CPSC 313, 06w Term 2— Midterm 2 — Solutions

Date: March 23, 2007; Instructor: Norm Hutchinson

1. **(12 marks)** Short answers.

1a. **(2 marks)** Describe the difference between stalling and creating a bubble.

Stalling means that an instruction remains in a particular pipeline stage for another cycle without making progress. Creating a bubble means injecting a dynamically created NOOP instruction into a particular pipeline stage.

1b. **(2 marks)** Is it possible to stall the pipeline without creating a bubble? Explain briefly.

No. When one instruction stalls in stage i, something must move into stage $i+1$, and that something is always a bubble (unless stage $i+1$ also stalls in which case the bubble happens in stage $i+2$).

1c. **(2 marks)** What is the difference between a dependency and a hazard?

A dependency is a relationship between instructions that forces them to execute in a particular order. A hazard is a situation that arises in an implementation where the dependency will not be honoured by the implementation without some additional effort.

1d. **(2 marks)** Give one example of an X86 (IA32) instruction that we have talked about in class that is impossible to execute on the pipelined processor described in class (PIPE-). Explain why it cannot be executed by PIPE-.

```
addl      %eax, (%ebx)
```

It cannot be executed because it references memory twice.

1e. **(2 marks)** What is branch prediction? Why is it useful?

Predicting early in the pipeline whether a branch will be taken. It is useful because predicting correctly allows the pipeline to remain full of useful instructions.

1f. **(2 marks)** What is the single most significant advantage of a pipelined processor implementation over a sequential implementation?

A pipelined implementation is able to keep more of the combinational logic of the processor busy and so is able to achieve higher throughput.

2. **(6 marks)** RISC vs. CISC

Each of the following statements is significantly more true of either a RISC or CISC instruction set architecture. Write "RISC" or "CISC" in the blank before each statement to indicate which.

1. CISC _____ has a rich collection of addressing modes
2. RISC _____ has many registers
3. CISC _____ has instructions suitable for many different purposes
4. CISC _____ has variable-length instructions
5. RISC _____ maps naturally to a pipelined implementation

6. RISC _____ has instructions with a regular structure

3. **(6 marks)** HCL

3a. **(2 marks)** Is it legal in HCL to define a signal in terms of itself? In other words, does HCL support recursion? Explain briefly.

No. HCL is translated into hardware and general recursion requires either loops or an unbounded amount of hardware to implement, neither of which are legal in combinational circuits.

3b. **(2 marks)** The simulator requires a signal named `need_valC` which is true whenever the Fetch stage should decode a 4 byte immediate value from the current instruction. With reference to the instruction encodings on page **??** and the stage outputs on page **??**, give an HCL implementation of `need_valC`.

```
int need_valC = icode in {IIRMOVL, IRMMOVL, IMRMOVL, IJXX, ICALL};
```

3c. **(2 marks)** Give the HCL description of a four-way multiplexor that selects as its output (named OUT) one of its four inputs A, B, C, or D, based on two separate selector inputs S_0 and S_1.

```
int OUT = [ !S0 && !S1 : A ;
            !S0 && S1 : B ;
            S0 && !S1 : C ;
            S0 && S1 : D ] ;
```

4. **(6 marks)** Structs and alignment

Consider the following structure definition:

```
struct {
  int a;
  char b;
  char c;
  int d;
  int e[2];
} foo;
```

4a. **(3 marks)** Give the size in bytes and offset in bytes from the beginning of the structure for the following elements of the structure foo.

Name	Size	Offset
b	1	4
c	1	5
d	4	8

4b. **(1 marks)** Assuming that the address of `foo` is in register `%eax`, give the shortest y86 code sequence to load the value of `foo.d` into register `%ecx`.

```
mrmovl    8(%eax), %ecx
```

4c. **(2 marks)** Assuming that the address of `foo` is in register `%eax` and that `i` is in register `%ebx`, give the shortest y86 code sequence to load the value of `foo.e[i]` into register `%ecx`.

```
rrmovl    %ebx, %ecx
iaddl     %ecx, %ecx
iaddl     %ecx, %ecx
iaddl     %eax, %ecx
mrmovl    12(%ecx), %ecx
```

If you are willing to clobber `%ebx` you can shave off an instruction.

5. **(10 marks)** Consider a 4-stage pipeline with stage delays of 50 ps, 120 ps, 100 ps and 120 ps and register (memory) delay of 20 ps per stage. Answer the following questions; you may just give formulas in terms of these numbers; you do not need to do the actual calculations. **Use proper units.**

5a. **(2 marks)** What is the throughput of this processor?

$\frac{1 instruction}{140 ps}$ or 7.14 Gips

5b. **(2 marks)** What is the latency of a single instruction in this processor?

$560ps$ $(4 \times (120 + 20))$

5c. **(2 marks)** Which of these two metrics, throughput or latency, is likely to be a better measure of the performance of the processor? Briefly justify your answer.

Throughput, because it measures the amount of work the processor can do in a unit of time. Latency really only matters when there are many situations that cause the pipeline to be reset (like context switches).

5d. **(2 marks)** You may change this processor into a **5 stage** pipeline by splitting one of the current stages in half. Which stage would you split? Briefly justify your answer, including giving the throughput and latency of the resulting design.

It wouldn't matter since there is still a 120ps stage left after splitting any one stage. The resulting processor would have the same throughput and increased latency ($700ps$ $(5 \times (120 + 20))$.

5e. **(2 marks)** Suppose that on a particular workload the original 4-stage processor achieved a CPI measure of 1.2. What would be the throughput of the processor in terms of useful instructions executed per second?

7.14 Gips / 1.2 = 5.95 Gips

6. **(10 marks)** Each of the following code snippets contains exactly one dependency. Identify the dependency by giving its name. State whether a hazard exists for this code in y86 PIPE (i.e., the best implementation discussed in class and in the textbook that does both data forwarding and branch prediction). Explain how the processor deals with the hazard (if one exists) including a count of bubbles if any are used.

6a.
```
addl    %ecx, %eax
addl    %eax, %ebx
```

dependency: causal

hazard (yes/no): yes

hazard resolution: forwarding

bubble count: 0

6b.
```
mrmovl  12(%ecx), %eax
addl    %ebx, %eax
```

dependency: causal

hazard (yes/no): yes

hazard resolution: forwarding plus bubble

bubble count: 1

6c.
```
    # Result of the last ALU operation is 0
    jne     a               # jump on not-equal-to-zero
    irmovl  $1, %eax
 a:
```

dependency: control

hazard (yes/no): yes

hazard resolution: branch prediction (incorrect)

bubble count: 2

6d.
```
    rrmovl  %eax, %ebx
    mrmovl  12(%ecx), %eax
```

dependency: anti

hazard (yes/no): no

hazard resolution: N/A

bubble count: 0