과목 명: 시스템프로그래밍

담당 교수 명: 김 지 환

<<Assignment 1>>

**서강대학교 컴퓨터학과**

**[20161620]**

**[이수연]**

목 차

1. 프로그램 개요 3

2. 프로그램 설명 3

2.1 프로그램 흐름도 3

3. 모듈 정의 3

3.1 모듈 이름 : main() 3

3.1.1 기능 3

3.1.2 사용 변수 3

3.2 모듈 이름: display\_instructions() 3

3.2.1 기능 3

3.2.2 사용 변수 3

3.3 모듈이름: get\_data() 4

3.3.1 기능 4

3.3.2 사용변수 4

3.4 모듈이름: display\_report() 4

3.4.1 기능 4

3.4.2 사용변수 4

3.5 모듈이름: continue\_function() 4

3.5.1 기능 4

3.5.2 사용변수 4

4. 전역 변수 정의 4

4.1 long income[100] 4

4.2 int month[100], day[100], year[100] 4

4.3 int ctr 4

4.4 int cont 5

# 프로그램 개요

앞으로 구현하게 될 SIC/XE머신을 구현하기 전의 단계로서 어셈블러, 링크, 로더 등을 실행하게 될 쉘파일을 만든다. 또한 object코드가 적재될 가상 메모리 공간을 만들고, opcode 처리를 위해 해시테이블을 이용하여 관련 명령어들을 구현한다.

# 프로그램 설명

## 프로그램 흐름도

main

1) 메인에서 명령어를 받는다

arrangeCommand

2) arrangeCommand에서 정상명령어인지

판단하여 정상이면 계속 진행,

비정상이면 다시 돌아가서 입력받는다

insertBack

3) insertBack에서 히스토리에

정상명령어를 넣는다

main

hashFunc

DumpEditFill

4)main, DumpEditFill, hashFunc등의 각 조건에 맞는 명령어 처리 함수를 시행한다.

그림 1> 프로그램 흐름도

# 모듈 정의

## 모듈 이름 : main()

### 기능

사용자를 통해 입력을 받아 arrangeCommand함수로 보낸다. 만약 정상 입력이 아닐 경우 리턴값으로 -1이 돌아오고 이때 continue를 통해 다른 활동을 하지 않고 새로운 입력을 받게끔 루프를 설정한다. 또한 help, dir, opcode, dump, 해시 테이블 작성 등의 비교적 간단한 명령어의 수행은 따로 함수로 분리하지 않고, main의 각 if문 안에서 처리되게끔 하였다.

### 사용 변수

DIR\* dir;//디렉토리

struct dirent\* ent;

struct stat buf;

char home[1024];

int i,j;

int isRightCommand=1;

signed long temp;

FILE \*fp;

unsigned int id;

Node\* pnode = NULL;//출력때 이용할 포인터

Table\* new = NULL;

## 모듈 이름: Create\_Node(char\* data)

### 기능

데이터(data)를 넘겨받아 새로운 노드를 생성하고 데이터 정보를 노드에 넣어주었다. 링크 포인터는 널로 초기화 하였다.

### 사용 변수

Node\* newnode=(Node\*)malloc(sizeof(Node));

## 모듈이름: insertBack(Node\*\* head, Node\*\* tail, Node\* newnode)

### 기능

전역변수로 선언한 헤드노드와 테일노드를 확인하여 새로운 정보를 담고있는 뉴노드를 넘겨받아 연결리스트의 끝에 삽입한다. (히스토리에 쓸 기능이므로 끝에 삽입하면 된다.)

### 사용변수

전역변수 – 헤드노드, 테일노드

## 모듈이름: arrangeCommand(char\* command)

### 기능

메인에서 입력 받은 명령어를 받아 정상적인 명령어인지 판단하고 앞, 뒤의 공백이나 탭을 제거하여 명령어를 정리해준다. 명령어는 전역변수이다. 정상 명령어이면 메인에 1을 반환하고 아니라면 -1을 반환한다. 여러가지 명령어의 예외처리를 여기서 해준다.

### 사용변수

int i,j = 0;

int start\_flag = 0;

int return\_flag1 = 1;

int return\_flag2 = 1;

int comma\_index1=-1;

int comma\_index2=-1;

char store[MAX] = {'\0',};

char store2[MAX] = {'\0',};//dump의 숫자명령어 처리 위해서 따로 저장

char dumpRange[MAX] = {'\0',};

char\* ptr;

## 모듈이름: DumpEditFill(int start, int end, int address, int value)

### 기능

인자를 가진 dump명령어, edit 명령어, fill 명령어의 수행을 처리해준다.

### 사용변수

signed long i,j;

signed long temp;

signed long endTemp;

unsigned int endLinenum;

## 모듈이름: Create\_Table(int key, char\* name)

### 기능

해시테이블에 사용하기 위한 테이블 한블럭을 만들고 정보를 넣어준다.

### 사용변수

Table\* new = (Table\*)malloc(sizeof(Table))

## 모듈이름: hashFunc(char\* mnemonic)

### 기능

해시테이블에 사용하기 위한 해시함수를 구성하였다. 명령어 mnemonic을 받아서 그 아이디 값을 해시 값(MAX\_HASH인 20으로 나눈 값)으로 변환하여 반환했다. 이를 통해 키 값으로 선형시간 안에 정보에 접근할 수 있다.

### 사용변수

unsigned int hashkey=30;

unsigned int key;

char name[MAX];

char format[MAX];

FILE \*fp;

# 전역 변수 정보

typedef struct Table{

int id;

char instruct[MAX];

struct Table\* hashNext;

}Table;

Table\* hashTable[MAX\_HASH];

unsigned int key;

char name[MAX];

char format[MAX];

char mnemonic[MAX];

typedef struct Node{

char data[MAX];

struct Node\* link;

}Node;

Node\* head = NULL;

Node\* tail = NULL;

signed long start = -1;

signed long end = -1;

signed long value = -1;

signed long address = -1; //보통 메모리 주소는 unsigned지만 예외처리를 위해 음수를 써야했음. 또 FFFFF가 최대니까 범위고 괜찮고..

unsigned int line\_num = 0;//가상메모리 해당 라인의 맨 왼쪽 주소

unsigned int dump\_p = 0;//가상메모리 각 주소 가리키는 용도

unsigned char virtual[MAX\_memory]="is sample program;;saoo3rklsnkada sd 20161620 suyeo leald;jdasAD";

char command[MAX];///전체 명령어를 관장하는 전역변수 ;; 밑의 arrangeCommand를 거치면 명령어가 정리됨

char store\_command[MAX];//히스토리 출력용을 위해서 처음 받은 명령어 저장

# 코드

(너무 길어서 작은 글씨로 첨부하였습니다.)

<20161620.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <sys/stat.h>

#define MAX 257

#define STACK\_MAX 1000

#define MAX\_memory 1048600

#define MAX\_HASH 20

#define HASH\_KEY(key) key%MAX\_HASH////해시 함수 키값으로의 변환 매크로

typedef struct Table{

int id;

char instruct[MAX];

struct Table\* hashNext;

}Table;

Table\* hashTable[MAX\_HASH];

unsigned int key;

char name[MAX];

char format[MAX];

char mnemonic[MAX];

typedef struct Node{

char data[MAX];

struct Node\* link;

}Node;

Node\* head = NULL;

Node\* tail = NULL;

signed long start = -1;

signed long end = -1;

signed long value = -1;

signed long address = -1; //보통 메모리 주소는 unsigned지만 예외처리를 위해 음수를 써야했음. 또 FFFFF가 최대니까 범위고 괜찮고..

unsigned int line\_num = 0;//가상메모리 해당 라인의 맨 왼쪽 주소

unsigned int dump\_p = 0;//가상메모리 각 주소 가리키는 용도

unsigned char virtual[MAX\_memory]="is sample program;;saoo3rklsnkada sd 20161620 suyeo leald;jdasAD";

char command[MAX];///전체 명령어를 관장하는 전역변수 ;; 밑의 arrangeCommand를 거치면 명령어가 정리됨

char store\_command[MAX];//히스토리 출력용을 위해서 처음 받은 명령어 저장

Node\* Create\_Node(char\* data);///새로운 노드 생성하고 초기화

void insertBack(Node\*\* head, Node\*\* tail, Node\* newnode);///알맞은 명령어일때 히스토리에 넣어줌

int arrangeCommand(char\* command);//명령어 각종 예외처리를 하고 정리해줌

void DumpEditFill(int start, int end, int address, int value);//dump, edit, fill 명령어 수행 함수

Table\* Create\_Table(int key, char\* name);///해시테이블에 사용할 새로운 노드 생성하고 초기화

int hashFunc(char\* mnemonic);///해시함수

<20161620.c>

#include “20161620.h"//라이브러리,전역변수,매크로

///////////////////함수 주석은 헤더파일에////////////////////////

int main (void)

{

DIR\* dir;//디렉토리

struct dirent\* ent;

struct stat buf;

char home[1024];

int i,j;

int isRightCommand=1;

signed long temp;

FILE \*fp;

unsigned int id;

Node\* pnode = NULL;//출력때 이용할 포인터

Table\* new = NULL;

while(1)

{

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

{

command[i] = '\0';

mnemonic[i] = '\0';

}

printf("\nsicsim> ");

fgets(command,MAX,stdin);//공백까지 읽으려고 fgets 사용

strcpy(store\_command,command);

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

start = end = value = address = -1;

///opcode 파일처리///

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

while(1)

{

if(fscanf(fp,"%X %s %s\n",&key,name,format)==EOF)

break;

else

{

new = Create\_Table(key,name);

if(hashTable[HASH\_KEY(key)] == NULL)

hashTable[HASH\_KEY(key)] = new;

else

{

new->hashNext = hashTable[HASH\_KEY(key)];

hashTable[HASH\_KEY(key)] = new;

}

}

}

fclose(fp);//파일 읽어서 테이블 만들기

isRightCommand = arrangeCommand(command);

if(isRightCommand == -1)

{

printf("error : 알맞은 형식의 입력(명령어)이 아닙니다. 다시 입력받으세요.\n");

continue;

}//이상한 문자 들어오면 루프위로가서 다시 시작;;history에도 안넣어주는게 처리된다!!

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

{

insertBack(&head,&tail,Create\_Node(store\_command));

printf("\nh[elp]\nd[ir]\nq[uit]\nhi[story]\n");

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

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

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

printf("reset\nopcode mnemonic\nopcodelist\n\n");

}

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

{

insertBack(&head,&tail,Create\_Node(store\_command));

strncpy(home,getenv("HOME"),sizeof(home));//로그인한 랜덤한 사용자의 홈을 받기 위함

dir = opendir(home);

if(dir != NULL)

{

printf("\n");

while((ent = readdir(dir)) != NULL)

{

stat(ent->d\_name,&buf);

if(S\_ISDIR(buf.st\_mode))

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

else if((buf.st\_mode & S\_IXUSR) != 0)

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

else

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

}

printf("\n");

closedir(dir);

}

else

{

perror("");

return EXIT\_FAILURE;

}

}

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

{

insertBack(&head,&tail,Create\_Node(store\_command));

i=1;

pnode=head;

while(pnode != NULL)

{

printf("%d %s\n",i,pnode->data);

pnode = pnode->link;

i++;

}

}

if(((!strcmp(command,"dump")) || (!strcmp(command,"du"))))

{

insertBack(&head,&tail,Create\_Node(store\_command));

if(line\_num>=1048560)

{

line\_num = 0;

dump\_p = 0;

}//이미 0xFFFFF값 넘었을때 다시 0부터 출력 처리

if((dump\_p % 16) == 0)

{

temp = 0;

for(i=0; i<10; i++)

{

if(line\_num>1048560)

break;

printf("%05X ",line\_num);

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

{

printf("%02X ",virtual[dump\_p]);

dump\_p++;

}

printf("; ");

dump\_p = dump\_p - 16;

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

{

if((virtual[dump\_p]>=0x20) && (virtual[dump\_p]<=0x7E))

printf("%c",virtual[dump\_p]);

else

printf(".");

dump\_p++;

}

printf("\n");

line\_num+=16;

}

}

else

{////dump만 쳤을때 첫번째 번지부터 출력하지 않는 경우도 처리

temp = dump\_p%16;

line\_num = (unsigned int)(dump\_p-temp);

for(i=0; i<10; i++)

{

if(line\_num > 1048560)

break;

printf("%05X ",line\_num);

if(i==0)

{

for(j=0; j<temp; j++)

printf(" ");

for(j=temp; j<16; j++)

{

printf("%02X ",virtual[dump\_p]);

dump\_p++;

}

printf("; ");

dump\_p = dump\_p-(unsigned int)(16-temp);

for(j=0; j<temp; j++)

printf(".");

for(j=temp; j<16; j++)

{

if((virtual[dump\_p]>=0x20) && (virtual[dump\_p]<=0x7E))

printf("%c",virtual[dump\_p]);

else

printf(".");

dump\_p++;

}

printf("\n");

}

else

{

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

{

printf("%02X ",virtual[dump\_p]);

dump\_p++;

}

printf("; ");

dump\_p = dump\_p - 16;

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

{

if((virtual[dump\_p]>=0x20) && (virtual[dump\_p]<=0x7E))

printf("%c",virtual[dump\_p]);

else

printf(".");

dump\_p++;

}

printf("\n");

}

line\_num+=16;

}

if(line\_num <= 1048560 && temp!=0)

{

printf("%05X ",line\_num);

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

{

printf("%02X ",virtual[dump\_p]);

dump\_p++;

}

for(i=temp; i<16; i++)

printf(" ");

printf("; ");

dump\_p = dump\_p - temp;

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

{

if((virtual[dump\_p]>=0x20) && (virtual[dump\_p]<=0x7E))

printf("%c",virtual[dump\_p]);

else

printf(".");

dump\_p++;

}

for(i=temp; i<16; i++)

printf(".");

printf("\n");//160개 출력을 위한 처리

}

}

}

if(start != -1 || end != -1 || value != -1 || address != -1)

{

DumpEditFill(start,end,address,value);//dump , fill, edit

}

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

{

insertBack(&head,&tail,Create\_Node(store\_command));

dump\_p=0;

for(i=0; i<MAX\_memory; i++)

{

virtual[dump\_p] = 0;

dump\_p++;

}

}

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

{

insertBack(&head,&tail,Create\_Node(store\_command));

for(i=0; i<MAX\_HASH; i++)

{

printf("%d : ",i);

if(hashTable[i] != NULL)

{

new = hashTable[i];

while(new->hashNext)

{

printf("[%s,%X]",new->instruct,new->id);

new = new->hashNext;

printf(" -> ");

}

printf("[%s,%X]",new->instruct,new->id);

}

printf("\n");

}

}

if(strlen(mnemonic) != 0)

{

id = hashFunc(mnemonic);

if(id<MAX\_HASH)

{

insertBack(&head,&tail,Create\_Node(store\_command));

if(!strcmp(hashTable[id]->instruct,mnemonic))

printf("opcode is %X\n",hashTable[id]->id);

else

{

new = hashTable[id];

while(new->hashNext)

{

if(!strcmp(new->hashNext->instruct,mnemonic))

printf("opcode is %X\n",new->hashNext->id);

new = new->hashNext;

}

}

}

else

printf("error! : 입력한 Mnemonic은 테이블에 없습니다.\n");

}

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

{

insertBack(&head,&tail,Create\_Node(store\_command));

break;

}

for(i=0; i<MAX\_HASH; i++)

hashTable[i] = NULL;//해시테이블 다시 채우기위해 초기화

}

printf("\n");

return 0;

}

int arrangeCommand(char\* command)

{

int i,j = 0;

int start\_flag = 0;

int return\_flag1 = 1;

int return\_flag2 = 1;

int comma\_index1=-1;

int comma\_index2=-1;

char store[MAX] = {'\0',};

char store2[MAX] = {'\0',};//dump의 숫자명령어 처리 위해서 따로 저장

char dumpRange[MAX] = {'\0',};

char\* ptr;

command[(strlen(command)-1)] = '\0';//fgets로 받았으니까 우선 개행 문자 짜름

for(i=strlen(command)-1;i>=0 ; i--)

{

if(command[i] == ' ' || command[i] == '\t')

command[i]='\0';

else

break;

} //뒤에 공백문자 있는경우 우선처리

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

{

if(command[i] == ' ' || command[i] == '\t')

continue;

else

break;

}

strcpy(command,command+i);//앞에 공백 있는거 자름

strcpy(store2,command);//앞뒤 없고 사이에는 공백 있는것

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

{

if(start\_flag == 0)

{

if((command[i]>='A' && command[i]<='Z') ||(command[i]>='a' && command[i]<='z') || (command[i]>='0' && command[i]<='9') || command[i]==',')

{

if(command[i]==' ' || command[i]=='\t')

continue;

if(command[i]>='A' && command[i]<='Z')

command[i] += 32;

store[j] = command[i];

j++;

start\_flag=1; //영어 대소문자, 숫자나 콤마 입력이 처음 들어옴

}

else

return\_flag1 = -1; //정상 입력이 아님

}

else if(start\_flag == 1)//문자가 시작되면 알파벳 외에 다른 문자가 들어오면 적절하지 않은 명령어 처리;;-1 리턴

{

if(command[i]==' ' || command[i]=='\t')

continue;

if((command[i]>='A' && command[i]<='Z') || (command[i]>='a' && command[i]<='z') || (command[i]>='0' && command[i]<='9') || command[i]==',')

{

if(command[i]>='A' && command[i]<='Z')

command[i] += 32;

store[j]=command[i];

j++;

}

else

return\_flag1 = -1;

}

}

strcpy(command,store);

/////////////////printf("command는 문자,숫자,콤마만 있는 명령어 :%s\n\n",command);

////여기서부터는 dump에서 뒤에 숫자 명령어 나올 경우를 위한 별도 처리/////

if(strstr(command,"dump") || strstr(command,"du"))

{

if(store2[0]=='d' && store2[1]=='u' && store2[2]=='m' && store2[3]=='p')

{

for(i=0; i<strlen(store2); i++)

{

if(store2[i]=='p' || store2[i]=='P')

break;

}

if((store2[i+1] != ' ') && (store2[i+1] != '\t') && store2[i+1] != '\0')

return\_flag2=-1;

else if(store2[i+1] == '\0')

return 1;

}

else if(store2[0]=='d' && store2[1]=='u')

{

for(i=0; i<strlen(store2); i++)

{

if(store2[i]=='u' || store2[i]=='U')

break;

}

if((store2[i+1] != ' ') && (store2[i+1] != '\t') && store2[i+1] != '\0')

return\_flag2=-1;

else if(store2[i+1] == '\0')

return 1;

}

else

{

printf("error : 입력받은 명령어에 대문자가 있거나 명령어 사이에 공백이 있습니다.\n");

return\_flag2=-1;

}/////////이 과정에서 정상 명령어라고 가정했을때 다음에 붙은 다른 문자가 있는지, 없고 공백만 있는지 확인한다

strcpy(dumpRange,store2+i+1);//dump 명령어 이후의 것만 dumpRange에 저장

////////////////printf("dumpRange :%s\n\n",dumpRange);

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

{

if(dumpRange[i] !=' ' && dumpRange[i] != '\t')

{

j=i;

break;

}

}

strcpy(dumpRange,dumpRange+j);//앞에 공백 잘라냄

//////////////////printf("dumpRange1 ; 앞의 공백 잘라낸거 :%s\n\n",dumpRange);

for(i=strlen(dumpRange)-1;i>=0 ; i--)

{

if(dumpRange[i] == ' ' || dumpRange[i] == '\t')

dumpRange[i]='\0';

else

break;

} //뒤에 공백 잘라냄

//////////////////////printf("dumpRange2 ; 뒤의 공백 잘라낸거 :%s\n\n",dumpRange);

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

{

if(dumpRange[i] == ',')

comma\_index1=i;

}

if(comma\_index1==-1) //콤마가 없으면;; start만 있으면

{

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

{

if((dumpRange[i]>='a' && dumpRange[i]<='f') || (dumpRange[i]>='A' && dumpRange[i]<='F') || (dumpRange[i]>='0' && dumpRange[i]<='9'))

continue;

else

{

return\_flag2=-1;

break;

}

}

if(return\_flag2 != -1)

{

start = strtoul(dumpRange,NULL,16);

return\_flag2 = 1;

}

}

else

{///콤마가 있을 때

start\_flag=0;

for(i=comma\_index1-1; i>=0; i--)

{

if(start\_flag==0 && ((dumpRange[i] != ' ') && (dumpRange[i] != '\t')))

start\_flag=1;

if(start\_flag==1)

{

if((dumpRange[i]>='a' && dumpRange[i]<='f') || (dumpRange[i]>='A' && dumpRange[i]<='F') || (dumpRange[i]>='0' && dumpRange[i]<='9'))

continue;

else

{

return\_flag2=-1;

break;

}

}

}////콤마 앞에서 정상명령어 아닌것 처리

start\_flag=0;

for(i=comma\_index1+1; i<strlen(dumpRange); i++)

{

if(start\_flag==0 && ((dumpRange[i] != ' ') && (dumpRange[i] != '\t')))

start\_flag=1;

if(start\_flag==1)

{

if((dumpRange[i]>='a' && dumpRange[i]<='f') || (dumpRange[i]>='A' && dumpRange[i]<='F') || (dumpRange[i]>='0' && dumpRange[i]<='9'))

continue;

else

{

return\_flag2=-1;

break;

}

}

}////콤마 뒤에서 정상명령어 아닌것 처리

}

if(comma\_index1 != -1 && return\_flag2 != -1)

{

ptr = strtok(dumpRange,",");//콤마를 기준으로 쪼갬

i=1;

while(ptr!=NULL)

{

if(i==1)

start = strtoul(ptr,NULL,16);//문자열을 16진수로 바꾸어서 변수에 저장

else if(i==2)

end = strtoul(ptr,NULL,16);

ptr = strtok(NULL,",");

i++;

}

return\_flag2 = 1;

}

}

///////////////여기서부터 앞의 dump명령어와 비슷하게 처리함///////////////////

if(strstr(command,"edit") || command[0]=='e')

{

if(store2[0]=='e' && store2[1]=='d' && store2[2]=='i' && store2[3]=='t')

{

for(i=0; i<strlen(store2); i++)

{

if(store2[i]=='t' || store2[i]=='T')

break;

}

if((store2[i+1] != ' ') && (store2[i+1] != '\t') && store2[i+1] != '\0')

return\_flag2=-1;

}

else if(store2[0]=='e')

{

for(i=0; i<strlen(store2); i++)

{

if(store2[i]=='e' || store2[i]=='E')

break;

}

if((store2[i+1] != ' ') && (store2[i+1] != '\t') && store2[i+1] != '\0')

return\_flag2=-1;

}

else

{

printf("error : 입력받은 명령어에 대문자가 있거나 명령어 사이에 공백이 있습니다.\n");

return\_flag2=-1;

}

strcpy(dumpRange,store2+i+1);//dump 명령어 이후의 것만 dumpRange에 저장

///////////////////////printf("dumpRange :%s\n\n",dumpRange);

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

{

if(dumpRange[i] !=' ' && dumpRange[i] != '\t')

{

j=i;

break;

}

}

strcpy(dumpRange,dumpRange+j);//앞에 공백 잘라냄

//////////////////printf("dumpRange1 ; 앞의 공백 잘라낸거 :%s\n\n",dumpRange);

for(i=strlen(dumpRange)-1; i>=0; i--)

{

if(dumpRange[i] == ' ' || dumpRange[i] == '\t')

dumpRange[i]='\0';

else

break;

} //뒤에 공백 잘라냄

//////////////////////printf("dumpRange2 ; 뒤의 공백 잘라낸거 :%s\n\n",dumpRange);

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

{

if(dumpRange[i] == ',')

comma\_index1=i;

}

if(comma\_index1==-1) //콤마가 없으면;; edit 은 인자 두개니까 콤마없으면 바로 에러!

return\_flag2=-1;

else

{

start\_flag=0;

for(i=comma\_index1-1; i>=0; i--)

{

if(start\_flag==0 && ((dumpRange[i] != ' ') && (dumpRange[i] != '\t')))

start\_flag=1;

if(start\_flag==1)

{

if((dumpRange[i]>='a' && dumpRange[i]<='f') || (dumpRange[i]>='A' && dumpRange[i]<='F') || (dumpRange[i]>='0' && dumpRange[i]<='9'))

continue;

else

{

return\_flag2=-1;

break;

}

}

}////콤마 앞

start\_flag=0;

for(i=comma\_index1+1; i<strlen(dumpRange); i++)

{

if(start\_flag==0 && ((dumpRange[i] != ' ') && (dumpRange[i] != '\t')))

start\_flag=1;

if(start\_flag==1)

{

if((dumpRange[i]>='a' && dumpRange[i]<='f') || (dumpRange[i]>='A' && dumpRange[i]<='F') || (dumpRange[i]>='0' && dumpRange[i]<='9'))

continue;

else

{

return\_flag2=-1;

break;

}

}

}////콤마 뒤

}

if(comma\_index1 != -1 && return\_flag2 != -1)

{

ptr = strtok(dumpRange,",");

i=1;

while(ptr!=NULL)

{

if(i==1)

address = strtoul(ptr,NULL,16);

else if(i==2)

value = strtoul(ptr,NULL,16);

ptr = strtok(NULL,",");

i++;

}

return\_flag2 = 1;

}

}

if(strstr(command,"fill") || command[0]=='f')

{

if(store2[0]=='f' && store2[1]=='i' && store2[2]=='l' && store2[3]=='l')

{

for(i=0; i<strlen(store2); i++)

{

if(store2[i]=='l' || store2[i]=='L')

break;

}

i++;//엘이 두개라서..

if((store2[i+1] != ' ') && (store2[i+1] != '\t') && store2[i+1] != '\0')

return\_flag2=-1;

}

else if(store2[0]=='f')

{

for(i=0; i<strlen(store2); i++)

{

if(store2[i]=='f' || store2[i]=='F')

break;

}

if((store2[i+1] != ' ') && (store2[i+1] != '\t') && store2[i+1] != '\0')

return\_flag2=-1;

}

else

{

printf("error : 입력받은 명령어에 대문자가 있거나 명령어 사이에 공백이 있습니다.\n");

return\_flag2=-1;

}

strcpy(dumpRange,store2+i+1);//dump 명령어 이후의 것만 dumpRange에 저장

///////////////////////printf("dumpRange :%s\n\n",dumpRange);

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

{

if(dumpRange[i] !=' ' && dumpRange[i] != '\t')

{

j=i;

break;

}

}

strcpy(dumpRange,dumpRange+j);//앞에 공백 잘라냄

//////////////////printf("dumpRange1 ; 앞의 공백 잘라낸거 :%s\n\n",dumpRange);

for(i=strlen(dumpRange)-1; i>=0; i--)

{

if(dumpRange[i] == ' ' || dumpRange[i] == '\t')

dumpRange[i]='\0';

else

break;

} //뒤에 공백 잘라냄

//////////////////////printf("dumpRange2 ; 뒤의 공백 잘라낸거 :%s\n\n",dumpRange);

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

{

if(dumpRange[i] == ',')

{

comma\_index1=i;

break;

}

}//첫번째 콤마

if(comma\_index1 != -1)

{

for(i=comma\_index1+1; i<strlen(dumpRange); i++)

{

if(dumpRange[i] == ',')

comma\_index2=i;

}

}//두번째 콤마

if(comma\_index1==-1 || comma\_index2==-1) //콤마가 하나라도 없으면;; fill은 콤마 없으면 바로 에러!

return\_flag2 = -1;

else

{

start\_flag=0;

for(i=comma\_index1-1; i>=0; i--)

{

if(start\_flag==0 && ((dumpRange[i] != ' ') && (dumpRange[i] != '\t')))

start\_flag=1;

if(start\_flag==1)

{

if((dumpRange[i]>='a' && dumpRange[i]<='f') || (dumpRange[i]>='A' && dumpRange[i]<='F') || (dumpRange[i]>='0' && dumpRange[i]<='9'))

continue;

else

{

return\_flag2=-1;

break;

}

}

}////콤마 앞

start\_flag=0;

for(i=comma\_index2-1; i>comma\_index1; i--)

{

if(start\_flag==0 && ((dumpRange[i] != ' ') && (dumpRange[i] != '\t')))

start\_flag=1;

if(start\_flag==1)

{

if((dumpRange[i]>='a' && dumpRange[i]<='f') || (dumpRange[i]>='A' && dumpRange[i]<='F') || (dumpRange[i]>='0' && dumpRange[i]<='9'))

continue;

else if((dumpRange[i] == ' ') || (dumpRange[i] == '\t'))

{

i--;

while(i>comma\_index1)

{

if((dumpRange[i]==' ') || (dumpRange[i]=='\t'))

continue;

else

{

return\_flag2=-1;

break;

}

}

}

else

{

return\_flag2=-1;

break;

}

}

}////콤마 사이;;;fill은 인자가 3개, 콤마가 두개이므로 처리과정 추가

start\_flag=0;

for(i=comma\_index2+1; i<strlen(dumpRange); i++)

{

if(start\_flag==0 && ((dumpRange[i] != ' ') && (dumpRange[i] != '\t')))

start\_flag=1;

if(start\_flag==1)

{

if((dumpRange[i]>='a' && dumpRange[i]<='f') || (dumpRange[i]>='A' && dumpRange[i]<='F') || (dumpRange[i]>='0' && dumpRange[i]<='9'))

continue;

else

{

return\_flag2=-1;

break;

}

}

}////콤마 뒤

}

if(comma\_index1 != -1 && comma\_index2 != -1 && return\_flag2 != -1)

{

ptr = strtok(dumpRange,",");

i=1;

while(ptr!=NULL)

{

if(i==1)

start = strtoul(ptr,NULL,16);

else if(i==2)

end = strtoul(ptr,NULL,16);

else if(i==3)

value = strtoul(ptr,NULL,16);

ptr = strtok(NULL,",");

i++;

}

return\_flag2 = 1;

}

}

if(strstr(command,"dir") || command[0]=='d')

{

if((!strcmp(store2,"dir")) || (!strcmp(store2,"d")))

return 1;

if(store2[0]=='d' && store2[1]=='i' && store2[2]=='r')

{

for(i=0; i<strlen(store2); i++)

{

if(store2[i]=='r' || store2[i]=='R')

break;

}

if((store2[i+1] != ' ') && (store2[i+1] != '\t') && store2[i+1] != '\0')

return\_flag2=-1;

}

else if(store2[0]=='d')

{

for(i=0; i<strlen(store2); i++)

{

if(store2[i]=='d' || store2[i]=='D')

break;

}

if((store2[i+1] != ' ') && (store2[i+1] != '\t') && store2[i+1] != '\0' && store2[i+1] != 'u')//dump의 맨앞 d와 겹쳐서 따로 처리해준것!

return\_flag2=-1;

else if((store2[i+1] == ' ') || (store2[i+1] == '\t'))

{

for(i=i+1;i<strlen(store2);i++)

{

if((store2[i] != ' ') && (store[i] != '\t'))

return -1;//이 명령어는 한글자로도 되기때문에(d) 그 뒤에 공백이 있는게 한글자라서인지 잘못된명령어인지 예외처리;;;뒤의 h,q등도 비슷

}

}

}

else

{

printf("error : 입력받은 명령어에 대문자가 있거나 명령어 사이에 공백이 있습니다.\n");

return\_flag2=-1;

}

}

if(strstr(command,"help") || command[0]=='h')

{

if((!strcmp(store2,"help")) || (!strcmp(store2,"h")))

return 1;

if(store2[0]=='h' && store2[1]=='e' && store2[2]=='l' && store2[3]=='p')

{

for(i=0; i<strlen(store2); i++)

{

if(store2[i]=='p' || store2[i]=='P')

break;

}

if((store2[i+1] != ' ') && (store2[i+1] != '\t') && store2[i+1] != '\0')

return\_flag2=-1;

}

else if(store2[0]=='h')

{

for(i=0; i<strlen(store2); i++)

{

if(store2[i]=='h' || store2[i]=='H')

break;

}

if((store2[i+1] != ' ') && (store2[i+1] != '\t') && store2[i+1] != '\0')

return\_flag2=-1;

else if((store2[i+1] == ' ') || (store2[i+1] == '\t'))

{

for(i=i+1;i<strlen(store2);i++)

{

if((store2[i] != ' ') && (store[i] != '\t'))

return -1;

}

}

}

else

{

printf("error : 입력받은 명령어에 대문자가 있거나 명령어 사이에 공백이 있습니다.\n");

return\_flag2=-1;

}

}

if(strstr(command,"history") || strstr(command,"hi"))

{

if((!strcmp(store2,"history")) || (!strcmp(store2,"hi")))

return 1;

if(store2[0]=='h' && store2[1]=='i' && store2[2]=='s' && store2[3]=='t' && store2[4]=='o' && store2[5]=='r' && store2[6]=='y')

{

for(i=0; i<strlen(store2); i++)

{

if(store2[i]=='y' || store2[i]=='Y')

break;

}

if((store2[i+1] != ' ') && (store2[i+1] != '\t') && store2[i+1] != '\0')

return\_flag2=-1;

}

else if(store2[0]=='h' && store2[1]=='i')

{

for(i=0; i<strlen(store2); i++)

{

if(store2[i]=='i' || store2[i]=='I')

break;

}

if((store2[i+1] != ' ') && (store2[i+1] != '\t') && store2[i+1] != '\0')

return\_flag2=-1;

else if((store2[i+1] == ' ') || (store2[i+1] == '\t'))

{

for(i=i+1;i<strlen(store2);i++)

{

if((store2[i] != ' ') && (store[i] != '\t'))

return -1;

}

}

}

else

{

printf("error : 입력받은 명령어에 대문자가 있거나 명령어 사이에 공백이 있습니다.\n");

return\_flag2=-1;

}

}

if(strstr(command,"quit") || command[0]=='q')

{

if((!strcmp(store2,"quit")) || (!strcmp(store2,"q")))

return 1;

if(store2[0]=='q' && store2[1]=='u' && store2[2]=='i' && store2[3]=='t')

{

for(i=0; i<strlen(store2); i++)

{

if(store2[i]=='t' || store2[i]=='T')

break;

}

if((store2[i+1] != ' ') && (store2[i+1] != '\t') && store2[i+1] != '\0')

return\_flag2=-1;

}

else if(store2[0]=='q')

{

for(i=0; i<strlen(store2); i++)

{

if(store2[i]=='q' || store2[i]=='Q')

break;

}

if((store2[i+1] != ' ') && (store2[i+1] != '\t') && store2[i+1] != '\0')

return\_flag2=-1;

else if((store2[i+1] == ' ') || (store2[i+1] == '\t'))

{

for(i=i+1;i<strlen(store2);i++)

{

if((store2[i] != ' ') && (store[i] != '\t'))

return -1;

}

}

}

else

{

printf("error : 입력받은 명령어에 대문자가 있거나 명령어 사이에 공백이 있습니다.\n");

return\_flag2=-1;

}

}

if(strstr(command,"reset"))

{

if((!strcmp(store2,"reset")))

return 1;

if(store2[0]=='r' && store2[1]=='e' && store2[2]=='s' && store2[3]=='e' && store2[4]=='t')

{

for(i=0; i<strlen(store2); i++)

{

if(store2[i]=='t' || store2[i]=='T')

break;

}

if((store2[i+1] != ' ') && (store2[i+1] != '\t') && store2[i+1] != '\0')

return\_flag2=-1;

}

else

{

printf("error : 입력받은 명령어에 대문자가 있거나 명령어 사이에 공백이 있습니다.\n");

return\_flag2=-1;

}

}

if(strstr(command,"opcode"))

{

if(!strcmp(store2,"opcodelist"))

return 1;

if(store2[0]=='o' && store2[1]=='p' && store2[2]=='c' && store2[3]=='o' && store2[4]=='d' && store2[5]=='e')

{

for(i=0; i<strlen(store2); i++)

{

if(store2[i]=='e' || store2[i]=='E')

break;

}

if((store2[i+1] != ' ') && (store2[i+1] != '\t') && store2[i+1] != '\0')

return\_flag2=-1;

}

else

{

printf("error : 입력받은 명령어에 대문자가 있거나 명령어 사이에 공백이 있습니다.\n");

return\_flag2=-1;

}

strcpy(dumpRange,store2+i+1);//dump 명령어 이후의 것만 dumpRange에 저장

///////////////////////printf("dumpRange :%s\n\n",dumpRange);

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

{

if(dumpRange[i] !=' ' && dumpRange[i] != '\t')

{

j=i;

break;

}

}

strcpy(dumpRange,dumpRange+j);//앞에 공백 잘라냄

//////////////////printf("dumpRange1 ; 앞의 공백 잘라낸거 :%s\n\n",dumpRange);

for(i=strlen(dumpRange)-1; i>=0; i--)

{

if(dumpRange[i] == ' ' || dumpRange[i] == '\t')

dumpRange[i]='\0';

else

break;

} //뒤에 공백 잘라냄

/////////////////////printf("dumpRange2 ; 뒤의 공백 잘라낸거 :%s\n\n",dumpRange);

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

{

if((dumpRange[i]>='A' && dumpRange[i]<='Z'))

continue;

else

{

return\_flag2=-1;

break;

}

}

if(return\_flag2 != -1)

{

strcpy(mnemonic,dumpRange);

return\_flag2 = 1;

}

else

printf("error! : 적절하지 않은 mnemonic입니다. (소문자, 숫자 등등)\n");

}

if(return\_flag1 == -1 || return\_flag2 == -1)

return -1;//정상 명령어 아니고 예외처리되어있음

else if(return\_flag1==1 && return\_flag2==1)

return 1; //정상 명령어임

}

Node\* Create\_Node(char\* data)

{

Node\* newnode=(Node\*)malloc(sizeof(Node));

strcpy(newnode->data,data);

newnode->link=NULL;

return newnode;

}//노드 생성

Table\* Create\_Table(int key, char\* name)

{

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

new->id = key;

strcpy(new->instruct,name);

new->hashNext=NULL;

return new;

}//테이블에 필요한 노드 생성

void insertBack(Node\*\* head, Node\*\* tail, Node\* newnode)

{

if((\*head) == NULL)

{

(\*head) = newnode;

(\*tail) = newnode;

}

else

{

(\*tail)->link = newnode;

(\*tail) = newnode;

}

}//히스토리에 이용하기 위한 연결리스트 삽입 함수

void DumpEditFill(int start, int end, int address, int value)

{

signed long i,j;

signed long temp;

signed long endTemp;

unsigned int endLinenum;

if(start!=-1 && end==-1 && address==-1 && value==-1 && start<=1048575)//dump start일때처리

{

insertBack(&head,&tail,Create\_Node(store\_command));

temp = start%16;

line\_num = (unsigned int)(start-temp);

dump\_p = (unsigned int)start;

for(i=0; i<10; i++)

{

if(line\_num > 1048560)

break;//FFFFF보다 큰 수 들어올때 처리

printf("%05X ",line\_num);

if(i==0)

{

for(j=0; j<temp; j++)

printf(" ");

for(j=temp; j<16; j++)

{

printf("%02X ",virtual[dump\_p]);

dump\_p++;

}

printf("; ");

dump\_p = dump\_p-(unsigned int)(16-temp);

for(j=0; j<temp; j++)

printf(".");

for(j=temp; j<16; j++)

{

if((virtual[dump\_p]>=0x20) && (virtual[dump\_p]<=0x7E))

printf("%c",virtual[dump\_p]);

else

printf(".");

dump\_p++;

}

printf("\n");

}

else

{

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

{

printf("%02X ",virtual[dump\_p]);

dump\_p++;

}

printf("; ");

dump\_p = dump\_p - 16;

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

{

if((virtual[dump\_p]>=0x20) && (virtual[dump\_p]<=0x7E))

printf("%c",virtual[dump\_p]);

else

printf(".");

dump\_p++;

}

printf("\n");

}

line\_num+=16;

}

if(line\_num <= 1048560 && temp!=0)

{

printf("%05X ",line\_num);

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

{

printf("%02X ",virtual[dump\_p]);

dump\_p++;

}

for(i=temp; i<16; i++)

printf(" ");

printf("; ");

dump\_p = dump\_p - temp;

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

{

if((virtual[dump\_p]>=0x20) && (virtual[dump\_p]<=0x7E))

printf("%c",virtual[dump\_p]);

else

printf(".");

dump\_p++;

}

for(i=temp; i<16; i++)

printf(".");

printf("\n");//160개 출력을 위한 처리

}

}

else if(start!=-1 && end==-1 && address==-1 && value==-1 && start>1048575)

printf("error! : 0xFFFFF 보다 큰 값은 주소로 들어올 수 없습니다.\n");

if(start!=-1 && end!=-1 && address==-1 && value==-1 && start<=1048575 && end<=1048575)//dump start end일때

{

if(start >= end)

{

printf("error! : start는 end값보다 크거나 같을 수 없습니다.\n");

return;

}//start>end일때 에러처리

insertBack(&head,&tail,Create\_Node(store\_command));

temp = start%16;

line\_num = (unsigned int)(start-temp);

endTemp = end%16;

endLinenum = (unsigned int)(end-endTemp);

dump\_p = (unsigned int)start;

if((line\_num == endLinenum) && ((end-start)<16))//범위가 한 라인 이내일때

{

printf("%05X ",line\_num);

if(line\_num == (unsigned int)(start-temp))

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

printf(" ");

for(i=temp; i<=endTemp; i++)

{

printf("%02X ",virtual[dump\_p]);

dump\_p++;

}

for(i=endTemp+1; i<16; i++)

printf(" ");

printf("; ");

dump\_p = dump\_p - (unsigned int)(endTemp-temp+1);

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

printf(".");

for(i=temp; i<=endTemp; i++)

{

if((virtual[dump\_p]>=0x20) && (virtual[dump\_p]<=0x7E))

printf("%c",virtual[dump\_p]);

else

printf(".");

dump\_p++;

}

for(i=endTemp+1; i<16; i++)

printf(".");

}

while((line\_num <= endLinenum) && ((end-start>=16)))//범위가 여러라인으로 넘어갈때

{

if(line\_num > 1048560)

break;

printf("%05X ",line\_num);

if(line\_num == (unsigned int)(start-temp))

{

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

printf(" ");

for(i=temp; i<16; i++)

{

printf("%02X ",virtual[dump\_p]);

dump\_p++;

}

printf("; ");

dump\_p = dump\_p - (unsigned int)(16-temp);

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

printf(".");

for(i=temp; i<16; i++)

{

if((virtual[dump\_p]>=0x20) && (virtual[dump\_p]<=0x7E))

printf("%c",virtual[dump\_p]);

else

printf(".");

dump\_p++;

}

}//첫줄일때

else if(line\_num == endLinenum)

{

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

{

printf("%02X ",virtual[dump\_p]);

dump\_p++;

}

for(i=(endTemp+1); i<16; i++)

printf(" ");

printf("; ");

dump\_p = dump\_p - (unsigned int)endTemp;

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

{

if((virtual[dump\_p]>=0x20) && (virtual[dump\_p]<=0x7E))

printf("%c",virtual[dump\_p]);

else

printf(".");

dump\_p++;

}

for(i=(endTemp+1); i<16; i++)

printf(".");

}//마지막줄일때

else

{

for(i=0; i<16; i++)

{

printf("%02X ",virtual[dump\_p]);

dump\_p++;

}

printf("; ");

dump\_p = dump\_p - 16;

for(i=0; i<16; i++)

{

if((virtual[dump\_p]>=0x20) && (virtual[dump\_p]<=0x7E))

printf("%c",virtual[dump\_p]);

else

printf(".");

dump\_p++;

}

}//나머지 가운데 줄일때

printf("\n");

line\_num = line\_num + 16;

}

}

else if(start!=-1 && end!=-1 && address==-1 && value==-1 && (start>1048575 || end>1048575))

printf("error! : 0xFFFFF 보다 큰 값은 주소로 들어올 수 없습니다.\n");

if(start==-1 && end==-1 && address!=-1 && value!=-1 && address<=1048575 && value<=255)//edit address value 일때

{

insertBack(&head,&tail,Create\_Node(store\_command));

dump\_p = (unsigned int)address;

virtual[dump\_p] = value;

}

else if(start==-1 && end==-1 && address!=-1 && value!=-1 && (address>1048575 || value>255))

printf("error! : 0xFFFFF 보다 큰 값은 주소로 들어올 수 없습니다. value값은 0xFF보다 클 수 없습니다.\n");

if(start!=-1 && end!=-1 && address==-1 && value!=-1 && start<=1048575 && end<=1048575 && value<=255)//fill start end value일때

{

insertBack(&head,&tail,Create\_Node(store\_command));

dump\_p = (unsigned int)start;

while(dump\_p <= (unsigned int)end)

{

virtual[dump\_p] = value;

dump\_p++;

}

}

else if(start!=-1 && end!= -1 && address==-1 && value!=-1 && (start>1048575 || end>1048575 || value>255))

printf("error! : 0xFFFFF 보다 큰 값은 주소로 들어올 수 없습니다. value값은 0xFF보다 클 수 없습니다.\n");

}

int hashFunc(char\* mnemonic)

{

unsigned int hashkey=30;

unsigned int key;

char name[MAX];

char format[MAX];

FILE \*fp;

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

while(1)

{

if(fscanf(fp,"%X %s %s\n",&key,name,format)==EOF)

break;

else

{

if(!strcmp(mnemonic,name))

hashkey = HASH\_KEY(key);//매크로에 있음

}

}

fclose(fp);

return hashkey;

}//해시함수 ;; 명령어 mnemonic을 받아서 그 명령어의 값을 함수를 통해 키값으로 변환하여 반환해줌.