In this program, you will write the TLB class. You will be given some general descriptors of what must be in the class, along with a statement of functionality. A set of tests will be prescribed so that you can verify that you’ve constructed the class correctly.

An entry in the TLB (whether micro- or L2) consists of a valid boolean, a tag, an unsigned 32-bit data field, and an integer which contains LRU (Least Recently Used) data (essentially a clock or cycle value used to show how long ago that entry was last accessed).

The TLB constructor takes three parameters, the number of entries in the TLB, the associativity, and the number of bits in the TLB tag.

The constructor would be declared as:

public TLB(int Entries, int Associativity, int TLBTagBits)

The class will have three other methods. One to get data from the TLB, one to set data into the TLB, and a third method to print the contents of the TLB. These methods are further specified as follows:

uint GetData(uint uAddress, uint uCycle)

This method will return the 16-bit physical page number from the TLB. In the case of a miss, the return value is 0xffffffff which would not be a 16-bit page number.

The TLB must take the requested address and decompose it into the set index and tag values. A tag match will be used to determine if there is a hit. In the case of a hit, the data value in the TLB entry is returned.

The cycle number is used to update the LRU field of the TLB entry if it exists. If an entry eviction is later required, the LRU values are used to determine which entry is evicted.

void SetData(uint uData, uint uAddress, uint uCycle)

This method enters data into the TLB. The address is decomposed to find out what set in which the tag will be searched for.

* If the tag is found, the data is written into the TLB entry data field and the LRU number is updated to the current cycle.
* If the tag is not found and an empty entry (valid == false) exists in the set, then the data is written to that entry.
* If no tag is found and no invalid entry exists in the set, the lowest LRU number if not equal to zero is overwritten with the new data, tag, and LRU cycle. An LRU entry of 0 with the valid bit set indicates that the TLB entry is locked, that is to say that it cannot be evicted.

void Print(String strFileName)

Print outputs a text file indicating the TLB contents.

The name of the output files is in the calling parameter.

If the TLB is fully associative, each entry is listed on a separate line. The data fields are entry number, valid bit, the tag, the LRU number and finally the data. All numbers are integer format except data which is output in hex. Fields are tab delimited.

If the TLB is not fully associative, then data from a single set appears on a line. The line does not appear if all entries in the set are invalid. If at least one entry is valid, then the line appears as follows. The set number is output as an integer. This is followed by n blocks od data where n is the associativity. So for a 4-way set associative TLB, the set number would be followed by information from the 4 blocks composing the set. The format of each block is the block number followed by valid status, the tag, the LRU number and the data. Again, only the data is shown in hex; the rest of the numbers are integers.

Test code for the fully associative TLB:

uint uCycle = 1;

//Micro TLB is 10 entries, fully associative, 16-bit data value

TLB MicroTLB = new TLB(10, 0, 16);

MicroTLB.SetData(0x10, 0xABCDEF12, uCycle++);

MicroTLB.SetData(0x20, 0xBBCDEF12, uCycle++);

MicroTLB.SetData(0x30, 0xCBCDEF12, uCycle++);

MicroTLB.SetData(0x40, 0xDBCDEF12, uCycle++);

MicroTLB.SetData(0x50, 0xEBCDEF12, uCycle++);

MicroTLB.SetData(0x60, 0xFBCDEF12, uCycle++);

MicroTLB.Print("MicroTLB1.txt");

//the following line fires up Notepad to save the time of doing it manually

System.Diagnostics.Process.Start("notepad.exe", "MicroTLB1.txt");

//for C++, use WinExec("Notepad MicroTLB1.txt", SW\_NORMAL);

//for Java, use ProcessBuilder pb = new ProcessBuilder("notepad.exe", "MicroTLB1.txt");

// pb.start();

MicroTLB.SetData(0x70, 0x0BCDEF12, uCycle++);

MicroTLB.SetData(0x80, 0x1BCDEF12, uCycle++);

MicroTLB.SetData(0x90, 0x2BCDEF12, uCycle++);

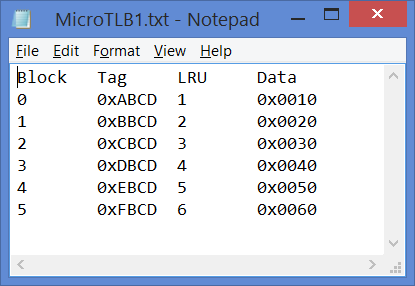
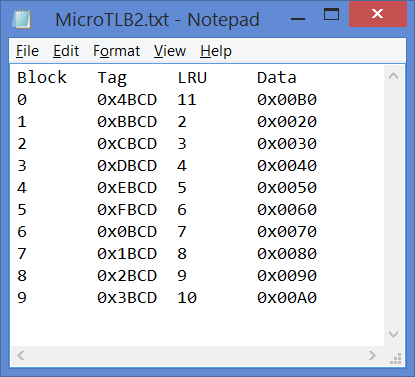
MicroTLB.SetData(0xA0, 0x3BCDEF12, uCycle++);

MicroTLB.SetData(0xB0, 0x4BCDEF12, uCycle++);

MicroTLB.Print("MicroTLB2.txt");

System.Diagnostics.Process.Start("notepad.exe", "MicroTLB2.txt");

The test output should look like

Test code for the n-way associative TLB:

//L2 TLB is 512 entries, 4-way, 16-bit data

TLB L2TLB = new TLB(512, 4, 16);

L2TLB.SetData(0x10, 0xABCDEF12, uCycle++);

L2TLB.SetData(0x20, 0xBBCDEF12, uCycle++);

L2TLB.SetData(0x30, 0xCBCDEF12, uCycle++);

L2TLB.SetData(0x40, 0xDBCDEF12, uCycle++);

L2TLB.SetData(0x50, 0xEBCDEF12, uCycle++);

L2TLB.SetData(0x60, 0xFBCDEF12, uCycle++);

L2TLB.SetData(0x70, 0x0BCDEF12, uCycle++);

L2TLB.SetData(0x80, 0x1BCDEF12, uCycle++);

L2TLB.SetData(0x90, 0x2BCDEF12, uCycle++);

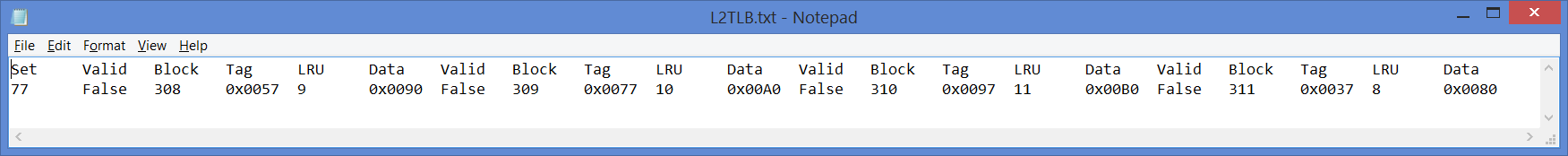
L2TLB.SetData(0xA0, 0x3BCDEF12, uCycle++);

L2TLB.SetData(0xB0, 0x4BCDEF12, uCycle++);

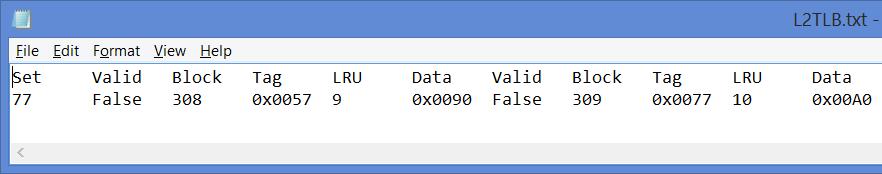
L2TLB.Print("L2TLB.txt");

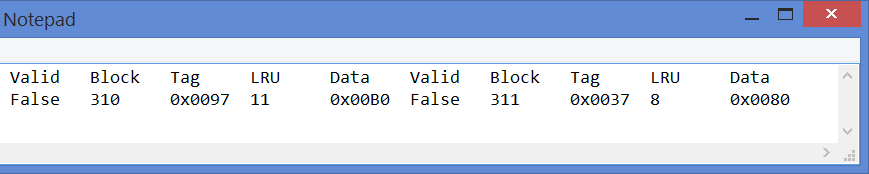
System.Diagnostics.Process.Start("notepad.exe", "L2TLB.txt");

The test output should be a relatively wide display which looks like:



Or enlarged:





These tests are not complete, but they cover 90% of the TLB functionality. The GetData call is not tested in these cases. You should be able to write a GetData call and ensure that you are obtaining the correct data for the input address, and that the LRU number for the TLB entry is being updated.

Turnin:

Please post your code files and your output files to Canvas. Acceptable files are .java, .cs, .cpp, .h, .hpp and .txt.