CPSC 313 BOOTLEG Final Exam
2007/8 Winter Term 2
April 17, 2008

Name: _____ Student ID: _____

Signature: _____

- You have **150,000 minutes** to write the **countable number of questions** on this examination.
- This examination consists of TODO pages (including this cover sheet). Check to ensure it is complete.
- A total of 8*? marks are available. **You may want to complete what you consider to be the easiest questions first!**
- Justify all of your answers.
- You can use anything you want on the bootleg. Still deciding on aids for the final exam, however.
- Keep your answers short. If you run out of space for a question, you have written too much.
- The number in square brackets to the right of the question number indicates the number of marks allocated to that question.
- Good luck!

Question	Max	Marks
1	?	
2	?	
3	?	
4	?	
5	?	
6	?	
7	?	
8	?	
Total	8*?	

UNIVERSITY REGULATIONS FOR FINAL EXAMS,
WHICH APPLY TO FINAL EXAMS BUT NOT TO BOOTLEG FINAL EXAMS:

- Each candidate should be prepared to produce, upon request, a UBC card for identification.
- Candidates are not permitted to ask questions of the invigilators, except in cases of supposed errors or ambiguities in examination questions.
- No candidate shall be permitted to enter the examination room after the expiration of one half hour from the scheduled starting time or to leave during the first half hour of the examination.
- CAUTION: candidates suspected of any of the following, or similar, dishonest practices shall be immediately dismissed from the examination and shall be liable to disciplinary action.
    1. Having at the place of writing, any books, papers or memoranda, calculators, computers, sound or image players/recorders/transmitters (including telephones), or other memory aid devices, other than those authorised by the examiners.
    2. Speaking or communicating with other candidates.
    3. Purposely exposing written papers to the view of other candidates or imaging devices. The plea of accident or forgetfulness shall not be received.
- Candidates must not destroy or mutilate any examination material; must hand in all examination papers; and must not take any examination material from the examination room without permission of the invigilator.
- Candidates must follow any additional examination rules or directions communicated by the instructor or invigilator.

PROBLEM 1 [? marks]

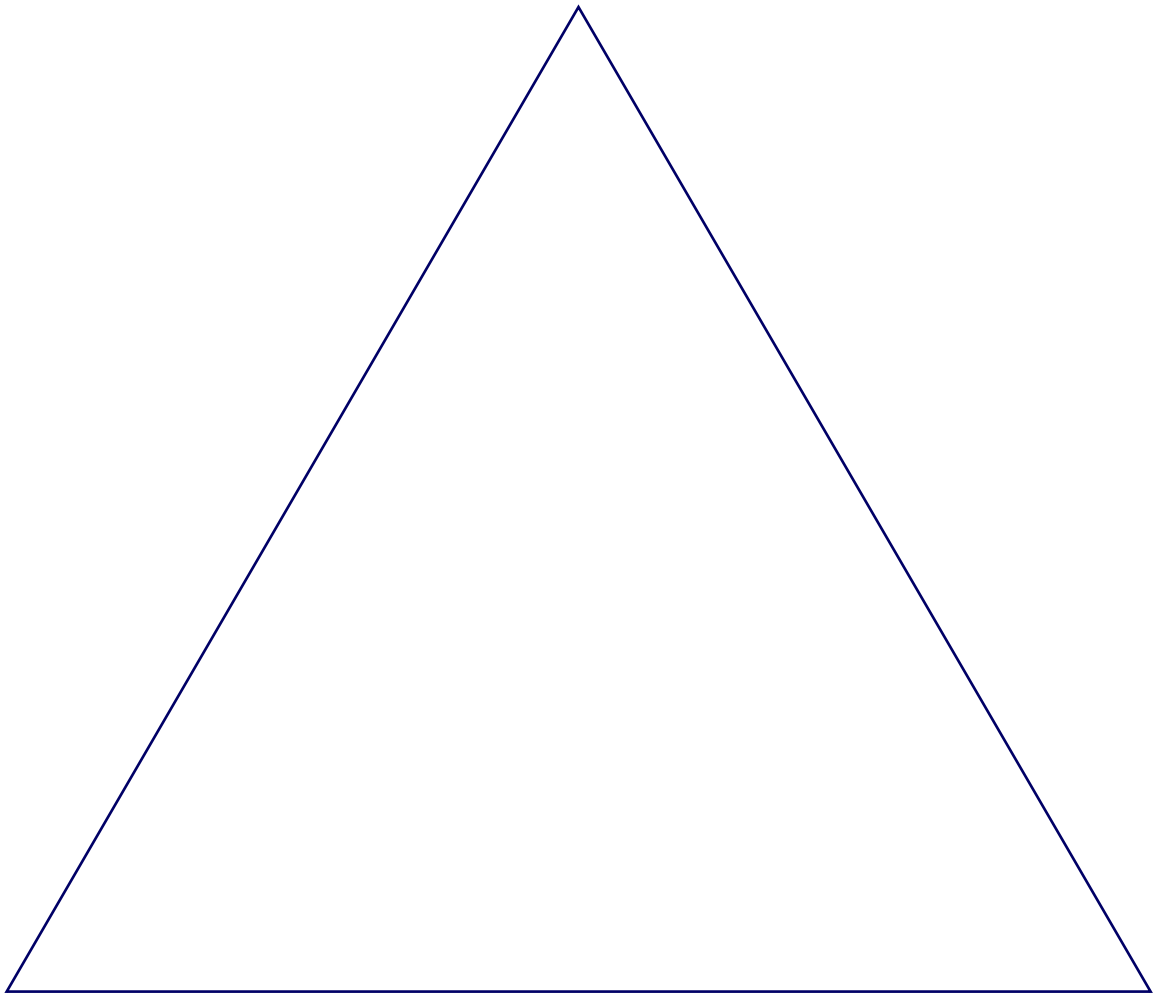
LGs:

- Illustrate with examples how different memory technologies provide different ratios of cost to speed.
- Explain why, despite the fact that VM uses main memory as a cache for disk, VM tends to have significantly different organization and mechanisms from caches.

[An exam problem based on this would probably be a bit more focused but hit similar points.]

Consider the following table of memory technologies. Build a sensible memory hierarchy from it by filling in the following pyramid and labelling the levels L0, L1, L2 …

Type	Speed (1X = 1 cycle)	Cost	Location
XFRAM	1X	500X	(on chip or off chip)
Registers	< 1X	1000X	(in CPU)
DMRAM	10X	100X	(off chip)
KENRAM	5X	500X	(on chip or off chip)
SUPERDISK	1000X	5X	(off board)
SWRAM	100X	10X	(off chip)
DISK	100000X	1X	(off board)
DRUM	100000000X	5X	(next door)



(b) Discuss how each “cache” in the levels of the memory hierarchy might be configured in terms of its size, replacement policy, write policies, and mapping/associativity. Assume the bottom level has 1TB of storage.

PROBLEM 2 [? marks]

LGs:

- Trace the translation of a virtual address to a physical address through a given translation lookaside buffer, page table, and cache to retrieve the physical address.
- Simulate the execution of a fully specified cache on a simple piece of C code or a list of memory requests (read and write), including indicating which requests result in cache hits and misses and the types of the misses (cold, compulsory, or conflict).

[A problem with nearly identical format would make a great final exam question.]

10.13 in the textbook.

PROBLEM 3 [? marks]

LGs:

- Define and give examples of spatial and temporal locality.
- Analyze sources of and impediments to spatial and temporal locality in common coding idioms (such as loops, function calls, variable use, etc.).
- Simulate the execution of a fully specified cache on a simple piece of C code or a list of memory requests (read and write), including indicating which requests result in cache hits and misses and the types of the misses (cold, compulsory, or conflict).

[This is somewhat too intricate for an exam question. Instead, the exam would likely focus on how changes to common algorithms (such as bubble sort) impact spatial and temporal locality (and whether we can tell how they impact these.]

Imagine an L1 cache with 4 8-byte blocks. The cache is fully associative and uses LRU replacement. `int`s are 4 bytes long. Note: LRU is **very** unusual in an L1 cache, but it's convenient so that our by-hand problems work out consistently and sensibly, unlike with random or round-robin replacement.

Here is code for binary search:

```
int findIndexOf(int * array, int length, int target)
{
  int low = 0, high = length - 1;
  while (low < high) {
    int mid = (low + high) / 2;
    int value = array[mid];
    if (value == target)
      return mid;
    else if (value > target)
      high = mid - 1;
    else
      low = mid + 1;
  }
  return -1;
}
```

Consider a binary search over a cache-block-aligned sorted array of 15 integers. Assume that only the array access "`array[mid]`" causes a memory reference. For each part below, assume we start with a cold cache.

(a) If we search for an element that is **not** in this array, are we guaranteed to be able to fit all the `array[mid]` elements we need for the search into our cache?

(b) If we search for the same element that is not in the array twice, what percentage of the `array[mid]` elements accessed in the second search will be hits?

(c) Imagine we searched, in order, for the 1st, then 3rd, then 5th, then 7th … elements of the array. In the following table, fill out which `array[mid]` accesses are hits and which are misses.

Index sought	1st `array[mid]` access	2nd `array[mid]` access	3rd `array[mid]` access	4th `array[mid]` access
0				
2				
4				
6				
8				
10				
12				
14				

(d) Imagine we searched, in order, for the 1st, then 9th, then 3rd, then 11th, then 5th, then 13th … elements of the array. In the following table, fill out which `array[mid]` accesses are hits and which are misses:

Index sought	1st `array[mid]` access	2nd `array[mid]` access	3rd `array[mid]` access	4th `array[mid]` access
0				
8				
2				
10				
4				
12				
6				
14				

(e) Justify the following statement based on what you saw above: "Binary search often has poor spatial locality, but it has good temporal locality for a small set of elements in the array." What is the "small set of elements"?

PROBLEM 4 [? marks]

LGs:

- Describe several approaches for improving the performance of CPU architectures, including pipelining and branch prediction.
- Analyze and manage the challenges that arise when adding these performance improving approaches to CPUs.
- Contrast hardware "algorithm" design with software design, particularly around the issue of parallelism.
- Identify control and data flow hazards in a proposed pipelining scheme.
- Propose and compare (in terms of correctness, efficiency, and complexity) solutions for hazards including data forwarding, branch prediction, stalls, and bubbles.
- Describe the key steps in implementing data forwarding, branch prediction, stalls, and bubbles in our pipelined y86 processor, including changes to data paths and register operation.

[The exam might have a question very similar in format but focused on somewhat different performance issues.]

(a) What properties of hardware and our sequential processor implementation make it so that a pipelined implementation is more efficient than the sequential implementation?

(b) Explain why forwarding (sending results from one instruction direct to the input of another instruction rather than communicating between the instructions through the register file) is unnecessary in the sequential processor implementation.

(c) Would it be **possible** to implement the pipelined processor without forwarding? If so, explain the strategy you would use to correctly execute the sequence of instructions:

```
addl %ecx, %eax
addl %edx, %eax
```

If not, explain what about this sequence of instructions makes forwarding necessary.

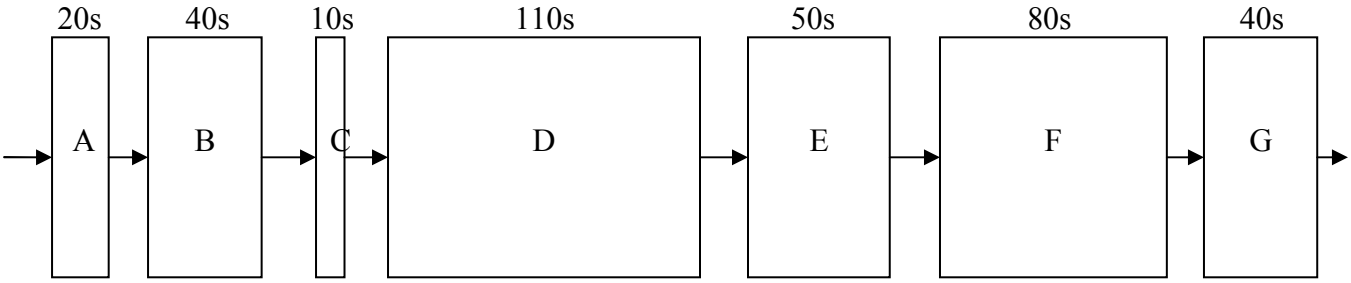
PROBLEM 5 [? marks]

LGs:

- Justify the need for these performance improving approaches based on their measurable impact on performance and their cost in hardware complexity.
- Propose and compare (by clock speed and throughput) alternate breakdowns of a computation into a pipeline given stage timing data (i.e., points at which the computation may be broken into stages, timing for each possible stage, and the registers' delay).
- Support or criticize modifications in the timing of a pipeline stage on the basis of those modifications' impact on throughput and clock speed.
- Propose and compare (in terms of correctness, efficiency, and complexity) solutions for hazards including data forwarding, branch prediction, stalls, and bubbles.

[An exam question might push more on either part (a+b or c+d, that is) of this problem.]

Suppose we analyze the combinational logic of a processor and determine that it can be separated into a sequence of seven blocks, named A to G, having delays of 20s, 40s, 10s, 110s, 50s, 80s, and 40s, respectively, illustrated as below. (Yes, seconds. This is a **very** slow processor!)



(a) What is the best way to divide it into 3 stages? How many cycles-per-second can we execute in the resulting pipeline at peak performance?

(b) What is the best way to divide it into 5 stage? How many cycles-per-second can we execute in the resulting pipeline at peak performance?

(c) The 5-stage version has higher throughput than the 3-stage version. Given that, why would anyone **ever** choose the 3-stage version over the 5-stage version. Give two very distinct, compelling reasons.

(i)

(ii)

(d) A designer proposes forwarding a result from the end of stage F to the beginning of stage B. Roughly how would this change the timing of the stages? Would the new timing change either your 3- or 5-stage solutions?

PROBLEM 6 [? marks]

LGs: various throughout the caching, pipelining, and x86 portions of the course.

[The exam may have a similar question in format, but it will likely cover a different set of class material. Also, it probably wouldn't be quite as open-ended!]

Give two potential disadvantages and two potential advantages of each of the following changes. Make your advantages and disadvantages as distinct from each other as you can!

(a) Increasing the number of pipeline stages in a pipelined processor.

(b) Increasing the size of the L1 cache.

(c) Increasing the associativity of a cache (e.g., going from direct-mapped to 2-way set associative or 2-way set associative to 4-way set associative).

(d) Forwarding some value from a pipeline stage to an earlier pipeline stage.

(e) Adding a `shll` and `shrl` instruction to Y86.

(f) Heavily optimizing a small piece of a C program, including tuning the assembly it produces.

PROBLEM 7 [? marks]

LGs: various throughout the caching, pipelining, and x86 portions of the course.

[The exam will not have a similar question, but several of the concepts touched on here will make appearances in questions on the final.]

Debunk each of the following misconceptions.

(a) If you had plenty of money and wanted to build an ultra-fast computer with tons of memory, you wouldn't bother with caching.

(b) Branch prediction is actually a bad policy. If we just waited two cycles until we knew whether the branch was going to be taken, we could avoid any wasted work!

(c) The more memory a program uses, the worse its locality is.

(d) If you specify the way a processor functions in its ISA, that's the way you have to implement it. [Careful! This isn't entirely a misconception.]

(e) Recursion is inefficient because it causes the stack to grow in proportion to the depth of the recursion.

PROBLEM 8 [? marks]

LGs:

- Translate from Y86 to bit-level encodings.
- Explain the "program prep" code in a Y86 program. (Note: prep code includes both the code at the top and the bottom of a typical full Y86 program. Bear in mind that `ncopy.ys` is actually only PART of a Y86 program! It needs a main (and this prep code) to run.)
- Translate from *very* simple Y86 code snippets to C.
- Translate back & forth between Y86 and X86 snippets.

[An exam question might actually have slightly harder code, as this is *very very* simple, although we wouldn't ask you to produce the prep code out of thin air. (We might ask you to explain it in some detail, however!) Finally, on an exam, this would be a great chance to bang on some x86 issues again, like whether you can clearly model how the ISA works in practice.]

Consider the following y86 code:

```
mystery:
     pushl     %ebp
     rrmovl    %esp, %ebp
     mrmovl    8(%ebp), %edx
     mrmovl    12(%ebp), %eax
     andl      %edx, %edx
     jle       L5
     popl      %ebp
     ret
L5:
     mrmovl    16(%ebp), %eax
     popl      %ebp
     ret
```

(a) Translate this code into C.

(b) Translate this code into x86. (Few changes are needed here. You may want to practice with something more challenging!)

(c) One instruction used here is legal and correct in both x86 and y86, but a different, semantically equivalent instruction would probably have been produced in x86. What is it?

(d) Add the necessary "program prep" code to actually run the function on the arguments `0`, `1`, and `2`, including stack setup! Comment each line of the code.

(e) The "tear down" code for the function is repeated twice. Rewrite the code (in Y86) to use only one copy of the tear down code. Which version is longer as bit-encoded object code and by how many bytes?

ADDITIONALLY:

For pre-midterm goals (assessed but not as heavily as post-midterm goals):

- The midterm and bootleg midterm.
- Your fellow students' midterm makeup questions.

For post-midterm goals:

- Dutch's practice questions on pipeline modification. We **will** have a similar question on the exam, which will also likely require modifying (though not writing) small pieces of HCL code.

And, of course, your problem sets, the challenges to the extent that we discussed them in class, your tutorials, the slides, and your textbook readings!

As a reminder, we are only looking at 10.1-10.6 in Chapter 10, and there will be exactly two questions focused on Chapter 10, one targeted at each learning goal (one of which is "spoiled" on this bootleg).

This bootleg is provided without any guarantees or warranties. It may explode if handled improperly. WARNING: bootleg exams are known to be addictive and can cause low birth weight if used before born.

SERIOUS WARNING: not everything you need to know is covered on this bootleg, but it is a good starting point. Look over the learning goals, especially the ones from Chapter 4 on, but to a lesser extent the Chapter 3 goals as well!

Oh, and post solutions to WebCT. What a pointless thing it would be to have a blatantly wrong answer to one of these questions and not give your instructor or fellow students the chance to correct it!