과목 명: 시스템프로그래밍

담당 교수 명: 박 운 상

<<Project 2>>

**서강대학교 컴퓨터학과**

**[학번] 20161565**

**[이름] 권기윤**

목 차

1. 프로그램 개요 3

2. 프로그램 설명 3

2.1 프로그램 흐름도

2.1 proj2 상세 흐름도

3. 모듈 정의

3.1 20161565.c 내부 추가함수

3.1.1 int main();

3.1.2 void init();

3.2 shell\_command.c 내부 추가 함수

3.2.1 int cmd\_help(int tok\_num);

3.2.2 int cmd\_typeFilename(char input[100][100]);

3.3 assembly\_command.c 내부함수

3.3.1 int pass1(FILE\* asm\_file);

3.3.2 int pass2();

3.3.3 int cmd\_assembleFilename(char input[100][100], int tok\_num);

3.3.4 int makeLstFile(FILE\* lstfile);

3.3.5 int makeObjFile(FILE\* Objfile);

3.3.6 int isdotAsm(char\* filename);

3.3.7 int cmd\_symbol(char input[100][100], int tok\_num);

3.3.8 int add\_symTab(char asm\_tok[100][100], int tok\_num, int LOCCTR);

3.3.9 int find\_register(char\* reg);

3.3.10 Opcode\_table\* find\_OpcodeNode(char\* mnemonic)

3.3.11 free\_symtab(symbolNode\* renove);

4. 전역 변수 정의 4

4.1 구조체 정의

4. 1. 1 추가 전역 구조체 정의

4.1.1.2 \_LstNode

4.1.1.3 \_symtab

5. 코드 설명

5.1 20161565.h

5.2 2016156.c

5.3 input\_handling.c

5.4 shell\_command.c

5.5 memory\_command.c

5.6 opcode\_command.c

5.7 assembly\_command.c

# 프로그램 개요

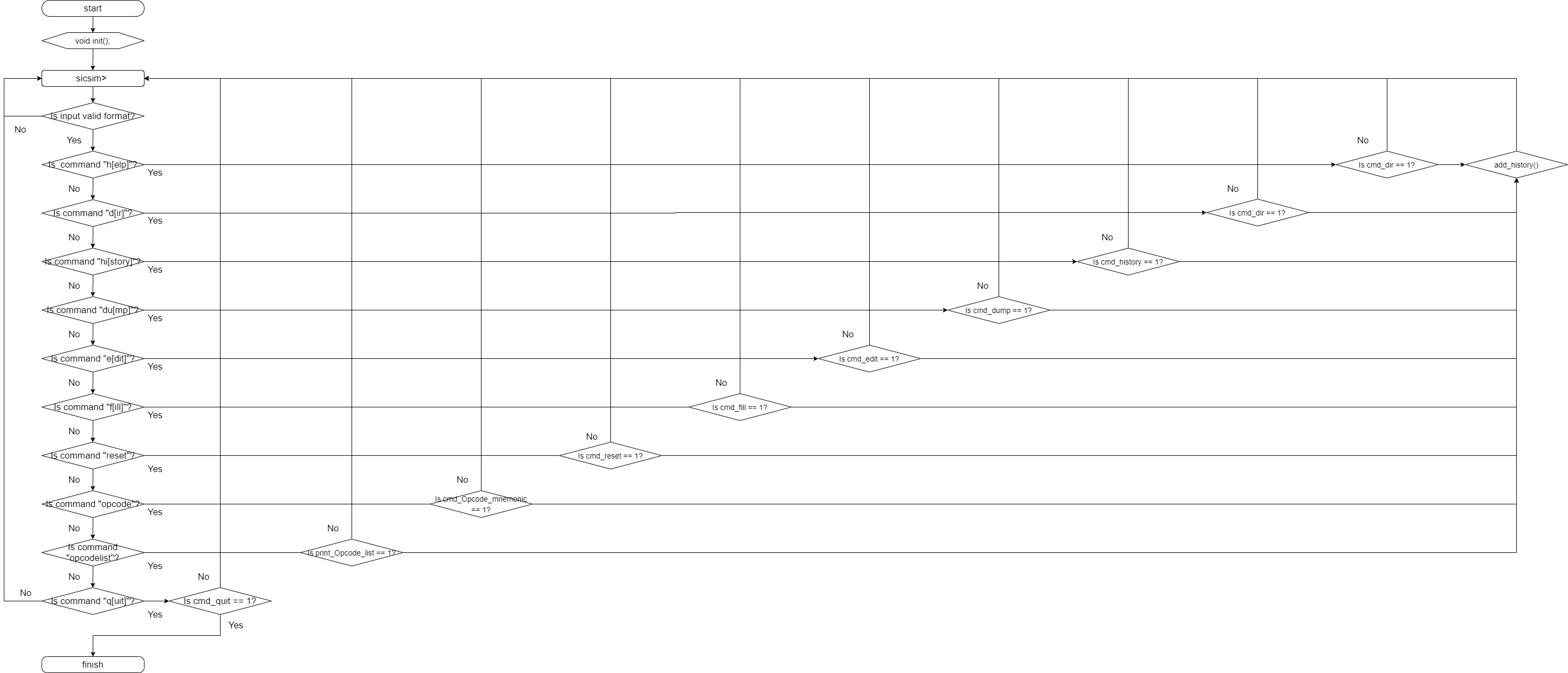
이 프로그램은 앞으로 구현하게 될 SIC/XE 머신을 위한 전 단계로 어셈블러, 링크, 로더를 실행할 쉘과 컴파일을 통해 만들어진 object 코드가 저장되고 실행될 메모리공간을 구현하였습니다. 또한 ADD, COMP와 같은 mnemonic 을 opcode 값으로 변화하여 hashTable에 저장하고 일부 명령어 (help, dir, quit, history, dump, edit, fill, reset, opcode, opcodelist)를 구현하는 프로그램입니다.

프로젝트 #1에서 구현한 셀(shell)에 assemble 기능을 추가하는 프로그램. SIC/XE의 assembly program source 파일을 입력 받아서 object파일을 생성하고, 어셈블리 과정 중 생 성된 symbol table과 결과물인 object 파일을 볼 수 있는 기능을 제공해야 함. 교재의 2.2까 지 설명된 SIC/XE 어셈블러의 기능을 구현함을 원칙으로 한다.

# 프로그램 설명

## 프로그램 흐름도

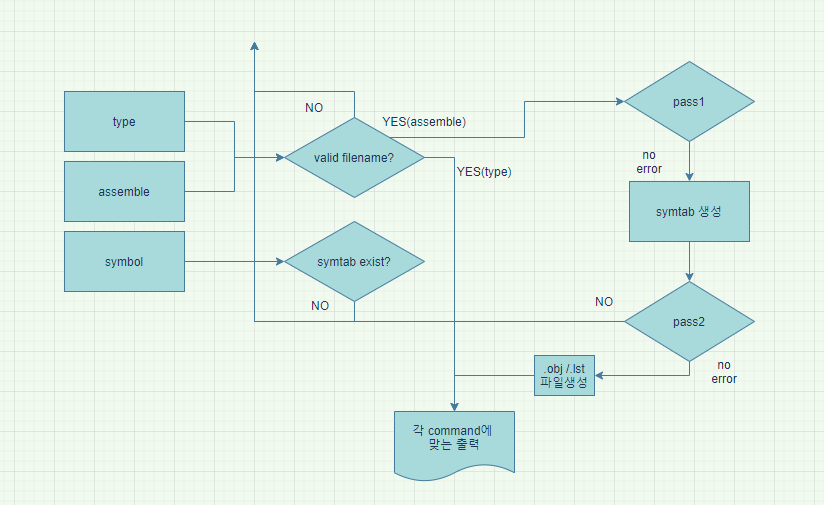
그림 1> 프로그램 흐름도



프로그램이 시작되면 void init(); 함수로 전역 구조체 동적할당, opcode hash tree 를 생성한다. 그 후 한 줄을 input\_str 에 입력받는다. 만약 올바른 형식으로 input\_str이 들어왔다면, input\_str 에서 tokenize한 명령어를 판별한다. 올바른 형식의 명령어와 명령어에 해당하는 올바른 형식이 들어온 것이 확인되면 add\_history 함수를 호출하여 input\_Str을 기록한다. 기록 후에는 다시 “sicsim> ”으로 돌아와 입력을 받으며, “q[uit]” 명령어가 입력되면 프로그램을 종료한다.

## Proj2 흐름도

그림 2> 프로그램 흐름도



# 모듈 정의

## 20161565.c 내부 추가함수

### void init();

proj2 에서 필요한 assembly command.c 에서 필요한 각종 전역변수들을 초기화한다.

### Int main();

Symbol, assemble, type 에 대한 명령어를 처리하는 코드를 추가했다.

## Shell\_command.c 내부함수

### int cmd\_help(int tok\_num);

h[elp] 명령어에 Symbol, assemble, type 을 추가했다

### int cmd\_typeFilename(char input[100][100]);

type 명령어를 처리한다. Type filename 에서 filename 이 현재 디렉토리에 있다면 해당 파일의 내용을 출력하고, 아니면 에러 메시지를 출력한다.

## assembly\_command.c 내부함수

### int pass1(FILE\* asm\_file)

pass1 에서 수행해야 하는 기능을 수행한다. 파일을 읽어 tokenize 한다.

주석과 빈문장을 처리하고, label 이 있다면 label에 해당하는 locctr 을 symtab에 추가한다. Label이 중복이라면 에러메시지를 출력한다. 그 후에 START,BASE,WORD,BYTE,RESW,RESB,END 와 같은 opcode에 없는 명령어를 처리한 뒤, opcode 에 따른 locctr과 label 유무를 lstArr 에 모두 저장한다.

### Int pass2();

Pass1에서 처리한 lstArr을 다시 읽어 pass2를 수행한다. 먼저 인자로 @이나 #이 입력되는지 확인하여, n, i를 지정해준다. 이후 mnemonic의 format에 따라서 x, b, p, e를 확인한다. Format3의 경우 만약 pc relative로 address를 나타낼 수 없다면 base relative로 되는지 확인하고 안 된다면 error를 return 한다. 각 경우에 맞게 b, p를 지정한다. 또한 X register를 사용하는지 확인하고 x를 지정한다. 그리고 extended mode이면 e를 1로 지정하고 addr을 저장한다. 이후 각 format에 맞게 object code를 계산하여 저장한다

### int cmd\_assembleFilename(char input[100][100], int tok\_num);

assemble filename 명령어를 처리한다. Filename을 읽어 적합한 파일인지 확인하고, 적합하다면 변수 초기화와 pass1 과 pass2를 실행한다. Pass1,2 실행 전에 symtab 을 tmp\_symtab에 임시로 저장하고, pass 1,2에서 error를 반환하고 tmp\_symtab 을 symtab 에 덮어 씌우고, 정상적으로 실행되었다면 tmp\_symtab을 초기화하고 filename.lst filename.obj를 생성한 후에 파일 내용을 채운다.

### int makeLstFile(FILE\* lstfile);

filename.lst에 해당하는 내용을 채운다. Line num, loc counter, line string, objcode 순으로 한줄 씩 파일출력한다.

### int makeObjFile(FILE\* Objfile);

filename.obj에 해당하는 내용을 채운다.

### int isdotAsm(char\* filename);

파일 확장자가 .asm 인지 확인한다.

### int cmd\_symbol(char input[100][100], int tok\_num);

symbol 명령어를 처리한다. Symbol table이 없다면 에러메시지를 출력하고, 아니라면 symtab 안의 내용을 순서대로 출력한다.

### int add\_symTab(char asm\_tok[100][100], int tok\_num, int LOCCTR);

pass 1에서 label 과 loc counter에 대한 정보를 가지고 symtab 에 추가한다. 추가할 때, label의 오름차순으로 출력한다.

### int find\_register(char\* reg);

sic/xe 에 해당하는 register인지 확인하고, 있다면 number를 없다면 -1을 return 한다.

### Opcode\_table\* find\_OpcodeNode(char\* mnemonic);

Proj1 에서 생성한 opcode hash table 에서 mnemonic 에 해당하는 opcode node를 return한다.

### free\_symtab(symbolNode\* remove);

symtab을 순차적으로 free한다.

# 전역 변수 정의

## 구조체 정의

### Struct \_LstNode

typedef struct \_lstNode{

int Loc\_counter;

* location counter를 저장한다.

int label\_flag;

* label 이 있는지 확인한다.

long long Objcode;

* Objcode 를 저장한다.

int printObjnum;

* Objcode를 출력할 format을 입력한다.

char line[100];

* filename 에서 읽어온 string을 저장한다.

char str[100];

* BTYE x ‘hex’ 에서 hex를 string 으로 저장한다.

}lstNode;

### Struct \_symtab

typedef struct \_symtab{

int Loc\_counter;

* label 의 location counter를 저장한다.

char Label[50];

* label을 저장한다.

struct \_symtab\* link;

* 다음 link를 지정하는 node

}symbolNode;

# 코드

## 20161565.h

//20161565 header file

#include <stdio.h>

#include <string.h>

#include <dirent.h>

#include <stdlib.h>

#include <sys/stat.h>

//라이브러리

#define max(a,b) ({ \_\_typeof\_\_ (a) \_a = (a); \_\_typeof\_\_ (b) \_b = (b); \_a > \_b ? \_a : \_b; }) // 타입 상관없이 a, b 중 큰값을 반환하는 매크로

#define min(a,b) ({ \_\_typeof\_\_ (a) \_a = (a); \_\_typeof\_\_ (b) \_b = (b); \_a < \_b ? \_a : \_b; })  // 타입 상관없이 a, b 중 작은 값을 반환하는 매크로

#define MAX\_MEMORY 1048576

// 매크로 함수

typedef struct history{// valid input에 대한  history 를 저장하는 struct

    int num;

    char command[100];

    struct history\* link;

} HISTORY;

typedef struct Opcode\_hash\_table{// "opcode.txt" 의opcode, mnemonic, format을 저장

    char key[10];

    char opcode\_format[10];

    int value;

    struct Opcode\_hash\_table\* link;

} Opcode\_table;

typedef struct \_lstNode{// struct that store information for pass1

    int Loc\_counter;

    int label\_flag;

    long long Objcode;

    int printObjnum;

    char line[100];

    char str[100];

}lstNode;

typedef struct \_symtab{// struct that store symtab

    int Loc\_counter;

    char Label[50];

    struct \_symtab\* link;

}symbolNode;

//전역 변수

int MEMORY\_IDX; // dump command를 수행할때 사용하는 unsigned char\* memory의 address

int st\_flag, Starting\_address; //st\_flag check .asm has"START", Starting\_address = start loc counter.

char programName[100]; // store program name

int pLength; // store program length

int BASE\_line\_idx, BASE\_flag; // store "BASE" line index and check if base

char BASE\_name[100]; //store base name;

int End\_flag; // check if end with "END"

int lst\_idx; // file.asm line idx

int Modify\_idx; // idx for modify

int modify[100]; // array for .obj modify

// 전역 linked list

Opcode\_table\*\* opTable; //opcode 를 저장하는 linked list

HISTORY\* history\_head;  // history를 저장하는 linked list

// 전역 struct array

lstNode lstArr[2000]; // struct array for pass1

symbolNode\* symtab, \*tmp\_symtab; //linked list struct for store symtab

unsigned char\* memory; // 1MByte(16\*65536)인 가상의 메모리 공간

//main

void init(); // Shell 환경을 구축하기 위해 전역변수의 동적할당, opcode\_table을 생성 한다.

//input handling

void parse\_input(char\* input ,char parsed\_tok[100][100], int\* tok\_num); // 들어온 input을 tokenize한다.

int isvalid\_char(char c);   // 해당하는 character가 valid 한지 확인한다.

int isvalid\_input(char\* input); //들어온 input string이 valid 한 형태를 띄고 있는지 확인한다.

char\* trim\_str(char\* input);    //input string의 앞, 뒤 공백을 삭제한다.

//shell command

int cmd\_help(int tok\_num);  // h[elp] 명령어를 처리한다.

int cmd\_dir(int tok\_num);   // d[ir] 명령어를 처리한다.

int cmd\_quit(int tok\_num);  // q[uit] 명령어를 처리한다.

int cmd\_history(char\* input\_str, int tok\_num);      // hi[story] 를 처리한다.

int cmd\_typeFilename(char input[100][100],int tok\_num);

void add\_history(char\* input\_command);  // valid 한 명령어를 history\_head 에 push한다.

// memory command

int cmd\_dump(char input[100][100], int tok\_num);    // du[mp] [start, end] 명령어를 처리한다.

int cmd\_edit(char input[100][100], int tok\_num);    // e[dit] address, value 명령어를 처리한다.

int cmd\_fill(char input[100][100], int tok\_num);    // f[ill] start, end, value 명령어를 처리한다.

int cmd\_reset(int tok\_num); // reset 명령어를 처리한다.

int strtoHex(char\* parameter);  // input 으로 들어온 start, end 에 해당하는 string을 hex 로 변환하여 반환한다.

int valid\_hexa(char\* c);    //  해당 문자열이 16진수의 형태를 띄고 있는지 확인한다.

int valid\_startendRange(int st\_addr,int en\_addr);   //  들어온 start, end 의 주소가 valid 한지 확인한다.

 //opcode table command

void Open\_Opcode(); //  "opcode.txt" 를 연다.

void gen\_Opcode\_table(Opcode\_table \*pNew,int idx);  // Open\_Opcode 함수에서 opcode, mnemonic, format 을 scan하여 opcode table을 구성한다.

int cmd\_Opcode\_mnemonic(char input[100][100], int tok\_num); // opcode mnemonic 명령어를 처리한다.

int find\_Opcode\_mnemonic(char\* mnemonic);   // cmd\_Opcode\_mneonic 함수에서 입력받은 mnemonic이 opcode table 에 있는지 확인한다.

int print\_Opcode\_list(int tok\_num); //  opcodelist 명령어를 처리한다.

int create\_hashfunction(char\* mnemonic);    // opcode table 을 구성할때 쓰이는 hashfunction을 반환한다.

//assembly command

int cmd\_assembleFilename(char input[100][100], int tok\_num); // function for cmd assemble

int cmd\_symbol(char input[100][100], int tok\_num);// function for cmd symbol

int pass1(FILE\* asm\_file); // process for pass1

int pass2(); // process for pass2

int isdotAsm(char\* filename); // check if file extension end with ".asm"

int add\_symTab(char asm\_tok[100][100], int tok\_num, int LOCCTR); // add label and LOCCTR in symtab

int find\_register( char\* reg); // function for find register

int find\_symbol(char\* operand); // function for find symbol

void makeLstFile(FILE\* lstfile); // write information at .lst file

void makeObjFile(FILE\* Objfile); // write information at .obj file

Opcode\_table\* find\_OpcodeNode(char\* mnemonic); // find opcode node

void free\_symtab(symbolNode\* remove); //free symbolnode

## 20161565.c

#include "20161565.h"

int main(){

    char input\_str[100];// SIC/XE 환경에서 들어온 input string을 저장

    char real\_input[100]; // input\_str 에서 받은 input에서 앞, 뒤 공백 제거한 string

    char tmp\_history[100]; // input history

    char input\_tok[100][100]; // parsing 된 real\_input 의 command, register를 저장

    char command[100]; // parsing 된input\_tok[0]을 저장

    int par\_num; //parsing 된 문자열의 개수

    int input\_switch;   // 명령어 관련 함수에서 반환되는 값을 저장하여 맞게 작동되었는지 확인한다.

                            //input\_switch = 1 : 함수가 올바르게 작동

                            //input\_switch = 0 : 함수가 올바르게 작동하지 못하여 함수 내에서 error message 를 출력

                            //input\_switch = -1: 함수가 올바르게 작동하지 못하여 error message 출력 필요

    init();// opcode hash table을 생성하고 전역변수의 동적할당을 실행한다.

    while(1){

        printf("sicsim> ");

        fgets(input\_str,100,stdin);

        if((int) strlen(input\_str) > 100){

            printf("INPUT COMMAND LENGTH MUST BE LESS THAN 100.\n");

            continue;

        }

        input\_str[strlen(input\_str) -1] = '\0'; // fgets는 '\n' 까지 입력받음으로 마지막 인덱스에 해당하는 값을 '\0'로 변경

        strcpy(real\_input,trim\_str(input\_str)); // 들어온 input string을 ltrim, rtrim 하여 real\_input으로 복사

        strcpy(tmp\_history, real\_input);

        //

        fflush(stdin);  // 남아있는 buffer를 비워준다.

        //

        if(isvalid\_input(real\_input)== -1){// input stirng 이 invalid 한 형태일경우

            printf("\"%s\" is a invalid input. Check 'h[elp]'.\n",real\_input);

            continue;

        }

        parse\_input(input\_str,input\_tok,&par\_num); // real\_input을 tokenize한다.

        if(real\_input[0] == '\0'){// 입력받은 input string이 없거나 공백일경우

            continue;

        }

        strcpy(command,input\_tok[0]); //첫번째 tokenize string 을 command 로 복사

        if(strcmp(command,"h") == 0 || strcmp(command,"help") == 0){// h[elp]

            input\_switch = cmd\_help(par\_num);

            if(input\_switch == 1){

                add\_history(tmp\_history);

            }

            else if(input\_switch == -1){

                printf("\"%s\" is invalid option. Check h[elp].\n",real\_input);

            }

        }

        else if(strcmp(command,"d") == 0 || strcmp(command,"dir") == 0){// d[ir]

            input\_switch = cmd\_dir(par\_num);

            if(input\_switch == 1){

                add\_history(tmp\_history);

            }

            else if(input\_switch == -1){

                printf("\"%s\" is invalid option. Check h[elp].\n",real\_input);

            }

        }

        else if(strcmp(command,"q") == 0 || strcmp(command, "quit") == 0){// q[uit]

            input\_switch = cmd\_quit(par\_num);

            if (input\_switch == 1) break;

            else if(input\_switch == -1){

                printf("\"%s\" is invalid option. Check h[elp].\n",real\_input);

            }

        }

        else if(strcmp(command, "hi") == 0 || strcmp(command, "history") == 0){//hi[story]

            input\_switch = cmd\_history(real\_input, par\_num);

            if(input\_switch == -1){

                printf("\"%s\" is invalid option. Check h[elp].\n",real\_input);

            }

        }

        else if(strcmp(command, "du") == 0 || strcmp(command, "dump") == 0 ){//du[mp] [start, end]

            input\_switch = cmd\_dump(input\_tok,par\_num);

            if(input\_switch == 1){

                add\_history(tmp\_history);

            }

            else if(input\_switch == -1){

                printf("\"%s\" is invalid option. Check h[elp].\n",real\_input);

            }

        }

        else if(strcmp(command,"e") == 0 || strcmp(command, "edit")==0){// e[dit] start, end

            input\_switch = cmd\_edit(input\_tok,par\_num);

            if(input\_switch == 1){

                add\_history(tmp\_history);

            }

            else if(input\_switch == -1){

                printf("e[dit]\"%s\" is invalid option. Check h[elp].\n",real\_input);

            }

        }

        else if(strcmp(command, "f") == 0 || strcmp(command, "fill") == 0){// f[ill] start, end, value

            input\_switch = cmd\_fill(input\_tok, par\_num);

            if(input\_switch == 1){

                add\_history(tmp\_history);

            }

            else if(input\_switch == -1){

                printf("\"%s\" is invalid option. Check h[elp].\n",real\_input);

            }

        }

        else if(strcmp(command, "reset") == 0){// reset

            input\_switch = cmd\_reset(par\_num);

            if(input\_switch == 1){

                add\_history(tmp\_history);

            }

            else if(input\_switch == -1){

                printf("\"%s\" is invalid option. Check h[elp].\n",real\_input);

            }

        }

        else if(strcmp(command, "opcode") == 0){// opcode mnemonic

            input\_switch = cmd\_Opcode\_mnemonic(input\_tok,par\_num);

            if(input\_switch == 1){

                add\_history(tmp\_history);

            }

            else if(input\_switch == -1){

                printf("\"%s\" is invalid option. Check h[elp].\n",real\_input);

            }

        }

        else if(strcmp(command, "opcodelist") == 0){// opcodelist

            input\_switch = print\_Opcode\_list(par\_num);

            if(input\_switch == 1){

                add\_history(tmp\_history);

            }

            else if(input\_switch == -1){

                printf("\"%s\" is invalid option. Check h[elp].\n",real\_input);

            }

        }

        else if(strcmp(command, "type")== 0){ // cmd\_type

            input\_switch = cmd\_typeFilename(input\_tok,par\_num);

            if(input\_switch == 1){

                add\_history(tmp\_history);

            }

            else if(input\_switch == -1){

                printf("\"%s\" is invalid option. Check h[elp].\n",real\_input);

            }

        }

        else if (strcmp(command, "assemble") == 0){// cmd\_assemble

            input\_switch = cmd\_assembleFilename(input\_tok,par\_num);

            if(input\_switch == 1){

                add\_history(tmp\_history);

            }

            else if(input\_switch == -1){

                printf("\"%s\" is invalid option. Check h[elp].\n",real\_input);

            }

        }

        else if(strcmp(command, "symbol") == 0){// cmd\_symbol

            input\_switch = cmd\_symbol(input\_tok,par\_num);

            if(input\_switch == 1){

                add\_history(tmp\_history);

            }

            else if(input\_switch == -1){

                printf("\"%s\" is invalid option. Check h[elp].\n",real\_input);

            }

        }

        else{ // invalid command

            printf("\"%s\" is invalid input. Check h[elp].\n",tmp\_history);

        }

    }

    return 0;

}

void init(){

    int i;

    history\_head = NULL;    //history\_head 초기화

    symtab = NULL;

    tmp\_symtab = NULL;

    opTable = (Opcode\_table \*\*)malloc(20 \* sizeof(Opcode\_table \*)); // opTable[20] 동적할당

    for(i = 0;i < 20 ; i++) opTable[i] = NULL;  //opTable 초기화

    Open\_Opcode();  // opcode hash table 생성

    memory = (unsigned char \*)calloc(MAX\_MEMORY,sizeof(unsigned char));// 가상 메모리 동적할당

    // init assembly\_command.c

    st\_flag = 0;

    Starting\_address = 0;

    pLength = 0;

    programName[0] = 0;

    BASE\_line\_idx = 0;

    BASE\_flag = 0;

    BASE\_name[0] = 0;

    End\_flag = 0 ;

    lst\_idx = 0;

    Modify\_idx = 0;

    for(int i = 0 ; i< 100 ;i++){

        modify[i] = 0;

    }

    for(int i = 0 ; i< 2000;i++){

        lstArr[i].Loc\_counter = 0;

        lstArr[i].label\_flag = 0;

        lstArr[i].Objcode = 0;

        lstArr[i].printObjnum = 0;

        lstArr[i].line[0] = 0;

        lstArr[i].str[0] = 0;

    }

}

## Input\_handling.c

#include "20161565.h"

void parse\_input(char\* input,char parsed\_tok[100][100],int\* tok\_num){// 들어온input을 tokenzie 한다.

    char\* token;

    \*tok\_num =0;

    token = strtok(input,"\t ,");   // '\t', ' ', ','을 기준으로 input string을 tokenize

    while(token != NULL){

        strcpy(parsed\_tok[(\*tok\_num)],token);

        (\*tok\_num)++;

        token = strtok(NULL,"\t ,");

    }

};

int isvalid\_input(char\* input){// 들어온 input string이 valid 한 형태인지 확인한다.

    int i;

    int flag = 0;

    //flag 0 : command 단독

    //flag 1 : command + ' 'or '\t'..

    //flag 2 : command + ' ' + arg..

    //flag 3 : command + ' ' + arg + ','..

    //flag 4 : command + ' ' + arg + ' '..

    char tmp\_str[100];

    int char\_func;

    strcpy(tmp\_str, input);

    for(i = 0 ; tmp\_str[i] != '\0';i++){

        char\_func = isvalid\_char(tmp\_str[i]);

        if(flag == 0){//command...

            switch(char\_func){

                case 0:

                    flag = 1;

                    break;

                case 1:

                    break;

                default:

                    return -1;

            }

        }

        else if (flag == 1){//command"\_\_"...

            switch(char\_func){

                case 0:

                    break;

                case 1:

                    flag = 2;

                    break;

                default:

                    return -1;

            }

        }

        else if(flag == 2){//command\_\_"arg"...

            switch(char\_func){

                case 2:

                    flag = 3;

                    break;

                case 1:

                    break;

                case 0:

                    flag = 4;

                    break;

            }

        }

        else if(flag == 3){//command\_\_arg","...

            switch(char\_func){

                case 2:

                    return -1;

                case 1:

                    flag = 2;

                    break;

            }

        }

        else if(flag == 4){//command\_\_arg"\_\_"

            switch(char\_func){

                case 1:

                    return -1;

                case 2:

                    flag = 3;

                    break;

            }

        }

    }

    if(flag == 3) return -1; // input string이 , 로 끝나는 경우

    else return 1; // valid한 형태인 경우

};

int isvalid\_char(char c ){// 해당 character 가 valid 한지 확인

    int i;

    for(i = 0; i < 26; i++){

        if(('a' + i) == c || ('A' + i) == c) return 1;

    }// character is 'a' ~ 'z' or 'A' ~'Z''

    for(i = 0; i < 10 ;i++){

        if(('0' + i) == c) return 1;

    }// character is '0' ~ '9'

    if(c == ' ' || c == '\t') return 0; //character is space or tab

    if (c == ',') return 2; // character is comma

    return 1;

}

char\* trim\_str(char\* input){// input string 의 앞, 뒤 공백 제거

    int i=0;

    int len;

    int ptr = 0;

    char trim[100];

    if(input == NULL){

        return input;

    }

    len = strlen(input);

    strcpy(trim,input);

    for(i = 0 ; i< len ;i++){// ltrim

        if(trim[i] == ' ' || trim[i] == '\t' || trim[i] == '\n'){

            ptr++;

        }

        else break;

    }

    for(i = len-2; i >= 0;i--){ //rtrim

        if(input[i] == ' ' || input[i] == '\t' || input[i] == '\n'){

            trim[i] = '\0';

        }

        else break;

    }

    return input + ptr; //trim 된 input string 반환

};

## shell\_command.c

#include "20161565.h"

int cmd\_help(int tok\_num){// h[elp] 명령어를 처리한다.

    if(tok\_num != 1){//invalid arg

        return -1;

    }

    //valid command

    printf("\t\th[elp]\n");

    printf("\t\td[ir]\n");

    printf("\t\tq[uit]\n");

    printf("\t\thi[story]\n");

    printf("\t\tdu[mp] [start, end]\n");

    printf("\t\te[dit] address, value\n");

    printf("\t\tf[ill] start, end, value\n");

    printf("\t\treset\n");

    printf("\t\topcode mnemonic\n");

    printf("\t\topcodelist\n");

    printf("\t\tassemble filename\n");

    printf("\t\ttype filename\n");

    printf("\t\tsymbol\n");

    return 1;

};

int cmd\_dir(int tok\_num){// d[ir] 명령어를 처리한다.

    if(tok\_num != 1){// invalid arg

        return -1;

    }

    int i = 1;

    DIR\* cur\_dir = NULL;

    struct dirent\* dir\_name;

    struct stat file\_info;

    cur\_dir = opendir("."); // 현재 directory open

    if(NULL != cur\_dir){

        while((dir\_name = readdir(cur\_dir))!= NULL){

            printf("%25s",dir\_name->d\_name);

            stat(dir\_name->d\_name, &file\_info);

            switch(file\_info.st\_mode & S\_IFMT){

                case S\_IFREG://일반 file 인 경우

                    printf("\*");

                    break;

                case S\_IFDIR://directory 인 경우

                    printf("/");

                    break;

            }

            if(i % 3 == 0){

                printf("\n");

                i %= 3;

            }

            i++;

        }

    }

    else{// 현재 directory에 어떤 file, directroy 도 존재하지 않는 경우

        printf("There is no files or directories in this directory.\n");

    }

    closedir(cur\_dir);// directory close

    printf("\n");

    return 1;

};

int cmd\_quit(int tok\_num){

    //free linked list

    // history, opcode hash Table

    if(tok\_num != 1){// invalid arg

        return -1;

    }

    HISTORY\* pRemove;

    while(history\_head != NULL){

        pRemove = history\_head;

        //

        //printf("%d\n",pRemove->num);

        //

        history\_head = history\_head->link;

        free(pRemove);

    }

    //linked list : history free

    //opcode\_hash\_table

    int i;

    Opcode\_table\* opFree, \*contents;

    for(i = 0; i<20; i++){

        contents = opTable[i];

        while(contents != NULL){

            opFree = contents;

            contents = contents->link;

            free(opFree);

        }

        free(contents);

    }

    //linked list : opcode hash table free

    free(memory);

    // 1MB memory free

    return 1;

};

int cmd\_history(char\* input\_str, int tok\_num){// hi[story] 명령어를 처리한다.

    if(tok\_num != 1){//invalid arg

        return -1;

    }

    add\_history(input\_str); //입력된 hi[story] 또한 history linked list 에 저장

    HISTORY \*pWalk;

    pWalk = history\_head;

    while(pWalk != NULL){

        printf("\t%-4d %s\n",pWalk->num,pWalk->command);

        pWalk = pWalk->link;

    }

    return 1;

};

int cmd\_typeFilename(char input[100][100], int tok\_num){

    if(tok\_num != 2){

        return -1;

    }

    char filename[100];

    FILE\* fp = NULL;

    strcpy(filename, input[1]);

    fp = fopen(filename, "r");

    if(fp == NULL){

        printf("Cannot find filename \"%s\" in this directory.\n",filename);

        return 0;

    }

    char print\_ch;

    while(1){

        print\_ch = fgetc(fp);

        if(print\_ch == EOF){

            break;

        }

        printf("%c",print\_ch);

    }

    fclose(fp);

    return 1;

}

void add\_history(char \*input\_command){// valid 한 input string 을 history\_head 에 push

    HISTORY\* pNew = (HISTORY \*)malloc(sizeof(HISTORY));

    HISTORY\* pMove;

    strcpy(pNew->command, input\_command);

    pNew->link = NULL;

    pNew->num = 1;

    if(!history\_head){

        history\_head = pNew;

        return;

    }

    pMove = history\_head;

    while(pMove->link != NULL){

        pMove = pMove->link;

    }

    pMove->link = pNew;

    pNew->num += pMove->num;

};

## memory\_command.c

#include "20161565.h"

#define MIN\_VALUE 32

#define MAX\_VALUE 126

int cmd\_dump(char input[100][100], int tok\_num){// du[mp] [start, end] 명령어를 처리한다.

    // no arg

    // start only

    // start, end

    int start, end;

    int ptr = MEMORY\_IDX;

    if(tok\_num > 3){

        return -1;

    }

    switch(tok\_num){

        case 1: //dump

            if(MEMORY\_IDX >0xFFFFF) MEMORY\_IDX = 0;

            start = MEMORY\_IDX;

            end = min(MEMORY\_IDX + 0x9F, 0xFFFFF);

            MEMORY\_IDX = end + 1;

        break;

        case 2: //dump start

            start = strtoHex(input[1]);

            end = min(start + 0x9F,0xFFFFF);

            MEMORY\_IDX = end + 1;

        break;

        case 3://dump start, end

            start = strtoHex(input[1]);

            end = strtoHex(input[2]);

            MEMORY\_IDX = end + 1;

        break;

    }

    if(start == -1 || end == -1){ // input start or end are out of range

        printf("check input. start or end address is not hexamecimal number.\n");

        MEMORY\_IDX = ptr;

        return 0;

    }

    if(valid\_startendRange(start, end) == -1){//check both start and end address are valid format

        MEMORY\_IDX = ptr;

        return 0;

    }

    int i,j;

    int dump\_start = (start / 16) \* 16; //print dump start line index

    int dump\_end = (end / 16) \* 16; //print dump end line  index

    int print\_line; // number of print line

    int line\_idx;

    print\_line = (dump\_end - dump\_start) / 16 + 1;

    for(i = 0; i < print\_line; i++){

        line\_idx = dump\_start + 16 \* i;

        printf("%05X ", line\_idx);

        for(j = 0; j <16 ; j++){

            if(line\_idx + j  < start || line\_idx + j> end){

                printf("   ");

            }

            else{

                printf("%02X ", memory[line\_idx + j] );

            }

        }

        printf("; ");

        for(j = 0; j < 16; j++){

            if(line\_idx + j < start || line\_idx + j > end){

                printf(".");

            }

            else if ( memory[line\_idx + j] < MIN\_VALUE || memory[line\_idx + j] > MAX\_VALUE){

                printf(".");

            }

            else{

                printf("%c",memory[line\_idx + j]);

            }

        }

        printf("\n");

    }

    return 1;

};

int strtoHex(char\* parameter){ // 해당 parameter를 16진수로 변환하여 return

    int hexToint = 0;

    if(valid\_hexa(parameter) == -1){//invalid character in parameter

        return -1;

    }

    hexToint = strtol(parameter, NULL, 16); // strTohex

    return hexToint;

}

int valid\_hexa(char\* c){// 입력받은 c string이 16진수의 형태를 띄고 있는지 확인한다.

    // check if st and end are right range

    int i;

    for(i = 0 ; i < strlen(c);i++){

        if(('a' <= c[i]&& c[i] <= 'f') || ('A' <= c[i] && c[i] <= 'F') || ('0' <= c[i] && c[i] <= '9')) continue;

        else {

            return -1;

        }

    }

    return 1;

};

int valid\_startendRange(int st\_addr, int en\_addr){// 입력받은 start end address 가 valid 한지 확인한다.

    if( st\_addr > 0xFFFFF || en\_addr > 0xFFFFF){ // address is too big

        printf("start or end address is less than 0xFFFFF.\n");

        return -1;

    }

    if(st\_addr > en\_addr){// start address is bigger than end address

        printf("end address must be greater than start address.\n");

        return -1;

    }

    return 1;

};

int cmd\_edit(char input[100][100], int tok\_num){// e[dit] address, value 명령어를 처리한다.

    int address;

    int value;

    if(tok\_num != 3) {//invalid args

        return -1;

    }

    address = strtoHex(input[1]);

    value = strtoHex(input[2]);

    if(address == -1 || value == -1){// address or value is not hexamecimal number

        printf("Check input. Address or value is not hexadecimal number.\n");

        return 0;

    }

    if(value > 0xFF){// value is too big

        printf("Check input. Value must be less than 0xFF.\n");

    }

    if (address > 0xFFFFF){// address is too big

        printf("Check input. Address must be less than 0xFFFFF.\n");

    }

    if((value > 0xFF) || (address > 0xFFFFF)){

        return 0;

    }

    unsigned char cpy\_val = value;

    memory[address] = cpy\_val;

    return 1;

};

int cmd\_fill(char input[100][100], int tok\_num){// f[ill] start, end, value 명령어를 처리한다.

    int start, end , value;

    int i;

    if(tok\_num != 4){

        return -1;

    }

    start = strtoHex(input[1]);

    end = strtoHex(input[2]);

    value = strtoHex(input[3]);

    if(start == -1 || end == -1 || value == -1){

        printf("Check input. Start, end address or value are not hexadecimal number.\n ");

        return 0;

    }

    //start, end range check

    if(valid\_startendRange(start,end) == -1){

        return 0;

    }

    //value range check;

    if(value > 0xFF){

        printf("Value must be less than 0xFF.\n");

        return 0;

    }

    unsigned char cpy\_val = value;

    //fill memory with value from start address to end address

    for(i = start; i < end + 0x01; i++){

        memcpy(memory + i , &cpy\_val, 1);

    }

    return 1;

};

int cmd\_reset(int tok\_num){// reset 명령어를 처리한다.

    if(tok\_num != 1){

        return -1;

    }

    int i;

    for(i = 0; i < MAX\_MEMORY; i++){

        memory[i] = 0x00;

    }

    return 1;

};

## opcode\_command.c

#include "20161565.h"

void Open\_Opcode(){// "opcode.txt"를 연다.

    FILE\* fp  = NULL;

    fp = fopen("opcode.txt","r"); // open file "opcode.txt"

    if(!fp){ // can't read '"opcode.txt"

        printf("Cannot read \"opcode.txt\" file.\n");

        printf("File not exist or currpted.\n");

        return;

    }

    int opcode,idx;

    char mnemonic[10];

    char format[10];

    while(fscanf(fp,"%X %s %s\n",&opcode, mnemonic, format)!=EOF){// "opcode.txt"에서 한줄씩 scan

        Opcode\_table\* pNew = (Opcode\_table \*)malloc(sizeof(Opcode\_table));

        // create new node

        pNew->link = NULL;

        pNew->value = opcode;

        strcpy(pNew->key, mnemonic);

        strcpy(pNew->opcode\_format, format);

        idx = create\_hashfunction(mnemonic);// 입력받은 mnemonic에 해당하는 hash function 생성

        gen\_Opcode\_table(pNew,idx); // opcode hash table 구성

    }

    fclose(fp); // close "opcode.txt"

};

int create\_hashfunction(char\* mnemonic){// 입력받은 mnemonic에 해당하는 hasg function 생성

    // add all character in mnemonic string

    // return mod this by 20

    int hash = 0,i;

    for(i = 0; i< strlen(mnemonic);i++){

        hash += (int) mnemonic[i];

    }

    return hash % 20;

}

void gen\_Opcode\_table(Opcode\_table \*pNew,int idx){// opcode hash table 을 구성한다.

    //hash function에 해당하는 index에 opcode, mneminic 저장

    if(opTable[idx] == NULL){

        opTable[idx] = pNew;

    }

    else{

        Opcode\_table\* pWalk;

        pWalk = opTable[idx];

        while(pWalk->link != NULL){

            pWalk = pWalk->link;

        }

        pWalk->link = pNew;

    }

};

int cmd\_Opcode\_mnemonic(char input[100][100],int tok\_num){// opcode mnemonic 명령어를 처리한다.

    // if input start with "opcode" and next input is mnemonic

    // find mnemonic in opTable and if mnemonic exist, print matchde opcode.

    char mnemonic[100];

    int flag;

    if(tok\_num != 2){//invalid args

        return -1;

    }

    strcpy(mnemonic, input[1]);

    flag = find\_Opcode\_mnemonic(mnemonic);//입력받은  mnemonic과 일치하는 opcode return

    if( flag == -1){

        printf("\"%s\"is not exist in Opcode table.\n",input[1]);

        return 0;

    }

    else{

        return 1;

    }

};

int find\_Opcode\_mnemonic(char\* mnemonic){//opcode hash table에서 입력받은  mnemonic 과 일치하는 opcode return

    // find Opcode\_mnemonic

    // if find input mnemonic in Opcode\_hash\_table, return 0.

    // if not print caution message and return 1.

    int i;

    Opcode\_table\* pFind;

    for(i = 0; i< 20 ; i++){

        pFind  = opTable[i];

        while(pFind != NULL){

            if(strcmp(pFind->key, mnemonic) == 0){

                printf("opcode is %X\n",pFind->value);

                return 1;

            }

            pFind = pFind->link;

        }

    }

    // if input mnemonic can not found in opTable.

    return -1;

}

int print\_Opcode\_list(int tok\_num){// opcodelist 명령어를 처리한다

    //print opTable

    if(opTable == NULL){//opcode hash table이 없는 경우

        printf("Opcode list not exist. Check your file.\n");

        return 0;

    }

    if(tok\_num != 1){//invalid args

        return -1;

    }

    int i;

    Opcode\_table \*pMove;

    for(i = 0; i<20;i++){

        pMove = opTable[i];

        printf("%2d : ",i);

        if(pMove == NULL){

            printf("\n");

            continue;

        }

        while(pMove->link != NULL){

            printf("[%s, %X] -> ",pMove->key, pMove->value);

            pMove = pMove->link;

        }

        printf("[%s %X]\n",pMove->key, pMove->value);

    }

    return 1;

};

## assembly\_command.c

#include "20161565.h"

int pass1(FILE\* asm\_file){ // process for pass1

    char asm\_tok[100][100]; //store tokenize file line

    char one\_line[100], tmp\_line[100];//file line string

    char\* token; //token string

    int asmToken; // number of tokenize string

    int LOCCTR = 0; //Location counter

    int label\_flag = 0; // check if line have label

    while(fgets(one\_line, 100, asm\_file) != NULL){// read one line in asm ifle

        if(strlen(one\_line) != 0){//enter 입력제외

            one\_line[strlen(one\_line)-1] = '\0';

            strcpy(tmp\_line,one\_line);

        }

        strcpy(lstArr[lst\_idx].line, one\_line);

        //parsing

        asmToken = 0;

        token = strtok(tmp\_line,", \t");

        while(token != NULL){

            strcpy(asm\_tok[asmToken], token);

            asmToken++;

            token = strtok(NULL,", \t");

        }

        if(asmToken == 0){//all blank

            lstArr[lst\_idx].Loc\_counter = -1;

            lstArr[lst\_idx].Objcode = -1;

            lstArr[lst\_idx].label\_flag = 2;

            lst\_idx++;

            continue;

        }

        switch(one\_line[0]){

            case '.': //주석

                label\_flag = -1;

                lstArr[lst\_idx].label\_flag = label\_flag;

                break;

            case ' ': case '\t': // no label

                label\_flag = 0;

                lstArr[lst\_idx].label\_flag = label\_flag;

                break;

            default://has label

                label\_flag = 1;

                lstArr[lst\_idx].label\_flag = label\_flag;

                break;

        }

        if(label\_flag == -1){//주석

            lstArr[lst\_idx].Loc\_counter = -1;

            lstArr[lst\_idx].Objcode = -1;

            lst\_idx++;

            continue;

        }

        else if(label\_flag == 1){//if have label, store in symtab

            int sym\_flag = add\_symTab(asm\_tok,asmToken,LOCCTR);

            if(sym\_flag == -1){

                printf("Line %3d : \"%s\" duplicate symbol.\n", 5 + lst\_idx \* 5, asm\_tok[0]);

            }

        }

        //assembly directive

        //"START"

        if(st\_flag == 0){//first line

            if((strcmp(asm\_tok[label\_flag],"START") == 0)){//OPCODE"START"

                Starting\_address = atoi(asm\_tok[label\_flag + 1]);

                lstArr[lst\_idx].Loc\_counter = Starting\_address;

                lstArr[lst\_idx].Objcode = -1;

                LOCCTR = Starting\_address;

                strcpy(programName,asm\_tok[0]);

                lst\_idx++;

                st\_flag = 1;

                continue;

            }

            // OPCODE "START" 미존재시 Starting address에 0 저장 후 LOCCTR 에 Starting address 저장

            LOCCTR = Starting\_address = 0;

            st\_flag = -1;

        }

        //"BASE"

        if(strcmp(asm\_tok[label\_flag],"BASE") == 0){

            lstArr[lst\_idx].Loc\_counter = -1;

            lstArr[lst\_idx].Objcode = -1;

            BASE\_flag = 1;

            BASE\_line\_idx = lst\_idx;

            strcpy(BASE\_name,asm\_tok[label\_flag+1]);

        }

        //"BYTE"

        else if(strcmp(asm\_tok[label\_flag],"BYTE") == 0){

            lstArr[lst\_idx].Loc\_counter = LOCCTR;

            lstArr[lst\_idx].Objcode = 0;

            if(asm\_tok[label\_flag+1][0] == 'X'){

                LOCCTR += 1;

            }

            else if(asm\_tok[label\_flag+1][0] == 'C'){

                LOCCTR += strlen(asm\_tok[label\_flag+1])-3; // C , ',' 제외

            }

        }

        //"WORD"

        else if(strcmp(asm\_tok[label\_flag],"WORD") == 0){

            lstArr[lst\_idx].Loc\_counter = LOCCTR;

            lstArr[lst\_idx].Objcode = 0;

            LOCCTR +=3;

        }

        //"RESB"

        else if(strcmp(asm\_tok[label\_flag],"RESB") == 0){

            lstArr[lst\_idx].Loc\_counter = LOCCTR;

            lstArr[lst\_idx].Objcode = -1;

            LOCCTR += atoi(asm\_tok[label\_flag +1]);

        }

        //RESW

        else if(strcmp(asm\_tok[label\_flag],"RESW") == 0){

            lstArr[lst\_idx].Loc\_counter = LOCCTR;

            lstArr[lst\_idx].Objcode = -1;

            LOCCTR += 3 \* atoi(asm\_tok[label\_flag + 1]);

        }

        //END

        else if(strcmp(asm\_tok[label\_flag],"END") == 0){

            lstArr[lst\_idx].Loc\_counter = -1;

            lstArr[lst\_idx].Objcode = -1;

            End\_flag = 1;

            pLength = LOCCTR - Starting\_address;

            lst\_idx++;

            break;

        }

        //opcode mnemonic

        else{

            char opcd[100];

            int ext\_flag =0;

            int format;

            if(asm\_tok[label\_flag][0] == '+'){//extended mode

                ext\_flag = 2;

                strcpy(opcd, asm\_tok[label\_flag] + 1);

            }

            else strcpy(opcd,asm\_tok[label\_flag]);

            Opcode\_table\* opFind = find\_OpcodeNode(opcd);

            if(opFind == NULL){ //cannot find opcode mnemonic in opcode hash table

                //error message

                printf("line %3d : Invalid opcode mnemonic \"%s\".\n",5 \* lst\_idx + 5, asm\_tok[label\_flag]);

                return 0;

            }

            lstArr[lst\_idx].Loc\_counter = LOCCTR;

            lstArr[lst\_idx].Objcode = 0;

            format = (int) opFind->opcode\_format[ext\_flag] - '0';

            // opcode node 에 저장된 정보 -> lstNode

            LOCCTR += format;

        }

        lst\_idx++;

    }

    return 1;

}

int pass2(){ // process for pass2

    int asmToken = 0; //number of tokenize string

    char asm\_tok[100][100]; //store tokenize string

    char tmp\_line[100], tmp\_str[100]; //store temp string

    char OPCODE[100]; // OPCODE string

    char OPERAND[100]; // OPERAND string

    char \*token; //tokenize  string

    int idx; // index

    int n ,i, x, b, p, e, opHex1,opHex2,disp; // bit n,i,x,b.p,e and opHex1,2 for store opcode ,and disp address

    int address\_loc, tmpValue; // address\_loc for symbol label

    long long Cal\_ObjCode = 0; // store calculated objccode

    long long nixbpe = 0; // store nixbpe bits

    for(idx = 0; idx < lst\_idx; idx++){

        if(lstArr[idx].Objcode == -1){ //dosen't need to calculate objcode

            continue;

        }

        //init

        n = i = x = b = p = e = opHex1 = opHex2 = disp = 0;

        strcpy(tmp\_line,lstArr[idx].line);

        // tokenize string

        asmToken = 0;

        token = strtok(tmp\_line,", \t");

        while(token != NULL){

            strcpy(asm\_tok[asmToken], token);

            asmToken++;

            token = strtok(NULL,", \t");

        }

        for(int k = 0; k < 100 ;k++){//init2

                asm\_tok[asmToken][k] = 0;

        }

        int idx\_label = lstArr[idx].label\_flag;

        if(strcmp(asm\_tok[idx\_label],"BYTE") == 0){//byte

            if(asm\_tok[idx\_label + 1][0] == 'C'){// store char

                strcpy(tmp\_str, asm\_tok[idx\_label+1] +2);

                tmp\_str[strlen(tmp\_str)-1] = '\0';

                for(int k = 0;k < strlen(tmp\_str);k++){

                    disp \*= 256;

                    disp += tmp\_str[k];

                }

                lstArr[idx].Objcode = disp;

            }

            else if(asm\_tok[idx\_label + 1][0] == 'X'){//store hex

                strcpy(tmp\_str, asm\_tok[idx\_label+1] +2);

                tmp\_str[strlen(tmp\_str)-1] = '\0';

                lstArr[idx].Objcode = strtoHex(tmp\_str);

                strcpy(lstArr[idx].str,tmp\_str);

                lstArr[idx].printObjnum = -1;

                if(lstArr[idx].Objcode == -1){

                    printf("Line %3d : Wrong Hexnumber.\n", idx\*5 + 5);

                }

            }

            else{// wrong operand

                printf("Line %3d : Wrong byte operand.\n", idx \* 5 + 5);

                return 0;

            }

        }

        else if(strcmp(asm\_tok[idx\_label],"WORD") == 0){//word

            lstArr[idx].Objcode = (long long)atoi(asm\_tok[idx\_label+1]);

            lstArr[idx].printObjnum = 6;

        }

        else if(strcmp(asm\_tok[idx\_label],"RESB") == 0){//resb

            lstArr[idx].Objcode = -1;

        }

        else if(strcmp(asm\_tok[idx\_label],"RESW") == 0){// resw

            lstArr[idx].Objcode = -1;

        }

        else{

            int format\_idx = 0;

            if(asm\_tok[idx\_label][0] == '+'){ // extend mode

                strcpy(OPCODE,asm\_tok[idx\_label]+1);

                e = 1;

            }

            else{

                strcpy(OPCODE,asm\_tok[idx\_label]);

            }

            if(e == 1) format\_idx = 2;

            Opcode\_table\* tmpOpnode = find\_OpcodeNode(OPCODE);

            opHex1 = tmpOpnode->value / 16;

            opHex2 = tmpOpnode->value % 16;

            if(tmpOpnode->opcode\_format[format\_idx] == '1'){

                //format 1

                lstArr[idx].Objcode = tmpOpnode->value;

                lstArr[idx].printObjnum = 2;

                //lstArr[]

            }

            else if(tmpOpnode->opcode\_format[format\_idx] == '2'){

                //format 2

                tmpValue = find\_register(asm\_tok[idx\_label+1]);

                if(tmpValue == -1){

                    printf("Line %3d : Cannot find register \"%s\".\n", idx\* 5 + 5, asm\_tok[idx\_label+1]);

                    return 0;

                }

                Cal\_ObjCode = (long long) (tmpOpnode->value << 8) + (tmpValue << 4);

                tmpValue = find\_register(asm\_tok[idx\_label+2]);

                if(tmpValue == -1){

                    printf("Line %3d : Cannot find register \"%s\".\n", idx\* 5 + 5, asm\_tok[idx\_label+1]);

                    return 0;

                }

                Cal\_ObjCode += (long long) tmpValue;

                lstArr[idx].Objcode = Cal\_ObjCode;

                lstArr[idx].printObjnum = 4;

            }

            else if(tmpOpnode->opcode\_format[format\_idx] == '3' ||tmpOpnode->opcode\_format[format\_idx] == '4' ){

                //format 3 or 4

                if(asmToken > idx\_label + 1){

                    if(asm\_tok[idx\_label+1][0] == '@'){// indirec addressing

                        n = 1;

                        i = 0;

                        strcpy(OPERAND,asm\_tok[idx\_label+1]+1);

                    }

                    else if(asm\_tok[idx\_label+1][0] == '#'){//immediate addressing

                        n = 0;

                        i = 1;

                        strcpy(OPERAND,asm\_tok[idx\_label+1]+1);

                    }

                    else{// simple addressing

                        n = 1;

                        i = 1;

                        strcpy(OPERAND,asm\_tok[idx\_label+1]);

                    }

                    if(asmToken == idx\_label + 3){//if number of operand is 2

                        x = 1;

                    }

                    address\_loc = find\_symbol(OPERAND);

                    if(address\_loc == -1){

                        //error message

                        disp = atoi(OPERAND);

                        if(disp == 0 && OPERAND[0] != '0'){// cannot find symbol

                            printf("Line %3d : Symbol \"%s\" dosen't exist.\n", idx\*5+5,OPERAND);

                            return 0;

                        }

                        //stored symbol LOCCTR out of range

                        if(e == 0 && disp >= 4096){//format 3

                            printf("Line %3d : Displacement out of range.\n", idx\*5+5);

                            return 0;

                        }

                        if(e == 1 && disp >= 1048576){//format 4

                            printf("Line %3d : Displacement out of range.\n", idx\*5+5);

                            return 0;

                        }

                    }

                    else if(e == 0){

                        int PC = address\_loc - lstArr[idx].Loc\_counter - 3;

                        if(-2048 <= PC && PC <= 2047){//PC relative

                            p = 1;

                            if(PC < 0){

                                PC += 4096;

                                disp = PC;

                            }

                            else{

                                disp = PC;

                            }

                        }

                        else {// base relative

                            b = 1;

                            if(BASE\_flag == 1){

                                int BASE\_LOC = find\_symbol(BASE\_name);

                                if(BASE\_LOC == -1){

                                    printf("Line %3d : Wrong Base name.\n",idx\*5 +5);

                                    return 0;

                                }

                                disp = address\_loc - BASE\_LOC;

                            }

                            else{// no "BASE" OPCODE

                                disp = -1;

                            }

                            if(0 > disp && disp > 4095){

                                printf("Line %3d : Address out of range.\n", idx\*5 + 5);

                                return 0;

                            }

                        }

                    }

                }

                else{

                    n = 1;

                    i = 1;

                }

                if(e){

                    b = 0;

                    p = 0;

                }

                //calculate objcode

                Cal\_ObjCode = (long long) tmpOpnode->value >> 2;

                nixbpe = (long long) (n << 5);

                nixbpe += (long long) (i << 4);

                nixbpe += (long long) (x << 3);

                nixbpe += (long long) (b << 2);

                nixbpe += (long long) (p << 1);

                nixbpe += (long long) e;

                lstArr[idx].Objcode = (Cal\_ObjCode << 6) + nixbpe;

                if(e == 0){// format 3

                    lstArr[idx].Objcode \*= (1 << 12);

                    lstArr[idx].Objcode += disp;

                    lstArr[idx].printObjnum = 6;

                }

                else{ //if extend

                    if(address\_loc != -1){

                        disp = address\_loc;

                        modify[Modify\_idx++] = lstArr[idx].Loc\_counter + 1;

                        lstArr[idx].printObjnum = 8;

                    }

                    lstArr[idx].Objcode \*= (1<<20);

                    lstArr[idx].Objcode += disp;

                    lstArr[idx].printObjnum = 8;

                }

            }

        }

    }

    return 1;

}

int cmd\_assembleFilename(char input[100][100], int tok\_num){// function for cmd assemble

    if(tok\_num != 2){//wrong option

        return -1;

    }

    FILE\* fp = NULL;

    char filename[100], lstfilename[100], objfilename[100];

    int step1,step2;

    strcpy(filename, input[1]);

    if(isdotAsm(filename) == 0){

        return 0;

    }

    fp  = fopen(filename, "r");

    if(fp == NULL){//file not exist

        printf("Cannot find filename \"%s\" in this directory.\n",filename);

        return 0;

    }

    //assemble init

    tmp\_symtab = symtab;

    symtab = NULL;

    st\_flag = 0;

    Starting\_address = 0;

    pLength = 0;

    programName[0] = 0;

    BASE\_line\_idx = 0;

    BASE\_flag = 0;

    BASE\_name[0] = 0;

    End\_flag = 0 ;

    lst\_idx = 0;

    Modify\_idx = 0;

    for(int i = 0 ; i< 100 ;i++){

        modify[i] = 0;

    }

    for(int i = 0 ; i< 2000;i++){

        lstArr[i].Loc\_counter = 0;

        lstArr[i].label\_flag = 0;

        lstArr[i].Objcode = 0;

        lstArr[i].printObjnum = 0;

        lstArr[i].line[0] = 0;

        lstArr[i].str[0] = 0;

    }

    //pass1 실행

    step1 = pass1(fp);

    if(step1 != 1){

        //free tmp\_symtab

        symtab = tmp\_symtab;

        fclose(fp);

        return 0;

    }

    //pass2 실행

    step2 = pass2();

    if(step2 != 1){

        //free symtab

        symtab = tmp\_symtab;

        fclose(fp);

        return 0;

    }

    // pass1 과 pass2 가 정상적으로 작동하였을때

    fclose(fp);

    //free\_symtab(symtab);

    tmp\_symtab = NULL;

    char tmp\_fname[100];

    char\* tok\_filename;

    strcpy(tmp\_fname,filename);

    // create filename

    tok\_filename = strtok(tmp\_fname,".");

    strcpy(lstfilename, tok\_filename);

    strcat(lstfilename,".lst");

    strcpy(objfilename, tok\_filename);

    strcat(objfilename,".obj");

    //filename.lst 생성

    fp = fopen(lstfilename,"w");

    makeLstFile(fp);

    fclose(fp);

    //filename.obj 생성

    fp = fopen(objfilename,"w");

    makeObjFile(fp);

    fclose(fp);

    printf("[%s], [%s]\n",lstfilename,objfilename);

    return 1;

}

void makeLstFile(FILE\* lstfile){// write information at .lst file

    int i;

    for(i = 0; i < lst\_idx; i++){

        fprintf(lstfile, "%-8d\t",i \* 5 + 5); //print line num

        if(lstArr[i].Loc\_counter == -1){

            fprintf(lstfile, "    \t");

        }

        else{

            fprintf(lstfile, "%04X\t", lstArr[i].Loc\_counter);

        }

        fprintf(lstfile, "%s", lstArr[i].line);

        if(lstArr[i].Objcode == -1){

            fprintf(lstfile, "\n");

        }

        else{

            for(int j = strlen(lstArr[i].line); j < 40;j++){

                fprintf(lstfile," ");

            }

            switch (lstArr[i].printObjnum)

            {

            case 2:

                fprintf(lstfile, "%02llX\n",lstArr[i].Objcode);

                break;

            case 4:

                fprintf(lstfile, "%04llX\n",lstArr[i].Objcode);

                break;

            case 6:

                fprintf(lstfile, "%06llX\n",lstArr[i].Objcode);

                break;

            case 8:

                fprintf(lstfile, "%08llX\n",lstArr[i].Objcode);

                break;

            default:

                fprintf(lstfile, "%02llX\n",lstArr[i].Objcode);

                break;

            }

        }

    }

}

void makeObjFile(FILE\* Objfile){// write information at .obj file

    int i;

    char str[300] = {0,},tmp\_oneline[100];

    fprintf(Objfile,"H%-6s%06X%06X\n",programName,lstArr[0].Loc\_counter,pLength);

    for(i = 0; i< lst\_idx;){

        str[0] = '\0';

        tmp\_oneline[0] = '\0';

        if(lstArr[i].Loc\_counter == -1 || lstArr[i].Objcode == -1){

            i++;

            continue;

        }

        fprintf(Objfile,"T%06X",lstArr[i].Loc\_counter);

        for(;i<lst\_idx;i++){

            if(lstArr[i].Loc\_counter == -1){

                continue;

            }

            if(lstArr[i].Objcode == -1){

                break;

            }

            else if(lstArr[i].printObjnum == -1){

                strcpy(tmp\_oneline,lstArr[i].str);

                //fprintf(Objfile,"%02llX",lstArr[i].Objcode);

            }

            else{

                switch(lstArr[i].printObjnum){

                    case 2:

                        sprintf(tmp\_oneline,"%02llX",lstArr[i].Objcode);

                        break;

                    case 4:

                        sprintf(tmp\_oneline,"%04llX",lstArr[i].Objcode);

                        break;

                    case 6:

                        sprintf(tmp\_oneline,"%06llX",lstArr[i].Objcode);

                        break;

                    case 8:

                        sprintf(tmp\_oneline,"%08llX",lstArr[i].Objcode);

                        break;

                    default:

                        sprintf(tmp\_oneline,"%llX",lstArr[i].Objcode);

                        break;

                }

            }

            if(strlen(str) + strlen(tmp\_oneline) > 60){

                i--;

                break;

            }

            strcat(str,tmp\_oneline);

        }

        fprintf(Objfile,"%02X%s\n",(int)strlen(str)/2,str);

        i++;

    }

    for(i = 0; i<Modify\_idx;i++){//modify

        fprintf(Objfile, "M%06X05\n",modify[i]);

    }

    //if(st\_flag == 1)

    fprintf(Objfile, "E%06X\n", lstArr[0].Loc\_counter);

}

int isdotAsm(char\* filename){// check if file extension end with ".asm"

    char\* dotAsm = ".asm";

    char\* ptr = strstr(filename, dotAsm);

    if(ptr == NULL || strcmp(dotAsm,ptr) != 0){

        printf("Valid filename extension of source file is \".asm\". Check your input.\n");

        return 0;

    }

    return 1;

}

int cmd\_symbol(char input[100][100], int tok\_num){// function for cmd symbol

    if(tok\_num != 1){

        return -1;

    }

    if(symtab == NULL){

        printf("SYMTAB not exist!\n");

        return 1;

    }

    symbolNode\* pWalk;

    pWalk = symtab;

    while(pWalk != NULL){

        printf("\t%s\t%04X\n",pWalk->Label,pWalk->Loc\_counter);

        pWalk = pWalk->link;

    }

    return 1;

}

int add\_symTab(char asm\_tok[100][100], int tok\_num,int LOCCTR){

    symbolNode\* pNew, \*pWalk, \*pPrev;

    pNew = (symbolNode \*)malloc(sizeof(symbolNode));

    pNew->Loc\_counter = LOCCTR;

    strcpy(pNew->Label, asm\_tok[0]);

    pNew->link = NULL;

    if(symtab == NULL){

        symtab = pNew;

        return 1;

    }

    pWalk = symtab;

    pPrev = pWalk;

    while(pWalk != NULL){

        if(strcmp(pWalk->Label, pNew->Label) == 0){

            return -1; // same label error

        }

        if(pWalk == symtab){

            if(strcmp(pWalk->Label,pNew->Label) > 0){

                pNew->link = pWalk;

                symtab = pNew;

                return 1;

            }

        }

        else if(strcmp(pWalk->Label,pNew->Label) > 0){

                pNew->link = pPrev->link;

                pPrev->link = pNew;

                return 1;

        }

        pPrev = pWalk;

        pWalk = pWalk->link;

    }

    pPrev->link = pNew;

    return 1;

}

int find\_symbol(char\* operand){// function for find symbol

    symbolNode\* pWalk;

    pWalk = symtab;

    while(pWalk != NULL){

        if(strcmp(pWalk->Label,operand) == 0){

            return pWalk->Loc\_counter;

        }

        pWalk = pWalk->link;

    }

    return -1;

}

int find\_register(char\* reg){// function for find register

    if(strcmp(reg, "A") == 0 || strcmp(reg, "") == 0){

        return 0;

    }

    else if(strcmp(reg, "X") == 0){

        return 1;

    }

    else if(strcmp(reg, "L") == 0){

        return 2;

    }

    else if(strcmp(reg, "B") == 0){

        return 3;

    }

    else if(strcmp(reg, "S") == 0){

        return 4;

    }

    else if(strcmp(reg, "T") == 0){

        return 5;

    }

    else if(strcmp(reg, "F") == 0){

        return 6;

    }

    else if(strcmp(reg, "PC") == 0){

        return 8;

    }

    else if(strcmp(reg, "SW") == 0){

        return 9;

    }

    // can not find register

    return -1;

}

Opcode\_table\* find\_OpcodeNode(char\* mnemonic){//opcode hash table에서 입력받은  mnemonic 과 일치하는 opcode return

    // find Opcode\_mnemonic

    // if find input mnemonic in Opcode\_hash\_table, return 0.

    // if not print caution message and return 1.

    int i;

    Opcode\_table\* pFind;

    for(i = 0; i< 20 ; i++){

        pFind  = opTable[i];

        while(pFind != NULL){

            if(strcmp(pFind->key, mnemonic) == 0){

                //printf("opcode is %X\n",pFind->value);

                return pFind;

            }

            pFind = pFind->link;

        }

    }

    // if input mnemonic can not found in opTable.

    return NULL;

}

void free\_symtab(symbolNode\* remove){//free symbolnode

    symbolNode\* pRemove;

    while(remove != NULL){

        pRemove = remove;

        //

        //printf("%d\n",pRemove->num);

        //

        remove = remove->link;

        free(pRemove);

    }

}