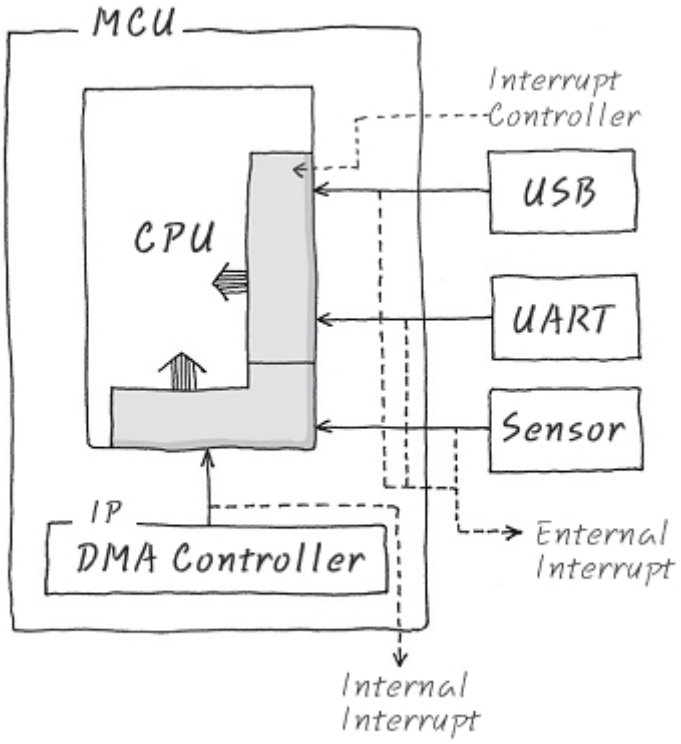
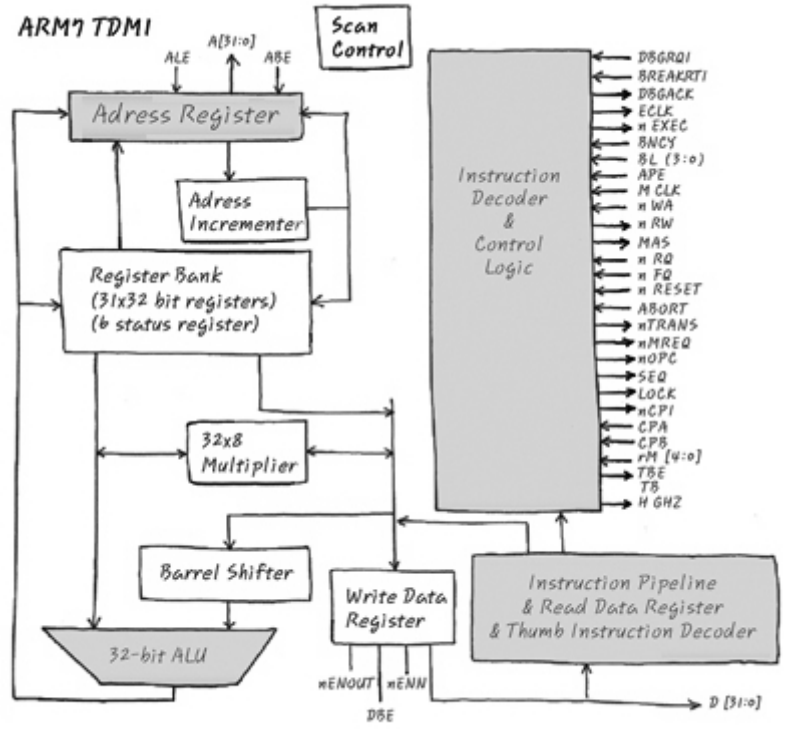
친절한 임베디드 시스템 개발자 되기 강좌 : ARM은 Interrupt 냄새를 어떻게 맡는가

이제까지 열심히 Mode도 알아 보았고, Exception과의 관계도 알아보았습니다만, 실은 Exception중에서도 가장 쓸모 있는 건 Interrupt이겠죠. ARM은 Interrupt가 걸리면 어떻게 행동하는가가 바로 Exception 처리 중 한가지이며, IRQ나 FIQ Handler 에 가기 까지가, MCU가 어떻게 행동하는가에 대한 대답인 겁니다. 그러니까 Interrupt가 걸리면 실행 주소가 Exception Vector중에 IRQ나 FIQ Vector로 강제로 바뀌어 지며, 그 Vector에는 실제 Handler로 branch하기 위한 code가 들어 있죠. IRQ, FIQ Handler는 MCU가 Interrupt에 대한 응답을 말하는 거겠죠. Handler 내부에는 Interrupt가 여러 가지 종류가 있을 테니, 그 각각의 Interrupt에 따른 응답이 다릅니다요. 그것을 Interrupt Service Routine이라고 부르고요, 약자로는 ISR이라고 불러요. 좀 유식해 보이죠. 이런 구체적인 얘기를 하고 있으면서도, Interrupt에 대한 정의가 제대로 되지 않았군요. Interrupt는 도대체 뭡니까?1) 짱나게 방해2) 짱나게 끼어들기3) 짱나게 중단하기4) 공몰고 가는데 짱나게 공 가로채기. 뭐, 다 맞는 얘기입니다만, 제 정의로서는 뭐 이렇습니다. 내가 뭔가 하고 있는데 Interrupt가 들어왔다는 건 그 Interrupt는 그렇게 큰 일이 아니면서도 내가 그 일을 먼저 하게 만들어서 그 일 먼저 처리하고 원래 하던 일을 계속 하게 하지요. 대부분 이런 Interrupt는 휴대전화가 그 예이죠. 친구랑 얘기하던 중에도 전화가 오면 그거 부터 받고 짧은 얘기면 곧바로 얘기를 모두 끝내고, 긴 얘기면 "이따가 다시 전화 할 께" 등으로 일을 처리하죠. 이런 게 Interrupt인 거에요.그럼 전자 System에서의 Interrupt의 정체는 뭘까요.뭐 당연히 전기 신호겠죠.

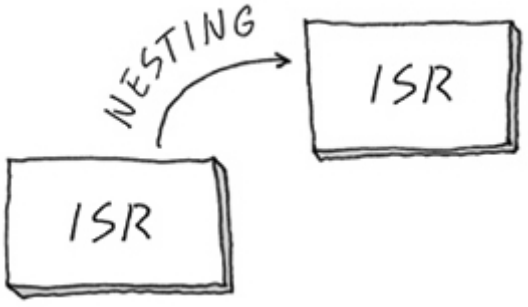


MCU안에는 Interrupt Controller라는 IP가 하나 달려 있고요, 외부에 나와 있는 pin이나, 내부에 있는 IP와 Interrupt Controller사이에 전깃줄 (크 control bus)이 하나 달려 있어서, 그 선을 통해서 신호를 주면 Interrupt Controller가 으앗! Interrupt가 들어 왔구나 하면서, CPU의 mode를 IRQ mode로 바꿔 준답니다. (IRQ와 FIQ는 어떻게 어떻게 다르냐 하면 Interrupt Controller의 어느 구멍에 연결해 주느냐에 따라 틀리겠죠.)



〈 ARM7-TDMI의 아키텍처 〉

CPU로의 Interface는 nIRQ나 nFIQ로 연결되고요, n은 여러 개 있을 수 있다는 의미이지요. 결국 Control Logic이라고 표시되어 있는 부분이 흔히 말하는 CU 부분이고요, 여기에 몇 번 Interrupt가 들어 왔는지를 알려줘요. 물론 전용 Readable Register가 있고요. 읽어보면 몇 번 ISR이 들어 왔는지 알 수 있겠죠. 어차피 Readable Register에는 nIRQ 신호를 통해 들어온 값이 들어 있고요, 그 각 bit의 값은 Hardware 구성에 따라 틀리겠죠. 결국에 IRQ mode에 들어간 후, Handler에서는 몇 번 ISR이 들어왔는지를 확인한 후에, 해당 Interrupt에 적당한 ISR을 만들어서 응답하는 게 그 정체라고 할 수 있지요. 여기에서 한가지 용어를 짚고 넘어가자면 Nesting이라는 용어인데요, Interrupt가 걸려서 ISR을 처리하는 동안에 Interrupt가 또 걸리는 게 Nesting이에요. 이건 System에 따라, Designer에 따라 천차만별인데, ISR을 처리 하는 동안에 Interrupt를 Disable해 놓고서, 아예 Nesting을 허용 안 하는 System이 있는가 하면 몇 번에 걸친 Nesting 까지는 처리해 주겠다, 뭐 그런 System도 있는 게죠. 무한히 Nesting을 처리해 주다 보면 원래 맨 처음에 처리하던 Interrupt는 무한히 그 순서가 밀리겠죠.



그리고, 우리 Interrupt 예를 들었을 때 "친구랑 얘기하던 중에도 전화가 오면 그거부터 받고 짧은 얘기면 곧바로 얘기를 모두 끝내고, 긴 얘기면 이따가 다시 전화할께라던가 하죠. " 이런 식으로 예를 들었었는데, ISR을 짤 때 조심해야 할 것이 어차피 원래 System이 하던 일이 있음에도 불구하고 잠시 짬을 내서 조금 급한 것을 먼저 처리해 주는 거니까, 아주 짧고 간결하고 굵게 짜야 합니다. 그런데, 아주 짧고 간결하고 굵게 짜고 싶지만, 의외로 할 것이 많아서, 당장 처리 하기에는 원래 하던 일을 방해할 수도 있겠죠. 이럴 때를 대비해서 일단 중요한 용건은 처리하고, ISR routine을 끝낸 후에, Software (Kernel Level)에게 나중에 다 못 끝낸 일을 처리할 수 있도록 일을 미뤄 놓는데요. 그런걸 DPC (Deferred Procedure Call), APC (Asynchronous Procedure Call), Bottom Half.. 뭐 이런 식의 용어로 부른답니다. DPC, APC나 Bottom Half등은 RTOS쪽에서 더욱 잘 살펴볼게요!