Quantum Computation Introduction for Physicists

Class exercise 1

To understand the model of quantum computation, we need to understand a little bit of classical computation.

1 Bits and binary strings

The most basic component of information in a classical computer is a **bit**. A classical bit is some system that can be in one of two states, which we notate by 0 and 1. You probably know today that these bits are usually manifested by circuits that have or don't have current running in them, but this is not important for the model — it could be electrical circuits, DNA molecules, socks in a drawer, as long as the system can have two states it can be in, and we can distinguish between the two. A system of n bits can be in any of 2^n states, so in a system of n bits, which we sometime call a **string of n bits**, we can store an answer to a question that has 2^n answers. An important form we use bits for storing information is **binary numbers**: For a bit string $S = \otimes_{i=0}^{n-1} S_i, S_i \in \{0,1\}$, the represented number is $\sum_{i=0}^{n-1} S_i * 2^i$. We usually have a fixed number of bits, so we will perform our calculations under the **Modulo action** (residue).

So, for example, the string $10110 = 0\cdot 2^0 + 1\cdot 2^1 + 1\cdot 2^2 + 0\cdot 2^3 + 1\cdot 2^4 = 22$. Note that the rightmost bit is the first bit in the string here, and determines the coefficient of 2^0. It is often called the least significant bit. similarly, the leftmost bit is called the most significant bit.

If we want to represent a decimal number with a binary string, we determine the bits from the most significant to the least significant. For example, say we want to represent 13 with a binary string. We find the highest power of 2 that is smaller or equal to 13. In our case, it is $2^3 = 8$. So we add 1 in the third bit, and our representation is 1___ for now. We now take what is left of the number, $13 - 8 = 5$, and perform the same process. The highest power of 2 that is smaller than 6 is $2^2 = 4$. So in the second bit we add 1: 11__. We are now left with $5 - 4 = 1 = 2^0$, so the zeroth bit also becomes 1. We are done with our number, so everything left has to be zero: 1101. You can check that substituting $S = 1101$ in our formula for binary strings gives us 13 again.

We now move to represent our bits in a quantum notation. It is not so useful in classical computation, but will make our transition to quantum computation pretty smooth. A single bit will now be written as

$$|0\rangle \quad \text{or} \quad |1\rangle,$$

and a string of bits will be written as

$$|0\rangle \otimes |0\rangle \otimes |1\rangle \quad \text{or} \quad |001\rangle.$$

If we keep the number of bits in a string is fixed, which is usually the case, we will sometimes also represent the string by the decimal number it manifests:

$$|010011\rangle \equiv |19\rangle_6 \equiv |19\rangle.$$

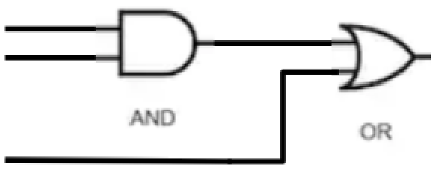
2 Boolean Gates

On top of storing information, we also want to manipulate the information in our bits and make calculations with it. One very popular model for this manipulation is the **boolean gates** model. We treat the 0 and 1 states as False and True, and define the following gates:

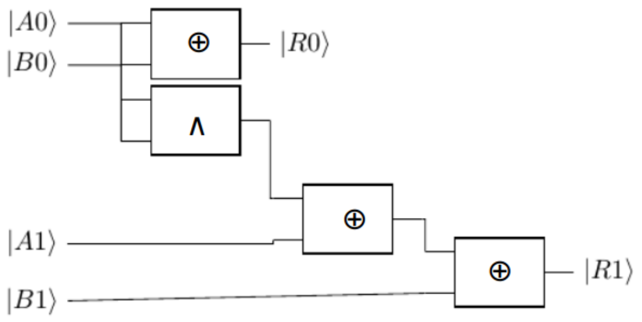
- NOT: $\neg 0 = 1, \quad \neg 1 = 0$
- AND: $0 \wedge 0 = 0 \wedge 1 = 1 \wedge 0 = 0, \quad 1 \wedge 1 = 1$

- OR: $0 \vee 0 = 0, \quad 0 \vee 1 = 1 \vee 0 = 1 \vee 1 = 1$
- XOR: $0 \oplus 0 = 1 \oplus 1 = 0, \quad 0 \oplus 1 = 1 \oplus 0 = 1.$

We define boolean circuits as a system of bits and boolean gates:



Question 1 Draw a circuit that adds two two-bit numbers (modulu 4).
Solution



The rightmost bit is (A0⊕B0) and the leftmost bit is ((A0∧B0)⊕A1)⊕B1. This is just like long addition in the binary basis.

Question 2
Universality of gates set: Prove that any boolean statement can be expressed using AND and NOT gates only.

Solution
Any boolean statement allows some possible strings of bits, so we can write it using only AND, OR and NOT gates: $(A \wedge B \wedge \neg C \dots) \vee (\neg A \wedge B \wedge \neg C \dots) \vee \dots$. This can be demonstrated with the following table:

expression	logical
001	$\neg x_1 \wedge \neg x_2 \wedge x_3$
010	$\neg x_1 \wedge x_2 \wedge \neg x_3$
011	$\neg x_1 \wedge x_2 \wedge x_3$
...	...

.

Say our boolean statement should end up as True for 001 and 011 and False for all others, we could write $(\neg x_1 \wedge \neg x_2 \wedge x_3) \vee (\neg x_1 \wedge x_2 \wedge x_3)$. Any additional string that satisfies our statement could be added to this expression by an additional OR.

We now show that any OR gate can be expressed only with NOT and AND: $A \vee B = \neg(\neg A \wedge \neg B)$. We persuade ourselves this is true using the following truth table:

A	B	$A \vee B$	$\neg(\neg A \wedge \neg B)$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

Thus, we can replace all OR gates in the general expression above with their AND and OR combination equivalent, and we are done.

Reversible computation What happens if we demand that all of our circuits will be reversible? Out of all of the gates we know so far, only NOT will be a gate we can still use. AND, OR and XOR are obviously not reversible: the input is two bits, so it can be in one of four states, but the output is only one bit, so it can be in one of two states — there is no way we can store all of the information of the input in the output. Instead, we can write other reversible two-bit gates:

$$\text{SWAP}|\alpha, \beta\rangle = |\beta, \alpha\rangle.$$

Now we can extract the input from the output easily.

On first glance, it may look as if we lose power when we restrict ourselves to reversible gates — we have to use more output bits, so we will need a bigger computer to make the same calculation. It turns out that this is not true. Erasing old, unused bits takes the same amount of effort from the computer, and the reversibility demand does not limit our computer's power. As physicists, this argument is easy to understand in terms of thermodynamics: A reversible process is adiabatic, hence creates no entropy. A non-reversible process creates entropy, which is here manifested in erasing the old bits so we can reuse them. The reason we care about reversible circuits is that this is the type of process we can apply for quantum systems using Hamiltonians, but this is something we will talk about next week.

So now that we work in the reversible computation model, all of our gates have the same number of input and output bits. It will be convenient to represent the bits' state in a vector, and the gates as matrices:

$$|0\rangle = \begin{pmatrix}1\\0\end{pmatrix}, \quad |1\rangle = \begin{pmatrix}0\\1\end{pmatrix}, \quad \text{NOT} = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix},$$

$$|00\rangle = |0\rangle_2 = \begin{pmatrix}1\\0\\0\\0\end{pmatrix}, \quad |01\rangle = |1\rangle_2 = \begin{pmatrix}0\\1\\0\\0\end{pmatrix}, \quad |10\rangle = |2\rangle_2 = \begin{pmatrix}0\\0\\1\\0\end{pmatrix}, \quad |11\rangle = |3\rangle_2 = \begin{pmatrix}0\\0\\0\\1\end{pmatrix}, \quad \text{SWAP} = \begin{pmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{pmatrix}.$$

We still use the circuit notation, but now all of our gates have the same number of input and output bits.

Our first requirement of a reversible gate is to have the same number of input and output bits. But we require more - we require all the matrices to be reversible, so no two input states can be mapped into the same output state. This means, in our model, that all matrices are **permutation matrices**, with each line and row vector having all entries equal to zero, except for one that is equal to 1. We can see this is the case for the two example gates we saw above.

An important 2-bit gate we are going to use is the **C-NOT**, or controlled-not, gate. This gate does nothing if the first bit is $|0\rangle$, and it flips the second bit if the first bit is $|1\rangle$:

$$\text{CNOT} = \begin{array}{c|cccc} & 00 & 01 & 10 & 11 \\ \hline 00 & 1 & 0 & 0 & 0 \\ 01 & 0 & 1 & 0 & 0 \\ 10 & 0 & 0 & 0 & 1 \\ 11 & 0 & 0 & 1 & 0 \end{array}.$$

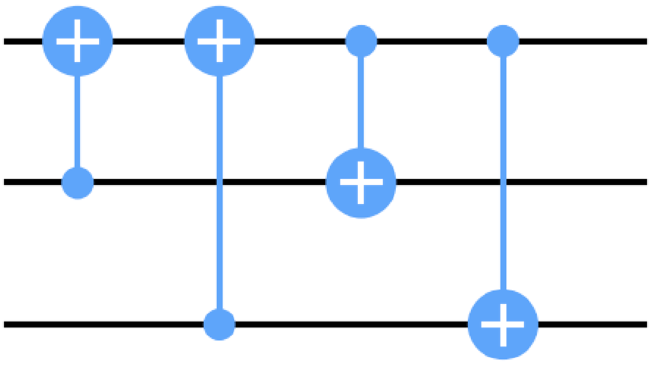
We call the first bit the **control bit** and the second bit the **target bit**.

Question 3

Draw a reversible circuit implements the SWAP gate using only CNOT gates. You can use an additional auxiliary bit which is initiated to $|0\rangle$.

Solution

The top bit is the auxiliary bit:



The gate in the picture in the common notation for the CNOT gate.