**과목명: 시스템프로그래밍**

**분반: 1반(홀수)**

**<<Project #2>>**

**서강대학교 컴퓨터공학과**

**학번 : 20161641**

**이름 : 정성엽**

**목 차**

1. 프로그램 개요

2. 프로그램 설명

2.1 프로그램 흐름도

2.2 project2 상세 흐름도

3. 모듈 정의

3.1 functions.c 추가함수

3.1.1 int typeFile(int argc,char argv[100][100]);

3.2 assembleFunc.c 내부함수

3.2.1 void initAssemble();

3.2.2 int assembleFile(int argv, char argc[100][100]);

3.2.3 int makeLocationCount(FILE \*fp);

3.2.4 int makeObjCode();

3.2.5 int retLocCount(int argc,char argv[100][100],int symFlag);

3.2.6 int storeSymbol(char str[], int locCount, int arrIdx, symbolNode\*\* sNow, symbolNode\*\* sPrev);

3.2.7 void makeListingFile(FILE \*fp);

3.2.8 int retRegister(char str[]);

3.2.9 void makeObjectFile(FILE \*fp);

3.2.10 int writeOneLine(FILE\* fp,int i,int lenMax);

3.2.11 int printSymbol(int argv, char argc[100][100]);

3.2.12 int recurFindSymbol(char str[],symbolNode \*node);

3.2.13 void recurPrintSymbol(symbolNode \*node);

4. 전역 변수 정의

4.1 구조체 정의

4.1.1 lstNode

4.1.2 symbolNode

4.2 매크로 정의

4.2.1 typeFile() 함수관련 에러 상수 정의

4.2.3 Assemble 관련 에러 상수 정의

4.3 assembleFunc.c 전역변수

4.3.1 lstNode lstArr[1000];

4.3.2 int modFormFour[100];

4.3.3 int mffMax, lstArrSize, baseIdx, baseLine;

4.3.4 char baseName[50];

4.3.5 char programName[50];

4.3.6 symbolNode\* symbolSetPrev = NULL;

4.3.7 symbolNode\* symbolSet = NULL;

4.3.8 int startFlag, endFlag, endLoc;

5. 코드 설명

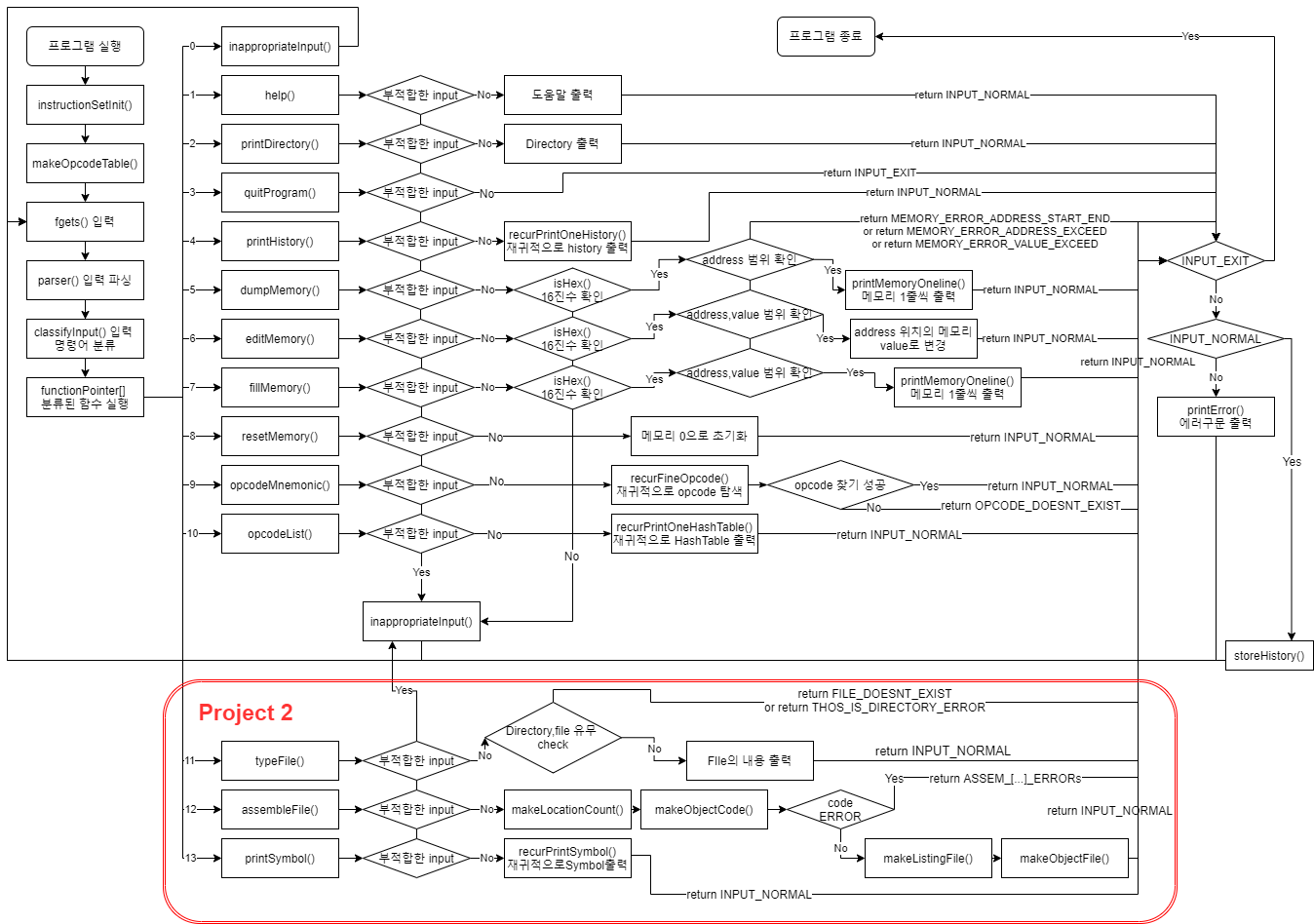
5.1 functions.c 추가함수

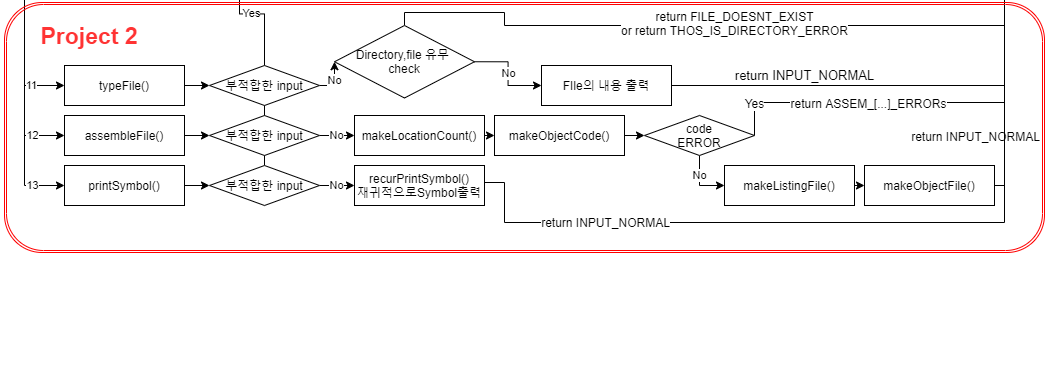
5.2 OpcodeTable.c 코드

1. 프로그램 개요

이 프로그램은 프로젝트1에서 구현한 셀(shell)에 assemble 기능을 구현한 프로그램입니다. SIC/XE의 assembly program source 파일을 입력 받아서 object파일을 생성하고, 어셈블리 과정 중 생성된 symbol table과 결과물인 object 파일을 볼 수 있는 기능을 제공합니다.

2. 프로그램 설명

 2.1 프로그램 흐름도

 2.2 project2 상세 흐름도

3. 모듈 정의

3.1 functions.c 추가함수

3.1.1 int typeFile(int argc,char argv[100][100]);

▶사용 변수

(1) int\* argv;

: 파싱된 명령어 인자들의 개수를 저장한다.

(2) char argc[100][100];

: 파싱된 명령어 인자들을 저장한다.

(3) DIR\* dir;

: opendir()함수를 이용해 현재 directory를 열고 그 포인터를 저장한다.

(4) struct dirent\* entry;

: readdir()함수를 이용해 읽어 들인 파일(혹은 directory) 이름을 저장한다.

(5) struct stat buf;

: 파일(혹은 directory) 정보를 저장한다.

(6) char str[1002];

: 파일의 내용을 한 줄씩 저장한다.

(7) int notEnter = 0;

: 파일의 마지막 줄에 개행이 이루어진 후 끝나는지 그 여부를 저장한다.

▶함수의 동작 및 기능

: 만약 argc가 1이 아니라면 잘못된 명령어가 입력된 것임으로 INPUT\_ERROR를 return한다. opendir()함수를 이용해 현재 directory를 열고, readdir()함수를 이용해 파일(혹은 directory)정보를 읽어 들인다. 이 정보를 이용해 만약 이름에 해당하는 객체가 directory라면 THIS\_IS\_DIRECTORY\_ERROR를 return 한다(S\_ISDIR(buf.st\_mode)사용). 아니라면, 예외적인 경우인 자신의 부모 디랙토리인 경우를 예외처리하고, 파일을 한 줄씩 읽어 출력한다. 만약 파일이 개행 없이 끝난다면 notEnter변수를 사용해 확인하고, 파일의 마지막 줄을 출력한 후 개행을 해준다. (아니라면 파일의 마지막 줄을 출력하고 이어서 sicsim> 이 출력되는 문제가 생긴다)

3.2 assembleFunc.c 내부함수

3.2.1 void initAssemble();

▶사용 변수

: 모든 전역변수

▶함수의 동작 및 기능

: 모든 전역변수의 초기화를 수행한다. 이때 만약 assemble이 실패하는 경우 이전에 성공한 Symbol Linked List를 계속 갖고 있어야 하기 때문에 이전 Symbol Linked List의 시작 포인터를 symbolSetPrev에 저장한다.

3.2.2 int assembleFile(int argv, char argc[100][100]);

▶사용 변수

(1) int\* argv;

: 파싱된 명령어 인자들의 개수를 저장한다.

(2) char argc[100][100];

: 파싱된 명령어 인자들을 저장한다.

(3) FILE \*fp;

: 파일을 열어 사용하기 위한 FILE pointer

(4) int err;

: assemble 과정에서 발생하는 error를 저장하는 변수.

(5) char fileName1[100],fileName2[100];

: [filename].asm에 대응하는 [filename].lst, [filename].obj 파일 이름을 저장하기 위한 변수.

▶함수의 동작 및 기능

: 만약 argc가 2가 아니라면 잘못된 명령어가 입력된 것임으로 INPUT\_ERROR를 return한다. 먼저 initAssemble(); 함수를 호출하여 초기화를 수행한다. 이후 파일을 fopen()하는데, 이때 (fp == NULL)이라면 FILE이 존재하지 않는다는 것 임으로 FILE\_DOESNT\_EXISTfmf return 한다. 이후 makeLocationCount(), makeObjCode() 함수들을 호출하여 Location Count, Object Code를 생성하고 lstArr에 저장한다. 만약 코드에 error가 있는 경우, 그에 해당하는 error를 err변수를 이용해 return 받고 이를 return 한다. 만약 여기까지의 과정에 문제가 없다면 코드상에 문제가 없다는 것임으로 이제 makeListingFile(), makeObjectFile() 함수들을 호출하여 .lst 파일과 .obj 파일을 생성한다. 모든 과정이 정상적으로 종료한 후 INPUT\_NORMAL을 return 한다.

3.2.3 int makeLocationCount(FILE \*fp);

▶사용 변수

(1) char str[100];

: 한 줄씩 읽은 파일의 내용을 저장한다.

(2) int\* argv;

: 파싱된 명령어 인자들의 개수를 저장한다.

(3) char argc[100][100];

: 파싱된 명령어 인자들을 저장한다.

(4) int locCount, locTemp;

: locCount는 각 코드 1줄의 Location Count를 저장한다. locTemp는 각 코드가 차지함으로써 증가하는 Location Count의 크기를 저장한다.

(5) int symFlag;

: SYMBOL이 입력되는 경우와 입력되지 않는 경우를 구분하는 flag.

▶함수의 동작 및 기능

: 파일의 내용을 한 줄씩 읽고, 먼저 각 파일이 Comment인지 (첫 글자로 '.'이 입력되는지 확인한다), 그리고 SYMBOL이 입력되는 경우인지, START, END인지 구분해 각각 해당하는 Location Count를 저장한다.

3.2.4 int makeObjCode();

▶사용 변수

(1) int\* argv;

: 파싱된 명령어 인자들의 개수를 저장한다.

(2) char argc[100][100];

: 파싱된 명령어 인자들을 저장한다.

(3) char str[100] ,tmp[100];

: str은 1줄의 코드를 저장하는 배열이고, tmp는 Mnemonic을 저장하는 배열이다.

(4) int op1,op2;

: opcode를 앞의 8비트, 뒤의 8비트로 나누어 저장하는 변수.

(5) int n,ii,x,b,p,e;

: Format 3, 4인 경우 사용하는 n,i,x,b,p,e를 저장하는 변수.

(6) int disp,addr;

: format 3인 경우 disp에 display을 저장하고, format 4인 경우 addr에 address를 저장한다.

(7) int addrIdx;

: lstArr의 index를 저장하는 변수.

▶함수의 동작 및 기능

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

3.2.5 int retLocCount(int argc,char argv[100][100],int symFlag);

▶사용 변수

(1) int\* argv;

: 파싱된 명령어 인자들의 개수를 저장한다.

(2) char argc[100][100];

: 파싱된 명령어 인자들을 저장한다.

(3) int plusFlag;

: mnemonic이 extended mode(format 4)인지 여부를 저장하는 flag.

(4) char str[100];

: mnemonic을 저장하는 배열.

(5) opcodeNode\* opTemp;

: retOpcode()함수에서 return 받은 opcodeNode를 저장하는 포인터.

▶함수의 동작 및 기능

: 각 operation 코드가 몇 바이트의 크기를 차지하는지 return 한다. 먼저 WORD, RESW, RESW, BYTE의 선언인지 확인하고, 각 경우에 해당하는 크기를 return 한다. Mnemonic 이라면 extended mode(format 4)인지 확인하고 retOpcode()함수를 사용해 해당하는 크기를 return한다.

3.2.6 int storeSymbol(char str[], int locCount, int arrIdx, symbolNode\*\* sNow, symbolNode\*\* sPrev);

▶사용 변수

(1) char str[];

: 저장할 symbol의 이름을 저장하는 변수.

(2) int locCount;

: 저장할 symbol의 Location Count를 저장하는 변수

(3) int arrIdx;

: 저장할 symbol이 저장된 lstArr의 index를 저장하는 변수

(4) symbolNode\*\* sNow;

: Symbol Linked List의 현재 노드를 가리키는 pointer.

(5) symbolNode\*\* sPrev;

: Symbol Linked List의 이전 노드를 가리키는 pointer.

(6) symbolNode\* newNode;

: 새롭게 저장할 symbolNode를 할당받을 pointer.

▶함수의 동작 및 기능

Symbol을 사전순으로 저장하기위해 재귀적으로 저장할 symbol의 위치를 찾는다. 위치를 발견한 다음, newNode에 구조체를 할당 받고 이를 저장한다.

3.2.7 void makeListingFile(FILE \*fp);

▶사용 변수

(1) FILE \*fp;

: 저장할 .lst 파일의 파일포인터가 저장된 변수.

▶함수의 동작 및 기능

: makeLocationCount(), makeObjCode() 함수들을 통해 생성된 Location Count, Object Code를 저장하고 있는 lstArr을 이용해 한 줄씩 Listing File을 저장한다.

3.2.8 int retRegister(char str[]);

▶사용 변수

(1) char str[];

: 확인할 register이름이 저장된 변수.

▶함수의 동작 및 기능

: str에 해당하는 register 번호를 return 한다.

3.2.9 void makeObjectFile(FILE \*fp);

▶사용 변수

(1) FILE \*fp;

: 저장할 .obj 파일의 파일포인터가 저장된 변수.

▶함수의 동작 및 기능

: 먼저 H라인을 저장하고, writeOneLine() 함수를 사용하여 한 줄씩, lstArr에 저장된 Object Code를 .obj file에 저장한다(T 라인들). 그리고 modFormFour에 저장된 정보를 사용해 M라인을 작성한다. 마지막으로 E라인을 저장하고 종료한다.

3.2.10 int writeOneLine(FILE\* fp,int i,int lenMax);

▶사용 변수

(1) FILE \*fp;

: 저장할 .obj 파일의 파일포인터가 저장된 변수.

(2) int I;

: 출력할 ObjectCode가 있는 위치 index를 저장하는 변수.

(3) int lenMax;

: 출력 가능한 최대길이

▶함수의 동작 및 기능

: makeObjectFile() 함수에서 호출되어 i부터 시작하여 변수가 있을 때 까지, 혹은 lenMax에 도달할 때 까지 .obj파일에 1줄을 저장한다.

3.2.11 int printSymbol(int argv, char argc[100][100]);

▶사용 변수

(1) int\* argv;

: 파싱된 명령어 인자들의 개수를 저장한다.

(2) char argc[100][100];

: 파싱된 명령어 인자들을 저장한다.

▶함수의 동작 및 기능

: 만약 argc가 1가 아니라면 잘못된 명령어가 입력된 것임으로 INPUT\_ERROR를 return한다. 정상적인 경우 recurPrintSymbol() 함수를 호출하여 한 줄씩 symbol을 출력한다. 완료 후 INPUT\_NORMAL을 return한다.

3.2.12 int recurFindSymbol(char str[],symbolNode \*node);

▶사용 변수

(1) char str[];

: 찾고있는 Symbol의 이름이 저장된 변수.

(2) symbolNode \*node;

: 탐색하는 Symbol Linked List의 노드를 가리키는 pointer.

▶함수의 동작 및 기능

: 재귀적으로 str에 해당하는 symbol을 찾을 때 까지 Symbol Linked List를 탐색한다.

3.2.13 void recurPrintSymbol(symbolNode \*node);

▶사용 변수

(1) symbolNode \*node;

: 출력하는 Symbol Linked List의 노드를 가리키는 pointer.

▶함수의 동작 및 기능

: 재귀적으로 Linked List를 따라 1개씩 Symbol과 그 Location Count를 출력한다.

4. 전역 변수 정의

4.1 구조체 정의

4.1.1 lstNode

▶원소

(1) int locCount;

: 코드의 Location Count를 저장.

(2) char str[50];

: assembly 코드 1줄을 저장.

(3) long long objCode;

: 코드의 Object Code를 저장.

(4) char objStr[10];

: 출력 규격의 이유로 Object Code를 숫자로 저장할 수 없는 경우, 문자열로 이를 저장한다. (이 경우 objCode = -1), 그리고 objCode가 정상적으로 이용되는 경우, objCode가 몇 자리 수로 출력되는지 저장한다.

▶역할

: 읽은 assembly 코드를 한 줄씩 그 코드의 Location Count, Object Code를 저장하는 배열의 노드.

4.1.2 symbolNode

▶원소

(1) int locCount;

: symbol의 Location Count를 저장.

(2) int arrIdx;

: lstArr에서 몇 번째 index에 저장되어있는지 그 index를 저장한다.

(3) char str[50];

: symbol의 이름을 저장한다.

(4) struct \_symbolNode\* link;

: Linked List의 다음 노드를 가리키는 pointer.

▶역할

: Symbol의 정보를 저장하는 Linked List의 노드. Symbol의 이름과 lstArr에서 몇 번째 index에 저장되어있는지 위치를 저장한다. 또한 그 symbol의 Location Count를 저장한다.

4.2 매크로 정의

4.2.1 typeFile() 함수관련 에러 상수 정의

#define FILE\_DOESNT\_EXIST (7)

#define THIS\_IS\_DIRECTORY\_ERROR (13)

4.2.3 Assemble 관련 에러 상수 정의

#define ASSEM\_OPCODE\_ERROR (8)

#define ASSEM\_SYMBOL\_DUPLICATION\_ERROR (9)

#define ASSEM\_BASE\_NAME\_ERROR (10)

#define ASSEM\_END\_OPCODE\_DOESNT\_EXIST (11)

#define ASSEM\_START\_OPCODE\_DOESNT\_EXIST (12)

#define ASSEM\_WRONG\_REGISTER (14)

#define ASSEM\_SYMBOL\_DOESNT\_EXIST (15)

#define ASSEM\_ADDRESS\_OUT\_OF\_RANGE (16)

#define ASSEM\_NUMBER\_OUT\_OF\_RANGE (17)

#define ASSEM\_BYTE\_WRONG\_OPERAND (18)

4.3 assembleFunc.c 전역변수

4.3.1 lstNode lstArr[1000];

: 읽은 assembly 코드를 한 줄씩 그 코드의 Location Count, Object Code를 저장하는 배열

4.3.2 int modFormFour[100];

: Object File를 만들 때 M라인을 적기 위해 format4인 코드의 위치를 저장하는 배열

4.3.3 int mffMax, lstArrSize, baseIdx, baseLine;

: mffMax는 4.3.2의 modFormFour 배열의 인덱스 범위를 저장한다. lstArrSize는 4.3.1의 lstArr배열의 인덱스 범위를 저장한다. baseIdx는 BASE 선언부가 lstArr에 저장된 위치를 저장한다. baseLine은 BASE로 선언된 SYMBOL이 lstArr에 저장된 위치를 저장한다.

4.3.4 char baseName[50];

: BASE로 선언된 SYMBOL의 이름을 저장한다.

4.3.5 char programName[50];

: START라인에 선언된 프로그램의 이름을 저장한다.

4.3.6 symbolNode\* symbolSetPrev = NULL;

: 만약 assemble이 실패하는 경우 이전에 성공한 Symbol Linked List를 계속 갖고 있어야 하기 때문에 이전 Symbol Linked List의 시작 포인터를 저장해주는 변수.

4.3.7 symbolNode\* symbolSet = NULL;

: Symbol Linked List의 시작주소를 저장하는 pointer.

4.3.8 int startFlag, endFlag, endLoc;

: startFlag는 START가 정상적으로 사용되었는지 여부를 저장한다. endFlag는 END가 정상적으로 사용되었는지 여부를 저장한다. endLoc는 프로그램 코드의 마지막 LocationCount를 저장한다.

5. 코드 설명

5.1 functions.c 추가/변경 함수 코드

void printError(int errorCase){ // 에러 종류별 에러구문 출력 //에러 처리부분 변경됨

switch(errorCase){

case MEMORY\_ERROR\_VALUE\_EXCEED:

printf("VALUE OUT OF RANGE\n");

break;

case MEMORY\_ERROR\_ADDRESS\_EXCEED:

printf("ADDRESS OUT OF RANGE\n");

break;

case MEMORY\_ERROR\_ADDRESS\_START\_END:

printf("START ADDRESS IS BIGGER THEN END ADDRESS\n");

break;

case OPCODE\_DOESNT\_EXIST:

printf("OPCODE DOESN'T EXIST\n");

break;

case FILE\_DOESNT\_EXIST:

printf("FILE DOESN'T EXIST\n");

break;

case ASSEM\_OPCODE\_ERROR:

printf("ASSEMBLER CODE : WRONG OPERATION\n");

break;

case ASSEM\_SYMBOL\_DUPLICATION\_ERROR:

printf("ASSEMBLER CODE : SYMBOL DUPLICATED\n");

break;

case ASSEM\_BASE\_NAME\_ERROR:

printf("ASSEMBLER CODE : WRONG BASE NAME\n");

break;

case ASSEM\_END\_OPCODE\_DOESNT\_EXIST:

printf("ASSEMBLER CODE : END DOESN'T EXIST\n");

break;

case ASSEM\_START\_OPCODE\_DOESNT\_EXIST:

printf("ASSEMBLER CODE : START DOESN'T EXIST\n");

break;

case THIS\_IS\_DIRECTORY\_ERROR:

printf("THIS IS DIRECTORY\n");

break;

case ASSEM\_WRONG\_REGISTER:

printf("ASSEMBLER CODE : WRONG REGISTER NAME\n");

break;

case ASSEM\_SYMBOL\_DOESNT\_EXIST:

printf("ASSEMBLER CODE : SYMBOL DOESN'T EXIST\n");

break;

case ASSEM\_ADDRESS\_OUT\_OF\_RANGE:

printf("ASSEMBLER CODE : ADDRESS OUT OF RANGE (Use format4)\n");

break;

case ASSEM\_NUMBER\_OUT\_OF\_RANGE:

printf("ASSEMBLER CODE : NUMBER OUT OF RANGE\n");

break;

case ASSEM\_BYTE\_WRONG\_OPERAND:

printf("ASSEMBLER CODE : WRONG BYTE OPERAND\n");

break;

}

}

int typeFile(int argc,char argv[100][100]){ // 입력받은 파일의 내용을 출력 (함수 포인터 11)

if(argc != 2)

return INPUT\_ERROR;

DIR\* dir = NULL;

struct dirent\* entry;

struct stat buf;

if((dir = opendir("./")) != NULL){

while((entry = readdir(dir)) != NULL){

lstat(entry->d\_name,&buf);

if(strcmp(entry->d\_name,argv[1]) == 0){

if(S\_ISDIR(buf.st\_mode)) // if it is diectory

return THIS\_IS\_DIRECTORY\_ERROR;

else

break;

}

}

closedir(dir);

}

if(strcmp("./",argv[1]) == 0)

return THIS\_IS\_DIRECTORY\_ERROR;

if(strcmp("../",argv[1]) == 0)

return THIS\_IS\_DIRECTORY\_ERROR;

char str[1002];

int notEnter = 0;

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

if(fp == NULL)

return FILE\_DOESNT\_EXIST;

while(fgets(str,1000,fp)!=NULL){ // read line by line

printf("%s",str);

if(strlen(str)>0 && str[strlen(str)-1] != '\n')

notEnter = 1;

else

notEnter = 0;

}

fclose(fp);

if(notEnter) // if file don't ends woth enter

printf("\n");

return INPUT\_NORMAL;

}

5.2 OpcodeTable.c 코드

#include "20161641.h"

lstNode lstArr[1000];

int modFormFour[100];

int mffMax = 0;

int lstArrSize = 0, baseIdx = 0, baseLine = 0;

char baseName[50];

char programName[50];

symbolNode\* symbolSetPrev = NULL;

symbolNode\* symbolSet = NULL;

int startFlag = -1,endFlag = -1;

int endLoc = 0;

void initAssemble(){ // lstNode 배열과 Symbol 리스트를 초기화 한다

symbolSetPrev = symbolSet; // 이후free 추가

symbolSet = NULL;

for(int i = 0;i<lstArrSize;i++){

lstArr[i].locCount = 0;

lstArr[i].str[0] = 0;

lstArr[i].objCode = 0;

}

lstArrSize = baseIdx = baseLine = mffMax = endLoc = 0;

startFlag = endFlag = -1;

}

int assembleFile(int argv, char argc[100][100]){ // 입력받은 파일의 object file과 listing file을 만든다

if(argv != 2)

return INPUT\_ERROR;

initAssemble();

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

if(fp == NULL)

return FILE\_DOESNT\_EXIST;

int err = makeLocationCount(fp);

fclose(fp);

if(err != 0){

symbolSet = symbolSetPrev;

return err;

}

err = makeObjCode();

if(err != 0){

symbolSet = symbolSetPrev;

return err;

}

/////////////////////////////////////////

char fileName1[100],fileName2[100];

int i = 0;

while(argc[1][i] != '.'){

fileName1[i] = fileName2[i] = argc[1][i];

i++;

}

fileName1[i] = fileName2[i] = '\0';

strcat(fileName1,".lst");

strcat(fileName2,".obj");

////////////////////////////////////////

fp = fopen(fileName1,"w");

if(fp == NULL)

return FILE\_DOESNT\_EXIST;

makeListingFile(fp);

fclose(fp);

///////////////////////////////////////

fp = fopen(fileName2,"w");

if(fp == NULL)

return FILE\_DOESNT\_EXIST;

makeObjectFile(fp);

fclose(fp);

return INPUT\_NORMAL;

}

int makeLocationCount(FILE \*fp){ // location count를 할당하고, symbol table을 만든다

char str[100];

int argc = 0;

char argv[100][100];

int locCount = 0,locTemp = 0;

int symFlag = 0;

while(fgets(str,100,fp)!=NULL){

if(strlen(str)>0)

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

strcpy(lstArr[lstArrSize].str,str);

if(str[0] != ' ' && str[0] != '\t') symFlag = 1;

else symFlag = 0;

parser(str,&argc,argv,", \f\n\r\t\v");

if(argc == 0){

lstArr[lstArrSize].locCount = -1;

lstArr[lstArrSize].objCode = -1;

lstArrSize++;

continue;

}

if(argv[0][0] == '.'){

lstArr[lstArrSize].locCount = -1;

lstArr[lstArrSize].objCode = -1;

lstArrSize++;

continue;

}

if(strcmp("START",argv[symFlag]) == 0){

startFlag = lstArrSize;

strcpy(programName,argv[0]);

locCount = strtol(argv[symFlag+1],NULL,16);

locTemp = locCount;

lstArr[lstArrSize].locCount = locCount;

lstArr[lstArrSize].objCode = -1;

lstArrSize++;

continue;

}

if(startFlag == -1){

printf("LINE : (%d) :",lstArrSize);

return ASSEM\_START\_OPCODE\_DOESNT\_EXIST;

}

if(strcmp("END",argv[symFlag]) == 0){

lstArr[lstArrSize].locCount = -1;

lstArr[lstArrSize].objCode = -1;

lstArrSize++;

endFlag = lstArrSize;

break;

}

if(strcmp("BASE",argv[symFlag]) == 0){

lstArr[lstArrSize].locCount = -1;

lstArr[lstArrSize].objCode = -1;

baseLine = lstArrSize;

lstArrSize++;

strcpy(baseName,argv[symFlag+1]);

continue;

}

locTemp = retLocCount(argc,argv,symFlag);

if(locTemp == -1){ // ERROR CASE

printf("LINE : (%d) :",lstArrSize);

return ASSEM\_OPCODE\_ERROR;

}

lstArr[lstArrSize].locCount = locCount;

lstArr[lstArrSize].objCode = 0;

if(symFlag&&!storeSymbol(argv[0],locCount,lstArrSize,&symbolSet,&symbolSet)){

printf("[%s],",argv[0]);

printf("LINE : (%d) :",lstArrSize);

return ASSEM\_SYMBOL\_DUPLICATION\_ERROR;

}

lstArrSize++;

locCount += locTemp;

}

endLoc = locCount;

if(endFlag == -1){

printf("LINE : (%d) :",lstArrSize-1);

return ASSEM\_END\_OPCODE\_DOESNT\_EXIST;

}

baseIdx = recurFindSymbol(baseName,symbolSet);

if(baseIdx == -1){

baseIdx = 0;

printf("LINE : (%d) :",baseLine);

return ASSEM\_BASE\_NAME\_ERROR;

}

return 0;

}

int retLocCount(int argc,char argv[100][100],int symFlag){ // 각 operation이 얼만큼의 크기를 갖는지 return한다

if(strcmp("WORD",argv[symFlag]) == 0)

return 3;

else if(strcmp("RESW",argv[symFlag]) == 0)

return 3\*atoi(argv[symFlag+1]);

else if(strcmp("RESB",argv[symFlag]) == 0)

return atoi(argv[symFlag+1]);

else if(strcmp("BYTE",argv[symFlag]) == 0){

if(argv[symFlag+1][0] == 'C')

return strlen(argv[symFlag+1])-3;

else if(argv[symFlag+1][0] == 'X')

return 1;

else return 0;

}

int plusFlag = 0;

char str[100];

if(argv[symFlag][0] == '+'){

plusFlag = 1;

strcpy(str,argv[symFlag]+1);

}

else

strcpy(str,argv[symFlag]);

opcodeNode\* opTemp = retOpcode(str);

if(opTemp == NULL)

return -1;

return opTemp->val[plusFlag];

}

int storeSymbol(char str[], int locCount, int arrIdx, symbolNode\*\* sNow, symbolNode\*\* sPrev){ // 재귀적으로 정렬을 유지하면서 symbol을 저장한다

if((\*sNow) != NULL){

int cmp = strcmp((\*sNow)->str,str);

if(cmp == 0) return 0;

else if(cmp < 0) {

return storeSymbol(str,locCount,arrIdx,&((\*sNow)->link),sNow);

}

}

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

strcpy(newNode->str,str);

newNode->locCount = locCount;

newNode->arrIdx = arrIdx;

if((\*sNow)== NULL && (\*sPrev) == NULL){

newNode->link = NULL;

(\*sPrev) = newNode;

}

else if((\*sNow) == (\*sPrev)){

newNode->link = (\*sNow);

(\*sPrev)=newNode;

}

else {

newNode->link = (\*sNow);

(\*sPrev)->link = newNode;

}

return 1;

}

int retRegister(char str[]){ // string에 해당하는 register return

if(strcmp(str,"A") == 0)

return 0;

else if(strcmp(str,"B") == 0)

return 3;

else if(strcmp(str,"X") == 0)

return 1;

else if(strcmp(str,"S") == 0)

return 4;

else if(strcmp(str,"L") == 0)

return 2;

else if(strcmp(str,"T") == 0)

return 5;

else if(strcmp(str,"PC") == 0)

return 8;

else if(strcmp(str,"F") == 0)

return 6;

else if(strcmp(str,"SW") == 0)

return 9;

else

return 0;

}

int makeObjCode(){ // ObjCode를 만든다

int argc = 0,symFlag = 0;

char argv[100][100];

char str[100],tmp[100];

int op1,op2;

int n,ii,x,b,p,e;

int disp,addr;

int addrIdx;

for(int i = 0;i<lstArrSize;i++){

op1 = op2 = n = ii = x = b = p = e = disp = addr = 0;

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

strcpy(str,lstArr[i].str);

if(str[0] != ' ' && str[0] != '\t') symFlag = 1;

else symFlag = 0;

parser(str,&argc,argv,", \f\n\r\t\v");

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

argv[i][0] = 0;

if(strcmp("WORD",argv[symFlag]) == 0){

lstArr[i].objCode = (long long)atoi(argv[symFlag+1]);

}

else if(strcmp("RESW",argv[symFlag]) == 0){

lstArr[i].objCode = -1;

}

else if(strcmp("RESB",argv[symFlag]) == 0){

lstArr[i].objCode = -1;

}

else if(strcmp("BYTE",argv[symFlag]) == 0){

int j = 2;

if(argv[symFlag+1][0] == 'C'){

while(argv[symFlag+1][j] != '\'' && argv[symFlag+1][j] != '\0'){

disp \*= 16\*16;

disp += argv[symFlag+1][j];

j++;

}

lstArr[i].objCode = disp;

}

else if(argv[symFlag+1][0] == 'X'){

while(argv[symFlag+1][j] != '\'' && argv[symFlag+1][j] != '\0'){

tmp[j-2] = argv[symFlag+1][j];

j++;

}

tmp[j-2] = '\0';

lstArr[i].objCode = -2;

strcpy(lstArr[i].objStr,tmp);

}

// else // error case

}

else{

if(argv[symFlag][0] == '+'){

e = 1;

strcpy(tmp,argv[symFlag]+1);

}

else

strcpy(tmp,argv[symFlag]);

opcodeNode\* opTemp = retOpcode(tmp);

op2 = opTemp->opcode;

op1 = op2/16;

op2 %= 16;

op2 /= 4;

if(opTemp->val[e] == 1){

lstArr[i].objCode = opTemp->opcode;

lstArr[i].objStr[0] = '2';

}

else if(opTemp->val[e] == 2){

op2\*=4;

lstArr[i].objCode=op1;

lstArr[i].objCode\*=16;

lstArr[i].objCode+=op2;

lstArr[i].objCode\*=16;

lstArr[i].objCode+=retRegister(argv[symFlag+1]);

lstArr[i].objCode\*=16;

lstArr[i].objCode+=retRegister(argv[symFlag+2]);

lstArr[i].objStr[0] = '4';

}

else if(opTemp->val[e] == 3 || opTemp->val[e] == 4){

if(argc>1+symFlag){

if(argv[symFlag+1][0] == '#'){

n = 0, ii = 1;

strcpy(tmp,argv[symFlag+1]+1);

}

else if(argv[symFlag+1][0] == '@'){

n = 1, ii = 0;

strcpy(tmp,argv[symFlag+1]+1);

}

else{

n = 1, ii = 1;

strcpy(tmp,argv[symFlag+1]);

}

if(argc == 3+symFlag)

x = 1;

addrIdx = recurFindSymbol(tmp,symbolSet);

if(addrIdx == -1){

disp = atoi(tmp); // 숫자가 아닌경우 error check 해줘야 함

}

else{

int pcCounter = lstArr[addrIdx].locCount-(lstArr[i].locCount+3);

if(-2048<=pcCounter && pcCounter<=2047){

p = 1;

if(pcCounter<0){ // 2의 보수를 취해준다

pcCounter += 4096;

disp = pcCounter;

}

else

disp = pcCounter;

}

else{

b = 1;

disp = lstArr[addrIdx].locCount-lstArr[baseIdx].locCount;

if(0>disp || disp >4095) // ERROR

disp = 0;

}

}

}

else{

n = 1, ii = 1;

}

if(e)

b = 0,p = 0;

lstArr[i].objCode=op1;

lstArr[i].objCode\*=4;

lstArr[i].objCode+=op2;

lstArr[i].objCode\*=2;

lstArr[i].objCode+=n;

lstArr[i].objCode\*=2;

lstArr[i].objCode+=ii;

lstArr[i].objCode\*=2;

lstArr[i].objCode+=x;

lstArr[i].objCode\*=2;

lstArr[i].objCode+=b;

lstArr[i].objCode\*=2;

lstArr[i].objCode+=p;

lstArr[i].objCode\*=2;

lstArr[i].objCode+=e;

if(e == 0){

lstArr[i].objCode\*=4096;

lstArr[i].objCode+=disp;

lstArr[i].objStr[0] = '6';

}

else{

if(addrIdx != -1){

disp = lstArr[addrIdx].locCount;

modFormFour[mffMax++] = lstArr[i].locCount+1;

}

lstArr[i].objCode\*=1048576;

lstArr[i].objCode+=disp;

lstArr[i].objStr[0] = '8';

}

}

}

}

}

return 0;

}

void makeListingFile(FILE \*fp){ // listing file을 만든다

for(int i = 0;i<lstArrSize;i++){

fprintf(fp,"%-7d ",i\*5+5);

if(lstArr[i].locCount < 0)

fprintf(fp," ");

else

fprintf(fp,"%04X",lstArr[i].locCount);

fprintf(fp," %s",lstArr[i].str);

if(lstArr[i].objCode == -1)

fprintf(fp,"\n");

else{

for(int j = (int)strlen(lstArr[i].str);j<30;j++)

fprintf(fp," ");

if(lstArr[i].objCode != -2){

if(lstArr[i].objStr[0] == '6')

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

else if(lstArr[i].objStr[0] == '8')

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

else if(lstArr[i].objStr[0] == '4')

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

else if(lstArr[i].objStr[0] == '2')

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

else

fprintf(fp,"%llX\n",lstArr[i].objCode);

}

else

fprintf(fp,"%s\n",lstArr[i].objStr);

}

}

}

void makeObjectFile(FILE \*fp){ // object file을 만든다

int lenMax = 60; // lenMax < 256

int idx = startFlag;

fprintf(fp,"H%-6s%06X%06X\n",programName,lstArr[startFlag].locCount,endLoc);

while(idx<endFlag){

idx = writeOneLine(fp,idx,lenMax);

}

for(int i = 0;i<mffMax;i++){

fprintf(fp,"M%06X05\n",modFormFour[i]);

}

fprintf(fp,"E%06X",lstArr[startFlag].locCount);

}

int writeOneLine(FILE\* fp,int i,int lenMax){ // 1줄씩 적어준다, 다음 시작 index를 return

int length = 0;

int varFlag = 0;

char str[300] = {0,};

char tmp[10];

if(lstArr[i].locCount == -1) // 공백, 주석, BASE 선언부

return i+1;

if(lstArr[i].objCode == -1) //변수

return i+1;

fprintf(fp,"T%06X",lstArr[i].locCount);

for(;i<endFlag;i++){

if(lstArr[i].locCount == -1) // 공백, 주석, BASE 선언부

continue;

if(lstArr[i].objCode == -1) //변수

break;

else if(lstArr[i].objCode == -2) //string 으로 저장

strcpy(tmp,lstArr[i].objStr);

else{

if(lstArr[i].objStr[0] == '6')

sprintf(tmp,"%06llX",lstArr[i].objCode);

else if(lstArr[i].objStr[0] == '8')

sprintf(tmp,"%08llX",lstArr[i].objCode);

else if(lstArr[i].objStr[0] == '4')

sprintf(tmp,"%04llX",lstArr[i].objCode);

else if(lstArr[i].objStr[0] == '2')

sprintf(tmp,"%02llX",lstArr[i].objCode);

else

sprintf(tmp,"%llX",lstArr[i].objCode);

}

if(strlen(str)+strlen(tmp)>lenMax){

i--;

break;

}

strcat(str,tmp);

}

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

return i+1;

}

int printSymbol(int argv, char argc[100][100]){ // symbol table을 출력한다

if(argv != 1)

return INPUT\_ERROR;

recurPrintSymbol(symbolSet);

return INPUT\_NORMAL;

}

int recurFindSymbol(char str[],symbolNode \*node){ // 재귀적으로 symbol을 탐색

if(node==NULL)

return -1;

if(strcmp(node->str,str) == 0)

return node->arrIdx;

return recurFindSymbol(str,node->link);

}

void recurPrintSymbol(symbolNode \*node){ // 재귀적으로 하나씩 출력

if(node==NULL)

return;

printf("\t%-9s%04X\n",node->str,node->locCount);

recurPrintSymbol(node->link);

}