```
(******************)

(* This notebook uses the QuESTlink system to
 enable Mathematica to do efficient quantum emulation. *)
(* As a demontration, there are functions that create
 an addition circuit performing A,B --> A+B, B *)
(* The method is that of Draper's from
 arXiv:quant-ph/0008033 and it uses Quantum Fourier Transform *)

(*****************)
```

Make sure to run the two lines below -- they are the ones that 'upgrade' Mathematica with all the QuESTlink functionality for quantum simulation

```
Import["https://qtechtheory.org/questlink.m"];
CreateDownloadedQuESTEnv[];
```

Run the code immediately below but don't worry about the details -- look further below for the fun part of the demo.

In more detail: The functions below construct circuits for Draper's scheme to sum two numbers. Note that Draper's paper uses **control-phase = Diag{1,1,1, exp(i ϕ)}** gates which are not presently supported by QuESTlink.
We hence instead use **C[Rx]=Diag{1,1,exp(-i ϕ/2),exp(i ϕ/2)}** gates, and so an additional **single-qubit Rz=Diag{exp(-ϕ/4),exp(ϕ/4)}** is put on the control qubit to create control-phase [up to irrelevant global phase]

```
circuitForQFT[lowQ_, highQ_]:= Module[
    {numQ,circ,outLp,inLp,phase,Sy},

    numQ = highQ - lowQ + 1;
    If[numQ ≤ 0,
        Return @ Message[circuitForQFT::error, lowQ, highQ]];

    circ = {H_numQ-1};

    For[outLp=numQ-1, outLp>0, outLp--,
        For[inLp=outLp-1, inLp≥0, inLp--,
            phase = Pi/2^(outLp-inLp);
            AppendTo[circ, C_inLp[Rz_outLp[phase]]]];

        AppendTo[circ, Rz_outLp-1[Sum[1/2^(1+p),{p,1,numQ-outLp}] * Pi]];
        AppendTo[circ,H_outLp-1]];
```

```
    circ = circ /. Table[Sy_{numQ-i} → Sy_{highQ+1-i}, {i,numQ}];
    Append[circ, G[Sum[(numQ-p)/2^(2+p), {p,1,numQ-outLp}] * Pi]]
]



circuitAddBtoQFTA[lowQa_,highQa_,lowQb_,highQb_]:= Module[
    {circ={},numQa,numQb,aSignif,bSignif,phaseFixers,globalPhase,a,b,c,q1,ϕ1,ϕ2},

    numQa = highQa-lowQa+1;
    numQb = highQb-lowQb+1;

    If[(numQa<1) || (numQb<1) || (Intersection[Range[lowQa,highQa],Range[lowQb,highQb]
        Return @ Message[circuitAddBsqrToQFTA::error, lowQa, highQa, lowQb, highQb]];

    For[aSignif=numQa-1, aSignif≥ 0, aSignif--, (* we loop through the bits of a, most
        For[bSignif=Min[aSignif,numQb-1], bSignif≥ 0, bSignif--,
            AppendTo[circ, C_{lowQb+bSignif}[Rz_{lowQa+aSignif}[Pi/2^(aSignif-bSignif)]]]]];

    phaseFixers = circ /. C_{q1_}[Rz_{q2_}[ϕ1_]] → Rz_{q1}[ϕ1/2] //. {a___,Rz_{q1_}[ϕ1_],b___,Rz_{q1_}[ϕ2
    globalPhase = phaseFixers /. Rz_{q1_}[ϕ1_] → G[ϕ1/2] //. {a___,G[ϕ1_],b___,G[ϕ2_],c___
    Join[circ, phaseFixers, globalPhase]
]


circuitForAdditionByQFT[lowQa_,highQa_,lowQb_,highQb_] := Module[
    {circ={},circ2={},ϕ,q},

    circ=circuitForQFT[lowQa,highQa];
    circ2=Reverse[ circ /. G[ϕ_] → G[-ϕ] /. Rz_q[ϕ_] → Rz_q[-ϕ]  ];
    Join[circ, circuitAddBtoQFTA[lowQa,highQa,lowQb,highQb], circ2]
]


circuitForQFT::error = "Invalid arguments (low=`1`, high=`2`).";
circuitAddBtoQFTA::error = "Invalid arguments (lowQa=`1`, highQa=`2`, lowQb=`3`, highQ
```

Below is the demonstration, new users of QuESTlink might like to look at it step-by-step

Choose the size of our registers (which we simulate with one large quantum register or Qureg)

```
numQubitsA = 6;
numQubitsB = 5;
totNumQubits = numQubitsA + numQubitsB;

qureg = CreateQureg[totNumQubits];
```

Choose the initial values to be represented by each register.

```
valueA = 17;
valueB = 27;
```

Ensure that these values can even fit into the respective registers, and that the sum will fit into the register currently holding A.
If not, no problem, the code will still act but bits will be 'lost'

```
valueA < 2^numQubitsA
valueB < 2^numQubitsB
valueA + valueB < 2^numQubitsA
```

```
True
```

```
True
```

```
True
```

These values will be encoded into classical states...

```
Ket @ IntegerString[valueA, 2, numQubitsA]
Ket @ IntegerString[valueB, 2, numQubitsB]
```

```
|010001⟩
```

```
|11011⟩
```

We'll prepare the initial state **B** to be on the **left** (most significant digits).

```
inputBA = Join[IntegerDigits[valueB, 2, numQubitsB],
  IntegerDigits[valueA, 2, numQubitsA]]
```

```
{1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1}
```

We'll make a circuit that creates that initial state when it acts on all-zero, by flipping the relevant qubits (remember QuEST labels qubits starting from 0)
The little loop below creates the circuit -- there are neater, one-line ways to do this in Mathematica, but this method may be easiest on the eye for non-experts

```
myCircuit = {};
For[i = 1, i ≤ totNumQubits, i++,
  If[inputBA[[i]] == 1, AppendTo[myCircuit, X_(totNumQubits-i)]]
 ];
myCircuit
InitZeroState[qureg];
ApplyCircuit[myCircuit, qureg];
```

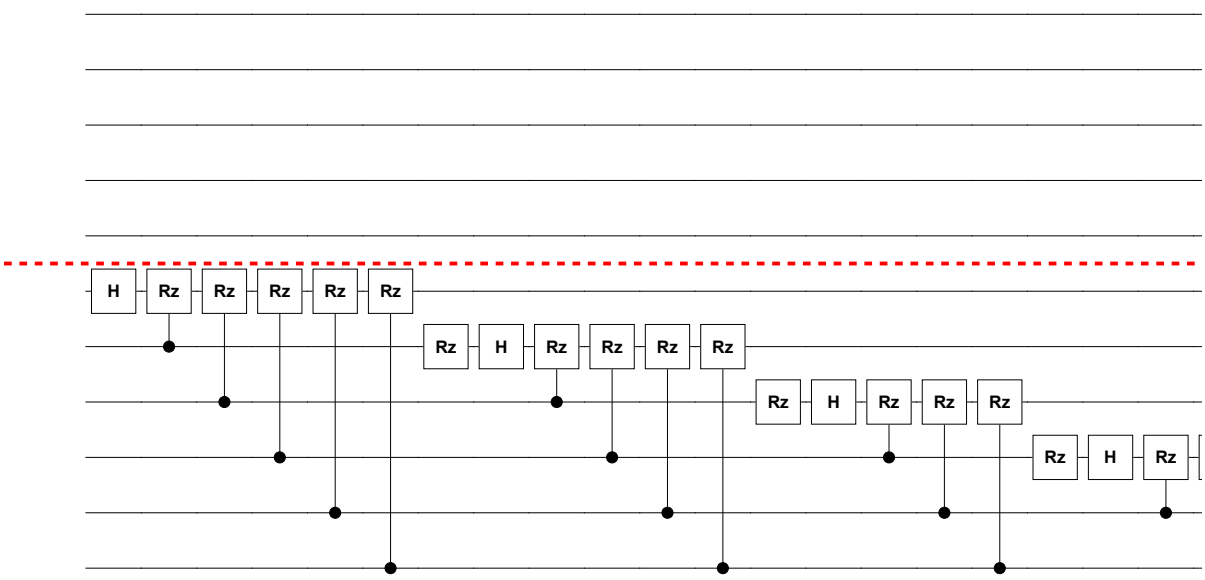
```
{X10, X9, X7, X6, X4, X0}
```

Consider the circuit for computing the sum of these registers. A red line separates the two input

registers.

```
circ = circuitForAdditionByQFT[0,
   numQubitsA - 1, numQubitsA, numQubitsA + numQubitsB - 1];

DrawCircuit[circ,
     GridLines → {{}, {numQubitsA}},
     GridLinesStyle → Directive[Thick, Red, Dashed]]
```



Let's apply this circuit and study its output state.

```
ApplyCircuit[circ, qureg];
outState = Chop @ GetQuregMatrix[qureg];
```

If successful, the final state is classical and so has a real maximum probability amplitude:

```
Max[outState]
```

```
1.
```

This was amplitude with index:

```
ind = Position[outState, Max[outState]][[1, 1]]
```

```
1773
```

and hence corresponds to the full-Hilbert computational basis state:

```
Ket @ IntegerString[ind - 1, 2, totNumQubits]
```

```
|11011101100⟩
```

Observe that the left register **B** remains in its encoded input state, while right register **A** now encodes...

```
IntegerDigits[ind - 1, 2, totNumQubits][[-numQubitsA ;;]]
FromDigits[%, 2]
```

```
{1, 0, 1, 1, 0, 0}
```

```
44
```

If successful, this is the sum of our initial encoded values:

```
valueA + valueB
```

```
44
```