Aaron Hong (ahong02)

Stephen Macris (smacris)

EE469, 5/19/23

**Lab 4 Report**

Procedure:

In this lab, CPUlator was used to examine and implement several different algorithms. In Part 1, two algorithms were examined. In Part 2, an algorithm to count the amount of 1’s in a 32-bit number was implemented. In Part 3, an algorithm to add positive floating point numbers was implemented. One issue we came across was that we got errors when loading registers with very large numbers. Our solution to this was to load large numbers as a combination of two smaller ones.

Results**:**

**Task 1**

*Part 1:*

1. Explain what these lines mean: “.text” “.align=2”

The lines “.text” and “.align=2” prepare the assembler to start assembling instructions correctly into the executable code section of memory and with the proper alignment. “.text” switches the section of code being written to the ‘text’ section. It indicates that the following instructions are meant for the code (or text) segment of memory where executable machine code resides. “.align=2” ensures that each data point is aligned to a 2^2= 4 byte boundary. This makes it easier to access the memory.

2. What is the value of R0, R1, R2, and PC at the start and at the end of the program?

At the start of the program, all these values are 0. At the end, R0 = 4, R1 = 5, R2 = 9, PC = c.

3. Explain the S: B S line of code (lines 10 and 11)

These two lines of code cause the program to run infinitely or until stopped by an external source. The ‘S’ is a label, and ‘B S’ is a branch instruction that makes the program jump to the location labeled ‘S’ when ‘B S’ is executed. Since there are no instructions between ‘S’ and ‘B S’, the program will loop infinitely.

4. Expand the program to solve 4+5+9-3 and save the result in the 40th word in memory. Take a

screenshot of the memory for your lab report.

A screenshot of a computer

Description automatically generated

*Part 2:*

1. What is the value in R0 after the program ends?

The value in R0 after the program ends is 6

2. If the value initially placed in R0 is equal to 5, what is the value in R0 when the program ends?

The value in R0 when the program ends is 120, or 00000078.

3. What does this program do?

This program computes the factorial of the number that is initially in R0. For example, when 3 was in R0, R0 held the value 6 when the program ended (3\*2\*1=6) and then R0 was initially 5, it held the value of 120 when the program ended (5\*4\*3\*2\*1=120).

4. If you replace the instructions at lines 2 and 10 in Figure 2 with PUSH {R0, R1} and POP {R1, R2} respectively, how will the program behave and why?

In the modified version, ‘lr’ is not being preserved, so when the program attempts to return from the subroutine using ‘mov pc, lr’ it does not have the correct return address, which is likely to cause errors in the program. Therefore, the new program will not behave as expected and will likely perform with errors and incorrect results.

5a. Replace the instructions at lines 2 and 10 in Figure 2 with PUSH {R3, LR} and POP {R3, LR} respectively.

Doing this will successfully preserve the link register, however, the value in R0 (which is manipulated throughout the recursive loop) will not be preserved, and the original value of R3 will be lost when R3 is popped off the stack. So, the program will not function as intended and will likely produce incorrect results.

5b. Replace the instructions at lines 2 and 10 in Figure 2 with PUSH {LR} and POP {LR} respectively.

This will save and restore LR, allowing the return from subroutine calls. However, as with scenario a, the value in R0 will not be preserved across subroutine calls and this is critical to the correct functioning of the recursion in this program. This will cause the program to malfunction.

5c. Delete the instruction at line 6.

This will leave the stack pointer unadjusted after returning from the subroutine. Since the PUSH instruction at the start of the loop adds two registers to the stack, this will cause the POP at the end of the loop to read the wrong values into R1 and LR, likely causing a crash when it tries to return to an incorrect address.

**Task 2**

Pseudocode:

initialize count to 0

initialize the 32-bit number

for i from 0 to 31

if the bit at position i in the number is 1

increment count

return count

A screenshot of a computer

Description automatically generated

This screenshot shows that the counting 1’s algorithm is working properly. The input is 0x0f0f0f0f, which has 16 1’s. This matches the algorithm’s output in r0.

**Task 3**

1. Write 2.0 as an IEEE single-precision floating point number.

0b01000000000000000000000000000000

0x40000000

2. Write 3.5 as an IEEE single-precision floating point number.

0b01000000011000000000000000000000

0x40600000

3. Write 0.50390625 as an IEEE single-precision floating point number.

0b00111111000000010000000000000000

0x3f010000

4. Write 65535.6875 as an IEEE single-precision floating point number.

0b01000111011111111111111110110000

0x477fffb0

5. Compute the sum of the numbers from (c) and (d) and express the result in IEEE floating point

format. Truncate the sum if necessary.

0b01000111100000000000000000011000

0x47800018

A screenshot of a computer

Description automatically generated

This screenshot shows that the floating point addition algorithm is functioning properly. It performs the same operation done in Question 5 of Task 3. As shown, 0x3f010000 is in r1 and 0x477fffb0 is in r2. The result is shown in r0, which is 0x47800018.