Experiment 26
Introduction to the ARM Microprocessor
Submission of Experimental Results

Name: Zheng Sun

ID: 201298762

Group:214

Experimental Results and Comments (Total 90 marks):

Please provide the required screenshots, photos, code, values highlighted in the script according to the following requirements (screenshots should be clear and readable):

1. Code of the Hello World programme (Section 4) after editing (put as text NOT as a screenshot, screenshots of code will receive zero marks) [5 marks]

Code:

```
#include "mbed.h"

DigitalOut myled(LED_GREEN);

Serial pc(USBTX, USBRX);

int main()

{

    int i = 0;

    pc.printf("Hello World!\n");


    while (true) {

        wait(2); // wait a small period of time

        pc.printf("%d \ n\ r ", i); // print the value of variable i

        i++; // increment the variable

        myled = !myled; // toggle a led

    }

}
```

Explanation: Programme after last editing.

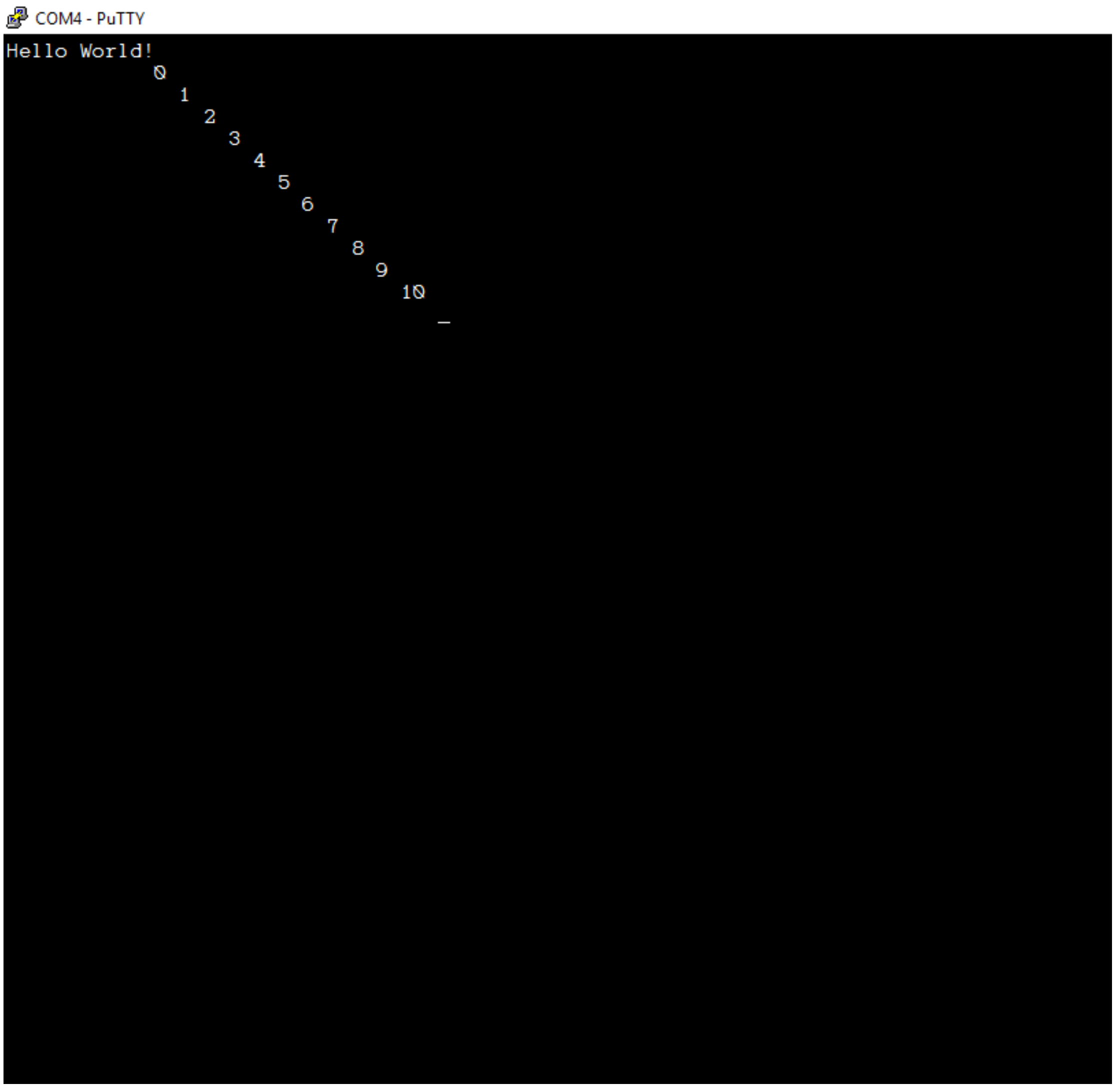
2. Answer to Q1 [5 marks]

Answer: Smallest chosen time: 0.01s (0.02s for a complete cycle), corresponding to 50 Hz.

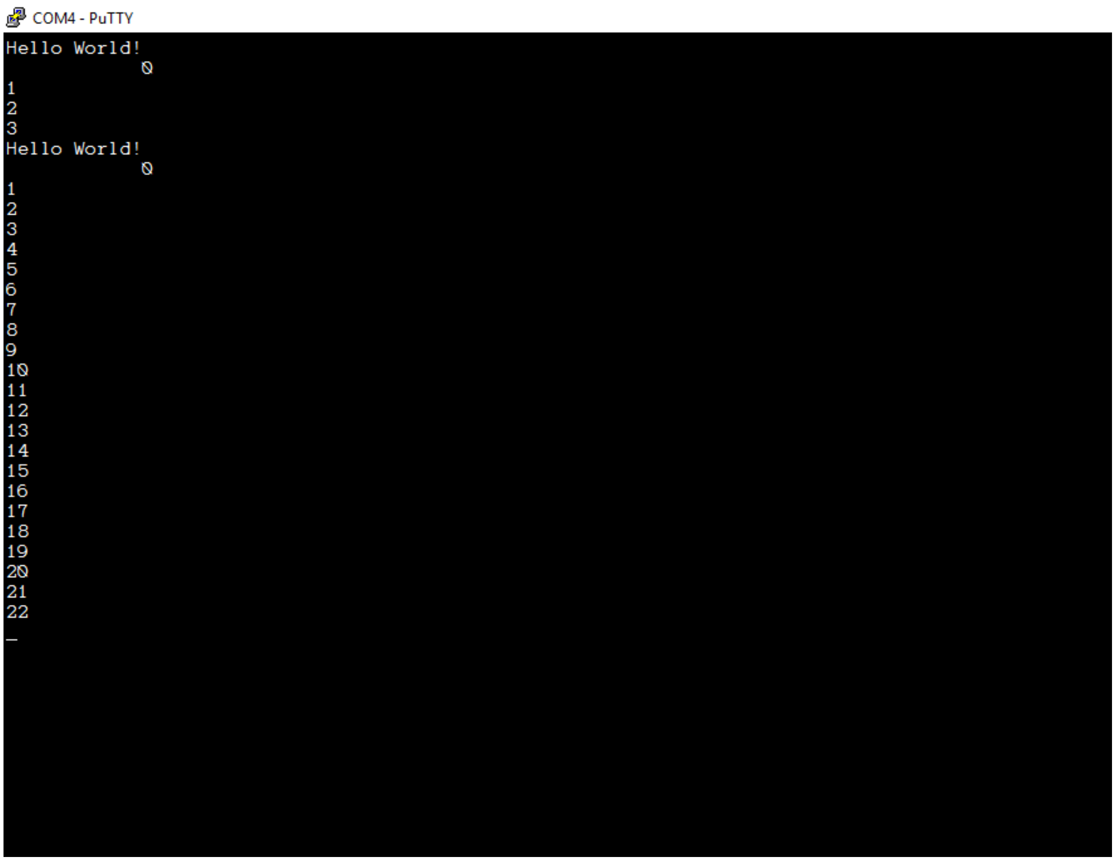
Explanation: It was determined that the smallest time to allow the lab group to see the LEDs flashing was 0.01s, which means a complete flash and black out cycle takes 0.02s, which corresponds to 50Hz.

3. Screenshot of the result of Section 5 part VIII (the output screen) [5 marks]

Screenshots:



```
Hello World!
            0
              1
                2
                  3
                    4
                      5
                        6
                          7
                            8
                              9
                                10
                                   _
```



COM4 - PuTTY

```
Hello World!
              0
1
2
3
Hello World!
              0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
_
```

Comment/Explanation: In the first screenshot of the programme, the message seems to “walk” across the screen, since the “\n” in C is new line instead of newline and carriage return. The printf was changed to use “\n\r” instead of just “\n” in the later version so as to add a carriage return.

4. The modified code of section 6 (put as text NOT as a screenshot, screenshots of code will receive zero marks) [5 marks]

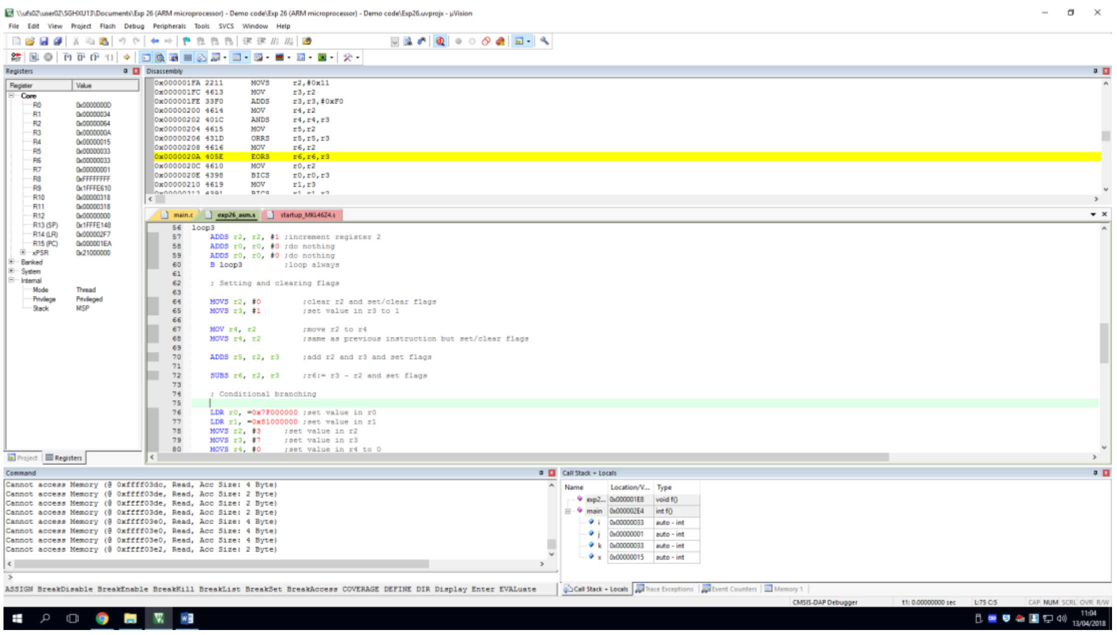
Code:

```
DigitalOut gpo(D0);

DigitalOut led(LED_RED);

DigitalOut led2(LED_GREEN);



    SLCD slcd;

int main()

{
    while (true) {
        float f=tsi.readPercentage();
        slcd.printf("%1.3f",f);
        if(f>0.5){
            led=true;
            }else{
            led=false;
            }
       if(f<0.5){
            led2=true;
            }else{
            led2=false;
            }
        wait(0.2f);
    }
}
```

Explanation: As shown above, the code was modified to set the green led illuminating when the read percentage of the LCD display f > 0.5, corresponding to touching the slider anywhere between the midpoint and the right-hand end; and set the red led illuminating when the read percentage of the LCD display f < 0.5, corresponding to touching the slider anywhere between the midpoint and the left-hand end

5. Screenshot of the result of Section 7 part VIII [3 marks]

Screenshots:



\\ufs02\user02\SGHXU13\Documents\Exp 26 (ARM microprocessor) - Demo code\Exp 26 (ARM microprocessor) - Demo code\Exp26.uvprojx - µVision

Register	Value
Core	
R0	0x0000000D
R1	0x00000034
R2	0x00000064
R3	0x0000000A
R4	0x00000015
R5	0x00000033
R6	0x00000033
R7	0x00000001
R8	0xFFFFFFFF
R9	0x1FFFE610
R10	0x00000318
R11	0x00000318
R12	0x00000000
R13 (SP)	0x1FFFE148
R14 (LR)	0x000002F7
R15 (PC)	0x000001EA
xPSR	0x21000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP

```
0x000001FA 2211      MOVS     r2,#0x11
0x000001FC 4613      MOV      r3,r2
0x000001FE 33F0      ADDS     r3,r3,#0xF0
0x00000200 4614      MOV      r4,r2
0x00000202 401C      ANDS     r4,r4,r3
0x00000204 4615      MOV      r5,r2
0x00000206 431D      ORRS     r5,r5,r3
0x00000208 4616      MOV      r6,r2
0x0000020A 405E      EORS     r6,r6,r3
0x0000020C 4610      MOV      r0,r2
0x0000020E 4398      BICS     r0,r0,r3
0x00000210 4619      MOV      r1,r3
```

```
loop3
    ADDS r2, r2, #1 ;increment register 2
    ADDS r0, r0, #0 ;do nothing
    ADDS r0, r0, #0 ;do nothing
    B loop3         ;loop always

    ; Setting and clearing flags

    MOVS r2, #0          ;clear r2 and set/clear flags
    MOVS r3, #1          ;set value in r3 to 1

    MOV r4, r2           ;move r2 to r4
    MOVS r4, r2          ;same as previous instruction but set/clear flags

    ADDS r5, r2, r3      ;add r2 and r3 and set flags

    SUBS r6, r2, r3      ;r6:= r3 - r2 and set flags

    ; Conditional branching

    LDR r0, =0x7F000000 ;set value in r0
    LDR r1, =0x81000000 ;set value in r1
    MOVS r2, #3     ;set value in r2
    MOVS r3, #7     ;set value in r3
    MOVS r4, #0     ;set value in r4 to 0
```

```
Cannot access Memory (@ 0xffff03dc, Read, Acc Size: 4 Byte)
Cannot access Memory (@ 0xffff03de, Read, Acc Size: 2 Byte)
Cannot access Memory (@ 0xffff03de, Read, Acc Size: 2 Byte)
Cannot access Memory (@ 0xffff03de, Read, Acc Size: 2 Byte)
Cannot access Memory (@ 0xffff03e0, Read, Acc Size: 4 Byte)
Cannot access Memory (@ 0xffff03e0, Read, Acc Size: 4 Byte)
Cannot access Memory (@ 0xffff03e0, Read, Acc Size: 4 Byte)
Cannot access Memory (@ 0xffff03e2, Read, Acc Size: 2 Byte)
>
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE DEFINE DIR Display Enter EVALuate
```

Name	Location/V...	Type
exp2...	0x000001E8	void f()
main	0x000002E4	int f()
i	0x00000033	auto - int
j	0x00000001	auto - int
k	0x00000033	auto - int
x	0x00000015	auto - int

Comment/Explanation: Debugging stared. The important windows of the 'Disassembly' and the 'Registers' and all the important values of the variables were displayed in the view correctly as expected.

6. Answer to Q2 [4 marks]

Answer: The bits from the sixth to the eighth bit (start from the left) of the machine code identifies the register used.

Disassembly: 0x000001EA 220E MOVS r2,#0x0E

Mnemonics: MOVS r2, #14 ;move a number into reigister 2

Machine Code 00100 010 00001110

Disassembly: 0x000001EC 2325 MOVS r3,#0x25

Mnemonics: MOVS r3, #37 ;move a 2nd no. into reg. 3

Machine Code 00100 011 00100101

Explanation: $010_2 = 2$ and $011_2 = 3$ justifies the using of corresponding registers, namely, r2 and r3.

7. Answer to Q3 [4 marks]

Answer: The bits from the ninth to the last bit (start from the left) of the machine code gives the number to be moved into the register.

Disassembly: 0x000001EA 220E MOVS r2,#0x0E

Mnemonics: MOVS r2, #14 ;move a number into reigister 2

Machine Code 00100 010 00001110

Disassembly: 0x000001EC 2325 MOVS r3,#0x25

Mnemonics: MOVS r3, #37 ;move a 2nd no. into reg. 3

Machine Code 00100 011 00100101

Explanation: 00001110_2= 0x0E and 00100101_2 = 0x25 justifies that this part of the machine code gives the number, namely, 14 and 37, to be moved into that register.

8. Answer to Q4 [4 marks]

Answer: result -- R4:0x00000015

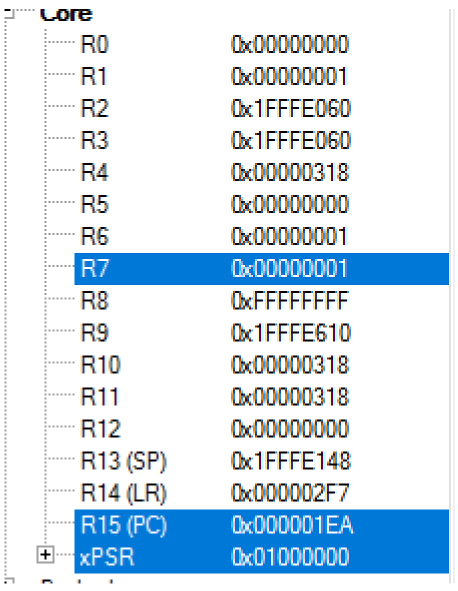
Explanation: The instruction "ADDS r4, r2, #7" put the sum of reg2 and 7 to reg 4, whiche is the addition of 0x0E and 0x07, equalling to 0x15.

9. Answer to Q5 [4 marks]

Answer: The order of the registers in the mnemonics is important.

See below the screenshots of the execution results of corresponding instructions:

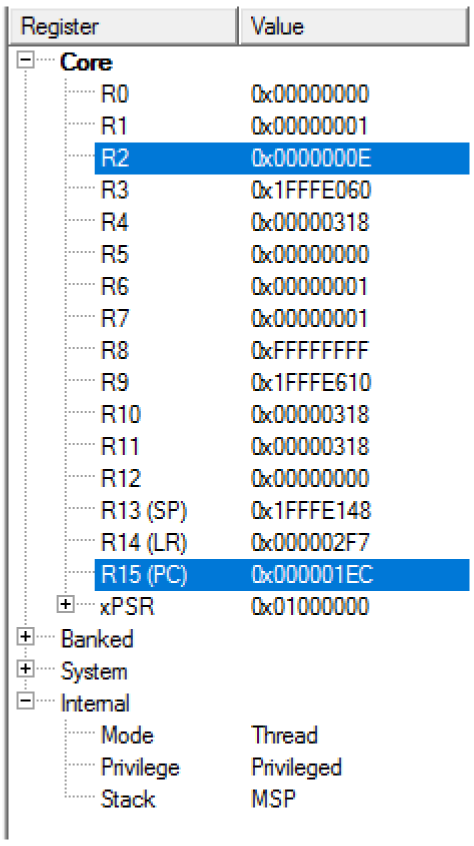
0x000001E8 1847 ADDS r7,r0,r1



Core

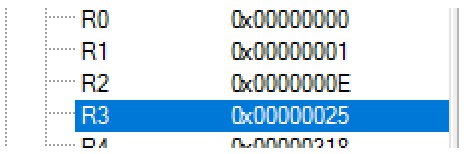
R0	0x00000000
R1	0x00000001
R2	0x1FFFE060
R3	0x1FFFE060
R4	0x00000318
R5	0x00000000
R6	0x00000001
R7	0x00000001
R8	0xFFFFFFFF
R9	0x1FFFE610
R10	0x00000318
R11	0x00000318
R12	0x00000000
R13 (SP)	0x1FFFE148
R14 (LR)	0x000002F7
R15 (PC)	0x000001EA
xPSR	0x01000000

0x000001EA 220E MOVS r2,#0x0E



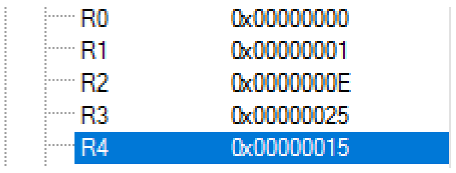
Register	Value
Core	
R0	0x00000000
R1	0x00000001
R2	0x0000000E
R3	0x1FFFE060
R4	0x00000318
R5	0x00000000
R6	0x00000001
R7	0x00000001
R8	0xFFFFFFFF
R9	0x1FFFE610
R10	0x00000318
R11	0x00000318
R12	0x00000000
R13 (SP)	0x1FFFE148
R14 (LR)	0x000002F7
R15 (PC)	0x000001EC
xPSR	0x01000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP

0x000001EC 2325 MOVS r3,#0x25



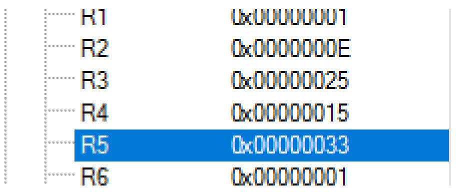
R0	0x00000000
R1	0x00000001
R2	0x0000000E
R3	0x00000025

```
0x000001EE 1DD4        ADDS      r4,r2,#7
```



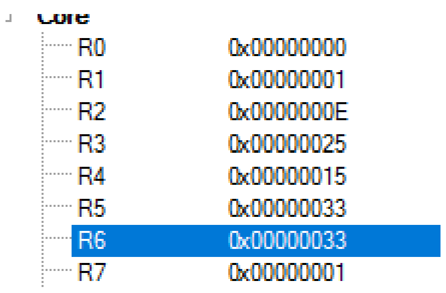
```
R0        0x00000000
R1        0x00000001
R2        0x0000000E
R3        0x00000025
R4        0x00000015
```

```
0x000001F0 18D5        ADDS      r5,r2,r3
```



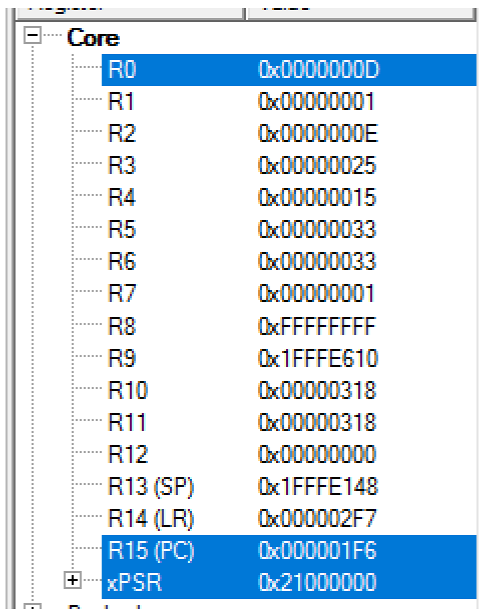
```
R1        0x00000001
R2        0x0000000E
R3        0x00000025
R4        0x00000015
R5        0x00000033
R6        0x00000001
```

```
0x000001F2 189E        ADDS      r6,r3,r2
```



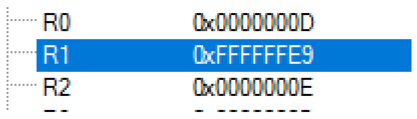
```
Core
R0        0x00000000
R1        0x00000001
R2        0x0000000E
R3        0x00000025
R4        0x00000015
R5        0x00000033
R6        0x00000033
R7        0x00000001
```

```
0x000001F4 1E50        SUBS      r0,r2,#1
```



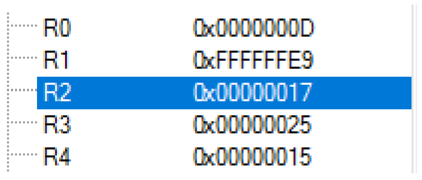
```
Core
R0          0x0000000D
R1          0x00000001
R2          0x0000000E
R3          0x00000025
R4          0x00000015
R5          0x00000033
R6          0x00000033
R7          0x00000001
R8          0xFFFFFFFF
R9          0x1FFFE610
R10         0x00000318
R11         0x00000318
R12         0x00000000
R13 (SP)    0x1FFFE148
R14 (LR)    0x000002F7
R15 (PC)    0x000001F6
xPSR        0x21000000
```

```
0x000001F6 1AD1        SUBS      r1,r2,r3
```



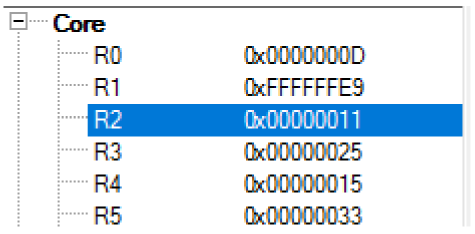
```
R0        0x0000000D
R1        0xFFFFFFE9
R2        0x0000000E
```

0x000001F8 1A9A SUBS r2,r3,r2



```
R0        0x0000000D
R1        0xFFFFFFE9
R2        0x00000017
R3        0x00000025
R4        0x00000015
```

0x000001FA 2211 MOVS r2,#0x11

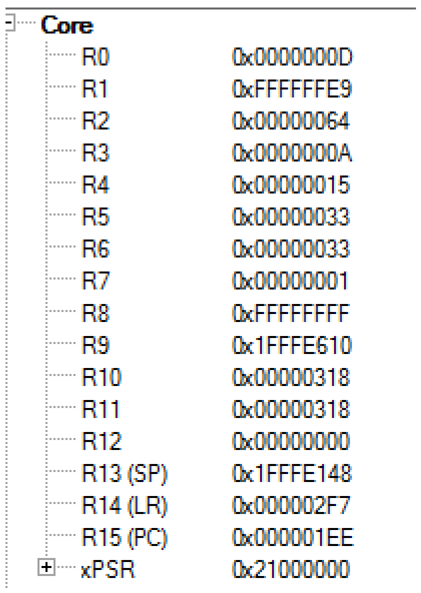


```
Core
  R0        0x0000000D
  R1        0xFFFFFFE9
  R2        0x00000011
  R3        0x00000025
  R4        0x00000015
  R5        0x00000033
```

Explanation: As shown above, the order of the registers in mnemonics distinguishes the registers which stores the operands and the registers for storing results, thus important.

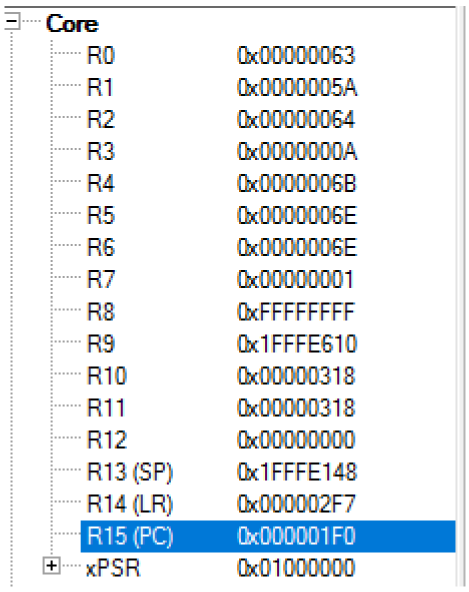
10. Answer to Q6 [4 marks]

Answer: Values are as expected. Register status before excuation:



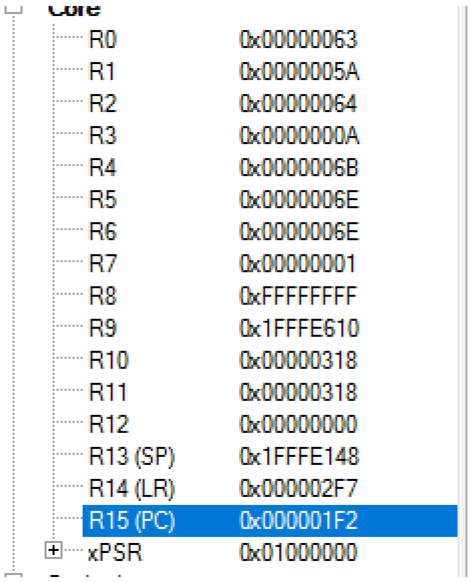
```
Core
  R0        0x0000000D
  R1        0xFFFFFFE9
  R2        0x00000064
  R3        0x0000000A
  R4        0x00000015
  R5        0x00000033
  R6        0x00000033
  R7        0x00000001
  R8        0xFFFFFFFF
  R9        0x1FFFE610
  R10       0x00000318
  R11       0x00000318
  R12       0x00000000
  R13 (SP)  0x1FFFE148
  R14 (LR)  0x000002F7
  R15 (PC)  0x000001EE
  xPSR      0x21000000
```

0x000001EE 1DD4 ADDS r4, r2, #7



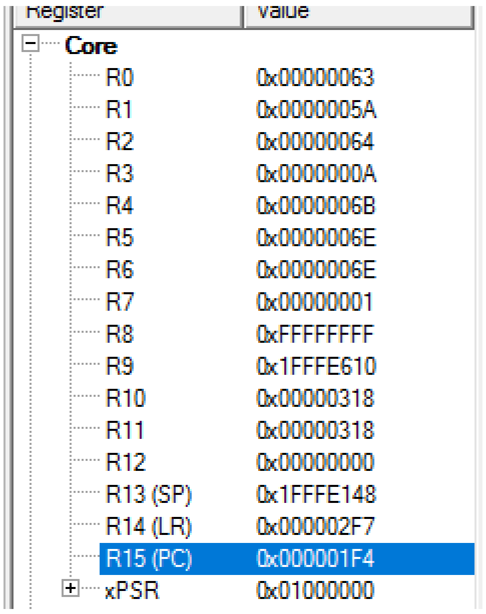
```
Core
  R0         0x00000063
  R1         0x0000005A
  R2         0x00000064
  R3         0x0000000A
  R4         0x0000006B
  R5         0x0000006E
  R6         0x0000006E
  R7         0x00000001
  R8         0xFFFFFFFF
  R9         0x1FFFE610
  R10        0x00000318
  R11        0x00000318
  R12        0x00000000
  R13 (SP)   0x1FFFE148
  R14 (LR)   0x000002F7
  R15 (PC)   0x000001F0
  xPSR       0x01000000
```

0x000001F0 18D5 ADDS r5, r2, r3



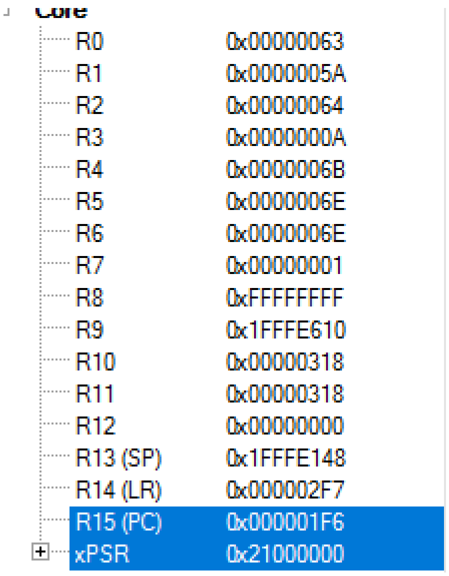
```
Core
  R0         0x00000063
  R1         0x0000005A
  R2         0x00000064
  R3         0x0000000A
  R4         0x0000006B
  R5         0x0000006E
  R6         0x0000006E
  R7         0x00000001
  R8         0xFFFFFFFF
  R9         0x1FFFE610
  R10        0x00000318
  R11        0x00000318
  R12        0x00000000
  R13 (SP)   0x1FFFE148
  R14 (LR)   0x000002F7
  R15 (PC)   0x000001F2
  xPSR       0x01000000
```

0x000001F2 189E ADDS r6, r3, r2



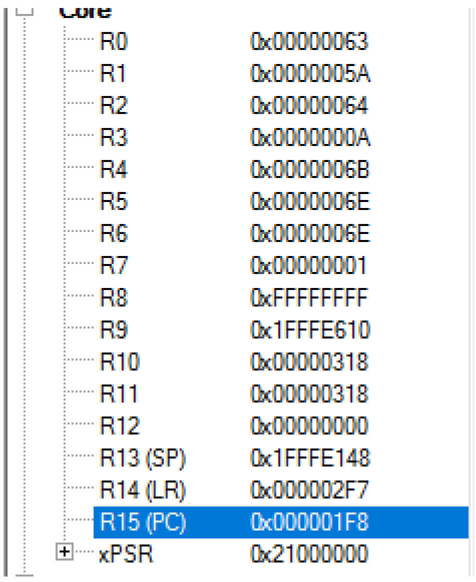
Register	Value
Core	
R0	0x00000063
R1	0x0000005A
R2	0x00000064
R3	0x0000000A
R4	0x0000006B
R5	0x0000006E
R6	0x0000006E
R7	0x00000001
R8	0xFFFFFFFF
R9	0x1FFFE610
R10	0x00000318
R11	0x00000318
R12	0x00000000
R13 (SP)	0x1FFFE148
R14 (LR)	0x000002F7
R15 (PC)	0x000001F4
xPSR	0x01000000

0x000001F4 1E50 SUBS r0, r2, #1



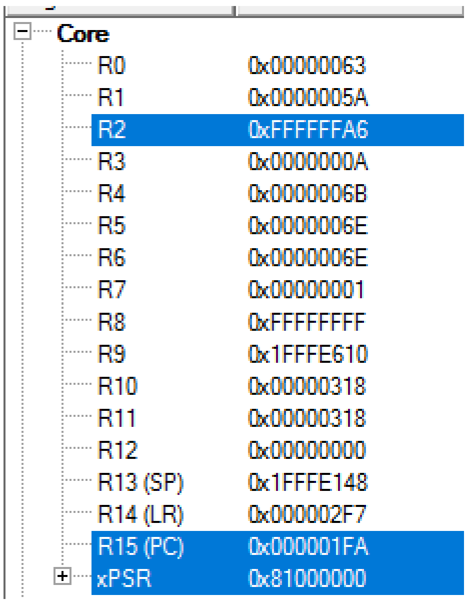
Core	
R0	0x00000063
R1	0x0000005A
R2	0x00000064
R3	0x0000000A
R4	0x0000006B
R5	0x0000006E
R6	0x0000006E
R7	0x00000001
R8	0xFFFFFFFF
R9	0x1FFFE610
R10	0x00000318
R11	0x00000318
R12	0x00000000
R13 (SP)	0x1FFFE148
R14 (LR)	0x000002F7
R15 (PC)	0x000001F6
xPSR	0x21000000

0x000001F6 1AD1 SUBS r1,r2,r3



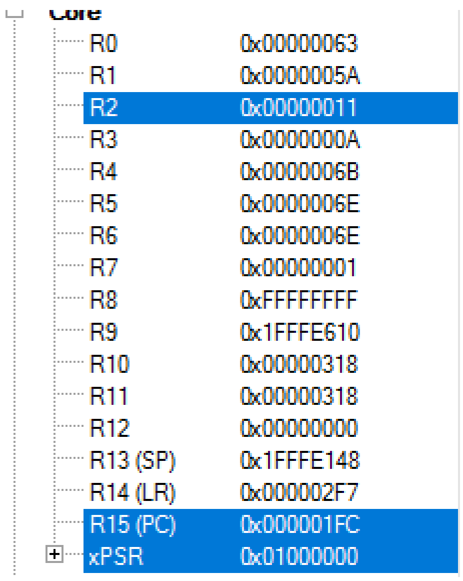
```
Core
  R0        0x00000063
  R1        0x0000005A
  R2        0x00000064
  R3        0x0000000A
  R4        0x0000006B
  R5        0x0000006E
  R6        0x0000006E
  R7        0x00000001
  R8        0xFFFFFFFF
  R9        0x1FFFE610
  R10       0x00000318
  R11       0x00000318
  R12       0x00000000
  R13 (SP)  0x1FFFE148
  R14 (LR)  0x000002F7
  R15 (PC)  0x000001F8
  xPSR      0x21000000
```

```
0x000001F8 1A9A      SUBS      r2,r3,r2
```



```
Core
  R0        0x00000063
  R1        0x0000005A
  R2        0xFFFFFFA6
  R3        0x0000000A
  R4        0x0000006B
  R5        0x0000006E
  R6        0x0000006E
  R7        0x00000001
  R8        0xFFFFFFFF
  R9        0x1FFFE610
  R10       0x00000318
  R11       0x00000318
  R12       0x00000000
  R13 (SP)  0x1FFFE148
  R14 (LR)  0x000002F7
  R15 (PC)  0x000001FA
  xPSR      0x81000000
```

```
0x000001FA 2211      MOVS      r2,#0x11
```



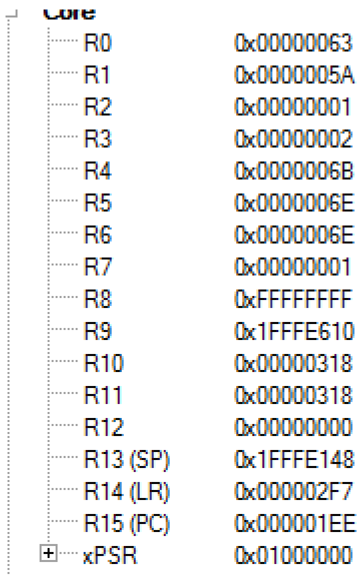
```
Core
  R0          0x00000063
  R1          0x0000005A
  R2          0x00000011
  R3          0x0000000A
  R4          0x0000006B
  R5          0x0000006E
  R6          0x0000006E
  R7          0x00000001
  R8          0xFFFFFFFF
  R9          0x1FFFE610
  R10         0x00000318
  R11         0x00000318
  R12         0x00000000
  R13 (SP)    0x1FFFE148
  R14 (LR)    0x000002F7
  R15 (PC)    0x000001FC
  xPSR        0x01000000
```

Explanation: The results are shown clearly in the screenshots of the single step execution process.

11. Answer to Q7 [4 marks]

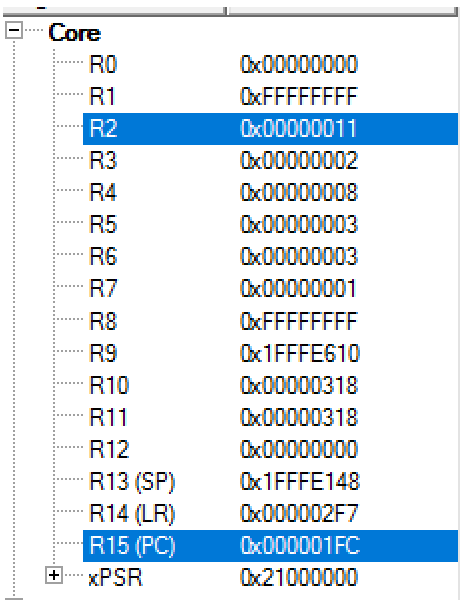
Answer: The values are as expected as shown below in the screenshots of the single step results.

Register status before execution:



```
Core
  R0          0x00000063
  R1          0x0000005A
  R2          0x00000001
  R3          0x00000002
  R4          0x0000006B
  R5          0x0000006E
  R6          0x0000006E
  R7          0x00000001
  R8          0xFFFFFFFF
  R9          0x1FFFE610
  R10         0x00000318
  R11         0x00000318
  R12         0x00000000
  R13 (SP)    0x1FFFE148
  R14 (LR)    0x000002F7
  R15 (PC)    0x000001EE
  xPSR        0x01000000
```

Result:



```
Core
  R0          0x00000000
  R1          0xFFFFFFFF
  R2          0x00000011
  R3          0x00000002
  R4          0x00000008
  R5          0x00000003
  R6          0x00000003
  R7          0x00000001
  R8          0xFFFFFFFF
  R9          0x1FFFE610
  R10         0x00000318
  R11         0x00000318
  R12         0x00000000
  R13 (SP)    0x1FFFE148
  R14 (LR)    0x000002F7
  R15 (PC)    0x000001FC
  xPSR        0x21000000
```

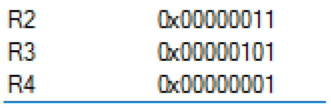
Explanation: The results are shown clearly in the screenshots of the single step execution process.

12. Answer to Q8 [4 marks]

Answer: The logic function AND was performed.

```
0x000001FA 2211    MOVS    r2,#0x11
0x000001FC 4613    MOV     r3,r2
0x000001FE 33F0    ADDS    r3,r3,#0xF0
0x00000200 4614    MOV     r4,r2
0x00000202 401C    ANDS    r4,r4,r3
0x00000204 4615    MOV     r5,r2
```

Results:



```
R2    0x00000011
R3    0x00000101
R4    0x00000001
```

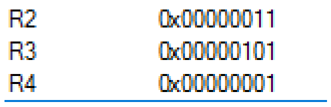
A(R2)	B(R3)	C(R4)
0x00000011	0x00000101	0x00000001

Explanation: C = A AND B, that is R4 = R2 AND R3, that is 0x00000001= 0x00000011 AND 0x00000101.

13. Answer to Q9 [4 marks]

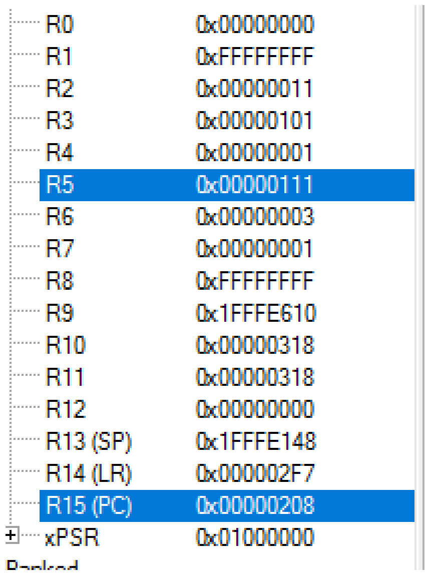
Answer: Yes, besides MOV, AND and SUB, it is also important for functions as BIC, ORR, EOR, etc.

```
0x00000204 4615        MOV        r5,r2
```



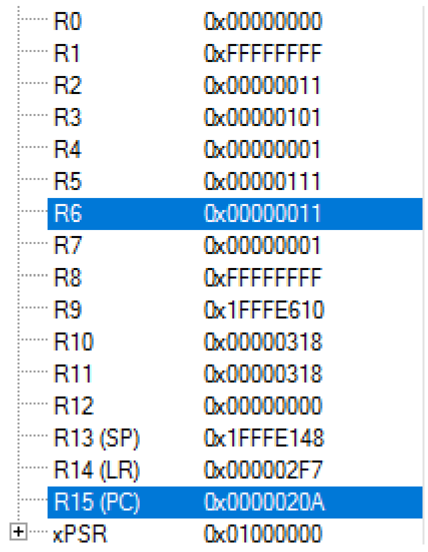
```
R2          0x00000011
R3          0x00000101
R4          0x00000001
```

```
0x00000206 431D        ORRS       r5,r5,r3
```



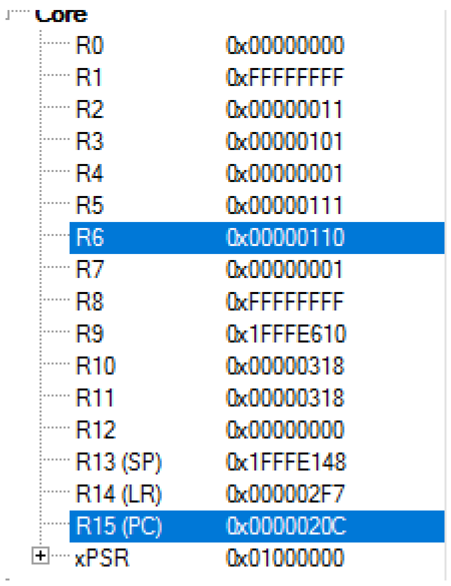
```
R0          0x00000000
R1          0xFFFFFFFF
R2          0x00000011
R3          0x00000101
R4          0x00000001
R5          0x00000111
R6          0x00000003
R7          0x00000001
R8          0xFFFFFFFF
R9          0x1FFFE610
R10         0x00000318
R11         0x00000318
R12         0x00000000
R13 (SP)    0x1FFFE148
R14 (LR)    0x000002F7
R15 (PC)    0x00000208
xPSR        0x01000000
```

```
0x00000208 4616        MOV        r6,r2
```



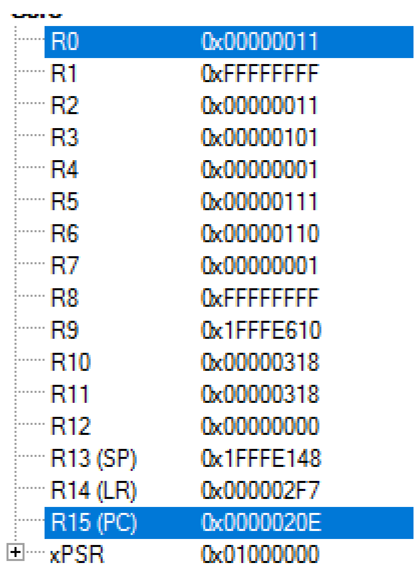
```
R0          0x00000000
R1          0xFFFFFFFF
R2          0x00000011
R3          0x00000101
R4          0x00000001
R5          0x00000111
R6          0x00000011
R7          0x00000001
R8          0xFFFFFFFF
R9          0x1FFFE610
R10         0x00000318
R11         0x00000318
R12         0x00000000
R13 (SP)    0x1FFFE148
R14 (LR)    0x000002F7
R15 (PC)    0x0000020A
xPSR        0x01000000
```

0x0000020A 405E EORS r6,r6,r3



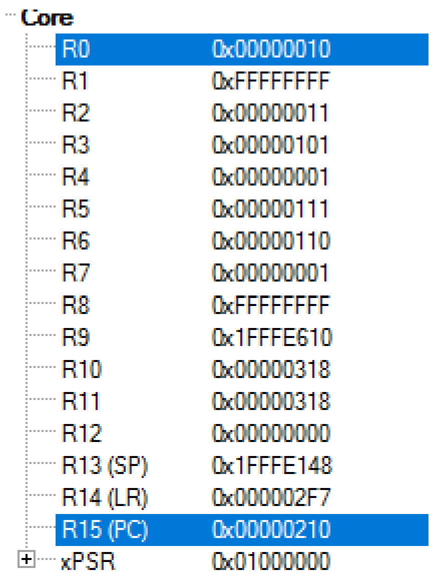
```
Core
R0          0x00000000
R1          0xFFFFFFFF
R2          0x00000011
R3          0x00000101
R4          0x00000001
R5          0x00000111
R6          0x00000110
R7          0x00000001
R8          0xFFFFFFFF
R9          0x1FFFE610
R10         0x00000318
R11         0x00000318
R12         0x00000000
R13 (SP)    0x1FFFE148
R14 (LR)    0x000002F7
R15 (PC)    0x0000020C
xPSR        0x01000000
```

0x0000020C 4610 MOV r0,r2



R0	0x00000011
R1	0xFFFFFFFF
R2	0x00000011
R3	0x00000101
R4	0x00000001
R5	0x00000111
R6	0x00000110
R7	0x00000001
R8	0xFFFFFFFF
R9	0x1FFFE610
R10	0x00000318
R11	0x00000318
R12	0x00000000
R13 (SP)	0x1FFFE148
R14 (LR)	0x000002F7
R15 (PC)	0x0000020E
xPSR	0x01000000

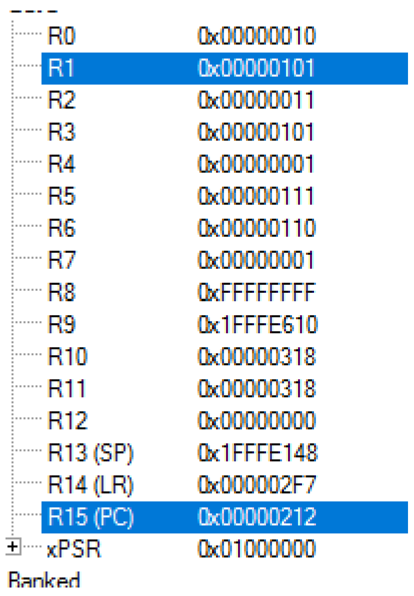
```
0x0000020E 4398        BICS        r0,r0,r3
```



Core

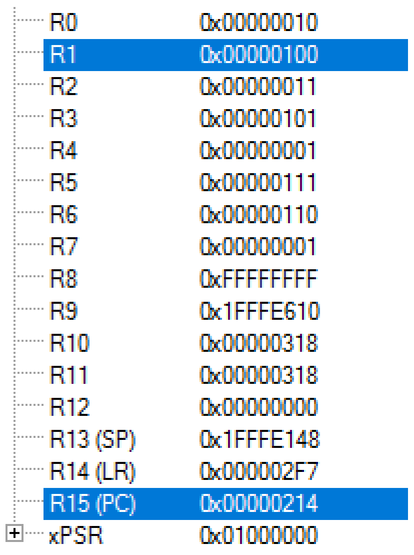
R0	0x00000010
R1	0xFFFFFFFF
R2	0x00000011
R3	0x00000101
R4	0x00000001
R5	0x00000111
R6	0x00000110
R7	0x00000001
R8	0xFFFFFFFF
R9	0x1FFFE610
R10	0x00000318
R11	0x00000318
R12	0x00000000
R13 (SP)	0x1FFFE148
R14 (LR)	0x000002F7
R15 (PC)	0x00000210
xPSR	0x01000000

```
0x00000210 4619        MOV        r1,r3
```



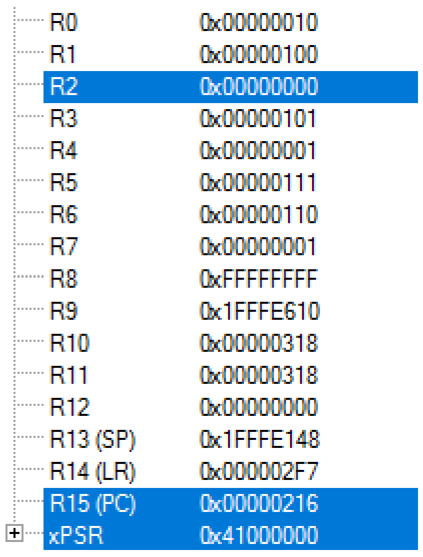
```
R0          0x00000010
R1          0x00000101
R2          0x00000011
R3          0x00000101
R4          0x00000001
R5          0x00000111
R6          0x00000110
R7          0x00000001
R8          0xFFFFFFFF
R9          0x1FFFE610
R10         0x00000318
R11         0x00000318
R12         0x00000000
R13 (SP)    0x1FFFE148
R14 (LR)    0x000002F7
R15 (PC)    0x00000212
xPSR        0x01000000
Banked
```

```
0x00000212 4391      BICS      r1,r1,r2
```



```
R0          0x00000010
R1          0x00000100
R2          0x00000011
R3          0x00000101
R4          0x00000001
R5          0x00000111
R6          0x00000110
R7          0x00000001
R8          0xFFFFFFFF
R9          0x1FFFE610
R10         0x00000318
R11         0x00000318
R12         0x00000000
R13 (SP)    0x1FFFE148
R14 (LR)    0x000002F7
R15 (PC)    0x00000214
xPSR        0x01000000
```

```
0x00000214 2200      MOVS      r2,#0x00
```



```
R0          0x00000010
R1          0x00000100
R2          0x00000000
R3          0x00000101
R4          0x00000001
R5          0x00000111
R6          0x00000110
R7          0x00000001
R8          0xFFFFFFFF
R9          0x1FFFE610
R10         0x00000318
R11         0x00000318
R12         0x00000000
R13 (SP)    0x1FFFE148
R14 (LR)    0x000002F7
R15 (PC)    0x00000216
xPSR        0x41000000
```

Explanation:

As shown above, the order of the registers in mnemonics distinguishes which registers to store the operands and which registers for storing results and the order to execute certain order-sensitive functions, thus important.

14. Answer to Q10 [4 marks]

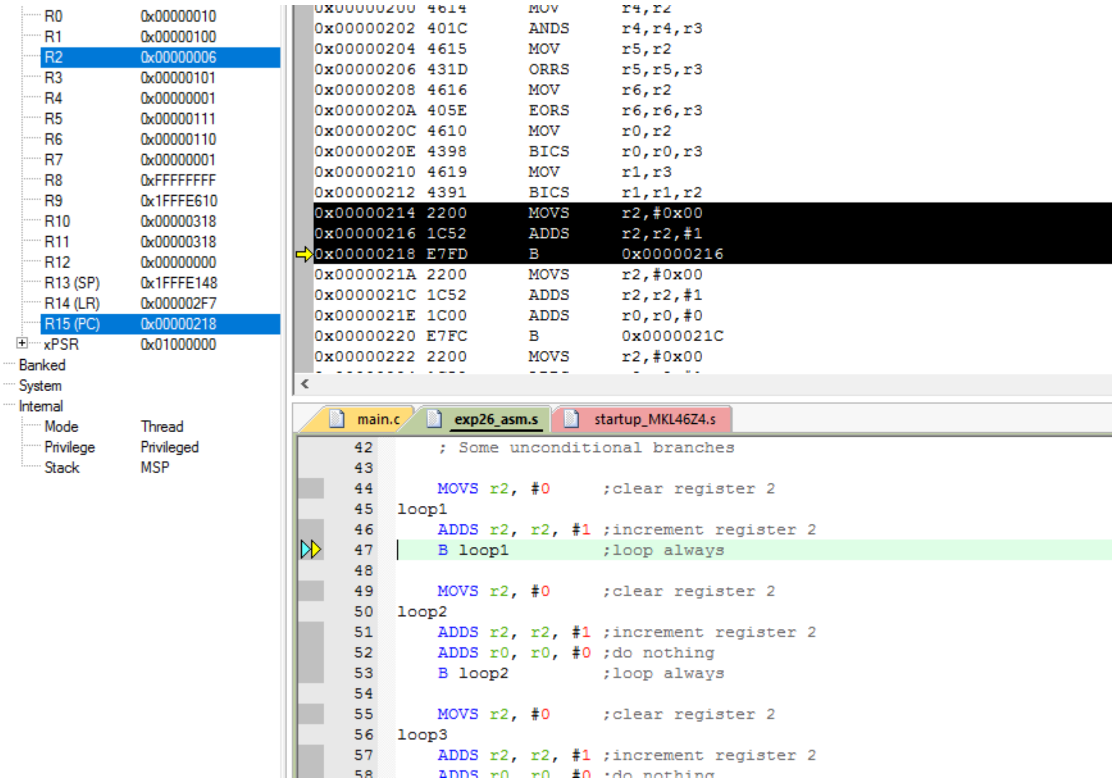
Answer: Branch function was performed, branches to the 0x00000216.

```
0x00000214 2200     MOVS     r2,#0x00
0x00000216 1C52     ADDS     r2,r2,#1
0x00000218 E7FD     B        0x00000216
```

Explanation: The operand for B, which stands for unconditional branch, dictates that the address of the next instruction to be executed was at 0x00000216.

15. Answer to Q11 [5 marks]

Answer: The contents of R2 will be 5 after the instruction at address is executed 5 times (in the screenshots the instruction at 0x00000216 was executed one more time thus R2 being 6).



```
R0          0x00000010
R1          0x00000100
R2          0x00000006
R3          0x00000101
R4          0x00000001
R5          0x00000111
R6          0x00000110
R7          0x00000001
R8          0xFFFFFFFF
R9          0x1FFFE610
R10         0x00000318
R11         0x00000318
R12         0x00000000
R13 (SP)    0x1FFFE148
R14 (LR)    0x000002F7
R15 (PC)    0x00000218
xPSR        0x01000000
Banked
System
Internal
  Mode      Thread
  Privilege Privileged
  Stack     MSP
```

```
0x00000200 4614     MOV      r4,r2
0x00000202 401C     ANDS     r4,r4,r3
0x00000204 4615     MOV      r5,r2
0x00000206 431D     ORRS     r5,r5,r3
0x00000208 4616     MOV      r6,r2
0x0000020A 405E     EORS     r6,r6,r3
0x0000020C 4610     MOV      r0,r2
0x0000020E 4398     BICS     r0,r0,r3
0x00000210 4619     MOV      r1,r3
0x00000212 4391     BICS     r1,r1,r2
0x00000214 2200     MOVS     r2,#0x00
0x00000216 1C52     ADDS     r2,r2,#1
0x00000218 E7FD     B        0x00000216
0x0000021A 2200     MOVS     r2,#0x00
0x0000021C 1C52     ADDS     r2,r2,#1
0x0000021E 1C00     ADDS     r0,r0,#0
0x00000220 E7FC     B        0x0000021C
0x00000222 2200     MOVS     r2,#0x00
```

main.c | exp26_asm.s | startup_MKL46Z4.s

```
    ; Some unconditional branches

    MOVS r2, #0       ;clear register 2
loop1
    ADDS r2, r2, #1 ;increment register 2
    B loop1           ;loop always

    MOVS r2, #0       ;clear register 2
loop2
    ADDS r2, r2, #1 ;increment register 2
    ADDS r0, r0, #0 ;do nothing
    B loop2           ;loop always

    MOVS r2, #0       ;clear register 2
loop3
    ADDS r2, r2, #1 ;increment register 2
```

Explanation: Every time after the "ADD 1 to r2" instruction at 0x00000216 has been executed, the instruction at 0x00000218 branches the program counter back to the "ADD 1 to r2" instruction at 0x00000216, thus when the branch instruction has been executed *exactly* 5 times, the "ADD 1 to r2" instruction at 0x00000216 will also be executed *exactly* 5 times, yielding the contents of R2 being 5.

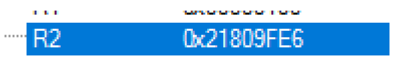
16. Answer to Q12 [4 marks]

Answer: The value held by R2 at 40.19s was approximately twice the value after the initial 20.25s, which means that code block of loop1 has been executed approximately twice times given twice execution time, indicating that the execution of the code block of Loop1is approximately the same.

Value of R2 at time 20.25s: 0x10DE1FD6

```
R2          0x10DE1FD6
```

Value of R2 at time 40.19s: 0x21809FE6


```
R2        0x21809FE6
```

Disassembly for Loop1

```
0x00000214 2200      MOVS      r2,#0x00
0x00000216 1C52      ADDS      r2,r2,#1
0x00000218 E7FD      B         0x00000216
```

Explanation: The execution of the same code block of Loop1is approximately the same.

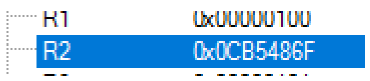
17. The graph of Section 11 (using MS Excel or MATLAB) [6 marks]

Screenshots:

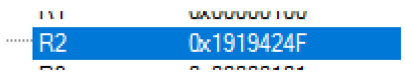
Loop2

```
0x0000021A 2200      MOVS      r2,#0x00
0x0000021C 1C52      ADDS      r2,r2,#1
0x0000021E 1C00      ADDS      r0,r0,#0
0x00000220 E7FC      B         0x0000021C
```

Value of R2 at time 20.22s: 0x0CB5486F


```
R1        0x00000100
R2        0x0CB5486F
```

Value of R2 at time 39.92s: 0x1919424F

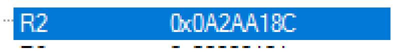

```
R2        0x1919424F
```

Loop3

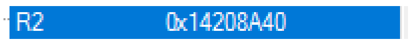
```
0x00000222 2200      MOVS      r2,#0x00
0x00000224 1C52      ADDS      r2,r2,#1
```

```
0x00000226 1C00          ADDS         r0,r0,#0

0x00000228 1C00          ADDS         r0,r0,#0

0x0000022A E7FB            B          0x00000224
```

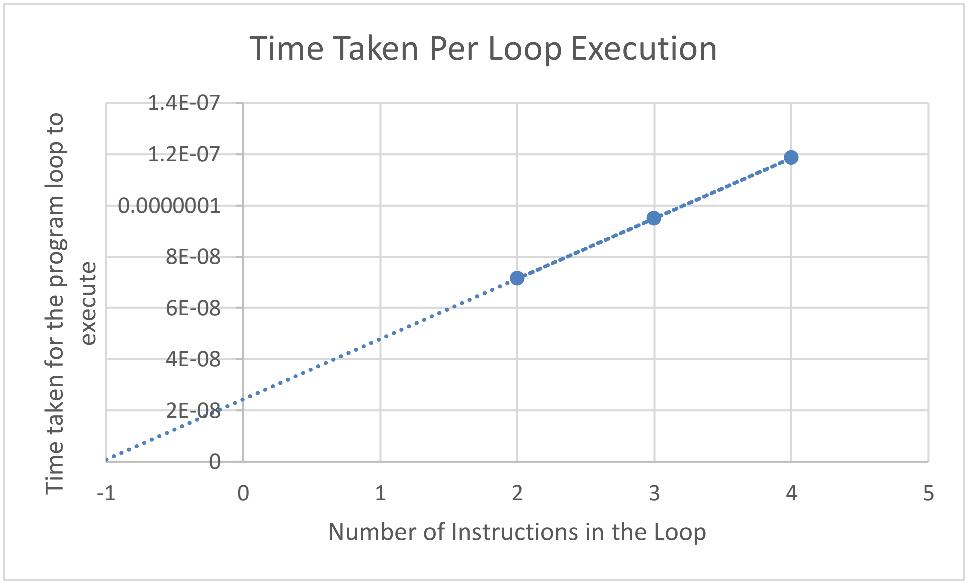
Value of R2 at time 20.22s: 0x0A2AA18C


R2 0x0A2AA18C

Value of R2 at time 40.04s: 0x14208A40


R2 0x14208A40

Comment/Explanation:



As can be seen from the backward extrapolation of the plotted graph, every ADD instruction takes 1 clock cycle to execute and the number of instructions in the loop goes to -1 when time goes to 0, indicating the branch instruction takes twice time as the ADD instruction to execute, that is, 2 clock cycles.

18. Answer to Q13 [4 marks]

Answer: The time taken by one clock cycle is 0.021 µs.

Explanation: It was checked out from the specification that the frequency of the board was 48MHz, which means that one clock cycle takes 0.021 µs.

19. Answer to Q14 [3 marks]

Answer: For the branch instruction, 2 clock cycles are required.

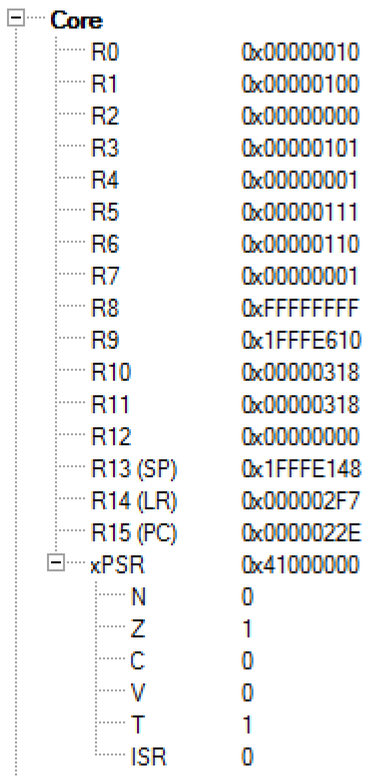
Explanation: As obtained from the specification of the Mbed and verified in the experiment, one branch instruction will take 2 clock cycles.

20. Screenshot of the result of Section 12 [3 marks]

Screenshots:

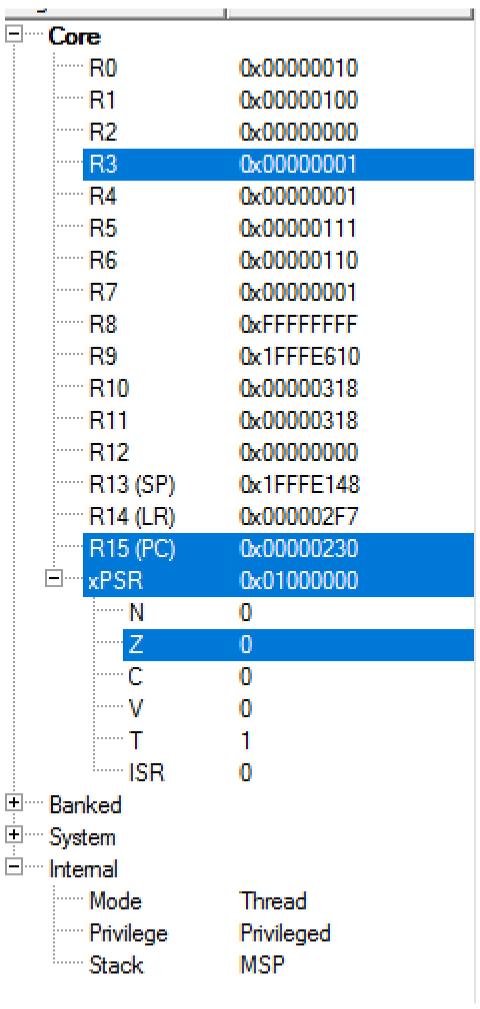
20.1 Results with R2 set as 0x00000000

```
0x0000022C 2200    MOVS    r2,#0x00
```



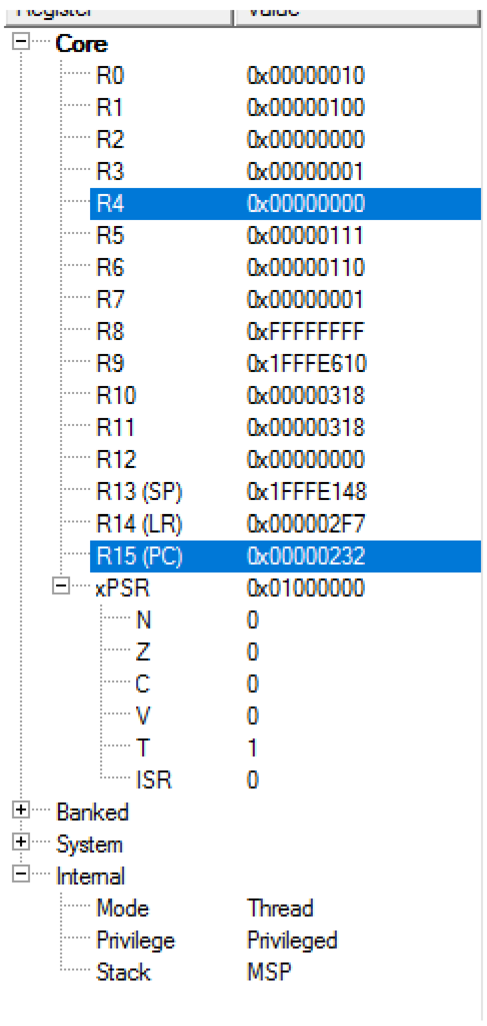
```
Core
  R0          0x00000010
  R1          0x00000100
  R2          0x00000000
  R3          0x00000101
  R4          0x00000001
  R5          0x00000111
  R6          0x00000110
  R7          0x00000001
  R8          0xFFFFFFFF
  R9          0x1FFFE610
  R10         0x00000318
  R11         0x00000318
  R12         0x00000000
  R13 (SP)    0x1FFFE148
  R14 (LR)    0x000002F7
  R15 (PC)    0x0000022E
  xPSR        0x41000000
    N         0
    Z         1
    C         0
    V         0
    T         1
    ISR       0
```

```
0x0000022E 2301    MOVS    r3,#0x01
```



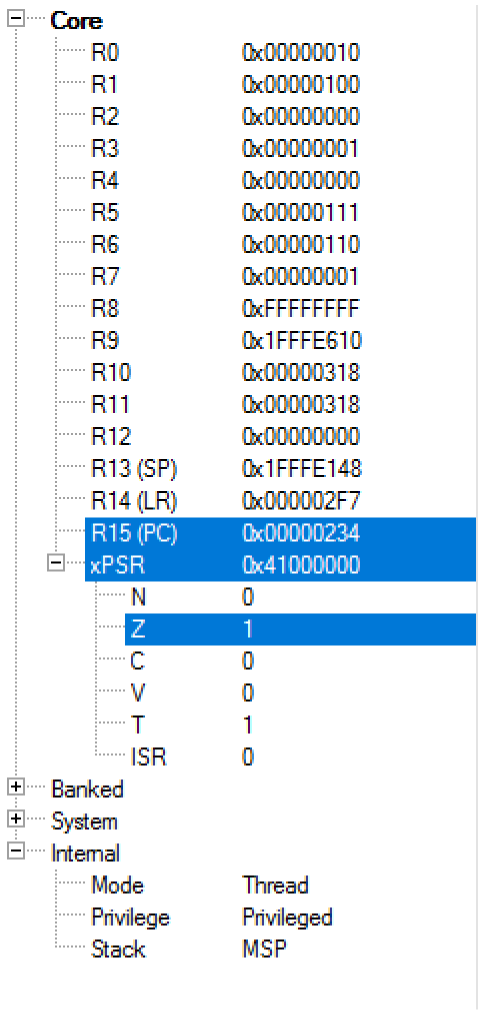
```
Core
    R0          0x00000010
    R1          0x00000100
    R2          0x00000000
    R3          0x00000001
    R4          0x00000001
    R5          0x00000111
    R6          0x00000110
    R7          0x00000001
    R8          0xFFFFFFFF
    R9          0x1FFFE610
    R10         0x00000318
    R11         0x00000318
    R12         0x00000000
    R13 (SP)    0x1FFFE148
    R14 (LR)    0x000002F7
    R15 (PC)    0x00000230
    xPSR        0x01000000
        N       0
        Z       0
        C       0
        V       0
        T       1
        ISR     0
Banked
System
Internal
    Mode        Thread
    Privilege   Privileged
    Stack       MSP
```

```
0x00000230 4614      MOV      r4,r2
```



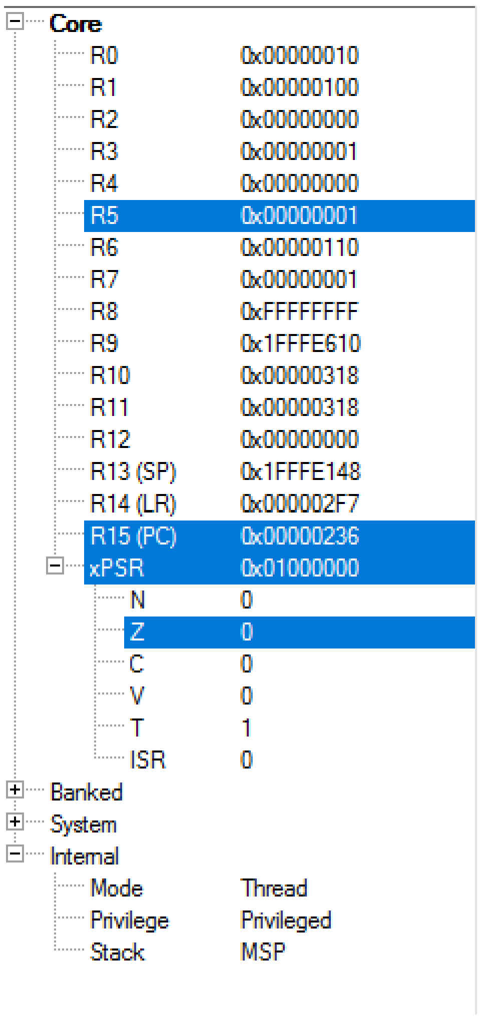
```
Core
  R0          0x00000010
  R1          0x00000100
  R2          0x00000000
  R3          0x00000001
  R4          0x00000000
  R5          0x00000111
  R6          0x00000110
  R7          0x00000001
  R8          0xFFFFFFFF
  R9          0x1FFFE610
  R10         0x00000318
  R11         0x00000318
  R12         0x00000000
  R13 (SP)    0x1FFFE148
  R14 (LR)    0x000002F7
  R15 (PC)    0x00000232
  xPSR        0x01000000
    N         0
    Z         0
    C         0
    V         0
    T         1
    ISR       0
Banked
System
Internal
  Mode        Thread
  Privilege   Privileged
  Stack       MSP
```

```
0x00000232 0014        MOVS        r4,r2
```



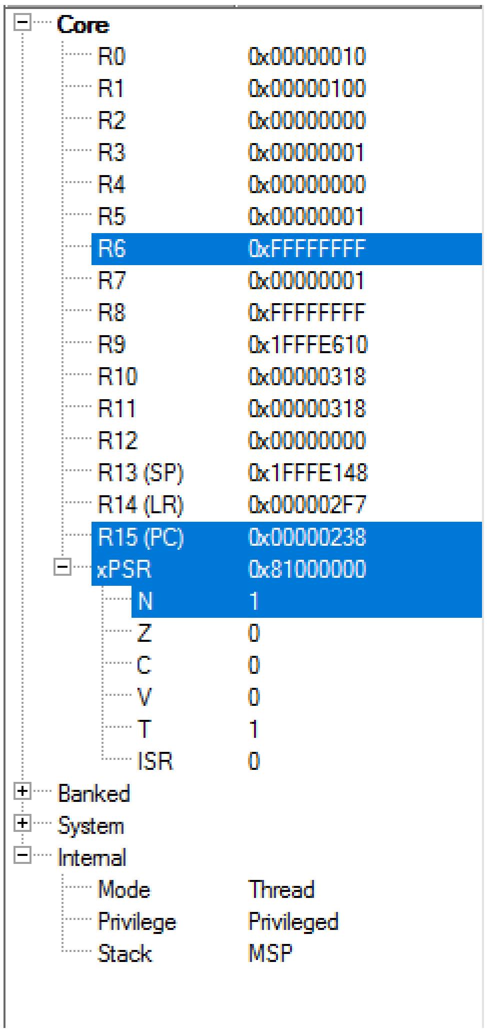
Core	
R0	0x00000010
R1	0x00000100
R2	0x00000000
R3	0x00000001
R4	0x00000000
R5	0x00000111
R6	0x00000110
R7	0x00000001
R8	0xFFFFFFFF
R9	0x1FFFE610
R10	0x00000318
R11	0x00000318
R12	0x00000000
R13 (SP)	0x1FFFE148
R14 (LR)	0x000002F7
R15 (PC)	0x00000234
xPSR	0x41000000
N	0
Z	1
C	0
V	0
T	1
ISR	0
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP

```
0x00000234 18D5          ADDS          r5,r2,r3
```



```
Core
  R0          0x00000010
  R1          0x00000100
  R2          0x00000000
  R3          0x00000001
  R4          0x00000000
  R5          0x00000001
  R6          0x00000110
  R7          0x00000001
  R8          0xFFFFFFFF
  R9          0x1FFFE610
  R10         0x00000318
  R11         0x00000318
  R12         0x00000000
  R13 (SP)    0x1FFFE148
  R14 (LR)    0x000002F7
  R15 (PC)    0x00000236
  xPSR        0x01000000
    N         0
    Z         0
    C         0
    V         0
    T         1
    ISR       0
Banked
System
Internal
  Mode        Thread
  Privilege   Privileged
  Stack       MSP
```

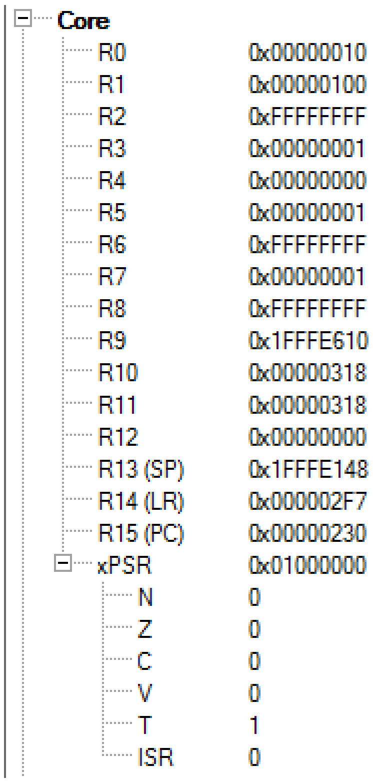
```
0x00000236 1AD6        SUBS       r6,r2,r3
```



```
Core
  R0          0x00000010
  R1          0x00000100
  R2          0x00000000
  R3          0x00000001
  R4          0x00000000
  R5          0x00000001
  R6          0xFFFFFFFF
  R7          0x00000001
  R8          0xFFFFFFFF
  R9          0x1FFFE610
  R10         0x00000318
  R11         0x00000318
  R12         0x00000000
  R13 (SP)    0x1FFFE148
  R14 (LR)    0x000002F7
  R15 (PC)    0x00000238
  xPSR        0x81000000
    N         1
    Z         0
    C         0
    V         0
    T         1
    ISR       0
Banked
System
Internal
  Mode        Thread
  Privilege   Privileged
  Stack       MSP
```

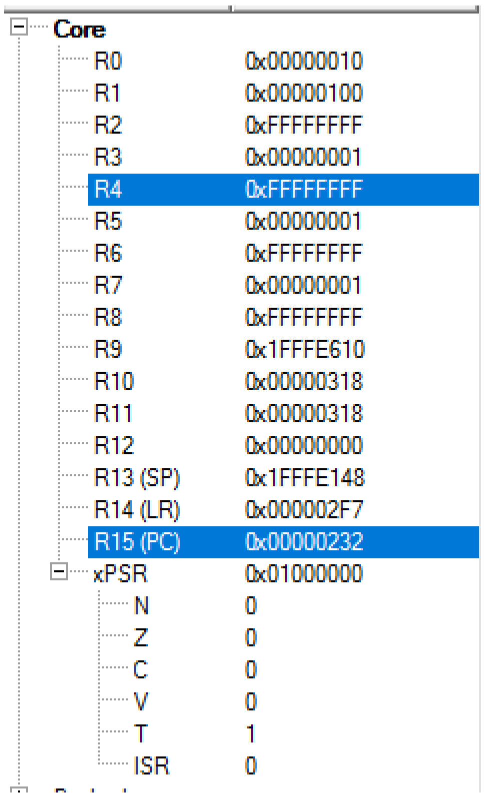
20.2 Results with R2 set to 0xFFFFFFFF

```
0x0000022E 2301        MOVS        r3,#0x01
```



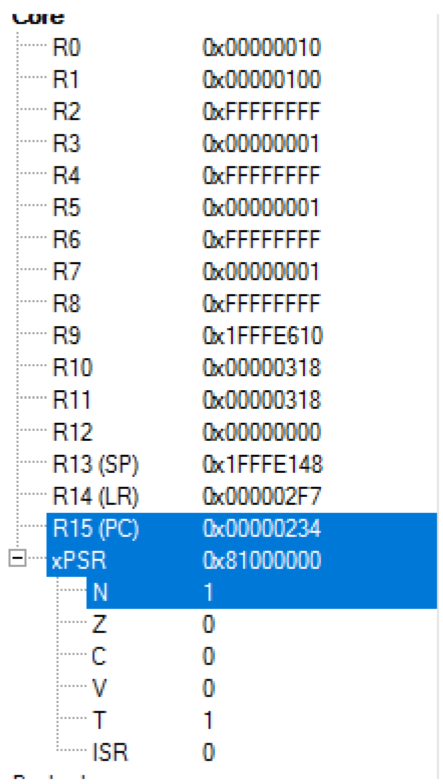
```
Core
  R0         0x00000010
  R1         0x00000100
  R2         0xFFFFFFFF
  R3         0x00000001
  R4         0x00000000
  R5         0x00000001
  R6         0xFFFFFFFF
  R7         0x00000001
  R8         0xFFFFFFFF
  R9         0x1FFFE610
  R10        0x00000318
  R11        0x00000318
  R12        0x00000000
  R13 (SP)   0x1FFFE148
  R14 (LR)   0x000002F7
  R15 (PC)   0x00000230
  xPSR       0x01000000
    N        0
    Z        0
    C        0
    V        0
    T        1
    ISR      0
```

0x00000230 4614 MOV r4,r2



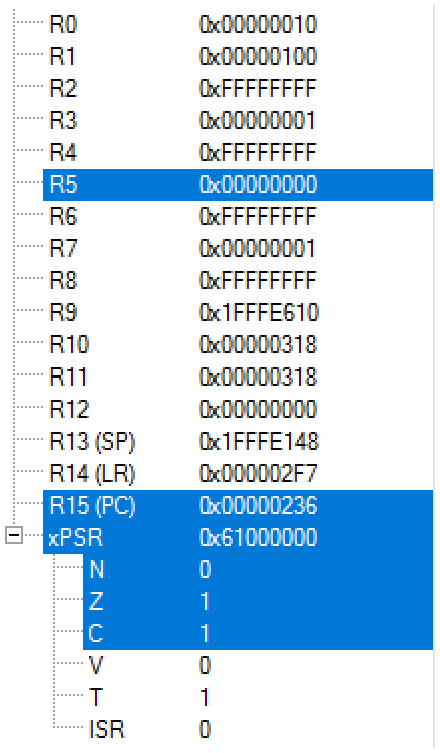
```
Core
  R0         0x00000010
  R1         0x00000100
  R2         0xFFFFFFFF
  R3         0x00000001
  R4         0xFFFFFFFF
  R5         0x00000001
  R6         0xFFFFFFFF
  R7         0x00000001
  R8         0xFFFFFFFF
  R9         0x1FFFE610
  R10        0x00000318
  R11        0x00000318
  R12        0x00000000
  R13 (SP)   0x1FFFE148
  R14 (LR)   0x000002F7
  R15 (PC)   0x00000232
  xPSR       0x01000000
    N        0
    Z        0
    C        0
    V        0
    T        1
    ISR      0
```

0x00000232 0014 MOVS r4,r2



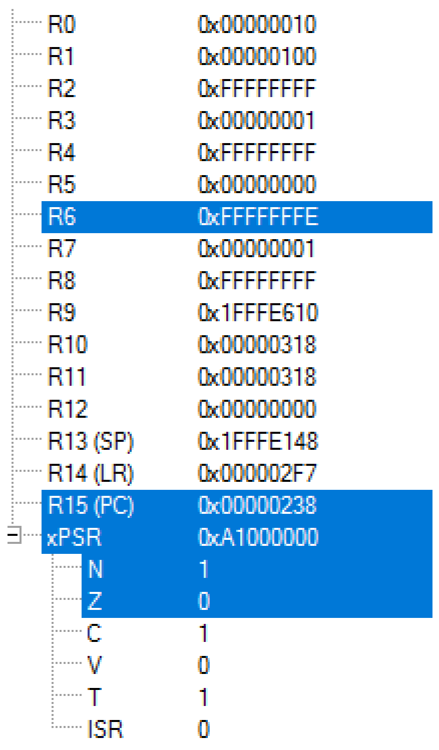
```
Core
  R0          0x00000010
  R1          0x00000100
  R2          0xFFFFFFFF
  R3          0x00000001
  R4          0xFFFFFFFF
  R5          0x00000001
  R6          0xFFFFFFFF
  R7          0x00000001
  R8          0xFFFFFFFF
  R9          0x1FFFE610
  R10         0x00000318
  R11         0x00000318
  R12         0x00000000
  R13 (SP)    0x1FFFE148
  R14 (LR)    0x000002F7
  R15 (PC)    0x00000234
  xPSR        0x81000000
    N         1
    Z         0
    C         0
    V         0
    T         1
    ISR       0
```

0x00000234 18D5 ADDS r5,r2,r3



```
  R0          0x00000010
  R1          0x00000100
  R2          0xFFFFFFFF
  R3          0x00000001
  R4          0xFFFFFFFF
  R5          0x00000000
  R6          0xFFFFFFFF
  R7          0x00000001
  R8          0xFFFFFFFF
  R9          0x1FFFE610
  R10         0x00000318
  R11         0x00000318
  R12         0x00000000
  R13 (SP)    0x1FFFE148
  R14 (LR)    0x000002F7
  R15 (PC)    0x00000236
  xPSR        0x61000000
    N         0
    Z         1
    C         1
    V         0
    T         1
    ISR       0
```

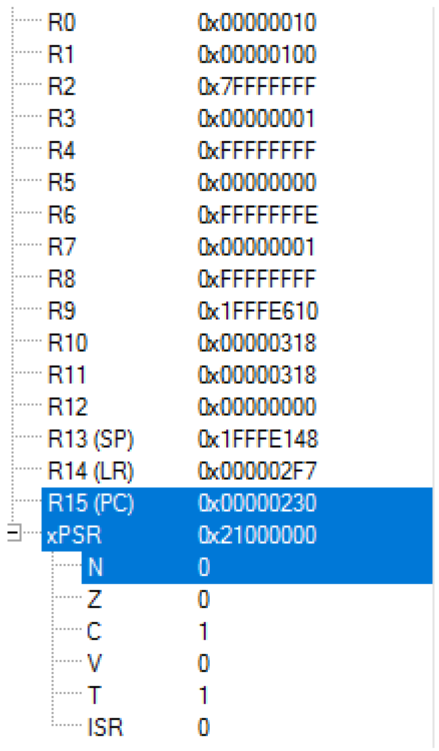
0x00000236 1AD6 SUBS r6,r2,r3



```
R0         0x00000010
R1         0x00000100
R2         0xFFFFFFFF
R3         0x00000001
R4         0xFFFFFFFF
R5         0x00000000
R6         0xFFFFFFFE
R7         0x00000001
R8         0xFFFFFFFF
R9         0x1FFFE610
R10        0x00000318
R11        0x00000318
R12        0x00000000
R13 (SP)   0x1FFFE148
R14 (LR)   0x000002F7
R15 (PC)   0x00000238
xPSR       0xA1000000
  N        1
  Z        0
  C        1
  V        0
  T        1
  ISR      0
```

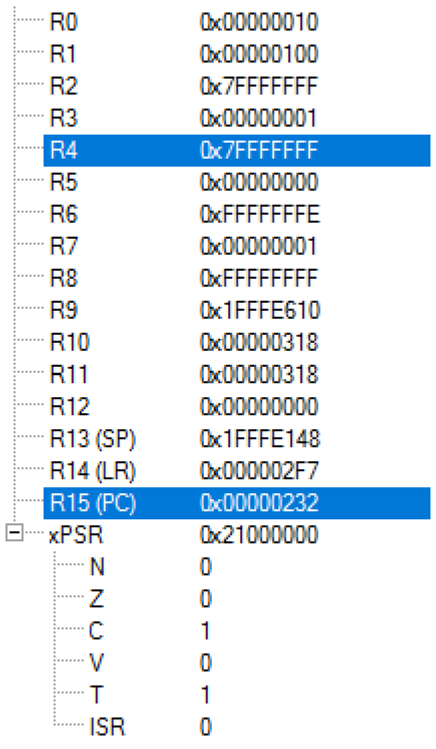
20.3 Results with R3 set to 0x7FFFFFFF

0x0000022E 2301 MOVS r3,#0x01



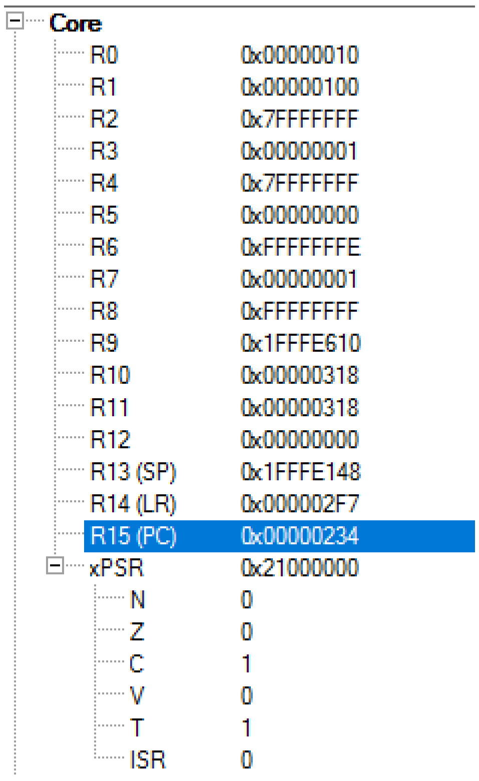
```
R0         0x00000010
R1         0x00000100
R2         0x7FFFFFFF
R3         0x00000001
R4         0xFFFFFFFF
R5         0x00000000
R6         0xFFFFFFFE
R7         0x00000001
R8         0xFFFFFFFF
R9         0x1FFFE610
R10        0x00000318
R11        0x00000318
R12        0x00000000
R13 (SP)   0x1FFFE148
R14 (LR)   0x000002F7
R15 (PC)   0x00000230
xPSR       0x21000000
  N        0
  Z        0
  C        1
  V        0
  T        1
  ISR      0
```

0x00000230 4614 MOV r4,r2



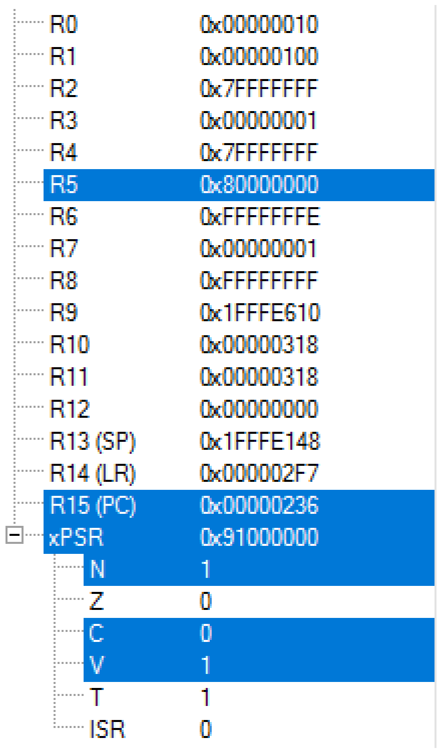
```
R0          0x00000010
R1          0x00000100
R2          0x7FFFFFFF
R3          0x00000001
R4          0x7FFFFFFF
R5          0x00000000
R6          0xFFFFFFFE
R7          0x00000001
R8          0xFFFFFFFF
R9          0x1FFFE610
R10         0x00000318
R11         0x00000318
R12         0x00000000
R13 (SP)    0x1FFFE148
R14 (LR)    0x000002F7
R15 (PC)    0x00000232
xPSR        0x21000000
  N         0
  Z         0
  C         1
  V         0
  T         1
  ISR       0
```

0x00000232 0014   MOVS   r4,r2



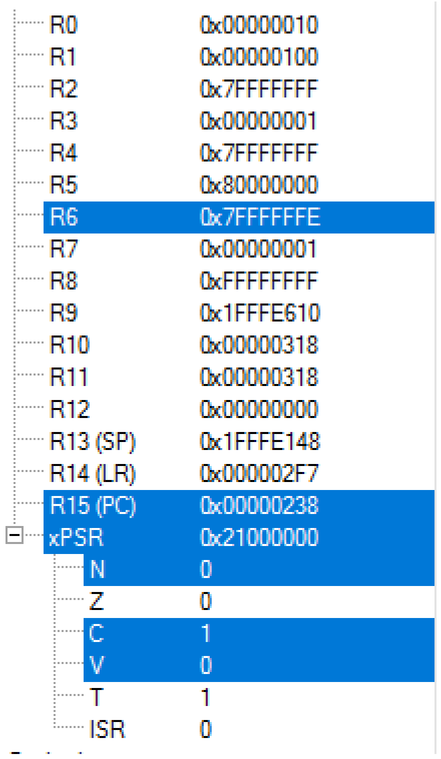
```
Core
  R0          0x00000010
  R1          0x00000100
  R2          0x7FFFFFFF
  R3          0x00000001
  R4          0x7FFFFFFF
  R5          0x00000000
  R6          0xFFFFFFFE
  R7          0x00000001
  R8          0xFFFFFFFF
  R9          0x1FFFE610
  R10         0x00000318
  R11         0x00000318
  R12         0x00000000
  R13 (SP)    0x1FFFE148
  R14 (LR)    0x000002F7
  R15 (PC)    0x00000234
  xPSR        0x21000000
    N         0
    Z         0
    C         1
    V         0
    T         1
    ISR       0
```

0x00000234 18D5   ADDS   r5,r2,r3



```
R0          0x00000010
R1          0x00000100
R2          0x7FFFFFFF
R3          0x00000001
R4          0x7FFFFFFF
R5          0x80000000
R6          0xFFFFFFFE
R7          0x00000001
R8          0xFFFFFFFF
R9          0x1FFFE610
R10         0x00000318
R11         0x00000318
R12         0x00000000
R13 (SP)    0x1FFFE148
R14 (LR)    0x000002F7
R15 (PC)    0x00000236
xPSR        0x91000000
  N         1
  Z         0
  C         0
  V         1
  T         1
  ISR       0
```

0x00000236 1AD6 SUBS r6,r2,r3



```
R0          0x00000010
R1          0x00000100
R2          0x7FFFFFFF
R3          0x00000001
R4          0x7FFFFFFF
R5          0x80000000
R6          0x7FFFFFFE
R7          0x00000001
R8          0xFFFFFFFF
R9          0x1FFFE610
R10         0x00000318
R11         0x00000318
R12         0x00000000
R13 (SP)    0x1FFFE148
R14 (LR)    0x000002F7
R15 (PC)    0x00000238
xPSR        0x21000000
  N         0
  Z         0
  C         1
  V         0
  T         1
  ISR       0
```

Comment/Explanation: Flags were observed to set only with the execution of instructions with a mnemonic ending with an S.

21. Answer to Q15 [3 marks]

Answer: Branch 1, 3, 6, and the last have been executed. At branch 1 BEQ, zero flag was set, thus branch 1 was executed; at branch 3 BCS, the carry flag was set, thus branch 3 was executed; at branch 6 BPL, the negative flag was clear, thus branch 6 was executed; at the last branch BVC, the overflow flag was clear, thus the last branch was executed;

Code:

```
        LDR r0, =0x7F000000    ;set value in r0

        LDR r1, =0x81000000    ;set value in r1

        MOVS r2, #3         ;set value in r2

        MOVS r3, #7         ;set value in r3

        MOVS r4, #0         ;set value in r4 to 0

        MOVS r5, #0         ;set value in r5 to 0

        MOV r8, r5          ;set value in r8 to 0

        MOV r9, r5          ;set value in r9 to 0


        ADDS r6, r1, r0        ;add r1 and r0 and set/clear flags


        BEQ next1              ;branch if zero flag set

        ADD r4, r2          ;add if zero flag clear

next1


        BNE next2              ;branch if zero flag clear

        ADD r4, r3          ;add if zero flag set

next2
```

```
        BCS next3               ;branch if carry flag set

        ADD r5, r2          ;add if carry flag clear

next3


        BCC next4               ;branch if carry flag clear

        ADD r5, r3          ;add if carry flag set

next4


        BMI next5               ;branch if negative flag set

        ADD r8, r2          ;add if negative flag clear

next5


        BPL next6               ;branch if negative flag clear

        ADD r8, r3          ;add if negative flag set

next6


        BVS next7               ;branch if overflow flag set

        ADD r9, r2          ;add if overflow flag clear

next7


        BVC last        ;branch if overflow flag clear

        ADD r9, r3          ;add if overflow flag set

Last
```

Explanation: At branch 1 BEQ, zero flag was set, thus branch 1 was executed; at branch 3 BCC, the carry flag was set, thus branch 3 was executed; at branch 6 BPL, the negative flag was clear, thus branch 6 was executed; at the last branch, the overflow flag was clear, thus the last branch was executed;

22. Answer to Q16 [3 marks]

Answer: Conditional branches 2, 4, 5, 7 were executed as expected. At branch 2 BNE, zero flag was clear, thus branch 2 was executed; at branch 4 BCC, the carry flag was clear, thus branch 4 was executed; at branch 5 BMI, the negative flag was set, thus branch 5 was executed; at branch 7 BVS, the overflow flag was clear, thus the last branch was executed.

R1 was set as 0x00000238, N and V is set

Explanation:

At branch 2 BNE, zero flag was clear, thus branch 2 was executed; at branch 4 BCC, the carry flag was clear, thus branch 4 was executed; at branch 5 BMI, the negative flag was set, thus branch 5 was executed; at branch 7 BVS, the overflow flag was clear, thus the last branch was executed.

Appendix:

Code:

```
  AREA asm_func, CODE, READONLY
    THUMB
    ; Export exp26_asm function location so that C compiler can find it and link
     EXPORT exp26_asm
exp26_asm   PROC
    ; Test routine that does something!

    ADDS r7, r0, r1    ;

    ; Some arithmetic

    MOVS r2, #14          ;move a number into reigister 2
    MOVS r3, #37          ;move a 2nd no. into reg. 3

    ADDS r4, r2, #7 ;put sum of reg.2 and 7 in reg.3

    ADDS r5, r2, r3  ;put sum of r2 and r3 in r5
    ADDS r6, r3, r2  ;reverse the order

    SUBS r0, r2, #1  ;subtract 1 from r2

    SUBS r1, r2, r3   ;subtract r3 from r2
    SUBS r2, r3, r2   ;reverse the order

    ; Some logic

    MOVS r2, #0x0011        ;set bit pattern 0011 (in hex)
    MOV r3, r2
    ADDS r3, #0x00F0   ;set bit pattern 0101 (in hex)

    MOV r4, r2
    ANDS r4, r3       ;AND
    MOV r5, r2
    ORRS r5, r3       ;OR
    MOV r6, r2
    EORS r6, r3       ;exclusive OR
    MOV r0, r2
    BICS r0, r3       ;bit clear
    MOV r1, r3
```

```
        BICS r1, r2         ;bit clear in reverse order


        ; Some unconditional branches


        MOVS r2, #0             ;clear register 2
loop1
        ADDS r2, r2, #1 ;increment register 2
        B loop1             ;loop always


        MOVS r2, #0             ;clear register 2
loop2
        ADDS r2, r2, #1 ;increment register 2
        ADDS r0, r0, #0 ;do nothing
        B loop2             ;loop always


        MOVS r2, #0             ;clear register 2
loop3
        ADDS r2, r2, #1 ;increment register 2
        ADDS r0, r0, #0 ;do nothing
        ADDS r0, r0, #0 ;do nothing
        B loop3             ;loop always


        ; Setting and clearing flags


        MOVS r2, #0                 ;clear r2 and set/clear flags
        MOVS r3, #1                 ;set value in r3 to 1
```

```
	MOV r4, r2		;move r2 to r4
	MOVS r4, r2			;same as previous instruction but set/clear flags


	ADDS r5, r2, r3		;add r2 and r3 and set flags


	SUBS r6, r2, r3		;r6:= r3 - r2 and set flags


	; Conditional branching


	LDR r0, =0x7F000000	;set value in r0
	LDR r1, =0x81000000	;set value in r1
	MOVS r2, #3		;set value in r2
	MOVS r3, #7		;set value in r3
	MOVS r4, #0		;set value in r4 to 0
	MOVS r5, #0		;set value in r5 to 0
	MOV r8, r5	;set value in r8 to 0
	MOV r9, r5	;set value in r9 to 0


	ADDS r6, r1, r0		;add r1 and r0 and set/clear flags


	BEQ next1		;branch if zero flag set
	ADD r4, r2	;add if zero flag clear
next1


	BNE next2		;branch if zero flag clear
	ADD r4, r3	;add if zero flag set
next2
```

```
    BCS next3           ;branch if carry flag set
    ADD r5, r2      ;add if carry flag clear
next3

    BCC next4           ;branch if carry flag clear
    ADD r5, r3      ;add if carry flag set
next4

    BMI next5           ;branch if negative flag set
    ADD r8, r2      ;add if negative flag clear
next5

    BPL next6           ;branch if negative flag clear
    ADD r8, r3      ;add if negative flag set
next6

    BVS next7           ;branch if overflow flag set
    ADD r9, r2      ;add if overflow flag clear
next7

    BVC last        ;branch if overflow flag clear
    ADD r9, r3      ;add if overflow flag set
last

    ; return value passed back to C in r0
    MOVS r0, r7
```

```
; Return to C using link register (Branch indirect using LR - a return)

BX          LR


ENDP


 END
```