과목 명: 시스템프로그래밍

담당 교수 명: 김 지 환

<<Assignment 2>>

**서강대학교 컴퓨터학과**

**[20161620]**

**[이수연]**

목 차

1. 프로그램 개요 3

2. 프로그램 설명 3

2.1 프로그램 흐름도 3

3. 모듈 정의 3

3.1 모듈 이름 : main() 3

3.1.1 기능 3

3.1.2 사용 변수 3

3.2 모듈 이름: findSpace() 3

3.2.1 기능 3

3.2.2 사용 변수 3

3.3 모듈이름: find\_reg\_num() 4

3.3.1 기능 4

3.3.2 사용변수 4

3.4 모듈이름: makeLST() 4

3.4.1 기능 4

3.4.2 사용변수 4

4

4. 전역 변수 정의 4

5. 코드

# 프로그램 개요

프로젝트1에서 구현한 셀에 assembler의 기능을 추가한다. SIC/XE의 assembly program source 파일을 입력 받아서 loc와 object code를 입력한 listing file, 그리고 그를 토대로object file을 생성한다.

# 프로그램 설명

## 프로그램 흐름도

main

1) 메인에서 명령어를 받는다

arrangeCommand

2) arrangeCommand에서 정상명령어인지

판단하여 정상이면 계속 진행,

비정상이면 다시 돌아가서 입력받는다

insertBack

3) insertBack에서 히스토리에

정상명령어를 넣는다

MakeLST

Main (type)

Main(symbol)

4)main에서 type과 symbol은 간단하게 구현하고 object code를 구성하는 실질적인 부분은 MakeLST의 함수를 따로 작성하였다.

findSpace

5).asm 파일의 명령어, 심볼 등을 구분하고

에러처리하는 findspace함수를 따로 작성하였다.

# 모듈 정의

## 모듈 이름 : main()

### 기능

사용자를 통해 입력을 받아 arrangeCommand함수로 보낸다. 만약 정상 입력이 아닐 경우 리턴값으로 -1이 돌아오고 이때 continue를 통해 다른 활동을 하지 않고 새로운 입력을 받게끔 루프를 설정한다. 또한 새로 처리할 명령어 중 비교적 간단한 type과 symbol의 수행은 따로 함수로 분리하지 않고, main의 각 if문 안에서 처리되게끔 하였다.

### 사용 변수

DIR\* dir;//디렉토리

struct dirent\* ent;

struct stat buf;

char home[1024];

int i,j;

int isRightCommand=1;

int fileFlag = 0;

signed long temp;

FILE \*fp;

unsigned int id;

Node\* pnode = NULL;//출력때 이용할 포인터

Table\* new = NULL;

symbolTab tempTab;

## 모듈 이름: int findSpace(char\* fileStr, int pass\_num)

### 기능

파일에서 읽어온 한 줄을 넘겨받아 이를 명령어, symbol, loc, operand 등으로 분리시켜준다. 또한 명령어 사이의 공백 등의 에러처리를 해준다. Pass\_num에 따라 처리하는 정보가 조금씩 달라진다. Pass\_num이 1일 경우 sic머신 어셈블리 알고리즘의 pass1을 수행하면서 읽어온 파일이다. Pass\_num이 2일 경우 알고리즘의 pass2을 수행하면서 읽어온 파일이다.

### 사용 변수

int i, j;

char storeSTR[MAX] = {'\0',};

char store[MAX] = {'\0',};

char temp[MAX] = {'\0',};

int number = 0;

## 모듈이름: int find\_reg\_num(char\* operand)

### 기능

Sic/xe 머신에서 사용되는 레지스터들과 그 번호를 저장하여 놓은 함수로 operand가 들어왔을 때 해당 레지스터의 번호를 반환하거나 알맞은 레지스터인지 여부를 판단해준다. 적절하지 않을 경우 에러 메세지를 출력한다.

### 사용변수

없음

## 모듈이름: void makeLST(char\* fileName\_assem)

### 기능

이번 두번째 프로젝트의 핵심 기능을 하는 함수로서, .asm파일을 받으면 파일을 읽어 object file을 만들어낸다. 우선 처음 파일을 읽으면서 레지스터와 포맷에 알맞은 loc을 지정해준다. (pass1) 그 다음 다시 파일을 읽으면서 opcode,n,I,x,b,p,e(레지스터),operand를 경우에 따라 병합하여 object code를 만들어준다.(pass2) loc와 object code의 정보를 .lst파일에 써준 후 이를 바탕으로 object program을 .obj파일에 써준다. 인자롤 받는 filename\_assem은 확장자 앞부분에 붙여줄 공통적인 파일 이름이다.

### 사용변수

int LSTline;

unsigned long loc = 0;

unsigned int first\_loc, last\_loc;

unsigned int opcode;

unsigned int id;

FILE \*fp\_ASM;

FILE \*fp\_LST\_pass1;

FILE \*fp\_LST\_pass2;

FILE \*fp\_OBJ;

int isComment;

int i, j;

int symTab\_flag;

int format1\_flag;

int format2\_flag;

int format4\_flag;

int immediate\_flag;

int number\_immediate\_flag;

int index\_flag;

int find\_flag;

char format[10] = {'\0',};

char storeAddr[15] = {'\0',};

char storeOperand[30] = {'\0',};

unsigned int TR\_flag, TR\_start, TR\_end;

int stopFlag;

int number;

Table\* new = NULL;.

# 전역 변수 정의

typedef struct Table{

int id;

char instruct[MAX];

char format[MAX];

struct Table\* hashNext;

}Table;

Table\* hashTable[MAX\_HASH];

unsigned int key;

char name[MAX];

char format[MAX];

char mnemonic[MAX];

char fileName[MAX];

char fileName\_assem[MAX];

char fileName\_assem\_store[MAX];

char fileStr[MAX];

typedef struct Node{

char data[MAX];

struct Node\* link;

}Node;

Node\* head = NULL;

Node\* tail = NULL;

typedef struct Assem{

char symbol[MAX];

char mnemonic[MAX];

char operand[MAX];

unsigned int loc;

}Assem;

struct Assem assem;

typedef struct symbolTab{

unsigned int loc;

char symbol[MAX];

}symbolTab; //심볼테이블에 사용할 구조체로 symbol과 loc을 저장

symbolTab symTab[MAX];

int symTab\_i;

typedef struct objcode{

unsigned int opcode;

unsigned int ni,x,b,p,e;

unsigned int xbpe;

unsigned int addr;

unsigned int toSub;//주소에서 빼줄것 저장;pc거나base거나immediate면 0저장

unsigned int reg1, reg2;//format2를 위함,예외처리 위해 -1받는경우 있어서 signed

}objcode;

objcode obj;

unsigned int base\_relative;

unsigned int pc\_relative;

typedef struct storeloc{

unsigned int loc;

int RES\_flag;

}storeloc;

storeloc storeLoc[MAX];

unsigned int storeLoc\_i;//loc을 저장하고 특히 변수가 저장되었던 loc엔 flag로 표시해서 obj파일 만들때 length에 들어가지 않게 처리

typedef struct Mod{

unsigned int loc;

unsigned int val;

}Mod;

Mod mod[MAX];

int mod\_i;

signed long start = -1;

signed long end = -1;

signed long value = -1;

signed long address = -1; //보통 메모리 주소는 unsigned지만 예외처리를 위해 음수를 써야했음. 또 FFFFF가 최대니까 범위고 괜찮고..

unsigned int line\_num = 0;//가상메모리 해당 라인의 맨 왼쪽 주소

unsigned int dump\_p = 0;//가상메모리 각 주소 가리키는 용도

unsigned int temp\_p = 0;//현재 dump\_p를 저장할 용도 ;reset때 필요;

unsigned char virtual[MAX\_memory]="is sample program;;saoo3rklsnkada sd 20161620 suyeo leald;jdasAD";

char command[MAX];///전체 명령어를 관장하는 전역변수 ;; 밑의 arrangeCommand를 거치면 명령어가 정리됨

char store\_command[MAX];//히스토리 출력용을 위해서 처음 받은 명령어 저장

char cmp\_command[MAX]; //실제 가능한 명령어(비교용)

char store2[MAX]={'\0', };//입력의 앞뒤 자르고 사이 공백은 냅둠, 필요없는 문자도 저장된 상태;

int return\_flag2 = 1;

char dumpRange[MAX] = {'\0', };//명령어 뒤에 인자만 받아 저장

int oprComma\_flag;//operand가 두개 이상이어서 콤마가 있는경우

# 코드

* 프로젝트가 커져 코드가 너무 길어져 저번 프로젝트의 내용은 제외하고 이번 프로젝트의 핵심 함수인 makeLST함수와 findSpace 함수만 첨부하였다.

void makeLST(char\* fileName\_assem)

{

int LSTline;

unsigned long loc = 0;

unsigned int first\_loc, last\_loc;

unsigned int opcode;

unsigned int id;

FILE \*fp\_ASM;

FILE \*fp\_LST\_pass1;

FILE \*fp\_LST\_pass2;

FILE \*fp\_OBJ;

int isComment;

int i, j;

int symTab\_flag;

int format1\_flag;

int format2\_flag;

int format4\_flag;

int immediate\_flag;

int number\_immediate\_flag;

int index\_flag;

int find\_flag;

char format[10] = {'\0',};

char storeAddr[15] = {'\0',};

char storeOperand[30] = {'\0',};

unsigned int TR\_flag, TR\_start, TR\_end;

int stopFlag;

int number;

Table\* new = NULL;

strcpy(fileName\_assem\_store,fileName\_assem);

strcat(fileName\_assem,".lst");

strcat(fileName\_assem\_store,".obj");

fp\_ASM = fopen(fileName,"r");

fp\_LST\_pass1 = fopen("temp.txt","w");

symTab\_i = 0;

loc = 0;

LSTline = 0;

storeLoc\_i = 0;

while(fgets(fileStr, MAX-1, fp\_ASM) != NULL)

{

LSTline += 5;

immediate\_flag = 0;

index\_flag = 0;

format1\_flag = 0;

format4\_flag = 0;

format2\_flag = 0;

fileStr[strlen(fileStr)-1] = '\0';//개행 없애고

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

{

assem.symbol[i] = '\0';

assem.mnemonic[i] = '\0';

assem.operand[i] = '\0';

}

isComment = findSpace(fileStr,1);

if(isComment == 0)//주석이다..

{

fprintf(fp\_LST\_pass1,"%s\n",fileStr);

continue;

}

else if(isComment == 1)//정상 소스파일이고 받은 문자열은 주석이 아님

{

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

{

format4\_flag = 1;

strcpy(assem.mnemonic, assem.mnemonic+1);

}

if(assem.operand[0] == '#')

{

immediate\_flag = 1;

strcpy(assem.operand, assem.operand+1);

}

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

{

strcpy(assem.operand, assem.operand+1);

}

if(!strcmp(assem.mnemonic,"START"))

{

fprintf(fp\_LST\_pass1,"%04X %s\n",(unsigned int)loc,fileStr);

loc += strtoul(assem.operand,NULL,10);

}

else if(!strcmp(assem.mnemonic,"BASE"))

fprintf(fp\_LST\_pass1,"%s\n",fileStr);

else if(!strcmp(assem.mnemonic,"END"))

fprintf(fp\_LST\_pass1,"%s\n",fileStr);

else

{

fprintf(fp\_LST\_pass1,"%04X %s\n",(unsigned int)loc,fileStr);

symTab\_flag = 0;

if(strlen(assem.symbol) != 0)

{

for(i=0; i<symTab\_i; i++)

{

if(!strcmp(symTab[i].symbol,assem.symbol))

{

symTab\_flag = 1;

break;

}

}

if(symTab\_flag != 1)//없으면 추가

{

symTab[symTab\_i].loc = loc;

strcpy(symTab[symTab\_i].symbol, assem.symbol);

symTab\_i++;

}

}//symbol table 만들기

id = hashFunc(assem.mnemonic);

if(id<MAX\_HASH)

{

if(!strcmp(hashTable[id]->instruct,assem.mnemonic))

{

opcode = hashTable[id]->id;

strcpy(format,hashTable[id]->format);

}

else

{

new = hashTable[id];

while(new->hashNext)

{

if(!strcmp(new->hashNext->instruct,assem.mnemonic))

{

opcode = new->hashNext->id;

strcpy(format,new->hashNext->format);

}

new = new->hashNext;

}

}

if(format4\_flag==1)

{

loc += 4;

}

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

{

format2\_flag = 1;

loc += 2;

}

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

{

format1\_flag = 1;

loc += 1;

}

else

loc += 3;

}//opcode찾아서 있으면 loc+3, format4면 +4

else if(!strcmp(assem.mnemonic,"RESB"))

loc += strtoul(assem.operand,NULL,10);

else if(!strcmp(assem.mnemonic,"RESW"))

loc += (3 \* (strtoul(assem.operand,NULL,10)));

else if(!strcmp(assem.mnemonic,"WORD"))

loc += 3;

else if(!strcmp(assem.mnemonic,"BYTE"))

{

for(i=0; (unsigned int)i<strlen(assem.operand); i++)

{

if(assem.operand[i] != '\'')

continue;

else if(assem.operand[i] == '\'')

break;

}

for(i = i+1; (unsigned int)i<strlen(assem.operand); i++)

{

if(assem.operand[i] != '\'')

{

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

loc++;

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

{

loc++;

i++;

}

}

else if(assem.operand[i] == '\'')

break;

}

}

else

{

if(format4\_flag != 0)

{

printf("error line %d : 명령어가 opcode table에 없습니다. 잘못된 asm 소스코드 입니다.\n",LSTline);

return;

}

}//변수 상수도 아니고 optable에 없음->에러

}

if(!strcmp(assem.mnemonic,"RESW") || !strcmp(assem.mnemonic,"RESB"))

storeLoc[storeLoc\_i].RES\_flag = 1;

else

storeLoc[storeLoc\_i].RES\_flag = 0;

if(strcmp(assem.mnemonic,"BASE") && strcmp(assem.mnemonic,"END"))

{

storeLoc[storeLoc\_i].loc = loc;

storeLoc\_i++;

}

}

else//정상 소스파일 아닐 때

return;

// storeLoc[storeLoc\_i].loc = loc;

// storeLoc\_i++;

}

first\_loc = storeLoc[0].loc;

storeLoc\_i -= 2;

last\_loc = storeLoc[storeLoc\_i].loc;

fclose(fp\_ASM);

fclose(fp\_LST\_pass1);////////////////pass1 끝

///////////////pass2시작////////////////////////

fp\_ASM = fopen("temp.txt","r");

fp\_LST\_pass2 = fopen(fileName\_assem,"w");

fp\_OBJ = fopen(fileName\_assem\_store,"w");

LSTline = 0;

TR\_flag = 0;

stopFlag = 0;

mod\_i = 0;

storeLoc\_i = 0;

TR\_start = TR\_end = first\_loc;

while(fgets(fileStr, MAX-1, fp\_ASM) != NULL)

{

find\_flag = 0;

format1\_flag = 0;

format2\_flag = 0;

format4\_flag = 0;

immediate\_flag = 0;

number\_immediate\_flag = 0;

index\_flag = 0;

LSTline += 5;

fprintf(fp\_LST\_pass2, "%4d ", LSTline);

obj.ni = 3;//대체로 심플어드레싱

obj.x = obj.b = obj.p = obj.e = 0;

obj.xbpe = obj.addr = obj.toSub = 0;

fileStr[strlen(fileStr)-1] = '\0';//개행 없애고

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

{

assem.symbol[i] = '\0';

assem.mnemonic[i] = '\0';

assem.operand[i] = '\0';

}

isComment = findSpace(fileStr,2);

if(isComment == 0)//주석이다..

{

fprintf(fp\_LST\_pass2," %s\n",fileStr);

continue;

}

else if(isComment == 1)//정상 소스파일이고 받은 문자열은 주석이 아님

{

pc\_relative = assem.loc + 3;//대체로 3차이

j=0;

for(i=0; (unsigned int)i<strlen(assem.operand); i++)

{

if(assem.operand[i] == ' ' || assem.operand[i] == '\t')

continue;

else

{

storeOperand[j] = assem.operand[i];

j++;

}

}

storeOperand[j] = '\0';

strcpy(assem.operand,storeOperand);

if(strstr(assem.operand,",X"))

{

index\_flag = 1;

obj.x = 1;

for(j=0; assem.operand[j] != ','; j++)

storeOperand[j] = assem.operand[j];

storeOperand[j] = '\0';

strcpy(assem.operand,storeOperand);

}

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

{

format4\_flag = 1;

strcpy(assem.mnemonic, assem.mnemonic+1);

obj.e = 1;

pc\_relative = assem.loc + 4;

}

if(assem.operand[0] == '#')

{

immediate\_flag = 1;

strcpy(assem.operand, assem.operand+1);

obj.ni = 1;//immediate

pc\_relative = assem.loc + 3;

}

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

{

strcpy(assem.operand, assem.operand+1);

obj.ni = 2;//indirect

pc\_relative = assem.loc + 3;

}

if(LSTline == 5)//첫줄 obj file에 입력

fprintf(fp\_OBJ,"H%-6s%06X%06X\n",assem.symbol,(unsigned int)strtoul(assem.operand,NULL,16),(last\_loc - first\_loc + 1));

if(LSTline != 5 && TR\_flag == 0 && TR\_start != last\_loc)

{

if(stopFlag == 1)

storeLoc\_i += 1;

TR\_start = storeLoc[storeLoc\_i].loc;

number = 0;

while(1)

{

if(storeLoc[storeLoc\_i].loc > last\_loc)

{

storeLoc[storeLoc\_i].loc = last\_loc;

storeLoc\_i++;

break;

}

if(storeLoc[storeLoc\_i].loc - TR\_start > 30)

break;

else

{

if(storeLoc[storeLoc\_i].RES\_flag == 1)

number++;

storeLoc\_i++;

}

}

if(stopFlag == 1)

{

storeLoc\_i -= 1;

number--;

}

storeLoc\_i = storeLoc\_i - 1;

TR\_end = storeLoc[storeLoc\_i].loc;

TR\_flag = 1;

if(TR\_start != last\_loc)

fprintf(fp\_OBJ,"T%06X%02X",TR\_start,TR\_end-TR\_start-(3\*number));

}

if(!strcmp(assem.mnemonic,"START"))

fprintf(fp\_LST\_pass2,"%-40s\n",fileStr);

else if(!strcmp(assem.mnemonic,"RESW") || !strcmp(assem.mnemonic,"RESB"))

fprintf(fp\_LST\_pass2,"%-40s\n",fileStr);

else if(!strcmp(assem.mnemonic,"BYTE"))

{

fprintf(fp\_LST\_pass2,"%-40s",fileStr);

for(i=0; (unsigned int)i<strlen(assem.operand); i++)

{

if(assem.operand[i] != '\'')

continue;

else if(assem.operand[i] == '\'')

break;

}

for(i = i+1; (unsigned int)i<strlen(assem.operand); i++)

{

if(assem.operand[i] != '\'')

{

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

{

fprintf(fp\_LST\_pass2,"%02X",(unsigned int)assem.operand[i]);

fprintf(fp\_OBJ,"%02X",(unsigned int)assem.operand[i]);

}

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

{

fprintf(fp\_LST\_pass2,"%c",assem.operand[i]);

fprintf(fp\_OBJ,"%c",assem.operand[i]);

}

}

else if(assem.operand[i] == '\'')

{

fprintf(fp\_LST\_pass2,"\n");

break;

}

}

}

else if(!strcmp(assem.mnemonic,"WORD"))

{

fprintf(fp\_LST\_pass2,"%-40s",fileStr);

fprintf(fp\_LST\_pass2,"%06X\n",(unsigned int)strtoul(assem.operand,NULL,10));

fprintf(fp\_OBJ,"%06X",(unsigned int)strtoul(assem.operand,NULL,10));

}

else if(!strcmp(assem.mnemonic,"BASE"))

{

fprintf(fp\_LST\_pass2," %-40s\n",fileStr);

i=0;

while(1)

{

if(strlen(symTab[i].symbol) == 0)

break;

if(!strcmp(symTab[i].symbol,assem.operand))

base\_relative = symTab[i].loc;

else if(immediate\_flag == 1)

base\_relative = strtoul(assem.operand,NULL,10);

i++;

}

}

else if(!strcmp(assem.mnemonic,"END"))

fprintf(fp\_LST\_pass2," %-40s\n",fileStr);

else

{

id = hashFunc(assem.mnemonic);

if(id<MAX\_HASH)

{

if(!strcmp(hashTable[id]->instruct,assem.mnemonic))

{

opcode = hashTable[id]->id;

strcpy(format,hashTable[id]->format);

}

else

{

new = hashTable[id];

while(new->hashNext)

{

if(!strcmp(new->hashNext->instruct,assem.mnemonic))

{

opcode = new->hashNext->id;

strcpy(format,new->hashNext->format);

}

new = new->hashNext;

}

}

i = 0;

while(1)

{

if(strlen(symTab[i].symbol) == 0)

break;

if(!strcmp(symTab[i].symbol,assem.operand))

{

obj.addr = symTab[i].loc;

find\_flag = 1;

break;

}

i++;

}

if(find\_flag == 0 && immediate\_flag == 1)

{

obj.addr = strtoul(assem.operand,NULL,10);

find\_flag = 1;

obj.xbpe = 0;

number\_immediate\_flag = 1;

}

if(find\_flag == 0 && find\_reg\_num(assem.operand) != -1)

find\_flag = 1;

if(find\_flag == 0 && strlen(assem.operand) != 0)

{

printf("error line %d : operand가 Symbol Table에 없습니다. 잘못된 asm 소스코드 입니다.\n",LSTline);

//return;

}

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

{

format4\_flag = 1;

obj.e = 1;

pc\_relative = assem.loc + 4;

}

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

{

pc\_relative = assem.loc + 2;

format2\_flag = 1;

}

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

{

pc\_relative = assem.loc + 1;

format1\_flag = 1;

}

if(number\_immediate\_flag != 1)

{

obj.toSub = pc\_relative;

if(((signed int)obj.addr -(signed int)obj.toSub) < -2048 || ((signed int)obj.addr - (signed int)obj.toSub) > 2047)//이때는 pc로 안되어서 base로 전환

{

obj.toSub = base\_relative;

obj.b = 1;

}

else

obj.p = 1;

}

if(format4\_flag == 1)

{

if(assem.mnemonic[0]=='J')

{

obj.opcode = opcode + obj.ni;

obj.xbpe = 1;

obj.addr = obj.addr;

}//점프이면 바로 로드

else

{

obj.opcode = opcode + obj.ni;

obj.xbpe = ((2\*2\*2\*obj.x) + (2\*2\*obj.b) + (2\*obj.p) + obj.e);

obj.addr = obj.addr - obj.toSub;

}

}

else if(format2\_flag == 1)

{

obj.opcode = opcode;

if(strstr(assem.operand,",") && (index\_flag != 1))

{

storeOperand[0] = assem.operand[0];

storeOperand[1] = '\0';

obj.reg1 = (unsigned int)find\_reg\_num(storeOperand);

for(j=0; (unsigned int)j<strlen(assem.operand); j++)

{

if(assem.operand[j] == ',')

break;

}

for(j =j+1; (unsigned int)j<strlen(assem.operand); j++)

{

if(assem.operand[j] != ' ' && assem.operand[j] != '\t')

{

storeOperand[0] = assem.operand[j];

storeOperand[1] = '\0';

obj.reg2 = (unsigned int)find\_reg\_num(storeOperand);

}

}

}//인자가 두개일때

else if(!strstr(assem.operand,","))//인자가 하나일때

{

storeOperand[0] = assem.operand[0];

storeOperand[1] = '\0';

obj.reg1 = (unsigned int)find\_reg\_num(storeOperand);

obj.reg2 = 0;

}

if(obj.reg1 == 7 || obj.reg2 == 7 || obj.reg1 > 9 || obj.reg2 > 9)

{

printf("error line %d : 잘못된 레지스터입니다. 알맞은 asm 소스코드가 아닙니다.\n",LSTline);

return;

}

}

else if(format1\_flag == 1)

obj.opcode = opcode;

else if(!strcmp(assem.mnemonic,"RSUB"))

{

obj.opcode = opcode + obj.ni;

obj.xbpe = 0;

obj.addr = 0;

}//modification

else//format3 일때

{

obj.opcode = opcode + obj.ni;

obj.xbpe = ((2\*2\*2\*obj.x) + (2\*2\*obj.b) + (2\*obj.p) + obj.e);

obj.addr = obj.addr - obj.toSub;

}

}

else

{

//에러메세지와 리턴

}

fprintf(fp\_LST\_pass2,"%-40s",fileStr);

if(format4\_flag == 1)

{

fprintf(fp\_LST\_pass2,"%02X%01X%05X\n",obj.opcode,obj.xbpe,obj.addr);//format3일 때

fprintf(fp\_OBJ,"%02X%01X%05X",obj.opcode,obj.xbpe,obj.addr);

}

else if(format2\_flag == 1)

{

fprintf(fp\_LST\_pass2,"%02X%01X%01X\n",obj.opcode,obj.reg1,obj.reg2);

fprintf(fp\_OBJ,"%02X%01X%01X",obj.opcode,obj.reg1,obj.reg2);

}

else if(format1\_flag == 1)

{

fprintf(fp\_LST\_pass2,"%02X\n",obj.opcode);

fprintf(fp\_OBJ,"%02X",obj.opcode);

}

else//format3일때

{

if((signed int)obj.addr < 0)

{

sprintf(storeAddr,"%X",obj.addr);

j=0;

for(i=strlen(storeAddr)-3; (unsigned int)i<strlen(storeAddr); i++)

{

storeAddr[j] = storeAddr[i];

j++;

}

storeAddr[j] = '\0';

obj.addr = strtoul(storeAddr,NULL,16);

}

fprintf(fp\_LST\_pass2,"%02X%01X%03X\n",obj.opcode,obj.xbpe,obj.addr);//드디어 object code!!

fprintf(fp\_OBJ,"%02X%01X%03X",obj.opcode,obj.xbpe,obj.addr);

}

}

if(stopFlag != 1 && (!strcmp(assem.mnemonic,"RESW") || !strcmp(assem.mnemonic,"RESB")))

stopFlag = 1;

if(LSTline != 5)

{

storeLoc\_i -= 1;

if(stopFlag != 1)

TR\_end = storeLoc[storeLoc\_i].loc;

if(assem.loc == TR\_end && TR\_start != last\_loc)

{

fprintf(fp\_OBJ,"\n");

storeLoc\_i += 1;

TR\_flag = 0;

}

else

storeLoc\_i += 1;

}

}

if(!strcmp(assem.mnemonic,"JSUB"))

{

if(format4\_flag == 1)

{

mod[mod\_i].loc = assem.loc+1;

mod[mod\_i].val = 5;

mod\_i++;

}

else

{

mod[mod\_i].loc = assem.loc+1;

mod[mod\_i].val = 3;

mod\_i++;

}

}

}

for(i=0; i<mod\_i; i++)

fprintf(fp\_OBJ,"M%06X%02X\n",mod[i].loc,mod[i].val);

if(!strcmp(assem.mnemonic,"END"))

fprintf(fp\_OBJ,"E%06X\n",first\_loc);

fclose(fp\_ASM);

fclose(fp\_LST\_pass2);

fclose(fp\_OBJ);

}

int findSpace(char\* fileStr, int pass\_num)

{

int i, j;

char storeSTR[MAX] = {'\0',};

char store[MAX] = {'\0',};

char temp[MAX] = {'\0',};

int number = 0;

strcpy(storeSTR,fileStr);

oprComma\_flag = 0;

j=0;

for(i=0; (unsigned)i<strlen(fileStr); i++)

{

if(fileStr[i]==' ' || fileStr[i]=='\t')

continue;

temp[j] = fileStr[i];

j++;

}

if(strstr(temp,","))

oprComma\_flag = 1;

if(pass\_num == 1)

{

while(strlen(fileStr) != 0)

{

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

store[i] = '\0';

cut\_space(fileStr);

number++;

j=0;

for(i=0; (unsigned)i<strlen(fileStr); i++)

{

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

return 0;//주석일때 예외처리

if((fileStr[i] == ' ' || fileStr[i] == '\t'))

{

if((number == 2 || number == 3) && oprComma\_flag == 1)

{

store[j] = fileStr[i];

j++;

}

else

break;

}

else

{

store[j] = fileStr[i];

j++;

}

}

if(number == 1)

strcpy(assem.symbol, store);

else if(number == 2)

strcpy(assem.mnemonic, store);

else if(number == 3)

strcpy(assem.operand, store);

else if(number > 3)

{

printf("정상 asm 파일이 아닙니다. 디버깅 할 수 없습니다.\n");

return -1;

}

strcpy(fileStr,fileStr+i);

}

if(number == 2)//symbol이 없을 때

{

strcpy(assem.operand,assem.mnemonic);

strcpy(assem.mnemonic,assem.symbol);

strcpy(assem.symbol,"\0");

}

else if(number == 1)//한 단어이면 mnemonic임

{

strcpy(assem.mnemonic,assem.symbol);

strcpy(assem.symbol,"\0");

}

strcpy(fileStr,storeSTR);

}

else if(pass\_num == 2)

{

while(strlen(fileStr) != 0)

{

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

store[i] = '\0';

cut\_space(fileStr);

number++;

j=0;

for(i=0; (unsigned)i<strlen(fileStr); i++)

{

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

return 0;//주석일때 예외처리

if((fileStr[i] == ' ' || fileStr[i] == '\t'))

{

if((number == 3 || number == 4) && oprComma\_flag == 1)

{

store[j] = fileStr[i];

j++;

}

else

break;

}

else

{

store[j] = fileStr[i];

j++;

}

}

if(number == 1)

{

if(!strcmp(store,"BASE") || !strcmp(store,"END"))

{

strcpy(assem.mnemonic,store);

number += 2;

}//base와 end의 경우에는 lco이 없어서 바로 나오는것 처리 필요

else

assem.loc = strtoul(store,NULL,16);

}

else if(number == 2)

strcpy(assem.symbol,store);

else if(number == 3)

strcpy(assem.mnemonic, store);

else if(number == 4)

strcpy(assem.operand, store);

else if(number > 5)

{

printf("정상 asm 파일이 아닙니다. 디버깅 할 수 없습니다.\n");

return -1;

}

strcpy(fileStr,fileStr+i);

}

if(number == 3)//symbol이 없을 때

{

strcpy(assem.operand,assem.mnemonic);

strcpy(assem.mnemonic,assem.symbol);

strcpy(assem.symbol,"\0");

}

else if(number == 2)//한 단어이면 mnemonic임 + END 와 BASE 일 때는 loc이 없어서 두단어임

{

strcpy(assem.mnemonic,assem.symbol);

strcpy(assem.symbol,"\0");

}

strcpy(fileStr,storeSTR);

}

return 1;

}