과목 명: 시스템프로그래밍

담당 교수 명: 박 운 상

<<Assignment 3>>

**서강대학교 컴퓨터학과**

**[20141603]**

**[황 순]**

목 차

1. 프로그램 개요 3

2. 프로그램 설명 3

2.1 프로그램 흐름도 3

3. 모듈 정의 3

3.1 모듈 이름 : finderrorinobj(char \*filename) 3

3.1.1 기능 3

3.1.2 사용 변수 3

3.2 모듈 이름: loaderone(char \*filename) 3

3.2.1 기능 3

3.2.2 사용 변수 3

3.3 모듈이름: findref(char \*str,int \*value,char plus) 4

3.3.1 기능 4

3.3.2 사용변수 4

3.4 모듈이름: loadertwo(char \*filename,int dic) 4

3.4.1 기능 4

3.4.2 사용변수 4

3.5 모듈이름: printload() 4

3.5.1 기능 4

3.5.2 사용변수 4

3.6 모듈이름: makebreakpoint(int bpval) 4

3.5.1 기능 4

3.5.2 사용변수 4

3.7 모듈이름: printbreakpoint() 4

3.5.1 기능 4

3.5.2 사용변수 4

3.9 모듈이름: clearbreakpoint() 4

3.5.1 기능 4

3.5.2 사용변수 4

4. 전역 변수 정의 4

4.1 int progaddr; 4

4.2 int csaddr; 4

4.3 int csnum = 0; 4

4.4 int epaddr; 4

4.5 int bpflag; 4

4.6 cstab \*C,\*cstmp; 4

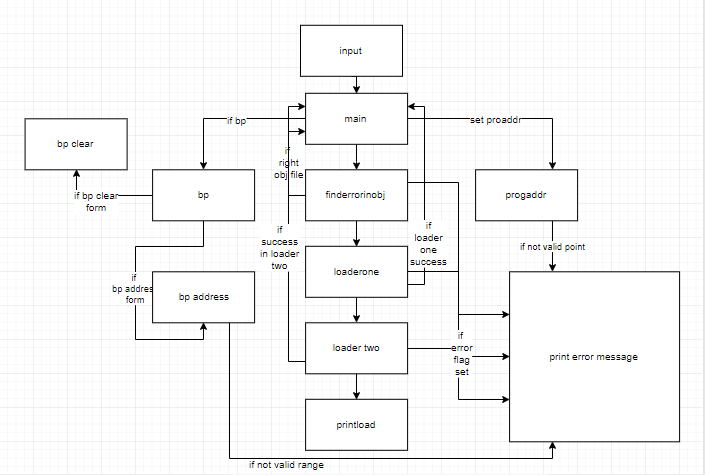
4.7 bpbox \*br; 5

# 프로그램 개요

프로젝트 #1, #2 에서 구현한 셀(shell)에 linking과 loading 기능을 추가하는 프로그램입니다. 프로젝트 #2 에서 구현된 assemble 명령을 통해서 생성된 object 파일을 link시켜 메모리에 올리는 일을 수행합니다.

# 프로그램 설명

## 프로그램 흐름도



# 모듈 정의

## 모듈 이름 : int finderrorinobj(char \*filename)

### 기능

알맞은 object file 인지를 검사한다.

### 사용 변수

int len; //길이 변수

int i; // 루프 변수

int comma=0; // comma 수

## 모듈 이름: int loaderone(char \*filename)

### 기능

로더 패스1을 수행한다.. H,R,D,E 레코드를 처리한다. Cstab을 완성한다.

### 사용 변수

FILE \*fp; //file pointer to open object file

cstab \*t; // control section tmp variable

char fname[20]; // variable for filename

char text[200]; // variable for string from the file

int CSLTH; // length variable in pass1(algorithm)

int len; // text length

int i; // loop variable

int j = 0; // variable for D array (D text)

int k; // loop variable

int Dseclen; // length of Dsection

int csstart; // start address of program

int flag=0; // flag variable for return(if succeed return 0 , if not return -1)

int ridx; // index variable

int idx; // index variable

## int findref(char \*str,int \*value,char plus)

### 기능

.str과 같은 symbol이 각 cstab의 D에 존재하는지 찾아준다.

### 사용변수

cstab \*tmp; // temp variable for the search

int i; // loop variable

int flag = 0; // flag variable

## 모듈이름: int loadertwo(char \*filename,int dic)

### 기능

로더 패스 2를 수행한다. T,M 레코드와 relocation, 메모리에 실제 배치를 수행한다.

### 사용변수

FILE \*fp; // file pointer for open object file

cstab \*t; // tmp variable for search or put info in the control section C

int i; // loop variable

int add; // address variable

int val; // value varialbe

int len; // lenght variable for modification record

int modlen; //length of modification record

int value; //variable for receive address from the D table

int memval=0; // full hexadecimal value for use in the modification(relocation)

int flag = 0; // flag variable

int sflag = 0; // search flag (if find the same ref name )

int vflag = 0; // flag if found same idx

char text[200]; // string variable for file; receive record

char ref[10]; // string variable for receive the R name

char mem[10]; //string variable for converting hexadecimal

## 모듈이름: void printload()

### 기능

Loader map을 출력한다.

### 사용변수

cstab \*tmp; // tmp variable for the search and print

int i; // loop variable

tmp = C;

## 모듈이름: void makebreakpoint(int bpval)

### 기능

Breakpoint를 설정한다. 프로그램이 존재하는 곳에서만 설정 가능하다.

### 사용변수

bp \*tmp,\*t; // bp variable for put it in the bpbox.

## 모듈이름 void printbreakpoint()

### 기능

존재하는 break point 들을 출력해준다.

### 사용변수

bp \*t; // variable for print

## 모듈이름: void clearbreakpoint()

### 기능

존재하는 breakpoint들을 없앤다.

### 사용변수

없음.

# 전역 변수 정의

## int progaddr;

프로그램의 시작위치를 지정한다..

## int csaddr;

control section 의 시작위치를 지정한다..

## int csnum = 0

프로그램에서 object file의 개수를 알려준다. 이를 통해 pass에서 알맞은 테이블을 찾아간다..

.

## int epaddr;

프로그램의 종료 위치를 나타낸다.

## cstab \*C,\*cstmp;

Control section table로 C는 전체 테이블을 의미하고 cstmp는 C에 새로운 테이블을 채울 때 사용할 변수이다.

## bpbox \*br;

breakpoint 들을 저장할 변수이다.

# 코드

20141603.h

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <ctype.h>

#include<sys/stat.h>

#include<dirent.h>

typedef struct Break\_Point {

int address;

struct Break\_Point \*next;

}bp;

typedef struct BP\_BOX {

int num;

bp \*bpoint;

}bpbox;

typedef struct Reference\_table

{

char refname[10];

int refnum;

int address;

}reftab;

typedef struct External\_symbol\_table

{

int address;

char exname[10];

}extab;

typedef struct ControlSection\_Table{

char csname[20];

int length;

int address;

int Dnum;

int Rnum;

extab \*D;

reftab \*R;

struct ControlSection\_Table \*next;

}cstab;

typedef struct symbol\_table {

char symbol[10];

int locc;

struct symbol\_table\* link;

}symtab;

typedef struct node{

int cnt;

char name[90];

struct node \*link;

}Node;

typedef struct hash // hash table 공간

{

char mneo[100];

int value;

char name[100];

struct hash \*next;

}hashtable;

char\* rtrim(char\* s);

char\* ltrim(char \*s);

char\* trim(char \*s);

void help();

int FindValidDumpInput(char \*str);

void dir();

void MakeHistory(Node \*\*hs, char \*str);

void PrintHistory(Node \*hs, int cnt);

int Dump(unsigned char \*\*Memory, char \*str, int spoint, int end);

void EditMemory(unsigned char \*\*Memory, int address, int value);

void fill(unsigned char \*\*Memory, int startA, int endA, int value);

int hash\_function(char \*s);

void OpcodeInput(hashtable \*\*ht);

void Opcodelist\_print(hashtable\*\* ht);

int FindOpPrint(char \*command,hashtable \*\*ht);

int findrighthex(char \*str);

int finddirect (char \*str);

int findopcode (char \*str, hashtable \*\*temp, hashtable \*ht[20]);

int findregi (char \*str);

int findsymbol (char \*str, symtab \*\*insisymhead);

int findcomma (char \*str1, char \*str2, char \*str3, char \*str);

int xregiuse (char \*str1, char \*str2, char \*str3);

int Makeobject (char \*command, int \*length, int \*bv, hashtable \*ht[20]);

void fillstringzero(char \*str);

int FinishObject (char \*fn, int \*bv, int \*length, hashtable \*ht[20]);

int PrintSymtab();

int PrintType(char\* Input\_change);

int finderrorinobj(char \*filename);

int loaderone(char \*filename);

int findref(char \*str, int \*value, char plus);

int loadertwo(char \*filename, int dic);

void printload();

void makebreakpoint(int bpval);

void printbreakpoint();

void clearbreakpoint();

20141603.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <sys/stat.h>

#include "20141603.h"

unsigned char \*\*Memory; //Memory

symtab \*st[100];

int sloc = 0; //start location

int lines[1000]; // array for line

int loc[1000]; //array for location

char filename\_lst[50];//filename.lst

char filename\_obj[50];//filename.obj

int progaddr; // beginning address in memory

int csaddr; // starting address assigned to control section currently being scaned by the loader

int csnum = 0;

int epaddr;

int bpflag;

cstab \*C,\*cstmp;

bpbox \*br;

char\* rtrim(char\* s) {

char t[90];

char \*end;

strcpy(t, s);

end = t + strlen(t) - 1;

while (end != t && isspace(\*end))

end--;

\*(end + 1) = '\0';

s = t;

return s;

}

//terminate blank in the right

char\* ltrim(char \*s) {

char\* begin;

begin = s;

while (\*begin != '\0') {

if (isspace(\*begin))

begin++;

else {

s = begin;

break;

}

}

return s;

}

// terminate blank in the left

char\* trim(char \*s) {

return rtrim(ltrim(s));

}

// terminate blank in the left and right

void help()

{

printf("h[elp]\n");

printf("d[ir]\n");

printf("q[uit]\n");

printf("hi[story]\n");

printf("du[mp] [start , end]\n");

printf("e[dit] address, value\n");

printf("f[ill] start , end, value\n");

printf("reset\n");

printf("opcode mnemonic\n");

printf("opcodelist\n");

printf("assemble filename\n");

printf("type filename\n");

printf("symbol\n");

printf("progaddr\n");

printf("loader\n");

printf("bp\n");

}

// function for printing the available commands

int FindValidDumpInput(char \*str)

{

int i;

int dflag = 0; // flag for the finding valid input

int len;

int lens, lene,leni;

int bflag = 0; // flag for the blank between the hexadecimal numbers

char start[30];

char end[30];

char imm[30];

char junk[30];

// strings for sscanf

sscanf(str, "%s %[^\n]", junk, start);

strcpy(start, trim(start));

len = strlen(start);

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

{

if (start[i] == ',')

dflag = 3;

else

dflag = 2;

}

// if there is ',' , dflag =3 , if not dflag = 2

if (dflag == 2)

{

lens = strlen(start);

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

{

if ((start[i] <= '9'&&start[i] >= '0') || (start[i] <= 'F'&&start[i] >= 'A') || (start[i] <= 'f'&&start[i] >= 'a'))

continue; // if the character is number or alphabet a to f, A to F

else if (start[i] == ' ' || start[i] == '\t')

{

bflag = -1;

break;

} //if there is blank between the hexadeciaml numbers like 2 3, make the bflag = -1

else

{

if (start[i] == ',')

{

dflag = 3;

break;

}

else

bflag = -1;

break;

}

}

if (bflag == -1)

{

return -1;

}

}

// for the dump start and case that dump start, which is not valid Input

if (dflag == 3)

{

sscanf(str, "%s %[^,] %\*c %[^\n]", junk, start, end);

lens = strlen(start);

lene = strlen(end);

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

{

if ((start[i] <= '9'&&start[i] >= '0') || (start[i] <= 'F'&&start[i] >= 'A') || (start[i] <= 'f'&&start[i] >= 'a'))

continue; // if the character is number or alphabet a to f, A to F

else if (start[i] == ' ' || start[i] == '\t')

{

bflag = -1;

break;

} //if there is blank between the hexadeciaml numbers like 2 3, make the bflag = -1

else

{

bflag = -1;

break;

}

}

if (bflag == -1)

{

return -1;

}

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

{

if ((end[i] <= '9'&&end[i] >= '0') || (end[i] <= 'F'&&end[i] >= 'A')|| (end[i] <= 'f'&&end[i] >= 'a'))

continue; // if the character is number or alphabet a to f, A to F

else if (end[i] == ' ' || end[i] == '\t')

{

bflag = -1;

break;

} //if there is blank between the hexadeciaml numbers like 2 3, make the bflag = -1

else

{

if (end[i] == ',')

{

dflag = 4; // in case of fill

break;

}

bflag = -1;

break;

}

}

if (bflag == -1)

{

return -1;

}

} //for the dump start,end and edit

if (dflag == 4)

{

sscanf(str, "%s %[^,] %\*c %[^,] %\*c %[^\n]", junk, start, end, imm);

lens = strlen(start);

lene = strlen(end);

leni = strlen(imm);

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

{

if ((start[i] <= '9'&&start[i] >= '0') || (start[i] <= 'F'&&start[i] >= 'A')||(start[i]<='f'&&start[i]>='a'))

continue; // if the character is number or alphabet a to f, A to F

else if (start[i] == ' ' || start[i] == '\t')

{

bflag = -1;

break;

} //if there is blank between the hexadeciaml numbers like 2 3, make the bflag = -1

else

{

bflag = -1;

break;

}

}

if (bflag == -1)

{

return -1;

}

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

{

if ((end[i] <= '9'&&end[i] >= '0') || (end[i] <= 'F'&&end[i] >= 'A')||(end[i]<='f'&&end[i]>='a'))

continue; // if the character is number or alphabet a to f, A to F

else if (end[i] == ' ' || end[i] == '\t')

{

bflag = -1;

break;

} //if there is blank between the hexadeciaml numbers like 2 3, make the bflag = -1

else

{

bflag = -1;

break;

}

}

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

{

if ((imm[i] <= '9'&&imm[i] >= '0') || (imm[i] <= 'F'&&imm[i] >= 'A')||(imm[i]<='f'&&imm[i]>='a'))

continue; // if the character is number or alphabet a to f, A to F

else if (imm[i] == ' ' || imm[i] == '\t')

{

bflag = -1;

break;

} //if there is blank between the hexadeciaml numbers like 2 3, make the bflag = -1

else

{

bflag = -1;

break;

}

}

if (bflag == -1)

{

return -1;

}

else

return 4;

}

else if (dflag == 2)

return 2;

else if (dflag == 3)

return 3;

else

return -1;

}

// find the valid input and if it is valid, return proper flag . if not, return -1

void dir()

{

DIR \*srcdir = opendir(".");

int result = 0;

struct dirent \*items;

struct stat file\_info;

if(srcdir)

{

while((items = readdir(srcdir))!=NULL)

{

if(strcmp(items->d\_name, ".") == 0 || strcmp(items->d\_name, "..")==0)

continue;

lstat(items->d\_name, &file\_info);

printf("\t\t%s", items->d\_name);

result++;

if(S\_ISDIR(file\_info.st\_mode)) printf("/");

else if(S\_IEXEC & file\_info.st\_mode) printf("\*");

if(result%3==0) printf("\n");

}

printf("\n");

}

closedir(srcdir);

}

// function for printing existing file in the directory

void MakeHistory(Node \*\*hs, char \*str)

{

Node \*temp;

temp = (Node\*)malloc(sizeof(Node));

strcpy(temp->name, str);

temp->link = NULL;

Node\* tmp;

if ((\*hs) == NULL)

(\*hs) = temp;

// if history node is empty, make the temp node directly link to it

else {

for (tmp = (\*hs); tmp->link != NULL; tmp = tmp->link);

tmp->link = temp;

}

// if there is any existing node, put temp node at the end

}

//function for make the array of the effective command by using linked list

void PrintHistory(Node \*hs, int cnt)

{

Node \*temp;

temp = (Node\*)malloc(sizeof(Node));

for (temp = hs->link; temp != NULL; temp = temp->link) {

printf("%d : %s\n", cnt, temp->name);

cnt++;

}

free(temp);

}

// function for printing the commands that is in the history node

int Dump(unsigned char \*\*Memory,char \*str,int spoint,int end)

{

int memv; // variable to show memory address

int i, j;

int srow, erow, scol,ecol; //variables for the row and col number of the startpoint and the endpoint

int epoint;

int enddot;

int startid = 0;

if (spoint < 0 || spoint > 1048575)

{

printf("This is not available range!\n");

return -1;

}

// if the startpoint is not in the available range, print the error message

else

{

if (end == -1)

{

epoint = spoint + 159;

} // in case of dump, dump %x

else

epoint = end; // in case of dump %x,%x

if (end > 1048575) {

printf("This is not available range!\n");

return -1;

}

// if the endpoint is bigger than existing, print error message

else {

startid = epoint + 1; // next start will be epoint+1

srow = spoint / 16;

scol = spoint % 16;

if (epoint > 1048575)

epoint = 1048575;

erow = epoint / 16;

ecol = epoint % 16;

// find the col and row of the startpoint, also endpoint

memv = srow \* 16;

for (i = srow; i < erow + 1; i++)

{

if (i == srow)

{

if (srow == erow)

enddot = ecol; // if the srow and erow are the same, enddot will be ecol

else

enddot = 15; // if srow!=erow , enddot should be the end of the line

printf("%05X ", memv);

memv += 16;

for (j = 0; j < scol; j++)

printf(" ");

for (j = scol; j < enddot + 1; j++)

{

printf("%02X ", Memory[i][j]);

}

printf(" ;");

for (j = 0; j < scol; j++)

printf(".");

for (j = scol; j < enddot+1; j++)

{

if (32 <= Memory[i][j] && Memory[i][j] <= 126)

printf("%c", Memory[i][j]);

else

printf(".");

} // if it is in the range, print the character if it is in available range

for (j = enddot + 1; j < 16;j++)

printf(".");

// if it is not in the range , print it '.'

}

else if (i == erow)

{

printf("%05X ", memv);

memv += 16;

for (j = 0; j < ecol + 1; j++)

{

printf("%02X ", Memory[i][j]);

}

for (j = ecol + 1; j < 16; j++)

printf(" ");

printf(" ;");

for (j = 0; j < ecol + 1; j++)

{

if (32 <= Memory[i][j] && Memory[i][j] <= 126)

printf("%c", Memory[i][j]);

else

printf(".");

} // if it is in the range, print the character if it is in available range

for (j = ecol + 1; j < 16; j++)

printf("."); // if it is not in the range , print it '.'

}

else

{

printf("%05X ", memv);

memv += 16;

for (j = 0; j < 16; j++)

{

printf("%02X ", Memory[i][j]);

}

printf(" ;");

for (j = 0; j < 16; j++)

{

if (32 <= Memory[i][j] && Memory[i][j] <= 126)

printf("%c", Memory[i][j]);

else

printf(".");

}

} // if it is in the range, print the character if it is in available range

printf("\n");

}

}

return startid;

}

}

// function that is used in the case of dump, dump %x, and dump %x,%x.

// use range to print appropriate characters

// print error message if startpoint or the endpoint is not in the available range.

void EditMemory(unsigned char \*\*Memory, int address, int value)

{

int Tarrow, Tarcol; //Target address's row and col

if (value < 0 || value > 255)

{

printf("This value is not in available range!\n");

return;

}

if (address < 0 || address > 1048575)

{

printf("Address is not in available range!\n");

return;

}

Tarrow= address / 16;

Tarcol = address % 16;

Memory[Tarrow][Tarcol] = value;

}

//function for edit value in the target address

//print error message if the value or the target address is not in the range.

void fill(unsigned char \*\*Memory, int startA, int endA, int value)

{

int srow, scol;

int erow, ecol;

int enddot;

int i, j;

//int tmp, val;

if (startA < 0 || 1048575 < startA)//error input

{

printf("Startpoint is not in available range!\n");

return;

}

if (endA < 0 || 1048575 < endA)

{

printf("Endpoint is not in available range!\n");

return;

}

if (endA < startA)

{

printf("Startpoint should be smaller than the Endpoint!\n");

return;

}

if (value < 0 || 255 < value)

{

printf("Value is not in available range!\n");

return;

}

srow = startA / 16;

scol = startA % 16;

erow = endA / 16;

ecol = endA % 16;

// find the row and the col of the startpoint and the endpoint

for (i = srow; i < erow + 1; i++)

{

if (i == srow)

{

if (srow == erow)

enddot = ecol; // if srow==erow , enddot should be the ecol

else

enddot = 15; // if not, we should print the end of the line so enddot is 15

for (j = scol; j < enddot + 1; j++)

{

Memory[i][j] = value;

}

// fill the value in the memory

}

else if (i == erow)

{

for (j = 0; j < ecol + 1; j++)

{

Memory[i][j] = value;

}

// fill the value in the memory

}

else {

for(j=0;j<16;j++)

Memory[i][j] = value;

} //fill the value in the memory

}

}

//function for fill value in the given range of the address

//print error message if value or startpoint or endpoint is not in available address

//print error message if startpoint is bigger than the endpoint

int hash\_function(char \*s)

{

int i, sum;

for (i = 0, sum = 0; i <strlen(s); i++)

{

sum += s[i];

}

return sum % 20;

}

//function for the hash

void OpcodeInput(hashtable \*\*ht)

{

FILE \*fp = fopen("opcode.txt", "r");

char tmpmneo[20];

char tmpname[20];

char tmpvalue[20];

hashtable \*temp, \*head;

int i;

int idx;

if (fp == NULL)

{

printf("There is no opcode.txt!!\n");

return;

}

// Print Error message if there is no such opcode.

else

{

for (i = 0; i < 20; i++)

{

ht[i] = NULL;

}

//make the head of the hashtable NULL

while (fscanf(fp,"%s %s %s",tmpvalue,tmpmneo,tmpname)!=EOF) // do fscanf until there is nothing.

{

temp = (hashtable\*)malloc(sizeof(hashtable));

temp->next = NULL;

strcpy(temp->mneo, tmpmneo);

strcpy(temp->name, tmpname);

sscanf(tmpvalue, "%x", &temp->value);

idx = hash\_function(temp->mneo);

//fill the tempnode by the data given by the file

if (ht[idx] == NULL)

{

ht[idx] = temp;

}

// put tempnode at the front of the idx

else

{

head = ht[idx];

do

{

if (head->next == NULL)break;

head = head->next;

} while (1);

head->next = temp;

}

// if there exist node int the front, put the newnode(temp) in the back.

}

}

fclose(fp);

}

//function for make the opcode hash table by using linked list

//put the hash struct in the appropriate position

void Opcodelist\_print(hashtable\*\* ht)

{

hashtable\* temp;//pointer for searching hash table

int i;//index

temp = (hashtable\*)malloc(sizeof(hashtable));

for (i = 0; i<20; i++)//printing the hash table

{

printf("%d : ", i);

for (temp = ht[i]; temp != NULL; temp = temp->next)

{

printf("[%s, %X]", temp->mneo, temp->value);

if (temp->next != NULL)

printf(" -> ");

}

printf("\n");

}

}

//function for printing the existing opcodes in the hash table

int FindOpPrint(char\* command, hashtable\*\* ht)

{

int index;//index of hash table

char tmpname[100];//string of inputing opcode

char tmpmeno[100];//string of inputing mnemonic

int flag = 0;

hashtable\* temp;//pointer for searching hash table

index = 0;

sscanf(command, "%s%s", tmpname, tmpmeno);

index = hash\_function(tmpmeno);

temp = ht[index];

for (temp = ht[index]; temp != NULL; temp = temp->next)

{

if (strcmp(temp->mneo, tmpmeno) == 0)

{

flag = 1;

break;

}

}

// search in the hash if there is same name of integer

if (flag == 0)

{

return 0;

}

// if there is no such name, return 0 for the failure

else {

printf("opcode is %X\n", temp->value);

return 1;

}

// if there is such name, return 1 for the success

}

//function that finds the opcode from the input command

// if there is same thing in hash table, print. if not, return 0

int findrighthex (char \*str)

{

int len;

int i;

int flag = 0;

len = strlen (str);

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

{

if ((str[i] <= '9'&&str[i] >= '0') || (str[i] <= 'F'&&str[i] >= 'A') || (str[i] <= 'f'&&str[i] >= 'a'))

continue;

else

flag = -1;

}

return flag;

}

// if it is right equation, return 0. if not return -1

int finddirect(char \*str)

{

int flag = 0;

if (!strcmp(str, "BYTE"))

flag = 1;

else if (!strcmp(str, "BASE"))

flag = 2;

else if (!strcmp(str, "RESB"))

flag = 3;

else if (!strcmp(str, "RESW"))

flag = 4;

else if (!strcmp(str, "WORD"))

flag = 5;

else if (!strcmp(str, "END"))

flag = 6;

else if (!strcmp(str, "START"))

flag = 7;

return flag;

}

//if it finds the directive, return the number of directive. if not return 0

int findopcode(char \*str, hashtable \*\*temp,hashtable \*ht[20])

{

int index;

int flag=0;

index = hash\_function(str);

\*temp = ht[index];

while(1)

{

if ((\*temp) == NULL)break;

if (strcmp((\*temp)->mneo, str) == 0) {

flag = 1;

break;

// if find opcode, flag =1 and break

}

(\*temp) = (\*temp)->next;

}

return flag;

}

//search the whole table if there is opcode that is same.

int findregi(char \*str)

{

int flag = -1;

if (strcmp (str, "A") == 0)

flag = 0;

else if (strcmp (str, "L") == 0)

flag = 2;

else if (strcmp(str, "X") == 0)

flag = 1;

else if (strcmp (str, "PC") == 0)

flag = 8;

else if (strcmp (str, "SW") == 0)

flag = 9;

else if (strcmp (str, "B") == 0)

flag = 3;

else if (strcmp (str, "T") == 0)

flag = 5;

else if (strcmp(str, "S") == 0)

flag = 4;

else if (strcmp (str, "F") == 0)

flag = 6;

return flag;

}

//find the name of register

int findsymbol (char \*str, symtab \*\*tmpsys)

{

int index;

int flag=0;

index = hash\_function (str);

\*tmpsys = st[index];

for (;;)

{

if ((\*tmpsys) == NULL)break;

if (strcmp((\*tmpsys)->symbol, str) == 0) {

flag = 1;

break;

}// if find the symbol, flag =1

(\*tmpsys) = (\*tmpsys)->link;

}

return flag;

}

// search the whole table if there is symbol

int findcomma(char \*str1, char \*str2, char \*str3, char \*str)

{

int i=0;

int comflag;

int flag1, flag2; //store the first register and second register

int len;

int eflag = 0, mflag = 0;

char tmp[20];

char tmp2[20]; // tmp string for storing the string that is divided by comma

len = strlen(str1);

comflag = 0, flag1 = 0, flag2 = 0;

if (str1[len - 1] == ',')

{

str1[len - 1] = 0;

comflag = 1;

} // check if there is comma

else

{

do

{

if (str1[i++] == ',')

{

mflag = 1;

break;

}

} while (i < len - 1); //if it is in the str

if (mflag == 1 && str2 == NULL)

{

sscanf(str1, "%[^,] %\*c %[^,]", tmp, tmp2);

flag1 = findregi(tmp);

flag2 = findregi(tmp2); // if there is comma, find the mnenumic number

}

}

if (mflag == 0)

flag1 = findregi(str1);

if (str2 == NULL && comflag == 1)

eflag = 1;

else if (mflag == 0 && str2 != NULL)

{

if (str2[0] == ',')

{

comflag = 1;

flag2 = findregi(str2 + 1);

if (str3 != NULL)

eflag = 1;

}

else if (strcmp (str2, ",") == 0)

{

comflag = 1;

if (str3 != NULL)

flag2 = findregi(str3);

else eflag = 1;

}

else if (comflag == 1)

{

if (str3 == NULL)

flag2 = findregi(str2);

else eflag = 1;

}

else eflag = 1;

}

if (strcmp(str, "CLEAR") != 0 && strcmp(str, "TIXR") != 0 && comflag == 0)

return -1; // because clear and tixr have one register

else if (eflag == 1)

return -1;

else

return (flag1 \* 10 + flag2);

}

//function for checking comma and format 2

int xregiuse(char \*str1, char \*str2, char \*str3)

{

int comflag = 0; // flag for comma

int len = 0; //variable for length

int flag = 0; //flag for using x register or erro

int eflag = 0;

len = strlen (str1);

if (str1[len - 1] == ',')

{

str1[len - 1] = '\0';

comflag = 1;

}

if(len >= 2 && ((str1[len-2]==','&&str1[len-1]=='x') || (str1[len-2]==','&&str1[len-2]=='X')) )

{

str1[len - 1] = '\0';

str1[len - 2] = '\0';

if (str2 != NULL)

eflag = -1;

else

flag = 1;

} // if there is x

else

{

if (str2 != NULL)

{

if (strcmp(str2, ",x") == 0 || strcmp(str2, ",X") == 0) // if there is x

flag = 1;

else if (strcmp (str2, ",") == 0 && str3!= NULL)

{

if(str3[0]=='x'||str3[0]=='X')

flag = 1;

else

eflag = -1;

}

else if (comflag == 1&&(str2[0]=='x'||str2[0]=='X'))

{

flag = 1;

}

else

eflag = -1;

}

}

if (comflag == 1 && flag == 0)

eflag = -1;

if (eflag == -1)

flag = -1;

return flag;

}

//check if the x register is used

int Makeobject(char \*command,int \*length,int \*base,hashtable \*ht[20])

{

FILE \*fp; // file pointer for opening the .asm file

int len; // variable for length

int eflag=0; // variable for error flag

int i; // variable for loop

int idx; //index

int line; // variable for lin

int plus = 0; //variable for flag if there is +

int value; // store the value

int endflag = 0; // flag for finding the END

int baseflag = 0; // flag for if there is base

int dflag = 0; // flag for the directive

int nflag; // flag for comma or x check

int findflag = 0; // flag for noticing if function finds the symbol or opcode

int sflag; // flag for noticing if the string has the right hexadecimal word form

int index;

int location = 0;

char br[100]; //variable for used when base register is using

char filename[50]; // variable for filename

char junk[50]; // string for put junk

char temp[100];

char \*p1, \*p2, \*p3, \*p4, \*p5; // pointer for saving tokenized strings

hashtable \*tmphash; // temporay variable to search hashtable

symtab \*newsym, \*tmpsym; // temporary variables for put the element in the symtab or search the symtab

idx = 0;

line = 5;

sscanf(command, "%s%s", junk, filename); //tokenize to get filename

len = strlen(filename);

if (len<5 || filename[len - 1] != 'm' || filename[len - 2] != 's' || filename[len - 3] != 'a' || filename[len - 4] != '.')

{

printf("This is not assemcode!\n");

return -1;

}

// error message

for (i = 0; i<len - 4; i++)//before ".asm" '.'character aprrears

{

if (filename[i] == '.')

{

printf("This is not assemcode!\n");

return -1;

}

}

strcpy(filename\_lst, filename);

strtok(filename\_lst, ".");

strcat(filename\_lst, ".lst");

strcpy(filename\_obj, filename);

strtok(filename\_obj, ".");

strcat(filename\_obj, ".obj"); //Make filenames about list file and object file

fp = fopen(filename, "rt"); //file open for read asm file

if (fp == NULL)

{

printf("No input file!\n"); // if there is no .asm file, print error

return -1;

}

else {

for (i = 0; i < 100; i++)

st[i] = NULL;

fgets(temp, sizeof(temp), fp);

lines[idx] = line;

loc[idx] = location;

idx++;

line = 5;

len = strlen(temp);

if (temp[len - 1] == '\n')

temp[len - 1] = '\0';

p1 = strtok(temp, " \t");

p2 = strtok(NULL, " \t");

p3 = strtok(NULL, " \t");

if (p1 != NULL && p2 != NULL && p3 != NULL && strcmp (p2, "START") == 0) // if the string is not empty and find START

{

sscanf (p3, "%x", &(\*length));

sloc = (\*length);

location = (\*length);

do

{

line = line + 5;

if (feof(fp))

break;

fgets(temp, sizeof(temp), fp);

len = strlen (temp);

if (temp[len - 1] == '\n')

temp[len - 1] = 0;

p1 = NULL, p2 = NULL, p3 = NULL, p4 =NULL, p5 = NULL;

p1 = strtok (temp, " \t");

p2 = strtok (NULL, " \t");

p3 = strtok (NULL, " \t");

p4 = strtok (NULL, " \t");

p5 = strtok (NULL, " \t"); // tokenize the string

if (p1 == NULL && endflag == 1) // if find end

break;

if (p1 != NULL)

{

lines[idx] = line; // store the information of line

loc[idx] = location; // store the information of loc

idx++;

if (endflag == 1)

eflag = 2;

else if (p1[0] != '.')

{

tmphash = NULL;

plus = 0;

if (p1[0] == '+')

plus = 1;

findflag = findopcode (p1 + plus, &tmphash, ht);

dflag = finddirect (p1);

if (tmphash == NULL && dflag == 0) // if it is not directive, search if it symbol

{

newsym = (symtab\*)malloc (sizeof (symtab));

newsym->link = NULL;

newsym->locc = location;

strcpy (newsym->symbol, p1);

index = hash\_function (p1);

if (st[index] == NULL)

st[index] = newsym;

else

{

tmpsym = st[index];

do

{

if (tmpsym->link == NULL)break;

tmpsym = tmpsym->link;

} while (1);

tmpsym->link = newsym;

}

}

else if (tmphash != NULL || dflag != 0)

{

p5 = p4;

p4 = p3;

p3 = p2;

p2 = p1;

}

tmphash = NULL;

plus = 0;

if (p2 != NULL)

{

if (p2[0] == '+') //format 4

plus = 1;

findflag = findopcode (p2 + plus, &tmphash, ht);

dflag = finddirect (p2);

}

if (findflag == 0 && dflag != 0) // if we find directive

{

if (dflag == 1&& p3!=NULL) // BYTE

{

len = strlen (p3);

if (len >= 3) {

if (p3[1] == (char)(39) && p3[len - 1] == (char)(39)) // if we find '

{

if (p3[0] == 'X' || p3[0] == 'x' )//|| p3[0] == 'C' || p3[0] == 'c')

{

if ((len - 3) % 2 == 0)

location = location + ((len - 3) / 2);

else

eflag = 2;

}

if (p3[0] == 'C' || p3[0] == 'c')

location = location + (len - 3);

}

}

}

else if (dflag == 2) //BASE

{

if (p3 != NULL)

{

strcpy(br, p3);

baseflag = 1;

}

}

else if (dflag == 3) //RESB

{

if (p3 == NULL)

eflag = 2;

else

{

sflag = findrighthex(p3);

if (sflag == 0) {

sscanf(p3, "%d", &value);

location = location + value;

}

else

eflag = 2;

}

}

else if (dflag == 4) //RESW

{

if (p3 != NULL)

{

sflag = findrighthex (p3);

if (sflag == 0) {

sscanf (p3, "%d", &value);

location = location + (value \* 3);

}

else

eflag = 2;

}

}

else if (dflag == 5) //WORD

{

if (p3 != NULL)

{

sflag = findrighthex(p3);

if (sflag == 0) {

sscanf(p3, "%d", &value);

location = location + 3;

}

else

eflag = 2;

}

else

eflag = 2;

}

else if (dflag == 6) //END

endflag = 1;

else eflag = 2;

}

else if (findflag != 0) // opcode

{

if (plus == 0) {

if (tmphash->name[0] == '1') //format 1

location = location + 1;

else if (tmphash->name[0] == '2') //format 2

{

if (p3 != NULL) {

nflag = findcomma(p3, p4, p5, tmphash->mneo);

if (nflag == -1)

eflag = 2;

else

location = location + 2;

}

}

else if (tmphash->name[0] == '3') // format 3

{

if (strcmp(tmphash->mneo, "RSUB") != 0)

{

if (p3 != NULL) {

nflag = xregiuse(p3, p4, p5);

if (nflag == -1)

eflag = 2;

else

location = location + 3;

}

}

else

{

if (p3 == NULL)

location += 3;

}

}

}

else if (plus == 1) // format 4

{

if (tmphash->name[2] == '4') {

if (p3 != NULL)

{

if (strcmp(tmphash->mneo, "RSUB") != 0)

{

nflag = xregiuse(p3, p4, p5);

if (nflag == -1)

eflag = 2;

else

location = location + 4;

}

}

if (p3 == NULL)

{

if (strcmp(tmphash->mneo, "RSUB") == 0)

location = location + 4;

}

}

}

else

eflag = 2;

}

else

eflag = 2; // if nothing

}

}

}while (eflag == 0);

}

else

eflag = 2;

fclose(fp);

}

if (baseflag == 1) // if there exist base, store it

{

tmpsym = NULL;

findflag = findsymbol (br, &tmpsym);

if (findflag == 0)

eflag = 1;

else

(\*base) = tmpsym->locc;

}

if (eflag == 2)

printf ("Error in line %d!\n", line);

(\*length) = location - (\*length); //caculate the lenght

return eflag;

}

//function for pass 1

//make the symbol table and the obj

void fillstringzero(char \*str)

{

int i;

int len=strlen(str);

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

{

if (str[i] == 0)

str[i] = '0';

if (str[i] >= 'a'&&str[i] <= 'f')

str[i] = str[i] - 'a' + 'A';

}

}

// fill the 0 in the NULL or change the 'a' to 'A'

int FinishObject(char \*filename, int \*base, int \*length,hashtable \*ht[20])

{

FILE \*fp = fopen(filename, "r");

FILE \*fp1 = fopen(filename\_lst, "w");

FILE \*fp2 = fopen(filename\_obj, "w");

// open the assemble file and lst file, obj file

hashtable \*tmphash;

symtab \*tmpsys; // tmp variables for search

int tmpvalue = 0,tmpvalue2 = 0; // variable for temporary value

int len = 0, leno = 0, lenob = 0; //variable for lenght

int dflag = 0, plus = 0; // variable for directive

int i, oflag = 0; // varialbe for loop and objectflag

int idx=0; // variable for index;

int locvt = 0, location = 0; // variable for location

int eflag = 0; //error flag

int linet = 0, line = 0; // variable for store the information of line

int endflag = 0; //flag for finding END

int findflag=0; // flag for noticing if the search succeed or not

int symflag = 0; // flag for noticing if the search succeed or not

int mod[100]; // array for modification

int midx = 0; // index for modification

int sidx = 0; // index for printing the modification

char \*p1 = NULL, \*p2 = NULL, \*p3 = NULL, \*p4 = NULL, \*p5 = NULL; // for tokenize

char temp[100], temp2[100]; //temp string for input

char numstr[15]; // string that has number in it

char obcode[100], object[200]; // stirng for object codes

len = strlen (filename);

linet = lines[idx];

locvt = loc[idx++]; // store the next information of loc and line

memset(numstr, 0, 15);

memset(obcode, 0, 10);

lenob = 0;

do

{

if (feof(fp))

break;

fgets(temp, sizeof(temp), fp); // if nothing to read, break

obcode[0] = 0;

len = strlen (temp);

if (temp[len - 1] == '\n')

temp[len - 1] = '\0';

strcpy (temp2, temp);

line = linet;

location = locvt;

linet = lines[idx];

locvt = loc[idx++];

//store the information of the line and loc from the array

p1 = strtok (temp, " \t");

p2 = strtok (NULL, " \t");

p3 = strtok (NULL, " \t"); //token the string

if (p1 == NULL && endflag == 1)

break;

if (p1[0] != '.')

{

tmphash = NULL;

plus = 0;

if (p1[0] == '+') // finding format 4

plus = 1;

findflag = findopcode (p1 + plus, &tmphash,ht); //find opcode

dflag = finddirect (p1);

if (findflag !=0 || dflag != 0)

{

p4 = p3;

p3 = p2;

p2 = p1;

}

tmphash = NULL;

plus = 0;

if (p2[0] == '+')

plus = 1; // findinf format 4

findflag = findopcode (p2 + plus, &tmphash,ht);

dflag = finddirect (p2);

if (findflag == 0 && dflag != 0) // if it is directive

{

oflag = 2;

if (dflag == 1) //BYTE

{

len = strlen (p3);

if (p3[0] == 'X' || p3[0] == 'x')

{

leno = 0;

strncpy(obcode + leno, p3 + 2, len - 3);

leno = leno + len -3;

obcode[leno] = 0;

}

else

{

leno = 0;

for (i = 2; i < len - 1; i++)

{

sprintf(numstr, "%02x", (int)(p3[i]));

fillstringzero(numstr);

strncpy(obcode + leno, numstr,2);

leno = leno + 2;

}

obcode[leno] = 0;

}

}

else if (dflag == 3 || dflag == 4)//RESB or RESW

{

obcode[0] = 0;

oflag = 3;

}

else if (dflag == 5) // WORD

{

sscanf (p2, "%d", &tmpvalue);

sprintf(numstr, "%06x", tmpvalue);

fillstringzero(numstr);

strncpy(obcode, numstr, 5);

obcode[6] = 0;

}

else if (dflag == 6) //END

{

obcode[0] = 0;

endflag = 1;

}

else if (dflag == 7)

oflag = 1;

}

else if (findflag != 0)

{

oflag = 2;

tmpvalue = tmphash->value;

if (tmphash->name[0] == '1') //format 1

{

sprintf(numstr, "%02x", tmpvalue);

fillstringzero(numstr);

strncpy(obcode, numstr, 2);

obcode[2] = 0;

}

else if (tmphash->name[0] == '2') //format 2

{

sprintf(numstr, "%02x", tmpvalue);

fillstringzero(numstr);

strncpy(obcode, numstr, 2);

p5 = strtok (NULL, " \t");

tmpvalue = findcomma(p3, p4, p5, p2);

sprintf(numstr, "%01x", tmpvalue /10);

fillstringzero(numstr);

obcode[2] = numstr[0];

sprintf(numstr, "%01x", tmpvalue %10);

fillstringzero(numstr);

obcode[3] = numstr[0];

obcode[4] = 0;

}

else

{

if (p3 != NULL)

{

if (p3[0] == '#') // immediate addressing

tmpvalue = tmpvalue + 1;

else if (p3[0] == '@') // indirect addressing

tmpvalue = tmpvalue + 2;

else

tmpvalue += 3;

}

else // simple addressing

tmpvalue = tmpvalue + 3;

sprintf(numstr, "%02x", tmpvalue);

fillstringzero(numstr);

strncpy(obcode, numstr, 2);

if (strcmp (p2, "RSUB") != 0) // if not RSUB

{

p5 = strtok (NULL, " \t");

tmpsys = NULL;

plus = 0;

if (xregiuse(p3, p4, p5) == 1)

tmpvalue2 = 8;

else

tmpvalue2 = 0;

if (p3[0] == '#' || p3[0] == '@')

plus = 1;

symflag = findsymbol (p3 + plus, &tmpsys);

if (symflag == 0) {

sscanf (p3 + plus, "%d", &tmpvalue);

}

else tmpvalue = tmpsys->locc;

if (p2[0] == '+' && tmpvalue >= 0) // format 4

{

tmpvalue2 += 1;

sprintf(numstr, "%05x", tmpvalue);

fillstringzero(numstr);

strncpy(obcode + 3, numstr, 5);

obcode[8] = 0;

sprintf(numstr, "%x", tmpvalue2);

fillstringzero(numstr);

obcode[2] = numstr[0];

if (tmpsys != NULL)

{

mod[midx++] = location + 1;

} //modfication

}

else if (symflag != 0)

{

tmpvalue = tmpvalue - locvt;

if (tmpvalue >= -2048 && tmpvalue <= 2047) //find if pc relative by the range

{

eflag = 0;

tmpvalue = tmpvalue & 0xfff; // 2's complement

sprintf(numstr, "%03x", tmpvalue);

fillstringzero(numstr);

tmpvalue2 = tmpvalue2 + 2;

}

else //if not, it will be base relative

{

tmpvalue = tmpvalue + locvt;

if (\*base != -1 && (tmpvalue - (\*base) >= 0 && tmpvalue - (\*base) <= 4095))

{

tmpvalue = tmpvalue - (\*base);

sprintf(numstr, "%03x", tmpvalue);

fillstringzero(numstr);

tmpvalue2 = tmpvalue2 + 4;

}

else if (\*base != -1)

tmpvalue = tmpvalue - (\*base);

else

eflag = 2;

}

if (eflag == 0)

{

strncpy(obcode + 3, numstr, 3);

obcode[6] = 0;

}

}

else if (tmpvalue >= 0) // if it is number, not symbol

{

sprintf(numstr, "%03x", tmpvalue);

fillstringzero(numstr);

strncpy(obcode+3, numstr, 3);

obcode[6] = 0;

}

else

eflag = 3;

sprintf(numstr, "%01x", tmpvalue2);

fillstringzero(numstr);

obcode[2] = numstr[0];

}

else // RSUB

{

if (p2[0] == '+') //check format 4

{

memset(obcode+2, '0', 6);

obcode[8] = 0;

}

else

{

memset(obcode + 2, '0', 4);

obcode[6] = 0;

}

}

}

}

else eflag = 3;

}

fprintf (fp1, "\t%d", line); // make .lst file

if (obcode[0] != 0)

{

sprintf(numstr, "%04x", location);

fillstringzero(numstr);

fprintf(fp1, "\t%s", numstr);

}

else

{

fprintf(fp1, "\t\t");

}

fprintf (fp1, "\t%-80s", temp2);

if (obcode[0] != 0)

fprintf (fp1, "\t%s", obcode);

fprintf (fp1, "\n");

if (oflag == 1) // make .obj file

{

fprintf (fp2, "H%-6s", p1);

sprintf(numstr, "%06x", sloc);

fillstringzero(numstr);

fprintf (fp2, "%s", numstr);

sprintf(numstr, "%06x", (\*length));

fillstringzero(numstr);

fprintf (fp2, "%s\n", numstr);

}

else

{

len = strlen (obcode);

if (lenob == 0 && obcode[0] != 0)

{

object[0] = 'T';

sprintf(numstr, "%06x", location);

fillstringzero(numstr);

strncpy(object + 1, numstr, 6);

lenob = 9;

}

if (oflag == 3) // RESW, RESB

{

if (lenob > 0)

{

object[lenob] = 0;

sprintf(numstr, "%02x", ((lenob - 9) / 2));

fillstringzero(numstr);

strncpy(object + 7, numstr, 2);

fprintf (fp2, "%s\n", object);

}

lenob = 0;

}

if (oflag == 2 && lenob + len >= 69 && obcode[0] != 0) // if exceed the lenght

{

object[lenob] = 0;

sprintf(numstr, "%02x", ((lenob - 9) / 2));

fillstringzero(numstr);

strncpy(object + 7, numstr, 2);

fprintf(fp2, "%s\n", object);

object[0] = 'T';

lenob = 0;

sprintf(numstr, "%06x", location);

fillstringzero(numstr);

strncpy(object + 1, numstr, 6);

strncpy(object + 9, obcode, len);

lenob = 9 + len;

}

else if (oflag == 2 && lenob + len < 69 ) // store the object code

{

if (obcode[0] != 0) {

strncpy(object + lenob, obcode, len);

lenob = lenob + len;

}

}

}

}while (eflag==0);

if (lenob != 0)

{

object[lenob] = 0;

sprintf(numstr, "%02x", ((lenob - 9) / 2));

fillstringzero(numstr);

strncpy(object + 7, numstr, 2);

fprintf (fp2, "%s\n", object);

}

if (midx != 0)

{

while (1)

{

if (sidx == midx)

break;

sprintf(numstr, "%05x", mod[sidx++]);

fprintf(fp2, "M%s05\n", numstr);

}

}

// print the modification

sprintf(numstr, "%06x", sloc);

fillstringzero(numstr);

fprintf (fp2, "E%s\n", numstr);

fclose (fp1);

fclose (fp2);

if (eflag != 0) // error

{

remove (filename\_obj);

remove (filename\_lst);

if (eflag == 2)

printf("Empty Base!\n");

if (eflag == 3)

printf("Error in line %d!\n", line);

}

else

{

printf("\toutput file : [%s], [%s]\n", filename\_lst, filename\_obj);

}

return eflag;

}

// function for the pass 2 algorithm.

// do the pass 2 and make the obj file and the lst file

int PrintSymtab()

{

symtab\* temp;

char symbol[100][20]; // string array for store sorted symbol

char tmp[20]; // tmp string

int locc[100]; // array for the loc of the symbol

int flag = 0; //flag for the success

int i; //variable for loop

int j=0; //variable for loop and sorting

int tmp2;

for (i = 99; i>-1; i--)

{

temp = st[i];

if (temp == NULL) // go to other table

continue;

while (1)

{

if (temp== NULL)

break;

strcpy(symbol[j], temp->symbol); //if find, store in the string array

locc[j++] = temp->locc;

flag++;

temp = temp->link;

}

}

if (flag == 0)

printf("No existing symbols!\n");

else

{

for (i = 0; i<flag-1; i++)

{

for (j = 0; j<flag-1 - i; j++)

{

if (strcmp(symbol[j], symbol[j + 1]) < 0)

{

strcpy(tmp, symbol[j]);

strcpy(symbol[j], symbol[j + 1]);

strcpy(symbol[j + 1], tmp);

tmp2 = locc[j];

locc[j] = locc[j + 1];

locc[j + 1] = tmp2;

}

}

}

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

printf("\t%s\t%04X\n", symbol[i], locc[i]);

}

//sort and print the symbol

free(temp);

return flag;

}

//function for printing the sorted symbol table

int PrintType(char\* command)

{

char junk[6];

char Input[100];

FILE\* fp;

int ret;

ret = 1;

sscanf(command, "%s %s", junk, Input);

fp = fopen(Input, "rt");

if (fp == NULL)

{

printf("There is no file that has the same name as the input!\n");

return -1;

}

else

{

while (1)

{

if (ret == EOF)

break;

else

printf("%c", ret);

ret = fgetc(fp);

}

fclose(fp);

return 1;

}

//print file or if there is no file , print the error message

}

// function for the print the existing file

int finderrorinobj(char \*filename)

{

int len;

int i;

int comma=0;

len = strlen(filename);

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

{

if (filename[i] == '.')

comma++; // find the number of '.'

}

if (comma == 1) {

if (filename[len - 1] == 'j'&&filename[len - 2] == 'b'&&filename[len - 3] == 'o'&&filename[len - 4] == '.')

{

return 1;

}

/// if it is object file, return 1

else

{

return -1;

}

}

else

return -1; //if not return -1

}

//find if the input file is obj file

int loaderone(char \*filename)

{

FILE \*fp; //file pointer to open object file

cstab \*t; // control section tmp variable

cstmp = (cstab\*)malloc(sizeof(cstab));

cstmp->next = NULL; // allocate the control section table for putting the value

char fname[20]; // variable for filename

char text[200]; // variable for string from the file

int CSLTH; // length variable in pass1(algorithm)

int len; // text length

int i; // loop variable

int j = 0; // variable for D array (D text)

int k; // loop variable

int Dseclen; // length of Dsection

int csstart; // start address of program

int flag=0; // flag variable for return(if succeed return 0 , if not return -1)

int ridx; // index variable

int idx; // index variable

fp = fopen(filename, "r");

if (fp == NULL)

flag = -2; //if file not exist

else {

while (1)

{

if (feof(fp))

break; // if endoffile

j = 0;

fgets(text, sizeof(text), fp); // put the all kinds of record in the text

//printf("text : %s\n", text);

if (text[0] == 'H')

{

sscanf(text + 1, "%6s %6x %6x", fname, &csstart, &CSLTH);

//printf("1st. %s %x %x\n", fname, csstart, CSLTH);

strcpy(cstmp->csname, fname);

cstmp->length = CSLTH;

//printf("csst: %x\n", csaddr);

cstmp->address = csstart + csaddr;

} // if H, set the start address and put length, section name, section address in the struct

else if (text[0] == 'D')

{

len = strlen(text);

Dseclen = (len - 1) / 12;

//printf("Dseclen = %d\n", Dseclen);

cstmp->Dnum = Dseclen;

cstmp->D = (extab\*)malloc(sizeof(extab)\*(Dseclen + 1));

// allocate extab

for (i = 1; i < len; i += 12)

{

sscanf(text + i, "%6s %6x", cstmp->D[j].exname, &cstmp->D[j].address);

for (k = 0; k < j; k++)

{

if (!strcmp(cstmp->D[k].exname, cstmp->D[j].exname))

flag = -1;

else

flag = 0;

}//if there is existing one-> error

if (flag == 0)

cstmp->D[j++].address += csaddr;

//put address and name in the estab and check if there is same one existing in the estab

}

}

else if (text[0] == 'R')

{

ridx = (strlen(text) - 1) / 8;

cstmp->R = (reftab\*)malloc(sizeof(reftab)\*(ridx + 10));

for (i = 1, j = 2; i < strlen(text); i += 8, j++)

{

sscanf(text + i, "%2d", &idx);

if (idx == 1)

{

printf("error in reference number!!\n");

return 0;

} // if idx == 1, it must be the program

sscanf(text + i + 2, "%6s", cstmp->R[idx].refname);

cstmp->R[idx].refnum = idx;

cstmp->Rnum = j + 1; // if not put the information in the reference table

strcpy(cstmp->R[1].refname, cstmp->csname);

cstmp->R[1].address = cstmp->address;

cstmp->R[1].refnum = 1; //for the idx == 1, put the information of the program

}

//for (i = 0; i < cstmp->Rnum+2; i++)

//printf("R[%d] name : %s num : %x\n", i, cstmp->R[i].refname, cstmp->R[i].refnum);

}

else if (text[0] == 'E')

{

csaddr += CSLTH; // plus the control section length

if (csaddr > 0xffff)

flag = -1; // if it exceed the available size, flag == -1

else

epaddr = csaddr; // store the endaddress

break; //if E , break

}

}

if (csnum == 0)

C = cstmp; // if there is no node, cstmp is the first

else

{

t = C;

for (; t->next != NULL; t = t->next);

t->next = cstmp;

} // if there exists, put cstmp in the back

csnum++;

fclose(fp); // close the file pointer

}

//fclose(fp);

//free(cstmp);

return flag;

}

// function for the linking loader pass one.

// handle the information of the H,R,D,E

int findref(char \*str,int \*value,char plus)

{

cstab \*tmp; // temp variable for the search

int i; // loop variable

int flag = 0; // flag variable

for (tmp = C; tmp != NULL; tmp = tmp->next) {

for (i = 0; i < tmp->Dnum; i++)

{

//printf("str : %s\n", str);

//printf("tmp name ; %s\n", tmp->D[i].exname);

if (!strcmp(str, tmp->D[i].exname)){

// printf("str : %s and compare : %s\n", str, tmp->D[i].exname);

// printf("address : %x\n", tmp->D[i].address);

flag = 1;

\*value = tmp->D[i].address;

}

}

} // search the symbol in the D that has same name . if found, set value as its address

return flag;

}

//function for finding if reference exists in the D table

int loadertwo(char \*filename,int dic)

{

FILE \*fp; // file pointer for open object file

cstab \*t; // tmp variable for search or put info in the control section C

int i; // loop variable

int add; // address variable

int val; // value varialbe

int len; // lenght variable for modification record

int modlen; //length of modification record

int value; //variable for receive address from the D table

int memval=0; // full hexadecimal value for use in the modification(relocation)

int flag = 0; // flag variable

int sflag = 0; // search flag (if find the same ref name )

int vflag = 0; // flag if found same idx

char text[200]; // string variable for file; receive record

char ref[10]; // string variable for receive the R name

char mem[10]; //string variable for converting hexadecimal

char plus; // flag variable for + or -

fp = fopen(filename, "r");

if (fp == NULL)

flag = -1; // if no file

else {

if (dic == 1)

t = C; // if 1st file

else if (dic == 2) {

t = C;

t = t->next;

} // if 2nd file

else if (dic == 3) {

t = C;

t = t->next;

t = t->next;

} // if 3rd file

while (1)

{

if (feof(fp))

break; // if end of file,break;

fgets(text, sizeof(text), fp);

//printf("text : %s\n", text);

if (text[0] == 'T')

{

sscanf(text + 1, "%6x", &add);

add += t->address;

sscanf(text + 6, "%2x", &len);

for (i = 9; i < strlen(text)-1; i += 2) {

sscanf(text + i, "%2x", &value);

fill(Memory, add + (i - 9) / 2, add + (i - 9) / 2, value);

}

} //T record. put the opcode in the memory

else if (text[0] == 'M')

{

modlen = strlen(text);

//printf("len : %d\n", modlen);

if (modlen == 13) { // proper form has length as 12 but '\n' is put by fgets

sscanf(text + 1, "%6x", &add);

sscanf(text + 7, "%2x", &len);

sscanf(text + 9, "%c", &plus);

sscanf(text + 10, "%2x", &val);

// scanf information from the M record

//printf("val : %x\n", val);

for (i = 0; i < t->Rnum; i++)

{

if (val == t->R[i].refnum) {

strcpy(ref, t->R[i].refname);

vflag = 1;

}

}

// find the name of symbol by the reference number

value = add;

//printf("formal value : %x\n", value);

//printf("ref : %s\n", ref);

if (vflag == 1) {

if (val != 1)

sflag = findref(ref, &value, plus); // find the symbol in the D

else if (val == 1)

value = t->address; // if val == 1 , it is program start address

//printf("later value : %x\n", value);

add += t->address; // move the address to real field

//printf("add : %x\n", add);

if (sflag == 1)

{

memval = (int)Memory[add / 16][add % 16];

memval \*= 256;

memval += (int)Memory[(add + 1) / 16][(add + 1) % 16];

memval \*= 256;

memval += (int)Memory[(add + 2) / 16][(add + 2) % 16];

// make the hexadecimal number to put it into the record

if (memval>0x800000)

{

memval = memval - 0x1000000;

} // if mod is -

//sprintf(hex, "%06x", value);

//sscanf(hex, "%06x", &value);

if (plus=='+')

memval += value; // if +, plus the value

else if (plus=='-')

memval -= value; // if -, minus the value

if (memval<0)

memval += 0xFFFFFF + 1; // if memval is -

sprintf(mem, "%06x", memval);

//printf("memval : %s\n", mem);

sscanf(mem, "%02X", &memval);

fill(Memory, add, add, memval);

sscanf(mem + 2, "%02X", &memval);

fill(Memory, add + 1, add +1, memval);

sscanf(mem + 4, "%02X", &memval);

fill(Memory, add + 2, add + 2,memval);

// put the memval into mem and convert to hexa, and fill it into memory one by one.

}

else

{

flag = -3;

} // if fail to find the symbol in the D

}

}

else {

flag = -1;

} // if modfication record lenght is not 12

}

else if (text[0] == 'E')

{

break; // if E , break;

}

}

}

if(flag == -1)

printf("This is not valid modification record!!\n");

if (flag == -3) {

printf("There is no symbol in reference table!!\n");

flag = -1;

}

fclose(fp); //close the file

return flag;

}

//function for loader pass 2.

//handle the T,M. do actual relocation here and put opcode in proper location.

void printload()

{

cstab \*tmp; // tmp variable for the search and print

int i; // loop variable

tmp = C;

printf("\tcontrol\tsymbol\taddress\tlength\n");

printf("\tsection\tname\n");

printf("\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

for (tmp = C; tmp != NULL; tmp = tmp->next) {

printf("\t%6s\t\t%6x\t%6x\n", tmp->csname, tmp->address,tmp->length);

for (i = 0; i < tmp->Dnum; i++)

printf("\t\t %6s\t%6x \n", tmp->D[i].exname, tmp->D[i].address);

}

}

// function for printing the loader map

void makebreakpoint(int bpval)

{

bp \*tmp,\*t; // bp variable for put it in the bpbox

tmp = (bp\*)malloc(sizeof(bp));

tmp->next = NULL;

tmp->address = bpval; // input is the address

if (bpflag == 1) {

if (br->num == 0)

br->bpoint = tmp; // if no element, put it in the front

else

{

for (t = br->bpoint; t->next != NULL; t = t->next);

t->next = tmp;

} // if there is element, put it in the back

br->num++; // inc the br->num

}

else

printf("Start address of program is not set yet!!\n");

}

//function for make new break point

void printbreakpoint()

{

bp \*t; // variable for print

if (br->num != 0) {

printf("breakpoint\n");

printf("----------\n");

for (t = br->bpoint; t != NULL; t = t->next)

printf("%x\n", t->address);

}

else

printf("There is no breakpoint!!\n");

}

// function for print existing break point

void clearbreakpoint()

{

br->num = 0;

br->bpoint = NULL;

}

//function for clearing existing break point

int main(void)

{

int i, j,k;

int len;

int cnt = 0;

int dstart=0;

int epoint;

int spoint;

int addr; //for edit

int value; // for edit and fill

int startA, endA; // address for fill

int opflag; // flag for the opcode -> if true return 1, else return 0

int eflag;

int length = 0;

int base = 0;

int lflag = 1;

int blanknum=0;

int bpval;

char \*command; // char string to put input without blanks

char \*input; // input string

Node \*hs; //Node for histroy

int flag=0;

char junk[45]; // use in sscanf for the mneomonics or junk

char stp[45]; // use in sscanf for the first string

char edp[45]; // use in sscanf for the middle string in fill, end string in edit and dump

char valp[45]; // use in sscanf for the end string in fill

//char namefile[45];

hashtable \*ht[20]; // hash for the mneomonics

br = (bpbox\*)malloc(sizeof(bpbox));

br->num = 0;

bpflag = 0;

hs = (Node\*)malloc(sizeof(Node));

hs->link = NULL;

Memory = (unsigned char\*\*)calloc(sizeof(unsigned char\*),65536);

progaddr = 0;

for (i = 0; i < 65536; i++)

Memory[i] = (unsigned char\*)calloc(sizeof(unsigned char),16);

// memory and the strings memory allocation

OpcodeInput(ht);

// function execution for the opcode

while (1)

{

printf("sicsim>");

command = (char\*)malloc(sizeof(char) \* 90);

input = (char\*)malloc(sizeof(char) \* 90);

fgets(input, 90, stdin);

len = strlen(input);

input[len - 1] = '\0'; // because \n is put after fgets, change it into NULL

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

{

if (input[i] != ' '&&input[i] != '\t')

break;

}

if (i == len)

printf("Input Error\n");

// if there is string that only have blank, print error message.

else

{

strcpy(command, trim(input)); //terminate existing blanks in the left side and the right side.

len = strlen(command);

if (len > 90 || len <= 0) {

printf("Input Error!\n");

}

// if too long or no input, print error message.

else {

if (!strcmp(command, "quit") || !strcmp(command, "q")) // if quit, free all dynamic things

{

free(input);

free(command);

for (i = 0; i < 65536; i++)

free(Memory[i]);

free(Memory);

free(hs);

break;

}

else if (!strcmp(command, "dir") || !strcmp(command, "d"))

{

MakeHistory(&hs, command);

dir();

free(input);

free(command);

}

else if (!strcmp(command, "help") || !strcmp(command, "h"))

{

MakeHistory(&hs, command);

free(input);

free(command);

help();

}

else if (!strcmp(command, "history") || !strcmp(command, "hi"))

{

if (hs->link == NULL)

MakeHistory(&hs, command); // If history node is empty, just put the command in the node. Not execute printing

else {

MakeHistory(&hs, command);

cnt = 0;

PrintHistory(hs, cnt);

} // if there exists node in the history node, print all commands

free(input);

free(command);

}

else if (!strcmp(command, "dump") || !strcmp(command, "du"))

{

MakeHistory(&hs, command);

if (dstart > 1048575)

dstart = 0;

dstart = Dump(Memory, command, dstart, -1);

free(input);

free(command);

}

else if ((command[0] == 'd' && command[1] == 'u' && (command[2] == ' ' || command[2] == '\t')) || (command[0] == 'd' && command[1] == 'u' && command[2] == 'm' && command[3] == 'p' && (command[4] == ' '||command[4]=='\t')))

{

flag = FindValidDumpInput(command); // by flag, determine error or the case.

if (flag == -1)

printf("This is not valid Input\n"); // if flag==-1 means something wrong in input. print error message

else if (flag == 2) //flag 2 means the case dump %x

{

MakeHistory(&hs, command);

sscanf(command, "%s %[^\n]", junk, stp);

sscanf(stp, "%x", &spoint);

dstart = Dump(Memory, command, spoint, -1);

if (dstart > 1048575)

dstart = 0;

}

else if (flag == 3) // flag 3 means the case dump %x,%x

{

sscanf(command, "%s %[^,] %\*c %[^\n]", junk, stp, edp);

sscanf(stp, "%x", &spoint);

sscanf(edp, "%x", &epoint);

if (spoint > epoint)

{

printf("Not available range!\n");

}

else {

MakeHistory(&hs, command);

dstart= Dump(Memory, command, spoint, epoint);

if (dstart > 1048575)

dstart = 0;

}

free(input);

free(command);

}

else

printf("Input Error!!\n");

}

else if ((command[0] == 'e' && (command[1] == ' ' || command[1] == '\t')) || (command[0] == 'e' && command[1] == 'd' && command[2] == 'i' && command[3] == 't' && (command[4] == ' ' || command[4] == '\t')))

{

flag = FindValidDumpInput(command); // find the valid form for edit

if (flag == 3)

{

MakeHistory(&hs, command);

sscanf(command, "%s %[^,] %\*c %[^\n]", junk, stp, edp);

sscanf(stp, "%x", &addr);

sscanf(edp, "%x", &value);

EditMemory(Memory, addr, value);

}

else

printf("Input Error!!\n");

free(input);

free(command);

}

else if ((command[0] == 'f' && command[1] == ' ') || (command[0] == 'f' && command[1] == 'i' && command[2] == 'l' && command[3] == 'l' && (command[4] == ' '||command[4]=='\t')))//if "f" or "fill" mnemonic ,

{

flag = FindValidDumpInput(command); // find the valid form for fill

if (flag == 4)

{

sscanf(command, "%s %[^,] %\*c %[^,] %\*c %[^\n]", junk, stp, edp,valp);

sscanf(stp, "%x", &startA);

sscanf(edp, "%x", &endA);

sscanf(valp, "%x", &value);

fill(Memory, startA, endA, value);

MakeHistory(&hs, command);

}

else

printf("This is not valid fill input form!!\n");

free(input);

free(command);

}

else if (strcmp("reset", command) == 0) //make all memory to 0

{

MakeHistory(&hs, command);

for (i = 0; i < 65536; i++)

for (j = 0; j < 16; j++)

Memory[i][j] = 0;

free(input);

free(command);

}

else if (command[0] == 'o' && command[1] == 'p' && command[2] == 'c' && command[3] == 'o' && command[4]=='d' && command[5]=='e'&&(command[6]==' '||command[6]=='\t'))

{

opflag = FindOpPrint(command,ht);

if (opflag == 0) {

printf("There is no such mnemonic in the opcodelist!\n");

}

else

MakeHistory(&hs, command);

free(input);

free(command);

}

// print opcode if there is same name

else if (!strcmp(command, "opcodelist"))

{

Opcodelist\_print(ht);

MakeHistory(&hs, command);

free(input);

free(command);

}// print opcodelist

else if (command[0]=='a'&&command[1]=='s'&&command[2]=='s'&&command[3]=='e'&&command[4]=='m'&&command[5]=='b'&&command[6]=='l'&&command[7]=='e'&&(command[8]==' '||command[9]=='\t'))

{

eflag = Makeobject(command, &length, &base,ht); // Pass One 호출

sscanf(command, "%s %s", junk, stp);

if (eflag == 0)

eflag = FinishObject(stp, &base, &length,ht); // Pass One 에서 오류가 발견 되지 않을 경우 Pass Two 호출

if (eflag == 0)

{

MakeHistory(&hs, command);

}

}

else if (!strcmp(command, "symbol"))

{

PrintSymtab();

MakeHistory(&hs, command);

free(command);

free(input);

}

else if (command[0] == 't'&&command[1] == 'y'&&command[2] == 'p'&&command[3] == 'e' && (command[4] == ' ' || command[4] == '\t'))

{

PrintType(command);

MakeHistory(&hs, command);

free(input);

free(command);

}

else if (!strncmp(command, "progaddr",7))

{

sscanf(command, "%s %x", junk, &progaddr);

MakeHistory(&hs, command);

bpflag = 1;

free(input);

free(command);

}

else if (!strncmp(command,"loader",6))

{

C = (cstab\*)malloc(sizeof(cstab) \* 4); //obj file cannot exceed 3

C->next = NULL;

csaddr = progaddr;

for (k = 0; k < strlen(command); k++)

{

if (command[k] == ' ' || command[k] == '\t')

blanknum++;

}

if (blanknum == 1) {

sscanf(command, "%s %s", junk, stp);

lflag = finderrorinobj(stp);

if (lflag == -1)

printf("This is not object file!\n");

else {

lflag = loaderone(stp);

if (lflag == -2)

printf("There is no existing file!!\n");

else if (lflag == -1)

printf("L1Error in %s!!\n", stp);

else {

lflag = loadertwo(stp, 1);

if (lflag == -1)

printf("L2Error in %s!!\n", stp);

}

}

csaddr = progaddr;

}

else if (blanknum == 2)

{

sscanf(command, "%s %s %s", junk, stp, edp);

lflag = finderrorinobj(stp);

lflag = finderrorinobj(edp);

if (lflag == -1)

printf("This is not object file!\n");

else

{

lflag = loaderone(stp);

if(lflag == -2)

printf("There is no existing file!!\n");

else if(lflag == -1)

printf("L1Error in %s!!\n", stp);

else

{

lflag = loaderone(edp);

if (lflag == -2)

printf("There is no existing file!!\n");

else if (lflag == -1)

printf("L1Error in %s!!\n", edp);

else

{

lflag = loadertwo(stp,1);

if (lflag == -1)

printf("L2Error in %s!!\n", stp);

else

{

lflag = loadertwo(edp,2);

if (lflag == -1)

printf("L2Error in %s!!\n", edp);

}

}

}

}

csaddr = progaddr;

}

else if (blanknum == 3)

{

sscanf(command, "%s %s %s %s", junk, stp, edp,valp);

lflag = finderrorinobj(stp);

lflag = finderrorinobj(edp);

lflag = finderrorinobj(valp);

//printload();

if (lflag == -1)

printf("This is not object file!!\n");

else

{

lflag = loaderone(stp);

if (lflag == -2)

printf("There is no existing file!!\n");

else if (lflag == -1)

printf("L1Error in %s!!\n", stp);

else

{

lflag = loaderone(edp);

if (lflag == -2)

printf("There is no existing file!!\n");

else if(lflag == -1)

printf("L1Error in %s!!\n", edp);

else

{

lflag = loaderone(valp);

if (lflag == -2)

printf("There is no existing file!!\n");

else if (lflag == -1)

printf("L1Error in %s!!\n", valp);

else {

lflag = loadertwo(stp, 1);

if (lflag == -1)

printf("L2Error in %s!!\n", stp);

else

{

lflag = loadertwo(edp, 2);

if (lflag == -1)

printf("L2Error in %s!!\n", edp);

else

{

lflag = loadertwo(valp, 3);

if (lflag == -1)

printf("L2Error in %s!!\n", valp);

}

}

}

}

}

csaddr = progaddr;

}

}

else

{

printf("Wrong loader Input!!\n");

}

if (lflag == -2)

lflag = -1;

if (lflag != -1)

{

printload();

}//no error

blanknum = 0;

free(C);

csnum = 0;

progaddr = epaddr;

MakeHistory(&hs, command);

free(input);

free(command);

//free(cstmp);

}

else if (!strncmp(command, "bp", 2))

{

if (!strcmp(command, "bp clear"))

{

clearbreakpoint();

}

else if (!strcmp(command, "bp"))

{

printbreakpoint();

}

else {

sscanf(command, "%s %x", junk, &bpval);

if (bpval > epaddr || bpval < csaddr)

printf("This is not valid breakpoint input!!\n");

else

{

makebreakpoint(bpval);

}

}

}

else

{

printf("There is no such command!\n");

free(input);

free(command);

}

// if Wrong command, print error

}

}

}

return 0;

}