Definitions

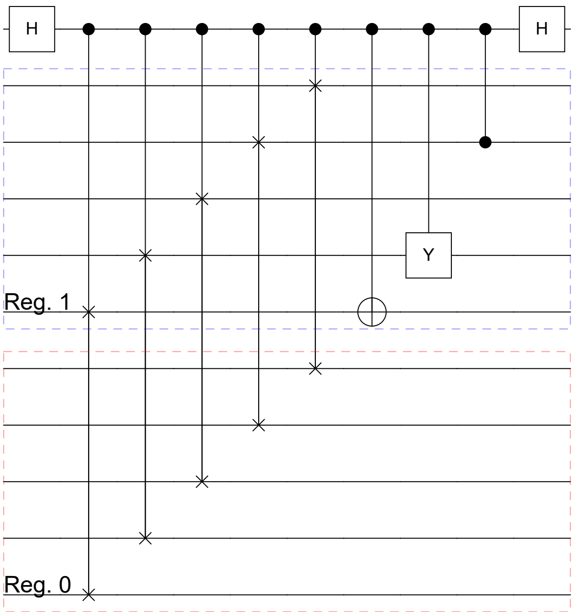
In[∘]:=

```
(* Import QuESTlink *)
Import["https://qtechtheory.org/QuESTlink.m"];
CreateDownloadedQuESTEnv[];

(* controlled SWAP operation between two full registers *)
cSWAPregs[reg1_, reg2_, qubitsperReg_, controlQB_] := Table[
  C_controlQB[SWAP_reg1*qubitsperReg+k,reg2*qubitsperReg+k], {k, 0, qubitsperReg - 1}]

(* Derangement circuit for 2 copies *)
D2[qubitsperReg_, observable_] := With[{controlQB = 2 * qubitsperReg},
  Join[{H_controlQB}, cSWAPregs[reg1 = 0, reg2 = 1, qubitsperReg, controlQB],
   Circuit[Evaluate@observable] /.
    {Id_n_ :> Id_n+qubitsperReg, O_n_ :> C_controlQB[O_n+qubitsperReg]}
   , {H_controlQB}]
 ]

isValidDerangement[swaps1_, nCopies_] := With[{cycles =
     PermutationProduct @@ Table[Cycles[{swaps1[[k]] + 1}], {k, 1, Length@swaps1}]},
   (* We check two conditions:
          1) the cyclic permutation representation of the swaps is a full n-cycle,
     where n is the number of copies
          2) the largest index that the permutations act
    on is identical to the number of copies
    *)
   {Dimensions[cycles /. {Cycles[cc_] :> cc}] == {1, nCopies} &&
     nCopies == Max[swaps1 + 1], cycles}
  ];

(* Derangement circuit for n copies *)
Dn[nCopies_, swaps_, qubitsperReg_, observable_] :=
 With[{controlQB = nCopies * qubitsperReg},
  Join[{H_controlQB},
   Flatten@Table[cSWAPregs[reg1 = swaps[[k, 1]],
      reg2 = swaps[[k, 2]], qubitsperReg, controlQB], {k, 1, Length@swaps}],
   Circuit[Evaluate@observable] /. {Id_n_ :> Id_n+qubitsperReg*(nCopies-1),
     O_n_ :> C_controlQB[O_n+qubitsperReg*(nCopies-1)]}
   , {H_controlQB}]
 ]

(* Draw derangement circuit with highligting the different registers *)
txt[in_] := Text[Style[in, FontFamily → "Times New Roman", FontSize → 12]]

DrawDerangementCircuit[derangementCirc_, numQs_, numCopies_] :=
 DrawCircuit[derangementCirc,
  Epilog → Join[{
       FaceForm[None], EdgeForm[Directive[Red, Dashed]]},
    Flatten@Table[
       {Text[
       Style[StringForm["Reg. ``", k], FontSize → 12], {0, k * numQs + .4}, {-1, -1}],
```

```
        EdgeForm[Directive[{Red, Blue}[[Mod[k, 2] + 1]], Dashed, Opacity[0.4]]],
            Rectangle[{0, k * numQs + .2}, {Length[derangementCirc],
          (k + 1) numQs - .2}]], {k, 0, numCopies - 1}]
        ]
 ]

(* Create n copies of a circuit -- this
 can be the input of the derangement circuit *)
CopiesOfCircuit[circuit_, numCopies_, qubitsperReg_] :=
 Flatten@Table[circuit /. {gate_n_ :> gate_(n+k*qubitsperReg),
     gate_n1_,n2_ :> gate_(n1+k*qubitsperReg,n2+k*qubitsperReg)}, {k, 0, numCopies - 1}]
```

1) Example of a simple derangement circuit

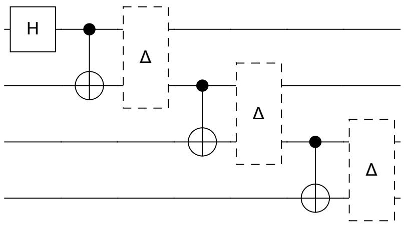
In[]:=
```
(* number of qubits in a register *)
numQs = 5;
(* set observable to be measured *)
observable = X_0 Y_1 Z_3;
(* generate derangement circuit for 2 copies *)
derangementCirc = D2[numQs, observable];
(* Draw derangement circuit and highligt the different registers *)
DrawDerangementCircuit[derangementCirc, numQs, 2]
```

Out[]=


2) Mitigate errors in noisy circuits

Noisy circuit that generates GHZ states

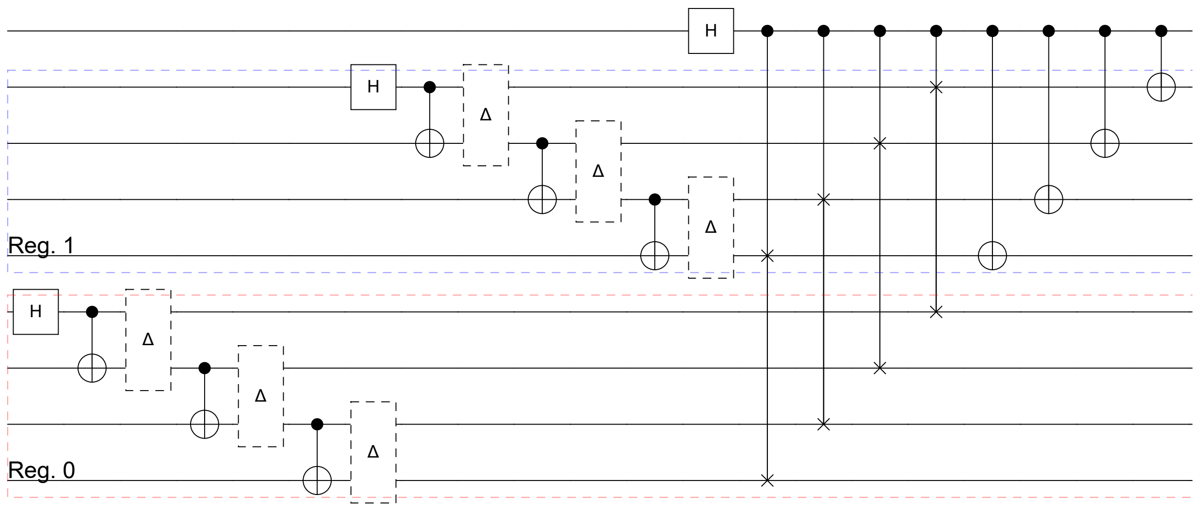
In[ ]:=
```
numQs = 4;
noisyGHZcirc =
  Join[{H_(numQs-1)}, Flatten@Table[{C_n[X_(n-1)], Depol_(n,n-1)[ϵ]}, {n, numQs - 1, 1, -1}]];
Print["Noisy GHZ circuit: " DrawCircuit[noisyGHZcirc]]

(* set observable with -- known ideal expectation value is 1*)
observable = Product[X_k, {k, 0, numQs - 1}];
idealExpectation = 1.;
(* compute expectation value errors for various levels of noise *)
{ρ, ϕ} = CreateDensityQuregs[numQs, 2];
errors = Table[
   ApplyCircuit[noisyGHZcirc /. {ϵ → 10^x}, InitZeroState@ρ];
   {10^x, Abs[CalcExpecPauliSum[ρ, observable, ϕ] - idealExpectation]}
   , {x, -3, -1, 0.2}];
DestroyAllQuregs[];
```

Noisy GHZ circuit: 

Setting up derangement circuit for 2 GHZ copies

In[*]:=
```
(* derangement circuit for 2 copies
 (identity observable required for correct normalisation) *)
{derangementCirc, derangementCircId} = D2[numQs, #] & /@ {observable, Id0};
{fullCirc, fullCircId} = Join[CopiesOfCircuit[noisyGHZcirc, 2, numQs], #] & /@
   {derangementCirc, derangementCircId};

Print["Full derangement circuit takes 2 copies of the GHZ preparation circuits: ",
  DrawDerangementCircuit[fullCirc, numQs, 2]
 ];
```

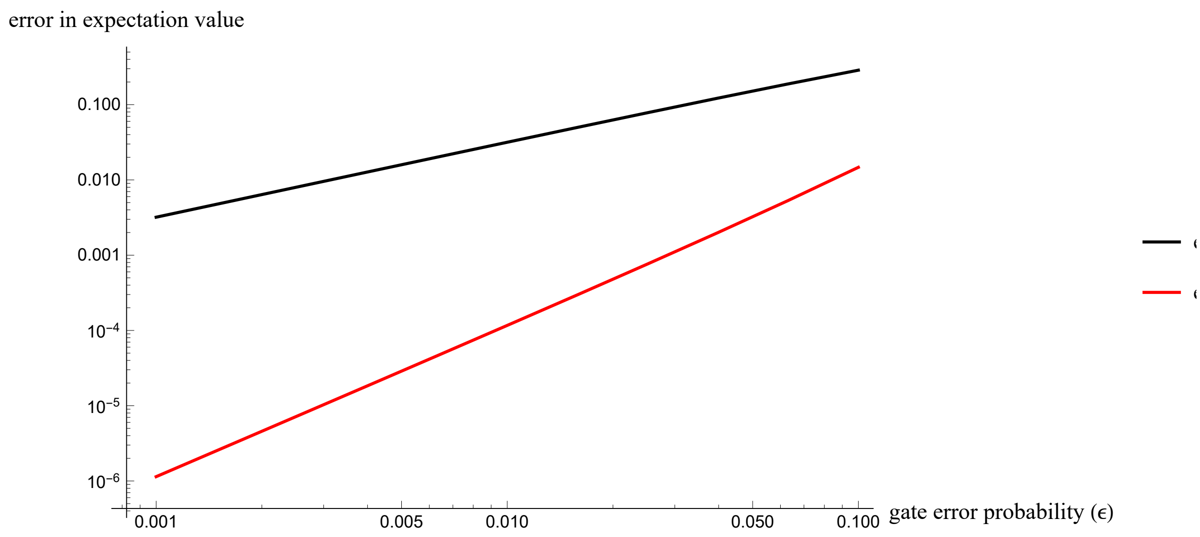
Full derangement circuit takes 2 copies of the GHZ preparation circuits:



Compare the mitigated and unmitigated errors

In[•]:=
```
ρ = CreateDensityQureg[2 * numQs + 1];
(* compute expectation value errors for various levels of noise *)
errorsWithDerangement = Table[
   ApplyCircuit[fullCirc /. {ϵ → 10^x}, InitZeroState@ρ];
   term1 = 2 CalcProbOfOutcome[ρ, 2 * numQs, 0] - 1;
   ApplyCircuit[fullCircId /. {ϵ → 10^x}, InitZeroState@ρ];
   term2 = 2 CalcProbOfOutcome[ρ, 2 * numQs, 0] - 1;
   {10^x, Abs[term1 / term2 - idealExpectation]}
   , {x, -3, -1, 0.2}];
DestroyAllQuregs[];

plot = ListLogLogPlot[{errors, errorsWithDerangement},
   PlotLegends →
    {txt@"error without mitigation", txt@"error with derangement circuit"},
   AxesLabel → {txt@"gate error probability (ϵ)", txt@"error in expectation value"}
   , ImageSize → Large, Joined → True, PlotStyle → {Black, Red}
  ];
Print["Plotting errors with and without error mitigation:", plot];
```

Plotting errors with and without error mitigation:



3) General derangement circuits for *n* copies

Checking validity of permutations

In[ ]:=
```
(* For more than 2 copies we have a number of different
 possiblities to implement derangements via different permutations *)
nCopies = 4;
numQs = 2;
observable = X0 Z1;
derangementPatterns = {
    {{0, 1}, {0, 2}, {0, 3}}
    ,
    {{0, 1}, {1, 2}, {2, 3}}
    ,
    {{0, 1}, {1, 2}, {1, 3}}
  };
(* we can check whether the given set of
 permutations represent a valid derangement operation *)
Print["Validity of derangement ", #, " and its cyclic permutation representation:  ",
     isValidDerangement[#, nCopies]] & /@ derangementPatterns;
```

```
Validity of derangement {{0, 1}, {0, 2}, {0, 3}}
  and its cyclic permutation representation:  {True, Cycles[{{1, 2, 3, 4}}]}
Validity of derangement {{0, 1}, {1, 2}, {2, 3}}
  and its cyclic permutation representation:  {True, Cycles[{{1, 4, 3, 2}}]}
Validity of derangement {{0, 1}, {1, 2}, {1, 3}}
  and its cyclic permutation representation:  {True, Cycles[{{1, 3, 4, 2}}]}
```

generate random ρ and compute the expectation value $\mathrm{Tr}[O\,\rho^n]$ for reference

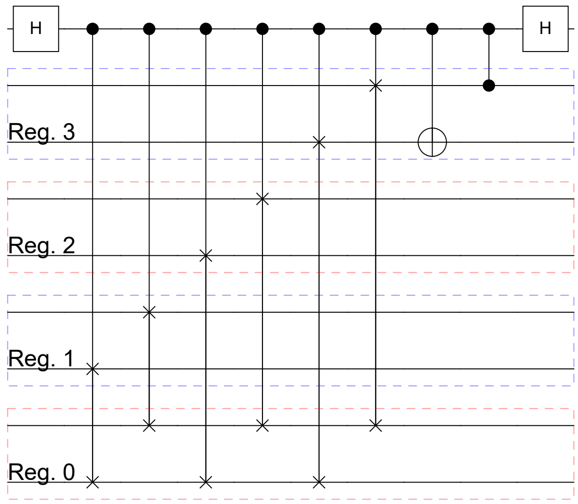
In[ ]:=
```
rand = With[{r = RandomComplex[{-1 - i, 1 + i}, {2^numQs, 2^numQs}]},
        r . r† / Tr[r . r†]];
{randρ, ϕ} = CreateDensityQuregs[numQs, 2];
SetQuregMatrix[randρ, MatrixPower[rand, nCopies]];

Print["expectation value ",
  StringForm["Tr[O ρ``]", nCopies], " of the observable O is: ",
  CalcExpecPauliSum[randρ, observable, ϕ]];
DestroyAllQuregs[];
```

```
expectation value Tr[O ρ^4] of the observable O is: -0.0724807
```

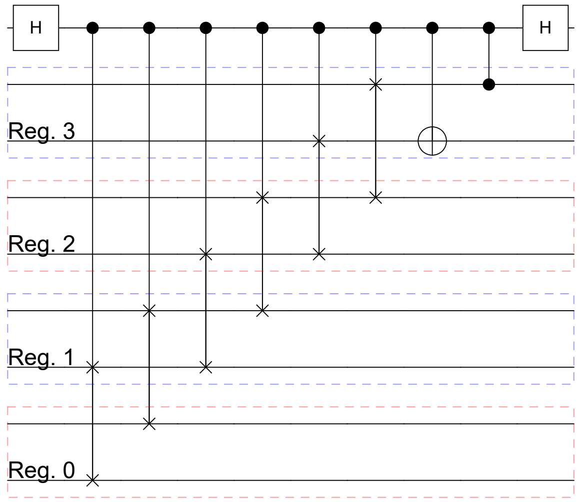
Simulate various *n*-copy derangement circuits

In[ ]:=
```
(* Generate n copies of the random state plus the ancilla qubit *)
randCopies = Module[{kp},
    kp = KroneckerProduct[{{1, 0}, {0, 0}}, KroneckerProduct[rand, rand]];
    Do[kp = KroneckerProduct[kp, rand], nCopies - 2];
    kp
   ];
ρ = CreateDensityQureg[nCopies * numQs + 1];

Do[
 swaps = derangementPatterns[[k]];
 derangementCirc = Dn[nCopies, swaps, numQs, observable];
 SetQuregMatrix[ρ, randCopies];
 ApplyCircuit[derangementCirc, ρ];
 Print["Derangement: ",
   swaps, ", expectation value: ", 2 CalcProbOfOutcome[ρ, nCopies * numQs, 0] - 1,
  " via the circuit:\n",
  DrawDerangementCircuit[derangementCirc, numQs, nCopies]
 ];

 , {k, 1, Length@derangementPatterns}]

DestroyAllQuregs[];
```

Derangement: {{0, 1}, {0, 2}, {0, 3}}, expectation value: -0.0724807 via the circuit:



Derangement: {{0, 1}, {1, 2}, {2, 3}}, expectation value: -0.0724807 via the circuit:



Derangement: {{0, 1}, {1, 2}, {1, 3}}, expectation value: -0.0724807 via the circuit: