과목 명: 시스템프로그래밍

담당 교수 명: 소 정 민

<<Assignment 1>>

**서강대학교 컴퓨터공학과**

**[20151619]**

**[최준수]**

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

3.1.2 사용 변수

3.2 모듈 이름: cmd\_help() 5

3.2.1 기능

3.3 모듈이름: cmd\_dump(int start, int end, int\* nextAdr) 6

3.3.1 기능

3.3.2 사용변수

3.4 모듈이름: cmd\_edit(int adr, int value) 6

3.4.1 기능

3.4.2 사용변수

3.5 모듈이름: cmd\_fill(int start, int end, int value) 7

3.5.1 기능

3.5.2 사용변수

3.6 모듈이름: cmd\_reset() 7

3.6.1 기능

3.7 모듈이름: cmd\_hashlistSearch(char\*) 7

3.7.1 기능

3.7.2 사용변수

3.8 모듈이름: linkedlist\_push(lptr\* head, char\* command) 7

3.8.1 기능

3.8.2 사용변수

3.9 모듈이름: linkedlist\_print(lptr head) 8

3.9.1 기능

3.9.2 사용변수

3.10 모듈이름: hashlist\_printAll(hptr\* head) 8

3.10.1 기능

3.10.2 사용변수

3.11 모듈이름: hashfunction(char\* mnem) 8

3.11.1 기능

3.11.2 사용변수

3.12 모듈이름: hashlist\_push(hptr\* head, char\* mnem, int opcode) 9

3.12.1 기능

3.12.2 사용변수

3.13 모듈이름: hashMain(char\* fname) 9

3.13.1 기능

3.13.2 사용변수

3.14 모듈이름: cmd\_dir() 9

3.14.1 기능

3.14.2 사용변수

3.15 모듈이름:compareString(char\* command, char\* shortcommand, char\* longcommand) 10

3.15.1 기능

3.16 모듈이름: checkRange(int adr) 10

3.16.1 기능

3.17 모듈이름: lowerToUpper(char\* word) 10

3.17.1 기능

3.18 모듈이름: checkComma(char\* command) 10

3.18.1 기능

3.18.2 사용변수

3.19 모듈이름: checkHex(char\* str) 11

3.19.1 기능

3.19.2 사용변수

4. 전역 변수 정의 11

4.1 char \*memory

4.2 hashlist \*\*optable

5. 코드 12

# 프로그램 개요

SIC/XE 머신 구현을 위해 사용자의 입력을 받아 임의로 만든 메모리에 접근하여 입력, 변경, 출력하고 Opcode와 Mnemonic을 Hash Table에 넣고 검색 가능하도록 한다.

# 프로그램 설명

## 프로그램 흐름도

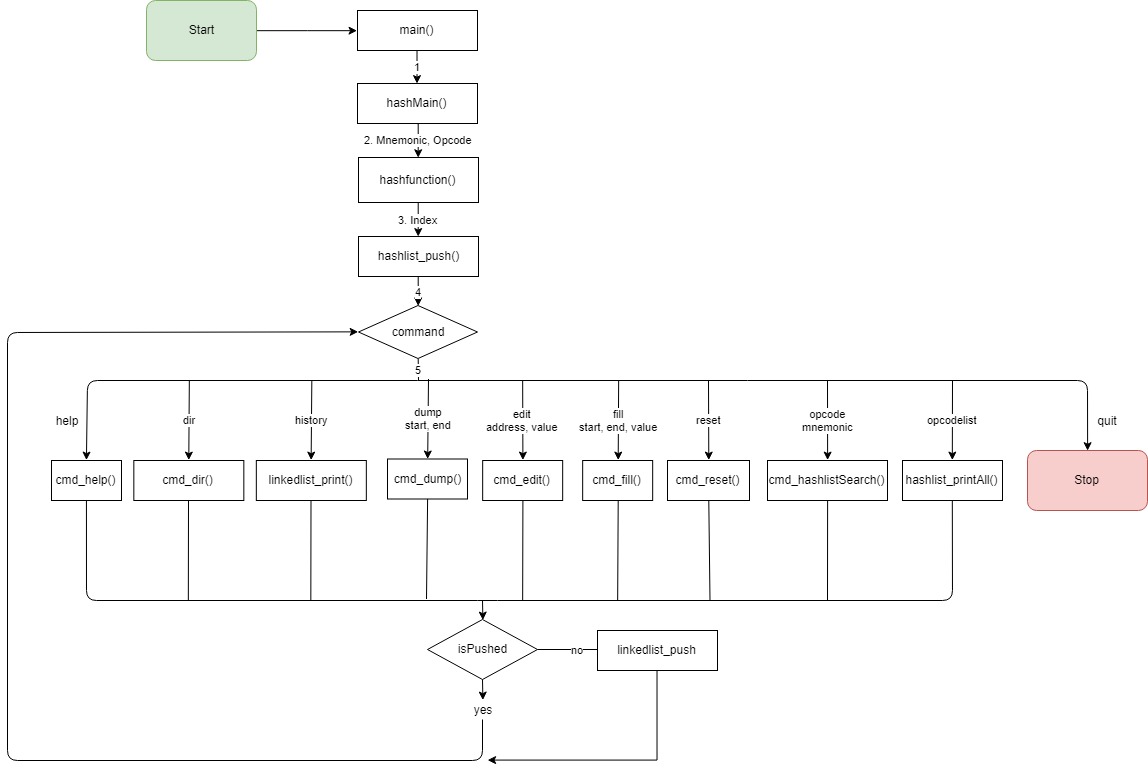


그림 1> 프로그램 흐름도

# 모듈 정의

## 모듈 이름 : main()

### 기능

fgets()로 입력을 받아 sscanf로 사용자의 command와 arguments를 분리하여준다. 그리고 잘못된 입력에 대한 예외처리를 한다. Main()에서 완료하지 못한 예외처리를 각 함수들 안에서 추가로 하게 되는데 각 함수의 리턴값으로 다른 예외사항이 발생하였는지를 판단한다. 예외사항이 없었을 경우 linkedlist\_push()를 사용하여 history에 저장하여준다. 예외가 발생한경우 isPushed를 TRUE로 바꾸어 linkedlist\_push()가 호출되지 않도록 한다.

### 사용 변수

|  |  |
| --- | --- |
| int isPushed  int argCount,  int bfrCount  char fullCmd[200]  char command[30]  char bfr1[30],  bfr2[30], bfr3[30],  bfr4[30], bfr5[30]  int arg1, arg2, arg3  int nextAdr | linkedlist로 구성된 history에 push할지 말지를 결정한다. TRUE이면 push하지 않는다.  argCount는 16진수로 입력을 받았을 때 sscanf가 반환한 성공적으로 저장된 argument의 개수, bfrCount는 문자열로 모든 입력을 받았을 때 sscanf가 반환한 개수로서 두개를 서로 비교하여 잘못된 입력이 들어왔는지 비교한다.  사용자가 입력한 전체 문자열  fullCmd에서 명령어만을 분리한 것  fullCmd의 문자열 중 argument들을 string으로 받아서 저장.  fullCmd의 문자열중 argument들을 int형(hexadecimal)로 저장.  Dump 명령어에서 start가 주어지지 않았을 경우 읽어올 다음 주소값 |

## 모듈 이름: cmd\_help()

### 기능

사용자가 입력할 수 있는 모든 명령어들을 출력해 준다.

## 모듈이름: cmd\_dump(int start, int end, int\* nextAdr)

### 기능

main에서 출력해야될 범위인 start와 end를 넘겨주는데 만약 사용자로부터 해당 입력이 들어오지 않았다면 INT\_MIN으로 세팅되어있을 것이다. INT\_MIN인지 아닌지 확인하여 만약 start, end 둘다 들어오지 않았을 경우 start를 nextAdr, end를 nextAdr + 159, start만 들어왔을 경우 end를 start + 159, 둘다 들어왔을 경우에는 그대로 사용한다. 실제 출력 작업으로 넘어가기전에 start와 end가 메모리 범위를 벗어났는지를 확인한 후 진행한다.

한줄에 출력해야되는 메모리가 총 16바이트이므로 처음에는 start / 16으로 start가 위치한 row를 출력한 후, 실제 메모리값을 출력할 때는 해당 row의 0번째부터 시작하지만 start에 도달할때까지는 공백만을 출력한다. 그리고 start부터 end까지는 저장되어 있는 값을 16진수로 출력한다. 만약 end가 위치한 row에 end범위를 벗어난 주소가 있다면 이 역시 공백을 출력한다. 메모리에 저장되어 있는 값을 출력한 후에는 해당 값들이 실제로 어떤 문자인지 매칭되는 ASCII를 출력해준다. 다만, 매뉴얼에 명시된 범위를 벗어나거나 값이 없을 경우에는 점으로 출력한다.

### 사용변수

|  |  |
| --- | --- |
| Int start  Int end  Int\* nextAdr  Int row  Int col  Int rowStart  Int rowEnd  Int curAdr  Int over | 출력할 메모리의 시작 주소  출력할 메모리의 마지막 주소  Start가 입력되지 않았을 경우 start대신 사용될 주소  For문 안에서 한 행을 다 출력하면 1씩 증가한다.  For문 안에서 한 칸을 출력하면 1씩 증가한다.  Start가 몇번째 row에 속해있는지를 저장하고 있다.  End가 몇번째 row에 속해있는지를 저장하고 있다.  Row \* 16 + col의 값을 저장하고 있으며 현재 주소값이다.  검사하는 주소값이 범위를 벗어났는지 확인후 종료한다. |

## 모듈이름: cmd\_edit(int adr, int value)

### 기능

인자로 받은 adr이 유효한 범위에 위치하는지 확인한 후, 해당 주소에 value를 저장한다.

### 사용변수

|  |  |
| --- | --- |
| Int value  Int adr | 저장하고 싶어하는 값  값을 저장하고자 하는 주소값 |

## 모듈이름: cmd\_fill(int start, int end, int value)

### 기능

인자로 넘어온 start부터 end까지 value로 저장해준다. 만약 start의 값이 end보다 클 경우, start나 end가 유효한 범위 내에 있지 않을 경우는 에러 메시지를 출력해준다.

### 사용변수

|  |  |
| --- | --- |
| Int start  Int end  Int value  Int rowStart  Int rowEnd  Int row  Int col  Int curAdr | 저장을 시작할 주소  저장을 할 마지막 주소  저장을 하고자 하는 값  Start가 속해있는 row번호  End가 속해 있는 row번호  현재 접근 중인 row  현재 접근 중인 col  Row \* 16 + col의 계산 값으로써 현재 접근중인 주소 |

## 모듈이름: cmd\_reset()

### 기능

메모리의 전체를 다 0으로 초기화 한다. (범위: 0x0 ~ 0xFFFFF)

## 모듈이름: cmd\_hashlistSearch(char\* mnem)

### 기능

입력받은 mnemonic을 대문자로 변환한 후 hashfunction에 입력하여 해당 mnemonic이 존재한다면 위치할 index를 받는다. 그리고 전역변수 optable의 해당 index의 링크드 리스트를 한칸씩 옮겨가며 입력받은 mnemonic이 존재하는지 확인한다. 만약 입력받은 mnemonic을 만나지 못하였는데 null에 다다랐다면 잘못된 입력이라는 에러메세지를 출력하고 발견하였다면 입력받은 mnemonic의 opcode를 출력하여준다.

### 사용변수

|  |  |
| --- | --- |
| Char\* mnem  Int index | 사용자가 입력한 mnemonic  hashfunction에서 반환한 입력받은 mnemonic이 있을 수 있는 index |

## 모듈이름: linkedlist\_push(lptr\* head, char\* command)

### 기능

linkedlist의 첫 시작 노드인 head를 전달받아 루프를 돌며 마지막 노드에 도착하면 새로운 노드를 생성하고 사용자가 입력한 command를 새로운 노드에 저장한 뒤 linkedlist의 끝에 연결한다.

### 사용변수

|  |  |
| --- | --- |
| lptr\* head  lptr temp  lptr newNode  char\* command | linkedlist의 시작 노드의 주소값을 가지고 있음.  linkedlist의 시작 노드의 주소값을 전달받아 루프를 돌때마다 한칸씩 이동함.  linkedlist의 끝에 추가될 새로운 노드.  newNode에 저장할 사용자가 입력한 커맨드 |

## 모듈이름: linkedlist\_print(lptr head)

### 기능

linkedlist를 처음 head노드부터 마지막 (next가 null)인 노드까지 한칸씩 이동하며 사용자가 지금까지 입력하였던 명령들을 순서대로 출력해준다.

### 사용변수

|  |  |
| --- | --- |
| int count  lptr temp | 몇번째 명령인지 출력하기 위해 한 노드를 이동할 때마다 1씩 증가한다.  head에 저장된 주소를 받아 한칸씩 노드를 이동한다. |

## 모듈이름: hashlist\_printAll(hptr\* head)

## 기능

|  |
| --- |
| [0] |
| [1] |
| [2] |

|

|  |
| --- |
| [19] |

위의 형식으로 되어있는 총 20개의 인덱스를 가진 hashtable을 한 인덱스씩 돌며 각 인덱스에 연결된 linkedlist의 끝까지 출력하고 다음 인덱스로 이동하며 출력한다.

### 사용변수

|  |  |
| --- | --- |
| hptr \*head  hptr temp  int i  int isFirst | hashtable의 시작 head 주소를 가지고 있다  head의 주소를 전달받아 한 칸씩 돌며 출력해준다  반복문에 사용되는 인자로서 현재 접근중인 인덱스를 나타낸다  각 인덱스에서 첫 노드의 앞에 화살표가 출력되지 않도록 flag 역할을 한다. |

## 모듈이름: hashfunction(char\* mnem)

### 기능

인자로 넘어온 mnem문자열의 모든 문자들을 정수로 바꿨을 때의 값을 모두 더한 뒤, 이를 20으로 나눈 나머지를 hashtable에 들어갈 index로 사용할 수 있도록 반환해 준다.

### 사용변수

|  |  |
| --- | --- |
| char\* mnem | 정수인 인덱스로 치환하고 싶은 문자열 |

## 모듈이름: hashlist\_push(hptr\* head, char\*\* mnem, int opcode)

### 기능

head는 optable중 한 인덱스의 주소값을 가지고 있는데, 이 인덱스에 새로운 노드를 추가하는 함수이다. 따라서 같이 인자로 넘어온 mnem과 opcode를 새로운 노드에 저장하여 끝에 이어준다. hptr temp가 head부터 null을 만날때까지 한칸씩 이동을 하고 null을 만나면 그 자리에 새로운 노드의 주소값을 넣어준다.

### 사용변수

|  |  |
| --- | --- |
| hptr \*head  hptr temp  char \*mnem  int opcode | hashtable의 한 index의 linkedlist가 시작하는 주소값을 저장하고 있다.  head의 주소를 저장하여 한칸씩 null을 만날때까지 이동한다.  새로운 노드에 저장할 mnemonic을 저장하고 있다.  새로운 노드에 저장할 opcode를 저장하고 있다. |

## 모듈이름: hashMain(char\* fname)

### 기능

hashMain은 처음에 opcode.txt 를 optable에 삽입하는 함수이다. opcode.txt.를 한줄씩 읽어와서 입력받은 mnemonic을 hashfunction에 입력하여 그 mnemonic이 들어가야될 index를 구하고, optable의 해당 index주소값을 mnemonic과 opcode와 함께 hashlist\_push()에 넘겨 삽입한다.

### 사용변수

|  |  |
| --- | --- |
| char \*fname  FILE \*fp  int opcode  char mnem[30]  char format[30]  int index | 로드할 파일의 이름을 받아온다.  파일을 열어 로드한 값을 가지고 있는 파일 포인터이다.  파일에서 받아온 opcode이다  파일에서 받아온 mnemonic이다  파일에서 받아온 format이다  파일에서 받아온 mnemonic을 사용하여 hashfunction이 반환하는 인덱스이다. |

## 모듈이름: cmd\_dir()

### 기능

현재 디렉토리에 들어있는 파일/디렉토리의 이름들을 출력하여준다. 만약 디렉토리라면 이름 뒤에 ‘/’를 추가하고, executable한 파일이라면 뒤에 ‘\*’를 추가하여 출력한다.

### 사용변수

|  |  |
| --- | --- |
| DIR \*dir  struct dirent \*file  struct stat info | 현재 디렉토리의 정보를 저장하고 있다.  현재 디렉토리에서 하나의 파일에 대한 정보를 저장하고 있다.  file이 가지고 있는 파일의 형식에 대한 정보를 가지고 있다. |

## 모듈이름: compareString(char\* command, char\* shortcommand, char\* longcommand)

### 기능

사용자가 입력한 command가 shortcommand(예: du)나 longcommand(예: dump) 중 하나와 일치하는지 확인하고 일치할 경우 TRUE를 반환한다.

### 사용변수

|  |  |
| --- | --- |
| Char\* command  Char\* shortcommand  Char\* longcommand | 사용자가 입력한 실제 명령어  체크하고자 하는 명령어의 짧은 줄임말  체크하고자 하는 명령어의 실제 명령어 |

## 모듈이름: checkRange(int adr)

### 기능

넘어온 인자인 adr이 메모리의 주소 범위를 벗어날 경우 FALSE, 유효한 범위일 경우 TRUE를 반환한다.

### 사용변수

|  |  |
| --- | --- |
| Int adr | 메모리의 유효한 범위 내인지 확인하고자 하는 주소 |

## 모듈이름: lowerToUpper(char\* word)

### 기능

opcode의 mnemonic을 입력받았을 때 opcode.txt에서 입력받은 mnemonic은 모두 대문자로 이루어져있기 때문에 소문자로 사용자가 입력하면 hashfunction이 리턴하는 값이 서로 다르므로 입력받은 값을 전부 대문자로 바꾸어준다.

### 사용변수

|  |  |
| --- | --- |
| Char\* word | 소문자에서 대문자로 바꾸고자 하는 문자열 |

## 모듈이름: checkComma(char\* command)

### 기능

command의 문자열의 제일 끝이 ‘,’로 끝나는지 확인하여 ‘,’로 끝날경우 FALSE, 아닐경우 TRUE를 리턴한다. 잘못된 입력을 예외처리하는 데에 사용된다.

### 사용변수

|  |  |
| --- | --- |
| Int len  Char\* command | Command의 문자열 길이를 저장  ‘,’로 끝나는지 확인하고자 하는 입력값 |

## 모듈이름: checkHex(char\* str)

### 기능

command의 문자열에 16진수가 아닌 문자가 포함되어있는지 체크한다. 0x가 들어오는 경우는 정상 입력으로 처리한다. 이는 sscanf의 특성상 예를 들어 dump 4hello를 입력한다면 dump 4를 입력한 것과 같이 처리되어 잘못된 입력임에도 정상적으로 실행되는 것을 방지하기 위함이다.

### 사용변수

|  |  |
| --- | --- |
| Char\* str | 16진수가 아닌 수가 포함되었는지 확인하고자 하는 입력값 |

# 전역 변수 정의

## char\* memory

이번 프로젝트에서 메모리의 역할을 하기위해 1MB의 크기로 됨 char형 배열이다.

# 4.2 hashlist \*\*optable

opcode, mnemonic, format을 linkedlist로 된 hashtable에 저장하고 있다. 크기는 매뉴얼을 따라 20이다.

# 코드

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <sys/stat.h>

#include <limits.h>

#include "util.h"

#include "commands.h"

#define TRUE 1

#define FALSE 0

#define EXECUTABLE 33279

#define DIRECTORY 16895

#define HASH\_SIZE 20

#define MEM\_SIZE 65536 \* 16

typedef struct linkedlist\* lptr;

typedef struct linkedlist {

char command[100];

lptr next;

}linkedlist;

void linkedlist\_push(lptr\*, char\*);

void linkedlist\_print(lptr);

typedef struct hashlist\* hptr;

typedef struct hashlist {

char mnem[20];

int opcode;

char format[20];

hptr next;

}hashlist;

void hashMain(char\*);

int hashfunction(char\*);

void hashlist\_push(hptr\*, char\*, int, char\*);

void hashlist\_printAll(hptr\*);

char \*memory;

hashlist \*\*optable;

#include "20151619.h"

//DIR Executable

int main()

{

int isPushed = FALSE, opcode;

int argCount=0, bfrCount=0, comCount=0;

char command[30],fullCmd[200];

char bfr[7][30];

unsigned int arg1,arg2,arg3;

int nextAdr = 0;

//Initialization

lptr history = NULL;

optable = (hptr\*)malloc(sizeof(hptr)\*HASH\_SIZE);

for (int i=0; i<HASH\_SIZE; i++)

optable[i] = NULL;

memory=(char\*)malloc(sizeof(char\*)\*MEM\_SIZE);

cmd\_reset();

for(int i=0; i<7; i++)

memset(bfr[i],0,sizeof(bfr[i]));

hashMain("opcode.txt");

while(TRUE)

{

printf("sicsim>");

//Get Input

fgets(fullCmd, sizeof fullCmd,stdin);

//save with the correct format

argCount = sscanf(fullCmd,"%s%x ,%x ,%x",command, &arg1, &arg2, &arg3);

//save everything as string for checking

bfrCount = sscanf(fullCmd,"%s%s%s%s%s%s%s",bfr[0],bfr[1],bfr[2],bfr[3],bfr[4],bfr[5],bfr[6]);

//For input with input value (space) , value (space) , value

if(!strcmp(",",bfr[2])){

for(int i=3; i<7; i++)

strcpy(bfr[i-1],bfr[i]);

comCount++;

}

if(!strcmp(",",bfr[3])){

for(int i=4; i<7; i++)

strcpy(bfr[i-1],bfr[i]);

comCount++;

}

if (compareString(command, "opcode", NULL) && bfrCount == 2) {

//correct format for OPCODE [instruction] inserted

opcode = cmd\_hashlistSearch(bfr[1]);

if (opcode != -1){

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

linkedlist\_push(&history,fullCmd);

}

//invalid [instruction]

else {

printf("Invalid Mnemonic!\n\n");

}

continue;

}

//cases such as dump 4, hello

if (argCount != bfrCount - comCount) {

isPushed = TRUE;

printf("Invalid Command!\n");

}

// cases such as dump 1 , 10 ,

else if (comCount != 0 && argCount - 2 < comCount){

isPushed = TRUE;

printf("Invalid Command!\n");

}

else if (compareString(command, "h", "help") && argCount == 1)

cmd\_help();

else if (compareString(command, "d", "dir") && argCount == 1)

cmd\_dir();

else if (compareString(command, "q", "quit") && argCount == 1)

break;

else if (compareString(command, "hi", "history") && argCount == 1) {

linkedlist\_push(&history, fullCmd);

linkedlist\_print(history);

isPushed = TRUE;

}

//Maximum number of input is 3

else if (compareString(command, "du", "dump") && argCount <= 3) {

switch (argCount) {

//when only dump is inserted

case 1:

arg1 = INT\_MIN;

arg2 = INT\_MIN;

break;

//when dump start is inserted

case 2:

arg2 = INT\_MIN;

isPushed = checkHex(bfr[1]) && checkComma(bfr[1]);

isPushed = !isPushed;

break;

//when dump start, end is inserted

case 3:

isPushed = checkHex(bfr[1]) && checkHex(bfr[2]) && checkComma(bfr[2]);

isPushed = !isPushed;

break;

}

if (!isPushed)

isPushed = !(cmd\_dump(arg1, arg2, &nextAdr));

//Comma at the end

else

printf("Invalid command!");

}

else if (compareString(command, "e", "edit") && argCount == 3){

isPushed = checkComma(bfr[2]) && checkHex(bfr[1]) && checkHex(bfr[2]);

isPushed = !isPushed;

if (!isPushed)

isPushed = !(cmd\_edit(arg1, arg2));

else

printf("Invalid command!");

}

else if (compareString(command, "f", "fill") && argCount == 4){

//check if the command finished with a comma or if input has non-hex

isPushed = checkComma(bfr[3]) && checkHex(bfr[1]) && checkHex(bfr[2]) && checkHex(bfr[3]);

isPushed = !isPushed;

if (!isPushed)

isPushed = !(cmd\_fill(arg1,arg2, arg3));

else

printf("Invalid command!");

}

else if (compareString(command, "reset", NULL) && argCount == 1)

cmd\_reset();

else if (compareString(command, "opcodelist", NULL) && argCount == 1)

hashlist\_printAll(optable);

else {

isPushed = TRUE;

printf("Invalid Command!\n");

}

if (isPushed == TRUE)

isPushed = FALSE;

else

linkedlist\_push(&history,fullCmd);

//RESET

for(int i=0; i<7; i++)

memset(bfr[i],0,sizeof(bfr[i]));

argCount = bfrCount = comCount = 0;

arg1 = arg2 = arg3 = INT\_MIN;

printf("\n");

}

}

//appends new node with command to the end of head linked list

void linkedlist\_push(lptr\* head,char\* command){

lptr temp = \*head;

lptr newNode = (lptr)malloc(sizeof(linkedlist));

newNode->next = NULL;

strcpy(newNode->command,command);

if(temp != NULL){

//goes to the last node

while(temp->next != NULL)

temp = temp->next;

temp->next = newNode;

}

else{

\*head = newNode;

}

return;

}

//prints all the nodes in the linked list my moving one node at a time

void linkedlist\_print(lptr head){

int count = 1;

lptr temp = head;

while(temp != NULL){

printf("%-4d %s",count++,temp->command);

temp = temp->next;

}

return;

}

//main function to create initial hashtable

void hashMain(char\* fname){

FILE \*fp = fopen(fname, "r");

int opcode,index;

char mnem[30], format[30];

if (fp != NULL){

while(fscanf(fp,"%X %s %s",&opcode,mnem,format) != EOF){

//finds the index for mnemonic to go in the hashtable

index = hashfunction(mnem);

//appends that mnemonic to the table

hashlist\_push(&(optable[index]),mnem,opcode,format);

}

}

else{

printf("Cannot open File\n!");

}

}

//returns index number for the mnemonic to go into in the hashtable

int hashfunction(char\* mnem){

int i,sum=0;

for(i=0; i<(int)strlen(mnem); i++)

sum += (int)mnem[i];

return sum % HASH\_SIZE;

}

//creates a new node with mnemonic, opcode and format. Then inserts the new node the end of head list

void hashlist\_push(hptr \*head,char\* mnem,int opcode, char\* format){

hptr temp = \*head;

//make new node

hptr newNode = (hptr)malloc(sizeof(hashlist));

newNode->next = NULL;

strcpy(newNode->mnem,mnem);

strcpy(newNode->format,format);

newNode->opcode = opcode;

if(\*head != NULL){

//go to the end of head

while(temp->next != NULL)

temp = temp->next;

//insert to new node

temp->next = newNode;

}else{

\*head = newNode;

}

return;

}

//prints whole hashtable

void hashlist\_printAll(hptr \*head){

int i, isFirst;

hptr temp;

for(i=0; i<HASH\_SIZE; i++){

temp = head[i];

isFirst = TRUE;

//if an index is not empty

if(temp != NULL){

printf("%d : ",i);

//iterate through the list

while(temp != NULL){

if(!isFirst)

printf(" -> ");

printf("[%s, %X]",temp->mnem,temp->opcode);

temp = temp->next;

isFirst = FALSE;

}

printf("\n");

}

}

}

void cmd\_dir();

void cmd\_help();

void cmd\_reset();

int cmd\_dump(int, int, int\*);

int cmd\_edit(int, int);

int cmd\_fill(int, int, int);

int cmd\_hashlistSearch(char\*);#include "commands.h"

#include "20151619.h"

//Prints Help

void cmd\_help() {

printf("h[elp]\nd[ir]\nq[uit]\nhi[story]\ndu[mp] [start,end]\n");

printf("e[dit] address, value\nf[ill] start, end, value\nreset\n");

printf("opcode mnemonic\nopcodelist\n");

return;

}

//Prints files in current directory

void cmd\_dir() {

DIR \*dir;

struct dirent \*file;

struct stat info;

if ((dir = opendir("./")) != NULL) {

/\* print all the files and directories within directory \*/

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

stat(file->d\_name, &info);

//If file is a directory

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

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

//If file is executable

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

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

//other files

else

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

}

closedir(dir);

}

else {

/\* could not open directory \*/

perror("");

printf("Error opening directory");

}

return;

}

//Prints memory

int cmd\_dump(int start, int end, int\* nextAdr) {

int row, col, over = FALSE;

int rowStart, rowEnd, curAdr;

//if ending addr is not inserted

if (end == INT\_MIN) {

//if starting addr is not inserted

if (start == INT\_MIN)

start = \*nextAdr;

end = start + 159;

}

else{

//if inserted range end is out of bound

if (checkRange(end) == FALSE || checkRange(start) == FALSE){

printf("Invalid Range\n");

return FALSE;

}

}

//if start addr is bigger than end addr

if (end < start) {

printf("Invalid Range\n");

return FALSE;

}

// if start is out of memory range

if (!checkRange(start)) {

printf("Invalid Range\n");

return FALSE;

}

rowStart = start / 16;

rowEnd = end / 16;

for (row = rowStart; row <= rowEnd; row++) {

// if row is out of range

if (row \* 16 > 0xFFFFF) {

over = TRUE;

break;

}

printf("%05X ", row \* 16);

//Content

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

curAdr = row \* 16 + col;

if (curAdr > 0xFFFFF) {

over = TRUE;

break;

}

//Range before start

if (curAdr < start)

printf(" ");

//Range after end

else if (curAdr > end)

printf(" ");

else

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

}

printf("; ");

//Value

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

{

curAdr = row \* 16 + col;

if (curAdr > 0xFFFFF) {

over = TRUE;

break;

}

//Range before start

if (curAdr < start)

printf(". ");

//Range after end

else if (curAdr > end)

printf(". ");

else {

if ((int)memory[curAdr] >= 0x20 && (int)memory[curAdr] <= 0x7E)

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

else

printf(". ");

}

}

printf("\n");

}

if (over)

\*nextAdr = 0;

else

\*nextAdr = end + 1;

return TRUE;

}

//change a memory's content

int cmd\_edit(int adr, int value) {

if (!checkRange(adr)) {

printf("Wrong Address!\n");

return FALSE;

}

memory[adr] = value;

return TRUE;

}

//changes memory to value from start adr to end adr

int cmd\_fill(int start, int end, int value) {

int rowStart, rowEnd, row, col, curAdr;

if (end < start) {

printf("Invalid Range!\n");

return FALSE;

}

if (!(checkRange(start) && checkRange(end))) {

printf("Invalid Range!\n");

return FALSE;

}

rowStart = start / 16;

rowEnd = end / 16;

for (row = rowStart; row <= rowEnd; row++) {

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

curAdr = row \* 16 + col;

//change curAdr memory to value if curAdr is not out of bound

if (curAdr >= start && curAdr <= end)

memory[curAdr] = value;

}

}

return TRUE;

}

//resets entire memory

void cmd\_reset() {

memset(memory, 0, sizeof(char)\*MEM\_SIZE);

}

int cmd\_hashlistSearch(char\* mnem) {

//Convert lowercase to uppercase

LowerToUpper(mnem);

int index = hashfunction(mnem);

hptr temp = optable[index];

//search untill the end of one optable's index

while (temp != NULL) {

//if the mnemonic that i'm looking for is found

if (!strcmp(mnem, temp->mnem))

return temp->opcode;

//otherwise continue to the next node

temp = temp->next;

}

return -1;

}

int checkRange(int);

int compareString(char\*, char\*, char\*);

int checkComma(char\*);

int checkHex(char\*);

void LowerToUpper(char\*);

#include "util.h"

#include "20151619.h"

//compares command with shortcommand and longcommand and see if command matches with one of them

int compareString(char\* command, char\* shortcommand, char\* longcommand) {

if (!strcmp(command, shortcommand) || (longcommand != NULL && !strcmp(command, longcommand)))

return TRUE;

else

return FALSE;

}

//check if adr is within the valid range

int checkRange(int adr) {

if (adr < 0 || adr >= 0x100000)

return FALSE;

return TRUE;

}

//check if str has any non-hex char

int checkHex(char\* str){

for(int i=0; i<(int)strlen(str); i++){

if((str[i] >=48 && str[i] <=57) || (str[i] >= 65 && str[i] <=70) || (str[i] >= 97 && str[i] <= 102) || str[i] == ',')

continue ;

else if(str[i] == 'X' || str[i] == 'x'){

if(i != 0)

if(str[i-1] == '0')

continue;

}

else{

return FALSE;

}

}

return TRUE;

}

//changes word to uppercase

void LowerToUpper(char\* word) {

for (int i = 0; i < (int)strlen(word); i++)

if ((int)word[i] > 96 && (int)word[i] < 123)

word[i] -= 32;

}

//check if command finishes with a comma

int checkComma(char\* command){

int len = (int)strlen(command);

if(command[len-1] == ',')

return FALSE;

return TRUE;

}