과목명: 시스템프로그래밍

담당 교수명: 박운상

<<Assignment 1>>

**서강대학교 컴퓨터공학과**

**20161662 허남규**

# 프로그램 개요

이번 프로젝트의 최종 목표는 Linux OS 상에서 C 표준 라이브러리 함수를 기반으로 SIC/XE 가상 머신을 구현하는 것입니다. 이번 프로그램에서는 명령어 입출력, 메모리 조작 및 출력, 그리고 연산자 mnemonic을 opcode 값으로 변환하는 테이블 등 SIC/XE 가상 머신의 기반이 되는 요소들을 구현했습니다.

# 프로그램 설명

## 프로그램 흐름도 (Pseudo-code)



# 모듈 정의

이 프로그램은 OOP 설계 원칙에 따라 모든 기능을 데이터와 그에 대한 함수로 모듈화하여 만들었습니다. 구현 언어가 C 언어로 제한되어 있기 때문에 클래스를 사용하는 대신 각각의 모듈을 파일로 분리했습니다. 파일로 분리된 모듈 내에는 구조체와 new, free 함수, 그리고 해당 구조체 타입을 첫 인자로 갖는 기타 함수를 통해 OOP 방법론을 수용하였습니다.

아래 각각의 모듈에 대해 핵심 구조체와 함수에 관한 설명을 추가했습니다. 내부적인 구현과 변수, 지역 함수 등을 제외하고 헤더에 정의된 요소에 관해서만 서술하였습니다.

**Note.** 모듈 간의 독립성을 향상시키고 향후에 유닛 테스트를 쉽게 진행할 수 있도록, 모든 출력 기능은 output stream을 이용하거나 또는 문자열을 반환하는 형태로 구현했습니다.

## dir [dir.h]

현재 디렉토리를 출력하기 위한 함수를 담고 있는 모듈입니다.

### 함수 설명

void fprint\_dir(FILE \*out);

*out* output stream으로 현재 디렉토리를 출력합니다.

## hashtable [hashtable.h]

연산자 mnemonic – opcode에 해당하는 key – value 쌍을 저장하기 위한 해쉬 테이블을 제공하는 모듈입니다.

### 함수 설명

typedef struct \_HashTable { … } HashTable;

해쉬 테이블 데이터를 담기 위한 구조체입니다.

**Implementation Note.** 내부 데이터는 <<데이터 쌍을 담은 TableNode로 이루어진 linked-list>에 대한 reference를 담은 BucketNode의 배열>로 저장됩니다.

HashTable \*new\_hash\_table();

필요한 메모리를 할당받아 새로운 해쉬 테이블을 초기화하여 반환합니다.

void free\_hash\_table(HashTable \*table);

해쉬 테이블에 쓰인 메모리를 해제합니다.

void add\_to\_hash\_table(HashTable \*table, char \*key, Value value);

해쉬 테이블에 새로운 key – value 쌍을 추가합니다.

int find\_from\_hash\_table(HashTable \*table, char \*key, Value \*value);

해쉬 테이블 내에서 특정 key를 찾아 그와 연결된 값을 pass-by-reference로 반환하고 성공 여부를 반환합니다.

void fprint\_hash\_table(FILE \*out, HashTable \*table);

해쉬 테이블을 지정한 output stream에 출력합니다.

## help [help.h]

이 프로그램에서 사용할 수 있는 명령어에 관한 도움말 출력 기능을 제공하는 모듈입니다.

### 함수 설명

void fprint\_help(FILE \*out);

도움말을 지정된 output stream에 출력합니다.

## history [history.h]

문자열을 저장하여 지금까지 저장된 문자열을 출력하는 기록을 제공하는 모듈로, 이번 프로그램에서는 사용자 입력 기록을 출력하는 history 명령어 구현에 쓰입니다.

### 함수 설명

typedef struct \_History { … } History;

문자열 기록 데이터를 담는 구조체입니다.

**Implementation Note.** 내부 데이터는 크기가 정해진 문자열 (문자 배열)의 linked-list로 저장됩니다.

History \*new\_history();

필요한 메모리를 할당받아 새로운 문자열 기록을 초기화하여 반환합니다.

void free\_history(History \*history);

문자열 기록에 쓰인 메모리를 해제합니다.

void add\_history(History \*history, char \*string);

기록에 새로운 문자열을 추가합니다.

void fprint\_history(FILE \*out, History \*history);

기록된 문자열을 지정된 output stream에 모두 출력합니다.

int has\_history(History \*history);

기록된 문자열이 있는지 반환합니다.

## memory [memory.h]

정해진 크기 (1MB)에 해당하는 메모리를 담을 수 있는 메모리 블록 (Block)에 대한 읽기, 쓰기, 초기화, 출력 기능을 제공하는 모듈입니다.

### 함수 설명

typedef struct \_Block { … } Block;

1MB에 해당하는 메모리를 저장하는 구조체입니다.

**Implementation Note.** 내부 데이터는unsigned char 배열로 저장됩니다.

Block \*new\_memory\_block();

필요한 메모리를 할당받아 새로운 메모리 블록을 초기화하여 반환합니다.

// void free\_memory\_block();

메모리 블록은 별다른 내부 작업 없이 free 함수를 이용하여 메모리를 해제할 수 있습니다.

void set\_memory(Block \*block, int location, unsigned char value);

메모리 블록 내 지정된 주소에 새로운 값을 저장합니다.

void fill\_memory(Block \*block, int start, int end, unsigned char value);

메모리 블록 내 시작 주소부터 끝 주소까지 (inclusive) 새로운 값을 저장합니다.

char \*dump\_memory(Block \*block, int start, int end);

메모리 블록 내 시작 주소부터 끝 주소까지 dump 형식으로 출력된 문자열을 반환합니다.

끝 주소 인자로 -1 값이 넘겨질 경우 끝 주소를 지정하지 않은 것으로 간주하여 시작 주소부터 160개의 값을 출력합니다.

시작 주소와 끝 주소 인자로 모두 -1 값이 넘겨질 경우, 이전까지 출력한 주소 다음부터 160개의 값을 출력합니다.

이때 반환되는 문자열은 동적 할당된 것이므로 사용 후 꼭 메모리를 해제해야 합니다.

void reset\_memory(Block \*block);

메모리 블록 내의 값을 모두 0으로 초기화합니다.

# 전역 변수

이 프로그램은 전역 변수를 사용하지 않도록 설계되었습니다. 이는 모듈 간 독립성을 유지하여 충돌을 방지하고 모듈 단위의 테스트를 쉽게 할 수 있기 위함입니다.

# 특이 사항

## Inline 함수

main.c 및 다른 모듈 내에는 readability 향상을 위한 몇가지 지역 함수가 있습니다. 이들 중 일부는 컴파일러 최적화를 위해 static inline으로 선언했습니다. C99 표준에서 소개된 이 기능을 사용할 수 있도록 makefile에서 -std=c99 컴파일 옵션을 추가하였습니다.

## Macro 정의를 이용한 모듈 테스트

모든 모듈의 module\_name.c 파일에는 모듈 테스트를 위한 main 함수가 정의되어 있습니다. 이는TEST라는 매크로 정의를 선언해야만 컴파일 단위에 포함이 되도록 IFDEF, ENDIF 전처리문으로 둘러쌓여 있습니다. 해당 main 함수를 이용해서 모듈을 테스트하려면 gcc 컴파일 명령어에 -D TEST 옵션을 추가해야 합니다. 이 경우 main 기호가 다중으로 선언되는 문제로 인해 다른 모듈과 링크를 하여 테스트할 수 없습니다. 모듈 단위의 테스트가 목적이기 때문에 위와 같은 방법으로 전처리기를 사용했습니다.