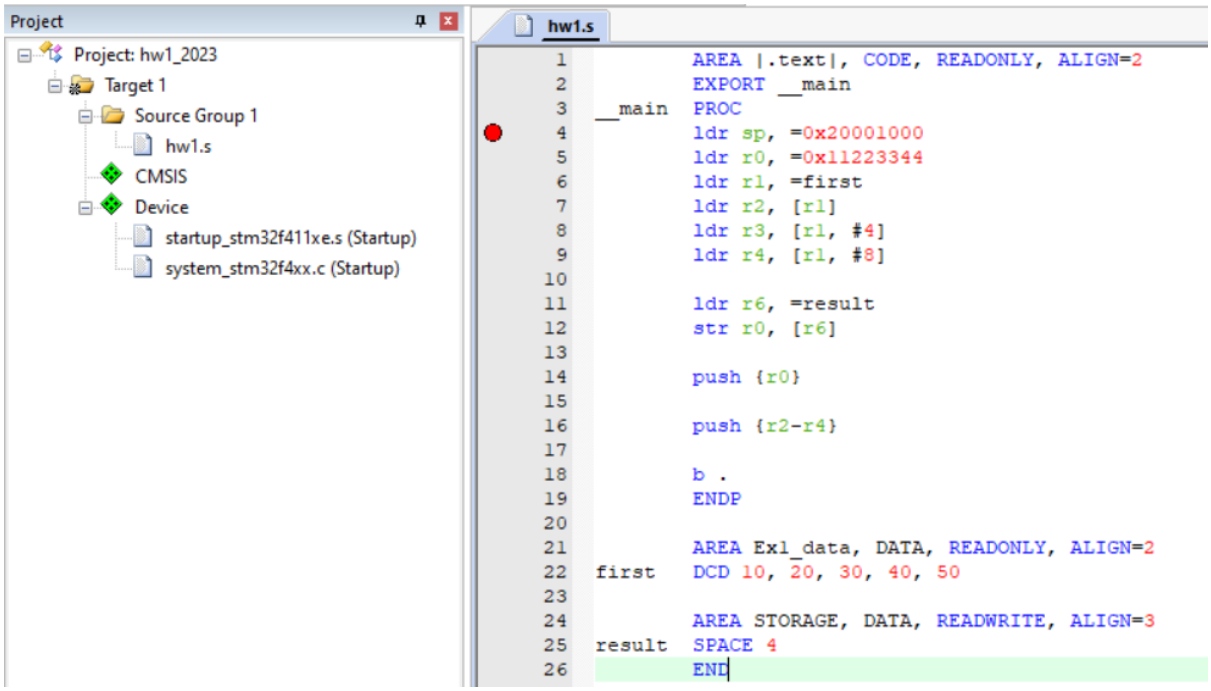
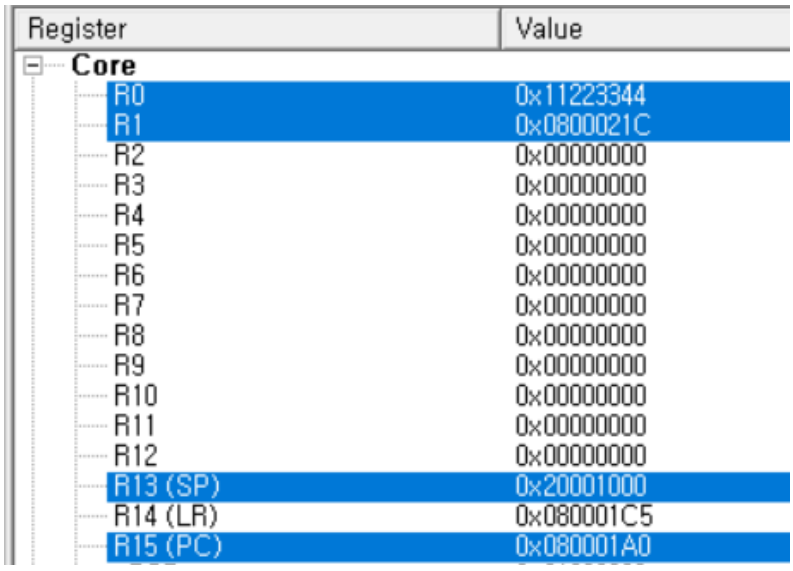
HW1

(교재 p67! 에 있는 MKD-ARM에서 project 생성방법을 차조하고, p.127 ~에 있는 simulator 사용 방법을 참조하십시오.)

다음과 같은 형태의 project를 만들어서 동작을 확인하십시오.



1. line 4 ~ 6까지를 debugger를 이용하여 실행한 다음, registers r0, r1, sp의 값이 어떻게 되었는지 결과를 capture 하여 제출하시오. 그리고 ldr <Rd>, =#<immediate number> 와 같은 형태의 명령어은 어떻게 동작하는 가를 설명하시오. (10점)



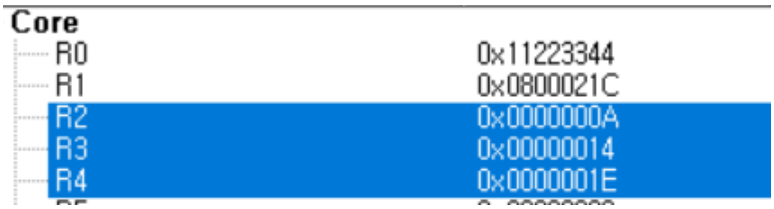
이 instruction은 target register에 주어진 immediate number를 할당하도록 하는 명령어이다, 즉 sp에 0x20001000, r0에 0x11223344, 그리고 r1에 first label이 위치한 메모리 주소값이 할당된다.

원래 ARM assembly instruction에 포함된 instruction은 아니고, programmer가 program을 용이하게 하기 위하여 만들어진 pseudo instruction이다. 즉 어떤 register에 직접 값을 할당하는 경우는 mov instruction을 사용하여 하지만 이 경우 주어지는 값은 제한이 있다. (why?)

Disassembly window에서 보는 바와 같이 이 pseudo instruction은 ldr.w <Rd>, [pc, #immediate number] 형태로 바뀌어 실행된다. (어떻게 동작하여야 위에서 언급한 바와 같이 동작할 수 있을까?)

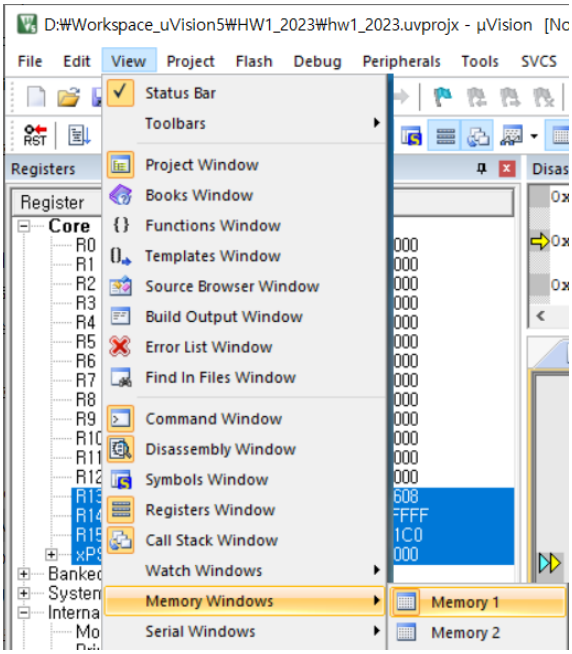
```
     4:                 ldr sp, =0x20001000
0x08000198 F8DFD014  LDR.W          sp,[pc,#20]   ; @0x080001B0
     5:                 ldr r0, =0x11223344
0x0800019C 4805      LDR            r0,[pc,#20]   ; @0x080001B4
     6:                 ldr r1, =first
0x0800019E 4906      LDR            r1,[pc,#24]   ; @0x080001B8
```

2. line 7-9까지를 debugger를 이용하여 실행한 다음, registers r2, r3, r4의 값이 어떻게 되었는지 결과를 capture하여 제출하시오. 그리고 ldr <Rd>, [<rd>, #<immediate number>]와 같은 형태의 명령어는 어떻게 동작하는 가를 설명하시오. (10점)



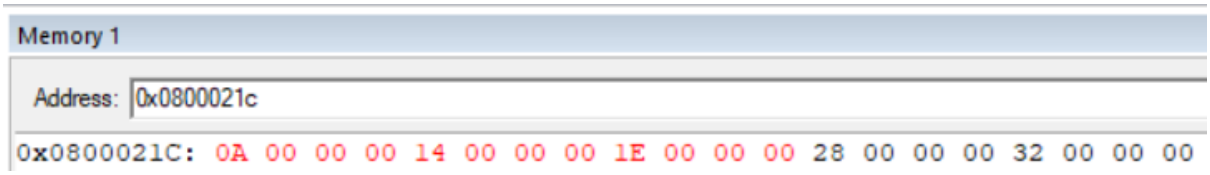
이 instruction은 <rd>+immediate number의 메모리 주소에 저장된 내용을 읽어서 register ㄱㅇ에 load하라는 명령어이다.

3. line 22의 동작을 설명하고, memory 창으로부터 어떻게 저장되어 있는 가를 capture하여 제출하시오. (메모리 내용을 보려면 debugger를 동작시키고, 아래 그림과 같이 View->Memory Windows -> memory1 (or 2,3,4)를 선택한 다음, 나타난 메모리 창에서 관찰하고자 하는 메모리를 주소를 입력하면 된다. 단 메모리 내용이 byte 단위로 표시되도록 한 상태에서 capture하시오.)



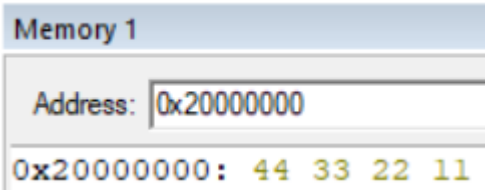
DCD는 Define Constant Data의 약어로 해당 메모리 주소부터 4 byte 단위롤 주어진 값을 할당하라는 의미이다. 이와 같은 것을 Directive라고 한다. (즉 assembler에게 주어지는 지시어로 생각하면 된다.)

위에서 보는 바와 같이 label "first"의 메모리 주소가 0x0800021C이므로 이 주소 영역에 저장되어 있는 매용을 살펴보면 된다.



메모리 창에서 보는 바와 같이 0x0800021C 번지에 0x0000000A (=10), 0x08000220 번지에 0x00000014 (20), 0x08000224 번지에 0x0000001E (30), 0x08000228 번지에 0x00000028 (40), 0x0800022C번지에 0x00000032 (50) 값이 저장되어 있다.

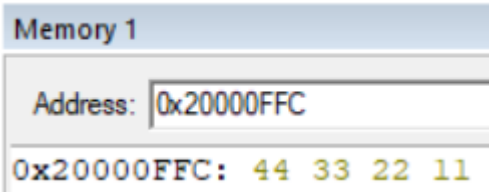
4. line 11, 12를 실행시킨 다음, 메모리에 어떤 내용이 어떻게 저장되었는 가를 capture하여 제출하시오. 그 결과에 의하면 해당 processor는 little endian 으로 동작하는지, 아니면 big endian 방식으로 동작하는 가를 설명하시오. (10점)



Label result는 0x20000000 번지를 나타내고 있다. (register r6 참조)

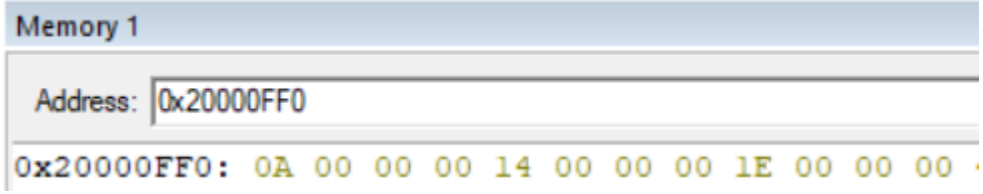
Register r6가 지정한 메모리에 r0 =0x11223344를 저장하는 명령어를 실행하면 위에 캡쳐한 형태로 r0 값이 저장된다. 0x20000000 번지에 0x44 즉 LSB가 저장되고 있기 때문에 little endian으로 동작함을 알 수 있다.

5. line 14를 실행시킨 다음, 결과를 capture하여 제출하시오. (SP 값은 어떻게 변하였는가? Stack 어떤 메모리 주소에 어떻게 저장되었는가?) 그렇게 동작하는 이유를 설명하시오. (10점)



원래 명령 실행 전 sp = 0x20001000 이었지만 push 명령 후에는 sp = 0x20000FFC 즉 '4' 만큼 감소되었다. 그리고 0x20000FFC – 0x20000FFF 메모리 영역에 각각 x044, 0x33, 0x22, 0x11이 저장됨을 알 수 있다. 즉 여기서도 little endian으로 동작하기 때문에 메모리 주소가 낮은 쪽에 LSB가 저장됨을 알 수 있다.

6. line 16을 실행시킨 다음, 결과를 captured하여 제출하시오. (stack에 저장된 내용과 SP 변경 값에 대해서.) 그렇게 동작하는 이유를 설명하시오. (10점)



3개의register를 stack에 저장하였기 때문에 sp 값은 12가 감소된다 즉 0x20000FFC – 0xC 가 되어서 0x2000FF0가 되고, 0x20000FF0 메모리 영역을 살펴보면 0x200000FF0-0x20000FF3 영역에 r2 값이, 0x20000FF4-0x20000FF7 영역에 r3 값이, 0x20000FF8-0x20000FFB 영역에 r4 값이 저장됨을

알 수 있다. 즉 여러 register 들이 하나의 명령어에 의해서 메모리에 저장될 때, register number 가 작은 register가 낮은 메모리 주소 영역에 저장됨을 알 수 있다.