LABORATORY MANUAL

SC1006/Cx1106
Computer Organization and Architecture

Lab Experiment #4

Cache and Virtual Memory Management

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY**

1. OBJECTIVES

1.1   Understand the behavior of direct mapped cache mapping.

1.2   Understand the principles of cache hit, cache miss.

1.3   Understand concept of Virtual Memory Implementation, including Physical Memory Address translation, Page Hit/Miss and the use of Translation Lookaside Buffer (TLB) to optimize speed of address translation.

2. LABORATORY

This experiment is conducted at the **Hardware Lab 2** at
N4-01b-05 (Tel: 67905036).

3. EQUIPMENT

3.1   Hardware

- Personal Computer with Windows XP or later operating system.

3.2   Software / User Manuals

- CAMERA User's Guide
- CAMERA simulator software

4. INTRODUCTION

A computer processor is very fast and interfaces constantly with the external memory (also known as main memory) for data and instructions. Main memory access times are typically slower than the processor speed, resulting in lower system performance. To improve the overall system performance, cache memory is used. Cache is a small and fast memory that stores information that the processor is likely to use in the very near future.

From the lectures, we know that when data is copied from the main memory (MM) to the cache, it is copied in blocks and the addressing scheme used are different for main memory and cache. To know where to access data corresponding to a specific MM address, the CPU uses a specific mapping scheme that "converts" the MM address into a cache location. This address conversion is done by dividing the bits in the MM address into distinct **fields** known as **TAG:BLOCK:OFFSET** when direct mapped cache is used.

In this experiment, the CAMERA simulator software will be used to study the direct mapped cache mapping scheme covered in the lecture. It acts as a visual and textual learning aid and provides you with interactive tutorials to better understand the fundamental concepts.

Camera also provides a workbench to reinforce the concepts in virtual memory and paging. Physical memory, virtual memory, the process page table, and the Translation Lookaside Buffer (TLB) are all available for inspection when progressing through a sequence of addresses. The number of TLB hits and misses, as well as the number of page hits and misses are also recorded. If a page fault occurs and this involves replacing another page in main memory, CAMERA uses the popular *LRU* replacement algorithm.

The simulator software may be downloaded from the course website on NTULearn for your own explorations.

5.   EXPERIMENT

5.1   Launching the CAMERA Simulator

Check with the lab instructor which directory is the CAMERA simulator software resided in. Open a command prompt window or powershell window and navigate to the camera directory. Type the following command *java Camera* at the command prompt as shown in Fig.1. Note the letter "C" must be capitalized.
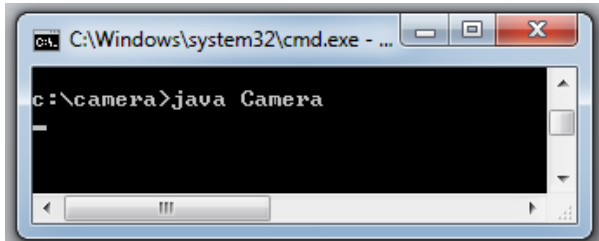

Fig.1: Start CAMERA Menu

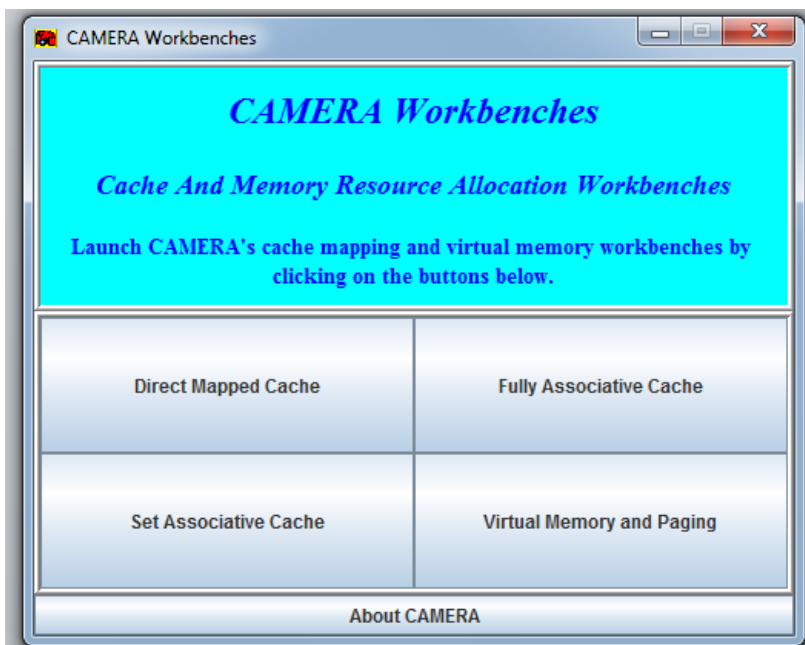This will initialize the CAMERA main menu as shown in Fig.2.

Fig. 2: CAMERA main menu

5.2 Direct Mapped Cache

5.2.1 Concept

Direct mapping maps block X of main memory to block Y of cache. Multiple main memory blocks may be mapped to the same cache block as shown in Fig. 3. The **tag field** is used to uniquely identify the main memory block that resides in a particular cache block at any instance. To perform direct mapping, the main memory is partitioned into the fields as shown in Fig. 4. Recap on the lecture: **TAG** is the bit field that is used to uniquely identify each MM blocks, **Block** is the bit field that points to the cache block that stores the target data and **Offset** is the bit field that indicate the target data's position offset from the start of the block.
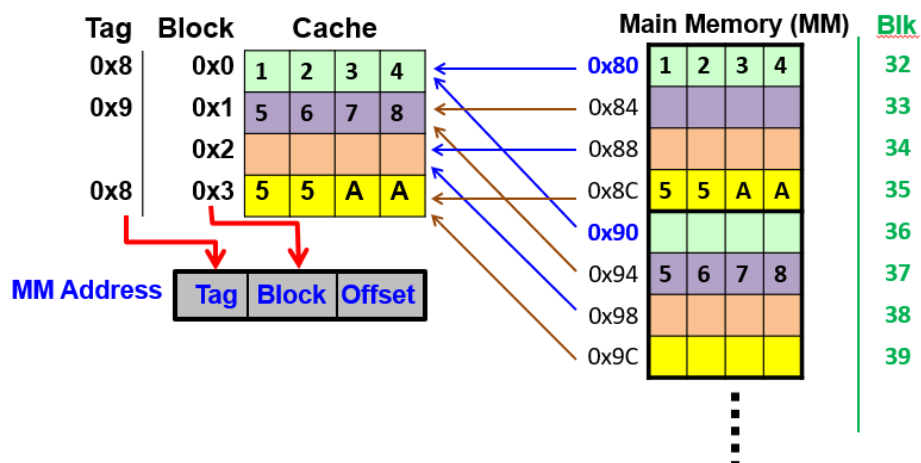
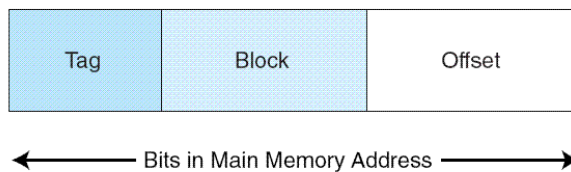**Fig. 3: Direct Mapping of Main Memory Blocks to Cache Blocks**

Fig. 4: The Format of Main Memory Address using Direct Mapping

5.2.2   Simulator

To enable direct mapped cache simulation, click on the "**Direct Mapped Cache**" button in the main menu. This initializes the simulator and call up the simulator window as shown in Fig. 5. In this instance, direct mapping maps 32 blocks (8 words per block) of main memory to 16 blocks of cache. Note that the basic data unit used in this simulator is 'word'. You can treat it as a 'byte' as discussed in the lecture.
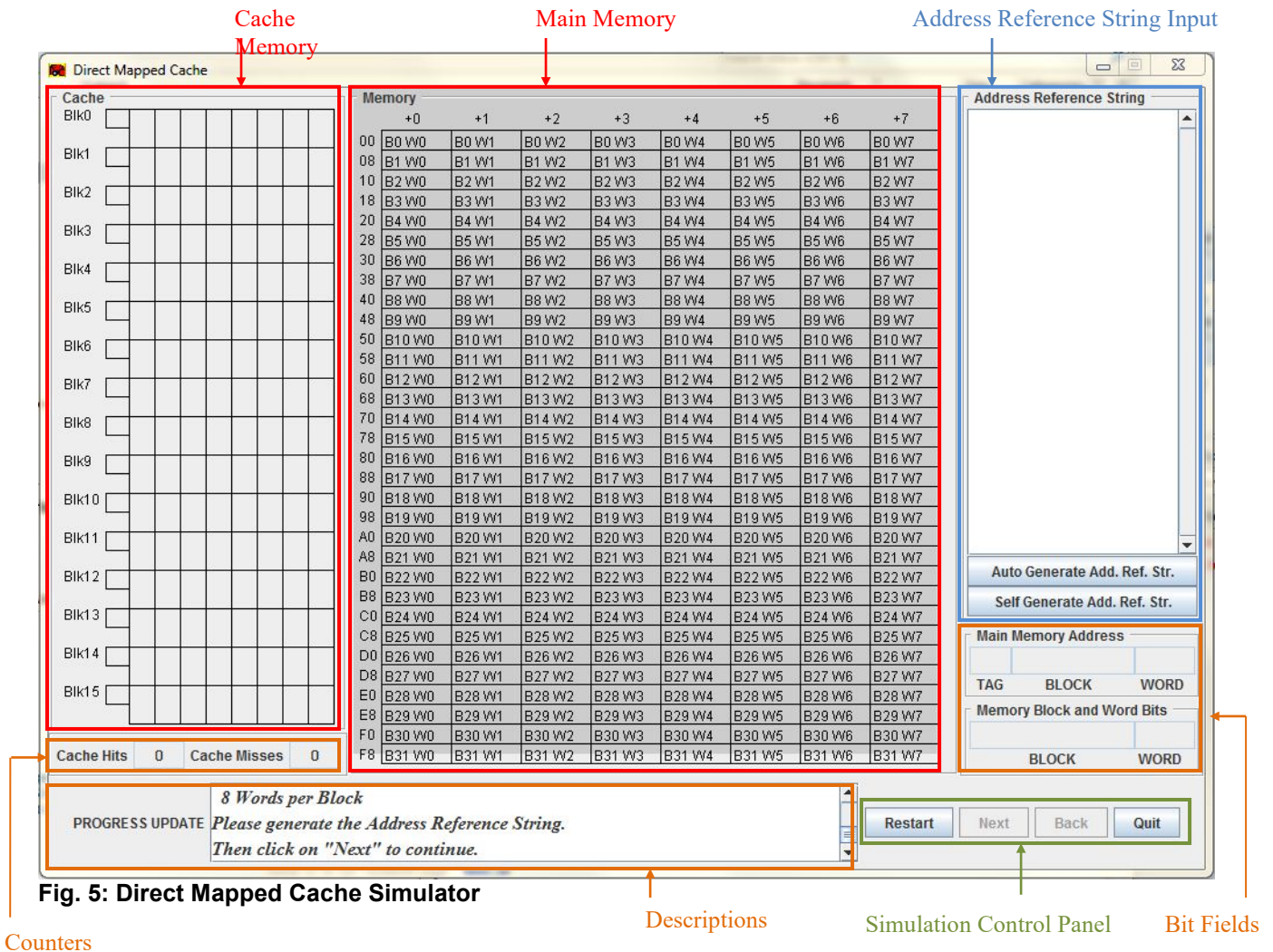

Fig. 5: Direct Mapped Cache Simulator

At the start of the simulation, the cache is cleared, and the user is prompted to generate an address reference string. Address reference string is a list of address that the CPU will access sequentially. The user can request for the application to automatically generate a string of 10 MM addresses to be executed by the simulator by clicking on the "Auto Generate Add. Ref. Str." button. Alternatively, the user can manually enter valid addresses one at a time up to a predefined maximum. Once the address reference string is created, the simulation can begin by using the "Next" button to step forward. Note that you will need to re-enter the address reference string again if you press 'Restart' button.

5.2.3 Sequential Addressing

#Task: Use the "Self Generate Add. Ref. Str." button to enter the main address value **A0** and press Enter. Note that addresses are in Hex format. To input another address, you need to activate the input window by clicking once at the text entry box. Repeat to input **A1, A2, A3, 25, 26 and 27**. At the end, click the "Done" button to complete the data entry. You should obtain the same reference string as shown in Fig. 6.

Fig. 6: Manually Entered Sequentially Increasing Addresses

#Task: Step through the simulation using the "**Next**" button. At each step, read the comments in the "**PROGRESS UPDATE**" window. **Note your observations in Table 1**. Each address in the generated address string goes through an access cycle. Follow the comments in the "**PROGRESS UPDATE**" window to understand the main steps in each access cycle. For a particular address, if the information resides in the cache memory, will the CPU still access the main memory? Note that Cache Replacement refers to scenario where existing data in the Cache is replaced by new data from the Main Memory. Transferring data into an empty cache block is not considered a Cache Replacement.

Table 1: Direct Mapping Sequentially Increasing Addresses

Address Reference String	Main Memory Address			Cache Hit/ Cache Miss	Cache Replace (Yes/No)
	TAG	BLOCK	WORD		
A0	1	0100	000	Cache Miss	No
A1	1	0100	001	Cache Hit	No
A2	1	0100	010	Cache Hit	No
A3	1	0100	011	Cache Hit	No
25	0	0100	101	Cache Miss	Yes
26	0	0100	110	Cache Hit	No
27	0	0100	111	Cache Hit	No

#Question:

- New data is fetched into the cache when a cache miss happens, what is the size (number of words) of the data fetched into the cache? **8 words as the whole cache block is cached at once**

- How many bits are there in the TAG, BLOCK and WORD field? How are the bits related to the structure of the cache?   **1:4:3**

5.2.4   Random Addressing

#Task: Restart the simulator by using the "Restart" button. Use the "Self Generate Add. Ref. Str." to input the following address reference string {*A0, 38, B2, 63, 22, F5*}. Note your observations in Table 2. Close the simulator window at the end of the simulation.

Table 2: Direct Mapping Random Addresses

Address Reference String	Main Memory Address			Cache Hit/ Cache Miss	Cache Replace (Yes/No)
	TAG	BLOCK	WORD		
A0	1	0100	000	Cache Miss	No
38	0	0111	000	Cache Miss	No
B2	1	0110	010	Cache Miss	No
63	0	1100	011	Cache Miss	No
22	0	0100	010	Cache Miss	Yes
F5	1	1110	101	Cache Miss	No

#Question:

- With each cycle, if the information resides in the cache memory, the Cache Hits counter is updated. If not, the Cache Misses counter is updated. Do you see any difference in the number of cache hits and cache misses when you compare Section 5.2.3 with Section 5.2.4?

- Is the effectiveness of the cache memory dependent on the Address Reference String? Which type of memory addressing do you expect to be more commonly encountered?

5.2.5 Cache Thrashing

#Task: Restart the simulator by using the "Restart" button. Use the "Self Generate Add. Ref. Str." to input the following address reference string { **A5, 22, A6, 23, A7 and 24**}. Note your observations in Table 2. Close the simulator window at the end of the simulation.

Table 3: Cache Thrashing

Address Reference String	Main Memory Address			Cache Hit/ Cache Miss	Cache Replace (Yes/No)
	TAG	BLOCK	WORD		
A5	1	0100	101	Cache Miss	No
22	0	0100	010	Cache Miss	Yes
A6	1	0100	110	Cache Miss	Yes
23	0	0100	011	Cache Miss	Yes
A7	1	0100	111	Cache Miss	Yes
24	0	0100	100	Cache Miss	Yes

#Question:

- What is the reason for the high cache miss count?

  The cache block is always the same for all the reference strings but with different tags, thus it keeps hitting cache miss

- Is the cache utilized effectively under this scenario?

  No, as the cache needs to re-written every aaddress

5.3   Virtual Memory

5.3.1   Concept

Virtual is a memory management technique that maps memory addresses used by program/code (virtual address) to actual addresses in the physical memory (physical address). Virtual memory to physical memory address translation is carried out by the Operating System. The address translation information is kept in a table known as the Page Table. As the Page Table resides in the main memory, access to data store in the Page Table may be slow. Hence, a Translation Lookaside Buffer (TLB) is used to speed up the process. TLB resides in fast memory and acts like a cache to the Page Table.

The CAMERA program simulates a system with 8 pages of virtual memory and 4 pages of physical memory. Page size is 32 words. TLB has 4 entries. Note that a **TLB Hit** refers to the case where the address translation information for a particular page can be found in the TLB when OS visits it. A PAGE Hit refers to the case where address translation information can be found in the page table when OS visits it.
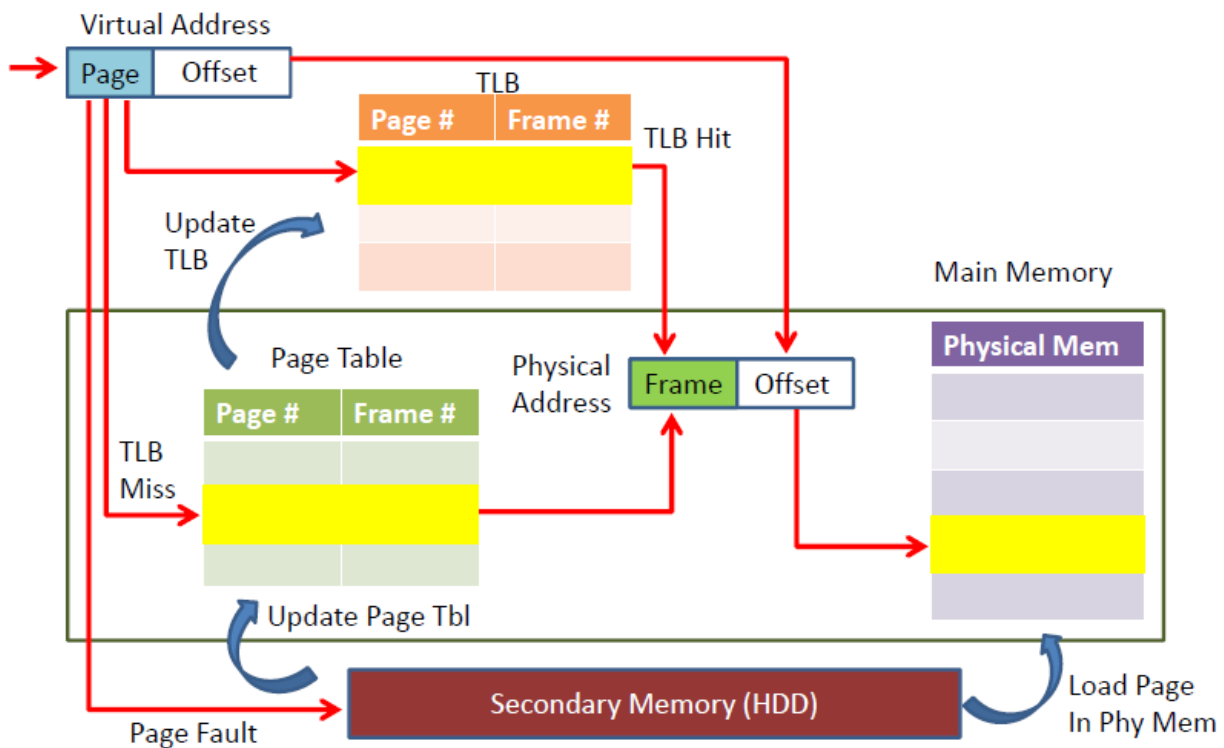

Fig. 7: Virtual Memory System

5.3.2   Simulation

#Task: Click on "**Auto Generate Add. Ref Str**" button and step through the list of addresses by clicking on the '**Next**' button. Go through in detail the comments displayed in the "PROGRESS UPDATE" Window to understand how the virtual memory system works. Note down the information listed in Table 6 for each address reference.

Repeat the same process for the following Hexadecimal Address Sequence: 00, 01, 02, 03, 04, 56, 57, 58, 59, 5A. Use "**Self Generate Add. Ref Str**" button to generate the list. Note down the corresponding information in Table 7.
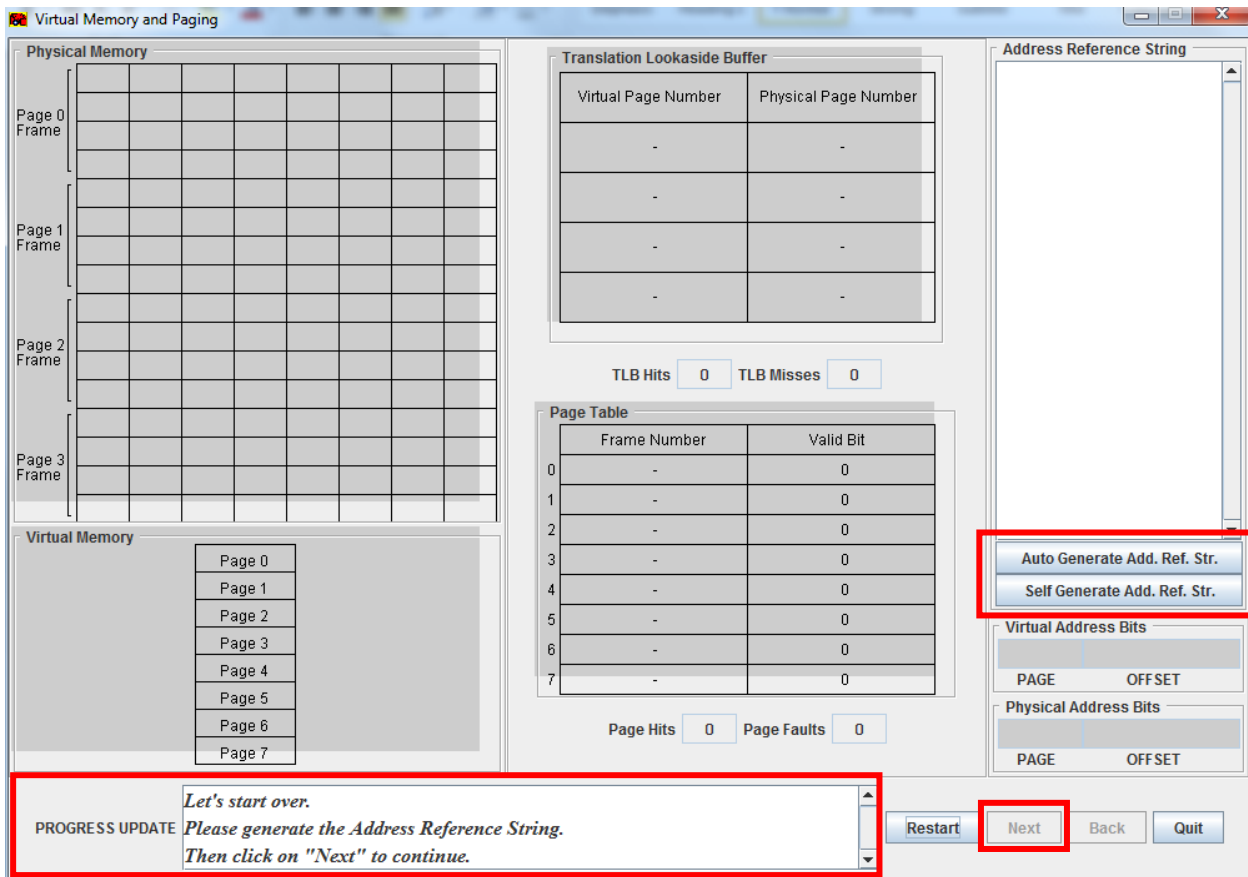
Fig. 8: Virtual Memory Simulator

Table 6: Random Address Reference

Virtual Address	Virtual Page	Physical Frame	TLB Hit	TLB Miss	Page Hit	Page Miss	Physical Address
011 10100	011	0		✓		✓	20
010 11111	010	1		✓		✓	63
000 11100	000	2		✓		✓	92
100 10000	100	3		✓		✓	112
000 11011	000	2	✓		✓		91
110 11100	110	0		✓		✓	28
100 11101	100	3	✓		✓		125
111 10101	111	1		✓		✓	53
110 00111	110	0	✓		✓		7
100 01100	100	3	✓		✓		108

Table 7: Sequential Address Reference

Virtual Address	Virtual Page	Physical Frame	TLB Hit	TLB Miss	Page Hit	Page Miss	Physical Address
00	0	0		✓		✓	0
01	0	0	✓		✓		1
02	0	0	✓		✓		2
03	0	0	✓		✓		3
04	0	0	✓		✓		4
56	2	1		✓		✓	54
57	2	1	✓		✓		55
58	2	1	✓		✓		56
59	2	1	✓		✓		57
5A	2	1	✓		✓		58

#Question:

- Which translation information does the CPU read first when it needs to access a piece of data from the memory? TLB or Page Table? Why?

- Does the matching Virtual Page and the Physical Frame need to be the same size? Why?

- Under what situation would a Page Hit occur in the simulator? Any Page Hit observed in the simulation conducted? Why?

- Which Address Reference string (Table 6 or 7) gives rise to more TLB Hits and less Page Faults/TLB Miss i.e. more efficient access?

- Which Address Reference string (Table 6 or 7) is more commonly found in real life coding?

6.   **REFERENCES**

- The CAMERA User Guide

- Null LM and Rao K, "CAMERA: Introducing Memory Concepts via Visualization", ACM Special Interest Group on Computer Science Education (SIGCSE 05), pp.96-100, 2005