**과목명: 시스템프로그래밍**

**홀수반**

**<<Project #1>>**

**서강대학교 컴퓨터공학부**

**20181593**

**계인혜**

목 차

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

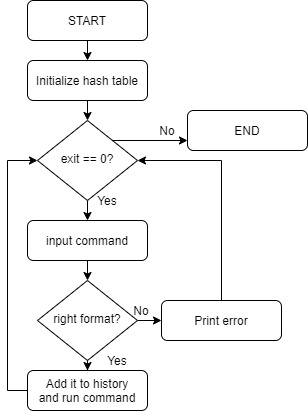
**6 테스트 케이스**

**1. 프로그램 개요**

프로젝트1에서 구현한 shell에 assemble 기능을 추가하는 프로그램으로, SIC/XE의 assembly program source 파일을 입력 받아서 object 파일을 생성하고, 어셈블리 과정 중 생성된 symbol table과 결과물인 object 파일을 볼 수 있는 기능을 제공한다. 이는 교재의 2.2 까지 설명된 SIC/XE 어셈블러의 기능을 구현하는 것을 원칙으로 한다.

**2. 프로그램 설명**

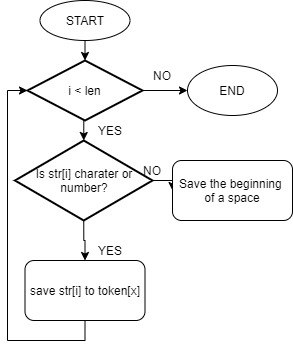
2.1 프로그램 흐름도



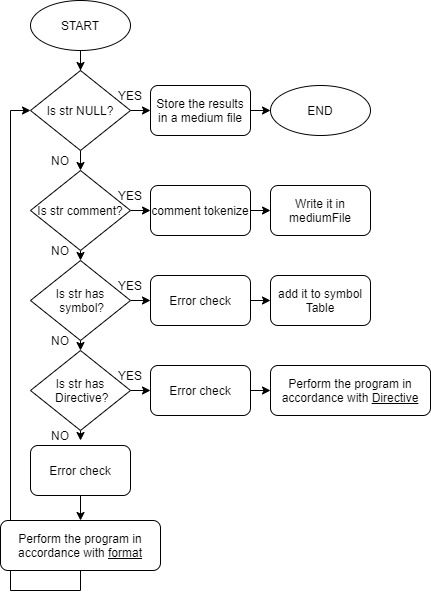
2.2 세부 프로그램 흐름도

① asm\_tokenize -> asm 파일을 한 줄씩 읽어 공백과 콤마, 탭, 엔터를 기준으로 나눠

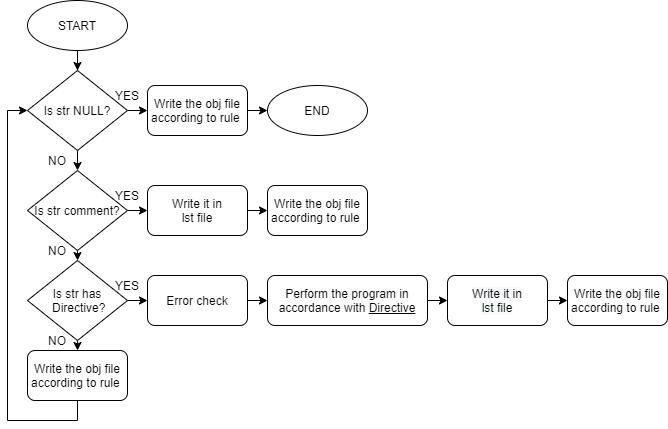
단어로 저장하는 함수

****

② makeSIC\_code -> asm 파일을 obj코드로 바꾸기 전에 sic/xe 코드로 바꿔 저장하는 함수



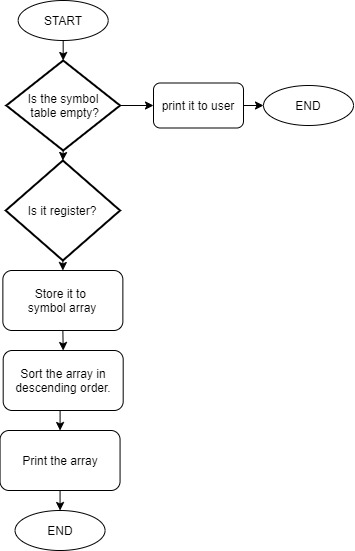
③ makeObjFile -> 이전에 만들었던 중단 단계의 파일(sic/xe)을 이용해 obj 코드를 만드는 함수

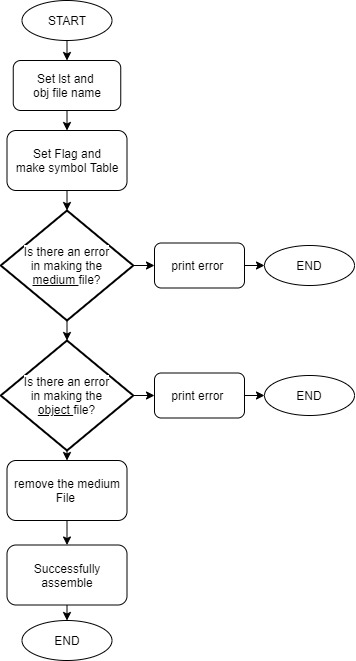


④ assemble -> assemble의 전반적인 과정을 ⑤ symbol -> symbol Table이 비어있는지

담당하는 함수로 에러 출력 및 중간 파일(sic/xe) 확인하고, symbol들을 저장한 이후에

을 삭제한다. symbol Table을 내림차순으로 정렬한다.





**3. 모듈 정의**

3.1 정의한 모듈에 대해 간략적인 설명

① assembler.c

이 모듈은 assemble, symbol 명령어를 처리하기 위한 함수들이 들어있는 모듈이다. 사용자가

입력한 어셈블리 코드를 이용하여 object 코드와 lst 파일을 생성한다. 이 과정에서 어셈블리 문법에

맞지 않는 경우, 심볼이 중복되어 나오는 경우, 메모리 범위를 벗어나는 경우, 입력 받은 asm파일이

존재하지 않는 경우 등을 예외 처리하였다. 소스 파일에 에러가 존재할 경우 lst 파일과 obj 파일을

생성하지 않고 에러 내용을 화면에 출력한다. 이 때 에러의 내용은 디버깅을 위해 어떤 라인에서 에

러가 발생했는지 출력한다.

또한, assemble 과정에 생성된 symbol table을 화면에 출력하는 symbol 명령어를 처리하며 가장

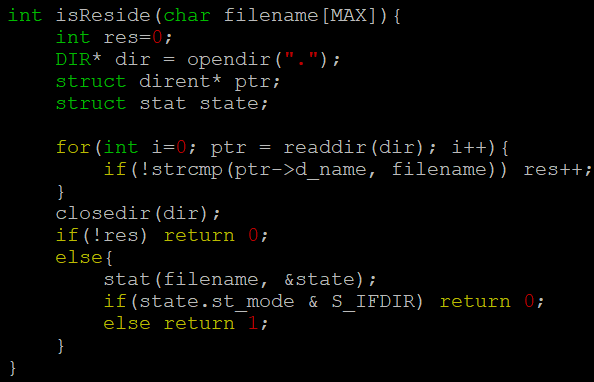
최근에 assemble 한 파일의 symbol table을 내림차순으로 정렬하여 출력한다.

**4. 전역 변수 정의**

|  |
| --- |
| symNode \*symTable[] : assemble 과정 중에 생성된 symbol table |
| symNode symArr[] : assemble 과정 중에 생성된 symbol table에서 register를  제외한 symbol들만 저장한 배열 |
| int symOn : symbol table이 비어있는지를 체크하는 변수 |
| char program[] : assemble 할 파일의 파일 이름 |
| int first, last, start : 파일의 시작주소와 끝 주소를 저장하는 변수 |
| char asmToken[][] : asm코드를 한 줄씩 받아서 공백과 콤마등을 기준으로 나누어  단어별로 저장한 배열 |
| int asmStart, asmEnd : asm 코드의 시작과 끝을 표시하는 flag |

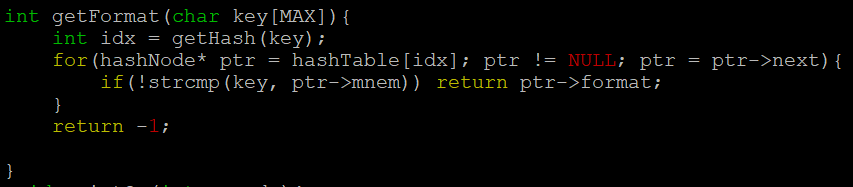
**5. 코드 설명**

① isReside(char filename[MAX])



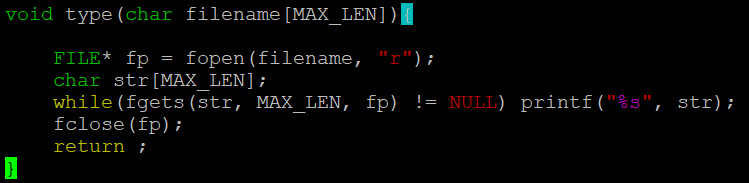
현재 디렉토리를 열어 입력받은 filename과 일치하는 파일이 있는지 확인한다. 만약 파일이 존재하지 않는다면 0을 리턴하고, 파일이 존재한다면 파일의 state를 확인하여 directory인 경우를 제외하고는 1을 리턴한다.

② getFormat(char key[MAX])



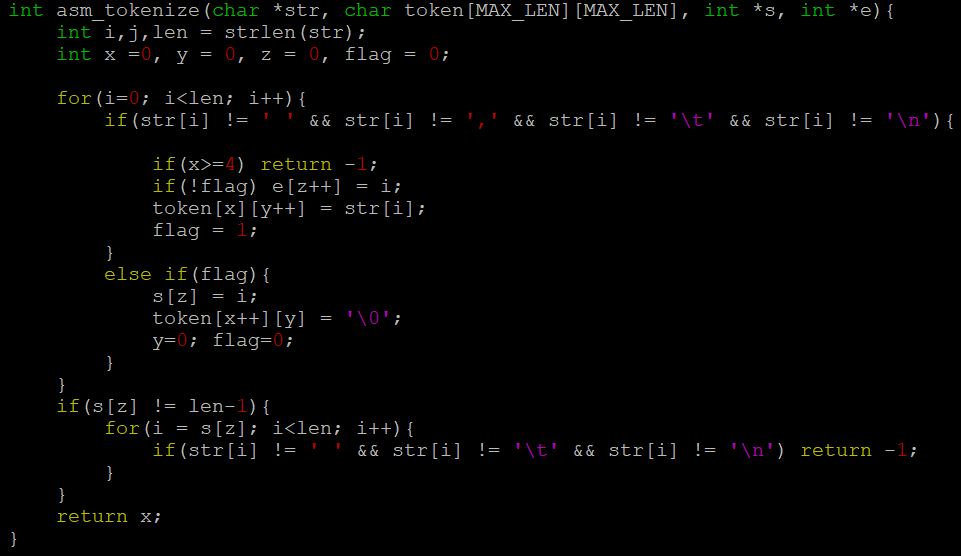
입력받은 key와 일치하는 hashTable을 찾아 해당 노드의 format을 리턴한다. 만약 일치하는 hashTable을 찾지 못했으면 -1을 리턴값으로 갖는다.

③ type(char filenmae[MAX\_LEN])



type 명령어를 실행하기 위한 함수로, 사용자로부터 입력받은 파일을 열어 그 파일이 끝나기 전까지 한 줄씩 출력한 후 파일 포인터를 닫는다.

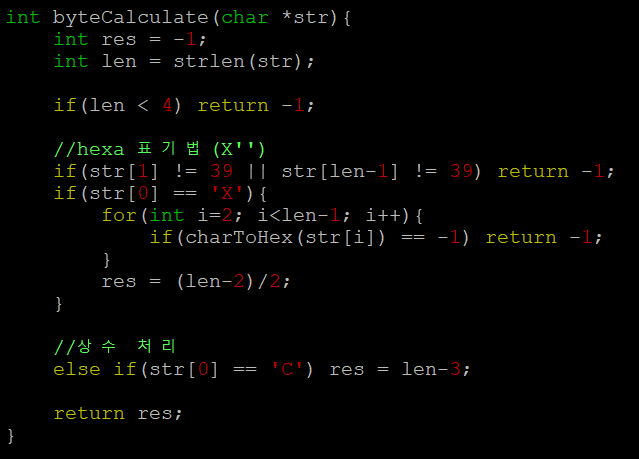
④ asm\_tokenize(char \*str, char token[MAX\_LEN][MAX\_LEN], int \*s, int \*e)



이 함수는 asm 파일 한 줄을 공백과 콤마, 탭, 엔터를 기준으로 나누어 단어별로 저장하는 함수이다. 이 때 s 배열은 공백이 시작하는 인덱스를, e 배열은 공백이 끝나는 인덱스를 담고 있으며, token[x]는 각 단어를 담고 있다. 단어의 마지막에는 '\0' 문자를 넣어 에러 가능성을 줄였다. 리턴값은 단어의 개수이다.

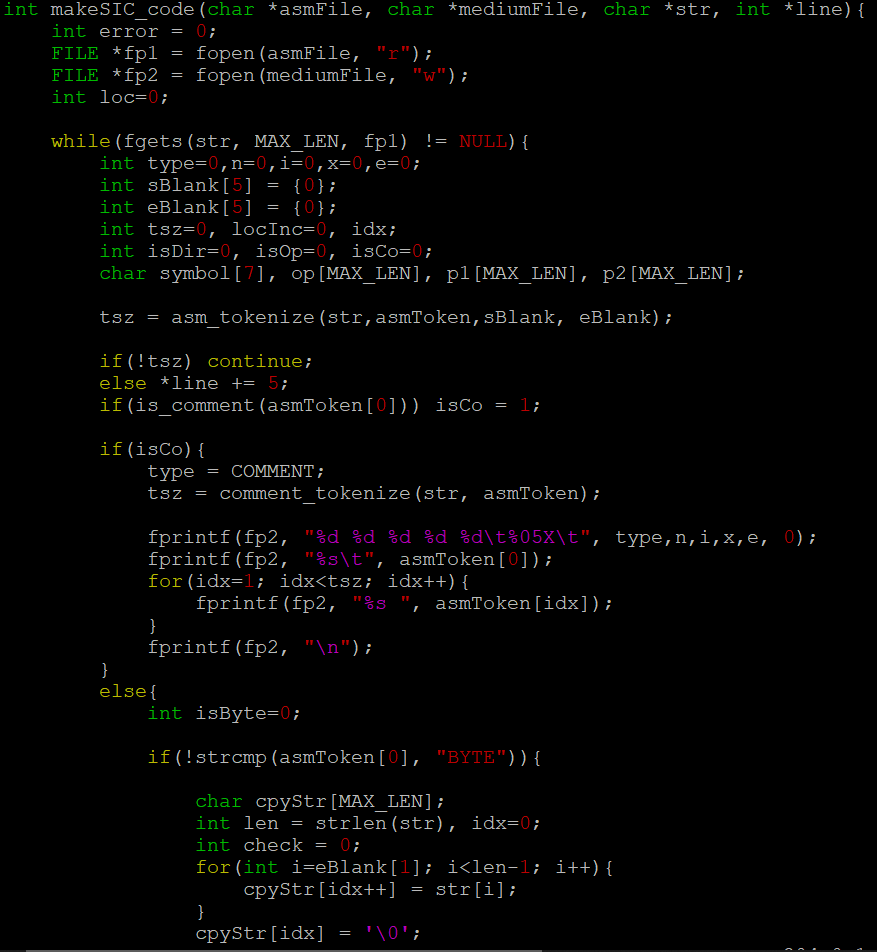
⑤ int byteCaculate(char \*str)

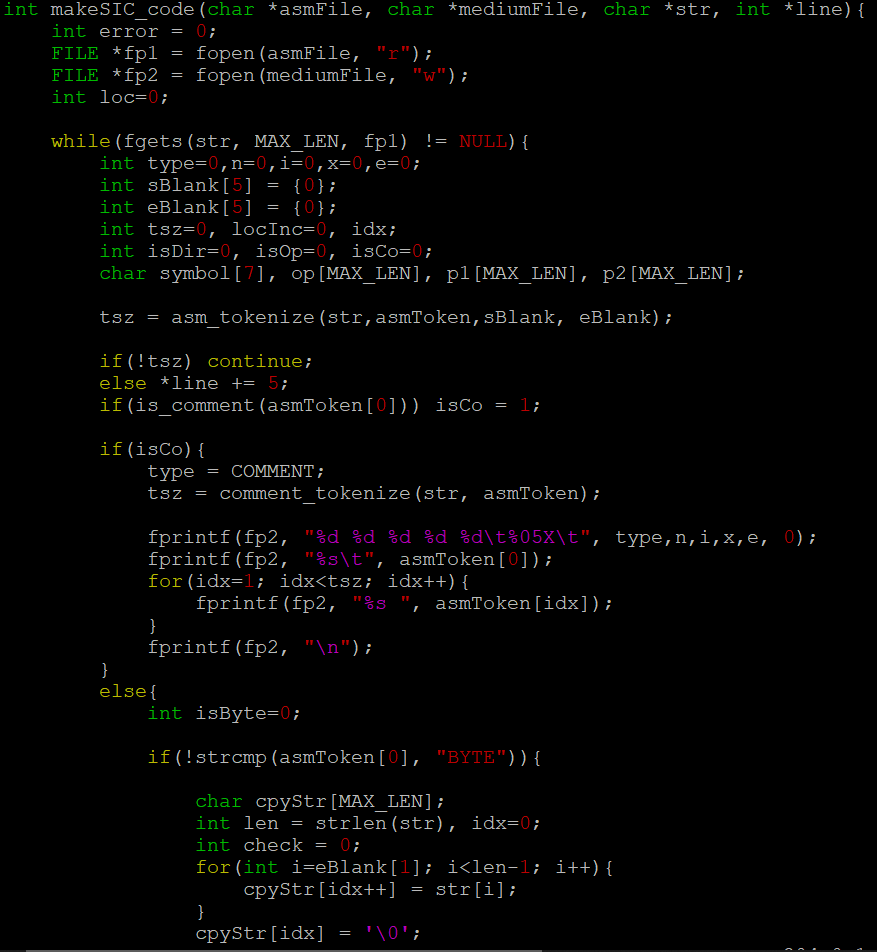
이 함수는 BYTE 상수를 처리하기 위한 함수이다. 16진수로 들어오는 경우와 문자열로 들어오는 경우를 나누어 처리하였으며 변수 len에는 BYTE 상수의 정확한 길이가 저장된다. 리턴 값은 해당 BYTE 상수를 나타내는 데 필요한 바이트 수이다.



⑥ int makeSIC\_code(char \*asmFile, char \*mediumFile, char \*str, int \*line)

이 함수는 obj 파일을 만들기 전에 중간 단계인 sic/xe code를 만드는 함수이다. 이 함수는 매우 길기 때문에 설명이 필요한 일부만 사진을 첨부하였다.

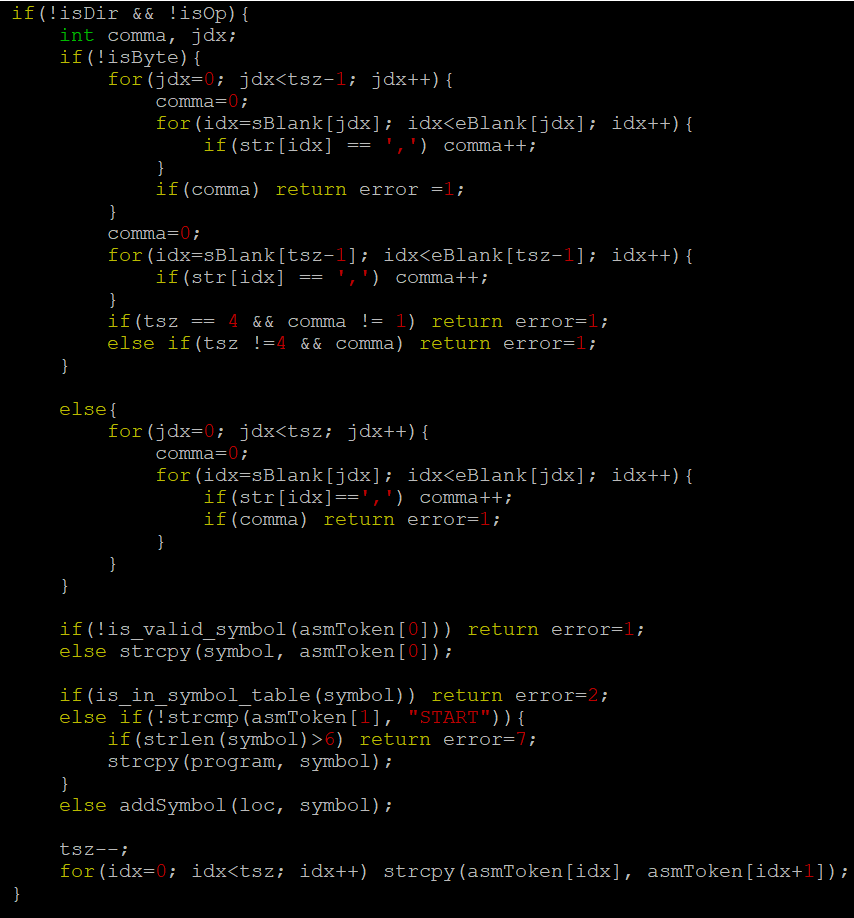




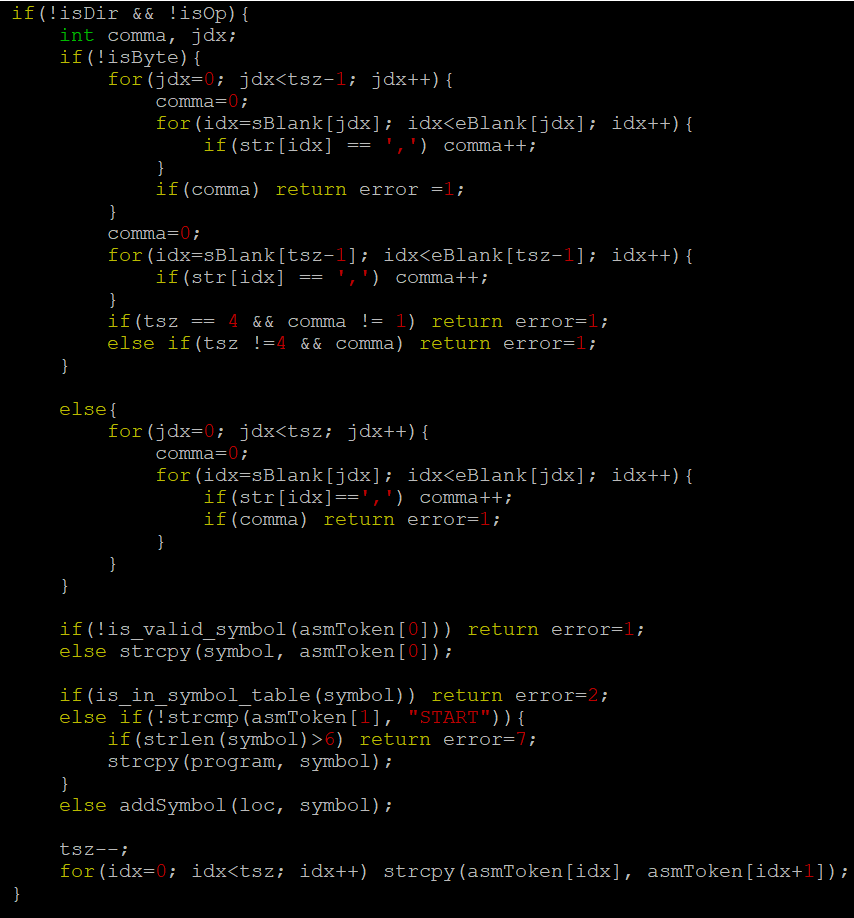
먼저 error는 에러의 종류를 구분하기 위한 변수이고, fp1, fp2는 각각 asm 파일을 읽기 위한 파일 포인터, mediumFile을 쓰기 위한 파일포인터이다. sBlank는 공백이 시작하는 인덱스 정보를 가지며 eBlank는 공백이 끝나는 인덱스 정보를 갖는다. isDir, isOp, isCo는 각각 Directive, Operator, Comment를 표시하기 위한 Flag이고 p1, p2는 parameter를 의미한다.

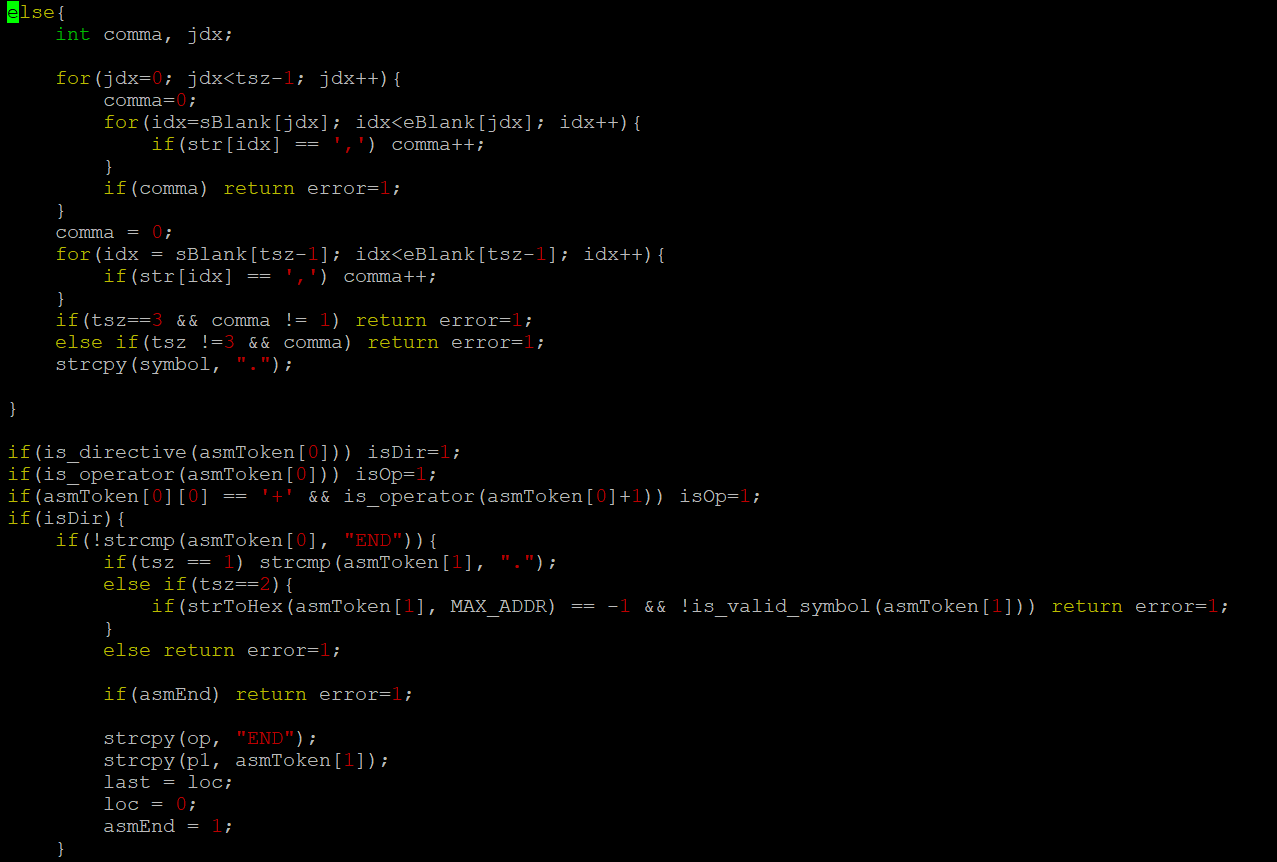
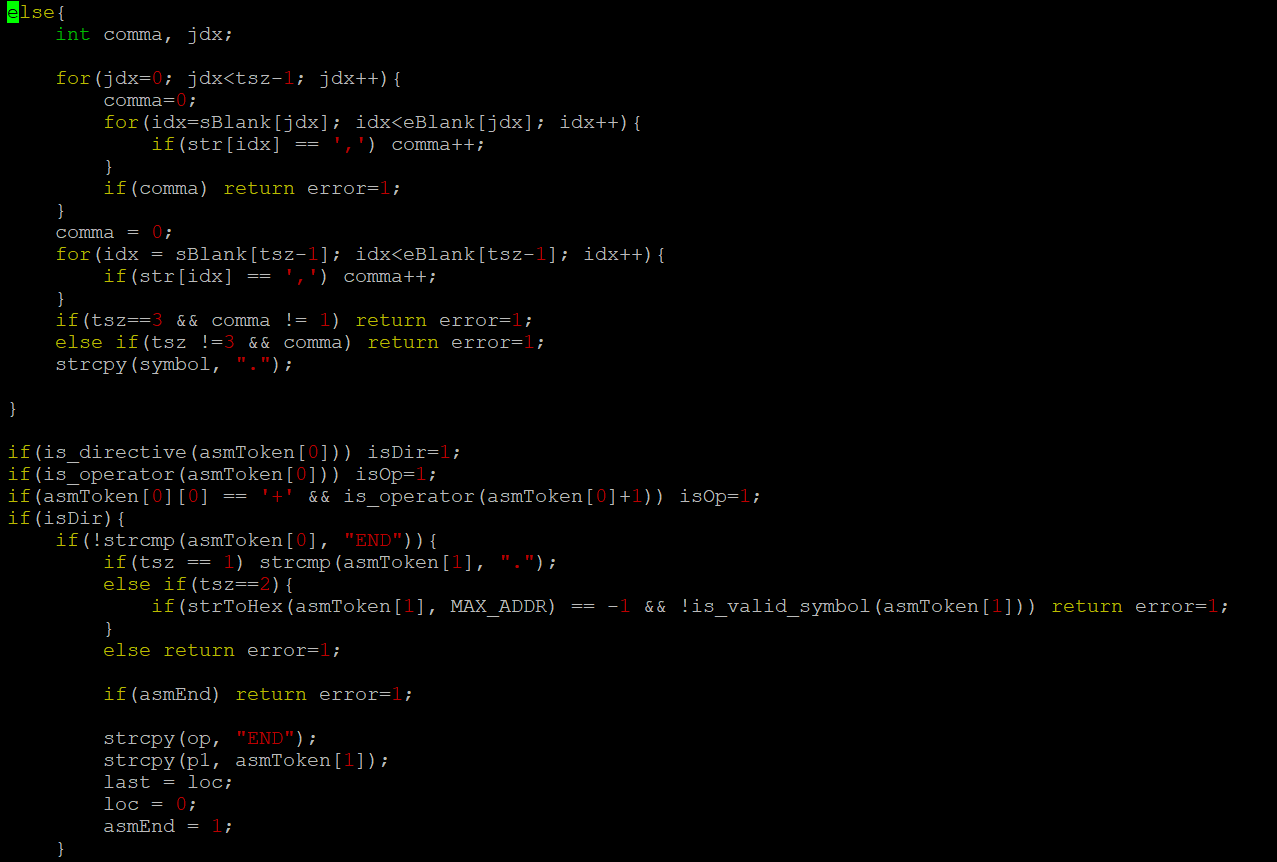


이 부분은 BYTE 상수가 들어왔을 때, 다음 토큰(asmToken[2])에 설정한 상수가 들어가도록 하며 마지막에는 널 문자를 추가하는 기능을 한다. 토큰의 개수를 설정하고 BYTE flag인 isByte를 켠다.



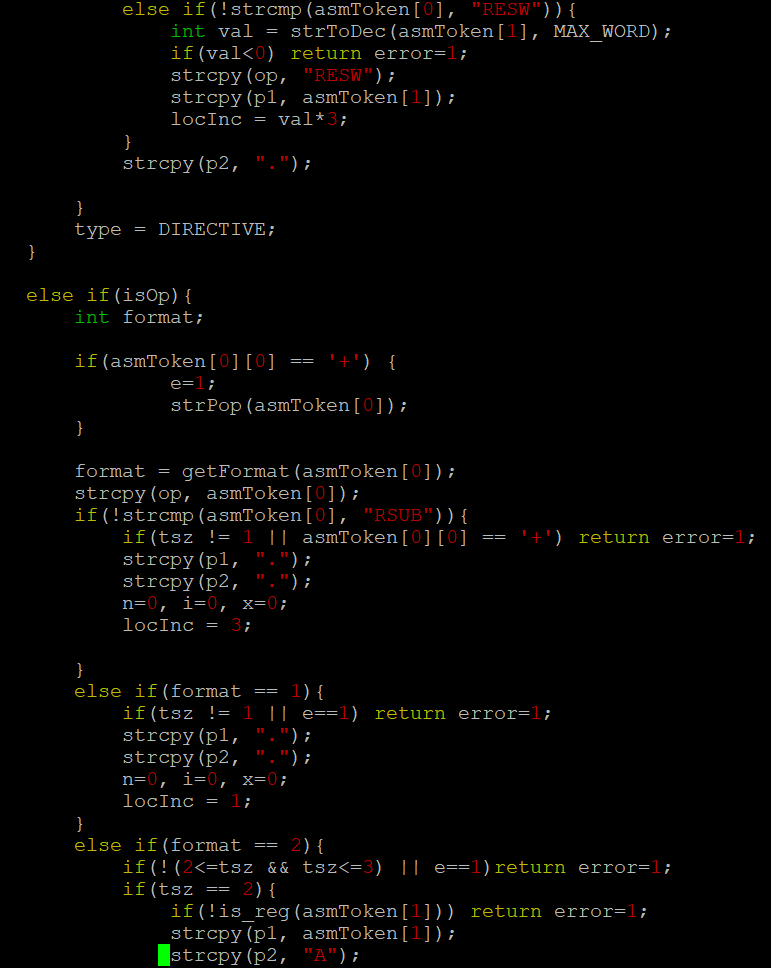
이 부분은 symbol이 나오는 부분을 처리하는 것으로 미리 저장해둔 공백 정보를 이용하여 콤마의 개수를 세고 콤마가 하나 이상은 나와야 하므로 콤마의 개수가 0이라면 에러처리를 해준다. 이어서 토큰 사이즈와 콤마의 개수를 이용하여 에러처리를 조금 더 꼼꼼하게 한다.

 이 부분은 START directive가 나오면 프로그램의 이름을 저장한다. 또한 symbol이 이미 symbol Table에 저장되어 있다면 이는 중복된 symbol이므로 에러 처리를 하고 그렇지 않다면 symbol Table에 symbol을 추가한다.

 이 부분은 END directive를 처리하기 위한 부분으로 만약 END만 들어와서 토큰의 개수가 1인 경우 parameter에 '.'을 넣는다. END program의 꼴로 토큰 개수가 2인 경우 해당 프로그램의 주소와 symbol의 유효여부를 확인한다. 아직 asm 파일이 끝나기 전이므로 소스의 끝을 의미하는asmEnd Flag가 켜져있다면 에러처리를 해주고 프로그램 의 끝 주소를 의미하는 last에 현재의 location을 넣는다.

이 부분은 다른 Directive를 처리하는 것으로 START directive가 들어오면 asmStart와 주소의 유효여부를 확인하여 에러처리를 한다. 마찬가지로 프로그램의 시작 주소를 의미하는 first에 현재 주소를 넣는다.

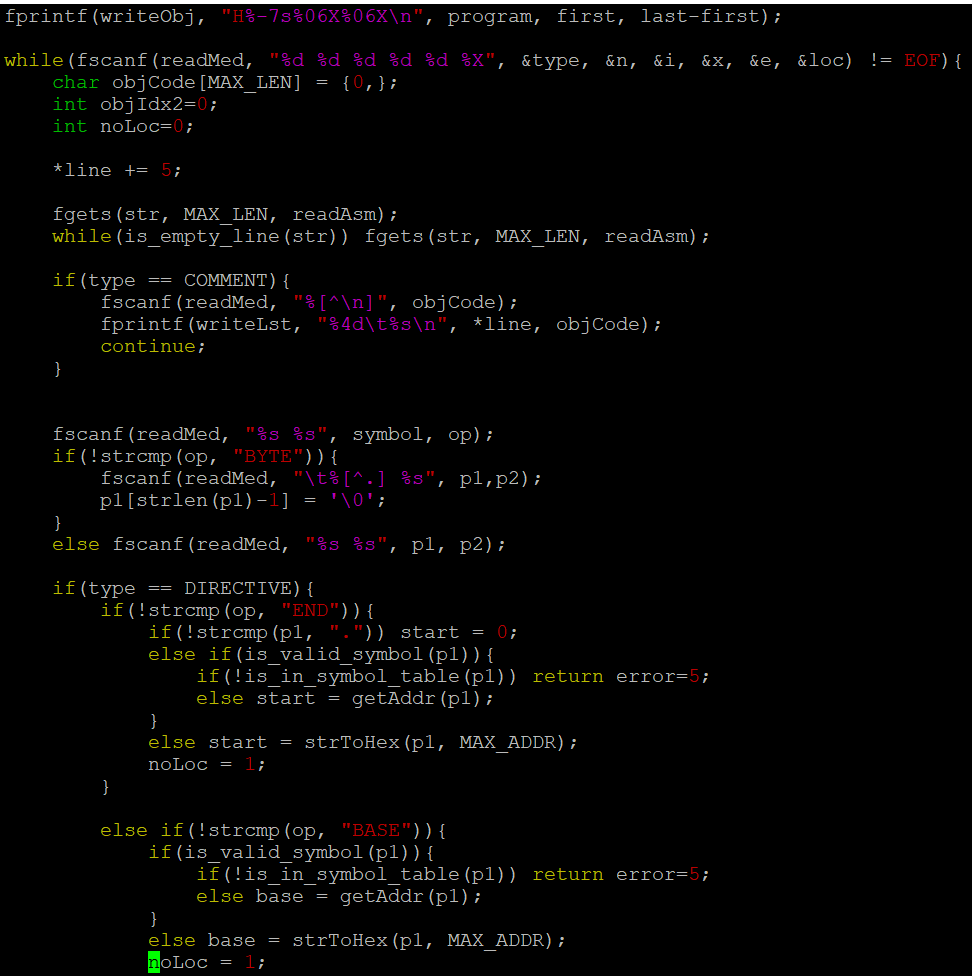
BYTE, WORD, RESB, RESW directive가 들어오면 입력받은 상수 혹은 변수가 저장 가능한 값인지 확인하고, parameter에 그 값을 넣는다. 상수 혹은 변수의 바이트 수만큼 location counter를 증가시킨다.



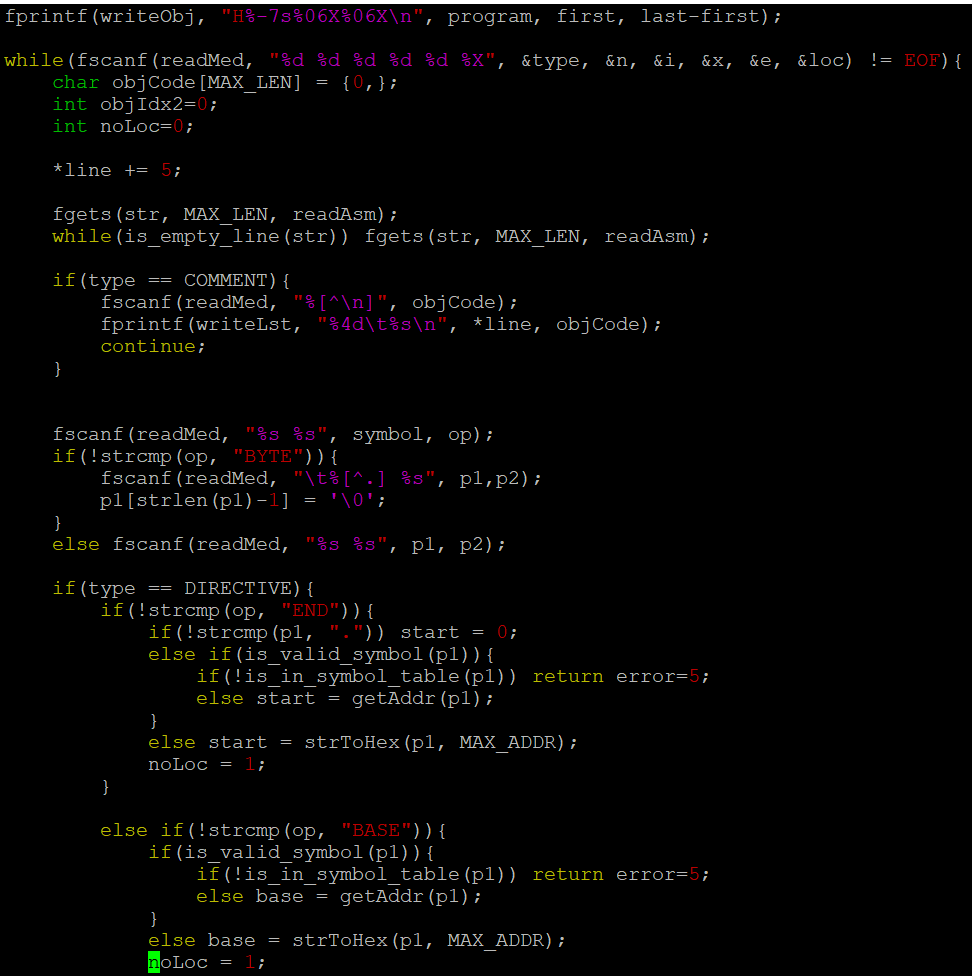
만약 Operator가 들어왔다면 format을 기준으로 나누어 명령을 처리한다. 포맷에 맞추어 location counter를 증가시키고, 토큰의 개수를 참고하여 간단한 에러처리를 해준다. 

포맷에 맞추어 처리를 한 후, 배열 레지스터의 사용여부와 addressing mode를 결정한다. 만약 배열 레지스터를 사용한다면 x레지스터를 1로 설정한다. 또한 '@'가 나온다면 이는 indirect addressing을 의미하는 것이므로 n, i를 각각 1,0으로 설정한다. 마지막으로 '#'이 나온다면 immediate addressing이므로 n, i를 각각 0,1로 설정한다. 그렇지 않은 경우는 simple addressing으로 n,i를 모두 1로 설정한다.

⑦ int makeObjFile(char \*mediumFile, char \*asmFIle, char \*objFile, char \*lstFile, char \*str, int\* line)

 이 함수는 이전에 만들어둔 중간 파일을 이용하여 objFile과 lstFile을 만드는 함수이다. 이 함수 역시 길이가 매우 길기 때문에 설명이 필요하다고 생각되는 일부분만 사진을 첨부하였다.

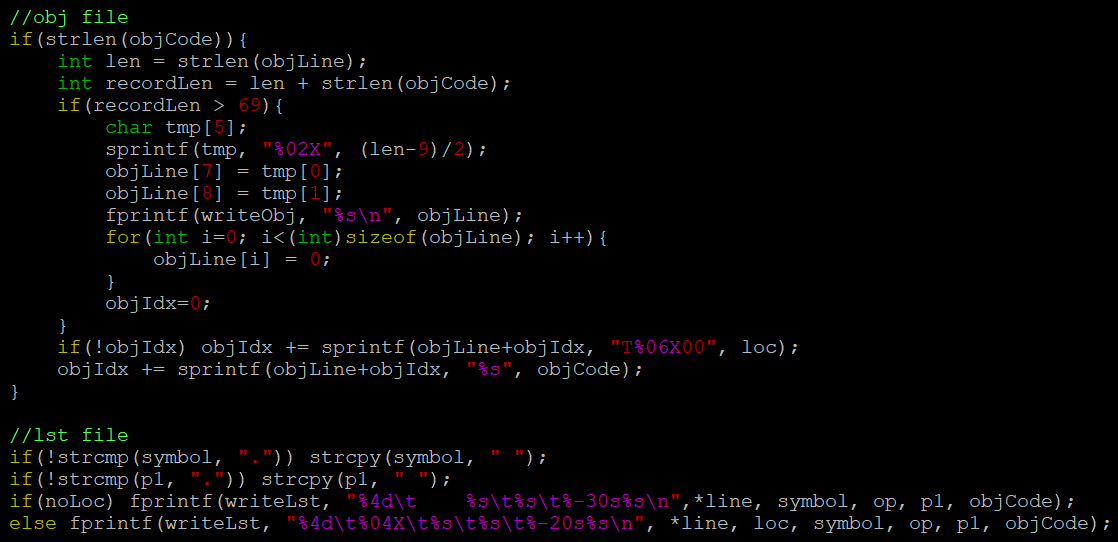
만약 BYTE 상수가 들어온다면, BYTE 상수의 끝을 의미하는 '.' 전까지 읽은 후 마지막에 널 문자를 추가한다.



다음은 directive를 기준으로 나누어 명령을 처리하는 부분이다. 먼저 END가 들어오면 parameter의 존재 여부에 따라 start 주소를 설정한다. 또한 BASE directive가 들어오면 base 주소를 설정한다.

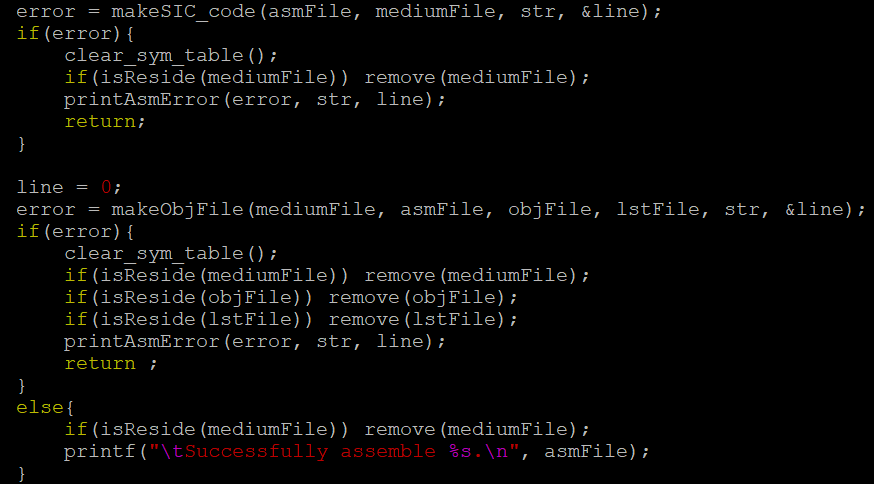


operator가 들어오면 중간 파일을 만들때와 같이 format을 기준으로 나누어 처리한다. 그러나 이번에는 예외가 되는 operator(RSUB)를 먼저 처리한다. format 3의 pc counter는 현재 보고있는 location보다 한 줄 앞을 보고 있으므로 pc = loc+3이다. ta는 target address를 의미하고 isSym은 symbol의 여부를 체크하는 Flag이다. 만약 target address(ta)가 -1이라면 이는 에러 처리를 해준다.



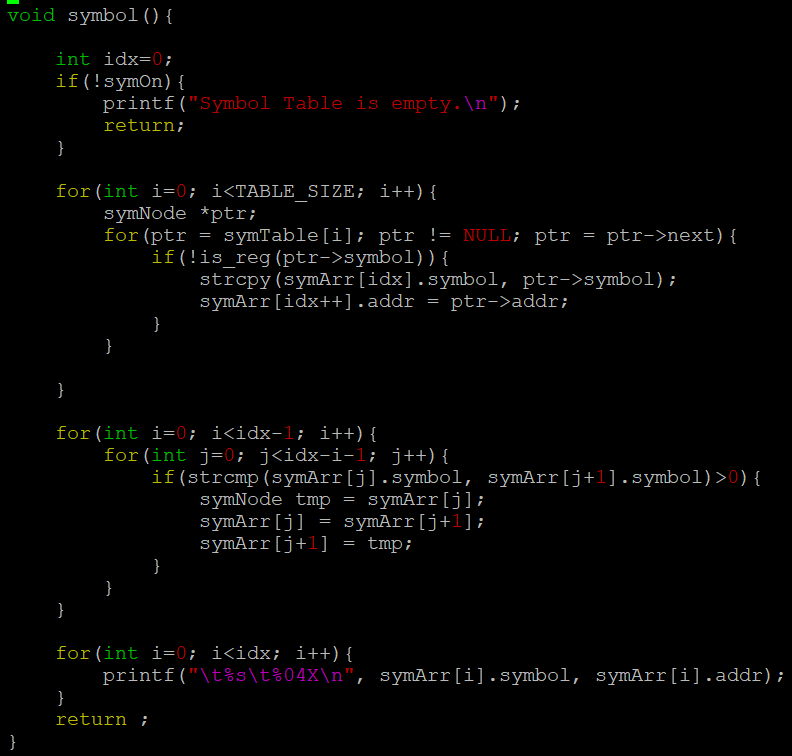
마지막으로 obj 파일과 lst 파일의 형식에 맞춰 파일을 생성한다. 앞 부분에 의해 objCode 배열에 저장된 것을 길이에 맞춰 차례로 출력하고 맨 마지막에는 program relocation을 위해 Modification record를 출력한다. 이때 Modification record에서 2~7번째 글자는 수정이 필요한 주소 필드의 프로그램 시작 위치에 대한 상대적인 주소를 16진수로 표현하고 있으며 8~9번째 글자는 수정될 주소의 필드 길이를 의미한다.

⑧ void assemble(char \*asmFile)



이 함수는 assemble 과정을 전반적으로 다루는 함수로 중간 파일과 obj파일을 만들기 전에 에러체크를 한다. 에러가 나면 디버깅을 위해 에러가 난 라인과 내용을 화면에 출력한다. 만약 에러가 나지 않는다면 성공적으로 obj file과 lst file을 생성하고 안내문을 출력한다. 함수를 종료하기 전에 생성한 중간 파일을 삭제한다.

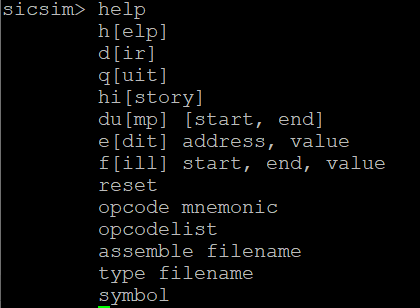
⑨ void symbol()



이 함수는 symbol 명령어를 처리하기 위한 함수로 symbol Table이 비어있다면 비어있다는 메세지를 출력하고, 그렇지 않다면 symbol Table에서 레지스터가 아닌 것들만을 따로 symArr에 저장한다. symArr는 symbol을 기준으로 내림차순으로 정렬한 후 형식에 맞춰 출력한다. 만약 여러개의 asm 파일을 assemble 했다면, 가장 최근에 assemble 한 파일의 symbol table을 형식에 맞춰 출력한다.

**6. 테스트 케이스**

① h[elp]

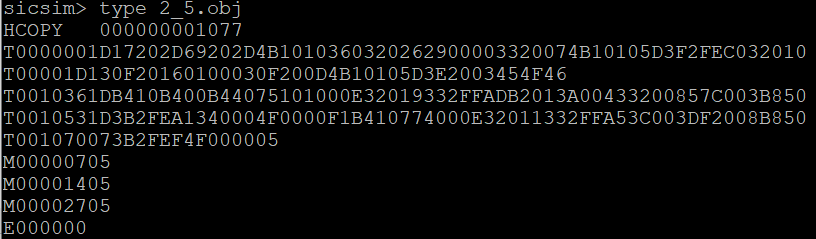


② assemble filename

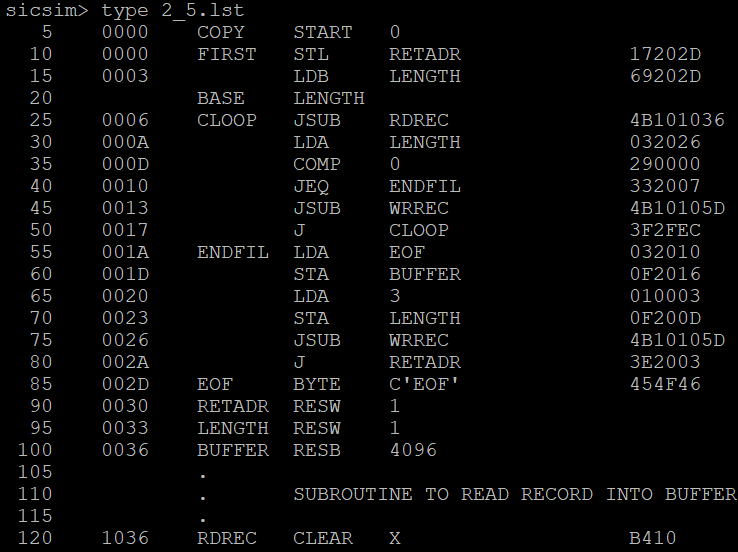


③ type filename

(obj file)



(lst file)



④ symbol

