**과목명: 시스템프로그래밍**

**분반2**

**<<Project #1>>**

**서강대학교 [학부명]**

**20161580[학번]**

**김현규[이름]**

목 차

1. **프로그램 개요**
2. **프로그램 설명**
   1. 프로그램 흐름도
3. **모듈 정의**
   1. 정의한 모듈에 대해 간략적인 설명
4. **전역 변수 정의**
5. **코드 설명**
6. 프로그램 개요

|  |
| --- |
| 프로젝트1에서 구현한 셀(shell)에 assemble 기능을 추가하는 프로그램. SIC/XE의 assembly program source 파일을 입력 받아서 object파일을 생성하고, 어셈블리 과정 중 생성된 symbol table과 결과물인 object 파일을 볼 수 있는 기능을 제공해야 함. 교재의 2.2까지 설명된 SIC/XE 어셈블러의 기능을 구현함을 원칙으로 한다. |

1. 프로그램 설명

|  |
| --- |
| 프로젝트1에서 구현한 셀(shell)에 assemble 기능을 추가하는 프로그램을 작성하는 프로젝트 로, SIC/XE machine의 assembly program source 파일을 입력 받아서 object파일을 생성하고, 어셈 블리 과정 중 생성된 symbol table과 결과물인 object 파일을 볼 수 있다.  -구현해야 할 사항들  1)Shell 관련 명령어: help, type  ① help: Shell에서 실행 가능한 모든 명령어들의 리스트를 화면에 출력해준다.  ② type filename: filename에 해당하는 파일을 현재 디렉터리에서 읽어서 화면에 출력한다. 현재 디렉터리에 해당 파일이 존재하지 않으면 에러 메시지를 출력한다. filename이 디렉토리인 경우는 고려하지 않는다. 시스템 콜을 사용하지 않는다.  2)SIC/XE 어셈블러 명령  ① assemble filename:  -filename에 해당하는 소스 파일을 읽어서 object파일과 리스팅 파일을 만든다. 소스 파일의 확장자는 .asm이다. 리스팅 파일의 파일명은 소스파일과 동일하고 확장자는 .lst이다. object 파일의 파일명은 소스 파일과 동일하고 확장자는 .obj이다. (lst 파일과 .obj 파일은 출력형식을 반드시 지킬 것)  -소스파일에 에러가 존재할 경우, 리스팅 파일과 object파일을 생성하지 않고 에러 내용을 화면에 출력한다. 에러 발생시 바로 명령을 종료한다. 에러의 내용은 디버깅을 위해 어떤 라인에서 에러가 발생했는지 출력한다. 라인번호는 5의 배수 단위로 지정하여 출력한다.  -각 문자열의 길이는 30자 이내로 가정한다. label은 영문, 숫자를 input으로 가정하며, 그 외 특수문자는 고려하지 않는다.  ② symbol assemble: 과정 중에 생성된 symbol table을 화면에 출력한다. |

2-1. 프로그램 흐름도

|  |
| --- |
| Assemble  -pass1    -pass2 |

1. 모듈 정의
   1. 모듈 이름: void type(char\* filename)
      1. 기능

-filename의 이름을 가진 파일을 한 문장씩 읽어서 buffer에 저장한다. 그리고 buffer을 stdout에 출력한다.

-파일의 끝에 도달할 때까지 위 과정을 반복한다.

* 1. 모듈 이름: void assemble(char\* filename)
     1. 기능

-symbol의 위치(loc)와 이름(label)을 저장하는 symtab을 초기화한다.

-\*.obj와 \*.lst 파일포인터를 초기화한다. 각각 op1, op2이다.

-Chap2에 있는 pass1과 pass2를 구현한다.

-assemble할 code의 시작 주소를 찾는다. 시작 주소는 starting\_address에 저장한다.

-pass1:

1 한 문장을 buffer에 읽어들인다. Buffer의 token\_count(조각의 개수)을 계산한다.

2 token\_count에 따라, Buffer의 정보를 읽어들인다. 새로운 LABEL을 발견했을 때 이를 symtab에 저장한다. 이 때 이 LABEL이 이미 symtab에 저장되어 있을 경우 ERROR 발생

3 buffer의 opcode에 따라 locctr(locate counter)을 더해준다. 현재 line의 locctr은 line\_locctr[line\_number]에 저장한다.

4 END가 나올 때까지 1~3 반복

5 Header Record를 \*.obj에 출력

-pass2:

1 한 문장을 buffer에 읽어들인다. Buffer의 token\_count를 계산한다.

2 line의 번호, 주소, buffer을 \*.lst 파일에 출력한다. buffer의 특징에 따라 처리해주는 작업이 다르다. 특징에 따라서 적절한 작업을 취해준다.

특징의 종류는 다음과 같다.

2-1. format이 2? 3? 4? (buffer의 opcode에 따라서 구분할 수 있다.)

2-2. address를 저장하는 법이 Program Counter relative? Base Relative?

2-3. operand에 register가 있는가? 없는가?

2-4. 데이터를 저장하는 작업(BYTE, RESB, RESW)인가?

경우에 따라 적절하게 object code를 만든다. object code는 \*.lst에 출력

나올 수 있는 ERROR 1: 상대 주소를 지정하는 과정에서 operand의 주소를 return받기 위해 symtab에 접근했는데 symtab에는 해당 operand가 존재하지 않을 경우

ERROR 2: buffer에서 사용한 opcode가 opcode\_table에 존재하지 않을 경우

3 Text Record를 \*.obj에 출력한다. 출력해야하는 상황은 기존의 Text Record string에서 계산한 objectcode를 채웠을 때 기준 길이를 넘어서는 경우, 또는 RESB, RESW 등 데이터 할당 과정 이후이다.

4 END가 나올 때까지 1~2 반복

지금까지 저장해둔 Text Record \*.obj에 출력. End Record \*.obj에 출력.

* 1. 모듈 이름: void AddSymtab(int loc, char\* name)
     1. 기능

-symtab에 새로운 label의 주소(loc)와 이름(name)을 저장한다.

* 1. 모듈 이름: int FindSymtab(char\* name)
     1. 기능

-symtab에 존재하는 name의 주소를 return한다. Symtab에 존재하지 않을 경우 NotInSymtab을 return한다.

* 1. 모듈 이름: char\* itoa(int num, char\* str, int base)
     1. 기능

-10진수인 num을 base진수로 바꾼 후 이 값을 char\* str에 저장한다.

-ex)char\* itoa(19, str, 16) -> 19(10) = 13(X) -> str = {“13”}

* 1. 모듈 이름: int regis\_to\_data(char c)
     1. 기능

-입력받은 c는 register이다. register에 해당하는 값을 return한다.

* 1. 모듈 이름: void symbol(void)
     1. 기능

-symtab에 존재하는 LABEL의 이름과 주소를 출력한다.

-이 때 알파벳 순서대로 LABEL을 정렬한 후 출력한다.

* 1. 모듈 이름: int opcode\_format(char\* Mnemonic)
     1. 기능

-입력받은 opcode(char\* Mnemonic)에 해당하는 format을 return한다.

4 전역 변수 정의

4.1 이름: SYMTAB\* symtab

4.1.1 기능: assemble한 code의 symbol의 이름과 주소를 저장한다.

4.2 이름: int symtabEXIST

4.2.1 기능: symtabEXIST가 1일 경우 이전에 assemble하여 symtab이 존재하고 있음을 나타낸다. 즉 이어 또 다른 code를 assemble할 경우 symtab을 초기화할 수 있도록 하는 flag 역할을 한다.

1. 코드 설명

-설명은 주석으로 상세히 기입하였다.

void symbol(void)

{

SYMTAB\* tempsymtab = (SYMTAB\*)malloc(sizeof(SYMTAB));

tempsymtab = symtab->next;

char str1[100][100]; int loc[100];

int i=0, j; char temp[100]; int temp\_loc;

//symtab의 정보를 str1(LABEL의 이름), loc(위치)에 옮긴다.

while(tempsymtab != NULL){

strcpy(str1[i], tempsymtab->name);

loc[i++] = tempsymtab->loc;

tempsymtab = tempsymtab->next;

}

int count = i;

//알파벳 순으로 정렬

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

for(j=i+1;j<count;j++){

if(strcmp(str1[i],str1[j])>0){

strcpy(temp,str1[i]);

strcpy(str1[i],str1[j]);

strcpy(str1[j],temp);

temp\_loc = loc[i];

loc[i] = loc[j];

loc[j] = temp\_loc;

}

}

for(i=0; i < count; i++){

printf(" %-10s %04X\n", str1[i], loc[i]);

}

}

void type(char\* filename)

{

filename[strlen(filename)] = '\0';

FILE\* ip;

//현재 디렉토리에 filename의 이름을 가진 file이 존재하지 않을 경우

if(!(ip = fopen(filename, "r"))){

fputs("file open error\n", stderr);

}

//존재할 경우

else{

char buffer[100];

while(!feof(ip)){

if(fgets(buffer, sizeof(buffer), ip) == NULL) break;

else printf("%s", buffer);

}

fclose(ip);

}

}

void assemble(char\* filename)

{

//이전에 이미 assemble한 파일이 있을 경우 symtab이 존재한다.

//따라서 free해준다.

if(symtabEXIST) free(symtab);

symtab = malloc(sizeof(SYMTAB));

symtab->next = NULL;

//파일명 생성

//op1\_filename: filename.lst

//op2\_filename: filename.obj

char temp\_filename[100]; char op1\_filename[100]; char op2\_filename[100];

char op1\_str[10] = {"lst"}; char op2\_str[10] = {"obj"};

strcpy(temp\_filename, filename);

temp\_filename[strlen(temp\_filename) - 3] = '\0';

strcpy(op1\_filename, temp\_filename);

strcpy(op2\_filename, temp\_filename);

strcat(op1\_filename, op1\_str);

strcat(op2\_filename, op2\_str);

op1\_filename[strlen(op1\_filename)] = '\0';

op2\_filename[strlen(op1\_filename)] = '\0';

//ip: \*.asm을 다루는 file pointer

FILE\* ip;

if(!(ip = fopen(filename, "r"))){

fputs("file open error\n", stderr);

return 0;

}

//op1: \*.lst를 다루는 file pointer

FILE\* op1 = fopen(op1\_filename, "w");

//op2: \*.obj를 다루는 file pointer

FILE\* op2 = fopen(op2\_filename, "w");

//string(pass 과정에서 읽히는 한 문장)을 다루는 parameter

char par1[100]; char par2[100]; char par3[100];

char temp\_LABEL[100]; char temp\_OPCODE[100]; char temp\_OPERAND[100];

char temp\_OPERAND1[100]; char temp\_OPERAND2[100];

//locctr 다루는 변수

int starting\_address = 1; int starting\_address\_flag = 0;

int line\_number; int locctr;

//string(pass 과정에서 읽는 한 문장)을 저장하는 공간

char buffer[100]; char temp\_buffer[100];

//각 line의 locctr

int line\_locctr[1000];

//임시변수들

char temp[100];

//Header Record 에 쓰이는 변수들

char program\_name[6];

char starting\_address\_str[6];

char program\_size[6];

//Text Record에 쓰이는 변수들

char ObjectCodeStr[61];

ObjectCodeStr[0] = '\0';

int Start\_str;

//Modified Record에 쓰이는 변수들

char Modified\_Start[100][100];

char Modified\_Size[100][100];

int m=0;

//pass1

//read first input line

fgets(buffer, sizeof(buffer), ip);

char \*ptr = strtok(buffer, " ");

//program의 정보 저장. Header Record에 쓰인다.

strcpy(program\_name, ptr);

//assemble하는 도중 ERROR가 발생할 경우 FLAG = 1

int ERROR\_FLAG = 0;

//Header Record의 프로그램 이름 저장

int name\_size = strlen(program\_name);

program\_name[6] = '\0';

for(int i = name\_size; i<6; ++i) program\_name[i] = ' ';

//program의 시작 주소 찾기

while (ptr != NULL) {

strcpy(temp, ptr);

//첫 줄에 시작 주소가 지정되어 있을 경우

if (starting\_address\_flag) starting\_address = strtol(temp, NULL, 16);

if (!strcmp("START", temp)) {

starting\_address\_flag = 1;

}

ptr = strtok(NULL, " ");

}

//첫 줄에 시작 주소가 지정되어 있지 않을 경우

if (!starting\_address\_flag) starting\_address = 0;

//pass1에 쓰일 변수들 초기화

line\_locctr[0] = starting\_address;

Start\_str = starting\_address;

locctr = starting\_address;

line\_number = 1;

//program의 시작 주소 저장

sprintf(starting\_address\_str, "%06x", starting\_address);

//한 줄의 parameter의 개수

int token\_count = 0;

//base의 주소 저장하는 string

char base\_address\_key[10];

//pass1

while (1)

{

int flag = 0;

fgets(buffer, sizeof(buffer), ip);

strcpy(temp\_buffer, buffer);

//공백을 기준으로 문장을 나눈다.

line\_locctr[line\_number++] = locctr;

char \*temp\_ptr = strtok(temp\_buffer, " ");

//token 하나하나에 접근한다.

while (temp\_ptr != NULL) {

temp\_ptr = strtok(NULL, " ");

token\_count++;

}

//token\_count가 n개일 때: n 덩어리

char \*ptr = strtok(buffer, " ");

if (token\_count == 3) {

strcpy(par1, ptr);

ptr = strtok(NULL, " ");

strcpy(par2, ptr);

ptr = strtok(NULL, " ");

strcpy(par3, ptr);

//두 번째 parameter의 마지막에 ','이 들어갈 경우

//ex) STCH BUFFER, X

if(par2[strlen(par2) - 1] == ','){

strcpy(temp\_OPCODE, par1);

strcpy(temp\_OPERAND1, par2);

strcpy(temp\_OPERAND2, par3);

}

else{

strcpy(temp\_LABEL, par1);

strcpy(temp\_OPCODE, par2);

strcpy(temp\_OPERAND, par3);

//Label일 경우 Symtab에 저장

//Symtab에 이미 해당 Label의 정보가 저장되어 있을 경우

if(FindSymtab(temp\_LABEL) != NotInSymtab){

printf("Line %04d: ERROR - 변수명이 중복으로 사용되었음.\n", line\_number\*5);

ERROR\_FLAG = 1;

break;

}

//저장되어 있지 않을 경우 Label의 정보를 새로 추가해준다

else AddSymtab(locctr, temp\_LABEL);

}

}

else if (token\_count == 2) {

strcpy(temp\_OPCODE, ptr);

ptr = strtok(NULL, " ");

strcpy(temp\_OPERAND, ptr);

}

else {

//ex) . SUBROUTINE TO ...

if (token\_count > 3) flag = 1;

else if (token\_count == 1) {

strcpy(temp\_OPERAND, ptr);

//ex) .

if (!strcmp(".\n", temp\_OPERAND)) flag = 1;

else {

//RSUB

}

}

}

//OPCODE에 따라 적절하게 locctr을 update한다.

if (!strcmp("WORD", temp\_OPCODE))

{

locctr += 3;

}

else if (!strcmp("RESW", temp\_OPCODE))

{

//-1하는 이유는 '\0'을 고려하기 때문이다

locctr += 3 \* (strlen(temp\_OPERAND) - 1);

}

else if (!strcmp("RESB", temp\_OPCODE))

{

locctr += strtol(temp\_OPERAND, NULL, 10);

}

else if (!strcmp("BYTE", temp\_OPCODE))

{

//-1하는 이유는 '\0'을 고려하기 때문이다

if(temp\_OPERAND[0] == 'C') locctr += (strlen(temp\_OPERAND) - 4);

else if(temp\_OPERAND[0] == 'X') locctr += (strlen(temp\_OPERAND) - 4) / 2;

}

else if (!strcmp("BASE", temp\_OPCODE))

{

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

strcpy(base\_address\_key, temp\_OPERAND);

}

//flag = 1인 상황에는 locctr을 더해줄 필요가 없다.

else if(flag == 1){

flag = 0;

}

else

{

if (opcode\_format(temp\_OPCODE) == 0){

if(temp\_OPCODE[0] == '+') locctr += 4;

}

else if (opcode\_format(temp\_OPCODE) == 1){locctr += 1;}

else if (opcode\_format(temp\_OPCODE) == 2){locctr += 2;}

else if (opcode\_format(temp\_OPCODE) == 3){locctr += 3;}

}

//OPCODE에 END가 입력될 경우 PASS1 종료

if(!strcmp("END", temp\_OPCODE)){

sprintf(program\_size, "%06X", locctr);

break;

}

//다음 buffer을 위하여 token\_count는 0으로 초기화

token\_count = 0;

}

fclose(ip);

//symtab을 만들었을 경우 symtabEXIST = 1

symtabEXIST = 1;

//Header Record: 프로그램 이름 program\_name (3byte) + 프로그램의 시작주소starting\_address\_str (6byte) + 프로그램의 길이program\_size (6byte)

fprintf(op2, "H%s%s%s\n", program\_name, starting\_address\_str, program\_size);

FILE\* ip2 = fopen(filename, "r");

//string의 parameter

//pass2에서 현재 다루고 있는 line

int presentline = 1;

//Base Relative로 상대 주소를 지정할 때 base의 주소

int base\_address = FindSymtab(base\_address\_key);

//.lst파일 작성

fgets(buffer, sizeof(buffer), ip2);

fprintf(op1, "%-4d %04X %s", 5, line\_locctr[0], buffer);

//Text Record 저장 관련 flag

int FindStart\_str = 0;

//입력된 TEMP\_OPCODE가 RESB나 RESW인 경우 FLAG = 1

int OPCODE\_RESB\_RESW\_FLAG = 0;

//pass2

while(1){

//pass1에서 ERROR가 발생했을 경우

if(ERROR\_FLAG) break;

int token\_count = 0;

//pass2에서 다룰 필요가 없는 문장을 입력했을 경우 해당 flag = 1

int toNextFlag = 0;

//X Register을 사용하고 Base Realtive일 경우flag = 1

int BaseRelativeFlagwithX = 0;

//Base Relative일 경우 flag = 1

int BaseRelativeFlag = 0;

//operand가 없을 경우 flag = 1

int NoOperandFlag = 0;

fgets(buffer, sizeof(buffer), ip2);

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

//.lst 작성

//line이 넘겨져도 locctr이 증가하지 않는 경우

if(line\_locctr[presentline] == line\_locctr[presentline + 1]) fprintf(op1, "%-4d %s", (presentline+1) \* 5, buffer);

else{

if(presentline != line\_number-1) fprintf(op1, "%-4d %04X %-30s", (presentline+1) \* 5, line\_locctr[presentline], buffer);

}

//line\_number는 pass1의 끝의 locctr, 즉 .asm의 마지막 loine의 locctr을 가지고 있다.

if(presentline == line\_number-1) fprintf(op1, "%-4d %s\n", (presentline+1)\*5, buffer);

strcpy(temp\_buffer, buffer);

//공백을 기준으로 문장을 나눈다.

char \*temp\_ptr = strtok(temp\_buffer, " ");

//단어 하나하나에 접근한다.

while (temp\_ptr != NULL) {

temp\_ptr = strtok(NULL, " ");

token\_count++;

}

//token\_count가 n개일 때: n 덩어리

char \*ptr = strtok(buffer, " ");

if (token\_count == 3) {

int flag = 0;

strcpy(par1, ptr);

ptr = strtok(NULL, " ");

strcpy(par2, ptr);

ptr = strtok(NULL, " ");

strcpy(par3, ptr);

//두 번째 parameter의 마지막에 ','이 들어갈 경우

//ex) STCH BUFFER, X

if(par2[strlen(par2) - 1] == ','){

strcpy(temp\_OPCODE, par1);

strcpy(temp\_OPERAND1, par2);

temp\_OPERAND1[strlen(temp\_OPERAND1) - 1] = '\0';

strcpy(temp\_OPERAND2, par3);

if(temp\_OPERAND1[0] == 'X' || temp\_OPERAND2[0] == 'X'){

BaseRelativeFlagwithX= 1;

}

}

else{

strcpy(temp\_LABEL, par1);

strcpy(temp\_OPCODE, par2);

strcpy(temp\_OPERAND, par3);

flag = 1;

}

if(!strcmp(temp\_OPCODE, "RESW") || !strcmp(temp\_OPCODE, "RESB") || !strcmp(temp\_OPCODE, "BASE")) toNextFlag = 1;

}

else if (token\_count == 2) {

strcpy(temp\_OPCODE, ptr);

ptr = strtok(NULL, " ");

strcpy(temp\_OPERAND, ptr);

}

else {

if (token\_count > 3) {toNextFlag = 1;}

else if (token\_count == 1) {

strcpy(temp\_OPCODE, ptr);

if (!strcmp(".", temp\_OPCODE)) { toNextFlag = 1;}

else {

//RSUB

NoOperandFlag = 1;

}

}

}

if(!strcmp(temp\_OPCODE, "RESW") || !strcmp(temp\_OPCODE, "RESB") || !strcmp(temp\_OPCODE, "BASE"))

{toNextFlag = 1;

}

//OPCODE에 END가 입력될 경우 PASS2 종료

if (!strcmp(temp\_OPCODE, "END")) {

//Text Record 출력

fprintf(op2,"T%06X%02X%s\n", Start\_str, (int)strlen(ObjectCodeStr)/2, ObjectCodeStr);

OPCODE\_RESB\_RESW\_FLAG = 1;

//Modified Record 출력

for(int M=0; M<m; ++M){

fprintf(op2, "M%s%s\n", Modified\_Start[M], Modified\_Size[M]);

}

//End Record 출력

fprintf(op2, "E%06X\n", line\_locctr[1]);

break;

}

//objectcode를 생성할 필요가 없는 line일 경우

if(toNextFlag) {

fprintf(op1, "\n");

presentline++;

//Text Record 출력

if((!strcmp(temp\_OPCODE, "RESB") || !strcmp(temp\_OPCODE, "RESW")) && OPCODE\_RESB\_RESW\_FLAG == 0){

fprintf(op2, "T%06X%02X%s\n", Start\_str, (int)strlen(ObjectCodeStr)/2, ObjectCodeStr);

memset(ObjectCodeStr, 0, sizeof(ObjectCodeStr));

FindStart\_str = 1;

OPCODE\_RESB\_RESW\_FLAG = 1;

}

continue;

}

//flag 초기화

OPCODE\_RESB\_RESW\_FLAG = 0;

char objectcode1\_B[10]; //opcode(2진수)

char objectcode2\_B[10]; //nixbpe(2진수)

//objectcode1\_B와 objectcode2\_B가 합해진 string(Binary)

char objectcode12\_B[20];

//objectcode12\_B가 16진수로 변환된 것

char objectcode12\_X[20];

char objectcode3\_B[12]; //displayment or address(2진수)

//objectcode1,2,3이 합해진 string

char objectcode123\_B[50];

//objectcode123\_B가 16진수로 변환된 것

char objectcode123\_X[20];

int opcode\_data;

//Findstart\_str: RESB || RESW 이후 TextRecord의 Starting Address를 찾는 중

if(FindStart\_str){

Start\_str = line\_locctr[presentline];

FindStart\_str = 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*objectcode1\_B 입력 과정\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//objectcode1\_B: opcode 정보(6bit)

if(temp\_OPCODE[0] == '+') opcode\_data = opcode(temp\_OPCODE + 1);

else opcode\_data = opcode(temp\_OPCODE); //14(X)

//ERROR: 정의되지 않은 instruction이 사용되었을 경우

if(opcode\_data == ERROR\_WrongOpcode && !(!strcmp("BYTE", temp\_OPCODE)) && !(!strcmp("RESW", temp\_OPCODE)) && !(!strcmp("RESB", temp\_OPCODE))){

printf("Line %04d: ERROR - 정의되지 않은 instruction이 사용되었음.\n", (presentline + 1)\*5);

ERROR\_FLAG = 1;

break;

}

//ex)opcode\_data = 20일 경우 objectcode1\_B = {"10100"}

itoa(opcode\_data, objectcode1\_B, 2);

//objectcode의 형식에 맞게 0을 채워준다.

//ex)10100 -> 000 10100

int Z = 8 - strlen(objectcode1\_B);

char Zero[13][20] = { "", "0", "00", "000","0000","00000","000000","0000000", "00000000", "000000000", "0000000000", "00000000000", "000000000000"};

char zero[13][20] = { "", "0", "00", "000","0000","00000","000000","0000000", "00000000", "000000000", "0000000000", "00000000000", "000000000000" };

strcat(zero[Z], objectcode1\_B);

strcpy(objectcode1\_B, zero[Z]);

int format = opcode\_format(temp\_OPCODE);

//opcode의 정보 8bit에서 마지막 2bit는 삭제한다.

if(format != 2) objectcode1\_B[strlen(objectcode1\_B) - 2] = '\0';

strcpy(zero[Z], Zero[Z]);

strcpy(objectcode12\_B, objectcode1\_B);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*objectcode2\_B 입력 과정\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//opcode format이 2일 때 objectcode2\_B: Register의 정보

//opcode format이 3,4일 때 objectcode2\_B: n, i, x, b, p, e(6bit)

if(format == 1){

presentline++;

sprintf(objectcode1\_X, "%X", (int)strtol(objectcode1\_B, NULL, 2));

fprintf(op1, "%s\n", objectcode1\_X);

}

//opcode가 다루는 format이 2일 경우

if(format == 2){

presentline++;

//ex)CLEAR X

if(!strcmp("CLEAR", temp\_OPCODE) || !strcmp("TIXR", temp\_OPCODE)){

//register의 정보

int data = regis\_to\_data(temp\_OPERAND[0]);

itoa(data, objectcode2\_B, 2);

Z = 4 - strlen(objectcode2\_B);

strcat(zero[Z], objectcode2\_B);

strcpy(objectcode2\_B, zero[Z]);

strcpy(zero[Z], Zero[Z]);

strcat(objectcode2\_B, zero[4]);

}

//ex)COMPR A,S

else{

int data1 = regis\_to\_data(temp\_OPERAND1[0]);

int data2 = regis\_to\_data(temp\_OPERAND2[0]);

//첫번째 register의 정보

itoa(data1, objectcode2\_B, 2);

Z = 4 - strlen(objectcode2\_B);

strcat(zero[Z], objectcode2\_B);

strcpy(objectcode2\_B, zero[Z]);

strcpy(zero[Z], Zero[Z]);

//두번째 register의 정보

char objectcode2\_B2[10];

itoa(data2, objectcode2\_B2, 2);

Z = 4 - strlen(objectcode2\_B2);

strcat(zero[Z], objectcode2\_B2);

strcpy(objectcode2\_B2, zero[Z]);

strcpy(zero[Z], Zero[Z]);

strcat(objectcode2\_B, objectcode2\_B2);

}

strcat(objectcode12\_B, objectcode2\_B);

sprintf(objectcode12\_X, "%X", (int)strtol(objectcode12\_B, NULL, 2));

//Text Record 작성

fprintf(op1, "%s\n", objectcode12\_X);

}

//opcode format이 3,4일 경우

else{

int n=0, i=1, x=2, b=3, p=4, e=5;

//Operand의 앞에 #가 불을 경우(Immediate Addressing)

if(temp\_OPERAND[0] == '#'){objectcode2\_B[n] = '0'; objectcode2\_B[i] = '1';}

//Operand의 앞에 @가 붙을 경우(Indirect Addressing)

else if(temp\_OPERAND[0] == '@'){objectcode2\_B[n] = '1'; objectcode2\_B[i] = '0';}

//Simple Addressing

else {objectcode2\_B[n] = '1'; objectcode2\_B[i] = '1';}

//X register을 이용하고, Base Relative로 상대 주소를 지정할 경우

if(BaseRelativeFlagwithX){

objectcode2\_B[x] = '1';

objectcode2\_B[b] = '1';

objectcode2\_B[p] = '0';

}

//Program Counter로 상대 주소를 지정할 경우

else{

objectcode2\_B[x] = '0';

objectcode2\_B[b] = '0';

objectcode2\_B[p] = '1';

}

//opcode format이 4일 경우

if(temp\_OPCODE[0] == '+'){objectcode2\_B[e] = '1'; objectcode2\_B[p] = '0';}

else objectcode2\_B[e] = '0';

//RSUB

if(NoOperandFlag) {

objectcode2\_B[n] = '1'; objectcode2\_B[i] = '1'; objectcode2\_B[x] = '0'; objectcode2\_B[b] = '0'; objectcode2\_B[p] = '0'; objectcode2\_B[e] = '0';

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*objectcode3\_B 입력 과정\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//objectcode3\_B: 주소를 저장(format 3: 12bit format 4: 20bit)

//presentline의 다음 line의 Program Counter

int PC = line\_locctr[presentline++ + 1];

//operand의 주소 저장하는 변수

int operand\_address;

//immediate addressing일 경우 1

int immediate = 0;

//Simple Addressing 일 경우

if(objectcode2\_B[n] == '1' && objectcode2\_B[i] == '1'){

operand\_address = FindSymtab(temp\_OPERAND);

//정의되지 않은 변수명이 사용되었을 경우

if(!(!strcmp(temp\_OPCODE, "BYTE"))){

if(FindSymtab(temp\_OPERAND) == NotInSymtab){

printf("Line %d: ERROR - 정의되지 않은 변수명이 사용되었음. %s\n", presentline\*5, temp\_OPERAND);

ERROR\_FLAG = 1;

break;

}

}

}

else {

//Immediate Addressing일 경우

if(temp\_OPERAND[0] == '#' && FindSymtab(temp\_OPERAND + 1) == NotInSymtab) {operand\_address = strtol(temp\_OPERAND+1, NULL, 10); objectcode2\_B[p] = '0'; immediate = 1;}

else {operand\_address = FindSymtab(temp\_OPERAND + 1);

//정의되지 않은 변수명이 사용되었을 경우

if(FindSymtab(temp\_OPERAND + 1) == NotInSymtab){

printf("Line %d: ERROR - 정의되지 않은 변수명이 사용되었음.\n", (presentline - 1) \* 5);

ERROR\_FLAG = 1;

break;

}

}

}

//X register을 사용하고 Base Relative로 주소를 지정하는 경우

if(BaseRelativeFlagwithX) {

if(temp\_OPERAND2[0] == 'X')

operand\_address = FindSymtab(temp\_OPERAND1);

else

operand\_address = FindSymtab(temp\_OPERAND2);

//정의되지 않은 변수명이 사용되었을 경우

if(FindSymtab(temp\_OPERAND1) == NotInSymtab){

printf("Line %d: ERROR - 정의되지 않은 변수명이 사용되었음.\n", (presentline - 1) \* 5);

ERROR\_FLAG = 1;

break;

}

}

//상대주소 저장

int address;

//Base Relative로 주소를 저장하는 경우(이 때는 X register를 사용하는 경우)

if(BaseRelativeFlagwithX) address = operand\_address - base\_address;

//operand가 없을 경우 ex)RSUB

else if(NoOperandFlag) address = 0;

//Program Counter Relative로 주소를 저장하는 경우

else address = operand\_address - PC;

//Base Relative로 주소를 저장하는 경우

if((address < -2048 || address > 2047) && objectcode2\_B[e] == '0')

{

BaseRelativeFlag = 1;

//Base Relative에 맞게 objectcode2\_B 변환

objectcode2\_B[x] = '0'; objectcode2\_B[b] = '1'; objectcode2\_B[p] = '0';

address = operand\_address - base\_address;

}

objectcode2\_B[6] = '\0';

strcat(objectcode12\_B, objectcode2\_B);

strcpy(objectcode123\_B, objectcode12\_B);

if(objectcode2\_B[e] == '1'){

//CASE: immediate addressing && format 4

if(immediate){

address = operand\_address;

itoa(address, objectcode3\_B, 2);

Z = 20 - strlen(objectcode3\_B);

strcat(zero[Z], objectcode3\_B);

strcpy(objectcode3\_B, zero[Z]);

strcpy(zero[Z], Zero[Z]);

}

//CASE: format 4

else{

address = operand\_address;

itoa(address, objectcode3\_B, 2);

//Modified Record 생성

int M\_start = line\_locctr[presentline - 1] + (strlen(objectcode12\_B)+ 1) / 8;

Z = 20 - strlen(objectcode3\_B);

strcat(zero[Z], objectcode3\_B);

strcpy(objectcode3\_B, zero[Z]);

strcpy(zero[Z], Zero[Z]);

//Modified Record 저장

int M\_size = strlen(objectcode3\_B) / 4;

char M\_start\_str[100]; char M\_size\_str[100];

sprintf(M\_start\_str, "%06X", M\_start);

sprintf(M\_size\_str, "%02X", M\_size);

strcpy(Modified\_Start[m], M\_start\_str);

strcpy(Modified\_Size[m++], M\_size\_str);

}

}

//CASE: format3 && immediate addressing

else if(immediate){

if(objectcode2\_B[e] == '0'){

address = operand\_address;

itoa(address, objectcode3\_B, 2);

Z = 12 - strlen(objectcode3\_B);

strcat(zero[Z], objectcode3\_B);

strcpy(objectcode3\_B, zero[Z]);

strcpy(zero[Z], Zero[Z]);

}

}

else{

//CASE: relative address >= 0

if(address >= 0){

itoa(address, objectcode3\_B, 2);

Z = 12 - strlen(objectcode3\_B);

strcat(zero[Z], objectcode3\_B);

strcpy(objectcode3\_B, zero[Z]);

strcpy(zero[Z], Zero[Z]);

}

//CASE: relative address < 0

else{

address \*= -1;

itoa(address, objectcode3\_B, 2);

Z = 12 - strlen(objectcode3\_B);

strcat(zero[Z], objectcode3\_B);

strcpy(objectcode3\_B, zero[Z]);

strcpy(zero[Z], zero[Z]);

for (int i = 0; i < strlen(objectcode3\_B); ++i){

if (objectcode3\_B[i] == '0') objectcode3\_B[i] = '1';

else objectcode3\_B[i] = '0';

}

int converted = strtol(objectcode3\_B, NULL, 2);

converted++;

itoa(converted, objectcode3\_B, 2);

}

}

strcat(objectcode123\_B, objectcode3\_B);

sprintf(objectcode123\_X, "%06X", (int)strtol(objectcode123\_B, NULL, 2));

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//OPCODE가 BYTE일 경우

if(!strcmp(temp\_OPCODE, "BYTE")){

memset(objectcode123\_X, 0, sizeof(objectcode123\_X));

char buffer[100]; char temp[100];

//입력받는 DATA의 TYPE을 고려하여 buffer에 objectcode123 저장

if(temp\_OPERAND[0] == 'X'){

for(int i = 2; i < strlen(temp\_OPERAND) - 1; ++i){

buffer[i-2] = temp\_OPERAND[i];

}

}

else if(temp\_OPERAND[0] == 'C'){

for (int i = 2; i < strlen(temp\_OPERAND) - 1; ++i) {

int k = temp\_OPERAND[i];

itoa(k, temp, 16);

if (i == 2) strcpy(buffer, temp);

else strcat(buffer, temp);

memset(temp, 0, sizeof(temp));

}

}

strcpy(objectcode123\_X, buffer);

for(int i=0; i<strlen(objectcode123\_X); i++){

if(objectcode123\_X[i] >= 'a' && objectcode123\_X[i] <= 'z')

objectcode123\_X[i] = (objectcode123\_X[i] - 'a') + 'A';

}

memset(buffer, 0, sizeof(buffer));

}

//Text Record 저장

fprintf(op1, "%s\n", objectcode123\_X);

}

//Text Record 저장

//format 2가 아닐 경우

if(format == 0 || format == 3){

//Text Record의 size가 기준을 넘지 않았을 경우

if(strlen(ObjectCodeStr) + strlen(objectcode123\_X) <= 60){

//ObjectCodeStr(Text Record에 들어가 있는 Objectcode)에 생성된 objectcode를 저장한다.

strcat(ObjectCodeStr, objectcode123\_X);

}

else{

//Text Record의 size가 기준을 넘을 경우

//기존의 Text Record를 출력한다.

//Text Record: 시작 주소: Start\_str(3byte) + objectcode의 길이 (int)strlen(ObjectCodeStr)/2(2byte) + objectcode(30byte)

fprintf(op2, "T%06X%02X%s\n", Start\_str, (int)strlen(ObjectCodeStr)/2, ObjectCodeStr);

//ObjectCodeStr 초기화

memset(ObjectCodeStr, 0, sizeof(ObjectCodeStr));

strcat(ObjectCodeStr, objectcode123\_X);

//Start\_str 초기화(Text Record의 시작 주소)

Start\_str = line\_locctr[presentline - 1];

}

}

//format 2일 경우

else if (format == 2){

//line 938 ~ 946과 설명 동일

if(strlen(ObjectCodeStr) + strlen(objectcode12\_X) <= 60 && !(!strcmp("RESB", temp\_OPERAND) || !strcmp("RESW", temp\_OPERAND))){

strcat(ObjectCodeStr, objectcode12\_X);

}

else{

fprintf(op2, "T%06X%02X%s\n", Start\_str, (int)strlen(ObjectCodeStr)/2, ObjectCodeStr);

memset(ObjectCodeStr, 0, sizeof(ObjectCodeStr));

strcat(ObjectCodeStr, objectcode12\_X);

Start\_str = line\_locctr[presentline - 1];

}

}

}

else{

//line 938 ~ 946과 설명 동일

if(strlen(ObjectCodeStr) + strlen(objectcode1\_X) <= 60 && !(!strcmp("RESB", temp\_OPERAND) || !strcmp("RESW", temp\_OPERAND))){

strcat(ObjectCodeStr, objectcode1\_X);

}

else{

fprintf(op2, "T%06X%02X%s\n", Start\_str, (int)strlen(ObjectCodeStr)/2, ObjectCodeStr);

memset(ObjectCodeStr, 0, sizeof(ObjectCodeStr));

strcat(ObjectCodeStr, objectcode1\_X);

Start\_str = line\_locctr[presentline - 1];

}

}

}

fclose(ip); fclose(op1); fclose(op2);

if(!ERROR\_FLAG) printf("Successfully assemble %s.\n", filename);

return 0;

}

int opcode\_format(char\* Mnemonic)

{

int Find = 0;

hash\* temphash = (hash\*)malloc(sizeof(hash));

//Mnemonic의 예로 Add가 있다.

//입력받은 opcode에 해당하는 format을 return한다.

for (int h = 0; h < tablesize; ++h) {

temphash = hashtable[h];

while(temphash != NULL){

if (!strcmp(temphash->order, Mnemonic))

{

Find = 1;

if(!strcmp(temphash->format, "1")) return 1;

else if(!strcmp(temphash->format, "2")) {return 2;}

else {return 3;}

}

if (Find) break;

temphash = temphash->next;

}

if (Find) break;

}

if(!Find) return 0;

}

void AddSymtab(int loc, char\* name)

{

//hashtable[key]에 char\* order, char\* format을 저장한다.

SYMTAB\* newsymtab = (SYMTAB\*)malloc(sizeof(SYMTAB));

newsymtab->next = symtab->next;

strcpy(newsymtab->name, name);

newsymtab->loc = loc;

symtab->next = newsymtab;

}

int FindSymtab(char\* name)

{

SYMTAB\* tempsymtab = (SYMTAB\*)malloc(sizeof(SYMTAB));

tempsymtab = symtab;

int Find = 0;

while(tempsymtab != NULL){

//Mnemonic을 찾았을 경우

if (!strcmp(tempsymtab->name, name))

{

Find = 1;

return tempsymtab->loc;

}

if (Find) break;

tempsymtab = tempsymtab->next;

}

if(!Find) return NotInSymtab;

}

char\* itoa(int num, char\* str, int base)

{

int i = 0;

int isNegative = 0;

/\* Handle 0 explicitely, otherwise empty string is printed for 0 \*/

if (num == 0)

{

str[i++] = '0';

str[i] = '\0';

return str;

}

// In standard itoa(), negative numbers are handled only with

// base 10. Otherwise numbers are considered unsigned.

if (num < 0 && base == 10)

{

isNegative = 1;

num = -num;

}

// Process individual digits

while (num != 0)

{

int rem = num % base;

str[i++] = (rem > 9) ? (rem - 10) + 'a' : rem + '0';

num = num / base;

}

// If number is negative, append '-'

if (isNegative)

str[i++] = '-';

str[i] = '\0'; // Append string terminator

// Reverse the string

//reverse(str, i);

i = 0;

int j = strlen(str) - 1; char temp;

while (i < j) {

temp = str[i];

str[i] = str[j];

str[j] = temp;

i++;

j--;

}

return str;

}

int regis\_to\_data(char c)

{

if(c == 'A') return 0;

else if(c == 'X') return 1;

else if(c == 'S') return 4;

else if(c == 'T') return 5;

}