과목 명: 시스템프로그래밍

담당 교수 명: 박 운 상

<<Assignment 1>>

**서강대학교 컴퓨터학과**

**[20141603]**

**[황 순]**

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: char\* rtrim(char\* s) 3

3.2.1 기능 3

3.2.2 사용 변수 3

3.3 모듈이름: char\* ltrim(char\* s) 4

3.3.1 기능 4

3.3.2 사용변수 4

3.4 모듈이름: char\* trim(char\* s) 4

3.4.1 기능 4

3.4.2 사용변수 4

3.5 모듈이름: int FindValidDumpInput(char \*str); 4

3.5.1 기능 4

3.5.2 사용변수 4

3.6 모듈이름: void help(); 4

3.6.1 기능 4

3.6.2 사용변수 4

3.7 모듈이름: void dir(); 4

3.7.1 기능 4

3.7.2 사용변수 4

3.8 모듈이름: int FindValidDumpInput(char \*str); 4

3.8.1 기능 4

3.8.2 사용변수 4

3.9 모듈이름: int FindValidDumpInput(char \*str); 4

3.9.1 기능 4

3.9.2 사용변수 4

3.10 모듈이름: void dir(); 4

3.10.1 기능 4

3.10.2 사용변수 4

3.11 모듈이름: void MakeHistory(Node \*\*hs, char \*str); 4

3.11.1 기능 4

3.11.2 사용변수 4

3.12 모듈이름: void PrintHistory(Node \*hs, int cnt); 4

3.12.1 기능 4

3.12.2 사용변수 4

3.13 모듈이름: int Dump(unsigned char \*\*Memory, char \*str, int spoint, int end); 4

3.13.1 기능 4

3.13.2 사용변수 4

3.14 모듈이름: void EditMemory(unsigned char \*\*Memory, int address, int value); 4

3.14.1 기능 4

3.14.2 사용변수 4

3.15 모듈이름: void fill(unsigned char \*\*Memory, int startA, int endA, int value); 4

3.15.1 기능 4

3.15.2 사용변수 4

3.16 모듈이름: int hash\_function(char \*s); 4

3.16.1 기능 4

3.16.2 사용변수 4

3.17 모듈이름: void OpcodeInput(hashtable \*\*ht); 4

3.17.1 기능 4

3.17.2 사용변수 4

3.18 모듈이름: void Opcodelist\_print(hashtable\*\* ht); 4

3.18.1 기능 4

3.18.2 사용변수 4

3.19 모듈이름: int FindOpPrint(char \*command,hashtable \*\*ht); 4

3.19.1 기능 4

3.19.2 사용변수 4

4. 전역 변수 정의 4

5. 코드

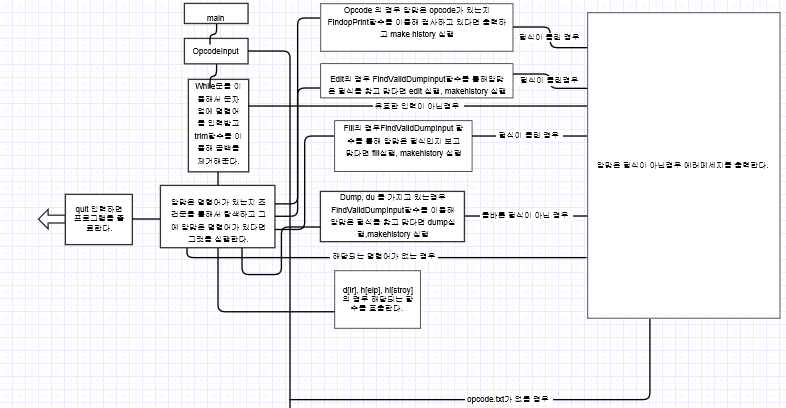
# 프로그램 개요

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

\

# 프로그램 설명

## 프로그램 흐름도



# 모듈 정의

## 모듈 이름 : main()

### 기능

Opcodeinput(ht) 함수를 통해서 opcode 정보를 받아오고, while문이 실행 되어 문자열을 입력받아 그 문자열에 해당되는 함수를 찾아서 기능을 실행하고, 해당되는 함수가 없을 시에는 에러메세지를 출력한다. ‘quit’ 명령어를 받을 때까지 계속 입력을 받는다.

### 사용 변수

int i, j;

int len; // 문자열의 길이표현

int cnt = 0; // history를 위한 count

int dstart=0; // dump의 시작점을 의미

int epoint; // 끝점

int spoint; //edit나 fill에서 쓰일 시작점

int addr; //for edit

int value; // for edit and fill

int startA, endA; // address for fill

int opflag; // flag for the opcode -> if true return 1, else return 0

char \*command; // char string to put input without blanks

char \*input; // input string

Node \*hs; //Node for histroy

int flag=0;

unsigned char \*\*Memory; //Memory

char junk[45]; // use in sscanf for the mneomonics

char stp[45]; // use in sscanf for the first string

char edp[45]; // use in sscanf for the middle string in fill, end string in edit and dump

char valp[45]; // use in sscanf for the end string in fill

hashtable \*ht[20]; // hash for the mneomonics

## 모듈 이름: Char \*rtrim(char \*s)

### 기능

Isspace를 이용해서 공백문자를 검색하고 이를 통해 오른쪽에 존재하는 공백문자들을 없애서 문자열의 오른쪽의 공백문자를 제거해주는 함수이다

### 사용 변수

char t[90]; // s를 저장해서 공백을 없앨 때 쓸 배열

char \*end; // 공백 부분을 가리키는 포인터

## 모듈이름: Char \*ltrim(char \*s)

### 기능

Isspace를 이용해서 공백문자를 검색하고 이를 통해 왼쪽에 존재하는 공백문자들을 없애서 문자열의 오른쪽의 공백문자를 제거해주는 함수이다

### 사용변수

Char \*begin // 공백문자를 탐색할 때 쓸 문자 포인터

## 모듈이름: Char \*trim(char \*s)

### 기능

Ltrim과 rtrim을 이용해서 문자열 양쪽의 공백을 제거해준다.

### 사용변수

없음

## 모듈이름: int FindValidDumpInput(char \*str)

### 기능

Bflag 와 dflag를 이용해 이것이 dump,edit,fill 세 함수에 알맞은 입력인지 아닌지를 판단해서 리턴해준다. 만약 bflag가 -1이면 올바르지 않은 서식의 입력이므로 -1을 리턴해주고, bflag가 -1이 아닌경우 문자열 검사를 통해서 이것이 dump , fill, edit 각 함수에 대하여 알맞은 서식인지 검사 후 , dflag값과 같은 값을 리턴한다. dflag값은 콤마가 없다면 1, 하나라면 2, 두개라면 3이 된다.

### 사용변수

int i; // 반복문에 쓰일 변수

int dflag = 0; // flag for the finding valid input

int len;

int lens, lene,leni; // 문자열을 체크할떄 쓰일 문자열 길이를 받기 위한 변수들

int bflag = 0; // flag for the blank between the hexadecimal numbers

char start[30];

char end[30];

char imm[30];

char junk[30];

// strings for sscanf

## 모듈이름: void help();

### 기능

존재하는 명령어들을 모두 프린트해준다. 소문자 h 나 help를 입력하면 실행된다

### 사용변수

없음

## 모듈이름: void dir();

### 기능

폴더 내에 있는 디렉터리와 파일들을 출력해준다.

### 사용변수

### DIR \*di = NULL; // 폴더 내부에서 정보를 알아내기 위해 쓰일 변수

struct dirent \*din = NULL; // 폴더 내 파일의 이름을 알아낼 때 사용한다.

struct stat dimo; // 폴더 안의 파일이 실행파일인지 디렉터리인지 검사할떄 쓰는 변수

## 모듈이름: void Makehistory(Node \*\*hs,char \*str)

### 기능

History 출력을 위해서 명령어를 저장할 노드를 생성하고, 만약 hs가 비었다면 바로 연결해주고, 다른 노드가 존재하는 경우라면 가장 끝노드의 link에 연결해준다.

### 사용변수

Node \*temp; // 새로운 history node를 삽입할 때 쓰일 임시노드

Node \*tmp; // hs를 돌면서 temp노드가 붙을 곳을 찾아줄 임시노드

## 모듈이름: void PrintHistory(Node \*hs,int cnt)

### 기능

Hs 노드에 존재하는 명령어들, 즉 전에 사용된 명령어들을 출력해준다. 전달받은 cnt를 통해 몇번째 명령어인지 알려준다.

### 사용변수

Node \*temp; // hs를 돌면서 마지막 노드를 찾을 때 까지 쓰일 임시 노드

## 모듈이름: int Dump(unsigned char \*\*Memory,char \* str, int spoint, int end);

### 기능

주어진 포맷에 따라 시작점부터 끝점 까지의 메모리 공간을 출력해주는 함수이다. 만약 dump 나 dump %x 와 같은 형식이라면 무조건 10라인을 출력한다. 이때 end == -1 이 되어 무조건 10라인을 출력할 수 있게 하고 dump %x,%x와 구분되게 한다. 시작점과 끝점의 범위 오류에 대해서 에러 메시지를 출력한다.

메모리 공간의 경우 32~126의 값을 가진 메모리만 아스키 코드가 출력되고 출력메모리의 범위 밖이거나 126초과, 32 미만의 값을 가지는 경우는 . 만 표시한다.

### 사용변수

int memv; // variable to show memory address

int i, j; // 반복문에 쓰일 변수

int srow, erow, scol,ecol; //variables for the row and col number of the startpoint and the endpoint

int epoint; // 끝점 표현에 쓰일 변수

int enddot; // 반복문의 마지막 줄이나 시작줄에서 끝 col을 표현할 변수

int startid = 0; // 이번 dump 이후 다음 시작점을 나타낼 변수. 리턴될 값

## 모듈이름: void EditMemory(unsigned char \*\*Memory,int value,int address);

### 기능

주소 값을 받아서 그곳의 메모리 값을 value로 바꾼다. value값이 0보다 작거나 255보다 큰 경우, 주소값이 메모리 공간을 초과하는 경우 등에는 에러메세지를 출력해준다.

### 사용변수

int Tarrow, Tarcol; //Target address's row and col

## 모듈이름: void fill(unsigned char \*\*Memory,int startA,int endA,int value);

### 기능

주어진 시작점부터 끝점까지 모든 메모리의 값들을 value로 바꿔 준다. 만약 시작점이 끝점보다 크거나, 시작점이나 끝점의 범위가 메모리 공간을 초과하거나 value 값이 0보다 작거나 255보다 큰 경우에는 에러메세지를 출력해준다.

### 사용변수

int srow, scol; // 시작점의 행번호와 열번호

int erow, ecol; // 끝점의 행번호와 열번호

int enddot; // 반복문의 마지막 줄이나 시작줄에서 끝 col을 표현할 변수

int i, j; // 반복문에 쓰일 변수

## 모듈이름: int hash\_function(char\* s);

### 기능

명령어(mnemonic)을 매개변수로 받아 hash table의 hash buckets의 인덱스를 구해주는 함수이다.

### 사용변수

int i, sum;

## 모듈이름: void OpcodeInput(hashtable \*\*ht)

### 기능

파일 스트림으로부터 opcode.txt.를 읽어오고, 그에 대한 정보를 해쉬테이블에 저장해주는 함수이다.

Opcode.txt가 없는경우에는 에러메세지를 출력한다.

### 사용변수

FILE \*fp // opcode.txt.를 받아올 파일스트림

char tmpmneo[20]; // 명령어 를 받아올 임시배열

char tmpname[20]; // 맨뒤를 받아올 임시배열

char tmpvalue[20]; // 16진수를 받아올 임시배열

hashtable \*temp, \*head; // 해쉬테이블 연결과 접근을 위한 변수

int i; // 반복문을 위한 변수

int idx; // 해쉬테이블 인덱스를 표현할 변수

## 모듈이름: void Opcodelist\_print(hashtable\* ht)

### 기능

Opcodelist에 존재하는 모든 정보들을 표현한 해쉬테이블을 프린트 해준다.

### 사용변수

hashtable\* temp;//pointer for searching hash table

int i;//index

## 모듈이름: int FindOpPrint(char\* command, hashtable\*\* ht)

### 기능

입력받은 문자열에서 opcode에 해당되는 이름이 있다면, 그 값을 찾아서 출력해준다. 없는 경우 없다고 메시지를 출력해준다.

### 사용변수

int index;//index of hash table

char tmpname[100];//string of inputing opcode

char tmpmeno[100];//string of inputing mnemonic

int flag = 0;

hashtable\* temp;//pointer for searching hash table

# 전역 변수 정의

사용하지 않았음.

# 코드

20141603.h

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <ctype.h>

#include<sys/stat.h>

#include<dirent.h>

typedef struct node{

int cnt;

char name[90];

struct node \*link;

}Node;

typedef struct hash // hash table 공간

{

char mneo[100];

int value;

char name[100];

struct hash \*next;

}hashtable;

char\* rtrim(char\* s);

char\* ltrim(char \*s);

char\* trim(char \*s);

int FindValidDumpInput(char \*str);

void help();

void dir();

void MakeHistory(Node \*\*hs, char \*str);

void PrintHistory(Node \*hs, int cnt);

int Dump(unsigned char \*\*Memory, char \*str, int spoint, int end);

void EditMemory(unsigned char \*\*Memory, int address, int value);

void fill(unsigned char \*\*Memory, int startA, int endA, int value);

int hash\_function(char \*s);

void OpcodeInput(hashtable \*\*ht);

void Opcodelist\_print(hashtable\*\* ht);

int FindOpPrint(char \*command,hashtable \*\*ht);

20141603.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <sys/stat.h>

#include "20141603.h"

char\* rtrim(char\* s) {

char t[90];

char \*end;

strcpy(t, s);

end = t + strlen(t) - 1;

while (end != t && isspace(\*end))

end--;

\*(end + 1) = '\0';

s = t;

return s;

}

//terminate blank in the right

char\* ltrim(char \*s) {

char\* begin;

begin = s;

while (\*begin != '\0') {

if (isspace(\*begin))

begin++;

else {

s = begin;

break;

}

}

return s;

}

// terminate blank in the left

char\* trim(char \*s) {

return rtrim(ltrim(s));

}

// terminate blank in the left and right

int FindValidDumpInput(char \*str)

{

int i;

int dflag = 0; // flag for the finding valid input

int len;

int lens, lene,leni;

int bflag = 0; // flag for the blank between the hexadecimal numbers

char start[30];

char end[30];

char imm[30];

char junk[30];

// strings for sscanf

sscanf(str, "%s %[^\n]", junk, start);

strcpy(start, trim(start));

len = strlen(start);

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

{

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

dflag = 3;

else

dflag = 2;

}

// if there is ',' , dflag =3 , if not dflag = 2

if (dflag == 2)

{

lens = strlen(start);

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

{

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

continue; // if the character is number or alphabet a to f, A to F

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

{

bflag = -1;

break;

} //if there is blank between the hexadeciaml numbers like 2 3, make the bflag = -1

else

{

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

{

dflag = 3;

break;

}

else

bflag = -1;

break;

}

}

if (bflag == -1)

{

return -1;

}

}

// for the dump start and case that dump start, which is not valid Input

if (dflag == 3)

{

sscanf(str, "%s %[^,] %\*c %[^\n]", junk, start, end);

lens = strlen(start);

lene = strlen(end);

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

{

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

continue; // if the character is number or alphabet a to f, A to F

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

{

bflag = -1;

break;

} //if there is blank between the hexadeciaml numbers like 2 3, make the bflag = -1

else

{

bflag = -1;

break;

}

}

if (bflag == -1)

{

return -1;

}

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

{

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

continue; // if the character is number or alphabet a to f, A to F

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

{

bflag = -1;

break;

} //if there is blank between the hexadeciaml numbers like 2 3, make the bflag = -1

else

{

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

{

dflag = 4; // in case of fill

break;

}

bflag = -1;

break;

}

}

if (bflag == -1)

{

return -1;

}

} //for the dump start,end and edit

if (dflag == 4)

{

sscanf(str, "%s %[^,] %\*c %[^,] %\*c %[^\n]", junk, start, end, imm);

lens = strlen(start);

lene = strlen(end);

leni = strlen(imm);

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

{

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

continue; // if the character is number or alphabet a to f, A to F

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

{

bflag = -1;

break;

} //if there is blank between the hexadeciaml numbers like 2 3, make the bflag = -1

else

{

bflag = -1;

break;

}

}

if (bflag == -1)

{

return -1;

}

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

{

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

continue; // if the character is number or alphabet a to f, A to F

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

{

bflag = -1;

break;

} //if there is blank between the hexadeciaml numbers like 2 3, make the bflag = -1

else

{

bflag = -1;

break;

}

}

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

{

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

continue; // if the character is number or alphabet a to f, A to F

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

{

bflag = -1;

break;

} //if there is blank between the hexadeciaml numbers like 2 3, make the bflag = -1

else

{

bflag = -1;

break;

}

}

if (bflag == -1)

{

return -1;

}

else

return 4;

}

else if (dflag == 2)

return 2;

else if (dflag == 3)

return 3;

else

return -1;

}

// find the valid input and if it is valid, return proper flag . if not, return -1

void help()

{

printf("h[elp]\n");

printf("d[ir]\n");

printf("q[uit]\n");

printf("hi[story]\n");

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

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

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

printf("reset\n");

printf("opcode mnemonic\n");

printf("opcodelist\n");

}

// function for printing the available commands

void dir()

{

DIR \*di = NULL;

struct dirent \*din = NULL;

struct stat dimo;

di = opendir(".");

for (;;)

{

din = readdir(di); // to the opendir one, read the first one, after we recall the readdir, it will read next one each time.

if (din == NULL)break;

lstat(din->d\_name, &dimo); //returns information about the link

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

if (S\_ISDIR(dimo.st\_mode))printf("/"); // distinguish whether it is a directory file . if directroy put /

else if (dimo.st\_mode & S\_IXUSR)printf("\*"); // if file put \*

printf("\t");

}

printf("\n");

}

// function for printing existing file in the directory

void MakeHistory(Node \*\*hs, char \*str)

{

Node \*temp;

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

strcpy(temp->name, str);

temp->link = NULL;

Node\* tmp;

if ((\*hs) == NULL)

(\*hs) = temp;

// if history node is empty, make the temp node directly link to it

else {

for (tmp = (\*hs); tmp->link != NULL; tmp = tmp->link);

tmp->link = temp;

}

// if there is any existing node, put temp node at the end

}

//function for make the array of the effective command by using linked list

void PrintHistory(Node \*hs, int cnt)

{

Node \*temp;

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

for (temp = hs->link; temp != NULL; temp = temp->link) {

printf("%d : %s\n", cnt, temp->name);

cnt++;

}

free(temp);

}

// function for printing the commands that is in the history node

int Dump(unsigned char \*\*Memory,char \*str,int spoint,int end)

{

int memv; // variable to show memory address

int i, j;

int srow, erow, scol,ecol; //variables for the row and col number of the startpoint and the endpoint

int epoint;

int enddot;

int startid = 0;

if (spoint < 0 || spoint > 1048575)

{

printf("This is not available range!\n");

return -1;

}

// if the startpoint is not in the available range, print the error message

else

{

if (end == -1)

{

epoint = spoint + 159;

} // in case of dump, dump %x

else

epoint = end; // in case of dump %x,%x

if (end > 1048575) {

printf("This is not available range!\n");

return -1;

}

// if the endpoint is bigger than existing, print error message

else {

startid = epoint + 1; // next start will be epoint+1

srow = spoint / 16;

scol = spoint % 16;

if (epoint > 1048575)

epoint = 1048575;

erow = epoint / 16;

ecol = epoint % 16;

// find the col and row of the startpoint, also endpoint

memv = srow \* 16;

for (i = srow; i < erow + 1; i++)

{

if (i == srow)

{

if (srow == erow)

enddot = ecol; // if the srow and erow are the same, enddot will be ecol

else

enddot = 15; // if srow!=erow , enddot should be the end of the line

printf("%05X ", memv);

memv += 16;

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

printf(" ");

for (j = scol; j < enddot + 1; j++)

{

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

}

printf(" ;");

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

printf(".");

for (j = scol; j < enddot+1; j++)

{

if (32 <= Memory[i][j] && Memory[i][j] <= 126)

printf("%c", Memory[i][j]);

else

printf(".");

} // if it is in the range, print the character if it is in available range

for (j = enddot + 1; j < 16;j++)

printf(".");

// if it is not in the range , print it '.'

}

else if (i == erow)

{

printf("%05X ", memv);

memv += 16;

for (j = 0; j < ecol + 1; j++)

{

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

}

for (j = ecol + 1; j < 16; j++)

printf(" ");

printf(" ;");

for (j = 0; j < ecol + 1; j++)

{

if (32 <= Memory[i][j] && Memory[i][j] <= 126)

printf("%c", Memory[i][j]);

else

printf(".");

} // if it is in the range, print the character if it is in available range

for (j = ecol + 1; j < 16; j++)

printf("."); // if it is not in the range , print it '.'

}

else

{

printf("%05X ", memv);

memv += 16;

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

{

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

}

printf(" ;");

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

{

if (32 <= Memory[i][j] && Memory[i][j] <= 126)

printf("%c", Memory[i][j]);

else

printf(".");

}

} // if it is in the range, print the character if it is in available range

printf("\n");

}

}

return startid;

}

}

// function that is used in the case of dump, dump %x, and dump %x,%x.

// use range to print appropriate characters

// print error message if startpoint or the endpoint is not in the available range.

void EditMemory(unsigned char \*\*Memory, int address, int value)

{

int Tarrow, Tarcol; //Target address's row and col

if (value < 0 || value > 255)

{

printf("This value is not in available range!\n");

return;

}

if (address < 0 || address > 1048575)

{

printf("Address is not in available range!\n");

return;

}

Tarrow= address / 16;

Tarcol = address % 16;

Memory[Tarrow][Tarcol] = value;

}

//function for edit value in the target address

//print error message if the value or the target address is not in the range.

void fill(unsigned char \*\*Memory, int startA, int endA, int value)

{

int srow, scol;

int erow, ecol;

int enddot;

int i, j;

//int tmp, val;

if (startA < 0 || 1048575 < startA)//error input

{

printf("Startpoint is not in available range!\n");

return;

}

if (endA < 0 || 1048575 < endA)

{

printf("Endpoint is not in available range!\n");

return;

}

if (endA < startA)

{

printf("Startpoint should be smaller than the Endpoint!\n");

return;

}

if (value < 0 || 255 < value)

{

printf("Value is not in available range!\n");

return;

}

srow = startA / 16;

scol = startA % 16;

erow = endA / 16;

ecol = endA % 16;

// find the row and the col of the startpoint and the endpoint

for (i = srow; i < erow + 1; i++)

{

if (i == srow)

{

if (srow == erow)

enddot = ecol; // if srow==erow , enddot should be the ecol

else

enddot = 15; // if not, we should print the end of the line so enddot is 15

for (j = scol; j < enddot + 1; j++)

{

Memory[i][j] = value;

}

// fill the value in the memory

}

else if (i == erow)

{

for (j = 0; j < ecol + 1; j++)

{

Memory[i][j] = value;

}

// fill the value in the memory

}

else {

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

Memory[i][j] = value;

} //fill the value in the memory

}

}

//function for fill value in the given range of the address

//print error message if value or startpoint or endpoint is not in available address

//print error message if startpoint is bigger than the endpoint

int hash\_function(char \*s)

{

int i, sum;

for (i = 0, sum = 0; i <strlen(s); i++)

{

sum += s[i];

}

return sum % 20;

}

//function for the hash

void OpcodeInput(hashtable \*\*ht)

{

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

char tmpmneo[20];

char tmpname[20];

char tmpvalue[20];

hashtable \*temp, \*head;

int i;

int idx;

if (fp == NULL)

{

printf("There is no opcode.txt!!\n");

return;

}

// Print Error message if there is no such opcode.

else

{

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

{

ht[i] = NULL;

}

//make the head of the hashtable NULL

while (fscanf(fp,"%s %s %s",tmpvalue,tmpmneo,tmpname)!=EOF) // do fscanf until there is nothing.

{

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

temp->next = NULL;

strcpy(temp->mneo, tmpmneo);

strcpy(temp->name, tmpname);

sscanf(tmpvalue, "%x", &temp->value);

idx = hash\_function(temp->mneo);

//fill the tempnode by the data given by the file

if (ht[idx] == NULL)

{

ht[idx] = temp;

}

// put tempnode at the front of the idx

else

{

head = ht[idx];

for (;;)

{

if (head->next == NULL)break;

head = head->next;

}

head->next = temp;

}

// if there exist node int the front, put the newnode(temp) in the back.

}

}

fclose(fp);

}

//function for make the opcode hash table by using linked list

//put the hash struct in the appropriate position

void Opcodelist\_print(hashtable\*\* ht)

{

hashtable\* temp;//pointer for searching hash table

int i;//index

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

for (i = 0; i<20; i++)//printing the hash table

{

printf("%d : ", i);

for (temp = ht[i]; temp != NULL; temp = temp->next)

{

printf("[%s, %X]", temp->mneo, temp->value);

if (temp->next != NULL)

printf(" -> ");

}

printf("\n");

}

}

//function for printing the existing opcodes in the hash table

int FindOpPrint(char\* command, hashtable\*\* ht)

{

int index;//index of hash table

char tmpname[100];//string of inputing opcode

char tmpmeno[100];//string of inputing mnemonic

int flag = 0;

hashtable\* temp;//pointer for searching hash table

index = 0;

sscanf(command, "%s%s", tmpname, tmpmeno);

index = hash\_function(tmpmeno);

temp = ht[index];

for (temp = ht[index]; temp != NULL; temp = temp->next)

{

if (strcmp(temp->mneo, tmpmeno) == 0)

{

flag = 1;

break;

}

}

// search in the hash if there is same name of integer

if (flag == 0)

{

return 0;

}

// if there is no such name, return 0 for the failure

else {

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

return 1;

}

// if there is such name, return 1 for the success

}

//function that finds the opcode from the input command

// if there is same thing in hash table, print. if not, return 0

int main(void)

{

int i, j;

int len;

int cnt = 0;

int dstart=0;

int epoint;

int spoint;

int addr; //for edit

int value; // for edit and fill

int startA, endA; // address for fill

int opflag; // flag for the opcode -> if true return 1, else return 0

char \*command; // char string to put input without blanks

char \*input; // input string

Node \*hs; //Node for histroy

int flag=0;

unsigned char \*\*Memory; //Memory

char junk[45]; // use in sscanf for the mneomonics

char stp[45]; // use in sscanf for the first string

char edp[45]; // use in sscanf for the middle string in fill, end string in edit and dump

char valp[45]; // use in sscanf for the end string in fill

hashtable \*ht[20]; // hash for the mneomonics

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

hs->link = NULL;

Memory = (unsigned char\*\*)calloc(sizeof(unsigned char\*),65536);

for (i = 0; i < 65536; i++)

Memory[i] = (unsigned char\*)calloc(sizeof(unsigned char),16);

// memory and the strings memory allocation

OpcodeInput(ht);

// function execution for the opcode

while (1)

{

printf("sicsim>");

command = (char\*)malloc(sizeof(char) \* 90);

input = (char\*)malloc(sizeof(char) \* 90);

fgets(input, 90, stdin);

len = strlen(input);

input[len - 1] = '\0'; // because \n is put after fgets, change it into NULL

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

{

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

break;

}

if (i == len)

printf("Input Error\n");

// if there is string that only have blank, print error message.

else

{

strcpy(command, trim(input)); //terminate existing blanks in the left side and the right side.

len = strlen(command);

if (len > 90 || len <= 0) {

printf("Input Error!\n");

}

// if too long or no input, print error message.

else {

if (!strcmp(command, "quit") || !strcmp(command, "q")) // if quit, free all dynamic things

{

free(input);

free(command);

for (i = 0; i < 65536; i++)

free(Memory[i]);

free(Memory);

free(hs);

break;

}

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

{

MakeHistory(&hs, command);

dir();

free(input);

free(command);

}

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

{

MakeHistory(&hs, command);

free(input);

free(command);

help();

}

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

{

if (hs->link == NULL)

MakeHistory(&hs, command); // If history node is empty, just put the command in the node. Not execute printing

else {

MakeHistory(&hs, command);

cnt = 0;

PrintHistory(hs, cnt);

} // if there exists node in the history node, print all commands

free(input);

free(command);

}

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

{

MakeHistory(&hs, command);

if (dstart > 1048575)

dstart = 0;

dstart = Dump(Memory, command, dstart, -1);

free(input);

free(command);

}

else if ((command[0] == 'd' && command[1] == 'u' && (command[2] == ' ' || command[2] == '\t')) || (command[0] == 'd' && command[1] == 'u' && command[2] == 'm' && command[3] == 'p' && (command[4] == ' '||command[4]=='\t')))

{

flag = FindValidDumpInput(command); // by flag, determine error or the case.

if (flag == -1)

printf("This is not valid Input\n"); // if flag==-1 means something wrong in input. print error message

else if (flag == 2) //flag 2 means the case dump %x

{

MakeHistory(&hs, command);

sscanf(command, "%s %[^\n]", junk, stp);

sscanf(stp, "%x", &spoint);

dstart = Dump(Memory, command, spoint, -1);

if (dstart > 1048575)

dstart = 0;

}

else if (flag == 3) // flag 3 means the case dump %x,%x

{

sscanf(command, "%s %[^,] %\*c %[^\n]", junk, stp, edp);

sscanf(stp, "%x", &spoint);

sscanf(edp, "%x", &epoint);

if (spoint > epoint)

{

printf("Not available range!\n");

}

else {

MakeHistory(&hs, command);

dstart= Dump(Memory, command, spoint, epoint);

if (dstart > 1048575)

dstart = 0;

}

free(input);

free(command);

}

else

printf("Input Error!!\n");

}

else if ((command[0] == 'e' && (command[1] == ' ' || command[1] == '\t')) || (command[0] == 'e' && command[1] == 'd' && command[2] == 'i' && command[3] == 't' && (command[4] == ' ' || command[4] == '\t')))

{

flag = FindValidDumpInput(command); // find the valid form for edit

if (flag == 3)

{

MakeHistory(&hs, command);

sscanf(command, "%s %[^,] %\*c %[^\n]", junk, stp, edp);

sscanf(stp, "%x", &addr);

sscanf(edp, "%x", &value);

EditMemory(Memory, addr, value);

}

else

printf("Input Error!!\n");

free(input);

free(command);

}

else if ((command[0] == 'f' && command[1] == ' ') || (command[0] == 'f' && command[1] == 'i' && command[2] == 'l' && command[3] == 'l' && (command[4] == ' '||command[4]=='\t')))//if "f" or "fill" mnemonic ,

{

flag = FindValidDumpInput(command); // find the valid form for fill

if (flag == 4)

{

sscanf(command, "%s %[^,] %\*c %[^,] %\*c %[^\n]", junk, stp, edp,valp);

sscanf(stp, "%x", &startA);

sscanf(edp, "%x", &endA);

sscanf(valp, "%x", &value);

printf("value : %x\n", value);

fill(Memory, startA, endA, value);

MakeHistory(&hs, command);

}

else

printf("This is not valid fill input form!!\n");

free(input);

free(command);

}

else if (strcmp("reset", command) == 0) //make all memory to 0

{

MakeHistory(&hs, command);

for (i = 0; i < 65536; i++)

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

Memory[i][j] = 0;

free(input);

free(command);

}

else if (command[0] == 'o' && command[1] == 'p' && command[2] == 'c' && command[3] == 'o' && command[4]=='d' && command[5]=='e'&&(command[6]==' '||command[6]=='\t'))

{

opflag = FindOpPrint(command,ht);

if (opflag == 0) {

printf("There is no such mnemonic in the opcodelist!\n");

}

else

MakeHistory(&hs, command);

free(input);

free(command);

}

// print opcode if there is same name

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

{

Opcodelist\_print(ht);

MakeHistory(&hs, command);

free(input);

free(command);

}

// print opcodelist

else

{

printf("There is no such command!\n");

free(input);

free(command);

}

// if Wrong command, print error

}

}

}

return 0;

}