**과목명: 시스템프로그래밍**

**홀수반**

**<<Project #3>>**

**서강대학교 컴퓨터공학부**

**20181593**

**계인혜**

목 차

**1 프로그램 개요**

**2 프로그램 설명**

2.1 프로그램 흐름도

2.2 세부 프로그램 흐름도

**3 모듈 정의**

**4 전역 변수 정의**

**5 코드 설명**

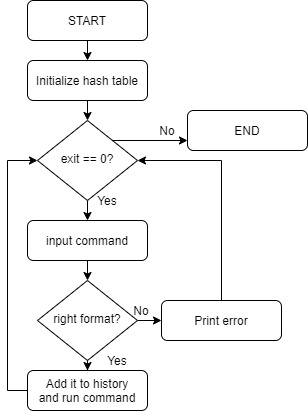
**6 테스트 케이스**

**1 프로그램 개요**

프로젝트 1, 2 에서 구현한 셀(shell)에 linking과 loading 기능을 추가하는 프로그램이다. 프로젝트 2 에서 구현된 assemble 명령을 통해서 생성된 object 파일을 link시켜 메모리에 올리는 일을 수행한다.

**2 프로그램 설명**

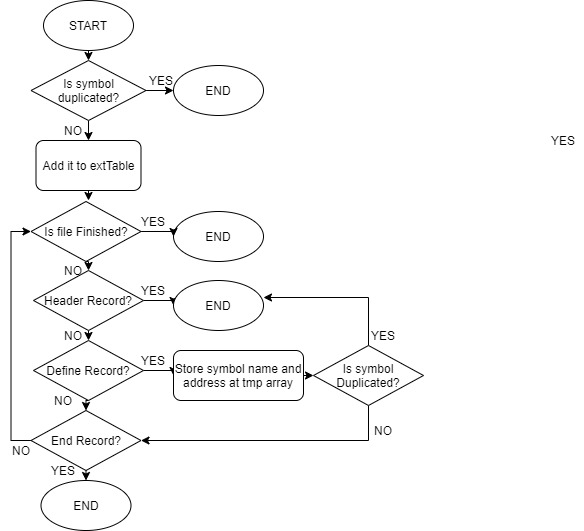
2.1 프로그램 흐름도



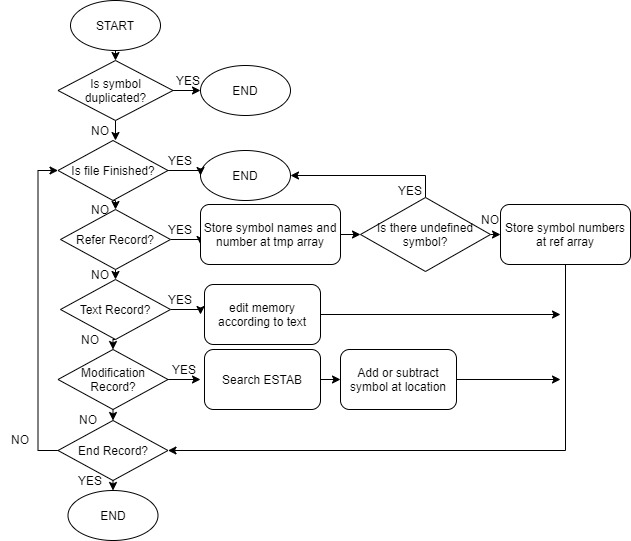
2.2 세부 프로그램 흐름도

① updateExt : loader 명령어가 들어오면 filename에 있는 symbol의 name과 addr을 저장하고,

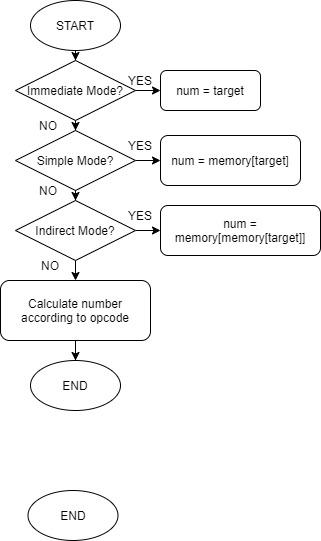
그 과정에서 중복으로 선언되는 symbol을 찾아 에러를 처리한다.



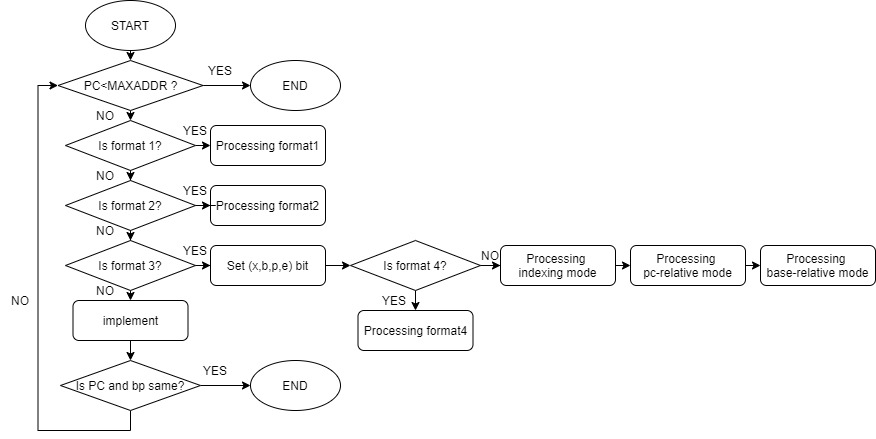
② linkingLoad : obj파일을 한 줄씩 읽어, record에 맞게 처리하고 그 과정에서 정의되지 않은 symbol에 접근하는 경우를 에러로 처리한다.



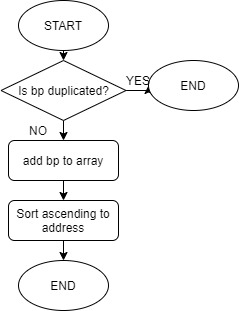
③ Implement : addressing mode에 따라 number를 설정하고, opcode와 일치하는 연산을 수행 후 그 결과를 레지스터에 저장한다.



④ runProcess : format에 맞게 명령을 처리하고, PC값을 증가시키며 n, i, x, b, p, e 비트 값을 변경한 다. 문제가 없을 시 명령을 수행하고(implement) 프로그램이 끝나거나 breakpoint를 만나면 레지스터 값을 출력하는 함수를 호출하고 함수를 종료한다.



⑤ setBp : breakpoint 배열에 사용자가 입력한 값이 존재하는지 확인한 후, 존재하지 않는다면 이를 추가하고 그렇지 않다면 추가 없이 함수를 종료한다. 또한 breakpoint 배열을 address 값 에 대해 오름차순으로 정렬한다.



**3 모듈 정의**

① linking\_loader.c

이 모듈은 progaddr, loader, run 명령을 실행하기 위한 함수들이 들어있는 모듈이다.

Loader 또는 run 명령어를 수행할 때 시작하는 주소를 지정하며, filename에 해당하는 obj 파일을 읽어 linking 작업을 수행한 후, 가상 메모리에 그 결과를 기록한다. 성공적으로 loader 명령어가 끝났다면, load map을 화면에 출력한다.

이런 방식으로 메모리에 로드된 프로그램을 실행하며 breakpoint가 있으면 거기서 멈추고 register 값을 화면에 출력한다. 만약 breakpoint가 없으면 프로그램을 끝까지 수행한 후 register 값을 화면에 출력한다.

② debug.c

이 모듈은 bp(breakpoint) 명령을 실행하기 위한 함수들이 들어있는 모듈이다.

현재 프로그램의 location counter를 기준으로 breakpoint를 설정하며, run 명령어를 수행하면 breakpoint까지 프로그램을 수행한다. 프로그램이 끝나기 전에 run 명령어를 다시 입력하면 정지했던 breakpoint부터 다시 프로그램을 돌려 그 다음 breakpoint까지 진행되는 방식이다. 만약, breakpoint가 없다면 프로그램 끝까지 수행한다.

bp를 입력하면 사용자가 입력한 breakpoint를 모두 화면에 출력하며, bp clear를 입력하면 저장된 모든 brekapoint를 삭제한다. bp address를 입력하면 address위치에 breakpoint를 새로 설정한다.

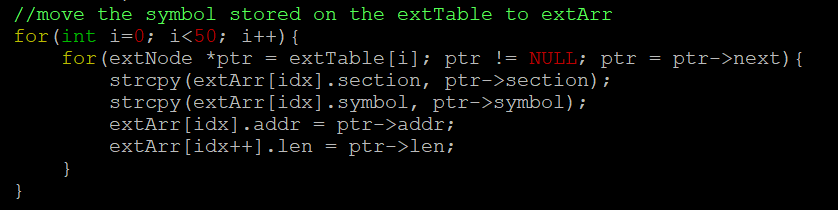
**4 전역 변수 정의**

|  |  |
| --- | --- |
| **breakpoint[]** | breakpoint의 address를 저장하는 배열 |
| **bpCnt** | 현재 저장된 breakpoint의 개수 |
| **startAddr** | 프로그램의 시작 주소 |
| **extTable** | external symbol을 hash function을 사용해 저장 |
| **extArr** | load-map을 출력하기 위해 symbol들을 모아서 저장한 배열 |
| **reg[]** | run 실행 후 레지스터에 저장된 값을 출력하기 위한 배열 |
| **debugF** | 프로그램의 시작이 처음인지 breakpoint인지 구분하기 위한 변수 |
| **errLine** | linking-load 과정에서 에러가 발생한 라인을 저장하는 변수 |
| **errSym** | linking-load 과정에서 에러가 발생한 symbol을 저장하는 변수 |

**5 코드 설명**

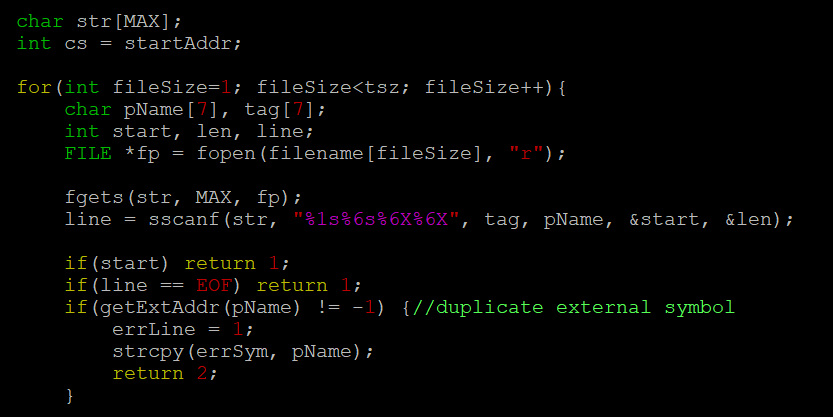
① void printExt()

external symbol table에 저장되어 있는 symbol들을 차례로 하나의 array(extArr)에 넣어 출력에 용이하게 만들었다. 첨부한 코드 이후 부분은 형식에 맞춰 출력하는 부분이라 생략하였다.



② int updateExt(int tsz, char filename[][])

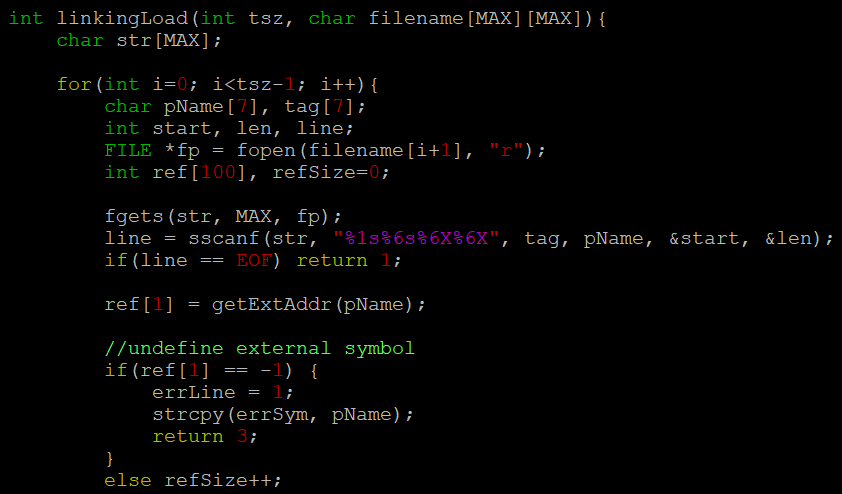
파일의 개수에 따라 각각의 obj파일을 한 줄씩 읽어 external symbol table을 갱신하는 함수이다. 먼저 H record에서 입력받은 프로그램 이름과 중복되는 symbol이 있다면 해당 라인과 심볼 이름을 저장하여 함수를 종료함으로써 에러를 처리한다. 에러가 아니라면 이를 external symbol table에 추가하고 D record가 있다면 이를 살핀다. D record에서도 마찬가지로 각각의 symbol과 address를 저장한 후 중복되는 symbol이 아니라면 이를 external symbol table에 추가한다. 이 과정에서 중복되는 symbol이 있다면 에러 처리를 하고 함수를 종료한다.

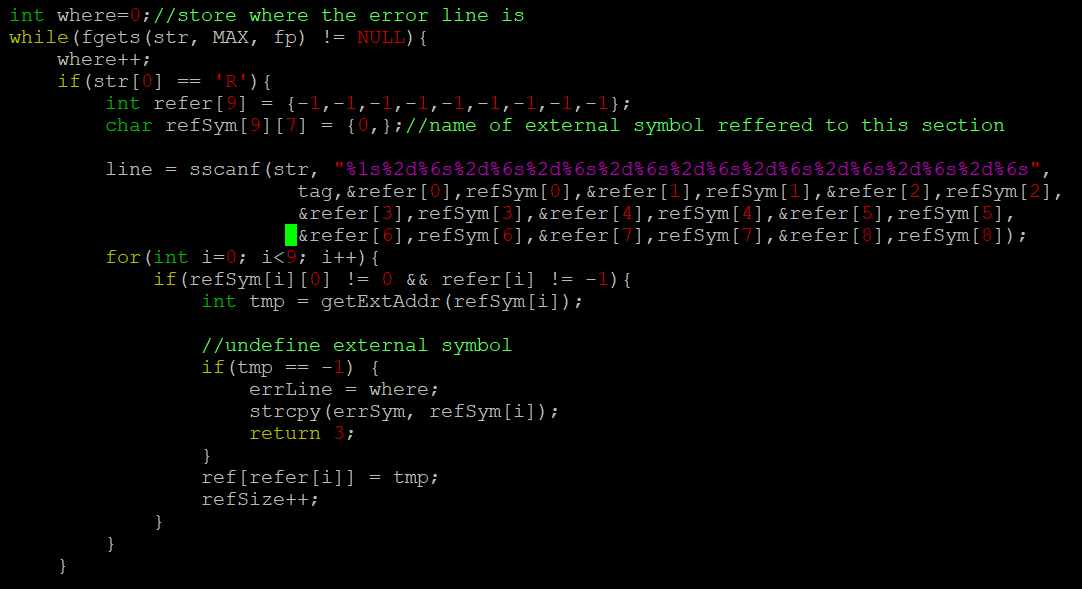




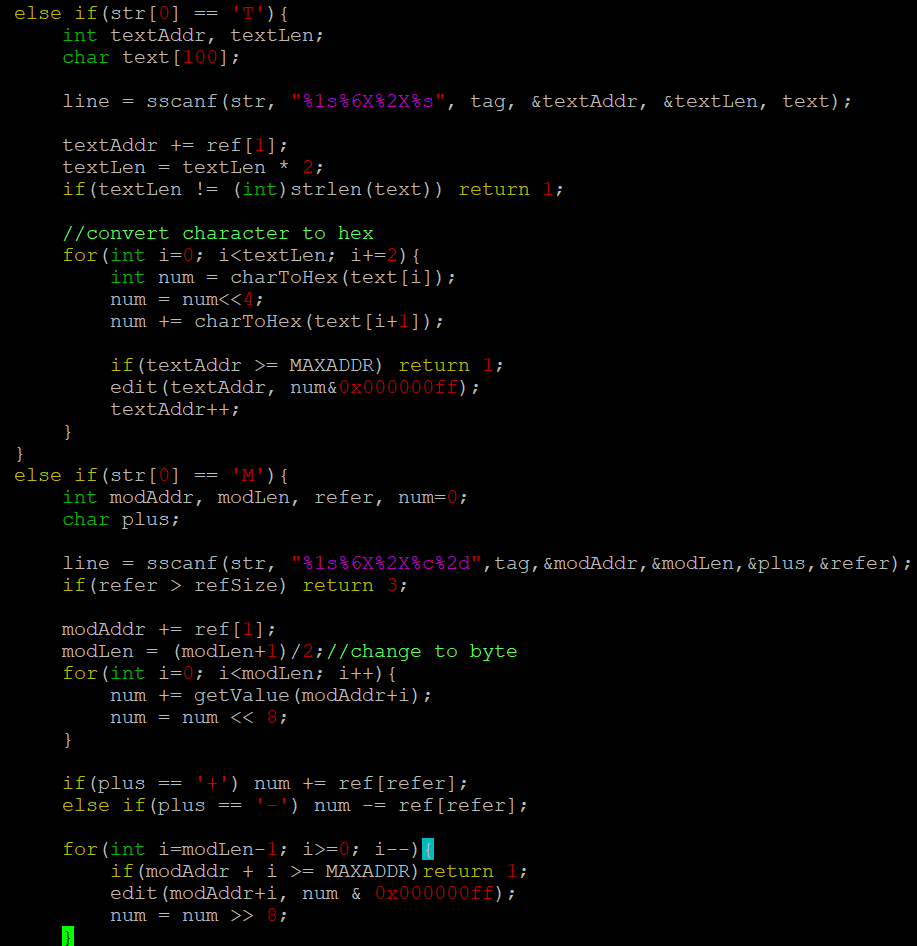
③ int linkingLoad(int tsz, char filename[][])

이 함수는 파일 개수에 맞춰 각각의 파일들을 한 줄씩 읽어 linking하는 함수이다. 먼저 H record가 있는 첫 줄을 읽어서 프로그램의 이름이 external symbol table에 저장이 되어 있지 않다면 오류를 출력한다. 첫 줄에 문제가 없으면 reference 배열의 크기를 늘린다.

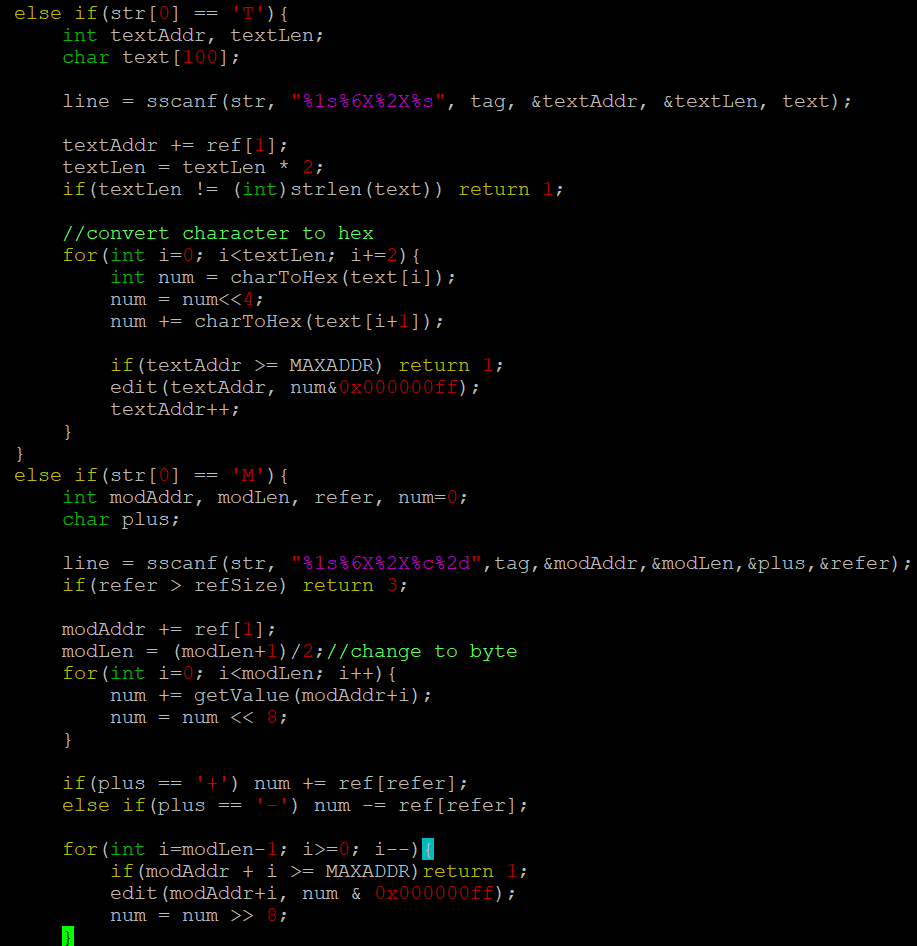




첫 줄에서 문제가 없었다면 파일이 끝나기 전까지 코드를 한 줄씩 읽어와서 record에 맞게 명령을 처리한다. Refer record라면, refer symbol을 refSym배열에 refer address를 refer배열에 저장한다. 이후 저장된 refer symbol, refer address들이 external symbol table에 저장되어 있는지 확인한다. 만약 하나라도 존재하지 않는다면 에러를 출력하고 함수를 종료한다.



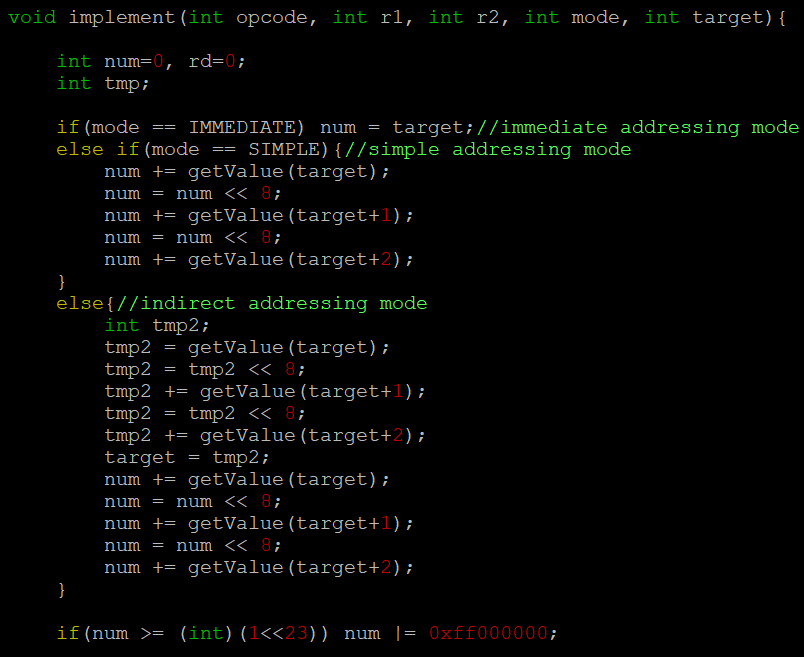
Text Record라면 형식에 맞춰 obj 코드를 입력받은 후, object code는 hexa(half byte)로 저장되기 때문에 byte단위로 text를 읽어온다. 이때 text는 자료형이 char 이기 때문에 사용하기 위하여 16진수 꼴로 바꿔준다.

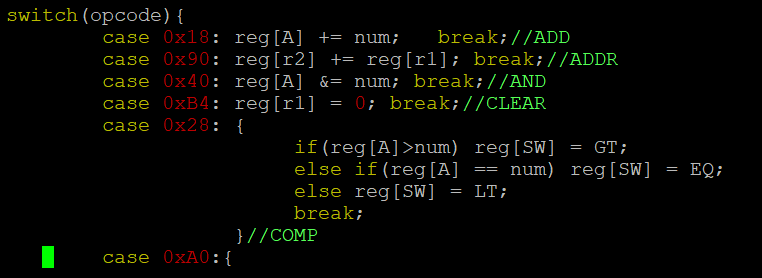


Modification Record라면 형식에 맞춰 obj 코드를 입력받은 후, 수정 레코드 주소를 control section의 상대주소로 설정한다. 또한 modLen은 half byte 단위이기 때문에 단위를 byte로 바꿔준 후, modification flag를 보고 더할지 뺄지를 결정한다.

④ void implement(int opcode, int r1, int r2, int mode, int target)

addressing mode에 따라 num을 수정한다. immediate addressing이면 바로 target값을 num에 대입하고, simple addressing이면 memory[target]의 값을 num에 대입한다. 마지막으로 indirect addressing이면 memory[memory[target]]값을 num에 대입한다.





Opcode에 맞는 연산을 수행한다. 명령어들의 종류가 너무 많아서 일부분의 코드만 첨부하였다.

switch문을 이용하여 obj코드를 수행하는데 사용되는 모든 명령어들을 표현하였다. 이때 LT(Less

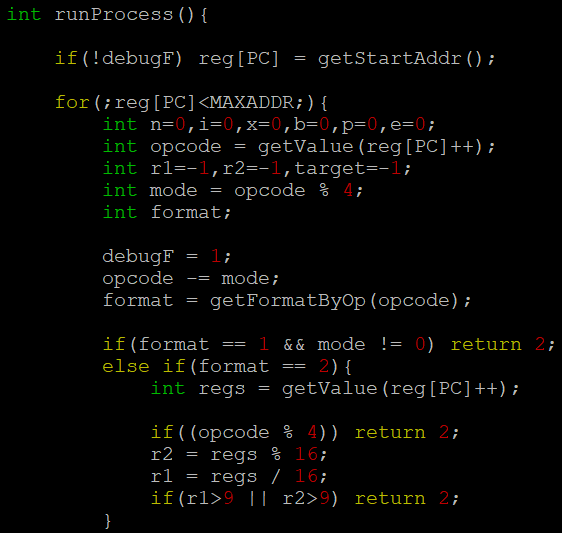
Than), GT(Greater than), EQ(Equal)는 각각 1, 2, 3 으로 define 했다.

⑤ int runProcess()

해당 프로그램의 시작 지점이 처음인지 breakpoint 이후인지를 먼저 판단한다. 이후 format에 맞

춰 레지스터 값을 설정한다.(format 2) 또한 포맷에 따라 명령을 처리 후 Program Counter의 값도 갱

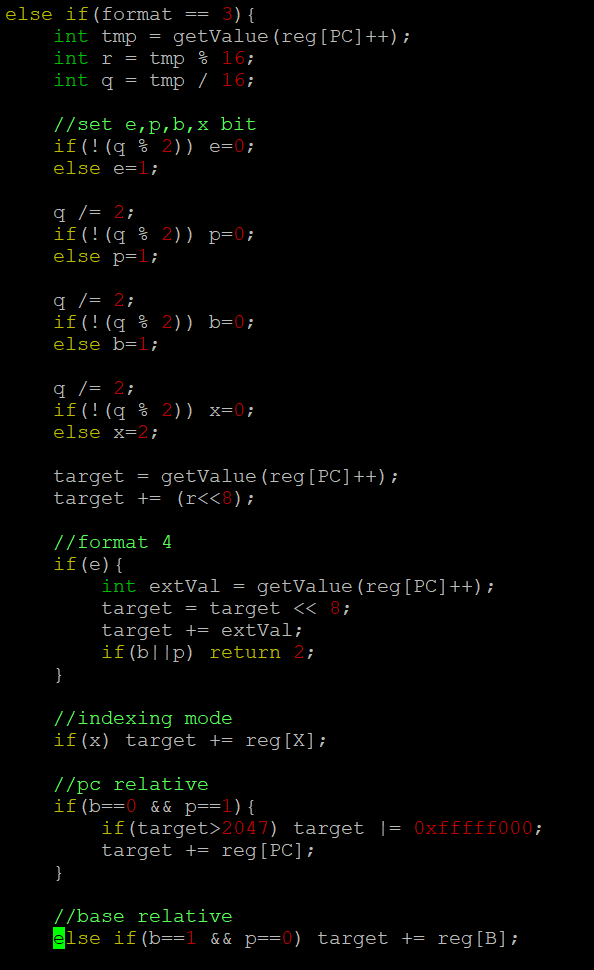
신한다. 만약 이 과정에서 에러가 생긴다면 함수를 종료하고 에러를 출력한다.



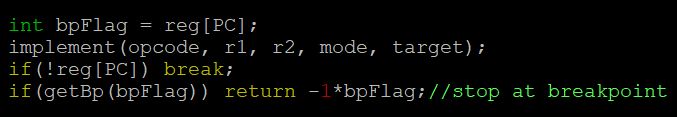
opcode의 format이 3이라면, 먼저 x, b, p, e 비트를 설정한다.설정된 e 비트의 값이 1이라면

format 4로 처리하고 그렇지 않다면 이어서 x를 체크하여 indexing mode여부를 확인한다.

마지막으로 pc relative인지 base relative인지 결정한 후 target 값에 reg[PC] 또는 reg[B]를 더한다.

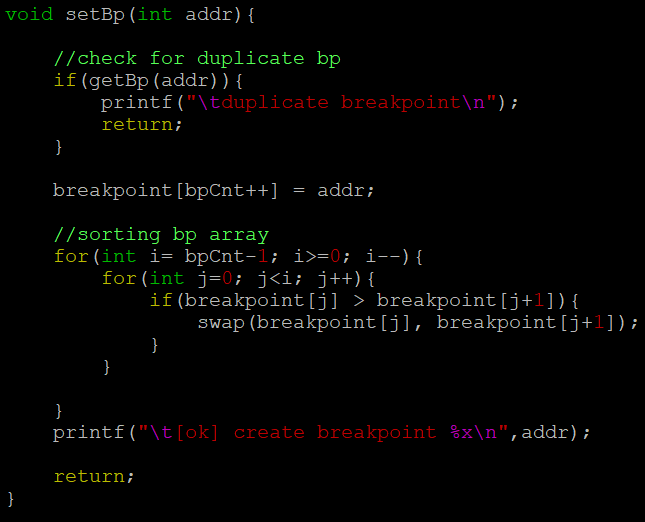


process를 run 하는 도중 사용자가 미리 설정해 둔 breakpoint를 지나는지 확인한다. 만약, 현재 Program counter가 사용자가 설정한 breakpoint라면 함수를 종료하고 레지스터 값들을 출력한다. 그렇지 않다면 object code를 수행하고 처음으로 돌아가서 프로그램이 끝나기 전까지 이를 반복한다.

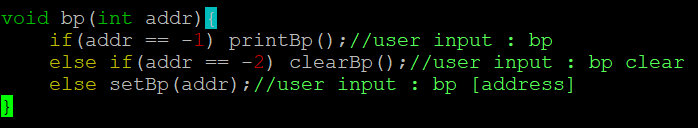


⑥ void setBp(int addr)

breakpoint를 설정하기 전에 중복되는 breakpoint가 있는지 확인한다. 중복되지 않는다면, breakpoint를 저장하는 배열에 입력받은 새 bp를 넣는다. 이후 주소에 대한 오름차순으로 breakpoint배열을 정렬한다. 모든 과정이 성공적으로 끝나면 breakpoint를 만들었다는 문구를 출력하고 함수를 종료한다.



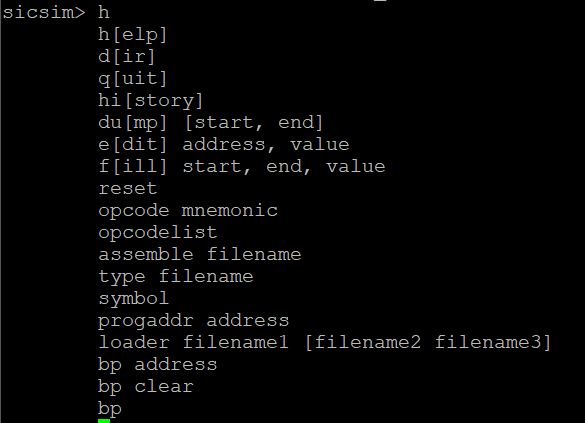
⑦ void bp(int addr)



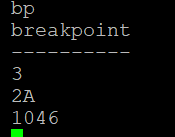
bp를 keyword로 하는 명령어들을 구분하기 위한 함수이다. 차례대로 bp, bp clear, bp address를 의미한다. 이때 bp는 설정된 모든 breakpoint를 출력하며, bp clear는 설정된 모든 breakpoint를 삭제한다. 마지막으로 bp address는 address에 설정된 breakpoint가 없다면 새롭게 breakpoint를 생성한다.

**6 테스트 케이스**

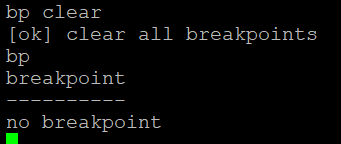
① h[elp]



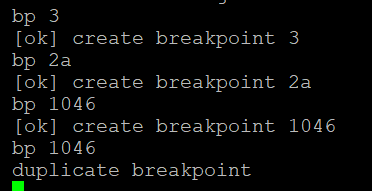
② bp



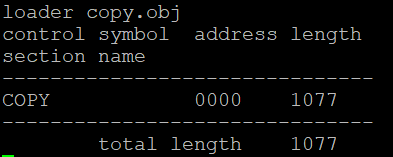
③ bp clear



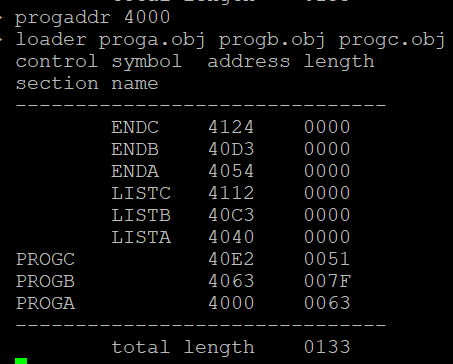
④ bp address



⑤ loader filename



⑥ (loader filename1, filename2, filename3) & (progaddr 4000)



⑦ run

