**Global variables:**

char \*prgName = NULL; //Name of Program

char \*iFileName = NULL; //Input file name

char \*oFileName = NULL; //Output file name

FILE\* infile = NULL; //Input file stream

FILE\* outfile = NULL; //Output file stream

Int error=0; //0 if no error; otherwise error code 1-4

Int origin; //Starting point; set by .orig

Int PC; //Program counter

Symbol table[100]; //Array of symbol objects for symbol table

**Imports:**

#include <stdio.h> /\* standard input/output library \*/

#include <stdlib.h> /\* Standard C Library \*/

#include <string.h> /\* String operations library \*/

#include <ctype.h> /\* Library for useful character operations \*/

#include <limits.h> /\* Library for definitions of common variable type characteristics \*/

**Hash defines:**

#define MAX\_LINE\_LENGTH 255

enum

{

DONE, OK, EMPTY\_LINE

};

**Provided functions:**

//Converts null terminated char string to a number

//Works on decimal or hex

Int toNum( char \* pStr );

//Takes a file and parses into elements; returns int based on findings

// FILE \* pInfile – file containing next line

// char \* pLine – location to put unparsed line

// char \*\* pLabel – Pointer to label

// char \*\* pOpcode – Pointer to opcode

// char \*\* pArg… - Pointer to up to 4 arguments

Int readAndParse( FILE \* pInfile, char \* pLine, char \*\* pLabel, char

\*\* pOpcode, char \*\* pArg1, char \*\* pArg2, char \*\* pArg3, char \*\* pArg4

)

**Primary functions:**

*File I/O*

Not too difficult. Mostly done in source code

*Generate symbol table*

Can be accomplished with a slightly modified version of readAndParse

*Generate code*

Meat of the problem. 21 opcodes plus 2 possible pseudo ops (.orig will be at the beginning). Will be a switch structure with an integer representing the specific opcode. Available will be:

* Int sBit – Steering bit set by the parseOpcode function
* strings written by the readAndParse function
* global variables

The individual case statements should write a complete line to the outfile.

**Helper functions:**

//Returns line number of a given symbol sym

//Return -1 if not found or invalid label

Int SearchSymbol(char \* sym);

//Returns integer corresponding to register of character string reg

//Returns -1 if not a valid register

Int RegNum(char \* reg);

//Generates an offset based on the current PC and an offset

//Sets isvalid to 0 if number of necessary digits for offset exceeds

//maxDigits; 1 otherwise

Int genOffsett(int offset, int maxDigits, int\* isValid);

//Returns integer equal to the binary representation of the opcode

//Return 16 for .fill, 17 for .end, and -1 for an invalid opcode

//sets steering if necessary for special cases due to overloaded opcodes

//Special Cases:

//JSR - steering=1

//JSRR – steering=0

//LSHF – steering=0

//RSHFL – steering=1

//RSHFA – steering=3

//NOP – steering=1

//HALT – steering=1

//NOT – steering=1

//JMP – steering=0

//RET – steering=1

//Otherwise – steering=0

Int parseOpcode(char \* opcode, int \* steering);

**Notes:**

* If an error is detected, close the file and then throw an error with the function exit(CODE); where CODE is the number representing the error code
* Represent the current binary code as an integer. It can then be outputted as below where pOutfile is the location of the output and lInstr is the integer:
  + fprintf( pOutfile, "0x%0.4X\n", lInstr );