Problem Set 4 Solutions

Department of Electrical and Computer Engineering
The University of Texas at Austin
EE 382N.1, Fall 2018
Instructor: Dam Sunwoo
TA: Pritesh Chhajed

Questions

Problem 1
We have been referring to the LC-3b memory as 2^16 bytes of memory, byte-addressable. This is the memory that the user sees, and may bear no relationship to the actual physical memory.

A. Suppose that the actual physical address space is 8K bytes, and our page size is 512 bytes. What is the size of the PFN?
B. Suppose we have a virtual memory system in which virtual memory is divided into User Space and System Space, and System Page Table remains resident in physical memory. System space includes trap vector table, interrupt vector table, operating system and supervisor stack as shown in Figure A.1 in Appendix A. The rest of the address space in Figure A.1 is user space. If each PTE contained, in addition to the PFN, a Valid bit, a modified bit, and two bits of access control, how many bits of physical memory would be required to store the System Page Table?

Size of a page is 512 bytes.

Number of bits of address required to calculate the offset within a page is 9.

Number of frames in physical memory is (8K bytes) ÷ (512 bytes) = $2^{13} \div 2^{9} = 2^{4}$.

Size of PFN is 4 bits.

Size of PTE equals 1 (Valid) + 1 (Modified) + 2 (access control) + 4 (PFN) = 8 bits = 1 byte.

Number of virtual pages in System Space is $(3 \times 2^{12}) \div (2^{9})$ = 24 pages.

Size of System Page Table is 24 × 1 byte = 24 bytes = 24 × 8 bits = 192 bits.

Problem 2

An ISA supports an 8-bit, byte-addressable virtual address space. The corresponding physical memory has only 128 bytes. Each page contains 16 bytes. A simple, one-level translation scheme is used and the page table resides in physical memory. The initial contents of the frames of physical memory are shown below.

Frame Number	Frame Contents
0	empty
1	Page 13
2	Page 5
3	Page 2
4	empty
5	Page 0
6	empty
7	Page Table

A three-entry Translation Lookaside Buffer that uses LRU replacement is added to this system. Initially, this TLB contains the entries for pages 0, 2, and 13. For the following sequence of references, put a circle around those that generate a TLB hit and put a rectangle around those that generate a page fault. What is the hit rate of the TLB for this sequence of references? (Note: LRU policy is used to select pages for replacement in physical memory.)

References (to pages): 0, 13, 5, 2, 14, 14, 13, 6, 6, 13, 15, 14, 15, 13, 4, 3.

1. At the end of this sequence, what three entries are contained in the TLB?
2. What are the contents of the 8 physical frames?

Reference	TLB hit	Page Fault
0	X	
13	X	
5		
2		
14		X
14	X	
13		
6		X
6	X	
13	X	
15		X
14		
15	X	
13	X	
4		X
3		X

TLB hit rate = 7/16.

TLB contains entries for pages 3, 4, and 13.

Solutions for the final contents of the frames of physical memory may differ slightly depending on what order the initially empty frames were allocated; however, no page should appear in more than one frame. Possible answers are shown below.

Frame Number	Frame Contents
Frame 0	Page 14 (or 6 or 15)
Frame 1	Page 13
Frame 2	Page 3
Frame 3	Page 2
Frame 4	Page 6 (or 14 or 15)
Frame 5	Page 4
Frame 6	Page 15 (or 6 or 14)
Frame 7	Page Table

Problem 3

A little-endian machine with 64KB, byte addressable virtual memory and 4KB physical memory has two-level virtual address translation similar to the VAX. The page size of this machine is 256 bytes. Virtual address space is partitioned into the P0 space, P1 space, system space and reserved space. The space a virtual address belongs to is specified by the most significant two bits of the virtual address, with 00 indicating P0 space, 01 indicating P1 space, and 10 indicating system space. Assume that the PTE is 32 bits and contains only the Valid bit and the PFN in the format V0000000..000PFN.
For a single load instruction the physical memory was accessed three times, *excluding instruction fetch*. The first access was at location x108 and the value read from that location (`x10B`,`x10A`,`x109`,`x108`) was `x80000004`. Hint: What does this value mean?

The second access was at location x45C and the third access was at location x942.
If SBR = `x100`, P0BR = `x8250` and P1BR = `x8350`,

We can determine from the given information that:

- A Virtual Address (VA) is 16 bits (2^{16} = 64KB)
- A Physical Address (PA) is 12 bits (2^{12} = 4KB)
- The number of bits for the offset is 8 (2^{8} = 256 bytes)

The breakdown for a Virtual Address (VA) must be:

- VA[15:14] (2 bits) : Denotes the region of memory (P0, P1, System)
- VA[13:8] (6 bits) : The Virtual Page Number (VPN)
- VA[7:0] (8 bits) : The offset

The breakdown for a Physical Address (PA) must be:

- PA[11:8] (4 bits) : The Page Frame Number (PFN)
- PA[7:0] (8 bits) : The offset

1. What is the virtual address address corresponding to physical address `x45C`?

Answer: x825C

Given the VAX 2-level translation scheme, we know that this VA must be in system space. Therefore, the top 2 bits of the VA (VA[15:14]) must be 10 (in binary). We also know that the offset of the VA is the same as the offset of the PA, so the bottom 8 bits of the VA (VA[7:0]) must be x5C (01011100 in binary). Now, all we have to figure out is the 6 bit Virtual Page Number (VPN). It was given that the 1st access to physical memory (let's call this PA1) was at location x108. Once again, given the VAX 2-level translation scheme, we know that PA1 = SBR + (size of PTE in bytes) × VPN. Solving this equation for VPN we get VPN = (PA1 - SBR) ÷ (size of PTE in bytes).

It was given that PA1 is x108, SBR is x100, and the size of a PTE is 4 bytes. Therefore, the VPN is x2 (000010 in binary). The complete VA is therefore 10 000010 01011100 (in binary) which is x825C.

2. What is 32 bit value read from location `x45C`?

Answer: x80000009

The contents of physical address x45C (the 2nd access to physical memory) is a PTE. We know that a PTE is a 32 bit value that consists of 1 valid bit (PTE[31]), and a 4-bit PFN (PTE[3:0]). All other bits of the PTE (PTE[30:4]) are 0. The contents of this PTE are used to form the address of the 3rd access to physical memory (x942). Therefore, the PFN bits of the PTE must be x9, and the valid bit must be 1. This implies that the PTE is x80000009.

3. What is the virtual address corresponding to physical address `x942`?

Answer : x0342

First, we must determine if this VA is in P0 space or P1 space. To determine this, we have to figure out if P0BR or P1BR was used to compute the virtual address x825C (the answer to part a). It was given that P0BR is x8250, and that P1BR is x8350. Since x8350 is greater than x825C, we know that we could not have used P1BR to compute x825C, and therefore we must have used P0BR which means the VA is in P0 space. Therefore, the top 2 bits of the virtual address (VA[15:14]) must be 00 (in binary). We also know that the offset of the VA is the same as the offset of the PA, so the bottom 8 bits of the VA (VA[7:0]) must be x42 (01000010 in binary). Now, all we have to figure out is the 6 bit Virtual Page Number (VPN). Once again, given the VAX 2-level translation scheme, we know that x825C = P0BR + ((size of PTE in bytes) × VPN).

Solving this equation for VPN we get VPN = (x825C - P0BR) ÷ (size of PTE in bytes).

It was given that P0BR is x8250, and the size of a PTE is 4 bytes. Therefore, the VPN is x3 (000011 in binary). The complete VA is therefore 00 000011 01000010 (in binary) which is x0342.

Problem 4

Note: In this problem, the user and system virtual address spaces are not sized equally (the system virtual address space is 1/4 of the total virtual address space, and the user virtual address space makes up the other 3/4). Thus you need to include the address region bits in your calculation of the user space virtual page number. To make it easier for the machine to index into the user space page table, PTBR points to 0x380, which is at an offset of -0x20 from the actual first entry in the user space page table at 0x3A0. To index into the user space page table, add (user space virtual page number * PTE size) to the PTBR. (Why does this work?)

Consider a processor that supports a 9-bit physical address space with byte addressable memory. We would like the processor to support a virtual memory system. The features of the virtual memory system are:

```
Virtual Memory Size : 4 Kbytes (12 bit address-space)
 Page Size          : 32 bytes
 PTBR               : 0x380
 SBR                : 0x1E0
```

The virtual memory is divided into two spaces: system space and user space. System space is the first kilobyte of the virtual address space (i.e., most significant two bits of the virtual address are 00). The rest of the virtual memory is user space. The system page table remains resident in physical memory. Each PTE contains, in addition to the PFN, a Valid bit, a modified bit and 2 bits for access control. The format of the PTE is

Valid	Modified	Access Control	PFN

(Valid bit is the most significant bit of the PTE and the PFN is stored in the least significant bits.)

1. How many virtual pages does the system accommodate?

virtual pages = virtual address space ÷ size of page = 2^{12} Bytes ÷ 2^5 Bytes/page = 2^7 pages

2. What is the size of the PFN? How big is the PTE?

physical frames = physical address space ÷ size of frame = 2^9Bytes ÷ 2^5 Bytes/frame = 2^4 frames. Therefore, 4 bits are needed to specify the PFN.

Size of PTE = Valid bit + Modified bit + access control bits + PFN bits = 1 + 1 + 2 + 4 = 8 bits (1 Byte)

3. How many bytes are required for storing the entire user space pagetable? How many pages does user space page table occupy?

User space = (3/4) × Virtual address space = (3/4) × 2^7 pages = 3 × 2^5 pages. Each page of user space will have a PTE in the user space page table.

Size of user page table is # of entries × size of PTE = (3 × 2^5 entries) × 1 Byte/entry = 96 Bytes.

# of pages = 96 Bytes ÷ 2^5 Bytes/page = 3 pages.

4. Since the user space page table can occupy a significant portion of the the physical memory, this system uses a 2 level address translation scheme, by storing the user space Page Table in virtual memory (similar to VAX).

Given the virtual address 0x7AC what is the Physical address?

The following table shows the contents of the physical memory that you may need:

Address	Data
x1F8	xBA
x1F9	xBB
x1FA	xBC
x1FB	xBD
x1FC	xBE
x1FD	xB8
x1FE	xB7
x1FF	xB6

Address	Data
x118	x81
x119	x72
x11A	x65
x11B	x34
x11C	x97
x11D	x83
x11E	xC6
x11F	xB2

We'll use the prefix “b” to indicate a binary number in this solution. Also, for clarity, we will call the virtual address x7AC the virtual address of X (VA_X).
Virtual address VA_X = x7AC
The three parts of this virtual address are:
VA_X[11:10]: b01 (indicates that this is an address in user space)
VA_X[11:5] (7 bits): Virtual Page Number = b0111101
VA_X[4:0] Offset within page: b01100

a. X is on page x03D of user space
b. VA of the PTE of the page containing x is VA_PTE_X = PTBR + (x03D × 1) = x380 + x03D = x3BD.
c. Virtual page of System Space of PTE is VA_PTE_X[11:5] = x01D.
d. PA of the PTE of this page of System Space is PA_PTE_PTE = SBR + VA_PTE_X[11:5] × 1 = x1E0 + x01D = x1FD.

The PTE of this page of System Space is:
PTE_PTE_X = Memory[x1FD] = xB8
PFN_PTE_X = PTE_PTE_X[3:0] = x8
PA of the PTE of the page containing X:
PA_PTE_X = PFN_PTE_X concatenated with VA_PTE_X[4:0] = x11D
PTE_X = Memory[x11D] = x83
PFN_X = PTE_X[3:0] = x3
PA_X = PFN_X concantenated with VA_X[4:0] = x06C

Problem 5

The virtual address of variable X is `x3456789A`. Find the physical address of X. Assume a Virtual Memory model similar to VAX.

Remember that in VAX each Virtual Address consists of:

1. 2 bits to specify the Address Space
2. 21 bits to specify Virtual Page Number
3. 9 bits to specify the byte on the page

You will need to know the contents of P0BR: `x8AC40000` and SBR: `x000C8000`.

You will also need to know the contents of the following physical memory locations:
`x1EBA6EF0: x80000A72`
`x0022D958: x800F5D37`

Some intermediate questions to help you:

- What virtual page of P0 Space is X on?
- What is VA of the PTE of the page containing X?
- What virtual page of System Space is this PTE on?
- What is the PA of the PTE of this page of System Space?
- What is the PA of the PTE of the page containing X?

We'll use the prefix “b” to indicate a binary number in this solution.

Virtual address VA_X = x3456789A
The three parts of this virtual address are:
VA_X[31:30]: b00 (indicates that this is an address in P0 space)
VA_X[29:9] (21 bits): Virtual Page Number = b 11 0100 0101 0110 0111 100 (x1A2B3C)
VA_X[8:0] Offset within page: b010011010

- x is on page x1A2B3C of P0 space
- VA of the PTE of the page containing X:
  VA_PTE_X = P0BR + (x1A2B3C × 4) = x8AC40000 + x68ACF0 = x8B2CACF0
- Virtual page of System Space of PTE is VA_PTE_X[29:9] = x59656
- PA of the PTE of this page of System Space is PA_PTE_PTE = SBR + VA_PTE_X[29:9] × 4 = x22D958
- The PTE of this page of System Space is PTE_PTE_X = Memory[x22D958] = x800F5D37
  PFN_PTE_X = PTE_PTE_X[20:0] = xF5D37
  PA of the PTE of the page containing X:
  PA_PTE_X = PFN_PTE_X concatenated with VA_PTE_X[8:0] = x1EBA6EF0
  PTE_X = Memory[x1EBA6EF0] = x80000A72
  PFN_X = PTE_X[20:0] = xA72
  PA_X = PFN_X concantenated with VA_X[8:0] = x14E49A

Problem 6

Given a VAX-like virtual memory with 8b virtual addresses where the most significant bit indicates P0 (MSB == 0) or system segment (MSB == 1), the next 4 bits are the virtual page number and the last 3 bits of address are the page offset. The machine is byte-addressable.

The format of a page table entry (PTE) is shown below.

7	6	5	4	3	2	1	0
Valid	0	Protection		Physical Frame Number			

Protection bits:
00: none
01: read-only
10: read-write
11: (invalid)

The following single instruction is executed
MEM[R0] = 3;
The state of the physical memory after executing that instruction is shown below. The blank frames represent frames to which no pages are mapped. No page faults occurred during the program run. P0LR and SLR are 1, and P1LR is 0.

a. What is the physical address containing 3?

Observing the state of memory after executing instruction, only one location (Location 26) contains the value 3. Hence, Physical address = 26 (0x1A)

b. What is the PTE of the page containing 3?

Physical Frame Number of Page containing 3 = 3 (Address 26 is in frame 3)
Therefore PTE = 0b10100011 = 0xA3

Valid = 1	0	Protection = Read-write	Physical Frame Number = 3
1	0	10	0011

c. What is SBR?

Observing the state of memory, only one location (Location 33) contains the value 0xA3. Hence, PA of PTE = 33 (0x21)
Physical Frame Number of Page containing PTE = 4 (Address 33 is in frame 4)

Therefore PTE of PTE = 0b10010100 = 0x94

Valid = 1	0	Protection = Read only	Physical Frame Number = 3
1	0	01	0100

Observing the state of memory after executing instruction, only one location (Location 49) contains the value 0x94. Hence, Physical address of PTE of PTE = 49

(As PTE is stored in system memory and we need to read PTE, the PTE should be atleast read only. Also Frame number is assigned by OS, so this location won't have write access to user. If we assume that there is write access, there is no entry with data 0xA4. So it's safe to assume read only)

PA of PTE (PTE)= SBR + VPN_{PTE} * Size of PTE
SBR + VPN * 1 = 49 (PTE size = 1 Byte)

VPN of System region can only be 0 as SLR=1

Therefore, SBR = 49 (0x31)

d. What is P0BR?

VA of PTE = 0b10000001 = 0x81 = 129

Region = System	VPN = 0000	Offset = Offset of PA of PTE = 001
1	0000	001

VA of PTE = P0BR + $VPN_{orig\ access}$ * PTE Size
129 = P0BR + VPN*1

Again, P0LR =1 , hence VPN should be 0

Therefore P0BR = 129 (0x81)

e. What is the virtual address of the location containing 3?

VA of Location containing 3 = 0b00000010 = 0x02

Region = P0	VPN = 0000	Offset = Offset of PA of Location containing 3= 010
0	0001	010

Frame	Addr	Dec	Hex	Bin	Frame	Addr	Dec	Hex	Bin
0	0				4	32	30	0x1e	0b0001 1110
	1					33	163	0xa3	0b1010 0011
	2					34	82	0x52	0b0101 0010
	3					35	10	0x0a	0b0000 1010
	4					36	105	0x69	0b0110 1001
	5					37	93	0x5d	0b0101 1101
	6					38	56	0x38	0b0011 1000
	7					39	4	0x04	0b0000 0100
2	16	37	0x25	0b0010 0101	6	48	28	0x1c	0b0001 1100
	17	123	0x7b	0b0111 1011		49	148	0x94	0b1001 0100
	18	96	0x60	0b0110 0000		50	21	0x15	0b0001 0101
	19	31	0x1f	0b0001 1111		51	86	0x56	0b0101 0110
	20	36	0x24	0b0010 0100		52	93	0x5d	0b0101 1101
	21	11	0x0b	0b0000 1011		53	80	0x50	0b0101 0000
	22	19	0x13	0b0001 0011		54	11	0x0b	0b0000 1011
	23	121	0x79	0b0111 1001		55	88	0x58	0b0101 1000
3	24	20	0x14	0b0001 0100	7				
	25	90	0x5a	0b0101 1010					
	26	3	0x03	0b0000 0011					
	27	81	0x51	0b0101 0001					
	28	18	0x12	0b0001 0010					
	29	31	0x1f	0b0001 1111					
	30	27	0x1b	0b0001 1011					
	31	24	0x18	0b0001 1000					

Problem 7

A computer has an 8KB write-through cache. Each cache block is 64 bits, the cache is 4-way set associative and uses a victim/next-victim pair of bits for each block for its replacement policy. Assume a 24-bit address space and byte-addressable memory. How big (in bits) is the tag store?

An 8KB cache size with a 8B line size, in a 4-way set associative cache means there are 8KB ÷ (4 × 8B) = 256 sets in the cache.

Since there are 256 or 2^8 sets, 8 bits are required to index into the correct set. Since there are 8B or 2^3 bytes in a cache line, 3 bits are required to find the correct byte within a block. Given a 24-bit address space, this leaves 24 - 8 - 3 = 13 bits left over for the tag store. Additionally, the tag store must hold 2 bits for the V/NV replacement policy and 1 valid bit. This means each cache line must have a 16-bit tag store associated with it. 2B of tag store times (256 × 4) cache lines in the cache means that the tag store, in total takes up 2048 bytes, which is 16384 bits

Problem 8

An LC-3b system ships with a two-way set associative, write back cache with perfect LRU replacement. The tag store requires a total of 4352 bits of storage. What is the block size of the cache? Please show all your work.

Hint: **$4352 = 2^{12} + 2^8$**.

The size of the tag store is $2^{12} + 2^8$. We know that the size of the tag store can be given as the product of number of sets and number of bits per set.

We also know that address space is 16-bits. Hence, tag + index + bib = 16

In order to find bib, we need to find index and tag. The following bits are necessary for each set of the tag store:

- 1 bit for LRU (remember: you only need one bit per set for a 2-way associative cache)
- 2 valid bits
- 2 dirty bits
- 2 × tag tag bits

Total number of bits per set is 5 + 2 × tag. An important conclusion that can be drawn is that number of bits per set will always be an odd number.

We will also use the fact that number of sets is always a power of 2 (since it is indexed using an integer number of bits). Given the size of the tag store, index has to be a number less than 8 (the size of the tag store is indivisible by 2^9 or greater).

The only value of index that fits both criterion is 8. Therefore:

Number of sets = $2^8 = 256$
Bits per row = 4352 ÷ 256 = 17
17 = 5 + 2 × tag ⇒ tag = 6
6 + 8 + bib = 16 ⇒ bib = 2

Hence, the cache block size is 4 bytes

Problem 9

Hamacher, pg.255, question 5.13. A byte-addressable computer has a small data cache capable of holding eight 32-bit words. Each cache block consists of one 32-bit word. When a given program is executed, the processor reads data from the following sequence of hex addresses:

`200, 204, 208, 20C, 2F4, 2F0, 200, 204, 218, 21C, 24C, 2F4`

This pattern is repeated four times.

1. Show the contents of the cache at the end of each pass throughout this loop if a direct-mapped cache is used. Compute the hit rate for this example. Assume that the cache is initially empty.

The address is divided into 3 portions:

- address[1:0] (2 bits) for the byte in the block
- address [4:2] (3 bits) for the cache index
- address[11:5] (7 bits) for the tag
-

The contents of the cache at the end of each pass are the same. They are shown below:

Valid	Tag	Data (addresses are written inside each byte)			
1	0010 000	203	202	201	200
1	0010 000	207	206	205	204
1	0010 000	20B	20A	209	208
1	0010 010	24F	24E	24D	24C
1	0010 111	2F3	2F2	2F1	2F0
1	0010 111	2F7	2F6	2F5	2F4
1	0010 000	21B	21A	219	218
1	0010 000	21F	21E	21D	21C

The hit/miss information for each pass is shown below:

Reference	200	204	208	20C	2F4	2F0	200	204	218	21C	24C	2F4
Pass 1:	M	M	M	M	M	M	H	H	M	M	M	H
Pass 2:	H	H	H	M	H	H	H	H	H	H	M	H
Pass 3:	H	H	H	M	H	H	H	H	H	H	M	H
Pass 4:	H	H	H	M	H	H	H	H	H	H	M	H

Hit rate is 33/48

2. Repeat part (a) for a fully-associative cache that uses the LRU-replacement algorithm.

The address is divided into two: 2 bits for identifying the byte in the block, 10 bits for the tag. No bits are needed for cache index. The following table shows the contents of the cache at the end of each pass (Valid bits and Tags are ignored, they should be obvious. Starting addresses of blocks in the cache are provided):

Pass	Way 0 Data	Way 1 Data	Way 2 Data	Way 3 Data	Way 4 Data	Way 5 Data	Way 6 Data	Way 7 Data
1	200	204	24C	20C	2F4	2F0	218	21C
2	200	204	21C	24C	2F4	20C	2F0	218
3	200	204	218	21C	2F4	24C	20C	2F0
4	200	204	2F0	218	2F4	21C	24C	20C

The hit/miss information for each pass is shown below:

Reference:	200	204	208	20C	2F4	2F0	200	204	218	21C	24C	2F4
Pass 1:	M	M	M	M	M	M	H	H	M	M	M	H
Pass 2:	H	H	M	M	H	M	H	H	M	M	M	H
Pass 3:	H	H	M	M	H	M	H	H	M	M	M	H
Pass 4:	H	H	M	M	H	M	H	H	M	M	M	H

Hit rate is 21/48

3. Repeat part (a) for a four-way set-associative cache that uses the LRU replacement algorithm.

The address is divided into 3 portions: 2 bits for identifying the byte in the block, 1 bit for the cache index, 9 bits for the tag. The contents of the cache at the end of Pass 1:

Way 0			Way 1			Way 2			Way 3		
V	Tag	Data	V	Tag	Data	V	Tag	Data	V	Tag	Data
1	0010 0000 0	203-200	1	0010 0000 1	20B-208	1	0010 1111 0	2F3-2F0	1	0010 0001 1	21B-218
1	0010 0000 0	207-204	1	0010 0100 1	24F-24C	1	0010 1111 0	2F7-2F4	1	0010 0001 1	21F-21C

The hit/miss information for each pass is shown below:

Reference:	200	204	208	20C	2F4	2F0	200	204	218	21C	24C	2F4
Pass 1:	M	M	M	M	M	M	H	H	M	M	M	H
Pass 2:	H	H	H	M	H	H	H	H	H	M	M	H
Pass 3:	H	H	H	M	H	H	H	H	H	M	M	H
Pass 4:	H	H	H	M	H	H	H	H	H	M	M	H

Contents of the second set of the cache (index equals 1) after pass 2, 3, and 4 (the first set remains the same):

Pass	Way 0			Way 1			Way 2			Way 3		
	V	Tag	Data	V	Tag	Data	V	Tag	Data	V	Tag	Data
2	1	0010 0000 0	207-204	1	0010 0001 1	21F-21C	1	0010 1111 0	2F7-2F4	1	0010 0100 1	24F-24C
3	1	0010 0000 0	207-204	1	0010 0100 1	24F-24C	1	0010 1111 0	2F7-2F4	1	0010 0001 1	21F-21C
4	1	0010 0000 0	207-204	1	0010 0001 1	21F-21C	1	0010 1111 0	2F7-2F4	1	0010 0100 1	24F-24C

Hit Rate is 30/48

Problem 10

Below, we have given you four different sequences of addresses generated by a program running on a processor with a data cache. Cache hit ratio for each sequence is also shown below. Assuming that the cache is initially empty at the beginning of each sequence, find out the following parameters of the processor's data cache:

- Associativity (1, 2, or 4 ways)
- Block size (1, 2, 4, 8, 16, or 32 bytes)
- Total cache size (256B, or 512B)
- Replacement policy (LRU or FIFO)

Assumptions: all memory accesses are one byte accesses. All addresses are byte addresses.

Number	Address Sequence	Hit Ratio
1	0, 2, 4, 8, 16, 32	0.33
2	0, 512, 1024, 1536, 2048, 1536, 1024, 512, 0	0.33
3	0, 64, 128, 256, 512, 256, 128, 64, 0	0.33
4	0, 512, 1024, 0, 1536, 0, 2048, 512	0.25

Block Size

The following table lists hit ratios for different cache block sizes given the access sequence 1 (0, 2, 4, 8, 16, 32):

Block Size	Hit Ratio
1B	0/6
2B	0/6
4B	1/6
8B	2/6
16B	3/6
32B	4/6

Since the hit ratio is reported as 0.33 for this sequence, the block size must be 8 bytes. Therefore, the accesses look like this:

Address	0	2	4	8	16	32
Hit/Miss	M	H	H	M	M	M

Associativity

Notice that all of the addresses of sequence 2 (0, 512, 1024, 1536, 2048, 1536, 1024, 512, 0) are multiples of 512. This means that they all map to the same set, since the size of the cache can only be 256 or 512 bytes. Therefore, the hit ratio would be 0/9 in case of a direct-mapped cache. In case of a 2-way cache, the accesses would behave as follows:

Address	0	512	1024	1536	2048	1536	1024	512	0
Hit/Miss	M	M	M	M	M	H	M	M	M

The resulting hit rate would be 1/9. Similarly, for 4-way cache:

Address	0	512	1024	1536	2048	1536	1024	512	0
Hit/Miss	M	M	M	M	M	H	H	H	M

The resulting hit rate would be 3/9, and thus the cache is 4-way set associative.

Cache Size

If the cache size were 256B, there would be 3 index bits and all of the adresses in sequence 3 (0, 64, 128, 256, 512, 256, 128, 64, 0) would map to the same set:

Address	0	64	128	256	512	256	128	64	0
Hit/Miss	M	M	M	M	M	H	H	H	M

The resulting hit ratio would be 3/9, and thus the cache size is 256B. For completeness, here's how the accesses would look like in case of a 512B cache (4 index bits):

Address	0	64	128	256	512	256	128	64	0
Hit/Miss	M	M	M	M	M	H	H	H	H

Replacement Policy

In case of the FIFO replacement policy, the accesses of sequence 3 (0, 512, 1024, 0, 1536, 0, 2048, 512) would look as follows:

Address	0	512	1024	0	1536	0	2048	512
Hit/Miss	M	M	M	H	M	H	M	H

The resulting hit ration would be 3/8. However, in case of the LRU replacement policy the hit rate would be 0.25:

0	512	1024	0	1536	0	2048	512
M	M	M	H	M	H	M	M

Thus the replacement policy is LRU.

The following problems are meant to help you study for the test. These problems do NOT need to be turned in.

Problem 1 (Ungraded)

Let's say we added a virtual memory system to the LC-3b. Which instructions can possibly generate a page fault? What is the maximum number of page faults an instruction can possibly generate while it is being processed? Which instructions can possibly generate that maximum number of page faults?

Assume that the virtual memory system added uses a one-level translation scheme and the page table is always resident in physical memory.

An instruction is said to generate a page fault if a page fault occurs at any time during the processing of that instruction.

Including instruction fetch, every instruction can generate a page fault. Ignoring instruction fetch, LDB, LDW, STB, STW, TRAP, RTI can generate a page fault (If the trap vector table or system stack is always in physical memory, then the TRAP or RTI won't generate a page fault).

Including the instruction fetch, RTI can generate the maximum number of page faults (3) and LDB, LDW, STB, STW, TRAP can generate the next most number of page faults (2).