친절한 임베디드 시스템 개발자 되기 강좌 : ARM/ Thumb PCS

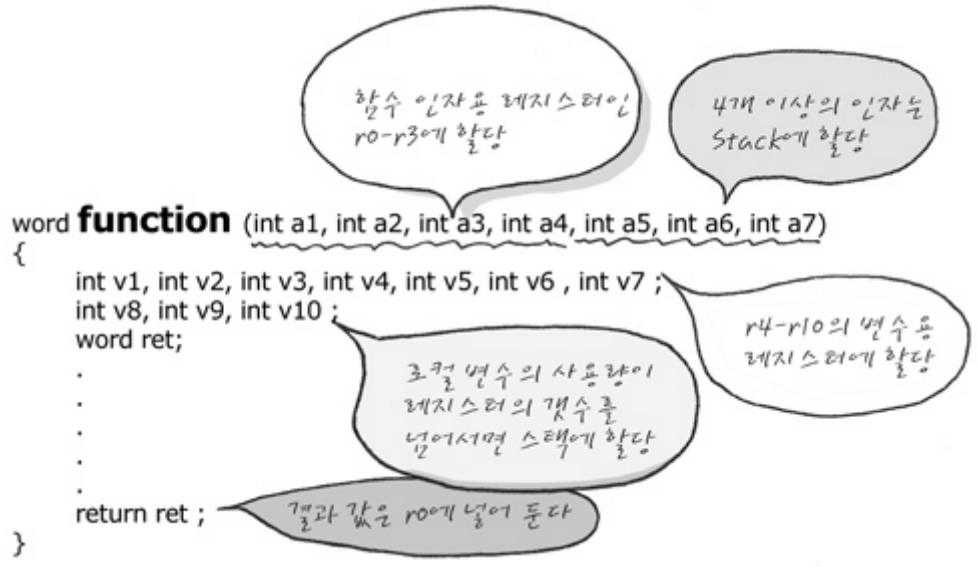
자자, ARM mode와 THUMB mode에 관한 얘기를 했고, 이 차이에 대해서는 ARM 구현 쪽에서 더 자세히 살펴 볼 테니, 이 정도로 담 넘는 구렁이 친화적인 상태로 은근슬쩍 넘어갈래요. 좀 이른 감이 없지 않아 있지만, Software얘기를 섞여 있는데, 어차피 C정도는 다뤄 보셨을 테니까, 대충~ 감만 잡고 넘어가시고, 본격적으로 Software 얘기할 때 다시 한번 더 봐주시면 감사. - 때 되면 다시 읽어 보시라고 comment할게요 - 일단은 ARM 내부의 Register얘기를 실컷 했는데요, Register에 관해서 얘기할 때, PC라든지, LR이라든지 하는 특수용법의 Register들이 은근슬쩍 나왔죠. 실은 특수용법의 Register이외의 일반 Register들도 쓰이는 쓰임새가 있습니다. 이런 약속을 APCS 즉, ARM Procedure Call Standard라고 부르고요, 이 Standard에 맞추어서 Compiler는 기계어를 만들어 낸답니다. 그런 APCS 약속들 이란 게 어떤 것들이냐?1) 함수를 부를 때 Register는 어떻게 사용해야 하는가? 2) return 값은 어떻게 돌려주는가? 3) Stack은 어떨 때, 어떻게 사용되는가? 4) 1)~3)에서 사용된 Register 이외의 Register들은 어떻게 사용되는가? 뭐 이런 것들이 있겠죠. 잠깐, 요 유명한 APCS라는 게, 2008년 10월 10일에 ARM이 새로운 standard를 내 놓았는데, 어차피 다 비스므리 합니다만, 이름을 다르게 붙였네요. 이런 Register 사용법을 총칭하여PCS (Procedure Call Standard)라고 부르고요, APCS : ARM Procedure Call Standard (구버전)TPCS : Thumb Procedure Call Standard (구버전)ATPCS : ARM-Thumb Procedure Call Standard (AAPCS의 선배)AAPCS : Procedure Call Standard for ARM Architecture (현재 최신버전)라고 이름 붙였네요. 결국 지금 사용되는 Procedure Call Standard (Register 사용법)은 이름 하야 AAPCS라고 부르는 게 맞겠습니다. 이 용법을 지금 잘 알아두면, 이후에 함수의 구조, Stack의 사용 등을 이해하기 쉬우니까, 꼭 알아 두셔야 해요. AAPCS에 의한 각 Register의 사용법은 Table과 같습니더.

Register	Synonym	Special	Role in the procedure call standard
r15		PC	The Program Counter.
r14		LR	The Link Register.
r13		SP	The Stack Pointer.
r12		IP	The Intra-Procedure-call scratch register.
r11	v8	FP	ARM-state variable-register 8. ARM-state frame pointer.
r10	v7	SL	ARM-state variable-register 7. Stack Limit pointer in stack-checked variants.
r9	v6	SB	ARM-state v-register 6.
r8	v5		ARM-state variable-register 5.
r7	v4		Variable register (v-register) 4.
r6	v3		Variable register (v-register) 3.
r5	v2		Variable register (v-register) 2.
r4	v1		Variable register (v-register) 1.
r3	a4		Argument / result / scratch register 4.
r2	a3		Argument / result / scratch register 3.
r1	a2		Argument / result / scratch register 2.
r0	a1		Argument / result / scratch register 1.

오호, R0~R15까지의 Register들에 대한 설명들이 나와있네요. . **Synonym** Table에 대해서 약간의 첨언을 하자면, Synonym은 R0~R11까지의 Register들을 다른 이름으로 부를 수도 있다는 걸 의미하고요, Special이라고 쓰여진 부분은 죄수번호 부르듯이 번호로 부르는 것 이외에, 더욱 Special한 기능을 한다고 하여, 이름을 또 하나 붙인 셈이죠. 이중에 가장 많이 사용하는 건, R0(a1), R1(a2), R2(a3), R3(a4), SP, LR, PC 정도고요. 요 녀석들은 꼭 알아 두셔야 합니다. 나머지는 외우지는 못하더라도, 필요할 때 찾아볼 수만 있으면 됩니다. 으흐흐.

Table의 한칸 한칸을 즈려 밟고 올라가 볼까요? . **R0~R3 (a1~a4) Descriptions (Argument/ Result/ Scratch)** . **1) Argument** R0(a1)~R3(a4)에 대한 설명이 다 똑같습니다. Argument/ Result/ Scratch Register. 일단은 R0~R3 까지는 함수에서 Argument를 넘길 때 사용합니다. 예를 들어,
int **function** (int a, int b, int c, int d)

라는 함수가 있다고 했을 때, 이 함수를 실제로 사용할 때는 (void)function (10, 20, 30, 40); 이런 식으로 사용하겠죠? 이럴 때 기본적으로 R0 := 10, R1 := 20, R2 := 30, R3 := 40 요런 식으로 Register를 사용하게 됩니다. 또한 이 함수의 원형이 Integer return type이므로, 이 함수의 결과 값이 100이라고 한다면, 함수의 return 값을 R0에 넣는 게 상례 입니다. f = function (10, 20, 30, 40); 함수 호출 시에는 R0 := 10, R1 := 20, R2 := 30, R3 := 40 이고요, 함수 return시에는 R0 : = 100, 결국 f = 100. 그러니까, 어떤 함수가 불렸을 때, 불리자 마자의 R0 ~ R3값은 Argument, 돌아올 때 R0의 값이 return값이라고 이해 하시면 편리합니다. 이때 R0~R3의 값은 function() 함수에 갔다 왔을 때 다른 값이 될 수도 있으므로, 즉, 호출한 함수 쪽에서는 이러한 R0~R3의 값이 예측 불가능한 값으로 돌아올 수도 있으므로, 만약 function()함수를 부르는 쪽의 함수가 R0~R3이 필요하다면 따로 backup을 해두는 수고로움을 해줘야 합니다. .**2) Result (Return Value)** 이때 result값이 R0~R3 모두에 걸쳐 있는데 이건 또 뭐냐! return값이 여러 개냐! 하시는 분이 있다면. 뭐 이런 겁니다. R0~R3에 pointer 값을 전달하게 되면, 실제 주소 값을 전달하여, 호출 되는 쪽 함수에서 장난을 치겠죠. 그럴 때, 돌아오는 순간에 그 주소 값을 그대로 다시 R1~R3에 넣어주면 다시 호출한 쪽에서 return값처럼 R1~R3를 그대로 가져다 쓰면 되겠죠. 부른 입장에서는 R1~R3의 값이 바꾸지 않는 것이라고 보믄 됩니다. 또는 Register 1개는 32bit 즉 4byte이니까, return값이 4byte보다 큰 녀석을 return할 때는 다른 Register들도 사용해야겠죠. 그러니까 최대 4byte * 4개 = 16byte까지 한번에 return할 수 있습니다. 그럼 4개 초과의 argument를 전달하면 어떻게 되느냐?! 4개 초과의 argument는 일단 stack에 push되는 것이죠. 음.. 이 시점에서 벌써 stack이라는 말이 나오니 어정쩡하지만, ARM Assembly section이후에 더 자세히 나올 예정이랍니다. 우후훗! 3) Scratch (연습장) 그럼 또, scratch란 건 또 뭐냐, 별거 아닙니다. 일단 어떤 함수가 불리면서 argument의 전달 수단으로 R0~R3이 쓰이고 나면, 불려진 함수 내부에서는 돌아갈 때 APCS의 약속만 잘 지켜주면, 불려진 함수 내에서는 그 Register들을 CPU가 마치 연습장처럼 임시저장 용도로 마구 사용 가능해요. 즉, 다른 용도로 쓸 수 있다는 뜻이에요. 별거 아니죠? 뭐, 이런 내용을 대충 그림으로 보면..



R4~R11 (v1~v8) R4~R11은 variable 용으로서, 함수 호출 후에 바뀌어서는 안 되는 값들로서, 호출당한 함수는 R4~R11을 사용하려면, stack에 저장 후에 사용하고, 호출당한 함수가 끝나기 전에 복원을 해줘야 합니다. . **R12~R15 (특수 Register들)** R12 (IP, Intra..) : ARM-Thumb interworking등에 또는 long branch시에 Veneer를 통해 주소 할당 시에 임시 보관소로 사용함. (음.. 어렵죠.. 이건 Veneer를 이해하면 좀 쉬울 거에요.) R13 (SP, Stack Pointer): Stack을 사용하기 위해 Stack Pointer로 사용합니다. R14 (LR, Link Register): 함수를

호출하거나, 어디론가 jump했을 때 돌아올 주소를 넣는 목적으로 사용해요. R15 (PC, Program Counter): 현재 실행하고 있는 주소. (Pipe line을 고려하면 Fetch하고 있는 주소) 뭐 이런 셈이죠. 실은 갑자기 이런 얘기를 꺼내니까 당황스러우시겠지만, 소프트웨어 얘기가 무르익어가면 굉장히 중요한 얘기로 떠오르니까 지금 잘 알아두세요. 좀 복잡 시끄러우면 나중에 다시 읽어 보셔도 무방합니다. 그럼 이런 규칙을 따르려면 어떻게 해야 되느냐!!! Compiler Option을 -apcs라고 주면 compiler가 이런 AAPCS 규칙에 따라 Compile을 해줍니다. 그리고, 마지막으로 하고 싶은 얘기는 R13 (SP), R14 (LR)도 General Register로 사용할 수 있습니다. 내가 원하면 뭔들 못하겠어요? 하지만 R13과 R14의 경우에 General Register로 쓰면 좀 위험합니다. 보통, R13의 경우 ARM을 기반으로 하여 ARM을 지원하는 OS들은 R13이 항상 유효한 Stack Pointer임을 가정하고 구현되어 있습니다. 또한 함수 호출에 관련해서도 R14는 유효한 Linked Register 값이 들어 있음을 compiler가 가정하기 때문에 그렇습니다.