The behavior of an unconditional branch is known
> The branch is decoded in stage 2
> The instruction following the branch is in stage 1
> This instruction could be flushed

Allowing the instruction to execute avoids a bubble
> The instruction is said to occupy the "branch delay slot"
> Compilers can move an instruction into the delay slot
> It is easier to fill a single slot than to fill 3

```
          li      $9,16                        li      $9,16
          li      $10,0xFFF                    li      $10,0xFFF
loop:     lw      $4,0($12)         loop:      lw      $4,0($12)
          and     $5,$7,$10                    and     $5,$7,$10
          add     $5,$4,$5                     add     $5,$4,$5
          addi    $9,$9,-1                     bne     $9,$0,loop
          bne     $9,$0,loop                   addi    $9,$9,-1
          nop
```

Branch delay slot instruction executes whether branch is taken or not
A cycle is saved by filling the slot with a useful instruction

```
        li      $9,16                           li      $9, 15  ←
        li      $10,0xFFF                       li      $10,0xFFF
loop:   lw      $4,0($12)               loop:   lw      $4,0($12)
        and     $5,$7,$10                       and     $5,$7,$10
        add     $5,$4,$5                        add     $5,$4,$5
        addi    $9,$9,-1                        bne     $9,$0,loop
        bne     $9,$0,loop                      addi    $9,$9,-1
        nop
```

The initial value in the loop control register has been set to 15 so that the number of loop iterations will be the same as for the original code.
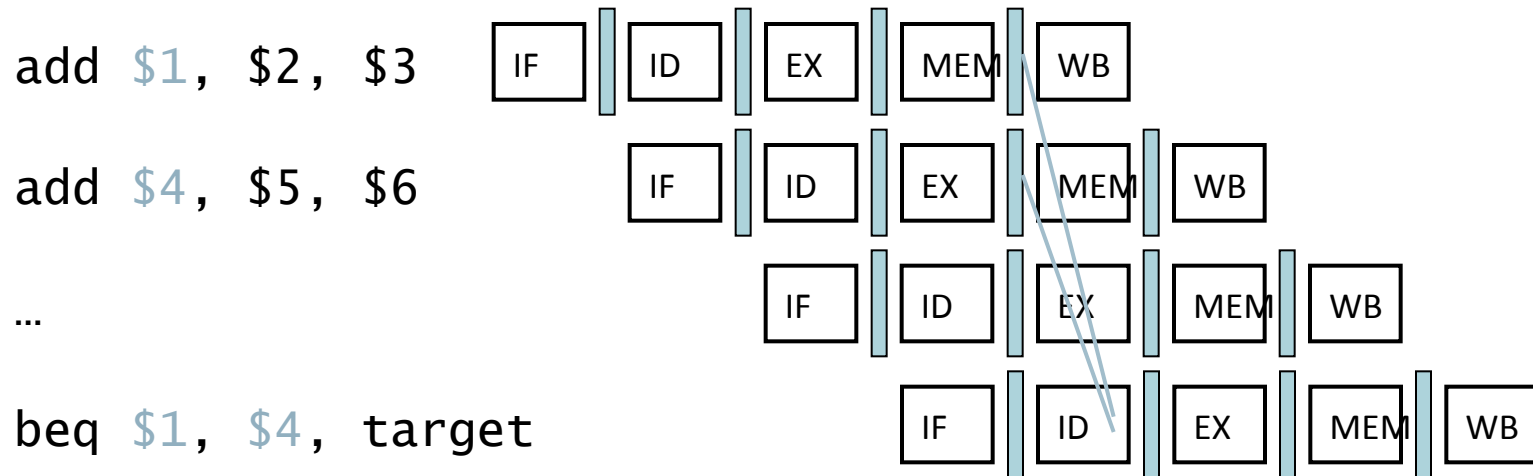
- Move hardware to determine outcome to ID stage
  - Target address adder
  - Register comparator
- Example: branch taken

```
36:  sub   $10, $4, $8
40:  beq   $1,  $3, 7
44:  and   $12, $2, $5
48:  or    $13, $2, $6
52:  add   $14, $4, $2
56:  slt   $15, $6, $7
     ...
72:  lw    $4, 50($7)
```
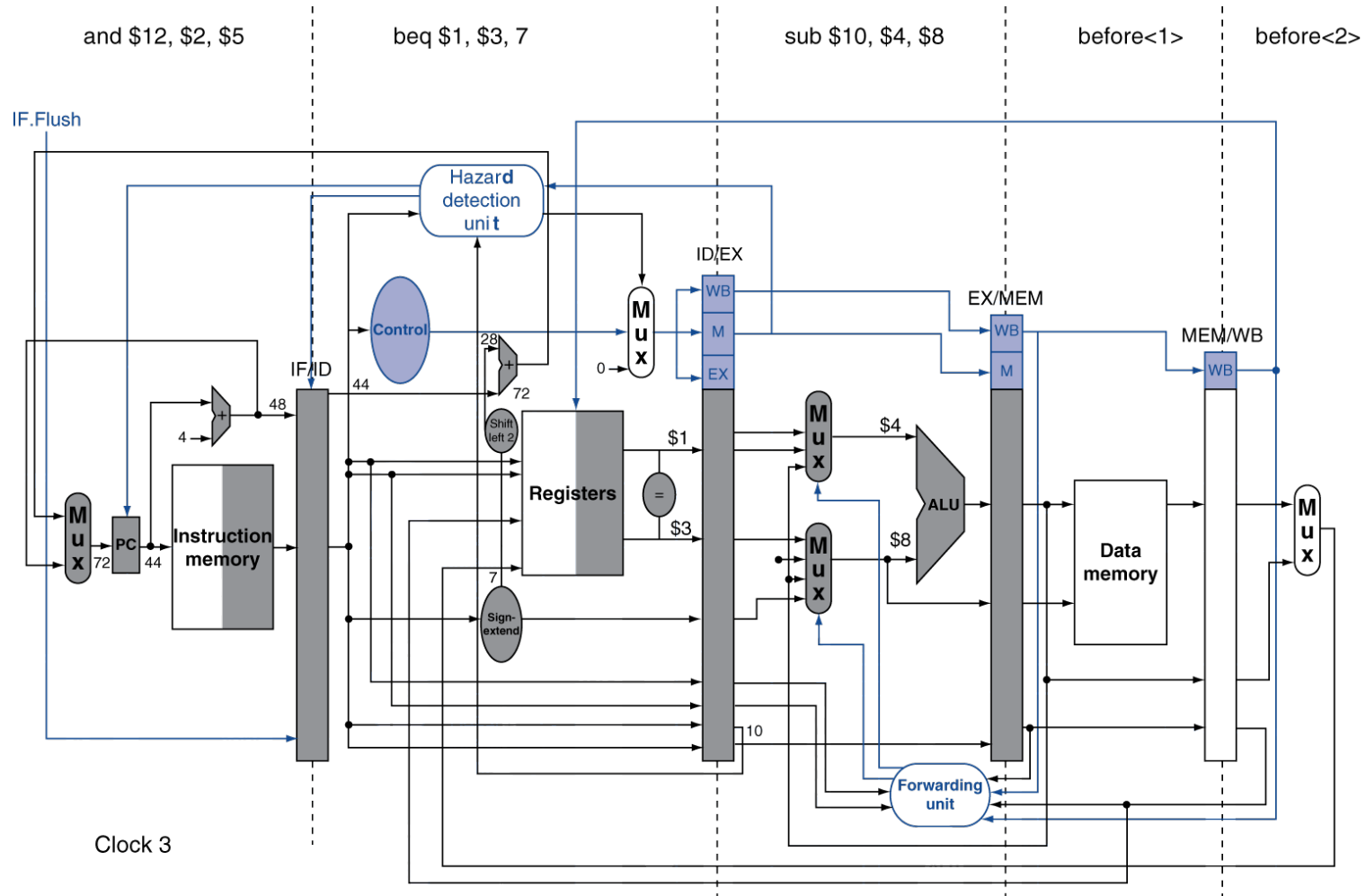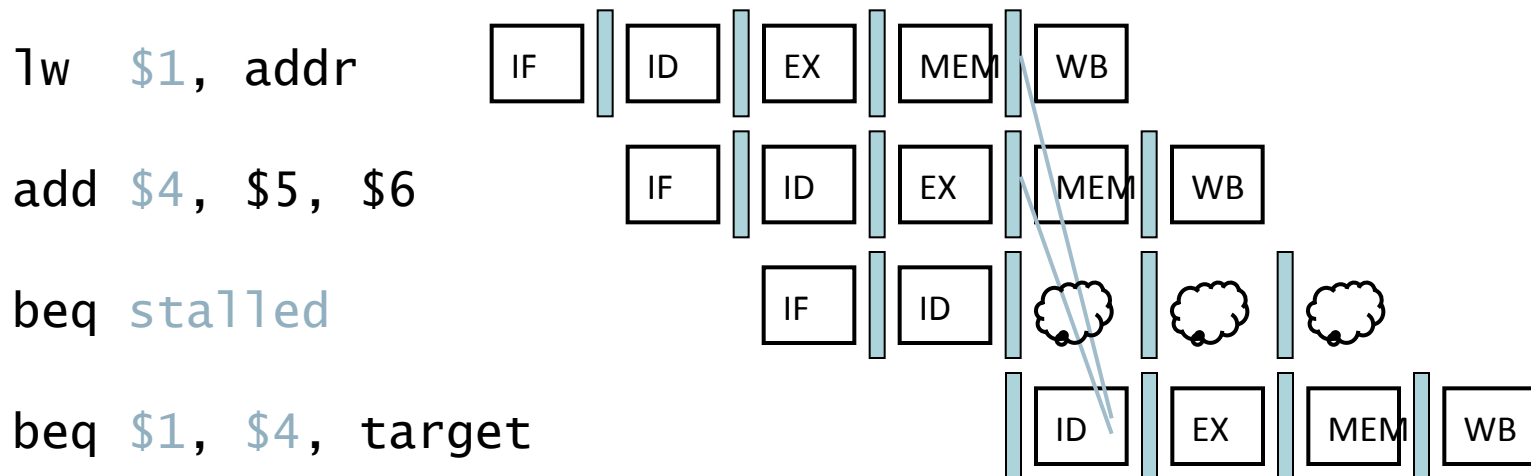
Delay slot is still needed since condition is tested in stage 2

- If a comparison register is a destination of 2nd or 3rd preceding ALU instruction

```
add  $1, $2, $3
```
| IF | ID | EX | MEM | WB |

```
add  $4, $5, $6
```
| IF | ID | EX | MEM | WB |

```
…
```
| IF | ID | EX | MEM | WB |

```
beq  $1, $4, target
```
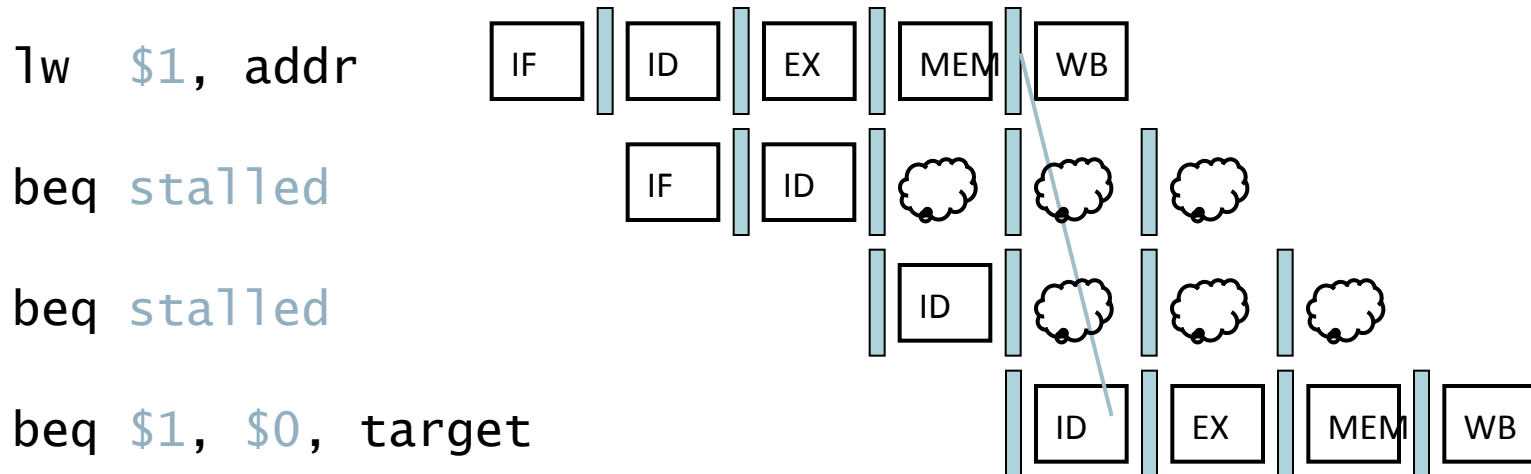| IF | ID | EX | MEM | WB |

- Can resolve using forwarding

- **If a comparison register is a destination of preceding ALU instruction or 2nd preceding load instruction**

  - Need 1 stall cycle

```
lw   $1, addr
```
IF | ID | EX | MEM | WB

```
add $4, $5, $6
```
IF | ID | EX | MEM | WB

```
beq stalled
```
IF | ID | ☁ | ☁ | ☁

```
beq $1, $4, target
```
ID | EX | MEM | WB

- **If a comparison register is a destination of immediately preceding load instruction**
  - Need 2 stall cycles

```
lw   $1, addr        [IF] [ID] [EX] [MEM] [WB]

beq stalled               [IF] [ID] ☁ ☁ ☁

beq stalled                    [ID] ☁ ☁ ☁

beq $1, $0, target                  [ID] [EX] [MEM] [WB]
```

Branches have more impact on deeper and superscalar pipelines
        More stages may have to be flushed
        Superscalar pipelines process multiple instructions per stage

Branch prediction will be examined as another technique