**/\*main.c\*/**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "command\_funcs.c"

#include "sicengine.c"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ana Cecilia Sánchez Ramos

SYS PROG - CSCI 3334.01

Phase 4.

Integrates command interpreter with

simulation code of SIC machine.

Missing implementation of

load, debug & dump

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(){

SICInit();

char ans[100]; // string that stores the whole line

char comm[10], param1[10], param2[20]; //command + parameters

int ans\_len, n; //string length + number of parameters

int tru = 1;

printf("Welcome!\n");

printf("Here's the available commands:\n\n");

menu();

while(1){

printf("\nPlease enter a command + parameters: ");

fgets(ans, 100, stdin); /\* read in a line \*/

ans\_len = strlen(ans) - 1; /\* get rid of trailing newline character \*/

if (ans[ans\_len] == '\n'){

ans[ans\_len] = '\0';

}

breakLine(ans, comm, param1, param2, &n);

tru = strcmp(comm, "exit");

if(tru == 0 && param1[0] == '\0' && param2[0] == '\0'){

exit(0);

}

else if (tru == 0 && param1[0] != '\0'){

printf("No need for parameters! check \"help\" for more info\n");

}

else{

check(comm,param1,param2);

}

}

return 0;

}

**/\*command\_funcs.c\*/**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "commands.h"

//\*FUNCTION-> break up string input\*/

void breakLine(char \*str, char \*comm, char \*param1, char \*param2, int \*n\_words) {

int str\_ct = 0, comm\_ct = 0, p1\_ct = 0, p2\_ct = 0;

memset(&comm[0],0,sizeof(comm));

memset(&param1[0],0,sizeof(param1));

memset(&param2[0],0,sizeof(param2));

\*n\_words = 0;

while(str[str\_ct] == ' '){ //check for spaces before command

str\_ct++;

}

while(str[str\_ct] != '\0' && str[str\_ct] != ' ') { //find command

comm[comm\_ct] = str[str\_ct];

str\_ct++;

comm\_ct++;

if(str[str\_ct] == '\0' || str[str\_ct] == ' '){ //null-terminate it

comm[comm\_ct] = '\0';

comm\_ct++;

(\*n\_words)++;

}

}

while(str[str\_ct] == ' '){

str\_ct++;

}

// find 1st parameter + store it

if(str[str\_ct] != '\0' && str[str\_ct] !=' '){

while(str[str\_ct] != '\0' && str[str\_ct]!=' '){

param1[p1\_ct] = str[str\_ct];

p1\_ct++;

str\_ct++;

if(str[str\_ct] == '\0' || str[str\_ct] == ' '){

param1[p1\_ct] = '\0';

p1\_ct++;

(\*n\_words)++;

}

}

}

else { param1[0]='\0'; }

while(str[str\_ct] == ' '){

str\_ct++;

}

//find 2nd parameter + store it

if(str[str\_ct] != '\0' && str[str\_ct] !=' ') {

while(str[str\_ct] != '\0' && str[str\_ct] != ' '){

param2[p2\_ct] = str[str\_ct];

p2\_ct++;

str\_ct++;

if(str[str\_ct] == '\0' || str[str\_ct] == ' '){

param2[p2\_ct] = '\0';

p2\_ct++;

(\*n\_words)++;

}

}

}

else{ param2[0]='\0'; }

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//\*FUNCTION-> check user input\*/

void check(char \*comm, char \*param1, char \*param2){

char \* commands[8]; //create array to store commands

char\* input;

int cmp;

int index;

//❤--------- commands ---------❤

commands[0] = "load";

commands[1] = "execute";

commands[2] = "debug";

commands[3] = "dump";

commands[4] = "help";

commands[5] = "assemble";

commands[6] = "directory";

//❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤

for(int i = 0 ; i < 8; i++){ /\*loop through array to find command\*/

cmp=strcmp(comm,commands[i]);

if(cmp == 0) {

input = comm;

index = i;

break;

}

}

/\*check each case\*/

if(index == 0){ //load

if(param1[0] != '\0' && param2[0] == '\0'){

load();

}

else if(param1[0] == '\0'){

printf("You're missing parameters, check \"help\" for more info\n");

}

else{

printf("Too many parameters! check \"help\" for more info\n");

}

}

else if(index == 1 || index == 2){ //execute or debug

if(param1[0] == '\0' && param2[0] == '\0'){

if(index == 1){ //execute

execute();

}

else{ //index = 2 -> debug

debug();

}

}

else{

printf("No need for parameters! check \"help\" for more info\n");

}

}

else if(index == 3){ //dump

if(param1[0] != '\0' && param2[0] != '\0'){

dump();

}

else if(param1[0] == '\0' || (param1[0] !='\0' && param2[0] =='\0')){

printf("You're missing parameters, check \"help\" for more info\n");

}

else{

printf("Too many parameters! check \"help\" for more info\n");

}

}

else if(index == 4){ //help

if(param1[0] == '\0' && param2[0] == '\0'){

menu();

}

else{

printf("No need for parameters! check \"help\" for more info\n");

}

}

else if(index == 5){ //assemble

if(param1[0] != '\0' && param2[0] == '\0'){

assemble(param1);

}

else if(param1[0] == '\0'){

printf("You're missing parameters, check \"help\" for more info\n");

}

else{

printf("Too many parameters! check \"help\" for more info\n");

}

}

else if(index == 6) { //directory

if(param1[0]=='\0' && param2[0] == '\0'){

system("ls");

}

else{

printf("No need for parameters! check \"help\" for more info\n");

}

}

else{

printf("Sorry, not a valid command!!! check \"help\" for more info\n");

}

}

**/\*commands.h\*/**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "phase3\_funcs.h"

/\*\*\*\*\*\*\*\*\*\*\*\*protoypes\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void menu(); //<--- used for "help" as well

void load();

void execute();

void debug();

void dump();

void assemble(char\*);

/\*\*\*\*\*\*\*\*\*\*\*\*definitions\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void menu(){

printf("load <filename> \nexecute\ndebug\ndump <start> <end>\nhelp\nassemble <filename>\ndirectory\nexit\n");

}

void load(){

printf("You've reached the load command\n");

}

void execute(){

unsigned long PCloc = 0;

SICRun(&PCloc,0);

printf("You've reached the execute command\n");

}

void debug(){

printf("You've reached the debug command \n");

}

void dump(){

printf("You've reached the dump command\n");

}

void assemble(char \* src\_file){

build\_optab();

pass1(src\_file);

symtab\_to\_file();

pass2();

}

**/\*phase3\_funcs.h\*/**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "phase2\_funcs.h"

//\*find the opcode from the intermediate file\*/

char\* find\_opcode(char \*line){

static char opcode[6];

int tab\_ct = 0; //<---- amount of tabs (max = 2)

int i = 0; //<---- moving

int op\_ct = 0; //<---- opcode counter

while(tab\_ct != 2){ //<--- count 2 tabs (#tabs before opcode)

if(line[i] == '\t'){

tab\_ct++;

}

i++;

}

while(line[i] != '\t'){ //<--- put opcode in string

opcode[op\_ct] = line[i];

op\_ct++;

i++;

}

opcode[op\_ct] = '\0'; //<--- null-terminate string

return opcode;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//\*find the operand from the intermediate file -> used to create object code

char\* find\_operand(char \*line){

static char operand[10];

int tab\_ct = 0; //<---- amount of tabs (max = 3)

int i = 0; //<---- moving

int op\_ct = 0; //<---- operand counter

while(tab\_ct != 3){ //<--- count 3 tabs (#tabs before opcode)

if(line[i] == '\t'){

tab\_ct++;

}

i++;

}

while(line[i] != '\t'){ //<--- put opcode in string

operand[op\_ct] = line[i];

op\_ct++;

i++;

}

operand[op\_ct] = '\0'; //<--- null-terminate string

return operand;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//\*write line to listing file

void w\_listing\_file(FILE\* output\_ptr, char\* line, char\* opcode, char\* mnemonic, char\* address){

char obj\_code[20];

int i = 0;

if(strcmp(opcode,"START") == 0 || strcmp(opcode, "END") == 0){

fprintf(output\_ptr,"\t%s\n", line);

}

else if(strcmp(opcode,"WORD") != 0 && strcmp(opcode,"RESW") != 0 && strcmp(opcode,"RESB") != 0){

for(i; i < strlen(mnemonic); i++){

obj\_code[i] = mnemonic[i];

}

obj\_code[i] = '\0';

if(strcmp(opcode,"RSUB") == 0){

strcat(obj\_code,"0000");

}

else{

strcat(obj\_code,address);

}

fprintf(output\_ptr,"%s\t%s\n",obj\_code, line);

}

else{

fprintf(output\_ptr,"\t%s\n", line); //testing purposes

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//\*get object code for BYTE

void getmnem\_BYTE(char\* operand, char\* mnemonic){

int m\_ct= 0;

if(operand[0] == 'C'){

char con\_mnem[5];

for(int i = 0 ; i < strlen(operand); i++){ //get rid of apostrophes + C

if(operand[i] != '\'' && operand[i] != 'C'){

con\_mnem[m\_ct] = operand[i];

m\_ct++;

}

}

for(int i = 0; i<strlen(con\_mnem); i++){ //get hex vals of constants

sprintf(mnemonic+i\*2, "%02X", con\_mnem[i]);

}

}

else{

for(int i = 0 ; i < strlen(operand); i ++){ //if X

if(operand[i] != 'X' && operand[i] != '\''){

mnemonic[m\_ct] = operand[i];

m\_ct++;

}

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void pass2(){

FILE \* file\_ptr;

file\_ptr = fopen("intermediate\_file.txt","r"); //<--- read from intermediate file

FILE \* output\_ptr;

output\_ptr = fopen("listing\_file.txt","w");

char line[80];

char opcode[7], mnemonic[7];

char operand[10], address[7];

char obj\_code[20];

if(file\_ptr == NULL){

printf("can't reach file\n");

}

else{

fgets(line,80,file\_ptr); //<--- get headers to ignore them

while(!feof(file\_ptr)){

memset(&opcode[0],0,sizeof(opcode));

memset(&opcode[0],0,sizeof(mnemonic));

memset(&operand[0],0,sizeof(operand));

memset(&address[0],0,sizeof(address));

memset(&obj\_code[0],0,sizeof(obj\_code));

fgets(line,80,file\_ptr);

strcpy(opcode,find\_opcode(line)); //<--- get opcode

if(strcmp(opcode,"RSUB")==0){

strcpy(mnemonic,mnem\_value(opcode)); //<--- get mnemonic

}

else if(strcmp(opcode,"START") != 0 && strcmp(opcode,"END") != 0 && strcmp(opcode,"RSUB") != 0 &&

strcmp(opcode, "BYTE") != 0 && strcmp(opcode,"WORD") != 0 && strcmp(opcode,"RESW") != 0 && strcmp(opcode,"RESB") != 0){

strcpy(operand,find\_operand(line)); //<---get operand

if(strcmp(operand,"BUFFER,X") == 0){ //exception

strcpy(operand, "BUFFER");

}

strcpy(address,address\_val(operand));

strcpy(mnemonic,mnem\_value(opcode)); //<--- get mnemonic

}

else{

operand[0] = '\0';

/\*check if BYTE or WORD\*/

if(strcmp(opcode,"BYTE") == 0){

strcpy(operand,find\_operand(line)); //<---get operand & call function to get object code

getmnem\_BYTE(operand,mnemonic);

}

else if(strcmp(opcode,"WORD") == 0){

/\*call function to get objcode\*/

}

}

w\_listing\_file(output\_ptr,line, opcode, mnemonic, address);

}

}

fclose(output\_ptr);

fclose(file\_ptr);

}

**/\*phase2\_funcs.h\*/**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "hashtable.h"

void build\_optab(){

hasht = malloc(max \* sizeof(linkedList));

init\_hasht();

char\* op[][2] = {{"ADD", "18"}, {"RSUB", "4C"},

{"AND", "58"}, {"STA", "0C"},

{"COMP", "28"}, {"STCH", "54"},

{"DIV", "24"}, {"STL", "14"},

{"J", "3C"}, {"STX", "10"},

{"JEQ", "30"}, {"SUB", "1C"},

{"JGT", "34"}, {"TD", "E0"},

{"JLT", "38"}, {"TIX", "2C"},

{"JSUB", "48"}, {"WD", "DC"},

{"LDA", "00"}, {"LDCH", "50"},

{"LDL", "08"}, {"LDX", "04"},

{"MUL", "20"}, {"OR", "44"},

{"RD", "D8"}};

for(int i=0; i<25; i++){

insert(op[i][0],op[i][1]);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void split\_into(char \*str, char\* label, char\* opcode, char\* operand) {

memset(&label[0],0,sizeof(label));

memset(&opcode[0],0,sizeof(opcode));

memset(&operand[0],0,sizeof(operand));

if(str[0] == '.'){ //<---comment

// ignore

}

else if(str[0] == ' ' || str[0] == '\t' || str[0] == '\n' || str[0] == '\0'){ //no label

sscanf(str, "%s %s", opcode, operand);

memset(&label[0],0,sizeof(label));

if(strcmp(opcode,"RSUB") == 0){

operand[0] = '\0';

}

}

else{

sscanf( str, "%s %s %s", label,opcode,operand);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int byte\_length(char\* operand, char\* error){

int b\_len = 0;

if(operand[0] == 'C'){

b\_len = strlen(operand) - 3;

}

else{ //<---if it's an X

b\_len = strlen(operand) - 3;

if(b\_len % 2 != 0){

strcat(error,"invalid operand! ");

} else{

b\_len = b\_len / 2;

}

}

return b\_len;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void handle\_line(int\* loc\_hexval, char\* locctr, char\* label, char\* opcode, char\* operand, char\* error){

//❤--------- check label !!! ---------❤

if(label[0] == '\0'){}

else if(lookup\_symbol(label) == 0){ //<--- symbol already exists

strcat(error,"duplicated label! ");

}else{

symtab\_insert(label, locctr);

}

//❤--------- check opcode + add to locctr !!! ---------❤

if(lookup\_opcode(opcode)==0 || strcmp(opcode,"WORD") == 0){ //add +3 to locctr

\*loc\_hexval = (int)strtol(locctr, NULL, 16);

\*loc\_hexval = (\*loc\_hexval) + 3;

// itoa(\*loc\_hexval,locctr,16);

sprintf(locctr, "%x", \*loc\_hexval); //converting loc decimal to hex

}

else if(strcmp(opcode,"RESB") == 0){ //add + #OPERAND to locctr

\*loc\_hexval = (int)strtol(locctr, NULL, 16);

int operand\_hexval = (int)strtol(operand, NULL, 10);

\*loc\_hexval = (\*loc\_hexval) + operand\_hexval;

// itoa(\*loc\_hexval,locctr,16);

sprintf(locctr, "%x", \*loc\_hexval); //converting loc decimal to hex

}

else if(strcmp(opcode,"RESW") == 0){ //add + 3 \* #OPERAND to locctr

\*loc\_hexval = (int)strtol(locctr, NULL, 16);

int operand\_hexval = (int)strtol(operand, NULL, 16);

\*loc\_hexval = (\*loc\_hexval) + (3 \* operand\_hexval);

// itoa(\*loc\_hexval,locctr,16);

sprintf(locctr, "%x", \*loc\_hexval); //converting loc decimal to hex

}

else if(strcmp(opcode, "BYTE") == 0){ //find length of constant in bytes & add length to locctr

if(operand[0] == 'C' || operand[0] == 'X'){

if(operand[1] == '\'' && operand[strlen(operand)-1] == '\''){

\*loc\_hexval = (int)strtol(locctr, NULL, 16);

\*loc\_hexval = (\*loc\_hexval) + (byte\_length(operand, error));

// itoa(\*loc\_hexval,locctr,16);

sprintf(locctr, "%x", \*loc\_hexval); //converting loc decimal to hex

}else{

strcat(error, "invalid operand! ");

}

}

else{

strcat(error, "invalid operand! ");

}

}

else{

strcat(error,"invalid opcode! ");

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void program\_length(int\* start, char\* start\_char, int\* end, char\* end\_char, int\* loc\_hexval){

\*start = (int)strtol(start\_char, NULL, 16);

\*end = (int)strtol(end\_char, NULL, 16);

\*end = (\*loc\_hexval)-(\*start);

//itoa(\*end, end\_char, 16);

sprintf(end\_char, "%x", \*end); //converting loc decimal to hex

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//\*----- P A S S 1 -----\*

void pass1(char \* src\_file){

//❤--------- VARIABLES ---------❤

char locctr[7], st\_address[7], end\_address[7];

locctr[0] = '\0', st\_address[0] = '\0', end\_address[0] = '\0';

int loc\_hexval = 0 ,int\_st = 0, int\_end = 0;

char line[80], label[7], opcode[7], operand[10], error[90];

error[0] = '\0';

FILE \* file\_ptr; //<---- file we read from

FILE \* output\_ptr; //<-----output file

//❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤

symbol\_table = malloc(symtab\_max \* sizeof(symtab\_linkedList));

build\_symtab();

if((file\_ptr = fopen(src\_file,"r")) == NULL){ //source file can't be opened

printf("can't reach file\n");

}

else{

if((output\_ptr = fopen("intermediate\_file.txt","w")) == NULL){

printf("can't open intermediate file\n"); //intermediate file can't be opened

}

else{

fprintf(output\_ptr,"%s\t%s\t%s\t%s\t%s\n", "LOCCTR","LABEL","OPCODE","OPERAND","ERRORS"); //intermediate file headers

fgets(line,80,file\_ptr); //store first string in line

split\_into(line, label, opcode, operand);

if(strcmp(opcode,"START") == 0){

strcpy(st\_address, operand); //set starting address + locctr

strcpy(locctr,operand);

loc\_hexval = (int)strtol(locctr, NULL, 16); //<---- hex value to decimal integer

int\_st = (int)strtol(st\_address,NULL,16);

fprintf(output\_ptr,"%x\t%s\t%s\t%s\t%s\n", loc\_hexval, label, opcode, operand, error);

fgets(line,80,file\_ptr); //read next line

}

while(!feof(file\_ptr)){

memset(&error[0],0,sizeof(error));

split\_into(line, label, opcode, operand);

if(label[0]=='\0' && opcode[0]=='\0' && operand[0]=='\0'){}

else if(line[0] != '.'){ //<----- not a comment

fprintf(output\_ptr,"%x\t%s\t%s\t%s\t%s\n", loc\_hexval, label, opcode, operand, error); //<---write to intermediate file

if(strcmp(opcode,"END") != 0){

handle\_line(&loc\_hexval, locctr, label, opcode, operand, error);

}

else{ //<--- +3 to locctr when END is reached

if(label[0] != '\0'){

strcat(error,"END does not need a label! ");

}

loc\_hexval = (int)strtol(locctr, NULL, 16);

loc\_hexval = (loc\_hexval) + 3;

// itoa(loc\_hexval,locctr,16);

sprintf(locctr, "%x", loc\_hexval); //converting loc decimal to hex\*/

}

}

fgets(line,80,file\_ptr);

}

}

fclose(output\_ptr);

}

program\_length(&int\_st, st\_address, &int\_end, end\_address, &loc\_hexval); //<--- calculate length of program

printf("File has been successfully assembled! \n");

fclose(file\_ptr);

}

**/\*hashtable.h\*/**

#include<stdio.h>

#include<stdlib.h>

#include <string.h>

//\*\*\*\*\*\*\*\*\*\*\*\*\*\* O P C O D E T A B L E \*\*\*\*\*\*\*\*\*\*\*\*\*//

typedef struct node {

char opcode[7];

char mnemonic[7];

struct node \*next;

}node;

typedef struct linkedList { //stores a list at each index

struct node \*head;

struct node \*tail;

}linkedList;

linkedList \*hasht = NULL;

int max = 26;

int hashFunction(char\* opcode){ //returns index where we'll store item

unsigned int wordVal = 0;

for(int i = 0; i < strlen(opcode); i++){

wordVal = 37 \* wordVal + (opcode[i] - 48); //convert opcode -> number(key)

}

return wordVal % max;

}

//prototypes

void init\_hasht();

void insert(char\* opc, char\* mnem);

int lookup\_opcode(char\* str);

//definitions

void insert(char\* opc, char\* mnem) {

int index = hashFunction(opc);

node \*list = hasht[index].head;

node \*item = malloc(sizeof(struct node)); //item to be added

strcpy(item->opcode,opc);

strcpy(item->mnemonic,mnem);

item->next = NULL;

if (list == NULL) //linked list is empty

{

hasht[index].head = item;

hasht[index].tail = item;

}

else { //add to end of list

hasht[index].tail->next = item;

hasht[index].tail = item;

}

}

int lookup\_opcode(char\* str){

int index = hashFunction(str);

node \*temp = hasht[index].head;

while(temp!=NULL){

if(strcmp(str,temp->opcode) == 0){

return 0;

}

temp = temp->next;

}

return 1;

}

char\* mnem\_value(char\* str){

int index = hashFunction(str);

node \*temp = hasht[index].head;

while(temp!=NULL){

if(strcmp(str,temp->opcode) == 0){

return temp->mnemonic;

}

temp = temp->next;

}

return "nop";

}

void init\_hasht(){ //initialize hash symbol\_table

int i = 0;

for (i = 0; i < max; i++)

{

hasht[i].head = NULL;

hasht[i].tail = NULL;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\* S Y M B O L T A B L E \*\*\*\*\*\*\*\*\*\*\*\*\*//

typedef struct symtab\_node {

char label[10];

char address[7];

struct symtab\_node \*next;

}symtab\_node;

typedef struct symtab\_linkedList { //stores a list at each index

struct symtab\_node \*head;

struct symtab\_node \*tail;

}symtab\_linkedList;

symtab\_linkedList \*symbol\_table = NULL;

int symtab\_max = 501;

int symtab\_hashFunction(char\* label){ //returns index where we'll store item

unsigned int wordVal = 0;

for(int i = 0; i < strlen(label); i++){

wordVal = 37 \* wordVal + (label[i] - 48); //convert opcode -> number(key)

}

return wordVal % symtab\_max;

}

//prototypes

void build\_symtab();

void symtab\_insert(char\* lbl, char\* addr);

void symtab\_display();

//definitions

void symtab\_insert(char\* lbl, char\* addr) {

int index = symtab\_hashFunction(lbl);

symtab\_node \*list = symbol\_table[index].head;

symtab\_node \*item = malloc(sizeof(struct symtab\_node)); //item to be added

strcpy(item->label,lbl);

strcpy(item->address,addr);

item->next = NULL;

if (list == NULL) { //linked list is empty

symbol\_table[index].head = item;

symbol\_table[index].tail = item;

}

else { //add to end of list

symbol\_table[index].tail->next = item;

symbol\_table[index].tail = item;

}

}

void build\_symtab(){ //initialize hash symbol\_table

int i = 0;

for (i = 0; i < symtab\_max; i++) {

symbol\_table[i].head = NULL;

symbol\_table[i].tail = NULL;

}

}

int lookup\_symbol(char\* str){

int index = symtab\_hashFunction(str);

symtab\_node \*temp = symbol\_table[index].head;

while(temp!=NULL){

if(strcmp(str,temp->label) == 0){

return 0;

}

temp = temp->next;

}

return 1;

}

char\* address\_val(char\* str){

int index = symtab\_hashFunction(str);

symtab\_node \*temp = symbol\_table[index].head;

while(temp!=NULL){

if(strcmp(str,temp->label) == 0){

return temp->address;

}

temp = temp->next;

}

return "nop";

}

void symtab\_to\_file(){

FILE \* file\_ptr;

if((file\_ptr = fopen("symtab\_file.txt","w")) == NULL){

printf("can't open intermediate file\n"); //file can't be opened

}else{

fprintf(file\_ptr,"%s | %s \n\n","LABEL", "ADDRESS"); //<---write to file

for (int i = 0; i < symtab\_max; i++) {

symtab\_node \*temp = symbol\_table[i].head;

if (temp != NULL) {

while (temp != NULL) {

fprintf(file\_ptr,"%s \t %s \n",temp->label, temp->address); //<---write to file

temp = temp->next;

}

}

}

}

fclose(file\_ptr);

}