Imports

In [2]:
```
import numpy as np
import qiskit.quantum_info as qi
from math import *
import scipy
# Importing standard Qiskit libraries
from qiskit import *
from qiskit.tools.jupyter import *
from qiskit.visualization import *
#from ibm_quantum_widgets import *
from qiskit.providers.aer import QasmSimulator
from qiskit.circuit.library import MCMT
```

In [3]:
```
from qiskit.quantum_info.operators import Operator, Pauli
from qiskit.circuit.library import QFT,RYGate
from qiskit.quantum_info import random_statevector
from qiskit.opflow import X,Y,Z,I,CX
from math import sqrt
pi = np.pi
sin = np.sin
cos = np.cos
exp = np.exp
```

In [4]:
```
import matplotlib.pyplot as plt
```

Definitions

In [5]:
```
#Operator to gate convertor
def qc(operator):
    qubit_list = list(range(int(np.log(len(operator))/np.log(2))))
    qc = QuantumCircuit(len(qubit_list))
    qc.unitary(operator,qubit_list)
    return qc
```

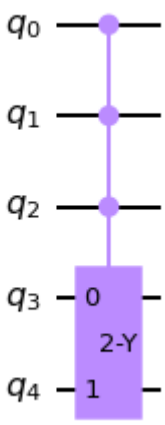
In [6]:
```
# will help to obtain the parameters for formation of B gate
#t here is len(betas-1), in this scheme
def theta_computer(betas):
    t = len(betas)-1
    A = sum(betas)
    alphas = np.sqrt(np.array(betas)/A)
    thetas = []
    for i in range(t):
        if i == 0 :
            thetas.append(acos(alphas[i]))
        else:
            fac = 0
            j = 0
            while j<i :
                fac+=alphas[j]**2
                j+=1
            thetas.append(acos(alphas[i]/sqrt(1-fac)))
    return thetas,A
```

In [7]:
```
#create gate
def negate(circ,neg = True):
    op = qi.Operator(circ)
    if neg == True:
        op = -op.data
    else:
        op = op.data
    return op
```

In [8]:
```
simulator = Aer.get_backend('statevector_simulator')
qasm = Aer.get_backend('qasm_simulator')
```

In [9]:
```
ccz = MCMT('cz',2,1)
ccx = MCMT('cx',2,1)
ccy = MCMT('cy',2,1)
cccx = MCMT('cx',3,1)
cccy = MCMT('cy',3,1)
cccz = MCMT('cz',3,1)
cccxx = MCMT('cx',3,2)
cccyy = MCMT('cy',3,2)
cccyy.draw('mpl')
```

Out[9]:



Defining Gates

leave this portion as I define the gates for each part in thoir individual sections

In [120…
```
zi = [[1,0,0,0],[0,1,0,0],[0,0,-1,0],[0,0,0,-1]]
iz_ = [[-1,0,0,0],[0,1,0,0],[0,0,-1,0],[0,0,0,1]]
xx_ = [[0,0,0,-1],[0,0,-1,0],[0,-1,0,0],[-1,0,0,0]]
yy_ = [[0,0,0,1],[0,0,-1,0],[0,-1,0,0],[1,0,0,0]]
xi = [[0,0,1,0],[0,0,0,1],[1,0,0,0],[0,1,0,0]]
ZI = qc(ZI).to_gate(label = 'ZI').control(4)
IZ_ = qc(IZ_).to_gate(label = 'IZ_').control(4)
XX_ = qc(XX_).to_gate(label = 'XX_').control(4)
YY_ = qc(YY_).to_gate(label = 'YY_').control(4)
XI = qc(XI).to_gate(label = 'XI').control(4)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-120-733d716be567> in <module>
      4 yy_ = [[0,0,0,1],[0,0,-1,0],[0,-1,0,0],[1,0,0,0]]
      5 xi = [[0,0,1,0],[0,0,0,1],[1,0,0,0],[0,1,0,0]]
----> 6 ZI = qc(ZI).to_gate(label = 'ZI').control(4)
      7 IZ_ = qc(IZ_).to_gate(label = 'IZ_').control(4)
      8 XX_ = qc(XX_).to_gate(label = 'XX_').control(4)

<ipython-input-3-827520be8055> in qc(operator)
      1 #Operator to gate convertor
      2 def qc(operator):
----> 3     qubit_list = list(range(int(np.log(len(operator))/np.log(2))))
      4     qc = QuantumCircuit(len(qubit_list))
      5     qc.unitary(operator,qubit_list)

TypeError: object of type 'ControlledGate' has no len()
```

In [9]:
```
ZI_ = [[-1,0,0,0],[0,-1,0,0],[0,0,1,0],[0,0,0,1]]
IZ_ = [[-1,0,0,0],[0,1,0,0],[0,0,-1,0],[0,0,0,1]]
ZZ = [[1,0,0,0],[0,-1,0,0],[0,0,-1,0],[0,0,0,1]]
XI_ = [[0,0,-1,0],[0,0,0,-1],[-1,0,0,0],[0,-1,0,0]]
XZ_ = [[0,0,-1,0],[0,0,0,1],[-1,0,0,0],[0,1,0,0]]
IX_ = [[0,-1,0,0],[-1,0,0,0],[0,0,0,-1],[0,0,-1,0]]
ZX = [[0,1,0,0],[1,0,0,0],[0,0,0,-1],[0,0,-1,0]]
gZI_ = qc(ZI_).to_gate(label = 'ZI_').control(4)
gIZ_ = qc(IZ_).to_gate(label = 'IZ_').control(4)
gZZ = qc(ZZ).to_gate(label = 'ZZ').control(4)
gXI_ = qc(XI_).to_gate(label = 'XI_').control(4)
gXZ_ = qc(XZ_).to_gate(label = 'XZ_').control(4)
gIX_ = qc(IX_).to_gate(label = 'IX_').control(4)
gZX = qc(ZX).to_gate(label = 'ZX').control(4)
```

In [10]:
```
ZZ_ = [[-1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,-1]]
XZ  = [[0,0,1,0],[0,0,0,-1],[1,0,0,0],[0,-1,0,0]]
gXZ = qc(XZ).to_gate(label = 'XZ').control(4)
gZZ_ = qc(ZZ_).to_gate(label = 'ZZ_').control(4)
```

In [11]:

```
ziii = QuantumCircuit(4)
ziii.z(3)
ZIII = qc(negate(ziii,neg = False)).to_gate(label = 'ZIII').control(4)
izii = QuantumCircuit(4)
izii.z(2)
IZII_ = qc(negate(izii)).to_gate(label = 'IZII_').control(4)
iizi = QuantumCircuit(4)
iizi.z(1)
IIZI_ = qc(negate(iizi)).to_gate(label = 'IIZI_').control(4)
iiiz = QuantumCircuit(4)
iiiz.z(0)
IIIZ_ = qc(negate(iiiz)).to_gate(label = 'IIIZ_').control(4)
xxii = QuantumCircuit(4)
xxii.x([2,3])
XXII_ = qc(negate(xxii)).to_gate(label = 'XXII_').control(4)
yyii = QuantumCircuit(4)
yyii.y([2,3])
YYII_ = qc(negate(yyii)).to_gate(label = 'YYII_').control(4)
ixxi = QuantumCircuit(4)
ixxi.x([1,2])
IXXI_ = qc(negate(ixxi)).to_gate(label = 'IXXI_').control(4)
iyyi = QuantumCircuit(4)
iyyi.y([1,2])
IYYI_ = qc(negate(iyyi)).to_gate(label = 'IYYI_').control(4)
iixx = QuantumCircuit(4)
iixx.x([0,1])
IIXX_ = qc(negate(iixx)).to_gate(label = 'IIXX_').control(4)
iiyy = QuantumCircuit(4)
iiyy.y([0,1])
IIYY_ = qc(negate(iiyy)).to_gate(label = 'IIYY_').control(4)
IXXI = qc(negate(ixxi,neg = False)).to_gate(label = 'IXXI').control(4)
IYYI = qc(negate(iyyi,neg = False)).to_gate(label = 'IYYI').control(4)
IIXX = qc(negate(iixx,neg = False)).to_gate(label = 'IIXX').control(4)
IIYY = qc(negate(iiyy,neg = False)).to_gate(label = 'IIYY').control(4)
ixix = QuantumCircuit(4)
ixix.x([0,2])
IXIX_ = qc(negate(ixix)).to_gate(label = 'IXIX_').control(4)
iyiy = QuantumCircuit(4)
iyiy.y([0,2])
IYIY_ = qc(negate(iyiy)).to_gate(label = 'IYIY_').control(4)
xixi = QuantumCircuit(4)
xixi.x([1,3])
XIXI_ = qc(negate(xixi)).to_gate(label = 'XIXI_').control(4)
yiyi = QuantumCircuit(4)
yiyi.y([1,2])
YIYI_ = qc(negate(yiyi)).to_gate(label = 'YIYI_').control(4)
xiix = QuantumCircuit(4)
xiix.x([0,3])
XIIX_ = qc(negate(xiix)).to_gate(label = 'XIIX_').control(4)
yiiy = QuantumCircuit(4)
yiiy.y([0,3])
YIIY_ = qc(negate(yiiy)).to_gate(label = 'YIIY_').control(4)
```

In [12]:
```
#ZZ = [[1,0,0,0,0,0,0,0],[0,1,0,0,0,0,0,0],[0,0,-1,0,0,0,0,0],[0,0,0,-1,0,0,0,0],[0,0,0
#YY = [[0,0,0,0,0,0,-1,0],[0,0,0,0,0,0,0,-1],[0,0,0,0,1,0,0,0],[0,0,0,0,0,1,0,0],[0,0,1
#IX = [[0,0,1,0,0,0,0,0],[0,0,0,1,0,0,0,0],[1,0,0,0,0,0,0,0],[0,1,0,0,0,0,0,0],[0,0,0,0
#ZX_ = [[0,0,-1,0,0,0,0,0],[0,0,0,-1,0,0,0,0],[1,0,0,0,0,0,0,0],[0,-1,0,0,0,0,0,0],[0,0
#XI_ = [[0,0,0,0,-1,0,0,0],[0,0,0,0,0,-1,0,0],[0,0,0,0,0,0,-1,0],[0,0,0,0,0,0,0,-1],[-1
XZ_= [[0,0,-1,0],[0,0,0,1],[-1,0,0,0],[0,1,0,0]]
YY = [[0,0,0,-1],[0,0,1,0],[0,1,0,0],[-1,0,0,0]]
IX = [[0,1,0,0],[1,0,0,0],[0,0,0,1],[0,0,1,0]]
ZX_ =[[0,-1,0,0],[-1,0,0,0],[0,0,0,1],[0,0,1,0]]
#gZZ = qc(ZZ).to_gate(label = 'ZZI').control(4)
#gYY = qc(YY).to_gate(label = 'YYI').control(4)
#gIX = qc(IX).to_gate(label = 'IXI').control(4)
#gZX_ = qc(ZX_).to_gate(label = 'ZXI_').control(4)
#gXI_ = qc(XI_).to_gate(label = 'XII_').control(4)
gXZ_ = qc(XZ_).to_gate(label = 'XZ_').control(4)
gYY = qc(YY).to_gate(label = 'YY').control(4)
gIX = qc(IX).to_gate(label = 'IX').control(4)
gZX_ = qc(ZX_).to_gate(label = 'ZX_').control(4)
```

Questions

- 1) Is the entire implementation for N = 2 JWT and GC, EFT is complete and correct? -> true
- 2) Why does V implementation in N = 4 GC giving wierd results? not concluded
- 3) Why is N = 3 case tricky? no problem
- 4) Changing order for gate preparation step in N = 2 is not affecting the result but for N = 4 it is affecting and none is correct?
- 5) How does Pooja Ma'am implement LCU for N = 4, what are the differences (how does she handle the negative signs)? - > same way
- 6) Is swap or destructive swap test required here? If so, how to perform it? we don't need to apply this
- 7) If problems LCU cannot be sovled can we try Hadamard test?

Hamiltonians (EFT)

N = 2,GC

In [9]:
```
#Defining the Hamiltonian to obtain eigenstate
#1 qubit, GC
#H2_GC = (5.906709 * I ) - \
 #       (6.34329 * Z ) - \
  #      (4.28661 * X )
H2_GC = (-6.34329 * Z ) - \
        (4.28661 * X )
e,v = np.linalg.eig(H2_GC.to_matrix())
v = np.transpose(v) #to obtain eigen state of Hamiltonian
e,v
```

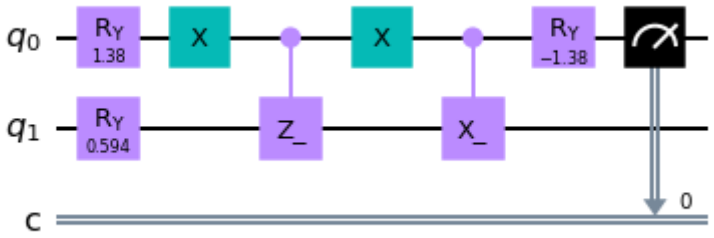
Out[9]:
```
(array([-7.65587051+0.j,  7.65587051+0.j]),
 array([[ 0.95617792+0.j,  0.29278626-0.j],
        [-0.29278626+0.j,  0.95617792+0.j]]))
```

In [83]:
```
#coefficients of hamiltonian stored in beta list
betas = [6.34329,4.28661]
#theta to create V matrices
theta,A = theta_computer(betas)
```

In [86]:
```
#unitary gates with sign consideration
z = [[-1,0],[0,1]]
x = [[0,-1],[-1,0]]
Z_ = qc(z).to_gate(label = 'Z_').control(1) #handling negative sign through matrices
X_ = qc(x).to_gate(label = 'X_').control(1) #gate
```

In [87]:
```
cir = QuantumCircuit(2,1)
cir.ry(0.594,1) #initial state which is groundstate
cir.ry(2*theta[0],0) # V gate
cir.x(0)
cir.append(Z_,[0,1]) #gate for -Z_0
cir.x(0)
cir.append(X_,[0,1]) #gate for -X_0
cir.ry(-2*theta[0],0)
cir.measure(0,0)
cir.draw('mpl')
```

Out[87]:



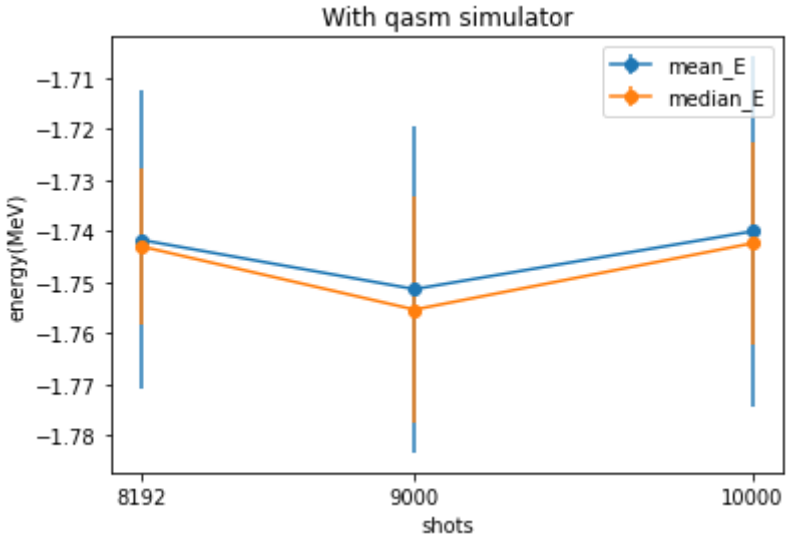
In [88]:
```
E = []
probs = []
shots_ = [8192]
for shots in shots_:
    E_temp = []
    prob_temp = []
    for runs in range(20):
        result = execute(cir, backend = simulator, shots = shots).result()
        count = result.get_counts(cir)
        prob_temp.append(count['0']/shots)
        E_temp.append(-A*np.sqrt(count['0']/shots)+5.906709)  #adding the extra coeffic
    E.append(E_temp)
    probs.append(prob_temp)
print(np.median(E))
```

```
-1.7529139009872936
```

In [216…

```
mean_E = []
std_E = []
med_E = []
mad_E = []
prob = []
for i in range(len(shots_)):
    mean_E.append(np.mean(E[i]))
    std_E.append(np.std(E[i]))
    med_E.append(np.median(E[i]))
    mad_E.append(mad(E[i]))
    prob.append(np.mean(probs[i]))

plt.errorbar(shots_,mean_E,yerr = std_E, label = 'mean_E',marker = 'o')
#plt.plot(shots,std_E,label = 'std_E')
plt.errorbar(shots_,med_E,yerr = mad_E,label = 'median_E',marker = 'o')
#plt.scatter(shots_,prob,label = 'prob')
#plt.plot(shots,mad_E,label = 'mad_E')
plt.xlabel('shots')
plt.ylabel('energy(MeV)')
plt.xticks([8192,9000,10000])
plt.title('With qasm simulator')
plt.legend(loc = 'best')
plt.savefig('DPEGc2.png')
```



In [193…

```
med_p_mad = np.median(np.array(E))+mad(np.array(E))
med_mad = np.median(E)-mad(np.array(E))
mean_p_std = np.mean(np.array(E))+np.std(np.array(E))
mean_std = np.mean(np.array(E))-np.std(np.array(E))
```

In [195…

```
med_p_mad,med_mad,mean_p_std,mean_std
```

Out[195…

```
(array([-1.73500581, -1.74237829, -1.71066793, -1.72674851, -1.73715182,
        -1.7097087 , -1.72191942, -1.73737565, -1.73714552, -1.74352584,
        -1.73593055, -1.74115973, -1.7230467 , -1.71853679, -1.73824924,
        -1.74010635, -1.73893176, -1.74304407, -1.72373249, -1.7103971 ]),
 array([-1.75239536, -1.74502288, -1.77673324, -1.76065266, -1.75024935,
        -1.77769246, -1.76548175, -1.75002552, -1.75025565, -1.74387532,
        -1.75147062, -1.74624144, -1.76435447, -1.76886438, -1.74915193,
        -1.74729482, -1.74846941, -1.7443571 , -1.76366867, -1.77700406]),
 -1.7121826009227985,
 -1.776573762505877)
```

for N = 2, JWT

In [14]:
```
#H2_op = (5.906709 * I ^ I) + \
#        (0.218291 * Z ^ I) - \
#        (6.125 * I ^ Z) - \
#        (2.143304 * X ^ X) - \
#        (2.143304 * Y ^ Y)
H2_op = (0.218291 * Z ^ I) - \
        (6.125 * I ^ Z) - \
        (2.143304 * X ^ X) - \
        (2.143304 * Y ^ Y)
#ingoring the identity part that we will add at the end
e,v = np.linalg.eig(H2_op.to_matrix())
v = np.transpose(v)
e,v[0]
```

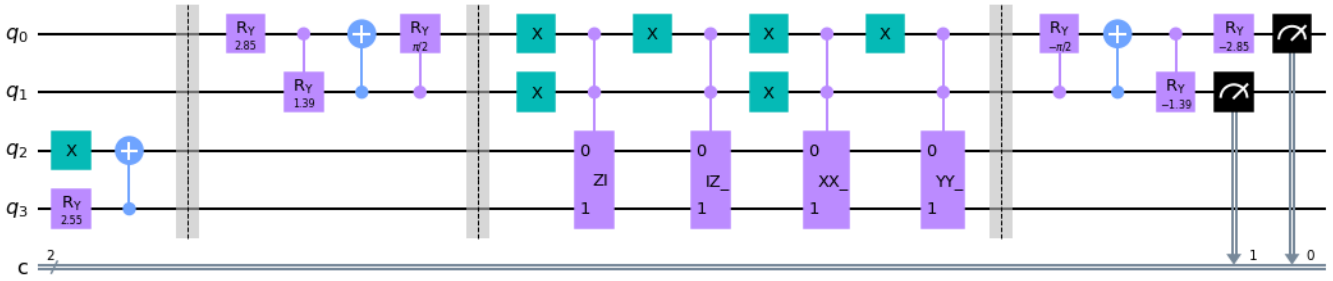
Out[14]:
```
(array([-7.65587022+0.j,  7.65587022+0.j, -5.906709  +0.j,  5.906709  +0.j]),
 array([-0.        -0.j,  0.29278612-0.j,  0.95617796+0.j, -0.        -0.j]))
```

In [20]:
```
betas = [0.218291,6.125,2.143304,2.143304]
thetas, A =theta_computer(betas)
```

In [21]:
```
#not sure which ordering is correct
zi = QuantumCircuit(2)
zi.z(0)
ZI = qc(negate(zi,neg = False)).to_gate(label = 'ZI').control(2)
iz = QuantumCircuit(2)
iz.z(1)
IZ_ = qc(negate(iz)).to_gate(label = 'IZ_').control(2)
xx = QuantumCircuit(2)
xx.x([0,1])
XX_ = qc(negate(xx)).to_gate(label = 'XX_').control(2)
yy = QuantumCircuit(2)
yy.y([0,1])
YY_ = qc(negate(yy)).to_gate(label = 'YY_').control(2)
```

In [22]:
```
H2_cir = QuantumCircuit(4,2)
#state preparation
H2_cir.ry(2.547304378732438,3)
H2_cir.x(2)
H2_cir.cx(3,2)
H2_cir.barrier()
#preparation of V gate using algorithm in Pooja Ma'am's paper
H2_cir.ry(2*thetas[0],0)
H2_cir.cry(2*thetas[1],0,1)
H2_cir.cx(1,0)
H2_cir.cry(2*thetas[2],1,0)
H2_cir.barrier()
#applying unitaries as sum
H2_cir.x([0,1])
H2_cir.append(ZI,[0,1,2,3])
H2_cir.x(0)
H2_cir.append(IZ_,[0,1,2,3])
H2_cir.x([0,1])
H2_cir.append(XX_,[0,1,2,3])
H2_cir.x(0)
H2_cir.append(YY_,[0,1,2,3])
H2_cir.barrier()
#B dagger
H2_cir.cry(-2*thetas[2],1,0)
H2_cir.cx(1,0)
H2_cir.cry(-2*thetas[1],0,1)
H2_cir.ry(-2*thetas[0],0)
H2_cir.measure([0,1],[0,1])
H2_cir.draw('mpl')
```

Out[22]:



In [23]:

```
shots = 8192
E = []
probs = []
for i in range(20):
    result = execute(H2_cir, backend = qasm, shots = shots).result()
    count = result.get_counts(H2_cir)
    probs.append(count['00']/shots)
    E.append(-A*np.sqrt(count['00']/shots)+5.906709)
print('energy = ',E,'\n')
```

```
energy =  [-1.75426429553979, -1.6637058735103052, -1.7731458967629852, -1.824163499772
7019, -1.7479600785115101, -1.730822401619097, -1.7937726242396934, -1.690078952526995
7, -1.734433527779081, -1.7605633290898774, -1.7605633290898774, -1.7524636201604586, -
1.68553840585842, -1.751563123699741, -1.7560645477783714, -1.7506625213418676, -1.7100
252017953466, -1.749761813049469, -1.7290161981170469, -1.7839145698208068]
```

N = 4, GC, EFT

In [40]:

```
#try doing with sum of 5 terms in H add 6th term
#do with initialization of V with 5 terms and then add 6 terms
```

In [71]:

```
#(14.328/trotter_number * I ^ I) - \
H4_GC = -(7.814 * X ^ I) - \
        (3.913 * I ^ X) + \
        (3.913 * Z ^ X) - \
        (1.422 * Z ^ I) - \
        (8.422 * I ^ Z) + \
        (3.527 * X ^ Z) - \
        (4.922 * Z ^ Z)
e,v = np.linalg.eig(H4_GC.to_matrix())
v = np.transpose(v)
e
```

Out[71]:

```
array([-16.47339042+0.j,  -8.25620643+0.j,   3.42374582+0.j,
       21.30585103+0.j])
```

In [43]:

```
(X^Z).to_matrix()
```

Out[43]:

```
array([[ 0.+0.j,  0.+0.j,  1.+0.j,  0.+0.j],
       [ 0.+0.j,  0.+0.j,  0.+0.j, -1.+0.j],
       [ 1.+0.j,  0.+0.j,  0.+0.j,  0.+0.j],
       [ 0.+0.j, -1.+0.j,  0.+0.j,  0.+0.j]])
```

In [86]:

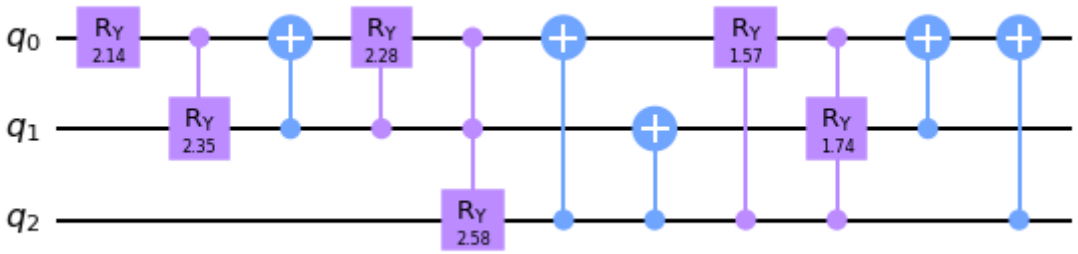
```
betas = [7.814,3.913,3.913,1.422,8.422,3.527,4.922]
thetas, A = theta_computer(betas)
```

In [87]:

```
ccry = RYGate(2*thetas[3]).control(2)
ccry2 = RYGate(2*thetas[5]).control(2)
```

In [88]:
```
#Qauntum Circuit for V_gate and V_dag
V_cir = QuantumCircuit(3)
V_cir.ry(2*thetas[0],0)
V_cir.cry(2*thetas[1],0,1)
V_cir.cx(1,0)
V_cir.cry(2*thetas[2],1,0)
V_cir.append(ccry,[0,1,2])
V_cir.cx(2,0)
V_cir.cx(2,1)
V_cir.cry(2*thetas[4],2,0)
V_cir.append(ccry2,[2,0,1])
V_cir.cx(1,0)
V_cir.cx(2,0)
V_dag = V_cir.inverse()
V_cir.draw('mpl')
```

Out[88]:



Why is this happening?

With just 6 terms at least the V is now in correct order

In [81]:
```
#to test correctness of V
shots = 20000
new = []
for beta in betas:
    new.append(beta/A)
display(new)
V_cir.measure_all()
result = execute(V_cir, backend = simulator, shots = shots).result()
count = result.get_counts(V_cir)
plot_histogram(count)
```

```
[0.23027731117201544,
 0.11531547461173489,
 0.11531547461173489,
 0.04190610909733887,
 0.24819497244570185,
 0.10394011728995374,
 0.14505054077152035]
```

Out[81]:

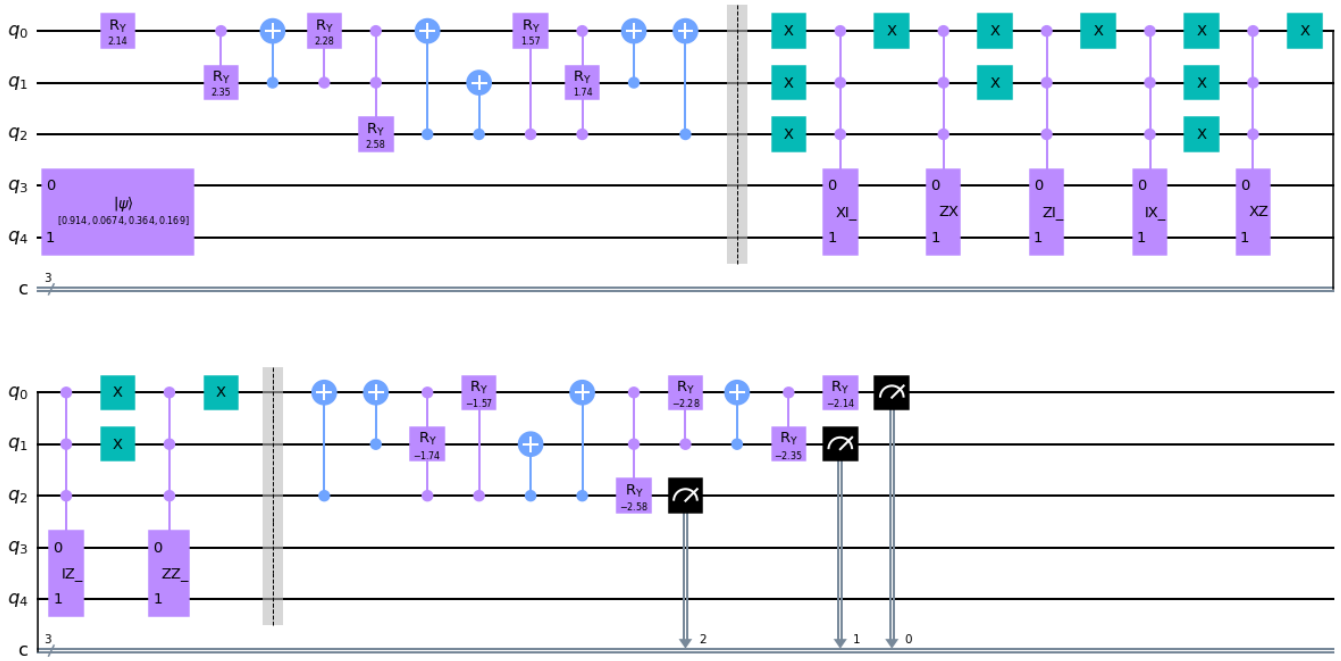
In [82]:
```
#controlled gates
xi = QuantumCircuit(2)
xi.x(1)
XI_ = qc(negate(xi)).to_gate(label='XI_').control(3)
ix = QuantumCircuit(2)
ix.x(0)
IX_ = qc(negate(ix)).to_gate(label='IX_').control(3)
zi = QuantumCircuit(2)
zi.z(1)
ZI_ = qc(negate(zi)).to_gate(label='ZI_').control(3)
iz = QuantumCircuit(2)
iz.z(0)
IZ_ = qc(negate(iz)).to_gate(label='IZ_').control(3)
xz = QuantumCircuit(2)
xz.x(1)
xz.z(0)
XZ = qc(negate(xz,neg = False)).to_gate(label='XZ').control(3)
zx = QuantumCircuit(2)
zx.z(1)
zx.x(0)
ZX = qc(negate(zx,neg = False)).to_gate(label='ZX').control(3)
zz = QuantumCircuit(2)
zz.z([0,1])
ZZ_ = qc(negate(zz)).to_gate(label='ZZ_').control(3)
```

In [48]:
```
negate(xz,neg = False)
```

Out[48]:
```
array([[ 0.+0.j,  0.+0.j,  1.+0.j,  0.+0.j],
       [ 0.+0.j,  0.+0.j,  0.+0.j, -1.+0.j],
       [ 1.+0.j,  0.+0.j,  0.+0.j,  0.+0.j],
       [ 0.+0.j, -1.+0.j,  0.+0.j,  0.+0.j]])
```

In [95]:
```
gc4 =QuantumCircuit(5,3)
#V gate
gc4.compose(V_cir,[0,1,2],inplace= True)
#state intialization
gc4.initialize(v[0],[3,4])
gc4.barrier()
#adder circuit
gc4.x([0,1,2])
gc4.append(XI_,[0,1,2,3,4])
gc4.x([0])
gc4.append(ZX,[0,1,2,3,4])
gc4.x([0,1])
gc4.append(ZI_,[0,1,2,3,4])
gc4.x(0)
gc4.append(IX_,[0,1,2,3,4])
gc4.x([0,1,2])
gc4.append(XZ,[0,1,2,3,4])
gc4.x(0)
gc4.append(IZ_,[0,1,2,3,4])
gc4.x([0,1])
gc4.append(ZZ_,[0,1,2,3,4])
gc4.x([0])
gc4.barrier()
gc4.compose(V_dag,[0,1,2],inplace = True)
gc4.measure([0,1,2],[0,1,2])
gc4.draw('mpl')
```

Out[95]:



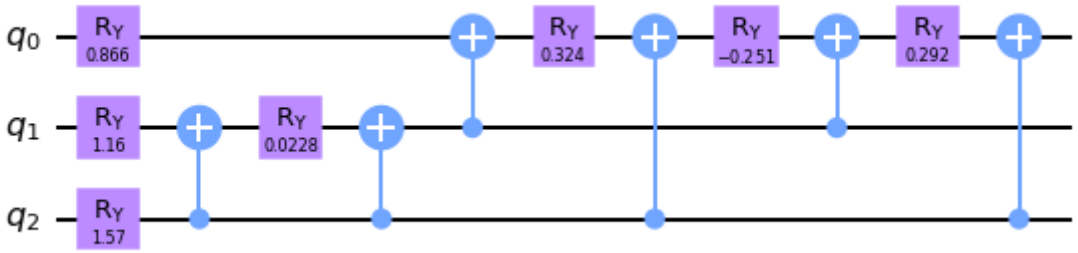
In [96]:

```
E = []
probs = []
shots = 5000
for i in range(1):
    result = execute(gc4, backend = simulator, shots = shots).result()
    count = result.get_counts(gc4)
    probs.append(count['000']/shots)
    E.append(-A*np.sqrt(count['000']/shots))
display(E,probs)
```

```
[-8.6512514577372]
[0.065]
```

In [63]:

```
# Trying with V initilalize
#trying all 7 terms in H with initialize V method
#V and V_dag
betas = [7.814,3.913,3.913,1.422,8.422,3.527,4.922,0]
A = sum(betas)
gammas = []
for beta in betas:
    gammas.append(np.sqrt(beta/A))
#for Vdag
Vdag =QuantumCircuit(3)
Vdag.initialize(gammas,[0,1,2])
transpilecir = transpile(Vdag,basis_gates = ['x','y','z','rx','ry','rz','cx'],optimizat
display(transpilecir.draw('mpl'))
V_dag = transpilecir.inverse()
```



In [67]:
```
gc4 =QuantumCircuit(5,3)
#V gate
gc4.compose(Vdag,[0,1,2],inplace= True)
#state intialization
gc4.initialize(v[0],[3,4])
gc4.barrier()
#adder circuit
gc4.x([0,1,2])
gc4.append(XI_,[0,1,2,3,4])
gc4.x([0])
gc4.append(IX_,[0,1,2,3,4])
gc4.x([0,1])
gc4.append(ZX,[0,1,2,3,4])
gc4.x(0)
gc4.append(ZI_,[0,1,2,3,4])
gc4.x([0,1,2])
gc4.append(IZ_,[0,1,2,3,4])
gc4.x(0)
gc4.append(XZ,[0,1,2,3,4])
gc4.x([0,1])
gc4.append(ZZ_,[0,1,2,3,4])
gc4.x([0])
gc4.barrier()
gc4.compose(V_dag,[0,1,2],inplace = True)
gc4.measure([0,1,2],[0,1,2])
```

Out[67]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x7f81c9884b50>
```

In [66]:
```
E = []
probs = []
shots = 5000
for i in range(1):
    result = execute(gc4, backend = simulator, shots = shots).result()
    count = result.get_counts(gc4)
    probs.append(count['000']/shots)
    E.append(-A*np.sqrt(count['000']/shots))
display(E,probs)
```

```
[-16.442630726778486]
[0.2348]
```

In [70]:
```
E[0]+14.328
```

Out[70]:
```
-2.1146307267784863
```

V by direct initialization method gave right results

How I obtained result for N = 4 last time?

Since H for N = 4 is : $$H_4 = H_3 + \cdots$$ I simply did LCU for rest terms and added eigenvalue for H_3 part in the end

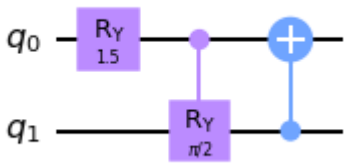
In [108…
```
#H_4 = H_3+ 13.125291I+.....
H_4 = (-13.125 * I^I^I^Z)-(5.671* I^I^X^X)-(5.671* I^I^Y^Y)
e,v = np.linalg.eig(H_4.to_matrix())
v = np.transpose(v)
e+13.12591-2.046
```

Out[108…
```
array([-6.2667493+0.j, 28.4265693+0.j, 28.4265693+0.j, -6.2667493+0.j,
       28.4265693+0.j, -6.2667493+0.j, 28.4265693+0.j, -6.2667493+0.j,
       -2.04509  +0.j, 24.20491  +0.j, -2.04509  +0.j, 24.20491  +0.j,
       -2.04509  +0.j, 24.20491  +0.j, -2.04509  +0.j, 24.20491  +0.j])
```

In [100…
```
betas = [13.125,5.671,5.671]
thetas, A =theta_computer(betas)
```

In [94]:
```
# V and V_dag gate
V_cir = QuantumCircuit(2)
V_cir.ry(2*thetas[0],0)
V_cir.cry(2*thetas[1],0,1)
V_cir.cx(1,0)
V_dag = V_cir.inverse()
V_cir.draw('mpl')
```

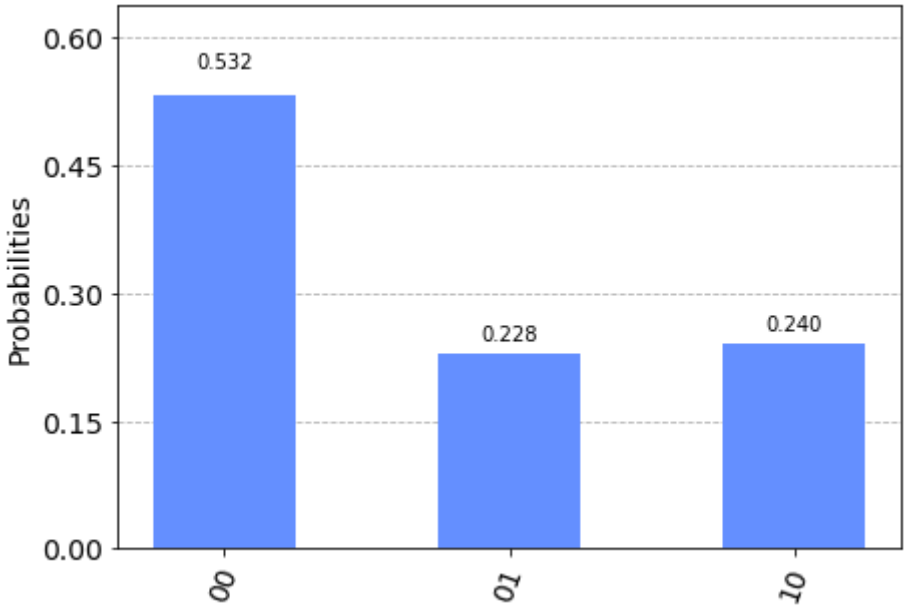
Out[94]:



In [91]:
```
#to test correctness of V
new = np.array(betas)/A
display(new)
V_cir.measure_all()
result = execute(V_cir, backend = simulator, shots = shots).result()
count = result.get_counts(V_cir)
plot_histogram(count)
```

```
array([0.53643683, 0.23178158, 0.23178158])
```

Out[91]:



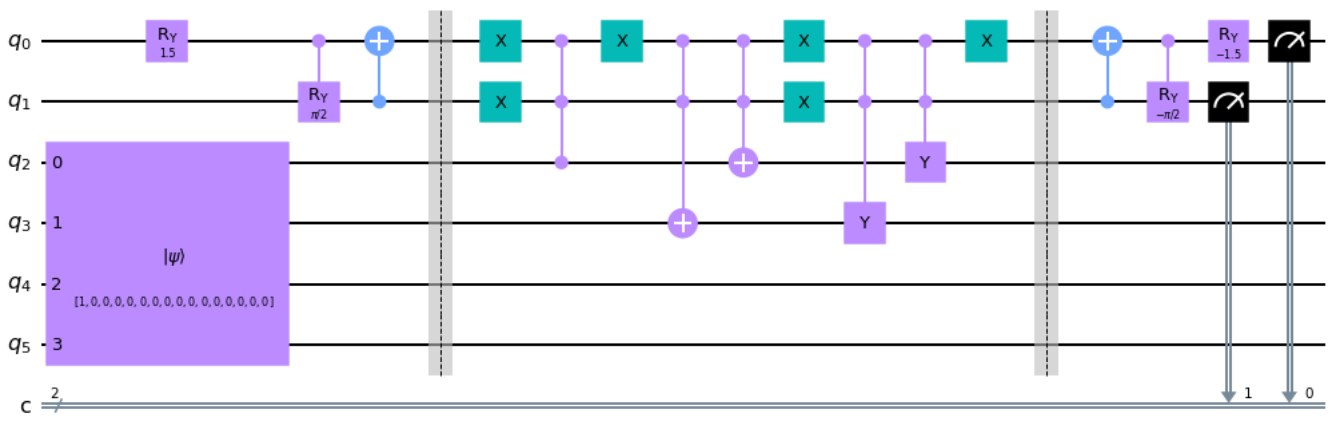
In [95]:
```
# defining gates
iiiz = QuantumCircuit(4)
iiiz.z(3)
IIIZ_ = qc(negate(iiiz)).to_gate(label = 'IIIZ_').control(2)
iixx = QuantumCircuit(4)
iixx.x([3,2])
IIXX_ = qc(negate(iixx)).to_gate(label = 'IIXX_').control(2)
iiyy = QuantumCircuit(4)
iiyy.y([3,2])
IIYY_ = qc(negate(iiyy)).to_gate(label = 'IIYY_').control(2)
```

In [103…
```
#defining gates by MCMT
ccz = MCMT('cz',2,1)
ccx = MCMT('cx',2,1)
ccy = MCMT('cy',2,1)
```

In [111…

```
H4s = QuantumCircuit(6,2)
#V gate
H4s.compose(V_cir,[0,1],inplace = True)
H4s.initialize(v[8],[2,3,4,5])
H4s.barrier()
H4s.x([0,1])
H4s.compose(ccz,[0,1,2],inplace = True)
H4s.x(0)
H4s.compose(ccx,[0,1,3],inplace = True)
H4s.compose(ccx,[0,1,2],inplace = True)
H4s.x([0,1])
H4s.compose(ccy,[0,1,3],inplace = True)
H4s.compose(ccy,[0,1,2],inplace = True)
H4s.x(0)
H4s.barrier()
H4s.compose(V_dag,[0,1],inplace = True)
H4s.measure([0,1],[0,1])
H4s.draw('mpl')
```

Out[111…



In [112…

```
shots = 8192
E = []
probs = []
for i in range(1):
    result = execute(H4s, backend = simulator, shots = shots).result()
    count = result.get_counts(H4s)
    probs.append(count['00']/shots)
    E.append(-A*np.sqrt(count['00']/shots))
print('energy = ',E,'\n')
print(E[0]+13.125-2.046)
```

```
energy =  [-13.101688165863674]

-2.022688165863674
```

N = 4, JWT

In [10]:

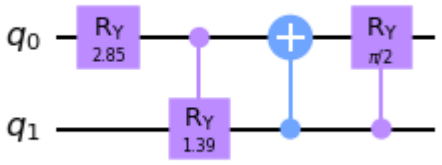
```
#(28.657* I ^ I ^ I^I) + \
H4_op = (0.218 * Z ^ I ^ I ^ I) - \
        (6.125 * I ^ Z ^ I^ I) - \
        (2.143 * X ^ X ^ I^I) - \
        (2.143 * Y ^ Y ^ I^I)# - \
        #(9.625 * I ^ I ^ Z^I) - \
        #(13.125 * I ^ I ^ I^Z) - \
        #(5.671 * I ^ I ^ X^X) - \
        #(5.671 * I ^I ^ Y^Y) - \
        #(3.913 * I ^ X ^ X^I) - \
        #(3.913 * I ^ Y ^ Y^I)
e,v = np.linalg.eig(H4_op.to_matrix())
v = np.transpose(v)
e
```

Out[10]:
```
array([-7.65528869+0.j,  7.65528869+0.j, -7.65528869+0.j,  7.65528869+0.j,
       -7.65528869+0.j,  7.65528869+0.j, -7.65528869+0.j,  7.65528869+0.j,
       -5.907     +0.j, -5.907     +0.j, -5.907     +0.j, -5.907     +0.j,
        5.907     +0.j,  5.907     +0.j,  5.907     +0.j,  5.907     +0.j])
```

In [11]:
```
betas = [0.218,6.125,2.143,2.143]
thetas, A = theta_computer(betas)
```

In [18]:
```
#Qauntum Circuit for V_gate and V_dag
V_cir = QuantumCircuit(2)
V_cir.ry(2*thetas[0],0)
V_cir.cry(2*thetas[1],0,1)
V_cir.cx(1,0)
V_cir.cry(2*thetas[2],1,0)
V_dag = V_cir.inverse()
V_cir.draw('mpl')
```

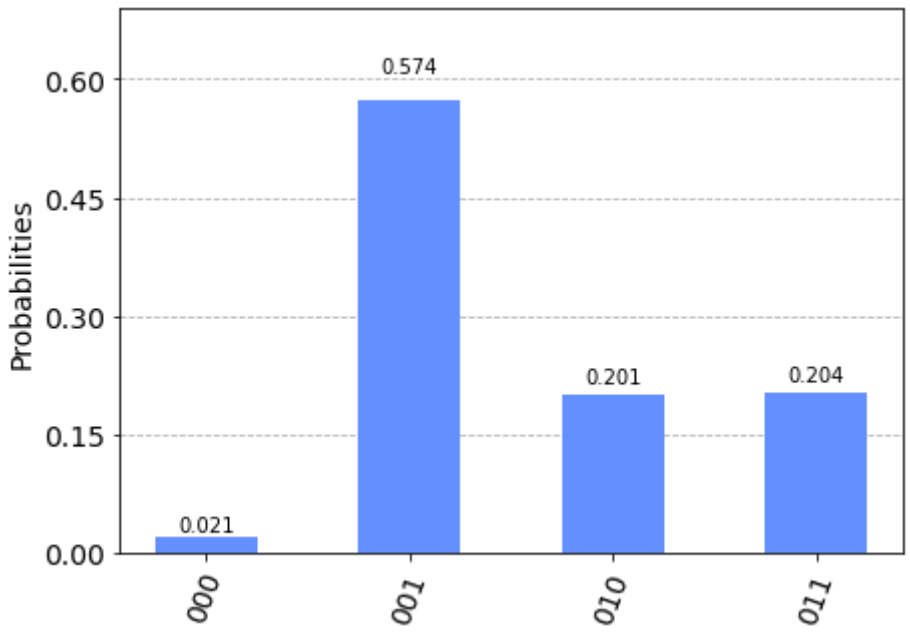
Out[18]:



In [14]:
```
#to test correctness of V
shots = 20000
new = []
for beta in betas:
    new.append(beta/A)
display(new)
V_cir.measure_all()
result = execute(V_cir, backend = simulator, shots = shots).result()
count = result.get_counts(V_cir)
plot_histogram(count)
```

```
[0.020509925675039984,
 0.5762536456863298,
 0.20161821431931504,
 0.20161821431931504]
```

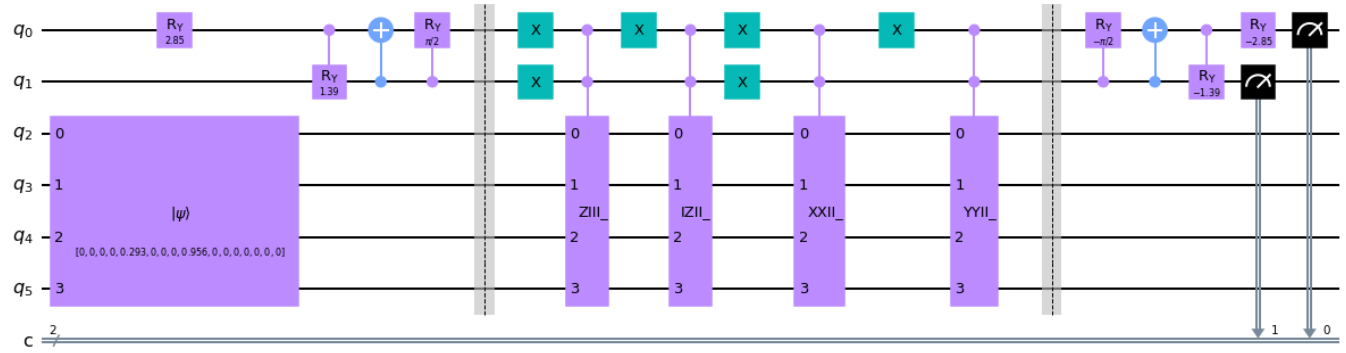
Out[14]:



In [15]:
```
#controlled gates
xxii = QuantumCircuit(4)
xxii.x([2,3])
XXII_ = qc(negate(xxii)).to_gate(label='XXII_').control(2)
ziii = QuantumCircuit(4)
ziii.z(3)
ZIII = qc(negate(ziii,neg = False)).to_gate(label='ZIII_').control(2)
izii = QuantumCircuit(4)
izii.z([2])
IZII_ = qc(negate(izii)).to_gate(label='IZII_').control(2)
yyii = QuantumCircuit(4)
yyii.y([2,3])
YYII_ = qc(negate(yyii)).to_gate(label='YYII_').control(2)
```

In [19]:
```
jwt4 =QuantumCircuit(6,2)
#V gate
jwt4.compose(V_cir,[0,1],inplace= True)
#state intialization
jwt4.initialize(v[0],[2,3,4,5])
jwt4.barrier()
#adder circuit
jwt4.x([0,1])
jwt4.append(ZIII,[0,1,2,3,4,5])
jwt4.x([0])
jwt4.append(IZII_,[0,1,2,3,4,5])
jwt4.x([0,1])
jwt4.append(XXII_,[0,1,2,3,4,5])
jwt4.x(0)
jwt4.append(YYII_,[0,1,2,3,4,5])
jwt4.barrier()
jwt4.compose(V_dag,[0,1],inplace = True)
jwt4.measure([0,1],[0,1])
jwt4.draw('mpl')
```

Out[19]:



In [25]:
```
E = []
probs = []
shots = 8192
for i in range(20):
    result = execute(jwt4, backend = qasm, shots = shots).result()
    count = result.get_counts(jwt4)
    probs.append(count['00']/shots)
    E.append(-A*np.sqrt(count['00']/shots))
display(E)
```

```
[-6.664887108831021,
 -6.609825918128916,
 -6.6327369952084965,
 -6.618166395791422,
 -6.647275656449945,
 -6.810215721202343,
 -6.67109181500843,
 -6.5941590889925,
 -6.78891948409131,
 -6.642086930613243,
 -6.758379880372358,
```