**과목명: 시스템프로그래밍**

**1분반**

**<<Project #2>>**

**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

본 프로그램은 시스템 프로그래밍의 두번째 프로젝트로 첫번째 프로젝트에 더해서 구현해야 할 명령은 크게 3가지이다.

1. filename.asm 파일을 읽고 assemble시켜서 filename.lst, filename.obj 파일을 만드는 assemble filename.asm 명령어

2. 해당 파일의 내용을 출력해주는 type filename 명령어

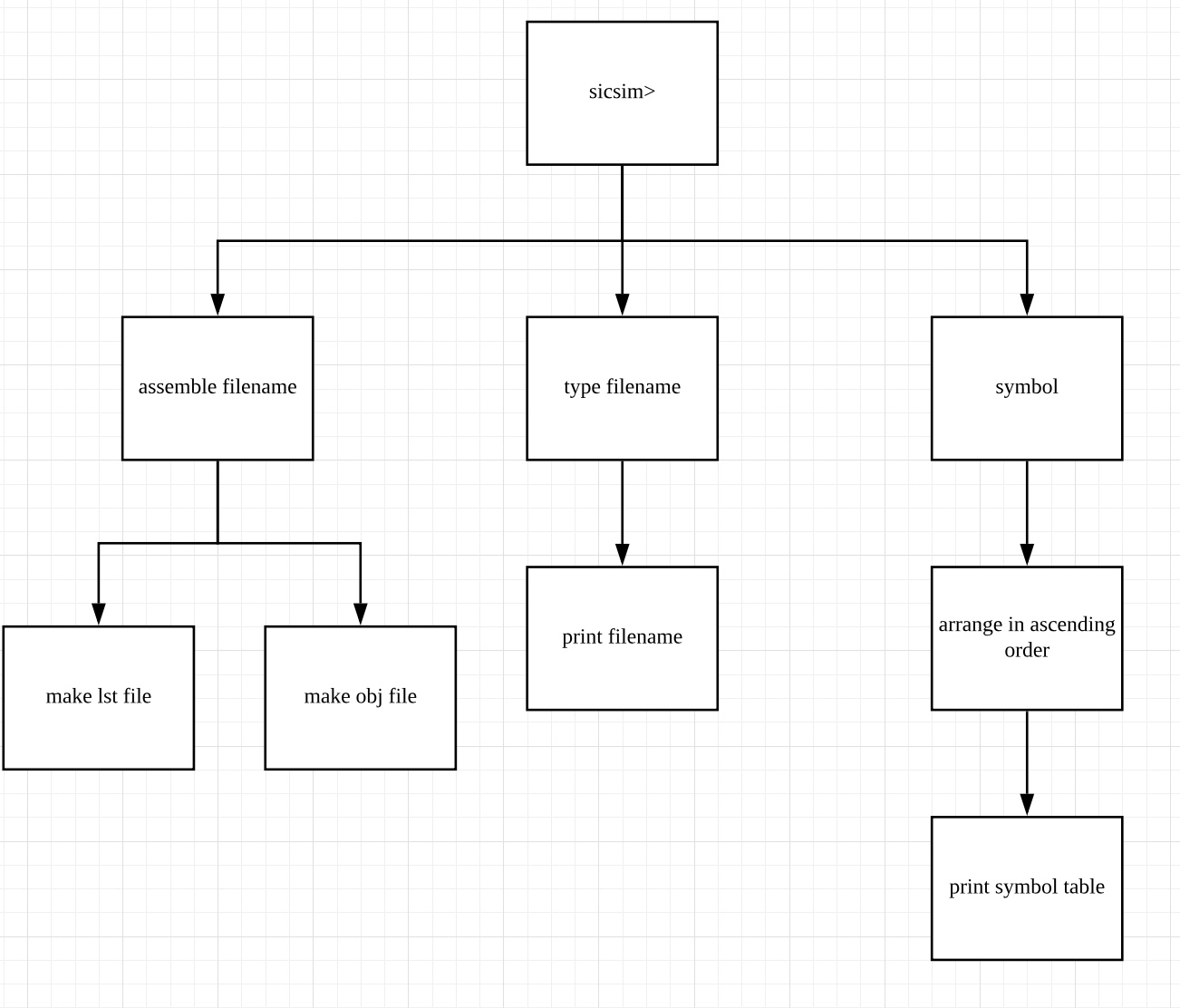
3. symbol을 순서대로 출력해주는 symbol 명령어

전체적인 프로그램 흐름도는 프로그램 1과 동일하다. 다만 추가적으로 linkedlist가 필요해서 그 부분을 구현하였고, 프로그램이 커지다보니 관리가 불편해서 이번프로젝트의 거의 모든내용은 20161571\_1.h라는 새로운 헤더파일을 만들어서 모두 그곳에 작성하였다.

2. 프로그램 설명

2-1. 프로그램 흐름도

프로젝트1을 제외하고 이번에 구현한 내용을 중점으로 작성하였다.



3. 모듈정의

3.1 정의한 모듈에 대해 간략적인 설명

1. void Initasm(asmList\* asmlist)

asmLinkedlist의 초기화를 해주는 함수

2. int makelist(asmList\* asmlist,char \*)

char로 전달된 이름의 asm파일을 열고 assemble 과정을 수행하는 함수. 성공적으로 수행하였다면 .obj .lst 파일이 생성되고 return 1된다. 해당하는 파일이 없어서 실패하였다면 return -1이 되고 생성되는 파일은 없다. assemble과정에서 asm파일의 data가 유효하지 않아서 실패하였다면 return 0이 되고 생성되는 파일은 역시 없다.

3. int printType(char\*)

전달된 문자열에 해당하는 파일의 내용을 출력하고 return 1한다. 해당파일이 없으면 에러메세지를 출력하고 return -1을 한다

4. void symbol(asmList\*)

assemble과정을 통해 생성된 symbol table을 사전순서대로 출력한다.

4. 전역 변수 정의

assemble 명령을 담기위한 linkedlist구조체를 제외하면 전역변수로 선언한 것은 없다.

5. 코드 설명

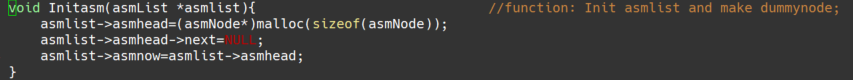
처음 프로젝트를 봤을 때, 일단 asm파일을 모두 linkedlist에 옮겨 담은다음에 그곳에서 모든 계산을 마치고 다시 lst , obj파일을 만들면 간단하게 해결할 수 있을 것 같았다.





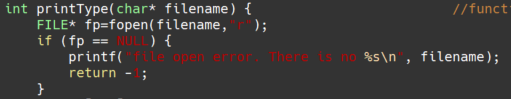
그래서 우선 그 내용을 담을 linkedlist를 하나 선언하였다.

-Initasm 함수

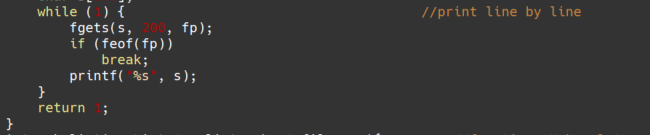


asmList를 초기화 해주는 함수로 앞서 프로젝트 1에서 구현한 linkedlist의 초기화와 동일하다.

-printType 함수

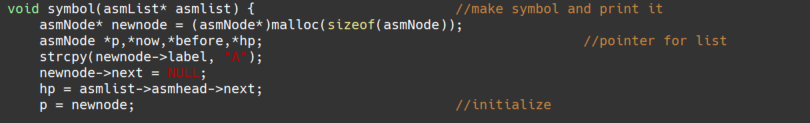


문자열을 하나 전달받아서 그 문자열에 해당하는 파일을 출력해주는 함수이다. 먼저 해당 파일이 없다면 에러 메시지를 출력하고 -1을 return한다.



해당파일이 존재한다면 한 줄씩 읽어서 출력해준다. 그런데 계속 마지막줄이 두번나와서 왜 그러나 했더니, while문의 조건판단에 이 문장을 넣으면 fgets를 한번더 한후에야 feof가 1이 되기때문에 while문이 한번더 돌아가서 그랬던 것을 알았다. feof(fp)로 파일의 끝을 판단할경우 fgets이후에 이 문장을 넣어줘야 정상적으로 작동하는것을 알았다. 1을 return해준다

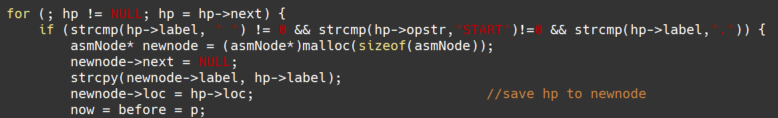
-symbol 함수



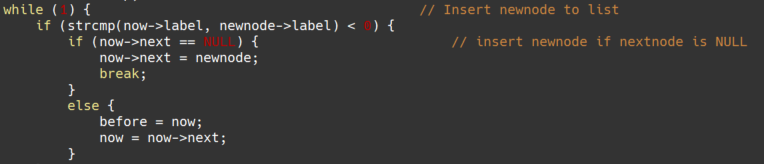
assemble과정에서 만들어진 symbol table을 label문자의 사전순서대로 출력해주는 함수이다.

새로운 연결리스트를 만들고, 기존의 리스트에서 값을 하나씩 읽어와서 애초에 순서대로 삽입ㅎ할 예정이다.

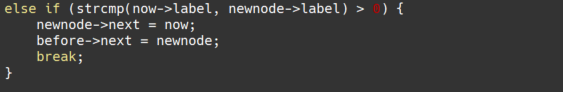
이를위해서 새로운 리스트의 첫부분이 될 “A”를 data로 가지는 더미노드를 만들었고, hp가 기존의 노드를 가리키게 설정하였다.



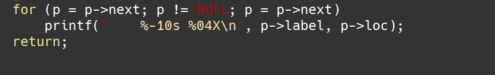
hp를 하나씩 넘겨가면서 값을 비교한다. 유효한 lable을 가지고 있을때에만 실행되게 하였고, label값이 유효하다면 새로운 node를 만들어서 그곳에 label data를 복사한다. 그리고 now before은 각각 현노드 전노드를 가리켜주는 포인터이다.



비교문이다. 리스트의 현재노드보다 삽입되어야 할 node값이 더 크다면 한칸 우측으로 보낸후 다시 탐색한다. 만약 첫 노드라면 그냥 삽입한다.

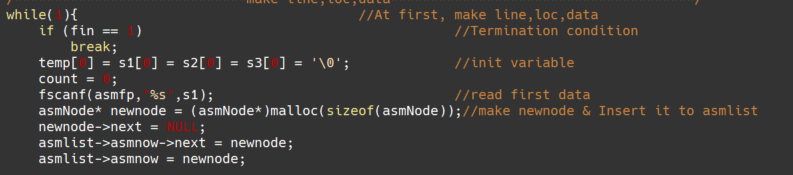


만약 리스트의 현재노드보다 삽입되여야 할 node값이 더 작다면 그 자리에 삽입한다.



마지막으로 리스트를 총 출력해준다. p는 dummy노드를 가르키고있다.

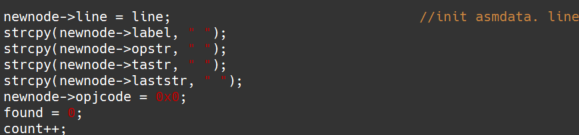
이제 이번프로젝트의 핵심인 makelist부분이다.



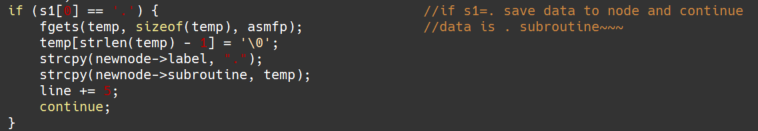
while문으로 종료조건인 fin==1일때까지 반복한다.

우선 모든 배열, 변수들을 초기화하고 newnode를 생성해서 리스트에 삽입한다.

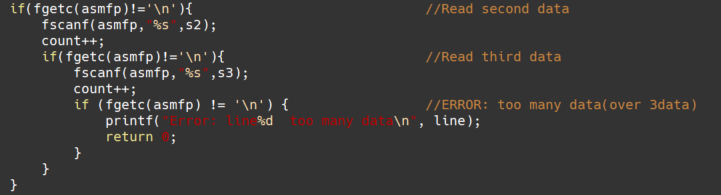
그리고 s1에 인자를 하나 입력받는다(asm file에서)



그 다음은 삽입한 newnode의 초기화를 한다. line은 정상적으로 넣어주고 나머지 값들은 모두 공백으로 처리한다.



asm file 부분에서 . subroutine~ 이라고 되어있는 부분을 처리하기 위한 코드이다. 만약 맨앞에 .을 만나면 통째로 한줄을 읽어버리고 그 한줄을 node에 저장한다. 그리고 line수를 증가시킨다음 continue 문을 통해서 다음 반복문을 시작한다.



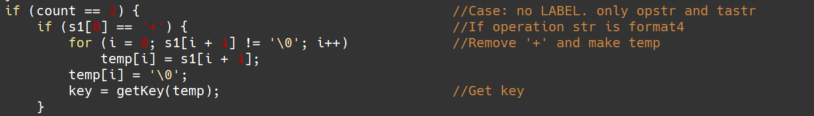
계속해서 data를 asm file에서 읽는 과정이다. 만약 데이터를 3개보다 더 많이 읽었다면 error처리를 해줄수 있게 하였고, s1,s2,s3에 해당데이터들을 저장하고 count변수를 통해서 총 몇 개의 데이터를 읽었는지 알 수 있게 하였다.



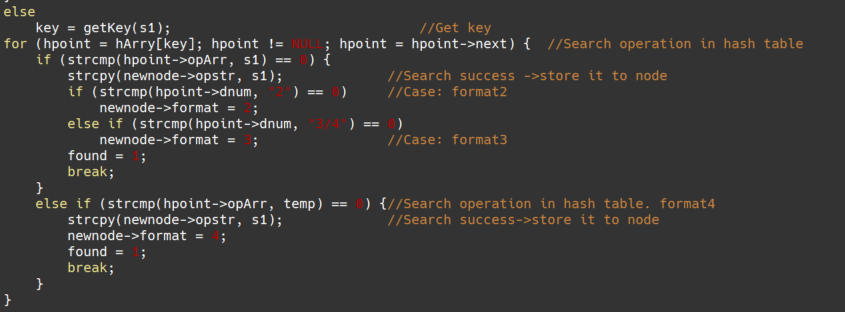
START 다음에 나오는 값이 시작주소임으로 그 값으로 초기화한다.



인자가 1개인경우는 RSUB밖에 없다. 해당되는 위치에 값을 넣는다.

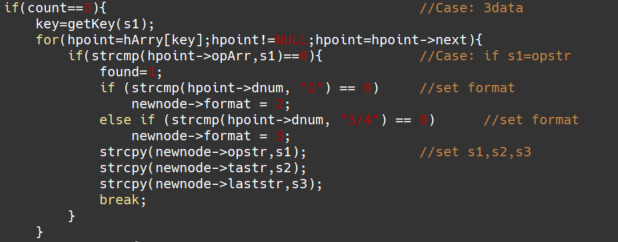


인자가 2개인경우, 우선 format 4인지 확인하고 4라면 +를 제거한 operation의 key값을 찾아준다.



아니라면 평범하게 key값을 찾아주고 앞프로젝트에서 구현한 hash table을 통해 해당 명령의 opcode를 찾는다. 이때 format역시 같이 찾아서 저장해준다.

count3일 경우도 위와 동일한데, 다만 인자들의 처리가 좀더 복잡하다는 점에 있다.

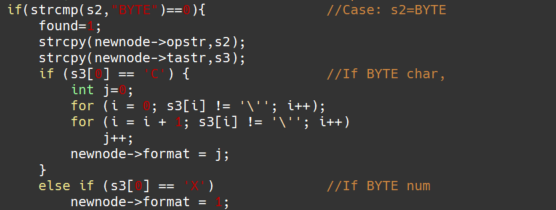


우선 첫번째 값이 operation이라면 바로 탐색을 시도하고 찾으면 값을 저장하고 found=1로 바꾼다

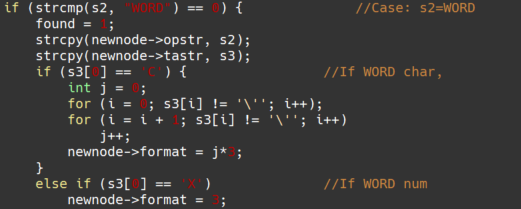


탐색에 실패했으면 첫번째 값은 label이라는 뜻이고 두번째 값으로 탐색을 시도한다.

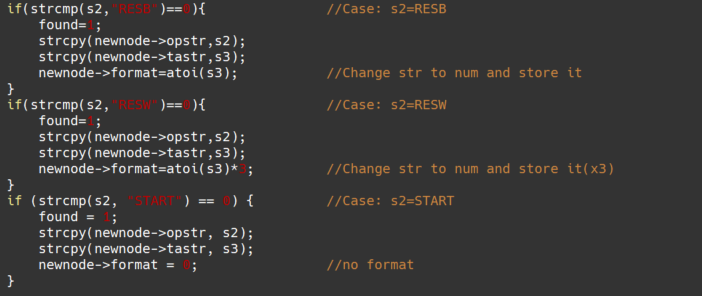
만약 두번째에도 탐색이 실패했으면 여러 special case인 경우를 생각해봐야한다.



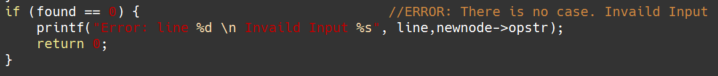
BYTE인 경우 char일 경우와 X일 경우를 구분해서 format 계산을 해준다.



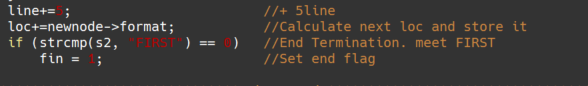
WORD인 경우 char일 경우와 X일 경우를 구분해서 format 계산을 해준다. byte와 거의 동일하다.



기타 변수들의 경우에도 해당node에 값을 넣어주고 각자 알맞은 format값을 할당해준다.



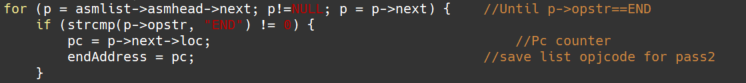
여기까지 했는데도 못찾았다면 Error 처리한다. 그리고 0을 return 한다.



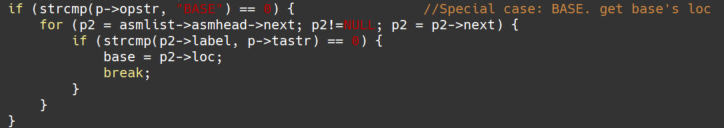
마지막으로 line과 loc를 설정한다. 이값들은 다음 node에 삽입될 것이다.

그리고 만약 FIRST를 찾았다면 asm 파일의 끝을 의미함으로 종료 flag인 fin=1로 설정해준다.

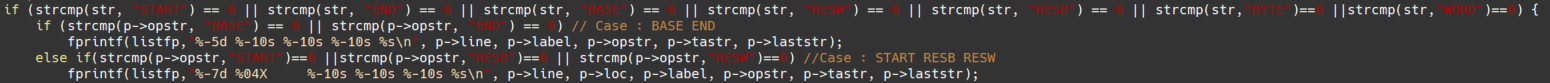
우리는 이제 위에서 만든 loc line이 +기존 asm 파일 내용이 들어가 있는 node를 가지고 opcode를 생성해서 기존 node의 opcode부분에 삽입할 것이다.



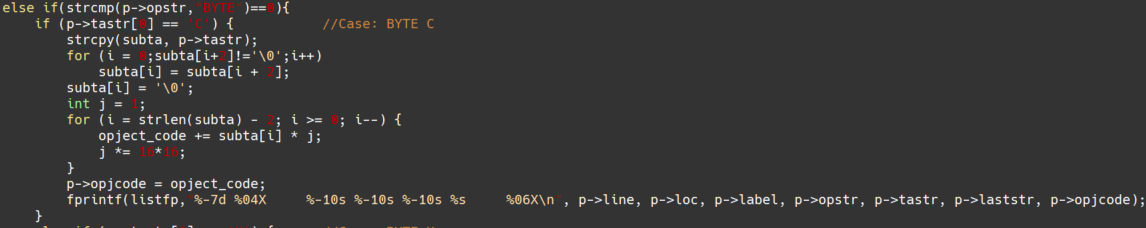
반복문을 시작한다. 종료조건은 END를 만났을때이고 endaddress가 obj파일 생성에 필요함으로 저장해 놓을수 있도록 하였다.

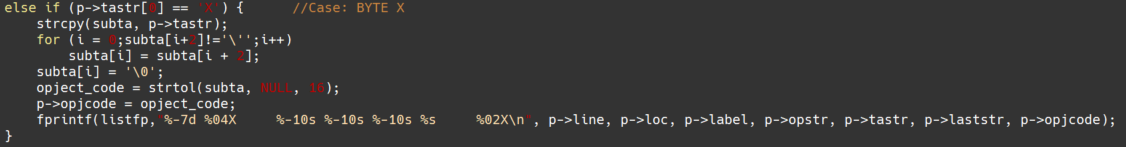


Base relative를 위한 Base의 loc를 저장해 놓자.



먼저 special case를 확인해 주자. 해당 case들은 opcode 구하는 방법이 다름으로 먼저 처리해 주자.

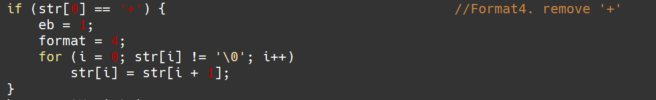




BYTE일 때 X일때와 C일때를 나누어서 opcode를 fprint해준다.

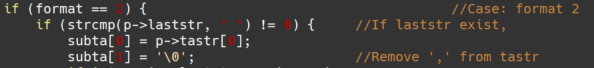
WORD도 마찬가지로 하면된다.

다음은 일반적인 경우에서의 opcode 생성법을 보자.

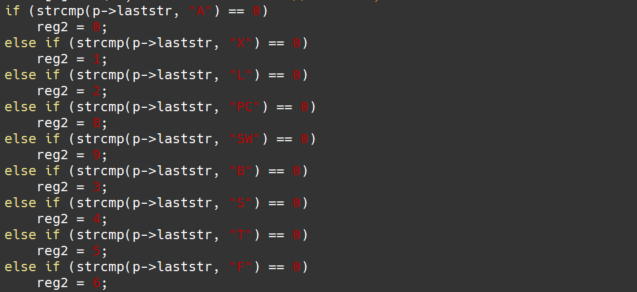


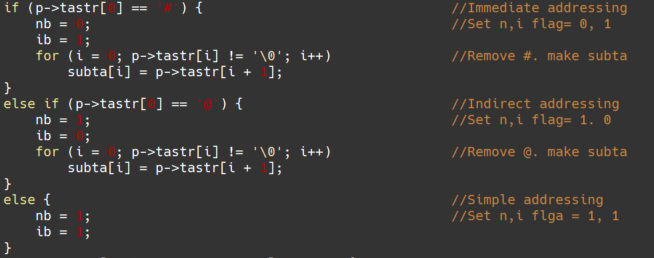
먼저 format 4인지 확인해주고 탐색을 위해+를 제거해준다

위에서 했던것과 동일하게 hash table 탐색후 각 format 에 따라 조건문에 들어간다.

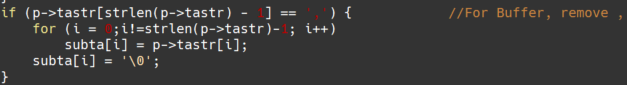


format==2일 때 우선 register인자가 1개인지 2개인지 확인한후 각 register를 해당하는 code로 바꾸어준다.

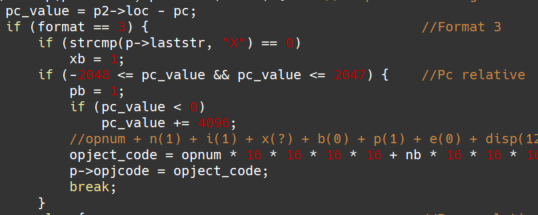


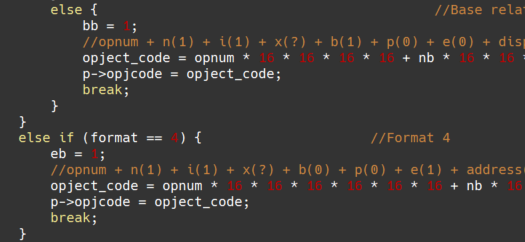


addressing의 종류를 판단한다. immediate/ indirect/ simple



Buffer의 경우에는 ,가 있기에 ,를 제거해준다.



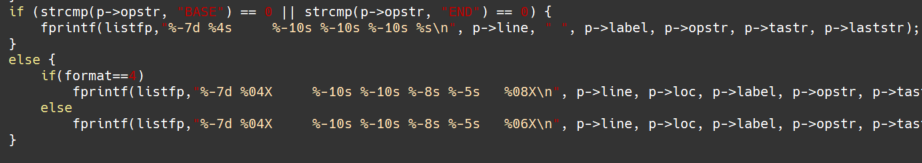


asmlist를 탐색해가며 pc인지 base인지 판단한다. 핵심코드는 위와같다. Pc의 차이를 구하고 값의 범위가 유효하면 pc relative아니면 base로 판단한다.

그리고 자리수, format에 맞춰서 opcode에 해당 값을 저장한다.

위 예시는 simple addressing의 경우지만 indirect, immediate의 경우에도 각 nixbpe flag의 값만

조금씩 다를뿐 동일하다.



마지막으로 lst파일에 모두 출력해준다.

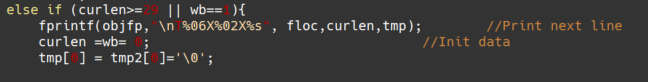
다음으로는 위에서 data들이 저장된 node를 이용해서 obj파일을 만들것이다.



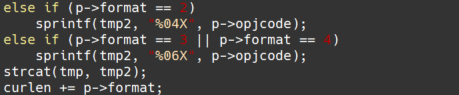
첫번째 라인인 H라인의 출력



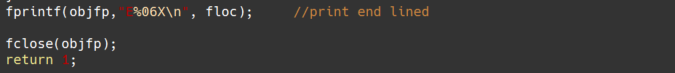
해당 값들을 만나면 멈춰줘야 하기 때문에 종료flag설정



종료해야 한다면 그때 한줄을 출력해준다.



평소에는 값들을 tmp2로 읽어서 tmp에 붙여준다. 길에 한줄을 계속 만드는 과정.



마지막으로 endline을 출력해주고 1을 return 해준다.