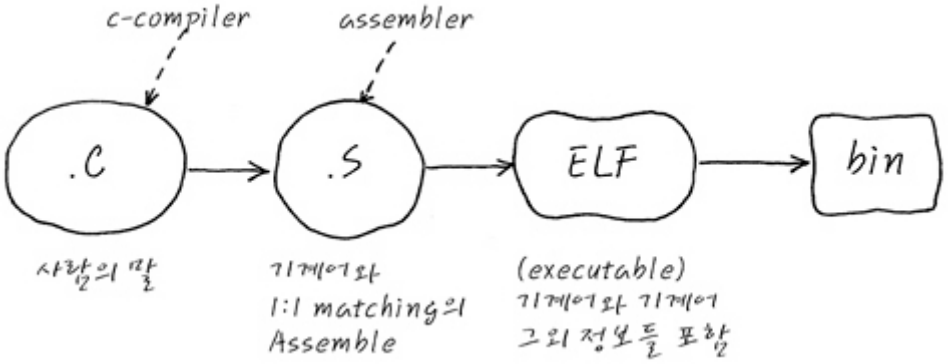
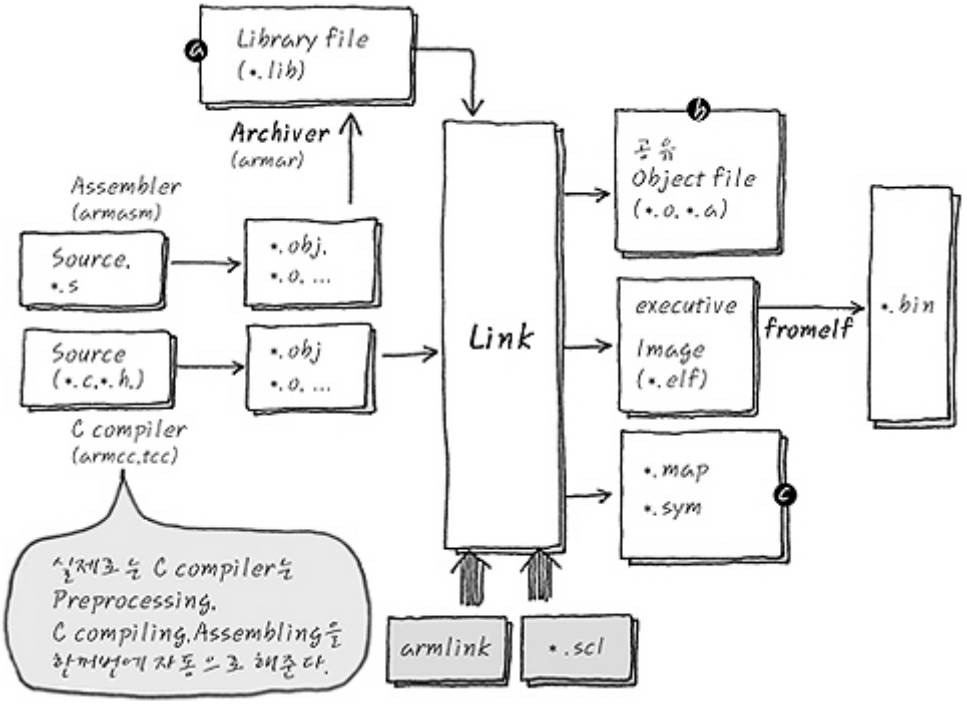
친절한 임베디드 시스템 개발자 되기 강좌 : 컴파일 공장 이야기.

컴파일 공장 이야기.컴파일 공장 이야기는 컴파일이 어떻게 이루어져서 결국에는 우리가 흔히 말하는 Binary Image라는 것을 어떻게 만들어 낼까 하는 뭐 그런 얘기를 말합니다. 큰걸 기대 했다면 약간 미안하지만 하하. 컴파일 공장을 설명하자면 상당히 공장에 온 것 같은 느낌이 듭니다만, 공장이라는 건 뭔가 시스템화 되어 있으니까 조금은 정리되어 있는 느낌도 있습니다. 컴파일공장의 목적은 바로 우리가 흔히 말하는 Native code의 집합인 Binary Image를 만들어 내는 것이 목적입니다. Binary의 정체는 바로 Processor만이 알아들을 수 있는 Native Code(기계어)의 집합입니다. 인간의 눈으로는 도저히 무슨 말인지 알아낼 수가 없는 뭐 그런 세상의 것입니다. 컴파일의 단상에서 불쑥 튀어나온 그 녀석입니다. Binary를 만들어 내기 위해서 원료가 필요합니다. 그 원료는 하하 바로 .c나 .h, 그리고 .s 등의 파일들입니다. (.c, .h는 C code와 header 그리고 .s는 ARM assembly file입니다.) .s의 경우에는 .c를 컴파일하고 나면 Assembly로 만들어 줄 수 있지만, 여러 가지 성능 측면에서 사람이 직접 Assembly로 짠 파일입니다. 뭐 별거 아니지요. 컴파일은 Source file들을 C complier (armcc, tcc)를 이용하여, Assembly로 만들고, 또 이를 Assembler (armasm)을 이용하여 실행 가능하고, Symbol정보를 가진 특정한 type (여기서는 Elf - Excutable and linkable format)으로 만든 후, 그 안에 있는 정말 Native Code(기계어)인 1010으로 이루어진 binary를 뽑아냅니다.



그런데, 아, 여기서 문제가 하나 발생했습니다. 간단한 Software야 위의 그림처럼 .c file하나 만들어서 compile하면 있고없고없고있고의 binary를 만들어 낼 수 있겠지만, 요즘 같은 세상에 그런 간단한 프로그램이 있을 리가 만무합니다. 음.. 그럼 요즘 같은 세상에는 어떻게 큰 프로그램이 만들어 질까요~? 대단히 긴~ .c file을 컴파일 해서 Assembly로 만든 후 Assembler로 기계어를 만들어 낼까요? 음, 인간은 의외로 똑똑 한 개념들을 만들어 내는 재주가 있지요. 인간은 여기에 link라는 개념을 만들었습니다. link라는 것은 여러 개의 .c file을 Assembly로 만든 후 Assembler를 이용하여 Object라는 새로운 기계어 형태를 만들어 내어 이런 object들을 link (연결) 하는 것을 말합니다. (오, 이렇게 하면 여러 개의 .c file을 이용하여 프로그램을 만들 수 있겠네요.) object는 각각의 기계어이긴 하지만 혼자서 완성된 형태는 아니며, 각기 다른 .c를 컴파일 한 object와 연결할 수도 있는 정보와, 디버깅이 가능한 symbol정보를 담고 있습니다. 이 형태는 요즘 흔히들 말하는 elf 형태를 말하는 것이며, elf 형태를 따른 object들은 linker를 이용하여 서로 link할 수 있게 되는 것이죠. 헉헉. 자 어쨌거나, 다시 말하면 컴파일은 Source file들을 C complier (armcc, tcc)를 이용하여, Assembly로 만들고, 또 이를 Assembler (armasm)을 이용하여 object file로 만듭니다. 이때 object file은 나중에 linker (armlink)를 이용하여 모두 엮이게 되며, 이때 output으로 나오는 것 또한 elf 라는 확장자를 가진 큰 elf가 나오게 됩니다. 음.. 상당히 쉬운 공장이죠?한가지 비밀을 더 폭로하자면, armcc, tcc는 내부적으로 *.c file을 *.s file로 만들어 주고 , 만들어진 *.s file을 다시 armasm을 이용하여 *.o (object file)로 만들어줍니다. 결국 알아서 두단계를 자동으로 해주니까 편리하네요. 그마저 자동으로 하는 거 보면 사람은 정말 귀찮은 게 싫은가 봅니다. 그리고, armcc, tcc가 *.c file을 *.s로 만들기 전에 Preprocessor(전처리기)라고 불리 우는 일을 합니다. 자꾸 복잡한 얘기가 끼어 들기 시작합니다만, 일단 C processor와 lint Processor 라는 걸 이용해서 웬만한 syntax적인 것들을 정리해

둡니다. lint processor라는 건 UNIX계열에서 사용하는 Syntax 정리기 비스므리한 것이에요. 선언된 macro나 define을 compile하기 전에 모두 바꿔 치기 해 놓는다거나, syntax error가 있는지 등을 점검한 후, armcc나 tcc가 제대로 컴파일 할 수 있는 단계까지 만들어 놓으면, armcc나 tcc는 Assembly로 만들고, 그를 최적화 하는 것에만 신경 쓸 수 있게 해주는 거지요. 아.. 사족이 길었네요. 결국 c-compiler로 .c file 하나를 컴파일 하면, ⓐ 전처리를 하고 (#define이나, #include등의 것을 잘 처리해서 끼워 넣어서 c 형식의 .i file을 만듦)ⓑ 전처리된 녀석을 mnemonic의 Assembly로 만들고 (기계어와 1:1 대응의 .s Assembly로 만들고)ⓒ Assembly를 실제 기계어로 만들어준다. (elf 형식의 .o file로 만든다)계속 허접하게 늘어놓은 말을 그림으로 그리면 다음과 같습니다.



ⓐ 뒤에서 다루겠지만, 몇 가지 더 덧붙이자면 lib file은 source code를 제공하고 싶지 않은 개발자가 object 형식으로 미리 컴파일 하여 제공하고, lib은 다른 컴파일 된 object들과 link되어 같이 물려 들어가는 형식을 취합니다. ⓑ 또한 link시에 scl이라는 것이 새로 들어가 있는데요, 이것은 Scatter Loading이라고 부르며, binary를 만들어 낼 때, 메모리의 주소 구성을 원하는 대로 주물럭 주물럭 댈 수 있게 해주는 script file입니다. ⓒ Map file이나 Sym file은 compile된 binary의 메모리 구성을 나타내 주는 text file인데요, 보통 compiler option을 줘서 만들어 낼 수 있는 option file이에요. 뭐 다 그렇고 그런 얘기 입니다만, 마지막에 큰 elf는 fromelf라는 Utility를 이용하여 symbol정보 같은 군더더기를 빼내고 나서 순수 기계어 덩어리로 만들어 낸 것입니다. - 자 결국에는 또 인간으로는 도저히 알아낼 수 없는 머 그런 세상입니다. 하하.

Cross Compile 이란.이제 와서 Cross Compile이란 걸 얘기하는 게 쑥스럽지만, 개발 된 Source Code를 다른 CPU에서 동작할 수 있도록 Target CPU에 맞도록 Compile해 주는 것을 의미하는데요, 예를 들어 X98 계열의 PC Host에서 개발한 Source Code를 ARM에서 동작할 수 있도록 PC Host에서 Compile을 해주는 것을 말하는 거지요. 말은 멋지지만 별건 아니죠

이제부터 Cross Compiler로는 ADS라는 걸 기준으로 가겠습니다. 그러니까 간단하게 말해서 ADS라고 말이 나오면, Compiler로구나! 라고 생각하심 되어요. ARM에서 동작하는 Software를 만들어 주는 Compiler라는 사실이라는 것만 알아두면 되어요. ADS는 ARM社에서 파는 ARM Developer's Suit이라고 하는 건데요, 이게 제일~ 상용으로 많이 쓰이니까 이걸 기준으로 하는 거에요. 실은 GNU 진영의 ARM GCC도 있고요, 머 여러가지 compiler가 있답니다.