Experiment 26
Introduction to the ARM Microprocessor
Experimental Results

Important: Marking of all coursework is anonymous. Do not include your name, student ID number, group number, email or any other personal information in your report or in the name of the file submitted via VITAL. A penalty will be applied to submissions that do not meet this requirement.

Instructions:

- Read this script carefully before attempting the experiment.
- Answer the pre-lab questions before attending the lab (worth 10%).
- Have a look at the document "Exp 26-Submission Template" available on Canvas to have an idea about the experimental results that you have to submit. Experimental results submission is worth 90% of the experiment mark.
- Keep a record of all screenshots, results, answers, comments made and graphs plotted in a logbook. When you submit your work, make sure that all the results, screenshots, etc., are clear and readable; otherwise, you'll lose marks.
- **Important**: When taking snapshots from the screen during the simulation, for each and every snapshot, please **FIRST** use **PrintScreen** under Microsoft Windows, or CMD+SHIFT+3 or **Screenshot** under macOS (which shows the entire desktop including date and time and helps to identify this as uniquely your work) **AND THEN ALSO** use the **Snipping** tool (Microsoft Windows) or **Screenshot** tool (macOS) to show only the relevant part of your simulation. If the entire desktop is not visible in your first screenshot, the associated 'focussed' screenshot will be ignored, and the corresponding section of your report will receive a mark of zero.
- Including screenshots of other people's work is considered academic malpractice and will be penalised in accordance with the University Codes of Practice.

- **All code and text should be included as text and not as screenshots.**

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. Answer to Q1 [4 marks]

Answer:

The second digit is used to identify the register used.

Explanation:

The instructions and machine codes are

2**2**0E MOVS **r2**, #0x0E

2**3**25 MOVS **r3,** #0x25

According to the underlined numbers, it is clear that the second digit of the machine code identifies the register used.

2. Answer to Q2 [4 marks]

Answer:

The third and fourth digits give the number to be used.

Explanation:

The instructions and machine codes are

22**0E** MOVS r2, #0x**0E**

23**25** MOVS r3, #0x**25**

From the underlined numbers, it is can be seen that the last two digits identify the value.

3. Answer to Q3 [4 marks]

Answer:

The value in R4 is 0x00000015

Explanation:

The instruction operates r4 = r2 + $7_{(10)}$. The value hold by R2 before the execution is 0x00000000E. Therefore, the value hold by r4 is 0x00000015.

4. Answer to Q4 [4 marks]

Answer:

The order of the instruction is important.

Explanation:

For examples, by executing the instructions after the addresses 0x00000266 and 0x00000268, the first instruction operates r1 = r2 – r3, while the other instruction operates r2 = r3 – r2. If change the order, r1 is equal to r3 – r2 and r2 is equal to r2 – r3. Those different instructions can execute different results. Therefore, it is important to check the order of the instructions.

5. Answer to Q5 [4 marks]

Answer:

These values are expected.

Explanation:

The initial value in R2 is 0x00000064, while for R3 is 0x0000000A. The following shows the instructions executed form 0x0000025E to 0x0000026A.

r4 = r2 + $7_{(10)}$ = 0x0000006B

r5 = r2 + r3 = 0x0000006E

r0 = r2 - $1_{(10)}$ = 0x00000063

r1 = r2 – r3 = 0x0000005A

r2 = r3 – r2 = 0XFFFFFFA6

It can be seen that R3 does not change. R2 changed to 0xFFFFFFA6.

6. Answer to Q6 [4 marks]

Answer:

These values are expected.

Explanation:

The initial value in r2 is 0x000000CD, while for r3 is 0x000000AB. The following shows the instructions executed form 0x0000025E to 0x0000026A.

r4 = r2 + $7_{(10)}$ = 0x000000D4

r5 = r2 + r3 = 0x00000178

r0 = r2 - $1_{(10)}$ = 0x000000CC

r1 = r2 – r3 = 0x00000022

r2 = r3 – r2 = 0xFFFFFFDE.

It is similar to Q5, the value held by R3 does not change. The value held by R2 changed.

7. Answer to Q7 [4 marks]

Answer:

AND logic function

Explanation:

The value in r2 is 0x00000011, while the value in r3 is 0x00000101. The result in r4 is 0x00000001.

Consider the last four digits,

Register	4^{th} digit	3^{rd} digit	2^{nd} digit	1^{st} digit
R2	0	0	1	1
R3	0	1	0	1
R4	0	0	0	1

Therefore, it is an AND logic.

8. Answer to Q8 [4 marks]

Answer:

The order is not important for other functions.

Explanation:

For AND, ORR and EOR, they consider the combination of each digit for the two values rather than the order.

For example,

ORRS r5, r5, r3 executes r5 = r5 or r3.

ORRS r5, r3, r5 executes r5 = r5 or r3.

These two instructions obtained the same result.

9. Answer to Q9 [4 marks]

Answer:

The program counter moves to the address at 0x00000288. Then, jump back to 0x000000286.

Explanation:

The instruction and address are

0x0000286 ADDS r2, r2, #1;

0x0000288 B 0x00000286;

For the branch instruction, it means go back to the address 0x00000286. Therefore, we can see the program complied in this loop.

10. Answer to Q10 [5 marks]

Answer:

The value is R2 changes from 0x00000000 to 0x00000005 after five cycles.

Explanation:

Register R2 works like a counter. After each time the instruction branch instruction is executed, the value in R2 will plus 1. Thus, after 5 cycles, the value is 0x00000005.

11. Answer to Q11 [4 marks]

Answer:

That is because the program complies at a certain speed. Therefore, the value hold in R2 after 40s shall be approximately twice the value in R2 after 20s.

Explanation:

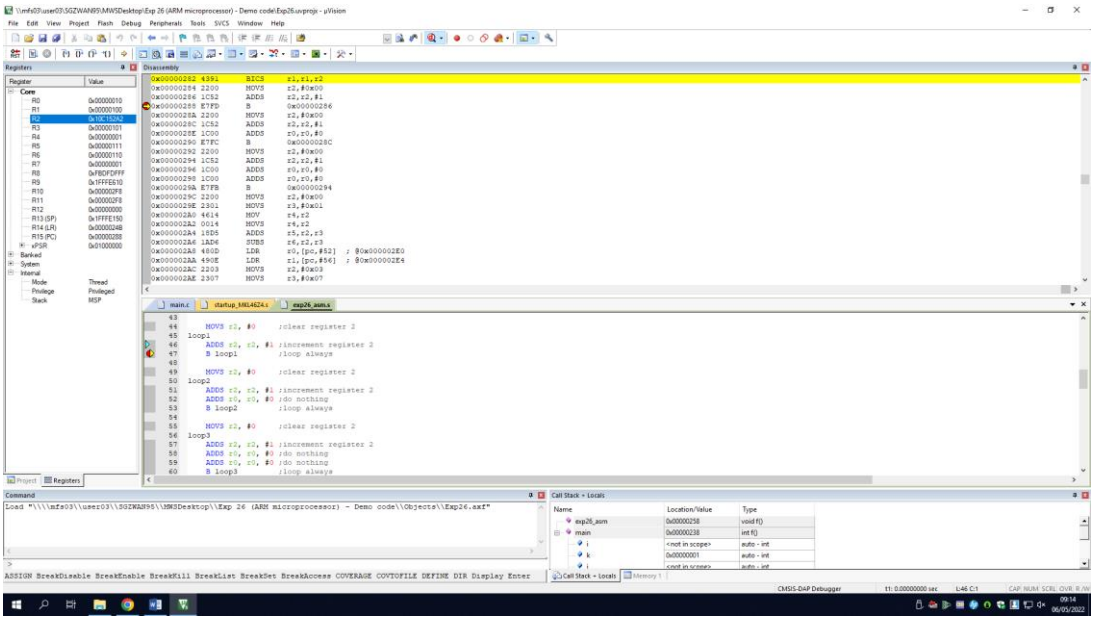


Figure 1: R_2 value

As Figure 1 shows, the value in R_2 is 0x10C152A2 after 20.15s, which is $281105058_{(10)}$.

When the time increases to 40.35s. The value hold in R_2 is 0x21C6D323, which is $566678307_{(10)}$. Figure 2 shows the result.

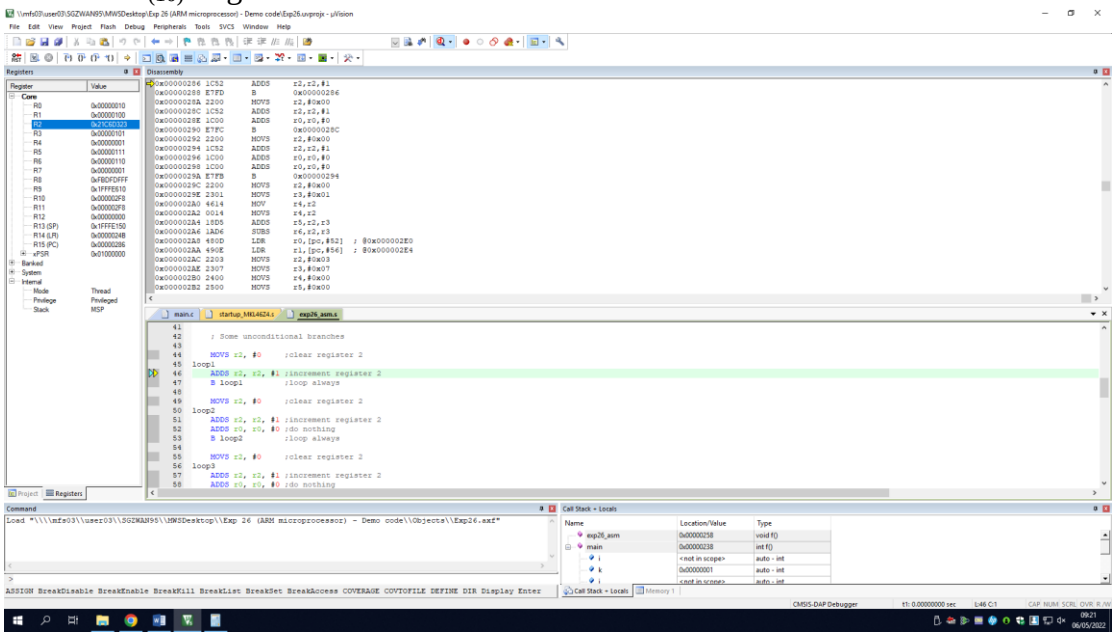


Figure 2: R_2 value

Thus, it can be concluded that the program complies at a certain speed and the value should be twice.

12. The graph of Section 9 (using MS Excel or MATLAB) [6 marks]

Screenshot (PrintScreen or Screenshot):

When counter is set to 0x0000028A

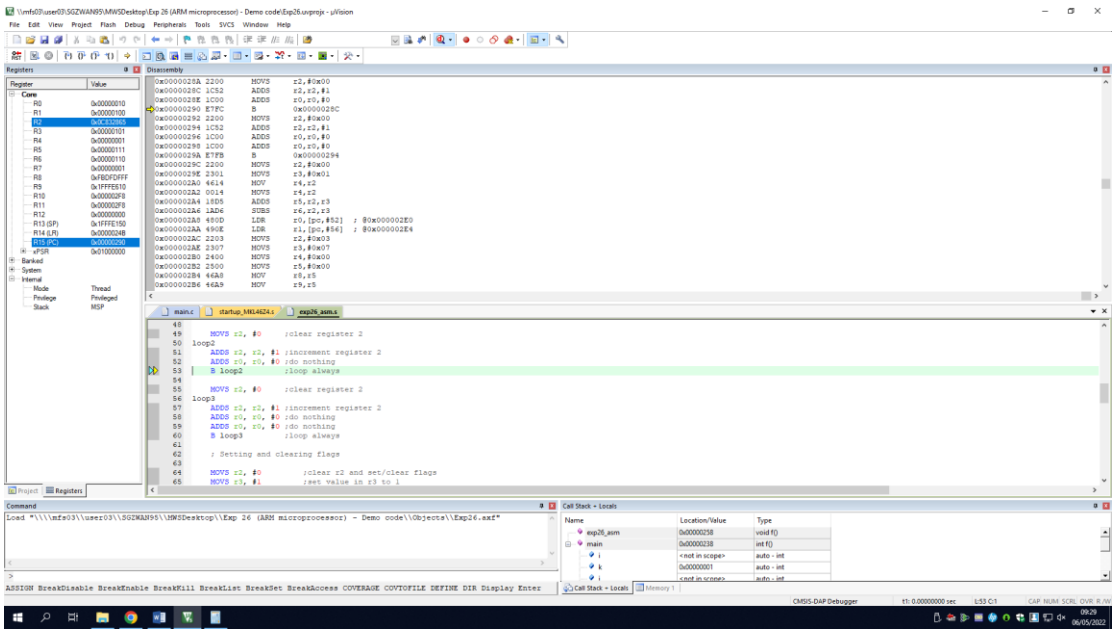


Figure 3: 0x0000028A result (20.10s)

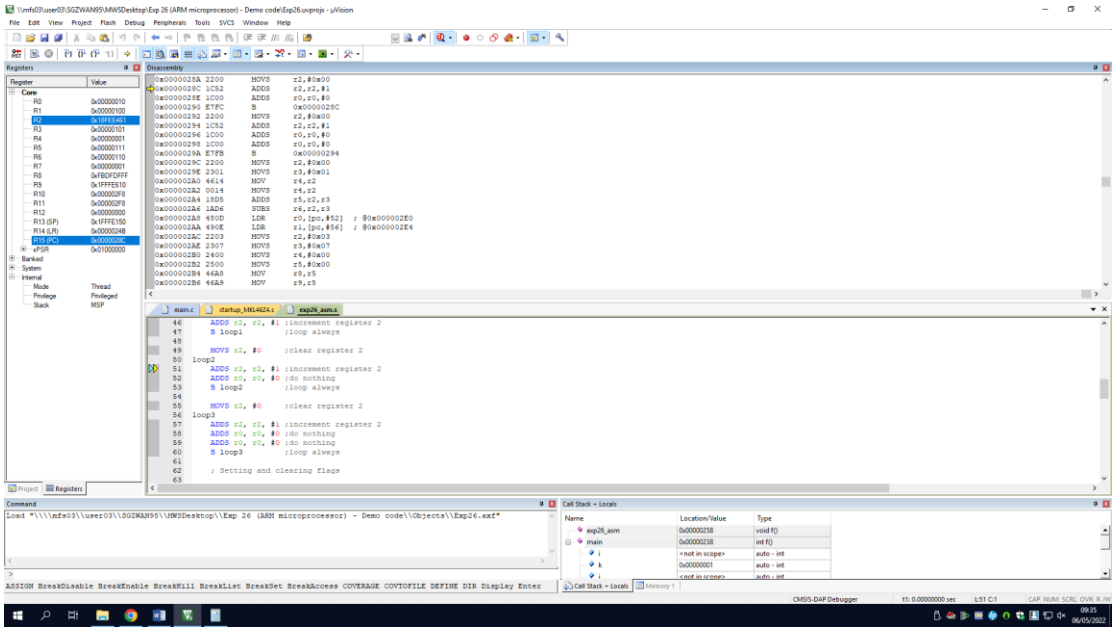


Figure 4: 0x0000028A result (40.09s)

Set counter to 0x00000292

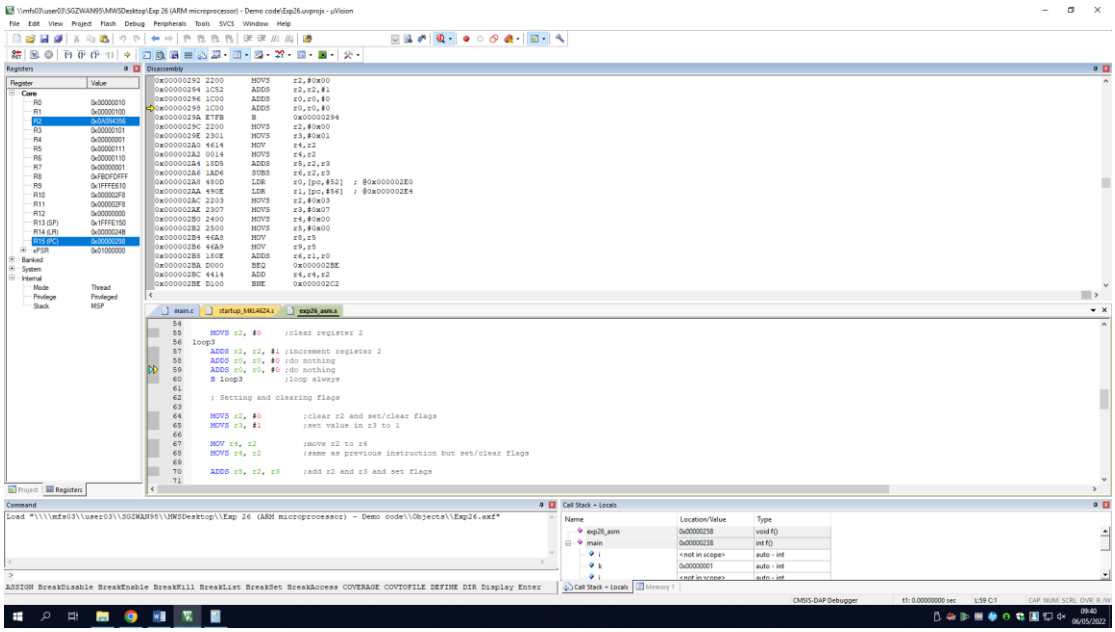


Figure 5: 0x00000292 result (20.00s)

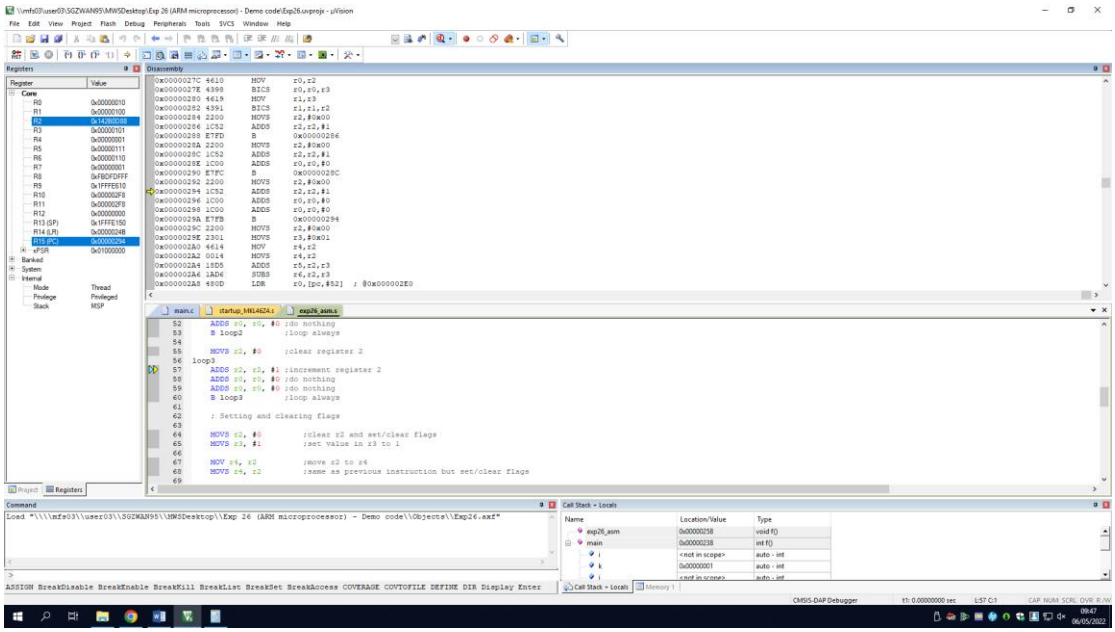


Figure 6: 0x00000292 result (40.15s)

Figure 7: plot

Screenshot (Snipping tool, Preview or equivalent):

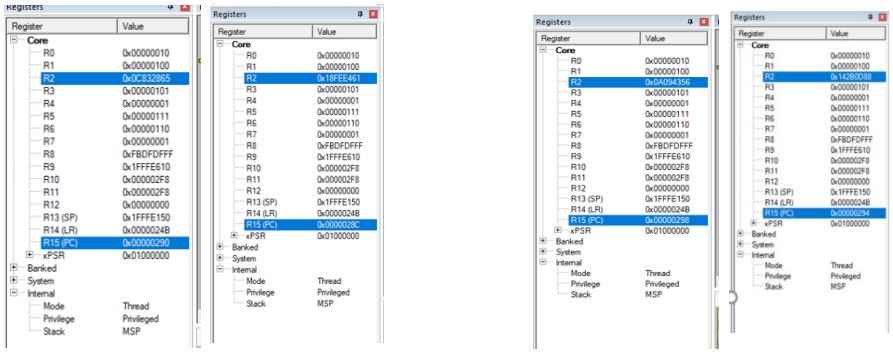


Figure 7: Registers

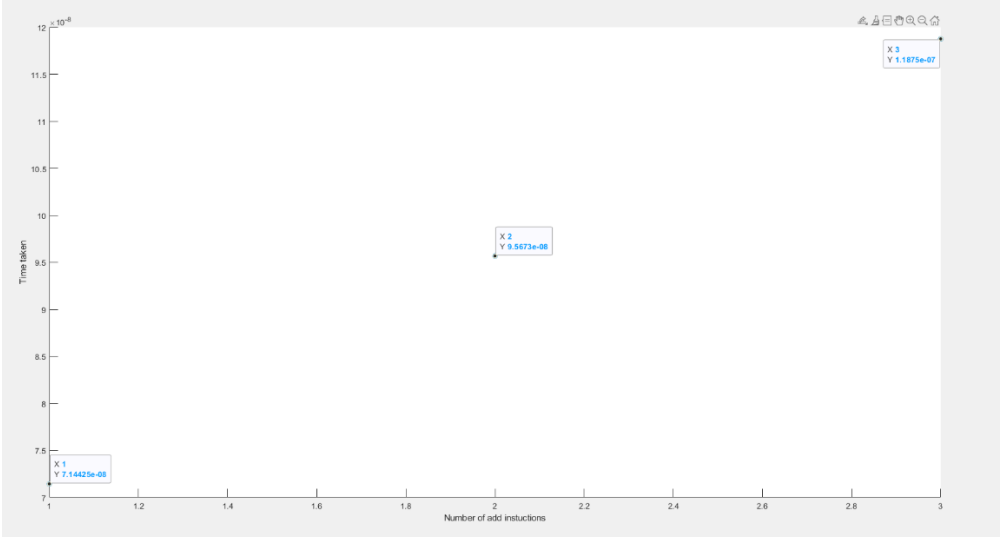


Figure 8: MATLAB plot

Comment/Explanation:

The detail of each condition is shown below:

Address	Time	Num. of add instruction	R_2 value	Value per second	Time per instruction
0x00000288	20.15s	1	0x10C152A2 $281105058_{(10)}$	$13950623.23_{(10)}$	$71.681*10^{-9}$s
0x00000288	40.35s	1	0x21C6D323 $566678307_{(10)}$	$14044072.04_{(10)}$	$71.204*10^{-9}$s
0x0000028A	20.10s	2	0x0C832865 $209921149_{(10)}$	$10443838.26_{(10)}$	$95.750*10^{-9}$s
0x0000028A	40.09s	2	0x18FEE461 $419357793_{(10)}$	$10460408.9_{(10)}$	$95.596*10^{-9}$s
0x00000292	20.00s	3	0xA094356 $168379222_{(10)}$	$8418961.1_{(10)}$	$1.188*10^{-7}$s
0x00000292	40.15s	3	0x142B0D88	$8427542.516_{(10)}$	$1.187*10^{-7}$s

From the table, the time average time for one instruction is about 71.4425ns. The average time for two instruction is about 95.673ns, while the average time for three instructions is about $1.1875*10^{-7}$s. These values can be seen on the MATLAB plot.

From the graph, it can be seen that the three points are roughly a straight line. That is also prove that the time for execute instruction is a certain value.

13. Answer to Q12 [4 marks]

Answer:

23.254s

Explanation:

By taking the results from section 9, the calculation steps are shown below:

(95.673n - 71.4425n)/2 = 23.077ns

$(1.1875*10^{-7} - 95.673n)/2 = 24.2305ns$

Therefore, the time for each instruction is about

(23.077n + 24.2305n)/2 = 23.654ns.

Since add instructions takes one clock to execute, the clock cycle shall be 23.564ns.

14. Answer to Q13 [4 marks]

Answer:

About 2 clock cycles.

Explanation:

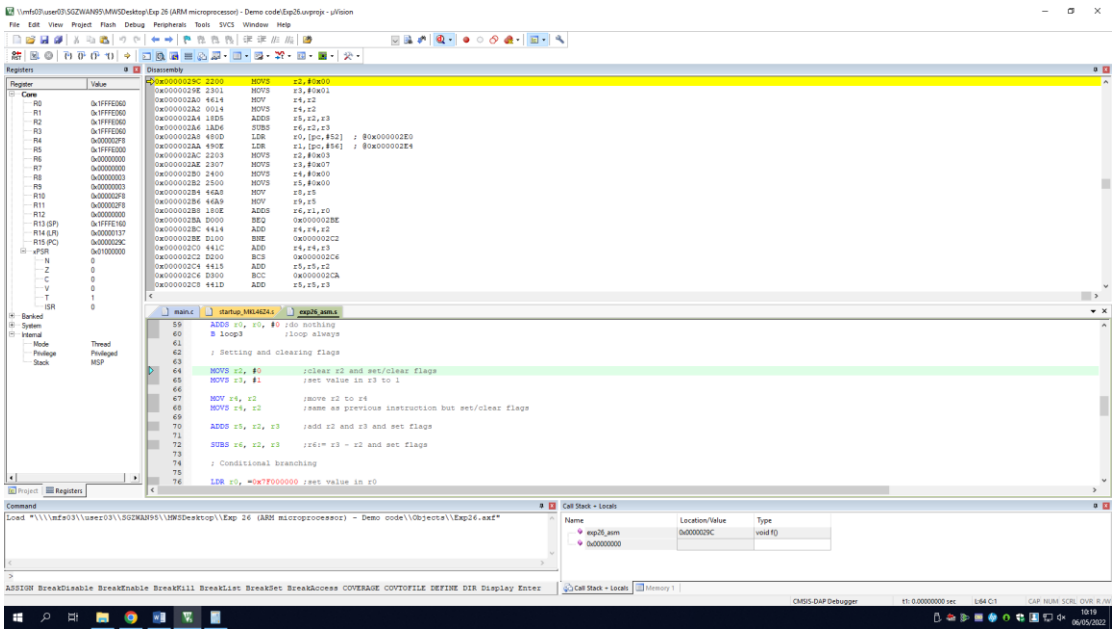
By taking the loop contain one add instruction to calculate the time.

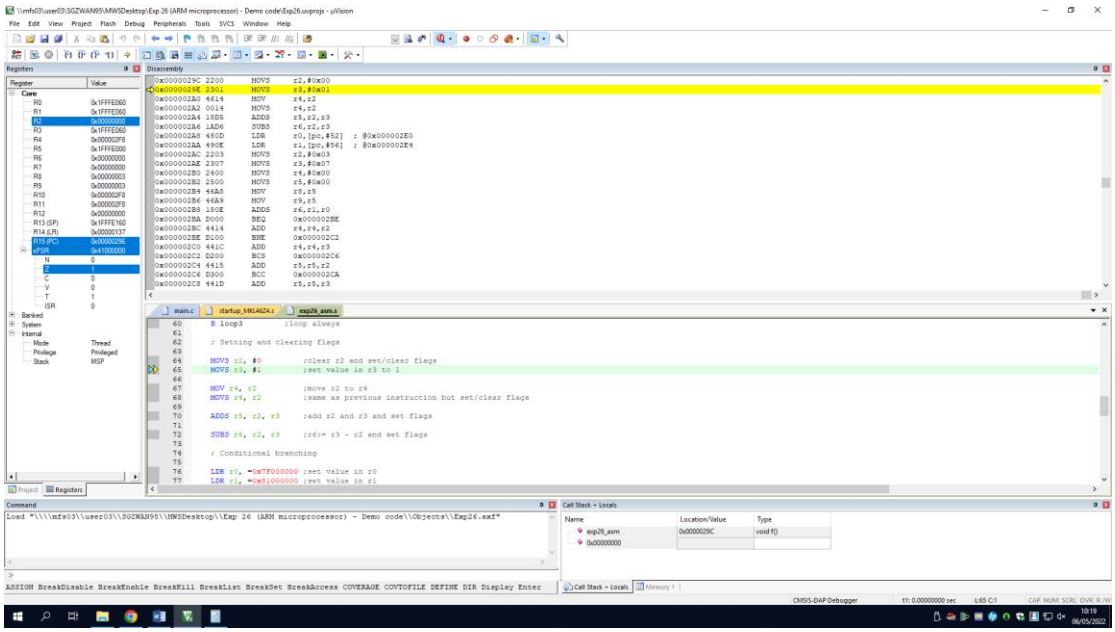
(71.4425-23.654)/23.654 = 2.02

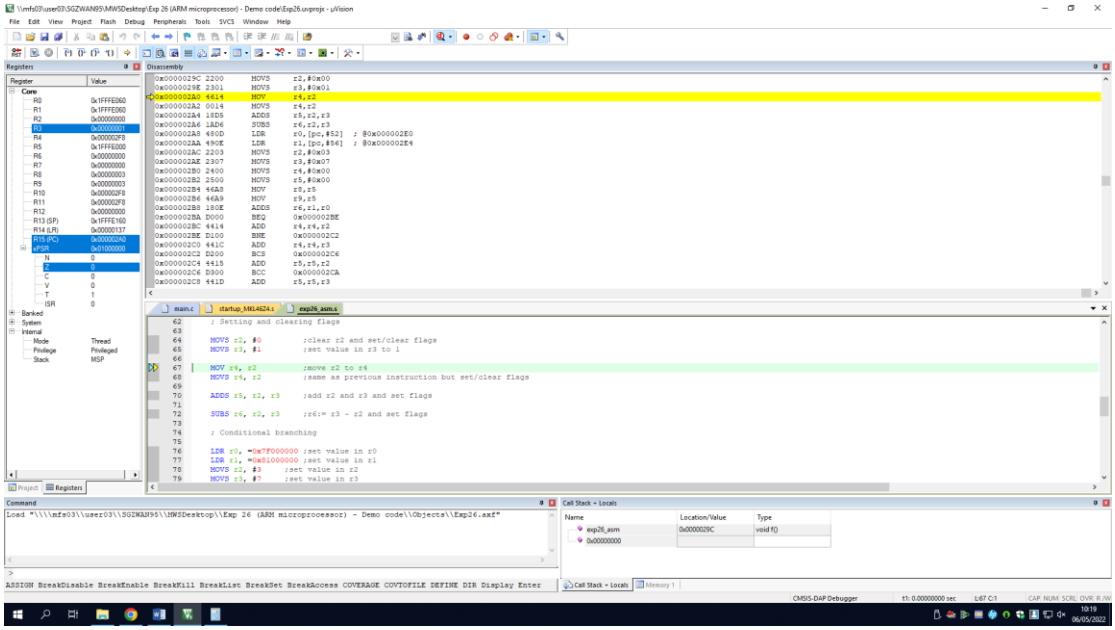
Therefore, the required clock cycle is about 2.

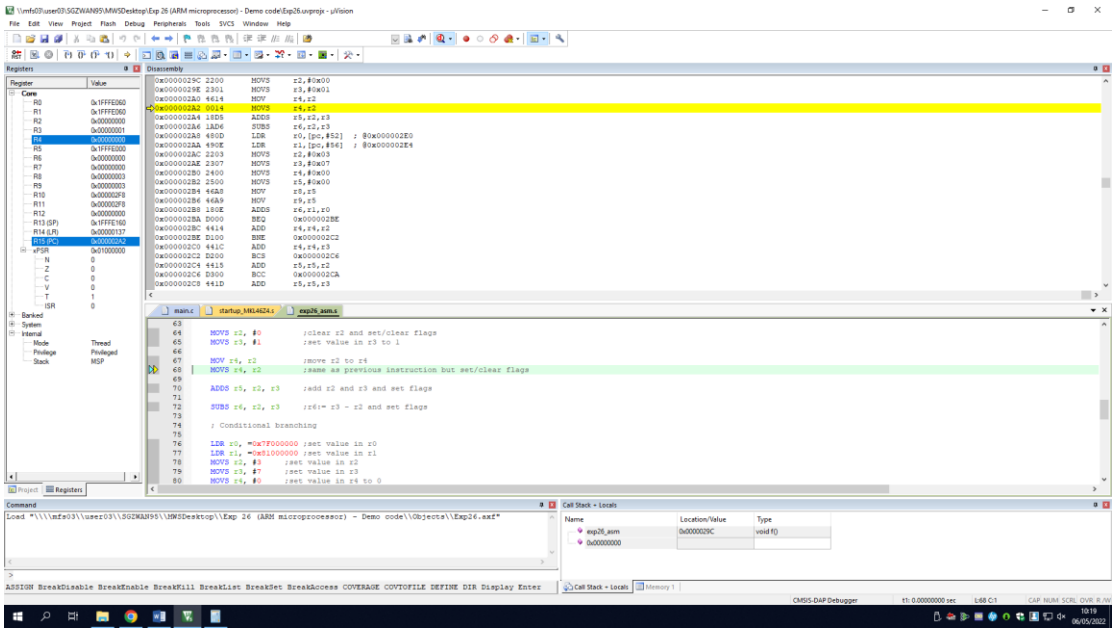
Screenshot of the result of Section 10 [3 marks]

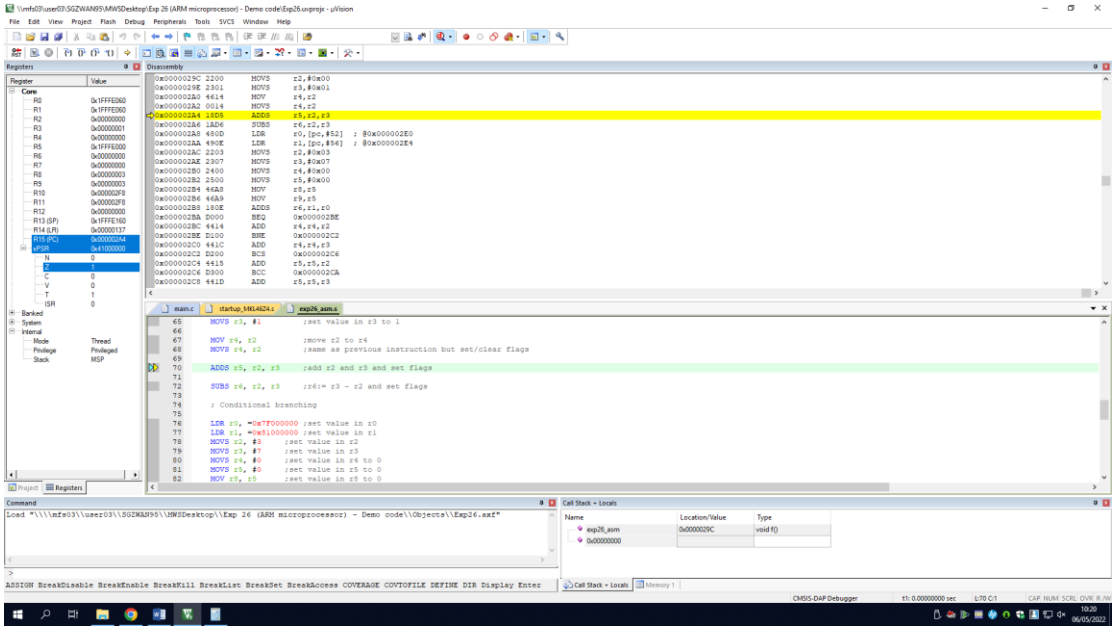
Screenshot (PrintScreen or Screenshot):

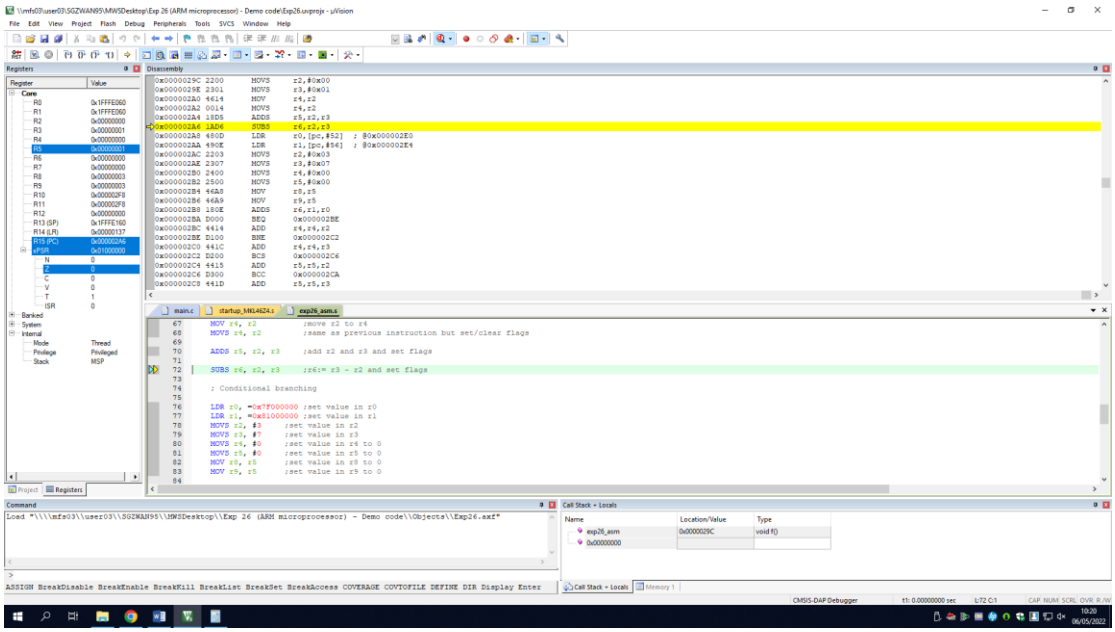


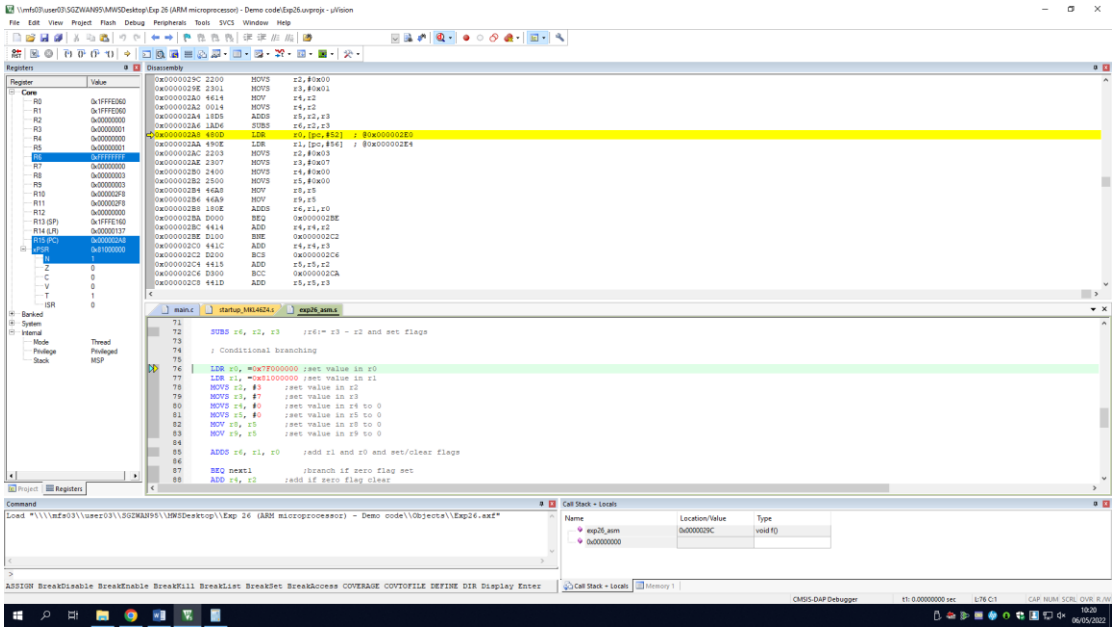


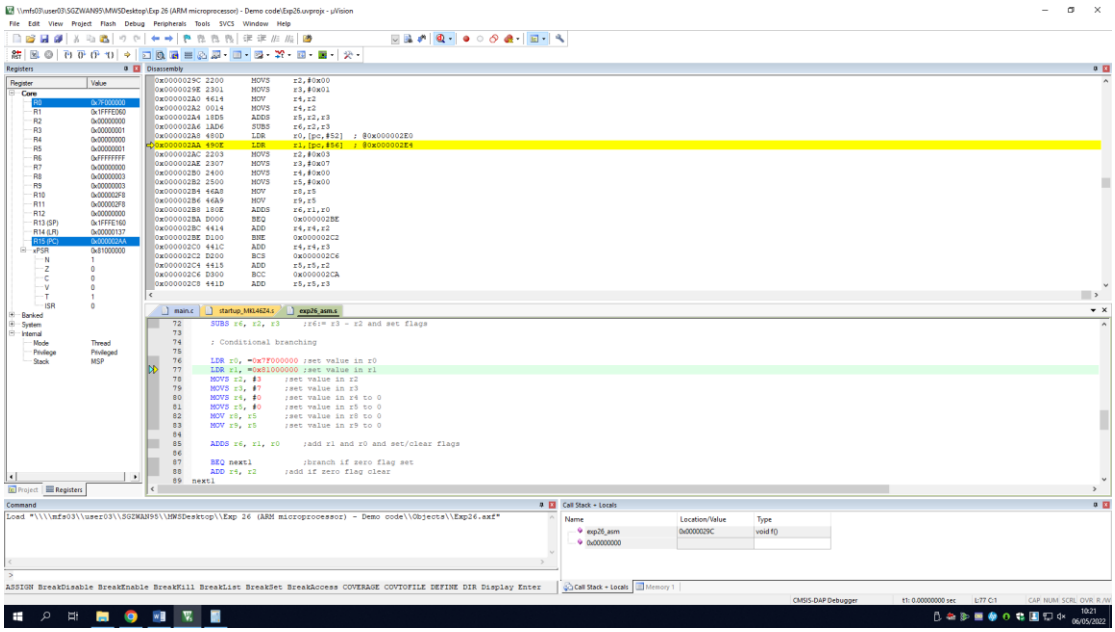




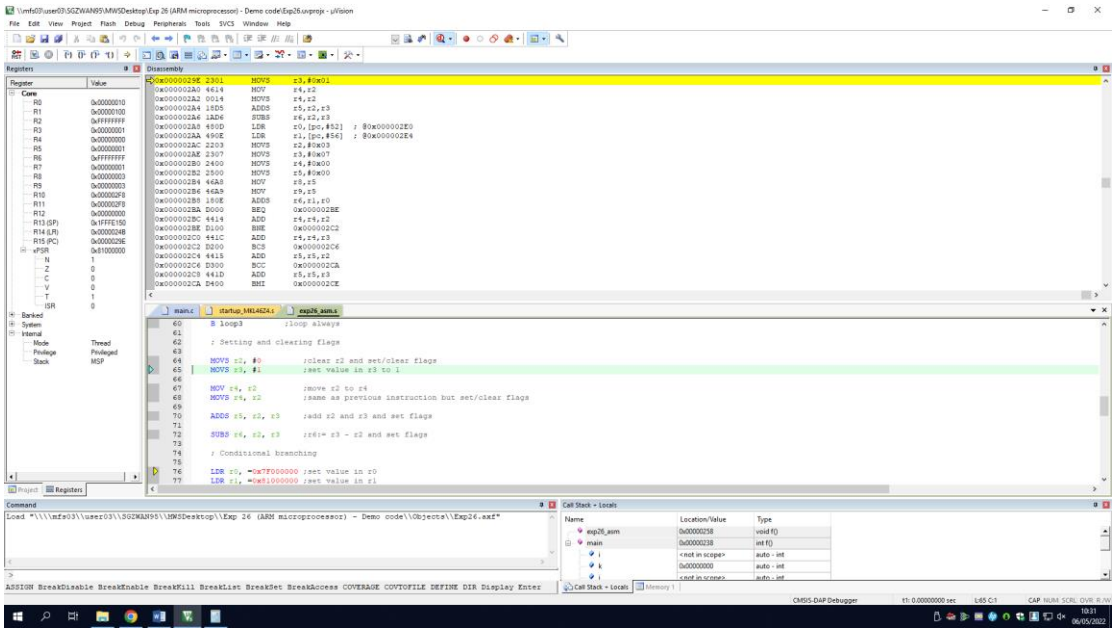


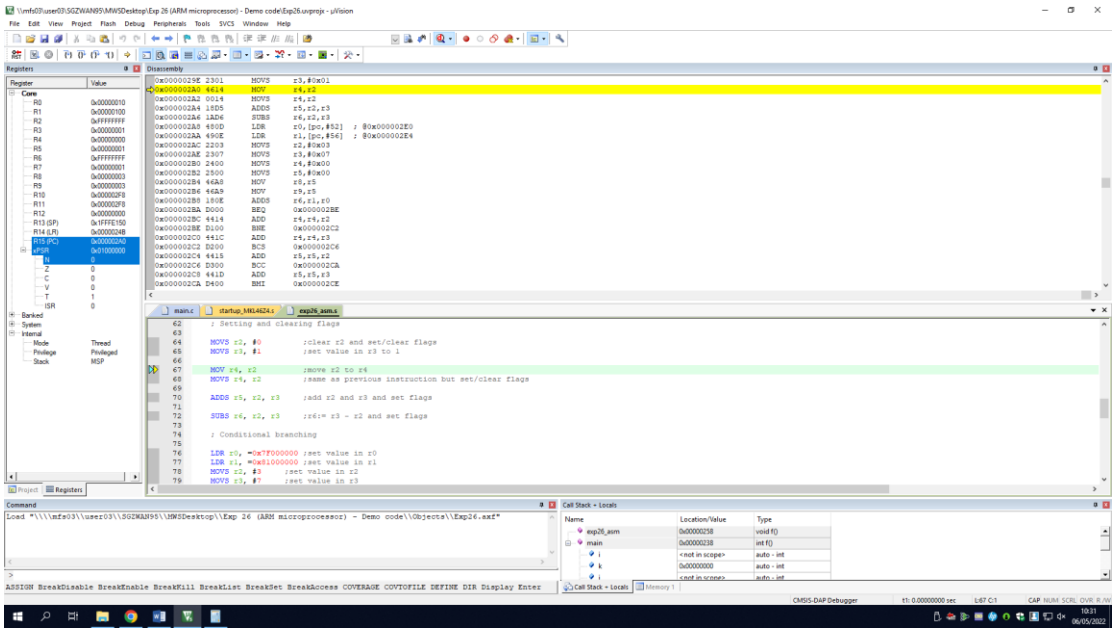


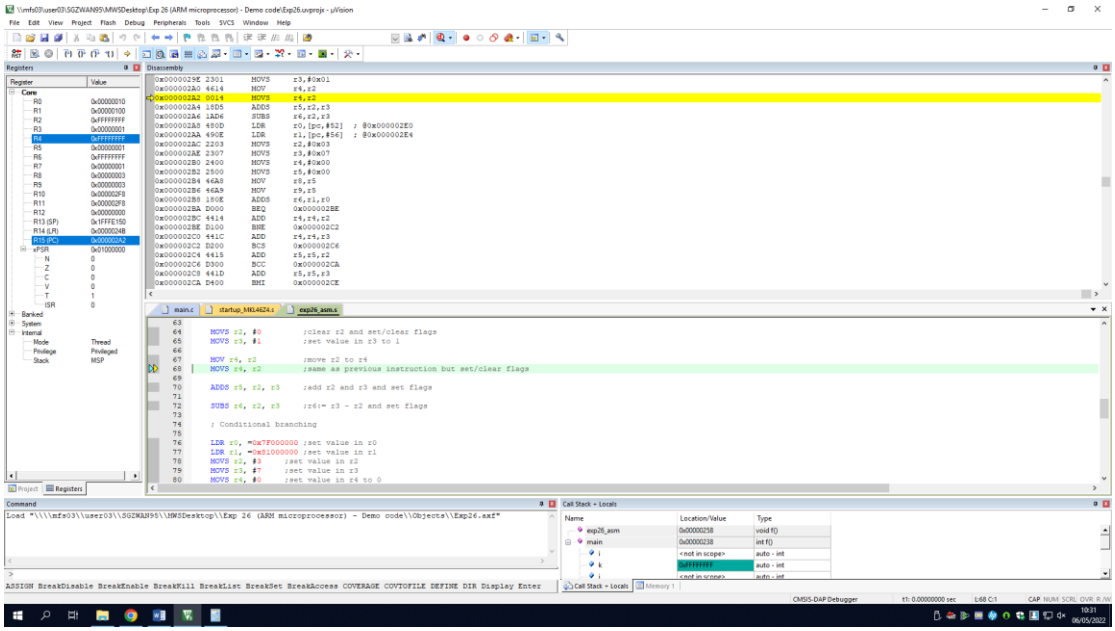


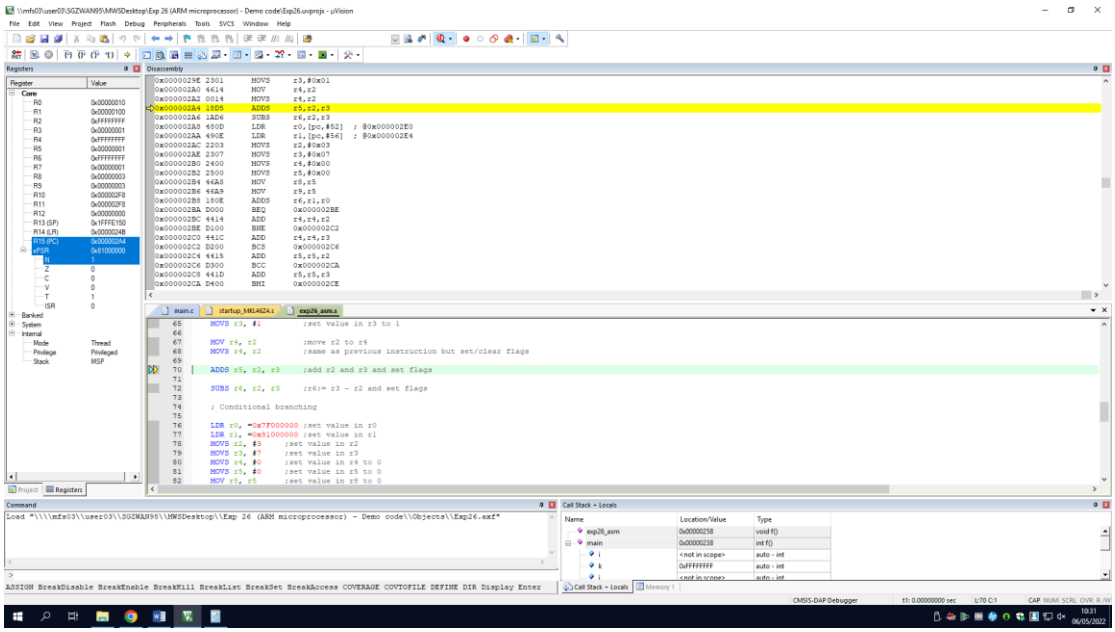


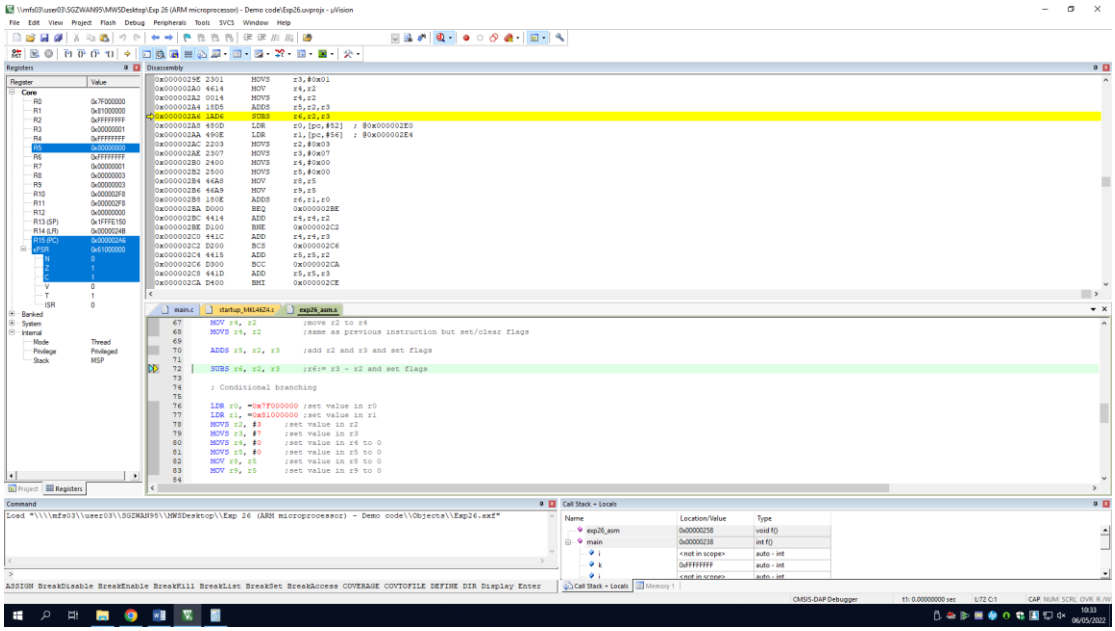
Change value of R_2 to 0xFFFFFFFF

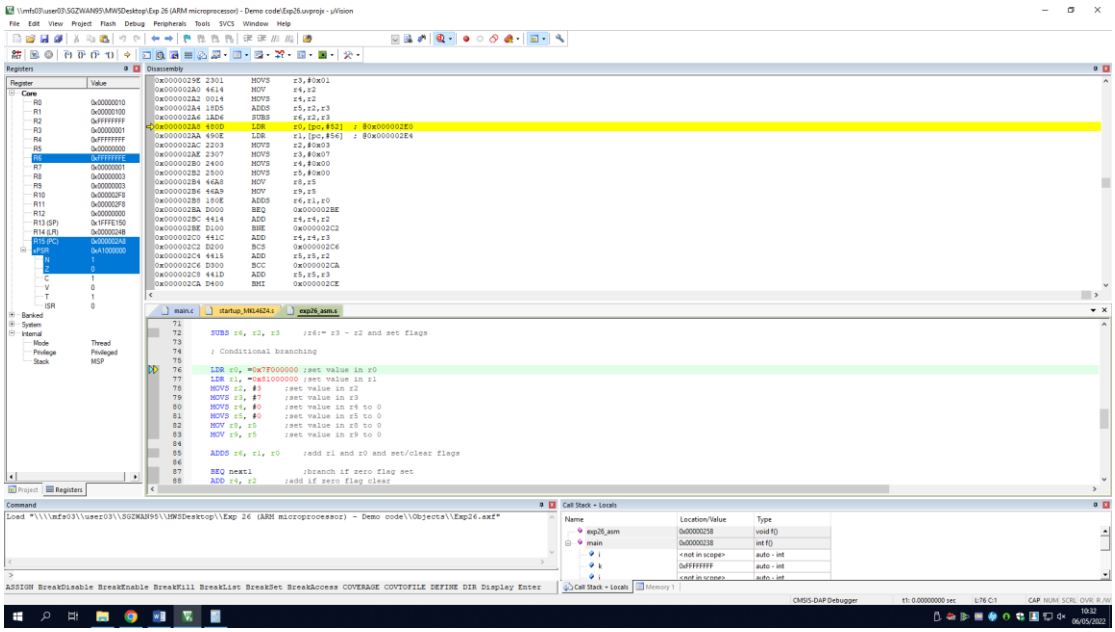


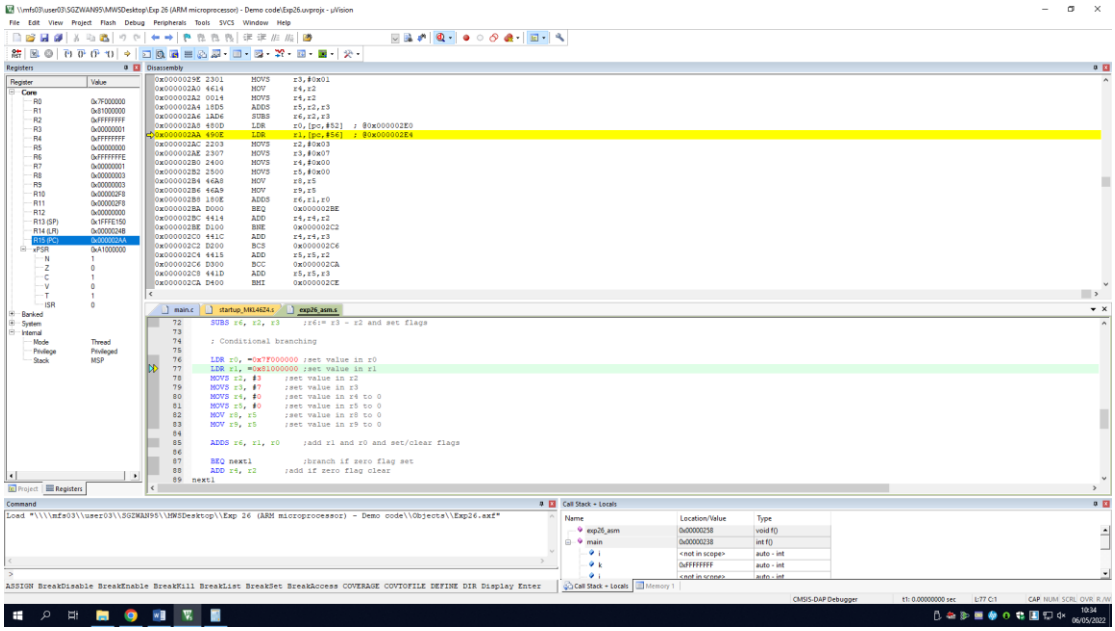




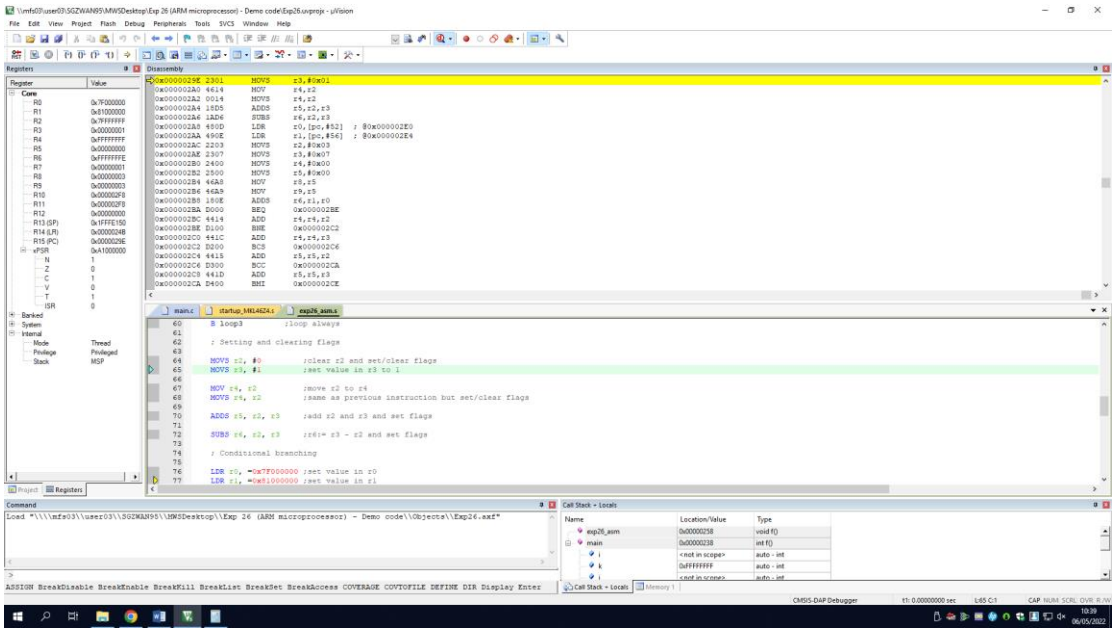


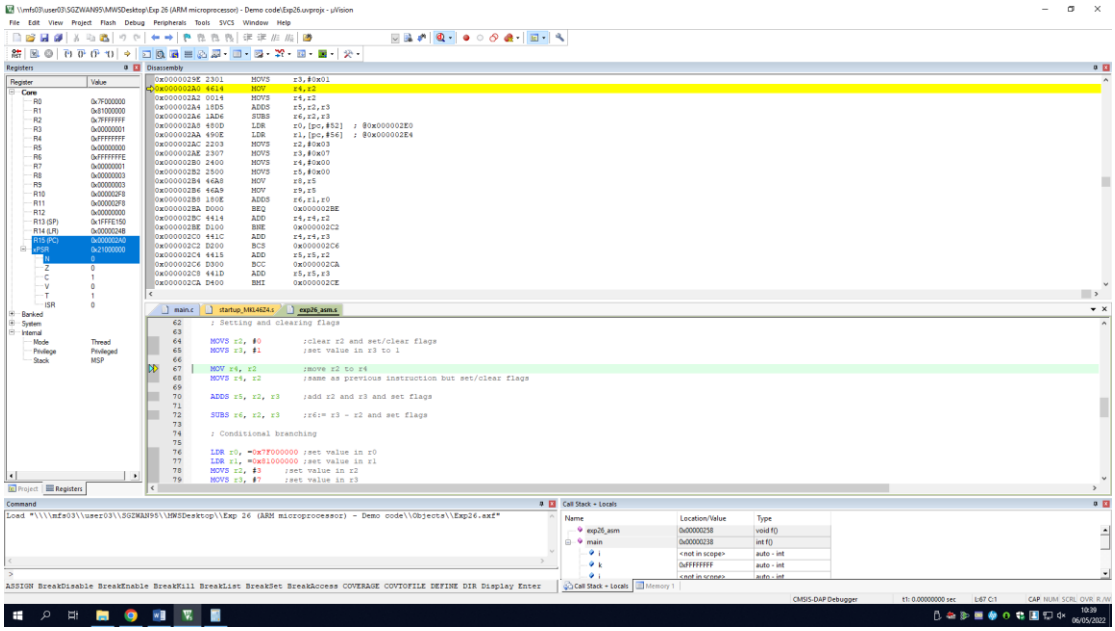


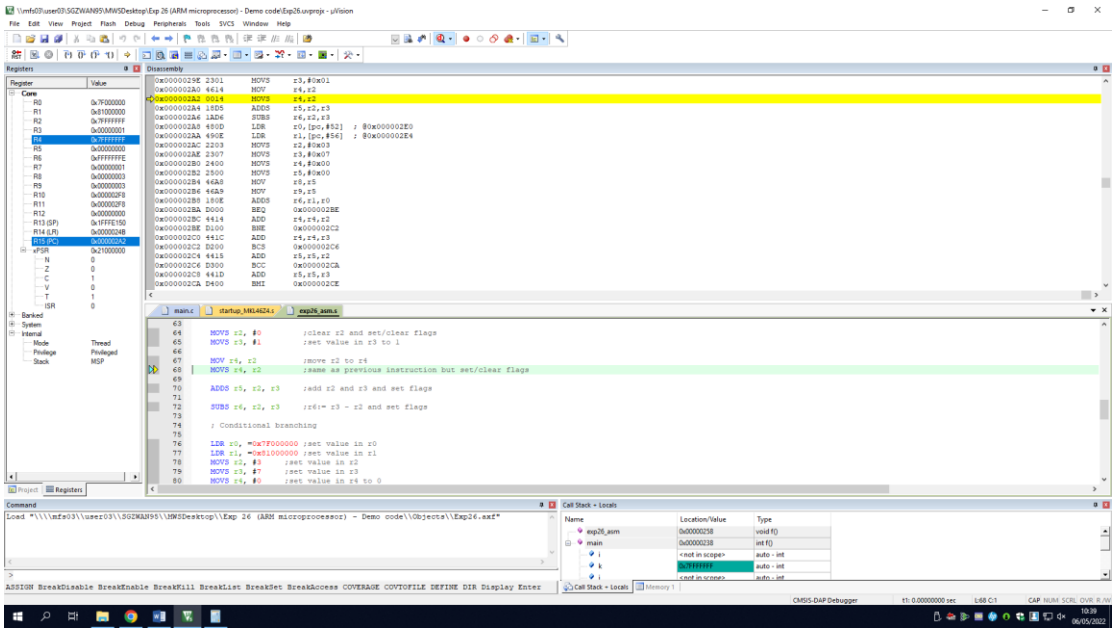


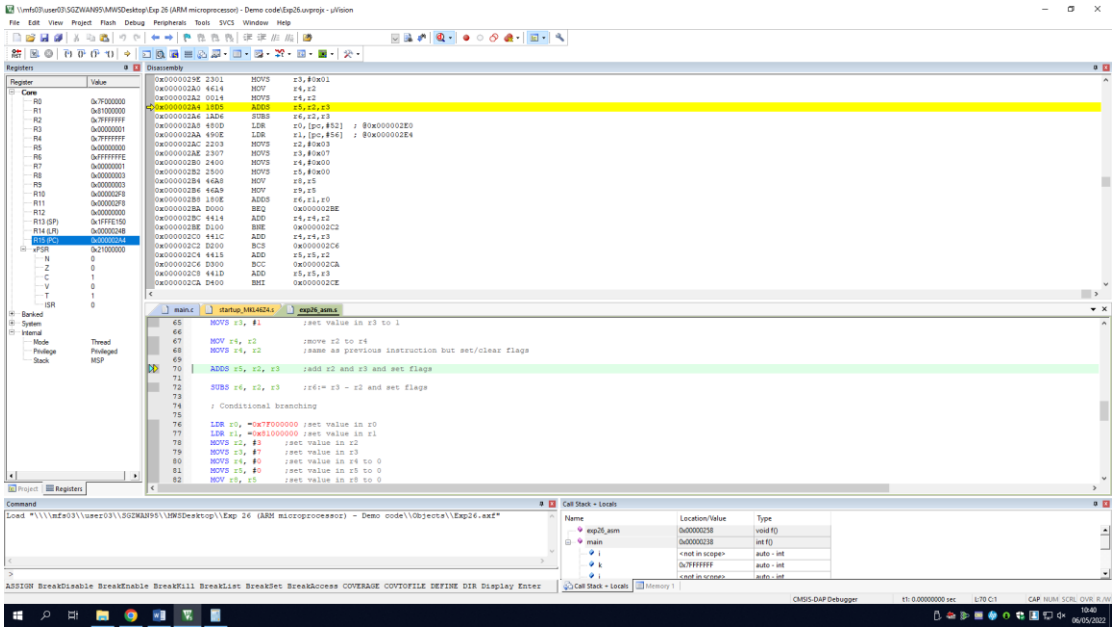


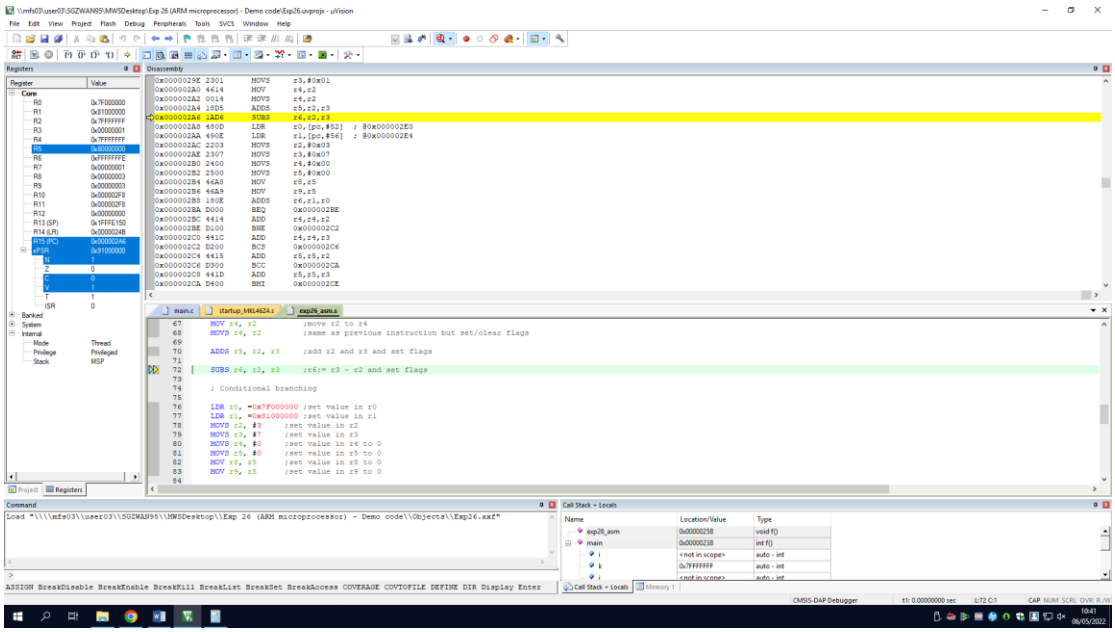
Set R2 0x7FFFFFFF

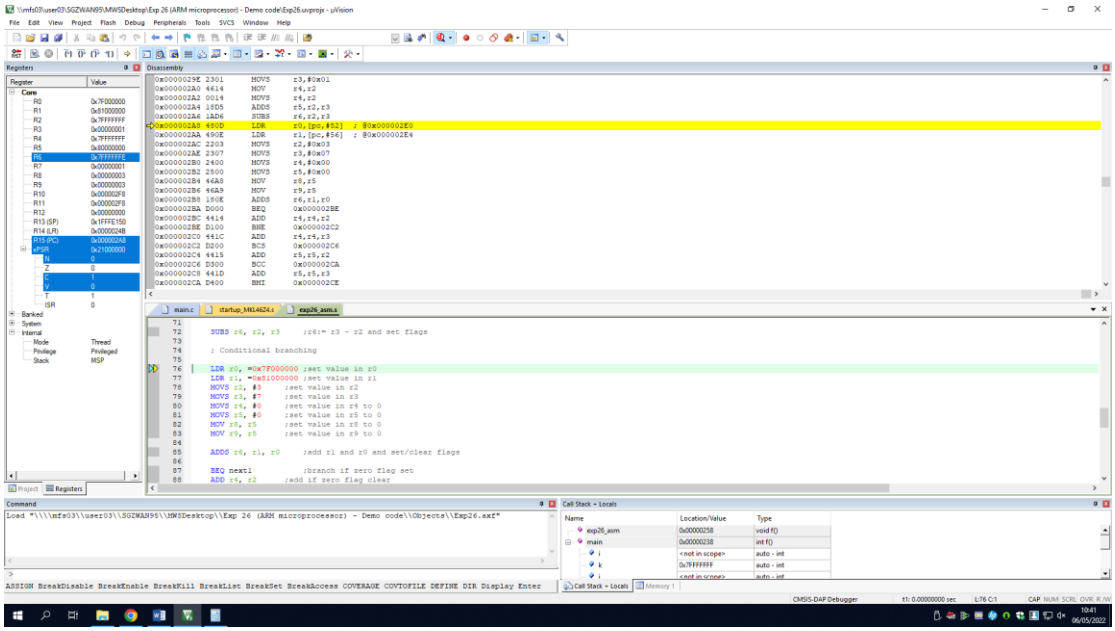


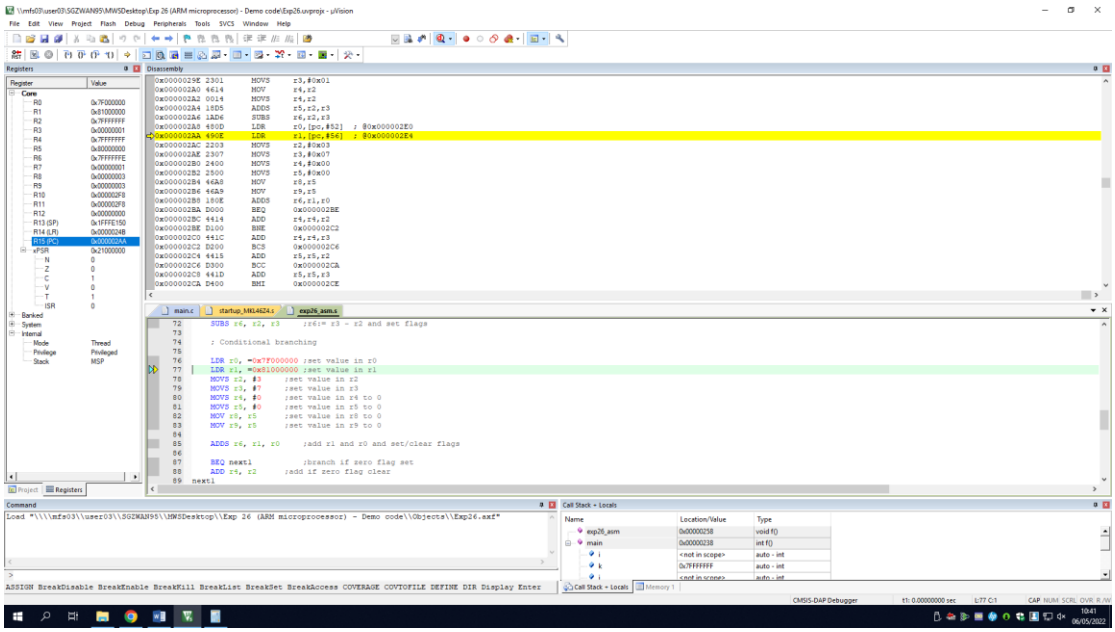




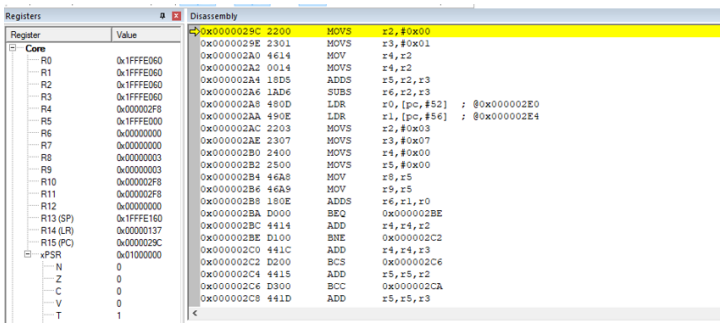


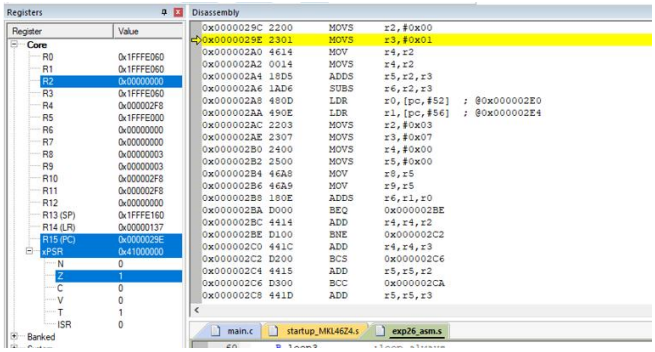


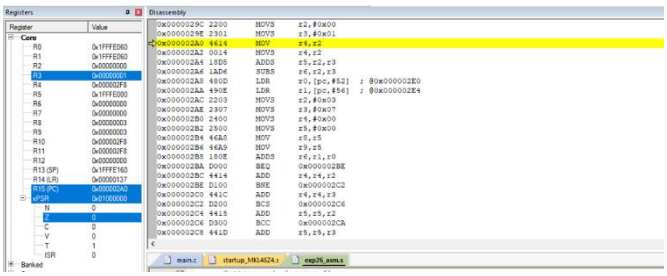


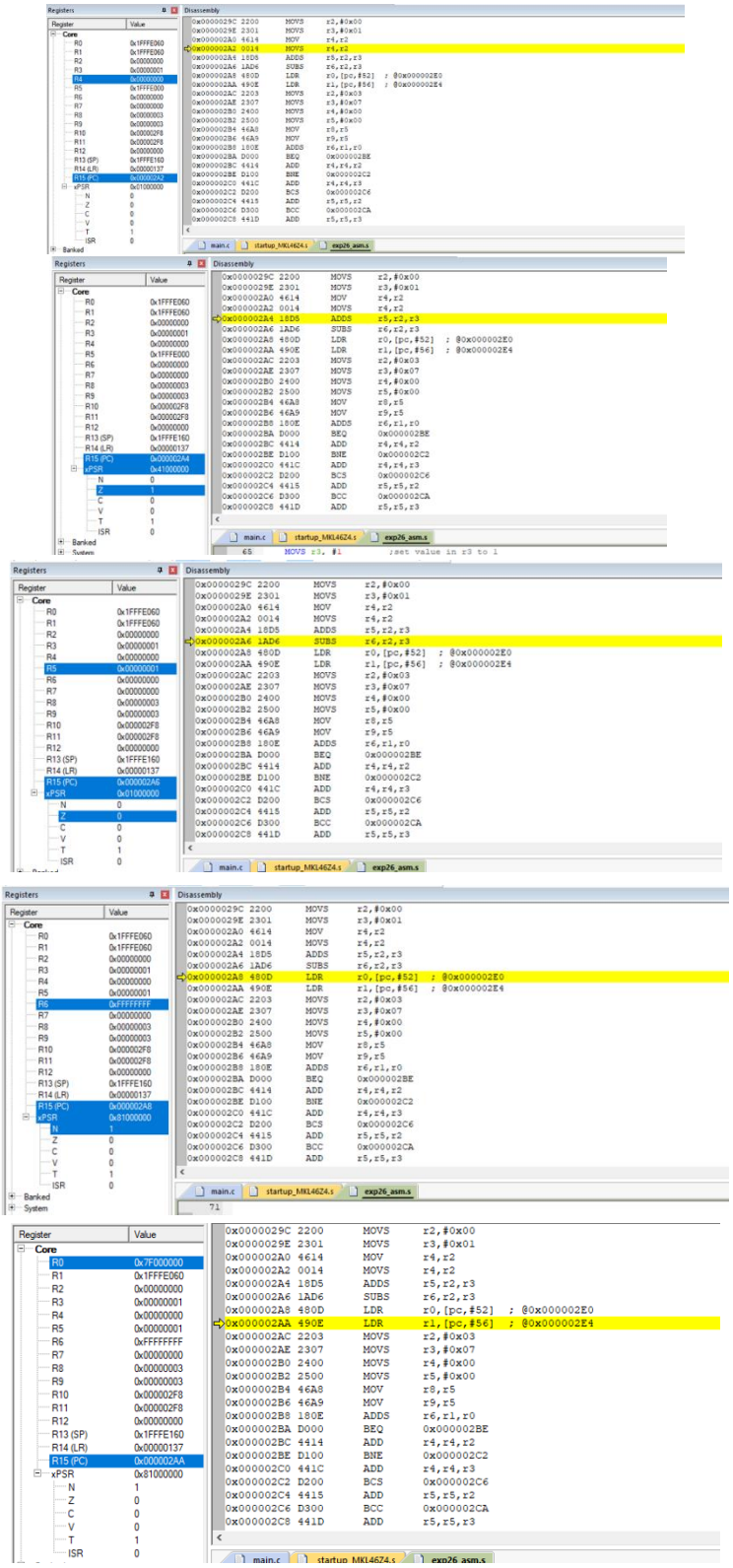


Screenshot (Snipping tool, Preview or equivalent):

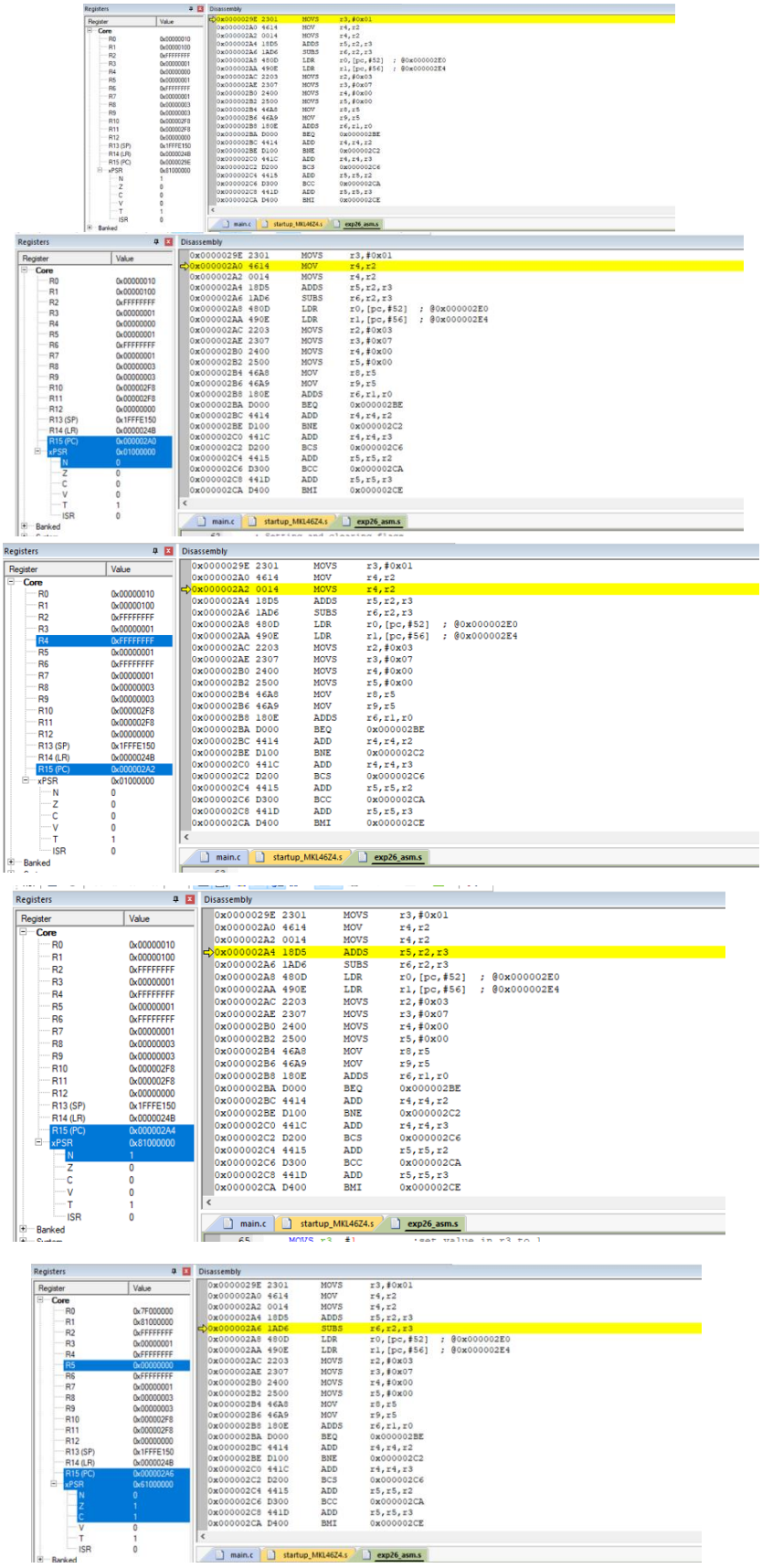


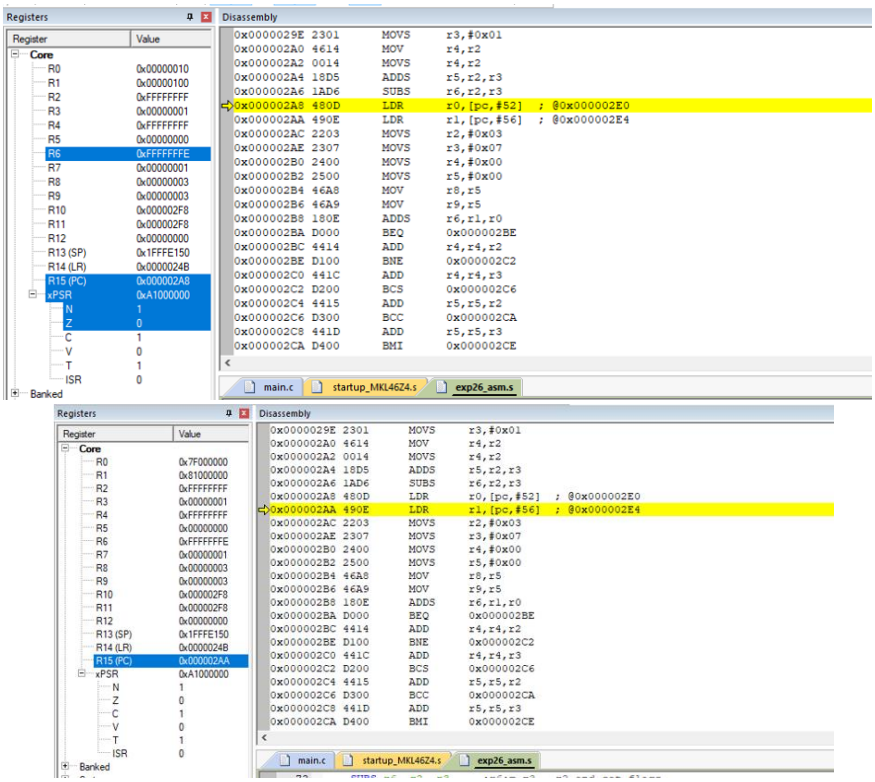




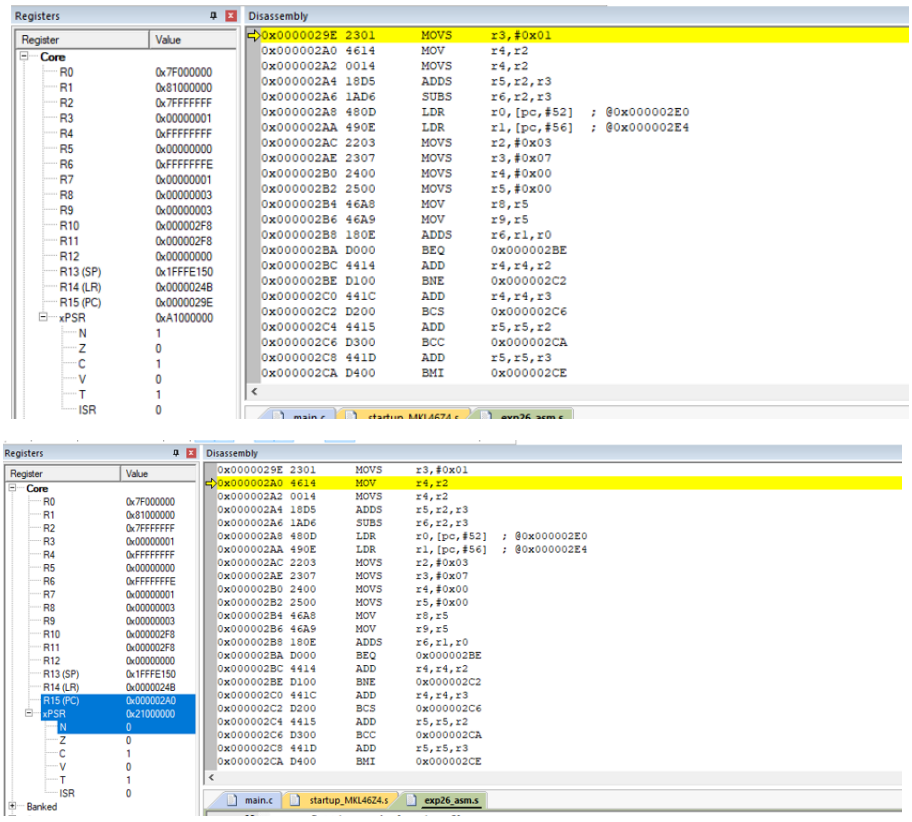


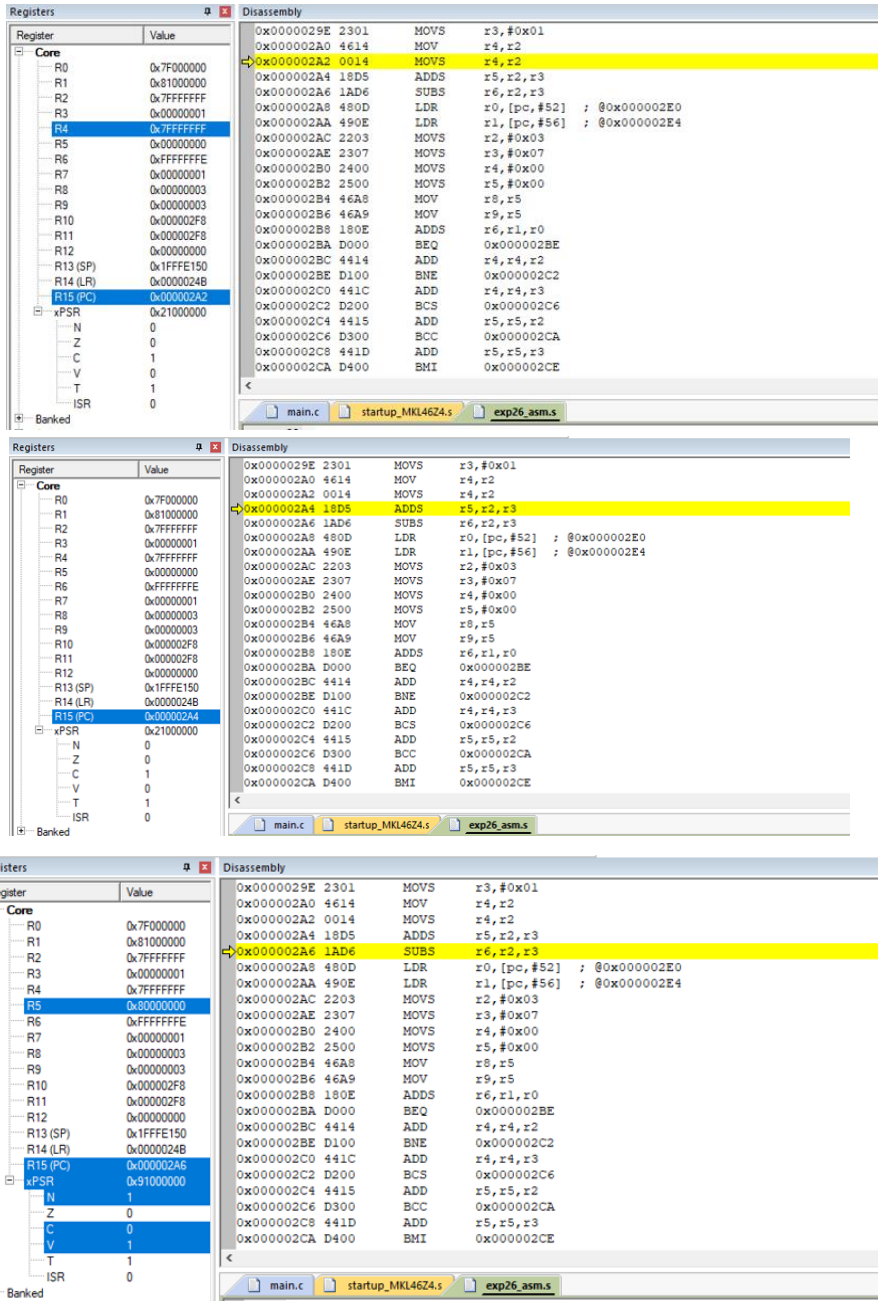
Change the value to 0xFFFFFFFF


Registers
Register
Value
Core
R0 0x00000010
R1 0x00000100
R2 0xFFFFFFFF
R3 0x00000001
R4 0x00000000
R5 0x00000001
R6 0xFFFFFFFF
R7 0x00000001
R8 0x00000003
R9 0x00000003
R10 0x000002F8
R11 0x000002F8
R12 0x00000000
R13 (SP) 0x1FFFE150
R14 (LR) 0x0000024B
R15 (PC) 0x000002A0
xPSR 0x01000000
N 0
Z 0
C 0
V 0
T 1
ISR 0
Banked
Disassembly
0x0000029E 2301 MOVS r3,#0x01
0x000002A0 4614 MOV r4,r2
0x000002A2 0014 MOVS r4,r2
0x000002A4 18D5 ADDS r5,r2,r3
0x000002A6 1AD6 SUBS r6,r2,r3
0x000002A8 480D LDR r0,[pc,#52] ; @0x000002E0
0x000002AA 490E LDR r1,[pc,#56] ; @0x000002E4
0x000002AC 2203 MOVS r2,#0x03
0x000002AE 2307 MOVS r3,#0x07
0x000002B0 2400 MOVS r4,#0x00
0x000002B2 2500 MOVS r5,#0x00
0x000002B4 46A8 MOV r8,r5
0x000002B6 46A9 MOV r9,r5
0x000002B8 180E ADDS r6,r1,r0
0x000002BA D000 BEQ 0x000002BE
0x000002BC 4414 ADD r4,r4,r2
0x000002BE D100 BNE 0x000002C2
0x000002C0 441C ADD r4,r4,r3
0x000002C2 D200 BCS 0x000002C6
0x000002C4 4415 ADD r5,r5,r2
0x000002C6 D300 BCC 0x000002CA
0x000002C8 441D ADD r5,r5,r3
0x000002CA D400 BMI 0x000002CE
main.c
startup_MKL46Z4.s
exp26_asm.s



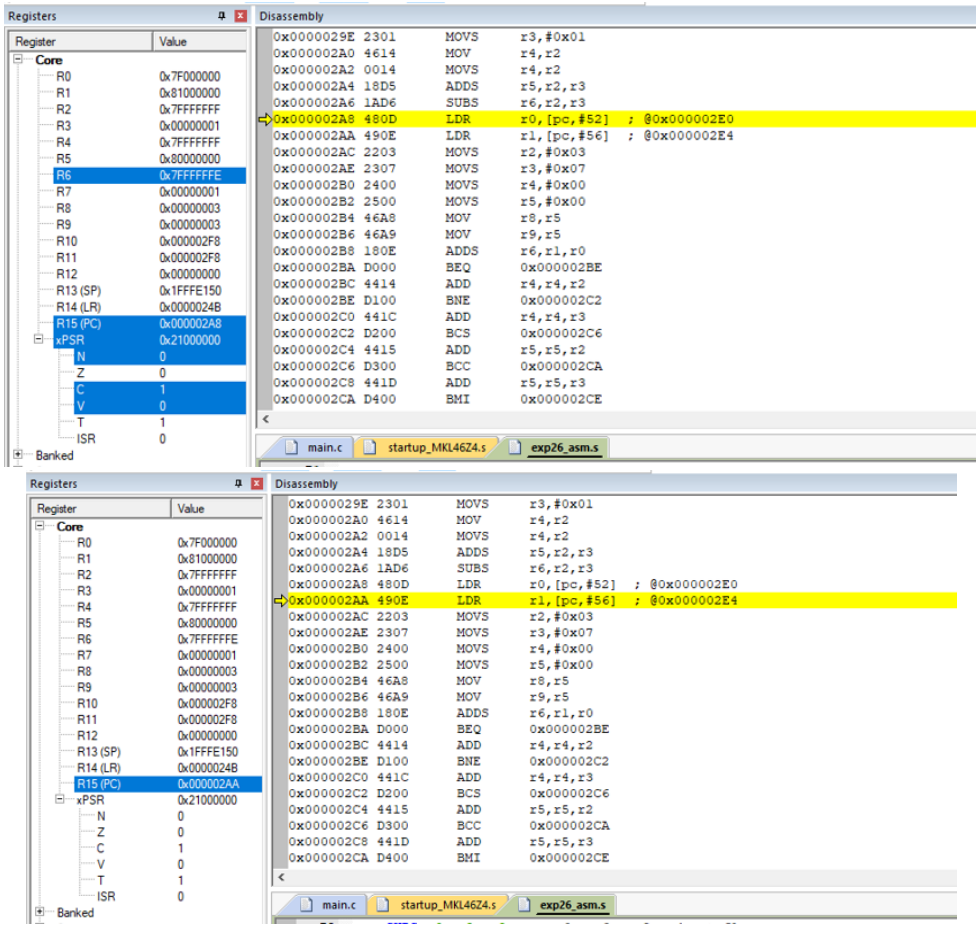
Set R2 0x7FFFFFFF




Registers
Register
Value
Core
R0 0x7F000000
R1 0x81000000
R2 0x7FFFFFFF
R3 0x00000001
R4 0x7FFFFFFF
R5 0x00000000
R6 0xFFFFFFFE
R7 0x00000001
R8 0x00000003
R9 0x00000003
R10 0x000002F8
R11 0x000002F8
R12 0x00000000
R13 (SP) 0x1FFFE150
R14 (LR) 0x0000024B
R15 (PC) 0x000002A2
xPSR 0x21000000
N 0
Z 0
C 1
V 0
T 1
ISR 0
Banked
Disassembly
0x0000029E 2301 MOVS r3,#0x01
0x000002A0 4614 MOV r4,r2
0x000002A2 0014 MOVS r4,r2
0x000002A4 18D5 ADDS r5,r2,r3
0x000002A6 1AD6 SUBS r6,r2,r3
0x000002A8 480D LDR r0,[pc,#52] ; @0x000002E0
0x000002AA 490E LDR r1,[pc,#56] ; @0x000002E4
0x000002AC 2203 MOVS r2,#0x03
0x000002AE 2307 MOVS r3,#0x07
0x000002B0 2400 MOVS r4,#0x00
0x000002B2 2500 MOVS r5,#0x00
0x000002B4 46A8 MOV r8,r5
0x000002B6 46A9 MOV r9,r5
0x000002B8 180E ADDS r6,r1,r0
0x000002BA D000 BEQ 0x000002BE
0x000002BC 4414 ADD r4,r4,r2
0x000002BE D100 BNE 0x000002C2
0x000002C0 441C ADD r4,r4,r3
0x000002C2 D200 BCS 0x000002C6
0x000002C4 4415 ADD r5,r5,r2
0x000002C6 D300 BCC 0x000002CA
0x000002C8 441D ADD r5,r5,r3
0x000002CA D400 BMI 0x000002CE
main.c
startup_MKL46Z4.s
exp26_asm.s

Registers
Register
Value
Core
R0 0x7F000000
R1 0x81000000
R2 0x7FFFFFFF
R3 0x00000001
R4 0x7FFFFFFF
R5 0x00000000
R6 0xFFFFFFFE
R7 0x00000001
R8 0x00000003
R9 0x00000003
R10 0x000002F8
R11 0x000002F8
R12 0x00000000
R13 (SP) 0x1FFFE150
R14 (LR) 0x0000024B
R15 (PC) 0x000002A4
xPSR 0x21000000
N 0
Z 0
C 1
V 0
T 1
ISR 0
Banked
Disassembly
0x0000029E 2301 MOVS r3,#0x01
0x000002A0 4614 MOV r4,r2
0x000002A2 0014 MOVS r4,r2
0x000002A4 18D5 ADDS r5,r2,r3
0x000002A6 1AD6 SUBS r6,r2,r3
0x000002A8 480D LDR r0,[pc,#52] ; @0x000002E0
0x000002AA 490E LDR r1,[pc,#56] ; @0x000002E4
0x000002AC 2203 MOVS r2,#0x03
0x000002AE 2307 MOVS r3,#0x07
0x000002B0 2400 MOVS r4,#0x00
0x000002B2 2500 MOVS r5,#0x00
0x000002B4 46A8 MOV r8,r5
0x000002B6 46A9 MOV r9,r5
0x000002B8 180E ADDS r6,r1,r0
0x000002BA D000 BEQ 0x000002BE
0x000002BC 4414 ADD r4,r4,r2
0x000002BE D100 BNE 0x000002C2
0x000002C0 441C ADD r4,r4,r3
0x000002C2 D200 BCS 0x000002C6
0x000002C4 4415 ADD r5,r5,r2
0x000002C6 D300 BCC 0x000002CA
0x000002C8 441D ADD r5,r5,r3
0x000002CA D400 BMI 0x000002CE
main.c
startup_MKL46Z4.s
exp26_asm.s

Registers
Register
Value
Core
R0 0x7F000000
R1 0x81000000
R2 0x7FFFFFFF
R3 0x00000001
R4 0x7FFFFFFF
R5 0x80000000
R6 0xFFFFFFFE
R7 0x00000001
R8 0x00000003
R9 0x00000003
R10 0x000002F8
R11 0x000002F8
R12 0x00000000
R13 (SP) 0x1FFFE150
R14 (LR) 0x0000024B
R15 (PC) 0x000002A6
xPSR 0x91000000
N 1
Z 0
C 0
V 1
T 1
ISR 0
Banked
Disassembly
0x0000029E 2301 MOVS r3,#0x01
0x000002A0 4614 MOV r4,r2
0x000002A2 0014 MOVS r4,r2
0x000002A4 18D5 ADDS r5,r2,r3
0x000002A6 1AD6 SUBS r6,r2,r3
0x000002A8 480D LDR r0,[pc,#52] ; @0x000002E0
0x000002AA 490E LDR r1,[pc,#56] ; @0x000002E4
0x000002AC 2203 MOVS r2,#0x03
0x000002AE 2307 MOVS r3,#0x07
0x000002B0 2400 MOVS r4,#0x00
0x000002B2 2500 MOVS r5,#0x00
0x000002B4 46A8 MOV r8,r5
0x000002B6 46A9 MOV r9,r5
0x000002B8 180E ADDS r6,r1,r0
0x000002BA D000 BEQ 0x000002BE
0x000002BC 4414 ADD r4,r4,r2
0x000002BE D100 BNE 0x000002C2
0x000002C0 441C ADD r4,r4,r3
0x000002C2 D200 BCS 0x000002C6
0x000002C4 4415 ADD r5,r5,r2
0x000002C6 D300 BCC 0x000002CA
0x000002C8 441D ADD r5,r5,r3
0x000002CA D400 BMI 0x000002CE
main.c
startup_MKL46Z4.s
exp26_asm.s



Comment/Explanation:

It can be seen that the flags results are different due to the different values. What is more, flags will update after execute each instruction.
For the initial condition, only N flag is set in the final result.
When R_2 is set to 0xFFFFFFFF, N and C flags are set in the final result.
When R_2 is set to 0x7FFFFFFF, only the C flag is set.

15. Answer to Q14 [4 marks]

Answer:

Four conditional branch instructions are executed, while other 4 instructions are not executed.

Explanation:

The execution results are shown in the following:

address	instructions	flags	results

0x000002BA	BEQ 0x000002BE	Z = 1	execute
0x000002BE	BNE 0x000002C2	Z = 1	Not execute
0x000002C2	BCS 0x000002C6	C = 1	execute
0x000002C6	BCC 0x000002CA	C = 1	Not execute
0x000002CA	BMI 0x000002CE	N = 0	Not execute
0x000002CE	BPL 0x000002D2	N = 0	execute
0x000002D2	BVS 0x000002D6	V = 0	Not execute
0x000002D6	BVC 0x000002DA	V = 0	execute

The program executed according to the instructions and flags. If the codes meet the requirements, the program will jump to the certain address accordingly. Otherwise, it will not change the order.

16. Answer to Q15 [4 marks]

Answer:

Four conditional branch instructions are executed, while other four instructions are not executed.

Explanation:

The execution results are shown in the following table:

address	instructions	flags	results
0x000002BA	BEQ 0x000002BE	Z = 0	Not execute
0x000002BE	BNE 0x000002C2	Z = 0	execute
0x000002C2	BCS 0x000002C6	C = 0	Not execute
0x000002C6	BCC 0x000002CA	C = 0	execute
0x000002CA	BMI 0x000002CE	N = 1	execute
0x000002CE	BPL 0x000002D2	N = 1	Not execute
0x000002D2	BVS 0x000002D6	V = 1	execute

0x000002D6	BVC 0x000002DA	V = 1	Not execute

The program executed according to the instructions and flags. If the codes meet the requirements, the program will jump to the certain address accordingly. Otherwise, it will not change the order.

17. Screenshot of the result of Section 13, Part IV [3 marks]

Screenshots showing that the "mbed_blinky" program has been successfully exported from the Mbed Compiler and opened in Keil uVision (no need to build it).

Screenshot (PrintScreen or Screenshot):

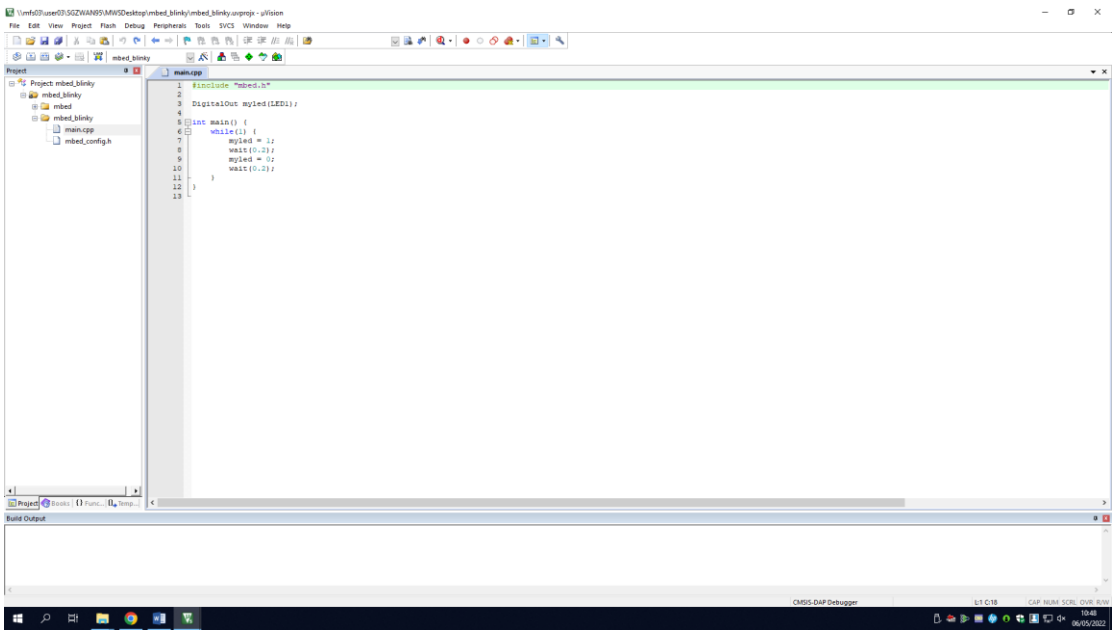


Figure 9

Screenshot (Snipping tool, Preview or equivalent):

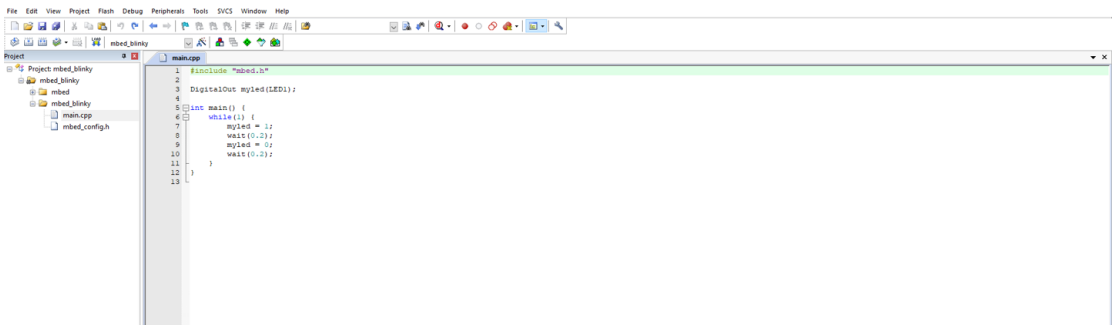


Figure 10

18. Code of the Hello World programme (Section 12) after editing (put as text NOT as a screenshot, screenshots of code will receive zero marks) [5 marks]

Code:

```
#include "mbed.h"

DigitalOut led2(LED2);
DigitalOut led1(LED1);

int main() {
    while(1) {

        led2 = 0;
        led1 = 1;
        wait(0.2);
        led1 = 0;
        led2 = 1;
        wait(0.2);

    }
}
```

```
#include "mbed.h"

DigitalOut myled1(LED1);
DigitalOut myled2(LED2);

int main() {
    while(1) {

        myled1 = 1;
        myled2 = 0;
        wait(0.015);
        myled1= 0;
        myled2 = 1;
        wait(0.015);

    }
}
```

Explanation:

For the first version of code, it changes the LED1 to LED2 means change the green LED to red LED instead. Then, the red LED will flash every 0.2 second.

For the second version of the code, we can observe that all the two LEDs are on.

19. Answer to Q16 [4 marks]

Answer:

The smallest time is 0.015s.

The frequency is $\frac{1}{T} = \frac{1}{2*0.015} = 33.33Hz$

Explanation:

When the smallest time set to 0.015s, people can not recognise the LED is flashing. They just think the LED is always on.

20. Screenshot of the result of Section 15, Part XI (the output screen) [3 marks]

Screenshot showing the effects of the changes made.

Screenshot (PrintScreen or Screenshot):

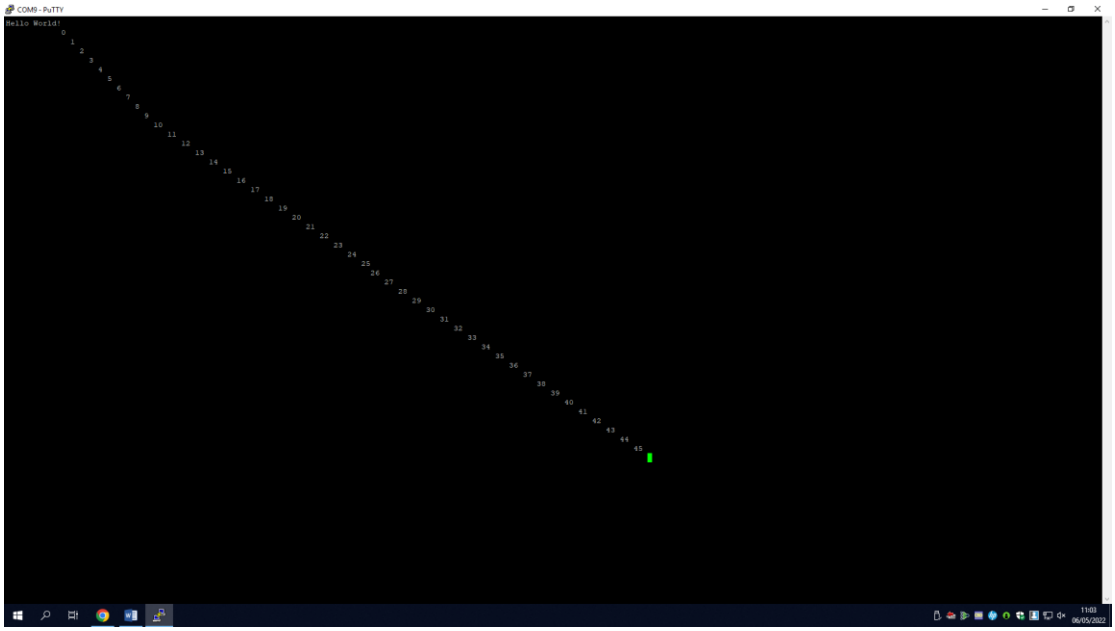


Figure 11: (\n)

Figure 12: (\n\r)

Screenshot (Snipping tool, Preview or equivalent):

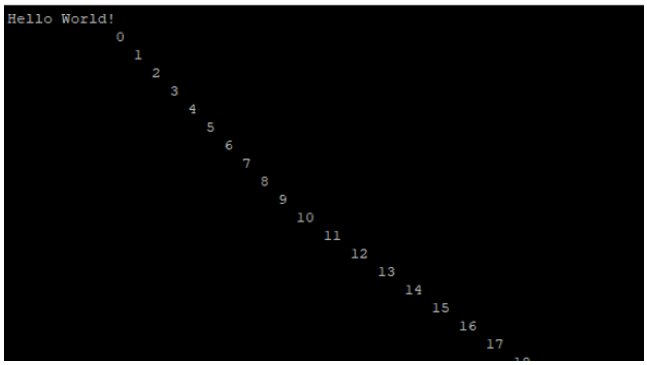


Figure 13

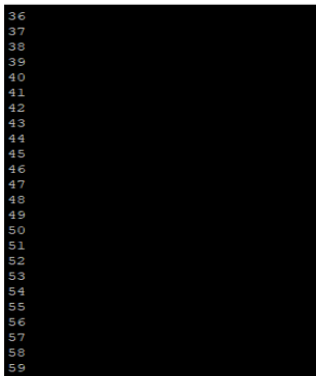


Figure 14

Comment/Explanation:

It can be found that “Hello World” will firstly print on the screen for the both conditions. Then, the program will count number from 0 and increment by 1. For the “\n” condition, it can be seen that each number has different horizonal and vertical position. But for “\n\r” condition, all the numbers are at the left side column. What is more, when speed is set to 115200, the number will print faster than the 9600 condition.

21. <u>The modified code of Section 16</u> (put as text NOT as a screenshot, screenshots of code will receive zero marks) [5 marks]

Code:

```
#include "mbed.h"
#include "SLCD.h"
#include "tsi_sensor.h"

// Very simple program to read the analog slider and print its value
// on the LCD. Also flashes the RED led.
// -- Al Williams

/* This defines will be replaced by PinNames soon */
#if defined (TARGET_KL25Z) || defined (TARGET_KL46Z)
  #define ELEC0 9
  #define ELEC1 10
#elif defined (TARGET_KL05Z)
  #define ELEC0 9
  #define ELEC1 8
#else
  #error TARGET NOT DEFINED
#endif

    TSIAnalogSlider tsi(ELEC0, ELEC1, 40);



DigitalOut gpo(D0);
DigitalOut led1(LED_RED);
DigitalOut led2(LED_GREEN);

    SLCD slcd;
int main()
{
    while (true) {
        float f=tsi.readPercentage();
```

```
        slcd.printf("%1.3f",f);

        if (f==0)
        {
        led1 = 1;
        led2 = 1;
        }
        else if ((0.5>f)&&(f>0))
        led1 =0; // toggle led
        led2=1;

        }
else
        {
            led2 = 0; // toggle led
            led1=1;

            }

    }
}
```

Explanation:

When the finger does not touch the area (f = 0), no LED is illuminated. When, the finger touches the point between the midpoint and the left-hand (f < 0.5), only the red LED is illuminated, while the green LED is illuminated for the other half area.