**과목명: 시스템프로그래밍**

**1분반**

**<<Project #3>>**

**서강대학교 컴퓨터공학과**

**20181643**

**손병호**

목차

1. 프로그램 개요
   1. 프로그램 소개
   2. 프로그램 요구사항
2. 프로그램 구조
   1. 프로그램 흐름도
   2. 프로그램 구성 파일
3. 모듈 정의

3.1 estab.c, estab.h

3.2 loader.c, loader.h

3.3 exec.c, exec.h

3.4 memory.c, memory.h

1. 전역변수
2. 코드

5.1 estab.c

5.2 loader.c

5.3 exec.c

5.4 memory.c

1. **프로그램 개요**

**1.1 프로그램 소개**

프로젝트 1,2에서 구현한 쉘(shell)에 linking 및 loading 기능을 추가한 프로그램이다. 또한 메모리에 올라간 프로그램을 시뮬레이션 할 수 있는 기능도 제공한다.

* 1. **프로그램 요구사항**

이미 제출한 프로젝트2에 아래의 명령어들을 추가한다.

1. progaddr
2. loader
3. run
4. bp
5. **프로그램 구조**

**2.1 프로그램 흐름도**

스크린샷이(가) 표시된 사진

자동 생성된 설명

**loader**

**exec**

**2.2 프로그램 구성 파일**

|  |  |
| --- | --- |
| **파일명** | **기능** |
| 20181643.out | 실행파일 |
| Makefile | 컴파일 스크립트 |
| 20181643.c, 20181643.h | 프로그램의 entry point |
| shell.c, shell.h | shell 관련 |
| memory.c, memory.h | memory 관련 |
| optab.c, optab.h | opcode table 관련 |
| utils.c, utils.h | parsing 관련 |
| linked\_list.c, linked\_list.h | linked list 관련 |
| symtab.c, symtab.h | symtab 관련 |
| assembler.c, assembler.h | assembler 관련 |
| loader.c, loader.h | loader 관련 |
| estab.c, estab.h | estab 관련 |
| exec.c, exec.h | run 관련 |

1. **모듈 정의**

**3.1 estab.c, estab.h**

ESTAB 관련 모듈이다.

* void add\_estab(ES\_NODE \*estab[20], char symbol[], int address, int length)
* estab(array of linked lists)에 (symbol, address, length) 형식의 노드를 삽입한다.
* ES\_NODE \*search\_estab(ES\_NODE \*estab[20], char symbol[])
* estab에서 symbol을 key로 가지는 노드를 찾은 후 리턴한다.
* void free\_estab(ES\_NODE \*estab[20])
* estab의 모든 노드를 제거 후 해제한다.
  1. **loader.c, loader.h**
* int loader(unsigned char mem[65546][16],ES\_NODE \*estab[20], int PROGADDR, int \*EXECADDR, char filename1[50], char filename2[50], char filename3[50])
* loader1과 loader2를 호출하는 wrapper이다.
* int loader1(ES\_NODE \*estab[20], int PROGADDR, int \*PROGLTH, char filename1[50], char filename2[50], char filename3[50])
* 다음의 방식으로 수행된다.

init CSADDR = PROGADDR

for each control section

set CSLTH from H record

insert CS name to ESTAB if not found

for each D record

for each symbol at loc

insert (symbol, CSADDR+loc) into ESTAB if not found

CSADDR += CSLTH

* int loader2(unsigned char mem[65536][16],ES\_NODE \*estab[20], int PROGADDR, char filename1[50], char filename2[50], char filename3[50]);
* 다음의 방식으로 수행된다.

CSADDR = PROGADDR

EXECADDR = PROGADDR

for each control section

set CSLTH from H record

for each line

if T then place content to CSADDR+loc

if M then modify content at CSADDR+loc

EXECADDR = (CSADDR+entry)

CSADDR += CSLTH

jump to EXECADDR

**3.3 exec.c, exec.h**

assembler 관련 함수를 구현한 파일이다. 세부 모듈은 아래와 같다.

* void run(unsigned char mem[65536][16], int PROGADDR, int PROGLTH, int reg[10], BP\_NODE \*breakpoint)
* loader에 의해 메모리에 올라간 프로그램을 EXECADDR부터 실행한다. 매 instruction을 수행할 때마다 레지스터 reg와 메모리mem의 상태를 변화시킨다. breakpoint가 존재할 경우 가장 처음 breakpoint에서 실행을 멈춘다. instruction을 4바이트 크기의 unsigned int로 받아와 비트 연산을 통해 디코딩하였다. instruction의 opcode에 따라 개별 의미에 맞는 연산을 수행하도록 코드를 작성하였다.
* void bp(BP\_NODE \*breakpoint, int address)
* 사용자로부터 입력받은 breakpoint들을 배열로 저장한다. breakpoint는 프로그램이 로드 된 이후에만 설정할 수 있다.

3.4 memory.c, memory.h

* unsigned int read\_hb(unsigned char mem[65536][16], int address, int length)
* 메모리의 address로부터 length만큼의 half-byte를 unsigned int로 읽어온다.
* void write\_hb(unsigned char mem[65536][16], int address, int length, unsigned int value)
* 메모리의 address로부터 length만큼의 half-byte에 value를 덮어쓴다.
* void modify\_hb(unsigned char mem[65536][16], int address, int length, int opmode, unsigned int operand)
* read\_hb, opmode(add,sub), write\_hb를 순서대로 합성한 것이다. 즉, 메모리의 특정 영역을 읽어와서 특정 연산을 수행한 후, 제자리에 덮어쓴다.

1. **전역변수 정의**

* ES\_NODE \*estab[20]
* array of linked list로, external symbol들을 (symbol, address, length) 형식의 노드로 저장한다. control section 또한 이곳에 저장된다.
* BP\_NODE breakpoint
* breakpoint들의 배열이다.
* int PROGADDR = 0
* 프로그램의 시작주소
* int EXECADDR = 0
* 프로그램의 진입점
* int PROGLTH = 0
* 프로그램의 길이
* int reg[10]
* 레지스터 배열

1. **코드**

프로젝트 3에서 추가한 부분만을 첨부하였으며, 상세 설명은 주석으로 작성하였다.

* 1. estab.c

**#include "estab.h"**

**void add\_estab(ES\_NODE \*estab[20], char symbol[], int address, int length) {**

**/\***

**\* adds (symbol, address,length) into symtab**

**\* Arguments:**

**\* ES\_NODE \*estab[] - array of 20 linked lists**

**\* char symbol[]**

**\* int address - for ordinary symbols**

**\* int length - for control sections**

**\*/**

**int bucket;**

**/\* make ES\_NODE \*/**

**ES\_NODE\* item = (ES\_NODE\*)malloc(sizeof(ES\_NODE));**

**strcpy(item->symbol,symbol);**

**item->address = address;**

**item->length = length;**

**/\* get bucket id \*/**

**bucket = hash(symbol);**

**/\* push item in front of symtab \*/**

**item->link = estab[bucket];**

**estab[bucket] = item;**

**}**

**ES\_NODE \*search\_estab(ES\_NODE \*estab[20], char symbol[]) {**

**/\***

**\* searches the node containing the given symbol**

**\* Arguments:**

**\* ES\_NODE \*Estab[] - array of 20 linked lists**

**\* char symbol[] - the key for the search**

**\* Returns:**

**\* the pointer of the found node, and NULL if not found**

**\*/**

**int bucket = hash(symbol);**

**ES\_NODE \*curr = estab[bucket];**

**while(curr) {**

**if(!strcmp(symbol,curr->symbol)) {**

**/\* found \*/**

**return curr;**

**}**

**curr = curr->link;**

**}**

**/\* not found \*/**

**return NULL;**

**}**

**void free\_estab(ES\_NODE \*estab[20]) {**

**/\***

**\* free estab**

**\* Arguments:**

**\* ES\_NODE \*estab[]**

**\*/**

**ES\_NODE \*curr, \*tmp;**

**for(int i=0; i<20; i++) {**

**curr = estab[i];**

**while(curr) {**

**tmp = curr;**

**curr = curr->link;**

**free(tmp);**

**}**

**estab[i] = NULL;**

**}**

**}**

**void print\_estab(ES\_NODE \*estab[20]) {**

**ES\_NODE \*curr;**

**int arrow; /\* flag for printing arrow \*/**

**for(int i=0; i<20; i++) {**

**curr = estab[i];**

**printf("%d : ",i);**

**arrow = 0;**

**while(curr) {**

**if(arrow) printf(" -> ");**

**printf("[%s,%.2X, %.2X]", curr->symbol, curr->address, curr->length);**

**curr = curr->link;**

**arrow = 1;**

**}**

**printf("\n");**

**}**

**}**

* 1. loader.c

**#include "loader.h"**

**int loader(unsigned char mem[65536][16], ES\_NODE \*estab[20], int PROGADDR, int \*EXECADDR, char filename1[50], char filename2[50], char filename3[50]) {**

**int PROGLTH=0;**

**int idx=0;**

**/\* file error handling - filetype \*/**

**if(strlen(filename1)) {**

**idx=0;**

**while(filename1[idx]!='.') {**

**if(filename1[idx]=='\0') {**

**printf("error! invalid filetype\n");**

**return -1;**

**}**

**idx++;**

**}**

**if(strcmp(&filename1[idx],".obj")) {**

**printf("error! invalid filetype\n");**

**return -1;**

**}**

**}**

**if(strlen(filename2)) {**

**idx=0;**

**while(filename2[idx]!='.') {**

**if(filename2[idx]=='\0') {**

**printf("error! invalid filetype\n");**

**return -1;**

**}**

**idx++;**

**}**

**if(strcmp(&filename2[idx],".obj")) {**

**printf("error! invalid filetype\n");**

**return -1;**

**}**

**}**

**if(strlen(filename3)) {**

**idx=0;**

**while(filename3[idx]!='.') {**

**if(filename3[idx]=='\0') {**

**printf("error! invalid filetype\n");**

**return -1;**

**}**

**idx++;**

**}**

**if(strcmp(&filename3[idx],".obj")) {**

**printf("error! invalid filetype\n");**

**return -1;**

**}**

**}**

**free\_estab(estab);**

**if(loader1(estab,PROGADDR,&PROGLTH,filename1,filename2,filename3)==-1) return -1;**

**if((\*EXECADDR=loader2(mem,estab,PROGADDR,filename1,filename2,filename3))==-1) return -1;**

**return PROGLTH;**

**}**

**int loader1(ES\_NODE \*estab[20], int PROGADDR, int \*PROGLTH, char filename1[50], char filename2[50], char filename3[50]) {**

**/\* pass1 \*/**

**int CSADDR,CSLTH;**

**char CSNAME[10];**

**FILE \*fp[3];**

**char line[100];**

**int idx;**

**char tok[50], tok2[50];**

**int a,b;**

**fp[0] = fopen(filename1,"r");**

**fp[1] = fopen(filename2,"r");**

**fp[2] = fopen(filename3,"r");**

**/\* file error handling - no such file \*/**

**if(strlen(filename1) && !fp[0]) {**

**printf("file doens't exist!\n");**

**return -1;**

**}**

**if(strlen(filename2) && !fp[1]) {**

**printf("file doens't exist!\n");**

**return -1;**

**}**

**if(strlen(filename3) && !fp[2]) {**

**printf("file doens't exist!\n");**

**return -1;**

**}**

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

**printf("section name\n");**

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

**CSADDR = PROGADDR;**

**for(int i=0; i<3; i++) { /\* for each file \*/**

**if(!fp[i]) continue;**

**/\* get CSNAME, CSLTH from header \*/**

**fscanf(fp[i],"%[^\n]s",line); fgetc(fp[i]); /\* read header line \*/**

**idx=1; get\_token(line,CSNAME,&idx,8); /\* get CSNAME \*/**

**tok[6]='\0'; for(idx=13;idx<=18;idx++)tok[idx-13]=line[idx]; /\* get CSLTH \*/**

**CSLTH = parse\_op(tok);**

**/\* insert CSNAME to estab \*/**

**printf("%-9s %-9X%.4X\n",CSNAME,CSADDR,CSLTH);**

**add\_estab(estab,CSNAME,CSADDR,CSLTH);**

**/\* read each D record \*/**

**for(int j=0; j<100; j++) line[j]='\0';**

**while(fscanf(fp[i],"%[^\n]s",line)!=EOF) {**

**fgetc(fp[i]);**

**if(line[0]=='D') {**

**idx=1;**

**do {**

**/\* symbol \*/**

**get\_substr(line,tok,idx,idx+5);**

**idx += 6;**

**/\* loc \*/**

**tok2[6]='\0'; for(int j=idx;j<idx+6;j++) tok2[j-idx]=line[j];**

**b = parse\_op(tok2);**

**idx += 6;**

**/\* insert the symbol to estab \*/**

**printf(" %-8s%-8X\n",tok,CSADDR+b);**

**add\_estab(estab,tok,CSADDR+b,-1);**

**} while(line[idx]!='\0');**

**}**

**for(int j=0; j<100; j++) line[j]='\0';**

**}**

**CSADDR += CSLTH;**

**}**

**\*PROGLTH = CSADDR - PROGADDR;**

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

**printf("\t\ttotal length %.4X\n",\*PROGLTH);**

**if(fp[0]) fclose(fp[0]);**

**if(fp[1]) fclose(fp[1]);**

**if(fp[2]) fclose(fp[2]);**

**return 0;**

**// takes PROGADDR**

**// init CSADDR = PROGADDR**

**// for each control section**

**// set CSLTH from H record**

**// insert CS name to ESTAB if not found**

**// for each D record**

**// for each symbol at loc, insert (symbol, CSADDR+loc) into ESTAB if not found**

**// CSADDR += CSLTH**

**}**

**int loader2(unsigned char mem[65536][16], ES\_NODE \*estab[20], int PROGADDR, char filename1[50], char filename2[50], char filename3[50]) {**

**/\* pass 2 \*/**

**int CSADDR,CSLTH,EXECADDR;**

**char CSNAME[10];**

**FILE \*fp[3];**

**char line[100];**

**int idx;**

**char tok[50], tok2[50];**

**int a,b;**

**int reftab[100]; /\* for the static indexing of R records \*/**

**int loc;**

**ES\_NODE \*es0;**

**fp[0] = fopen(filename1,"r");**

**fp[1] = fopen(filename2,"r");**

**fp[2] = fopen(filename3,"r");**

**CSADDR = PROGADDR;**

**EXECADDR = PROGADDR;**

**for(int i=0; i<3; i++) { /\* for each file \*/**

**if(!fp[i]) continue;**

**/\* get CSNAME, CSLTH from header \*/**

**fscanf(fp[i],"%[^\n]s",line); fgetc(fp[i]); /\* read header line \*/**

**idx=1; get\_token(line,CSNAME,&idx,8); /\* get CSNAME \*/**

**reftab[1] = CSADDR;**

**tok[6]='\0'; for(idx=13;idx<=18;idx++)tok[idx-13]=line[idx]; /\* get CSLTH \*/**

**CSLTH = parse\_op(tok);**

**for(int j=0; j<100; j++) line[j]='\0';**

**while(fscanf(fp[i],"%[^\n]s",line)!=EOF) {**

**fgetc(fp[i]);**

**if(line[0]=='T') {**

**tok[6]='\0'; for(idx=1;idx<=6;idx++)tok[idx-1]=line[idx]; /\* get loc \*/**

**loc = parse\_op(tok) + CSADDR;**

**/\* get length \*/**

**tok[2]='\0'; for(idx=7;idx<=8;idx++)tok[idx-7]=line[idx];**

**a = parse\_op(tok); /\* record length \*/**

**for(int j=0; j<a; j++) {**

**/\* read byte \*/**

**tok[2]='\0'; for(int k=idx;k<idx+2;k++) tok[k-idx]=line[k];**

**b = parse\_op(tok);**

**edit(mem,loc,parse\_op(tok));**

**idx += 2; loc += 1;**

**}**

**}**

**else if(line[0]=='M') {**

**tok[6]='\0'; for(idx=1;idx<=6;idx++)tok[idx-1]=line[idx]; /\* get loc \*/**

**loc = parse\_op(tok) + CSADDR;**

**/\* get length \*/**

**tok[2]='\0'; for(idx=7;idx<=8;idx++)tok[idx-7]=line[idx];**

**a = parse\_op(tok); /\* content length \*/**

**/\* get operand \*/**

**tok[2]='\0'; for(idx=10;idx<=11;idx++)tok[idx-10]=line[idx];**

**b = reftab[parse\_op(tok)];**

**if(line[9]=='+') {**

**modify\_hb(mem,loc,a,1,b);**

**} else if(line[9]=='-') {**

**modify\_hb(mem,loc,a,0,b);**

**}**

**}**

**else if(line[0]=='R') {**

**idx=1;**

**do {**

**/\* reference number \*/**

**tok2[2]='\0'; for(int j=idx;j<idx+2;j++) tok2[j-idx]=line[j];**

**b = parse\_op(tok2);**

**idx += 2;**

**/\* symbol \*/**

**get\_substr(line,tok,idx,idx+5);**

**idx += 6;**

**/\* insert the symbol to reftab \*/**

**if((es0=search\_estab(estab,tok))) reftab[b] = es0->address;**

**else {**

**printf("%s : undefined symbol\n",tok);**

**if(fp[0])fclose(fp[0]);if(fp[1])fclose(fp[1]);if(fp[2])fclose(fp[2]);**

**return -1;**

**}**

**} while(line[idx]!='\0');**

**}**

**else if(line[0]=='E'&&line[1]=='0') {**

**tok[6]='\0'; for(idx=1;idx<=6;idx++)tok[idx-1]=line[idx]; /\* get loc \*/**

**EXECADDR = parse\_op(tok) + CSADDR;**

**}**

**for(int j=0; j<100; j++) line[j]='\0';**

**}**

**CSADDR += CSLTH;**

**}**

**if(fp[0]) fclose(fp[0]);**

**if(fp[1]) fclose(fp[1]);**

**if(fp[2]) fclose(fp[2]);**

**return EXECADDR;**

**// CSADDR = PROGADDR**

**// EXECADDR = PROGADDR**

**// for each control section**

**// set CSLTH from H record**

**// for each line**

**// if T then place content to CSADDR+loc**

**// if M then modify content at CSADDR+loc**

**// EXECADDR = (CSADDR+entry)**

**// CSADDR += CSLTH**

**// jump to EXECADDR**

**}**

* 1. exec.c

**#include "exec.h"**

**void bp(BP\_NODE \*breakpoint, int address) {**

**breakpoint->arr[breakpoint->size] = address;**

**(breakpoint->size)++;**

**printf("\t\t[ok] create breakpoint %X\n",address);**

**}**

**void run(unsigned char mem[65536][16], int PROGADDR, int PROGLTH, int reg[10], BP\_NODE \*breakpoint) {**

**/\* A:0 , X:1 , L:2 , B:3 , S:4 , T:5 , F:6 , PC:8 , SW:9 \*/**

**unsigned int buffer,inst,opcode,reg1,reg2,addr;**

**int n,i,x,b,p,e; /\* bit flags \*/**

**int quit=0;**

**int PC;**

**for(int j=0; j<breakpoint->size; j++) {**

**if(reg[8] == PROGADDR + breakpoint->arr[j]) {**

**quit=1;**

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

**printf("A : %.6X X : %.6X\n",reg[0],reg[1]);**

**printf("L : %.6X PC : %.6X\n",reg[2],reg[8]);**

**printf("B : %.6X S : %.6X\n",reg[3],reg[4]);**

**printf("T : %.6X\n",reg[5]);**

**printf("\t\tStop at checkpoint[%X]\n",breakpoint->arr[j]);**

**breakpoint->arr[j] = -1;**

**}**

**}**

**if(reg[8]>=PROGADDR+PROGLTH) {**

**quit=1;**

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

**printf("A : %.6X X : %.6X\n",reg[0],reg[1]);**

**printf("L : %.6X PC : %.6X\n",reg[2],reg[8]);**

**printf("B : %.6X S : %.6X\n",reg[3],reg[4]);**

**printf("T : %.6X\n",reg[5]);**

**printf("\t\tEnd Program\n");**

**breakpoint->size=0;**

**}**

**while (!quit) {**

**inst = read\_hb(mem,reg[8],8);**

**opcode = (inst & 0xFC000000) >> 24;**

**reg1 = (inst & 0x00F00000) >> 20;**

**reg2 = (inst & 0x000F0000) >> 16;**

**n = (inst & 0x02000000) >> 25;**

**i = (inst & 0x01000000) >> 24;**

**x = (inst & 0x00800000) >> 23;**

**b = (inst & 0x00400000) >> 22;**

**p = (inst & 0x00200000) >> 21;**

**e = (inst & 0x00100000) >> 20;**

**/\* relative addressing for format 3 \*/**

**addr = (inst & 0x000FFF00) >> 8; /\* get disp \*/**

**if(addr>>11==1) addr |= 0xFFFFF000; /\* sign extend disp \*/**

**if(b) addr += (PROGADDR + reg[3]); /\* base relative \*/**

**else if(p) addr += (PROGADDR + 3 + reg[8]); /\* pc relative \*/**

**if(x) addr += reg[1]; /\* index mode \*/**

**/\* direct addressing for format 4 \*/**

**if(e) {**

**addr = inst & 0x000FFFFF;**

**addr += PROGADDR;**

**}**

**/\* FORMAT 2 \*/**

**if(opcode==0xB4) { /\* CLEAR \*/**

**reg[8] += 2;**

**reg[reg1] = reg[reg2] = 0;**

**//dpc += 2;**

**}**

**else if(opcode==0xA0) { /\* COMPR \*/**

**reg[8] += 2; /\* increment PC \*/**

**reg[9] = (reg[reg1] - reg[reg2]); /\* set CC \*/**

**}**

**else if(opcode==0xB8) { /\* TIXR \*/**

**reg[8] += 2; /\* increment PC \*/**

**reg[1]++; /\* X <- X + 1 \*/**

**reg[9] = (reg[1] - reg[reg1]); /\* (X) : (r1) \*/**

**}**

**/\* FORMAT 3 \*/**

**else if(opcode==0x28) { /\* COMP \*/**

**reg[8] += (e ? 4 : 3); /\* increment PC \*/**

**reg[9] = reg[0] - read\_hb(mem,addr,6); /\* (A) : (m..m+2) \*/**

**if(!n) reg[9] = reg[0] - addr; /\* if immediate \*/**

**//printf("i: %d, comp %X and %X => CC = %X\n",i,reg[0],addr,reg[9]);**

**}**

**else if(opcode==0x3C) { /\* J \*/**

**reg[8] += (e ? 4 : 3); /\* increment PC \*/**

**reg[8] = addr; /\* PC <- m \*/**

**if(!i) reg[8] = read\_hb(mem,addr,5); /\* if indirect \*/**

**}**

**else if(opcode==0x30) { /\* JEQ \*/**

**reg[8] += (e ? 4 : 3); /\* increment PC \*/**

**if(!reg[9]) reg[8] = addr; /\* PC <- m if CC set to = \*/**

**}**

**else if(opcode==0x38) { /\* JLT \*/**

**reg[8] += (e ? 4 : 3); /\* increment PC \*/**

**if(reg[9]<0) reg[8] = addr; /\* PC <- m if CC set to < \*/**

**//printf("(jlt) addr:%X\n",addr);**

**}**

**else if(opcode==0x48) { /\* JSUB \*/**

**reg[8] += (e ? 4 : 3); /\* increment PC \*/**

**reg[2] = reg[8]; /\* L <- (PC) \*/**

**reg[8] = addr; /\* PC <- m \*/**

**}**

**else if(opcode==0x00) { /\* LDA \*/**

**reg[8] += (e ? 4 : 3); /\* increment PC \*/**

**reg[0] = !n ? addr : (buffer=read\_hb(mem,addr,6)); /\* A <- (m..m+2) \*/**

**// printf("read: %X, i: %d, addr: %X, reg[0]:%X\n",buffer,i,addr,reg[0]);**

**// printf("inst: %X\n",inst);**

**}**

**else if(opcode==0x68) { /\* LDB \*/**

**reg[8] += (e ? 4 : 3); /\* increment PC \*/**

**reg[3] = !n ? addr : read\_hb(mem,addr,6); /\* B <- (m..m+2) \*/**

**}**

**else if(opcode==0x50) { /\* LDCH \*/**

**reg[8] += (e ? 4 : 3); /\* increment PC \*/**

**reg[0] = read\_hb(mem,addr,2); /\* A <- (m) \*/**

**reg[0] &= 0xFF; /\* A <- A[rightmost byte] \*/**

**}**

**else if(opcode==0x74) { /\* LDT \*/**

**reg[8] += (e ? 4 : 3); /\* increment PC \*/**

**reg[5] = !n ? addr : read\_hb(mem,addr,6); /\* T <- (m..m+2) \*/**

**//printf("(ldt) addr: %X\n",addr);**

**}**

**else if(opcode==0xD8) { /\* RD \*/**

**reg[8] += (e ? 4 : 3); /\* increment PC \*/**

**}**

**else if(opcode==0x4C) { /\* RSUB \*/**

**reg[8] += (e ? 4 : 3); /\* increment PC \*/**

**reg[8] = reg[2]; /\* PC <- (L) \*/**

**}**

**else if(opcode==0x0C) { /\* STA \*/**

**reg[8] += (e ? 4 : 3); /\* increment PC \*/**

**buffer = reg[0] & 0xFFFFFF; /\* buffer <- (A) \*/**

**write\_hb(mem,addr,6,buffer); /\* m <- (A) \*/**

**}**

**else if(opcode==0x54) { /\* STCH \*/**

**reg[8] += (e ? 4 : 3); /\* increment PC \*/**

**buffer = reg[0] & 0xFF; /\* rightmost byte of (A) \*/**

**write\_hb(mem,addr,2,buffer); /\* m <- (A) rightmost byte \*/**

**}**

**else if(opcode==0x14) { /\* STL \*/**

**reg[8] += (e ? 4 : 3); /\* increment PC \*/**

**buffer = reg[2] & 0xFFFFFF; /\* buffer <- (L) \*/**

**//printf("(stl) addr: %X, buffer: %X\n",addr,buffer);**

**write\_hb(mem,addr,6,buffer); /\* m <- (L) \*/**

**}**

**else if(opcode==0x10) { /\* STX \*/**

**reg[8] += (e ? 4 : 3); /\* increment PC \*/**

**buffer = reg[1] & 0xFFFFFF; /\* buffer <- (X) \*/**

**write\_hb(mem,addr,6,buffer); /\* m..m+2 <- buffer \*/**

**}**

**else if(opcode==0xE0) { /\* TD \*/**

**reg[8] += (e ? 4 : 3); /\* increment PC \*/**

**reg[9] = -1; /\* CC set to < \*/**

**}**

**else if(opcode==0xDC) { /\* WD \*/**

**reg[8] += (e ? 4 : 3); /\* increment PC \*/**

**}**

**for(int j=0; j<breakpoint->size; j++) {**

**if(reg[8] == PROGADDR + breakpoint->arr[j]) {**

**quit=1;**

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

**printf("A : %.6X X : %.6X\n",reg[0],reg[1]);**

**printf("L : %.6X PC : %.6X\n",reg[2],reg[8]);**

**printf("B : %.6X S : %.6X\n",reg[3],reg[4]);**

**printf("T : %.6X\n",reg[5]);**

**printf("\t\tStop at checkpoint[%X]\n",breakpoint->arr[j]);**

**breakpoint->arr[j] = -1;**

**}**

**}**

**if(reg[8]>=PROGADDR+PROGLTH) {**

**quit=1;**

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

**printf("A : %.6X X : %.6X\n",reg[0],reg[1]);**

**printf("L : %.6X PC : %.6X\n",reg[2],reg[8]);**

**printf("B : %.6X S : %.6X\n",reg[3],reg[4]);**

**printf("T : %.6X\n",reg[5]);**

**printf("\t\tEnd Program\n");**

**breakpoint->size=0;**

**}**

**}**

**}**

* 1. memory.c

**unsigned int read\_hb(unsigned char mem[65536][16], int address, int length) {**

**/\***

**\* read memory in half-bytes and wrap into unsigned int**

**\* Arguments:**

**\* char mem[][]**

**\* int address - the smallest address containing at least one of the desired half-bytes**

**\* int length - the number of half-bytes desired**

**\*/**

**unsigned int buffer=0,ret=0;**

**int n = length/2;**

**int idx = 0;**

**if(length%2) { /\* if length is odd \*/**

**buffer = mem[address/16][address%16]; /\* read byte \*/**

**buffer &= 0xF; /\* get rid of upper halfbyte \*/**

**buffer <<= 8\*n; /\* match the digit \*/**

**ret |= buffer; /\* put it to ret \*/**

**//buffer = 0; /\* clear buffer \*/**

**address++; /\* regard it as if it were even in the first place \*/**

**}**

**while(idx<n) { /\* for each byte \*/**

**buffer = mem[(address+idx)/16][(address+idx)%16]; /\* read byte \*/**

**buffer <<= 8\*(n-1-idx); /\* match the digit \*/**

**ret |= buffer; /\* put it to ret \*/**

**//buffer = 0; /\* clear buffer \*/**

**idx++;**

**}**

**return ret;**

**}**

**void write\_hb(unsigned char mem[65536][16], int address, int length, unsigned int value) {**

**/\***

**\* inverse operation of read\_hb()**

**\* Arguments:**

**\* char mem[][]**

**\* int address - the smallest address containing at least one of the desired half-bytes**

**\* int length - the number of half-bytes desired**

**\* unsigned int value - value to write**

**\*/**

**unsigned int mask,buffer;**

**int n = length/2;**

**int idx = 0;**

**if(length%2) { /\* if length is odd \*/**

**mask = 0xF; /\* mask for halfbyte \*/**

**mask <<= 8\*n; /\* match the digit \*/**

**buffer = value&mask; /\* get the desired halfbyte \*/**

**buffer >>= 8\*n; /\* put it in the rightmost \*/**

**mem[address/16][address%16] &= 0xF0; /\* clear mem \*/**

**mem[address/16][address%16] |= (unsigned char)buffer; /\* write halfbyte \*/**

**address++; /\* regard it as if it were even in the first place \*/**

**}**

**while(idx<n) { /\* for each byte \*/**

**mask = 0xFF; /\* mask for byte \*/**

**mask <<= 8\*(n-1-idx); /\* match the digit \*/**

**buffer = value&mask; /\* get the desired halfbyte \*/**

**buffer >>= 8\*(n-1-idx); /\* put it in the rightmost \*/**

**mem[(address+idx)/16][(address+idx)%16] = (unsigned char)buffer; /\* write halfbyte \*/**

**idx++;**

**}**

**}**

**void modify\_hb(unsigned char mem[65536][16], int address, int length, int opmode, unsigned int operand) {**

**/\***

**\* the composite (write\_hb) \* (opmode) \* (read\_hb)**

**\* Arguments:**

**\* int opmode -- add if 1, sub if 0**

**\*/**

**unsigned int content;**

**content = read\_hb(mem,address,length);**

**content = (opmode ? (content+operand) : (content-operand));**

**write\_hb(mem,address,length,content);**

**}**