Homework 8

Chinese Name: 王鹏豪

Pinyin Name: Penghao Wang

Student ID: 2021533138

E-Mail ... @shanghaitech.edu.cn: wangph1

10 1. **Put T (True) or F (False) for each of following statement. [10 points]**

(1) (F) One of the main advantages of using virtual memory is that it allows the OS to use a single address space for all processes.

(2) (F) The translation lookside buffer (TLB) generally holds the most-recently-used page table entries that are used to translate memory page numbers to corresponding disk block numbers where pages are stored on disk.

(3) (F) The size of the virtual address space accessible to the program cannot be larger than the size of the physical address space.

(4) (F) In practice, most of virtual memory systems in modern OS employ direct-mapped strategy to place memory pages for simplicity and low miss rate.

(5) (T) Page tables make it possible to store the pages of a program non-contiguously and add disk in the memory hierarchy

20 **2. TLB Replacement [20 points]**

A processor has 16-bit addresses, 256 byte pages, and an 8-entry fully associative TLB with LRU replacement (the LRU field is 3 bits and encodes the order in which pages were accessed, 0 being the most recent). At some time instant, the TLB for the current process is the initial state given in the Table 2. Assume that all current page table entries are in the initial TLB. Assume also that all pages can be read from and written to.

Fill in the final state of the TLB in Table 3 according to the access pattern in Table 1. Free physical pages: 0x17, 0x18, 0x19.

Table 1: Access Pattern for Memory

No.	Access Pattern
1	Write 0x2333
2	Read 0x11F0
3	Write 0x2023
4	Write 0x1301
5	Read 0x20FF
6	Write 0x3415

Table 2: Initial TLB

VPN	PPN	Valid	Dirty	LRU
0x01	0x11	1	1	0
0x00	0x00	0	0	7
0x10	0x13	1	1	1
0x20	0x12	1	0	5
0x00	0x00	0	0	7
0x11	0x14	1	0	4
0xAC	0x15	1	1	2
0xFF	0x16	1	0	3

Table 3: Final TLB

VPN	PPN	Valid	Dirty	LRU
0x01	0x11	1	1	5
0x23	0x17	1	1	4
0x10	0x13	1	1	6
0x20	0x12	1	1	1
0x13	0x18	1	1	2
0x11	0x14	1	1	3
0xAC	0x15	1	0	7
0x34	0x19	1	1	0

25 **3. Page Table Walk [25 points]**

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 the Page Table Base Register. The system uses 20-bit virtual address and 20-bit physical address. Figure 1 summarizes the page table structure in this system.

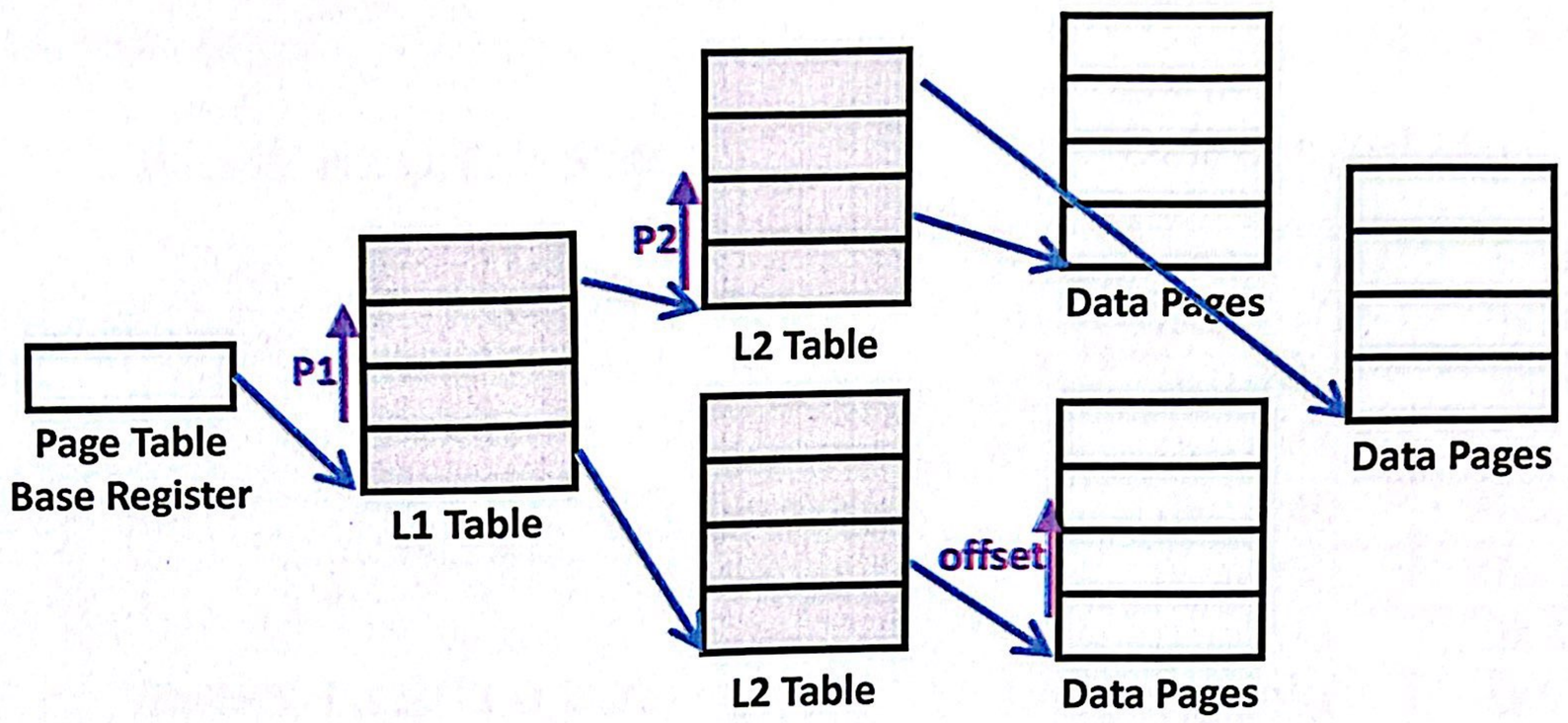
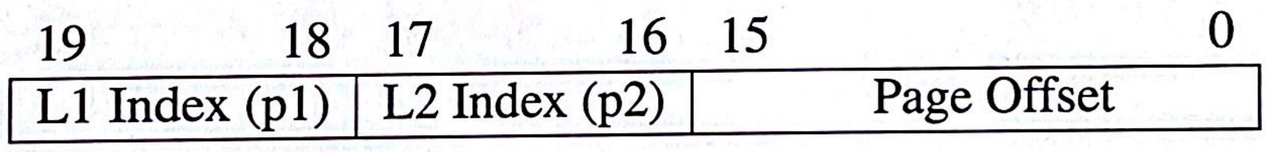


Figure 1: Two-level Hierarchical Page Table

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. The breakdown of a virtual address is:



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. As described in Figure 1, 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.

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

10 (a) **A [10 points]**

Assuming the initial TLB and memory states are in Table 4 and Table 6, respectively, what will be the final TLB states after accessing the virtual address given below? Please fill out the table with the final TLB states in Table 5. 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).

Table 4: Initial TLB State

VPN	PPN
0x8	0x3
–	–
–	–
–	–

Table 5: Final TLB State

VPN	PPN
0x8	0x3
0xD	0x5

VA: 0xD180B ⟶ PA: 0x5180B

Table 6: Initial Memory State

Address (PA)	Content
0x0104C	0x71A02
0x01048	0x30044
0x01044	0x20560
0x01040	0xA0FFF
0x0103C	0xCD031
0x01038	0xA6213
0x01034	0x91998
0x01030	0xDAB04
0x0102C	0xFA000
0x01028	0x60020
0x01024	0x51040
0x01020	0x4AA40
0x0101C	0x310EF
0x01018	0xBEA46
0x01014	0x2061B
0x01010	0x10040
0x0100C	0x01020
0x01008	0x01048
0x01004	0x01010
0x01000	0x01038

15 (b) **B [15 points]**

Li Hua wants 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. Other system specifications remain the same. (The size of both level 1 and level 2 page table entries is 4 bytes.)

Assuming the initial TLB and memory states are in Table 7 and Table 9, respectively, what will be the final TLB states after accessing the virtual address given below? Please fill out the table with the final TLB states in Table 8. 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.

Table 7: Initial TLB State

VPN	PPN
0x8	0x1
–	–
–	–
–	–

Table 8: Final TLB State

VPN	PPN
0x8	0x1
0x2	0x1
0x9	0x4

VA: 0x987DA ⟶ PA: 0x487DA

Table 9: Initial Memory State

Address (PA)	Content
0x11048	0xA0044
0x11044	0xD0560
0x11040	0x10FFF
0x1103C	0xCD031
0x11038	0xA6213
0x11034	0x91998
0x11030	0xC2022
.....	.....
0x10018	0xFA000
0x10014	0x60020
0x10010	0x11040
0x1000C	0x4AA40
0x10008	0x310EF
0x10004	0xBEA46
0x10000	0xD001B
.....	.....
0x01010	0x10040
0x0100C	0x01020
0x01008	0x20008
0x01004	0x80010
0x01000	0x81038

25 4. **Huge Page [25 points]**

Small fixed-sized pages (e.g. 4 KB) reduce internal fragmentation and the page fault penalty compared to large fixed-sized pages. However, when we run an application with a large working set, they may degrade a processor's performance by incurring a number of TLB misses because of their small TLB reach. Therefore, researchers have proposed to support variable-sized pages to increase the TLB reach without losing the benefits of small fixed-sized pages. For example, Linux supports 2MB and 1GB huge pages, besides 4KB data pages.

Li Hua wants to support 2MB huge in his processor. Assume that the system uses 43-bit virtual addresses. 4KB pages are mapped using a three-level hierarchical page table. 2MB pages are mapped using the first two levels of the same page table. An L2 Page Table Entry (PTE) contains information which indicates if it points to an L3 table or a 2MB data page. All PTEs are 8 Bytes. Figure 2 summarizes the page table structure and indicates the sizes of the page tables and data pages.

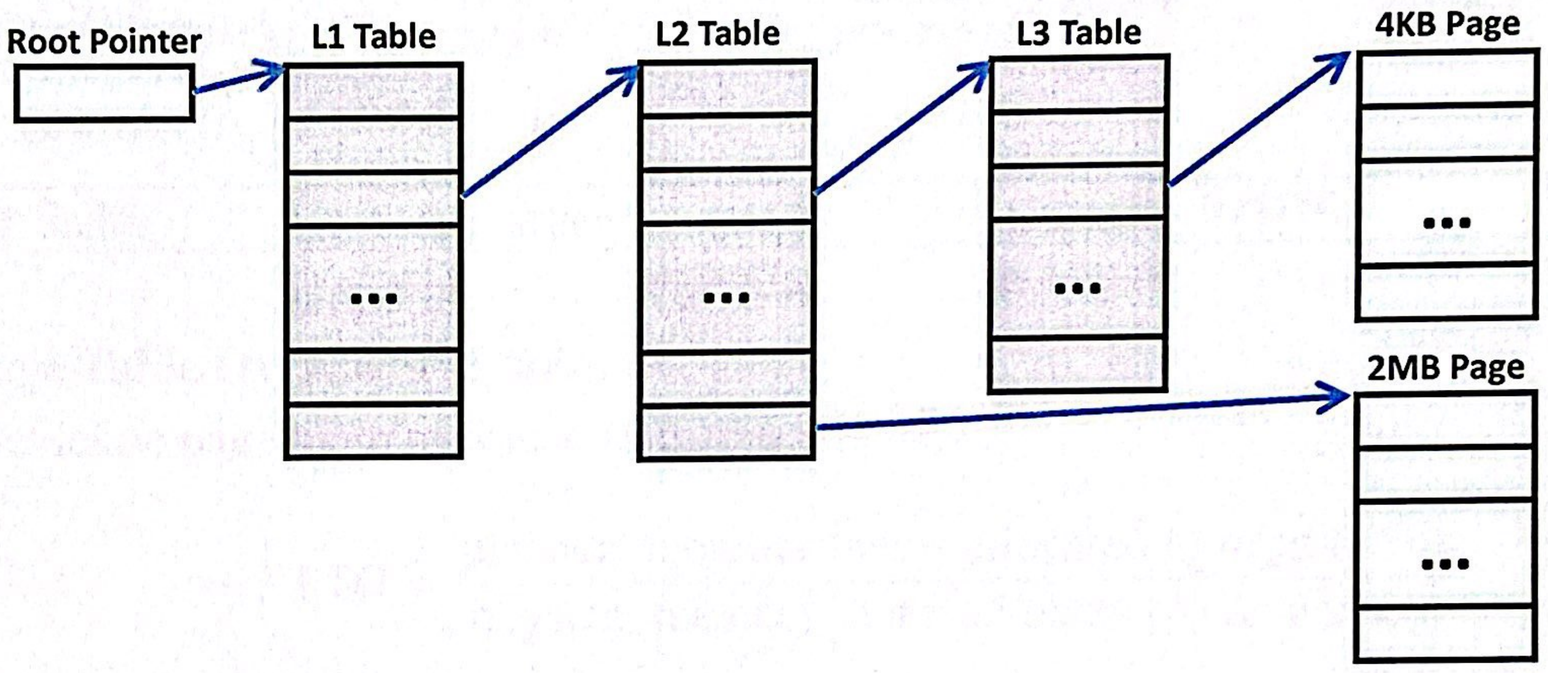


Figure 2: Page Table Structure for Variable-sized Pages

The processor has a data TLB with 64 entries, and each entry can map either a 4KB page or a 2MB 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.

You also need to help Li Hua 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):

```
1 uint8_t A[1048576] = 0; // 1MB array
2 uint8_t B[1048576] = 0; // 1MB array
3 uint8_t C[1048576] = 0; // 1MB array
4 for (int j = 0; j < 1048576; j++) {
5       C[j] = A[j] + B[j];
6 }
```

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

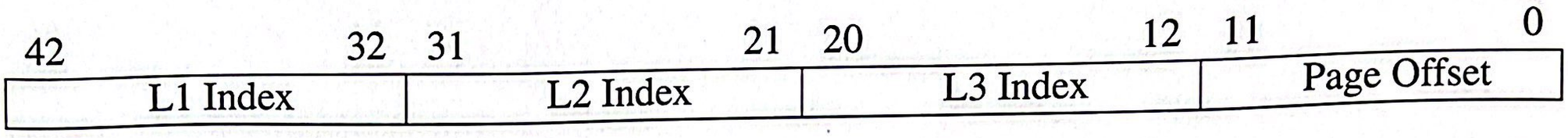
- 4KB: the arrays are mapped using 768 4KB pages (each array uses 256 pages).

- 2MB: the arrays are mapped using two 2MB pages.

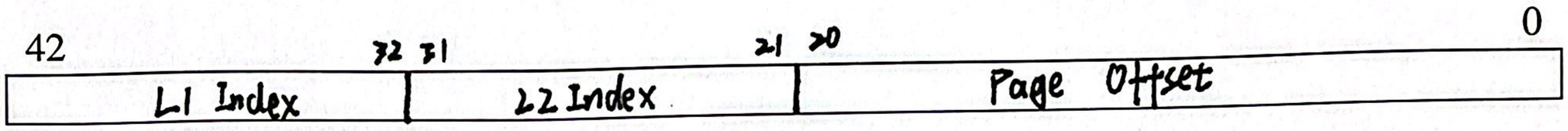
For the following questions, assume that the above program is the only process in the system, and ignore any instruction memory or operating system overheads. Assume that the arrays are aligned in memory to minimize the number of page table entries needed. The system also is trying to use the least number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

5 (a) **Virtual Address [5 points]**

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 2MB page. Include the field names and bit ranges in your answer. 21 2×1024×2×1024



5 (b) **Page Table Overhead [5 points]**

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?

$\text{PTO}_{4KB} =$ $\frac{3}{256}$

$\text{PTO}_{2MB} =$ $\frac{1}{128}$

5 (c) **Page Fragmentation Overhead [5 points]**

We define page fragmentation overhead (PFO) as:

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

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

$\text{PFO}_{4KB} =$ 0

$\text{PFO}_{2MB} =$ $\frac{1}{3}$

10

(d) **TLB [10 points]**

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?

Mapping	Data TLB misses	Page table memory references (per miss)
4KB	768	3
2MB	2	2

20 **5. Virtual Memory and TLB [20 points]**

Li Hua creates a machine which 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.

L1 Index	L2 Index	L3 Index	Page Offset
4 bits	4 bits	4 bits	8 bits

6 **(a) Physical Address [6 points]**

What is the page size of Li Hua's machine?

256 Byte

How many bits do physical page number (PPN) and page offset need in physical address, respectively?

PPN: 8 bits

Page offset: 8 bits

14 **(b) TLB [14 points]**

Li Hua 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 0x6000 through physical address 0x1FFC. Assume array and sum have been suitably initialized.

```
1 int List[4096] = 0;
2 while (1) {
3       for (j = 1; j < 5; j++) {
4             sum += List[j * 512]
5       }
6 }
```

512
1024
~~2048~~
1536
2048

3×512
1536

4×512

The processor manual states this machine has a TLB with 16 entries. Assume that variables j and sum are stored in registers, and ignore address translation for instruction fetches; only accesses to array require address translation.

In steady state, how many misses from the TLB and total memory accesses will Li Hua observe per iteration of the while loop (lines 3, 4) on average, if (state your reasoning):

1. the TLB is direct-mapped

Misses from the TLB: 4

Total memory accesses: 16

2. the TLB is fully-associative (assume LRU replacement policy)

Misses from the TLB: 0

Total memory accesses: 4