EE3002/IM2002

Microprocessors

Tutorial 7

1. Write an assembly program to construct a table of Fibonacci numbers for n = 0 to 20. The starting address of the table in memory is 0x40000000. In mathematics, the **Fibonacci numbers** or **Fibonacci series** or **Fibonacci sequence** are the numbers in the following integer sequence. Assume that the values are 32-bit data.

   0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 ….

   By definition, the first two numbers in the Fibonacci sequence are 0 and 1, and each subsequent number is the sum of the previous two. Hence, the sequence F_n of Fibonacci numbers is defined by the recurrence relation

$$F_n = F_{n-1} + F_{n-2}$$

   With seed values , $F_0 = 0$, $F_1 = 1$.

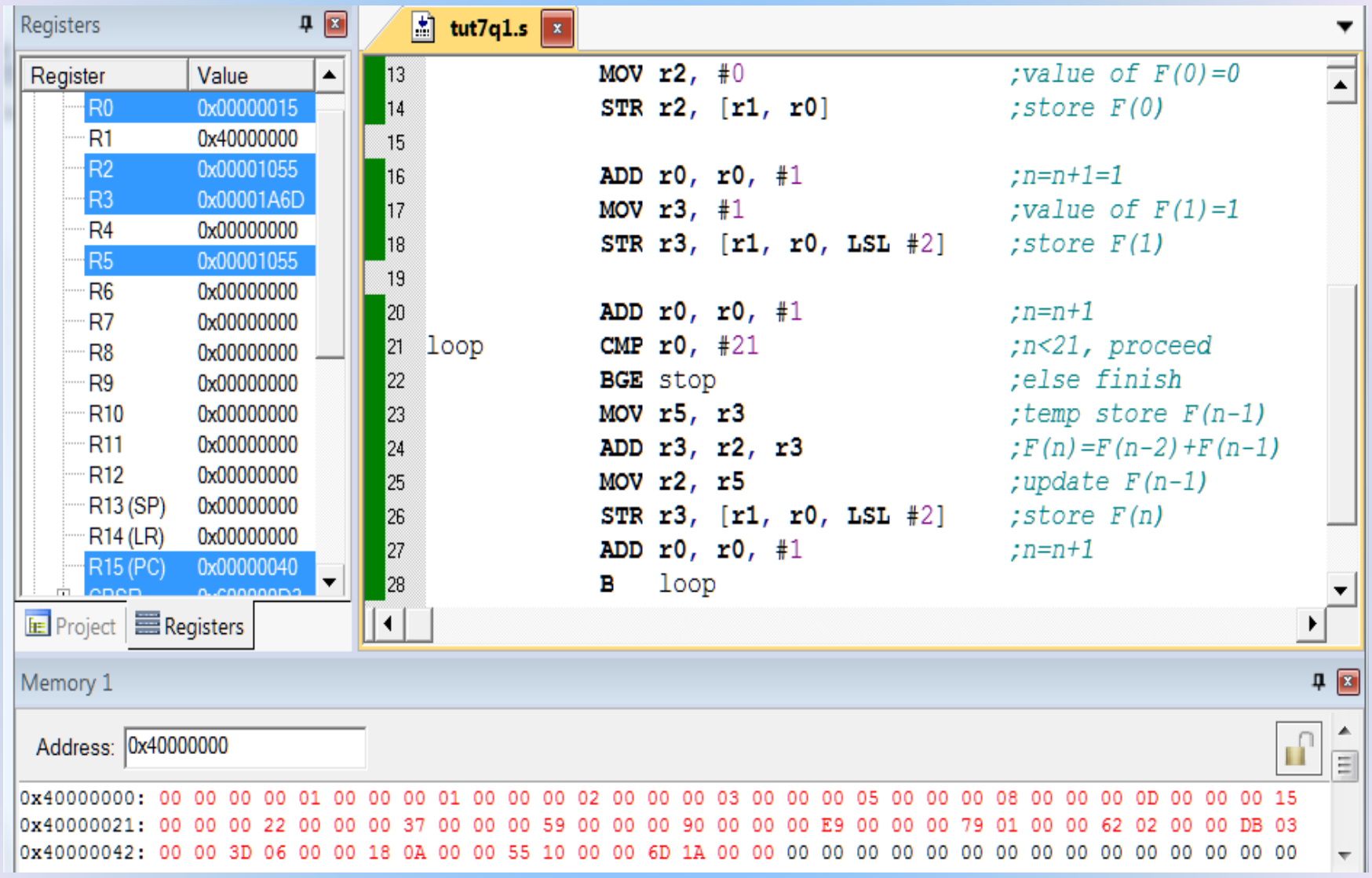
Q1:

```
                AREA FIBONACCI, CODE, READONLY
                ; Registers used:
                ; r0 = value of n
                ; r1 = starting address of table in memory
                ; r2 = temporary for Fibonacci value, F(n-2)
                ; r3 = temporary for Fibonacci value, F(n-1)
                ; r5 = temporary register
TABLE_BASE      EQU     0x40000000
                ENTRY
                MOV     r0, #0                  ;n = 0
                MOV     r1, #TABLE_BASE         ;base of table
                MOV     r2, #0                  ;value of F(0)
                STR     r2, [r1, r0]            ;store F(0)
                ADD     r0, r0, #1              ;n = n + 1 = 1
                MOV     r3, #1                  ;value of F(1)
                STR     r3, [r1, r0, LSL#2]     ;store F(1)
```

```
        ADD     r0, r0, #1              ;n = n + 1
loop    CMP     r0, #21                 ;n < 21, proceed
        BGE     stop                    ;else finish!

        MOV     r5, r3                  ;temp store F(n-1)
        ADD     r3, r2, r3              ;F(n)=F(n-2) + F(n-1)
        MOV     r2, r5                  ;update F(n-1)
        STR     r3, [r1, r0, LSL#2]     ;store F(n)
        ADD     r0, r0, #1              ;n = n + 1
        B       loop

stop    B       stop
        END
```


Registers
Register
Value
R0 0x00000015
R1 0x40000000
R2 0x00001055
R3 0x00001A6D
R4 0x00000000
R5 0x00001055
R6 0x00000000
R7 0x00000000
R8 0x00000000
R9 0x00000000
R10 0x00000000
R11 0x00000000
R12 0x00000000
R13 (SP) 0x00000000
R14 (LR) 0x00000000
R15 (PC) 0x00000040
Project
Registers
tut7q1.s
13 MOV r2, #0 ;value of F(0)=0
14 STR r2, [r1, r0] ;store F(0)
15
16 ADD r0, r0, #1 ;n=n+1=1
17 MOV r3, #1 ;value of F(1)=1
18 STR r3, [r1, r0, LSL #2] ;store F(1)
19
20 ADD r0, r0, #1 ;n=n+1
21 loop CMP r0, #21 ;n<21, proceed
22 BGE stop ;else finish
23 MOV r5, r3 ;temp store F(n-1)
24 ADD r3, r2, r3 ;F(n)=F(n-2)+F(n-1)
25 MOV r2, r5 ;update F(n-1)
26 STR r3, [r1, r0, LSL #2] ;store F(n)
27 ADD r0, r0, #1 ;n=n+1
28 B loop
Memory 1
Address: 0x40000000
0x40000000: 00 00 00 00 01 00 00 00 01 00 00 00 02 00 00 00 03 00 00 00 05 00 00 00 08 00 00 00 0D 00 00 00 15
0x40000021: 00 00 00 22 00 00 00 37 00 00 00 59 00 00 00 90 00 00 00 E9 00 00 00 79 01 00 00 62 02 00 00 DB 03
0x40000042: 00 00 3D 06 00 00 18 0A 00 00 55 10 00 00 6D 1A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

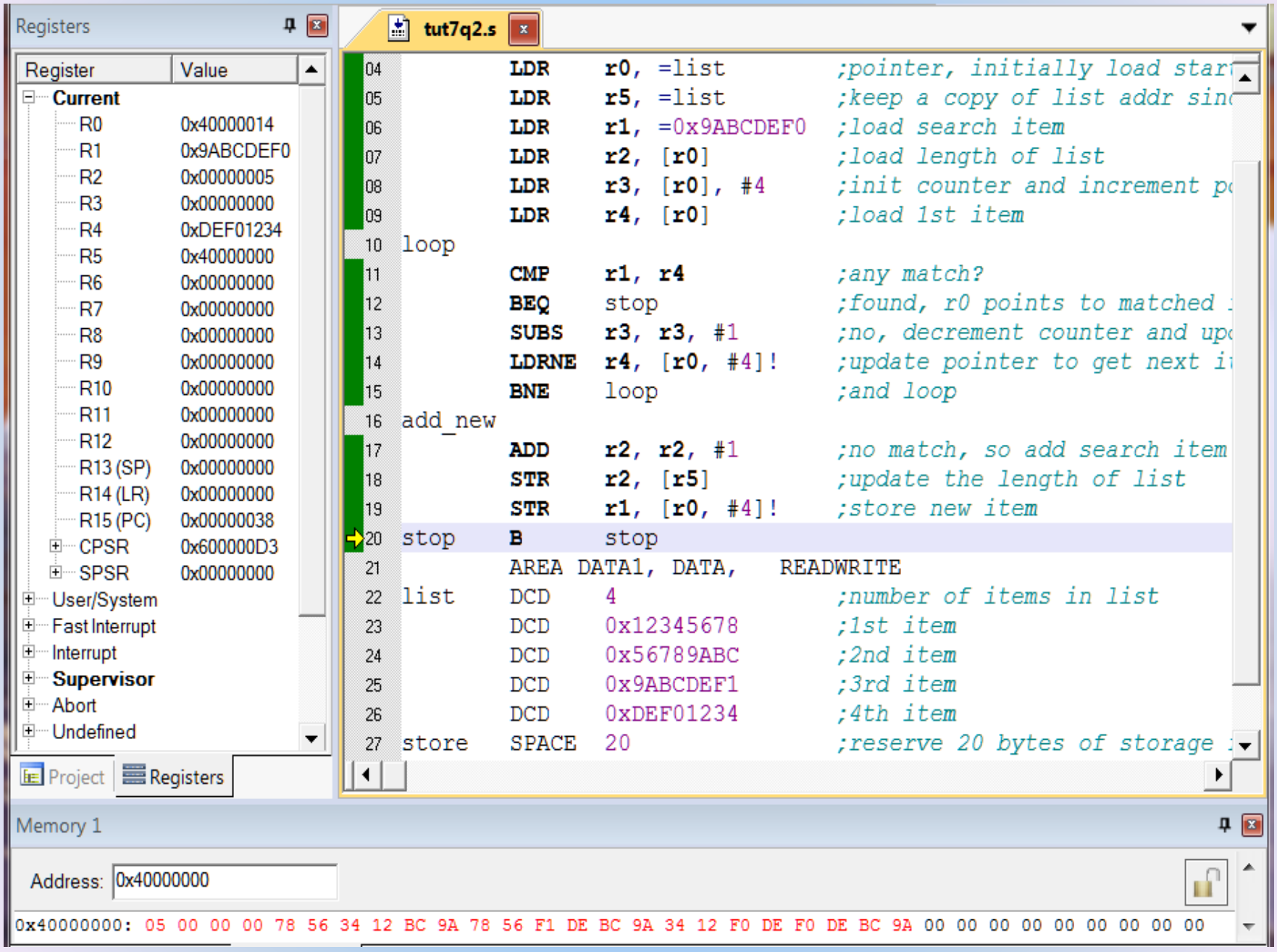
2. Write an assembly program to examine a table (a list of 32-bit values) sequentially for a match with a search key. Store the search key as a new entry if no match is found by adding it to the end of the list (highest address).

   Assume that the first value in the list indicates the length of the list and load it to register r2 (assume =4). Register r0 initially point to the starting address of the list. It will eventually point to the address of the matched item if there is a match. Register r1 contains the search item (assume =0x9ABCDEF0).

   You need to include the initialization file: tut7q2_L2104.ini .

```
AREA SearchTable, CODE, READONLY
        ENTRY
main
        LDR   r0, =list             ;pointer, initially load starting address of list
        LDR   r5, =list             ; keep a copy in case of updating the length
        LDR   r1, =0x9ABCDEF0       ;load search item
        LDR   r2, [r0]              ;load length of list. Keep it for updating
        LDR   r3, [r0], #4          ;init counter and increment pointer
        LDR   r4, [r0]              ;load 1st item
loop
        CMP   r1, r4                ;any match?
        BEQ   stop                  ;found, r0 points to matched item
        SUBS  r3, r3, #1            ;no, decrement counter and update the flags
        LDRNE r4, [r0, #4]!         ;update pointer to get next item, if counter<>0
        BNE   loop                  ;and loop
add_new
        ADD   r2, r2, #1            ;no match, increase the length of list by 1
        STR   r2, [r5]              ;update the length of list
        STR   r1, [r0, #4]!         ;add search item to the end of list
stop    B     stop
```

```
        AREA Data1, DATA, READWRITE
list    DCD   4                   ;no of items in the list
        DCD   0x12345678          ;1st item
        DCD   0x56789ABC          ;2nd item
        DCD   0x9ABCDEF0          ;3rd item
        DCD   0xDEF01234          ;4th item
store   SPACE       20            ;reserve 20 bytes of storage
        END
```


Registers
Register
Value
Current
R0 0x40000014
R1 0x9ABCDEF0
R2 0x00000005
R3 0x00000000
R4 0xDEF01234
R5 0x40000000
R6 0x00000000
R7 0x00000000
R8 0x00000000
R9 0x00000000
R10 0x00000000
R11 0x00000000
R12 0x00000000
R13 (SP) 0x00000000
R14 (LR) 0x00000000
R15 (PC) 0x00000038
CPSR 0x600000D3
SPSR 0x00000000
User/System
Fast Interrupt
Interrupt
Supervisor
Abort
Undefined
Project
Registers
tut7q2.s
04 LDR r0, =list ;pointer, initially load start
05 LDR r5, =list ;keep a copy of list addr sinc
06 LDR r1, =0x9ABCDEF0 ;load search item
07 LDR r2, [r0] ;load length of list
08 LDR r3, [r0], #4 ;init counter and increment po
09 LDR r4, [r0] ;load 1st item
10 loop
11 CMP r1, r4 ;any match?
12 BEQ stop ;found, r0 points to matched i
13 SUBS r3, r3, #1 ;no, decrement counter and upd
14 LDRNE r4, [r0, #4]! ;update pointer to get next it
15 BNE loop ;and loop
16 add_new
17 ADD r2, r2, #1 ;no match, so add search item
18 STR r2, [r5] ;update the length of list
19 STR r1, [r0, #4]! ;store new item
20 stop B stop
21 AREA DATA1, DATA, READWRITE
22 list DCD 4 ;number of items in list
23 DCD 0x12345678 ;1st item
24 DCD 0x56789ABC ;2nd item
25 DCD 0x9ABCDEF1 ;3rd item
26 DCD 0xDEF01234 ;4th item
27 store SPACE 20 ;reserve 20 bytes of storage
Memory 1
Address: 0x40000000
0x40000000: 05 00 00 00 78 56 34 12 BC 9A 78 56 F1 DE BC 9A 34 12 F0 DE F0 DE BC 9A 00 00 00 00 00 00 00 00 00

3) Write an assembly program, using a jump table that contains the addresses of 3 subroutines: DoAdd; DoSubtract; DoMultiply, to perform operation on two operands.

The operation to be performed is determined by register r0:
r0 = 0 for add;
r0 = 1 for subtract and;
r0 = 2 for multiply.

The two operands to these functions are contained in registers r1 and r2.

Test the program using the Keil uVision Simulator with different value of r0.

The three subroutines are shown below.

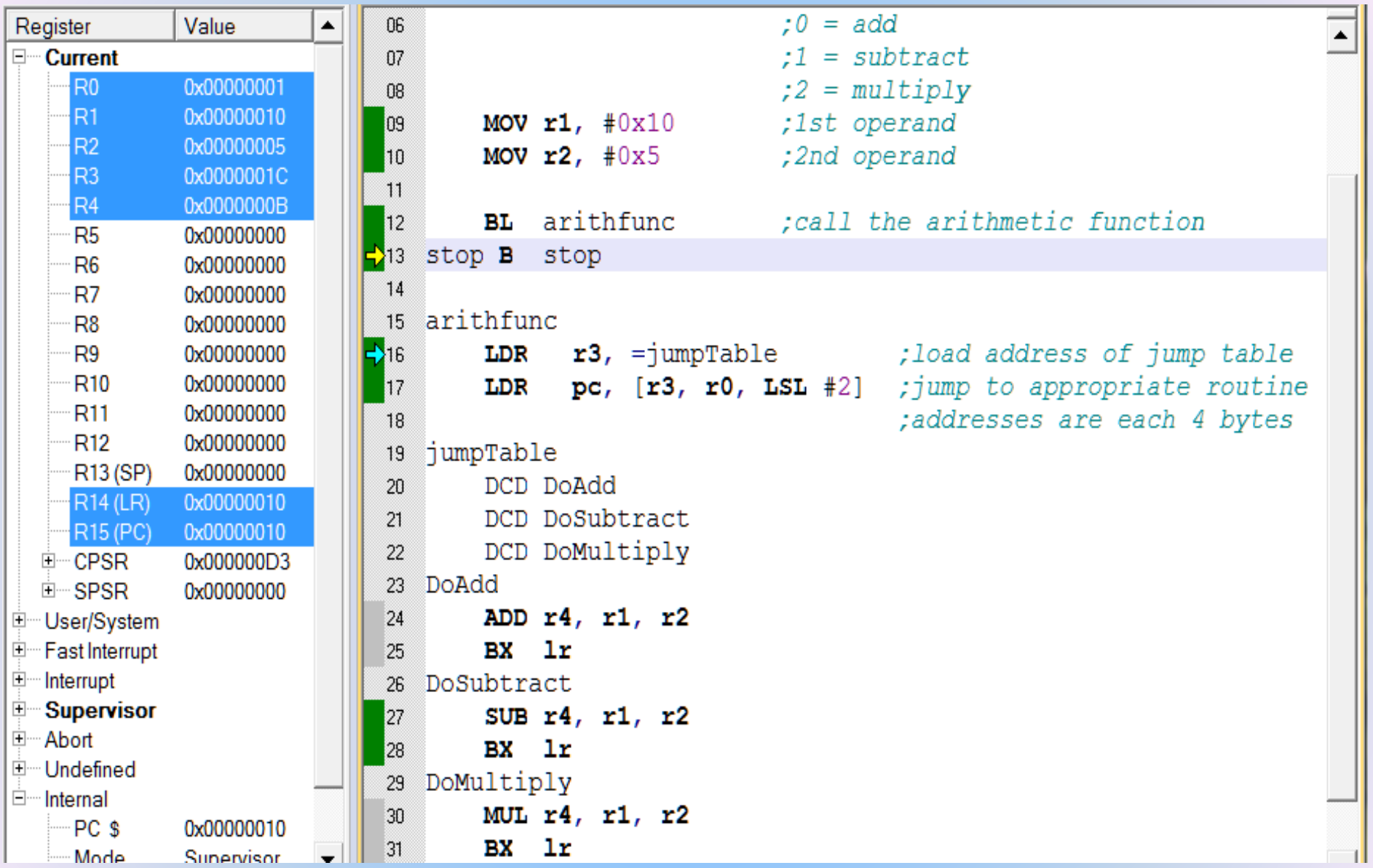
```
DoAdd
      ADD r4, r1, r2
      BX  lr            ;return from subroutine
DoSubtract
      SUB r4, r1, r2
      BX  lr
DoMultiply
      MUL r4, r1, r2
      BX  lr
```

Q3.

```
      AREA JmpTable, CODE, READONLY
      ENTRY
start
      MOV r0, #1  ;first parameter determines
                  ;function to be performed
                  ;0 = add
                  ;1 = subtract
                  ;2 = multiply
      MOV r1, #0x10      ;1st operand
      MOV r2, #0x5       ;2nd operand
      BL  arithfunc      ;call the arithmetic function
stop  B   stop
arithfunc
      LDR r3, =jumpTable       ;load address of jump table
      LDR pc, [r3, r0, LSL #2];jump to appropriate routine
                               ;addresses are 4 bytes apart
jumpTable
      DCD DoAdd
      DCD DoSubtract
      DCD DoMultiply
```

Q3.

```
DoAdd
                ADD r4, r1, r2
                BX  lr
DoSubtract
                SUB r4, r1, r2
                BX  lr
DoMultiply
                MUL r4, r1, r2
                BX  lr
END
```


Register
Value
Current
R0 0x00000001
R1 0x00000010
R2 0x00000005
R3 0x0000001C
R4 0x0000000B
R5 0x00000000
R6 0x00000000
R7 0x00000000
R8 0x00000000
R9 0x00000000
R10 0x00000000
R11 0x00000000
R12 0x00000000
R13 (SP) 0x00000000
R14 (LR) 0x00000010
R15 (PC) 0x00000010
CPSR 0x000000D3
SPSR 0x00000000
User/System
Fast Interrupt
Interrupt
Supervisor
Abort
Undefined
Internal
PC $ 0x00000010
Mode Supervisor
06 ;0 = add
07 ;1 = subtract
08 ;2 = multiply
09 MOV r1, #0x10 ;1st operand
10 MOV r2, #0x5 ;2nd operand
11
12 BL arithfunc ;call the arithmetic function
13 stop B stop
14
15 arithfunc
16 LDR r3, =jumpTable ;load address of jump table
17 LDR pc, [r3, r0, LSL #2] ;jump to appropriate routine
18 ;addresses are each 4 bytes
19 jumpTable
20 DCD DoAdd
21 DCD DoSubtract
22 DCD DoMultiply
23 DoAdd
24 ADD r4, r1, r2
25 BX lr
26 DoSubtract
27 SUB r4, r1, r2
28 BX lr
29 DoMultiply
30 MUL r4, r1, r2
31 BX lr

The end of Tutorial 7