ANSWERS

CMS330 – Spr 2019 **Project 4** **Due: 4/8/19 by 9pm to me!**

The following problems cover virtual memory. For all problems, show your work! Answers without work will not receive full credit.

PROBLEM 1: (30pts) Consider a system with a 64 byte virtual address space, 16 byte pages, and 128 bytes of physical memory.

2 a) How many bits are required to encode each of the 64 virtual addresses? 6 bits How about the 128 physical addresses? 7bits

4 b) Show how to divide each of the following virtual addresses into a virtual page number (VPN) and offset:

i. 7 00 0111
ii. 12 01 1100
iii. 21 01 0101
iv. 50 11 0010

4 c) Each page table entry has the following format:

```
| 1 valid bit | 3 PFN bits |
```

Why are there 3 PFN bits? 8 pgs in pys mem. $2^x=8$ $x=3$ bits for PFN

8 d) The page table has four entries. For each entry, indicate if it is valid or not and give the associated PFN for each virtual page

Page table entry (in hex)	Valid?	PFN
0xA	✓	010
0xF	✓	111
0x0		— (000)
0xC	✓	100

12 e) Translate the following virtual addresses their corresponding physical addresses

i. 7 010 0111
ii. 12 010 1100
iii. 50 100 0010
iv. 36 invalid/seg fault

PROBLEM 2: (30pts) For this example, assume a system with an 8K virtual address space, a page size of 1K, and 64K of physical memory.

Suppose we want to execute the following instruction, which is stored at VIRTUAL ADDRESS 4096 (4K)

```
load 2048, r1
```

4 a) List the VIRTUAL ADDRESSES generated by this instruction. 2048, 4096

4 b) Show how to divide the address into its virtual page number (VPN) and offset. 3 bits for VPN, 10 for offset

4 c) How many bits are required to encode each:

i. virtual address? 13 bits
ii. physical address? 16 bits

4 d) Suppose the system is equipped with a hardware-managed linear page table. The page table base register (PTBR) is set to 32K. Each entry is a single byte and has the following format:

```
| 1 valid bit | 6 PFN bits | 1 write-protected bit |
```

Why are there 6 PFN bits in each page table entry?

14 e) The contents of the page table are: 64 pages of phys. memory
$2^x=64$
$x=6$
6 bits for PFN

000	0x00
001	0x00
010	0xA8
011	0x00
100	0x8C
101	0x00
110	0x00
111	0x00

page table lookup 0x8004
fetch instr 0x[illegible]800
page table lookup 0x8002
load data 0x[illegible]000
5

List all physical memory accesses that are performed when

```
load 2048, r1
```

executes. *Remember that page table lookups also access physical memory!*

PROBLEM 3: (20pts) Label each of the following statements as either True or False AND explain your reasoning.

a) A smaller page size leads to smaller page tables. F. smaller pgs → more pages → more entries

b) A smaller page size leads to more TLB misses. T. smaller pgs → each contains less ~~data~~ addr less likley to have data we need. TLB is fixed size!

c) A smaller page size leads to fewer page faults. T. smaller pgs → more pgs in mem

~~d) A smaller page size reduces paging I/O throughput.~~

PROBLEM 4: (20pts) In this exercise, we consider a simple machine with a MMU that implements virtual memory based on segmentation. The main specifications of this machine are as follows:

- The MMU hardware has two pairs of (base, bounds/limit) registers (i.e., a process can at most have two segments).
- Virtual addresses (including the explicit segment ID) are stored in 10 bits and physical addresses are stored in 16 bits.
- The machine is equipped with a capacity of 16 kB of physical memory (RAM).

Now let's consider a process with two segments, for which the MMU configuration is as follows:

- segment 0:
  - base = 0x8400
  - limit (size) = 0x100 bytes
- segment 1:
  - base = 0x0c00
  - limit (size) = 0 bytes

Unfortunately, it appears that the above MMU configuration is incorrect because the values written in some of the MMU registers have been corrupted due to some hardware defects. More precisely, the wrong values have exactly one flipped bit: i.e., compared to the originally written (good) value, there is exactly one bit that has been accidentally modified (from 0 to 1 or vice-versa).

Let us assume that we also know (for sure!) that:

- virtual address 0x300 should cause a segmentation violation (invalid address)
- virtual address 0x2ff is a valid address

7 a) Which of the segmentation registers (among seg. 0 base, seg. 0 limit, seg. 1 base, seg. 1 limit) determines whether these two addresses are valid or not? Explain your answer. b/c both address map to seg 1 but neither could be valid b/c of limit=0.

7 b) What should the correct value be for the register identified in your previous answer? Explain your answer. 0x100

6 c) Given the correct value identified in the previous answer (and if there is no other error for this segment), what physical address should virtual address 0x2ff translate to and why?

0[illegible]0 1111 1111 seg1

0x 0c00 + 0x ff (offset) = 0x cff

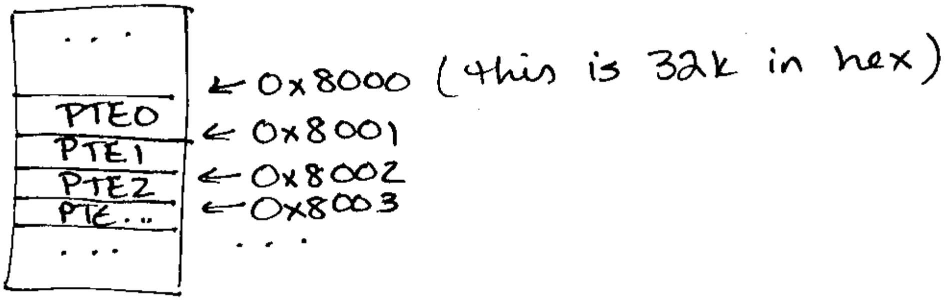
0x0c00 + 100 = 0x0d00

0x cff < 0x0d00 → valid!

2e. Remember that page table accesses are also physical mem. accesses! So there are 4 phys. mem. accesses for this one instr:

1) page table lookup for instr. addr
2) instr fetch
3) page table lookup for data addr.
4) data load

Part d told us the PTBR holds a value of 32k, which is where the page table starts. Each PTE is 8 bits/1 byte (part d again). So, the Page table in memory would be:



the instr is @ addr. 4096 (virt addr.) or

$0b\ \underbrace{100}_{VPN}\ \underbrace{0000000000}_{offset}$. This means we will access PTE4 which will be at phys. addr. 0x8004.

Once we decode VPN of 4_{10} (100_b) to PTE 0x8C we have to extract the PFN $\underset{valid}{1}\underbrace{000\ 110}_{PFN}\underset{write}{0}$.

Our 2nd physical mem. access will be at PFN + offset ~~0~~001 1000 0000 0000 = 0x1800.

You can repeat this decoding process for the data load at virt. addr. 2048.

Prob 3.

a. False. smaller pages mean more pages, and thus more page table entries. Since the page table can grow, more PTEs means a bigger PTE.

b. True. Think about an extreme: what if we have 1 very large page (say, the size of phys. mem.) All mem. accesses would hit in the TLB! Now imagine we have 2 pages, each equal to ½ phys. mem. As long as our TLB has room for 2 translations, we will still have 100% hits. However a TLB is a <u>fixed size</u>. It won't grow! So as our page size gets smaller and smaller, the chance of hitting in the TLB reduces (ie there are more + more misses).

c. Left as exercise to reader.

Prob 4

We know that virtual addr. are 10 bits and these include a segment id. Since there are only 2 segments, we only need 1 bit to rep. the segment: 0 for seg0 and 1 for seg 1. The two virtual addresses both map to segment 1:

0x300 = 1|1 0000 0000
0x2ff = 1|0 1111 1111

segment 1!

You treat the other 9 bits as the "offset" to be added to seg1's base value. Since 0x2ff should be valid, we can conclude there is an error in seg1's bounds/limit reg. (a)

Since 2ff is valid and 300 is not, we can conclude that 2ff should be between the base and the (base + bounds) while 300 is not. Notice above that if we add 1 to 2ff, we get 300. Therefore we know exactly where the bounds should be:

0x0c00 (seg1 base)
$\underline{\quad\quad ff}$ (2ff's offset)
0x0cff (valid)
(c)

0x0c00 (seg1 base)
$\underline{\quad\quad 100}$ (300's offset)
0x0d00 (invalid!)

↑ base + bounds must be here!

Therefore the bounds reg. should have the value 0x100 (b) in it. This conforms to the problem limitation that exactly 1 bit was flipped.