Fall 2013: COMP 7300 Advanced Computer Architecture

Test 1 (100 pts) (1h:15)

LAST PAGE EXERCISE IS BONUS

Grading policy:

¼ Credit for correct answer

I mean by "credit", the credit of the question, not the credit for the full test.

Guidelines for grading concepts response

1) Bad/wrong facts (-1/3)
2) Wrong inference from facts / bad reasoning(-1/2)
3) Right answer without ANY justification (-3/4)
4) Right answer with clumsy/shaking arguments (-1/4)

Guidelines for Exercises on cache

1) Wrong NUMBER of bits for Offset, index, or tag (-10%) but, the following results will be accepted as long as they are consistent with the mistake
2) Wrong miss/hit: (-5%)
3) Wrong offset, index, tag in the cell (-1/40 per wrong cell)

Guidelines for grading quantitative exercises:

0) Wrong application of formula (not plugging the right numbers in formula) (-1/3 credit)
1) Missing expression (-1/3 of credit) with right result
2) Missing expression without a right answer (- 100% credit)
3) Good expression with a wrong final result (no credit is taken)
4) Wrong concepts/expression (-1/2)

¾ Credit for well written and solid justification/facts/arguments.

A) Introduction (15 points)

1) (**3 points**) This semester, we stressed that modern computer architecture focuses on meeting specific requirements of the target machine and aims to maximize performance within the following key constraints: cost, power/energy, and speed [**Fill in the blank**]

2) **(12 points) Exercise**:

We consider a program ***P*** executed on an imaginary processor with a clock rate of **4 GHz**. This table provides the number of executions for each type of instruction and the number of cycles. Answer the questions based on this table.

	Number of instructions	(CPI) Clock cycle Per Instruction
Load/Store	400	6
Arithmetic	500	4
Logical	300	2

a) **(1.5 point)** How many clock cycles in total are needed to execute Program ***P***?

Total Number of Clock Cycles = 400*6 + 500 * 4 + 300 * 2 = 2400 + 2000 + 600 = 5000 cc

b) **(1.5 point)** What is the average CPI (average number of clock cycles per instruction) for Problem ***P***?

There are (400 + 500 + 300) instructions, CPI = Total Number of clock cycles (cc) /Number of instructions

= 5000 cc/ 1200 instructions = 4.17 cc/instruction

c) **(1 point)** What is the latency (in ns) of **one** *Load/Store* instruction?

One Load/store takes 6 cc. One cc duration = 1/clock rate = 1 / $4*10^9$Hz = 1 / $4*10^9$ *1/s = 0.25 10^{-9} s = 0.25 ns. Therefore one Load/store takes 6 * 0.25n = 1.5 ns

d) (1 point) How long does Program ***P*** take to execute (in **microseconds**)?
The program takes 5000 cc to execute, then P will execute in 5000 * 0.25ns = 1250 ns = 1.25 µs.

e) (3 points) To speed up program P, **John** proposes to speed up **only** Load/Store instructions by a factor of 6.

$$\text{Execution time}_{\text{new}} = \text{Execution time}_{\text{old}} \times \left((1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right)$$

Using Amdahl's formula, what is the new execution time for Program P?
Fraction enhanced = Number of Load/Store instructions x 6 / total number of clock cycles = 2400/5000 = 12/25. Speed up = 6. Then
New Time = 1.25 µs x [(1-12/25)+12/25/6) = 1.25 µs x [13/25 +2/25] = 1.25 µs x [15/25]
1.25 µs x [3/5]

f) (3 points) To speed up program P, **Paul** prefers to p to speed up **all instructions OTHER** than Load/Store instructions by a factor of 2. Using Amdahl's formula, what is the new execution time for Program P?
Fraction enhanced = Number of Load/Store instructions / total number of instruction
= (500 x 4 + 300 * 2) /5000 = 2600/5000= 26/50 = 13/25. Speed up = 2. (bad parameters -1 point)
Then
New Time = 1.25 µs x [(1-13/25)+13/25/2) = 1.25 µs x [12/25 +13/50] = 1.25 µs x [37/50]

g) (1 point) Which improvement (John's or Paul's) yields the best overall speed-up?
John Improvement time / Paul's Improvement time = 3/5 / (37/50) = 30/37 < 1 Then John's time is smaller => John's improvement is better.

B) (65 points) Memory Concepts

1) (3 points) Two key **requirements** lead to build a memory as an **hierarchical** memory system. **Cite** these two requirements?
Requirement 1: User/programmer wants large/infinite memory
Requirement 2: User/programmer wants fast memory

2) (3 points) Two key **facts** lead to build a memory as an **hierarchical** memory system. **Cite** these two facts?
Fact 1: Faster memory is more expensive
Fact 2: Programs in general exhibit some temporal locality

3) (20 points) We consider a **2-way** associative cache with size 64 bytes and a block size of 8 bytes. Assume 16 bit addresses.

a) (2.5 points) How many bits does the offset have?
A block has 8 bytes = 2^3 then Offset has 3 bits

b) (2.5 points) How many sets does the cache have?
The total number of blocks in the cache is equal to Cache Size/ Block Size = 64 / 8 = 8 blocks.
The cache is 2-way associative then each set contains 2 blocks, then number of sets = number of blocks / number of blocks per set = 8 / 2 = 4 sets

c) (2.5 points) How many bits does the index have?
Number of sets = 4 = 2^2 then Index takes 2 bits

d) **(2.5 points)** How many bits does the tag have?

Number of bits in tag = Total number of bits in address – number of bits in index – number of bits in offset = 16 – 2 -3 = 11 bits

e) **(10 points)** The first column on the table below shows the addresses (to bytes) that a CPU generates. You may omit leading 0s. First, complete the table and then tell whether each address is a hit or a miss assuming that the LRU replacement policy is used.

Address (Hexa)	Address (Binary)	Tag (binary)	Index	offset	Miss/Hit
0x98	100 11 000	100	11	000	miss
0xBF	101 11 111	101	11	111	miss
0x78	011 11 000	011	11	000	miss
0xAF	101 01 111	101	01	111	miss
0x88	100 01 000	100	01	000	miss
0xA8	101 01 000	101	01	000	hit
0xB8	101 11 000	101	11	000	hit
0xCF	110 01 111	110	01	111	miss
0xAC	101 01 100	101	01	100	hit
0x9F	100 11 111	100	11	111	miss

)

4) **(20 points)** Below is a *questionable* (i.e., maybe not true) table summarizing the characteristics memory chips:

Memory Type	Access Time	Cost per GB
DSRAM	50 ns – 70 ns	~$50
Flash	20 ns – 35 ns	~$2
SRAM	0.5ns – 2.5 ns	$2,000 - $5,000
Magnetic Disk	5 ms – 20 ms	$3

a) **(4 points)** This table may contain some *questionable* (i.e, doubtful, not true) facts. Which cells/facts are questionable? Why?

The access time of the Flash is less than the SDRAM's and the Flash costs less than SDRAM's. If this was true, main memory would be made of Flash. (DSRAM is a typo of SDRAM, but do not take off points)

b) **(5 points) Assume for now that the table is accurate.** Based on the table above, which memory type would most likely be used for L1 caches (closest to CPU)?

SRAM because it has the smallest access time

c) **(5 points) Assume that the table above is accurate (true)**. Based on the table above, which type of memory should be used for the main memory?

Based on the table, the flash should be used because it is faster and cheaper than SDRAM. Furthermore, it is more expensive than SRAM.

d) **(6 points) Assume that the table above is accurate (true),** how many levels will this memory system have? Cite each memory level and the memory type used at each level

Based on the table, the Flash is faster and less expensive than SDRAM and the magnetic disk. In addition, it is non volatile. If the table was accurate, the system would have only two levels: cache (SRAM) and main memory (flash)

5) **(17 points)** Paul and John are arguing about the size of a cache block. Paul wants a larger block size.

a. **(3 points)** John states that a larger block size leads to a higher miss penalty. Is John right?

John is right because the loading time increases with the size of a block, therefore, the miss penalty will increase.

b. **(4 points)** John states that a larger block size will **always** decrease the miss rate. Is John right?

John is not right. On one side, a larger block size will decrease the number of compulsory misses if the program has a good spatial locality. On the other end, larger blocks mean fewer blocks in the cache leading therefore to more conflict and capacity misses. So, larger block will not ALWAYS decrease the miss rate,

c. **(4 points)** John argues that *Early Restart* will halve (divide by 2) on average the miss penalty regardless of the block size.

John is right because the needed word will be on average in the middle of the block. Since the word requested is delivered as soon as the word is encountered during the loading. Therefore, the requested word is "delivered" on average after loading half the block, so the miss penalty will get halved.

d. **(3 points)** Paul argues that if *Critical-Word-First* is used, the miss penalty will not increase with a larger block. Explain what *Critical-Word-First* is. Is Paul right?

With Critical=Word-First, the loading starts at the requested word. Therefore, the requested word will be delivered as soon as the loading starts. So, the miss penalty does not vary with the size of a block.

e. **(3 points)** John argues that if *Critical-Word-First* is used, a larger block will decrease the miss rate for program with high special locality. Is John right?

Critical-Word-First does not have any impact on the miss rate: this strategy is taken AFTER a miss occurs. It has an impact on the miss penalty.

Exercises (30 points)

6) **(22 points) Exercise 1**: **In addition** to the data block storage, a cache must store a validity bit and a tag for each data block. The validity bit and the tag incur a ***storage overhead***. The *storage overhead* is expressed as:

This exercise explores the storage overhead for direct-mapped and n-way associative caches. We assume 16-bit addresses with a cache ***C*** that accommodates 32 KB of data. The block size is16-words (one word is four bytes).

a. **(8 points)** How many total bits are required if the cache ***C*** is a direct-mapped cache?

Each entry includes a validity bit, a tag, and a block of data. We need the size of a tag.
The block size is 16 words = 64 bytes = 2^6 bytes, then offset has 6 bits.
The total number of blocks – Cache size / Block size = 32 KB/64 = $2^{15}/2^6 = 2^9$. Then, the number of sets is equal to 2^9 because the cache is direct-mapped (one block per set). Then the index has 9 bits. Tag has then 16 – 9 – 6 = 1 bit.Total number of bits = number of sets X number of bits per entry
= 2^9 * (1 bit for Validity + 1 bit for tag + 9 bits of the index = 2^9(1 + 1+ 64 * 8) = 2^9 x (2 + 2^9)
= 2^9 x (514) = 2^{10} x 257 = 257 Kbits.

b. **(2 point)** What is the storage overhead for the above direct-mapped cache?

By the definition on the test, the storage for validity bit and tag = 2^9 x (1 +1) = 2^{10} bits = 1 Kbits.
Storage of data is provided as 32 KB, So, Overhead Storage = 1 Kbits/ 32 KB = 1/(32 x 2^{10} x 2^3)
= $2^{10}/2^{18}$ = $1/2^8$

Another way to do it, the storage overhead can be computed directly from looking at ONE entry in the cache. One entry contains (1 bit (v) + 1 bit (Tag) + 64*8) = 2 bits (overhead) + 2^9
The storage overhead is then 2 / 2^9 = $1/2^8$

c. **(8 points)** How many total bits are required if the cache ***C*** is a 4-way associative cache?

If the cache is 4-way associative, then the number of bits in the index changes because the number of sets is different. With direct-mapped, the number of sets was 2^9. Since the cache is 4-way, the number of sets becomes 2^9 / 4 = 2^7. Then the index takes 7 bits. Then tag will take 16 – 7 bits (index) – 6 (offset) = 3 bits
One cache entry will contain 4 blocks with their tags and validity
= V1 + Tag 1 + one data block data +
+ V2 + Tag 2 + one data block
+ V3 + Tag 3 + one data block
+ V4 + Tag 4 + one data block
= 4 x (1 bit + 3 bits + 2^6 bytes)
= 4 bits + 12 bits + 4 x 2^6 bytes
= 4 bits + 12 bits + 4 x 2^9 bits

The total storage will be number of sets x the size of one entry = 2^7 x (16+ 2^{11} bits) = 258Kbits

d. **(2 points)** What is the storage overhead for the above 4-way associative cache?

As shown above, the storage overhead = 16 /2^{11}=$1/2^7$.

e. **(2 points)** Which method (direct-mapped or 4-way associative) offers the best overhead?

With direct-mapping the storage overhead was $1/2^8$ while with the 4-way associative cache, it is $1/2^7$.
Therefore the storage overhead for direct-mapping's is half the 4-way, direct-mapping is better.

7) **(10 points) BONUS Exercise 2:** The objective of this exercise is to derive the formula for the average number ***AvgCPI*** of clock cycles : ***AvgCPI = CPI + mr * mp*** where *mr* is the miss rate, *mp* is the miss penalty, and *CPI* is the number of clock cycles per instruction for an **ideal** cache (*mr* = 0).

a) **(1 point)** Suppose that the cache is **ideal**, express ***AvgCPI*** as a function of CPI.

AvgCPI = CPI because there is no miss with an ideal cache

b) **(1 point)** For **all the following** questions, we assume that the cache is not ideal and that a program has ***n*** instructions. Let us compute the number ***h*** of instructions that experience a cache hit. Express ***h*** as a function ***n*** and the miss rate ***mr***.

Since the miss rate is mr, an instruction experiences a hit with probability (1 – mr).
Then h = n x (1 –mr)

c) **(1 point)** Express the number of clock cycles that ***h*** instructions (hits) take.

Since the h instructions are hits, each one of them takes CPI. So,
The h instructions will take h x CPI = n x (1-mr) x CPI

d) **(1 point)** Let us compute the number ***m*** of instructions that have a cache miss. Express ***m*** as a function ***n*** and the miss rate ***mr***.

Each instruction will trigger a miss with probability mr.
Therefore m = n x mr

e) **(1 point)** Express the number of clock cycles that ***m*** instructions (hits) take.

Each instruction that get a miss will take CPI + miss penalty = CPI + mp
m instructions will then take m x (CPI + mp) = n x mr (CPI + mp)

f) **(3 points)** Express the average CPI as a function of ***h***, ***m***, ***n***, ***mp***, and ***CPI.***

The average CPI is the total time taken by hits and misses divided by n
AvgCPI = [n x (1-mr) x CPI + n x mr (CPI + mp)]/n

g) **(2 points)** Derive from Question f) the formula ***Average CPI = CPI + mr * mp***

Simplifying by n the expression above, we get

AvgCPI = (1-mr) x CPI + mr (CPI + mp) = CPI + mr x CPI + mr x CPI + mr x mp
= CPI + mr x mp