**과목명: 시스템프로그래밍**

**분반 01**

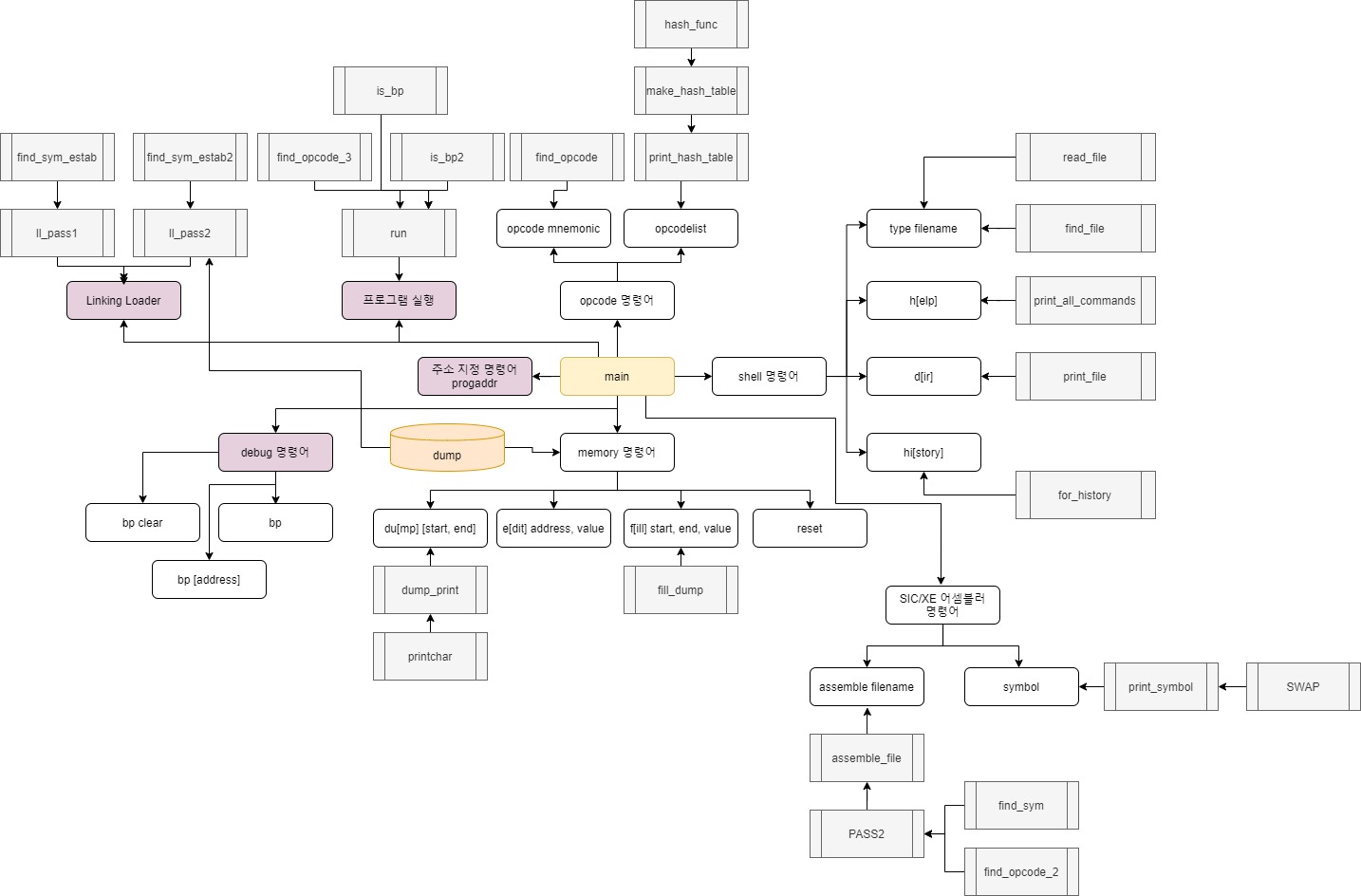
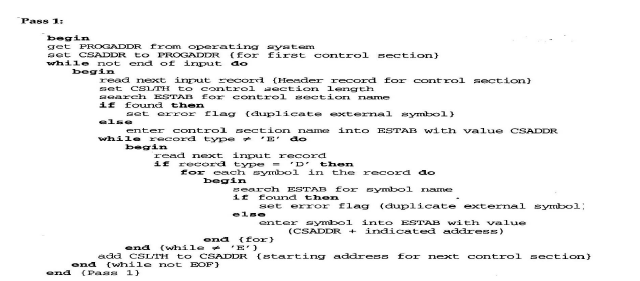
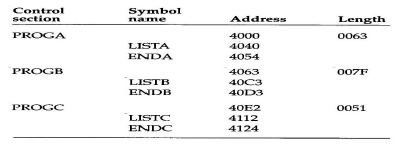
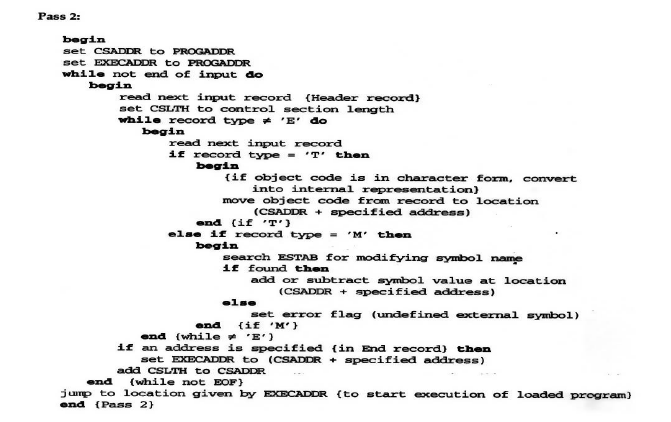
**<<Project #3>>**

**서강대학교 컴퓨터공학과**

**20181679**

**장서우**

목 차

1. **프로그램 개요**
   1. 목표
   2. 구현
2. **프로그램 설명**
   1. 프로그램 흐름도
3. **모듈 정의**
   1. main();
      1. 기능
      2. 주요변수
   2. int find\_sym\_estab(char \*sym);
      1. 기능
      2. 주요변수
   3. int find\_sym\_estab(char \*sym);
      1. 기능
      2. 주요변수
   4. int ll\_pass1(char \*file1, char \*file2, char \*file3);
      1. 기능
      2. 주요변수
   5. int ll\_pass2(char \*file1, char \*file2, char \*file3, unsigned char \*\*dump);
      1. 기능
      2. 주요변수
   6. int is\_bp(unsigned int value);
      1. 기능
      2. 주요변수
   7. int is\_bp2(unsigned int value);
      1. 기능
      2. 주요변수
   8. char\* find\_opcode\_3(unsigned int opcode)
      1. 기능
      2. 주요변수
   9. int run(unsigned char \*\*dump, unsigned int \*PC, unsigned int \*L, unsigned int \*A, unsigned int \*X, unsigned int \*B, unsigned int \*S, unsigned int \*T);
      1. 기능
      2. 주요변수
4. **전역 변수 정의**
   1. unsigned int PROG\_ADDR
   2. unsigned in bp[]
   3. int bp\_idx
   4. unsigned int RETADR
   5. unsigned int LENGTH
   6. type struct \_estab{…}ESTAB
   7. ESTAB \*eshead[]
   8. char \*\*ref
5. **코드 설명**
   1. 20181679.h
   2. 20181679.c
6. **프로그램 개요**
   1. 목표
      1. 프로젝트 1, 2에서 구현한 셀(shell)에 linking과 loading 기능을 추가한다. 프로젝트 2에서 구현된 assemble 명령을 통해서 생성된 object 파일을 link시켜 메모리에 올리는 일을 수행한다.
   2. 구현
      1. 주소 지정 명령어 (progaddr)
      2. Linking Loader (loader)
      3. 프로그램 실행 (run)
      4. debug 명령어 (bp)
7. **프로그램 설명**
   1. 프로그램 흐름도
8. **모듈 정의**
   1. main()
      1. 기능
         1. 프로젝트1, 2의 기능에서 다음과 같은 명령어들이 추가되었다.
            1. progaddr [address]: 주소 지정 명령어 -> 전역변수 PROG\_ADDR를 적당한 값으로 지정해준다.
            2. loader [object filename1] [object filename2] […]: Linking Loader ->ll\_pass1(), ll\_pass2() 함수 실행
            3. bp [address] / bp / bp clear: debug 명령어 -> 전역변수 bp[]를 초기화 시키고 address값을 break point로 지정하여 bp[]에 넣어준다. bp만 입력됐을 경우 존재하는 breakpoint를 전부 화면에 출력한다. bp clear가 입력됐을 경우 존재하는 breakpoint를 전부 삭제한다.
            4. run: 프로그램 실행 ->run()함수 실행
      2. 주요변수
         1. unsigned int PC, L, A, X, B, S, T: run 프로그램 실행에서 사용되는 레지스터 목록 + PC 값이다. main에서 각각 적당한 값으로 initialize 해주고, call by reference로 run()에 넘겨준다.
   2. int find\_sym\_estab(char \*sym)
      1. 기능
         1. ESTAB에서 symbol 또는 control section과 일치하는 symbol이 있는지 찾고, 있을 경우 1을 return, 없을 경우 0을 return한다.
         2. ll\_pass1()에서 ERROR check를 위해 사용된다.
      2. 주요변수
         1. 없음
   3. int find\_sym\_estab2(char \*sym)
      1. 기능
         1. 3.2와 똑 같은 기능을 한다. 찾았을 경우 해당 index를 return, 아닐 경우 -1을 return한다.
         2. ll\_pass2()에서 M을 처리하는 과정에서 해당 sym의 address를 얻어 오기 위해 사용된다.
      2. 주요변수
         1. 없음
   4. int ll\_pass1(char \*file1, char \*file2, char \*file3);
      1. 기능
         1. file1, file2, … 에 해당하는 object 파일을 읽어서 ESTAB을 생성한다.
         2. 이번 프로젝트에서 file의 최대 개수는 3개이다.
         3. PASS1 알고리즘은 다음과 같다.
         4. PROGADDR를 OS로부터 가져온다. 이번 프로젝트에서는 주소 지정 명령어 progaddr [address]를 사용하여 지정한 PROGADDR를 사용한다.
         5. 첫번째 control section의 주소(CSADDR)를 PROGADDR로 지정해준다.
         6. H 레코드를 읽어 control section의 길이를 지정해주고 ESTAB에 cs를 검색하여 없을 경우 cs와 해당 address, length를 추가하고, 있을 경우 에러 메시지가 출력 되도록 한다.
         7. D 레코드를 읽어 external symbol를 ESTAB에 추가한다. cs와 마찬가지로 ESTAB에서 우선 검색하고 이미 존재하는 symbol일 경우는 에러메시지를 출력한다. 없을 경우 ESTAB에 추가하고, CSADDR에 해당address를 더한 만큼을 ESTAB에 저장시켜준다.
         8. E 레코드에 도달하면 반복을 멈추고 CSADDR에 control section의 길이만큼 더해준다.
         9. 입력된 파일을 전부 읽을 때까지 반복한다.
         10. PASS1이 끝나면 다음과 같은 형식의 ESTAB이 형성된다. 
      2. 주요변수
         1. unsigned int CSADDR: control section의 주소를 PROGADDR에 따라 적절한 값으로 담는다.
         2. int cslth: control section의 길이를 담는다.
   5. int ll\_pass2(char \*file1, char \*file2, char \*file3, unsigned char \*\*dump);
      1. 기능 
         1. object 파일들의 실질적인 linking 작업을 수행한다.
         2. 프로젝트1에서 만들었던 가상메모리(1M) dump에 결과를 올린다.
         3. CSADDR는 pass1과 동일한 방식으로 사용된다.
         4. R 레코드를 읽어 Reference number를 지정해준다. 이에 사용되는 것이 char \*\*ref이다. 01에 해당하는 ref는 cs 이름이다.
         5. 각 T레코드를 읽으면서 object code를 dump에 올려준다.
         6. M 레코드를 읽어 ESTAB에서 해당하는 ref가 있으면 해당 address를 구한다. operator + 또는 -에 따라 symbol 값을 해당 location에 맞게 계산해준다. 이렇게 바뀐 값을 다시 dump에 올려준다. ESTAB에 해당 ref가 없을 경우는 에러메시지가 뜨도록 한다.
         7. E 레코드까지 반복한다.
         8. ref를 free시킨다.
         9. 입력된 파일을 전부 읽을 때까지 반복한다.
         10. PASS2가 끝나면 dump에 PROGADDR부터 total length 만큼의 dump에 linking loader가 구현된 것을 확인할 수 있다.
      2. 주요변수
         1. unsigned int CSADDR: control section의 주소를 PROGADDR에 따라 적절한 값으로 담는다.
         2. int cslth: control section의 길이를 담는다.
   6. int is\_bp(unsigned int value)
      1. 기능
         1. run()에서 bp[]에 있는 break point에서 멈춰야 하기 때문에 해당 value값이 bp[]에 존재하는지 알려주는 일을 한다.
         2. 있을 경우 1 return, 아닐 경우 0 return
      2. 주요변수
         1. 없음
   7. int is\_bp2(unsigned int value)
      1. 기능
         1. break point를 지나면 초기화 시켜준다.
      2. 주요변수
         1. 없음
   8. char\* find\_opcode\_3(unsigned int opcode)
      1. 기능
         1. run()에서 사용된다.
         2. opcode가 주어졌을 때 opcode에 해당하는 opcode name 즉 mnemonic을 return해준다. 해당되는 mnemonic이 없을 경우 “ “를 return한다.
      2. 주요변수
         1. unsigned int opcode: 찾고자 하는 opcode
   9. int run(unsigned char \*\*dump, unsigned int \*PC, unsigned int \*L, unsigned int \*A, unsigned int \*X, unsigned int \*B, unsigned int \*S, unsigned int \*T)
      1. 기능
         1. loader 명령어의 수행으로 메모리에 load된 프로그램을 실행한다.
         2. progaddr 명령어로 지정한 주소부터 실행된다.
         3. 실행 결과로 register A, X, L, PC, B, S, T를 출력한다.
         4. break point까지 실행되고 남은 break point가 없으면 프로그램 끝까지 실행한다.
         5. 이번 프로젝트에서는 copy.obj, progaddr = 0일 경우에 한정한다.
         6. PC는 프로그램 시작 주소, L은 프로그램 길이, 나머지는 0으로 초기화 시켜준다.
         7. TD는 CC를 ‘<’로 변경되었다고 가정하고 다음 instruction으로 넘어간다.
         8. RD는 input device로 아무것도 받지 못했다고 가정한다. CC=’=’라 가정
         9. WD는 다음 instruction으로 넘어간다.
         10. 메모리에 load된 object code를 해석하여 명령어를 구한 뒤, 각 명령어에 해당되는 동작을 수행해주며 적절한 값으로 register의 값을 변경한다.
      2. 주요변수
         1. int opcode, ni, xbpe: object 코드를 해석하여 각각의 값을 담는다.
         2. char op[100]: mnemonic을 담음
         3. unsigned int cur\_addr: current address
         4. int CC: condition code(TD값에 따라 변화)
         5. int comp: mnemonic COMP에서 적용되는 변수. 같을 경우 1로 설정. 기본은 0.
9. **전역 변수 정의**
   1. unsigned int PROG\_ADDR: progaddr [address]로 지정되는 address 값
   2. unsigned int bp[]: breakpoint를 담은 unsigned int 배열
   3. int bp\_idx: bp[]의 index
   4. unsigned int RETADR: run에서 사용되는 return address 값
   5. unsigned int LENGTH: run에서 사용되는 length 값
   6. type struct \_estab{…}ESTAB: pass1에서 생성되는 ESTAB
   7. ESTAB \*eshead[]: ESTAB의 head node
   8. char \*\*ref: pass2에서 사용되는 reference number을 담은 2차원 배열
10. **코드 설명**
    1. 20181679.h

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>// DIR entry

#include <sys/stat.h>// stat>

#define ANSI\_COLOR\_GREEN "\x1b[32m"

#define ANSI\_COLOR\_RESET "\x1b[0m"

unsigned int PROG\_ADDR = 0x00;

unsigned int bp[0x1077];

int bp\_idx = 0;

unsigned int RETADR;

unsigned int BUFFER;

unsigned int LENGTH=0x3;

typedef struct \_node{

char cmd\_name[100];

struct \_node \*next;

}NODE;

NODE \*head = NULL;

typedef struct \_opnode{

unsigned int opcode;

char opcode\_name[10];

char code\_format[5];

struct \_opnode \*next;

}OPNODE;

OPNODE \*ophash[20];

typedef struct \_symtab{

char symbol[20];

unsigned int locctr;

int flag;

struct \_symtab \*next;

}SYM;

SYM \*shead[20];

typedef struct \_estab{

char cs[20];

char symbol[20];

unsigned int addr;

int cslth;

int flag;

struct \_estab \*next;

}ESTAB;

ESTAB \*eshead[20];

char \*\*ref;

//dump, edit, fill, opcode mnemonic commands use this function.

//classify words from command.

void cmd\_classifier(char\* command, char\*\* cmd\_arr, int\* idx);

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

//h[elp]

void print\_all\_commands();

//d[ir]

int print\_file();

//hi[story]

//find right command

void for\_history(char\* command, NODE\* tmp);

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

//change to ASCII

void printchar(int val);

//du[mp] [start, end]

//print the dump memory with start address and end address

//last address is used for dump again

void dump\_print(int start, int end, int \*last\_addr, unsigned char \*\*dump);

//f[ill] [start, end, value]

//from start to end, fill dump with the value

void fill\_dump(int start, int end, int value, unsigned char \*\*dump);

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

//make random hash key for opcodes

int hash\_func(int opcode,char\* opcode\_name);

//make hash table with opcodes

void make\_hash\_table(int opcode, char \*opcode\_name, char \*code\_format);

//opcodelist

void print\_hash\_table();

//opcode mnemonic

//find and print mnemonic's opcode

int find\_opcode(char \*opcode\_name);

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

int find\_opcode\_2(char \*opcode\_name);

int find\_sym(char \*op);

void PASS2(int prog\_len, FILE \*fp, char \*f\_name);

int assemble\_file(FILE \*fp, char \*filename);

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

int find\_file(char \*filename);

void read\_file(char \*filename);

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

//for align

void SWAP(int j);

//print the symbol table

void print\_symbol();

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

//find symbol or cs in estab

//return 1 or 0

int find\_sym\_estab(char \*sym);

//return index

int find\_sym\_estab2(char \*sym);

//linking loader PASS1

int ll\_pass1(char \*file1, char \*file2, char \*file3);

//linking loader PASS2

int ll\_pass2(char \*file1, char \*file2, char \*file3, unsigned char \*\*dump);

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

//do value exists in bp?

int is\_bp(unsigned int value);

//erase used bp

int is\_bp2(unsigned int value);

//return mnemonic

char\* find\_opcode\_3(unsigned int opcode);

//run

int run(unsigned char \*\*dump, unsigned int \*PC, unsigned int \*L, unsigned int \*A, unsigned int \*X, unsigned int \*B, unsigned int \*S, unsigned int \*T);

* 1. 20181679.c

#include "20181679.h"

//for h[help]

//print the kinds of the commands exist.

void print\_all\_commands(){

printf("h[elp]\nd[ir]\nq[uit]\nhi[story]\ndu[mp] [start, end]\ne[dit] address, value\nf[ill] start, end, value\nreset\nopcode mnemonic\nopcodelist\nassemble filename\ntype filename\nsymbol\nprogaddr [address]\nloader [object filename1] [object filename2] [...]\nbp [address]\nrun\n");

}

//for d[ir]

//print the files in present directory

//use dirent.h, sys/stat.h

int print\_file(){

DIR \*dp = NULL;//현재 디렉토리 가리킴

struct dirent \*d = NULL;

struct stat buf;

if((dp = opendir(".")) == NULL){

printf("현재 디렉토리를 열 수 없습니다...\n");

return -1;

}

//디렉토리 읽음

while((d = readdir(dp)) != NULL){

lstat(d->d\_name,&buf);//dirent 구조체는 d\_name에 filename 저장되어있음.

if(S\_ISDIR(buf.st\_mode))//directory인지 확인

printf("%s/\t",d->d\_name);

else if((buf.st\_mode&S\_IXUSR)||(buf.st\_mode&S\_IXGRP)||(buf.st\_mode&S\_IXOTH))//실행 권한 확인(user,group,etc)

printf("%s\*\t",d->d\_name);

else if(S\_ISREG(buf.st\_mode))//일반 파일 확인

printf("%s\t",d->d\_name);

}

printf("\n");

closedir(dp);//디렉토리 스트림 닫음

return 0;

}

//store the command in NODE tmp

void for\_history(char\* command, NODE\* tmp){

if(!strcmp(command,"help") || !strcmp(command,"h") || !strcmp(command,"dir") || !strcmp(command,"d") || !strcmp(command,"history") || !strcmp(command,"hi") || !strncmp(command,"dump",4) || !strncmp(command,"du",2) || !strncmp(command,"edit",4) || !strncmp(command,"e",1) || !strncmp(command,"fill",4) || !strncmp(command,"f",1) || !strncmp(command,"opcode ",7) || !strcmp(command, "opcodelist") || !strcmp(command,"reset") || !strncmp(command,"type ",5) || !strncmp(command,"type ",5) || !strncmp(command,"assemble ",9) || !strcmp(command,"symbol") || !strncmp(command,"progaddr ",9) || !strncmp(command,"loader ",7) || !strncmp(command,"bp ",3) || !strcmp(command,"run")) {

strcpy(tmp->cmd\_name,command);

}

else

strcpy(tmp->cmd\_name,"");

}

//change value to ASCII if value is in range(0x20,0x7e)

void printchar(int val){

if(0x20 <= val && val <= 0x7E)

printf("%c",(char)val);

else

printf(".");

}

//print dump memory

void dump\_print(int start, int end,int\* last\_addr,unsigned char\*\* dump){

int i,j;

//du[mp]

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

if(\*last\_addr >= 0xfffff)

start = 0;

else

start = \*last\_addr+1;

end = start+160-1;

if(end>0xfffff)

end = 0xfffff;

}

//du[mp] start

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

end = start+160-1;

if(end>0xfffff)

end = 0xfffff;

}

//du[mp] start, end

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

//start can't bigger than end

if(start>end){

printf("errormsg");

}

if(start%16!=0){

printf("%05X ",start-start%16);

for(i=0;i<start%16\*3;i++)

printf(" ");

}

for (i=start;i<=end;i++){

// 16바이트 단위로 시작 주소 출력

if(i%16==0)

printf("%05X ",i);

// hex 값으로 메모리의 내용 출력

printf("%02X ",(\*dump)[i]);

// 해당 16진수들을 각각 아스키 문자로 출력

if(i%16-15==0) {

printf(";");

if(i==start+(16-start%16)-1){

for(j=0;j<(start%16);j++)

printf(".");

for(j=start;j<=i;j++)

printchar((\*dump)[j]);

}

else{

for(j=i-15;j<=i;j++)

printchar((\*dump)[j]);

}

printf("\n");

}

}

// 마지막 라인이 16바이트 이하일 경우 정렬 유지

if(end%16!=0xf){

for(i=0;i<(16-(end%16+1))\*3;i++)

printf(" "); // 부족한 공간만큼 space로 이동한 후,

printf(";");

for(i=end-(end%16);i<=end;i++)

printchar((\*dump)[i]); // 남은 아스키 문자 값들 출력

for(i=0;i<16-(end%16+1);i++)

printf(".");

printf("\n");

}

}

\*last\_addr = end;

}

//classify the commands by space

void cmd\_classifier(char\* command,char\*\* cmd\_arr,int\* idx){

int i=0;

char \*ptr = strtok(command," ");

while(ptr!=NULL){

cmd\_arr[i] = ptr;

i++;

ptr = strtok(NULL," ");

}

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

if(cmd\_arr[i]!=NULL){

(\*idx)++;

}

else

continue;

}

}

//for fill and reset

void fill\_dump(int start, int end, int value, unsigned char\*\* dump){

for(int i=start;i<=end;i++)

(\*dump)[i] = value;

}

//hash\_func

//make hash key

int hash\_func(int opcode,char\* opcode\_name){

int re = (strlen(opcode\_name)-opcode)%20;

return re;

}

//make hash table with opcode

//opnode has information of opcode, opcode name and code format

void make\_hash\_table(int opcode, char \*opcode\_name, char \*code\_format){

OPNODE\* op = malloc(sizeof(OPNODE));

op->opcode = opcode;

strcpy(op->opcode\_name,opcode\_name);

strcpy(op->code\_format,code\_format);

int key = hash\_func(opcode,opcode\_name);

if(ophash[key] == NULL)

ophash[key] = op;

else{

op->next = ophash[key];

ophash[key] = op;

}

}

//print the hash table

void print\_hash\_table(){

OPNODE\* tmp = malloc(sizeof(OPNODE));

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

printf("%d : ",i);

tmp = ophash[i];

while(tmp!=NULL){

if(tmp->next==NULL)

printf("[%s,%X]",tmp->opcode\_name,tmp->opcode);

else

printf("[%s,%X]->",tmp->opcode\_name,tmp->opcode);

tmp = tmp->next;

}

printf("\n");

}

}

//for opcode mnemonic

//read hash table to find opcode

int find\_opcode(char\* opcode\_name){

OPNODE\* tmp = malloc(sizeof(OPNODE));

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

tmp = ophash[i];

while(tmp!=NULL){

if(!strcmp(tmp->opcode\_name,opcode\_name)){

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

return 1;

}

tmp = tmp->next;

}

}

return 0;

}

//for assemble pass1 algorithm

//read hash table to find opcode

//해당 mnemonic을 optab에서 찾는다.

//opcode를 반환, 없을 경우 -1 반환

int find\_opcode\_2(char\* opcode\_name){

OPNODE\* tmp = malloc(sizeof(OPNODE));

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

tmp = ophash[i];

while(tmp!=NULL){

if(!strcmp(tmp->opcode\_name,opcode\_name)){

return tmp->opcode;

}

tmp = tmp->next;

}

}

return -1;

}

//run()

char\* find\_opcode\_3(unsigned int opcode){

OPNODE\* tmp = malloc(sizeof(OPNODE));

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

tmp = ophash[i];

while(tmp!=NULL){

if(tmp->opcode==opcode){

return tmp->opcode\_name;

}

tmp = tmp->next;

}

}

return " ";

}

int find\_file(char \*filename){

DIR \*dp = NULL;//현재 디렉토리 가리킴

struct dirent \*d = NULL;

struct stat buf;

if((dp = opendir(".")) == NULL){

printf("현재 디렉토리를 열 수 없습니다...\n");

return -1;

}

//디렉토리 읽음

while((d = readdir(dp)) != NULL){

lstat(d->d\_name,&buf);

if(!strcmp(filename,d->d\_name) && !S\_ISDIR(buf.st\_mode)){

return 1;

}

}

closedir(dp);//디렉토리 스트림 닫음

return 0;

}

//read and print file

void read\_file(char \*filename){

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

char ch;

while((ch = fgetc(fp))!=EOF){

putchar(ch);

}

fclose(fp);

}

int assemble\_file(FILE\* fp,char \*filename){

char \*f\_name = (char\*)malloc(sizeof(char)\*strlen(filename));

strcpy(f\_name,filename);

FILE \*inter = fopen("inter.txt","w");

int line\_num = 5, locctr=0, start\_addr=0, prog\_len=0;

int idx=0, format=3;

char str[100]={0,};

char label[20]={0,}, mnemonic[100]={0,};

char op[100]={0,}, op2[100]={0,}, tmp[100]={0,};

/\*make SYMTAB\*/

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

SYM \*tmp;

SYM \*cur = (SYM\*)malloc(sizeof(SYM));

cur = shead[i];

while(cur!=NULL){

tmp = cur->next;

free(cur);

cur = tmp;

}

shead[i] = NULL;

}

/\*PASS1\*/

do{

/\*read line by line\*/

/\*get label, mnemonic, op(, op2)\*/

fgets(str,80,fp);

if(sscanf(str,"%[^,],%[^\n]",tmp,op2)==1)

strcpy(op2," ");

idx = sscanf(str,"%s %s %s", label, mnemonic, op);

if(idx==2){

strcpy(op,mnemonic);

strcpy(mnemonic,label);

strcpy(label," ");

}

else if(idx==1){

if(!strcmp(label,"RSUB")){

strcpy(mnemonic,label);

strcpy(label," ");

strcpy(op," ");

}

else{

strcpy(mnemonic," ");

strcpy(op," ");

}

}

else if(idx==3){

if(!strcmp(label,".")){

strcpy(mnemonic," ");

strcpy(op," ");

}

if(strcmp(op2," ")){

strcpy(op,mnemonic);

sscanf(op,"%[^,], %s",op,op2);

strcpy(mnemonic,label);

strcpy(label," ");

}

}

/\*make symbol table\*/

//if there is symbol in label field then

if(strcmp(label," ") && strcmp(label,".") && strcmp(mnemonic,"START")){

//search symtab for label

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

if(shead[i]==NULL){

SYM \*temp = (SYM\*)malloc(sizeof(SYM));

strcpy(temp->symbol,label);

temp->locctr = locctr;

temp->flag = 0;

temp->next = NULL;

shead[i] = temp;

break;

}

//error flag(duplicate symbol)

else{

if(!strcmp(shead[i]->symbol,label)){

printf("ERROR: symbol already exists. | %d\n",line\_num);

break;

}

else

continue;

}

}//for

}//symtab

fprintf(inter,"%04x|",locctr);

/\*if opcode == start, set start addr(locctr)\*/

char \*ptr; format = 3;

if(!strcmp(mnemonic,"START")){

start\_addr = strtol(op,&ptr,16);

locctr = start\_addr;

}

else{

if(find\_opcode\_2(mnemonic)!=-1){

if(!strcmp(mnemonic,"CLEAR") || !strcmp(mnemonic,"COMPR") || !strcmp(mnemonic,"TIXR")){

locctr += 0x0002;

format = 2;

}

else

locctr += 0x0003;

}

else{

if(!strcmp(mnemonic,"WORD")){

locctr += 0x0003;

format = 5;

}

else if(!strcmp(mnemonic,"RESW")){

locctr += 0x0003\*atoi(op);

format = 5;

}

else if(!strcmp(mnemonic,"RESB")){

locctr += 0x0001\*atoi(op);

format = 5;

}

else if(!strcmp(mnemonic,"BYTE")){

format = 5;

if(!strncmp(op,"C",1))

locctr += 0x0001\*(strlen(op)-3);

else if(!strncmp(op,"X",1))

locctr += 0x0001\*((strlen(op)-3)/2);

}

else if(!strncmp(mnemonic,"+",1)){

locctr += 0x0004;

format = 4;

}

else if(!strcmp(label,".") || !strcmp(mnemonic,"BASE") || !strcmp(mnemonic,"END")){

}

else

printf("ERROR: invalid operation code | %d\n",line\_num);

}

}//else\_full

fprintf(inter,"%d|%d|%04X|%s|%s|%s|%s\n",line\_num,format,locctr,label,mnemonic,op,op2);

line\_num += 5;

}while(strcmp(mnemonic,"END"));

prog\_len = locctr - start\_addr;

fclose(inter);

PASS2(prog\_len, fp, f\_name);

return 0;

}

//for PASS2

//find label in symtab

int find\_sym(char \*op){

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

if(shead[i]!=NULL){

if(!strcmp(shead[i]->symbol,op)){

return shead[i]->locctr;

}

else

continue;

}

else

break;

}

return -1;

}

//PASS2 algorithm

void PASS2(int prog\_len,FILE\* fp,char \*f\_name){

char \*f\_name2 = (char\*)malloc(sizeof(char)\*strlen(f\_name));

strcpy(f\_name2,f\_name);

FILE \*lst = fopen(strcat(f\_name,".lst"),"w");

FILE \*obj = fopen(strcat(f\_name2,".obj"),"w");

FILE \*temp = fopen("temp.txt","w");

FILE \*temp2 = fopen("temp\_modi.txt","w");

free(f\_name2);

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

int line\_num=0, locctr=0, format=3, locctr\_real=0;

int opcode=0, xbpe=2;

int disp=0;

char label[20]={0,}, mnemonic[100]={0,};

char op[100]={0,}, op2[100]={0,}, str[100]={0,};

char base\_op[100]; int base\_loc=0;

char buf[30], obj\_code[256]={0,};

int r1=0, r2=0, start\_addr=0;

fgets(str,100,inter);

sscanf(str,"%04X|%d|%d|%04X|%[^|]|%[^|]|%[^|]|%[^\n]\n",&locctr\_real,&line\_num,&format,&locctr,label,mnemonic,op,op2);

if(!strcmp(mnemonic,"START")){

start\_addr = locctr\_real;

fprintf(temp,"%d",start\_addr);

fprintf(obj,"H%-6s%06X%06X\n",label,locctr\_real,prog\_len);

fprintf(lst,"%-6d %04X\t%-6s %-6s %-8s %-2s\n",line\_num,locctr\_real,label,mnemonic,op,op2);

}

do{

fgets(str,100,inter);

sscanf(str,"%04X|%d|%d|%04X|%[^|]|%[^|]|%[^|]|%[^\n]\n",&locctr\_real,&line\_num,&format,&locctr,label,mnemonic,op,op2);

//initialize xbpe

xbpe = 2;

if(format == 4){

strcpy(mnemonic,strtok(mnemonic,"+"));

xbpe |= 1;

}

//base\_location counter

if(!strcmp(label,base\_op)){

base\_loc = locctr-0x0003\*atoi(op);

}

//optab에 존재

if((opcode = find\_opcode\_2(mnemonic))!=-1){

/\*ni==11

simple addressing\*/

opcode |= 2;

opcode |= 1;

//operand가 존재

if(strcmp(op," ")){

//symtab에 없을 경우

if((disp = find\_sym(op)) == -1){

if(!strncmp(op,"#",1)){

/\*ni==01

immediate addressing\*/

opcode ^= 2;

char\* ptr;

char tmp[100];

strcpy(tmp,op);

strcpy(op,strtok(op,"#"));

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

disp = strtol(op,&ptr,10);

xbpe ^= 2;

}

else{

disp = find\_sym(op) - locctr;

if(disp<0){

disp &= 0x00000FFF;

}

}

strcpy(op,tmp);

}

else if(!strncmp(op,"@",1)){

/\*ni==10

indirect addressing\*/

char tmp[100];

strcpy(tmp,op);

opcode ^= 1;

strcpy(op,strtok(op,"@"));

disp = find\_sym(op) - locctr;

strcpy(op,tmp);

}

else if(format == 2){

/\*ni==00\*/

opcode ^= 1;

opcode ^= 2;

if(!strcmp(op,"X"))

r1 = 1;

else if(!strcmp(op,"A"))

r1 = 0;

else if(!strcmp(op,"S"))

r1 = 4;

else if(!strcmp(op,"T"))

r1 = 5;

if(!strcmp(op2," X"))

r2 = 1;

else if(!strcmp(op2," A"))

r2 = 0;

else if(!strcmp(op2," S"))

r2 = 4;

}

else{

printf("ERROR: there is no such variable | %d\n",line\_num);

disp = 0x0000;

}

}

//symtab에 존재

else{

if(format == 4){

xbpe ^= 2;

}

else{

if(locctr - disp >= 0x1000){

xbpe |= 4;

xbpe ^= 2;

if(!strcmp(op,"X") || !strcmp(op2," X")){

xbpe |= 8;

}

disp -= base\_loc;

}

else{

disp -= locctr;

if(disp<0){

disp &= 0x00000FFF;

}

}

}

}

}

//RSUB " "

else{

disp = 0x0000;

xbpe = 0x0000;

}

/\*출력 관련\*/

if(format == 4){

char tmp[100] = "+";

strcat(tmp,mnemonic);

strcpy(mnemonic,tmp);

}

if(strcmp(op2," ")){

strcat(op,",");

}

fprintf(lst,"%-6d %04X\t%-6s %-6s %-8s %-2s ",line\_num,locctr\_real,label,mnemonic,op,op2);

if(format == 2){

fprintf(lst," %02X%01X%01X\n",opcode,r1,r2);

sprintf(obj\_code+strlen(obj\_code),"%02X%01X%01X",opcode,r1,r2);

r1=0; r2=0;

}

else if(format == 3){

fprintf(lst," %02X%01X%03X\n",opcode,xbpe,disp);

sprintf(obj\_code+strlen(obj\_code),"%02X%01X%03X",opcode,xbpe,disp);

}

else if(format == 4){

fprintf(lst," %02X%01X%05X\n",opcode,xbpe,disp);

sprintf(obj\_code+strlen(obj\_code),"%02X%01X%05X",opcode,xbpe,disp);

if(strncmp(op,"#",1) && strncmp(op,"@",1))

fprintf(temp2,"%04X\n",locctr\_real);

}

}

//optab에 없을 경우

else{

//그 값을 목적코드로 변환

if(!strcmp(mnemonic,"BYTE")){

if(!strncmp(op,"C",1)){

char\* str;

char tmp[100];

strcpy(tmp,op);

str = strtok(op,"'");

for(int i=0;i<1;i++)

str = strtok(NULL,"'");

sprintf(buf,"%02X%02X%02X",(int)str[0],(int)str[1],(int)str[2]);

strcpy(op,tmp);

}

else if(!strncmp(op,"X",1)){

char\* str;

char tmp[100];

strcpy(tmp,op);

str = strtok(op,"'");

for(int i=0;i<1;i++)

str = strtok(NULL,"'");

sprintf(buf,"%s",str);

strcpy(op,tmp);

}

sprintf(obj\_code+strlen(obj\_code),"%-s",buf);

}

//BASE

else if(!strcmp(mnemonic,"BASE")){

strcpy(base\_op,op);

sprintf(buf," ");

}

/\*출력관련\*/

if(!strcmp(mnemonic,"BASE") || !strcmp(label,".") || !strcmp(mnemonic,"END"))

fprintf(lst,"%-6d %4s\t%-6s %-6s %-8s %-2s ",line\_num," ",label,mnemonic,op,op2);

else

fprintf(lst,"%-6d %04X\t%-6s %-6s %-8s %-2s ",line\_num,locctr\_real,label,mnemonic,op,op2);

fprintf(lst," %-6s\n",buf);

strcpy(buf," ");

}

if(((int)strlen(obj\_code)/2)>28){

if(strlen(obj\_code)!=0){

fprintf(temp," %s\n%d",obj\_code,locctr);

}

memset(obj\_code,'\0',sizeof(obj\_code));

}

else{

if(!strcmp(mnemonic,"RESW") || !strcmp(mnemonic,"END")){

fprintf(temp," %s\n%d",obj\_code,locctr);

memset(obj\_code,'\0',sizeof(obj\_code));

}

}

}while(strcmp(mnemonic,"END"));

fclose(temp);

fclose(temp2);

fclose(inter);

fclose(lst);

temp = fopen("temp.txt","r");

while(fgets(str,100,temp)!=NULL){

sscanf(str,"%d %s\n",&locctr,obj\_code);

if(strcmp(obj\_code,""))

fprintf(obj,"T%06X%02X%s\n",locctr,(int)strlen(obj\_code)/2,obj\_code);

memset(obj\_code,'\0',sizeof(obj\_code));

}

fclose(temp);

temp2 = fopen("temp\_modi.txt","r");

while(fscanf(temp2,"%x\n",&locctr)!=EOF){

fprintf(obj,"M%06X%02X\n",locctr+1,0x05);

}

fclose(temp2);

fprintf(obj,"E%06X\n",start\_addr);

fclose(obj);

remove("temp.txt");

remove("temp\_modi.txt");

remove("inter.txt");

fclose(fp);

}

//for print symbol

//alphabet align

void SWAP(int j){

char temp[20]={0,}; int tmp;

strcpy(temp,shead[j+1]->symbol);

strcpy(shead[j+1]->symbol,shead[j]->symbol);

strcpy(shead[j]->symbol,temp);

tmp = shead[j+1]->locctr;

shead[j+1]->locctr = shead[j]->locctr;

shead[j]->locctr = tmp;

}

//print symbol table

void print\_symbol(){

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

for(int j=0;j<19;j++){

if(shead[j]!=NULL && shead[j+1]!=NULL){

if((shead[j]->symbol)[0] > (shead[j+1]->symbol)[0])

SWAP(j);

else if((shead[j]->symbol)[0] == (shead[j+1]->symbol)[0])

if((shead[j]->symbol)[1] > (shead[j+1]->symbol)[1])

SWAP(j);

}

else

continue;

}

}

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

if(shead[i]!=NULL){

printf("\t%-8s%04X\n",shead[i]->symbol,shead[i]->locctr);

}

else

continue;

}

}

//find symbol or cs in estab

int find\_sym\_estab(char \*sym){

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

if(eshead[i]!=NULL){

if(!strcmp(eshead[i]->symbol,sym)||!strcmp(eshead[i]->cs,sym)){

return 1;

}

else

continue;

}

else

break;

}

return 0;

}

//find symbol or cs in estab for pass2

int find\_sym\_estab2(char \*sym){

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

if(eshead[i]!=NULL){

if(!strcmp(eshead[i]->symbol,sym)||!strcmp(eshead[i]->cs,sym)){

return i;

}

else

continue;

}

else

break;

}

return -1;

}

//linking loader PASS1

int ll\_pass1(char \*file1, char \*file2, char \*file3){

int j=0, cslth=0;

unsigned int CSADDR = PROG\_ADDR;

char \*filename[3] = {file1,file2,file3};

/\*make ESTAB\*/

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

eshead[k] = (ESTAB\*)malloc(sizeof(ESTAB));

eshead[k]->next = NULL;

eshead[k]->flag = 0;

strcpy(eshead[k]->cs,"");

strcpy(eshead[k]->symbol,"");

}

for(int i=0;(i<3 && strcmp(filename[i],""));i++){

FILE \*fp = fopen(filename[i],"r");

char str[100];

//PASS1

do{

fgets(str,sizeof(str),fp);

if(!strncmp(str,"H",1)){

char cs[20];

strcpy(str,strtok(str,"H"));

strcpy(cs,strtok(str," "));

strcpy(str,strtok(NULL,"\n"));

if(find\_sym\_estab(cs)==1)

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

else{

while(eshead[j]->flag!=0)

j++;

strcpy(eshead[j]->cs,cs);

eshead[j]->addr = CSADDR;

cslth = strtol(str,NULL,16);

eshead[j]->cslth = cslth;

eshead[j]->flag = 1;

}

}

do{

fgets(str,sizeof(str),fp);

if(!strncmp(str,"D",1)){

char symbol[20];

unsigned int addr=0x00;

int start=1;

while(sscanf(str+start,"%6s%6X",symbol,&addr) != EOF) {

start += 12;

if(find\_sym\_estab(symbol)==1)

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

else{

while(eshead[j]->flag!=0)

j++;

strcpy(eshead[j]->symbol,symbol);

eshead[j]->addr = CSADDR+addr;

eshead[j]->flag = 1;

}

}//end {while}

}//end {if}

}while(strncmp(str,"E",1));// end {while{} 'E'}

}while(!feof(fp));

CSADDR += cslth;

fclose(fp);

}

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

printf("section name\n");

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

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

if(eshead[i]->flag!=0){

if(strcmp(eshead[i]->cs,""))

printf("%-6s %6s %04X %04X \n",eshead[i]->cs,eshead[i]->symbol,eshead[i]->addr,eshead[i]->cslth);

else

printf("%-6s %6s %04X \n",eshead[i]->cs,eshead[i]->symbol,eshead[i]->addr);

}

else

continue;

}

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

printf(" total length %04X \n",CSADDR-PROG\_ADDR);

return 0;

}

//linking loader PASS2

int ll\_pass2(char \*file1, char \*file2, char \*file3, unsigned char \*\*dump){

unsigned int CSADDR = PROG\_ADDR;

int cslth=0;

char \*filename[3] = {file1,file2,file3};

for(int i=0;(i<3 && strcmp(filename[i],""));i++){

FILE \*fp = fopen(filename[i],"r");

char str[100];

ref = (char\*\*)malloc(sizeof(char\*)\*10);

for(int j=0;j<10;j++)

ref[j] = (char\*)calloc(sizeof(char),5);

//PASS2

do{

fgets(str,sizeof(str),fp);

if(!strncmp(str,"H",1)){

char cs[20];

strcpy(str,strtok(str,"H"));

strcpy(cs,strtok(str," "));

strcpy(str,strtok(NULL,"\n"));

cslth = strtol(str,NULL,16);

strcpy(ref[1],cs);

}

do{

fgets(str,sizeof(str),fp);

if(!strncmp(str,"R",1)){

unsigned int num=0x00;

int start=1;

char symbol[20];

strcpy(symbol,"");

while(sscanf(str+start,"%2X%6s",&num,symbol) != EOF) {

start += 8;

if(1<num && num<10)

strcpy(ref[num],symbol);

}//end {while}

}//end {R}

else if(!strncmp(str,"T",1)){

unsigned int addr=0x00;

int tlth=0,k=0;

char obj[100];

sscanf(str+1,"%6X%2X%[^\n]",&addr,&tlth,obj);

for(int j=0;j<tlth\*2;j+=2){

char tmp[2];

sprintf(tmp,"%c%c",obj[j],obj[j+1]);

(\*dump)[CSADDR+addr+k] = strtol(tmp,NULL,16);

k++;

}

}//end {T}

else if(!strncmp(str,"M",1)){

unsigned int addr = 0x00;

int mlth,num;

char op;

sscanf(str+1,"%6X%2X%c%02X",&addr,&mlth,&op,&num);

if(find\_sym\_estab(ref[num])==1){

int idx = find\_sym\_estab2(ref[num]);

int tmp = eshead[idx]->addr;

char temp[6]; int value;

if(mlth%2==1)

mlth++;

sprintf(temp,"%02X%02X%02X",(\*dump)[CSADDR+addr+(mlth/2)-3],(\*dump)[CSADDR+addr+(mlth/2)-2],(\*dump)[CSADDR+addr+(mlth/2)-1]);

value = strtol(temp,NULL,16);

if(op=='+')

value += tmp;

else

value -= tmp;

(\*dump)[CSADDR+addr+(mlth/2)-3] = (value&0xFF0000)>>16;

(\*dump)[CSADDR+addr+(mlth/2)-2] = (value&0xFF00)>>8;

(\*dump)[CSADDR+addr+(mlth/2)-1] = value&0xFF;

}

else{

int k=0;

while(eshead[k]->flag!=0)

k++;

strcpy(eshead[k]->symbol,ref[num]);

eshead[k]->addr = CSADDR+addr+(mlth/2);

eshead[k]->flag = 1;

}

}//end {M}

}while(strncmp(str,"E",1));// end {while{} 'E'}

if(!strncmp(str,"E",1)){

unsigned int addr = 0x00;

sscanf(str+1,"%6X",&addr);

}

}while(!feof(fp));

CSADDR += cslth;

for(int j=0;j<10;j++)

free(ref[j]);

free(ref);

fclose(fp);

}

return 0;

}

int is\_bp(unsigned int value){

for(int i=0;i<0x1077;i++){

if(bp[i]==value){

return 1;

}

else

continue;

}

return 0;

}

int is\_bp2(unsigned int value){

for(int i=0;i<0x1077;i++){

if(bp[i]==value){

bp[i] = 0x1078;

return 1;

}

else

continue;

}

return 0;

}

int run(unsigned char \*\*dump, unsigned int \*PC, unsigned int \*L, unsigned int \*A, unsigned int \*X, unsigned int \*B, unsigned int \*S, unsigned int \*T){

int opcode, ni, xbpe;

char op[100];

unsigned int cur\_addr = \*PC;

int CC = 0,comp=0;

do{

cur\_addr = \*PC;

opcode = (\*dump)[cur\_addr] & 252;

strcpy(op,find\_opcode\_3(opcode));

ni = (\*dump)[cur\_addr] & 3;

xbpe = ((\*dump)[cur\_addr+1] & 0xF0)>>4;

if(!strcmp(op,"CLEAR") || !strcmp(op,"TIXR") || !strcmp(op,"COMPR"))

\*PC += 2;

else if(!strcmp(op,"JSUB"))

\*PC += 4;

else

\*PC += 3;

if(!strncmp(op,"ST",2)){

if(op[2]=='L')

RETADR = \*L;

else if(op[2]=='C')

BUFFER = \*X;

else if(op[2]=='X')

LENGTH = \*X;

else if(op[2]=='A'){

if(cur\_addr == 0x001D)

BUFFER = \*A;

else if(cur\_addr == 0x23)

LENGTH = \*A;

}

}

else if(!strncmp(op,"LD",2)){

if(op[2]=='B'){

\*B = (\*dump)[cur\_addr+2]+(((\*dump)[cur\_addr+1]&15)<<8)+(\*PC);

}

else if(op[2]=='A'){

if(ni==1)

\*A = (\*dump)[cur\_addr+2]+(((\*dump)[cur\_addr+1]&15)<<8);

else

\*A = (\*dump)[cur\_addr+2]+(((\*dump)[cur\_addr+1]&15)<<8)+(\*PC);

}

else if(op[2]=='T'){

if(cur\_addr == 0x103C){

(\*PC)++;

\*T = (\*dump)[cur\_addr+3]+(((\*dump)[cur\_addr+2])<<8)+(((\*dump)[cur\_addr+1]&15)<<16);

}

else if((xbpe&4) == 4){

\*T = LENGTH;

}

else

\*T = (\*dump)[cur\_addr+2]+(((\*dump)[cur\_addr+1]&15)<<8)+(\*PC);

}

else if(op[2]=='C'){

\*A = ((\*dump)[BUFFER++]);

}

}

else if(!strcmp(op,"JSUB")){

\*L = (\*PC);

\*PC = (\*dump)[cur\_addr+3]+(((\*dump)[cur\_addr+2])<<8)+(((\*dump)[cur\_addr+1]&15)<<16);

cur\_addr = \*PC;

}

else if(!strcmp(op,"CLEAR")){

int tmp = (\*dump)[cur\_addr+1];

if(tmp == 0x10)

\*X = 0;

else if(tmp == 0x00)

\*A = 0;

else if(tmp == 0x40)

\*S = 0;

}

else if(!strcmp(op,"TIXR")){

if(\*X != \*T)

(\*X)++;

}

else if(!strcmp(op,"RSUB")){

\*PC = \*L;

}

else if(!strncmp(op,"J",1)){

if(ni==2){

\*PC = RETADR;

break;

}

else{

if(op[1]=='L'){

if(CC == 1 && (\*T > \*X))

\*PC = (\*dump)[cur\_addr+2]+(((\*dump)[cur\_addr+1]&15)<<8)+(\*PC)-0x1000;

}

else if(op[1]=='E'){

if(CC == 0 && comp==1)

\*PC = (\*dump)[cur\_addr+2]+(((\*dump)[cur\_addr+1]&15)<<8)+(\*PC);

}

else if(op[1]=='\0'){

\*L = \*PC;

\*PC = (\*dump)[cur\_addr+2]+(((\*dump)[cur\_addr+1]&15)<<8)+(\*PC);

}

}

}

else if(!strcmp(op,"RD")){

LENGTH = 0;

CC = 0;//'='

}

else if(!strcmp(op,"TD")){

CC = 1;//'<'

}

else if(!strncmp(op,"COMP",4)){

if(op[4]=='R'&&(\*A==\*S)){

comp = 1; CC = 0;

}

else if(\*A == (\*dump)[cur\_addr+2]+(((\*dump)[cur\_addr+1]&15)<<8)+(\*PC))

comp=1;

}

}while((is\_bp(\*PC)!=1)&&(\*PC != 0x1077)&&(cur\_addr<0x1077));

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

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

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

printf("T : %06X\n",\*T);

if(is\_bp2(\*PC)==1)

printf("\tStop at checkpoint[%X]\n",(\*PC));

else

printf("\tEnd Program\n");

return 0;

}

//main function

int main (){

char command[100];

unsigned char\* dump = (unsigned char\*)calloc(sizeof(unsigned char),0xfffff);

int last\_addr = -1;

NODE \*head = malloc(sizeof(NODE));

NODE \*tmp = malloc(sizeof(NODE));

head->next = tmp;

strcpy(tmp->cmd\_name,"");

//read opcode.txt and make hash table

int opcode = 0;

char opcode\_name[10] = {0,};

char code\_format[5] = {0,};

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

while(fscanf(fp,"%x %s %s",&opcode,opcode\_name,code\_format)==3)

make\_hash\_table(opcode,opcode\_name,code\_format);

//bp initialize

for(int i=0;i<0x1077;i++)

bp[i]= 0x100000;

//run

unsigned int PC = 0;

unsigned int L = 0x1077;

unsigned int A=0, X=0, B=0, S=0, T=0;

//반복 명령 시작

while(1){

printf("sicsim> ");

fgets(command,sizeof(command),stdin);

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

//잘못된 명령어를 입력하는 경우, 수행하지 않고 history에 안남음.

//h[elp]

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

print\_all\_commands();

for\_history(command,tmp);

}

//d[ir]

else if(!strcmp(command,"dir") || !strcmp(command,"d")){

print\_file();

for\_history(command,tmp);

}

//hi[story]

else if(!strcmp(command,"history") || !strcmp(command,"hi")){

for\_history(command,tmp);

NODE\* cur = head->next;

int nu = 1;

while(cur != NULL){

if(strcmp(cur->cmd\_name,"")!=0)

printf("%d\t%s\n",nu++,cur->cmd\_name);

cur = cur->next;

}

}

//du[mp] [start, end]

else if(!strncmp(command,"dump",4) || !strncmp(command,"du",2)){

int idx=0;

int dec\_start=-1,dec\_end=-1;

char \*cmd\_arr[3] = {NULL,};

char \*ptr1, \*ptr2;

char \*cmd = (char\*)malloc(sizeof(char)\*strlen(command));

strcpy(cmd,command);

cmd\_classifier(command,cmd\_arr,&idx);

if(!strcmp(cmd\_arr[0],"du") || !strcmp(cmd\_arr[0],"dump")){

//du[mp]

if(idx == 1){

dec\_start = -1; dec\_end = -1;

dump\_print(dec\_start,dec\_end,&last\_addr,&dump);

}

//du[mp] start

else if(idx == 2){

dec\_start = strtol(cmd\_arr[1],&ptr1,16);

dec\_end = -1;

//주소 범위 체크

if(dec\_start>=0x00000 && dec\_start<=0xfffff){

dump\_print(dec\_start,dec\_end,&last\_addr,&dump);

}

else{

printf("ERROR:wrong value!!\n");

strcpy(cmd,"");

}

}

//du[mp] start, end

else if(idx == 3){

if(!strcmp(&cmd\_arr[1][strlen(cmd\_arr[1])-1],",")){

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

dec\_start = strtol(cmd\_arr[1],&ptr1,16);

dec\_end = strtol(cmd\_arr[2],&ptr2,16);

//주소 범위 체크

if(dec\_start>=0x00000 && dec\_start<=0xfffff && dec\_end>=0x00000 && dec\_end<=0xfffff && dec\_start<=dec\_end){

dump\_print(dec\_start,dec\_end,&last\_addr,&dump);

}

else{

printf("ERROR:wrong value!!\n");

strcpy(cmd,"");

}

}

else{

printf("ERROR:wrong value or wrong command!!\n");

strcpy(cmd,"");

}

}

else{

printf("ERROR:wrong command!! | du[mp] [start, end]\n");

strcpy(cmd,"");

}

}

else{

printf("ERROR:wrong command!! | du[mp] [start, end]\n");

strcpy(cmd,"");

}

for\_history(cmd,tmp);

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

cmd\_arr[i] = NULL;

}

//e[dit] address, value

else if(!strncmp(command,"edit",4) || !strncmp(command,"e",1)){

int idx=0;

int dec\_addr=0;

int value=0;

char \*cmd\_arr[3] = {NULL,};

char \*ptr1, \*ptr2;

char \*cmd = (char\*)malloc(sizeof(char)\*strlen(command));

strcpy(cmd,command);

cmd\_classifier(command,cmd\_arr,&idx);

if(idx == 3){

if(!strcmp(&cmd\_arr[1][strlen(cmd\_arr[1])-1],",")){

//erase comma

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

//get value

dec\_addr = strtol(cmd\_arr[1],&ptr1,16);

value = strtol(cmd\_arr[2],&ptr2,16);

//주소 범위, value 범위 체크

if(value>=0x00 && value<=0xff && dec\_addr>=0x00000 && dec\_addr<=0xfffff){

dump[dec\_addr] = value;

}

else{

printf("ERROR:wrong value or wrong address!!\n");

strcpy(cmd,"");

}

}

else{

printf("ERROR:wrong command!! | e[dit] address, value\n");

strcpy(cmd,"");

}

}

else{

printf("ERROR:wrong command!! | e[dit] address, value\n");

strcpy(cmd,"");

}

for\_history(cmd,tmp);

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

cmd\_arr[i] = NULL;

}

//f[ill] start, end, value

else if(!strncmp(command,"fill",4) || !strncmp(command,"f",1)){

int idx=0;

char \*cmd\_arr[4] = {NULL,};

int start\_addr=0,end\_addr=0,value=0;

char \*ptr1, \*ptr2;

char \*cmd = (char\*)malloc(sizeof(char)\*strlen(command));

strcpy(cmd,command);

cmd\_classifier(command,cmd\_arr,&idx);

if(idx == 4){

if(!strcmp(&cmd\_arr[1][strlen(cmd\_arr[1])-1],",") && !strcmp(&cmd\_arr[2][strlen(cmd\_arr[2])-1],",")){

//erase comma

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

cmd\_arr[2][strlen(cmd\_arr[2])-1] = '\0';

//get value

start\_addr = strtol(cmd\_arr[1],&ptr1,16);

end\_addr = strtol(cmd\_arr[2],&ptr1,16);

value = strtol(cmd\_arr[3],&ptr2,16);

//주소 범위, value 범위 체크

if(value>=0x00 && value<=0xff && start\_addr>=0x00000 && start\_addr<=0xfffff && end\_addr>=0x00000 && end\_addr<=0xfffff && start\_addr <= end\_addr){

fill\_dump(start\_addr, end\_addr, value,&dump);

}

else{

printf("ERROR:wrong value or wrong address!!\n");

strcpy(cmd,"");

}

}

else{

printf("ERROR:wrong command!! | f[ill] start, end, value\n");

strcpy(cmd,"");

}

}

else{

printf("ERROR:wrong command!! | f[ill] start, end, value\n");

strcpy(cmd,"");

}

for\_history(cmd,tmp);

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

cmd\_arr[i] = NULL;

}

//reset

else if(!strcmp(command,"reset")){

fill\_dump(0,0xfffff,0,&dump);

for\_history(command,tmp);

}

//opcode mnemonic

else if(!strncmp(command,"opcode ",7)){

int idx=0;

char \*cmd = (char\*)malloc(sizeof(char)\*strlen(command));

strcpy(cmd,command);

char \*cmd\_arr[2] = {NULL,};

cmd\_classifier(command,cmd\_arr,&idx);

if(idx == 2){

//if opcode exists, find\_opcode() return is 1

//else 0

if(find\_opcode(cmd\_arr[1])==0){

printf("ERROR:there is no such mnemonic\n");

strcpy(cmd,"");

}

}

else{

printf("ERROR:wrong command!! | opcode mnemonic \n");

strcpy(cmd,"");

}

for\_history(cmd,tmp);

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

cmd\_arr[i] = NULL;

}

//opcodelist

else if(!strcmp(command,"opcodelist")){

print\_hash\_table();

for\_history(command,tmp);

}

//type filename

else if(!strncmp(command,"type ",4)){

int idx=0;

char \*cmd = (char\*)malloc(sizeof(char)\*strlen(command));

strcpy(cmd,command);

char \*cmd\_arr[2] = {NULL,};

cmd\_classifier(command,cmd\_arr,&idx);

if(idx == 2){

//if file do not exists error message

if(find\_file(cmd\_arr[1])==0){

printf("ERROR:there is no such file\n");

strcpy(cmd,"");

}

else{

read\_file(cmd\_arr[1]);

}

}

else{

printf("ERROR:wrong command!! | type filename\n");

strcpy(cmd,"");

}

for\_history(cmd,tmp);

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

cmd\_arr[i] = NULL;

}

//assemble filename

else if(!strncmp(command,"assemble ",9)){

int idx=0;

char \*cmd = (char\*)malloc(sizeof(char)\*strlen(command));

strcpy(cmd,command);

char \*cmd\_arr[2] = {NULL,};

cmd\_classifier(command,cmd\_arr,&idx);

char \*cmd2 = (char\*)malloc(sizeof(char)\*strlen(cmd\_arr[1]));

strcpy(cmd2,cmd\_arr[1]);

char \*p = strtok(cmd\_arr[1],".");

char \*filename = (char\*)malloc(sizeof(char)\*strlen(p));

char \*filetype = (char\*)malloc(sizeof(char)\*3);

strcpy(filename,p);

p = strtok(NULL," ");

strcpy(filetype,p);

if(idx == 2){

if(strcmp(filetype,"asm")){

printf("ERROR:filetype error\n");

strcpy(cmd,"");

}

else{

FILE \*fp = fopen(cmd2,"r");

if(fp == NULL){

printf("ERROR:there is no such file!\n");

continue;

}

else{

assemble\_file(fp,filename);

printf(ANSI\_COLOR\_GREEN "Successfully" ANSI\_COLOR\_RESET " assemble %s.\n",cmd2);

}

}

}

else{

printf("ERROR:wrong command!! | assemble filename\n");

strcpy(cmd,"");

}

for\_history(cmd,tmp);

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

cmd\_arr[i] = NULL;

}

//symbol

else if(!strcmp(command,"symbol")){

print\_symbol();

for\_history(command,tmp);

}

//progaddr [address]

else if(!strncmp(command,"progaddr ",9)){

int idx=0;

char \*cmd = (char\*)malloc(sizeof(char)\*strlen(command));

strcpy(cmd,command);

idx = sscanf(command,"%\*s %x\n",&PROG\_ADDR);

if(idx != 1){

printf("ERROR:wrong command!! | progaddr [address] \n");

strcpy(cmd,"");

}

else if(PROG\_ADDR>0xfffff){

printf("ERROR:wrong value!!\n");

PROG\_ADDR = 0x00000;

strcpy(cmd,"");

}

for\_history(cmd,tmp);

}

//linker loader

else if(!strncmp(command,"loader ",7)){

char file1[20],file2[20],file3[20];

int idx = sscanf(command,"%\*s %s %s %s",file1,file2,file3);

if(idx > 3){

printf("ERROR:wrong command!! | progaddr [address] \n");

strcpy(command,"");

}

else{

ll\_pass1(file1,file2,file3);

ll\_pass2(file1,file2,file3,&dump);

}

strcpy(file1,"");strcpy(file2,"");strcpy(file3,"");

for\_history(command,tmp);

}

//break point

else if(!strncmp(command,"bp",2)){

if(!strcmp(command,"bp clear")){

for(int i=0;i<0x1077;i++)

bp[i]= 0x100000;

printf("\t["ANSI\_COLOR\_GREEN"ok"ANSI\_COLOR\_RESET"] clear all breakpoints\n");

}

else if(command[3]=='\0'){

printf("\tbreakpoint\n");

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

for(int i=0;i<0x1077;i++){

if(bp[i]!=0x100000)

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

}

}

else{

sscanf(command,"%\*s %x",&bp[bp\_idx]);

if(bp[bp\_idx]>0x1077){

printf("ERROR::wrong value!\n");

bp[bp\_idx] = 0x100000;

strcpy(command,"");

}

else

printf("\t["ANSI\_COLOR\_GREEN "ok"ANSI\_COLOR\_RESET"] create breakpoint %x\n",bp[bp\_idx++]);

}

for\_history(command,tmp);

}

//run

else if(!strcmp(command,"run")){

run(&dump,&PC,&L,&A,&X,&B,&S,&T);

for\_history(command,tmp);

}

//q[uit]

else if(!strcmp(command,"quit") || !strcmp(command,"q")){

break;

}

//else is wrong command

else

printf("ERROR:wrong command!! check again!!\n");

//for\_history\_linkedList

NODE \*new = malloc(sizeof(NODE));

tmp->next = new;

strcmp(new->cmd\_name,"");

new->next = NULL;

tmp = new;

}

return 0;

}