

Computer Architecture I 2026 Homework 7

Chinese Name: _____

Pinyin Name: _____

Student ID: _____

E-Mail prefix: _____

10 1. Put T (True) or F (False) for each of following statement. [2 points each]

- (a) () If two processes share the same physical page, then they must use the same virtual address to access that page.
- (b) () A TLB miss can occur even when the referenced page is present in physical memory and its page-table entry is valid.
- (c) () If a machine has 32-bit virtual addresses and 30-bit physical addresses, then no process can use more than 2^{30} bytes of virtual memory.
- (d) () In a multi-level page table, the operating system may avoid allocating lower-level page tables for virtual address regions that are never used.
- (e) () A page fault always implies that some page currently in memory must be written back to disk before the faulting access can complete.

20 **2. Set-Associative TLB Replacement and Address Translation [20 points]**

A processor has **20-bit virtual addresses**, **256-byte pages**, and an **8-entry TLB** organized as a **2-way set associative cache with 4 sets**. The TLB uses **LRU replacement within each set**. In each set, the **LRU field is 1 bit**, where **0 means most recently used (MRU)** and **1 means least recently used (LRU)**.

At some time instant, the TLB for the current process is in the initial state shown in Table 2. Assume:

- all valid entries in the initial TLB correctly reflect the current page table state;
- all referenced pages are readable and writable;
- if a referenced VPN is found in the TLB, the access uses the existing translation;
- if a referenced VPN is not found in the TLB, assume for this problem that it does not already have a resident physical page; the system immediately creates a new translation for that VPN and allocates a new physical page from the free list;
- on a TLB miss, if the VPN is not already in the TLB, a new TLB entry is created;
- when a new translation must be created, the system first uses an **invalid entry in the target set** if one exists; otherwise, it replaces the **LRU entry in that set**;
- on a write access, the corresponding TLB entry's Dirty bit becomes 1; on a read access, the Dirty bit is unchanged;
- newly allocated physical pages are taken in order from the free list.

Free physical pages: **0x031, 0x032, 0x033, 0x034**.

Fill in the final state of the TLB according to the access pattern in Table 1, and answer the following questions.

Table 1: Access Pattern for Memory

No.	Access Pattern
1	Write 0x2333A
2	Read 0x11AF0
3	Write 0x20244
4	Write 0x13001
5	Read 0x20FEE
6	Write 0x34155

Table 2: Initial TLB

Set	Way	Tag	PPN	Valid	Dirty	LRU
0	0	0x044	0x024	1	0	1
0	1	0x02B	0x026	1	1	0
1	0	0x004	0x021	1	1	0
1	1	0x000	0x000	0	0	1
2	0	0x080	0x022	1	1	1
2	1	0x046	0x025	1	0	0
3	0	0x03F	0x027	1	0	0
3	1	0x000	0x000	0	0	1

- 8 (a) For each access in Table 1, determine the following:

- the **VPN**;
- the **TLB set index**;
- the **TLB tag**;
- whether it is a **TLB hit** or **TLB miss**.

No.	Virtual Address	VPN	Set Index	Tag	Hit / Miss
1	0x2333A				
2	0x11AF0				
3	0x20244				
4	0x13001				
5	0x20FEE				
6	0x34155				

12 (b) Fill in the **final state of the TLB** in Table 3 after all six accesses.

Table 3: Final TLB

Set	Way	Tag	PPN	Valid	Dirty	LRU
0	0					
0	1					
1	0					
1	1					
2	0					
2	1					
3	0					
3	1					

20 **3. Page Table Walk [20 points]**

Suppose there is a virtual memory system with **64KB pages** which has a **2-level hierarchical page table**. The physical address of the base of the level 1 page table (**0x02000**) is stored in the **Page Table Base Register**. The system uses **20-bit virtual addresses** and **20-bit physical addresses**. Figure 1 summarizes the page table structure in this system.

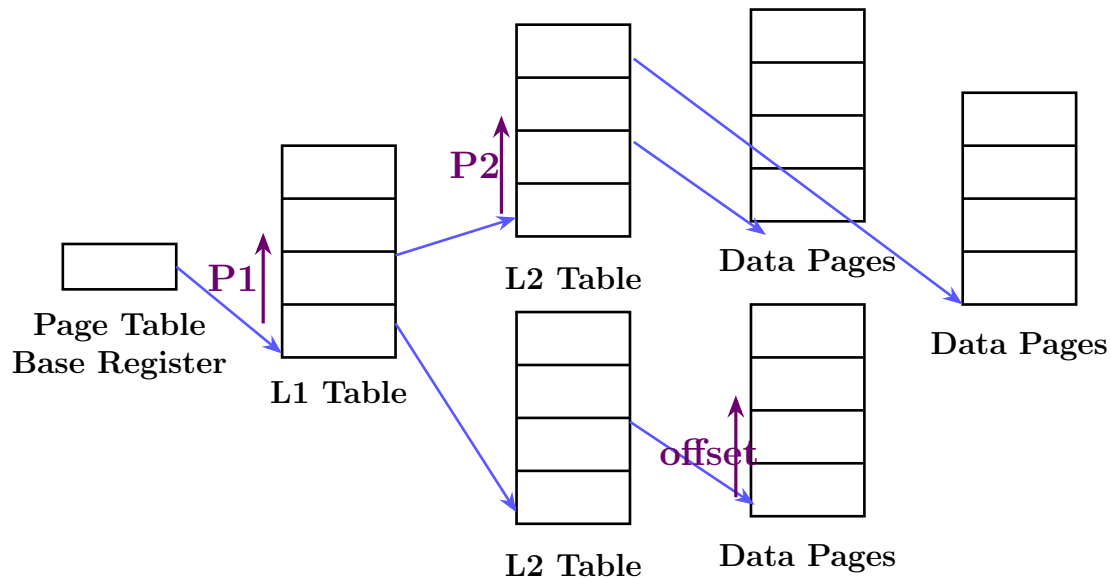


Figure 1: Two-level Hierarchical Page Table

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:

19	1817	1615	0
L1 Index (p1)	L2 Index (p2)	Page Offset	

Each entry of the level 1 page table contains the **physical address** of the base of each level 2 page tables, and each entry of the level 2 page table entries contains the **PTE** of the data page. As described in Figure 1, L1 index and L2 index are used as indices to locate the corresponding **4-byte entry** in the level 1 and level 2 page tables.

A PTE in level 2 page tables can be broken into the following fields (do not worry about status bits for the entire problem):

31	20	19	16	15	0
0	Physical Page Number (PPN)		Status Bits		

8 **(a) Part A [8 points]**

Assuming the initial TLB and memory states are in **Table 4** and **Table 6**, respectively, what will be the final TLB state after accessing the virtual address given below? Please fill out the table with the final TLB state 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
0x4	0x2
–	–
–	–
–	–

Table 5: Final TLB State

VPN	PPN
0x4	0x2

Table 6: Initial Memory State

Address (PA)	Content
0x0200C	0x34000
0x02008	0x2C000
0x02004	0x2A000
0x02000	0x28000
0x2A00C	0x450AA
0x2A008	0x53011
0x2A004	0x4E022
0x2A000	0x39004

VA: $0x5689A \rightarrow$ PA: _____

Show the key intermediate steps:

- $p1 =$ _____ $p2 =$ _____ $offset =$ _____
- L1 entry address = _____ L1 entry content = _____
- L2 entry address = _____ L2 entry content = _____
- TLB hit or miss: _____

12 (b) Part B [12 points]

To reduce the amount of physical memory required to store the page table, the designer decides to only put the level 1 page table in 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 a level 2 page table, and each entry of the level 2 page table 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 state after accessing the virtual address given below? Please fill out the table with the final TLB state in **Table 8**. 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 to the physical address.

Table 7: Initial TLB State

VPN	PPN
0x8	0x1
–	–
–	–
–	–

Table 8: Final TLB State

VPN	PPN
0x8	0x1

Table 9: Initial Memory State

Address (PA)	Content
0x0200C	0x00000
0x02008	0x84010
0x02004	0x90008
0x02000	0x88020
0x1401C	0x00000
0x14018	0x5C012
0x14014	0x3D055
0x14010	0x43001

VA: $0x9D234 \rightarrow$ PA: _____

Show the key intermediate steps:

- $p1 =$ _____ $p2 =$ _____ $offset =$ _____
- L1 entry address = _____ L1 entry content = _____
- Virtual address of L2 table base = _____
- Physical address of translated L2 table base = _____
- Address of the L2 PTE of given VA = _____ L2 entry content = _____

25

4. End-to-End Address Translation and Cache Access [25 points]

A small memory system has the following parameters:

- Memory is **byte-addressable**
- Virtual addresses are **14 bits**
- Physical addresses are **12 bits**
- Page size is **64 bytes**
- The TLB is **4-way set associative** with a total of **16 entries**
- Assume the system uses a single **unified instruction/data TLB** for this problem; only the address translations for the listed virtual addresses need to be considered.
- The L1 data cache is **physically addressed** and **direct-mapped**
- Cache block size is **4 bytes**
- The cache has **16 sets**

Abbreviation notes:

- **VA:** virtual address
- **PA:** physical address
- **VPN:** virtual page number
- **VPO:** virtual page offset
- **PPN:** physical page number
- **PPO:** physical page offset
- **TLB:** translation lookaside buffer
- **TLBI:** TLB index
- **TLBT:** TLB tag
- **PTE:** page table entry
- **CT:** cache tag
- **CI:** cache index
- **CO:** cache block offset
- **V:** valid bit

The current TLB state, page table contents, and L1 d-cache state are given below.

TLB State

Set Index	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	03	–	0	09	0D	1	00	–	0	07	02	1
1	03	2D	1	02	–	0	04	–	0	0A	–	0
2	02	–	0	08	–	0	06	–	0	03	–	0
3	07	–	0	03	0D	1	0A	34	1	02	–	0

Page Table (only first 16 PTEs shown)

VPN	PPN	Valid	VPN	PPN	Valid
00	28	1	08	13	1
01	–	0	09	17	1
02	33	1	0A	09	1
03	02	1	0B	–	0
04	–	0	0C	–	0
05	16	1	0D	2D	1
06	–	0	0E	11	1
07	–	0	0F	0D	1

L1 d-cache State

Index	CT	V	Byte 0	Byte 1	Byte 2	Byte 3
0	19	1	99	11	23	11
1	15	0	–	–	–	–
2	1B	1	00	02	04	08
3	36	0	–	–	–	–
4	32	1	43	6D	8F	09
5	0D	1	36	72	F0	1D
6	31	0	–	–	–	–
7	16	1	11	C2	DF	03
8	24	1	3A	00	51	89
9	2D	0	–	–	–	–
A	2D	1	93	15	DA	3B
B	0B	0	–	–	–	–
C	12	0	–	–	–	–
D	16	1	04	96	34	15
E	13	1	83	77	1B	D3
F	14	0	–	–	–	–

For each of the following virtual addresses, explain how the system translates the virtual address to a physical address and accesses the cache. Indicate whether a TLB hit occurs, whether a page fault occurs, whether a cache hit occurs, and the returned cache byte.

If a **cache miss** occurs, write – in the “Returned cache byte” field. If a **page fault** occurs, write – in the “PPN” field and leave Part C and Part D blank.

9 (a) **Virtual Address 0x03D7**

A. Virtual Address Format

Fill in the virtual-address fields **VPN** and **VPO**.

Parameter	Value
VPN	
VPO	

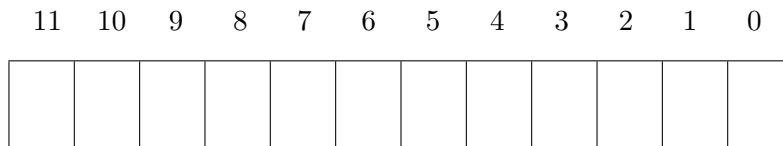
B. Address Translation

Fill in the translation-related fields: **TLBI**, **TLBT**, whether a **TLB hit** occurs, whether a **page fault** occurs, and the resulting **PPN**.

Parameter	Value
TLBI	
TLBT	
TLB hit? (Y/N)	
Page fault? (Y/N)	
PPN	

C. Physical Address Format

Fill in the **12-bit physical address (PA)** in the diagram below using **PPN || PPO**. Do **not** fill cache tag/index/offset in this part.



D. Physical Memory Reference

Using the physical address from Part C, fill in the cache-access fields: **byte offset**, **cache index**, **cache tag**, whether a **cache hit** occurs, and the **returned cache byte**.

Parameter	Value
Byte offset	
Cache index	
Cache tag	
Cache hit? (Y/N)	
Returned cache byte	

9 (b) Virtual Address 0x027C

A. Virtual Address Format

Fill in the virtual-address fields **VPN** and **VPO**.

Parameter	Value
VPN	
VPO	

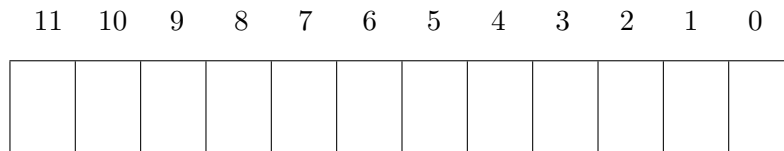
B. Address Translation

Fill in the translation-related fields: **TLBI**, **TLBT**, whether a **TLB hit** occurs, whether a **page fault** occurs, and the resulting **PPN**.

Parameter	Value
TLBI	
TLBT	
TLB hit? (Y/N)	
Page fault? (Y/N)	
PPN	

C. Physical Address Format

Fill in the **12-bit physical address (PA)** in the diagram below using **PPN || PPO**. Do **not** fill cache tag/index/offset in this part.



D. Physical Memory Reference

Using the physical address from Part C, fill in the cache-access fields: **byte offset**, **cache index**, **cache tag**, whether a **cache hit** occurs, and the **returned cache byte**.

Parameter	Value
Byte offset	
Cache index	
Cache tag	
Cache hit? (Y/N)	
Returned cache byte	

7 (c) Virtual Address 0x0109

A. Virtual Address Format

Fill in the virtual-address fields **VPN** and **VPO**.

Parameter	Value
VPN	
VPO	

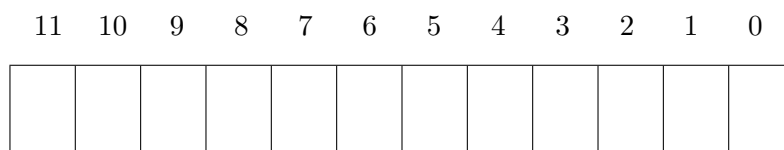
B. Address Translation

Fill in the translation-related fields: **TLBI**, **TLBT**, whether a **TLB hit** occurs, whether a **page fault** occurs, and the resulting **PPN**.

Parameter	Value
TLBI	
TLBT	
TLB hit? (Y/N)	
Page fault? (Y/N)	
PPN	

C. Physical Address Format

Fill in the **12-bit physical address (PA)** in the diagram below using **PPN** || **PPO**. If a page fault occurs, leave this part blank.



D. Physical Memory Reference

Using the physical address from Part C, fill in the cache-access fields: **byte offset**, **cache index**, **cache tag**, whether a **cache hit** occurs, and the **returned cache byte**. If a page fault occurs, leave this part blank.

Parameter	Value
Byte offset	
Cache index	
Cache tag	
Cache hit? (Y/N)	
Returned cache byte	

25 **5. Page Faults Under Demand Paging [25 points]**

A system uses **demand paging** with **byte-addressable memory**. Virtual addresses are **32 bits**. The page size is **1KB**. The size of an **int** is **4 bytes**. Two-dimensional arrays are stored in **row-major order**.

At the beginning of execution, all pages containing the arrays below are **not present in main memory**. Ignore code pages, stack pages, and page-table pages. Only consider the data pages of the arrays. The process is allocated exactly **2 physical page frames**, and the system uses **LRU page replacement**. Assume the TLB is large enough, so **TLB misses can be ignored**.

The arrays start at the following **page-aligned** virtual addresses:

- A[32][32] starts at 0x001800
- B[32][32] starts at 0x005000

Consider the following program:

```
int A[32][32];
int B[32][32];
int sum = 0;

for (int j = 0; j < 32; j++) {
    for (int i = 0; i < 32; i++) {
        A[i][j] = A[i][j] + 1;
        sum += B[i][j];
    }
}
```

4 (a) How many pages does A occupy? How many pages does B occupy?

A occupies: _____ B occupies: _____

9 (b) How many **total page faults** occur during the entire execution of the program?

Total page faults: _____

8 (c) Now suppose the loops are exchanged as follows, while all other conditions remain unchanged:

```
for (int i = 0; i < 32; i++) {
    for (int j = 0; j < 32; j++) {
        A[i][j] = A[i][j] + 1;
        sum += B[i][j];
    }
}
```

How many **total page faults** occur now?

Total page faults after loop interchange: _____

4 (d) Briefly explain why the results of (b) and (c) are very different.
