과목 명: 시스템프로그래밍

담당 교수 명: 박 운 상

<<Assignment 1>>

서강대학교 컴퓨터학과

[학번] 20151623

[이름] 한상구

목 차

1. 프로그램 개요

이 프로그램은 SIC/XE머신을 구현하기 위해 작성되었습니다. 기본적인 shell 명령어들을 수행할 수 있으며, 이후 어셈블러, 링커, 로더들을 실행하게 될 것입니다. 컴파일을 통해 만들어진 .o file (object code)가 적재되고 실행될 메모리공간과 mnemonic을 opcode로 변환하는 opcode table을 구현했습니다.

주어진 명령어뿐만 아니라, cat, cmp, copy, touch, head, echo 등 추가적으로 shell에서 사용되는 명령어도 구현하였습니다.

2. 프로그램 설명

2.1 프로그램 흐름도

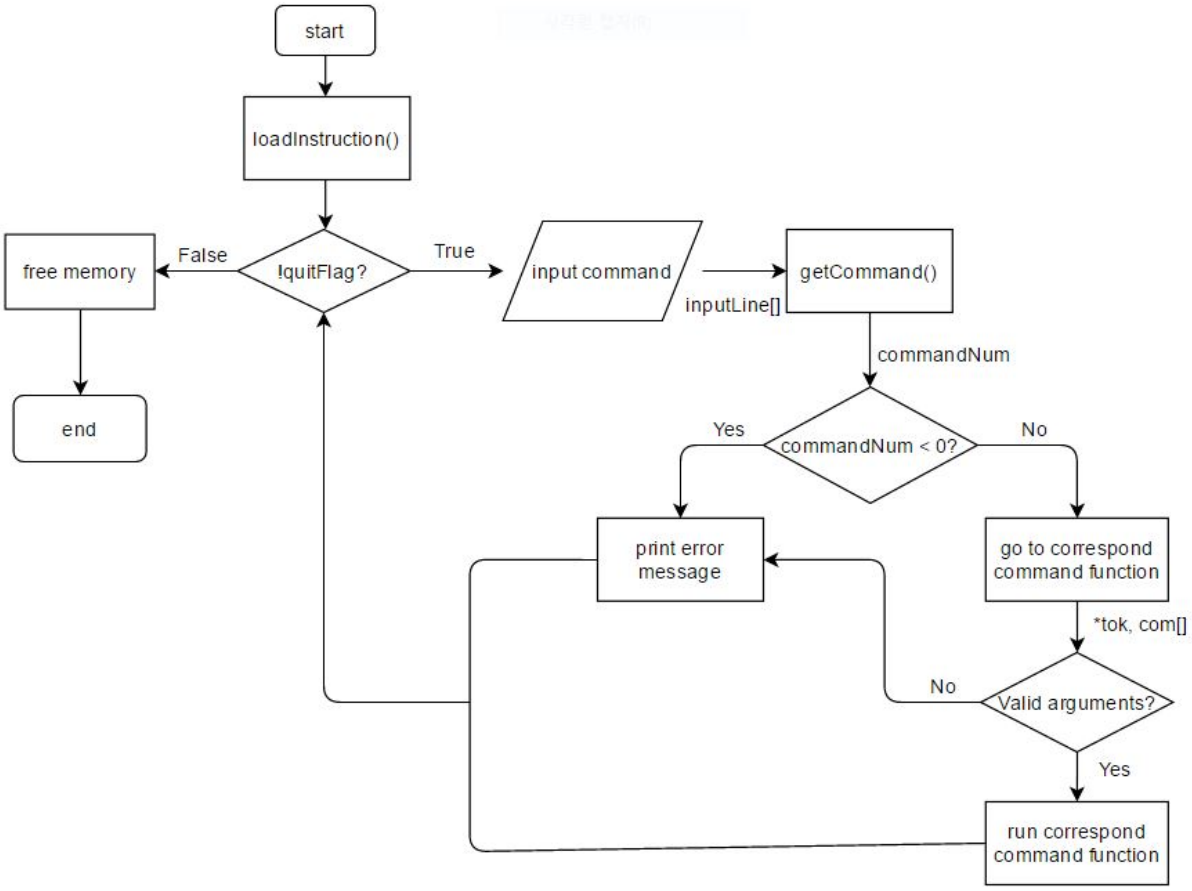

그림 1> 프로그램 흐름도

3. 모듈 정의

3.1 모듈 이름 : main()

3.1.1 기능

Shell 을 실행하며, quitFlag 가 1 로 설정되기 전까지 사용자에게 command 를 입력받아 유효한 경우 그 command 를 수행, 적절한 결과를 출력 혹은 저장한다. shell 종료 이후에는 동적으로 할당된 메모리를 전부 deallocating 해준다.

3.1.2 사용 변수

int commandNum – shell 을 실행하며 적절한 command 인 경우 그에 해당하는 command number 을 저장한다. 유효하지 않은 입력시 '-1'의 값을 가진다.
int i – shell 종료 시, opcode table deallocating 시 사용되는 loop 를 위한 변수.
char inputLine[111] – shell 실행 중 사용자에게 입력 받는 line 전체를 저장하는 문자열.
char *tok – inputLine 의 elemnet 를 때에 따라 적절한 위치를 가리키는 pointer.
History *curr – linked list deallocating 시 사용되는 pointer.
Instruction *it, *prev – opcode table deallocating 시 사용되는 pointer.
comFuncPtr comFunc[] – 함수포인터로써 command 에 따라 실행되는 함수가 다른 특징 때문에 불필요하게 길어져 가독성을 해치는 것을 방지하기 위해 command 를 실행하는 함수를 묶어 가리킨다. commandNum 을 통해 적절한 index 로 접근하여 그에 해당하는 command function 을 수행한다.
bool quitFlag – 1 로 설정되면 shell 을 종료한다.

3.2 모듈 이름: hashFunction(char *val)

3.2.1 기능

opcode.txt 에서 읽어온 instruction 을 bucket size 가 20 인 hash table 에 저장하기 위해 index 를 배정하는 hash function 이다. Mnemonic 의 길이를 *l* 이라고 할 때, 주어진 mnemonic 을 *l* 자리의 39 진수로 간주한다. (이 때, 각 자리의 숫자는 'A'와의 거리로 한다. 예를 들어, B = 1, C = 2, …) 이를 10 진수로 변환한 뒤, mod 20 의 int 형 값을 반환한다.
39 진수로 가정한 이유는, 26 과 relatively prime 인 여러 수 (13, 17, 19, 23, 29, 31, 43, …)를 사용하여 opcode table 을 구축한 결과 39 를 base 로 할 때 가장 균등한 bucket 의 분배를 보였기 때문이다.

3.2.2 사용 변수

int ret – 주어진 39 진수를 10 진수로 변환할 때 값을 저장하는데 쓰인다. Return value.
char *val – parameter 로 넘어오는 문자열이다. Mnemonic 을 저장하고 있으며 ret 에 담길 39 진수의 수를 담고있다.

3.3 모듈이름: getCommand(char str[], int *commandNum)

3.3.1 기능

Simulator 실행 중 입력되는 문장에서 command 만을 잘라내어 유효한 command 이면 이에 맞는 commandNum(help 실행 시 출력되는 순서대로 0 부터 9 까지 command 에 숫자를 부여했음)을 부여한다. str 이라는 문자열은 입력된 문장 전체를 담고 있으며, 본 함수에서는 command 이전의 공백 (' ', '\t')은 무시한다. 유효한 command 가 아닌 경우 commandNum 에 -1 을 저장하여 이를 명시한다.
잘라낸 이후 , parameter 가 command 뒤에 존재하면 parameter 의 첫 글자를, 존재하지 않으면 문장의 끝을 가리키는 char type pointer 를 반환한다.

3.3.2 *사용변수*

char str[] – Simulator 에 입력된 문장 전체를 담고 있는 문자열로, 본 함수에 인자로 들어온다.
char *com – char type pointer 로 잘라낸 command 의 첫 글자를 가리킨다
char tmp – char 형 변수 임시 저장소로써, command 를 임시로 잘라내어 사용할 때 command 의 마지막 글자 다음 글자를 임시로 저장한다. 이후 command 비교가 끝난 뒤, '\0'으로 설정해두었던 마지막 글자 다음 글자를 복원시킬 때 사용한다.
int *commandNum – int 형 변수의 주소로써 입력된 command 가 유효할 경우 그에 맞는 commandNum 을, 유효하지 않을 경우 -1 을 저장한다.

3.4 모듈이름: hexstrToInt(char *str)

3.4.1 *기능*

주어진 문자열인 str 을 10 진수로 변환한다. 본 함수에서 체크하는 에러는 총 3 가지로, 주어진 문자열이 16 진수의 형태인지, 양수인지, 5 자리를 넘어가지는 않는지(본 프로그램에서 사용하는 최대값은 0xFFFFF 이므로)이다. 각각에 대한 에러코드는 -1, -3, -2 이고, 에러가 발생하지 않을 시 정상적으로 16 진수를 10 진수로 변환한 값을 return 한다.

3.4.2 *사용변수*

char *str – 10 진수로 변환하기 위해 인자로 넘어오는 문자열.
int ret – 변환되는 값을 저장하는 변수.
int len – 주어진 문자열의 길이를 저장하는 변수. 변환하는 동안 자릿수를 세며, len 이 6 이될 시 본 함수는 작동을 멈추고 -2 를 return 한다.
bool errFlag – 주어진 문자열에 16 진수의 형태가 아닌 글자가 포함되어있으면 1, 아니면 0 을 갖는다. errFlag 가 1 로 설정된 경우 함수 작동을 멈추고 -1 을 return 한다.
bool negFlag – 주어진 문자열의 처음에 '-' 부호가 포함되면 1, 아니면 0 을 갖는다. negFlag 가 1 로 설정된 경우 함수 종료 시 -3 을 return 한다.

3.5 모듈이름: addHistory(char com[])

3.5.1 *기능*

com 문자열에 담겨서 넘어온 문장을 history 정보를 담고 있는 linked list 에 추가한다. 이 때, 넘어온 문자열은 유효한 command 임이 보장되어 있다.

3.5.2 *사용변수*

char com[] – 새로 추가할 history 에 입력될 문자열을 담고 있다.
History *newNode – linked list 에 추가할 node 를 dynamically allocate 하기 위해 사용할 임시변수.

3.6 모듈이름: commandHelp(char *tok, char com[])

3.6.1 *기능*

getCommand 에서 command 가 'h' 혹은 'help'인 경우 호출된다.
tok 라는 char type pointer 는 'h' 혹은 'help'이후 공백이 아닌 첫 글자를 가리키는 pointer 이다. 즉, tok 가 가리키는 글자가 '\0'이 아니라면 본 함수는 실행되면 안 된다. 이 경우, "Invalid command"라는 메시지를 화면에 출력한다.
tok 가 가리키는 글자가 '\0'이라면 본 함수가 실행되며, simulator 에서 사용할 수 있는 command 를 화면상에 출력한다. 이후, addHistory 를 호출하여 입력된 문장을 history 에 추가한다.

3.6.2 *사용 변수*

char *tok – command 이후 공백이 아닌 첫 글자를 가리키는 char type pointer.

char com[] – 입력된 문장 전체

3.7 모듈이름: commandDir(char *tok, char com[])

3.7.1 기능

getCommand 에서 command 가 'd' 혹은 'dir'인 경우 호출된다.
본 프로그램 실행파일인 '20151623.out'이 위치한 directory 의 file 들을 출력한다.
3.6 과 같은 이유로, tok 가 가리키는 글자가 '\0'일 경우만 본 함수를 실행하고 history 에 com[]을 추가한다. 아닌경우, "Invalid command"라는 메시지를 화면에 출력한다.
opendir(), readdir() 등의 함수를 통해 현재 directory 의 정보 및 directory 하위에 존재하는 파일들에 접근할 수 있고, stat() 함수를 통해 접근한 파일에 대한 보다 상세한 정보를 알아낼 수 있다.
실행파일 뒤에는 '*'이, directory 뒤에는 '/'이 따라붙고, 이는 structure stat 내의 st_mode 와 S_ISDIR, S_IXUSR 를 가지고 정의된 매크로 혹은 bitwise and 를 통해 파악하고 구현하였다.

3.7.2 사용 변수

char *tok – command 이후 공백이 아닌 첫 글자를 가리키는 char type pointer.
char com[] – 입력된 문장 전체
DIR *currDir – 현재 directory 의 정보를 담을 DIR type pointer
struct dirent *currFile – 현재 directory 에 존재하는 file 의 정보를 담은 struct dirent 의 pointer. file 의 이름을 struct dirent 의 element 인 d_name 을 통해 얻을 수 있다.
struct stat currStat – 현재 directory 에 존재하는 file 의 정보를 담은 struct stat 형 변수. *currFile 이 가리키는 것이 directory 인지 실행파일인지 등을 알려주는 정보를 담은 st_mode 라는 element 가 이 변수 내에 존재한다.

3.8 모듈이름: commandQuit(char *tok, char com[])

3.8.1 기능

Simulator 를 종료하기 위해 'q' 혹은 'quit'이라는 명령어가 입력되었는지 확인한다.
3.6 과 같은 이유로, tok 가 가리키는 글자가 '\0'이 아닌 경우 "Invalid command"라는 메시지를 화면에 출력하고, '\0'인 경우 quitFlag 를 1 로 설정한다. quitFlag 가 1 이 되는 경우, 본 simulator 가 종료되기 때문에 addHistory 는 호출하지 않는다.

3.8.2 사용 변수

char *tok – command 이후 공백이 아닌 첫 글자를 가리키는 char type pointer.
char com[] – 입력된 문장 전체
bool quitFlag – 주어진 command 가 유효할 경우 1 로 설정한다. 이 경우, simulator 는 dynamically allocating 된 memory 전부를 deallocating 한 후 종료된다.

3.9 모듈이름: commandHistory(char *tok, char com[])

3.9.1 기능

getCommand 에서 command 가 'hi' 혹은 'history'인 경우 호출된다.
Simulator 실행 중 입력된 유효한 command 들을 linked list 에 저장해두는데, 이를 순서대로 (오래된 입력 command 먼저) 화면상에 출력한다.
3.6 과 같은 이유로, rok 가 가리키는 글자가 '\0'이 아닌 경우 "Invalid command"라는 메시지를 화면에 출력하고, '\0'인 경우 저장되어있는 command 들을 화면상에 출력한다.

3.9.2 사용 변수

char *tok – command 이후 공백이 아닌 첫 글자를 가리키는 char type pointer.
char com[] – 입력된 문장 전체
int i – history 출력 시 command 좌측에 표시될 숫자를 저장하기 위한 counter

History *curr – linked list 를 훑으며 출력하기 위한 pointer
History *historyHead – history 를 저장하고 있는 linked list 의 head node

3.10 모듈이름: commandDump(char *tok, char com[])

3.10.1 기능

getCommand 에서 command 가 'du' 혹은 'dump'인 경우 호출된다.
본 simulator 는가상 메모리공간을 구현했는데, 범위가 주어진다면 주어진 범위만큼의, 아니라면 (마지막으로 출력한 주소 + 1 ~ 마지막으로 출력한 주소 + 160) 의 메모리공간을 화면상에 출력한다. 크게 세 부분으로 나누어 0x000a0 와 같이 0x10 단위의 주소를 표현을 좌측에, 중앙에는 한 줄에 0x10 개의 메모리에 저장된 값을, 우측에는 그에 대응되는 ASCII 값을 출력한다.
이 command 에서 처리한 에러는 크게 4 가지인데, 주어진 값이 0xFFFFF 를 넘어가거나 음수 (ex. –FF)가 주어진 경우, 두 인자가 주어졌으나 쉼표 (',')로 구분이 되어있지 않은 경우, 쉼표가 존재하나 그 이후 인자가 존재하지 않는 경우, start address 가 end address 보다 큰 경우이다.

3.10.2 사용 변수

char *tok – command 이후 공백이 아닌 첫 글자를 가리키는 char type pointer.
char com[] – 입력된 문장 전체
int i , j – loop 를 위한 변수.
int start – start address 가 주어진 경우 주어진 인자를, 주어지지 않은 경우 전역변수인 dumpLastAddr 값을 가진다.
int end – end address 가 주어진 경우 주어진 인자를, 주어지지 않은 경우 start + 159 의 값을 가진다.
int dumpLastAddr – 전역변수로서, 마지막으로 출력한 address + 1 의 값을 담는다.

3.11 모듈이름: commandEdit(char *tok, char com[])

3.11.1 기능

getCommand 에서 command 가 'e' 혹은 'edit'인 경우 호출된다.
구현된 가상메모리상의 한 메모리를 특정 값(0x00 – 0xFF)으로 설정한다.
이 command 에서 처리한 에러는 인자가 정확하게 주어지지 않은 경우, 쉼표로 두 인자를 구분하지 않은 경우, address 나 value 의 범위를 넘어난 경우이다.

3.11.2 사용 변수

char *tok – command 이후 공백이 아닌 첫 글자를 가리키는 char type pointer.
char com[] – 입력된 문장 전체
int addr – 인자로 넘어온 address 를 저장한다.
int val – 인자로 넘어온 value 를 저장한다.
unsigned char virtualMem – 유효한 commnad 가 입력될 시 특정 address 의 값을 value 로 설정한다.

3.12 모듈이름: commandFill(char *tok, char com[])

3.12.1 기능

getCommand 에서 command 가 'f' 혹은 'fill'인 경우 호출된다
주어진 범위의 값을 특정 값으로 설정하는 기능을 제공한다. start address 에서 end address 까지 메모리를 value 로 설정한다.
3.11 과 비슷한 에러처리를 했으며, 인자가 3 개인 것을 제외하면 비슷한 구조를 가지고 있다.

3.12.2 사용 변수

char *tok – command 이후 공백이 아닌 첫 글자를 가리키는 char type pointer.
char com[] – 입력된 문장 전체
int i – loop 를 위한 변수

int start – 인자로 넘어온 start address 를 저장한다.
int end – 인자로 넘어온 end address 를 저장한다.
int val – 인자로 넘어온 value 를 저장한다.
unsigned char virtualMem – 유효한 commnad 가 입력될 시 [start, end] address 의 값을 value 로 설정한다.

3.13 모듈이름: commandReset(char *tok, char com[])

3.13.1 기능

getCommand 에서 command 가 'reset' 인 경우 호출된다
3.11, 3.12 가 메모리 값을 원하는 값으로 설정하는 기능을 제공했다면, 본 함수는 메모리 전체를 0x00 으로 초기화하는 기능을 제공한다.
3.6 과 같은 이유로 tok 가 '\0'을 가리키는 경우만 실행된다.

3.13.2 사용 변수

char *tok – command 이후 공백이 아닌 첫 글자를 가리키는 char type pointer.
char com[] – 입력된 문장 전체
unsigned char virtualMem – command 가 유효할 시 메모리 전체를 0x00 의 값으로 초기화한다.

3.14 모듈이름: commandMnemonic(char *tok, char com[])

3.14.1 기능

getCommand 에서 command 가 'opcode'인 경우 호출된다
이후 주어진 mnemonic 이 opcode table 에 존재하는 경우 mnemonic 에 해당하는 opcode 를 화면에 출력한다.
mnemonic 이 주어지지 않은 경우나, opcode table 에 존재하지 않는 mnemonic 의 경우는 에러메시지를 출력한다.

3.14.2 사용 변수

char *tok – command 이후 공백이 아닌 첫 글자를 가리키는 char type pointer.
char com[] – 입력된 문장 전체.
int idx – 주어진 mnemonic 의 index 를 hash function 을 이용해 구한 뒤 저장한다.
char *it – tok 부터 mnemonic 까지를 잘라내기 위해 사용하는 iterator.
Instruction *curr – opcode table 에서 mnemonic 을 탐색하기 위해 사용하는 iterator.
Instruction *instructionHead[20] – opcode table.

3.15 모듈이름: commandOplist(char *tok, char com[])

3.15.1 기능

getCommand 에서 command 가 'opcodelist'인 경우 호출된다
opcode table 을 화면상에 출력한다.
3.6 과 같은 이유로 tok 가 '\0'을 가리키고 있을 때에만 동작한다.

3.15.2 사용 변수

char *tok – command 이후 공백이 아닌 첫 글자를 가리키는 char type pointer.
char com[] – 입력된 문장 전체.
int i – loop 를 위한 변수.
Instruction *it – opcode table 을 순회하며 출력하기 위해 사용하는 iterator.
Instruction *instructionHead[20] – opcode table.

3.16 모듈이름: commandCat(char *tok, char com[])

3.16.1 기능

(실행 방법) # cat [filename(s)]
getCommand 에서 command 가 'cat'인 경우 호출된다
실제 shell 에서 사용되는 cat 과 같은 기능을 수행한다.
command 이후 인자는 파일이름이 오며, 오지 않을 경우 stdin 을 통해 입력 받는다. (이 경우 'ctrl+c'등을 통해 빠져 나와야 하는데, 이 경우 shell 이 같이 종료되므로 파일이름을 입력하여 사용하자.)
정상적인 파일이름이 온 경우 그 파일을 화면상에 출력한다.
인자가 여러 개의 파일이므로, history 에 command 를 추가하는 기준은 주어진 파일 중 하나라도 정상적으로 화면에 출력되었는가 (즉, 주어진 파일 중 하나라도 directory 상에 존재하는 유효한 파일인가) 이다.
입력된 파일이 존재하지 않는 경우, 에러 메시지를 출력한다.
while(~(c = fgetc(fp))) 의 동작 원리는 다음과 같다.
1. fgetc(fp)는 fp 가 가리키는 파일에서 1byte 를 읽어온다. 이 때, c = fgetc(fp) 이므로 이 값은 c 에 저장된다.
2. fgetc 가 EOF 에 도달하였을 때, c 는 EOF 값을 저장하게 되고 이는 -1 과 같다. (gcc 를 기준)
이를 2 진수로 표현하면 1111 1111 …. 1111 과 같다.
3. ~c 를 함으로써 bitwise NOT 을 수행한다. 곧 c 는 0000 0000 … 0000 을 저장하게 되고, 이는 0 과 같다. while 문의 condition 이 0 이 되었으므로 더 이상 loop 는 수행되지 않는다.

3.16.2 사용 변수

char *tok – command 이후 공백이 아닌 첫 글자를 가리키는 char type pointer.
char com[] – 입력된 문장 전체.
FILE *fp – 파일 이름이 인자로 넘어온 경우 파일을 열기 위해 사용하는 file pointer.
char fileName[111] – 파일 이름을 저장할 char type array.
int i - tok 에서 파일 이름을 fileName 으로 복사할 때 사용할 iterator.
int c – stdin 혹은 파일에서 읽어와 화면에 출력할 때 사용할 임시 변수 저장소.
int catCounter – 정상적으로 화면에 출력된 file 의 수를 count 한다.

3.17 모듈이름: commandCmp(char *tok, char com[])

3.17.1 기능

(실행방법) # cmp filename1 filename2
getCommand 에서 command 가 'cmp'인 경우 호출된다
command 뒤에 주어진 두 파일을 비교하여 다른 부분을 출력하는 기능을 한다.
첫 번째 인자로 들어온 파일을 기준으로, 몇 번째 라인 몇 번째 바이트에서 차이가 처음으로 발생하는지 출력하며, 두 파일이 동일할 시 아무 메시지도 출력하지 않는다.
인자가 두 개가 들어오지 않는 경우, 파일이 존재하지 않는 경우는 에러처리하였다.
두 인자의 구분은 쉼표가 아니라 공백으로 하였는데, 이는 실제 shell 에서의 구분자를 따라갔다.
for 문의 ~s && ~d 는 3.16 에서 설명한 것과 같은 원리로, 두 파일 중 하나라도 EOF 를 만나는 순간 종료되는 condition 이다.

3.17.2 사용 변수

char *tok – command 이후 공백이 아닌 첫 글자를 가리키는 char type pointer.
char com[] – 입력된 문장 전체.
FILE *sfp, *dfp – 각각 첫 번째와 두 번째로 주어진 파일을 가리키는 file pointer 이다.
char sourceName[111], destName[111] – 각각 첫 번째와 두 번째로 주어진 파일의 이름을 저장하는

char type array 이다.
int i – 인자로 넘어온 파일 이름을 복사할 때 iterator 로, 두 파일을 비교할 때는 byte counter 로 사용한다.
int line – 두 파일을 비교할 때 line counter 로 사용한다
int s, d – 각 파일에서 읽어온 문자를 임시로 저장할 변수이다.
bool diffFlag – 두 파일이 같은지 다른지 저장하는 flag 이다. 두 파일이 같으면 0, 다르면 1 로 설정한다.

3.18 모듈이름: commandCopy(char *tok, char com[])

3.18.1 기능

(실행방법) # copy filename1 filename2
getCommand 에서 command 가 'copy'인 경우 호출된다
command 뒤에 주어진 파일을 그 뒤에 주어진 이름을 가진 파일로 복사하는 기능을 한다.
수행 이후 dir 등의 command 를 통해 생성됨을 확인할 수 있고, cat 혹은 cmp 등을 통해 두 파일이 동일한 지 확인할 수 있다.
역시 인자의 구분은 쉼표가 아니라 공백임에 주의해야 한다.

3.18.2 사용 변수

char *tok – command 이후 공백이 아닌 첫 글자를 가리키는 char type pointer.
char com[] – 입력된 문장 전체.
FILE *sfp, *dfp – 각각 복사할 파일과 복사되어 생성될 파일을 가리키는 file pointer 이다.
char sourceName[111], destName[111] – 각각 복사할 파일과 복사되어 생성될 파일의 이름을 저장하는 char type array 이다.
int i – 인자로 넘어온 파일 이름을 복사할 때 iterator 로 사용한다.
int c – 파일복사 시 사용할 임시 변수 저장소.

3.19 모듈이름: commandTouch(char *tok, char com[])

3.19.1 기능

(실행방법) # touch filename(s)
크기가 0 바이트인 파일을 생성한다.
이미 존재하는 파일이 인자로 넘어올 시, 그 파일에는 아무 변화도 일어나지 않는다.
인자의 구분은 쉼표가 아니라 공백임에 주의해야 한다.

3.19.2 사용 변수

char *tok – command 이후 공백이 아닌 첫 글자를 가리키는 char type pointer.
char com[] – 입력된 문장 전체.
FILE *fp – 파일을 생성할 때 사용할 file pointer.
char filename[111] – 생성할 파일의 이름을 저장할 char type array.
int i – 파일 이름을 복사할 때 iterator 로 사용한다.

3.20 모듈이름: commandHead(char *tok, char com[])

3.20.1 기능

(실행방법) # head lines filename
getCommand 에서 command 가 'head'인 경우 호출된다
파일의 위에서부터 n 줄을 출력한다.
N 은 16 진수로 읽어 들이며, 범위는 [0x00000, 0xFFFFF] 이다.
N 이 파일 내의 줄 수보다 많으면, 파일 전체를 화면상에 출력한다.

역시 인자의 구분은 쉼표가 아니라 공백임에 주의해야 한다.

3.20.2 사용 변수

char *tok – command 이후 공백이 아닌 첫 글자를 가리키는 char type pointer.
char com[] – 입력된 문장 전체.
FILE *fp – 파일에 접근할 때 사용할 file pointer.
char filename[111] – 출력할 파일의 이름을 저장할 char type array.
int i – 파일 이름을 복사할 때 iterator 로 사용한다.
int c –파일에서 읽어와 화면에 출력할 때 사용할 임시 변수 저장소.
int line – 몇 줄을 출력할 것인지 저장할 변수.

3.21 모듈이름: commandEcho(char *tok, char com[])

3.21.1 기능

(실행방법) # echo [message]
getCommand 에서 command 가 'echo'인 경우 호출된다
사용자가 shell command prompt 상에서 입력한 문자열을 화면에 출력한다.
빈 문자열일 경우 빈 문자열이 출력된다.

3.21.2 사용 변수

char *tok – command 이후 공백이 아닌 첫 글자를 가리키는 char type pointer.
char com[] – 입력된 문장 전체.
char *it – command 이후 첫 번째 공백 다음 글자를 가리키는 char type pointer

3.22 모듈이름: loadInstruction()

3.22.1 기능

프로그램이 실행되면 opcode table 생성을 위해 실행되는 함수이다.
opcode.txt 에서 opcode, mnemonic, format 의 정보를 읽어와 struct Instruction 형태에 저장하고, 이를 opcode table 으로 저장한다.

3.22.2 사용 변수

FILE *fp – opcode.txt 에서 정보를 읽어오기 위한 file pointer.
char mnemonic[6], formats[4] – opcode.txt 에 기록된 mnemonic 과 format 을 저장하기 위한 문자열.
int opcode – opcode.txt 에 기록된 opcode 를 저장하기 위한 변수.
int idx – mnemonic 이 hash function 을 통해 얻은 인덱스를 저장하기 위한 변수.
Instruction *newNode, *it – 각각 새로운 node 를 할당하기위한 pointer, opcode table 생성을 위한 iterator.

4. 전역 변수 및 구조체, 매크로, typedef 정의

4.1 #define FALSE 0

FALSE 를 0 으로 정의한다.

4.2 #define TRUE 1

TRUE 를 1 으로 정의한다.

4.3 #define endString(x) (*(x) == EOF || *(x) == '\0')

인자로 넘겨받은 x 가 가리키는 글자가 문자열의 끝인지 확인하는 매크로이다.
문자열의 끝이라면 1 을, 아니라면 0 을 반환한다.
('\n'을 포함하지 않은 이유는, main()에서 '\n'을 '\0'으로 치환했기 때문)

4.4 #define indent(x) (*(x) == ' ' || *(x) == '\t')

인자로 넘겨받은 x 가 가리키는 글자가 공백인지 아닌지 확인하는 매크로이다.
공백이라면 1 을, 아니라면 0 을 반환한다.

4.5 typedef void (*comFuncPtr)(char *, char *)

위에 기술한 command…(char *tok, char com[]) 함수들을 function pointer 로 묶어 main()을 보다 깔끔하게 작성하기 위해 정의.

4.6 typedef struct _History

history 구현을 위해 linked list 구현이 필요했고, 이를 위해 선언한 structure 이다. Element 로는 command 를 저장하는 char type array 와 다음 node 를 연결할 link (struct _Histroy type pointer)가 있다.

4.7 typedef struct _Instruction

opcode mnemonic, opcodelist 를 구현하기 위해 hash table 구현이 필요했고, 이를 위해 선언한 structure 이다. Elemnet 로는 mnemonic 과 format 을 저장하는 char type array 와 다음 node 를 연결할 link (struct _Instruction type pointer)가 있다.

4.8 unsigned char virtualMem[1048576]

가상메모리를 위해 1MB (= 2^20)의 배열을 잡았다. 메모리 한 cell 가 표현할 수 있는 범위는 0x00 – 0xFF 이기 때문에 unsigned char 형 변수 하나로 표현될 수 있고, 이를 이용하여 2^20 개의 unsigned char 를 갖는 배열을 선언하였다. Simulator 에서 가상메모리로 사용된다.

4.9 History *historyHead, *last

history command 구현을 위해 linked list 가 필요했고, 이를 위해 선언하였다. historyHead 는 linked list 의 head 부분을 가리키며, last 는 tail 부분을 가리킨다. 프로그램 시작 시 NULL 로 초기화된다.

4.10 Instruction *instructionHead[20]

hash table 구현을 위해 선언하였다. 프로그램 시작 시 각 bucket 은 NULL 로 초기화된다.

4.11 int dumpLastAddr

dump command 구현 시 필요. dump command 수행 시 출력된 마지막 Address + 1 의 값을 저장하고 있다. (마지막 address 를 저장하고 있어도 되나 수행 시 1 을 증가시키는 것과 차이가 없어 다음 시작점을 가리키도록 하였음)

4.12 bool quitFlag

quit command 구현을 위해 선언하였다. quit command 가 정상적으로 입력되었을 시 1 로 설정되며, 이 경우 simulator 는 모든 memory 를 deallocating 한 뒤 종료된다. 0 으로 초기화 되어있다.

5. 코드

5.1 20151623.h

```
/*포함되는 파일*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<stdbool.h>
#include<dirent.h>
#include<sys/stat.h>

/*정의되는 상수 및 매크로*/
```

```
#define FALSE 0
#define TRUE 1
#define endString(x) (*(x) == EOF || *(x) == '\0')
#define indent(x) (*(x) == ' ' || *(x) == '\t')

/*함수 포인터를 위한 typedef*/
typedef void (*comFuncPtr)(char *, char *);

/*Linked list, Hash table 을 위한 구조체*/
typedef struct _History {
	char command[111];
	struct _History *link;
} History;	// structure for history

typedef struct _Instruction {
	char mnemonic[7], format[4];
	int opcode;
	struct _Instruction *link;
} Instruction;  // structure for instructions

/*변수*/
unsigned char virtualMem[1048576];	// for virtual memory, 1048576 = 16 ^ 5

History *historyHead = NULL, *last = NULL;	// empty linked list
Instruction *instructionHead[20] = {};  // empty hashtable
int dumpLastAddr = 0;   // last address of dump command
bool quitFlag = FALSE;  // when quitFlag stores 1 (TRUE), terminates this program

/*-----------------------------------------------------------------------------------------------*/
/*함수 : hashFunction                                                                            */
/*목적 :  문자열을 인자로 넘겨받고, 이에 해당하는 hash index 를 반환한다  */
/*리턴값 : hash table 에서의 index                                                               */
/*-----------------------------------------------------------------------------------------------*/
int hashFunction(char *val){
	int ret = 0;
	while(*val){
		ret *= 39;
		ret += *val - 'A';
		val++;
	}	// converts to hash
	//	let each character represents number which how far from 'A'
	//	and let that string is base39 digit
	return ret % 20;
}

/*-----------------------------------------------------------------------------------------------*/
/*함수 : getCommand                                                                              */
/*목적 :  입력된 문장에서 command 만을 tokenize 하고, 그 command 가       */
/*		유효한지를 판단, 유효하면 해당하는 commandNum 을, 유효하지   */
/*		않다면 commandNum 에 -1 을 저장한다.                                                    */
/*리턴값 : command 이후 공백이 아닌 첫 글자를 가리키는 char 형 포인터   */
/*-----------------------------------------------------------------------------------------------*/
```

```
char* getCommand(char str[], int *commandNum){
	char *com, tmp;

	while(!endString(str) && indent(str)) str++;
	// when this loop ends, str points the first character which is not indent or somethin

	for(com = str; !endString(str) && !indent(str); str++);
	tmp = *str, *str = '\0';
	// set the very first character after first word (command) as NULL, com now points only command string

	switch(*com){
		case 'c' : *commandNum = strcmp(com, "cat") ? strcmp(com, "cmp") ? strcmp(com, "copy") ? -1 : 12 :
11 : 10;
				break;
		case 'd' : *commandNum = strcmp(com, "d") * strcmp(com, "dir") ? strcmp(com, "du") * strcmp(com,
"dump") ? -1 : 4 : 1;
				break;
		case 'e' : *commandNum = strcmp(com, "e") * strcmp(com, "edit") ?  strcmp(com, "echo") ? -1 : 15 : 5;
				break;
		case 'f' : *commandNum = strcmp(com, "f") * strcmp(com, "fill") ? -1 : 6;
				break;
		case 'h' : *commandNum = strcmp(com, "h") * strcmp(com, "help") ? strcmp(com, "hi") * strcmp(com,
"history") ? strcmp(com, "head") ? -1 : 14 : 3 : 0;
				break;
		case 'o' : *commandNum = strcmp(com, "opcode") ? strcmp(com, "opcodelist") ? -1 : 9 : 8;
				break;
		case 'q' : *commandNum = strcmp(com, "q") * strcmp(com, "quit") ? -1 : 2;
				break;
		case 'r' : *commandNum = strcmp(com, "reset") ? -1 : 7;
				break;
		case 't' : *commandNum = strcmp(com, "touch") ? -1 : 13;
				break;
		default : *commandNum = -1;
				break;
	}	// switch for given command is vaild or not

	*str = tmp;
	// restore given line

	while(!endString(str) && indent(str)) str++;
	// when this loop ends, str points the first parameter or end of string

	return str;
}

/*-------------------------------------------------------------------------------*/
/*함수 : hexstrToInt                                                             */
/*목적 :  16 진수 문자열을 정수로 변환한다.                                      */
/*리턴값 : 에러가 없다면 변환된 정수, 에러가 있다면 아래와 같은 값  */
/*    -1 : 16 진수가 아님 / -2 : 0xFFFFF 를 넘어감 / -3 : -를 포함한 음수  */
/*-------------------------------------------------------------------------------*/
int hexstrToInt(char *str){     // for convert hex string to deciaml integer
```

```
    int ret = 0, len = 0;   // integer to store the result, length of str
    bool errFlag = FALSE, negFlag = FALSE;  // flag to check whether given string is headecimal or not, negative
or not

    if(*str == '-') negFlag = TRUE, str++;

    while(!endString(str) && !indent(str) && len < 6 && *str != ','){
        ret *= 16;
        if(*str < '0' || (*str > '9' && *str < 'A') || (*str > 'F' && *str < 'a') || *str > 'f'){
          errFlag = TRUE;
          break;
        }
        ret += *str <= '9' ? *str - '0' : *str <= 'Z' ? *str - 'A' + 10 : *str - 'a' + 10;
        str++; len++;
    }

    return len < 6 ? errFlag ? -1 : negFlag ? -3 : ret : -2;
    // -1 for invalid number, -2 for overflow ( over 0xFFFFF ), -3 for negative number
}

/*-------------------------------------------------------------------------------------*/
/*함수 : addHistory                                                                    */
/*목적 : 유효한 입력 값을 linked list 에 저장한다                                      */
/*리턴값 : 없음                                                                        */
/*-------------------------------------------------------------------------------------*/
void addHistory(char com[]){
    History *newNode;
    newNode = (History *)malloc(sizeof(History));
    strcpy(newNode->command, com);
    newNode->link = NULL;   // set new node with given command

    if(!historyHead) historyHead = newNode, last = newNode;
    else last->link = newNode, last = newNode;      // add new node to list's last node
}

/*-------------------------------------------------------------------------------------*/
/*함수 : commandHelp                                                                   */
/*목적 : shell 상에서 사용할 수 있는 명령어를 화면에 출력한다                          */
/*리턴값 : 없음                                                                        */
/*-------------------------------------------------------------------------------------*/
void commandHelp(char *tok, char com[]){
    if(!*tok){
        printf("h[elp]\nd[ir]\nq[uit]\nhi[story]\ndu[mp] [start, end]\ne[dit] address, value\nf[ill] start, end,
value\nreset\nopcode mnemonic\nopcodelist\ncat [filename(s)]\ncmp filename1 filename2\ncopy filename1
filename2\ntouch filename(s)\nhead lines filename\necho [message]\n");     // print the list
        addHistory(com);        // add command to list
    } else puts("Invalid command"); // if any character that is not indent followed by command, it's invalid
command
    return;
}

/*-------------------------------------------------------------------------------------*/
```

```
/*함수 : commandDir                                                              */
/*목적 : 실행파일이 위치한 directory 내의 item 을 화면에 출력한다     */
/*          directory 뒤에는 '/'이, 실행파일 뒤에는 '*'이 붙는다.          */
/*리턴값 : 없음                                                                  */
/*--------------------------------------------------------------------------------*/
void commandDir(char *tok, char com[]){
      DIR *currDir;
      struct dirent *currFile;
      struct stat currStat;

      if(!*tok){
            if((currDir = opendir(".")){   // for read current directory. When opendir returns NULL, it means error
occured.
                  while((currFile = readdir(currDir))){   // when readdir returns NULL, that means it reaches to the end
of directory
                        stat(currFile->d_name, &currStat);
                        printf("\t%s", currFile->d_name);
                        if(S_ISDIR(currStat.st_mode)) putchar('/');     // when this item is directory
                        else if(S_IXUSR & currStat.st_mode) putchar('*');       // when this item is executable
                  }
                  closedir(currDir);
                  puts(""); // prints new line
            }
            addHistory(com);        // add command to list
      } else puts("Invalid command"); // if any character that is not indent followed by command, it's invalid
command
}

/*--------------------------------------------------------------------------------*/
/*함수 : commandQuit                                                             */
/*목적 : shell 을 종료한다. quitFlag 를 1 로 설정, main 함수에서 loop 를 */
/*          종료시킨다.                                                          */
/*리턴값 : 없음                                                                  */
/*--------------------------------------------------------------------------------*/
void commandQuit(char *tok, char com[]){
      if(!*tok) quitFlag = TRUE;
      else puts("Invalid command"); // if any character that is not indent followed by command, it's invalid command
      return;
}

/*--------------------------------------------------------------------------------*/
/*함수 : commandHistory                                                          */
/*목적 : 현재까지 입력된 유효한 command 를 시간순서대로 출력한다  */
/*리턴값 : 없음                                                                  */
/*--------------------------------------------------------------------------------*/
void commandHistory(char *tok, char com[]){
      int i = 0;      // for history index
      History *curr;  // for loop

      if(!*tok){
            addHistory(com);        // add command to list
            curr = historyHead;
```

```
            while(curr){    // while the history remains
                printf("%-5d %s\n", ++i, curr->command);        // print the history
                curr = curr->link;      // jump to next node
            }
        } else puts("Invalid command"); // if any character that is not indent followed by command, it's invalid
command
        return;
}

/*-------------------------------------------------------------------------------------------*/
/*함수 : commandDump                                                                         */
/*목적 : 가상메모리 상의 값을 화면상에 출력한다. 형식은 아래와 같다                          */
/*          0x10 단위 주소 - 16 개의 값 – 값에 해당하는 문자                                */
/*          start, end 가 주어지지 않을 시 160 개의 값을 출력하는 것이                      */
/*          default 로 설정되어있다.                                                        */
/*리턴값 : 없음                                                                             */
/*-------------------------------------------------------------------------------------------*/
void commandDump(char *tok, char com[]){
        int i, j, start, end; // loop variables, integer which stores start and end address each
        start = dumpLastAddr; end = 0;

        if(*tok){       // when parameters have entered after 'du' or 'dump'
            start = hexstrToInt(tok);

            if(start < 0 || *tok == ','){   // when error occurs
                if(start == -1 || *tok == ',') puts("Invalid start address has entered");
                else if(start == -2) puts("Please enter the address between 0x00000 through 0xFFFFF");
                else puts("Negative number has entered");
                return;
            }

            while(!endString(tok) && !indent(tok) && *tok != ',') tok++;
            while(!endString(tok) && indent(tok)) tok++;
            // now tok points the first character that is not indent right after first parameter, or end of string
            if(*tok == ','){        // when end exists
                tok++;  // for point character after ','
                while(!endString(tok) && indent(tok)) tok++;
                // now tok points the first character of second parameter or end of string

                if(!*tok){      // when comma has entered but no parameter followed by
                    puts("After comma need to enter end address");
                    return;
                }

                end = hexstrToInt(tok);

                if(end < 0){    // when error occurs
                    if(end == -1) puts("Invalid end address has entered");
                    else if(end == -2) puts("Please enter the address between 0x00000 through 0xFFFFF");
                    else puts("Negative number has entered");
                    return;
                }
```

```
            if(start > end){        // when start is bigger than end
                puts("Start address value is bigger than end address value");
                return;
            }
        } else if(*tok){
            puts("Need to use comma to classify two addresses");    // when two parameters have entered without
comma
            return;
        } else dumpLastAddr = start;    // when only start exists
    }

    end = end ? end : start + 159 > 0xFFFFF ? 0xFFFFF : start + 159;        // set end address

    for(i = start / 16 * 16; i <= end / 16 * 16; i += 16){
        printf("%05x ", i);
        for(j = i; j < i + 16; j++){
            j >= start && j <= end ? printf("%02x ", virtualMem[j]) : printf("   ");
        }
        printf("; ");
        for(j = i; j < i + 16; j++){
            j >= start && j <= end && virtualMem[j] >= 0x20 && virtualMem[j] <= 0x7E ?
putchar(virtualMem[j]) : putchar('.');
        }
        puts("");
    }     // print

    dumpLastAddr = end + 1;

    addHistory(com);        // add command to list
}

/*-------------------------------------------------------------------------------------*/
/*함수 : commandEdit                                                                   */
/*목적 : 메모리에서 입력된 Address 의 값을 주어진 value 로 설정한다 */
/*리턴값 : 없음                                                                        */
/*-------------------------------------------------------------------------------------*/
void commandEdit(char *tok, char com[]){
    int addr, val;  // stores address, value each
    if(!*tok){      // when only 'e' or 'edit' were given
        puts("Parameters required : no parameters have entered");
        return;
    } else{
        addr = hexstrToInt(tok);

        if(addr < 0 || *tok == ','){    // when error occurs
            if(addr == -1 || *tok == ',') puts("Invalid address has entered");
            else if(addr == -2) puts("Please enter the address between 0x00000 through 0xFFFFF");
            else puts("Negative number has entered");
            return;
        }
```

```
        while(!endString(tok) && !indent(tok) && *tok != ',') tok++;
        while(!endString(tok) && indent(tok)) tok++;
        // now tok points the first character that is not indent right after first parameter, or end of string
        if(*tok == ','){        // when value exists
            tok++;  // for point character after ','
            while(!endString(tok) && indent(tok)) tok++;
            // now tok points the first character of second parameter or end of string

            if(!*tok){      // when comma has entered but no parameter followed by
                puts("After comma need to enter value");
                return;
            }

            val = hexstrToInt(tok);

            if(val < 0 || val > 0xFF){      // when error occurs
                if(val == -1) puts("Invalid value has entered");
                else if(val == -2 || val > 0xFF) puts("Please enter the value between 0x00 through 0xFF");
                else puts("Negative number has entered");
                return;
            }
        } else if(*tok){
            puts("Need to use comma to classify two parameters");   // when two parameters have entered
without comma
            return;
        } else{ // when only address exists
            puts("Please enter the value");
            return;
        }
    }
    virtualMem[addr] = val;
    addHistory(com);        // add command to list
}

/*-------------------------------------------------------------------------------------*/
/*함수 : commandFill                                                                   */
/*목적 : 메모리 위 [start, end]에 해당하는 값을 value 로 설정한다                      */
/*리턴값 : 없음                                                                       */
/*-------------------------------------------------------------------------------------*/
void commandFill(char *tok, char com[]){
    int i, start, end, val; // loop variable, stores start, end address and value each

    if(!*tok){      // when only 'f' or 'fill' were given
        puts("Parameters required : no parameters have entered");
        return;
    } else{
        start = hexstrToInt(tok);

        if(start < 0 || *tok == ','){   // when error occurs
            if(start == -1 || *tok == ',') puts("Invalid address has entered");
            else if(start == -2) puts("Please enter the address between 0x00000 through 0xFFFFF");
            else puts("Negative number has entered");
```

```
        return;
}

while(!endString(tok) && !indent(tok) && *tok != ',') tok++;
while(!endString(tok) && indent(tok)) tok++;
// now tok points the first character that is not indent right after first parameter, or end of string

if(*tok == ','){        // when end address exists
        tok++;  // for point character after ','
        while(!endString(tok) && indent(tok)) tok++;
        // now tok points the first character of second parameter or end of string

        if(!*tok){      // when comma has entered but no parameter followed by
                puts("After comma need to enter end address");
                return;
        }

        end = hexstrToInt(tok);

        if(end < 0){    // when error occurs
                if(end == -1) puts("Invalid value has entered");
                else if(end == -2) puts("Please enter the value between 0x00 through 0xFF");
                else puts("Negative number has entered");
                return;
        }

        if(start > end){        // when start is bigger than end
                puts("Start address value is bigger than end address value");
                return;
        }

        while(!endString(tok) && !indent(tok) && *tok != ',') tok++;
        while(!endString(tok) && indent(tok)) tok++;
        // now tok points the first character that is not indent right after second parameter, or end of string

        if(*tok == ','){        // when value exists
                tok++;
                while(!endString(tok) && indent(tok)) tok++;
                // now tok points the first character of third parameter or end of string

                if(!*tok){      // when comma has entered but no parameter followed by
                        puts("After comma need to enter value");
                        return;
                }

                val = hexstrToInt(tok);

                if(val < 0 || val > 0xFF){      // when error occurs
                        if(val == -1) puts("Invalid value has entered");
                        else if(val == -2 || val > 0xFF) puts("Please enter the value between 0x00 through 0xFF");
                        else puts("Negative number has entered");
                        return;
                }
```

```
                } else if(*tok){
                    puts("Need to use comma to classify end address and value");
                    return;
                } else{
                    puts("Please enter the value");
                    return;
                }

            } else if(*tok){
                puts("Need to use comma to classify three parameters"); // when two parameters have entered
without comma
                return;
            } else{ // when only start address exists
                puts("Please enter start address and value");
                return;
            }
        }
        for(i = start; i <= end; i++) virtualMem[i] = val;      // fill
        addHistory(com);        // add command to list
}

/*-------------------------------------------------------------------------------------*/
/*함수 : commandReset                                                                  */
/*목적 : 메모리상의 모든 값을 0 으로 초기화한다                                        */
/*리턴값 : 없음                                                                        */
/*-------------------------------------------------------------------------------------*/
void commandReset(char *tok, char com[]){
        if(!*tok){
            memset(virtualMem, 0, sizeof(virtualMem));      // set every element in virtual memory 0
            addHistory(com);        // add command to list
        } else puts("Invalid command"); // if any character that is not indent followed by command, it's invalid
command
        return;
}

/*-------------------------------------------------------------------------------------*/
/*함수 : commandMnemonic                                                               */
/*목적 : 입력된 mnemonic 에 해당하는 opcode 를 출력한다                                */
/*리턴값 : 없음                                                                        */
/*-------------------------------------------------------------------------------------*/
void commandMnemonic(char *tok, char com[]){
        int idx;        // stores index for given parameter
        char *it;       // iterator
        Instruction *curr;      // iterator
        if(!*tok) puts("No mnemonic has entered");
        else{
            for(it = tok; !endString(it) && !indent(it); it++);
            *it = '\0';     // tokenize given mnemonic
            idx = hashFunction(tok);
            curr = instructionHead[idx];
            while(curr && strcmp(curr->mnemonic, tok)) curr = curr->link;
```

```
            // if given mnemonic exists in table, curr must point some node.
            if(!curr) puts("Invalid mnemonic");
            else{
                printf("opcode is %x\n", curr->opcode);
                addHistory(com);        // add command to list
            }
        }
}

/*-----------------------------------------------------------------------------------------*/
/*함수 : commandOplist                                                                     */
/*목적 : Opcode table 에 존재하는 instruction 을 전부 출력한다                             */
/*리턴값 : 없음                                                                            */
/*-----------------------------------------------------------------------------------------*/
void commandOplist(char *tok, char com[]){
    int i;  // loop variable
    Instruction *it;        // iterator
    if(!*tok){
        for(i = 0; i < 20; i+=puts("")){
            printf("%3d : ", i);
            it = instructionHead[i];
            while(it){
                printf("[%s,%x]", it->mnemonic, it->opcode);
                it = it->link;
                if(it) printf(" -> ");
            }
        }       // print the hash table
        addHistory(com);        // add command to list
    } else puts("Invalid command"); // if any character that is not indent followed by command, it's invalid
command
}

/*-----------------------------------------------------------------------------------------*/
/*함수 : commandCat                                                                        */
/*목적 : 입력된 파일의 내용을 출력한다.                                                    */
/*          주어진 인자 중 유효한 인자만 출력되며, 인자가 주어지지      */
/*         않은 경우, 표준 입력을 입력으로 본다.                                          */
/*리턴값 : 없음                                                                            */
/*-----------------------------------------------------------------------------------------*/
void commandCat(char *tok, char com[]){
    FILE *fp;
    char fileName[111];
    int i, c, catCounter = 0;       // iterator, temporary character storage, counts successfully done file
    if(!*tok) while(~(c = getchar())) putchar(c);
    // when no file name has entered, cat command uses stdin
    else{   // when file name has entered
        do{
            i = 0;
            while(!endString(tok) && !indent(tok)) fileName[i++] = *tok++;
            fileName[i] = '\0';
            // copy file name
            if((fp = fopen(fileName, "r"))){        // when file exists
```

```
                while(~(c = fgetc(fp))) putchar(c);
                fclose(fp);
                ++catCounter;
            } else printf("cat: %s: no such file\n", fileName);
            // when file does not exists
            while(!endString(tok) && indent(tok)) tok++;
            // now tok points next file name's first character or end of string
        } while(*tok);
    }
    if(catCounter) addHistory(com); // if there is any of given file done successfully, add command to list
}

/*-------------------------------------------------------------------------------------*/
/*함수 : commandCmp                                                                    */
/*목적 : 주어진 두 파일을 비교한다                                                     */
/*          다른 경우 몇 번째 라인 몇 번째 바이트에서 다른지 출력한다 */
/*          기준은 첫 번째로 주어진 파일이다                                           */
/*리턴값 : 없음                                                                       */
/*-------------------------------------------------------------------------------------*/
void commandCmp(char *tok, char com[]){
    FILE *sfp, *dfp;
    char sourceName[111], destName[111];
    int i = 0, line, s, d;  // iterator, stores line number, temporary storage
    bool diffFlag = 0;      // flag that shows whether two files are different or not
    if(!*tok){      // when no file name has entered
        puts("File name required");
        return;
    } else{
        while(!endString(tok) && !indent(tok)) sourceName[i++] = *tok++;
        sourceName[i] = '\0';
        // copy first file name

        while(!endString(tok) && indent(tok)) tok++;
        // now tok points second file name's first character of end of string

        if(!*tok){      // when there are no second file name
            puts("Second file name required");
            return;
        }

        i = 0;
        while(!endString(tok) && !indent(tok)) destName[i++] = *tok++;
        destName[i] = '\0';
        // copy second file name

        if((sfp = fopen(sourceName, "r")) && (dfp = fopen(destName, "r"))){     // when both file exists
            s = d = 0;      // initiallize storage
            for(i = line = 1; ~s && ~d; i++){
                s = fgetc(sfp); d = fgetc(dfp);
                if(s ^ d){      // when diffrence have found
                    diffFlag = 1;
                    break;
```

```
                }
                if(s == '\n') ++line, i = 0;    // when line changes
            }
            if(diffFlag) printf("%s %s differ : byte %d, Line %d\n", sourceName, destName, i, line);
        } else{ // when some of files does not exist
            puts("Invalid file name");
            return;
        }
    }
    addHistory(com);        // add command to list
}

/*-------------------------------------------------------------------------------------*/
/*함수 : commandCopy                                                                   */
/*목적 : 파일을 두 번째 인자를 이름으로 하는 파일에 복사한다                          */
/*리턴값 : 없음                                                                       */
/*-------------------------------------------------------------------------------------*/
void commandCopy(char *tok, char com[]){
    FILE *sfp, *dfp;
    char sourceName[111], destName[111];
    int i = 0, c;   // iterator, temporary storage
    if(!*tok){      // when no file name has entered
        puts("File name required");
        return;
    } else{
        while(!endString(tok) && !indent(tok)) sourceName[i++] = *tok++;
        sourceName[i] = '\0';
        // copy source file name

        while(!endString(tok) && indent(tok)) tok++;
        // now tok points destiation file name's first character or end of string

        if(!*tok){      // when there are no destination file name
            puts("Destination file name required");
            return;
        }

        i = 0;
        while(!endString(tok) && !indent(tok)) destName[i++] = *tok++;
        destName[i] = '\0';
        // copy destination file name

        if((sfp = fopen(sourceName, "r"))){     // when source file exists
            if(!(dfp = fopen(destName, "w"))){      // when failed to open destination file
                puts("Failed to make / access file");
                return;
            }
            while(~(c = fgetc(sfp))){
              if(fputc(c, dfp) == EOF){
                    puts("File writing error occured");
                    return;
              }
```

```
                }       // copy
                fclose(sfp); fclose(dfp);
            } else{ // when file does not exist
                printf("copy: %s: no such file\n", sourceName);
                return;
            }
        }
        addHistory(com);        // add command to list
}

/*-------------------------------------------------------------------------------------*/
/*함수 : commandTouch                                                                  */
/*목적 : 주어진 문자열을 이름으로 하는 크기가 0 바이트인                               */
/*          새로운 파일을 생성한다                                                     */
/*리턴값 : 없음                                                                        */
/*-------------------------------------------------------------------------------------*/
void commandTouch(char *tok, char com[]){
        FILE *fp;
        char fileName[111];
        int i = 0;      // iterator
        if(!*tok){
                puts("File name required");
                return;
        } else{
                do{
                        i = 0;
                        while(!endString(tok) && !indent(tok)) fileName[i++] = *tok++;
                        fileName[i] = '\0';
                        // copy given file name
                        if((fp = fopen(fileName, "a"))){
                                fclose(fp);
                                // create 0-byte new file or touch already created file
                        } else puts("File create / access error");
                        while(!endString(tok) && indent(tok)) tok++;
                        // now tok points next file name's first character or end of string
                } while(*tok);
        }
        addHistory(com);
}

/*-------------------------------------------------------------------------------------*/
/*함수 : commandHead                                                                   */
/*목적 : 주어진 숫자 (16 진수)만큼 주어진 파일의 위로부터 출력한다 */
/*          숫자가 n 이면, n 줄을 출력한다.                                            */
/*리턴값 : 없음                                                                        */
/*-------------------------------------------------------------------------------------*/
void commandHead(char *tok, char com[]){
        FILE *fp;
        char fileName[111];
        int line, i = 0, c;     // for store given line, iterator
        if(!*tok){      // when no parameters are entered
                puts("No parameters had entered");
```

```
            return;
        } else{
            line = hexstrToInt(tok);
            if(line < 0){   // given number is not hex
                puts("Please enter correct hexadecimal");
                return;
            }

            while(!endString(tok) && !indent(tok)) tok++;
            while(!endString(tok) && indent(tok)) tok++;
            // now tok points the first character of file name or end of string

            if(!*tok){      // when there are no more parameter left
                puts("Please enter file name");
                return;
            }

            while(!endString(tok) && !indent(tok)) fileName[i++] = *tok++;
            fileName[i] = '\0';
            // copy file name

            if((fp = fopen(fileName, "r"))){
                while(~(c = fgetc(fp)) && line){
                    putchar(c);
                    if(c == '\n') --line;
                }
                fclose(fp);
            } else{
                printf("head: %s: no such file\n", fileName);
                return;
            }
        }
        addHistory(com);
}

/*-------------------------------------------------------------------------------------*/
/*함수 : commandEcho                                                                   */
/*목적 : 유저에게 입력 받은 문자열을 화면상에 출력한다                                */
/*리턴값 : 없음                                                                        */
/*-------------------------------------------------------------------------------------*/
void commandEcho(char *tok, char com[]){
        char *it;
        it = com;
        while(!endString(it) && indent(it)) it++;
        while(!endString(it) && !indent(it)) it++;
        // now it points right after the command echo

        if(*it) it++;
        printf("%s\n", it);
        addHistory(com);
}
```

```
/*-------------------------------------------------------------------------------------*/
/*함수 : loadInstruction                                                               */
/*목적 : 프로그램 실행 시, 'opcode.txt'에서 정보를 읽어와                              */
/*        이를 토대로 opcode table 를 구축한다.                                        */
/*리턴값 : 없음                                                                        */
/*-------------------------------------------------------------------------------------*/
void loadInstruction(){
    FILE *fp;      // for reading opcode.txt file
    char mnemonic[6], formats[4];   // for store mnemonic and given formats
    int opcode, idx;       // for store opcode, index
    Instruction *newNode, *it;      // for allocate new nodes, and iterator
    fp = fopen("opcode.txt", "r");
    while(~fscanf(fp, "%x %s %s", &opcode, mnemonic, formats)){     // until file pointer reaches EOF
        newNode = (Instruction *)malloc(sizeof(Instruction));
        newNode->opcode = opcode;
        strcpy(newNode->mnemonic, mnemonic);
        strcpy(newNode->format, formats);
        newNode->link = NULL;   // set new node with inputs from file
        idx = hashFunction(mnemonic);   // get hash through hash function
        if(instructionHead[idx]){
            it = instructionHead[idx];
            while(it->link) it = it->link;
            it->link = newNode;
        } else instructionHead[idx] = newNode;  // link to table
    }
    fclose(fp);
    return;
}
```

5.2 20151623.c

```
/*포함되는 파일*/
#include"20151623.h"

/*프로그램 시작*/
int main(){
    int commandNum = 0, i;  // to save the command
    char inputLine[111], *tok;      // inputLine for get input from user, tok for tokenizing
    History *curr;  // for deallocating the list
    Instruction *it, *prev; // for deallocating the hashtable
     comFuncPtr comFunc[] = { commandHelp, commandDir,
        commandQuit, commandHistory, commandDump, commandEdit,
        commandFill, commandReset, commandMnemonic, commandOplist,
        commandCat, commandCmp, commandCopy, commandTouch,
        commandHead, commandEcho };
    // function pointer to reduce code cluster
    loadInstruction();
    // load instructions
    do{
        printf("sicsim>");      // shell
        fgets(inputLine, sizeof(inputLine), stdin);     // get input in inputLine
        inputLine[strlen(inputLine)-1] = '\0';  // set the last character of given line as NULL
        tok = getCommand(inputLine, &commandNum);       // tokenizing given line
```

```
        // now tok points first character of parameter, of end of string
        if(commandNum < 0) puts("Invaild command"); // if commandNum stores negative value, it means that
given command is invaild
        else comFunc[commandNum](tok, inputLine);       // else run function which mathces to given command
    }while(!quitFlag);

    while(historyHead){
        curr = historyHead;
        historyHead = historyHead->link;
        free(curr);
    }     // deallocating the list

    for(i = 0; i < 20; i++){
        it = instructionHead[i];
        while(it){
            prev = it;
            it = it->link;
            free(prev);
        }
    }     // deallocating the table

    return 0;
}
```