Department of Electrical Engineering & Electronics

Experiment 26
Introduction to the ARM Microprocessor
Submission Template of Experimental Results

February 2023, Ver. 7.1

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 Canvas. 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 (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) 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 bit of the machine code.

Explanation:

We know from the experimental script that the second part is the machine code, which is usually a four-digit hexadecimal number.

```
0x00000276 220E          MOVS       r2,#0x0E
0x00000278 2325          MOVS       r3,#0x25
```

Register r3 is given by the 2nd digit

0x2372

And for this machine code, the second bit represents the serial number of the register. This can also be seen in lecture 19. I found that when operating directly on numbers, the second bit of the machine code represents the register used, because only one register is used at that point. In other cases it is not always the second bit that represents the register's serial number.

2. Answer to Q2 [4 marks]

Answer:

The last two digits of the machine code.

Explanation:

When operating directly with numbers, the second bit represents the serial number of the register, and the last two bits represent the number being deposited into the register.

The value 114 is given in the least significant byte

0x2372

3. Answer to Q3 [4 marks]

Answer:

0x00000015 (0xE + 7 = 0x15)

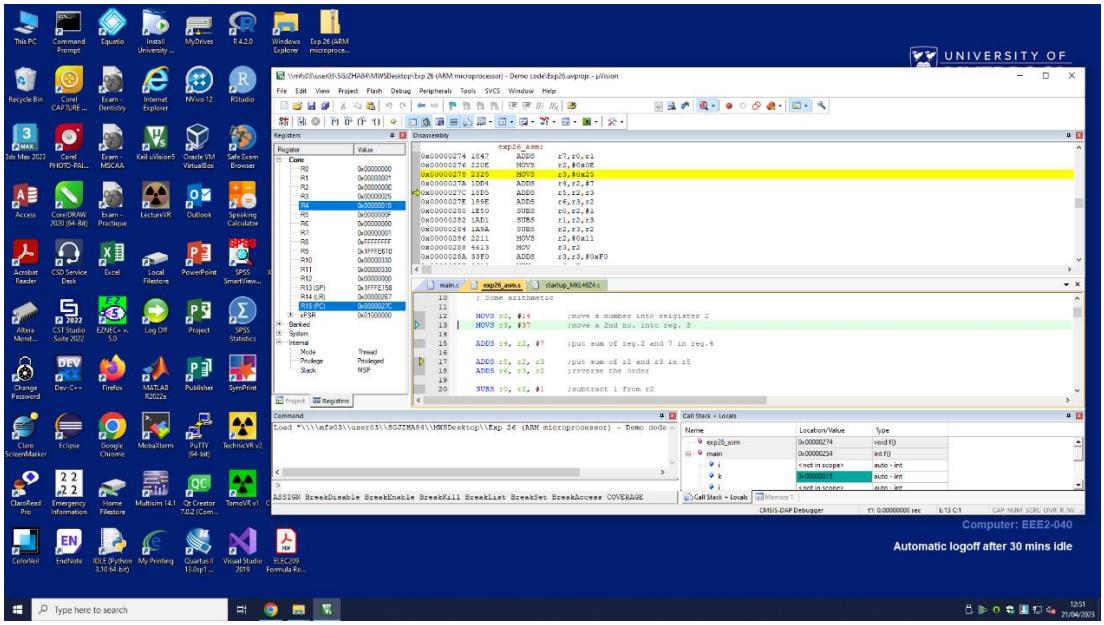
Explanation:

We can see by the first few lines of assembly instructions that the operations from 0x276 to 0x27A are

1: store the hexadecimal number 0x0E into register r2

2: store the hexadecimal number 0x25 into register r3

3: add 7 to the number stored in register r4



4. Answer to Q4 [4 marks]

Answer:

Important

Explanation:

Because we can see that the sequence of registers following the instruction represents different operations, for example

ADDS r2, r3, r4

means that the value stored in r2 is added to the value stored in r4, and then the resulting value is stored in r2. If the order of the registers is changed, the meaning of the instruction is changed. We can see by the first few lines of assembly instructions that the operation from 0x27A to 0x286 is

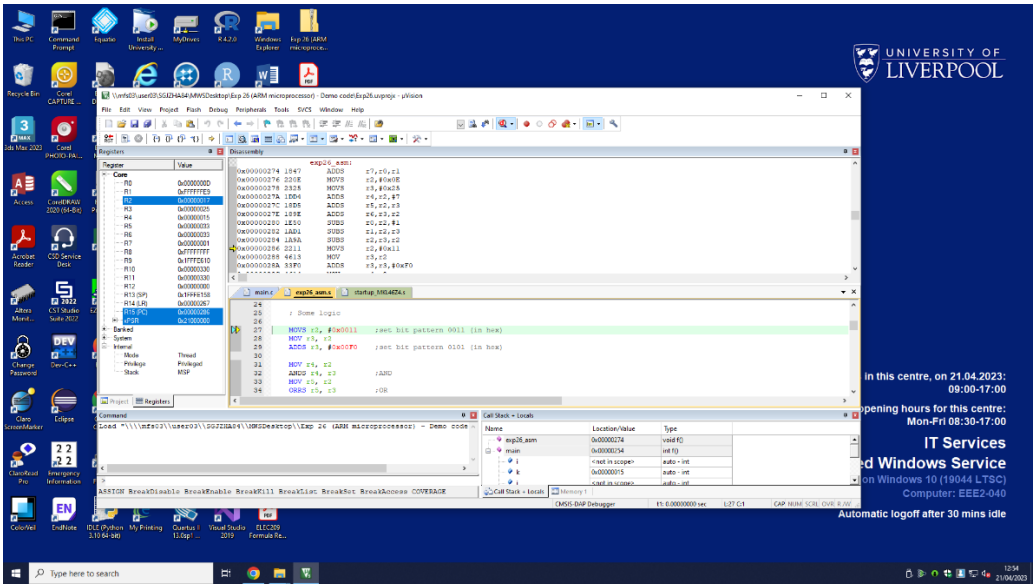
1: add the number stored in register r2 to the number stored in register r3, and then store it in register r5

2: add the number stored in register r3 to the number stored in register r2, and then store it in register r6

3: Subtract 1 from the number stored in register r2 and store it in register r0

4: Subtract the number stored in register r2 from the number stored in register r3, then store it in register r1

5: Subtract the number stored in register r3 from the number stored in register r2, and then store it in register r2



5. Answer to Q5 [4 marks]

Answer:

Yes

Explanation:

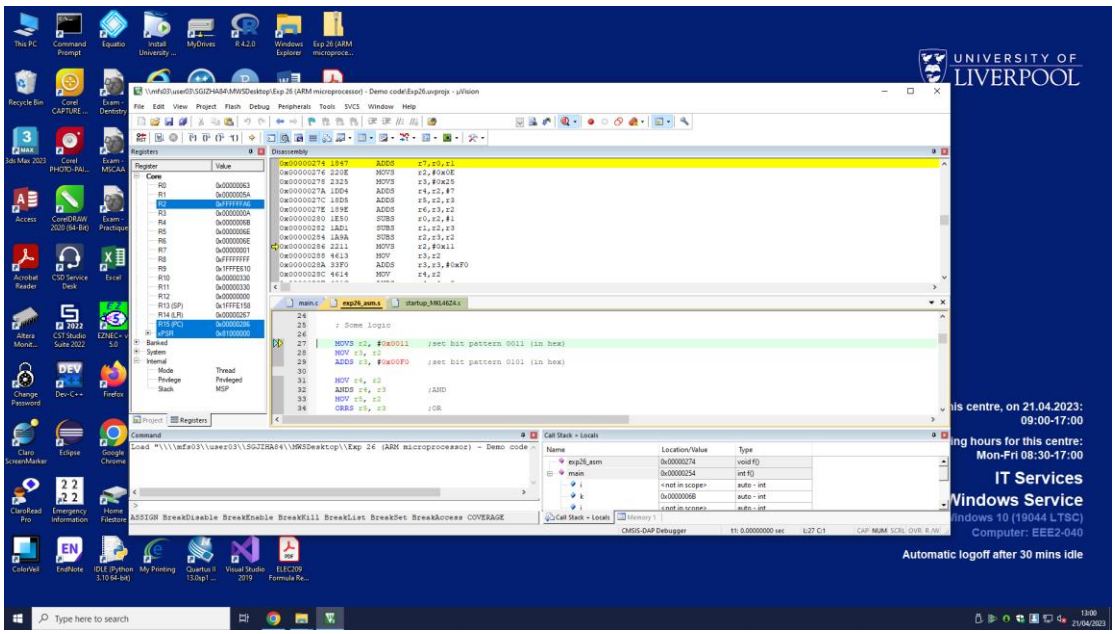
We know that negative numbers are presented in the form of 2's complement. So, the value obtained after a series of addition and subtraction can be found to be correct after calculation and verification. Since the final equation is the value stored in r3 minus the value stored in r2, the resulting value is then stored in r2. Since the question requires that

r2: 0x64

r3: 0x0A

We can easily obtain by using a calculator

0x0A - 0x64 = 0xFFFF FFFF FFFF FFA6



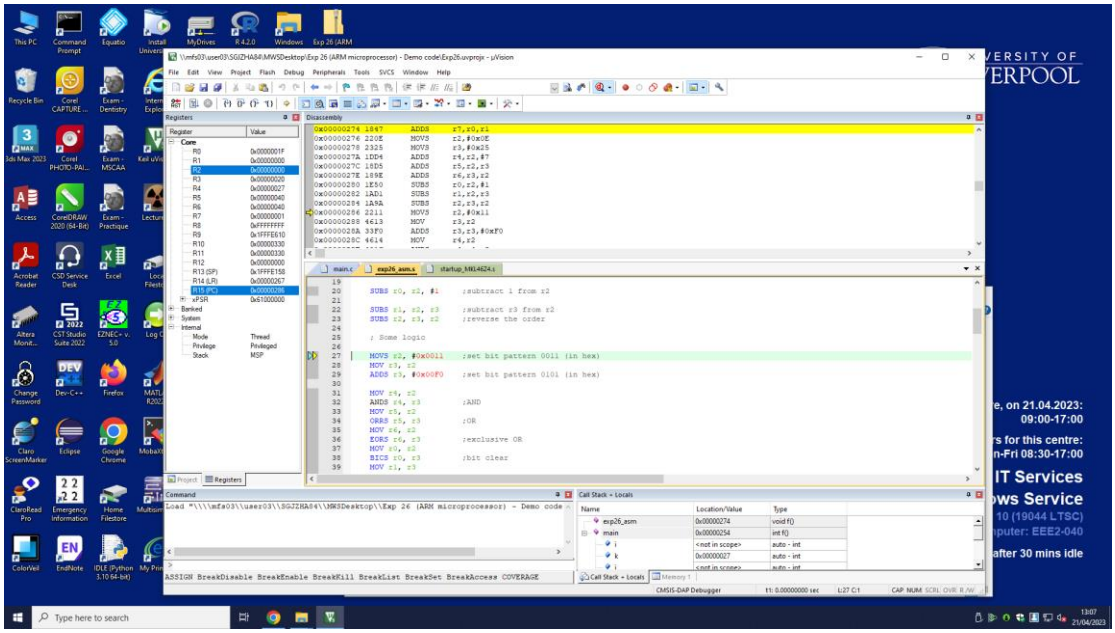
6. Answer to Q6 [4 marks]

Answer:

Yes

Explanation:

Similar to the previous question, when we set both r2 and r3 to 0x20. After calculation we get a value that should be 0, and the result verifies this.



7. Answer to Q7 [4 marks]

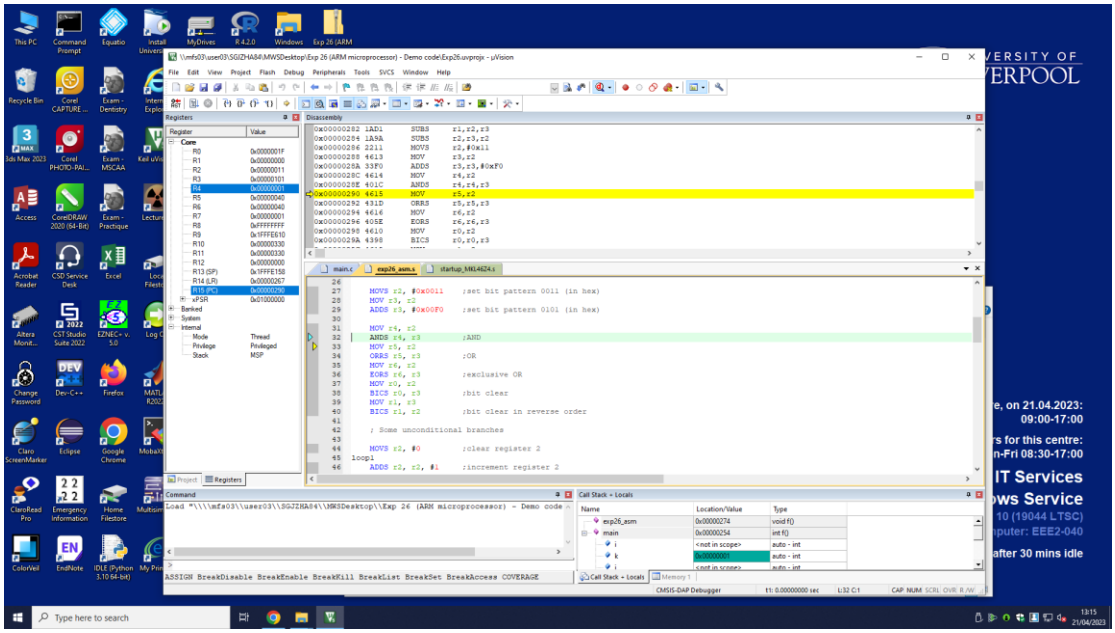
Answer:

AND gate

Explanation:

We can see that r2 is 0x0011, r3 is 0x0101 and r4 is 0x0001.The following truth table shows that the logic function with the AND gate is implemented:

A (r2)	B (r3)	C (r4)
0	0	0
0	1	0
1	0	0
1	1	1



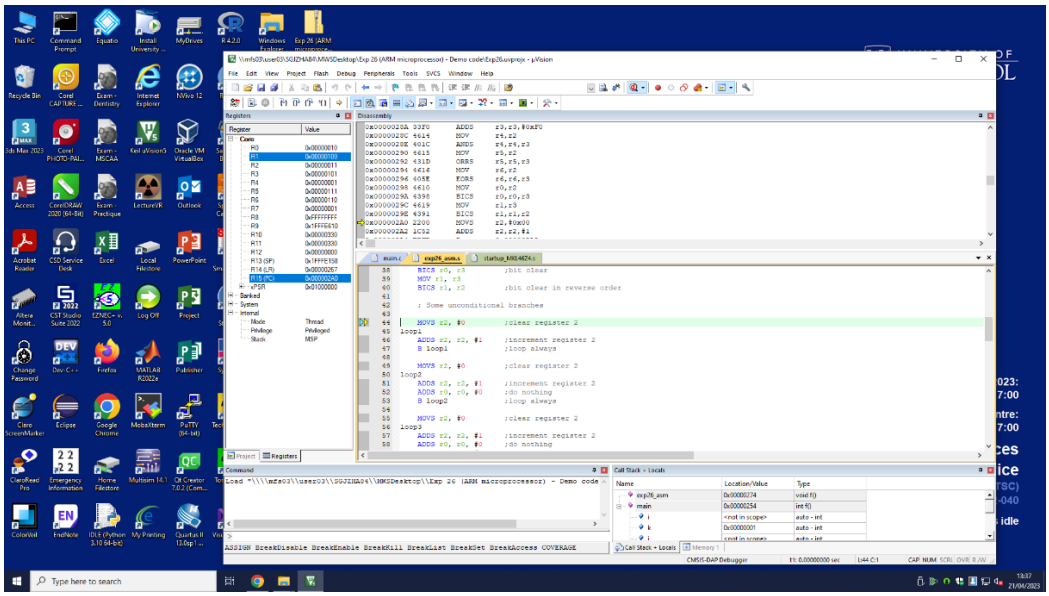
8. Answer to Q8 [4 marks]

Answer:

Important

Explanation:

For several other functions such as ORRS, EORS and ANDS. The latter two registers, although the order does not affect the results obtained, affect the location where the results are stored. For example: **ORRS r2 r2 r3** means that r2 and r3 perform ORR logic operations and store the result in register r2. If the order is changed, the position of the stored result will be changed.



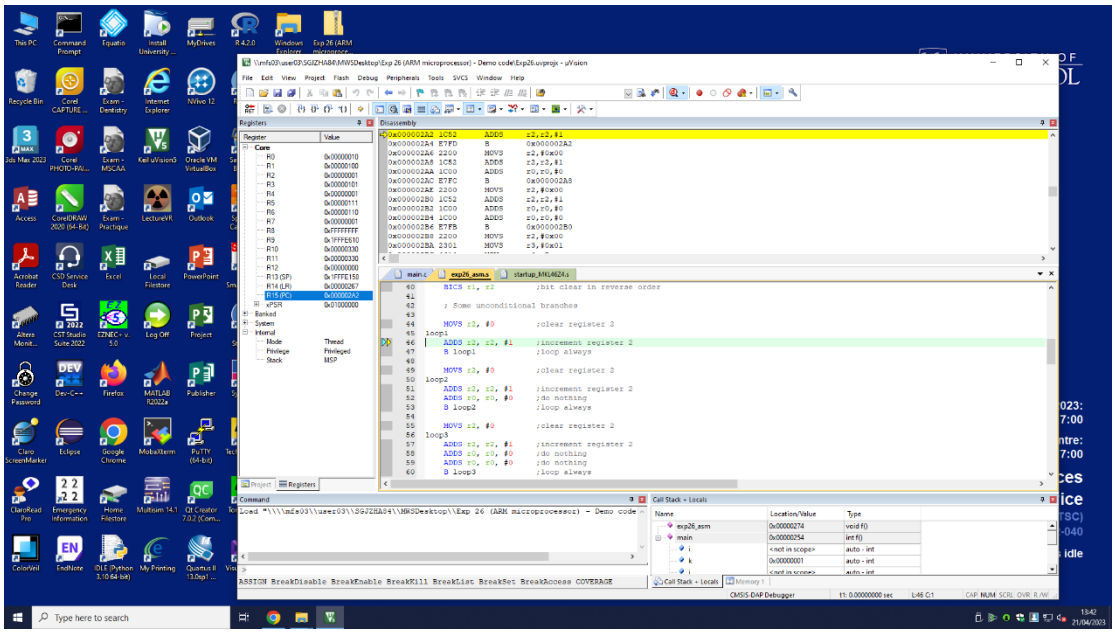
9. Answer to Q9 [4 marks]

Answer:

The program counter will change back to 0x000002A2

Explanation:

The operand of this branch instruction B is the memory address of the previous step, so the meaning of this branch instruction is to go back to the previous step. Therefore the c program counter will change back to 0x000002A2.



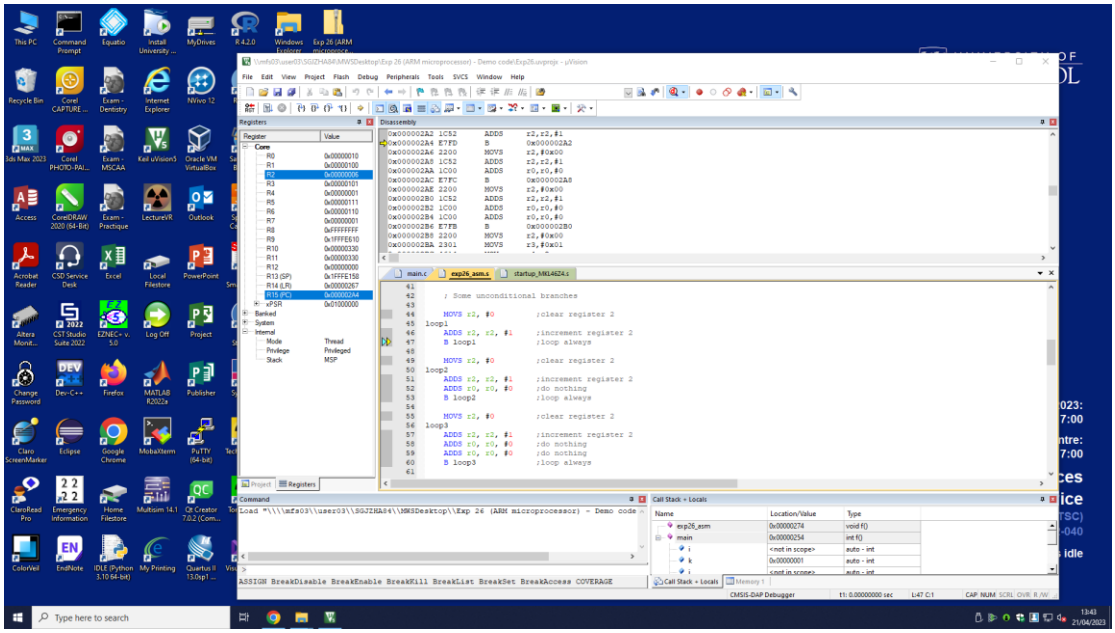
10. Answer to Q10 [4 marks]

Answer:

The content in r2 is 0x00000006

Explanation:

After executing the instruction 0x000002A4 5 times, when the counter points to 0x000002A4 again, it is equivalent to adding 1 to r2 5 times. and the value of r2 is 1 before running, so the content of r2 is 1 + 5 = 6.



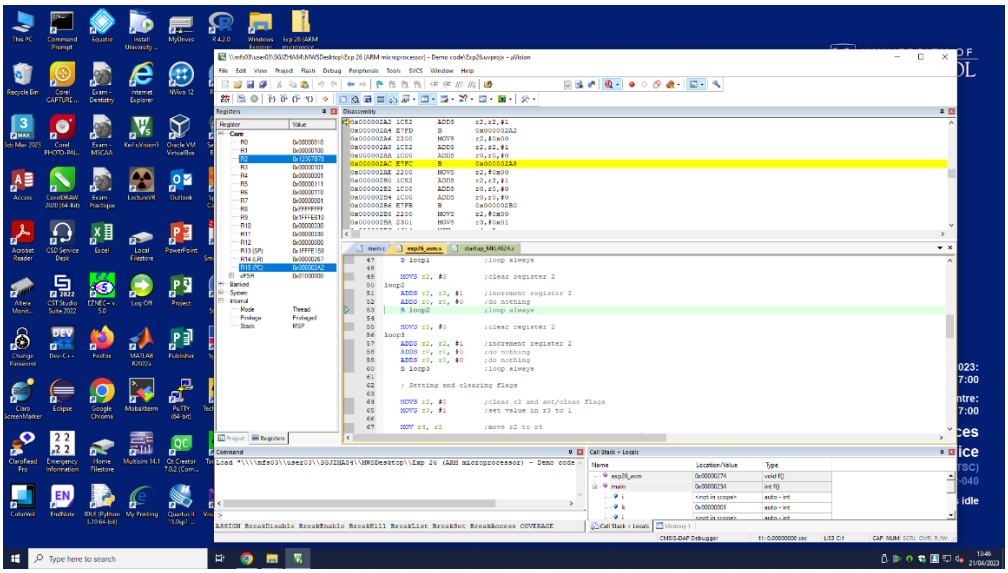
11. Answer to Q11 [4 marks]

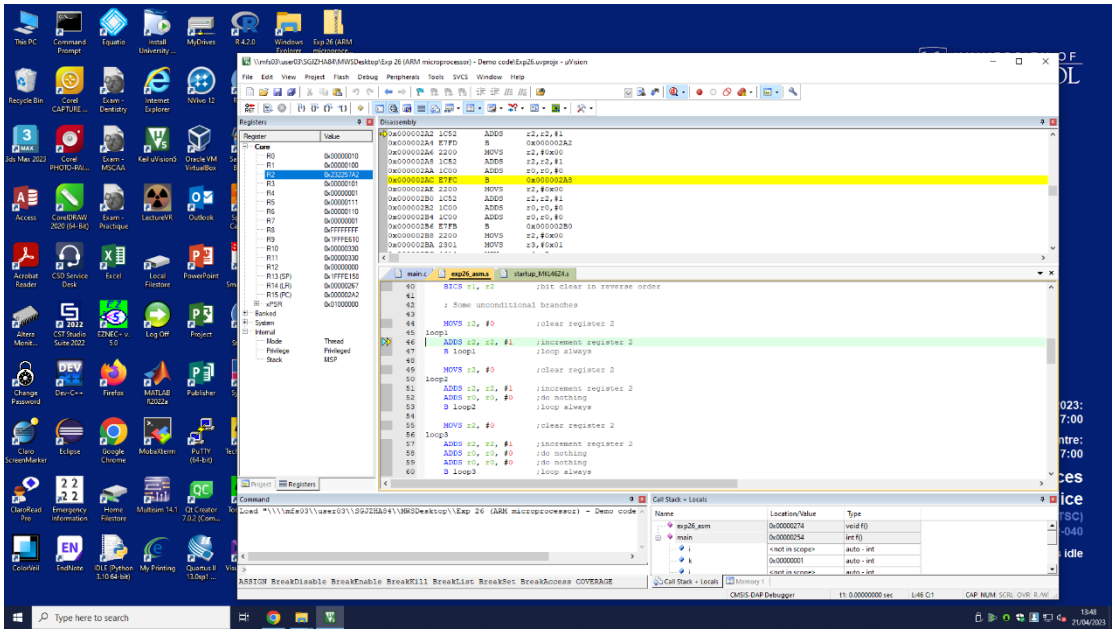
Answer:

This is because the circuit takes the same amount of time to execute the same instruction each time.

Explanation:

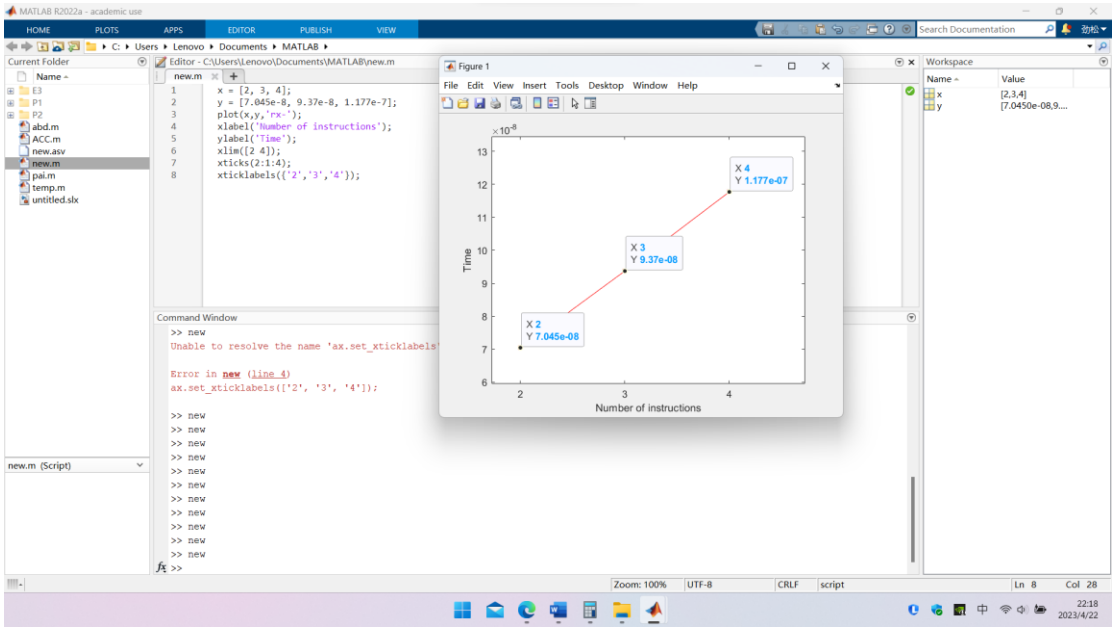
After the first 20 seconds the value of r2 becomes 0x12367B78. after running the second 20 seconds the value of r2 becomes 0x232257A2. after running the second 20 seconds the value of r2 is indeed twice as high as the first. This is because it takes time to execute the two instructions 0x2A2 and 0x2A4, and it takes the same amount of time to run both instructions each time. So, the number obtained by executing +1 all the way through 40 seconds is twice as large as it was in 20 seconds.



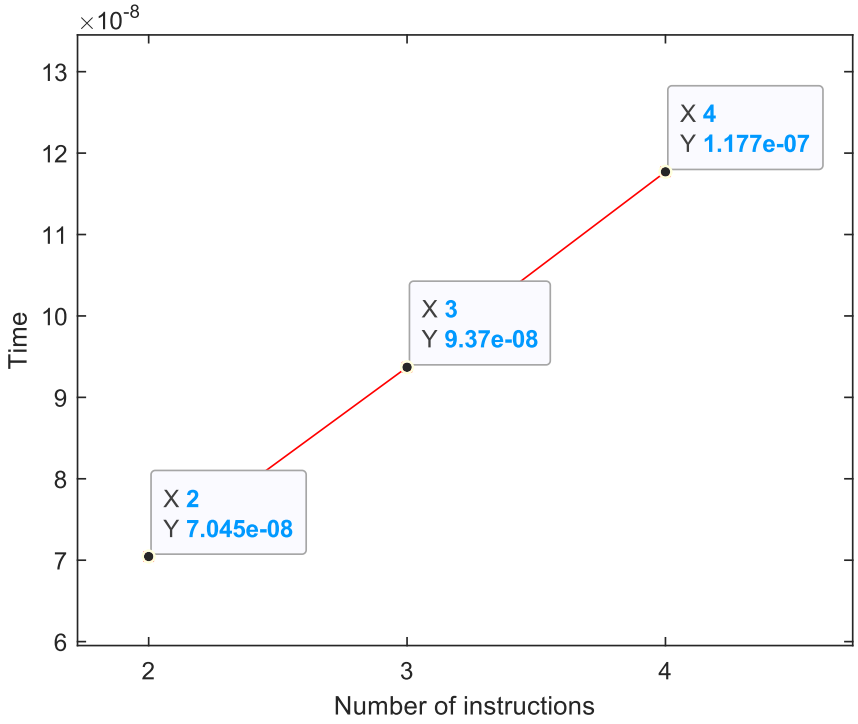


12. The graph of Section 9 (using MS Excel or MATLAB) [5 marks]

Screenshot (PrintScreen):



Screenshot (Snipping tool or equivalent):



20 to 40 seconds for a single instruction:

R1	0x00000100	R1	0x00000100
R2	0x12367B78	R2	0x232257A2
R3	0x00000101	R3	0x00000101

20 to 40 seconds for two instructions:

R1	0x00000100	R1	0x00000100
R2	0x0CEA5879	R2	0x19A32B85
R3	0x00000101	R3	0x00000101

20 to 40 seconds for three instructions:

R1	0x00000100	R1	0x00000100
R2	0x0A168010	R2	0x14366D0C
R3	0x00000101	R3	0x00000101

Comment/Explanation:

The above screenshot contains the value of r2 at 20 to 40 seconds per loop. We can use this to calculate the time required for each loop. We know that the first loop executes two instructions, the second loop is three instructions, and the third loop is four instructions. In each of these loops, there are two instructions that add one to the value of r2 and return the program counter to the previous one. The second loop has one instruction to add 0 to the value of r2, and the third loop has two instructions to

add 0 to the value of r2. Although the instructions added in the second and third loops do not appear to do anything, they do consume time.

13. Answer to Q12 [4 marks]

Answer:

$2.4 \times 10^{-8}\ s$

Explanation:

The difference between the third measurement and the second measurement is one ADDS command, and the difference between them is the time required for one command. The ADDS instruction, like most instructions, is a single-cycle instruction. Therefore, the time consumed by one clock cycle is equal to the time consumed by one ADDS instruction. According to the conclusion drawn from the previous question, one clock cycle is:

$$1.117 \times 10^{-7} - 9.37 \times 10^{-8} = 2.4 \times 10^{-8}\ s$$

14. Answer to Q13 [4 marks]

Answer:

Two clock cycles are required.

Explanation:

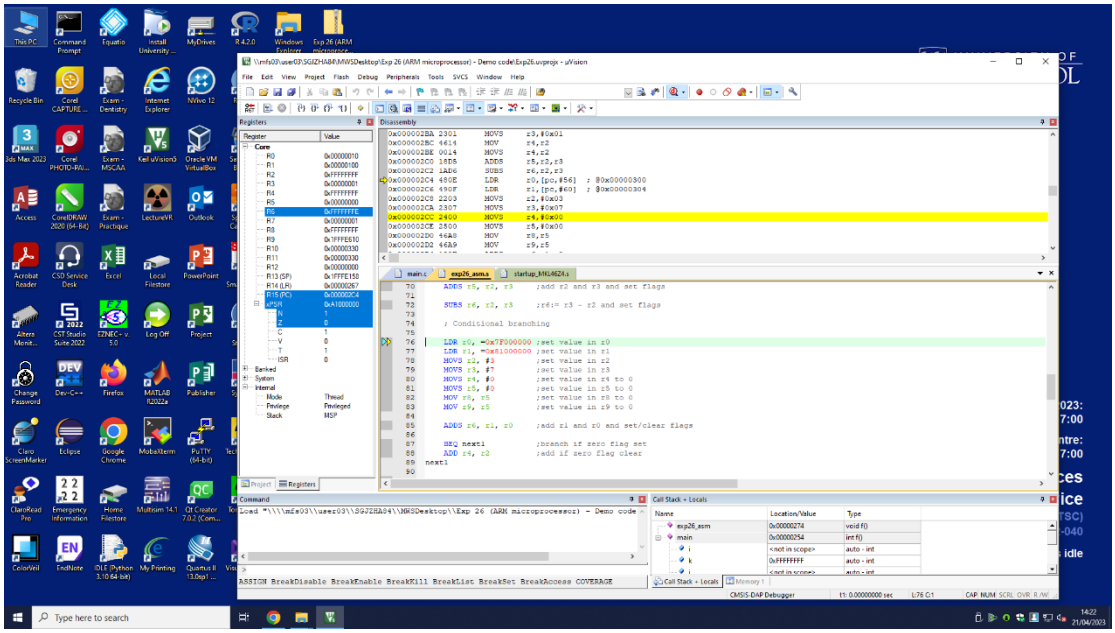
Since the first data is more unstable, we use the second data to calculate it. The total time of the second loop minus the time required for two instructions is the time required for a branch instruction. We can derive this from the following equation:

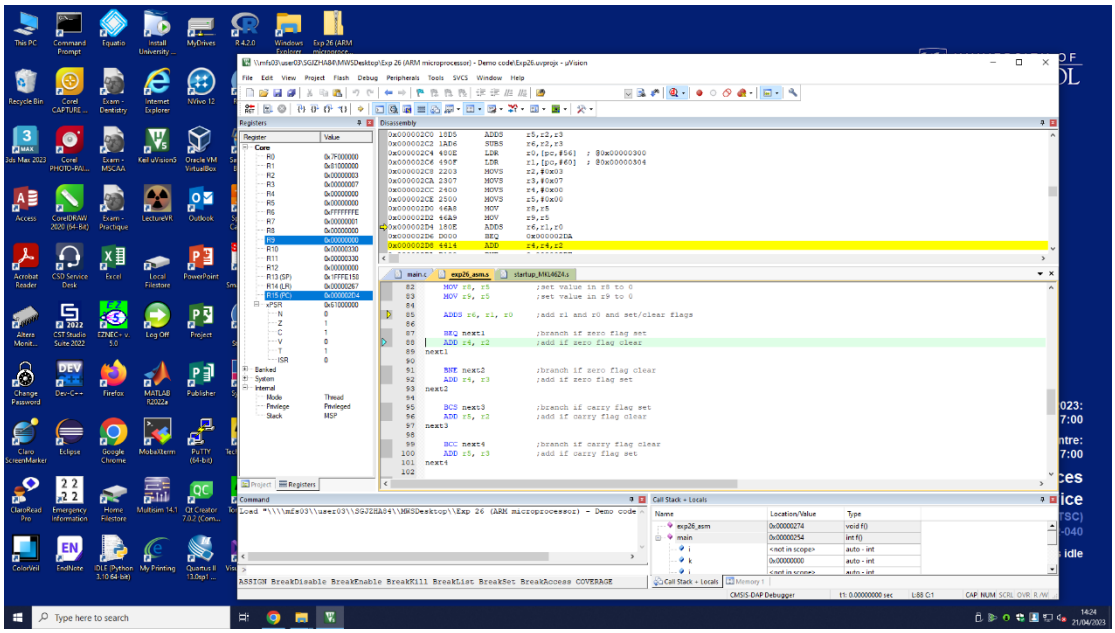
$$9.37 \times 10^{-8} - 4.8 \times 10^{-8} = 4.57 \times 10^{-8}\ s \approx 4.8 \times 10^{-8} s$$

We can see that one clock cycle is $2.4 \times 10^{-8}\ s$ and the time required for a branch instruction is approximately two clock cycles. Some errors may be due to manual timing or instability in the running of the program.

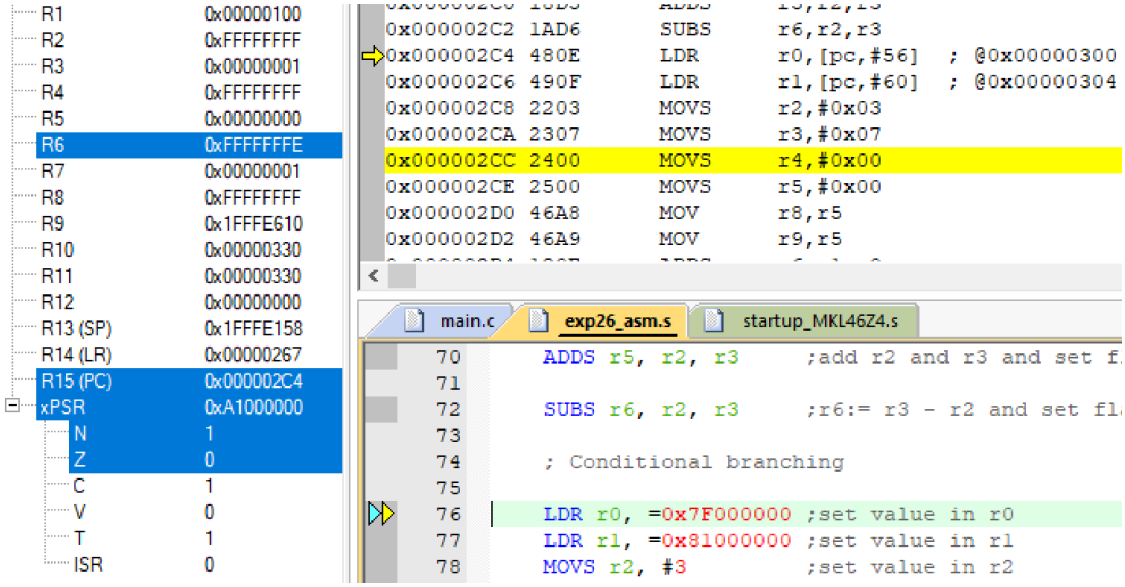
15. Screenshot of the result of Section 10 [2 marks]

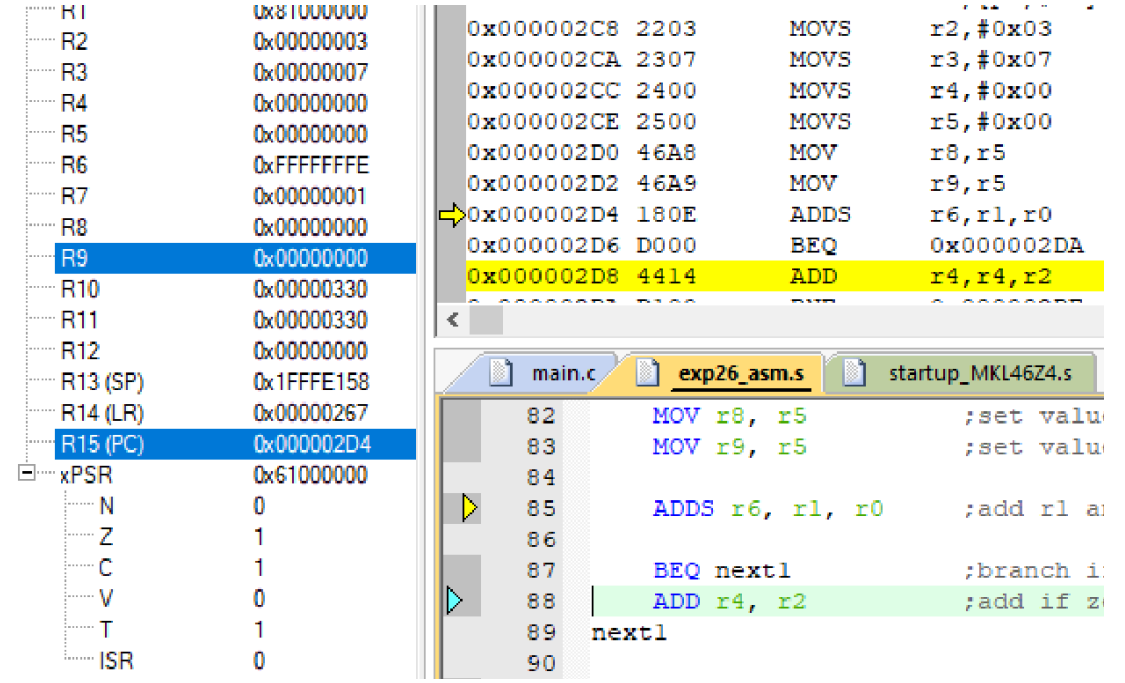
Screenshot (PrintScreen):





Screenshot (Snipping tool or equivalent):





Comment/Explanation:

For the flag part, I did not take a screenshot in the experiment. But we can find out the changing pattern of the flags by the screenshot in the next part. There are four instructions with S between 0x2C4 and 0x2D4, and they are all MOVS. These four instructions perform operations on registers r2, r3, r4, and r5, respectively. The specific contents of the registers change as follows:

R2 0xFFFFFFFF → 0x00000003
R3 0x00000001 → 0x00000007
R4 0xFFFFFFFF → 0x00000000
R5 0x00000000 → 0x00000000

From the above operations we can see that the contents of r2 and r4 are negative. Therefore, the N flag should change from 1 to 0 after the first MOVS instruction is executed. the second MOVS does not involve any operation contained in the flag, so the q flag remains unchanged. After the third MOVS operation is executed the contents of r4 will become 0, which involves the Z flag in the flag. So, the Z flag will also change from 0 to 1. The last MOVS is the same as the third one, so Z will remain 1. In summary, the state transformation of the flag is consistent with the results in the experiment.

16. Answer to Q14 [4 marks]

Answer:

0x000002D6 BEQ 0x000002DA

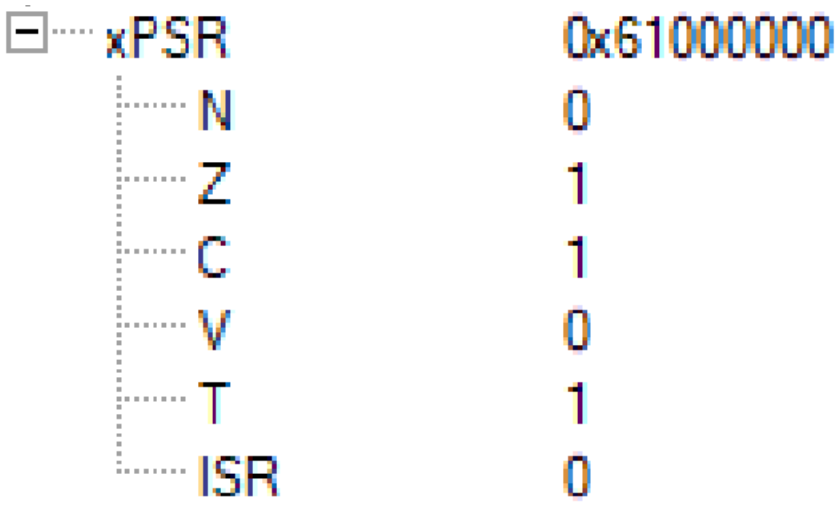
0x000002DE BCS 0x000002E6

0x000002EA BPL 0x000002EE

0x000002F2 BVC 0x000002FC

Explanation:

In the previous problem we already know that the current state of the flag is Z, C, and T, which are the set states, i.e. 1, and the other N and V are unset, i.e. 0. The conditional branch BEQ will execute when the Z flag is 1. We already know that Z is in state 1, so this branch instruction will be executed. The BCS conditional instruction is executed when the carry is set, and we know that the C flag is currently set so this is also executed. The BPL conditional instruction is executed when a negative flag is clear, and we know that the N flag is currently clear so this is also executed. The BVC conditional instruction is executed when the overflow flag is clear, and we know that the V flag is currently clear so this is also executed.



17. Answer to Q15 [4 marks]

Answer:

Yes, the following instructions are executed:

0x000002DA BNE 0x000002E2

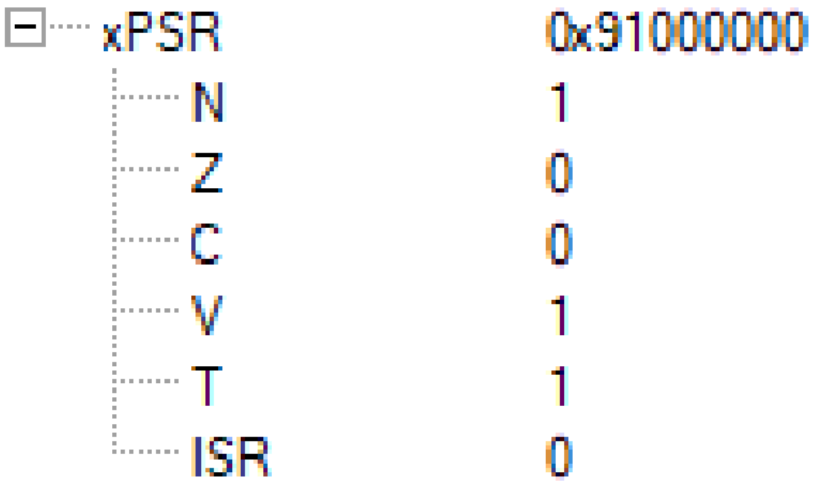
0x000002E2 BCC 0x000002E6

0x000002E6 BMI 0x000002EA

0x000002EE BVS 0x000002F2

Explanation:

First, we know that the current state is the opposite of the previous one: negative: N (1), zero: Z (0), carry: C (0), overflow: V (1). We know that the BNE instruction is executed when the Z flag is cleared. BCC is executed when the C flag is cleared. BMI is executed when the N flag is set. BVS is executed when the V flag is set. Therefore, conditional branching instructions are executed as expected.



18. Answer to Q16 [4 marks]

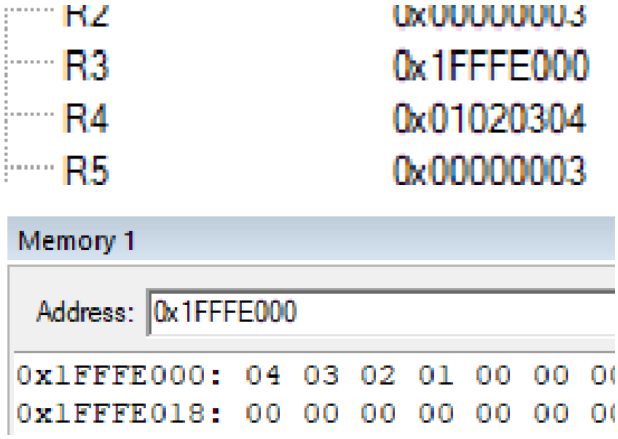
Answer:

Little endian

Explanation:

We can see that the values are stored in reverse order in the memory of the corresponding address, so the memory configuration should be little endian. this can be seen in the memory box below, where the contents of register r4 (0x01020304) are

stored into memory in order 04 03 02 01. therefore, the memory configuration is little endian.



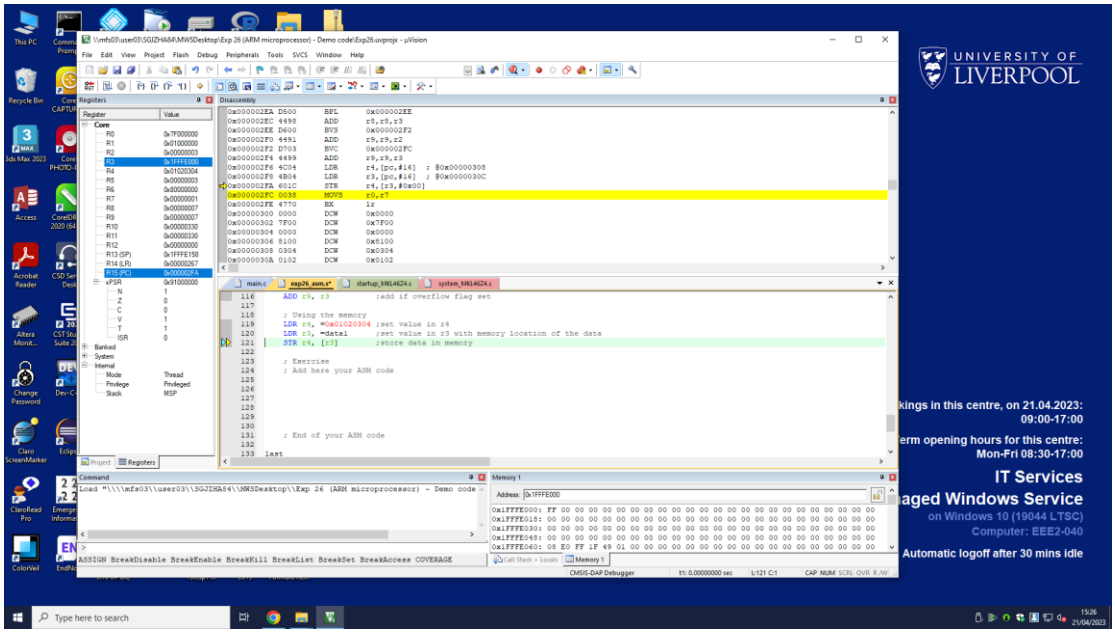
19. Answer to Q17 [4 marks]

Answer:

0x1FFFE000 255

Explanation:

When the single step is run to 0x000002FA, we can see that the content stored in register r3 is the address of data1 in memory. It is stored in r3 as a hexadecimal number, but it is also the address of data1 in memory. When we enter this address in the address box below, we can see that the first byte is FF. Converting it to a decimal number is 255.



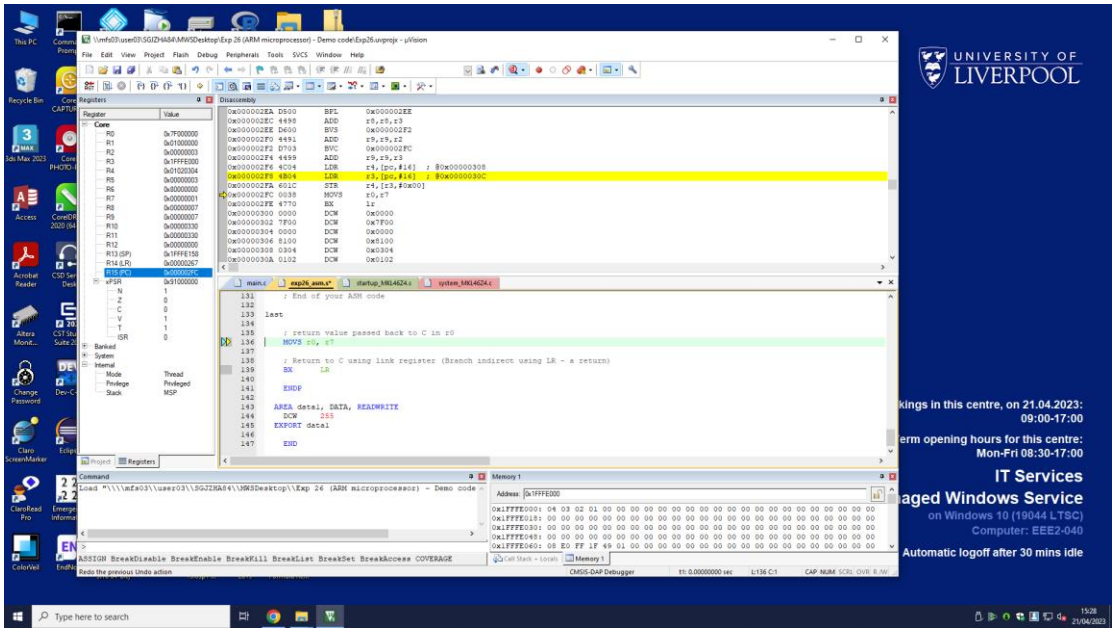
20. Answer to Q18 [4 marks]

Answer:

Yes, it has the value 04 03 02 01

Explanation:

The meaning of this instruction is to store the contents of r4 in the memory with the address of the contents of r3, and the data stored in r4 is 01 02 03 04. However, the data is stored in memory in reverse order so that the microprocessor can easily read the data. Therefore, it is 04 03 02 01 in the memory.



21. ASM code of the exercise in Section 13 (put as text NOT as a screenshot, screenshots of code will receive zero marks) [7 marks]

Code:

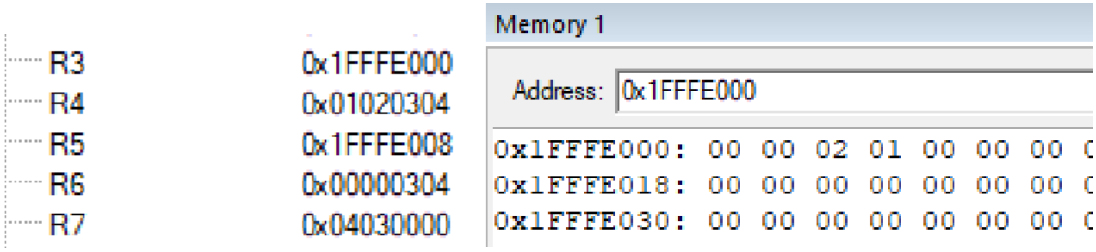
```
LDRH r6, [R3]

REV r7, r6

STRH r7, [r3]
```

Explanation:

The first line of code means to load a half-word from the contents of r3 for the memory address into register r6. This will make it 0x000304 in r6.

The second line of code means to reverse the contents of register r6 and store it in register r7. This will cause the contents of r7 to become 0x04030000.

The third line is to store a half-word from the contents of register r7 to a memory address addressed by the contents of r3. This will cause the contents of the memory address addressed by the contents of r3 to change from 04 03 02 01 to 00 00 02 01. Because storing a half word is storing the second half.




```
; Exercise
; Add here your ASM code

LDRH r6, [r3]

REV r7, r6

STRH r7, [r3]



; End of your ASM code
```

22. Answer to Q19 [4 marks]

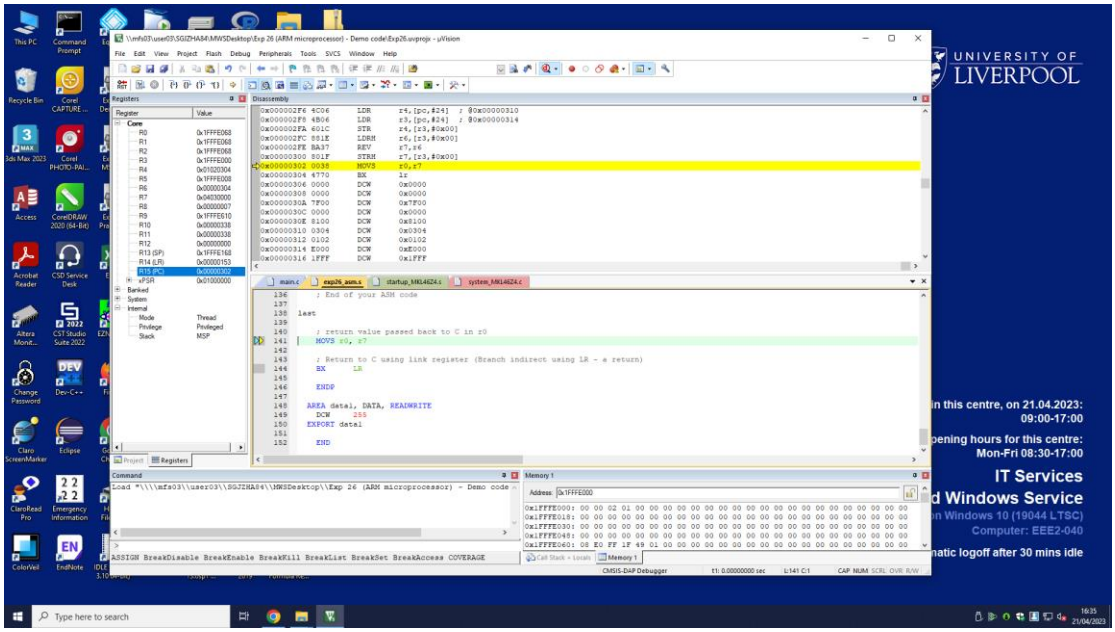
Answer:

Yes! This is exactly the value I would expect to see in memory.

Explanation:

Based on the analysis in the previous 21, we have written the code, explained and analysed the results. We can find it all perfectly in line with our expectations. First

load a half-word from data1 (the address of data1 is stored in r3) into register r6. Then the value in r6 is reversed and stored in register r7. Finally store a half-word from r7 into data1 (with the value in r3 as the memory address 0x1FFFE000).



In addition, we tried storing the whole word into data1, i.e., changing the STRH instruction to a STR instruction. This would cause the contents of data1(0x1FFFE000) to become 00 00 03 04. We found that the results still fit perfectly as expected.