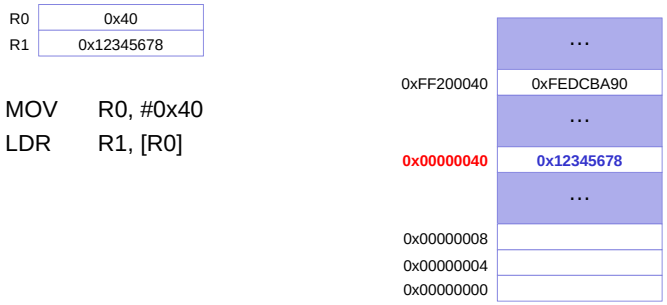
5-2. ARM기반 프로그램(2)

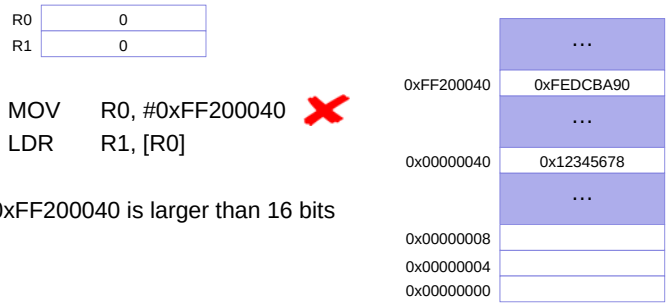
Load and Store Instructions

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



Load and Store Instructions

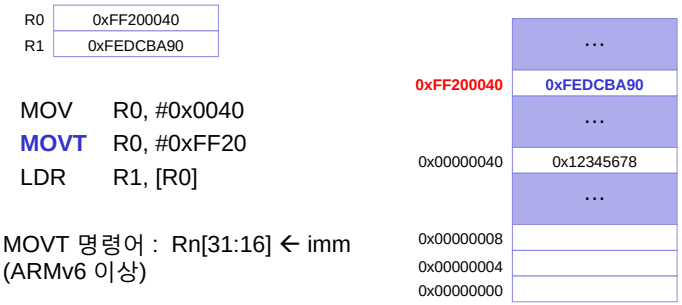
- Reading data from memory
  - How do we read the memory contents at address **0xFF200040** into R1



- 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



- MOVT 명령어 : Rn[31:16] ← imm (ARMv6 이상)

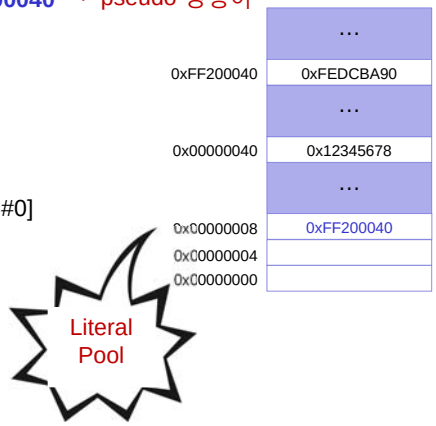
Load and Store Instructions

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

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

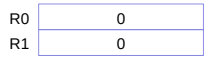
Assembled code:

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



Load and Store Instructions

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




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

Assembled code:

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

Addressing Modes

- Register indirect:
  - Register offset: **[Rn]**
- Offset mode: — Index
  - Immediate offset: **[Rn, #offset]** — [Rn+offset]
  - Register offset: [Rn, ± Rm, shift]
- Pre-indexed mode: (Auto indexed) — Pre-inc/dec
  - Immediate offset: **[Rn, #offset]!** — Rn←Rn+offset, [Rn]
  - Register offset: [Rn, ± Rm, shift]!
- Post-indexed mode: — Post-inc/dec
  - Immediate offset: **[Rn], #offset** — [Rn], Rn←Rn+offset
  - Register offset: [Rn], ± Rm, shift

Storing and Loading Multiple Registers

- Two pseudo-instructions
  - PUSH and POP
  - For storing and loading multiple registers

PUSH {R1, R3-R5} → STM**DB** SP!, {R1, R3-R5}

POP {R1, R3-R5} → LDM**IA** SP!, {R1, R3-R5}

  - DB : Decrement Before
  - IA : Increment After

Exercise 2: Dot product

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
```


Exercise 2: Dot product

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
```


Exercise 2: Dot product

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
```


Exercise 2: Dot product

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
```

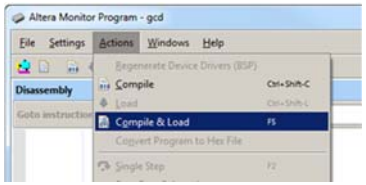
Exercise 2: Dot product

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

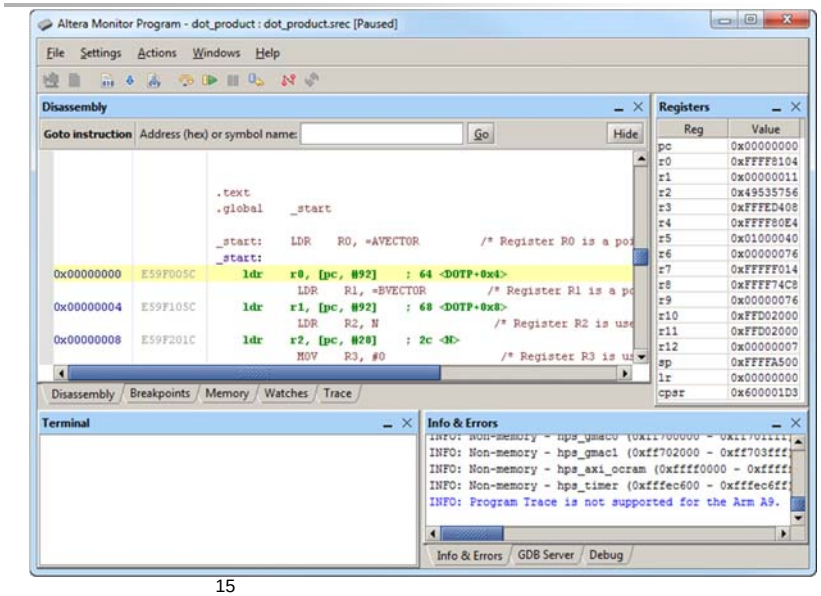
```
.text
.global     _start
_start:     LDR     R0, =AVECTOR        /* Register R0 is a pointer to vector A. */
            LDR     R1, =BVECTOR        /* Register R1 is a pointer to vector B. */
            LDR     R2, N               /* Register R2 is used as the counter for loop iterations. */
            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
```


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

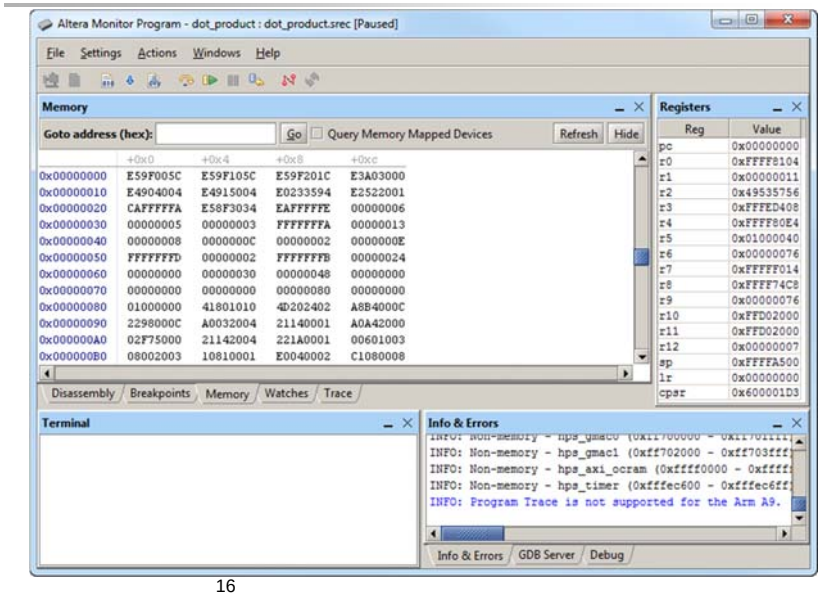
- Create a new project
  - New Project
  - Specify a project name & directory
  - Specify a system
    - ARM Cortex-A9
  - Specify a program type
    - Assembly Program
  - Specify program details
    - Add program files (.s)
  - Specify system parameters
  - Specify program memory settings
- Compile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



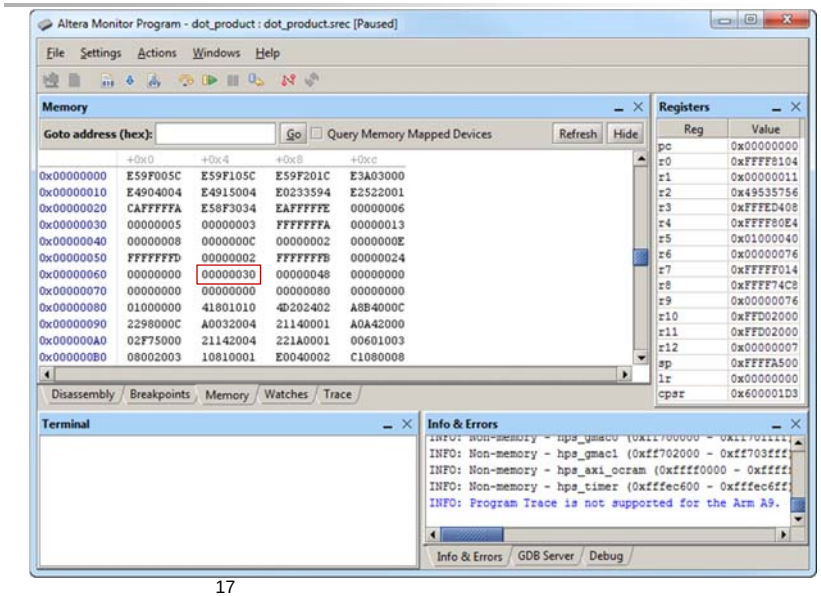
Step 3: Notice the use of the Literal Pool



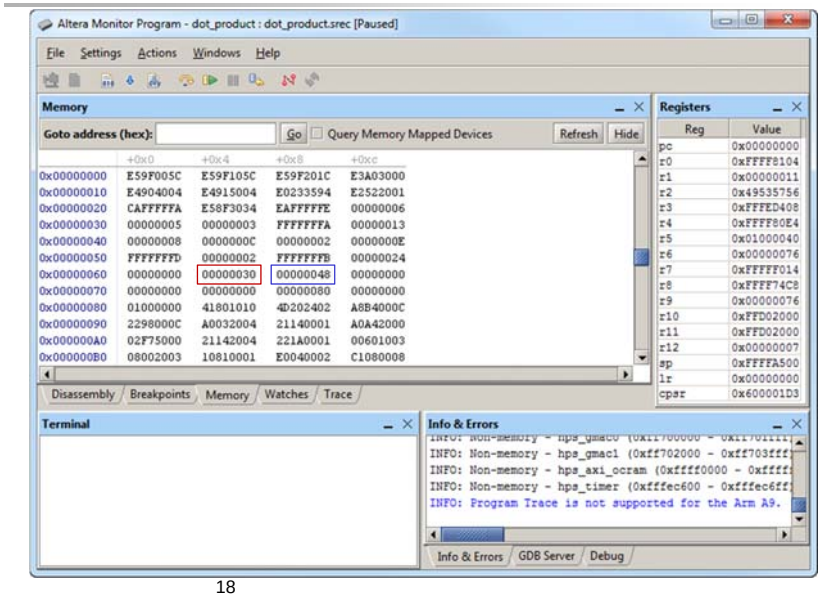
Step 4: Go to the Memory Window



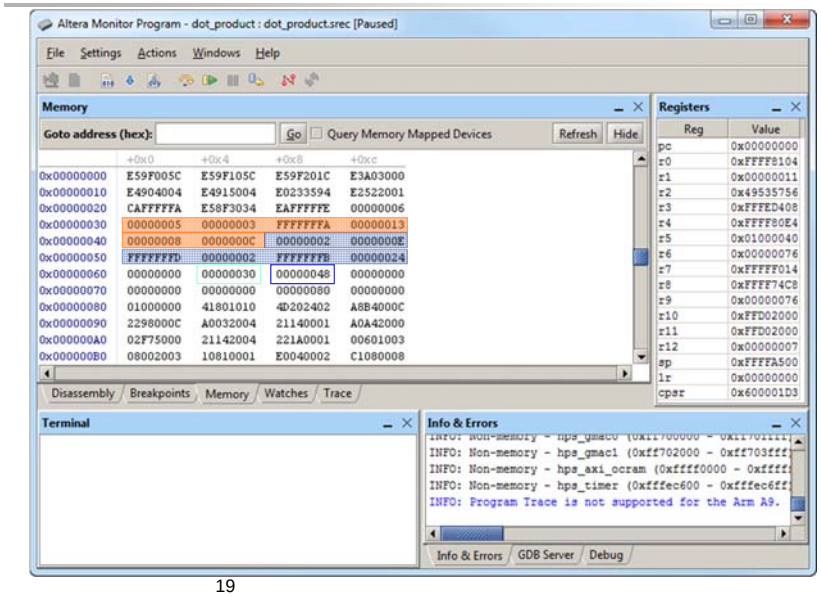
Step 5: Find the Address of AVECTOR



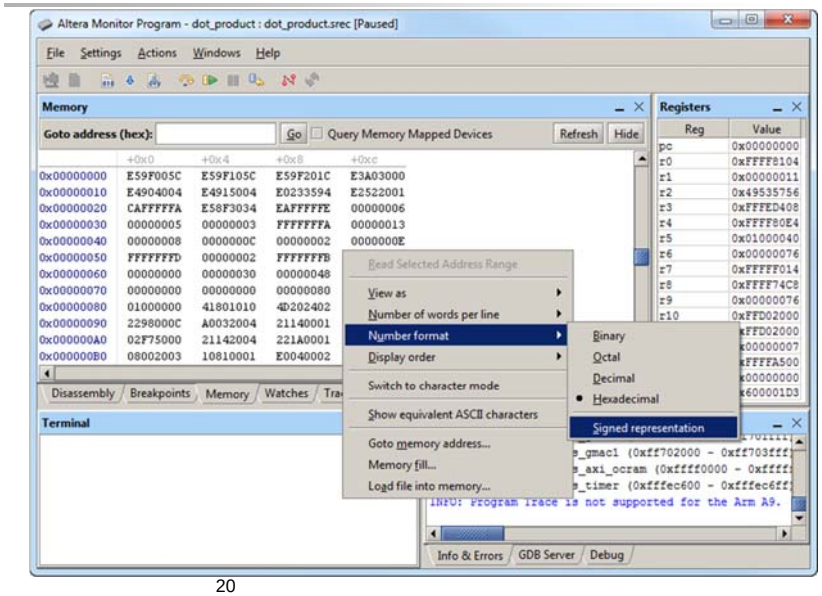
Step 6: Find the Address of BVECTOR



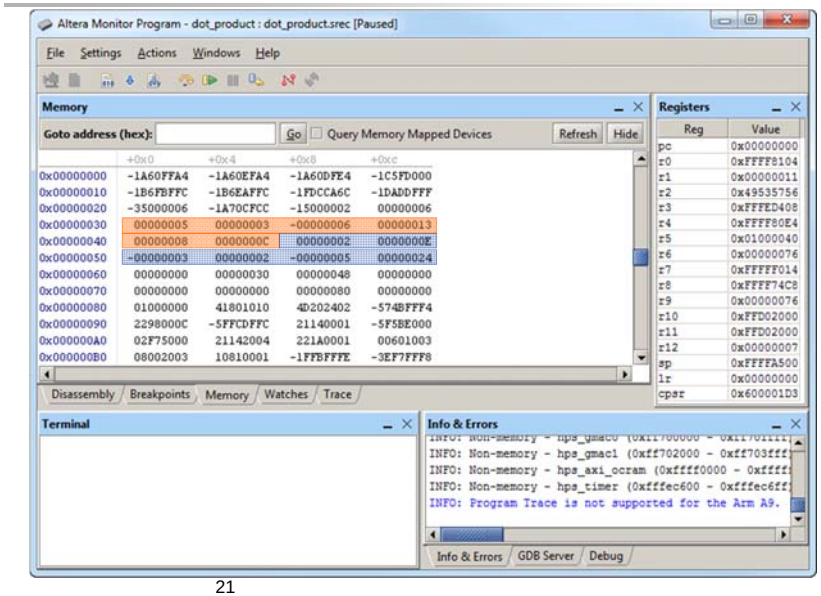
Step 7: Find the Data of the Vectors



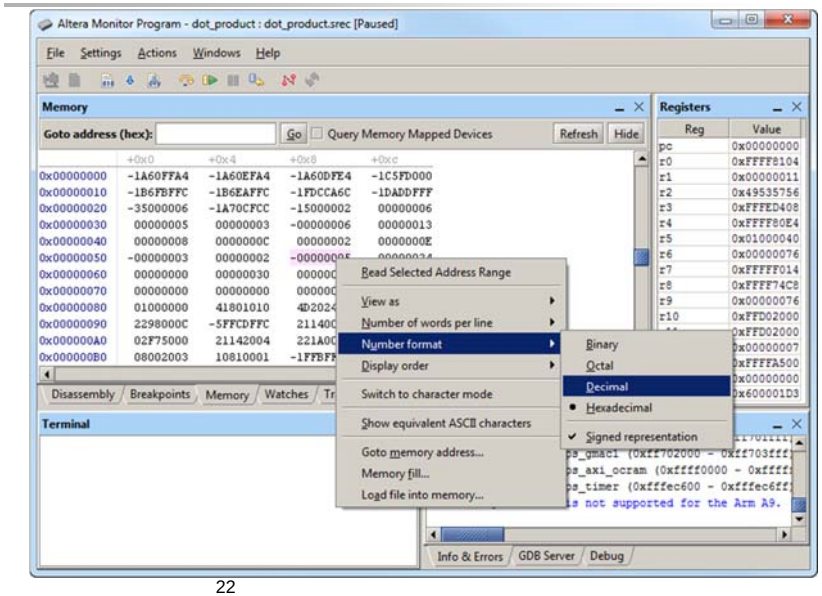
Step 8: Change to Signed Representation



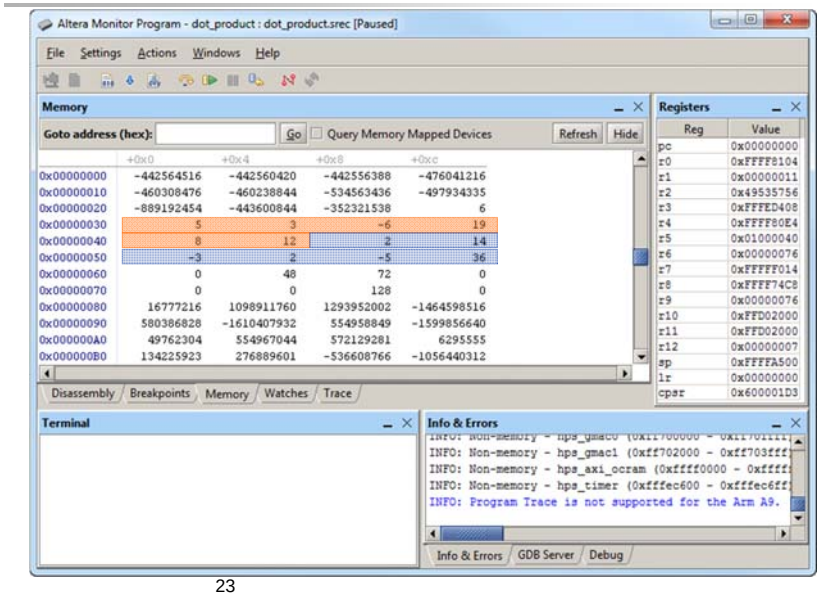
Step 8: Change to Signed Representation



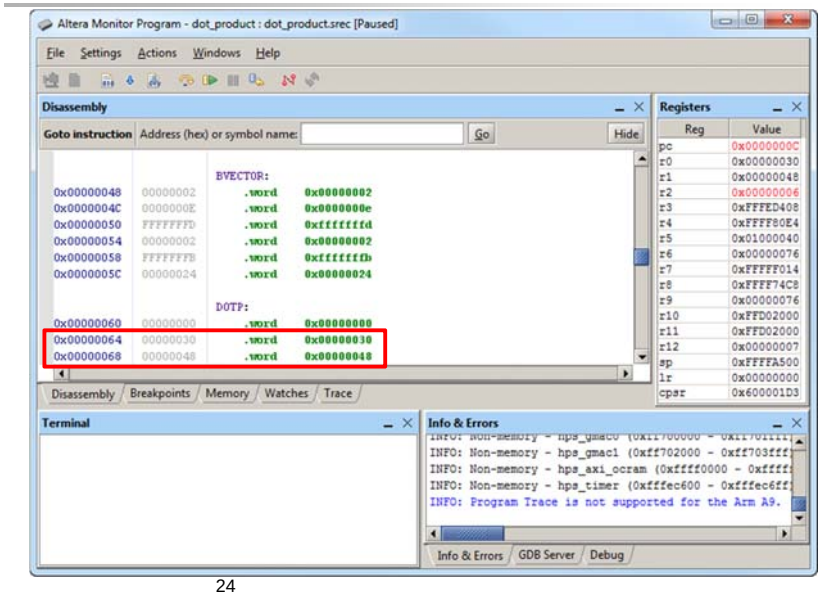
Step 9: Change to Decimal Format



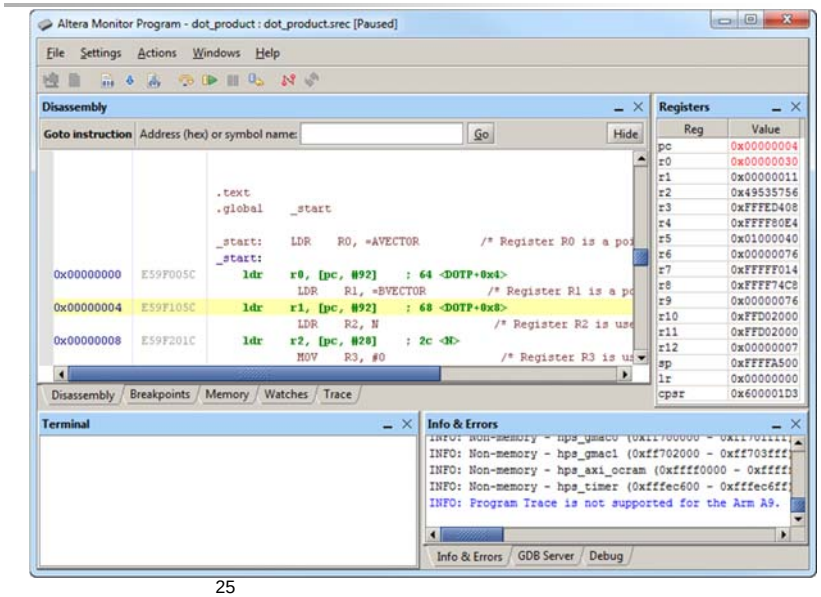
Step 9: Change to Decimal Format



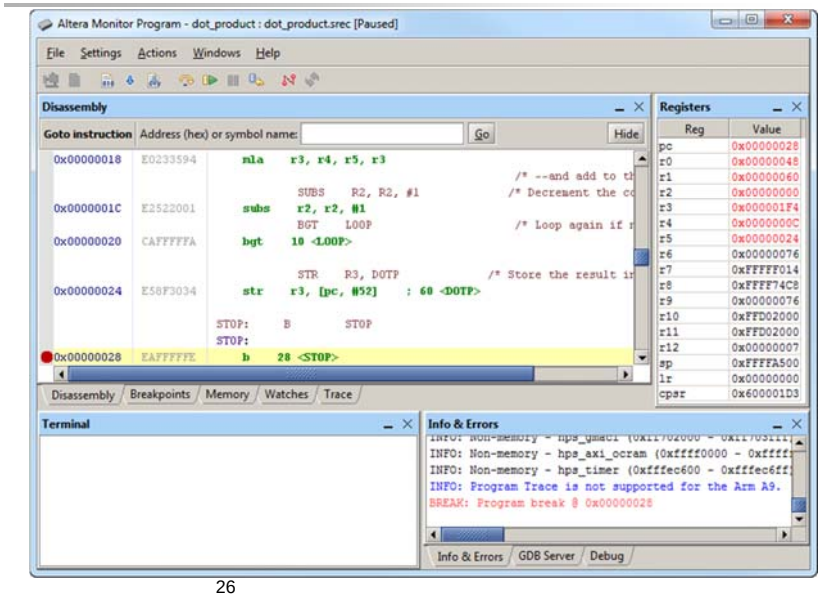
Step 10: Find Literal Pool in the Disassembly Window



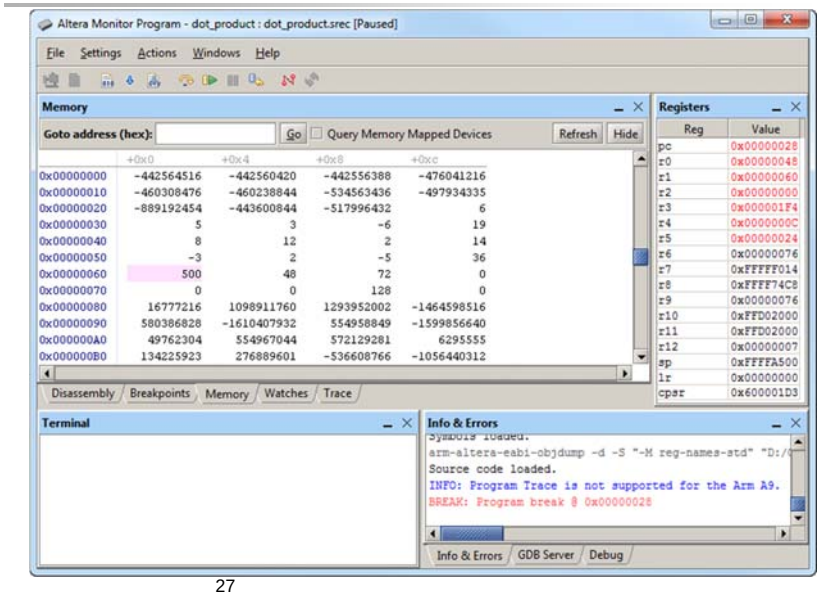
Step 11: Single Step or Set Breakpoints and Run



Step 12: Final Result in R3



Step 12: … And in the Memory Location DOTP



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
```