CMPE 3334 Program Assignment

Phase 2 (Assembler Pass 1)

due February 18, 2014

The second phase of your semester project is to write pass one of a two‑pass assembler for the SIC assembler language program. Since the only link between the assembler and the SIC simulator will be through the object code file, the assembler can be written and tested independently of the simulator. When testing is complete, modify your command line interpreter program so that it properly calls the assembler. Note that this and all of the other phases must run successfully on the Linux boxes in the Computer Science Computer Lab to receive credit.

Pass one will read each line of the source file, and begin the process of translating it to object code. (Note: it will be to your advantage to have a separate procedure handle reading, and perhaps tokenizing, the source file.) Among other things, this pass will create the symbol table. At the end of the pass, the symbol table should be printed, showing each symbol together with its location. This pass must also recognize and handle the assembler mnemonics in addition to the directives START, END, BYTE, WORD, RESB, and RESW. It should also be able to recognize the following types of errors (note that this is not an exhaustive list):

Duplicate labels

Illegal label (note: could specify why it is illegal, but not necessary)

Illegal operation

Missing or illegal operand on data storage directive

Missing or illegal operand on START directive

Missing or illegal operand on END directive (note: you do not need to check if the symbol is defined)

Too many symbols in source program

Program too long

After flagging an error, your assembler should continue processing the source file so that all errors are flagged in a single run. These errors will not be printed, but must be passed to the second pass of the assembler.

NOTE: Your assembler should not terminate abnormally because of anything (legal or not) in the source program being assembled.

Since some processing must take place on each line during pass one, it is advantageous to save this information in an intermediate file for use by pass two. Some things you may wish to place in this file are:

copy of source line

value of location counter

values of mnemonics used (since they had to be looked up)

operand (since you had to get it)

error messages (from pass 1 - best to use codes and not the actual message)

Note that pass two will do the actual encoding of the opcode and operand into object code, and create the listing file together with all error messages.

Hand in: 1. A documented listing of the assembler source code.

2. Two listings of assembler language source files (one with no errors, one with errors).

3. A listing of the symbol tables produced.

4. Copies of both intermediate files (write on the printout to indicate the salient features).

5. Make sure a copy is in your Linux directory in cmpe3334/phase2

The SIC Assembler Specifications

The SIC assembler language will consist of the 25 mnemonics listed in the instruction set (copy attached), together with the assembler directives (START, END, BYTE, WORD, RESB, RESW). The assembler language statements will consist of lines in the following free format:

{label} instruction {operand{,X}} {comment}

The items in curly braces, {..}, are optional, depending on the instruction and/or the programmer. A line may consist of just a comment, or just a label. If present, the label must begin in column 1. A period (.) in column 1 indicates the entire line is a comment. A blank in column 1 indicates that no label is present. Blank lines will be ignored. The only restriction is that there must be at least one space or tab separating each field present. The only exception is if indexing is requested ‑‑ don't put a space between the comma and the X.

A symbol is a string of up to 6 alphanumeric characters (letters and digits). The first character must be a letter. The assembler will not be case sensitive. You may assume there will be a maximum of 500 distinct labels.

The BYTE directive may take two types of operands: character strings or hexadecimal digits. These must be in the form C'...' or X'...'. The maximum length of a character string is 30 characters. The maximum length of the hexadecimal string is 16 bytes (32 hex digits). The number of hexadecimal digits must be even.

Instruction operands must be of the form operand or operand,X , where operand is either a symbol that is used as a label in the source program, or an actual hexadecimal address. Hexadecimal addresses that would begin with 'A' through 'F', must have a leading '0' to distinguish them from labels.

SIC Instruction Set

Mnemonic Opcode Effect

ADD m 18 A <- (A) + (m..m+2)

AND m 58 A <- (A) & (m..m+2) [bitwise]

COMP m 28 cond code <- (A) : (m..m+2)

DIV m 24 A <- (A) / (m..m+2)

J m 3C PC <- m

JEQ m 30 PC <- m if cond code set to =

JGT m 34 PC <- m if cond code set to >

JLT m 38 PC <- m if cond code set to <

JSUB m 48 L <- (PC); PC <- m

LDA m 00 A <- (m..m+2)

LDCH m 50 A[rightmost byte] <- (m)

LDL m 08 L <- (m..m+2)

LDX m 04 X <- (m..m+2)

MUL m 20 A <- (A) \* (m..m+2)

OR m 44 A <- (A) | (m..m+2) [bitwise]

RD m D8 A[rightmost byte] <- data from device specified by (m)

RSUB 4C PC <- (L)

STA m 0C m..m+2 <- (A)

STCH m 54 m <- (A)[rightmost byte]

STL m 14 m..m+2 <- (L)

STX m 10 m..m+2 <- (X)

SUB m 1C A <- (A) - (m..m+2)

TD m E0 Test device specified by (m)

TIX m 2C X <- (X) + 1; compare X and (m..m+2)

WD m DC Device specified by (m) <- (A)[rightmost byte]

SIC Assembler Directives

START n Program is to be loaded at location n (given in hexadecimal)

END label Physical end of program; label is first executable program statement

BYTE v Stores either character strings (C'...') or hexadecimal values (X'...')

WORD v Stores the value v in a WORD

RESB n Reserves space for n bytes

RESW n Reserves space for n words (3n bytes)