**과목명: 시스템프로그래밍**

**1반**

**<<Project #1>>**

**서강대학교 컴퓨터공학과**

**20161563**

**고성빈**

목 차

1 프로그램 개요

2 프로그램 설명

2.1 프로그램 흐름도

3 모듈 및 전역 변수 정의[[1]](#footnote-1)

3.1 메인 함수

3.1.1 int main()

3.2 명령어 입력 구현

3.2.1 전역 변수

3.2.2 int scan\_order(char\* order)

3.2.3 int order\_link(char\* order)

3.2.4 int order\_to\_num(char\* order)

3.2.5 void dump\_rest()

3.3 help 명령어 구현

3.3.1 int function\_help()

3.4 dir 명령어 구현

3.4.1 int function\_dir()

3.5 quit 명령어 구현

3.5.1 function\_quit()

3.6 history 명령어 구현

3.6.1 구조체 및 전역 변수

3.6.2 function\_history()

3.6.3 void set\_history()

3.6.4 void insert\_history(int order\_num, char\* rest)

3.6.5 void print\_order()

3.6.6 void free\_history()

3.7 dump 명령어 구현

3.7.1 전역 변수

3.7.2 int function\_dump()

3.7.3 void print\_memory(int st, int en)

3.7.4 void print\_memory\_by\_line(int line, int st, int en)

3.8 edit 명령어 구현

3.8.1 int function\_edit()

3.9 fill 명령어 구현

3.9.1 int function\_fill()

3.10 reset 명령어 구현

3.10.1 function\_reset()

3.11 opcode 명령어 구현

3.11.1 구조체 및 전역 변수

3.11.2 function\_opcode()

3.11.3 set\_hash\_table()

3.11.4 word\_to\_hash(char\* name)

3.11.5 free\_hash\_table()

3.12 opcodelist 명령어 구현

3.12.1 function\_opcodelist()

4 코드 설명

4.1 20161563.h

4.2 20161563.c

1 프로그램 개요

이 프로그램은 이후 프로젝트를 통해 구현할 SIC/XE 머신을 구현하기 위한 기반 단계로, 셸(Shell)을 만들어 어셈블러, 링크, 로더를 실행하고, 메모리 공간을 만들어 컴파일된 object코드를 저장할 수 있어야 한다. 또한 OPCODE 테이블로 mnemonic(ADD, COMP 등)을 opcode 값으로 변화시킬 수 있어야 하며, 이와 관련된 명령어를 입력받아 실행할 수 있어야 한다.

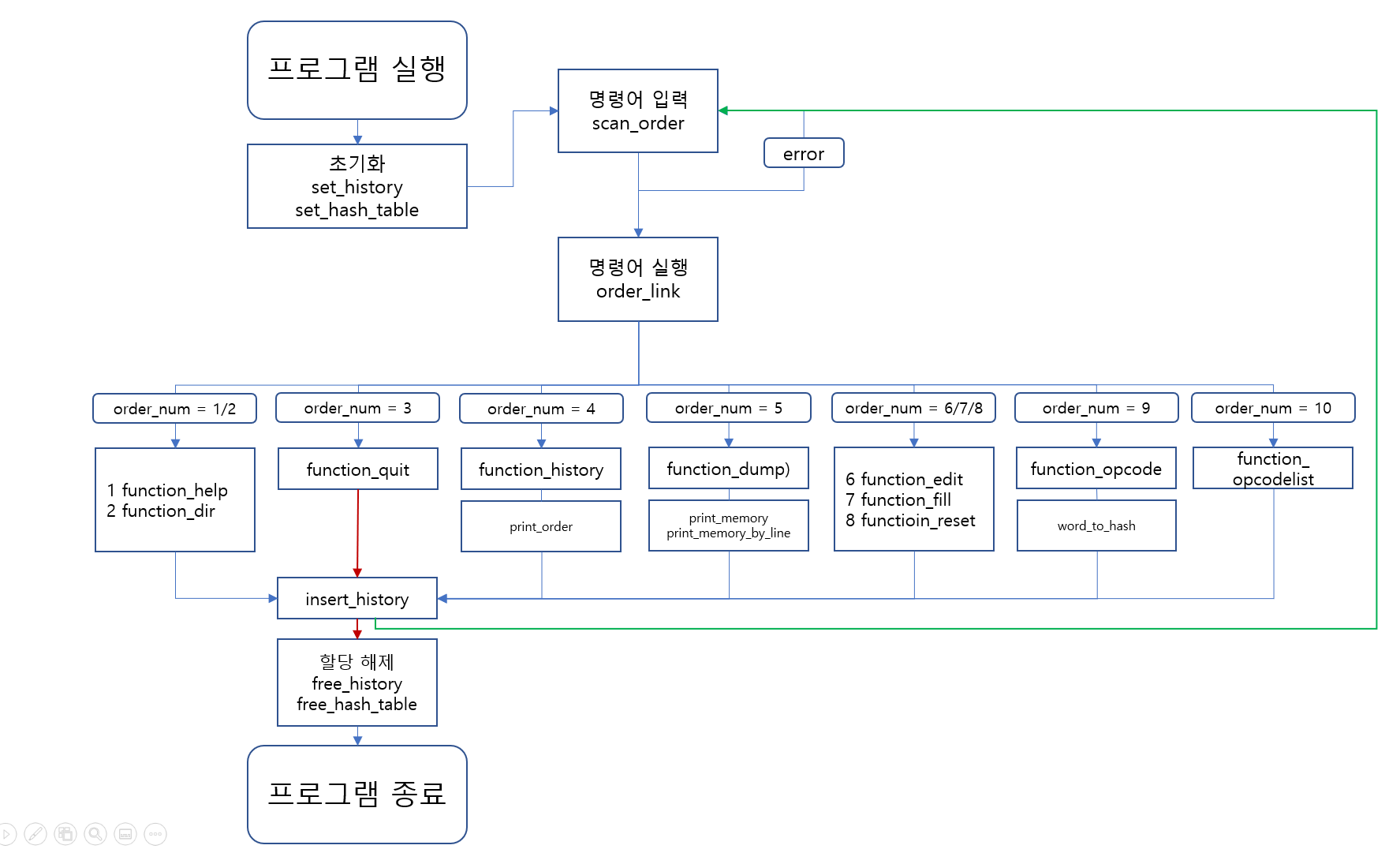
2 프로그램 설명

프로그램은 실행되고 나서 초기화 단계를 거친 후, 종료되기 전까지 다음과 같은 단계를 반복한다.

scan\_order로 명령어를 입력받고, order\_link를 통해 해당 명령어의 함수를 실행한다. 해당 함수가 종료된 후 insert\_history 함수를 통해 기록하고 다시 명령어 입력 단계로 돌아온다. quit 명령어가 입력된 경우에만 반복문을 종료하고, 할당된 메모리를 해제한 뒤 프로그램을 종료한다.

2.1 프로그램 흐름도

프로그램 흐름도는 다음과 같다.



3 모듈 정의

3.1 메인 함수

메인 함수는 프로그램의 시작과 동시에 작동하는 함수로써, 명령어를 입력받기 전 단계에서의 history와opcode 저장을 위한 linked list를 initialize 해준다. 이후 while 반복문을 통해 명령어를 입력받고, 해당 명령어를 읽고 실행하는 함수를 호출한다.

q(uit) 명령을 통해 order\_link 함수의 반환값이 3이 올 경우 반복문을 종료하며, 처음 할당한 history와 opcode hash table을 할당 해제하고 프로그램을 종료한다.

3.1.1 int main()

- main 함수는 set\_history와 set\_hash\_table 함수를 통한 linked list 생성과 반복문 내에서 명령어

입력(scan\_order)과 실행(order\_link) 함수 호출, 이후 프로그램 종료 전 free\_history와

free\_hash\_table을 통해 linked list 할당 해제하는 기능을 한다.

- 사용 변수 :

char order[15] : 입력받은 명령어를 저장한다.

Int error\_check : 입력받은 명령어로 order\_link() 함수에서 제대로 실행했는지 확인한다.

만약 q(uit) 명령어가 정상 실행됐다면 3이 저장되며 반복문을 종료한다.

- 호출 함수 :

void set\_history() : history 명령어를 위한 linked list를 초기화한다.

void set\_hash\_table() : opcode.txt 파일에서 mnemonic을 읽어 hash table을 작성한다.

int scan\_order(char\* order) : order에 입력받은 명령어를 저장한다.

int order\_link(char\* order) : order에 저장된 명령어에 해당하는 함수를 호출한다.

3.2 명령어 입력 구현

명령어 입력은 명령어를 입력받아 저장하는 scan\_order 함수와 입력받은 명령어를 분석해 해당 명령을 실행하는 함수로 연결해 주는 order\_link 및 order\_to\_num 함수, 명령어를 모두 입력받은 뒤 입력 버퍼에 남아있는 데이터를 지워주는 dump\_rest 함수가 있다.

인식하는 입력은 다음과 같다.

인자가 없을 경우 “order”

인자가 1개일 경우 “order” + ‘ ‘ + “data1”

인자가 2개일 경우 “order” + ‘ ‘ + “data1” + ‘,’ + ‘ ‘ + “data2”

인자가 3개일 경우 “order” + ‘ ‘ + “data1” + ‘,’ + ‘ ‘ + “data2” + ‘,’ + ‘ ‘ + “data3”

해당 형식으로 들어오는 입력만 인식하고 이외의 경우는 예외처리한다.

3.2.1 전역 변수

int rest\_exist : 입력을 받을 때, 처음 오는 명령어 뒤에 인자가 있는지 확인하는 변수

3.2.2 int scan\_order(char\* order)

- scan\_order 함수는 scanf()를 통해 character 단위로 입력받아 ‘ ‘ 또는 ‘\n’이 오기 전까지의 입력을

명령어로 저장하는 역할을 한다. 이후 입력이 ‘ ‘일 경우 rest\_exist = 1로 설정해 준다.

만약 입력이 너무 길 경우(15글자 이상) 입력을 더 이상 받지 않고 에러처리한다.

- 사용 변수 :

char c : 문자를 하나씩 입력받아 order에 저장한다.

3.2.3 int order\_link(char\* order)

- order\_link 함수는 order\_to\_num 함수를 호출해 입력받은 명령어를 숫자로 바꾸어준다. 그 이후 바뀐

숫자로 해당하는 명령 함수를 호출해 준다. 그리고 명령 함수의 반환값을 반환한다.

- 사용 변수 :

int order\_num : order\_to\_num 함수를 통해 숫자로 변환된 명령어를 저장한다.

int error\_check : 호출한 명령 함수의 반환값을 저장한다. -1이면 오류, 0이면 정상실행, 3이면 quit.

- 호출 함수 :

int order\_to\_num(char\* order) : order을 입력받아 해당하는 숫자를 반환한다.

int function\_\*() : 10개의 명령어의 실행 파일들.

3.2.4 int order\_to\_num(char\* order)

- order\_to\_num 함수는 order를 가능한 명령어(“help”, “dir” 등)와 비교해 일치할 경우 그 명령어에

해당하는 숫자를 반환한다. 해당 명령어는 1부터 10까지 있으며, 아무것에도 해당되지 않을 경우 0을

반환한다.

3.2.5 void dump\_rest()

- dump\_rest 함수는 scanf(“%[^\n]”)와 scanf(“%\*c”)를 통해 입력 버퍼에 남아있는 문자를 모두 제거한

뒤 에러 메시지를 출력하는 함수이다.

3.3 help 명령어 구현

help 명령어는 인자를 가지지 않으며, 호출 시 10개의 명령어를 출력한다.

3.3.1 int function\_help()

- function\_help 함수는 rest\_exist를 확인해 추가적인 인자가 있는지 보고, 없을 경우 명령어 목록을

출력한다. 인자가 있을 경우에는 에러처리한다.

3.4 dir 명령어 구현

dir 명령어는 인자를 가지지 않으며, 호출 시 현재 디렉토리에 있는 파일과 폴더의 이름을 출력한다.

3.4.1 int function\_dir()

- function\_dir 함수는 rest\_exist를 확인해 추가적인 인자가 있는지 보고, 없을 경우 opendir 함수를

통해 현재 디렉토리를 확인하고 readdir 함수로 파일명을 하나씩 읽고 출력한다. 실행 파일은 \*표시를

디렉토리는 /표시를 한다. 만약 인자가 있을 경우 에러처리 후 종료한다.

3.5 quit 명령어 구현

quit 명령어는 인자를 가지지 않으며, 호출 시 프로그램이 종료되도록 한다.

3.5.1 function\_quit()

- function\_quit 함수는 rest\_exist를 확인해 있을 경우 에러처리한다. 없을 경우, 3을 반환한다. 이는

메인함수까지 전달되고, 메인 함수에서 이를 확인할 경우 반복문을 중단하고 프로그램을 마친다.

3.6 history 명령어 구현

history 명령어는 인자를 가지지 않으며, 호출 시 history linked list에 있는 정보를 차례로 출력한다. 또한 모든 명령어를 실행 시 명령어의 번호와 인자를 linked list에 저장한다.

3.6.1 구조체 및 전역 변수

struct history : int num(노드의 index 저장), int order\_num(명령어를 숫자로 바꾸어 저장),

char rest(명령어 외의 인자를 문자열 형태로 저장), history\* next(다음 노드의 주소값 저장)

hptr(=history\*) head : linked list의 첫 노드를 가리킨다. 전체 linked list를 읽을 때 사용한다.

hptr tail : linked list의 마지막 노드를 가리킨다. 노드를 추가할 때 사용한다.

3.6.2 function\_history()

- function\_history 함수는 rest\_exist를 확인해 1일 경우 에러처리한다. 0일 경우 다른 함수 실행 시

추가된 history linked list를 처음부터 끝까지 확인한다.

- 사용 변수 :

hptr reader : head 부터 linked list의 끝까지 이동하며 데이터를 읽는다.

3.6.3 void set\_history()

- set\_history 함수는 main 함수 initialize 단계에서 실행되는 함수로, 전역 변수인 head에 history

structure을 할당해 주고, tail 변수 역시 head structure을 가리키도록 한다.

3.6.4 void insert\_history(int order\_num, char\* rest)

- insert\_history 함수는 모든 명령어 함수의 정상 실행시마다 호출되는 함수로, 현재 tail 포인터가 가리

키는 structure의 다음 structure를 할당하고, 넘겨받은 데이터를 저장한다. 인자가 여러 개인 명령어

에서 set\_history를 호출할 경우 미리 sprintf 함수를 사용해 문자열형으로 묶어서 인자로 받는다.

# insert\_history 함수는 모든 function\_\*함수에서 호출되므로 각 함수의 호출 함수 목록에서는

insert\_history를 생략한다.

3.6.5 void print\_order(int order\_num)

- print\_order 함수는 order number를 받아 1부터 10까지의 숫자에 해당하는 명령어를 출력해 준다.

3.6.6 void free\_history()

- free\_history 함수는 main 함수의 종료 전 호출되는 함수로, head부터 시작해 history linked list 상에 할당된 structure을 모두 할당 해제한다.

3.7 dump 명령어 구현

dump 명령어는 인자를 0개에서 2개까지 가질 수 있다. dump 명령어는 memory 범위 내에 있는 메모리를 모두 읽어온다. 인자가 2개일 경우 두 인자가 start와 end범위가 되고, 1개일 경우는 입력받은 주소값부터 160개의 데이터를, 0개일 경우에는 마지막 dump함수의 end범위 다음부터 160개의 데이터를 읽는다.

3.7.1 전역 변수

unsigned char memory[65536][16] : memory에는 0x00부터 0xFF까지의 값이 저장된다. dump, edit, fill,

reset 명령어를 통해 memory에 있는 값이 변하거나 출력된다.

const int memory\_end = 1048576 : 메모리의 마지막 주소값 + 1이다.(65536 \* 16)

int last\_dump : last\_dump에는 가장 최근 dump 탐색한 경우의 end값이 저장되어 있다. 인자가 없이

dump 명령어가 실행될 경우 시작 주소값을 last\_dump에 저장된 값으로 한다.

3.7.2 int function\_dump()

- function\_dump는 0~2개의 인자를 가진다. rest\_exist를 확인해 인자를 추가로 입력받을 지 결정한다.

인자가 존재한다면 start, end 순으로 입력을 받는다. 이후 범위가 맞게 설정되어있는지 체크한다. 이후

print\_memory 함수를 호출해 start부터 end 범위 사이에 있는 값을 줄 단위로 출력한다.

- 사용 변수 :

int st : 탐색 시작값을 저장한다. 인자가 없을 경우 last\_dump 값이 저장된다. 0 이상 memory\_end

미만의 값을 가져야 한다.

int en : 탐색 종료값+1을 저장한다. 인자가 없을 경우 st + 160 값이 저장된다. 0 초과 memory\_end

이하의 값을 가져야 한다. st보다 큰 값을 가져야 한다.

int var\_num : 이후 입력받을 인자의 개수를 저장한다.

char c : 인자 입력 사이 ‘,’ 등을 처리할 때 사용한다.

char rest\_to\_word[30] : 인자를 문자열 값으로 저장한다. 이후 이를 insert\_history에 사용된다.

- 호출 함수 :

void print\_memory(int st, int en) : 해당 범위 사이에 있는 memory의 데이터를 출력한다.

3.7.3 void print\_memory(int st, int en)

- print\_memory는 0번째 주소부터 65536번째 주소까지 각 줄마다 print\_memory\_by\_line을 호출한다.

- 사용 변수 :

int i : 0부터 65536까지 변하며 각 줄을 나타낸다.

- 호출 함수 :

void print\_memory\_by\_line(int line, int st, int en) : 확인할 줄, 탐색 범위를 받아 해당 줄이 탐색

범위 안에 있으면 출력한다.

3.7.4 void print\_memory\_by\_line(int line, int st, int en)

- print\_memory\_by\_line은 memory의 line번째 줄의 데이터가 st 이상 en 미만의 범위에 포함되는지

확인한다. 줄 전체가 범위 바깥에 있을 경우 아무것도 출력하지 않고 종료하고, 줄 전체가 범위 내에 있을

경우 해당 줄의 시작 주소값을 5자리 16진수로 표현하고, 16개의 데이터를 16진수로, character형으로

한 번씩 출력한다. 만약 일부만 걸쳐 있을 경우 범위 바깥은 16진수로는 공백, character형으로는 ‘.’으로

바꿔서 표현한다. (character 형으로 표현 시 32~126까지의 수만 표현하고 그 외에는 ‘.’으로 표현한다.)

- 사용 변수 :

int line\_st : line 번째 줄의 시작 값이다. 16\*line이 저장된다. 이 값이 en 이상일 경우 해당 줄은

범위 바깥에 있다.

int line\_en : line 번째 줄의 마지막 값 + 1이다. 16\*line + 16이 저장된다. 이 값이 st 이하일 경우

해당 줄은 범위 바깥에 있다.

unsigned char mem : memory의 line번째 줄의 0~15번째 값을 차례로 가지는 변수이다. mem에

저장된 값을 16진수/character형으로 출력한다.

3.8 edit 명령어 구현

edit 명령어는 인자를 2개 가진다. edit 명령어는 주어진 memory 주소값에 있는 데이터를 수정한다. 첫 번째 인자는 address, 두 번째 인자는 수정할 value가 된다.

3.8.1 int function\_edit()

- function\_edit은 주소값을 입력받아 memory의 해당 번지에 있는 값을 value 값으로 수정한다.

address는 0 이상 memory\_end(2^64) 미만의 값을, value는 0x20 이상 0x7E 이하의 값을 가져야 한다.

- 사용 변수 :

int pos : 수정할 메모리의 주소값이 저장된다. 실제 탐색 위치는 memory[pos/16][pos%16]이다.

int val : 수정할 메모리의 값이 저장된다. pos를 통해 탐색한 값에 val의 값을 저장해 준다.

char rest\_to\_word[30] : 입력받은 인자를 문자열로 바꾸어 저장한다. 이후 insert\_history함수를

통해 history를 저장할 때 사용한다.

3.9 fill 명령어 구현

fill 명령어는 범위의 start, end와 바꿀 value까지 총 3개의 인자를 가진다. memory의 start 부터 end 주소값에 있는 데이터를 모두 value의 값으로 수정한다. 범위는 start와 end(>=start)는 memory 범위 이내, value값은 0x20이상 0x7E 이하의 값이어야 한다.

3.9.1 int function\_fill()

- function\_fill은 주소 범위와 변경 값을 입력받아 memory의 해당 범위에 있는 값을 모두 value 값으로

수정한다. 범위 설정이 맞게 되었는지, 입력받은 값이 저장 가능한 범위에 있는지 확인해 에러가 있을

경우 실행하지 않고 에러 메시지를 출력한다.

- 사용 변수 :

int st : 수정 범위의 시작값이 저장된다. 0 이상 memory\_end 미만이어야 한다.

int en : 수정 범위의 마지막 값 + 1이 저장된다. st보다 크고, 0 초과 memory\_end 미만이어야 한다.

int val : 수정할 값이 저장된다. 0x20 이상 0x7E 이하의 값을 가져야 한다.

char rest\_to\_word[30] : 입력받은 인자들이 문자열 형태로 바뀌어 저장된다.

3.10 reset 명령어 구현

reset 명령어는 memory에 있는 모든 데이터를 0으로 바꾸어 준다. 만약 인자를 받을 경우 에러처리 후 종료한다.

3.10.1 function\_reset()

- function\_reset은 memory의 전체 범위를 반복문을 통해 전부 0으로 초기화한다. 만약 인자가 있을

경우 에러처리 후 종료한다.

3.11 opcode 명령어 구현

opcode 명령어는 mnemonic 1개를 인자로 가진다. opcode hash table로부터 자료를 읽어 해당 mnemonic의 정보를 확인하고 출력해 준다. 만약 일치하는 mnemonic을 찾지 못했을 경우 에러메시지를 보내고 종료한다.

3.11.1 구조체 및 전역 변수

struct optable : int num(mnemonic의 번호 저장), char name[10](mnemonic의 이름 저장),

char data[5](opcode 파일의 부가 정보 저장), optalbe\* next(다음 노드의 주소값 저장)

optr\*(=optable\*\*) opcode : hash table의 시작 배열 20개를 할당해 가지고 있다. 전체 hash table을 탐색

할 때 사용한다.

3.11.2 function\_opcode()

- function\_opcode는 1개의 인자를 가진다. 인자를 mnemonic 문자열에 저장한 뒤 word\_to\_hash 함수

를 통해 입력받은 mnemonic의 hash를 확인한다. 이후 hash table을 따라가며 해당 mnemonic을 탐색

한다. 찾지 못할 경우 에러 메시지를 출력한다. 찾게 되면 해당 mnemonic의 opcode를 출력한다.

- 사용 번수

char mnemonic[10] : 인자를 입력받아 저장한다. 이후 word\_to\_hash 함수를 통해 hash값을 탐색

하고, hash table을 따라가며 mnemonic과 저장된 데이터를 비교해 확인한다.

optr now : opcode의 hash번째 index부터 링크를 따라가며 optable 구조체를 탐색한다.

- 호출 함수

int word\_to\_hash(char\* name) : 입력받은 mnemonic의 hash값을 확인한다.

3.11.3 set\_hash\_table()

- set\_hash\_table 함수는 메인 함수에서 프로그램 시작 시 “opcode.txt” 파일을 읽어 mnemonic으로

hash table을 만든다. hash table의 폭은 20이고, linked list 형태로 중복 문제를 피했다. hash 값은

각 자리의 알파벳을 A를 0부터 Z를 25까지로 전환해 모두 더한 뒤 20으로 나눈 값이다. table의 각 열의

마지막에는 num에는 -1이 next에는 NULL이 저장되어 있다.

- 사용 변수

FILE\* fp : “input.txt”를 읽기 전용으로 열기 위해 사용하는 파일 포인터이다.

int num / char name[10] / char s[5] : opcode의 정보를 입력받아 저장하는 변수이다.

int index : 입력받은 name으로 구한 hash 값이 저장된다.

optr now : index를 통해 opcode[index]부터 링크를 통해 마지막까지 이동해 새 노드를 할당한다.

- 호출 함수

int word\_to\_hash(char\* name) : “opcode.txt” 파일에서 입력받은 mnemonic의 이름으로 hash값을

3.11.4 word\_to\_hash(char\* name)

- word\_to\_hash 함수는 문자열을 입력받아 해당 문자열의 hash값을 반환하는 함수이다. hash table의

크기가 20이기 때문에 hash값은 0에서 19까지 나올 수 있다. hash 값을 구하는 공식은 알파벳 A = 0,

B = 1, …, Z = 25로 숫자로 바꾼 뒤, 해당 문자열의 값을 모두 더해 20으로 나눈 값이 hash값이다.

- 사용 변수

int index : hash값을 저장하고 있는 변수로, 각 자리 알파벳의 변환값을 더해 저장한다.

3.11.5 free\_hash\_table()

- free\_hash\_table 함수는 메인 함수에서 반복문이 종료될 때 호출되는 함수이다. 먼저 hash table의 각

row를 할당 해제한 뒤, hash table의 각 row의 시작값을 가지고 있는 opcode를 할당 해제한다.

- 사용 변수

optr now : hash table을 link를 타고 차례로 이동하며 할당 해제할 structure를 가리킨다.

optr next : 다음으로 삭제할 주소값을 저장하는 포인터이다.

3.12 opcodelist 명령어 구현

opcodelist 명령어는 hash table의 정보를 읽어 0번째 index부터 19번째 index까지 차례대로 mnemonic의 이름과 opcode를 출력한다. 인자를 가지지 않으며, 인자가 없을 경우 에러메시지를 출력한다.

3.12.1 function\_opcodelist()

- function\_opcodelist는 set\_hash\_table 함수에서 만든 opcode hash table을 index 0부터 19까지 읽고

화면에 출력하는 함수이다. 이 경우 hash table 전체를 출력해야 하므로, hash값을 확인할 필요는 없다.

- 사용 변수

optr now : 전역 변수 opcode의 각 index의 시작점부터 마지막까지 이동하며 노드를 확인한다.

4 코드 설명

4.1 20161563.h

/\* structures \*/

typedef struct \_history {

int num;

int order\_num;

char rest[100];

struct \_history\* next;

} history;

typedef history\* hptr;

typedef struct \_optable {

int num;

char name[10];

char data[5];

struct \_optable\* next;

} optable;

typedef optable\* optr;

/\* functions \*/

// inputs

int scan\_order(char\*); // get order from keyboard

int order\_link(char\*); // from input, excute order function

int order\_to\_num(char\*); // change order to number

void dump\_rest(); // scan left words

// order help

int function\_help();

// order dir

int function\_dir();

// order quit

int function\_quit();

// order history

int function\_history();

void set\_history(); // make head pointer of history

void insert\_history(int, char\*); // insert data of order to history

void print\_order(int); // from order number, print order in word

void free\_history(); // free all history structures

// order dump

int function\_dump();

void print\_memory(int, int); // from start & end, print memory in range

void print\_memory\_by\_line(int, int, int); // print memory by line

// order edit

int function\_edit();

// order fill

int function\_fill();

// order reset

int function\_reset();

// order opcode

int function\_opcode();

void set\_hash\_table(); // make hash table

int word\_to\_hash(char\*); // change opcode to hash

void free\_hash\_table(); // free hash table

// order opcodelist

int function\_opcodelist();

/\* global variables \*/

// input

int rest\_exist; // 1 if rest inputs exist

// history

hptr head, tail; // pointer for checking history

// memory

unsigned char memory[65536][16]; // where memory are saved

const int memory\_end = 1048576; // value of 65536 \* 16

// dump function

int last\_dump = 0; // the last end range of dump order

// opcode table

optr\* opcode; // pointer for reading hash table

4.2 20161563.c

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <dirent.h>

#include <sys/stat.h>

#include "20161563.h"

**/\* inputs \*/**

// scanf order from input

int scan\_order(char\* order)

{

char c;

rest\_exist = 0;

// scanf order until ' ' or '\n' comes.

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

{

if(i >= 15) // if input is too long, skip the input

{

scanf("%\*[^\n]"); // get input until '\n'

scanf("%\*c"); // get '\n'

printf("Please input valid order\n");

return 1;

}

// check if rest exists & set rest\_exist variable

scanf("%c", &c);

if(c == ' ' || c == '\n')

{

if(c == ' ')

rest\_exist = 1;

order[i] = '\0';

break;

}

order[i] = c;

}

return 0;

}

// get order and excute that function

int order\_link(char\* order)

{

// get the number of order(1 to 10)

int order\_num = order\_to\_num(order);

int error\_check = 0;

if (order\_num == 1) error\_check = function\_help();

else if (order\_num == 2) error\_check = function\_dir();

else if (order\_num == 3) error\_check = function\_quit();

else if (order\_num == 4) error\_check = function\_history();

else if (order\_num == 5) error\_check = function\_dump();

else if (order\_num == 6) error\_check = function\_edit();

else if (order\_num == 7) error\_check = function\_fill();

else if (order\_num == 8) error\_check = function\_reset();

else if (order\_num == 9) error\_check = function\_opcode();

else if (order\_num == 10)error\_check = function\_opcodelist();

return error\_check;

/\*

error\_check == -1 -> real error

error\_check == 0 -> no error

error\_check == 3 -> quit order

\*/

}

// change valid order to number 1 to 10. in error case, return 0

int order\_to\_num(char\* s)

{

// change valid order to number 1 to 10

if(strcmp(s, "h") == 0 || strcmp(s, "help") == 0) return 1;

if(strcmp(s, "d") == 0 || strcmp(s, "dir") == 0) return 2;

if(strcmp(s, "q") == 0 || strcmp(s, "quit") == 0) return 3;

if(strcmp(s, "hi")== 0 || strcmp(s, "history")== 0) return 4;

if(strcmp(s, "du")== 0 || strcmp(s, "dump") == 0) return 5;

if(strcmp(s, "e") == 0 || strcmp(s, "edit") == 0) return 6;

if(strcmp(s, "f") == 0 || strcmp(s, "fill") == 0) return 7;

if(strcmp(s, "reset") == 0) return 8;

if(strcmp(s, "opcode") == 0) return 9;

if(strcmp(s, "opcodelist")== 0) return 10;

// if unvalid order, return 0

return 0;

}

// get rest inputs and print error message

void dump\_rest()

{

scanf("%\*[^\n]");

scanf("%\*c");

printf("Please input valid order\n");

return;

}

**/\* help(1) : print name of orders \*/**

int function\_help()

{

// check additional word -> error

if(rest\_exist == 1)

{

dump\_rest();

return -1;

}

// make history record

insert\_history(1, "");

// print all orders

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

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

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

return 0;

}

**/\* dir(2) : print files in current directory \*/**

int function\_dir()

{

// check additional word -> error

if(rest\_exist == 1)

{

dump\_rest();

return -1;

}

// make history record

insert\_history(2, "");

// read files from current directory

DIR\* now\_dir = opendir("./");

if(now\_dir != NULL)

{

struct dirent\* inp;

struct stat buf;

for(int i = 0; (inp = readdir(now\_dir)) != NULL; i++)

{

lstat(inp->d\_name, &buf);

if(i % 4 == 0 && i != 0)

printf("\n");

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

// print additional symbol(\*, /) by the type of file

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

printf("/");

else if(S\_ISREG(buf.st\_mode))

printf("\*");

}

printf("\n");

closedir(now\_dir);

}

else // if failed to read directory, send error message

{

printf("DIRECTORY READ ERROR!\n");

return -1;

}

return 0;

}

**/\* quit(3) : quit the program \*/**

int function\_quit()

{

// check additional word -> error

if(rest\_exist == 1)

{

dump\_rest();

return -1;

}

// make history record

insert\_history(3, "");

return 3;

}

**/\* history(4) : read the history linked list from head to the end \*/**

int function\_history()

{

// check additional word -> error

if(rest\_exist == 1)

{

dump\_rest();

return -1;

}

// make history record

insert\_history(4, "");

// check if history exists

if(head->next == NULL)

{

printf("NO DATA IN HISTORY!\n");

return -1;

}

// read history linked list from head to end & print all data

hptr reader = head->next;

while(1)

{

printf("%-5d", reader->num);

print\_order(reader->order\_num);

printf("%s\n", reader->rest);

if(reader->next == NULL)

break;

reader = reader->next;

}

return 0;

}

// malloc head & tail pointer of history

void set\_history()

{

head = (hptr)malloc(sizeof(history));

head->num = 0;

head->next = NULL;

tail = head;

return;

}

// insert history in linked list

void insert\_history(int order\_num, char\* rest)

{

tail->next = (hptr)malloc(sizeof(history));

tail->next->num = tail->num + 1;

tail->next->order\_num = order\_num;

strcpy(tail->next->rest, rest);

tail->next->next = NULL;

tail = tail->next;

return;

}

// get order number and print in sentense (only used for history)

void print\_order(int order\_num)

{

if (order\_num == 1) printf("h[elp] ");

else if (order\_num == 2) printf("d[ir] ");

else if (order\_num == 3) printf("q[uit] ");

else if (order\_num == 4) printf("hi[story] ");

else if (order\_num == 5) printf("du[mp] ");

else if (order\_num == 6) printf("e[dit] ");

else if (order\_num == 7) printf("f[ill] ");

else if (order\_num == 8) printf("reset ");

else if (order\_num == 9) printf("opcode ");

else if (order\_num == 10)printf("opcodelist ");

return;

}

// free history linked list

void free\_history()

{

hptr next;

while(1)

{

next = head->next;

free(head);

if(next == NULL)

break;

head = next;

}

return;

}

**/\* dump(5) : get range(start, end) and print data in memory \*/**

int function\_dump()

{

int st, en;

int var\_num = 0;

// if any words behind, scan them int hexadecimal

if(rest\_exist == 1)

{

scanf("%x", &st);

// check if start is in range

if(st >= memory\_end || st < 0)

{

printf("out of range\n");

scanf("%\*[^\n]");

scanf("%\*c");

return -1;

}

var\_num = 1;

char c;

scanf("%c", &c);

if(c == '\n') // one number (start)

{

en = st + 160;

if(en > memory\_end)

en = memory\_end;

last\_dump = en;

if(last\_dump == memory\_end)

last\_dump = 0;

}

else if(c == ',') // two numbers (start, end)

{

scanf("%x", &en);

en++;

var\_num = 2;

// scanf others left

scanf("%\*[^\n]");

scanf("%\*c");

// check if start < end + 1

if(st >= en)

{

printf("end must be same or bigger than start\n");

return -1;

}

// check if end is in range

if(en > memory\_end)

{

printf("out of range\n");

return -1;

}

last\_dump = en;

if(last\_dump == memory\_end)

last\_dump = 0;

}

else // input error

{

dump\_rest();

return -1;

}

}

else // no letter behind

{

st = last\_dump;

en = st + 160;

if(en > memory\_end)

en = memory\_end;

last\_dump = en;

if(last\_dump == memory\_end)

last\_dump = 0;

}

// print memory in range

print\_memory(st, en);

// make history record

if(var\_num == 0)

insert\_history(5, "");

else

{

char rest\_to\_word[30];

if(var\_num == 1)

sprintf(rest\_to\_word, "%X", st);

else

sprintf(rest\_to\_word, "%X, %X", st, en - 1);

insert\_history(5, rest\_to\_word);

}

return 0;

}

// check range and call print function by line

void print\_memory(int st, int en)

{

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

print\_memory\_by\_line(i, st, en);

return;

}

// check range and print data in one given line

void print\_memory\_by\_line(int line, int st, int en)

{

int line\_st = line \* 16;

int line\_en = line \* 16 + 16;

// check if the line is out of range

if(en <= line\_st || line\_en <= st)

return;

// print first address

printf("%05X ", line\_st);

// print data of line in hexadecimal

for(int i = line\_st; i < line\_en; i++)

{

unsigned char mem = memory[line][i - line\_st];

if(st <= i && i < en)

printf("%02X ", mem);

else

printf(" ");

}

printf("; ");

// print data of line in character

for(int i = line\_st; i < line\_en; i++)

{

unsigned char mem = memory[line][i - line\_st];

if(st <= i && i < en)

if(mem >= 32 && mem <= 126)

printf("%c", mem);

else

printf(".");

else

printf(".");

}

printf("\n");

return;

}

**/\* edit(6) : change a given address of memory to a given value \*/**

int function\_edit()

{

// check additional word -> if not, error

if(rest\_exist == 0)

{

dump\_rest();

return -1;

}

int pos, val;

scanf("%x, %x", &pos, &val);

scanf other words behind

scanf("%\*[^\n]");

scanf("%\*c");

// check if address is in range

if (pos < 0 || pos >= memory\_end)

{

printf("out of range\n");

return -1;

}

// check if value is in data range

if(val < 32 || val > 126)

{

printf("input value between 20 and 7E (in hexadeximal)\n");

return -1;

}

memory[pos/16][pos%16] = val;

// make history record

char rest\_to\_word[30];

sprintf(rest\_to\_word, "%X, %X", pos, val);

insert\_history(6, rest\_to\_word);

return 0;

}

**/\* fill(7) : set memory from start to end by given value \*/**

int function\_fill()

{

// check additional word -> if not, error

if(rest\_exist == 0)

{

printf("input valid order\n");

return -1;

}

int st, en, val;

scanf("%x, %x, %x", &st, &en, &val);

en++;

scanf("%\*[^\n]");

scanf("%\*c");

// check inputs are in valid range

if(st < 0 || en < 0 || st >= memory\_end || en > memory\_end)

{

printf("out of range\n");

return -1;

}

if(st >= en)

{

printf("end must be same or bigger than start\n");

return -1;

}

if(val < 32 || val > 126)

{

printf("input value between 20 and 7E (in hexadeximal)\n");

return -1;

}

// change data of memory

for(int i = st; i < en; i++)

memory[i/16][i%16] = val;

// make history record

char rest\_to\_word[30];

sprintf(rest\_to\_word, "%X, %X, %X", st, en - 1, val);

insert\_history(7, rest\_to\_word);

return 0;

}

**/\* reset(8) : change every data in memory to 0 \*/**

int function\_reset()

{

// check additional word -> error

if(rest\_exist == 1)

{

dump\_rest();

return -1;

}

// set all memory to 0

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

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

memory[i][j] = 0;

// make history record

insert\_history(8, "");

return 0;

}

**/\* opcode(9) : get mnemonnc and print its number \*/**

int function\_opcode()

{

// check additional word -> if not, error

if(rest\_exist == 0)

{

printf("input valid order\n");

return -1;

}

char mnemonic[10];

scanf("%s", mnemonic);

// dump words left in input buffer

scanf("%\*[^\n]");

scanf("%\*c");

// change mnemonic to hash

int index = word\_to\_hash(mnemonic);

optr now = opcode[index];

// check mnemonic in hash table

while(1)

{

if(strcmp(now->name, mnemonic) == 0)

{

printf("opcode is %X\n", now->num);

break;

}

// last structure has num = -1 & next = NULL

if(now->num == -1)

{

printf("there is no mnemonic\n");

return -1;

}

now = now->next;

}

// make history record

insert\_history(9, mnemonic);

return 0;

}

// make hash table

void set\_hash\_table()

{

opcode = (optr\*)malloc(20 \* sizeof(optr));

// make empty hash table

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

{

opcode[i] = (optr)malloc(sizeof(optable));

opcode[i]->num = -1;

opcode[i]->next = NULL;

}

// open "opcode.txt" file

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

if(fp == NULL)

{

printf("cannot read opcode file\n");

return;

}

int num;

char name[10], s[5];

// scanf "opcode.txt" file and make hash\_table

while(fscanf(fp, "%x %s %s", &num, name, s) != EOF)

{

int index = word\_to\_hash(name);

optr now = opcode[index];

while(1)

{

if(now->num == -1)

{

now->num = num;

strcpy(now->name, name);

strcpy(now->data, s);

now->next = (optr)malloc(sizeof(optable));

now = now->next;

now->num = -1;

now->next = NULL;

break;

}

else

now = now->next;

}

}

return;

}

// change mnemonic to hash number

int word\_to\_hash(char\* name)

{

int index = 0;

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

index += name[i] - 'A';

return index % 20;

}

// free hash table

void free\_hash\_table()

{

// free every row of table (0 to 19)

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

{

optr now, next;

now = opcode[i];

while(1)

{

next = now->next;

free(now);

if(next->num == -1)

{

free(next);

break;

}

now = next;

}

}

free(opcode);

return;

}

**/\* opcodelist(10) : print all mnamonic data in hash table \*/**

int function\_opcodelist()

{

// check additional word -> error

if(rest\_exist == 1)

{

dump\_rest();

return -1;

}

// print every data of hash table

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

{

printf("%d : ", i);

optr now = opcode[i];

while(1)

{

printf("[%s,%02X]", now->name, now->num);

now = now->next;

if(now->num == -1)

break;

printf(" -> ");

}

printf("\n");

}

// make history record

insert\_history(10, "");

return 0;

}

**/\* main : initialize history & hash table and recursively get order and excute it \*/**

int main()

{

// initialize history and hash table

set\_history();

set\_hash\_table();

while(1)

{

printf("sicsim>");

char order[15];

if(scan\_order(order)) // if wrong order, return 1.

{

printf("Please input valid order\n");

continue;

}

// read order number & link to each order

int error\_check = order\_link(order);

if(error\_check == 3) // case of quit order

break;

}

// free history linked list and opcode hash table

free\_history();

free\_hash\_table();

return 0;

}

1. 원활한 설명을 위해 모듈 정의 단계와 전역 변수 정의 단계를 합쳤습니다. [↑](#footnote-ref-1)