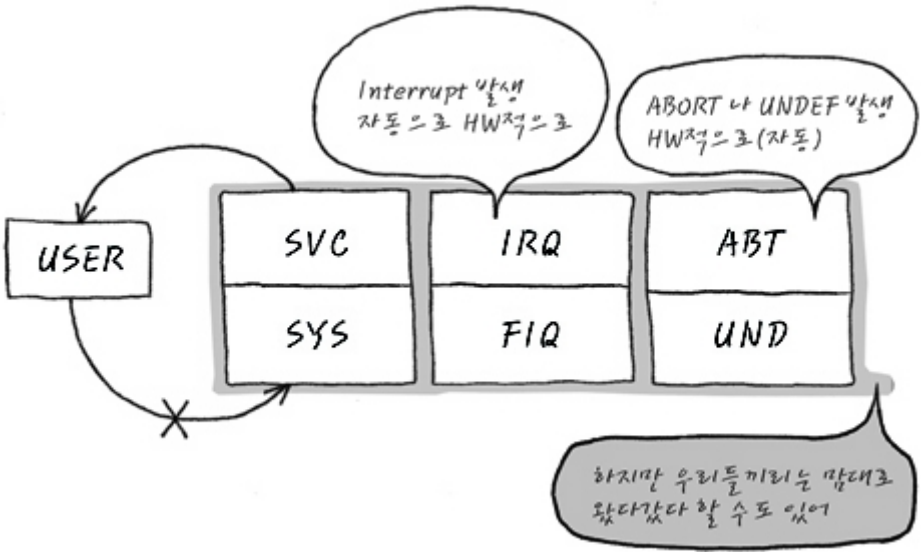
친절한 임베디드 시스템 개발자 되기 강좌 : ARM Exceptions and Modes

ARM Mode과 Register들을 알아보았으니, 이런 Mode들은 어떻게 진입하는지 알아야겠죠. 물론 Privileged Mode는 자기 마음대로 mode를 바꿀 수 있지만, 그거 이외에 Hardware적으로 자동으로 특정 Mode에 진입할 수가 있습니다. 그런 Mode 진입 유발자가 바로 Exception입니다. Exception이라는 건 Interrupt를 포함한 더 큰 사건을 의미합니다. 그러니까 외부 요청 이라던가 오류에 관련된 사건인 것이죠.
Interrupt는 Exception의 한 종류에요. Exception이라는 사건을 통해서 Hardware적으로 정해진 특별한 reaction이 발생합니다. 그 reaction이라는 것은 Exception이 발생하면 진행하던 동작을 멈추고, Exception의 종류에 해당하는 mode에 진입하고 그 Exception에 물려 있는 해당 주소로 pc를 jump시킨 후 Exception에 대한 처리를 하는 것이 그 줄거리인 거에요. 그 내용들은, SVC mode는 ARM에 전원을 인가하거나, reset을 시키면 SVC mode로 진입하면서, PC를 0x0 (Low vector인 경우)로 jump 시킵니다. IRQ와 FIQ mode는 Interrupt가 발생하면, IRQ 또는 FIQ mode로 진입하면서, PC를 0x1C 또는 0x18로 jump 시킵니다. ABORT mode의 경우에도, Data abort가 난 경우에는 0x10으로, Prefetch abort의 경우에는 0x0C로 jump를 하고 ABT mode 진입합니다. UNDEF mode의 경우에도, Undef exception이 났을 경우에 UNDEF mode로 진입하면서 PC를 0x04로 jump 시키죠. - PC를 어딘가로 jump 시킨다는 얘기는 PC값을 그값을 setting한다는 얘기이고, setting을 하면 거기서부터 Software를 실행한다고 봐야겠습니다 - 그걸 표로 엮으면 다음처럼 되겠네요. 아 깔끔해.

Low vector	High vector	Exception	Entry mode
0x00000000	0xFFFF0000	Reset	SVC
0x00000004	0xFFFF0004	Undefined instruction	UNDEF
0x00000008	0xFFFF0008	SWI(소프트웨어인터럽트)	SVC
0x0000000C	0xFFFF000C	Abort (prefetch)	ABT
0x00000010	0xFFFF0010	Abort (Data)	ABT
0x00000014	0xFFFF0014	Reserved	Reserved
0x00000018	0xFFFF0018	IRQ	IRQ
0x0000001C	0xFFFF001C	FIQ	FIQ

이런 Exception이 일어났을 때 jump하는 Address들을 모아 Exception Vector table이라고 부릅니다. 다시 말해 Exception이 발생하면 그 Exception에 해당하는 미리 정해진 어드레스의 프로그램을 하드웨어적으로 수행하는데, 이 어드레스를 Exception Vector라 하고, 각각의 Exception에 대해서 Exception Vector를 정의해 놓은 테이블을 Exception Vector Table이라고 한다는 말이죠. 재미있는 case는 user mode와 system mode로 진입하는 Exception은 없다는 거죠. 그러니까, 이걸 다시 해석하면 Privileged mode에 의해서 user mode로 한번 가면 Exception 없이는 다른 Privileged mode로 돌아 올 수 없다는 얘기고요. 그나마 user mode가 너무 힘이 없으니까, system mode라는 넘을 만들어서, Privileged user mode라는걸 만들었다고 보면 좀 수월할까요?



User mode의 인생은 참으로 힘듭니다.자, 그럼 각각의 Exception들은 어떨 때 발생하는지 생각해 봅슬레이?1. SVC mode SVC mode는 Power on이나 reset이 일어난 경우에 SVC mode에 진입합니다. 2. IRQ mode Hardware적인 Interrupt가 발생하여 ARM Core에 알려주면, IRQ mode로 진입합니다. 2. FIQ mode Interrupt중 Fast Interrupt가 발생하면 진입합니다. 3. ABT mode Abort mode는 Access 하려는 주소가 Access 할 수 없는 주소이거나, Instruction fetch를 해오려는 데 못해 온 경우에 진입합니다. MMU나, MPU를 사용하는 예에서는 Access Protection이 걸려 있는 주소를 함부로 Access하려고 했을 때 ABT exception이 발생합니다. 4. UND mode Undefined mode는 Instruction을 decode했는데 ARM이 모르는 것일 경우에 진입하게 되며, 보통 Memory Corruption이 났을 때, 발생합니다만, 이걸 응용하게 되면, ARM이 사용하지 않는 코드를 일부러 삽입하게 하여, UND vector 주소로 jump하게 만들어 디버깅 Code등의 의도된 일을 행하게 할 수 있습니다. ARM을 다루는데 있어 가장 먼저 나오는 용어 중 하나가, 바로 ARM 동작 Mode일 것입니다. ARM을 접근 하는데 가장 기초적인 내용이니까 열심히 보고 또 보고 해서 익혀 두세용 잘 해두면 뭐든 쓸모 있으니까. Mode Identifier를 잘 matching해서 외워 두세요. 뭐, 밑줄 쫙?

CPSR[4:0]	CPSR(0x)	Mode	Registers	진입 유발자(Exception)
10000	0x10	User	User	
10001	0x11	FIQ	_fiq	Fast interrupts
10010	0x12	IRQ	_irq	Standard Interrupts
10011	0x13	SVC	_svc	Reset. Power on. SWI
10111	0x17	Abort	_abt	Memory faluts
11011	0x18	Undef	_und	Undefined instruction trpas
11111	0x1F	System	User	

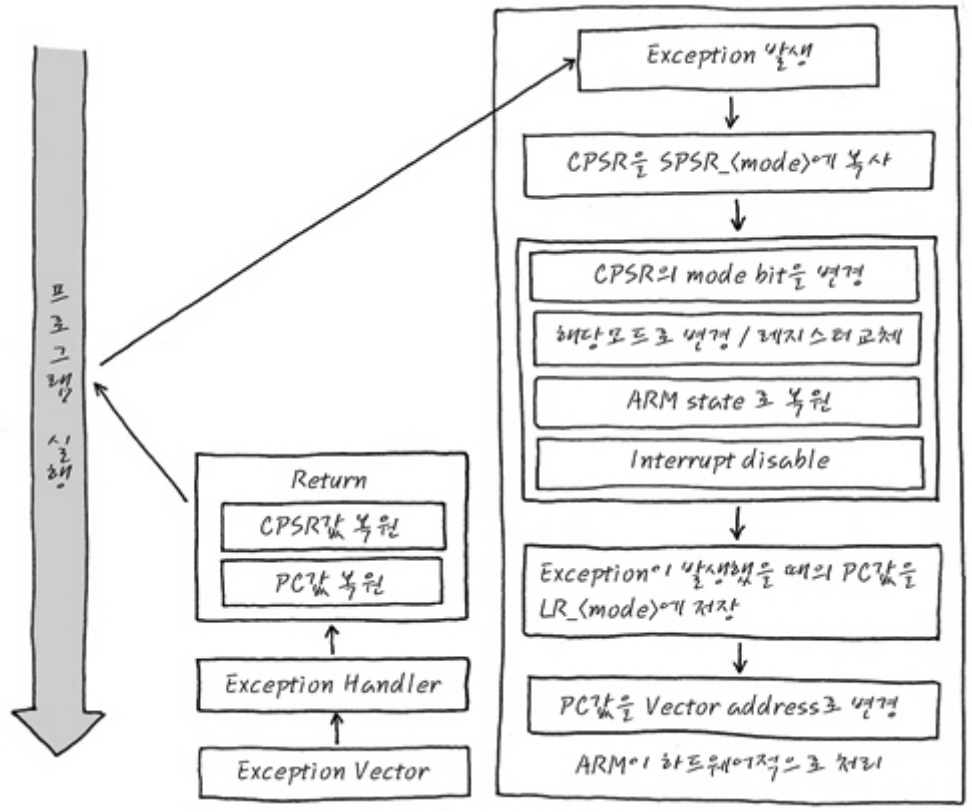
위의 그림에서 볼 수 있듯이, ARM은 User, Fast Interrupt, Interrupt, Supervisor, Abort, System, Undefined 이렇게 모두 7개의 동작 Mode에서 동작을 합니다. 이 동작 mode는 CPSR이라는 Register의 [4:0]까지의 5bit로 구분이 되면 CPSR의 Register [4:0]의 값이 어떻게 setting되느냐에 따라 mode가 구분이 되며 그 mode에서의 특성이 결정되는 이치 이지요. 그 mode의 정체라는 건 바로 이 CPSR [4:0]에 어떤 값이 들어 있냐는 것이지요. CPSR에 들어가 있는 값에 따라 사용되는 Register set과 Stack이 바뀐다는 거지요.뒤로 넘어가기 전에 CPSR의 Hex값과 Mode를 matching해서 외워두면 엄청 편리합니다. 외우는 방법으로는 user mode와 System mode는 너무 쉬우니까 아래처럼 외우는 것을 강추 합니다. ㅋ 뭐 어차피 CPSR 5번째 bit는 모두 1로 같으니까 [3:0]만 가지고도 따질 수 있습니다요. 각 mode의 첫글자와 CPSR의 하위 1byte만을 table로 만들면 아래와 같아요

.

mode	CPSR
F	1
I	2
S	3
A	7
U	B

F가 한 개(1), 나(I)는 2개, S는 삼삼(3)하고, A(에이) 칠칠(7)맞지 못하고 , 유(U)비(B)무한도전 ㅋ. 뭐 이런 식으로 외우면 좋을까요? 별루 라구요? 나는 그냥 외우는 방법을 추천했을 뿐이고!. 외워두면 좋을 뿐이고.알아서 들 잘 외워 두세요. ㅋ그리고 마지막 한가지, Exception에도 우선 순위가 있어요. 한꺼번에 Exception이 발생했을 때 어느 것이 처리 것이냐는 문제이지요. 그 인기 순위는 .... (-,.-)우선순위는 System이 맛탱이 가게 하는 순으로 더 높습니다. System 자체가 맛탱이가 가면 모든 게 소용없으니까 그걸 먼저 처리하는 거죠. 정상적인 넘 일수록 순위가 낮아요~ 으흐흣. 대망의 1등은 Reset .. 당돌 빠따죠. Reset은 영원한 꼴통 입니다. 2위는 Data Abort .. 그렇죠 Data를 읽어 올 수 없으면 다 소용없습니다. 3위와 4위는 interrupt 관련인데 당돌 FIQ가 IRQ보다 쎄겠죠. 5위는 Prefetch Abort 에요. Prefetch Abort는 Pipe line중 가장 처음인 fetch 단계에서 발생하니까, 굳이 순위를 높여 놓지 않아도 다른 넘들 보다 먼저 발생합니다. 6위는 Undefined instruction은 왜 순위가 낮으냐, 하면 Undefined Instruction은 보통 ARM은 해석은 못하지만, 다른 coprocessor라든가, 주변기기를 직접 Assembly로 control할 때 일부러 Undefined instruction을 발생시켜 그 handler에서 coprocessor나 주변기기를 직접 handling합니다. 근데, 보통은 Memory에서 Fetch를 해왔는데 모르는 명령어다.. 라는 뜻이므로, Memory가 Corruption (bit가 바뀌거나 해서 Memory 정상 상태와는 다른 깨져 있는 상태) 이 일어나서 ARM이 모르는 명령어를 만나서 그렇게 되는 경우도 있슴다. 7위는 당연돌이 하나 남은 SWI. SWI는 프로그래머가 일부러 "Intentionally" 발생시키는 것이므로, 굳이 Exception 순위가 높지 않아도 되요. 표의 맨 뒤에 나오는 Vector라는 거 바로 뒷 section인 Exception Vector에서 다루어 놓았으니, 어리둥절 하면 절대 안됩니다. Software 실행 중, Exception이 발생하면 어떤 일이 벌어지느냐! 를 한번 더듬어 볼까요. 꺄 부끄러. 한번 생각해 보세요. 고려해야 하는 것들은.. 1) Exception mode가 발생 한 후, 이전 mode로 돌아갈 수 있어야 합니다. 2) 지금 쓰던 Register 값들을 다시 사용할 수 있어야 합니다. 다시 말해, 이전에 쓰던 Context를 다시 복원할 수 있어야 한단 말입니다. 3) 이전 mode로 돌아갔을 때 원래 수행 하던 곳으로 돌아 갈 수 있어야 합니다. 4) 그리고 자동으로 해당 Exception Vector로 짬푸~ 이 세가지를 만족시키려면 어떤 짓꺼리을 해야 될까요? 1) CPSR을 저장할 수 있어야겠죠. CPSR에는 현재 mode가 어떤 mode 이었는지에 대한 정보가 들어 있으니까 말이죠. 2) Context를 Stack에 저장하면 좋겠지요? Banked Register를 제외한 나머지 register 들 R0~R12까지는 저장해야 합니다. 3) 원래 수행하던 곳으로 돌아가려면, 돌아가려는 주소를 저장해야겠죠. 4) 그리고 자동으로 해당 Exception Vector로 짬푸~ 뭐, 간단하죠? 이런 것을 하기 위해 Exception이 발생하면 하드웨어적으로 다음과 같은 짓거리를 자동으로 합니다. 예를 들어, SVC mode에서 동작하던 중 IRQ가 발생했다고 치자구요. Exception이 발생하면, CPSR의 mode를 IRQ로 변경하면서, IRQ mode의 Banked Register인, R13_irq, R14_irq, SPSR_irq로 현재 context가 Hardware적으로 자동으로 바뀝니다. 그 순서를 자세히 살펴 보면, 1) CPSR을 SPSR_irq에 복사해 넣고, (SVC에서의 값이겠죠?) 2) CPSR의 mode를 IRQ로 변경, 결과적으로 stack pointer도 IRQ mode의 stack pointer로 변경. 3) IRQ disable 함, ARM mode로 변경. "ARM mode로 변경" 이 말은 나중에 ARM/ Thumb mode 얘기할 때 나올 텐데, 어쨌거나 Exception이 발생하면 무조건 32bit ARM mode로 ARM state가 바뀝니다. IRQ는 어떻게 disable하느냐! CPSR의 I bit를 1로 setting하면 되고요, ARM mode도 T bit를 0으로 setting하면 됩니다요. 이 말은 Exception에 관련된 처리를 하는 Exception Vector 부분은 32bit ARM mode로 프로그래밍 해줘야 한다는 말과도 같은 말이니까 주의 깊게 기억해 주세요. 4) R14_irq := 현재 PC 5) 그러고 나서 어떤 일이? IRQ Exception Vector 주소인 0x12로 가야 하니까, PC : = 0x12 이에요. 여기까지는 Hardware적으로 다 알아서 처리가 되고요. Software 적으로는 0x12에 있는 IRQ handler에서 처리해야 할 것들이 남아 있는 게죠. 6) R0~R12를 R13_irq (stack pointer)가 가리키는 stack에 저장하고요, 7) 돌아갈 주소를 보정합니다. R14_irq (LR) = PC를 넣었으니까, interrupt 걸린 순간에는 pipe line에 의해서 2개 opcode가 이미 진행되었으니까, sub lr, lr, #4 처럼 lr을 보정해야 제대로 돌아갑니다. ? Assembly가 나

와 버렸는데, 이건 LR = LR - 4라고 보시면 되어요. 그럼 Exception 처리를 다하고 돌아갈 때는? - backup 했던 넘들을 다시 가져오면 되겠죠. 이건 Software 적으로 알아서 해줘야 해요. 올 때는 자동으로 와도, 갈 때는 편의를 봐주지 않아요. 젠장찌게. 1) CPSR := SPSR_irq를 넣으면 → 이전 mode SVC로 돌아가고요, 2) Stack에 저장했던 Register 값들을 복원해 주고요. 2) PC := R14_irq 를 넣으면 → SVC mode에서 처리 하던 곳으로 돌아가겠사옵니다. 뭐 간단하죠? 이런 걸 그림으로는 아래처럼 표현 하던데, 뭐 말되죠?



mode의 장점　　각 mode마다 register set을 가지고 있어, 각각의 mode들을 적당하게 사용만 한다면 mode 전환 시에 빠르게 전환이 가능하게 됩니다. 예를 들어, User mode에서 interrupt가 걸려 IRQ mode로 전환이 된다면, User mode에서 쓰던 register들을 모두 저장할 필요가 없고 IRQ mode로 전환 되면 된다는 야그 입니다. 또는 User mode에서 application을 동작하게 한다면, Interrupt가 걸려 IRQ 또는 FIQ mode로 전환된다던가 하는 특별한 mode change가 이루어 지지 않는 한, application software는 ARM을 자기 마음대로 control하지 못하니까 security가 좋다고 할 수도 있고요. 또는 User mode에서는 IRQ나 FIQ를 일어나지 못하게 하는 일을 할 수가 없습니다. - 유식한 말로는 CPSR을 control하지 못한다고 해야 하겠습니다. - 이런 Mode에 대한 이해가 잘 되어 있지 않다면, Reset Debugging이라던가, Interrupt service routine들에 대한 Debugging을 잘 하지 못합니다. ARM을 이해하는데 가장 기초적인 문제에 봉착한다는 말씀입니다.

System/ User mode의 SPSR은 왜 없는 거죠?　　가만 들여다 보면 System/ User mode에는 SPSR이 없어요. 왜 없을까? Exception 종류를 보시면 System/ User mode로 들어가는 Exception이 없어요. 그러다 보니, System/ User mode로 들어 갈 때는 SPSR에 CPSR을 저장할 수가 없는 거죠. 게다가 Kernel을 장착한 복잡한 system에서는 user mode가 보통 평소에 동작하는 mode이니까 어느 Exception 때문에 user mode/ system mode에 진입했는지 알 필요가 없다고나 할까요? 하지만 굳이 이전에 어떤 mode에서 왔는지 알아야 할 때는 어떻게 처리 하냐믄~ Stack에 넣어주지요~ 냐하~

System/ User mode의 진입은 어떻게?　System/ User mode의 진입은 CPSR의 mode bit을 setting함으로서

진입할 수 있습니다. 실은 모든 mode는 Exception 말고도 이렇게 CPSR의 mode bit을 setting함으로서 억지로 진입할 수도 있어요. 물론

Exception Vector Table의 정체는? Exception이 발생하면 Hardware적으로 자동으로 정해진 Vector Table의 각 주소로 jump하게 되는데 각 주소에는 Exception에 걸 맞는 처리 루틴, 유식한 말로는 Exception Handler들을 Software Engineer가 알아서 구현해 주어야 합니다. ARM Programming 구현에서 Assembly를 좀 알아보고 나서, 어떻게 구현하는지 살펴 보도록 해야겠어요.

High Vector와 Low Vector는 또 뭐냐? Low Vector는 말 그대로 Low Address에 Vector Table이 존재하는 방식이고, High Vector는 높은 주소에 Vector Table이 존재하는 방식으로서, 보통 MCP를 처음 Design할 때 어떤 걸 쓸 건지를 정해 놓습니다. 원래 idea는 Low Vector를 쓰는 게 맞습니다만, 최근 들어 NAND Flash에 Software를 넣어두고, 막상 실행할 때는 SDRAM에 NAND의 내용을 복사한 후 실행하는 방식이 늘어나다 보니, SDRAM을 init 하기 전에는 SDRAM이 무용지물이니까, 언제든지 사용 가능한 아~주 작은 크기의 SRAM을 bootloader용으로 MCP내부에 embed하고 그 영역의 주소를 High Vector로 사용하는 예가 많이 늘었어요.