CSE 325 **Spring 2021**

Computer Project #7

Assignment Overview

This assignment focuses on the interactions between registers, the data cache, and primary storage, and is the first milestone in a larger simulation. You will design and implement the C/C++ program which is described below.

It is worth 35 points (3.5% of course grade) and must be completed no later than 11:59 PM on Thursday, 3/18.

Assignment Deliverables

The deliverables for this assignment are the following files:

`proj07.makefile` – the makefile which produces **`proj07`**
`proj07.student.cpp` – the source code file for your solution

Be sure to use the specified file names and submit them for grading via the CSE Handin system before the project deadline.

Assignment Specifications

The system to be simulated is a 16-bit microprocessor: it has sixteen 16-bit general purpose registers and 16-bit physical addresses (and thus has 65,536 bytes of RAM). The data cache is a direct-mapped, write-back cache which contains eight lines. Each cache line contains control bits (valid bit, modified bit, and tag bits) and an 8-byte data block.

1. The program will simulate the interactions between the register unit, the data cache, and primary storage by processing a file which contains zero or more instructions. Each line of the file will contain the following information:

a) operation ("LDR" for "load into register" or "STR" for "store from register")
b) register number (one hexadecimal digit)
c) physical address (four hexadecimal digits, with leading zeroes)

Items in the line will be separated by exactly one space, and each line will terminate with a newline. For example:

```
LDR 5 ebd8
STR 4 0ac2
LDR a 0ad0
```

The first line represents an instruction where the two bytes at address ebd8 of RAM are loaded into register 5, and the second line represents an instruction where the two bytes in register 4 are stored into address 0ac2 of RAM.

2. For each memory reference in the file, the program will display one line with the following information:

a) operation ("LDR" or "STR")
b) register number (one hexadecimal digit)
c) physical address (four hexadecimal digits, with leading zeroes)
d) tag bits (three hexadecimal digits, with leading zeroes)
e) cache line accessed (one hexadecimal digit)
f) byte offset (one hexadecimal digit)

Items in the line will be separated by exactly one space, and the line will terminate with a newline. For example:

```
LDR 5 ebd8 3af 3 0
STR 4 0ac2 02b 0 2
LDR a 0ad0 02b 2 0
```

3. After the simulation is completed, the program will display the contents of the registers, the data cache, and RAM.

a) The contents of all sixteen registers will be displayed, where each 4-bit register number is displayed in hexadecimal and the 16-bit contents are displayed in hexadecimal (four hexadecimal digits, with leading zeroes). For example:

```
R0: 0000   R4: 0000   R8: 0000   Rc: 0000
R1: 0000   R5: 0000   R9: 0000   Rd: 0000
R2: 0000   R6: 0000   Ra: 0000   Re: 0000
R3: 0000   R7: 0000   Rb: 0000   Rf: 0000
```

b) The contents of the data cache will be displayed, with one line for each data cache entry:

a) index of the data cache entry (one hexadecimal digit)
b) V bit (one character; '0' for not valid, '1' for valid)
c) M bit (one character; '0' for not modified, '1' for modified)
d) tag bits (three hexadecimal digits, with leading zeroes)
e) data block (eight bytes of data, where each byte is given as two hexadecimal digits, with leading zeroes)

The data cache display will include appropriate column headers. For example:

```
     V M Tag  0  1  2  3  4  5  6  7
-----------------------------------
[0]: 0 0 000 00 00 00 00 00 00 00 00
[1]: 0 0 000 00 00 00 00 00 00 00 00
[2]: 0 0 000 00 00 00 00 00 00 00 00
[3]: 0 0 000 00 00 00 00 00 00 00 00
[4]: 0 0 000 00 00 00 00 00 00 00 00
[5]: 0 0 000 00 00 00 00 00 00 00 00
[6]: 0 0 000 00 00 00 00 00 00 00 00
[7]: 0 0 000 00 00 00 00 00 00 00 00
```

c) The contents of the first 128 bytes of RAM will be displayed. Each line of that display will include an address (four hexadecimal digits, with leading zeroes) and the sixteen bytes of data starting at that address (two hexadecimal digits, with leading zeroes). For example:

```
0000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

4. Command line arguments will be used to specify the name of the file which contains the instructions to be processed (the "-input" option), the name of the file which contains the initial contents of RAM (the "-ram" option), and to display debugging information (the "-debug" option).

a) The "-input" option will be followed by the name of the file which contains zero or more instructions (as defined above).

b) If selected, the "-ram" option will be followed by the name of the file which contains the initial contents of a subset of RAM. Each line of that file will contain an address which is a multiple of 16 (four hexadecimal digits, with leading zeroes), followed by sixteen bytes of data (two hexadecimal digits, with leading zeroes). For example:

```
0040 10 45 1b 00 ff 00 00 00 00 00 00 00 00 00 00 00
0070 00 00 a7 00 00 9c 00 00 00 00 00 00 00 00 00 00
ffc0 00 11 22 33 00 00 00 00 00 00 00 00 cc dd ee ff
```

5. If selected, the "-debug" option will cause the program to display debugging information.

a) After all program initialization has occurred and immediately before the first line of the "-input" file is processed, the program will display the contents of the registers, the data cache, and RAM (using the same format as #3 above).

b) The contents of the data cache after each instruction is processed.

6. The program will include appropriate logic to handle exceptional cases and errors.

Assignment Notes

1. As stated above, your source code file will be named "proj07.student.cpp" and you must use "g++" to translate your source code file in the CSE Linux environment.
.
2. Valid executions of the program might appear as follows:

```
proj07 –input testA
proj07 –debug –input testB
proj07 –ram initialRAM –input testC
proj07 –ram initialRAM –input testD -debug
```

3. Your program may assume that the "-input" and the "-ram" files are formatted correctly:

a) If the "-input" file can be accessed, the contents will be valid as per #1 under "Assignment Specifications".

b) If the "-ram" file can be accessed, the contents will be valid as per #4 under "Assignment Specifications".

4. Your program must create the following data structures:

a) A data structure representing the registers. All sixteen registers will be initialized to zero at the start of the simulation.

For this assignment, processing the instructions will not update the registers. In subsequent assignments, your program will actively manage the registers (and thus the contents will change over time).

b) A data structure representing the data cache. All of the entries will be initialized to zero at the start of the simulation.

For this assignment, processing the instructions will not update the data cache. In subsequent assignments, your program will actively manage the data cache (and thus the contents will change over time).

c) A data structure representing the RAM. All of the entries will be initialized to zero at the start of the simulation. If the user selects the "-ram" option, a subset of the RAM will be re-initialized using the contents of the specified file.

For this assignment, processing the instructions will not update the RAM. In subsequent assignments, your program will actively manage the RAM (and thus the contents will change over time).

5. An example of a program which illustrates one approach to modeling the registers is posted:

/user/cse325/Projects/project07.registers.cpp

You are not required to use that data structure, but you might wish to review that program for ideas about modeling the registers, the data cache, and the RAM.