                .
skip:   or    $8,$4,$0
```

If $11 = 0, the 3 instructions that follow beq should not complete.
The next instruction to execute should be or $8,$4,$0 instruction

Control Hazard Example

Cycle	IF	ID	EX	MEM	WB
1	sub $11,$6,$5				
2	beq $11,$0,skip	sub $11,$6,$5			
3	add $4,$7,$3	beq $11,$0,skip	sub $11,$6,$5		
4	sw $9,4($7)	add $4,$7,$3	beq $11,$0,skip	sub $11,$6,$5	
5	add $9,$2,$9	sw $9,4($7)	add $4,$7,$3	beq $11,$0,skip	sub $11,$6,$5
6	**or $8,$4,$0**	bubble	bubble	bubble	beq $11,$0,skip

Flushing adds 3 extra clock cycles in this case

This is known as the branch penalty

Control Hazard Options

1. Delay fetching instructions until branch behavior is known
   - Still causes 3-cycle penalty (outcome is known in stage 4)
2. Employ delayed branches
   - Fill delay slots with instructions needing to execute in any case
   - Compiler or programmer fills delay slots
   - Use NOPs if useful instructions cannot be identified
3. Evaluate the branch condition early
   - Requires computing target address in stage 2
   - Requires comparator in stage 2
4. Predict the behavior of the branch instruction
   - Requires recording previous branch behavior
   - Flushing is still required if prediction is wrong