Lebanese American University

School of Engineering

Electrical and Computer Engineering Department

Course: COE 323 Microprocessors

Project: Floating Point

Wednesday 13th May, 2020

In this project you will create an assembly code that would be able to interpret and perform multiplications on floating point numbers.

The floating point number format uses 32 bits and follows the format: sign (s) - exponent (e) - mantissa (F): $-1^{s}*2^{e-b}*1.F$

Subroutine_1:
This subroutine is provided its input and outputs using PEA, thus it should be called in the following fashion, where Input and Exponent_Bits are inputs and the Sign, Exponent, and Mantissa are outputs.

PEA Input
PEA Exponent_Bits
PEA Sign
PEA Exponent
PEA Mantissa
BSR Subroutine_1

Input is a 32-bit floating point number and Exponent_Bits is the number of bits to be used for the exponent.
This subroutine provides the separate components of a floating point number in a word for the sign (s), a long for the exponent (e-b), and a long for the mantissa (F).
As an example:
Subroutine_Input: Exponent_Bits: 00000013, Input: 40001400
Subruotine_Output: Sign: 0000, Exponent: 00000002, Mantissa: 40000000 (is filled from the most significant bit downwards)

The use of registers inside this subroutine should have no effect on Main.

Subroutine_2:

This subroutine calculates the product of the floating point numbers provided from Main and provide the output as:

PEA Input_1

PEA Input_2

PEA Exponent_Bits

PEA Output

This subroutine has to use Subroutine_1 twice.

The use of registers inside this subroutine should have no effect on Main.

Use LINK and UNLK to store the full product of the mantissas before reporting the result.

In the assigned groups write the code and show the simulation results by filling the Google Forms at the link you received on Webex (Only one person from the group should use the form, make sure you elect a delegate):

1- A screen shot that shows the listing file (in the simulator window, your full code should be visible along with cycle count). For each of the test examples.
2- The memory map at the data area that proves your result. For each of the test examples.
3- Your code file (*.x68)
4- A report that explains your code.

The grade is based on a working simulation using the provided examples, speed of operation, quality of the report, along with following all the instructions exactly.

Your code should use $1000 as the data area, address $400 for Main, $2000 for Subroutine_1 and $3000 for Subroutine_2.

The stack should be restored to its original form after it is used.

The subroutines have to use the input as provided with PEA (not Input_1 as a global variable for example). Your subroutines should be able to work with a different “Main” as long as the inputs are provided in the same manner. (You cannot use the labels found in the area at $1000 within the subroutines)

Example 1:

```
        ORG $1000
Input_1         DC.L    ?
Input_2         DC.L    ?
Exponent_Bits   DC.W    ?
Product         DS.L    1
Sign_1          DS.W    1
Exponent_1      DS.L    1
Mantissa_1      DS.L    1
Sign_2          DS.W    1
```

```
Exponent_2      DS.L    1
Mantissa_2      DS.L    1


        ORG     $400
.
.
PEA Input_1
PEA Exponent_Bits
PEA Sign_1
PEA Exponent_1
PEA Mantissa_1
BSR Subroutine_1
.
.
```

Example 2:
Same as Example 1 with

```
Input_1         DC.L    ?
Input_2         DC.L    ?
Exponent_Bits   DC.W    ?

.
.
```

Example 3:
Same area at $1000 for Example 1

```
        ORG     $400
.
.
PEA Input_1
PEA Input_2
PEA Exponent_Bits
PEA Product
BSR Subroutine_2
.
.
```

Example 4:
Same area at $1000 for Example 2

```
        ORG     $400
.
.
PEA Input_1
```

```
PEA Input_2
PEA Exponent_Bits
PEA Product
BSR Subroutine_2
.
.
```