**과목명: 시스템프로그래밍**

**분반 01**

**<<Project #1>>**

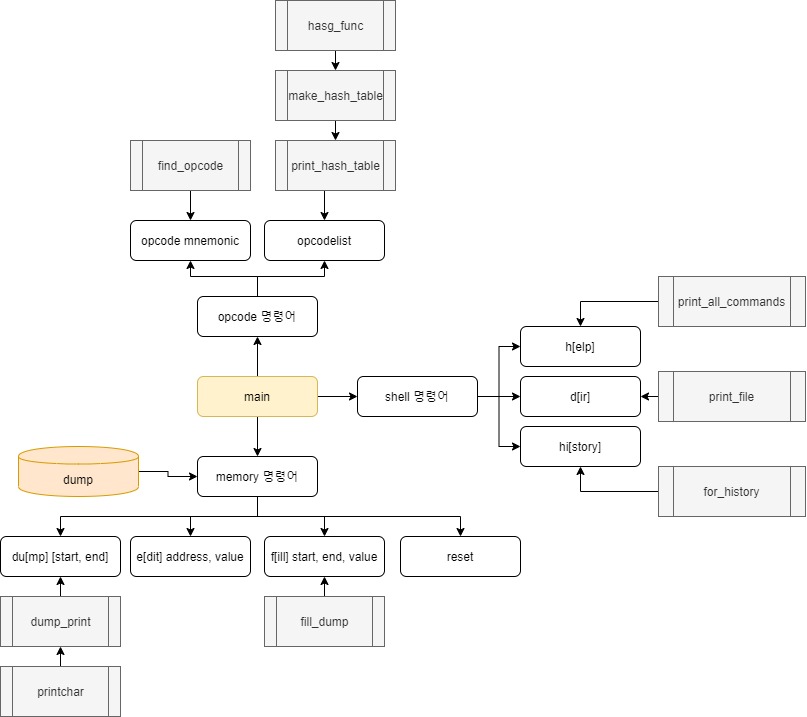
**서강대학교 컴퓨터공학과**

**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. void print\_all\_commands()
      1. 기능
      2. 주요변수
   3. int print\_file()
      1. 기능
      2. 주요변수
   4. void for\_history(char\* command, NODE\* tmp)
      1. 기능
      2. 주요변수
   5. void printchar(int val)
      1. 기능
      2. 주요변수
   6. void dump\_print(int start, int end, int \*last\_addr, unsigned char \*\*dump)
      1. 기능
      2. 주요변수
   7. void fill\_dump(int start, int end, int value, unsigned char \*\*dump)
      1. 기능
      2. 주요변수
   8. int hash\_func(int opcode, char \*opcode\_name)
      1. 기능
      2. 주요변수
   9. void make\_hash\_table(int opcode, char \*opcode\_name, char \*code\_format)
      1. 기능
      2. 주요변수
   10. void print\_hash\_table()
       1. 기능
       2. 주요변수
   11. int find\_opcode(char \*opcode\_name)
       1. 기능
       2. 주요변수
   12. void cmd\_classifier(char\* command, char\*\* cmd\_arr, int\* idx)
       1. 기능
       2. 주요변수
4. **전역 변수 정의**
   1. typedef struct \_node{……}NODE
   2. NODE \*head
   3. typedef struct \_opnode{……}OPNODE
   4. OPNODE \*ophash[20]
5. **코드 설명**
   1. 20181679.h
   2. 20181679.c
6. **프로그램 개요**
   1. 목표
      1. 이 프로그램은 앞으로 구현하게 될 SIC/XE머신을 구현하기 위한 전 단계로서 어셈블러, 링크, 로더들을 실행하게 될 **셸(shell)**과 컴파일을 통해서 만들어진 object코드가 적재되고 실행될 **메모리공간**과 mnemonic을 opcode값으로 변환하는 **OPCODE 테이블과 관련 명령어**들을 구현하는 프로그램이다.
   2. 구현
      1. 셸 (sicsim>)
      2. 셸 관련 명령어들 (help, dir, quit, history)
      3. 메모리공간 (1MB의 메모리를 할당해서 사용)
      4. 메모리공간 관련 명령어들 (dump, edit, fill, reset)
      5. opcode 테이블 (HashTable로 만들어야 한다.)
      6. opcode 관련 명령어들 (opcode, opcodelist)
7. **프로그램 설명**
   1. 프로그램 흐름도



1. **모듈 정의**
   1. main()
      1. 기능
         1. sicsim> 과 같은 unix shell과 유사한 입력 프롬프트 상태가 되도록 한다.
         2. 위와 같은 상태에서 명령어들을 입력 받고, 그에 해당되는 기능을 수행시킨다.
         3. 명령어를 받아 linked list를 구현한다. 정의되지 않은 명령어가 입력되었을 경우를 위한 예외처리를 해준다.
         4. “opcode.txt” 파일을 읽어 hash table을 생성한다.
         5. 명령어는 다음과 같은 기능들이 있다.
            1. h[elp]: shell에서 실행 가능한 모든 명령어들의 리스트를 화면에 출력한다.
            2. d[ir]: shell 명령어 dir을 구현한다. 현재 디렉터리에 있는 파일들을 출력한다.
            3. q[uit]: sicsim을 종료한다.
            4. hi[story]: 현재까지 사용한 명령어들을 순서대로 번호와 함께 출력한다.
            5. du[mp] [start, end]: 할당되어 있는 메모리의 내용을 출력한다.
            6. e[dit] address, value: 메모리의 address번지의 값을 value에 지정된 값으로 바꾼다.
            7. f[ill] start, end, value: 메모리의 start번지부터 end번지까지의 값은 value에 지정된 값으로 변경한다.
            8. reset: dump[]메모리 전체의 내용을 0으로 변경시킨다.
            9. opcode mnemonic: 해당 mnemonic의 opcode값을 출력한다.
            10. opcodelist: opcode Hash Table의 내용을 출력한다.
      2. 주요변수
         1. char command[100]: 명령어를 받는다.
         2. unsigned char\* dump: dump memory(가상의 메모리 공간)를 생성한다.
         3. int last\_addr: 후에 dump에서 쓰일 마지막 번지를 기억한다.
   2. void print\_all\_commands()
      1. 기능
         1. h[help] 명령어를 입력하였을 때, shell에서 실행 가능한 모든 명령어들을 출력한다.
      2. 주요변수
         1. 없음
   3. int print\_file()
      1. 기능
         1. d[ir] 명령어를 입력하였을 때, 현재 디렉터리에 있는 파일들을 출력한다.
         2. dirent.h를 사용하여 파일 이름을 얻을 수 있다.
         3. sys/stat.h를 사용하여 파일 모드를 구분할 수 있다. 실행 파일은 파일 이름 옆에 ‘\*’표시, 디렉터리는 ‘/’표시를 한다.
      2. 주요변수
         1. 없음
   4. void for\_history(char\* command, NODE\* tmp)
      1. 기능
         1. 정의되어 있는 명령어를 받아서 linked list의 node에 저장한다.
      2. 주요변수
         1. char\* command: 명령어
         2. NODE\* tmp: 저장 받을 node 공간
   5. void printchar(int val)
      1. 기능
         1. 범위(0x20~0x7e)에 해당하는 값을 ASCII code로 출력한다.
         2. 범위에 없을 경우 ‘.’으로 출력한다.
      2. 주요변수
         1. int val: ASCII code로 바꿔야하는 값
   6. void dump\_print(int start, int end, int \*last\_addr, unsigned char \*\*dump)
      1. 기능
         1. 할당되어 있는 메모리의 내용을 출력한다.
         2. 가장 왼쪽 칼럼은 출력하는 해당 row의 메모리의 시작 주소를 의미한다. 주소는 5자리로 고정하고 16진수로 출력한다.
         3. 가운데 칼럼은 메모리의 내용을 16 진수 형태로 보여준다. 내용에 들어가는 값의 범위는 0x00부터 0xFF이다.
         4. 가장 오른쪽 칼럼은 3.5를 이용하여 해당되는 메모리 내용을 ASCII code로 변환한 값이다.
         5. 경우는 다음 세가지로 나뉜다. dump/dump start/dump start, end
            1. dump: 기본적으로 10라인을 출력한다. 다시 dump를 실행시키면 이전 du[mp] [start, end] 명령어의 마지막 (address + 1) 번지부터 출력된다. dump 명령어가 처음 시작될 때는 0 번지부터 출력된다. 계속된 dump 출력 시, 범위를 확인하여 주소의 끝(0xFFFFF)까지 출력하고 다시 dump 명령어를 입력하였을 경우 0번지부터 출력한다.
            2. dump start: start 번지부터 10라인을 출력한다. dump와 마찬가지로 범위를 확인하여 주소를 넘어간 경우 주소의 끝(0xFFFFF)까지 출력한다.
            3. dump start, end: start 번지부터 end 번지까지 출력한다. start 주소가 end 주소보다 작은 값이 들어온 경우, 에러가 뜨게 한다. start, end의 주소 범위를 확인하고 에러처리를 한다.
      2. 주요변수
         1. int start/int end: start 번지와 end 번지를 받는다. dump의 경우 start, end 모두 -1, dump start의 경우 end만 -1의 값을 가진다.
         2. int\* last\_addr: 다시 dump를 입력했을 때의 경우를 위해 이전 du[mp] [start, end] 명령어 수행 시 마지막 번지를 저장한다.
         3. unsigned char\*\* dump: 메모리 내용이 담겨져 있는 가상의 dump 메모리 공간이다.
   7. void fill\_dump(int start, int end, int value, unsigned char \*\*dump)
      1. 기능
         1. 메모리의 start 번지부터 end 번지까지의 값을 value에 지정된 값으로 변경한다.
      2. 주요변수
         1. int start/int end: start 번지, end 번지
         2. int value: 메모리를 채울 값
   8. int hash\_func(int opcode, char\* opcode\_name)
      1. 기능
         1. Hash Table에 필요한 hash key를 생성해준다.
      2. 주요변수
         1. 없음
   9. void make\_hash\_table(int opcode, char \*opcode\_name, char \*code\_format)
      1. 기능
         1. opcode, opcode name, code format이 담긴 OPNODE를 생성하고 linked list 형태로 Hash Table을 구현한다.
         2. Hash Table의 사이즈는 20으로 고정되어 있고, mnemonic 수는 훨씬 많으므로 3.8 hash function을 활용하여 mnemonic마다 해당하는 hash key에 따라 저장한다.
      2. 주요변수
         1. OPNODE\* op: opcode의 정보들을 담을 node
         2. int opcode: opcode
         3. char \*opcode\_name: mnemonic
         4. char \*code\_format: 이후에 쓰일 3형식 or 4형식
   10. void print\_hash\_table()
       1. 기능
          1. Hash Table을 출력해준다.
       2. 주요변수
          1. 없음
   11. int find\_opcode(char \*opcode\_name)
       1. 기능
          1. 입력한 mnemonic에 해당하는 opcode를 출력한다.
          2. 해당 mnemonic이 있을 경우 1, 없을 경우 0을 리턴한다.
       2. 주요변수
   12. void cmd\_classifier(char\* command, char\*\* cmd\_arr, int\* idx);
       1. 기능
          1. du[mp] [start, end]/f[ill] start, end, value/e[dit] address, value/opcode mnemonic 의 경우 명령어에서 값을 추출하는 역할을 한다.
       2. 주요변수
          1. char\* command: 명령어
          2. char\*\* cmd\_arr: 추출된 값들
2. **전역 변수 정의**
   1. typedef struct \_node{……}NODE: history에 쓰일 명령어 linked list를 위한 node 구조체
   2. NODE \*head: history에 쓰일 명령어 linked list의 head node
   3. typedef struct \_opnode{……}OPNODE: opcode hash table에 쓰일 명령어 linked list
   4. OPNODE \*ophash[20]: opcode hash table을 hash function에 따라 저장하기 위한 head node
3. **코드 설명**
   1. 20181679.h

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>// DIR entry

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

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];

//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);

* 1. 20181679.c

#include "20181679.h"

#include <ctype.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\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")){

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)++;

}

}

//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;

}

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);

//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){

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);

}

//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;

}