과목 명: 시스템프로그래밍

담당 교수 명: 소 정 민

<<Assignment 1>>

**서강대학교 컴퓨터공학과**

**[20151561]**

**[신용하]**

목 차

1. 프로그램 개요 4

2. 프로그램 설명 4

2.1 프로그램 흐름도 4

3. 모듈 정의 5

3.1 모듈 이름 : main() 5

3.1.1 기능 5

3.1.2 사용 변수 5

3.2 모듈 이름: help() 5

3.2.1 기능 5

3.2.2 사용 변수 5

3.3 모듈이름: dir() 5

3.3.1 기능 5

3.3.2 사용변수 5

3.4 모듈이름: quit() 5

3.4.1 기능 5

3.4.2 사용변수 6

3.5 모듈이름: history(); 6

3.5.1 기능 6

3.5.2 사용변수 6

3.6 모듈이름: dump() 6

3.6.1 기능 6

3.6.2 사용변수 6

3.7 모듈이름: edit() 6

3.7.1 기능 6

3.7.2 사용변수 6

3.8 모듈이름: fill() 6

3.8.1 기능 6

3.8.2 사용변수 6

3.9 모듈이름: reset() 6

3.9.1 기능 6

3.9.2 사용변수 6

3.10 모듈이름: opmn() 7

3.10.1 기능 7

3.10.2 사용변수 7

3.11 모듈이름: oplist() 7

3.11.1 기능 7

3.11.2 사용변수 7

3.12 모듈이름: makenum() 7

3.12.1 기능 7

3.12.2 사용변수 7

3.13 모듈이름: perform() 7

3.13.1 기능 7

3.13.2 사용변수 7

3.14 모듈이름: putnode() 8

3.14.1 기능 8

3.14.2 사용변수 8

3.15 모듈이름: puthashnode() 8

3.15.1 기능 8

3.15.2 사용변수 8

4. 전역 변수 정의 8

4.1 unsigned char memory[0x100000] 8

4.2 char com[500], tmp[500] 8

4.3 int dumpptr 8

4.4 char mntmp[10] 8

4.5 struct node\* head, tail 8

4.6 struct hashnode\* hashtable[20] 8

5. 코드 9

# 프로그램 개요

1MB의 메모리와 opcode table을 가진 shell environment를 직접 구현한 프로그램이다. 사용자는 shell, memory, opcode에 관련된 여러 command를 입력하여 기능을 수행할 수 있다.

# 프로그램 설명

## 프로그램 흐름도

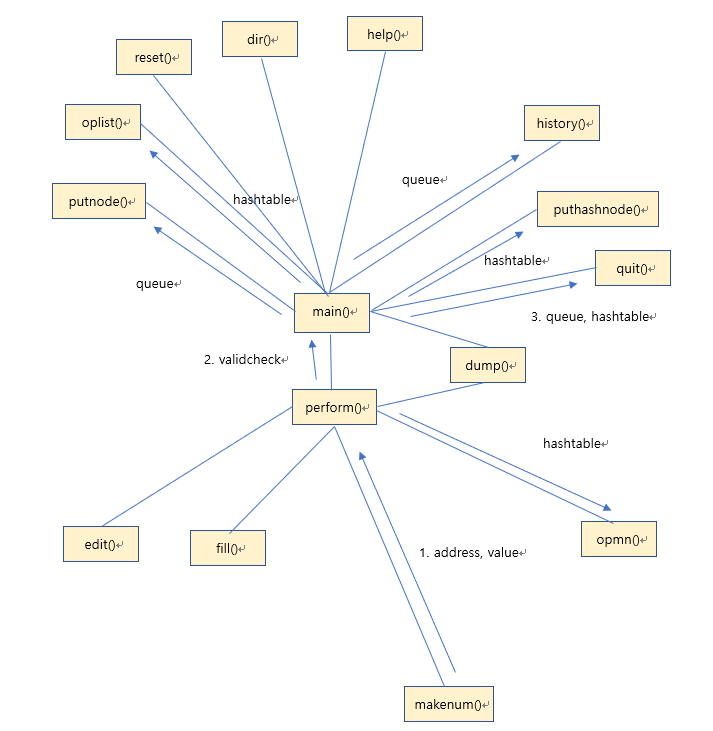


그림 1> 프로그램 흐름도

# 모듈 정의

## 모듈 이름 : main()

### 기능

사용자에게 실행 화면을 보여주고 command를 받아 적절한 다른 모듈로 보낸다.

opcode.txt 파일을 읽어서 opcode table 을 hash table의 형태로 만든다.

### 사용 변수

int i, j - 반복문을 위한 변수

int l - 입력 command의 길이를 저장

int mnvalue – mnemonic의 hash 값을 저장하여 그에 해당하는 bucket에 넣기 위한 변수

int validcheck - 1이면 command가 제대로 작동, 0이면 invalid한 command

char optmp[4] - opcode.txt의 opcode 부분을 문자열로 저장

char fotmp[5] – opcode.txt의 format 부분을 문자열로 저장

char \*pos – strtoul함수를 실행하기 위한 변수

unsigned int opvalue – 문자열로 저장된 opcode를 숫자로 변환하여 저장

FILE \*fp – opcode.txt 를 열기 위한 변수

## 모듈 이름: help()

### 기능

사용자에게 입력할 수 있는 command를 알려준다..

### 사용 변수

없음

## 모듈이름: dir()

### 기능

사용자에게 현재 위치의 디렉토리 안의 파일들을 보여준다. 이 때, 파일 이름의 끝에 / 가 있으면 디렉토리이고, \*가 있으면 실행파일이다.

### 사용변수

DIR \* dp – 현재 디렉토리의 정보를 받는 변수

suruct dirent \*dirp – 디렉토리의 정보가 이 구조체에 저장된다.

char filename[500] – 디렉토리 안의 파일 이름을 저장해주는 변수

struct stat st 파일의 정보를 알려주는 구조체

## 모듈이름: quit()

### 기능

동적 할당된 메모리를 모두 해제하고 프로그램을 종료한다.

### 사용변수

struct node\* a, b – 입력 command 를 저장하는 큐를 해제하기 위한 변수

struct hashnode\* c, d – hashtable 구조체를 해제하기 위한 변수

int i – 반복문을 위한 변수

## 모듈이름: history()

### 기능

사용자에게 지금까지 입력한 모든 valid한 command 들을 순서대로 보여준다. (history 또는 hi 입력을 하면 마지막에 이 command도 포함하여 보여준다)

### 사용변수

int i - command들의 순서를 알려준다.

struct node\* ptr – command 들이 저장된 큐의 내용을 읽기 위한 변수

## 모듈이름: dump()

### 기능

사용자에게 memory에 저장된 값을 16진수로 보여준다.

사용자는 세 가지 경우의 입력을 할 수 있는데, dump만 입력하면 마지막으로 보여준 값의 다음부터 160byte의 값을 보여준다. 만약 dump를 처음에 실행한다면, 첫번째부터 160byte를 보여준다. dump start로 입력하면 start 부터 160byte를 보여준다. 만약 현재 주소로부터 160byte가 메모리의 끝부분을 넘는다면 메모리의 끝까지 출력하고 후에 dump를 실행할 시 처음부터 보여준다. dump start, end 로 입력했을 시 start 부터 end까지 보여준다. 메모리의 주소를 넘는 값과 end보다 큰 start 값은 오류이다.

### 사용변수

int i, j – 반복문을 위한 변수

## 모듈이름: edit()

### 기능

사용자가 입력한 주소에 입력한 값을 집어넣어준다. 입력 주소는 메모리 범위 안에 있어야 하고, 입력값은 00 ~ FF 여야 한다.

### 사용변수

없음

## 모듈이름: fill()

### 기능

사용자가 입력한 주소 start 부터 end 까지 입력한 값을 채운다. 입력 주소는 메모리 범위여야 하고, start 가 end 보다 클 수 없고, 입력값은 00 ~ FF 사이여야 한다.

### 사용변수

int i – 반복문을 위한 변수

## 모듈이름: reset()

### 기능

메모리의 값을 모두 0으로 초기화한다.

### 사용변수

int i – 반복문을 위한 변수

## 모듈이름: opmn()

### 기능

사용자가 입력한 opcode mnemonic과 일치하는 것을 opcode.txt 을 읽어 만든 hashtable에서 찾아준다.

### 사용변수

struct hashnode\* ptr – hashtable 의 내용을 읽어 원하는 mnemonic을 찾기 위한 변수

int i – 반복문을 위한 변수

char tmp1[10], tmp2[10] – tmp1은 사용자가 입력한 mnemonic을 저장, tmp2는 hashtable 안에 있는 mnemonic을 비교하기 위해 저장.

## 모듈이름: oplist()

### 기능

Opcode hashtable을 bucket 순서대로 보여준다.

### 사용변수

int i, j – 반복문을 위한 변수

struct hashnode \*ptr – hashtable의 내용을 읽기 위한 변수

## 모듈이름: makenum()

### 기능

사용자에게 입력 받은 주소와 값은 문자열로 저장이 되어있으므로 이 모듈을 통해 문자열에서 16진수로 변환시킨다.

### 사용변수

int i – 반복문을 위한 변수

value – 주소 또는 값이 저장될 변수

## 모듈이름: perform()

### 기능

command 중에 dump, edit, fill, opcode mnemonic은 뒤에 주소나 값이 붙기 때문에 main()에서 처리하면 내용이 길어져 가독성이 떨어지므로 perform()에서 처리한다. 입력된 값이나 주소가 주어진 command에서 valid한가 체크하고, valid하다면 기능을 수행하고 main()의 validcheck를 1로 만든다. Invalid하다면 실행하지 않고 validcheck를 0으로 만든다.

### 사용변수

int i – 반복문을 위한 변수

int a, b, c – 입력된 command 뒤의 주소와 값을 a, b, c에 각각 저장한다.(command 뒤에는 3개의 입력이 최대이기 때문)

int comcheck – 입력된 값과 주소로 command를 실행했을 때 잘 실행되었다면 1, 오류가 있다면 음수가 된다.

int commaptr1, commaptr2, commacount – 입력될 값이나 주소가 최대 3개라면 그들을 구분할 comma의 개수도 최대 2개여야만 할 것이고, 이를 각각 commaptr1, commaptr2에 저장한다. commacount는 문자열의 comma 개수를 세어주는 역할을 한다. comma의 위치를 알면 그 사이에 문자열이 있기 때문에 값이나 주소를 각각 읽는데 도움을 준다.

## 모듈이름: putnode()

### 기능

사용자가 입력한 command를 저장하는 큐에 valid 한 command를 저장할 때 사용하는 모듈.

### 사용변수

struct node \*new – 큐에 넣을 새로운 구조체를 만들 때 사용.

## 모듈이름: puthashnode()

### 기능

Hash된 mnemonic을 hashtable의 알맞은 bucket에 넣는다. 이 때, collision이 발생했다면 같은 bucket에 스택 형식으로 저장시킨다.

### 사용변수

struct hashnode \*new – hashtable에 넣을 새로운 구조체를 만들 때 사용

# 전역 변수 정의

## unsigned char memory[0x100000]

1mb의 메모리를 만든다. 값은 모두 0으로 초기화된다.

## char com[500], tmp[500]

사용자에게 입력 받은 command를 com[500]에 저장하고, 이를 그대로 tmp[500]에 저장한다. com[500]는 프로그램 수행 도중에 수정되기 때문이다. tmp[500]에 저장해 두는 이유는 나중에 vaild한 command라면, 이를 큐에 저장해야 되기 때문이다. 크기가 500인 것은 입력될 command가 500자를 넘지 않을 것이라 가정한 것이다.

## int dumpptr

dump를 수행할 때 필요한 memory 의 시작 주소를 가지고 있다.

## char mntmp[10]

opcode command에서 사용자에게 입력 받은 mnemonic이 저장된다.

## struct node\* head, tail

command를 저장하는 큐의 시작 부분은 head, 끝부분은 tail 이 가리킨다.

## struct hashnode\* hashtable[20]

Bucket이 20개인 hashtable을 만든다.

# 코드

//20151561.h

void help();

int dir();

void quit();

void history();

int dump(int start, int end);

int edit(int address, int value);

int fill(int start, int end, int value);

void reset();

int opmn(int start, int end, int a);

void oplist();

int makenum(int start, int end);

int perform(int numptr, int whatcom);

void putnode();

void puthashnode(int opvalue, int mnvalue);

//20151561.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <sys/stat.h>

#include "20151561.h"

unsigned char memory[0x100000] = {0, };

char com[500];

char tmp[500];

int dumpptr = 0;

char mntmp[10];

struct node{

char command[500];

struct node \*next;

};

struct node \*head;

struct node \*tail;

struct hashnode{

char mn[10];

int op;

struct hashnode \*next;

};

struct hashnode\* hashtable[20];

int main(){

int i, j, l, mnvalue, validcheck = 0;

char optmp[4];

char fotmp[5];

char \*pos = NULL;

unsigned int opvalue;

FILE \*fp;

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

head = (struct node\*)malloc(sizeof(struct node));

head->next = NULL;

tail = head;

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

hashtable[i] = NULL;

}

while(1){

opvalue = 0;

mnvalue = 0;

fscanf(fp, "%s%s%s", optmp, mntmp, fotmp);

if(feof(fp)){

break;

}

opvalue = strtoul(optmp, &pos, 16);

for(i=0; i<strlen(mntmp); i++){

mnvalue = mnvalue + mntmp[i];

}

mnvalue = mnvalue%20;

puthashnode(opvalue, mnvalue);

}

fclose(fp);

while(1){

printf("sicsim> ");

fgets(com, 500, stdin);

l = strlen(com) - 1;

com[l] = '\0';

if(com[0] == '\0'){

printf("Enter the command\n");

continue;

}

strcpy(tmp, com);

for(i=0; i<strlen(com); i++){

if(com[i] == '\t'){

com[i] = ' ';

}

}

while(1){

if(com[0] == ' '){

for(i=0; i<strlen(com); i++){

com[i] = com[i+1];

}

}

else{

break;

}

}

for(i=strlen(com)-1; i>=0; i--){

if(com[i] == ' '){

com[i] = '\0';

}

else{

break;

}

}

i=0;

while(com[i] != '\0'){

if((com[i] == ' ') && ((com[i+1] == ' ') || (com[i+1] == ',') || (com[i-1] == ','))){

for(j=i;j<strlen(com); j++){

com[j] = com[j+1];

}

continue;

}

i++;

}

for(i=0; i<strlen(com); i++){

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

com[i] = com[i] + 32;

}

}

if(((strncmp(com, "help", 4) == 0) && (com[4] == '\0')) || ((com[0] == 'h') && (com[1] == '\0'))){

help();

validcheck = 1;

}

else if(((strncmp(com, "dir", 3) == 0) && (com[3] == '\0')) || ((com[0] == 'd') && (com[1] == '\0'))){

validcheck = dir();

}

else if(((strncmp(com, "quit", 4) == 0) && (com[4] == '\0')) || ((com[0] == 'q') && (com[1] == '\0'))){

quit();

}

else if(((strncmp(com, "history", 7) == 0) && (com[7] == '\0')) || ((com[0] == 'h') && (com[1] == 'i') && (com[2] == '\0'))){

putnode();

history();

}

else if(((strncmp(com, "dump", 4) == 0) && (com[4] == '\0')) || ((com[0] == 'd') && (com[1] == 'u') && (com[2] == '\0'))){

if(dumpptr+159 > 0xFFFFF){

validcheck = dump(dumpptr, 0xFFFFF);

}

else{

validcheck = dump(dumpptr, dumpptr+159);

}

}

else if(((strncmp(com, "dump", 4) == 0) && (com[4] == ' ')) || ((com[0] == 'd') && (com[1] == 'u') && (com[2] == ' '))){

if(strncmp(com, "dump", 4) == 0){

validcheck = perform(5, 1);

}

else{

validcheck = perform(3, 1);

}

}

else if(((strncmp(com, "edit", 4) == 0) && (com[4] == ' ')) || ((com[0] == 'e') && (com[1] == ' '))){

if(strncmp(com, "edit", 4) == 0){

validcheck = perform(5, 2);

}

else{

validcheck = perform(2, 2);

}

}

else if(((strncmp(com, "fill", 4) == 0) && (com[4] == ' ')) || ((com[0] == 'f') && (com[1] == ' '))){

if(strncmp(com, "fill", 4) == 0){

validcheck = perform(5, 3);

}

else{

validcheck = perform(2, 3);

}

}

else if(strncmp(com, "reset", 5) == 0){

reset();

validcheck = 1;

}

else if((strncmp(com, "opcode", 6) == 0) && (com[6] == ' ')){

validcheck = perform(7, 4);

}

else if((strncmp(com, "opcodelist", 10) == 0) && (com[10] == '\0')){

oplist();

validcheck = 1;

}

else{

printf("Invalid command\n");

}

if(validcheck == 1){

putnode();

}

validcheck = 0;

}

return 0;

}

void help(){

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");

}

int dir(){

DIR \*dp;

struct dirent \*dirp;

char filename[500];

struct stat st;

dp = opendir(".");

if(dp == NULL){

printf("opendir error\n");

return 0;

}

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

if((strcmp(dirp->d\_name, ".") == 0) || (strcmp(dirp->d\_name,"..") == 0)){

continue;

}

sprintf(filename, "./%s", dirp->d\_name);

if(stat(filename, &st) == -1){

printf("stat error\n");

return 0;

}

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

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

printf("/ ");

}

else if(st.st\_mode & S\_IXUSR){

printf("\* ");

}

else{

printf(" ");

}

}

closedir(dp);

printf("\n");

return 1;

}

void quit(){

struct node\* a = head;

struct node\* b = head->next;

struct hashnode\* c;

struct hashnode\* d;

int i;

while(b != NULL){

a->next = NULL;

free(a);

a = b;

b = a->next;

}

free(a);

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

if(hashtable[i] == NULL){

continue;

}

else{

c = hashtable[i];

d = hashtable[i]->next;

while(d != NULL){

c->next = NULL;

free(c);

c = d;

d = c->next;

}

free(c);

}

}

exit(1);

}

void history(){

int i = 1;

struct node\* ptr = head->next;

while(ptr != NULL){

printf("%d %s\n", i, ptr->command);

ptr = ptr->next;

i++;

}

}

int dump(int start, int end){

int i, j;

if((start < 0) || (end > 0xFFFFF) || (start > 0xFFFFF) || (end < 0)){

return -1;

}

if(start > end){

return -2;

}

printf("%05X ", start - (start%0x10));

for(i=0; i<(start%0x10); i++){

printf(" ");

}

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

printf("%02X ", memory[i]);

if((i+1)%0x10 == 0){

printf("; ");

for(j=i-0xF; j<=i; j++){

if(j < start){

printf(".");

}

else{

if((memory[j] >= 0x20) && (memory[j] <= 0x7E)){

printf("%c", memory[j]);

}

else{

printf(".");

}

}

}

printf("\n");

if(i != end){

printf("%05X ", i+1);

}

}

}

if(end%0x10 != 15){

for(i=end+1; i%0x10 != 0; i++){

printf(" ");

}

printf("; ");

for(i=end-end%0x10; i<=end; i++){

if(i < start){

printf(".");

}

else{

if((memory[i] >= 0x20) && (memory[i] <= 0x7E)){

printf("%c", memory[i]);

}

else{

printf(".");

}

}

}

for(i=end+1; i%0x10 != 0; i++){

printf(".");

}

printf("\n");

}

dumpptr = end + 1;

if(dumpptr > 0xFFFFF){

dumpptr = 0;

}

return 1;

}

int edit(int address, int value){

if((address < 0) || (address > 0xFFFFF) || (value > 0xFF)){

return -1;

}

memory[address] = value;

return 1;

}

int fill(int start, int end, int value){

int i;

if((start < 0) || (start > 0xFFFFF) || (end < 0) || (end > 0xFFFFF) || (value > 0xFF)){

return -1;

}

if(start > end){

return -2;

}

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

memory[i] = value;

}

return 1;

}

void reset(){

int i;

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

memory[i] = 0;

}

}

int opmn(int start, int end, int a){

struct hashnode\* ptr;

int i;

char tmp1[10];

char tmp2[10];

ptr = hashtable[a];

if((end-start+1) > 9){

return -1;

}

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

tmp1[i-start] = com[i] - 'a' + 'A';

}

tmp1[end-start+1] = com[i];

while(ptr != NULL){

strcpy(tmp2, ptr->mn);

if(strcmp(tmp1, tmp2) == 0){

printf("opcode is %02X.\n", ptr->op);

return 1;

}

ptr = ptr->next;

}

return -1;

}

void oplist(){

int i, j;

struct hashnode \*ptr;

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

printf("%d :", i);

ptr = hashtable[i];

j = 0;

while(ptr != NULL){

if(j == 0){

printf(" [%s, %02X]", ptr->mn, ptr->op);

}

else{

printf(" -> [%s, %02X]", ptr->mn, ptr->op);

}

j++;

ptr = ptr->next;

}

printf("\n");

}

}

int makenum(int start, int end){

int i, value = 0;

if(end-start == 1){

if((com[start] == '0') && (com[end] == 'x')){

return -1;

}

}

if((end-start > 1) && ((com[start] == '0') && (com[start+1] == 'x'))){

for(i=start+2; i<=end; i++){

if((com[i] >= 'a') && (com[i] <= 'f')){

value = (value\*0x10) + com[i] - 'a' + 10;

}

else if((com[i] >= '0') && (com[i] <= '9')){

value = (value\*0x10) + com[i] - '0';

}

else{

return -1;

}

}

}

else{

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

if((com[i] >= 'a') && (com[i] <= 'f')){

value = (value\*0x10) + com[i] - 'a' + 10;

}

else if((com[i] >= '0') && (com[i] <= '9')){

value = (value\*0x10) + com[i] - '0';

}

else{

return -1;

}

}

}

return value;

}

int perform(int numptr, int whatcom){

int i, a, b, c, comcheck, commaptr1, commaptr2, commacount = 0;

int comlen = strlen(com);

if(com[numptr] == ','){

printf("Invalid command\n");

return 0;

}

if(com[comlen-1] == ','){

printf("Invalid command\n");

return 0;

}

for(i=numptr; i<comlen; i++){

if(com[i] == ','){

commacount++;

}

if(com[i] == ' '){

printf("Invalid command\n");

return 0;

}

}

if(commacount == 1){

for(i=numptr; i<comlen; i++){

if(com[i] == ','){

commaptr1 = i;

}

}

}

if(commacount == 2){

for(i=numptr; i<comlen; i++){

if(com[i] == ','){

commaptr1 = i;

break;

}

}

if(com[commaptr1 + 1] == ','){

printf("Invalid command\n");

return 0;

}

else{

for(i=commaptr1+1; i<comlen; i++){

if(com[i] == ','){

commaptr2 = i;

}

}

}

}

if(commacount > 2){

printf("Invalid command\n");

return 0;

}

if(commacount == 0){

if(whatcom == 4){

a = 0;

for(i=numptr; i<comlen; i++){

a = a + com[i] - 'a' + 'A';

}

a = a%20;

}

else{

a = makenum(numptr, comlen-1);

if(a == -1){

printf("Invalid input number\n");

return 0;

}

}

}

if(commacount == 1){

a = makenum(numptr, commaptr1-1);

if(a == -1){

printf("Invalid input number\n");

return 0;

}

b = makenum(commaptr1+1, comlen-1);

if(b == -1){

printf("Invalid input number\n");

return 0;

}

}

if(commacount == 2){

a = makenum(numptr, commaptr1-1);

if(a == -1){

printf("Invalid input number\n");

return 0;

}

b = makenum(commaptr1+1, commaptr2-1);

if(b == -1){

printf("Invalid input number\n");

return 0;

}

c = makenum(commaptr2+1, comlen-1);

if(c == -1){

printf("Invalid input number\n");

return 0;

}

}

if(whatcom == 1){

if(commacount == 0){

if(a+159 > 0xFFFFF){

comcheck = dump(a, 0xFFFFF);

if(comcheck == -1){

printf("Not within the boundary\n");

return 0;

}

}

else{

comcheck = dump(a, a+159);

}

}

else if(commacount == 1){

comcheck = dump(a, b);

if(comcheck == -1){

printf("Not within the boundary\n");

return 0;

}

if(comcheck == -2){

printf("Start is larger than end\n");

return 0;

}

}

else{

printf("Invalid command\n");

return 0;

}

}

else if(whatcom == 2){

if(commacount == 1){

comcheck = edit(a, b);

if(comcheck == -1){

printf("Invalid address or value\n");

return 0;

}

}

else{

printf("Invalid command\n");

return 0;

}

}

else if(whatcom == 3){

if(commacount == 2){

comcheck = fill(a, b, c);

if(comcheck == -1){

printf("Invalid address or value\n");

return 0;

}

if(comcheck == -2){

printf("Start is larger than end\n");

return 0;

}

}

else{

printf("Invalid command\n");

return 0;

}

}

else{

comcheck = opmn(numptr, comlen-1, a);

if(comcheck == -1){

printf("There is no corresponding mnemonic\n");

return 0;

}

}

return 1;

}

void putnode(){

struct node \*new;

new = (struct node\*)malloc(sizeof(struct node));

strcpy(new->command, tmp);

new->next = NULL;

tail->next = new;

tail = tail->next;

}

void puthashnode(int opvalue, int mnvalue){

struct hashnode \*new;

new = (struct hashnode\*)malloc(sizeof(struct hashnode));

strcpy(new->mn, mntmp);

new->op = opvalue;

if(hashtable[mnvalue] == NULL){

new->next = NULL;

hashtable[mnvalue] = new;

}

else{

new->next = hashtable[mnvalue];

hashtable[mnvalue] = new;

}

}