Problem M4.1: Virtual Memory Bits

This problem requires the knowledge of Handout #6 (Virtual Memory Implementation) and Lecture 4 and 5. Please, read these materials before answering the following questions.

In this problem we consider simple virtual memory enhancements.

Problem M4.1.A

Whenever a TLB entry is replaced we write the entire entry back to the page table. Ben thinks this is a waste of memory bandwidth. He thinks only a few of the bits need to be written back. For each of the bits explain why or why not they need to be written back to the page table.

With this in mind, we will see how we can minimize the number of bits we actually need in each TLB entry throughout the rest of the problem.

Problem M4.1.B

Ben does not like the TLB design. He thinks the TLB Entry Valid bit should be dropped and the kernel software should be changed to ensure that all TLB entries are always valid. Is this a good idea? Explain the advantages and disadvantages of such a design.

Problem M4.1.C

Alyssa got wind of Ben's idea and suggests a different scheme to eliminate one of the valid bits. She thinks the page table entry valid and TLB Entry Valid bits can be combined into a single bit.

On a refill this combined valid bit will take the value that the page table entry valid bit had. A TLB entry is invalidated by writing it back to the page table and setting the combined valid bit in the TLB entry to invalid.

How does the kernel software need to change to make such a scheme work? How do the exceptions that the TLB produces change?

Problem M4.1.D

Now, Bud Jet jumps into the game. He wants to keep the TLB Entry Valid bit. However, there is no way he is going to have two valid bits in each TLB entry (one for the TLB entry one for the page table entry). Thus, he decides to drop the page table entry valid bit from the TLB entry.

How does the kernel software need to change to make this work well? How do the exceptions that the TLB produces change?

Problem M4.1.E

Compare your answers to Problem M4.1.C and M4.1.D. What scheme will lead to better performance?

Problem M4.1.F

How about the R bit? Can we remove them from the TLB entry without significantly impacting performance? Explain briefly.

Problem M4.1.G

The processor has a kernel (supervisor) mode bit. Whenever kernel software executes the bit is set. When user code executes the bit is not set. Parts of the user's virtual address space are only accessible to the kernel. The supervisor bit in the page table is used to protect this region—an exception is raised if the user tries to access a page that has the supervisor bit set.

Bud Jet is on a roll and he decides to eliminate the supervisor bit from each TLB entry. Explain how the kernel software needs to change so that we still have the protection mechanism and the kernel can still access these pages through the virtual memory system.

Problem M4.1.H

Alyssa P. Hacker thinks Ben and Bud are being a little picky about these bits, but has devised a scheme where the TLB entry does not need the M bit or the U bit. It works as follows. If a TLB miss occurs due to a load, then the page table entry is read from memory and placed in the TLB. However, in this case the W bit will always be set to 0. Provide the details of how the rest of the scheme works (what happens during a store, when do the entries need to be written back to memory, when are the U and M bits modified in the page table, etc.).

Problem M4.2: Page Size and TLBs (2005 Fall Part D)

This problem requires the knowledge of Handout #6 (Virtual Memory Implementation) and Lecture 5. Please, read these materials before answering the following questions.

Assume that we use a hierarchical page table described in Handout #6.

The processor has a data TLB with 64 entries, and each entry can map either a 4KB page or a 4MB page. After a TLB miss, a hardware engine walks the page table to reload the TLB. The TLB uses a first-in/first-out (FIFO) replacement policy.

We will evaluate the memory usage and execution of the following program which adds the elements from two 1MB arrays and stores the results in a third 1MB array (note that, 1MB = 1,048,576 Bytes):

```
byte A[1048576]; // 1MB array
byte B[1048576]; // 1MB array
byte C[1048576]; // 1MB array

for(int i=0; i<1048576; i++)
  C[i] = A[i] + B[i];
```

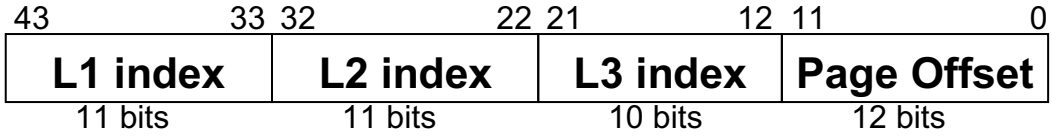
We assume the A, B, and C arrays are allocated in a contiguous 3MB region of physical memory. **We will consider two possible virtual memory mappings**:

- **4KB**: the arrays are mapped using 768 4KB pages (each array uses 256 pages).
- **4MB**: the arrays are mapped using a single 4MB page.

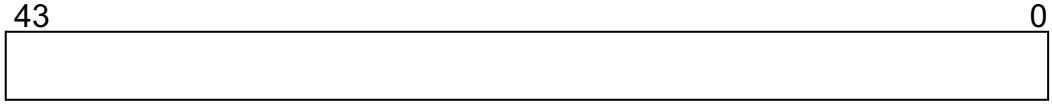
For the following questions, assume that the above program is the only process in the system, and ignore any instruction memory or operating system overheads. Assume that the arrays are aligned in memory to minimize the number of page table entries needed.

Problem M4.2.A

This is the breakdown of a virtual address which maps to a 4KB page:



Show the corresponding breakdown of a virtual address which maps to a 4MB page. Include the field names and bit ranges in your answer.



Problem M4.2.B **Page Table Overhead**

We define page table overhead (PTO) as:

$$\text{PTO} = \frac{\text{Physical memory that is allocated to page tables}}{\text{Physical memory that is allocated to data pages}}$$

For the given program, what is the PTO for each of the two mappings?

$$\text{PTO}_{4KB} = \frac{\quad}{\quad}$$

$$\text{PTO}_{4MB} = \frac{\quad}{\quad}$$

Problem M4.2.C **Page Fragmentation Overhead**

We define page fragmentation overhead (PFO) as:

$$\textbf{PFO} = \frac{\text{Physical memory that is allocated to data pages but is never accessed}}{\text{Physical memory that is allocated to data pages and is accessed}}$$

For the given program, what is the PFO for each of the two mappings?

PFO_{4KB} =	

PFO_{4MB} =	

Problem M4.2.D

Consider the execution of the given program, assuming that the data TLB is initially empty. For each of the two mappings, how many TLB misses occur, and how many page table memory references are required per miss to reload the TLB?

	Data TLB misses	Page table memory references (per miss)
4KB:		
4MB:		

Problem M4.2.E

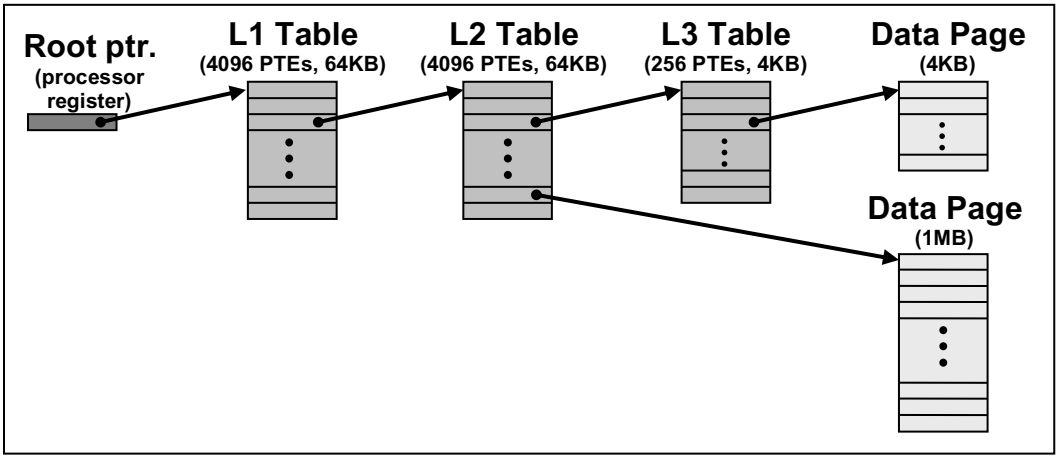
Which of the following is the best estimate for how much longer the program takes to execute with the 4KB page mapping compared to the 4MB page mapping?
Circle one choice and **briefly explain** your answer (about one sentence).

1.01×	**10×**	**1,000×**	**1,000,000×**

Problem M4.3: Page Size and TLBs

This problem requires the knowledge of Handout #6 (Virtual Memory Implementation) and Lecture 5. Please, read these materials before answering the following questions.

The configuration of the hierarchical page table in this problem is similar to the one in Handout #6, but we modify two parameters: 1) this problem evaluates a virtual memory system with two page sizes, 4KB and 1MB (instead of 4 MB), and 2) all PTEs are 16 Bytes (instead of 8 Bytes). The following figure summarizes the page table structure and indicates the sizes of the page tables and data pages (not drawn to scale):



The processor has a data TLB with 64 entries, and each entry can map either a 4KB page or a 1MB page. After a TLB miss, a hardware engine walks the page table to reload the TLB. The TLB uses a first-in/first-out (FIFO) replacement policy.

We will evaluate the execution of the following program which adds the elements from two 1MB arrays and stores the results in a third 1MB array (note that, 1MB = 1,048,576 Bytes, the starting address of the arrays are given below):

```
byte A[1048576]; // 1MB array 0x00001000000
byte B[1048576]; // 1MB array 0x00001100000
byte C[1048576]; // 1MB array 0x00001200000

for(int i=0; i<1048576; i++)
  C[i] = A[i] + B[i];
```

Assume that the above program is the only process in the system, and ignore any instruction memory or operating system overheads. The data TLB is initially empty.

Problem M4.3.A

Consider the execution of the program. There is no cache and each memory lookup has 100 cycle latency.

If all data pages are 4KB, compute the ratio of cycles for address translation to cycles for data access.

If all data pages are 1MB, compute the ratio of cycles for address translation to cycles for data access.

Problem M4.3.B

For this question, assume that in addition, we have a PTE cache with one cycle latency. A PTE cache contains page table entries. If this PTE cache has unlimited capacity, compute the ratio of cycles for address translation to cycles for data access for the 4KB data page case.

Problem M4.3.C

With the use of a PTE cache, is there any benefit to caching L3 PTE entries? Explain.

Problem M4.3.D

What is the minimum capacity (number of entries) needed in the PTE cache to get the same performance as an unlimited PTE cache? (Assume that the PTE cache does not cache L3 PTE entries and all data pages are 4KB)

Problem M4.4: 64-bit Virtual Memory

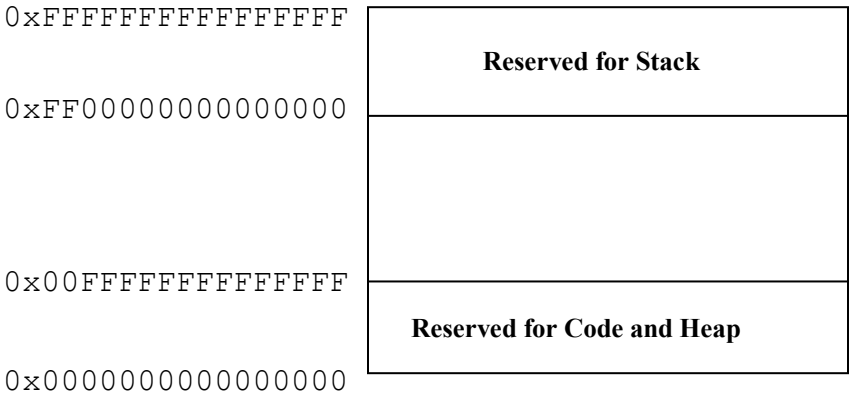
This problem examines page tables in the context of processors with 64-bit addressing.

Problem M4.4.A — Single level page tables

For a computer with 64-bit virtual addresses, how large is the page table if only a single-level page table is used? Assume that each page is 4KB, that each page table entry is 8 bytes, and that the processor is byte-addressable.

Problem M4.4.B — Let's be practical

Many current implementations of 64-bit ISAs implement only part of the large virtual address space. One way to do this is to segment the virtual address space into three parts as shown below: one used for stack, one used for code and heap data, and the third one unused.



A special circuit is used to detect whether the top eight bits of an address are all zeros or all ones before the address is sent to the virtual memory system. If they are not all equal, an invalid virtual memory address trap is raised. This scheme in effect removes the top seven bits from the virtual memory address, but retains a memory layout that will be compatible with future designs that implement a larger virtual address space.

The MIPS R10000 does something similar. Because a 64-bit address is unnecessarily large, only the low 44 address bits are translated. This also reduces the cost of TLB and cache tag arrays. The high two virtual address bits (bits 63:62) select between user, supervisor, and kernel address spaces. The intermediate address bits (61:44) must either be all zeros or all ones, depending on the address region.

How large is a single-level page table that would support MIPS R10000 addresses? Assume that each page is 4KB, that each page table entry is 8 bytes, and that the processor is byte-addressable.

Problem M4.4.C — Page table overhead

A three-level hierarchical page table can be used to reduce the page table size. Suppose we break up the 44-bit virtual address (VA) as follows:

VA[43:33]	VA[32:22]	VA[21:12]	VA[11:0]
1st level index	2nd level index	3rd level index	Page offset

If page table overhead is defined as (in bytes):

$$\frac{\text{PHYSICAL MEMORY USED BY PAGE TABLES FOR A USER PROCESS}}{\text{PHYSICAL MEMORY USED BY THE USER CODE, HEAP, AND STACK}}$$

Remember that a complete page table page (1024 or 2048 PTEs) is allocated even if only one PTE is used. Assume a large enough physical memory that no pages are ever swapped to disk. Use 64-bit PTEs. What is the smallest possible page table overhead for the three-level hierarchical scheme?

Assume that once a user page is allocated in memory, the whole page is considered to be useful. What is the largest possible page table overhead for the three-level hierarchical scheme?

Problem M4.4.D **PTE Overhead**

The MIPS R10000 uses a 40 bit physical address. The physical translation section of the TLB contains the physical page number (also known as PPN), one "valid," one "dirty," and three "cache status" bits.

What is the minimum size of a PTE assuming all pages are 4KB?

MIPS/Linux stores each PTE in a 64 bit word. How many bits are wasted if it uses the minimum size you have just calculated?

Problem M4.4.E **Page table implementation**

The following comment is from the source code of MIPS/Linux and, despite its cryptic terminology, describes a three-level page table.

```
/*
 * Each address space has 2 4K pages as its page directory, giving 1024
 * 8 byte pointers to pmd tables. Each pmd table is a pair of 4K pages,
 * giving 1024 8 byte pointers to page tables. Each (3rd level) page
 * table is a single 4K page, giving 512 8 byte ptes.
 * /
```

Assuming 4K pages, how long is each index?

Index	Length (bits)
Top-level ("page directory")	
2nd-level	
3rd-level	

Problem M4.4.F **Variable Page Sizes**

A TLB may have a *page mask* field that allows an entry to map a page size of any power of four between 4KB and 16MB. The page mask specifies which bits of the virtual address represent the page offset (and should therefore not be included in translation). What are the maximum and minimum reach of a 64-entry TLB using such a mask? The R10000 actually doubles this reach with little overhead by having each TLB entry map *two* physical pages, but don't worry about that here.

Problem M4.4.G **Virtual Memory and Caches**

Ben Bitdiddle is designing a 4-way set associative cache that is virtually indexed and virtually tagged. He realizes that such a cache suffers from a *homonym* aliasing problem. The *homonym* problem happens when two processes use the same virtual address to access different physical locations. Ben asks Alyssa P. Hacker for help with solving this problem. She suggests that Ben should add a PID (Process ID) to the virtual tag. Does this solve the *homonym* problem?

Another problem with virtually indexed and virtually tagged caches is called *synonym* problem. *Synonym* problem happens when distinct virtual addresses refer to the same physical location. Does Alyssa's idea solve this problem?

Ben thinks that a different way of solving *synonym* and *homonym* problems is to have a direct mapped cache, rather than a set associative cache. Is he right?

Problem M4.5: Cache Basics (2005 Fall Part A)

Questions in Part A are about the operations of virtual and physical address caches in two different configurations: direct-mapped and 2-way set-associative. The direct-mapped cache has 8 cache lines with 8 bytes/line (i.e. the total size is 64 bytes), and the 2-way set-associative cache is the same size (i.e. 32 bytes/way) with the same cache line size. The page size is 16 bytes.

Please answer the following questions.

Problem M4.5.A

We ask you to follow step-by-step operations of the virtually indexed, physically tagged, 2-way set-associative cache shown in the previous question (Figure B). You are given a snapshot of the cache and TLB states in the figure below. Assume that the smallest physical tags (i.e. no index part contained) are taken from the high order bits of an address, and that Least Recently Used (LRU) replacement policy is used.
(Only valid (V) bits and tags are shown for the cache; VPNs and PPNs for the TLB.)

Index	V	Tags (way0)	V	Tags (way1)
0	**1**	0x45	**0**	
1	**1**	0x3D	**0**	
2	**1**	0x1D	**0**	
3	**0**		**0**	

Initial cache tag states

VPN	PPN	VPN	PPN
0x0	0x0A	0x10	0x6A
0x1	0x1A	0x20	0x7A
0x2	0x2A	0x30	0x8A
0x3	0x3A	0x40	0x9A
0x5	0x4A	0x50	0xAA
0x7	0x5A	0x70	0xBA

TLB states

After accessing the address sequence (all in virtual address) given below, what will be the final cache states? Please fill out the table at the bottom of this page with the new cache states. You can write tags either in binary or in hexadecimal form.

Address sequence: 0x34 -> 0x38 -> 0x50 -> 0x54 -> 0x208 -> 0x20C -> 0x74 -> 0x54

Index	V	Tags (way0)	V	Tags (way1)

0				
1				
2				
3				

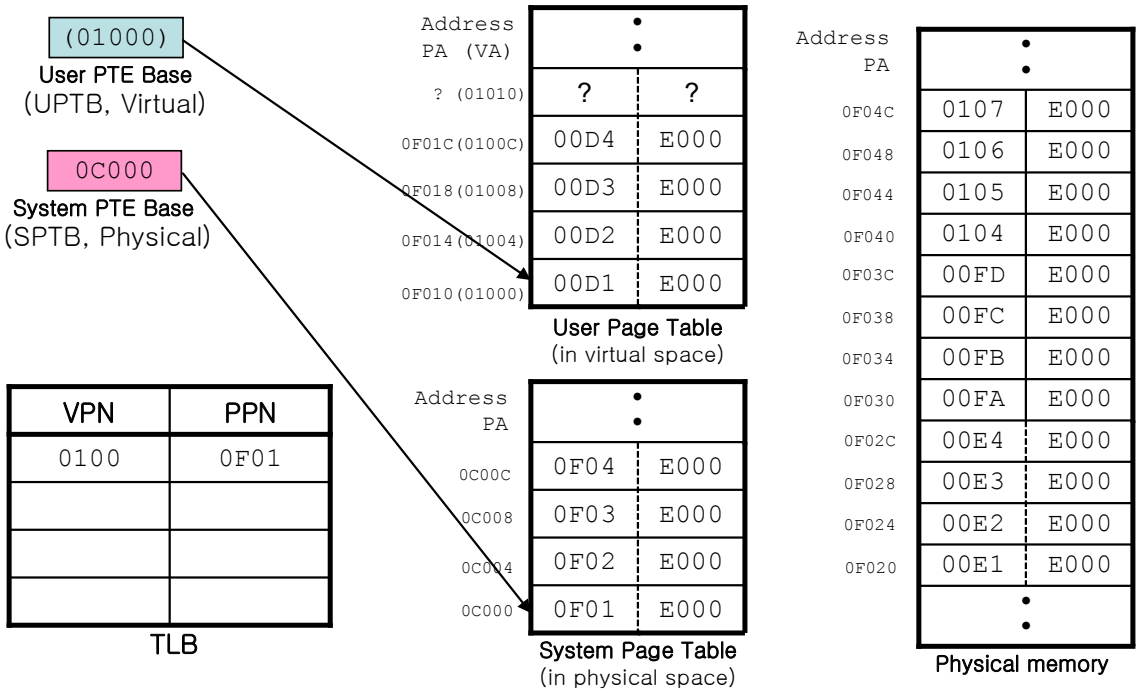
Final cache tag states

Problem M4.5.B

Assume that a cache hit takes one cycle and that a cache miss takes 16 cycles. What is the average memory access time for the address sequence of 8 words given in Question M4.5.A?

Problem M4.6: Handling TLB Misses (2005 Fall Part B)

In the following questions, we ask you about the procedure of handling TLB misses. The following figure shows the setup for this part and each component's initial states.



Notes 1. All numbers are in hexadecimal.
2. Virtual addresses are shown in parentheses, and physical addresses without parentheses.

For the rest of this part, we assume the following:

1) The system uses 20-bit virtual addresses and 20-bit physical addresses.
2) The page size is 16 bytes.
3) We use a linear (not hierarchical) page table with 4-byte page table entry (PTE). A PTE can be broken down into the following fields. (Don't worry about the status bits, PTE[15:0], for the rest of Part B.)

31 … 16	15	14	13	12	11	10	9 … 0
Physical Page Number (PPN)	V	R	W	U	M	S	0000000000

4) The TLB contains 4 entries and is fully associative.

On the next page, we show a pseudo code for the TLB refill algorithm.

```
// On a TLB miss, "MA" (Miss Address) contains the address of that
// miss.  Note that MA is a virtual address.

// UTOP is the top of user virtual memory in the virtual address
// space.  The user page table is mapped to this address and up.
#define UTOP 0x01000

// UPTB and SPTB stand for User PTE Base and System PTE Base,
// respectively.  See the figure in the previous page.

if (MA < UTOP) {
   // This TLB miss was caused by a normal user-level memory access

   // Note that another TLB miss can occur here while loading a PTE.
   LW Rtemp, UPTB+4*(MA>>4);   // load a PTE using a virtual address
}
else {
   // This TLB miss occurred while accessing system pages (e.g. page
    // tables)

   // TLB miss cannot happen here because we use a physical address.
   LW_physical Rtemp, SPTB+4*((MA-UTOP)>>4);  // load a PTE using a
                                              // physical address
}

(Protection check on Rtemp); // Don't worry about this step here
(Extract PPN from Rtemp and store it to the TLB with VPN);
(Restart the instruction that caused the TLB miss);
```

TLB refill algorithm

Problem M4.6.A

What will be the physical address corresponding to the virtual address 0x00030? Fill out the TLB states below after an access to the address 0x00030 is completed.

Virtual address 0x00030 -> Physical address (0x _____)

VPN	PPN
0x0100	0x0F01

TLB states

Problem M4.6.B

What will be the physical address corresponding to the virtual address 0x00050? Fill out the TLB states below after an access to the address 0x00050 is completed. (Start over from the initial system states, not from your system states after solving the previous question.)

Virtual address 0x00050 -> Physical address (0x _____)

VPN	PPN
0x0100	0x0F01

TLB states

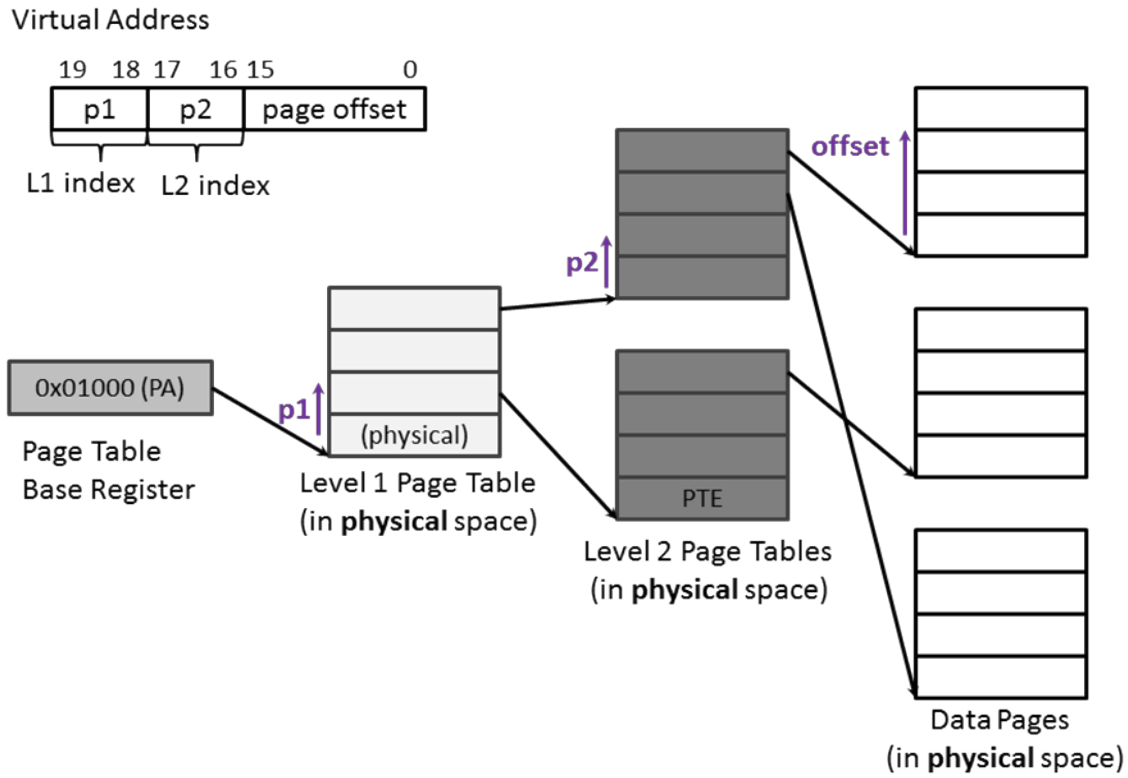
Problem M4.6.C

We integrate virtual memory support into our baseline 5-stage MIPS pipeline using the TLB miss handler. We assume that accessing the TLB does not incur an extra cycle in memory access in case of hits.

Without virtual memory support (i.e. we had only a single address space for the entire system), the average cycles per instruction (CPI) was 2 to run Program X. If the TLB misses 10 times for instructions and 20 times for data in every 1,000 instructions on average, and it takes 20 cycles to handle a TLB miss, what will be the new CPI (approximately)?

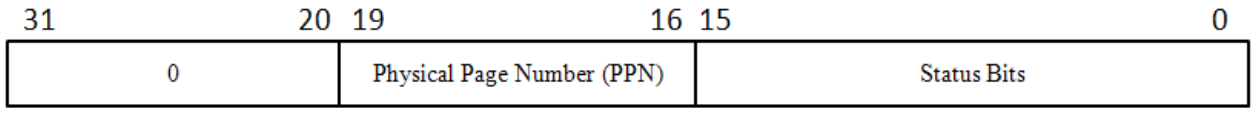
Problem M4.7: Hierarchical Page Table & TLB (Fall 2010 Part B)

Suppose there is a virtual memory system with 64KB page which has 2-level hierarchical page table. The **physical address** of the base of the level 1 page table (**0x01000**) is stored in a special register named Page Table Base Register. The system uses **20-bit** virtual address and **20-bit** physical address. The following figure summarizes the page table structure and shows the breakdown of a virtual address in this system. The size of both level 1 and level 2 page table entries is **4 bytes** and the memory is byte-addressed. Assume that all pages and all page tables are loaded in the main memory. Each entry of the level 1 page table contains the **physical address** of the base of each level 2 page tables, and each of the level 2 page table entries holds the **PTE** of the data page (the following diagram is not drawn to scale). As described in the following diagram, L1 index and L2 index are used as an index to locate the corresponding ***4-byte entry*** in Level 1 and Level 2 page tables.



2-level hierarchical page table

A PTE in level 2 page tables can be broken into the following fields (Don't worry about status bits for the entire part).



Problem M4.7.A

Assuming the TLB is initially at the state given below and the initial memory state is to the right, what will be the final TLB states after accessing the virtual address given below? Please fill out the table with the final TLB states. You only need to write VPN and PPN fields of the TLB. The TLB has 4 slots and is fully associative and if there are empty lines they are taken first for new entries. Also, translate the virtual address (VA) to the physical address (PA). ***For your convenience, we separated the page number from the rest with the colon ":".***

VPN	PPN
0x8	0x3

Initial TLB states

Address (PA)	
0x0:104C	0x7:1A02
0x0:1048	0x3:0044
0x0:1044	0x2:0560
0x0:1040	0xA:0FFF
0x0:103C	0xC:D031
0x0:1038	0xA:6213
0x0:1034	0x9:1997
0x0:1030	0xD:AB04
0x0:102C	0xF:A000
0x0:1028	0x6:0020
0x0:1024	0x5:1040
0x0:1020	0x4:AA40
0x0:101C	0x3:10EF
0x0:1018	0xB:EA46
0x0:1014	0x2:061B
0x0:1010	0x1:0040
0x0:100C	0x0:1020
0x0:1008	0x0:1048
0x0:1004	0x0:1010
0x0:1000	0x0:1038

The part of the memory (in physical space)

<u>Virtual Address</u>:

0xE:17B0 (1110:0001011110110000)

VPN	PPN
0x8	0x3

Final TLB states

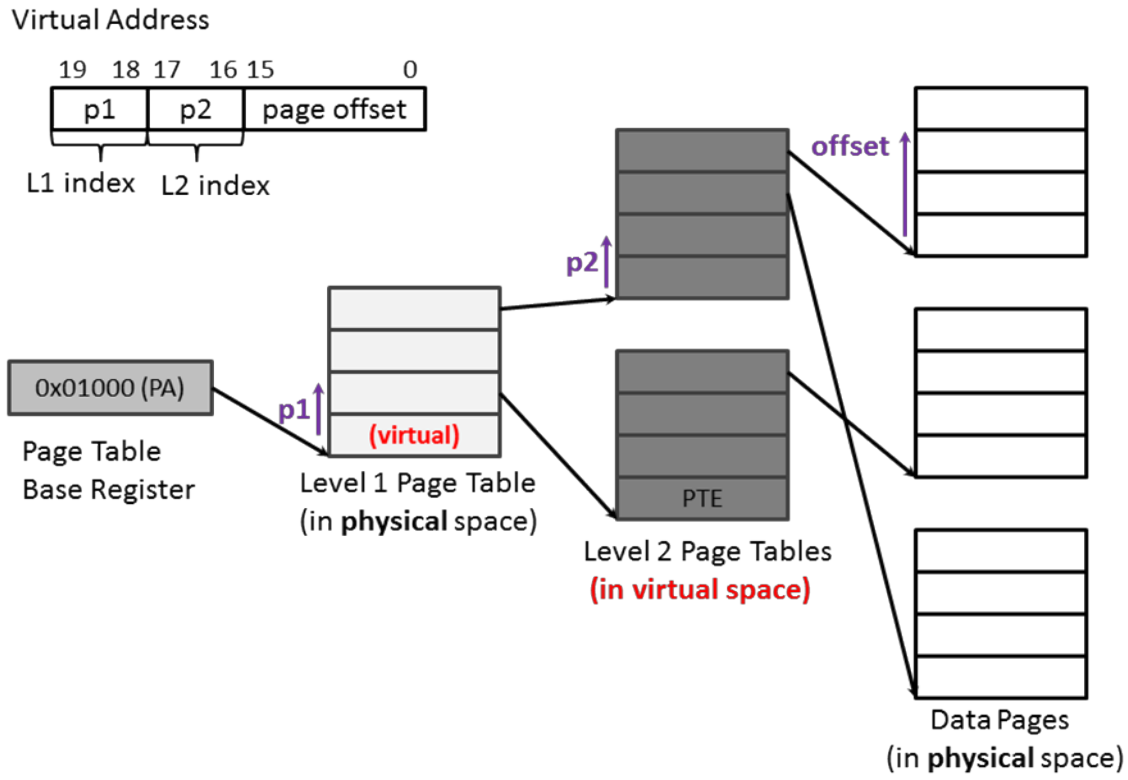
VA 0xE17B0 => PA _____

Problem M4.7.B

What is the total size of memory required to store both the level 1 and 2 page tables?

Problem M4.7.C

Ben Bitdiddle wanted to reduce the amount of physical memory required to store the page table, so he decided to only put the level 1 page table in the physical memory and use the virtual memory to store level 2 page tables. Now, each entry of the level 1 page table contains the **virtual address** of the base of each level 2 page tables, and each of the level 2 page table entries contains the **PTE** of the data page (the following diagram is not drawn to scale). Other system specifications remain the same. (The size of both level 1 and level 2 page table entries is **4 bytes.**)



Ben's design with 2-level hierarchical page table

Assuming the TLB is initially at the state given below and the initial memory state is to the right (**different** from M5.9.A), what will be the final TLB states after accessing the virtual address given below? Please fill out the table with the final TLB states. You only need to write VPN and PPN fields of the TLB. The TLB has 4 slots and it is fully associative and if there are empty lines they are taken first for new entries. Also, translate the virtual address to the physical address. ***Again, we separated the page number from the rest with the colon ":".***

VPN	PPN
0x8	0x1

Initial TLB states

Address (PA)	
.....	.....
0x1:1048	0x3:0044
0x1:1044	0x2:0560
0x1:1040	0x1:0FFF
0x1:103C	0x1:D031
0x1:1038	0xA:6213
0x1:1034	0x9:1997
.....	.....
0x1:0018	0xF:A000
0x1:0014	0x6:0020
0x1:0010	0x1:1040
0x1:000C	0x4:AA40
0x1:0008	0x3:10EF
0x1:0004	0xB:EA46
.....	.....
0x0:1010	0x1:0040
0x0:100C	0x0:1020
0x0:1008	0x2:0010
0x0:1004	0x8:0010
0x0:1000	0x8:1038

The part of the memory (in physical space)

Virtual Address:

0xA:0708 (1010:0000011100001000)

VPN	PPN
0x8	0x1

Final TLB states

VA 0xA0708 => PA _____

Problem M4.7.D

Alice P. Hacker examines Ben's design and points out that his scheme can result in infinite loops. Describe the scenario where the memory access falls into infinite loops.

Problem M4.8: Caches and Virtual Memory (Spring 2015 Quiz 1, Part B)

Problem M4.8.A

Consider a reference stream that repetitively **loops over four addresses, A, B, C, and D (ABCDABCDABCD….)**. We will study how different replacement policies perform on this reference stream, using a small, 2-entry, fully-associative cache.

1. Find out how the cache performs with LRU replacement. Fill the table below to show the cache contents over time, and note whether each access is a hit or a miss. Then, compute the long-term miss ratio (i.e., discounting cache warm-up).

Access	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Address	A	B	C	D	A	B	C	D	A	B	C	D
Entry 1	-	A	A									
Entry 2	-	-	B									
Hit?	N	N	N									

What is the long-term miss ratio under LRU? _____

2. Find out how the cache performs under optimal replacement. **This cache cannot bypass accesses, i.e., on every miss, it must replace an existing block and insert the new block.** Fill the time diagram below, and find the long-term hit rate.

Access	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Address	A	B	C	D	A	B	C	D	A	B	C	D
Entry 1	-	A	A									
Entry 2	-	-	B									
Hit?	N	N	N									

What is the long-term miss ratio under optimal replacement? _____

In the example, is there a simple policy that, without knowing the future, performs as well as the optimal one? If so, which one?

Problem M4.8.B

Consider a byte addressing system with **16-bit virtual and physical addresses**. The system has a cache with the following properties:

- 8 sets with 128 bytes per block
- 4-way set-associative organization
- Virtually-indexed, physically-tagged

1. Suppose we use **256-byte pages**. Where in the cache can **virtual address** 0xABCD live? Please use crosses (X) to mark its possible locations in the diagram below.
   (The binary representation of 0xABCD is 1010 1011 1100 1101.)

Index	**Cache Contents**			
	Way 0	Way 1	Way 2	Way 3
0				
1				
2				
3				
4				
5				
6				
7				

2. As before, suppose we use **256-byte pages**. Where in the cache can **physical address** 0xABCD live? Please use crosses (X) to mark its possible locations.

Index	**Cache Contents**			
	Way 0	Way 1	Way 2	Way 3
0				
1				
2				
3				
4				
5				
6				
7				

3. Suppose we use **1024-byte pages** instead. Where in the cache can **physical address** 0xABCD live? Please use crosses (X) to mark its possible locations.

Index	**Cache Contents**			
	Way 0	Way 1	Way 2	Way 3
0				
1				
2				
3				
4				
5				
6				
7				

Problem M4.8.C

We'd like our memory system to support **two page sizes**: **256-byte small pages** and **1024-byte large pages**. A common approach to support multiple page sizes is to use separate TLBs, one for each page size. Instead, to reduce area overheads, we will use a single TLB to cache translations of both small and large pages, shown in Figure B-1. The TLB has 8 sets and 2 ways. The L bit denotes whether the cached PTE is for a large page.

V = valid bit **L = large page bit (set to 1 when a large page is stored)**
PPN = physical page number

	Way 0				Way 1			
	V	L	Tag	PPN	V	L	Tag	PPN
0								
1								
2								
3								
4								
5								
6								
7								

Figure B-1. TLB for multiple page sizes.

Each TLB access consists of three steps. First, the TLB checks for a small-page match, using the tag and index bits shown in Figure B-2. Second, if it does not find a small-page match, it checks for a large-page match, using the tag and index bits in Figure B-3. Third, if the second lookup misses as well, it results in a TLB miss and a page table walk.

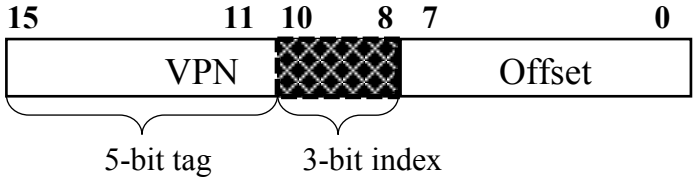


Figure B-2. Tag and index bits for small (256-byte) pages.

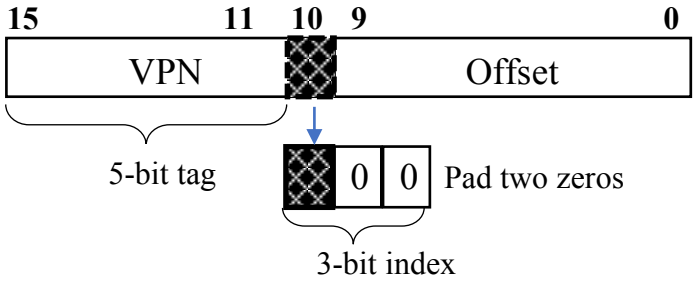


Figure B-3. Tag and index bits for large (1024-byte) pages.

Assume virtual address 0xABBA translates to physical address 0x47BA.

1. If virtual address 0xABBA **belongs to a small (256-byte) page**, fill in the fields of the TLB entry, and mark all possible TLB locations it can be in.

TLB entry

L	Tag	PPN

Possible locations

	Way 0	Way 1
0		
1		
2		
3		
4		
5		
6		
7		

2. If virtual address 0xABBA **belongs to a large (1024-byte) page**, fill in the fields of the TLB entry, and mark all possible TLB locations it can be in.

TLB entry

L	Tag	PPN

Possible locations

	Way 0	Way 1
0		
1		
2		
3		
4		
5		
6		
7		

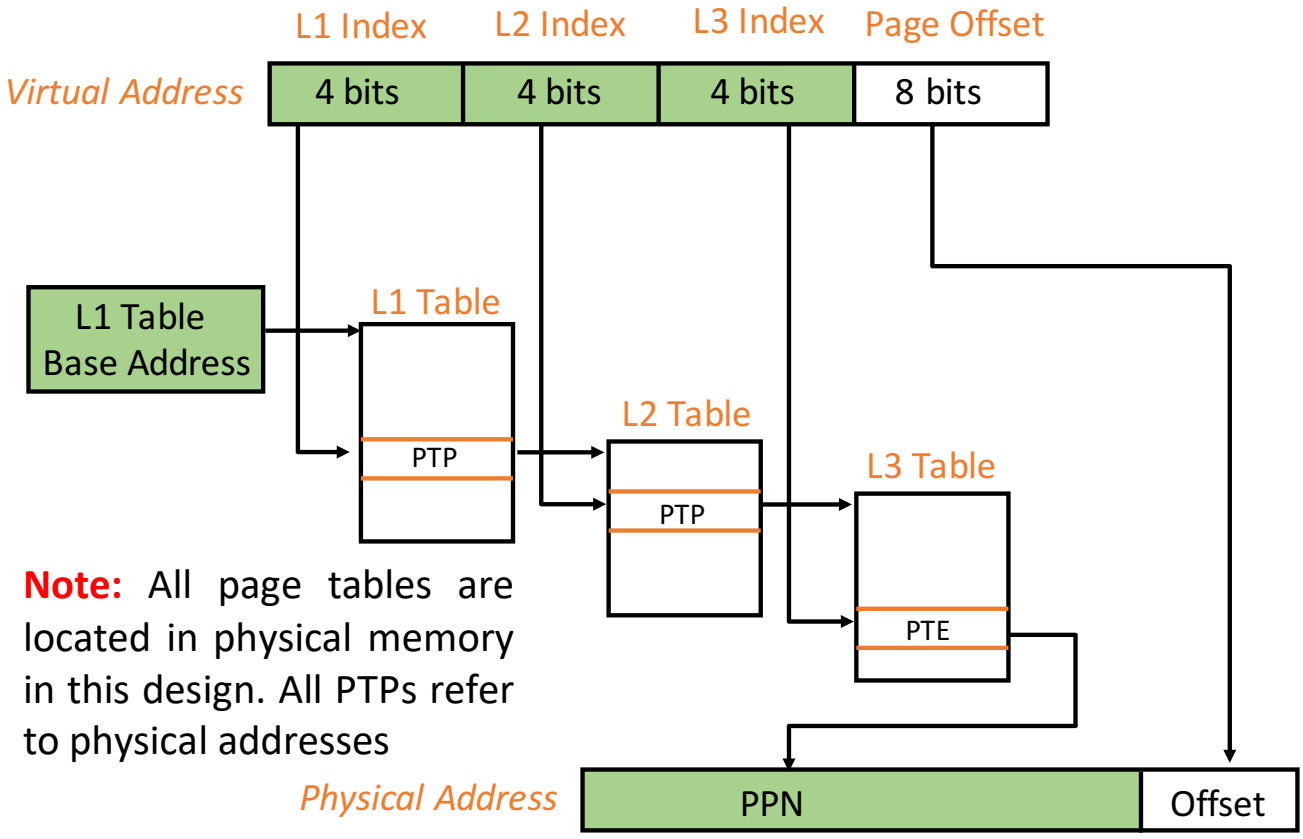
3. What is the reach of this TLB? (TLB reach = maximum amount of memory accessible without TLB misses)

4. This TLB has a utilization problem for large pages. Explain why it happens and how to solve it.

Problem M4.9: Caches and Virtual Memory (Spring 2016 Quiz 1, Part B)

Ben Bitdiddle purchases a new processor to run his 6.823 lab experiments. The processor manual informs Ben that the machine is byte-addressed with 20-bit virtual addresses and 16-bit physical addresses.

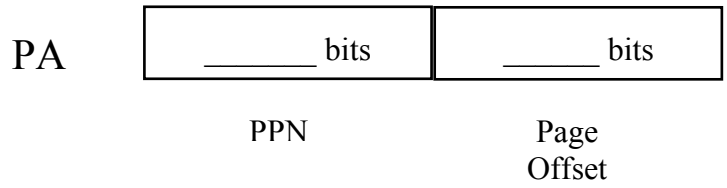
The processor manual only specifies that the machine uses a 3-level page table with the following virtual-address breakdown.



Problem M4.9.A

What is the page size of Ben's machine? _____

Demarcate the physical address into the following fields: Physical Page Number (PPN), Page Offset



Ben executes the following snippet of code on his new processor. Assume integers are 4-bytes long, and the array elements are mapped to virtual addresses `0x0000` through `0x1ffc`. Assume `array` and `sum` have been suitably initialized.

```
1     int array[2048];
2     while (1) {
3       for (int i = 0; i < 4; i++)
4           sum += array[i * 256];
5     }
```

The processor manual states this machine has a TLB with 4 entries. Assume that variables `i` and `sum` are stored in registers, and ignore address translation for instruction fetches; only accesses to `array` require address translation.

Problem M4.9.B

In steady state, how many misses from the TLB will Ben observe per iteration of the `while` loop (lines 3, 4) on average, if (state your reasoning):

a) the TLB is direct-mapped _____

b) the TLB is fully-associative _____
(assume LRU replacement policy)

Problem M4.9.C

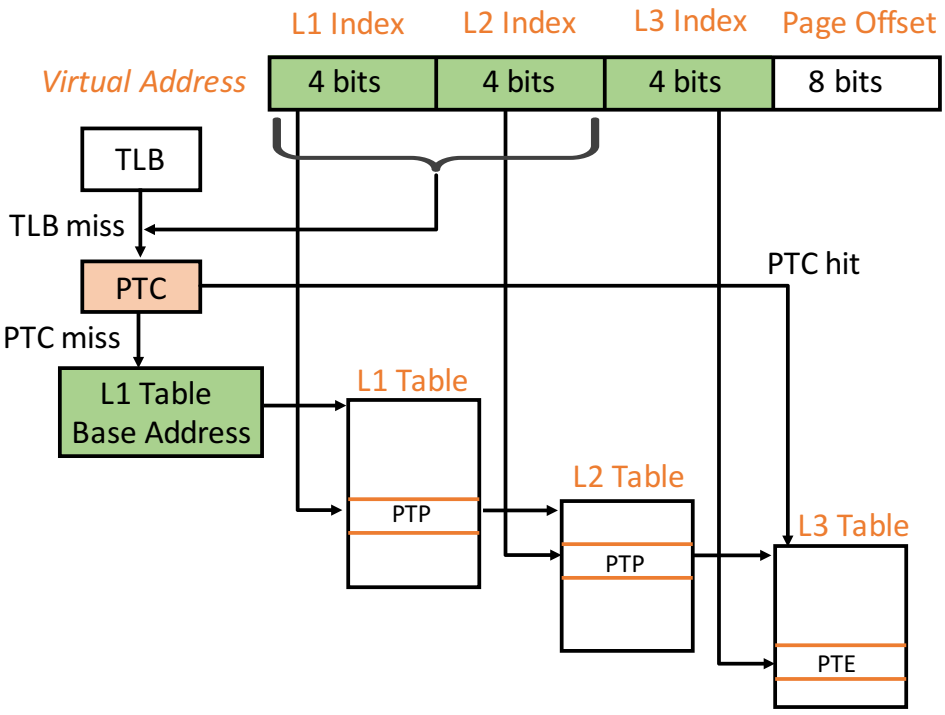
In steady state, how many total memory accesses will Ben observe per iteration of the `while` loop (lines 3, 4) on average, if (state your reasoning):

a) the TLB is direct-mapped _____

b) the TLB is fully-associative _____
(assume LRU replacement policy)

Problem M4.9.D

Ben wonders if he can reduce the number of memory accesses required to perform the address translations. His friend Alyssa P. Hacker suggests adding a *partial-translation cache (PTC)*, in addition to the TLB. The PTC stores a mapping of the higher-order bits of the virtual address to a L3 page table entry. If a translation misses in the TLB, but hits in the PTC, the MMU issues an access to the corresponding L3 page table directly, *skipping* the L1 and L2 page tables. On a TLB miss + PTC miss, the page walk returns the PPN and also installs a translation from VPN to L3 Page Table id in the PTC, and this incurs no additional cost.



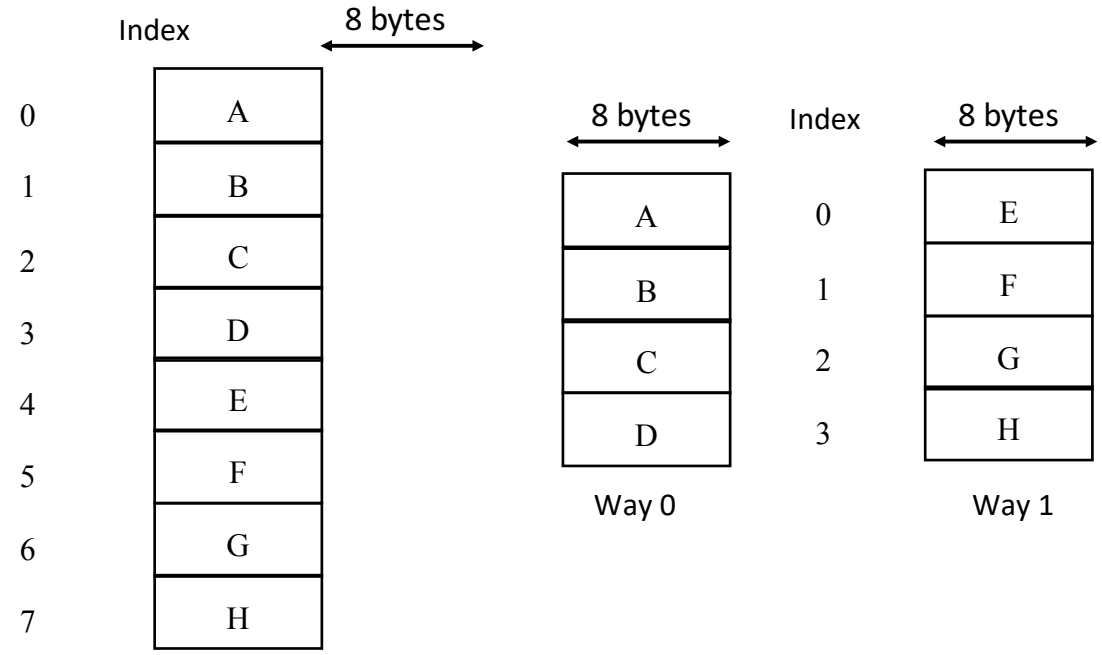
Alyssa proposes adding a PTC with 1 entry to the processor. Does this addition benefit the code snippet in Question 2? How many total memory accesses will Ben observe now, if:

a) the TLB is direct-mapped _____

b) the TLB is fully-associative _____
(assume LRU replacement policy)

Problem M4.9.E

Alyssa's processor contains a 64 byte L1 cache with eight **8-byte cache blocks**, denoted A—H in the figure below. For each configuration shown in the figure, which block(s) can virtual (byte) address `0x34` be mapped to? Assume a **page size of 16 bytes**. Fill out the table at the bottom of the page, indicating each of the possible blocks by its assigned letter (A—H).



(a) Direct-Mapped Cache

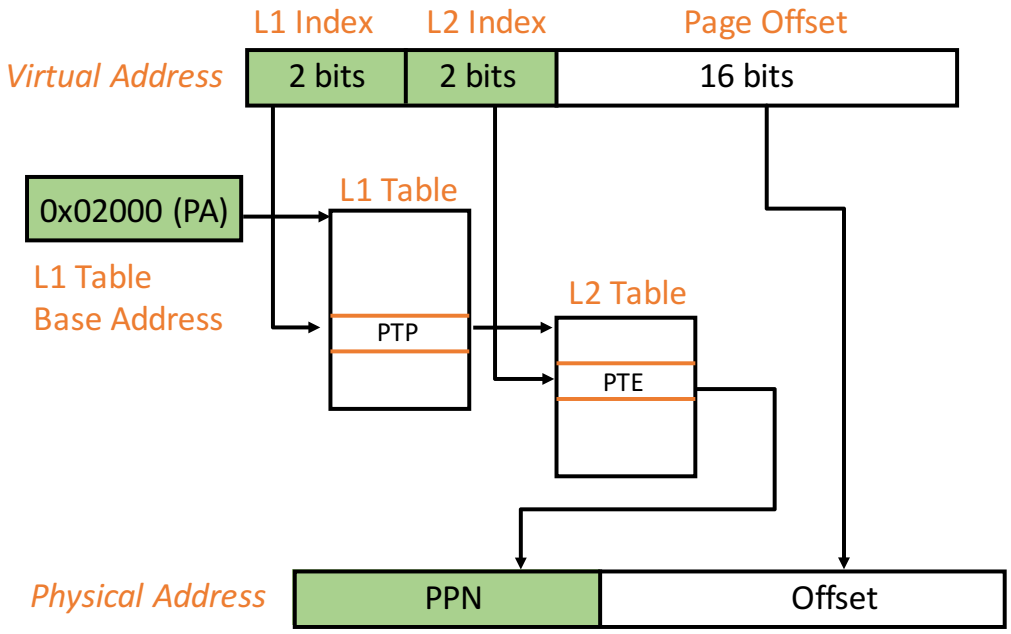
(b) Two-Way Set-Associative Cache

	Virtually Indexed	Physically Indexed
Direct-mapped (a)		
2-way set-associative (b)		

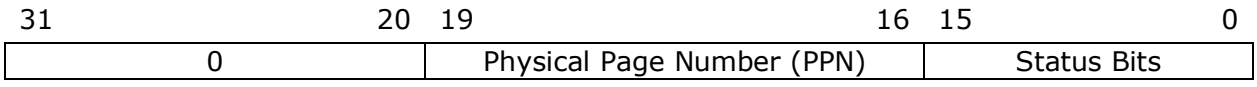
Problem M4.10: Nested paging (Spring 2017 Quiz 1, Part B)

This problem requires the knowledge of Handout #7 (Nested Paging) and Lecture 4 and 5. Please read these materials before answering the following questions.

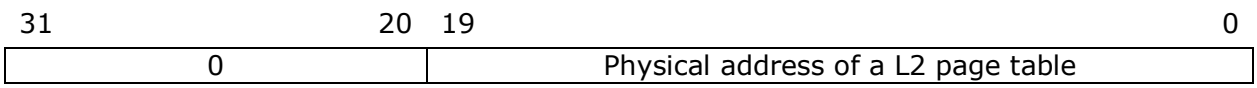
Ben Bitdiddle purchases a new processor to run his 6.823 labs. The processor manual indicates that the machine is byte-addressed with 20-bit virtual addresses and 20-bit physical addresses. The following figure summarizes the 2-level page table structure and shows the breakdown of a virtual address in this system. The physical address of the base of the Level 1 page table (0x02000) is stored in the L1 Table Base Address register. The L1 and L2 page tables are located in physical memory. The size of both L1 and L2 page table entries is 4 bytes. Each entry of the L1 page table contains the physical address of the base of each Level 2 page table (a PTP), and each of the L2 page table entries holds the PTE of the data page.



A PTE in L2 page tables can be broken into the following fields. (Don't worry about status bits).



A PTP in the L1 page table appears as follows.



Problem M4.10.A

Assuming the initial memory state is as shown to the right, what is the physical page number (PPN) of virtual address (VA) 0xB29A0? What is the physical address (PA)? Show and explain your work for full credit. **For your convenience, we separate the page number from the offset with a colon ":".**

Address (PA)	
0x0:2000	0x0:2048
0x0:2004	0x0:2010
0x0:2008	0x0:2038
0x0:200C	0x0:2028
0x0:2010	0x1:0084
0x0:2014	0x5:0DA8
0x0:2018	0x6:11A0
0x0:201C	0xB:9944
0x0:2020	0xC:7FFF
0x0:2024	0x4:B000
0x0:2028	0x7:30B1
0x0:202C	0xD:2E5C
0x0:2030	0x3:A000
0x0:2034	0x6:010C
0x0:2038	0xA:74C0
0x0:203C	0x8:A524
0x0:2040	0x9:FFEE
0x0:2044	0x2:93A4
0x0:2048	0xA:74D0
0x0:204C	0x3:FD40

Snapshot of physical memory

Virtual Address

0xB:29A0 = 0b 1011:0010100110100000

VPN 0xB => PPN _____

VA 0xB29A0 => PA _____

Unable to run Pin in his own environment, Ben's friend, Alyssa P. Hacker, refers him to the *Handout #7 (Nested Paging)* to learn how to run his labs in a virtual machine (much to the TA's dismay!) However, Ben is frustrated by the worst-case performance. Let's find out why.

Problem M4.10.B

Ben starts his foray into virtualization by thinking about `gPA=>hPA` translation.

a) Assuming Ben's host physical memory has the same snapshot as in Question 1, what is the host physical address (`hPA`) of guest physical address (`gPA`) `0xB29A4`? Explain.

`gPA 0xB29A4 => hPA` _____

b) Assuming no TLB, how many accesses to host physical memory are required to access the data associated with a `gPA` (i.e., perform the `gPA=>hPA` translation and fetch the data)? Explain.

Problem M4.10.C

Given a guest virtual address (gVA), the first step of a nested page table walk is to load the relevant guest L1 page table PTP. This provides the base gPA of the guest L2 table. Ben is shocked at how much work is required!

a) Assume host physical memory is initialized as in Question 1 and as shown to the right, the Guest Table Base Address register holds 0xB29A0 (a gPA), and the Host Table Base Address register holds 0x02000 (a hPA). During a nested page table walk of guest virtual address (gVA) 0x61EAC, what are the contents of the guest L1 page table PTP entry? **For your convenience, we separate the page number from the offset with a colon ":".**

Address (PA)	
0x0:2000	0x0:2048
0x0:2004	0x0:2010
0x0:2008	0x0:2038
0x0:200C	0x0:2028
0x0:2010	0x1:0084
...	...
0x2:2998	0xD:2E5C
0x2:299C	0x3:A000
0x2:29A0	0x6:010C
0x2:29A4	0xA:74CC
0x2:29A8	0x7:30B1
...	...
0x5:2994	0x6:11A0
0x5:2998	0xB:F149
0x5:299C	0xC:7BFF
0x5:29A0	0x4:B020
0x5:29A4	0xF:A120
...	...
0xA:299C	0x8:A624
0xA:29A0	0x9:FEED
0xA:29A4	0x2:93A4
0xA:29A8	0xA:7440
0xA:29AC	0x3:FD40

Snapshot of **host** physical memory

Guest Virtual Address (gVA)

0x6:1EAC = 0b 0110:0001111010101100

Guest L1 Table PTP = _____

b) Assume no TLB. Starting from some gVA, how many accesses to host physical memory are required to determine the guest L1 PTP entry of a guest virtual address (gVA)? Explain.

Problem M4.10.D

For the 2-level nested page table in the *Handout #7 (Nested Paging)*, assuming no TLB, how many accesses to host physical memory are required to perform a guest memory access? (i.e. given a gVA, find its corresponding hPA **and fetch the data**). Explain.

Problem M4.10.E

For an *M*-level hierarchical guest page table and an *N*-level hierarchical host page table, assuming no TLB, how many accesses to host physical memory are required to perform a guest memory access? (i.e. given a gVA, find its corresponding hPA **and fetch the data**). Explain.