CPSC 313, 06w Term 2— Midterm 2

Date: March 23, 2007; Instructor: Norm Hutchinson

This is a closed book exam; no notes; you may use a calculator if you wish. Answer in the space provided; use the backs of pages if needed.

There are **6** questions on **8** pages, totaling **50** marks.

You have **50 minutes** to complete the exam.

On the last two pages you will find summaries of the Y86 instructions and stage outputs of the sequential processor implementation. You may find it profitable to (carefully) remove these pages from the exam.

You should write this exam in pen - I will not consider requests to regrade solutions that are written in pencil.

NAME: _____ **SCORE:** _____ / 50

STUDENT NUMBER: _____

SIGNATURE: _____

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

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

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

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

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-.

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

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

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

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 7 and the stage outputs on page 8, give an HCL implementation of `need_valC`.

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.

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		
c		
d		

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`.

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`.

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?

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

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.

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.

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?

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:

hazard (yes/no):

hazard resolution:

bubble count:

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

dependency:

hazard (yes/no):

hazard resolution:

bubble count:

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

dependency:

hazard (yes/no):

hazard resolution:

bubble count:

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

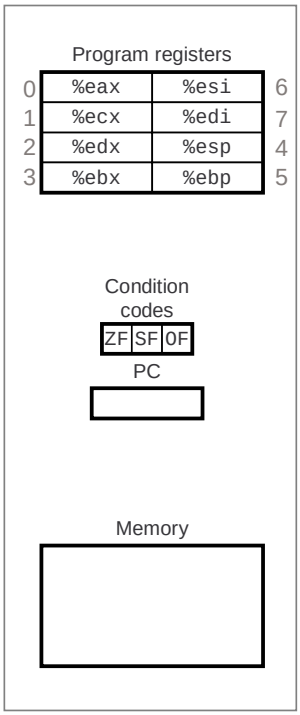
dependency:

hazard (yes/no):

hazard resolution:

bubble count:

y86: a RISC-like and IA32-like ISA



Byte	0	1	2	3	4	5
`nop`	0	0				
`halt`	1	0				
`rrmovl` rA, rB	2	0	rA	rB		
`irmovl` V, rB	3	0	8	rB	V	
`rmmovl` rA, D(rB)	4	0	rA	rB	D	
`mrmovl` D(rB), rA	5	0	rA	rB	D	
`OPl` rA, rB	6	fn	rA	rB		
`jXX` Dest	7	fn	Dest			
`call` Dest	8	0	Dest			
`ret`	9	0				
`pushl` rA	A	0	rA	8		
`popl` rA	B	0	rA	8		

y86: integer and branch function codes

`OPl` rA, rB	6	fn	rA	rB
`jXX` Dest	7	fn	Dest	

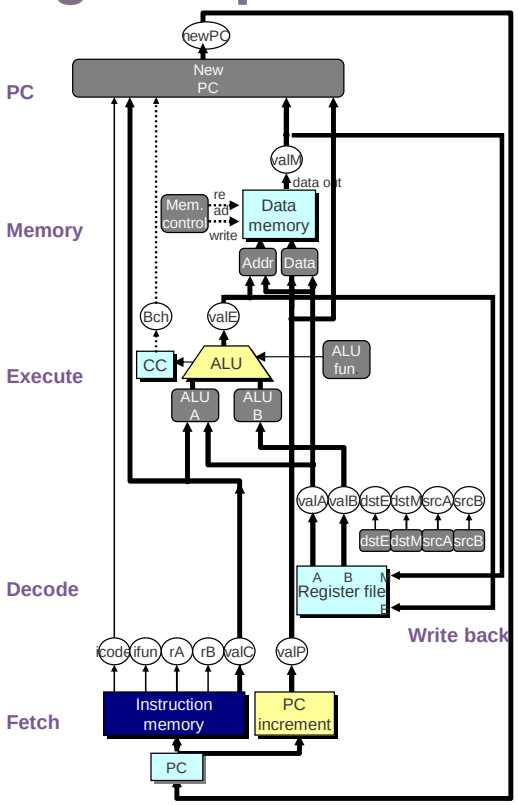
Integer operations

`addl`	6	0
`subl`	6	1
`andl`	6	2
`xorl`	6	3

Branches

`jmp`	7	0
`jle`	7	1
`jl`	7	2
`je`	7	3
`jne`	7	4
`jge`	7	5
`jg`	7	6

stage outputs (the things we need combination circuits to compute)



fetch
- icode, ifun, rA, rB, valC
- valP

decode
- valA, valB
- srcA, srcB

execute
- aluA, aluB, alufun
- set_cc
- Bch
- valE

memory
- mem_addr, mem_data
- mem_read, mem_write
- valM

write back
- dstE, dstM

pc
- new_pc