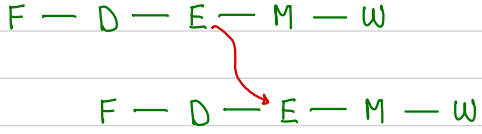
Method 2: Forwarding

Example A:

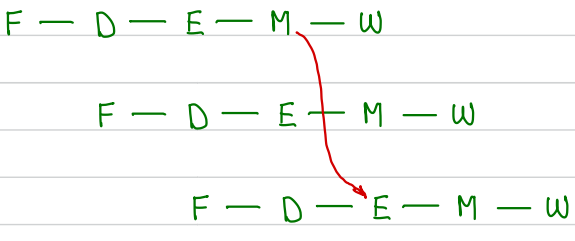
```
add   $t0, $t1, $t2
sub   $t3, $t4, $t0
```



- grab the result of the add instruction at the right time to use in sub's execute stage.

Example B:

```
lw    $t0, ($t1)
add   $t2, $t3, $t4
slt   $t6, $t0, $t5
```

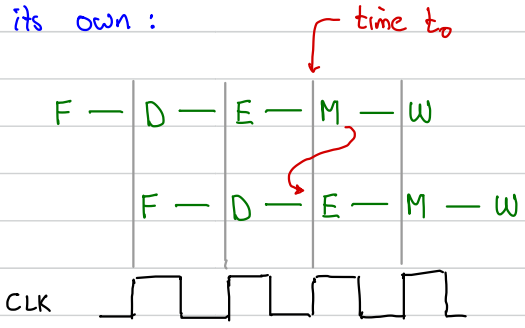


- grab the result from lw's memory stage to use in slt's execute stage.

Method 3: Forwarding and stalling

Forwarding sometimes doesn't work on its own:

```
lw    $t0, ($t1)
add   $t3, $t0, $t2
```



- can't send data back in time!
- memory contents ($t0) only available <u>after</u> time t_0.

→ how to fix: stall for one clock period, then forward.

Solutions to control hazards

Branching — decision to branch not made in time

Jump — also has problems

1. Stall — don't start the fetch of the next instruction right away

2. Predict — processor guesses what the next address will be; acts on it right away
   — processor checks that the guess was right; cancels instructions if wrong.

Dynamic branch prediction

- As the program is running, processor keeps track of recent branch instructions; updates its predictions
- very complex!

3. Delayed branch and jump rules

Older and less sophisticated than branch prediction

- used by real MIPS processors
- Idea: As the processor determines the destination address of a branch or jump, give it some useful work to do.

For example:

```
        slt   $t0, $t1, $t2
        beq   $t0, $zero, L1
        add   $t4, $t5, $t6     <-- always execute one next instruction
        lw    $t7, ($t8)
L1:     or    [...]
        sub   [...]
```

Instruction flow if $t0 != 0

slt → beq → add → lw → or → sub ...

delayed branch slot

If $t0 == 0

slt → beq → add → or → sub ...

branch taken

branch delay slot

Example C-code:

```
do {
    if (*p < 0)
        count++;
    p++;
} while (p != past_end);
```

Translation with delay slots:

```
L1: lw   $t0 ($a0)
    slt  $t1, $zero, $t0
    beq  $t1, $zero, L2
    addiu $a0, $a0, 4      # p++
    add  $t9, $t9, 1       # count++
L2: bne  $a0, $a1, L1      # if (p != past_end)
                           #     goto L1
```

branch delay slot; instruction executed regardless of branch (do something useful!)

no operation (if nothing useful to be done) → nop

Conditional instructions

```
movn  $t0, $t1, $t2    if ($t2 != 0)
                           $t0 = $t1

movz  $t0, $t1, $t2    if ($t2 == 0)
                           $t0 = $t1
```

Making pipelining work in hardware

Textbook presents a series of designs

- simple and incomplete at first
- then progressively more complex and correct.

Figure 7.47

- This handles R-type, lw, sw correctly (assuming no hazards)
- makes an attempt to handle beq, but doesn't get it right (executes 3 delay-slot instructions)

Pipeline registers

- pipelining made possible by using more DFFs to make pipeline registers.