과목 명: 시스템프로그래밍

담당 교수 명: 박 운 상

<<Assignment 3>>

**서강대학교 컴퓨터학과**

**[학번] 20121625**

**[이름] 이정명**

목 차

2. 프로그램 설명

2.1 프로그램 흐름도

3. 모듈 정의

3.1 모듈 이름 : main()

3.1.0 함수 원형

3.1.1 기능

3.1.2 사용 변수

3.2 모듈 이름:isValidLine

3.2.0 함수 원형

3.2.1 기능

3.2.2 사용 변수

3.3 모듈이름: readFrag

3.3.0 함수 원형

3.3.1 기능

3.3.2 사용변수

3.4 모듈이름:isDuplicate

3.4.0 함수 원형

3.4.1 기능

3.4.2 사용변수

3.5 모듈이름: relocation

3.5.0 함수 원형

3.5.1 기능

3.5.2 사용변수

3.6 모듈이름: loader

3.6.0 함수 원형

3.6.1 기능

3.6.2 사용변수

3.7 모듈이름: operate

3.7.0 함수 원형

3.7.1 기능

3.7.2 사용변수

3.8 모듈이름: run

3.8.0 함수 원형

3.8.1 기능

3.8.2 사용변수

3.9 모듈이름: breakpoint

3.9.0 함수 원형

3.9.1 기능

3.9.2 사용변수

4. 구조체 및 전역 변수 정의

4.1 SYMB\_NODE \*\*symb\_head, \*\*symb\_tail;

4.2 ADR\_NODE \*adr\_head, \*adr\_tail;

4.3 M\_NODE \*m\_head,\*m\_tail; HIS\_NODE/ his\_head, his\_tail

5. 코드

# 프로그램 개요

프로젝트 #1, #2 에서 구현한 셀(shell)에 linking과 loading 기능을 추가하는 프로그램입니다.

프로젝트 #2 에서 구현된 assemble 명령을 통해서 생성된 object 파일을 link시켜 메모리에

올리는 일을 수행한다.

loader는 relocation loader를 구현하고 PROGADDER는 사용자로부터 입력받는다.

메모리에 올라간 프로그램을 수행할 수 있도록 한다.

# 프로그램 설명

## 프로그램 흐름도

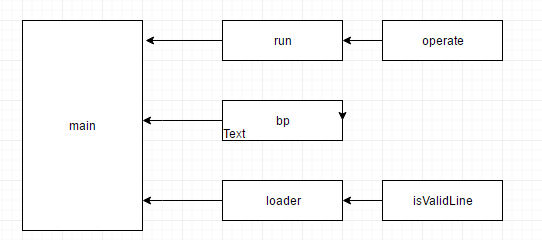


그림 > 프로그램 흐름도

# 모듈 정의

## 모듈 이름 : main()

## *3.1.0. 함수원형* int main(void)

## *3.1.1. 기능*

사용자로부터 PROGADDR를 입력받아 메모리에 로딩되는 시작 주소를 결정한다.

PROGADDR 의 default 값은 0이다.

PROGADDR가 결정되면 loader()을 호출해 assemble 된 obj파일을 메모리에 link, load한다.

link load가 완료되면 bp()를 호출해 breakpoint를 결정한다.

run을 수행시 breakpoint 지점까지 수행한다.

## *3.1.2. 사용변수*

char\*sh\_inst[24] = {"q", "quit","h", ...} : const values

char\*input, buf[100],full\_inst[100] : tokenizing pointer / input buffer / full instruction buffer

char \*inst\_tok[3], \*comma\_pos[3] : storage for string token / storage for comma's address

int j,j : indexer

int tok\_cnt, comma\_cnt, inst\_flag : token counter / comma counter

HIS\_NODE \*curr : ADT pointer for history linked list

FILE \*fp\_asm,\*fp\_lst, \*fp\_obj : filepointer

## 모듈 이름: isValisLine(char\* str)

## *3.2.0. 함수원형*

## int isValisLine(char\* objLine,char REC) {

## *3.2.1. 기능*

## assemble 된 objfile의 format이 올바른지 check한다. 올바른 형식일 경우 1, 그렇지

## 않을 경우 0을 반환한다

## *3.2.2. 사용변수/parameters*

int lineLen:variable for length of objline(should satisfy record format);

char flag=0 : flag variable for appropriate record(H, D, R, T, M, E)

## 모듈 이름: readFrag(int fSize, char\*\* f)

## *3.3.0. 함수원형*

## char\* readFrag(int fSize, char\*\* f)

## *3.3.1. 기능*

string으로부터 fragment lenth 만큼 tokenize 후 tokenize 된 fragment string을 반환한다.

## *3.3.2. 사용변수/parameters*

char\* frag : string pointer for allocating memory & return

int i : indexing variable

## 모듈 이름: isDuplicate(char\* str,int idx,char REC)

## *3.4.0. 함수원형*

int isDuplicate(char\* str,int idx,char REC)

## *3.4.1. 기능*

ESTAB을 탐색해 이미 저장된 progname, 혹은 symbol name 인지 결정한다. 중복됬을 경우 1을

반환한다.

## *3.4.2. 사용변수*

ESTAB\_NODE \*curr　: ESTAB\_NODE pointer for searching ESTAB.

## 모듈이름: relocation(char \*REFLINE, int file\_num,char \*line)

## *3.5.0. 함수원형*

## int relocation(char \*REFLINE, int file\_num,char \*line)

## *3.5.1. 기능*

reference number를 확인해 메모리에 올라간 object code를 수정한다.

R record에 해당하는 reference number가 없거나 ESTAB에 없을 경우 0을 반환한다.

## *3.5.2. 사용변수/parameter*

int i , j : indexer

int sflag : flag for search

int pow ; temporary variable for calculating

int addr : address to be modified

int sign : -1 or 1 mapped to '-' , '+'

int REFLEN : length of R record

int halfB : size to be modified in half byte

int temp = 0, vtemp = 0 : temporary value

char \*reloc : pointer for reference number string

char \*symb, : pointer for symbol name in ESTAB

char \*v\_p : pointer pointing virtual memory

## 모듈이름: loader(int file\_num,char\*\* obj\_file)

## *3.6.0. 함수원형*

## int loader(int file\_num,char\*\* obj\_file)

## *3.6.1. 기능*

사용자로부터 object file name을 받아 메모리에 load 한다.

## *3.6.2. 사용변수/parameter*

## FILE\* fp[3] : file pointer for reading objfile

int i, j : indexer

int T\_ADDR, T\_LEN : T record address / T record length

int total\_len=0 : total length of the program

int REL\_ADDR : temporary variable for relocation

char buff[100] : buffer

char REC\_TYPE : record type

char \*linep : pointer forwarding record

char line\_len : length of the record line

char \*temp : temporary pointer

char REFLINE[72] : array storing R record

ESTAB\_NODE \*temp\_est : ESTAB pointer for searching ESTAB

## 

## 모듈이름: operate(int fSize,char\*frag,char \*op,int\* XBPE,int NI)

## *3.7.0. 함수원형*

## int operate(int fSize,char\*frag,char \*op,int\* XBPE,int NI)

## *3.7.1. 기능*

opcode.txt에 저장된 mnemonic 에 해당하는 기능을 수행한다.

## *3.7.2. 사용변수/parameter*

## int i : indexer

## int temp : temporary variable

## int xor : variable for xor

char r1,r2 : register number

char rt1[2],rt2[2] : register number string

int MEM\_REF : displacement

int TA,TA\_VAL=0 : target address / target address value

## 

## 모듈이름:

## createMnTable(INST\_NODE\* newNode, int idx)

* 1. **createEstabNode(char\* objLine,char REC)**
  2. **createEstab(ESTAB\_NODE\* newNode,int idx)**
  3. **createBpNode(int adr)**

## *3.8.0. 함수원형*

## void createMnTable(INST\_NODE\* newNode, int idx)

* 1. ESTAB\_NODE\* createEstabNode(char\* objLine,char REC)
  2. void createEstab(ESTAB\_NODE\* newNode,int idx)
  3. BP\_NODE\* createBpNode(int adr)

## *3.8.1. 기능*

## Mnmonice table(hash)을 생성한다.

## ESTAB 노드를 생성한다.

## ESTAB(hash)을 생성한다

## break point를 저장하는 linked list 의 노드를 생성한다.

## *3.8.2. 사용변수/parameter*

## 1. 사용 변수 없음

## 2. char\* symb, \*ctrlsect : symbol name pointer / control section pointer

* 1. int adr, len : address / length
  2. ESTAB\_NODE\* newNode : returning ESTAB node

## 3. 사용변수 없음

## 4. 사용변수 없음

## 모듈이름: run()

## *3.9.0. 함수원형*

## int run()

## *3.9.1. 기능*

## 메모리에 로드된 obect code 를 해석해 operate을 호출한다..

## *3.9.2. 사용변수/parameter*

int endBp ; address to be stopped

char op[3], ta[7] : array for storing opcode / array for storing displacement

INST\_NODE \* mn\_curr : INST\_NODE pointer to search mnemonic table

int I : indexer

int,temp temporary variable

int NI,XBPE[4] : NI / XBPE

int fSize,sum;

## static int first\_run = 0;

## 모듈이름: breakpoint(char\*para)

## *3.9.0. 함수원형*

## int breakpoint(char \*para)

## *3.9.1. 기능*

## 사용자에게 입력받은 bp들을 list 형태로 저장한다.

## *3.9.2. 사용변수/parameter*

BP\_NODE\* bp\_curr,\*bp\_prev,\*bp\_temp;

int bp\_adr,flag=0;

# 구조체 전역 변수 정의

# typedef struct \_ESTAB\_NODE {

# char symb[7];

# char ctrlsect[7];

# int adr;

# int len;

# struct \_ESTAB\_NODE \*next;

# }ESTAB\_NODE;

# typedef struct \_BP\_NODE {

# int adr;

# struct \_BP\_NODE \*next;

# }BP\_NODE;

ESTAB\_NODE \*\*est\_head, \*\*est\_tail;

# BP\_NODE \*bp\_head, \*bp\_tail,\*bp\_start,\*bp\_end;

int PROGADDR, CSADDR, CSLTH, CSNUM,PROGLEN;

# int REG[10];

# 코드

**5.1 linkload.c**

#include "20121625.h"

int isValisLine(char\* objLine,char REC) {

int lineLen=strlen(objLine);

char flag=0;

int i;

switch (REC) {

case 'H': if (lineLen != 18) return 0;

flag = 1; break;

case 'D': if ((lineLen) % 12!=0 || lineLen>72) return 0;

flag = 1; break;

case 'R': if (lineLen>72) return 0;

while (lineLen>0) {

if (hexToDec(readFrag(2, &objLine), 's') < 0) return 0;

objLine += 8;

lineLen -= 8;

}

flag = 1; break;

case 'T': if (lineLen < 8) return 0;

objLine += 6;

if (hexToDec(readFrag(2, &objLine), 's') < 0) return 0;

flag = 1; break;

case 'M': flag = 1; break;

case 'E': if (!(lineLen == 0 || lineLen==6)) return 0;

flag = 1; break;

case '.': flag = 1; break;

}

if (!flag)return 0;

return 1;

}

char\* readFrag(int fSize, char\*\* f) {

char\* frag=(char\*)malloc(sizeof(char)\*(fSize+1));

int i;

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

frag[i] = (\*f)[i];

frag[i] = '\0';

strtok(frag, " \n\t");

return frag;

}

int isDuplicate(char\* str,int idx,char REC) {

ESTAB\_NODE \*curr;

for (curr = est\_head[idx]; curr; curr = curr->next) {

if (REC == 'H') {

if (!strcmp(curr->ctrlsect, str)) return 1;

}

if (REC == 'M') {

if (!strcmp(curr->symb, str)) return 1;

}

}

return 0;

}

int relocation(char \*REFLINE, int file\_num,char \*line) {

ESTAB\_NODE \*est\_curr;

int i,j,sflag=0,pow=1;

int addr, sign = 1, REFLEN = strlen(REFLINE), halfB, temp = 0, vtemp = 0;

char \*reloc,\*symb,\*v\_p;

if ((addr = hexToDec(readFrag(6, &line), 's')) == -1) return 0; // Address

addr+=CSADDR; // needed to be modified

line += 6;

if((halfB = hexToDec(readFrag(2, &line), 's'))==-1) return 0; // halfbytes

line += 2; // to be modified

if (line[0] == '-') sign = -1; // positive

line++; // or negative

reloc = readFrag(6, &line); // get reference number

// or reference symbol

while (REFLEN>0) {

if (!strcmp("01", reloc)) {

sflag = 1;

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

if (est\_head[i]->adr == CSADDR) symb = est\_head[i]->symb;

}

}

else if (!strcmp(readFrag(2, &REFLINE), reloc)) {

sflag = 1;

REFLINE += 2;

symb=readFrag(6, &REFLINE);

break;

}

REFLINE += 8;

REFLEN -= 8;

}

if (!sflag) return 0;

sflag = 0;

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

for (est\_curr = est\_head[i]; est\_curr; est\_curr = est\_curr->next) {

if (!strcmp(est\_curr->symb, symb)) {

temp = est\_curr->adr;

for (j = 2; j >= 0; j--) {

vtemp += v\_mem[addr + j] \* pow;

pow \*= 256;

}

vtemp += sign\*temp;

for (j = 2; j >= 0; j--) {

v\_mem[addr + j] = vtemp%256;

vtemp /= 256;

}

sflag = 1;

}

}

}

if (!sflag) return 0;

return 1;

}

int loader(int file\_num,char\*\* obj\_file){

FILE\* fp[3];

int i, j;

int T\_ADDR, T\_LEN,total\_len=0,REL\_ADDR;

char buff[100],REC\_TYPE;

char \*linep,line\_len,\*temp,REFLINE[72];

ESTAB\_NODE \*temp\_est;

CSNUM = file\_num;

for (i = 0; i < 10; i++) REG[i] = 0; // init register to 0

/\* pass 1 \*/

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

if (!(fp[i] = fopen(obj\_file[i], "r"))) {

printf("no program to load on memory!\n");

return 0;

}

while (fgets(buff, 100, fp[i])) {

linep = buff;

strtok(buff, "\n\t");

REC\_TYPE = buff[0];

linep++;

if (!(isValisLine(linep,REC\_TYPE)))

return 0;

if (REC\_TYPE == 'H') {

if(!i) CSADDR = PROGADDR;

line\_len = strlen(linep);

temp = linep+6;

REL\_ADDR = hexToDec(readFrag(6, &temp), 'a');

temp\_est = createEstabNode(linep, REC\_TYPE);

if (isDuplicate(temp\_est->ctrlsect, i, REC\_TYPE)) {

printf("duplicate external symbol\n"); return 0;

}

createEstab(temp\_est,i);

}

else if (REC\_TYPE == 'D') {

line\_len = strlen(linep);

while (line\_len) {

temp\_est = createEstabNode(linep, REC\_TYPE, NULL);

if (isDuplicate(temp\_est->symb, i, REC\_TYPE)) {

printf("duplicate external symbol\n"); return 0;

}

createEstab(temp\_est, i);

linep += 12;

line\_len /= 12;

}

}

}

CSADDR += CSLTH;

}

/\* pass 2 \*/

CSADDR = PROGADDR;

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

fseek(fp[i], 0, 0); //file pointer reset

while (fgets(buff, 100, fp[i])) {

linep = buff;

strtok(buff, "\n\t");

REC\_TYPE = buff[0];

linep++;

if (!(isValisLine(linep, REC\_TYPE)))

return 0;

if (REC\_TYPE == 'H' || REC\_TYPE == 'D') continue;

else if (REC\_TYPE == 'R') strcpy(REFLINE ,linep);

else if (REC\_TYPE == 'T') {

T\_ADDR = hexToDec(readFrag(6, &linep), 's') - REL\_ADDR + CSADDR; linep += 6;

T\_LEN = hexToDec(readFrag(2, &linep), 's'); linep += 2;

for (j = 0; j < T\_LEN; j++) { // load objcode to memory;

v\_mem[T\_ADDR + j] = hexToDec(readFrag(2, &linep),'s');

linep += 2;

}

}

else if (REC\_TYPE == 'M') {

if (!relocation(REFLINE, file\_num, linep)) {

printf("reference error\n");

return 0;

}

}

else if (i == file\_num - 1 && REC\_TYPE == 'E') {

CSADDR = PROGADDR;

}

}

CSADDR += est\_head[i]->len;

fclose(fp[i]);

}

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

printf("section\tname\n");

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

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

for (temp\_est = est\_head[i]; temp\_est; temp\_est = temp\_est->next) {

if (temp\_est == est\_head[i]) {

printf("%-6s\t", temp\_est->ctrlsect);

printf("% \t");

printf("%+04X \t", temp\_est->adr);

printf("%+04X\n", temp\_est->len);

}

else {

printf("% \t");

printf("%-6s\t", temp\_est->symb);

printf("%+04X\n", temp\_est->adr);

}

}

total\_len += est\_head[i]->len;

PROGLEN = total\_len;

CSADDR = PROGADDR;

}

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

printf(" \t \ttotal length\t%X\n", PROGLEN);

}

int operate(int fSize,char\*frag,char \*op,int\* XBPE,int NI) {

int i,temp,shift;

char r1,r2,rt1[2],rt2[2];

int DISP;

int TA,TA\_VAL=0;

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

char\*opSet[58] = { "18","58","90","40","B4","28","88","A0","24","64","9C",

"C4","C0","F4","3C","30","34","38","48","00","68","50","70","08","6C",

"74","04","D0","20","60","98","C8","44","D8","AC","4C","A4","F0","EC",

"0C","78","54","80","D4","14","7C","E8","84","10","1C","5C","94","B0",

"E0","F8","2C","B8","DC" };

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

if (fSize == 1) {

rt1[0] = frag[0]; rt1[1] = '\0';

rt2[0] = frag[1]; rt2[1] = '\0';

r1 = hexToDec(rt1, 's') % 16;

r2 = hexToDec(rt2, 's') % 16;

}

else if (fSize >= 2) {

DISP = hexToDec(frag, 's');

if (frag[0] > '7') {

DISP = (4096 - DISP)\*-1;

}

TA = DISP;

if (XBPE[3])

TA += REG[1]; // indexing mode

if (XBPE[2])

TA += REG[3]; // BASE relative

else if (XBPE[1])

TA += REG[8]; // PC relative

if (NI == 1) { // immediate

TA\_VAL = TA;

}

else if (NI == 2) { // indirect

TA = v\_mem[TA] \* 0x10000 + v\_mem[TA + 1] \* 0x100 + v\_mem[TA + 2];

TA\_VAL = v\_mem[TA] \* 0x10000 + v\_mem[TA+1]\* 0x100 +v\_mem[TA+2];

}

else if (NI == 3) { // simple

TA\_VAL = v\_mem[TA]\* 0x10000 + v\_mem[TA+1] \* 0x100 + v\_mem[TA+2];

}

}

TA\_VAL %= 0x1000000;

for (i = 0; i < 58; i++) {

if (!strcmp(op, opSet[i])) break;

}

switch (i) {

case 0: REG[0] += TA\_VAL; REG[0]; break; //ADD

case 1: break; //ADDF

case 2: REG[r2] += REG[r1]; break; //ADDR

case 3: REG[0] &= TA\_VAL; break; //AND

case 4: REG[r1] = 0; break; //CLEAR

case 5: if (REG[0] == TA\_VAL) REG[9] = '='; //COMP

else if (REG[0] < TA\_VAL) REG[9] = '<';

else REG[9] = '>'; break;

case 6: break; //COMPF

case 7: if (REG[r1]==REG[r2]) REG[9] = '='; //COMPR

else if (REG[r1] < REG[r2]) REG[9] = '<';

else REG[9] = '>'; break;

case 8: REG[0] /= TA\_VAL; break; //DIV

case 9: break; //DIVF

case 10:REG[r2] /= REG[r1]; break; //DIVR

case 11:break; //FIX

case 12:break; //FLOAT

case 13:break; //HIO

case 14:REG[8] = TA; break;

case 15:if (REG[9] == '=') REG[8] = TA; break; //JEQ

case 16:if (REG[9] == '<') REG[8] = TA; break; //JGT

case 17:if (REG[9] == '>') REG[8] = TA; break; //JLT

case 18:REG[2] = REG[8]; REG[8] = TA; break; //JSUB

case 19:REG[0] = TA\_VAL; break; //LDA

case 20:REG[3] = TA\_VAL; break; //LDB

case 21:REG[0] = TA\_VAL % 0X100; break; //LDCH

case 22:break; //LDF

case 23:REG[2] = TA\_VAL; break; //LDL

case 24:REG[4] = TA\_VAL; break; //LDS

case 25:REG[5] = TA\_VAL; break; //LDT

case 26:REG[1] = TA\_VAL; break; //LDX

case 27:break; //LPS

case 28:REG[0] \*= TA\_VAL; break; //MUL

case 29:break; //MULF

case 30:REG[r2] \*= REG[r1]; break; //MULR

case 31:break; //NORM

case 32:REG[0] |= TA\_VAL; break; //OR

case 33: break; //RD

case 34:REG[r2] = REG[r1]; break; //RMO

case 35:REG[8] = REG[2]; break; //RSUB

case 36:shift=1; //SHIFTL

for (i = 0; i <REG\_SIZE - (r2+1); i++) {

shift \*= 2;

}

temp = (REG[r1] / (0x1000000/shift)) % shift;

REG[r1] \*= shift;

REG[r1] += temp;

break;

case 37:break; //SIO

case 38:break; //SSK

case 39:temp = REG[0]; //STA

v\_mem[TA + 2] = temp % 0x100; temp /= 0x100;

v\_mem[TA + 1] = temp % 0x100; temp /= 0x100;

v\_mem[TA + 0] = temp % 0x100; break;

case 40:temp = REG[3]; //STB

v\_mem[TA + 2] = temp % 0x100; temp /= 0x100;

v\_mem[TA + 1] = temp % 0x100; temp /= 0x100;

v\_mem[TA + 0] = temp % 0x100; break;

case 41:v\_mem[TA]= REG[0] % 0x100; //STCH

case 42:break; //STF

case 43:break; //STI

case 44:temp = REG[2]; //STL

v\_mem[TA + 2] = temp % 0x100; temp /= 0x100;

v\_mem[TA + 1] = temp % 0x100; temp /= 0x100;

v\_mem[TA + 0] = temp % 0x100; break;

case 45:temp = REG[4]; //STS

v\_mem[TA + 2] = temp % 0x100; temp /= 0x100;

v\_mem[TA + 1] = temp % 0x100; temp /= 0x100;

v\_mem[TA + 0] = temp % 0x100; break;

case 46:temp = REG[9]; //STSW

v\_mem[TA + 2] = temp % 0x100; temp /= 0x100;

v\_mem[TA + 1] = temp % 0x100; temp /= 0x100;

v\_mem[TA + 0] = temp % 0x100; break;

case 47:temp = REG[5]; //STT

v\_mem[TA + 2] = temp % 0x100; temp /= 0x100;

v\_mem[TA + 1] = temp % 0x100; temp /= 0x100;

v\_mem[TA + 0] = temp % 0x100; break;

case 48:temp = REG[1]; //STX

v\_mem[TA + 2] = temp % 0x100; temp /= 0x100;

v\_mem[TA + 1] = temp % 0x100; temp /= 0x100;

v\_mem[TA + 0] = temp % 0x100; break;

case 49:REG[0] -= TA\_VAL; //SUB

case 50:break; //SUBF

case 51:REG[r2] -= REG[r1]; break; //SUBR

case 52:break; //SVC

case 53:REG[9] = '<'; break; //TD

case 54:break; //TIO

case 55:REG[1]++; //TIX

if (REG[1] == TA\_VAL) REG[9] = '=';

else if(REG[1]<TA\_VAL) REG[9] = '<';

else REG[9] = '>';

break;

case 56: //TIXR

REG[1]++;

if (REG[1] == REG[r1])

REG[9] = '=';

else if (REG[1]<REG[r1]) REG[9] = '<';

else REG[9] = '>';

case 57:break; //WD

}

}

char\* hexToStr(int fSize,int offset,int NI) {

char \*str = (char\*)calloc(fSize \* 2+1,sizeof(char));

int i, pow = 1, res, val = 0;

for (i = fSize-1; i >= 0; i--) {

val += v\_mem[REG[8] + i+offset] \* pow;

pow \*= 0x100;

}

if(NI)

val -= NI;

if (fSize == 2) val %= 0x1000;

else if (fSize == 3) val %=0x100000;

str[strlen(str)] = '\0';

switch (fSize) {

case 1:

res = val % 16;

if (10 <= res&&res <= 15) str[1]= res - 10 + 'A';

else if (0 <= res&&res <= 9) str[1] = res + '0';

val /= 16;

res = val % 16;

if (10 <= res&&res <= 15) str[0] = res - 10 + 'A';

else if (0 <= res&&res <= 9) str[0] = res + '0'; break;

case 2:

res = val % 16;

if (10 <= res&&res <= 15) str[3] = res - 10 + 'A';

else if (0 <= res&&res <= 9) str[3] = res + '0';

val /= 16;

res = val % 16;

if (10 <= res&&res <= 15) str[2] = res - 10 + 'A';

else if (0 <= res&&res <= 9) str[2] = res + '0';

val /= 16;

res = val % 16;

if (10 <= res&&res <= 15) str[1] = res - 10 + 'A';

else if (0 <= res&&res <= 9) str[1] = res + '0';

val /= 16;

res = val % 16;

if (10 <= res&&res <= 15) str[0] = res - 10 + 'A';

else if (0 <= res&&res <= 9) str[0] = res + '0'; break;

case 3:

res = val % 16;

if (10 <= res&&res <= 15) str[5] = res - 10 + 'A';

else if (0 <= res&&res <= 9) str[5] = res + '0';

val /= 16;

res = val % 16;

if (10 <= res&&res <= 15) str[4] = res - 10 + 'A';

else if (0 <= res&&res <= 9) str[4] = res + '0';

val /= 16;

res = val % 16;

if (10 <= res&&res <= 15) str[3] = res - 10 + 'A';

else if (0 <= res&&res <= 9) str[3] = res + '0';

val /= 16;

res = val % 16;

if (10 <= res&&res <= 15) str[2] = res - 10 + 'A';

else if (0 <= res&&res <= 9) str[2] = res + '0';

val /= 16;

res = val % 16;

if (10 <= res&&res <= 15) str[1] = res - 10 + 'A';

else if (0 <= res&&res <= 9) str[1] = res + '0';

val /= 16;

res = val % 16;

if (10 <= res&&res <= 15) str[0] = res - 10 + 'A';

else if (0 <= res&&res <= 9) str[0] = res + '0'; break;

}

return str;

}

int run(){

int addr,endBp;

char op[3], \*frag,ta[7];

INST\_NODE \* mn\_curr;

int i,temp,NI,XBPE[4],fSize,sum;

static int first\_run = 0;

if (first\_run == 0) { // on the first run

REG[8] = PROGADDR; // PC to PROGADDR

bp\_end = bp\_head;

if(bp\_end) endBp = bp\_head->adr; // run to break point

first\_run = 1; // set flag

REG[2] = PROGADDR + PROGLEN;

}

else { // not first run

if (!bp\_end) {

bp\_end = bp\_head;

if (bp\_end) {

if (bp\_end->adr < REG[8]) {

REG[2] = PROGADDR + PROGLEN;

endBp = PROGADDR + PROGLEN;

}

else

endBp = bp\_end->adr;

}

}

else endBp = bp\_end->adr;

}

if (!bp\_head) endBp = PROGADDR + PROGLEN; // run until end of the program

while (REG[8] < endBp) {

XBPE[3] = XBPE[2] = XBPE[1] = XBPE[0] = 0; // init XBPE to 0

temp = v\_mem[REG[8]];

if (!temp) {

REG[8]++;

continue;

}

NI = temp % 4;

temp -= NI;

strcpy(op,hexToStr(1, 0 ,NI)); // get opcode string

/\* DECREPTION \*//////////////////////////////////////

sum = 0;

for (i = 0; i<strlen(op); i++)

sum += op[i] + (op[i] - 'A')\*(i); // calculating index of hashtable

for (mn\_curr = mn\_head[sum % 20]; mn\_curr; mn\_curr=mn\_curr->next) { // forwarding linked list

if (!strcmp(mn\_curr->opcode,op)) { // if found

if (!strcmp(mn\_curr->inst\_type, "1")) fSize = 0; // format 1

else if (!strcmp(mn\_curr->inst\_type, "2")) fSize = 1; // format 2

else if (!strcmp(mn\_curr->inst\_type, "3/4")) { // format 3/4

temp = (v\_mem[REG[8]+1]/16 )%16; // \*

for (i = 0; i < 4; i++) { // get

XBPE[i] = temp % 2; // XBPE

temp /= 2; // bit

} // \*

if (XBPE[0]) fSize = 3; // format 3

else fSize = 2; // format 4

}

strcpy(ta,hexToStr(fSize, 1,0)); // get displacement string

REG[8] += fSize + 1; // PC register increment

operate(fSize,ta,mn\_curr->opcode,XBPE,NI); // call SIC/XE operator

break;

}

}

}

/\* PRINT REGISTER VALUE \*////////////////////////////

printf("A:%012X\t",REG[0]); printf("X:%012X\n", REG[1]);

printf("L:%012X\t", REG[2]); printf("PC:%012X\n", REG[8]);

printf("B:%012X\t", REG[3]); printf("S:%012X\n", REG[4]);

printf("T:%012X\n", REG[5]);

/\* SETTING BREAK POINT \*/

if (REG[8]!=PROGADDR+PROGLEN) { // program stopped on breakpoint

printf("Stop at check point[%04X]\n", bp\_end->adr);

if (bp\_end) bp\_end = bp\_end->next;

}

else { // program ended

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

bp\_end = bp\_head; // break point back to first

printf("End program\n");

}

}

int breakPoint(char \*para) {

BP\_NODE\* bp\_curr,\*bp\_prev,\*bp\_temp;

int bp\_adr,flag=0;

if (!est\_head[0]) {

printf("no program loaded\n");

return 0;

}

if (!para) {

printf("breakpoint\n");

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

for (bp\_curr = bp\_head; bp\_curr; bp\_curr = bp\_curr->next) {

printf("%-04X\n", bp\_curr->adr);

}

}

else if (!strcmp(para, "clear")) {

while (bp\_head) {

bp\_curr = bp\_head;

bp\_head = bp\_head->next;

free(bp\_curr);

}

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

}

else{

bp\_adr = hexToDec(para, 'a');

if (!(0 < bp\_adr&&bp\_adr < MAX\_MEM)) return 0;

if (!bp\_head) {bp\_tail = bp\_head = createBpNode(bp\_adr);}

else {

for (bp\_curr = bp\_head; bp\_curr; bp\_curr = bp\_curr->next) {

if (bp\_curr->adr == bp\_adr) { printf("already existing breakpoint\n"); return 0; }

else if (bp\_curr->adr > bp\_adr) {

bp\_temp = createBpNode(bp\_adr);

if (bp\_curr == bp\_head) {

bp\_temp->next = bp\_head;

bp\_head = bp\_temp;

}

else {

bp\_temp->next = bp\_curr;

bp\_prev->next = bp\_temp;

}

flag = 1;

break;

}

bp\_prev = bp\_curr;

}

if (!flag) {

bp\_tail->next = createBpNode(bp\_adr);

bp\_tail = bp\_tail->next;

}

}

printf("[ok] create breakpoint %04X\n", bp\_adr);

}

return 1;

}

void freeESTAB() {

int i;

ESTAB\_NODE \*est\_curr;

for (i = 0; i<3; i++) {

while (est\_head[i]) {

est\_curr = est\_head[i];

est\_head[i] = est\_head[i]->next;

free(est\_curr);

}

}

}