CPSC 313, 06w Term 1— Midterm 2

Date: November 17, 2006; 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: _____

_____ **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.

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

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

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.

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

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

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.

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		
d		
f		

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`.

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`.

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.

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.

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 8) 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.

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.

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?

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. **(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.

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:

hazard (yes/no):

bubble count:

bubble justification:

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

dependency:

hazard (yes/no):

bubble count:

bubble justification:

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:

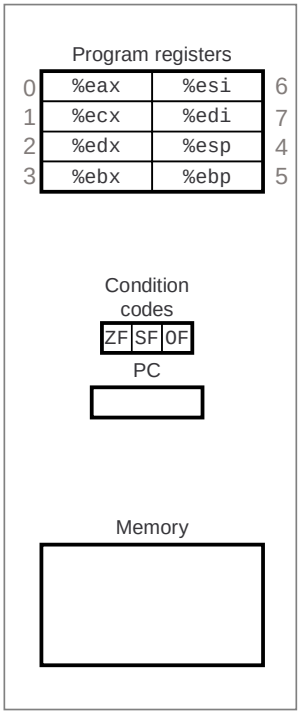
hazard (yes/no):

bubble count:

bubble justification:

You may (carefully, so as to not destroy the staple) remove these last 2 pages from the exam and use them as a reference.

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	

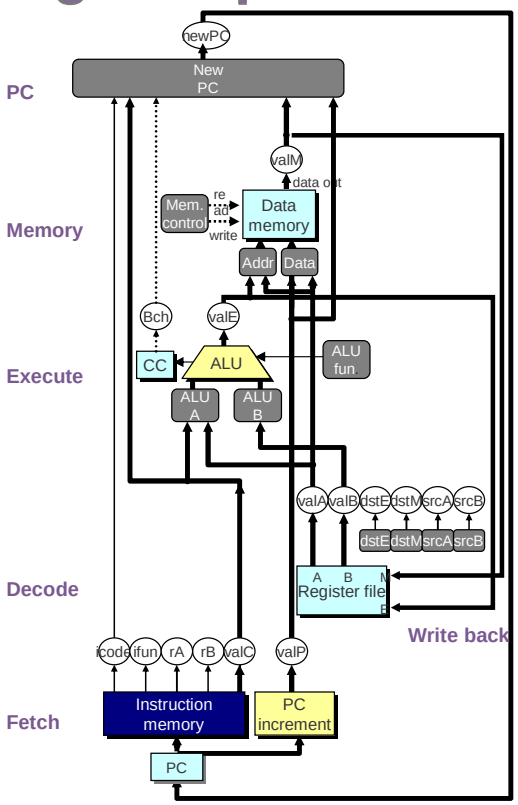
Integer operations

`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