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

LAB #3: Assembler Lab

DUE: 10:30 a.m., Friday, February 26

Write an assembler for the 560 machine's assembly language. An *assembly language source* 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 $32 \leq b \leq 126$. Optionally, the last line of the file may be empty. All other lines are not empty and are described below as records. Input records for this language will have the following format:

Record Position	Meaning
1-6	Label, if any, left justified
7-9	Unused
10-12	Operation field
13	Unused
14-end of record	Operands and comments
Exception:	A semicolon (;) in the first record position indicates that the entire record is a comment.

Notes

1. In the "operands and comments" field, the "operand" part is considered terminated when a blank is encountered, except for the CCD pseudo op.

2. Symbols and labels are up to 6 alphanumeric characters, with the first character alphabetic. Alphabetic characters can be upper or lower case, and an upper case character should be treated differently from its corresponding lower case character.

3. In the operand field of a machine instruction: The "R" part can be a symbol or a decimal integer. Its value must be between 0 and 3, inclusive (for symbols, note that this also means the value of the symbol can't be relocatable).

   The "S" field can be a symbol, a decimal integer in the range [0, 255], or a "literal." "Literal"s begin with an "=" and are followed by a decimal integer in the range [-2^{19}, $2^{19}-1$]. The use of the "Literal" =153 will cause the assembler to generate a location for the literal (after the last programmer defined location), place the value indicated by the operand (interpreted as decimal, i.e., 153) in that location, and use the address of that location in the instruction. Literals are not valid on loading an immediate value, branch, store, shift, IO for reading, or IO for writing a character, and cannot be used with an indexed instruction.

   The "X" part is indicated by a parenthesized term following the "S" field. The term inside the parentheses can be a symbol or a decimal integer. In either case, its value must be between

0 and 3 (for symbols, this also means the value of the symbol can't be relocatable). If there is no parenthesized term, the operation does not have indexing.

4. Your assembler should be able to handle all of the machine instructions given in the "W10-560 machine definition" handout from Lab 2. The machine instruction mnemonics are given in that handout.

5. For the purposes of this assignment, you may assume a maximum of 100 symbols, 50 literals, and 200 source records in any given program. However, these constraints should be easy to change (be sure your programmer's guide gives instructions for changing them).

6. Provide the capability of handling the following pseudo-ops.

Mnemonic	Meaning
ORI	[**ORIgin**] This must be the first non-comment record in the source program. The operand, if present, must be a non-negative *decimal* integer ≤ 255. The operand indicates the absolute address at which the program is to be loaded. If the operand is absent, the program is assumed relocatable. The ORI statement also must have a label, which is the name of the segment.
END	[**END**] Signifies the end of the input program. An optional operand (non-negative *decimal* integer ≤ 255 or a symbol) indicates the address at which execution is to begin. If no operand is present, execution begins with the line containing the first non-pseudo-op. If the program is relocatable, the operand, if present, must be a symbol relative to this input program's load address.
EQU	[**EQUate**] Equates the symbol in the label field with the "value" of the operand field, essentially creating a constant within the assembler. The operand field can be a previously defined symbol or a non-negative decimal integer, having value ≤ 255.
NMD	[**NuMeric Data**] Defines a one word quantity whose contents is the value of the decimal integer in the operand field. This value is placed by the assembler in the word of memory that the NMD op occupies. The decimal integer can be positive or negative, in the range $[-2^{19}, 2^{19}-1]$. The assembler location counter is moved forward one word.
CCD	[**CharaCter Data**] Defines a one word quantity whose contents is defined by the two character ASCII string at the beginning of the operand field. These two characters are placed by the assembler in the upper sixteen bits of the word of memory that the CCD op occupies. The least significant four bits are all set to zero. The assembler location counter is moved forward one word. The character immediately following the second character of the string should either be a newline character or the space character. If it is not, the assembler should report an error. That the assembler enforces this restriction should make it more clear to the source code reader that the succeeding characters on the line belong to a comment.

Mnemonic	Meaning
RES	[**REServe storage**] Sets up a block of storage. The number of words in the block must be at least one and at most 255, as indicated by the positive decimal integer or previously defined absolute symbol in the operand field. This command moves the assembler location counter forward this number of words. No values are placed in this block by this pseudo op.

The ORI and EQU pseudo op instructions **require** labels. Labels are optional on the RES, CCD, and NMD pseudo ops. There are no labels on END pseudo ops. RES, CCD, and NMD require memory allocation while ORI, EQU, and END do not.

7. Provide the capability of handling the following synthetic instructions.

Mnemonic	Meaning
RET	[**RETurn from subroutine**] Is a convenient abbreviation for a branch (BR) instruction with R=3 and S=0. The operand field can be an absolute symbol or a decimal integer, having value 1, 2, or 3. Let's call the operand X. Then, "RET X" abbreviates "BR 3,0(X)", and means "return from this subroutine, using register X as the return address".
GTC	[**GeT Character**] Is a convenient abbreviation for an IO instruction with R=1 (and S=0). The operand field can be an absolute symbol or a decimal integer, having value 0, 1, 2, or 3. Let's call the operand X. Then, "GTC X" abbreviates "IO 1,0(X)", and means "get a character from input, placing it in (part of) register X".
PTC	[**PuT Character**] Is a convenient abbreviation for an IO instruction with R=3 (and S=0). The operand field can be an absolute symbol or a decimal integer, having value 0, 1, 2, or 3. Let's call the operand X. Then, "PTC X" abbreviates "IO 3,0(X)", and means "put a character to output from (part of) register X".

Output

Your assembler should have two primary outputs: an object file (which will subsequently be the input file for the linker/loader you will write in Lab 4) and a listing for the user.

Your object file should provide all of the information needed by a linker loader to generate the input to the 560 machine simulator defined in Lab 2. In particular, your object file should provide the memory contents associated with the instructions and data of the source program, along with information for the loader concerning the relocatability of the object file information and the size of the program's address space. Don't forget to deal with the location at which execution is to begin. For acceptance of this lab, the object file should be written to a file in the file system, information about which should be specified in your user's guide. Remember that this file will be "consumed" by (i.e., processed by, used as input for) the program you write in the next Lab.

The object file should define a memory image (footprint) that is laid out in the same order as presented in the assembly language input file. That is to say, by the order of its presentation, the assembly language file implicitly determines the text records' addresses. (While it is common for the text records in the object file to be listed in ascending order by address, this ordering is not a requirement. It is only the correct association between addresses and contents that is required.)

The listing you output for the user should contain the source program and its assembly, in some

suitable format. Remember that this output is for human consumption and should be designed to be as useful as possible to programmers. An example format follows.

L I S T I N G						
O B J C O D E			S O U R C E			
Loc Ctr (hex)	Contents (hex)	Reloc Info	Record Number (dec)	Label	Op'n	Operands/Comments
	.					
	.					
	.					

Your assembler should print meaningful diagnostics if errors in assembly are encountered. These diagnostics should appear in the standard error file (stderr), or in the listing file, or in both files. Your assembler should be capable of detecting errors involving each of the following conditions: invalid operation, invalid label, invalid operand (symbol, literal, integer, register), undefined reference, and multiple definition of a symbol.

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 three 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:

```
program-name assembly-language-source object-output listing-output
```

You must turn in a programmer's guide, user's guide, test plan with test results, meeting minutes, and a peer evaluation of the other members of your group. While a design review is not required for this lab, the grader and instructor are available during office hours and by appointment for consultation and informal design reviews. You should not begin coding until your design and pseudocode are done. It is much easier to change pseudocode than C++.

Approximate breakdown of Lab 3 grade

User's guide: 20%
Programmer's guide: 15%
Meeting minutes: 5%
Coding: 10%
Your testing: 10%
Our testing: 25%
Individual peer review: 15%

Sample Input (Lab 3)

Below is a sample assembly language program. The actual input file will obviously not include the "Addr" column and headings, and will conform to the format specifications given above.

Addr	Label	Instruction	Comments
	atest1	ORI 1	program header
01	X	NMD 22	M[1] = 22
02	Y	NMD –17	M[2] = –17
03	Z	NMD 5	M[3] = 5
04		RES 6	skip 6 words
0A		LD 0,Y	R0 ← (–17)
0B		MUL 0,Z	R0 ← (–17)*5 = (–85)
0C		ST 0,4	M[4] ← (–85)
0D		LD 1,1	R1 ← 22
0E		DIV 1,3	R1 ← 22/5 = 4
0F		ST 1,5	M[5] ← 4
10		LD 2,1	R2 ← 22
11		AND 2,3	R2 ← 0b10110 ∧ 0b00101 = 0b00100 = 4
12		ST 2,6	M[6] ← 4
13		LD 3,2	R3 ← -17
14		OR 3,3	R3 ← 0xFFFEF ∨ 0x5 = 0xFFFEF = -17
15		ST 3,7	M[7] ← -17
16		LD 1,Y	R1 ← -17
17		SHR 1,5	R1 ← 0xFFFEF shf.rt.5 = 0xFFFFF = -1
18		ST 1,8	M[8] ← -1
19		LD 0,X	R0 ← 22
1A		SHL 0,3	R0 ← 0b10110 shf.lf.3 = 0b10110000 = 176
1B		ST 0,9	M[9] ← 176
1C		SHR 0,3(2)	R0 ← 0b10110000 shf.rt.7 = 0b1 = 1
1D	;		Print "Done" (next 9 instructions)
1D		LDI 2,C	use R2 as a base address for char data
1E		LD 1,0(2)	print from R1
1F		IO 3,0(1)	print 'D'
20		SHL 1,8	bring 'o' into position
21		PTC 1	print 'o'
22		LD 3,1(2)	print from R3 this time ("ne" at offset 1)
23		PTC 3	print 'n'
24		SHL 3,8	bring 'e' into position
25		PTC 3	print 'e'
26		BR 0,0	halt quietly
27	C	CCD Do	M[39] contains "Do"
28		CCD ne	M[40] contains "ne"
		END 10	Execution is to start at Addr 0xA (= 10)