CPSC 313, 06w Term 1— Midterm 2 — Solutions

Date: November 17, 2006; Instructor: Norm Hutchinson

1. **(10 marks)** Short answers.

1a. **(2 marks)** In the standard pipeline (F, D, E, M, W), does stalling stage D one cycle cause stalls or bubbles for any other cycles? List all of them, indicate which stalls and which bubbles.

Bubble in E and stall in F.

1b. **(2 marks)** Briefly explain why the pipelined version of the y86 processor has control hazards.

Fetching the next instruction in F, but not knowing the address until E or M.

1c. **(2 marks)** How is instruction-level parallelism related to data dependencies?

Data dependencies enforce order on instructions and thus remove instruction-level parallelism.

1d. **(2 marks)** Explain by giving reference to the pipelined implementation of the Y86 processor a very significant difference between RISC and CISC instruction set architectures.

A RISC instruction set includes only operations that fit into the pipelined implementation well, so that excludes complicated instructions, in particular ones whose execution time is highly variable: block move instructions, instructions that reference memory multiple times, etc.

1e. **(2 marks)** What is data forwarding? Why is it useful?

Allowing a early stage to access signals produced by a later stage. It is useful to handle data dependencies without stalling (or with less stalling) by allowing the decode stage to use results computed by instructions in execute and memory stages, even though they have not yet been written back to the register file.

2. **(5 marks)** HCL

2a. **(3 marks)** Give the HCL description of a single circuit that takes two 32-bit integer inputs, A and B, and computes a result OUT that evaluates to:

- 0 if either A or B is negative
- A if A is equal to B
- A - B if A is greater than B
- B - A if B is greater than A

```
int OUT = [ A < 0 || B < 0 : 0;
            A == B         : A;
            A > B          : A - B;
            1              : B - A]
```

2b. **(2 marks)** Give the HCL description of a three input NAND gate (a NAND gate computes the logical AND of its boolean inputs, and then inverts the result). Assume the inputs are named A, B, and C and the output is named OUT.

```
bool OUT = !(A & B & C)
```

3. **(8 marks)** Structs and alignment

Consider the following structure definition:

```
struct {
  int a;
  char b;
  char c;
  int d;
  char e;
  int f[4];
} foo;
```

3a. **(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
d	4	8
f	16	16

3b. **(1 marks)** Assuming that the address of `foo` is in register `%eax`, give the shortest x86 code sequence to load the value of `foo.d` into register `%ecx`.

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

3c. **(2 marks)** Assuming that the address of `foo` is in register `%eax` and that `i` is in register `%ebx`, give the shortest x86 code sequence to load the value of `foo.f[i]` into register `%ecx`.

```
  movl    16(%eax, %ebx, 4), %ecx
```

3d. **(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.f[i]` into register `%ecx`. Note that your code sequence should change only the value of `%ecx`, neither `%eax` nor `%ebx` should change.

```
  rrmovl    %ebx, %ecx       # i in %ecx
  iaddl     %ecx, %ecx       # i * 2 in %ecx
  iaddl     %ecx, %ecx       # i * 4 in %ecx
  iaddl     %eax, $ecx       # &foo + i * 4 in %ecx
  mrmovl    16(%ecx), %ecx
```

4. **(8 marks)** The sequential processor implementation

We wish to add a new instruction to the Y86 instruction set, an indirect branch that transfers control to the address contained in a register. The assembler format for this instruction is:

```
ijmp  *rA
```

4a. **(2 marks)** Devise an encoding for this instruction.

Use a two byte instruction with an unused icode (C0), and the standard encoding of registers (rA, 8) in the second byte.

4b. **(3 marks)** The implementations of the Fetch and Decode stages of this instruction are obvious, and the Memory and WriteBack stages are empty. List what stage outputs (refer to page **??**) will need to be modified for the Execute stage to implement this instruction and sketch how the HCL that computes these outputs will need to be changed.

The only significant outputs that change are aluA and aluB, which need to be valA and 0 respectively. So the HCL for aluA needs to include a clause:

```
  icode == IIJMP : valA;
```

and the HCL for aluB needs to include a clause:

```
  icode == IIJMP : 0;
```

.

4c. **(3 marks)** The `new_pc` output of the PC Update stage is:

```
      int new_pc = [
        icode == ICALL : valC;
        icode == IJXX && Bch : valC;
        icode == IRET : valM;
        1 : valP;
    ];
```

Modify this HCL code to implement the `ijmp` instruction.

```
  int new_pc = [
    icode == ICALL : valC;
    icode == IJXX && Bch : valC;
    icode == IRET : valM;
    icode == IIJMP : valE;
    1 : valP;
];
```

5. **(10 marks)** Consider a 4-stage pipeline with stage delays of 50 ps, 150 ps, 100 ps and 100 ps and memory delay of 10 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?

$\frac{1 instruction}{160 ps}$ or 6.25 Gips

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

$640ps\ (4 \times (150 + 10))$

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 pipeline stalls.

5d. **(4 marks)** Now, 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.

Stage 2 (the 150 ps stage) since the resulting processor would have a throughput of $\frac{1 instruction}{110 ps}$ or 9.09 Gips. The latency would be reduced to $5 \times (100 + 10)$ or 550 ps.

6. **(9 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 implementation discussed in class and in the textbook that does not include data forwarding nor branch prediction). Give the number of pipeline bubbles created when executing the code (if any) and briefly justify this last answer.

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

dependency: anti

hazard (yes/no): no

bubble count: 0

bubble justification: no hazard → no stall, no stall → no bubble

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

dependency: causal

hazard (yes/no): yes

bubble count: 3

bubble justification: use stalls in D until load is through W

6c.
```
   # result of last ALU instruction was zero
   jne     a               # jump on not-equal-to-zero
   irmovl  $1, %eax
   irmovl  $2, %ebx
   irmovl  $3, %ecx
a: irmovl  $4, %edx
   irmovl  $5, %edi
   irmovl  $6, %esi
```

dependency: control

hazard (yes/no): yes

bubble count: 2

bubble justification: target of branch discovered when jne is in E, next instruction stalls twice in F until branch finishes E