**과목명: 시스템프로그래밍**

**분반**

**<<Project #3>>**

**서강대학교 수학과**

**20141218**

**김찬**

목 차

1. **프로그램 개요**

**2 코드 설명**

1. **프로그램 개요**

SIC/XE Machine Linking loader를 구현한다. Control section 단위로 나뉜 여러 object file을 linking하여 메모리에 load하고 loading된 프로그램을 실행하고 break point를 통해 디버그를 한다.

**2.코드**

**“20141218.c”**

#include "20141218.h"

int locationflag = 0;

int menuflag; //명령어 값

int edit\_address, edit\_value; //edit 명령어 시 주소와 value 값

int hash\_value; //hash 함수 return 값

int start, end; //dump 명령어 시 시작 주소와 끝 주소 값.

int fill\_start, fill\_end, fill\_value; //fill 명령어 시 시작, 끝 주소와 value 값

int last\_memory; //dump 입력시 마지막 주소 값

int basevalue; //base 값

int locstart, loclength; //시작 주소값

int location[100], line[100];//location, line 저장 값

int registers[10];

char memory[65536][16]; //메모리

int progaddr,progaddr\_end, progaddr\_link, progaddr\_load; // loader 또는 run명령어를 수행할 때 시작하는 주소

LinkHistory \*Head, \*Tail; //History Linked List

LinkOpcode \*OpHead[20], \*OpTail[20]; //Opcode Hash Linked List

Estab \*EsHead[20], \*EsTail[20]; //ESTAB

Symbol \*SymHead, \*SymTail; //Symbol Linked List

Modify\* ModifyHead, \* ModifyTail; //Modification Linked List

BP \*BPHead, \* BPTail;

char inputs[12] = " SIC/XE\0\0"; // virtual input device for testing copy.obj

char outputs[13] = { '\0' };

int in\_idx, out\_idx;

//main 함수

int main() {

// 초기화

progaddr = 0;

last\_memory = -1;

Head = NULL, Tail = NULL;

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

OpHead[i] = NULL, OpTail[i] = NULL;

}

makehash();

while (1) {

menuflag = 0;

menu();

if (menuflag == 1) menu\_help();

else if (menuflag == 3) break;

else if (menuflag == 4) menu\_history();

else if (menuflag == 5) menu\_dump();

else if (menuflag == 6) menu\_edit();

else if (menuflag == 7) menu\_fill();

else if (menuflag == 8) menu\_reset();

else if (menuflag == 9) menu\_opcode();

else if (menuflag == 10) menu\_opcodelist();

else if (menuflag == 13) menu\_symbol();

else if (menuflag == 16) printbp();

else if (menuflag == 17) clearbp();

else if (menuflag == 19) runobj();

}

return 0;

}

//16진수를 10진수로 바꾸는 함수

int makedecimal(char\* address) {

int result = 0;

int len = strlen(address);

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

result \*= 16;

if (address[i] >= '0' && address[i] <= '9') {

result += address[i] - '0';

}

//16진수 대소문자 모두 가능

else if (address[i] >= 'A' && address[i] <= 'F') {

result += (address[i] - 'A') + 10;

}

else if (address[i] >= 'a' && address[i] <= 'f') {

result += (address[i] - 'a') + 10;

}

else {

result = -1;

break;

}

}

return result;

}

//명령어를 입력받고 올바른 명령어인지 확인 후 맞다면 history에 추가 그렇지 않다면 에러 체크.

void menu() {

LinkHistory\* TempHistory;

LinkOpcode\* TempHashHead;

BP\* TempBP;

char instr[100], newinstr1[100], typecontext[100], asminstr[100];

char\* newinstr;

int len, error = 0, hashindex;

int bp\_address = 0;

printf("sicsim> ");

fgets(instr, sizeof(instr), stdin);

len = strlen(instr);

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

strcpy(newinstr1, instr);

newinstr = strtok(newinstr1, " ");

if (newinstr != NULL) {

if (strcmp(newinstr, "h") == 0 || strcmp(newinstr, "help") == 0) {

menuflag = 1;

}

else if (strcmp(newinstr, "d") == 0 || strcmp(newinstr, "dir") == 0) {

menuflag = 2;

}

else if (strcmp(newinstr, "q") == 0 || strcmp(newinstr, "quit") == 0) {

menuflag = 3;

}

else if (strcmp(newinstr, "hi") == 0 || strcmp(newinstr, "history") == 0) {

menuflag = 4;

}

else if (strcmp(newinstr, "du") == 0 || strcmp(newinstr, "dump") == 0) {

start = -1, end = -1;

newinstr = strtok(NULL, " ,");

if (newinstr == NULL) menuflag = 5; //명령어에 dump 입력 시

else {

start = makedecimal(newinstr);

newinstr = strtok(NULL, " ");

if (!newinstr) { //명령어에 dump start 입력 시

if (start >= 0 && start < 65536 \* 16) menuflag = 5; //start주소가 0과 2^20사이면 올바른 명령어

else error = 1;

}

else { //명령어에 dump start, end 입력 시

end = makedecimal(newinstr);

if (start >= 0 && start < 65536 \* 16 && end >= 0 && end < 65536 \* 16) { //start, end 주소가 0과 2^20사이이고

if (start <= end) menuflag = 5; //start주소가 end주소 이하면 올바른 명령어

else error = 2;

}

else

error = 1;

}

}

}

else if (strcmp(newinstr, "e") == 0 || strcmp(newinstr, "edit") == 0) {

newinstr = strtok(NULL, " ,");

edit\_address = makedecimal(newinstr);

newinstr = strtok(NULL, " ");

edit\_value = makedecimal(newinstr);

if (edit\_address >= 0 && edit\_address < 16 \* 65536 && edit\_value >= 0 && edit\_value < 256) menuflag = 6;

else error = 1;

}

else if (strcmp(newinstr, "f") == 0 || strcmp(newinstr, "fill") == 0) {

newinstr = strtok(NULL, " ,");

fill\_start = makedecimal(newinstr);

newinstr = strtok(NULL, " ,");

fill\_end = makedecimal(newinstr);

newinstr = strtok(NULL, " ");

fill\_value = makedecimal(newinstr);

if (fill\_start >= 0 && fill\_start < 16 \* 65536 && fill\_end >= 0 && fill\_end < 16 \* 65536 && fill\_value >= 0 && fill\_value < 256) {

if (fill\_start <= fill\_end) menuflag = 7;

else error = 2;

}

else

error = 1;

}

else if (strcmp(newinstr, "reset") == 0) menuflag = 8;

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

hash\_value = -1;

newinstr = strtok(NULL, " ");

hashindex = hash(newinstr);

for (TempHashHead = OpHead[hashindex]; TempHashHead != NULL; TempHashHead = TempHashHead->nextOpcode) {

//해당 hashindex값이 hashtable안에 있으면 올바른 명령어

if (strcmp(newinstr, TempHashHead->Opcode) == 0) {

hash\_value = TempHashHead->Value;

break;

}

}

if (hash\_value != -1) menuflag = 9;

else error = 3;

}

else if (strcmp(newinstr, "opcodelist") == 0) menuflag = 10;

else if (strcmp(newinstr, "assemble") == 0) {

newinstr = strtok(NULL, " ");

FILE\* fileasm = fopen(newinstr, "r");

if (fileasm) {

if (!assemble(fileasm)) {

fclose(fileasm);

fileasm = fopen(newinstr, "r");

assemble1(fileasm);

menuflag = 11;

}

}

}

else if (strcmp(newinstr, "type") == 0) {

newinstr = strtok(NULL, " ");

FILE\* typefile = fopen(newinstr, "r");

if (typefile) {

while (fgets(typecontext, sizeof(typecontext), typefile) != NULL) {

len = strlen(typecontext);

for (int i = 0; i < len; i++) printf("%c", typecontext[i]);

menuflag = 12;

}

}

else error = 4;

}

else if (strcmp(newinstr, "symbol") == 0) {

menuflag = 13;

}

else if (strcmp(newinstr, "progaddr") == 0) {

newinstr = strtok(NULL, " ");

progaddr = strtol(newinstr, NULL, 16);

progaddr\_link = progaddr\_load = progaddr;

menuflag = 14;

}

else if (strcmp(newinstr, "loader") == 0) {

for (int i = 0; i < 20; i++) EsHead[i] = NULL;

for (newinstr = strtok(NULL, " "); newinstr != NULL; newinstr = strtok(NULL, " ")) link(newinstr);

progaddr\_link = progaddr;

for (int i = 0; i < len; i++) newinstr1[i] = instr[i];

newinstr1[len] = NULL;

newinstr = strtok(newinstr1, " ");

progaddr = progaddr\_load;

for (newinstr = strtok(NULL, " "); newinstr != NULL; newinstr = strtok(NULL, " ")) loader(newinstr);

menuflag = 15;

printload();

}

else if (strcmp(newinstr, "bp") == 0) {

newinstr = strtok(NULL, " ");

if (newinstr == NULL) menuflag = 16;

else if (newinstr != NULL) {

if (strcmp(newinstr, "clear") == 0) {

menuflag = 17;

}

else {

bp\_address = strtol(newinstr, NULL, 16);

for (int i = 0; i < 4; i++) newinstr1[i] = newinstr[i];

newinstr1[4] = NULL;

newinstr = strtok(NULL, " ");

if (bp\_address >= 0 && newinstr == NULL) {

TempBP = (BP\*)malloc(sizeof(BP));

TempBP->address = bp\_address;

TempBP->nextBP = NULL;

if (BPHead == NULL) {

BPHead = TempBP;

BPTail = TempBP;

}

else {

BPTail->nextBP = TempBP;

TempBP = BPTail;

}

printf("\t[ok] create breapoint %s\n", newinstr1); // 메세지 출력

menuflag = 18;

}

}

}

}

else if (strcmp(newinstr, "run") == 0) menuflag = 19;

}

//명령어가 유효하면 history linked list에 삽입.

if (menuflag) {

TempHistory = (LinkHistory\*)malloc(sizeof(LinkHistory));

TempHistory->nextHistory = NULL;

for (int i = 0; i < len; i++) TempHistory->Historyins[i] = instr[i];

if (!Head) {

Head = TempHistory;

Tail = TempHistory;

}

else {

Tail->nextHistory = TempHistory;

Tail = TempHistory;

}

}

//그렇지 않다면 에러표시

else {

if (error == 0) printf("ERROR : WRONG INSTRUCTION!!\n");

else if (error == 1) printf("ERROR : OUT OF RANGE!!\n");

else if (error == 2) printf("ERROR : START IS GREATER THAN END!!\n");

else if (error == 3) printf("ERROR : MNEMONIC DOESN'T EXIST!!\n");

else if (error == 4) printf("ERROR : FILE DOESN'T EXIST!!\n");

}

}

int hextodec(char\* s) {

int i, ret;

int len;

ret = 0;

len = strlen(s);

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

{

ret = ret \* 16;

if (s[i] >= '0' && s[i] <= '9')// 변경 및 전달된 16진수 범위 검사

{

ret = ret + (int)(s[i] - '0');

}

else if (s[i] >= 'A' && s[i] <= 'F')

{

ret = ret + (int)(s[i] - 'A') + 10;

}

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

{

ret = ret + (int)(s[i] - 'a') + 10;

}

}

return ret;

}

//명령어에 help 입력시 명령어 list 출력

void menu\_help() {

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

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

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

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

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

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

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

printf("reset\n");

printf("opcode mnemonic\n");

printf("opcodelist\n");

printf("assemble filename");

printf("type filename");

printf("symbol");

printf("progaddr");

printf("loader");

printf("bp");

printf("run");

}

//명령어에 dir입력시 디렉토리 내 파일들 출력

//명령어에 history입력시 현재까지 입력한 명령어 출력

void menu\_history() {

int num = 0;

LinkHistory\* Temp;

for (Temp = Head; Temp != NULL; Temp = Temp->nextHistory) {

num++;

printf("%d\t%s\n", num, Temp->Historyins);

}

}

//명령어에 dump or dump start or dump start, end 입력 시 주소 출력

void menu\_dump() {

int address\_start, address\_end;

int row, column;

int unit;

int value = 0;

char str[10];

if (start == -1 && end == -1) {

address\_start = last\_memory + 1;

if (address\_start >= 16 \* 65536) address\_start = 0;

address\_end = address\_start + 159;

last\_memory = address\_end;

}

else if (end == -1) {

address\_start = start;

address\_end = start + 159;

}

else {

address\_start = start;

address\_end = end;

}

if (address\_end >= 16 \* 65536) address\_end = 16 \* 65536 - 1;

row = address\_start / 16 \* 16, column = (address\_end+16) / 16 \* 16;

for (int i = row; i < column ; i+=16) {

{

printf("%05X ", i);

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

unit = i + j;

if (unit <address\_start || unit > address\_end)

printf(" ");

else

{

value = (int)memory[i/16][j];

printf("%02X ", (unsigned char)value);

}

}

printf("; ");

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

unit = i + j;

if (unit<address\_start || unit>address\_end)

printf(".");

else {

if (0x20 <= (int)memory[i/16][j] && (int)memory[i/16][j] <= 0x7E)

printf("%c", memory[i/16][j]);

else

printf(".");

}

}

printf("\n");

}

}

}

//명령어에 edit address, value입력 시 해당 address 주소 값을 value로 변경

void menu\_edit() {

int row, col;

row = edit\_address / 16;

col = edit\_address % 16;

memory[row][col] = edit\_value;

}

//명령어에 fill start, end, value입력 시 시작 주소부터 끝 주소까지의 주소 값을 value로 변경

void menu\_fill() {

for (int i = fill\_start; i <= fill\_end; i++) {

memory[i / 16][i % 16] = fill\_value;

}

}

//memory값을 0으로 초기화

void menu\_reset() {

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

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

memory[i][j] = 0;

}

//hash table을 만드는 함수

void makehash() {

LinkOpcode\* TempHash;

char line[100], opcode[20], format[20];

int value, index;

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

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

sscanf(line, "%X %s %s", &value, opcode, format);

TempHash = (LinkOpcode\*)malloc(sizeof(LinkOpcode));

TempHash->Value = value;

strcpy(TempHash->Opcode, opcode);

strcpy(TempHash->Format, format);

TempHash->nextOpcode = NULL;

index = hash(opcode);

if (!OpHead[index])

{

OpHead[index] = TempHash;

OpTail[index] = TempHash;

}

else {

OpTail[index]->nextOpcode = TempHash;

OpTail[index] = TempHash;

}

}

}

//hash\_function

int hash(char\* s) {

int len = strlen(s);

int sum = 0;

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

sum += s[i];

}

return sum % 20;

}

//명령어에 입력 받은 값의 해당 opcode를 출력

void menu\_opcode() {

printf("opcode is %X\n", hash\_value);

}

//hash\_Table을 출력

void menu\_opcodelist() {

LinkOpcode\* Temp;

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

printf("%d : ", i);

for (Temp = OpHead[i]; Temp != NULL; Temp = Temp->nextOpcode) {

printf("[%s, %X]", Temp->Opcode, Temp->Value);

if(Temp->nextOpcode)

printf(" -> ");

}

printf("\n");

}

}

//강의시간 pass1에 해당하는 함수

int assemble(FILE\* fileasm) {

int loc\_start = 0, flag = 0, symbolflag = 0, error = 0, endflag = 0, baseflag = 0;

char asminstr1[100], asm1[20], asm2[20], asm3[20], asm4[20], asm5[20], base[10];

int i = 0, lvalue = 0, len = 0;

LinkOpcode\* FindOpcode;

Symbol\* TempSymbol=NULL;

while (fgets(asminstr1, sizeof(asminstr1), fileasm) != NULL) {

if (endflag || error)

break;

lvalue += 5;

location[i] = loc\_start;

line[i] = lvalue;

flag = 0;

symbolflag = 0;

sscanf(asminstr1, "%s %s %s %s", asm1, asm2, asm3, asm4);

if (asm2[strlen(asm2) - 1] == ',') asm2[strlen(asm2) - 1] == NULL;

if (strcmp(asm2, "start") == 0) {

loc\_start = makedecimal(asm3);

locstart = loc\_start;

}

else if (asm1[0] != '.') {

FindOpcode = NULL;

if (strcmp(asm1, "BASE") == 0 || strcmp(asm2, "START") == 0 || strcmp(asm1, "END") == 0) flag = 1;

if (asm1[0] == '+')

FindOpcode = FindOp(asm1 + 1);

else

FindOpcode = FindOp(asm1);

if (!FindOpcode && flag != 1) { //symbol인 경우 symbol linked list에 추가

makeSymbol(asm1, loc\_start);

symbolflag = 1;

}

if (!symbolflag) { //symbol이 아닌경우 symbol인 경우와 자리를 맞춰준다.

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

asm4[i] = asm3[i];

asm3[i] = asm2[i];

asm2[i] = asm1[i];

}

}

FindOpcode = NULL;

flag = 0;

if (strcmp(asm2, "BASE") == 0) flag = 1;

else if (strcmp(asm2, "BYTE") == 0) flag = 2;

else if (strcmp(asm2, "RESW") == 0) flag = 3;

else if (strcmp(asm2, "RESB") == 0) flag = 4;

else if (strcmp(asm2, "END") == 0) flag = 5;

if (asm2[0] == '+')

FindOpcode = FindOp(asm2 + 1);

else

FindOpcode = FindOp(asm2);

if (FindOpcode) { //opcode이면

if (asm2[0] == '+' && strcmp(FindOpcode->Format, "3/4") == 0) // Format 4

{

if (strcmp(FindOpcode->Opcode, "JSUB") == 0)

loc\_start = loc\_start + 4;

else if (strcmp(FindOpcode->Opcode, "LDT") == 0)

loc\_start = loc\_start + 4;

}

else if (asm2[0] != '+' && strcmp(FindOpcode->Format, "3/4") == 0) // Format 3

{

if (strcmp(FindOpcode->Opcode, "RSUB") == 0) loc\_start = loc\_start + 3;

else if (strcmp(FindOpcode->Opcode, "RSUB") != 0)

loc\_start = loc\_start + 3;

}

else if (asm2[0] != '+' && strcmp(FindOpcode->Format, "2") == 0) //Format2

{

loc\_start = loc\_start + 2;

}

else if (asm2[0] != '+' && strcmp(FindOpcode->Format, "1") == 0) //Format1

loc\_start = loc\_start + 1;

else error = 1;

}

else if (!FindOpcode && flag) { //directive

if (flag == 1) { //BASE

strcpy(base, asm3);

baseflag = 1;

}

else if (flag == 2) { //BYTE

len = strlen(asm3);

if (asm3[0] == 'C') loc\_start = loc\_start + (len - 3);

else if (asm3[0] == 'X') loc\_start = loc\_start + (len - 3) / 2;

else error = 1;

}

else if (flag == 3) { //RESW

locationflag = atoi(asm3);

loc\_start = loc\_start + (3 \* locationflag);

}

else if (flag == 4) { //RESB

locationflag = atoi(asm3);

loc\_start = loc\_start + locationflag;

}

else if (flag == 5) { // END

endflag = 1;

}

}

}

i++;

}

if (baseflag)

{

for (TempSymbol = SymHead; TempSymbol != NULL; TempSymbol = TempSymbol->nextSymbol) {

if (strcmp(TempSymbol->sym, base) == 0) {

basevalue = TempSymbol->loc;

break;

}

}

}

loclength = loc\_start - locstart;

if (error) printf("ERROR : %d LINE", lvalue);

return error;

}

//Opcode Linked List에서 해당 코드의 주소를 반환

LinkOpcode\* FindOp(char\* Code) {

LinkOpcode\* LinkTemp = NULL;

int index;

index = hash(Code);

for (LinkTemp = OpHead[index]; LinkTemp != NULL; LinkTemp = LinkTemp->nextOpcode) {

if (strcmp(LinkTemp->Opcode, Code) == 0)

break;

}

return LinkTemp;

}

//Symbol Linked List에서 해당 코드의 주소를 반환

Symbol\* FindSymbol(char\* Code) {

Symbol\* TempSymbol = NULL;

for (TempSymbol = SymHead; TempSymbol != NULL; TempSymbol = TempSymbol->nextSymbol) {

if (strcmp(TempSymbol->sym, Code) == 0)

break;

}

return TempSymbol;

}

//강의에서 pass2에 해당하는 부분

void assemble1(FILE\* fileasm) { //lst와 obj파일을 만든다.

char asm\_instr[100], asm\_instr1[20], asm\_instr2[20], asm\_instr3[20], asm\_instr4[20];

char object[10], obtemp[20], obtemp1[20], objecttemp[200];

int oblen = 0, len = 0, opvalue = 0, obflag = 0;

int endflag = 0, error = 0, flag = 0, xvalue = 0, pcflag = 0, baseflag = 0;

int locationtemp = 0;

int count = 0;

LinkOpcode\* FindOpcode;

Symbol\* TempSymbol;

Modify\* ModifyTemp;

int oblen1 = 0, oblen2 = 0;

FILE\* fp1 = fopen("2\_5.obj", "w");

FILE\* fp2 = fopen("2\_5.lst", "w");

while (fgets(asm\_instr, sizeof(asm\_instr), fileasm) != NULL) {

flag = 0, pcflag = 0, baseflag = 0;

for (int i = 0; i < 10; i++)object[i] = 0;

for (int i = 0; i < 20; i++)obtemp[i] = 0;

locationtemp = location[count] - locstart;

if (endflag || error)

break;

sscanf(asm\_instr, "%s %s %s %s", asm\_instr1, asm\_instr2, asm\_instr3, asm\_instr4);

for (int i = 0; i < 20; i++) if (asm\_instr2[i] == ',') asm\_instr2[i] = NULL;

if (asm\_instr1[0] != '.') {

FindOpcode = NULL;

if (asm\_instr1[0] == '+')

FindOpcode = FindOp(asm\_instr1 + 1);

else

FindOpcode = FindOp(asm\_instr1);

if (strcmp(asm\_instr1, "BASE") == 0 || strcmp(asm\_instr1, "START") == 0 || strcmp(asm\_instr1, "END") == 0) flag = 1;

if (FindOpcode || flag) {

for (int i = 0; i < 20; i++) { //symbol이 아닌경우와 맞춰준다.

asm\_instr4[i] = asm\_instr3[i];

asm\_instr3[i] = asm\_instr2[i];

asm\_instr2[i] = asm\_instr1[i];

}

}

FindOpcode = NULL;

if (asm\_instr2[0] == '+')

FindOpcode = FindOp(asm\_instr2 + 1);

else

FindOpcode = FindOp(asm\_instr2);

flag = 0;

obflag = 0;

if (strcmp(asm\_instr2, "BASE") == 0) flag = 1;

else if (strcmp(asm\_instr2, "BYTE") == 0) flag = 2;

else if (strcmp(asm\_instr2, "RESW") == 0) flag = 3;

else if (strcmp(asm\_instr2, "RESB") == 0) flag = 4;

else if (strcmp(asm\_instr2, "END") == 0) flag = 5;

else if (strcmp(asm\_instr2, "START") == 0)flag = 6;

if (flag) {

obflag = 3;

if (flag == 2) {

len = strlen(asm\_instr3);

if (asm\_instr3[0] == 'X') { //BYTE

oblen = 0;

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

object[oblen++] = asm\_instr3[i];

}

object[oblen] = NULL;

}

else {

oblen = 0;

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

sprintf(obtemp, "%02x", (int)asm\_instr3[i]);

object[oblen] = obtemp[0];

object[oblen + 1] = obtemp[1];

oblen += 2;

}

object[oblen] = 0;

}

}

else if (flag == 3 || flag == 4) { //RESB와 RESW인 경우

obflag = 1;

}

else if (flag == 5) endflag = 1;

else if (flag == 6) //START인 경우

obflag = 2;

}

else if (FindOpcode) { //Opcode이면

obflag = 3;

opvalue = FindOpcode->Value;

if (asm\_instr2[0] != '+' && strcmp(FindOpcode->Format, "1") == 0) //Format1

{

sprintf(obtemp, "%02x", opvalue);

object[0] = obtemp[0];

object[1] = obtemp[1];

object[2] = NULL;

}

else if (asm\_instr2[0] != '+' && strcmp(FindOpcode->Format, "2") == 0) //Format2

{

sprintf(obtemp, "%02x", opvalue);

object[0] = obtemp[0];

object[1] = obtemp[1];

object[2] = regist(asm\_instr3);

object[3] = regist(asm\_instr4);

object[4] = NULL;

}

else {

if (asm\_instr3[0] == '@') opvalue += 2;//indirect addressing(n=1, i=0)

else if (asm\_instr3[0] == '#') opvalue += 1; //immediate addressing;(n=0, i=1)

else opvalue += 3; //simple addressing(n=1, i=1)

sprintf(obtemp, "%02x", opvalue);

object[0] = obtemp[0];

object[1] = obtemp[1];

if (strcmp(asm\_instr2, "RSUB") != 0) {

if (asm\_instr4[0] == 'X')

xvalue = 8; //x=1

else xvalue = 0;

TempSymbol = NULL;

if (asm\_instr3[0] == '#' || asm\_instr3[0] == '@')

TempSymbol = FindSymbol(asm\_instr3 + 1);

else

TempSymbol = FindSymbol(asm\_instr3);

if (!TempSymbol) {

if (asm\_instr3[0] == '#' || asm\_instr3[0] == '@') opvalue = atoi(asm\_instr3 + 1);

else opvalue = atoi(asm\_instr3);

}

else

opvalue = TempSymbol->loc;

if (asm\_instr2[0] == '+') {

xvalue = xvalue + 1; //e=1

sprintf(obtemp, "%x", xvalue);

object[2] = obtemp[0];

sprintf(obtemp, "%05x", opvalue);

for (int i = 0; i < 5; i++)object[i + 3] = obtemp[i];

object[8] = NULL;

if (TempSymbol) //symbol이면 modification linked list에 추가

{

ModifyTemp = (Modify\*)malloc(sizeof(Modify));

ModifyTemp->nextModify = NULL;

ModifyTemp->loc = locationtemp + 1;

if (ModifyHead == NULL) {

ModifyHead = ModifyTemp;

ModifyTail = ModifyTemp;

}

else {

ModifyTail->nextModify = ModifyTemp;

ModifyTail = ModifyTemp;

}

}

}

else if (TempSymbol) {

opvalue = opvalue - location[count+1];

if (opvalue >= -2048 && opvalue <= 2047) // pc

{

pcflag = 1;

opvalue = opvalue & 0xFFF;

sprintf(obtemp, "%03x", opvalue);

xvalue = xvalue + 2; //p = 1

}

else { //base

opvalue = opvalue + location[count+1];

opvalue = opvalue - basevalue;

if (opvalue >= 0 && opvalue <= 4095) {

sprintf(obtemp, "%03x", opvalue);

xvalue = xvalue + 4; //b=1

baseflag = 1;

}

}

if (pcflag) {

for (int i = 0; i < 3; i++) object[i + 3] = obtemp[i];

object[6] = NULL;

}

if (baseflag) {

for (int i = 0; i < 3; i++) object[i + 3] = obtemp[i];

object[6] = NULL;

}

}

else {

sprintf(obtemp, "%03x", opvalue);

for (int i = 0; i < 3; i++) object[i + 3] = obtemp[i];

object[6] = NULL;

}

sprintf(obtemp, "%x", xvalue);

object[2] = obtemp[0];

}

else {

for (int i = 2; i < 6; i++) object[i] = '0';

object[6] = NULL;

}

}

}

}

//lst파일을 만드는 코드

asm\_instr[strlen(asm\_instr) - 1] = NULL;

fprintf(fp2, "\t%d", line[count]);

if (object[0] == NULL && (flag != 2 && flag != 3 && flag != 4)) fprintf(fp2, "\t\t\t");

else {

sprintf(obtemp1, "\t%04x", location[count]);

fprintf(fp2, "\t%s", obtemp1);

}

fprintf(fp2, "\t%-30s", asm\_instr);

if (object[0] != NULL) fprintf(fp2, "\t%s", object);

fprintf(fp2, "\n");

//obj파일을 만드는 코드

if (obflag == 2) {

fprintf(fp1, "HCOPY ", stdin);

fprintf(fp1, "%06x", locstart);

fprintf(fp1, "%06x\n", loclength);

}

else {

oblen1 = strlen(object);

if (!oblen2 && object[0])

{

objecttemp[0] = 'T';

sprintf(obtemp, "%06x", location[count]);

for (int i = 0; i < 6; i++)objecttemp[i + 1] = obtemp[i];

oblen2 = 9;

}

if (obflag == 1) // RESW, RESB 가 입력 될 경우

{

if (oblen2 > 0)

{

objecttemp[oblen2] = 0;

sprintf(obtemp, "%02x", (oblen2 - 9) / 2 );

objecttemp[7] = obtemp[0];

objecttemp[8] = obtemp[1];

fprintf(fp1, "%s\n", objecttemp);

}

oblen2 = 0;

}

if (obflag == 3 && oblen2 + oblen1 < 69 && object[0]) {

for (int i = 0; i < oblen1; i++) objecttemp[oblen2 + i] = object[i];

oblen2 = oblen2 + oblen1;

}

else if (obflag == 3 && oblen2 + oblen1 >= 69 && object[0]) {

objecttemp[oblen2] = 0;

sprintf(obtemp, "%02x", (oblen2 - 9) / 2 );

objecttemp[7] = obtemp[0];

objecttemp[8] = obtemp[1];

fprintf(fp1, "%s\n", objecttemp);

objecttemp[0] = 'T';

oblen2 = 0;

sprintf(obtemp, "%06x", location[count]);

for (int i = 0; i < 6; i++) objecttemp[i + 1] = obtemp[i];

for (int i = 0; i < oblen1; i++) objecttemp[i + 9] = object[i];

oblen2 = 9 + oblen1;

}

}

count++;

}

if(oblen2){

objecttemp[oblen2] = 0;

sprintf(obtemp, "%02x", (oblen2 - 9) / 2 );

objecttemp[7] = obtemp[0];

objecttemp[8] = obtemp[1];

fprintf(fp1, "%s\n", objecttemp);

}

if (ModifyHead != NULL) // Modification Record 출력

{

for (ModifyTemp = ModifyHead; ModifyTemp != NULL; ModifyTemp = ModifyTemp->nextModify)

fprintf(fp1, "M%06x05\n", ModifyTemp->loc);

}

fprintf(fp1, "E%06x\n", locstart);

fclose(fp2);

fclose(fp1);

}

//해당 레지스터 값을 반환

char regist(char\* str) {

char flag = '0';

if (strcmp(str, "A") == 0)flag = '0';

else if (strcmp(str, "X") == 0)flag = '1';

else if (strcmp(str, "L") == 0)flag = '2';

else if (strcmp(str, "PC") == 0)flag = '8';

else if (strcmp(str, "SW") == 0)flag = '9';

else if (strcmp(str, "B") == 0)flag = '3';

else if (strcmp(str, "S") == 0)flag = '4';

else if (strcmp(str, "T") == 0)flag = '5';

else if (strcmp(str, "F") == 0)flag = '6';

return flag;

}

//Symbol Linked List를 만드는 함수

void makeSymbol(char\* asm1, int loc\_start) {

Symbol\* TempSymbol;

TempSymbol = (Symbol\*)malloc(sizeof(Symbol));

TempSymbol->nextSymbol = NULL;

strcpy(TempSymbol->sym, asm1);

TempSymbol->loc = loc\_start;

if (!SymHead) {

SymHead = TempSymbol;

SymTail = TempSymbol;

}

else {

SymTail->nextSymbol = TempSymbol;

SymTail = TempSymbol;

}

}

//Symbol을 내림차순으로 출력하는 함수

void menu\_symbol() {

char SymTemp[5];

Symbol\* SymbolHead, \* SymbolTail, \* SymbolTemp1, \* SymbolTemp2, \* SymbolTemp3, \* SymbolTemp4;

SymbolHead = NULL, SymbolTail = NULL;

for (SymbolTemp1 = SymHead; SymbolTemp1 != NULL; SymbolTemp1 = SymbolTemp1->nextSymbol) {

SymbolTemp2 = (Symbol\*)malloc(sizeof(Symbol));

SymbolTemp2->nextSymbol = NULL;

strcpy(SymbolTemp2->sym, SymbolTemp1->sym);

SymbolTemp2->loc = SymbolTemp1->loc;

if (!SymbolHead) {

SymbolHead = SymbolTemp2;

SymbolTail = SymbolTemp2;

}

else {

if (strcmp(SymbolTail->sym, SymbolTemp2->sym) < 0) {

SymbolTail->nextSymbol = SymbolTemp2;

SymbolTail = SymbolTemp2;

}

else {

SymbolTemp4 = NULL;

for (SymbolTemp3 = SymbolHead; SymbolTemp3 != NULL; SymbolTemp3 = SymbolTemp3->nextSymbol) {

if (strcmp(SymbolTemp3->sym, SymbolTemp2) > 0)break;

SymbolTemp4 = SymbolTemp3;

}

if (SymbolTemp4 == NULL) {

SymbolTemp4 = SymbolTemp2;

SymbolTemp4->nextSymbol = SymbolTemp3;

SymbolHead = SymbolTemp4;

}

else {

SymbolTemp4->nextSymbol = SymbolTemp2;

SymbolTemp2->nextSymbol = SymbolTemp3;

}

}

}

}

for (int i = 0; i < 5; i++) SymTemp[i] = 0;

for (SymbolTemp1 = SymbolHead; SymbolTemp1 != NULL; SymbolTemp1 = SymbolTemp1->nextSymbol) {

sprintf(SymTemp, "%04x", SymbolTemp1->loc);

printf("\t%s\t%s\n", SymbolTemp1->sym, SymTemp);

}

}

void makeEstab(Estab\* TempEstab) {

int index;

index = hash(TempEstab->esymbol);

if (!EsHead[index]) {

EsHead[index] = TempEstab;

EsTail[index] = TempEstab;

}

else {

EsTail[index]->nextEstab = TempEstab;

EsTail[index] = TempEstab;

}

}

void link(FILE\* link\_file) {

FILE\* linkfile = fopen(link\_file, "r");

int CSLTH = 0;

int link\_len = 0, link\_loc = 0, link\_index;

char link\_instr[100], location[20];

Estab\* TempEstab, \*TempEstab1;

for (int i = 0; i < 100; i++) link\_instr[i] = 0;

for (int i = 0; i < 20; i++) location[i] = 0;

while (fgets(link\_instr, sizeof(link\_instr), linkfile) != NULL) {

link\_len = strlen(link\_instr);

link\_instr[link\_len] = NULL;

if (link\_instr[link\_len - 1] == '\n') link\_instr[link\_len - 1] = NULL;

if (link\_instr[0] == 'H') {

TempEstab = (Estab\*)malloc(sizeof(Estab));

TempEstab->flag = 0;

TempEstab->nextEstab = NULL;

for(int i=0;i<6;i++) TempEstab->esymbol[i] = link\_instr[i + 1];

TempEstab->esymbol[6] = NULL;

for (int i = 0; i < 6; i++) location[i] = link\_instr[i + 7];

link\_loc = strtol(location, NULL, 16);

for (int i = 0; i < 6; i++) location[i] = link\_instr[i + 13];

link\_len = strtol(location, NULL, 16);

TempEstab->loc = link\_loc + progaddr;

TempEstab->len = link\_len;

CSLTH = link\_len;

link\_index = hash(TempEstab->esymbol);

makeEstab(TempEstab);

}

else if (link\_instr[0] == 'D') {

TempEstab = (Estab\*)malloc(sizeof(Estab));

TempEstab->flag = 1;

TempEstab->nextEstab = NULL;

for (int i = 0; i < 6; i++) TempEstab->esymbol[i] = link\_instr[i + 1];

TempEstab->esymbol[6] = NULL;

for (int i = 0; i < 6; i++) location[i] = link\_instr[i + 7];

link\_loc = strtol(location, NULL, 16);

TempEstab->loc = link\_loc + progaddr;

makeEstab(TempEstab);

TempEstab1 = (Estab\*)malloc(sizeof(Estab));

TempEstab1->flag = 1;

TempEstab1->nextEstab = NULL;

for (int i = 0; i < 6; i++) TempEstab1->esymbol[i] = link\_instr[i + 13];

TempEstab1->esymbol[6] = NULL;

for (int i = 0; i < 6; i++) location[i] = link\_instr[i + 19];

link\_loc = strtol(location, NULL, 16);

TempEstab1->loc = link\_loc + progaddr;

makeEstab(TempEstab1);

}

else if (link\_instr[0] == 'E') {

progaddr = progaddr + CSLTH;

}

}

fclose(linkfile);

}

void loader(FILE\* fp) {

int len = 0, CSLTH = 0, index = 0, load\_loc = 0, dif = 0, value = 0, flag = 0, reference[6], cnt;

char load\_instr[100], location[10], symbol[10];

Estab\* TempEstab;

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

reference[1] = progaddr;

while (fgets(load\_instr, sizeof(load\_instr), loaderfile) != NULL) {

len = strlen(load\_instr);

load\_instr[len] = NULL;

if (load\_instr[len - 1] == '\n') load\_instr[len - 1] = NULL;

if (load\_instr[0] == 'H') {

for (int i = 0; i < 6; i++) location[i] = load\_instr[i + 13];

location[6] = NULL;

CSLTH = strtol(location, NULL, 16);

}

else if (load\_instr[0] == 'R') {

len = 1;

for (int i = 0; i < 4; i++) {

for (int j = len; j <= len + 1; j++) location[j - len] = load\_instr[j];

location[2] = NULL;

for (int j = 0; j < 6; j++) symbol[j] = ' ';

for (int k = len + 2; k <= len + 7; k++) {

if (load\_instr[k] == NULL) break;

symbol[k - len - 2] = load\_instr[k];

}

symbol[6] = NULL;

index = hash(symbol);

TempEstab = EsHead[index];

for (TempEstab; TempEstab != NULL; TempEstab = TempEstab->nextEstab) {

if (strcmp(TempEstab->esymbol, symbol) == 0) break;

}

index = strtol(location, NULL, 10);

reference[index] = TempEstab->loc;

len = len + 8;

}

}

else if (load\_instr[0] == 'T') {

for (int i = 0; i < 6; i++) location[i] = load\_instr[i + 1];

location[6] = NULL;

load\_loc = strtol(location, NULL, 16);

dif = load\_loc + progaddr;

location[0] = load\_instr[7];

location[1] = load\_instr[8];

location[2] = NULL;

len = strtol(location, NULL, 16);

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

location[0] = load\_instr[9 + (2 \* i)];

location[1] = load\_instr[10 + (2 \* i)];

location[2] = NULL;

value = strtol(location, NULL, 16);

memory[dif / 16][dif % 16] = value;

dif++;

}

}

else if (load\_instr[0] == 'M') {

for (int i = 0; i < 6; i++) location[i] = load\_instr[i + 1];

location[6] = NULL;

load\_loc = hextodec(location);

dif = load\_loc + progaddr;

location[0] = load\_instr[7];

location[1] = load\_instr[8];

location[2] = NULL;

cnt = hextodec(location);

for (int i = 10; i <= 11; i++) location[i - 10] = load\_instr[i];

location[2] = NULL;

index = hextodec(location);

int modAddress = memory[dif / 16][dif % 16] % (cnt % 2 ? 0x10 : 0x100);

for (int j = 1; j <= (cnt - 1) / 2; j++) {

modAddress \*= 0x100;

modAddress += memory[(dif + j) / 16][(dif + j) % 16];

}

modAddress += (load\_instr[9] == '+' ? 1 : -1) \* reference[index];

for (int j = (cnt - 1) / 2; j; j--) {

memory[(dif + j) / 16][(dif + j) % 16] = modAddress % 0x100;

modAddress /= 0x100;

}

memory[dif / 16][dif % 16] = (cnt % 2 ? (memory[dif / 16][dif % 16] / 0x10) \* 0x10 + modAddress : modAddress);

}

else if (load\_instr[0] == 'E') {

progaddr\_end = progaddr + CSLTH;

progaddr = progaddr + CSLTH;

break;

}

}

}

void printload() {

Estab\* EstabHead, \*EstabTail,\* EstabTemp1, \* EstabTemp2, \* EstabTemp3, \* EstabTemp4;

EstabHead = NULL, EstabTail = NULL;

char str[10];

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

EstabTemp1 = EsHead[i];

if (EstabTemp1) {

for (;;) {

if (!EstabTemp1) break;

EstabTemp2 = (Estab\*)malloc(sizeof(Estab));

strcpy(EstabTemp2->esymbol, EstabTemp1->esymbol);

EstabTemp2->flag = EstabTemp1->flag;

EstabTemp2->loc = EstabTemp1->loc;

EstabTemp2->nextEstab = NULL;

if (EstabTemp2->flag == 0) EstabTemp2->len = EstabTemp1->len;

if (EstabHead == NULL) {

EstabHead = EstabTemp2;

EstabTail = EstabTemp2;

}

else {

if (EstabTail->loc < EstabTemp2->loc) {

EstabTail->nextEstab = EstabTemp2;

EstabTail = EstabTemp2;

}

else {

EstabTemp4 = NULL;

EstabTemp3 = EstabHead;

for (;;) {

if (EstabTemp3 == NULL) break;

if (EstabTemp3 ->loc > EstabTemp2->loc) break;

EstabTemp4 = EstabTemp3;

EstabTemp3 = EstabTemp3->nextEstab;

}

if (EstabTemp4 == NULL) {

EstabTemp4 = EstabTemp2;

EstabTemp4->nextEstab = EstabTemp3;

EstabHead = EstabTemp4;

}

else {

EstabTemp4->nextEstab = EstabTemp2;

EstabTemp2->nextEstab = EstabTemp3;

}

}

}

EstabTemp1 = EstabTemp1->nextEstab;

}

}

}

int length = 0;

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

printf("section\tname\n");

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

for (EstabTemp1 = EstabHead; EstabTemp1 != NULL; EstabTemp1 = EstabTemp1->nextEstab) {

if (EstabTemp1->flag == 0) {

printf("%s\t\t%04X", EstabTemp1->esymbol, EstabTemp1->loc);

printf("\t%04X\n", EstabTemp1->len);

length = length + EstabTemp1->len;

}

else {

printf("\t%s", EstabTemp1->esymbol);

printf("\t%04X\n", EstabTemp1->loc);

}

}

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

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

}

void transstring(int k, char\* s, int len)

{

int i, hex = 1;

for (i = 0; i < len - 1; i++) hex = hex \* 16;

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

{

if (k / hex >= 0 && k / hex <= 9) s[i] = (char)((k / hex) + '0');

else if (k / hex >= 10 && k / hex <= 15) s[i] = (char)(((k / hex) - 10) + 'A');

k = k % hex;

hex = hex / 16;

}

s[len] = NULL;

}

void printbp() {

BP\* TempBP = NULL;

char str[10];

TempBP = BPHead;

if (BPHead == NULL) {

printf("\tBreakpoint does not exist\n");

}

else {

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

for (; TempBP != NULL; TempBP = TempBP->nextBP) {

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

}

}

}

void clearbp() {

BP\* TempBP = NULL;

if (BPHead == NULL) {

printf("\tBreakpoint does not exist\n");

}

else {

printf("\t[ok] clear all breakpoints\n");

TempBP = BPHead;

for (;;)

{

if (TempBP == NULL)break;

TempBP = BPHead->nextBP;

free(BPHead);

BPHead = TempBP;

}

}

}

int getTarAdd(int cur\_addr, int format) {

int addr = memory[cur\_addr / 16][cur\_addr % 16] & 3;

int target;

switch (addr) {

case 1: //n=0, i=0

target = SICAddress(cur\_addr);

break;

case 2: //n=0, i=1

target = immediateAddress(cur\_addr, format);

break;

case 3://n=1, i=0

target = indirectAddress(cur\_addr, format);

break;

case 4://n=1, i=1

target = simpleAddress(cur\_addr, format);

break;

default:

break;

}

if (memory[(cur\_addr + 1) / 16][(cur\_addr + 1) % 16] & 0x80) //x=1

target += registers[1];

return target;

}

int SICAddress(int add) {

return getMemo(add, 5) & 0x7FFF; // lower 15byte

}

int immediateAddress(int add, int format) {

return simpleAddress(add, format);

}

int indirectAddress(int add, int format) {

int target = simpleAddress(add, format);

return getMemo(target, 6);

}

int simpleAddress(int add, int format) {

int sett = (memory[(add+1) / 16][(add+1) % 16] / 0x10) & 6; // b, p

int target = getMemo(add + 1, format == 3 ? 3 : 5);

if (sett == 2) { // PC

if (target & (format == 3 ? 0x800 : 0x80000)) // displacement

target = target | (format == 3 ? 0xFFFFF000 : 0xFFF00000);

target += registers[8];

}

else if (sett == 4) // Base

target += registers[3];

return target;

}

int getMemo(int addr, int h) {

int val = 0;

int temp = 0;

if (memory[addr / 16][addr % 16] < 0) temp = 256 + memory[addr / 16][addr % 16];

else temp = memory[addr / 16][addr % 16];

val = (temp) % (h % 2 ? 0x10 : 0x100); // half or full

for (int i = 1; i <= (h - 1) / 2; i++) {

val \*= 0x100; //

val += memory[(addr + i) / 16][(addr + i) % 16];

}

return val;

}

void SetMemo(int add, int b, int val) {

int i;

for (i = add + b - 1; i >= add; i--) {

memory[i / 16][i % 16] = val & 0xFF;

val /= 0x100;

}

}

void DumpReg() {

printf("\t A : %06X X : %06X\n", registers[0], registers[1]);

printf("\t L : %06X PC: %06X\n", registers[2], registers[8]);

printf("\t B : %06X S : %06X\n", registers[3], registers[4]);

printf("\t T : %06X", registers[5]);

}

void runobj() {

int tar\_addr, tar\_val;

int cur\_addr = -1;

int last\_BP = -1;

obj curOBJ;

CCstatus = 4;

if (cur\_addr == -1)

cur\_addr = progaddr\_load;

registers[8] = cur\_addr; //PC REGISTER

for (; cur\_addr < progaddr\_end;) {

curOBJ.mnemonic = memory[cur\_addr / 16][cur\_addr % 16] & 0xFC; //Decode opcode

switch (memory[cur\_addr / 16][cur\_addr % 16] / 0x10) //instruction format

{

case 0:

case 1:

case 2:

case 3:

case 4:

case 5:

case 6:

case 7:

case 8:

case 0xD:

case 0xE:

curOBJ.format = 3;

if (memory[(cur\_addr + 1) / 16][(cur\_addr + 1) % 16] & 0x10) //e = 1;

curOBJ.format = 4;

break;

case 9:

case 0xA:

case 0xB:

curOBJ.mnemonic += memory[cur\_addr / 16][cur\_addr % 16] & 0x3;

curOBJ.format = 2;

case 0xC:

case 0xF:

curOBJ.mnemonic += memory[cur\_addr / 16][cur\_addr % 16] & 0x3;

curOBJ.format = 1 ;

break;

default :

break;

}

if (!memory[cur\_addr / 16][cur\_addr % 16])

curOBJ.format = 1;

for (int i = cur\_addr; i < cur\_addr + curOBJ.format; i++)

if (i > last\_BP && bpfind(i)) {

last\_BP = i;

DumpReg();

printf("\n\tStop at checkpoint[%04X]\n", i);

return;

}

if (!memory[cur\_addr / 16][cur\_addr % 16]) {

registers[8] += 1;

cur\_addr = registers[8];

continue;

}

registers[8] += curOBJ.format; //increase PC

curOBJ.address = 4; //simple

switch (curOBJ.format) {

case 3:

case 4:

curOBJ.operand.tar = getTarAdd(cur\_addr, curOBJ.format);

curOBJ.address = memory[cur\_addr / 16][cur\_addr % 16] & 3;

if (memory[cur\_addr / 16][cur\_addr % 16] & 0x80) //x = 1

curOBJ.operand.tar += registers[1];

if (curOBJ.address == 2) //immediate

curOBJ.operand.immediate = curOBJ.operand.tar;

break;

case 2:

curOBJ.operand.reg[0] = memory[(cur\_addr + 1) / 16][(cur\_addr + 1) % 16] / 0x10; //reg1

curOBJ.operand.reg[1] = memory[(cur\_addr + 1) / 16][(cur\_addr + 1) % 16] % 0x10; //reg2

break;

case 1:

break;

default:

break;

}

tar\_addr = curOBJ.operand.tar;

tar\_val = (curOBJ.address == 2 ? curOBJ.operand.immediate : getMemo(curOBJ.operand.tar, 6));

//execute

switch (curOBJ.mnemonic) {

case ADD:

registers[0] += tar\_val;

break;

case ADDF:

registers[6] += tar\_val;

break;

case ADDR:

registers[curOBJ.operand.reg[1]] += registers[curOBJ.operand.reg[0]];

break;

case AND:

registers[0] &= tar\_val;

break;

case CLEAR:

registers[curOBJ.operand.reg[0]] = 0;

break;

case COMP:

if (registers[6] < tar\_val)

CCstatus = lt;

else if (registers[6] == tar\_val)

CCstatus = eq;

else

CCstatus = gt;

break;

case COMPR:

if (registers[curOBJ.operand.reg[0]] < registers[curOBJ.operand.reg[1]])

CCstatus = lt;

else if (registers[curOBJ.operand.reg[0]] == registers[curOBJ.operand.reg[1]])

CCstatus = eq;

else

CCstatus = gt;

break;

case DIV:

registers[6] /= tar\_val;

break;

case DIVR:

registers[curOBJ.operand.reg[1]] /= registers[curOBJ.operand.reg[0]];

break;

case FIX:

registers[0] = registers[6];

break;

case FLOAT:

registers[6] = registers[0];

break;

case HIO:

break;

case J:

registers[8] = tar\_addr;

break;

case JEQ:

if (CCstatus == eq)

registers[8] = tar\_addr;

break;

case JGT:

if (CCstatus == gt)

registers[8] = tar\_addr;

break;

case JLT:

if (CCstatus == lt)

registers[8] = tar\_addr;

break;

case JSUB:

registers[2] = registers[8];

registers[8] = tar\_addr;

break;

case LDA:

registers[0] = tar\_val;

break;

case LDB:

registers[3] = tar\_val;

break;

case LDCH:

registers[0] = (registers[0] & 0xFFFFFF00) + (tar\_val / 0x10000);

break;

case LDF:

registers[6] = tar\_val;

break;

case LDL:

registers[2] = tar\_val;

break;

case LDS:

registers[4] = tar\_val;

break;

case LDT:

registers[5] = tar\_val;

break;

case LDX:

registers[1] = tar\_val;

break;

case LPS:

break;

case MUL:

registers[0] \*= tar\_val;

break;

case MULF:

registers[6] \*= tar\_val;

break;

case MULR:

registers[curOBJ.operand.reg[1]] \*= registers[curOBJ.operand.reg[0]];

break;

case NORM:

break;

case OR:

registers[0] |= tar\_val;

break;

case RD:

registers[0] = (registers[0] & 0xFFFFFF00) + inputs[in\_idx++];

break;

case RMO:

registers[curOBJ.operand.reg[1]] = registers[curOBJ.operand.reg[0]];

break;

case RSUB:

registers[8] = registers[2];

break;

case SHIFTL:

registers[curOBJ.operand.reg[0]] = registers[curOBJ.operand.reg[0]] << registers[curOBJ.operand.reg[1]];

break;

case SIO:

break;

case SSK:

break;

case STA:

SetMemo(tar\_addr, 3, registers[0]);

break;

case STB:

SetMemo(tar\_addr, 3, registers[3]);

break;

case STCH:

memory[tar\_addr/16][tar\_addr%16] = registers[0] & 0xFF;

break;

case STF:

SetMemo(tar\_addr, 6, registers[6]);

break;

case STI:

break;

case STL:

SetMemo(tar\_addr, 3, registers[2]);

break;

case STS:

SetMemo(tar\_addr, 3, registers[4]);

break;

case STSW:

SetMemo(tar\_addr, 3, registers[9]);

break;

case STT:

SetMemo(tar\_addr, 3, registers[5]);

break;

case STX:

SetMemo(tar\_addr, 3, registers[1]);

break;

case SUB:

registers[0] -= tar\_val;

break;

case SUBF:

registers[6] -= tar\_val;

break;

case SUBR:

registers[curOBJ.operand.reg[1]] -= registers[curOBJ.operand.reg[0]];

break;

case SVC:

break;

case TD:

CCstatus = lt;

break;

case TIO:

CCstatus = lt;

break;

case TIX:

registers[1]++;

if (registers[1] < tar\_val)

CCstatus = lt;

else if (registers[1] == tar\_val)

CCstatus = eq;

else

CCstatus = gt;

break;

case TIXR:

registers[1]++;

if (registers[1] < registers[curOBJ.operand.reg[0]])

CCstatus = lt;

else if (registers[1] == registers[curOBJ.operand.reg[0]])

CCstatus = eq;

else

CCstatus = gt;

break;

case WD:

outputs[out\_idx++] = registers[0] & 0xFF;

break;

default: // not an opcode

registers[8] = registers[8] - curOBJ.format + 1; // increase PC

cur\_addr = registers[8];

continue;

break;

}

cur\_addr = registers[8]; // increase curAddress

}

registers[8] = progaddr\_end;

DumpReg();

printf("\n\tEnd program.\n");

for (int i = 0; i < 10; i++) registers[i] = 0;

registers[2] = progaddr\_end;

cur\_addr = -1;

last\_BP = -1;

CCstatus = 4;

in\_idx = out\_idx = 0;

for (int i = 0; i < 13; i++) outputs[i] = NULL;

}

int bpfind(int i) {

BP\* BPtemp = NULL;

for (BPtemp = BPHead; BPtemp != NULL; BPtemp = BPtemp->nextBP) {

if (BPtemp->address == i)

return 1;

}

return 0;

}

**20141218.h**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/stat.h>

typedef enum {

ADD = 0x18,

ADDF = 0x58,

ADDR = 0x90,

AND = 0x40,

CLEAR = 0xB4,

COMP = 0x28,

COMPF = 0x88,

COMPR = 0xA0,

DIV = 0x24,

DIVF = 0x64,

DIVR = 0x9C,

FIX = 0xC4,

FLOAT = 0xC0,

HIO = 0xF4,

J = 0x3C,

JEQ = 0x30,

JGT = 0x34,

JLT = 0x38,

JSUB = 0x48,

LDA = 0x00,

LDB = 0x68,

LDCH = 0x50,

LDF = 0x70,

LDL = 0x08,

LDS = 0x6C,

LDT = 0x74,

LDX = 0x04,

LPS = 0xD0,

MUL = 0x20,

MULF = 0x60,

MULR = 0x98,

NORM = 0xC8,

OR = 0x44,

RD = 0xD8,

RMO = 0xAC,

RSUB = 0x4C,

SHIFTL = 0xA4,

SIO = 0xF0,

SSK = 0xEC,

STA = 0x0C,

STB = 0x78,

STCH = 0x54,

STF = 0x80,

STI = 0xD4,

STL = 0x14,

STS = 0x7C,

STSW = 0xE8,

STT = 0x84,

STX = 0x10,

SUB = 0x1C,

SUBF = 0x5C,

SUBR = 0x94,

SVC = 0xB0,

TD = 0xE0,

TIO = 0xF8,

TIX = 0x2C,

TIXR = 0xB8,

WD = 0xDC

}Mnemonic;

enum {

lt, eq, gt

} CCstatus;

typedef struct {

Mnemonic mnemonic;

int format;

int address;

union {

int tar;

int immediate;

int reg[2];

}operand;

int index;

} obj;

typedef struct LinkHistory {

char Historyins[100];

struct LinkHistory\* nextHistory;

}LinkHistory;

typedef struct LinkOpcode {

char Opcode[20], Format[20];

int Value;

struct LinkOpcode\* nextOpcode;

}LinkOpcode;

typedef struct Symbol {

char sym[20];

int loc;

struct Symbol\* nextSymbol;

}Symbol;

typedef struct Modify {

int loc;

struct Modify\* nextModify;

}Modify;

typedef struct Estab {

char esymbol[20];

int loc;

int len;

int flag;

struct Estab\* nextEstab;

}Estab;

typedef struct BP {

int address;

struct BP\* nextBP;

}BP;

int makedecimal(char\* address);

void menu();

void menu\_dir();

void menu\_help();

void menu\_history();

void menu\_dump();

void menu\_edit();

void menu\_fill();

void menu\_reset();

void menu\_opcode();

void makehash();

void menu\_symbol();

int hash(char \*s);

void menu\_opcodelist();

int assemble(FILE\* fp);

void assemble1(FILE\* fp);

char regist(char\* re);

LinkOpcode\* FindOp(char\* Code);

Symbol\* FindSymbol(char\* Code);

void makeSymbol(char\* asm1, int loc\_start);

void link(FILE\* fp);

void makeEstab(Estab\* TempEstab);

void loader(FILE\* fp);

void printload();

void transstring(int i, char\* s, int len);

void printbp();

void clearbp();

int hextodec(char\* s);

void runobj();

int getTarAdd(int cur\_addr, int format);

int getMemo(int add, int h);

void SetMemo(int add, int b, int val);

void DumpRegs();

int immediateAddress(int add, int format);

int simpleAddress(int add, int format);

int SICAddress(int add);

int indirectAddress(int add, int format);

int bpfind(int i);