CPE 325: Embedded Systems Laboratory
Laboratory Assignment #5

Assignment [150 pts]

Q1 [100] Write an assembly program that performs the sum of all elements raised to a certain exponent on all elements of an integer array. You should have two different implementations, by calling two separate subroutines, **SW_Mult** and **HW_Mult**. The array should be initialized in the main program along with a constant to describe its length and the value of the exponent. The calculation to be performed in software and hardware is:

```
for (i = 0; i < input_size; i++)
    temp_sum = input[i]
    for (j = 0; j < exponent-1; j++) {
            temp_sum *= input[i];
    }
    sum += temp_sum
```

Example: If exponent is a cube (exponent = 3), you have to raise each element to the power of 3 and add them up. The first loop iterates through all the elements of an array. The second has to compute the cube of 1 element. The formula for a cube is (x * x * x), so you have two multiplications. That is why the second loop iterates 1 less time than the value represented by the exponent (exponent == 3 => iterations == 2).

Inputs: array = [2, 3, 1], array_size = 3, exponent = 3

Raise each element to the power of 3: $2^3 + 3^3 + 1^3 = 8 + 27 + 1 = 36$

Output: sum = 36

This program should initialize the input array with at least 6 values ranging between -8 and +8 (this can be hard-coded). It should also call two subroutines that perform the calculation (one in software, one in hardware) and pass them the starting address of the array, the array size and the exponent as detailed further below. You don't have to cover cases when the exponent is negative, zero or one. Just when the exponent = 2, 3, 4, 5….

Measure the number of clock cycles used by each subroutine. Present your findings and explain which subroutine is more efficient and why?

a) One of the subroutines that you need to implement is **SW_Mult** that uses the **Shift-and-Add multiplication algorithm** when calculating. This algorithm is described in the provided pdf file.

b) The other subroutine that you need to implement is **HW_Mult** which should use **Hardware Multiplier** to multiply numbers. Use the program stack only to pass parameters and results

Hint: For your SW_Mult implementation, you will end up multiplying the squared input by the original input again. The square of the input may end up larger than 8 bits, due to the

specified range of inputs. You can still use shift-and-add multiplication, but make sure that your multiplier (the value being shifted right) is the original input, not the square of the input. And use similar approach when calculating the quadratic power as well.

Q2 [50] Write an assembly program that interfaces switches, SW1 and SW2, and LEDs, LED1 and LED2, that meets the following requirements:

1) **You must use interrupts to interface the switches.**
2) Initially, Both LEDs should be off.
3) When SW1 is pressed for the first time, LED2 should be turned on. The next time SW 1 is pressed, LED2 is turned off, and so on. Hence, each press changes the state of LED2.
4) When SW2 is pressed, LED1 blinks 4 times at 8 Hz, then toggles LED2.

Questions To Be Addressed

Please make sure that you have addressed following questions in your demonstration:

Q1:

1. How do you pass parameters to a subroutine using stack? Explain how you extract parameters that you pass using this technique.
2. There is a lot going on in subroutines, make sure you really understand and can explain your code.
3. Which is better, HW or SW multiplication and why.

Q2:

1. What happens when Switch 1 is pressed while LED2 is blinking? Is it disrupted? Why?
2. How did you calculate the delay between blinks?

Topics For Theory

1. Subroutines
2. Interrupt vector
3. Hardware Multiplier
4. Passing parameters (3 different ways)

Deliverables

1. Lab report with screenshots of final outputs
2. Commentary on efficiency of each subroutine (in terms of clock cycles taken for operation)

Source files (.asm files) or as instructed.

Notes:

1. Try different inputs (Place picture in the report) before you conclude which method is more efficient. In your explanation, include the inputs as well.
2. Assume that none of the results will exceed 16-bits.