**과목명: 시스템프로그래밍**

**분반 2**

**<<Project #1>>**

**서강대학교 [컴퓨터공학]**

**[20161598]**

**[손동현]**

목 차

1. **프로그램 개요** 3
2. **프로그램 설명**
   1. 프로그램 흐름도 3
3. **모듈 정의**
   1. 모듈 이름 : main() 4
      1. 기능 4
      2. 사용 변수 4
   2. 모듈 이름 : void ommand\_distinct(char\* arr); 4
      1. 기능 4
      2. 사용 변수 4
   3. 모듈 이름 : void command\_help(); 4
      1. 기능 4
      2. 사용 변수 4
   4. 모듈 이름 : void command\_dir(); 4
      1. 기능 4
      2. 사용 변수 4
   5. 모듈 이름 : void command\_history(); 5
      1. 기능 5
      2. 사용 변수 5
   6. 모듈 이름 : void command\_dump(int num\_flag); 5
      1. 기능 5
      2. 사용 변수 5
   7. 모듈 이름 : void command\_edit(); 5
      1. 기능 5
      2. 사용 변수 5
   8. 모듈 이름 : void edit\_add\_val(char\* arr, int\* addr, int\* val); 5
      1. 기능 5
      2. 사용 변수 5
   9. 모듈 이름 : void command\_fill(); 6
      1. 기능 6
      2. 사용 변수 6
   10. 모듈 이름 : void fill\_addr\_val(char\* arr, int\* start, int\* end, int\* val); 6
       1. 기능 6
       2. 사용 변수 6
   11. 모듈 이름 : void command\_reset(); 6
       1. 기능 6
       2. 사용 변수 6
   12. 모듈 이름 : int dump\_start\_end(char\* arr, int\* start, int\* end); 6
       1. 기능 6
       2. 사용 변수 6
   13. 모듈 이름 : void print\_memory(int first\_line, int last\_line, int dump\_start, int dump\_end); 7
       1. 기능 7
       2. 사용 변수 7
   14. 모듈 이름 : void command\_quit(); 7
       1. 기능 7
       2. 사용 변수 7
   15. 모듈 이름 : void make\_hashtb(); 7
       1. 기능 7
       2. 사용 변수 7
   16. 모듈 이름 : void command\_opcodelist(); 8
       1. 기능 8
       2. 사용 변수 8
   17. 모듈 이름 : void command\_opcode(); 8
       1. 기능 8
       2. 사용 변수 8
   18. 모듈 이름 : int distinct\_ascii(char x) 8
       1. 기능 8

3.18.2 사용 변수 8

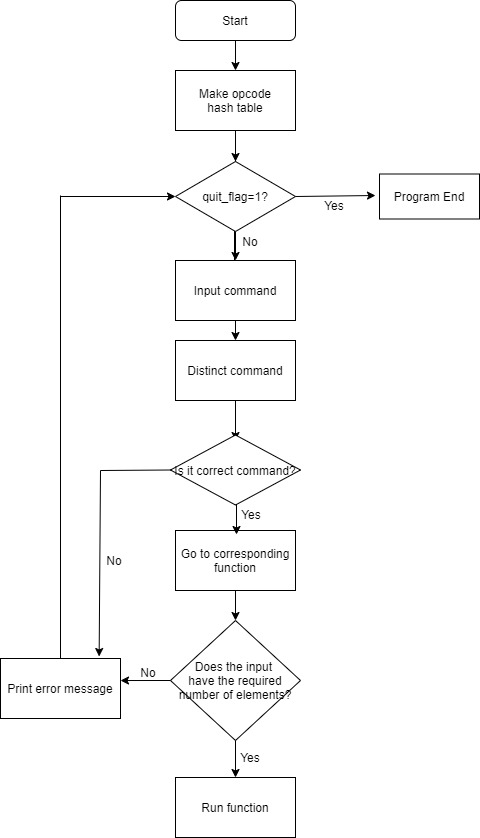
1. **전역 변수, 구조체 정의**
   1. typedef struct \_Node 9
   2. typedef struct \_Hash 9
   3. char command[100]; 9
   4. int line\_num; 9
   5. int quit\_flag; 9
   6. int error\_flag; 9
   7. int memory\_address; 9
   8. unsigned char memory[1048576];9
   9. Node\* head; 9
   10. Node \*prev; 9
   11. Hash\* table[20]; 9
2. **코드 설명**
   1. 20161598.h 9
   2. 20161598.c 25

**1 . 프로그램 개요**

이 프로그램은 SIC/XE 머신을 구현하기 위해 작성되었다. 기본적인 shell 명령어들을 수행할 수 있고, 이후 어셈블러, 링크, 로더들을 실행하게 될 것이다. 컴파일을 통해 만들어진 object 코드가 적재되고 실행될 메모리공간과 mnemonic(ADD, COMP, FLOAT, etc…) 을 opcode 값으로 변환하는 OPCODE 테이블과 관련 명령어들을 구현하였다.

**2 . 프로그램 설명**

2.1 프로그램 흐름도



위의 흐름도처럼 프로그램이 실행되면 opcode에 관련된 hash table을 만든 후, input 입력 받기를 대기한다. 입력 받은 command를 각 명령어에 맞게 구분하여 올바른 입력이 들어온 경우, 해당 함수를 호출하고, 각 명령어가 요구로 하는 수의 index, value 등이 뒤에 올바르게 들어왔는지 판단한다. 올바르게 들어온 경우 명령어가 실행되게 하고, 잘못된 명령어가 들어온 경우, 에러 메시지를 출력 후 다시 입력 대기를 한다. 입력은 quit\_flag가 1이 될 때까지 계속 반복적으로 받는데, quit 또는 q가 입력된 경우, quit\_flag를 1로 바꾼 후 프로그램이 종료된다.

**3 . 모듈 정의**

* 1. 모듈 이름 : main()

3.1.1 기능

Shell을 실행하여 opcode에 관련된 hash table 만드는 함수인 make\_hashtb()함수를 호출한 후, quit\_flag가 1이 되긴 전까지 계속 input을 입력 받고, 받은 명령어를 구분하기 위해 command\_distinct() 함수를 호출하여 준다.

3.1.2 사용 변수

int i = 0; //입력받은 명령어의 끝에 ‘\0’을 넣기 위해 사용하는 배열의 index

* 1. 모듈 이름 : void command\_distinct(char\* arr);
     1. 기능

입력 받은 command를 판별하여 해당 상응하는 기능이 있는 function을 호출하여 준다. 우선 history에 사용할 node를 만들어 받은 command와 해당 line 번호를 저장하고, 잘못된 command인 경우, 해당 node를 다시 free해준다.

* + 1. 사용 변수

Node\* node = (Node\*)malloc(sizeof(Node)); //history에 저장하기 위해 node를 새로 만듦

int dump\_flag = 0; //dump 명령어를 제대로 받았는지 여부 판별하기 위한 변수, 잘못된 명령어인 경우 1

int edit\_flag = 0; //edit 명령어를 제대로 받았는지 여부 판별하기 위한 변수, 잘못된 명령어인 경우 1

int fill\_flag = 0; //fill 명령어를 제대로 받았는지 여부 판별하기 위한 변수, 잘못된 명령어인 경우 1

int k = 2; //반복문을 위한 변수

* 1. 모듈 이름 : void command\_help();
     1. 기능

h나 help 입력된 경우, 모든 명령어 리스트 출력한다.

* + 1. 사용 변수

없음

* 1. 모듈 이름 : void command\_dir();
     1. 기능

d나 dir이 입력된 경우, 현재 디렉터리에 있는 파일들을 출력한다.

* + 1. 사용 변수

DIR\* dir\_info = NULL; //현재 디렉터리의 정보를 담는 포인터

struct dirent\* dir\_entry = NULL; //현재 디렉터리에 존재하는 파일의 정보를 담은 struct dirent의 pointer.file의 이름을 struct dirent의 요소인 d\_name을 통해 얻는다.

struct stat buf; //현재 디렉터리에 존재하는 파일의 정보를 담은 struct stat 변수이다. dir\_entry가 가리키는 것이 디렉터리인지 실행파일인지 알려주는 정보를 담은 st\_mode라는 요소가 변수로 존재한다.

int num\_index=0; //출력한 파일의 개수 저장하는 변수

* 1. 모듈 이름 : void command\_history();
     1. 기능

hi나 history가 입력된 경우, 현재까지 사용한 명령어들을 순서대로 번호와 함께 출력한다.

* + 1. 사용 변수

Node\* cur = head; //현재 node를 가리키는 포인터이다.

int i = 0; //해당 노드의 command 정보를 가져올 때, 배열의 index이다.

* 1. 모듈 이름 : void command\_dump(int num\_flag);
     1. 기능

du나 dump가 입력된 경우, du 또는 dump만 입력 받았는지, 그 뒤에 start index만 받았는지, end까지 받았는지 판별하기 위해 dump\_start\_end 함수를 호출하고, 해당 경우에 맞게 print\_memory 함수를 호출하여 메모리를 출력하여 준다. 만일 잘못된 command인 경우, 에러 메시지 출력한다.

* + 1. 사용 변수

int num\_flag; //du 또는 dump만 입력 받은 경우 1, 그 뒤로 주소 index를 받으면 0

int difference = 8 - (0xFFFF0 - memory\_address) / 16; //difference는 현재부터 0xFFFF0까지 출력후 그 다음에 출력해야할 줄의 수를 저장하는 변수.

int dump\_start, dump\_end; //dump\_start는 start 번지, dump\_end는 end 번지를 나타내는 변수.

int dump\_valid = dump\_start\_end(command, &dump\_start, &dump\_end); //start, end 번지를 판별하여 올바르게 둘다 들어오면 1, start만 들어오면 0, 잘못된 주소 받은 경우 -1을 갖는다.

* 1. 모듈 이름 : void command\_edit();
     1. 기능

e나 edit이 입력된 경우, edit\_add\_val 함수를 호출하여 주소와 값에 대한 정보를 얻은 뒤, 잘못된 명령어이면 에러 메시지를 출력하고, 올바른 명령어인 경우, 해당 메모리 주소에 값을 삽입한다.

* + 1. 사용 변수

int addr = 0; //메모리 주소 값을 갖는 변수

int val = 0; // 바꿀 값을 갖는 변수

* 1. 모듈 이름 : void edit\_add\_val(char\* arr, int\* addr, int\* val);
     1. 기능

edit command 뒤에 입력된 주소, 값을 판별하여, string에서 16진수 숫자로 바꾸어 \*addr, \*val에 각각 저장하여 준다. 그리고 만일 주소만 받고, 바꿀 값을 안 받은 경우, \*val에 -1을 넣어 command\_edit에서 오류 처리할 수 있게 해준다. 또한 바꿀 값 뒤에 추가적으로 입력 받는 경우, 잘못된 명령어이므로 에러 처리할 수 있게 해준다.

* + 1. 사용 변수

char\* arr //입력 받은 command가 저장된 배열

int\* addr //주소 값을 가리키는 포인터

int\* val //바꿀 값을 가리키는 포인터

int i, j=0; //i는 arr의 index, j는 edit\_value, edit\_address의 index,

int number\_flag=0; //0이면 arr[i]가 주소, 1또는 2이면 바꿀 값을 의미한다.

char edit\_value[10] = { 0 }; //바꿀 값을 저장할 char 배열

char edit\_address[10] = { 0 }; //주소 값을 저장할 char 배열

* 1. 모듈 이름 : void command\_fill();
     1. 기능

f나 fill가 입력된 경우, fill\_addr\_val 함수를 호출하여 시작, 끝 주소 값, 바꿀 값에 대한 정보를 얻고, 이를 비교하여 잘못된 command인 경우 에러메시지를 출력한다. 만일 올바른 command인 경우, start부터 end 지점까지 메모리를 value 값으로 채운다.

* + 1. 사용 변수

int start, end, val=-1; //start는 시작, end는 끝 주소, val은 바꿀 값을 가리키는 변수

int k; //반복문을 위해 사용하는 변수

* 1. 모듈 이름 : void fill\_addr\_val(char\* arr, int\* start, int\* end, int\* val);
     1. 기능

fill command 뒤에 입력된 start, end 주소 값을 판별하여, string에서 16진수 숫자로 바꾸어 \*start, \*end, \*val에 각각 저장하여 준다. 그리고 만일 주소만 받고, 바꿀 값을 안 받은 경우, \*val에 -1을 넣어 command\_fill에서 오류 처리할 수 있게 해준다. 또한 바꿀 값 뒤에 추가적으로 입력 받는 경우, 잘못된 명령어이므로 에러 처리할 수 있게 해준다.

* + 1. 사용 변수

char\* arr //입력 받은 command가 저장된 배열

int\* start //start 주소 값을 가리키는 포인터

int\* end //end 주소 값을 가리키는 포인터

int\* val //바꿀 값을 가리키는 포인터

int i, j = 0; // i는 arr의 index, j는 fill\_val, fill\_start, fill\_end의 index,

int number\_flag=0; //0이면 arr[i]가 start 주소, 1이면 end 주소, 2이면 바꿀 값을 의미한다.

char fill\_val[10] = { 0 }; //바꿀 값을 저장할 char 배열

char fill\_start[10] = { 0 }; //start 주소 값을 저장할 char 배열

char fill\_end[10] = { 0 }; //end 주소 값을 저장할 char 배열

* 1. 모듈 이름 : void command\_reset();
     1. 기능

reset이 입력 된 경우, 메모리 전부를 (char)0으로 변경한다.

* + 1. 사용 변수

없음

* 1. 모듈 이름 : int dump\_start\_end(char\* arr, int\* start, int\* end);
     1. 기능

dump 명령어 뒤에 start index만 있는지, start end index 모두 있는지 판별하여 잘못된 명령어인 경우, command\_dump에서 에러 처리할 수 있게 해준다. 올바른 명령어인 start, end index를 string에서 16진수로 변환하여 저장해준다.

* + 1. 사용 변수

char\* arr //입력 받은 command가 저장된 배열

int\* start //start 주소 값을 가리키는 포인터

int\* end //end 주소 값을 가리키는 포인터

char start\_num[10] = {0}; //시작 주소 값을 저장할 char 배열

char end\_num[10] = {0}; //끝 주소 값을 저장할 char 배열

int i, j=0; //i는 arr의 index, j는 start\_num, end\_num의 index

int number\_flag = 0; //0이면 arr[i]가 start index, 1이면 end index을 의미한다.

* 1. 모듈 이름 : void print\_memory(int first\_line, int last\_line, int dump\_start, int dump\_end);
     1. 기능

입력 받은 start 번지부터 end 번지까지의 메모리를 형식에 맞게 출력하여 준다. 만일 입력 받은 주소가 메모리 주소 범위에 벗어날 시, 에러 메시지를 출력한다.

* + 1. 사용 변수

int first\_line //시작 메모리 주소(16의 배수)를 의미

int last\_line //끝 메모리 주소(16의 배수)를 의미

int dump\_start //시작 메모리 번지를 의미

int dump\_end // 끝 메모리 번지를 의미

int line\_start = dump\_start; //시작 번지를 저장하는 변수

int hex\_memory\_line = first\_line; //메모리에서 바이트 부분 반복문 출력을 위해 사용하는 변수

int hex\_line = hex\_memory\_line / 16; //현재 줄의 16으로 나눈 몫을 갖는 변수, 첫 줄, 마지막 줄 판단 여부에 사용

int k = line\_starat; //메모리에서 char 부분 반복문 출력을 위해 사용하는 변수

* 1. 모듈 이름 : void command\_quit();
     1. 기능

q나 quit이 입력된 경우, quit\_flag를 1로 설정 후, history로 만든 linked list와 opcode hash table을 전부 free 하여 메모리 해제 시켜준다.

* + 1. 사용 변수

Node\* cur = head; //현재 Node를 가리키는 포인터

Node\* prev = cur; //메모리 해제하기 위해 사용되는 포인터

int i=0; //반복문에서 hash table index를 나타내는 변수

Hash\* cur = table[i]; //현재 Hash를 가리키는 포인터

Hash\* prev = cur; //메모리 해제하기 위해 사용되는 포인터

* 1. 모듈 이름 : void make\_hashtb();
     1. 기능

“opcode.txt”를 열어 한줄씩 읽어와서 각 줄에서 value와 mnemonic에 대한 정보를 얻은 뒤, 해당 mnemonic의 hash table의 index를 계산하여 각 index에 맞게 hash table을 linked list로 생성

* + 1. 사용 변수

FILE \*fp = fopen("opcode.txt", "r"); //열어볼 파일을 가리키는 파일 포인터

int hash\_index = 1; //mnemonic의 hash\_table의 index를 의미.

char temp\_str[30] = { 0 }; //파일에서 입력 받은 한 줄을 저장하는 char 배열

char temp\_val[2] = { 0 }; //opcode에 해당 하는 값을 저장하는 char 배열

char temp\_mnemonic[7] = { 0 }; //mnemonic에 해당하는 문자를 저장하는 char 배열

int i; //반복문에서 temp\_str의 index

int j = 0; //j:반복문에서 temp\_mnemonic을 가리키는 index

int mnemonic\_size=0 //mnemonic의 길이를 나타내는 변수

Hash\* Hash\_Node = (Hash\*)malloc(sizeof(Hash)); //mnemonic과 value를 저장할 hash node

Hash\* cur = table[hash\_index]; //해당 index의 hash table에 노드가 이미 존재하는 경우, 끝에 도달하기 위해 필요한 포인터

* 1. 모듈 이름 : void command\_opcodelist();
     1. 기능

opcodelist가 입력된 경우, 해당 hash table을 형식에 맞게 출력한다.

* + 1. 사용 변수

int i = 0 //hash table 방문을 위해 반복문에서 사용하는 변수

Hash\* cur = table[i]; //현재 Hash 노드를 가리키는 포인터

* 1. 모듈 이름 : void command\_opcode();
     1. 기능

opcode가 입력된 경우, opcode 뒤에 온 mnemonic의 hash index를 계산하여 hash table에서 찾아 해당하는 value를 출력한다. 만일 해당 mnemonic이 없을 경우, 에러 메시지 출력한다.

* + 1. 사용 변수

char temp\_mnemonic[7] = { 0 }; //입력 받은 mnemonic을 저장하는 char 배열

int mnemonic\_num = 1; //mnemonic의 hash\_index를 계산하기 위한 변수

int i = 7; //입력 받은 command의 index

int j = 0, mnemonic\_length = 0; //j는 temp\_mnemonic의 index, mnemonic\_length는 mnemonic의 길이

Hash\* cur = table[mnemonic\_num]; //해당 hash\_index의 hash table을 방문할 때 필요한 포인터

* 1. 모듈 이름 : int distinct\_ascii(char x);
     1. 기능

입력된 메모리 주소 또는 값의 각각 char가 0~9, A~F, a~f, 빈칸 이렇게 중에 있어야한다. 따라서 들어온 char x를 판별하여 이 중 해당 하면 0을, 아니면 1을 리턴하여 에러 처리할 수 있게 해준다.

* + 1. 사용 변수

char x //입력된 command의 각 char 1byte를 의미, x의 아스키코드를 판별함.

1. **전역 변수, 구조체 정의**
   1. typedef struct \_Node

history를 입력받은 경우, 해당 line 번호와 입력 받은 command를 출력해야 한다. 따라서 struct \_Node는 line 번호를 저장할 int index와 입력 받은 command를 저장할 char comm[30], 그리고 다음 노드를 연결할 struct \_Node\* next로 구성되어 있다.

* 1. typedef struct \_Hash

hash table에는 mnemonic과 해당 value 값을 갖고 있어야 한다. 따라서 struct \_Hash는 mnemonic char 배열인 char mnemonic[7]과 해당 value 값인 int value, 그리고 다음 노드를 연결할 struct \_Hash \*next로 구성되어 있다.

* 1. char command[100];

입력되는 명령어를 저장할 char 배열이다.

* 1. int line\_num;

history에서 출력할 라인 번호를 갖고 있는 변수이다.

* 1. int quit\_flag;

main 함수에서 while문 탈출 유무를 판별할 변수이다. q나 quit이 입력된 경우 quit\_flag가 1이 된다.

* 1. int error\_flag;

잘못된 명령어가 들어온 경우, error\_flag가 1이 되어 command\_distinct에서 잘못된 명령어에 대한 에러 처리를 할 수 있게 해준다.

* 1. int memory\_address;

가장 마지막에 출력한 메모리 주소의 다음 메모리 주소 값을 갖는다. dump나 du만 입력될 때는 계속해서 10줄씩 출력해야 하므로 memory\_address에 저장된 주소부터 10줄을 출력하게 된다.

* 1. unsigned char memory[1048576];

메모리 값들을 저장하고 있는 배열이다. FF 값까지 들어가야하므로 unsigned char로 type을 설정해주었다.

* 1. Node\* head;

history linked list의 시작 부분을 가리키고 있는 포인터이다.

* 1. Node \*prev;

History linked list에서 노드를 추가할 때, 가장 마지막에 추가한 이전 노드를 가리키고 있는 포인터이다.

* 1. Hash\* table[20];

hash table을 만들 때, hash table 배열의 각각의 시작 부분을 가리키고 있는 포인터이다.

1. **코드 설명**
   1. 20161598.h

include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/stat.h>

#include <dirent.h>

typedef struct \_Node{

int index;

char comm[30];

struct \_Node\* next;

}Node; //history를 저장할 구조체 Node

Node\* head; //head는 history 제일 처음을 가리킴

Node \*prev; //prev는 이전 Node를 가리킴

typedef struct \_Hash {

char mnemonic[7];

int value;

struct \_Hash\* next;

}Hash; //Hash table을 저장할 구조체 Hash

Hash\* table[20]; //그리고 table이라는 배열에 opcode Hash table 저장.

char command[100]; //입력되는 명령어를 받을 변수.

int line\_num; //history에 출력할 라인 index를 갖는 변수

int quit\_flag; //quit 명령어 입력 유무 판별하는 변수. 1이면 프로그램 종료.

int error\_flag; //에러가 발생햇는지 여부 갖는 변수.

int memory\_address; //마지막에 출력한 메모리 주소갖는 변수

unsigned char memory[1048576]; //메모리 표현하는 배열

void command\_distinct(char\* arr);

void command\_help();

void command\_dir();

void command\_history();

void command\_dump(int num\_flag);

void command\_edit();

void edit\_add\_val(char\* arr, int\* addr, int\* val);

void command\_fill();

void fill\_addr\_val(char\* arr, int\* start, int\* end, int\* val);

void command\_reset();

int dump\_start\_end(char\* arr, int\* start, int\* end);

void print\_memory(int first\_line, int last\_line, int dump\_start, int dump\_end);

void command\_quit();

void make\_hashtb();

void command\_opcodelist();

void command\_opcode();

/\*-------------------------------------\*/

/\* 함수명 : command\_distinct \*/

/\* 목적 : 입력 받은 command 판별 후 \*/

/\* 해당 command 관련 함수 호출 \*/

/\* 반환값 : 없음 \*/

/\*-------------------------------------\*/

void command\_distinct(char\* arr)

{

//error\_flag가 1이면 명령어 이외의 문자가 들어온 경우.

error\_flag = 1;

//입력 받는 command들 node에 저장 후, linked list로 구현.

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

node->index = ++line\_num;

strcpy(node->comm, arr);

node->next = NULL;

if (head == NULL)

head = node;

else

prev->next = node;

//help나 h입력 받은 경우, command\_help() 호출

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

{

//제대로 입력 받은 경우이므로, error\_flag는 0으로 만들어줌.

error\_flag--;

command\_help();

}

//d나 dir 입력 받은 경우, command\_dir() 호출

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

{

error\_flag--;

command\_dir();

}

//q나 quit 입력 받은 경우, command\_quit() 호출

else if (!strcmp(arr, "q") || !strcmp(arr, "quit"))

{

error\_flag--;

command\_quit();

}

//hi나 history 입력 받은 경우, command\_history() 호출

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

{

error\_flag--;

command\_history();

}

//du나 dump 입력 받은 경우, command\_dump() 호출

else if (!strncmp(arr, "du", 2) || !strncmp(arr, "dump ", 4))

{

int dump\_flag = 0; //du나 dump 이외의 비슷한 문자열이 들어온 경우, 1이 됨.

for (int k = 2; k <= 4; k++)

{

if (k == 2 && arr[k] == ' ')

break;

else if (k == 2 && arr[k] == '\0')

{

error\_flag--;

//du만 들어온 경우, num\_flag 1을 파라미터로 넘겨서 해당 메모리 주소부터 10줄 출력.

command\_dump(1);

dump\_flag = 1;

}

else if (k == 2 && arr[k] == 'm')

continue;

else if (k == 3 && arr[k] == 'p')

continue;

else if (k == 4 && arr[k] == '\0')

{

error\_flag--;

//dump만 들어온 경우, num\_flag 1을 파라미터로 넘겨서 해당 메모리 주소부터 10줄 출력.

command\_dump(1);

dump\_flag = 1;

}

else if (k == 4 && arr[k] == ' ')

continue;

else

{

//잘못된 명령어 입력 받은 경우. dump나 du 뒤에 빈칸이 아닌 문자 등이 들어온 경우.

dump\_flag = 1;

break;

}

}

if (!dump\_flag)

{

//dump나 du 뒤에 최소한 start index를 받은 경우.

//number\_flag 0을 파라미터로 넘겨서 start index, end\_index를 계산하게 한다.

error\_flag--;

command\_dump(0);

}

}

//e나 edit을 입력 받은 경우, command\_edit() 호출

else if (!strncmp(arr, "e", 1) || !strncmp(arr, "edit", 4))

{

int edit\_flag = 0; //e나 edit 이외의 비슷한 문자열이 들어온 경우, 1이 됨.

for (int k = 1; k <= 4; k++)

{

if (k == 1 && arr[k] == ' ')

break;

else if (k == 1 && arr[k] == 'd')

continue;

else if (k == 2 && arr[k] == 'i')

continue;

else if (k == 3 && arr[k] == 't')

continue;

else if (k == 4 && arr[k] == ' ')

continue;

else

{

//잘못된 명령어 입력 받은 경우. edit이나 e 뒤에 빈칸이 아닌 문자 등이 들어온 경우.

edit\_flag = 1;

break;

}

}

if (!edit\_flag)

{

//edit이나 e 뒤에 addr index를 받은 경우

error\_flag--;

command\_edit();

}

}

//fill이나 f를 입력 받은 경우, command\_fill() 호출

else if (!strncmp(arr, "f", 1) || !strncmp(arr, "fill", 4))

{

int fill\_flag = 0; //f나 fill 이외의 비슷한 문자열이 들어온 경우, 1이 됨.

for (int k = 1; k <= 4; k++)

{

if (k == 1 && arr[k] == ' ')

break;

else if (k == 1 && arr[k] == 'i')

continue;

else if (k == 2 && arr[k] == 'l')

continue;

else if (k == 3 && arr[k] == 'l')

continue;

else if (k == 4 && arr[k] == ' ')

continue;

else

{

//잘못된 명령어 입력 받은 경우. fill이나 f 뒤에 빈칸이 아닌 문자 등이 들어온 경우.

fill\_flag = 1;

break;

}

}

if (!fill\_flag)

{

//fill이나 f 뒤에 addr index를 받은 경우

error\_flag--;

command\_fill();

}

}

//reset 입력 받은 경우, command\_reset() 호출

else if (!strcmp(arr, "reset"))

{

error\_flag--;

command\_reset();

}

//opcodelist 입력 받은 경우, command\_opcodelist() 호출

else if (!strncmp(arr, "opcodelist", 10))

{

//만일 opcodelist 뒤에 문자가 더 없다면, 즉 제대로 된 경우.

if(arr[10]=='\0'){

error\_flag--;

command\_opcodelist();}

}

//opcode 입력 받은 경우, command\_opcode() 호출

else if (!strncmp(arr, "opcode", 6))

{

//opcode 뒤에 다른 문자 없이 빈칸이 들어와 mnemonic을 입력 받은 경우

if (arr[6] == ' ')

{

error\_flag--;

command\_opcode();

}

}

//error\_flag가 1이면 잘못된 명령어이므로, 해당 노드를 history에서 삭제.

if(error\_flag)

{

free(node);

line\_num--; //history line수를 하나 줄여준다.

//처음에 입력 받고, 잘못된 명령어였던 경우, 다시 head를 null로 설정.

if (prev == NULL)

head = NULL;

return;

}

//prev은 이전을 가리키는 node이므로 현재 노드 삽입 후, 다시 다음 명령어 입력 대기.

prev = node;

}

/\*-------------------------------------\*/

/\* 함수명 : command\_help \*/

/\* 목적 : shell 상에서 사용가능한 \*/

/\* 명령어들을 출력 \*/

/\* 반환값 : 없음 \*/

/\*-------------------------------------\*/

void command\_help()

{

//h나 help 입력된 경우, 모든 명령어 리스트 출력.

printf("\th[elp]\n\td[ir]\n\tq[uit]\n\thi[story]\n\tdu[mp][start, end]\n\te[dit] address, value\n\tf[ill] start, end, value\n\treset\n\topcode mnemonic\n\topcodelist\n");

}

/\*--------------------------------------\*/

/\* 함수명 : command\_dir \*/

/\* 목적 : 현재 디렉터리에 있는 파일들을 \*/

/\* 출력 \*/

/\* 반환값 : 없음 \*/

/\*--------------------------------------\*/

void command\_dir()

{

DIR\* dir\_info = NULL;

struct dirent\* dir\_entry = NULL;

struct stat buf;

int num\_index=0; //출력한 파일의 개수 저장하는 변수

if((dir\_info = opendir("."))==NULL) //디렉토리 open

{

printf("Error! Directory open error\n");

error\_flag = 1;

return;

};

//디렉토리 안에 있는 모든 파일과 디렉토리 출력

while ((dir\_entry = readdir(dir\_info)) != NULL)

{

// '.' 과 '..'는 현재 폴더와 이전 폴더를 가리키므로 제외해서 검색.

if ((strcmp(dir\_entry->d\_name, "..") == 0) || (strcmp(dir\_entry->d\_name, ".") == 0))

continue;

stat(dir\_entry->d\_name, &buf); //첫번째 파라미터에 해당하는 파일 읽음.

printf("\t%s",dir\_entry->d\_name);

if(S\_ISDIR(buf.st\_mode)) //읽어들인 파일이 디렉토리인 경우 '/' 출력

printf("/");

else if(S\_IXUSR & buf.st\_mode) //읽어드린 파일이 실행파일인 경우 '\*' 출력

printf("\*");

num\_index++;

if(num\_index % 4 ==0) //파일 4개 출력시 개행

printf("\n");

}

closedir(dir\_info); //디렉토리 포인터 close

if(num\_index % 4 !=0)

printf("\n");

}

/\*-------------------------------------\*/

/\* 함수명 : command\_history \*/

/\* 목적 : 현재까지 사용한 명령어들을 \*/

/\* 순서대로 번호와 함께 출력 \*/

/\* 반환값 : 없음 \*/

/\*-------------------------------------\*/

void command\_history()

{

//hi나 history 입력된 경우, 현재까지 사용한 명령어들을 순서대로 번호와 함께 출력.

if (head)

{

//cur은 현재 node를 가리킴.

Node\* cur = head;

while (1)

{

int i = 0;

//현재 history node의 line index 출력

printf("\t %d\t", cur->index);

//현재 history node가 갖고 있는 입력어들 전부 출력.

while(cur->comm[i] != '\0')

printf("%c", cur->comm[i++]);

printf("\n");

//cur 다음 node로 옮겨줌.

cur=cur->next;

//linked list 끝에 도달하면 종료.

if (cur == NULL)

break;

}

}

}

/\*-------------------------------------\*/

/\* 함수명 : command\_dump \*/

/\* 목적 : start, end index 유무 판별 후 \*/

/\* print\_memory 함수를 케이스에 맞게 호출\*/

/\* 반환값 : 없음 \*/

/\*-------------------------------------\*/

void command\_dump(int num\_flag)

{

//start, end 번지 없이 dump나 du명령어만 들어온 경우.

if(num\_flag)

{

//메모리 주소에 9 더하면 FFFFF 넘는 경우.

if (memory\_address > 0xFFF60)

{

//difference는 현재부터 0xFFFF0까지 출력후 그 다음에 출력해야할 줄의 수를 저장하는 변수.

int difference = 8 - (0xFFFF0 - memory\_address) / 16;

//현재 memory\_address부터 0xFFFF0까지 출력.

print\_memory(memory\_address, 0xFFFF0, memory\_address, 0xFFFFF);

memory\_address = 0; //memory\_address 0으로 초기화

//memory\_address 0부터 difference를 더한 만큼 메모리 출력.

print\_memory(memory\_address, memory\_address + 16 \* difference, memory\_address, memory\_address + 16 \* difference + 15);

}

else

//메모리 주소에 9 더해도 FFFFF 넘지 않으므로, 현재 메모리 주소부터 10줄 출력.

print\_memory(memory\_address, memory\_address + 16 \* 9, memory\_address, (memory\_address + 10 \* 16) - 1);

}

//start또는 end 번지와 함께 dump나 du명령어가 들어온 경우.

else

{

//dump\_start는 start 번지, dump\_end는 end 번지를 나타내는 변수.

int dump\_start, dump\_end;

int dump\_valid = dump\_start\_end(command, &dump\_start, &dump\_end); //올바르게 둘 다 들어오면 1, start만 들어오면 0, 잘못된 번지 갖으면 -1을 갖는다.

//dump\_start\_end를 통해, 시작 번지와 끝 번지를 숫자로 구함. 둘 다 들어온 경우, 1 return 받아 아래 if문 실행

if (dump\_valid == 1)

//start, end 번지 둘 다 주어진 경우, 시작 번지부터 끝 번지까지 print

print\_memory((dump\_start / 16) \* 16, (dump\_end / 16) \* 16, dump\_start, dump\_end);

//잘못된 메모리 주소를 받은 경우,

else if(dump\_valid == -1)

{

printf("Error : wrong memory index!\n ");

error\_flag=1;

return;

}

//start만 주어진 경우, 해당 line부터 10개 출력.

else

{

//현재 시작 번지의 메모리 주소에 9 더하면 FFFFF 넘는 경우.

if ((dump\_start / 16) \* 16 > 0xFFF60)

{

//시작이 0xFFFFF 넘어 메모리 범위 넘는 경우, error 메시지 출력.

if(dump\_start > 0xFFFFF)

{

printf("Error : Memory index out of bound\n");

error\_flag=1;

return;

}

else{

//메모리 주소에 시작 번지의 메모리 주소 삽입.

memory\_address = (dump\_start / 16) \* 16;

//difference는 현재부터 0xFFFF0까지 출력후 그 다음에 출력해야할 줄의 수를 저장하는 변수.

int difference = 8 - (0xFFFF0 - memory\_address) / 16;

//현재 memory\_address부터 0xFFFF0까지 출력.

print\_memory(memory\_address, 0xFFFF0, dump\_start, 0xFFFFF);

memory\_address = 0; //memory\_address 0으로 초기화

//memory\_address 0부터 difference를 더한 만큼 메모리 출력.

print\_memory(memory\_address, memory\_address + 16 \* difference, memory\_address, memory\_address + 16 \* difference + 15);}

}

else

//메모리 주소에 9 더해도 FFFFF 넘지 않으므로, 현재 시작 번지부터 10줄 출력.

print\_memory((dump\_start / 16) \* 16, (dump\_start / 16) \* 16 + 16 \* 9, dump\_start, ((dump\_start / 16) + 10) \* 16 - 1);

}

}

}

/\*-----------------------------------------------\*/

/\* 함수명 : dump\_start\_end \*/

/\* 목적 : dump 명령어 뒤에 start만 있는지 \*/

/\* start end 모두 있는지 판별 \*/

/\* 반환값 : 모두 있으면 1, start만 존재시 0 반환 \*/

/\*-------------------------------------\*/

int dump\_start\_end(char\* arr, int \*start, int \*end)

{

char start\_num[10] = {0};

char end\_num[10] = {0};

int i, j=0;

int number\_flag = 0; //숫자가 2개 들어오면 1, 1개면 0

//du [start,end]인 경우, index 3부터 시작, dump [start, end]인 경우, index 6부터 시작

if (!strncmp(arr, "dump", 4))

i = 5;

else

i = 3;

while (arr[i] != '\0')

{

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

{

number\_flag = 1;

j = 0;

i++;

continue;

}

//,가 안들어와서 아직 첫번째 메모리 주소인 경우, number\_flag가 0

if (!number\_flag)

{

if (distinct\_ascii(arr[i]))

{

error\_flag = 1;

return -1;

}

start\_num[j++] = arr[i];}

else{

//if arr[i]가 0~9, or a~f, A~F, 빈칸 아니면 에러

if(distinct\_ascii(arr[i])){

error\_flag=1;

return -1;

}

end\_num[j++] = arr[i];}

i++;

}

//시작 주소 \*start에 저장

\*start = strtol(start\_num, NULL, 16);

if (number\_flag)

{

//끝 주소 \*end에 저장

\*end = strtol(end\_num, NULL, 16);

return 1;

}

else

return 0;

}

/\*---------------------------------------\*/

/\* 함수명 : print\_memory \*/

/\* 목적 : 입력 받은 start부터 end번지까지\*/

/\* 내용 출력 \*/

/\* 반환값 : 없음 \*/

/\*---------------------------------------\*/

void print\_memory(int first\_line, int last\_line, int dump\_start, int dump\_end)

{

//메모리 주소 범위 벗어날 시에, 에러 메시지 출력

if (dump\_start < 0 || dump\_end >0xFFFFF || dump\_start > 0xFFFFF || dump\_end < 0)

{

printf("Error : Memory index out of bound!\n");

error\_flag = 1;

return;

}

//끝 주소가 시작 주소보다 큰 경우 에러 메시지 출력

else if (dump\_start > dump\_end)

{

printf("Error : Start index is bigger than End index!\n");

error\_flag = 1;

return;

}

int line\_start = dump\_start;

for (int hex\_memory\_line = first\_line; hex\_memory\_line <= last\_line; hex\_memory\_line+=16)

{

//메모리 주소 5자리 출력

printf("%.5X ", hex\_memory\_line);

//맨 마지막 줄 출력시, 해당 memory 주소 다음 주소 memory\_address 변수에 저장.

if (hex\_memory\_line == last\_line)

memory\_address = last\_line + 16;

//메모리 주소가 FFFF0 넘어가면 0으로 초기화

if(memory\_address > 0xFFFF0)

memory\_address=0;

int hex\_line = hex\_memory\_line / 16;

//첫번째 메모리 주소 일 때

if (first\_line/16 == hex\_line) {

//시작 지점이 메모리 주소의 첫 부분이 아닌 경우, 해당 부분은 빈칸 출력

if (dump\_start != hex\_line \* 16)

{

for (int k = hex\_line\*16; k < dump\_start; k++)

printf(" ");

}

}

//마지막 메모리 주소가 아닐 때는, 16개 채워서 끝까지 출력

if (hex\_line != last\_line/16) {

//해당 메모리 내용 하프 바이트로 출력

for (int k = line\_start; k <= hex\_line \* 16 + 15; k++)

printf("%.2X ", memory[k]);

}

//마지막 메모리 주소일 때

else if (hex\_line == last\_line/16)

{

//끝 주소까지 하프바이트 단위 출력

for (int k = line\_start; k <= dump\_end; k++)

printf("%.2X ", memory[k]);

//메모리 주소 끝까지 빈칸 출력

for (int k = dump\_end + 1; k < hex\_line \* 16 + 16; k++)

printf(" ");

}

printf("; ");

//첫번째 메모리 주소 일 때

if (first\_line/16 == hex\_line) {

//시작 지점이 메모리 주소의 첫 부분이 아닌 경우, 해당 부분은 0번 아스키 코드 출력.

if (dump\_start != hex\_line \* 16)

{

for (int k = hex\_line\*16; k < dump\_start; k++)

printf(".");

}

}

//마지막 메모리 주소가 아닐 때는, 16개 채워서 끝까지 출력

if (hex\_line != last\_line/16) {

//해당 메모리 내용 하프 바이트로 출력

for (int k = line\_start; k <= hex\_line \* 16 + 15; k++)

{

if ((int)memory[k] >= 32 && (int)memory[k] <= 126)

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

else

printf(".");

}

}

//마지막 메모리 주소일 때

else if (hex\_line == last\_line/16)

{

for (int k = line\_start; k <= dump\_end; k++)

{

//범위 안에 들어가면 해당 내용 (char)출력

if ((int)memory[k] >= 32 && (int)memory[k] <= 126)

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

//아닐 경우, ‘.’ 출력

else

printf(".");

}

//메모리 주소 끝까지 ‘.’출력

for (int k = dump\_end + 1; k < hex\_line \* 16 + 16; k++)

printf(".");

}

printf("\n");

//line\_start에 다음 메모리 주소의 첫번째 위치 index 대입

line\_start = (++hex\_line) \* 16;

}

}

/\*-------------------------------------\*/

/\* 함수명 : command\_edit \*/

/\* 목적 : 입력 받은 주소에 해당하는 \*/

/\* 값을 대입 \*/

/\* 반환값 : 없음 \*/

/\*-------------------------------------\*/

void command\_edit()

{

int addr = 0;

int val = 0;

edit\_add\_val(command, &addr, &val);

//value가 범위 벗어난 경우, 에러 메시지 출력

if (val < 0 || val >255)

{

printf("Error : Value Error!\n");

error\_flag = 1;

return;

}

//주소가 범위 벗어난 경우, 에러 메시지 출력

if (addr < 0 || addr>0xFFFFF) {

printf("Error : Address out of bound!\n");

error\_flag = 1;

return;

}

//올바른 명령어 입력 시 메모리 주소에 값 대입

if(!error\_flag)

memory[addr] = val;

}

/\*-------------------------------------\*/

/\* 함수명 : edit\_add\_val \*/

/\* 목적 : edit command 뒤에 입력된 \*/

/\* 주소, 값 판별 \*/

/\* 반환값 : 없음 \*/

/\*-------------------------------------\*/

void edit\_add\_val(char \*arr, int \*addr, int \*val)

{

int i, number\_flag=0, j=0;

char edit\_value[10] = { 0 };

char edit\_address[10] = { 0 };

//edit 뒤에 숫자 입력 받은 경우, index 4부터, e만 받은 경우 index 2부터 시작.

if (!strncmp(command, "edit", 4))

i = 5;

else

i = 2;

while (arr[i] != '\0')

{

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

{

number\_flag++;

j = 0;

i++;

continue;

}

//,가 안들어와서 메모리 주소인 경우, number\_flag가 0

if (!number\_flag)

{

//if arr[i]가 0~9, or a~f, A~F 아니면 error

if (distinct\_ascii(arr[i]))

{

error\_flag = 1;

return;

}

edit\_address[j++] = arr[i];

}

else if(number\_flag == 1 || number\_flag == 2)

{

//if arr[i]가 0~9, or a~f, A~F 아니면 error

if (distinct\_ascii(arr[i]))

{

error\_flag = 1;

return;

}

edit\_value[j++] = arr[i];

number\_flag = 2; }

else

{

//값을 더 많이 받게 된 경우 에러 처리

error\_flag = 1;

return;

}

i++;

}

//\*addr에 주소 16진수로 대입

\*addr = strtol(edit\_address, NULL, 16);

//만일 뒤에 값이 입력 안된경우, \*val에 -1 대입

if (number\_flag != 2 && 0 == strtol(edit\_value, NULL, 16))

\*val = -1;

//제대로 값이 온 경우, \*val에 16진수 value 대입

else

\*val = strtol(edit\_value, NULL, 16);

}

/\*---------------------------------------\*/

/\* 함수명 : command\_fill \*/

/\* 목적 : 입력 받은 start에서 end번지에\*/

/\* value 값 전부 대입 \*/

/\* 반환값 : 없음 \*/

/\*---------------------------------------\*/

void command\_fill()

{

int start, end, val=-1;

fill\_addr\_val(command, &start, &end, &val);

//value가 범위 벗어난 경우, 에러 메시지 출력

if (val < 0 || val >255)

{

printf("Error : Value error!\n");

error\_flag = 1;

return;

}

//주소가 범위 벗어난 경우, 에러 메시지 출력

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

printf("Error : Address out of bound!\n");

error\_flag = 1;

return;

}

//시작 주소는 나오지만, 끝 주소가 나오지 않은 경우, 에러 메시지 출력

else if (end == 0 && start != 0)

{

printf("Error : Need end index!\n");

error\_flag = 1;

return;

}

//시작이 끝 보다 큰 경우, 에러 메시지 출력ㄴ

else if (start > end)

{

printf("Error : Start index is bigger than End index!\n");

error\_flag = 1;

return;

}

//올바른 명령어 입력 시, start-end index에 값 대입

if(!error\_flag){

for(int k=start; k<=end; k++)

memory[k] = val;}

}

/\*-------------------------------------\*/

/\* 함수명 : fill\_add\_val \*/

/\* 목적 : fill command 뒤에 입력된 start, end \*/

/\* 주소, 값 판별 \*/

/\* 반환값 : 없음 \*/

/\*-------------------------------------\*/

void fill\_addr\_val(char\* arr, int\*start, int\*end, int\* val)

{

int i, number\_flag = 0, j = 0;

char fill\_val[10] = { 0 };

char fill\_start[10] = { 0 };

char fill\_end[10] = { 0 };

//edit 뒤에 숫자 입력 받은 경우, index 4부터, e만 받은 경우 index 2부터 시작.

if (!strncmp(command, "fill", 4))

i = 5;

else

i = 2;

while (arr[i] != '\0')

{

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

{

number\_flag++;

j = 0;

i++;

continue;

}

//,가 안들어와서 메모리 start 번지인 경우, number\_flag가 0, 그 다음에는 end 번지인 경우 number\_flag가 1, value인 경우 2

if (number\_flag == 0)

{

//if arr[i] 가 0~9, or a~f, A~F 아니면 error

if (distinct\_ascii(arr[i]))

{

error\_flag = 1;

return;

}

fill\_start[j++] = arr[i];

}

else if (number\_flag == 1)

{

//if arr[i] 가 0~9, or a~f, A~F 아니면 error

if (distinct\_ascii(arr[i]))

{

error\_flag = 1;

return;

}

fill\_end[j++] = arr[i];

}

else if(number\_flag ==2)

{

//if arr[i] 가 0~9, or a~f, A~F 아니면 error

if (distinct\_ascii(arr[i]))

{

error\_flag = 1;

return;

}

fill\_val[j++] = arr[i];

}

else

{

//값을 더 받게 된 경우 에러 처리

error\_flag=1;

return;

}

i++;

}

//시작, 끝 번지에 해당 주소 값 16진수로 변환 후 대입

\*start = strtol(fill\_start, NULL, 16);

\*end = strtol(fill\_end, NULL, 16);

//start, end 번지는 들어왔으나, val값이 입력되지 않은 경우.

if (number\_flag == 1 && strtol(fill\_val, NULL, 16) == 0)

\*val = -1;

//제대로 입력된 경우, \*val에 넣을 값 16진수 변환 후 대입

else

\*val = strtol(fill\_val, NULL, 16);

}

/\*-------------------------------------\*/

/\* 함수명 : command\_reset \*/

/\* 목적 : 메모리 전부를 0으로 \*/

/\* 변경시킨다 \*/

/\* 반환값 : 없음 \*/

/\*-------------------------------------\*/

void command\_reset()

{

//memset함수를 통해 메모리 전부를 (char)0으로 변경.

memset(memory, (char)0, sizeof(memory));

}

/\*-------------------------------------\*/

/\* 함수명 : command\_quit \*/

/\* 목적 : sicsim을 종료하며, 저장했던 \*/

/\* linked\_list 메모리 해제 \*/

/\* 반환값 : 없음 \*/

/\*-------------------------------------\*/

void command\_quit()

{

//quit\_flag 1로 바꾸어 main 함수 while 탈출하게 함.

quit\_flag = 1;

//linked\_list가 존재하는 경우, 하나씩 방문하며 메모리 해제.

if(head)

{

Node\* cur = head;

while (1)

{

if (cur == NULL)

break;

Node\* prev = cur;

cur = cur->next;

free(prev);

}

}

//Hash\_Table이 존재하는 경우, 전부 메모리 해제.

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

{

if (table[i])

{

Hash\* cur = table[i];

while (1)

{

if (cur == NULL)

break;

Hash\* prev = cur;

prev = cur;

cur = cur->next;

free(prev);

}

}

}

}

/\*---------------------------------------\*/

/\* 함수명 : make\_hashtb \*/

/\* 목적 : opcode.txt 파일을 읽어 들여서 \*/

/\* linked\_list로 hash table 구현 \*/

/\* 반환값 : 없음 \*/

/\*---------------------------------------\*/

void make\_hashtb() {

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

while (1)

{

int hash\_index = 1; //hash\_table의 index를 의미.

char temp\_str[30] = { 0 };

char temp\_val[2] = { 0 };

char temp\_mnemonic[7] = { 0 };

int i;

if (fgets(temp\_str, 30, fp) == NULL)

break;

//mnemonic의 value 값 temp\_val에 copy

for (i = 0; i <= 1; i++)

temp\_val[i] = temp\_str[i];

while (temp\_str[i] < (char)65)

i++;

int j = 0, mnemonic\_size=0;

//mnemonic string을 temp\_mnemonic에 copy

for (; temp\_str[i] >= (char)65 && temp\_str[i] <= (char)90; i++)

{

temp\_mnemonic[j++] = temp\_str[i];

hash\_index += (int)temp\_str[i];

mnemonic\_size++;

}

hash\_index += mnemonic\_size\*19;

//hash\_table의 size가 20이므로 앞서 나온 모든 명령어의 각 알파벳 정수 값을 더한 뒤,

//명령어의 길이에 19를 곱한 값을 더하고, 20으로 나눈 나머지로 hash \_index 설정

hash\_index %= 20;

Hash\* Hash\_Node;

Hash\_Node = (Hash\*)malloc(sizeof(Hash));

//현재 Hash 노드에 value와 mnemonic 대입

strcpy(Hash\_Node->mnemonic, temp\_mnemonic);

Hash\_Node->value = strtol(temp\_val, NULL, 16);

Hash\_Node->next = NULL;

//만일 table[hash\_index]에 아무것도 없으면 현재 노드 가리키게 함

if (table[hash\_index] == NULL)

table[hash\_index] = Hash\_Node;

else

{

Hash\* cur = table[hash\_index];

//linked list 끝까지 도달하게 하는 반복문

while (cur->next != NULL)

cur = cur->next;

//linked list 끝에 현재 노드 삽입

cur->next = Hash\_Node;

}

}

}

/\*-------------------------------------\*/

/\* 함수명 : command\_opcodelist \*/

/\* 목적 : opcode Hash Table의 내용을 \*/

/\* 형식에 맞게 출력. \*/

/\* 반환값 : 없음 \*/

/\*-------------------------------------\*/

void command\_opcodelist() {

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

{

//hash table의 index 출력

printf("\t%d : ", i);

if (table[i])

{

Hash\* cur = table[i];

//Hash\* prev = NULL;

//모든 hash 노드의 mnemonic과 value 값 출력

while (1)

{

printf("[%s,%.2X]", cur->mnemonic, cur->value);

//prev = cur;

cur = cur->next;

if (cur == NULL)

break;

printf(" -> ");

}

printf("\n");

}

}

}

/\*-------------------------------------\*/

/\* 함수명 : command\_opcode \*/

/\* 목적 : 명령어를 입력하면 해당하는 \*/

/\* opcode를 출력. \*/

/\* 반환값 : 없음 \*/

/\*-------------------------------------\*/

void command\_opcode() {

char temp\_mnemonic[7] = { 0 };

int mnemonic\_num = 1;

int i = 7; //i는 opcode 다음 빈칸 다음 index인 7부터 시작.

int j = 0, mnemonic\_length = 0;

//입력 받은 command의 mnemonic 값 temp\_mnemonic에 copy

while (command[i] != '\0')

{

temp\_mnemonic[j++] = command[i];

mnemonic\_num += (int)command[i++];

mnemonic\_length++;

}

//입력 받은 command의 mnemonic 값의 hash index 계산

mnemonic\_num += mnemonic\_length \* 19;

mnemonic\_num %= 20;

Hash\* cur = table[mnemonic\_num];

while (1)

{

if (cur == NULL)

{

//대응하는 mnemonic이 hash table에 없어 찾지 못한 경우, 에러메시지 출력

printf("Error ! No such Mnemonic on the opcodelist.\n");

error\_flag = 1;

break;

}

if (!(strcmp(cur->mnemonic, temp\_mnemonic)))

{

//찾은 경우, 해당 값 출력

printf("opcode is %.2X\n", cur->value);

break;

}

cur = cur->next;

}

}

/\*-------------------------------------\*/

/\* 함수명 : distinct\_ascii \*/

/\* 목적 : 주소, 값에 해당하는 char 값이 들어왔는지 \*/

/\* 판별 \*/

/\* 반환값 : A~F, a~f, 0~9, blank인 경우 0 아니면 1 \*/

/\*-------------------------------------\*/

int distinct\_ascii(char x){

//만일 x가 빈칸일 때

if(x==(char)32)

return 0;

else if(x < (char)48)

return 1;

else if(x>=(char)58 && x <= (char)64)

return 1;

else if (x >= (char)71 && x <= (char)96)

return 1;

else if (x >= (char)103)

return 1;

//만일 x가 0~9, A~F, a~f, return 0

else

return 0;

}

* 1. 20161598.c

#include "20161598.h"

int main()

{

//opcode에 관련된 hash table 생성.

make\_hashtb();

//quit 입력 받기 전까지 계속 반복.

while (!quit\_flag) {

//q 또는 quit이 입력되기 전까지 계속 입력 받기 반복.

//i는 입력받는 명령어 배열에서 index

int i = 0;

printf("sicsim> ");

if(fgets(command, 100, stdin)!=NULL)

{

//문자 끝에 개행 대신 '\0' 대입.

while (command[i]!='\n')

i++;

command[i] = '\0';

//command 구분하기 위해 command\_distinc 함수 호출.

command\_distinct(command);

}

}

return 0;

}