과목 명: 시스템프로그래밍

담당 교수 명: 김 지 환

<<Assignment 2>>

**서강대학교 컴퓨터학과**

**[20151616]**

**[최승환]**

목 차

1. 프로그램 개요 3

2. 프로그램 설명 3

2.1 프로그램 흐름도 3

3. 모듈 정의 3

3.1 모듈 이름 : int IsAssemble(char\* input) 3

3.1.1 기능 3

3.1.2 사용 변수 3

3.2 모듈 이름: void Pass1(char\* input) 3

3.2.1 기능 3

3.2.2 사용 변수 3

3.3 모듈이름: void Pass2(char\* input, int endloc) 4

3.3.1 기능 4

3.3.2 사용변수 4

3.4 모듈이름: void PrintSymbol() 4

3.4.1 기능 4

3.4.2 사용변수 4

3.5 모듈이름: int PrintFile() 4

3.5.1 기능 4

3.5.2 사용변수 4

4. 전역 변수 정의 4

4.1 SYM 4

4.2 SYM\* symboltable 4

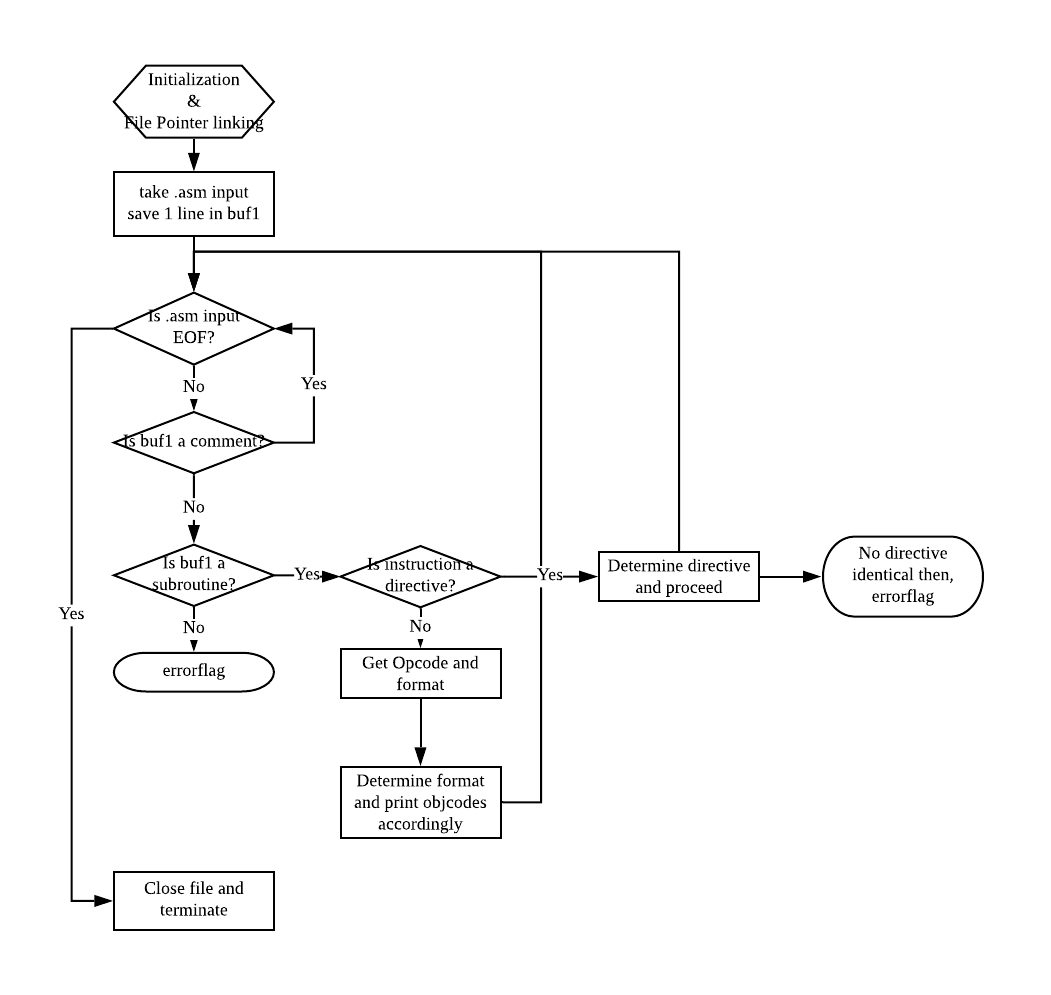
4.3 enum Reg, enum Flags 4

# 프로그램 개요

Pass1, Pass2 를 포함한 어셈블러가 구현되어 있고, 이를 통해 리스트파일과 오브젝트 파일을 생성할 수 있는 프로그램이다. 리스트파일에는 Loc, Objectcode 가 포함되어 있고, 오브젝트 파일은 Header record Text Record End Record ,Modification Record 로 구성되어 있다.

# 프로그램 설명

## 프로그램 흐름도



**그림 1> 프로그램 흐름도**

# 모듈 정의

## 모듈 이름 : int IsAssemble(char\* input)

### 기능

Display\_instructions를 호출해 프로그램이 하는 일을 설명한 후, 사용자의 입력을 받아서 나머지 모듈을 호출하거나 끝낸다.

### 사용 변수

* + - 1. int wnum, opcode, niflag, xbpeflag, disp, format, errorflag

wnum 은 sscanf 를 통해 입력받은 인수의 개수를 저장하는 변수이다. 이를 통해 어떤 형식의 입력이 들어왔는지 판별할 수 있다.

opcode는 instruction의 opcode를 저장하는 변수이다.

niflgag, xbpeflag는 타입에 따라서 플래그를 저장하는 변수이다. 저장하는 값은 enum을 이용한다.

disp는 displacement를 저장하는 값이다.

format은 instruction의 타입을 저장하는 변수이다. 이를 이용하면 다음 loc의 값을 구할 수 있고, 출력형식도 결정할 수 있다.

errorflag는 error를 찾아 에러처리를 하기 위한 변수이다.

* + - 1. int line, len, loc, bloc, byte, reg1, reg2

line 은 현재 참조하고 있는 줄의 줄 수를 저장하는 변수이다. 이 값은 반복할때마다 5씩 늘어난다. len 은 참조하고 있는 줄의 loc을 계산하기 위해 사용된다. 현재 참조하고 있는 줄의 형식을 이 변수에 저장한다. 그 후 연산이 끝나며 그 값을 loc에 더하고 초기화시킨다. bloc 은 b 레지스터가 가리키고 있는 주소를 저장한다. 이 때문에 한 줄을 입력받을 때마다 해당 instruction 이 LDB인지 확인해준다. 맞을 경우에 이 값을 참조하고 있는 줄의 주소 즉, loc 값으로 바꿔준다. byte 는 byte 명령어를 수행할 때 그 값을 저장하는 변수이다. reg1 reg2 는 레지스터 값을 참조하는 변수이다. 2형 명령어가 나왔을 경우 이 변수들에 값을 저장해 출력시킨다. 가독성을 위해서 이 과정은 GetReg 라는 함수를 통해 레지스터 값을 전달받도록 했다.

* + - 1. char\* buf1, oper, oper2

buf1은 입력받은 한 줄을 저장하는 배열이다. while문 이 한 번 반복할 때마다 새로운 값이 입력되며, 이 행렬의 첫번째 char 가 알파벳인지, 공백인지, . 인지를 구분해 이 줄이 어떤 명령어를 구사할지 판별한다.

oper 과 oper2는 입력받는 operand를 입력받는 변수이다.

## 모듈 이름: void Pass1(char\* input)

### 기능

symbol table을 만들어주는 역할을 한다. 이를 위해서는 각 줄의 loc의 값을 계산해줘야 하는데, 이 때문에 각 줄의 instruction과 operand가 어떻게 입력되는지를 파악해야 한다. 이 때문에 기본 구조는 IsAssemble과 거의 같다고 할 수 있다. 다른 점이 있다면, 형식에 따라 .lst 나 .obj 파일에 출력하는 과정을 생략하고, loc만 계산해, symboltable을 우선적으로 만들어준다는 것이다. 또한, 이 과정이 있어야 IsAssemble에서 objectcode를 생성해낼 수 있다.

### 사용 변수

* + - 1. char\* buf1, sub,iname, oper, oper2

buf1은 IsAssemble과 같이 한 줄을 입력받아 저장하는 행렬이다. sub iname은 각각 서브루틴과 명령어를 저장하는 변수이다. oper oper2도 IsAssemble에서와 같이 operand를 저장하는 두 행렬이다.

* + - 1. int loc, len, format, wnum,opcode,errorflag

loc len format 은 각각 주소 각 줄의 objectcode 길이 형식 을 저장하는 변수이다. wnum은 sscanf의 리턴값을 저장하는 변수로, 몇 개의 단어가 있는지를 파악하기 위한 변수이다. opcode errorflag는 IsAssemble에서와 용처가 같다.

## 모듈이름: void Pass2(char\* input, int endloc)

### 기능

### Pass2는 Objectcode를 만들어주는 것을 의미하지만 이 코드의 경우에는 그 역할을 IsAssemble에서 처리해주고 있다. Pass2 함수는 이 objectcode를 이용해 object file을 생성해주는 역할을 한다. 이 object file은 헤더 레코드, 텍스트 레코드, 엔드 레코드, 그리고 모디피케이션 레코드로 구성되어 있고, .lst 파일을 참조해 이 레코드들을 완성한다.

### 사용변수

* + - 1. int wnum, loc, line, start, cflag, len, modcnt, endrecord, endflag

wnum, loc, line, len 은 IsAssemble과 동일한 역할을 한다. cflag 는 출력 형식을 맞추기 위해 한 줄이 바뀌었을 때 체크해주는 역할을 한다. cflag가 1일 경우, 줄이 바뀌었고, 그 후 레코드가 쓰이지 않은 상태라는 뜻이다. 즉, 현재 줄의 첫번째 자리에 커서가 있다는 뜻이다.

* + - 1. char\* subr, iname, oper, oper2, buf1, objcode, trecord

subr 은 서브루틴을 저장하는 행렬, iname은 instruction을 저장하는 변수, oper, oper2는 operand를 저장하는 변수, objcode는 입력받은 줄의 objectcode를 저장하는 변수, trecord는 text record를 저장하는 변수이다. trecord의 경우 objectcode를 strcat을 이용해 반복적으로 뒤에 붙여준다. 이후 이 길이가 최대 길이 수를 초과하게 될 경우에는 줄을 바꾸고 초기화시켜준다.

## 모듈이름: void PrintSymbol()

### 기능

Symboltable을 출력해주는 함수이다. Pass1을 통해서 symboltable에 링크드리스트 형태로 저장해둔 값을 참조하며 출력해준다.

### 사용변수

### SYM\* currnode

currnode는 각 노드를 참조하기 위해 사용하는 변수이다.

## 모듈이름: int PrintFile(char\* input)

### 기능

파일 내용을 출력해주는 함수이다. 만약 파일이 존재하지 않는다면 에러처리를 해준다.

### 사용변수

* + - 1. char\* buf1, fname

buf1은 한 줄 입력을 받는 변수이다. fname은 파일 이름을 저장하는 변수이다.

# 전역 변수 정의

수입을 저장한다.

## SYM

symboltable의 각 노드를 저장하는 구조체이다. loc과 subr, link 를 변수로 가지고 있다. 각각, 주솟값, 서브루틴, 다음 노드의 링크가 저장되어 있다.

## SYM\* symboltable

symboltable의 헤드 역할을 하는 변수이다. CreateSym을 호출하면 이 변수를 참조해 노드를 검색하거나 주솟값을 넣어준다.

## enum Reg, Flags

레지스터를 사용할 때 objectcode에 출력해주는 값은 레지스터 별로 다르며, 코드에는 총 8개가 존재한다. 가독성을 높여주기 위해 이 값을 각각, enum 으로 선언해뒀다.

Flags 는 xbpeflag에 더할 값을 의미한다. 16진수로 바꿨을 때 각각 1, 2, 4, 8 을 더해주면 flag가 조정되기 때문에 이 값을 enum으로 선언했다.

# 코드

* 20151616.c

#include "20151616.h"

int main()

{

NODE\* head = NULL;

//headnode of the linked list of history

char input[MAX];

//array to save the input line

int num1=0,num2=0,num3=0;

//int variables to save the number inputs in dump, edit, fill etc.

int addr=0;

//saves the index of the last visited memory

memset(mem,0,sizeof(char)\*1048576);

CreateHash();

//Creates Hash table with hash function. Constructs linked list. If the file crashes in the first place, first point to check

do

{

#include "20151616.h"

int main()

{

NODE\* head = NULL;

//headnode of the linked list of history

char input[MAX];

//array to save the input line

int num1=0,num2=0,num3=0;

//int variables to save the number inputs in dump, edit, fill etc.

int addr=0;

//saves the index of the last visited memory

memset(mem,0,sizeof(char)\*1048576);

CreateHash();

//Creates Hash table with hash function. Constructs linked list. If the file crashes in the first place, first point to check

do

{

printf("sicsim> ");

fgets(input,MAX,stdin);

input[strlen(input)-1] = '\0';

//changes the last index from '\n' to '\0'. Enables it to be used in strcmp and check it's end

if(!strcmp(input,instruction[0])||!strcmp(input,instruction[10]))

//compares the input and instruction to determine the operation. Either two kinds of input is valid for 'help' instruction

{

InputHistory(&head, input);

//When Valid instruction was input, saves the line in the history lists.

PrintInstruction();

//Operation for 'help'

}

else if(!strcmp(input,instruction[1])||!strcmp(input,instruction[11]))

{

InputHistory(&head, input);

PrintDir();

//Operation for 'dir'

}

else if(!strcmp(input,instruction[3])||!strcmp(input,instruction[13]))

{

InputHistory(&head, input);

PrintHistory(head);

//Operation for 'history'

}

else if(!strncmp(input,instruction[4],4)||!strncmp(input,instruction[14],2))

//for dump, edit and fill instruction, program needs to compare first few characters only. Therefore used strncmp.

{

if(IsFormal(&num1,&num2,&addr,input))

//'dump' instruction has several forms. IsFormal function checks if each of the instruction is valid and returns 1 for valid ones, 0 for wrong ones.

{

InputHistory(&head,input);

//Only valid inputs will be saved in the history.

}

}

else if(!strncmp(input, instruction[5],4)||!strncmp(input,instruction[15],1))

{

if(IsFormalE(&num1, &num2, input))

//Similar kind to IsFormal function. Used for Edit instruction.

{

InputHistory(&head,input);

}

}

else if(!strncmp(input, instruction[6],4)||!strncmp(input,instruction[16],1))

{

if(IsFormalF(&num1, &num2, &num3, input))

{

InputHistory(&head,input);

}

}

else if(!strcmp(input,instruction[7]))

{

memset(mem,0,sizeof(char)\*1048576);

//initializes the memory's value to 0. 1048576 is the decimal value of hexadecimal FFFFF.

InputHistory(&head, input);

}

else if(!strncmp(input,instruction[8],7))

{

if(PrintOp(input))

InputHistory(&head, input);

}

else if(!strcmp(input, instruction[9]))

{

PrintOpList();

InputHistory(&head, input);

}

else if(!strncmp(input, instruction[17],9))

{

if(!IsAssemble(input))

//take file name input in the function and determine.

{

printf("File unavailable!\n");

continue;

}

InputHistory(&head, input);

}

else if(!strncmp(input, instruction[18], 5))

//compares till space

{

if(PrintFile(input))

{

InputHistory(&head, input);

}

else

{

printf("File with such name does not exist\n");

}

}

else if(!strcmp(input, instruction[19]))

{

PrintSymbol();

InputHistory(&head, input);

}

}while((strcmp(input, "quit")&&strcmp(input, "q")));

//terminates the program when input was q or quit

return 1;

}

void PrintInstruction()

//Prints all instructions that can be done in the program.

{

printf("h[elp]\nd[ir]\nq[uit]\nhi[story]\ndu[mp] [start, end]\ne[dit] address, value\nf[ill] start, end, value\nreset\nopcode mnemonic\nopcodelist\nassemble filename\ntype filename\nsymbol\n");

return;

}

int IsAssemble(char\* input)

{

int i=0, j=0,opcode=0,niflag=0,xbpeflag=0,disp=0, format=0,errorflag=0;

int line=0, loc=0, bloc=0, byte=0,reg1=0,reg2=0;

int len=0;

char\* fname = malloc(sizeof(char)\*20);

char\* buf1=malloc(sizeof(char)\*100);

char\* buf2=malloc(sizeof(char)\*20);

char\* sub = malloc(sizeof(char)\*7);

//sub for subroutine

char \*iname = malloc(sizeof(char)\*7);

//iname for instruction name

char \*oper = malloc(sizeof(char)\*20);

//oper for operand

char\* oper2=malloc(sizeof(char)\*20);

FILE\* fp, \*lstfp, \*objfp;

int wnum=0;

ResetSym();

//wnum for word number(number of words in a single line)

sscanf(input,"%s %s",buf1,fname);

for(i=0;fname[i] != '.';++i)

//find the name part of file, ex) 2\_5.asm-> 2\_5

{

if(fname[i] == '\0')

{

printf("Source file extension not available!\n");

return 0;

}

}

if(strcmp(".asm",fname+i))

{

printf("Source file extension not available!\n");

return 0;

}

Pass1(fname);

if(!(fp = fopen(fname,"r")))

{

printf("No existing file named %s\n",fname);

return 0;

}

fname[i] = '\0';

strcat(fname,".lst");

if(!(lstfp = fopen(fname,"w")))

{

printf("\nFailed to create a file %s\n",fname);

return 0;

}

printf("\t\toutput file : [%s]",fname);

fname[i] = '\0';

strcat(fname,".obj");

if(!(objfp = fopen(fname,"w")))

{

printf("\nFailed to create a file %s\n",fname);

return 0;

}

printf(", [%s]\n",fname);

fname[i] = '\0';

//completed printing output file

while(fgets(buf1,MAX,fp))

{

line+=5;

if(sscanf(buf1,"%s",buf2)<1)

{

break;

}

fprintf(lstfp,"%5d\t",line);

if(buf1[0] == '.')

{

fprintf(lstfp,"%s",buf1);

}

else

{

if(IsCap(buf1[0]))

//First letter is a capital. Which means the line is a subroutine

{

wnum=sscanf(buf1,"%s %s %s %s",sub,iname,oper,oper2);

buf1[strlen(buf1)-1]='\t';

//directives@@@

if((GetOp(iname)<0)&&(GetForm(iname)!=4))

{

format=0;

if(!strcmp(iname,"START"))

{

fprintf(lstfp,"%04X\t%s\n",loc,buf1);

}

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

{

fprintf(lstfp,"\t%s\n",buf1);

}

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

{

fprintf(lstfp,"%04X\t%s\n",loc,buf1);

format = (StrToDec(oper)\*3);

}

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

{

fprintf(lstfp,"%04X\t%s\n",loc,buf1);

format = StrToDec(oper);

}

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

{

fprintf(lstfp,"%04X\t%s%06X\n",loc,buf1,StrToDec(oper));

}

else if(!strcmp(iname,"BYTE"))

{

fprintf(lstfp,"%04X\t%s",loc,buf1);

if(oper[0] == 'X')

{

sscanf(oper+2,"%X",&byte);

format=1;

fprintf(lstfp,"%02X\n",byte);

}

else if(oper[0] == 'C')

{

format = strlen(oper)-3;

i=0;

while(oper[i+2]!='\0')

{

oper[i] = oper[i+2];

i++;

}

oper[i]='\0';

for(j=0;oper[j]!='\0';++j)

{

if(IsCap(oper[j]))

{

fprintf(lstfp,"%02X",(int)oper[j]);

}

}

fprintf(lstfp,"\n");

}

else

{

errorflag=1;

}

}

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

{

fprintf(lstfp,"\t%s\n",buf1);

}

else

{

errorflag =1;

}

len = format;

format =5;

}

//end of directives

else

{

fprintf(lstfp,"%04X\t",loc);

fprintf(lstfp,"%s",buf1);

if(wnum == 4)

//When input was a subroutine and 2 operands

{

opcode = GetOp(iname);

format = GetForm(iname);

if(GetCom(oper))

{

strchr(oper,',')[0] = '\0';

}

}

else if(wnum == 3)

//a subroutine command with one operand

{

opcode = GetOp(iname);

format = GetForm(iname);

if(GetCom(oper))

{

strchr(oper,',')[0] = '\0';

}

}

else if(wnum == 2)

{

opcode = GetOp(iname);

format = GetForm(iname);

if(GetCom(oper))

{

strchr(oper,',')[0] = '\0';

}

}

else if(wnum == 1)

{

}

//format 4 disp calculation

len = format;

}

}

else if(buf1[0] == ' ')

{

wnum = sscanf(buf1,"%s %s %s",iname,oper,oper2);

buf1[strlen(buf1)-1]='\t';

if((GetOp(iname)<0)&&(GetForm(iname)!=4))

{

format=0;

if(!strcmp(iname,"START"))

{

fprintf(lstfp,"%04X\t%s\n",loc,buf1);

}

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

{

fprintf(lstfp,"\t%s",buf1);

}

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

{

fprintf(lstfp,"%04X%s\n",loc,buf1);

format = (StrToDec(oper)\*3);

}

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

{

fprintf(lstfp,"%04X%s\n",loc,buf1);

format = StrToDec(oper);

}

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

{

fprintf(lstfp,"%04X\t%s%06X\n",loc,buf1,StrToDec(oper));

}

else if(!strcmp(iname,"BYTE"))

{

fprintf(lstfp,"%04X%s",loc,buf1);

if(oper[0] == 'X')

{

sscanf(oper+2,"%X",&byte);

format=1;

fprintf(lstfp,"%02X\n",byte);

}

else if(oper[0] == 'C')

{

format = strlen(oper)-3;

i=0;

while(oper[i+2]!='\0')

{

oper[i] = oper[i+2];

i++;

}

oper[i]='\0';

for(j=0;oper[j]!='\0';++j)

{

if(IsCap(oper[j]))

{

fprintf(lstfp,"%02X",(int)oper[j]);

}

}

fprintf(lstfp,"\n");

}

else

{

errorflag=1;

}

}

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

{

fprintf(lstfp,"\t%s\n",buf1);

}

else

{

errorflag=1;

}

len = format;

format =5;

}

else

{

fprintf(lstfp,"%04X\t",loc);

fprintf(lstfp,"%s",buf1);

if(wnum == 3)

{

opcode = GetOp(iname);

format = GetForm(iname);

if(GetCom(oper))

{

strchr(oper,',')[0] = '\0';

}

}

else if(wnum == 2)

{

opcode = GetOp(iname);

format = GetForm(iname);

if(GetCom(oper))

{

strchr(oper,',')[0] = '\0';

}

}

else if(wnum == 1)

{

opcode = GetOp(iname);

format = GetForm(iname);

}

len = format;

}

}

else

{

errorflag = 1;

}

if(oper[0] == '@')

{

niflag = 2;

oper = oper + 1;

}

else if(oper[0] == '#')

{

niflag = 1;

oper = oper + 1;

}

else

{

niflag = 3;

}

//@@@@@@@@@@@@@@niflag=0 case consider???

//if base register changed, save its value

if(!strcmp(iname,"LDB"))

{

bloc = GetSymLoc(oper);

}

//starting disp calculation

loc += len;

if(!strcmp(iname,"RSUB"))

{

fprintf(lstfp,"4F0000\n");

continue;

}

switch (format)

{

//fprint according to the format

case 1:

//format1

fprintf(lstfp,"\n");

break;

case 2:

//format2

fprintf(lstfp,"%02X%01X",opcode,GetReg(oper));

if(GetReg(oper2)>0)

{

fprintf(lstfp,"%1X\n",GetReg(oper2));

}

else

{

fprintf(lstfp,"0\n");

}

break;

case 3:

//format3

//flag set

if(!strcmp(oper2,"X"))

//b flag set ex) SBUF STCH BUFFER, X

{

xbpeflag += FlagX;

}

if(IsNum(oper[0]))

{

disp = StrToDec(oper);

}

else

{

disp = GetSymLoc(oper);

{

if((disp - loc) > 2048 || (disp-loc) < -2048)

//out of pc relative range. use base register

{

xbpeflag += FlagB;

disp -= bloc;

}

else

{

//in pc relative range

xbpeflag += FlagP;

disp -= loc;

}

}

}

disp = disp&4095;

//cut unwanted bits

fprintf(lstfp,"%02X%01X%03X\n",opcode+niflag,xbpeflag,disp);

break;

case 4:

//format(4

opcode = GetOp(iname+1);

xbpeflag += FlagE;

//set e flag

if(IsNum(oper[0]))

{

disp = StrToDec(oper);

}

else

{

if((disp = GetSymLoc(oper)) == -1)

{

errorflag=1;

}

}

fprintf(lstfp,"%02X%01X%05X\n",opcode+niflag,xbpeflag,disp);

break;

}

niflag = 0;

xbpeflag = 0;

format=0;

len=0;

memset(oper,0,strlen(oper));

memset(oper2,0,strlen(oper2));

}

if(errorflag)

{

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

ResetSym();

fclose(lstfp);

fclose(objfp);

fclose(fp);

strcat(fname,".obj");

remove(fname);

fname[i] = '\0';

strcat(fname,".lst");

remove(fname);

return 0;

}

}

fclose(lstfp);

fclose(objfp);

fclose(fp);

Pass2(fname,loc);

return 1;

}

void PrintSymbol()

{

SYM\* currnode = symboltable;

if(currnode == NULL)

{

printf("Empty symbol node!\n");

return;

}

while(currnode != NULL)

{

printf("\t%s\t%04X\n",currnode->subr,currnode->loc);

currnode = currnode -> link;

}

return;

}

int PrintFile(char\* input)

{

FILE\* fp;

char\* buf1 = malloc(sizeof(char)\*MAX);

char\* fname = malloc(sizeof(char)\*MAX);

int wnum=0;

memset(buf1,0,MAX);

sscanf(input,"%s %s",buf1,fname);

if(fp = fopen(fname,"r"))

{

while(fgets(buf1,MAX-1,fp))

{

printf("%s",buf1);

}

}

else

{

printf("File %s unavailable access for reading!\n",input);

free(buf1);

free(fname);

return 0;

}

printf("\n");

free(buf1);

free(fname);

fclose(fp);

return 1;

}

void PrintDir()

//Prints all the files in the current directory. prints different characters for different types of files.

{

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

//directory pointer is used to save names and types of the files in the directory

struct dirent\* dir;

//dirent\* structure is used to check whether there is files left to check or not

struct stat filestat;

//file's type is saved in filestat variable

while((dir=readdir(dname)) != NULL)

//When dir is NULL, there is no more file to check.

{

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

//print file's name

stat(dir->d\_name,&filestat);

//stat saves file's name in variable dir, and saves file type in filestat

if(S\_ISDIR(filestat.st\_mode))

//Checks if the current file is a directory

{

printf("/");

}

else if(filestat.st\_mode&S\_IEXEC)

//Checks if the file is executable.

{

printf("\*");

}

printf("\t");

}

printf("\n");

closedir(dname);

return;

}

void InputHistory(NODE\*\* head, char\* Input)

//Saves the input in a linked list. Validity of the input will be checked before operating this function

{

NODE\* newnode;

NODE\* currnode=\*head;

newnode = (NODE\*)malloc(sizeof(NODE));

//Allocate newnode a size of NODE

if(newnode == NULL)

//When malloc was unsuccessful print error message and return

{

printf("Error creating new node!\n");

return;

}

strcpy(newnode->InputString,Input);

//Copy the input to the newnode with strcpy

newnode->link = NULL;

//initializes the next link to NULL

if(\*head == NULL)

{

\*head = newnode;

//When the linked list is empty, set newnode as the first node. therefore, newnode becomes the head of the list.

return;

}

while(currnode->link != NULL)

{

currnode = currnode -> link;

//Find the last node of the linked list.

}

currnode->link = newnode;

//Set the last node as newnode by setting the last node's link as the newnode

return;

}

void PrintHistory(NODE\* head)

{

int count=0;

NODE\* currnode = head;

if(head == NULL)

{

printf("list is empty!\n");

return;

}

do

{

printf("%d\t%s\n",++count,currnode->InputString);

currnode = currnode->link;

//Print the current node's string until you find the last node

}

while(currnode != NULL);

//when you find the NULL pointer, it means there is no more nodes to go through

return;

}

int IsIname(char\* input)

{

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

int index=0;

HASH\* currnode=NULL;

index = HashFunc(input);

//identical instruction will have identical hash value

currnode = hashtable[index];

while(currnode != NULL)

{

if(!strcmp(input,currnode->opinstruction))

break;

//check if such instruction exists in this index of hashtable

currnode = currnode->link;

}

if(currnode == NULL)

{

printf("instruction %s not found!\n",input);

//when input does not match any instruction in the linked list print error message

return 0;

}

printf("opcode for %s is %X\n",currnode->opinstruction,currnode->opcode);

//print the right opcode for the instruction

return 1;

}

int IsOper(char\* input)

{

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

return 1;

}

int IsNum(char inputchar)

//Returns 1 if a character is a ascii code for number 0 to 9

{

if('/'<inputchar && inputchar<':')

{

return 1;

}

return 0;

}

int IsCap(char inputchar)

//Returns 1 if a character is a Ascii code for capital letter

{

if('@'<inputchar && inputchar<'[')

{

return 1;

}

return 0;

}

int IsCapHex(char inputchar)

//Returns 1 if a character is a Ascii code for capital letter for Hexadecimal A to F

{

if('@'<inputchar && inputchar<'G')

{

return 1;

}

return 0;

}

int IsLow(char inputchar)

//Returns 1 if a character is a Ascii code for lowercase letter

{

if('`'<inputchar && inputchar<'{')

{

return 1;

}

return 0;

}

int IsHex(char\* input)

{

int i=0;

for(;input[i]!='\0';++i)

{

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

return 0;

}

return 1;

}

int StrToDec(char\* input)

{

int i=0;

int val=0;

while(input[i] != '\0')

{

val \*= 10;

val += (input[i]-'0');

++i;

}

return val;

}

int GetCom(char\* input)

//Returns the number of ','(commas) in a string

{

int i=0;

int count=0;

for(i=0;input[i]!='\0';++i)

{

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

{

++count;

}

}

return count;

}

int GetReg(char\* oper)

{

if(!strcmp(oper,"A"))

{

return RegA;

}

else if(!strcmp(oper,"X"))

{

return RegX;

}

else if(!strcmp(oper,"L"))

{

return RegL;

}

else if(!strcmp(oper,"B"))

{

return RegB;

}

else if(!strcmp(oper,"S"))

{

return RegS;

}

else if(!strcmp(oper,"T"))

{

return RegT;

}

else if(!strcmp(oper,"F"))

{

return RegF;

}

else if(!strncmp(oper,"PC",2))

{

return RegPC;

}

else if(!strncmp(oper,"SW",2))

{

return RegSW;

}

return -1;

}

int IsFormal(int\* num1, int\* num2, int\* addr, char\* input)

{

char buf1[6],buf2[6],buf3[6],buf4[6],buf5[6];

//buf arrays are used to take strings to check errors

int n1=0,n2=0;

//n1 and n2 saves the number parts of the input

int wnum=0;

wnum=sscanf(input,"%s%X ,%X",buf1,&n1,&n2);

//wnum(wordnum) saves the number of words taken by sscanf. sscanf returns the number of words that are successfully saved

if( (n1<0) || (n2<0))

{

//negative input error check

return 0;

}

if((wnum == 3)&&(GetCom(input)==1))

//when 3 parts of arguments were passed, there should only be one comma

{

if(sscanf(input,"%s%s%s%s%s",buf1,buf2,buf3,buf4,buf5)>4)

//if the input contains unwanted string parts, return 0. Reason for 5 buffers is because hexadecimals can be considered as strings too

{

return 0;

}

if((n1 > n2) || (n2>1048575))

//When n1 is bigger than n2 or n2 is bigger than maximum index fail.

{

return 0;

}

else

{

PrintMem(n1,n2,addr);

//PrintMem will print from start(num1) to end(num2)

}

}

else if(wnum == 2&&(GetCom(input)==0))

//when only one number input present

{

if(sscanf(input,"%s%s%s",buf1,buf2,buf3)>2)

{

return 0;

}

if(!IsHex(buf2))

{

return 0;

}

if(n1>1048575)

{

//when n1 is bigger than max index fail.

return 0;

}

else

{

PrintMem(n1,n1+159,addr);

//No need to check n1+ 160 since printmem checks it

}

}

else if(wnum == 1&&(strchr(input,' ')==NULL))

//When only one word was input. 'dump' or 'du'. Then there should be no other words, therefore no spaces should exist.

{

PrintMem(\*addr+1,\*addr+160,addr);

}

else

//cases where no input was existent or 4 or more words were contained

{

return 0;

}

return 1;

}

int IsFormalE(int\* num1, int\* num2, char\* input)

//Operation for edit

{

int i=0;

int val=0;

if(input[1] == '\0')

{

//when user typed 'e' only

return 0;

}

if(!strncmp(input,instruction[5],4) && input[4] == '\0')

{

//when user typed 'edit' only

return 0;

}

if(!strncmp(input,instruction[5],4))

{

//when user typed 'edit' + sth

i=4;

}

else if(!strncmp(input,instruction[15],1))

{

//when user typed 'e' + sth

i=1;

}

else

{

return 0;

}

//error check for types like 'east' or 'edin'

if(input[i] != ' ')

{

return 0;

}

//input's index i is currently on index of ' ' after 'edit' or 'e'

while(input[i+1] == ' ')

{

//when you type multiple spaces after edit like 'e ' or 'edit '

++i;

}

if(!(IsNum(input[i+1])||IsCapHex(input[i+1])))

{

//after 'edit ' or 'e ' must come a number or a capital

return 0;

}

++i;

while(IsNum(input[i])||IsCapHex(input[i]))

{

val\*=16;

if(IsNum(input[i]))

val += (input[i]-48);

else

val += (input[i]-55);

++i;

}

\*num1 = val;

val =0;

//end of num1 input

while(input[i] == ' ')

{

//when you type multiple spaces after num1 like 'e %d ' or 'edit %d '

++i;

}

if(input[i] != ',')

{

return 0;

}

while(input[i+1] == ' ')

{

//when you type multiple spaces after comma like 'e %d, ' or 'edit %d, '

++i;

}

if(!(IsNum(input[i+1])||IsCapHex(input[i+1])))

{

//when the first letter after 'edis %d, ' is not a number or capital letter, exception error

return 0;

}

++i;

while(IsNum(input[i])||IsCapHex(input[i]))

{

val\*=16;

if(IsNum(input[i]))

val += (input[i]-48);

else

val += (input[i]-55);

++i;

}

\*num2 = val;

val=0;

//end of num2 input

if(input[i] == '\0')

{

//when user typed 'edit %d, %d' or 'e %d, %d'

if(\*num1 > \*num2)

{

return 0;

}

EditMem(\*num1, \*num2);

//EditMem will Edit value of address in num1 into value of num2

return 1;

}

return 0;

}

int IsFormalF(int\* num1, int\* num2, int\* num3, char\* input)

{

int i=0;

int val=0;

if(input[1] == '\0')

{

//when user typed 'f' only

return 0;

}

if(!strncmp(input,instruction[6],4) && input[4] == '\0')

{

//when user typed 'fill' only

return 0;

}

if(!strncmp(input,instruction[6],4))

{

//when user typed 'fill' + sth

i=4;

}

else if(!strncmp(input,instruction[16],1))

{

//when user typed 'f' + sth

i=1;

}

else

{

return 0;

}

//error check for types like 'finn' or 'fire'

if(input[i] != ' ')

{

return 0;

}

//input's index i is currently on index of ' ' after 'fill' or 'f'

while(input[i+1] == ' ')

{

//when you type multiple spaces after fill like 'f ' or 'fill '

++i;

}

if(!(IsNum(input[i+1])||IsCapHex(input[i+1])))

{

//after 'fill ' or 'f ' must come a number or a capital

return 0;

}

++i;

while(IsNum(input[i])||IsCapHex(input[i]))

{

val\*=16;

if(IsNum(input[i]))

val += (input[i]-48);

else

val += (input[i]-55);

++i;

}

\*num1 = val;

val =0;

//end of num1 input

while(input[i] == ' ')

{

//when you type multiple spaces after num1 like 'f %d ' or 'fill %d '

++i;

}

if(input[i] != ',')

{

return 0;

}

while(input[i+1] == ' ')

{

//when you type multiple spaces after comma like 'f %d, ' or 'fill %d, '

++i;

}

if(!(IsNum(input[i+1])||IsCapHex(input[i+1])))

{

//when the first letter after 'fill %d, ' is not a number or capital letter, exception error

return 0;

}

++i;

while(IsNum(input[i])||IsCapHex(input[i]))

{

val\*=16;

if(IsNum(input[i]))

val += (input[i]-48);

else

val += (input[i]-55);

++i;

}

\*num2 = val;

val=0;

//end of num2 input

while(input[i] == ' ')

{

//when you type multiple spaces after num1 like 'f %d, %d ' or 'fill %d, %d '

++i;

}

if(input[i] != ',')

{

return 0;

}

while(input[i+1] == ' ')

{

//when you type multiple spaces after comma like 'f %d, %d, ' or 'fill %d, %d, '

++i;

}

if(!(IsNum(input[i+1])||IsCapHex(input[i+1])))

{

//when the first letter after 'fill %d, %d, ' is not a number or capital letter, exception error

return 0;

}

++i;

while(IsNum(input[i])||IsCapHex(input[i]))

{

val\*=16;

if(IsNum(input[i]))

val += (input[i]-48);

else

val += (input[i]-55);

++i;

}

\*num3 = val;

val=0;

//end of num3 input

if(input[i] == '\0')

{

//when user typed 'fill %d, %d, %d' or 'f %d, %d, %d'

if(\*num1 > \*num2)

{

return 0;

}

FillMem(\*num1, \*num2, \*num3);

//FillMem will Fill values of address in num1 to num2 into value of num3

return 1;

}

return 0;

}

void PrintMem(int start, int end, int\* addr)

//Prints the memory in range of start to end

{

int curridx=start;

int temp=start/16\*16;

//temp saves the starting address of the line

int endidx=0;

if(end > 1048575)

{

//when the print range exceeds the maximum memory index

end = 1048575;

}

for(curridx=start;curridx<=end;curridx+=16,temp+=16)

{

endidx = temp+15;

if(endidx > end)

endidx = end;

//print the memory addr of the start of the line

PrintHex(curridx, endidx);

//print the hexadecimal numbers

printf("; ");

PrintAsc(curridx, endidx);

//print the ascii codes after the hexadecimal part

printf("\n");

curridx = curridx/16\*16;

//at the end of the line

}

\*addr = end;

return;

}

void PrintHex(int start, int end)

//prints the hexadecimal parts of the line

{

int curridx=0;

int hchk=65536;

int temp=start/16\*16;

printf("%05X ",temp);

if(start%16)

//when start is not the first index of the line

{

//when start is not divided by 16

for(curridx = start/16\*16; curridx < start; ++curridx)

//printing from the start of the line, if it is not in the range of start to end, print spaces

{

printf(" ");

}

for(; curridx<=end; ++curridx)

{

printf("%02X ",mem[curridx]);

}

for(;curridx<temp+16;curridx++)

//when the end index is not the end of the line, print spaces until the end of the line

{

printf(" ");

}

return;

}

else

//when start is the first index of the line

{

for(curridx = start; curridx<=end;++curridx)

//print hexadecimals until the end index

{

printf("%02X ",mem[curridx]);

}

if(curridx%16)

{

//when end is not the end of the line

for(temp=curridx/16\*16+16;curridx<temp;++curridx)

{

printf(" ");

}

}

return;

}

return;

}

void PrintAsc(int start, int end)

{

int curridx=0;

int temp=start/16\*16;

if(start%16)

{

//when start is not divided by 16

for(curridx = start/16\*16; curridx < start; ++curridx)

//print '.' for memory not in range

{

printf(".");

}

for(; curridx<=end; ++curridx)

//print the ascii codes from start to end

{

printf("%c",mem[curridx]);

}

for(;curridx<temp+16;curridx++)

{

printf(".");

}

return;

}

else

{

//when start is the first index of the line

for(curridx = start; curridx<=end;++curridx)

{

printf("%c",mem[curridx]);

}

if(curridx%16)

{

//when end is not the end of the line

for(temp=curridx/16\*16+16;curridx<temp;++curridx)

{

printf(".");

}

}

return;

}

return;

}

void EditMem(int idx, int val)

{

if((idx<0)||(idx>1048575))

{

printf("Wrong range of memory acces detected!\n");

return;

}

mem[idx] = val;

//error is checked before entering this function, so set the memory at the index to val

return;

}

void FillMem(int idxs, int idxe, int val)

//similar to function EditMem but edits several index of memory

{

int i=0;

if((idxs<0)||(idxs>1048575))

{

printf("Wrong range of memory access detected!\n");

return;

}

if((idxe<0)||(idxe>1048575))

{

printf("Wrong range of memory access detected!\n");

return;

}

for(i=idxs;i<=idxe;++i)

{

mem[i] = val;

}

return;

}

void CreateSym(char\* newsubr, int newline)

{

SYM\* currnode = symboltable;

SYM\* newnode = (SYM\*)malloc(sizeof(SYM));

newnode->loc = newline;

strcpy(newnode->subr,newsubr);

newnode->link = NULL;

if(currnode == NULL)

{

symboltable = newnode;

return;

}

if(strcmp(currnode->subr,newsubr)>0)

{

symboltable = newnode;

newnode->link = currnode;

return;

}

while((currnode->link != NULL)&&(strcmp(currnode->link->subr,newsubr)<0))

{

currnode = currnode -> link;

}

newnode->link = currnode->link;

currnode->link = newnode;

return;

}

void ResetSym()

{

SYM\* deletenode = symboltable;

while(symboltable != NULL)

{

deletenode = symboltable;

symboltable = symboltable->link;

deletenode->link = NULL;

free(deletenode);

}

return;

}

void CreateHash()

{

int idx=0, code=0;

char buf[100];

//buf saves the opcode table line by line

//saves opcode of the line

char iname[6];

//saves instruction name of the line

int format;

//saves format of the line

HASH\* newnode = malloc(sizeof(HASH));

HASH\* currnode=NULL;

hashtable = (HASH\*\*)malloc(sizeof(HASH\*)\*20);

//create HASH\* array size of 20

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

while(1)

{

newnode = malloc(sizeof(HASH));

if(fgets(buf,100,fp) != NULL)

{

sscanf(buf,"%X%s%x",&code,iname,&format);

//completed input

}

else

{

fclose(fp);

return;

}

strcpy(newnode->opinstruction,iname);

newnode->opcode=code;

newnode->format=format;

newnode->link = NULL;

//newnode initialization complete

idx = HashFunc(iname);

currnode = hashtable[idx];

if(currnode == NULL)

{

//when certain hashtable index is an empty linked list. newnode becomes the head of the list

hashtable[idx] = newnode;

continue;

}

while(currnode->link != NULL)

{

currnode = currnode->link;

}

currnode->link = newnode;

//link the new node at the end of the linked list

}

fclose(fp);

return;

}

int HashFunc(char\* input)

//returns the hash value of the string

{

int hash=0;

int i=0;

for(i=0;input[i]!='\0';++i)

{

hash += (int)input[i];

hash+=2;

//this functions's hash value is the sum of all the ascii codes in the string plus 2\*length of the string

}

hash%=20;

//hash value should not exceed 20 which is the maximum index of the hashtable

return hash;

}

int GetOp(char\* input)

//takes instruction name as a parameter and returns its opcode in integer

{

HASH\* currnode = NULL;

int index = HashFunc(input);

int opcode=0;

if(!strcmp(input,"RESW")||!strcmp(input,"RESB")||!strcmp(input,"START")||!strcmp(input,"BYTE")||!strcmp(input,"WORD")||!strcmp(input,"BASE")||!strcmp(input,"FIRST"))

//Cases where the line doesn't have an object code

{

return -1;

}

currnode = hashtable[index];

while(currnode!=NULL)

{

if(!strcmp(input,currnode->opinstruction))

break;

currnode = currnode->link;

}

if(currnode == NULL)

{

return -1;

}

opcode = currnode->opcode;

return opcode;

}

int GetForm(char\* input)

//takes instruction name as a parameter and returns its opcode in integer

{

HASH\* currnode = NULL;

int index = HashFunc(input);

int format=0;

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

{

return 4;

}

currnode = hashtable[index];

while(currnode!=NULL)

{

if(!strcmp(input,currnode->opinstruction))

break;

currnode = currnode->link;

}

if(currnode == NULL)

{

return 0;

}

format = currnode->format;

return format;

}

int GetSymLoc(char\* input)

{

SYM\* currnode=symboltable;

if((strlen(input) == 0)||(input[0] == 0))

{

return -1;

}

if(currnode == NULL)

{

return -1;

}

while(currnode != NULL)

{

if(!strcmp(currnode->subr,input))

{

return currnode->loc;

}

currnode = currnode->link;

}

printf("No symbol matches the instruction!\n");

return -1;

}

int PrintOp(char\* input)

{

HASH\* currnode = NULL;

int index = 0;

char iname[6];

sscanf(input,"opcode %s", iname);

//input an instruction name and save it on iname

index = HashFunc(iname);

//identical instruction will have identical hash value

currnode = hashtable[index];

while(currnode != NULL)

{

if(!strcmp(iname,currnode->opinstruction))

break;

//check if such instruction exists in this index of hashtable

currnode = currnode->link;

}

if(currnode == NULL)

{

printf("instruction not found!\n");

//when input does not match any instruction in the linked list print error message

return 0;

}

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

//print the right opcode for the instruction

return 1;

}

void PrintOpList()

{

int i=0;

HASH\* currnode=NULL;

for(;i<20;++i)

{

printf("%d : ",i);

//print the index of the hashtable

for(currnode=hashtable[i];currnode != NULL;currnode = currnode->link)

{

if(currnode->link != NULL)

{

printf("[%s,%X] -> ",currnode->opinstruction,currnode->opcode);

//when current node is not the end node print the node with arrow

}

else

{

printf("[%s,%X]",currnode->opinstruction,currnode->opcode);

//when current node is the end of the node print the node without an arrow

}

}

printf("\n");

}

}

void Pass1(char\* input)

{

FILE\* fp;

char\* buf1 = (char\*)malloc(sizeof(char)\*MAX);

int loc=0,len=0,format=0,wnum=0,opcode=0,errorflag=0;

char\* sub = malloc(sizeof(char)\*10);

char\* iname = malloc(sizeof(char)\*10);

char\* oper = malloc(sizeof(char)\*10);

char\* oper2 = malloc(sizeof(char)\*10);

fp = fopen(input,"r");

while(fgets(buf1,MAX,fp))

{

if(IsCap(buf1[0]))

//First letter is a capital. Which means the line is a subroutine

{

wnum=sscanf(buf1,"%s %s %s %s",sub,iname,oper,oper2);

buf1[strlen(buf1)-1]='\t';

//directives@@@

if((GetOp(iname)<0)&&(GetForm(iname)!=4))

{

format=0;

if(!strcmp(iname,"START"))

{

}

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

{

}

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

{

format = (StrToDec(oper)\*3);

}

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

{

format = StrToDec(oper);

}

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

{

}

else if(!strcmp(iname,"BYTE"))

{

if(oper[0] == 'X')

{

format=1;

}

if(oper[0] == 'C')

{

format = strlen(oper)-3;

}

}

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

{

}

len = format;

format =5;

}

//end of directives

else

{

if(wnum == 4)

//When input was a subroutine and 2 operands

{

opcode = GetOp(iname);

format = GetForm(iname);

}

else if(wnum == 3)

//a subroutine command with one operand

{

opcode = GetOp(iname);

format = GetForm(iname);

}

else if(wnum == 2)

{

opcode = GetOp(iname);

format = GetForm(iname);

}

else if(wnum == 1)

{

}

//format 4 disp calculation

//array x flag control

len = format;

}

CreateSym(sub,loc);

}

else if(buf1[0] == ' ')

{

wnum = sscanf(buf1,"%s %s %s",iname,oper,oper2);

if((GetOp(iname)<0)&&(GetForm(iname)!=4))

{

format=0;

if(!strcmp(iname,"START"))

{

}

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

{

}

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

{

format = (StrToDec(oper)\*3);

}

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

{

format = StrToDec(oper);

}

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

{

}

else if(!strcmp(iname,"BYTE"))

{

if(oper[0] == 'X')

{

format=1;

}

if(oper[0] == 'C')

{

format = strlen(oper)-3;

}

}

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

{

}

len = format;

format =5;

}

else

{

if(wnum == 3)

{

opcode = GetOp(iname);

format = GetForm(iname);

}

else if(wnum == 2)

{

opcode = GetOp(iname);

format = GetForm(iname);

}

else if(wnum == 1)

{

opcode = GetOp(iname);

format = GetForm(iname);

}

len = format;

}

}

loc += len;

}

fclose(fp);

return;

}

void Pass2(char \*input,int endloc)

{

FILE \*lstfp, \*objfp;

int wnum=0,loc=0,line=0,start=0,cflag=0,len=0,modcnt=0,endrecord=0,endflag=0,i=0;

char\* subr = malloc(sizeof(char)\*100);

char\* iname = malloc(sizeof(char)\*100);

char\* oper = malloc(sizeof(char)\*100);

char\* oper2 = malloc(sizeof(char)\*100);

char\* buf1 = malloc(sizeof(char)\*300);

char\* objcode = malloc(sizeof(char)\*100);

char\* trecord = malloc(sizeof(char)\*100);

int modlist[200];

strcat(input,".obj");

objfp = fopen(input,"w");

strchr(input,'.')[0] = '\0';

strcat(input,".lst");

lstfp = fopen(input,"r");

fgets(buf1,MAX,lstfp);

sscanf(buf1,"%d %X %s %s %X",&line,&loc,subr,iname,&start);

rewind(lstfp);

if(!strcmp(iname,"START"))

{

fprintf(objfp,"H%-6s%06X%06X\nT%06X",subr,start,endloc-start,start);

}

else

{

fprintf(objfp,"H 000000%X\nT000000",endloc-start);

}

cflag=0;

//set cflag to 1 to start from next line after header record

memset(trecord,'\0',100);

while(fgets(buf1,2\*MAX,lstfp))

{

memset(objcode,'\0',100);

memset(iname,'\0',100);

memset(oper,'\0',100);

memset(oper2,'\0',100);

wnum = sscanf(buf1,"%d %X\t",&line,&loc);

if(wnum < 2)

//line without loc. has no objectcode nor length. ex)110 . subroutine to read

{

continue;

}

//check first line @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

i=0;

while(buf1[i+11]!='\0')

{

buf1[i] = buf1[i+11];

i++;

}

buf1[i]='\0';

//scanf source statement and object code to remove loc and line

if(IsCap(buf1[0]))

{

wnum = sscanf(buf1,"%s %s %s %s %s",subr,iname,oper,oper2,objcode);

//check rsub here first

if(!strcmp(iname,"RSUB"))

{

strcpy(objcode,oper);

}

if(wnum == 3)

//a subroutine directive without objectcode

{

}

else if(wnum == 4)

//a subroutine with one operand and objectcode

{

strcpy(objcode,oper2);

}

else if(wnum == 5)

//a subroutine with two operand and objectcode

{

}

}

else if(buf1[0] == ' ')

{

wnum = sscanf(buf1,"%s %s %s %s",iname,oper,oper2,objcode);

//check rsub here first

if(!strcmp(iname,"RSUB"))

{

strcpy(objcode,oper);

}

if(wnum == 2)

//instruction with operand without opcde ex) resw 1

{

}

else if(wnum == 3)

//instruction with operand and objcode but not a subroutine

{

strcpy(objcode,oper2);

}

else if(wnum == 4)

//instruction with 2 operands and an objcode but not a subroutine

{

}

}

if((iname[0] == '+')&&(oper[0] != '#'))

{

modlist[modcnt] = loc;

modcnt++;

}

if(strlen(objcode)!=0)

{

if((len+strlen(objcode)>60)||cflag==1)

{

fprintf(objfp,"%02X%s\nT%06X",len/2,trecord,loc);

strcpy(trecord,objcode);

len=strlen(objcode);

}

else

{

len += strlen(objcode);

strcat(trecord,objcode);

}

if(!endflag)

{

endflag=1;

endrecord = loc;

}

cflag=0;

}

loc+= strlen(objcode)/2;

if((!strcmp(iname,"RESW")||!strcmp(iname,"RESB"))&&cflag==0)

{

cflag=1;

}

}

fprintf(objfp,"%02X%s",len/2,trecord);

if(modcnt>0)

{

i=0;

while(i<modcnt)

{

fprintf(objfp,"\nM%06X05",modlist[i]+1);

i++;

}

}

fprintf(objfp,"\nE%06X",endrecord);

fclose(lstfp);

fclose(objfp);

}

printf("sicsim> ");

fgets(input,MAX,stdin);

input[strlen(input)-1] = '\0';

//changes the last index from '\n' to '\0'. Enables it to be used in strcmp and check it's end

if(!strcmp(input,instruction[0])||!strcmp(input,instruction[10]))

//compares the input and instruction to determine the operation. Either two kinds of input is valid for 'help' instruction

{

InputHistory(&head, input);

//When Valid instruction was input, saves the line in the history lists.

PrintInstruction();

//Operation for 'help'

}

else if(!strcmp(input,instruction[1])||!strcmp(input,instruction[11]))

{

InputHistory(&head, input);

PrintDir();

//Operation for 'dir'

}

else if(!strcmp(input,instruction[3])||!strcmp(input,instruction[13]))

{

InputHistory(&head, input);

PrintHistory(head);

//Operation for 'history'

}

else if(!strncmp(input,instruction[4],4)||!strncmp(input,instruction[14],2))

//for dump, edit and fill instruction, program needs to compare first few characters only. Therefore used strncmp.

{

if(IsFormal(&num1,&num2,&addr,input))

//'dump' instruction has several forms. IsFormal function checks if each of the instruction is valid and returns 1 for valid ones, 0 for wrong ones.

{

InputHistory(&head,input);

//Only valid inputs will be saved in the history.

}

}

else if(!strncmp(input, instruction[5],4)||!strncmp(input,instruction[15],1))

{

if(IsFormalE(&num1, &num2, input))

//Similar kind to IsFormal function. Used for Edit instruction.

{

InputHistory(&head,input);

}

}

else if(!strncmp(input, instruction[6],4)||!strncmp(input,instruction[16],1))

{

if(IsFormalF(&num1, &num2, &num3, input))

{

InputHistory(&head,input);

}

}

else if(!strcmp(input,instruction[7]))

{

memset(mem,0,sizeof(char)\*1048576);

//initializes the memory's value to 0. 1048576 is the decimal value of hexadecimal FFFFF.

InputHistory(&head, input);

}

else if(!strncmp(input,instruction[8],7))

{

if(PrintOp(input))

InputHistory(&head, input);

}

else if(!strcmp(input, instruction[9]))

{

PrintOpList();

InputHistory(&head, input);

}

else if(!strncmp(input, instruction[17],9))

{

if(!IsAssemble(input))

//take file name input in the function and determine.

{

printf("File unavailable!\n");

continue;

}

InputHistory(&head, input);

}

else if(!strncmp(input, instruction[18], 5))

//compares till space

{

if(PrintFile(input))

{

InputHistory(&head, input);

}

else

{

printf("File with such name does not exist\n");

}

}

else if(!strcmp(input, instruction[19]))

{

PrintSymbol();

InputHistory(&head, input);

}

}while((strcmp(input, "quit")&&strcmp(input, "q")));

//terminates the program when input was q or quit

return 1;

}

void PrintInstruction()

//Prints all instructions that can be done in the program.

{

printf("h[elp]\nd[ir]\nq[uit]\nhi[story]\ndu[mp] [start, end]\ne[dit] address, value\nf[ill] start, end, value\nreset\nopcode mnemonic\nopcodelist\nassemble filename\ntype filename\nsymbol\n");

return;

}

int IsAssemble(char\* input)

{

int i=0, j=0,opcode=0,niflag=0,xbpeflag=0,disp=0, format=0,errorflag=0;

int line=0, loc=0, bloc=0, byte=0,reg1=0,reg2=0;

int len=0;

char\* fname = malloc(sizeof(char)\*20);

char\* buf1=malloc(sizeof(char)\*100);

char\* buf2=malloc(sizeof(char)\*20);

char\* sub = malloc(sizeof(char)\*7);

//sub for subroutine

char \*iname = malloc(sizeof(char)\*7);

//iname for instruction name

char \*oper = malloc(sizeof(char)\*20);

//oper for operand

char\* oper2=malloc(sizeof(char)\*20);

FILE\* fp, \*lstfp, \*objfp;

int wnum=0;

ResetSym();

//wnum for word number(number of words in a single line)

sscanf(input,"%s %s",buf1,fname);

for(i=0;fname[i] != '.';++i)

//find the name part of file, ex) 2\_5.asm-> 2\_5

{

if(fname[i] == '\0')

{

printf("Source file extension not available!\n");

return 0;

}

}

if(strcmp(".asm",fname+i))

{

printf("Source file extension not available!\n");

return 0;

}

Pass1(fname);

if(!(fp = fopen(fname,"r")))

{

printf("No existing file named %s\n",fname);

return 0;

}

fname[i] = '\0';

strcat(fname,".lst");

if(!(lstfp = fopen(fname,"w")))

{

printf("\nFailed to create a file %s\n",fname);

return 0;

}

printf("\t\toutput file : [%s]",fname);

fname[i] = '\0';

strcat(fname,".obj");

if(!(objfp = fopen(fname,"w")))

{

printf("\nFailed to create a file %s\n",fname);

return 0;

}

printf(", [%s]\n",fname);

fname[i] = '\0';

//completed printing output file

while(fgets(buf1,MAX,fp))

{

line+=5;

if(sscanf(buf1,"%s",buf2)<1)

{

break;

}

fprintf(lstfp,"%5d\t",line);

if(buf1[0] == '.')

{

fprintf(lstfp,"%s",buf1);

}

else

{

if(IsCap(buf1[0]))

//First letter is a capital. Which means the line is a subroutine

{

wnum=sscanf(buf1,"%s %s %s %s",sub,iname,oper,oper2);

buf1[strlen(buf1)-1]='\t';

//directives@@@

if((GetOp(iname)<0)&&(GetForm(iname)!=4))

{

format=0;

if(!strcmp(iname,"START"))

{

fprintf(lstfp,"%04X\t%s\n",loc,buf1);

}

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

{

fprintf(lstfp,"\t%s\n",buf1);

}

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

{

fprintf(lstfp,"%04X\t%s\n",loc,buf1);

format = (StrToDec(oper)\*3);

}

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

{

fprintf(lstfp,"%04X\t%s\n",loc,buf1);

format = StrToDec(oper);

}

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

{

fprintf(lstfp,"%04X\t%s%06X\n",loc,buf1,StrToDec(oper));

}

else if(!strcmp(iname,"BYTE"))

{

fprintf(lstfp,"%04X\t%s",loc,buf1);

if(oper[0] == 'X')

{

sscanf(oper+2,"%X",&byte);

format=1;

fprintf(lstfp,"%02X\n",byte);

}

else if(oper[0] == 'C')

{

format = strlen(oper)-3;

i=0;

while(oper[i+2]!='\0')

{

oper[i] = oper[i+2];

i++;

}

oper[i]='\0';

for(j=0;oper[j]!='\0';++j)

{

if(IsCap(oper[j]))

{

fprintf(lstfp,"%02X",(int)oper[j]);

}

}

fprintf(lstfp,"\n");

}

else

{

errorflag=1;

}

}

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

{

fprintf(lstfp,"\t%s\n",buf1);

}

else

{

errorflag =1;

}

len = format;

format =5;

}

//end of directives

else

{

fprintf(lstfp,"%04X\t",loc);

fprintf(lstfp,"%s",buf1);

if(wnum == 4)

//When input was a subroutine and 2 operands

{

opcode = GetOp(iname);

format = GetForm(iname);

if(GetCom(oper))

{

strchr(oper,',')[0] = '\0';

}

}

else if(wnum == 3)

//a subroutine command with one operand

{

opcode = GetOp(iname);

format = GetForm(iname);

if(GetCom(oper))

{

strchr(oper,',')[0] = '\0';

}

}

else if(wnum == 2)

{

opcode = GetOp(iname);

format = GetForm(iname);

if(GetCom(oper))

{

strchr(oper,',')[0] = '\0';

}

}

else if(wnum == 1)

{

}

//format 4 disp calculation

len = format;

}

}

else if(buf1[0] == ' ')

{

wnum = sscanf(buf1,"%s %s %s",iname,oper,oper2);

buf1[strlen(buf1)-1]='\t';

if((GetOp(iname)<0)&&(GetForm(iname)!=4))

{

format=0;

if(!strcmp(iname,"START"))

{

fprintf(lstfp,"%04X\t%s\n",loc,buf1);

}

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

{

fprintf(lstfp,"\t%s",buf1);

}

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

{

fprintf(lstfp,"%04X%s\n",loc,buf1);

format = (StrToDec(oper)\*3);

}

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

{

fprintf(lstfp,"%04X%s\n",loc,buf1);

format = StrToDec(oper);

}

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

{

fprintf(lstfp,"%04X\t%s%06X\n",loc,buf1,StrToDec(oper));

}

else if(!strcmp(iname,"BYTE"))

{

fprintf(lstfp,"%04X%s",loc,buf1);

if(oper[0] == 'X')

{

sscanf(oper+2,"%X",&byte);

format=1;

fprintf(lstfp,"%02X\n",byte);

}

else if(oper[0] == 'C')

{

format = strlen(oper)-3;

i=0;

while(oper[i+2]!='\0')

{

oper[i] = oper[i+2];

i++;

}

oper[i]='\0';

for(j=0;oper[j]!='\0';++j)

{

if(IsCap(oper[j]))

{

fprintf(lstfp,"%02X",(int)oper[j]);

}

}

fprintf(lstfp,"\n");

}

else

{

errorflag=1;

}

}

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

{

fprintf(lstfp,"\t%s\n",buf1);

}

else

{

errorflag=1;

}

len = format;

format =5;

}

else

{

fprintf(lstfp,"%04X\t",loc);

fprintf(lstfp,"%s",buf1);

if(wnum == 3)

{

opcode = GetOp(iname);

format = GetForm(iname);

if(GetCom(oper))

{

strchr(oper,',')[0] = '\0';

}

}

else if(wnum == 2)

{

opcode = GetOp(iname);

format = GetForm(iname);

if(GetCom(oper))

{

strchr(oper,',')[0] = '\0';

}

}

else if(wnum == 1)

{

opcode = GetOp(iname);

format = GetForm(iname);

}

len = format;

}

}

else

{

errorflag = 1;

}

if(oper[0] == '@')

{

niflag = 2;

oper = oper + 1;

}

else if(oper[0] == '#')

{

niflag = 1;

oper = oper + 1;

}

else

{

niflag = 3;

}

//@@@@@@@@@@@@@@niflag=0 case consider???

//if base register changed, save its value

if(!strcmp(iname,"LDB"))

{

bloc = GetSymLoc(oper);

}

//starting disp calculation

loc += len;

if(!strcmp(iname,"RSUB"))

{

fprintf(lstfp,"4F0000\n");

continue;

}

switch (format)

{

//fprint according to the format

case 1:

//format1

fprintf(lstfp,"\n");

break;

case 2:

//format2

fprintf(lstfp,"%02X%01X",opcode,GetReg(oper));

if(GetReg(oper2)>0)

{

fprintf(lstfp,"%1X\n",GetReg(oper2));

}

else

{

fprintf(lstfp,"0\n");

}

break;

case 3:

//format3

//flag set

if(!strcmp(oper2,"X"))

//b flag set ex) SBUF STCH BUFFER, X

{

xbpeflag += FlagX;

}

if(IsNum(oper[0]))

{

disp = StrToDec(oper);

}

else

{

disp = GetSymLoc(oper);

{

if((disp - loc) > 2048 || (disp-loc) < -2048)

//out of pc relative range. use base register

{

xbpeflag += FlagB;

disp -= bloc;

}

else

{

//in pc relative range

xbpeflag += FlagP;

disp -= loc;

}

}

}

disp = disp&4095;

//cut unwanted bits

fprintf(lstfp,"%02X%01X%03X\n",opcode+niflag,xbpeflag,disp);

break;

case 4:

//format(4

opcode = GetOp(iname+1);

xbpeflag += FlagE;

//set e flag

if(IsNum(oper[0]))

{

disp = StrToDec(oper);

}

else

{

if((disp = GetSymLoc(oper)) == -1)

{

errorflag=1;

}

}

fprintf(lstfp,"%02X%01X%05X\n",opcode+niflag,xbpeflag,disp);

break;

}

niflag = 0;

xbpeflag = 0;

format=0;

len=0;

memset(oper,0,strlen(oper));

memset(oper2,0,strlen(oper2));

}

if(errorflag)

{

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

ResetSym();

fclose(lstfp);

fclose(objfp);

fclose(fp);

strcat(fname,".obj");

remove(fname);

fname[i] = '\0';

strcat(fname,".lst");

remove(fname);

return 0;

}

}

fclose(lstfp);

fclose(objfp);

fclose(fp);

Pass2(fname,loc);

return 1;

}

void PrintSymbol()

{

SYM\* currnode = symboltable;

if(currnode == NULL)

{

printf("Empty symbol node!\n");

return;

}

while(currnode != NULL)

{

printf("\t%s\t%04X\n",currnode->subr,currnode->loc);

currnode = currnode -> link;

}

return;

}

int PrintFile(char\* input)

{

FILE\* fp;

char\* buf1 = malloc(sizeof(char)\*MAX);

char\* fname = malloc(sizeof(char)\*MAX);

int wnum=0;

memset(buf1,0,MAX);

sscanf(input,"%s %s",buf1,fname);

if(fp = fopen(fname,"r"))

{

while(fgets(buf1,MAX-1,fp))

{

printf("%s",buf1);

}

}

else

{

printf("File %s unavailable access for reading!\n",input);

free(buf1);

free(fname);

return 0;

}

printf("\n");

free(buf1);

free(fname);

fclose(fp);

return 1;

}

void PrintDir()

//Prints all the files in the current directory. prints different characters for different types of files.

{

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

//directory pointer is used to save names and types of the files in the directory

struct dirent\* dir;

//dirent\* structure is used to check whether there is files left to check or not

struct stat filestat;

//file's type is saved in filestat variable

while((dir=readdir(dname)) != NULL)

//When dir is NULL, there is no more file to check.

{

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

//print file's name

stat(dir->d\_name,&filestat);

//stat saves file's name in variable dir, and saves file type in filestat

if(S\_ISDIR(filestat.st\_mode))

//Checks if the current file is a directory

{

printf("/");

}

else if(filestat.st\_mode&S\_IEXEC)

//Checks if the file is executable.

{

printf("\*");

}

printf("\t");

}

printf("\n");

closedir(dname);

return;

}

void InputHistory(NODE\*\* head, char\* Input)

//Saves the input in a linked list. Validity of the input will be checked before operating this function

{

NODE\* newnode;

NODE\* currnode=\*head;

newnode = (NODE\*)malloc(sizeof(NODE));

//Allocate newnode a size of NODE

if(newnode == NULL)

//When malloc was unsuccessful print error message and return

{

printf("Error creating new node!\n");

return;

}

strcpy(newnode->InputString,Input);

//Copy the input to the newnode with strcpy

newnode->link = NULL;

//initializes the next link to NULL

if(\*head == NULL)

{

\*head = newnode;

//When the linked list is empty, set newnode as the first node. therefore, newnode becomes the head of the list.

return;

}

while(currnode->link != NULL)

{

currnode = currnode -> link;

//Find the last node of the linked list.

}

currnode->link = newnode;

//Set the last node as newnode by setting the last node's link as the newnode

return;

}

void PrintHistory(NODE\* head)

{

int count=0;

NODE\* currnode = head;

if(head == NULL)

{

printf("list is empty!\n");

return;

}

do

{

printf("%d\t%s\n",++count,currnode->InputString);

currnode = currnode->link;

//Print the current node's string until you find the last node

}

while(currnode != NULL);

//when you find the NULL pointer, it means there is no more nodes to go through

return;

}

int IsIname(char\* input)

{

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

int index=0;

HASH\* currnode=NULL;

index = HashFunc(input);

//identical instruction will have identical hash value

currnode = hashtable[index];

while(currnode != NULL)

{

if(!strcmp(input,currnode->opinstruction))

break;

//check if such instruction exists in this index of hashtable

currnode = currnode->link;

}

if(currnode == NULL)

{

printf("instruction %s not found!\n",input);

//when input does not match any instruction in the linked list print error message

return 0;

}

printf("opcode for %s is %X\n",currnode->opinstruction,currnode->opcode);

//print the right opcode for the instruction

return 1;

}

int IsOper(char\* input)

{

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

return 1;

}

int IsNum(char inputchar)

//Returns 1 if a character is a ascii code for number 0 to 9

{

if('/'<inputchar && inputchar<':')

{

return 1;

}

return 0;

}

int IsCap(char inputchar)

//Returns 1 if a character is a Ascii code for capital letter

{

if('@'<inputchar && inputchar<'[')

{

return 1;

}

return 0;

}

int IsCapHex(char inputchar)

//Returns 1 if a character is a Ascii code for capital letter for Hexadecimal A to F

{

if('@'<inputchar && inputchar<'G')

{

return 1;

}

return 0;

}

int IsLow(char inputchar)

//Returns 1 if a character is a Ascii code for lowercase letter

{

if('`'<inputchar && inputchar<'{')

{

return 1;

}

return 0;

}

int IsHex(char\* input)

{

int i=0;

for(;input[i]!='\0';++i)

{

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

return 0;

}

return 1;

}

int StrToDec(char\* input)

{

int i=0;

int val=0;

while(input[i] != '\0')

{

val \*= 10;

val += (input[i]-'0');

++i;

}

return val;

}

int GetCom(char\* input)

//Returns the number of ','(commas) in a string

{

int i=0;

int count=0;

for(i=0;input[i]!='\0';++i)

{

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

{

++count;

}

}

return count;

}

int GetReg(char\* oper)

{

if(!strcmp(oper,"A"))

{

return RegA;

}

else if(!strcmp(oper,"X"))

{

return RegX;

}

else if(!strcmp(oper,"L"))

{

return RegL;

}

else if(!strcmp(oper,"B"))

{

return RegB;

}

else if(!strcmp(oper,"S"))

{

return RegS;

}

else if(!strcmp(oper,"T"))

{

return RegT;

}

else if(!strcmp(oper,"F"))

{

return RegF;

}

else if(!strncmp(oper,"PC",2))

{

return RegPC;

}

else if(!strncmp(oper,"SW",2))

{

return RegSW;

}

return -1;

}

int IsFormal(int\* num1, int\* num2, int\* addr, char\* input)

{

char buf1[6],buf2[6],buf3[6],buf4[6],buf5[6];

//buf arrays are used to take strings to check errors

int n1=0,n2=0;

//n1 and n2 saves the number parts of the input

int wnum=0;

wnum=sscanf(input,"%s%X ,%X",buf1,&n1,&n2);

//wnum(wordnum) saves the number of words taken by sscanf. sscanf returns the number of words that are successfully saved

if( (n1<0) || (n2<0))

{

//negative input error check

return 0;

}

if((wnum == 3)&&(GetCom(input)==1))

//when 3 parts of arguments were passed, there should only be one comma

{

if(sscanf(input,"%s%s%s%s%s",buf1,buf2,buf3,buf4,buf5)>4)

//if the input contains unwanted string parts, return 0. Reason for 5 buffers is because hexadecimals can be considered as strings too

{

return 0;

}

if((n1 > n2) || (n2>1048575))

//When n1 is bigger than n2 or n2 is bigger than maximum index fail.

{

return 0;

}

else

{

PrintMem(n1,n2,addr);

//PrintMem will print from start(num1) to end(num2)

}

}

else if(wnum == 2&&(GetCom(input)==0))

//when only one number input present

{

if(sscanf(input,"%s%s%s",buf1,buf2,buf3)>2)

{

return 0;

}

if(!IsHex(buf2))

{

return 0;

}

if(n1>1048575)

{

//when n1 is bigger than max index fail.

return 0;

}

else

{

PrintMem(n1,n1+159,addr);

//No need to check n1+ 160 since printmem checks it

}

}

else if(wnum == 1&&(strchr(input,' ')==NULL))

//When only one word was input. 'dump' or 'du'. Then there should be no other words, therefore no spaces should exist.

{

PrintMem(\*addr+1,\*addr+160,addr);

}

else

//cases where no input was existent or 4 or more words were contained

{

return 0;

}

return 1;

}

int IsFormalE(int\* num1, int\* num2, char\* input)

//Operation for edit

{

int i=0;

int val=0;

if(input[1] == '\0')

{

//when user typed 'e' only

return 0;

}

if(!strncmp(input,instruction[5],4) && input[4] == '\0')

{

//when user typed 'edit' only

return 0;

}

if(!strncmp(input,instruction[5],4))

{

//when user typed 'edit' + sth

i=4;

}

else if(!strncmp(input,instruction[15],1))

{

//when user typed 'e' + sth

i=1;

}

else

{

return 0;

}

//error check for types like 'east' or 'edin'

if(input[i] != ' ')

{

return 0;

}

//input's index i is currently on index of ' ' after 'edit' or 'e'

while(input[i+1] == ' ')

{

//when you type multiple spaces after edit like 'e ' or 'edit '

++i;

}

if(!(IsNum(input[i+1])||IsCapHex(input[i+1])))

{

//after 'edit ' or 'e ' must come a number or a capital

return 0;

}

++i;

while(IsNum(input[i])||IsCapHex(input[i]))

{

val\*=16;

if(IsNum(input[i]))

val += (input[i]-48);

else

val += (input[i]-55);

++i;

}

\*num1 = val;

val =0;

//end of num1 input

while(input[i] == ' ')

{

//when you type multiple spaces after num1 like 'e %d ' or 'edit %d '

++i;

}

if(input[i] != ',')

{

return 0;

}

while(input[i+1] == ' ')

{

//when you type multiple spaces after comma like 'e %d, ' or 'edit %d, '

++i;

}

if(!(IsNum(input[i+1])||IsCapHex(input[i+1])))

{

//when the first letter after 'edis %d, ' is not a number or capital letter, exception error

return 0;

}

++i;

while(IsNum(input[i])||IsCapHex(input[i]))

{

val\*=16;

if(IsNum(input[i]))

val += (input[i]-48);

else

val += (input[i]-55);

++i;

}

\*num2 = val;

val=0;

//end of num2 input

if(input[i] == '\0')

{

//when user typed 'edit %d, %d' or 'e %d, %d'

if(\*num1 > \*num2)

{

return 0;

}

EditMem(\*num1, \*num2);

//EditMem will Edit value of address in num1 into value of num2

return 1;

}

return 0;

}

int IsFormalF(int\* num1, int\* num2, int\* num3, char\* input)

{

int i=0;

int val=0;

if(input[1] == '\0')

{

//when user typed 'f' only

return 0;

}

if(!strncmp(input,instruction[6],4) && input[4] == '\0')

{

//when user typed 'fill' only

return 0;

}

if(!strncmp(input,instruction[6],4))

{

//when user typed 'fill' + sth

i=4;

}

else if(!strncmp(input,instruction[16],1))

{

//when user typed 'f' + sth

i=1;

}

else

{

return 0;

}

//error check for types like 'finn' or 'fire'

if(input[i] != ' ')

{

return 0;

}

//input's index i is currently on index of ' ' after 'fill' or 'f'

while(input[i+1] == ' ')

{

//when you type multiple spaces after fill like 'f ' or 'fill '

++i;

}

if(!(IsNum(input[i+1])||IsCapHex(input[i+1])))

{

//after 'fill ' or 'f ' must come a number or a capital

return 0;

}

++i;

while(IsNum(input[i])||IsCapHex(input[i]))

{

val\*=16;

if(IsNum(input[i]))

val += (input[i]-48);

else

val += (input[i]-55);

++i;

}

\*num1 = val;

val =0;

//end of num1 input

while(input[i] == ' ')

{

//when you type multiple spaces after num1 like 'f %d ' or 'fill %d '

++i;

}

if(input[i] != ',')

{

return 0;

}

while(input[i+1] == ' ')

{

//when you type multiple spaces after comma like 'f %d, ' or 'fill %d, '

++i;

}

if(!(IsNum(input[i+1])||IsCapHex(input[i+1])))

{

//when the first letter after 'fill %d, ' is not a number or capital letter, exception error

return 0;

}

++i;

while(IsNum(input[i])||IsCapHex(input[i]))

{

val\*=16;

if(IsNum(input[i]))

val += (input[i]-48);

else

val += (input[i]-55);

++i;

}

\*num2 = val;

val=0;

//end of num2 input

while(input[i] == ' ')

{

//when you type multiple spaces after num1 like 'f %d, %d ' or 'fill %d, %d '

++i;

}

if(input[i] != ',')

{

return 0;

}

while(input[i+1] == ' ')

{

//when you type multiple spaces after comma like 'f %d, %d, ' or 'fill %d, %d, '

++i;

}

if(!(IsNum(input[i+1])||IsCapHex(input[i+1])))

{

//when the first letter after 'fill %d, %d, ' is not a number or capital letter, exception error

return 0;

}

++i;

while(IsNum(input[i])||IsCapHex(input[i]))

{

val\*=16;

if(IsNum(input[i]))

val += (input[i]-48);

else

val += (input[i]-55);

++i;

}

\*num3 = val;

val=0;

//end of num3 input

if(input[i] == '\0')

{

//when user typed 'fill %d, %d, %d' or 'f %d, %d, %d'

if(\*num1 > \*num2)

{

return 0;

}

FillMem(\*num1, \*num2, \*num3);

//FillMem will Fill values of address in num1 to num2 into value of num3

return 1;

}

return 0;

}

void PrintMem(int start, int end, int\* addr)

//Prints the memory in range of start to end

{

int curridx=start;

int temp=start/16\*16;

//temp saves the starting address of the line

int endidx=0;

if(end > 1048575)

{

//when the print range exceeds the maximum memory index

end = 1048575;

}

for(curridx=start;curridx<=end;curridx+=16,temp+=16)

{

endidx = temp+15;

if(endidx > end)

endidx = end;

//print the memory addr of the start of the line

PrintHex(curridx, endidx);

//print the hexadecimal numbers

printf("; ");

PrintAsc(curridx, endidx);

//print the ascii codes after the hexadecimal part

printf("\n");

curridx = curridx/16\*16;

//at the end of the line

}

\*addr = end;

return;

}

void PrintHex(int start, int end)

//prints the hexadecimal parts of the line

{

int curridx=0;

int hchk=65536;

int temp=start/16\*16;

printf("%05X ",temp);

if(start%16)

//when start is not the first index of the line

{

//when start is not divided by 16

for(curridx = start/16\*16; curridx < start; ++curridx)

//printing from the start of the line, if it is not in the range of start to end, print spaces

{

printf(" ");

}

for(; curridx<=end; ++curridx)

{

printf("%02X ",mem[curridx]);

}

for(;curridx<temp+16;curridx++)

//when the end index is not the end of the line, print spaces until the end of the line

{

printf(" ");

}

return;

}

else

//when start is the first index of the line

{

for(curridx = start; curridx<=end;++curridx)

//print hexadecimals until the end index

{

printf("%02X ",mem[curridx]);

}

if(curridx%16)

{

//when end is not the end of the line

for(temp=curridx/16\*16+16;curridx<temp;++curridx)

{

printf(" ");

}

}

return;

}

return;

}

void PrintAsc(int start, int end)

{

int curridx=0;

int temp=start/16\*16;

if(start%16)

{

//when start is not divided by 16

for(curridx = start/16\*16; curridx < start; ++curridx)

//print '.' for memory not in range

{

printf(".");

}

for(; curridx<=end; ++curridx)

//print the ascii codes from start to end

{

printf("%c",mem[curridx]);

}

for(;curridx<temp+16;curridx++)

{

printf(".");

}

return;

}

else

{

//when start is the first index of the line

for(curridx = start; curridx<=end;++curridx)

{

printf("%c",mem[curridx]);

}

if(curridx%16)

{

//when end is not the end of the line

for(temp=curridx/16\*16+16;curridx<temp;++curridx)

{

printf(".");

}

}

return;

}

return;

}

void EditMem(int idx, int val)

{

if((idx<0)||(idx>1048575))

{

printf("Wrong range of memory acces detected!\n");

return;

}

mem[idx] = val;

//error is checked before entering this function, so set the memory at the index to val

return;

}

void FillMem(int idxs, int idxe, int val)

//similar to function EditMem but edits several index of memory

{

int i=0;

if((idxs<0)||(idxs>1048575))

{

printf("Wrong range of memory access detected!\n");

return;

}

if((idxe<0)||(idxe>1048575))

{

printf("Wrong range of memory access detected!\n");

return;

}

for(i=idxs;i<=idxe;++i)

{

mem[i] = val;

}

return;

}

void CreateSym(char\* newsubr, int newline)

{

SYM\* currnode = symboltable;

SYM\* newnode = (SYM\*)malloc(sizeof(SYM));

newnode->loc = newline;

strcpy(newnode->subr,newsubr);

newnode->link = NULL;

if(currnode == NULL)

{

symboltable = newnode;

return;

}

if(strcmp(currnode->subr,newsubr)>0)

{

symboltable = newnode;

newnode->link = currnode;

return;

}

while((currnode->link != NULL)&&(strcmp(currnode->link->subr,newsubr)<0))

{

currnode = currnode -> link;

}

newnode->link = currnode->link;

currnode->link = newnode;

return;

}

void ResetSym()

{

SYM\* deletenode = symboltable;

while(symboltable != NULL)

{

deletenode = symboltable;

symboltable = symboltable->link;

deletenode->link = NULL;

free(deletenode);

}

return;

}

void CreateHash()

{

int idx=0, code=0;

char buf[100];

//buf saves the opcode table line by line

//saves opcode of the line

char iname[6];

//saves instruction name of the line

int format;

//saves format of the line

HASH\* newnode = malloc(sizeof(HASH));

HASH\* currnode=NULL;

hashtable = (HASH\*\*)malloc(sizeof(HASH\*)\*20);

//create HASH\* array size of 20

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

while(1)

{

newnode = malloc(sizeof(HASH));

if(fgets(buf,100,fp) != NULL)

{

sscanf(buf,"%X%s%x",&code,iname,&format);

//completed input

}

else

{

fclose(fp);

return;

}

strcpy(newnode->opinstruction,iname);

newnode->opcode=code;

newnode->format=format;

newnode->link = NULL;

//newnode initialization complete

idx = HashFunc(iname);

currnode = hashtable[idx];

if(currnode == NULL)

{

//when certain hashtable index is an empty linked list. newnode becomes the head of the list

hashtable[idx] = newnode;

continue;

}

while(currnode->link != NULL)

{

currnode = currnode->link;

}

currnode->link = newnode;

//link the new node at the end of the linked list

}

fclose(fp);

return;

}

int HashFunc(char\* input)

//returns the hash value of the string

{

int hash=0;

int i=0;

for(i=0;input[i]!='\0';++i)

{

hash += (int)input[i];

hash+=2;

//this functions's hash value is the sum of all the ascii codes in the string plus 2\*length of the string

}

hash%=20;

//hash value should not exceed 20 which is the maximum index of the hashtable

return hash;

}

int GetOp(char\* input)

//takes instruction name as a parameter and returns its opcode in integer

{

HASH\* currnode = NULL;

int index = HashFunc(input);

int opcode=0;

if(!strcmp(input,"RESW")||!strcmp(input,"RESB")||!strcmp(input,"START")||!strcmp(input,"BYTE")||!strcmp(input,"WORD")||!strcmp(input,"BASE")||!strcmp(input,"FIRST"))

//Cases where the line doesn't have an object code

{

return -1;

}

currnode = hashtable[index];

while(currnode!=NULL)

{

if(!strcmp(input,currnode->opinstruction))

break;

currnode = currnode->link;

}

if(currnode == NULL)

{

return -1;

}

opcode = currnode->opcode;

return opcode;

}

int GetForm(char\* input)

//takes instruction name as a parameter and returns its opcode in integer

{

HASH\* currnode = NULL;

int index = HashFunc(input);

int format=0;

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

{

return 4;

}

currnode = hashtable[index];

while(currnode!=NULL)

{

if(!strcmp(input,currnode->opinstruction))

break;

currnode = currnode->link;

}

if(currnode == NULL)

{

return 0;

}

format = currnode->format;

return format;

}

int GetSymLoc(char\* input)

{

SYM\* currnode=symboltable;

if((strlen(input) == 0)||(input[0] == 0))

{

return -1;

}

if(currnode == NULL)

{

return -1;

}

while(currnode != NULL)

{

if(!strcmp(currnode->subr,input))

{

return currnode->loc;

}

currnode = currnode->link;

}

printf("No symbol matches the instruction!\n");

return -1;

}

int PrintOp(char\* input)

{

HASH\* currnode = NULL;

int index = 0;

char iname[6];

sscanf(input,"opcode %s", iname);

//input an instruction name and save it on iname

index = HashFunc(iname);

//identical instruction will have identical hash value

currnode = hashtable[index];

while(currnode != NULL)

{

if(!strcmp(iname,currnode->opinstruction))

break;

//check if such instruction exists in this index of hashtable

currnode = currnode->link;

}

if(currnode == NULL)

{

printf("instruction not found!\n");

//when input does not match any instruction in the linked list print error message

return 0;

}

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

//print the right opcode for the instruction

return 1;

}

void PrintOpList()

{

int i=0;

HASH\* currnode=NULL;

for(;i<20;++i)

{

printf("%d : ",i);

//print the index of the hashtable

for(currnode=hashtable[i];currnode != NULL;currnode = currnode->link)

{

if(currnode->link != NULL)

{

printf("[%s,%X] -> ",currnode->opinstruction,currnode->opcode);

//when current node is not the end node print the node with arrow

}

else

{

printf("[%s,%X]",currnode->opinstruction,currnode->opcode);

//when current node is the end of the node print the node without an arrow

}

}

printf("\n");

}

}

void Pass1(char\* input)

{

FILE\* fp;

char\* buf1 = (char\*)malloc(sizeof(char)\*MAX);

int loc=0,len=0,format=0,wnum=0,opcode=0,errorflag=0;

char\* sub = malloc(sizeof(char)\*10);

char\* iname = malloc(sizeof(char)\*10);

char\* oper = malloc(sizeof(char)\*10);

char\* oper2 = malloc(sizeof(char)\*10);

fp = fopen(input,"r");

while(fgets(buf1,MAX,fp))

{

if(IsCap(buf1[0]))

//First letter is a capital. Which means the line is a subroutine

{

wnum=sscanf(buf1,"%s %s %s %s",sub,iname,oper,oper2);

buf1[strlen(buf1)-1]='\t';

//directives@@@

if((GetOp(iname)<0)&&(GetForm(iname)!=4))

{

format=0;

if(!strcmp(iname,"START"))

{

}

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

{

}

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

{

format = (StrToDec(oper)\*3);

}

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

{

format = StrToDec(oper);

}

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

{

}

else if(!strcmp(iname,"BYTE"))

{

if(oper[0] == 'X')

{

format=1;

}

if(oper[0] == 'C')

{

format = strlen(oper)-3;

}

}

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

{

}

len = format;

format =5;

}

//end of directives

else

{

if(wnum == 4)

//When input was a subroutine and 2 operands

{

opcode = GetOp(iname);

format = GetForm(iname);

}

else if(wnum == 3)

//a subroutine command with one operand

{

opcode = GetOp(iname);

format = GetForm(iname);

}

else if(wnum == 2)

{

opcode = GetOp(iname);

format = GetForm(iname);

}

else if(wnum == 1)

{

}

//format 4 disp calculation

//array x flag control

len = format;

}

CreateSym(sub,loc);

}

else if(buf1[0] == ' ')

{

wnum = sscanf(buf1,"%s %s %s",iname,oper,oper2);

if((GetOp(iname)<0)&&(GetForm(iname)!=4))

{

format=0;

if(!strcmp(iname,"START"))

{

}

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

{

}

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

{

format = (StrToDec(oper)\*3);

}

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

{

format = StrToDec(oper);

}

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

{

}

else if(!strcmp(iname,"BYTE"))

{

if(oper[0] == 'X')

{

format=1;

}

if(oper[0] == 'C')

{

format = strlen(oper)-3;

}

}

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

{

}

len = format;

format =5;

}

else

{

if(wnum == 3)

{

opcode = GetOp(iname);

format = GetForm(iname);

}

else if(wnum == 2)

{

opcode = GetOp(iname);

format = GetForm(iname);

}

else if(wnum == 1)

{

opcode = GetOp(iname);

format = GetForm(iname);

}

len = format;

}

}

loc += len;

}

fclose(fp);

return;

}

void Pass2(char \*input,int endloc)

{

FILE \*lstfp, \*objfp;

int wnum=0,loc=0,line=0,start=0,cflag=0,len=0,modcnt=0,endrecord=0,endflag=0,i=0;

char\* subr = malloc(sizeof(char)\*100);

char\* iname = malloc(sizeof(char)\*100);

char\* oper = malloc(sizeof(char)\*100);

char\* oper2 = malloc(sizeof(char)\*100);

char\* buf1 = malloc(sizeof(char)\*300);

char\* objcode = malloc(sizeof(char)\*100);

char\* trecord = malloc(sizeof(char)\*100);

int modlist[200];

strcat(input,".obj");

objfp = fopen(input,"w");

strchr(input,'.')[0] = '\0';

strcat(input,".lst");

lstfp = fopen(input,"r");

fgets(buf1,MAX,lstfp);

sscanf(buf1,"%d %X %s %s %X",&line,&loc,subr,iname,&start);

rewind(lstfp);

if(!strcmp(iname,"START"))

{

fprintf(objfp,"H%-6s%06X%06X\nT%06X",subr,start,endloc-start,start);

}

else

{

fprintf(objfp,"H 000000%X\nT000000",endloc-start);

}

cflag=0;

//set cflag to 1 to start from next line after header record

memset(trecord,'\0',100);

while(fgets(buf1,2\*MAX,lstfp))

{

memset(objcode,'\0',100);

memset(iname,'\0',100);

memset(oper,'\0',100);

memset(oper2,'\0',100);

wnum = sscanf(buf1,"%d %X\t",&line,&loc);

if(wnum < 2)

//line without loc. has no objectcode nor length. ex)110 . subroutine to read

{

continue;

}

//check first line

i=0;

while(buf1[i+11]!='\0')

{

buf1[i] = buf1[i+11];

i++;

}

buf1[i]='\0';

//scanf source statement and object code to remove loc and line

if(IsCap(buf1[0]))

{

wnum = sscanf(buf1,"%s %s %s %s %s",subr,iname,oper,oper2,objcode);

//check rsub here first

if(!strcmp(iname,"RSUB"))

{

strcpy(objcode,oper);

}

if(wnum == 3)

//a subroutine directive without objectcode

{

}

else if(wnum == 4)

//a subroutine with one operand and objectcode

{

strcpy(objcode,oper2);

}

else if(wnum == 5)

//a subroutine with two operand and objectcode

{

}

}

else if(buf1[0] == ' ')

{

wnum = sscanf(buf1,"%s %s %s %s",iname,oper,oper2,objcode);

//check rsub here first

if(!strcmp(iname,"RSUB"))

{

strcpy(objcode,oper);

}

if(wnum == 2)

//instruction with operand without opcde ex) resw 1

{

}

else if(wnum == 3)

//instruction with operand and objcode but not a subroutine

{

strcpy(objcode,oper2);

}

else if(wnum == 4)

//instruction with 2 operands and an objcode but not a subroutine

{

}

}

if((iname[0] == '+')&&(oper[0] != '#'))

{

modlist[modcnt] = loc;

modcnt++;

}

if(strlen(objcode)!=0)

{

if((len+strlen(objcode)>60)||cflag==1)

{

fprintf(objfp,"%02X%s\nT%06X",len/2,trecord,loc);

strcpy(trecord,objcode);

len=strlen(objcode);

}

else

{

len += strlen(objcode);

strcat(trecord,objcode);

}

if(!endflag)

{

endflag=1;

endrecord = loc;

}

cflag=0;

}

loc+= strlen(objcode)/2;

if((!strcmp(iname,"RESW")||!strcmp(iname,"RESB"))&&cflag==0)

{

cflag=1;

}

}

fprintf(objfp,"%02X%s",len/2,trecord);

if(modcnt>0)

{

i=0;

while(i<modcnt)

{

fprintf(objfp,"\nM%06X05",modlist[i]+1);

i++;

}

}

fprintf(objfp,"\nE%06X",endrecord);

fclose(lstfp);

fclose(objfp);

}

* 20151616.h

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <dirent.h>

#include <sys/stat.h>

#include <sys/types.h>

#define MAX 1000

char\* instruction[] = {"help", "dir", "quit", "history", "dump", "edit", "fill", "reset", "opcode ", "opcodelist", "h", "d", "q", "hi", "du", "e", "f", "assemble filename", "type filename", "symbol"};

typedef struct \_\_node

{

char InputString[MAX];

struct \_\_node\* link;

} NODE;

typedef struct HASH\_\_node

{

unsigned int opcode;

char opinstruction[6];

int format;

struct HASH\_\_node\* link;

} HASH;

typedef struct SYM\_\_node

{

unsigned int loc;

char subr[10];

struct SYM\_\_node\* link;

} SYM;

enum Reg{RegA=0,RegX=1,RegL=2,RegB=3,RegS=4,RegT=5,RegF=6,RegPC=8,RegSW=9};

enum Flags{FlagX=8,FlagB=4,FlagP=2,FlagE=1};

int IsAssemble(char\* input);

void PrintInstruction();

void PrintDir();

void PrintHistory(NODE\*);

void PrintSymbol();

void InputHistory(NODE\*\*, char\*);

int IsCap(char inputchar);

int IsNum(char inputchar);

int IsIname(char\* iname);

//returns 0 if iname is not an instruction. references opcode.txt

int IsOper(char\* oper);

//returns 0 if oper is not an operand. references symbol table

int IsFormal(int\* num1, int\* num2, int\* addr, char\* input);

//Checks if the input is in right form. If the form is not right, return 0. Depending on the form, calls the PrintMem in correct index

int IsFormalE(int\* num1, int\* num2, char\* input);

int IsFormalF(int\* num1, int\* num2, int\* num3, char\* input);

int PrintFile(char\* input);

void PrintMem(int start, int end, int\* addr);

//Prints the specific memory addr from start to end. Sets the addr to end index

void PrintHex(int start, int end);

void PrintAsc(int start, int end);

void EditMem(int idx, int val);

void FillMem(int idxs, int idxe, int val);

void CreateHash();

void CreateSym(char\* newsubr,int newline);

void ResetSym();

int HashFunc(char\* input);

int PrintOp(char\* input);

void PrintOpList();

void PrintTable();

int GetCom(char\* input);

int GetOp(char\* input);

int GetForm(char \*input);

int GetReg(char\* input);

int GetSymLoc(char\* input);

//returns the location of the symbol. if input is not in the symboltable return -1

int StrToDec(char\* input);

void Pass1(char\* input);

void Pass2(char\* input,int endloc);

unsigned char mem[1048576];

HASH\*\* hashtable;

SYM\* symboltable = NULL;