과목 명: 시스템프로그래밍

담당 교수 명: 박 운 상

[Assignment 2]

**서강대학교 컴퓨터공학과**

**20161665**

**황세현**

목 차

1. 프로그램 개요
2. 프로그램 설명
   1. 프로그램 흐름도
3. 모듈 정의
   1. 모듈 이름 : main()
      1. 기능
      2. 추가된 변수
   2. 모듈 이름 : func\_type(char \*)
      1. 기능
      2. 추가된 변수
   3. 모듈 이름 : Pow10(int a)
      1. 기능
      2. 추가된 변수
   4. 모듈 이름 : pass1(char \*filename)
      1. 기능
      2. 추가된 변수
   5. 모듈 이름 : if\_directive(FILE\* fp, char \*origin, char\*order, int \*locctr, int \*start)
      1. 기능
      2. 추가된 변수
   6. 모듈 이름 : if\_mnemonic(FILE\* fp, char \*origin, char\*order, int \*locctr, int \*start)
      1. 기능
      2. 추가된 변수
   7. 모듈 이름 : string\_to\_number2(char \*tmp)
      1. 기능
      2. 추가된 변수
   8. 모듈 이름 : add\_no\_to\_symTable(char \*, int)
      1. 기능
      2. 추가된 변수
   9. 모듈 이름 : func\_symbol()
      1. 기능
      2. 추가된 변수
   10. 모듈 이름 : delete\_symbolTable()
       1. 기능
       2. 추가된 변수
   11. 모듈 이름 : pass2(char \*filename)
       1. 기능
       2. 추가된 변수
   12. 모듈 이름 : loc\_write\_to\_file(FILE\* fp,int locctr, int flag)
       1. 기능
       2. 추가된 변수
   13. 모듈 이름 : OP2\_choose\_reg(char \*reg,char \*objcode)
       1. 기능
       2. 추가된 변수
   14. 모듈 이름 : decInt\_to\_hexStr(int a,char \*str, int flag)
       1. 기능
       2. 추가된 변수
   15. 모듈 이름 : find\_symbol(char \*)
       1. 기능
       2. 추가된 변수
4. 전역변수의 정의
   1. symbolNode\* symbolTable[26]
   2. modHead modi
   3. assemble\_num
   4. lastsAssemble
   5. prog\_length
5. 코드
6. 기타
   1. 주의사항
   2. 예외처리(다 쓸 수 없어서 일부만 기입)

# 프로그램 개요

SIC machine과 구분되는 SIC/XE machine의 특징에 대해 학습한 뒤, 이를 기반으로 리스팅 파일과 오브젝트 파일을 작성하는 어셈블러를 구현한다.

# 프로그램 설명

## 프로그램 흐름도

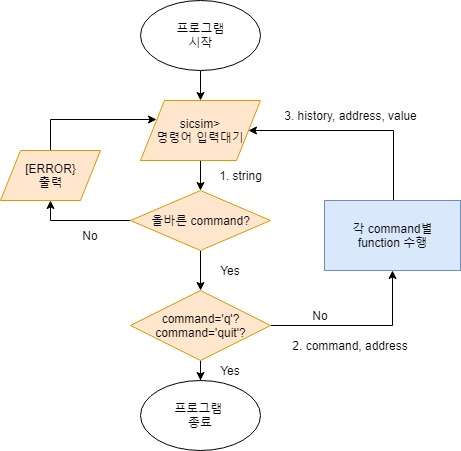


그림 1> 프로그램 흐름도

# 모듈 정의 [project2에 추가되거나 변경된 함수]

## 모듈 이름 : main()

### 기능

사용자의 명령어 입력을 받아 올바른 입력이라면 기능을 수행하도록 함수를 호출하고, 올바른 입력이 아니라면 에러메세지를 띄워준다. 이번 프로젝트에서는 ‘symbol’ 과 ‘type filename’ , ‘assemble filename’ 이 새로운 명령어로 추가되었다.

### 추가된 변수

없음

## 모듈 이름: func\_type(char \*)

### 기능

현재 디렉토리에 존재하는 파일이름을 받아 파일의 내용을 그대로 출력해주는 기능을 수행한다.

### 사용 변수

* FILE \*fp : 인자로 입력된 파일을 열어주는 포인터
* char tmp : 파일에서 받은 문자를 저장할 변수

## 모듈이름: Pow10(int a)

### 기능

인자로 받은 a만큼 10의 제곱승을 계산하여 반환하는 함수이다.

### 사용변수

없음

## 모듈이름: pass1(char \*filename)

### 기능

인자로 전달받은 이름의 .asm파일을 열어 assembly code를 한 줄 씩 처리하는 기능을 수행한다. 만약 중간에 올바르지 않은 code가 존재한다면 에러 코드를 반환하고, 모든 code가 정상적으로 입력되었다면 intermediate file인 ‘pass1.txt’를 생성한다. 이 텍스트 파일은 후에 pass2의 입력파일로 사용된다. 또한 code를 변환하면서 symbol이 존재한다면 이를 바탕으로 symbol table을 형성한다.

### 사용변수

* FILE\* fp\_read, \*fp\_write : 인자로 받은 파일을 열고, ‘pass1.txt’파일을 작성하는 파일 포인터
* char sentence[100],order[30],name[20], symbol[20] : sentence에는 fgets로 받아온 1문장이 저장된다. 모듈을 이용하여 sentence를 order, mnemonic, name, symbol등으로 분리해내고 이들은 각각 order[30], name[20], symbol[20] 에 저장된다.
* int start, prog\_start : prog\_start에는 프로그램 시작 locctr이 저장되고, start에는 문장의 시작 idx를 저장한다
* int locctr : location counter 정보를 저장하는 변수
* int err\_line, cur\_line : 현재 위치하는 라인과, 에러가 발생한 라인을 저장하는 변수
* int no\_start : START directive가 존재하는지를 판단하는 flag. 존재한다면 no\_start=False;

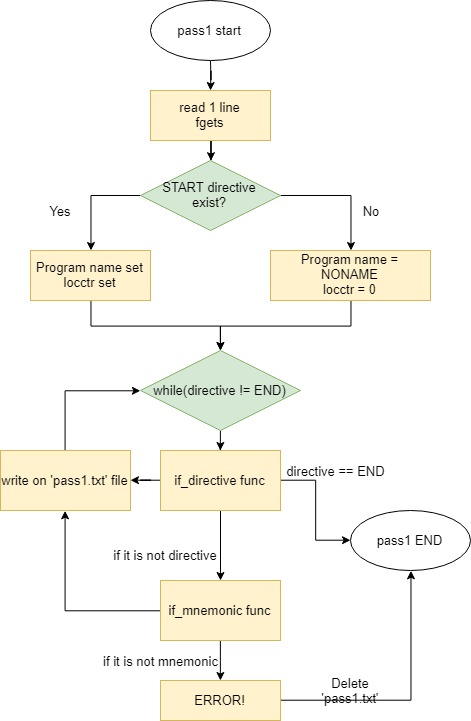


Figure 1 pass1 flow chart

[pass1 알고리즘]

pass1 알고리즘은 맨 처음 입력으로 들어온 .asm 파일을 읽어들여서 그 다음 pass2함수의 input으로 입력될 intermediate file을 생성한다. 우선 .asm 파일의 첫 줄을 읽어서 공백 여부, START directive의 존재 유무를 파악한다. 만약 공백 줄이라면 공백 줄이 아닐 때까지 읽어들이고, START directive가 존재하지 않는다면 program name=NONAME, locctr =0; 으로 각각 초기화 시킨 다음 계속해서 읽어들인다. pass1알고리즘은 .asm 파일에 END directive가 존재한다는 가정하에 END directive를 만날 때까지 계속해서 프로그램을 읽어들이므로 만약 .asm파일에 END directive가 존재하지 않는다면 에러가 발생하게 된다. pass1 의 주된 기능은 directive에 대한 예외처리, mnemonic opcode의 저장, symboltable의 생성이며 만약 directive를 검사하는 중에 에러가 발생하면 intermediate file과 symboltable을 삭제한 뒤, 프로그램을 종료한다. mnemonic에 대한 예외처리는 pass2에서 진행하도록 하였다. 만약 모든 directive가 정상적이라면 pass2함수로 넘어간다.

## 모듈이름: if\_directive(FILE\* fp,char \*origin, char \*order, int \*locctr, int \*start)

### 기능

문자열 전체를 인자로 받아 해석할 수 있는 단위로 분리해낸 후, 분리된 문자열이 directive인지 아닌지를 판단하는 함수이다. 만약 directive라면 1을 반환하고 그렇지 않다면 -1, directive 해석 중 에러가 발생한다면 0을 반환한다.

### 사용변수

* int hexa\_flag : directive뒤에 존재하는 문자열이 hexa decimal인지 아닌지를 판단하는 변수
* int num, I, len : len에는 directive의 길이가 저장되고, 나머지 변수는 함수의 부수적인 연산에 쓰인다.

## 모듈이름: if\_mnemonic(FILE\* fp,char \*origin, char \*order, int \*locctr, int \*start)

### 기능

문자열 전체를 인자로 받아 해석할 수 있는 단위로 분리해낸 후, 분리된 문자열이 mnemonic인지 아닌지를 판단하는 함수이다. 만약 mnemonic이라면 1을 반환하고 그렇지 않다면 -1을 반환한다. mnemonic에 대한 에러판단은 pass2에서 진행한다.

### 사용변수

* char format : 파일에서 읽어온 mnemonic의 형식을 저장하는 변수
* unsigned int opcode : mnemonic의 opcode를 부호가 없는 int형 변수에 저장한다.
* int plus, len : len에는 mnemonic의 길이를 저장하고, plus는 format 4의 mnemonic을 판단하는 변수이다.

## 모듈이름: string\_to\_number2(char \*temp)

### 기능

10진수 문자열을 10진수 정수로 바꿔서 반환하는 함수이다.

### 사용변수

* int len : 인자로 받은 string 의 길이를 저장하는 변수
* int I, sum : 10진수 정수를 계산하기 위해 사용되는 변수.

## 모듈이름: add\_node\_to\_symTable(char \*,int )

### 기능

pass1 에서 찾은 symbol 을 symbolTable에 symbol의 location counter과 함께 저장하는 함수이다.

### 사용변수

* int idx : symbol이 저장될 symbolTable의 idx를 담고있는 변수

## 모듈이름: func\_symbol()

### 기능

현재 존재하는 symbolTable을 출력해주는 함수이다. 만약 symbolTable이 존재하지 않는다면 아무것도 출력되지 않는다.

### 사용변수

* int digit : symbol의 주소가 16진수로 몇 자리에 해당되는지 저장하는 변수
* symbolNode\* tmp : linked list로 구현된 symboltable을 출력하기위해 사용되는 symbolNode 포인터

## 모듈이름: delete\_symbolTable()

### 기능

새로운 assemble 명령어가 입력될 때, 기존에 존재하는 symbol table을 삭제한다. (동적으로 할당된 memory의 free를 담당하는 함수.)

### 사용변수

* symbolNode\* del : linked list 로 구현된 symbolTable을 따라가면서 symbolNode들의 삭제를 담당한다.

## 모듈이름: pass2(char \*filename)

### 기능

pass1이 정상적으로 종료되었을 경우, 생성된 pass1.txt파일을 입력으로 받아 obj file과 lst file을 생성한다. 만약 중간에 에러코드가 존재할 경우 생성된 obj, lst, txt파일을 모두 삭제한뒤 -1을 반환하고, 정상적으로 종료될 경우 1을 반환한다. 또한 pass1에서 아직 처리되지 않은 mnemonic에 대한 예외처리를 담당한다. 코드 처리 과정에서 nixbpe 비트 설정, modification record 등 sic/xe machine의 핵심적인 기능을 수행하는 함수이다.

### 사용변수

* FILE \*fp\_origin, fp\_pass1 : r모드로 파일을 열어준다. 기존의 .asm와 pass1.txt파일의 정보를 읽어들인다.
* FILE \* fp\_lst, \*fp \_obj : 해석된 코드를 바탕으로 lst 파일과 obj파일을 작성을 담당
* char file1[30], file2[30], name[20] : .lst 파일과 .obj 그리고 프로그램 이름을 저장
* char sentence[120], tmp[50], tmp2[50], tmp3[50] : .asm파일에서 fgets로 받아온 한 문장을 sentence에 저장하고, 이를 의미있는 문자열 단위로 분리하여 각각 tmp, tmp2, tmp3,등에 저장한다.
* char objcode[61], objrecord[30][61] : objcode에는 생성된 object code를 저장하고, .obj file에 쓰여질 코드 정보는 2차원 배열인 objrecord 에 저장된다.
* int cur\_line, err\_line : 현재 라인 정보와 에러가 발생한 라인 정보가 저장된다.
* int nixbpe[6] : nixbpe 비트 정보가 0, 1로 저장된다.
* int flag, objflag : 프로그램 흐름 상 필요한 flag 변수
* what\_type type : 입력된 명령어가 OPCODE1,2,3,4 와 END, START directive 중 어느 type에 해당하는지를 알려주는 enum type
* modeNode\* newNode, \*tmpNode : modification record의 기록을 담당하는 노드 포인터

[pass2 알고리즘]

pass1이 정상적으로 종료되어 1을 반환할때만 pass2가 실행된다. pass2는 입력으로 pass1에서 생성된 intermediate file을 받으며 intermediate file에는 현재 줄의 order에 대한 정보가 정수로 저장되어있다. ex) BYTE=0, WORD=1, RESB=2;(총 13가지 type이 존재한다. ) pass2의 주된 역할은 이렇게 type이 구분지어진 명령어에 대해서 알맞은 수행을 통해 object code를 생성하는 것이다. 공백 줄, white space, no START directive, directive error등의 예외 처리는 모두 pass1에서 완료되었으므로 pass2에서는 mnemonic에 대한 예외처리만을 진행하면 된다. 만약 mnemonic에 대한 검사 중 에러가 발생하였다면 에러코드를 화면에 출력하고, 모든 생성된 파일을 삭제한 뒤 프로그램을 종료한다. 마지막 줄까지 검사가 완료되었다면 .lst .obj 파일이 정상적으로 생성된 상태로 프로그램이 종료된다.

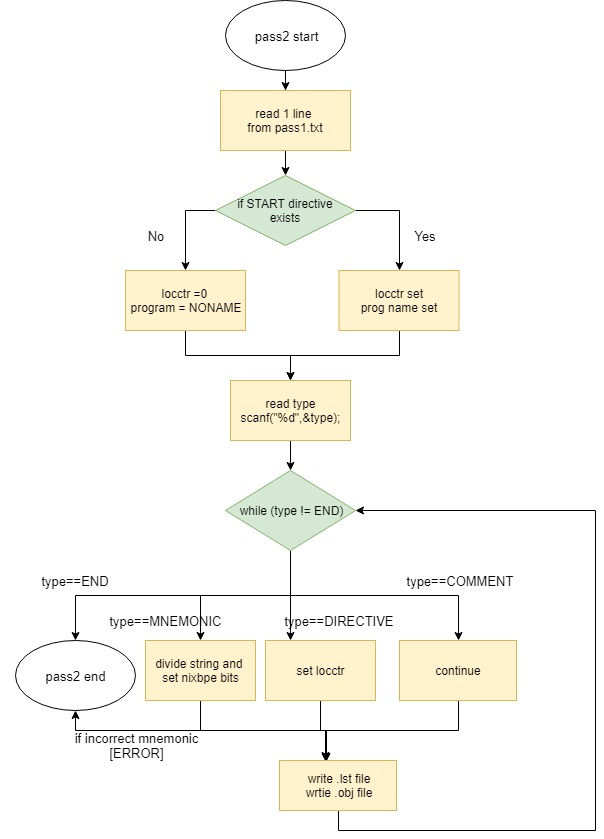


Figure 2 pass2 flow chart

## 모듈이름: loc\_write\_to\_file(FILE\* fp,int locctr, int flag)

### 기능

인자로 입력받은 locctr을 flag자리의 16진수로 파일에 입력하는 기능을 담당한다.

### 사용변수

없음

## 모듈이름: OP2\_choose\_reg(char \*reg, char \*objcode)

### 기능

인자로 받은 register을 검사하고 이를 objcode에 덧붙이는 기능을 수행한다. 만약 올바른 register라면 1을 반환하고, 올바르지 않다면 -1을 반환한다. opcode format2를 처리할 때 이용된다.

### 사용변수

* char tmp[2] : register가 저장될 변수
* Register r : register 정보를 정수로 담고있는 enum형 변수

## 모듈이름: decInt\_to\_hexStr(int a, char \*str, int flag)

### 기능

10진수 정수를 입력으로 받아 16진수 문자열로 반환하는 함수. 인자로 전달받은 문자열에 16진수를 담아 반환한다.

### 사용변수

* int len : 입력받은 수가 16진수로 몇 자리 수 인지를 저장하는 변수
* int I,tmp, tmp2 : 기타 부수적인 연산에 쓰일 변수

## 모듈이름: find\_symbol(char \*symbol)

### 기능

인자로 전달받은 symbol이 전역으로 선언된 symbolTable에 존재하는지 확인하고 존재한다면 location counter을 반환한다.

### 사용변수

* symbolNode\* : linked list로 구현된 symbolTable을 탐색하기 위해 사용되는 노드 포인터
* int find : symbol이 존재하는지 아닌지를 True/False로 저장한다.

# 전역 변수 정의

## symbolNode\* symbolTable[26]

hashTable형태로 구현된 symbolTable.

## modHead modi

modification record저장을 담당한다.

## int assemble\_num

assemble 명령이 성공적으로 수행된 횟수를 기록한다.

## int lastAssemble

마지막으로 수행된 assemble 명령의 성공여부를 담고있다.

## int prog\_length

.obj file에 기록될 program length를 저장할 변수이다.

# 코드

<주된 assembler 기능을 담당하는 함수>

1. **pass1**

/\* 에러 발생시 -1반환, 정상 종료시 1반환 \*/

int pass1(char \*filename){

FILE \*fp\_read,\*fp\_write;

char sentence[100],order[30],name[20],temp[20],symbol[20];

int start=0,locctr=0,err\_line=-1,cur\_line=0;

int no\_start=False,res,symbol\_loc; //첫째줄에 start directive의 존재유무 판단 flag

int prog\_start,i=0;

if(!(fp\_read=fopen(filename,"r"))){

printf("[ERROR]No files having such a name [%s]\n",filename);

return -1;

}

fp\_write=fopen("pass1.txt","w");

/\* 1. .asm 파일 첫번째줄 읽어들이기 \*/

fgets(sentence,sizeof(sentence),fp\_read);

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

get\_string(sentence,order,&start,0); //program 이름 추출

get\_string(sentence,temp,&start,0); //start address 추출

//program name & start address 설정장

if(!strcmp(temp,"START\0")){

strcpy(name,order);

get\_string(sentence,order,&start,0);

locctr=string\_to\_number(order,1);

cur\_line=1;

}

else{

strcpy(name,"NONAME\0");

no\_start=True;

}

prog\_start=locctr; //locctr 처음 주소 저장

fprintf(fp\_write,"%X %s\n",locctr,name);

/\* 2. END directive에 도달할때까지 어셈블링 과정 반복\*/

while(1){ //error line발견시 또는 END 도달시 prog 종료

cur\_line++;

if(!no\_start)

start=0;

symbol\_loc=locctr;

fprintf(fp\_write,"%X ",locctr);

/\* a)문장의 첫번째 order 처리하기 : symbol 또는 mnemonic 또는 END\*/

if(!no\_start) //start directive가 없으면 뛰어넘기

{

fgets(sentence,sizeof(sentence),fp\_read);

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

get\_string(sentence,order,&start,0);

}

if(!strcmp(order,"END\0")){ //종료 조건(type 4)

fprintf(fp\_write,"4\n");

prog\_length=locctr-prog\_start; //prog 길이 저장(.obj에 쓰임)

break;

}

else if(!strcmp(order,"BASE\0")){ //base directive (type 10)

fprintf(fp\_write,"10 ");

//나머지 뒤에있는 문자열 그대로 저장

for(i=start;sentence[i]!='\0';i++)

fprintf(fp\_write,"%c",sentence[i]);

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

continue;

}

else if(!strcmp(order,"NOBASE\0")){ //nobase directive (type 11)

fprintf(fp\_write,"11\n");

continue;

}

else if(order[0]=='.'){ //comment line (type 9)

fprintf(fp\_write,"9\n");

continue;

}

//mnemonic인지 아닌지를 검사

if(if\_mnemonic(fp\_write,sentence,order,&locctr,start)==1){ //뉴모닉이라면

no\_start=False;

continue;

}

strcpy(symbol,order); //임시로 symbol에 저장

/\* b)문장의 두번째 order 처리하기(여기서 symbol인지 아닌지를 판별)\*/

if(!no\_start)

get\_string(sentence,order,&start,0); //문자열 1개더 읽음

else

strcpy(order,temp);

//mnemonic인지 아닌지를 검사

if(if\_mnemonic(fp\_write,sentence,order,&locctr,start)==1){ //뉴모닉이라면

no\_start=False;

}

//에러가 발생할 경우

else if((res=if\_directive(fp\_write,sentence,order,&locctr,&start))!=1){

printf("[ERROR]Invalid directive (line %d)\n",cur\_line\*5);

////////////////////에러코드 수정필요!!!!!

err\_line=cur\_line;

fclose(fp\_write);fclose(fp\_read);

remove("pass1.txt");

return -1;

}

/\* c)이 모든 과정을 거쳐왔다면 처음의 order은 symbol이다. \*/

add\_node\_to\_symTable(symbol,symbol\_loc);

no\_start=False;

}

fclose(fp\_read);fclose(fp\_write); //만약 중간에 에러발생시 pass1.txt지워주기

return 1;

}

1. **pass2**

int pass2(char \*filename){

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

FILE\* fp\_pass1=fopen("pass1.txt","r");

FILE\*fp\_lst,\*fp\_obj;

char file1[30],file2[30],name[20];

char sentence[120]; //\*.asm 파일의 1문장씩 담아낼 공간

char tmp[50],tmp2[50],tmp3[50];;

char objcode[61], objrecord[30][61]; //임시로 obj code를 담을 공간

int len,cur\_line=1,locctr,format=-1,opcode,err\_line=-1;

int valid=0,byte=0,objLoc=0; //objLoc=obj에서 시작 주소 담음

int nixbpe[6]={0}; //nixbpe값을 저장하는 배열

int hexa,i,j,start=0,prog\_start=0;

int flag=True,objflag=True; //flag는 obj코드 적는 시기를 알려주는 변수

int p=True,b=False,baseLoc; //p bit를 검사하는 flag/BASE가 사용가능조사 flag

int pc,disp,ni=0,xbpe=0;

char c;

what\_type type=0; //type check하는 enum형 변수

modeNode \*newNode,\*tmpNode;

/\* .lst 와 .obj 파일 생성하는 과정\*/

len=strlen(filename);

strncpy(file1,filename,len-3);strncpy(file2,filename,len-3);

file1[len-3]='\0';file2[len-3]='\0';

strcat(file1,"lst");strcat(file2,"obj");

fp\_lst=fopen(file1,"w");fp\_obj=fopen(file2,"w");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 시작 주소 & 프로그램 이름 받아오기 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

fscanf(fp\_pass1,"%x %s\n",&locctr,name);

objLoc=locctr;prog\_start=locctr;

/\* .lst .obj 첫번째 줄 작성 \*/

//1. .lst 파일

fprintf(fp\_lst,"%3d\t",cur\_line\*5);

if(!strcmp(name,"NONAME\0") && locctr==0){ //START directive 가 없었다면

loc\_write\_to\_file(fp\_lst,locctr,4);

fprintf(fp\_lst,"\t%-6s %X\n",name,locctr);

}

else{ //START directive 가 있었다면

loc\_write\_to\_file(fp\_lst,locctr,4);

fprintf(fp\_lst,"\t%-6s START %X\n",name,locctr);

fgets(sentence,sizeof(sentence),fp\_origin);//기존파일 첫째줄 제거

}

//2. .obj 파일

fprintf(fp\_obj,"H%-6s",name);

loc\_write\_to\_file(fp\_obj,locctr,6);

loc\_write\_to\_file(fp\_obj,prog\_length,6);

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

cur\_line++;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*반복적으로 명령어 처리 시작(MAIN)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

while(type!=END && err\_line==-1){ //END directive 가 나오기 전까지 반복

//변수 초기화

format=0;objcode[0]='\0';

/\* obj file첫 주소를 저장(col 2-7)\*/

if(objflag && type!=RESW && type!=RESB && type!=COMMENT && type!= BASE && type!=NOBASE){

objLoc=locctr;

objflag=False;

}

/\* source code(.asm) 받아오기 \*/

fgets(sentence,sizeof(sentence),fp\_origin);

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

len=strlen(sentence);

for(i=len;i<30;i++)

sentence[i]=' ';

if(len<30)

sentence[30]='\0';

/\* locctr 과 type 받아오기 \*/

fscanf(fp\_pass1,"%X %d",&locctr,&i);type=i;

//1. 먼저 lst파일에 기록

if(type==BASE || type==NOBASE || type==COMMENT || type==END){

fprintf(fp\_lst," \t \t");

fprintf(fp\_lst,"%s\n",sentence);

}

else{

fprintf(fp\_lst,"%3d\t",cur\_line\*5);

loc\_write\_to\_file(fp\_lst,locctr,4);

fprintf(fp\_lst,"\t%s\t\t",sentence); //source code까지 기록

}

/\* type 별로 각각 처리하기 \*/

switch(type) {

case BYTE : fscanf(fp\_pass1,"%s",tmp);

fscanf(fp\_pass1,"%s",objcode);

if(!strcmp(tmp,"X\0")){ //hexa일때

//////////////////////////////\* 만약 최대 1byte수를 가져야한다면(가정) \*/

if(strlen(objcode)==1) //길이는 1또는 2라고 가정

{

objcode[1]=objcode[0];

objcode[0]='0';objcode[2]='\0';

}

format=1;

}

else /////////////////문자열 범위도 지정되어있다고 가정

format=strlen(objcode)/2;

fprintf(fp\_lst,"%s\n",objcode);

break;

case WORD : fscanf(fp\_pass1,"%s",tmp);

len=strlen(tmp);

objcode[0]='0';objcode[1]='0';objcode[2]='0';

objcode[3]='0';objcode[4]='0';objcode[5]='0';objcode[6]='\0';

memcpy(objcode+(6-len),tmp,len);

fprintf(fp\_lst,"%s\n",objcode);

format=3;

break;

case RESB : fprintf(fp\_lst,"\n");

break;

case RESW : fprintf(fp\_lst,"\n");

break;

case END : //할거 없음

break;

case OP1 : fscanf(fp\_pass1,"%s",objcode);i=0;

while((c=fgetc(fp\_pass1))!='\n')

tmp[i++]=c; //tmp는 최대 50자.

tmp[i]='\0';

if(test\_tail(tmp,0)!=1){ //에러처리

err\_line=cur\_line;

//too many parameters

break;

}

fprintf(fp\_lst,"%s\n",objcode);/////////////////아닐수도///////

format=1;break;

case OP2 : fscanf(fp\_pass1,"%s",objcode);i=0; //opcode저장

while((c=fgetc(fp\_pass1))!='\n') //뒤에 있는 문자열 추출

tmp[i++]=c;

tmp[i]='\0';start=0;

/\* reg개수 1개, 2개로 나뉨 \*/

if(!strcmp(objcode,"B4\0") || !strcmp(objcode,"B8\0")){

if(get\_string(tmp,tmp2,&start,0)==1) //오류발생

err\_line=cur\_line;

//incorrect opcode

else if(!test\_tail(tmp,start))

err\_line=cur\_line; //오류발생

//too many paramaters

if(OP2\_choose\_reg(tmp2,objcode)!=1) //reg오류

err\_line=cur\_line;

//invalid register %s, tmp2

objcode[3]='0';

}

else{

if(get\_string(tmp,tmp2,&start,0)==1) //첫번째 reg

err\_line=cur\_line;

else if(OP2\_choose\_reg(tmp2,objcode)!=1)

err\_line=cur\_line;

if(get\_string(tmp,tmp2,&start,1)==1)

err\_line=cur\_line;

else if(!test\_tail(tmp,start))

err\_line=cur\_line; //오류발생

if(OP2\_choose\_reg(tmp2,objcode)!=1) //reg범위 오류

err\_line=cur\_line;

}

//////////////////어떤 에러인지 에러코드 넣어주기///////////////

if(err\_line==cur\_line) break;

objcode[4]='\0';format=2;

fprintf(fp\_lst,"%s\n",objcode);

break;

case OP3 :

case OP\_PLUS :

/\* nixbpe 값 초기화 \*/

for(i=0;i<6;i++){nixbpe[i]=0;}

if(type==OP\_PLUS){

nixbpe[5]=1;format=4;

}

else

format=3;

/\* opcode 및 문자열 추출 \*/

fscanf(fp\_pass1,"%s",objcode);i=0; //opcode저장

while((c=fgetc(fp\_pass1))!='\n') //뒤에 있는 문자열 추출(tmp)

tmp[i++]=c;

tmp[i]='\0';start=0;

if(strlen(objcode)==1){ //opcode 1자리->2자리 보정

objcode[2]=objcode[1];

objcode[1]=objcode[0];

objcode[0]='0';

}

if(!strcmp(objcode,"4C\0")){ //RSUB의 경우!

if(!test\_tail(tmp,start))

err\_line=cur\_line; //too many parameters

else if(type==OP3)

strcpy(objcode,"4F0000\0");

else if(type==OP\_PLUS)

strcpy(objcode,"4F000000\0");

}

else{ //나머지 3/4format 처리

if(get\_string(tmp,tmp2,&start,0)==1) //첫번째 문자열추출

err\_line=cur\_line; //에러코드

//0. n,i bit 설정하기

if(tmp2[0]=='@' || tmp2[0]=='#'){

if(tmp2[0]=='@')nixbpe[0]=1; //indirect

else if(tmp2[0]=='#')nixbpe[1]=1; //immediate

//flag를 set 했으므로 @나 #를 제거

len=strlen(tmp2);

for(i=1;i<len;i++)

tmp2[i-1]=tmp2[i];

tmp2[len-1]='\0';

//ex)@4096, X 같은 경우 예외처리

if(!test\_tail(tmp,start)){

err\_line=cur\_line; break;

}

}

else //simple

nixbpe[0]=nixbpe[1]=1;

//1. 만약 뒤에 상수가 온다면 ex)#4096 --> p또는 b비트 설정할 필요 X

flag=True;len=strlen(tmp2);

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

if(!(tmp2[i]>='0' && tmp2[i]<='9'))

flag=False;

}

//상수가 맞지만 뒤에 뭔가 더 있을때

if(!test\_tail(tmp,start) && flag){

if(get\_string(tmp,tmp3,&start,1)==1){// 에러 발생

err\_line=cur\_line; break;

}

else if(!test\_tail(tmp,start)){ //에러

err\_line=cur\_line; break;

}

else if(!strcmp(tmp3,"X\0"))

nixbpe[2]=1;

}

if(flag){ //만약 뒤에 오는 수가 상수라면

hexa=string\_to\_number2(tmp2);

//만약 4096이상의 수가 왔다면 에러

if(hexa>=4096 && type==OP3){ //format3초과 범위의 수일때

err\_line=cur\_line; break;

}

decInt\_to\_hexStr(hexa,tmp3,1); //16진수 문자열로 변경후

for(i=2;i<10;i++)

objcode[i]='0';//objcode에 이어붙이기

if(type==OP3){ //format3 일때

if(hexa<Pow16[1] && hexa>=0)

memcpy(objcode+5,tmp3,1);

else if(hexa<Pow16[2] && hexa>=Pow16[1])

memcpy(objcode+4,tmp3,2);

else if(hexa<Pow16[3] && hexa>=Pow16[2])

memcpy(objcode+3,tmp3,3);

objcode[6]='\0';

}

else{ //format4 일때

if(hexa<Pow16[1] && hexa>=0)

memcpy(objcode+7,tmp3,1);

else if(hexa<Pow16[2] && hexa>=Pow16[1])

memcpy(objcode+6,tmp3,2);

else if(hexa<Pow16[3] && hexa>=Pow16[2])

memcpy(objcode+5,tmp3,3);

else if(hexa<Pow16[4] && hexa>=Pow16[3])

memcpy(objcode+4,tmp3,4);

else if(hexa<Pow16[5] && hexa>=Pow16[4])

memcpy(objcode+3,tmp3,5);

objcode[8]='\0';

}

}

//2. symbol이 온다면

else{ //추가해주기 //뒤에오는게 상수가 아닐때

/\* indexed bit을 설정하는 과정 \*/

if((i=find\_symbol(tmp2))==-1){ //존재하지 않는 sym

err\_line=cur\_line; break; //no existing sym

//symbol 이름도 적어주면 더욱이 좋음

}

else if(!test\_tail(tmp,start)){ //symbol table에 존재하지만 뒤에 뭐가 있을때

if(get\_string(tmp,tmp2,&start,1)==1){

err\_line=cur\_line; break; //에러 발생

}

else if(!test\_tail(tmp,start)){ //에러 발생

err\_line=cur\_line; break;

}

else if(strcmp(tmp2,"X\0")!=0){ //에러 발생

err\_line=cur\_line; break; //ex)BUFFER, Y

}

else

nixbpe[2]=1; //x 비트 설정

}

/\* p와 b bit을 설정하는 과정, i에 symbol값 저장 \*/

if(type==OP3){ //format3

pc=locctr+3;

if((disp=i-pc)>=-2048 && disp<2048)

nixbpe[4]=1;

else if(b && (disp=i-baseLoc)>=0 && disp<4096)

nixbpe[3]=1;

else{ //p도 b도 아닐때 에러

err\_line=cur\_line; break;

}

if(disp<0) //disp<0이하의 수일때

decInt\_to\_hexStr(disp\*-1,tmp3,-1);

else

decInt\_to\_hexStr(disp,tmp3,1);

/\* objcode의 3,4,5 자리에 문자열 복사하기 \*/

if(disp<0){

disp \*= -1;

for(i=3;i<6;i++) objcode[i]='F';

}

else{

for(i=2;i<6;i++) objcode[i]='0';//objcode에 이어붙이기

}

if(disp<16)

memcpy(objcode+5,tmp3,1);

else if(disp<256)

memcpy(objcode+4,tmp3,2);

else if(disp<4096)

memcpy(objcode+3,tmp3,3);

objcode[6]='\0';

}

else{ //format4

/\*modification record를 위한 노드 생성\*/

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

newNode->next=NULL;

newNode->addr=locctr+1;

if(modi.head==NULL)

modi.head=newNode;

else{

tmpNode=modi.head;

while(tmpNode->next!=NULL)

tmpNode=tmpNode->next;

tmpNode->next=newNode;

}

modi.num++;

/\* objcode 저장하기\*/

decInt\_to\_hexStr(i,tmp3,1);

for(j=2;j<10;j++)

objcode[j]='0';//objcode에 이어붙이기

if(i<Pow16[1])

memcpy(objcode+7,tmp3,1);

else if(i<Pow16[2])

memcpy(objcode+6,tmp3,2);

else if(i<Pow16[3])

memcpy(objcode+5,tmp3,3);

else if(i<Pow16[4])

memcpy(objcode+4,tmp3,4);

else if(i<Pow16[5])

memcpy(objcode+3,tmp3,5);

objcode[8]='\0';

}

}

/\* n,i,x,b,p,e, 비트에 따라 objcode 수정해주기 \*/

//1. n,i 계산하기

c=objcode[1];

if(c>='0' && c<='9') ni=c-'0';

else ni=c-'A'+10;

if(nixbpe[0] && nixbpe[1]) //simple

ni += 3;

else if(nixbpe[0] && !nixbpe[1]) //@indirect

ni += 2;

else if(!nixbpe[0] && nixbpe[1]) //#immediate

ni += 1;

if(ni>=10)

objcode[1]='A'+ni-10;

else

objcode[1]='0'+ni;

//2. x b p e 계산하기

xbpe=0;

if(nixbpe[2])

xbpe += 8;

if(nixbpe[3])

xbpe += 4;

if(nixbpe[4])

xbpe += 2;

if(nixbpe[5])

xbpe += 1;

if(xbpe>=10)

objcode[2]='A'+xbpe-10;

else

objcode[2]='0'+xbpe;

}

fprintf(fp\_lst,"%s\n",objcode);

break;

case COMMENT : //할거 없음

break;

case BASE : b=True;i=0;

while((c=fgetc(fp\_pass1))!='\n') //뒤에 있는 문자열 추출(tmp)

tmp[i++]=c;

tmp[i]='\0';start=0;

if(get\_string(tmp,tmp2,&start,0)==1){

err\_line=cur\_line; break; //에러발생

}

else if(!test\_tail(tmp,start)){

err\_line=cur\_line; break; //에러 발생

}

else if((i=find\_symbol(tmp2))==-1){ //BASE추출

err\_line=cur\_line; break;

}

else

baseLoc=i; //BASE 주소를 저장(int)

break;

case NOBASE : b=False;

break;

default : printf("[ERROR]Invalid type error\n"); break;

}

////////////////에러발생시 파일삭제(임시)///////////////////////////

if(err\_line!=-1){

printf(" (line %d)\n",cur\_line\*5);

fclose(fp\_pass1);fclose(fp\_origin);fclose(fp\_lst);fclose(fp\_obj);

remove(file1);remove(file2);remove("pass1.txt");

return -1;

}

//2. .obj 파일에 기록하기 위해 정보를 수집

//.obj에 기록해야한다면

if((valid>=30 ||byte+format>30 || type==RESW || type==RESB || type==END)){

objflag=True;

if(valid\*byte!=0){ //objrec에 아무정보도 없다면 기록X

fprintf(fp\_obj,"T"); //RESW 나 RESB때 이러한 경우 발생

loc\_write\_to\_file(fp\_obj,objLoc,6);

loc\_write\_to\_file(fp\_obj,byte,2);

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

fprintf(fp\_obj,"%s",objrecord[i]);

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

valid=0;byte=0;

if(type!=RESW && type!=RESB && type!=END && objcode[0]!='\0'){

strcpy(objrecord[valid++],objcode);

byte+=format;

}

}

if(type==END){ //E&M record 기록하고 종료

if(modi.num!=0){ //메모리 free와 동시에 기록

while(modi.head){

tmpNode=modi.head;

modi.head=modi.head->next;

fprintf(fp\_obj,"M");

loc\_write\_to\_file(fp\_obj,tmpNode->addr,6);

fprintf(fp\_obj,"05\n");

free(tmpNode);

}

}

fprintf(fp\_obj,"E");

loc\_write\_to\_file(fp\_obj,prog\_start,6);

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

}

}

else if(type!=BASE && type!=NOBASE && type!=COMMENT){

//obj파일 작성할필요 없다면 정보담아놓기(BASE,NOBASE,COMMENT제외)

strcpy(objrecord[valid++],objcode);

byte += format;

}

cur\_line++;

if(type==END) break;

}

printf("output file : [%s], [%s]\n",file1,file2);

fclose(fp\_origin);fclose(fp\_pass1);fclose(fp\_lst);fclose(fp\_obj);

return 1;

}

# 기타

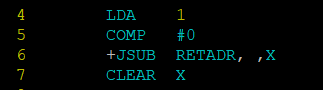
6.1.주의 사항

* + 1. type[filename] 처럼 filename을 입력해야 하는 경우, filename의 최대 길이를 30자로 설정했습니다.
    2. .asm 파일을 한 줄씩 읽어들일 때 asm code 한 줄 당 최대 길이를 120bytes로 설장했습니다.
    3. BYTE directive 에서 X’hexa’ 명령시 hexa decimal은 최대 6half bytes로 설정했습니다.
    4. BYTE directive 에서 C’string’ 명령시 string의 최대 길이를 20자로 설정했습니다.

6.2 예외처리 (다 쓸 수 없어서 일부만 기입)

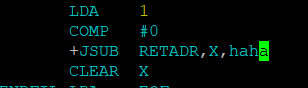
- 올바르지 않은 입력(추가 인자가 존재, comma가 존재 등)

[comma 오류]



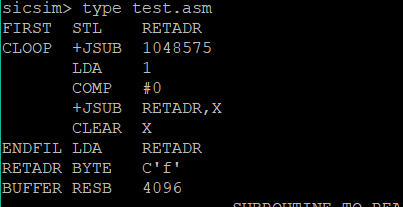


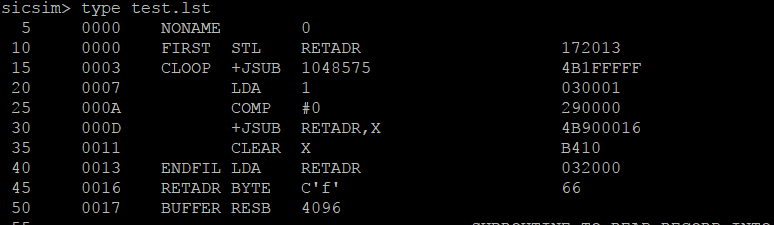
[추가 인자 존재]





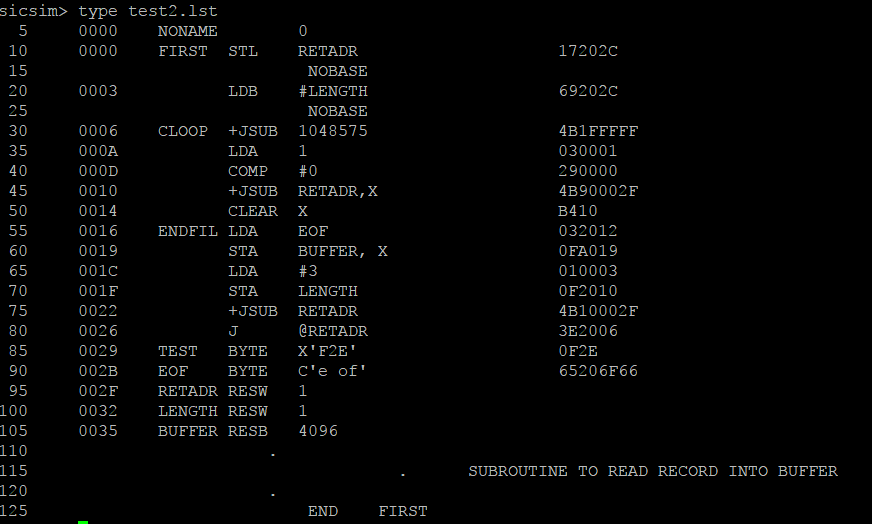
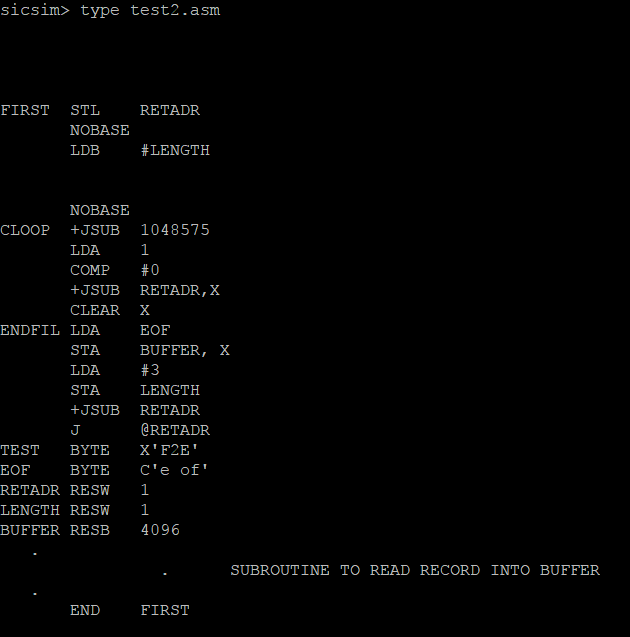
- start directive가 없는 경우 🡪 program name = NONAME, locctr =0





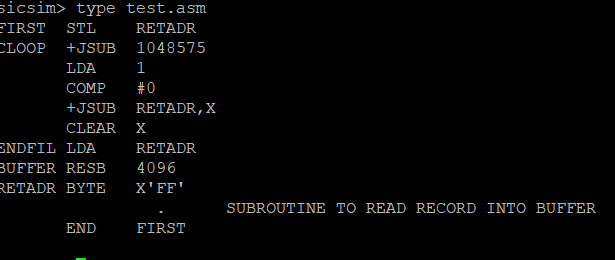
- 공백 줄이 존재할 경우(첫째, 중간)

[.lst 파일이 정상적으로 출력됨을 확인 가능]



- 형식 오류(format 1, format 2, format3, format 4)

[ pc, base relative mode가 모두 불가할 때]



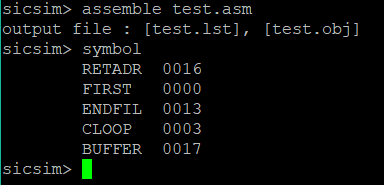


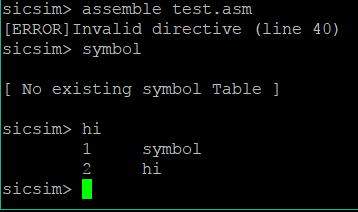
[immediate 또는 indirect에서 index의 사용]





- symbol table (가장 최근의 symbol table을 저장. 에러 발생시에도 history에 추가)





- 존재하지 않는 symbol

