**과목명: 시스템프로그래밍**

담당 교수 명: 박 운 상

**<<Assignment 3>>**

**서강대학교 컴퓨터공학과**

**20171609**

**김 민 식**

**목 차**

1. **프로그램 개요**
2. **프로그램 설명**
   1. 프로그램 흐름도

2.1.1 전체적인 flow chart

2.1.2 Linking Loader Pass1 Flow chart

2.1.3 Linking Loader Pass2 Flow chart

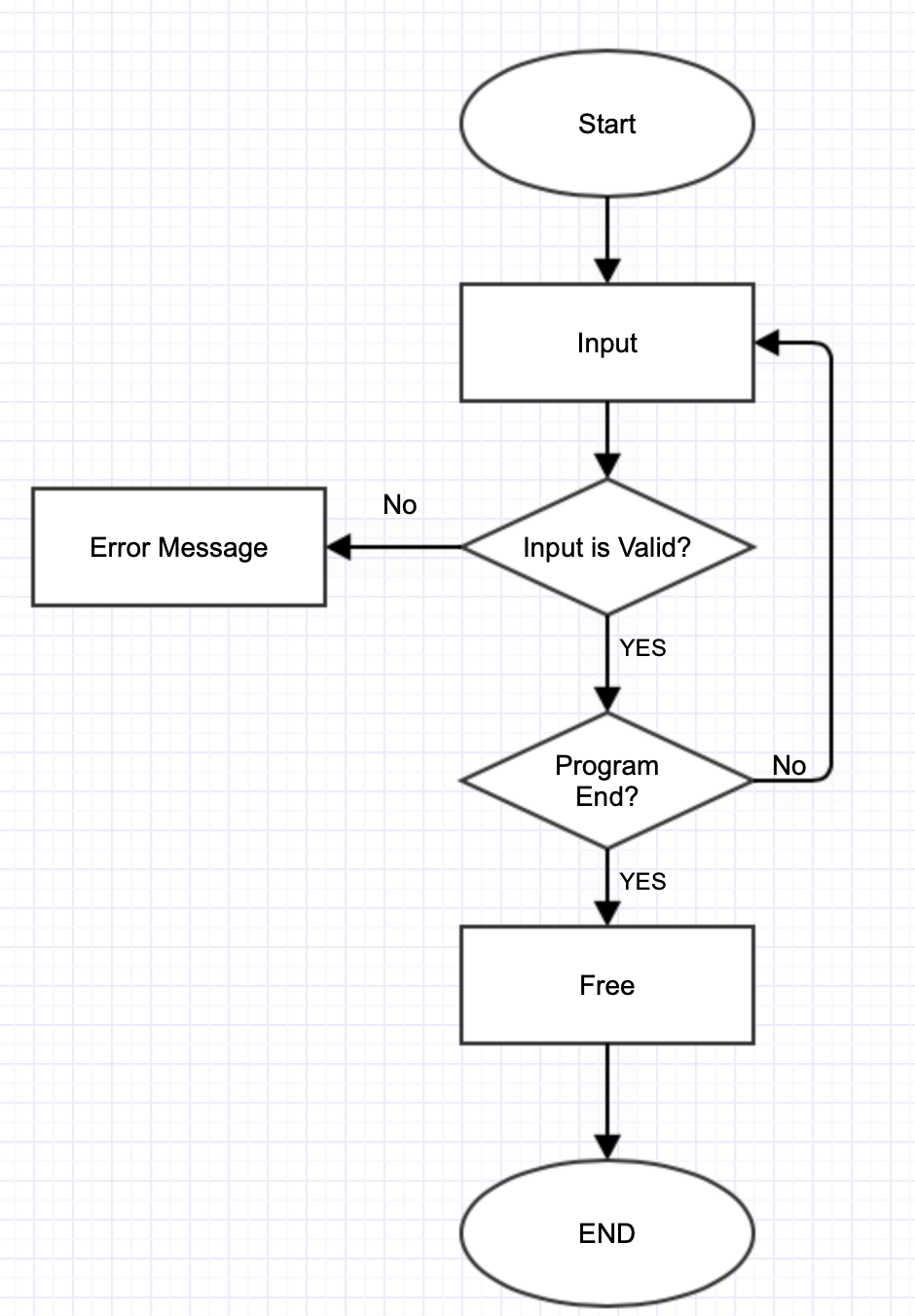
2.1.4 Run Flow Chart

1. **모듈 정의**
   1. void Func\_Progaddr()
      1. 기능
      2. 변수
   2. int Func\_HAshTab
      1. 기능
      2. 변수
   3. Void Func\_LoaderPass1
      1. 기능
      2. 변수
   4. Void Func\_LoaderPass2
      1. 기능
      2. 변수
   5. void Func\_Loader
      1. 기능
      2. 변수
   6. int \*RegisterAddress
      1. 기능
      2. 변수
   7. void Func\_Inst
      1. 기능
      2. 변수
   8. void Func\_Run()
      1. 기능
      2. 변수
   9. void Func\_Bp()
      1. 기능
      2. 변수
2. **전역 변수 정의**
   1. typedef struct \_ESTABNODE
   2. ESTAB \*ESTable[3]
   3. int ProgaAddress
   4. int CSAddress
   5. int RunAddres
   6. int bpNum
   7. int bp[100]
   8. int EndSwitch
   9. int ProgSize[3]
   10. int NUM\_A, NUM\_X, NUM\_L, NUM\_B, NUM\_S, NUM\_T, PC
   11. char Operator
3. **코드 설명**
   1. 20171609.c
   2. 20171609.h
   3. Makefile
4. **프로그램 개요**

Project3는 project 2에서 구현한 shell에 linking과 loader 기능을 추가하는 프로그램이다. ‘loader’ 명령어로 Linking, loader 기능이 실행되며 ‘run’ 명령어로 Linking 및 loading된 프로그램을 실행한다.

‘bp’ 명령어로 breakpoint를 만들어 디버깅 기능을 제공한다. Bp를 통해 프로그램을 멈출 때 마다 A, X, L, PC, B, S, T 각 레지스터에 들어가 있는 값을 출력한다.

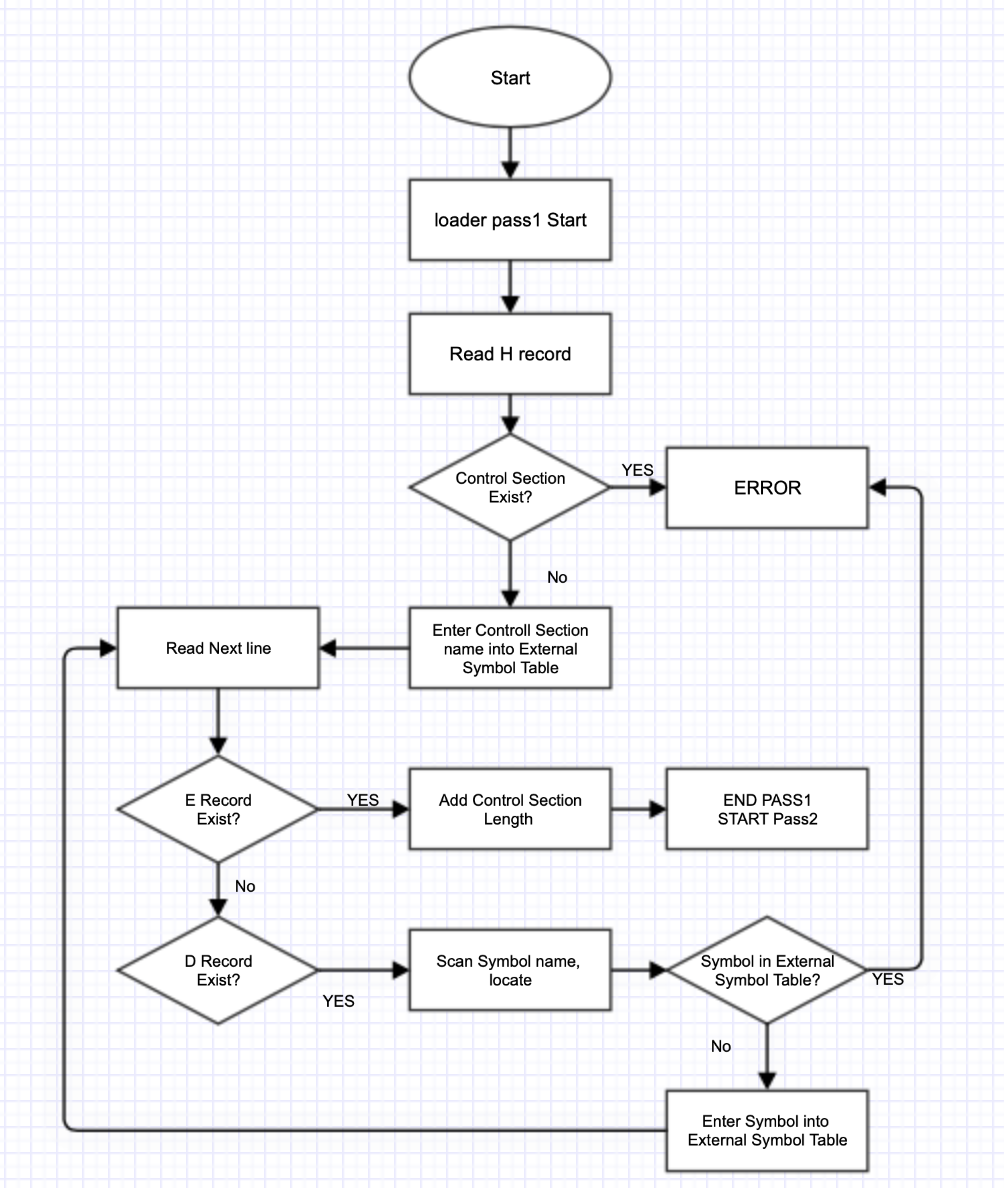
1. **프로그램 설명**
   1. **프로그램 흐름도**
2. 전체적인 flow chart

****

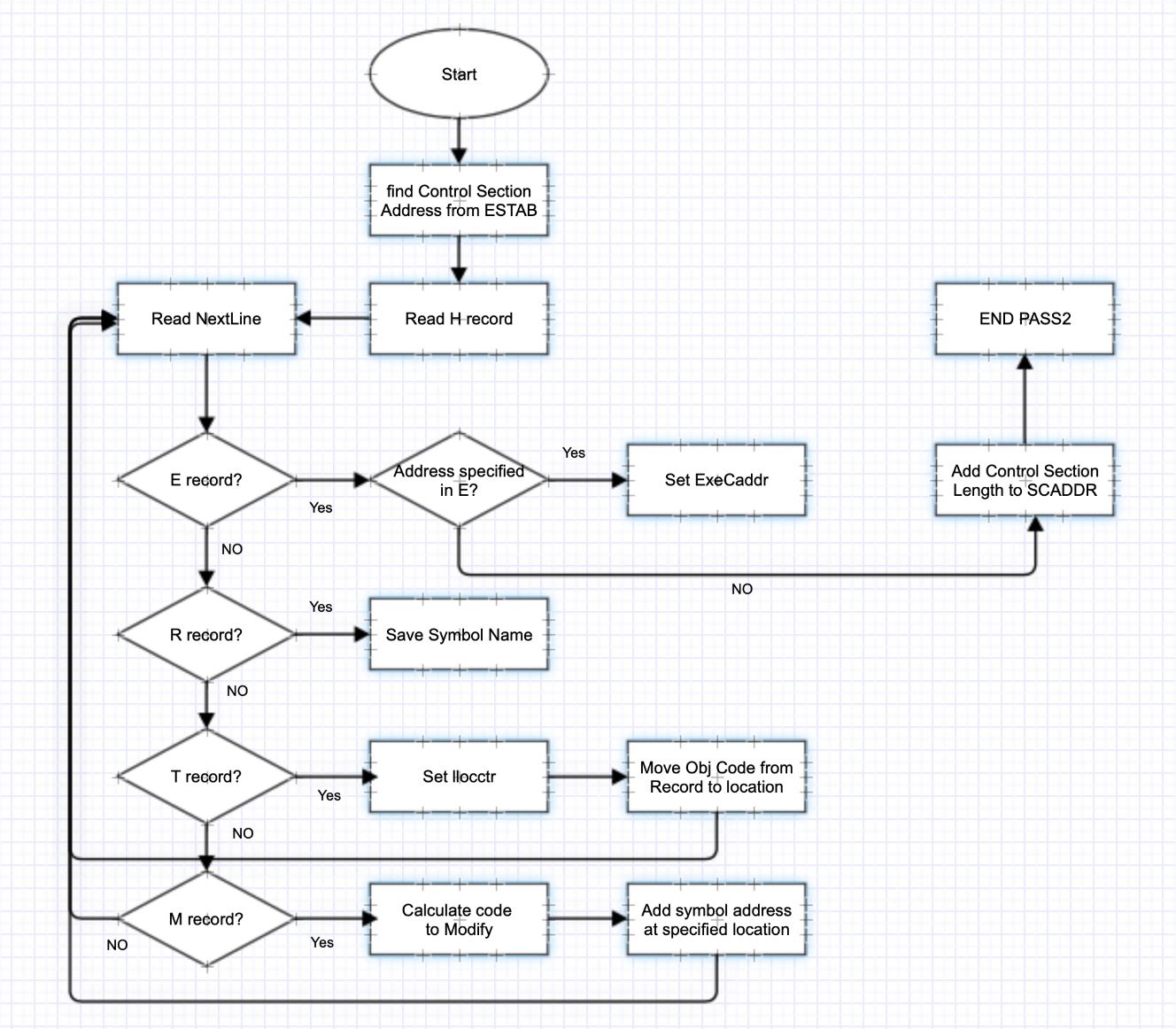
프로그램이 실행되면 sicsim> 이 나오면서 명령어를 입력 받는다.

Input받은 명령어가 invalid 하면 에러를 출력하고, valid 하면 해당 명령어의 기능을 수행한다. 프로그램을 종료하는 명령어가 나온다면 할당된 메모리를 모두 해제 한 후 프로그램을 종료한다.

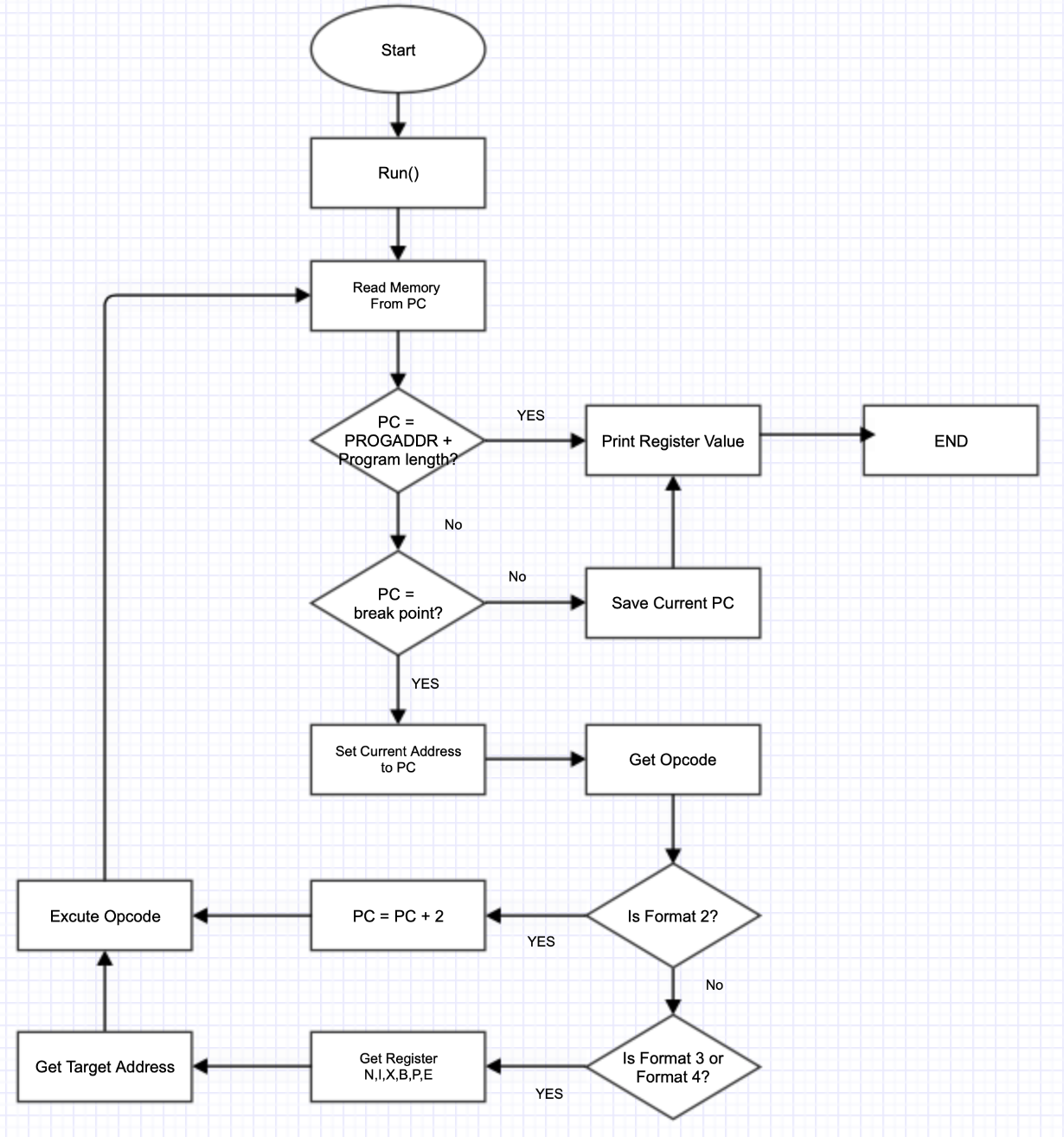
1. Linking Loader Pass1 Flow chart



1. Linking Loader Pass2 Flow chart



1. Run flow chart



1. **모듈 정의**
   1. **void Func\_Progaddr()**
      1. **기능**

‘loader’ ,’run’ 명령어를 수행할 때 시작 주소를 지정한다.

프로그램이 시작되면 progaddr는 기본적으로 0x00 을 주소로 지정하며, Progaddr 함수와 함께 입력되는 주소 인자는 한개만 가능하다. 따라서 두개 이상 인자를 입력할 경우 error 처리 한다.

* + 1. **변수**

- int ProgaAddress : 입력받은 address 인자 값을 저장한다. 초기값은 0이다.

* 1. **int Func\_HashTab(char EstabCsec[], char symbol[], int HashCurAddr, int HashFile)**
     1. **기능**

함수는 External Symbol Table을 생성하는 함수이다. Csec나 symbol을 입력 받고, 입력 받은 값이 External Symbol Table 안에 존재하지 않는다면 External Symbol Table에 새로 저장시켜 준다. 만약 입력받은 Csec나 Symbol이 이미 External Symbol Table에 있으면 Error를 출력하여준다.

* + 1. **변수**

- ESTAB \*HashEstabNode : External Symbol Table 탐색을 위해 사용되는 Node 변수 이다.

- ESTAB \*HashEstabInputNode : 입력되어 External Symbol Table에 없을 경우 테이블 안에 저장될 때 사용되는 변수이다.

- char char EstabCsec[] : 입력 받은 csec를 저장하는 array 변수이다.

- char char symbol[] : 입력 받은 symbol을 저장하는 array 변수이다.

- int HashCurAddr : csec와 symbol이 저장 될 때 그때 노드의 주소를 나타내는 변수이다.

- int int HashFile : 입력 받은 object 파일 개수이다.

- int HashSwitch : Func\_HashTab 함수의 작동 유무에 대한 스위치 변수이다.

- int i : 인덱스용 변수

* 1. **Func\_LoaderPass1(FILE \*LoaderFP[], int Pass1File)**
     1. **기능**

Func\_LoaderPass1 함수는 Loader Pass1 알고리즘을 수행하는데 사용되는 함수이다.

입력받은 파일을 1 Line 씩 읽고, record에 따라 각자 다른 방식으로 수행된다. 만약 H record 인 경우 프로그램이름을 csec name으로 저장하여 program 전체 크기를 저장한다. Csec name을 저장할 때 만약 external symbol table안에 이미 존재한다면 에러를 출력하고 종료한다.

Define record일 때는 Symbol과 그 주소를 따로 저장하며 symbol 또한 External Symbol Table 에 이미 존재한다면 에러를 출력하고 종료하게 된다.

한줄 씩 읽다가 입력 받은 파일을 모두 다 읽게 되면 프로그램의 크기를 csaddr에 더하여 다음 csec를 위한 주소를 계산하고 함수를 종료한다.

* + 1. **변수**

- char Pass1Input[100],: 파일을 한 줄씩 읽어 들일 때 읽어들인 문자열을 저장하는 배열 변수이다.

- char EstabCsec[7],: H record의 control section 이름을 저장 할 때 사용되는 변수이다.

- char symbol[7],: D record의 symbol 이름을 저장할 때 사용되는 변수이다.

- char Pass1Addr[7];: D record의 주소를 저장할 때 사용되는 변수이다.

- int Pass1CurAddr: 함수를 실행중 일 때 현재의 주소를 저장할 때 사용되는 변수이다.

- int i, j, k : 배열 탐색을 위한 인덱스 변수이다.

- FILE \*LoaderFP[] : 읽어들인 파일을 입출력하는데 사용되는 변수이다.

- int ProgaAddress : ProgaAddress를 저장하는데 사용되는 변수이며 csaddress를

입력받은 Progaddress로 초기화하는데 사용된다.

- int SCAddress : 다음 control section 주소를 계산할 때 사용되는 변수이다.

- int file\_num : 입력받은 파일의 개수를 나타내는 변수이다.

- int ProgSize[3] : 다음 control section 주소 계산을 위해 각 프로그램 크기를 저장한다.

* 1. **void Func\_LoaderPass2(FILE \*LoaderFP[], int Pass2File)**
     1. **기능**

Func\_LoaderPass2 함수는 Loader Pass2 알고리즘을 수행할 때 사용되는 함수이다.

입력받은 파일은 한 줄씩 읽으며 record에 따라 다르게 처리한다. R record인 경우 각 key값과 symbol name을 저장한다. Reference 사전 처음에는 csec name을 저장한다.

T record인 경우 1 바이트씩 읽어 virtual memory에 저장하며 M record인 경우 레퍼런스 번호에 따라 object Code를 relocation 한다.

Csec 와 symbol이 external symbol table에 존재하는지 확인한 후 존재하지 않은 csec, symbol을 참조할 경우 에러를 출력한다.

E record인 경우 프로그램 크기를 cs address에 더하여 다음 control section을 위한 주소를 계산하고 종료한다.

* + 1. **변수**

- char Pass2Input[100] : 파일을 한 줄씩 읽어 들일 때 읽어들인 문자열을 저장하는 배열 변수이다.

- char Pass2Dic[100][7] : R record의 각 key, value를 저장하는 array 변수이다.

- char Pass2Num[3] : R record의 key 값을 저장한다.

- char Pass2Symbol[7] : R record의 value 값을 저장한다.

- char Pass2Temp[7] : 입력받은 문자열을 저장하는 temp 변수이다.

- char ModiStr[10] : Modify 과정에서 object code를 문자열로 변환한 값을 저장하는 변수이다.

- int ModiObj : 문자열로 저장한 objcode를 16진수로 변환하여 저장할 때 사용하는 변수이다.

- int ModiAddr: M record에서 modify되어야 하는 주소를 계산하여 저장한다.

- int LPTemp : Loadpass 함수에서 사용되는 temp 변수이다.

- int TAddr : T record의 시작 주소를 저장한다.

- int TSize : T record의 한 줄의 길이를 저장한다.

- int TOneByte : 메인 메모리에서의 1 바이트를 읽어 들인다.

- int TMemory: T record에서 1byte에 해당하는 값을 저장할 메모리의 주소를 저장한다.

- int LPSwitch : Loadpass 함수에서 사용되는 switch 변수이다.

- int i, j, k : 배열 탐색 등에 사용되는 인덱스용 변수들이다.

- FILE \*LoaderFP[] : object file의 파일 포인터를 저장한다.

- int ProgaAddress : ProgaAddress를 저장하는데 사용되는 변수이며 csaddress를

입력받은 Progaddress로 초기화하는데 사용된다.

- int SCAddress : 다음 control section 주소를 계산할 때 사용되는 변수이다.

- int file\_num : 입력받은 파일의 개수를 나타내는 변수이다.

- int ProgSize[3] : 다음 control section 주소 계산을 위해 각 프로그램 사이즈를 저장한다.

* 1. **void Func\_Loader()**
     1. **기능**

Input 으로 들어온 object file을 읽어서 linking 작업을 수행시켜주는 함수이다. PASS1 단계는 Func\_loadPass1 함수를 호출하여 실행하고 PASS2 단계는 Func\_LoadPass2 함수를 통해 실행된다. Linking 작업이 다 끝나면 memory에 그 결과를 기록해준다.

* + 1. **변수**

- FILE \*LoaderFP[3] : input을 통해 입력 받은 오브젝트 파일에 대한 파일 포인터 배열을 나타내기 위한 변수이다.

- ESTABNODE \*LoadSearchNode : controlsection과 symbol을 탐색하고, 이들을 출력할 때

사용되는 변수이다.

- char LoaderTemp[100] : 입력 받은 instruction에 사용되는 temp 변수이다.

- char LoaderProgName[3][100] : 각 오브젝트 파일의 prog name을 저장한다.

- char \*temp : 문자열 포인터에 대한 temp 변수이다.

- int LoaderLength : 문자열의 길이를 저장하기 위한 변수이다.

- int i : 인덱스용 변수이다.

- ESTAB \*EsTable[] : External symbol table을 탐색, 초기화 할 때 사용되는 노드 변수이다.

- int loader\_error : PASS1 또는 PASS2에서 에러가 있을 경우 TRUE, 아니면 FALSE이다. 만약 TRUE라면 프로그램을 종료한다.

- int NUM\_A, NUM\_X, NUM\_L, NUM\_B, NUM\_S, NUM\_T, PC : 각 레지스터 주소를

나타내는 변수이다.

* 1. **int \*RegisterAddress(char ResAdd)**
     1. **기능**

register의 실제 주소를 리턴할 때 사용되는 함수이다. 매개 변수로 register의 번호를 불러온 다음 그 register의 실제 주소를 리턴한다.

* + 1. **변수**

- int NUM\_A, NUM\_X, NUM\_L, NUM\_B, NUM\_S, NUM\_T, PC : 입력 값에 따라 레지스터

주소 값을 반환한다.

* 1. **void Func\_Inst(int OP, unsigned char \*InstMemory, int InstImmediate, int TA, int InstExtend)**
     1. **기능**

Run 이 입력되었을 때 input으로 들어온 opcode에 따라서 각각의 기능을 수행한다. register A, B, T, S, X, L, PC, SW 값을 업데이트 하며, immediate, indirect, simple, sic addressing 각각에 대해 수행을 한다.

* + 1. **변수**

- int InstVar : 메모리 값을 가져와 저장하고, 연산에 사용되는 변수이다.

- int InstSwitch : Func\_Inst 함수에서 사용되는 스위치 변수이다.

- int OP : opcode 의 값을 저장할 때 사용되는 변수이다.

- unsigned char \* InstMemory : TA가 가리키는 memory 주소값을 나타낼 때 사용되는 변수이다.

- int InstImmediate : immediate addressing의 경우 해당 값이, 아니면 NULL이 저장된다.

- int TA : Target Adress를 저장할 때 사용되는 변수이다.

- int InstExtend : format4 인지 아닌지 판별할 때 사용되는 변수이다.

- int I : index 변수

* 1. **void Func\_Run()**
     1. **기능**

Loader 명령어를 통해 메모리에 프로그램이 할당 되었으면 run 명령어를 통해 그 프로그램을 실행한다. 현재 progaddr로 설정된 주소에 저장되어있는 object code를 토대로 프로그램을 실행하고, 입력된 break point에 PC가 다다르면 디버깅의 기능으로 프로그램 실행을 중단한다.

다음에 또 run 명령어가 입력된다면 이전의 중단점(break point)부터 프로그램을 다시 실행하고 반복되다가 더 이상 break point가 존재하지 않는 경우 프로그램이 끝까지 실행된다.

프로그램이 중단될 때, 프로그램이 끝날 때마다 A, X, L, PC, B, S, T 레지스터 안에 들어있는 값을 출력한다.

* + 1. **변수**

- char RunByte1[3], RunByte2[3], RunByte3[3], RunByte4[3] : byte를 저장할 때 마다 사용되는 임시 temp 변수이다.

- char RunHalfByte[2] : half byte를 저장하는 문자열 변수이다.

- int byte1, byte2, byte3 : n번째 byte를 저장할 때 사용되는 변수이다.

- int byte\_half : half byte을 저장한다.

- int RunEndAddr : 프로그램 마지막 주소를 저장할 때 사용되는 변수이다.

- int RunLength : 프로그램 사이즈를 저장할 때 사용되는 변수이다.

- int RLength : 문자열의 길이를 저장하는 변수이다.

- int I : 문자열 탐색, 반복문을 위한 인덱스 변수이다.

- int Endswitch : 프로그램이 끝나는지 판단하는 스위치 변수이다.

- int ProgAddress : 실행되고 있는 프로그램의 현재 주소값을 나나태는 변수이다.

- int NUM\_A, NUM\_X, NUM\_L, NUM\_B, NUM\_S, NUM\_T, PC : 각 레지스터 주소를

나타내는 변수이다.

* 1. **void Func\_Bp()**
     1. **기능**

[Bp + address] 입력시 입력 주소값에 break point가 지정된다. 입력 주소값이 16진수가 아니거나 프로그램을 벗어나는 크기일 경우 에러를 출력한다.

[bp] 입력 시 지금까지 입력하여 저장되었던 break point 를 화면에 출력해준다.

[bp clear] 입력 시 현재 존재하는 모든 break point를 삭제해준다.

* + 1. **변수**

- int BPMaxnum : 프로그램의 전제 크기를 나타내며, break point의 범위를 나타낸다.

- int lengthBP : 문자열의 길이를 저장하는 변수이다.

- char \*BPAddr : 입력 받은 address를 저장할 때 사용되는 변수이다.

- char BPTemp[100]: 입력 받은 명령어를 복사하여 저장한다.

- int bpNum : Bp 명령문을 통해 break point가 생길 때마다 1씩 증가해주는 변수이다.

- int i : 문자열 탐색, 반복문을 위한 인덱스 변수이다.

1. **전역 변수 정의**

**typedef struct \_ESTABNODE**

**{**

**char EstabCsec[7], symbol[7];**

**int address, Establength;**

**struct \_ESTABNODE \*nNode;**

**}ESTABNODE;**

External Symbol Table을 표현하는 구조체이다.

Controlsection, symbol을 저장하는 문자열 변수와 주소와 길이를 저장하는 int 변수, 그리고 다음 노드를 가리키는 포인터 변수를 가진다.

* 1. **ESTAB \*ESTable[3]**

입력 받은 object file의 control section의 이름과 시작 주소, 그리고 그 안에 정의된 external symbol들의 이름과 주소를 저장하는 구조체 변수이다.

같은 control section 안의 symbol들은 Linked list로 연결 시킨다. control section의 개수만큼의 크기를 가지는 배열로 사용할 것이기 때문에 포인터 변수로 선언하였다

* 1. **int ProgaAddress**

Progaddr address 명령어를 입력 받았을 때 address 값을 시작 주소로 할 때 사용되는 변수이다.

이외에도 linking, loading에서 프로그램의 상대적 주소를 사용자가 원하는 주소에 맞춰 변환할 때, run, loader에서 csaddr을 초기화하는 등에도 사용되는 변수이다.

* 1. **int CSAddress**

현재 사용하고 있는 control section의 시작 주소를 저장할 때 쓰는 변수이다. 처음에는 ProgaAddress로 초기화 되었다가, 다음 control section으로 넘어갈 때마다 이전 프로그램의 사이즈를 더한다.

* 1. **int RunAddress**

run에서 사용하는 가상메모리의 주소를 나타내는 정수형 변수이다.

* 1. **int bpNum**

[bp + address] 명령어를 통해 break point가 생길 때마다 1씩 증가시킨다. 즉, break point의 개수를 나타낼 때 사용되는 변수이다.

* 1. **int bp[100]**

입력 받은 break point들을 저장하는 정수형 배열 변수이다. run 명령어 실행 시 이 배열에 있는 bp 값에서 run을 멈추는 작동을 하게 된다.

* 1. **int EndSwitch**

프로그램이 종료되는지 판단할 수 있는 스위치 변수이다. TRUE면 프로그램이 종료된다.

* 1. **int ProgSize[3]**

Object file에서 각 control section 마다 프로그램의 크기를 저장하는 정수형 배열 변수이다.

Loader Pass1에서 Header record에 적혀있는 프로그램 사이즈를 저장하게 된다.

* 1. **Int NUM\_A, NUM\_X, NUM\_L, NUM\_B, NUM\_S, NUM\_T, PC;**

Linking, loading, run 명령어를 사용 할 때 각 레지스터의 값들을 나타내는 변수이다.

* 1. **char Operator**

Linking, loading, run 명령어를 사용 할 때 사용되는 레지스터 이다. ‘=’, ‘<’, ‘>’ 문자를 표현하기 위해 character 형으로 선언한다.

1. **코드 설명**
   1. **20171609.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <sys/stat.h>

#include "20171609.h"

int main()

{

FILE \*fp;

HNODE = (NAMENODE\*)malloc(sizeof(NAMENODE));

LNODE = HNODE;

char input1[7], input2[5], input3[11];

//input1 : 처음으로 입력 받는 명령어

//input2 : 명령어 부분만 입력받는 변수

//input3 : 명령어 뒷부분 저장하는 변수

int InputKey, opcode;

int i;

memset(size, '\0', sizeof(size)); //변수 초기화

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

while(fscanf(fp, "%X %s %s", &opcode, input1, input2) != EOF)

{

InputKey = (input1[0] + input1[strlen(input1)-1]) %20;

HASHNODE \*First\_hash = (HASHNODE\*)malloc(sizeof(HASHNODE));

strcpy(First\_hash->hashnodename, input1);

strcpy(First\_hash->hashformat, input2);

First\_hash->opcode = opcode;

if(OPHnode[InputKey] != NULL)

{

First\_hash->nNode = OPHnode[InputKey];

OPHnode[InputKey] = First\_hash;

}

else

{

OPHnode[InputKey] = First\_hash;

First\_hash->nNode = NULL;

}

}

fclose(fp);

for(i = 0; i < 3; i = i + 1) ESTable[i] = (ESTABNODE\*)malloc(sizeof(ESTABNODE));

while(1)

{

printf("sicsim> ");

Func\_Input();

if(ins[0] != '\0')

{

Func\_instruction(input3);

if(strcmp(input3,"help")==FALSE || strcmp(input3,"h")==FALSE) //명령어가 h, help일 때

{

if(Func\_Cal\_Fac()==FALSE)

{

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

printf("f[ill] start, end, value\nreset\nopcode mnemonic\nopcodelist\n");

printf("assemble filename\ntype filename\nsymbol\n");

printf("progaddr address\nloader object-filename1 object-filename2 ...\nrun\nbp\nbp clear\nbp address\n");

Func\_History\_Save(); //명령어 history에 추가

}

else printf("Error : Input Error.\n"); //인덱스가 비어있지 않으면 에러 출력

}

else if(strcmp(input3,"dir")==FALSE || strcmp(input3,"d")==FALSE) // d, dir일 경우

{

if(Func\_Cal\_Fac()==FALSE)

{

DIR \*dir;

struct dirent \*files;

struct stat f\_stat;

dir = opendir(".");

while ((files = readdir(dir)) != NULL)

{

lstat(files->d\_name, &f\_stat);

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

if (S\_ISDIR(f\_stat.st\_mode)) printf("/"); //폴더인 경우 /추가

else if (S\_IEXEC & f\_stat.st\_mode) printf("\*"); //실행파일인 경우 \*추가

}

printf("\n");

closedir(dir);

Func\_History\_Save(); //명령어 history에 추가

}

else printf("Error : Input Error.\n"); //인덱스가 비어있지 않으면 에러 출력

}

else if(strcmp(input3,"quit")==FALSE || strcmp(input3,"q")==FALSE) // q, quit일 경우

{

if(Func\_Cal\_Fac()==FALSE) break;

else printf("Error : Input Error.\n"); //인덱스가 비어있지 않으면 에러 출력

}

else if(strcmp(input3,"history")==FALSE || strcmp(input3,"hi")==FALSE)

{

if(Func\_Cal\_Fac()==FALSE)

{

Func\_History\_Save();

int count = 1; //history 개수 나타내는 변수

NAMENODE \*Show\_node = HNODE->nNode;

while(Show\_node != NULL)

{

printf("%d\t%s\n", count, Show\_node->namenodename);

Show\_node = Show\_node->nNode;

count = count + 1;

}

}

else printf("Error : Input Error.\n"); //인덱스가 비어있지 않으면 에러 출력

}

else if(strcmp(input3,"reset")==FALSE)

{

if(Func\_Cal\_Fac()==FALSE)

{

memset(size, '\0', sizeof(size));

Func\_History\_Save(); //명령어 history에 추가

}

else printf("Error : Input Error.\n"); //인덱스가 비어있지 않으면 에러 출력

}

else if(strcmp(input3,"opcodelist")==FALSE) //opcodelist 입력시

{

if(Func\_Cal\_Fac()==FALSE)

{

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

{

printf("%d : ", i);

HASHNODE \*node = OPHnode[i];

while(node)

{

printf("[%s,%02X]", node->hashnodename, node->opcode);

if(node->nNode != NULL) printf(" -> ");

node = node->nNode;

}

printf("\n");

}

Func\_History\_Save(); //명령어 history에 추가

}

else printf("Error : Input Error.\n"); //인덱스가 비어있지 않으면 에러 출력

}

else if(strcmp(input3,"opcode")==FALSE)

{

Func\_OPcode();

switch1 = 0;

}

else if(strcmp(input3,"dump")==FALSE || strcmp(input3,"du")==FALSE) Func\_Dump(); //d, dump 입력

else if(strcmp(input3,"fill")==FALSE || strcmp(input3,"f")==FALSE) Func\_Fill();//f, fill 인 경우

else if(strcmp(input3,"edit")==FALSE || strcmp(input3,"e")==FALSE) Func\_Edit(); //e, edit인 경우

else if(strcmp(input3, "assemble")==FALSE) //assemble 입력

{

Assemble();

switchFilename = 0;

}

else if(strcmp(input3, "type")==FALSE) //type 입력

{

Func\_Type();

switchFilename = 0;

}

else if(strcmp(input3, "symbol")==FALSE) Func\_Symbol(); //symbol 입력

else if(strcmp(input3, "run")==FALSE) Func\_Run(); //run 입력

else if(strcmp(input3, "bp")==FALSE) Func\_Bp(); //bp 입력

else if(strcmp(input3, "loader")==FALSE) Func\_Loader(); //loader 입력

else if(strcmp(input3, "progaddr")==FALSE) Func\_Progaddr(); //progaddr 입력

else printf("Error : Input Error.\n");

}

memset(ins, '\0', sizeof(ins));

}

return 0;

}

* 1. **20171609.h**

#define TRUE 1

#define FALSE 0

#define HEX 16

#define MAX\_\_NUM 0xFFFFF

#define MAXE 0x1E

unsigned char size[1048576]; //가상 메모리 저장 변수

char ins[100], info[3][100];//인자 저장 변수

char Operator;

char \*arr\_fac[12] = {"du","dump","e","edit","f","fill","opcode", "assemble", "type", "progaddr", "loader", "bp"};

char \*arr\_ins[12] = {"h","help","d","dir","q","quit","hi","history","reset","opcodelist", "symbol", "run"};

//시작 주소, 길이 등 저장 변수

int lo\_start = 0, lo\_end = 0, bpNum = 0, bp[100];

//각 프로그램, 변수 주소값 저장 변수

int ProgaAddress = 0, CSAddress = 0, runAddress = 0, address = -1;

//프로그램 종료, 함수 종료 등의 switch 변수

int LoaderSwitch = FALSE, EndSwitch = FALSE, switch1 = FALSE, switchFilename = FALSE;

//register 변//history의 제일 처음을 가르키는 포인터

int ProgSize[3], NUM\_A, NUM\_X, NUM\_L, NUM\_B, NUM\_S, NUM\_T, PC;

//history linked list 저장 구조체

typedef struct \_NAMENODE

{

char namenodename[100]; //이름 저장 변수

struct \_NAMENODE \*nNode; //다음 노드 link

}NAMENODE;

//opcode hash table 저장 구조체

typedef struct \_HASHNODE

{

char hashnodename[7], hashformat[5];

struct \_HASHNODE\* nNode; //다음 노드 link

int opcode;

}HASHNODE;

//symbol table 저장 구조체

typedef struct \_SYMNODE

{

int locctr; //위치 저장 변수.

char symbol[10]; //symbol 이름 저장 변수

struct \_SYMNODE\* nNode; //다음 노드 link

}SYMNODE;

//object code 저장 구조

typedef struct \_COMNODE

{

char LineNumber[10],Comloc[10],symbol[10],opcode[10];

char ComOper1[100],ComOper2[100];

}COMNODE;

//external symbol table 저장 구조체

typedef struct \_ESTABNODE

{

char EstabCsec[7], symbol[7]; //control section, symbol 저장 변수

int address, Establength; //주소, 길이 저장 변수

struct \_ESTABNODE \*nNode; //다음 노드 ㅣink

}ESTABNODE;

NAMENODE \*HNODE, \*LNODE; //history의 제일 처음과 마지막을 가리키는 포인터 변/

HASHNODE \*OPHnode[20], \*HashTargetNode; //hashtable pointer 변수

SYMNODE \*SHnode[26], \*RecentSHnode[26]; //symboltable pointer 변수

ESTABNODE \*ESTable[3]; //extend symbol table 변수

//문자열 입력 받는 함수

//100 이상의 크기를 입력받으면 에러 리턴

void Func\_Input()

{

int i = 0, j = 0;

while(1)

{

if(i > 100) //max size 초과

{

i = 0;

printf("Error : Invalid Size.\n");

memset(ins, '\0', sizeof(ins));

j = sizeof(ins);

break;

}

scanf("%c", &ins[i]);

if(ins[i] == '\n')

{

ins[i] = '\0';

i = i - 1;

break;

}

i = i + 1;

}

}

//입력된 명령어를 new node에 저장하는 함수이다.

//history 입력 시 함수를 호출하여 지금까지 저장된 명령어를 출력한다.

void Func\_History\_Save()

{

NAMENODE \*save\_new\_node = (NAMENODE\*)malloc(sizeof(NAMENODE));

NAMENODE \*save\_second\_node = (NAMENODE\*)malloc(sizeof(NAMENODE));

strcpy(save\_new\_node->namenodename, ins);

strcpy(save\_second\_node->namenodename, ins);

LNODE->nNode = save\_new\_node;

LNODE = save\_new\_node;

save\_new\_node->nNode = NULL;

}

//명령어의 인자의 유효성을 판단하는 함수이다.

int Func\_Cal\_Fac()

{

char arr\_facstr[100]; //명령어의 instruction 인자 위치부터 저장되는 문자열 변수.

memset(info, '\0', sizeof(info)); //초기화

memset(arr\_facstr, '\0', sizeof(arr\_facstr));

//길이, switch, count 변ㅅ

//com : comma 개수.

int com = 0, switchFactor = FALSE, FacCount = 0, count2, i, j, k = 0;

int FacLength = 0, CalSwitch = FALSE;

//명령어의 명령어 다음 위치에 공백이 아닌 다른 값이 입력되면 새로운 문자열에 저장시킨다.

for(i = lo\_end; i < strlen(ins); i = i + 1)

{

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

{

FacLength = strlen(ins);

for(j = i; j < strlen(ins); j = j + 1) arr\_facstr[j-i] = ins[j];

FacLength = FacLength + 1;

break;

}

}

arr\_facstr[strlen(ins) - lo\_end] = '\0';

for(i = 0; i < strlen(arr\_facstr); i = i + 1) if(arr\_facstr[i] == ',') com = com + 1;

//컴마의 개수가 2개 초과하면 에러 출력.

if(com > 2)

{

printf("Error : Input Error.\n");

return -1;

}

for(i = 0; i < strlen(arr\_facstr); i = i + 1) if(arr\_facstr[i] != ' ' && arr\_facstr[i] != '\t') switchFactor = TRUE;

if(switchFactor != TRUE) return FALSE;

//파일을 입력받는다.

for(i = 0; i < strlen(arr\_facstr); i = i + 1)

{

if(arr\_facstr[i] == ',')

{

k = 0;

FacCount = FacCount + 1; //카운트 증가

}

else

{

if(switch1 != TRUE && arr\_facstr[i] != '\t' && arr\_facstr[i] != ' ')

{

if(switchFilename)

{

CalSwitch = TRUE;

if(info[FacCount][0] != '\0' && (arr\_facstr[i-1] == ' ' || arr\_facstr[i-1] == '\t')) return -1;

info[FacCount][k] = arr\_facstr[i];

k = k + 1;

}

else

{

//파일 이름에 인자가 숫자와 영어 대소문자만 가능하게 하는 조건문이다.

if('0' <= arr\_facstr[i] && arr\_facstr[i] <= '9')

{

CalSwitch = TRUE;

if(info[FacCount][0] != '\0' && (arr\_facstr[i-1] == ' ' || arr\_facstr[i-1] == '\t')) return -1;

info[FacCount][k] = arr\_facstr[i];

k = k + 1;

}

else if('A' <= arr\_facstr[i] && arr\_facstr[i] <= 'F')

{

CalSwitch = TRUE;

if(info[FacCount][0] != '\0' && (arr\_facstr[i-1] == ' ' || arr\_facstr[i-1] == '\t')) return -1;

info[FacCount][k] = arr\_facstr[i];

k = k + 1;

}

else if('a' <= arr\_facstr[i] && arr\_facstr[i] <= 'f')

{

CalSwitch = TRUE;

if(info[FacCount][0] != '\0' && (arr\_facstr[i-1] == ' ' || arr\_facstr[i-1] == '\t')) return -1;

info[FacCount][k] = arr\_facstr[i];

k = k + 1;

}

else return -1;

}

}

else if(switch1 && (arr\_facstr[i] == ' ' || arr\_facstr[i] == '\t')!=TRUE)

{

CalSwitch = TRUE;

if(info[FacCount][0] != '\0' && (arr\_facstr[i-1] == '\t' || arr\_facstr[i-1] == ' ')) return -1;

info[FacCount][k] = arr\_facstr[i];

k = k + 1;;

}

}

if(CalSwitch == TRUE)

{

FacLength = FacCount;

}

}

FacCount = 0;

for(i = 0; i < com + 1; i++) if(info[i][0]) FacCount = FacCount + 1;

if(FacCount != com + 1) return -1;

else return FacCount;

}

//opcode 에 대한 값을 출력시켜주는 함수이다.

void Func\_OPcode()

{

switch1 = TRUE;

//인자가 유효하지 않으면 에러

if(Func\_Cal\_Fac() != TRUE)

{

printf("Error : OPcode Input Error.\n");

return;

}

//포인터 변수를 이용하여 key값 계산

HASHNODE \*find\_node = OPHnode[(info[0][0] + info[0][strlen(info[0])-1]) %20];

int val\_opcode = -1;

while(find\_node != NULL)

{

if(strcmp(find\_node->hashnodename, info[0])==FALSE) val\_opcode = find\_node->opcode;

find\_node = find\_node->nNode;

}

//opcode가 valid 하지않으면 에러 출력, 아니면 opcode key 값 출력

if(val\_opcode == -1)

{

printf("Error : Not Found OPcode.\n");

return;

}

else printf("opcode is %02X\n", val\_opcode);

Func\_History\_Save();

}

//메모리를 출력해주는 함수

//인자의 개수에 따라 10줄을 출력할지, start~end 를 출력할지 결정한다.

void Func\_Dump()

{

int DumpStart = 0, DumpEnd = 0; //메모리의 시작과 끝 주소

int start, end, i, j; //출력 메모리 시작 끝 주소

//입력 받은 인자가 valid 하지 않으면 에러 출력

if(Func\_Cal\_Fac() > 2 || Func\_Cal\_Fac() == -1)

{

printf("Error : Input Error.\n");

return;

}

else if(Func\_Cal\_Fac() == 2)

{

start = strtoul(info[0], NULL, HEX);

end = strtoul(info[1], NULL, HEX);

if(end > 1048576-1)

{

printf("Error : Input Error.\n");

return;

}

}

else if(Func\_Cal\_Fac() == TRUE)

{

start = strtoul(info[0], NULL, HEX);

end = start + 159;

if(end > 1048576-1) end = 1048576-1; //주소 넘어가면 끝까지 출력

}

else

{

if(address >= 1048576-1) address = -1; //주소 끝까지 넘어간 경우 초기화

start = address + 1; //시작주소는 끝 주소의 1을 더한 값이다

end = start + 159;

if(end > 1048576-1) end = 1048576-1;

}

//시작주소와 끝 주소가 유효한 범위에 존재하지 않으면 에러

if(start < 0 || start > 1048576-1 || end < 0)

{

printf("Error : Input Error.\n");

return;

}

if(start > end) //시작주소가 끝보다 더 크면 에

{

printf("Error : Input Error.\n");

return;

}

DumpStart = start - start % HEX;

DumpEnd = end - end % HEX;

//메모리에 있는 정보를 출력한다.

for(i = DumpStart; i <= DumpEnd; i = i + HEX)

{

printf("%05X ", i);

for(j = i; j < i+HEX; j = j + 1)

{

if(j < start || j > end) printf(" ");

else printf("%02X", size[j]);

printf(" ");

}

printf("; ");

for(j = i; j < i + HEX; j = j + 1)

{

if(size[j] < 0x20 || size[j] > 0x7E || (j < start) || (j > end)) printf(".");

else printf("%c", size[j]);

}

printf("\n");

}

address = end;

Func\_History\_Save();

}

//지정한 범위에 메모리 값을 채워주는 함수이다.

void Func\_Fill()

{

int i;

if(Func\_Cal\_Fac() != 3) //start, end, value 값이 제데로 입력되지 않은 경우

{

printf("Error : Input Error\n");

return;

}

if(strtoul(info[0], NULL, HEX) < 0 || strtoul(info[0], NULL, HEX) > (1<<20)-1 || strtoul(info[1], NULL, HEX) < 0 || strtoul(info[1], NULL, HEX) > (1<<20)-1 || strtoul(info[2], NULL, HEX) > 0xFF || strtoul(info[2], NULL, HEX) < 0x00) //start, end, value 범위가 벗어날 경우.

{

printf("Error : Input Error\n");

return;

}

if(strtoul(info[0], NULL, HEX) > strtoul(info[1], NULL, HEX)) //시작 번호가 끝 번호보다 큰 경우.

{

printf("Error : Wrong Input\n");

return;

}

for(i = strtoul(info[0], NULL, HEX); i < strtoul(info[1], NULL, HEX) + 1; i=i+1) size[i] = strtoul(info[2], NULL, HEX);

Func\_History\_Save();

}

//메모리의 값을 변경해주는 함수이다.

void Func\_Edit()

{

if(Func\_Cal\_Fac() != 2) //edit을 입력받았을 때 인자가 2개가 아니라면 입력 에러

{

printf("Error : Input Error\n");

return;

}

//주소나 그 값이 잘못 입력 되었을 경우

if(strtoul(info[0], NULL, HEX) < 0 || strtoul(info[0], NULL, HEX) > (1<<20)-1 || strtoul(info[1], NULL, HEX) > 0xFF || strtoul(info[1], NULL, HEX) < 0x00)

{

printf("Error : Input Error(Address, Value)\n");

return;

}

size[strtoul(info[0], NULL, HEX)] = strtoul(info[1], NULL, HEX);

Func\_History\_Save();

}

//입력된 명령어를 분석한 후 명령어가 valid 하다면 해당 명령어를 저장시켜주는 함수이다.

void Func\_instruction(char\* input\_inst)

{

int i, j, sizeCommand;

int InsSwitch = FALSE;

char InsCommand[100]; //인자 등을 제외한 순수한 명령어만 저장하는 변수이다.

memset(InsCommand, '\0', sizeof(InsCommand)); //초기화

//공백이 아닌 문자가 나온 인덱스를 lo\_start에 저장한다.

sizeCommand = strlen(ins);

for(i = 0; i < strlen(ins); i = i + 1) if((ins[i] == ' ' || ins[i] == '\t')!=TRUE) break;

lo\_start = i;

//명령어를 탐색하여 공백이 나오기 전ㅈ까지의 명령어를 저장한다.

for(i = lo\_start; i < strlen(ins); i = i + 1)

{

InsSwitch = TRUE;

if(ins[i] == ' ' || ins[i] == '\t') break;

InsCommand[i-lo\_start] = ins[i];

j = i;

}

lo\_end = i;

j = ++lo\_end; //명령어 마지막 위치의 다음을 접근하기 위해 1 증가.

//추출된 순수한 명령어가 single\_inst 에 있는 경우, 인자 또는 다른 문자가 있으면 error 출력

for(i = 0; i < 12; i = i + 1)

{

if(strcmp(InsCommand, arr\_ins[i])==FALSE)

{

for(; j < strlen(ins); j = j + 1)

{

if((ins[j] == ' ') || (ins[j] == '\t')) break;

else

{

InsSwitch = TRUE;

strcpy(input\_inst, "\0");

return;

}

}

strcpy(input\_inst, InsCommand);

return;

}

}

//arr\_fac 배열에 있는지 확인한 후 없으면 에러 출력.

for(i = 0; i < 12; i = i + 1)

{

if(strcmp(InsCommand, arr\_fac[i]) == FALSE)

{

strcpy(input\_inst, InsCommand);

return;

}

else strcpy(input\_inst, "\0");

}

}

//Symbol Table을 생성하는 함수이다.

//만약 테이블에 이미 존재하는 심볼일 경우 에러를 출력시킨다.

int Func\_Make\_Symbol(char \*symbol, long locctr)

{

//symbol table 탐색 포인트 노드 변수

SYMNODE \*SymInput = (SYMNODE\*)malloc(sizeof(SYMNODE)), \*SymCurrnet = SHnode[(symbol[0] - 'A')], \*SymPrevNode;

SymInput->locctr = (int)locctr;

strcpy(SymInput->symbol, symbol);

//테이블 안에 해당 키가 존재하지 않으면

if(SHnode[(symbol[0] - 'A')] == NULL)

{

SymCurrnet = SymInput;

SHnode[(symbol[0] - 'A')] = SymInput;

SHnode[(symbol[0] - 'A')]->nNode = NULL;

return FALSE;

}

else //테이블 안에 해당 키가 존재한다면

{

while(SymCurrnet != NULL) //테이블 탐색

{

if(strcmp(SymCurrnet->symbol, symbol)==FALSE) return -1; //중복일 경우 -1 리턴

SymCurrnet = SymCurrnet->nNode;

}

SymPrevNode = SHnode[(symbol[0] - 'A')];

SymCurrnet = SHnode[(symbol[0] - 'A')];

while(SymCurrnet != NULL) //테이블 탐

{

if (strcmp(SymCurrnet->symbol, symbol) > FALSE)//이미 존재하는 symbol보다 작은 경우

{

if(SymPrevNode == SymCurrnet) //table에 존재하지 않아 처음 들어갈 경우

{

SHnode[(symbol[0] - 'A')] = SymInput;

SHnode[(symbol[0] - 'A')]->nNode = SymCurrnet;

return TRUE;

}

SymPrevNode->nNode = SymInput;

SymInput->nNode = SymCurrnet;

return TRUE;

}

else if(strcmp(SymCurrnet->symbol, symbol) < FALSE) //이미 존재하는 symbol보다 큰 경우

{

SymPrevNode = SymCurrnet;

SymCurrnet = SymCurrnet->nNode;

}

}

SymPrevNode->nNode = SymInput;

SymInput->nNode = NULL;

return TRUE; //1을 리턴시킨다

}

}

//symbol table 안에 symbol이 존재하는지 확인시켜주는 함수이다.

int Func\_Symbol\_Int(char \*T)

{

//symbol이 테이블 안에 존재하면 그 symbol의 location을, 없으면 -1를 리턴한다.

int i;

for(i = 0 ; i < 26; i = i + 1)

{

SYMNODE \*SymFindNode = SHnode[i];

while(SymFindNode != NULL)

{

if(strcmp(SymFindNode->symbol, T)==FALSE) return SymFindNode->locctr; //locctr 리턴

SymFindNode = SymFindNode->nNode;

}

}

return -1;

}

//symbol table 의 메모리를 모두 해제 시켜주는 함수이다.

void Func\_Free\_SymTab(SYMNODE\*\* T)

{

int i;

for(i = 0; i < 26; i = i + 1)

{

SYMNODE \*SymFreeNode = T[i];

if(SymFreeNode == NULL) continue;

while(T[i] != NULL)

{

T[i] = T[i]->nNode;

SymFreeNode = NULL;

SymFreeNode = T[i];

}

}

}

//Assemble 명령어가 입력되었을 때 가장 최근에 생성된 symbol table을 recentshnode에 저장하는 함수이다.

void Func\_Copy\_SymTab()

{

Func\_Free\_SymTab(RecentSHnode); //먼저 메모리를 해제

int i;

for(i = 0; i < 26; i++)

{

//symbol table 탐색 포인터 노드 변수

SYMNODE \*SymCurNode = RecentSHnode[i], \*SymCopyNode = SHnode[i], \*SymNewNode;

SymNewNode = (SYMNODE\*)malloc(sizeof(SYMNODE));

if(SymCopyNode == NULL) continue;

while(SymCopyNode != NULL)

{

strcpy(SymNewNode->symbol, SymCopyNode->symbol);

//symcopynode에 정보 입력

SymNewNode->locctr = SymCopyNode->locctr;

SymCopyNode = SymCopyNode->nNode;

SymNewNode->nNode = NULL;

if(RecentSHnode[i] != NULL)

{

SymCurNode->nNode = SymNewNode;

SymCurNode = SymCurNode->nNode;

}

else

{

SymCurNode = SymNewNode;

RecentSHnode[i] = SymNewNode;

}

}

}

}

//asm 파일을 ASSEMBLE 시켜서 리스팅 파일과 오브젝트 파일로 만든다.

void Assemble()

{

switchFilename = TRUE;

if(Func\_Cal\_Fac() != TRUE)

{

printf("Error : Input Error.\n");

return;

}

//파일 입출력 변수

FILE \*f = fopen(info[0], "r");

FILE \*AssemFP = fopen("middle.txt", "w"); //중간 단계 텍스트 파일

COMNODE \*ComponentNode = (COMNODE\*)malloc(sizeof(COMNODE));

char AssemInput[100], AssemCom[4][100]; //입력 받을 때 문자열 저장하는 변수, opcode, operand 저장하는 변수

char AssemLstFile[strlen(info[0])], AssemObjFile[strlen(info[0])]; //리스팅, 오브젝트 파일 변수

//menemonic값을 저장하는 변수

char \*Assemtoken = NULL, \*hashnodename = NULL, \*AssemExtend = strtok(info[0], ".");

char AssemBufferLine[100] = {0}; //현재까지 계산된 objcode 누적하여 저장

char AssemBufferOper[100] = {0}; //해당 라인의 objcode를 계산하여 저장

char AssemBufferModi[100] = {0}; //format4이고 operand가 symbol인 경우 modifiction 내용 저장

char AssemBufferTmp[100] = {0}; //objcode를 저장하는 데에 사용되는 temp 역할

//location, locctr\_up, 시작 주소 등 변수

long AssemLocctr = 0, AssemLocctr\_Up = 0, AssemStartAddr = 0, AssemBaseLocctr= 0, AssemSizeLength = 0;

int AssemTmp = 0, AssemTmp2 = 0; //Temp 변수

int AssemSwitch1 = FALSE, AssemSwitch2 = FALSE;

int LineNumber = 0, AssemExist = 0, ExistOPcode = -1, AssemSize = 0, i = 0, j = 0, k = 0, AssemObjSize = 0, AssemObjCode = 0;

int StrlenA = 0;

//파일이 존재하지 않으면 에러 출력

if(f == NULL)

{

printf("Error : Not Exist File.\n");

return;

}

//asm 파일

AssemExtend = strtok(NULL, " ");

if(strcmp(AssemExtend, "asm"))

{

printf("Error : Not .asm File.\n");

return;

}

Func\_Free\_SymTab(SHnode);

while(fgets(AssemInput, sizeof(AssemInput), f) != NULL) //명령어 한줄 씩 입력 받는다

{

i = 0;

AssemLocctr\_Up = 0;

memset(AssemCom, '\0', sizeof(AssemCom)); //초기화

LineNumber = LineNumber + 5; //라인넘버 증가

ExistOPcode = -1;

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

//txt에 라인 번호 출력

fprintf(AssemFP, "%-5d", LineNumber);

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

{

AssemTmp = AssemTmp + LineNumber;

}

//input이 '\0'이면

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

{

AssemSwitch1 = TRUE;

AssemSwitch2 = FALSE;

fprintf(AssemFP, "\n");

continue;

}

//주석인 경우

if(AssemInput[0] == '\0' || AssemInput[0] == '.')

{

AssemSwitch1 = FALSE;

AssemSwitch2 = TRUE;

fprintf(AssemFP, "%s\n", AssemInput);

continue;

}

//공백인 경우 공백을 저장

if(AssemInput[0] == ' ' || AssemInput[0] == '\t')

{

AssemSwitch1 = FALSE;

AssemSwitch2 = FALSE;

strcpy(AssemCom[0]," ");

i = 1;

}

Assemtoken = strtok(AssemInput," \t");//token이 opcode 시작주소 가리킨다.

AssemTmp = i;

//AssemCom에 opcode, oprand를 저장하고 다음 token을 입력 받는다.

while(Assemtoken != NULL)

{

if(i >= 4)

{

printf("Error : %d Line Error. (OPERAND 개수)\n", LineNumber);

return;

}

AssemSizeLength = AssemTmp;

strcpy(AssemCom[i],Assemtoken);

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

i = i + 1;

}

//범위 이탈시 에러 출력

if(AssemLocctr < 0 || AssemLocctr > MAX\_\_NUM || AssemStartAddr < 0)

{

AssemSizeLength = AssemLocctr;

printf("Error : Not Range Address.\n");

return;

}

//symbol이 공백이 아닌경우 이미 테이블에 존재하기 때문에 에러 출력

if(strcmp(AssemCom[0], " "))

{

if(strcmp(AssemCom[1], "START")) AssemExist = Func\_Make\_Symbol(AssemCom[0], AssemLocctr);

if(AssemExist == TRUE)

{

AssemTmp = 1;

}

if(AssemExist == -1)

{

printf("Error : %d Line Error.", LineNumber);

printf("%s is Exist in SymbolTable.\n", AssemCom[0]);

return;

}

}

//format4 형식은 +를 제외한 mnemonic 문자열 을 저장한다.

if(AssemCom[1][0] == '+')

{

AssemSwitch2 = TRUE;

hashnodename = (char\*)malloc(sizeof(AssemCom[1]));

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

}

else

{

AssemSwitch2 = TRUE;

hashnodename = (char\*)malloc(sizeof(AssemCom[1]));

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

hashnodename[i] = '\0';

}

//#, @으로 시작하는데 Operand가 2개 존재하면 에러 출력

if(AssemCom[2][0] == '#' || AssemCom[2][0] == '@')

if(strcmp(AssemCom[3], "\0")==TRUE)

{

printf("Error : %d Line Error. Not Exist Second Factor.\n", LineNumber);

return;

}

HASHNODE \*find\_node = OPHnode[(hashnodename[0] + hashnodename[strlen(hashnodename)-1]) %20];

while(find\_node != NULL)

{

if(strcmp(find\_node->hashnodename, hashnodename)==FALSE)

{

HashTargetNode = find\_node;

ExistOPcode = find\_node->opcode;

}

find\_node = find\_node->nNode;

}

//opcode table에 존재하지 않는 경우

if(ExistOPcode == -1)

{

if(strcmp(AssemCom[2], "\0")==FALSE) //첫번째 operand가 없을 경우 에러 출력

{

printf("Error : %d Line Error. Not Exist Operand.\n", LineNumber);

return;

}

if(strcmp(AssemCom[3], "\0")==TRUE) //두번째 operand가 없을 경우 에러 출력

{

printf("Error : %d Line Error. Only 1 Operand Possible.\n", LineNumber);

return;

}

if(strcmp(AssemCom[1], "START")==FALSE)

{

if(LineNumber == 5)

{

AssemSizeLength = strlen(AssemCom[2]);

for(i = 0; i < strlen(AssemCom[2]); i = i + 1)

if(!(('0' <= AssemCom[2][i] && AssemCom[2][i]<= '9') || ('A' <= AssemCom[2][i] && AssemCom[2][i] <= 'F') || ('a' <= AssemCom[2][i] && AssemCom[2][i] <= 'f')))

{

printf("Error : %d Line Error.", LineNumber);

printf("Start Address는 16진수만 가능합니다.\n");

return;

}

//입력 받은 명령어가 16진수가 아닌 경우 에러 출력

for(int q = 0; q < AssemSizeLength; q = q + 1)

{

AssemTmp = q;

AssemTmp2 = AssemTmp;

AssemTmp = FALSE;

}

AssemStartAddr = strtoul(AssemCom[2], NULL, 16);

AssemLocctr = AssemStartAddr;

AssemTmp2 = AssemStartAddr;

fprintf(AssemFP, "%04lX %-8s %-8s %-8s\n", AssemLocctr, AssemCom[0], AssemCom[1], AssemCom[2]);

continue;

}

else

{

printf("Error : %d Line Error.", LineNumber);

printf("Not START in Middle of Program.\n");

return;

}

}

else if(strcmp(AssemCom[1], "WORD")==FALSE)

{

//WORD 범위 초과하면 에러

if(!(0 < atoi(AssemCom[2]) && atoi(AssemCom[2]) < 0xFFFFFF))

{

printf("Error : %d Line Error. WORD는 3byte를 초과할 수 없습니다.\n", LineNumber);

return;

}

StrlenA = strlen(AssemCom[2]);

for(i = 0; i < strlen(AssemCom[2]); i = i + 1) //숫자가 아닐 경우 에러

if(AssemCom[2][i] < '0' && AssemCom[2][i] >'9')

{

printf("Error : %d Line Error.", LineNumber);

printf("WORD is only Number.\n");

return;

}

AssemLocctr\_Up = 3;

}

else if(strcmp(AssemCom[1], "BYTE")==FALSE)

{

if(AssemCom[2][0] == 'X')

{

StrlenA = strlen(AssemCom[2])-3;//strlen(AssemCom[2])-3 는 작은 따옴표 사이 길이를 말한다.

if(strlen(AssemCom[2])-3 > MAXE)

{

printf("Error : %d Line Error.", LineNumber);

printf("MAX 30 byte.\n");

return;

}

for(i = 2; i < strlen(AssemCom[2])-1; i = i + 1) //16진수가 아닌 경우 에러 출력

if(!(('0' <= AssemCom[2][i] && AssemCom[2][i] <= '9') || ('A' <= AssemCom[2][i] && AssemCom[2][i] <= 'F') || ('a' <= AssemCom[2][i] && AssemCom[2][i] <= 'f')))

{

printf("Error : %d Line Error.", LineNumber);

printf("Not Hexademical.\n");

return;

}

if((strlen(AssemCom[2])-3)%2 != 0)//invalid operand

{

printf("Error : %d Line Error.", LineNumber);

printf("Invalied Operand.\n");

return;

}

AssemLocctr\_Up = (strlen(AssemCom[2])-3) / 2;

}

else if(AssemCom[2][0] == 'C')

{

if(strlen(AssemCom[2])-3 > MAXE) //0Xffff 범위를 넘었을 경우

{

printf("Error : %d Line Error.", LineNumber);

printf("MAX 30 byte.\n");

return;

}

for(i = 2; i < strlen(AssemCom[2])-1; i = i + 1)

if(!(0x20 <= AssemCom[2][i] && AssemCom[2][i] <= 0x7E)) //16진수가 아니므로 에러 출/

{

printf("Error : %d Line Error.", LineNumber);

printf("Not Hexademical.\n");

return;

}

AssemLocctr\_Up = strlen(AssemCom[2])-3;

}

else

{

printf("Error : %d Line Error.", LineNumber);

printf("Byte Format Error.\n");

return;

}

}

else if(strcmp(AssemCom[1], "RESW")==FALSE)

{

for(i = 0; i < strlen(AssemCom[2]); i = i + 1)

if('0' > AssemCom[2][i] || AssemCom[2][i] > '9')

{

printf("Error : %d Line Error.", LineNumber);

printf("10 진수가 아닙니다.\n");

return;

}

AssemLocctr\_Up = atoi(AssemCom[2]) \* 3;

}

else if(strcmp(AssemCom[1], "RESB")==FALSE)

{

for(i = 0; i < strlen(AssemCom[2]); i = i + 1)

if('0' > AssemCom[2][i] || AssemCom[2][i] > '9')

{

printf("Error : %d Line Error.", LineNumber);

printf("10 진수가 아닙니다.\n");

return;

}

AssemLocctr\_Up = atoi(AssemCom[2]);

}

else if(strcmp(AssemCom[1], "BASE")==FALSE)

{

for(i = 0; i < 14; i = i + 1) fprintf(AssemFP, " ");

fprintf(AssemFP, "%-8s %s\n", AssemCom[1], AssemCom[2]);

continue;

}

else if(strcmp(AssemCom[1], "END")==FALSE)

{

if(Func\_Symbol\_Int(AssemCom[2]) != AssemStartAddr)

{

printf("Error : %d Line Error.", LineNumber);

printf("Error in END Operand.\n");

return;

}

AssemSize = (int)(AssemLocctr - AssemStartAddr); //assemsize에 프로그램 전체 크기 저장

for(i = 0; i < 14; i = i + 1) fprintf(AssemFP, " ");

fprintf(AssemFP, "%-8s %s", AssemCom[1], AssemCom[2]);

break;

}

else

{

printf("Error : %d Line Error.", LineNumber);

printf("Invalied Input.\n");

return;

}

}

else //opcode 가 opcode table에 존재하는 경

{

if(strcmp(AssemCom[1], "RSUB")==TRUE && strcmp(AssemCom[2], "\0")==FALSE && strcmp(HashTargetNode->hashformat, "1")==TRUE)

{

printf("Error : %d Line Error.", LineNumber);

printf("Not Exist Operand.\n");

return;

}

if(strcmp(HashTargetNode->hashformat, "1")==FALSE)//format 1 이므로 location counter 1 증가

{

if(strcmp(AssemCom[2], "\0"))

{

printf("Error : %d Line Error.", LineNumber);

printf("Not Exist Operand.\n");

return;

}

AssemLocctr\_Up = 1;

}

else if(strcmp(HashTargetNode->hashformat, "2")==FALSE) AssemLocctr\_Up = 2; //format 2 이므로 location counter 2 증가

else if(strcmp(HashTargetNode->hashformat, "3/4")==FALSE)

{

//두번째 operand가 X 또는 공백이 아니면 에러 출력

if(strcmp(AssemCom[3], "\0")==TRUE && strcmp(AssemCom[3], "X")==TRUE)

{

printf("Error : %d Line Error.", LineNumber);

printf("Invalid Second Operand.\n");

return;

}

//두번째 operand 가 없을 경우 에러 출력

if((strcmp(AssemCom[1], "STCH")==FALSE || !strcmp(AssemCom[1], "LDCH")) && strcmp(AssemCom[3], "X")==TRUE)

{

printf("Error : %d Line Error.", LineNumber);

printf("Invalid Second Operand.\n");

}

if(AssemCom[1][0] != '+') AssemLocctr\_Up = 3; //format 3 이므로 location counter 3 증가

else AssemLocctr\_Up = 4; //format 4 이므로 location counter 4 증가

}

}

fprintf(AssemFP, "%04lX %-8s %-8s %s %s\n", AssemLocctr, AssemCom[0], AssemCom[1], AssemCom[2], AssemCom[3]);

AssemLocctr = AssemLocctr + AssemLocctr\_Up;

}

//메모리 해제

fclose(f);

fclose(AssemFP);

free(hashnodename);

free(Assemtoken);

strcpy(AssemObjFile, strtok(info[0], "."));

strcpy(AssemLstFile, strtok(info[0], "."));

strcat(AssemObjFile, ".obj");

strcat(AssemLstFile, ".lst");

//연산에 사용되는 Temp 변

int AssemPass2Temp = 0, AssemPass2Temp2 = 0, AssemPass2Legnth = 0;

int AssemPassSwitch = FALSE, AssemPass2End = FALSE;

//lst, obj file에 대한 파일 포인터 변수

FILE \*AssemLstfp = fopen(AssemLstFile, "w");

FILE \*AssemObjfp = fopen(AssemObjFile, "w");

AssemFP = fopen("middle.txt", "r");

while(fgets(AssemInput, sizeof(AssemInput), AssemFP) != NULL)//명령어 한줄씩 입력

{

//파일이 끝날 때 까지 입력 받는다

memset(ComponentNode, '\0', sizeof(\*ComponentNode)); //초기

AssemPass2Legnth = strlen(AssemInput);

for(i = 0; i < strlen(AssemInput); i = i + 1)

{

//input[i] : i = 0,5,10,19,28 마다 새로운 인자가 존재

if(i == 0)

{

for(k = 0; AssemInput[k] != ' '; k = k + 1) ComponentNode->LineNumber[k] = AssemInput[k];

ComponentNode->LineNumber[k] = '\0';

for(int qq = 0; qq < 26; qq = qq + 1)

{

AssemPass2Temp = AssemPass2Temp + qq;

}

AssemPassSwitch = TRUE;

}

else if(i == 5)

{

for(k = i; (AssemInput[k] != ' ') && (AssemInput[k] != '\n'); k = k + 1) ComponentNode->Comloc[k-5] = AssemInput[k];

ComponentNode->Comloc[k-5] = '\0';

for(int qq = 0; qq < 26; qq = qq + 1)

{

AssemPass2Temp = AssemPass2Temp + qq;

}

AssemPassSwitch = TRUE;

}

else if(i == 10)

{

for(k = i; AssemInput[k] != ' '; k = k + 1) ComponentNode->symbol[k-10] = AssemInput[k];

ComponentNode->symbol[k-10] = '\0';

for(int qq = 0; qq < 26; qq = qq + 1)

{

AssemPass2Temp = AssemPass2Temp + qq;

}

AssemPassSwitch = TRUE;

}

else if(i == 19)

{

for(k = i; AssemInput[k] != ' '; k = k + 1) ComponentNode->opcode[k-19] = AssemInput[k];

ComponentNode->opcode[k-19] = '\0';

for(int qq = 0; qq < 26; qq = qq + 1)

{

AssemPass2Temp = AssemPass2Temp + qq;

}

AssemPassSwitch = TRUE;

}

else if(i == 28)

{

for(k = i; (AssemInput[k] != ' ') && (AssemInput[k] != ',') && (AssemInput[k] != '\n'); k = k + 1) ComponentNode->ComOper1[k-28] = AssemInput[k];

ComponentNode->ComOper1[k-28] = '\0';

for(int qq = 0; qq < 26; qq = qq + 1)

{

AssemPass2Temp = AssemPass2Temp + qq;

}

AssemPassSwitch = TRUE;

if(AssemInput[k] == ',') //operand가 2개 있는 경우

{

for(k = k+1; k < strlen(AssemInput); k = k + 1)

if(AssemInput[k] != ' ') break;

int operand\_2\_start = k;

for(; (k < strlen(AssemInput)) && (AssemInput[k] != '\n'); k = k + 1) ComponentNode->ComOper2[k-operand\_2\_start] = AssemInput[k];

ComponentNode->ComOper2[k-operand\_2\_start] = '\0';

}

else strcpy(ComponentNode->ComOper2, "\0");

}

if(AssemPassSwitch == TRUE)

{

AssemPass2End = TRUE;

}

else AssemPass2End = FALSE;

}

//START가 없는 경우

if(strcmp(ComponentNode->LineNumber, "5")==FALSE && strcmp(ComponentNode->opcode, "START")==TRUE)

{

fprintf(AssemObjfp, "H%-6s%06lX%06X\n", " ", AssemStartAddr, AssemSize);

}

if(strcmp(ComponentNode->opcode, "START")==FALSE) //START

{

fprintf(AssemLstfp, "%s", AssemInput);

fprintf(AssemObjfp, "H%-6s%06lX%06X\n", ComponentNode->symbol, AssemStartAddr, AssemSize);

continue;

}

else if(strcmp(ComponentNode->opcode, "END")==FALSE) //END

{

fprintf(AssemLstfp, "%s", AssemInput);

fprintf(AssemObjfp, "%02X", AssemObjSize);

fprintf(AssemObjfp, "%s\n", AssemBufferLine);

AssemPass2Legnth = strlen(AssemBufferModi)-1;

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

//null값 저장 후 파일 출력

fprintf(AssemObjfp, "%s\n", AssemBufferModi);

fprintf(AssemObjfp, "E%06lX", AssemStartAddr);

sprintf(AssemBufferModi, "NULL");

break;

}

else

{

//주석의 경우 주석 처리 해줌

if(strcmp(ComponentNode->Comloc, ".")==FALSE)

{

fprintf(AssemLstfp, "%s", AssemInput);

continue;

}

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

fprintf(AssemLstfp, "%s", AssemInput);

//lst file에서 object code line 맞추는 용도

for(k = 0; k < 43 - strlen(AssemInput); k++) fprintf(AssemLstfp, " ");

for(int qq = 0; qq < 26; qq = qq + 1)

{

AssemPass2Temp = AssemPass2Temp + qq;

}

AssemPassSwitch = TRUE;

int hashformat = 0; //format 4인지 아닌지 구분하기 위해서 사용되는 변수

if(ComponentNode->opcode[0] == '+') //+면

{

hashformat = 4; //4형식

AssemPass2Legnth = strlen(ComponentNode->opcode)-1;

for(k = 0; k < strlen(ComponentNode->opcode)-1; k = k + 1) ComponentNode->opcode[k] = ComponentNode->opcode[k+1];

ComponentNode->opcode[k] = '\0';

}

ExistOPcode = -1;

HASHNODE \*find\_node = OPHnode[((ComponentNode->opcode)[0] + (ComponentNode->opcode)[strlen((ComponentNode->opcode))-1]) %20];

while(find\_node != NULL)

{

if(strcmp(find\_node->hashnodename, ComponentNode->opcode)==FALSE)

{

HashTargetNode = find\_node;

ExistOPcode = find\_node->opcode;

AssemPassSwitch = TRUE;

}

find\_node = find\_node->nNode;

}

int AssemCountt = 0;

AssemLocctr = strtoul(ComponentNode->Comloc, NULL, 16);

if(!AssemObjSize) //오브젝트 파일 각 줄의 첫번째인 경우

{

if(strcmp(ComponentNode->opcode, "RESB") && strcmp(ComponentNode->opcode, "RESW") && strcmp(ComponentNode->Comloc, ".")) fprintf(AssemObjfp, "T%06lX", AssemLocctr);

AssemCountt = AssemCountt + 1;

}

if(strcmp(ComponentNode->opcode, "BASE")==FALSE)

{

AssemBaseLocctr = Func\_Symbol\_Int(ComponentNode-> ComOper1);

if(AssemBaseLocctr == -1)

{

printf("Error : %s line Error. Not BASE in SYMTAB.\n", ComponentNode->LineNumber);

fclose(AssemLstfp);

fclose(AssemObjfp);

remove(AssemLstFile);

remove(AssemObjFile);

return;

}

}

if(ExistOPcode == -1) //오브젝트 테이블에 없다면

{

if(strcmp(ComponentNode->opcode, "RESB")==FALSE || (strcmp(ComponentNode->opcode, "RESW"))==FALSE)

{

if(AssemObjSize != 0) //변수인 경우 object code 없음

{

fprintf(AssemObjfp, "%02X", AssemObjSize);

fprintf(AssemObjfp, "%s\n", AssemBufferLine);

memset(AssemBufferLine, '\0', sizeof(AssemBufferLine));

AssemObjSize = 0;

}

}

else if(strcmp(ComponentNode->opcode, "WORD")==FALSE)

{

fprintf(AssemLstfp, "%06X", atoi(ComponentNode->ComOper1));

sprintf(AssemBufferOper, "%06X", atoi(ComponentNode->ComOper1));

AssemObjSize = AssemObjSize + 3;

//0x1E 초과하면 obj file에 현재까지 저장된 버퍼 라인까지 출력

if(AssemObjSize > MAXE)

{

//초과하기 전까지 저장된 objsize 저장

AssemObjSize = AssemObjSize - strlen(ComponentNode->ComOper1)-3;

fprintf(AssemObjfp, "%02X", AssemObjSize);

fprintf(AssemObjfp, "%s\n", AssemBufferLine);

fprintf(AssemObjfp, "T%06lX", AssemLocctr);

//각 인자들 출력

memset(AssemBufferLine, '\0', sizeof(AssemBufferLine));

AssemObjSize = 3;

}

strcat(AssemBufferLine, AssemBufferOper);

}

else if(strcmp(ComponentNode->opcode, "BYTE")==FALSE)

{

memset(AssemBufferOper, '\0', sizeof(AssemBufferOper));

int siPass2 = strlen(ComponentNode->ComOper1);

for(k = 2; k < strlen(ComponentNode->ComOper1)-1; k = k + 1) //X' ' 또는 C' ' 제외한 값 출력

{

for(int ww=0; ww < k; ww ++)

{

siPass2 = siPass2 + ww;

}

if(ComponentNode->ComOper1[0] == 'X') //16진수로 출력한다/

{

AssemPassSwitch = TRUE;

sprintf(AssemBufferTmp, "%c", ComponentNode->ComOper1[k]);

strcat(AssemBufferOper, AssemBufferTmp);

fprintf(AssemLstfp, "%c", ComponentNode->ComOper1[k]);

}

else if(ComponentNode->ComOper1[0] == 'C') //문자열 이기 때문에 아스키코드를 사용하여 출력한다.

{

AssemPassSwitch = TRUE;

sprintf(AssemBufferTmp, "%X", ComponentNode->ComOper1[k]);

strcat(AssemBufferOper, AssemBufferTmp);

fprintf(AssemLstfp, "%X", ComponentNode->ComOper1[k]);

}

AssemPass2Temp = siPass2;

}

if(ComponentNode->ComOper1[0] == 'X') AssemObjSize = AssemObjSize + (strlen(ComponentNode->ComOper1)-3)/2;

else AssemObjSize = AssemObjSize + strlen(ComponentNode->ComOper1)-3;

//0x1E 초과하면

if(AssemObjSize > MAXE)

{

//초과하기 전까지만 저장

if(ComponentNode->ComOper1[0] == 'X') AssemObjSize = AssemObjSize - (strlen(ComponentNode->ComOper1)-3)/2;

else AssemObjSize = AssemObjSize - strlen(ComponentNode->ComOper1)-3;

AssemPassSwitch = TRUE;

fprintf(AssemObjfp, "%02X", AssemObjSize);

fprintf(AssemObjfp, "%s\n", AssemBufferLine);

fprintf(AssemObjfp, "T%06lX", AssemLocctr);

for(int ww=0; ww < k; ww ++)

{

siPass2 = siPass2 + ww;

}

memset(AssemBufferLine, '\0', sizeof(AssemBufferLine));

if(ComponentNode->ComOper1[0] == 'X') AssemObjSize = (strlen(ComponentNode->ComOper1)-3)/2;

else AssemObjSize = strlen(ComponentNode->ComOper1)-3;

if(AssemPassSwitch==TRUE)

{

AssemPass2Temp = AssemPass2Temp + siPass2;

}

}

strcat(AssemBufferLine, AssemBufferOper);

}

fprintf(AssemLstfp, "\n");

}

else

{

int countPass = 0;

AssemObjCode = HashTargetNode->opcode;

Assemtoken = (char\*)malloc(sizeof(ComponentNode->ComOper1));

strcpy(Assemtoken, ComponentNode->ComOper1);

if(strcmp(ComponentNode->opcode, "RSUB")==FALSE) //simple addressing, opcode 값만 더하기

{

AssemObjCode = AssemObjCode + 3;

AssemObjCode = AssemObjCode << 16;

countPass = countPass + AssemObjCode;

fprintf(AssemLstfp, "%06X\n", AssemObjCode);

AssemObjSize = AssemObjSize + 3;

int newCopy = 0;

//0x1E 초과하

if(AssemObjSize > MAXE)

{

AssemPassSwitch = TRUE;

AssemObjSize = AssemObjSize - 3;

fprintf(AssemObjfp, "%02X", AssemObjSize);

fprintf(AssemObjfp, "%s\n", AssemBufferLine);

fprintf(AssemObjfp, "T%06lX", AssemLocctr);

newCopy = AssemObjSize;

memset(AssemBufferLine, '\0', sizeof(AssemBufferLine));

AssemObjSize = 3;

}

newCopy = newCopy + 1;

sprintf(AssemBufferOper, "%06X", AssemObjCode);

strcat(AssemBufferLine, AssemBufferOper);

AssemObjCode = FALSE;

continue;

}

//indirect mode

if(ComponentNode->ComOper1[0] == '@')

{

AssemPassSwitch = TRUE;

Assemtoken = strtok(Assemtoken, "@");

AssemObjCode = AssemObjCode + 2; //2형식

}

else if (ComponentNode->ComOper1[0] == '#') //immediate mode

{

AssemPassSwitch = TRUE;

Assemtoken = strtok(Assemtoken, "#");

AssemObjCode = AssemObjCode + 1; //1형식

}

else AssemObjCode = AssemObjCode + 3; //3형식

int BooleanSymbol = Func\_Symbol\_Int(Assemtoken);

//첫번째 operand가 symbol이면 symbol의 locctr, 없으면 ERROR

int BoolSym = FALSE;

//operand2가 존재하는데 operand1이 심볼이면 에러를 출력한다.

if((strcmp(ComponentNode->ComOper2, "\0")) && (BooleanSymbol != -1) && (strcmp(ComponentNode->ComOper1, "BUFFER")))

{

AssemPassSwitch = TRUE;

fclose(AssemLstfp);

fclose(AssemObjfp);

BoolSym = AssemPassSwitch;

printf("Error : %s line Error.", ComponentNode->LineNumber);

printf("Invalid Second Operand.\n");

remove(AssemLstFile);

remove(AssemObjFile);

return;

}

//immediate mode인데 symbol이 아니면 에러 출력

if((ComponentNode->ComOper1[0] == '#') && (BooleanSymbol == -1))

{

AssemPassSwitch = TRUE;

for(i = 0; i < strlen(Assemtoken); i = i + 1)

if(!('0' <= Assemtoken[i] && Assemtoken[i] <= '9')) //10진수 아니면 에러

{

fclose(AssemLstfp);

fclose(AssemObjfp);

BoolSym = AssemPassSwitch;

printf("Error : %s line Error.", ComponentNode->LineNumber);

printf("Invalid Operand.\n");

remove(AssemLstFile);

remove(AssemObjFile);

return;

}

}

int cntObjCode = 0;

if(hashformat == 4)

{

AssemObjCode = AssemObjCode << 4;

AssemObjCode = AssemObjCode + 1;

if(strcmp(ComponentNode->ComOper2, "X")==FALSE) AssemObjCode = AssemObjCode + 8;

AssemObjCode = AssemObjCode << 20;

if(BooleanSymbol != -1)

{

AssemPassSwitch = TRUE;

AssemObjCode = AssemObjCode + BooleanSymbol;

sprintf(AssemBufferTmp, "M%06lX05\n", AssemLocctr+1);

BoolSym = AssemLocctr;

strcat(AssemBufferModi, AssemBufferTmp);

}

else if(ComponentNode->ComOper1[0] == '#') AssemObjCode = AssemObjCode + atoi(Assemtoken);

else

{

AssemPassSwitch = TRUE;

fclose(AssemLstfp);

fclose(AssemObjfp);

printf("Error : %s line Error. Invalid Operand.\n", ComponentNode->LineNumber);

remove(AssemLstFile);

remove(AssemObjFile);

return;

}

AssemObjSize = AssemObjSize + 4;

if(AssemObjSize > MAXE)

{

AssemObjSize = AssemObjSize - 4;

fprintf(AssemObjfp, "%02X", AssemObjSize);

fprintf(AssemObjfp, "%s\n", AssemBufferLine);

fprintf(AssemObjfp, "T%06lX", AssemLocctr);

AssemObjSize = 4;

memset(AssemBufferLine, '\0', sizeof(AssemBufferLine));

}

sprintf(AssemBufferOper, "%06X", AssemObjCode);

strcat(AssemBufferLine, AssemBufferOper);

}

else if(strcmp(HashTargetNode->hashformat, "3/4")==FALSE)

{

AssemObjCode = AssemObjCode << 4;

//첫번째 operand가 symbol이 아닌 경우

if(BooleanSymbol == -1 && ComponentNode->ComOper1[0] == '#')

{

AssemPassSwitch = TRUE;

AssemObjCode = AssemObjCode << 12;

countPass = AssemObjCode;

AssemObjCode = AssemObjCode + atoi(Assemtoken);

}

else if(BooleanSymbol != -1)

{

//첫번째 operand가 symbol이면 symbol의 LOC, 없으면 ERROR 반환

if(strcmp(ComponentNode->ComOper2, "X")==FALSE) AssemObjCode = AssemObjCode + 8;

long Num\_Disp = BooleanSymbol - (AssemLocctr+3);

countPass = FALSE;

if(-2048 <= Num\_Disp && Num\_Disp <= 2047)

{

if(Num\_Disp < 0) Num\_Disp = 0x1000 + Num\_Disp; //2의 보수로 변환시켜준다.

AssemObjCode = AssemObjCode + 2;

}

else if(Num\_Disp <= 4095) //base relative

{

Num\_Disp = BooleanSymbol - AssemBaseLocctr;

AssemObjCode = AssemObjCode + 4;

}

AssemObjCode = AssemObjCode << 12;

AssemObjCode = AssemObjCode + Num\_Disp;

}

else //첫번째 operand가 symbol도 아니고 '#'도 아니면 에러

{

fclose(AssemObjfp);

fclose(AssemLstfp);

printf("Error : %s line Error. ", ComponentNode->LineNumber);

printf("Invalid Operand.\n");

remove(AssemLstFile);

remove(AssemObjFile);

return;

}

AssemObjSize = AssemObjSize + 3;

if(AssemObjSize > MAXE)

{

AssemObjSize = AssemObjSize - 3;

fprintf(AssemObjfp, "%02X", AssemObjSize);

fprintf(AssemObjfp, "%s\n", AssemBufferLine);

fprintf(AssemObjfp, "T%06lX", AssemLocctr);

memset(AssemBufferLine, '\0', sizeof(AssemBufferLine));

AssemObjSize = 3;

}

sprintf(AssemBufferOper, "%06X", AssemObjCode);

strcat(AssemBufferLine, AssemBufferOper);

}

else if(strcmp(HashTargetNode->hashformat, "2")==FALSE)

{

AssemObjCode = AssemObjCode - 3;

AssemObjCode = AssemObjCode << 4;

if((ComponentNode->ComOper1[0] == '#') && !(strcmp(ComponentNode->ComOper2, "\0"))) {

strcpy(Assemtoken, ComponentNode->ComOper1);

Assemtoken = strtok(Assemtoken, "#");

if(atoi(Assemtoken) > 15) //2형식에서 10진수 입력 범위가 15이기 때문에 넘어가면 에러 출력

{

fclose(AssemObjfp);

fclose(AssemLstfp);

printf("Error : %s line Error. ", ComponentNode->LineNumber);

printf("Invalid Range of Operand in Format2.\n");

remove(AssemLstFile);

remove(AssemObjFile);

return;

}

AssemObjCode = AssemObjCode + atoi(Assemtoken);

}

else if(strcmp(ComponentNode->ComOper1, "A")==FALSE)

{

}

else if(strcmp(ComponentNode->ComOper1, "X")==FALSE) AssemObjCode = AssemObjCode + 1;

else if(strcmp(ComponentNode->ComOper1, "L")==FALSE) AssemObjCode = AssemObjCode + 2;

else if(strcmp(ComponentNode->ComOper1, "B")==FALSE) AssemObjCode = AssemObjCode + 3;

else if(strcmp(ComponentNode->ComOper1, "S")==FALSE) AssemObjCode = AssemObjCode + 4;

else if(strcmp(ComponentNode->ComOper1, "T")==FALSE) AssemObjCode = AssemObjCode + 5;

else if(strcmp(ComponentNode->ComOper1, "F")==FALSE) AssemObjCode = AssemObjCode + 6;

else if(strcmp(ComponentNode->ComOper1, "PC")==FALSE) AssemObjCode = AssemObjCode + 8;

else if(strcmp(ComponentNode->ComOper1, "SW")==FALSE) AssemObjCode = AssemObjCode + 9;

else

{

fclose(AssemObjfp);

fclose(AssemLstfp);

printf("Error : %s line Error. ", ComponentNode->LineNumber);

printf("Invalid Operand in Format2.\n");

remove(AssemLstFile);

remove(AssemObjFile);

return;

}

AssemObjCode = AssemObjCode << 4;

if(strcmp(ComponentNode->ComOper2, "\0")==TRUE)

{

if(ComponentNode->ComOper2[0] == '#')

{

strcpy(Assemtoken, ComponentNode->ComOper2);

Assemtoken = strtok(Assemtoken, "#");

if(atoi(Assemtoken) > 15) //2형식에서 10진수 입력 범위가 15이기 때문에 넘어가면 에러 출력

{

AssemPassSwitch = TRUE;

fclose(AssemObjfp);

fclose(AssemLstfp);

printf("Error : %s line Error. ", ComponentNode->LineNumber);

printf("Invalid Operand in Format2.\n");

remove(AssemLstFile);

remove(AssemObjFile);

return;

}

AssemObjCode = AssemObjCode + atoi(Assemtoken);

}

else if(strcmp(ComponentNode->ComOper2, "A")==FALSE)

{

}

else if(strcmp(ComponentNode->ComOper2, "X")==FALSE) AssemObjCode = AssemObjCode + 1;

else if(strcmp(ComponentNode->ComOper2, "L")==FALSE) AssemObjCode = AssemObjCode + 2;

else if(strcmp(ComponentNode->ComOper2, "B")==FALSE) AssemObjCode = AssemObjCode + 3;

else if(strcmp(ComponentNode->ComOper2, "S")==FALSE) AssemObjCode = AssemObjCode + 4;

else if(strcmp(ComponentNode->ComOper2, "T")==FALSE) AssemObjCode = AssemObjCode + 5;

else if(strcmp(ComponentNode->ComOper2, "F")==FALSE) AssemObjCode = AssemObjCode + 6;

else if(strcmp(ComponentNode->ComOper2, "PC")==FALSE) AssemObjCode = AssemObjCode + 8;

else if(strcmp(ComponentNode->ComOper2, "SW")==FALSE) AssemObjCode = AssemObjCode + 9;

else

{

AssemPassSwitch = TRUE;

fclose(AssemObjfp);

fclose(AssemLstfp);

printf("Error : %s line Error. ", ComponentNode->LineNumber);

printf("Invalid Operand in Format2.\n");

remove(AssemLstFile);

remove(AssemObjFile);

return;

}

}

AssemObjSize = AssemObjSize + 2;

int sizeObjCode = 0;

if(AssemObjSize > MAXE)

{

AssemObjSize = AssemObjSize - 2;

fprintf(AssemObjfp, "%02X", AssemObjSize);

fprintf(AssemObjfp, "%s\n", AssemBufferLine);

fprintf(AssemObjfp, "T%06lX", AssemLocctr);

sizeObjCode = sizeof(AssemBufferLine);

memset(AssemBufferLine, '\0', sizeof(AssemBufferLine));

AssemObjSize = 2;

}

sizeObjCode = sizeObjCode + 1;

fprintf(AssemLstfp, "%04X\n", AssemObjCode);

sprintf(AssemBufferOper, "%04X", AssemObjCode);

strcat(AssemBufferLine, AssemBufferOper);

continue;

}

else if(strcmp(HashTargetNode->hashformat, "1")==FALSE)

{

AssemObjCode = AssemObjCode - 3;

AssemObjSize = AssemObjSize + 1;

sprintf(AssemBufferOper, "%02X", AssemObjCode);

if(AssemObjSize > MAXE)

{

AssemObjSize = AssemObjSize - 1;

fprintf(AssemObjfp, "%02X", AssemObjSize);

fprintf(AssemObjfp, "%s\n", AssemBufferLine);

fprintf(AssemObjfp, "T%02lX", AssemLocctr);

memset(AssemBufferLine, '\0', sizeof(AssemBufferLine));

AssemObjSize = 1;

}

cntObjCode = AssemObjSize;

fprintf(AssemLstfp, "%02X\n", AssemObjCode);

strcat(AssemBufferLine, AssemBufferOper);

continue;

}

cntObjCode = cntObjCode + 1;

fprintf(AssemLstfp, "%06X\n", AssemObjCode);

AssemObjCode = 0;

}

}

}

if(strcmp(AssemBufferModi, "NULL")) //END가 없기 때문에 에러 출력

{

AssemPassSwitch = TRUE;

fclose(AssemLstfp);

fclose(AssemObjfp);

printf("Error : File Error.");

printf("Not Exist END.\n");

remove(AssemLstFile);

remove(AssemObjFile);

return;

}

printf("Successfully assemble %s.asm.\n", info[0]);

fclose(AssemLstfp);

fclose(AssemObjfp);

free(ComponentNode);

AssemSwitch2 = TRUE;

Func\_History\_Save();

Func\_Copy\_SymTab();

Func\_Free\_SymTab(SHnode);

}

//파일의 내용을 출력하는 함수

void Func\_Type()

{

int typenum = 0;

switchFilename = TRUE;

if(Func\_Cal\_Fac() != TRUE)

{

printf("Error : Input Error.\n");

return;

}

char TypeValue, \*filename = info[0];

FILE \*fp\_type = fopen(filename, "r");

if(fp\_type == NULL)

{

printf("Error : Not Exsit File.\n");

typenum = typenum + 1;

return;

}

//입력 받은 파일의 확장자를 저장한다.

char \*TypeVar = strtok(info[0], ".");

TypeVar = strtok(NULL, " ");

//파일이 아닌 경우 에러를 출력한다.

if(TypeVar == NULL && strcmp(info[0], "README")==TRUE && strcmp(info[0], "Makefile")==TRUE)

{

printf("Error : Not File.\n");

typenum = typenum + 1;

return;

}

while((TypeValue = fgetc(fp\_type)) != EOF) printf("%c", TypeValue);

printf("\n");

switchFilename = FALSE;

fclose(fp\_type);

Func\_History\_Save();

}

//loader, run이 실행될 때 시작주소를 지정하는 함수이다.

//첫 시작 시 기본으로 0 주소를 가진다.

void Func\_Progaddr()

{

//인자가 유효하지 않으면 에러

if(Func\_Cal\_Fac() != TRUE)

{

printf("Error : Input Error.\n");

return;

}

//메모리 범위를 벗어난 경우 에러

ProgaAddress = strtoul(info[0], NULL, HEX);

if(ProgaAddress < 0 || ProgaAddress > MAX\_\_NUM)

{

ProgaAddress = FALSE;

printf("Error : Not Range ProgaAddress.\n");

return;

}

Func\_History\_Save();

}

//assemble 과정에서 생기는 symbol table을 출력한다.

void Func\_Symbol()

{

int i;

for(i = 0; i < 26; i = i + 1)

{

if(RecentSHnode[i] == NULL) continue;

SYMNODE \*SymPrintNode = RecentSHnode[i];

while(SymPrintNode != NULL)

{

printf("\t%s\t%04X\n", SymPrintNode->symbol, SymPrintNode->locctr);

SymPrintNode = SymPrintNode->nNode;

}

}

Func\_History\_Save();

}

//Extend Symbol Table을 생성한다.

//csec와 symbol 이 존재하지 않으면 테이블에 정보를 넣고, 존재하면 에러를 출력한다.

int Func\_HashTab(char EstabCsec[], char symbol[], int HashCurAddr, int HashFile)

{

//Extend Symbol Table 탐색시 사용되는 포인터 노드 변수

ESTABNODE \*HashEstabNode;

ESTABNODE \*HashEstabInputNode = (ESTABNODE\*)malloc(sizeof(ESTABNODE));

int i, j, HashSwitch = FALSE;;

//symbol을 입력받지 않은 경우 (CSEC을 입력받은 경우)

if(strcmp(symbol, "\0")==FALSE)

{

for(i = 0; i < HashFile; i = i + 1)

if(strcmp(ESTable[i]->EstabCsec, EstabCsec)==FALSE) return -1;

//for(j = 0; j < HashFile; j = j + 1)

strcpy(ESTable[HashFile]->EstabCsec, EstabCsec); //저장

ESTable[HashFile]->Establength = ProgSize[HashFile];

ESTable[HashFile]->address = CSAddress;

}

else //symbol을 입력받은 경우

{

int cntcnt = 0;

for(i = 0; i < 3; i = i + 1)

{

ESTABNODE \*HashEstabNode = ESTable[i];

while(HashEstabNode != NULL)

{

//SYMBOL이 이미 존재하는 경우 에러

if(strcmp(HashEstabNode->symbol, symbol)==FALSE) return -1;

HashEstabNode = HashEstabNode->nNode;

}

}

memset(HashEstabInputNode, '\0', sizeof(\*HashEstabInputNode)); //초기화

strcpy(HashEstabInputNode->symbol, symbol);

HashEstabInputNode->address = CSAddress + HashCurAddr;

HashEstabInputNode->nNode = NULL;

cntcnt = cntcnt + 1;

HashSwitch = TRUE;

HashEstabNode = ESTable[HashFile];

//심볼이 존재하지 않기 때문에 테이블에 심볼 저장시켜준다.

while(HashEstabNode->nNode != NULL) HashEstabNode = HashEstabNode->nNode;

if(HashSwitch==TRUE)

{

cntcnt = FALSE;

HashSwitch = FALSE;

}

HashEstabNode->nNode = HashEstabInputNode;

}

return FALSE;

}

//load pass1 알고리즘을 수행하는 함수

void Func\_LoaderPass1(FILE \*LoaderFP[], int Pass1File)

{

//input, control section, symbol 등을 저장하는 변수

char Pass1Input[100], EstabCsec[7], symbol[7], Pass1Addr[7];

int Pass1CurAddr, i, j, k, Pass1Switch=FALSE;

int Pass1Tempint = 0;

CSAddress = ProgaAddress;

//H, D 레코드 처리한다.

for(i = 0; i < Pass1File; i = i + 1)

{

fgets(Pass1Input, sizeof(Pass1Input), LoaderFP[i]); //파일 첫 줄 읽기

if(Pass1Input[0] == 'H') //record 가 H 면

{

Pass1Switch = TRUE;

ProgSize[i] = strtoul(Pass1Input+13, NULL, HEX);

//control section 이름 저

for(j = 0; j < 6; j = j + 1)

{

if(Pass1Input[j+1] == ' ') break;

EstabCsec[j] = Pass1Input[j+1];

}

if(Pass1Switch == TRUE)

{

Pass1Tempint = Pass1Tempint + 1;

}

EstabCsec[j] = '\0';

//CSEC이 중복인 경우

if(Func\_HashTab(EstabCsec, "\0", 0, i) == -1)

{

printf("Error : %s은 ESTABNODE에 이미 존재하는 CSEC 입니다.\n", EstabCsec);

LoaderSwitch = -1;

return;

}

}

fgets(Pass1Input, sizeof(Pass1Input), LoaderFP[i]); //파일 첫 줄 읽기

if(Pass1Input[0] == 'D') //record 가 D면

{

Pass1Switch = TRUE;

for(j = 0; j < strlen(Pass1Input); j = j + 12)

{

//공백이면 읽기 중단

if(Pass1Input[j+1] == ' ' || Pass1Input[j+1] == '\n') break;

for(k = 0; k < 6; k = k + 1)

{

if(Pass1Input[j + k + 1] == ' ' || Pass1Input[j + k + 1] == '\n') break;

symbol[k] = Pass1Input[j+k+1];

}

Pass1Switch = FALSE;

strncpy(Pass1Addr, Pass1Input + j + 7, 6);

if(Pass1Switch == TRUE)

{

Pass1Tempint = Pass1Tempint + 1;

}

symbol[k] = Pass1Addr[6] = '\0';

Pass1CurAddr = strtoul(Pass1Addr, NULL, HEX);

//SYMBOL이 중복인 경우 고려

if(Func\_HashTab("\0", symbol, Pass1CurAddr, i) == -1)

{

printf("Error : %s은 ESTABNODE에 이미 존재하는 SYMBOL 입니다.\n", symbol);

LoaderSwitch = -1;

return;

}

}

}

CSAddress = CSAddress + ProgSize[i];

}

}

//load pass2 알고리즘을 수행하는 함수

void Func\_LoaderPass2(FILE \*LoaderFP[], int Pass2File)

{

//extend symbol table 포인터 노드 변수

ESTABNODE \*Pass2ResultNode = NULL;

//pass2 알고리즘에서 사용되는 input, count, symbol, temp 등에 대한 변수

char Pass2Input[100], Pass2Dic[100][7], Pass2Num[3], Pass2Symbol[7];

char Pass2Temp[7], Pass2Temp2[3], ModiStr[10];

//modification 레코드에 대한 변수

int ModiObj, ModiAddr, i, j, k;

int TAddr, TSize, TOneByte, TMemory;

int LPTemp = 0, LPTemp2 = 0, SizeLP = 0, ll = 0;

int LPSwitch = FALSE;

CSAddress = ProgaAddress;

for(i = 0; i < Pass2File; i = i + 1)

{

memset(Pass2Dic, '\0', sizeof(Pass2Dic));

while(fgets(Pass2Input, sizeof(Pass2Input), LoaderFP[i]) != NULL) //한줄 읽어오기

{

//Reference record 읽기

if(Pass2Input[0] == 'R')

{

LPTemp = 0;

strcpy(Pass2Dic[1], ESTable[i]->EstabCsec); //reference에 control section 이름을 저장한다.

SizeLP = strlen(Pass2Input);

for(j = 0; j < strlen(Pass2Input); j = j + 8)

{

if(Pass2Input[j+1] == '\n' || Pass2Input[j+1] == ' ') break;

//공백이나 줄바꿈의 경우 중단한다.

SizeLP = SizeLP + j;

strncpy(Pass2Num, Pass2Input+j+1, 2);

for(k = 0; k < 6; k = k + 1)

{

LPTemp = LPTemp + SizeLP;

}

for(k = 0; k < 6; k = k + 1)

{

if(Pass2Input[j + k + 3] == '\n' || Pass2Input[j + k + 3] == ' ') break;

ll = ll + 1;

Pass2Symbol[k] = Pass2Input[j+k+3];

}

LPSwitch = TRUE;

Pass2Num[2] = Pass2Symbol[k] = '\0';

strcpy(Pass2Dic[atoi(Pass2Num)], Pass2Symbol); //reference dict 생성

}

}

if(Pass2Input[0] == 'T') //T record

{

int CountPass2input = 0;

strncpy(Pass2Temp, Pass2Input+1, 6);

strncpy(Pass2Temp2, Pass2Input+7, 2);

CountPass2input = CSAddress;

Pass2Temp[6] = Pass2Temp2[2] = '\0';

TAddr = strtoul(Pass2Temp, NULL, HEX);

TSize = strtoul(Pass2Temp2, NULL, HEX);

TMemory = CSAddress + TAddr;

//메모리에 1 byte 씩 저장시켜준다.

for(j = 0; j < TSize; j = j + 1)

{

strncpy(Pass2Temp2, Pass2Input + 9 + j \* 2, 2);

Pass2Temp2[2] = '\0';

TOneByte = strtoul(Pass2Temp2, NULL, HEX);

size[TMemory] = TOneByte;

TMemory = TMemory + 1;

}

}

if(Pass2Input[0] == 'M') // M record

{

strncpy(Pass2Temp, Pass2Input+1, 6);

strncpy(Pass2Temp2, Pass2Input+7, 2);

Pass2Temp[6] = Pass2Temp2[2] = '\0';

LPTemp2 = LPTemp;

LPTemp = 0;

//주소 저장

ModiAddr = CSAddress + strtoul(Pass2Temp, NULL, HEX);

ModiObj = strtoul(Pass2Temp2, NULL, HEX);

memset(ModiStr, '\0', sizeof(ModiStr));

for(j = 0; j < 3; j = j + 1)

{

sprintf(Pass2Temp2, "%02X", size[ModiAddr+j]);

strcat(ModiStr, Pass2Temp2);

}

ModiObj = strtoul(ModiStr, NULL, HEX);

if(ModiStr[0] == '8' || ModiStr[0] == '9' || ('A' <= ModiStr[0] && ModiStr[0] <= 'F')) ModiObj = -(0x1000000 - ModiObj);

//obj code가 원래 음수였으면 실제로 음수값이 저장되도록 변경시켜 준다.

strncpy(Pass2Num, Pass2Input+10, 2);

Pass2Num[2] = '\0';

//control section, symbol을 탐색한다.

for(j = 0; j < 3; j = j + 1)

{

ESTABNODE \*Pass2SearchNode = ESTable[j];

while(Pass2SearchNode != NULL)

{

if(!strcmp(Pass2Dic[atoi(Pass2Num)], Pass2SearchNode->symbol))

{

LPSwitch = TRUE;

Pass2ResultNode = Pass2SearchNode;

break;

}

Pass2SearchNode = Pass2SearchNode->nNode;

}

if(strcmp(Pass2Dic[atoi(Pass2Num)], ESTable[j]->EstabCsec)==FALSE)

{

Pass2ResultNode = ESTable[j];

LPSwitch = TRUE;

break;

}

}

//reference record가 없는 경우 에러 출력

if(Pass2ResultNode == NULL)

{

printf("Error : %s의 %s은 ESTABNODE에 존재하지 않습니다.\n", ESTable[Pass2File-1]->EstabCsec, Pass2Dic[atoi(Pass2Num)]);

LoaderSwitch = -1;

return;

}

//4형식

if(Pass2Input[9] == '+') ModiObj = ModiObj + Pass2ResultNode->address;

else ModiObj = ModiObj - Pass2ResultNode->address;

if(ModiObj < 0) ModiObj = ModiObj + 0x1000000;

sprintf(ModiStr, "%06X", ModiObj);

for(j = 0; j < 3; j = j + 1)

{

strncpy(Pass2Temp2, ModiStr+j\*2, 2);

Pass2Temp2[2] = '\0';

TOneByte = strtoul(Pass2Temp2, NULL, HEX);

size[ModiAddr++] = TOneByte;

}

}

//E record 종료

if(Pass2Input[0] == 'E') CSAddress = CSAddress + ProgSize[i];

}

}

}

//입력된 레지스터의 실제 주소를 리턴시켜주는 함수이다.

int \*RegisterAddress(char ResAdd)

{

if(ResAdd == '0') return &NUM\_A;

else if(ResAdd == '1') return &NUM\_X;

else if(ResAdd == '2') return &NUM\_L;

else if(ResAdd == '3') return &NUM\_B;

else if(ResAdd == '4') return &NUM\_S;

else if(ResAdd == '5') return &NUM\_T;

else if(ResAdd == '8') return &PC;

else return NULL;

}

//object 파일을 읽어들인 후 linking 을 한다.

//그 결과를 가상 메모리에 기록하는 역할을 하는 함수이다.

void Func\_Loader()

{

FILE \*LoaderFP[3]; //입력받는 오브젝트 파일 저장 변수

ESTABNODE \*LoadSearchNode; //심볼테이블 탐색 변수

int i, LoaderFile=0, LoaderLength = 0, SizeofLoad = 0;

int LOSwitch = FALSE;

char LoaderTemp[100], LoaderProgName[3][100];

LoaderSwitch = FALSE;

strcpy(LoaderTemp, ins);

char \*LoaderTemp2;

LoaderTemp2 = strtok(LoaderTemp, " ");

LOSwitch = TRUE;

//LoaderTemp2가 존재하면 ' '으로 구분하여 파일 이름 저장

while(LoaderTemp2 != NULL)

{

LoaderTemp2 = strtok(NULL, " "); //공백으로 구분

if(LoaderTemp2 == NULL) break;

SizeofLoad = SizeofLoad + 1;

strcpy(LoaderProgName[LoaderFile], LoaderTemp2);

LoaderFP[LoaderFile] = fopen(LoaderProgName[LoaderFile], "r");

//파일이 존재하지 않을 경우 에러 출력

if(LoaderFP[LoaderFile++] == NULL)

{

printf("Error : %s is Not Exist File.\n", LoaderProgName[--LoaderFile]);

return;

}

}

if(LOSwitch == TRUE)

{

SizeofLoad = 0;

LOSwitch = FALSE;

}

if(!LoaderFile)

{

LOSwitch = TRUE;

printf("Error : 인자가 한개 이상 존재해야 합니다.\n");

return;

}

//각각의 extend symbol table을 초기화

for(i = 0; i < 3; i = i + 1)

{

ESTable[i]->nNode = NULL;

memset(ESTable[i], '\0', sizeof(ESTable));

}

//레지스터 초기화

NUM\_A = 0;

NUM\_X = 0;

NUM\_B = 0;

NUM\_S = 0;

NUM\_T = 0;

Operator = '\0';

//load pass1 알고리즘 수행행

Func\_LoaderPass1(LoaderFP, LoaderFile);

if(LoaderSwitch == -1) return;

for(i = 0; i < LoaderFile; i = i + 1) rewind(LoaderFP[i]);

//load pass2 알고리즘 수행

Func\_LoaderPass2(LoaderFP, LoaderFile);

if(LoaderSwitch == -1) return;

printf("control symbol address length\n");

printf("section name\n");

printf("-------------------------------\n");

int loadcnt = 0;

for(i = 0; i < LoaderFile; i = i + 1)

{

LoadSearchNode = ESTable[i];

while(LoadSearchNode != NULL)

{

if(strcmp(LoadSearchNode->EstabCsec, "\0"))

{

//control section 출력시켜준다.

printf("%-6s %13.04X %6.04X\n", LoadSearchNode->EstabCsec, LoadSearchNode->address, LoadSearchNode->Establength);

loadcnt = FALSE;

LoaderLength = LoaderLength + LoadSearchNode->Establength;

loadcnt = loadcnt + LoaderLength;

}

else printf("%13s %6.04X\n", LoadSearchNode->symbol, LoadSearchNode->address);

LoadSearchNode = LoadSearchNode->nNode;

//symbol 출력시켜준다.

}

}

printf("-------------------------------\n\t total length %04X\n", LoaderLength);

for(i = 0; i < LoaderFile; i = i + 1) fclose(LoaderFP[i]);

Func\_History\_Save(); //명령어 history에 저장

}

//run 입력 받았을 때 opcode에 따른 기능을 수행하는 함수이다.

void Func\_Inst(int OP, unsigned char \*InstMemory, int InstImmediate, int TA, int InstExtend)

{

int i, InstVar, InstTemp = 0;

int InstSwitch = FALSE;

if(InstMemory != NULL)

{

InstVar = \*InstMemory;

for(i = 1; i <= 2; i = i + 1) InstVar = (InstVar << 8) + \*(InstMemory + i);

}

if(OP == 0x00) //LDA

{

InstSwitch = TRUE;

if(InstMemory == NULL) NUM\_A = InstImmediate;

else NUM\_A = InstVar;

}

else if (OP == 0x04) //LDX

{

InstSwitch = TRUE;

if(InstMemory == NULL) NUM\_X = InstImmediate;

else NUM\_X = InstVar;

}

else if (OP == 0x50) //LDCH

{

InstSwitch = TRUE;

if(InstMemory == NULL) NUM\_A = InstImmediate;

else NUM\_A = \*InstMemory;

}

else if (OP == 0x68) //LDB

{

InstSwitch = TRUE;

if(InstMemory == NULL) NUM\_B = InstImmediate;

else NUM\_B = InstVar;

}

else if (OP == 0x74) //LDT

{

InstSwitch = TRUE;

if(InstMemory == NULL) NUM\_T = InstImmediate;

else NUM\_T = InstVar;

}

else if (OP == 0x14) //STL

{

InstSwitch = TRUE;

for(i = 0; i < 3; i = i + 1) \*(InstMemory + i) = NUM\_L >> 8 \* (2 - i);

}

else if (OP == 0x10) //STX

{

InstSwitch = TRUE;

for(i = 0; i < 3; i = i + 1) \*(InstMemory + i) = NUM\_X >> 8 \* (2 - i);

}

else if (OP == 0x0C) //STA

{

InstSwitch = TRUE;

for(i = 0; i < 3; i = i + 1) \*(InstMemory + i) = NUM\_A >> 8 \* (2 - i);

}

else if (OP == 0x30) //JEQ

{

InstSwitch = TRUE;

if(Operator == '=') runAddress = PC = TA;

}

else if (OP == 0x48) //JSUB

{

InstSwitch = TRUE;

NUM\_L = PC;

runAddress = PC = TA;

}

else if (OP == 0x38) //JLT

{

InstSwitch = TRUE;

if(Operator == '<') runAddress = PC = TA;

}

else if (OP == 0x3C) runAddress = PC = TA; //J

else if (OP == 0x54) \*InstMemory = NUM\_A & 0x0000FF; //STCH

else if (OP == 0x4C) runAddress = PC = NUM\_L; //RSUB

else if (OP == 0x28) //COMP

{

InstSwitch = TRUE;

if(InstMemory == NULL) InstVar = InstImmediate;

if(NUM\_A == InstVar) Operator = '=';

else if(NUM\_A < InstVar) Operator = '<';

else Operator = '>';

}

else if (OP == 0xD8) NUM\_A = 0; //RD

else if (OP == 0xDC) \*InstMemory = NUM\_A; //WD

else if (OP == 0xE0) Operator = '<'; //TD

if(InstSwitch == TRUE)

{

InstTemp = InstVar;

InstSwitch = FALSE;

}

InstSwitch = FALSE;

}

//loader 명령어로 메모리에 할당 된 프로그램을 실행하는 함수이다.

void Func\_Run()

{

//byte 저장하는 TEmp 문자 변수

char RunByte1[3], RunByte2[3], RunByte3[3], RunByte4[3], RunHalfByte[2], StringTA[7];

int RunEndAddr = 0, RunLength = 0, RLength = 0; //시작,끝 주소 저장 변수

int byte1, byte2, byte3, Halfbyte;

int is\_start = 1, i, j, k;

int Num\_ni, Num\_xbpe[4], TA = 0;

int RunSwitch = FALSE, RUNENDSwitch = FALSE, RunTemp = 0;

while(ESTable[i] != NULL) RunLength = RunLength + ESTable[i++]->Establength; //읽어들이기

RunEndAddr = ProgaAddress + RunLength;

RLength = RunEndAddr;

RLength = RLength + 1;

//프로그램이 마지막 실행이 아니면 초기화

if(!EndSwitch)

{

RunEndAddr = RunLength;

runAddress = ProgaAddress; //시작주소로 초기화

PC = ProgaAddress;

NUM\_L = RunLength; //프로그램 길이로 초기화

}

//메모리를 탐색한다.

while(runAddress <= RunEndAddr)

{

if(PC >= RunEndAddr) break; //PC가 EndAddr보다 크면 탐색 중단

RunSwitch = TRUE;

runAddress = PC;

sprintf(RunByte1, "%02X", size[runAddress]);

byte1 = size[runAddress];

//1형식인 경우

if(RunByte1[0] == 'C' || RunByte1[0] == 'F')

{

RunSwitch = TRUE;

PC = PC + 1;

runAddress = PC;

}

else if(RunByte1[0] == '9' || RunByte1[0] == 'A' || RunByte1[0] == 'B') //2형식

{

RunSwitch = TRUE;

PC = PC + 2;

sprintf(RunByte2, "%02X", size[++runAddress]);

int \*Register1 = RegisterAddress(RunByte2[0]);

int \*Register2 = RegisterAddress(RunByte2[1]);

RunTemp = byte1;

if(byte1 == 0x90) //ADDR

{

RunSwitch = TRUE;

}

else if(byte1 == 0xB4)//CLEAR : R1 0으로 초기화

{

RunSwitch = TRUE;

\*Register1 = 0;

}

else if(byte1 == 0xA0) //COMPR : R1, R2 비교한 결과를 SW에 저장

{

RunSwitch = TRUE;

if(\*Register1 == \*Register2) Operator = '=';

else if(\*Register1 < \*Register2) Operator = '<';

else Operator = '>';

}

else if(byte1 == 0x9C) RunSwitch = TRUE; //DIVR

else if(byte1 == 0x98) RunSwitch = TRUE; //MULR

else if(byte1 == 0xAC) RunSwitch = TRUE; //RMO

else if(byte1 == 0xA4) RunSwitch = TRUE; //SHIFTL

else if(byte1 == 0x94) RunSwitch = TRUE; //SUBR

else if(byte1 == 0xB0) RunSwitch = TRUE; //SVC

else if(byte1 == 0xB8) //TIXR : X 증가시키고 R1과 비교한 결과를 SW에 저장

{

RunSwitch = TRUE;

NUM\_X = NUM\_X + 1;

if(NUM\_X == \*Register1) Operator = '=';

else if(NUM\_X < \*Register1) Operator = '<';

else Operator = '>';

}

else //모두 아닌 경우 에러 출력

{

RunSwitch = FALSE;

printf("Error : object code Error.\n");

return;

}

runAddress = PC;

if(RunSwitch == TRUE)

{

RUNENDSwitch = TRUE;

RunSwitch = FALSE;

}

}

else //3형식, 4형식 인 경우

{

int RunTA = 0;

//메모리 초기화

memset(StringTA, '\0', sizeof(StringTA));

sprintf(RunByte2, "%02X", size[++runAddress]);

byte2 = size[runAddress];

sprintf(RunByte3, "%02X", size[++runAddress]);

byte3 = size[runAddress];

//target address 불러오기

RunTA = byte2 + byte3;

RunTemp = RunTA;

memset(StringTA, '\0', sizeof(StringTA));

StringTA[0] = RunByte2[1];

strcat(StringTA, RunByte3);

RunTA = byte2 + byte3;

//xbpe 읽어오기

RunHalfByte[0] = RunByte2[0];

RunHalfByte[1] = '\0';

Halfbyte = strtoul(RunHalfByte, NULL, HEX);

for(i = 0; i < 4; i = i + 1)

{

Num\_xbpe[i] = Halfbyte % 2;

Halfbyte = Halfbyte / 2;

}

PC = PC + 3;

if(Num\_xbpe[0] == 1)

{

PC = PC + 1;

sprintf(RunByte4, "%02X", size[++runAddress]);

strcat(StringTA, RunByte4);

}

TA = strtoul(StringTA, NULL, HEX);

//target address가 음수인 경우 3형식

if(StringTA[0] == '8' || StringTA[0] == '9' || ('A' <= StringTA[0] && StringTA[0] <= 'F')) TA = TA - 0x1000;

if(Num\_xbpe[3] == 1) TA = TA + NUM\_X;

if(Num\_xbpe[1] == 1) TA = TA + PC;

else if(Num\_xbpe[2] == 1) TA = TA + NUM\_B;

//ni를 불러온다.

Num\_ni = byte1 % 0x4;

int OP = byte1 - Num\_ni;

if(Num\_ni == 0) //ni=0, 즉 sic mode인 경우

{

RunSwitch = TRUE;

memset(StringTA, '\0', sizeof(StringTA));

if(Num\_xbpe[3]) byte2 = byte2 - 128;

TA = byte2;

TA = TA << 8;

TA = TA + byte3;

Func\_Inst(OP, &size[TA], -1, TA, Num\_xbpe[0]);

}

else if(Num\_ni == 1) //immediate mode 인 경우

{

RunSwitch = TRUE;

Func\_Inst(OP, NULL, TA, TA, Num\_xbpe[0]);

}

else if(Num\_ni == 2) //indirect mode 인 경우

{

RunSwitch = TRUE;

int indirect\_TA;

for(i = 0; i < 3; i = i + 1)

{

indirect\_TA += size[TA+i];

if(i != 2) indirect\_TA = indirect\_TA << 8;

}

Func\_Inst(OP, &size[size[TA]], -1, indirect\_TA, Num\_xbpe[0]);

}

else if(Num\_ni == 3) //simple mode 인 경우

{

RunSwitch = TRUE;

Func\_Inst(OP, &size[TA], -1, TA, Num\_xbpe[0]);

}

runAddress = PC;

}

if(RunSwitch == TRUE)

{

RUNENDSwitch = TRUE;

RunSwitch = FALSE;

}

for(i = 0; i < bpNum; i = i + 1)

if(PC == bp[i])

{

EndSwitch = TRUE;

break;

}

if(EndSwitch == TRUE)

{

EndSwitch = -1;

break;

}

}

if(PC >= RunEndAddr)PC = RunEndAddr;

printf("A : %06X X : %06X\n", NUM\_A, NUM\_X);

printf("L : %06X PC : %06X\n", NUM\_L, PC);

printf("B : %06X S : %06X\n", NUM\_B, NUM\_S);

printf("T : %06X\n", NUM\_T);

if(PC == RunEndAddr)

{

printf("\t End Program\n");

EndSwitch = FALSE;

}

else printf("\t Stop at checkpoint[%X]\n", PC);

Func\_History\_Save();

}

//break point를 생성하고, 명령어에 따라 출력 및 clear 시키는 한수이다.

void Func\_Bp()

{

int i = 0, BPMaxnum = 0, lengthBP = 0, BPVar = 0;;

char \*BPAddr, BPTemp[100];

while(ESTable[i] != NULL) BPMaxnum = BPMaxnum + ESTable[i++]->Establength;

//table을 읽어 bp의 개수를 구한다.

//bp 입력 시 현재 있는 모든 bp 를 출력

if(strcmp(ins, "bp")==FALSE)

{

printf("\t\tbreakpoint\n\t\t----------\n");

for(i = 0; i < bpNum; i++) printf("\t\t%X\n", (int)bp[i]);

}

else if(strcmp(ins, "bp clear")==FALSE) //bp를 모두 초기화

{

printf("\t\t[ok] clear all breakpoints\n");

memset(bp, '\0', sizeof(bp));

bpNum = 0;

}

else

{

strcpy(BPTemp, ins);

strtok(BPTemp, " ");

BPAddr = strtok(NULL, " ");

//공백으로 address 분리

if(strtok(NULL, " ") != NULL) //address 없으면 에러 출력

{

printf("Error : bp Input Error.\n");

return;

}

lengthBP = strlen(BPAddr);

for(i = 0; i < lengthBP; i = i + 1)

if(!('0' <= BPAddr[i] && BPAddr[i]<= '9' )) //인자가 16진수가 아닌 경우

{

if(!('a' <= BPAddr[i] && BPAddr[i] <= 'f') && !('A' <= BPAddr[i] && BPAddr[i] <= 'F') )

{

printf("Error : bp address is HEX.\n");

return;

}

}

BPVar = strtoul(BPAddr, NULL, HEX);

//bp 범위를 넘으면 에러 출력

if(BPMaxnum < BPVar)

{

printf("Error : Not Range BP Address.\n");

return;

}

printf("\t\t[ok] create breakpoints %s\n", BPAddr);

bp[bpNum++] = BPVar;

}

Func\_History\_Save();

}

* 1. **Makefile**

20171609.out : 20171609.o

gcc -Wall -o 20171609.out 20171609.o

20171609.o : 20171609.h 20171609.c

clean :

rm 20171609.o

rm 20171609.out