과목 명: 시스템프로그래밍

담당 교수 명: 소 정 민

<<Assignment 2>>

**서강대학교 컴퓨터공학과**

**20171601**

**강주형**

**목 차**

1. 프로그램 개요 4

2. 프로그램 설명 4

2.1 프로그램 흐름도 4

3. 모듈 정의 5

3.1 모듈 이름 : main() 5

3.1.1 기능 5

3.1.2 사용 변수 5

3.2 모듈 이름: check1(char \*origin,char \*\*command, Sym \*\*\*symhead, int \*symbol\_print\_flag), check2(char \*origin,char \*\*command, int TOKEN\_SIZE, Sym \*\*\*symhead,int \*symbol\_print\_flag) 5

3.2.1 기능 6

3.2.2 사용 변수 6

3.3 모듈이름: help() 6

3.3.1 기능 6

3.3.2 사용변수 6

3.4 모듈이름: type(char \*origin, char \*\*command) 6

3.4.1 기능 6

3.4.2 사용변수 6

3.5 모듈이름: assemble(char \*origin, char \*\*command, Op \*\*ophead, Sym \*\*\*output, int \*symbol\_print\_flag) 6

3.5.1 기능 6

3.5.2 사용변수 7

3.6 모듈이름: build\_symtab(Sym\*\* symhead, Sym\* node, int line\_num) 8

3.6.1 기능 8

3.6.2 사용변수 8

3.7 모듈이름: check\_mnemonic(char \*mnemonic, char \*format, int \*opcode, OPCODE \*\*ophead) 8

3.7.1 기능 8

3.7.2 사용변수 8

3.8 모듈이름: location\_counter(char \*format, unsigned int \*location, char \*mnemonic 9

3.8.1 기능 9

3.8.2 사용변수 9

3.9 모듈이름: find\_symbol(Sym \*\*symhead, char \*symbol) 9

3.9.1 기능 9

3.9.2 사용변수 9

3.10 모듈이름: reg\_num(char \*reg) 9

3.10.1 기능 9

3.10.2 사용변수 9

3.11 모듈이름: objectcode\_insert(int format, int first, int mid, int end, int location, Ob\*\* head, Ob\*\* tail, char\* var, int enter\_flag) 9

3.11.1 기능 9

3.11.2 사용변수 9

3.12 모듈이름: allocate(Sym \*data) 9

3.12.1 기능 9

3.12.2 사용변수 9

3.13 모듈이름: symbol(Sym \*\*symhead, int symbol\_print\_flag) 10

3.13.1 기능 10

3.13.3 사용변수

4. 전역 변수 정의 10

4.1 Node \*head 10

4.2 Op \*ophead[20] 10

4.3 unsigned char MEMORY[MEMORY\_SIZE] 10

4.4 unsigned int address 10

5. 코드 10

5.1 20171601.h 10

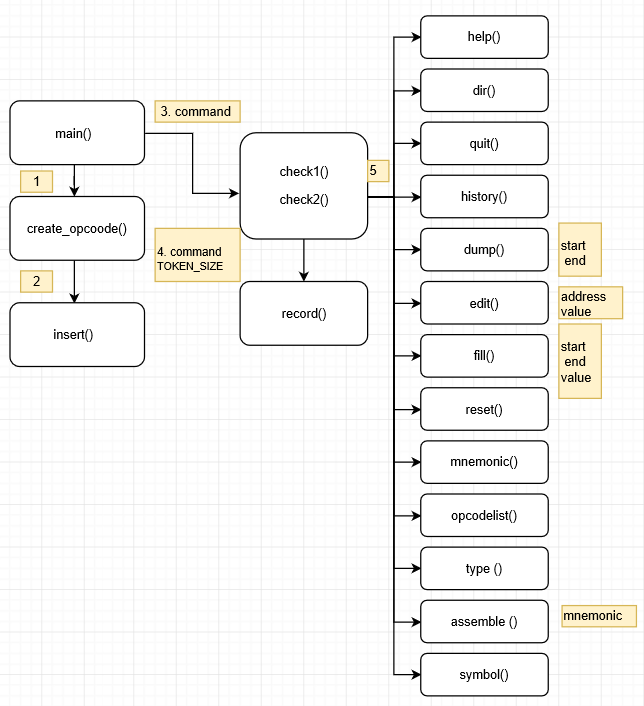
5.2 20171601.c 11

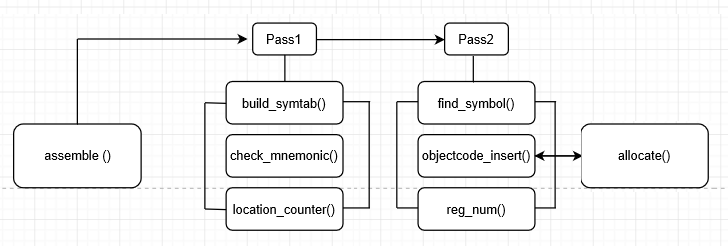
1. **프로그램 개요**

이 프로그램은 프로젝트 1에서 개발한 쉘 프로그램에 추가적인 기능을 추가할 것이다. 이제 프로그램은 SIC/XE 어셈블리 프로그램 원본 파일을 입력으로 읽고 객체 파일과 어셈블리 목록 파일을 만들 것이다. 또한 프로그램은 사용자를 위한 기호 테이블과 객체 파일을 표시할 수 있어야 한다.

프로그램 실행 시unix shell과 유사한 입력 프롬프트상태가 되며, 해당 상태에서 프로그램에 내장된 명령어들을 입력할 때, 그에 해당하는 기능을 수행한다. 명령어는 크게 셸 관련 명령어(help, dir, quit, history, type)와 메모리공간 관련 명령어(dump, edit, fill, reset), opcode와 관련된 명령어(opcodelist, opcode mnemonic), SIC/XE 어셈블러 관련 명령어 (assemble, symbol)가 있다.

1. **프로그램 설명**
   1. **프로그램 흐름도**



****

**그림 1> 프로그램 흐름도**

1. **모듈 정의**
   1. **모듈 이름 : main()**
      1. *기능*

unix shell과 유사한 입력 프롬프트 상태(sicsim> )를 출력하며 사용자로부터 직접 명령어를 입력받으며 quit 명령어가 입력되기 전까지 계속해서 입력받는다. 또한 입력받은 명령어를 공백과 탭을 기준으로 TOKEN화 하여 해당 명령어 수행을 위해 command\_check 모듈로 넘겨주고 return값에 따라 프로그램을 종료할지, history에 저장할지 결정하는 모듈

* + 1. *사용 변수*

char command[INPUT\_SIZE] : 입력받은 명령어를 저장할 배열

char command\_temp[INPUT\_SIZE] : 입력받은 명령어를 TOKEN화 하기 위해 임시로 command를 저장할 배열

char \*\*command\_token : 입력받은 명령어를 TOKEN으로 만들어 저장할 배열

char \*ptr : TOKEN화 하면서 자를 위치를 가리킬 포인터

int flag : quit명령어를 통해 프로그램의 종료를 알리기 위해 true, false 값을 저장할 변수

int TOKEN\_SIZE : 입력받은 명령어의 TOKEN의 개수를 저장할 변수

int symbol\_print\_flag : symbol 명령어 수행시 출력을 결정을 알리기 위해 true, fals값을 저장할 변수

int i : 반복문 수행을 위한 변수

Sym \*\*symhead : symbol table의 각 인덱스의 헤드포인터를 담은 구조체

* 1. **모듈 이름: check1(char \*origin,char \*\*command, Sym \*\*\*symhead, int \*symbol\_print\_flag), check2(char \*origin,char \*\*command, int TOKEN\_SIZE, Sym \*\*\*symhead,int \*symbol\_print\_flag)**
     1. *기능*

Main 모듈에서 넘겨받은 명령어를 TOKEN\_SIZE기준으로 나누어 프로그램에 내장된 명령어들과 동일한 명령어를 입력 받았는지 확인하여, 올바른 명령어를 입력 받았다면 해당명령어를 수행하는 모듈을 수행할 수 있도록 해당 모듈을 부르며, 잘못된 명령어의 경우 에러메시지를 출력하는 모듈

* + 1. *사용 변수*

unsigned int start – dump와 fill 명령어 수행 시 시작번지를 저장할 변수

unsigned int end – dump와 fill 명령어 수행 시 끝번지를 저장할 변수

unsigned int addr – edit 명령어 수행 시 타겟번지를 저장할 변수

unsigned int new – edit와 fill 명령어 수행 시 입력할 data를 저장할 변수

unsigned int trash – 정해진 parameter 이외에 값이 들어올 경우를 체크할 변수

int count – 입력받아야 할 parameter가 존재하는 명령어들에 대하여 comma 개수를 담을 변수

int i – 반복문 수행을 위한 변수

* 1. **모듈이름: help()**
     1. *기능*

프로그램에 내장된 모든 명령어들을 출력한다.

* + 1. *사용변수*

없음

* 1. **모듈이름: type(char \*origin, char \*\*Command)**
     1. *기능*

사용자가 입력한 filename에 대하여 현재 디렉토리 내에서 찾아서 화면에 그대로 출력하는 모듈

해당 파일이 현재 디렉토리 내에 존재하지 않는 경우 에러를 출력한다.

* + 1. *사용변수*

FILE \*fp – filename에 해당하는 파일을 열기 위한 파일포인터

char string[500] – file의 내용을 읽어서 담기 위한 배열

* 1. **모듈이름: assemble(char \*origin, char \*\*command, Op\*\*ophead, Sym \*\*\*output, int \*symbol\_print\_flag)**
     1. *기능*

사용자가 입력한 filename에 대하여 현재 디렉토리 내에서 찾아서 assemble하여 listing file과 object file을 생성하며 file의 확장자는 .asm이어야 한다. 없는 파일이거나 확장자가 다른 경우 에러 출력한다. listing file의 파일명은 filename의 확장자를 lst로, object file의 파일명은 obj로 변경한 것이다. 소스파일에 에러가 존재하는 경우 해당 라인과 에러 내용을 화면에 출력하며 listing, object file은 생성하지 않는다

* + 1. *사용변수*

FILE \*fp – filename에 해당하는 파일을 열기 위한 파일포인터

FILE \*mid – intermediate 파일 작성을 위한 파일 포인터

FILE \*lst – listing 파일 작성을 위한 파일 포인터

FILE \*obj – object 파일 작성을 위한 파일 포인터

char \*lstname – listing 파일 파일명을 저장할 배열

char \*objname – object 파일 파일명을 저장할 배열

char temp[INPUT\_SIZE] – intermediate 파일에서 읽은 string을 담기 위한 배열

char operator[10] – assemble 과정에서 operator를 담을 배열

char operand1[255] – assemble 과정에서 첫번째 operand를 담을 배열

(operand2와 size가 다른 이유는 BYTE에서 선언할 수 있는 문자열의 길이를 cover하기 위해)

char operand2[10] – assemble 과정에서 두번째 operand를 담을 배열

char format[5] – assemble 과정에서 object code format을 담을 배열

char optemp[10] – operand의 연산을 위해 임시로 operand를 담을 배열

char name[10] – directive 명령 START를 만났을 때 program name을 담기 위한 배열

int opcode – assemble 과정에서 opcode를 담기 위한 변수

int reg1 – assemble 과정에서 format2의 register1을 담을 변수

int reg2 – assemble 과정에서 format2의 register2를 담을 변수

int num – operand 개수 판단을 위한 sscanf 함수의 return값을 받을 변수

int disp – format 3,4에서 최종 TA계산을 위해 displacement 값을 저장할 변수

int pc – pc relative mode를 위한 변수

int base – base relative mode를 위한 변수

int base\_flag – BASE 선언시 base relative 활성화를 위한 flag

int start\_flag – 소스파일에서 START를 발견했다고 알리기 위한 flag

int end\_flag – 소스파일에서 END를 발견했다고 알리기 위한 flag

int symbol\_flag – operand에 해당하는 symbol의 location 반환 후 타입 체크를 위한 flag

int first\_data – object code의 앞의 2개의 hexa data를 담을 변수

int mid\_data – object code의 앞의 2개의 hexa data와 address부분을 제외한 hexa data를 담을 변수

int i – 반복문 수행을 위한 변수

int var – WORD, RESW, RESB 만났을 시 operand 상수를 담을 변수

unsigned int line\_num – 현재 assemble 중인 line의 number를 저장할 변수

unsigned int location – location count를 위한 변수

unsigned int length – 전체 assemble code의 길이를 담을 변수

Sym \*\*symhead – assemble 과정에서 생성되는 symbol table

Sym \*node – symbol table build를 위한 node

Ob \*head – OBJECT CODE를 담은 linked list의 head 포인터

Ob \*tail – OBJECT CODE를 담은 linked list의 tail 포인터

Ob \*ptr1 – OBJECT CODE 출력시 현재 위치를 가리킬 포인터

Ob \*ptr2 – OBJECT CODE 출력시 이전 위치를 가리킬 포인터

* 1. **모듈이름: build\_symtab(Sym\*\* symhead, Sym\* node, int line\_num)**
     1. *기능*

assemble 명령어 수행 시 소스파일에서 symbol을 읽어 symbol table을 build하는 모듈

symbol table 내에 중복된 symbol 발견 시 duplicate symbol error 출력.

* + 1. *사용변수*

int index – hash\_function을 통해 symbol table의 index를 저장할 변수

Sym \*temp – 함수 파라메터로 받은 linked list node의 정보를 복사할 node

Sym \*ptr – 새로운 node 연결을 위해 symbol table을 탐색할 포인터

* 1. **모듈이름: check\_mnemonic(char \*mnemonic, char \*format, int \*opcode, Op \*\*ophead)**
     1. *기능*

assemble 명령어 수행 시 소스파일에서 읽어낸 operator의 opcode를 얻기 위해 프로그램에서 관리하는 opcode table에서 opcode를 탐색하는 모듈

* + 1. *사용변수*

char data[10] : format 4의 경우 operator에 ‘+’문자가 있으므로 operator만 담기 위한 배열

Op \*temp : opcode table 탐색을 위한 포인터

* 1. **모듈이름: location\_counter(char \*format, unsigned int \*location, char \*mnemonic**
     1. *기능*

listing file 작성을 위해 각 line의 location을 계산하기 위한 모듈

* + 1. *사용변수*

없음

* 1. **모듈이름 find\_symbol(Sym \*\*symhead, char \*symbol)**
     1. *기능*

object code 계산을 위해 operand symbol의 location 값을 찾기 위한 모듈, symtab에 없는 symbol에 대한 탐색 수행 시 에러 출력. operand가 상수인 경우 flag를 통해 상수인지 아닌지 알려줌.

* + 1. *사용변수*

Sym \*temp – symbol table 탐색을 위한 포인터

int const\_flag – operator symbol이 상수인지 아닌지 체크하기 위한 flag

int i – 반복문 수행을 위한 변수

* 1. **모듈이름: reg\_num(char \*reg)**
     1. *기능*

format2의 object code인 경우 register name에 해당하는 register number를 반환하는 모듈

* + 1. *사용변수*

없음

* 1. **모듈이름: objectcode\_insert(int format, int first, int mid, int end, int location, Ob\*\* head, Ob\*\* tail, char\* var, int enter\_flag)**
     1. *기능*

object file작성을 위해 계산된 object code를 linked list로 연결하기 위한 모듈.

* + 1. *사용변수*

Ob \*node – 계산된 OBJECT CODE의 data를 복사할 linked list node

* 1. **모듈이름: allocate(Sym \*data)**
     1. *기능*

SYMBOL 명령어 수행 시 symbol table에 저장된 node들을 내림차순으로 정렬해야 하므로 정렬된 상태의 linked list를 만들기 위한 기존 node의 data를 새로운 node에 복사해주는 모듈

* + 1. *사용변수*

Sym \*node – 새로 메모리를 할당하여 복사한 정보를 담을 node

* 1. **모듈이름: symbol(Sym \*\*symhead, int symbol\_print\_flag)**
     1. *기능*

symbol 명령어 입력 시 가장 최근에 assemble에 성공한 소스파일의 symbol table 을 탐색하여 symbol을 기준으로 내림차순으로 정렬하여 출력하는 모듈, 가장 최근에 assemble한 파일이 없거나 assemble에 실패한 경우 symbol table이 비어있다고 출력.

* + 1. *사용변수*

Sym\* node – symbol table 각 행의 head를 가리킬 포인터

Sym\* list – 내림차순으로 정렬된 node가 연결된 linked lsit의 head 포인터

Sym\* ptr – 내림차순으로 정렬할 때 node를 삽입할 위치를 탐색할 포인터

Sym\* follow – 내림차순으로 정렬할 때 node를 삽입할 위치를 가리킬 포인터

Sym\* temp – 내림차순으로 정렬된 list에 삽입하기 위해 기존 node의 데이터가 복사된 구조체

1. **전역 변수 정의**
   1. **Node \*head**

history에 저장할 linked list의 head의 위치를 가리킬 포인터

* 1. **Op \*ophead[20]**

opcode.txt 파일을 읽어 opcodelist를담을 20개의 사이즈 hash table head 포인터

* 1. **unsigned char MEMORY[MEMORY\_SIZE]**

shell 내에서 사용하는 1MB의 MEMORY의 값을 담을 변수

* 1. **unsigned int address**

dump 명령어에서 start, end 주소 모두가 주어진 경우를 제외하고 출력을 시작할 위치와 출력을 끝낸 후 다음 명령어 수행 시 출력시작 위치를 저장할 변수

1. **코드**

**5.1**  **20171601.h**

/\*정의되는 상수\*/

#define INPUT\_SIZE 500

#define MEMORY\_SIZE 0x100000

/\*history에 사용될 linked list node\*/

typedef struct \_Node {

char a[INPUT\_SIZE];

struct \_Node \*list;

}Node;

typedef struct \_Op {

int num;

char inst[10];

char type[5];

struct \_Op \*list;

}Op;

typedef struct \_Ob{

int first, mid, end, location;

struct \_Ob \*list;

int format, enter\_flag;

char var[255];

}Ob;

typedef struct \_Sym{

int location;

char name[10];

struct \_Sym \*list;

}Sym;

/\*함수 원형\*/

void symbol(Sym \*\*symhead, int flag);

Sym \*allocate(Sym \*data);

int check\_mnemonic(char \*mnemonic, char \*format, int \*opcode, Op \*\*ophead);

int build\_symtab(Sym \*\*symhead, Sym \*node, int line\_num);

int find\_symbol(Sym \*\*symhead, char \*symbol);

void location\_counter(char \*format, unsigned int \*location, char \*mnemonic);

void objectcode\_insert(int format, int first, int mid, int end, int location, Ob\*\* head, Ob\*\* tail, char \*var, int enter\_flag);

int reg\_num(char \*reg);

void type(char \*origin, char \*\*command);

void assemble(char \*origin,char \*\*command, Op \*\*ophead, Sym \*\*\*output, int \*symbol\_print\_flag);

void symbol();

void opcodelist();

void mnemonic(char \*origin,char \*\*command);

void create\_opcode();

void insert(Op \*node);

void check1(char \*origin,char \*\*command,Sym \*\*\*symhead, int \*symbol\_print\_flag);

void check2(char \*origin,char \*\*command, int TOKEN\_SIZE, Sym \*\*\*symhead, int \*symbol\_print\_flag);

void quit();

void record(char \*origin,char \*\*command, int TOKEN\_SIZE);

void help();

void history();

void dir();

void dump(unsigned int start, unsigned int end);

void edit(unsigned int origin, unsigned int new);

void fill(unsigned int start, unsigned int end, unsigned int new);

void reset();

**5.2**  **20171601.c : 이번 프로젝트에서 새롭게 추가한 함수들만 기록함**

/\*포함되는 파일\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <sys/stat.h>

#include "20171601.h"

/\*전역 변수\*/

Node \*head;

Op \*ophead[20];

unsigned char MEMORY[MEMORY\_SIZE];

unsigned int address;

/\*프로그램 시작\*/

void main()

{

create\_opcode(); // opcode 생성

char command[INPUT\_SIZE] = { "\0" }; // 입력받는 명령어

char command\_temp[INPUT\_SIZE] = { "\0" }; // 명렁어를 나누기 위한 임의의 배열

char \*\*token; // 나눈 명령들 저장

char \*ptr; // 나눈 덩어리 저장을 위한 볏누

Sym \*\*symhead = NULL;

int flag = 0; // flag = 1일 경우 종료

int TOKEN\_SIZE; // token 덩어리의 갯수

int i, symbol\_print\_flag = 0;

while (!flag)

{

TOKEN\_SIZE = 1;

printf("sicsim> ");

fgets(command, sizeof(command), stdin);

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

strcpy(command\_temp, command);

//TOKEN의 갯수 세기

ptr = strtok(command\_temp, " \t\n");

if (!ptr) continue;

while (1)

{

ptr = strtok(NULL, " \t\n");

if (!ptr) break;

TOKEN\_SIZE++;

}

token = (char \*\*)malloc(sizeof(char \*)\*(TOKEN\_SIZE + 1));

//TOKEN 분리

strcpy(command\_temp, command);

ptr = strtok(command\_temp, " \t\n");

if (!ptr) continue;

token[0] = (char \*)malloc(sizeof(char) \* (INPUT\_SIZE));

strcpy(token[0], ptr);

for (int i = 1; i < TOKEN\_SIZE; i++) {

ptr = strtok(NULL, " \t\n");

if (ptr == NULL) break;

token[i] = (char \*)malloc(sizeof(char) \* (INPUT\_SIZE));

strcpy(token[i], ptr);

}

if (strcmp(\*token, "quit") == 0 || strcmp(\*token, "q") == 0) // command가 quit일 경우

{

quit(); // 메모리 해제

flag = 1; // 종료

}

else

{

if (TOKEN\_SIZE == 1) // token의 갯수가 1개라면

check1(command,token,&symhead,&symbol\_print\_flag); // check1() 함수로 이동

else if (TOKEN\_SIZE >= 2) // token의 갯수가 2개 이상이라면

check2(command,token, TOKEN\_SIZE, &symhead,&symbol\_print\_flag); // check2() 함수로 이동

}

//메모리 해제

for (i = 0; i < TOKEN\_SIZE; i++)

{

free(token[i]);

token[i] = NULL;

}

}

}

/\*------------------------------------------------------------------------------------\*/

/\*함수 : check1(char \*origin,char \*\*command, Sym \*\*\*symhead, int \*symbol\_print\_flag)

/\*목적 : 입력받은 명령어를 인식하여 해당 명렁을 수행한다.

/\*리턴값 : 없음

/\*------------------------------------------------------------------------------------\*/

//Token의 갯수가 1개인 경우

void check1(char \*origin,char \*\*command, Sym \*\*\*symhead, int \*symbol\_print\_flag)

{

unsigned int start, end = 0;

// command가 help인 경우

if (strcmp(\*command, "help") == 0 || strcmp(\*command, "h") == 0)

{

record(origin,command, 1); // history에 기록

help();

}

// command가 dir인 경우

else if (strcmp(\*command, "dir") == 0 || strcmp(\*command, "d") == 0)

{

record(origin,command, 1);

dir();

}

//command가 history인 경우

else if (strcmp(\*command, "history") == 0 || strcmp(\*command, "hi") == 0)

{

record(origin,command, 1);

history();

}

//dump : case 1(no argumennt)

else if (strcmp(\*command, "dump") == 0 || strcmp(\*command, "du") == 0)

{

record(origin,command, 1);

//(last address + 1)+159가 범위를 초과할 경우

if ((address + 160) > 0xFFFFF)

{

//last address=를 0xfffff로 제한 & 다음 주소는 0x00000로지정

start = address;

end = 0xFFFFF;

address = 0;

}

// 범위를 초가하지 않는 경우 start, end address 지정

else

{

start = address;

end = address + 159;

address = end + 1;

}

dump(start, end);

}

//command가 reset인 경우

else if (strcmp(\*command, "reset") == 0)

{

record(origin,command, 1);

reset();

}

// command가 opcodelist인 경우

else if (strcmp(\*command, "opcodelist") == 0)

{

record(origin,command, 1);

opcodelist();

}

else if (strcmp(\*command, "symbol") == 0)

{

record(origin, command, 1);

symbol(\*symhead, \*symbol\_print\_flag);

}

// 그 외긔 명령어가 들어오면 에러 출력

else printf("Error : Invalid Parameter!!\n");

}

/\*------------------------------------------------------------------------------------\*/

/\*함수 : check2(char \*origin,char \*\*command, int TOKEN\_SIZE, Sym \*\*\*symhead,int \*symbol\_print\_flag)

/\*목적 : 입력받은 명령어를 인식하여 해당 명렁을 수행한다.

/\*리턴값 : 없음

/\*------------------------------------------------------------------------------------\*/

//command token이 2개 이상인 경우

void check2(char \*origin,char \*\*command, int TOKEN\_SIZE, Sym \*\*\*symhead,int \*symbol\_print\_flag)

{

unsigned int start, end, addr, new = 0,trash = 0;

int i, count = 0;

if ((strcmp(\*command, "dump") == 0 || strcmp(\*command, "du") == 0) && (TOKEN\_SIZE <=4))

{

// dump : case 1(1 argument)

if ((TOKEN\_SIZE == 2) && (strchr(command[1],',') == NULL))

{

// start parameter 조건

for(i= 0; i < strlen(command[1]); i++)

{

if((command[1][i] >= '0' && command[1][i] <= '9') ||(command[1][i] >= 'a' && command[1][i] <= 'f') ||

(command[1][i] >= 'A' && command[1][i] <= 'F') ||command[1][i] == ','|| (command[1][0] == '0' && strtol(command[1],NULL,16) >= 0x00000 && strtol(command[1],NULL,16) <= 0xfffff))

continue;

printf("Error : Invalid Parameter!!\n"); // dump XYZ

return;

}

//조건 만족하면 변수에 저장

start = strtol(command[1],NULL,16);

// Boundary check

if(start <= 0xfffff && start >= 0x00000)

{

if((start + 160) > 0xfffff)

{

end = 0xfffff;

address = 0;

}

else

{

end = start + 159;

address = end +1;

}

dump(start, end);

record(origin,command, 2);

}

else printf("Error : Invalid Parameter!!\n");

}

// token이 2개 이상일 경우

else

{

for(i = 1; i < TOKEN\_SIZE - 1; i++)

{

strcat(command[1], " ");

strcat(command[1], command[i + 1]);

}

for(i = 0; i < strlen(command[1]); i++)

if(command[1][i] == ',') count++;

//컴마의 갯수가 2개 이상이면 에러메세지 출력

if(count >= 2) printf("Error : Invalid Parameter!!\n");

else

{

i = sscanf(command[1], "%x , %x %x", &start, &end,&trash);

// start와 end를 읽었다면

if(i == 2)

{

// Parameter Check

for(i = 0; i < strlen(command[1]); i++)

{

if((command[1][i] >= '0' && command[1][i] <= '9') || (command[1][i] >= 'a' && command[1][i] <= 'f') ||

(command[1][i] >= 'A' && command[1][i] <= 'F') ||command[1][i] == ',' || command[1][i] == ' '||(start>=0x00000 &&start<=0xfffff && end<=0xfffff && end >= 0x00000))

continue;

printf("Error : Invalid Parameter!!\n");

return;

}

// Boundary Check

if(start <= end)

{

if((start >= 0 &&start <= 0xfffff) && (end >= 0 && end <= 0xfffff))

{

dump(start, end);

record(origin,command, 2);

if(end == 0xfffff) address = 0;

else address = end + 1;

}

else printf("Error : Invalid Parameter!!\n");

}

else printf("Error : Invalid Parameter!!\n");

}

else printf("Error : Invalid Parameter!!\n");

}

}

}

// command가 edit인 경우

else if (strcmp(\*command, "edit") == 0 || strcmp(\*command, "e") == 0)

{

//Parameter Check

if (TOKEN\_SIZE == 2 && strchr(command[1],',') == NULL)

{

printf("Error : Invalid Parameter!!\n");

return;

}

else

{

for(i = 1; i < TOKEN\_SIZE - 1; i++)

{

strcat(command[1], " ");

strcat(command[1], command[i + 1]);

}

for(i = 0; i < strlen(command[1]); i++)

if(command[1][i] == ',') count++;

// 컴마 갯수 Check

if(count >= 2) printf("Error : Invalid Parameter!!\n");

else

{

i = sscanf(command[1], "%x , %x %x", &addr, &new,&trash);

if(i == 2)

{

//Parameter Chech

for(i = 0; i < strlen(command[1]); i++)

{

if((command[1][i] >= '0' && command[1][i] <= '9') || (command[1][i] >= 'a' && command[1][i] <= 'f') ||

(command[1][i] >= 'A' && command[1][i] <= 'F') ||command[1][i] == ',' || command[1][i] == ' ' || (addr >= 0x00000 && addr<=0xfffff))

continue;

printf("Error : Invalid Parameter!!\n");

return;

}

// Boundary Check

if(addr >= 0x00000 && addr <= 0xFFFFF)

{

if(new >= 0x00 && new <= 0xFF)

{

edit(addr, new);

record(origin,command, TOKEN\_SIZE);

}

else printf("Error : Invalid Parameter!!\n");

}

else printf("Error : Invalid Parameter!!\n");

}

else printf("Error : Invalid Parameter!!\n");

}

}

}

// command가 fill인 경우

else if (strcmp(\*command, "fill") == 0 || strcmp(\*command, "f") == 0)

{

for(i = 1; i < TOKEN\_SIZE - 1; i++)

{

strcat(command[1], " ");

strcat(command[1], command[i + 1]);

}

for(i = 0; i < strlen(command[1]); i++)

if(command[1][i] == ',') count++;

if(count != 2) printf("Error : Invalid Parameter!!\n");

else

{

i = sscanf(command[1], "%x , %x , %x %x", &start, &end,&new,&trash);

if(i == 3)

{

for(i = 0; i < strlen(command[1]); i++)

{

if((command[1][i] >= '0' && command[1][i] <= '9') || (command[1][i] >= 'a' && command[1][i] <= 'f') ||

(command[1][i] >= 'A' && command[1][i] <= 'F') ||command[1][i] == ',' || command[1][i] == ' ' || (start >= 0x00000 && start <= 0xfffff && end >=0x00000 && end <= 0xfffff))

continue;

printf("Error : Invalid Parameter!!\n");

return;

}

if((start <= end)&&(start >= 0x00000 && start <= 0xFFFFF)&&((end >= 0x00000 && end <= 0xFFFFF)))

{

if(new >= 0x00 && new <= 0xFF)

{

fill(start, end, new);

record(origin,command, TOKEN\_SIZE);

}

else printf("Error : Invalid Parameter!!\n");

}

else printf("Error : Invalid Parameter!!\n");

}

else printf("Error : Invalid Parameter!!\n");

}

}

else if (strcmp(\*command, "opcode") == 0)

mnemonic(origin,command);

else if (strcmp(\*command, "type") == 0)

{

if(TOKEN\_SIZE == 2)

type(origin, command);

else

{

printf("Error : Invalid Parameter!!\n");

return;

}

}

else if (strcmp(\*command, "assemble") == 0)

{

if(TOKEN\_SIZE == 2)

assemble(origin, command, ophead, symhead, symbol\_print\_flag);

else

{

printf("Error : Invalid Parameter!!\n");

return;

}

}

//그 외 명령어가 들어올 경우 에러메세지 출력

else printf("Error : Invalid Parameter!!\n");

}

/\*------------------------------------------------------------------------------------\*/

/\*함수 : symbol(Sym \*\*symhead, int flag)

/\*목적 : 가장 최근에 assemble에 성공한 소스파일의 symbol table 을 탐색하여 symbol을 기준으로 내림차순으로 정렬하여 출력한다.

/\*리턴값 : 없음

/\*------------------------------------------------------------------------------------\*/

void symbol(Sym \*\*symhead, int flag)

{

Sym \*node = NULL, \*list = NULL, \*ptr = NULL, \*follow = NULL, \*temp;

int i;

if(!flag)

{

printf("SYMTAB IS EMPTY\n");

return;

}

//assemble한 적이 없거나 최근에 assemble한 file이 실패한 file일 경우

//symtab이 존재하지 않으므로 비어있다고 알림

//symtab이 존재하는 경우 내림차순으로 symbol 정렬

for(i = 0; i < 100; i++)

{

node = symhead[i];

//내림차순이기 때문에 ASCII값을 비교하는 반복문

while(node)

{

ptr = follow = list;

while(ptr && strcmp(ptr->name, node->name) > 0)

{

//strcmp값이 양수이면 다음값과 비교

follow = ptr;

ptr = ptr->list;

}

//strcmp값이 음수이면 그 전 노드에 연결

temp = allocate(node);

temp->list = ptr;

if(ptr == list)

{

temp = allocate(node);

temp->list = ptr;

list = temp;

}

else

follow->list = temp;

node = node->list;

}

ptr = list;

}

//내림차순으로 정렬된 SYMBOL 출력

while(ptr)

{

printf("\t%s\t%04X", ptr->name, ptr->location);

ptr = ptr->list;

printf("\n");

}

}

/\*------------------------------------------------------------------------------------\*/

/\*함수 : allocate(Sym\* data)

/\*목적 : symbol을 내림차순으로 정렬할 때 새로운 node에 메모리를 할당하는 함수

/\*리턴값 : 메모리가 할당된 노드

/\*------------------------------------------------------------------------------------\*/

Sym\* allocate(Sym\* data)

{

Sym \*node = (Sym\*)malloc(sizeof(Sym));

node->location = data->location;

strcpy(node->name, data->name);

node->list = data->list;

return node;

}

/\*------------------------------------------------------------------------------------\*/

/\*함수 : assemble(char \*origin,char \*\*command, Op \*\*ophead, Sym \*\*\*output, int \*symbol\_print\_flag)

/\*목적 : 사용자가 입력한 filename에 대하여 현재 디렉토리 내에서 찾아서 assemble하여 listing file과 object file을 생성한다.

/\*리턴값 : 없음

/\*------------------------------------------------------------------------------------\*/

void assemble(char \*origin, char \*\*command, Op \*\*ophead, Sym \*\*\*output, int \*symbol\_print\_flag)

{

FILE \*fp = fopen(command[1], "rt");

FILE \*lst, \*obj, \*mid;

char temp[INPUT\_SIZE];

char \*lstname = (char\*)malloc(sizeof(char) \* strlen(command[1]));

char \*objname = (char\*)malloc(sizeof(char) \* strlen(command[1]));

char operator[10], operand1[255], operand2[10], format[5], optemp[10], name[10];

int i, opcode, reg1, reg2, num, disp, pc = 0, base = 0, base\_flag = 0, var;

int first\_data = 0, mid\_data = 0, symbol\_flag = 0, end\_flag = 0, start\_flag = 0, error\_flag = 0;

unsigned int line\_num = 5, location = 0, length;

Sym \*\*symhead = (Sym\*\*)malloc(sizeof(Sym\*) \* 100), \*node = (Sym\*)malloc(sizeof(Sym));

Ob \*head = NULL, \*tail = NULL, \*ptr1, \*ptr2;

if(!fp)

{

printf("There is no %s file\n", command[1]);

return;

}

//없는 파일 load시 ERROR

if(strcmp(command[1] + strlen(command[1]) - 4, ".asm"))

{

printf("%s file is not assemble file\n", command[1]);

return;

}

//.asm 파일 아닌 파일 load시 ERROR

for(int i=0;i<100;i++)

symhead[i] = NULL;

mid = fopen("mid.txt","wt");

\*symbol\_print\_flag = 0;

while(fgets(temp, sizeof(temp),fp))

{

if(error\_flag) return;

strcpy(operand1,"");

strcpy(operand2,"");

strcpy(operator,"");

strcpy(format,"");

strcpy(node->name,"");

strcpy(optemp,"");

if(temp[0] == '.')

{

fprintf(mid, "=%4d\t%s",line\_num, temp);

line\_num += 5;

continue;

}

else if(temp[0] == '\n') continue;

else if(temp[0] == ' ' || temp[0] == '\t')

{

num = sscanf(temp, "%s %s %s",operator, operand1, operand2);

if(num == 2 && strchr(operand1,','))

{

strcpy(operand2, strchr(operand1, ',')+1);

for(i=0; i<strlen(operand1);i++)

{

if(operand1[i] == ',')

{

operand1[i+1]='\0';

break;

}

num = 3;

}

}

}

//SYMBOL 존재O

else

{

num = sscanf(temp, "%s %s %s %s", node->name , operator, operand1, operand2);

if(!strcmp(operator, "BYTE") && operand1[0] == 'C' && num == 4)

{

sscanf(temp, "%s %s %[^\n]s", node->name, operator, operand1);

num = 3;

}

if(num == 3 && strchr(operand1, ','))

{

strcpy(operand2, strchr(operand1, ',') + 1);

for(i = 0; i < strlen(operand1); i++)

if(operand1[i] == ',')

{

operand1[i + 1] = '\0';

break;

}

num = 4;

}

}

//directive 명령어 + RSUB + BASE + NOBASE 처리

if(!strcmp(operator, "START"))

{

fprintf(mid, "=%4d\t%04X\t%-10s\t%-10s\t0\nS%s\n", line\_num, location, node->name, operator,node->name);

line\_num += 5;

location = 0;

node-> location = location;

error\_flag = build\_symtab(symhead, node, line\_num);

continue;

}

else if(!strcmp(operator, "END"))

{

fprintf(mid, "=%4d\t \t%-10s\t%-10s\t%-20s\nE\n", line\_num, node->name, operator, operand1);

line\_num += 5;

break;

}

else if(!strcmp(operator, "BYTE"))

{

node->location = location;

if(operand1[0] == 'X')

{

if((strlen(operand1) - 3) % 2)

{

printf("LINE : %d HEXA DATA ERROR\n", line\_num);

return;

//16진수 data인데 홀수개만 선언한 경우 ERROR

}

fprintf(mid, "%4d\t%04X\t%-10s\t%-10s\t%-20s\n!", line\_num, location, node->name, operator, operand1);

for(i = 2; i < strlen(operand1) - 1; i++)

{

if(operand1[i] >= '0' && operand1[i] <= '9');

else if(operand1[i] >= 'A' && operand1[i] <= 'F');

else if(operand1[i] >= 'a' && operand1[i] <= 'f');

else

{

printf("LINE : %d HEXA DATA ERROR\n", line\_num);

return;

}

fprintf(mid, "%c", operand1[i]);

}

fprintf(mid, "\n");

error\_flag = build\_symtab(symhead, node, line\_num);

node->location = location = location + (strlen(operand1) - 3) / 2;

}

else if(operand1[0] == 'C')

{

fprintf(mid, "%4d\t%04X\t%-10s\t%-10s\t%-20s\n!", line\_num, location, node->name, operator, operand1);

for(i = 2; i < strlen(operand1) - 1; i++) fprintf(mid, "%X", operand1[i]);

fprintf(mid, "\n");

error\_flag = build\_symtab(symhead, node, line\_num);

node->location = location = location + strlen(operand1) - 3;

}

line\_num += 5;

continue;

}

else if(!strcmp(operator, "WORD"))

{

fprintf(mid, "%4d\t%04X\t%-10s\t%-10s\t%-20s\n!", line\_num, location, node->name, operator, operand1);

sscanf(operand1, "%d", &var);

if(var >= -0x800000 && var <= 0x7FFFFF)

if(var <= -1 && var >= -0x800000)

var += 0x1000000;

else

{

printf("LINE : %d WORD value overflow!\n", line\_num);

return;

}

//WORD value overflow처리

fprintf(mid, "%06X\n", var);

error\_flag = build\_symtab(symhead, node, line\_num);

node->location = location = location + 3;

line\_num += 5;

continue;

}

else if(!strcmp(operator, "RESW"))

{

sscanf(operand1, "%d", &var);

fprintf(mid, "=%4d\t%04X\t%-10s\t%-10s\t%-20s\n~\n", line\_num, location, node->name, operator, operand1);

error\_flag = build\_symtab(symhead, node, line\_num);

node->location = location = location + 3 \* var;

line\_num += 5;

continue;

}

else if(!strcmp(operator, "RESB"))

{

sscanf(operand1, "%d", &var);

fprintf(mid, "=%4d\t%04X\t%-10s\t%-10s\t%-20s\n~\n", line\_num, location, node->name, operator, operand1);

error\_flag = build\_symtab(symhead, node, line\_num);

node->location = location = location + var;

line\_num += 5;

continue;

}

else if(!strcmp(operator, "BASE"))

{

fprintf(mid, "=%4d\t \t%-10s\t%-10s\t%-20s\nB%s\n", line\_num, node->name, operator, operand1, operand1);

line\_num += 5;

continue;

}

else if(!strcmp(operator, "NOBASE"))

{

fprintf(mid, "=%4d\t \t%-10s\t%-10s\t\nN\n", line\_num, node->name, operator);

line\_num += 5;

continue;

}

//directive 이외에 opcode table에 없는 잘못된 mnemonic 입력시 ERROR

else

{

if(!check\_mnemonic(operator, format, &opcode, ophead))

{

printf("LINE : %d %s is not in OPCODE TABLE\n", line\_num, operator);

return;

}

}

//directive 이외 opcode table에 있는 mnemonic일 경우 정상 처리

node->location = location;

if(strcmp(node->name, ""))

error\_flag = build\_symtab(symhead, node, line\_num);

location\_counter(format, &location, operator);

strcpy(optemp, operand1);

if(strcmp(operand2, ""))

{

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

strcat(optemp, "\t");

strcat(optemp, operand2);

strcat(operand1, " ");

strcat(operand1, operand2);

}

//intermediate 파일에 필요한 자료 모두 입력

fprintf(mid, "%4d\t%04X\t%-10s\t%-10s\t%-20s\n\t%02X\t%2s\t%10s\t%10s\n", line\_num, node->location, node->name, operator, operand1, opcode, format, operator, optemp);

line\_num += 5;

}

length = location;

fclose(mid);

strcpy(lstname, command[1]);

lstname[strlen(command[1]) - 4] = '\0';

lst = fopen(strcat(lstname, ".lst"), "wt");

////pass2////

mid = fopen("mid.txt", "rt");

while(fgets(temp, sizeof(temp), mid))

{

//엔터있을 경우 출력양식 맞추기 위해 제거

if(temp[strlen(temp) - 1] == '\n') temp[strlen(temp) - 1 ] = '\0';

//첫문자가 ! : BYTE, WORD

if(temp[0] == '!')

{

fprintf(lst, "%s\n", temp + 1);

objectcode\_insert(5, 0, 0, 0, location, &head, &tail, temp + 1, 0);

}

//첫문자가 S : START

else if(temp[0] == 'S')

{

strcpy(name, temp + 1);

start\_flag = 1;

}

//첫문자가 = : objectcode 계산 불필요 따라서 바로 출력

else if(temp[0] == '=') fprintf(lst, "%s\n", temp + 1);

//첫문자가 N : NOBASE

else if(temp[0] == 'N') base\_flag = 0;

//첫문자가 B : BASE

else if(temp[0] == 'B') base\_flag = 1;

//첫문자가 E : END

else if(temp[0] == 'E') end\_flag = 1;

//첫문자가 ~ : RESB, RESW

else if(temp[0] == '~') tail->enter\_flag = 1; //링크드리스트에서 개행해야된다고 알려주기

//첫문자가 탭 : obectcode 계산

else if(temp[0] == '\t')

{

num = sscanf(temp, " %x %s %s %s %s ", &opcode, format, operator, operand1, operand2);

//operand가 없는 경우 : RSUB, 1형식

if(num == 3)

{

//RSUB

if(!strcmp(operator, "RSUB"))

{

fprintf(lst, "%-10s\n", "4F0000");

objectcode\_insert(3, 0x4f, 0, 0, location, &head, &tail, "", 0);

}

//1형식

else if(!strcmp(format, "1"))

{

pc = location + 1;

fprintf(lst, "%02X\n", opcode);

objectcode\_insert(1, opcode, 0, 0, location, &head, &tail, "", 0);

}

//이외 경우 ERROR

else

{

printf("LINE : %d OPERAND ERROR\n", line\_num);

fclose(lst);

remove(lstname);

return;

}

}

//operand가 있는 경우 : 2, 3, 4형식

else if(num >= 4)

{

//2형식

if(!strcmp(format, "2"))

{

pc = location + 2;

//operand가 하나인 2형식

if(num == 4)

{

if(strcmp(operator, "TIXR") && strcmp(operator, "CLEAR") && strcmp(operator, "SVC"))

{

printf("LINE : %d FORMAT2 OPERAND ERROR\n", line\_num);

fclose(lst);

remove(lstname);

return;

}

if(!strcmp(operator, "SVC")) sscanf(operand1, "%d", &reg1);

else reg1 = reg\_num(operand1);

if(reg1 == -1)

{

printf("LINE : %d REGISTER ERROR\n", line\_num);

fclose(lst);

remove(lstname);

return;

}

fprintf(lst, "%02X%01X0\n", opcode, reg1);

objectcode\_insert(2, opcode, reg1\*16, 0, location, &head, &tail, "", 0);

}

//operand가 두개인 2형식

else if(num == 5)

{

if(!strcmp(operator, "TIXR") || !strcmp(operator, "CLEAR") || !strcmp(operator, "SVC"))

{

printf("LINE : %d FORMAT2 OPERAND ERROR\n", line\_num);

fclose(lst);

remove(lstname);

return;

}

if(!strcmp(operator, "SHIFTL") || !strcmp(operator, "SHIFTR"))

{

reg1 = reg\_num(operand1);

sscanf(operand2, "%d", &reg2);

}

else

{

reg1 = reg\_num(operand1);

reg2 = reg\_num(operand2);

}

if(reg1 == -1 || reg2 == -1)

{

printf("LINE : %d REGISTER ERROR\n", line\_num);

fclose(lst);

remove(lstname);

return;

}

fprintf(lst, "%02X%01X%01X\n", opcode, reg1, reg2);

objectcode\_insert(2, opcode, reg1\*16 + reg2, 0, location, &head, &tail, "", 0);

}

}

//3형식, 4형식

else if(!strcmp(format, "3") || !strcmp(format, "4"))

{

if(!strcmp(format, "3")) pc = location + 3;

else if(!strcmp(format, "4")) pc = location + 4;

//immediate addressing

if(operand1[0] == '#')

{

first\_data = opcode + 1;

strcpy(operand1, operand1 + 1);

}

//indirect addressing

else if(operand1[0] == '@')

{

first\_data = opcode + 2;

strcpy(operand1, operand1 + 1);

}

//simple addressing

else first\_data = opcode + 3;

//operand symbol location값 계산

symbol\_flag = find\_symbol(symhead, operand1);

//잘못된 operand symbol일 경우 ERROR

if(symbol\_flag == -1)

{

printf("LINE : %d SYMBOL ERROR\n", line\_num);

fclose(lst);

remove(lstname);

return;

}

//operand symbol이 상수인경우

else if(symbol\_flag == -2)

{

sscanf(operand1, "%d", &disp);

//4형식 overflow 체크

if(!strcmp(format, "4"))

{

if(disp < 0 || disp >= 0x100000)

{

printf("LINE : %d disp overflow\n", line\_num);

fclose(lst);

remove(lstname);

return;

}

mid\_data = 1;

}

//3형식 overflow 체크

else if(!strcmp(format, "3"))

{

if(disp < 0 || disp >= 0x1000)

{

printf("LINE : %d disp overflow\n", line\_num);

fclose(lst);

remove(lstname);

return;

}

}

}

//operand symbol 상수가 아니고 정상 확인된 경우

else

{

//4형식

if(!strcmp(format, "4"))

{

disp = symbol\_flag;

mid\_data = 1;

}

//3형식

// PC -> BASE relative overflow check

else if(!strcmp(format, "3"))

{

disp = symbol\_flag - pc;

if(disp >= -0x800 && disp <= 0x7FF)

{

if(disp >= -0x800 && disp <= -1)

{

disp += 0x1000;

}

mid\_data = 2;

}

else

{

if(!base\_flag)

{

printf("LINE : %d BASE FLAG IS FALSE\n",line\_num);

fclose(lst);

remove(lstname);

return;

}

symbol\_flag = find\_symbol(symhead, operand1);

disp = symbol\_flag - base;

if(disp >= 0 && disp <= 0xFFF) mid\_data = 4;

else

{

printf("LINE : %d disp overflow\n", line\_num);

fclose(lst);

remove(lstname);

return;

}

}

}

}

//indexed mode

if(num == 5)

{

if(!strcmp(operand2, "X") || !strcmp(operand2, "x")) mid\_data += 8;

else

{

printf("LINE : %d %s must be X register\n", line\_num, operand2);

fclose(lst);

remove(lstname);

return;

}

}

//3형식 출력

if(!strcmp(format, "3"))

{

fprintf(lst, "%02X%01X%03X\n", first\_data, mid\_data, disp);

if(symbol\_flag == -2) objectcode\_insert(3, first\_data, mid\_data, disp, location, &head, &tail, "#", 0);

else objectcode\_insert(3, first\_data, mid\_data, disp, location, &head, &tail, "", 0);

}

//4형식 출력

else if(!strcmp(format, "4"))

{

fprintf(lst, "%02X%01X%05X\n", first\_data, mid\_data, disp);

if(symbol\_flag == -2) objectcode\_insert(4, first\_data, mid\_data, disp, location, &head, &tail, "", 0);

else objectcode\_insert(4, first\_data, mid\_data, disp, location, &head, &tail, "#", 0);

}

first\_data = mid\_data = 0;

}

else

{

printf("LINE : %d FORMAT ERROR\n", line\_num);

fclose(lst);

remove(lstname);

return;

}

//LDB인 경우 B register값 저장 -> 차후 BASE 명령어 입력시 relative mode 활성화 위해

if(strstr(operator, "LDB"))

base = symbol\_flag;

}

}

// 나머지 : objectcode 계산이 필요

else

{

sscanf(temp, "%d %x", &line\_num, &location);

fprintf(lst, "%s", temp);

}

}

//파일 내에 START가 없을 시 ERROR

if(!start\_flag)

{

printf("START NOT FOUND\n");

fclose(lst);

remove(lstname);

return;

}

//파일 내에 END가 없을 시 ERROR

if(!end\_flag)

{

printf("END NOT FOUND\n");

fclose(lst);

remove(lstname);

return;

}

//OBJECT FILE 작성

ptr1 = ptr2 = head;

strcpy(objname, command[1]);

objname[strlen(command[1]) - 4] = '\0';

obj = fopen(strcat(objname, ".obj"), "wt");

//HEADER RECORD

fprintf(obj, "H%-6s000000%06X\n", name, length);

length = 0;

//TEXT RECORD

while(ptr1)

{

fprintf(obj, "T%06X", ptr1->location);

//한 LINE출력을 위해 LENGTH 계산

while(ptr2)

{

if(ptr2->format == 5)

{

if(length + strlen(ptr2->var) / 2 > 30) break;

else

{

length = length + strlen(ptr2->var) / 2;

if(ptr2->enter\_flag)

{

ptr2 = ptr2->list;

break;

}

}

}

else

{

if(length + ptr2->format > 30) break;

else

{

length = length + ptr2->format;

if(ptr2->enter\_flag)

{

ptr2 = ptr2->list;

break;

}

}

}

ptr2 = ptr2->list;

}

//한 LINE 계산 완료시

//출력

fprintf(obj, "%02X", length);

while(ptr1 != ptr2)

{

if(ptr1->format == 1) fprintf(obj, "%02X", ptr1->first);

else if(ptr1->format == 2) fprintf(obj, "%02X%02X", ptr1->first, ptr1->mid);

else if(ptr1->format == 3) fprintf(obj, "%02X%01X%03X", ptr1->first, ptr1->mid, ptr1->end);

else if(ptr1->format == 4) fprintf(obj, "%02X%01X%05X", ptr1->first, ptr1->mid, ptr1->end);

else if(ptr1->format == 5) fprintf(obj, "%s", ptr1->var);

ptr1 = ptr1->list;

}

fprintf(obj, "\n");

length = 0;

}

ptr1 = head;

//MODIFICATION RECORD

while(ptr1)

{

if(ptr1->format == 4 && strcmp(ptr1->var, "")) fprintf(obj, "M%06X05\n",ptr1->location + 1);

ptr1 = ptr1-> list;

}

//END RECORD

fprintf(obj, "E000000\n");

// "assemble" command 정상 수행 종료 알림

record(origin, command, 2);

printf("output file : [%s], [%s]\n", lstname, objname);

free(lstname);

free(objname);

lstname = objname = NULL;

fclose(mid);

fclose(fp);

fclose(lst);

fclose(obj);

\*symbol\_print\_flag = 1;

\*output = symhead;

remove("mid.txt");

return;

}

/\*------------------------------------------------------------------------------------\*/

/\*함수 : check\_mnemonic(char \*mnemonic, char \*format, int \*opcode, Op \*\*ophead)

/\*목적 : assemble command 수행 시 나타나는 mnemonic의 opcode를 반환하는 함수

/\*리턴값 : opcode

/\*------------------------------------------------------------------------------------\*/

int check\_mnemonic(char \*mnemonic, char \*format, int \*opcode, Op \*\*ophead)

{

Op \*temp;

char data[10];

int index, sum = 0;

if(mnemonic[0] == '+') strcpy(data, mnemonic + 1);

//4형식일 경우 +문자 지우고 mnemonic만 받기

else strcpy(data, mnemonic);

//아닐 경우 그대로 진행

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

sum += data[i]; // 모든 아스키 값 더하기

index = sum % 20; // key값 정의

temp = ophead[index];

while(temp){

if(!strcmp(temp->inst, data)){

\*opcode = temp->num;

strcpy(format, temp->type);

return 1;

//중도 발견시 성공 return value 1

}

temp = temp->list;

}

return 0;

//실패시 실패 return value 0

}

/\*------------------------------------------------------------------------------------\*/

/\*함수 : build\_symtab(Sym\*\* symhead, Sym\* node, int line\_num)

/\*목적 : assemble 과정에서 나타나는 symbol들을 저장하기 위한 symbol table build 함수 /\*리턴값 : 없음

/\*------------------------------------------------------------------------------------\*/

int build\_symtab(Sym\*\* symhead, Sym\* node, int line\_num)

{

int sum =0;

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

sum += node->name[i]; // 모든 아스키 값 더하기

int index = sum % 100; // key값 정의

Sym \*temp, \*ptr;

if(node->name[0] >= '0' && node->name[0] <= '9')

{

printf("LINE : %d SYMBOL NAME ERROR\n", line\_num);

return 1;

}

//SYMBOL의 첫글자가 숫자인 경우 ERROR

if(!symhead[index])

{

symhead[index] = (Sym\*)malloc(sizeof(Sym));

strcpy(symhead[index]->name, node->name);

symhead[index]->location = node->location;

symhead[index]->list = NULL;

}

//head가 비어있다면 head에 연결

else

{

ptr = symhead[index];

while(ptr)

{

if(!strcmp(ptr->name, node->name))

{

printf("LINE : %d DUPLICATE SYMBOL ERROR\n", line\_num);

return 1;

//중복된 SYMBOL이 발견되었다면 DUPLICATE SYMBOL ERROR

}

ptr = ptr->list;

}

temp = (Sym\*)malloc(sizeof(Sym));

strcpy(temp->name, node->name);

temp->location = node->location;

temp->list = symhead[index]->list;

symhead[index]->list = temp;

//head가 비어있지 않다면 head의 뒤에 연결

}

return 0;

//build 성공시 성공 return value 0

//build 실패시 실패 return value 1

}

/\*------------------------------------------------------------------------------------\*/

/\*함수 : find\_symbol(Sym\*\*symhead, char \*symbol)

/\*목적 : object code 계산을 위해 operand symbol의 location값을 얻기 위한 함수 /\*리턴값 : 없음

/\*------------------------------------------------------------------------------------\*/

int find\_symbol(Sym\*\*symhead, char \*symbol)

{

Sym \*temp;

int i, index, sum = 0, const\_flag = 0;

for(i = 0; i < strlen(symbol); i++)

{

if(symbol[i] >= '0' && symbol[i] <= '9') continue;

else const\_flag = 1;

}

//OPERAND가 상수인지 체크, 상수면 test = 0, 상수가 아니면 test = 1;

if(const\_flag == 0) return -2;

//OPERAND 상수인 경우 상수 return value -2;

for(i=0;i<strlen(symbol);i++)

sum += symbol[i];

index = sum % 100;

temp = symhead[index];

while(temp)

{

if(!strcmp(temp->name, symbol))

return temp->location;

//OPERAND SYMBOL 발견시 SYMBOL의 location값 return

temp = temp->list;

}

return -1;

//탐색 실패시 실패 return value -1;

}

/\*------------------------------------------------------------------------------------\*/

/\*함수 : location\_counter(char \*format,unsigned int \*location, char \*mnemonic)

/\*목적 : assemble과정 중 location을 계산하기 위한 함수

/\*리턴값 : 없음

/\*------------------------------------------------------------------------------------\*/

void location\_counter(char \*format,unsigned int \*location, char \*mnemonic)

{

if(!strcmp(format, "1")) \*location += 1;

else if(!strcmp(format, "2")) \*location += 2;

else if(!strcmp(format, "3")) \*location += 3;

else if(!strcmp(format, "4")) \*location += 4;

else if(!strcmp(format, "3/4"))

{

if(mnemonic[0] == '+')

{

\*location += 4;

strcpy(format, "4");

}

else

{

\*location += 3;

strcpy(format, "3");

}

}

}

/\*------------------------------------------------------------------------------------\*/

/\*함수 : objectcode\_insert(int format, int first, int mid, int end, int location, Ob\*\* head, Ob\*\* tail, char \*var, int enter\_flag)

/\*목적 : 계산된 objectcode를 linked list node로 연결하는 함수

/\*리턴값 : 없음

/\*------------------------------------------------------------------------------------\*/

void objectcode\_insert(int format, int first, int mid, int end, int location, Ob\*\* head, Ob\*\* tail, char \*var, int enter\_flag)

{

Ob \*node = (Ob\*)malloc(sizeof(Ob));

node->list = NULL;

node->first = first;

node->mid = mid;

node->end = end;

node->location = location;

node->format = format;

strcpy(node->var, var);

node->enter\_flag = enter\_flag;

//자료 복사

if(!(\*head))

{

\*head = node;

\*tail = node;

}

//head가 비어있다면 head에 연결

else

{

(\*tail)->list = node;

\*tail = node;

}

//head가 비어있지 않으면 tail에 연결

}

/\*------------------------------------------------------------------------------------\*/

/\*함수 : reg\_num(char \*reg)

/\*목적 : format2의 object code인 경우 register name에 해당하는 register number를 반환한다.

/\*리턴값 : register number

/\*------------------------------------------------------------------------------------\*/

int reg\_num(char \*reg)

{

if(!strcmp(reg, "A")) return 0;

else if(!strcmp(reg, "X")) return 1;

else if(!strcmp(reg, "L")) return 2;

else if(!strcmp(reg, "B")) return 3;

else if(!strcmp(reg, "S")) return 4;

else if(!strcmp(reg, "T")) return 5;

else if(!strcmp(reg, "F")) return 6;

else if(!strcmp(reg, "PC")) return 8;

else if(!strcmp(reg, "SW")) return 9;

else return -1;

}

/\*------------------------------------------------------------------------------------\*/

/\*함수 : type(char \*origin, char \*\*command)

/\*목적 : 입력받은 사용자가 입력한 filename에 대하여 현재 디렉토리 내에서 찾아서 화면에 그대로 출력한다.

/\*리턴값 : 없음

/\*------------------------------------------------------------------------------------\*/

void type(char \*origin, char \*\*command)

{

FILE \*fp = fopen(command[1],"r");

char string[500];

if(!fp)

{

printf("There is no %s file\n",command[1]);

return ;

}

record(origin, command, 2);

while (!feof(fp))

{

fgets(string, 500, fp);

printf("%s",string);

}

fclose(fp); //file close

}