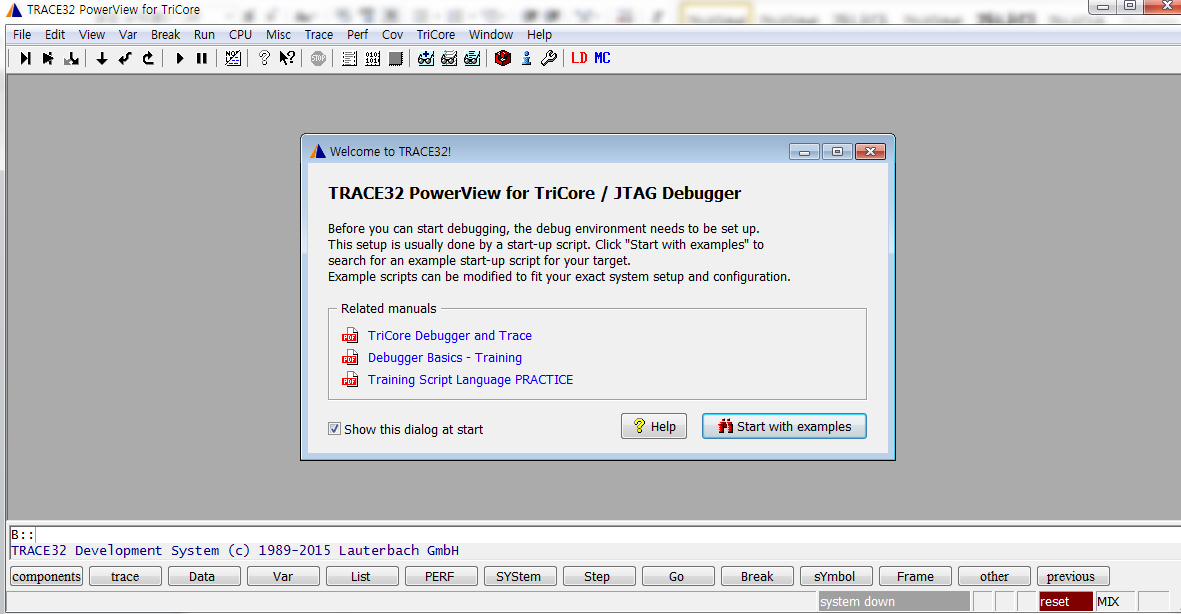
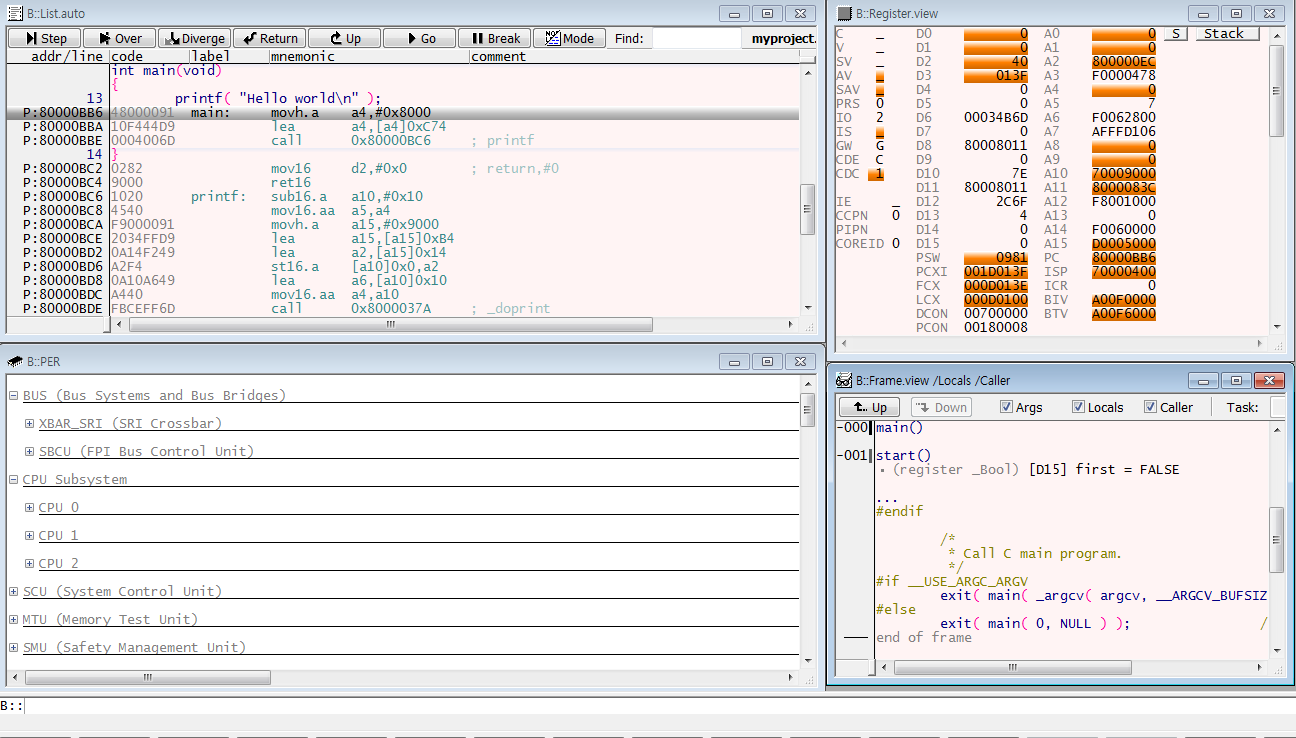
Tasking IDE를 실행하고 C/c+프로젝트를 생성하고

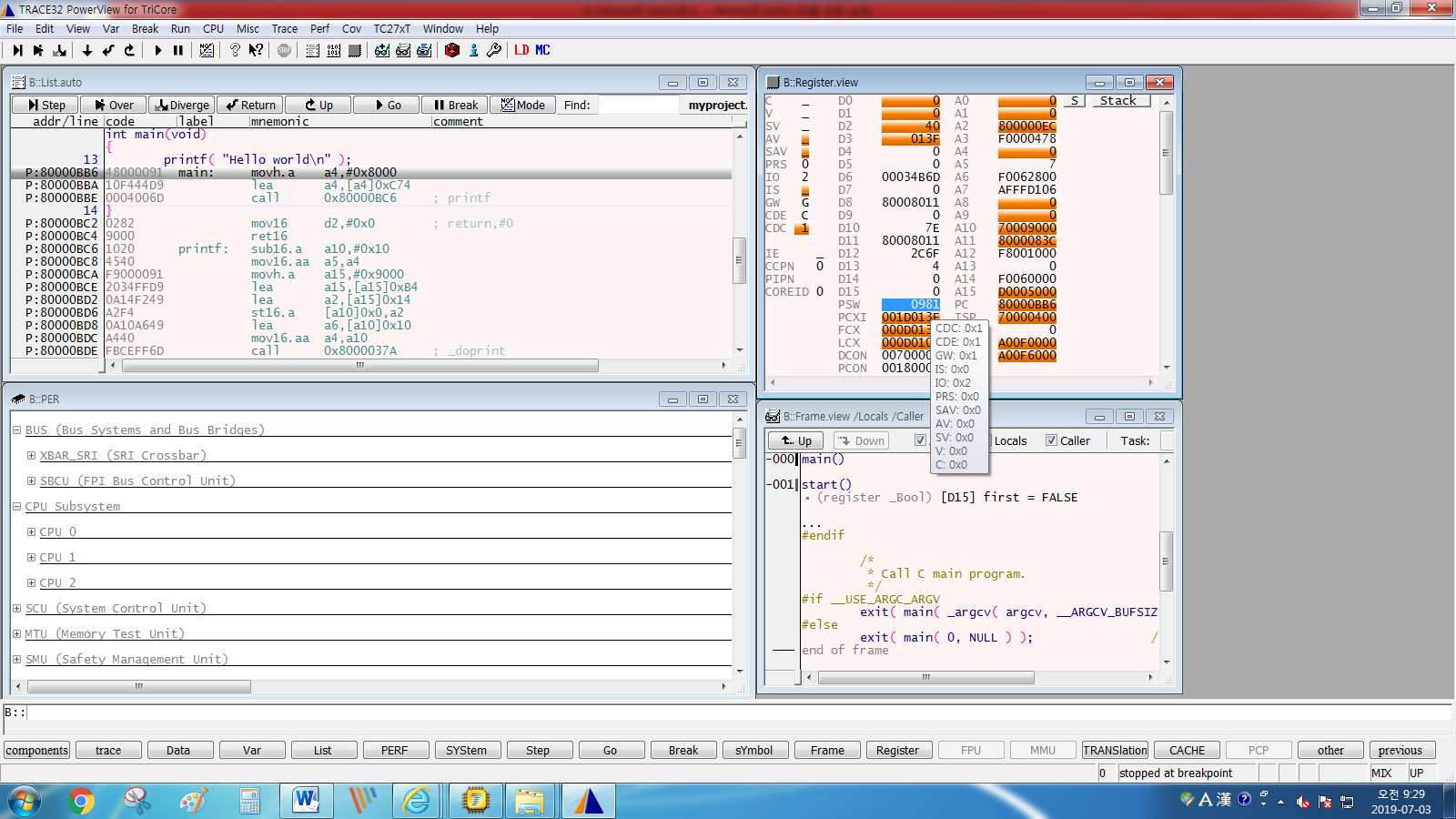
빌드하고 c:\에 T32 압출풀고, temp 폴더 만들어놓고

C:\T32\bin\windows\t32mtc.exe 기억할것 🡺 실행





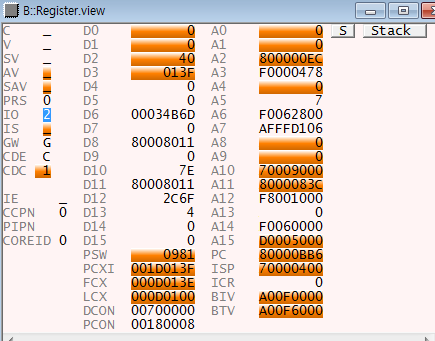
🡪해당 화면 나오면 준비됨



Context(local로 사용하는 피연산자)

Switching :함수 호출, 인터럽트 시 백업하는 것들, 인터럽트 수행하고 restore하고

↑



🡪여기선 수정이 되나 code를 돌리면 permission 때문에 돌아가지 않을 수 있음

그래서 디버거가 필요

영역이 RAM(70000000)이면 명령어도 writing할수 잇다 🡪 A\_0의 값이 0xF로 변경

But global영역이라 쉽게 변경되면 안되므로 수정으로부터 보호(디버거가 피드백)🡨permission문제

Trap :메모리에 data writing시 문제가 생기면 엉뚱한 경로(permission 잘못된 경우)로 전달됨을 보임, 문제를 알림

ABI(EABI)문서를 보자🡪standard calling convention

Calling convention 🡪 스택, resistor를 어떻게 관리할지에 대한 내용

함수 호출시 레지스터를 통해서 함수를 calling

레지스터 사용갯수 범위를 초과할 경우 스택을 활용하여 확장?

Argument

Data나 포인터가 넘어갈수 잇는데 이에 대한 레지스터가 정해져 잇다

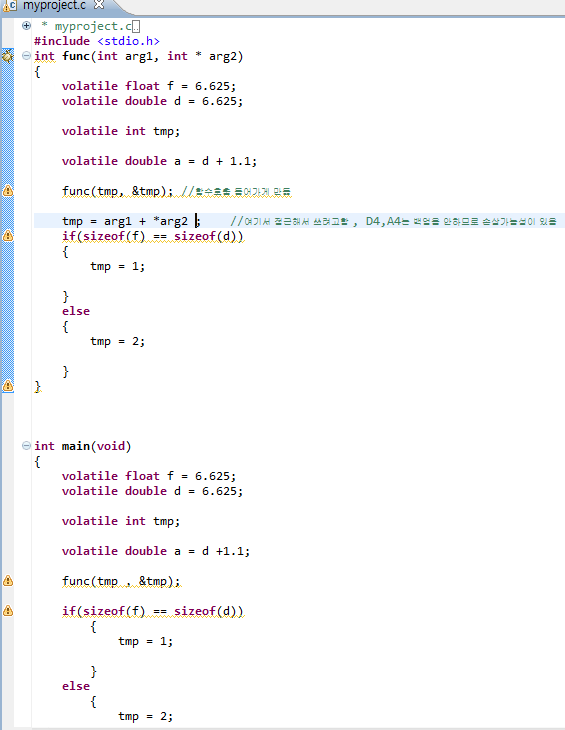
Non포인터:data(자료type 등 / lower context) : 백업 없이 사용하면 끝

* 포인터 4개, data 4개 씩 레지스터 사용 / 그 이상의 arg를 사용할 경우 스택 사용(속도저하)

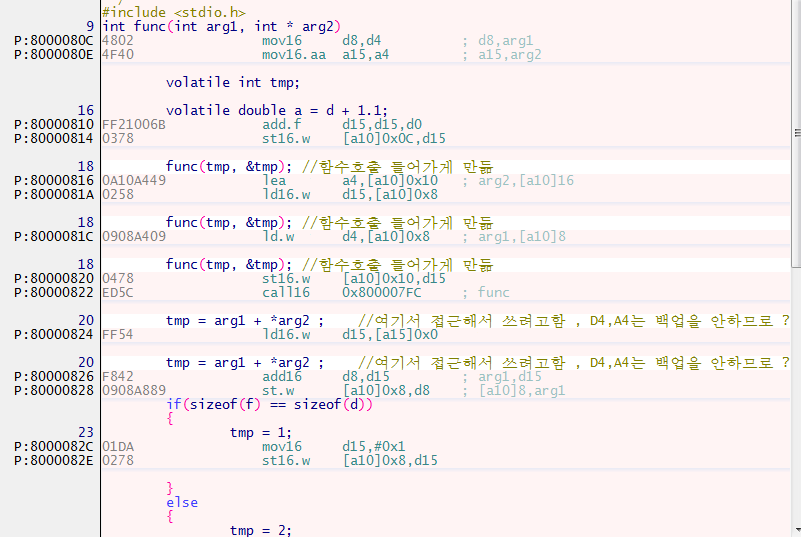
이것들은 Argument 패싱하면 할일 끝

배열은 문법적으로 return 불가, 구조체는 return해버림, 파라미터로 쓰면 스택에 저장되어 속도가겁나게 떨어짐 그래서 포인터로 활용

Aurix에서는 함수호출 하고 arg넘겨주면 그걸로 끝 (피연산자 등 알아서 사용될수 잇게끔 code는 알아서 정리해줌)

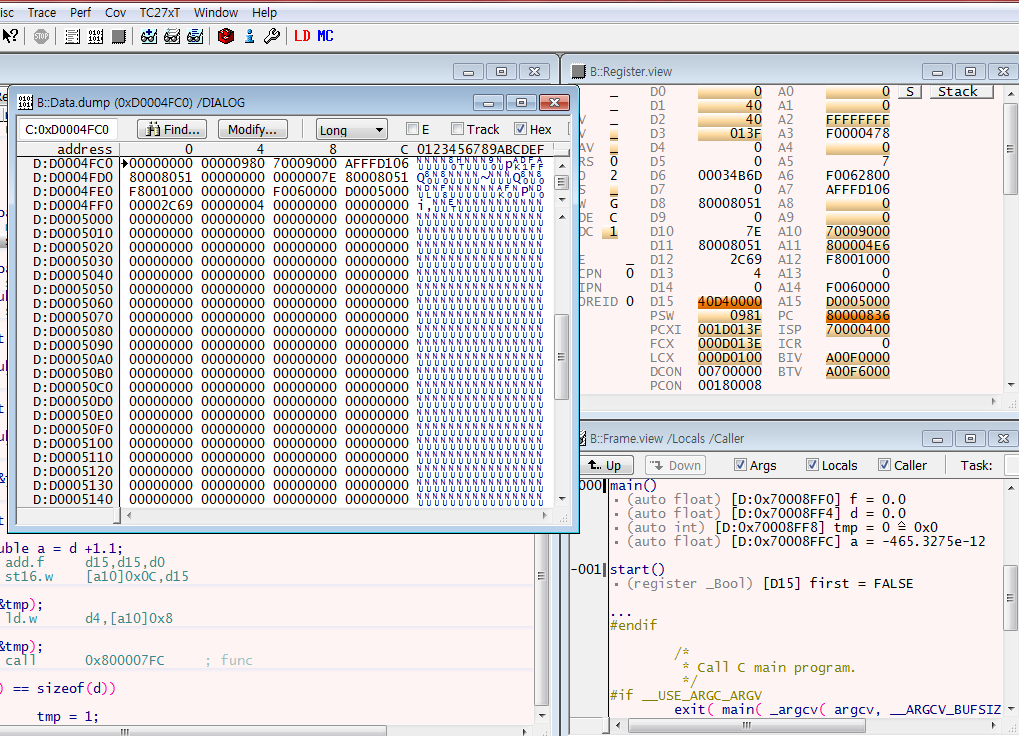


이걸 LD

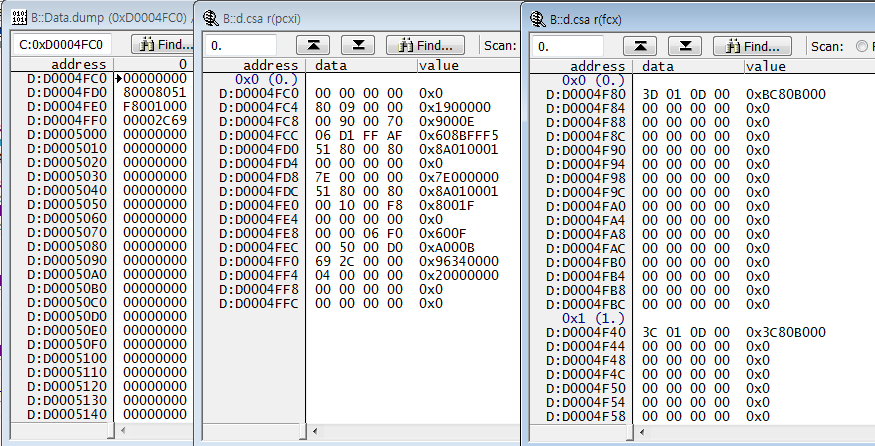
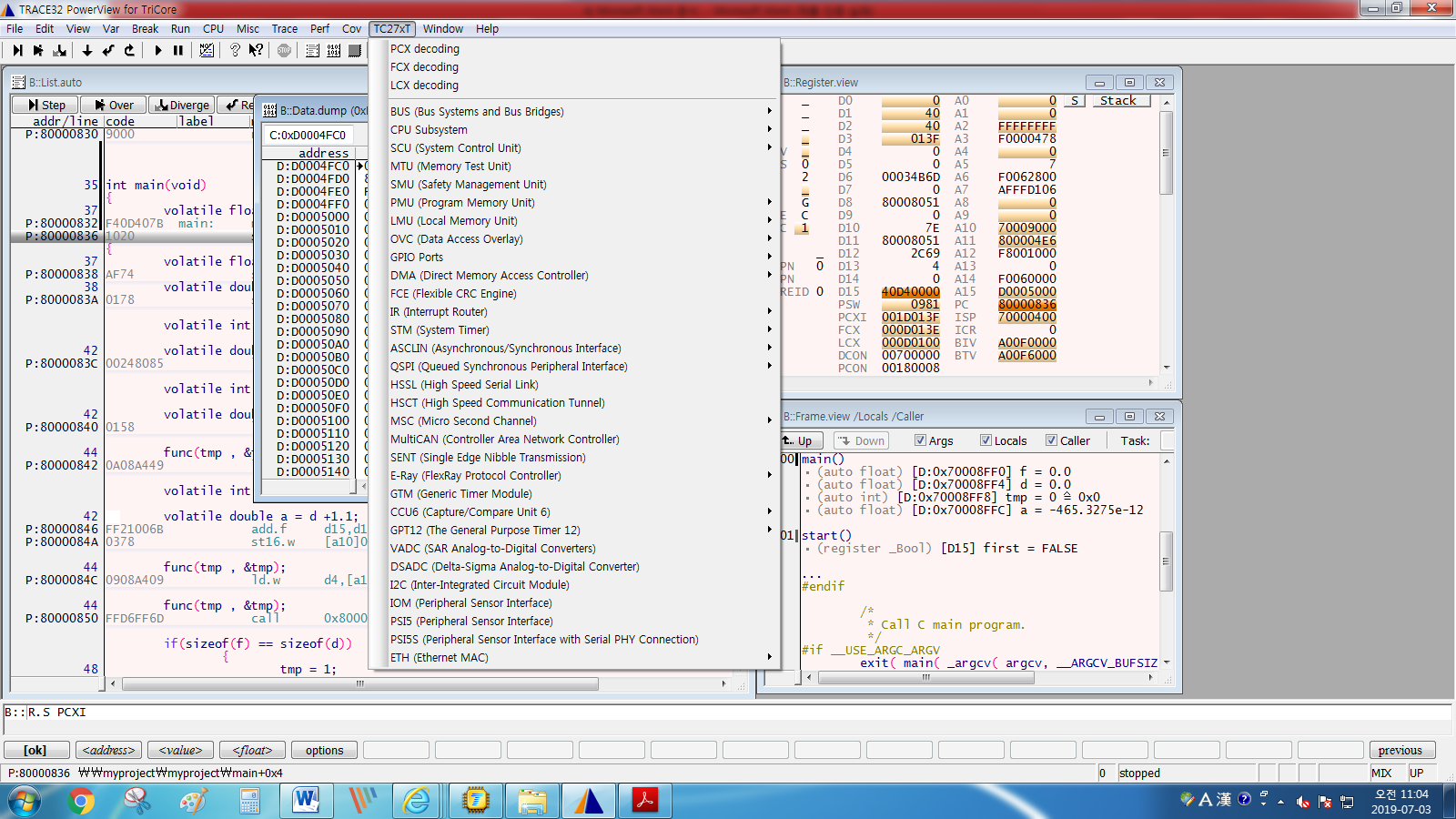


🡪함수 호출 정의 block에서 함수호출시 main함수에서 활용해야 할수 있음에 대비 중요data로 백업화 (a15 레지스터)

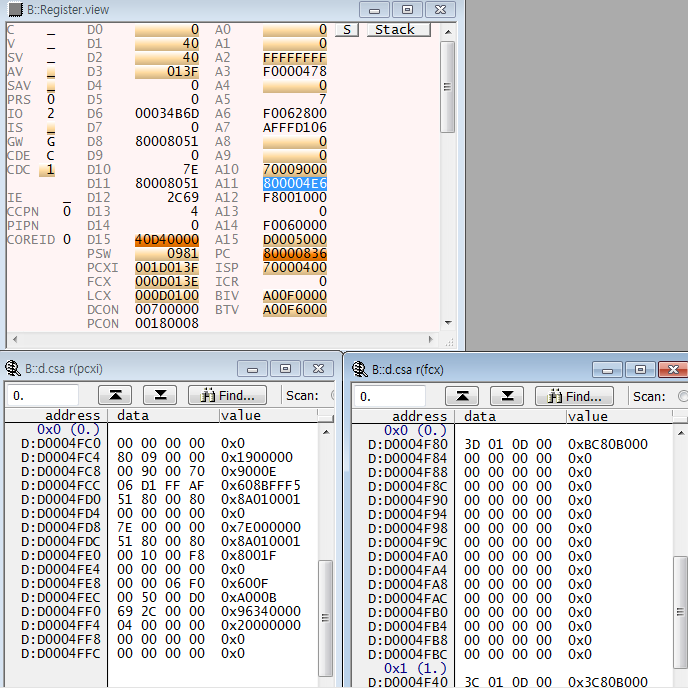
↓

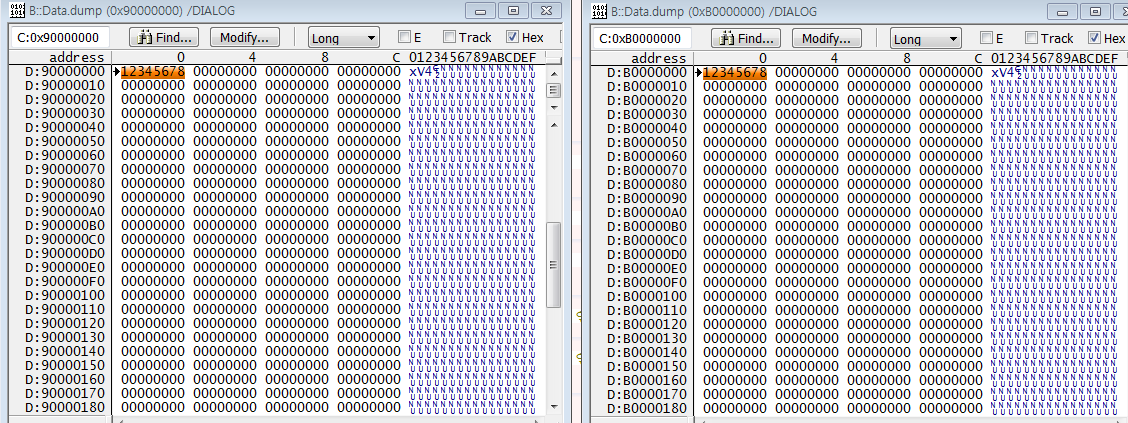


스택 뷰의 주소값을 물리주소로 가공

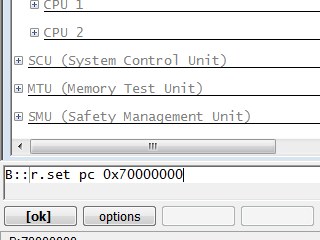


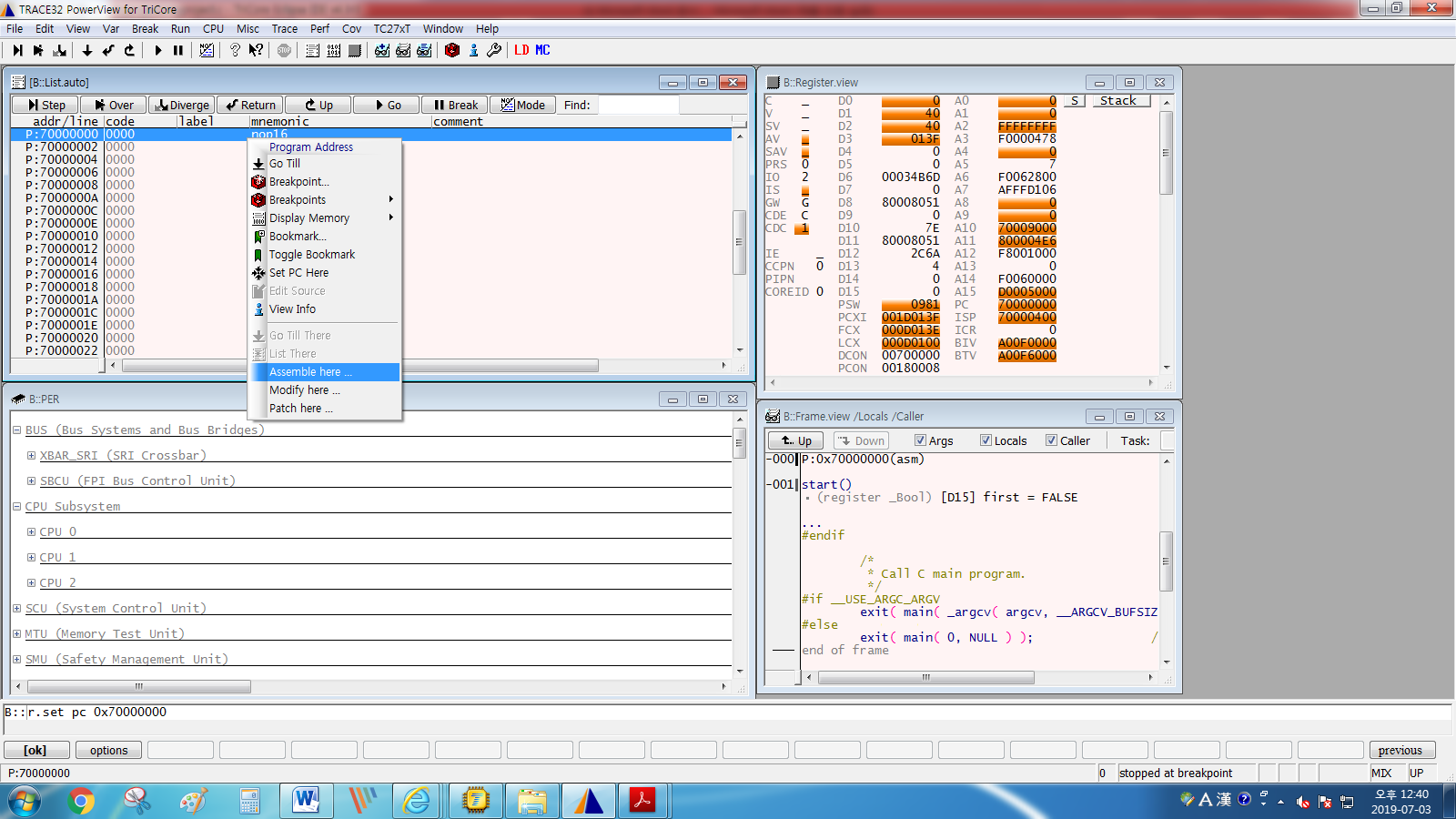
Little endian



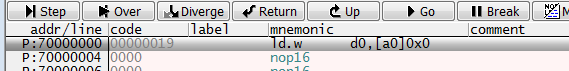


* Seg9 에 data를 쓰면 segB에는 cach를 거치지 않고 bypass되어 data 공유

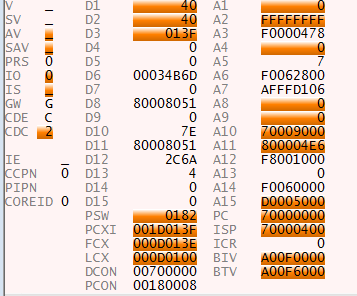




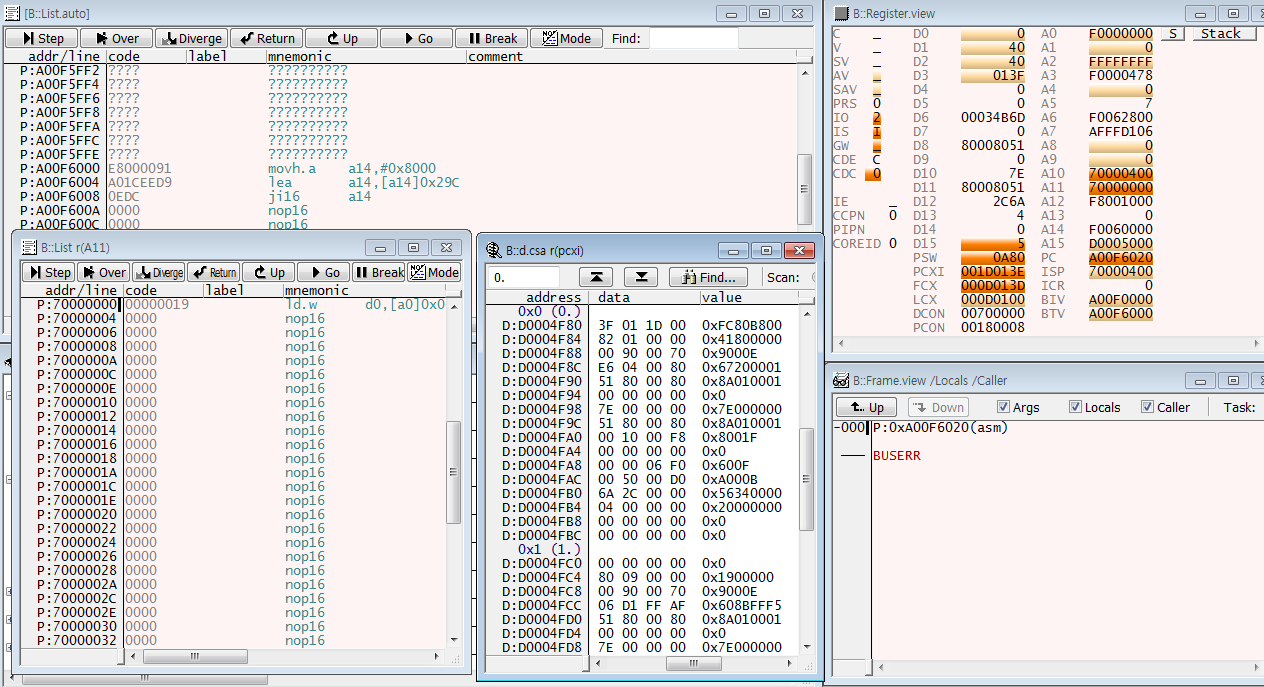


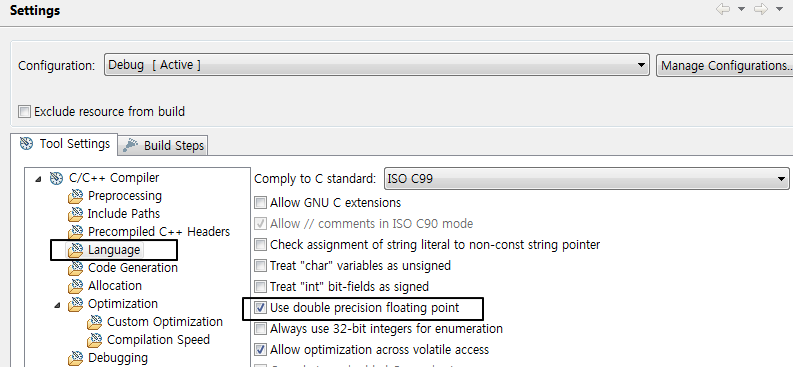


//peripheral 영역



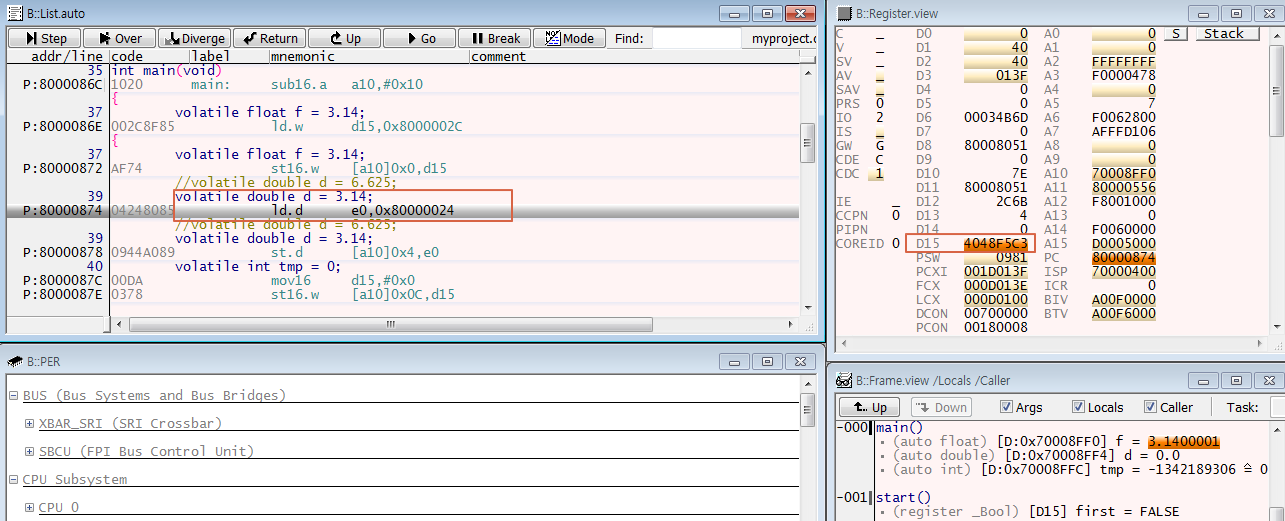
🡪User0(IO:0)

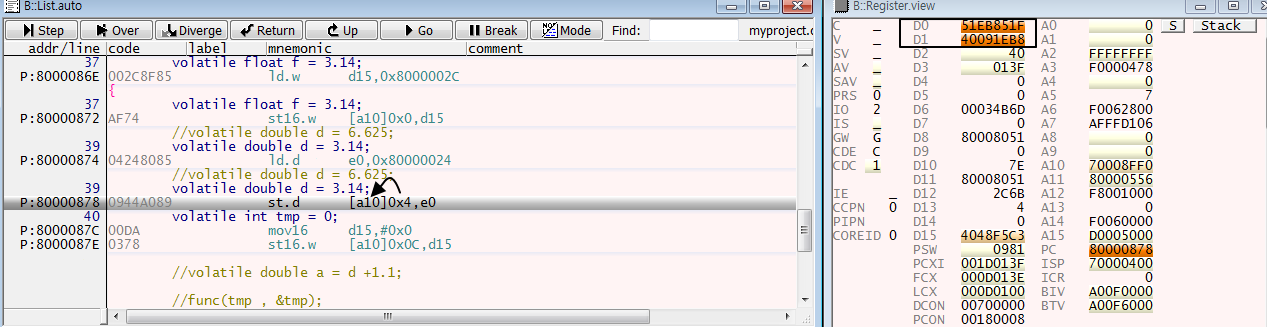




IDE에서 file🡪properties

T32에서

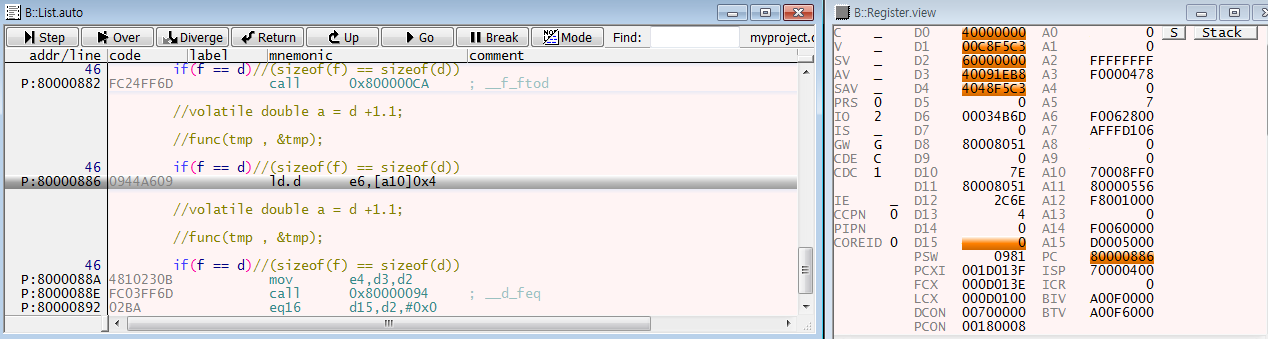


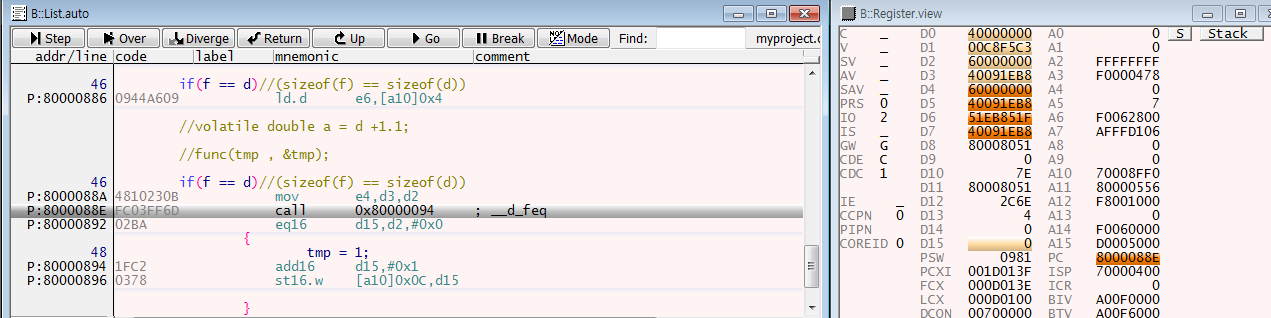


St16.w (d15-float:4byte)와 st.d(d0,1-double:8byte)의 data type이 맞지 않아서 형변환

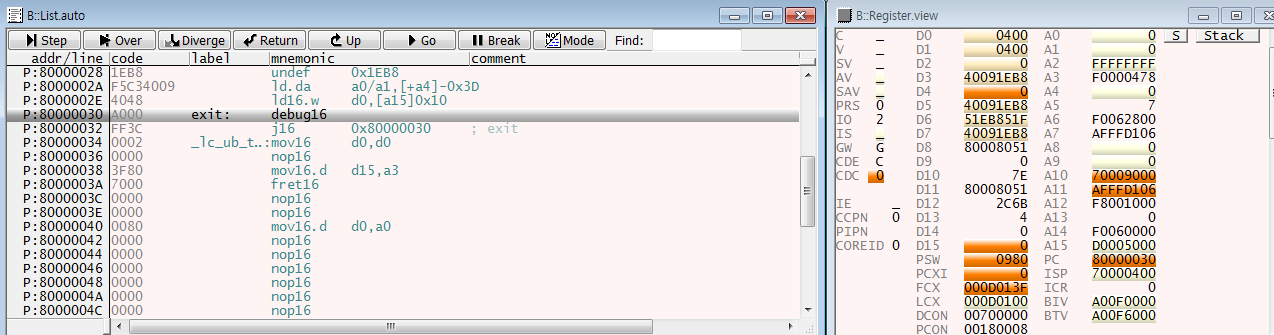
하나는 4byte, 다른 하나는 8byte인 경우 🡪 data 손실을 줄이기 위해 4byte를 8byte로 늘림

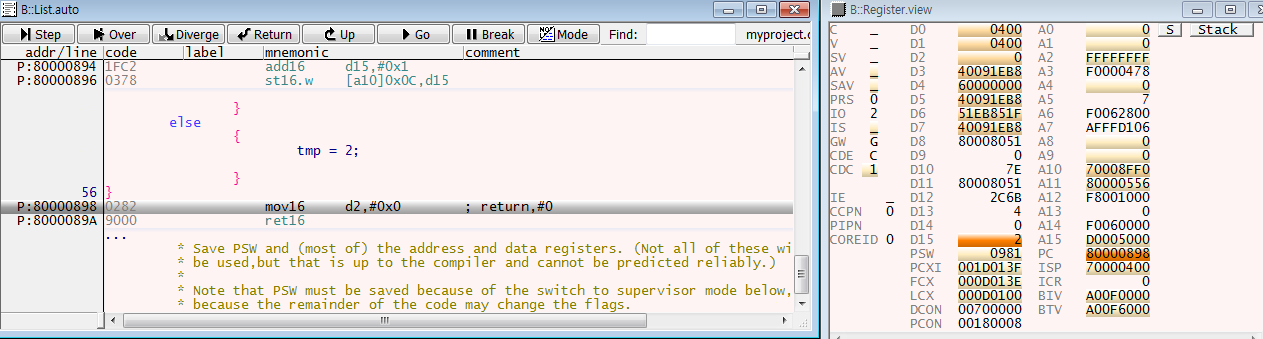
이때 4🡪8byte로 변경됫을 때 4byte의 빈공간은 어떤 값으로 채워질지 알수없어 위험이 있음



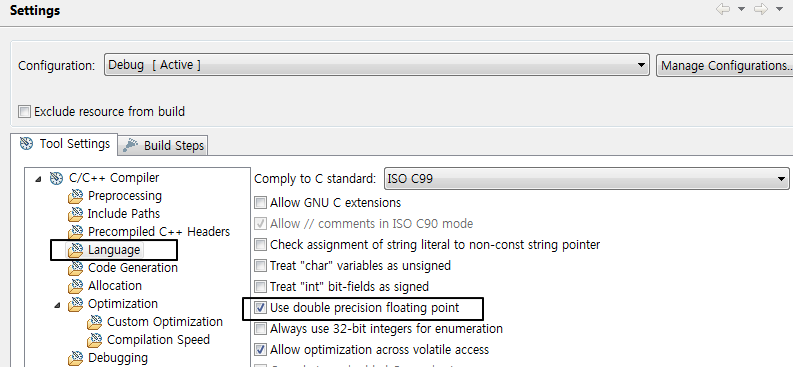


비교





Floating point는 비교할수 없다?



해제하면 모든 double이 상수화

🡪같은 공간에 동일 저장?