CMPU 334 Quiz1
Fall 2019

Name: SOLUTIONS

Instructions:

This is a closed book, closed notes exam. No electronic devices, including calculators, are allowed. You have 120 minutes. There are 8 problems and 12 pages to this exam.

Good Luck!

1 (10)	
2 (10)	
3 (8)	
4 (15)	
5 (8)	
6 (15)	
7 (10)	
8 (24)	
Total (100)	

1. (10 points) Assuming a replacement policy of **FIFO** (first in first out) and a memory that can hold 3 pages, figure out whether each of the following page references is a hit or miss in the page cache (main memory). Show whether each page reference results in a hit or a miss, and the resulting state of the pages (i.e., which virtual pages are currently in memory after the page is accessed). Draw the state of the memory such that the next page to be evicted is first on the list, followed by the second page to be evicted (if in memory), and finally the third page to be evicted (if in memory).

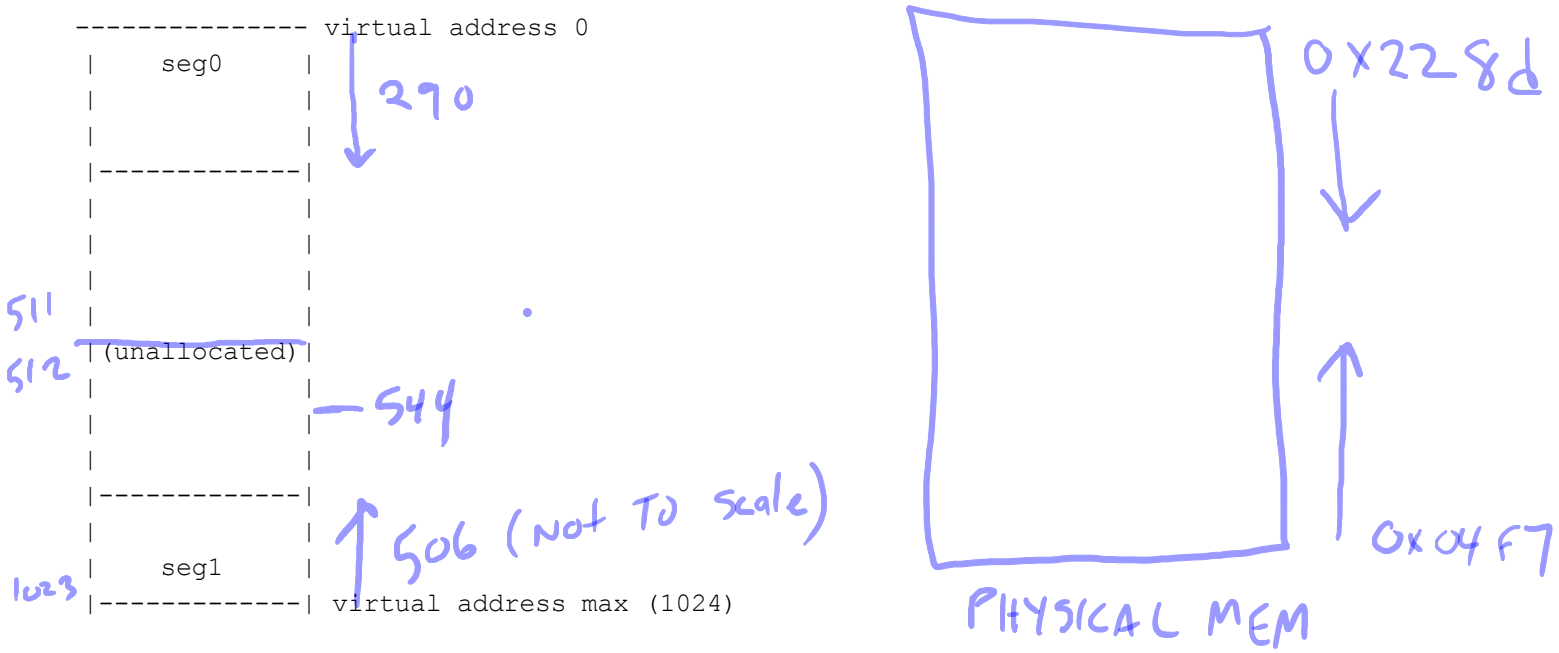
Virtual Page Access	Hit or Miss? (Circle One)	State of Memory (3 pages)
1	Hit **Miss**	**1**
3	Hit **Miss**	**1, 3**
3	**Hit** Miss	**1, 3**
8	Hit **Miss**	**1, 3, 8**
3	**Hit** Miss	**1, 3, 8**
1	**Hit** Miss	**1, 3, 8**
8	**Hit** Miss	**1, 3, 8**
5	Hit **Miss**	**3, 8, 5**
7	Hit **Miss**	**8, 5, 7**
8	**Hit** Miss	**8, 5, 7**

Now assume a **LRU** replacement policy with the same parameters as above.

Virtual Page Access	Hit or Miss?	State of Memory (3 pages)
0	Hit **Miss**	**0**
6	Hit **Miss**	**0, 6**
4	Hit **Miss**	**0, 6, 4**
6	**Hit** Miss	**0, 4, 6**
1	Hit **Miss**	**4, 6, 1**
3	Hit **Miss**	**6, 1, 3**
6	**Hit** Miss	**1, 3, 6**
3	**Hit** Miss	**1, 6, 3**
7	Hit **Miss**	**6, 3, 7**
8	Hit **Miss**	**3, 7, 8**

2. (10 points) For each virtual address below, either write down the physical address it translates to **or** write down that it is an out-of-bounds address (a segmentation violation). For this problem, you should assume a simple address space with two segments: the top bit of the virtual address is used to check whether the virtual address is in segment 0 (msb=0) or segment 1 (msb=1). Note that the base/limit pairs given to you grow in different directions, depending on the segment, i.e., segment 0 grows in the positive direction, whereas segment 1 in the negative. Physical memory is 16k in size. **Write the translated physical address in the line below (in hexadecimal) or circle segmentation violation if the address could not be translated.**

Visually, the address space looks like this:



Segment register information:

Segment 0 base (grows positive) : 0x228d
Segment 0 limit : 290 (decimal)

Segment 1 base (grows negative) : 0x04f7
Segment 1 limit : 506 (decimal)

Virtual Address Trace

140 <= 290 valid; To Get PA ADD offset To BASE

seg 0 VA 0: 0x08c (decimal: 140) --> PA ___**0x2319**_____ or segmentation violation?

228d (13)
+ 08c (12)
2319

seg 0 VA 1: 0x142 (decimal: 322) --> PA 322 > 290 or **segmentation violation**?

seg 0 VA 2: 0x1dc (decimal: 476) --> PA 476 > 290 or **segmentation violation**?

seg 1 VA 3: 0x220 (decimal: 544) --> PA ___**0x0317**_____ or segmentation violation?

Grows UPWARDS
Need To Get NEGATIVE OFFSET

seg 1 VA 4: 0x3a6 (decimal: 934) --> PA ___**0x049d**_____ or segmentation violation?

1. Get offset From Segment
2. Get Negative offset
3. Subtract From BASE

0x220
- 200 (1024)
0x 20

0x200
- 0x 20
1E0

4F7
1E0
0x317

Bounds check
1024
- 544
480 < 506

3. (8 points) The following 4 jobs (with their running times listed) **arrive at time 0** in the following order:

Job 0 (length = 10)
Job 1 (length = 5)
Job 2 (length = 3)
Job 3 (length = 4)

Compute the **response time** and the **turnaround time** for the following scheduling algorithms. Assume the RR order is the same as the order in which the jobs are listed and a time quanta of 2.

SJF (Shortest Job First)

Job	Response Time	Turnaround Time
0	**12**	**22**
1	**7**	**12**
2	**0**	**3**
3	**3**	**7**

FIFO (First In First Out)

Job	Response Time	Turnaround Time
0	**0**	**10**
1	**10**	**15**
2	**15**	**18**
3	**18**	**22**

RR (Round Robin)

Job	Response Time	Turnaround Time
0	**0**	**22**
1	**2**	**18**
2	**4**	**13**
3	**6**	**15**

Which one of the above algorithms is the most fair?

Circle one: SJF FIFO **RR**

4. (15 points). The MLFQ (multi-level feedback queue) scheduler has a number of rules. Circle the rules that are actually part of the final MLFQ policy.

1. **If Priority(A) > Priority(B), A runs (B does not)**
2. If Priority(A) > Priority(B), A is run to completion, followed by B
3. If Priority(A) < Priority(B), A runs (B does not)
4. If Priority(A) < Priority(B), A is run to completion, followed by B
5. **If Priority(A) = Priority(B), A and B are run in a round-robin fashion**
6. If Priority(A) = Priority(B), they are run to completion in FIFO order
7. **When a job enters the system, it is placed at the highest priority queue (the topmost queue)**
8. When a job enters the system, it is placed at the lowest priority queue (the bottommost queue)
9. When a job enters the system, it is placed in the queue with the fewest number of jobs
10. Once a job uses up its time slice at a given level, it moves to the end of the round-robin queue
11. **Once a job uses up its time slice at a given level, its priority is reduced**
12. Once a job uses up its time slice it is given a priority boost
13. **After some time period, move all the jobs in the system to the highest-priority queue**
14. After some time period, move each job in the system one level higher
15. After some time period, move each job in the system one level lower

5. (8 points) We have a system with three processes A, B, and C on a system with a single CPU. Processes can be in one of three states: Running, Ready, or Blocked. If the process does not exist (yet) or has exited, just put a "-------" down for that entry.

	State?		
	A	B	C
Process A is loaded into memory and starts executing in main().	**Run**	**------**	**------**
Process A calls fork() and creates process B, (but A, the parent, keeps running).	**Run**	**Ready**	**------**
Process A issues a request of the disk; B starts executing at the return from fork().	**Blocked**	**Run**	**-------**
B calls fork(), creating process C; B keeps running.	**Blocked**	**Run**	**Ready**
B's time slice expires; C runs.	**Blocked**	**Ready**	**Run**
C calls exec(); there are no other changes.	**Blocked**	**Ready**	**Run**
A's I/O completes but there are no other changes.	**Ready**	**Ready**	**Run**
C waits for user input. A runs.	**Run**	**Ready**	**Blocked**

6. (15 points) Multi-level page tables

Assume we have a system with the following parameters:

- The page size is very small 32 bytes (note: this is unrealistically small for a real system)
- The virtual address space is 1024 pages (so 32 KB virtual memory in total)
- Physical memory consists of 128 pages (so 4 KB physical memory in total)

How many bits are there in the virtual address? **15**_____

How many bits are there in the physical address? **12**_____

The system assumes a multi-level page table. The upper five bits of a virtual address are used to index into a page directory; the page directory entry (PDE), if valid, points to a page of the page table. Each page table page holds 32 page-table entries (PTEs). Each PTE, if valid, holds the desired translation (physical frame number, or PFN) of the virtual page in question.

Page table entries (PTE) are 1 byte long. The msb (most significant bit) of each PTE is a valid bit, and the lower 7 bits contain the PFN if the valid bit is 1. For example, a PTE of 0x81 would be a valid entry, with a PFN of 1.

Page directory entries (PDE) are also 1 byte long, and like the PTE, the msb is the valid bit. If the valid bit is 1, the lower 7 bits contain the PFN of that portion of the page table.

For this problem, the page directory base register (PDBR) is located at the physical page 8. This means the PDE is located at PFN 8.

Below is a complete dump of each page of memory. A page dump looks like this:

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

page 0: 08 00 01 15 11 1d 1d 1c 01 17 15 14 16 1b 13 0b ...
page 1: 19 05 1e 13 02 16 1e 0c 15 09 06 16 00 19 10 03 ...
page 2: 1d 07 11 1b 12 05 07 1e 09 1a 18 17 16 18 1a 01 ...
...
```

which shows the 32 bytes found on pages 0, 1, 2, and so forth. The first byte (offset 0) on page 0 has the value 0x08, the second byte (offset 1) is 0x00, the third (offset 2) 0x01, and so forth.

You are given below a list of virtual addresses to translate. Use the PDBR to find the relevant page table entries for this virtual page. Then find if it is valid. If so, use the translation to form a final physical address. Using this address, you can find the VALUE that the memory reference is looking for. The virtual address may not be valid and thus generate a fault. In that case write FAULT as your answer. **Give all numeric answers in hexadecimal.**

Virtual Address	PDE Index	PDE Contents	PTE Index	PTE Contents	Physical Address	Value at Address
556a	**0x15**	**0xbd**	**0xb**	**0xd7**	**0xaea**	**03**
74dc	**0x1d**	**0xaf**	**0x6**	**0x7f**	**FAULT**	**FAULT**
6f59	**0x1b**	**0xcc**	**0x1a**	**0xd8**	**0xb19**	**19**

Translation of address 0x556A

Page size = 32 bytes so 5 bits for offset

1024 virtual pages so 10 bits for VPN

128 physical pages so 7 bits for PFN

Each page table entry is one byte, so 32 PTEs to one page. We need $\frac{1024}{32} = 32$ pages of PTEs.

Each page directory entry is 1 byte, so 32 PDEs to one page, so we only need 1 page of the page directory

Virtual address = 15 bits

10-bits	5-bits
VPN	offset

Physical address = 12 bits

7-bits	5-bits
PFN	offset

VA

5-bits PDE index	5-bits PTE index	5-bits offset
VPN 10 bits		

0x556A => 0|101 01|01 011|0 1010

PDE index | PTE index | offset

21 (0x15) | 11 (0xb) | 10 (0xa)

upper 5 bits of VPN

PDE index = 0x15 (21 decimal)

PDE contents: From the problem description, we know that the page directory is at PFN #8. The PDE contents are at row 8, column 21 of the page dump and has the value 0x3D => 1011 1101

1011 1101: valid bit | PTE location

PTE index: lower 5-bits of VPN = 11 0xb

PTE contents: The PTE is located at page 0111101 => 0x3D (61 decimal)

The PTE contents are at row 61, column 11 and has the value 0xD7

1101 0111: valid bit | PFN location

Physical address: PFN + offset (7-bits, 5-bits)

Value at address: 1010 1110 1010 => 0xAEA

PFN = 0x57 => 87 decimal (5*16+7)

offset = 0xA (10 decimal)

Value is located at row 87 and column 10 => 0x3

PTE INDEX

PDE INDEX

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
page	0:	13	17	09	1c	15	0f	0a	06	12	0e	07	0c	09	10	08	1b	16	01	12	09	0a	11	1b	1d	1c	0c	16	0a	0a	07	09	0e
page	1:	01	12	18	01	0c	19	19	16	12	15	02	04	02	11	18	13	19	11	1a	0e	0e	1e	19	0b	0d	17	18	09	07	14	12	00
page	2:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page	3:	7f	7f	7f	a8	a4	7f	7f	7f	7f	7f	7f	7f	7f	7f	e7	7f	7f	7f	7f	7f	7f	7f	7f	de	7f	7f	7f	7f	7f	7f	7f	7f
page	4:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page	5:	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	be	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	99	7f	7f
page	6:	04	1e	16	07	1e	0d	1e	13	14	05	15	07	12	0f	07	18	1a	07	06	00	13	1a	1c	05	18	06	09	19	0d	1b	0f	17
page	7:	7f	7f	7f	7f	7f	7f	7f	c7	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	90	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f
page	8:	96	ad	a1	d6	fe	d1	c4	e4	7f	c0	a0	7f	9e	8b	f7	fb	9f	fc	87	c2	cb	bd	ee	83	ac	fd	85	cc	7f	af	95	ae
page	9:	07	0d	1c	10	00	0a	10	03	19	0c	0a	0d	07	0c	16	16	16	1c	0c	13	18	14	02	1d	19	01	0c	02	1c	0d	11	0f
page	10:	06	0d	04	12	10	02	1c	13	04	1b	19	12	1b	1c	09	10	0b	16	1e	09	0e	12	00	03	01	15	02	07	0e	08	11	05
page	11:	7f	7f	7f	7f	7f	7f	f8	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	dd	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	b2
page	12:	19	19	10	09	06	06	05	1b	13	0d	1a	12	18	14	01	10	12	16	0d	1e	0d	09	08	1a	00	03	1b	1c	1c	08	1b	11
page	13:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page	14:	02	01	05	15	05	1b	18	01	19	09	16	04	04	05	04	07	1d	12	1a	16	16	0e	04	07	18	0d	04	10	18	0e	0d	11
page	15:	02	14	08	18	06	1e	1e	1b	01	01	17	1b	0c	12	1e	1e	1a	1a	0d	0a	1e	02	18	00	03	10	04	12	0c	19	09	1e
page	16:	08	1c	18	06	08	1b	13	1c	06	11	0b	08	0d	17	19	1b	1d	17	15	08	15	0a	12	1a	14	18	0a	1b	1d	0b	1d	09
page	17:	18	0a	0d	0c	18	09	0a	09	13	19	00	05	19	1e	10	10	1c	1a	16	0b	01	0a	07	09	07	04	02	0d	1e	02	19	02
page	18:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page	19:	15	02	13	1b	09	05	16	0c	0b	04	00	02	1c	1c	0a	0c	02	1d	09	14	10	06	01	0e	0e	11	06	08	18	15	15	0d
page	20:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page	21:	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	f1	7f	7f	
page	22:	8a	e0	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	a7	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f
page	23:	0b	15	05	0c	1b	02	18	1a	12	08	0a	0d	17	14	04	13	13	05	09	15	02	14	01	1e	01	15	14	15	04	12	1a	07
page	24:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page	25:	0e	0c	1b	11	03	07	17	09	06	0b	03	0f	08	03	0a	17	11	00	0f	0f	19	05	19	1e	1e	03	10	0e	10	02	06	1a
page	26:	05	03	08	07	01	12	16	08	0d	03	09	07	1b	16	08	0e	0d	11	1a	13	0b	15	10	06	06	1d	0c	04	11	17	03	1b
page	27:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page	28:	13	12	1e	03	08	10	16	10	0e	09	03	1a	1d	12	1d	14	16	00	03	08	05	07	17	1b	07	03	10	0b	1c	07	0f	11
page	29:	1e	17	00	08	12	12	10	00	13	05	16	17	11	18	09	0d	05	03	13	13	0a	03	10	13	0b	19	1c	15	0b	1e	1d	08
page	30:	7f	7f	d4	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	93	7f	7f	7f	7f	7f	7f	7f	7f	7f
page	31:	7f	7f	7f	7f	7f	7f	7f	b7	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	c6	7f	7f	7f	7f	7f	7f
page	32:	7f	7f	7f	7f	7f	7f	7f	80	7f	7f	7f	7f	7f	e6	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f
page	33:	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	b8	7f	7f	7f	b6	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f
page	34:	10	05	03	06	0d	0b	0f	0e	0b	0f	18	03	08	1a	0a	05	12	03	10	0a	1a	07	0d	1c	15	13	1a	07	1e	18	08	12
page	35:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page	36:	19	11	13	17	09	13	18	07	11	00	09	08	07	04	11	0e	1a	0d	18	0a	17	04	1a	10	18	1e	01	13	19	12	0a	04
page	37:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page	38:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page	39:	09	1b	1a	15	16	12	09	02	08	05	17	13	0d	13	18	05	00	04	15	0d	1b	1b	12	16	02	07	08	0e	1e	19	07	13
page	40:	1b	1a	09	1b	16	16	07	17	05	03	07	0c	1a	1e	15	01	07	1a	02	1e	0a	0e	1e	0a	1a	07	13	16	16	11	0a	15
page	41:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page	42:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page	43:	06	1b	0a	19	1a	13	15	14	0f	05	05	08	03	13	1d	09	1b	18	0d	13	11	14	04	18	03	13	12	06	06	02	0a	04
page	44:	7f	7f	b4	7f	7f	ef	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f
page	45:	7f	7f	7f	7f	7f	7f	7f	7f	86	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	bb	7f
page	46:	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	e1	7f	7f
page	47:	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	eb	7f	7f	7f
page	48:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page	49:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page	50:	1d	08	18	07	13	0d	04	1b	06	0f	1c	19	16	11	19	0e	06	08	11	1a	0b	1c	15	19	1a	10	0f	1b	14	08	17	00
page	51:	15	0e	00	06	10	11	10	03	0d	12	07	01	15	02	0d	1b	03	01	17	10	10	1c	07	02	14	0b	12	19	04	15	00	09
page	52:	05	1d	0a	03	08	1e	02	09	0b	03	17	12	19	04	16	1e	03	00	1d	05	19	00	1d	02	0c	01	0a	15	03	05	0b	00
page	53:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page	54:	00	13	04	1e	0c	1e	05	0f	00	05	15	15	02	1d	19	0f	08	0a	0e	13	1a	07	0e	0a	13	1a	0d	09	1d	00	08	1b
page	55:	10	1a	0a	10	1e	09	0a	15	1a	19	0c	12	04	09	0a	0f	18	0d	09	05	06	0c	1b	18	0b	1a	1d	11	0c	03	00	1b
page	56:	0b	1a	12	18	1c	07	19	0f	1b	10	1a	0d	1c	1b	09	1a	18	0f	03	06	14	0c	1e	03	0f	17	09	04	15	10	15	13
page	57:	00	16	00	0d	1b	0b	02	1e	00	13	10	05	09	01	14	13	1a	0e	03	03	13	0e	1b	07	13	0c	05	14	02	05	14	13
page	58:	0c	16	1e	1b	15	0b	15	10	02	12	04	00	1e	18	13	0e	07	13	15	10	04	03	1e	0a	11	07	0d	0c	16	0c	09	1d
page	59:	1e	12	15	19	1a	05	1c	05	10	08	03	10	1b	1a	0d	1d	04	0f	11	04	03	1b	14	19	10	15	04	14	16	17	01	0f
page	60:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page	61:	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	d7	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f
page	62:	08	07	08	1c	0c	1d	16	15	0d	08	04	05	08	06	1a	1b	14	0c	09	1a	10	0c	00	07	17	0d	19	09	05	09	1b	12
page	63:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

PAGE DIRECTORY

PAGE TABLE ENTRY

OFFSET

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
page 64:	7f	7f	7f	7f	7f	7f	81	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f
page 65:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page 66:	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	91	7f	7f	7f	7f	7f	7f	7f	7f	7f	8c	7f	7f	7f	
page 67:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page 68:	7f	7f	89	7f	7f	7f	ba	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	ab	f0	
page 69:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page 70:	14	09	12	11	11	04	15	1c	16	05	15	08	12	10	1c	14	05	0c	19	08	03	09	1d	11	1a	17	13	1d	1c	13	1c	01
page 71:	1b	0f	0b	16	0d	01	15	11	16	1c	12	02	17	03	09	05	1c	1a	0f	15	05	0c	1d	10	02	07	1a	15	05	11	08	08
page 72:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page 73:	13	19	07	19	08	17	0a	1b	02	0b	07	05	01	14	14	04	1e	18	06	0d	1b	0e	16	04	0d	16	08	19	0a	1a	1a	0c
page 74:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page 75:	7f	7f	7f	7f	7f	8f	b9	7f	7f	7f	7f	7f	9c	7f	f6	8e	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f
page 76:	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	d8	7f	7f	7f	7f	7f
page 77:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page 78:	12	1e	09	19	1a	1a	08	04	06	02	11	03	09	19	1b	0e	14	03	1c	1b	0f	0a	02	14	09	17	10	09	00	07	0c	0c
page 79:	06	00	13	03	1c	0b	03	18	0a	0d	0f	04	06	0e	12	02	03	10	1c	08	1a	1b	11	04	05	1c	00	07	00	1c	12	1a
page 80:	11	0c	1a	02	10	02	06	01	0a	07	1e	00	1a	08	1e	1b	1c	0b	05	0b	00	1a	11	07	08	1e	19	10	01	10	01	17
page 81:	7f	7f	7f	7f	7f	7f	7f	df	7f	ec	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f
page 82:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page 83:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page 84:	05	10	08	1d	13	0f	02	15	0d	0a	04	02	05	06	19	00	0c	1c	1c	06	02	1b	02	19	0f	00	00	0e	08	03	09	05
page 85:	0c	0d	08	04	18	0e	0c	18	06	15	15	18	0c	17	13	12	06	1c	15	1a	07	11	0e	06	1c	12	12	09	17	15	14	10
page 86:	7f	7f	7f	7f	7f	7f	7f	7f	7f	9d	7f	7f	7f	7f	7f	ea	7f	7f	7f	7f	7f	b3	7f	7f	f5	7f	7f	7f	7f	7f	7f	db
page 87:	08	0d	13	0c	10	1d	12	0f	17	10	03	0d	0a	08	13	03	00	03	00	15	1b	13	03	0b	1b	1d	0a	18	16	07	14	10
page 88:	14	13	1e	10	08	0d	18	15	0a	06	0b	03	01	1b	09	09	1e	0a	1e	10	18	01	09	09	1a	19	06	10	01	17	0a	07
page 89:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page 90:	09	1b	1b	08	1c	00	00	0c	16	0e	0e	1c	12	0f	08	17	19	01	06	11	13	08	05	02	00	0f	0d	14	00	0c	12	01
page 91:	0f	05	1e	18	0b	0b	0d	1b	0d	05	19	0b	14	16	19	04	18	10	0f	01	03	12	04	0f	01	14	01	19	08	1a	1a	0f
page 92:	07	10	04	0b	0b	04	12	05	02	0f	18	0a	0a	0f	14	11	0c	02	1c	05	07	11	08	16	17	04	0e	17	04	1a	01	05
page 93:	17	1e	02	01	17	1b	18	1b	13	04	10	01	18	0e	18	07	19	1e	19	18	13	18	19	0f	0a	11	13	18	0c	08	1e	16
page 94:	12	13	00	1d	01	15	02	01	1d	10	0e	13	1d	17	11	09	19	12	02	09	03	11	0f	14	0c	04	0a	11	13	1a	18	0b
page 95:	09	0b	10	0c	0f	02	16	02	0d	15	04	0a	13	09	10	16	0c	01	00	03	0c	1b	12	18	19	08	0f	07	0b	0e	0b	00
page 96:	06	11	0d	18	04	0d	18	0c	07	07	03	05	0d	0d	1e	06	16	0d	15	0a	11	0a	1d	15	04	01	0c	08	0c	07	03	0e
page 97:	1b	17	0f	1e	18	0b	0e	0a	07	0f	01	01	13	0c	01	1c	10	14	10	05	08	06	15	17	0d	00	01	09	0f	10	19	04
page 98:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page 99:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page 100:	7f	da	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	f3	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f
page 101:	05	11	14	12	08	0a	13	01	14	0a	13	06	06	1e	0e	0a	0f	06	10	18	0a	11	0b	11	0b	0d	14	10	1d	0c	1c	0b
page 102:	16	00	0b	06	08	0b	0d	18	03	0f	1b	10	19	1d	11	1a	09	06	11	0e	0a	1d	14	0c	0a	05	15	0b	08	1e	14	0a
page 103:	11	13	09	0e	03	11	0a	0b	03	01	12	00	0a	0c	0e	06	03	12	1e	16	0d	17	09	07	12	12	0c	10	05	13	0d	12
page 104:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page 105:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page 106:	03	0d	17	1b	01	06	1b	17	09	07	0f	0a	13	0e	05	0c	05	05	19	14	11	06	04	06	0b	00	0d	05	1a	15	0a	0c
page 107:	10	1c	15	16	1c	0b	16	0d	07	0f	13	1d	0e	08	1d	19	1b	02	17	0f	04	07	1d	0b	10	1a	05	12	03	13	16	16
page 108:	10	03	0f	1d	08	1d	11	0d	10	19	11	0e	07	19	15	0e	1a	15	06	18	11	12	16	14	0b	1c	1e	08	0c	1a	14	03
page 109:	0f	1e	03	07	17	00	1e	1d	04	05	07	15	01	09	1e	17	0e	06	18	00	0d	0e	01	13	1c	0e	00	00	1a	03	0a	09
page 110:	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	cf	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	c9	7f	7f	7f	7f	7f	d5
page 111:	02	0c	0e	01	03	14	08	02	05	11	16	13	02	0f	14	16	0b	19	1a	06	09	08	13	04	1b	13	0b	0a	02	02	09	11
page 112:	1d	0a	06	19	13	03	12	17	08	14	02	1d	0e	10	0a	17	00	06	06	07	00	0c	12	15	02	11	0c	1d	1c	1d	16	10
page 113:	1d	10	14	09	13	1d	00	0d	06	0f	1e	07	12	16	13	12	0b	14	02	01	14	01	0c	16	0d	17	1e	13	0a	02	04	18
page 114:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page 115:	04	01	02	02	04	13	0f	00	07	12	09	18	08	1d	16	1b	1b	1e	11	1b	07	0d	1e	08	17	02	09	01	04	0a	09	10
page 116:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page 117:	17	02	16	1a	08	07	19	16	19	17	17	0b	0c	05	09	04	0d	05	05	1a	0c	1d	0a	03	1c	01	1b	15	03	11	10	00
page 118:	0e	14	07	0a	1c	0b	11	19	11	10	0b	10	1a	00	0e	14	02	11	15	00	0a	09	04	09	17	1d	09	15	14	1c	15	13
page 119:	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	d0	7f	7f	7f	7f	7f	7f	7f
page 120:	1e	00	13	1d	1d	01	02	11	15	1d	0b	16	14	1c	18	0f	07	03	15	13	1c	0f	17	15	12	04	1a	00	08	10	00	09
page 121:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
page 122:	1e	00	06	1c	0a	14	12	17	03	09	0c	0b	0f	0f	12	08	07	06	0b	02	1c	1e	09	12	11	0a	04	13	0b	06	16	12
page 123:	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	97	7f	7f	7f	7f	7f	7f	7f	7f
page 124:	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	a2	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	ce	7f	7f	7f	7f
page 125:	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	9a	7f	7f	7f	7f	7f	fa	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f
page 126:	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	dc	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	7f	ed	7f	7f	7f	7f	7f	7f	7f	7f	7f
page 127:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

PHYSICAL FRAME

7. (10 points) The following question traces TLB behavior over time. Each question will give you a few assumptions; you should then produce a series of hits ("H") and misses ("m"). The string "mHHm" would mean a "TLB miss" followed by two "TLB Hits", followed by a "TLB miss". You can assume that instruction references do not affect the TLB and that the array (discussed below) is paged aligned (i.e., it starts that the beginning of the page). If the pattern of hits and misses repeats, you can just write out the pattern and mention that it repeats.

a. Assume you have a 1-entry TLB. Assume you access contiguous 4-byte integers in a large array, starting at index 0 and going to some large number. Assume the page size is 32-bytes. What is the hit/miss pattern for this access?

[mHHHHHHH] repeated until done

32 byte Page Size / 4 bytes Per INT → 8 INTEGERS Per PAGE

b. Assume you have a 2-entry (fully associative) TLB with LRU replacement of TLB entries. Assume you access contiguous 4-byte integers in a large array, again starting at 0. Assume, like above, the page size is 32-bytes. What is the hit/miss pattern for this access?

[mHHHHHHH] repeated until done (same as above)

c. Assume you have a 1-entry TLB. Assume you access **every other** 4-byte integer in contiguous array, starting at index 0, then index 2, 4, 6, and so on (i.e., a stride-2 access pattern). Assume the page size is 32-bytes. What is the hit/miss pattern for this access?

[mHHH] repeated until done

d. Assume you have a 8-entry (fully associative) TLB with FIFO replacement. Assume you repeatedly access all 4-byte integers in a small contiguous array of 24 integers in a loop. Again, assume the page size is 32-bytes. What is the hit/miss pattern for the **first** run of the loop?

[mHHHHHHH] repeated 3 times

8 ints Per Page
3 PAGES TOTAL

e. For the above problem (d), what is the hitt/miss pattern for the **second** run of the loop?

[H] repeated 24 times, all hits

ALL PAGES ARE IN THE TLB

8. (24 points) Multiple Choice and short answer

(a) In a Limited Direct Execution environment, **circle all the ways** the OS can regain control of the system.

(i) The program volunteers to stop running with a call to yield()

(ii) A hardware interrupt occurs

(iii) A program issues a syscall

(iv) The program executes an illegal instruction

(b) Which of the following happens during a system call (syscall)? **Circle all that apply.**

(i) The registers of the currently running process are saved

(ii) The memory the calling process is swapped in from disk

(iii) The OS moves the currently running process to the BLOCKED state

(iv) The TLB is invalidated

(c) Which of the following are **mechanisms**? **Circle all that apply.**

(i) A context switch

(ii) Using LRU for selecting a page to evict

(iii) Base and bounds registers for dynamic relocation

(iv) Splitting and Coalescing free blocks

(d) Which of the following can lead to more **internal** fragmentation? **Circle all that apply.**

(i) Having a larger page size (e.g., 2 meg)

(ii) Using paging versus using segments

(iii) Having a smaller page size (e.g., 512 bytes)

(iv) Using the worst fit algorithm and splitting blocks

For the following two questions, you should be able to give your answer only a few short sentences.

(e) In Linux, creating a process is a two step process, using `fork()` followed by `exec()`. What is the advantage of doing it in this way?

It allows the shell to easily implement file redirection and pipes by changing what STDIN, STDOUT, and STDERR refer to.

(f) The VMS virtual memory implementation was a hybrid system, using both segments and pages. Briefly describe how that system worked and how it was used to reduce the size of the page tables.

The system had a segment for the kernel, and two user segments, one for the stack, and one for everything else (code, data, heap). Each segment had a base and bounds registers. The base contained the start of each page table, and the bounds had the number of page table entries in the segment. Unallocated memory past the end of the segment didn't need page table entries.