Assignment 4

Due: Nov 24rd, 11:59 pm
Weight: 10% of total class grade

Overview

In this assignment, students will add a cache between the CPU and data memory

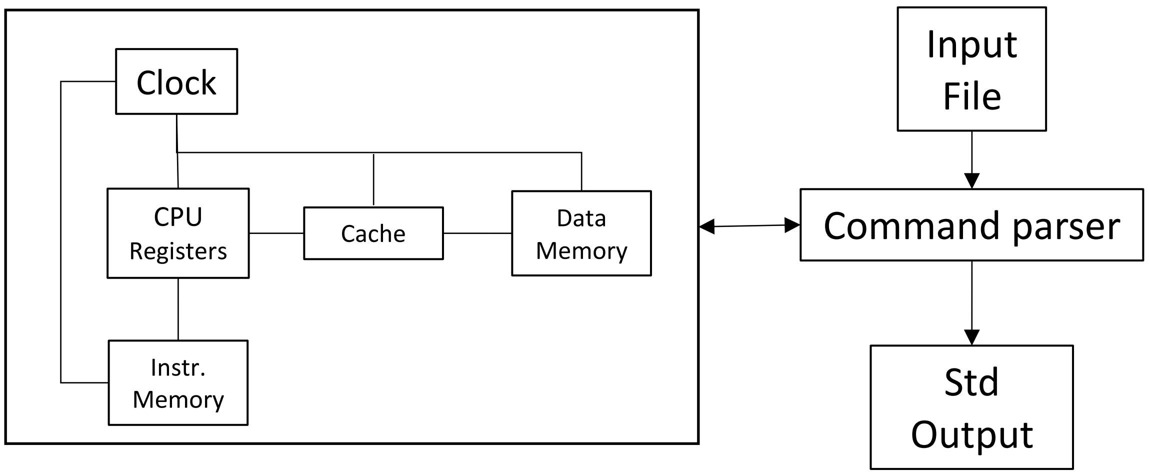
Program Execution

cs3421_emul <data_file>

Input Details

The cs3421_emul program reads an arbitrary number of lines, one line at a time from the data file. Each line will begin with a device identifier. The format of the rest of the line will be device dependent. For this assignment, possible devices are "clock", "cpu", "memory", "imemory", and "cache".

Devices



Clock

The clock device is unchanged from Assignment 2.

Memory

The memory device is unchanged from Assignment 2, except for its interface to the cache device described below.

Instruction Memory

The instruction memory device is unchanged from Assignment 2.

CPU

The CPU device is the same as in assignment 2, except the student will add a 16 bit tick counter register "TC" that will record the number of ticks the CPU has acted upon, and which stops incrementing after the CPU executes a halt instruction (count tick needed to halt CPU). On CPU reset, the TC register is set to zero.

CPU commands:

The CPU commands remain unchanged from Assignment 2. The output of the "dump" command should be expanded to show the new TC tick count register as a decimal value as follows:

```
PC: 0xAA
RA: 0x23
RB: 0x14
RC: 0xFF
RD: 0x44
RE: 0xAA
RF: 0x00
RG: 0x09
RH: 0x18
TC: 148
```

Cache

The cache device sits between the CPU and Data Memory, and can improve memory performance. The cache contains 8 bytes of high speed memory, and contains logic to handle data passing through it from read/lw instructions and write/sw instructions.

Cache Reads

The cache mirrors the contents of 8 contiguous bytes of Data Memory. The cache maintains a 5 bit “cache line offset” (CLO) register which identifies which block of 8 bytes of Data Memory is currently stored in the cache. A value of 0 identifies bytes 0-7, 1 identifies bytes 8-15, 2 identifies bytes 16-23, etc, with 31 representing bytes 248-255. The cache also keeps a Boolean if there is valid data in the cache.

When the CPU reads from Data Memory via the “lw” instruction, the cache will compute to which cache line the data to be retrieved is associated. For instance a “lw” getting data from address 0x12 would be associated with cache line 2. If the computed cache line matches the value stored in the CLO register, and there is valid data in the cache, that is a “cache hit”. On a cache hit, the cache will return the requested byte to the CPU within 1 clock tick (meaning the “lw” instruction will complete in a single tick).

If there is no valid data in the cache, or the requested byte is not within the stored cache line (computed cache line != CLO), that is a “cache miss”. On a cache miss, the cache will request 8 bytes from Data Memory to fill the 8 bytes in the cache associated with the computed cache line. In the example of a “lw” from 0x12 (decimal 18), a cache miss would request bytes 0x10-0x18 (decimal 16-23) from Data Memory. Due to hypothetical multi-port parallelization, the Data Memory will be able to satisfy this request in the regular 5 clock cycles. Due to hypothetical pipelining, the cache will be able to satisfy the CPU request in the regular 5 clock cycles. After getting the 8 bytes from Data Memory on a cache miss, the cache will update the CLO with the computed cache line, and mark the data in the cache as valid.

The CPU can force the cache to mark the data as invalid within the cache. On a “lw” to address 0xFF, the cache will always return a value of 0 within a single clock cycle, and mark all data in the cache as invalid.

Cache Writes

On a write/sw to Data Memory, the cache will compute the cache line associated with the destination. If the computed cache line matches the CLO (cache hit), or the cache has no valid data, the data is stored in the appropriate offset within the cache within a single clock tick (meaning the “sw” instruction will complete in a single tick). A “data written” Boolean should be associated with each byte within the cache. When updating a byte in the cache, the associated Boolean for that byte should be set to true.

On a cache miss (computed cache line != CLO), the cache will check if any of the “data written” Booleans are true. If so, the cache will write these bytes to Data Memory, known as a “cache flush”. After writing to Data Memory, all “data written” flags will be cleared, the CLO will be set to the computed cache line, the byte being written will be stored in the appropriate offset within the cache, and the "data written" flag for that byte will be set to true. Again due to hypothetical optimizations in Data Memory, this write operation will be able to complete in the normal 5 clock ticks needed for regular Data Memory writes.

The CPU can force the cache to flush any modified contents to memory. On a "sw" to address 0xFF, the cache will write any modified data in the cache to Data Memory, taking the regular 5 clock ticks. If there are no modified bytes, no write to Data Memory will occur, and the instruction will complete in a single clock tick. Regardless, no data is stored at memory address 0xFF.

Cache Read & Write

After doing a read from memory, the cache will have a mirrored copy of the data. If there is a write to any of the bytes in cache, it will simply update the value stored in cache, set the "data written" flag for that byte. Any reads & writes from the cache line will finish in a single cycle. On a cache miss (read or write), the modified bytes will be written back to Data Memory, taking the usual 5 clock ticks. Then normal read cache or write cache behavior will occur, depending on the instruction type.

Cache Write & Read

After doing a write & cache update, any subsequent read to that same address will return the written value stored in cache in a single clock tick (same as Cache Read). A read of a byte NOT WRITTEN will be a cache miss, forcing a normal read of Data Memory (5 ticks). For bytes that have been written in cache, the cache will discard the corresponding value returned from Data Memory. For bytes that have not been written in cache (invalid data), the corresponding values from Data Memory will be stored in cache, setting the "valid data" flag.

Cache Commands

reset

The "reset" command resets the cache to disabled, setting CLO to zero, and data to be invalid.

on

The "on" command enables the cache, causing it keep copies of data transferred between the CPU and Data Memory. The coder may assume cache will only be enabled when the CPU is idle (about to start a new instruction).

off

The "off" command disables the cache, causing the system to act as if no cache were present. Any data written to the cache should be written to memory (cache flush). The coder may assume cache will only be disabled when the CPU is idle (about to start a new instruction).

dump

The dump command will show the contents of cache, and the states of those contents. The first line shows the 5 bit CLO as two hex digits. The second line shows the contents of the 8 cache bytes in hex. The third line shows the flags indicating the state of each byte, with "I" meaning invalid, "V" meaning valid, and "W" meaning it has been written. Two examples follow:

```
CLO        : 0x04
cache data : 0x00 0x18 0x22 0xFF 0xEE 0x27 0x1E 0xAE
Flags      :   I    W    I    I    I    W    I    I

CLO        : 0x01
cache data : 0x00 0x18 0x22 0xFF 0xEE 0x27 0x1E 0xAE
Flags      :   V    V    W    W    V    V    V    V
```

Design philosophy

While it is believed that assignment describes the complete behavior of the cache, it is possible some edge cases have not have been covered in the assignment. The student should adopt the Principle of Least Astonishment, meaning the system should perform as a user of the system would expect. In particular, the system should produce the same output for every program, whether the cache is on or off. The only difference would be the number of clock ticks necessary. The student should also adhere to the laws of physics. For instance, if the latency to access Data Memory is 5 ticks, retrieving Data Memory through the cache (cache miss) can't complete any sooner than 5 ticks.

Language

The developer may use either Java or C/C++.

Submission Details

Submission will be via Canvas. Create a directory matching your username, and place all files you plan to submit with that directory. To submit your assignment, create a zip file of that directory (<your_username>.zip), and submit the zip file via Canvas. You should verify that the zip file has the correct directory structure, meaning when extracted without any special options, it WILL create a directory corresponding to the username of the student.

The zip file for Assignment 1 (<your_username>.zip) should contain the following files:

<your_username> - directory corresponding to your username which holds all files
- Author.txt – a text file with a single line containing your first & last name
- Language.txt – a single line containing the word "C" or "JAVA" (all upper case, no quotes) indicating the language used for the assignment
- Readme.txt – Any comments you'd like to share regarding the submission
- Makefile – an optional file that will compile your code into the cs3421_emul executable
- cs3421_emul source & header files necessary for your project

Note, if you have known issues with your program, DOCUMENT THAT in the Readme.txt file. Any use of non-standard libraries or packages should also be documented, as well as how to retrieve & install them.

Grading

Programs will be primarily graded on the correctness of the output, **and ability to follow submission guidelines**. Given the large number of submissions, the evaluation of the output will be automated. This means it is **<u>CRITICAL</u>** to follow the sample output above. Sample data & output will also be provided to test your programs. The assignment grade may also be based on style, comments, etc. Programs that crash, fail to produce the correct output, or don't compile/run may lose up to 100% of the points, depending upon severity of the error. Some effort may be made during grading to correct errors, but students should not depend upon this. For grading, programs will be compiled and run on Ubuntu Linux 18.04 LTS.