Final Exam

CSC 252
8 May 2018
Computer Science Department
University of Rochester

Instructor: Yuhao Zhu
TAs: Alan Beadle, Sayak Chakraborti, Michael Chavrimootoo, Alan Chiu,
Akshay Desai, Benjamin Nemeth, Eric Weiss, Jie Zhou

Name:_____

Problem 0 (2 points):	_____
Problem 1 (43 points):	_____
Problem 2 (15 points):	_____
Problem 3 (30 points):	_____
Problem 4 (30 points):	_____
Problem 5 (15 points):	_____
Total (135 points):	_____

Remember "I don't know" is given 15% partial credit, but you must erase/cross everything else.

Please be sure your name is on each sheet of the exam.

Your answers to all questions must be contained in the given boxes. Use spare space to show all supporting work to earn partial credit.

You have 2 hours and 45 minutes to work (19:15 -- 22:00).

Please sign the following. I have not given nor received any unauthorized help on this exam.

Signature:_____

GOOD LUCK!!!
(And Have a Great Semester Break)

Problem 0: Warm-up (2 points)

Facebook is hiring hardware engineers. What do you think they are building?

Any answer is accepted.

Problem 1 (43 points)

Part a (5 points): A microarchitecture is predicting whether a branch is taken or not taken using a 1-bit predictor. The last five branches were: taken, taken, taken, taken, not taken. What does the branch predictor predict (choose): Taken or Not Taken?

Not Taken

Part b (5 points): In a typical Linux/Unix terminal, when you hit Ctrl + Z, which state are you putting the foreground process to (choose): Running, Stopped, Terminated?

Stopped

Part c (5 points): Cache blocking is a software-level performance optimization technique that improves what aspect of a program (choose): Locality, Parallelism, Concurrency, Security?

Locality

Part d (5 points): An application that is 90% parallelizable is executed on a single processor in 1.5 hours. If the application is allowed to run with an unlimited number of processors, what is the lower bound on its execution time?

9 mins. With an unlimited number of processors, the parallelizable part of a program would finish in no time, and the execution time is equivalent to the sequential part, which is 90 mins * 0.1 = 9 mins.

Part e (5 points): On a page fault, the operating system often loads a page from the disk into memory. How does the operating system know whether it is necessary to write the previously occupied page in the memory back to the disk? Answer in fifteen words or fewer.

Check the dirty bit in the corresponding page table entry.

Part f (5 points): What is the cycle time of a 1 GHz processor?

1 ns

Part g (5 points): What is the fundamental reason that process context switch has a much higher overhead than thread context switch in Linux? Answer in twenty words or fewer.

Threads share virtual address space while processes have separate virtual address spaces.

Part h (5 points): Suppose we have two 4-bit 2's complement numbers:

1111
1110

Does the sum of the two numbers result in an overflow?

No.

Part i (3 points): Recall that the crux of tracing-based GC algorithms such as Mark-and-sweep and Mark-sweep-compact is to start from "root" variables and then identify all the reachable variables. In the following code snippet, suppose the program just finishes executing L7, which variables are regarded as "root"? Name only those that point to variables on the heap.

```
L1:    int *p3;

L2:    int* foo(int n) {
L3:      int i, *p1;
L4:      p1 = (int *) malloc(n * sizeof(int));
L5:      for (i=0; i<n; i++)
L6:        p1[i] = i;
L7:      p3 = p1[2];
L8:      return p1;
L9:    }

L10:   void bar() {
L11:     int *p2 = foo(5);
L12:   }
```

`p1` and `p3`.

Problem 2 (15 points)

We assume that IEEE decided to add a new 8-bit representation with its main characteristics consistent with the 32/64-bit representations. Consider the following four 8-bit numbers:

A: `11100101`
B: `00111001`
C: `00001100`
D: `00011101`

The decimal values represented by the above numbers are as follows, *in no particular order*:

$$3\frac{1}{8},\ -21,\ \frac{29}{32},\ \frac{3}{8}$$

Part a (3 points): Represent decimal value $\frac{3}{8}$ in binary normalized form

1.1 x 2^-2

Part b (3 points): Which 8-bit floating point number represents -21 (choose from A, B, C, D)?

A

Part c (3 points): Which 8-bit floating point number represents $\frac{29}{32}$ (choose from A, B, C, D)?

D

Part d (6 points): Given the above information, figure out the following:
(2 points) Number of bits needed for exponent:

3

(2 points) Number of bits needed for fraction:

4

(2 points) Bias:

3

Problem 3 (30 points)

A byte-addressable, write-back cache of *fixed total size* and *fixed cache line (a.k.a., block) size* is implemented as both a direct mapped cache and also as an N-way set-associative cache. In both cases, we will assume the cache is initially empty.

First, consider the cache organized as a direct mapped cache. The following sequence of 11 accesses generates the hits/misses shown. Some miss/hit entries are intentionally left blank.

Address	Read/ Write	Direct Mapped (Hit/Miss)
0100001010	R	
1100100111	R	Miss
1110101000	R	Miss
0011000101	R	
0110111100	R	
1010110101	R	Miss
1100100000	R	Miss
0100001111	R	Hit
0101111111	W	Miss
0110110100	R	
0110100101	R	Miss

Note high order bits are the same but still miss.

Hit because of this miss

* The hit must be because of the first access because only the first address has the same high order bits as the hit address.
* Comparing the hit address and the first address, we know that offset is at least 3 bits.
* From the last two misses we that the offset is at most 4 bits.
* The miss immediately before the hit and the second miss have the same first 7 bits. So there must be an access in-between that evicts the line brought in by the second miss.
* There also could not be a miss that evicts the line brought in by the first miss for the hit to be a hit.
* All these could only be possible if offset is 3, index is 3.

Part a (4 points): How many cache lines does each set have in a direct mapped cache?

1

Part b (2 points): What is the cache line (a.k.a., block) size?

16 Bytes

Part c (2 points): What are the number of index bits for the direct mapped cache?

3

Now consider the cache organized as a N-way set-associative cache, with the same total size and same cache line size as before. The total size of "overhead" for this N-way set associative cache is 112 bits. Assume that in this particular cache, overhead in *each* cache line includes tag bits and 10 additional bits for bookkeeping (e.g., the valid bit, modified bit, LRU bits) that do not affect this problem. We have expanded the table to show the hit/misses for the same sequence of accesses when the cache is organized as an N-way set-associative cache.

Address	Read/ Write	Direct Mapped (Hit/Miss)	N-way associative (Miss/Hit)
0100001010	R	**Miss**	**Miss**
1100100111	R	Miss	Miss
1110101000	R	Miss	**Miss**
0011000101	R	**Miss**	Miss
0110111100	R	**Miss**	Miss
1010110101	R	Miss	**Miss**
1100100000	R	Miss	**Hit**
0100001111	R	Hit	**Hit**
0101111111	W	Miss	Miss
0110110100	R	**Miss**	Hit
0110100101	R	Miss	**Miss**

* T bits for tag, I bits for set index; we have (10+T) * 2^I * N = 112
* Since the cache line size is the same as before, the offset must be 4, and so we have T + I = 10 - 4 = 6.
* 10<=10+T<=16. only 14 and 16 are divisible by 112. So T is either 4 or 6.
* If T is 6, then 2^I * N = 7, so I = 0 and N = 7. If T is 4 then 2^I * N = 8, so I = 2 and N = 2 or I = 1 and N = 4.
* Since the total size is fixed and the cache line size is fixed, the total number of cache lines 2^I * N must be 8, same as before.
* So I must be 2 and N must be 2.

Part d (4 points): What is N?

2

Part e (4 points): What is the number of index bits for the N-Way set associative cache?

2

Part f (4 points): Is this a write-allocate cache?

No

Part g (10 points; 1 point per blank): Please complete the second table above by filling in "Hit" or "Miss" for each of the blank entries. "I Don't Know" is accepted on a per blank basis.

Problem 4 (30 points)

We wish to enhance the x86 ISA by adding a new instruction. The new instruction is called `STI`, "Store Indirect", and its format is:

```
STI Ra,Rb,Offset
```

The opcode of `STI` is 1010, and its binary encoding is (2-Byte long):

Opcode <4-bit>	Ra <3-bit>	Rb <3-bit>	Offset <6-bit>

`STI` operates as follows: We compute a virtual address (call it `A`) by adding the sign-extended `Offset` to the contents of register `Rb`. The memory location specified by `A` contains the virtual address `B`. We wish to store the contents of register `Ra` into the address specified by `B`.

The processor has a simple one-level virtual memory system. There is also a 2-entry TLB. You are given the following information:

- Virtual Address Space: 64 KB
- Physical Memory Size: 4 KB
- PTE Size: 2 Bytes
- The format of a PTE is shown below. The MSB is the valid bit, and the lower several bits are for the physical page number (PPN). Note that the exact number of bits for PPN is for you to determine. The rest bits are always padded with 0.

Valid <1-bit>	0...0	Physical Page Number

- `%eax`: 0x8000
- `%ebx`: 0x401E
- Program Counter (`%eip`): 0x3048

The TLB state before any instructions related to this problem are executed:

Valid	Virtual Page Number (VPN)	PTE	
		Valid	Physical Page Number (PPN)
1	0x0C1 0000 1100 0001	1	0x01A
1	0x182 0001 1000 0010	1	0x024

Part a (2 points): In this particular TLB, the Valid bit in the first column and the Valid bit in the third column are the same in both TLB entries. In general, is it possible that these two valid bits have different values?

Yes

Part b (6 points): What is binary encoding for `STI %eax,%ebx,0`? Assume that `%eax` is encoded as 0 and `%ebx` is encoded as 1.

1010 000 001 000000

Part c (4 points): To process the `STI` instruction, one must go through the Fetch, Decode, etc. instruction cycle. What is the maximum number of physical addresses that can be accessed in processing an `STI` instruction?

Hints:

1. Instruction fetch is the necessary first step in processing any instruction
2. In the one-level virtual memory system, the page table lives in the physical memory

6

This instruction accesses virtual memory three times: fetch instruction, load from address A, and store to address B. Each virtual memory access could at most lead to 2 physical memory accesses: one for accessing the PTE and the other for accessing the actual data.

Part d (18 points): Now the processor executes `STI %eax,%ebx,0`. It turned out that five physical memory accesses were needed. The table below shows the Virtual Address (VA), Physical Address (PA), Data, and whether or not there was a TLB hit for each of these five physical memory accesses in the order they occurred. Some of the blanks are intentionally left for you to fill in.

	Virtual Address	**Physical Address**	**Data**	**TLB Hit?**
Fetch Inst.	0x3048 PC	0x488	0xA040	Yes
Get PTE for A	**N/A**	**0x660**	**0x8040**	No
Load from A	0x401E %ebx (A)	0x81E	**0x40FE** B	No
Get PTE for B	**N/A**	0x66E	0x800E	No
Write to B	0x40FE	**0x1DE**	0x8000 %eax	No

Complete the table and fill in the following three boxes. You can assume that no page faults occurred.

Hints:

1. What does the first TLB hit mean?
2. Recall how to use PTBR and VPN to access the page table.
3. Use the first and last accesses to figure out the page size first. Everything else will follow.

(3 points): What is the page size?

32 Bytes

(3 points): What is the total number of physical pages?

128

(3 points): What is the data in the page table base register (PTBR)?

0x260

(9 points; 1 point per blank): Please complete the table above. "I Don't Know" is accepted on a per blank basis.

* The first access is a TLB hit. Analyzing the VA and the two TLB entires, you would know that the page offset must be either 5 or 6.

```
0x0C1:  0000 1100 0001
0x1DE:  0001 1101 1110
0x40FE: 0100 0000 1111 1110
```

* The second access must be to get the PTE of virtual address A, which returns 0x8040 as the PTE. Using the PTE, we form the physical address, which the third access uses to get the data 0x40FE, which is essentially B.
* The fourth access must be then to get the PTE of virtual address B and form its physical address, which is 0x1DE. The fifth access must be to store %eax to 0x1DE (corresponding to virtual address B). So its VA must be 0x40FE, same as the data returned from the third access. The data in the fifth access is 0x8000 (i.e., %eax).
* Focus on the last access. We know its VA is 0x40FE and the PA is 0x1DE. Their page offsets must match. Analyzing the two address, we know the page offset must be 5 bits rather than 6.

```
0x1DE:       0001 1101 1110
0x40FE: 0100 0000 1111 1110
```

* If the page offset is 5 bits, we could easily get that the physical page number is 12-5=7 bits since the total physical memory size is 4K = 2^12.
* Given the 5 bits page offset, we could also easily get know that the TLB hit in the first access hits on the second TLB entry, which allows us to form the physical addresses of the first and the third access, which are 0x480 and 0x81E, respectively.
* To get the PTBR value, focus on the second physical address, which is calculated by PTBR + VPN * 2. Since we know the page offset is 5, we would know that VPN in that access is 0x200.

Problem 5 (15 points)

A programmer writes the following two C code segments. She wants to run them concurrently on a multicore processor, called SC, using two different threads, each of which will run on a different core.

Thread `T1`

```
a = X[0];
b = a + Y[0];
while(*flag == 0);
Y[0] += 1;
```

Infinite loop until flag becomes 1, at which point Y[0] must be 1. So in the end Y[0] could only be 2.

Thread `T2`

```
Y[0] = 1;
*flag = 1;
X[1] *= 2;
a = 0;
```

`X`, `Y`, and `flag` have been allocated in main memory, while `a` and `b` are contained in the processor registers. A read or write to any of these variables generates a single memory request. The initial values of all memory locations and variables are 0. Assume each line of the C code segment of each thread translates to a single machine instruction.

Part a (5 points): Both threads have a variable `a`. Are they referring to the same variable?

No

Part b (5 points): What are the possible final value(s) of `Y[0]` after both threads finish execution? Consider all the possible thread interleavings.

2

Part c (5 points): What are the possible final value(s) of `b` after both threads finish execution? Consider all the possible thread interleavings.

0 and 1