**과목명: 시스템프로그래밍**

**1 분반**

**<<Project #1>>**

**서강대학교 유럽문화학과**

**20170301**

**황선애**

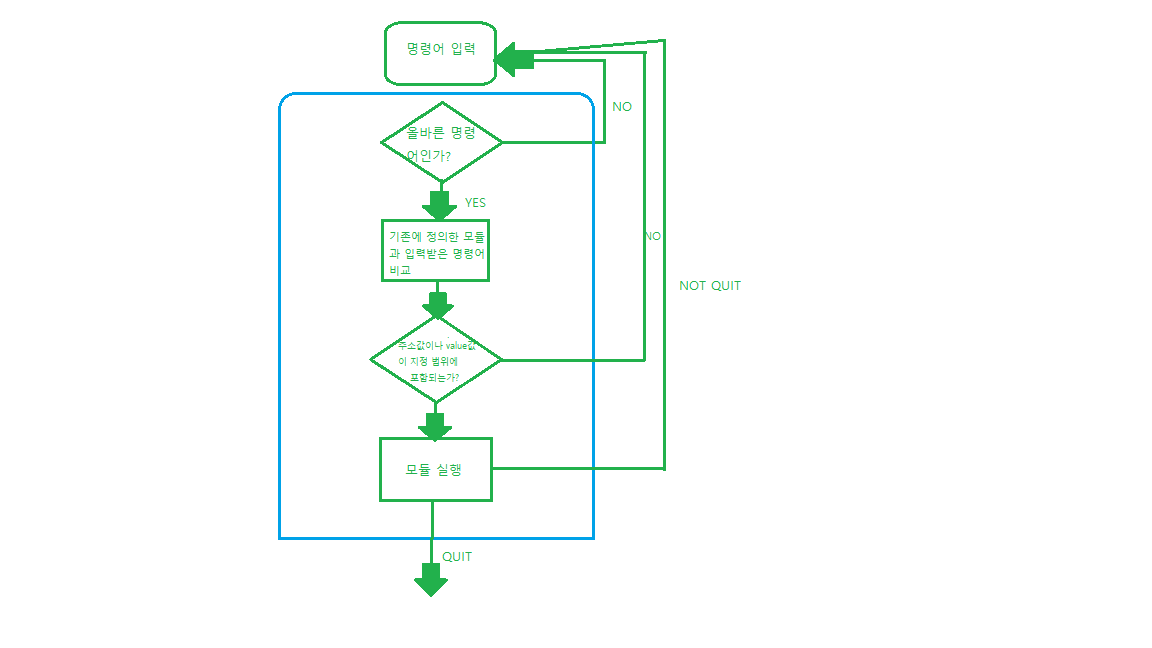
목 차

1. **프로그램 개요**
2. **프로그램 설명**
   1. 프로그램 흐름도
3. **모듈 정의**
   1. 정의한 모듈에 대해 간략적인 설명
4. **전역 변수 정의**
5. **코드 설명**
6. **프로그램 개요**

이 프로그램은 앞으로 구현하게 될 SIC/XE머신을 구현하기 위한 전 단계로서 어셈블러, 링크, 로더들을 실행하게 될 셸(shell)과 컴파일을 통해서 만들어진 object코드가 적재 되고 실행될 메모리공간과 mnemonic (ADD, COMP, FLOAT, etc …)을 opcode값으로 변환하 는 OPCODE 테이블과 관련 명령어들을 구현하는 프로그램입니다.

1. **프로그램 설명**

**2.1 프로그램 흐름도**

****

프로그램을 실행시키면 sicsim> 이라는 입력프롬프트가 뜬다. 사용자가 명령어를 입력하면, main 함수에서 해당 명령어의 철자(잘못된 입력 예시 “Hepl”) 혹은 입력 방식(잘못된 입력 예시 “edit 4”)을 검사한다. 만약, 잘못되었다면 **아무 출력 없이continue문으로 다시 입력을 받는다. history에도 입력이 되지 않는다.**

올바른 입력이라면 해당 명령어를 수행하는 모듈을 호출한다. 형식만 올바르고 입력값이 알맞게 입력되지 않은 경우(ex. fill 24,34,fff) 해당 모듈을 곧바로 종료시키고 main함수로 돌아가 **아무 출력 없이 또 다시 입력을 받게 한다. history에도 입력이 되지 않는다.**

사용자가 “quit” 입력하면 해당 프로그램은 while문에서 빠져나와 종료된다.

**3 모듈 정의**

**3.1 Shell 관련 명령어를 수행하기 위한 모듈**

**3.1.1 int help(void);**

쉘에서 실행 가능한 모든 명령어들의 리스트를 화면에 출력해준다.

**3.1.2 int dir(void);**

현재 디렉터리에 있는 파일들을 출력하되, 실행 파일은 이름 옆에 ‘\*’표시를, 디렉터리는 ‘/’표시를 한다.

**3.1.3 int history(void);**

현재까지 사용한 명령어들을 순서대로 번호와 함께 보여준다.

**3.2 메모리 관련 명령어를 수행하기 위한 모듈**

**3.2.1 int dump(void);**

dump의 실행으로 출력된 마지막 번지 바로 다음부터 10라인씩 메모리의 주소, 메모리에 저장된 내용을 16진수 형태와 아스키 코드 형태로 보여준다. 만약, 처음 dump 명령이 실행되는 것이라면 0번지부터 10개의 행(한 라인은 16개의 16진수로 구성되어 있음)을 화면에 출력한다.

**3.2.2 int dump\_start(void);**

3.2.1의 모듈과 동일한 동작을 수행하지만, 시작 번지수가 사용자에 의해 지정된다.

**3.2.3 int dump\_startend(int start, int end);**

3.2.1의 모듈과 동일한 동작을 수행하지만, 시작 번지수와 끝 번지수가 사용자에 의해 지정된다.

**3.2.4 int edit(int address, int value);**

메모리의 address 번지의 값을 value에 지정된 값으로 변경한다. 주소값이 메모리를 벗어나는 경우, value값이 범위를 벗어나는 경우 에러메시지를 출력한다.

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

사용자가 지정한 start 번지부터 end 번지까지의 값을 value에 지정된 값으로 변경한다. 주소값이 메모리 범위를 벗어나는 경우, value값이 범위를 벗어나는 경우 에러메시지를 출력한다.

**3.3 OPCODE TABLE 관련 명령어를 수행하기 위한 모듈**

**3.3.1 int makeTable(void);**

main함수가 실행될 때 수행되는 모듈로 text file을 열고, 하나의 라인씩 읽은 후 해시테이블을 만들기 위한 모듈인 insert\_hash를 호출한다.

**3.3.2 int findOpcode(char\* mnemonic);**

입력된 명령어에 대응하는 opcode를 해시테이블에서 찾아 16진수 형태로 출력한다.

**3.3.3 int showHt(void);**

해시테이블의 내용을 출력한다.

**3.4 위의 모듈 수행에 필요한 부가적인 모듈**

**3.4.1 int insert(char\* string);**

history 모듈을 위해 작성된 모듈로, 명령어가 올바르게 입력될 때마다 입력순서와 입력내용을 저장한다.

**3.4.2 int h(char\* mnemonic);**

해시테이블을 만들기 위해 작성된 제산함수이다.

**3.4.3 int insert\_hash(int hb, bucket\* newNode)**

opcode.txt를 읽고, 해시함수를 이용해 명령어와 opcode 정보가 담긴 체이닝 해시테이블을 만드는 역할을 한다.

**4 전역 변수 정의**

**nodePointer first**

// 구조체 node를 가리키는 포인터 변수로서, history와 insert 모듈에서 사용된다. 가장 먼저 insert된 명령어에 대한 정보를 갖고 있다.

**nodePointer worker**

// 구조체 node를 가리키는 포인터 변수로서, history 함수가 수행될 때 first부터 시작하여 마지막으로 입력된 명령어까지 link를 타고 이동하면서 가리키는 노드를 바꿈으로써 각각의 노드가 저장하고 있는 명령어 정보와 순서를 출력할 수 있게끔 한다.

**char memory[row][col]**

// 1Mbyte(16\*65536) 크기의 가상 메모리 공간이다.

**int address**

// 메모리 관련 명령어를 수행하는 모듈에서 사용되는 변수로, dump의 실행으로 출력된 마지막 (address+1) 번지를 기억한다.

**int insert\_idx**

// insert 함수가 호출될 때마다 (즉, 새로운 명령어가 입력될 때마다) 증가하면서 명령어의 입력순서를 저장한다.

**bucket\* ht[20]**

// 링크드리스트 기반의 해시테이블에서, 키 값 0부터 19까지 대응하는 20개의 체인에 대한 정보를 저장하고 있다. ht[0]은 키값이 0인 버킷들(체인) 중 가장 처음 원소에 대한 정보를 갖고 있다.

**5 코드 설명**

**20170301.h**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <dirent.h>

#include <sys/stat.h>

#define col 16

#define row 65536

typedef struct node\* nodePointer;

typedef struct node{

int key;

int number;

char\* str;

nodePointer link;

} node;

typedef struct bucket\* bucketPtr;

typedef struct bucket{

char\* key;

char\* format;

int opcode;

bucketPtr link;

} bucket;

// 전역 변수 선언

nodePointer first=NULL; // 가장 먼저 호출된 명령어 가리킴

nodePointer worker=NULL; // 움직이면서 링크드리스트를 이어줌.

char memory[row][col]; // 가상 메모리

int address; // 메모리 번지수

int insert\_idx=1; //insert할때마다 증가할 인덱스

bucket\* ht[20];

// 함수 정의

int help(void);

int insert(char\* string);

int history(void);

int dir(void);

int dump(void);

int dump\_start(int start);

int dump\_startend(int start, int end);

int edit(int address, int value);

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

int h(char\* mnemonic);

int makeTable(void);

int insert\_hash(int hb,bucket\* newNode);

int findOpcode(char\* mnemonic);

int showHt(void);

**20170301.c**

#include "20170301.h"

int main(){

char command[82];

char\* processed[5]; // 쪼개진 명령어가 저장될 포인터 배열

char\* original;

int i;

int check=-1; // 명령어가 올바르게 입력되었는지 체크

int check\_hash=-1; // 해시테이블의 중복생성을 막기 위한 변수

while(1){

check=-1;

if (check\_hash==-1) check\_hash=makeTable(); // 해쉬 테이블 생성

printf("sicsim>");

fgets(command,82,stdin);

original=malloc(sizeof(char)\*82);

strcpy(original,command); // 명령어를 다른 배열에 저장해놓음

// 입력받은 command를 가공하는 부분

processed[0]=strtok(command," \n"); // 명령어는 indent 혹은 개행으로 구분

for (i=1;i<5;i++){

processed[i]=strtok(NULL,", \n"); // value는 반점,indent, 개행으로 구분

}

// \*\*\*여기서부터 shell 관련 명령어\*\*\*

//help

if (!strcmp(processed[0],"h")||!strcmp(processed[0],"help")&&!processed[1]&&!processed[2]&&!processed[3])

{

help();

insert(original);

}

//history

else if (!strcmp(processed[0],"hi")||!strcmp(processed[0],"history")&&!processed[1]&&!processed[2]&&!processed[3])

{

insert(original);

history();

}

//quit

else if (!strcmp(processed[0],"q")||!strcmp(processed[0],"quit")&&!processed[1]&&!processed[2]&&!processed[3])

{

return 0;

}

//dir

else if (!strcmp(processed[0],"d")||!strcmp(processed[0],"dir")&&!processed[1]&&!processed[2]&&!processed[3])

{

dir();

insert(original);

}

// \*\*\*여기서부터 memory 관련 명령어\*\*\*

// dump[[1]](#footnote-1)

else if (!strcmp(processed[0],"du")||!strcmp(processed[0],"dump")){

if (!processed[1]){ // dump only

dump();

insert(original);

}

else if (!processed[2]){ // dump start

int dec\_s=strtol(processed[1],NULL,16);[[2]](#footnote-2)

check=dump\_start(dec\_s);

if (!check) insert(original);[[3]](#footnote-3)

}

else if (!processed[3]){ //dump start,end

int dec\_s=strtol(processed[1],NULL,16);

int dec\_e=strtol(processed[2],NULL,16);

check=dump\_startend(dec\_s,dec\_e);

if (!check) insert(original);

}

else continue;

}

// edit

else if (!strcmp(processed[0],"edit")||!strcmp(processed[0],"e")&&processed[1]&&processed[2]&&!processed[3]) {

int dec=strtol(processed[1],NULL,16);

int dec\_v=strtol(processed[2],NULL,16);

check=edit(dec,dec\_v);

if(!check) insert(original);

}

else if ((!strcmp(processed[0],"fill")||!strcmp(processed[0],"f"))&&processed[1]&&processed[2]&&processed[3]&&!processed[4])

{

int dec\_s=strtol(processed[1],NULL,16);

int dec\_e=strtol(processed[2],NULL,16);

int dec\_v=strtol(processed[3],NULL,16);

check=fill(dec\_s,dec\_e,dec\_v);

if(!check) insert(original);

}

else if (!strcmp(processed[0],"reset")&&!processed[1]&&!processed[2]&&!processed[3]){

memset(memory,0,sizeof(char)\*col\*row); // 메모리 모두 초기화

insert(original);

}

else if (!strcmp(processed[0],"opcode")&&processed[1]&&!processed[2]){

check=findOpcode(processed[1]);

if (!check) insert(original); // 값을 찾은 경우에만 history에 들어가게

}

else if (!strcmp(processed[0],"opcodelist")&&!processed[1]){

insert(original);

showHt();

}

else continue; // 정의되지 않은 명령어

}

}

int 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");

return 0;

};

int history(){

for (worker=first;worker->link!=NULL;worker=worker->link){

printf("%d %s\n",worker->number,worker->str);

}

return 0;

}

int insert(char\* string){

strtok(string,"\n");

nodePointer newNode=(nodePointer)malloc(sizeof(node));

if (!newNode) return 0; // memory allocation failed.

newNode->str=string;

newNode->number=insert\_idx++;

newNode->link=NULL;

if (!first){ // 처음 삽입일 때만 작동

first=newNode;

worker=first;

}

else {

worker->link=newNode;

worker=worker->link;

}

return 0;

}

int dir(){

DIR \*dir\_ptr=NULL;

struct dirent \*file=NULL;

struct stat buf;

if((dir\_ptr=opendir("."))==NULL) return 0; // file open error

while ((file=readdir(dir\_ptr))!=NULL){

stat(file->d\_name,&buf); // 파일의 속성이 buf에 저장됨

if (S\_ISDIR(buf.st\_mode)) // 여기서 쓰는 함수는 매크로

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

else if ((S\_IEXEC & buf.st\_mode)!=0) // st\_mode값과 and연산으로 소유자의 실행 권한 확인

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

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

}

printf("\n");

closedir(dir\_ptr);

return 0;

}

int dump(){

if (address+159>0xfffff) return dump\_startend(address,0xfffff);

else return dump\_startend(address,address+159);

}

int dump\_start(int start){

if (start+159>0xfffff) return dump\_startend(start,0xfffff);

else return dump\_startend(start,start+159);

}

int dump\_startend(int start, int end){

// 번지수를 10진수로 변환 후 16으로 나눈 몫은 memory의 행, 나눈 나머지는 열

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

printf("Error : Invalid input.\n");

return -1;

}

int r=start/16; // will be the row of memory[][]

int c=start%16; // will be the col of memory[][]

int num=end-start+1; // 출력할 항의 총 개수(decimal)

int i,j,l; //i는 출력되는 항의 수를 카운트, j는 하나의 행당 16개씩 출력하기위한 인덱스, l은 ASCII 컬럼에서 쓰일 인덱스

for (i=0;i<num;){

printf("%04X0 ",r); // address field

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

if (i==0&&j<c) printf(" "); // c열 이전의 열들은 출력 x

else if (i>=num) break;

else {

printf("%02X ",memory[r][j]); // memory field

i++;

}

}

for (j;j<16;j++) printf(" ");

// ASCII field

printf(" ; ");

for(l=0;l<16;l++){

if (i==0&&l<c) printf("."); // c열 이전의 행은 . 으로 출력

else {

if(memory[r][l]<0x20||memory[r][l]>0x7e) printf(".");

else printf("%c",memory[r][l]);

}

}

r=r+1;

printf("\n");

}

// address 갱신

if (end==0xfffff) address=0;

else address=end+1;

return 0;

}

int edit(int address, int value){

if (address>0xfffff||address<0) {

printf("Error : Invalid address\n");

return -1;

}

if (value<0||value>0xff) {

printf("Error : Invalid value\n");

return -1;

}

int r=address/16;

int c=address%16;

memory[r][c]=value;

return 0;

}

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

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

printf("Error : Invalid address\n");

return -1;

}

if (value<0||value>0xff){

printf("Error : Invalid value\n");

return -1;

}

int r=start/16;

int c=start%16;

int num=end-start+1;

int i=r,j=c;

// start-end+1 개의 memory의 값을 수정해나감.

while(num>0){

if(j==16){ j=0; i++;} // 새로운 행으로..

memory[i][j++]=value; // edit the value of memory

num--;

}

return 0;

}

int h(char\* mnemonic){ //해시함수로, 나머지연산(%)을 이용.

int sum=0;

int i=0;

int total=strlen(mnemonic);

for (i=0;i<total;i++) sum+=mnemonic[i];

return sum%20;

}

int insert\_hash(int hb, bucket\* newNode){

if(!ht[hb]) ht[hb]=newNode; // 해당 버킷에 처음 삽입된 경우

else{

newNode->link=ht[hb];

ht[hb]=newNode;

}

return 0;

}

int makeTable(){

char mnemonic[10];

char fmt[10];

int op;

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

if (fp==NULL){

printf("file open error\n");

return -1;

}

while (fscanf(fp,"%x %s %s\n",&op,mnemonic,fmt)!=EOF){

bucket\* newNode=(bucket\*)malloc(sizeof(bucket)); // ht에 삽입될 노드 생성

newNode->key=malloc(sizeof(char)\*10);

strcpy(newNode->key,mnemonic);

newNode->opcode=op;

newNode->link=NULL;

int hb=h(newNode->key); // newNode의 버킷 주소 생성

insert\_hash(hb,newNode); // 한 줄마다 insert\_hash 모듈을 콜함.

}

fclose(fp);

return 0;

}

int findOpcode(char\* mnemonic){

int flag=0;

bucketPtr worker=ht[h(mnemonic)]; // 동일한 해시값을 가진 체인의 가장 첫번째 원소를 가리킴.

while(worker){

if (!strcmp(worker->key,mnemonic)){ // 동일한 명령어를 찾은 경우 break.

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

flag=1;

break;

}

worker=worker->link;

}

if (flag==1) return 0; // history에 insert 여부를 위한 플래그 변수.

else return -1; // not insert.

}

int showHt(void){ // 해시 테이블의 형식과 내용을 출력.

bucketPtr worker;

int i;

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

printf("%d : ",i);

if (!ht[i]){ printf("\n"); continue;}

for(worker=ht[i];worker->link!=NULL;worker=worker->link){

printf("[%s,%02X] -> ",worker->key,worker->opcode);

}

printf("[%s,%02X]",worker->key,worker->opcode);

printf("\n");

}

return 0;

}

1. Dump 명령어는 dump만 들어오는 경우, dump와 start가 들어오는 경우, dump와 start, end값 모두가 매개변수인 경우가 있어 매개변수의 수에 따라 3개의 함수를 만들었습니다 (하지만, dump와 dump\_start 모듈 모두 동작은 결국 dump start,end와 똑같이 하므로 전자 2개의 함수 내에서 dump\_startend를 호출하도록 하였습니다.) [↑](#footnote-ref-1)
2. strtol ; 16진수 형태의 문자열을 10진수로 변환 e.g. “1D” -> 29 [↑](#footnote-ref-2)
3. Address와 value 모두 범위 안에 들어온 올바른 명령어라면, insert 수행. [↑](#footnote-ref-3)