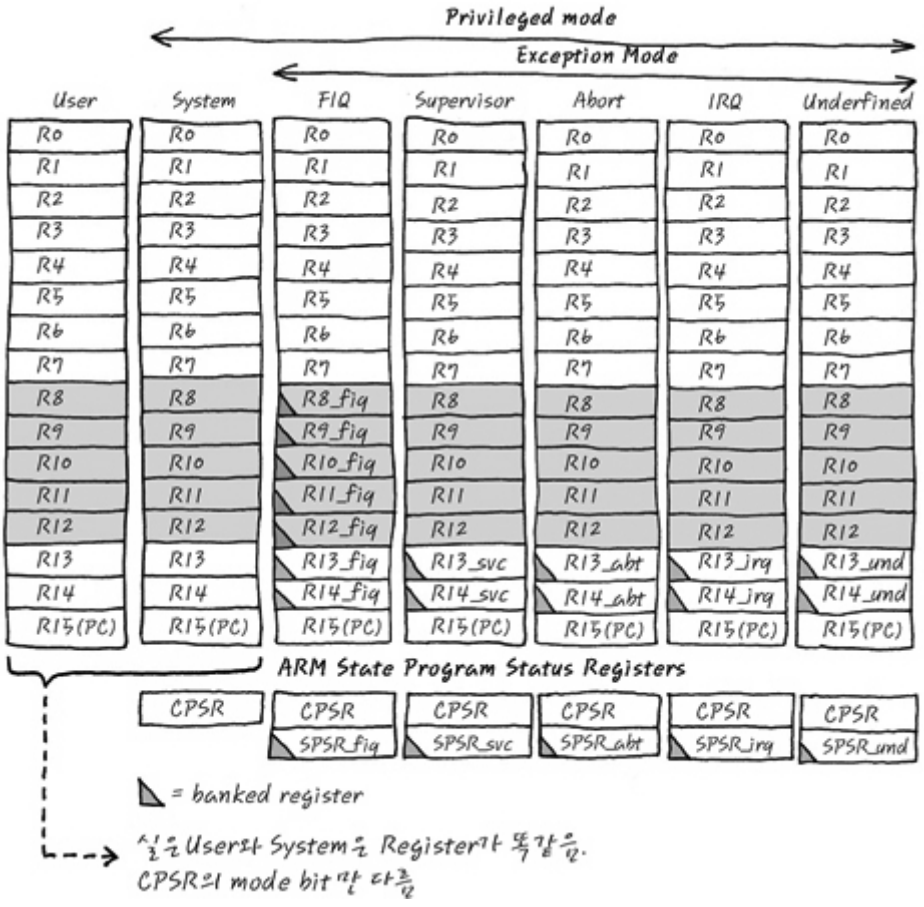
2009년 06월 04일

ARM Register와 Context

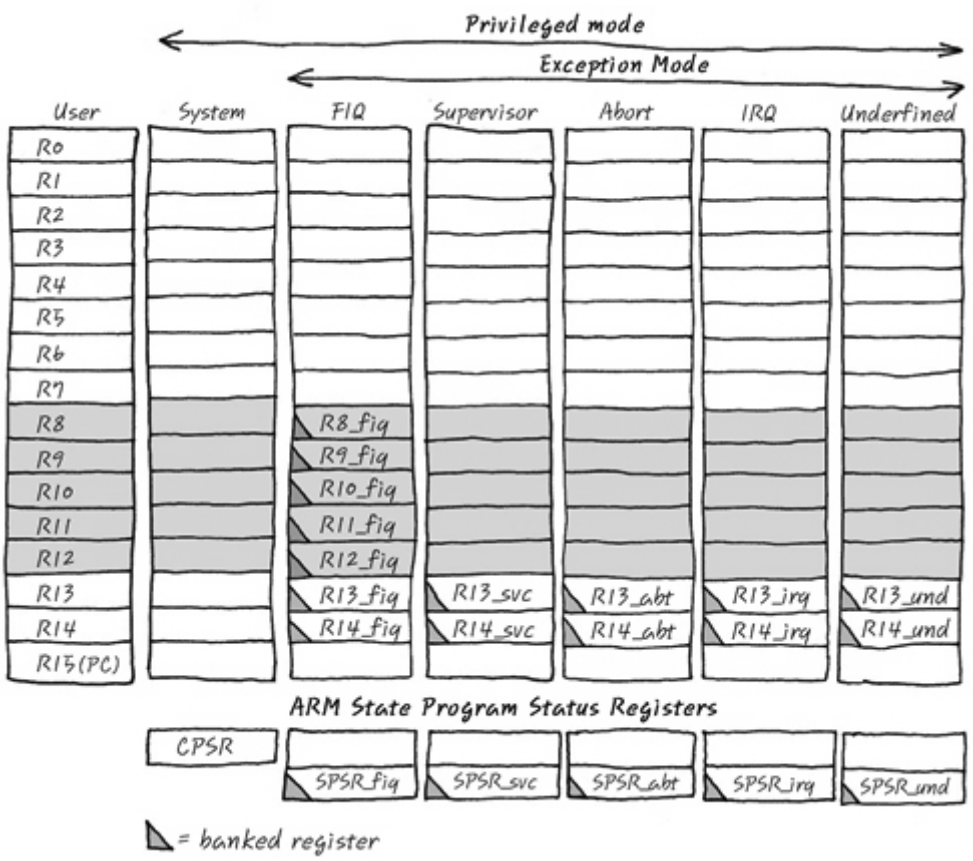
ARM Core를 잘 이해하기 위해서는 ARM Core에 내장되어 있는 기본 Register들이 어떻게 구성되어 있고, 사용되는지를 잘 알면 상당히 편리 하거든요. Register들은 Core가 사용할 수 있는 저장 매체 중에서 가장 빠른 속도를 자랑하며, ARM의 동작은 모두 아래의 Register들을 어떻게 장난 칠 것이냐가 그 실체이자 정체 인 거죠.

ARM core는 아래의 Register file을 가지고 있고요. 상당히 복잡해 보이지만, 별거 아닙니다. ARM core는동작 mode가 바뀌면 사용하는 Register set도 바뀝니다. 그림과 같이 7개의 동작 mode에 대하여 레지스터 set이 따로 존재하게 되며, ARM core는 32 bit register를 한 mode당 R0~R15까지 16개하고, CPSR + SPSR까지 더해서 18개씩 가지며, 총 7개 mode가 있으니까, 18 registers X 7개 mode = 126개의 Register를 가집니다....아아아아아아... 일줄 알았겠습니다만, 그러면 얼마나 좋으련만 아~ 틀렸습니다.

ARM core는 총 37개의 register를 가지고요. 왜냐! 삼각형이 그려진 register들을 banked register라고 부르며 그것들만이 각 mode별로 따로 있고요 나머지는 mode마다 모두 공용으로 사용합니다.

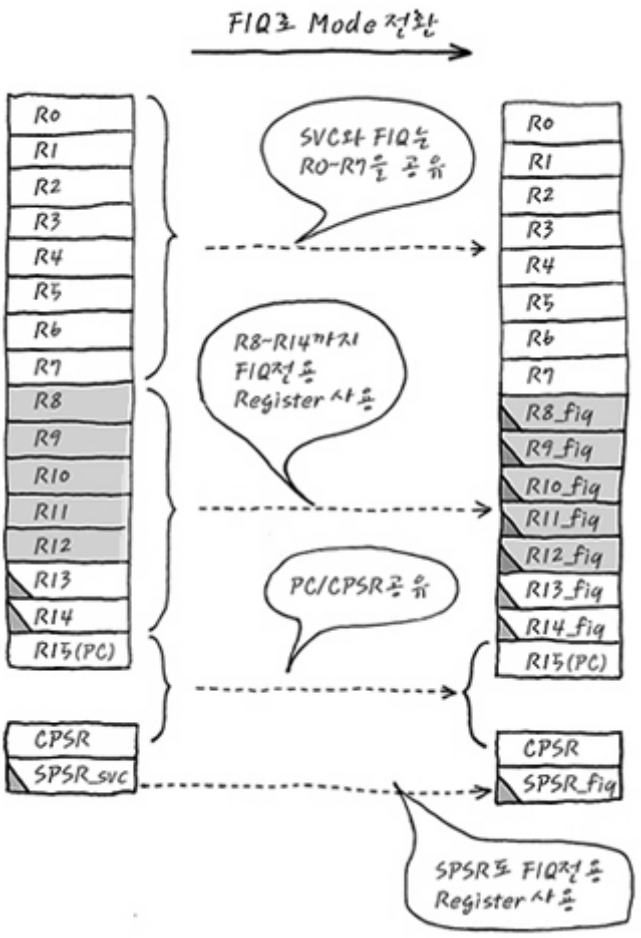


그러니까, 총 R0~R15 (16개) + CPSR (1개) + FIQ의 R8~R14, SPSR (8개) + SVC의 R13, R14, SPSR (3개) + Abort의 R13, R14, SPSR (3개) + IRQ의 R13, R14, SPSR (3개), UND의 R13, R14, SPSR (3개) = 16 + 1 + 8 + 3 + 3 + 3 + 3= 37개 입니다.
이걸 실제 물리적으로 보면 이렇겠죠.. 으흐흐 구멍 숭숭 시원하겠네요.



뭐 여하간 mode가 바뀌면 쓰이는 Register가 바뀐다는 사실만 알면 됩니다. ㅋㅋ
뭐 예를 들면 다음처럼 들 수 있겠네요.

SVC mode로 돌다가 Fast Interrupt가 걸려서 FIQ mode로 진입했을 경우의 Register상태는



인 게지요. 우후후~ 뭐 괜찮아 보여요?

정리하면, 모든 mode가 공유하는 Register는 CPSR, PC, R0~R7까지는 모두가 공유합니다. 다른 말로는 CPSR, PC, R0~R7까지는 ARM core안에 한 개씩 밖에 없다~ 이거지요.

또 특이한 사항 하나는, System mode와 User mode는 완전히 똑같은 register set을 쓴다는 거에요. R0~R15, CPSR을 모두 같이 쓰며 딱 하나의 차이는 CPSR mode bits의 값만이 차이가 납니다. 여기서 주목할 점은 왜 System mode는 SPSR이 없느냐 하는 문제인데, 잘 생각해 보세요. 계속 즐겨볼 Exception에 그 해답이 있습니다.

이렇게 얘기 하다 보니, CPSR이라는 Register의 정체가 큰 역할을 하는 모양인데, 이 녀석의 정체를 살펴 봐야 하겠사옵니다. 도대체 CPSR, CPSR하는데 도대체 이놈은 뭐 하는 넘 이냐. CPSR은 특별한 Register 입니다. 특별한 용도의 Register들은 CPSR, SPSR, R15 (PC), R14 (LR), R13 (SP)가 있어요. 뭐, 그것들을 하나 하나 뜯어 먹어 보면...

CPSR은 약자를 풀어 놓으면 Current Program Status Register가 그 정체이고요. 그 모양새는

이에요.
32bit register이고요. 앞에 NZCV는 Flag field라고 해서, 뭔가 연산한 후에 set되는 register입니다.
이 field는 방금 처리된 연산 결과의 상태를 나타내요.

1) N : Negative : 연산결과가 마이너스인 경우에 set됩니다.
2) Z : Zero : 연산결과가 0인 경우에 set되요.
3) C : Carry : 연산결과에 자리 올림이 발생한 경우에 set 됩니다.
4) V : oVer flow : 연산의 결과가 overflow 났을 경우에 set되는데, Over flow라는건 넘치는 경우니까
원래 가져야 하는 Range보다 결과 값이 큰 경우가 그 경우에 해당됩니다.
이것의 필요성은 ARM의 철학이기도 한데, ARM Assembly section에서 또 다루기는 하겠지만,
ARM core는 Opcode를 Memory에서 가져오자 마자 (Fetch) 이를 무조건 실행하는 것이 아니라
condition flag인 NZCV를 보고 바로 앞 opcode의 실행결과를 보고 실행할지 말지를
결정할 수 있어요. Default는 AL "Always" , condition과 관계없이 항상 실행 이긴 하지만요.
뒤쪽에 IF는 IRQ나 FIQ가 걸릴 수 있는지에 관련한 field로서, 7번째 bit는 IRQ, 6번째 bit는 FIQ
Enable/ Disable을 나타내는 fiied입니다. 1로 set하면 Disable, 0으로 set하면 enable이죠.
이걸로 Interrupt가 걸리지 않도록 control가능합니다.

5번째 T 는 Thumb mode이냐, ARM mode이냐를 나타내는 field로서, ARM/ Thumb mode는
ARM Thumb mode section에서 자세히 다루니까 너무 급하게 생각 안하셔도 되요.

0~4의 5개 bit는 현재의 mode를 나타내는데,
현재 SVC인지, UND 인지, ABT 인지등을 나타냅니다. 이 register가 바로 현재 CPU의 상태
를 나타나겠죠. 아~주 중요한 register입니다.

그러니까, CPSR의 값은 현재의 mode도 확인 가능한 register 인거죠.
실은 CPSR의 하위 5bit를 원하는 mode로 setting하면 그 mode로 전환도 됩니다.
System mode와 User mode는 이 CPSR의 mode bit만 차이가 나고 나머지 Register는
모두 같이 사용합니다. 마치 부부처럼 한 몸입니다. *^^* 부끄.

여기서, SPSR은 또 뭐냐, Saved Program Status Register입니다.
SPSR은 말 그대로 CPSR을 복사해 넣는 특수 Register이지요. 그럼 언제 써먹느냐? CPSR을
backup 할 때 써먹지요. 그럼 언제 backup하느냐? SPSR에 CPSR의 값을 backup해 놓고
mode를 바꾸게 되었을 때, SPSR의 값을 CPSR에 다시 집어 넣으면 이전 mode로
곧바로 복귀할 수 있다는 것이지요!

R14 (Linked Register)는 또 뭐냐. 별거 아닙니다요. ARM은 어딘가로 branch (jump)를 할 때
어디서 branch 해 왔는지를 표시해 둡니다. 헨젤과 그래텔처럼 빵가루를 흘려두는 용도로 사용하는
거에요. 그렇게 하면 함수 같은거 불렀을 때 나중에 어디로 돌아가야 할지 LR만 보면 돌아 갈 수 있겠죠?
으흐흐.

R13 (Stack Pointer)는 현재 Stack을 어디까지 쌓아 두었는지를 가르킵니다.
이 이야그는 나중에 Stack에 대한 소고하고요, 함수의 구조 뭐 이런데서 더 자세히 폭로할 예정인데,
어쨌거나, Stack Pointer가 있으므로 해서 많은 것들이 가능하게 된거죠.

R15 (Program counter) 이 녀석은 현재 어디를 수행하고 있는 건지를 나타내는 거에요.
확장 to the CPU에서도 보았겠지만, 현재 Instruction을 Fetch해 온 위치를 가리키고 있어요. 요거 중요한 사실이니까 잘 기억해
두세요. 실행하는 위치가 아니라 Fetch해온 위치를 가리키는 거다~ 그거죠.

나머지 R0~R12까지는 CPU의 동작중의 저장용도로 적절히 사용되는 거지요. 그래서 R0~R12까지는 적당히 Register라고 부르고,
R13~R15, CPSR, SPSR은 특별한 용도가 있다고 하여, Special Purpose Register라고 까지 구분해요. 쳇.

Register 얘기를 꺼내고 나니, Context라는 거에 대해서 정의를 내리고 가야겠습니다. 이 Context라는 게 상당히 중요한 얘기거든요.
Context를 문맥이라고들 많이 들 번역해 놓았는데, 너무 어렵죠. 저는 Context라는 말을 처음 접했을 때 도대체 이게 무신 소린가
했는데 이제서야 참으로 적당하고 철학적인 말이구나 하고 무릎을 탁 칠 때도 있긴 하거든요. Context Switching을 문맥전환 이라는
둥 해서 표현해 놓았는데, 너무 어렵더란 말이죠.

그래서 제가 고심한 끝에 더 쉬운 말은 없을까~ 해서 이렇게 표현을 하고 삽니다. "상황"

Context라는 건 MCU의 현재 상황, Context Switching이라는 건 상황을 바꿔 침. 뭐 그런 거죠. 그럼 그 상황이라는 건 뭐냐.

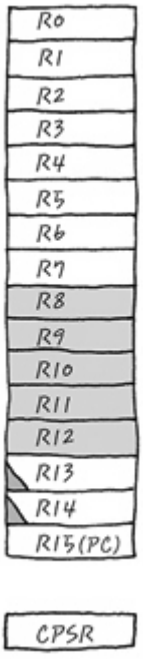
중대장 : 그 쪽 상황은 어때?
이등병 : 현재 김병장님은 야식 먹고 똥 싸러 가셨고, 저는 보초를 잘 서고 있습니다. 이상 무.
중대장 : OK 알겠다.

이렇듯이 모든 것을 종합한 상태를 말하죠. 그럼 이런 Software에서의 상황은 뭐냐?

바로 Register Set의 Snap Shot입니다. 그러니까 R0~R15, CPSR의 값들이 현재 상황이죠. 어느 순간에 찰칵 하고 Register에 들어 있는 값들을 사진 찍는 거에요. 이 값들만 있으면 현재 MCU가 뭘 하려고 했었는지, 뭘 하고 있었는지, 뭘 했는지 등등을 다 파악할 수 있습니다.

이 말은~ 다른 일을 하더라도 특정순간에 저장했던 Context를 다시 복원만 해줄 수 있다면 그 특정순간으로 돌아갈 수 있다~ 는 말입니다. 그 상황을 재현한다고도 말할 수 있겠죠.

결국엔 아래 Register 한 줄이 Context라고 보믄 됩니다. 어느 순간이든 아래의 Register set을 이용해서 MCU가 동작하고 있겠죠?



<나는 context>

완전 그림이 재탕 삼탕 이네요. ㅋㅋ.
요놈의 Context라는 개념을 알고 있으면 어느 Processor에나 들이댈 수 있습니다.
왜? 어느 Processor건 간에 어느 순간에건 이런 Register set을 이용해서 뭔가 하고 있으니까요.


R8~R12는 왜 색깔이 다른가요?
뒤에 나오겠지만 ARM, Thumb mode의 차이가 그 답입니다. Thumb mode는 R0~R7만 쓰고 ARM mode는 R0~R12까지 모두 씁니다. (R13, R14는 ARM/ Thumb에 상관없이 모두 쓰여요)


왜 FIQ는 banked register가 남들보다 많을까요?
그것이 mode가 있는 이유 중에 하나 입죠. 여러 가지 용도의 다른 mode로의 전환 시에 굳이 stack에 register backup을 하나라도 덜하면 시간 절약이 되겠습니다. 그런 이유에서 FAST!!! IRQ mode로의 전환은 더욱 빠르게 되어야 하니까, 전용 register를 하나라도 더 둔거죠. 뭐 쩝.


뭐 아직까지 어리둥절 하겠지만 천천히 더 따라가시면 더 잘 알게 될 터이니 급하게 생각 마세요. 우후훗!

■ SPSR의 용도는 다음 Exception에서 만나요~

Linked at at 2009/10/01 12:32

... ⓔ ARM Register와 Context<a href="http://recipes.egloos.com/4986854" designtimesp="8183"> ... more

Commented by SMILE at 2009/07/07 10:13

오늘도 한가지 머리에 넣고 떠납니다.
또 낼도~
감사

Commented by 히언 at 2009/07/07 22:02

완전 감사합니다. 저야말로 매일 머리에 넣어야 할텐데 쩝...