

5-2. ARM기반 프로그램(2)

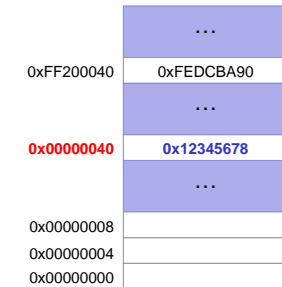
Load and Store Instructions

■ Reading data from memory

- How do we read the memory contents at address **0x40** into register R1

R0	0x40
R1	0x12345678

```
MOV R0, #0x40
LDR R1, [R0]
```



Load and Store Instructions

■ Reading data from memory

- How do we read the memory contents at address **0xFF200040** into R1

R0	0
R1	0

```
MOV R0, #0xFF200040
LDR R1, [R0]
```

- 0xFF200040 is larger than 16 bits



Load and Store Instructions

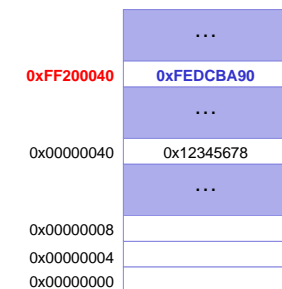
■ Reading data from memory

- How do we read the memory contents at address **0xFF200040** into R1

R0	0xFF200040
R1	0xFEDCBA90

```
MOV R0, #0x0040
MOVT R0, #0xFF20
LDR R1, [R0]
```

- MOVT 명령어 : $Rn[31:16] \leftarrow imm$ (ARMv6 이상)



Load and Store Instructions

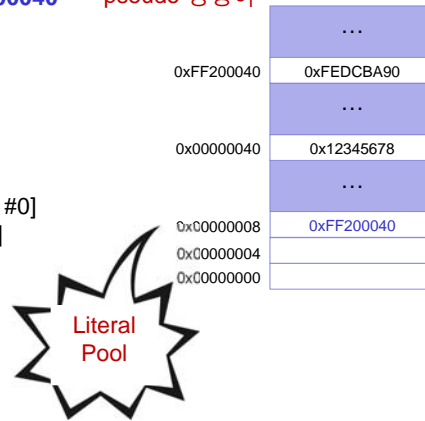
Reading data from memory

- How do we read the memory contents at address 0xFF200040 into R1

```
LDR R0, =0xFF200040 → pseudo 명령어
LDR R1, [R0]
```

Assembled code:

```
LDR R0, [pc, #0]
LDR R1, [R0]
```



5

Load and Store Instructions

Writing data to memory

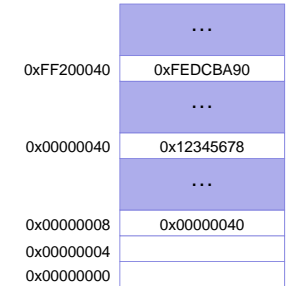
- How do we write the contents of R1 to the memory location 0x40

```
R0 0
R1 0
```

```
LDR R0, =0x00000040
STR R1, [R0]
```

Assembled code:

```
LDR R0, [pc, #0]
STR R1, [R0]
```



6

Addressing Modes

Register indirect:

- Register offset: [Rn]

Offset mode:

- Immediate offset: [Rn, #offset]
- Register offset: [Rn, ± Rm, shift]

Pre-indexed mode: (Auto indexed)

- Immediate offset: [Rn, #offset]!
- Register offset: [Rn, ± Rm, shift]!

Post-indexed mode:

- Immediate offset: [Rn], #offset
- Register offset: [Rn], ± Rm, shift

Index

[Rn+offset]

Pre-inc/dec

Rn ← Rn+offset, [Rn]

Post-inc/dec

[Rn], Rn ← Rn+offset

7

Storing and Loading Multiple Registers

Two pseudo-instructions

- PUSH and POP
- For storing and loading multiple registers

```
PUSH {R1, R3-R5} → STMDB SP!, {R1, R3-R5}
```

```
POP {R1, R3-R5} → LDMIA SP!, {R1, R3-R5}
```

- DB : Decrement Before
- IA : Increment After

8

Exercise 2: Dot product

$$\text{Dot product} = \sum_{i=0}^{n-1} A(i) \times B(i)$$

```
.text
.global _start
_start: LDR R0, =AVECTOR /* Register R0 is a pointer to vector A. */
        LDR R1, =BVECTOR /* Register R1 is a pointer to vector B. */
        LDR R2, N /* Register R2 is used as the counter for loop iterations. */
        MOV R3, #0 /* Register R3 is used to accumulate the product. */
LOOP:   LDR R4, [R0], #4 /* Load the next element of vector A. */
        LDR R5, [R1], #4 /* Load the next element of vector B. */
        MLA R3, R4, R5, R3 /* Compute the product of next pair of elements, */
        /* and add to the sum. */
        SUBS R2, R2, #1 /* Decrement the counter. */
        BGT LOOP /* Loop again if not finished. */
        STR R3, DOTP /* Store the result in memory. */
STOP:   B STOP

N:      .word 6 /* Specify the number of elements. */
AVECTOR: .word 5, 3, -6, 19, 8, 12 /* Specify the elements of vector A. */
BVECTOR: .word 2, 14, -3, 2, -5, 36 /* Specify the elements of vector B. */
DOTP:   .space 4 /* Space for the final dot product. */
.end
```

9

Exercise 2: Dot product

$$\text{Dot product} = \sum_{i=0}^{n-1} A(i) \times B(i)$$

```
.text
.global _start
_start: LDR R0, =AVECTOR /* Register R0 is a pointer to vector A. */
        LDR R1, =BVECTOR /* Register R1 is a pointer to vector B. */
        LDR R2, N /* Register R2 is used as the counter for loop iterations. */
        MOV R3, #0 /* Register R3 is used to accumulate the product. */
LOOP:   LDR R4, [R0], #4 /* Load the next element of vector A. */
        LDR R5, [R1], #4 /* Load the next element of vector B. */
        MLA R3, R4, R5, R3 /* Compute the product of next pair of elements, */
        /* and add to the sum. */
        SUBS R2, R2, #1 /* Decrement the counter. */
        BGT LOOP /* Loop again if not finished. */
        STR R3, DOTP /* Store the result in memory. */
STOP:   B STOP

N:      .word 6 /* Specify the number of elements. */
AVECTOR: .word 5, 3, -6, 19, 8, 12 /* Specify the elements of vector A. */
BVECTOR: .word 2, 14, -3, 2, -5, 36 /* Specify the elements of vector B. */
DOTP:   .space 4 /* Space for the final dot product. */
.end
```

10

Exercise 2: Dot product

$$\text{Dot product} = \sum_{i=0}^{n-1} A(i) \times B(i)$$

```
.text
.global _start
_start: LDR R0, =AVECTOR /* Register R0 is a pointer to vector A. */
        LDR R1, =BVECTOR /* Register R1 is a pointer to vector B. */
        LDR R2, N /* Register R2 is used as the counter for loop iterations. */
        MOV R3, #0 /* Register R3 is used to accumulate the product. */
LOOP:   LDR R4, [R0], #4 /* Load the next element of vector A. */
        LDR R5, [R1], #4 /* Load the next element of vector B. */
        MLA R3, R4, R5, R3 /* Compute the product of next pair of elements, */
        /* and add to the sum. */
        SUBS R2, R2, #1 /* Decrement the counter. */
        BGT LOOP /* Loop again if not finished. */
        STR R3, DOTP /* Store the result in memory. */
STOP:   B STOP

N:      .word 6 /* Specify the number of elements. */
AVECTOR: .word 5, 3, -6, 19, 8, 12 /* Specify the elements of vector A. */
BVECTOR: .word 2, 14, -3, 2, -5, 36 /* Specify the elements of vector B. */
DOTP:   .space 4 /* Space for the final dot product. */
.end
```

11

Exercise 2: Dot product

$$\text{Dot product} = \sum_{i=0}^{n-1} A(i) \times B(i)$$

```
.text
.global _start
_start: LDR R0, =AVECTOR /* Register R0 is a pointer to vector A. */
        LDR R1, =BVECTOR /* Register R1 is a pointer to vector B. */
        LDR R2, N /* Register R2 is used as the counter for loop iterations. */
        MOV R3, #0 /* Register R3 is used to accumulate the product. */
LOOP:   LDR R4, [R0], #4 /* Load the next element of vector A. */
        LDR R5, [R1], #4 /* Load the next element of vector B. */
        MLA R3, R4, R5, R3 /* Compute the product of next pair of elements, */
        /* and add to the sum. */
        SUBS R2, R2, #1 /* Decrement the counter. */
        BGT LOOP /* Loop again if not finished. */
        STR R3, DOTP /* Store the result in memory. */
STOP:   B STOP

N:      .word 6 /* Specify the number of elements. */
AVECTOR: .word 5, 3, -6, 19, 8, 12 /* Specify the elements of vector A. */
BVECTOR: .word 2, 14, -3, 2, -5, 36 /* Specify the elements of vector B. */
DOTP:   .space 4 /* Space for the final dot product. */
.end
```

12

Exercise 2: Dot product

$$\text{Dot product} = \sum_{i=0}^{n-1} A(i) \times B(i)$$

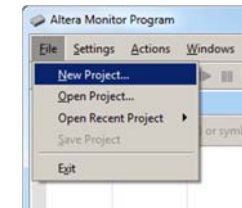
```
.text
.global _start
_start:
    LDR    R0, =AVECTOR    /* Register R0 is a pointer to vector A. */
    LDR    R1, =BVECTOR    /* Register R1 is a pointer to vector B. */
    LDR    R2, N            /* Register R2 is used as the counter for loop iterations. */
    MOV    R3, #0          /* Register R3 is used to accumulate the product. */
LOOP:
    LDR    R4, [R0], #4     /* Load the next element of vector A. */
    LDR    R5, [R1], #4     /* Load the next element of vector B. */
    MLA    R3, R4, R5, R3   /* Compute the product of next pair of elements,
                             /* and add to the sum. */
    SUBS   R2, R2, #1       /* Decrement the counter. */
    BGT    LOOP            /* Loop again if not finished. */
STR:
    STR    R3, DOTP        /* Store the result in memory. */
STOP:
    B      STOP

N:        .word    6        /* Specify the number of elements. */
AVECTOR:  .word    5, 3, -6, 19, 8, 12 /* Specify the elements of vector A. */
BVECTOR:  .word    2, 14, -3, 2, -5, 36 /* Specify the elements of vector B. */
DOTP:     .space   4        /* Space for the final dot product. */
.end
```

13

Step 1, 2: Create a New Project/Compile-Load

- Create a new project
 - New Project
 - Specify a project name & directory
 - Specify a system
 - ARM Cortex-A9
 - Specify a program type
 - Assembly Program
 - Specify program details
 - Add program files (.s)
 - Specify system parameters
 - Specify program memory settings
- Compile and Load



14

Step 3: Notice the use of the Literal Pool

The screenshot shows the Disassembly window with the following instruction highlighted:

```
0x00000000  E59F005C  ldr r0, [pc, #92] : 64 <DOTP+0x4>
```

The Registers window shows the current state of registers, and the Terminal window shows the program trace output.

15

Step 4: Go to the Memory Window

The screenshot shows the Memory window with the following memory contents:

```
0x00000000  E59F005C  E59F105C  E59F201C  E3A03000
0x00000010  E4904004  E4915004  E0233594  E2522001
0x00000020  CAFFFFFFA  E58F3034  EAFFFFFFE  00000006
0x00000030  00000005  00000003  FFFFFFFF  00000013
0x00000040  00000008  0000000C  00000002  0000000E
0x00000050  FFFFFFFD  00000002  FFFFFFFB  00000024
0x00000060  00000000  00000030  00000048  00000000
0x00000070  00000000  00000000  00000080  00000000
0x00000080  01000000  41801010  4D202402  A8B4000C
0x00000090  2298000C  A0032004  21140001  A0A42000
0x000000A0  027F5000  21142004  221A0001  00601003
0x000000B0  08002003  10810001  E0040002  C1080008
```

The Registers window shows the current state of registers, and the Terminal window shows the program trace output.

16

Step 5: Find the Address of AVECTOR

Altera Monitor Program - dot_product : dot_product.srec [Paused]

Memory

Address	Value
0x00000000	E59F005C E59F105C E59F201C E3A03000
0x00000010	E4904004 E4915004 E0233594 E2522001
0x00000020	CAFFFFFFA E58F3034 EAFFFFFFE 00000006
0x00000030	00000005 00000003 FFFFFFFFA 00000013
0x00000040	00000008 0000000C 00000002 0000000E
0x00000050	FFFFFFFD 00000002 FFFFFFFFB 00000024
0x00000060	00000000 00000030 00000048 00000000
0x00000070	00000000 00000000 00000080 00000000
0x00000080	01000000 41801010 4B202402 A8B4000C
0x00000090	2298000C A0032004 21140001 A0A42000
0x000000A0	02F75000 21142004 221A0001 00601003
0x000000B0	08002003 10810001 E0040002 C1080008

Registers

Reg	Value
pc	0x00000000
r0	0xFFFFF8104
r1	0x000000011
r2	0x49535756
r3	0xFFFFED408
r4	0xFFFFF80E4
r5	0x010000040
r6	0x000000076
r7	0xFFFFF014
r8	0xFFFFF74C8
r9	0x000000076
r10	0xFFD02000
r11	0xFFD02000
r12	0x000000007
sp	0xFFFFFA500
lr	0x000000000
cpair	0x600001D3

Terminal

```
INFO: Non-memory - hpa_gmscv (0x1f700000 - 0x1f701fff)
INFO: Non-memory - hpa_gmac1 (0x1f702000 - 0x1f703fff)
INFO: Non-memory - hpa_axi_ocram (0xfffff0000 - 0xfffff)
INFO: Non-memory - hpa_timer (0xfffec600 - 0xfffec6ff)
INFO: Program Trace is not supported for the Arm A9.
```

17

Step 6: Find the Address of BVECTOR

Altera Monitor Program - dot_product : dot_product.srec [Paused]

Memory

Address	Value
0x00000000	E59F005C E59F105C E59F201C E3A03000
0x00000010	E4904004 E4915004 E0233594 E2522001
0x00000020	CAFFFFFFA E58F3034 EAFFFFFFE 00000006
0x00000030	00000005 00000003 FFFFFFFFA 00000013
0x00000040	00000008 0000000C 00000002 0000000E
0x00000050	FFFFFFFD 00000002 FFFFFFFFB 00000024
0x00000060	00000000 00000030 00000048 00000000
0x00000070	00000000 00000000 00000080 00000000
0x00000080	01000000 41801010 4B202402 A8B4000C
0x00000090	2298000C A0032004 21140001 A0A42000
0x000000A0	02F75000 21142004 221A0001 00601003
0x000000B0	08002003 10810001 E0040002 C1080008

Registers

Reg	Value
pc	0x00000000
r0	0xFFFFF8104
r1	0x000000011
r2	0x49535756
r3	0xFFFFED408
r4	0xFFFFF80E4
r5	0x010000040
r6	0x000000076
r7	0xFFFFF014
r8	0xFFFFF74C8
r9	0x000000076
r10	0xFFD02000
r11	0xFFD02000
r12	0x000000007
sp	0xFFFFFA500
lr	0x000000000
cpair	0x600001D3

Terminal

```
INFO: Non-memory - hpa_gmscv (0x1f700000 - 0x1f701fff)
INFO: Non-memory - hpa_gmac1 (0x1f702000 - 0x1f703fff)
INFO: Non-memory - hpa_axi_ocram (0xfffff0000 - 0xfffff)
INFO: Non-memory - hpa_timer (0xfffec600 - 0xfffec6ff)
INFO: Program Trace is not supported for the Arm A9.
```

18

Step 7: Find the Data of the Vectors

Altera Monitor Program - dot_product : dot_product.srec [Paused]

Memory

Address	Value
0x00000000	E59F005C E59F105C E59F201C E3A03000
0x00000010	E4904004 E4915004 E0233594 E2522001
0x00000020	CAFFFFFFA E58F3034 EAFFFFFFE 00000006
0x00000030	00000005 00000003 FFFFFFFFA 00000013
0x00000040	00000008 0000000C 00000002 0000000E
0x00000050	FFFFFFFD 00000002 FFFFFFFFB 00000024
0x00000060	00000000 00000030 00000048 00000000
0x00000070	00000000 00000000 00000080 00000000
0x00000080	01000000 41801010 4B202402 A8B4000C
0x00000090	2298000C A0032004 21140001 A0A42000
0x000000A0	02F75000 21142004 221A0001 00601003
0x000000B0	08002003 10810001 E0040002 C1080008

Registers

Reg	Value
pc	0x00000000
r0	0xFFFFF8104
r1	0x000000011
r2	0x49535756
r3	0xFFFFED408
r4	0xFFFFF80E4
r5	0x010000040
r6	0x000000076
r7	0xFFFFF014
r8	0xFFFFF74C8
r9	0x000000076
r10	0xFFD02000
r11	0xFFD02000
r12	0x000000007
sp	0xFFFFFA500
lr	0x000000000
cpair	0x600001D3

Terminal

```
INFO: Non-memory - hpa_gmscv (0x1f700000 - 0x1f701fff)
INFO: Non-memory - hpa_gmac1 (0x1f702000 - 0x1f703fff)
INFO: Non-memory - hpa_axi_ocram (0xfffff0000 - 0xfffff)
INFO: Non-memory - hpa_timer (0xfffec600 - 0xfffec6ff)
INFO: Program Trace is not supported for the Arm A9.
```

19

Step 8: Change to Signed Representation

Altera Monitor Program - dot_product : dot_product.srec [Paused]

Memory

Address	Value
0x00000000	E59F005C E59F105C E59F201C E3A03000
0x00000010	E4904004 E4915004 E0233594 E2522001
0x00000020	CAFFFFFFA E58F3034 EAFFFFFFE 00000006
0x00000030	00000005 00000003 FFFFFFFFA 00000013
0x00000040	00000008 0000000C 00000002 0000000E
0x00000050	FFFFFFFD 00000002 FFFFFFFFB 00000024
0x00000060	00000000 00000030 00000048 00000000
0x00000070	00000000 00000000 00000080 00000000
0x00000080	01000000 41801010 4B202402 A8B4000C
0x00000090	2298000C A0032004 21140001 A0A42000
0x000000A0	02F75000 21142004 221A0001 00601003
0x000000B0	08002003 10810001 E0040002 C1080008

Registers

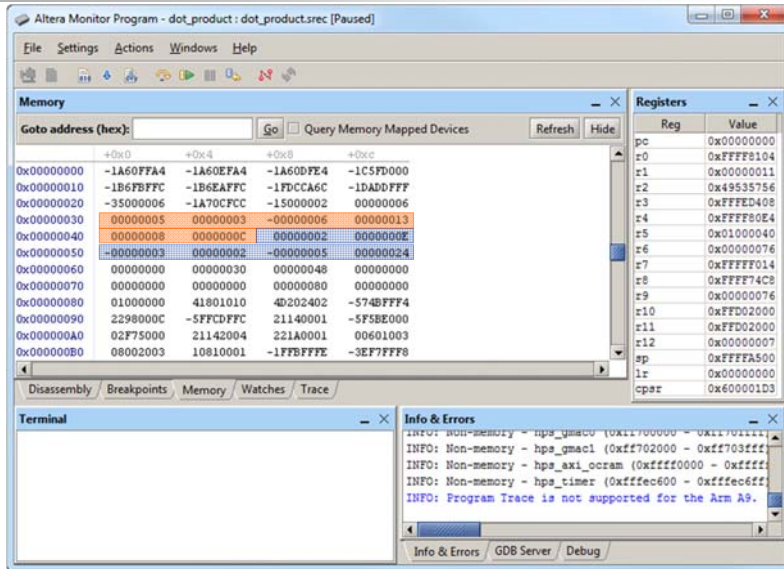
Reg	Value
pc	0x00000000
r0	0xFFFFF8104
r1	0x000000011
r2	0x49535756
r3	0xFFFFED408
r4	0xFFFFF80E4
r5	0x010000040
r6	0x000000076
r7	0xFFFFF014
r8	0xFFFFF74C8
r9	0x000000076
r10	0xFFD02000

Terminal

```
INFO: Non-memory - hpa_gmscv (0x1f700000 - 0x1f701fff)
INFO: Non-memory - hpa_gmac1 (0x1f702000 - 0x1f703fff)
INFO: Non-memory - hpa_axi_ocram (0xfffff0000 - 0xfffff)
INFO: Non-memory - hpa_timer (0xfffec600 - 0xfffec6ff)
INFO: Program Trace is not supported for the Arm A9.
```

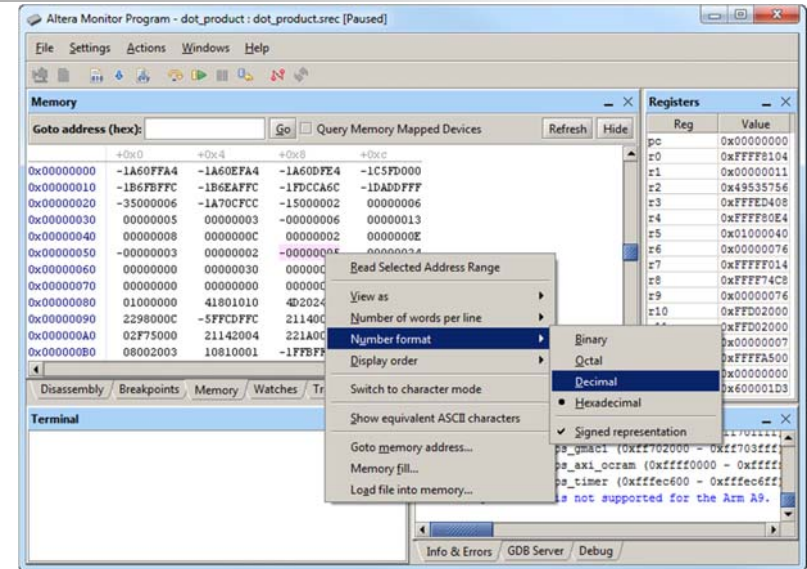
20

Step 8: Change to Signed Representation



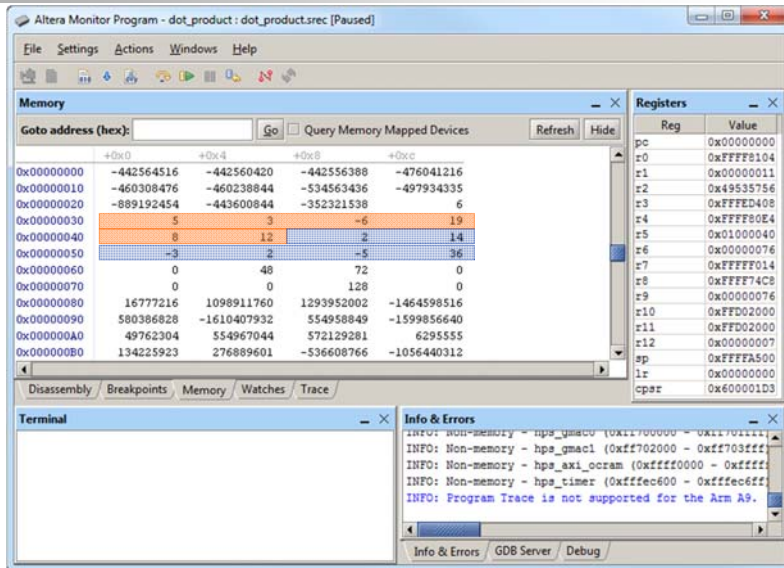
21

Step 9: Change to Decimal Format



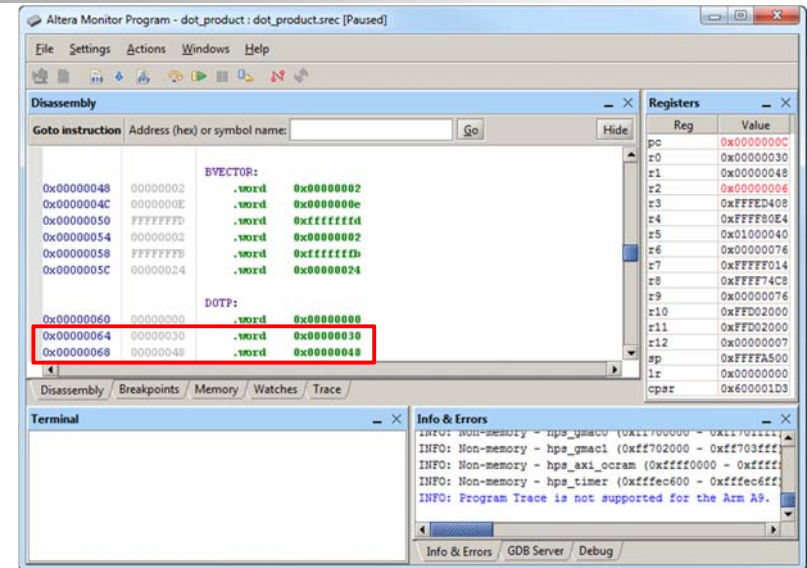
22

Step 9: Change to Decimal Format



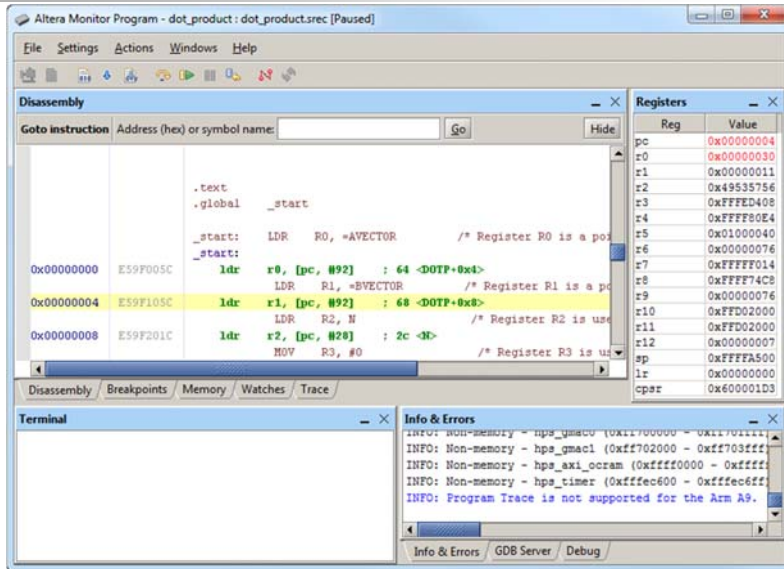
23

Step 10: Find Literal Pool in the Disassembly Window



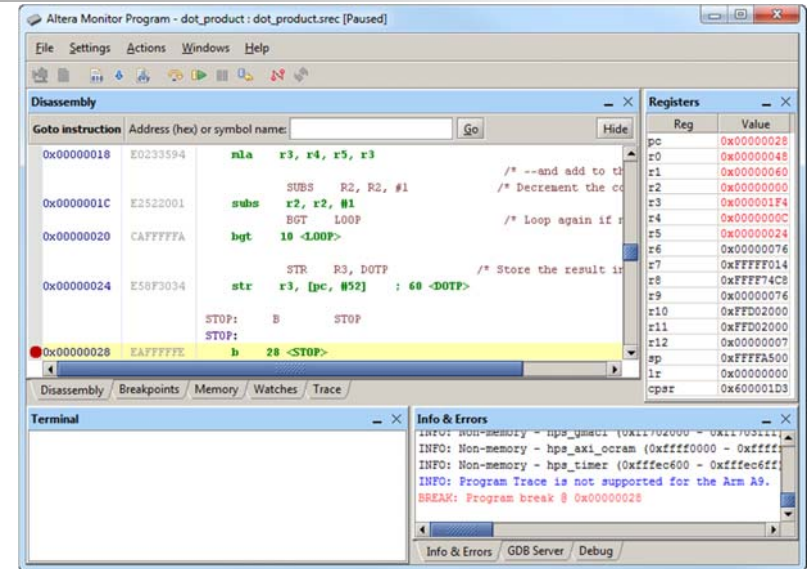
24

Step 11: Single Step or Set Breakpoints and Run



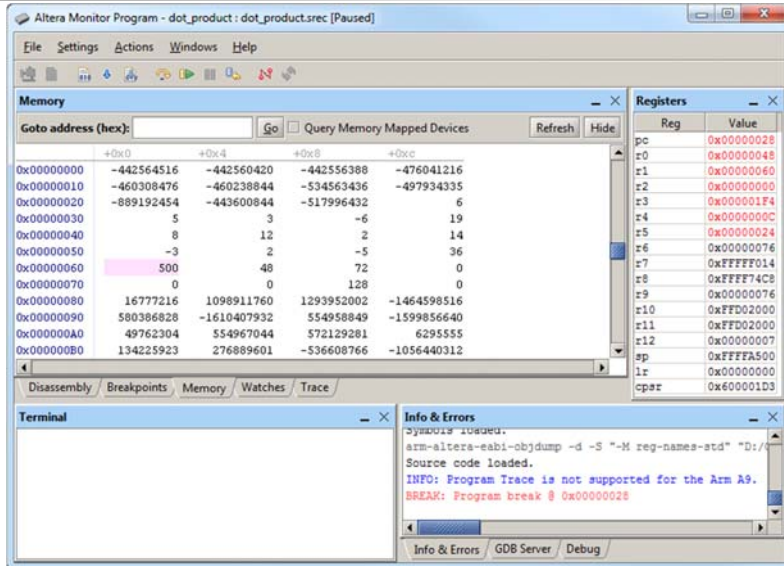
25

Step 12: Final Result in R3



26

Step 12: ... And in the Memory Location DOTP



27

Exercise: Memory Block Transfer

```

1  .text
2  .global _start
3  _start:
4      ldr    r0, =avector
5      ldr    r1, =bvector
6      ldr    r2, n
7  loop:
8      ldmia r0!, {r3-r10}
9      stmia r1!, {r3-r10}
10     subs  r2, r2, #8
11     bgt   loop
12 stop:
13     b     stop
14
15 n:    .word 32
16 avector: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
17         .word 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
18         .word 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32
19 bvector: .space 4*32
20 .end

```

Exercise: Subroutine Call

```
.text
.global _start
_start:
    LDR    R0, N
    BL    FINDSUM
END:     B     END

/* FINDSUM subroutine: calculates the sum of numbers from 1 to N
 * Parameters: N is in R0
 * Returns: sum in R0
 */
FINDSUM:
    MOVS   R1, R0
    MOV    R0, #0
SUM_LOOP:
    BLE   END_LOOP
    ADD   R0, R0, R1
    SUBS  R1, #1
    B     SUM_LOOP
END_LOOP:
    MOV   PC, LR

N:      .word 5
.end
```

Exercise: Recursive Subroutine Call

```
.text
.global _start
_start:
    LDR    R0, N
    BL    FINDSUM
END:     B     END

FINDSUM:
    PUSH  {R4, LR}           // save state
    MOVS  R4, R0             // save N in R4, and check for 0
    BLE  RETURN             // if N == 0, just return N
RECURSE:
    SUB   R0, R4, #1        // set R0 = N-1
    BL   FINDSUM
    ADD  R0, R4, R0        // R0 = N + FINDSUM(N-1)
RETURN:
    POP  {R4, PC}          // the return value is in R0

N:      .word 5

.end
```