과목 명: 시스템프로그래밍

담당 교수 명: 박 운 상

<<Project 3>>

**서강대학교 컴퓨터학과**

**[20161634]**

**[장수길]**

목 차

1. **프로그램 개요**
2. **프로그램 설명**

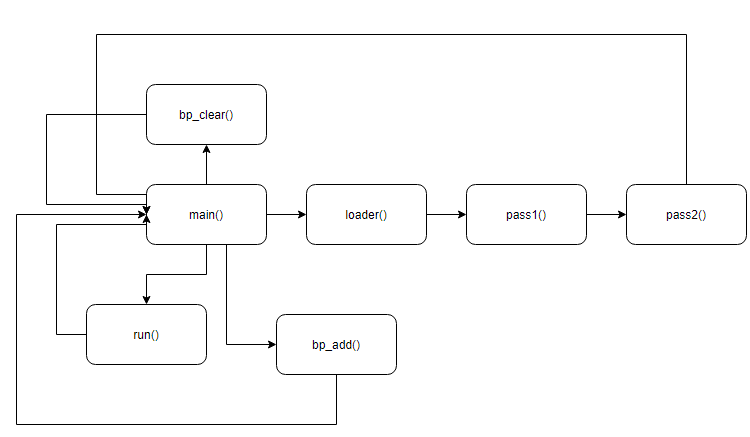
**2.1** 프로그램 흐름도

1. **구조체 및 헤더 정의**
2. **모듈 정의**
   1. 모듈 이름: void print\_regs()
      1. 기능
      2. 사용 변수
   2. 모듈 이름: void print\_bp()
      1. 기능
      2. 사용 변수
   3. 모듈 이름: void clear\_bp()
      1. 기능
      2. 사용 변수
   4. 모듈 이름: void add\_bp(int new\_bp)
      1. 기능
      2. 사용 변수
   5. 모듈 이름: int decHex(const char \*hex)
      1. 기능
      2. 사용 변수
   6. 모듈 이름: bool file\_exists(char \*name)
      1. 기능
      2. 사용 변수
   7. 모듈 이름: void destroy\_EST(struct ESTNODE \*\*est\_head)
      1. 기능
      2. 사용 변수
   8. 모듈 이름: void print\_EST(struct ESTNODE \*\*est\_head)
      1. 기능
      2. 사용 변수
   9. 모듈 이름: int EST\_search(const char \*name, int type, struct ESTNODE \*\*est\_head)
      1. 기능
      2. 사용 변수
   10. 모듈 이름: void EST\_add(char \*csect, char \*symbol, int address, int length, int type, struct ESTNODE \*\*est\_head)
       1. 기능
       2. 사용 변수
   11. 모듈 이름: int pass1(char \*filename)
       1. 기능
       2. 사용 변수
   12. 모듈 이름: int pass2(char \*filename)
       1. 기능
       2. 사용 변수
   13. 모듈 이름: int run(void)
       1. 기능
       2. 사용 변수
   14. 모듈 이름: int loader(char \*cmd)
       1. 기능
       2. 사용 변수
3. **전역 변수 정의**
4. **코드**
   1. **defines.h**
   2. **20161634.h**
   3. **20161634.c**
   4. **mem\_cmd.h**
   5. **mem\_cmd.c**
   6. **shell\_cmd.h**
   7. **shell\_cmd.c**
   8. **opcode.h**
   9. **opcode.c**
   10. **assembler.c**
   11. **assembler.h**
   12. **loader.c**
   13. **loader.h**
5. **프로그램 개요**

SIC/XE 머신 구현을 위한 마지막 단계이다.

주어진 obj 파일을 가상 메모리 위에 load하고 입력된 breakpoint에 따라서 obj파일을 실행해준다.

1. **프로그램 설명**
   1. **프로그램 흐름도**

  
<그림 1: 전체적인 구조도>

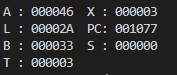
1. **구조체 및 헤더 정의**

이번 프로젝트 3에 새롭게 추가된 구조체는 다음과 같다.

|  |  |
| --- | --- |
| REGISTERS | A,X,L,PC,S,B,L,T 레지스터들을 저장한다. |
| ESTNODE | ESTAB의 노드 정보를 저장해준다. |

1. **모듈 정의**
   1. **모듈 이름 void print\_regs()**
      1. *기능*

레지스터들의 내용들을 다음과 같이 출력해준다.



* + 1. *사용 변수*

*없음*

* 1. **모듈 이름 void print\_bp()**
     1. *기능*

Breakpoint들을 모두 출력해준다.

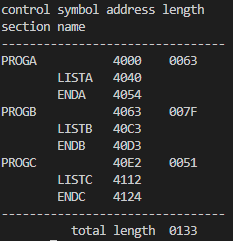
* + 1. *사용 변수*

없음

* 1. **모듈 이름 void clear\_bp()**
     1. *기능*설정한 breakpoint들을 모두 해제시켜준다.
     2. *사용 변수*없음
  2. **모듈 이름 void add\_bp()**
     1. *기능*새로운 breakpoint를 추가해준다. Realloc 과 Malloc을 사용하여 새로운 bp가 추가될경우 삽입하였다.
     2. *사용 변수*없음
  3. **모듈 이름 int decHex(const char \*hex)**
     1. *기능*3byte의 16진수를 받아 4byte의 16진수(현대 PC CPU 기준) 으로 변환시켜준다. 부호 변환 또한 이루어질 수 있다.
     2. *사용 변수*char neg[10], char pos[10]: 음수/양수일경우를 고려하여 변환시켜줄 때 저장해준다.
  4. **모듈 이름 bool file\_exists(char \*name)**
     1. *기능*

name에 해당되는 파일이 존재하는지 알려준다.

* + 1. *사용 변수*없음
  1. **모듈 이름 void destroy\_EST(struct ESTNODE \*\*est\_head)**
     1. *기능*ESTAB을 모두 해제시켜준다.
     2. *사용 변수*없음
  2. **모듈 이름 void print\_EST(struct ESTNODE \*\*est\_head)**
     1. *기능*

ESTAB의 내용을 다음과 같은 형식으로 출력한다.  


* + 1. *사용 변수*없음
  1. **모듈 이름 int EST\_search(const char \*name, int type, struct ESTNODE \*\*est\_head)**
     1. *기능*ESTAB을 검색하여 해당하는 Control section 또는 symbol의 주소값을 반환해준다. 없을 경우 -1을 반환한다.
     2. *사용 변수*
  2. **모듈 이름 void EST\_add(char \*csect, char \*symbol, int address, int length, int type, struct ESTNODE \*\*est\_head)**
     1. *기능*ESTAB에 control section 또는 symbol을 추가시켜준다.
     2. *사용 변수*
  3. **모듈 이름 int pass1(char \*filename)**
     1. *기능*Pass 1 알고리즘을 수행한다 ESTAB를 생성한다.
     2. *사용 변수*
  4. **모듈 이름 int pass2(char \*filename)**
     1. *기능*Pass2를 수행한다. 메모리에 object code를 load시켜준다.
     2. *사용 변수*

|  |  |
| --- | --- |
| int CSLTH | Control section의 길이를 저장 |
| int n | 임시 변수 |
| char buffer[100] | line을 parsing하기 위한 변수 |
| char temp[100] | 임시 변수 |
| int num | parsing을 위한 변수 |

* 1. **모듈 이름 int run(void)**
     1. *기능*로드된 메모리를 기반으로 프로그램을 실행시켜준다. 주어진 Breakpoint에 따라서 가장 가까운 breakpoint가 나올때까지 실행시켜주고, 프로그램의 끝까지 실행시킬 경우 레지스터들을 초기화시켜주고 PC값또한 초기화시켜준다.
     2. *사용 변수*

|  |  |
| --- | --- |
| int CSLTH | Control section의 길이를 저장 |
| int n | 임시 변수 |
| char buffer[100] | line을 parsing하기 위한 변수 |
| char temp[100] | 임시 변수 |
| int num | parsing을 위한 변수 |
| int sign | modification record의 부호를 저장 |
| int value | modification 을 위해 메모리를 저장 |

* 1. **모듈 이름 int loader(char \*cmd)**
     1. *기능*Pass1과 Pass2함수를 호출하여 프로그램을 로드시킨다.
     2. *사용 변수*

|  |  |
| --- | --- |
| bool break\_flag | bp를 만난 것을 표시 |
| bool prog\_end | 프로그램의 끝에 도달한 것을 표시 |
| int operand | opcode표에서의 m값 (operand) |
| bool PC\_edit | PC값을 변화시킨 것을 표시 |
| int opcode | opcode값을 저장 |
| int n,I,x,b,p,e | 각각 nixbpe 비트들을 저장 |
| int format | 명령어의 format을 저장 |
| int len | object code 한줄의 길이를 저장 |

1. **전역변수 정의**

|  |  |
| --- | --- |
| ESTNODE \*est\_head | ESTAB를 구성하는 연결리스트의 헤드 노드이다. |
| int csaddr | CSADDR값을 저장한다. |
| int execaddr | EXECADDR을 저장한다. |
| int proglength | 프로그램의 길이를 저장한다. |
| int bp\_num | Breakpoint의 개수를 저장한다. |
| struct REGISTERS reg | 레지스터들을 저장한다. |
| char REFTAB | REFTAB을 저장한다. |
| bool next\_CC | TD 명령을 수행하기 위해서 플래그를 저장한다. |

1. **코드**
   1. **defines.h**

#ifndef \_\_CHECK\_\_

#define \_\_CHECK\_\_

#include <stdarg.h>

#include <stdio.h>

#include <string.h>

#include <stdbool.h>

#include <dirent.h>

#include <sys/stat.h>

#include <ctype.h>

#include <stdlib.h>

#define MAX\_HASH 20

struct NODE{

    struct NODE \*next;

    char format[10];

    char cmd[100];

    int num;

};

#endif

* 1. **20161634.c**

#include "20161634.h"

#include "shell\_cmd.h"

#include "mem\_cmd.h"

#include "opcode.h"

#include "defines.h"

#include "assembler.h"

//head of linked list for history cmd

extern struct NODE \*head;

extern struct NODE \*symbol;

extern struct NODE \*last\_success\_symtab;

//hash table pointer

extern struct NODE \*hash[MAX\_HASH];

//works as 1MB virtual memory 1byte \* 2^20

unsigned char m[2<<20];

int dump\_adr=0;

int main(void){

    head=NULL;

    //resets virtual memory with value=

    memset(m, 0, sizeof(m));

    //reads from opcode.txt and initiates hash table

    init\_hash(hash);

    while(1){

        start();

    }

    return 0;

}

/\*---------------------------------------------------------------------\*/

/\*Function: hex\*/

/\*Purpose: reads a hexadecimal string and returns positive decimal value\*/

/\*Return value: if invalid -1 / if valid positive decimal value\*/

/\*---------------------------------------------------------------------\*/

int hex(const char\* num){

    if(strlen(num)==0) return -1;

    for(int i=0;i<strlen(num);i++){

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

            return -1;

        }

    }

    int ret;

    sscanf(num, "%x", &ret);

    return ret;

}

/\*---------------------------------------------------------------------\*/

/\*Function: argnum\*/

/\*Purpose: parses multi-word commands and calls appropriate functions depending on input\*/

/\*Return value: if invalid -1 / if valid 0 \*/

/\*---------------------------------------------------------------------\*/

int argnum(char\* cmd){

    char what[100];

    //variables for start end value address

    int s,e,val, adr;

    //variable to check arguments

    int args=1;

    //checks validity of input

    int space=0, comma=0;

    //sets flag according to command

    bool editflag=false, dumpflag=false, fillflag=false;

    bool spaceflag=false;

    s=e=val=adr=-1;

    //string to add to history list

    char history\_string[100];

    char op[100], mne[100], trash[100]; trash[0]='\0';

    int words=sscanf(cmd, "%s %s %s", op, mne, trash);

    //TWO WORDED INPUTS

    if(words==2 && trash[0]=='\0'){

        if(!strcmp(op, "opcode")){

            //ex: opcode add

            op\_cmd(mne, hash, &head); return 0;

        }

        if(!strcmp(op, "type")){

            type\_filename(mne, &head); return 0;

        }

        if(!strcmp(op, "assemble")){

            int ret=assemble(mne, &head);

            if(ret!=-1){

                sprintf(history\_string, "assemble %s", mne);

                history\_add(history\_string, &head);

            }

            return 0;

        }

    }

    //checks comma numbers

    for(int i=0;i<strlen(cmd);i++){

        if(!spaceflag && isspace(cmd[i])){

            space=i; spaceflag=true;

        }

        if(cmd[i]==',') comma++;

    }

    sscanf(cmd, "%s", what);

    char \*tokenize=cmd;

    tokenize+=space;

    char \*ptr;

    //decides command flags

    if(!strcmp("e", what)||!strcmp("edit", what)) editflag=true;

    if(!strcmp("du", what)||!strcmp("dump", what)) dumpflag=true;

    if(!strcmp("f", what)||!strcmp("fill", what)) fillflag=true;

    char hexStart[10], hexEnd[10],hexAddress[10], hexValue[10];

    //splits command by comma, and does boundary check and argument checks

    while((ptr=strsep(&tokenize,","))){

        int tempVal=hex(trim(ptr));

        //if invalid hexadecimal error

        if(tempVal==-1) return -1;

        if(editflag){

            //edit : check second argument range 00~FF

            //edit : first argument should be 00~FFFFF

            if(args==1 && 0<=tempVal && tempVal<=0xFFFFF){

                adr=tempVal; args++;

                strcpy(hexAddress, trim(ptr));

            }

            else if(args==2 && 0<=tempVal && tempVal<=0xFF){

                val=tempVal; args++;

                strcpy(hexValue, trim(ptr));

            }

            else return -1;

        }

        else if(dumpflag){

            //dump : all arguments should be 00~FFFFF

            if(0<=tempVal && tempVal<=0xFFFFF){

                if(args==1){

                    s=tempVal; args++;

                    strcpy(hexStart, trim(ptr));

                }

                else if(args==2){

                    e=tempVal; args++;

                    strcpy(hexEnd, trim(ptr));

                }

            }

            //if else invalid input ex: du 123A

            else return -1;

        }

        else if(fillflag){

            //fill : first and second argument (00~FFFFF) third argument (00~FF)

            //printf("tempval:%d\n", tempVal);

            if(args==1 && 0<=tempVal && tempVal<=0xFFFFF){

                s=tempVal; args++;

                strcpy(hexStart, trim(ptr));

            }

            else if(args==2 && 0<=tempVal && tempVal<=0xFFFFF){

                e=tempVal; args++;

                strcpy(hexEnd, trim(ptr));

            }

            else if(args==3 && 0<=tempVal && tempVal<=0xFF){

                val=tempVal; args++;

                strcpy(hexValue, trim(ptr));

            }

            else return -1;

        }

    }

    //command should be checked

    //all s,e,val should never be -1

    //comma number and argument number should match

    if(comma+2!=args) return -1;

    //call functions based on command and append to history list

    if(args==2 && (s!=-1) && (!strcmp(what, "du")||!strcmp(what,"dump")) ){

        sprintf(history\_string, "%s %s", what, hexStart);

        history\_add(history\_string, &head);

        dump(m,1,s); return 0;

    }

    if(args==3 && (s!=-1 && e!=-1) &&  (!strcmp(what, "du")||!strcmp(what,"dump"))){

        sprintf(history\_string, "%s %s, %s", what, hexStart,hexEnd);

        history\_add(history\_string, &head);

        dump(m,2,s,e); return 0;

    }

    if(args==4 && (s!=-1 && e!=-1 && val!=-1) && (!strcmp(what, "f")||!strcmp(what,"fill"))){

        sprintf(history\_string, "%s %s, %s, %s", what, hexStart,hexEnd,hexValue);

        history\_add(history\_string, &head);

        fill(m,s,e,val); return 0;

    }

    if(editflag && args==3 && (adr!=-1 && val!=-1)){

        sprintf(history\_string, "%s %s, %s", what, hexAddress, hexValue);

        history\_add(history\_string, &head);

        edit(m,adr, val); return 0;

    }

    return -1;

}

/\*---------------------------------------------------------------------\*/

/\*Function: start\*/

/\*Purpose: reads user input and handles single word inputs\*/

/\*Return: void\*/

/\*---------------------------------------------------------------------\*/

void start(void){

    printf("sicsim>");

    char buffer[MAX\_INPUT];

    if(fgets(buffer, sizeof(buffer), stdin)){

        size\_t len=strlen(buffer);

        if(len>0 && buffer[len-1]=='\n') buffer[len-1]='\0';

    }

    char \*cmd=trim(buffer);

    if(strlen(cmd)==0) return;

    //one word inputs (trivial) - call appropriate functions

    if(!strcmp(cmd, "help") || !strcmp(cmd, "h")){

        shell\_help();

        history\_add(cmd, &head); return;

    }

    if(!strcmp(cmd, "dir") || !strcmp(cmd, "d")){

        shell\_dir();

        history\_add(cmd, &head); return;

    }

    if(!strcmp(cmd, "quit") || !strcmp(cmd, "q")){

        exit(0);

    }

    if(!strcmp(cmd, "history") || !strcmp(cmd, "hi")){

        history\_add(cmd, &head);

        history\_print(&head); return;

    }

    if(!strcmp(cmd, "du")||!strcmp(cmd,"dump")){

        dump(m,1, dump\_adr); dump\_adr+=160;

        history\_add(cmd, &head); return;

    }

    if(!strcmp(cmd, "opcodelist")){

        history\_add(cmd, &head);

        op\_list(hash); return;

    }

    if(!strcmp(cmd, "reset")){

        history\_add(cmd, &head);

        reset(m); return;

    }

    if(!strcmp(cmd, "symbol")){

        history\_add(cmd, &head);

        symtab\_print(&last\_success\_symtab); return;

    }

    //memory commands with multi word inputs

    if(argnum(cmd)==-1){

        printf("invalid input\n"); return;

    }

}

**20161634.h**

#pragma once

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_INPUT 100

void start(void);

int argnum(char\* cmd);

int hex(const char\* num);

**assembler.c**

#include "assembler.h"

#include "opcode.h"

#include <inttypes.h>

#define MAX\_FILENAME 100

#define ERROR -1

#define MEMSET(string) memset(string, '\0', sizeof(string));

//head of linked list for history cmd

struct NODE \*head;

//head of linked list for last success symbol table

struct NODE \*last\_success\_symtab;

//hash table pointer

struct NODE \*hash[MAX\_HASH];

//head of linked list for current symbol table

struct NODE \*symbol;

FILE \*lst, \*obj, \*itm;

/\*---------------------------------------------------------------------\*/

/\*Function: file\_ext\_appender\*/

/\*Purpose: Get a file name and change the extension format with parameter\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void file\_ext\_appender(char\* filename, const char\* extension) {

    int flen = strlen(filename);

    for (int i = 0; i < 3; i++) {

        filename[flen - 1 - i] = '\0';

    }

    strcat(filename, extension);

}

/\*---------------------------------------------------------------------\*/

/\*Function: remove\_files\*/

/\*Purpose: remove filename.obj, filename.lst, filename.itm\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void remove\_files(const char \*filename){

    char fname[MAX\_FILENAME];

    int rem=0;

    strcpy(fname, filename);

    file\_ext\_appender(fname, "lst");

    rem=remove(fname);

    if(rem==-1) printf("deletion failed\n");

    file\_ext\_appender(fname, "obj");

    rem=remove(fname);

    if(rem==-1) printf("deletion failed\n");

    file\_ext\_appender(fname, "itm");

    rem=remove(fname);

    if(rem==-1) printf("deletion failed\n");

}

/\*---------------------------------------------------------------------\*/

/\*Function: creates\_files\*/

/\*Purpose: creates filename.obj, filename.lst, filename.itm\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void create\_files(const char \*filename){

    //make files

    char fname[MAX\_FILENAME];

    strcpy(fname, filename);

    file\_ext\_appender(fname, "lst");

    lst=fopen(fname, "w+");

    file\_ext\_appender(fname, "obj");

    obj=fopen(fname, "w+");

    file\_ext\_appender(fname, "itm");

    itm=fopen(fname, "w+");

}

/\*---------------------------------------------------------------------\*/

/\*Function: symbol\_clear\*/

/\*Purpose: removes symbol table\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void symbol\_clear(struct NODE \*\*head){

    struct NODE \*cur=(\*head);

    struct NODE\* next;

    while(cur!=NULL){

        next=cur->next;

        free(cur);

        cur=next;

    }

    \*head=NULL;

}

/\*---------------------------------------------------------------------\*/

/\*Function: symtab\_retval\*/

/\*Purpose: retrieves value from corresponding symbol from symbol table\*/

/\*Return: symbol data if exists, -1 if doesn't exists\*/

/\*---------------------------------------------------------------------\*/

int symtab\_retval(const char \*label, struct NODE \*\*head){

    for(struct NODE \*cur=(\*head);cur!=NULL;cur=cur->next){

        if(!strcmp(label, cur->cmd)){

            return cur->num;

        }

    }

    return -1;

}

/\*---------------------------------------------------------------------\*/

/\*Function: symtab\_exist\*/

/\*Purpose: determine whether label exists in symbol table\*/

/\*Return: true if exists, false if doesn't exists\*/

/\*---------------------------------------------------------------------\*/

bool symtab\_exist(const char \*label, struct NODE \*\*head){

    for(struct NODE \*cur=(\*head);cur!=NULL;cur=cur->next){

        if(!strcmp(label, cur->cmd)){

            return true;

        }

    }

    return false;

}

/\*---------------------------------------------------------------------\*/

/\*Function: symtab\_add\*/

/\*Purpose: add data to symbol table\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void symtab\_add(const char \*label, int locctr, struct NODE \*\*head){

    struct NODE \*newNode=(struct NODE\*)malloc(sizeof(struct NODE));

    strcpy(newNode->cmd, label);

    newNode->num=locctr;

    newNode->next=NULL;

    struct NODE \*cur=(\*head);

    if(\*head==NULL || strcmp((\*head)->cmd, newNode->cmd)>=0){

        newNode->next=(\*head);

        \*head=newNode;

    }

    else{

        while(cur->next!=NULL && strcmp(cur->next->cmd,newNode->cmd)<0){

            cur=cur->next;

        }

        newNode->next=cur->next;

        cur->next=newNode;

    }

}

/\*---------------------------------------------------------------------\*/

/\*Function: is\_register\*/

/\*Purpose: checks if given string is a register\*/

/\*Return: corresponding register number, -1 if not a register\*/

/\*---------------------------------------------------------------------\*/

int is\_register(const char \*operand){

    if(!strcmp(operand,"A")) return 0;

    if(!strcmp(operand,"X")) return 1;

    if(!strcmp(operand,"L")) return 2;

    if(!strcmp(operand,"B")) return 3;

    if(!strcmp(operand,"S")) return 4;

    if(!strcmp(operand,"T")) return 5;

    if(!strcmp(operand,"F")) return 6;

    if(!strcmp(operand,"PC")) return 8;

    if(!strcmp(operand,"SW")) return 9;

    return -1;

}

/\*---------------------------------------------------------------------\*/

/\*Function: appendString\*/

/\*Purpose: Makes object code and prints into corresponding files\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void appendString(char \*target, int xbpe, int temp, int mode, bool indexed){

    char xbpe\_str[5];

    char OPADR\_str[10];

    char t[10];

    sprintf(xbpe\_str, "%X", xbpe);

    if(mode==0){

        strcat(target, xbpe\_str);

        if(temp>=0){

        sprintf(OPADR\_str, "%03X", temp);

        strcat(target, OPADR\_str);

        }

        else{

            sprintf(t, "%X", temp);

            int tlen=strlen(t);

            for(int i=0;i<4;i++){

                t[i]=t[i+tlen-3];

            }

            strcat(target, t);

        }

    }

    if(mode==1){

        if(temp>=0){

            if(!indexed) sprintf(OPADR\_str, "1%05X", temp);

            else sprintf(OPADR\_str, "9%05X", temp);

            strcat(target, OPADR\_str);

        }

        else{

            sprintf(t, "%X", temp);

            int tlen=strlen(t);

            for(int i=0;i<6;i++){

                t[i]=t[i+tlen-5];

            }

            strcat(target, t);

        }

    }

}

/\*---------------------------------------------------------------------\*/

/\*Function: LSTprinter\*/

/\*Purpose: prints to console with format (for debugging)\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void LSTprinter(int line\_number, int LOCCTR, const char \*label, const char \*opcode, const char \*oper, const char\*idx, bool index\_exists){

        printf("%d\t", line\_number);

        if(LOCCTR!=-1) printf("%04X\t", LOCCTR);

        else printf("\t");

        if(label[0]!='\*') printf("%s\t", label);

        else printf("\t");

        if(opcode[0]!='\*') printf("%s\t", opcode);

        else printf("\t");

        if(index\_exists){

            //BUFFER, X

            if(strlen(oper)+strlen(idx)>5) printf("%s, %s ", oper, idx);

            else printf("%s, %s\t", oper, idx);

        }

        else{

            if(oper[0]!='\*') printf("%s\t", oper);

            else printf("\t");

        }

        printf("\t");

}

/\*---------------------------------------------------------------------\*/

/\*Function: LSTfprinter\*/

/\*Purpose: prints to listing file with format\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void LSTfprinter(FILE \*fp, int line\_number, int LOCCTR, const char \*label, const char \*opcode, const char \*oper, const char\*idx, bool index\_exists){

        fprintf(fp, "%d\t", line\_number);

        if(LOCCTR!=-1) fprintf(fp, "%04X\t", LOCCTR);

        else fprintf(fp, "\t");

        if(label[0]!='\*') fprintf(fp, "%s\t", label);

        else fprintf(fp, "\t");

        if(opcode[0]!='\*') fprintf(fp, "%s\t", opcode);

        else fprintf(fp, "\t");

        if(index\_exists){

            //BUFFER, X

            if(strlen(oper)+strlen(idx)>5) fprintf(fp, "%s, %s ", oper, idx);

            else fprintf(fp, "%s, %s\t", oper, idx);

        }

        else{

            if(oper[0]!='\*') fprintf(fp, "%s\t", oper);

            else fprintf(fp, "\t");

        }

        fprintf(fp, "\t");

}

/\*---------------------------------------------------------------------\*/

/\*Function: getINSTtype\*/

/\*Purpose: gets instruction type\*/

/\*Return: instruction type, and -1 if invalid\*/

/\*---------------------------------------------------------------------\*/

int getINSTtype(const char \*label, const char \*opcode, const char \*oper, const char\*idx){

    if(label[0]!='\*' && opcode[0]!='\*' && oper[0]!='\*' && idx[0]!='\*' ) return 1;

    if(label[0]=='\*' && opcode[0]!='\*' && oper[0]!='\*' && idx[0]!='\*' ) return 2;

    if(label[0]!='\*' && opcode[0]!='\*' && oper[0]!='\*' && idx[0]=='\*' ) return 3;

    if(label[0]=='\*' && opcode[0]!='\*' && oper[0]!='\*' && idx[0]=='\*' ) return 4;

    if(label[0]!='\*' && opcode[0]!='\*' && oper[0]=='\*' && idx[0]=='\*' ) return 5;

    if(label[0]=='\*' && opcode[0]!='\*' && oper[0]=='\*' && idx[0]=='\*' ) return 6;

    return -1;

}

/\*---------------------------------------------------------------------\*/

/\*Function: onlynumber\*/

/\*Purpose: checks if string contains only digits\*/

/\*Return: true/false\*/

/\*---------------------------------------------------------------------\*/

bool onlynumber(const char \*s){

    for(int i=0;i<strlen(s);i++){

        if(!(s[i]>='0' && s[i]<='9')) return false;

    }

    return true;

}

/\*---------------------------------------------------------------------\*/

/\*Function: assemble\*/

/\*Purpose: executes assembling process\*/

/\*Return: -1 if error occurs, 1 if successful\*/

/\*---------------------------------------------------------------------\*/

int assemble(const char \*filename, struct NODE \*\*head){

    FILE \*fp;

    int len=strlen(filename);

    //if no file

    if(!(fp=fopen(filename,"r"))){

        printf("no file\n");

        return -1;

    }

    //if invalid extension

    if(!(filename[len-1]=='m' && filename[len-2]=='s' && filename[len-3]=='a' && filename[len-4]=='.')){

        fclose(fp);

        printf("invalid extension\n");

         return -1;

    }

    int line\_number=5;      //current line number

    char line[120];         //current line string

    struct INST cur;        //tokenized data of current line

    cur.comment=false;

    lst=obj=NULL;

    bool ERR\_FLAG=false;    //ERROR FLAG

    int LOCCTR=0;           //Location counter of current line

    int start\_adr=0;        //Starting address of program

    int program\_length=0;   //Program length

    int num;                //multipurpose

    char hexa[2];           //to convert digits to hexadecimal

    int modif\_record\_num=0; //number of modification records

    int \*modif;             //modification records dynamically allocated

    create\_files(filename);

    //========================================PASS 1========================================

    //read first input line

    if(fgets(line, sizeof(line), fp)!=NULL){

        cur=myparser(line, line\_number);

        if(!strcmp(cur.opcode, "START")){

            //SAVE  #[OPERAND] AS STARTING ADDRESS

            sscanf(cur.oper, "%X", &start\_adr);

            LOCCTR=start\_adr;

            fprintf(itm, "%d\t%04X\t%s\t%s\t%s \*\n", line\_number,LOCCTR,  cur.label, cur.opcode, cur.oper);

            //INITIALIZE LOCCTR TO STARTING ADDRESS

        }

        else{

            //INITIALIZE LOCCTR TO 0

            LOCCTR=0;

        }

        line\_number+=5;

    }

    while(fgets(line, sizeof(line), fp)!=NULL){

        int dx=0;

        cur=myparser(line, line\_number);

        if(!strcmp(cur.opcode,"END")){

            fprintf(itm,"%d\t", line\_number);

            fprintf(itm, "-1\t\*\t%s\t%s \*\n", cur.opcode, cur.oper);

            break;

        }

        //if this is not a comment line

        if(!cur.comment){

            //if there is a symbol in the label field

            if(cur.label[0]!='\0'){

                if(symtab\_exist(cur.label, &symbol)){

                    //set error flag

                    ERR\_FLAG=true;

                    printf("Error at line %d : duplicate symbol\n", line\_number);

                    remove\_files(filename);

                    symbol\_clear(&symbol);

                    return -1;

                }

                else{

                    symtab\_add(cur.label, LOCCTR, &symbol);

                }

            }

            sscanf(cur.oper, "%d", &num);

            //if opcode is found in optab

            if(op\_return(cur.opcode, hash)!=-1){

                if(cur.format==4) dx+=4;

                else{

                    dx+=op\_format(cur.opcode,hash);

                }

            }

            else if(!strcmp(cur.opcode,"WORD")){

                //if OPCODE is WORD add 3 to LOCCTR

                dx+=3;

            }

            else if(!strcmp(cur.opcode,"RESW")){

                //if OPCODE is RESW add 3 \* #[OPERAND] to LOCCTR

                dx+=3\*num;

            }

            else if(!strcmp(cur.opcode, "RESB")){

                //if OPCODE is RESB add #[OPERAND] to LOCCTR

                dx+=num;

            }

            else if(!strcmp(cur.opcode, "BYTE")){

                //if OPCODE is BYTE find length of constant in bytes and add to LOCCTR

                if(cur.oper[0]=='C'){

                    dx+=(int)strlen(cur.oper)-3;

                }

                else if(cur.oper[0]=='X'){

                    dx+=(int)(strlen(cur.oper)-3)/2;

                }

            }

            else{

                //if OPCODE is not BASE, return ERROR

                if(strcmp(cur.opcode, "BASE")){

                    printf("Error at line %d : invalid opcode\n", line\_number);

                    remove\_files(filename);

                    symbol\_clear(&symbol);

                    return -1;

                }

                //if invalid format, return ERROR

                if(cur.type==-1){

                    ERR\_FLAG=true;

                    printf("Error at line %d : invalid format instruction\n", line\_number);

                    remove\_files(filename);

                    symbol\_clear(&symbol);

                    return -1;

                }

            }

            //if format 4

            if(cur.format==4){

                int templen=strlen(cur.opcode);

                for(int i=templen;i>0;i--){

                    cur.opcode[i]=cur.opcode[i-1];

                }

                cur.opcode[0]='+';

                modif\_record\_num++;

            }

            //print to intermediate file in a more accessible format

            fprintf(itm, "%d\t", line\_number);

            if(!strcmp(cur.opcode, "BASE")) fprintf(itm, "-1\t");

            else fprintf(itm, "%04X\t", LOCCTR);

            if(cur.label[0]=='\0') fprintf(itm, "\*\t");

            else fprintf(itm, "%s\t", cur.label);

            if(cur.opcode[0]=='\0') fprintf(itm, "\*\t");

            else fprintf(itm, "%s\t", cur.opcode);

            if(cur.oper[0]=='\0') fprintf(itm, "\*\t");

            else fprintf(itm, "%s\t", cur.oper);

            if(cur.idx[0]=='\0') fprintf(itm, "\*\n");

            else fprintf(itm, "%s\n", cur.idx);

            LOCCTR+=dx;

        }

        else{

            fprintf(itm, "%s", line);

        }

        line\_number+=5;

    }

    program\_length=LOCCTR-start\_adr;

    //===================================END OF PASS 1========================================

    //REWIND FILE POINTER BACK

    rewind(itm);

    int operand\_address=0;      //operand address

    int BASE=0;                 //BASE value for base relative instructions

    int recordlen=0;            //length of record

    bool newline=false;         //for RESB, RESW instructions which makes newline in obj files

    char record[100];           //for record in obj file

    char appendOBJ[100];        //for appending

    char append[10];            //for appending

    //Set modification dynamic array

    modif=(int\*)malloc(sizeof(int)\*modif\_record\_num);

    int modif\_index=0;

    int first\_executable=-1;

    int record\_global=0;

    //Reset record associated char arrays

    memset(record, '\0', sizeof(record));

    memset(append,'\0', sizeof(append));

    record[0]='T';

    //===================================PASS 2 START=========================================

    if(fgets(line, sizeof(line), itm)!=NULL){

        sscanf(line, "%d %d %s %s %s %s", &line\_number, &LOCCTR, cur.label, cur.opcode, cur.oper, cur.idx);

        if(!strcmp(cur.opcode, "START")){

            fprintf(lst, "%d\t%04X\t%s\t%s\t%s\t\n", line\_number, LOCCTR, cur.label, cur.opcode, cur.oper);

        }

        fprintf(obj, "H%s %06X%06X\n", cur.label, start\_adr, program\_length);

    }

    while(fgets(line, sizeof(line), itm)!=NULL){

        sscanf(line, "%d %X %s %s %s %s", &line\_number, &LOCCTR, cur.label, cur.opcode, cur.oper, cur.idx);

        int xbpe;                   //xbpe value

        int OP\_val;                 //Opcode value

        char OBJ\_str[100];          //Object code

        ERR\_FLAG=ERR\_FLAG;

        bool immediate=false;       //checks immediate

        bool indirect=false;        //checks indirect

        int temp;

        bool extended=false;        //checks extended

        bool index\_exists=false;    //checks index

        int format=getINSTtype(cur.label, cur.opcode, cur.oper, cur.idx);

        if(format==1 || format==2) index\_exists=true;

        if(line[0]=='.'){

            fprintf(lst, "%d\t\t%s", line\_number, line);

            continue;

        }

        LSTfprinter(lst, line\_number, LOCCTR, cur.label, cur.opcode, cur.oper, cur.idx, index\_exists);

        //search OPTAB for OPCODE

        if(cur.opcode[0]=='+'){

            temp=strlen(cur.opcode); extended=true;

            for(int i=0;i<temp;i++) cur.opcode[i]=cur.opcode[i+1];

        }

        //CHECKED FOR FORMAT 4 OPERATIONS

        //if opcode == BASE, set base value

        if(!strcmp(cur.opcode, "BASE")){

            BASE=symtab\_retval(cur.oper, &symbol);

        }

        if(op\_return(cur.opcode, hash)!=-1){

            //if there is a symbol in OPERAND field

            if(cur.oper[0]!='\*'){

                if(format==1 || format==2){

                    //IF INDEX EXISTS, REMOVE COMMA AT BACK

                    index\_exists=true;

                    temp=strlen(cur.oper);

                }

                //if immediate addressing

                if(cur.oper[0]=='#'){

                    temp=strlen(cur.oper); immediate=true;

                    for(int i=0;i<temp;i++) cur.oper[i]=cur.oper[i+1];

                    //+LDT #4096

                    if(onlynumber(cur.oper)) sscanf(cur.oper, "%d", &operand\_address);

                    //LDB #LENGTH

                    else{

                        if(symtab\_exist(cur.oper, &symbol)){

                            operand\_address=symtab\_retval(cur.oper, &symbol);

                        }

                    }

                }

                //if indirect addressing

                else if(cur.oper[0]=='@'){

                    temp=strlen(cur.oper); indirect=true;

                    for(int i=0;i<temp;i++) cur.oper[i]=cur.oper[i+1];

                }

                //IF NORMAL OPCODE STL RETADR

                if(symtab\_exist(cur.oper, &symbol)){

                    operand\_address=symtab\_retval(cur.oper, &symbol);

                }

                //IF IMMEDIATE CONSTANT : LDA #3

                else if(onlynumber(cur.oper)){

                    sscanf(cur.oper,"%d", &operand\_address);

                }

                //IF REGISTER ex : CLEAR X

                else if(is\_register(cur.oper)!=-1){

                    operand\_address=is\_register(cur.oper);

                }

                else{

                    operand\_address=0;

                    printf("Error at line %d : undefined symbol\n", line\_number);

                    remove\_files(filename);

                    symbol\_clear(&symbol);

                    return -1;

                }

                //ASSEMBLE OBJECT CODE INSTRUCTION

                //if operand is a register

                OP\_val=op\_return(cur.opcode, hash);

                if(is\_register(cur.oper)!=-1){

                    sprintf(OBJ\_str, "%02X", OP\_val);

                    int first=is\_register(cur.oper);

                    int second=is\_register(cur.idx);

                    if(second==-1) second=0;

                    char f[2], s[2];

                    f[0]=first+'0'; s[0]=second+'0'; s[1]=f[1]='\0';

                    strcat(OBJ\_str, f);

                    strcat(OBJ\_str, s);

                    line\_number+=5;

                    fprintf(lst, "%s\n", OBJ\_str);

                    goto TEXTRECORD;

                }

                else if(immediate){

                    sprintf(OBJ\_str, "%02X", OP\_val+1);

                }

                else if(indirect){

                    sprintf(OBJ\_str, "%02X", OP\_val+2);

                }

                else{

                    sprintf(OBJ\_str, "%02X", OP\_val+3);

                }

                //CALCULATE DISPLACEMENT

                //TRY PC FIRST

                temp=operand\_address-LOCCTR-3;

                if(immediate && onlynumber(cur.oper)){

                    sscanf(cur.oper, "%d", &temp);

                }

                if(extended){

                    if(!onlynumber(cur.oper) && !immediate && !indirect){

                        modif[modif\_index++]=record\_global+1;

                    }

                    appendString(OBJ\_str, xbpe, operand\_address, 1, index\_exists);

                }

                else if(-2048<=temp && temp<=2047){

                    //calculate displacement PC relative

                    xbpe=0;

                    if(extended) xbpe+=1;

                    if(index\_exists) xbpe+=8;

                    xbpe+=2;

                    if(onlynumber(cur.oper)) xbpe=0;

                    appendString(OBJ\_str, xbpe, temp,0, index\_exists);

                }

                else{

                    //TRY BASE

                    temp=operand\_address-BASE;

                    if(immediate && onlynumber(cur.oper)){

                        sscanf(cur.oper, "%d", &temp);

                    }

                    if(0<=temp && temp<=4095){

                        xbpe=0;

                        if(index\_exists)xbpe+=8;

                        xbpe+=4;

                        appendString(OBJ\_str, xbpe, temp,0, index\_exists);

                    }

                    //if nothing works, print error

                    else{

                        printf("Error at line %d: unreachable address\n", line\_number);

                        remove\_files(filename);

                    }

                }

            }

            else{

                OP\_val=op\_return(cur.opcode, hash);

                sprintf(OBJ\_str, "%X0000", OP\_val+3);

            }

        }

        //if OPCODE = BYTE

        else if(!strcmp(cur.opcode, "BYTE")){

            memset(OBJ\_str, '\0', sizeof(OBJ\_str));

            //calculates constant length in bytes

            if(cur.oper[0]=='C'){

                for(int i=2;i<strlen(cur.oper)-1;i++){

                    sprintf(hexa, "%02X", (int)cur.oper[i]);

                    strcat(OBJ\_str, hexa);

                }

            }

            else if(cur.oper[0]=='X'){

                for(int i=2;i<strlen(cur.oper)-1;i++){

                    OBJ\_str[i-2]=cur.oper[i];

                }

                OBJ\_str[strlen(cur.oper)-3]='\0';

            }

        }

        else if(!strcmp(cur.opcode, "WORD")){

            //generate object code with WORD directive

            memset(OBJ\_str, '\0', sizeof(OBJ\_str));

            int word\_directive;

            sscanf(cur.oper, "%d", &word\_directive);

            sprintf(OBJ\_str, "%06d", word\_directive);

        }

        else{

            memset(OBJ\_str, '\0', sizeof(OBJ\_str));

        }

        fprintf(lst, "%s\n", OBJ\_str);

        line\_number+=5;

        //Generates OBJECT CODE to write to .obj file, and write necessary newlines(RESW, RESB or full column)

        TEXTRECORD:

        if(first\_executable==-1) first\_executable=LOCCTR;

        //if record is full, or opcode is RESB or RESW, write a newline

        if(recordlen+strlen(OBJ\_str)/2>=30 || ((!strcmp(cur.opcode, "RESB") || !strcmp(cur.opcode, "RESW")))){

            if(!newline){

                sprintf(append, "%02X", recordlen);

                strcat(record, append);

                strcat(record, appendOBJ);

                fprintf(obj, "%s\n", record);

                memset(append, '\0', sizeof(append));

                memset(record, '\0', sizeof(record));

                memset(appendOBJ, '\0', sizeof(appendOBJ));

                recordlen=0;

                record[0]='T';

            }

            else{

                memset(append, '\0', sizeof(append));

                memset(record, '\0', sizeof(record));

                memset(appendOBJ, '\0', sizeof(appendOBJ));

                record[0]='T';

                continue;

            }

        }

        //Initialize new record with current LOCCTR

        if(recordlen==0){

            if(LOCCTR!=-1){

            sprintf(append, "%06X", LOCCTR);

            strcat(record, append);

            }

        }

        //Append OBJECT STRINGS to record

        if(recordlen+strlen(OBJ\_str)/2<30){

            strcat(appendOBJ, OBJ\_str);

            record\_global+=strlen(OBJ\_str)/2;

            recordlen+=strlen(OBJ\_str)/2;

        }

        //If RESB OR RESW, indicate newline

        if(!strcmp(cur.opcode, "RESB") || !strcmp(cur.opcode, "RESW"))  newline=true;

        else newline=false;

        //if opcode is END, write remaining object codes and write Modification records and End record

        if(!strcmp(cur.opcode, "END")){

            if(strlen(record)>6){

                sprintf(append, "%02X", recordlen);

                strcat(record, append);

                strcat(record, appendOBJ);

                fprintf(obj, "%s\n", record);

            }

            for(int i=0;i<modif\_index;i++){

                fprintf(obj, "M%06X%02X\n", modif[i], 5);

            }

            memset(record, '\0', sizeof(record));

            sprintf(append,"%06X", first\_executable);

            record[0]='E';

            strcat(record, append);

            fprintf(obj, "%s\n", record);

            break;

        }

    }

    //===================================END OF PASS 2=========================================

    //Close files

    fclose(itm);

    fclose(lst);

    fclose(obj);

    //Assemble success, so move symbol table to last success

    last\_success\_symtab=symbol;

    symbol=NULL;

    //remove intermediate file

    char fname[MAX\_FILENAME];

    strcpy(fname, filename);

    file\_ext\_appender(fname, "itm");

    remove(fname);

    file\_ext\_appender(fname, "lst");

    printf("[%s],", fname);

    file\_ext\_appender(fname, "obj");

    printf(" [%s]\n", fname);

    return 1;

}

/\*---------------------------------------------------------------------\*/

/\*Function: myparser\*/

/\*Purpose: tokenizes given string\*/

/\*Return: a struct containing tokenized information\*/

/\*---------------------------------------------------------------------\*/

struct INST myparser(const char \*line, int line\_number){

    struct INST cur;

    cur.comment=false;

    cur.line\_number=line\_number;

    cur.format=-1;

    int words=0;

    char label[30], opcode[30], oper[30], idx[30], trash[30];

    //INITIALIZE VALUES

    cur.label[0]=cur.opcode[0]=cur.oper[0]=cur.idx[0]=cur.trash[0]='\0';

    label[0]=opcode[0]=oper[0]=idx[0]=trash[0]='\0';

    words=sscanf(line, "%s %s %s %s %s", label, opcode, oper, idx, trash);

    //IF + is in front of OPCODE format 4

    if(label[0]=='+'){

        for(int i=0;i<strlen(label);i++) label[i]=label[i+1]; cur.format=4;

    }

    if(opcode[0]=='+'){

        for(int i=0;i<strlen(opcode);i++) opcode[i]=opcode[i+1]; cur.format=4;

    }

    int oper\_len=strlen(oper), opcode\_len=strlen(opcode);

    //assembler directives without labels

    if(words==2 && !strcmp(label, "END")){

        cur.type=4;

        strcpy(oper, opcode);

        strcpy(opcode, label);

        label[0]='\0';

        copy\_strings(&cur, label, opcode, oper, idx);

        return cur;

    }

    if(words==2 && !strcmp(label, "BASE")){

        cur.type=4;

        strcpy(oper, opcode);

        strcpy(opcode, label);

        label[0]='\0';

        copy\_strings(&cur, label, opcode, oper, idx);

        return cur;

    }

    if(label[0]=='.'){

        //LINE IS COMMENT

        cur.comment=true;

        return cur;

    }

    if(words==4 && oper[oper\_len-1]==','){

        //LABEL OPCODE OPERAND, INDEX

        cur.type=1;

        oper[oper\_len-1]='\0';

        copy\_strings(&cur, label, opcode, oper, idx);

        return cur;

    }

    else if(words==3 && idx[0]=='\0'){

        if(opcode[opcode\_len-1]==','){

            //  OPCODE OPERAND, INDEX

            strcpy(idx, oper); strcpy(oper, opcode); strcpy(opcode, label);

            oper\_len=strlen(oper);

            oper[oper\_len-1]='\0';

            label[0]='\0';

            cur.type=2;

            copy\_strings(&cur, label, opcode, oper, idx);

            return cur;

        }

        else{

            // LABEL OPCODE OPERAND

            cur.type=3;

            copy\_strings(&cur, label, opcode, oper, idx);

            return cur;

        }

    }

    else if(words==2 && idx[0]=='\0' && oper[0]=='\0'){

        if(op\_return(label, hash)!=-1){

            //OPCODE OPERAND

            cur.type=4;

            strcpy(oper, opcode); strcpy(opcode, label);

            label[0]='\0';

        }

        else if(op\_return(opcode, hash)!=-1){

            //LABEL OPCODE

            cur.type=5;

        }

        else{

            //NO OPCODE PRESENT (set errorflag)

            cur.type=-1;

        }

        copy\_strings(&cur, label, opcode, oper, idx);

        return cur;

    }

    else if(words==1){

        cur.type=6;

        strcpy(opcode, label);

        label[0]='\0';

        copy\_strings(&cur, label, opcode, oper, idx);

        return cur;

    }

    else{

        //CALL DEBUGGER

        cur.type=-1;

        printf("ERROR AT LINE %d\n", line\_number);

        return cur;

    }

}

/\*---------------------------------------------------------------------\*/

/\*Function: copy\_strings\*/

/\*Purpose: copies from string to string (multiple)\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void copy\_strings(struct INST \*target, const char \*label, const char \*opcode, const char \*oper, const char \*idx){

    strcpy(target->label, label); strcpy(target->oper, oper);

    strcpy(target->opcode, opcode); strcpy(target->idx, idx);

}

**assembler.h**

#include "defines.h"

struct INST{

    bool comment;

    int line\_number;

    int type;

    int format;

    /\*

    LABEL OPCODE OPERAND, INDEX  - type 1

          OPCODE OPERAND, INDEX  - type 2

    LABEL OPCODE OPERAND         - type 3

          OPCODE OPERAND         - type 4

    LABEL OPCODE                 - type 5

          OPCODE                 - type 6

    \*/

    char trash[30];

    char label[30];

    char oper[30];

    char opcode[30];

    char idx[30];

};

void file\_ext\_appender(char\* filename, const char\* extension);

void LSTfprinter(FILE \*fp, int line\_number, int LOCCTR, const char \*label, const char \*opcode, const char \*oper, const char\*idx, bool index\_exists);

void LSTprinter(int line\_number, int LOCCTR, const char \*label, const char \*opcode, const char \*oper, const char\*idx, bool index\_exists);

void symbol\_clear(struct NODE \*\*head);

void appendString(char \*target, int xbpe, int temp, int mode, bool indexed);

int is\_register(const char \*operand);

bool onlynumber(const char \*s);

int symtab\_retval(const char \*label, struct NODE \*\*head);

bool symtab\_exist(const char \*label, struct NODE \*\*head);

void symtab\_add(const char \*label, int locctr, struct NODE \*\*head);

void create\_files( const char \*filename);

void remove\_files(const char \*filename);

void copy\_strings(struct INST \*target, const char \*label, const char \*opcode, const char \*oper, const char \*idx);

struct INST myparser(const char \*line, int line\_number);

int assemble(const char \*filename, struct NODE \*\*head);

**mem\_cmd.c**

#include "mem\_cmd.h"

/\*---------------------------------------------------------------------\*/

/\*Function: edit\*/

/\*Purpose: edits memory based on address and value\*/

/\*Return: void\*/

/\*---------------------------------------------------------------------\*/

void edit(unsigned char mem[], int address, int value){

    mem[address]=(char)value;

}

/\*---------------------------------------------------------------------\*/

/\*Function: reset\*/

/\*Purpose: resets whole memory to 0\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void reset(unsigned char mem[]){

    for(int i=0;i<=0xFFFFF;i++){

        mem[i]=0;

    }

}

/\*---------------------------------------------------------------------\*/

/\*Function: fill\*/

/\*Purpose: assigns value to memory inside given address range\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void fill(unsigned char mem[], int start, int end, int value){

    //range overflow exception

    if(start>end) {

        printf("invalid range\n");

        return;

    }

    for(int i=start;i<=end;i++){

        mem[i]=value;

    }

}

/\*---------------------------------------------------------------------\*/

/\*Function: print\*/

/\*Purpose: prints formatted contents of virtual memory\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void print(unsigned char mem[], int start, int end){

    //ex: du 15 57 (decimal)  -->  0 ~ 59

    int s=(start/16)\*16;

    int e=(end/16+1)\*16-1;

    //range overflow exception

    if(e>0xFFFFF) e=0xFFFFF;

    for(int i=s;i<=e;i+=16){

        //address | memory contents | ASCII CODE

        printf("%05X ", i);

        for(int j=i;j<i+16;j++){

            if(j<start || j>end) printf("   ");

            else{

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

            }

        }

        printf("; ");

        for(int j=i;j<i+16;j++){

            if(j<start || j>end) printf(".");

            else if(mem[j]<0x20 || mem[j]>0x7E) printf(".");

            else printf("%c", mem[j]);

        }

        printf("\n");

    }

}

/\*---------------------------------------------------------------------\*/

/\*Function: dump\*/

/\*Purpose: variadic function - checks argument size and calls corresponding function\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void dump(unsigned char mem[], int args, ...){

    va\_list ap;

    va\_start(ap, args);

    int num;

    if(args==1){

        //if only dump or dump start

        int start,end;

        for(int i=0;i<args;i++){

            num=va\_arg(ap,int);

            if(i==0) start=num;

        }

        end=start+159;

        va\_end(ap);

        print(mem, start, end);

    }

    else if(args==2){

        //dump start end

        int start, end;

        for(int i=0;i<args;i++){

            num=va\_arg(ap, int);

            if(i==0) start=num;

            if(i==1) end=num;

        }

        va\_end(ap);

        //range exception when dump 4,3

        if(start>end){

            printf("range error!\n"); return;

        }

        print(mem, start, end);

    }

}

**mem\_cmd.h**

#include "defines.h"

void dump(unsigned char mem[], int args, ...);

void print(unsigned char mem[], int start, int end);

char \*intToHex(int n, int len);

void fill(unsigned char mem[], int start, int end, int value);

void edit(unsigned char mem[], int address, int value);

void reset(unsigned char mem[]);

**opcode.c**

#include "opcode.h"

#define MAX\_HASH 20

void history\_add(char\* command, struct NODE \*\*head);

/\*---------------------------------------------------------------------\*/

/\*Function: calc\_key\*/

/\*Purpose: calculates hash function\*/

/\*Return: integer value of hash key\*/

/\*---------------------------------------------------------------------\*/

int calc\_key(const char\*cmd){

    int key=0;

    for(int i=0;i<strlen(cmd);i++){

        key+=(int)cmd[i];

    }

    return key%MAX\_HASH;

}

/\*---------------------------------------------------------------------\*/

/\*Function: addHash\*/

/\*Purpose: adds a node to hash table\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void addHash(struct NODE \*node, struct NODE \*hash[]){

    int key=calc\_key(node->cmd);

    if(hash[key]==NULL){

        hash[key]=node;

    }

    else{

        node->next=hash[key];

        hash[key]=node;

    }

}

/\*---------------------------------------------------------------------\*/

/\*Function: init\_hash\*/

/\*Purpose: reads opcode.txt and makes hash table\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void init\_hash(struct NODE \*hash[]){

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

    int cmd\_num;

    char op[10];

    char format[10];

    while(EOF!=fscanf(fp,"%x %s %s", &cmd\_num, op, format)){

        struct NODE \*newNode=(struct NODE\*)malloc(sizeof(struct NODE));

        strcpy(newNode->cmd, op);

        strcpy(newNode->format, format);

        newNode->num=cmd\_num;

        newNode->next=NULL;

        addHash(newNode, hash);

    }

}

/\*---------------------------------------------------------------------\*/

/\*Function: op\_cmd\*/

/\*Purpose: takes mnemonic input and prints corresponding opcode\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void op\_cmd(char \*mnemonic, struct NODE \*hash[], struct NODE \*\*head){

    int key=calc\_key(mnemonic);

    char history\_string[100];

    struct NODE\*cur=hash[key];

    while(cur!=NULL){

        if(!strcmp(cur->cmd, mnemonic)){

            sprintf(history\_string,"opcode %s", mnemonic);

            history\_add(history\_string, head);

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

            return;

        }

        cur=cur->next;

    }

    printf("no such opcode\n"); return;

}

/\*---------------------------------------------------------------------\*/

/\*Function: op\_ret\*/

/\*Purpose: get corresponding opcode\*/

/\*Return: opcode for mnemonic\*/

/\*---------------------------------------------------------------------\*/

int op\_return(const char\* mnemonic, struct NODE\* hash[]){

    int key=calc\_key(mnemonic);

    struct NODE\*cur=hash[key];

    while(cur!=NULL){

        if(!strcmp(cur->cmd, mnemonic)) return cur->num;

        cur=cur->next;

    }

    return -1;

}

int op\_format(const char\* mnemonic, struct NODE\* hash[]){

    int key=calc\_key(mnemonic);

    struct NODE\*cur=hash[key];

    while(cur!=NULL){

        if(!strcmp(cur->cmd, mnemonic)){

            if(!strcmp(cur->format, "1")) return 1;

            else if(!strcmp(cur->format, "2")) return 2;

            else if(!strcmp(cur->format, "3/4")) return 3;

        }

        cur=cur->next;

    }

    return -1;

}

/\*---------------------------------------------------------------------\*/

/\*Function: op\_list\*/

/\*Purpose: Prints all opcodes in format\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void op\_list(struct NODE \*hash[]){

    for(int i=0;i<MAX\_HASH;i++){

        printf("%2d : ", i);

        if(hash[i]!=NULL){

            struct NODE \*cur=hash[i];

            while(cur!=NULL){

                printf("[%s, %X] ", cur->cmd, cur->num);

                cur=cur->next;

                if(cur!=NULL){

                    printf(" -> ");

                }

            }

        }

        printf("\n");

    }

}

**opcode.h**

#include "defines.h"

int op\_format(const char\* mnemonic, struct NODE\* hash[]);

int op\_return(const char\* mnemonic, struct NODE\* hash[]);

void init\_hash(struct NODE\*hash[]);

void op\_cmd(char \*mnemonic, struct NODE\*hash[], struct NODE \*\*head);

void op\_list(struct NODE\*hash[]);

**shell\_cmd.c**

#include "shell\_cmd.h"

/\*---------------------------------------------------------------------\*/

/\*Function: trim\*/

/\*Purpose: removes leading and trailing spaces in string\*/

/\*Return: char pointer of trimmed string\*/

/\*---------------------------------------------------------------------\*/

char\* trim(char \*cmd){

    //trims leading and trailing spaces

    while(isspace(\*cmd))cmd++;

    if(\*cmd==0) return cmd;

    int end=strlen(cmd)-1;

    while(isspace(cmd[end])){

        end--;

    }

    cmd[end+1]='\0';

    return cmd;

}

/\*---------------------------------------------------------------------\*/

/\*Function: checkExe\*/

/\*Purpose: checks if given file name is executable\*/

/\*Return: true if executable, false if not executable \*/

/\*---------------------------------------------------------------------\*/

bool checkExe(const char \*name){

    struct stat sb;

    if (stat(name, &sb) == 0 && sb.st\_mode & S\_IXUSR){

        /\* executable \*/

        return true;

    }

    else return false;

}

/\*---------------------------------------------------------------------\*/

/\*Function: checkDir\*/

/\*Purpose: checks if given file name is directory\*/

/\*Return: true if directory / false if not directory \*/

/\*---------------------------------------------------------------------\*/

bool checkDir(const char\* name){

    //checks if given file name is a directory

    struct stat status;

    stat(name,&status);

    return S\_ISDIR(status.st\_mode);

}

/\*---------------------------------------------------------------------\*/

/\*Function: shell\_help\*/

/\*Purpose: Print available commands\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void shell\_help(void){

    printf("h[elp]\n");

    printf("d[ir]\n");

    printf("q[uit]\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");

}

/\*---------------------------------------------------------------------\*/

/\*Function: shell\_dir\*/

/\*Purpose: prints files/directories in current directory\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void shell\_dir(void){

    DIR \*dir\_i=opendir(".");

    struct dirent \*dir\_e;

    if(dir\_i!=NULL){

        while((dir\_e=readdir(dir\_i))){

            char filename[100];

            //copy file name

            strcpy(filename, dir\_e->d\_name);

            int len=strlen(filename);

            //check if execution file (.out)

            if(checkExe(dir\_e->d\_name)){

                filename[len]='\*';

                filename[len+1]='\0';

            }

            //check if directory

            if(checkDir(dir\_e->d\_name)){

                filename[len]='/';

                filename[len+1]='\0';

            }

            printf("%s\n", filename);

        }

    }

    closedir(dir\_i);

}

/\*---------------------------------------------------------------------\*/

/\*Function: history\_add\*/

/\*Purpose: adds command to history linked list\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void history\_add(char\* command, struct NODE \*\*head){

    struct NODE \*newNode=(struct NODE\*)malloc(sizeof(struct NODE));

    strcpy(newNode->cmd, command);

    newNode->next=NULL;

    struct NODE \*cur;

    if(\*head==NULL){

        \*head=newNode; return;

    }

    else{

        for(cur=(\*head);cur->next!=NULL;cur=cur->next){}

            cur->next=newNode;

    }

}

/\*---------------------------------------------------------------------\*/

/\*Function: history\_print\*/

/\*Purpose: prints contents of linked list \*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void history\_print(struct NODE \*\*head){

    if(\*head==NULL) return;

    //prints contents of linked list

    int cnt=1;

    for(struct NODE \*cur=(\*head);cur!=NULL;cur=cur->next, cnt++){

        printf("%d %s\n", cnt, cur->cmd);

    }

}

/\*---------------------------------------------------------------------\*/

/\*Function: type\_filename\*/

/\*Purpose: prints contents of file\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void type\_filename(const char \*filename, struct NODE \*\*head){

    if(checkDir(filename)) {

        printf("%s is a directory. Abort\n", filename); return;

    }

    FILE \*fp;

    if((fp=fopen(filename, "r"))){

        char content=fgetc(fp);

        while(content!=EOF){

            printf("%c", content);

            content=fgetc(fp);

        }

        printf("\n");

        fclose(fp);

        char history\_string[100];

        sprintf(history\_string, "type %s", filename);

        history\_add(history\_string, head);

        return;

    }

    printf("File doesn't exist\n"); return;

}

/\*---------------------------------------------------------------------\*/

/\*Function: symbol\_print\*/

/\*Purpose: prints symbol table in lexicographical order\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void symtab\_print(struct NODE \*\*head){

    if(\*head==NULL) return;

    for(struct NODE \*cur=(\*head);cur!=NULL;cur=cur->next){

        printf("\t%s\t%04X\n", cur->cmd, cur->num);

    }

}

**shell\_cmd.h**

#include "defines.h"

void symtab\_print(struct NODE \*\*head);

void history\_add(char command[], struct NODE \*\*head);

void history\_print(struct NODE \*\*head);

void shell\_dir(void);

void shell\_help(void);

void type\_filename(const char \*filename, struct NODE \*\*head);

char\* trim(char \*cmd);

**Makefile**

CC=gcc

CFLAGS=-g -Wall

OBJS=20161634.o shell\_cmd.o mem\_cmd.o opcode.o assembler.o

TARGET=20161634.out

$(TARGET): $(OBJS)

    $(CC) -o $@ $(OBJS)

20161634.o: 20161634.c shell\_cmd.h mem\_cmd.h opcode.h assembler.h defines.h

shell\_cmd.o: shell\_cmd.c shell\_cmd.h

mem\_cmd.o: mem\_cmd.c mem\_cmd.h

opcode.o: opcode.c opcode.h

assembler.o: assembler.c assembler.h

clean:

    rm -f \*.o

    rm -f $(TARGET)

**loader.c**

#include "loader.h"

#include "mem\_cmd.h"

#include "opcode.h"

#define CSECT 0

#define SYMBOL 1

#define PLUS 0

#define MINUS 1

#define MEMSET(string) memset(string, '\0', sizeof(string));

struct ESTNODE \*est\_head;

struct NODE \*hash[MAX\_HASH];

int csaddr;

int execaddr;

int proglength;

int \*bp;

int bp\_num=0;

int bp\_idx=0;

int bp\_cu=0;

int BASE=0x33;

bool next\_CC=false;

int CC=0; //'=' 0 '>:1' '<:-1'

struct REGISTERS{

    int A;  int X;

    int L;  int PC;

    int B;  int S;

    int T;

};

struct REGISTERS reg;

char REFTAB[20][20]={0};

unsigned char m[2<<20];

/\*---------------------------------------------------------------------\*/

/\*Function: print\_regs\*/

/\*Purpose: prints contents of registers\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void print\_regs(void){

    printf("A : %06X  X : %06X\n", reg.A, reg.X);

    printf("L : %06X  PC: %06X\n", reg.L, reg.PC);

    printf("B : %06X  S : %06X\n", reg.B, reg.S);

    printf("T : %06X\n", reg.T);

}

/\*---------------------------------------------------------------------\*/

/\*Function: print\_bp\*/

/\*Purpose: prints breakpoints\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void print\_bp(){

    printf("           breakpoint\n");

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

    for(int i=0;i<bp\_num;i++){

        printf("           %X\n", bp[i]);

    }

}

/\*---------------------------------------------------------------------\*/

/\*Function: clear\_bp\*/

/\*Purpose: clears all breakpoints\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void clear\_bp(){

    bp\_num=0;

    reg.A=reg.X=reg.PC=reg.B=reg.S=reg.T=0;

    reg.L=proglength;

    free(bp);

}

/\*---------------------------------------------------------------------\*/

/\*Function: add\_bp\*/

/\*Purpose: adds breakpoints\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void add\_bp(int new\_bp){

    if(bp\_num==0){

        if((bp=(int\*)malloc(sizeof(int)))==NULL){

            printf("malloc error.\n");

        }

        bp\_num++;

    }

    else{

        bp\_num++;

        if((bp=(int\*)realloc(bp,sizeof(int)\*bp\_num))==NULL){

            printf("realloc error.\n");

        }

    }

    bp[bp\_num-1]=new\_bp;

}

/\*---------------------------------------------------------------------\*/

/\*Function: decHex\*/

/\*Purpose: converts 4byte word to 3byte word\*/

/\*Return: hexadecimal\*/

/\*---------------------------------------------------------------------\*/

int decHex(const char \*hex){

    char neg[10]="FF";

    char pos[10]="00";

    int ret;

    if(hex[0]==9 || hex[0]==8 || (hex[0]>='A' && hex[0]<='F')){

        //negative

        strcat(neg, hex);

        sscanf(neg, "%X", &ret);

    }

    else{

        strcat(pos, hex);

        sscanf(pos, "%X", &ret);

    }

    return ret;

}

/\*---------------------------------------------------------------------\*/

/\*Function: file\_exists\*/

/\*Purpose: checks if input file exists\*/

/\*Return: true if exists, false if not exists\*/

/\*---------------------------------------------------------------------\*/

bool file\_exists(char \*name){

    struct stat temp;

    return (stat(name, &temp)==0);

}

/\*---------------------------------------------------------------------\*/

/\*Function: destroy\_EST\*/

/\*Purpose: destroys ESTAB\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void destroy\_EST(struct ESTNODE \*\*est\_head){

    struct ESTNODE\* cur=\*est\_head;

    struct ESTNODE\* next;

    while(cur!=NULL){

        next=cur->next;

        free(cur);

        cur=next;

    }

    \*est\_head=NULL;

}

/\*---------------------------------------------------------------------\*/

/\*Function: print\_EST\*/

/\*Purpose: prints ESTAB\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void print\_EST(struct ESTNODE \*\*est\_head){

    proglength=0;

    printf("control symbol address length\n");

    printf("section name\n");

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

    for(struct ESTNODE \*cur=(\*est\_head);cur!=NULL;cur=cur->next){

        if(cur->length==-1){

            printf("%s\t%s\t%04X\n", cur->csect, cur->symbol, cur->address);

        }

        else{

            printf("%s\t%s\t%04X\t%04X\n", cur->csect, cur->symbol, cur->address, cur->length);

            proglength+=cur->length;

        }

    }

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

    printf("          total length  %04X\n", proglength);

}

/\*---------------------------------------------------------------------\*/

/\*Function: EST\_search\*/

/\*Purpose: searches ESTAB for given CSECT or SYMBOL\*/

/\*Return: corresponding address\*/

/\*---------------------------------------------------------------------\*/

int EST\_search(const char \*name, int type, struct ESTNODE \*\*est\_head){

    //if 0-csect 1-symbol

    for(struct ESTNODE \*cur=(\*est\_head);cur!=NULL;cur=cur->next){

        if(type==CSECT){

            if(!strcmp(name, cur->csect)){

                return cur->address;

            }

        }

        else{

            if(!strcmp(name, cur->symbol)){

                return cur->address;

            }

        }

    }

    return -1;

}

/\*---------------------------------------------------------------------\*/

/\*Function: EST\_add\*/

/\*Purpose: adds to ESTAB\*/

/\*Return: none\*/

/\*---------------------------------------------------------------------\*/

void EST\_add(char \*csect, char \*symbol, int address, int length, int type,  struct ESTNODE \*\*est\_head){

    struct ESTNODE \*newNode=(struct ESTNODE\*)malloc(sizeof(struct ESTNODE));

    strcpy(newNode->csect, csect);

    strcpy(newNode->symbol, symbol);

    newNode->address=address;

    newNode->length=length;

    newNode->type=type;

    newNode->next=NULL;

    struct ESTNODE \*cur;

    if(\*est\_head==NULL){

        \*est\_head=newNode;

    }

    else{

        for(cur=(\*est\_head);cur->next!=NULL;cur=cur->next){}

        cur->next=newNode;

    }

}

/\*---------------------------------------------------------------------\*/

/\*Function: pass1\*/

/\*Purpose: proceeds pass 1 of loader\*/

/\*Return: 1 on success/ -1 on fail\*/

/\*---------------------------------------------------------------------\*/

int pass1(char \*filename){

    int CSLTH;

    FILE \*file=fopen(filename, "r");

    int n;

    char buffer[100];

    char temp[100]; int num;

    fgets(buffer, sizeof(buffer), file);

    sscanf(buffer, "%s %x", temp, &num);

    n=strlen(temp);

    for(int i=0;i<n;i++) temp[i]=temp[i+1];

    CSLTH=num;

    //search ESTAB for control section name

    if(EST\_search(temp, CSECT, &est\_head)!=-1){

        printf("Error: duplicate external symbol\n");

        return -1;

    }

    else

        EST\_add(temp, "", csaddr, num, CSECT, &est\_head);

    while(1){

        fgets(buffer, sizeof(buffer), file);

        if(buffer[0]=='E'){

            break;

        }

        if(buffer[0]=='D'){

            n=strlen(buffer);

            for(int i=0;i<n;i++)

                buffer[i]=buffer[i+1];

            char \*s=buffer;

            while(sscanf(s,"%12[^\n]s", temp)==1){

                s+=12;

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

                strncpy(t, temp+6, 6);

                sscanf(t, "%X", &num);

                for(int i=0;i<6;i++){

                    if(temp[i]==' '){

                        t[i]='\0';

                        break;

                    }

                    t[i]=temp[i];

                }

                if(EST\_search(t, SYMBOL, &est\_head)!=-1){

                    printf("Error: duplicate external symbol\n");

                    return -1;

                }

                else

                    EST\_add("", t, csaddr+num, -1, SYMBOL, &est\_head);

                free(t);

            }

        }

    }

    csaddr+=CSLTH;

    fclose(file);

    return 0;

}

/\*---------------------------------------------------------------------\*/

/\*Function: pass2\*/

/\*Purpose: proceeds pass 2 of \*/

/\*Return: 1 on success -1 on fail\*/

/\*---------------------------------------------------------------------\*/

int pass2(char \*filename){

    int CSLTH;

    FILE \*file=fopen(filename, "r");

    char buffer[100];

    char temp[100]; int num,n;

    fgets(buffer, sizeof(buffer), file);

    sscanf(buffer, "%s %x", temp, &num);

    n=strlen(temp);

    for(int i=0;i<n;i++) temp[i]=temp[i+1];

    strcpy(REFTAB[1], temp);

    CSLTH=num;

    while(1){

        fgets(buffer, sizeof(buffer), file);

        if(buffer[0]=='E'){

            if(strlen(buffer)!=1){

                int END\_ADDR=0;

                sscanf(buffer, "E%06X", &END\_ADDR);

                execaddr=csaddr+END\_ADDR;

            }

            break;

        }

        if(buffer[0]=='R'){

            char \*t=buffer+1;

            int offset;

            while(sscanf(t, "%d%s%n", &num, temp, &offset)==2){

                t+=offset;

                strcpy(REFTAB[num], temp);

            }

        }

        if(buffer[0]=='T'){

            MEMSET(temp);

            char \*t=buffer+1;

            strncpy(temp, t, 6);

            int target, len;

            //T000020

            sscanf(temp, "%x", &target);

            target+=csaddr;

            t+=6;

            MEMSET(temp);

            strncpy(temp, t, 2);

            sscanf(temp, "%x", &len);

            t+=2;

            for(int i=0;i<len;i++){

                MEMSET(temp);

                strncpy(temp, t,2);

                sscanf(temp, "%X", &num);

                t+=2;

                edit(m, target+i, num);

            }

        }

        if(buffer[0]=='M'){

            MEMSET(temp);

            char \*t=buffer+1;

            strncpy(temp, t, 6);

            int target, len;

            sscanf(temp, "%x", &target);

            t+=6;

            MEMSET(temp);

            strncpy(temp, t, 2);

            sscanf(temp, "%x", &len);

            t+=2;

            int sign;

            if(\*t=='+') sign=PLUS;

            else sign=MINUS;

            t+=1;

            MEMSET(temp);

            strncpy(temp, t, 2);

            int ref;

            sscanf(temp, "%d", &ref);

            target+=csaddr;

            MEMSET(temp);

            sprintf(temp, "%02X%02X%02X", m[target], m[target+1], m[target+2]);

            int value=decHex(temp);

            int EST\_addr;

            if(ref==1)

                EST\_addr=EST\_search(REFTAB[ref], CSECT, &est\_head);

            else EST\_addr=EST\_search(REFTAB[ref], SYMBOL, &est\_head);

            if(sign==PLUS)

                value+=EST\_addr;

            if(sign==MINUS)

                value-=EST\_addr;

            char word[10];

            char \*h=word;

            char byte[3]; int n;

            sprintf(h, "%08X", value);

            h+=2;

            for(int i=0;i<3;i++){

                strncpy(byte, h,2);

                sscanf(byte, "%X", &n);

                edit(m, target+i,n);

                h+=2;

            }

        }

    }

    csaddr+=CSLTH;

    return 0;

}

/\*---------------------------------------------------------------------\*/

/\*Function: run\*/

/\*Purpose: runs program\*/

/\*Return: 1 on success -1 on fail\*/

/\*---------------------------------------------------------------------\*/

int run(void){

    bool break\_flag=false;

    bool prog\_end=false;

    int operand;

    while(1){

        bool PC\_edit=false;

        if(execaddr>=progaddr+proglength){

            prog\_end=true;

            break;

        }

            for(int i=0;i<bp\_num;i++){

                if(execaddr==bp[i]){

                    bp\_cu++;

                    break\_flag=true;

                    print\_regs();

                    printf("Stop at checkpoint[%X]\n", execaddr);

                    break;

                }

        }

        int opcode= m[execaddr] & 0b11111100;

        //n,i,x,b,p,e;

        int n=(m[execaddr]&0b00000010)>>1;

        int i=(m[execaddr]&0b00000001);

        int x=(m[execaddr+1]>>7) & 1;

        int b=(m[execaddr+1]>>6) & 1;

        int p=(m[execaddr+1]>>5) & 1;

        int e=(m[execaddr+1]>>4) & 1;

        operand=0;

        x=x;

        //1=extended 0=not extended

        int format=get\_format(opcode, hash);

        int len;

        if(opcode==0x44 || opcode==0xF0 || opcode==0x04){

            //BYTE directive

            if(opcode==0x44) len=3;

            else len=1;

        }

        else if(m[execaddr]==0x00){

            //empty reserved space by RESW or RESB

            len=1;

        }

        else if(format==1){

            //format 1

            len=1;

        }

        else if(format==2){

            //format 2

            len=2;

            operand=m[execaddr+1];

        }

        else if(format==3){

            //format 3/4

            if(e==1) {

                len=4;

                operand=operand|m[execaddr+3];

                operand=operand|(m[execaddr+2]<<8);

                operand=operand|((m[execaddr+1]%16)<<16);

            }

            else{

                len=3;

                operand=operand|m[execaddr+2];

                operand=operand|((m[execaddr+1]%16)<<8);

            }

            char hex[10];

            sprintf(hex, "%03X", operand);

            if (hex[0] == 9 || hex[0] == 8 || (hex[0] >= 'A' && hex[0] <= 'F')) {

                //if negative number

                operand = operand | 0xFFFFF000;

            }

            if(b==0 && p==1){

                //if PC

                if(n==1 && i==1){

                    //simple

                    operand+=(execaddr+len);

                }

                else if(n==1 && i==0){

                    //indirect

                    operand+=(execaddr+len);

                }

                else if(n==0 && i==1){

                    //immediate

                    operand+=(execaddr+len);

                }

            }

            if(b==0 && p==0){

                //if direct

                if(n==1 && i==1){

                }

                else if(n==1 && i==0){

                }

                else if(n==0 && i==1){

                }

            }

            if(b==1 && p==0){

                //if BASE

                if(n==1 && i==1){

                    //simple

                    operand+=BASE;

                }

                else if(n==1 && i==0){

                }

                else if(n==0 && i==1){

                }

            }

        }

        //--------------FORMATCHECKEND-------------------------------

        if(opcode==0x14){//STL

            m[operand]=(reg.L&0x00FF0000)>>16;

            m[operand+1]=(reg.L&0x0000FF00)>>8;

            m[operand+2]=(reg.L&0x000000FF);

        }

        if(opcode==0x68){ //LDB m

            reg.B=operand;

        }

        if(opcode==0x48){ //JSUB

            reg.L=execaddr+len;

            execaddr=reg.PC=operand;

            PC\_edit=true;

        }

        if(opcode==0x74){//LDT

            int t=0;

            t=t|(m[operand]<<16);

            t=t|(m[operand+1]<<8);

            t=t|(m[operand+2]);

            if(n==0 && i==1) t=operand;

            reg.T=t;

        }

        if(opcode==0x00){ //LDA

            int t=0;

            t=t|(m[operand]<<16);

            t=t|(m[operand+1]<<8);

            t=t|(m[operand+2]);

            if(n==0 && i==1) t=operand;

            reg.A=t;

        }

        if(opcode==0x0C){//STA

            m[operand]=(reg.A&0x00FF0000)>>16;

            m[operand+1]=(reg.A&0x0000FF00)>>8;

            m[operand+2]=(reg.A&0x000000FF);

        }

        if(opcode==0x28){//COMP

            if(reg.A>operand) CC=-1;

            else if(reg.A<operand) CC=1;

            else CC=0;

        }

        if(opcode==0x30){//JEQ

            if(CC==0){

                execaddr=reg.PC=operand;

                PC\_edit=true;

            }

        }

        if(opcode==0xB4){//CLEAR

            if(operand>>4==0) reg.A=0;

            if(operand>>4==1) reg.X=0;

            if(operand>>4==4) reg.S=0;

        }

        if(opcode==0xE0){//TD

            CC=-1;

        }

        if(opcode==0xD8){//RD

            next\_CC=true;//if next\_CC=true next COMPR CC is = (0)

        }

        if(opcode==0xA0){//COMPR

            //might make some inprovements

            if(next\_CC) CC=0;

            else{

                if(reg.A>reg.S) CC=1;

                else if(reg.A<reg.S) CC=-1;

                else CC=0;

            }

        }

        if(opcode==0x54){//STCH

             m[operand+reg.X]=reg.A&0xFF;

        }

        if(opcode==0xB8){//TIXR

            //might make some improvements

            reg.X++;

            if(reg.X<reg.T) CC=-1;

            else if(reg.X>reg.T) CC=1;

            else CC=0;

        }

        if(opcode==0x38){//JLT

            if(CC==-1){

                execaddr=reg.PC=operand;

                PC\_edit=true;

            }

        }

        if(opcode==0x10){//STX

            m[operand]=(reg.X&0x00FF0000)>>16;

            m[operand+1]=(reg.X&0x0000FF00)>>8;

            m[operand+2]=(reg.X&0x000000FF);

        }

        if(opcode==0x4C){//RSUB

            execaddr=reg.PC=reg.L;

            PC\_edit=true;

        }

        if(opcode==0x50){//LDCH

            reg.A=0;

            reg.A=reg.A|m[operand+reg.X];

        }

        if(opcode==0xDC){

            //WD do nothing

        }

        if(opcode==0x3C){

            if(n==1&&i==0){

               int t=0;

               t=t|(m[operand]<<16);

               t=t|(m[operand+1]<<8);

               t=t|(m[operand+2]);

               execaddr=reg.PC=t;

               PC\_edit=true;

            }

        }

        if(!PC\_edit){

            execaddr+=len;

            reg.PC=execaddr;

        }

        if(break\_flag){

            break;

        }

    }

    if(prog\_end){

        print\_regs();

        printf("            End Program\n");

        execaddr=progaddr;

        reg.A=reg.X=reg.PC=reg.B=reg.S=0;

        reg.L=proglength;

    }

    return 0;

}

/\*---------------------------------------------------------------------\*/

/\*Function: loader\*/

/\*Purpose: loads program\*/

/\*Return: 1 on success -1 on fail\*/

/\*---------------------------------------------------------------------\*/

int loader(char \*cmd){

    destroy\_EST(&est\_head);

    char inst[10], obj1[100], obj2[100], obj3[100];

    int file\_num = sscanf(cmd, "%s %s %s %s", inst, obj1, obj2, obj3);

    csaddr=progaddr;

    execaddr=progaddr;

    if(file\_num==2){

        if(!file\_exists(obj1)){

            printf("load error: no file\n"); return -1;

        }

        if(pass1(obj1)==-1) return -1;

        csaddr=progaddr; execaddr=progaddr;

        if(pass2(obj1)==-1) return -1;

        print\_EST(&est\_head);

        reg.L=proglength;

    }

    else if(file\_num==3){

        if(!file\_exists(obj1) || !file\_exists(obj2)){

            printf("load error: no file\n"); return -1;

        }

        if(pass1(obj1)==-1) return -1;

        if(pass1(obj2)==-1) return -1;

        csaddr=progaddr; execaddr=progaddr;

        if(pass2(obj1)==-1) return -1;

        if(pass2(obj2)==-1) return -1;

        print\_EST(&est\_head);

        reg.L=proglength;

    }

    else if(file\_num==4){

        if(!file\_exists(obj1) || !file\_exists(obj2) || !file\_exists(obj3)){

            printf("load error: no file\n"); return -1;

        }

        if(pass1(obj1)==-1) return -1;

        if(pass1(obj2)==-1) return -1;

        if(pass1(obj3)==-1) return -1;

        csaddr=progaddr; execaddr=progaddr;

        if(pass2(obj1)==-1) return -1;

        if(pass2(obj2)==-1) return -1;

        if(pass2(obj3)==-1) return -1;

        print\_EST(&est\_head);

        reg.L=proglength;

    }

    else{

        return -1;

    }

    return 0;

}

**loader.h**

#include "defines.h"

int loader(char \*cmd);

bool file\_exists(char \*filename);

int pass1(char \*filename);

int EST\_search(const char \*name, int type, struct ESTNODE \*\*est\_head);

void EST\_add(char \*csect, char \*symbol, int address, int length, int type,  struct ESTNODE \*\*est\_head);

void print\_bp();

void clear\_bp();

void add\_bp(int new\_bp);

int run(void);

void destroy\_EST(struct ESTNODE \*\*est\_head);