**과목명: 시스템프로그래밍**

**담당 교수 명: 김 지 환**

**2분반**

**<<Project #3>>**

**서강대학교 컴퓨터학과**

**20181618**

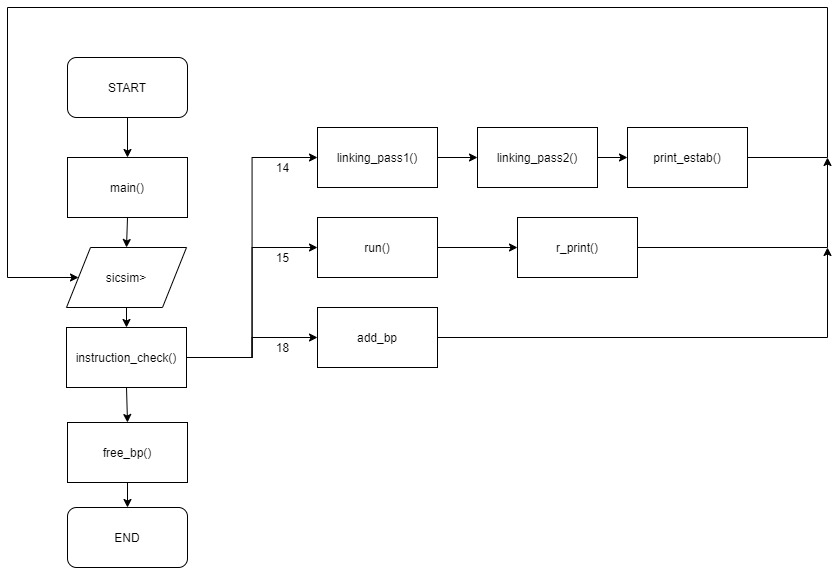
**김하늘**

목 차

1. **프로그램 개요**
2. **프로그램 설명**
   1. 프로그램 흐름도
3. **모듈 정의**
   1. 모듈 이름 : void print\_estab();
      1. 기능
      2. 사용 변수
   2. 모듈 이름 : void linking\_pass1(int idx);
      1. 기능
      2. 사용 변수
   3. 모듈 이름 : void linking\_pass2(int n);
      1. 기능
      2. 사용 변수
   4. 모듈 이름 : void add\_bp(int n);
      1. 기능
      2. 사용 변수
   5. 모듈 이름 : void run();
      1. 기능
      2. 사용 변수
   6. 모듈 이름 : int m\_value(int tInt, int xbpe[], int x, int y, int type, int value);
      1. 기능
      2. 사용 변수
4. **전역 변수 정의**
   1. int prevBP
   2. int curaddr
   3. int tempaddr
   4. int linkFlag
   5. int linkEndFlag
   6. int cc
   7. int rA, rX, rL, rPC, rB, rS, rT
5. **코드**
   1. 20181618.c
   2. 20181618.h
6. 프로그램 개요

본 프로젝트 #2 에서 구현한 셀(shell)에 linking과 loading 기능을 추가하는 프로그램으로, 프로젝트 #2 에서 구현된 assemble 명령을 통해서 생성된 object 파일을 link시켜 메모리에 올리는 일을 수행한다. 주소 지정 명령어, Linking Loader, 프로그램 실행 명령어, debug 명령어 등을 구현한다.

1. 프로그램 설명
   1. 프로그램 흐름도



위 프로그램 흐름도는 프로젝트 3에서 추가된 내용을 위주로 작성하였다. 실행되는 함수는 대략적으로 위의 흐름을 따라가고, 몇몇 간단한 수행은 함수를 이용하지 않았다.

우선 progaddr 명령어는 입력받은 값을 전역변수에 저장한다. 함수는 사용하지 않았다.

loader 명령어는 pass1, pass2 두 단계로 진행된다. linking\_pass1()함수에서 파일이 유효한지 확인한 뒤 Header recode와 Define record를 처리한다. H record는 파일 이름, 시작 주소, 프로그램 길이를 저장받아오고 D record에선 심볼 이름, 시작 주소를 받아온다. 이 정보들은 구조체를 통해 list를 이용해 저장한다.

linking\_pass2()함수에선 값을 프로젝트1에서 구현한 가상 메모리에 저장한 뒤 pass1에서 처리한 estable을 이용하여 relocation을 수행한다. 우선 R record에서 참조할 심볼을 estab을 통해 해당 주소값을 배열에 인덱싱하여 저장한다. 배열의 0번 인덱스에는 현재 파일의 상대주소가 저장되어있다. T record에선 위치와 길이를 받은 뒤 가상메모리에 모두 저장한다. M record에선 메모리에 저장된 값에 접근하여 symbol을 적당히 더하고 빼주면서 relocation을 진행했다. 계산한 값을 다시 메모리에 저장하면서 함수는 끝이 나며 이후 estable을 출력한다.

bp 명령어는 3가지로 나뉜다. bp [address]으로 break point를 지정한다. 입력된 bp값은 모두 리스트의 형태로 저장된다. bp clear를 입력하면 현재까지 저장된 모든 bp의 값을 지운다. bp만 입력하면 현재까지 저장된 모든 bp list를 출력한다.

run은 메모리에 로드된 프로그램을 실행시키는 명령어이다. 처음 실행되면 모든 레지스터는 초기화되고 progaddr에 지정된 주소값부터 실행되며, 우선 2바이트 분량을 읽어 어떤 명령어인지, nixbpe의 정보를 구한 뒤 필요에따라 1바이트 또는 3바이트를 더 읽어와서 TA값을 구한다. TA에 접근해서 레지스터 있는 값을 저장하거나 TA에 저장된 값을 레지스터에 저장하는 등의 수행은 모두 m\_value()을 통해 이루어진다. 미리 구한 nixbpe를 통해 적절히 pc나 base, x 값을 더해주어 TA를 구하고 명령어의 종류에 따라 type 변수를 파라미터로 받아 처리한다.

1. 모듈 정의
   1. 모듈 이름 : **void print\_estab();**
      1. 기능

linking & loading이 완료된 로드맵을 출력한다.

* + 1. 사용 변수

int total - 총 길이를 계산하기 위한 변수

* 1. 모듈 이름 : **void linking\_pass1(int idx);**
     1. 기능

Loader 과정 중 첫번째 pass1의 단계를 구현하였다. obj 파일이 존재하는지 확인한 뒤 H record와 D record를 처리한다. H recode에서는 프로그램의 이름, 시작 주소, 길이를 받고 D에선 정의되는 변수들의 이름 및 주소를 받아온다. 각 심볼은 list의 형태로 저장된다.

* + 1. 사용 변수

FILE\* f - file 포인터

char buf[500] - 한 줄 읽은 문자열이 저장됨

char temp[10] - 범용 문자열

int i, j, idx - 인덱싱에 사용되는 변수

int tAddr - address값이 임시로 저장되는 변수

* 1. 모듈 이름 : **void linking\_pass2(int n);**
     1. 기능

loader 과정의 두번째 단계로, pass2를 구현하였다. obj 파일을 한 번 더 읽으며 T record의 값을 우선 메모리에 올린 후, pass1에서 linking하는 모든 파일들의 symbol을 정리한 리스트를 이용하여 M record에서 요구하는 사항들을 처리한다.

* + 1. 사용 변수

FILE\* f - 파일 포인터

char buf[500] - 한 줄을 읽어 저장한다.

char temp[10] - 범용 문자열

int i, idx, ref\_idx - 범용 인덱스 변수

int rFlag - 공백 구분 플래그

int ref[10] - reference 변수의 주소 값이 저장된다.

* 1. 모듈 이름 : **void add\_bp(int n);**
     1. 기능

입력받은 bp 주소 값이 파라미터로 들어오며 이를 bp 리스트에 추가시켜주는 함수이다.

* + 1. 사용 변수

bp\* node - 리스트에 추가될 노드 초기화

bp\* temp - 리스트를 서치할때 사용되는 포인터

* 1. 모듈 이름 : **void run();**
     1. 기능

run 명령어를 수행하는 함수로, 메모리에 로드된 object code를 읽어 역어셈블하여 프로그램을 실행시킨다. 모든 명령어를 정의하지는 않았다. 입력받은 bp list를 object code 한 줄 마다 체크하며 break point와 pc가 동일할경우 현재 레지스터 값을 출력하며 run을 종료시킨다. 단, run이 실행되고 최초로 어셈블하는 코드는 bp체크를 하지 않는다. 프로그램의 끝에 다다를 경우 최종 레지스터 값을 출력하며 종료된다.

* + 1. 사용 변수

char temp[10], temp2[10] - 범용 temp 문자열이다.

int xbpe[10] - object code의 xbpe값을 저장한다.

int st\_i, st\_j - 메모리에 접근할 인덱스가 저장된다.

int tInt, ta, a, x, y - 범용 정수형 변수이다.

int key - 명령어를 구분할 값이 저장되어있다.

int bpFlag- bp일때 프로그램을 종료시키기 위한 플래그이다.

* 1. 모듈 이름 : **int m\_value(int tInt, int xbpe[], int x, int y, int type, int value);**
     1. 기능

명령어에 따른 TA와 관련된 값을 반환한다. type에 입력된 값에 따라 TA 주소값, TA에 저장된 메모리 값, 또는 레지스터의 값을 바로 메모리에 저장한다. type = 0일때 TA에 저장된 메모리 값이 반환되고, type = 1일때 value로 들어온 값을 TA 메모리에 저장하며, type = 2일때는 TA의 주소값이 그대로 반환된다.

* + 1. 사용 변수

int tInt - 3이면 simple, 2면 indirect, 1이면 immediate

int xbpe[10] - 명령어의 xbpe 정보가 저장되어있다

int x, y - 접근할 메모리의 인덱스 정보

int value - 메모리에 저장될 값

char temp[10], temp2[10] - 범용 temp 문자열이다.

int ta - TA값이 저장된다.

1. 전역 변수 정의
   1. int prevBP

직전에 멈춘 BP값이 저장된다. 반복문과 같이 접근했던 코드에 다시 접근할 경우를 위해 지정하였다.

* 1. int curaddr

progaddr에 입력된 값이 저장되어있다. load 과정을 거치며 값이 변한다.

* 1. int tempaddr

progaddr 값이 저장된다. 다시 progaddr값이 입력되기 전까지 바뀌지 않는다.

* 1. int linkFlag

loader 명령어에 입력된 파일의 갯수가 저장되어있다.

* 1. int linkEndFlag

읽어온 파일이 정상적이지 않을 경우 1이 된다.

* 1. int cc

부등호 비교 결과가 저장된다. >이면 1, =이면 0, <이면 -1이다.

* 1. int rA, rX, rL, rPC, rB, rS, rT

run을 실행시킬 때 사용되는 레지스터이다.

1. 코드
   1. 20181618.h

//----------- 링킹로더 결과 심볼 정보가 저장되는 구조체----------------

typedef struct loader{

char name[10];

int addr;

int length;

int type;

struct loader \*next;

}loader;

//------------ bp 리스트 구조체-----------------

typedef struct bp{

int addr;

struct bp \*next;

}bp;

//============로드맵 출력===============

// linking & loading이 완료된 로드맵을 출력한다.

//=====================================

void print\_estab();

//============pass 1===============

// H, D record를 분석하여 estable에 추가한다.

//=====================================

void linking\_pass1(int idx);

//============pass 2===============

// T record 값을 메모리에 올린 뒤

// R record의 심볼 정보를 바탕으로

// M record의 요구사항을 수행한다.

//=====================================

void linking\_pass2(int n);

//==========BP list===========

// 입력받은 bp 주소를 리스트에 추가한다.

//==========================

void add\_bp(int n);

//============RUN=============

// 메모리에 로드된 값을 이용해

// 프로그램을 실행시킨다.

//=============================

void run();

//============메모리 접근===============

// object 코드를 분석하여 TA의 값을 구해

// 반환하거나 접근하여 수정한다.

//=====================================

int m\_value(int tInt, int xbpe[], int x, int y, int type, int value);

//==========register print===========

// 현재 레지스터에 저장된 값 출력

//====================================

void r\_print();

int prevBP; // 직전에 멈춘 BP값이 저장된다.

int curaddr; // progaddr 값 저장

int tempaddr; // 임시로 저장되는 progaddr

int linkFlag; // loader 명령어에 입력된 파일의 개수

int linkEndFlag; // 링킹시에 파일 이름 잘못되었을 때 1이 된다.

int cc; // 부등호 결과 저장

int rA, rX, rL, rPC, rB, rS, rT; // 레지스터

loader \*l\_table; // 심볼 리스트 head

loader \*tNode; // 심볼 임시 노드

bp \*bpList; // bp 리스트 head

* 1. 20181618.c

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

#include <dirent.h>

#include "20181618.h"

int main(){

int i;

int k\_inst = 0;

hash\* temp;

bp\* temp1;

char inst[50];

char\* t\_str;

readfile(); // opcode.txt를 읽고 저장

mem\_reset(); // memory 초기화

history = (his\*)malloc(sizeof(his));

history->next = NULL; // 명령어 history list의 head 초기화

l\_head = (line\*)malloc(sizeof(line));

l\_head->next = NULL;

l\_head->loc = -1;

bpList = (bp\*)malloc(sizeof(bp));

bpList->next = NULL;

cur\_add = 0; end\_flag = 0; cur\_i = 0; rPC = 0;

while(end\_flag == 0){ // end\_flag가 1이 될때까지 반복

printf("sicsim> ");

fgets(inst, 50, stdin);

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

k\_inst = instruction\_check(inst); // 올바른 명령어인지, 어떤 명령어인지 반환받는다

if(k\_inst != -1)

add\_history(inst); // 올바른 명령어일경우 history에 추가

switch(k\_inst){

case 0: // help

printf("h[elp]\nd[ir]\nq[uit]\nhi[story]\ndu[mp] [start, end]\ne[dit] address, value\n");

printf("f[ill] start, end, value\nreset\nopcode mnemonic\nopcodelist\n");

printf("assemble filename\ntype filename\nsymbol\n");

break;

case 1: // dir

show\_directory();

break;

case 2: // quit

end\_flag = 1;

break;

case 3: // history

print\_history();

break;

case 41: // dump

Dump(1);

break;

case 42: // dump start

Dump(2);

break;

case 43: // dump start, end

Dump(3);

break;

case 5: // dump address, value

edit();

break;

case 6: //fill start, end, value

fill();

break;

case 7: // reset

mem\_reset();

break;

case 8: // opcode

t\_str = hash\_function(mne);

if(strcmp(t\_str, "\0") == 0) printf(" not exist!\n"); // 리스트에 없을때

else printf("opcode is %s\n", t\_str);

break;

case 9: // opcode list

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

printf(" %d ",i);

temp = h\_table[i]->next;

while(temp != NULL){

printf("-> [%s,%s,%d] ", temp->mnemonic, temp->op, temp->type);

temp = temp->next;

}

printf("\n");

}

break;

case 10://assemble filename

symbol = (sym\*)malloc(sizeof(sym));

symbol->next = NULL;

sym\_flag=1;

if(assemble() == 0){

printf("Successfully assemble %s.asm\n", filename);

}

break;

case 11://type filename

type();

break;

case 12://symbol

if(sym\_flag==1) sort\_symbol();

else printf("not exist symbol\n");

break;

case 13://progaddr

tempaddr=curaddr; // curaddr는 pass1에서 수정되므로 저장해놓는다.

break;

case 14://loader

l\_table = (loader\*)malloc(sizeof(loader));

l\_table->next = NULL;

tNode = l\_table; // 링킹로더 심볼 정보 저장할 list head

curaddr = tempaddr;

linkEndFlag = 0;

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

linking\_pass1(i);

} // 파일 수 만큼 pass1 진행

if(linkEndFlag==1) break; // 파일이 존재하지 않을 경우 종료한다.

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

linking\_pass2(i);

} // 파일 수 만큼 pass2 진행

print\_estab(); // estable 출력

prevBP=-1; // 이후 run에 사용될 BP 플래그 초기화

break;

case 15: //run

run();

break;

case 16: //bp

temp1 = bpList;

if(temp1->next == NULL){

printf(" Not exist! \n");

} // bp 리스트에 아무것도 없을 경우

else{

printf(" breakpoint \n");

printf("============\n");

temp1 = temp1->next;

while(temp1 != NULL){

printf(" %s\n", hex(temp1->addr, 4));

temp1 = temp1->next;

}

}

break;

case 17: //bp clear

bpList->next = NULL;

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

break;

case 18: //bp [addr]

break;

default:

break;

}

}

//동적할당한 메모리들을 free해준다.

free\_history();

free\_opcode();

free\_bp();

if(sym\_flag==1){

free\_symbol();

free\_line();

}

return 0;

}

//==========BP list===========

// 입력받은 bp 주소를 리스트에 추가한다.

//==========================

void add\_bp(int n){

bp\* node = (bp\*)malloc(sizeof(bp));

node->addr = n;

node->next = NULL;

bp\* temp = bpList;

while(temp->next != NULL){

temp = temp->next;

}

temp->next = node;

printf("\t\t[ok] create breakpoint %s\n", hex(n, 4));

}

//==========register print===========

// 현재 레지스터에 저장된 값 출력

//====================================

void r\_print(){

printf("A : %s ", hex(rA, 6));

printf("X : %s\n", hex(rX, 6));

printf("L : %s ", hex(rL, 6));

printf("PC : %s\n", hex(rPC, 6));

printf("B : %s ", hex(rB, 6));

printf("S : %s\n", hex(rS, 6));

printf("T : %s\n\n", hex(rT, 6));

}

//============RUN=============

// 메모리에 로드된 값을 이용해

// 프로그램을 실행시킨다.

//=============================

void run(){

char temp[10], temp2[10]; // 범용 temp 문자열이다.

int xbpe[10]; // object code의 xbpe값을 저장한다.

int st\_i, st\_j; // 메모리에 접근할 인덱스가 저장된다.

int tInt=0, key;

int ta, x, y, a, bpFlag = 0; // 기타 범용 정수형 변수

bp\* bp\_temp; // bp list 탐색 포인터

if(rPC == 0){

cc=rA=rX=rPC=rB=rS=rT=0;

rL = 4215;

} // 프로그램 시작시 레지스터를 초기화한다.

while(1){

if(strcmp(hex(rPC, 4), "1077")==0){

r\_print();

printf("\t\t End Program\n");

rL = 4215;

rPC = 0;

break;

} // 프로그램이 끝일때 종료

bp\_temp = bpList->next;

while(bp\_temp != NULL){

if(rPC == prevBP){

bp\_temp = bp\_temp->next;

continue;

} // 직전에 도달한 BP이면 건너뛴다. (반복문)

if(rPC == bp\_temp->addr){

r\_print();

printf("\t\t Stop at checkpoint[%s]\n", hex(rPC,4));

bpFlag = 1;

prevBP = rPC;

break;

} // bp에서 정지

bp\_temp = bp\_temp->next;

}

if(bpFlag==1){

bpFlag = 0;

break;

}

// 배열올라가는 시작 주소 인덱스 계산

strcpy(temp , hex(rPC, 6));

st\_j = hexTodec("", temp[5], 1);

temp[5] = '\0';

st\_i = hexTodec(temp, 0, 0);

// pc에 해당하는 메모리 접근

temp[0] = '\0';

strcat(temp, hex(m[st\_i][st\_j++],2));

if(st\_j==16){

st\_i +=1; st\_j=0;

}

strcat(temp, hex(m[st\_i][st\_j],2));

// 3 simple 2 indirect 1 immediate

tInt = hexTodec("", temp[1], 1)%4;

// xbpe 값 계산

xbpe[3] = (hexTodec("", temp[2], 1))%2;

xbpe[2] = (hexTodec("", temp[2], 1)/2)%2;

xbpe[1] = (hexTodec("", temp[2], 1)/4)%2;

xbpe[0] = (hexTodec("", temp[2], 1)/8)%2;

xbpe[4] = '\0';

// 명령어 key 값 계산

temp2[0] = temp[0]; temp2[1] = temp[1];

temp2[2] = '\0';

key = hexTodec(temp2, 0, 0)-tInt;

//tInt로 ni, xbpe[]로 xbpe 값들 넘겨준다.

switch(key){

case 0: // LDA

ta = m\_value(tInt, xbpe, st\_i, st\_j, 0, 0);

rA = ta;

break;

case 104://LDB

ta = m\_value(tInt, xbpe, st\_i, st\_j, 0, 0);

rB = ta;

break;

case 80://LDCH

ta = m\_value(tInt, xbpe, st\_i, st\_j, 2, 0);

// 접근할 메모리 인덱스 계산

temp[0]='\0';

strcpy(temp, hex(ta, 6));

y = hexTodec("", temp[5], 1);

temp[5] = '\0';

x = hexTodec(temp, 0, 0);

// A 끝바이트에 값 저장

temp2[0] = '\0';

strcpy(temp2, hex(rA, 6));

temp2[4] = hex(m[x][y],2)[0];

temp2[5] = hex(m[x][y],2)[1];

temp2[6] = '\0';

rA = hexTodec(temp2, 0, 0);

break;

case 116://LDT

ta = m\_value(tInt, xbpe, st\_i, st\_j, 0, 0);

rT = ta;

break;

case 12://STA

ta = m\_value(tInt, xbpe, st\_i, st\_j, 1, rA);

break;

case 84://STCH

ta = m\_value(tInt, xbpe, st\_i, st\_j, 2, rA);

// 메모리 접근할 인덱스 계산

temp[0]='\0';

strcpy(temp, hex(ta, 6));

y = hexTodec("", temp[5], 1);

temp[5] = '\0';

x = hexTodec(temp, 0, 0);

// A의 끝 바이트의 값 추출 및 저장

temp[0]='\0';

strcat(temp, hex(rA, 6));

temp[0] = temp[4];

temp[1] = temp[5];

temp[2] = '\0';

m[x][y] = hexTodec(temp, 0, 0);

break;

case 20://STL

ta = m\_value(tInt, xbpe, st\_i, st\_j, 1, rL);

break;

case 16://STX

ta = m\_value(tInt, xbpe, st\_i, st\_j, 1, rX);

break;

case 40://COMP

ta = m\_value(tInt, xbpe, st\_i, st\_j, 0, 0);

if(rA>ta) cc=1;

else if(rA==ta) cc=0;

else cc=-1;

break;

case 160://COMPR 2형식

cc = 0;

rPC += 2;

break;

case 48://JEQ

if(cc==0) rPC = m\_value(tInt, xbpe, st\_i, st\_j, 2, 0);

else rPC += 3;

break;

case 60://J

rPC = m\_value(tInt, xbpe, st\_i, st\_j, 2, 0);

break;

case 56://JLT

if(cc<0) rPC = m\_value(tInt, xbpe, st\_i, st\_j, 2, 0);

else rPC += 3;

break;

case 72://JSUB

rL = rPC+4;

rPC = m\_value(tInt, xbpe, st\_i, st\_j, 2, 0);

break;

case 76://RSUB

rPC = rL;

break;

case 180://CLEAR 2형식

rPC += 2;

a = hexTodec("", temp[2], 1);

if(a==0) rA = 0;

else if(a==1) rX = 0;

else if(a==4) rS = 0;

break;

case 224://TD

rPC += 3;

cc = -1;

break;

case 216://RD

rPC += 3;

cc = 0;

break;

case 220://WD

rPC += 3;

break;

case 184://TIXR 2형식

rPC += 2;

rX += 1;

if(rX>rT) cc=1;

else if(rX==rT) cc=0;

else cc=-1;

break;

default:

break;

}

}

}

//============메모리 접근===============

// object 코드를 분석하여 TA의 값을 구해

// 반환하거나 접근하여 수정한다.

//=====================================

int m\_value(int tInt, int xbpe[10], int x, int y, int type, int value){

char temp[10], temp2[10];

int ta;

// dis 또는 addr를 3 half byte만큼 추출한다.

temp[0] = hex(m[x][y],2)[1];

temp[1] = '\0';

y++;

if(y==16){

x++; y=0;

}

strcat(temp, hex(m[x][y], 2));

if(xbpe[3]==0){ //3형식

rPC += 3;

ta = hexTodec(temp, 0, 0);

if(temp[0] == 'F'){

ta \*= -1;

ta = hexTodec(two\_comp(ta, 3), 0, 0)\*(-1);

} // 보수화

// x, b, p 설정

if(xbpe[0]==1) ta += rX;

if(xbpe[1]==1) ta += rB;

else if(xbpe[2]==1) ta+= rPC;

if(tInt==2){ //indirect

// 메모리 인덱스 계산 및 계산

temp[0]='\0';

strcpy(temp, hex(ta, 6));

y = hexTodec("", temp[5], 1);

temp[5] = '\0';

x = hexTodec(temp, 0, 0);

temp[0] = '\0';

strcat(temp, hex(m[x][y], 2));

y++;

if(y==16){

x++; y=0;

}

strcat(temp, hex(m[x][y], 2));

y++;

if(y==16){

x++; y=0;

}

strcat(temp, hex(m[x][y], 2));

ta = hexTodec(temp, 0, 0);

}

}

else{ //4형식

rPC += 4;

y++;

if(y==16){

x++; y=0;

}

strcat(temp, hex(m[x][y], 2));

ta = hexTodec(temp, 0, 0);

if(xbpe[0]==1) ta += rX;

}

if(type == 1){ //메모리에 값 저장한다

temp2[0] = '\0';

strcpy(temp2, hex(value, 6));

temp[0]='\0';

strcat(temp, hex(ta, 6));

y = hexTodec("", temp[5], 1);

temp[5] = '\0';

x = hexTodec(temp, 0, 0);

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

temp[0] = temp2[i\*2];

temp[1] = temp2[i\*2+1];

temp[2] = '\0';

m[x][y] = hexTodec(temp, 0, 0);

y++;

if(y==16){

x++; y=0;

}

} // 메모리 접근해서 값 저장

return ta;

}

else if(type == 2) return ta; // 메모리의 주소를 반환한다.

if(tInt == 1) return ta; // immdiate의 경우 TA의 값을 그대로 반환한다.

temp[0]='\0';

strcpy(temp, hex(ta, 6));

y = hexTodec("", temp[5], 1);

temp[5] = '\0';

x = hexTodec(temp, 0, 0);

temp[0] = '\0';

strcat(temp, hex(m[x][y], 2));

y++;

if(y==16){

x++; y=0;

}

strcat(temp, hex(m[x][y], 2));

y++;

if(y==16){

x++; y=0;

}

strcat(temp, hex(m[x][y], 2));

return hexTodec(temp, 0, 0); //접근할 메모리에 있는 3바이트 값 반환

}

//============로드맵 출력===============

// linking & loading이 완료된 로드맵을 출력한다.

//=====================================

void print\_estab(){

int total = 0;

tNode = l\_table->next;

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

printf("section name \n");

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

while(tNode != NULL){

if(tNode->type != 0){

printf("%s\t\t%s\t%s\n", tNode->name, hex(tNode->addr,4), hex(tNode->length, 4));

total += tNode->length;

}// H

else{

printf("\t%s\t%s\n", tNode->name, hex(tNode->addr, 4));

}// D

tNode = tNode->next;

}

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

printf("\t\t total length %s\n", hex(total, 4));

}