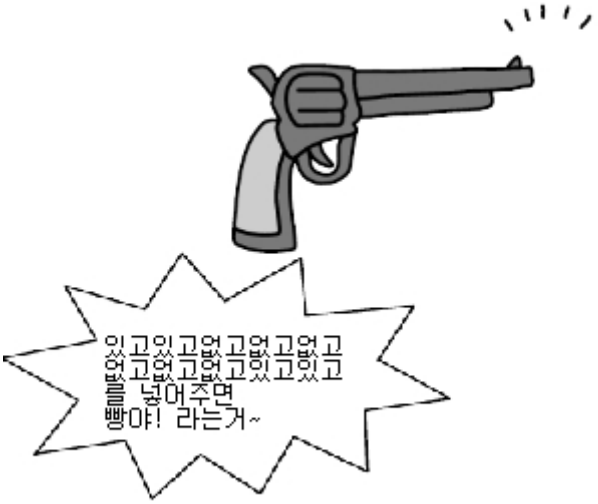
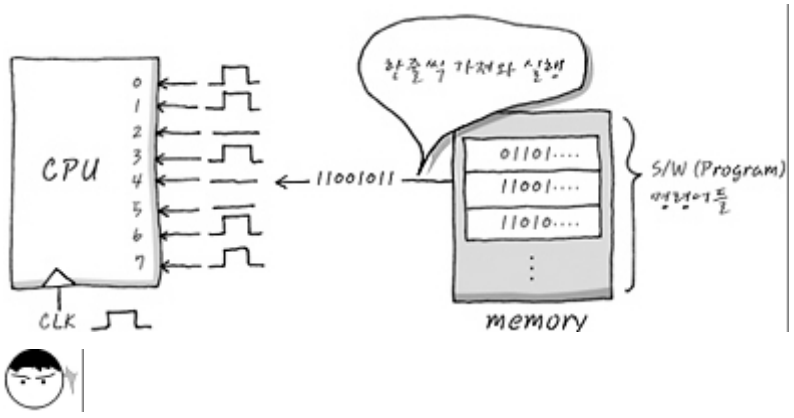
친절한 임베디드 시스템 개발자 되기 강좌 : 컴파일에 대한 단상

컴파일에 대한 단상프로세서는 많은 전기적 스위치로 이루어져 있으며, 어떤 특정한 전기 스위치를 작동 시키기 위해서는 데이터 버스 선을 따라 전압이 "있고 있고 없고 없고 없고 없고 없고 있고 있고"의 상태로 만들어 주면 된다라는 의미 자, 처음으로 Software에 입문하시는 분들은 Compile이라는 단어를 아주 자주 듣게 되는데요, Compile이라는 단어를 영어 사전에서 찾아보면 책을 편집하다, 안내서를 만들다 라는 뜻이에요. 그럼, Computer에게 안내서를 만든다는 것은 무엇을 의미하는 것이죠?프로세서가 해석할 수 있는 것은 단지 명령어로서, 이 명령어는 "기계어"라고 불리 우는 특정 bit pattern을 말합니다. "Processor는 약속되어진 특정한 Bit Pattern에 반응한다."는 의미에서 특정 Bit Pattern을 명령어라고 부를 수 있습니다. 쉽게 말해 프로세서는 많은 전기적 스위치로 이루어져 있으며, 어떤 특정한 전기 스위치를 작동 시키기 위해서는 데이터 버스 선을 따라 전압이 "있고 있고 없고 없고 없고 없고 없고 있고 있고"의 상태로 만들어 주면 된다라는 의미와 일맥상통할 수 있을 것입니다. 예를 들어, Processor가 "11000001 10100000 101000 111011"라는 bit pattern (즉 16진수로는 C1 A0 28 3B 라는 32bit pattern)을 Register 1과 Register 2를 더해서 Register 1에 결과를 저장하라는 내용으로 약속을 한다면, Processor에게 Register 1과 Register 2를 더해서 Register 1에 결과를 저장하라고 일을 시키고 싶을 때는 11000001 10100000 101000 111011 의 32bit bit열을 Data Bus를 통하여 Processor에게 던져주면 Processor는 그 일을 실행하게 됩니다. - 막상 이런 pattern을 해석하는 디지털 회로는 Decoder에요 우훗 -



프로그램이란 이런 일련의 기계어 명령의 순차적인 집합이며, 예전에는 이런 기계어 명령을 사람이 직접 Processor에게 입력시켜 주었었드랬습니다. 그러다 보니, 사람이 이런 이진수의 나열을 해석하기 쉽지 않으니까 - 이런 기계어 명령을 Native Code라고 부른다고 했었죠. -Assembly라는 개념을 도입하게 되었는데, Native Code(기계어)와 1:1 matching이 되는 그나마 사람이 보기 쉬운 표기 체제를 만들어 냈습니다. 예를 들어 Memory에 Data를 읽어 오는 "LDR R0,= 0xFF"를 bit pattern으로 분석했더니, 0x3CD1FF과 완전히 같은 말이며, 이것은 bit pattern으로는 111100110100011111111111이 되며, 이것들은 서로 완전히 1:1 대응이 되어 Assembly로 표현 되었을 때 사람이 보기에 훨씬 편하게 됩니다. 이런 이유로 인간은 bit pattern을 직접 입력하지 않고, Assembly로 coding을 하게 되었으며, 이를 Native Code(기계어)로 바꾸어 주는 것이 compile의 목적이었습니다. 그래서, 예전에는 인간이 직접 code표를 찾아서 기계어로 바꾸어주는 일을 했죠. 예전에 태어나지 않은 것이 천만 다행입니다. Compile을 손으로 직접 하다 보니, 성격을 많이 버리게 된 거죠. 인간은 귀찮은 일은 딱 질색으로 하는 인종인지라, 코드 표를 찾는 일을 대신 해주는 Assembler라는 것을 만들게 되었습니다. 그것이 compiler의 최초 태동입니다. 원래는 LDR R0,=0xFF 이런 명령어는 기계어에 존재치 않고,

111100110100011111111111 만 있었다는 얘기죠. 자, 그런데 이 Native Code(기계어), Assembler라는 것이 특정 Processor에만 통한다는 사실!. Processor는 약속 되어 진 특정한 Bit Pattern에 반응한다는 사실 때문에, Processor마다 다른 Assembler를 지원하므로 특정 Processor에 맞추어서 만들어진 프로그램은 다른 종류의 Processor에서는 동작하지 않는다는 문제점이 있었습니다. 흔히들 말하는 Compatibility 호환성에 한계를 드러냈는데, 서로 다른 Processor에 맞는 Assembly를 만들어내는 Compiler가 있었으면 좋겠다는 인간의 귀차니즘이 또 하나의 새로운 entity를 만들어 냅니다. 그것이 무엇이냐 하면, C나 C++같은 High Level Language compiler입니다. C같은 High Level Language로 Coding을 하고 나면, 내가 동작시키고자 하는 Processor에 맞는 C compiler를 이용해서 그 Processor에서 동작 가능한 Assembly를 generation하는 Compiler를 만들어 낸 것이지요. 여기에서 또 하나! 보통 사람들이 C가 이식 성이 좋다고 하는데, 이식 성이 좋다고 함은 많은 Processor들에 C Compiler가 제공된다는 말입니다. C 자체가 이식 성이 좋다고 하기는 쫌 그르치요~ 결국 ARM core를 예를 든다면, C로 coding을 한 후, compile한다는 의미는 C compiler를 이용하여, ARM이 해석할 수 있는 Assembly를 만들어 낸 후, ARM Assembler를 이용하여, ARM core가 해석할 수 있는 일련의 Bit pattern의 만들어 낸다고 할 수 있겠습니다. 이런 bit pattern을 한 덩어리 뭉쳐놓은 것을 흔히들 말하는 Executable binary image라고 합니다.



mnemonic : mnemonic은 사람의 기억을 돕기 위하여 사용되는 Symbol을 의미하는데, 결국 assembly는 기계어와 1:1로 matching되므로 Assembly 코드를 mnemonic이라고 불러도 무방합니다. 실은 중요한 것은 이렇게 메모리 안에 들어 있는 숫자들이 CPU가 가져다가 실행하면 명령어고, 수정하거나 고치면 데이터라는 사실이죠.흔히들 cross compile 환경이라는 말을 많이 쓰는데, Cross Compile이란 실제 Target에서 돌아갈 binary image를 PC 상에서 compile 할 수 있게 해주는 환경을 말합니다. 흔해 PC에서 컴파일 한 것은 PC에서 실행하는 게 정상인데, embedded system에서는 target자체에서 컴파일을 수행하기에 너무 작은 시스템이며, 불편하기에 이런 환경을 만들어 binary image를 PC상에서 만들어 내는 거죠.