Department of Electrical and Computer Engineering
University of Texas at Austin

EE460N Spring 2021
Y. N. Patt, Instructor
Siavash Zangeneh, Ben Lin, Juan Paez, TAs
Exam 2
April 21st, 2021

Name:	Solutions
EID:	Extra Explanations in Green

Problem 1: 20 points
Problem 2: 10 points
Problem 3: 20 points
Problem 4: 25 points
Problem 5: 25 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:	

Good Luck!

General Instructions:

1. You are free to use anything in the Handouts section of the course website that is listed under "Powerpoint Presentations", "Other Handouts", "LC-3b Handouts", and "Spring 2021 Exam 2 Buzzwords." In particular, Appendix A and Appendix C may be of use. Anything else from the course website is not allowed. Anything from any textbooks or the Internet is also 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. Do not 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 will take the exam by editing a Google Doc. Unlike exam 1, we will not accept handwritten answers by either printing the exam or editing a pdf using a tablet.
6. Read the instructions below.
7. **You are required to stop working on the exam promptly at 6:30 PM.**

Editing a Google Doc:

1. Open the Google Doc version of the exam from here.
2. Save a copy of the document to your Google Drive. Click "File"-> "Make a copy."
3. 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.
4. When you are ready to submit your exam, click "File"-> "Print" and select "Save as PDF". Save the edited PDF as "Exam 1 <your name>".
5. Upload the PDF to Gradescope **by 6:35 PM.**
6. If you fail to upload your exam to Gradescope on time, email your exam to the TAs as soon as possible. Late penalties may apply.

Problem 1 (20 points): Answer each question in 20 words or fewer. Note: For each of the four answers below, if you leave the box empty, you will receive one point of the five.

Part a (5 points): A program executes instructions 1,2,3,4,5,6,7,8 out-of-order. They produce their results in the cycles shown in the table below.

Instruction	Cycle
1	05
2	07
3	16
4	11
5	14
6	15
7	20
8	23

The microarchitecture allows up to 5 instructions to retire each cycle. If we insist on maintaining precise exceptions, when can instruction 5 retire? Explain.

Cycle 16. To maintain program order, Instruction 5 needs to wait for Instructions 1-4 to execute and retire first. Since up to 5 instructions can retire at the same time, 5 can retire with 3 and 4 in cycle 16. If you answered "after cycle 16," I gave you full credit even though the hardware can do it in cycle 16 along with instructions 3 and 4.

Part b (5 points): Every PTE contains an M bit. What information is contained in the M bit, and why is it useful?

It means that the data in the page is modified. It is useful because if the data is not modified, the page does not have to be written back to disk if it needs to be paged out.

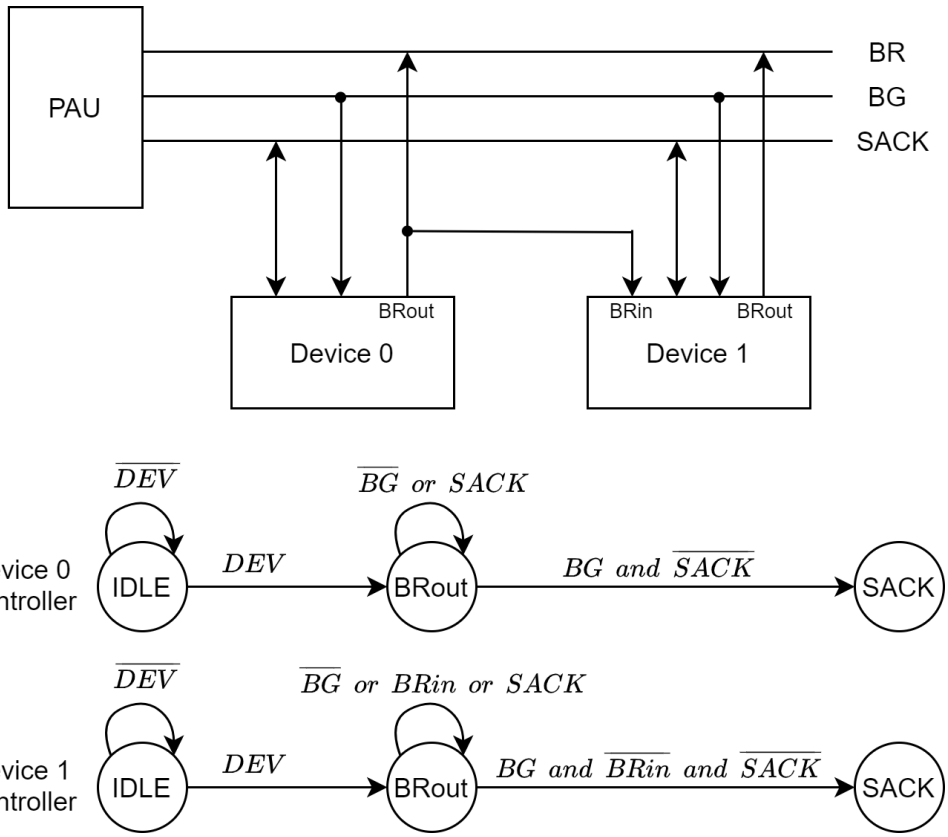
Part c (5 points): In the asynchronous I/O system discussed in class, multiple devices requested the bus at the same priority. The PAU granted the bus in a daisy chain manner. Does that say anything about the relative priorities of the devices that requested the bus all at the identical priority level? Explain.

The device controllers closer to the PAU in the daisy chain have a higher relative priority since they will see BG first; i.e., each controller has a higher priority than all controllers further away from the PAU.

Part d (5 points): A user program generated a virtual address that was on page x83 of process space. The process page table consists of x81 PTEs. The hardware's job is to determine the frame of physical memory containing virtual page x83. In this case, what does the hardware do? Explain.

x81 PTEs means process space consists of x81 pages, starting with page x00.
The VA having a page x83 means that the address is not part of process space.
Therefore, the hardware would take an exception since the process is trying to access an unmapped address.

Problem 2 (10 points): A student decided to change the asynchronous bus arbitration protocol that we showed in class. Instead of daisy chaining the devices on the same priority, the student tried to solve the problem by letting Device 1 read the BR signal produced by Device 0. Only the relevant states in the state machine are shown.



Unfortunately, this solution does not work because of a race condition.

Part a (3 points): What bad event will be caused as the result of this race condition?

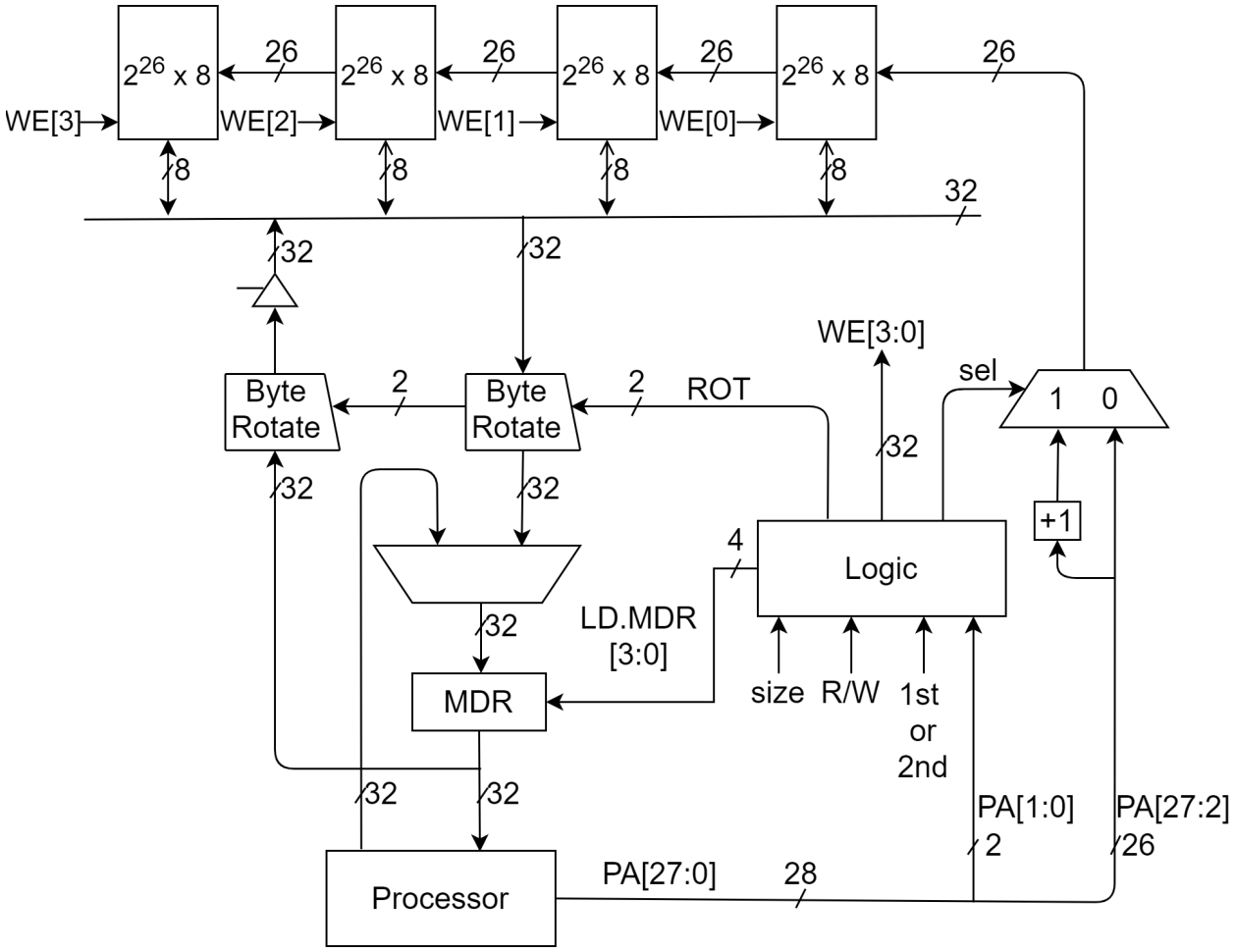
Both devices will go into the SACK state and try to use the bus in the next bus cycle.

Part b (7 points): What events must happen and in which order to cause the race condition? Please be precise but not wordy.

At the start of arbitration, Device 1 is the only device that wants the bus, goes into the BRout state, receives the BG and moves into the SACK state. Device 0 decides that it wants the bus after Device 1 asserts SACK but before the time it takes for SACK to propagate to Device 0. So, Device 0 sees BG and NOT(SACK) and also moves into the SACK state.

Problem 3 (20 points)

We have the following memory, similar to the one in Problem Set 3, which supports unaligned accesses. The ISA contains `LDByte`, `LDWord`, `STByte` , and `STWord` instructions, where a word is 32 bits.



Both byte rotators in the figure are right rotators.

Assume we have an array, `products_array`, with 2048 elements, where each element is a 5 byte struct of type `Product`, made up of a 4 byte field `PRICE`, and a 1 byte field `ON_SALE`, as shown below:

```
typedef struct Product_Struct {
    int  PRICE;                /* 4 bytes */
    char ON_SALE;              /* 1 byte  */
} Product;
Product products_array[2048];  /* begins at physical address 0x0  */
```

The array `products_array` begins at physical address 0x0.

We wish to read in the entire `products_array` array from memory. For each of the 2048 array elements, we will perform a `LDWord` on the 4 byte field `PRICE`, and a `LDByte` on the 1 byte field `ON_SALE`.

Part a (6 points):

Any 4-byte word which is not aligned (i.e., low two bits of the physical address are not 00) will span multiple chip addresses and require a 2nd access.

Since each `Product` struct is 5 bytes:

- `products_array[0].PRICE` will be at address 0x0, which IS 4-byte aligned (low 2 bits of the physical address are 00)
- `products_array[1].PRICE` will be at address 0x5, which IS NOT 4-byte aligned (low 2 bits of the physical address are 01)
- `products_array[2].PRICE` will be at address 0xA, which IS NOT 4-byte aligned (low 2 bits of the physical address are 10)
- `products_array[3].PRICE` will be at address 0xF, which IS NOT 4-byte aligned (low 2 bits of the physical address are 11)
- `products_array[4].PRICE` will be at address 0x14, which IS 4-byte aligned (low 2 bits of the physical address are 00)

The pattern continues. So for every 4 elements of `products_array`:

- 1 will have its `PRICE` field be aligned, while
- the other 3 will not

Hence ¾ of the 2048 `LDWord` operations will require a 2nd access.

1. Of the 2048 `LDWord` operations, how many will require a 2nd access?

   (¾) * 2048 = 1536

Any byte access whose low two bits of address are 10 will require a right rotate value of 2.

For every 4 elements of `products_array`:

- 1 will have its `ON_SALE` field be at an address where the low two bits of the physical address are 10, while
- the other 3 will not (their low two bits of the address will be 00, 01, and 11)

Hence ¼ of the 2048 `LDByte` operations will require a right rotate value of 2

2. Of the 2048 `LDByte` operations, how many will require a right rotate value of 2?

   (¼) * 2048 = 512

Part b (4 points):
Instead of an array of structs, suppose we stored the same information in two separate arrays `PRICE_ARRAY` and `ON_SALE_ARRAY`, as shown below:

```
int  PRICE_ARRAY[2048];              /* 4 bytes each element, 8192 bytes total*/
char ON_SALE_ARRAY[2048];            /* 1 byte each element,  2048 bytes total*/
```

The two arrays are stored in contiguous physical memory, with the array `PRICE_ARRAY` located at physical address 0x0. We read in both arrays, performing a `LDWord` on each of the 2048 elements of `PRICE_ARRAY`, and a `LDByte` on each element of `ON_SALE_ARRAY`.

Because now we store all the 4 byte words together in the same array, and the start of the array is aligned at address 0x0, all the 4 byte words will be aligned, and none of them will require a 2nd access.

1. Of the 2048 `LDWord` operations, how many will require a 2nd access?

0

For every 4 elements of `ON_SALE_ARRAY`:

- 1 will have have an address where the low two bits of the physical address are 10, while
- the other 3 will not (their low two bits of the address will be 00, 01, and 11)

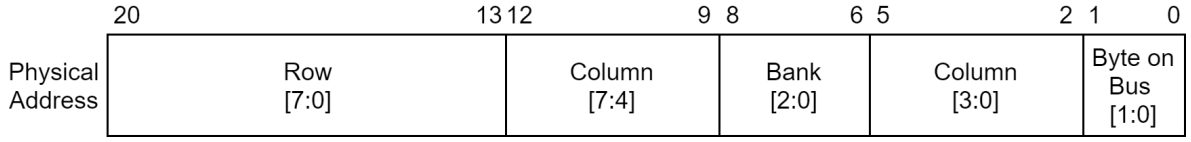
Hence ¼ of the 2048 `LDByte` operations will require a right rotate value of 2, as before

2. Of the 2048 `LDByte` operations, how many will require a right rotate value of 2?

(¼) * 2048 = 512

Part c (10 points):

Suppose the array `PRICE_ARRAY` from part b is stored in DRAM memory which is organized as follows:



Note the 8 bits of column address are split into two parts, with the high 4 bits coming from bits [12:9] of the physical address, and the low 4 bits coming from bits [5:2] of the physical address.

Also note that this physical memory setup is completely independent from that of part a and b.

The array `PRICE_ARRAY` begins at physical address 0x0. We access all 2048 elements of the array in sequential order. **Assume all row buffers are initially empty**.

The `PRICE_ARRAY` array is 8KB and begins at address 0x0. Since the row address bits only begin at bit 13 of the physical address, we will not toggle any of the row address bits while accessing `PRICE_ARRAY`. This means the only possible source of row buffer misses come from the initial access to each bank, since the problem assumes row buffers are initially empty.
Hence the last row buffer miss will be the first access to the last bank, bank 7. This will be at address:
00000000 0000 111 0000 00
Which in hex is address x1C0

1. What is the address of the last array element that causes a row buffer miss?

   x1C0

There are 2 byte on bus bits. So the width of the bus is 2^2 = 4

2. What is the width of the bus in bytes?

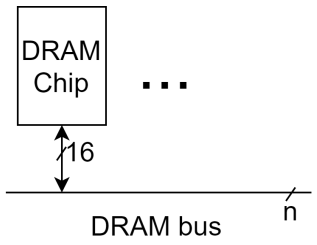
   2^2 = 4

There are 8 column address bits total, meaning there are 2^8 = 256 columns per row per bank. However, each column refers to an entire bus width's worth of data (i.e., 4 bytes). Hence the size of the row buffer is the product of the two:

3. What is the size of the row buffer (of a bank across all chips) in bytes?

   (2^8) * 4 = 1024B (1KB)

4. If we used DRAM chips with 16 bit data interfaces, as shown in the figure below:



The width of the bus is 32 bits (4 bytes). Since each DRAM chip can provide 16 bits of data, we need two DRAM chips to provide a full bus width's worth of data.

How many DRAM chips in total are there in the system?

2

There are two ways to approach this problem:

1. Since there are two DRAM chips in total in the system, the capacity per chip is simply the capacity of the entire physical memory divided by two. Since there are 21 physical address bits ([20:0]), the entire physical memory is 2MB, meaning each DRAM chip is 1MB (1048576 B)
2. In total, our memory has
    - 2^3 = 8 banks (since there are 3 bits of bank address)
    - 2^8 = 256 rows (since there are 8 bits of row address)
    - 2^8 = 256 columns (since there are 8 bits of column address)

   Each specific (bank, row, column) address tuple refers to a full bus width's worth of data, which is 4 bytes (32 bits). However, of the 4 bytes, 2 of them come from the first DRAM chip, and 2 of them come from the second DRAM chip.

   That is to say, given a (bank, row, column) address tuple, each DRAM chip will provide 2 bytes of data. This makes sense, because we said the DRAM chips have 16 bit (2 byte) data interfaces.

   Hence the capacity per DRAM chip is:

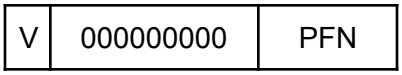
   8 banks * 256 (rows/bank) * 256 (columns/row) * 2 bytes = 1048576 B = 1MB

   What is the capacity per DRAM chip in bytes?

1048576 B (1MB)

Problem 4 (25 points): We wish to enhance the LC-3b ISA by adding virtual memory support. Virtual memory will be implemented by a VAX-like scheme as we studied in class. The 16-bit addresses you are familiar with are virtual addresses. Physical memory is 16KB. Page size is 256 bytes.

The memory management system uses the two-level page table scheme. Virtual memory is partitioned into two halves. User space starts at x0000, System space starts at x8000. The TLB contains 8 entries and is fully-associative. The TLB is only used for storing user process PTEs. A PTE is 16 bits. For purposes of this question only, we will assume the PTE has the following form:



We wish to execute the following two instructions sequentially in a program fragment:

At address x3000: LDW R0, R1, #0 (instruction encoding: x6040)
At address x3002: STW R0, R2, #0 (instruction encoding: x7080)

Before the execution of the two instructions, R1 = x4000, R2 = x5002, and the TLB is initially empty. After the execution of the two instructions, R0 = xA000.

The execution of the two instructions leads to the following physical address accesses in an unspecified order. This means that the order of the list of physical addresses is not necessarily the order in which they are accessed. Note that address x0240 is accessed three times.

Physical addresses: x0240, x0240, x0240, x0360, x0380, x03A0, x0400, x0402, x0502, x0600

Assume SBR points to the starting address of a frame.

Part a (3 points): How many TLB hits do we observe?

1

The program fragment will access 4 virtual addresses in the user space: 2 instruction fetches, and 2 data accesses. All we have to do is to look at the page numbers of the addresses and see which pages we access more than once. Note that system page accesses do not matter since the question states that we use the TLB for storing only user pages.

x3000: page number = x30 (first time access -> TLB miss)
x4000: page number = x40 (first time access -> TLB miss)
x3005: page number = x30 (second time access -> TLB hit)
x5002: page number = x50 (first time access -> TLB miss)

Part b (16 points): Specify the 2-byte word at each physical address after the two instructions are executed.

Address	x0240	x0360	x0380	x03A0	x0400	x0402	x0502	x0600
Content	x8003	x8004	x8006	x8005	x6040	x7080	xA000	xA000

Let's first list all translation steps and the accesses to physical memory:
---- 1st, 2nd, and 4th user access ---------
Since they're TLB misses, they each need 3 physical accesses: 1) system page PTE access, 2) user page PTE access, 3) the data access.
---- 2nd user access ---------
Since it's a TLB hit , it's just one 1) the data access.

Since access x0240 is repeated 3 times, x0240 must be the system page PTE address that happens to be the same for all the three translations. Note that it cannot be the user page PTE because the three translations correspond to three different user pages, so the address of their PTEs would be different.

Since the user page numbers are x30, x40, and x50, the virtual addresses for the user page PTEs are: PBR + x60, PBR + x80, PBR + xA0. The only physical addresses with the same distance interval are x0360, x0380, and x03A0, so they should correspond to user page PTEs.

Now that we have accounted for the PTE accesses, x0400, x0402, x0502, x0600 must be the remaining data accesses. Virtual addresses x3000, and x3002 have the same page number, therefore they must also have the same frame number, so they correspond to physical addresses x0400, and x0402 respectively. By matching their offsets, we know that virtual address x4000 corresponds to physical address x600, and virtual address x5002 corresponds to physical address x502.

To determine the contents of the physical addresses corresponding to PTEs, we add the PFN that they correspond to x8000 (for the valid bit).
Content of x0240 = x8000 + x03 (frame number of x0360, x0380, and x03A0) = x8003
Content of x0360 = x8000 + x04 (frame number of x0400, and x0402) = x8004
Content of x0380 = x8000 + x06 (frame number of x0600) = x8006
Content of x03A0 = x8000 + x05 (frame number of x0502) = x8005

Content of x400 = x6040 (instruction at VA = x3000)
Content of x402 = x7080 (instruction at VA = x3002)
Content of x502 = Content of x600 = xA000 (value of R0 at the execution of the two instructions)

Part c (3 points): What is the value of the System Base Register (SBR)?

x0200

The size of the system page table = (number of system pages) * (PTE size) = (2^7) * 2 = 256B.

Since the question states that SBR points to the start of a frame, the only valid SBR within 256B distance of x0240 is x0200, so SBR = x0200.

Part d (3 points): What is the value of the User Process Base Register (PBR)?

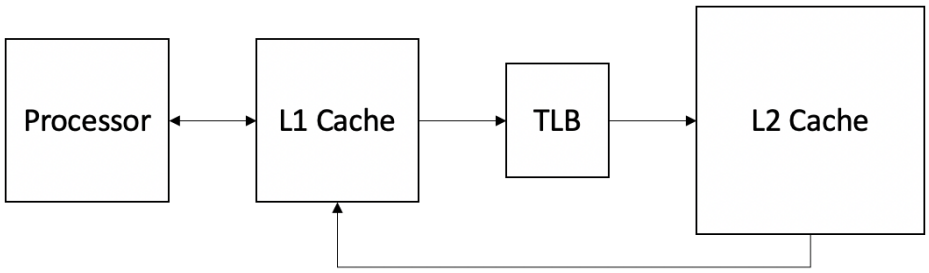
xA000

x0240 = SBR + 2 * Page number of (user page PTE)
Therefore,
Page number of (user page PTE) = (x0240 - SBR)/2 = (x0240 -x0200)/2 = x20.

Therefore, the virtual addresses corresponding to PA x0360, x0380, and x03A0 are xA060, xA080, and xA0A0 (the offset bits in VAs and PAs are the same).

As discussed earlier, user page PTE addresses are PBR + x60, PBR + x80, PBR + x0A0,
therefore PBR = xA000.

Problem 5 (25 points):
Consider the following two-level cache memory system.



We know the following:

- The memory is byte addressable.
- A virtual address is 16 bits, and a physical address is 14 bits.
- The size of a page is 256 bytes.
- The L1 cache has a capacity of 1 KB and is virtually indexed, virtually tagged.
- The L2 cache has a capacity of 4 KB and is physically indexed, physically tagged.
- The L1 and L2 caches have the same cache block size and the same number of sets.
- The L2 is inclusive of L1
- Both caches begin with a cold start.
- The TLB hit rate is 100%.

The processor is executing part of a program that needs to read five values from memory in the sequence shown below. Note that six cells in the L2 cache sequence table are left blank for you to fill in.

L1 Cache Access Sequence

Address	**Hit/Miss**
x3020	Miss
x3040	Miss
x3000	Hit
x3440	Miss
x3050	Miss

L2 Cache Access Sequence

Address	**Hit/Miss**
x0120	Miss
x0140	Miss
x0940	Miss
x0150	Hit

Part a (12 points):

1. What is the size of a cache block?

   64 Bytes

   Accesses 1,2,3 are miss, miss, hit respectively. Since access 3 is a hit, it must be on the same block as access 1 (if it was on the same block as 2, then they would all be on the same block because 3020 is between 3000 and 3040, and access 2 would have hit). Since access 1 and 3 differ only by bit 5 and access 2 and 3 differ only by bit 6, bits [0:5] represent the offset. Thus, a cache block is $2^6 = 64$ bytes.

2. What is the associativity of the L1 cache?

   Direct Mapped

   Access 5 has the same high 10 bits as access 2, meaning it is in the same cache block. However, it misses. This must be because access 4 kicked out the block.

3. What is the associativity of the L2 cache?

   4-Way

   We know the L1 and L2 caches have the same number of index bits, but the L2 cache as 4x the capacity. Thus, it must have 4 ways per set.

4. How many bits are required for tag, index, and offset?

Cache	# Tag Bits	# Index Bits	# Offset Bits
L1	6	4	6
L2	4	4	6

   We already found the offset bits, which applies to both L1 and L2 since they have the same cache block size. The L1 cache is 1 KB (2^{10}), and a block is 2^6. Thus, there are 2^4 sets, meaning that we have 4 index bits. The tag bits can be calculated from the difference.

5. Is it possible to access the TLB in parallel with the L2 cache? Why or why not?

   Not possible. The index bits of the physical address [9:6] do not fit inside the offset on the page [7:0]. Thus, we must perform a TLB lookup first.

Part b (5 points): Fill out the six empty cells in the L2 Cache Access Sequence table.

Address Explanation:

A page is 256 bytes, or 8 bits. Since x30 translates to x01, we know the other two are x0140 and x0150 by just adding the offset.

Hit/Miss Explanation:

The first miss is compulsory due to the cold start.

We know that the second access is in a different cache block than the first (else, the second access would have hit in L1). Thus, L2 misses here also.

Same reasoning as 2.

We know that L2 is 4-way set associative. Even though accesses 2 and 3 map to the same index, access 4 will be a hit because it is in the same block as access 2.

Part c (8 points):

Now consider a **different** two-level cache system of a byte-addressable memory. In this system:

- The L1 cache is 1 KB and is 2-way set associative.
- The L2 cache is 2 KB and is 4-way set associative.
- Both caches are physically indexed, physically tagged; physical addresses are still 14 bits.
- The replacement policy is LRU.
- Both caches have 16-byte blocks.
- The L2 is inclusive of L1

We would like to execute the following C-code snippet. You may assume that the caches are empty before the program's execution, and that the values of integers `i` and `x` are always stored in a register. `A`, `B`, and `C` are arrays of 32-bit integers.

```
for (int i = 0; i < 128; i ++) {
    x += A[i]*B[i] + C[i];
}
```

In each iteration of the loop, the following operations occur in order: A[i] is read, then B[i], then C[i], then x and i are updated. Array `A` begins at physical address x0400, `B` begins at physical address x0800, and `C` begins at physical address x0C00.

For the questions below, consider only data accesses (i.e., ignore instruction accesses).

1. What is the L1 cache hit ratio when executing this snippet?

   0

   This is a constant pattern of miss A → miss B → miss C (kick A out) → miss A (kick B out) → miss B (kick C out) …

2. What is the L2 cache hit ratio when executing this snippet?

   3/4

   The higher associativity allows us to store A, B, and C without conflict. At the start of each block, we miss A, B, and C once, and then hit them each 3 times after that (since a block can fit 4 integers).

3. What is the L1 cache hit ratio if we execute this snippet again, immediately after executing it the first time, with a warm cache?

   0

   A full cache does not help, since the pattern will continue.

4. What is the L2 cache hit ratio if we execute this snippet again, immediately after executing it the first time, with a warm cache?

   1

   The full cache with the higher associativity means that the entirety of A, B, and C will be in the cache.