CSE 560 **Winter 2010, 10:30 a.m. Section**

LAB #2: Simulator Lab

DUE: 10:30 a.m., Friday, January 29 (Preliminary documentation due 10:30 a.m.., Tuesday, January 19)

Using the machine definition and instruction set descriptions given in class, design, code, test, and debug a program to simulate the execution of the W10-560 machine. An *executable* input file (i.e., a *correct* one) is a sequence of lines as described in the Warm-Up Assignment, Lab #1. Lines are separated by next-line markers. Each byte b in a line is such that $b = 32$, $48 \leq b \leq 57$, $65 \leq b \leq 90$, or $97 \leq b \leq 122$. Optionally, the last line of the file may be empty. All other lines are not empty and are described below as records.

In this description, a "k Hex character" address or value is a sequence of k bytes where each byte b is such that $48 \leq b \leq 57$ or $65 \leq b \leq 70$. (At your team's option, it may also permit $97 \leq b \leq 102$.) This sequence of bytes represents a nonnegative number according to the usual simple base 16 (hexadecimal) positional notation system when the bytes are interpreted as characters according to the ASCII encoding scheme.

Also, in this description, a character stands for the byte that would encode it according to ASCII; for example, 'H' stands for the byte 72. The term "record position" means the byte number in a line, counting from one. Hence "record position 1" is the first byte in a line. An executable input file contains two kinds of records as described on the following page:

1. A Header Record
   record position 1: H
   record positions 2-3: a 2 Hex character address at which execution is to begin
   record positions 4-9: a 6 character segment name
   record positions 10-11: a 2 Hex character value denoting the segment load address
   record positions 12-13: a 2 Hex character value denoting the length of the segment

2. Text Records
   record position 1: T
   record positions 2-3: a 2 Hex character address at which the information is to be stored
   record positions 4-8: Initial value at that address, as a 5 Hex character value

Below is a sample executable input file:

```
H66SAMPLE6412
T6500002
T66B0064
T67B1000
T68B2064
T6990008
T6AB1000
T6B90004
T6CB3000
T6D90008
T6EB3000
T6F01074
T7000474
T7131065
T7290402
T73C0C00
T7400001
T7500005
```

Like all (correct) executable input files, it starts with a single header record, followed by a sequence of text records.

Do not assume that initial contents will be given to you for each of the 256 memory addresses.

Your program should have two major components: one to "load" the executable input information into the data structures that represent the memory and registers of the 560 machine, and the second to simulate the interpretation of 560-machine instructions. This second component will read from a *process* input file and write to two files, a *process* output file and a process *trace* file. The "IO" instruction reads from the process input file and writes to the process output file. The "BR" instruction also writes, as directed, to the process output file.

Your instruction interpreter must be capable of executing each of the machine's instructions as specified. The instruction interpreter must detect and provide, in the process output file, appropriate messages under the following error conditions:

- address range error
- attempt to divide by zero
- shift exception (attempt to shift more than 19 bits)
- time limit exceeded (when execution of more than 200 instructions has been attempted)
- plus others that you may identify

Note: only the "time limit exceeded" condition or a fatal I/O error will stop execution. The other conditions result in no-ops, and the program being interpreted continues at the next sequential location.

The loader must detect and provide, in the process output file (optionally also to stderr), appropriate messages under the following error conditions:

- invalid contents (i.e., illegal bytes in an executable input record)
- other errors you identify

The process *trace* file should contain a trace of the execution of each instruction, including the contents of the registers and memory locations that are relevant to the instruction. Also include, in the process trace file, a complete display of all memory and all registers of the 560 machine immediately after the initial configuration is loaded and again upon termination (either normally or via a time-out error detected by the interpreter) of a test program.

To make life a lot easier on some of your users (in particular, your graders), your program must allow a command-line interface. Furthermore, it must allow the four file path names to be specified as arguments on the command line. On a command line, a user can use the "file name completion" feature of the shell (usually the Tab key invokes this feature). This helps the user know whether the file name is spelled correctly (informing the user whether the file exists), and the user can then often accomplish the task with much less typing. Of course, your program should provide an informative error message if it is invoked with the wrong number of command line arguments. Such a message often includes or consists of instructions for proper use of the program, sometimes preceded by the terse noun phrase "Usage:". Your program must interpret the following command-line format, in the order specified, as being correct:

```
program-name executable-input process-input process-output process-trace
```

You are to submit an on-line writeup that contains:

- a table of contents, in HTML

- a user's guide which, in addition to a brief description of what the program is all about, tells how the program is envisioned to be run, including any Unix commands and files needed to access the program, the input and output requirements and conventions, and error conditions and their associated error messages
- a programmer's guide describing the overall design of your solution
- a description of the testing you did attempting to reveal defects in your program, including the test data inputs, their associated expected outputs, and some rationale for your choice of tests
- a summary of each of your group's meetings, including decisions made by the group and responsibilities assigned to individual group members

You are also to turn in a peer review "grade" for each of the other members in your group. A template for this peer review is available on the class web site. This review should be sent by email to the grader(s) by 5pm on the day the lab is due.

In your peer evaluation, rate each person in your group (including yourself) on a scale of 0 to 10. If everyone pitched in reasonably, then everyone should get an 8 (reserve a 10 for truly exceptional work). **The total of the individual ratings must not exceed $8n$, where n is the number of people in your group**. Include comments to justify and clarify the ratings you have assigned.

You should produce "professional-looking" on-line documentation: documentation that you would be proud to show your boss. At least the top level, the table of contents, should in HTML, viewable by a web browser. Use a good word processor (such as Netscape Composer, Microsoft FrontPage, HTML in emacs, Framemaker, LaTeX or Microsoft Word) to facilitate document preparation. You may use whatever other tools you wish to facilitate diagramming and communication among the team.

Your documents should be well written, logically structured, and pleasingly formatted. Hint: leave time for other members of your group to proof read what you write, and vice versa since your grade hinges partially on the rest of your group. This applies to both native and non-native English speakers. Give each other advice on their writing style. Leave time to "smooth over" the writing: the documentation should flow as if all of it was written by the same person, even though not all of the documentation is tailored to the same audience.

You are to work in teams of 4 if at all possible (otherwise 5 if absolutely necessary), with partners of your own choosing.

Divide up the workload conveniently to try to avoid overloading any group member. **Schedule meetings** to discuss problems and progress, and to ensure that each team member does his/her own fair share. It is important that all members of the group have access to the up-to-date status of the design of your program. Keep records ("minutes") of the proceedings these meetings, since they will be turned in as part of your lab writeup. Use these meetings, email, and your group accounts to facilitate this. Talk about aspects of your solutions among one another, so that each of you understands them. Having someone else look at your work may help uncover flaws in your logic.

Do not make any changes to the lab requirements without authorization from me! Also

avoid the temptation to make changes to the decisions of the group without consulting the group for agreement.

Let me know as soon as possible if the team seems to be having trouble (e.g., if one member drops the course or isn't coming to meetings, or if the group isn't scheduling meetings frequently enough).

Each group is required to schedule an approximately 20-minute meeting with me and the grading assistant to discuss your design and to answer questions. All members of the group are expected to be present for the entire design review. The schedule for these reviews will be set during the second week of the quarter. However, please don't hesitate to ask questions earlier than that in class or during office hours, especially if there is a point that appears vague or unintelligible to you. Please note that **a draft of your programmer's guide (the outputs of the tasks in the "560 Software Design" handout) is due *prior* to your scheduled design review meeting**. Other documents (e.g., user's guide and test plan) are not required for the design review, *but you are urged to turn in a draft of them so that we may give you valuable feedback on everything.*

Breakdown for the lab 2 grade

Group part:

User's guide: 20%
Programmer's guide: 20%
Coding: 10%
Your testing: 10%
Our testing: 20%
Meeting minutes: 10%

Individual part:

Peer review: 10%

The documents will be graded based on both their technical quality and their English.

[This page was left blank intentionally.]