HW08: Virtual Memory
CSE 30321 Computer Architecture
Names: Aaron Wang and Ethan Little

Preamble:

1. ***Copy this assignment from our Google Drive directory, not from Gradescope. Downloading the PDF from Gradescope may not preserve all answer boxes.***

2. ***Reminder:*** point totals differ across assignments mainly as a way to allow for partial credit, *but all assignments carry the same weight* (see the syllabus for details).

3. Enter your answers in the boxes provided. Each empty box should be filled in with an answer. You can either type your answers directly or insert images as long as they are legible. For solutions that require code, use a **fixed-width font (Consolas preferred) no smaller than 10 points with proper indentation**. Save your solutions as a single PDF file and upload them to Gradescope.

4. You are encouraged to work in groups of 2. Please type your names at the top of this document.

5. "Dad, this homework does not make any sense" - Michael D. 🙂

Problem 1: VPNs, PFNs, Offsets - Oh My!
Question 1

Assume a machine has a 40-bit physical address space, a 50-bit virtual address space, and an 8 KiB page size. Determine the size of the virtual page number (VPN), physical frame number (PFN), and page offset. **Show your work.**

Number of bits for VPN: 50 - 13 = 37
Number of bits for PFN: 40 - 13 = 27
Number of bits for offset: $\log_2(2^3 * 2^{10}) = \log_2(2^{13}) = 13$

In addition, fill out the bit ranges for the virtual and physical addresses.
Virtual Address:

Bit Range:	49	13	12	0
Field	Virtual Page Number (VPN)		Offset	

Physical Address:

Bit Range:	39	13	12	0
Field	Physical Frame Number (PFN)		Offset	

Question 2

Assume a certain byte-addressable machine has a 52-bit physical address space, and a 57-bit virtual address space, as is the case on recent Intel x86 systems. x86 supports multiple page sizes in the hardware, including 4KiB and 2 MiB, and possibly 1 GiB pages. 2 MiB is the default "huge page" size. Answer the following questions assuming the operating system is using 2 MiB "huge pages" In addition, assume the hardware has support for a small, fully associative TLB with 8 entries. **Show your work/explain your reasoning for each sub-question.**

a. How many virtual pages are in the virtual address space?

2 MiB = 2 * 2^{20} = 2^{21} so 21 is the offset. Thus there are 57 - 21 = 36 bits for the virtual page address and as such 2^{36} virtual pages.

b. How many physical frames are in the physical address space?

2 MiB = 2 * 2^{20} = 2^{21} so 21 is the offset. Thus there are 52 - 21 = 31 bits for the physical frame address and as such 2^{31} physical frames.

c. How many independent virtual-to-physical translations can a **single** TLB entry support? Reminder: this is a byte-addressable machine.

One single TLB entry can support the number of bytes in a single page: 2 MiB = 2 * 2^{20} = 2^{21} translations

Problem 2: Translation "Simulation"

Question 1

Assume a byte addressable machine with a 4-entry TLB. The machine uses a 20-bit virtual and an 18-bit physical address. The machine/OS are using a 64 KiB page size. A single process, Process X, is running on the CPU.

Show how the state of the TLB and page table change for the list of accesses in the table below.

Some requirements, read carefully:

a. Show all work.
   i. In the **first table**, include a short summary for each access indicating the translation critical path for each access. In class on 3/25, for example, we had an access which could be summarized as: "Lookup VPN (tag) in TLB → miss; lookup in page table → translation found in page table; update TLB (invalid way

available); PA = 00961AF". This is the level of detail you should include - the more the better.

ii. In the **TLB**, update the (true) LRU state. Keep past history for each row, as we did in our in-class examples.
    1. If more than 1 invalid way is available in the TLB, always fill the top-most invalid TLB way.
    2. Suggestion: in Question 2, you will fill the TLB for pseudo-LRU for this same access pattern, so it might help to think about how PLRU behaves (and/or complete Question 2 in parallel).

iii. In the **page table**, make sure to update the PFN column and valid bit as needed.

b. If the page table indicates a virtual page currently maps to a location on the hard drive ("On disk"), it will need to be swapped in. The operating system allocates physical frames sequentially starting from the next available physical frame number. Currently, all physical frames are **in use**. Assume the LRU physical frame is PFN 0, and the next LRU is PFN 1, followed by PFN 2, and finally PFN 3 (the MRU frame). The OS uses a true LRU policy to decide which frame to send to disk on a swap.
    i. Keep track of the LRU ordering for the physical frames in the "Summary" for each access as is done in the header.

c. Virtual pages that are not allocated are indicated as such – there is no corresponding physical frame in physical memory or on disk.

d. The TLB uses an LRU replacement policy.
LRU = 0 means the translation is LRU; LRU = 3 means the translation is MRU.

e. All VPNs and PFNs are in hex, but the 0x is omitted – you may do the same when filling the tables to save space.

Access (i.e., Address Referenced, assume loads)	Summary Initial physical frame LRU ordering: (LRU) 0, 1, 2, 3 (MRU)
0x24601	Lookup VPN 2 in TLB → miss; lookup in page table → Page Fault; OS replaces PFN 0; lookup VPN 2 in TLB → miss; lookup in page table → PFN 0; add translation to TLB → PA = 04601 physical frame LRU ordering: (LRU) 1, 2, 3, 0 (MRU)
0x2FEED	Lookup VPN 2 in TLB → hit → PA = 0FEED physical frame LRU ordering: (LRU) 1, 2, 3, 0 (MRU)

0x98765	Lookup VPN 9 in TLB → miss; lookup in page table → Page Fault; OS replaces PFN 1; lookup VPN 9 in TLB → miss; lookup in page table → PFN 1; add translation to TLB → PA = 18765 physical frame LRU ordering: (LRU) 2, 3, 0, 1 (MRU)
0x18151	Lookup VPN 1 in TLB → miss; lookup in page table → PFN 3; add translation to TLB → PA = 38151 physical frame LRU ordering: (LRU) 2, 0, 1, 3 (MRU)
0x78900	Lookup VPN 7 in TLB → miss; lookup in page table → PFN 2; add translation to TLB → PA = 28900 physical frame LRU ordering: (LRU) 0, 1, 3, 2 (MRU)
0x86753	Lookup VPN 8 in TLB → miss; lookup in page table → Page Fault; OS replaces PFN 0; lookup VPN 8 in TLB → miss; lookup in page table → PFN 0; add translation to TLB → PA = 06753 physical frame LRU ordering: (LRU) 1, 3, 2, 0 (MRU)
0x241AE	Lookup VPN 2 in TLB → miss; lookup in page table → Page Fault; OS replaces PFN 1; lookup VPN 2 in TLB → miss;lookup in page table → PFN 1; add translation to TLB → PA = 141AE physical frame LRU ordering: (LRU) 3, 2, 0, 1 (MRU)
0xFADED	Lookup VPN F in TLB → miss; lookup in page table → Page fault; OS finds no valid translation on disk → seg fault physical frame LRU ordering: (LRU) 3, 2, 0, 1 (MRU)

TLB:

Valid	LRU	VPN	PFN
0 1	0 3 3 2 1 0 3 2	2 8	0
0 1	1 0 0 3 2 1 0 3	9 2	1
0 1	2 1 1 0 3 2 1 0	1	3
0 1	3 2 2 1 0 3 2 1	7	2

Page Table for Process X

VPN	PFN	Valid
0	On disk	0
1	3	1
2	~~On disk~~ ~~0~~ ~~On disk~~ 1	0 1 0 1
3	On disk	0
4	~~1~~ On disk	1 0
5	Not allocated	0
6	Not allocated	0
7	2	1
8	~~On disk~~ 0	0 1
9	~~On disk~~ ~~1~~ On disk	0 1 0
A	Not allocated	0
B	~~0~~ On disk	1 0
C	Not allocated	0
D	Not allocated	0
E	Not allocated	0
F	Not allocated	0

Question 2

Indicate the TLB state for the same access pattern from Problem 1 for pseudo-LRU. You do not need to show the state of the other tables. Again, show the entire history by crossing out old values in the replacement policy column.

- For pseudo-LRU:
  - Indicate which direction the PLRU bit points, and the history. 0 means “look up” and 1 means “look down.”
  - The initial state is shown below.
  - Make sure to update any valid bits, PLRU bits, VPNs and PFNs that change on each access.

`TLB for PLRU:`

Valid	PLRU		VPN	PFN
0 1	0 1 0 1 0 1 0	0 1 0 1	2 8	0
0 1			1	3
0 1		0 1 0 1	9 2	1
0 1			7	2

Problem 3: Virtual Memory on a Real System

In this last problem, you will do a little research to investigate the virtual and physical memory characteristics of `student13.cse.nd.edu`. Answer the following questions and make sure for each to 1) cite any sources used for your research; 2) include screenshots showing where you found any information you gathered from the machine itself; 3) explain your reasoning.

a. How much physical memory (RAM) exists on the system?

https://www.baeldung.com/linux/proc-meminfo

```
awang27@student13:~$ cat /proc/meminfo
MemTotal:       32163068 kB
```

When you do cat /proc/meminfo, MemTotal says 32163068 kB.

b. The physical pages that are not resident in RAM will be on the hard drive (secondary storage) in a “swap file.” How much space is set aside for the swap file, how much is currently in use, and where is the swap located? Tip: look in `man proc` for information on “swap”

```
awang27@student13:~$ cat /proc/meminfo | grep "Swap"
SwapCached:          126168 kB
SwapTotal:         67108856 kB
SwapFree:          64077952 kB
```

```
NAME      TYPE      SIZE   USED PRIO
/dev/sda3 partition  32G   3.2G   -2
/dev/sdb3 partition  32G 332.1M   -3
```

When you do cat /proc/meminfo, SwapTotal says 67108856 kB and SwapFree says 64077952 kB.
The file swap files are both in /dev and are called sda3 and sdb3

c. How many bits are the virtual and physical addresses? Tip: look at `/proc/cpuinfo`

```
awang27@student13:~$ cat /proc/cpuinfo | grep "address"
address sizes   : 46 bits physical, 48 bits virtual
```

Looking at proc/cpuinfo we see that
physical address bits: 46
virtual address bits: 48

Run the mem_test.c program we used in class. Hit any key to force the program to display its PID and other information. Inspect the contents of the process’s mapped virtual memory by executing `cat /proc/PID/maps` (replace PID with the process ID reported by the program).

d. What ranges of virtual memory are allocated for the process’s heap and stack? Is the array at the very start of the heap; if not, how far into the heap’s address space does the array start?

01cd1000-01cf2000 rw-p 00000000 00:00 0 [heap]
7ffea2bc6000-7ffea2be8000 rw-p 00000000 00:00 0

```
awang27@student13:~$ ./mem_test

I am process 2552632, my array starts at 0x22102a0 (dec addr 35717792), arr[0] is 2552632
```

```
awang27@student13:~$ cat /proc/2552632/maps | grep "\(heap\)\|\(stack\)"
02210000-02231000 rw-p 00000000 00:00 0                                  [heap]
7ffe18b2e000-7ffe18b50000 rw-p 00000000 00:00 0                          [stack]
```

[stack]
Man proc shows that the first column is the address ranges. So we know where the heap starts

I am process 3638403, my array starts at 0x22102a0 (dec addr 35717792), arr[0] is 2552632
The array starts 0x2a0 after the heap allocation start which is given by the first number in the first row.

e. What page size is in use for the process? Tip: look instead at `/proc/PID/smaps` (and look in man proc for details on `smaps`)

```
awang27@student13:~$ cat /proc/3638403/smaps | head
00400000-00401000 r-xp 00000000 00:36 9280772224
Size:                  4 kB
KernelPageSize:        4 kB
MMUPageSize:           4 kB
```

according to maps and the man proc, the page size is 4kB

What to Turn In

Fill in the boxes on the previous pages with answers to all the questions. Save your Google Doc as a PDF and upload it to **Gradescope** for grading. Note Gradescope can be accessed via our Canvas site, or by visiting gradescope.com. Below is a checklist for this assignment:

Problem 1 (20 points)

	Deliverable	Points
1.	Question 1	10
2.	Question 2	10

Problem 2 (50 points)

	Deliverable	Points
1.	Question 1	35
2.	Question 2	15

Problem 3 (15 points)

	Deliverable	Points
1.	3 points each question	15