과목 명: 시스템프로그래밍

담당 교수 명: 소 정 민

<<Assignment 1>>

**서강대학교 컴퓨터공학과**

**[20151597]**

**[임준묵]**

목 차

1. 프로그램 개요 5

1.1 프로그램 흐름도 5

2. 모듈 정의(20151597.c) 5

2.1 모듈 이름 : void print\_help() 5

2.1.1 기능 5

2.1.2 사용 변수 5

2.2 모듈 이름: int convert\_dec(char \*input) 5

2.2.1 기능 5

2.2.2 사용 변수 5

2.3 모듈이름: void reset(unsigned char \*\*memory) 5

2.3.1 기능 5

2.3.2 사용변수 5

2.4 모듈이름: void print\_dump1(unsigned char \*\*memory, int \*start) 6

2.4.1 기능 6

2.4.2 사용변수 6

2.5 모듈이름: int print\_dump2(unsigned char \*\*memory, char \*input, int \*start) 6

2.5.1 기능 6

2.5.2 사용변수 6

2.6 모듈 이름 : int print\_dump3(unsigned char \*\*memory, char \*input, int \*start) 7

2.6.1 기능 7

2.6.2 사용 변수 7

2.7 모듈 이름: int edit(unsigned char \*\*memory, char \*input) 7

2.7.1 기능 7

2.7.2 사용 변수 7

2.8 모듈이름: int fill(unsigned char \*\*memory, char \*input) 8

2.8.1 기능 8

2.8.2 사용변수 8

2.9 모듈이름: void print\_directory() 9

2.9.1기능 9

2.9.2사용변수 9

2.10모듈이름: history \*create\_history\_node(char \*input) 9

2.10.1기능 9

2.10.2사용변수 9

2.11모듈이름: void history\_made(history \*\*header, history \*newnode) 9

2.11.1기능 9

2.11.2사용변수 9

2.11모듈이름: void print\_history(history \*header) 9

2.11.1기능 9

2.11.2사용변수 9

2.12모듈이름: int hash\_function(char \*input) 9

2.12.1기능 9

2.12.2사용변수 10

2.13모듈이름: hash \*create\_hash\_node(FILE \*fp) 10

2.13.1기능 10

2.13.2사용변수 10

2.14 모듈이름: void hashtable\_made(hash \*header[20], FILE \*fp) 10

2.14.1기능 10

2.14.2사용변수 10

2.15 모듈이름: void opcodelist\_print(hash \*\*header) 10

2.15.1 기능 10

2.15.2 사용변수 10

2.16 모듈이름: int search\_opcode\_mnemonic(hash \*\*header, char \*input) 10

2.16.1 기능 10

2.16.2 사용변수 11

2.17 모듈이름: void revise\_input(char \*input1, char \*input2) 11

2.17.1 기능 11

2.17.2 사용변수 11

2.18 모듈이름: int mode(char \*input) 11

2.18.1 기능 11

2.18.2 사용변수 11

2.19 모듈이름: void free\_memory\_array(unsigned char \*\*memory) 12

2.19.1 기능 12

2.19.2 사용변수 12

2.20 모듈이름: void free\_history\_node(history \*\*head) 12

2.20.1 기능 12

2.20.2 사용변수 12

2.21 모듈이름: void free\_hash\_node(hash \*\*head) 12

2.21.1 기능 12

2.21.2 사용변수 12

2.22 모듈이름: int main(void) 12

2.22.1 기능 12

2.22.2 사용변수 12

3. 헤더파일(20151597.h) 13

3.1 History 구조체 13

3.1.1 선언 13

3.1.2 기능 13

3.1.3 구성원 13

3.2 Hash 구조체 13

3.2.1 선언 13

3.2.2 기능 13

3.2.3 구성원 13

4. 코드 14

5. 실행 결과 32

5.1 help command 입력 시 32

5.2 directory command 입력 시 33

5.3 quit command 입력 시 33

5.4 history command 입력 시 33

5.5 dump command 입력 시 33

5.5.1 dump 만 입력 시 33

5.5.2 dump ,start address 만 입력 시 34

5.5.3 dump, start address, end address 입력 시 34

5.6 edit 입력 시 34

5.7 fill 입력 시 34

5.8 reset 입력 시 35

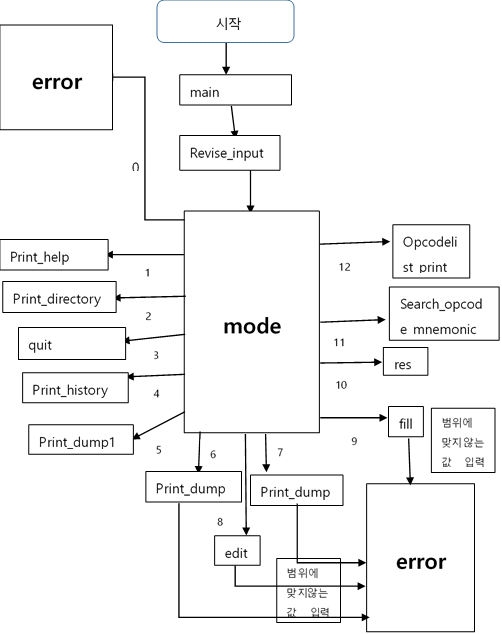
5.9 opcode mnemonic 입력 시 35

5.10 opcodelist 입력 시 35

# 프로그램 개요

이 프로그램은 sic/xe machine을 수행하기 위한 기초적인 수행 환경을 만들어 내는 프로그램이다. 우리는 명령어를 수행하기 위한 쉘 환경을 만들고 mnemonic와 opcode를 저장하기 위해서 해시 테이블을 만들어서 입력된 mnemonic의 opcode를 찾아준다.

## 프로그램 흐름도



# 모듈 정의(20151597.c)

## 모듈 이름 : void print\_help()

### 기능

main에서 help명령어를 입력받았을 때 사용 가능한 명령어들을 출력해주는 함수이다.

### 사용 변수

없음

## 모듈 이름: int convert\_dec(char \*input)

### 기능

Dump, fill, edit 명령을 수행할 때 명령어 뒤에 나오는 16진수들을 문자로 받아서 10진수로 변환해준다. 올바른 16진수를 받았으면(0~9, a~f) 그것의 10진수로 변환된 값을 반환해주고 그렇지 않은 경우에는 -1을 반환해준다.

### 사용 변수

Int I : 반복문을 위한 index

Int len : 입력받은 16진수들의 길이

Int result : 문자열로 받은 16진수들을 10진수로 변환했을 때의 결과

Char \*input : 10진수로 변환해줄 문자열(ex “AA”, “2A”) 을 가리키는 매개변수

## 모듈이름: void reset(unsigned char \*\*memory)

### 기능

memory영역을 0으로 reset 해주는 함수이다.

### 사용변수

Int I : 반복문을 위한 index

Int j : 반복문을 위한 index

Unsigned char \*\*memory : 메모리를 가리키는 변수를 받을 매개변수

## 모듈이름: void print\_dump1(unsigned char \*\*memory, int \*start)

### 기능

dump라는 명령어를 입력받았을 시에 메모리 영역과 메모리 주소와 메모리 영역의 원소가 나타내주는 문자를 출력해주는 함수이다. 처음에 dump를 수행했을 때 start address는 0 이고 이 address를 기준으로 160바이트 씩 출력해준다. Dump를 또 다시 수행했을 때 start address는 last address+1 이다. 그리고 last address가 0xFFFFF보다 크게 되면 dump 는 0xFFFFF 까지만 출력되고 last address는 0xFFFFF가 된다. 그 다음에 다시 dump가 수행되면 start address는 0이 된다.

### 사용변수

Unsigned char \*\*memory : 메모리를 가리키는 변수를 받을 매개변수

Int \*start : start address를 가리키는 변수를 받을 매개변수. Dump를 수행할 시에 시작 위치가 계속 바뀌어야 하기 때문에 포인터 변수로 선언해 줌으로 써 모듈 내에서 변경이 가능하도록 하였다.

Int first\_column : memory address를 저장하는 변수.

Int start\_index : 메모리가 2차원 배열이기 때문에 start address를 2차원 배열에서 접근하기 쉽게 하기 위해서 선언하였다. 메모리 에서의 행 index로 쓰인다.

Int end\_index : end address를 2차원 배열에서 접근하기 쉽게 하기 위해서 선언하였다. 메모리에서의 행 index로 쓰인다.

Int start\_index2 : start address를 2차원 배열에서 접근하기 쉽게 하기 위해서 선언하였다. 메모리 에서의 열 index로 쓰인다.

Int end\_index2 : end address를 2차원 배열에서 접근하기 쉽게 하기 위해서 선언하였다. 메모리에서의 열 index로 쓰인다.

Int start1, end1 : start address와 end address를 받기 위해서 선언하였다.

Int I, j : 반복문을 위한 index

## 모듈이름: int print\_dump2(unsigned char \*\*memory, char \*input, int \*start)

### 기능

Dump 명령어에서 start address가 추가된 함수이다. Dump와 마찬가지로 start address를 기준으로 160 바이트 씩 출력해준다. 만약 last address가 0xFFFFF를 넘어가게 되면 0xFFFFF 까지만 출력을 해주고 start address를 다시 0으로 설정을 해준다. 그리고 start address가 0x00000~0xFFFFF의 범위 내에 없으면 에러 메시지를 출력해준다. 에러 없이 잘 수행된다면 0을 반환해주고 그렇지 않은 경우에는 1을 반환해준다.

### 사용변수

Unsigned char \*\*memory : 메모리를 가리키는 변수를 받을 매개변수.

Char \*input : 사용자가 입력한 명령어를 가리키는 변수를 받을 매개변수(여기서의 명령어는 revise\_input 모듈을 통해서 수정된 명령어이다.)

Int \*start : start address를 가리키는 변수를 받을 매개변수. Start address가 계속 바뀌어야 하기 때문에 포인터 변수로 선언해 줌으로 써 모듈 내에서 변경이 가능하도록 하였다.

Int start1, end1 : start address와 end address를 저장해주는 변수.

Int I, j : 반복문을 위한 index

Int first\_column : memory address를 저장하는 변수

Int start\_index, end\_index : 메모리가 2차원 배열이기 때문에 start address 와 end address를 2차원 배열에서 쉽게 접근하기 위해서 선언하였다. 메모리에서의 행 index로 쓰인다.

Int start\_index2, end\_index2 : start address와 end address를 2차원 배열에서 쉽게 접근하기 위해서 선언하였다. 메모리에서의 열 index로 쓰인다.

Char \*start\_point : 캐릭터형 포인터 변수로 서 input에서 start address를 받기 위해 선언한 변수이다.

Int flag : 오류 없이 잘 수행되었는지를 가리키기 위해서 선언한 변수이다. Start address가 0x00000~0xFFFFF를 넘어가게 되면 -1을 저장하고 에러 없이 잘 수행되었다면 0을 저장한다.

## 모듈 이름 : int print\_dump3(unsigned char \*\*memory, char \*input, int \*start)

### 기능

Dump 명령어에서 start address와 last address 가 추가된 함수이다. 메모리 영역을 start address부터 end address 까지 출력을 해준다. Start address와 last address가 0x00000~0xFFFFF의 범위 내에 없거나 start address가 end address보다 더 크면 에러 메시지를 출력해준다. 에러 없이 잘 수행되었다면 0을 반환해주고 그렇지 않은 경우에는 -1을 반환해준다.

### 사용 변수

Unsigned char \*\*memory : 메모리를 가리키는 변수를 받을 매개변수.

Char \*input : 사용자가 입력한 명령어를 가리키는 변수를 받을 매개변수(여기서의 명령어는 revise\_input 모듈을 통해서 수정된 명령어이다.)

Int \*start : start address를 가리키는 변수를 받을 매개변수. Start address가 계속 바뀌어야 하기 때문에 포인터 변수로 선언해 줌으로 써 모듈 내에서 변경이 가능하도록 하였다.

Int start1, end1 : start address와 end address를 저장해주는 변수.

Int I, j : 반복문을 위한 index

Int first\_column : memory address를 저장하는 변수

Int start\_index, end\_index : 메모리가 2차원 배열이기 때문에 start address 와 end address를 2차원 배열에서 쉽게 접근하기 위해서 선언하였다. 메모리에서의 행 index로 쓰인다.

Int start\_index2, end\_index2 : start address와 end address를 2차원 배열에서 쉽게 접근하기 위해서 선언하였다. 메모리에서의 열 index로 쓰인다.

Char \*start\_point : 캐릭터형 포인터 변수로서 input에서 start address를 받기 위해 선언한 변수이다. 이 변수를 convert\_dec을 통해 10진수로 변환해 줌으로 써 start address를 구할 수 있다.

Char \*end\_point : 캐릭터형 포인터 변수로서 input에서 last address를 받기 위해 선언한 변수이다. 이 변수를 convert\_dec을 통해 10진수로 변환해 줌으로 써 end address를 구할 수 있다.

Char \*temp : 캐릭터형 포인터 변수로서 input에서 “dump”를 받기 위해 선언한 변수이다.

Int flag : 오류 없이 잘 수행되었는지를 가리키기 위해서 선언한 변수이다. 에러가 있다면 -1을 저장하고 에러가 없다면 0을 저장한다.

## 모듈 이름: int edit(unsigned char \*\*memory, char \*input)

### 기능

Target address와 value를 입력 받아 메모리의 target address에 있는 값을 입력 받은 value로 변경해주는 함수이다. Target address가 0x00000~0xFFFFF의 범위에 없거나 value가 0x00~0xFF의 범위 내에 없다면 에러 메시지를 출력해주고 -1을 반환해준다. 그렇지 않다면 0을 반환해준다.

### 사용 변수

Unsigned char \*\*memory : 메모리를 가리키는 변수를 받을 매개변수

Char \*input : 사용자가 입력한 명령어를 가리키는 변수를 받을 매개변수(여기서의 명령어는 revise\_input모듈을 통해서 수정된 명령어이다.)

Int I, j : 반복문을 위한 index

Char \*target\_point : 캐릭터형 포인터 변수로서 input에서 target address를 받기 위해 선언한 변수이다. 이 변수를 convert\_dec 모듈을 통해 10진수로 변환해 줌으로 써 target address를 구할 수 있다.

Char \*value\_ : 캐릭터형 포인터 변수로서 input에서 value를 받기 위해 선언한 변수이다. 이 변수를 convert\_dec 모듈을 통해 10진수로 변환해 줌으로 써 value를 구할 수 있다.

Char \*temp : 캐릭터형 포인터 변수로서 input에서 “edit”을 받기 위해 선언한 변수이다.

Int target, value : target address와 value를 저장하기 위해 선언한 변수 이다.

Int target\_index : 메모리가 2차원 배열이기 때문에 target address에 더 쉽게 접근하기 위해서 선언하였다. 메모리에서의 행 index를 가리킨다.

Int target\_index2 : 메모리가 2차원 배열이기 때문에 target address에 더 쉽게 접근하기 위해서 선언하였다. 메모리에서의 열 index를 가리킨다.

Int flag : 오류 없이 잘 수행되었는지를 가리키기 위해 선언한 변수이다. 에러 없이 잘 수행되었다면 0을 저장하고 에러가 있다면 -1을 저장한다.

## 모듈이름: int fill(unsigned char \*\*memory, char \*input)

### 기능

Start address와 end address, value를 입력받아 메모리 영역의 start address부터 end address 까지를 입력받은 value로 변경해 준다. Start address 와 end address가 0x00000~0xFFFFF의 범위 내에 없고 value가 0x00~0xFF의 범위 내에 없고 start address가 end address보다 큰 경우에는 에러 처리를 하고 에러메시지를 출력을 해준다. 에러가 있으면 -1을 반환해주고 그렇지 않다면 0을 반환해준다.

### 사용변수

Unsigned char \*\*memory : 메모리를 가리키는 변수를 받을 매개변수

Char \*input : 사용자가 입력한 명령어를 가리키는 변수를 받을 매개변수(여기서의 명령어는 revise\_input모듈을 통해서 수정된 명령어이다.)

Char \*start\_point, \*end\_point : 캐릭터형 포인터 변수로서 input에서 start address와 end address를 받기 위해 선언한 변수이다. 이 변수를 convert\_dec모듈을 통해 10진수로 변환해 줌으로써 start address 와 end address를 구할 수 있다.

Char \*value\_ : 캐릭터형 포인터 변수로서 input에서 value를 받기 위해 선언한 변수이다. 이 변수를 convert\_dec모듈을 통해 10진수로 변환해 줌으로 써 value를 구할 수 있다.

Char \*temp : 캐릭터형 포인터 변수로서 input에서 “fill”을 받기 위해 선언한 변수이다.

Int start\_index, end\_index : 메모리가 2차원 배열이기 때문에 start address와 end address에 더 쉽게 접근하기 위해서 선언하였다. 메모리에서의 행 index를 가리킨다.

Int start\_index2, end\_index2 : 메모리가 2차원배열이기 때문에 start address와 end address에 더 쉽게 접근하기 위해서 선언하였다. 메모리에서의 열 index를 가리킨다.

Int value, start, end : 각각 value, start address, end address를 저장하기위해 선언한 변수이다.

Int I, j : 반복문을 위한 index

Int flag : 오류 없이 잘 수행되었는지를 가리키기 위해 선언한 변수이다. 에러 없이 잘 수행되었다면 0을 저장하고 에러가 있다면 -1을 저장한다.

## 모듈이름: void print\_directory()

### 2.9.1기능

현재 디렉토리에 있는 파일, 디렉토리들을 출력해준다. 실행 파일의 경우에는 이름 뒤에 \*을 붙여주고 디렉토리인 경우에는 이름 뒤에 /을 붙여준다. 그리고 실행 파일과 디렉토리가 아닌 경우에는 .그냥 파일 이름을 출력해준다.

### 2.9.2사용변수

DIR \*dir : 디렉토리 스트림을 저장하는 변수이다.

Struct dirent \*dir\_entry : 디렉토리 내에 있는 파일, 또는 디렉토리의 정보를 저장하는 구조체 변수이다.

Struct stat astat : 파일의 정보를 저장하는 변수이다.

## 2.10모듈이름: history \*create\_history\_node(char \*input)

### 2.10.1기능

명령어를 저장하기 위한 history 연결리스트를 만들기 위해서 선언한 모듈로서 연결리스트의 노드를 만들어주는 모듈이다.

### 2.10.2사용변수

History \*tmp : 이 변수를 동적 할당 하여서 반환해준다.

Char \*input : history node가 저장할 명령어의 이름이다.

## 2.11모듈이름: void history\_made(history \*\*header, history \*newnode)

### 2.11.1기능

Valid한 명령어를 저장하는 history 연결리스트를 만들어 주는 모듈이다.

### 2.11.2사용변수

History \*tmp : 이 변수를 동적 할당 하여서 반환해준다.

History \*\*header : history 연결 리스트를 가리켜 주는 포인터 이다.

History \*newnode : 연결리스트에 추가해 줄 노드 이다.

## 2.11모듈이름: void print\_history(history \*header)

### 2.11.1기능

History 연결리스트를 출력해주는 함수이다.

### 2.11.2사용변수

History \*ptr : history 연결리스트의 헤더를 저장하는 변수이다.

## 2.12모듈이름: int hash\_function(char \*input)

### 2.12.1기능

Opcode.txt의 mnemonic들에 대한 hash key값을 정해주는 모듈이다. key 값은 mnemonic의 각 문자의 아스키 코드 값을 더한 뒤 20으로 나누었을 때의 나머지 값으로 설정하였다.

### 2.12.2사용변수

Int result : hash function의 결과를 저장하는 변수이다.

Int I : 반복문을 위한 index이다.

Int len : mnemonic의 길이를 저장하기 위한 변수이다.

# 2.13모듈이름: hash \*create\_hash\_node(FILE \*fp)

### 2.13.1기능

Hash table을 만들기 위한 hash 노드를 생성해 주는 모듈이다. Opcode.txt를 한줄씩 읽으면서 노드에 opcode, mnemonic, format을 저장해준다.

### 2.13.2사용변수

FILE \*fp : opcode.txt의 파일 스트림을 저장하는 변수이다.

Hash \*tmp : 이 변수를 동적할당 하여서 opcode, mnemonic, format을 저장한 뒤 반환해준다.

## 2.14 모듈이름: void hashtable\_made(hash \*header[20], FILE \*fp)

### 2.14.1기능

해시테이블을 만들어 주는 모듈이다. 연결리스트를 사용하여 생성하였다.

### 2.14.2사용변수

Hash \*header[20] : hash table을 위한 포인터 변수 이다.

FILE \*fp : opcode.txt의 파일 스트림을 저장하는 변수이다. 이 파일 스트림 변수를 바탕으로 hash node를 생성한다.

Hash \*pointer : hash table의 각 bucket을 가리키는 포인터 변수이다.

Hash \*newnode : create\_hash\_node를 통해 생성된 hash node를 저장하는 변수이다.

Int I : 반복문을 위한 index 이다.

Int index : hash\_function을 통해 생성된 key값을 저장하기 위한 변수이다.

## 모듈이름: void opcodelist\_print(hash \*\*header)

### 기능

생성된 hash table을 출력해주는 모듈이다.

### 사용변수

Hash \*\*header : hash table을 가리키는 포인터 변수이다.

Int i : 반복문을 위한 index

Hash \*ptr : hash table의 bucket을 가리키는 포인터 변수이다.

## 모듈이름: int search\_opcode\_mnemonic(hash \*\*header, char \*input)

### 기능

Opcode mnemonic명령어를 입력했을 때 hash table에 mnemonic이 있다면 mnemonic의 opcode를 출력해주고 0을 반환해준다. 그렇지 않은 경우에는 error 메시지를 출력해주고 -1을 반환해준다.

### 사용변수

Hash \*\*header : hash table을 가리키는 포인터를 저장하는 포인터 변수이다.

Char \*input : 사용자가 입력한 명령어를 저장하는 변수이다.

Char \*opcode : 사용자의 명령어(“opcode mnemonic”)에서 opcode만을 저장하기 위한 변수이다.

Char \*mnemonic : 사용자의 명령어(“opcode mnemonic”)에서 mnemonic만을 저장하기 위한 변수이다.

Int hash\_key : 사용자가 입력한 mnemonic의 hash key를 저장하는 변수이다.

Int flag : hash table에 사용자가 입력한 mnemonic이 있는지 없는지를 알려주는 변수이다. 처음에는 0으로 초기화 되고 mnemonic이 hash table내에 있다면 1을 저장한다.

Int result : search\_opcode\_mnemonic이 잘 수행되었는지를 판단하기 위해서 쓰이는 변수이다. 처음에 0으로 초기화 되고hash table 내에 입력한 mnemonic이 없다면 result는 -1의 값을 갖는다. 이 값이 반환된다.

Hash \*ptr : hash table 의 bucket을 가리키는 포인터를 저장하는 포인터 변수이다.

## 모듈이름: void revise\_input(char \*input1, char \*input2)

### 기능

사용자가 입력한 command를 내가 설정한 규칙으로 바꾸어 주는 모듈이다. 맨 앞, 맨 뒤의 공백을 제거하고 comma 앞 뒤의 공백도 제거하고 dump, edit 과 같이 명령어 뒤에 value 들이 들어오는 command에 대해서 명령어와 value들 사이에 공백을 한 개로 만들고 공백 대신에 “\_”를 집어넣었다. (\_\_\_\_\_dump\_\_\_\_\_\_aa\_\_\_\_\_,\_\_\_\_bb\_\_\_\_\_ -> dump\_aa,bb)

### 사용변수

Char \*input1 : 사용자가 입력한 command를 저장하는 변수이다.

Char \*input2 : 사용자가 입력한 command를 설정한 규칙으로 바꾸어준 command를 저장하는 변수이다.

Int I : 반복문을 위한 index 이다.

Int len1 : 사용자가 입력한 command의 길이를 저장하는 변수이다.

Int len2 : 규칙을 바탕으로 바꾼 command의 길이를 저장하는 변수이다.

Int space : 사용자가 입력한 command의 공백의 개수를 저장해주는 변수이다.

Int comma : 사용자가 입력한 command의 콤마의 개수를 저장해주는 변수이다.

## 모듈이름: int mode(char \*input)

### 기능

Revise\_input 모듈을 이용해서 바꾼 command를 바탕으로 사용자가 입력한 command에 대한 mode를 정해주는 함수이다.

### 사용변수

Char \*input : revise\_input모듈을 이용해서 바꾼 사용자가 입력한 command를 저장하는 변수이다.

Int result : mode를 저장해주는 변수 이다.

Int comma : input에서 comma의 개수를 저장해주는 변수이다.

Int underbar : input에서 underbar의 개수를 저장해주는 변수이다.

Int character : comma 이후의 문자의 개수를 저장해주는 변수이다.

Int I : 반복문을 위한 index 이다.

Int len : input의 길이를 저장해주는 변수이다.

Int comma1 : 두 번째 comma의 index를 저장하는 변수 이다.

Int character1 : 두 번째 comma 이후에 나오는 문자의 개수를 저장하는 변수이다.

## 모듈이름: void free\_memory\_array(unsigned char \*\*memory)

### 기능

memory영역의 메모리를 해제해주는 모듈이다.

### 사용변수

Int I : 반복문을 위한 index 이다.

Unsigned char \*\*memory : 메모리 영역을 가리키는 변수이다.

## 모듈이름: void free\_history\_node(history \*\*head)

### 기능

History 연결리스트를 해제해주는 모듈이다.

### 사용변수

History \*tmp : history \*\*head를 저장해주는 변수이다.

History \*\*head : history 연결리스트를 가리키는 포인터 이다.

## 모듈이름: void free\_hash\_node(hash \*\*head)

### 기능

Hash table 연결리스트를 해제해주는 모듈이다.

### 사용변수

Int I : 반복문을 위한 index

Hash \*tmp : hash \*\*head를 저장해주는 변수이다.

Hash \*\*head : 해시테이블 연결리스트를 가리키는 포인터 이다.

## 모듈이름: int main(void)

### 기능

프로그램의 main부분이다.

### 사용변수

Char \*input : 사용자가 입력하는 input을 저장하는 변수 이다.

Char \*std\_input, \*std\_input2 : revise\_input을 이용하여 바꾼 사용자의 input을 저장하는 변수이다.

Int len : 사용자가 입력한 input의 길이를 저장해주는 변수이다.

Int result : 사용자가 입력한 input에 대한 mode를 저장해 주는 변수이다.

Int I : 반복문을 위한 index 이다.

Int dump\_start : dump 수행시에 start address를 저장하기 위한 변수 이다. 0으로 초기화한다.

int len2 : std\_input의 길이를 저장해주는 변수이다.

History \*head : history 연결리스트를 가리키는 포인터를 위한 변수이다.

History \*tmp : history 연결리스트의 노드를 생성하기 위한 포인터 변수이다.

Unsigned char \*\*memory : 메모리 영역을 저장해주는 변수이다. 2차원 배열로 선언한다.

FILE fp1 : opcode.txt의 파일 스트림을 저장하기 위한 변수이다.

Hash \*hash\_head[20] : hash table의 bucket을 가리키는 변수이다.

Hash \*temp : hash table 연결리스트의 노드를 생성하기 위한 변수이다.

# 헤더파일(20151597.h)

## History 구조체

### 선언

Typedef struct \_history{

Char name[100];

Int index;

Struct \_history\* link;

} history;

### 기능

History 연결리스트를 만들기 위해서 선언한 구조체 이다.

### 구성원

Char name[100] : 사용자가 입력한 input을 저장하기 위해서 선언하였다.

Int index : 사용자가 입력한 input의 순서를 저장하기 위해서 선언하였다.

Struct \_history\* link : \_history 포인터 이다. 연결리스트를 만들기 위해서 선언하였다.

## Hash 구조체

### 선언

Typedef struct \_hash{

Char opcode[10];

Char instructions[10];

Char ss[10];

Struct \_hash \*link;

} hash;

### 기능

Hash table을 만들기 위해서 선언한 구조체이다. Hash table은 연결리스트로 만든다.

### 구성원

Char opcode[10] : mnemonic의 opcode를 저장하기 위해서 선언하였다.

Char instructions[10] : mnemonic을 저장하기 위해서 선언하였다.

Char ss[10] : mnemonic의 format을 저장하기 위해서 선언하였다.

Struct \_hash \*link : \_hash 포인터 이다. 연결리스트를 만들기 위해서 선언하였다.

# 코드

4-1. 20151597.c

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "20151597.h"

//for print help

void print\_help(){

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

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

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

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

printf("opcodelist\n");

}

//string convert to decimal

int convert\_dec(char \*input){

int i, len, result;

len = strlen(input);

result = 0;

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

if(input[i]>='0' && input[i]<='9'){

result = 16\*result+(input[i]-'0');

}

else if(input[i]>='A' && input[i]<= 'F'){

result = 16\*result+(input[i]-'A'+10);

}

else if(input[i]>='a' && input[i]<= 'f')

result = 16\*result+(input[i]-'a'+10);

else if(input[i]=='x')

result = 16 \* result + (input[i] - 'x');

else if(input[i]=='X')

result = 16 \* result + (input[i] - 'X');

else

result = -1;

}

return result;

}

//memory array 0

void reset(unsigned char \*\*memory){

int i, j;

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

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

memory[i][j] = 0;

}

}

}

//dump1(no start address, no end address)

void print\_dump1(unsigned char \*\*memory, int \*start){

int start1, end1, i, j;

int first\_column;

int start\_index, end\_index;

int start\_index2, end\_index2;

start1 = \*start;

first\_column = 0;

end1 = start1 +159;

start\_index = start1/16;

start\_index2 = start1%16;

first\_column = start\_index\*16;

if(end1>1048575){

end1 = 1048575;

end\_index = end1/16;

end\_index2 = end1%16;

}

else{

end\_index = end1/16;

end\_index2 = end1%16;

}

for(i = start\_index; i<=end\_index; i++){

printf("%05X\t", first\_column);

first\_column = first\_column+16;

if(i == start\_index){

for(j = 0; j<start\_index2; j++)

printf("\t");

for(j = start\_index2; j<16; j++){

printf("%02X\t", memory[i][j]);

}

printf(";");

for (j = 0; j < start\_index2; j++)

printf(".");

for(j = start\_index2; j<16; j++){

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

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

else

printf(".");

}

printf("\n");

}

else if(i==end\_index){

for(j = 0; j<=end\_index2; j++){

printf("%02X\t", memory[i][j]);

}

for(j = end\_index2+1; j<16; j++)

printf("\t");

printf(";");

for(j = 0; j<=end\_index2; j++){

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

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

else

printf(".");

}

for (j = end\_index2 + 1; j<16;j++)

printf(".");

printf("\n");

}

else{

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

printf("%02X\t", memory[i][j]);

printf(";");

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

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

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

else

printf(".");

}

printf("\n");

}

}

if(end1 == 1048575){

\*start = 0;

}

else{

\*start = end1+1;

}

}

//dump2(start address, no end address)

int print\_dump2(unsigned char \*\*memory, char \*input, int \*start){

int start1, end1, i, j;

int first\_column;

int start\_index, end\_index;

int start\_index2, end\_index2;

char k[100];

char \*start\_point;

char \*buffer;

int flag = 0;

start\_point = strtok(input, "\_");

start\_point = strtok(NULL, "");

start1 = convert\_dec(start\_point);

if(start1<0||start1>1048575){

printf("dump error : start input range over\n");

flag = -1;

}

else {

end1 = start1 + 159;

start\_index = start1 / 16;

start\_index2 = start1 % 16;

first\_column = start\_index \* 16;

if (end1 > 1048575) {

end1 = 1048575;

end\_index = end1 / 16;

end\_index2 = end1 % 16;

}

else {

end\_index = end1 / 16;

end\_index2 = end1 % 16;

}

for (i = start\_index; i <= end\_index; i++) {

printf("%05X\t", first\_column);

first\_column = first\_column + 16;

if (i == start\_index) {

for(j = 0; j<start\_index2; j++){

printf("\t");

}

for (j = start\_index2; j < 16; j++) {

printf("%02X\t", memory[i][j]);

}

printf(";");

for(j = 0; j<start\_index2; j++){

printf(".");

}

for (j = start\_index2; j < 16; j++) {

if (memory[i][j] >= 32 && memory[i][j] <= 126) {

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

}

else

printf(".");

}

printf("\n");

}

else if (i == end\_index) {

for (j = 0; j <= end\_index2; j++) {

printf("%02X\t", memory[i][j]);

}

for(j=end\_index2+1; j<16; j++)

printf("\t");

printf(";");

for(j = end\_index2+1;j<16; j++)

printf(" ");

for (j = 0; j <= end\_index2; j++) {

if (memory[i][j] >= 32 && memory[i][j] <= 126) {

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

}

else

printf(".");

}

printf("\n");

}

else {

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

printf("%02X\t", memory[i][j]);

}

printf(";");

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

if (memory[i][j] >= 32 && memory[i][j] <= 126) {

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

}

else

printf(".");

}

printf("\n");

}

}

if (end1 == 1048575)

\*start = 0;

else

\*start = end1 + 1;

}

return flag;

}

//dump3(start address, end address)

int print\_dump3(unsigned char \*\*memory, char \*input, int \*start) {

int start1, end1, i, j;

int first\_column;

int start\_index, start\_index2;

int end\_index, end\_index2;

int flag;

char \*start\_point, \*end\_point, \*temp;

flag = 0;

temp = strtok(input, "\_");

start\_point = strtok(NULL, ",");

end\_point = strtok(NULL, "");

start1 = convert\_dec(start\_point);

end1 = convert\_dec(end\_point);

if (start1 < 0 || start1>1048575) {

printf("dump error : start input range over\n");

flag = -1;

}

if (end1 < 0 || end1>1048575) {

printf("dump error : end input range over\n");

flag = -1;

}

if (start1 > end1) {

printf("dump error : start input larger than end input\n");

flag = -1;

}

if (flag == 0) {

start\_index = start1 / 16;

start\_index2 = start1 % 16;

first\_column = start\_index \* 16;

end\_index = end1 / 16;

end\_index2 = end1 % 16;

for (i = start\_index; i <= end\_index; i++) {

printf("%05X\t", first\_column);

first\_column = first\_column + 16;

if (i == start\_index && i == end\_index) {

for (j = 0; j < start\_index2; j++)

printf("\t");

for (j = start\_index2; j <= end\_index2; j++)

printf("%02X\t", memory[i][j]);

for (j = end\_index2 + 1; j < 16; j++)

printf("\t");

printf(";");

for (j = 0; j < start\_index2; j++)

printf(".");

for (j = start\_index2; j <= end\_index2; j++) {

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

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

else

printf(".");

}

for (j = start\_index2 + 1; j < 16; j++)

printf(".");

printf("\n");

}

else if (i == start\_index) {

for (j = 0; j < start\_index2; j++)

printf("\t");

for (j = start\_index2; j < 16; j++)

printf("%02X\t", memory[i][j]);

printf(";");

for (j = 0; j < start\_index2; j++)

printf(".");

for (j = start\_index2; j < 16; j++) {

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

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

else

printf(".");

}

printf("\n");

}

else if (i == end\_index) {

for (j = 0; j <= end\_index2; j++)

printf("%02X\t", memory[i][j]);

for (j = end\_index2 + 1; j < 16; j++)

printf("\t");

printf(";");

for (j = 0; j <= end\_index2; j++) {

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

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

else

printf(".");

}

for (j = end\_index2 + 1; j < 16; j++)

printf(".");

printf("\n");

}

else {

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

printf("%02X\t", memory[i][j]);

printf(";");

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

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

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

else

printf(".");

}

printf("\n");

}

}

if (end1 < 1048575)

\*start = end1 + 1;

else if (end1 == 1048575)

\*start = 0;

}

return flag;

}

//target address's value -> value of user input

int edit(unsigned char \*\*memory, char \*input) {

char \*value\_, \*target\_point, \*temp;

int target, value;

int target\_index, target\_index2;

int flag = 0;

temp = strtok(input, "\_");

target\_point = strtok(NULL, ",");

value\_ = strtok(NULL, "");

target = convert\_dec(target\_point);

value = convert\_dec(value\_);

if (target < 0 || target>1048575) {

printf("edit error : target point range over!\n");

flag = -1;

}

if (value > 255 || value < 0) {

printf("edit error : value range over!\n");

flag = -1;

}

if (flag == 0) {

target\_index = target / 16;

target\_index2 = target % 16;

memory[target\_index][target\_index2] = value;

}

return flag;

}

//from start address to end address, memory's value->value of user input

int fill(unsigned char \*\*memory, char \*input) {

char \*start\_point, \*end\_point, \*value\_, \*temp;

int start\_index, end\_index;

int start\_index2, end\_index2;

int value, start, end;

int i, j;

int flag = 0;

temp = strtok(input, "\_");

start\_point = strtok(NULL, ",");

end\_point = strtok(NULL, ",");

value\_ = strtok(NULL, "");

start = convert\_dec(start\_point);

end = convert\_dec(end\_point);

value = convert\_dec(value\_);

if (start < 0 || start>1048575) {

printf("fill error : start input range over!\n");

flag = -1;

}

if (end < 0 || end>104875) {

printf("fill error : end input range over!\n");

flag = -1;

}

if (value < 0 || value>255) {

printf("fill error : value input range over!\n");

flag = -1;

}

if(start>end){

printf("fill error : start input more than end input!\n");

flag = -1;

}

if (flag == 0) {

start\_index = start / 16;

start\_index2 = start % 16;

end\_index = end / 16;

end\_index2 = end % 16;

for (i = start\_index; i <= end\_index; i++) {

if (i == start\_index && i == end\_index) {

for (j = start\_index2; j <= end\_index2; j++) {

memory[i][j] = value;

}

}

else if (i == start\_index) {

for (j = start\_index2; j < 16; j++)

memory[i][j] = value;

}

else if (i == end\_index) {

for (j = 0; j <= end\_index2; j++)

memory[i][j] = value;

}

else {

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

memory[i][j] = value;

}

}

}

return flag;

}

//print file, directory in current directory

void print\_directory(){

DIR \*dir;

struct dirent \*dir\_entry;

struct stat astat;

dir = opendir(".");

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

lstat(dir\_entry->d\_name, &astat);

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

if(strncmp(dir\_entry->d\_name, ".", 1)!=0 && strncmp(dir\_entry->d\_name, "..",2)!=0){

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

printf("/\n");

}

}

else{

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

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

}

else

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

}

}

}

//create history node for history linked list

history \*create\_history\_node(char \*input){

history \*tmp;

tmp = (history\*)malloc(sizeof(history));

strcpy(tmp->name, input);

tmp->link = NULL;

return tmp;

}

//make history linked list

void history\_made(history \*\*header, history \*newnode){

history \*tmp = \*header;

if(header == NULL){

\*header = newnode;

}

else{

while(tmp->link != NULL){

tmp = tmp->link;

}

tmp->link = newnode;

tmp->link->index = tmp->index+1;

}

}

//print history linked list

void print\_history(history \*header){

history \*ptr = header;

while(ptr->link !=NULL){

printf("%d\t%s\n", ptr->link->index, ptr->link->name);

ptr = ptr->link;

}

}

//generate hash key for mnemonic

int hash\_function(char \*input) {

int result,i, len;

result = 0;

len = strlen(input);

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

result = (int)input[i]+result;

}

result = result%20;

return result;

}

//create hash node for hash table

hash \*create\_hash\_node(FILE \*fp) {

hash \*tmp;

tmp = (hash\*)malloc(sizeof(hash));

fscanf(fp, "%s%s%s", tmp->opcode, tmp->instructions, tmp->ss);

tmp->link = NULL;

return tmp;

}

//make hash table

void hashtable\_made(hash \*header[20], FILE \*fp) {

hash \*pointer;

hash \*newnode;

int i, index;

for (i = 0; i < 58; i++) {

newnode = create\_hash\_node(fp);

index = hash\_function(newnode->instructions);

pointer = header[index];

if (header[index] == NULL)

header[index] = newnode;

else {

while (pointer->link != NULL)

pointer = pointer->link;

pointer->link = newnode;

}

}

}

//print hash table

void opcodelist\_print(hash \*\*header) {

int i;

hash \*ptr;

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

ptr = header[i];

printf("%d : ", i);

if(ptr!=NULL){

printf("[%s, %s]", ptr->instructions, ptr->opcode);

ptr = ptr->link;

}

while (ptr != NULL) {

printf("->[%s, %s]", ptr->instructions, ptr->opcode);

ptr = ptr->link;

}

printf("\n");

}

}

//search mnemonic's opcode in hash table

int search\_opcode\_mnemonic(hash \*\*header, char \*input){

char \*opcode, \*mnemonic;

int hash\_key, flag;

int result = 0;

hash \*ptr;

flag = 0;

opcode = strtok(input, "\_");

mnemonic = strtok(NULL, "");

hash\_key = hash\_function(mnemonic);

ptr = header[hash\_key];

while(ptr!=NULL){

if(strcmp(ptr->instructions, mnemonic)==0){

flag++;

break;

}

ptr = ptr->link;

}

if(flag==1)

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

else {

printf("this mnemonic doesn't exist in hash table\n");

result = -1;

}

return result;

}

//user input->standard input

//standard input : command\_startaddress,endaddress,value

void revise\_input(char \*input1, char \*input2){

int i, len1, len2, j;

int space, comma;

space = comma = 0;

len2 = 0;

len1 = strlen(input1);

input1[len1-1] = '\0';

for(i = 0; i<len1; i++){

if(input1[i] != ' ' && input1[i] != '\t'){

input2[len2] = input1[i];

len2++;

}

else if(input1[i] == ' '|| input1[i] == '\t'){

if(input1[i-1] != ' ' && input1[i+1] != ',' && input1[i-1] != ',' && i-1>=0 && i+1<len1 && input1[i-1] != '\t')

input2[len2++] = '\_';

}

}

if(input2[len2-2] == '\_')

input2[len2-2] = '\0';

for(i = 0; i<len2; i++){

if(input2[i] == '\_' && i+1 < len2 && input2[i+1] == ','){

for(j = i; j<len2; j++){

input2[j] = input2[j+1];

}

}

}

}

//determine mode for user input

int mode(char \*input){

int result, comma, underbar, character;

int i, len;

int comma1, character1;

comma = result = 0;

comma1 = 0;

underbar = character = character1 = 0;

len = strlen(input);

if(input[0] == '\0')

result = 0;

else{

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

if(input[i] == ','){

comma++;

comma1 = i;

if(input[i+1] != '\0')

character++;

}

else if(input[i] == '\_')

underbar++;

}

if(input[comma1+1]!='\0')

character1++;

char \*token, \*token2;

token = strtok(input, "\_");

if(strcmp(token, "help")==0 || strcmp(token, "h")==0)

result = 1;

else if(strcmp(token, "dir")==0 || strcmp(token, "d")==0)

result = 2;

else if(strcmp(token, "quit")==0 || strcmp(token, "q")==0)

result = 3;

else if(strcmp(token, "history")==0 || strcmp(token, "hi")==0)

result = 4;

else if(strcmp(token, "dump")==0 || strcmp(token, "du")==0){

if(comma==0 && underbar ==0)

result = 5;

else if(comma==0 && underbar > 0)

result = 6;

else if(comma == 1 && underbar ==1&&character > 0)

result = 7;

}

else if(strcmp(token, "edit")==0 || strcmp(token, "e")==0){

if(comma==1 && underbar==1&&character>0)

result = 8;

}

else if(strcmp(token, "fill")==0 || strcmp(token, "f")==0){

if(comma==2&&character1>0)

result = 9;

}

else if(strcmp(token, "reset")==0)

result = 10;

else if(strcmp(token, "opcode")==0)

result = 11;

else if(strcmp(token, "opcodelist")==0)

result = 12;

}

return result;

}

//free memory array

void free\_memory\_array(unsigned char \*\*memory){

int i;

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

free(memory[i]);

free(memory);

}

//free history linked list

void free\_history\_node(history \*\*head){

history \*tmp;

while(\*head){

tmp = \*head;

\*head = (\*head)->link;

free(tmp);

}

}

//free hash table

void free\_hash\_node(hash \*\*head){

int i;

hash \*tmp;

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

while(\*head){

tmp = head[i];

head[i] = head[i]->link;

free(tmp);

}

}

}

//main for program

int main(void){

char \*input;

char \*std\_input, \*std\_input2;

int len, result, i, dump\_start;

//std\_input2's length

int len2;

history \*head;

history \*tmp;

unsigned char \*\*memory;

FILE \*fp1;

hash \*hash\_head[20];

hash \*temp;

fp1 = fopen("opcode.txt", "r");

head = (history\*)malloc(sizeof(history));

head->link = NULL;

head->index = 0;

dump\_start = 0;

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

hash\_head[i] = NULL;

}

hashtable\_made(hash\_head, fp1);

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

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

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

while(1){

printf("sicsim>");

input = (char\*)malloc(sizeof(char)\*100);

fgets(input, sizeof(char)\*100, stdin);

len = strlen(input);

std\_input = (char\*)malloc(sizeof(char)\*101);

revise\_input(input, std\_input);

len2 = strlen(std\_input);

std\_input2 = (char\*)malloc(sizeof(char)\*len2);

strcpy(std\_input2, std\_input);

result = mode(std\_input);

//print help

if (result == 1) {

tmp = create\_history\_node(input);

history\_made(&head, tmp);

print\_help();

}

//print directory

else if (result == 2) {

tmp = create\_history\_node(input);

history\_made(&head, tmp);

print\_directory();

}

//quit

else if (result == 3) {

free(input);

free(std\_input);

free(std\_input2);

free\_hash\_node(hash\_head);

free\_history\_node(&head);

free\_memory\_array(memory);

break;

}

//print history

else if (result == 4) {

tmp = create\_history\_node(input);

history\_made(&head, tmp);

print\_history(head);

}

//print dump 1

else if (result == 5) {

tmp = create\_history\_node(input);

history\_made(&head, tmp);

print\_dump1(memory, &dump\_start);

}

//print dump 2

else if (result == 6 && !print\_dump2(memory, std\_input2, &dump\_start)) {

tmp = create\_history\_node(input);

history\_made(&head, tmp);

}

//print dump3

else if (result == 7 && !print\_dump3(memory, std\_input2, &dump\_start)) {

tmp = create\_history\_node(input);

history\_made(&head, tmp);

}

//edit

else if (result == 8 && !edit(memory, std\_input2)) {

tmp = create\_history\_node(input);

history\_made(&head, tmp);

}

//fill

else if (result == 9 && !fill(memory, std\_input2)) {

tmp = create\_history\_node(input);

history\_made(&head, tmp);

}

//reset

else if (result == 10) {

tmp = create\_history\_node(input);

history\_made(&head, tmp);

reset(memory);

}

//opcode mnemonic

else if (result == 11&&!search\_opcode\_mnemonic(hash\_head, std\_input2)) {

tmp = create\_history\_node(input);

history\_made(&head, tmp);

}

//opcodelist

else if (result == 12) {

tmp = create\_history\_node(input);

history\_made(&head, tmp);

opcodelist\_print(hash\_head);

}

//wrong command

else if(result ==0)

printf("wrong command!\n");

}

}

4-2. 20151597.h

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <sys/stat.h>

typedef struct \_history{

char name[100];

int index;

struct \_history\* link;

} history;

typedef struct \_hash{

char opcode[10];

char instructions[10];

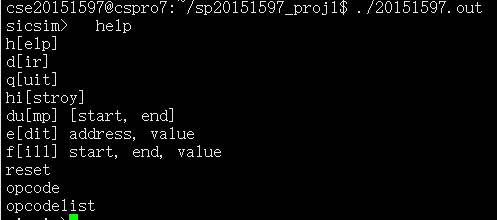
char ss[10];

struct \_hash \*link;

}hash;

# 실행 결과

## help command 입력 시



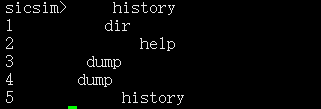
## directory command 입력 시



## quit command 입력 시

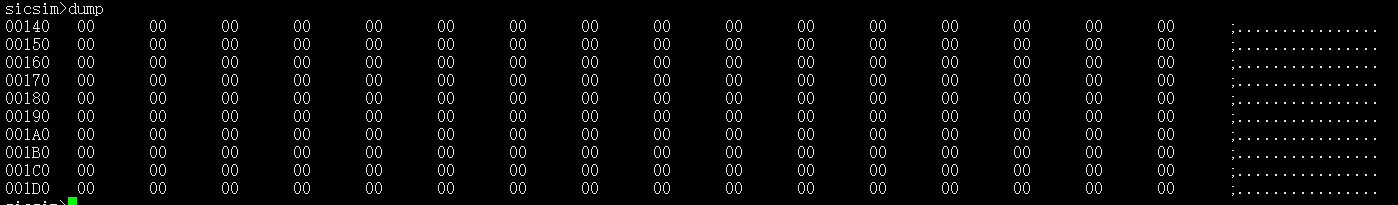


## history command 입력 시

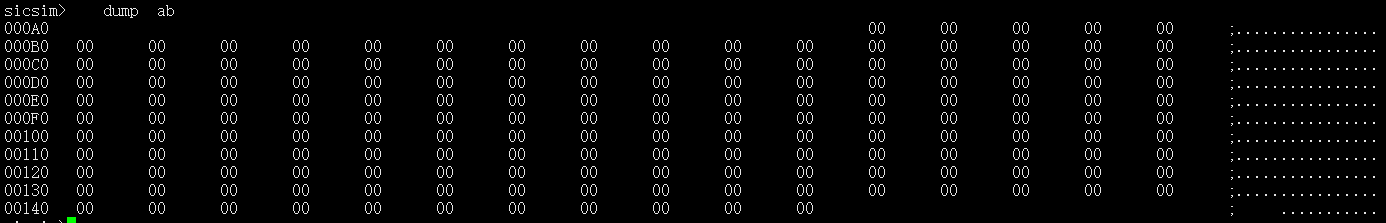


## dump command 입력 시

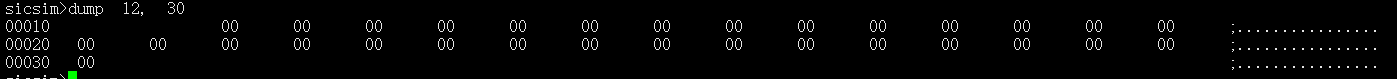
### dump 만 입력 시



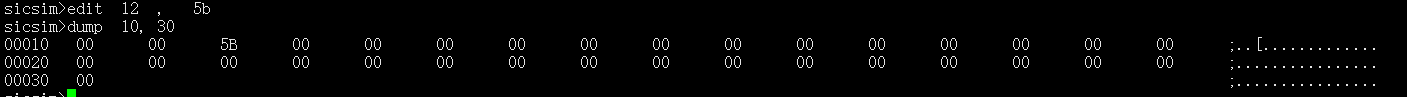
### dump ,start address 만 입력 시



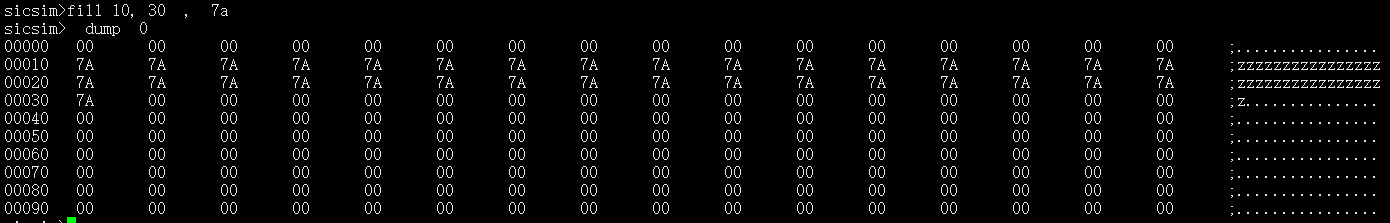
### dump, start address, end address 입력 시



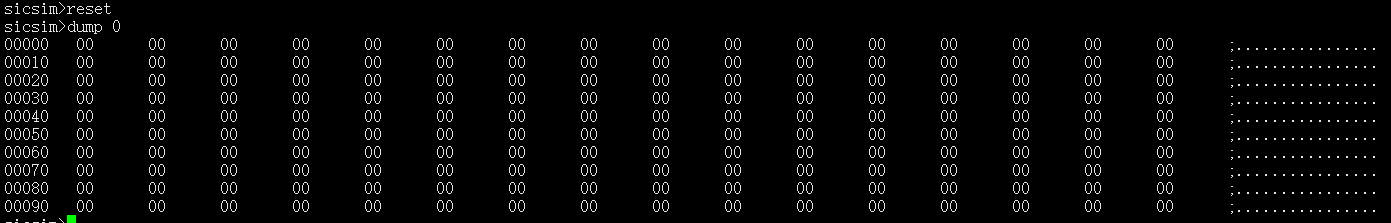
## edit 입력 시



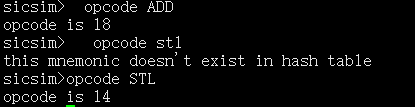
## fill 입력 시



## reset 입력 시



## opcode mnemonic 입력 시



## opcodelist 입력 시

