Problem Sheet 10

Henri Sota
h.sota@jacobs-university.de
Computer Science 2022

November 22, 2019

Problem 10.1

a) Machine Code Table Decoder

#	Machine Code	Assembly Code	Description
0	001 1 0001	LOAD #1	Load the value 1 into the accumulator
1	010 0 1111	STORE 15	Store the value of the accumulator in memory location 15
2	001 1 0000	LOAD #0	Load the value 0 into the accumulator
3	101 1 0100	EQUAL #4	Skip next instruction if accumulator equal to the value 4
4	110 1 0110	JUMP #6	Jump to instruction 6 (set program counter to 6)
5	111 1 0000	HALT	Stop execution
6	001 0 0011	LOAD 3	Load the value of memory location 3 into the accumulator
7	100 1 0001	SUB #1	Subtract the value 1 from the accumulator
8	010 0 0011	STORE 3	Store the value of the accumulator in memory location 3
9	001 0 1111	LOAD 15	Load the value of memory location 15 into the accumulator
10	011 0 1111	ADD 15	Add the value of memory location 15 to the accumulator
11	010 0 1111	STORE 15	Store the value of the accumulator in memory location 15
12	110 1 0010	JUMP #2	Jump to instruction 2 (set program counter to 2)
13	000 0 0000		no instruction / data, initialized to 0
14	000 0 0000		no instruction / data, initialized to 0
15	000 0 0000		no instruction / data, initialized to 0

b)
- Program starts by loading constant value 1 to accumulator and stores that into memory address 15
- It loads the constant value 0 into the accumulator
- It checks if the value of the accumulator is equal to constant value 4
- If it is equal to 4, it skips next instruction and reads the HALT command which stops the execution
- If it isn't equal to 4, it loads the following instruction which tells the program to jump to memory address 6
- It loads the value of the memory address 3 into the accumulator, subtracts constant value 1 from it and it stores it back again into memory address 3 (which now has the instruction "EQUAL #3"
- It loads the value of the memory address 15 to the accumulator, it adds the value of memory address 15 to the accumulator and it stores it back again into memory address 15

- Last instruction tells the program to jump back to instruction in memory address 2
- The same loop goes on until memory address 3 holds the value of instruction "EQUAL #0" which skips over the "JUMP #6" instruction found in memory address 4 and reads the HALT instruction in memory address 5 which stops the execution of the program
- During each loop while the memory address 3 changes from "EQUAL #4" to "EQUAL #3", ... , "EQUAL #0", the value in memory address 15 doubles, going from 1 to 2, then 4, then 8 and finally 16 (which can not be stored as "10000" in memory address 15 as we have only assigned 4 bits to hold a constant value, but it will be stored as 0, "0000" in binary, because of overflow). In the end memory address 15 will be "00010000".

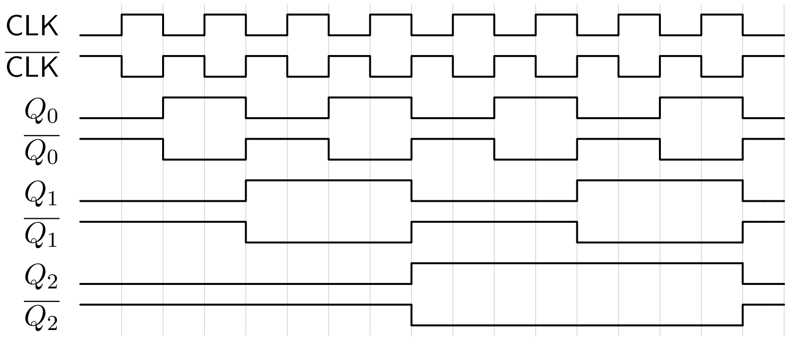
c) The program is calculating 2 to the power of n depending on the amount of times the loop will run. This can be expressed in general mathematical terms as a geometrical sequence from which we can choose the $n-th$ element which matches the output of the program where the inner loop iterates n times:

$$g_n = 2^n \qquad \text{for } n \in \mathbb{N}$$

In our problem, we had to calculate $g_4 = 2^4 = 16$.

Problem 10.2

a) Timing diagram for a 3-bit ripple counter consisting of three positive edge triggered D flip flops and negation gate on the clock input C



The first D flip flop is triggered on a transition from high to low of clock CLK, a low to high transition for negation of CLK which is the input to the D flip flop. The other D flip flops are also positive edge triggered and depend on the transition from low to high of the negation of their master's Q output. The first time the D flip flop is triggered, its data input is 0, therefore Q is 0 and $\overline{\text{Q}}$ is 1. Since we ignore the impact of gate delays, D now becomes 1 from $\overline{\text{Q}}$. In this moment since the D flip flop has been triggered, the data goes through to the output Q as 1. As the clock keeps toggling $\overline{\text{Q}}$ will change too and this will act as the "clock" for the next D flip flop, but with half frequency and double period of the previous clock cycle. In the end we can see that the frequency has become $\frac{1}{8}$th of the input clock frequency.

b) Having an asynchronous binary counter created from chaining D flips flops, theoretically means that we can chain infinitely many of them in series, thus being able to represent a number up to 2^{n+1} in binary format, where n is the number of D flip flops. Practically there might be a problem with the propagation of the whole signal through the counter as it is not synchronous.

Problem 10.3

a)
```
-- a) Code Area
-- Function triangleArea takes 3 points and returns the area (a triangle)
    bounded by the lines connecting them
-- It calculates the area using the determinant method
triangleArea :: Point -> Point -> Point -> Double
triangleArea p1 p2 p3 = ((x p1) * ((y p2) - (y p3)) + (x p2) * ((y p3) - (y p1
    )) + (x p3) * ((y p1) - (y p2))) / 2

class Area a where
    area :: a -> Double

instance Area Rectangle where
    area (Rectangle p1 p2) = ((x p2) - (x p1)) * ((y p2) - (y p1))

instance Area Circle where
    area (Circle m r) = pi * r * r

instance Area Triangle where
    area (Triangle a b c) = triangleArea a b c
```

Note: For calculating the area of the triangle I've also used Heron's formula in a function that has been commented out in the .txt file. The formula used to calculate the area of the triangle is:

$$A = \frac{P_{1.x}(P_{2.y} - P_{3.y}) + P_{2.x}(P_{3.y} - P_{1.y}) + P_{3.x}(P_{1.y} - P_{2.y})}{2}$$

where $P_{n.c}$ is c component (x or y) of point n

b)
```
-- b) Code BoundingBox
-- Functions minx, maxx, miny, maxy take 3 points as input and return a double
-- Respectively they find minimum x, maximum x, minimum y, maximum y between
    the 3 points
minx :: Point -> Point -> Point -> Double
minx a b c = (x a) `min` (x b) `min` (x c)

maxx :: Point -> Point -> Point -> Double
maxx a b c = (x a) `max` (x b) `max` (x c)

miny :: Point -> Point -> Point -> Double
miny a b c = (y a) `min` (y b) `min` (y c)

maxy :: Point -> Point -> Point -> Double
maxy a b c = (y a) `max` (y b) `max` (y c)

class (Area a) => BoundingBox a where
    bbox :: a -> Rectangle

instance BoundingBox Rectangle where
    bbox (Rectangle p1 p2) = Rectangle { p1 = p1 , p2 = p2 }

instance BoundingBox Circle where
    bbox (Circle m r) = Rectangle p1 p2
        where
            p1 = Point ((x m) - r) ((y m) - r)
            p2 = Point ((x m) + r) ((y m) + r)

instance BoundingBox Triangle where
    bbox (Triangle a b c) = Rectangle p1 p2
        where
```

```
            p1 = Point (minx a b c) (miny a b c)
            p2 = Point (maxx a b c) (maxy a b c)
```

Test cases found in the slides have been added as print function calls inside the main of .txt file and they give the predicted output.