NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

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.

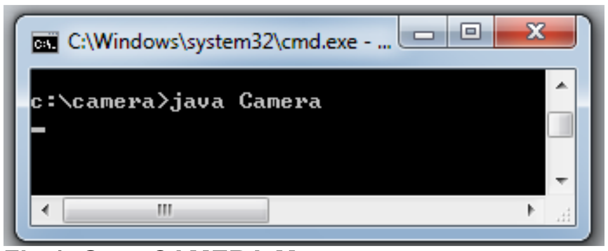
Camera also provides a workbench to reinforce the concepts in virtual memory and paging. Physical memory, virtual memory, the process page table, and the Translation Lookaside Buffer (TLB) are all available for inspection when progressing through a sequence of addresses. The number of TLB hits and misses, 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

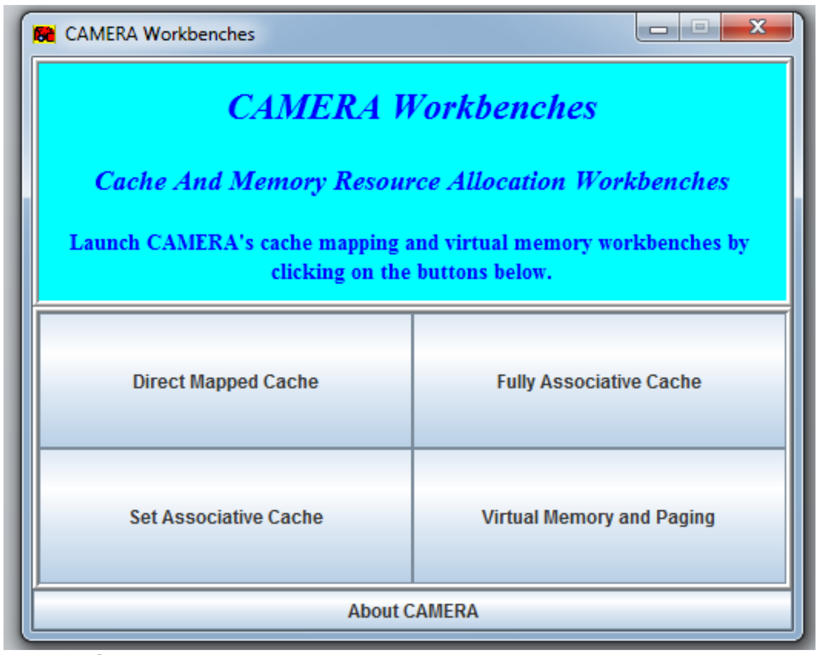
Check with the lab instructor which directory is the CAMERA simulator software resided in. Open a command prompt window or powershell window and navigate to the camera directory. Type the following command ***java Camera*** at the command prompt as shown in Fig.1. Note the letter “C” must be capitalized.



```
c:\camera>java Camera
```

Fig.1: Start CAMERA Menu

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



CAMERA Workbenches

Cache And Memory Resource Allocation Workbenches

Launch CAMERA's cache mapping and virtual memory workbenches by clicking on the buttons below.

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.

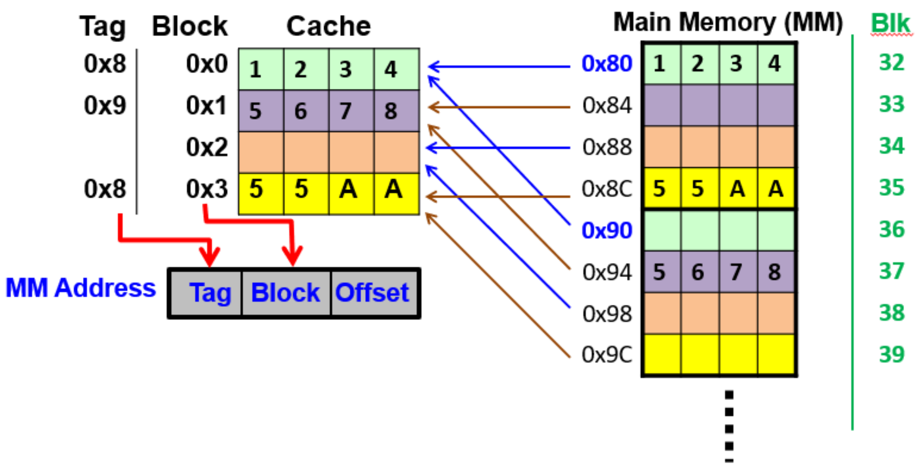


Fig. 3: Direct Mapping of Main Memory Blocks to Cache Blocks

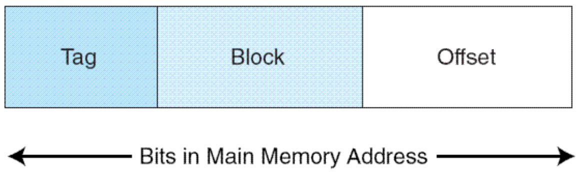
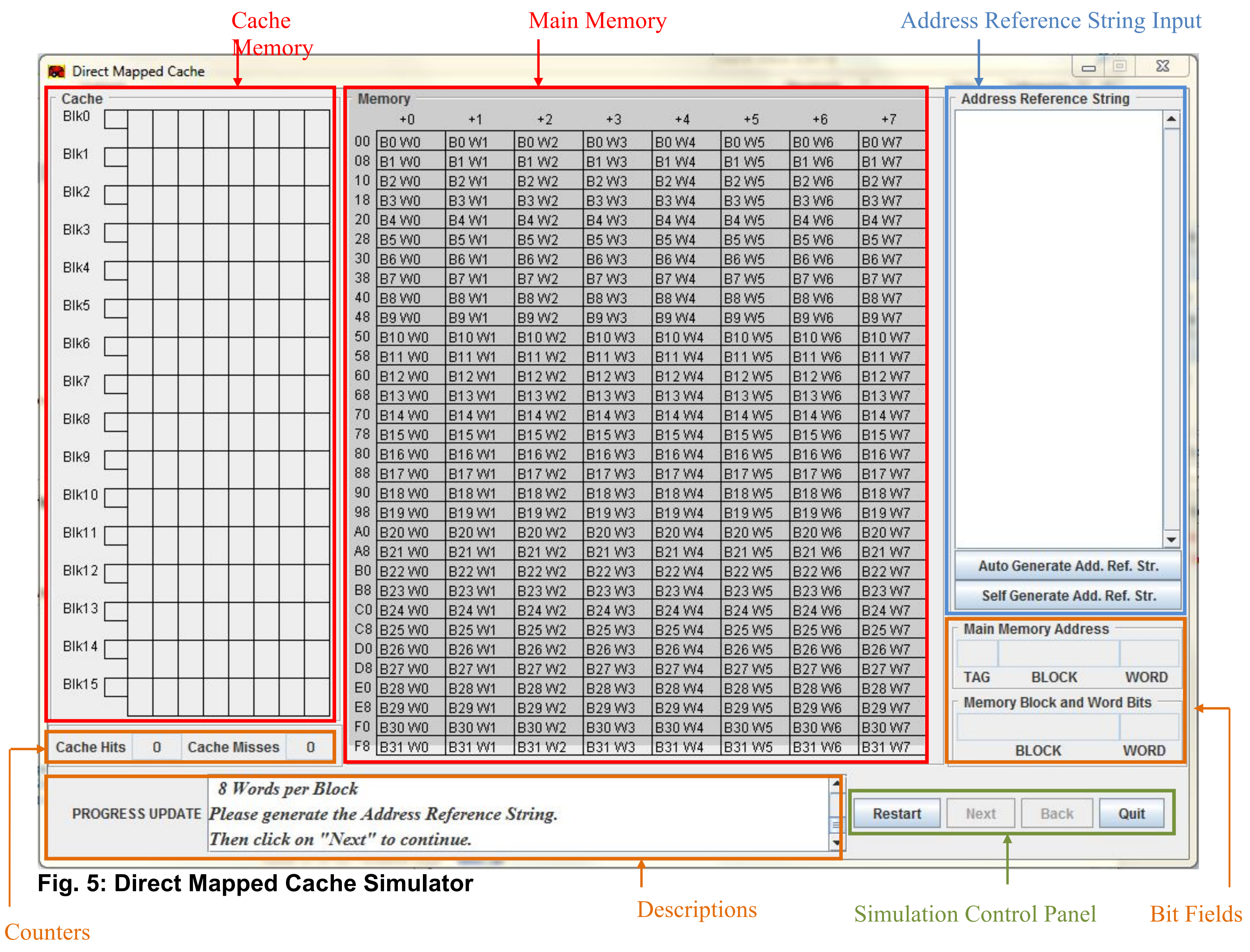


Fig. 4: The Format of Main Memory Address using Direct Mapping

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.



	+0	+1	+2	+3	+4	+5	+6	+7
00	B0 W0	B0 W1	B0 W2	B0 W3	B0 W4	B0 W5	B0 W6	B0 W7
08	B1 W0	B1 W1	B1 W2	B1 W3	B1 W4	B1 W5	B1 W6	B1 W7
10	B2 W0	B2 W1	B2 W2	B2 W3	B2 W4	B2 W5	B2 W6	B2 W7
18	B3 W0	B3 W1	B3 W2	B3 W3	B3 W4	B3 W5	B3 W6	B3 W7
20	B4 W0	B4 W1	B4 W2	B4 W3	B4 W4	B4 W5	B4 W6	B4 W7
28	B5 W0	B5 W1	B5 W2	B5 W3	B5 W4	B5 W5	B5 W6	B5 W7
30	B6 W0	B6 W1	B6 W2	B6 W3	B6 W4	B6 W5	B6 W6	B6 W7
38	B7 W0	B7 W1	B7 W2	B7 W3	B7 W4	B7 W5	B7 W6	B7 W7
40	B8 W0	B8 W1	B8 W2	B8 W3	B8 W4	B8 W5	B8 W6	B8 W7
48	B9 W0	B9 W1	B9 W2	B9 W3	B9 W4	B9 W5	B9 W6	B9 W7
50	B10 W0	B10 W1	B10 W2	B10 W3	B10 W4	B10 W5	B10 W6	B10 W7
58	B11 W0	B11 W1	B11 W2	B11 W3	B11 W4	B11 W5	B11 W6	B11 W7
60	B12 W0	B12 W1	B12 W2	B12 W3	B12 W4	B12 W5	B12 W6	B12 W7
68	B13 W0	B13 W1	B13 W2	B13 W3	B13 W4	B13 W5	B13 W6	B13 W7
70	B14 W0	B14 W1	B14 W2	B14 W3	B14 W4	B14 W5	B14 W6	B14 W7
78	B15 W0	B15 W1	B15 W2	B15 W3	B15 W4	B15 W5	B15 W6	B15 W7
80	B16 W0	B16 W1	B16 W2	B16 W3	B16 W4	B16 W5	B16 W6	B16 W7
88	B17 W0	B17 W1	B17 W2	B17 W3	B17 W4	B17 W5	B17 W6	B17 W7
90	B18 W0	B18 W1	B18 W2	B18 W3	B18 W4	B18 W5	B18 W6	B18 W7
98	B19 W0	B19 W1	B19 W2	B19 W3	B19 W4	B19 W5	B19 W6	B19 W7
A0	B20 W0	B20 W1	B20 W2	B20 W3	B20 W4	B20 W5	B20 W6	B20 W7
A8	B21 W0	B21 W1	B21 W2	B21 W3	B21 W4	B21 W5	B21 W6	B21 W7
B0	B22 W0	B22 W1	B22 W2	B22 W3	B22 W4	B22 W5	B22 W6	B22 W7
B8	B23 W0	B23 W1	B23 W2	B23 W3	B23 W4	B23 W5	B23 W6	B23 W7
C0	B24 W0	B24 W1	B24 W2	B24 W3	B24 W4	B24 W5	B24 W6	B24 W7
C8	B25 W0	B25 W1	B25 W2	B25 W3	B25 W4	B25 W5	B25 W6	B25 W7
D0	B26 W0	B26 W1	B26 W2	B26 W3	B26 W4	B26 W5	B26 W6	B26 W7
D8	B27 W0	B27 W1	B27 W2	B27 W3	B27 W4	B27 W5	B27 W6	B27 W7
E0	B28 W0	B28 W1	B28 W2	B28 W3	B28 W4	B28 W5	B28 W6	B28 W7
E8	B29 W0	B29 W1	B29 W2	B29 W3	B29 W4	B29 W5	B29 W6	B29 W7
F0	B30 W0	B30 W1	B30 W2	B30 W3	B30 W4	B30 W5	B30 W6	B30 W7
F8	B31 W0	B31 W1	B31 W2	B31 W3	B31 W4	B31 W5	B31 W6	B31 W7

Cache Hits 0 Cache Misses 0

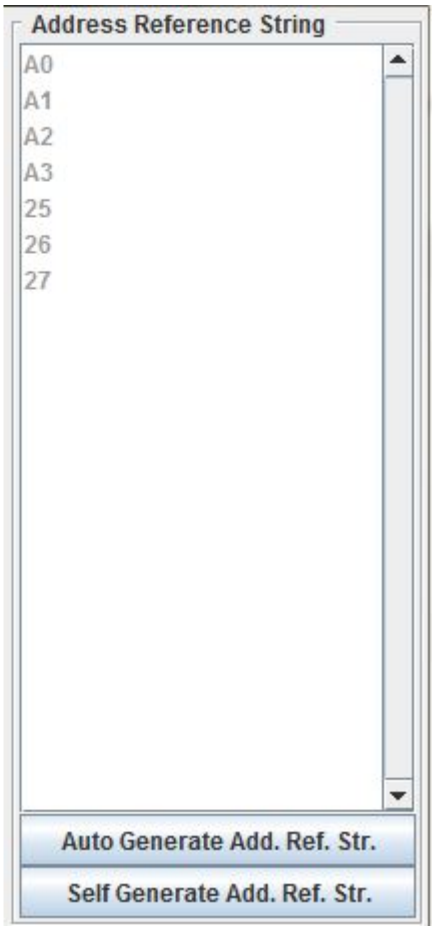
8 Words per Block
Please generate the Address Reference String.
Then click on "Next" to continue.

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. <u>Note that you will need to re-enter the address reference string again if you press ‘Restart’ button</u>.

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.


Address Reference String

A0
A1
A2
A3
25
26
27

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? <u>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.</u>

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	M	N
A1	1	0100	001	H	N
A2	1	0100	010	H	N
A3	1	0100	011	H	N
25	0	0100	101	M	Y
26	0	0100	110	H	N
27	0	0100	111	H	N

#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
- 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	m	N
38	0	0111	000	m	N
B2	1	0110	010	m	N
63	0	1100	011	M	N
22	0	0100	010	m	Y
F5	1	1110	101	m	N

#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	M	N
22	0	0100	010	m	Y
A6	1	0100	110	m	Y
23	0	0100	011	m	Y
A7	1	0100	111	m	Y
24	0	0100	100	m	Y

#Question:

- What is the reason for the high cache miss count? → Constant swap
- Is the cache utilized effectively under this scenario? → NO

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.

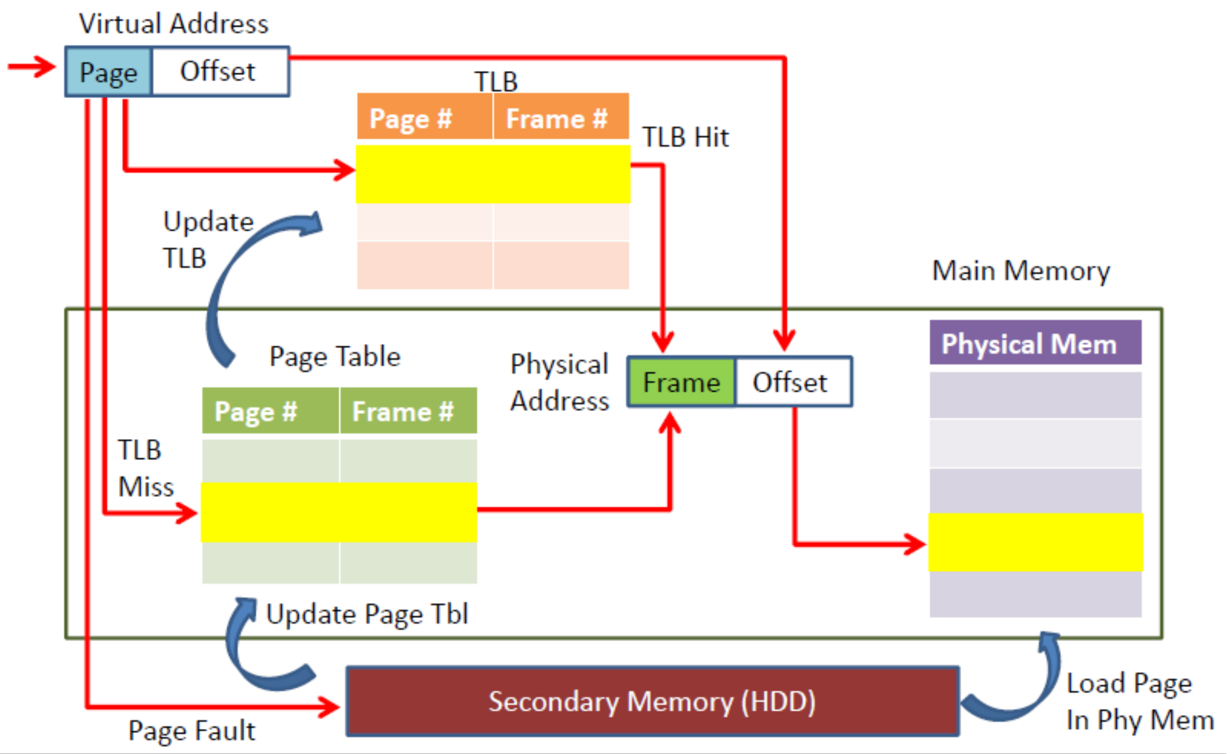
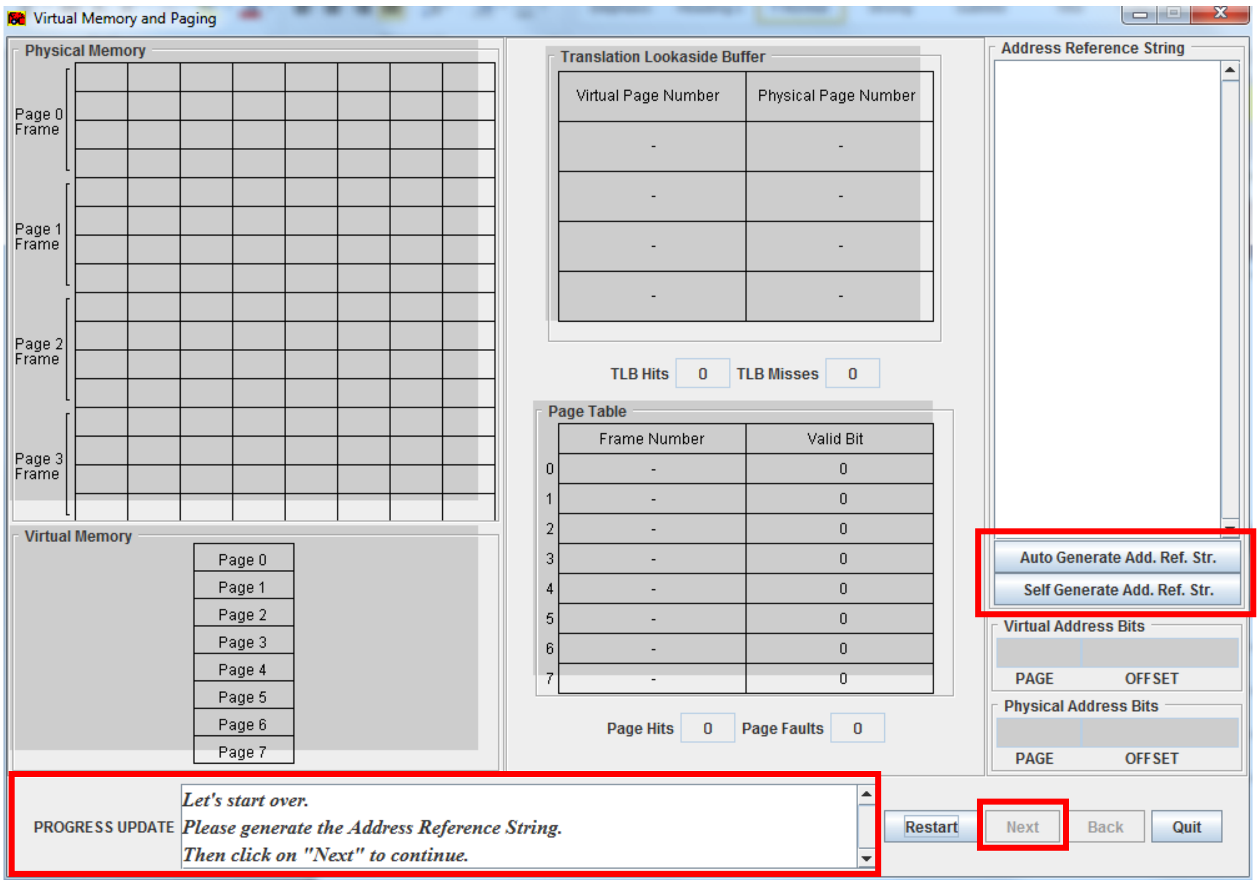


Fig. 7: Virtual Memory System

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.



Let's start over.
Please generate the Address Reference String.
Then click on "Next" to continue.

Fig. 8: Virtual Memory Simulator

C4 → 1100 0100
63 → 0110 0011

Table 6: Random Address Reference

Virtual Address	Virtual Page	Physical Frame	TLB Hit	TLB Miss	Page Hit	Page Miss	Physical Address
C4	110	00		✓		✓	00000100
63	011	01		✓		✓	01 00011
BF	101	10		✓		✓	10 11111
2A	001	11		✓		✓	11 01010
17	000	00		✓		✓	00 10111
DA	110	01		✓		✓	01 11010
A3	101	10	✓		✓		10 00011
5E	010	11		✓		✓	11 11110
41	010	11	✓		✓		11 00001
5A	010	11	✓		✓		11 11010

Table 7: Sequential Address Reference

Virtual Address	Virtual Page	Physical Frame	TLB Hit	TLB Miss	Page Hit	Page Miss	Physical Address
00	000	00		✓		✓	00 00000
01	000	00	✓		✓		00 00001
02	000	00	✓		✓		00 00010
03	000	00	✓		✓		00 00011
04	000	00	✓		✓		00 00100
56	010	01		✓		✓	01 10110
57	010	01	✓		✓		01 10111
58	010	01	✓		✓		01 11000
59	010	01	✓		✓		01 11001
5A	010	01	✓		✓		01 11010

#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? Yes. xfer
- Under what situation would a Page Hit occur in the simulator? Any Page Hit observed in the simulation conducted? Why? No page hit, as phys mem = TLB size
- 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? 6 miss
- Which Address Reference string (Table 6 or 7) is more commonly found in real life coding?

7 common

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

11/00 0000 00 11

10 0x803

01/00 00/01

10/00 01/11

10/01 00/11

11/00 01/11

10/10 00/11

01/00 00/01