CAMERA User's Guide

1 Quick Start

CAMERA is a collection of concise, intuitive and visually inspiring workbenches for cache mapping schemes and virtual memory. The CAMERA workbenches are standalone applications that can be used independently of each other and any external tools. The package is comprised of four independent Java Swing applications illustrating the following concepts:

a. Cache Mapping Schemes
   - Direct Mapped Cache
   - Fully Associative Cache
   - Set Associative Cache

b. Virtual Memory and Paging

They are most easily launched from the main menu application. To do this, all the class files must reside in the same directory.

1.1 Installation

After downloading Camera.zip, move the file to its desired location. Then unzip the files. The files may be compiled by typing the following command:

```
javac *.java
```

Issuing the following command will bring up the menu screen:

```
java Camera
```

The CAMERA workbenches are written in Java Swing and can be run on any platform on which the Java Runtime Environment *(JRE)*, version 1.3.1 or greater, resides.

1.2 Model System Specifications

The menu screen allows the user to launch any of the workbenches by clicking the appropriate button.

When the user launches an application, the progress field supplies the specifications of the system that has been modeled in the application, such as cache and memory size, virtual memory space, page and block sizes, and binary address size. In the cache mapping applications, cache size is 16 blocks; memory size is 32 blocks with 8 words per block hence the binary address is 8 bits in length. In the virtual memory application, the virtual memory space for each process has been set at 2^8 words, which translates to 8 pages of 32 words each. The physical memory contains 2^7 words hence it is divided into 4 page frames of 32 words each. The TLB can hold up to 4 entries. Where applicable, the replacement algorithm used for choosing the "victim" block is the LRU algorithm.

1.3 Beginning a Simulation

The user is required to generate an initial address reference string, which will be used to begin a simulation cycle. There are two options available to the user:

(i) Choose to let the application generate a string of 10 hexadecimal addresses in the range 00 – FF by clicking on the button, shown in Figure 17, on the left hand side of the screen:

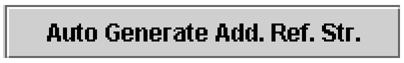


Figure 1: Auto Generate String

The "Auto Generate Add. Ref. Str." button may be clicked repeatedly before clicking "Next" if the user wishes to generate more than 10 addresses at once.

(ii) Enter individual address references by clicking on the button, illustrated in Figure 18, on the left hand side of the screen:

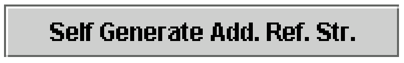


Figure 2: Self Generate String

The user can then proceed to enter up to 10 valid hexadecimal addresses ranging from 00 – FF. (If more addresses are desired, additional addresses can be added later.)

In the set associative cache mapping simulator, there is an additional option for specifying the number of "ways" (cache blocks per set) in the cache. The user is prompted to pick one of the available choices and the cache is redesigned dynamically. This option can be exercised by clicking on the button, illustrated in Figure 19, located under the cache on the lower left area of the screen:

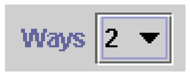


Figure 3: Number of Ways

These generation buttons and the button for selecting the number of ways are disabled during a simulation cycle so that the flow of the simulation cannot be disturbed. They are enabled again after the address reference string has been exhausted. Clicking on the “Restart” button can allow the user to stop the current simulation and bring the simulator back to starting position if so desired. The “Restart” button may be used at any time to start afresh, that is, clear the contents of the cache and the address reference string, and reinitialize the count of cache hits and misses so the user can start a new simulation with a new address reference string.

At the end of a simulation cycle, when the address string has been exhausted, the generation buttons are re-enabled to permit the user to generate another set of strings. In this case, the cache is not cleared and the statistics on cache hits and misses are not reinitialized. Since the application restricts the number of addresses generated each time to a maximum of 10, this feature of the address generation buttons allows a user to observe simulation cycles following one after another and thus model a real life example of a set of addresses generated through the execution of a program. The addresses can be added on at the end of each simulation cycle while retaining the state of the memory hierarchy.

1.4 Stepping Through a Simulation

Once the address reference string has been created, the user can step through a simulation cycle while observing the process followed by the CPU to obtain data residing at the address specified by the address reference. The navigation buttons shown in Figure 20 are located at the bottom of the application screen:

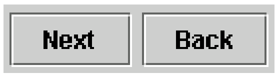
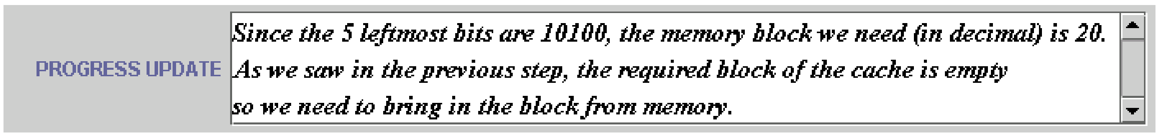


Figure 4: Navigation Buttons

The “Next” and “Back” buttons may be used, as their names suggest, for stepping forward to the next stage or stepping backward to the previous stage in the simulation.

At each step, the architectural element being accessed by the CPU is highlighted, demonstrating the process. Additionally, the progress field located at the bottom of the screen explains the steps textually and guides the user to the next step:



Since the 5 leftmost bits are 10100, the memory block we need (in decimal) is 20.
As we saw in the previous step, the required block of the cache is empty
so we need to bring in the block from memory.

Figure 5: Progress Field

2 The Details

2.1 Introduction

CAMERA is a collection of concise, intuitive and visually inspiring workbenches for cache mapping schemes and virtual memory. The best way to learn any new concept is hands-on experience, and CAMERA’s primary goal is to provide students with the means to step along while it demonstrates various concepts of cache mapping and virtual memory. Workbenches, by their very nature, lend themselves to exploration and with CAMERA, students can modify some basic features of the components and follow their changes through the cycle of the application. Although a well-written book with diagrams and illustrations can be very educational, a simulation that models the actual process and also allows user input and manipulation can provide an entirely new perspective to learning. CAMERA was developed to satisfy this need for a dynamic new instructional tool. It gives students graphical representations of the memory system and furnishes them with appropriately flexible interfaces to observe and experiment with the system.

Another objective for developing CAMERA was to use it in conjunction with the memory architecture simulator, MarieSim (also available on the textbook web page). Toward the goal of helping students understand how computers really work, we created a memory architecture called MARIE and made it as simple as possible, hence the name *Machine Architecture that is Really Intuitive and Easy*. Real architectures, although interesting, often have far too many peculiarities to make them usable in an introductory class. To alleviate these problems, we designed MARIE specifically for pedagogical use. MARIE allows students to learn the essential concepts of computer organization and architecture, including assembly language, without getting caught up in the unnecessary and confusing details that exist in real architectures. Despite its simplicity, it simulates a functional system. The MARIE machine simulator, MarieSim, has a user-friendly GUI that allows students to: (1) create and edit source code; (2) assemble source code into machine object code; (3) run machine code; and, (4) debug programs. It is an environment within which users can write their own programs and watch how they would be run in a real "von Neumann architecture" computer system. This way, they would get to see their programs in action, and get a taste of how to program in an assembler language, without the need to learn any particular assembly language. Together, MarieSim and CAMERA make up a very nice learning tool set for memory architecture concepts.

2.2 Cache Mapping

The three cache mapping schemes depicted in CAMERA are detailed below.

Direct Mapped Cache

In this mapping scheme, each block of memory is mapped to exactly one cache block in a modular fashion. This mapping is determined by the memory address bits that are, in turn, dependent on the physical characteristics of main memory and cache. The memory address is partitioned into three fields: *tag*, *block*, and *word (or offset)*. The least significant bits form the word field, which identifies a word in a specific block. The middle bits form the block field, which specifies the cache block to which this memory address maps. The remaining bits form the tag field, which uniquely identifies a memory block in cache. Since there are more blocks in memory than in cache, it is obvious that there is contention for the cache blocks. The tag field is stored with each memory block when it is placed in cache so the block can be uniquely identified. When cache is searched for a specific memory block, the CPU knows exactly where to find the block just by looking at the main memory address bits.

Fully Associative Cache

Instead of specifying a unique location for each main memory block, we can look at the opposite extreme: allowing a block of memory to be placed anywhere in cache. To implement this mapping scheme, we require associative memory so it can be searched in parallel. The memory address is partitioned into two fields: *tag* and *word*. The least significant bits form the word field, which identifies a word in a specific block. The remaining bits form the tag field, which uniquely identifies a memory block in cache and is stored with the main memory block in cache. When cache is searched for a specific memory block, the tag field of the main memory block is compared to all the valid tags in cache and, if a match is found, the block is found. If there is no match, the memory block needs to be brought into the cache. As mentioned before, since the number of blocks in memory exceeds the number of blocks in cache, there will be contention for the cache blocks. While there are empty blocks in cache, the memory blocks are placed in the nearest available free cache block. Once cache is full, a replacement algorithm is used to evict an existing memory block from cache and place the new memory block in its place. In CAMERA, the replacement algorithm used is the *least recently used (LRU)* algorithm.

Set Associative Cache

The third mapping scheme is N-way set associative cache mapping. It is similar to direct mapped cache because we use the memory address to map to a cache location. The difference is that an address maps to a set of cache blocks instead of a single cache block. The memory address is partitioned into three fields: *tag, set,* and *word*. The tag and word fields behave the same as before. The set field indicates into which cache set the memory block maps. When cache is searched for a specific memory block, the CPU knows to look in a specific cache set with the help of the main memory address bits. The tag bits then identify the memory block. A replacement algorithm is needed to determine the “victim” block that will be removed from cache to make available free space for a new memory block. In CAMERA, the replacement algorithm used is the *LRU* algorithm.

2.3 Virtual Memory and Paging

Camera also provides a workbench to reinforce the concepts in virtual memory and paging. Physical memory, virtual memory, the process page table, and the TLB are all available for inspection as students progress though 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.

2.4 CAMERA Features

CAMERA has many distinctive features:

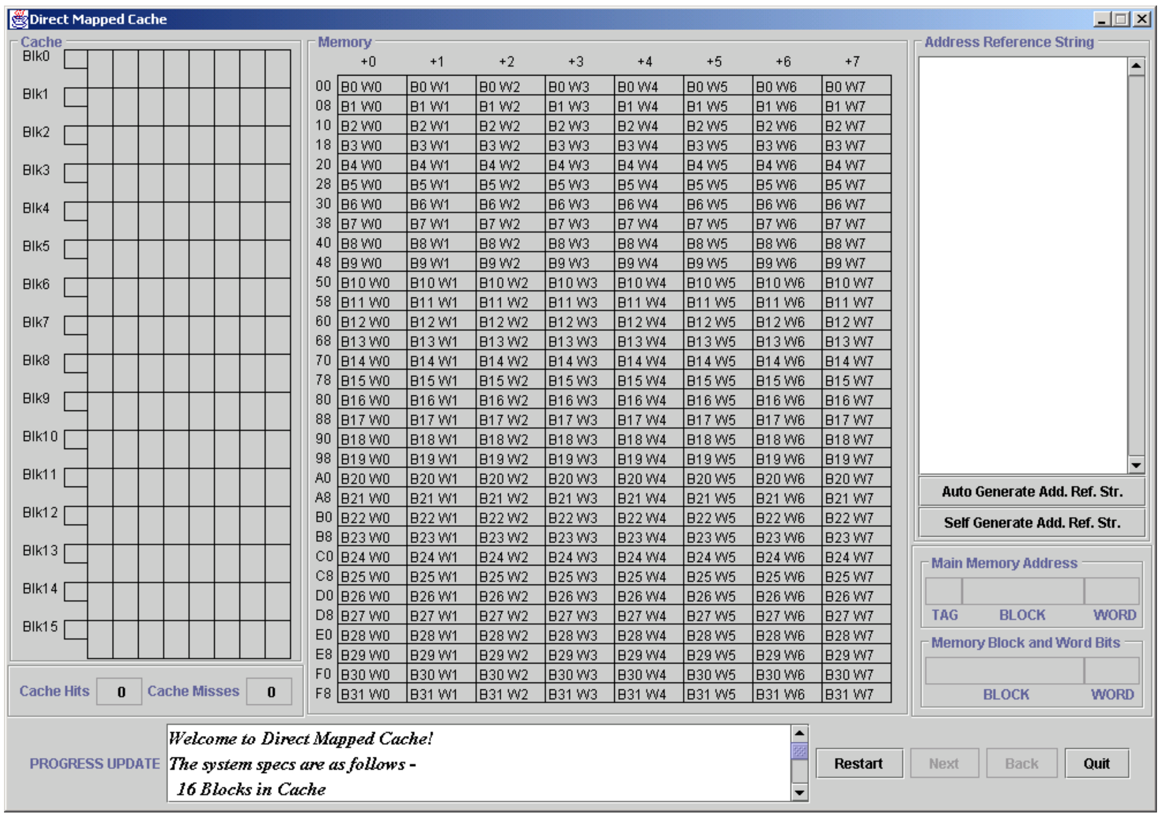
- Progression through each simulation is illustrated with visual effects and reinforced by the current updates offered by the progress status field.
- Along with the detailed explanation the progress field presents, it also provides guidance to the user on how to proceed to the next step.
- The main memory address retrieved from the address reference string is converted to its binary equivalent at each step to clearly show the user the mapping between the memory address and its corresponding cache location.
- The navigation buttons provide easy maneuverability back and forth so the user doesn't overlook any important information during a simulation cycle.
- The user is offered several choices in the generation of the address reference string.
- Each workbench represents a snapshot of the memory architecture so the user can perceive it in its entirety in one glance. There are no scrolling controls and no multiple windows that clutter the interface.
- CAMERA, with its workbenches for cache mapping as well as virtual memory, is a comprehensive package of applications for memory management concepts.

- The workbenches are great teaching aids and can assist instructors in explaining memory management concepts and in providing students with interesting tools to further their knowledge of the same.

2.4 CAMERA Functionality

The following screenshots outline the sequence of simulations in CAMERA.

Since the cache mapping simulators have a very similar look and feel, only one of them will be addressed in detail. To assist in clearly describing the cache mapping applications, a walkthrough of Direct Mapped Cache is shown in Figures 6 through 11:



	+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

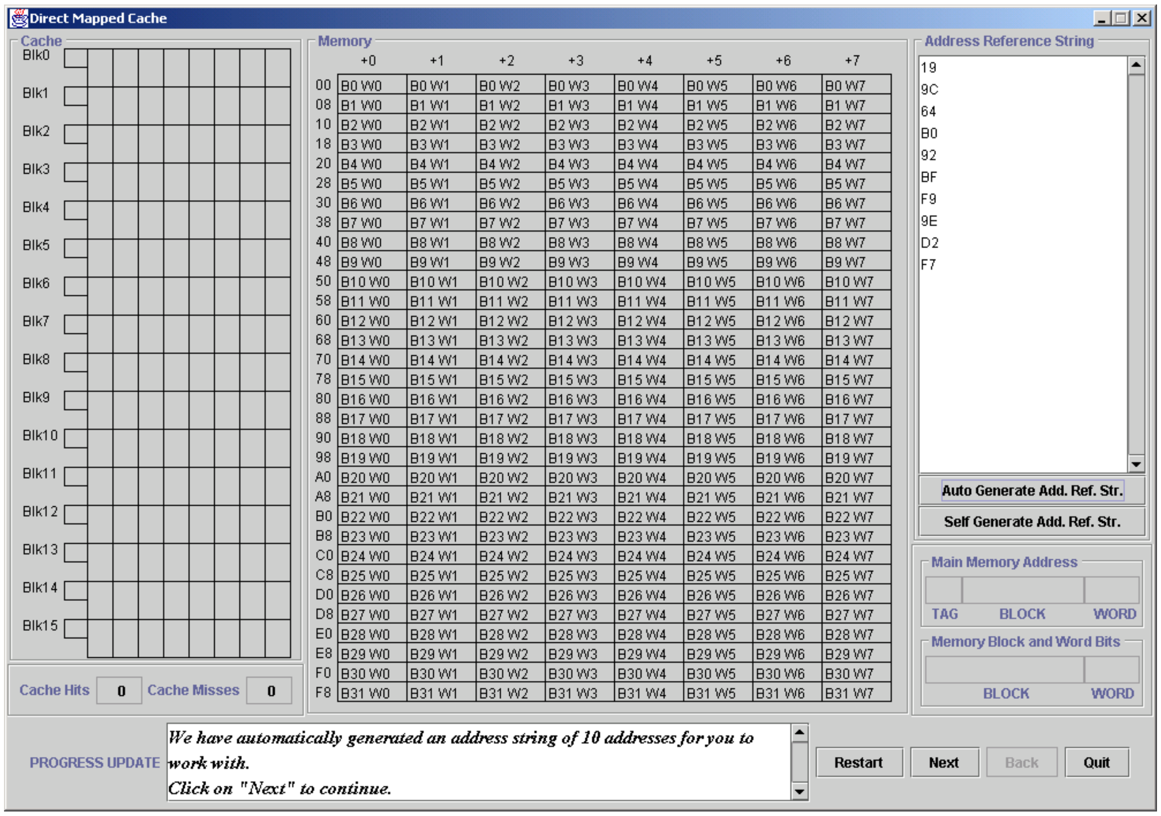
PROGRESS UPDATE *Welcome to Direct Mapped Cache!*
The system specs are as follows -
16 Blocks in Cache

Figure 6: Direct Mapped Cache - Introduction

On the start of the simulation, the cache is clear and the user is prompted to generate an address reference string. There are two options for creation of the string:

(i) The user can request that application generate an address reference string of a predefined length. This string would be a segment of an actual address string created during execution of a simple program. (This information can be retrieved from MarieSim)

(ii) The user can enter single valid addresses one at a time up to a predefined maximum. The main goal of this feature is to allow a user to run a sample program in MarieSim, capture the address reference string from it, and then execute CAMERA's workbenches to view the flow of data between memory and cache and understand mapping concepts.



	+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

19
9C
64
B0
92
BF
F9
9E
D2
F7

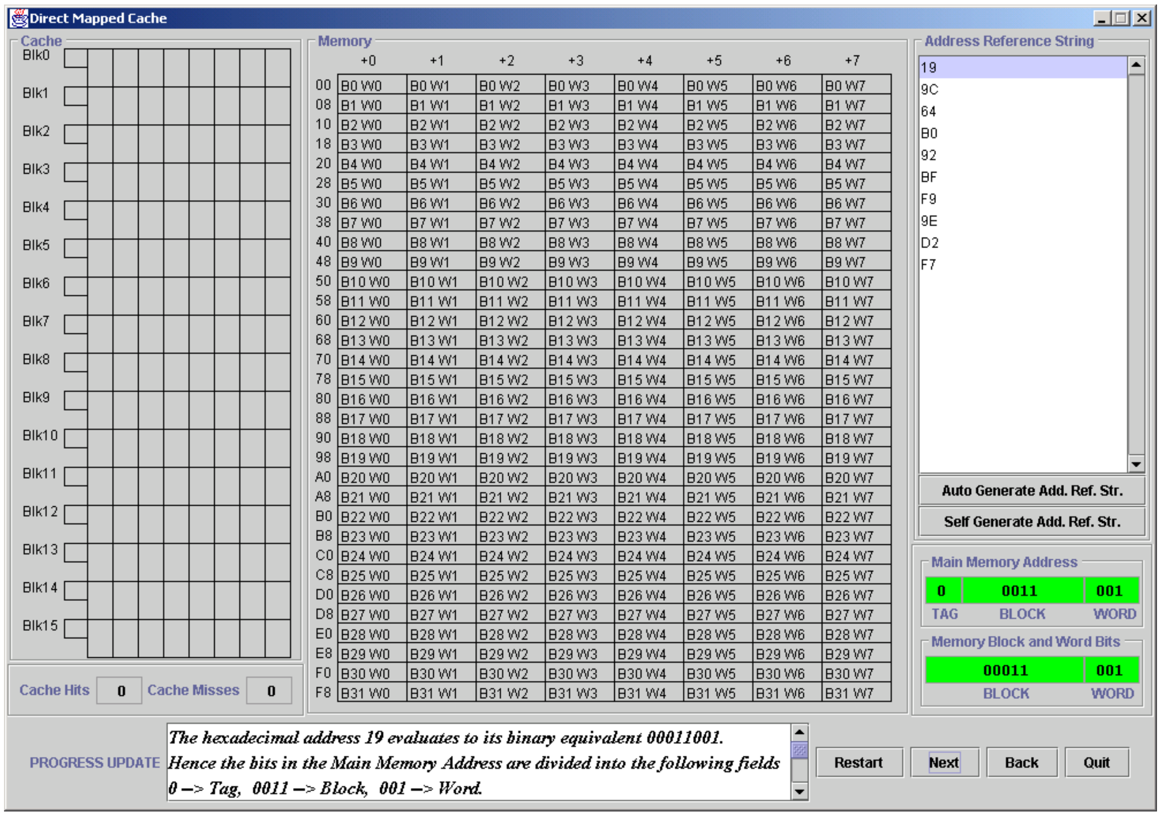
Cache Hits 0 Cache Misses 0

We have automatically generated an address string of 10 addresses for you to work with.
Click on "Next" to continue.

Figure 7: Direct Mapped Cache – Address Reference String

Once the address reference string is created, the simulation can begin. As the user steps forward using the "Next" button, the application retrieves an address from the string, converts it into its binary equivalent and displays the bits, which are partitioned into the tag, block, set and word fields as appropriate to that particular cache mapping scheme.

In the case of Direct Mapped Cache, the main memory address bits are divided into tag, block and word.



Cache Hits 0 Cache Misses 0

	+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

```
19
9C
64
B0
92
BF
F9
9E
D2
F7
```

Main Memory Address

0	0011	001
TAG	BLOCK	WORD

Memory Block and Word Bits

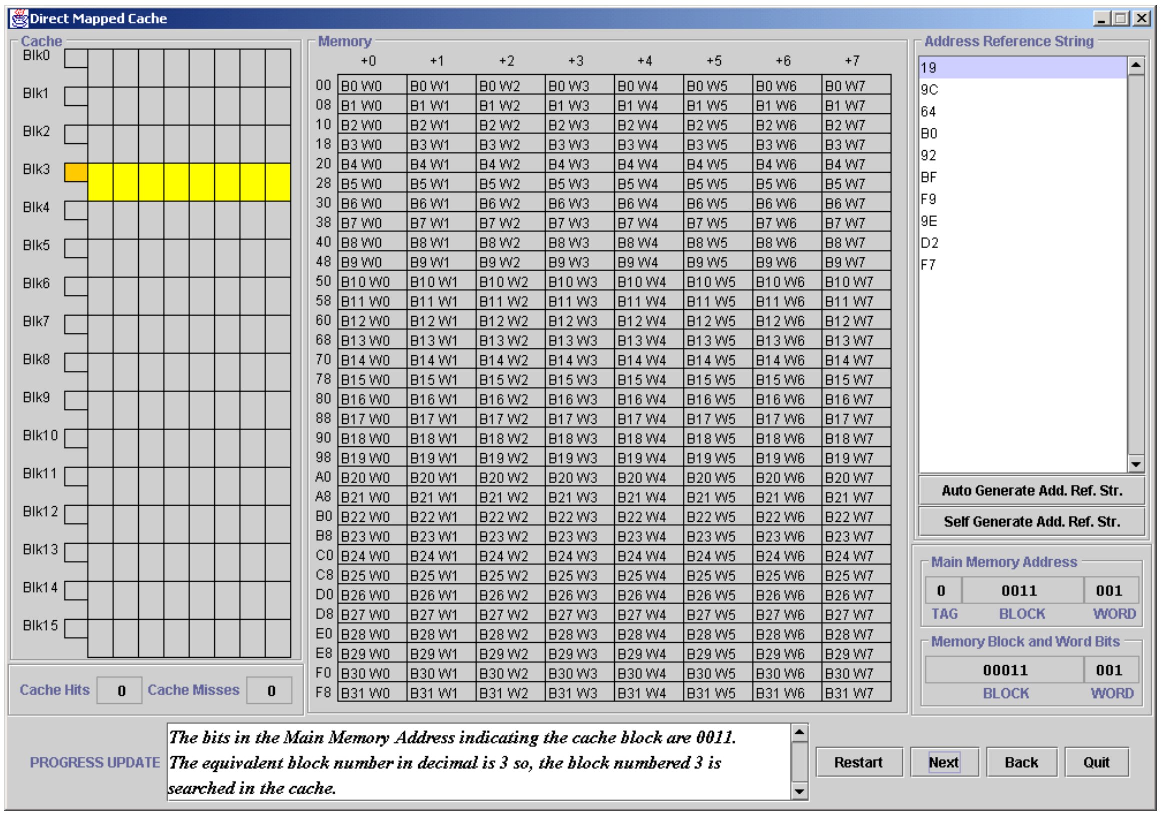
00011	001
BLOCK	WORD

The hexadecimal address 19 evaluates to its binary equivalent 00011001.
Hence the bits in the Main Memory Address are divided into the following fields
0 —> Tag, 0011 —> Block, 001 —> Word.

Figure 8: Direct Mapped Cache – Address Bits

There are two sets of bits highlighted in green. The bits on top are the actual generated main memory address bits that indicate the cache location to which the memory block maps. The bits below are not part of the architecture, but simply serve an informational purpose, indicating the main memory block and word in question.

Now that the cache location is determined, the CPU looks in the appropriate cache location for the memory block.



Memory

	+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

Address Reference String: 19, 9C, 64, B0, 92, BF, F9, 9E, D2, F7

Main Memory Address

TAG	BLOCK	WORD
0	0011	001

Memory Block and Word Bits

BLOCK	WORD
00011	001

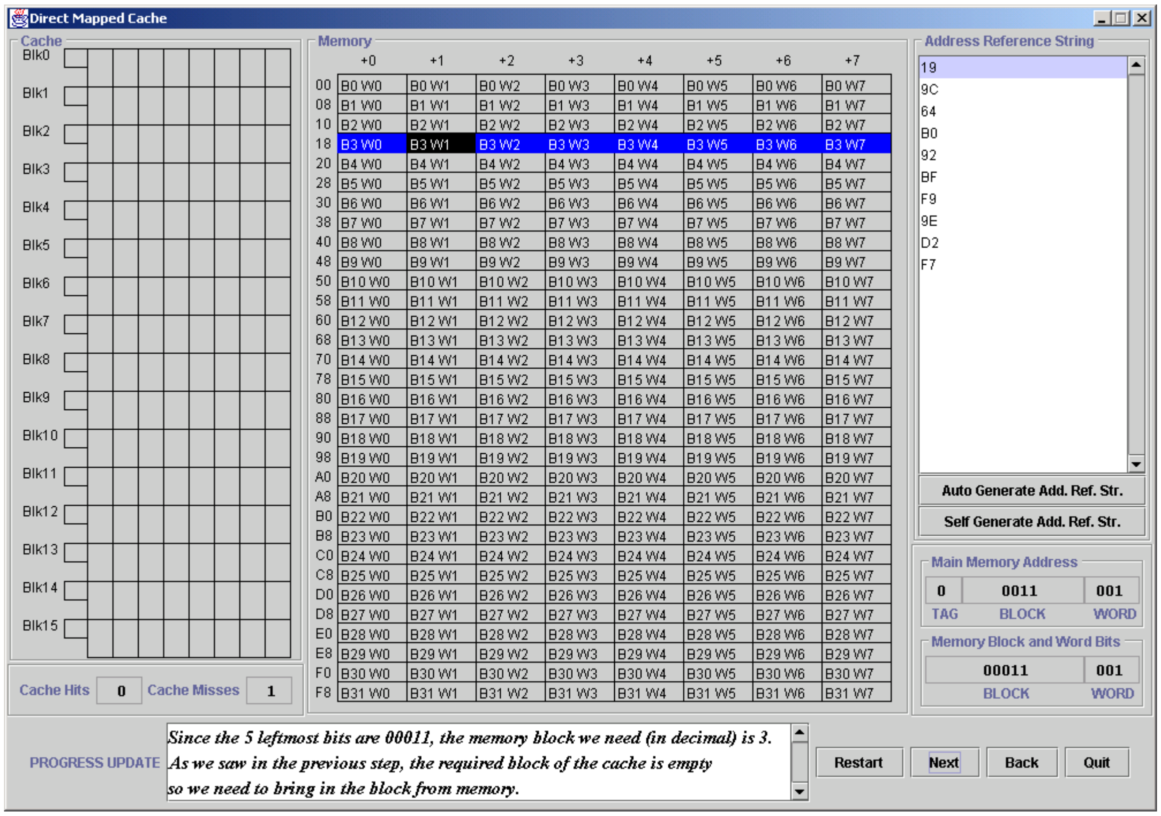
Cache Hits 0 Cache Misses 0

The bits in the Main Memory Address indicating the cache block are 0011. The equivalent block number in decimal is 3 so, the block numbered 3 is searched in the cache.

Figure 9: Direct Mapped Cache – Cache Block (empty)

The cache block accessed is highlighted in yellow, and the tag in orange. In this example, there is a cache miss because the required cache block is empty. The count of cache misses is incremented.

Now the memory block needs to be brought into cache. Therefore the CPU goes to memory to access the memory block.



	+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

Address Reference String: 19, 9C, 64, B0, 92, BF, F9, 9E, D2, F7

Cache Hits: 0 Cache Misses: 1

Main Memory Address

TAG	BLOCK	WORD
0	0011	001

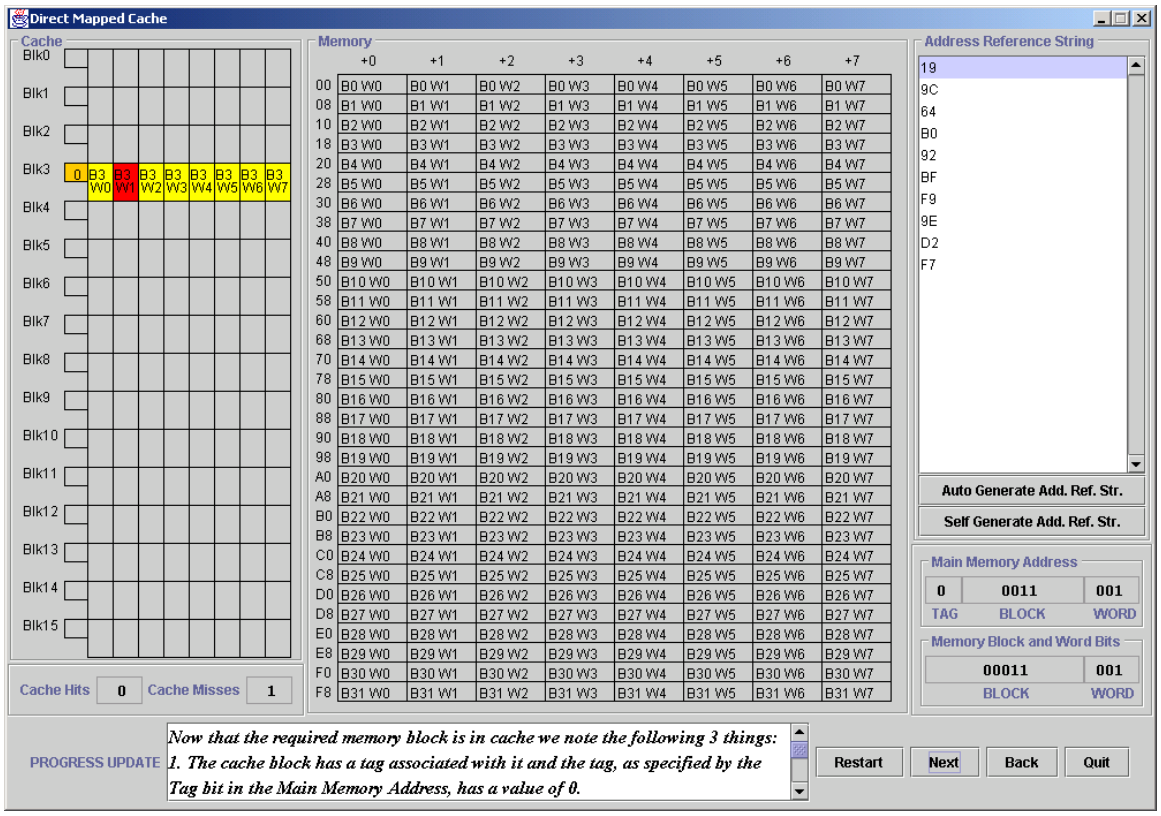
Memory Block and Word Bits

BLOCK	WORD
00011	001

Since the 5 leftmost bits are 00011, the memory block we need (in decimal) is 3. As we saw in the previous step, the required block of the cache is empty so we need to bring in the block from memory.

<u>Figure 10: Direct Mapped Cache – Memory Block</u>

The memory block is highlighted in blue, and the required word is highlighted in black. This block of memory is now transferred into cache.



Cache

Blk0								
Blk1								
Blk2								
Blk3	0	B3 W0	B3 W1	B3 W2	B3 W3	B3 W4	B3 W5	B3 W6	B3 W7
Blk4								
Blk5								
Blk6								
Blk7								
Blk8								
Blk9								
Blk10								
Blk11								
Blk12								
Blk13								
Blk14								
Blk15								

Cache Hits 0 Cache Misses 1

Memory

	+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

Address Reference String

19
9C
64
B0
92
BF
F9
9E
D2
F7

Main Memory Address

TAG	BLOCK	WORD
0	0011	001

Memory Block and Word Bits

BLOCK	WORD
00011	001

PROGRESS UPDATE

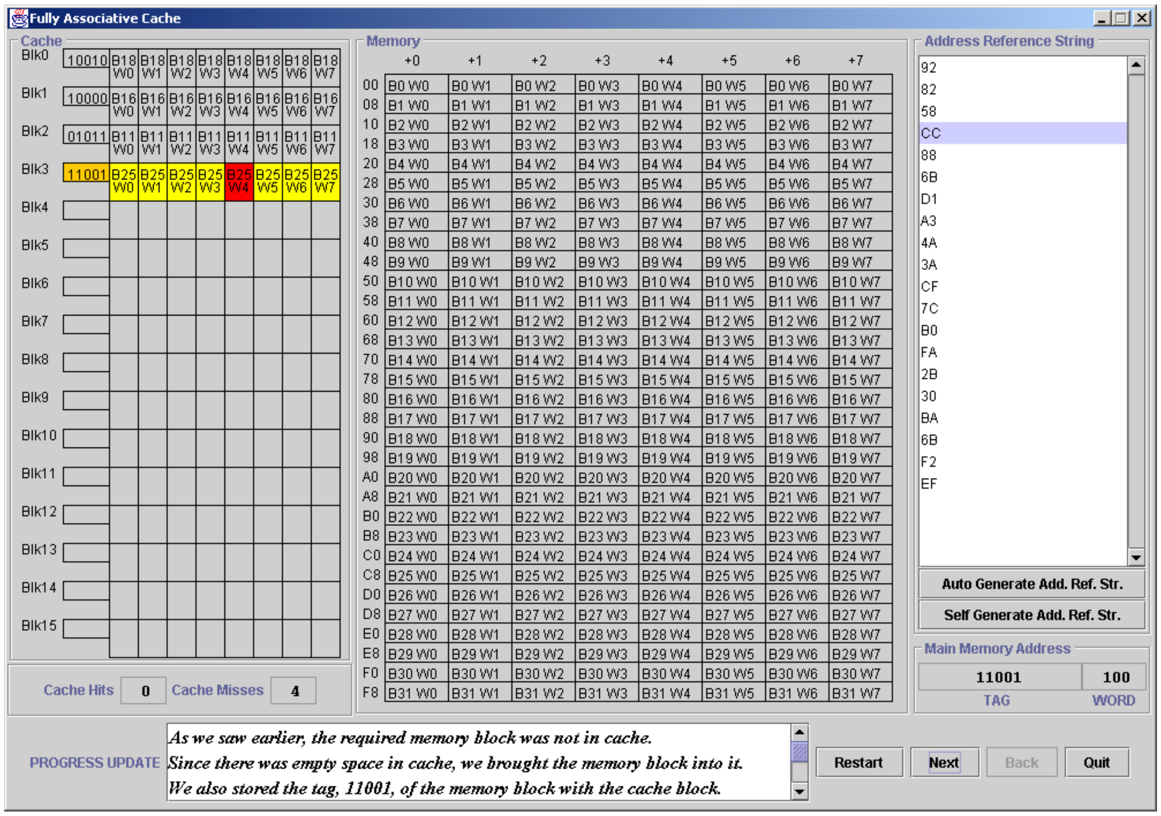
Now that the required memory block is in cache we note the following 3 things:
1. The cache block has a tag associated with it and the tag, as specified by the Tag bit in the Main Memory Address, has a value of 0.

Figure 11: Direct Mapped Cache – Cache Block (full)

The cache now contains the memory block and the CPU can now access the main memory block from here. The required block is highlighted in yellow while the word is highlighted in red. The tag field is also updated and highlighted in orange.

The simulators for fully associative cache and set associative cache have much of the same functionality as direct mapped cache in the way that they illustrate the process starting from the generation of the memory address to the mapping of memory to cache and the retrieval of the memory block. The only differences in the interfaces lie in the partitioning of the bits in main memory address and the cache design.

Here is a screenshot of one of the simulation phases in Fully Associative Cache:



Cache

Blk0	10010	B18 W0	B18 W1	B18 W2	B18 W3	B18 W4	B18 W5	B18 W6	B18 W7
Blk1	10000	B16 W0	B16 W1	B16 W2	B16 W3	B16 W4	B16 W5	B16 W6	B16 W7
Blk2	01011	B11 W0	B11 W1	B11 W2	B11 W3	B11 W4	B11 W5	B11 W6	B11 W7
Blk3	11001	B25 W0	B25 W1	B25 W2	B25 W3	B25 W4	B25 W5	B25 W6	B25 W7
Blk4									
Blk5									
Blk6									
Blk7									
Blk8									
Blk9									
Blk10									
Blk11									
Blk12									
Blk13									
Blk14									
Blk15									

Cache Hits 0 Cache Misses 4

Memory

	+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

Address Reference String

92
82
58
CC
88
6B
D1
A3
4A
3A
CF
7C
B0
FA
2B
30
BA
6B
F2
EF

Main Memory Address

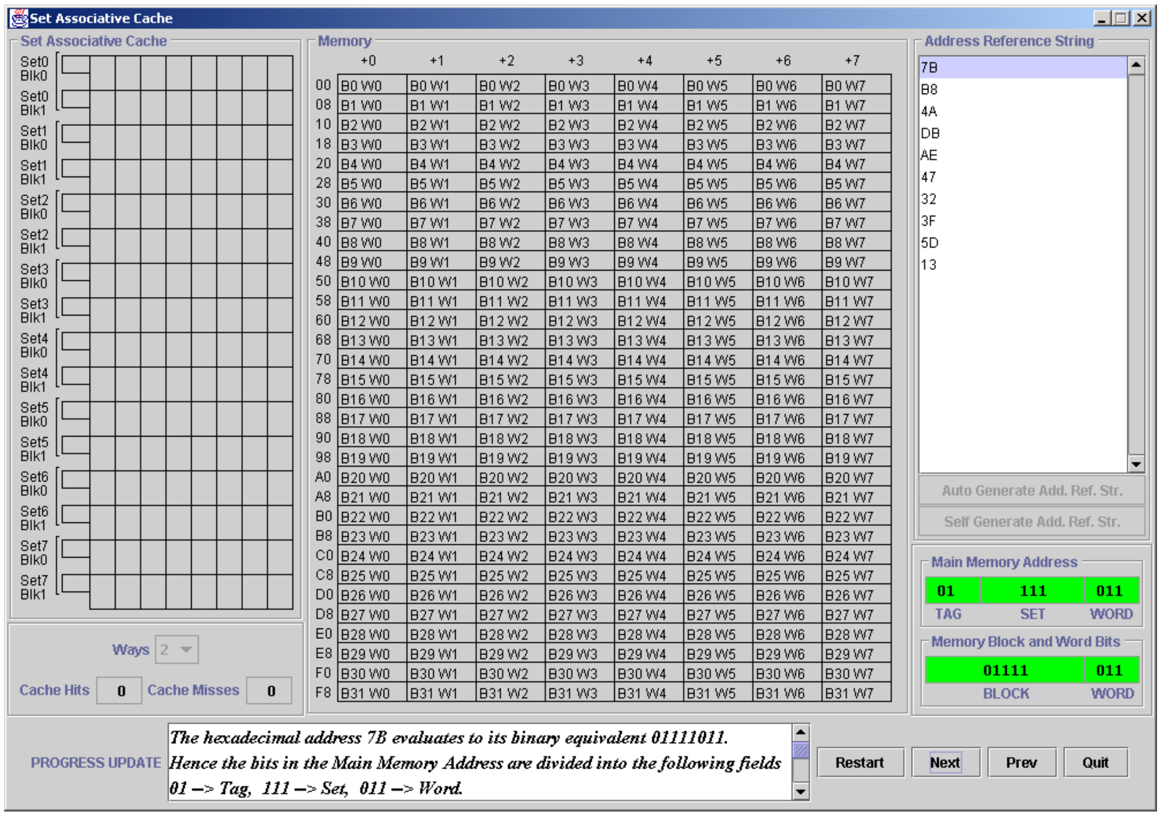
TAG	WORD
11001	100

PROGRESS UPDATE

As we saw earlier, the required memory block was not in cache.
Since there was empty space in cache, we brought the memory block into it.
We also stored the tag, 11001, of the memory block with the cache block.

Figure 12: Fully Associative Cache

Here is a screenshot of the interface for 2-way Set Associative Cache:



Ways: 2

Cache Hits: 0 Cache Misses: 0

	+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

Address Reference String:

```
7B
B8
4A
DB
AE
47
32
3F
5D
13
```

Main Memory Address

TAG	SET	WORD
01	111	011

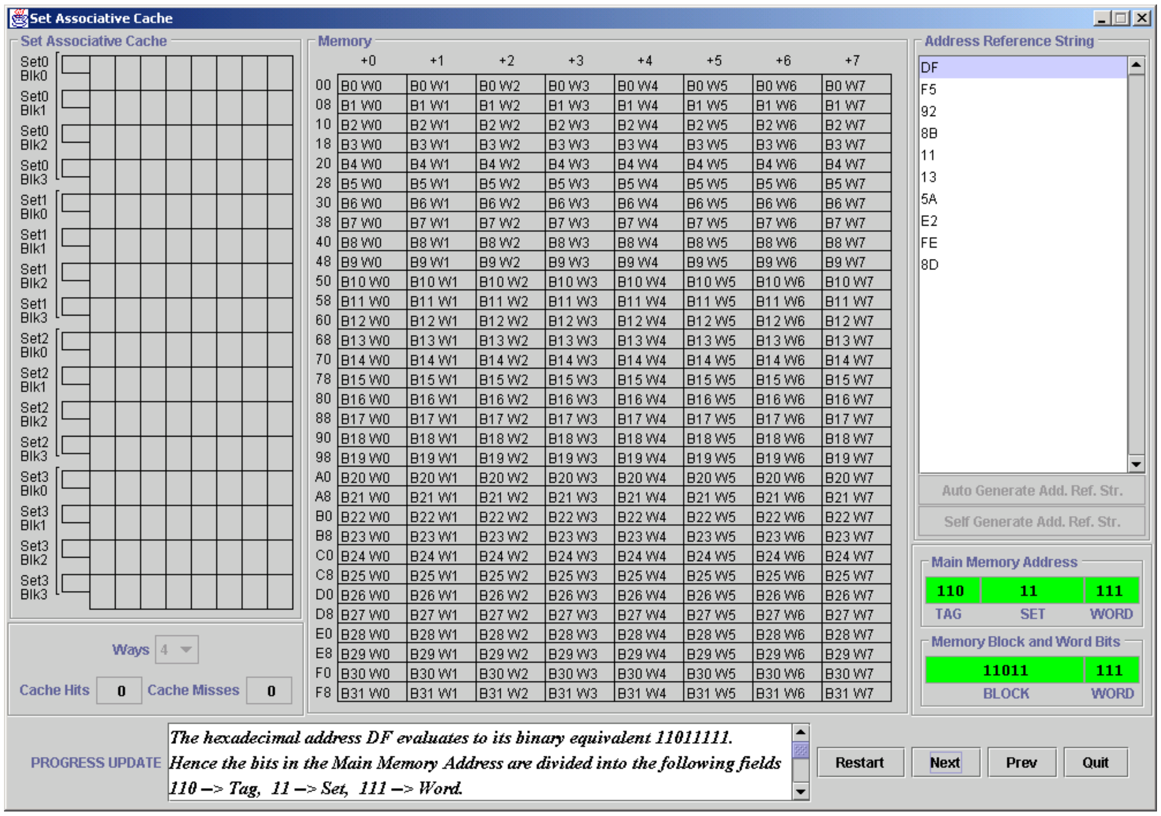
Memory Block and Word Bits

BLOCK	WORD
01111	011

PROGRESS UPDATE: *The hexadecimal address 7B evaluates to its binary equivalent 01111011. Hence the bits in the Main Memory Address are divided into the following fields 01 –> Tag, 111 –> Set, 011 –> Word.*

Figure 13: 2-way Set Associative Cache

Here is a screenshot of the interface for 4-way Set Associative Cache:


Set Associative Cache

Set Associative Cache

Set0 Blk0, Set0 Blk1, Set0 Blk2, Set0 Blk3, Set1 Blk0, Set1 Blk1, Set1 Blk2, Set1 Blk3, Set2 Blk0, Set2 Blk1, Set2 Blk2, Set2 Blk3, Set3 Blk0, Set3 Blk1, Set3 Blk2, Set3 Blk3

Ways 4

Cache Hits 0 Cache Misses 0

Memory

	+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

Address Reference String

DF
F5
92
8B
11
13
5A
E2
FE
8D

Auto Generate Add. Ref. Str.

Self Generate Add. Ref. Str.

Main Memory Address

110	11	111
TAG	SET	WORD

Memory Block and Word Bits

11011	111
BLOCK	WORD

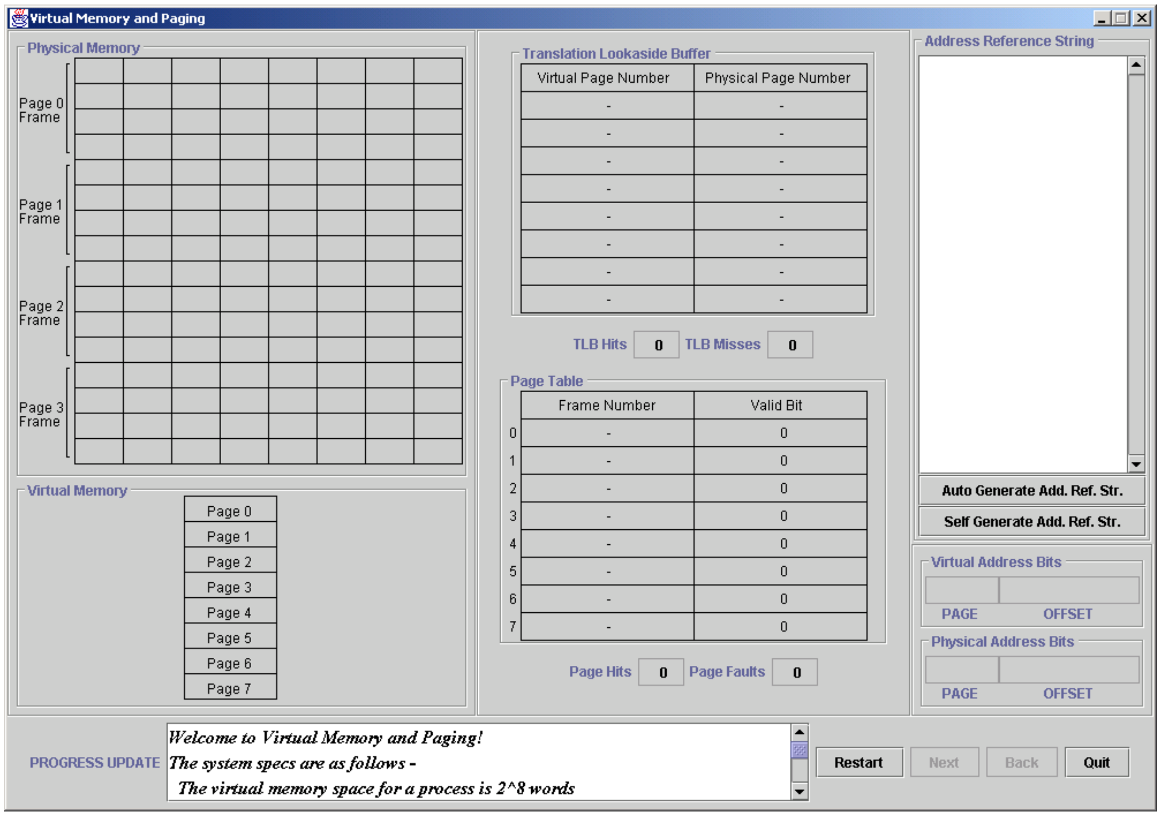
PROGRESS UPDATE

The hexadecimal address DF evaluates to its binary equivalent 11011111.
Hence the bits in the Main Memory Address are divided into the following fields
110 --> Tag, 11 --> Set, 111 --> Word.

Restart Next Prev Quit

Figure 14: 4-way Set Associative Cache

To assist in clearly describing the Virtual Memory simulator, Figures 15 through 21 outline the sequence of a simulation:



Virtual Page Number	Physical Page Number
-	-
-	-
-	-
-	-
-	-
-	-
-	-
-	-

TLB Hits 0 TLB Misses 0

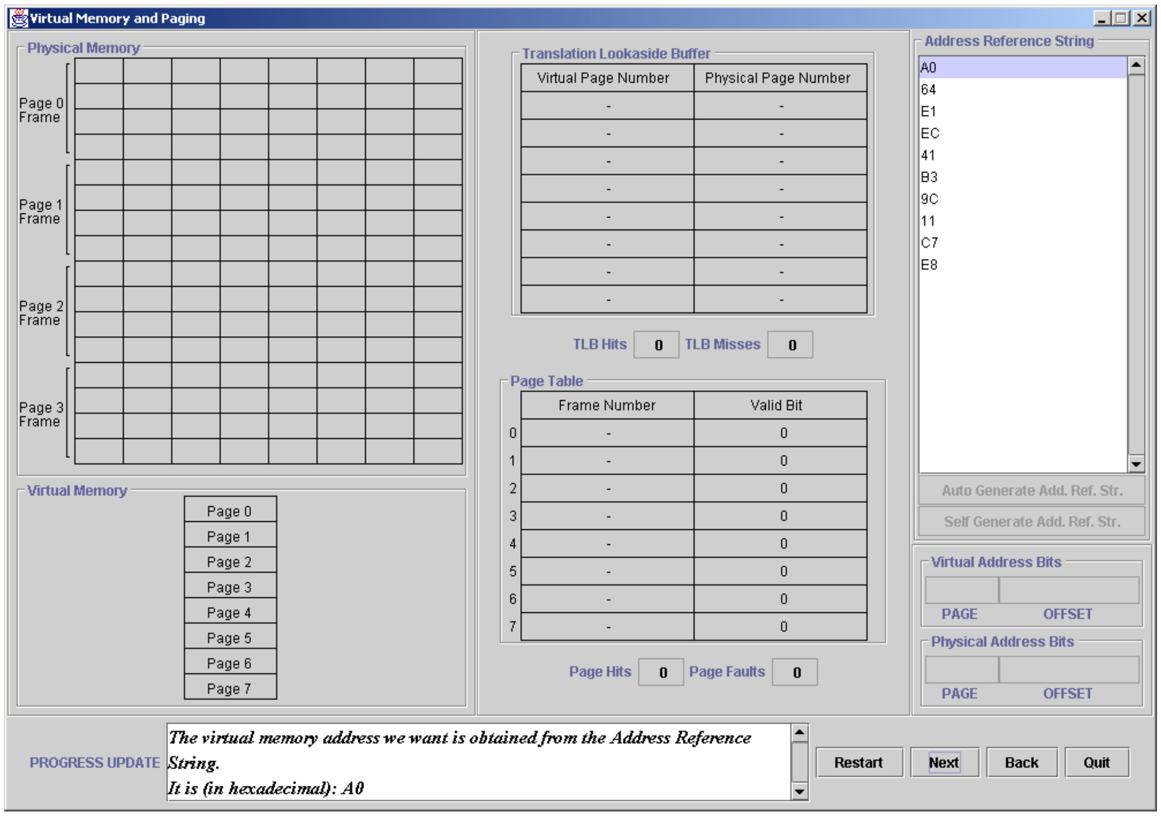
	Frame Number	Valid Bit
0	-	0
1	-	0
2	-	0
3	-	0
4	-	0
5	-	0
6	-	0
7	-	0

Page Hits 0 Page Faults 0

Welcome to Virtual Memory and Paging!
The system specs are as follows -
The virtual memory space for a process is 2^8 words

Figure 15: Virtual Memory - Introduction

On the start of the simulation, the physical memory is empty and the user is prompted to generate an address reference string. The two options available are the same as in the cache mapping tutorials.



Translation Lookaside Buffer

Virtual Page Number	Physical Page Number
-	-
-	-
-	-
-	-
-	-
-	-
-	-
-	-

TLB Hits 0 TLB Misses 0

Page Table

	Frame Number	Valid Bit
0	-	0
1	-	0
2	-	0
3	-	0
4	-	0
5	-	0
6	-	0
7	-	0

Page Hits 0 Page Faults 0

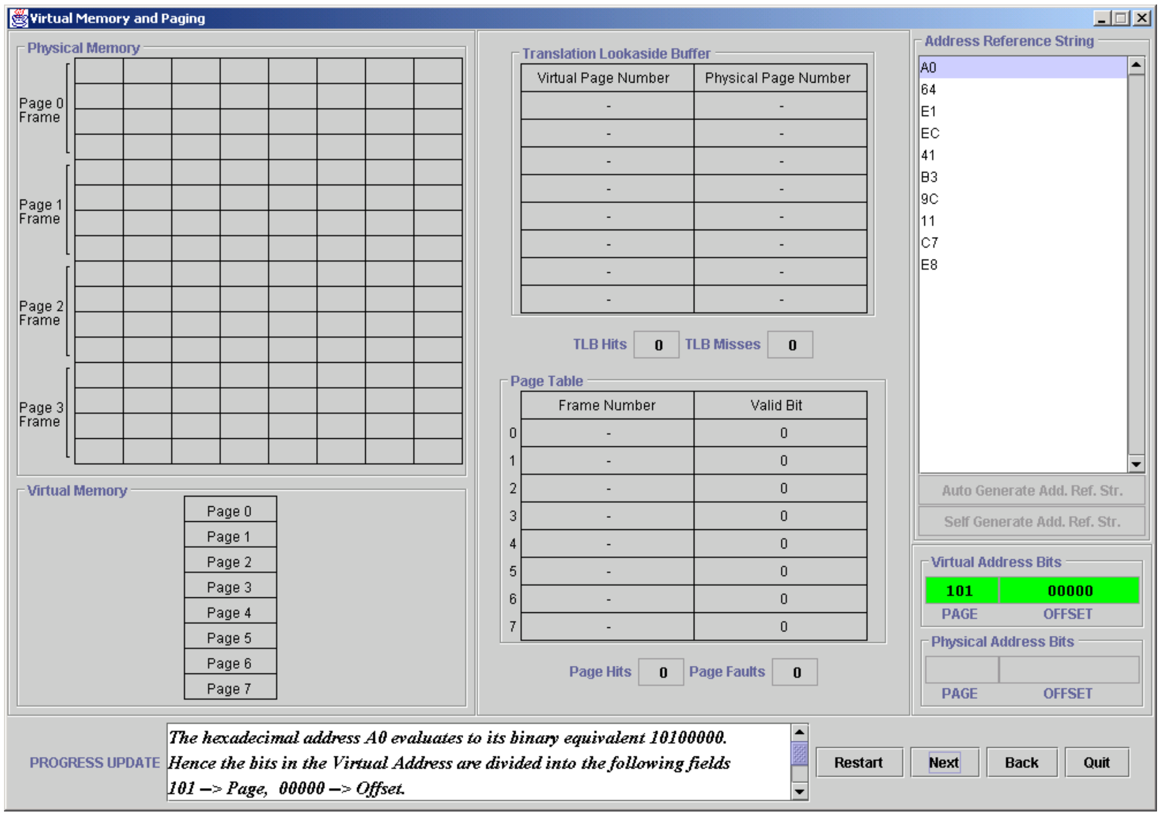
Address Reference String

A0
64
E1
EC
41
B3
9C
11
C7
E8

PROGRESS UPDATE *The virtual memory address we want is obtained from the Address Reference String.*
It is (in hexadecimal): A0

Figure 16: Virtual Memory – Address Reference String

Once the address reference string is created, the simulation can begin. As the user steps forward using the “Next” button, the application retrieves a virtual address from the string, converts it into its binary equivalent and displays the bits, which are partitioned into the page and offset fields.



Translation Lookaside Buffer

Virtual Page Number	Physical Page Number
-	-
-	-
-	-
-	-
-	-
-	-
-	-
-	-

TLB Hits 0 TLB Misses 0

Page Table

	Frame Number	Valid Bit
0	-	0
1	-	0
2	-	0
3	-	0
4	-	0
5	-	0
6	-	0
7	-	0

Page Hits 0 Page Faults 0

Address Reference String

A0
64
E1
EC
41
B3
9C
11
C7
E8

Virtual Address Bits

PAGE	OFFSET
101	00000

Physical Address Bits

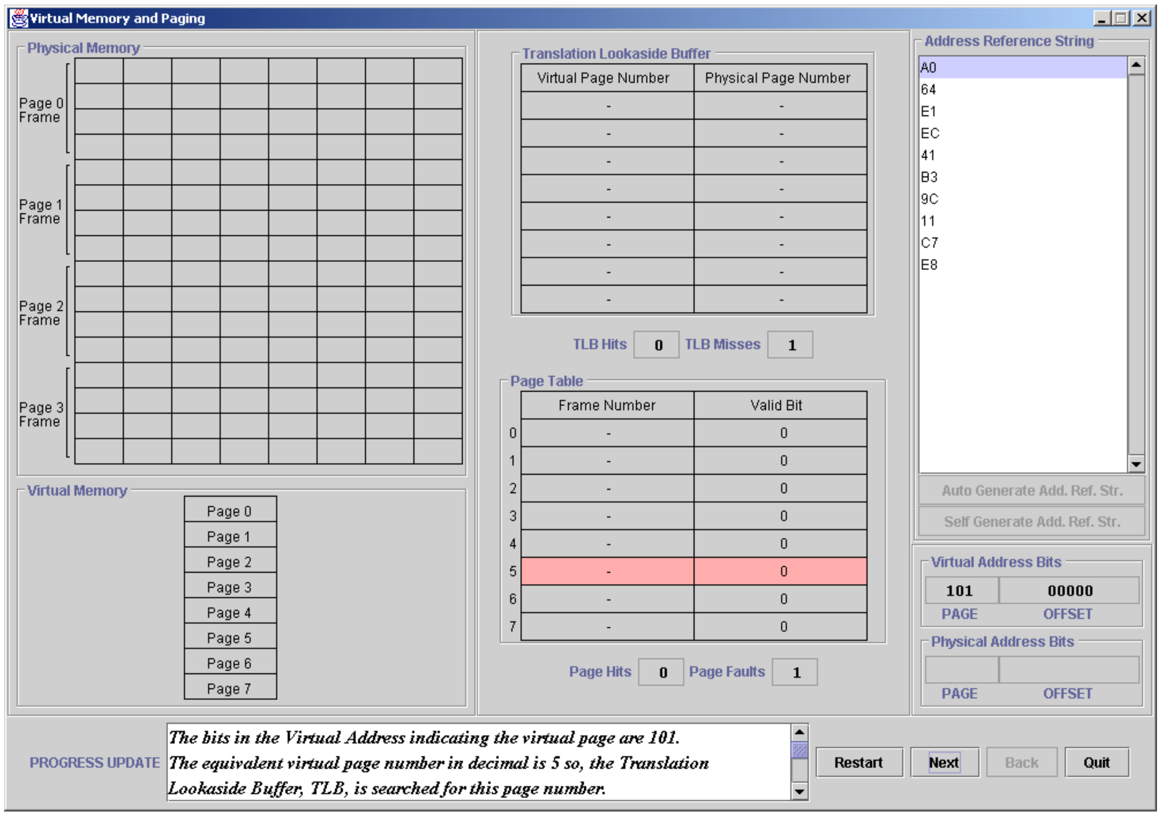
PAGE	OFFSET

PROGRESS UPDATE *The hexadecimal address A0 evaluates to its binary equivalent 10100000. Hence the bits in the Virtual Address are divided into the following fields 101 --> Page, 00000 --> Offset.*

Figure 17: Virtual Memory – Virtual Address Bits

The virtual address bits are highlighted in green. The set of bits below them are the physical address bits, which will be evaluated once the page frame in memory where the virtual page should reside is determined

Now that the virtual page number is determined, the CPU looks in the TLB for a matching entry. Since the TLB contains pairs of virtual page numbers and physical page numbers, a match on the virtual page number can provide the physical page frame where the virtual page resides, if at all.



Translation Lookaside Buffer

Virtual Page Number	Physical Page Number
-	-
-	-
-	-
-	-
-	-
-	-
-	-
-	-

TLB Hits 0 TLB Misses 1

Page Table

	Frame Number	Valid Bit
0	-	0
1	-	0
2	-	0
3	-	0
4	-	0
5	-	0
6	-	0
7	-	0

Page Hits 0 Page Faults 1

Address Reference String

A0
64
E1
EC
41
B3
9C
11
C7
E8

Virtual Address Bits

PAGE	OFFSET
101	00000

Physical Address Bits

PAGE	OFFSET

PROGRESS UPDATE *The bits in the Virtual Address indicating the virtual page are 101. The equivalent virtual page number in decimal is 5 so, the Translation Lookaside Buffer, TLB, is searched for this page number.*

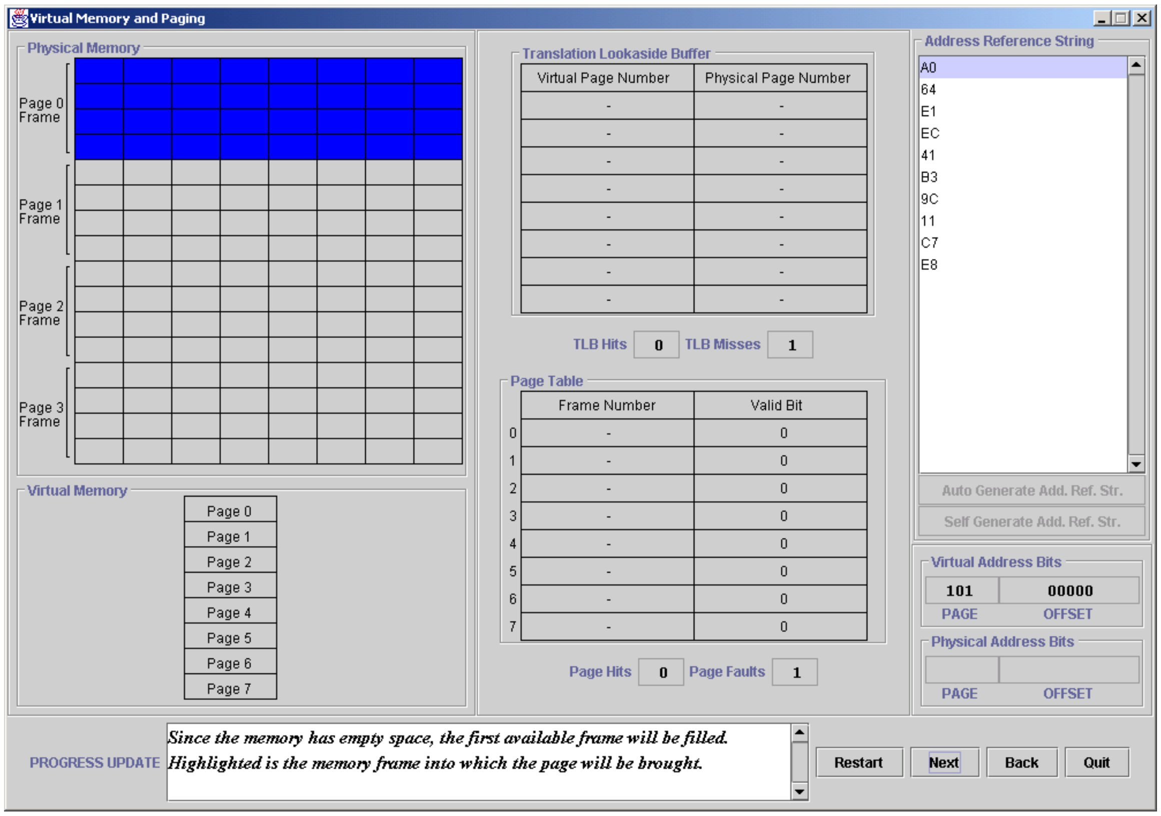
Figure 18: Virtual Memory – TLB and Page Table

In this particular instance, there is a TLB miss since there is no matching entry for the virtual page number. Thus the CPU goes to the page table to try and retrieve the information. The highlighted page table entry shows the entry, which should be looked up. In this example, there is a page fault because the entry for the virtual page has a valid bit of 0, indicating that the page must be brought in from the disk.

But before the page is brought into memory from disk, the simulator highlights, in blue, the memory frame into which the page will be brought. The memory frame is chosen by the following criteria:

(i) The first available empty memory frame is chosen.

(ii) If memory is full, then a “victim” block is chosen for replacement. As in the earlier cache mapping tutorials, the LRU algorithm was used for the purpose of this workbench.



Translation Lookaside Buffer

Virtual Page Number	Physical Page Number
-	-
-	-
-	-
-	-
-	-
-	-
-	-
-	-

TLB Hits 0 TLB Misses 1

Page Table

	Frame Number	Valid Bit
0	-	0
1	-	0
2	-	0
3	-	0
4	-	0
5	-	0
6	-	0
7	-	0

Page Hits 0 Page Faults 1

Address Reference String: A0, 64, E1, EC, 41, B3, 9C, 11, C7, E8

Virtual Address Bits

PAGE	OFFSET
101	00000

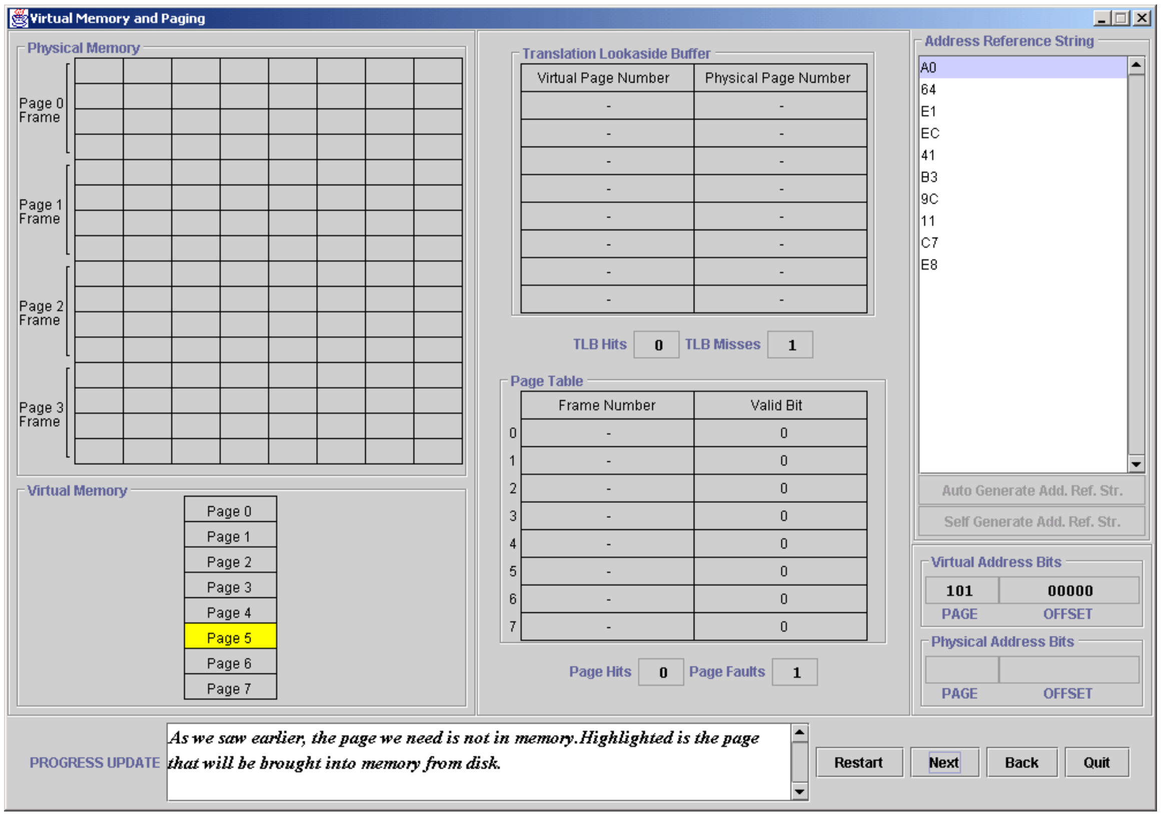
Physical Address Bits

PAGE	OFFSET

Since the memory has empty space, the first available frame will be filled.
Highlighted is the memory frame into which the page will be brought.

Figure 19: Virtual Memory – Page Frame (empty)

Now the CPU goes to the hard disk to retrieve the page in question. This virtual page is highlighted in yellow.



Translation Lookaside Buffer

Virtual Page Number	Physical Page Number
-	-
-	-
-	-
-	-
-	-
-	-
-	-
-	-

TLB Hits 0 TLB Misses 1

Page Table

	Frame Number	Valid Bit
0	-	0
1	-	0
2	-	0
3	-	0
4	-	0
5	-	0
6	-	0
7	-	0

Page Hits 0 Page Faults 1

Address Reference String

A0
64
E1
EC
41
B3
9C
11
C7
E8

Virtual Address Bits

PAGE	OFFSET
101	00000

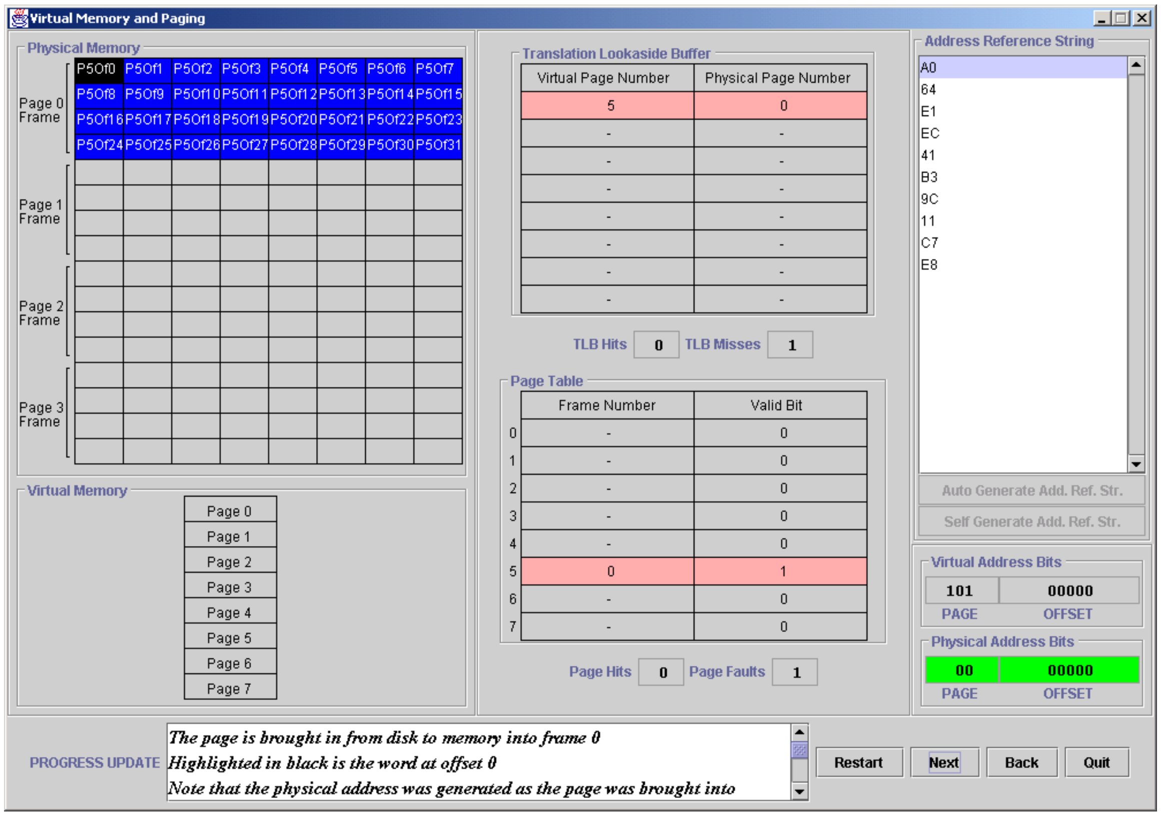
Physical Address Bits

PAGE	OFFSET

PROGRESS UPDATE *As we saw earlier, the page we need is not in memory. Highlighted is the page that will be brought into memory from disk.*

Figure 20: Virtual Memory – Virtual Page

This page is then brought into the memory page frame, and the memory frame is highlighted along with the data it now contains. Additionally, the page table and TLB are updated to reflect this change, and the physical address bits are highlighted to illustrate the translation of the virtual address into the physical address.



Virtual Memory and Paging
Physical Memory
Virtual Memory
Translation Lookaside Buffer
Page Table
Address Reference String
Auto Generate Add. Ref. Str.
Self Generate Add. Ref. Str.
Restart
Next
Back
Quit

Physical Memory

Page 0 Frame	P5Of0	P5Of1	P5Of2	P5Of3	P5Of4	P5Of5	P5Of6	P5Of7
	P5Of8	P5Of9	P5Of10	P5Of11	P5Of12	P5Of13	P5Of14	P5Of15
	P5Of16	P5Of17	P5Of18	P5Of19	P5Of20	P5Of21	P5Of22	P5Of23
	P5Of24	P5Of25	P5Of26	P5Of27	P5Of28	P5Of29	P5Of30	P5Of31
Page 1 Frame								
Page 2 Frame								
Page 3 Frame								

Virtual Memory

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

Translation Lookaside Buffer

Virtual Page Number	Physical Page Number
5	0
-	-
-	-
-	-
-	-
-	-
-	-
-	-

TLB Hits 0 TLB Misses 1

Page Table

	Frame Number	Valid Bit
0	-	0
1	-	0
2	-	0
3	-	0
4	-	0
5	0	1
6	-	0
7	-	0

Page Hits 0 Page Faults 1

Address Reference String

A0
64
E1
EC
41
B3
9C
11
C7
E8

Virtual Address Bits

101	00000
PAGE	OFFSET

Physical Address Bits

00	00000
PAGE	OFFSET

PROGRESS UPDATE

The page is brought in from disk to memory into frame 0
Highlighted in black is the word at offset 0
Note that the physical address was generated as the page was brought into

Figure 21: Virtual Memory – Page Frame (full)