

LDR ? ; 메모리 값을 메모리로 복사

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.

이 주소에 M 주소에 S를 넣어
메모리에 쓰기

```
Project: hw1_2023
  Target 1
    Source Group 1
      hw1.s
    CMSIS
    Device
      startup_stm32f411xe.s (Startup)
      system_stm32f4xx.c (Startup)

hw1.s
1 AREA .text!, CODE, READONLY, ALIGN=2
2 EXPORT __main
3 _main
4 PROC
5 ldr sp, =0x20001000
6 ldr r0, =0x11223344
7 ldr r1, =flist
8 ldr r2, [r1]
9 ldr r3, [r1, #4]
10 ldr r4, [r1, #8]
11 ldr r6, =result
12 str r0, [r6]
13
14 push {r0}
15
16 push {r2-r4}
17
18 b .
19 ENDP
20
21 AREA Ex1_data, DATA, READONLY, ALIGN=2
22 first DCD 10, 20, 30, 40, 50
23
24 AREA STORAGE, DATA, READWRITE, ALIGN=3
25 result SPACE 4
26 END
```

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

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

3. line 22의 동작을 설명하고, memory 창으로부터 어떻게 저장되어 있는 가를 capture하여 제출 하시오. (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.)

1. sp ?

Registers

Register	Value
Core	
R0	0x08000199
R1	0xE000ED88
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20001000
R14 (LR)	0x080001C5
R15 (PC)	0x0800019C
xPSR	0x01000000

Disassembly

```

0x08000190 01DB    DCW      0x01DB
0x08000192 0800    DCW      0x0800
0x08000194 01DB    DCW      0x01DB
0x08000196 0800    DCW      0x0800
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
7:           ldr r2, [r1]
0x080001A0 680A  LDR      r2,[r1,#0x00]
8:           ldr r3, [r1, #4]
0x080001A2 684B  LDR      r3,[r1,#0x04]

```

hw1.s*

```

1      AREA .text!, CODE, READONLY, ALIGN=2
2      EXPORT __main
3      _main PROC
4          ldr sp, =0x20001000
5          ldr r0, =0x11223344
⇒ 0x0800019E 4906  ldr r1, =first
6          ldr r2, [r1]
7          ldr r3, [r1, #4]
8          ldr r4, [r1, #8]
9
10

```

堆에 4바이트을 실행한 후
堆의 주소는 0x11223344
원래 주소는 R1(51)이
value는 초기화되었음
0x20001000로 되어 있음
?!

r0 ?

Registers

Register	Value
Core	
R0	0x11223344
R1	0xE000ED88
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20001000
R14 (LR)	0x080001C5
R15 (PC)	0x0800019E
xPSR	0x01000000

Disassembly

```

0x08000190 01DB    DCW      0x01DB
0x08000192 0800    DCW      0x0800
0x08000194 01DB    DCW      0x01DB
0x08000196 0800    DCW      0x0800
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
7:           ldr r2, [r1]
0x080001A0 680A  LDR      r2,[r1,#0x00]
8:           ldr r3, [r1, #4]
0x080001A2 684B  LDR      r3,[r1,#0x04]

```

hw1.s*

```

1      AREA .text!, CODE, READONLY, ALIGN=2
2      EXPORT __main
3      _main PROC
4          ldr sp, =0x20001000
5          ldr r0, =0x11223344
6          ldr r1, =first
7          ldr r2, [r1]
8          ldr r3, [r1, #4]
9          ldr r4, [r1, #8]
10

```

堆에 0x20001000로 되어 있음
堆의 주소는 R0(51)
堆의 주소는 0x11223344로 되어 있음
堆의 주소는 초기화되었음

r1 ?

Registers

Register	Value
Core	
R0	0x11223344
R1	0x080001C5
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20001000
R14 (LR)	0x080001C5
R15 (PC)	0x080001A0
xPSR	0x01000000

Disassembly

```

0x08000190 01DB    DCW      0x01DB
0x08000192 0800    DCW      0x0800
0x08000194 01DB    DCW      0x01DB
0x08000196 0800    DCW      0x0800
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
7:           ldr r2, [r1]
0x080001A0 680A  LDR      r2,[r1,#0x00]
8:           ldr r3, [r1, #4]
0x080001A2 684B  LDR      r3,[r1,#0x04]

```

hw1.s*

```

1      AREA .text!, CODE, READONLY, ALIGN=2
2      EXPORT __main
3      _main PROC
4          ldr sp, =0x20001000
5          ldr r0, =0x11223344
6          ldr r1, =first
7          ldr r2, [r1]
8          ldr r3, [r1, #4]
9          ldr r4, [r1, #8]
10

```

64bit로 32비트로 된 경우
堆의 주소는 0x0800021C이 됨
0x0800021C로 되어 있음

$\lambda \text{dr } \langle R \downarrow \rangle, = \# \langle \text{immediate number} \rangle$

RJ 2121 스타일 immediate number 상수를 확장 해준다,

#은 불이느 까우느 160ES를 초과 하거나 음수로 깨어에 불여준다

2.

r_2 ?

The screenshot shows the QEMU debugger interface with three tabs: Registers, Disassembly, and Assembly.

- Registers Tab:**

Register	Value
Core	
R0	0x11223344
R1	0x080001C
R2	0x000000A
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20000100
R14 (LR)	0x080001C5
R15 (PC)	0x080001A2
xPSR	0x01000000
- Disassembly Tab:**

```

0x08000190 01DB    DCW      0x01DB
0x08000192 0800    DCW      0x0800
0x08000194 01DB    DCW      0x01DB
0x08000196 0800    DCW      0x0800
    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
    7:           ldr r2, [r1]
0x080001A0 680A    LDR      r2,[r1,#0x00]
    8:           ldr r3, [r1, #4]
0x080001A2 684B    LDR      r3,[r1,#0x04]
    <

```
- Assembly Tab:**

```

hw1.s*
1      AREA .text!, CODE, READONLY, ALIGN=2
2      EXPORT _main
3      _main PROC
4          ldr sp, =0x20001000
5          ldr r0, =0x11223344
6          ldr r1, =first
7          ldr r2, [r1]
8          ldr r3, [r1, #4]
9          ldr r4, [r1, #8]
10
11         ldr r6, =result
12         str r0, [r6]
13

```

$$R_2 = Q_X 0000000A$$

Registers

Register	Value	
Core		
R0	0x11223344	
R1	0x0000001C	
R2	0x00000000	
R3	0x00000014	
R4	0x00000000	
R5	0x00000000	
R6	0x00000000	
R7	0x00000000	
R8	0x00000000	
R9	0x00000000	
R10	0x00000000	
R11	0x00000000	
R12	0x00000000	
R13 (SP)	0x20001000	
R14 (LR)	0x080001C5	
R15 (PC)	0x080001A4	
xPSR	0x01000000	

Disassembly

```

7:           ldr r2, [r1]
             r2,[r1,#0x00]
8:           ldr r3, [r1, #4]
             r3,[r1,#0x04]
9:           ldr r4, [r1, #8]
10:          ldr r4, [r1, #8]
11:          ldr r6, =result
12:          ldr r6, [pc,#20] ; @0x080001BC
13:          str r0, [r6]
14:          push {r0}
15:

```

hw1.s*

```

1      AREA .text!, CODE, READONLY, ALIGN=2
2
3      EXPORT __main
4      _main PROC
5          ldr sp, =0x20001000
6          ldr r0, =0x11223344
7          ldr r1, =first
8          ldr r2, [r1]
9          ldr r3, [r1, #4]
10         ldr r4, [r1, #8]
11         ldr r6, =result

```

$$R_3 = 0x0000\ 0014$$

The screenshot shows the QEMU debugger interface with three main panes:

- Registers** pane: Shows the ARM寄存器 (Register) and Value for each register. The R4 register has its value highlighted with a red circle.
- Disassembly** pane: Shows the assembly code with addresses and opcodes. The instruction at address 0x080001A6 is highlighted with a yellow box.
- Assembly** pane: Shows the assembly code for the current function, with labels like AREA, PROC, and EXPORT.

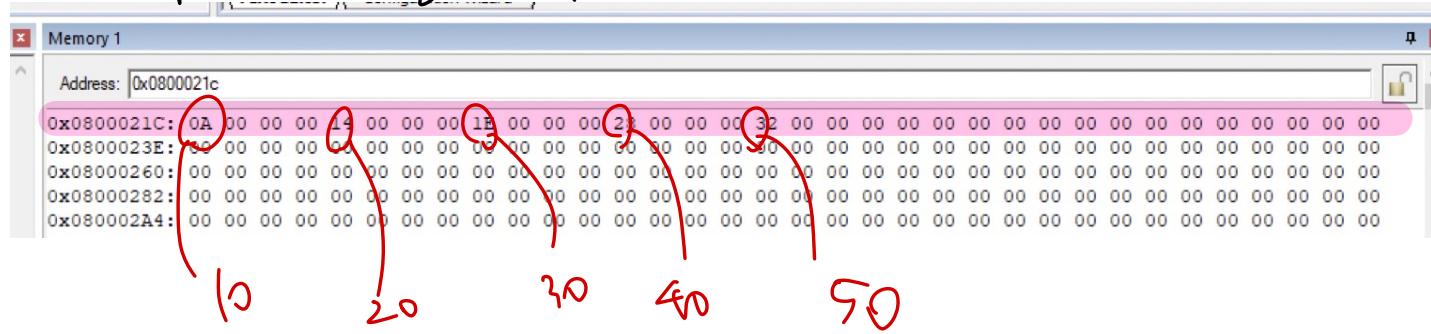
$$R_f = 0x000000/E$$

ldr <Rd>, [<rd>, #<immediate number>] ?.

<rd> 와 같은 주소값을 나☞널, <rd> 와 주소값을
기준으로 immediate number 만큼 주소값을 이동 시킬 (byte 단위)
로 확장한 주소의 메모리에서 값을 꺼내와 <rd>에
넣어준다.

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.)

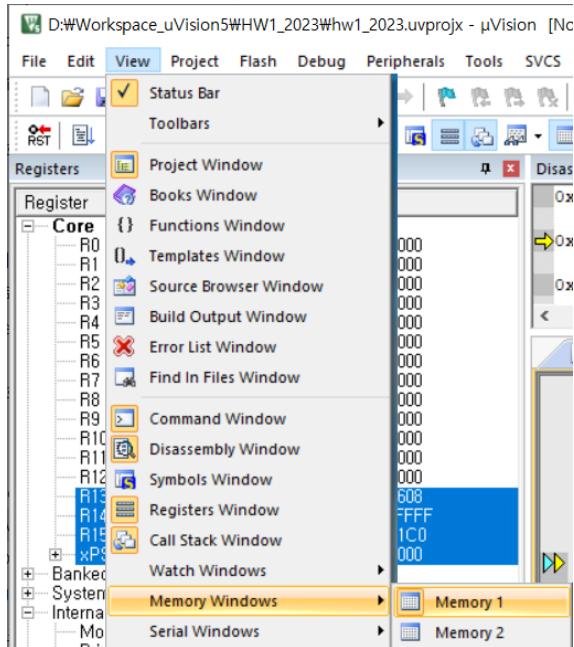
$$r_r = 0 \times 0800021C$$



First 는 맨 처음 배열의 주소 값을 가지고 나온다. DC0 는
32bytes (4 bytes) 크기의 정수로 배열을 만들 때 사용된다.

10, 20, 30, 40, 50은 r에 대한 최대값을 주는 3

부터 32-bits 간격으로 각각 저장되어 있다.



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

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

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

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

result of memory loss after stroke.

The screenshot shows the JTAG-ICE debugger interface with three main panes:

- Registers**: Shows the state of various registers. The R6 register is highlighted with a red circle around its value, 0x20000001.
- Disassembly**: Shows the assembly code being executed. The current instruction at address 0x080001A8 is highlighted in yellow: STR r0, [r6, #0x00].
- Assembly**: Shows the source code for the program. The `main` function is defined as follows:

```
1 AREA .text!, CODE, READONLY, ALIGN=2
2 EXPORT __main
3 __main PROC
4 ldr sp, =0x20001000
5 ldr r0, =0x11223344
6 ldr r1, =first
7 ldr r2, [r1]
8 ldr r3, [r1, #4]
9 ldr r4, [r1, #8]
10
11 ldr r6, =result
12 str r0, [r6]
13
```

A handwritten note "result" is written next to the `ldr r6, =result` line, and "2" is written next to the `str r0, [r6]` line.

리메이크 주소가
제 깊됨

The screenshot shows the JTAG-ICE debugger interface with three main panes:

- Registers**: Shows the CPU register values. The PC (R15) is highlighted at 0x080001AA.
- Disassembly**: Shows the assembly code corresponding to the memory addresses. The instruction at R15 is highlighted as a PUSH {r0}.
- Assembly**: Shows the source code for the hw1.s program. The main function is defined with an EXPORT directive. The assembly output matches the disassembly pane.

A handwritten note on the right side of the screen says "0x11223344 (r)" and "Result all 2 of 2".

```

Registers
Register Value
Core
  R0      0x11223344
  R1      0x0800021C
  R2      0x0000000A
  R3      0x00000014
  R4      0x0000000E
  R5      0x00000000
  R6      0x20000000
  R7      0x00000000
  R8      0x00000000
  R9      0x00000000
  R10     0x00000000
  R11     0x00000000
  R12     0x00000000
  R13 (SP) 0x20001000
  R14 (LR) 0x080001C5
  R15 (PC) 0x080001AA
  xPSR    0x01000000

Banked
System
Internal
  Mode      Thread
  Privilege Privileged
  Stack     MSP
  States    36
  Sec       0,00000300
FPU

Disassembly
0x080001A0 680A      LDR          r2,[r1,#0x00]
                      ldr r3, [r1, #4]
0x080001A2 684B      LDR          r3,[r1,#0x04]
                      ldr r4, [r1, #8]
                      10:
0x080001A4 688C      LDR          r4,[r1,#0x08]
                      ldr r6, =result
0x080001A6 4E05      LDR          r6,[pc,#20] ; @0x080001BC
                      str r0, [r6]
                      13:
0x080001A8 6030      STR          r0,[r6,#0x00]
                      push {r0}
                      15:
0x080001AA B401      PUSH         {r0}

Assembly
hw1.s
1 AREA .text!, CODE, READONLY, ALIGN=2
2
3 _main PROC
4   ldr sp, =0x20001000
5   ldr r0, =0x11223344
6   ldr r1, =first
7   ldr r2, [r1]
8   ldr r3, [r1, #4]
9   ldr r4, [r1, #8]
10
11   ldr r6, =result
12   str r0, [r6]
13
14   push {r0}

```

Qx11223344 (r₀)의 $\frac{dF}{dr}$ 이
0보다 작을 때 차량은 정지됨.

=) 토이의 솔루션 비아는 단위로 액수으로 저장되었다. 이는 높은

주도에 낮은 주도의 바이드 부터 시작하는 Little endian
방식을 사용하였음을 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)

Qx2000/099

The screenshot shows the QEMU debugger interface with three main panes:

- Registers** pane: Shows CPU registers. The R13 (SP) register is highlighted in blue and has a red circle around its value, 0x2000FFC. The R15 (PC) register is also highlighted in blue and has a red circle around its value, 0x080001AC.
- Disassembly** pane: Displays assembly instructions from memory address 0x080001A8 to 0x080001B8. The instruction at R15 (0x080001AC) is highlighted in yellow and is labeled "18: b .". Other instructions include STR, PUSH, DCW, and LDR.
- Assembly** pane: Shows the assembly code for the program. The file is named "hw1.s". The code defines a main function with the following assembly:

```

AREA .text!, CODE, READONLY, ALIGN=2
EXPORT __main
main PROC
    ldr sp, =0x20001000
    ldr r0, =0x11223344
    ldr r1, =first
    ldr r2, [r1]
    ldr r3, [r1, #4]
    ldr r4, [r1, #8]

    ldr r6, =result
    str r0, [r6]

    push {r0}

```

$$R_{13}(sp) = 0x2000 \text{ OFFC}$$

push를 하면 stack pointer가 4 bytes 뒤로 포진
little endian 방식으로 저장된다. 이 때 stack pointer는

Full descending 방식으로 책은 주소 값으로 늘어난다.

Stacker $0x2000\text{ off}$ ~ $0x2000\text{ fff}$ 메모리에

지강 되어 암습니다

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

The screenshot shows the JTAG-ICE debugger interface with three main tabs: Registers, Disassembly, and Source.

- Registers Tab:** Displays the state of various processor registers. The R15 (PC) register is highlighted in blue, showing its value as 0x080001AE. Other visible registers include R0 through R14, xPSR, and internal modes like Thread, Privileged, MSP, and Sec.
- Disassembly Tab:** Shows the assembly code corresponding to the memory address of the PC. The current instruction at address 0x080001AE is a B instruction, which branches to address 0x080001AE. Other instructions shown include STR, PUSH, DCW, and LDR.
- Source Tab:** Displays the C source code for the main function. The code initializes stack pointer (sp), registers r0 and r1, and performs a loop where r2 is loaded into r1, r3 is loaded into r2, and r4 is loaded into r3.

$Sp = 0x2000\ 0FF0$

push { r2 - rf } ∈

Stack pointer of all off

4 hours ~~간 7월 3~~ ~~된다.~~

2) stack의 자료형

1. 1) 일주소법의 이해

로의 소극적의 힘

이외에 r_2 가 작을 때
이와 같은 방식으로 r_3, r_4
도 standard에 저장된다.