Assembler.c

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#define MAXLINELENGTH 1000

int readAndParse(FILE \*, char \*, char \*, char \*, char \*, char \*);

int isNumber(char \*);

int returnIndex(FILE \*, char \*);

int Char2Num(char \*);

void check(int);

int main(int argc, char \*argv[]) {

char \*inFileString, \*outFileString;

FILE \*inFilePtr, \*inFilePtr2, \*outFilePtr;

char label[MAXLINELENGTH], opcode[MAXLINELENGTH], arg0[MAXLINELENGTH];

char arg1[MAXLINELENGTH], arg2[MAXLINELENGTH];

if (argc!= 3) {

printf("error: usage: %s <assembly-code-file> <machine-code-file>\n"

, argv[0]);

exit(1);

}

inFileString = argv[1];

outFileString = argv[2];

inFilePtr = fopen(inFileString, "r");

inFilePtr2 = fopen(inFileString, "r");

//check if files exist and have values

if (inFilePtr == NULL) {

printf("error in opening %s\n", inFileString);

exit(1);

}

outFilePtr = fopen(outFileString, "w");

if (outFilePtr == NULL) {

printf("error in opening %s\n", outFileString);

}

/\*

while (readAndParse(inFilePtr, label, opcode, arg0, arg1, arg2)) {

//reached end of file

if (strcmp(label, "")) {

int same = 0;

char currentLabel[MAXLINELENGTH];

strcpy(currentLabel, label);

while (readAndParse(inFilePtr2, label, opcode, arg0, arg1, arg2)) {

if (!strcmp(currentLabel, label)) {

same++;

}

}

if (same >= 2) {

printf("Duplicated!\n");

exit(1);

}

rewind(inFilePtr2);

}

}\*/

rewind(inFilePtr);

rewind(inFilePtr2);

int PC = 0;

while (readAndParse(inFilePtr, label, opcode, arg0, arg1, arg2)) {

int op;

int arg0num, arg1num, arg2num, offset;

int mc;

if (!strcmp(opcode, "add")) {

op = 0;

arg0num = Char2Num(arg0);

arg1num = Char2Num(arg1);

arg2num = Char2Num(arg2);

offset = arg2num;

}

else if (!strcmp(opcode, "nor")) {

op = 1;

arg0num = Char2Num(arg0);

arg1num = Char2Num(arg1);

arg2num = Char2Num(arg2);

offset = arg2num;

}

else if (!strcmp(opcode, "lw")) {

op = 2;

arg0num = Char2Num(arg0);

arg1num = Char2Num(arg1);

if (isNumber(arg2)) {

arg2num = Char2Num(arg2);

offset = arg2num;

}

else {

arg2num = returnIndex(inFilePtr2, arg2);

offset = arg2num;

}

}

else if (!strcmp(opcode, "sw")) {

op = 3;

arg0num = Char2Num(arg0);

arg1num = Char2Num(arg1);

if (isNumber(arg2)) {

arg2num = Char2Num(arg2);

offset = arg2num;

}

else {

arg2num = returnIndex(inFilePtr2, arg2);

offset = arg2num;

}

}

else if (!strcmp(opcode, "beq")) {

op = 4;

arg0num = Char2Num(arg0);

arg1num = Char2Num(arg1);

if (isNumber(arg2)) {

arg2num = Char2Num(arg2);

offset = arg2num;

}

else {

arg2num = returnIndex(inFilePtr2, arg2);

offset = arg2num - PC - 1;

}

}

else if (!strcmp(opcode, "jalr")) {

op = 5;

arg0num = Char2Num(arg0);

arg1num = Char2Num(arg1);

offset = 0;

}

else if (!strcmp(opcode, ".fill")) {

if (isNumber(arg0)) {

arg0num = Char2Num(arg0);

mc = arg0num;

fprintf(outFilePtr, "%d\n", mc);

PC++;

continue;

}

else {

mc = returnIndex(inFilePtr2, arg0);

fprintf(outFilePtr, "%d\n", mc);

PC++;

continue;

}

}

else if (!strcmp(opcode, "halt")) {

op = 6;

arg0num = arg1num = arg2num = offset = 0;

}

else if (!strcmp(opcode, "noop")) {

op = 7;

arg0num = arg1num = arg2num = offset = 0;

}

else {

printf("undefined op: %s\n", opcode);

exit(1);

}

check(offset);

if (offset < 0) {

offset += 65536;

}

mc = op \* 4194304 + arg0num \* 524288 + arg1num \* 65536 + offset;

fprintf(outFilePtr, "%d\n", mc);

printf("\t0x%X\n", mc);

PC++;

}

printf("Created Machine Code File\n");

fclose(inFilePtr2);

fclose(inFilePtr);

fclose(outFilePtr);

exit(0);

return 0;

}

int readAndParse(FILE \*inFilePtr, char \*label,

char \*opcode, char \*arg0, char \*arg1, char \*arg2) {

char line[MAXLINELENGTH];

char \*ptr = line;

//delete prior values

label[0] = opcode[0] = arg0[0] = arg1[0] = arg2[0] = '\0';

if (fgets(line, MAXLINELENGTH, inFilePtr) == NULL) {

return 0;

}

if (strchr(line, '\n') == NULL) {

printf("error: line too long\n");

exit(1);

}

ptr = line;

if (sscanf(ptr, "%[^\t\n\r ]", label)) {

ptr += strlen(label);

}

//Parse the rest of the line

sscanf(ptr, "%\*[\t\n\r ]%[^\t\n\r ]%\*[\t\n\r ]%[^\t\n\r ]%\*[\t\n\r "

"]%[^\t\n\r ]%\*[\t\n\r ]%[^\t\n\r ]", opcode, arg0, arg1, arg2);

return 1;

}

int isNumber(char \*string) {

int i;

return ((sscanf(string, "%d", &i)) == 1);

}

int returnIndex(FILE \*inFilePtr2, char \*inputlabel) {

int address;

int success = 0;

int counter = 0;

char label[MAXLINELENGTH], opcode[MAXLINELENGTH], arg0[MAXLINELENGTH],

arg1[MAXLINELENGTH], arg2[MAXLINELENGTH];

rewind(inFilePtr2);

while (readAndParse(inFilePtr2, label, opcode, arg0, arg1, arg2)) {

if (!strcmp(label, inputlabel)) {

address = counter;

success = 1;

}

counter++;

}

if (success) {

return address;

} else {

printf("Label undefined!\n");

exit(1);

}

}

int Char2Num(char \*string) {

int i;

sscanf(string, "%d", &i);

return i;

}

void check(int offset) {

if (offset >= -32768 && offset <= 32767) {

}

else {

printf("Overflow offset\n");

exit(1);

}

}

Multiplication.c

#include "assembler.h"

int

main(int argc, char \*argv[])

{

int bRes, bOpcode, bRem;

char \*inFileString, \*outFileString;

FILE \*inFilePtr, \*outFilePtr;

char label[MAXLINELENGTH], opcode[MAXLINELENGTH], arg0[MAXLINELENGTH],

arg1[MAXLINELENGTH], arg2[MAXLINELENGTH];

if (argc != 3) {

printf("error: usage: %s <assembly-code-file> <machine-code-file>\n", argv[0]);

exit(1);

}

inFileString = argv[1];

outFileString = argv[2];

inFilePtr = fopen(inFileString, "r");

if (inFilePtr == NULL) {

printf("error in opening %s\n", inFileString);

exit(1);

}

outFilePtr = fopen(outFileString, "w");

if (outFilePtr == NULL) {

printf("Error in opening %s\n", outFileString);

exit(1);

}

//Phase 1

while (1) {

if (!readAndParse(inFilePtr, label, opcode, arg0, arg1, arg2)) {

/\* reached end of file \*/

break;

}

if (strlen(label) > 6) {

printf("Error : label is too long.\n");

exit(1);

}

if (!strcmp(label, "")) {

PC++;

continue;

}

if (getLabelAddress(label) != -1) {

printf("Error : duplicated label.\n ");

exit(1);

}

strcpy(Label\_Map[PC++], label);

}

MEMEND = PC;

//Phase 2

PC = 0;

rewind(inFilePtr);

/\* here is an example for how to use readAndParse to read a line from

inFilePtr \*/

while (1) {

bRes = bOpcode = bRem = 0;

if (!readAndParse(inFilePtr, label, opcode, arg0, arg1, arg2)) {

break;

}

/\* after doing a readAndParse, you may want to do the following to test the

opcode \*/

if (!strcmp(opcode, "add")) {

bOpcode = OP\_ADD;

bRem = encodeRType(arg0, arg1, arg2);

}

else if (!strcmp(opcode, "nor")) {

bOpcode = OP\_NOR;

bRem = encodeRType(arg0, arg1, arg2);

}

else if (!strcmp(opcode, "lw")) {

bOpcode = OP\_LW;

bRem = encodeIType(arg0, arg1, arg2, 0);

}

else if (!strcmp(opcode, "sw")) {

bOpcode = OP\_SW;

bRem = encodeIType(arg0, arg1, arg2, 0);

}

else if (!strcmp(opcode, "beq")) {

bOpcode = OP\_BEQ;

bRem = encodeIType(arg0, arg1, arg2, 1);

}

else if (!strcmp(opcode, "jalr")) {

bOpcode = OP\_JALR;

bRem = encodeJType(arg0, arg1);

}

else if (!strcmp(opcode, "halt")) {

bOpcode = OP\_HALT;

}

else if (!strcmp(opcode, "noop")) {

bOpcode = OP\_NOOP;

}

else if (!strcmp(opcode, ".fill")) {

bRem = encodeFType(arg0);

}

else {

exit(1);

}

bRes = (bOpcode | bRem);

Memory[PC++] = bRes;

}

printMemory(outFilePtr);

return 0;

}

void

printMemory(FILE \*fpr) {

for (PC = 0; PC < MEMEND; PC++) {

printf("(address %d): %d (hex 0x%x)\n", PC, Memory[PC], Memory[PC]);

if (fpr != NULL) {

fprintf(fpr, "%d\n", Memory[PC]);

}

}

}

/\*

\* Read and parse a line of the assembly-language file. Fields are returned

\* in label, opcode, arg0, arg1, arg2 (these strings must have memory already

\* allocated to them).

\*

\* Return values:

\* 0 if reached end of file

\* 1 if all went well

\*

\* exit(1) if line is too long.

\*/

int

readAndParse(FILE \*inFilePtr, char \*label, char \*opcode, char \*arg0,

char \*arg1, char \*arg2)

{

char line[MAXLINELENGTH];

char \*ptr = line;

/\* delete prior values \*/

label[0] = opcode[0] = arg0[0] = arg1[0] = arg2[0] = '\0';

/\* read the line from the assembly-language file \*/

if (fgets(line, MAXLINELENGTH, inFilePtr) == NULL) {

/\* reached end of file \*/

return(0);

}

/\* check for line too long (by looking for a \n) \*/

if (strchr(line, '\n') == NULL) {

/\* line too long \*/

printf("error: line too long\n");

exit(1);

}

/\* is there a label? \*/

ptr = line;

if (sscanf(ptr, "%[^\t\n\r ]", label)) {

/\* successfully read label; advance pointer over the label \*/

ptr += strlen(label);

}

/\*

\* Parse the rest of the line. Would be nice to have real regular

\* expressions, but scanf will suffice.

\*/

sscanf(ptr, "%\*[\t\n\r ]%[^\t\n\r ]%\*[\t\n\r ]%[^\t\n\r ]%\*[\t\n\r ]%[^\t\n\r ]%\*[\t\n\r ]%[^\t\n\r ]",

opcode, arg0, arg1, arg2);

return(1);

}

int

isNumber(char \*string)

{

/\* return 1 if string is a number \*/

int i;

return((sscanf(string, "%d", &i)) == 1);

}

int

getLabelAddress(char \*label) {

int i;

for (i = 0; i < MEMEND; i++) {

if (!strcmp(Label\_Map[i], label)) {

return i;

}

}

return -1;

}

int encodeRType(char \*arg0, char \*arg1, char \*arg2) {

if (!(isNumber(arg0) && isNumber(arg1) && isNumber(arg2))) {

printf("Error : some of arguments are not a number. \n");

exit(1);

}

int bRegA = atoi(arg0) << REG\_A;

int bRegB = atoi(arg1) << REG\_B;

int bOffset = atoi(arg2);

return bRegA | bRegB | bOffset;

}

int encodeIType(char \*arg0, char \*arg1, char \*arg2, int beq) {

if (!(isNumber(arg0) && isNumber(arg1))) {

printf("Error : some of arguments are not a number.\n");

exit(1);

}

if (!strcmp(arg2, "")) {

printf("Error : offset is empty.\n");

exit(1);

}

int bRegA = atoi(arg0) << REG\_A;

int bRegB = atoi(arg1) << REG\_B;

int bOffset;

if (isNumber(arg2)) {

bOffset = atoi(arg2);

}

else {

if ((bOffset = getLabelAddress(arg2)) == -1) {

printf("Error : unknown label.\n");

printf("%s %s %s\n", arg0, arg1, arg2);

while (1);

exit(1);

}

if (beq) {

bOffset -= (PC + 1);

}

}

if (bOffset < -32768 || bOffset > 32767) {

printf("Error : out of range. \n");

exit(1);

}

if (beq) {

bOffset &= MASK;

}

return bRegA | bRegB | bOffset;

}

int encodeJType(char \*arg0, char \*arg1) {

if (!(isNumber(arg0) && isNumber(arg1))) {

printf("Error : some of arguments are not a number. \n");

exit(1);

}

int bRegA = atoi(arg0) << REG\_A;

int bRegB = atoi(arg1) << REG\_B;

return bRegA | bRegB;

}

int encodeFType(char \*arg0) {

if (!strcmp(arg0, "")) {

printf("Error : offset is empty.\n");

exit(1);

}

int n;

if (isNumber(arg0)) {

n = atoi(arg0);

}

else {

if ((n = getLabelAddress(arg0)) == -1) {

printf("Error : unknown label.\n");

exit(1);

}

}

return n;

}