

# 日本語訳 『Qiskit Textbook』 勉強会 第1章 1.1- 1.2節



---

Yuri Kobayashi

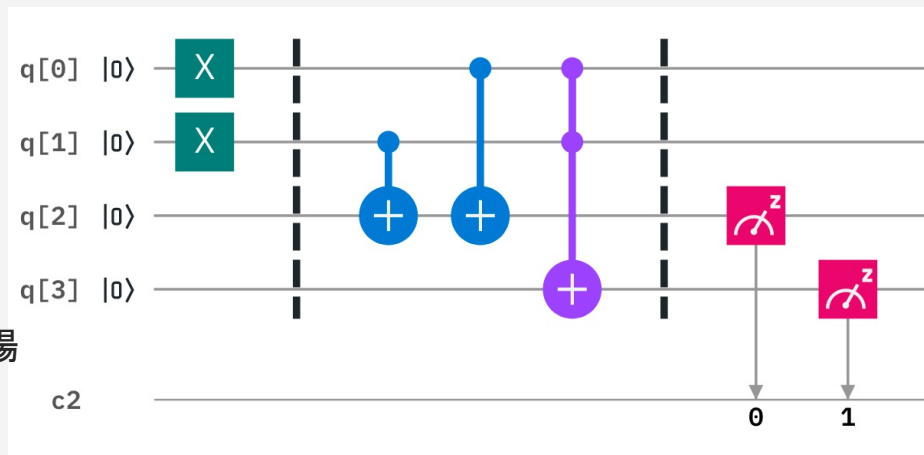
Quantum Developer Community

# 1.1節と1.2節について

- 1.3節と1.4節に比べてボリュームが格段に少ないです。
- メインコンテンツ：
  - 古典コンピューターの計算は量子ゲートで再現できる
  - 量子回路作成のステップ
  - 回路作成の演習「半加算器のつくり方」

# 1.1 はじめに

- ボールやバナナ vs 量子
- 「ビット」とは？ 英語のbitが複数の意味を持っていることを知らない面白くない話
  - 《工具》ビット ドリルの先端に取り付ける交換式の刃
  - 《馬具》はみ
  - 《情報量の最小単位》0または1
- 量子ビットの世界 異なるルールに従う
- 量子回路とブロッホ球の図
- 量子回路の説明はここではないが1.2に改めて登場



1.2

# 計算の原子

## 2進数：2を底（基準）として数を表す方法

 $10^3 \quad 10^2 \quad 10^1 \quad 10^0$ 

|   |   |   |   |
|---|---|---|---|
| 9 | 2 | 1 | 3 |
|---|---|---|---|

$$+(\mathbf{1}\times 2^6)+(\mathbf{1}\times 2^5)+(\mathbf{1}\times 2^4)+(\mathbf{1}\times 2^3)+(\mathbf{1}\times 2^2)+(\mathbf{0}\times 2^1)+(\mathbf{1}\times 2^0)$$
$$2^{13} \quad 2^{12} \quad 2^{11} \quad 2^{10} \quad 2^9 \quad 2^8 \quad 2^7 \quad 2^6 \quad 2^5 \quad 2^4 \quad 2^3 \quad 2^2 \quad 2^1 \quad 2^0$$
[illegible]

# 練習問題

練習問題：

1. 好きな数字を考えて、2進法で書いてみてください。
2.  $n$ 個のビットがあるとしたら、それは何種類の状態になるでしょうか？

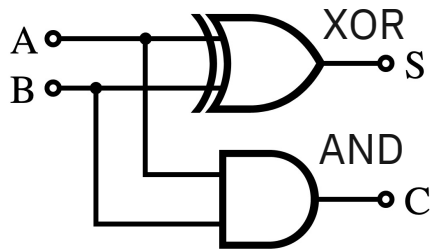


Localに落としたQiskit Textbookでウィジットを使うには

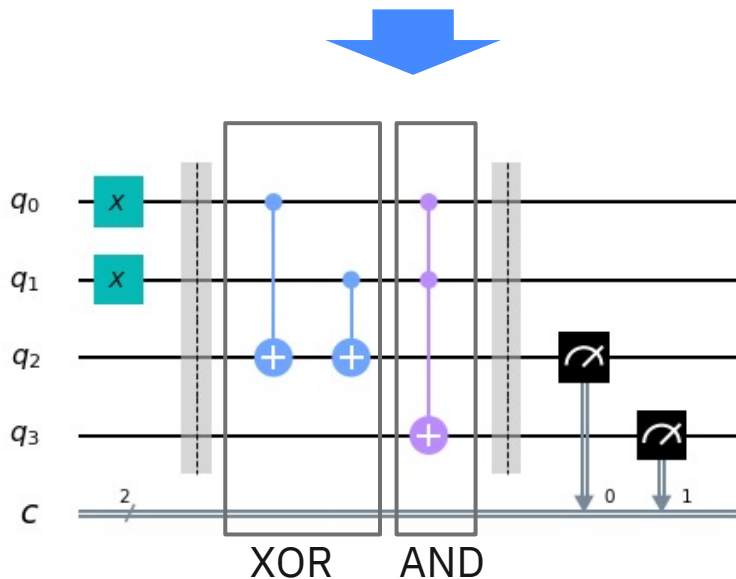
自身のqiskit-textbookのディレクトリー上で `pip install -e ./qiskit-textbook-src` を実行すると qiskit-textbookがインストールされます。

# ダイアグラムを使った計算

古典の論理ゲート



加算回路



YES



| INPUT |  | OUTPUT |
|-------|--|--------|
| A     |  |        |
| 0     |  | 0      |
| 1     |  | 1      |

NOT



| INPUT |  | OUTPUT |
|-------|--|--------|
| A     |  |        |
| 0     |  | 1      |
| 1     |  | 0      |

AND



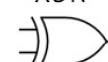
| INPUT |   | OUTPUT |
|-------|---|--------|
| A     | B |        |
| 0     | 0 | 0      |
| 1     | 0 | 0      |
| 0     | 1 | 0      |
| 1     | 1 | 1      |

OR



| INPUT |   | OUTPUT |
|-------|---|--------|
| A     | B |        |
| 0     | 0 | 0      |
| 1     | 0 | 1      |
| 0     | 1 | 1      |
| 1     | 1 | 1      |

XOR



| INPUT |   | OUTPUT |
|-------|---|--------|
| A     | B |        |
| 0     | 0 | 0      |
| 1     | 0 | 1      |
| 0     | 1 | 1      |
| 1     | 1 | 0      |

NAND



| INPUT |   | OUTPUT |
|-------|---|--------|
| A     | B |        |
| 0     | 0 | 1      |
| 1     | 0 | 1      |
| 0     | 1 | 1      |
| 1     | 1 | 0      |

NOR



| INPUT |   | OUTPUT |
|-------|---|--------|
| A     | B |        |
| 0     | 0 | 1      |
| 1     | 0 | 0      |
| 0     | 1 | 0      |
| 1     | 1 | 0      |

XNOR

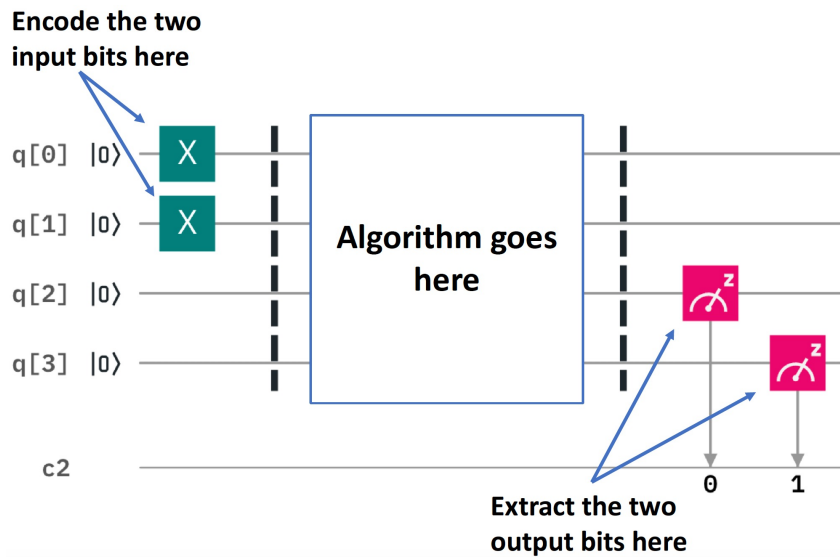


| INPUT |   | OUTPUT |
|-------|---|--------|
| A     | B |        |
| 0     | 0 | 1      |
| 1     | 0 | 0      |
| 0     | 1 | 0      |
| 1     | 1 | 1      |

# はじめての量子回路Ⅱ

量子回路を作成するには3つのステップが必要です。

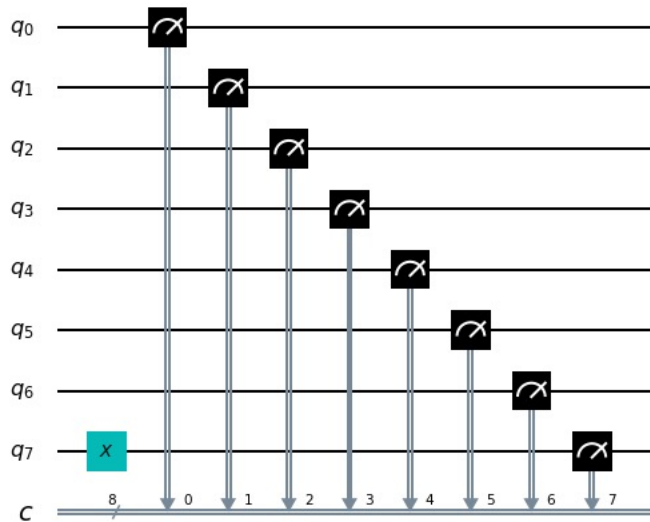
1. 最初に入力をエンコードする
2. 実際の計算を行う (量子ビットに何かしらの操作をする)
3. 最後に出力を抽出する (測定)





# はじめての量子回路Ⅱ

次の回路をQiskitで組んでみましょう。



```
qc = QuantumCircuit(8,8)
qc.x(7)
for i in range(8):
    qc.measure(i,i)
qc.draw()
```



# はじめての量子回路Ⅱ



## 教科書の中のコード

```
n = 8 #量子ビットの数を定義
n_q = n #量子レジスタを定義
n_b = n #出力結果を保存するための古典レジスタを定義

qc_output = QuantumCircuit(n_q,n_b)    #'qc_output'という名の出力用回路を作成

for j in range(n):    #'qc_output'に測定を追加し、量子ビット`j`の出力を古典レジスタ`j`に書き込むよう指示
    qc_output.measure(j,j)

qc_encode = QuantumCircuit(n)    #'qc_encode'という名の入力用回路を作成
qc_encode.x(7)    #8番目の量子ビットにxゲートを置く

qc = qc_encode + qc_output #入力用と出力用回路を連結した最終的な回路`qc`を定義

qc.draw(output='mpl',justify='none')    #'qc'回路の描画

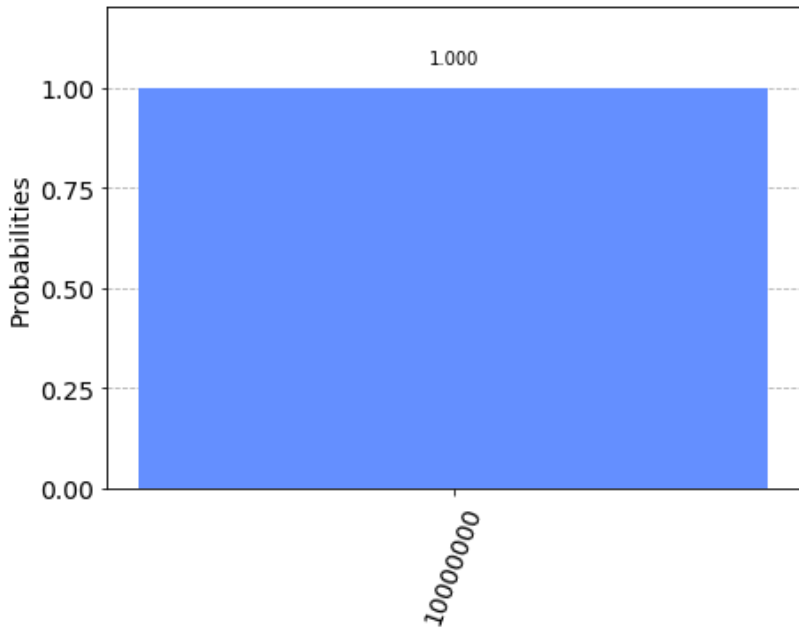
counts = execute(qc,Aer.get_backend('qasm_simulator')).result().get_counts()
plot_histogram(counts)    #'qc'回路のシミュレーターによる測定結果をカウントしてヒストグラムにプロット
```

# はじめての量子回路II



## 出力について

```
counts = execute(qc_output, Aer.get_backend('qasm_simulator')).result().get_counts()  
plot_histogram(counts)
```



q7(8番目の量子ビット) にXゲートが適用されて該当ビットが0 → 1に反転しています。

※Qiskitでは文字列のビットに右から左に順番に番号をつけます。一般の教科書とは逆のため注意が必要。


# 練習問題

好きな数字をエンコードしてみましょう。

例. **108** という数字をエンコードします。108のバイナリ表示を検索。

108 = 0b1101100 （「0b」が含まれていたら無視.ここでは0に置き換え）→ 01101100

先ほどの回路に量子ゲートを適用してエンコードみましょう。

 108という数字のエンコードの仕方（例）

```
qc_encode = QuantumCircuit(n)
qc_encode.x(2)
qc_encode.x(3)
qc_encode.x(5)
qc_encode.x(6)
qc = qc_encode + qc_output
counts = execute(qc,Aer.get_backend('qasm_simulator')).result().get_counts()
plot_histogram(counts)
```

# 練習問題

好きな数字をエンコードしてみましょう。

例. 108 という数字をエンコードします。バイナリ表示は何かを検索。

108 = 0b1101100 （「0b」が含まれていたら無視.ここでは0に置き換え）→ 01101100

先ほどの回路に量子ゲートを適用してエンコードみましょう。



108という数字のエンコードの仕方（例）

```
qc_encode = QuantumCircuit(n)
qc_encode.x(2)
qc_encode.x(3)
qc_encode.x(5)
qc_encode.x(6)
qc = qc_encode + qc_output
counts = execute(qc, Aer.get_backend('qasm_simulator')).result().get_counts()
plot_histogram(counts)
```

# ビットの足し算について(半加算器のつくり方)

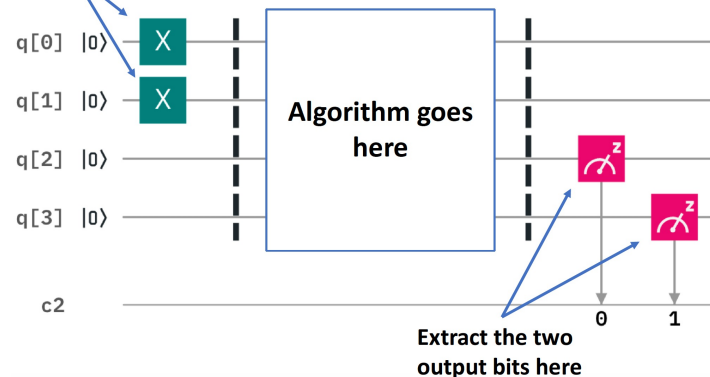
$$\begin{array}{r} 100011111111101 \\ + \quad 00011100111110 \\ \hline = 10101100111011 \end{array}$$



0+0 = 00 (十進数の場合は 0+0=0)  
0+1 = 01 (十進数の場合は 0+1=1)  
1+0 = 01 (十進数の場合は 1+0=1)  
1+1 = 10 (十進数の場合は 1+1=2)



Encode the two  
input bits here



q[0]とq[1]は入力用  
q[2]とq[3]は出力用

# 半加算器の作り方

どのようなゲートが必要かを考えるためにもう一度半加算器にさせたいことをみてみましょう。  
半加算器にさせたいこと

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

2つのビットが同じときは「0」

2つのビットが異なるときは「1」

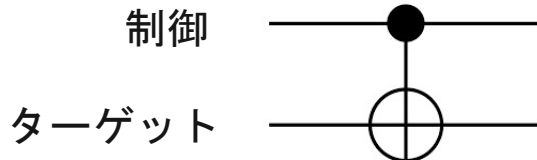
→ 足し合わせる2つのビットが同じか異なるかを判断できる論理ゲートがあればよい → XORゲート

→ 量子論理ゲートではCNOT Qiskitでは `cx` と書く



| INPUT |   | OUTPUT |
|-------|---|--------|
| A     | B |        |
| 0     | 0 | 0      |
| 1     | 0 | 1      |
| 0     | 1 | 1      |
| 1     | 1 | 0      |

CNOT



・ 2量子ゲート

・ 制御（入力）とターゲット（出力）

# 半加算器の作り方

```
qc_ha = QuantumCircuit(4,2)
```

```
# 量子ビット0と1に入力値をエンコードする
```

```
qc_ha.x(0) # 入力値 a=0のときは除く.入力値 a=1のときは残す
```

```
qc_ha.x(1) # 入力値 b=0のときは除く.入力値 b=1のときは残す
```

```
qc_ha.barrier()
```

```
# cnotをつかって入力値のXORした結果を量子ビット2に書き込む
```

```
qc_ha.cx(0,2)
```

```
qc_ha.cx(1,2)
```

```
qc_ha.barrier()
```

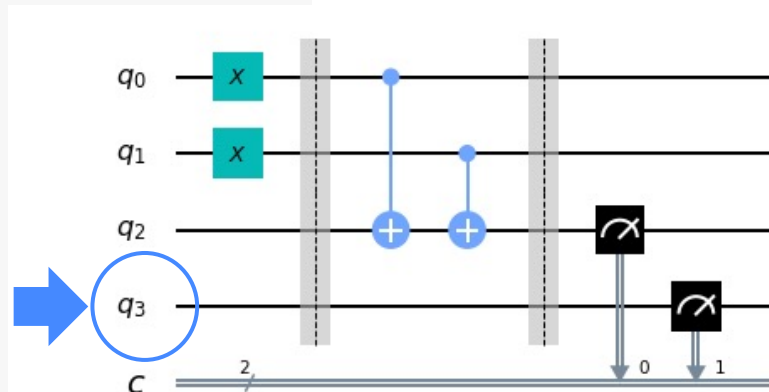
```
# 結果を抽出する (測定)
```

```
qc_ha.measure(2,0) # XORした結果を抽出
```

```
qc_ha.measure(3,1)
```

```
qc_ha.draw()
```

最後のビットがまだ  
残っています





# 半加算器の作り方

残された最後の量子ビットq3にどのようなゲートが必要かを考えましょう。

半加算器にさせたいこと

$$0+0 = 0\ 0$$

$$0+1 = 0\ 1$$

$$1+0 = 0\ 1$$

$$1+1 = \mathbf{1}\ 0$$

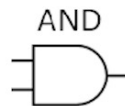
4つの組合せのうち唯一「1」になるのは  $1 + 1$  の時のみ  
上記以外は「0」

→ 足し合わせる2つのビットが両方とも「1」かを判断  
できればよい

→ 論理ゲートのANDゲート

→ 量子ゲートのトフォリ(Toffoli)ゲートに相当

→ Qiskitでは `ccx` と書く



| INPUT |   | OUTPUT |
|-------|---|--------|
| A     | B |        |
| 0     | 0 | 0      |
| 1     | 0 | 0      |
| 0     | 1 | 0      |
| 1     | 1 | 1      |

# 半加算器の作り方

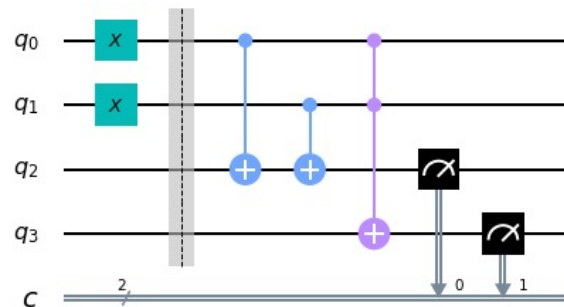
```
qc_ha = QuantumCircuit(4,2)
|
# 量子ビット0と1に入力値をエンコードする
qc_ha.x(0) # 入力値 a=0のときは除く.入力値 a=1のときは残す
qc_ha.x(1) # 入力値 b=0のときは除く.入力値 b=1のときは残す
qc_ha.barrier()

# cnotをつかって入力値のXORした結果を量子ビット2に書き込む
qc_ha.cx(0,2)
qc_ha.cx(1,2)

# ccxをつかってふたつの入力値のANDをとった結果を量子ビット3に書き込む
qc_ha.ccx(0,1,3)

# 結果を抽出する (測定)
qc_ha.measure(2,0) # XORした結果を抽出
qc_ha.measure(3,1)

qc_ha.draw()
```

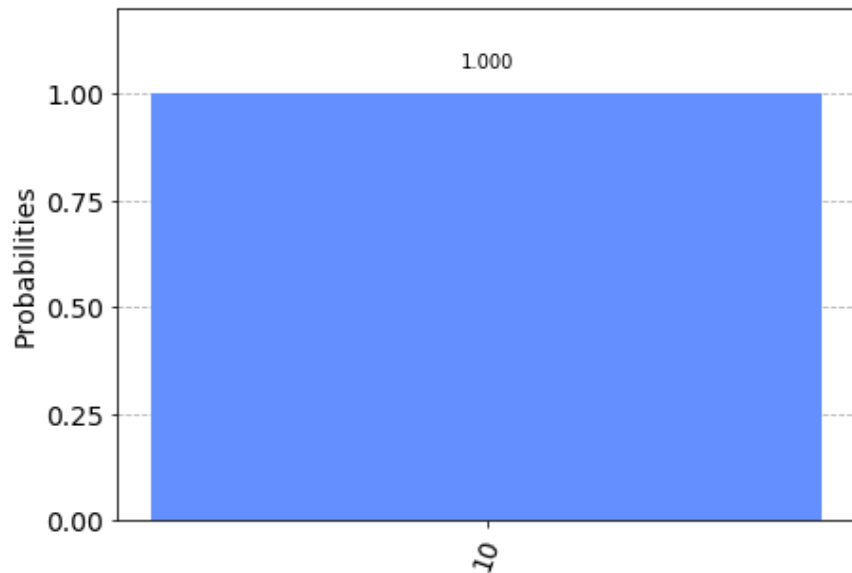


# 1 + 1の結果をみてみましょう



出力について

```
counts = execute(qc_output,Aer.get_backend('qasm_simulator')).result().get_counts()  
plot_histogram(counts)
```



$$1 + 1 = 2$$

## 1.1と1.2章のまとめ

- 古典コンピューターの計算は量子ゲートで再現できる
- 量子回路の作成は以下の3ステップ
  1. 入力をエンコードする（入力）
  2. 量子ビットを操作する（計算）
  3. 最後に出力を抽出する（測定）
- ここまでのところで「量子的」な要素はゼロです。笑

# おまけ

## 全加算器について

下の位からの桁上げを入力として含む「**全加算器**」の作り方を知りたい人は昨年度のIBM Quantum Challengeの**Week 1の演習**をおこなってみてください

## IBM Quantum Challenge 2019

<https://github.com/quantum-challenge/2019>

今年のIBM Quantum Challenge (5/8に終了) にもよかったら挑戦してみてください。

[https://github.com/qiskit-community/may4\\_challenge\\_exercises](https://github.com/qiskit-community/may4_challenge_exercises)

# Thank you

Yuri Kobayashi  
Quantum Developer Community  
yurik@jp.ibm.com

© Copyright IBM Corporation 2020. All rights reserved. The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. Any statement of direction represents IBM's current intent, is subject to change or withdrawal, and represent only goals and objectives. IBM, the IBM logo, and ibm.com are trademarks of IBM Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available at [Copyright and trademark information](#).