Introduction to Computer Science
Jacobs University Bremen
Dr. Jürgen Schönwälder

Course: CH-232-A
Date: 2019-11-15
Due: 2019-11-22

ICS 2019 Problem Sheet #10

Problem 10.1: *simple cpu machine code* (2+2+1 = 5 points)

The following program has been written for the simple central processing unit introduced in class. The table below shows the initial content of the 16 memory cells. The first column denotes the memory address.

#	Machine Code	Assembly Code	Description
0	001 1 0001		
1	010 0 1111		
2	001 1 0000		
3	101 1 0100		
4	110 1 0110		
5	111 1 0000		
6	001 0 0011		
7	100 1 0001		
8	010 0 0011		
9	001 0 1111		
10	011 0 1111		
11	010 0 1111		
12	110 1 0010		
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

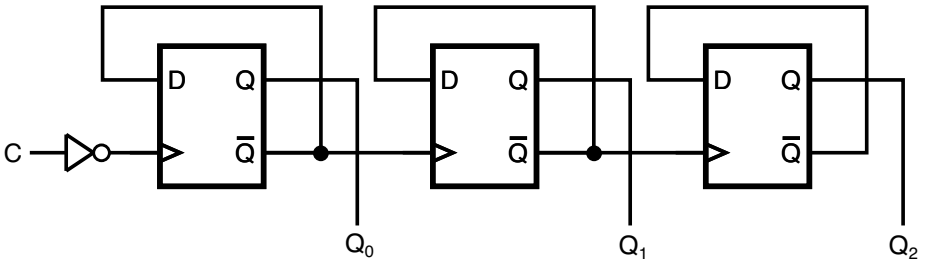
a) Explain what the instructions are doing by filling in the assembly code column. Add meaningful text to the description column to describe the action performed by an instruction.

b) Explain how the program proceeds with its calculation. Describe the execution of the program. Which cells change and what is the purpose of the changes? What is the result left in memory cell 15 when the program stops execution?

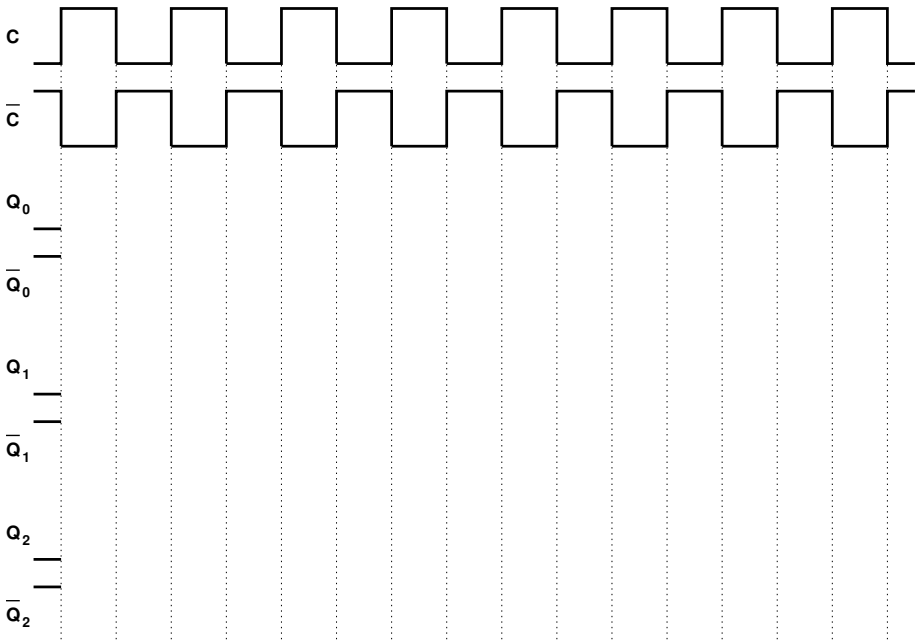
c) Can you describe in general terms what the program is doing using a mathematical expression?

Problem 10.2: *ripple counter using d flip flops* (2+1 = 3 points)

The following circuit shows a 3-bit ripple counter consisting of three positive edge triggered D flip flops and a negation gate on the clock input C.



a) Complete the following timing diagram. Assume that gate delays are very short compared to the speed of the clock signal (i.e., you can ignore the impact of gate delays).



b) Can you make ripple counters arbitrary “long” or is there a limit on the number of D flip flops that can be chained? Explain.

Problem 10.3: *type classes (haskell)* (1+1 = 2 points)

The following Haskell module defines types for the two-dimensional shapes `Rectangle`, `Circle`, and `Triangle`.

```haskell
{- |
   Module: Shape.hs
-}

module Shape where

data Point = Point { x :: Double, y :: Double } deriving (Show)

-- Rectangles

data Rectangle = Rectangle { p1 :: Point, p2 :: Point } deriving (Show)

-- Circles

data Circle = Circle { m :: Point, r :: Double } deriving (Show)

-- Triangles

data Triangle = Triangle { a :: Point, b :: Point, c :: Point } deriving (Show)
```

a) Define a type class `Area` declaring a function `area`, which returns the area covered by a shape type as a `Double`. The types `Rectangle`, `Circle`, and `Triangle` shall become instances of the `Area` type class.

b) Define a type class `BoundingBox` extending the `Area` type class and declaring a function `bbox`, which returns a `Rectangle` representing the bounding box of a shape. The types `Rectangle`, `Circle`, and `Triangle` shall become instances of the `BoundingBox` type class.

Your implementation should at least pass the following test cases:

```haskell
{- |
   Module: shapetest.hs
```

```haskell
-}

module Main where

import Shape
import Test.HUnit

pa = Point { x =  0, y =  0 }
pb = Point { x = 10, y = 10 }
pc = Point { x =  0, y = 20 }

box      = Rectangle { p1 = pa, p2 = pb }
circle   = Circle    { m  = pa, r  = 10 }
triangle = Triangle  { a  = pa, b  = pb, c = pc }

tests = TestList [ TestCase (100.0 @=? (Shape.area box))
                 , TestCase (314   @=? (floor (Shape.area circle)))
                 , TestCase (100.0 @=? (Shape.area triangle))
                 , TestCase (100.0 @=? (Shape.area $ Shape.bbox box))
                 , TestCase (400.0 @=? (Shape.area $ Shape.bbox circle))
                 , TestCase (200.0 @=? (Shape.area $ Shape.bbox triangle)) ]

main = runTestTT tests
```