**과목명: 시스템프로그래밍**

**분반 2**

**<<Project #2>>**

**서강대학교 [컴퓨터공학]**

**[20161598]**

**[손동현]**

목 차

1. **프로그램 개요** 3
2. **프로그램 설명**
   1. 프로그램 흐름도 4,5
3. **모듈 정의**
   1. 모듈 이름 : void command\_type(); 6
      1. 기능 6
      2. 사용 변수 6
   2. 모듈 이름 : void command\_assemble(); 6
      1. 기능 6
      2. 사용 변수 6
   3. 모듈 이름 : void get\_filename(char\* arr, int i);6
      1. 기능 6
      2. 사용 변수 6
   4. 모듈 이름 : int divide\_one\_line(char\* arr, int pass\_num); 6
      1. 기능 6
      2. 사용 변수 6
   5. 모듈 이름 : int find\_opcode(char\* arr, int i); 7
      1. 기능 7
      2. 사용 변수 7
   6. 모듈 이름 : void command\_symbol(); 7
      1. 기능 7
      2. 사용 변수 7
   7. 모듈 이름 : void free\_symtab(); 8
      1. 기능 8
      2. 사용 변수 8
   8. 모듈 이름 : void distinct\_format(char\* opcode, char\* operand); 8
      1. 기능 8
      2. 사용 변수 8
   9. 모듈 이름 : void num\_to\_hexchar(char\* value, int num); 8
      1. 기능 8
      2. 사용 변수 8
   10. 모듈 이름 : int find\_symbol(char\* operand);8
       1. 기능 8
       2. 사용 변수 8
   11. 모듈 이름 : void change\_hex(char\* bin, char\* hex, int format\_num); 8
       1. 기능 8
       2. 사용 변수 8
   12. 모듈 이름 : void disp\_addr\_set(char\* arr, int num); 8
       1. 기능 8
       2. 사용 변수 8
   13. 모듈 이름 : char hex\_char\_return(int num); 8
       1. 기능 8
       2. 사용 변수 8
   14. 모듈 이름 : void xbpe\_setting(char\* value, int x, int b, int p, int e); 9
       1. 기능 9
       2. 사용 변수 9
   15. 모듈 이름 : void register\_num(char\* arr, char\* reg); 9
       1. 기능 9
       2. 사용 변수 9
4. **전역 변수, 구조체 정의**
   1. typedef struct \_\_SYM 10
   2. typedef struct \_Modi 10
   3. int loc; 10
   4. int start\_address; 10
   5. int program\_length; 10
   6. int pc; 10
   7. int symbol\_flag; 10
   8. char file\_name[31]; 10
   9. unsigned char line\_opcode[31]; 10
   10. unsigned char line\_symbol[31]; 10
   11. unsigned char line\_operand[31]; 10
   12. int opcode\_val; 11
   13. int text\_flag; 11
   14. char text\_record[61]; 11
   15. int text\_index; 11
   16. char result\_obj\_code[11]; 11
   17. char BASE\_str[31]; 11
   18. int obj\_loc; 11
   19. int asm\_start\_flag; 11
   20. int lst\_line\_num = 5; 11
   21. int pass\_1\_flag; 11
   22. SYM\* first; 11
   23. SYM\* before; 11
   24. Modi\* modi\_head; 11
   25. Modi\* modi\_prev; 11
   26. int SYMTAB\_flag; 11
5. **코드 설명**
   1. 20161598.h 11
   2. 20161598.c 50

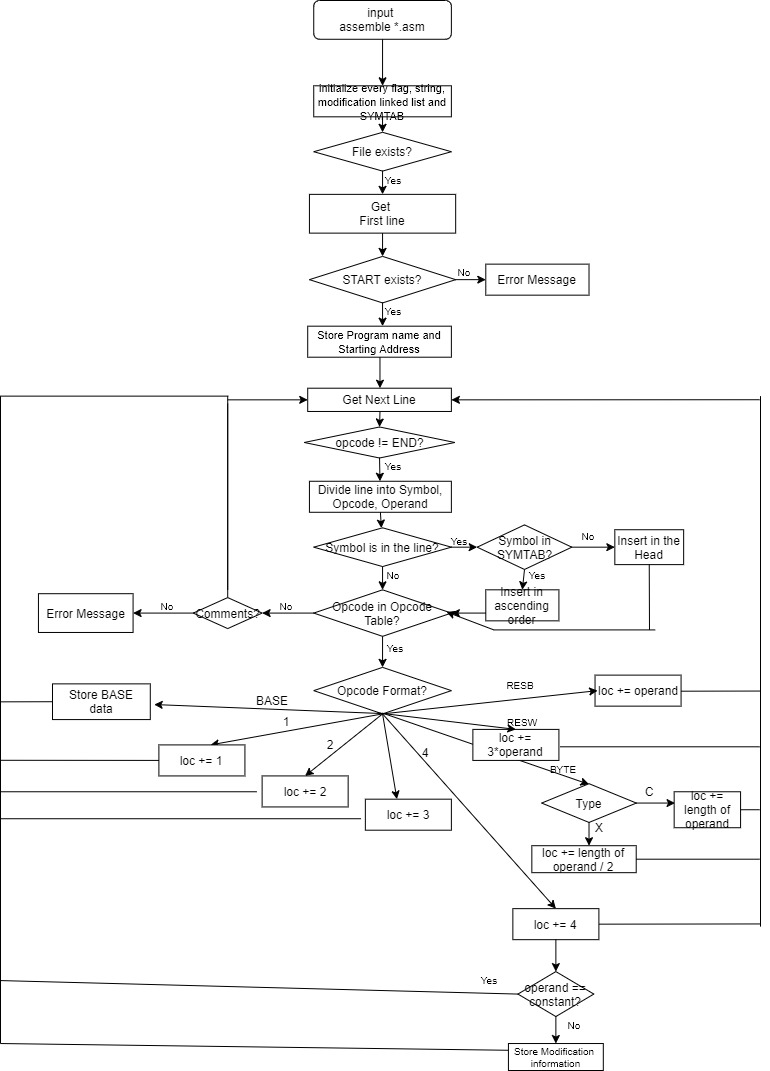
**1 . 프로그램 개요**

이 프로그램은 프로젝트 1에서 구현한 SIC를 토대로 작성하였다. 우선 \*.asm 파일을 assemble하는 assembler를 구현하였다. 해당 \*.asm 파일을 읽어서 pass1, pass2에 걸쳐 파일에 대한 정보들을 저장하고 error가 없는 코드라면 이에 해당하는 \*.lst, \*.obj 파일을 만들어내도록 하였다. 만일 error가 있는 코드라면 \*.lst, \*.obj 파일을 만들지 않고 error를 출력하도록 하였다. 또한 만들어진 파일 및 디렉터리에 존재하는 파일은 type 명령어를 통해 내용을 볼 수 있고, symbol 명령어를 통해 가장 최근에 assemble된 SYMTAB를 오름차순으로 볼 수 있다. 이러한 명령어에 대한 내용은 help를 통해 알 수 있게 구현하였다.

**2 . 프로그램 설명**

2.1 프로그램 흐름도

PASS 1



Valid Opcode?

YES

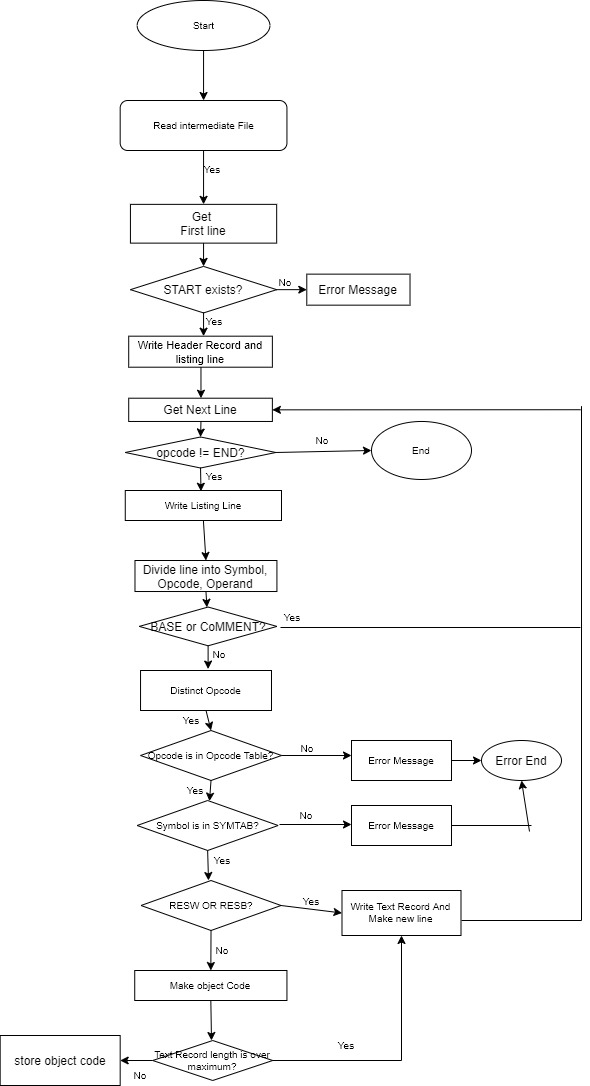
Error Message

insert

No

No

PASS 2



Put object Code in the last of listing line if it exists

위의 흐름도처럼 assemble이라는 명령어가 들어오면 해당 파일 이름이 디렉터리에 존재하는지 판별하고, 존재하면 파일을 열어 pass1을 시작한다. 먼저 첫번째 라인을 읽어서 START가 존재하는지 확인한다. 존재하면 해당 줄에 있는 프로그램명과 시작주소를 저장하고 다음 줄을 읽고, 아니면 error message를 출력 후 종료한다. 이제 각 줄은 END가 들어오기 전까지 계속 읽는데, 한 줄을 읽은 뒤 해당 줄을 빈칸 또는 \t를 기준으로 나눠서 symbol, opcode, operand로 하여 정보를 받아들이고, 만일 symbol이 있는 경우, 해당 symbol을 SYMTAB에 insert한다. 만일 symbol이 이미 SYMTAB에 존재하는 경우, error 메시지를 출력한다. 그 다음 opcode가 타당한 opcode인지 살펴본다. 만일 타당한 opcode이면 해당 opcode를 각 형식에 맞게 판별하여 해당 opcode에 대한 loc값을 계산해주고, 타당한 opcode가 아닐 경우, comments인지 확인 후 맞으면 다음 줄으로, 틀리면 에러 메시지를 출력한다. 여기서 format 4일경우, object code에 쓰일 modification을 위해 해당 줄의 modification information을 저장한다. 모든 줄을 읽을 때마다 각 줄에 대한 line number와 loc 값을 제일 앞에 붙여서 중간 파일인 intermediate 파일에 써준다.

그 다음은 pass2이다. Pass2에서는 pass1에서 만든 중간 파일인 intermediate을 읽어서 pass1과 비슷한 방법으로 진행한다. 먼저 첫번째 줄을 읽어서 START가 존재하면, 해당 줄을 읽어서 프로그램 이름, 시작 주소, 총 길이를 갖는 Header Record를 작성하고 listing line도 작성한다. 그 다음은 END를 받을 때까지 계속 한 줄씩 읽는데, 한 줄을 읽은 뒤, BASE 또는 Comment이면 listing line을 작성하고 다음 줄로 넘어간다. 그게 아니라면 opcode를 opcode table에서 확인 후 타당하면 symbol을 찾아본다. Opcode table에 없을 경우에는 error message를 출력한다. Operand로 받은 symbol이 SYMTAB에 존재하면 다음 과정으로 넘어가지만, 없을 경우 에러 메시지를 출력한다. 만일 RESW 또는 RESB이면 변수이므로 object code를 만들지 않고, 해당 줄에서 Text Record가 MAX 길이가 아니더라도 바로 이전 line object code까지 써준다. 변수가 아니고, 제대로 된 line이 들어온 경우, 각 format 또는 byte, word 상수에 맞게 object code를 생성한다. 만일 타당해보이나, 주소 값 차이등을 범위가 넘어간다면 error message를 출력해주고 종료한다. 생성한 object code를 써주면 Text Record의 Max 길이를 넘는지 확인한다. 만일 넘으면 해당 줄 이전까지의 object code를 쓰고, 새로운 Text Record를 만들고, 아니면 object code들을 계속 저장한다. 그 후, object 코드가 생성되었다면 listing line의 제일 마지막에 각 줄의 object code를 적은 후 다음 줄을 받을 수 있도록 한다.

**3 . 모듈 정의**

* 1. 모듈 이름 : void command\_type()

3.1.1 기능

type \*filename을 입력하면 해당 filename의 내용을 읽어서 출력하여 어떤 내용을 갖고 있는지 보여준다.

3.1.2 사용 변수

char temp; //한 문자씩 입력받을 때 필요한 변수이다.

* 1. 모듈 이름 : void get\_filename(char\* arr, int i)
     1. 기능

command에서 명령어 filename 이런식으로 들어오기 때문에 filename만 따로 빼서 file\_name array에 저장하여 준다.

* + 1. 사용 변수

char\* arr; //입력 받은 command array

int i ; //command array의 index

int j = 0; //file name의 index

* 1. 모듈 이름 : int divide\_one\_line(char\* arr, int pass\_num)
     1. 기능

한 줄을 읽어들였을 때, 해당 줄을 빈칸 또는 \t을 기준으로 symbol, opcode, operand로 나눈다. 그리고 start가 들어오면 0, end가 들어오면 1, comments가 들어오면 -1, error 발생시 -4, 나머지는 2를 리턴하여 해당 줄을 판별할 수 있게 해준다.

* + 1. 사용 변수

char\* arr; //읽은 한 줄에 대한 char array

int pass\_num //pass1인지 pass2인지에 대한 정보

int i = 0, j = 0, k = 0, l = 0; // line\_symbol, line\_operand, line\_opcode, line\_loc에 대한 index

char line\_loc[4]; // pass2인 경우 처음에 들어오면 loc 값 저장하는 배열

int cnt\_blank = 0; //빈칸 또는 \t를 기준으로 나누기 위해 쓰이는 flag

int comma\_flag = 0; //comma\_flag : 1 빈칸이 있어도 뒤에 어떤 operand를 더 받아야한다는 의미

* 1. 모듈 이름 : void command\_assemble()
     1. 기능

Assemble에 대한 기능을 수행한다. Pass1에서 해당 파일을 읽어서 파일에 대한 intermediate 파일을 만들고, pass2에서 intermediate 파일을 읽어서 .obj, .lst 파일을 만든다.

* + 1. 사용 변수

char obj\_name[31] = { 0 }; //.obj 파일이름 저장하는 string

char lst\_name[31] = { 0 }; //.lst 파일이름 저장하는 string

int file\_index = 0, x=0; //파일 이름의 index, obj\_name, lst\_name의 index

char one\_line[100] = { 0 }; //.asm file의 한 line을 저장하는 string

int insert\_flag = 0; //SYMTAB에 symbol 저장했는지 알기 위한 flag

SYM\* node = (SYM\*)malloc(sizeof(SYM)); //SYMTAB 만들기 위한 dynamic memory

int constant\_length = 0; //constant byte의 길이

int i = 2; // line\_operand의 index

Modi\* modi\_node = (Modi\*)malloc(sizeof(Modi)); //Modification 정보 저장하기 위한 dynamic memory

int opcode\_found\_flag = find\_opcode(line\_opcode, 0); //opcode가 opcode table에 존재하는지 알기 위한 flag

int variable\_flag = 0; //만일 변수면, 해당 object line 적기 위해 쓰이는 flag

int printed\_flag = 0; //만일 아직 안쓰인 object line이 있는데 end가 들어온 경우, object line print하기 위해 쓰이는 flag

int obj\_index = 0, cnt=0; // result\_obj\_code의 index, 생성한 object code의 길이

* 1. 모듈 이름 : int find\_opcode(char \*arr, int i)
     1. 기능

프로젝트 1에서 만든 hashtable에서 opcode를 찾는다. 만일 찾으면 해당 opcode가 가능한 format return 아니면 0 return.

* + 1. 사용 변수

char temp\_mnemonic[7] = { 0 }; //들어온 mnemonic을 저장하는 string

int mnemonic\_num = 1; //hash table의 index를 판별하기 위한 변수

int mnemonic\_format = 0; // mnemonic의 format을 저장하는 변수

int j = 0, mnemonic\_length = 0; //temp\_mnemonic의 index, mnemonic 길이

* 1. 모듈 이름 : void command\_symbol()
     1. 기능

가장 최근에 assemble한 파일의 SYMTAB을 전부 출력

* + 1. 사용 변수

SYM\* cur = first; //현재 SYM 노드를 가리키는 포인터

* 1. 모듈 이름 : void free\_symtab()
     1. 기능

모든 SYMTAB와 modification linked list를 free한다.

* + 1. 사용 변수

없음

* 1. 모듈 이름 : void distinct\_format(char\* opcode, char\*operand)
     1. 기능

Opcode와 operand를 입력 받아, 해당하는 format을 판별하여 object code를 생성한다.

* + 1. 사용 변수

char\* opcode, char\*operand //opcode와 operand 저장하는 배열

char result\_obj\_before\_bind[130] = { 0 }; //4개씩 묶기 전의 10101으로 된 object 코드

int opcode\_format = 0; //opcode의 format number 저장

int x\_flag = 0; //x register 사용 유무

int b\_flag = 0; //BASE relative인지 아닌지

int p\_flag = 0; //PC relative인지 아닌지

int e\_flag = 0; //Extension인지 아닌지

int addr = 0; //format4에서의 주소

char pure\_operand[31] = { 0 }; //만일 #이나 @가 붙은 경우, 이걸 제외한 순수한 operand 값 저장 배열

int operand\_index = 1, pure\_index = 0; //operand와 pure\_operand의 index

int find = find\_symbol(pure\_operand); //symbol을 SYMTAB에서 찾았는지 판별 위한 flag

int i = 2; //operand의 index

int j = 0; //result\_object\_code의 index

int num = atoi(operand); //operand의 실제 값 저장 위한 변수

int hexx //16진수로 변환할 때 필요한 변수

int quotient = num / hexx; //16진수로 변환할 때 필요한 변수

int comma\_flag = 0; //comma가 들어왔는지 판별 위한 flag

char register\_arr[3] = { 0 }; // register type 저장하는 배열

int register\_index = 0; //register\_arr의 index

int disp = 0; //format 3의 disp

int symbol\_length = find\_symbol(pure\_operand); //symbol의 길이를 알기 위한 변수, symbol 찾았는지 알기 위한 flag로도 사용

* 1. 모듈 이름 : void num\_to\_hexchar(char \*value, int num)
     1. 기능

받은 int 정수를 2진수로 바꾸어 value array에 저장.

* + 1. 사용 변수

char \*value; //2진수 값을 저장하는 character array

Int num; // 2진수로 바꾸고 싶은 정수.

int x = 0, i=0; //x는 16으로 나눈 몫이나 나머지 저장, i는 value arr의 index

* 1. 모듈 이름 : void xbpe\_setting(char \*value, int x, int b, int p, int e)
     1. 기능

x, b, p, e에 해당하는 값을 받아 value에 저장.

* + 1. 사용 변수

char \*value; //값이 저장될 array

int x, int b, int p, int e; // x, b, p, e 값에 대한 정보를 갖는 flag

* 1. 모듈 이름 : int find\_symbol(char\* operand)
     1. 기능

SYMTAB에서 operand를 찾는다. 있으면 해당 loc값 return, 없으면 error 메시지 출력

* + 1. 사용 변수

Char \*operand //operand에 대한 정보 저장하는 배열

SYM \*cur = first; //현재 노드를 가리키는 포인터

* 1. 모듈 이름 : void change\_hex(char\* bin, char\* hex, int format\_num)
     1. 기능

01010로 된 2진수 array를 hex array로 바꾸어준다.

* + 1. 사용 변수

char\* bin; //2진수 array

char\* hex; //16진수 array

int format\_num //format number 갖는 변수

int k = 0; //hex의 index

int two = 8, num = 0; //16진수로 변환하기 위해 필요한 변수

* 1. 모듈 이름 : char hex\_char\_return(int num)
     1. 기능

입력 받은 int 정수를 판별하여 Hex char로 리턴하여 준다.

* + 1. 사용 변수

Int num; //hex로 바꿀 정수

* 1. 모듈 이름 : void disp\_addr\_set(char \*arr, int num, int format\_type)
     1. 기능

Disp나 addr을 result object code에 hex로 바꾸어 저장.

* + 1. 사용 변수

char \*arr //hex 값을 저장할 array

int num //disp또는 addr 값

int format\_type //format 타입

int i = 3, quotient = 0; //arr의 index, 16진수로 바꿀 때 쓰이는 몫

int hexx=0; //16진수로 바꿀 때 쓰이는 변수

char temp\_num[13] = { 0 }; //음수일 때 2’s complement 사용하기 위한 array

char temp\_hex[3] = { 0 }; // 2진수 16진수로 변환하기 위한 array

* 1. 모듈 이름 : void register\_num(char \*arr, char\* reg)
     1. 기능

Register의 이름을 입력 받으면 해당 숫자로 바꾸어 array에 저장해준다.

* + 1. 사용 변수

char \*arr //숫자로 바꾸어 저장할 array

char\* reg//Register 이름 갖고 있는 배열,

1. **전역 변수, 구조체 정의**
   1. typedef struct \_SYM

symbol들을 입력 받은 경우, Symbol들을 저장하기 위한 linked list가 필요하다. Struct\_SYM은 symbol의 label을 저장하는 char label[31]과 해당 loc 값을 저장하는 int locctr, 그리고 다음 노드를 연결할 struct \_SYM \*next로 구성되어 있다.

* 1. typedef struct \_Modi

object Code의 Text Record 다음에는 format 4에 대한 modification 정보가 필요하다. 따라서 이에 대한 정보를 저장하는 linked list가 필요하다. struct \_Modi는 해당 modification의 시작 주소를 갖는 int start\_addr, 그리고 다음 노드를 연결할 struct \_Modi \*next로 구성되어 있다.

* 1. int loc;

현재 줄의 loc 값을 저장하는 변수이다.

* 1. int start\_address;

.asm 파일의 시작 주소를 저장하는 변수이다.

* 1. int program\_length;

.asm 파일의 총 길이를 저장하는 변수이다.

* 1. int pc;

pc의 값을 저장하는 변수이다.

* 1. int symbol\_flag;

symbol이 들어왔는지 판별하는 flag이다. Symbol이 들어오면 1이 되고, 아니면 0이다.

* 1. char file\_name[31];

파일의 이름을 저장하는 배열이다.

* 1. unsigned char line\_opcode[31];

입력 받은 줄의 opcode를 저장하는 배열이다.

* 1. unsigned char line\_symbol[31];

입력 받은 줄의 symbol을 저장하는 배열이다.

* 1. unsigned char line\_operand[31];

입력 받은 줄의 operand를 저장하는 배열이다.

* 1. int opcode\_val;

opcode의 value 값을 저장하는 변수이다.

* 1. int text\_flag;

text record 작성 시 필요한 flag이다. 같은 라인에 작성하면 1을 갖고 있다.

* 1. char text\_record[61];

Text record에 대한 정보를 갖고 있는 배열이다.

* 1. int text\_index;

text record의 index이다.

* 1. char result\_obj\_code[31];

해당 줄의 object code에 대한 정보를 갖는 배열이다.

* 1. char BASE\_str[31];

BASE 지시어에 들어온 label에 대한 정보를 갖는 배열이다.

* 1. int obj\_loc;

Text record 작성 시 제일 처음에 들어가는 해당 줄의 시작 주소를 갖고 있는 변수이다.

* 1. int asm\_start\_flag;

.asm 파일에 start가 존재하는지 판별하기 위한 flag이다. 존재하면 1이 된다.

* 1. int lst\_line\_num = 5;

해당 줄의 line 번호를 갖고 있는 변수이다.

* 1. int pass\_1\_flag;

pass1이 성공적으로 실행되었는지 알려주는 flag이다. 성공적으로 실행되면 1이 된다.

4.22 int SYMTAB\_flag;

SYMTAB이 정상적으로 생성되었는지 확인하는 flag이다.

1. **코드 설명**
   1. 20161598.h

프로젝트2에서 구현한 부분만 첨부하였다.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/stat.h>

#include <dirent.h>

typedef struct \_Node{

int index;

char comm[100];

struct \_Node\* next;

}Node; //Node : struct which store history

Node\* head; //head is pointing first of history

Node \*prev; //prev is pointing previous node

typedef struct \_Hash {

char mnemonic[7];

int value;

char format;

struct \_Hash\* next;

}Hash; // Hash : struct which store Hash table

Hash\* table[20]; //arr named table storing opcode Hash table

typedef struct \_SYM {

char label[31];

int locctr;

struct \_SYM\* next;

}SYM; // SYM : STMTAB 저장 위한 구조체

SYM\* first; //SYMTAB의 처음 가리킴

SYM\* before; //SYMTAB의 이전 노드 가리킴

typedef struct \_Modi {

int start\_addr;

struct \_Modi\* next;

}Modi; //Modi : Modifcation 저장 위한 구조체

Modi\* modi\_head; //modification linked list의 처음 가리킴

Modi\* modi\_prev; //modification의 이전 노드 가리킴

char command[100]; //variable which get input command

int line\_num; //variable which get history's line index

int quit\_flag; //distinct if input gets 'quit', if 1 program end

int error\_flag; //distinct if error occurs

int memory\_address; //variable which stores last printed memory address+16

unsigned char memory[1048576]; //arr which stores memory

int loc; //현재 줄의 loc 값을 저장하는 변수이다.

int start\_address; //.asm 파일의 시작 주소를 저장하는 변수이다.

int program\_length; //.asm 파일의 총 길이를 저장하는 변수이다.

int pc; // pc의 값을 저장하는 변수이다.

int symbol\_flag; // symbol이 들어왔는지 판별하는 flag이다. Symbol이 들어오면 1이 되고, 아니면 0이다.

char file\_name[31]; //파일의 이름을 저장하는 배열이다.

unsigned char line\_opcode[31]; //입력 받은 줄의 opcode를 저장하는 배열이다.

unsigned char line\_symbol[31]; //입력 받은 줄의 symbol을 저장하는 배열이다

unsigned char line\_operand[31]; //입력 받은 줄의 operand를 저장하는 배열이다.

int opcode\_val; //opcode의 value 값을 저장하는 변수이다.

int text\_flag; //text record 작성 시 필요한 flag이다. 같은 라인에 작성하면 1을 갖고 있다.

char text\_record[61]; //Text record에 대한 정보를 갖고 있는 배열이다.

int text\_index; //text record의 index이다.

char result\_obj\_code[31]; //해당 줄의 object code에 대한 정보를 갖는 배열이다

char BASE\_str[31]; //BASE 지시어에 들어온 label에 대한 정보를 갖는 배열이다.

int obj\_loc; //Text record 작성 시 제일 처음에 들어가는 해당 줄의 시작 주소를 갖고 있는 변수이다.

int asm\_start\_flag; //.asm 파일에 start가 존재하는지 판별하기 위한 flag이다. 존재하면 1이 된다.

int lst\_line\_num = 5; //해당 줄의 line 번호를 갖고 있는 변수이다.

int pass\_1\_flag; //pass1이 성공적으로 실행되었는지 알려주는 flag이다. 성공적으로 실행되면 1이 된다.

int SYMTAB\_flag; //SYMTAB이 정상적으로 생성되었는지 확인하는 flag이다.

void command\_distinct(char\* arr);

void command\_help();

void command\_dir();

void command\_history();

void command\_dump(int num\_flag);

void command\_edit();

void edit\_add\_val(char\* arr, int\* addr, int\* val);

void command\_fill();

void fill\_addr\_val(char\* arr, int\* start, int\* end, int\* val);

void command\_reset();

int dump\_start\_end(char\* arr, int\* start, int\* end);

void print\_memory(int first\_line, int last\_line, int dump\_start, int dump\_end);

void command\_quit();

void make\_hashtb();

void command\_opcodelist();

void command\_opcode();

int distinct\_ascii(char x);

void command\_type();

void command\_assemble();

void get\_filename(char\* arr, int i);

int divide\_one\_line(char\* arr, int pass\_num);

int find\_opcode(char\* arr, int i);

void command\_symbol();

void free\_symtab();

void distinct\_format(char\* opcode, char\* operand);

void num\_to\_hexchar(char\* value, int num);

int find\_symbol(char\* operand);

void change\_hex(char\* bin, char\* hex, int format\_num);

void disp\_addr\_set(char\* arr, int num, int format\_type);

char hex\_char\_return(int num);

void xbpe\_setting(char\* value, int x, int b, int p, int e);

void register\_num(char\* arr, char\* reg);

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

/\* func\_name : command\_distinct \*/

/\* purpose : distinct input command, \*/

/\* call function related to command \*/

/\* return : none \*/

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

void command\_distinct(char\* arr)

{

//if error\_flag is 1, wrong command input

error\_flag = 1;

//만일 type이 들어온 경우, call command\_type()

else if (!strncmp(arr, "type", 4))

{

//If got blank after type

if (arr[4] == ' ')

{

error\_flag--;

command\_type();

}

}

//만일 assemble이 들어온 경우, call command\_assemble()

else if (!strncmp(arr, "assemble", 8))

{

//이전에 assemble한 파일들의 SYMTAB와 modificaiton linked list를 free해야 하므로 아래 함수 call

free\_symtab();

//opcode, operand, symbol and symbol\_flag등을 저장하는 배열 및 flag들 전부 초기화

memset(line\_opcode, 0, sizeof(line\_opcode));

memset(line\_operand, 0, sizeof(line\_operand));

memset(line\_symbol, 0, sizeof(line\_symbol));

memset(file\_name, 0,sizeof(file\_name));

memset(BASE\_str, 0, sizeof(BASE\_str));

memset(text\_record,0,sizeof(text\_record));

symbol\_flag = 0;

asm\_start\_flag = 0;

first = NULL;

before = NULL;

modi\_head = NULL;

modi\_prev = NULL;

pass\_1\_flag=0;

lst\_line\_num = 5;

//assemble 뒤에 빈칸 받은 경우.

if (arr[8] == ' ')

{

error\_flag--;

command\_assemble();

//error\_flag가 0이면 성공적으로 assemble 되었으므로 아래 내용 출력.

if (error\_flag == 0)

{printf("Successfully assemble %s.\n", file\_name);

SYMTAB\_flag = 1;

}

else

{ //pass1 실패시

if(pass\_1\_flag==0)

{

//intermediate 파일 삭제

remove("intermediate");

}

}

}

}

//만일 symbol이 들어온 경우, call command\_symbol()

else if (!strncmp(arr, "symbol", 6))

{

error\_flag--;

command\_symbol();

}

//if error\_flag is 1 then invalid command, so delete the node from the history.

if(error\_flag)

{

free(node);

line\_num--; //decrease history line

//set head to null again if first input and incorrect command.

if (prev == NULL)

head = NULL;

return;

}

//prev is a node that points to a previous one, so insert the current node

prev = node;

}

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

/\* func\_name : command\_type \*/

/\* purpose : 파일을 읽어서 \*/

/\* 파일 내용 출력\*/

/\* return : none \*/

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

void command\_type()

{

//get\_filename 함수로 파일 이름 얻는다.

get\_filename(file\_name, 5);

char temp; //파일의 한 단어를 저장하는 변수

FILE \*fp = fopen(file\_name, "r");

if(fp == NULL) //존재하지 않으면 오류 메시지 출력

printf("Error : No such file in the directory!\n");

else{

//내용 출력

while(fscanf(fp, "%c", &temp)!=EOF)

printf("%c", temp);

}

}

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

/\* func\_name : get\_filename \*/

/\* purpose : 파일 이름을 얻어서 \*/

/\* file\_name 배열에 저장\*/

/\* return : none \*/

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

void get\_filename(char\* arr, int i)

{

int j = 0; //file\_name의 index

//아래 while문으로 file\_name 얻는다.

while (1)

{

if (command[i] == '\0')

break;

file\_name[j++] = command[i++];

}

}

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

/\* func\_name : divide\_one\_line \*/

/\* purpose : 읽은 한 라인을 나누어 opcode, operand, symbol 정보를 얻는다. \*/

/\* return : 0 : opcode=start, 1 : opcode=end, -1 : comment line \*/

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

int divide\_one\_line(char\* arr, int pass\_num)

{

int i = 0, j = 0, k = 0, l = 0, cnt\_blank = 0, comma\_flag = 0; //comma\_flag : 1 -> ,가 있으므로, 빈칸이 있어도 다음 값 받아야함

//만일 pass2에서 불린 경우

if (pass\_num == 2)

{

char line\_loc[4] = { 0 };

//라인 넘버와 빈칸은 skip

while (arr[i++] != '\t');

//이 라인의 loc 값 copy

while (arr[i] != '\t')

line\_loc[j++] = arr[i++];

//loc 값 hexa 로 바꾸어 저장

loc = strtol(line\_loc, NULL, 16);

//loc 이후의 blank skip

i++;

j = 0;

}

//라인이 . 으로 시작하면 주석이다.

if (arr[i] == '.')

return -1;

//빈칸으로 시작하면 opcode와 operand만 존재

else if (arr[i] == ' ' || arr[i]=='\t')

symbol\_flag = 0;

//그게 아니면 ,symbol이다. line\_symbol 에 저장.

else

{

symbol\_flag = 1;

line\_symbol[j++] = arr[i++];

}

int opcode\_cnt = 0; //opcode인지 operand인지 판별 위한 변수.

//라인을 빈칸에 의해 나누어 opcode와 symbol에 관한 정보 얻는다.

while (arr[i] != '\n')

{

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

comma\_flag = 1;

if ((arr[i] == ' ' && comma\_flag==0) || (arr[i]=='\t' && comma\_flag==0))

{

if (symbol\_flag == 1)

cnt\_blank++;

else if (symbol\_flag == 0 && opcode\_cnt != 0)

cnt\_blank++;

while (arr[i] == ' ' || arr[i]=='\t')

i++;

}

//if arr[i]가 symbol

if (symbol\_flag == 1 && cnt\_blank == 0)

line\_symbol[j++] = arr[i];

//if arr[i]가 opcode

else if ((symbol\_flag==1 &&cnt\_blank == 1) || (symbol\_flag == 0 && cnt\_blank == 0))

{

line\_opcode[k++] = arr[i];

opcode\_cnt++;

}

//if arr[i]가 operand

else if ((symbol\_flag == 1 && cnt\_blank == 2) || (symbol\_flag == 0 && cnt\_blank == 1))

{

line\_operand[l++] = arr[i];

//만일 comma 있으면, 다른 operand 받아야함.

if (comma\_flag)

{

i++;

while (arr[i] == ' ' || arr[i]=='\t')

i++;

comma\_flag = 0;

continue;

}

}

i++;

}

//opcode == 'START'

if (!asm\_start\_flag) {

//START 있는 경우

if (!strcmp(line\_opcode, "START"))

{

asm\_start\_flag = 1;

return 0;

}

//START 없으면 오류 메시지 출력

else

{

printf("Assemble Error in line %d : No START opcode\n", lst\_line\_num);

error\_flag = 1;

return -4;

}

}

//opcode가 END인 경우

else if (!strcmp(line\_opcode, "END"))

return 1;

else return 2;

}

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

/\* func\_name : command\_assemble \*/

/\* purpose : 파일을 pass1, pass2를 통해 assemble하고 \*/

/\* .obj, .lst file 만듦 \*/

/\* return : none \*/

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

void command\_assemble()

{

char obj\_name[31] = { 0 };

char lst\_name[31] = { 0 };

get\_filename(file\_name, 9);

int file\_index = 0, x=0; //file\_name, obj\_name, lst\_name의 index

//.asm 없이 obj\_name and lst\_name에 file name copy

while (1)

{

if (file\_name[file\_index] == '.' && file\_name[file\_index+1] == 'a')

break;

if(file\_name[file\_index] == '\0')

break;

obj\_name[x] = file\_name[file\_index];

lst\_name[x++] = file\_name[file\_index++];

}

strcat(obj\_name, ".obj");

strcat(lst\_name, ".lst");

//Pass1

FILE\* fp = fopen(file\_name, "r");

FILE\* inter = fopen("intermediate", "w");

char one\_line[100] = { 0 }; //.asm 파일의 한 라인 정보 받는 배열

//파일 없는 경우, 에러 메시지

if(fp == NULL)

{

printf("File Open error : No such file in the directory!\n");

error\_flag=1;

return;

}

//첫번째 줄 입력

fgets(one\_line, 100, fp);

//If OPCODE = 'START'

if (divide\_one\_line(one\_line,1) == 0)

{

//#[OPERAND] 를 staring address로 저장

start\_address = strtol(line\_operand, NULL, 16);

//loc를 시작 주소로 초기화

loc = start\_address;

}

//Else, LOCCTR을 0으로 초기화

else

{

loc = 0;

error\_flag = 1;

return;

}

//intermediate file에 해당 line 작성

fprintf(inter, "%d\t%.4X\t%s", lst\_line\_num, loc, one\_line);

lst\_line\_num += 5; // 5를 line number에 더함

while (1)

{

//line, opcode, operand, symbol and symbol\_flag에 관한 정보 갖는 array 초기화

memset(one\_line, 0, sizeof(one\_line));

memset(line\_opcode, 0, sizeof(line\_opcode));

memset(line\_operand, 0, sizeof(line\_operand));

memset(line\_symbol, 0, sizeof(line\_symbol));

symbol\_flag = 0;

//한 줄 입력

if (fgets(one\_line, 100, fp) == NULL)

{

//NULL이면 end 메시지 없는 것이므로 오류 메시지 출력.

printf("Assemble Error on line %d : No END opcode in the .asm file!\n", lst\_line\_num);

error\_flag = 1;

return;

}

//symbol, opcode, operand에 대한 정보 divide\_one\_line function 함수 call 하여 얻는다.

int end\_flag = divide\_one\_line(one\_line,1);

//OPCODE = END

if (end\_flag == 1)

{

//마지막 line을 intermediate file에 쓴다.

fprintf(inter, "%d\t \t%s", lst\_line\_num, one\_line);

break;

}

//COMMENT STATEMENTS

else if (end\_flag == -1)

{

//현재 라인을 intermediate file에 쓴다.

fprintf(inter, "%d\t \t%s", lst\_line\_num, one\_line);

lst\_line\_num += 5; //line number에 5 더함.

continue;

}

else if (end\_flag == 2)

{

//LABEL field에 symbol 있는 경우

if (symbol\_flag)

{

//SYMTAB에서 label 찾음

SYM\* sym\_cur = first;

if (first)

{

while (sym\_cur != NULL) {

if (!strcmp(sym\_cur->label, line\_symbol))

{

//만일 찾은 경우, 중복 되므로 에러 메시지 출력

printf("Assemble Error on line %d : same label has many LOC\n", lst\_line\_num);

error\_flag = 1;

return;

}

sym\_cur = sym\_cur->next;

}

}

//LABEL,LOCCTR)을 SYMTAB에 insert

int insert\_flag = 0; //SYMTAB에 insert하면 1로 바뀜

//insert하기 위해 새로운 node 생성

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

strcpy(node->label, line\_symbol);

node->locctr = loc;

node->next = NULL;

//처음 head가 없는 경우, 처음에다가 노드 insert

if (first == NULL)

{

first = node;

insert\_flag = 1;

}

SYM\* cur = first; //SYMTAB에서 현재 노드 가리킴

before = first;

int cnt = 0; //SYMTAB에서 얼마나 많은 노드를 지나갔는지 알기 위한 변수

//SYMTAB에 insert하기 전까지 계속 반복

while (!insert\_flag) {

//만일 새로운 label이 현재 노드 label 보다 작은 경우, insert(오름차순)

int ret = strcmp(node->label, cur->label);

if (ret < 0)

{

//SYMTAB의 처음에 대입

if (before == first && cnt==0){

node->next = cur;

first = node;

}

//처음 아닌 곳에 대입

else{

before->next = node;

node->next = cur;

}

insert\_flag = 1;

break;

}

//큰 경우, 오름차순이 안되므로 다음 노드로 이동

//cur와 before가 다른 경우, before 이동

if (cur != before)

before = before->next;

//cur를 다음 노드로 이동

cur = cur->next;

cnt++;

//마지막에 삽입

if(cur == NULL)

{

before->next = node;

insert\_flag = 1;

break;

}

}

}

if (!strcmp(line\_opcode, "WORD"))

{

//숫자가 들어온게 아니면 오류 메시지 출력.

if ((int)line\_operand[0] >= 58)

{

printf("Assemble Error in line %d : Only Decimal number is possible for WORD constant\n", lst\_line\_num);

error\_flag = 1;

return;

}

//intermediate file에 작성

fprintf(inter, "%d\t%.4X\t%s", lst\_line\_num, loc, one\_line);

lst\_line\_num += 5; //line number에 5 더함

loc += 3; //LOCCTR에 3 더함.

}

else if (!strcmp(line\_opcode, "RESW"))

{

//intermediate file에 작성

fprintf(inter, "%d\t%.4X\t%s", lst\_line\_num, loc, one\_line);

lst\_line\_num += 5; //line number에 5 더함

loc += 3 \* atoi(line\_operand); //loc에 operand 십진수 값 \* 3 더함.

}

else if (!strcmp(line\_opcode, "RESB"))

{

//intermediate file에 작성

fprintf(inter, "%d\t%.4X\t%s", lst\_line\_num, loc, one\_line);

lst\_line\_num += 5; //line number에 5 더함

loc += atoi(line\_operand); //loc에 operand 십진수 값 더함

}

else if (!strcmp(line\_opcode, "BYTE"))

{

//상수의 길이 찾음

int constant\_length = 0;

//만일 operand가 C인 경우

if (line\_operand[0] == 'C')

{

int i = 2; //line operand의 index

//아래 while문으로 길이 찾음

while (line\_operand[i++] != '\'')

constant\_length++;

//intermediate file에 작성

fprintf(inter, "%d\t%.4X\t%s", lst\_line\_num, loc, one\_line);

lst\_line\_num += 5; //line number에 5 더함

//char는 1 byte, 따라서 loc + 1\*constant\_legnth

loc += constant\_length;

}

//operand가 X인 경우

else if (line\_operand[0] == 'X')

{

int i = 2; //line operand의 index

//아래 while문으로 길이 찾음

while (line\_operand[i++] != '\'')

constant\_length++;

//intermediate file에 작성

fprintf(inter, "%d\t%.4X\t%s", lst\_line\_num, loc, one\_line);

lst\_line\_num += 5; //line number에 5 더함

//constant는 하프바이트 단위로 쓰인다. 따라서 짝수가 아니면 1을 추가해줘야한다.

if (constant\_length % 2 == 1)

constant\_length += 1;

loc += ((constant\_length) / 2);

}

}

else if (!strcmp(line\_opcode, "BASE"))

{

//BASE의 operand BASE\_str에 copy

strcpy(BASE\_str, line\_operand);

//intermediate file에 작성

fprintf(inter, "%d\t \t%s", lst\_line\_num, one\_line);

lst\_line\_num += 5; //line number에 5 더함

continue;

}

else if (line\_opcode[0] == '+') {

//temp\_opcode opcde table에서 찾음, 0이 되면 해당 opcode 없음

if (find\_opcode(line\_opcode, 1) == 0)

{

printf("Assembly File Error : No %s on the opcodelist\n", line\_opcode);

error\_flag = 1;

return;

}

//intermediate file에 작성

fprintf(inter, "%d\t%.4X\t%s", lst\_line\_num, loc, one\_line);

lst\_line\_num += 5; //line number에 5 더함

//상수 immediate addressing이 아니면 format 4 modification 정보 저장.

if (!(line\_operand[0] == '#' && (line\_operand[1] >= '0' && line\_operand[1] <= '9'))) {

//modification 위한 dynamic memory 할당

Modi\* modi\_node = (Modi\*)malloc(sizeof(Modi));

modi\_node->start\_addr = loc+1;

modi\_node->next = NULL;

//modification 위한 linked list 생성

if (modi\_head == NULL)

modi\_head = modi\_node;

else

{

Modi\* cur = modi\_head;

modi\_prev = modi\_head;

while (1)

{

cur = cur->next;

if (cur == NULL)

{

modi\_prev->next = modi\_node;

break;

}

modi\_prev = modi\_prev->next;

}

}

}

//opcode에서 찾았으면, loc에 4더함.

loc += 4;

}

//format 1, 2, 3 instruction

else{

//opcode opcode table에서 찾음

int opcode\_found\_flag = find\_opcode(line\_opcode, 0);

//없는 경우, 오류 메시지 출력

if (opcode\_found\_flag == 0)

{

printf("Assembly Error in line %d : No %s on the opcodelist\n",lst\_line\_num, line\_opcode);

error\_flag = 1;

return;

}

//찾은 경우.

//intermediate file에 작성

fprintf(inter, "%d\t%.4X\t%s", lst\_line\_num, loc, one\_line);

lst\_line\_num += 5; //line number에 5 더함

if (opcode\_found\_flag == 1)

loc += 1; //add 1(format 1) to LOCCTR

else if (opcode\_found\_flag == 2)

loc += 2; //add 2 (register instruction) to LOCCTR

else

loc += 3; //add 3 (instruction length) to LOCCTR

}

}

}

//Close File Pointer

fclose(fp);

fclose(inter);

//program의 총 길이 program\_length에 저장

program\_length = loc - start\_address;

//에러가 없으면 pass2 실행

if(!error\_flag){

pass\_1\_flag=1;

FILE\* interm = fopen("intermediate", "r");

FILE\* obj\_fp = fopen(obj\_name, "w");

FILE\* lst\_fp = fopen(lst\_name, "w");

//line, opcode, operand, symbol and symbol\_flag에 관한 값 초기화

memset(one\_line, '\0', sizeof(one\_line));

memset(line\_opcode, '\0', sizeof(line\_opcode));

memset(line\_operand, '\0', sizeof(line\_operand));

memset(line\_symbol, '\0', sizeof(line\_symbol));

symbol\_flag = 0;

int variable\_flag = 0; //만일 변수면, 해당 object line 적기 위해 쓰이는 flag

int printed\_flag = 0; //만일 아직 안쓰인 object line이 있는데 end가 들어온 경우, object line print하기 위해 쓰이는 flag

asm\_start\_flag = 0;

//첫번째 라인 입력

fgets(one\_line, 100, interm);

//If OPCODE = 'START'

if (divide\_one\_line(one\_line,2) == 0)

{

//listing line 작성

fprintf(lst\_fp, "%s", one\_line);

//.obj header record 작성

fprintf(obj\_fp, "H%-6s%.6X%.6X\n", line\_symbol, start\_address, program\_length);

}

lst\_line\_num = 5;

//opcode가 end일 때까지 계속 반복

while (!error\_flag)

{

//line, opcode, operand, symbol, result\_obj\_code에 관한 값 초기화

memset(one\_line, 0, sizeof(one\_line));

memset(line\_opcode, 0, sizeof(line\_opcode));

memset(line\_operand, 0, sizeof(line\_operand));

memset(line\_symbol, 0, sizeof(line\_symbol));

memset(result\_obj\_code, 0, sizeof(result\_obj\_code));

symbol\_flag = 0;

lst\_line\_num += 5;

//다음 줄 입력

fgets(one\_line, 100, fp);

//ymbol, opcode, operand에 관한 정보 divide\_one\_line function 통해 얻음

int end\_flag = divide\_one\_line(one\_line, 2);

//OPCODE = END

if (end\_flag == 1)

{

//END line은 object code 생성 안함.

//아직 안 쓰인 text record가 있으면 해당 내용 적음

if (printed\_flag != 0) {

fprintf(obj\_fp, "T%.6X", obj\_loc);

text\_record[0] = hex\_char\_return(((text\_index - 2)/2) / 16);

text\_record[1] = hex\_char\_return(((text\_index - 2)/2) % 16);

fprintf(obj\_fp, "%s\n", text\_record);

}

//Modification Record 작성

Modi\* modi\_cur = modi\_head;

while(modi\_cur != NULL)

{

fprintf(obj\_fp, "M%.6X05\n", modi\_cur->start\_addr);

modi\_cur = modi\_cur->next;

}

//해당 줄 .lst 파일에 작성

fprintf(lst\_fp, "%s", one\_line);

fprintf(obj\_fp, "E%.6X\n", start\_address);

break;

}

//COMMENT STATEMENTS

else if (end\_flag == -1)

{

//주석은 object code 만들지 않음

//해당 줄 .lst 파일에 작성

fprintf(lst\_fp, "%s", one\_line);

}

else {

//if opcode is BASE, object code 만들지 않음

if (!(strcmp(line\_opcode, "BASE")))

{

//BASE 뒤에 나오는 symbol이 SYMTAB에 없으면 오류 메시지 출력

int find = find\_symbol(line\_operand);

if (find == -1)

{

printf("Assemble Error in line %d : No such symbol in SYMTAB!\n", lst\_line\_num);

error\_flag = 1;

return;

}

fprintf(lst\_fp, "%s", one\_line);

continue;

}

//if opcode is RESW, object code 만들지 않음

else if (!(strcmp(line\_opcode, "RESW")))

{

fprintf(lst\_fp, "%s", one\_line);

if (!variable\_flag) {

//변수이면, 새로운 text record line 만듦. text\_record는 .obj 파일에 적음

fprintf(obj\_fp, "T%.6X", obj\_loc);

text\_record[0] = hex\_char\_return(((text\_index - 2)/2) / 16);

text\_record[1] = hex\_char\_return(((text\_index - 2)/2) % 16);

fprintf(obj\_fp, "%s\n", text\_record);

text\_flag = 0;

variable\_flag = 1;

printed\_flag = 0;

}

continue;

}

//if opcode is RESB, object code 만들지 않음

else if (!(strcmp(line\_opcode, "RESB")))

{

fprintf(lst\_fp, "%s", one\_line);

if (!variable\_flag) {

///변수이면, 새로운 text record line 만듦. text\_record는 .obj 파일에 적음

fprintf(obj\_fp, "T%.6X", obj\_loc);

text\_record[0] = hex\_char\_return(((text\_index - 2)/2) / 16);

text\_record[1] = hex\_char\_return(((text\_index - 2)/2) % 16);

fprintf(obj\_fp, "%s\n", text\_record);

text\_flag = 0;

variable\_flag = 1;

printed\_flag = 0;

}

continue;

}

//obj 코드 계산, 아래 distinc\_format 함수 통해서.

distinct\_format(line\_opcode, line\_operand);

//one line에 \n 제거

int one\_line\_index = 0;

while (1)

{

if (one\_line[one\_line\_index] == '\n')

{

one\_line[one\_line\_index] = '\0';

break;

}

one\_line\_index++;

}

//.lst 파일에 현재 라인 적음

fprintf(lst\_fp, "%-35s\t%-35s\n", one\_line, result\_obj\_code);

printed\_flag++;

//변수가 나왔다면 이전까지 만든 text record 프린트 되었을 테니까 다시 0으로 만듦

if (variable\_flag)

variable\_flag = 0;

int obj\_index = 0, cnt=0; //result\_obj\_code의 index, object code의 길이

while(result\_obj\_code[obj\_index++] != '\0') //object code 길이 계산.

cnt++;

//만일 이 object code 적었을 때 max 넘어가면 이 object code 이전까지의 text record .obj에 작성

if (text\_index + cnt - 1 > 60)

{

if (!variable\_flag)

{

fprintf(obj\_fp, "T%.6X", obj\_loc);

text\_record[0] = hex\_char\_return(((text\_index - 2)/2) / 16);

text\_record[1] = hex\_char\_return(((text\_index - 2)/2) % 16);

fprintf(obj\_fp, "%s\n", text\_record);

printed\_flag = 0;

text\_flag = 0;

}

}

//새로운 text record line 만듦

if (!text\_flag)

{

//text\_record and text\_index 초기화

memset(text\_record, 0, sizeof(text\_record));

text\_index = 2;

text\_flag = 1;

obj\_loc = loc;

}

obj\_index = 0;

//object code를 text\_record array에 copy

while (result\_obj\_code[obj\_index] != '\0')

text\_record[text\_index++] = result\_obj\_code[obj\_index++];

}

}

//Close File Pointer

fclose(interm);

fclose(obj\_fp);

fclose(lst\_fp);

}

//intermediate file 제거

remove("intermediate");

if(error\_flag)

{

//pass2에서 error가 났으므로 .obj, .lst 파일 제거

remove(obj\_name);

remove(lst\_name);

}

}

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

/\* func\_name : find\_opcode \*/

/\* purpose : 들어온 opcode를 opcode table에서 찾음 \*/

/\* return : 못 찾은 경우 0, 찾은 경우 format type 반환 \*/

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

int find\_opcode(char \*arr, int i) {

char temp\_mnemonic[7] = { 0 }; //들어온 mnemonic을 저장하는 string

int mnemonic\_num = 1; //hash table의 index를 판별하기 위한 변수

int mnemonic\_format = 0; // mnemonic의 format을 저장하는 변수

int j = 0, mnemonic\_length = 0; //temp\_mnemonic의 index, mnemonic 길이

//들어온 mnemominc을 temp\_mnemonic array에 copy

while (arr[i] != '\0')

{

//잘못된 mnemonic이 들어온 경우

if (j == 6)

return 0;

temp\_mnemonic[j++] = arr[i];

mnemonic\_num += (int)arr[i++];

mnemonic\_length++;

}

//menmonic의 hash index 계산

mnemonic\_num += mnemonic\_length \* 19;

mnemonic\_num %= 20;

Hash\* cur = table[mnemonic\_num]; //cur은 현재 hash table node 가리킴

while (1)

{

//못 찾은 경우, 에러 메시지 출력

if (cur == NULL)

{

return 0;

break;

}

//찾은 경우

if (!(strcmp(cur->mnemonic, temp\_mnemonic)))

{

//format number와 value 값 저장

mnemonic\_format = (int)cur->format - 48;

opcode\_val = cur->value;

break;

}

cur = cur->next;

}

return mnemonic\_format;

}

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

/\* func\_name : command\_symbol \*/

/\* purpose : SYMTAB의 모든 label과 loc 오름차순 출력 \*/

/\* return : none \*/

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

void command\_symbol() {

SYM\* cur = first;

//SYMTAB이 있는 경우, 출력

if (first) {

for (; cur != NULL; cur = cur->next)

printf("\t%s\t%.4X\n", cur->label, cur->locctr);

}

else

{

//SYMTAB이 없는 경우

error\_flag = 1;

printf("Error : No SYMTAB!\n");

}

}

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

/\* func\_name : free\_symtab \*/

/\* purpose : SYMTAB과 modification linked list 전부 free \*/

/\* return : none \*/

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

void free\_symtab()

{

//SYMTAB이 존재하면 전부 free

if (first)

{

SYM\* cur = first; //cur은 현재 노드 가리킴

while (1)

{

if (cur == NULL)

break;

SYM\* previous = cur; //previous은 이전 노드 가리킴

cur = cur->next;

free(previous);

}

}

//Modification의 linked list가 존재하면 전부 free

if (modi\_head)

{

Modi\* modi\_cur = modi\_head; //cur은 현재 노드 가리킴

while (1)

{

if (modi\_cur == NULL)

break;

modi\_prev = modi\_cur; //prev은 이전 노드 가리킴

modi\_cur = modi\_cur->next;

free(modi\_prev);

}

}

}

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

/\* func\_name : distinct\_format \*/

/\* purpose : opcode와 operand를 받아서 \*/

/\* object code를 만듦 \*/

/\* return : object code character array \*/

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

void distinct\_format(char\* opcode, char\*operand)

{

char result\_obj\_before\_bind[130] = { 0 }; //4개씩 묶기 전의 10101으로 된 object 코드

int opcode\_format = 0; //opcode의 format number 저장

int x\_flag = 0; //x register 사용 유무

int b\_flag = 0; //BASE relative인지 아닌지

int p\_flag = 0; //PC relative인지 아닌지

int e\_flag = 0; //Extension인지 아닌지

//Format4

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

{

int addr = 0; //format4에서의 주소

find\_opcode(opcode, 1);

//opcode value를 result\_obj\_before\_bind array에 저장

num\_to\_hexchar(result\_obj\_before\_bind, opcode\_val);

//format 4, extension, 주소의 절댓값 사용

e\_flag = 1;

//Indirect addresssing

if (operand[0] == '@')

{

char pure\_operand[31] = { 0 }; //만일 #이나 @가 붙은 경우, 이걸 제외한 순수한 operand 값 저장 배열

//indirect addressing의 n, i value 저장

result\_obj\_before\_bind[6] = '1'; result\_obj\_before\_bind[7] = '0';

//x,b,p,e value 저장

xbpe\_setting(result\_obj\_before\_bind, x\_flag, b\_flag, p\_flag, e\_flag);

int operand\_index = 1, pure\_index = 0;

//@를 제외하고 pure\_operand에 operand 내용 저장

while (operand[operand\_index] != '\0')

pure\_operand[pure\_index++] = operand[operand\_index++];

int find = find\_symbol(pure\_operand);

if (find == -1)

{

printf("Assemble Error in line %d : No such symbol in SYMTAB\n", lst\_line\_num);

error\_flag = 1;

return;

}

else

addr = find;

}

//Immediate addressing

else if (operand[0] == '#')

{

char pure\_operand[31] = { 0 }; //만일 #이나 @가 붙은 경우, 이걸 제외한 순수한 operand 값 저장 배열

//immediate addressing의 n, i value 저장

result\_obj\_before\_bind[6] = '0'; result\_obj\_before\_bind[7] = '1';

int s\_flag = 0;

//symbol이 들어온 경우 s\_flag = 1

if ((int)operand[1] >= 65)

s\_flag = 1;

int operand\_index = 1, pure\_index = 0;

//#를 제외하고 pure\_operand에 operand 내용 저장

while (operand[operand\_index] != '\0')

pure\_operand[pure\_index++] = operand[operand\_index++];

//x,b,p,e value 저장

xbpe\_setting(result\_obj\_before\_bind, x\_flag, b\_flag, p\_flag, e\_flag);

//operand의 disp 저장

if (s\_flag)

{

int find = find\_symbol(pure\_operand);

if (find == -1)

{

printf("Assemble Error in line %d : No such symbol in SYMTAB!\n", lst\_line\_num);

error\_flag = 1;

return;

}

else

addr = find;

}

else

addr = atoi(pure\_operand);

}

else {

//n, i 값 저장

result\_obj\_before\_bind[6] = '1'; result\_obj\_before\_bind[7] = '1';

//x,b,p,e value 저장

xbpe\_setting(result\_obj\_before\_bind, x\_flag, b\_flag, p\_flag, e\_flag);

//SYMTAB에서 symbol 찾음

addr = find\_symbol(operand);

if (addr == -1)

{

printf("Assemble Error in line %d : No such symbol in SYMTAB!\n", lst\_line\_num);

error\_flag = 1;

return;

}

}

if (addr >= 1048576)

{

printf("Assemble Error in the line %d : Address out of bound!\n", lst\_line\_num);

error\_flag = 1;

return;

}

//binary number를 hexa number로 바꿈

change\_hex(result\_obj\_before\_bind, result\_obj\_code, 3);

//disp number 저장

disp\_addr\_set(result\_obj\_code, addr,4);

}

//Byte 상수가 들어온 경우

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

{

int i = 2; //operand의 index

int j = 0; //result\_object\_code의 index

//C 상수인 경우

if (operand[0] == 'C')

{

//아래 while문 통해서 16진수로 result\_obj\_code에 저장

while (operand[i] != '\'')

{

result\_obj\_code[j++] = hex\_char\_return(((int)operand[i] / 16));

result\_obj\_code[j++] = hex\_char\_return(((int)operand[i++] % 16));

}

}

//16진수 상수인 경우

else if (operand[0] == 'X')

{

int operand\_length = 0;

//operand 길이 계산

while(operand[i++] != '\'')

operand\_length++;

//홀수면 앞에 0붙임, half byte 단위이므로 짝수여야함.

if (operand\_length % 2 == 1)

result\_obj\_code[j++] = '0';

i = 2;

//16진수 값을 result\_obj\_code에 저장.

while (operand[i] != '\'')

result\_obj\_code[j++] = operand[i++];

}

}

//WORD 상수가 들어온 경우

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

{

//num에 operand 10진수 값 저장.

int num = atoi(operand);

int result\_index = 0; //result\_obj\_code의 index

//num을 6 byte hex character array로 바꿈

int hexx = 65536; //6byte 16진수로 바꾸기 위해 필요한 변수

while (1)

{

if (num == 0)

break;

int quotient = num / hexx;

num = num - (quotient \* hexx);

result\_obj\_code[result\_index++] = hex\_char\_return(quotient);

hexx /= 16;

}

}

else

{

//opcode의 format과 value 값을 받는다.

opcode\_format = find\_opcode(opcode, 0);

//opcode의 value 값을 result\_obj\_before\_bind array에 저장.

num\_to\_hexchar(result\_obj\_before\_bind, opcode\_val);

//Format 1

if (opcode\_format == 1)

{

//opcode만 존재 1 byte

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

{

printf("Assemble Error in the line %d : Format 1 cannot have operand!\n", lst\_line\_num);

error\_flag = 1;

return;

}

//pc 는 next instruction 가리킴

pc = loc + 1;

//binary number를 hexa number로 바꿈

change\_hex(result\_obj\_before\_bind, result\_obj\_code, 1);

}

//Format2

else if (opcode\_format == 2)

{

//pc 는 next instruction 가리킴

pc = loc + 2;

//Register instruction

int comma\_flag = 0; //1: 2개의 register가 operand에 있음을 의미

int operand\_index = 0; //operand의 index

char register\_arr[3] = { 0 }; //register 타입 저장하는 arra

int register\_index = 0; //register의 index 저장하는 변수

while (operand[operand\_index] != '\0')

{

if (operand[operand\_index] == ',')

{

//register number를 result\_obj\_before\_bind arr에 저장

register\_num(result\_obj\_before\_bind, register\_arr);

register\_index = 0;

operand\_index++;

comma\_flag++;

}

register\_arr[register\_index++] = operand[operand\_index++];

}

//register number를 result\_obj\_before\_bind arr에 저장

if (comma\_flag)

register\_num(result\_obj\_before\_bind, register\_arr);

else

{

//하나의 register만 존재하는 경우

register\_num(result\_obj\_before\_bind, register\_arr);

//register 2에는 전부 0을 넣음

strcat(result\_obj\_before\_bind, "0000");

}

//binary number 에서 hexa number로 바꿈

change\_hex(result\_obj\_before\_bind, result\_obj\_code,2);

}

//Format3

else

{

//pc는 다음 instruction 가리킴

pc = loc + 3;

p\_flag = 1;

int disp = 0;

//operand가 없는 경우

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

{

//n, i value 저장

result\_obj\_before\_bind[6] = '1'; result\_obj\_before\_bind[7] = '1';

//pc relative가 아님

p\_flag = 0;

//x,b,p,e value 저장

xbpe\_setting(result\_obj\_before\_bind, x\_flag, b\_flag, p\_flag, e\_flag);

//binary number를 hexa number로 바꾸어줌

change\_hex(result\_obj\_before\_bind, result\_obj\_code, 3);

//"000"을 result\_obj\_code 마지막에 붙여줌

strcat(result\_obj\_code, "000");

return;

}

//Indirect addresssing

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

{

char pure\_operand[31] = { 0 }; //@ 제외한 operand를 저장하는 배열

//n, i value 값 저장

result\_obj\_before\_bind[6] = '1'; result\_obj\_before\_bind[7] = '0';

//indirect addressing, pc relative

p\_flag = 1;

//x,b,p,e value 저장

xbpe\_setting(result\_obj\_before\_bind, x\_flag, b\_flag, p\_flag, e\_flag);

int operand\_index = 1, pure\_index = 0;

//operand를 pure\_operand에 #, @ 제외하고 저장

while (operand[operand\_index] != '\0')

pure\_operand[pure\_index++] = operand[operand\_index++];

//symbol을 SYMTAB에서 찾음

int find = find\_symbol(pure\_operand);

if (find == -1)

{

//symbol이 SYMTAB에 없는 경우

printf("Assemble Error in line %d : No such symbol in SYMTAB!\n", lst\_line\_num);

error\_flag = 1;

return;

}

else

disp = find - pc; //symbol의 loc에서 pc 값 빼서 disp 계산

}

//Immediate addressing

else if(operand[0]=='#')

{

char pure\_operand[31] = { 0 }; //# 제외한 operand를 저장하는 배열

//n, i value 값 저장

result\_obj\_before\_bind[6] = '0'; result\_obj\_before\_bind[7] = '1';

//operand로 symbol 받으면, pc relative

if ((int)operand[1] >= 65)

p\_flag = 1;

//operand로 상수 받으면, pc relative 아님

else

p\_flag = 0;

int operand\_index = 1, pure\_index=0;

//operand를 pure\_operand에 #, @ 제외하고 저장

while (operand[operand\_index] != '\0')

pure\_operand[pure\_index++] = operand[operand\_index++];

//x,b,p,e value 저장

xbpe\_setting(result\_obj\_before\_bind, x\_flag, b\_flag, p\_flag, e\_flag);

//operand의 disp 저장

if (p\_flag)

{

//symbol을 SYMTAB에서 찾음

int find = find\_symbol(pure\_operand);

if (find == -1)

{

//symbol이 SYMTAB에 없는 경우

printf("Assemble Error in line %d : No such symbol in SYMTAB!\n",lst\_line\_num);

error\_flag = 1;

return;

}

else

disp = find\_symbol(pure\_operand) - pc; //symbol의 loc에서 pc 값 빼서 disp 계산

}

else

disp = atoi(pure\_operand); //pc relative 아니면 operand의 10진수 값 그대로 저장.

}

//Simple addressing

else

{

//n, i value 값 저장

result\_obj\_before\_bind[6] = '1'; result\_obj\_before\_bind[7] = '1';

char pure\_operand[31] = { 0 }; //, 제외한 operand를 저장하는 배열

int operand\_index = 0, pure\_index = 0, comma\_flag=0; //operand and pure\_operand의 index, comma가 존재할 때 comma\_flag는 1이 됨.

//operand를 pure\_operand에 , 제외하고 저장

while (operand[operand\_index] != '\0')

{

if (operand[operand\_index] == ',')

{

comma\_flag = 1;

operand\_index++;

}

//comma와 X가 존재하면, then it x flag는 1이 됨

if (comma\_flag == 1 && operand[operand\_index] == 'X')

x\_flag = 1;

else if (comma\_flag == 1 && operand[operand\_index] != 'X' && operand[operand\_index] != ' ')

{

//comma 이후 잘못된 instruction이 들어옴, 에러 메시지 출력

printf("Assemble Error in the line %d : wrong register after , !\n",lst\_line\_num);

error\_flag = 1;

return;

}

if(!comma\_flag)

pure\_operand[pure\_index++] = operand[operand\_index];

operand\_index++;

}

//symbol을 SYMTAB에서 찾음

int symbol\_length = find\_symbol(pure\_operand);

if (symbol\_length == -1)

{

//symbol이 SYMTAB에 없는 경우

printf("Assemble Error in line %d : No such symbol in SYMTAB!\n",lst\_line\_num);

error\_flag = 1;

return;

}

//symbol loc 이용하여 disp 계산

disp = symbol\_length - pc;

//만일 disp의 절댓값이 2047을 넘는 경우, format 3가 불가능하므로 BASE relative

if (abs(disp)>2047)

{

//disp가 음수 -2048인 경우는 format 3가능.

if (disp < 0 && disp == 0x8000)

{

b\_flag = 0;

p\_flag = 0;

}

else{

//아닌 경우, 불가능하므로 BASE relative

b\_flag = 1;

p\_flag = 0;}

}

//x,b,p,e value 저장

xbpe\_setting(result\_obj\_before\_bind, x\_flag, b\_flag, p\_flag, e\_flag);

//If BASE relative

if (b\_flag)

{

//BASE의 symbol을 SYMTAB에서 찾음.

int find = find\_symbol(BASE\_str);

if (find == -1)

{

//symbol이 SYMTAB에 없는 경우

printf("Assemble Error in line %d : No such symbol in SYMTAB!\n",lst\_line\_num);

error\_flag = 1;

return;

}

else

{

//BASE relative로 disp 계산

disp = symbol\_length - find;

//여전히 범위 넘기면 오류 메시지 출력

if (abs(disp) > 2047)

{

if(disp > 0 || (disp <0 && abs(disp)>2048)){

printf("Assemble Error in line %d : Address out of bound!\n", lst\_line\_num);

error\_flag = 1;

return;}

}

}

}

}

//binary number를 hexa number로 변환

change\_hex(result\_obj\_before\_bind, result\_obj\_code, 3);

//disp number를 result\_obj\_code에 저장.

disp\_addr\_set(result\_obj\_code, disp, 3);

}

}

}

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

/\* func\_name : num\_to\_hexchar \*/

/\* purpose : 받은 int 정수를 2진수로 바꾸어 value array에 저장. \*/

/\* return : none \*/

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

void num\_to\_hexchar(char\* value, int num)

{

int x = 0, i=0; //x는 16으로 나눈 몫이나 나머지 저장, i는 value arr의 index

for (int cnt = 0; cnt < 2; cnt++)

{

if (cnt == 0)

x = num / 16; //16으로 나눈 몫

else

x = num % 16; //16으로 나눈 나머지

switch (x) {

case 0:

value[i + 0] = '0'; value[i + 1] = '0'; value[i + 2] = '0'; value[i + 3] = '0';

break;

case 1:

value[i + 0] = '0'; value[i + 1] = '0'; value[i + 2] = '0'; value[i + 3] = '1';

break;

case 2:

value[i + 0] = '0'; value[i + 1] = '0'; value[i + 2] = '1'; value[i + 3] = '0';

break;

case 3:

value[i + 0] = '0'; value[i + 1] = '0'; value[i + 2] = '1'; value[i + 3] = '1';

break;

case 4:

value[i + 0] = '0'; value[i + 1] = '1'; value[i + 2] = '0'; value[i + 3] = '0';

break;

case 5:

value[i + 0] = '0'; value[i + 1] = '1'; value[i + 2] = '0'; value[i + 3] = '1';

break;

case 6:

value[i + 0] = '0'; value[i + 1] = '1'; value[i + 2] = '1'; value[i + 3] = '0';

break;

case 7:

value[i + 0] = '0'; value[i + 1] = '1'; value[i + 2] = '1'; value[i + 3] = '1';

break;

case 8:

value[i + 0] = '1'; value[i + 1] = '0'; value[i + 2] = '0'; value[i + 3] = '0';

break;

case 9:

value[i + 0] = '1'; value[i + 1] = '0'; value[i + 2] = '0'; value[i + 3] = '1';

break;

case 10:

value[i + 0] = '1'; value[i + 1] = '0'; value[i + 2] = '1'; value[i + 3] = '0';

break;

case 11:

value[i + 0] = '1'; value[i + 1] = '0'; value[i + 2] = '1'; value[i + 3] = '1';

break;

case 12:

value[i + 0] = '1'; value[i + 1] = '1'; value[i + 2] = '0'; value[i + 3] = '0';

break;

case 13:

value[i + 0] = '1'; value[i + 1] = '1'; value[i + 2] = '0'; value[i + 3] = '1';

break;

case 14:

value[i + 0] = '1'; value[i + 1] = '1'; value[i + 2] = '1'; value[i + 3] = '0';

break;

case 15:

value[i + 0] = '1'; value[i + 1] = '1'; value[i + 2] = '1'; value[i + 3] = '1';

break;

}

i += 4; //다음 4개의 index로 간다.

}

}

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

/\* func\_name : xbpe\_setting \*/

/\* purpose : x,b,p,e value에 따라 xbpe 값을 set \*/

/\* return : none \*/

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

void xbpe\_setting(char \*value, int x, int b, int p, int e)

{

if (x == 1)

value[8] = '1';

else

value[8] = '0';

if (b == 1)

value[9] = '1';

else

value[9] = '0';

if (p == 1)

value[10] = '1';

else

value[10] = '0';

if (e == 1)

value[11] = '1';

else

value[11] = '0';

}

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

/\* func\_name : find\_symbol \*/

/\* purpose : SYMTAB에서 symbol을 찾고 \*/

/\* symbol's LOCCTR을 리턴 \*/

/\* return : symbol's LOCCTR \*/

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

int find\_symbol(char\* operand) {

SYM\* cur=first;

while (cur != NULL)

{

//찾은 경우, loc 값 return

if (!(strcmp(cur->label, operand)))

return cur->locctr;

cur = cur->next;

}

//찾지 못하면 return -1

return -1;

}

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

/\* func\_name : change\_hex \*/

/\* purpose : 2진수로 된 array를 16진수로 바꾸어서 저장. \*/

/\* return : none \*/

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

void change\_hex(char\* bin, char\* hex, int format\_num)

{

int k = 0; //16진수 array의 index

//만일 format\_number가 1이면, 8 비트만 16진수로 바꾸면 됨.

if (format\_num == 1)

{

for (int i = 0; i < 8; )

{

int two = 8, num = 0;

for (int j = 0; j < 4; j++)

{

num += two \* ((int)bin[i + j] - 48);

two /= 2;

}

hex[k++] = hex\_char\_return(num);

i += 4;

}

}

//만일 format\_number가 2이면, 16비트를 16진수로 바꾸어준다.

else if (format\_num == 2)

{

for (int i = 0; i < 15; )

{

int two = 8, num = 0;

for (int j = 0; j < 4; j++)

{

num += two \* ((int)bin[i + j] - 48);

two /= 2;

}

hex[k++] = hex\_char\_return(num);

i += 4;

}

}

//format 1,2가 아니면 12bit를 16진수로 바꾸어준다.

else

{

for (int i = 0; i < 11; )

{

int two = 8, num=0;

for (int j = 0; j < 4; j++)

{

num += two \* ((int)bin[i + j] - 48);

two /= 2;

}

hex[k++]=hex\_char\_return(num);

i += 4;

}

}

}

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

/\* func\_name : hex\_char\_return \*/

/\* purpose : 입력 받은 정수에 대한 16진수 char 값을 return \*/

/\* return : none \*/

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

char hex\_char\_return(int num)

{

switch (num)

{

case 0:

return '0';

break;

case 1:

return '1';

break;

case 2:

return '2';

break;

case 3:

return '3';

break;

case 4:

return '4';

break;

case 5:

return '5';

break;

case 6:

return '6';

break;

case 7:

return '7';

break;

case 8:

return '8';

break;

case 9:

return '9';

break;

case 10:

return 'A';

break;

case 11:

return 'B';

break;

case 12:

return 'C';

break;

case 13:

return 'D';

break;

case 14:

return 'E';

break;

case 15:

return 'F';

break;

}

}

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

/\* func\_name : disp\_addr\_set \*/

/\* purpose : disp 또는 addr을 16진수로 바꾸어 result object code에 저장\*/

/\* return : symbol's LOCCTR \*/

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

void disp\_addr\_set(char \*arr, int num, int format\_type)

{

int i = 3, quotient = 0; // arr의 index, 16진수로 나눈 몫을 저장하는 변수

int hexx = 0; //16의 제곱들을 저장하는 변수

//disp가 음수인 경우, 2's complement

if (num < 0)

{

//num에 -1을 곱한다.

num \*= -1, hexx = 256;

char temp\_num[13] = { 0 };

//-1곱한 것을 2진수로 바꾸어 저장.

for (int j = 0; j < 12; )

{

quotient = num / hexx;

num = num - (quotient \* hexx);

switch (hex\_char\_return(quotient)) {

case '0':

temp\_num[j + 0] = '0'; temp\_num[j + 1] = '0'; temp\_num[j + 2] = '0'; temp\_num[j + 3] = '0';

break;

case '1':

temp\_num[j + 0] = '0'; temp\_num[j + 1] = '0'; temp\_num[j + 2] = '0'; temp\_num[j + 3] = '1';

break;

case '2':

temp\_num[j + 0] = '0'; temp\_num[j + 1] = '0'; temp\_num[j + 2] = '1'; temp\_num[j + 3] = '0';

break;

case '3':

temp\_num[j + 0] = '0'; temp\_num[j + 1] = '0'; temp\_num[j + 2] = '1'; temp\_num[j + 3] = '1';

break;

case '4':

temp\_num[j + 0] = '0'; temp\_num[j + 1] = '1'; temp\_num[j + 2] = '0'; temp\_num[j + 3] = '0';

break;

case '5':

temp\_num[j + 0] = '0'; temp\_num[j + 1] = '1'; temp\_num[j + 2] = '0'; temp\_num[j + 3] = '1';

break;

case '6':

temp\_num[j + 0] = '0'; temp\_num[j + 1] = '1'; temp\_num[j + 2] = '1'; temp\_num[j + 3] = '0';

break;

case '7':

temp\_num[j + 0] = '0'; temp\_num[j + 1] = '1'; temp\_num[j + 2] = '1'; temp\_num[j + 3] = '1';

break;

case '8':

temp\_num[j + 0] = '1'; temp\_num[j + 1] = '0'; temp\_num[j + 2] = '0'; temp\_num[j + 3] = '0';

break;

case '9':

temp\_num[j + 0] = '1'; temp\_num[j + 1] = '0'; temp\_num[j + 2] = '0'; temp\_num[j + 3] = '1';

break;

case 'A':

temp\_num[j + 0] = '1'; temp\_num[j + 1] = '0'; temp\_num[j + 2] = '1'; temp\_num[j + 3] = '0';

break;

case 'B':

temp\_num[j + 0] = '1'; temp\_num[j + 1] = '0'; temp\_num[j + 2] = '1'; temp\_num[j + 3] = '1';

break;

case 'C':

temp\_num[j + 0] = '1'; temp\_num[j + 1] = '1'; temp\_num[j + 2] = '0'; temp\_num[j + 3] = '0';

break;

case 'D':

temp\_num[j + 0] = '1'; temp\_num[j + 1] = '1'; temp\_num[j + 2] = '0'; temp\_num[j + 3] = '1';

break;

case 'E':

temp\_num[j + 0] = '1'; temp\_num[j + 1] = '1'; temp\_num[j + 2] = '1'; temp\_num[j + 3] = '0';

break;

case 'F':

temp\_num[j + 0] = '1'; temp\_num[j + 1] = '1'; temp\_num[j + 2] = '1'; temp\_num[j + 3] = '1';

break;

}

j += 4;

hexx /= 16;

}

//모든 2진수 값 XOR 연산

for (int j = 0; j < 12; j++)

{

if (temp\_num[j] == '0')

temp\_num[j] = '1';

else

temp\_num[j] = '0';

}

//마지막에 1을 더한다.

int temp\_index = 11;

while (temp\_index >= 0)

{

//temp\_num[temp\_index]가 0이면, 1로 바꾸어줌

if (temp\_num[temp\_index] == '0')

{

temp\_num[temp\_index] = '1';

break;

}

//temp\_num[temp\_index]가 1이면, carry 계산을 해준다.

else

temp\_num[temp\_index] = '0';

temp\_index--;

}

//temp\_number을 16진수로 바꾸어 temp\_hex\_arra에 저장

char temp\_hex[3] = { 0 };

change\_hex(temp\_num, temp\_hex, 3);

//16진수를 arr에 저장.

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

arr[i++] = temp\_hex[j];

}

//format 4, address 16진수로 변환

else if (num > 4095 || format\_type == 4)

{

hexx = 65536;

for (int j = 0; j < 5; j++) {

quotient = num / hexx;

num = num - (quotient \* hexx);

arr[i++] = hex\_char\_return(quotient);

hexx /= 16;

}

}

//format 3, disp 16진수로 변환

else

{

hexx = 256;

for (int j = 0; j < 3; j++) {

quotient = num / hexx;

num = num - (quotient \* hexx);

arr[i++] = hex\_char\_return(quotient);

hexx /= 16;

}

}

}

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

/\* func\_name : register\_num \*/

/\* purpose : register character를 입력 받아 해당 번호 저장\*/

/\* return : none \*/

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

void register\_num(char \*arr, char\* reg)

{

if (!strcmp(reg, "A"))

strcat(arr, "0000");

else if (!strcmp(reg, "X"))

strcat(arr, "0001");

else if (!strcmp(reg, "L"))

strcat(arr, "0010");

else if (!strcmp(reg, "PC"))

strcat(arr, "1000");

else if (!strcmp(reg, "SW"))

strcat(arr, "1001");

else if (!strcmp(reg, "B"))

strcat(arr, "0101");

else if (!strcmp(reg, "S"))

strcat(arr, "0100");

else if (!strcmp(reg, "T"))

strcat(arr, "0101");

else if(!strcmp(reg, "F"))

strcat(arr, "0110");

else

{

//이외의 값이 들어온 경우. 에러 메시지 출력.

printf("Assemble Error on line %d : No such register!\n", lst\_line\_num);

error\_flag;

return;

}

}

* 1. 20161498.c

memset(file\_name, '\0', sizeof(file\_name)); //file\_name 초기화 추가.

이외에 프로젝트 1과 차이 없음.

int main()

{

while(!quit\_flag)

{

command\_distinct(command);

memset(file\_name, '\0', sizeof(file\_name)); //file\_name 초기화 추가.

}

}