Arithmetic using qubits

Binary addition

Suppose we've to add two numbers in binary form! Say... 5 and 7.

In binary form, 5 can be represented by 101 and 7 can be represented by 111. Remember when adding two numbers in binary form,

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 0 = 1$
- $1 + 1 = 0$ with a carry of 1

So adding 5 and 7 in binary form looks somewhat like this.

- $\quad {}_{1}{}_{1}{}_{1}{}_{0} \leftarrow$ **Carry**
- $\quad\ 1\ 0\ 1$
- $+\ 1\ 1\ 1$
- $\quad \_\_\_\_\_$
- $\quad 1\ 1\ 0\ 0$

Starting from right, and moving to left

- $1 + 1 = 0$ with a carry of 1
- $0 + 1 = 1$ which is added to the carried value, which is 1, so $1 + 1 = 0$ with a carry of 1, again
- $1 + 1 = 0$ with a carry of 1. The result is added again to the carried value, which was 1, so $0 + 1 = 1$
- The remaining carried value, which is 1, is put as it is.

The result is $[(1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (0 \times 2^0)] = 12$.

Quantum addition circuit for one pair of qubits

We'll try making the adder circuit for addition of two numbers $(0-7)$. We'll need 3 qubits to represent each of the two numbers, which means a total of 6 qubits.

```
using Yao,YaoPlots
```

```
1×3 Array{ArrayReg{1,Complex{Float64},Array{Complex{Float64},2}},2}:
 ArrayReg{1, Complex{Float64}, Array...}
    active qubits: 1/1  …  ArrayReg{1, Complex{Float64}, Array...}
    active qubits: 1/1
```

```
begin
    FirstNumber = [ArrayReg(bit"1") ArrayReg(bit"1") ArrayReg(bit"1")] #The first number is 7
    SecondNumber = [ArrayReg(bit"1") ArrayReg(bit"0") ArrayReg(bit"1")] #The second number is 5
end
```

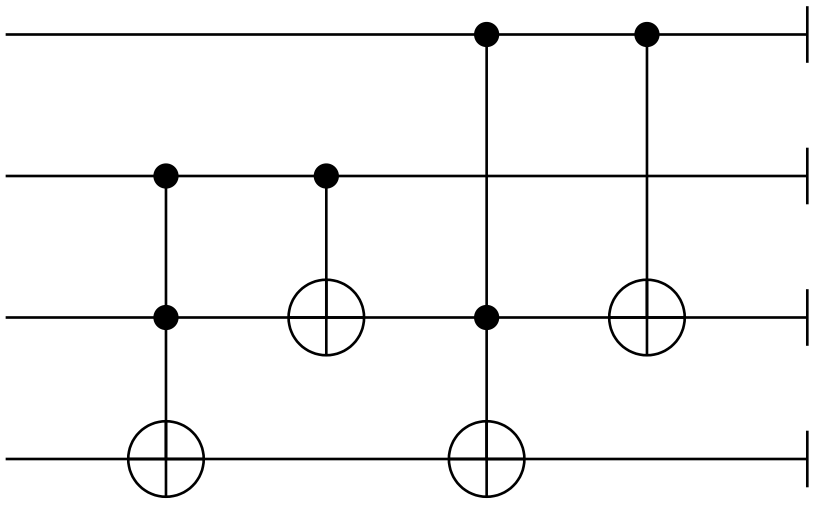
For adding each pair of qubits, we'll need two more qubits to hold the carried in value and carried out value. Since the carry out of the 1st pair acts as the carry in of second pair, the carry out of the second pair acts as the carry in of the 3rd pair, we need 4 more qubits for carry in and carry out, with

- 1 qubit for the carry-in of first pair
- 1 qubit for the carry-out of first pair and the carry-in of the second pair
- 1 qubit for the carry-out of second pair and the carry-in of the third pair
- 1 qubit for the carry-out of third pair

In total, we require a total of 10 qubits to add two numbers in the range $0-7$.

So lets try making the quantum circuit for adding two qubits. We need a qubit for carry-in, which will be zero for the rightmost pair, and a qubit for carry-out, which will act as the carry in for the next pair

Consider this circuit



```
begin
```

```
    OneqbQFA = chain(4, control(2:3, 4=>X), control(2, 3=>X), control([1 3], 4=>X), control(1, 3=>X))
    plot(OneqbQFA)
end
```

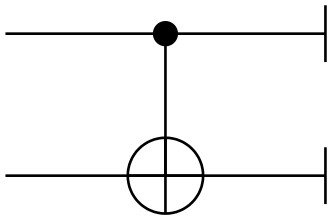
- The top qubit holds the carry-in value.
- The bottom qubit has the state $|0\rangle$.
- The 2nd and the 3rd qubits hold the pair to be added.

After passing throught the circuit,

- The top qubit holds the carry-in value.
- The bottom qubit holds the carry-out value.
- The 2nd qubit is as it was, while the 3rd qubit holds the addition of the 2nd and 3rd qubit.

The use of CX gate in Arithmetics

Consider this circuit



```
begin
    a = chain(2, control(1, 2=>X))
    plot(a)
end
```

Lets try passing different values through this circuit

```
BitBasis.BitStr{2,Int64}[00 ₍₂₎]
```

```
measure((ArrayReg(bit"00") |> a)) #The output is 00, when we passed 00
```

```
BitBasis.BitStr{2,Int64}[11 ₍₂₎]
```

```
measure(ArrayReg(bit"01") |> a) #The output is 11, when we passed 01
#Remember, the circuit takes the qubits as inputs, in reverse order. The rightmost qubit is the 1st qubit here, and since its |1⟩, the 2nd qubit gets flipped to |1⟩ too.
```

```
BitBasis.BitStr{2,Int64}[10 ₍₂₎]
```

```
measure(ArrayReg(bit"10") |> a) #The output is 10, when we passed 10
```

```
#Remember, the circuit takes the qubits as inputs, in reverse order. The rightmost
qubit is the 1st qubit here, and since its |0⟩, the 2nd qubit is left untouched.
```

```
BitBasis.BitStr{2,Int64}[01 ₍₂₎]
```

```
measure(ArrayReg(bit"11") |> a) #The output is 01, when we passed 11
#Remember, the circuit takes the qubits as inputs, in reverse order. The rightmost
qubit is the 1st qubit here, and since its |1⟩, the 2nd qubit is flipped to |0⟩.
```

Lets analyze the output

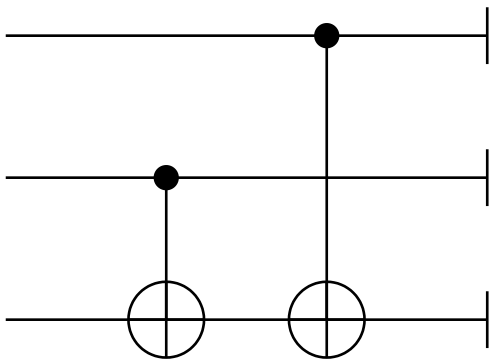
- When the top qubit is $|0\rangle$ and the bottom qubit is $|0\rangle$, the bottom qubit is left untouched.
- When the top qubit is $|1\rangle$ and the bottom qubit is $|0\rangle$, the bottom qubit is flipped to $|1\rangle$.
- When the top qubit is $|0\rangle$ and the bottom qubit is $|1\rangle$, the bottom qubit is left untouched.
- When the top qubit is $|1\rangle$ and the bottom qubit is $|1\rangle$, the bottom qubit is flipped to $|0\rangle$.

Compare this with,

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 0 = 1$
- $1 + 1 = 0$

In all the cases, the bottom qubit holds the added value.

How about adding 3 qubits?



```
begin
    b = chain(3, control(2, 3=>X), control(1, 3=>X))
    plot(b)
end
```

```
Array{BitBasis.BitStr{3,Int64},1}[BitBasis.BitStr{3,Int64}[000 ₍₂₎],  BitBasis.BitStr{3,I
```

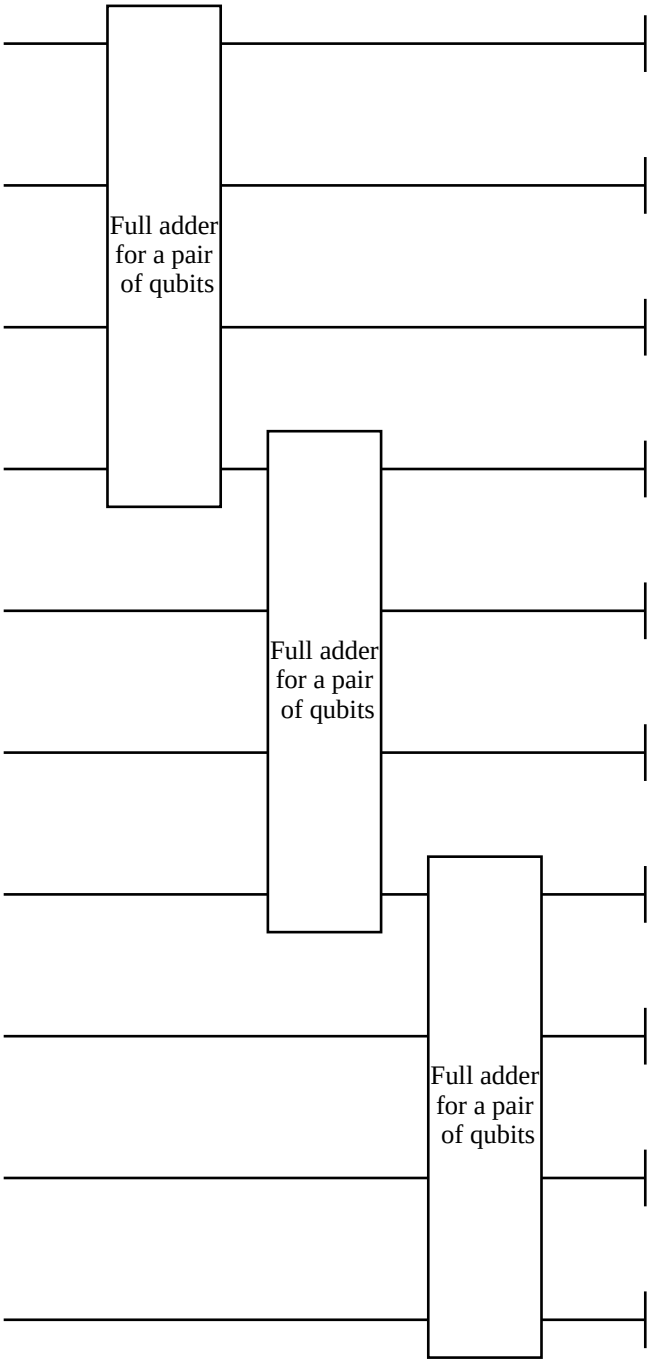
```
measure.([(ArrayReg(bit"000") |> b)
        (ArrayReg(bit"001") |> b)
        (ArrayReg(bit"010") |> b)
        (ArrayReg(bit"011") |> b)
```

```
        (ArrayReg(bit"110") |> b)
        (ArrayReg(bit"101") |> b)
        (ArrayReg(bit"100") |> b)
        (ArrayReg(bit"111") |> b)])
#If the output is not visible, click on the output to expand it.
```

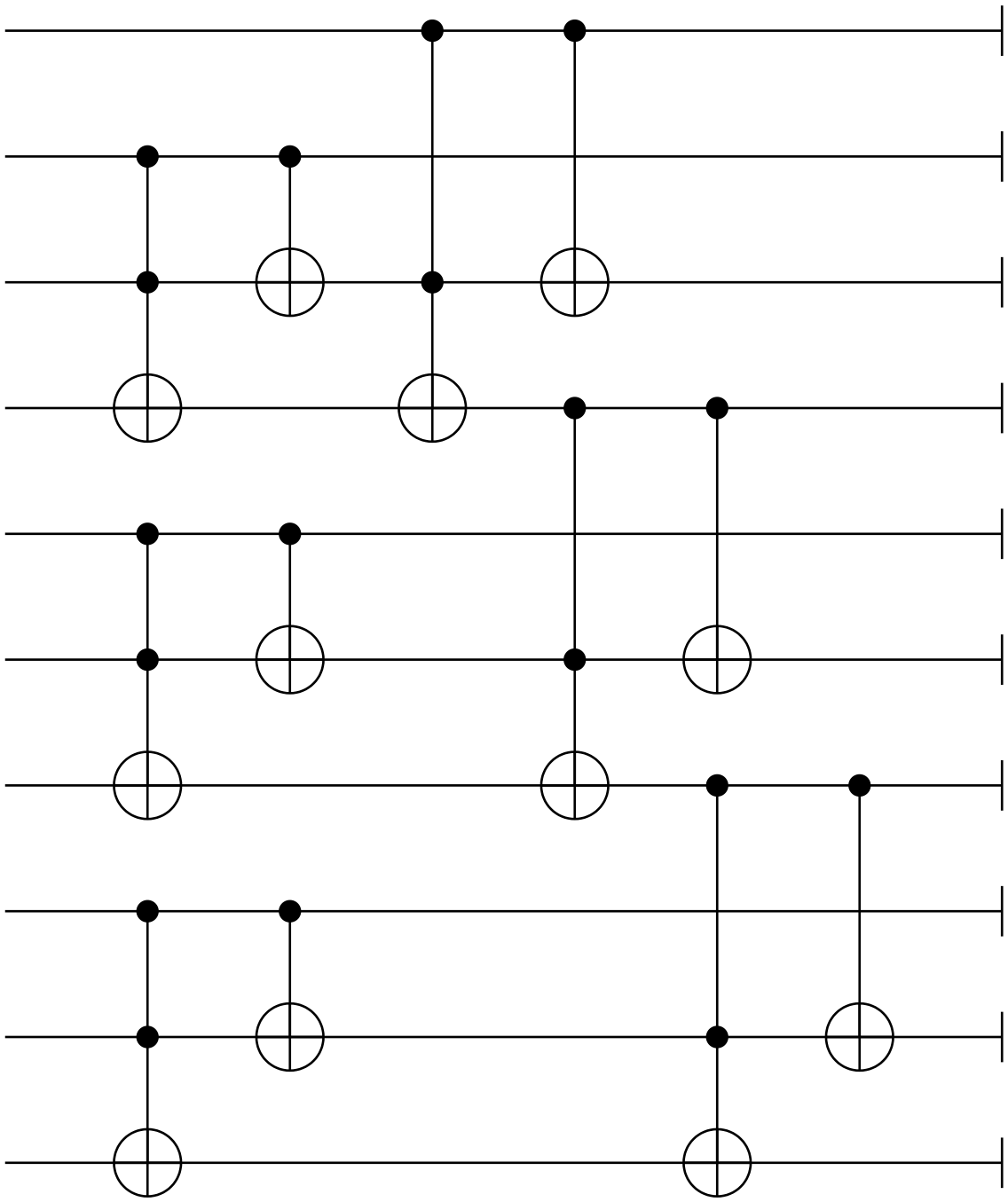
The toffoli gate acts as the carry-out. If both input qubits are $|1\rangle$, the 3rd qubit is flipped.

The Quantum Adder Circuit

We already made the circuit for adding two qubits. Remember that the carry-out for the first pair, becomes the carry in for the second pair. With this, here's the circuit for a quantum adder for 2 numbers between $0 - 7$.



Below is a complete circuit for quantum adder for 2 numbers between $0 - 7$.



```
begin
    nqbQFA = chain(10, put(1:4=>OneqbQFA), put(4:7=>OneqbQFA), put(7:10=>OneqbQFA))
    plot(nqbQFA)
end
```

The input to the circuit being -

```
input = ArrayReg{1, Complex{Float64}, Array...}
            active qubits: 10/10
```

```
input = join(zero_state(1), SecondNumber[1], FirstNumber[1], zero_state(1),
SecondNumber[2], FirstNumber[2], zero_state(1), SecondNumber[3], FirstNumber[3],
zero_state(1))
```

```
results =
```

```
▸ BitBasis.BitStr{10,Int64}[1111011010 ₍₂₎, 1111011010 ₍₂₎, 1111011010 ₍₂₎, 1111011010 ₍₂
```

```
results = input |> nqbQFA |> r->measure(r, nshots=1024)
```

```
stringres = "0101101111"
```

```
stringres = reverse(string(Int(results[1]), base=2, pad=10)) #To convert it to string
```

Remember, the 3, 6 and 9th qubits hold the added values. The 10 qubit holds the final carry-out, if any.

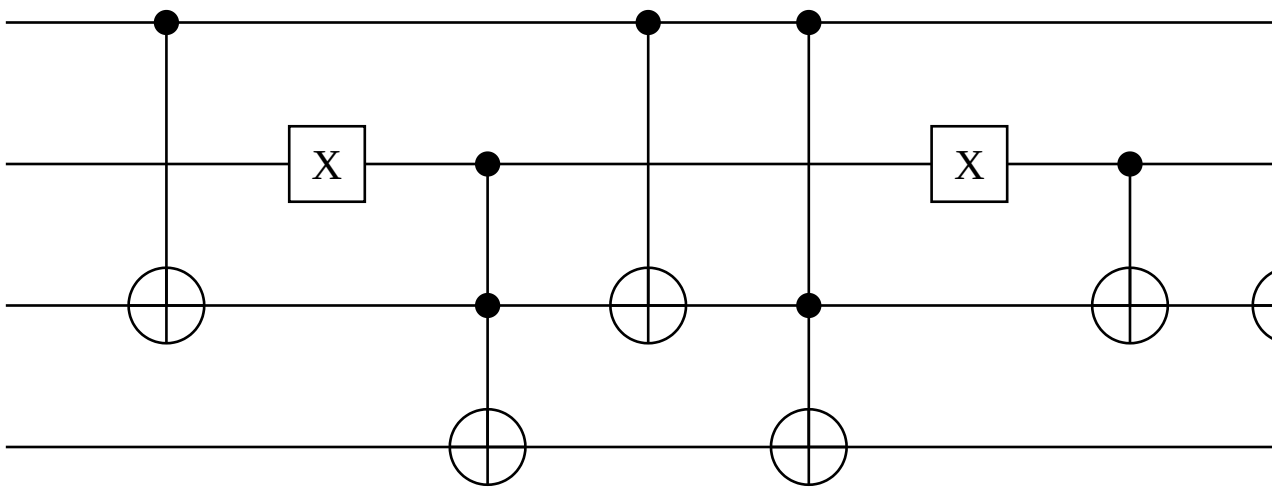
```
output = 12
```

```
output = parse(Int64, reverse(stringres[3] * stringres[6] * stringres[9] *
stringres[10]), base=2)
```

Quantum subtractor

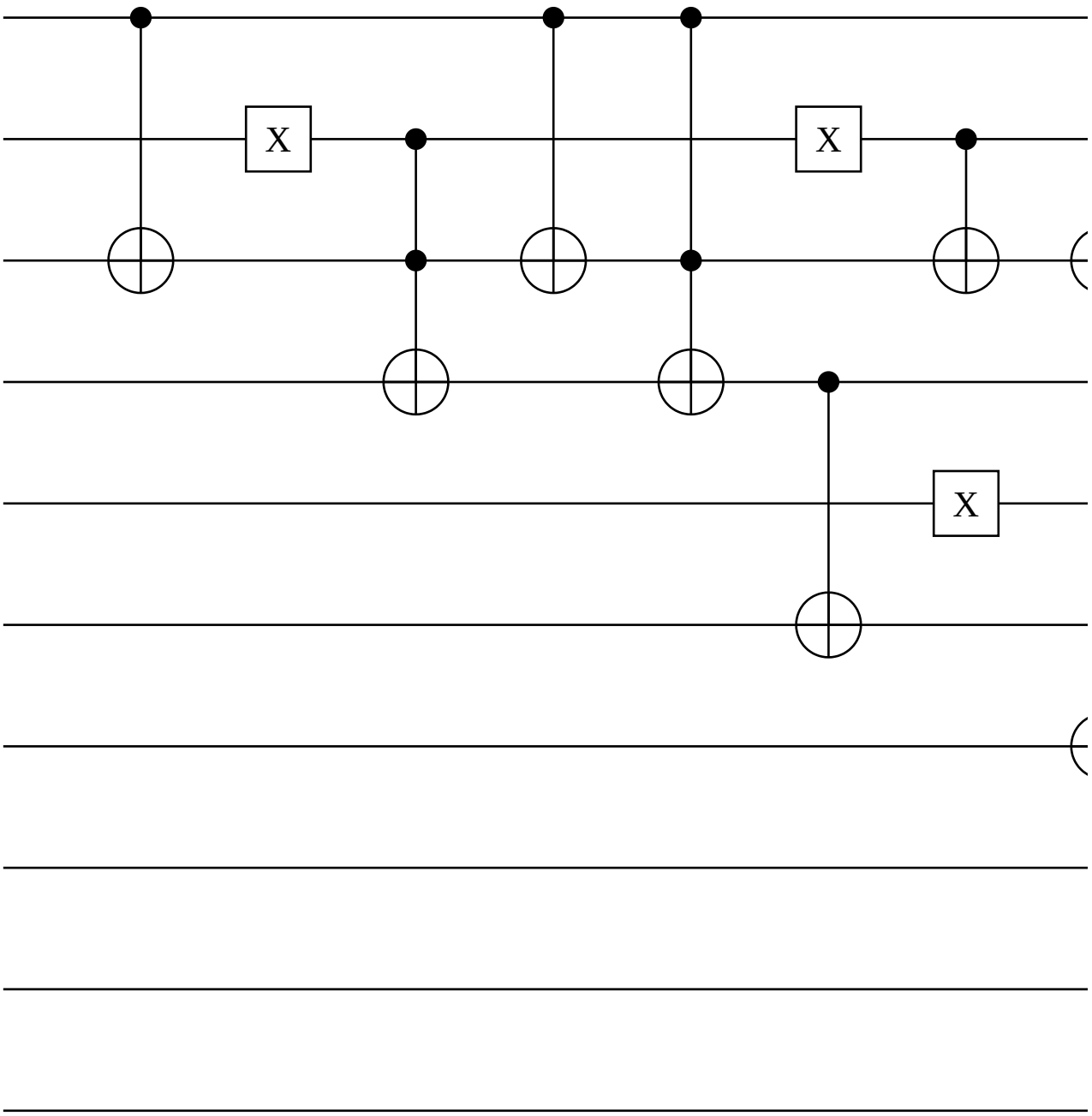
Binary subtraction has a borrow instead of carry. The rules of binary subtraction look somewhat like this.

- $0 - 0 = 0$
- $0 - 1 = 1$ with a borrow of 1
- $1 - 0 = 1$
- $1 - 1 = 0$




```
begin
    OneqbSubtractor = chain(4, control(1, 3=>X), put(2=>X), control(2:3, 4=>X),
control(1, 3=>X), control([1 3], 4=>X), put(2=>X), control(2, 3=>X), control(1,
3=>X))
    plot(OneqbSubtractor)
```

```
end
```




```
begin
    nqbQFS = chain(10, put(1:4=>OneqbSubtractor), put(4:7=>OneqbSubtractor), put(7:10=>OneqbSubtractor))
    plot(nqbQFS)
end
```

▸ BitBasis.BitStr{4,Int64}[1100 ₍₂₎]

```
measure(join(zero_state(1), ArrayReg(bit"1"), ArrayReg(bit"0"),zero_state(1)) |> OneqbSubtractor)
```

```
x = 1×3 Array{ArrayReg{1,Complex{Float64},Array{Complex{Float64},2}},2}:
     ArrayReg{1, Complex{Float64}, Array...}
        active qubits: 1/1  …  ArrayReg{1, Complex{Float64}, Array...}
        active qubits: 1/1
```

```
x = [ArrayReg(bit"1") ArrayReg(bit"0") ArrayReg(bit"0")]
```

```
y = 1×3 Array{ArrayReg{1,Complex{Float64},Array{Complex{Float64},2}},2}:
     ArrayReg{1, Complex{Float64}, Array...}
        active qubits: 1/1  …  ArrayReg{1, Complex{Float64}, Array...}
        active qubits: 1/1
```

```
y = [ArrayReg(bit"0") ArrayReg(bit"1") ArrayReg(bit"0")]
```

```
input1 = ArrayReg{1, Complex{Float64}, Array...}
             active qubits: 10/10
```

```
input1 = join(zero_state(1), y[1], x[1], zero_state(1), y[2], x[2], zero_state(1),
y[3], x[3], zero_state(1))
```

```
result =
```

```
▸ BitBasis.BitStr{10,Int64}[0011100000 ₍₂₎, 0011100000 ₍₂₎, 0011100000 ₍₂₎, 0011100000 ₍₂
```

```
result = input1 |> nqbQFS |> r->measure(r, nshots=1024)
```

```
stringsub = "0000011100"
```

```
stringsub = reverse(string(Int(result[1]), base=2, pad=10)) #To convert it to string
```

```
out = 2
```

```
out =  parse(Int64, reverse(stringsub[3] * stringsub[6] * stringsub[9] *
stringsub[10]), base=2)
```

This circuit only subtracts numbers if the answer is expected to be positive. It can't solve for calculations like $5 - 6 = -1$.