CSEE W3827

Fundamentals of Computer Systems

Homework Assignment 6

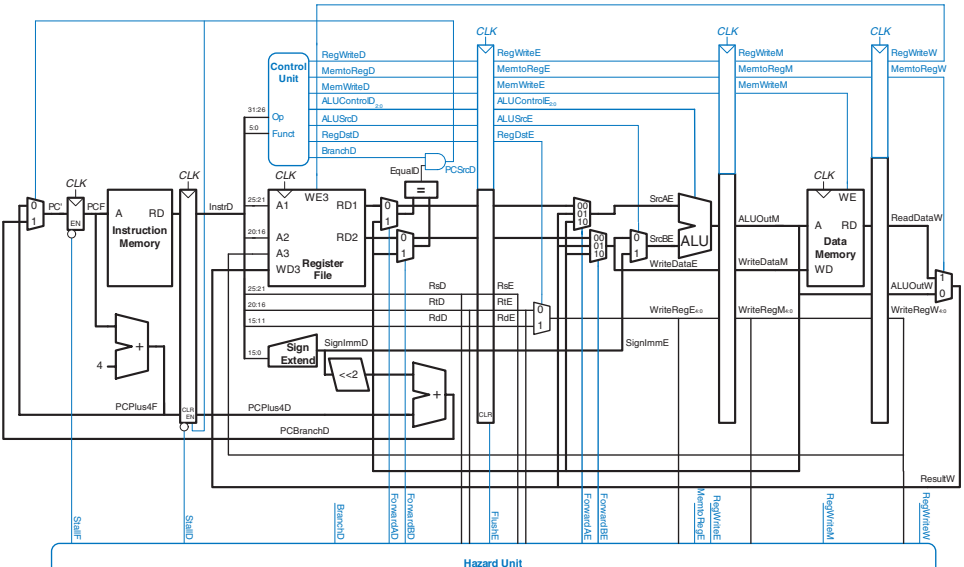
Solutions

Prof. Stephen A. Edwards

Columbia University

Due December 6th, 2011 at 10:35 AM

Show your work for each problem; we are more interested in how you get your answer than whether you get the right answer.

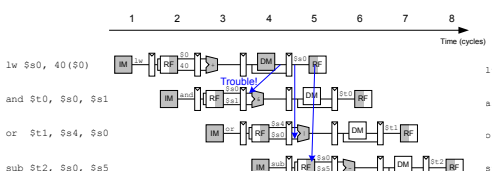


The fully bypassed five-stage (F, D, E, M, W) MIPS pipeline with stall logic

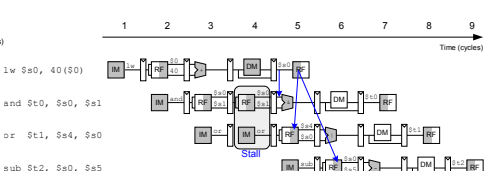
1. (30 pts.) For the five-stage MIPS pipeline with full bypass and stall logic discussed in class, in the book, and included in this assignment, explain the hazard, if any, in each sequence of code below and explain how the processor will resolve it, e.g., “stall two cycles,” “bypass W to E,” “bypass M to D.” The stages are abbreviated F, D, E, M, and W (Fetch, Decode, Execute, Memory, and Writeback). For one example from the slides,

```
lw  $s0, 40($0)
and $t0, $s0, $s1
or  $t1, $s4, $s0
sub $t2, $s0, $s5
```

The *and* must stall a cycle then use a W-to-E bypass to get $s0.
The *or* is already in the pipeline when the *and* stalls, so it, too, must stall.



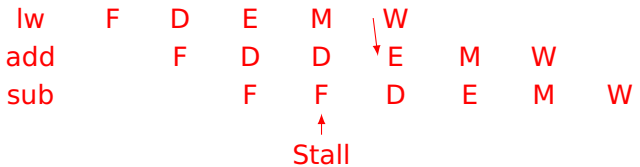
Hazard Illustrated



Resolution of Hazard

```
    lw  $t1, 42($t1)
(a) add $t2, $t1, $t3
    sub $t4, $t1, $t3
```

The *add* must stall a cycle before using a W-to-E bypass to get $t1. The *sub* must also stall.




```
    lw  $t1, 42($t1)
(b) add $t4, $t2, $t3
    sub $t5, $t2, $t3
```

No hazard; $t1 and $t4 not used here

```
    add $t1, $t2, $t3
(c) add $t1, $t1, $t4
    add $t1, $t1, $t5
```

The second and third *add*s each use the M-to-E bypass to get $t1; no stalling is necessary.


```
    lw  $t1, 42($t1)
(d) add $t3, $t2, $t3
    sub $t4, $t1, $t3
```

The *sub* needs to use a W-to-E bypass for $t1 from the *lw* and a M-to-E for $t3 from the *add*. No stalling needed.

2. (30 pts.) Consider the following fragment of MIPS code:

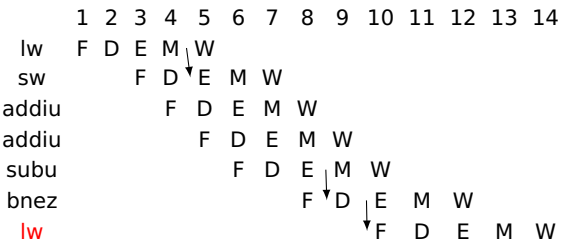
```
        li    $t0, 64
loop:   lw    $t1, 0($t2)
        sw    $t1, 0($t3)
        addiu $t2, $t2, 4
        addiu $t3, $t3, 4
        subu  $t0, $t0, 1
        bnez  $t0, loop
```

(a) When this code fragment is run, how many instructions will be executed total?

$1 + 64 \cdot 6 = 385$

(b) How many cycles will it take to execute on the fully bypassed MIPS processor?

```
   li    $t0, 64
l: lw    $t1, 0($t2)
   sw    $t1, 0($t3)
   addiu $t2, $t2, 4
   addiu $t3, $t3, 4
   subu  $t0, $t0, 1
   bnez  $t0, l
```



The *sw* will have to stall a cycle because the data from the *lw* is only available at the beginning of the 5th cycle, but there is no bypass from the W stage to the M stage, but there is one from W to E. The result of the *subu* is needed by the *bnez* instruction, and although there's an M-to-D bypass, a single cycle stall is needed. Furthermore, the instruction after the *bnez* is always fetched but flushed for all but the last iteration, causing another cycle of delay. Thus,
$1 + 63 \cdot (6 + 3) + 1 \cdot (6 + 2) = 576$.

Now, consider this fragment of code:

```
        li    $t0, 64
loop:   subu  $t0, $t0, 1
        lw    $t1, 0($t2)
        addiu $t2, $t2, 4
        sw    $t1, ($t3)
        addiu $t3, $t3, 4
        bnez  $t0, loop
```

(c) When this code fragment is run, how many instructions will be executed total?

$1 + 64 \cdot 6 = 385$

(d) How many cycles will it take to execute on the fully bypassed MIPS processor?

Instruction have been arranged to avoid most stalls; the one exception is the *bnez*, which still takes two cycles. $1 + 64 \cdot 7 = 449$.

3. (25 pts.) Consider a computer with a *direct mapped* cache of 64 16-byte blocks backed by 2^{24} bytes of main memory.

(a) How many blocks does main memory contain?

$2^{24} \div 16 = 2^{20}$

(b) How are memory addresses interpreted, i.e., how many bits each are the tag, set, and byte offset fields?

24 bits total: 4 for byte offset; 6 for set number; remaining 14 for tag

(c) To which cache set will the address `0xDECADE` map?

`0xDECADE` = $1101\,1110\,1100\,1010\,1101\,1110_2$
Set number = bits 4–9 = $101101 = 45_{10}$

(d) Assuming the cache starts empty, what sequence of events would be produced by reading bytes in the following sequences of addresses? Classify each event as a compulsory miss, a conflict miss, a spatial locality hit, or a temporal locality hit.

Address	Event
0xDECADE	Compulsory Miss
0xDECAD8	Spatial locality hit (same block as above)
0xDECAE8	Compulsory Miss (different block)
0xBECADE	Compulsory Miss
0xDECADE	Conflict Miss (same block; different tag)

4. (15 pts.) Consider a computer with a *fully associative* cache of 32 64-byte blocks backed by 2^{16} bytes of main memory.

   (a) How many blocks does main memory contain?

$2^{16} \div 64 = 2^{10} = 1024$

   (b) How are memory addresses interpreted, i.e., how many bits each are the tag, set, and byte offset fields?

16 bits total: 6 byte offset; 0 set number; remaining 10 tag

   (c) To which cache set will the address `0xF00D` map?

Fully associative caches have exactly one set: everything maps to it