



Ch3: Quantum Instruction Sets & Quantum Circuits



Quantum bit

A qubit state is a unit vector in a two-dimensional complex vector space – Hilbert space:

- Superposition: $|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$
- Probability amplitudes: $\alpha, \beta \in \mathbb{C}$
- Normalization: $|\alpha|^2 + |\beta|^2 = 1$



Computational basis

The special states $|0\rangle$ and $|1\rangle$ are called computational basis states:

- Classical bit 0: $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$
- Classical bit 1: $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$



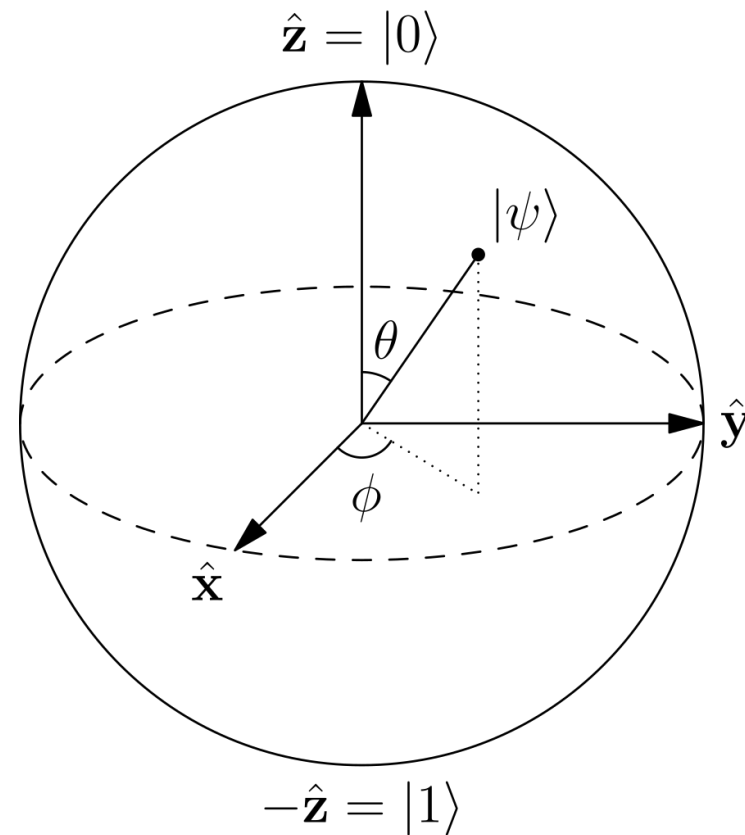
Bloch sphere representation

A qubit state can be seen as a vector pointing at a sphere surface ($\mathbb{C}^2 \mapsto \mathbb{R}^3$):

$$|\psi\rangle = \cos \frac{\theta}{2} |0\rangle + e^{i\phi} \sin \frac{\theta}{2} |1\rangle$$

- Amplitude parameter: $0 \leq \theta \leq \pi$
- Phase parameter: $0 \leq \phi \leq 2\pi$

- Bloch vector coordinates:
$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \sin \theta \cos \phi \\ \sin \theta \sin \phi \\ \cos \theta \end{pmatrix}$$





Unitary evolution

A general unitary transformation must be able to take the computational basis state $|0\rangle$ to any desired state $|\psi\rangle = \cos \frac{\theta}{2} |0\rangle + e^{i\phi} \sin \frac{\theta}{2} |1\rangle$. That is:

$$U = \begin{pmatrix} \cos(\theta / 2) & a \\ e^{i\phi} \sin(\theta / 2) & b \end{pmatrix}$$

where a and b are complex numbers constrained such that $UU^\dagger = I$. This gives:

$$U = \begin{pmatrix} \cos(\theta / 2) & -e^{i\lambda} \sin(\theta / 2) \\ e^{i\phi} \sin(\theta / 2) & e^{i\lambda + i\phi} \cos(\theta / 2) \end{pmatrix}.$$

where $0 \leq \lambda < 2\pi$. This is the most general form of a single qubit unitary.



Physical gates

All single-qubit operations are compiled down to gates known as $u1$, $u2$ and $u3$ before running on real quantum hardware. For that reason they are sometimes called the physical gates:

- $u3$ -gate:
$$u3(\theta, \phi, \lambda) = \begin{pmatrix} \cos \frac{\theta}{2} & -e^{i\lambda} \sin \frac{\theta}{2} \\ e^{i\phi} \sin \frac{\theta}{2} & e^{i(\lambda+\phi)} \cos \frac{\theta}{2} \end{pmatrix}$$
- $u2$ -gate:
$$u2(\phi, \lambda) = u3\left(\frac{\pi}{2}, \phi, \lambda\right) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -e^{i\lambda} \\ e^{i\phi} & e^{i(\lambda+\phi)} \end{pmatrix}$$
- $u1$ -gate:
$$u1(\lambda) = u3(0, 0, \lambda) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{pmatrix}$$



Pauli operators

The simplest quantum gates are the Pauli operators: X, Y and Z. Their action is to perform a half rotation of the Bloch sphere around the x, y and z axes.

- X-gate: $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = u3(\pi, 0, \pi)$

$$[X, Y] = XY - YX = 2iZ$$

$$[Y, Z] = 2iX$$

$$[Z, X] = 2iY$$

- Y-gate: $Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} = u3(\pi, \frac{\pi}{2}, \frac{\pi}{2})$

$$\{X, Y\} = XY + YX = 0$$

$$\{Y, Z\} = 0$$

$$\{Z, X\} = 0$$

- Z-gate: $Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = u1(\pi)$

$$X^2 = Y^2 = Z^2 = I$$



Rotation operators

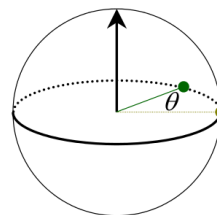
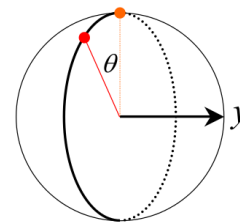
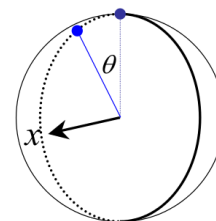
The Pauli operators give rise to three useful classes of unitary operators when they are exponentiated, the rotations operators about the x, y and z axes of the Bloch sphere:

$$R_P(\theta) = e^{-i\theta P} = \cos \frac{\theta}{2} I - i \sin \frac{\theta}{2} P$$

- R_x -gate:
$$R_X(\theta) = \begin{pmatrix} \cos \frac{\theta}{2} & -i \sin \frac{\theta}{2} \\ -i \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{pmatrix} = u3(\theta, -\frac{\pi}{2}, \frac{\pi}{2})$$

- R_y -gate:
$$R_Y(\theta) = \begin{pmatrix} \cos \frac{\theta}{2} & -\sin \frac{\theta}{2} \\ \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{pmatrix} = u3(\theta, 0, 0)$$

- R_z -gate:
$$R_Z(\theta) = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix} = u1(\theta)$$





Clifford gates

The Clifford gates are the elements of the Clifford group, a set of mathematical transformations which effect permutations of the Pauli operators:

- H-gate: $H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = u2(0, \frac{\pi}{2})$

$$\begin{aligned} HXH &= Z \\ HYH &= -Y \\ HZH &= X \end{aligned}$$

- S-gate: $S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} = u1(\frac{\pi}{2})$

$$\begin{aligned} SXS^\dagger &= iY \\ SYS^\dagger &= iX \\ SZS^\dagger &= Z \end{aligned}$$

- T-gate: $T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix} = u1(\frac{\pi}{4})$



Computational basis

To describe the state of a two-qubit system, by convention, we use the computational basis states $\{00, 10, 01, 11\}$:

$$|00\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ 0 \begin{pmatrix} 1 \\ 0 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$|01\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ 0 \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

$$|10\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ 1 \begin{pmatrix} 1 \\ 0 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$|11\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ 1 \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Product states

Consider two non-interacting systems A and B, with respective Hilbert spaces \mathcal{H}_A and \mathcal{H}_B . The Hilbert space of the composite system is the tensor product: $\mathcal{H}_A \otimes \mathcal{H}_B$.

If the first system is in state $|\psi\rangle_A$ and the second in state $|\phi\rangle_B$, the state of the composite system is: $|\psi\rangle_A \otimes |\phi\rangle_B$.

States of the composite system that can be represented in this form are called separable states, or product states. An example of separable states is given by:

$$\frac{|00\rangle + |10\rangle + |01\rangle + |11\rangle}{2}$$

where $|\psi\rangle_A = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$ and $|\phi\rangle_B = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$.



Entangled states

Not all states are separable states (and thus product states). If a state is inseparable, it is called an entangled state. See for instance the so-called Bell states:

$$\begin{cases} |\phi_{\pm}\rangle_{AB} = \frac{|00\rangle \pm |11\rangle}{\sqrt{2}} \\ |\psi_{\pm}\rangle_{AB} = \frac{|10\rangle \pm |01\rangle}{\sqrt{2}} \end{cases}$$

Such states are examples of maximally entangled states.

In a quantum computer, entanglement leads to correlations between the qubit systems, even though they are spatially separated.



Controlled operations

Most of the two-qubit gates are of the controlled type (the SWAP gate being the exception). In general, a controlled two-qubit gate C_U acts to apply the single-qubit unitary U to the second qubit when the state of the first qubit is in $|1\rangle$. Suppose U has a matrix representation:

$$U = \begin{pmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{pmatrix}$$

Then the action of C_U is in matrix form:

$$C_U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & u_{00} & u_{01} \\ 0 & 0 & u_{10} & u_{11} \end{pmatrix}$$

Controlled-NOT gate

A critical operator for quantum computing is the controlled NOT gate. We use it to entangle two qubits.

- CNOT-gate: $CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$



SWAP gate

The SWAP gate exchanges the states of the two qubits.

- SWAP-gate:
$$SWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Universal gate set

Any unitary can be decomposed using:

- Arbitrary single qubit gates
- The 2-qubit CNOT gate

Problem: it is not realistic to be able to perform arbitrary single-qubit gates with infinite precision. We would like a finite gate set.

Kitaev-Solovay theorem

The following sets allow to approximate any unitary arbitrary well:

- CNOT, Hadamard, T-gate
- Hadamard and Toffoli (3-qubit CCNOT gate) if the unitary have only real entries

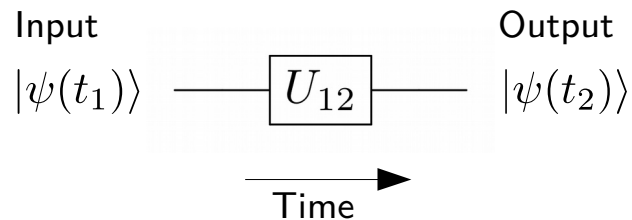
Kitaev-Solovay: any 1 or 2-qubit unitary can be approximated up to an error ϵ using $\text{polylog}(1/\epsilon)$ gates from the set.



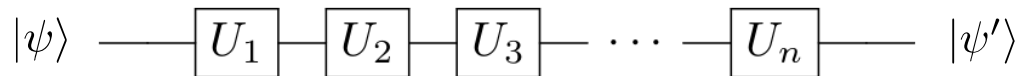
Quantum registers

Circuit diagrams are to be read from left to right. Each qubit is represented by a single horizontal wire and most gates are represented by boxes:

- A quantum register of one qubit with a single single qubit operation:



- Successive implementation:



$$|\psi'\rangle = U_n \cdots U_3 U_2 U_1 |\psi\rangle$$

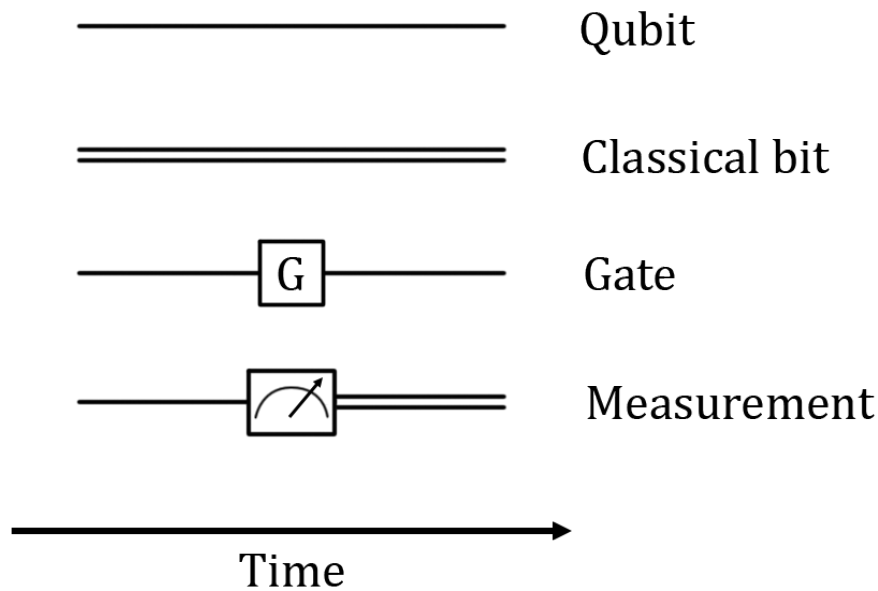
Time



3.6 Quantum circuit diagrams

Classical registers











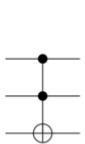
We also need classical registers, represented by doubled horizontal wires, to get the measurement result after completion of a computation.

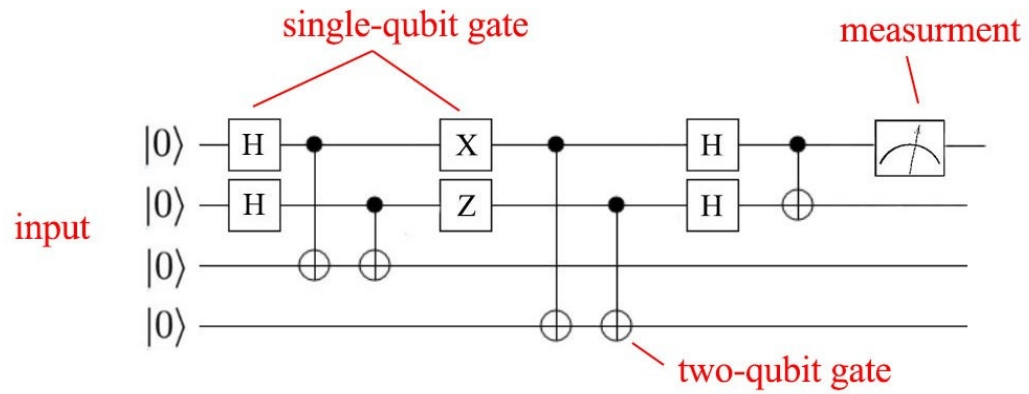
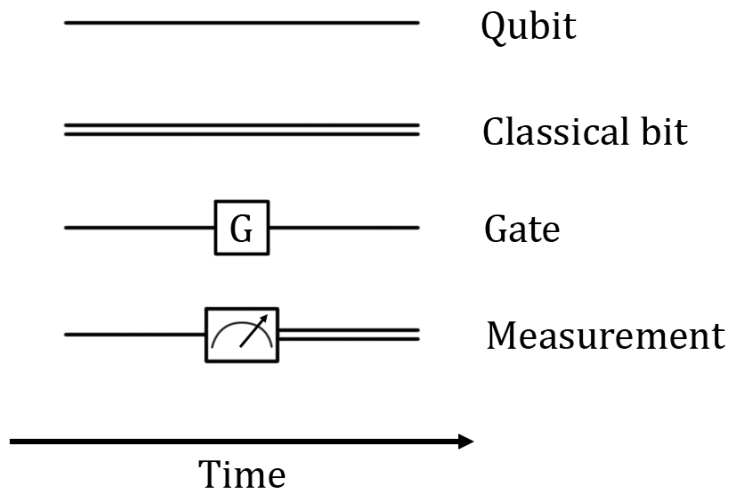




3.6 Quantum circuit diagrams

Building blocks

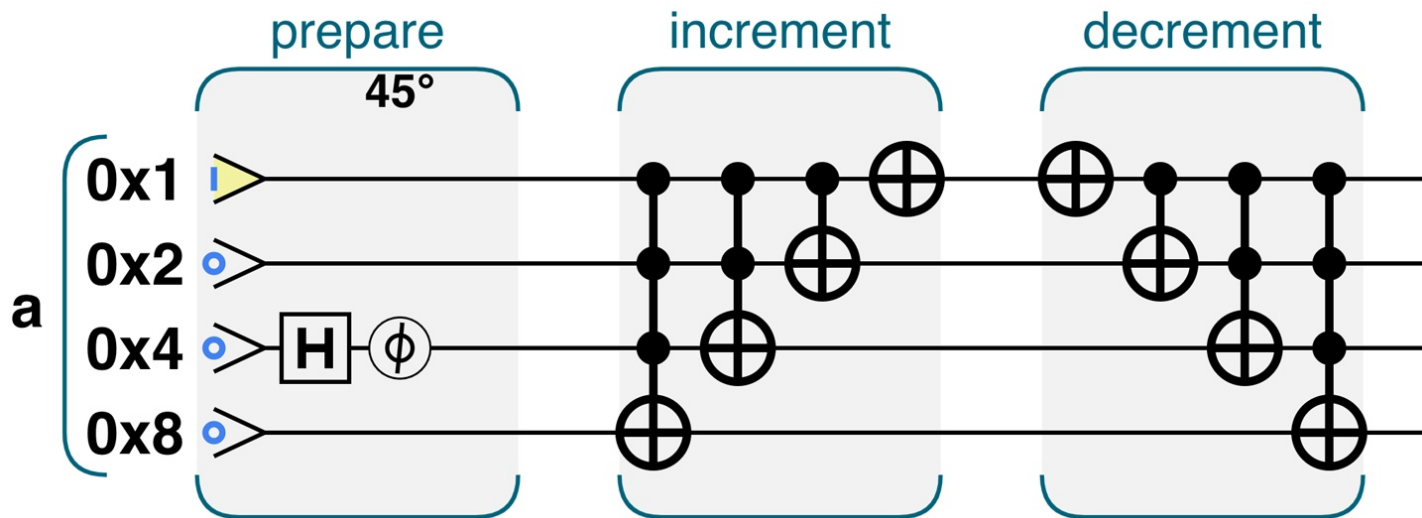
Operator	Gate(s)	Matrix
Pauli-X (X)	 	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y (Y)		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z (Z)		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard (H)		$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Phase (S, P)		$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
$\pi/8$ (T)		$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$
Controlled Not (CNOT, CX)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
Controlled Z (CZ)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$
SWAP		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Toffoli (CCNOT, CCX, TOFF)		$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$





Building increment and decrement operators

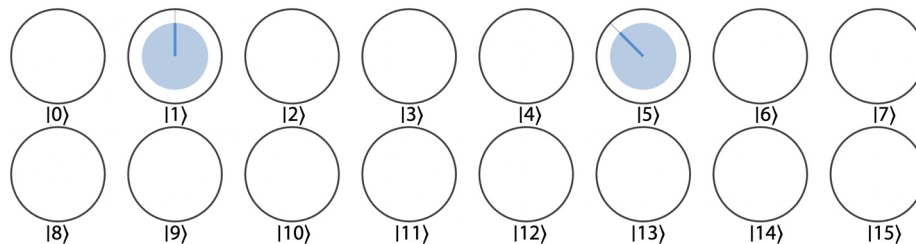
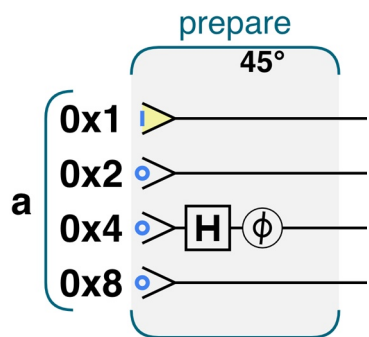
Two of the simplest useful integer arithmetic operations we can perform are those for incrementing and decrementing a number. Operations performing increment-by-1 and decrement-by-1:





Building increment and decrement operators

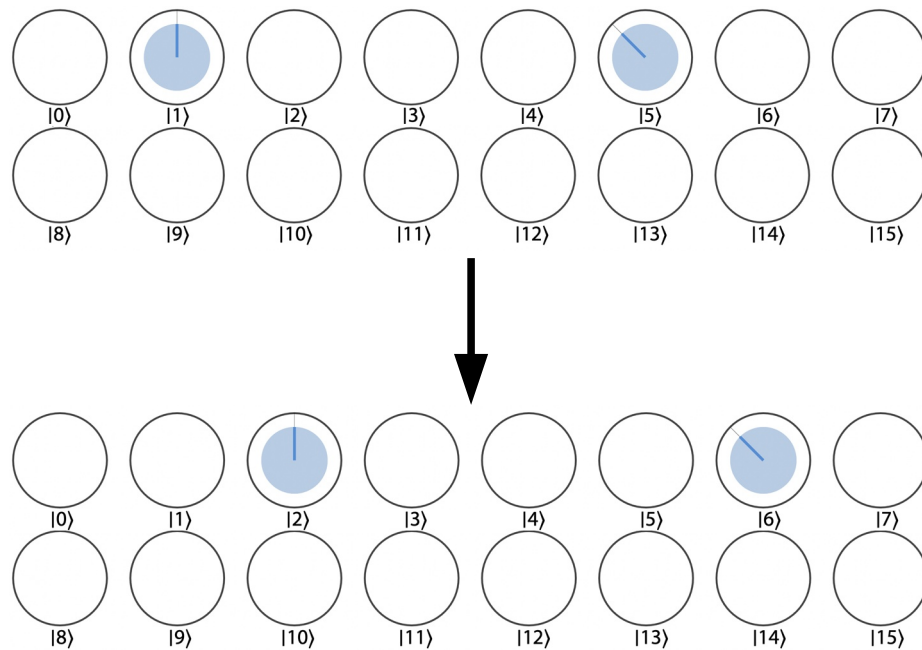
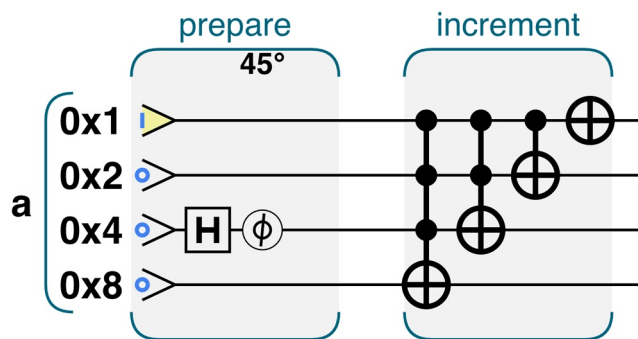
Two of the simplest useful integer arithmetic operations we can perform are those for incrementing and decrementing a number. Operations performing increment-by-1 and decrement-by-1:





Building increment and decrement operators

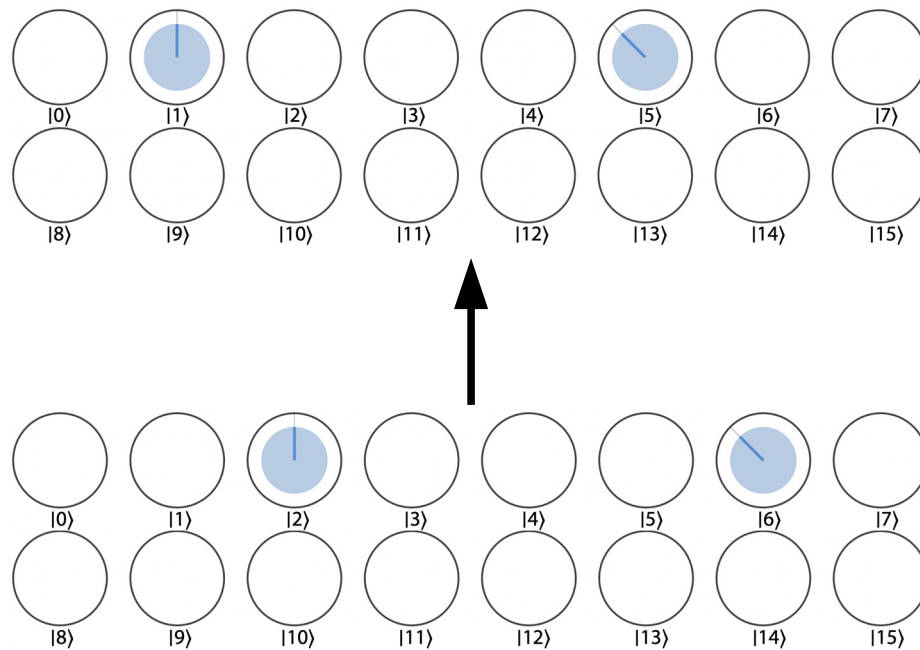
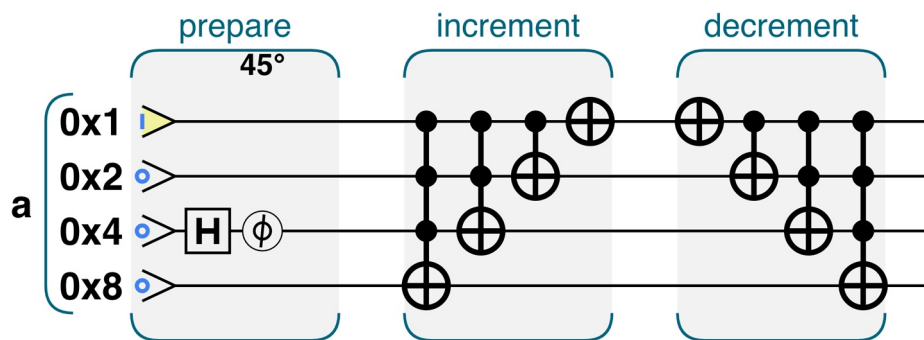
Two of the simplest useful integer arithmetic operations we can perform are those for incrementing and decrementing a number. Operations performing increment-by-1 and decrement-by-1:





Building increment and decrement operators

Two of the simplest useful integer arithmetic operations we can perform are those for incrementing and decrementing a number. Operations performing increment-by-1 and decrement-by-1:

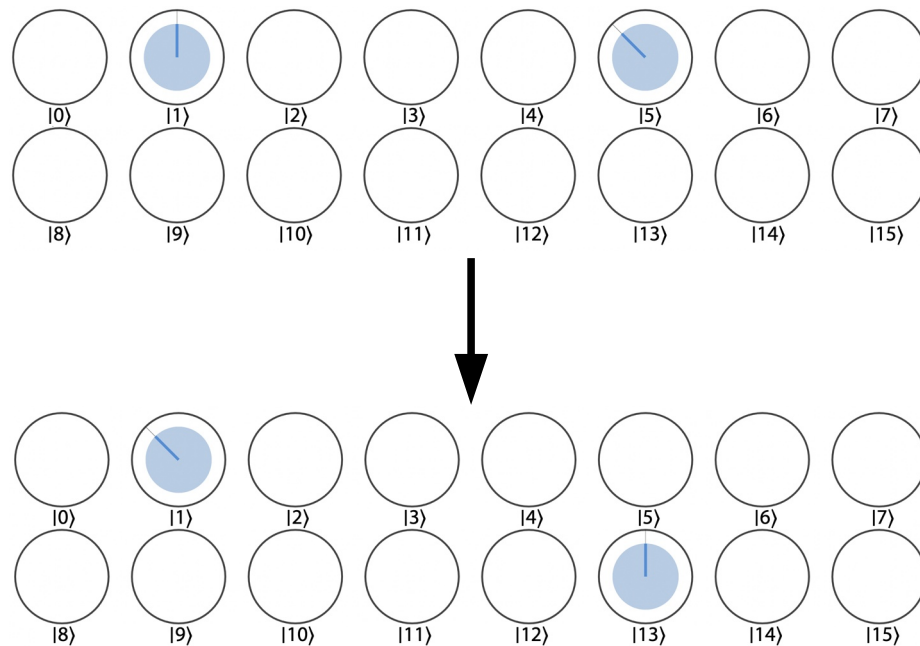
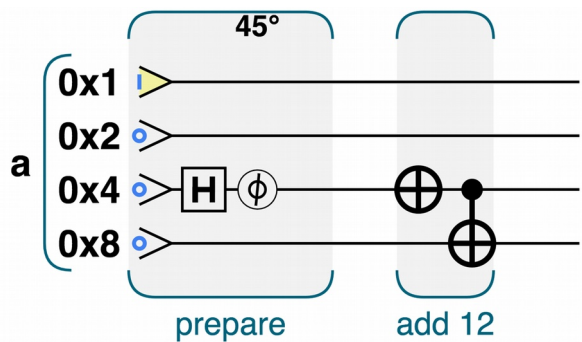




3.7 Quantum arithmetic and logic

Adding two 'quantum' integers

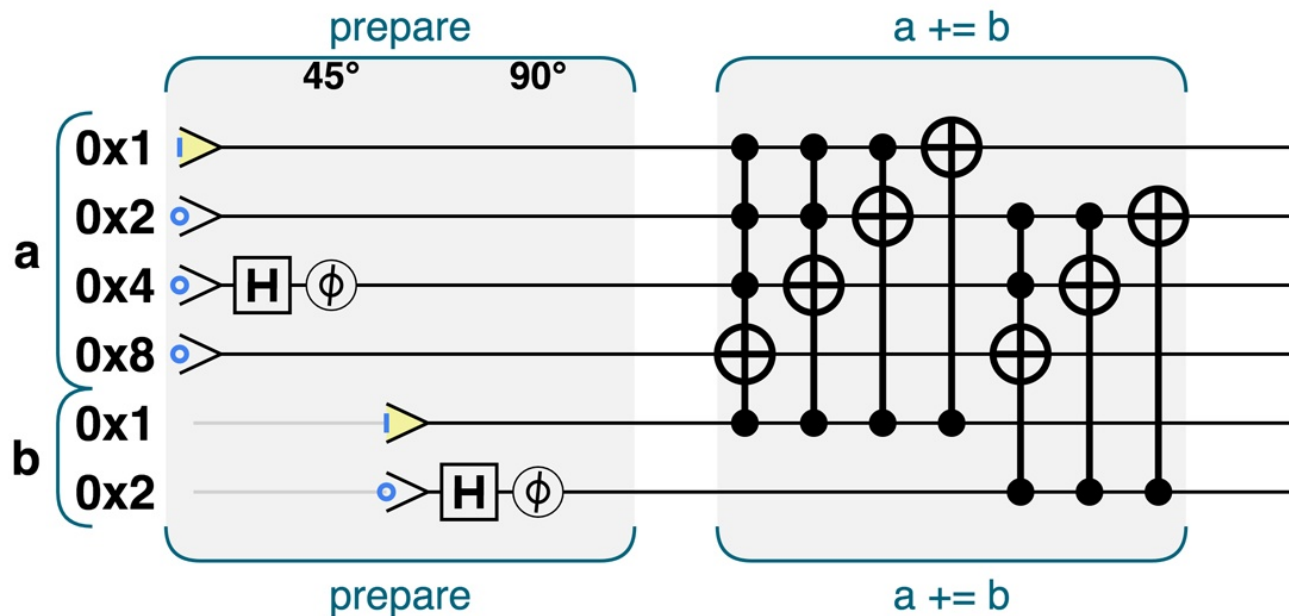
We can go beyond simple incrementing and decrementing. For example, `add(12)` produces the circuit:





Adding two 'quantum' integers

Unlike the previous approach for adding a conventional digital integer to a quantum register, here the gates don't need to be reconfigured every time the input values change:





Negative integers

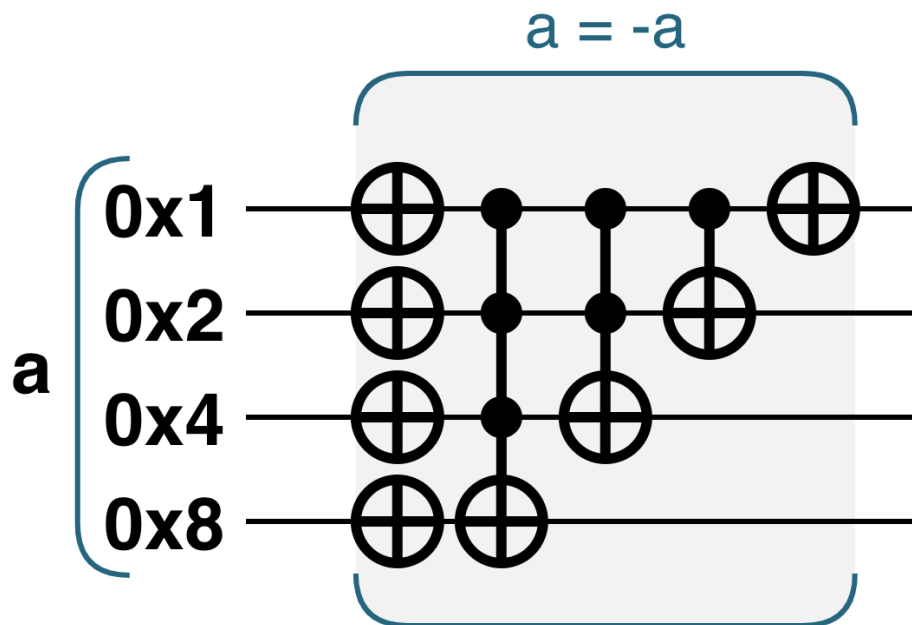
For a given number of bits, we simply associate half the values with negative numbers, and half with positive numbers. For example, a three-bit register allows us to represent the integers -4 , -3 , -2 , -1 , 0 , 1 , 2 , and 3 as:

0	1	2	3	-4	-3	-2	-1
000	001	010	011	100	101	110	111



Negative integers

To negate a number in two's complement, we simply flip all of the bits, and then add 1:

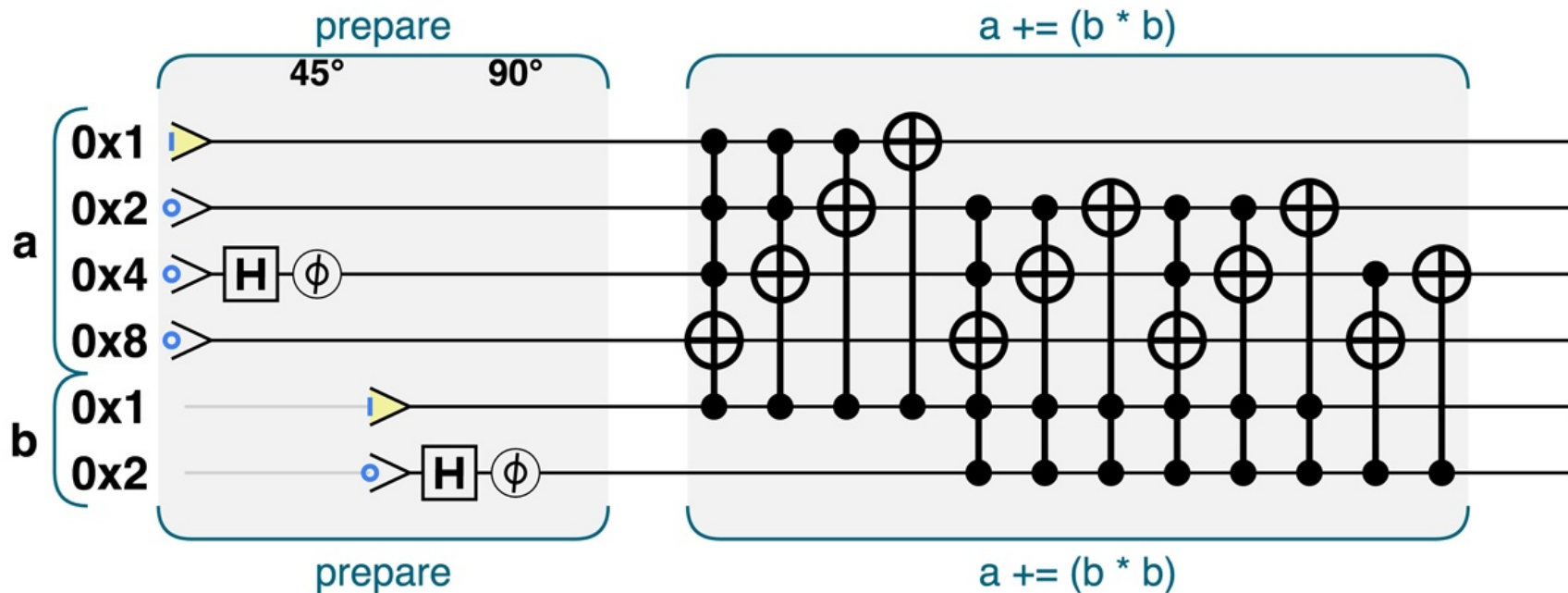




3.7 Quantum arithmetic and logic

More complicated maths

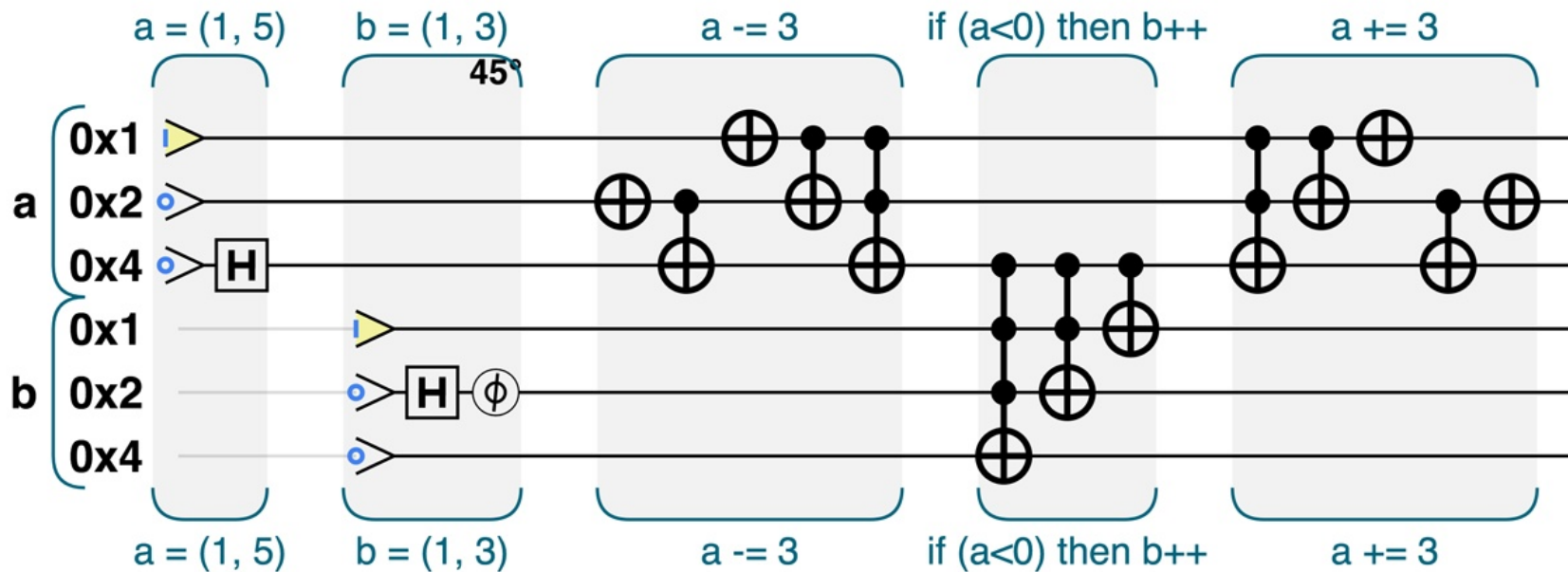
One more example before getting really quantum:





Quantum-conditional execution

With a QPU, a value can be both set and not set in superposition, so an operation conditional on that value will both execute and not execute in superposition:

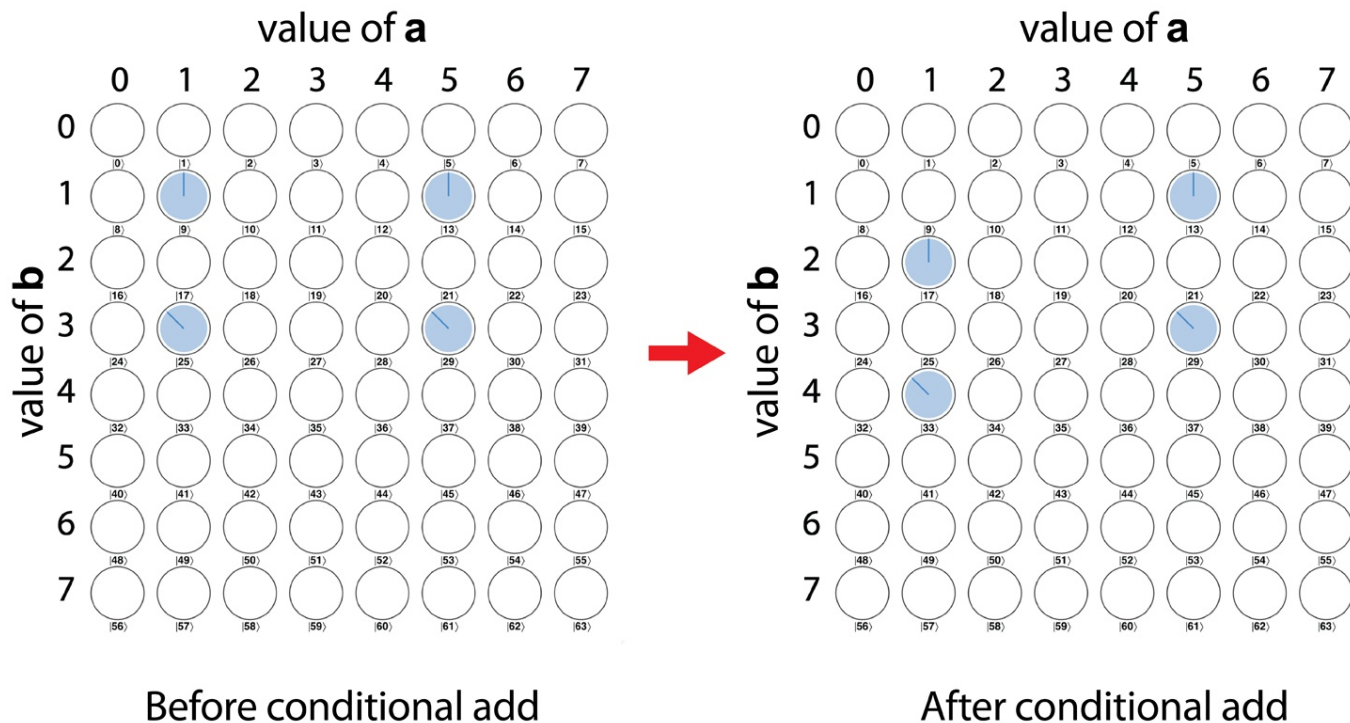




3.7 Quantum arithmetic and logic

Quantum-conditional execution

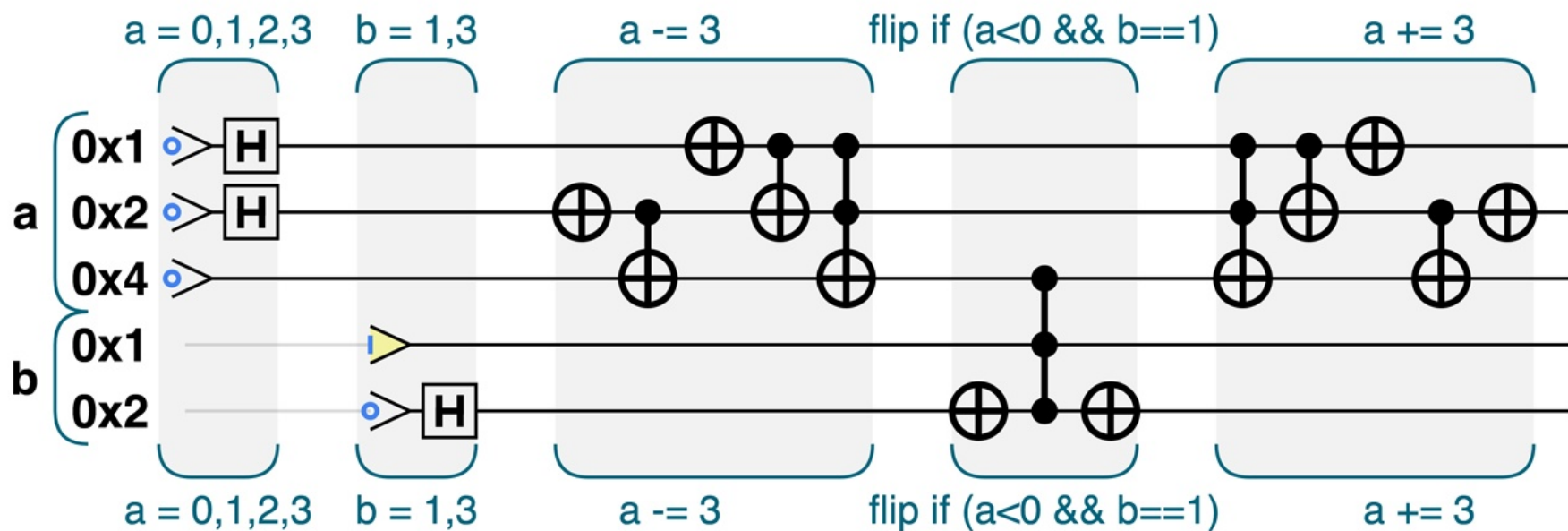
With a QPU, a value can be both set and not set in superposition, so an operation conditional on that value will both execute and not execute in superposition:





Phase encoded result

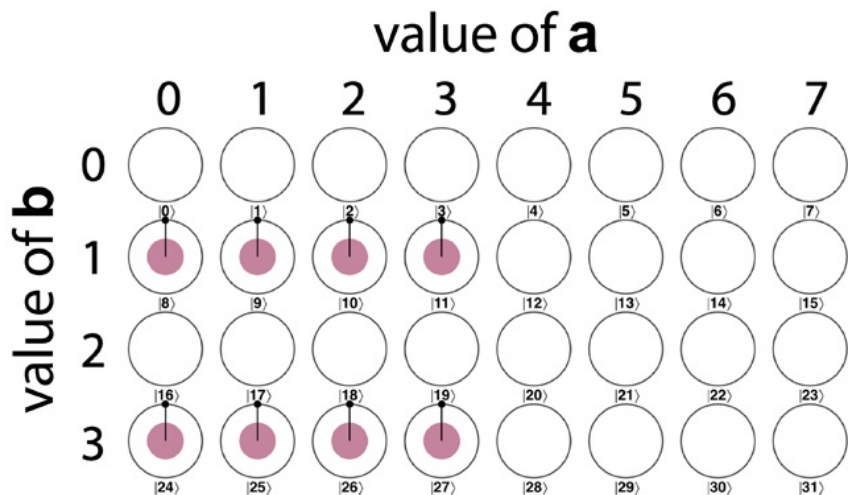
Our QPU versions of arithmetic operations can also be modified to encode the output of a calculation in the relative phases of the original input qubit register – something completely impossible using conventional bits:



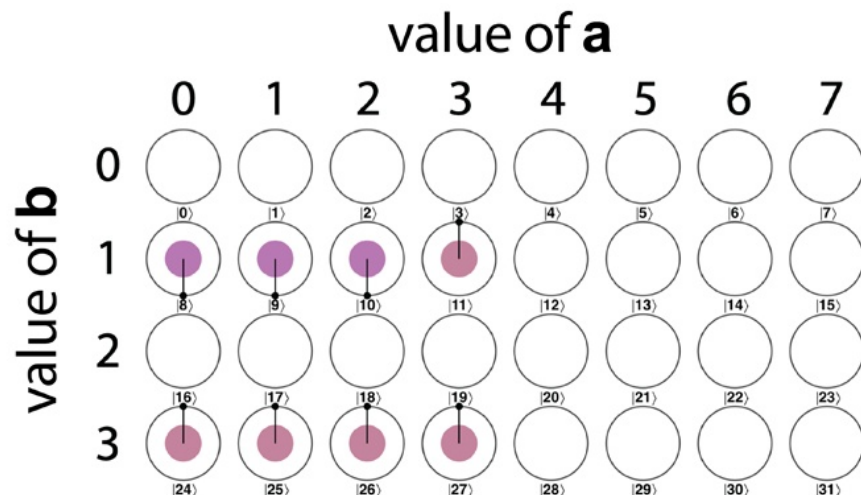


Phase encoded result

Our QPU versions of arithmetic operations can also be modified to encode the output of a calculation in the relative phases of the original input qubit register—something completely impossible using conventional bits.



Before conditional phase

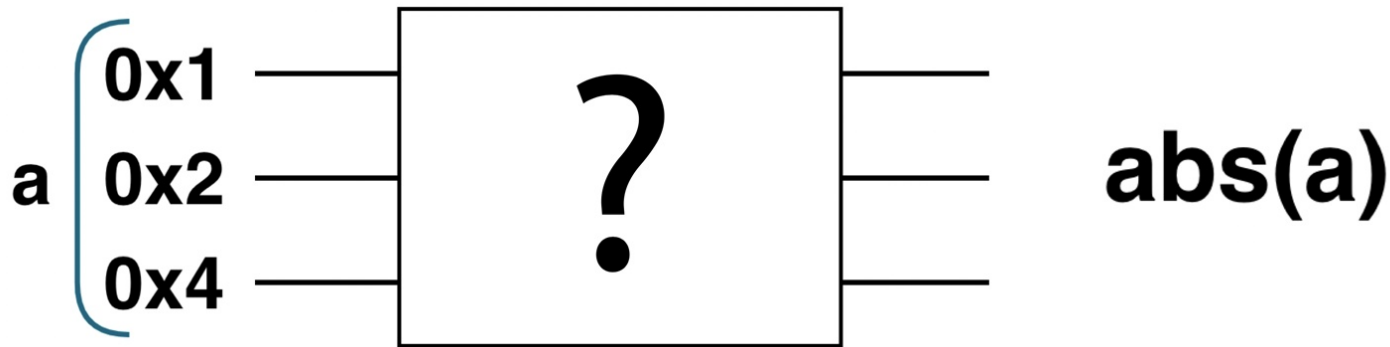


After conditional phase



Reversibility and scratch qubits

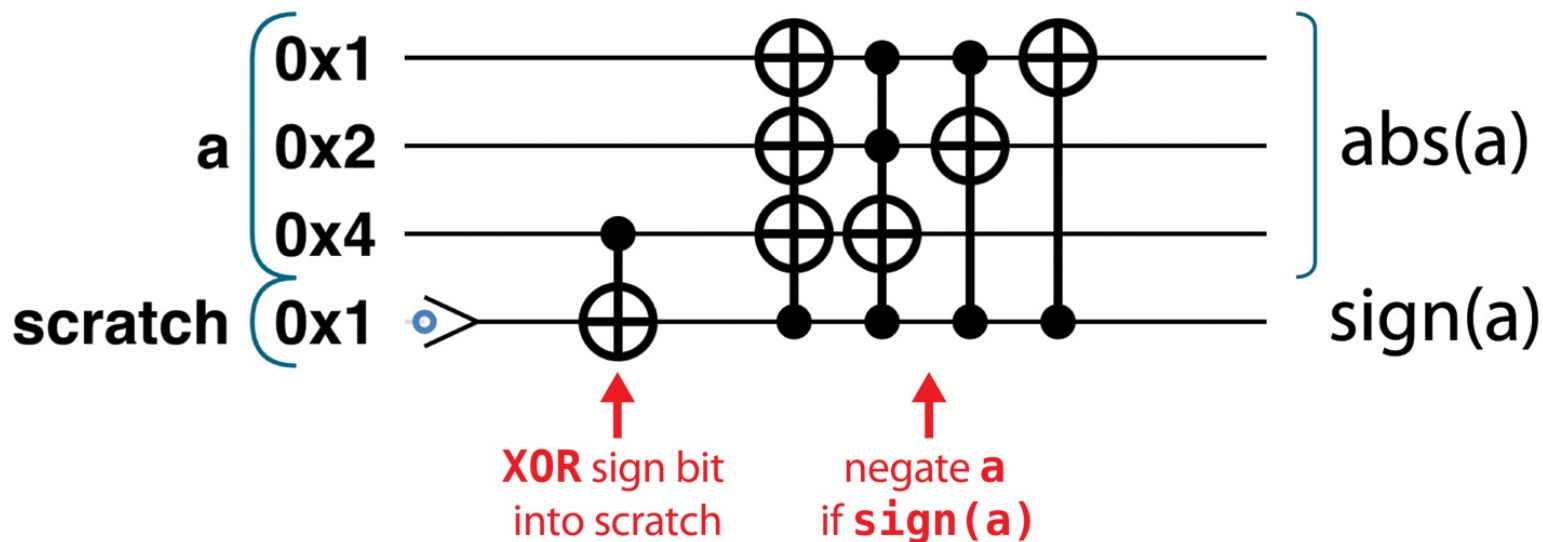
As already said, QPU operations need to be reversible. Although there's no hard-and-fast way to convert an arithmetical operation into a form that is reversible (and therefore more likely to work on a QPU), a helpful technique is the use of scratch qubits.





Reversibility and scratch qubits

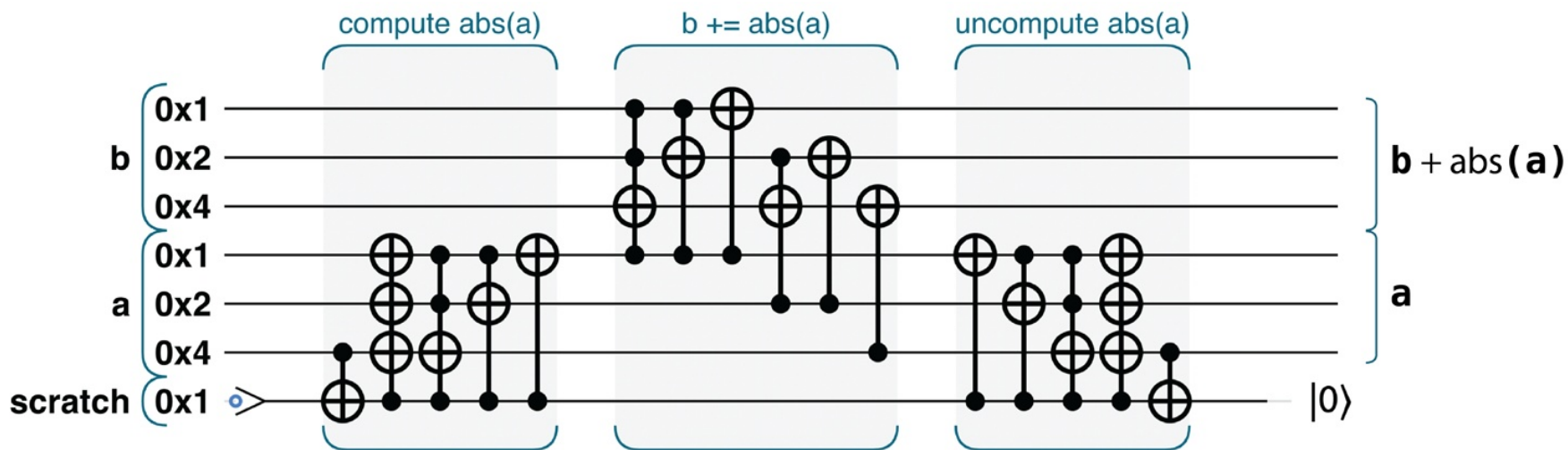
As already said, QPU operations need to be reversible. Although there's no hard-and-fast way to convert an arithmetical operation into a form that is reversible (and therefore more likely to work on a QPU), a helpful technique is the use of scratch qubits:





Uncomputing

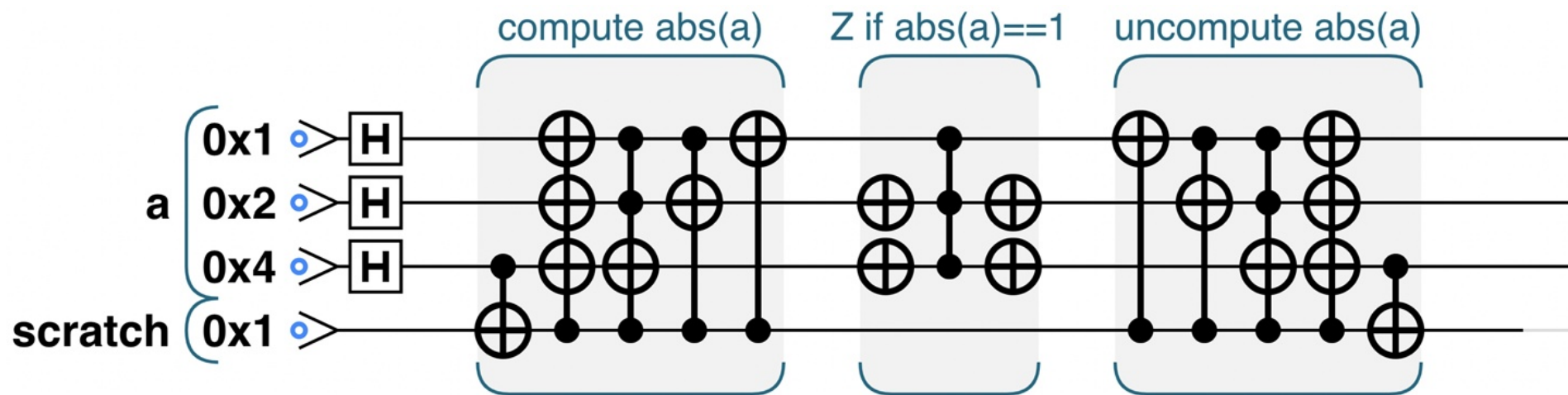
Although scratch qubits are frequently a necessity, they tend to get entangled with our QPU registers. This is bad news, since we now need to reset these scratch qubits to make them usable again further down the line in our QPU. Fortunately, there's a trick that solves this problem: it's called uncomputing. The idea of uncomputing is to reverse the operations that entangled the scratch qubit, returning it to its initial state:





Uncomputing

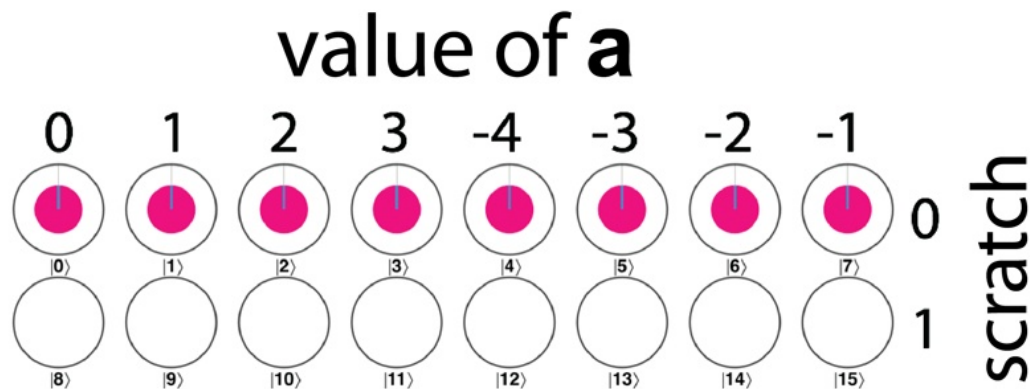
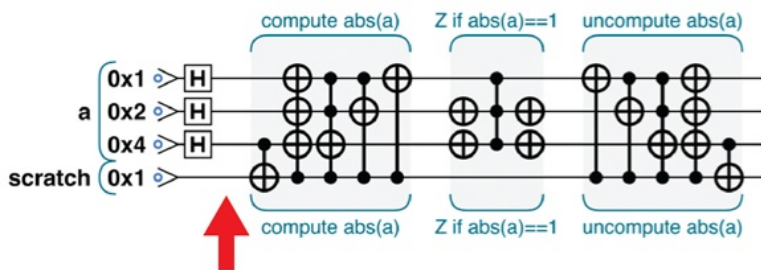
Another extremely common application of uncomputation involves performing a computation (potentially using scratch qubits), storing the results in the relative phases of the output register, and then uncomputing the result. So long as the initial computation (and thus also the final uncomputation step) does not interfere with the relative phases of the output registers, they will survive the process intact:





Uncomputing

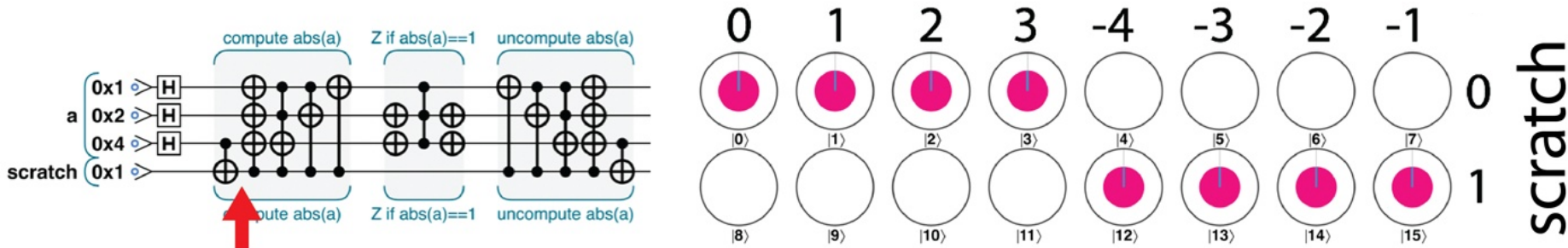
Another extremely common application of uncomputation involves performing a computation (potentially using scratch qubits), storing the results in the relative phases of the output register, and then uncomputing the result. So long as the initial computation (and thus also the final uncomputation step) does not interfere with the relative phases of the output registers, they will survive the process intact:





Uncomputing

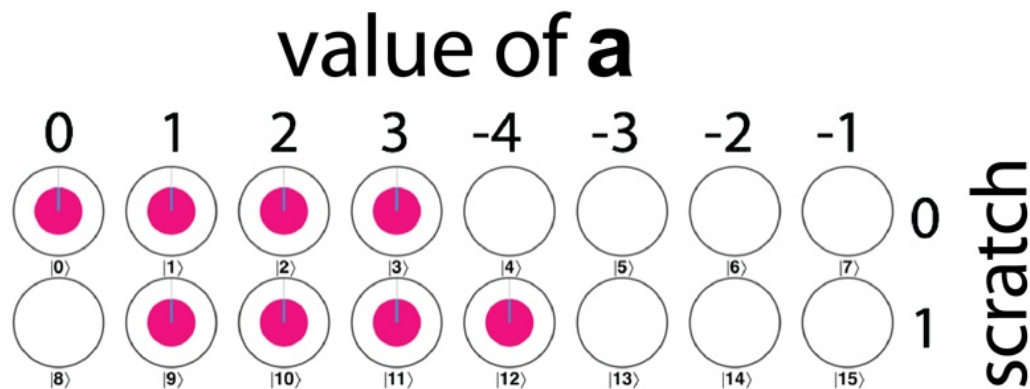
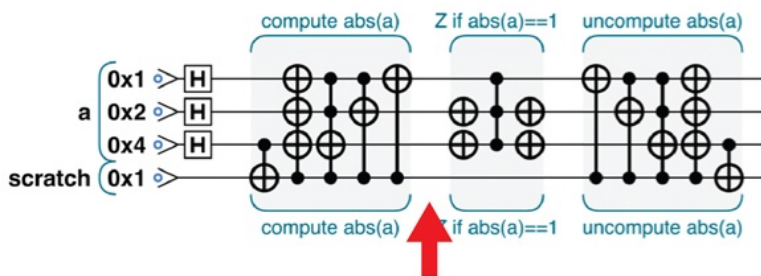
Another extremely common application of uncomputation involves performing a computation (potentially using scratch qubits), storing the results in the relative phases of the output register, and then uncomputing the result. So long as the initial computation (and thus also the final uncomputation step) does not interfere with the relative phases of the output registers, they will survive the process intact:





Uncomputing

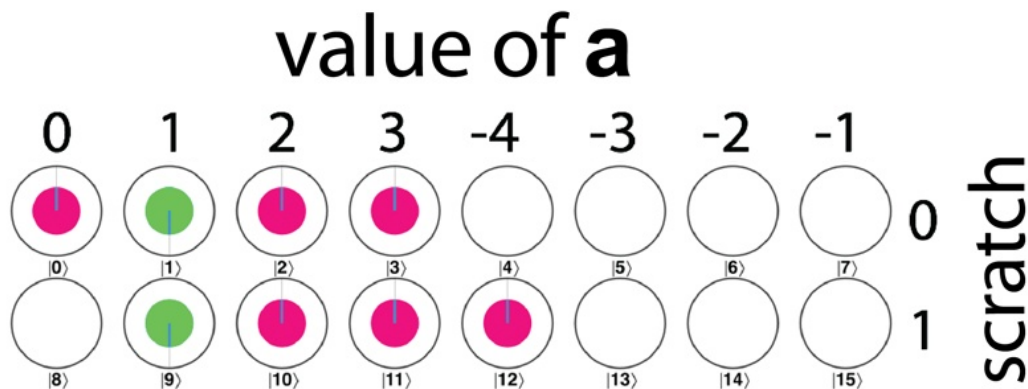
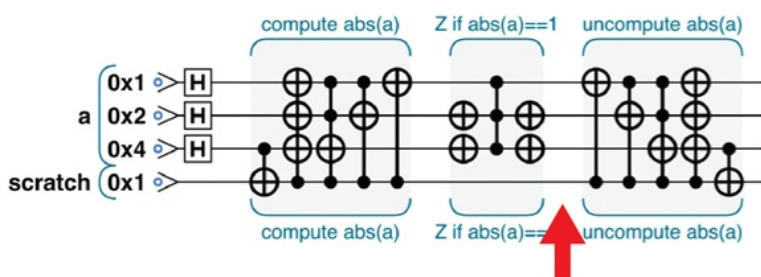
Another extremely common application of uncomputation involves performing a computation (potentially using scratch qubits), storing the results in the relative phases of the output register, and then uncomputing the result. So long as the initial computation (and thus also the final uncomputation step) does not interfere with the relative phases of the output registers, they will survive the process intact.





Uncomputing

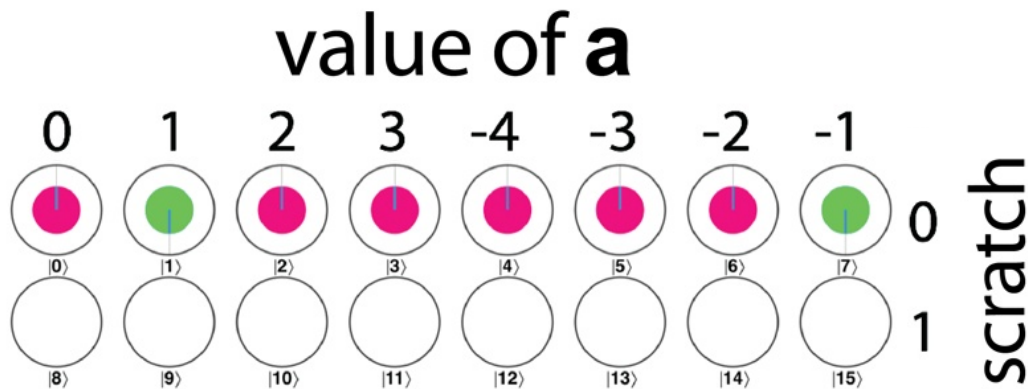
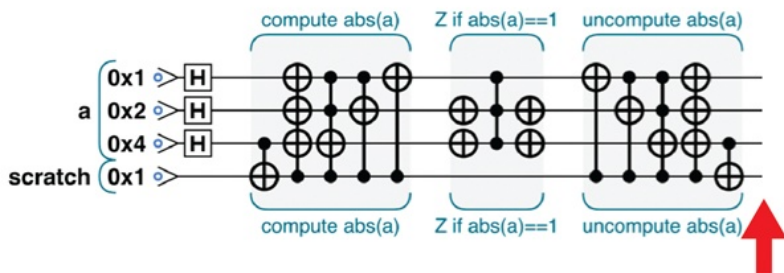
Another extremely common application of uncomputation involves performing a computation (potentially using scratch qubits), storing the results in the relative phases of the output register, and then uncomputing the result. So long as the initial computation (and thus also the final uncomputation step) does not interfere with the relative phases of the output registers, they will survive the process intact.





Uncomputing

Another extremely common application of uncomputation involves performing a computation (potentially using scratch qubits), storing the results in the relative phases of the output register, and then uncomputing the result. So long as the initial computation (and thus also the final uncomputation step) does not interfere with the relative phases of the output registers, they will survive the process intact.





Uncomputing

Another extremely common application of uncomputation involves performing a computation (potentially using scratch qubits), storing the results in the relative phases of the output register, and then uncomputing the result. So long as the initial computation (and thus also the final uncomputation step) does not interfere with the relative phases of the output registers, they will survive the process intact.

