Midterm Exam

ECE 741 – Advanced Computer Architecture, Spring 2009

Instructor: Onur Mutlu

TAs: Michael Papamichael, Theodoros Strigkos, Evangelos Vlachos

February 25, 2009

EXAM 1 SOLUTIONS

Problem	Points	Score
1	40	
2	20	
3	15	
4	20	
5	25	
6	20	
7 (bonus)	15	
Total	**140+15**	

- This is a closed book midterm. You are allowed to have only two letter-sized cheat sheets.
- No electronic devices may be used.
- This exam lasts 1 hour 50 minutes.
- If you make a mess, clearly indicate your final answer.
- For questions requiring brief answers, please provide brief answers. Do not write an essay. You can be penalized for verbosity.
- Please show your work when needed. We cannot give you partial credit if you do not clearly show how you arrive at a numerical answer.
- **Please write your name on *every* sheet.**

Problem 1 (Short answers – 40 points)

i. (3 points)
A cache has the block size equal to the word length. What property of program behavior, which usually contributes to higher performance if we use a cache, does not help the performance if we use THIS cache?

Spatial locality

ii. (3 points)
Pipelining increases the performance of a processor if the pipeline can be kept full with useful instructions. Two reasons that often prevent the pipeline from staying full with useful instructions are (in two words each):

Data dependencies	**Control dependencies**

iii. (3 points)
The reference bit (sometimes called "access" bit) in a PTE (Page Table Entry) is used for what purpose?

Page replacement

The similar function is performed by what bit or bits in a cache's tag store entry?

Replacement policy bits (e.g. LRU bits)

iv. (3 points)
The fundamental distinction between interrupts and exceptions is that

interrupts are caused by **external events**

and exceptions are caused by **events internal to the running process**.

Interrupts are handled mostly when convenient. Why?

Interrupts are usually not critical to the execution of the running process.

Why are interrupts not always handled when convenient? Give an example.

Because some interrupts are critical to system correctness. For example, machine check interrupt, power failure.

v. (3 points)

A microprocessor manufacturer decides to advertise its newest chip based only on the metric IPC (Instructions per cycle). Is this a good metric? Why or why not? (Use less than 20 words)

No, because the metric does not take into account frequency or number of executed instructions, both of which affect execution time.

If you were the chief architect for another company and were asked to design a chip to compete based solely on this metric, what important design decision would you make (in less than 10 words)?

Make the cycle time as long as possible and process many instructions per cycle.

vi. (3 points)

p% of a program is perfectly parallelizable. We wish to run it on a multiprocessor. Assume we have an unlimited number of processing elements. If the maximum speedup achievable on this program is 100, what is p?

$$a)\ assuming\ p = 100x: \frac{1}{1-p} = 100 \Rightarrow p = \mathbf{99}\%$$

$$b)\ assuming\ p = 100\%: \frac{1}{1-p} = 1 + 100\% = 2 \Rightarrow p = \mathbf{50}\%$$

vii. (5 points)

Name two techniques that eliminate the need for hardware-based interlocking in a pipelined processor:

Compiler has knowledge of micro-arch. and ensures no interlocks are needed.	**Fine-grain multithreading**

viii. (6 points)

Remember that the history buffer is a structure that enables precise exceptions in a pipelined machine. Briefly describe how the history buffer works during the execution of a register-to-register ADD instruction (ADD RD ← RS1 + RS2) by completing the following sentences:

When the add instruction is decoded **an entry is allocated at the tail of the HB and tail pointer is incremented.**

When the add instruction completes execution **the old value of RD is stored in the HB entry.**

When the add instruction is the oldest completed instruction in the pipeline and

i) An exception has occurred during its execution,

the HB is traversed from tail to head and the old value of destination register in each HB entry is written into the register file.

ii) No exception has occurred during its execution,

the HB entry of ADD is deallocated and the head pointer of the HB is incremented.

Assume we would like to use the exact same solution (history buffer) for executing a store instruction to memory. Why is this difficult to do?

It is difficult to UNDO the effect of a store instruction. Another processor might read and use the value supplied by the store instruction before the store is undone.

Very briefly describe one solution to handling store instructions in conjunction with a history buffer. (Use less than 30 words)

A separate store buffer which commits stores to the memory system in program order. No store is made visible to other processors out of program order.

ix. (5 points)

Suppose you are designing a small, 16KB, 2-way set-associative L1 and a large, 32MB, 32-way set-associative L3 cache for the next processor your company will build. Which one of the following design decisions would you make and why? Justify your choice.

Access L1 tag store and data store: in parallel OR series (circle one and explain)

Parallel. L1 access latency is critical to performance. Parallel tag/data access reduces average L1 access latency.

Access L3 tag store and data store in parallel OR series (circle one and explain)

Serial access. L3 access is not in the critical path of pipelined execution. Serial access also saves energy, as there is no need to fire up all 32 cache way and then discard all (but one, sometimes).

x. (6 points)

Suppose you are designing a computer from scratch and that your company's budget allows a very small amount of memory bandwidth. Which of the following characteristics would you choose in the ISA and the microarchitecture, and why? Explain briefly.

Variable length instructions or fixed length instructions?

Variable length → smaller code footprint → less bandwidth to transfer instructions

Complex instructions or simple instructions?

Complex instructions → smaller code footprint (each instruction is more powerful) → less bandwidth to transfer instructions

A **large L2 cache** or a small L2 cache? (L2 is the last-level cache)

Large L2 cache → higher hit rate → less need to access memory

An aggressive prefetcher or a **conservative prefetcher**?

Conservative prefetcher → likely to be more accurate → less wasted memory bandwidth

Large cache blocks vs. **small cache blocks**?

Small cache blocks → likely that a larger fraction of each block is useful → less need to transfer useless data on a cache miss

Problem 2 (20 points)

You designed a microprocessor. It came back from the fab with an error: one of the bits is stuck. We call the bit a stuck-at-0 fault if the bit is always 0 (i.e., you cannot store a 1 in it). We call the bit a stuck-at-1 fault if the bit is always 1 (you cannot store a 0 in it).

Consider each of the structures below independently. Assume the structure contains a stuck at 0 or 1 fault. Does the fault affect the correctness of the chip? Does the fault affect performance? Explain. (Note: For each structure, consider separately stuck-at-0 and stuck-at-1 faults.) No error detection and correction mechanisms are present.

A bit in the register scoreboard:

Affects correctness in both cases. If a bit in the scoreboard is stuck at 0, the register will always be 'ready' and that will break dependencies.
If it is stuck at 1, the register will always be 'busy' and that will eternally stall the processor waiting for it to become ready.

The dirty bit in one of the tag store entries of the L2 cache:

Stuck-at-0: affects correctness. A modified block will never be written to memory, causing wrong results in the program.
Stuck-at-1: affects (reduces) performance. Even an unmodified block will be written back to memory, which wastes memory bandwidth and causes contention.

The LRU bit for one of the sets of a 2-way set associative cache:

Performance in both cases. LRU bit is only for replacement urposes. The LRU replacement will be broken, as we will always evict the block in either way 0 (stuck-at-0) or way 1 (stuck-at-1), but this does not break correctness.

A bit in the "predicted next instruction address" supplied by the branch predictor:

Does not affect correctness since the predicted next instruction address is speculative anyway. A wrong prediction due to a stuck-at fault will simply trigger recovery.

Affects performance: reduces prediction accuracy since the predicted target address is wrong more often → more pipeline flushes

A bit in the stride value stored in a stride prefetcher:

Does not affect correctness since prefetching a wrong address is not related to correctness (prefetching is purely speculative).

Affects performance: reduces prediction accuracy for the prefetched addresses → likely wrong data is prefetched, wasting memory bandwidth and causing cache/prefetch-buffer pollution.

Problem 3 (15 points)

Your job is to evaluate the potential performance of two processors, each implementing a different ISA. The evaluation is based on its performance on a particular benchmark. On the processor implementing ISA A, the best compiled code for this benchmark performs at the rate of 10 IPC. That processor has a 500 MHz clock. On the processor implementing ISA B, the best compiled code for this benchmark performs at the rate of 2 IPC. That processor has a 600 MHz clock.

What is the performance in MIPS (millions of instructions per sec) of the processor implementing ISA A?

$$10\ IPC \cdot 500\ MHz = \mathbf{5000}\ \boldsymbol{MIPS}$$

What is the performance in MIPS (millions of instructions per sec) of the processor implementing ISA B?

$$2\ IPC \cdot 600\ MHz = \mathbf{1200}\ \boldsymbol{MIPS}$$

Which is the higher performance processor? A B **Don't know**
Explain.

Not enough information provided. We don't know the number of instructions executed on each processor, so we can't compare performance. Processor A for example may have very simple instructions (or NOPs). Processor B could be faster if its instructions actually accomplish significantly more work than Processor A's.

The MIPS metric does not take into account number of executed instructions, which also affects the performance (i.e. execution time) of each processor.

Problem 4 (20 points)

As you remember from the lecture, SPEC numbers are measures of the time it takes to execute certain representative benchmark programs. A higher number means the execution time of the corresponding benchmark(s) is smaller. Some have argued that this gives unfair advantage to processors that are designed using a faster clock, and have suggested that the SPEC numbers should be normalized with respect to the clock frequency, since faster clocks mean shorter execution time and therefore better SPEC numbers. Is this suggestion a good or a bad idea? Explain.

Bad idea. Using this metric penalizes processors that have shorter clock cycle time (i.e. higher clock frequency).

If you are told that your design will be evaluated on the basis of its SPEC/MHz number, what major design decision would you make?

$$\frac{SPEC}{MHz} = \left(\frac{instructions}{program} \cdot \frac{cycles}{instruction} \cdot MHz\right)^{-1} \cdot \frac{1}{MHz} = \left(\frac{cycles}{program}\right)^{-1}$$

Extend the clock cycle time (i.e. lower the clock frequency) as much as possible and process as many instructions as possible in a single clock cycle (e.g., very wide issue width).

Problem 5 (25 points)

We have a byte-addressable toy computer that has a physical address space of 512 bytes. The computer uses a simple, one-level virtual memory system. The page table is always in physical memory. The page size is specified as 8 bytes and the virtual address space is 2 KB.

Part A.

i. (1 point)
How many bits of each virtual address is the virtual page number?

$$\log_2\left(\frac{virtual\ address\ space\ size}{page\ size}\right) = \log_2\left(\frac{2\ K}{8}\right) = \mathbf{8\ bits}$$

ii. (1 point)
How many bits of each physical address is the physical frame number?

$$\log_2\left(\frac{physical\ address\ space\ size}{page\ size}\right) = \log_2\left(\frac{512}{8}\right) = \mathbf{6\ bits}$$

We would like to add a 128-byte *write-through* cache to enhance the performance of this computer. However, we would like the cache access and address translation to be performed simultaneously. In other words, we would like to index our cache using a virtual address, but do the tag comparison using the physical addresses (virtually-indexed physically-tagged). The cache we would like to add is direct-mapped, and has a block size of 2 bytes. The replacement policy is LRU. Answer the following questions:

iii. (1 point)
How many bits of a virtual address are used to determine which byte in a block is accessed?

$$\log_2(block\ size) = \log_2(2) = \mathbf{1\ bit}$$

iv. (2 point)
How many bits of a virtual address are used to index into the cache? Which bits exactly?

$$\log_2(\#blocks) = \log_2\left(\frac{cache\ size}{block\ size}\right) = \log_2(64) = \mathbf{6\ bits}$$

Bits 1..6 (Bit 0 is used for byte in block)

v. (1 point)
How many bits of the virtual page number are used to index into the cache?

4 bits. Bits 3..6 (Bits 0..2 are the page offset and bits 1..6 are used as index).

vi. (5 points)
What is the size of the tag store in bits? Show your work.

Per cache block we have tag + valid bit (no dirty bit for write-through cache).
Given that we use bits of the virtual page number to index in the cache (see v.), and we have physically tagged cache, we must use the entire physical page number as the tag.
$Tag\ size = 6\ bits\ (from\ ii)$ → Tag store entry size = 6 bits + 1 bit (valid bit) == 7 bits

$$Total\ tag\ store = \#blocks \cdot (tag\ store\ entry\ size) = 64 \cdot 7 = \mathbf{448\ bits = 56\ bytes}$$

Part B.

Suppose we have two processes sharing our toy computer. These processes share some portion of the physical memory. Some of the virtual page-physical frame mappings of each process are given below:

PROCESS 0	
Virtual Page	Physical Frame
Page 0	Frame 0
Page 3	Frame 7
Page 7	Frame 1
Page 15	Frame 3

PROCESS 1	
Virtual Page	Physical Frame
Page 0	Frame 4
Page 1	Frame 5
Page 7	Frame 3
Page 11	Frame 2

vii. (2 points)
Give a complete physical address whose data can exist in two different locations in the cache.

Any byte in frame 3 is seen as 2 different virtual pages from processes 0, 1.
Physical address of the first byte in frame 3 = **000011000**

viii. (3 points)
Give the indexes of those two different locations in the cache.

Corresponding virtual address from process 0 (Page 15) = 0000**111100**0 (index in bold)
Corresponding virtual address from process 1 (Page 7) = 0000**011100**0 (index in bold)

ix. (5 points)
We do not want the same physical address stored in two different locations in the 128-byte cache. We can prevent this by increasing the associativity of our virtually-indexed physically-tagged cache. What is the minimum associativity required?

4 bits of the index come from the virtual page number. All of these might change during translation. To eliminate all these bits from the index, we should have at least 2^4 ways in the cache → minimum associativity = **16 ways**

x. (4 points)
Assume we would like to use a direct-mapped cache. Describe a solution that ensures that the same physical address is never stored in two different locations in the 128-byte cache.

Page coloring: The should ensure that the virtual page number and physical frame number always have the same lower four bits (i.e., the cache index bits that come from the virtual page number do not change during the virtual→physical address translation)

Alternate answer: When a cache fill happens for a physical address, extra hardware checks all other possible locations in the cache where the same physical address can be mapped to and invalidates those other locations after merging their modified data with the incoming block from memory.

Problem 6 (20 points)

A processor has an 8-bit physical address space and a physically addressed cache. Memory is byte addressable. The cache uses perfect LRU replacement.

The processor supplies the following sequence of addresses to the cache. The cache is initially empty. The hit/miss outcome of each access is shown.

Address	Outcome
0	Miss
2	Hit
4	Miss
128	Miss
0	Hit
128	Hit
64	Miss
4	Hit
0	Miss
32	Miss
64	Hit

Your job: Determine the block size, associativity, and size of the cache. Note: It is not necessary to give an explanation for every step, but you should show sufficient work for us to know that you know what you are doing.

- Sequence 0M → 2H → 4M ⟹ 2 (2 hit) < block size ≤ 4 (4 miss) ⟹
  block size = 4 bytes (always power of 2)
- Sequence 0M → 128M → 0H → 128H → 64M → 0M
  64 Miss affected 0 ⟹ 64 maps in the same set as 0 ⟹ 128 maps in the same set as zero
  But the first miss of 128 did not affect the presence of 0 ⟹ at least 2-way set associative.
  If it was more than 2-way set-associative then 64 wouldn't have evicted 0. ⟹
  Exactly **2-way set-associative**.
- 64 maps to the same block as 0 ⟹ size is at most 64 bytes/way
  Consider the case where it has 32 bytes/way or less or exactly 64 bytes/way, and focus on set 0. The following table shows the address of the first byte for each of the 2 blocks of set 0.

Access	Outcome	64 bytes/way		≤ 32 bytes/way	
		MRU	LRU	MRU	LRU
0	Miss	0+		0	
128	Miss	128	0	128	0
0	Hit	0	128	0	128
128	Hit	128	0	128	0
64	Miss	64	128	64	128
0	Miss	0	64	0	64
32 *	Miss	0	64	32	0
64	Hit	64	0	64 would miss	

  * 32 maps to different set for the large size.
  Thus exactly 64 bytes/way * 2 ways ⟹ **cache size = 128 bytes**.

BONUS Problem 7 (15 points)

Suppose you have designed the next fancy hardware prefetcher for your system. You analyze its behavior and find the following:

i) The prefetcher successfully prefetches block A into the cache before it is required by a load instruction. The prefetched block evicts a never-to-be-used block from the cache, so it does not cause cache pollution. Furthermore, you find that the prefetch request does not waste bus bandwidth needed by some other request.

ii) The prefetcher successfully prefetches block B into the cache before it is required by a load instruction. The prefetched block evicts a never-to-be-used block from the cache, so it does not cause cache pollution. Furthermore, you find that the prefetch request does not waste bus bandwidth needed by some other request.

Upon further analysis, you find that the prefetching of block A actually reduced execution time of the program whereas prefetching of block B did *not* reduce execution time significantly. Describe why this could happen. Draw two execution timelines, one with and one without the prefetcher, to illustrate the concept.

Given that both prefetch requests were used, no prefetch wasted bandwidth or caused cache pollution, but only A reduced execution time significantly, we must conclude that the miss of A was more expensive than the miss of B. Possible reason: Memory Level Parallelism. The miss of B was parallel with another miss (e.g. C) that wasn't prefetched (i.e., the processor still needs to stall for the parallel miss) whereas the miss of A was an isolated miss, eliminating which reduced the stall time.