Experiment 26
Introduction to the ARM Microprocessor
Submission Template of 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:

0x0000025A 220E MOVS r2, #0x0E

0x0000025C 2325 MOVS r3, #0x25

In the sixteen bits, bits 8-10 represent the address of the register. The second number in the machine code represents the register used, since MOVS corresponds to the Opcode code of 00100.

Explanation:

In the sixteen bits, bits 8-10 represent the address of the register. Since the Opcode of MOVS is 00100, for the first instruction, the binary number corresponding to MOVS and Rd is 00100 010, where Rd: 010 corresponds to r2. For the second instruction the binary number corresponding to MOVS and Rd is 00100 011, where Rd: 011 corresponds to r3. For example:

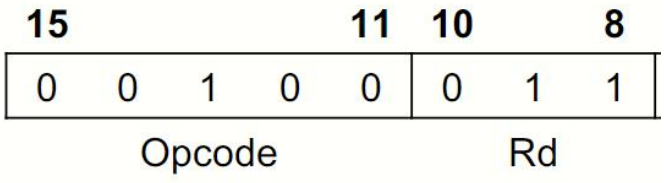

Figure 1: The Opcode and Rd

The above figure corresponds to Rd 011 (3), which means the register used is r3.

2. Answer to Q2 [4 marks]

Answer:

0x0000025A 220E MOVS r2, #0x0E

0x0000025C 2325 MOVS r3, #0x25

The last two digits of machine code gives the number to be moved into that register

Explanation:

The last two digits of the machine code represent the given number. For the 16 bits number represented by MOVS, the last eight bits represent the immediate value. For the first instruction, the immediate value is 0000 1110 (0E). For the second instruction, the immediate value is 0010 0101 (25).

3. Answer to Q3 [4 marks]

Answer:

The value of r4 becomes 0x00000015

Explanation:

The mnemonic(instruction) at address 0x0000025E is ADDS r4, r2, #7. The meaning of this instruction is to add 7 to the present value of r2, and then the value of r4 will be the calculated result. Due to the previous MOVS instruction, the value of r2 is now 0x0000000E. Adding 7 to this value gives 0x00000015, which is now the value of r4.

4. Answer to Q4 [4 marks]

Answer:

Important

Explanation:

For mnemonic ADDS r5,r2,r3, it means that the values of r2 and r3 are added together and the final result is given to r5. If the order is not the same, the result will be different. The Microprocessor does not know which register will be given the final result. Therefore, the microprocessor can only rely on the order of the registers to help it determine this. Although the different order of r2 and r3 here does not affect the

value of r5, for the SUBS instruction, if the order of the registers following the SUBS is different, the microprocessor will perform a different operation, which will result in an error. Also, for example for SUBS r1,r2,r3. If the order of r2 and r3 is swapped, the result is completely different. For SUBS r1,r2,r3, r1=r2-r3, if the order of r2 and r3 is swapped, the final calculation is r1=r3-r2 and the result is completely different. Therefore, the order of register is very important.

5. Answer to Q5 [4 marks]

Answer:

Yes

Explanation:

When setting the value of r2 to 0x00000064 and the value of r3 to 0x0000000A, the changes caused by each step, starting with address IA+0x6(0x0000025E), will be:

0x0000025E 1DD4 ADDS r4,r2,#7

This mnemonic means to add 7 to r2 and give the final result to r4, so that the value of r4 is now 0x0000006B and the values of r2 and r3 remain unchanged at 0x00000064 and 0x0000000A respectively.

0x00000260 18D5 ADDS r5,r2,r3

This mnemonic means to add r2 and r3 and give the final result to r5, so that the value of r5 will be 0x0000006E and the values of r2 and r3 remain unchanged at 0x00000064 and 0x0000000A respectively.

0x00000262 189E ADDS r6,r3,r2

This mnemonic means to add r2 and r3 and give the final result to r6, so that the value of r6 will be 0x0000006E and the values of r2 and r3 remain unchanged at 0x00000064 and 0x0000000A respectively.

0x00000264 1E50 SUBS r0,r2,#1

This mnemonic means to subtract 1 from r2 and give the resulting result to r0, so that the value of r0 will be 0x00000063 and the values of r2 and r3 remain unchanged at 0x00000064 and 0x0000000A respectively.

0x00000266 1AD1 SUBS r1,r2,r3

This mnemonic means to subtract r3 from r2 and give the resulting result to r1, so that the value of r1 will be 0x0000005A and the values of r2 and r3 remain unchanged at 0x00000064 and 0x0000000A respectively.

0x00000268 1A9A SUBS r2,r3,r2

This mnemonic means to subtract r2 from r3 and give the resulting result to r2. Since the result obtained is negative and the negative number are given in 2's complement format. Therefore, the first step is to find the value of -r2 and then to get the final result by r3+(-r2). First, we need to convert each digital to its 1's complement, which gives us 0xFFFFFF9B. Then we add one more to get 0xFFFFFF9C. Therefore, the value of -r2 is 0xFFFFFF9C. After that, we calculate r3+(-r2) to get 0xFFFFFFA6, which is the value of r2 now. Thus, the values of r2 and r3 are 0xFFFFFFA6 and 0x0000000A.

0x0000026A 2211 MOVS r2,#0x11

This mnemonic means change the value of r2 to 0x00000011 and the value of r3 remains.

Each of the above mnemonic has been verified on the computer and meets expectations.

6. Answer to Q6 [4 marks]

Answer:

I set r2 to 0x00000000 and r3 to 0x000000001

All of the results are expected.

Explanation:

0x0000025E 1DD4 ADDS r4,r2,#7

This mnemonic means to add 7 to r2 and give the final result to r4, so that the value of r4 is now 0x00000007 and the values of r2 and r3 remain unchanged at 0x00000000 and 0x00000001 respectively.

0x00000260 18D5 ADDS r5,r2,r3

This mnemonic means to add r2 and r3 and give the final result to r5, so that the value of r5 will be 0x00000001 and the values of r2 and r3 remain unchanged at 0x00000000 and 0x00000001 respectively.

0x00000262 189E ADDS r6,r3,r2

This mnemonic means to add r2 and r3 and give the final result to r6, so that the value of r6 will be 0x00000001 and the values of r2 and r3 remain unchanged at 0x00000000 and 0x00000001 respectively.

0x00000264 1E50 SUBS r0,r2,#1

This mnemonic means to subtract 1 from r2 and give the resulting result to r0, so that the value of r0 will be 0x00000000 (00000001-1=00000000) and the values of r2 and r3 remain unchanged at 0x00000000 and 0x00000001 respectively.

0x00000266 1AD1 SUBS r1,r2,r3

This mnemonic means to subtract r2 from r3 and give the resulting result to r1. Since the result obtained is negative and the negative number are given in 2's complement format. Therefore, the first step is to find the value of -r3 and then to get the final result by r2+(-r3). First, we need to convert each digital to its 1's complement, which gives us 0xFFFFFFFE. Then we add one more to get 0xFFFFFFFF. Therefore, the value of -r3 is 0xFFFFFFFF. After that, we calculate r2+(-r3) to get 0xFFFFFFFF, which is the value of r1 now. Then, the values of r2 and r3 are 0x00000000 and 0x00000001 unchanged.

0x00000268 1A9A SUBS r2,r3,r2

This mnemonic means to subtract r2 from r3 and give the resulting result to r2, so that the value of r1 will be 0x00000001 and the values of r2 and r3 remain unchanged at 0x00000000 and 0x00000001 respectively.

0x0000026A 2211 MOVS r2,#0x11

This mnemonic means change the value of r2 to 0x00000011 and the value of r3 remains.

7. Answer to Q7 [4 marks]

Answer:

"And" function is performed because the mnemonic ANDS r4,r4,r3 in 0x00000272 is executed.

Explanation:

Before the AND function is executed, the values in r2 and r3 r4 are 0x00000011 and 0x00000101, 0x00000011 respectively. The mnemonic ANDS r4,r4,r3 means that the logical AND function is used between r4 and r3, and the result is given back to r4 . The truth table for logical AND is:

Table 1: The truth table for AND

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

Because the value in A is 0x00000011 and the value in B is 0x00000101,

Therefore, the final result for the logic function (AND) performed bit by bit will be:

For A and B and the result C:

A: 0 0 0 0 0 0 1 1

B: 0 0 0 0 0 1 0 1

C: 0 0 0 0 0 0 0 1

As a result, after running the logic function AND, the value of r4 becomes 0x00000001.

8. Answer to Q8 [4 marks]

Answer:

The order of the BIC is important. The order of the registers corresponding to the other operation is not important, e.g. ANDS,ORRS,EORS.

Explanation:

For logical AND AND,OR,EOR, the operation is the same and the result is the same no matter who registers first or second, and the order of the registers does not affect the result. For example, the truth table for logical OR is as follows.

Table 2: The logical OR

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

Changing the order of registers is only equivalent to changing the order of A and B, but the order of A and B does not affect the final answer because the final output C is independent of the order of A and B, which is also can be seen above. However, for the logical BICS, the order of registers affects the final result because the output of BIC is $\boldsymbol{C = A.\overline{B}}$ This means that the output of the BIC is related to the order of the inputs, which means that the output of the BIC is related to the order of the register.

9.Answer to Q9 [4 marks]

Answer:

The value of program counter becomes 0x00000286

Explanation:

Since the program executes the branch instruction: B 0x00000286, this means that the next execution of the program is at 0x00000286. Since the value in the program counter is the address of the next execution of the program. Therefore, the value in the program counter will also be 0x00000286.

10. Answer to Q10 [5 marks]

Answer:

0x00000005

Explanation:

Since every time branch B 0x00000286 is executed, the program will execute ADDS r2, r2, #1 once. Therefore, each time the branch instruction is executed, the value of r2 will be increased by 1. If the program executes the branch instruction 5 times, the value of r2 will increase by 5 because the ADDS instruction is before the branch instruction. Before the first branch instruction is executed, r2 has already changed to 0x00000001, so the result of r2 after five runs is 0x00000005. If the change in r2 before the first branch instruction is disregarded, the result after five runs is 0x00000004.

11. Answer to Q11 [4 marks]

Answer:

The value in r2 is 0x11989DED for 20s and 0x21528DCF for 40s. 0x21528DCF is approximately twice as large as 0x11989DED.

Explanation:

This is because every time the branch instruction is executed, the system will also execute ADDS r2,r2,#1 once, which will add 1 to r2. The system runs double the time means that the program executes twice as many branch instructions and executes twice as many ADDS r2,r2,#1, which doubles the value of r2.

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

Screenshot (PrintScreen or Screenshot):

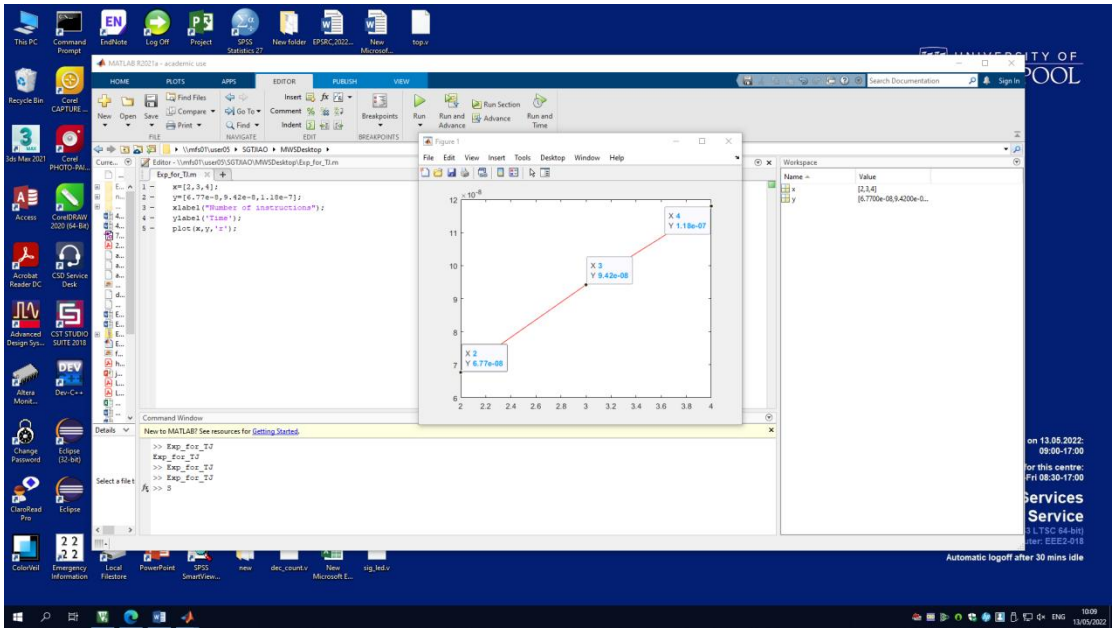

Figure 2: The full screenshot of the matlab result

Screenshot (Snipping tool, Preview or equivalent):

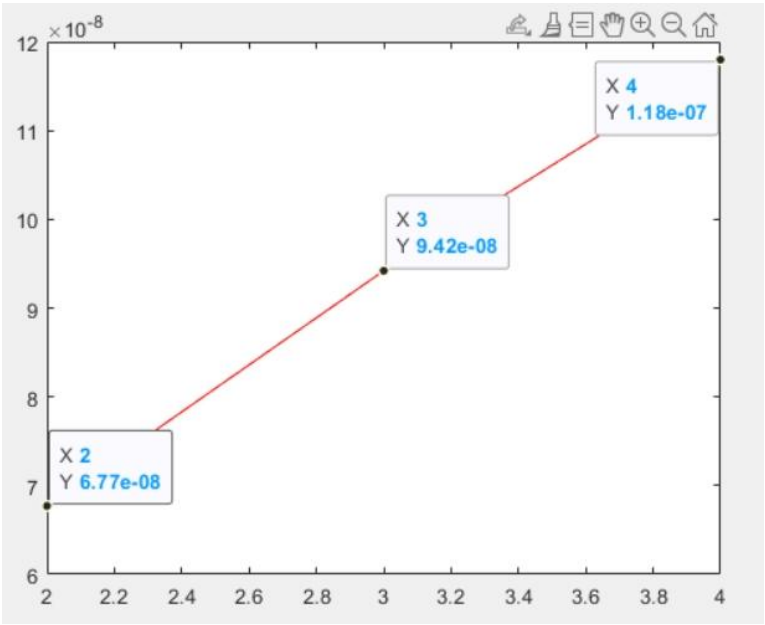

Figure 3:The final result of the executed time

Comment/Explanation:

When the system is running only two instructions (1 branch and 1 ADDS), the value of r2 obtained by timing 20 seconds is 0x11989DED, and converting this hexadecimal number to decimal yields 295214573, so the time required for the system to execute a loop is about $6.77 * 10^{-8}$s.

When the system is running three instructions (1 branch and 2 ADDS), the value of r2 obtained by timing 20 seconds is 0x0CA6C5EA, and converting this hexadecimal number to decimal yields 212256234, so the time required for the system to execute a loop is about $9.42 * 10^{-8}$s.

When the system runs four instructions (1 branch and 3 ADDS), the value of r2 obtained by timing 20 seconds is 0x0A14510A, and converting this hexadecimal number to decimal yields 169103626, so the time required for the system to execute a loop is about $1.18 * 10^{-7}$s.

Plotting the above times and the corresponding number of instructions on a graph shows that the three points are nearly on a straight line. In the next question, we will calculate the time it takes for each instruction to execute.

13. Answer to Q12 [4 marks]

Answer:

$2.65 * 10^{-8}$s

Explanation:

In the second measurement, the time for one ADDS and one branch instruction was measured. In the third measurement, the time for two ADDS and one branch instruction was measured. The time difference between these two measurements is the time it takes for the system to run one add. Since the add instruction takes one clock cycle to execute, the time taken by one clock cycle will be $9.42 * 10^{-8} - 6.77 * 10^{-8} = 2.65 * 10^{-8}$s.

14. Answer to Q13 [4 marks]

Answer:

$4.12 * 10^{-8}$s

Explanation:

In the first measurement, the time taken to run a branch instruction and an ADDS instruction was measured, and the time taken to run an ADDS instruction was calculated in the last question. Therefore, the time difference (branch time) between the two times is:$6.77 * 10^{-8} - 2.65 * 10^{-8} = 4.12 * 10^{-8}$s

15.Screenshot of the result of Section 10 [3 marks]

Screenshot (PrintScreen or Screenshot):

Below is the screenshot of the instructions that caused the flags to change because there are too many the screenshots which can cause flag changing. Only one of these cases is listed in this section, the rest of the screenshots are in the appendix

For the original result for the system executing the instruction 0x000002A2 MOVS r4,r2

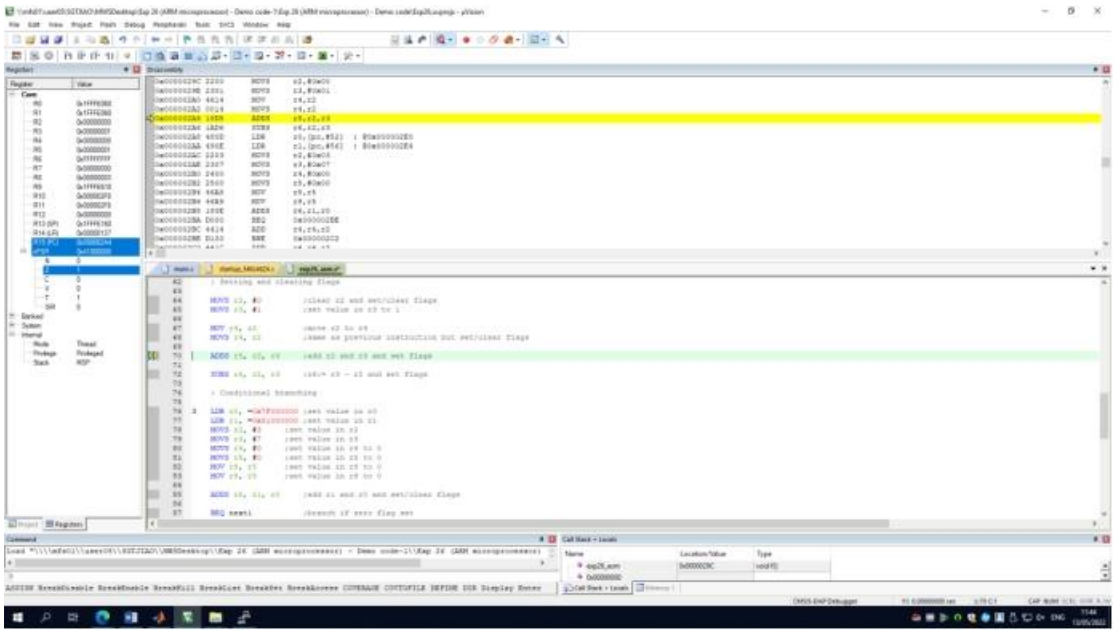

Figure 4: The full screen shot for the executing the instruction MOVS r4,r2

Screenshot (Snipping tool, Preview or equivalent):

For the original result for the system executing the instruction 0x000002A2 MOVS r4,r2

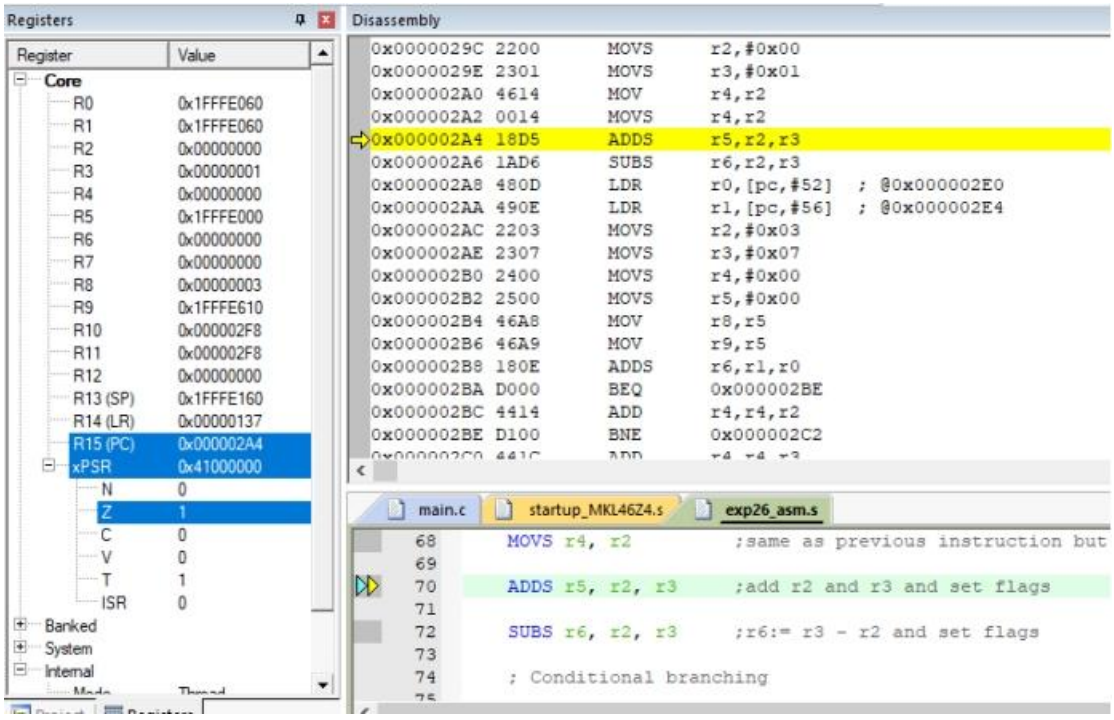

Figure 5: The screen shot for the address 0x000002A2

Comment/Explanation:

When the system runs from 0x0000029C to 0x000002A8, the values of r2 and r3 are reset to 0x00000000 and 0x00000001. The flag changes as follows.

1.When the system executes 0x000002A2 MOVS r4,r2, the zero flag will be set because the value of r4 becomes 0.

2.When the system executes 0x000002A4 ADDS r5, r2, r3, the zero flag will be clear because the value of r5 is no longer 0.

3.When the system executes 0x000002A6 SUBS r6,r2,r3, the negative flag will be set since the result of the calculation is negative.

When the system runs from 0x0000029E to 0x000002A8 and the value of r2 is set to 0xFFFFFFFF and the value of r3 is still 0x00000001. The flag changes as follows:

1.When the system executes 0x000002A2 MOVS r4,r2, the negative flag will be set because the value of r4 becomes negative.

2.When the system executes 0x000002A4 ADDS r5, r2, r3, the carry flag will be set because the result of the calculation exceeds 32bits, and the last 32 bits are all 0, therefore, zero flag will also be set and negative flag will be clear.

3.When the system executes 0x000002A6 SUBS r6,r2,r3, the negative flag will be set because the result of the calculation is negative, and the zero flag will be clear

When the system runs from 0x0000029E to 0x000002A8 and the value of r2 is set to 0x7FFFFFFF and the value of r3 is still 0x00000001. The changes in Flag are as follows:

1.When the system executes 0x000002A4 ADDS r5, r2, r3, the negative flag will be set because the value of r5 becomes negative, and the overflow flag will also be set because two positive numbers are added together but a negative number is obtained.

2.When the system executes 0x000002A6 SUBS r6,r2,r3, the carry flag will be set because the result of the calculation exceeds 32bits, and the overflow and negative flag will be clear.

16. Answer to Q14 [4 marks]

Answer:

0x000002BA D000 BEQ 0x000002BE

0x000002C2 D200 BCS 0x000002C6

0x000002CE D500 BPL 0x000002D2

0x000002D6 D700 BVC 0x000002DA

Explanation:

For instruction 0x000002BA D000 BEQ 0x000002BE:

Since the system executes the instruction of MOVS r4, #0x00 in address 0x000002B0, the zero flag will be set. And when the system runs to BEQ 0x000002BE, the zero flag is still set, so the instruction of BEQ 0x000002BE will be executed.

For instruction 0x000002C2 D200 BCS 0x000002C6:

Since the system is executing ADDS r6,r1,r0 in address 0x000002B8, the result of the calculation is out of range, so the carry flag will be set. Until the system runs to BCS 0x000002C6, the carry flag is still set, so BCS 0x000002C6 will be executed.

For instruction 0x000002CE D500 BPL 0x000002D2：

Since the negative flag is always clear, the system will execute BPL 0x000002D2.

For instruction 0x000002D6 D700 BVC 0x000002DA:

Since the overflow flag is always clear, the system will execute BVC 0x000002DA.

17. Answer to Q15 [4 marks]

Answer:

Yes, the following conditional branch instructions are executed:

0x000002BE D100 BNE 0x000002C2

0x000002C6 D300 BCC 0x000002CA

0x000002CE D500 BPL 0x000002D2

0x000002D6 D700 BVC 0x000002DA

Explanation:

For instruction 0x000002BE D100 BNE 0x000002C2:

Since the value of r4 is no longer 0, which means the zero flag is clear, the system will execute BNE 0x000002C2.

For instruction 0x000002C6 D300 BCC 0x000002CA:

Because the system's carry flag has not been set, the system will execute BCC 0x000002CA

For instruction 0x000002CE D500 BPL 0x000002D2:

Again, because the system's negative flag is always clear, the system will execute BPL 0x000002D2.

For instruction 0x000002D6 D700 BVC 0x000002DA:

Since the overflow flag is always clear, the system will execute BVC 0x000002DA.

18. 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):

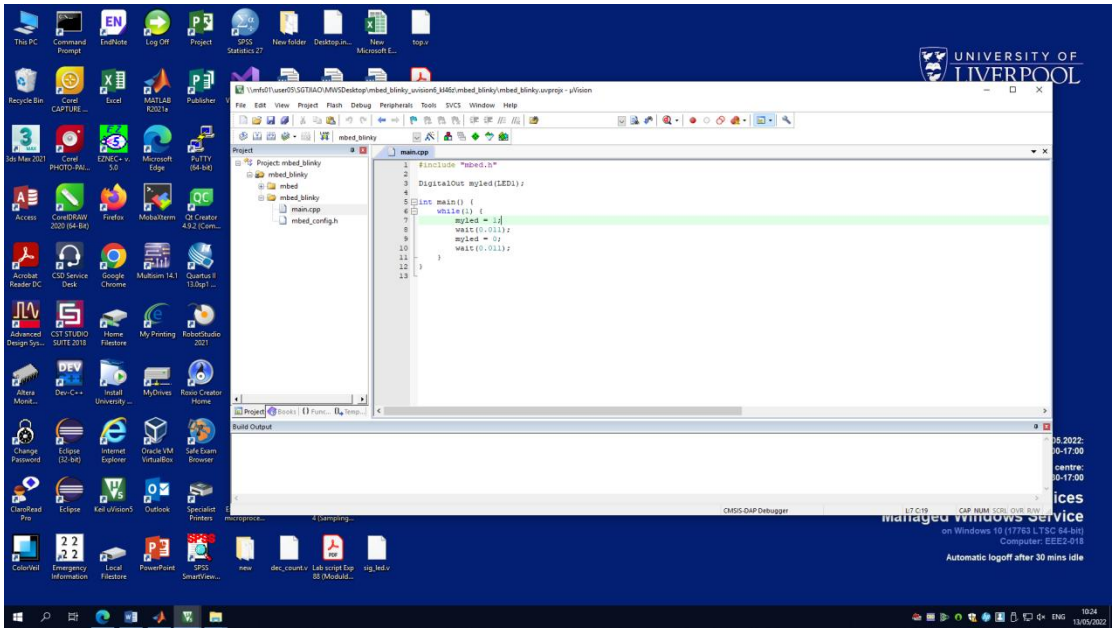

Figure 6: The full screen shot for successfully exporting from the Mbed compiler

Screenshot (Snipping tool, Preview or equivalent):

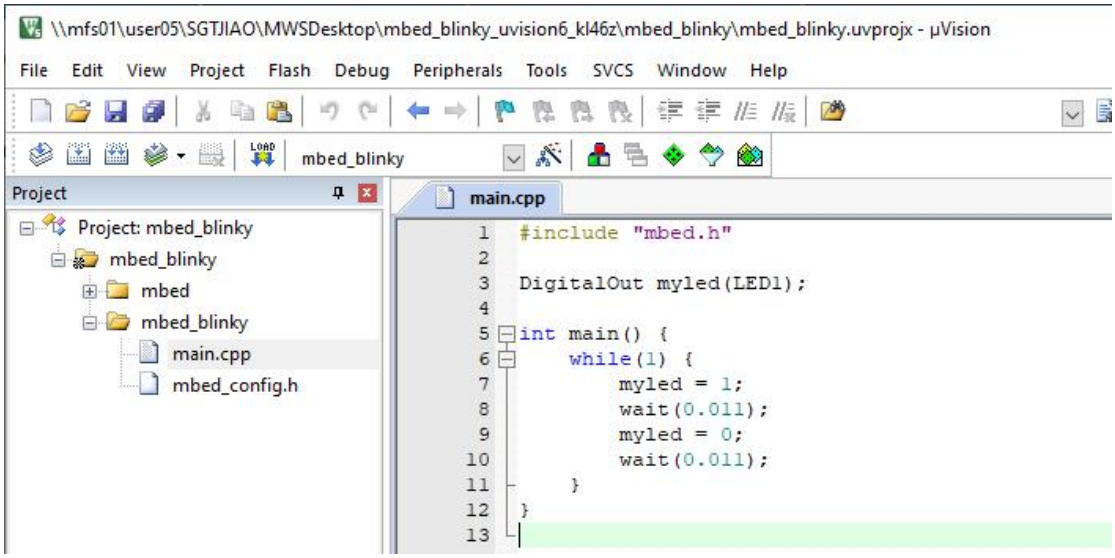

Figure 7: The screenshot for successfully exporting from the Mbed compiler

19. 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 myled(LED1);
DigitalOut myled1(LED2);
int main() {
    while(1) {
        myled = 1;
        myled1 = 0;
        wait(0.02);
        myled = 0;
        myled1 = 1;
        wait(0.02);
    }
}
```

Explanation:

I add two LEDs (red and green) to my program, where “myled” refers to the red LED and “myled1” refers to the green LED. when myled is 0, myled1 will be 1. when myled is 1, myled1 will be 0, so that the two LEDs blink alternately. At the same time, by changing the wait time, I also adjusted the frequency of the LED blinking. The program is now used to blink every 0.02 seconds

20. Answer to Q16 [4 marks]

Answer:

0.011s is the interval between LED flashing and off and it is also the interval between LED off and flashing

Explanation:

$$f = \frac{1}{T*2} = \frac{1}{0.011*2} \approx 46Hz$$

This time is the minimum time interval at which people can observe the flashing. If this is converted to a frequency, which is the maximum frequency that can be recognised by the human eyes, once the flash frequency of the led is greater than this value, people cannot distinguish whether the led is flashing or not. A search of the literature shows that this value is consistent.

21. 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):

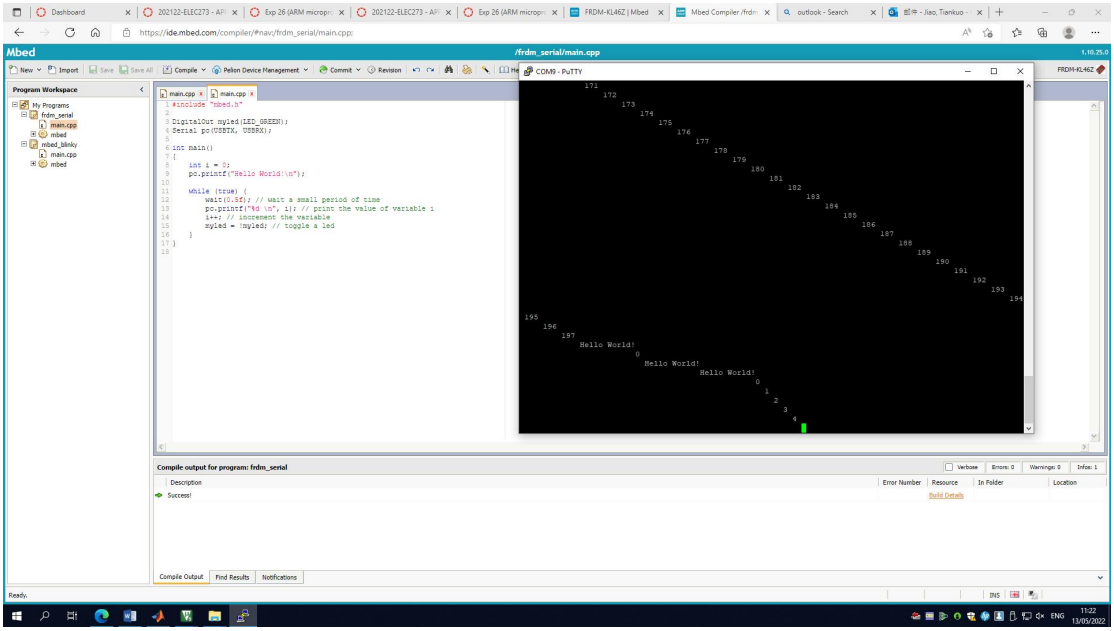

Figure 8: The original result for the serial communication

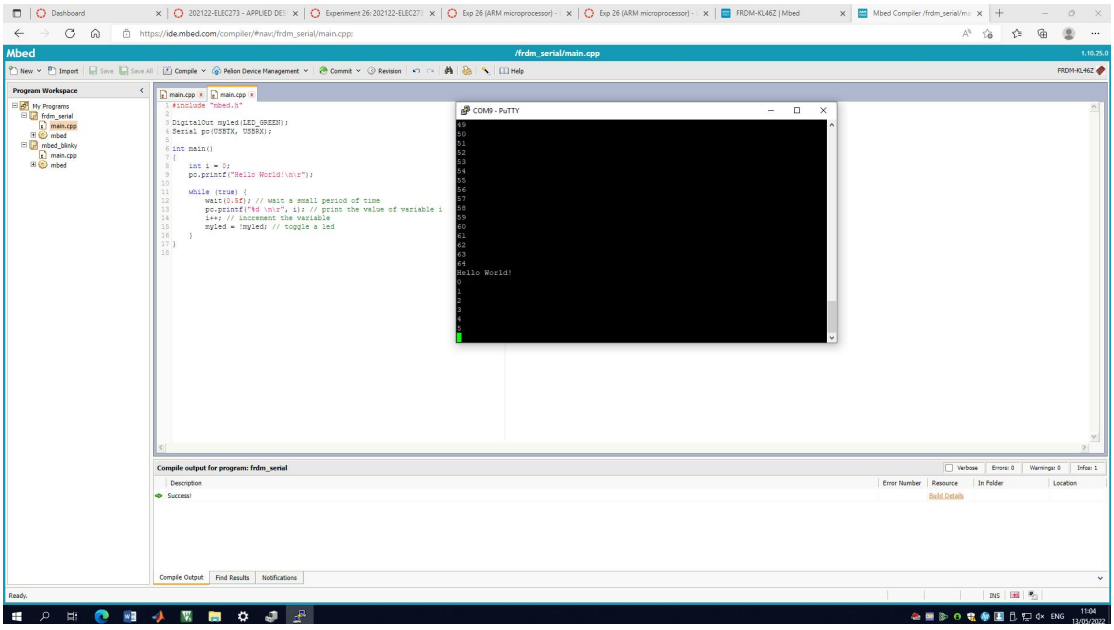

Figure 9: The final result for the serial communication after changing the baud speed

Screenshot (Snipping tool, Preview or equivalent):

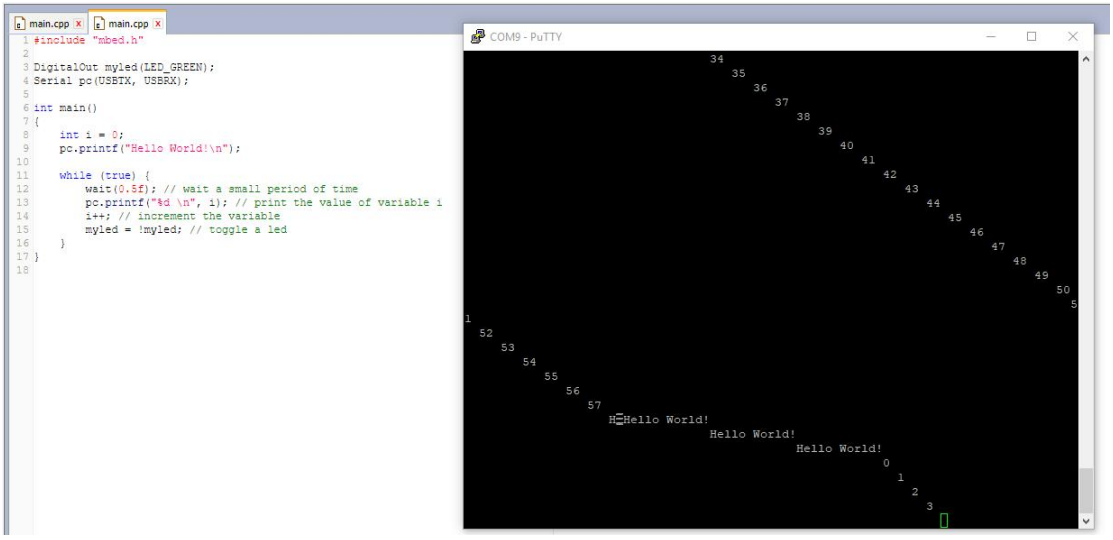

Figure 10: The original result for the serial communication

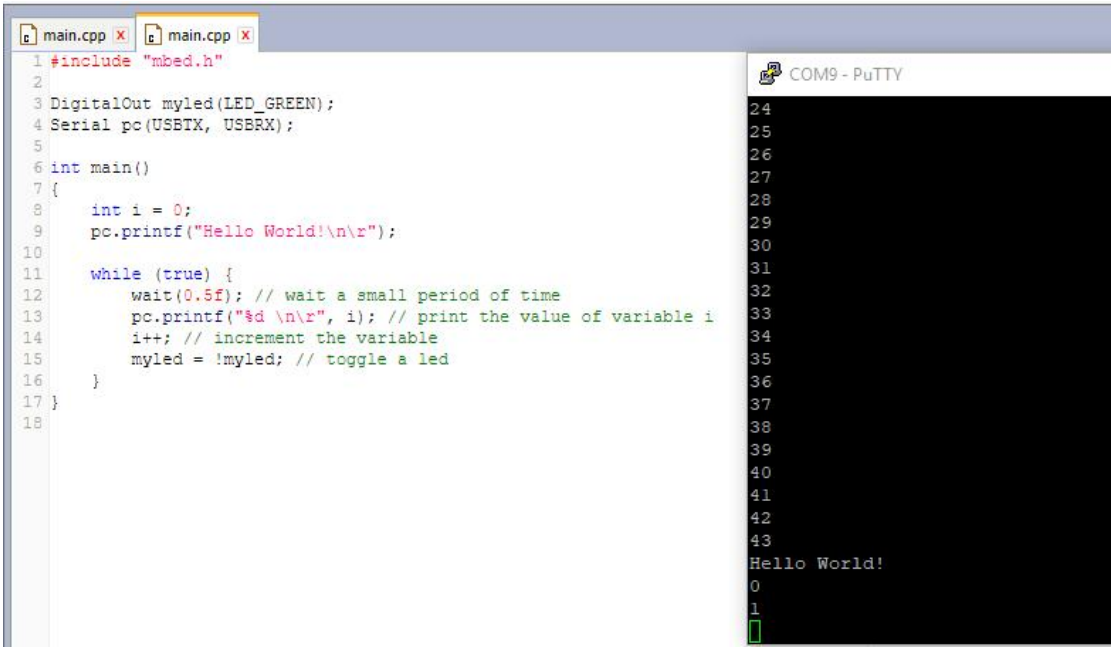

Figure 11: The final result for the serial communication after changing the baud speed

Comment/Explanation:

When adjusting the baud speed to 115200, the time interval between each occurrence of digits becomes shorter and the system runs faster. This is because the transmission rate of the system is doubled after changing the baud speed, from a baud rate of 9600

to baud rate of 115200. Also, I adjusted the baud rate to 115200 in the "putty" interface. Also, as required by the question, I also change the "printf" in the main program to use \n\r instead of using \n. Therefore, all digits and word "Hello World!" will all remain 。 in one vertical row.

22. The modified code of Section 16 (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 myled(LED1);
DigitalOut myled1(LED2);


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

```
35.         slcd.printf("%1.3f",f);
36.         if(f==0)
37.         {
38.         myled1=1;
39.         myled=1;
40.         }
41.         else if(f<0.5 &f>0)
42.         {
43.         myled=1;
44.         myled1=0; \\red
45.         }
46.         else
47.         {
48.         myled=0; \\green
49.         myled1=1;
50.         }
51.
52.     }
53. }
```

Explanation:

First, according to the lab script, we need to set two states, the green LED will be illuminated when the right half of the slider is touched by the hand, and the red LED will be illuminated when the left half of the slider is touched by the hand. Moreover, to make the situation more completed, I assume that both LED will not be illuminated when the whole slider is not touched by the hand.

Therefore, my program sets three conditions, which include:

1.when f==0, which is the case when the hand does not touch the slider.

2.When f<0.5 & f>0, the red LED will be illuminated and the other LED will be off.

3.For other values of f, the green LED will be illuminated and the other LED will be off.

As the two LEDs here are triggered low, “myled” is used to refer to the red LED, when “myled” is 0 the red LED will be illuminated and when “myled” is 1, the red LED will be off.

For the green LED, “myled1” is used to refer to it. When “myled1” is 0, the green LED will be illuminated and when “myled” is 1, the green LED will be off.

Alternatively, when both myled and myled1 are 1, both LEDs will be off, which is used for the case that there is without touching.

Appendix:

For the original results from 0x0000029C to 0x000002A8

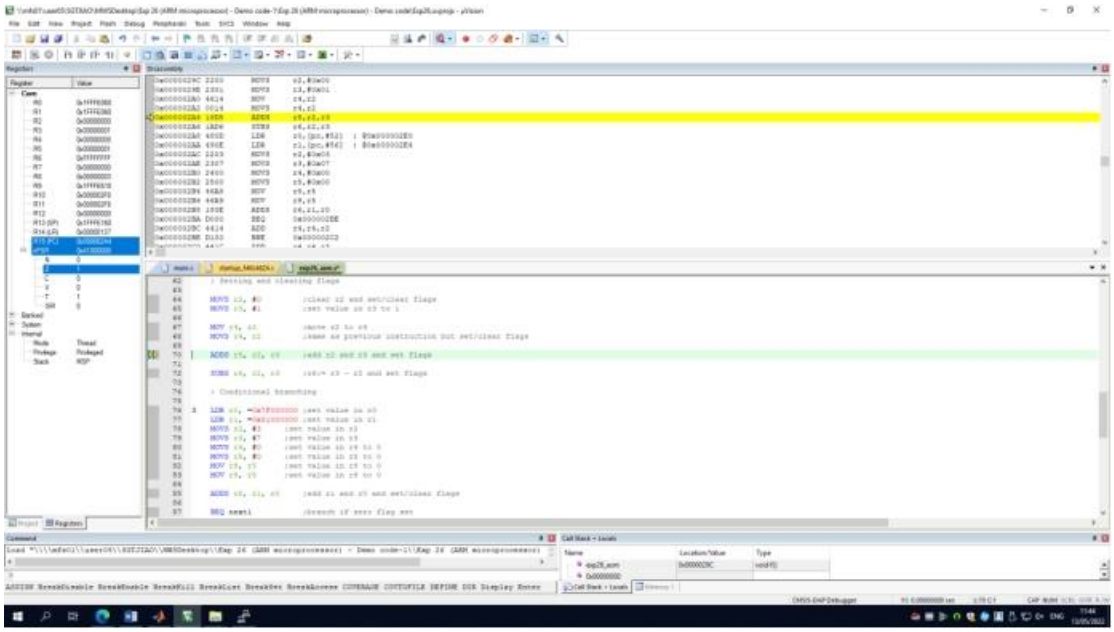

Figure 12: The full screen shot for the address 0x000002A2

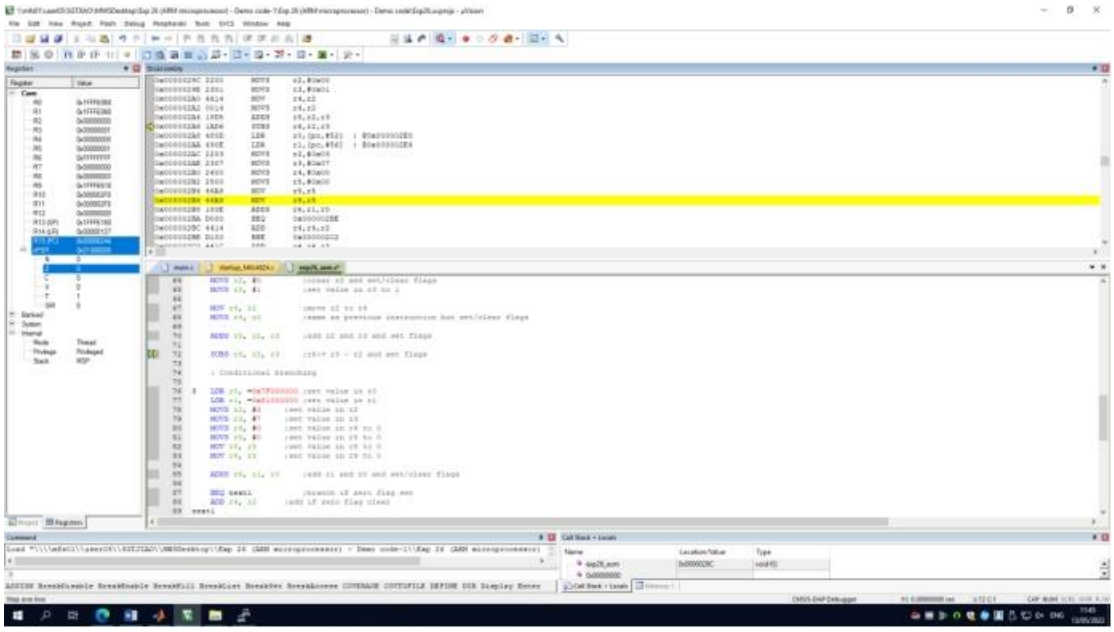

Figure 13: The full screen shot for the address 0x000002A4

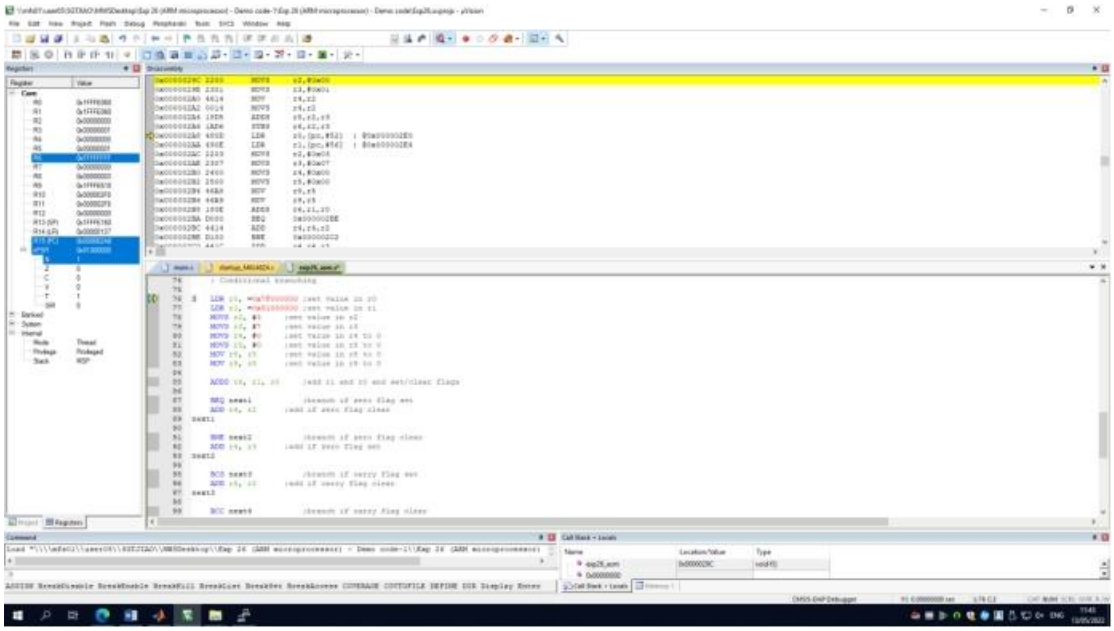

Figure 14: The full screen shot for the address 0x000002A6

After changing the value of r2 to 0xFFFFFFFF

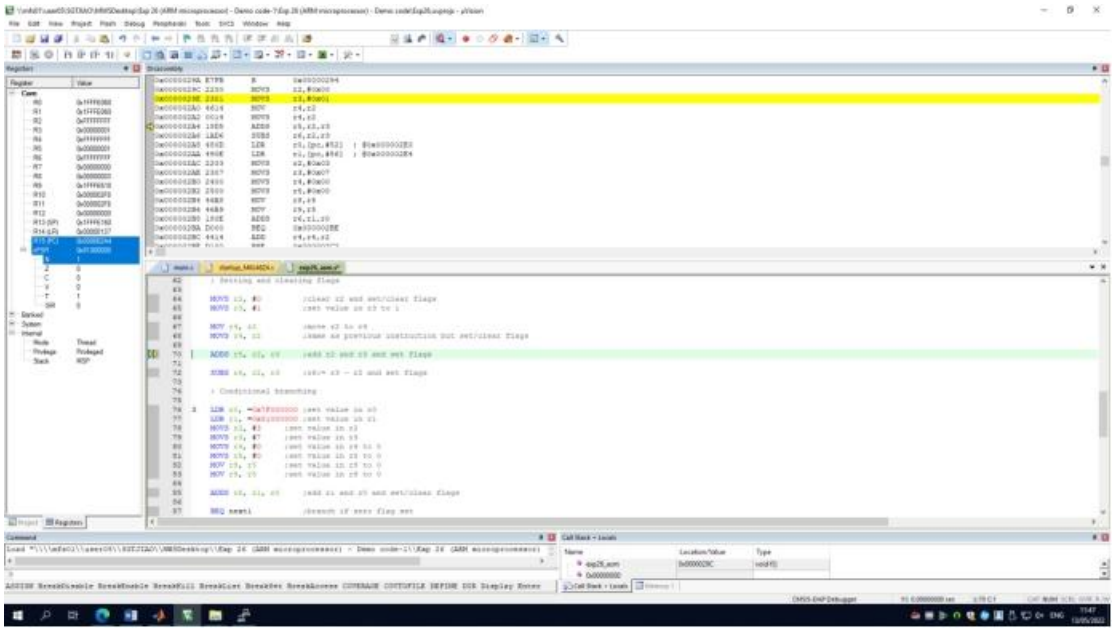

Figure 15: The full screen shot for the address 0x000002A2

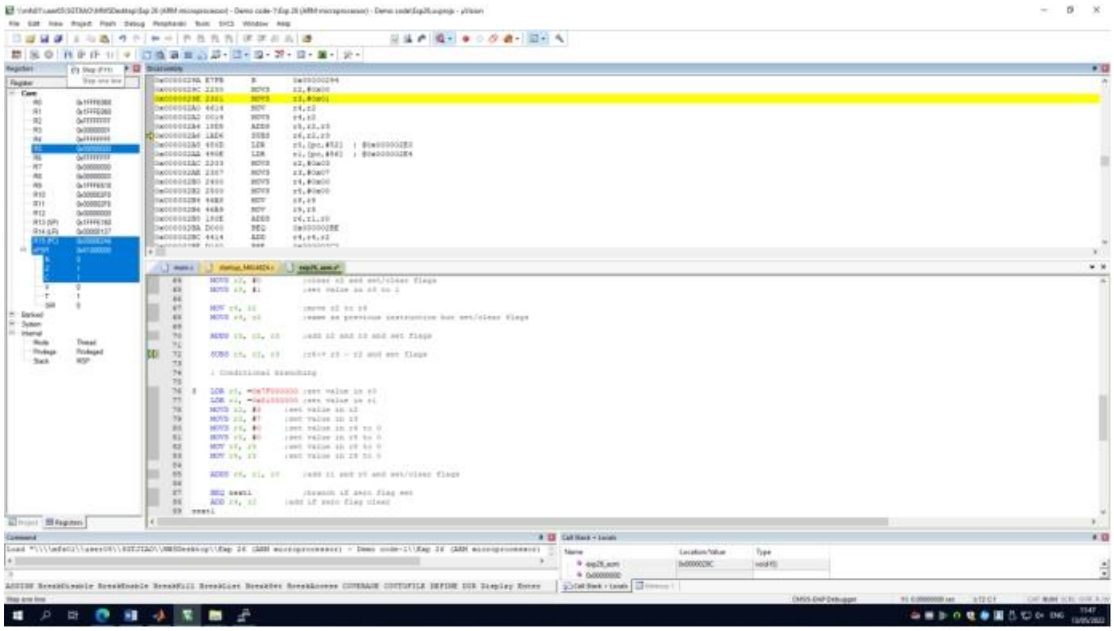

Figure 16: The full screen shot for the address 0x000002A4

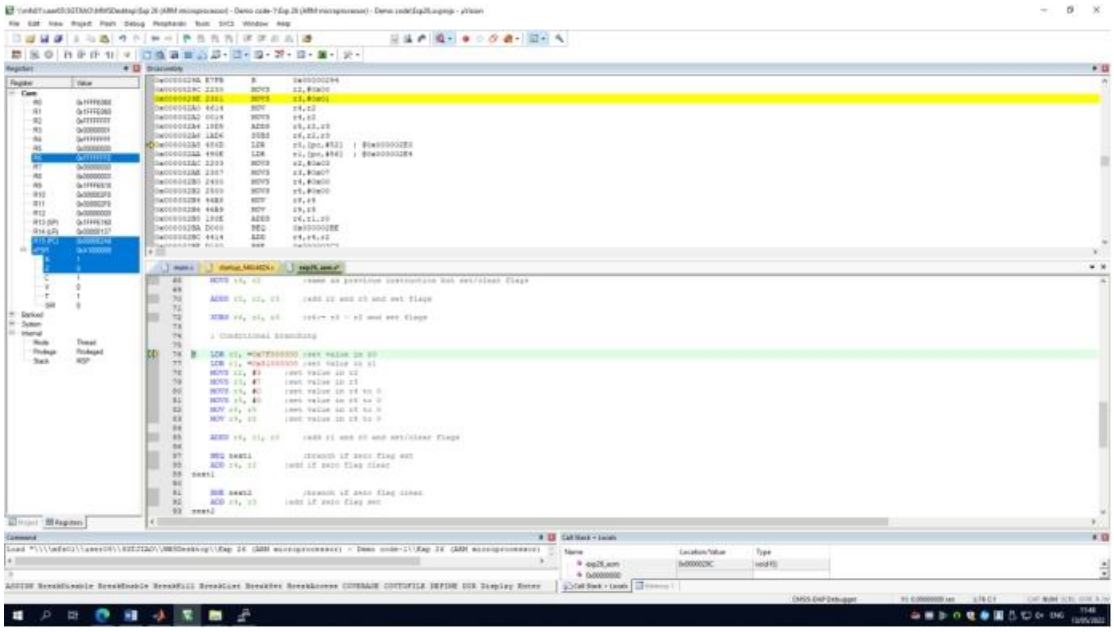

Figure 17: The full screen shot for the address 0x000002A6

After changing the value of r2 to 0x7FFFFFFF

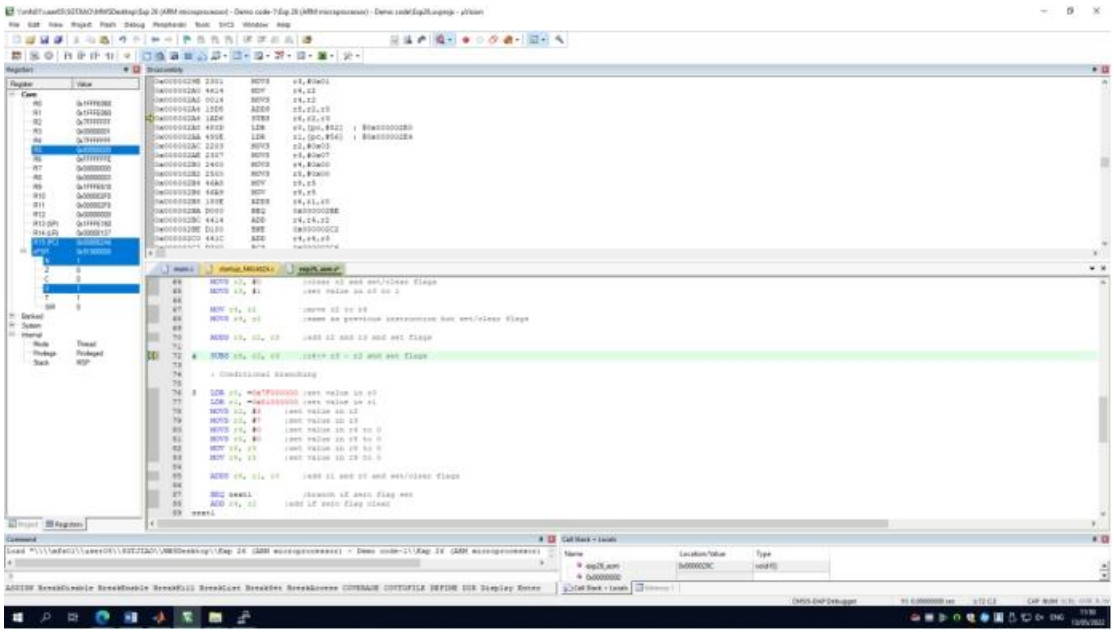

Figure 18: The full screen shot for the address 0x000002A4

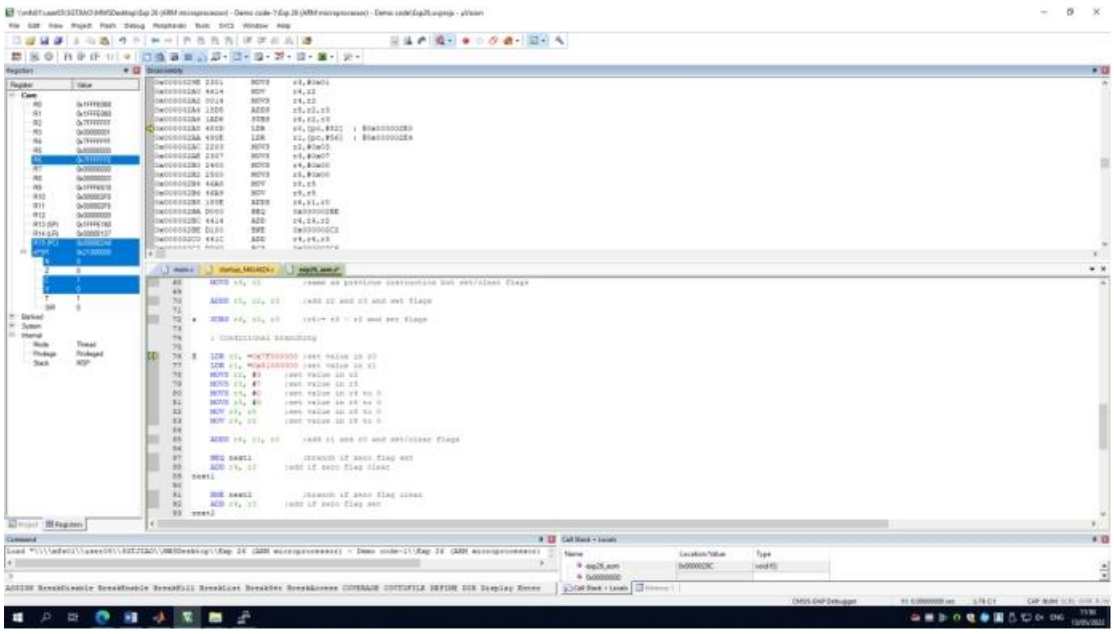

Figure 19: The full screen shot for the address 0x000002A6

Screenshot (Snipping tool, Preview or equivalent):

The original results from 0x0000029C to 0x000002A8

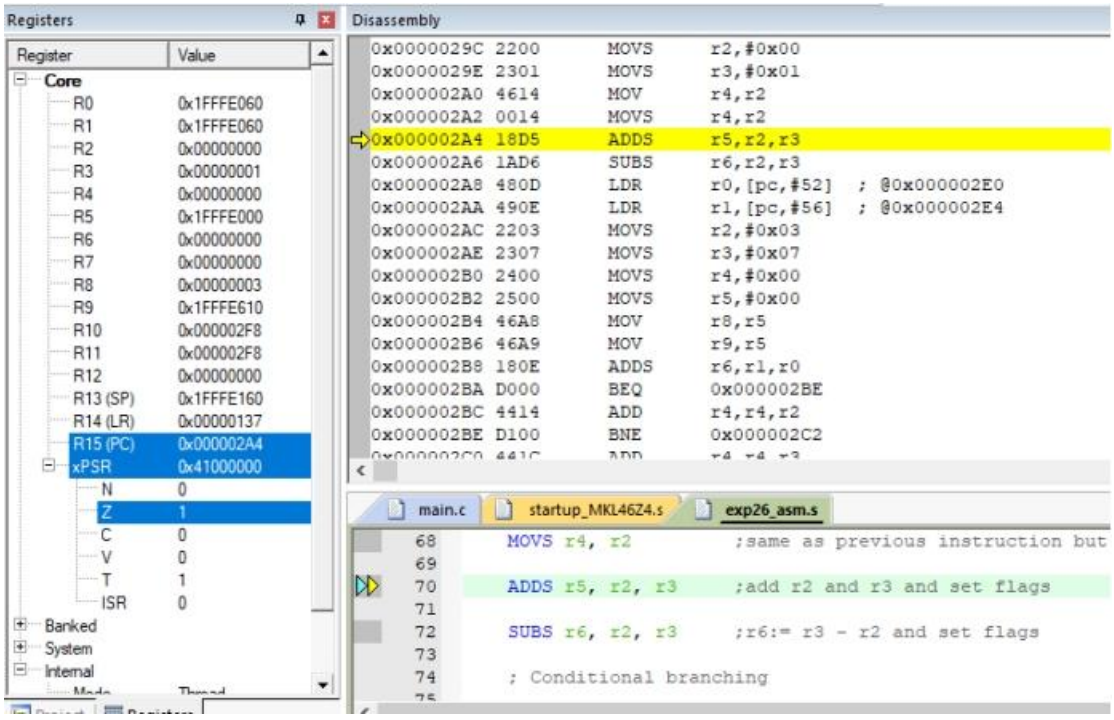

Figure 20: The screen shot for the address 0x000002A2

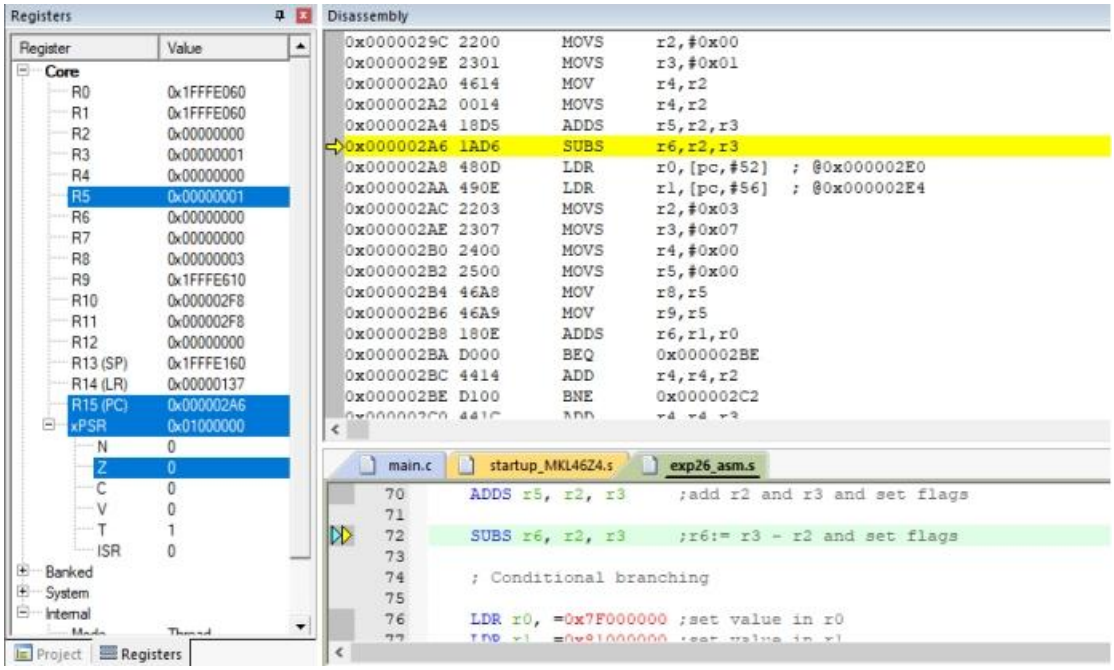

Figure 21: The screen shot for the address 0x000002A4

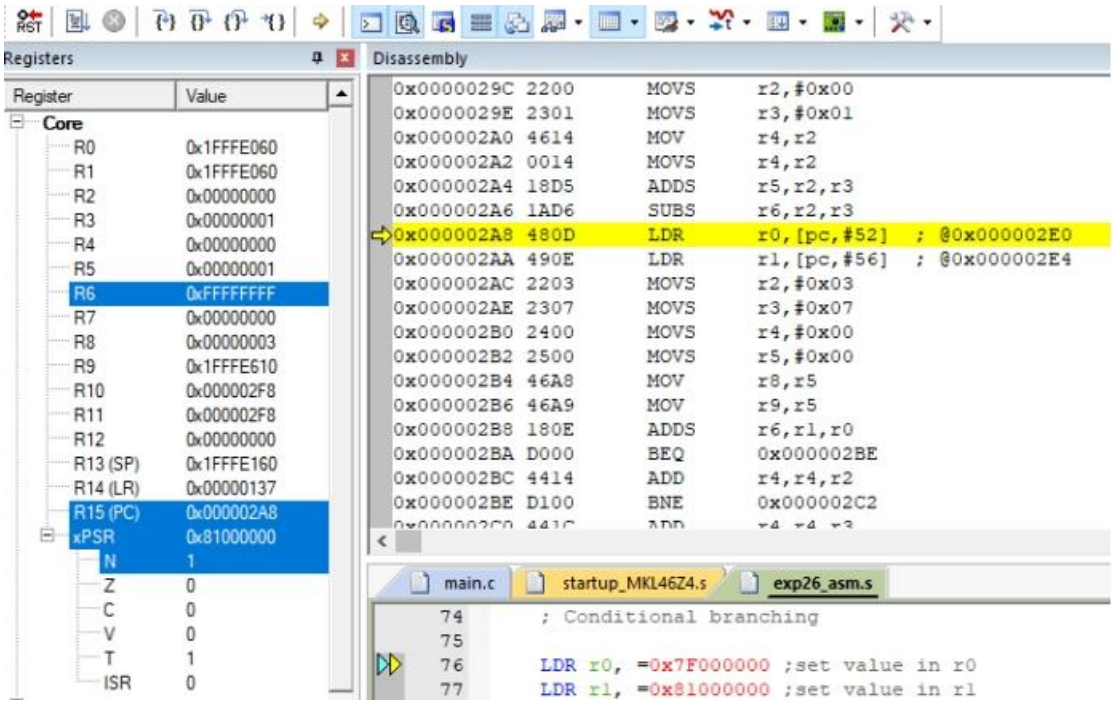

Figure 22: The screen shot for the address 0x000002A6

After changing the value of r2 to 0xFFFFFFFF

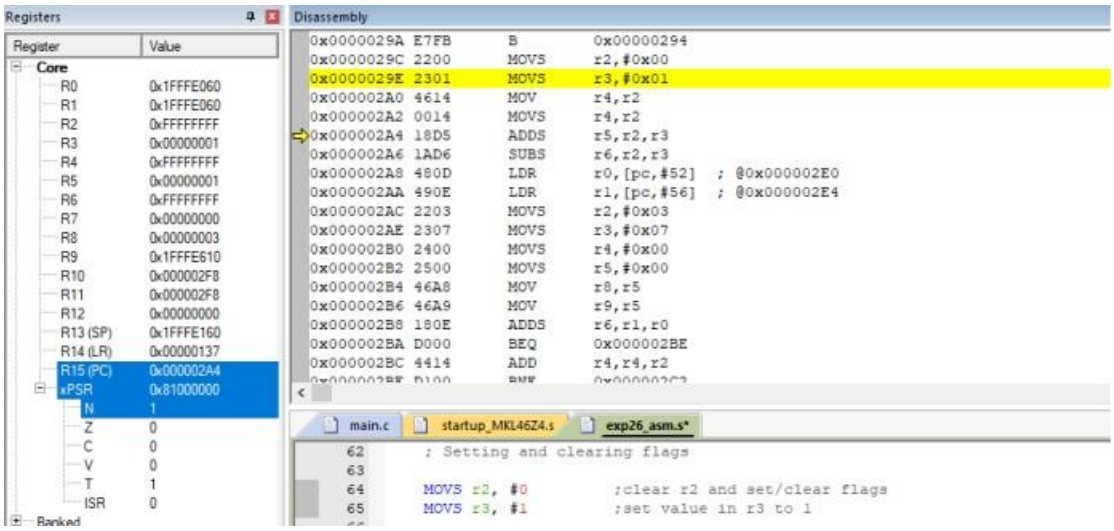

Figure 23: The screen shot for the address 0x000002A2

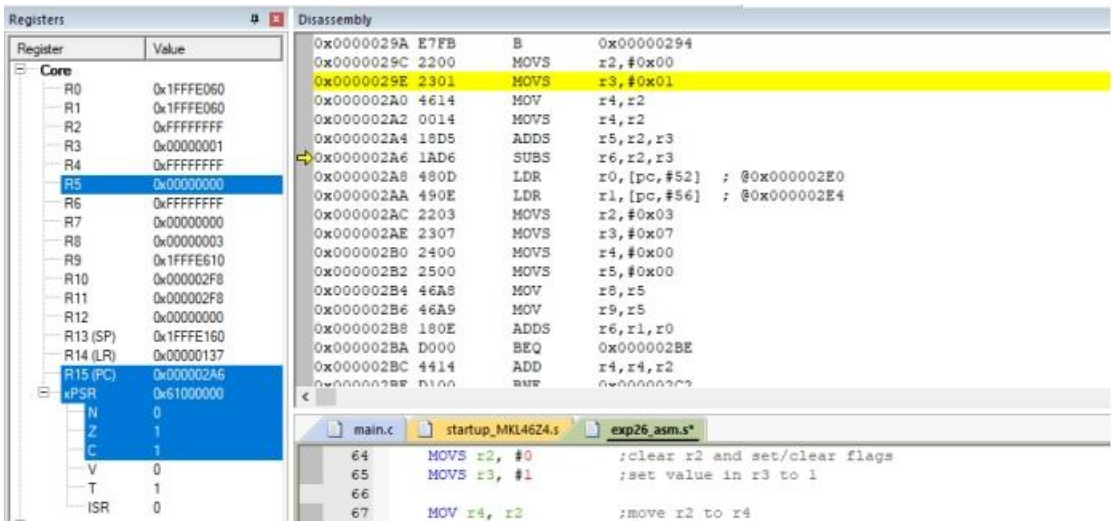

Figure 24: The screen shot for the address 0x000002A4

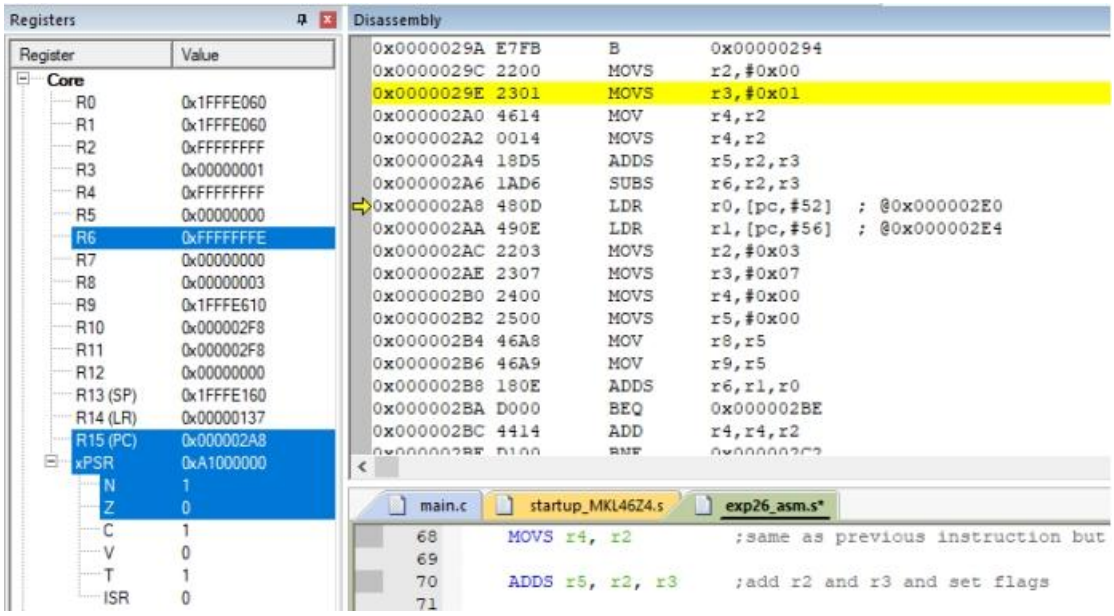

Figure 25: The screen shot for the address 0x000002A6

After changing the value of r2 to 0x7FFFFFFF

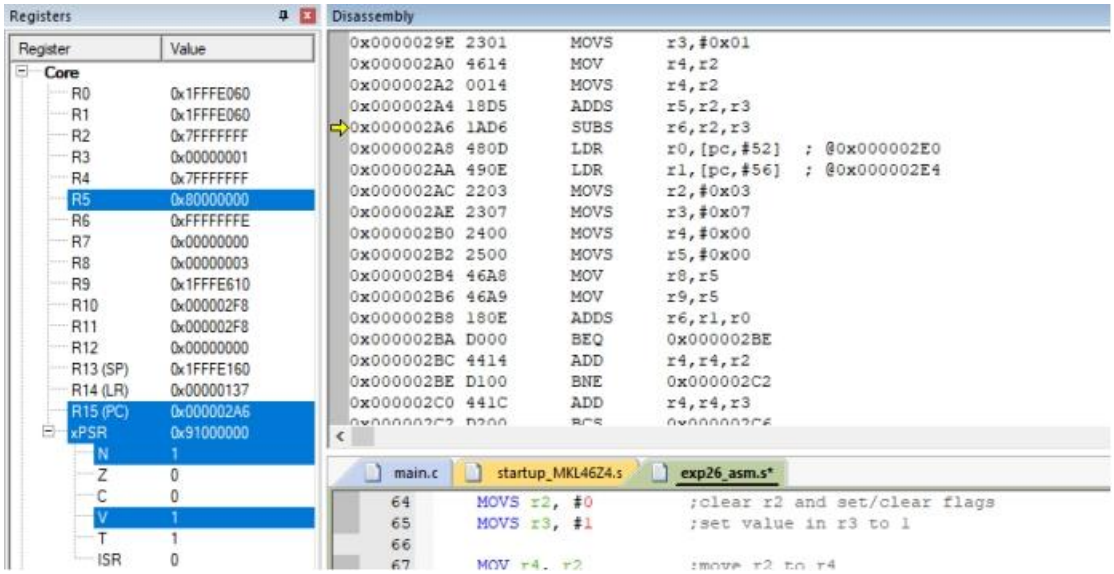

Figure 26: The screen shot for the address 0x000002A4

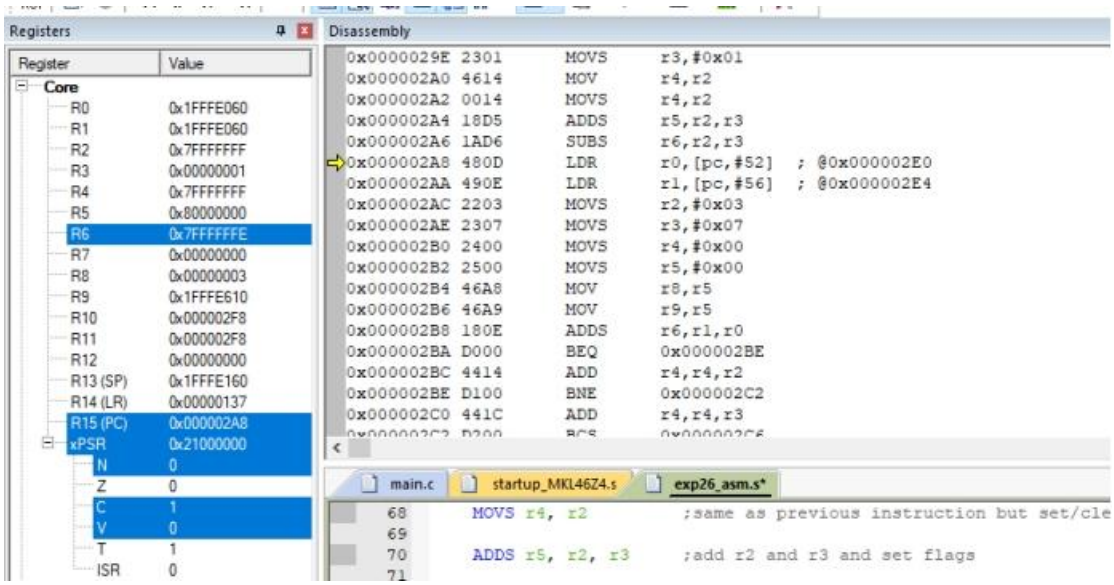

Figure 27: The screen shot for the address 0x000002A6