EE3002/IM2002

Microprocessors

Tutorial 10

1. Based on the following exception handlers addresses, complete the partial Vector Table with the appropriate instructions at their respective exception address.

Exception	Handler Address
…	…
Software Interrupt	0x40000000
Undefined Instruction	0x3020
Reset	0x40001000 Assume that this 32-bit address is stored at 0xEFC

address	Exception	Instruction
0x10	…	…
0x0C	…	…
0x08	Software interrupt	?
0x04	Undefined instruction	?
0x00	Reset	?

Types of Vector Table Entry

The vector table entries commonly contain branch instructions of one of the following forms

- MOV pc, #immediate

  This MOV copies an immediate value into the PC. It let you span the full address space but at limited alignment (the address must be an 8-bit immediate rotated *right* by an *even* number of bits).
- B <addr>

  This branch instruction provides a branch relative from pc.
  offset = (addr – pc) < +/- 32M
- LDR pc, [pc, #offset]

  This LDR instruction loads the handler address from memory to the pc. The address is an absolute 32-bit value stored close to the vector table. Loading this absolute literal value will result in a slight delay in branching to a specific handler due to the extra memory access. However, you can branch to any address in memory.

Software Interrupt

- Use MOV pc, #0x40000000
- Byte rotation scheme: MOV pc, #04, 4

Undefined instruction exception

- Use B undefinedHandler
- As the handler offset address from current pc is < 32MB (0x1FFFFFFF)
- pc = 0x4 + 0x8 (2 instructions ahead) = 0xC
- (0x3020 – 0xC = 0x3014) <0x1FFFFFF

Reset exception

- Use LDR pc, [pc, #0xEF4]
- Exception address = 0x0
- pc = 0x0 + 0x8 (2 instruction ahead) = 0x8
- Hence offset address = 0xEFC – 0x8 = 0xEF4
- Okay as 0xEFC < 0xFFC (4KB)

2. Name three ways in which FIQ interrupts are handled more quickly than IRQ interrupts.

i. FIQs have a higher priority than IRQs when they occur simultaneously

ii. The FIQ vector is at the top of the vector table allowing you to place the handler there, eliminating the need to change the PC, as you must with the IRQ vector

iii. FIQ mode contains extra FIQ-specific registers (r8-r12) which allows you to not have to push some data on the stack on an FIQ call.

3. How many external interrupt lines does the ARM7TDMI have? If you have eight interrupting devices, how would you handle this?

- Two.
- If more than two sources of interrupt, they will need to be connected to an interrupt controller
- An *Interruption Controller* (IC) connects multiple external interrupts to one of the two ARM interrupt requests.
- It also prioritizes the interrupt sources and signals the ARM processor via the two interrupt lines.
- Sophisticated controller can be programmed to allow an external interrupt source to cause either an IRQ or FIQ exception.

4. Write an SWI handler that accepts the number 0x1234.

   When the handler sees this value, it should reverse the bits in register r9 and store the result in r2.

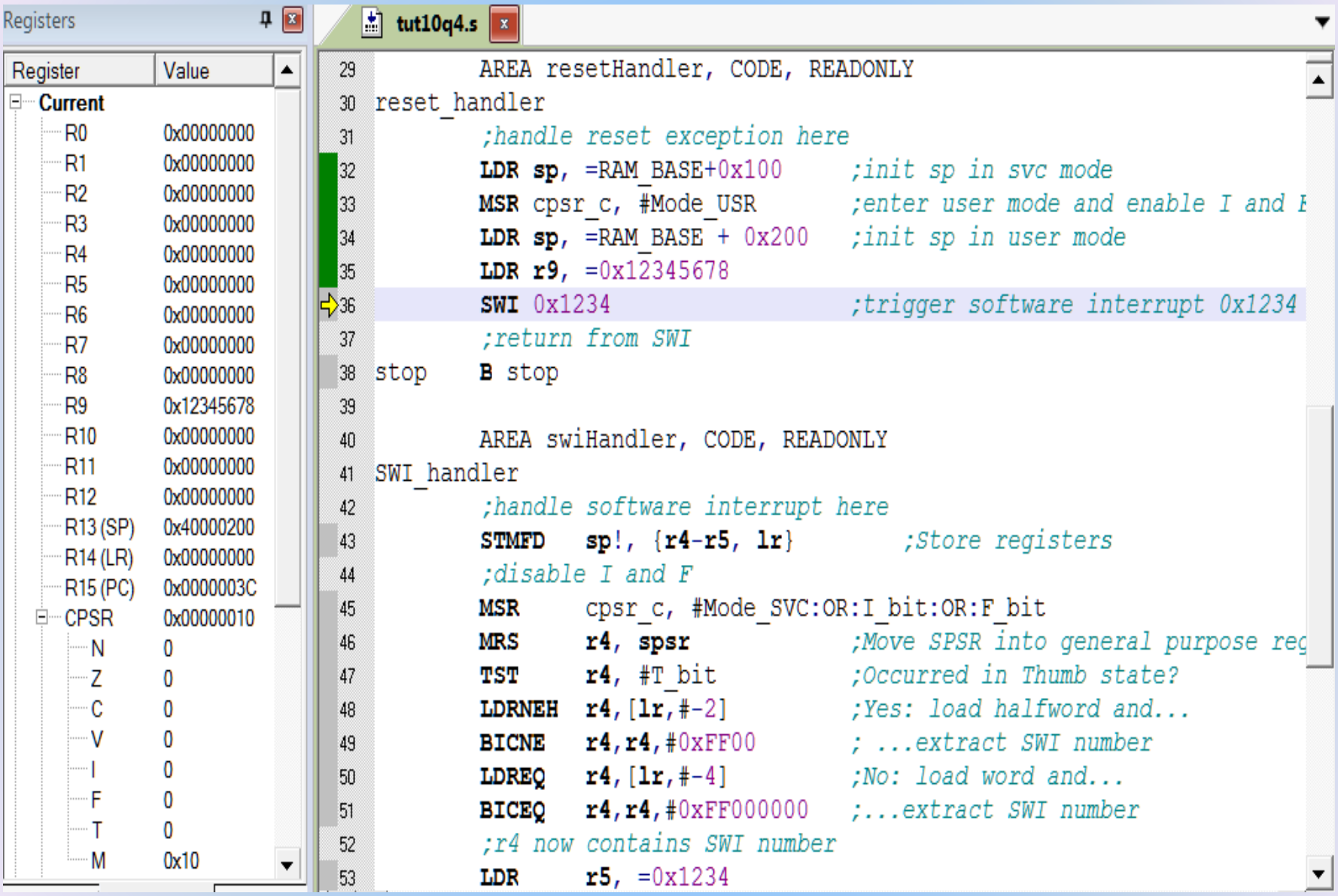
   The SWI exception handler should examine the actual SWI instruction in memory to determine its number.

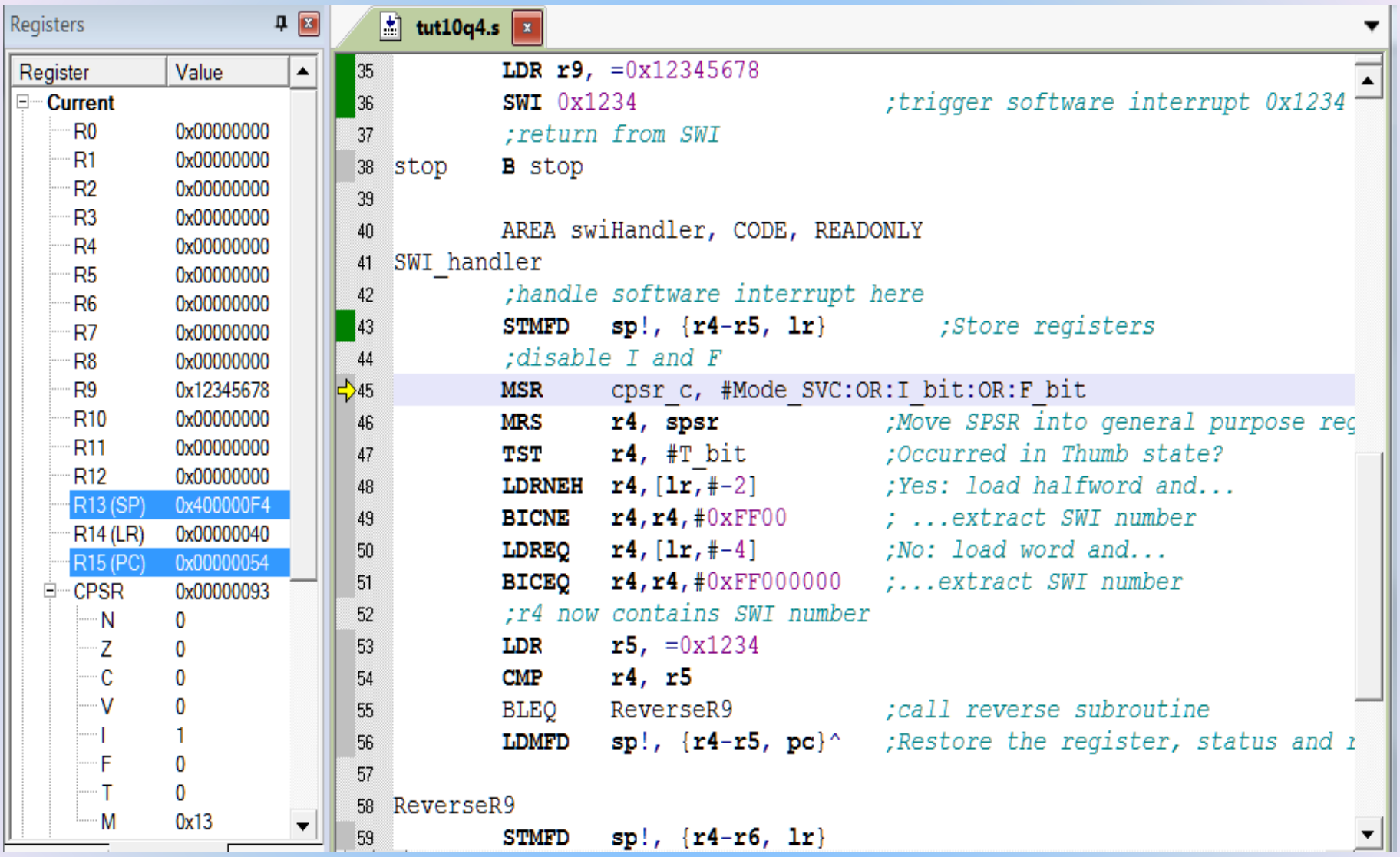
```
;input r9 = 0x12345678
;output r2 = 0x1E6A2C48
RAM_BASE        EQU 0x40000000
T_bit           EQU      0x20           ;Thumb bit of CPSR/SPSR, that is, bit 5
Mode_SVC        EQU 0x13                ;bits for supervisor mode
Mode_USR        EQU 0x10                ;bits for user mode
I_bit           EQU 0x80                ;I bit = 1 disable IRQ
F_bit           EQU 0x40                ;F bit = 1 disable FIQ
                AREA SWIexample, CODE, READONLY
                ENTRY
                ;init exception vector table
                B reset_handler
                B undefined_instruction_handler
                B SWI_handler
                B prefetch_abort_handler
                B data_abort_handler
                SPACE 4
                B IRQ_handler
FIQ_handler
                ;handle FIQ here
```

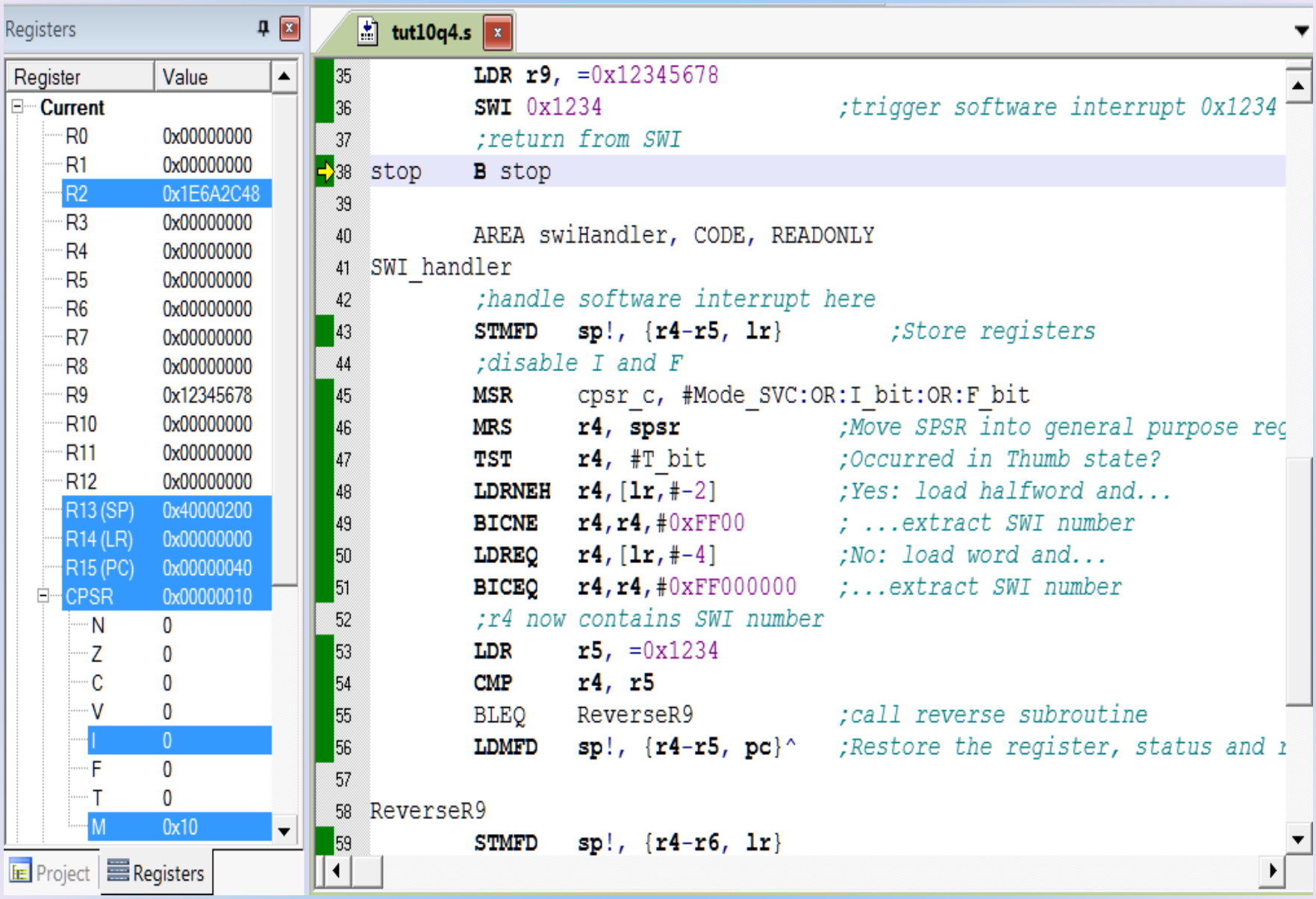
```
undefined_instruction_handler   B undefined_instruction_handler
prefetch_abort_handler          B prefetch_abort_handler
data_abort_handler              B data_abort_handler
IRQ_handler                     B IRQ_handler

                AREA resetHandler, CODE, READONLY
reset_handler
                ;handle reset exception here
                LDR  sp, =RAM_BASE+0x100        ;init sp in svc mode
                MSR cpsr_c, #Mode_USR           ;enter user mode and
                                                ;enable I and F
                LDR  sp, =RAM_BASE + 0x200      ;init sp in user mode
                LDR  r9, =0x12345678
                SWI  0x1234                     ;trigger software interrupt
                                                ;x1234
                                                ;return from SWI
stop            B stop
```

```
                AREA swiHandler, CODE, READONLY
SWI_handler
                ;handle software interrupt here
                STMFD   sp!, {r4-r5, lr}          ;Store registers
                                                  ;disable I and F
                MSR     cpsr_c, #Mode_SVC:OR:I_bit:OR:F_bit
                MRS     r4, spsr                  ;Move SPSR into general
                                                  ;purpose register
                TST     r4, #T_bit                ;Occurred in Thumb state?
                LDRNEH r4,[lr,#-2]                ;Yes: load halfword and...
                BICNE    r4,r4,#0xFF00            ; ...extract SWI number
                LDREQ    r4,[lr,#-4]              ;No: load word and...
                BICEQ    r4,r4,#0xFF000000        ;...extract SWI number
                ;r4 now contains SWI number
                LDR     r5, =0x1234
                CMP      r4, r5
                BLEQ     ReverseR9                ;call reverse subroutine
                LDMFD   sp!, {r4-r5, pc}^         ;Restore the register,
                                                  ;status and return
```

```
ReverseR9
                STMFD   sp!, {r4-r6, lr}
                MOV     r4,     #32             ;r4 is loop counter
                MOV     r5, r9                  ;save a copy of r9
                MOV     r2, #0                  ;result in r2
loop            AND     r6, r5, #1              ;r7=LSB of reg to be
                                                ;reversed
                ADD     r2, r6, r2, LSL #1      ;shift result left 1
                MOV     r5, r5, LSR #1          ;look at new LSB of
                                                ;reg to be reversed
                SUBS    r4, r4, #1              ;decrement counter
                BNE     loop                    ;loop if not finish
                LDMFD           sp!, {r4-r6, pc}        ;return
                END
```


Registers
Register
Value
Current
R0 0x00000000
R1 0x00000000
R2 0x00000000
R3 0x00000000
R4 0x00000000
R5 0x00000000
R6 0x00000000
R7 0x00000000
R8 0x00000000
R9 0x12345678
R10 0x00000000
R11 0x00000000
R12 0x00000000
R13 (SP) 0x40000200
R14 (LR) 0x00000000
R15 (PC) 0x0000003C
CPSR 0x00000010
N 0
Z 0
C 0
V 0
I 0
F 0
T 0
M 0x10
tut10q4.s
29 AREA resetHandler, CODE, READONLY
30 reset_handler
31 ;handle reset exception here
32 LDR sp, =RAM_BASE+0x100 ;init sp in svc mode
33 MSR cpsr_c, #Mode_USR ;enter user mode and enable I and F
34 LDR sp, =RAM_BASE + 0x200 ;init sp in user mode
35 LDR r9, =0x12345678
36 SWI 0x1234 ;trigger software interrupt 0x1234
37 ;return from SWI
38 stop B stop
39
40 AREA swiHandler, CODE, READONLY
41 SWI_handler
42 ;handle software interrupt here
43 STMFD sp!, {r4-r5, lr} ;Store registers
44 ;disable I and F
45 MSR cpsr_c, #Mode_SVC:OR:I_bit:OR:F_bit
46 MRS r4, spsr ;Move SPSR into general purpose reg
47 TST r4, #T_bit ;Occurred in Thumb state?
48 LDRNEH r4,[lr,#-2] ;Yes: load halfword and...
49 BICNE r4,r4,#0xFF00 ; ...extract SWI number
50 LDREQ r4,[lr,#-4] ;No: load word and...
51 BICEQ r4,r4,#0xFF000000 ;...extract SWI number
52 ;r4 now contains SWI number
53 LDR r5, =0x1234


Registers
Register
Value
Current
R0 0x00000000
R1 0x00000000
R2 0x00000000
R3 0x00000000
R4 0x00000000
R5 0x00000000
R6 0x00000000
R7 0x00000000
R8 0x00000000
R9 0x12345678
R10 0x00000000
R11 0x00000000
R12 0x00000000
R13 (SP) 0x400000F4
R14 (LR) 0x00000040
R15 (PC) 0x00000054
CPSR 0x00000093
N 0
Z 0
C 0
V 0
I 1
F 0
T 0
M 0x13
tut10q4.s
35 LDR r9, =0x12345678
36 SWI 0x1234 ;trigger software interrupt 0x1234
37 ;return from SWI
38 stop B stop
39
40 AREA swiHandler, CODE, READONLY
41 SWI_handler
42 ;handle software interrupt here
43 STMFD sp!, {r4-r5, lr} ;Store registers
44 ;disable I and F
45 MSR cpsr_c, #Mode_SVC:OR:I_bit:OR:F_bit
46 MRS r4, spsr ;Move SPSR into general purpose reg
47 TST r4, #T_bit ;Occurred in Thumb state?
48 LDRNEH r4,[lr,#-2] ;Yes: load halfword and...
49 BICNE r4,r4,#0xFF00 ; ...extract SWI number
50 LDREQ r4,[lr,#-4] ;No: load word and...
51 BICEQ r4,r4,#0xFF000000 ;...extract SWI number
52 ;r4 now contains SWI number
53 LDR r5, =0x1234
54 CMP r4, r5
55 BLEQ ReverseR9 ;call reverse subroutine
56 LDMFD sp!, {r4-r5, pc}^ ;Restore the register, status and r
57
58 ReverseR9
59 STMFD sp!, {r4-r6, lr}


Registers
Register
Value
Current
R0 0x00000000
R1 0x00000000
R2 0x1E6A2C48
R3 0x00000000
R4 0x00000000
R5 0x00000000
R6 0x00000000
R7 0x00000000
R8 0x00000000
R9 0x12345678
R10 0x00000000
R11 0x00000000
R12 0x00000000
R13 (SP) 0x40000200
R14 (LR) 0x00000000
R15 (PC) 0x00000040
CPSR 0x00000010
N 0
Z 0
C 0
V 0
I 0
F 0
T 0
M 0x10
Project
Registers
tut10q4.s
35 LDR r9, =0x12345678
36 SWI 0x1234 ;trigger software interrupt 0x1234
37 ;return from SWI
38 stop B stop
39
40 AREA swiHandler, CODE, READONLY
41 SWI_handler
42 ;handle software interrupt here
43 STMFD sp!, {r4-r5, lr} ;Store registers
44 ;disable I and F
45 MSR cpsr_c, #Mode_SVC:OR:I_bit:OR:F_bit
46 MRS r4, spsr ;Move SPSR into general purpose reg
47 TST r4, #T_bit ;Occurred in Thumb state?
48 LDRNEH r4,[lr,#-2] ;Yes: load halfword and...
49 BICNE r4,r4,#0xFF00 ; ...extract SWI number
50 LDREQ r4,[lr,#-4] ;No: load word and...
51 BICEQ r4,r4,#0xFF000000 ;...extract SWI number
52 ;r4 now contains SWI number
53 LDR r5, =0x1234
54 CMP r4, r5
55 BLEQ ReverseR9 ;call reverse subroutine
56 LDMFD sp!, {r4-r5, pc}^ ;Restore the register, status and r
57
58 ReverseR9
59 STMFD sp!, {r4-r6, lr}

5. When handling interrupts, why must the link register be adjusted before returning from the exception?

- Interrupt is only handled upon completing execution of the current executing instruction
- Interrupt handler called after execution is completed
- This implies that pc will be updated before the lr value is calculated
- lr = (pc - 4) actually points to two instructions beyond where interrupt has occurred
- Hence lr need to be adjusted by one instruction on return from interrupt
- Therefore return address, lr = (lr – 4)

Exception	Return address
Reset	None
Data abort	lr-8
Prefetch abort	lr-4
FIQ, IRQ	lr-4
SWI, undefined instruction	lr