Department of Electrical and Computer Engineering
University of Texas at Austin

EE460N Fall 2020
Y. N. Patt, Instructor
Chester Cai, Sean Stephens, Arjun Ramesh, TAs
Exam 2
November 18th, 2020

Name:	Solution
EID:	

Problem 1: 25 points
Problem 2: 15 points
Problem 3: 30 points
Problem 4: 30 points
Total: 100 points

Note: Please be sure that your answers to all questions (and all supporting work that is required) are contained in the space provided.

Please read the following sentence, and if you agree, sign/print your name where requested: I have not given or received any unauthorized help on this exam.

Signature:	Solution

Good Luck!

General Instructions:

1. You are free to use anything in the Handouts section of the course website that is listed under "Course Related Handouts" or "LC-3b Handouts." In particular, Appendix A and Appendix C may be of use. Anything other than that from the course website, textbooks, or the Internet is not allowed and considered unauthorized access.
2. Use of a calculator is not required but is permitted.
3. If you have any questions, join the class Zoom link and ask a TA. You do not need to stay on the Zoom call during the exam unless you have questions.
4. Announcements will be posted here. Check this page periodically throughout the exam.
5. You may take the exam by printing it, editing a PDF, or editing a Google Doc. Read the instructions for your preferred method below.
6. **You are required to stop working on the exam promptly at 6:30 PM.**

Printing or editing a PDF:

1. Download and save the PDF.
2. Edit the PDF to fill in answers with a software of your choice. Feel free to show your work in the available space. You may also choose to print the exam and solve it on paper.
3. When you are ready to submit your exam, save the edited PDF as "Exam 2 <your name>"; if you printed your exam, scan in your written answers as a PDF with the same name. You may use a scanner or an app such as CamScanner.
4. Upload the PDF to Gradescope by 6:40 PM. The entry code for Gradescope is **9RPGX3**.

Editing a Google Doc:

1. Save a copy of the document to your Google Drive.
2. While working on the exam, **DO NOT expand any boxes that are given to you.** Feel free to show your work in the available space. If you need more space, you are writing too much.
3. When you are ready to submit your exam, click "File"-> "Print" and select "Save as PDF". Save the edited PDF as "Exam 2 <your name>".
4. Upload the PDF to Gradescope by 6:40 PM. The entry code for Gradescope is **9RPGX3**.

Problem 1 (25 points): If you leave the box empty, you will receive one point.

Part a (5 points): Recall the VAX virtual memory consists of 4GB of virtual address space: 2GB of user space, 1GB of System Space, and 1GB reserved for future use.
True or False: A downside of VAX virtual memory is that the unused "Reserved" region wastes frames in physical memory. Explain your true/false answer in 15 words or fewer.

False, the reserved region is in virtual memory, which does not use frames unless allocated.

Part b (5 points): True or False: In VAX virtual memory, the values of the Process Base Register (P0BR or PBR) and the System Base Register (SBR) point to the starting locations of the Process Region and System Region, respectively. Explain your true/false answer in 15 words or fewer.

False, they point to the process page table and the system page table respectively

Part c (5 points): In the asynchronous I/O system discussed in class, if device controller X receives a BG but does not want the bus, it passes BG on (in a daisy chain fashion) to the next controller operating at the same priority level.
True or False: Device controller X stops asserting BG when it receives the SACK signal from the device controller that accepted BG. Explain in 20 words or fewer.

False, the device controller stops asserting BGout when its BGin signal is no longer asserted, otherwise there is a race condition.

Part d (5 points): Most ISAs have, in addition to N and Z, a C condition code which stores the carry resulting from a 2's complement integer ADD instruction. Many ISAs have an opcode ADC that adds two register operands plus the contents of this Carry condition code in a single instruction. Why is it helpful? Be specific, but please limit your response to not more than 20 words.

This is helpful for implementing long integer adds.

Part e (5 points): True or False: In order to satisfy a page fault, when a virtual page is evicted from a frame of physical memory, the evicted page must be written back to the disk. Explain your true/false answer in 15 words or fewer.

False, only need to write to the disk if the evicted page is dirty

Problem 2 (15 points): A byte-addressable machine has a 64KB physical memory with a 32 bit data bus. The physical memory has the following parameters:

- 1 channel
- 8 ranks
- 4 chips per rank
- 2 banks
- 16 columns per row

Part a (6 points): Calculate the number of row bits.

64 KB -> 16 bits
16 - 3(rank) - 1(bank) - 4(column) - 2(BoB) = 6 row bits

Part b (9 points): The CPU issues the following six one-byte memory accesses in some order.

```
0xB2AD  (0b 1011 0010 1010 1101)

0x36AF  (0b 0011 0110 1010 1111)

0x96AC  (0b 1001 0110 1010 1100)

0xB4AE  (0b 1011 0100 1010 1110)

0xB6A8  (0b 1011 0110 1010 1000)

0xB68F  (0b 1011 0110 1000 1111)
```

Specify which of the 16 address bits are row bits, which are column bits, etc. in the table, so that regardless of the order, the six accesses take the **most amount of time** to complete. Note: there are multiple correct answers to this problem. Your job: specify one of them.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Row		Row			Row	Row				Row			Row	BoB	BoB

What we want is for all of them to go to the same bank, same rank(so no interleaving), and different rows(no row buffer hits). The column bits do not matter at all. The rest of the bits does not matter as long as the number matches

Problem 3 (30 points) Computing the dot product of two vectors is a very common operation in many applications. The following code computes the dot product of vectors A and B, where A and B are stored as one dimensional arrays in sequential locations of memory.

```
int result = 0;
for(int i = 0; i < N; i++) {
      result += VectorA[i] * VectorB[i];
}
```

If the vector is sparse (i.e., most of the vector elements are zero), it can be represented in memory by two one-dimensional arrays, a value array and an index array. The value array contains the values of each of the non-zero elements and the index array entries contain the indexes of the corresponding entries of the value array. The number of elements in each array is the number of non-zero elements in the original vector. We call this representation a compressed sparse row (CSR).

For example, the CSR representation of the vector 0, 0, 1, 5, 0, 0, 2 is

Index Array:	2	3	6
Value Array:	1	5	2

The following code computes the dot product of vectors A and B, represented in CSR form.

```
int result = 0;
for(int i = 0; i < NumOfNonZeroInA; i++) {
      for(int j = 0; j < NumOfNonZeroInB; j++) {
            if(IndexA[i] == IndexB[j])
                  result += ValueA[i] * ValueB[j];
      }
} // NOTE: this is not an efficient way to compute the dot product
  // of 2 vectors in CSR form, but it is a simple implementation to
  // make the exam easier.
```

Your job: Compare the data cache performance obtained by executing the two algorithms above on the corresponding representations of vectors A and B.

The data cache is 32KB, 2-way set associative, 16 byte line size, LRU replacement. The two vectors each consist of 256K 32-bit integers, of which only 1024 are non-zero. The index for each element in the CSR arrays is represented by a 32 bit integer.

Case 1: Representing each vector as a one-dimensional array in sequential locations of memory.
Part a (3 points): Array A starts at memory location x8000 0000, Array B starts at memory location xC000 0000. What is the cache hit ratio? Assume the loop variable and result are stored in registers.

3/4. This is a streaming pattern, so you get a miss when you bring the line in, then 3 hits for the rest of the line.

Part b (3 points): What is the cache hit ratio if the cache is direct mapped with the same cache size?

0. A and B map to the same sets. So each of them would kick the other one out.

Case 2 (24 points): Representing each vector in CSR form.
Part c: To simplify the problem, assume the if statement evaluates true on average 1 out of 2048 times, and the accesses to valueA and valueB are always cache misses. Below is the starting location of each array.

Array Name	**Starting Address**
IndexA	x8000 0000
IndexB	x8000 1000
ValueA	x8000 2000
ValueB	x8000 3000

What is the total number of accesses and cache misses? Show your work

Everything fits in the cache
Accesses: 1k * 1k * 2 (accesses on Index A and B) + 1k * 1k * 1/2048 *2 (misses on Value A and B)= 2M + 1K
Misses: 1024 / 4 * 2 (misses on index A and B) + 1k * 1k * 1/2048 * 2(misses on Value A and B) = 512 + 1024

Part d: Would the hit ratio a) stay the same, b) decrease by a little, c) decrease significantly if the cache is only 4KB while the associativity and line-size stay the same? Explain in 20 words or fewer.

Decrease a little. Only one of the arrays fits in the cache. On each iteration of the outer loop, only 1 line of indexA needs to be in the cache to get the hits. Most of the indexB will stay in the cache between the outer iterations.

Part e: Would the hit ratio a) stay the same, b) decrease by a little, c) decrease significantly if the cache is only 1KB while the associativity and line-size stay the same? Explain in 20 words or fewer.

Decrease significantly. The cache can only fit ¼ of one of the arrays. Now even indexB won't fit in the cache, so won't be able to get the reuse pattern from indexB anymore between iterations

Problem 4 (30 points) Suppose the LC-3b ISA has a 13-bit byte-addressable virtual address space with two levels of virtual-to-physical address translation, similar to the VAX. The virtual address space is divided evenly into two regions. The first half is system space, and the second half is user space. The virtual address consists of 1 region bit, a 4-bit VPN, and an 8-bit page offset. Each PTE is 2 bytes.

The microarchitecture includes a single TLB which is used to translate both system and user virtual addresses. The interrupt/exception table is stored in virtual memory, with a base address of 0x0A00. The system stack pointer is initialized to 0x1000 and the stack grows towards lower addresses.

TLB misses can be handled in hardware or software. In this problem we handle it in software with an exception handler that is executed when a TLB miss occurs. The exception handler takes the address that caused the exception, finds the corresponding PTE, and loads it into the TLB. The exception vector for TLB misses is 0x05. Assume the TLB has infinite capacity.

Upon detecting a TLB miss exception, the hardware performs the following operations:
(**Note:** This is different from what you implemented in Lab 4.)

- Saves the PSR and PC on the system stack
- Sets the privilege mode to supervisor level privilege
- Saves registers R0 through R7 on the system stack
- Sets R6 to point to the system stack if that is not already the case
- Sets R0 to the faulting address (i.e. the address that caused the TLB miss)
- Sets R4 and R5 to P0BR and SBR, respectively

The RTI instruction does the following:

- Restores registers R0 through R7 from the system stack
- Restores PC and the PSR from the system stack

Two new instructions are added to the ISA, User_TLB_Load and System_TLB_Load, which load one user PTE and one system PTE into the TLB, respectively. User_TLB_Load takes a register that contains the VA of the PTE as an input. System_TLB_Load takes a register that contains the PA of the PTE as an input.

Part a (5 points): What location in the virtual address space needs to be filled with the starting address (0x0E00) of the TLB miss exception handler in order for it to be executed during a TLB miss? 0x0A00 + 0x05(exception vector) * 2 (size of each entry in the exception vector table)

0x0A0A

Part b (15 points): Your job: Complete the TLB-miss exception handler. Put each instruction you write into a separate box. We have provided space for more instructions than you need. Use as many as you need. Enter "NOP" in the boxes that you do not use. Note in this question, the handler deals only with one level. If another level of translation is needed, then a nested exception is triggered during the execution of User_TLB_Load.

```
        .ORIG 0x0E00
        ;R0 = Faulting Address, R4 = P0BR,
        ;R5 = SBR, R6 = System Stack Pointer
```

RSHFL R1, R0, #8

```
        AND R2, R1, x10
        BRz LABEL
```

AND R1, R1, 0x0F
LSHFT R1, R1, #1
ADD R1, R4, R1
User_TLB_Load R1

```
        RTI
```

```
LABEL
```

LSHFT R1, R1, #1
ADD R1, R5, R1
System_TLB_Load R1

```
        RTI
```

Part c (10 points): During the execution of the first instruction, if the TLB is empty, the machine would trigger an infinite number of TLB miss exceptions. In order to fix the problem above, some PTEs will need to be present in the TLB at all times. What pages do those PTEs correspond to?

Page A (exception vector table), Page E (exception handler), Page F (system stack)
Otherwise, whenever you start to translate something, trigger an exception, try to access the stack, trigger another exception, try to access the stack, trigger another exception...