| galois |

Lab: Specification of a multiplier

Exercise 1:
The `State` of a multiplier at any point of definition includes registers `F1`, `F2`, `A`, `X`, `LOW` and the Carry (`C`) and Zero (`Z`) bits. Define the multiplier `State` as a tuple of "`type State`" in Cryptol. Make the tuple look like this: `(F1,F2,A,X,LOW,C,Z)` keeping the `State` variables in that order. Defining `State` as a type is not necessary but is very convenient.

Exercise 2:
Every line of the multiplier results in a change of state. Therefore, the operation of the code may be tracked with functions that input State and output State for each line. The output State becomes the input to the function that is designed for the next instruction to be executed.

As an example, the first line of the code is to load 8 into the `X` register. So, define a function called `step01` which, given input `State`, returns an output `State` where the only part of the `State` that is changed is the `X` register. The function `step01` may be written as follows:

```
step01 : State -> State
step01 (f1,f2,a,_,l,c,z) = step02 (f1,f2,a,8,l,c,z)
```

So execution of `step01` results in a change of `State` that is passed to `step02`. Next `step02` will pass its new State to `step03` like this:

```
step02 : State -> State
step02 (f1,f2,_,x,l,c,z) = step03 (f1,f2,0,x,l,c,z)
```

The input to `step01` has `_` for register `X` because its value is going to be 8 irrespective of all other inputs, after `step01` is executed. The `_` is a "don't care" value. When `step02` is executed its `F1`,`F2`,`A`,`LOW`,`C`, and `Z` values will be the same as for `step01`, hence its input values will not change from `step01` input values and the `step01` arguments for those variables will be the same arguments for `step02`. But 8 will be the input for register `X` in `step02`. Then `step02` calls `step03` where the only change is to the `A` register as per the 2nd instruction which is "Load 0 into register `A`".

The 3rd line says "clear the carry bit". Your exercise is to create `step03` similar to the two steps above. In this case `step04` will follow `step03`.

Observe that if the `State` tuple had not been defined as a type, the type signature for the `stepxx` functions would be this:

```
step01 : ([8],[8],[8],[8],[8],Bit,Bit) -> ([8],[8],[8],[8],[8],Bit,Bit)
```

Exercise 3:
Line step04 changes F1 via rotation through the Carry bit. So, F1 should be represented by its bit structure and C, so rotation can be performed on the representation and handed back

to the new F1 as an argument of step05, the next line to be executed after step04. F1 is represented as a sequence of 8 bits. The Cryptol language allows a construction like this:

```
[b7,b6,b5.b4,b3,b2,b1,b0] = F1
```

which fills b0 to b7 with F1's bit values. Then right rotation through the C bit looks like this:

```
F1' = [C,b7,b6,b5,b4.b3,b2,b1]
```

where the new value for F1 is F1'. Write step04 and don't forget about what happens to the C bit.

Exercise 4:
There is a conditional jump at step05. If the C bit is False then advance to step08 without changing State. Otherwise proceed to step06 without changing State. Write step05.

Exercise 5:
At step06 the C bit is cleared. This is a repeat of Exercise 2. Write step06.

Exercise 6:
step07 accomplishes the addition of the contents of A and the contents of F1. The result of step07 is a change in the A register, which will hold the result of the addition, and the C and Z bits which are given values depending on the result of the addition. Now 8 bit quantities are being added but the addition may result in a 9 bit quantity due to the carry. The carry will be detected in that 9th bit and given to the C input of step08, the next step to be executed. Due to the strong typing of Cryptol the best way to handle the 9 bit addition is to place a 0 in front of the number represented by A and the number represented by F2 and 1 if C is True. The addition could look like this:

```
a'Large : [9]
a'Large = (zero # a) + (zero # f2) + (if c then (1:[9]) else (0:[9]))
```

Now the new C bit value can be obtained from the following test:

```
c' = a'Large > (255:[9])
```

Observe typing the number 255 to 9 bits is needed to avoid a type mismatch with a'Large. The new Z bit can be obtained from a comparison:

```
z' = a' == 0
```

As for the new value of register A, a simple

```
a' = a + f2 + (if c then (1:[8]) else (0:[8]))
```

will do as any resulting carry will simply disappear due to the 8 bit addition. Write step07.

Exercise 7:

step08 and step09 are right rotations that essentially repeat Exercise 3. Write step08 and step09.

Exercise 8:

Register X is decremented in step10. This results in two changes: the X register and the Z bit. To get the value of the X register use x'= x-1. To get the value of the Z bit use x'==0. Write step10.

Exercise 9:

`step11` is another branch like Exercise 4. This time the condition is `z==0`. If this is `True` then advance to `step04` with no change in `State`. But otherwise, there is no step to advance to. Instead the current `State` becomes the output. Write `step11`.

Exercise 10:

There needs to be a way to start the multiplier. The `A`, `X`, and `LOW` registers and `C` and `Z` bits need to be initialized. Input parameters `F1` and `F2` need to be given. Then `step01` is executed and the output should be a tuple containing the values of the `A` and `LOW` registers. Call this function `legato`. Make the parameter list anything you want and write `legato`.