친절한 임베디드 시스템 개발자 되기 강좌 : Little Endian과 Big Endian

사람은 processor보다 위대한 존재이죠. 그러니까 형 (big)이고요, processor는 동생 (little)입니다. - 걸리버 여행기 - 숫자로 가득 메워진 - 영화 메트릭스의 오프닝에 다다다다 올라가는 듯한 - 기계의 언어 세상에서 그 숫자들을 제대로 해석하려면 한 가지 익혀두어야 하는 법칙이 하나 있습니다. 메모리 영역이나, 기계어 영역의 숫자들을 제대로 이해하려면 꼭 알아둬야 하는 법칙이기도 합니다. 아래의 내용은 어디선가 주워 들은 얘기를 다시 한번 재미 삼아 들려 드릴 테니, 잘 들어보세요. 흥미진진합니다.

" 같은 일을 하는데 방법이 두 가지가 있다면, 아마도 두 개의 서로 다른 회사는 서로 다른 방법을 채용할 가능성이 다분할 것입니다. 머피의 법칙이기도 한 이런 가능성은 서로 다른 칩디자이너가 메모리에서 데이터를 서열화하는 방법에도 또한 마찬가지로 통용됩니다. 걸리버 여행기에 등장하는 소인국은 매우 작은 나라인데, 그에 걸맞게 사소한 정치적인 문제를 가지고 있었습니다. Big-Endian당과 Little-Endian당이 격론을 벌이는데, 그 격론의 내용이라는 것이 반숙된 달걀을 깨고자 할 때, 뭉툭한 끝(Big-End)을 먼저 깰 것인가 아니면 뾰족한 끝(Litttle-End)부터 먼저 깰 것인가라는 것 이었습니다. 1980년 4월 1일, 대니 코헨(Danny Cohen)이 지금에서는 유명하게 된 "On Holy Wars and a Plea for Peace"라는 책에서 워드에서의 바이트 오더링(Byte Ordering in Words)에 대해 논하면서 '엔디안'이라는 용어를 이 문제를 지칭하는데 처음 사용했습니다. 그 후 곧바로 이 용어는 고착되었고, 엔디안 이라는 용어는 Byte Ordering을 의미하게 되었습니다. 유닉스 프로그래머는 "NUXI 문제"라고 부르기도 하는데, 바이트 오더링에 착오가 생기면 'UNIX"라는 단어의 입력에 대해 앞뒤가 뒤바뀌어 'NUXI'라는 출력이 나오니까 말입니다. " 재미있죠?이 이야기에서부터 시작하면 모든 processor는 Little Endian 또는 Big Endian중 하나를 사용하게 되는데, 이는 무엇을 의미하는 걸까요? 이것은 processor가 memory에 저장하는 방식을 의미하는데, 저장 방식이 다를 뿐이죠. 이 방식을 이해하지 못하면, 디버깅 시에 오류에 빠질 수가 있으니 꼭 이해해야 하는 부분 임을 이해해 주세요. 당부. 아래 그림을 보면 이해가 확실히 될 것입니다.. 자, 그럼 예를 들어 dword 0x12345678, word 0x1234, 또는 byte 0x12를 0x1000번

지부터 저장하는 방식을 볼까요? 번지 하나에는 1byte가 들어갑니다요. (dword는 4byte, word는 2byte, byte는 8bit 크기로 다룰께요.) 0x1000 0x1001 0x1002 0x1003 Big Endian dword 0x12 0x34 0x56 0x78 word 0x12 0x34 byte 0x12 Little Endian dword 0x78 0x56 0x34 0x12 word 0x34 0x12 byte 0x12 자, 그림을 보니 확 느낌이 오죠? - 라고 말했지만, 중요한 것은 Little Endian은 그 크기만큼 무조건 거꾸로 읽는다 가 힌트입니다. - 자, 그럼 Big Endian과 Littel Endian은 어떤 차이가 있을까요? 쉽게 말해서 "Little Endian은 상위bit (MSB)를 상위 주소에 저장을 하고 있습니다요"가 힌트입니다. 네 그렇습니다. Little Endian으로 처리를 하게 되면, Processor는 Software에서 정해준 type 크기 만큼 그대로 읽어와서 처리를 하게 되고요, Big Endian의 경우에는 대신 낮은 주소부터 읽어와서 MSB로 넣어주면 됩니다. (MSB가 LSB쪽으로 정렬되어 있으니까요, MSB와 LSB는 사족에!) dword type의 0x12345678을 저장하는 것을 다시 그림을 그려서 본다면 아래와 같습니다.

0x12345678 MSB LSB Big Endian Little Endian

0x1003 0x78 0x12 0x1002 0x56 0x34

0x1001 0x34 0x56 0x1000 0x12 0x78 사실 어느 게 더 좋다고 말하긴 어려우나, ARM의 경우에는 Little Endian을 지원하니, Little Endian의 경우를 잘 알아 두는 것이 좋겠지요? 근데, 또 재미난 사실이 하나 있습니다. 우리가 소프트웨어를 구성할 때 Little Endian이냐 Big Endian이냐를 compile환경에서 설정 해 줄 수가 있습니다. 물론 ARM 환경의 Embedded system이라면, Little Endian으로 Memory의 내용을 인식하는 것이 Default이니까, Compile할 때는 Little Endian으로 option을 주고 compile해야겠지요? 컴파일 버전 중 사용하는 ads 의 Compile flag들을 에서 확인해 보면, 다음과 같은 구문을 확인할 수 있을 것입니다. #---------- # Compiler code generation options #----------END = -littleend# # Compile for little endian memory architecture ..... (생략) ..... CODE = $(END) ..... (생략) ..... 자, 어떤가요? 간단하죠? 교훈이 하나 있어요. 뭐냐하면, 다음과 같은 논리입니다. 사람은 processor보다 위대한 존재이죠. 그러니까 형 (big)이고요, processor는 동생 (little)입니다. 그런 의미에서면, 사람이 읽기 편하게 메모리에 저장되면 big endian, 사람이 읽기 불편하면 little endian이라고 기억해 두면 잘 기억되겠죠? * 사족 : NUXI 문제를 word씩 읽는다고 가정했을 때, Little Endian으로 NUXI으로 저장되어 있다면, Little Endian으로 읽으면 (1 word = 2 bytes) UNIX가 되지만, 그대로 Big Endian으로 읽으면 NUXI가 됩니다. 마치 띄어쓰기를 잘 못해 아버지가 방에 들어가신다와 비슷한 꼴인 듯한 기분인데요. ■

ARM에서는 어떻게 Big Endian/ Little Endian 지원하는가요? 기왕 Little Endian과 Big Endian 얘기가 또 나왔으니까 하는 말입니다. ARM의 경우, Little, Big Endian을 모두 지원하는데, 이건 어떻게 결정하느냐, ARM core를 채용해서 구현한 chip이 어떻게 생겼느냐에 따라 다릅니다. 보통은 외부에서 pin을 하나 설정해서 Low/ High로 Endain을 다르게 동작하게 만드는 것이 Alternative입니다만, chip에 따라서는 한가지만 지원하도록 아예 SoC를 하는 경우가 많습니다. 이런 설정에 따라서 compile을 할 때, Little, Big Endian을 제대로 setting해서 compile해줘야 합니다. 또는 이렇게 따로 설정하지 않은 경우에는 ARM은 default Little Endian이며, Co processor 레지스터 CP 15를 통해서 Big Endian으로 만들 수 있습니다.

LSB와 MSB는 bit를 따질 때 높은 쪽 자리의 숫자이냐, 낮은 쪽 자리의 숫자이냐를 따질 때 쓰는 용어인데, 예를 들어, 8bit의 이진수가 10001000있다고 치면, 이때 맨 왼쪽 자리를 MSB (Most Significant Bit)라고 부르고요, 이 예에서는 1이 되겠죠. 그리고 맨 오른쪽 자리를 LSB (Least Significant Bit)라고 부르고요, 이 예에서는 0이 되겠네요. 그래서 상위 2bit를 지칭 할 때는 MSB 2bit이라고 하고 맨 왼쪽 두 자리 10이 되고요, 하위 3bit를 지칭 할 때는 LSB 3bit라고 부르고, 맨 오른쪽 세자리 000을 의미해요.