Dalhousie University
Department of Electrical and Computer Engineering

ECED 3403 – Computer Architecture

Assignment 3: Cache memory

1 Objectives

This assignment requires the design, implementation, and testing of cache memory to be added to the TI 430 emulator developed in assignment 2. The objective is to demonstrate the effects of different cache organizations and two cache replacement policies.

2 Requirements

A single cache memory is to be placed between the CPU and the external program memory in order to reduce memory access times. Since cache memory is usually several times faster than primary memory, storing data and instructions in cache should reduce CPU memory-access times, thereby increasing the speed of the machine. However, in order to experience the speed-up, the contents of the target address must be in the cache (this is referred to as a **hit**); if the contents are not in the cache, the contents of the target address must be fetched and stored in the cache (this is referred to as a **miss**). Various techniques can be used to increase the **hit ratio**; however, due to costs, cache memory can be several orders of magnitude smaller than primary memory.

The cache memory and the two replacement policies are to be emulated in software. Executable programs are to be used to show the benefits and limitations of the different organizations and replacement policies.

2.1 Cache organizations

Regardless of the organization, a cache memory is made up of a series of **lines**, where each line contains at a minimum, the address of the contents of the primary memory location and the contents of the location. When a memory access takes place (that is, a read or a write) the cache is inspected, if the address in question exists in the cache, a hit has occurred.

There are different ways in organizing cache memory in order to help increase the hit ratio; two extremes are to be emulated in this assignment:

Associative. In associative, the cache circuitry inspects each cache line for the target address (this is effectively a linear search). There is no relationship between the target address and its location in the cache memory.

Direct mapping. In direct mapping, the target address is mapped directly into a cache line (this is a hash mapping). There is a direct relationship between the target address and its location in the cache memory.

The cache memory is to contain 32 lines, regardless of organization.

2.2 Replacement policies

When a read or a write operation occurs, a 16-bit target address is supplied to the cache and the cache is searched for the target address (see above). There are four possible combinations, as shown in Table 1.

Table 1: Cache actions

Operation	Result	Action
READ	HIT	Return contents to CPU.
READ	MISS	Must fetch contents from primary memory and place address and contents in a cache line as well as return contents to CPU. See discussion below.
WRITE	HIT	Update cache. See discussion below.
WRITE	MISS	Find cache line and write address and contents to this cache line. See discussion below.

Memory reads that result in a hit require no further action; however, the remaining three combinations require the application of replacement policies. The general policies are as follows:

- Whenever a memory access (read or write) results in a cache miss, the cache must be updated with the new target address and the (fetched) contents of primary memory. This is to ensure **cache consistency**. The location to be used depends upon the cache organization (described above).

  In an associative cache, since there is no relationship between a line and a memory address, any line in the cache can be chosen. However, it makes little sense to select a line that is used repeatedly as replacing it may subsequently result in a miss, requiring another fetch. A common solution is to use a **Least Recently Used** (or **LRU**) policy that replaces, as the name suggests, the least recently used line in the cache.

  This problem does not occur in direct mapping as there is only one line that can be replaced.

- When a write occurs that results in a hit, the cache must be updated in order to ensure that a subsequent read from that line returns the correct value. If the cache is updated but primary memory is not, the two memories are inconsistent. There are two cache replacement policies adopted by most caches. In the first, when a write miss occurs, the contents should be written back to primary memory (this policy is referred to as **Write Back** or **WB**). In the second, the value written to the cache is written to primary memory in parallel, meaning that both the cache and primary memory are consistent (this policy is referred to as **Write Through** or **WT**).

- It is not always necessary to write a value back to primary memory; for example, cache lines containing constants (including data that has not yet been changed) or instructions are read but never written. In these cases, if the replacement policy indicates that the line should be replaced there is no need to write it back to primary memory. A **dirty-bit** can be used to indicate that a cache line has been written to, thereby allowing the cache algorithm to decide whether or not the line needs to be returned to primary memory.

3 Design suggestions

The cache can be implemented as a function that resides between the CPU and the bus. Rather than calling the bus, the CPU should call the cache when accessing program memory:

```
cache(mar, &mbr, rdwr, bw);
```

The cache itself can be represented as an array of cache lines where each line typically consists of a primary memory address, the contents of the primary memory address (a byte), and possibly a dirty "bit":

```
#define CACHE_SIZE 32

struct cache_line
{
WORD address;        /* 0x0000…0xFFFF */
BYTE contents;       /* 0x00…0xFF */
BYTE dirty;          /* TRUE or FALSE */
};

struct cache_line cache_mem[CACHE_SIZE];
```

The mapping between the primary memory address and the cache address is either a modulus (for direct mapping) or a linear search (for associative memory). For example, in direct mapping one could write:

```
cache_address = mar % CACHE_SIZE;        /* Value from 0 to CACHE_SIZE – 1 */
```

This is essentially a simple hashing function with the **cache_address** as the key.

A byte is read from or written to one cache line while a word must be read from or written to two cache lines. The cache address will be determined by the mapping algorithm in use.

Checking whether the cache line has the required address is done with an "if":

```
if (mar == cache_mem[cache_address] . address)
        /* HIT – contents are in the cache */
else
        /* MISS – contents are not in cache */
```

Finally, if there is a miss or write-through being used, the bus (and primary memory) should be called:

```
bus(mar, &mbr, rdwr, bw);
```

4 Possible exam question

Show, with assembly program examples, how the two different caches can be defeated. Explain why this occurs. How can it be fixed? Are the same programs used to defeat the caches the same?

In anticipation of such a fiendish question appearing on the final examination, you might want to try this out for yourself…

5 Marking

The cache emulator will be marked as follows:

Design
The design description must include a brief introduction as to the purpose of the software and describe the algorithms and data structures used (do *not* describe the software).

Total points: 5.

Software
A fully commented, indented, magic-numberless, tidy piece of software that meets the requirements described above and follows the design description.

Total points: 10.

Testing
Develop a set of tests that demonstrate that each of the cache-emulators work as required. Tests should follow the format described in class.

Total points: 5.

The assignment, including answers to the questions, must be submitted on paper.

6 Important Dates

Available: 27 June 2016

Due: 19 July 2016

Late assignments will be penalized 0.5 points per day or fraction thereof.

The emulator must be demonstrated before the assignment will be marked.

7 Miscellaneous

If you are having *any* difficulty with this assignment or the course, *please* contact Dr. Hughes as soon as possible.