1.

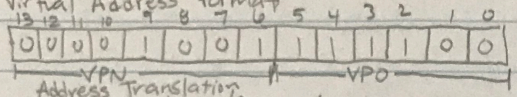
Jonathan Armknecht
Virtual Memory Assignment

- The memory is byte addressable
- Memory accesses are to 1-byte words
- Virtual addresses are 14 bits wide (n=14)
- Physical addresses are 12 bits wide (m=12)
- Page size is 64 bytes (P=64)
- TLB is 4-way set associative w/ 16 entries
- L1 d-cache is physically addressed & direct mapped w/ a 4-byte line size & 16 sets

9.11

VA: 0x027c

A. Virtual Address format



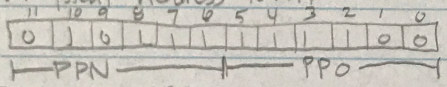
B. VPO is $64 = 2^6$ so 6-bits

VPN is 8 bits

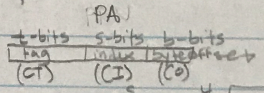
TLB has four sets so $2^2 = 4$ thus 2-bits for the TLBI and 6-bits for the TLBT

Parameter	Value
VPN	0x09
TLB index	0x01
TLB tag	0x02
TLB hit? (y/n)	no
Page fault? (y/n)	no (in Page Table)
PPN	0x17

C. Physical Address Format



PA = 0x5FC



D. Physical memory reference

S is number of L1 d-cache sets $16 = 2^s = 2^4$ $\boxed{s=4}$

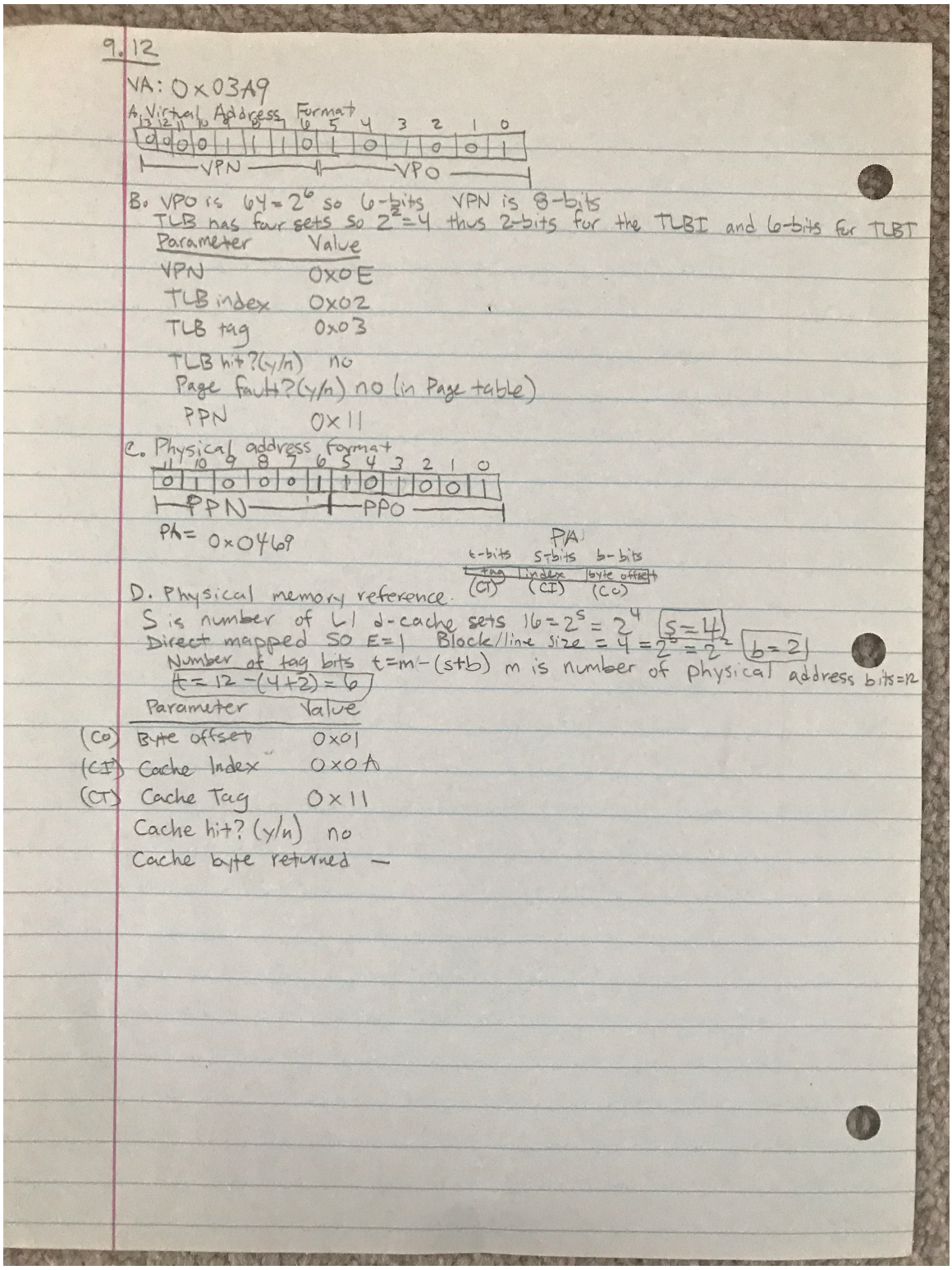
Direct mapped so E=1 Block/line size $= 4 = 2^b = 2^2 \Rightarrow \boxed{b=2}$

Number of tag bits $t = m - (s+b)$ m is number of physical address bits = 12

$\boxed{t = 12 - (4+2) = 6}$

Parameter	Value
(CO) Byte offset	0x0
(CI) Cache index	0x0F
(CT) Cache Tag	0x47
Cache hit? (y/n)	no
Cache byte returned	—

2.



9.12

VA: 0x03A9

A. Virtual Address Format

B. VPO is $64 = 2^6$ so 6-bits VPN is 8-bits
TLB has four sets so $2^2 = 4$ thus 2-bits for the TLBI and 6-bits for TLBT

Parameter	Value
VPN	0x0E
TLB index	0x02
TLB tag	0x03
TLB hit? (y/n)	no
Page fault? (y/n)	no (in Page table)
PPN	0x11

C. Physical address format

PA = 0x0469

PA
t-bits s-bits b-bits
tag index byte offset
(CT) (CI) (CO)

D. Physical memory reference.
S is number of L1 d-cache sets $16 = 2^5 = 2^4$ $s = 4$
Direct mapped so E=1 Block/line size = 4 = $2^2 = 2^2$ $b = 2$
Number of tag bits $t = m - (s+b)$ m is number of physical address bits = 12
$t = 12 - (4+2) = 6$

	Parameter	Value
(CO)	Byte offset	0x01
(CI)	Cache Index	0x0A
(CT)	Cache Tag	0x11
	Cache hit? (y/n)	no
	Cache byte returned	—

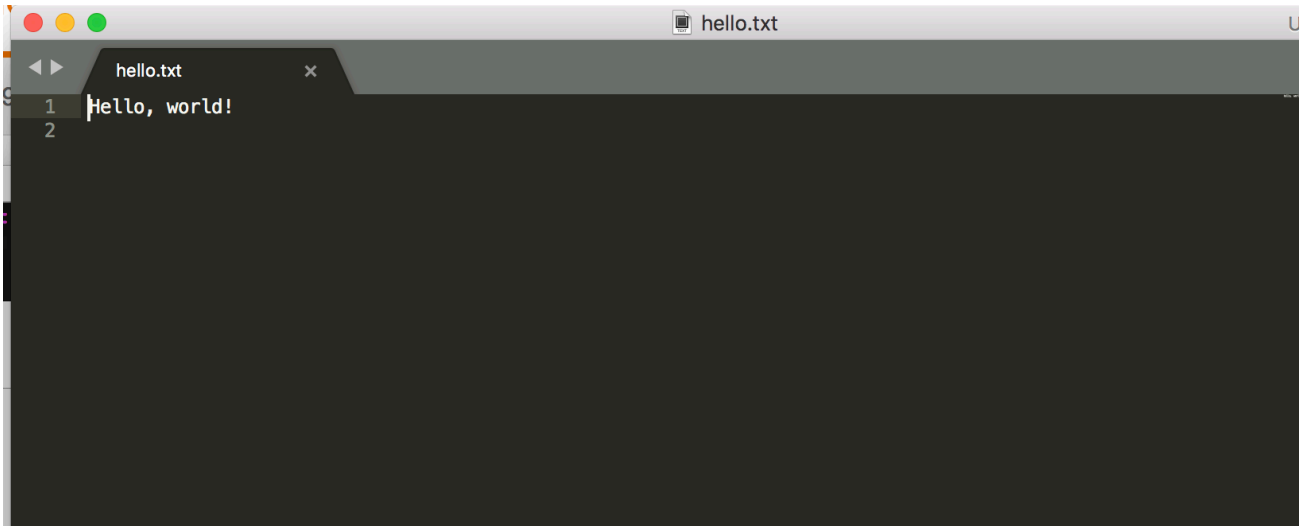
3.

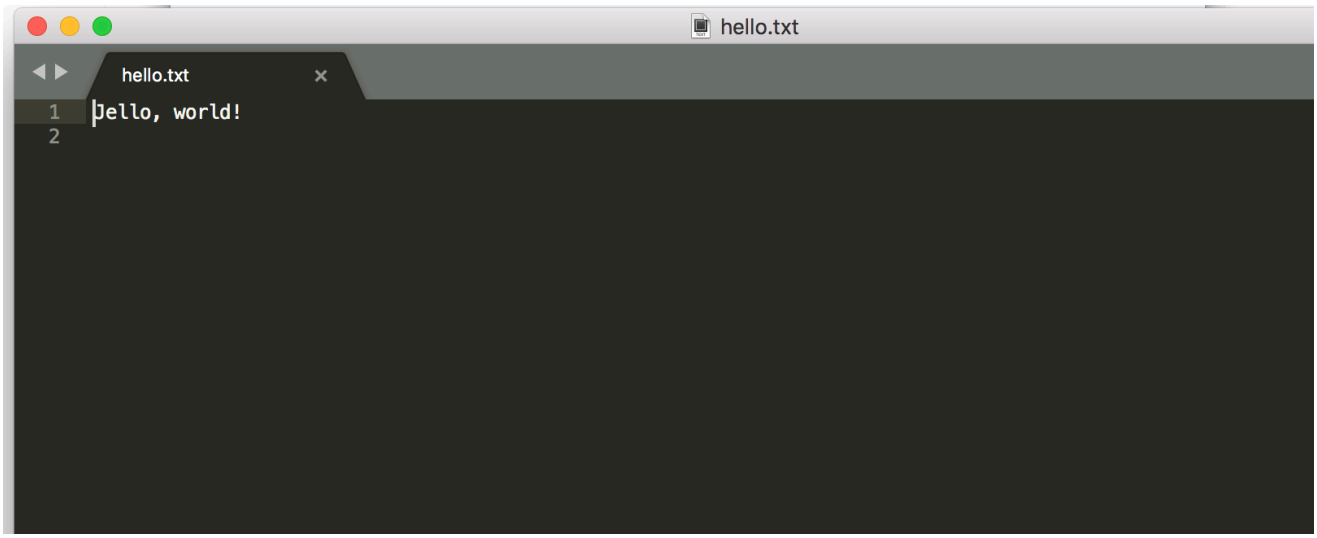
```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
```

```
#include <unistd.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/types.h>

int main (int argc, char* argv[]) {
	int fd; //Holds file descriptor
	struct stat sb; //Holds stats of the file
	char *buf; //Holds what mmap returns pointer to the mapped region
	size_t size; //Holds the offset size of the file from the stats struct


	fd = open("hello.txt", O_RDWR, NULL); //Opens hello.txt for reading and
writing
	if (fd < 0) { //if fd is < 0 open failed
		fprintf(stderr, "Could not open file\n");
		exit(1);
	}
	if (fstat(fd, &sb) < 0) { //Calls fstat which returns info about a file and stores
that info in sb
		//if fstat failed -1 will be returned
		fprintf(stderr, "Could not fstat file\n");
		exit(1);
	}
	size = sb.st_size; //Gets the total size in bytes of the file descriptor fd

	/*
	* Tells kernel to create a new area of virtual memory where
	* pages can be written and read and it is a shared object between processes
containing
	* size bytes. Buf will hold the address of the new area created
	*/
	buf = mmap(NULL, size, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);

	if (buf < 0) { //Mmap failed!
		fprintf(stderr, "Could not mmap\n");
		exit(1);
	}

	printf("%s", buf); //Reads the virtual memory area and prints it to the
console
	buf[0] = 'J'; //Takes buf and writes J as the first letter in the buffer
	printf("%s", buf); //Reads the virtual memory area and prints it to the
console
```

```
        if (munmap(buf, size) < 0) { //unmaps the virtual memory area that is
pointed to by buf and consists of size bytes
                //if munmap fails it returns -1
                fprintf(stderr, "Could not unmap\n");
                exit (1);
        }

        if (fd != -1) { //a file needs to be closed
                if (close(fd) < 0) { //closes the text file
                        //closing the file failed
                        fprintf(stderr, "Could not close file\n");
                        exit(1);
                }
        }
        return 0;
}
```

```
[Jonathan: ~/Documents/Class Documents/Grad/Fall Semester 2018/CS324/VirtualMemAssignment $ ./mmap
Hello, world!
Jello, world!
```





4.

Full Name: Jonathan Armknecht

Section: 2

Problem 1. (25 points):

The following problem concerns the way virtual addresses are translated into physical addresses.

- The memory is byte addressable.
- Memory accesses are to 4-byte words.
- Virtual addresses are 20 bits wide.
- Physical addresses are 16 bits wide.
- The page size is 4096 bytes. 4 KB 2^{12} VPO is 12-bits
- The TLB is 4-way set associative with 16 total entries.

$4 = 2^t$ $t = 2$ TLBI = 2-bits TLBT = 6-bits

In the following tables, **all numbers are given in hexadecimal**. The contents of the TLB and the page table for the first 32 pages are as follows:

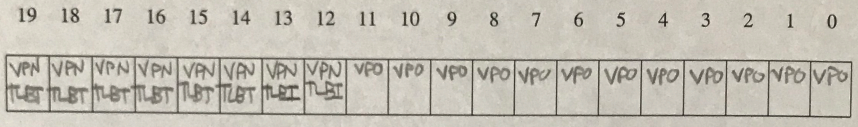
TLB			
Index	Tag	PPN	Valid
0	03	B	1
	07	6	0
	28	3	1
	01	F	0
1	31	0	1
	12	3	0
	07	E	1
	0B	1	1
2	2A	A	0
	11	1	0
	1F	8	1
	07	5	1
3	07	3	1
	3F	F	0
	10	D	0
	32	0	0

Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
00	7	1	10	6	0
01	8	1	11	7	0
02	9	1	12	8	0
03	A	1	13	3	0
04	6	0	14	D	0
05	3	0	15	B	0
06	1	0	16	9	0
07	8	0	17	6	0
08	2	0	18	C	1
09	3	0	19	4	1
0A	1	1	1A	F	0
0B	6	1	1B	2	1
0C	A	1	1C	0	0
0D	D	0	1D	E	1
0E	E	0	1E	5	1
0F	D	1	1F	3	1

A. Part 1

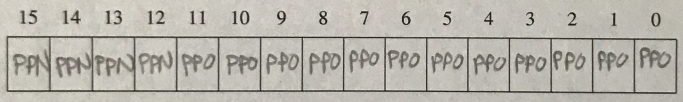
(a) The box below shows the format of a virtual address. Indicate (by labeling the diagram) the fields (if they exist) that would be used to determine the following: (If a field doesn't exist, don't draw it on the diagram.)

VPO The virtual page offset
VPN The virtual page number
TLBI The TLB index
TLBT The TLB tag



(b) The box below shows the format of a physical address. Indicate (by labeling the diagram) the fields that would be used to determine the following:

PPO The physical page offset =VPO 12-bits
PPN The physical page number 4-bits



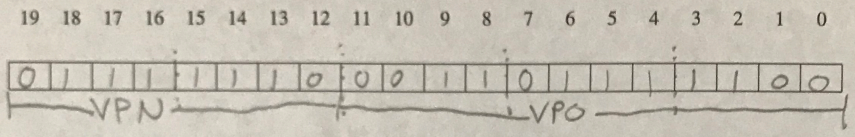
B. Part 2

For the given virtual addresses, indicate the TLB entry accessed and the physical address. Indicate whether the TLB misses and whether a page fault occurs.

If there is a page fault, enter "-" for "PPN" and leave part C blank.

Virtual address: 7E37C

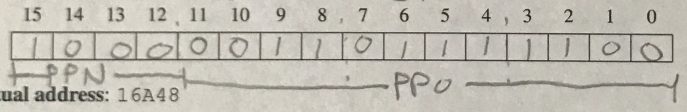
(a) Virtual address (one bit per box)



(b) Address translation

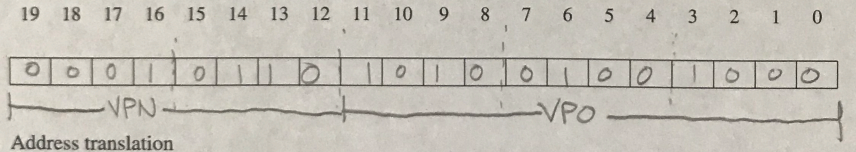
Parameter	Value
VPN	0x 7E
TLB Index	0x 02
TLB Tag	0x 1F
TLB Hit? (Y/N)	yes
Page Fault? (Y/N)	no
PPN	0x 08

(c) Physical address (one bit per box) 0x 837C



Virtual address: 16A48

(a) Virtual address (one bit per box)

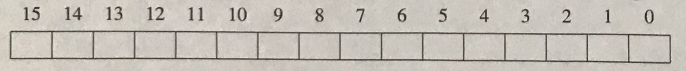


(b) Address translation

Parameter	Value
VPN	0x 16
TLB Index	0x 02
TLB Tag	0x 05
TLB Hit? (Y/N)	no
Page Fault? (Y/N)	yes
PPN	0x —

→ its not valid so page fault. It is allocated though so it is in swap space. The physical page number will change cause of this and will be anything.

(c) Physical address (one bit per box)



2^{32} = 32-bits 4KB = 2^{12} 2^2 = 4 bytes if not using multi-level page table would need $2^{32} * 2^{-12} * 2^2 = 2^{22}$ need a 4MB page table all the time

This problem deals with virtual memory address translation using a multi-level page table, in particular the 2-level page table for a 32-bit Intel system with 4 KByte page tables. Each page table entry is 4 bytes.

32-bits

$\frac{2^{12}\text{ KByte PT}}{\text{PTE } 2^2} = 2^{10}$ 10 page table entries

Specify how many bits are used for each of the following:

- Page offset = 12-bits 4KB = 2^{12} so VPO is 12-bits left over with 20-bits
- Offset into level-1 of the page table = 10-bits
- Offset into level-2 of the page table = 10-bits

VPN = 20 bit

Draw a diagram of how a memory lookup would be performed using the 2-level page table. You should include in your diagram the use of all three pieces of the virtual address to translate a virtual address to a physical address. Include the TLB in your diagram as well and indicate how it would be used to make the address lookup quicker.