Project I

Illya Starikov

Due Date: October 4th, 2016

When coding, I started by breaking up the assignments into subproblems; problems small enough that could be handled in a subroutine of their own.

To begin, the first specified task was to read an input from `P1`; which was nothing more than a couple of `MOV`s and a loop (`JZ`, to be precise). Then I made sure to clear the upper four-bits (because they were to be set as output ports) to prevent an extra 240[1] iterations from occurring. Then one of two things could occur based on the input: read from `ROM` or calculate the Fibonacci sequence on the fly.

Because `ROM` has to be read only, and we can write only at compile time, Fibonacci (up to the first 15 digits) had to be defined before the start of the program via a `DB` at `200H`. However, calculation on the fly was a bit more complicated. We know Fibonacci to be

$$0,\ 1,\ 1,\ 2,\ 3,\ 5, \ldots, a_{n+1} = a_n + a_{n-1} = \sum_{k=n-2}^{n-1} a_k \tag{1}$$

with the summation's initial conditions being $a_0 = 0$ and $a_1 = 1$. Using this more compact summation, we can write this programmatically. Store the initial conditions in two registers, add together to get a result, rotate the result into the register (and removing a_{n-1}) can produce an effective Fibonacci sequence generator.

Next is to output the value; this depends on the original input (because that determine how Fibonacci would be generated). There are, again, two methods based on the input.

One would be to read from the value, a simple enough tasks. `MOV` the `DPTR` to the location of the value, and read it in. However, the fifteenth value

[1] $11110000_2 = 240_{10}$

is odd — we had stored 121_{10} in ROM, which is 79_{16} in base-16 encoding. We account for this by appending the `CY` flag. Now, Fibonacci(15) = `CY` + $79_{16} = 179_{16} = 377_{10}$. But this doesn't affect lower numbers; Fibonacci(14) = `CY` + $90_{16} = 090_{16} = 144_{10}$.

Outputting the result of the generated Fibonacci number is even easier; our value generated for the Fibonacci is still in the accumulator. After making `P3` an output port, `MOV` from the accumulator to `P3`. The carry flag was already set during the arithmetic if the number was 15[2].

After the output, there is to be a delay. This delay has to be $27,250\mu s$, which is a relatively large number since the loops can only have repeat 255 times. So to do this, I knew I needed to undershoot by a small margin (effectively $< 553.35\mu s$). So arbitrarily choosing 255 as one of the loops, a reasonable assumption would be there has to an inner loop of 47 iteration to give $26,007.45\mu s$ for the main loop. The rest of the subroutines was just asymptotically approaching $27,250\mu s$. This is summarized in Table 1, where the final value of the delay $27,250.86\mu s$.

Code	MC	Iter	Time
`MOV R3, #255`	2	1	$2.17\mu s$
`DELAY1: MOV R4, #47`	2	255	$553.35\mu s$
`DELAY2: DJNZ R4, DELAY2`	2	255×47	$26007.45\mu s$
`DJNZ R3, DELAY1`	2	255	$553.35\mu s$
`MOV R3, #109`	2	1	$2.17\mu s$
`DELAY3: DJNZ R3, DELAY3`	2	61	$132.37\mu s$

Table 1: Calculation of the delay

[2]Fibonacci(15) $> 255_{10} \implies$ carry $\implies$ `CY` $= 1$