**과목명: 시스템프로그래밍**

**분반**

**<<Project #3>>**

**서강대학교 [컴퓨터공학과]**

**[20161571]**

**[김도연]**

목 차

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

**1. 프로그램 개요**

이번 프로젝트3의 목적은 obj 파일을 읽어서 linking하고 메모리에 loading하는 것이다. 이를 위해서 프로젝트 1에서 만든 메모리배열이 사용된다. 구현해야 할 것은 크게 3가지로 나뉘어져 있다.

-1 PASS1

obj 파일에서 H, D recode를 읽고 ESTAB을 만드는 것 (R recode를 읽고 reference Number 또한 사용가능하게 만드는 것을 포함한다)

-2 PASS2

T recode를 읽어서 해당 메모리 주소에 해당 값을 올리는 것.

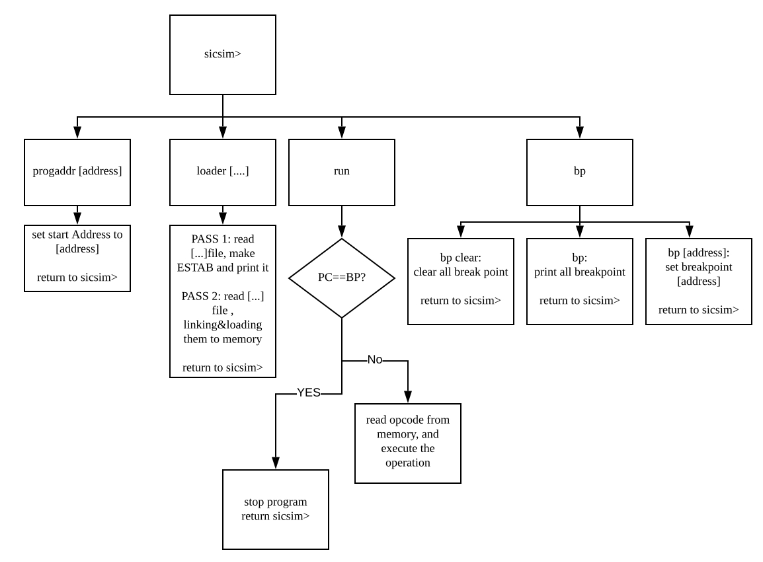
M recode를 읽어서 해당 메모리 주소의 값을 Modification 시키는것

-3 RUN

PASS1과 PASS2를 거치면 linking과 loading 과정이 끝나게 되고 모든 명령이 메모리에 올라가져 있는 상태가 된다. 이 파트의 목표는 그 명령을 실행시키는 것이다. (Debug를 위한 break point 생성과 디버깅 모드로의 run실행을 포함한다)

**2.프로그램 설명**

2.1 프로그램 흐름도



**3. 모듈정의**

3.1 정의한 모듈에 대해 간략적인 설명

void InitEnode(void);

ESTAB을 저장하기 위한 ENode의 초기화

ENode\* InsertEnode(void);

ENode를 생성하고 리스트에 삽입해주는 함수

InitBnode(void);

break point를 저장하기위한 BNode의 초기화

void InsertBnode(int bp);

BNode를 생성하고 리스트에 삽입해 주는 함수

void createbp(int bp);

InitBnode()와 InsertBnode(bp)를 호출하는 함수

void clearbp(void);

설정되어 있는 모든 break point를 제거하는 함수

int CharToDecimal(char ch)

char 값 0~9, A~F를 decimal값으로 변경해주는 함수

int gerAddrValue(int address)

해당 주소 메모리의 값을 반환하는 함수

int progaddr(int address)

프로그램의 시작주소를 address값으로 변경해 주는 함수

void linking\_loader(char\* obj1, char\* obj2, char\* obj3)

linking과 loading을 수행하는 함수. PASS1, PASS2를 실행하고 ESTAB을 출력한다

void get\_flag(void);

메모리에서 읽은 opcode로부터 n I x b p e를 얻어내는 함수

void get\_pc\_mem(void);

포멧에따라 PC값을 결정해주고 얻어온 데이터를 온전한 opcode로 조합해주는 함수

void addressing\_type(void);

PC relative, Base relative 등에 따라 target address를 만드는 함수

void init\_fm();

비트와 메모리를 받을 변수들의 초기화

void clear\_reg();

CLEAR reg 포멧에서 쓰일 함수로 reg 레지스터를 초기화 해주는 함수

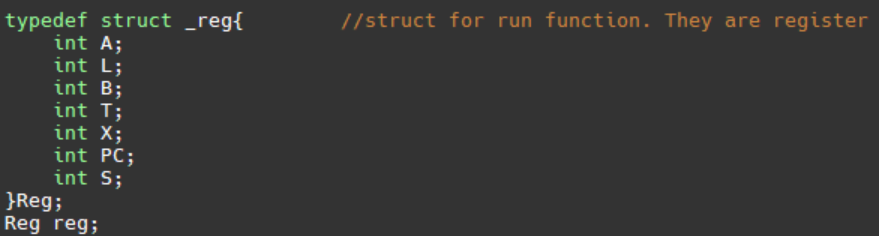
void print\_reg();

현재 register의 값을 출력해주는 함수

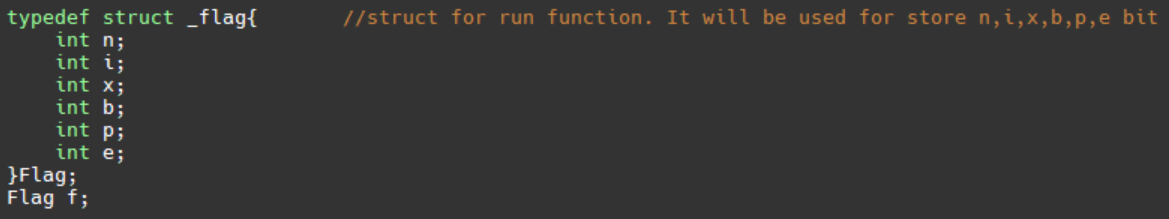
void run();

메모리에서 값을 읽어 프로그램을 실행시키기 위한 함수

4. 전역 변수 정의



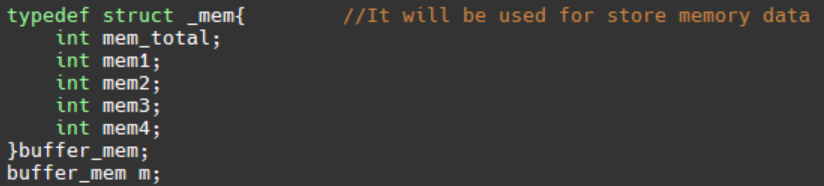
7개의 레지스터들을 관리하기 위한 구조체 & 구조체 변수



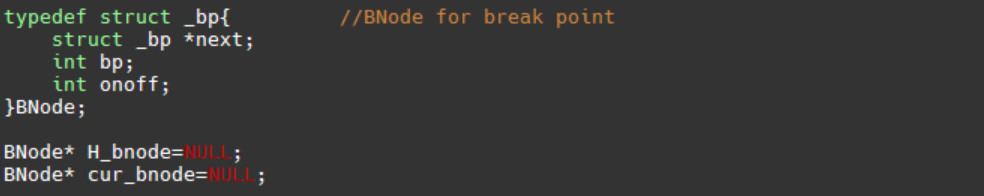
6개의 n I x b p e 비트를 관리하기 위한 구조체 & 구조체 변수



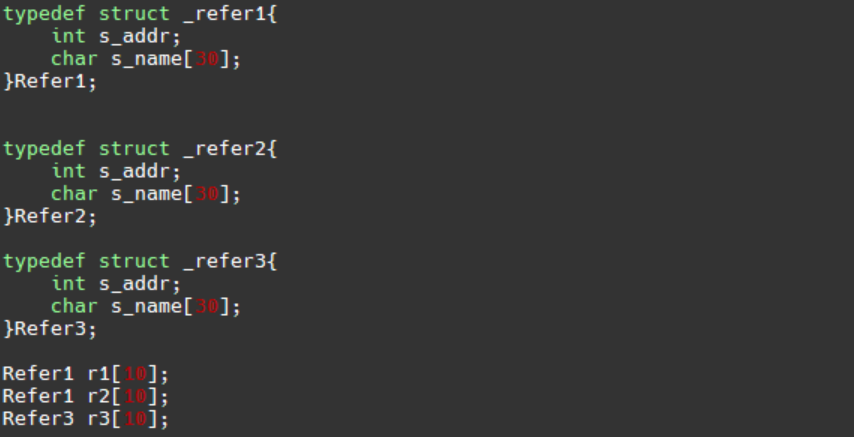
target address 값을 저장하기 위한 변수



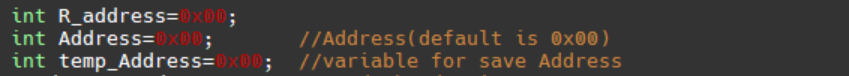
메모리에서 읽은 데이터들을 임시로 저장하고 계산 수행을 하기 위한 구조체 & 구조체 변수



break point를 저장하기 위한 구조체와 포인터 두개



R recode를 읽어서 저장해 놓기 위한 refer 구조체 1,2,3 & 구조체 배열



각종 Address를 저장해 놓기 위한 전역변수들



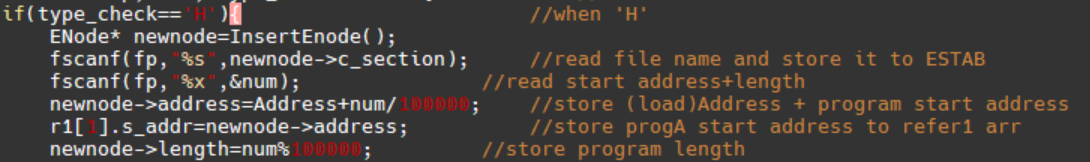
condition 판정을 위한 CC 변수와 루프나 데이터 read의 종료 판정을 위한 exit\_flag

5. 코드 설명

PASS1: (H, D, R recode의 처리)



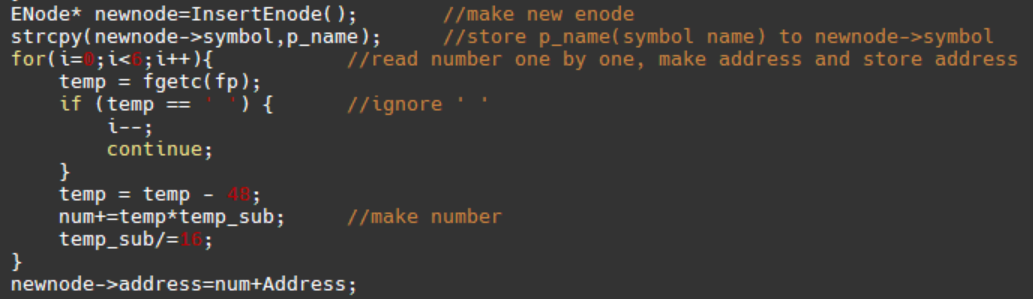
obj 파일의 각 줄 가장 앞 한글자를 읽는다. EOF에 도달할때까지 반복된다.



H recode 일 경우 :

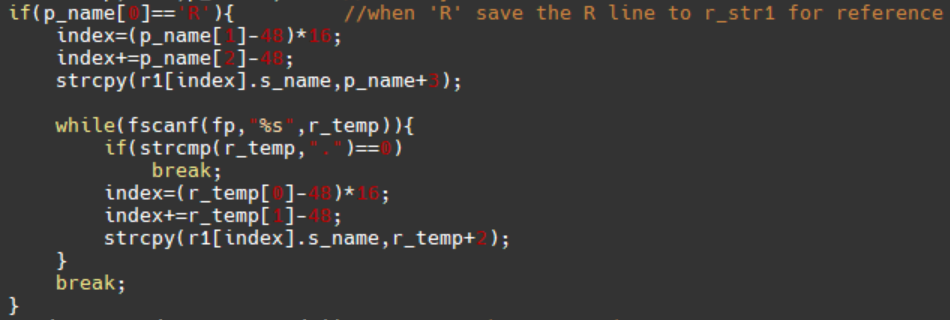
load map 출력을 위해 프로그램 이름과 시작주소를 node에 저장하고 list에 삽입한다.

D recode 일 경우 :



마찬가지로 load map 출력을 위해 해당 symbol들의 주소와 이름을 node에 저장하고 list에 삽입한다.

R recode의 경우:



refer 변수를 구조체 배열로 선언하였기 때문에 우선 index값을 저장해 준다.

그리고 symbol이름 을 r[index].s\_name에 저장한다.

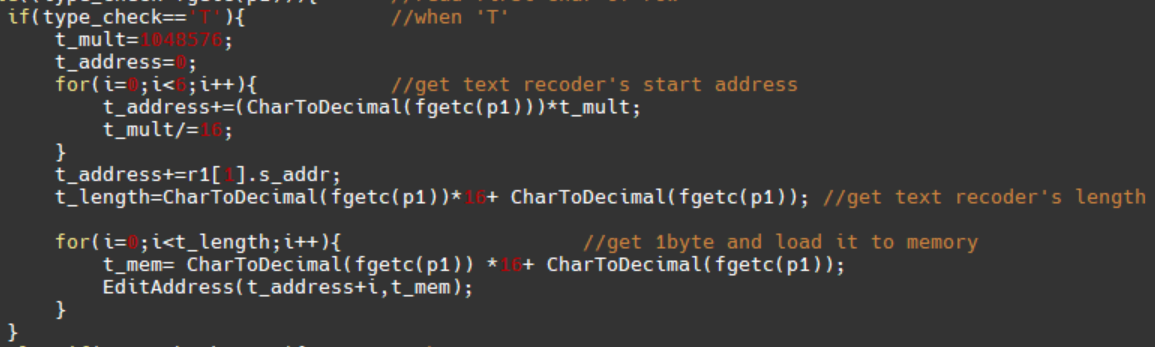
위 과정을 파일 마다 반복하면 PASS1이 종료된다.

PASS2: (T, M, E recode의 처리)



obj 파일의 첫 줄 한 글자를 읽는다.

T recode 일 경우:



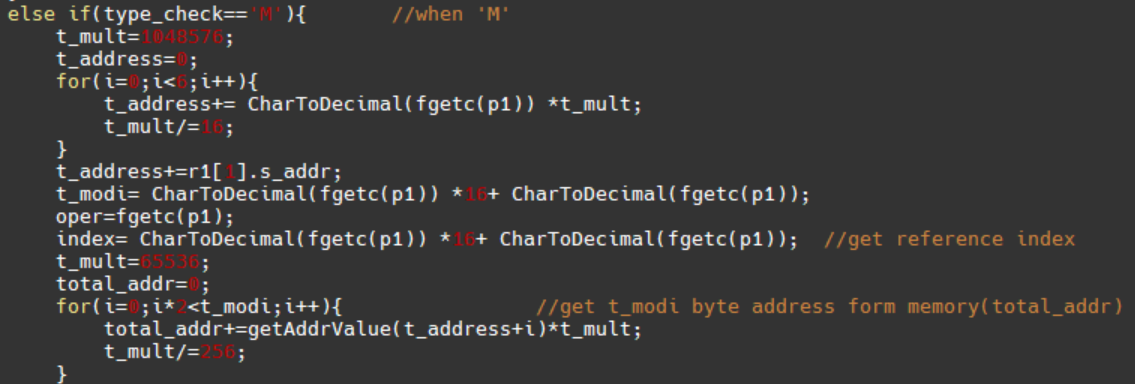
파일에서 6글자를 읽고 자리수에 맞게 계산해서 t\_address(text recode의 시작주소)를 구한다.

이어 두자리를 읽고, Text recode의 길이를 t\_length에 저장한다.

t\_length만큼 반복하는데, 파일에서 두글자를 읽고 t\_mem변수에 넣어준다. 그 다음 그 값을 메모

리의 t\_address+i 주소에 저장한다.

M recode 일 경우



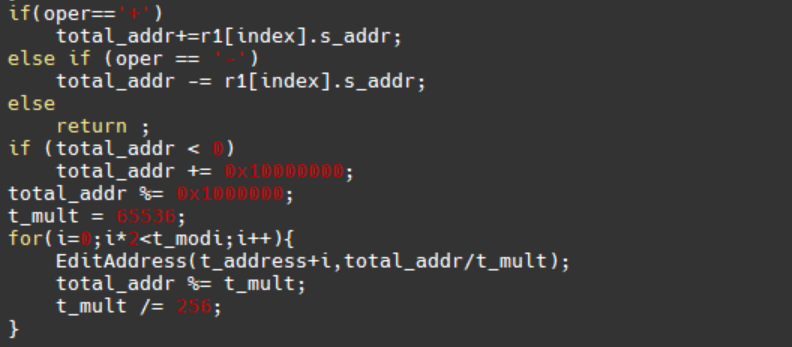
먼저 modification을 시작할 주소인 t\_address를 구한다.

그 다음 modification을 몇 개에 대해서 할건지에 대한 t\_modi 값을 구한다.

+인지 -인지 판별하기 위해 다음 값을 oper에 넣는다.

무엇을 더할것인지 reference number를 읽는다.

마지막으로 이미 메모리에 있던 값을 꺼내와서 total\_addr에 저장해준다.



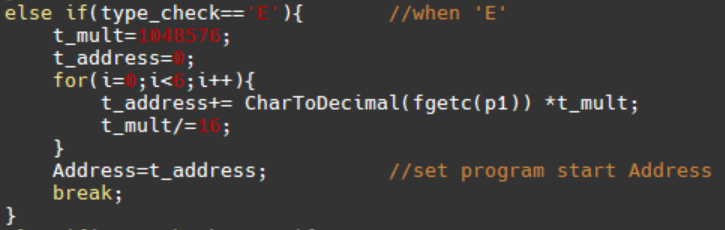
+,-일 경우 각각 해당되는값을 더하거나 빼준다.

이때 음수값의 처리를 위해서 만약 최종 결과가 음수일 경우 0x10000000을 더해준후 0x1000000

으로 나눠줌으로서 올바른 값을 찾는다.

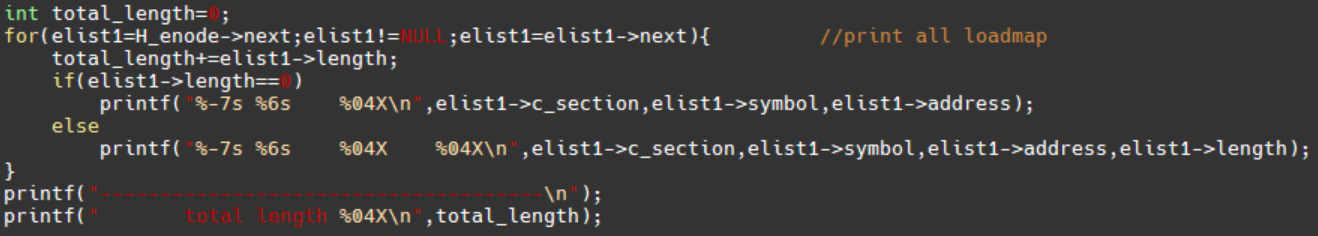
마지막으로 그 값을 다시 메모리에 올려주면 Modifiction과정이 종료된다.

E recode 일때:

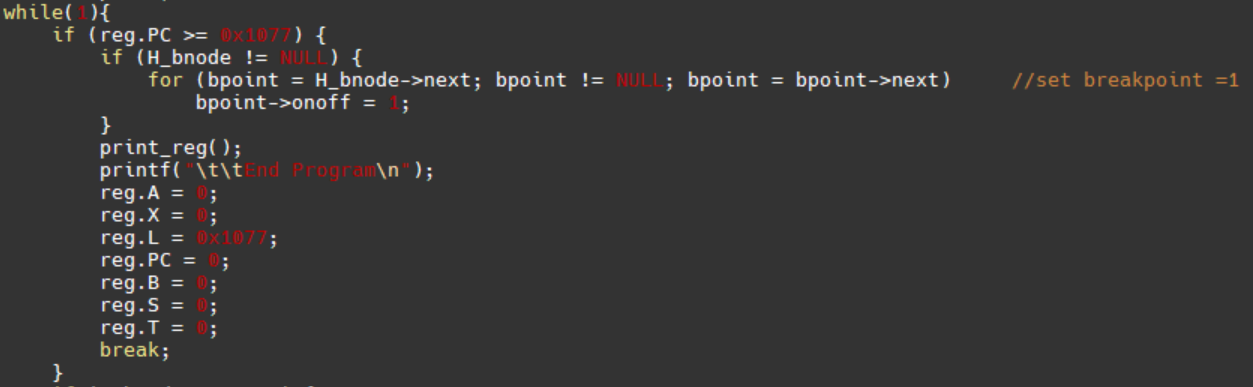


E레코드에 값이 있다면 그 값을 읽어서 그 값을 시작주소로 설정해준다. 그리고 반복문을 종료한다.

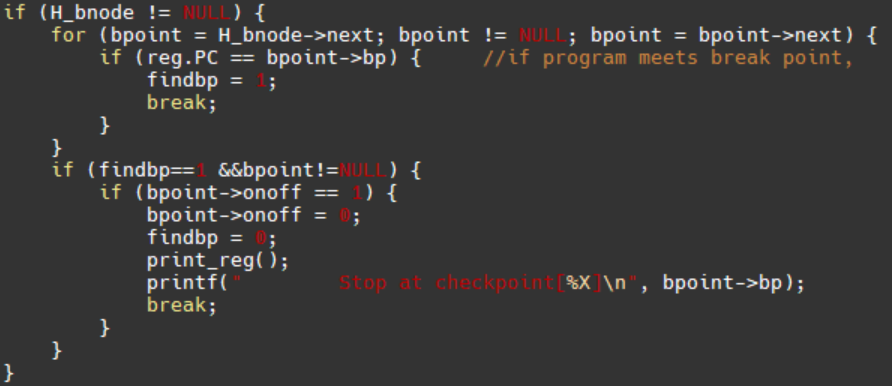
위 과정을 각 obj 파일에 대해 반복하면 PASS2가 종료된다. 마지막으로 loadmap을 출력하고 함수를 종료한다.



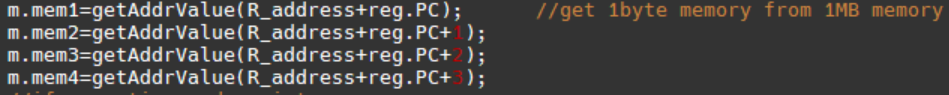
RUN



반복문의 종료조건. pc가 END에 도달했을때 (copy.obj에서는 0x1077) 모든 break point들을 다시 on 해주고 register들을 출력해주고 초기화 해준후 함수를 종료한다.



break point를 만난다면(findbp==1) break point를 off해주고 현재 register들을 출력한후 프로그램을 종료한다.

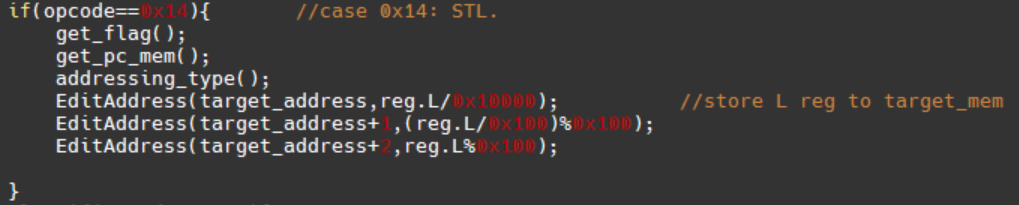


각 메모리에서 1byte씩 읽어서 차례대로 mem 1,2,3,4에 저장한다.



opcode를 얻는다. 뒤의 조건문들은 각 opcode의 경우에 따라서 적절한 명령을 수행하게 하였다.

STL일 때:



get\_flag(), get\_pc\_mem(), addressing\_type() 이 세 함수를 거쳐 target address가 완성된다.

자리수에 맞춰서 target address에 L register의 값을 넣어준다.

나머지 명령도 동일하다. 전부다 쓰기보다는 각 명령별로 따로 처리한 부분을 추가로 설명하겠다.

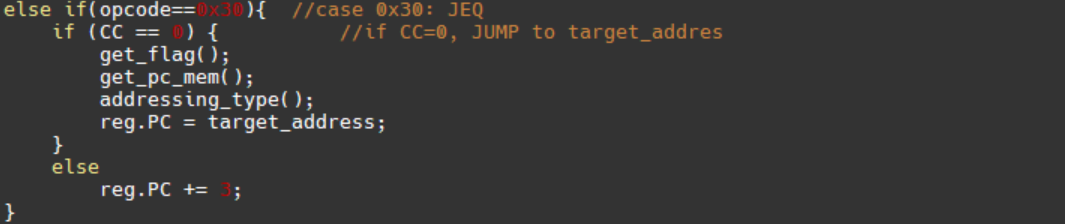
JSUB의 처리:



L register에 return 주소(현재 PC값)을 저장하고

PC를 target address로 바꾼다(JUMP 한다.)

JEQ의 처리:



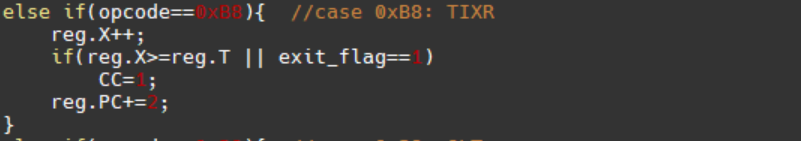
CC에 따라서 해당되면 JUMP 아니면 다음 명령어로(reg.PC+=3);

RSUB의 처리:



원래 함수로 return 해줌 (reg.PC=저장해놓았던 L값)

TIXR의 처리:



값 비교후 조건에 맞으면 CC=1로 설정