CPSC 313, Winter 2016 Term 1 — Sample

Date: October 2016; Instructor: Mike Feeley

NOTE: This sample contains all of the question from the two CPSC 313 midterms for Summer 2015. This term's midterm will be approximately one third as long as this one.

This is a closed book exam. No notes or electronic calculators. The last page contains the ISA description; for convenience you may remove this page.

Answer in the space provided. Show your work; use the backs of pages if needed. There are **14** questions on **7** pages, totaling **117** marks. You have **180 minutes** to complete the exam.

STUDENT NUMBER: _____

NAME: _____

SIGNATURE: _____

Q1	/ 5
Q2	/ 6
Q3	/ 5
Q4	/ 8
Q5	/ 10
Q6	/ 10
Q7	/ 6
Q8	/ 8
Q9	/ 4
Q10	/ 8
Q11	/ 14
Q12	/ 10
Q13	/ 15
Q14	/ 8

1 (5 marks) The classic RISC pipeline we are studying consists of 5 stages. *Briefly* explain the role of each stage in plain English (i.e., without referring to details such as register names).

2 (6 marks) Consider the following questions about pipeline organization that examine the relative order of each stage. **Include in your answer a Y86 instruction that could not be implemented if the specified ordering was reversed.**

2a Why is the memory stage after the execute stage?

2b Why is the execute stage after the decode stage?

2c Why is the write-back stage after the memory stage?

3 (5 marks) RISC instruction sets do not allow an ALU operation (e.g., addl) to read from memory.

3a Explain how the structure of the pipeline leads to this restriction.

3b Describe how the pipeline could be modified to lift this restriction. You may not change the number of pipeline stages or their basic function. Your revised pipeline does not need to be able to implement every Y86 instruction (in fact it would not be able to) but it must be able to execute instructions like:

```
addl (%eax), %ebx   # r[ebx] = r[ebx] + m[r[eax]]
```

3c This revised pipeline must place some restrictions on the revised ISA that it implements. One of the impacts is on `mrmovl` and `rmmovl`. Describe the problem and modify the instructions so that they will work with the new pipeline. Note that the revised versions need not be as powerful as the original ones, but they must still load and store between memory and a register.

4 (8 marks) Consider the following five-stage pipeline with stage delays (including overheads) of 34 ps, 42 ps, 75 ps, 50 ps and 18 ps. *(NOTE: To simplify the math, you can give answers like 1/100 instead of 0.01 or 1+2+3 instead of 6. Be sure to include units.)*

4a What is the maximum clock rate (i.e., fastest) acceptable for this pipeline?

4b What is the maximum throughput of this pipeline?

4c What, if anything, might cause the actual throughput of programs to be lower than this maximum?

4d What is the minimum instruction latency of this pipeline?

5 (10 marks) Consider the following piece of Y86 assembly code.

```
[0]  addl    %eax, %ebx
[1]  irmovl  $1, %eax
[2]  mrmovl  %(ebx), %ebx
[3]  addl    %ebx, %eax
```

5a List **all** of the data dependencies present in this code. Describe each dependency carefully. Indicate its type (causal, output or anti/alias) and the instructions and registers or memory locations involved.

5b List all of the data hazards present in this code for the Y86, five-stage pipeline.

5c For each hazard, indicate the total number of bubbles added by the Pipe-Minus implementation.

5d For each hazard, indicate the total number of bubbles added by the Pipe implementation.

6 (10 marks) Describe the implementation of the following new instruction for Y86 Seq as you did in Homework 2. This instruction is similar to `mrmovl` except that it adds 4 to `rB` and does not have a static-displacement operand. It would be useful, for example, for iterating over an array of integers.

Syntax:

```
mrmovincl (rB), rA
```

Semantics:

```
r[rA] <= m[r[rB]]
r[rB] <= r[rB] + 4
```

Memory Layout:

5	F	rA	rB

Describe each stage using a relaxed syntax similar as shown below. The Fetch and PC Update stages are complete. List only the code that would be added for this instruction.

Fetch:

```
f.iCd  = m[F.pc] >> 4
f.iFn  = m[F.pc] & 0xf
f.rA   = m[F.pc+1] >> 4
f.rB   = m[F.pc+1] & 0xf
f.valP = F.pc + 2
```

Decode:

Execute:

Memory:

Write Back:

PC Update (pseudo stage):

```
w.pc = W.valP
```

7 (6 marks) Write Y86 assembly code that computes the sum of an array of integers where the address of the array is stored in `%ebx` and the length of the array is in `%ecx`. Place the sum in `%eax`.

8 (8 marks) Answer these questions by saying what the specified feature is and why it is important. Your explanation must refer to specific architectural feature(s) of the processor or memory hierarchy that are designed to execute programs faster when the specified feature exists.

8a spatial locality

8b temporal locality

8c instruction-level parallelism

8d thread-level parallelism

9 (4 marks) Polymorphic dispatch common to object-oriented languages like Java is implemented using a double-indirect call instruction that reads an address from memory and then jumps to it. Explain why it is challenging to implement such an instruction without stalling in a pipelined processor.

10 (8 marks) Consider the following instruction-execution frequencies for a program running on the standard *Y86 Pipe* processor. The table shows, for example, that 7% of all instructions executed read the value of a register immediately after the preceding instruction modified that register by writing into it a value that came from memory.

7%	read register immediately after an instruction writes into that register a value it reads from memory
6%	read register immediately after an instruction writes into that register a value computed in Execute
12%	conditional jump that is taken
8%	conditional jump that is *not* taken
5%	`call`
5%	`ret`
57%	the remaining introduce no bubbles

Your answers can be in the form of a formula using the specific numbers involved. Show your work.

10a What is the CPI (average cycles per instruction) for this execution?

10b What is the throughput of this execution on a 3-Ghz processor (i.e., 3×10^9 cycles per second)?

11 (14 marks) In this question you will consider adding a new stage to *Y86 Pipe* between D and E called M0 and renaming the existing M stage to M1. The resulting pipeline thus looks like this F-D-M0-E-M1-W. For the first five questions, M0 does all memory reads and M1 does all memory writes (thus allowing ALU instructions to operate on values read from memory; e.g., "`addl (%eax), %ebx`". Of course, this does require dropping the base-plus-displacement addressing mode for these instructions, but we will ignore this.

11a This design introduces a new hazard (for the new memory-ALU instructions it enables). Describe it.

11b When this hazard can be resolved by data forwarding, indicate the stage *to* which data is forwarded and state whether it is forward to the beginning or end of that stage.

11c When this hazard can be resolved by data forwarding, indicate the stage or stages *from* which data is forwarded and state whether it is forwarded from the beginning or end of those stage(s).

11d Does this hazard ever require stalling? Explain.

11e Does this new design improve the performance (i.e., reduce stalls) for any of the original *Y86* instructions? Explain.

11f Finally, consider a different design in which M0 and M1 can both read and write memory (and additions to the instruction set to fully utilize this capability). Does this new design introduce the possibility of new hazards? Carefully explain what they are.

12 (10 marks) Consider how to handle the hazards associated with the following new instruction added to *Y86 Pipe* (the one discussed in class, not the modified one you considered in the previous question).

```
cmrmovlXX D(rB), rA
```

Its `iCd` is `I_CMR` and it has the following semantics where XX is one of (le, l, e, ne, ge, g):

```
r[rA] <= m[D +r[rB]] if condition codes indicate that condition XX holds
```

When answering this question, use a simplified syntax for accessing pipeline-stage registers that leaves out `.get()` and `.set()` etc. (but be sure to correctly capitalize the pipeline id). For example you can say things like `if E.valC == W.valP {d.srcA = 0}`. Recall that `cnd` indicates whether the XX condition holds for an instruction (1 if it holds; 0 if it does not).

12a Give the complete data-forwarding logic for `d.srcA` for this instruction (and only this one) as it would appear in the DECODE stage (just `srcA`).

12b Give the complete pipeline-control logic for this instruction (and only this one) that would stall the pipeline when and if necessary.

13 (15 marks) Consider the following Y86 code that contains three data hazards and two control hazards that cause bubbles in *Y86 Pipe*. Note that the program's inputs are `n` and the array `a`; and its output is `s`.

```
[01]        irmovl $1, %eax
[02]        irmovl $4, %ebx
[03]        irmovl $s, %ecx       # ecx = &s
[04]        irmovl $n, %edx       # edx = &n
[05]        irmovl $a, %edi       # temp_a = &a[0] = a
[06]        mrmovl (%edx), %edx   # temp_n = n
[07] LOOP:  subl   %eax, %edx     # temp_n = temp_n - 1
[08]        jl     DONE           # goto DONE if temp_n < 0
[09]        mrmovl (%edi), %esi   # esi = *temp_a
[10]        andl   %esi, %esi     # set CC for *temp_a
[11]        jl     L0             # goto L0 if *temp_a < 0
[12]        mrmovl (%ecx), %ebp   # temp_s = s
[13]        addl   %esi, %ebp     # temp_s = temp_s + *temp_a
[14]        rmmovl %ebp, (%ecx)   # s = temp_s
[15] L0:    addl   %ebx, %edi     # temp_a = temp_a + 4
[16]        jmp    LOOP
[17] DONE:  ...                   # more instructions

.pos 0x1000
s:  .long 0
n:  .long 4
a:  .long 1
    .long 2
    .long 3
    .long 4
```

13a Identify the three data hazards that cause bubbles by giving the line numbers involved.

13b Rearrange the code to eliminate as many data-hazard bubbles as possible for *this particular input*; you may not add or change any instructions. You may describe your change with line numbers or by annotating the code; just be sure its clear what you mean.

Code replicated for convenience:

```
[01]        irmovl $1, %eax
[02]        irmovl $4, %ebx
[03]        irmovl $s, %ecx       # ecx = &s
[04]        irmovl $n, %edx       # edx = &n
[05]        irmovl $a, %edi       # temp_a = &a[0] = a
[06]        mrmovl (%edx), %edx   # temp_n = n
[07] LOOP:  subl   %eax, %edx     # temp_n = temp_n - 1
[08]        jl     DONE           # goto DONE if temp_n < 0
[09]        mrmovl (%edi), %esi   # esi = *temp_a
[10]        andl   %esi, %esi     # set CC for *temp_a
[11]        jl     L0             # goto L0 if *temp_a < 0
[12]        mrmovl (%ecx), %ebp   # temp_s = s
[13]        addl   %esi, %ebp     # temp_s = temp_s + *temp_a
[14]        rmmovl %ebp, (%ecx)   # s = temp_s
[15] L0:    addl   %ebx, %edi     # temp_a = temp_a + 4
[16]        jmp    LOOP
[17] DONE:  ...                   # more instructions
```

13c Identify the two control hazards that sometimes cause bubbles by giving their line numbers.

13d Rewrite the code to eliminate as many total bubbles (data- and control-hazards) as possible for *this input*. In this case you *are* permitted to change or add instructions. If you re-write the entire piece of code (not required), clearly indicate where you made changes and why you made them.

13e If you did this right, your solution will not work equally well (i.e., the same number of bubbles) if the input array contains negative numbers. Explain why and then carefully explain whether it is possible to modify the CPU-pipeline implementation so that this piece of code works nearly as well for arrays that contain all negative values — without changing the number of bubbles for arrays with all positive values — for sufficiently large arrays.

14 (8 marks) Answer these questions that consider changing an existing cache hierarchy to achieve a specific goal. In each case, describe a single change that you think would be most effective in achieving the stated goal. If there are multiple changes that you think would be equally effective, just describe one of them. Justify your answers.

14a Improve the performance of programs that have significant spatial locality.

14b Improve the performance of programs that have a significant number of capacity misses.

14c Improve the performance of programs that have a significant number of conflict misses.

14d Improve the performance of programs that have a significant number of cache hits.

Instructions Encoding

Byte	0		1		2	3	4	5
halt	0	0						
nop	1	0						
rrmovl **rA**, **rB**	2	0	**rA**	**rB**				
cmovXX **rA**, **rB**	2	fn	**rA**	**rB**				
irmovl **V**, **rB**	3	0	F	**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**	F				
popl **rA**	B	0	**rA**	F				

Y86 ISA Reference

Instruction	Semantics	Example
rrmovl %rs, %rd	r[rd] ← r[rs]	rrmovl %eax, %ebx
irmovl $i, %rd	r[rd] ← i	irmovl $100, %eax
rmmovl %rs, D(%rd)	m[D + r[rd]] ← r[rs]	rmmovl %eax, 100(%ebx)
mrmovl D(%rs), %rd	r[rd] ← m[D + r[rs]]	mrmovl 100(%ebx), %eax
addl %rs, %rd	r[rd] ← r[rd] + r[rs]	addl %eax, %ebx
subl %rs, %rd	r[rd] ← r[rd] - r[rs]	subl %eax, %ebx
andl %rs, %rd	r[rd] ← r[rd] & r[rs]	andl %eax, %ebx
xorl %rs, %rd	r[rd] ← r[rd] ⊕ r[rs]	xorl %eax, %ebx
jmp D	goto D	jmp foo
jle D	goto D if last alu result ≤ 0	jle foo
jl D	goto D if last alu result < 0	jl foo
je D	goto D if last alu result = 0	je foo
jne D	goto D if last alu result ≠ 0	jne foo
jge D	goto D if last alu result ≥ 0	jge foo
jg D	goto D if last alu result > 0	jg foo
call D	pushl %esp; jmp D	call foo
ret	popl pc	ret
pushl %rs	m[r[esp] - 4] ← r[rs]; r[esp] = r[esp] - 4	pushl %eax
popl %rd	r[rd] ← m[r[esp]]; r[esp] = r[esp] + 4	popl %eax