# Assignment 4

Computer Architecture CS3350B

Albirawi, Zaid
250626065

April 6th, 2015

1. Consider the following MIPS instructions to be executed on a pipelined processor:

```
and     $s0, $s1, $s2
xor     $s2, $s0, $s1
```

where \$s1 and \$s2 are assumed to hold values before executing these two instructions. The processor uses a 5-stage pipeline, as defined in class. The successive five stages of this pipeline are denoted by IF, ID, EXE, MEM, WB.

1.1. Indicate dependences and their type (read after write or write after read) among the above two MIPS instructions.

\$s0 of xor depends on \$s0 of add. RAW.
\$s2 of xor depends on \$s2 of add. WAR.

1.2. Assume that there is no forwarding mechanisms in this pipelined processor. Then, indicate hazards and the appropriate add nop (or stall) instructions to eliminate those hazards in the following pipeline execution diagram:

Data Hazard: xor must be delayed 3 cycles as the ID stage of the xor instruction must wait for the WB stage of the and instruction to finish. However, if there is a hardware solution to allow reads and writes to the same register in the same cycle then the xor instructions only needs a delay of 2 cycles.

The WAR dependency will not cause any hazards as the write to #s2 will not happen until the the fifth clock cycle. However, the \$s2 read will execute on the third clock cycle.

| | Clock Cycles | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Instruction | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| and | IF | ID | EX | MEM | WB | | | | | |
| nop | | - | - | - | - | - | | | | |
| nop | | | - | - | - | - | | | | |
| xor | | | | IF | ID | EX | MEM | WB | | |

1.3. Assume that there is an ALU-ALU forwarding but no other forwarding mechanisms, like a forwarding from the MEM to the EX stage. Indicate hazards and add the appropriate nop instructions to eliminate those hazards, if any.

No hazards present.

| | Clock Cycles | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Instruction | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| and | IF | ID | EX | MEM | WB | | | | | |
| xor | | IF | ID | EX | MEM | WB | | | | |

2. This exercise is intended to help you understand the relationship between delay slots, control hazards and branch execution in a pipelined processor. We assume that the following MIPS code

```
loop:   lw      $t2, 0($s2)
        addi    $s2, $s2, 4
        bne     $t2, $0, loop
        sll     $t2, $t0, 2      # assuming $t0 holds some value
        sw      $t2, 0($s2)
```

is executed on a pipelined processor with a 5-stage pipeline and full forwarding (that is all the forwarding mechanisms defined in class). We assume that $s2 initially stores the base address of a 32-bit integer array A in C code. Assume that A[] = {5, 73, 0} and that there is no special branch comparator inserted in the processor.

2.1. Draw the pipeline execution diagram of the above MIPS code applied to the above array A, using a table similar in format to Table 1 above.

| | Clock Cycles | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Instruction | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| lw | IF | ID | EX | MEM | WB | | | | | |
| addi | | IF | ID | EX | MEM | WB | | | | |
| bne | | | IF | ID | EX | MEM | | | | |
| nop | | | | - | - | - | - | - | | |
| nop | | | | | - | - | - | - | - | |
| lw | | | | | | IF | ID | EX | MEM | WB |
| | Clock Cycles | | | | | | | | | |
| Instruction | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| addi | IF | ID | EX | MEM | WB | | | | | |
| bne | | IF | ID | EX | MEM | WB | | | | |
| nop | | | - | - | - | - | - | | | |
| nop | | | | - | - | - | - | - | | |
| lw | | | | | IF | ID | EX | MEM | WB | |
| addi | | | | | | IF | ID | EX | MEM | WB |
| | Clock Cycles | | | | | | | | | |
| Instruction | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| bne | IF | ID | EX | MEM | WB | | | | | |
| nop | | - | - | - | - | - | | | | |
| nop | | | - | - | - | - | - | | | |
| sll | | | | IF | ID | EX | MEM | WB | | |
| sw | | | | | IF | ID | EX | MEM | WB | |

2.2. Assume that the processor uses none of the following techniques with branch instruction:

- delayed branch (see Slide 25 of the PDF version of Lecture 6.2)
- branch prediction (see Slide 27 of the PDF version of Lecture 6.2, as well as http://en.wikipedia.org/wiki/Branch_predictor

Also, assume that branches execute in the EX stage. Draw the pipeline execution diagram until the second iteration finishes. For each forwarding, indicate its type.

| | Clock Cycles | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Instruction | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | Forwarding |
| lw | IF | ID | EX | MEM | WB | | | | | $t0 ALU-ALU to bne |
| addi | | IF | ID | EX | MEM | WB | | | | none |
| bne | | | IF | ID | EX | MEM | WB | | | none |
| nop | | | | - | - | - | - | - | | none |
| nop | | | | | - | - | - | - | - | none |
| | Clock Cycles | | | | | | | | | |
| Instruction | | | | | | | | | | Forwarding |
| lw | IF | ID | EX | MEM | WB | | | | | $t0 ALU-ALU to bne |
| addi | | IF | ID | EX | MEM | WB | | | | none |
| bne | | | IF | ID | EX | MEM | WB | | | none |
| nop | | | | - | - | - | - | - | | none |
| nop | | | | | - | - | - | - | - | none |

2.3. Assume that delay slots are used and that a predict-taken branch predictor (see http: //www.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/branchPred.html) uses the following policy on bne: not taken. In the given code, the predicted instruction is now the delay slot instruction for the branch. Draw the pipeline execution diagram until the above code ends.

| | Clock Cycles | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Instruction | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| lw | IF | ID | EX | MEM | WB | | | | | |
| addi | | IF | ID | EX | MEM | WB | | | | |
| bne | | | IF | ID | EX | MEM | WB | | | |
| lw | | | | IF | ID | EX | MEM | WB | | |
| nop | | | | | - | - | - | - | - | |
| addi | | | | | | IF | ID | EX | MEM | WB |
| | Clock Cycles | | | | | | | | | |
| Instruction | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| bne | IF | ID | EX | MEM | WB | | | | | |
| lw | | IF | ID | EX | MEM | WB | | | | |
| nop | | | - | - | - | - | - | | | |
| addi | | | | IF | ID | EX | MEM | WB | | |
| bne | | | | | IF | ID | EX | MEM | WB | |
| lw | | | | | | IF | ID | EX | MEM | WB |
| | Clock Cycles | | | | | | | | | |
| Instruction | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| nop | - | - | - | - | - | | | | | |
| sll | | IF | ID | EX | MEM | WB | | | | |
| sw | | | IF | ID | EX | MEM | WB | | | |

3. Consider the following C code:

```
for (i = 0; i < n; ++i)
        a[i] = b[i] + i;
```

where a and b are 32-bit integer arrays of size n. We give a corresponding MIPS instruction sequence:

```
        add $t0, $0, $0         # $t0 = 0, which corresponds to i in C code
loop:   lw $s1, 0($s4)          # assume $s4 stores the base address of array b
        add $s0, $s1, $t0       # $s0 gets b[i] + i
        sw $s0, 0($s2)          # assume $s2 stores the base address of array a
        addi $t0, $t0, 1        # ++i
        addi $s2, $s2, 4        # get address of a[i+1]
        addi $s4, $s4, 4        # get address of b[i+1]
        slt $t2, $t0, $s5       # assume that $s5 holds n
        bne $t2, $0, loop       # if $t2 == 1, go to loop
```

Assume that the above MIPS instructions will be executed on a 5-stage pipelined processor (as defined in class). Also, one can ignore control hazards (but not data hazards, of course) and assume that full-forwarding (as defined in class) is implemented.

3.1. Draw the pipeline execution diagram without unrolling (one iteration of the loop would be enough) and compute the average CPI (clock cycle per instruction) of the loop. You may consider the two cases: using or not using instruction reordering.

Without instruction reordering.

| | Clock Cycles | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Instruction | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| add | IF | ID | EX | MEM | WB | | | | | |
| lw | | IF | ID | EX | MEM | WB | | | | |
| nop | | | - | - | - | - | - | | | |
| add | | | | IF | ID | EX | MEM | WB | | |
| sw | | | | | IF | ID | EX | MEM | WB | |
| addi | | | | | | IF | ID | EX | MEM | WB |
| | Clock Cycles | | | | | | | | | |
| Instruction | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| addi | IF | ID | EX | MEM | WB | | | | | |
| addi | | IF | ID | EX | MEM | WB | | | | |
| slt | | | IF | ID | EX | MEM | WB | | | |
| bne | | | | IF | ID | EX | MEM | WB | | |

$Average_{CPI} = CC \div Number\ of\ instructions\ = 9 \div 8 = 1.125$

Therefore, the average CPI of the loop, without the add $t0, $0, $0 instruction , and without instruction reordering is 1.125.

With instruction reordering.

```
        add $t0, $0, $0       # $t0 = 0, which corresponds to i in C code
loop:   lw $s1, 0($s4)        # assume $s4 stores the base address of array b
        addi $s2, $s2, 4      # get address of a[i+1]
        add $s0, $s1, $t0     # $s0 gets b[i] + i
        sw $s0, 4($s2)        # assume $s2 stores the base address of array a
        addi $t0, $t0, 1      # ++i
        addi $s4, $s4, 4      # get address of b[i+1]
        slt $t2, $t0, $s5     # assume that $s5 holds n
        bne $t2, $0, loop     # if $t2 == 1, go to loop
```

| | Clock Cycles | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Instruction | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| add | IF | ID | EX | MEM | WB | | | | | |
| lw | | IF | ID | EX | MEM | WB | | | | |
| addi | | | IF | ID | EX | MEM | WB | | | |
| add | | | | IF | ID | EX | MEM | WB | | |
| sw | | | | | IF | ID | EX | MEM | WB | |
| addi | | | | | | IF | ID | EX | MEM | WB |
| | Clock Cycles | | | | | | | | | |
| Instruction | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| addi | IF | ID | EX | MEM | WB | | | | | |
| slt | | IF | ID | EX | MEM | WB | | | | |
| bne | | | IF | ID | EX | MEM | WB | | | |

$$Average_{CPI} = CC \div Number\ of\ instructions\ = 8 \div 8 = 1$$

Therefore, the average CPI of the loop, without the add $t0, $0, $0 instruction , and with instruction reordering is 1.

3.2. Apply loop unrolling (as well as instruction re-ordering, if you like) on the above MIPS code for two iterations. Write the corresponding MIPS instruction code. You may consider the two cases: using or not using 2-issue MIPS instructions, like on Slides 12 and 13 of the PDF version of the set of slides 6.2.

New code with w-iteration loop unrolling  and instruction reordering.

```
        add $t0, $0, $0        # $t0 = 0
loop:   lw $s1, 0($s4)         # load b[i]
        lw $s3, 4($s4)         # load b[i + 1]

        add $t1, $s1, $t0      # $t1 = b[i] + i
        addi $t0, $t0, 1       # ++i

        add $t3, $s3, $t0      # $t3 = b[i + 1] + i + 1
        addi $t0, $t0, 1       # ++i

        sw $t1, 0($s2)         # a[i] = $t1
        sw $t3, 4($s2)         # a[i + 1] = $t3

        addi $s2, $s2, 8       # $s2 = a[i + 2]
        addi $s4, $s4, 8       # $s4 = b[i + 2]

        slt $t2, $t0, $s5      # assume that $s5 holds n
        bne $t2, $0, loop      # if $t2 == 1, go to loop
```

3.3. Draw the pipeline execution diagram of your MIPS instructions (one iteration of the new loop would be enough) and compute the average CPI of the loop.

Using single-issue MIPS instruction.

| | Clock Cycles | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Instruction | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| add | IF | ID | EX | MEM | WB | | | | | |
| lw | | IF | ID | EX | MEM | WB | | | | |
| lw | | | IF | ID | EX | MEM | WB | | | |
| add | | | | IF | ID | EX | MEM | WB | | |
| addi | | | | | IF | ID | EX | MEM | WB | |
| add | | | | | | IF | ID | EX | MEM | WB |
| | Clock Cycles | | | | | | | | | |
| Instruction | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| addi | IF | ID | EX | MEM | WB | | | | | |
| sw | | IF | ID | EX | MEM | WB | | | | |
| sw | | | IF | ID | EX | MEM | WB | | | |
| addi | | | | IF | ID | EX | MEM | WB | | |
| addi | | | | | IF | ID | EX | MEM | WB | |
| slt | | | | | | IF | ID | EX | MEM | WB |
| | Clock Cycles | | | | | | | | | |
| Instruction | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| bne | IF | ID | EX | MEM | WB | | | | | |

$$Average_{CPI} = CC \div Number\ of\ instructions\ = 12 \div 12 = 1$$

Therefore, the loop's average CPI using single-issue MIPS instructions is 1.

Using 2-issue MIPS instruction.

| | ALU or Branch | Data Transfer | CC |
|---|---|---|---|

| | | | |
|---|---|---|---|
| | add $t0, $0, $0 | nop | 1 |
| loop: | addi $s2, $s2, 8 | lw $s1, 0($s4) | 2 |
| | add $t1, $s1, $t0 | lw $s3, 4($s4) | 3 |
| | addi $t0, $t0, 1 | sw $t1, -8($s2) | 4 |
| | add $t3, $s3, $t0 | nop | 5 |
| | addi $t0, $t0, 1 | sw $t3, -4($s2) | 6 |
| | addi $s2, $s2, 8 | nop | 7 |
| | slt $t2, $t0, $s5 | nop | 8 |
| | bne $t2, $0, loop | nop | 9 |

| | | Clock Cycles | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Instruction | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| add | nop | IF | ID | EX | MEM | WB | | | | | |
| addi | lw | | IF | ID | EX | MEM | WB | | | | |
| add | lw | | | IF | ID | EX | MEM | WB | | | |
| addi | sw | | | | IF | ID | EX | MEM | WB | | |
| add | nop | | | | | IF | ID | EX | MEM | WB | |
| addi | sw | | | | | | IF | ID | EX | MEM | WB |
| | | Clock Cycles | | | | | | | | | |
| Instruction | | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| addi | nop | IF | ID | EX | MEM | WB | | | | | |
| slt | nop | | IF | ID | EX | MEM | WB | | | | |
| bne | nop | | | IF | ID | EX | MEM | WB | | | |

$$Average_{CPI} = CC \div Number\ of\ instructions\ = 8 \div 12 = 2 \div 3 = 0.66$$

Therefore, the loop’s average CPI using 2-issue MIPS instructions is 0.66.

4. A 4-processor shared-memory multiprocessor configuration implements write-back cache using the MESI (**M**odified, **E**xclusive, **S**hared, **I**nvalid) algorithm for cache coherency. Assume that location 0x0010 is not in any cache at the start of the following sequence.

Consider the following read/write operations:

a) Processor 0 reads from location 0x0010
b) Processor 0 writes to location 0x0010
c) Processor 2 reads from location 0x0010
d) Processor 3 reads from location 0x0010
e) Processor 2 writes to location 0x0010
f) Processor 1 reads from location 0x0010
g) Processor 3 writes to location 0x0010

Show the state (M, E, S or I) for the cache line containing location 0x0010 in each processor cache after each operation. Also note any transfers to/from memory if any occurs.
Solutions to operations (a) and (b) are given in the following table. Complete the table for operations (c) - (g).

| | State | | | | |
|---|---|---|---|---|---|
| Action | P0 | P1 | P2 | P3 | Memory transfers |
| P0 read miss | E | I | I | I | P0 reads a cache line from memory |
| P0 write hit | M | I | I | I | - |
| P2 read miss | S | I | S | I | P0 sends value over the bus updating the main memory and sharing the value with P2. |
| P3 read miss | S | I | S | S | P0 or P2 send value over the bus sharing the value with P3. |
| P2 write hit | I | I | M | I | - |
| P1 read miss | I | S | S | I | P3 sends value over the bus updating the main memory and sharing the value with P2. |
| P3 write miss | I | I | I | M | P3 reads a cache line from memory or bus |

5. Consider a shared-memory multiprocessor that consists of three processor/cache units where cache coherence is maintained by a MESI protocol. The private caches are direct mapped. Assume that words X1, X2 and X3 are in the same cache line. Given the

following sequence of events, identify each miss as a cold miss (CM), a true sharing miss (TM), a false sharing miss (FM), or a hit (H). Explain briefly the reasons.

| Clock | Processor 1 | Processor 2 | Processor 3 | CM, TM, FM or H |
|---|---|---|---|---|
| 1 | Read X1 | | | CM, the cache line was not read before. |
| 2 | | Read X2 | | CM, the cache line was not read before. |
| 3 | | | Read X3 | CM, the cache line was not read before. |
| 4 | Write X1 | | | H, cache line was read. |
| 5 | | | Write X3 | FM, X1 was written in processor 1. |
| 6 | | Read X1 | | TM, X1 was written in Processor 1. |
| 7 | Write X2 | | | FM, X3 was written in Processor 3. |
| 8 | | | Read X1 | FM, X2 was written in Processor 1. |
| 9 | | | Read X2 | H, no writes occurred after line was read. |

6. Consider a pipelined process (like the laundry example given in class) with s stages. Assuming n tasks are being processed by this pipeline. In each of the following scenario, compute

   1. the speedup w.r.t a serial execution,

2. the percentage of time during which the pipeline runs at full occupancy

6.1. Each stage runs within the same amount of time (as we did in Quiz 3)

Since n + s - 1 is equal to the amount of clock cycles needed to finish the pipeline process, and since n * s is the amount of clock cycles needed to finish the same process without pipelining. Then,

$$speed\_up = (n + s - 1) \div (n * s)$$

If the amount of clock cycles that the process is not at full occupancy is the 2 * (s - 1. Then,

$$\neg full\_occupancy = 2 * (s - 1) \div (n + s - 1)$$

Therefore,

$$\begin{aligned} full\_occupancy &= 1 - [2 * (s - 1) \div (n + s - 1)] \\ &= [n + s - 1 - 2s + 2] \div (n + s - 1) \\ &= (n - s + 1) \div (n + s - 1) \end{aligned}$$

6.2. Each stage, but the first one, runs within t units of time (say pico-seconds) meanwhile the first stage runs within r t units of time where r is a constant greater than one, thus, the first stage is slower than the other ones.

Therefore, Since (n * r * t) + t * (s - 1) is equal to the amount of time needed to finish the pipeline process, and since n * (r * t) + n * [t * (s - 1)] is the amount of clock cycles needed to finish the same process without pipelining. Then,

$$\begin{aligned} speed\_up &= [(n * r * t) + t * (s - 1)] \div \{n * (r * t) + n * [t * (s - 1)]\} \\ &= t * (n * r + s - 1) \div t * n * (r + s - 1) \\ &= (n * r + s - 1) \div n * (r + s - 1) \end{aligned}$$

Furthermore, if the amount of time that the process is not at full occupancy is the t * r + t * (s - 2) + t * (s - 1). Then,

$$\neg full\_occupancy = t * r + t * (s - 2) + t * (s - 1) \div (n * r * t) + t * (s - 1)$$

Therefore,

$$\begin{aligned} full\_occupancy &= 1 - \{t * [r + (s - 2) + (s - 1)] \div t * [(n * r) + (s - 1)]\} \\ &= 1 - (r + s - 2 + s - 1) \div (n * r + s - 1) \\ &= 1 - (r + 2s - 3) \div (n * r + s - 1) \\ &= [(n * r + s - 1) - (r + 2s - 3)] \div (n * r + s - 1) \\ &= (n * r + s - 1 - r - 2s + 3) \div (n * r + s - 1) \\ &= (n * r - r - s + 2) \div (n * r + s - 1) \\ &= [r * (n - 1) - s + 2] \div (n * r + s - 1) \end{aligned}$$

6.3. Each stage, but the last one, runs within t units of time (say pico-seconds) meanwhile the last stage runs within r t units of time where r is a constant greater than one, thus, the last stage is slower than the other ones.

Same answers as question 6.2

$$\begin{aligned} speed\_up &= (n * r + s - 1) \div n * (r + s - 1) \\ full\_occupancy &= [r * (n - 1) - s + 2] \div (n * r + s - 1) \end{aligned}$$