EE3002/IM2002

Microprocessors

Tutorial 11

ARM and Thumb States

Tutorials Feedback

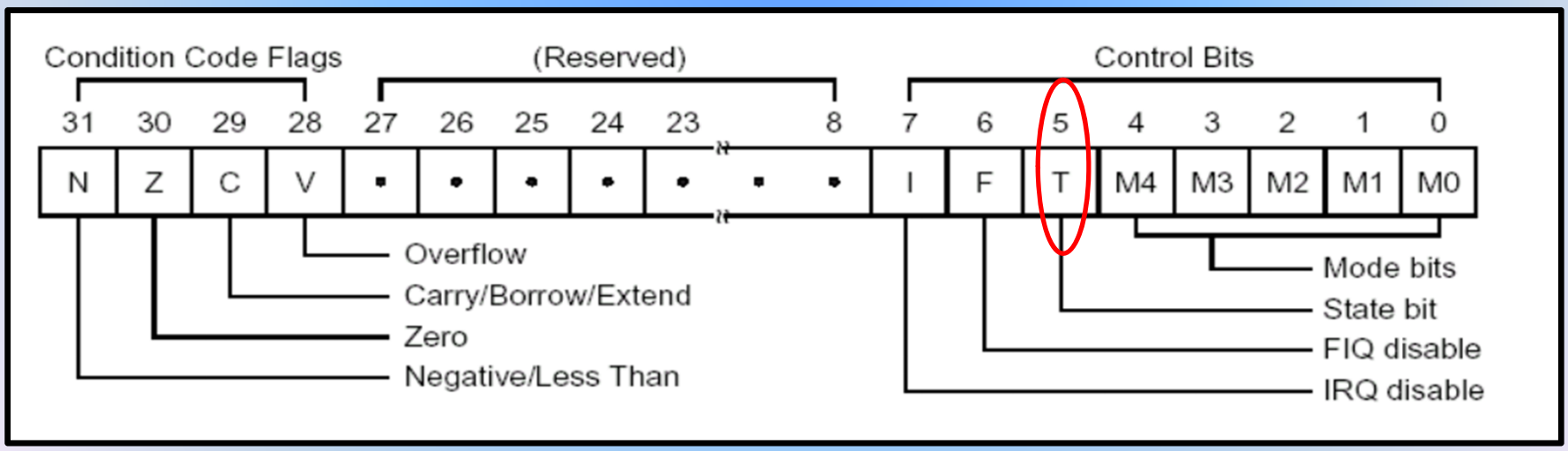
Tutorial Group	Date/Time
TA8	Thursday (9:30-10:30)
TA11	Friday (11:30-12:30)
TA12	Friday (16:30-17:30)

1. Which bit in the registers CPSR and SPSR indicate whether the processor is in ARM or THUMB state?

Ans:

Bit 5: 0 → ARM state

1 → THUMB state



2. Explain how the branch instruction *BX rd* is used to switch between the ARM and THUMB states.

Ans: ***BX rd***

- This instruction is a branch to the address stored in *rd* that can change processor state at runtime.
- The least significant bit (LSB) of the target address *rd* specifies the state that the processor is switching to :
  - ✓ If the LSB of *rd* = 0, it switches to ARM state
  - ✓ If the LSB of *rd* = 1, it switches to THUMB state

3. Write the THUMB instruction(s) that are equivalent to the following ARM instructions

a. ADDS r0, r3, r2, LSL #2
b. LDR r1, [r4, r5, LSL #2]

If the source registers CAN be modified

a) ADDS r0, r3, r2, LSL #2 replace by:

```
LSL   r2, #2
ADD   r3, r2
MOV r0, r3
```

OR

```
LSL  r2, #2
ADD  r0, r3, r2
```

b) LDR r1, [r4, r5, LSL #2] replace by:

```
LSL    r5, #2
LDR    r1, [r4, r5]
```

If the source registers are NOT to be modified

a) ADDS r0, r3, r2, LSL #2 replace by:

```
MOV r4, r2
LSL  r4, #2
ADD  r4, r3
MOV r0, r4
```

or

```
MOV r4, r2
LSL  r4, #2
ADD  r0, r4, r3
```

b) LDR r1, [r4, r5, LSL #2] replace by:

```
MOV  r6, r5
LSL   r6, #2
LDR   r1, [r4, r6]
```

4. The following program fragment is written in ARM code.
   - Determine what it is doing and compute the code size of this fragment.
   - Convert it into a THUMB program fragment and also determine its code size.

```
                ;IN:    r0 (value) , r1 (divisor)
                ;OUT: r2 (remainder), r3 (quotient)
                CODE32
                MOV     r0, #10
                MOV     r1, #3
                MOV     r3, #0
loop            SUBS    r0, r0, r1
                ADDGE   r3, r3, #1
                BGE     loop
                ADD     r2, r0, r1
```

The fragment divides r0 with r1, ie., r0/r1 = 10/3.

Division by using repeated subtractions

Register r2 = 1 (remainder), Register r3 = 3 (quotient)

The code size = 7*4 = **28 bytes**

```
                ;IN:    r0 (value) , r1 (divisor)
                ;OUT: r2 (remainder), r3 (quotient)
                CODE32
                MOV     r0, #10     ; input the number
                MOV     r1, #3      ; set the divisor
                MOV     r3, #0      ; initialise the quotient
loop            SUBS    r0, r0, r1  ; division by using subtraction
                ADDGE   r3, r3, #1  ; update quotient
                BGE     loop
                ADD     r2, r0, r1  ; adjust rem. due to last SUBS
                                    ; remainder cannot be negative
```

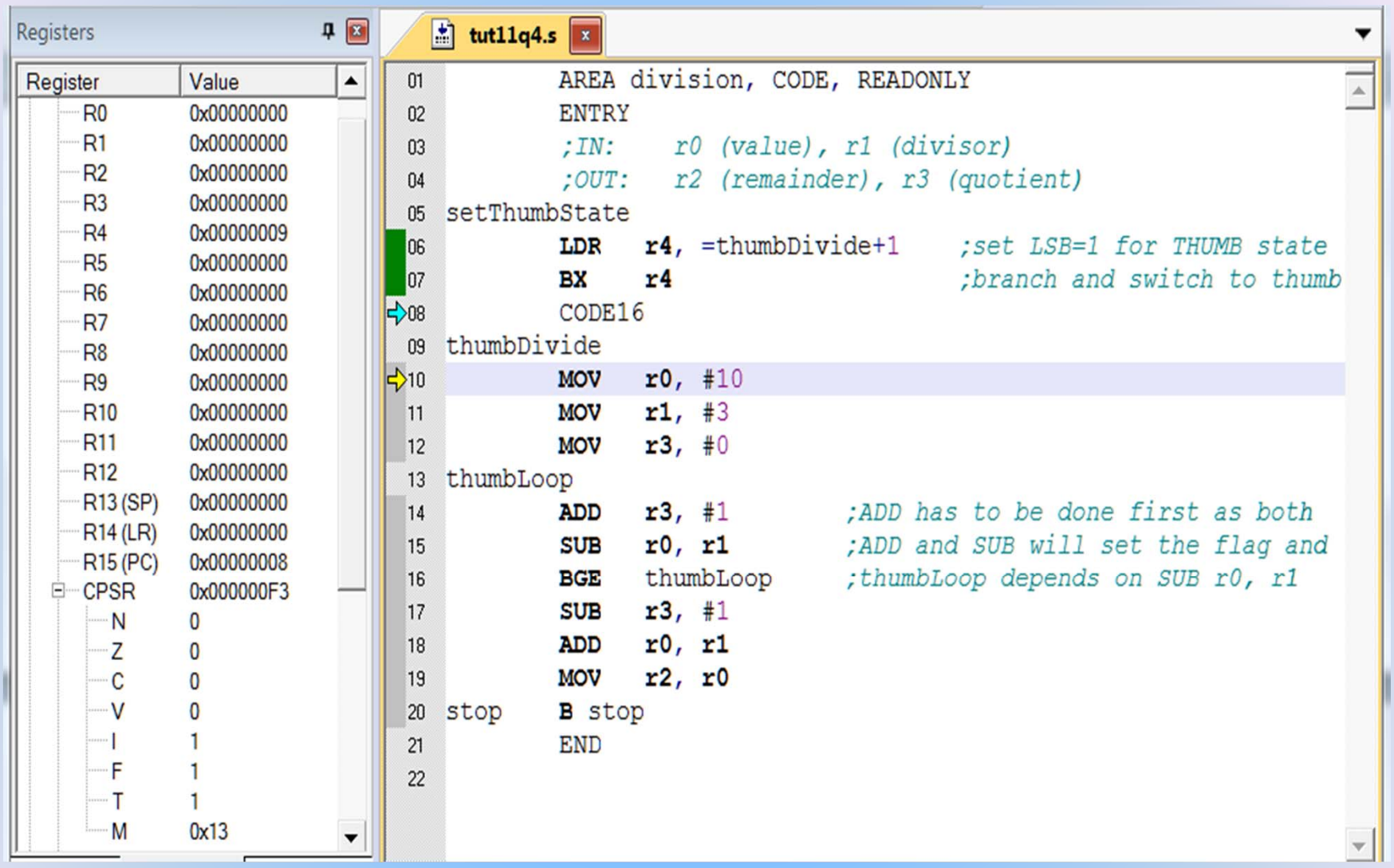
10
3

THUMB code:

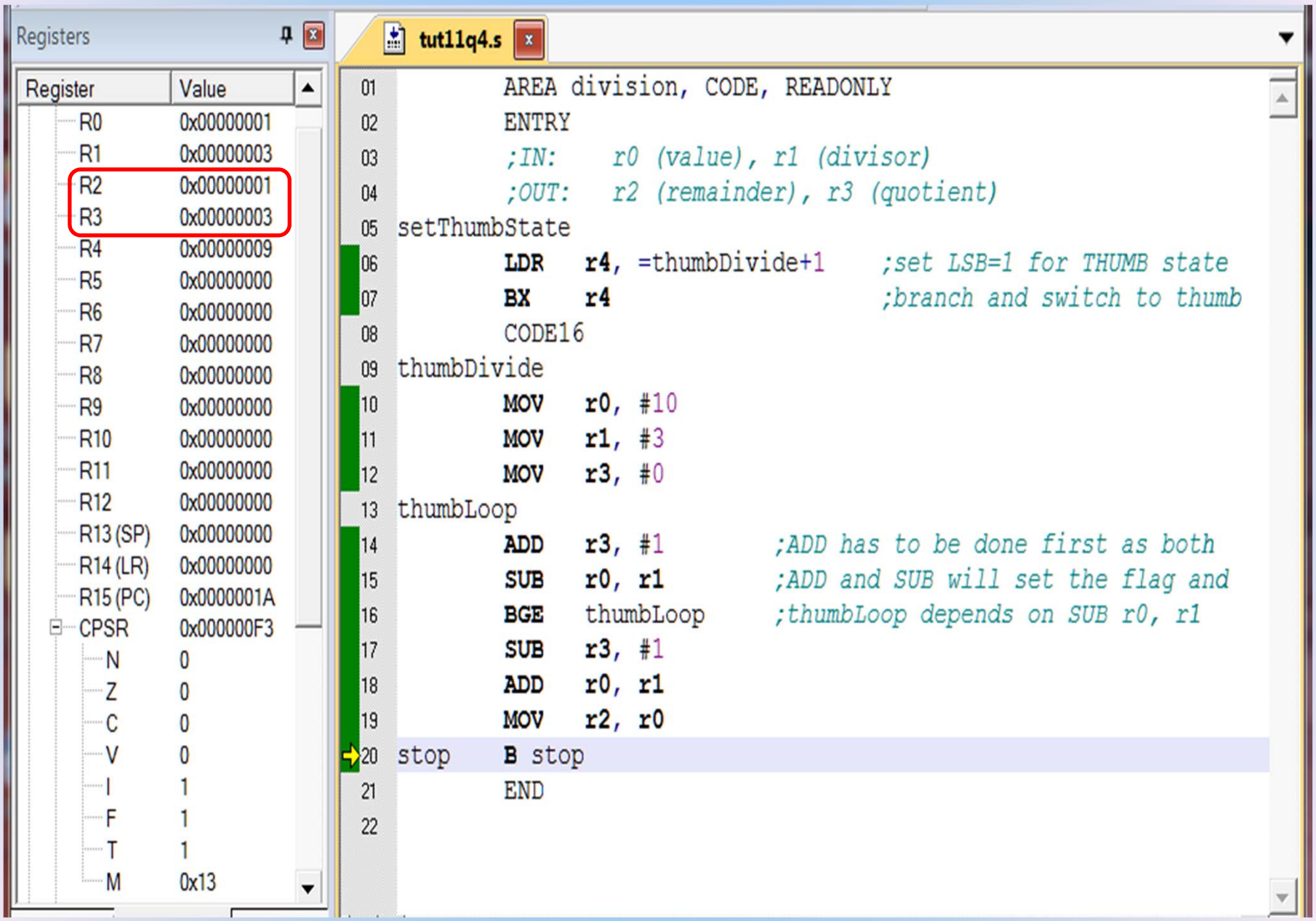
```
        ;IN:    r0 (value) , r1 (divisor)
        ;OUT: r2 (remainder), r3 (quotient)
        CODE16

        MOV   r0, #10
        MOV   r1, #3
        MOV   r3, #0          ; clear r3 , the quotient
Loop    ADD   r3, #1          ; ADD has to be done first as
        SUB   r0, r1          ; both ADD and SUB will set the flag
        BGE   loop            ; loop depends on SUB r0, r1
        SUB   r3, #1          ; offset the first add
        ADD   r0, r1          ; adjust the remainder due to SUB
        MOV   r2, r0          ; or using ADD r2, r0, r1
```

- Code size = 9 * 2 = 18 bytes



Note: switch to thumb state


Registers
Register
Value
R0 0x00000001
R1 0x00000003
R2 0x00000001
R3 0x00000003
R4 0x00000009
R5 0x00000000
R6 0x00000000
R7 0x00000000
R8 0x00000000
R9 0x00000000
R10 0x00000000
R11 0x00000000
R12 0x00000000
R13 (SP) 0x00000000
R14 (LR) 0x00000000
R15 (PC) 0x0000001A
CPSR 0x000000F3
N 0
Z 0
C 0
V 0
I 1
F 1
T 1
M 0x13
tut11q4.s
01 AREA division, CODE, READONLY
02 ENTRY
03 ;IN: r0 (value), r1 (divisor)
04 ;OUT: r2 (remainder), r3 (quotient)
05 setThumbState
06 LDR r4, =thumbDivide+1 ;set LSB=1 for THUMB state
07 BX r4 ;branch and switch to thumb
08 CODE16
09 thumbDivide
10 MOV r0, #10
11 MOV r1, #3
12 MOV r3, #0
13 thumbLoop
14 ADD r3, #1 ;ADD has to be done first as both
15 SUB r0, r1 ;ADD and SUB will set the flag and
16 BGE thumbLoop ;thumbLoop depends on SUB r0, r1
17 SUB r3, #1
18 ADD r0, r1
19 MOV r2, r0
20 stop B stop
21 END
22

5. **Write an assembly program that will call a subroutine (written in THUMB code) with register r0 = 5 and register r1 = 3.**

   The THUMB subroutine computes the sum of the squares of the values stored in register r0 and register r1. It should place the result in register r2.

$$r2 = r0^2 + r1^2 = 25 + 9 = 34 = 0x22$$

- **You should write code (veneer) that will enable the switch to different states**
- **Ensure that any other registers used in the subroutine are preserved upon entry and restored before return.**

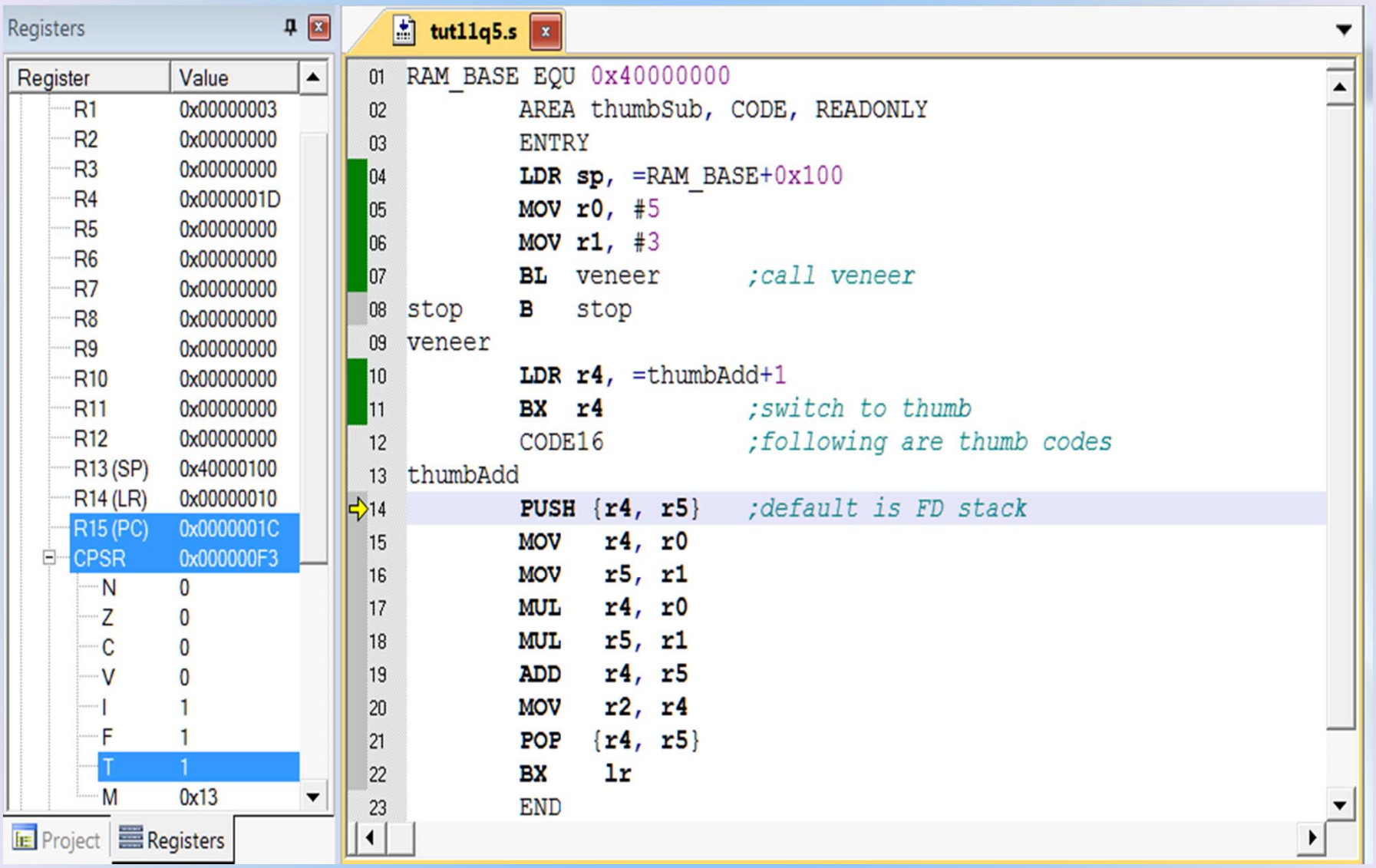
```
RAM_BASE  EQU   0x40000000
          AREA thumbSub, CODE, READONLY
          ENTRY
          CODE32

          LDR   sp, =RAM_BASE+0x100
          MOV  r0, #5
          MOV  r1, #3
          BL    veneer              ; call veneer
stop      B     stop
```

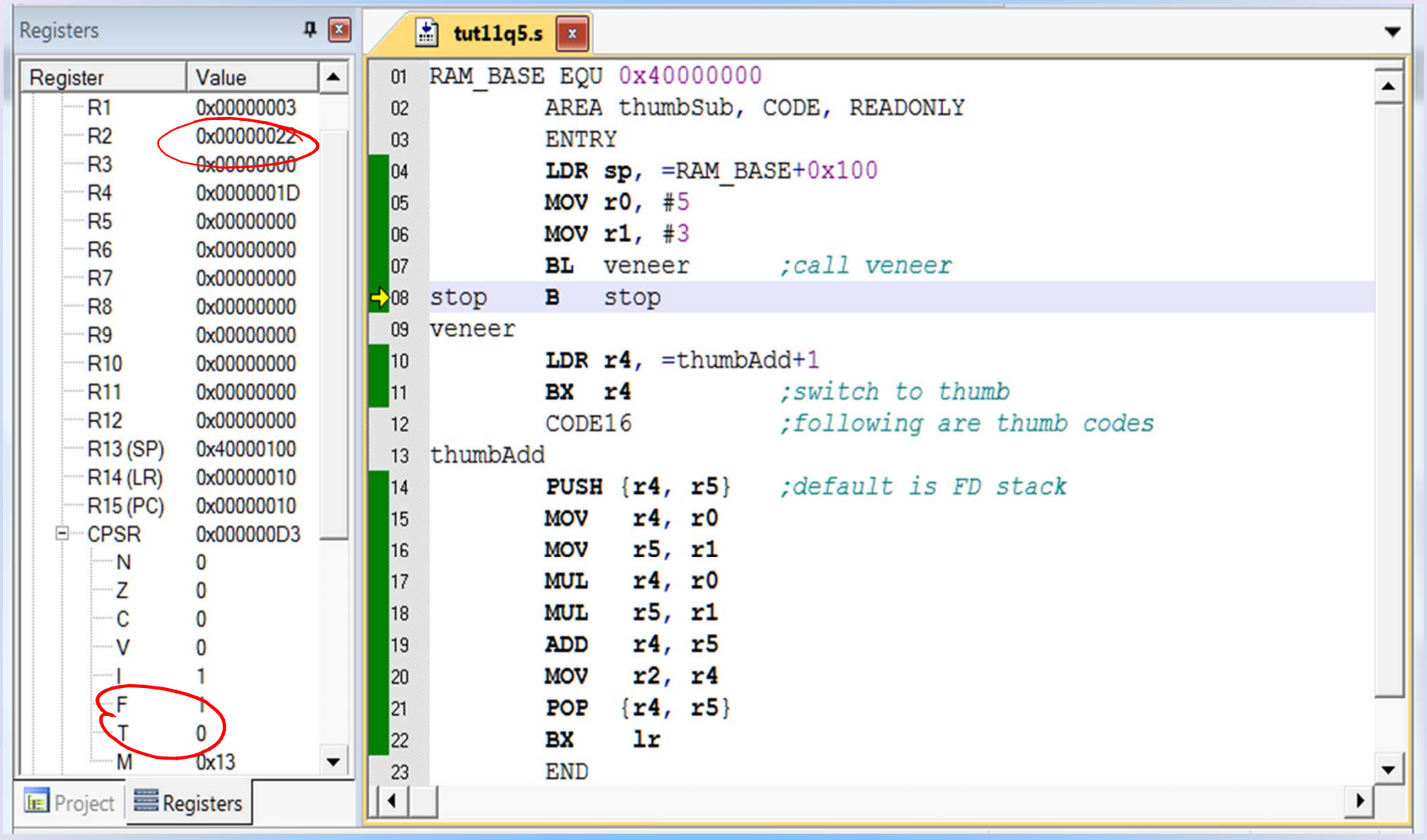
```
veneer      LDR    r4, =thumbAdd+1     ; set LSB to 1
            BX     r4                  ; switch to thumb

            CODE16                     ;following are thumb codes
thumbAdd
            PUSH  {r4, r5}             ;default is FD stack
            MOV   r4, r0
            MOV   r5, r1
            MUL   r4, r0               ; r4 = r0 x r0
            MUL   r5, r1               ; r5 = r1 x r1
            ADD   r4, r5               ; r4 = r0 x r0 + r1 x r1
            MOV   r2, r4               ; r2 = r0 x r0 + r1 x r1
            POP   {r4, r5}
            BX    lr

            END
```



Note: switch to Thumb state in thumb subroutine



Note: switch back to ARM state after returning from thumb subroutine