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

LAB #4: Linker/Loader Lab

DUE: 10:30 a.m. Friday, March 12

Part 1 (Modifying Assembler)

Modify the assembler you built in Lab 3 so that it is capable of dealing with external symbols.

Specifically, your assembler should be capable of processing the following two pseudo-ops, in addition to the ones described in Lab 3.

Mnemonic	Meaning
ENT	[**ENTry name**] The operand field of this pseudo-op is a list of symbols that must be defined in the current segment, and are permitted to be referenced by other segments.
EXT	[**EXTernal name**] The operand field is a list of symbols that may be referenced legitimately by this segment, but are not defined in this segment. The symbols must be defined in some other segment.

For both the ENT and EXT pseudo ops, there must be no label, and there must be at least one, but no more than four operands. You may at your option require that these pseudo-ops, if present, precede other pseudo-ops and executable instructions (except, of course, ORI). Appropriate error messages should be written if there are violations of the syntax rules for the ENT and EXT pseudo-ops.

The object file produced by your modified assembler is up to you, but it must contain all the essential information (e.g., text, relocation information, length, start address for execution) as well as appropriate information about external symbols, so the loader can perform any linking needed. Describe the format of this file thoroughly in your documentation.

Your modified programmer's guide, user's guide, and test plan should highlight the parts that have changed as a result of the modification.

Part 2 (Linking Loader)

Write a (relocating) linking loader for our abstract machine. Your system must be capable of handling multiple input files and linking them together in order to form the executable file that is input to the simulator.

The format of each input file is the same as the object file produced by your modified assembler. The primary output of the loader should be an absolute executable file, ready to be passed to the

560 machine simulator, which will interpret the instructions. *The format of the primary output file should be exactly that described in lab 2 as the input to the simulator.*

Note that your loader will need an initial program load address, which normally is provided by the operating system. Because we do not have a "560-machine operating system", your loader will have to obtain an initial program load address from a human user. Your loader should report to the user on the standard output device (stdout) the size of the segment to be loaded, prompting the user to enter on the standard input device (stdin) a load address.

Have your loader be capable of detecting "invalid relocation information" errors, "undefined external symbol" errors, errors in any link records used, as well as the other errors associated with the loader specified in the lab 2 handout. These errors should be reported in an error log output file.

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 all file path names to be specified as arguments on the command line. 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:". The best place to send this message is to the standard error device (stderr). Your program must interpret the following command-line format, in the order specified, as being correct (where "obj-1 ... obj-n" stands for one or more input object files, exactly one of which has a segment name of "Main ", from which the linking loader will obtain the initial value of the program counter.

```
prog-name output-executable output-error-log obj-1 . . . obj-n
```

Part 3 (Integration)

Integrate the assembler (produced in part 1 of Lab 4), the loader (produced in part 2 of Lab 4), and the 560-machine simulator that you wrote in lab 2, to produce a "system" capable of processing (correct) source programs written in the 560-machine's assembly language from translation through linking, loading, and execution. Integration is primarily a documentation and testing task, not a programming task. That is to say, it doesn't make too much sense to automate the process of assembling several files, linking them into an executable file, and sending this executable to the simulator. It is more sensible for the user to control these activities manually. Provide a guide showing the way to the user, and provide a whole-system test plan for programmers whose job it is to modify your system.

The documentation you turn in for lab 4 should include a brief appendix that gives instructions for using the three components in tandem. Include here any appropriate references to the user's guides for the emulator, assembler and loader for more detailed information about these components. You should also produce an integration test plan for the assembler, loader, and simulator working in tandem, and the results of this testing. It may be somewhat more brief than the test plans for the individual labs because it is easier to demonstrate that the three components work together given that they work separately.

Approximate breakdown of Lab 4 grade

Modified Assembler Documentation, Testing, Code: 15%
Loader User's guide: 10%
Loader Programmer's Guide: 10%
Loader Test Plan: 5%
Integration User's Guide and Testing and Meeting Minutes: 20%
Grader's evaluation of "smoothness of integration": 5%
Grader's Testing and evaluation: 25%
Individual grade: 10%