How a quantum computer works: a step-by-step guide

Table of contents

The journey of information inside a quantum computer, from qubit preparation to final measurement, can be followed without a degree in physics, provided you take it in the order in which it actually happens. That is the order of this page. We will look at what occurs in each of the three phases of the computation, at what quantum logic gates and circuits really are, at the part played by superposition, interference and entanglement, at why cryogenics is needed and at the languages used to program these machines. The final section is for anyone who wants to get hands-on without owning the hardware, through quantum emulation. The unit of all this is the qubit.

How does a quantum computer work, in brief?

The three phases are preparation, evolution and measurement. In the first, the **qubits** are initialised in a known state, typically $|0\rangle$. In the second, a programmed sequence of quantum logic gates, that is the circuit, manipulates amplitudes and phases, so as to create the superpositions that encode the problem and to orchestrate **interference** between the different components of the state. In the third, the state is queried and collapses into a string of classical bits, **which is the only readable output**. Since the outcome is probabilistic, the cycle is repeated many times and the answer emerges from the distribution of results. Everything that sets a quantum computer apart from a classical one happens in the second phase, which stays invisible from the outside and, unlike classical computation, remains reversible. It is worth keeping this scheme in mind while reading the sections that follow, because every algorithm fills those three slots differently.

What happens during qubit preparation?

Every platform has its own way of returning the **qubits** to a known initial state. Superconducting circuits either wait for natural relaxation to the ground state or apply an active reset that shortens the wait considerably. In trapped ions, “optical pumping” drives the electrons into the desired level, while in photonic systems preparation coincides with generation itself. The quality of the initialisation is measured and declared, because an imperfect preparation would contaminate everything computed afterwards. Every circuit starts from this state of rest, the register $|00...0\rangle$. In the specifications of a commercial quantum computer, state preparation and readout fidelity is listed alongside gate fidelity, and it is one of the figures to look at when comparing different machines.

Typical state preparation and measurement fidelity (SPAM)

> To give a sense of the orders of magnitude, on superconducting processors the readout error sits around 2% or 3%, whereas on trapped-ion systems it falls below 0.1%, which explains why the two families of machines are chosen for different tasks.

What are quantum logic gates?

They are the machine’s elementary instructions, that is unitary and therefore reversible transformations of the state of the **qubits**. The ones in common use form a small alphabet. The **Hadamard** turns a definite state into a **balanced superposition**. Phase rotations, such as the S and T gates and the parametric rotations, rotate the components of the state and encode information in the phases, which is how an algorithm prepares the ground for **interference**. The CNOT, acting on two **qubits**, ties the target to the control and generates entanglement. A theorem guarantees that a finite set of gates, for instance Hadamard, T and CNOT, is universal, in the sense that any quantum computation can be approximated to arbitrary precision using those three building blocks alone. Universality is what makes a quantum computer a fully programmable machine, since the same hardware runs chemistry simulations or optimisation problems simply by changing the sequence of the gates described above.

Qubit counts and one- and two-qubit gate fidelity today

> The asymmetry between the two kinds of gate is the figure that matters most, because one-qubit gates fail once in tens of thousands of operations, whereas two-qubit gates, the very ones that generate entanglement, fail a hundred to a thousand times more often, and it is on these that the quality of a machine is really measured.

How is a quantum circuit assembled?

A circuit is drawn with one horizontal line per **qubit** and the gate symbols laid out in sequence, with time running from left to right. The minimal example that produces quantum correlation takes only two instructions.

A **Hadamard** on the first **qubit** and a **CNOT** between the first and the second produce the **Bell state** (|00⟩ + |11⟩)/√2, a perfectly correlated pair. Real circuits add depth, that is layers of gates in succession, and since **decoherence** degrades the state as the computation proceeds, the depth that can be executed before the result becomes unusable is one of the practical limits of current hardware, as discussed in the comparison between quantum and classical computers. Reading a circuit is quickly learnt, whereas writing one that turns **interference** to your advantage takes far more practice.

What roles do interference and entanglement play during the computation?

Interference is the mechanism by which the algorithm selects the answer. The components of the state that lead to wrong outcomes are driven into **antiphase**, so that they cancel one another, while the correct ones add up. **Entanglement** does something different, because it holds the register together through correlations between **qubits** that no independent description of the individual **qubits** can capture, and without which a classical computer would be able to simulate the circuit efficiently. In practical terms, a circuit with no **entanglement** is computing nothing that a laptop could not reproduce. The number of qubits only starts to weigh once the state is genuinely correlated, which is precisely the case in which classical simulation becomes expensive.

The threshold beyond which classical simulation becomes impractical

> The barrier is exponential in nature, since describing n qubits exactly requires 2^n complex amplitudes, that is 16 GiB of memory at 30 qubits, 16 TiB at 40 and 16 PiB at 50, which is why exact simulation of the full state vector stops today at around 40 qubits, unless one turns to compression techniques such as tensor networks. This limit applies to state-vector simulation. Tensor-network techniques reach well beyond these figures.

The circuit formalism and the Born rule

> The state of a register of n **qubits** lives in a **Hilbert space** of dimension 2^n, and the overall evolution produced by the circuit is written $|\psi_{fin}\rangle = U_m \cdots U_2 U_1 |00\ldots0\rangle$, where each U_k is the unitary matrix associated with a gate. The design of that product of unitaries is the algorithm itself. Measurement in the computational basis returns the string x with probability $p(x) = |\langle x|\psi_{fin}\rangle|^2$, that is the squared modulus of the amplitude associated with that outcome, according to the Born rule. One qualification is worth adding, which popular accounts usually leave out: **entanglement** is a necessary but not sufficient condition for quantum advantage, because the Gottesman-Knill theorem shows that there exist strongly entangled circuits, those made solely of Clifford gates, which a classical machine can still simulate in polynomial time. Entanglement is therefore required, and so is a circuit that steps outside the Clifford group, typically by means of **T gates**.

How does the final measurement happen?

Queried in the computational basis, each **qubit** returns 0 or 1 with a probability equal to the squared modulus of its amplitude, and the **superposition** vanishes. Readout is itself a physical process, with its own declared fidelity, carried out by collecting photons from a resonator in superconducting circuits or by observing the fluorescence of ions in traps. The result of a single run says little, which is why the standard protocol repeats the circuit hundreds or thousands of times, the so-called **shots**, and reconstructs the distribution of outcomes. A good algorithm is one that makes that distribution converge, with dominant probability, on the solution being sought. The number of shots is also a concrete economic lever, because commercial hardware is billed per run, and estimating in advance how many shots are needed for the required precision is an integral part of sizing any project, in both budget and time.

Typical shot counts and cost per run

> As of summer 2026, these are the commercially most telling figures. On Amazon Braket the per-task fee is $0.30 across all machines, while the per-shot price varies enormously, from $0.00090 on Rigetti Ankaa and $0.00145 on IQM Garnet up to $0.03 on IonQ Aria and $0.08 on IonQ Forte. It follows that a 100-shot circuit costs $8.30 on IonQ Forte and $0.34 on Rigetti, whereas a variational optimisation of 200,000 shots reaches $6,000 on IonQ Aria against $85 on Rigetti. It should be added that error mitigation on IonQ Aria imposes a minimum of 2,500 shots per task.

Why are cryogenics and isolation needed?

In physical quantum machines, **decoherence** is the factor that severely limits every computation. Any uncontrolled exchange of energy or information with the environment acts as an unintended measurement and destroys the **superposition** before the circuit has finished. Superconducting processors therefore operate at around 15 millikelvin, a temperature lower than that of the cosmic microwave background, inside dilution refrigerators that are the most expensive and bulkiest component of the whole hardware assembly, while isolated ions are confined in ultra-high-vacuum chambers by electromagnetic ion traps shielded from fields and vibrations. Not every platform requires extreme cold, since photons propagate at room temperature, although the most efficient single-photon detectors still work in a cryogenic regime. For this reason the study and prototyping stage is better carried out elsewhere, as we shall see in the closing sections.

Footprint, cooling power and consumption of a dilution refrigerator

> Here too the figures help to picture the proportions of this hardware. An industrial-class dilution refrigerator removes just a few tens of microwatts of heat at operating temperature, and only reaches a milliwatt at one hundred millikelvin, while requiring tens of litres of helium-3, a costly and hard-to-source resource. Cooling down to ten millikelvin takes about a day on an empty system, and appreciably longer once the refrigerator is fully wired, and every installation calls for a layout drawing dedicated to the room that houses it, while the chip at the centre of all this measures a few centimetres across.

Which languages and frameworks are used?

Programming is almost always done in Python, through open source frameworks that describe circuits at the gate level. Qiskit, Cirq and PennyLane are the most widespread, flanked by dedicated languages such as Q#. The workflow mirrors that of ordinary software, because you write the circuit, run it on a backend, whether real or emulated, analyse the measurement distributions and iterate until the expected behaviour appears. Intermediate standards such as OpenQASM make circuits portable across different platforms. The syntax is learnt in a few days, whereas reasoning in terms of amplitudes and interference takes a good deal of guided practice. Structured training on circuits in emulation considerably shortens the time a team needs to grow comfortable with this way of thinking.

How long a corporate training path typically takes

> HumanQ's training experience points to a wide range, which depends almost entirely on the starting point. A team already at ease with the concepts of superposition, entanglement and interference, and used to developing in Python, takes its first steps in emulation within a few weeks. Those who have to build both the conceptual foundations and the development skills from scratch need at least six months before working independently on circuits of their own.

How do you verify that the result is correct?

The probabilistic nature of the outcome imposes a discipline of verification that classical computing does not require. For problems whose solution is easy to check, such as factorisation, the proposed answer is verified classically. For the others, the statistical distributions obtained are compared with those expected on reduced instances, calibration circuits and error mitigation techniques are used, and the overall fidelity of the run is estimated. This is where emulation becomes a verification tool that is hard to replace, because the same sequence of gates, executed in an exact and noise-free environment, provides the reference distribution against which the hardware is judged. Without these baselines, telling an algorithm error apart from a machine error or a disturbance would be all but impossible. The same holds in production, where maintaining a regression pipeline in emulation, run at every change to the circuit, is the practice that separates a genuine industrial project from a mere demonstration.

Gate error rates and the threshold required

> The current order of magnitude on two-qubit gates is 10^{-3}, against a theoretical surface-code threshold of around 1% and a desirable operating value well below that, since the cost in physical qubits per logical qubit grows rapidly as the threshold is approached. The most telling comparison remains the one with cryptographic algorithms, given that Shor's algorithm requires millions to billions of operations to factor the integers underpinning RSA-2048 encryption, and with a 0.3% error per gate the errors pile up long before the computation reaches its end. At a ratio of 1,000 physical qubits per logical qubit, a thousand-qubit machine implements roughly a single logical qubit. This is the number that explains, better than any other, why we are still in the NISQ era, Noisy Intermediate-Scale Quantum.

How can you try all this without quantum hardware?

Through quantum emulation. The emulation platform used by HumanQ reproduces the entire cycle described on this page, from preparation through the gates to measurement, on high-performance classical infrastructure, and it offers two properties that real hardware, by its very nature, cannot provide. The state can be inspected at every point of the circuit, amplitudes and phases included, and every run is perfectly reproducible. You can watch the **Hadamard** create the **superposition** and the CNOT entangle the **qubits**, then check how **interference** selects the outcome. HumanQ develops quantum emulation projects precisely in order to work on what is already feasible today, without waiting for physical machines to become available, and for research teams and institutions this is an enormous opportunity for operational quantum computing.

Maximum size of circuits that can be emulated

> The dimensional envelope can be worked out in advance, since representing the state of n qubits exactly requires 2^n complex amplitudes. A stand-alone machine installed at the customer's premises works today on circuits of 30 or 31 qubits, which take up some thirty gigabytes of memory, and can be extended to 34 qubits with 256 gigabytes. For a research centre or a public administration this is the configuration that really counts, because the entire processing cycle, data included, stays inside the perimeter of the organisation and never crosses third-party infrastructure. Anyone without constraints of this kind can go further in SaaS mode, but the choice should be made knowingly, since it trades data sovereignty for computing capacity that may well go unused.

Run your first quantum circuit today

Get in touch with HumanQ and work through every phase of quantum computation, from preparation to measurement, on real use cases and with the support of our team. Discover quantum emulation, assess your company's quantum readiness or contact HumanQ.

Tags: entanglement, superposition, logic gates, Hadamard, qubit, quantum interference, quantum measurement, NISQ, quantum emulation.