南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Embedded System and Microcomputer Principle

LAB4 ARMv7 Assembly Programming

2024 Fall
wangq9@mail.sustech.edu.cn

CONTENTS

1 Cortex-M3 introduction

2 CPUlator system simulator

3 Basic assembly instructions

4 Practice

01

Cortex-M3 introduction

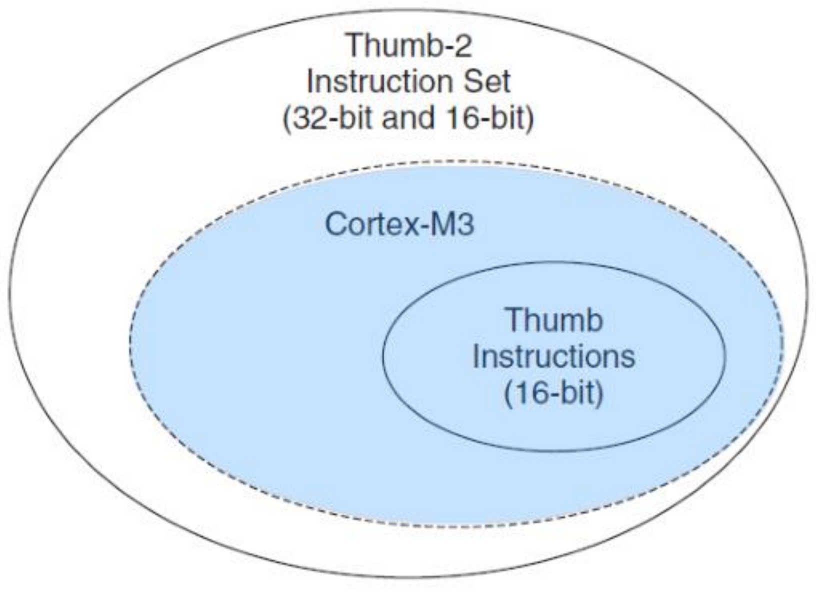
1. Cortex-M3 introduction

- Instruction Set Development
  - Historically (since ARM7TDMI), two different instruction sets are supported on the ARM processor: the ARM instructions that are 32-bit and Thumb instructions that are 16-bit.
  - During program execution, the processor can be dynamically switched between the ARM state or the Thumb state to use either one of the instruction sets.
  - The Thumb instruction set provides only a subset of the ARM instructions, but it can provide higher code density.
  - The Thumb instruction set is useful for products with tight memory requirements.

1. Cortex-M3 introduction

- The Thumb-2 Instruction Set Architecture (ISA)
  - In 2003, ARM announced the Thumb-2 instruction set, which is a new superset of Thumb instructions that contains both 16-bit and 32-bit instructions.
  - The Cortex-M3 supports only the Thumb-2 (and traditional Thumb) instruction set.
  - With support for both 16-bit and 32-bit instructions in the Thumb-2 instructions set, there is no need to switch the processor between Thumb state (16-bit instructions) and ARM state (32-bit instructions).

The Relationship between Thumb-2 and Thumb Instruction Sets


Thumb-2 Instruction Set (32-bit and 16-bit)
Cortex-M3
Thumb Instructions (16-bit)

1. Cortex-M3 introduction

- Registers (R0 to R15)
  - General-Purpose Registers R0–R7 (low registers)
    - They can be accessed by all 16-bit Thumb and all 32-bit Thumb-2 instructions.
    - They are all 32-bit; the reset value is unpredictable.
  - General-Purpose Registers R8–R12 (high registers)
    - They are accessible by all Thumb-2 but not by all 16-bit Thumb instructions.
    - These registers are all 32-bit; the reset value is unpredictable.
  - R13: Stack Pointers
    - The Cortex-M3 contains two stack pointers, R13. They are banked so that only one is visible at a time.
  - R14: The Link Register
    - When a subroutine is called, the return address is stored in the link register.
  - R15: The Program Counter
    - The program counter is the current program address. This register can be written to control the program flow.

1. Cortex-M3 introduction

- Special Registers
  - They have special functions and can be accessed only by special instructions.
  - They cannot be used for normal data processing.
  - The Cortex-M3 processor also has a number of special registers:
    - Program Status Registers (PSRs: APSR, IPSR, EPSR)
    - Interrupt Mask Registers (PRIMASK, FAULTMASK, BASEPRI)

Register	Function
xPSR	Provide ALU flags (Z, C), execution status, and current executing interrupt number
PRIMASK	Disable all interrupts except the nonmaskable interrupt (NMI) and HardFault
FAULTMASK	Disable all interrupts except the NMI
BASEPRI	Disable all interrupts of specific priority level or lower priority level
CONTROL	Define privileged status and stack pointer selection

02

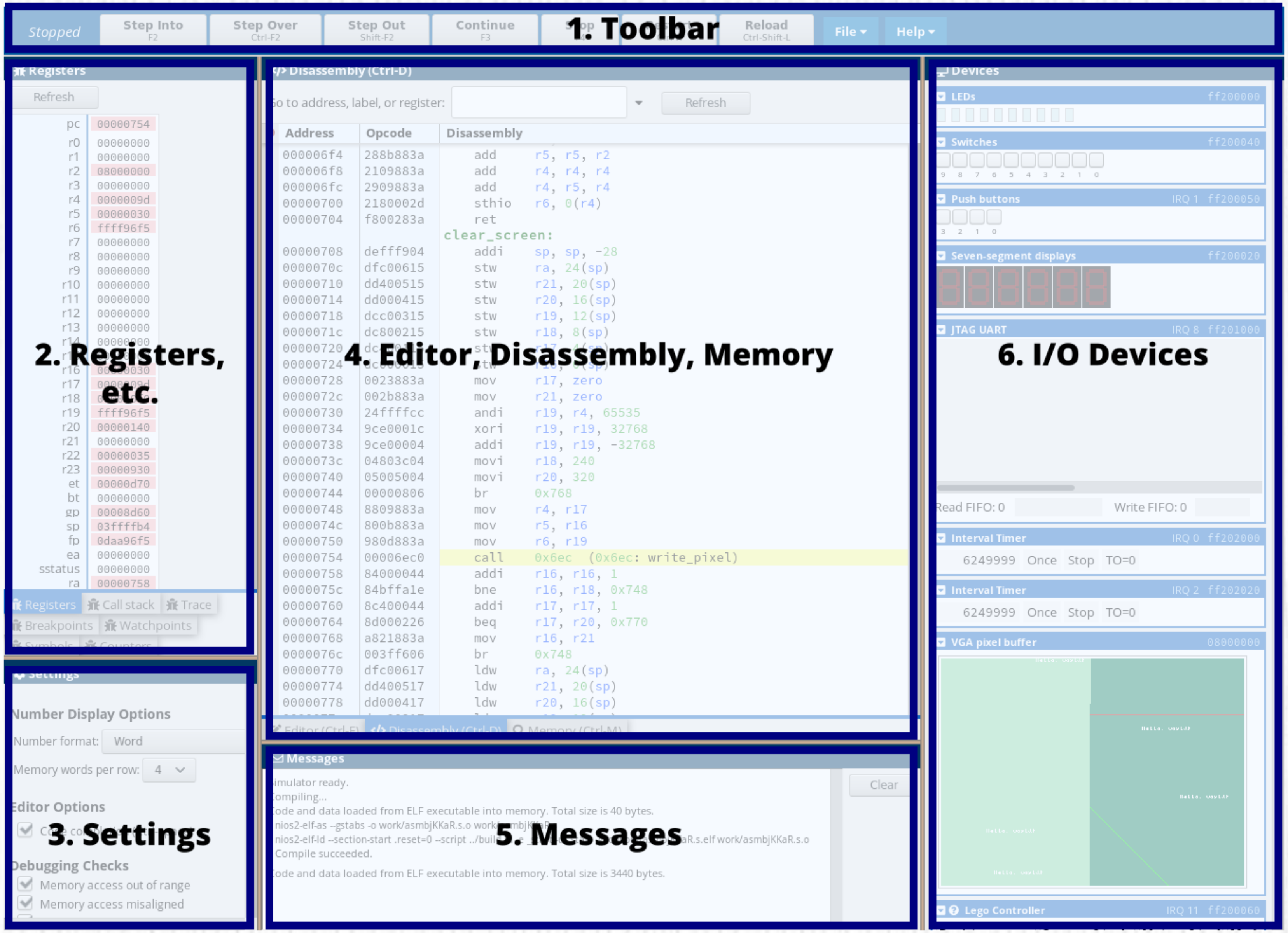
CPUlator system simulator

2. CPUlator system simulator

- CPUlator is a simulator and debugger of a computer system (CPU, memory, and I/O devices) that runs inside a web browser.
- https://cpulator.01xz.net/?sys=arm
- Features
  - No download
  - No account sign up
  - Only Internet

2. CPUlator system simulator

- The simulator interface consists of the toolbar and a collection of movable panels.



1. Toolbar

2. Registers, etc.

```
     pc  00000754
     r0  00000000
     r1  00000000
     r2  08000000
     r3  00000000
     r4  0000009d
     r5  00000030
     r6  ffff96f5
     r7  00000000
     r8  00000000
     r9  00000000
    r10  00000000
    r11  00000000
    r12  00000000
    r13  00000000
    r14  00000000
    r1[illegible]
    r16  00000030
    r17  0000009d
    r18  [illegible]
    r19  ffff96f5
    r20  00000140
    r21  00000000
    r22  00000035
    r23  00000930
     et  00000d70
     bt  00000000
     gp  00008d60
     sp  03ffffb4
     fp  0daa96f5
     ea  00000000
sstatus  00000000
     ra  00000758
```

3. Settings

Number Display Options

Number format: Word

Memory words per row: 4

Editor Options

☑ C[illegible]

Debugging Checks

☑ Memory access out of range

☑ Memory access misaligned

4. Editor, Disassembly, Memory

Go to address, label, or register:

```
Address   Opcode    Disassembly
000006f4  288b883a      add    r5, r5, r2
000006f8  2109883a      add    r4, r4, r4
000006fc  2909883a      add    r4, r5, r4
00000700  2180002d      sthio  r6, 0(r4)
00000704  f800283a      ret
                    clear_screen:
00000708  defff904      addi   sp, sp, -28
0000070c  dfc00615      stw    ra, 24(sp)
00000710  dd400515      stw    r21, 20(sp)
00000714  dd000415      stw    r20, 16(sp)
00000718  dcc00315      stw    r19, 12(sp)
0000071c  dc800215      stw    r18, 8(sp)
00000720  dc[illegible]  stw    r17, 4(sp)
00000724  dc[illegible]  stw    r16, 0(sp)
00000728  0023883a      mov    r17, zero
0000072c  002b883a      mov    r21, zero
00000730  24ffffcc      andi   r19, r4, 65535
00000734  9ce0001c      xori   r19, r19, 32768
00000738  9ce00004      addi   r19, r19, -32768
0000073c  04803c04      movi   r18, 240
00000740  05005004      movi   r20, 320
00000744  00000806      br     0x768
00000748  8809883a      mov    r4, r17
0000074c  800b883a      mov    r5, r16
00000750  980d883a      mov    r6, r19
00000754  00006ec0      call   0x6ec  (0x6ec: write_pixel)
00000758  84000044      addi   r16, r16, 1
0000075c  84bffa1e      bne    r16, r18, 0x748
00000760  8c400044      addi   r17, r17, 1
00000764  8d000226      beq    r17, r20, 0x770
00000768  a821883a      mov    r16, r21
0000076c  003ff606      br     0x748
00000770  dfc00617      ldw    ra, 24(sp)
00000774  dd400517      ldw    r21, 20(sp)
00000778  dd000417      ldw    r20, 16(sp)
```

5. Messages

```
imulator ready.
ompiling...
ode and data loaded from ELF executable into memory. Total size is 40 bytes.
 nios2-elf-as --gstabs -o work/asmbjKKaR.s.o work/asmbjKKaR...
 nios2-elf-ld --section-start .reset=0 --script ../buil[illegible]aR.s.elf work/asmbjKKaR.s.o
 Compile succeeded.
ode and data loaded from ELF executable into memory. Total size is 3440 bytes.
```

6. I/O Devices

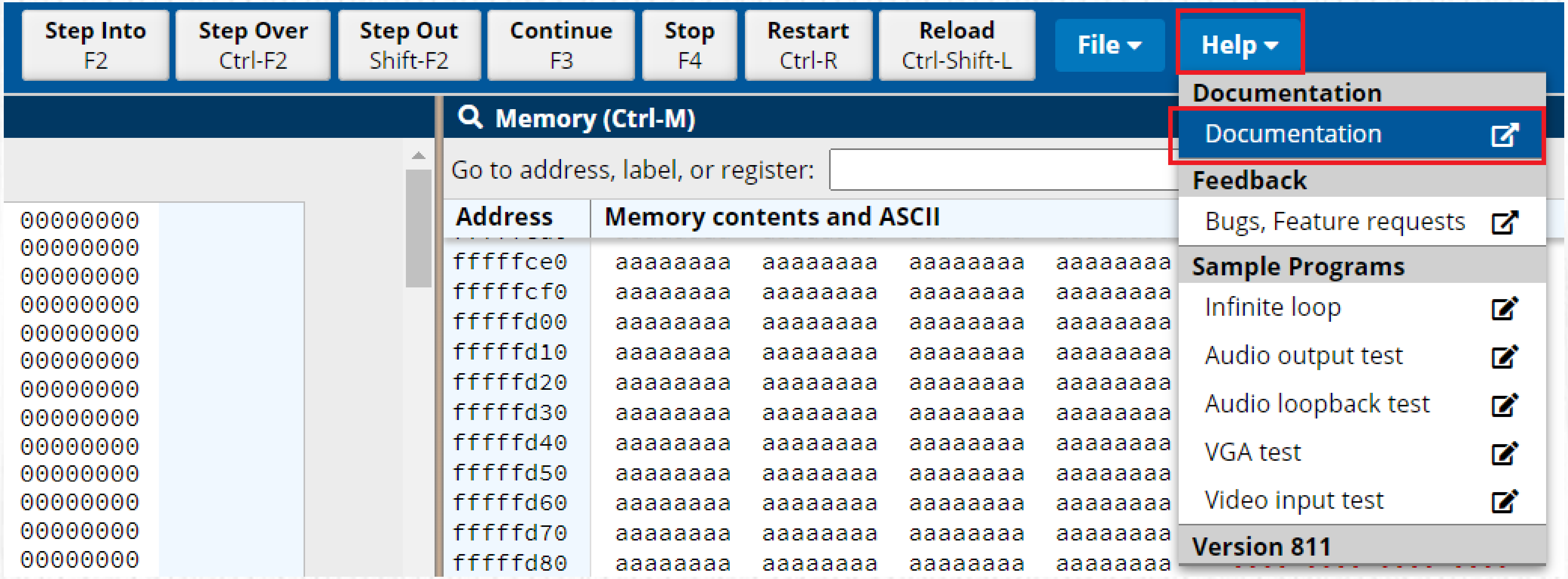
Read FIFO: 0   Write FIFO: 0

Interval Timer IRQ 0 ff202000: 6249999 Once Stop TO=0

Interval Timer IRQ 2 ff202020: 6249999 Once Stop TO=0

2. CPUlator system simulator

- The help documentation can provide more information.

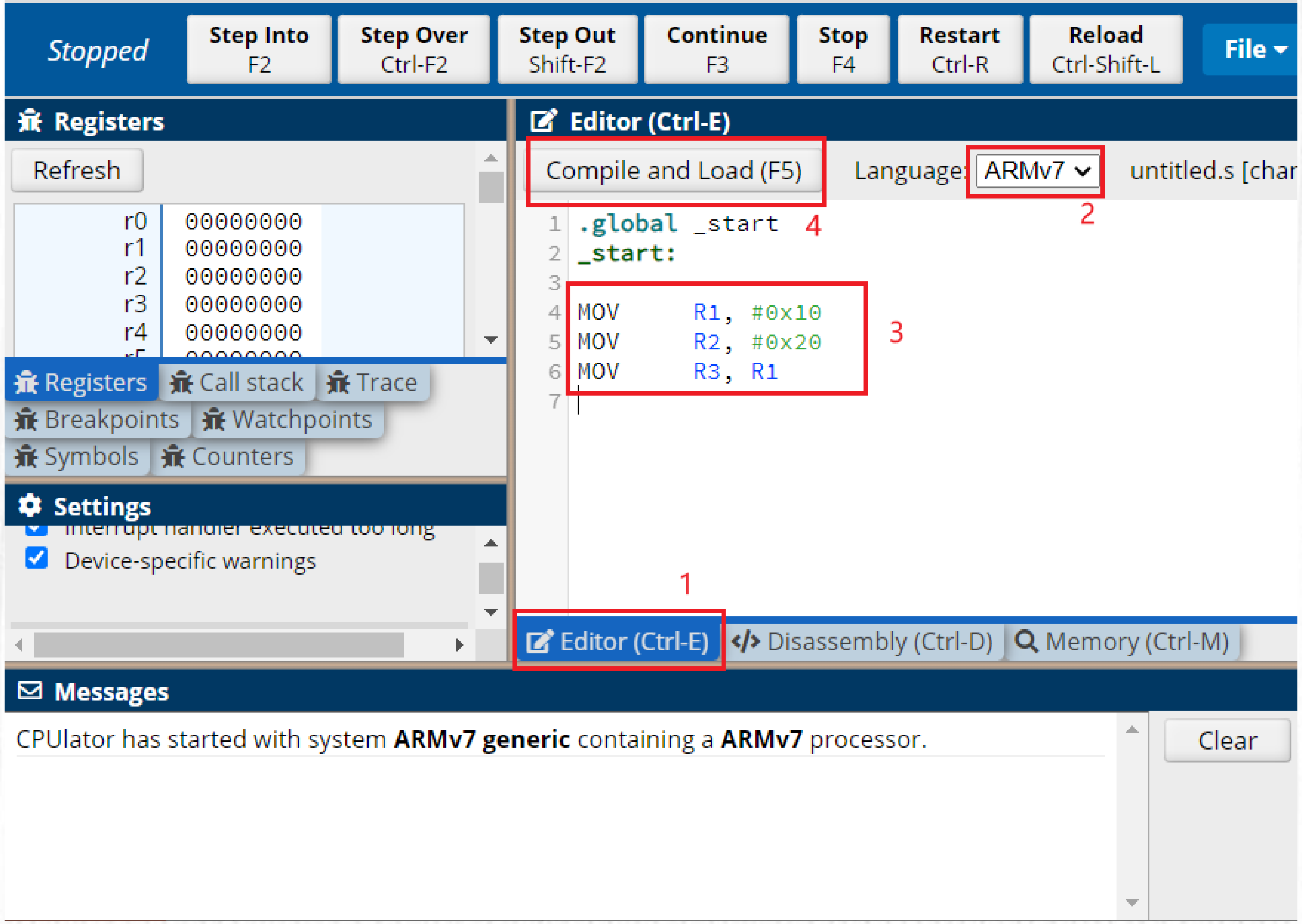


```
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
```

```
fffffce0   aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa
fffffcf0   aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa
fffffd00   aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa
fffffd10   aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa
fffffd20   aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa
fffffd30   aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa
fffffd40   aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa
fffffd50   aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa
fffffd60   aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa
fffffd70   aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa
fffffd80   aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa
```

2. CPUlator system simulator

- Load assembly codes.



Language: ARMv7

```
r0  00000000
r1  00000000
r2  00000000
r3  00000000
r4  00000000
```

```
.global _start
_start:

MOV      R1, #0x10
MOV      R2, #0x20
MOV      R3, R1

```

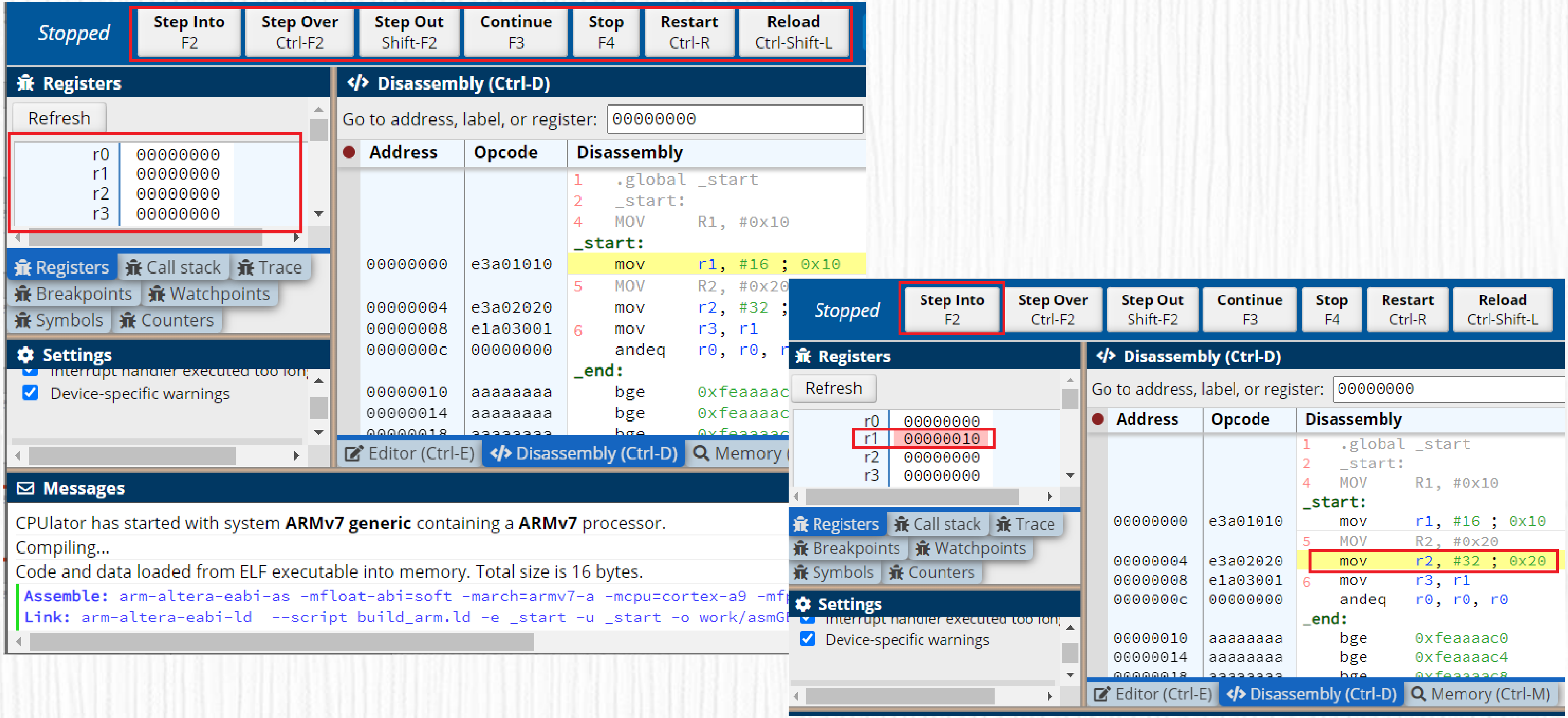
☑ Interrupt handler executed too long

☑ Device-specific warnings

CPUlator has started with system **ARMv7 generic** containing a **ARMv7** processor.

2. CPUlator system simulator

- Run the codes.



```
r0  00000000
r1  00000000
r2  00000000
r3  00000000
```

Interrupt handler executed too lon

☑ Device-specific warnings

Go to address, label, or register: 00000000

```
Address   Opcode    Disassembly
                    1    .global _start
                    2    _start:
                    4    MOV      R1, #0x10
                    _start:
00000000  e3a01010       mov      r1, #16 ; 0x10
                    5    MOV      R2, #0x20
00000004  e3a02020       mov      r2, #32 ;
00000008  e1a03001  6    mov      r3, r1
0000000c  00000000       andeq    r0, r0, r
                    _end:
00000010  aaaaaaaa       bge      0xfeaaaac
00000014  aaaaaaaa       bge      0xfeaaaac
00000018  aaaaaaaa       bge      0xfeaaaac
```

CPUlator has started with system **ARMv7 generic** containing a **ARMv7** processor.
Compiling...
Code and data loaded from ELF executable into memory. Total size is 16 bytes.

```
Assemble: arm-altera-eabi-as -mfloat-abi=soft -march=armv7-a -mcpu=cortex-a9 -mf
Link: arm-altera-eabi-ld  --script build_arm.ld -e _start -u _start -o work/asmG
```

```
r0  00000000
r1  00000010
r2  00000000
r3  00000000
```

Interrupt handler executed too lon

☑ Device-specific warnings

Go to address, label, or register: 00000000

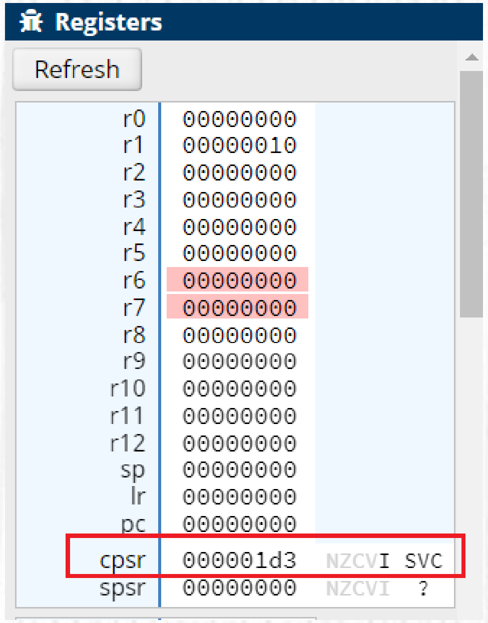
```
Address   Opcode    Disassembly
                    1    .global _start
                    2    _start:
                    4    MOV      R1, #0x10
                    _start:
00000000  e3a01010       mov      r1, #16 ; 0x10
                    5    MOV      R2, #0x20
00000004  e3a02020       mov      r2, #32 ; 0x20
00000008  e1a03001  6    mov      r3, r1
0000000c  00000000       andeq    r0, r0, r0
                    _end:
00000010  aaaaaaaa       bge      0xfeaaaac0
00000014  aaaaaaaa       bge      0xfeaaaac4
00000018  aaaaaaaa       bge      0xfeaaaac8
```

03

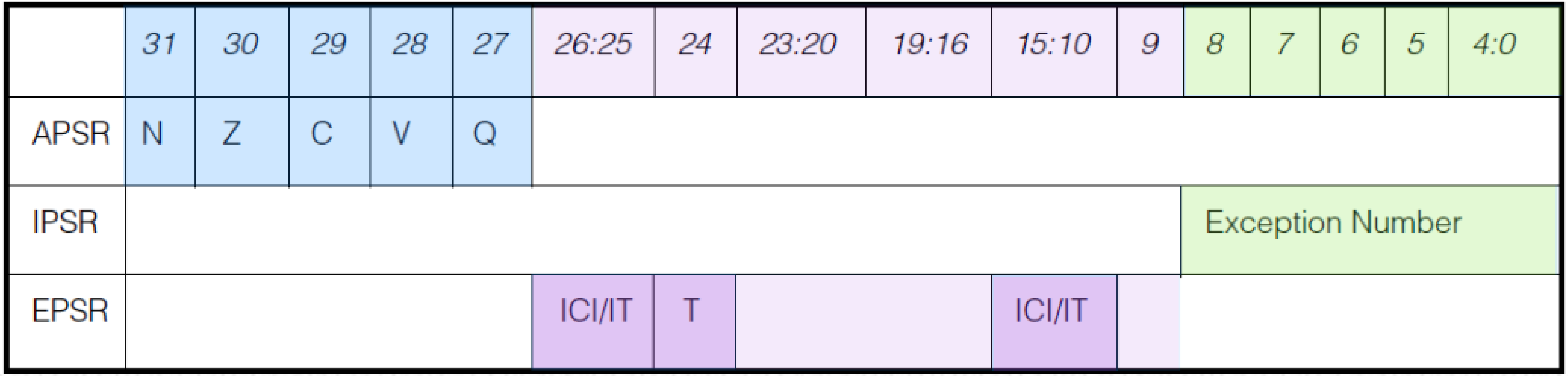
Basic assembly instructions

3. Basic assembly instructions

- Bit Fields in Cortex-M3 Program Status Registers
  - N: Negative
  - Z: Zero
  - C: Carry / Borrow
  - V: Overflow



```
  r0  00000000
  r1  00000010
  r2  00000000
  r3  00000000
  r4  00000000
  r5  00000000
  r6  00000000
  r7  00000000
  r8  00000000
  r9  00000000
 r10  00000000
 r11  00000000
 r12  00000000
  sp  00000000
  lr  00000000
  pc  00000000
cpsr  000001d3   NZCVI SVC
spsr  00000000   NZCVI   ?
```

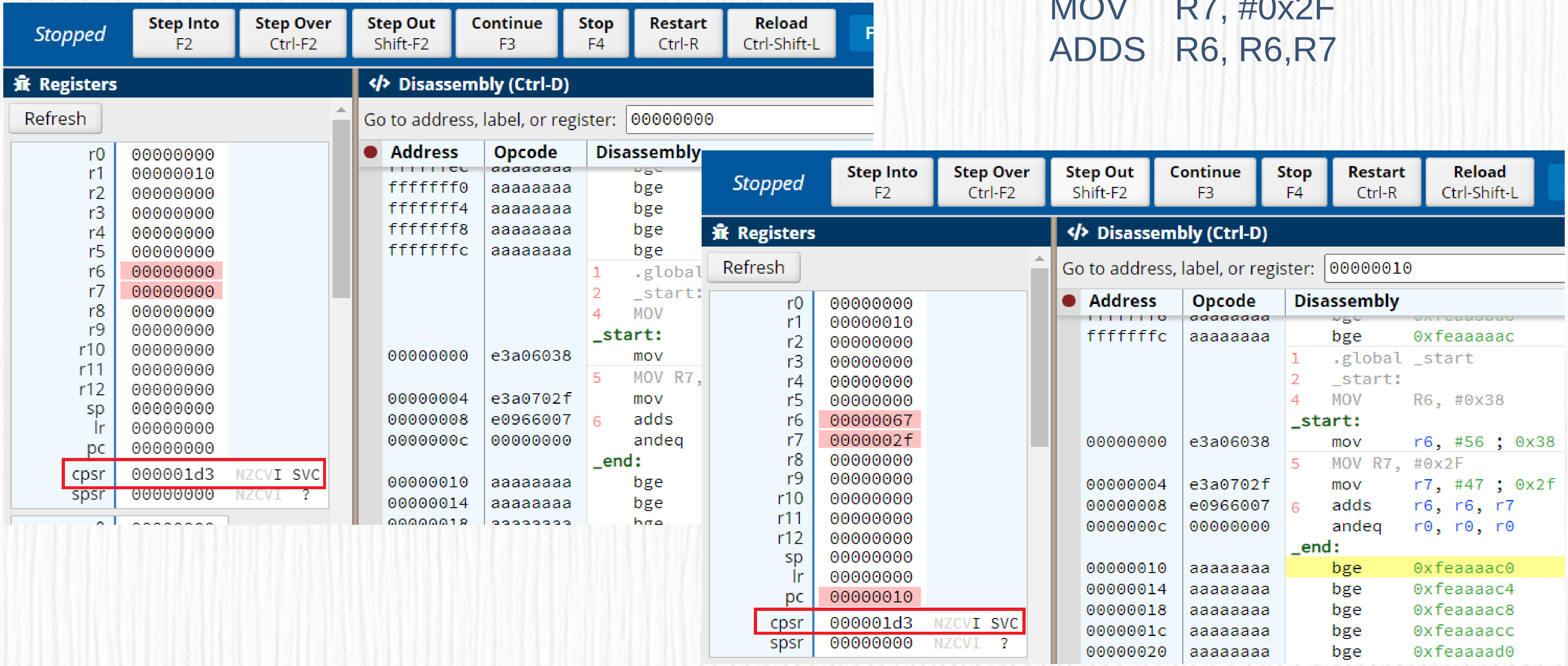


	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
APSR	N	Z	C	V	Q											
IPSR												Exception Number				
EPSR						ICI/IT	T			ICI/IT						

3. Basic assembly instructions

- Flag related instructions demo1

```
MOV     R6, #0x38
MOV     R7, #0x2F
ADDS    R6, R6,R7
```



Go to address, label, or register: 00000000

```
r0    00000000
r1    00000010
r2    00000000
r3    00000000
r4    00000000
r5    00000000
r6    00000000
r7    00000000
r8    00000000
r9    00000000
r10   00000000
r11   00000000
r12   00000000
sp    00000000
lr    00000000
pc    00000000
cpsr  000001d3   NZCVI SVC
spsr  00000000   NZCVI  ?
```

```
Address    Opcode     Disassembly
fffffff0   aaaaaaaa       bge
fffffff4   aaaaaaaa       bge
fffffff8   aaaaaaaa       bge
fffffffc   aaaaaaaa       bge
                     1    .global
                     2    _start:
                     4    MOV
                     _start:
00000000   e3a06038       mov
                     5    MOV R7,
00000004   e3a0702f       mov
00000008   e0966007  6    adds
0000000c   00000000       andeq
                     _end:
00000010   aaaaaaaa       bge
00000014   aaaaaaaa       bge
```

Go to address, label, or register: 00000010

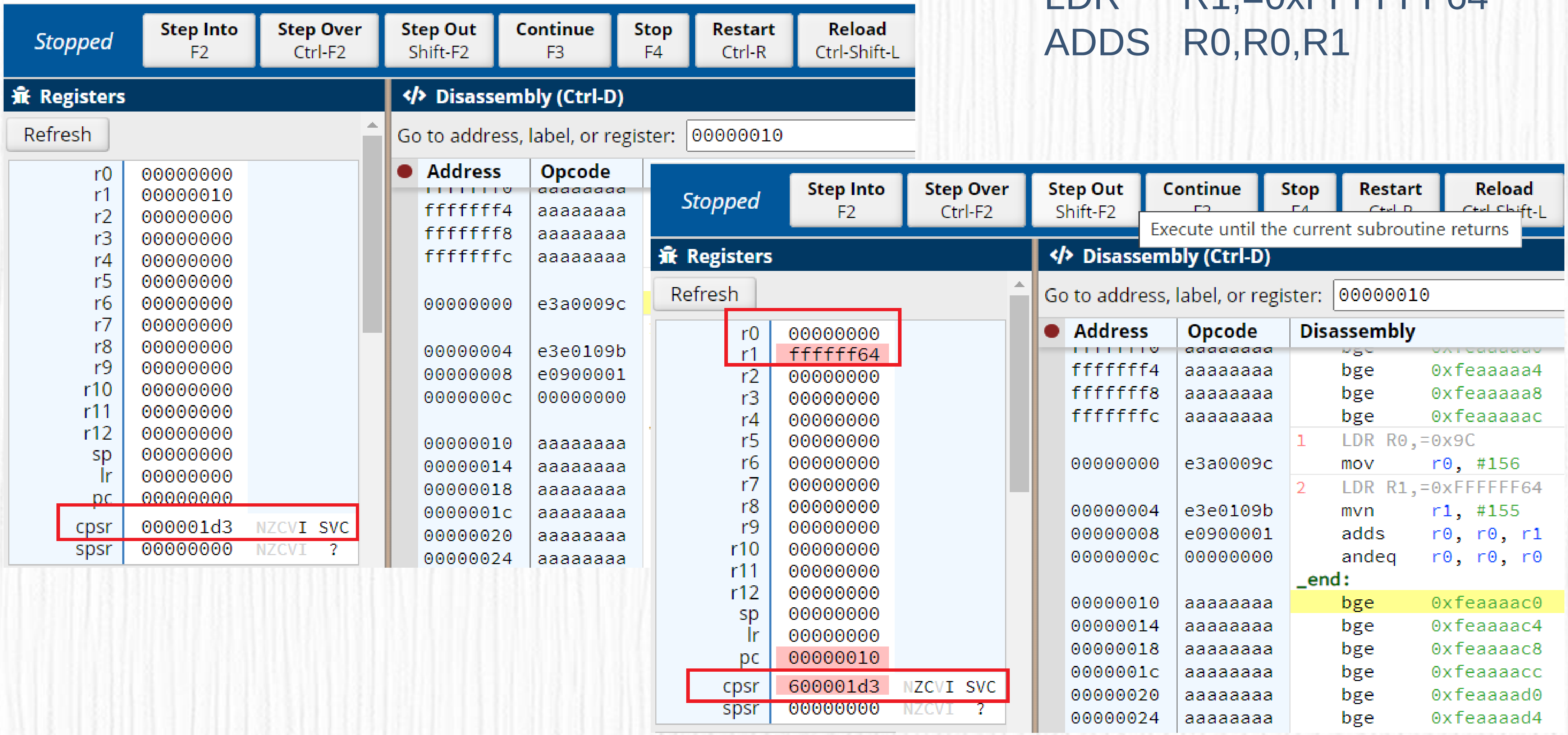
```
r0    00000000
r1    00000010
r2    00000000
r3    00000000
r4    00000000
r5    00000000
r6    00000067
r7    0000002f
r8    00000000
r9    00000000
r10   00000000
r11   00000000
r12   00000000
sp    00000000
lr    00000000
pc    00000010
cpsr  000001d3   NZCVI SVC
spsr  00000000   NZCVI  ?
```

```
Address    Opcode     Disassembly
fffffffc   aaaaaaaa       bge     0xfeaaaaac
                     1    .global _start
                     2    _start:
                     4    MOV     R6, #0x38
                     _start:
00000000   e3a06038       mov     r6, #56 ; 0x38
                     5    MOV R7, #0x2F
00000004   e3a0702f       mov     r7, #47 ; 0x2f
00000008   e0966007  6    adds    r6, r6, r7
0000000c   00000000       andeq   r0, r0, r0
                     _end:
00000010   aaaaaaaa       bge     0xfeaaaac0
00000014   aaaaaaaa       bge     0xfeaaaac4
00000018   aaaaaaaa       bge     0xfeaaaac8
0000001c   aaaaaaaa       bge     0xfeaaaacc
00000020   aaaaaaaa       bge     0xfeaaaad0
```

3. Basic assembly instructions

- Flag related instructions demo2

```
LDR     R0,=0x9C
LDR     R1,=0xFFFFFF64
ADDS    R0,R0,R1
```



Go to address, label, or register: 00000010

```
r0    00000000
r1    00000010
r2    00000000
r3    00000000
r4    00000000
r5    00000000
r6    00000000
r7    00000000
r8    00000000
r9    00000000
r10   00000000
r11   00000000
r12   00000000
sp    00000000
lr    00000000
pc    00000000
cpsr  000001d3  NZCVI SVC
spsr  00000000  NZCVI  ?
```

```
Address    Opcode
fffffff4   aaaaaaaa
fffffff8   aaaaaaaa
fffffffc   aaaaaaaa

00000000   e3a0009c

00000004   e3e0109b
00000008   e0900001
0000000c   00000000

00000010   aaaaaaaa
00000014   aaaaaaaa
00000018   aaaaaaaa
0000001c   aaaaaaaa
00000020   aaaaaaaa
00000024   aaaaaaaa
```

Go to address, label, or register: 00000010

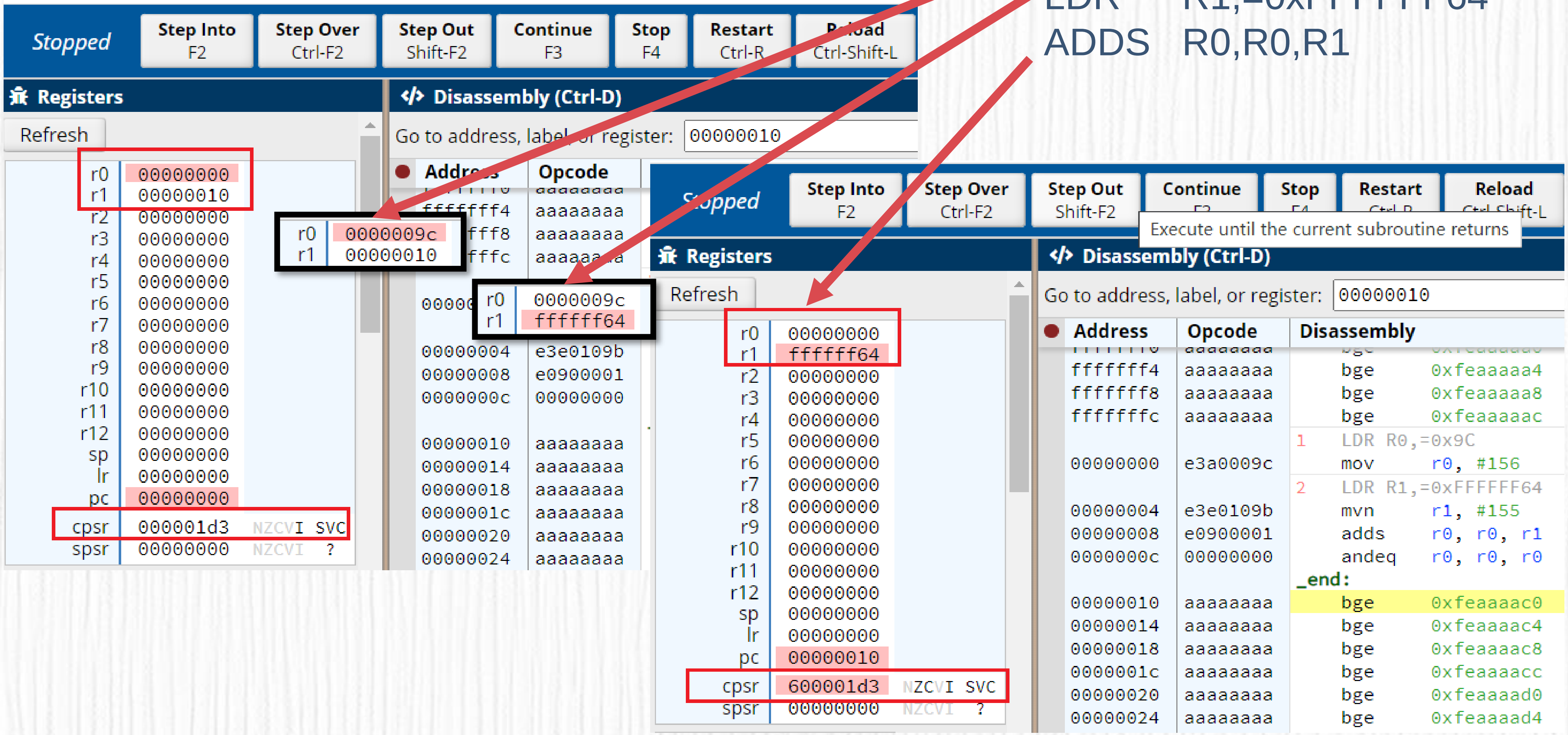
```
r0    00000000
r1    ffffff64
r2    00000000
r3    00000000
r4    00000000
r5    00000000
r6    00000000
r7    00000000
r8    00000000
r9    00000000
r10   00000000
r11   00000000
r12   00000000
sp    00000000
lr    00000000
pc    00000010
cpsr  600001d3  NZCVI SVC
spsr  00000000  NZCVI  ?
```

```
Address    Opcode     Disassembly
fffffff4   aaaaaaaa   bge   0xfeaaaaa4
fffffff8   aaaaaaaa   bge   0xfeaaaaa8
fffffffc   aaaaaaaa   bge   0xfeaaaaac
                      1  LDR R0,=0x9C
00000000   e3a0009c      mov    r0, #156
                      2  LDR R1,=0xFFFFFF64
00000004   e3e0109b      mvn    r1, #155
00000008   e0900001      adds   r0, r0, r1
0000000c   00000000      andeq  r0, r0, r0
                      _end:
00000010   aaaaaaaa      bge    0xfeaaaac0
00000014   aaaaaaaa      bge    0xfeaaaac4
00000018   aaaaaaaa      bge    0xfeaaaac8
0000001c   aaaaaaaa      bge    0xfeaaaacc
00000020   aaaaaaaa      bge    0xfeaaaad0
00000024   aaaaaaaa      bge    0xfeaaaad4
```

3. Basic assembly instructions

- Flag related instructions demo2

```
LDR     R0,=0x9C
LDR     R1,=0xFFFFFF64
ADDS    R0,R0,R1
```



```
r0    00000000
r1    00000010
r2    00000000
r3    00000000
r4    00000000
r5    00000000
r6    00000000
r7    00000000
r8    00000000
r9    00000000
r10   00000000
r11   00000000
r12   00000000
sp    00000000
lr    00000000
pc    00000000
cpsr  000001d3  NZCVI SVC
spsr  00000000  NZCVI  ?
```

```
Go to address, label, or register: 00000010

Address   Opcode
fffffff4  aaaaaaaa
fffffff8  aaaaaaaa
fffffffc  aaaaaaaa

0000000
00000004  e3e0109b
00000008  e0900001
0000000c  00000000

00000010  aaaaaaaa
00000014  aaaaaaaa
00000018  aaaaaaaa
0000001c  aaaaaaaa
00000020  aaaaaaaa
00000024  aaaaaaaa
```

```
r0  0000009c
r1  00000010
```

```
r0  0000009c
r1  ffffff64
```

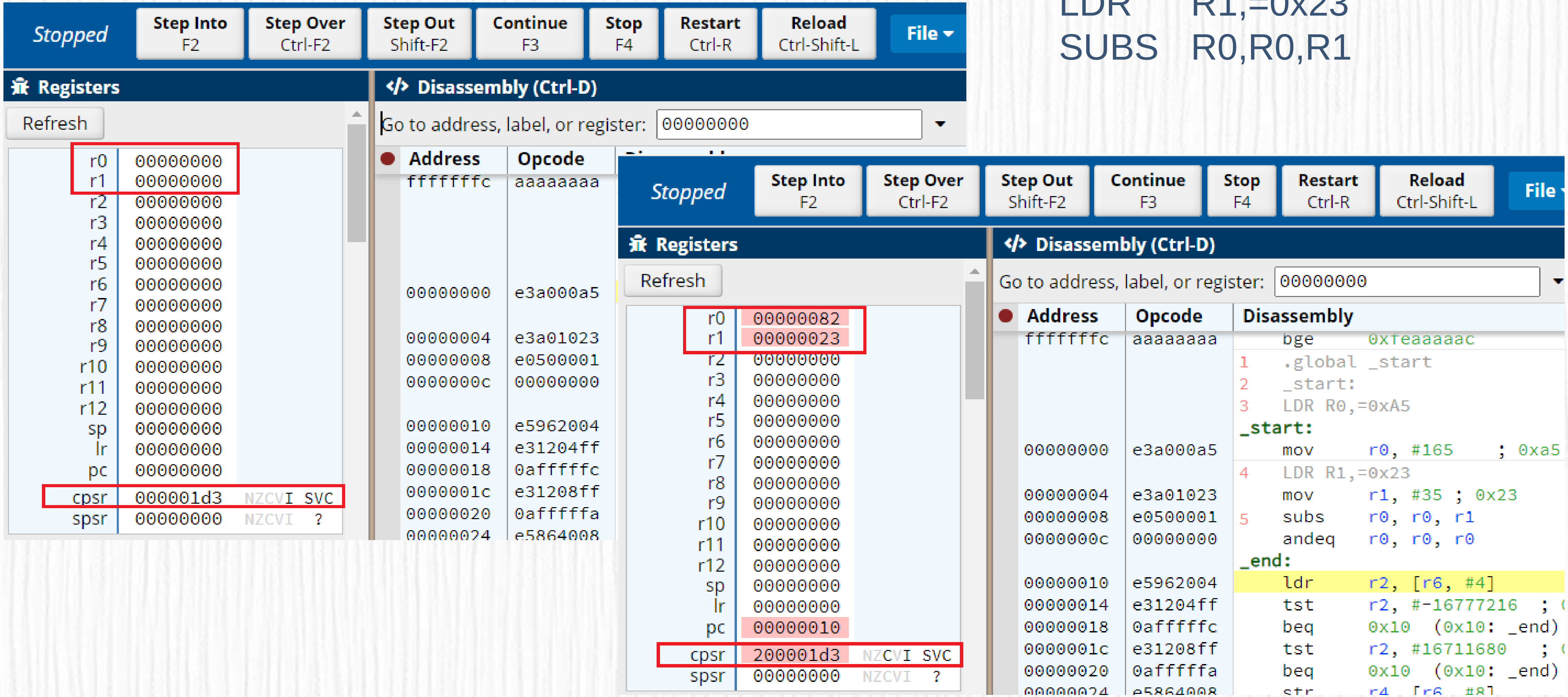
```
r0    00000000
r1    ffffff64
r2    00000000
r3    00000000
r4    00000000
r5    00000000
r6    00000000
r7    00000000
r8    00000000
r9    00000000
r10   00000000
r11   00000000
r12   00000000
sp    00000000
lr    00000000
pc    00000010
cpsr  600001d3  NZCVI SVC
spsr  00000000  NZCVI  ?
```

```
Go to address, label, or register: 00000010

Address   Opcode    Disassembly
fffffff4  aaaaaaaa    bge     0xfeaaaaa4
fffffff8  aaaaaaaa    bge     0xfeaaaaa8
fffffffc  aaaaaaaa    bge     0xfeaaaaac
                    1 LDR R0,=0x9C
00000000  e3a0009c    mov     r0, #156
                    2 LDR R1,=0xFFFFFF64
00000004  e3e0109b    mvn     r1, #155
00000008  e0900001    adds    r0, r0, r1
0000000c  00000000    andeq   r0, r0, r0
                    _end:
00000010  aaaaaaaa    bge     0xfeaaaac0
00000014  aaaaaaaa    bge     0xfeaaaac4
00000018  aaaaaaaa    bge     0xfeaaaac8
0000001c  aaaaaaaa    bge     0xfeaaaacc
00000020  aaaaaaaa    bge     0xfeaaaad0
00000024  aaaaaaaa    bge     0xfeaaaad4
```

3. Basic assembly instructions

- Flag related instructions demo3

```
LDR     R0,=0xA5
LDR     R1,=0x23
SUBS    R0,R0,R1
```



Go to address, label, or register: 00000000

```
r0    00000000
r1    00000000
r2    00000000
r3    00000000
r4    00000000
r5    00000000
r6    00000000
r7    00000000
r8    00000000
r9    00000000
r10   00000000
r11   00000000
r12   00000000
sp    00000000
lr    00000000
pc    00000000
cpsr  000001d3  NZCVI SVC
spsr  00000000  NZCVI  ?
```

```
Address   Opcode
fffffffc  aaaaaaaa

00000000  e3a000a5

00000004  e3a01023
00000008  e0500001
0000000c  00000000

00000010  e5962004
00000014  e31204ff
00000018  0afffffc
0000001c  e31208ff
00000020  0afffffa
00000024  e5864008
```

Go to address, label, or register: 00000000

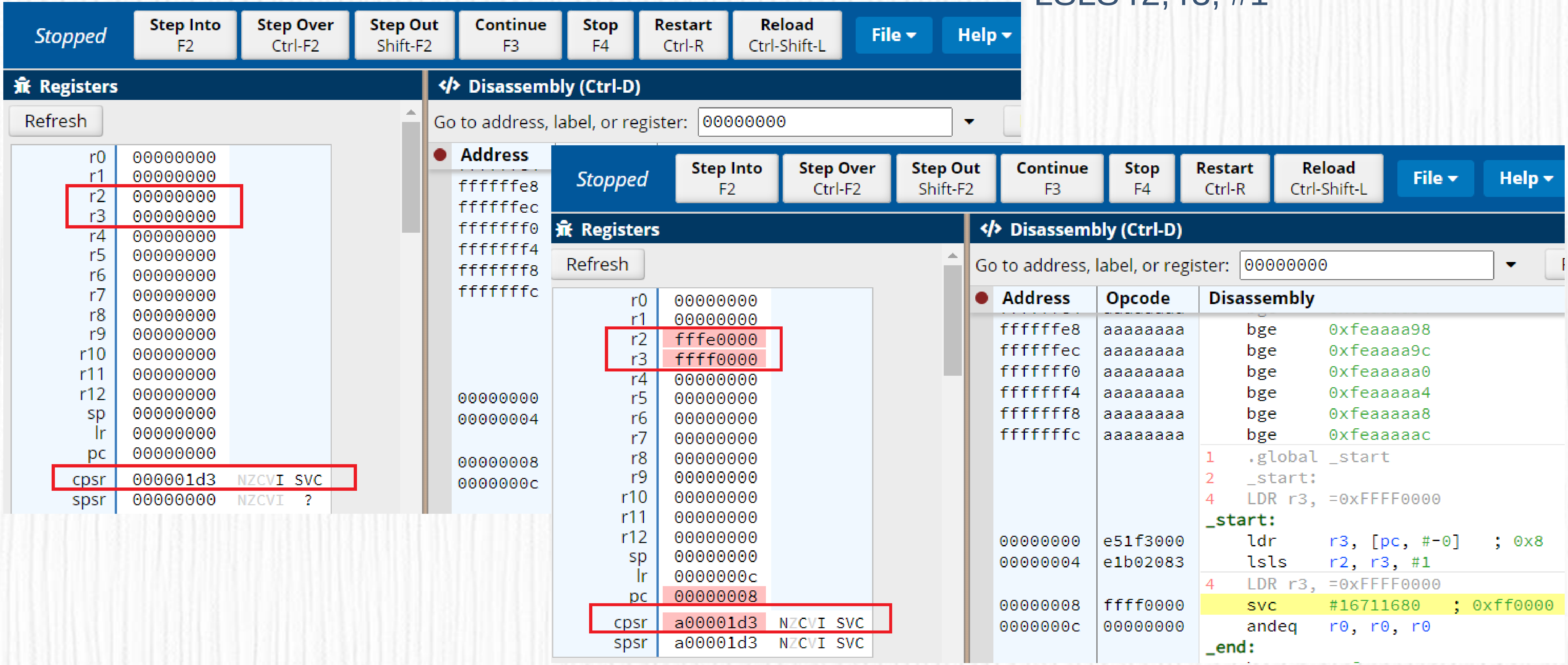
```
r0    00000082
r1    00000023
r2    00000000
r3    00000000
r4    00000000
r5    00000000
r6    00000000
r7    00000000
r8    00000000
r9    00000000
r10   00000000
r11   00000000
r12   00000000
sp    00000000
lr    00000000
pc    00000010
cpsr  200001d3  NZCVI SVC
spsr  00000000  NZCVI  ?
```

```
Address   Opcode    Disassembly
fffffffc  aaaaaaaa    bge     0xfeaaaaac
                      .global _start
                      _start:
                      LDR R0,=0xA5
                    _start:
00000000  e3a000a5    mov     r0, #165    ; 0xa5
                      LDR R1,=0x23
00000004  e3a01023    mov     r1, #35  ; 0x23
00000008  e0500001    subs    r0, r0, r1
0000000c  00000000    andeq   r0, r0, r0
                    _end:
00000010  e5962004    ldr     r2, [r6, #4]
00000014  e31204ff    tst     r2, #-16777216   ; (
00000018  0afffffc    beq     0x10  (0x10: _end)
0000001c  e31208ff    tst     r2, #16711680    ; (
00000020  0afffffa    beq     0x10  (0x10: _end)
00000024  e5864008    str     r4, [r6, #8]
```

3. Basic assembly instructions

- Flag related instructions demo4

```
LDR r3, =0xFFFF0000
LSLS r2, r3, #1
```



Go to address, label, or register: 00000000

```
r0     00000000
r1     00000000
r2     00000000
r3     00000000
r4     00000000
r5     00000000
r6     00000000
r7     00000000
r8     00000000
r9     00000000
r10    00000000
r11    00000000
r12    00000000
sp     00000000
lr     00000000
pc     00000000
cpsr   000001d3  NZCVI SVC
spsr   00000000  NZCVI  ?
```

```
Address
ffffffe8
ffffffec
fffffff0
fffffff4
fffffff8
fffffffc

00000000
00000004

00000008
0000000c
```

Go to address, label, or register: 00000000

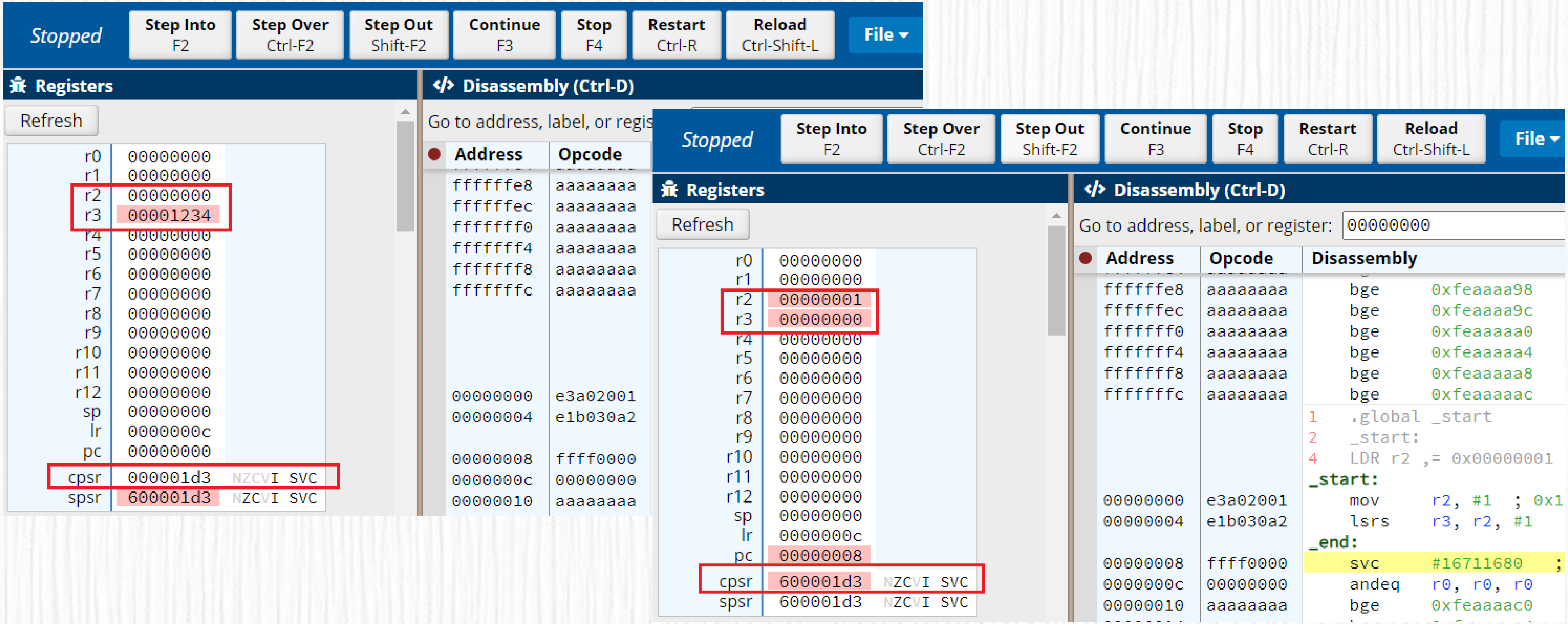
```
r0     00000000
r1     00000000
r2     fffe0000
r3     ffff0000
r4     00000000
r5     00000000
r6     00000000
r7     00000000
r8     00000000
r9     00000000
r10    00000000
r11    00000000
r12    00000000
sp     00000000
lr     0000000c
pc     00000008
cpsr   a00001d3  NZCVI SVC
spsr   a00001d3  NZCVI SVC
```

```
Address   Opcode    Disassembly
ffffffe8  aaaaaaaa     bge     0xfeaaaa98
ffffffec  aaaaaaaa     bge     0xfeaaaa9c
fffffff0  aaaaaaaa     bge     0xfeaaaaa0
fffffff4  aaaaaaaa     bge     0xfeaaaaa4
fffffff8  aaaaaaaa     bge     0xfeaaaaa8
fffffffc  aaaaaaaa     bge     0xfeaaaaac
                    1  .global _start
                    2  _start:
                    4  LDR r3, =0xFFFF0000
                    _start:
00000000  e51f3000     ldr     r3, [pc, #-0]    ; 0x8
00000004  e1b02083     lsls    r2, r3, #1
                    4  LDR r3, =0xFFFF0000
00000008  ffff0000     svc     #16711680     ; 0xff0000
0000000c  00000000     andeq   r0, r0, r0
                    _end:
```

3. Basic assembly instructions

- Flag related instructions demo5

```
LDR r2 ,= 0x00000001
LSRS r3, r2, #1
```



```
r0    00000000
r1    00000000
r2    00000000
r3    00001234
r4    00000000
r5    00000000
r6    00000000
r7    00000000
r8    00000000
r9    00000000
r10   00000000
r11   00000000
r12   00000000
sp    00000000
lr    0000000c
pc    00000000
cpsr  000001d3  NZCVI SVC
spsr  600001d3  NZCVI SVC
```

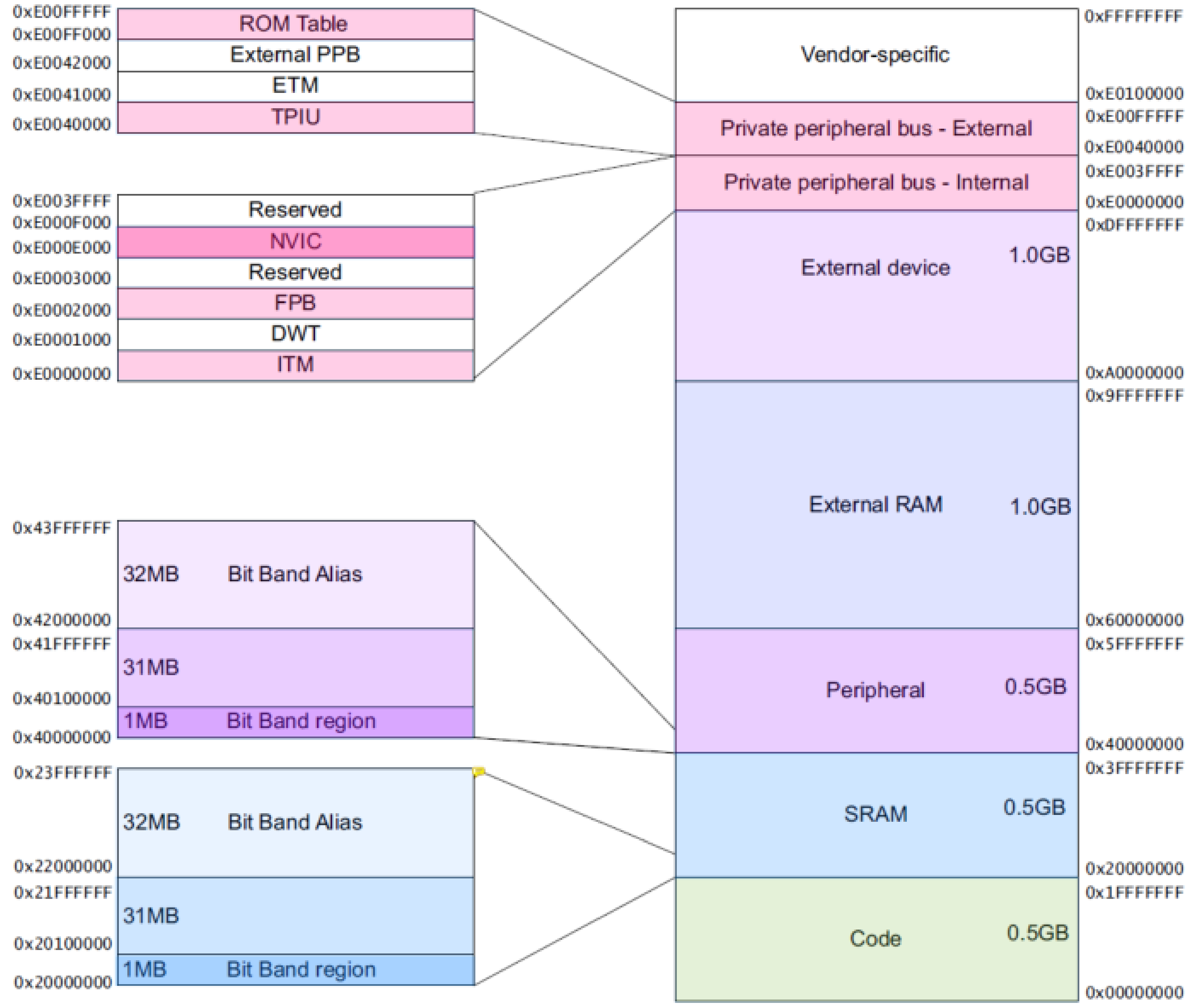
```
Address   Opcode
fffffe8   aaaaaaaa
ffffffec  aaaaaaaa
fffffff0  aaaaaaaa
fffffff4  aaaaaaaa
fffffff8  aaaaaaaa
fffffffc  aaaaaaaa

00000000  e3a02001
00000004  e1b030a2

00000008  ffff0000
0000000c  00000000
00000010  aaaaaaaa
```

```
r0    00000000
r1    00000000
r2    00000001
r3    00000000
r4    00000000
r5    00000000
r6    00000000
r7    00000000
r8    00000000
r9    00000000
r10   00000000
r11   00000000
r12   00000000
sp    00000000
lr    0000000c
pc    00000008
cpsr  600001d3  NZCVI SVC
spsr  600001d3  NZCVI SVC
```

```
Address   Opcode    Disassembly
ffffffe8  aaaaaaaa      bge     0xfeaaaa98
ffffffec  aaaaaaaa      bge     0xfeaaaa9c
fffffff0  aaaaaaaa      bge     0xfeaaaaa0
fffffff4  aaaaaaaa      bge     0xfeaaaaa4
fffffff8  aaaaaaaa      bge     0xfeaaaaa8
fffffffc  aaaaaaaa      bge     0xfeaaaaac
                    1   .global _start
                    2   _start:
                    4   LDR r2 ,= 0x00000001
                    _start:
00000000  e3a02001      mov     r2, #1   ; 0x1
00000004  e1b030a2      lsrs    r3, r2, #1
                    _end:
00000008  ffff0000      svc     #16711680   ;
0000000c  00000000      andeq   r0, r0, r0
00000010  aaaaaaaa      bge     0xfeaaaac0
```

3. Basic assembly instructions

- The Cortex-M3 processor has a total of 4 GB of address space.
- Program code can be located in the Code region, the SRAM region, or the External RAM region.
- Another 0.5 GB block of address range is allocated to on-chip peripherals.
- Two slots of 1 GB memory space are allocated for external RAM and external devices.
- The last 0.5 GB of memory is for the system-level omponents, internal peripheral buses, external peripheral bus, and vendor-specific system peripherals.



0xE00FFFFF
ROM Table
0xE00FF000
External PPB
0xE0042000
ETM
0xE0041000
TPIU
0xE0040000

0xE003FFFF
Reserved
0xE000F000
NVIC
0xE000E000
Reserved
0xE0003000
FPB
0xE0002000
DWT
0xE0001000
ITM
0xE0000000

0x43FFFFFF
32MB Bit Band Alias
0x42000000
0x41FFFFFF
31MB
0x40100000
1MB Bit Band region
0x40000000

0x23FFFFFF
32MB Bit Band Alias
0x22000000
0x21FFFFFF
31MB
0x20100000
1MB Bit Band region
0x20000000

0xFFFFFFFF
Vendor-specific
0xE0100000
0xE00FFFFF
Private peripheral bus - External
0xE0040000
0xE003FFFF
Private peripheral bus - Internal
0xE0000000
0xDFFFFFFF
External device 1.0GB
0xA0000000
0x9FFFFFFF
External RAM 1.0GB
0x60000000
0x5FFFFFFF
Peripheral 0.5GB
0x40000000
0x3FFFFFFF
SRAM 0.5GB
0x20000000
0x1FFFFFFF
Code 0.5GB
0x00000000

3. Basic assembly instructions

- Common memory access instructions

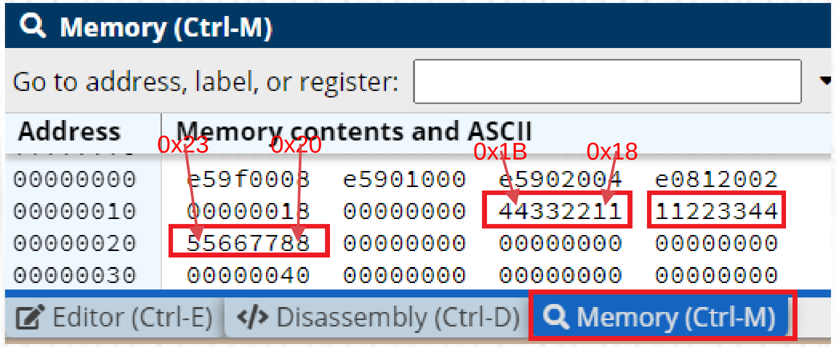
Example	Description
LDRB Rd, [Rn, #offset]	Read byte from memory location Rn+offset
LDRH Rd, [Rn, #offset]	Read half-word from memory location Rn+offset
LDR Rd, [Rn, #offset]	Read word from memory location Rn+offset
LDRD Rd1, Rd2, [Rn, #offset]	Read double word (64-bits) from memory location Rn+offset, set Rd1 as lower 32-bits and Rd2 as higher 32-bits
STRB Rd, [Rn, #offset]	Store byte to memory location Rn+offset
STRH Rd, [Rn, #offset]	Store half-word to memory location Rn+offset
STR Rd, [Rn, #offset]	Store word to memory location Rn+offset
LDRD Rd1, Rd2, [Rn, #offset]	Store double word to memory location Rn+offset

3. Basic assembly instructions

- Memory access instruction demo

```
.global _start
_start:
        LDR R0,=list
        LDR R1,[R0]
        LDR R2,[R0, #4]
        ADD R2, R1, R2
.data
list:
        .byte 0x11, 0x22, 0x33, 0x44
        .word 0x11223344, 0x55667788
```

Read this piece of codes, tell the result of R2, describe how to load data from memory?



```
Address   Memory contents and ASCII
00000000  e59f0008  e5901000  e5902004  e0812002
00000010  00000018  00000000  44332211  11223344
00000020  55667788  00000000  00000000  00000000
00000030  00000040  00000000  00000000  00000000
```

After loading, we can observe data in memory window.

- Q1: What is the starting address of list?
- Q2: Does byte 0x11, 0x22, 0x33, 0x44 and word 0x11223344 be stored the same in memory?
- Q3: Does ARMv7 adopt large endianness or small endianness storage mode?

3. Basic assembly instructions

- Common conditional branch instructions

Instruction	Function
B	Branch
BL	Branch and link
BEQ	Branch if two values are equal
BNE	Branch if two values are not equal
BCC	Branch if Carry = 0
BCS	Branch if Carry = 1
TBB	Table branch byte; forward branch using a table of single byte offset
TBH	Table branch half word; forward branch using a table of half word offset

3. Basic assembly instructions

- Conditional branch instruction demo

```
.global _start
_start:
        MOV R1, #0xa
        MOV R2, #0xb
        CMP R1, #0x1
        BNE else     //go to else if  _?_
then:
        MOV R2, #3     //R2 = 3
        B   endif    //go to endif
else:
        MOV R2, #4   //b = 4
endif:
        LDR R1, =dst
        STR R2, [R1]
dst:
        .asciz “00000000"
```

Read this piece of codes, tell the results of following questions.

- Q1: What condition(s) should be filled in blank _?_ according to the codes?
- Q2: What are the address of lables then, else, endif and dst individually?
- Q3: Which address will be accessed after BNE instruction?
- Q4: What value will be stored in memory?
- Q5: If we want to store another value into memory, how should we change the codes?

3. Basic assembly instructions

- Conditional branch instruction demo -- loop

```
.global _start
_start:
        MOV r0, #0  // i
        MOV r1, #0  // sum

loop:
        CMP r0, #10 // check whether i < 10
        BGE endloop // skip if _?_
        ADD r1, r1, r0  // sum += i
        ADD r0, r0, #1  // i++
        B   loop
endloop:
```

Read this piece of codes, tell the results of following questions.

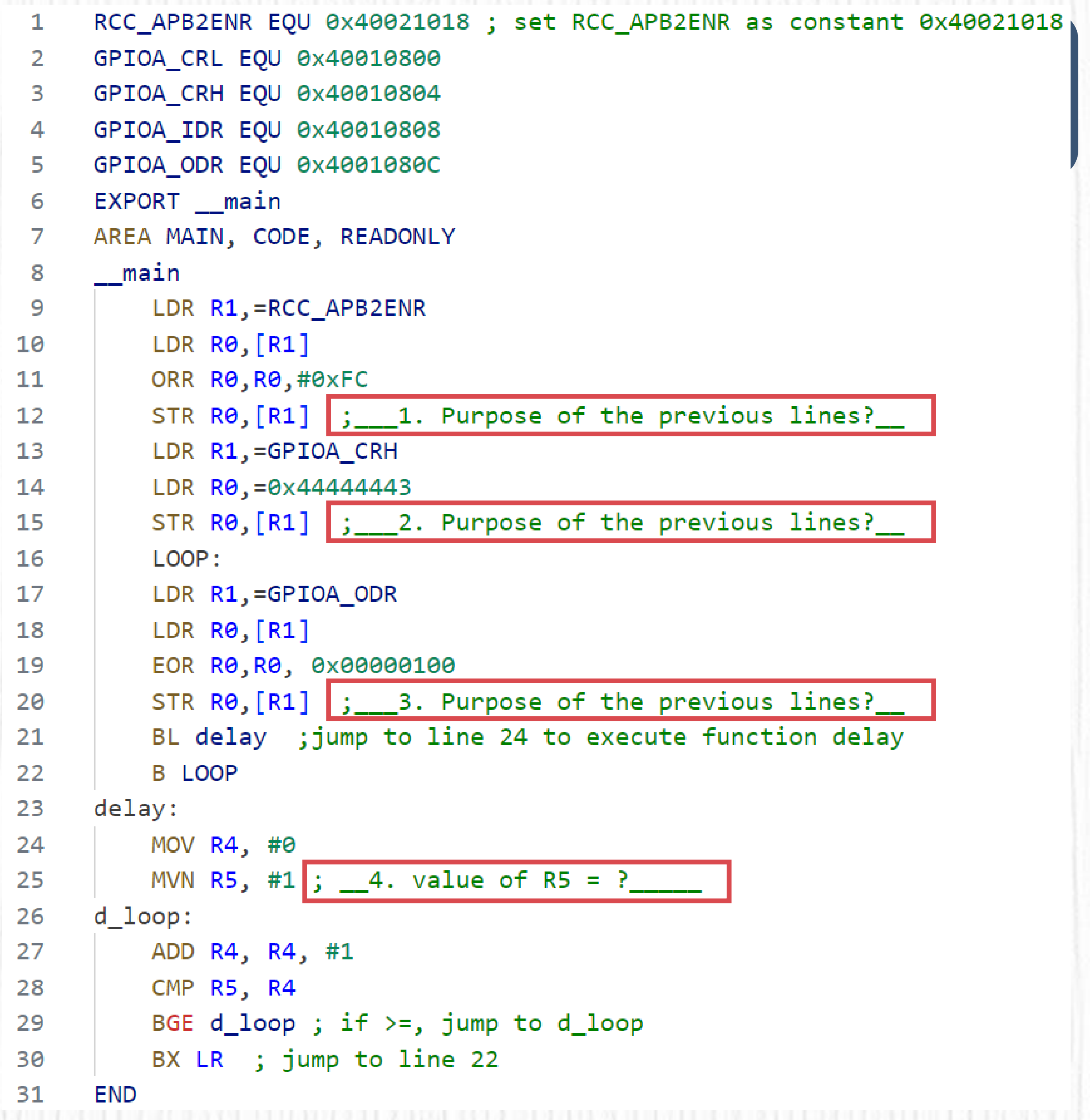
- Q1: What condition(s) should be filled in blank _?_ according to the codes?
- Q2: How many times did the loop code block execute?
- Q3: What’s the function of the codes?
- Q4: What are the final values of R0, R1?

04

Practice

4. Practice

- 1. Write your own codes and test in emulator, which sets C=1, and V=1;
- 2. Read the codes on right
  - Answer the questions 1~4
  - What’s the usage of the code?
  - **NOTES:** this piece of codes **can not** run in CPUlator.



```
1   RCC_APB2ENR EQU 0x40021018 ; set RCC_APB2ENR as constant 0x40021018
2   GPIOA_CRL EQU 0x40010800
3   GPIOA_CRH EQU 0x40010804
4   GPIOA_IDR EQU 0x40010808
5   GPIOA_ODR EQU 0x4001080C
6   EXPORT __main
7   AREA MAIN, CODE, READONLY
8   __main
9       LDR R1,=RCC_APB2ENR
10      LDR R0,[R1]
11      ORR R0,R0,#0xFC
12      STR R0,[R1]  ;___1. Purpose of the previous lines?__
13      LDR R1,=GPIOA_CRH
14      LDR R0,=0x44444443
15      STR R0,[R1]  ;___2. Purpose of the previous lines?__
16      LOOP:
17      LDR R1,=GPIOA_ODR
18      LDR R0,[R1]
19      EOR R0,R0, 0x00000100
20      STR R0,[R1]  ;___3. Purpose of the previous lines?__
21      BL delay  ;jump to line 24 to execute function delay
22      B LOOP
23  delay:
24      MOV R4, #0
25      MVN R5, #1 ; __4. value of R5 = ?_____
26  d_loop:
27      ADD R4, R4, #1
28      CMP R5, R4
29      BGE d_loop ; if >=, jump to d_loop
30      BX LR  ; jump to line 22
31  END
```