

2009년 06월 13일

Assembly로 만드는 방법

spaghetti.c 를 Assembly로 만든다는 건, ARM core에서 동작하는 Assembly로 spaghetti.c를 가공하는 일입니다. spaghetti.c를 Assembly로 만드는 일 역시 tcc, armcc가 한꺼번에 알아서 해주니까 해 볼일은 없겠지만, 한번 해보는 것도 의미는 있으리라 생각합니다. - 과연 의미 있을지는 두고 봐야 하겠지요 하하 -

RAM에 아주~ 간단한 ram 상주 프로그램을 올릴 때, 이런 기법이 사용되기도 하니, Software 동작원리 이후에 아주 재미있는 간단한 요리법을 하나 알려드리지요. 잊지 마세요!

Assembly로 만드는 방법은 아주 간단합니다. -S option을 사용하지요

```
.
tcc -S spaghetti.c

saghetti.s -----

; generated by Thumb C Compiler, ADS1.2 [Build 805]

; commandline [-O2 -S -IC:\Wapps\Wads12\INCLUDE]      CODE16

        AREA ||.text||, CODE, READONLY

main PROC

        MOV     r0,#3

        MOV     r1,#4

        ADD     r0,r0,r1

        ADD     r0,r2,r0

        BX      lr

        DCW     0000

ENDP


        AREA ||.bss||, NOINIT, ALIGN=2

||zi||||.bss$7||      % 4

        AREA ||.data||, DATA, ALIGN=2

||.data$10||||rw||      DCD      0x00000003

        AREA |area_number.1|, DATA, ALIGN=2      EXPORTAS |area_number.1|, ||.data||||.data$15||relocate      DCD      0x00000003

EXPORT main      EXPORT relocate      EXPORT ||rw||      EXPORT ||zi||

IMPORT __main      IMPORT __main      IMPORT ||Lib$$Request$$armlib||, WEAK

KEEP

||BuildAttributes$$THUMB_ISAv1$M$PE$A:L22$X:L11$S22$~IW$~STKCKD$~SHL$OSPACE$PRES8||||BuildAttributes$$THUMB_ISAv1$M$PE$A:L22$X:L11$S22$~IW$~STKCKD$~SHL$OSPACE$PRES8||
EQU 0

        ASSERT {ENDIAN} = "little"      ASSERT {SWST} = {FALSE}      ASSERT {NOSWST} = {TRUE}      ASSERT {INTER} = {FALSE}      ASSERT {ROPI} = {FALSE}      ASSERT {RWPI} =
{FALSE}      ASSERT {NOT_SHL} = {TRUE}      ASSERT {FULL_JEE} = {FALSE}      ASSERT {SHL1} = {FALSE}      ASSERT {SHL2} = {FALSE}      END
```

으악! 간단하다고 하더니, 막상 해보니까, 엄청 긴 파일이 생성이 되어 버렸네요. 하지만, 기죽을 거 없습니다.
다 필요 없는 얘기들이고, spaghetti.c 의 main 함수가 어떻게 컴파일 되었는지 만 확인해 보면 될 것 같습니다. 다른 것들이야 어떻게 되든 말든. 흥.

원래 c로 되어 있는 main 함수

```
int main ()
{
    int stack; volatile int local,local2,loca3; local = 3; local2 = 4; local3 = add (local, local2); stack += local3; return stack;
}
```

Assembly의 main 함수

```
main PROC

        MOV     r0,#3      ; r0 = 3

                                ; local = 3

        MOV     r1,#4      ; r1 = 4      ; local2 = 4

        ADD     r0,r0,r1      ; r0=r0+r1      ; local3 = local + local2
```

```
ADD    r0,r2,r0    ; r0=r2+r0    ; stack = stack + local3


BX     lr

DCW    0000

ENDP
```

참고로 위의 함수 중 main PROC는 main이라는 녀석이 함수라는 뜻이고 그 끝은 ENDP로 표기됩니다.

가만히 보니까 원래 main() 함수하고 비교해 보면, local (r0) = 3, local2 (r1) = 4를 해주고 나서, local3 (r0) 에다가 local (r0) + local2(r1)을 넣어주고, 다시, stack (r0) = stack (r2) + local3 (r0)를 해준다 그게 요점이며, 결국 return 값 r0에다가 stack 값을 넣는 것으로 assembly가 구성되어 있습니다. 일단, main() 함수 중 add (local, local2)에서, 함수를 호출하지 않은 이유는 compiler 입장에서 코드를 잘 훑어 보니, 굳이 함수를 호출할 것이 아니라, "두 개의 인자를 더하는 함수다" 라는 사실을 눈치를 채버려서, compiler가 할 일은 최적화라는 입장을 잊지 않고, 똑같은 효과의 assembly를 만들어 낸 것이지요. Compiler라는 녀석은 상당히 교활한 녀석이면서도 똑똑한 녀석입니다.

 사실 이 Assembly의 register 관계에서 ATPCS (ARM Thumb procedure call standard)라는 규약이 적용되어 있는데, "함수의 호출과정"편에서 더 자세히 다루는 편이 훨씬 마음에 와 닿을 것 같으니, 지금은 이러니 저러니 해서 이런 식으로 assembly가 만들어 지는구나 정도만 음음 하고 생각하셔도 될 것 같습니다.

Linked at at 2009/10/01 12:32

... (f) Assembly로 만드는 방법 ... [more](#)