과목 명: 시스템프로그래밍

담당 교수 명: 소 정 민

<<Assignment 1>>

**서강대학교 컴퓨터공학과**

**[학번] 20171665**

**[이름] 이선호**

목 차

1. 프로그램 개요 4

2. 프로그램 설명 5

2.1 프로그램 흐름도 5

3. 모듈 정의 6

3.1 모듈 이름: main 6

3.1.1 기능 6

3.1.2 사용 변수 6

3.2 모듈 이름: RemoveBlank 6

3.2.1 기능 6

3.2.2 사용 변수 6

3.3 모듈이름: HexStringToDec 7

3.3.1 기능 7

3.3.2 사용변수 7

3.4 모듈이름: ReadShellCommand 7

3.4.1 기능 7

3.4.2 사용변수 7

3.5 모듈이름: HelpCommand 8

3.5.1 기능 8

3.5.2 사용변수 8

3.6 모듈 이름: DirCommand 8

3.1.1 기능 8

3.1.2 사용 변수 8

3.7 모듈 이름: RecordPreviousCommand 8

3.2.1 기능 8

3.2.2 사용 변수 8

3.8 모듈이름: HistoryCommand 8

3.3.1 기능 8

3.3.2 사용변수 8

3.9 모듈이름: DeleteHistory 9

3.5.1 기능 9

3.5.2 사용변수 9

3.10 모듈 이름: DumpCommand 9

3.1.1 기능 9

3.1.2 사용 변수 9

3.11 모듈 이름: EditCommand 9

3.2.1 기능 9

3.2.2 사용 변수 9

3.12 모듈이름: FillCommand 9

3.3.1 기능 9

3.3.2 사용변수 9

3.13 모듈 이름: ResetCommand 10

3.2.1 기능 10

3.2.2 사용 변수 10

3.14 모듈이름: HashFunction 10

3.3.1 기능 10

3.3.2 사용변수 10

3.15 모듈이름: OpcodeCommand 10

3.3.1 기능 10

3.3.2 사용변수 10

3.16 모듈 이름: OpcodeListCommand 10

3.2.1 기능 10

3.2.2 사용 변수 10

3.17 모듈이름: DeleteHashTable 11

3.3.1 기능 11

3.3.2 사용변수 11

3.18 모듈이름: CreateHashTable 11

3.3.1 기능 11

3.3.2 사용변수 11

4. 전역 변수 정의 11

4.1 typedef struct Node NODE 11

4.2 typedef struct Hash HASH 12

4.3 HASH HashTable[20] 12

4.4 char DumpMemory[0x100000] 12

4.5 int LastAddress 12

5. 코드 12

5.1 20171665.h 12

5.2 20171665.c 13

# 프로그램 개요

SIC(Simplified Instruction Computer) machine의 기본적인 동작과 shell 환경에서 실행하는command를 이해하기 위해 1MB의 가상 메모리를 할당하고 사용자로부터 이와 관련한command line을 입력받아서 각 command에 맞는 기능을 수행한다. 또한object code가 저장되는 memory space를 할당하고, opcode mnemonic와 그에 대응되는 opcode를 찾을 수 있는 opcode table을 생성하는 프로그램을 제작한다.

# 프로그램 설명

## 프로그램 흐름도

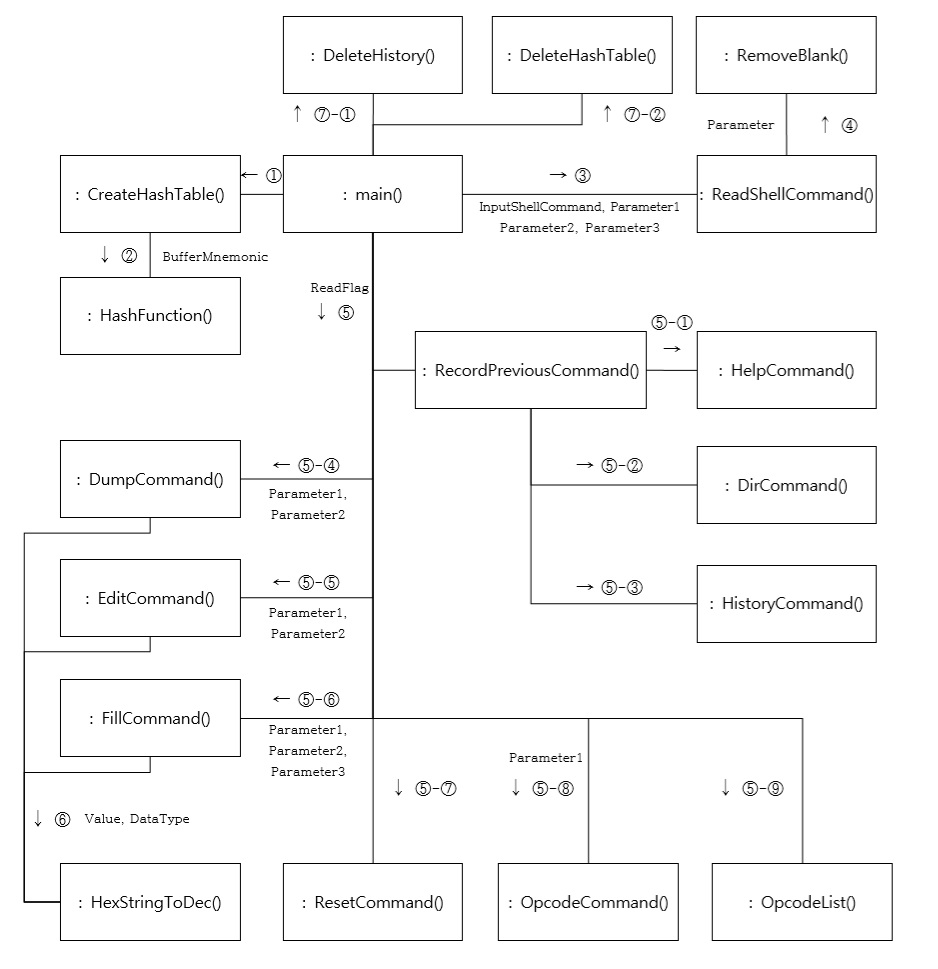


그림 1> 프로그램 흐름도

이 프로그램에서 각 sub-routine들이 어떻게 상호작용하고 실행되는지를 프로그램 흐름도로 그렸다. 프로그램 처음 실행할 때 main 함수에서 CreateHashTable 함수를 호출한다. 이 함수에서는 opcode.txt 파일을 읽어서 opcode mnemonic을 인수로 하여 HashFunction 함수에서 hash value를 구하여 hash table을 완성한다. 이후 사용자로부터 quit command를 입력받기 전까지 command를 매번 입력받아 그에 맞는 sub-routine을 실행하기 위해 while loop을 돈다. 사용자로부터 commad line을 입력받아서 command와 parameter들의 공백을 제거하여 분리하는 작업을 ReadShellCommand와 RemoveBlank 함수에서 실행한다. 이를 통해 얻은 command 에 일대일 대응되는 값인 ReadFlag에 따라switch문을 통해 어떠한 sub-routine을 실행할지 결정한다. ReadFlag가 HelpCommand, DirCommand, HistoryCommand 함수에 대응될 경우 우선 RecordPreviousCommand 함수를 실행하여 이전 command 내역을 저장하는 linked list에 저장하고 나서 실행한다. 이 외의DumpCommand, EditCommand, FillCommand, ResetCommand, OpcodeCommand, OpcodeList 함수는 필요한 개수의 Parameter들을 인수로 전달받아서 실행한다. 특히, DumpCommand, EditCommand, FillCommand를 실행할 때는 memory address 또는 value에 해당하는 값을 16진수에서 10진수로 변환해야 하므로 HexStringToDec 함수를 호출하여 integer 또는 character data type에 맞는 10진수 값을 구한다. 사용자로부터 quit command를 입력받으면 DeleteHistory와 DeleteHashTable 함수를 실행하여 이전에 저장한 command 내역과 hash table을 생성하기 위해 할당한 메모리를 해제한다.

# 모듈 정의

## 모듈 이름 : main()

### 기능

CreateHashTable 함수를 호출해 hash table을 만든 후, ReadShellCommand를 통해 사용자의 입력을 받아서 어떠한 command에 해당하는 모듈을 수행할지를 결정한다. 사용자가 ‘quit’ command를 입력하면 이전에 저장된 command 내역과 hash table을 각각 DeleteHistory와 DeleteHashTable 함수를 호출함으로써 메모리를 해제한다.

### 사용 변수

InputShellCommand – 사용자의 입력을 그대로 RecordPreviousCommand 함수를 통해 저장하기 위해 사용한다.

Parameter1 – 사용자의 command line 입력에서 첫 번째로 전달받은 인자다.

Parameter2 – 사용자의 command line 입력에서 두 번째로 전달받은 인자다.

Parameter3 – 사용자의 command line 입력에서 세 번째로 전달받은 인자다.

ReadFlag – ReadShellCommand 함수의 반환값이며, 이를 가지고 사용자의 입력에 따라 어떠한 command에 해당하는 함수를 수행할지를 결정하기 위해 사용한다.

## 모듈 이름: RemoveBlank(char \*\*Parameter)

### 기능

ReadShellCommand 함수를 통해 사용자의 입력에서 전달받은 Parameter 앞에 위치한 스페이스바나 Tab 문자를 지운다.

### 사용 변수

Index – 공백을 지운 후의 Parameter를 구하기 위해 사용한다.

TempIndex – 공백을 지우기 전의 Parameter에서 공백을 제거하기 위해 사용한다.

## 모듈이름: HexStringDec(char \*Parameter, short DataType)

### 기능

ReadShellCommand 함수를 통해 사용자의 입력에서 전달받은16진수의 Parameter 값을 strtol 함수를 통해 10진수의 integer로 변환한다. 16진수에 해당하지 않는 값 또는 변환하고자 하는 data type의 범위를 초과하는 값이 입력으로 들어올 경우 -1를 반환한다.

### 사용변수

Buffer – 10진수로 바꾼 값을 함수의 return값으로 반환하기 위해 사용한다.

EndPointer – strtol 함수를 사용할 때 정상적인 16진수의 입력값이 들어왔는지를 확인하기 위해 사용한다.

errno - 10진수로 변환한 값이 변환하고자 하는 data type의 범위를 초과하는지 확인하기 위해 사용한다.

## 모듈이름: ReadShellCommand(char \*\*InputshellCommand, char \*\*Parameter1, char \*\*Parameter2, char \*\*Parameter3)

### 기능

사용자로부터 받은 입력이 정해진 형식대로 정확히 입력되었는지 확인한다. 또한, 유효한 command와 parameter들을 추출하고, parameter가 총 몇 개 입력되었는지 확인한다. RemoveBlank 함수를 호출하여 각 parameter 메모리 앞에 입력받아 저장된 공백을 제거한다. Switch 문을 사용하여 command에 대응하는 mnemonic value를 return 값으로 반환한다.

### 사용변수

BufferCounter – 사용자로부터 받은 입력의 문자열 길이가 몇인지를 값으로 갖는다.

ResultValue – Command에 대응되는 mnemonic value를 return 값으로서 반환하기 위해 사용한다.

Flag – Parmeter가 총 몇 개 입력되었는지 개수를 저장한다.

Invalid – 유효하지 않은 입력이 들어올 경우 경고문을 출력하기 위해 사용한다.

CommandCounter – Command의 문자열 길이가 몇인지를 값으로 갖는다.

ParameterCounter1 – 첫 번째 인자의 문자열 길이가 몇인지를 값으로 갖는다.

ParameterCounter2 – 두 번째 인자의 문자열 길이가 몇인지를 값으로 갖는다.

ParameterCounter3 – 세 번째 인자의 문자열 길이가 몇인지를 값으로 갖는다.

BufferChar – 사용자로부터 입력을 받을 때 character 단위로 입력받아 command 또는 해당되는 parameter문자열에 저장하기 위해 사용한다.

BeforeChar – 사용자로부터 입력받을 때 바로 이전에 입력받은 character를 저장하기 위해 사용한다. 유효하지 않은 입력이 들어왔는지를 확인하기 위해서다.

BufferCommand – 입력받은 Command를 저장하기 위해 사용한다.

## 모듈이름: HelpCommand()

### 기능

사용자가 입력할 수 있는 유효한 command와 parameter 형식을 모두 출력한다.

### 사용변수

없음

## 모듈이름: DirCommand()

### 기능

현재 실행 파일이 위치하고 있는 디렉토리 내의 모든 파일과 폴더를 읽어서 그 목록을 출력한다. 폴더와 실행 가능한 파일을 구별할 수 있도록 각각 ‘/’, ‘\*’ character를 폴더 또는 파일명 뒤에 넣어서 출력한다.

### 사용변수

DirectoryInfo –현재 디렉토리의 정보를 갖고 있는 디렉토리 포인터 변수이다.

DirectoryEntry – DirectoryInfo에 해당하는 디렉토리의 폴더 또는 파일을 하나씩 읽어들여서 그에 대한 정보를 확인할 때 사용한다.

BufferEntry - DirectoryEntry에서 갖고 있는 폴더 또는 파일의 정보를 멤버 변수로 추출하여 폴더인지 아닌지 혹은 실행 가능한 파일인지 아닌지 등을 확인하기 위해 사용한다.

## 모듈이름: RecordPreviousCommand(char \*InputShellCommand)

### 기능

사용자로부터 받은 입력 중에서 valid한 command를 linked list에 기록한다. Linked list가 생성되지 않았으면 첫 node를 생성하고, 이미 생성된 상태이면 맨 마지막 node에 linking하여 새로운 node를 추가한다.

### 사용변수

TempNode – Valid한 command를 기록하기 위해 새롭게 생성되는 node이다.

PreviousNode – 맨 마지막 tail node에 TempNode를 linking하기 위해 linked list를 탐색하면서 tail node를 찾기 위해 사용한다.

## 모듈이름: HistoryCommand()

### 기능

RecordPreviousCommand 함수에 의해 저장된 사용자의 이전 valid command들을 목록으로 모두 출력한다. Linked list를 탐색하면서 모든 node의 command data를 출력한다.

### 사용변수

TempNode – Linked list의 첫 번째 node부터 tail node까지 탐색하면서 node들의 command 데이터를 출력하기 위해 사용한다.

## 모듈이름: DeleteHistory()

### 기능

사용자로부터 ‘quit’ command를 입력받아서 프로그램을 종료할 때, 이전에 저장된 command들을 모두 free하여 메모리를 해제한다.

### 사용변수

TempNode – Linked list의 첫 번째 node부터 tail node까지 탐색하면서 이전 node의 메모리를 해제하기 위해 사용한다.

DeleteNode – 메모리를 해제하고자 하는 node이다.

## 모듈이름: DumpCommand(char \*Start, char \*End)

### 기능

SIC에서 설정된 1MB 메모리에서 사용자가 입력한 parameter에 대응되는 address 범위와 그에 해당하는 1byte 메모리 value들을 출력한다. Start parameter 또는 end parameter가 valid하게 입력되지 않으면 error를 출력한다. 1줄에는 16개의 byte 주소와 value를 출력하고자 address를 16으로 나누었을 때 나머지가 0이 되는 시점에서 1byte에 대응되는 character value를 아스키 코드로 출력하고 줄바꿈 문자를 출력한다.

### 사용변수

i, j – 주소와 1byte value, 그리고 이에 대응되는 아스키 코드를 한 줄당 16바이트로 출력하는 형식에 맞추고자 사용한다.

Counter – 출력할 byte의 개수를 갖는다. 사용자가 아무런 parameter를 입력하지 않으면 기본적으로 값을 160으로 갖는다.

BufferStartParameter – 사용자가 입력한 첫 번째 parameter이며, 출력을 시작할 메모리 주소이다.

BufferEndParameter – 사용자가 입력한 두 번째 parameter이며, 출력을 끝낼 메모리 주소이다.

Buffer – 1 byte 메모리 value에 해당하는 아스키 코드를 출력하기 위해 사용한다.

## 모듈이름: EditCommand(char \*Address, char \*Value)

### 기능

사용자로부터 값을 수정하고자 하는 1 byte의 메모리 address와 바꾸고자 하는 value를 입력받은 데이터를 갖고 배열로 접근해 직접 수정한다.

### 사용변수

BufferAddress – 사용자로부터 입력받은 1byte의 16진수 address를 10진수로 변환한 값을 갖는다.

BufferValue – 사용자로부터 입력받은 16진수 value를 10진수로 변환한 값을 갖는다.

## 모듈이름: FillCommand(char \*Start, char \*End, char \*Value)

### 기능

사용자로부터 값을 수정하고자 하는 1 byte의 메모리 address 범위와 바꾸고자 하는 value를 입력받은 데이터를 갖고 배열로 접근해 주어진 address 범위에서 for loop을 돌면서 수정한다. Valid한 주소 범위인지 판단하기 위해 start parameter와 end parameter의 값과 대소 관계를 비교한다. Invalid한 주소 범위이면 error를 출력한다.

### 사용변수

BufferStartParameter – 값을 수정하고자 하는 1 byte들의 주소 범위의 시작 address이다.

BfuferEndParamter – 값을 수정하고자 하는 1 byte들의 주소 범위의 끝 address이다.

Index – 사용자로부터 입력받은 주소 범위의 메모리 value를 for loop을 돌면서 수정할 때 참조하게 되는 인덱스를 의미한다.

BufferValue – 수정하고자 하는16진수 값을 10진수로 변환한 값을 갖는다.

## 모듈이름: ResetCommand()

### 기능

‘reset’ command를 사용자로부터 입력받으면 SIC에서 설정한 모든 1MB의 메모리 값을 0으로 초기화한다.

### 사용변수

없음

## 모듈이름: HashFunction(char \*Mnemonic)

### 기능

사용자로부터 찾고나 하는 opcode mnemonic을 입력받거나 opcodelist를 저장할 hash table을 생성할 때 이에 일대일 대응되는 hash value를 구해준다. Opcode mnemonic의 각 자리 character 값을 integer data type으로 변환한 10진수 값을 19로 나눈 나머지들의 합을 다시 20으로 나눠서 hash value를 구한다.

### 사용변수

Index – Opcode mnemonic의 각 자리 character를 하나씩 참조하기 위해 사용한다.

ReturnValue – 최종적으로 구한 opcode mnemonic의 hash value를 갖는다.

## 모듈이름: OpcodeCommand(char \*Mnemonic)

### 기능

HashFunction 함수를 통해 구한 hash value를 사용하여 hash table를 탐색하면서 hash table에 저장된mnemonic의 opcode를 구한다. Linked list의 자료구조를 갖는 hash table에서 hash value index에 해당하는 원소를 탐색하고, 만일 찾고자 하는 opcode mnemonic이 없으면 error를 출력한다.

### 사용변수

HashCode – Hash function에 의해 구한 mnemonic의 hash value를 값으로 갖는다.

TempHash – Linked list 를 자료구조로 갖는 hash value를 index로 갖는 원소에서 첫 node에서 tail node까지 탐색하기 위해 사용한다.

## 모듈이름: OpcodeListCommand()

### 기능

프로그램을 처음 실행할 때, opcode.txt에서 읽은 opcode 데이터를 가지고 HashFunction를 사용하여 생성한 hash table를 출력한다.

### 사용변수

Index – Linked list에서 첫 번째 node에 해당되는 hash table의 20개의 원소를 탐색할 때 필요한 index이다.

TempHash – Linked list 를 자료구조로 갖는 hash value를 index로 갖는 원소에서 첫 node에서 tail node까지 탐색하기 위해 사용한다.

## 모듈이름: DeleteHashTable()

### 기능

사용자로부터 ‘quit’ command를 입력받아 프로그램을 종료할 때, 기존에 생성된 hash table의 메모리를 free로 해제한다.

### 사용변수

Index – Linked list에서 첫 번째 node에 해당되는 hash table의 20개의 원소를 탐색할 때 필요한 index이다.

TempHash – Linked list 를 자료구조로 갖는 hash value를 index로 갖는 원소에서 첫 node에서 tail node까지 삭제하기 위해 사용한다.

## 모듈이름: CreateHashTable()

### 기능

프로그램을 실행하고 나서 opcode.txt 파일에서 읽은 데이터를 HashFunction 함수를 사용하여 hash table을 구성한다. fscanf 함수를 통해 파일에서 한 줄씩 데이터를 읽고, opcode와 opcode mnemonic, opcode format을 읽어서 이 중 mnemonic을 가지고 HashFunction 함수를 통해 hash value를 얻는다. 만약 hash value 값을 index로 갖는 hash table의 원소가 비어 있는 상태이면 데이터를 복사하여 저장하고, 그렇지 않으면 해당 원소의 linked list를 tail node까지 탐색하여 맨 마지막에 새로운 node를 linking한다.

### 사용변수

FilePointer – opcode.txt 파일에서 데이터를 읽기 위한 파일 포인터 변수이다.

Buffer – 데이터에서 읽은 mnemonic을 가지고 HashFunction 함수를 통해 얻은 hash value를 값으로 갖는다. Hash table에 저장할 적절한 index를 찾기 위해 사용한다.

BufferNumber[5] – 데이터로부터 읽은opcode를 임시로 저장하기 위해 사용한다.

BufferMnemonic[10] – 데이터로부터 읽은 mnemonic을 임시로 저장하기 위해 사용한다.

BufferFormat [10] – 데이터로부터 읽은 format을 임시로 저장하기 위해 사용한다.

# 전역 변수 정의

## typedef struct Node { char \*Value, struct Node\* Next} NODE

사용자가 입력한 이전의 valid command들을 데이터를 저장하기 위해 사용한다. NODE는 사용자의 valid한 입력과 다음 node를 linking하기 위한 자기 자신을 data type으로 갖는 포인터 변수를 멤버 변수로 갖는다.

## typedef struct Hash { char Number[5], char Mnemonic[10], char Format[10], struct Hash\* Next} HASH

opcode.txt에서 읽은 데이터를 가지고 hash table을 구성할 때 table 각각의 node의 데이터를 저장하기 위해 사용한다. HASH는 opcode에 해당하는 Number와 이에 대응되는 Mnemonic과 Format, 그리고 다음 node를 linking하기 위한 자기 자신을 data type으로 갖는 포인터 변수를 멤버 변수로 갖는다.

## HASH HashTable[20]

Opcode와 mnemonic을 저장할 크기가 20인 hash table이다.

## char DumpMemory[0x100000]

SIC에서 사용하기 위해 선언한 1MB 크기의 character type의 배열이다.

## int LastAddress

‘dump’ command를 수행할 때 출력한 마지막 1 byte 메모리의 address를 저장하기 위해 사용한다.

.

# 코드

## 20171665.h

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <dirent.h>

#include <unistd.h> // 리눅스에서 io.h 대신 사용

//#include <io.h> // 비주얼스튜디오에서 unistd.h 대신 사용

#include <errno.h>

#define Help 1

#define Dir 2

#define Quit 3

#define History 4

#define Dump 5

#define Edit 6

#define Fill 7

#define Reset 8

#define Opcode 9

#define OpcodeList 10

#define Character 11

#define Integer 12

typedef struct Node {

char \*Value;

struct Node\* Next;

}NODE;

typedef struct Hash {

char Number[5];

char Mnemonic[10];

char Format[10];

struct Hash\* Next;

}HASH;

NODE \*FirstNode = NULL;

HASH HashTable[20]; // Hash Table 선언

char DumpMemory[0x100000] = { 0 };

int LastAddress = -1;

## 20171665.c

#include "20171665.h"

//#include <crtdbg.h> // 메모리 누수 확인

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

/\*함수 : RemoveBlank(char \*\*Parameter)\*/

/\*목적 : 전달받은 Parameter에서 공백을 제거한다.

/\*리턴값 : 없음\*/

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

void RemoveBlank(char \*\*Parameter) {

int Index = 0, TempIndex = 0;

while ((\*Parameter)[TempIndex] == ' ' || (\*Parameter)[TempIndex] == '\t') {

TempIndex++;

}

while ((\*Parameter)[TempIndex] != '\0') {

(\*Parameter)[Index++] = (\*Parameter)[TempIndex++];

}

if (!Index & \*Parameter != NULL) {

free(\*Parameter);

\*Parameter = NULL;

}

else {

(\*Parameter)[Index] = '\0';

}

return;

}

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

/\*함수 : HexStringToDec(char \*Parameter, short DataType)\*/

/\*목적 : 전달받은 Parameter에서 16진수로 된 string을 10진수의 integer 값으로 바꾼다.

/\*리턴값 : 16진수를 10진수의 integer로 변환한 값을 반환한다. 입력한 값이 16진수가 아니거나 변환하고자 하는 data type의 범위를 초과할 경우 -1을 반환한다.\*/

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

int HexStringToDec(char \*Parameter, short DataType) {

long Buffer = -1;

char \*EndPointer = NULL;

errno = 0;

Buffer = strtol(Parameter, &EndPointer, 16);

if (EndPointer == Parameter || \*EndPointer != '\0') {

printf("No digits were found.\n");

return -1;

}

if ((Buffer == 2147483647L || Buffer == -2147483647L - 1) && errno == ERANGE) {

printf("The value you insert is invalid. Please write the valid value.\n");

return -1;

}

if (!(Buffer >= 0x00 && Buffer <= 0xFF) && DataType == Character) {

printf("The value you insert is out of boundary. (Boundary: 0x00 ~ 0xFF)\n");

return -1;

}

return (int)Buffer;

}

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

/\*함수 : ReadShellCommand(char \*\*InputShellCommand, char \*\*Parameter1, char \*\*Parameter2, char\*\* Parameter3)\*/

/\*목적 : 전달받은 Command와 Parameter에 따라 유효한 명령을 수행할 수 있도록 한다.

/\*리턴값 : 유효하지 않을 경우 0을 반환하고, 유효할 경우 입력받은 명령에 따라서 Header 파일에 정의된 명령값들을 반환한다.\*/

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

int ReadShellCommand(char \*\*InputShellCommand, char \*\*Parameter1, char \*\*Parameter2, char\*\* Parameter3) {

int BufferCounter = 0, ResultValue = 0;

short Flag = 0, Invalid = 0;

int CommandCounter = 0, ParameterCounter1 = 0, ParameterCounter2 = 0, ParameterCounter3 = 0;

char BufferChar = EOF, BeforeChar = EOF, \*BufferCommand = NULL;

printf("sicsim> ");

BufferCommand = (char\*)realloc(BufferCommand, sizeof(char) \* (CommandCounter + 2));

\*Parameter1 = (char\*)realloc(\*Parameter1, sizeof(char) \* (ParameterCounter1 + 2));

\*Parameter2 = (char\*)realloc(\*Parameter2, sizeof(char) \* (ParameterCounter2 + 2));

\*Parameter3 = (char\*)realloc(\*Parameter3, sizeof(char) \* (ParameterCounter3 + 2));

while ((BufferChar = getchar()) != '\n' && BufferChar != EOF) {

\*InputShellCommand = realloc(\*InputShellCommand, sizeof(char) \* (BufferCounter + 2));

(\*InputShellCommand)[BufferCounter++] = BufferChar;

if (BufferChar == ' ' || BufferChar == '\t') {

// 명령만 입력받은 상태에서 공백 또는 Tab이 입력된 경우

if (CommandCounter && Flag == 0) {

Flag = 1;

}

BeforeChar = BufferChar;

continue;

}

else if (BufferChar == ',') {

// Parameter를 한 개 입력받은 상태에서 ','가 입력된 경우

if (ParameterCounter1 && Flag == 1) {

Flag = 2;

}

// Parameter를 두 개 입력받은 상태에서 ','가 입력된 경우

else if (ParameterCounter2 && Flag == 2) {

Flag = 3;

}

// Parameter를 세 개 입력받은 후 또는 Command가 입력된 후 ','가 입력된 경우

else if ((ParameterCounter3 && Flag == 3) || (!ParameterCounter1 && Flag == 1) || (Flag == 0)) {

Invalid = 1;

}

BeforeChar = BufferChar;

continue;

}

else if (BeforeChar == ' ') {

// Parameter에서 띄어쓰기 처리가 된 경우

if ((ParameterCounter1 && Flag == 1) || (ParameterCounter2 && Flag == 2) || (ParameterCounter3 && Flag == 3)) {

Invalid = 1;

}

}

switch (Flag) {

case 0:

BufferCommand = (char\*)realloc(BufferCommand, sizeof(char) \* (CommandCounter + 2));

BufferCommand[CommandCounter++] = BufferChar;

break;

case 1:

\*Parameter1 = (char\*)realloc(\*Parameter1, sizeof(char) \* (ParameterCounter1 + 2));

(\*Parameter1)[ParameterCounter1++] = BufferChar;

break;

case 2:

\*Parameter2 = (char\*)realloc(\*Parameter2, sizeof(char) \* (ParameterCounter2 + 2));

(\*Parameter2)[ParameterCounter2++] = BufferChar;

break;

case 3:

\*Parameter3 = (char\*)realloc(\*Parameter3, sizeof(char) \* (ParameterCounter3 + 2));

(\*Parameter3)[ParameterCounter3++] = BufferChar;

break;

default:

break;

}

BeforeChar = BufferChar;

}

BufferCommand[CommandCounter] = (\*Parameter1)[ParameterCounter1] = (\*Parameter2)[ParameterCounter2] = (\*Parameter3)[ParameterCounter3] = (\*InputShellCommand)[BufferCounter] = '\0';

RemoveBlank(Parameter1);

RemoveBlank(Parameter2);

RemoveBlank(Parameter3);

if (Invalid != 1) {

// 입력받은 Command에서 Parameter가 하나도 없는 경우

if (\*Parameter1 == NULL) {

if (!strcmp(BufferCommand, "h") || !strcmp(BufferCommand, "help")) {

ResultValue = Help;

}

else if (!strcmp(BufferCommand, "d") || !strcmp(BufferCommand, "dir")) {

ResultValue = Dir;

}

else if (!strcmp(BufferCommand, "q") || !strcmp(BufferCommand, "quit")) {

ResultValue = Quit;

}

else if (!strcmp(BufferCommand, "hi") || !strcmp(BufferCommand, "history")) {

ResultValue = History;

}

else if (!strcmp(BufferCommand, "du") || !strcmp(BufferCommand, "dump")) {

ResultValue = Dump;

}

else if (!strcmp(BufferCommand, "reset")) {

ResultValue = Reset;

}

else if (!strcmp(BufferCommand, "opcodelist")) {

ResultValue = OpcodeList;

}

}

//Parameter가 최소 한 개 이상인 경우

else {

if ((!strcmp(BufferCommand, "du") || !strcmp(BufferCommand, "dump")) && Flag < 3) {

ResultValue = Dump;

}

else if ((!strcmp(BufferCommand, "e") || !strcmp(BufferCommand, "edit")) && Flag < 3) {

ResultValue = Edit;

}

else if (!strcmp(BufferCommand, "f") || !strcmp(BufferCommand, "fill")) {

ResultValue = Fill;

}

else if (!strcmp(BufferCommand, "opcode") && Flag < 2) {

ResultValue = Opcode;

}

}

}

if (BufferCommand != NULL) {

free(BufferCommand);

}

return ResultValue;

}

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

/\*함수 : HelpCommand()\*/

/\*목적 : 사용자가 어떠한 Command와 그에 맞는 형식으로 Input을 입력해야 하는지를 화면에 출력한다.

/\*리턴값 : 없음\*/

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

void HelpCommand() {

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");

}

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

/\*함수 : DirCommand()\*/

/\*목적 : 현재 실행 파일이 위치하고 있는 디렉토리 내의 모든 파일과 폴더를 출력한다.

/\*리턴값 : 없음\*/

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

void DirCommand() {

DIR \*DirectoryInfo = opendir(".");

struct dirent \*DirectoryEntry;

struct stat BufferEntry;

if (DirectoryInfo != NULL) {

int Counter = 1;

while ((DirectoryEntry = readdir(DirectoryInfo)) != NULL) {

if (Counter % 3 == 1) {

printf("\n");

}

else {

printf("\t");

}

if (stat(DirectoryEntry->d\_name, &BufferEntry) != -1) {

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

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

}

if (S\_ISREG(BufferEntry.st\_mode)) {

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

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

printf("\*");

}

}

}

Counter++;

}

printf("\n");

closedir(DirectoryInfo);

}

else {

printf("An error occurs while reading directory.\n");

}

}

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

/\*함수 : RecordPreviousCommand()\*/

/\*목적 : 사용자가 입력한 Valid Command들을 Linked List에 기록한다.

/\*리턴값 : 없음\*/

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

void RecordPreviousCommand(char\* InputShellCommand) {

NODE \*TempNode = NULL, \*PreviousNode = NULL;

TempNode = (NODE\*)malloc(sizeof(NODE));

TempNode->Value = (char\*)malloc(sizeof(char) \* strlen(InputShellCommand)+ 1);

TempNode->Next = NULL;

strcpy(TempNode->Value, InputShellCommand);

if (FirstNode == NULL) {

FirstNode = TempNode;

}

else {

PreviousNode = FirstNode;

while (PreviousNode->Next != NULL) {

PreviousNode = PreviousNode->Next;

}

PreviousNode->Next = TempNode;

}

}

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

/\*함수 : HistoryCommand()\*/

/\*목적 : RecordPreviousCommand에 의해 저장된 사용자의 이전 Command들을 모두 출력한다.

/\*리턴값 : 없음\*/

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

void HistoryCommand() {

NODE \*TempNode = FirstNode;

int NodeIndex = 0;

if (TempNode == NULL) {

printf("No Command Recorded!\n");

}

else {

while (TempNode != NULL) {

printf("%-5d %s\n", (++NodeIndex), TempNode->Value);

TempNode = TempNode->Next;

}

}

}

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

/\*함수 : DeleteHistory()\*/

/\*목적 : Quit Command를 통해 프로그램을 종료할 때, 저장된 Command들을 모두 free하여 메모리 해제한다.

/\*리턴값 : 없음\*/

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

void DeleteHistory() {

NODE \*TempNode = FirstNode, \*DeleteNode = NULL;

if (TempNode != NULL) {

while (TempNode != NULL) {

DeleteNode = TempNode;

TempNode = TempNode->Next;

free(DeleteNode->Value);

free(DeleteNode);

}

}

return;

}

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

/\*함수 : DumpCommand(char \*Start, char \*End)\*/

/\*목적 : 사전에 설정된 메모리(1MB)에서 사용자가 입력한 Parameter에 대응되는 주소 범위의 1 Byte 메모리 value를 출력한다.

/\*리턴값 : 정상적인 Parameter가 입력이 들어오고 정확히 수행되면 0을 반환하고, 그렇지 않으면 -1을 반환한다.\*/

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

int DumpCommand(char \*Start, char \*End) {

int i = 0, j;

int Counter, BufferStartParameter, BufferEndParameter;

char Buffer;

if (Start != NULL) {

if ((BufferStartParameter = HexStringToDec(Start, Integer)) == -1) {

return -1;

}

if (!(BufferStartParameter >= 0 && BufferStartParameter <= 0xFFFFF)) {

printf("[Start] Address is out of the boundary!\n");

return -1;

}

LastAddress = BufferStartParameter - 1;

}

LastAddress = (LastAddress + 1 <= 0xFFFFF) ? LastAddress : -1;

if (End != NULL) {

if ((BufferEndParameter = HexStringToDec(End, Integer)) == -1) {

return -1;

}

if (BufferEndParameter < BufferStartParameter) {

printf("[End] Address must be larger than [Start] Address.\n");

return -1;

}

if (!(BufferEndParameter >= 0 && BufferEndParameter <= 0xFFFFF)) {

printf("[End] Address is out of the boundary.\n");

return -1;

}

Counter = BufferEndParameter - LastAddress;

}

else {

Counter = 160;

}

while (i < Counter) {

if ((LastAddress + i + 1) > 0xFFFFF) {

break;

}

if ((i++) == 0) {

printf("%05X ", ((LastAddress + i) / 16) \* 16);

j = 0;

for (j = ((LastAddress + i) / 16) \* 16; j < (LastAddress + i); j++) {

printf(" ");

}

}

else if ((LastAddress + i) % 16 == 0) {

printf(" ; ");

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

if ((Buffer = DumpMemory[LastAddress + i + j - 16]) >= 32 && Buffer <= 126) {

printf("%c", Buffer);

}

else {

printf(".");

}

}

printf("\n%05X ", (LastAddress + i));

}

printf("%02X ", DumpMemory[LastAddress + i] & 0xFF);

}

if ((LastAddress + i + 1) % 16 == 0) {

printf(" ; ");

for (j = ((LastAddress + i) / 16) \* 16; j <= (LastAddress + i); j++) {

if ((Buffer = DumpMemory[j]) >= 32 && Buffer <= 126) {

printf("%c", Buffer);

}

else {

printf(".");

}

}

}

else {

for (j = (LastAddress + i + 1); j < ((LastAddress + i) / 16 + 1) \* 16; j++) {

printf(" ");

}

printf(" ; ");

for (j = ((LastAddress + i) / 16) \* 16; j <= (LastAddress + i); j++) {

if ((Buffer = DumpMemory[j]) >= 32 && Buffer <= 126) {

printf("%c", Buffer);

}

else {

printf(".");

}

}

for (j = (LastAddress + i + 1); j < ((LastAddress + i) / 16 + 1) \* 16; j++) {

printf(".");

}

}

printf("\n");

LastAddress = LastAddress + i;

return 0;

}

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

/\*함수 : EditCommand(char \*Address, char \*Value)\*/

/\*목적 : 메모리 주소 1 Byte와 수정하고자 하는 Value 값을 입력받아서 주소에 맞는 메모리 byte의 Value를 수정한다.

/\*리턴값 : 정상적인 Parameter가 입력이 들어오고 정확히 수행되면 0을 반환하고, 그렇지 않으면 -1을 반환한다.\*/

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

int EditCommand(char \*Address, char \*Value) {

int BufferAddress = 0, BufferValue = 0;

if (Address != NULL) {

if ((BufferAddress = HexStringToDec(Address, Integer)) == -1) {

return -1;

}

if (!(BufferAddress >= 0 && BufferAddress <= 0xFFFFF)) {

printf("The address is out of the boundary. (Boundary: 0x00000 ~ 0xFFFFF)\n");

return -1;

}

if (Value == NULL) {

printf("Please write the valid value.\n");

return -1;

}

if ((BufferValue = HexStringToDec(Value, Character)) == -1) {

return -1;

}

DumpMemory[BufferAddress] = (char)BufferValue;

}

else {

printf("Please write the valid address.\n");

return -1;

}

return 0;

}

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

/\*함수 : FillCommand(char \*Start, char \*End, char \*Value)\*/

/\*목적 : 메모리 주소 범위와 수정하고자 하는 Value 값을 입력받아서 주소 범위에 속하는 메모리 byte의 Value를 수정한다.

/\*리턴값 : 정상적인 Parameter가 입력이 들어오고 정확히 수행되면 0을 반환하고, 그렇지 않으면 -1을 반환한다.\*/

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

int FillCommand(char \*Start, char \*End, char \*Value) {

int BufferStartParameter = 0, BufferEndParameter = 0, Index = 0, BufferValue = 0;

if (Start != NULL) {

if ((BufferStartParameter = HexStringToDec(Start, Integer)) == -1) {

return -1;

}

if (!(BufferStartParameter >= 0 && BufferStartParameter <= 0xFFFFF)) {

printf("The [start] address is out of the boundary. (Boundary: 0x00000 ~ 0xFFFFF)\n");

return -1;

}

if (End != NULL) {

if ((BufferEndParameter = HexStringToDec(End, Integer)) == -1) {

return -1;

}

if (BufferEndParameter < BufferStartParameter) {

printf("The [end] address should not be smaller than [start] address.\n");

return -1;

}

if (!(BufferEndParameter >= 0 && BufferEndParameter <= 0xFFFFF)) {

printf("The [end] address is out of the boundary. (Boundary: 0x00000 ~ 0xFFFFF)\n");

return -1;

}

}

if (Value == NULL) {

printf("Please write the vaild value.\n");

return -1;

}

if ((BufferValue = HexStringToDec(Value, Character)) == -1) {

return -1;

}

for (Index = BufferStartParameter; Index <= BufferEndParameter; Index++) {

DumpMemory[Index] = (char)BufferValue;

}

}

return 0;

}

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

/\*함수 : ResetCommand()\*/

/\*목적 : Reset Command를 사용자로부터 입력받으면 메모리(1MB)의 모든 Value 값을 0으로 초기화한다.

/\*리턴값 : 정상적으로 수행되면 0을 반환한다.\*/

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

int ResetCommand() {

memset(DumpMemory, 0, sizeof(DumpMemory));

return 0;

}

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

/\*함수 : HashFunction()\*/

/\*목적 : Opcode Mnemonic을 일대일 함수를 통해 대응되는 uniqly identified한 함수값을 구한다.

/\*리턴값 : Opcode Mnemonic과 일대일 대응되는 독자적인 Hash 값을 반환한다.\*/

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

int HashFunction(char \*Mnemonic) {

int Index = 0, ReturnValue = 0;

while (Mnemonic[Index] != '\0') {

ReturnValue += (int)Mnemonic[Index++] % 19;

}

ReturnValue %= 20;

return ReturnValue;

}

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

/\*함수 : OpcodeCommand()\*/

/\*목적 : HashFunction 함수를 통해 구한 Hash 값을 사용하여 Hash Table을 탐색해서 Mnemonic에 대응되는 opcode를 구한다.

/\*리턴값 : Opcode Mnemonic과 일대일 대응되는 opcode를 반환한다.\*/

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

int OpcodeCommand(char \*Mnemonic) {

int HashCode;

HASH\* TempHash = NULL;

if (!((HashCode = HashFunction(Mnemonic)) >= 0 && HashCode < 20)) {

printf("Error occurs while reading hash table.\n");

return -1;

}

if ((TempHash = &(HashTable[HashCode])) == NULL) {

printf("Error occurs while reading hash table.\n");

}

while ((strcmp(TempHash->Mnemonic, Mnemonic) != 0)) {

if (TempHash->Next == NULL) {

printf("There isn't an opcode you want to find.\n");

return -1;

}

TempHash = TempHash->Next;

}

printf("opcode is %s\n", TempHash->Number);

return 0;

}

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

/\*함수 : OpcodeListCommand()\*/

/\*목적 : HashFunction 함수에 의해 구성된 Hash Table의 형태를 출력한다.

/\*리턴값 : Hash Table 탐색이 정상적으로 이루어지고 모든 opcode가 출력되면 0을 반환한다.\*/

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

int OpcodeListCommand() {

int Index;

HASH \*TempHash = NULL;

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

printf("%d : ", Index);

if (strcmp(HashTable[Index].Mnemonic, "")) {

TempHash = &(HashTable[Index]);

while (TempHash->Next != NULL) {

printf("[%s, %s] -> ", TempHash->Mnemonic, TempHash->Number);

TempHash = TempHash->Next;

}

printf("[%s, %s]\n", TempHash->Mnemonic, TempHash->Number);

}

else {

printf("\n");

}

}

return 0;

}

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

/\*함수 : DeleteHashTable()\*/

/\*목적 : Quit Command를 통해 프로그램을 종료할 때 기존에 저장된 Hash Table를 free로 메모리 해제한다.

/\*리턴값 : 없음\*/

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

void DeleteHashTable() {

int Index;

HASH \*TempHash = NULL, \*DeleteHash = NULL;

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

if (strcmp(HashTable[Index].Mnemonic, "")) {

TempHash = &(HashTable[Index]);

if ((TempHash = TempHash->Next) != NULL) {

while (TempHash != NULL) {

DeleteHash = TempHash;

TempHash = TempHash->Next;

free(DeleteHash);

}

}

}

}

return;

}

int CreateHashTable() {

FILE \*FilePointer = NULL;

if ((FilePointer = fopen("opcode.txt", "r")) == NULL) {

printf("An error occurs while opening 'opcode.txt' file.\n");

return -1;

}

int Buffer;

char BufferNumber[5] = { '\0' }, BufferMnemonic[10] = { '\0' }, BufferFormat[10] = { '\0' };

while (fscanf(FilePointer, "%s %s %s\n", BufferNumber, BufferMnemonic, BufferFormat) != EOF) {

if (!strcmp(HashTable[(Buffer = HashFunction(BufferMnemonic))].Mnemonic, "")) {

strcpy(HashTable[Buffer].Number, BufferNumber);

strcpy(HashTable[Buffer].Mnemonic, BufferMnemonic);

strcpy(HashTable[Buffer].Format, BufferFormat);

HashTable[Buffer].Next = NULL;

}

else {

HASH \*NewHash = NULL, \*TempHash = &(HashTable[Buffer]);

NewHash = (HASH\*)malloc(sizeof(HASH));

strcpy(NewHash->Number, BufferNumber);

strcpy(NewHash->Mnemonic, BufferMnemonic);

strcpy(NewHash->Format, BufferFormat);

NewHash->Next = NULL;

while (TempHash->Next != NULL) {

TempHash = TempHash->Next;

}

TempHash->Next = NewHash;

}

}

fclose(FilePointer);

return 0;

}

/\*프로그램 시작\*/

int main() {

char \*InputShellCommand = NULL;

char \*Parameter1 = NULL, \*Parameter2 = NULL, \*Parameter3 = NULL;

int ReadFlag = 0;

if (CreateHashTable() == -1) {

printf("Error occurs while creating hash table\n.");

return -1;

}

while ((ReadFlag = ReadShellCommand(&InputShellCommand, &Parameter1, &Parameter2, &Parameter3)) != Quit) {

if (ReadFlag != 0) {

switch (ReadFlag) {

case Help:

RecordPreviousCommand(InputShellCommand);

HelpCommand();

break;

case Dir:

RecordPreviousCommand(InputShellCommand);

DirCommand();

break;

case History:

RecordPreviousCommand(InputShellCommand);

HistoryCommand();

break;

case Dump:

if (!DumpCommand(Parameter1, Parameter2)) {

RecordPreviousCommand(InputShellCommand);

}

break;

case Edit:

if (!EditCommand(Parameter1, Parameter2)) {

RecordPreviousCommand(InputShellCommand);

}

break;

case Fill:

if (!FillCommand(Parameter1, Parameter2, Parameter3)) {

RecordPreviousCommand(InputShellCommand);

}

break;

case Reset:

if (!ResetCommand()) {

RecordPreviousCommand(InputShellCommand);

}

break;

case Opcode:

if (!OpcodeCommand(Parameter1)) {

RecordPreviousCommand(InputShellCommand);

}

break;

case OpcodeList:

if (!OpcodeListCommand()) {

RecordPreviousCommand(InputShellCommand);

}

break;

default:

printf("Please write the vaild command. You can refer to the valid command list by executing 'help' command.\n");

break;

}

}

else {

printf("Please write the vaild command. You can refer to the valid command list by executing 'help' command.\n");

}

}

DeleteHistory();

DeleteHashTable();

if (InputShellCommand != NULL) {

free(InputShellCommand);

}

if (Parameter1 != NULL) {

free(Parameter1);

}

if (Parameter2 != NULL) {

free(Parameter2);

}

if (Parameter3 != NULL) {

free(Parameter3);

}

//\_CrtSetDbgFlag(\_CRTDBG\_ALLOC\_MEM\_DF | \_CRTDBG\_LEAK\_CHECK\_DF); // 메모리 누수 확인

return 0;

}