Points pertaining to Physics

1. The most important sub-product of the project, described below, is far more deserving of discussion- especially for its connect with simulating QFT. However, I neglected this, because the actual idea took so long to conceive, that I had hardly any time to implement it. It will surely be implemented, but it is the idea itself that is fairly intriguing.

First, a prologue somewhat divorced from physics. In implementing the simulation of a stored program computer, I have been forced to make an indirect distinction between the part of memory that holds *data* and the part that holds *program instruction codes*. This is something that is regrarded by computer scientists as antithetical to the rational foundation of the stored-program concept. What was this distinction? The *instruction codes* are stored in *classical memory*, whereas the *data* is stored in *quantum memory*. This is, mainly, because it is easy to implement this- and the analogy with classical systems is greatly simplified.

However, the problem of removing this distinction and "unifying memory", so to speak, has a highly instructive solution. If the instruction code C of some instruction is stored in some n qubits, then we can achieve a superposition of instructions. Let us say that the instruction argues *data* stored in some m qubits. Then the instruction code may be prepared as a quantum state:

$$|C\rangle = a_0|C_0\rangle + a_1|C_1\rangle + \cdots + a_{2^n-1}|C_{2^n-1}\rangle$$

arguing qubits $q_0, q_1, ..., q_m$.

We interpret this as: "*Execute instruction* C_i *with probability* $|a_i|^2$ *with* $q_0, ..., q_m$ *as the arguments*". You will notice that this operation, again, asymmetrically treats the instruction codes and the data (arguments)- only magnitudes are argued for the former, whereas the latter is argued whole. It turns out that the implementation below argues both of these whole, but this interpretation is useful for a number of reasons, at least one of which will become apparent below.

Let U_i represent the (unitary) operator corresponding to the i^{th} instruction. Then, what we need to do is:

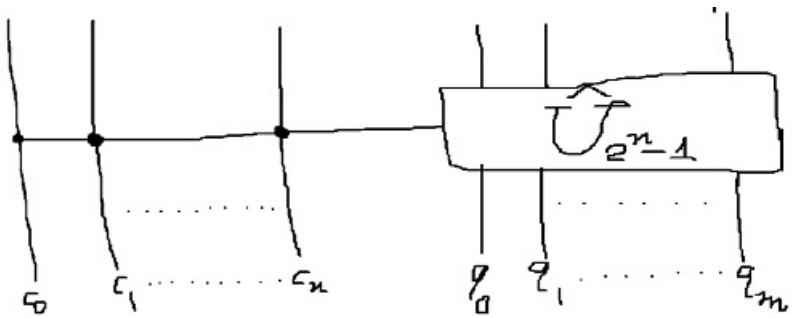
1) Begin with i at $2^n - 1$.

2) Execute the instruction corresponding to instruction code C_i on $q_0, ..., q_m$, but **controlled** by $c_0, ..., c_n$.

3) Decrease i by 1 and apply X-gates to $c_0, ..., c_n$ according to the below scheme.

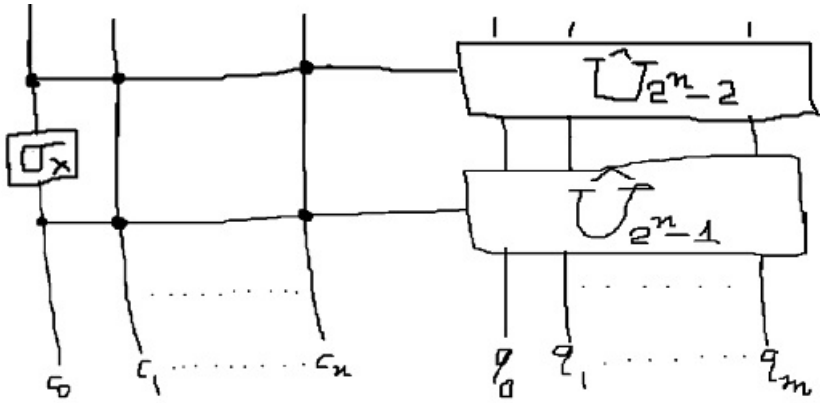
4) If i>0, go back to (2).

Now, we understand that the circuit:



applies U_{2^n-1} to $q_0, ..., q_m$ with the probability $|a_{2^n-1}|^2$.

Now, if we extend it as:



then it applies- U_{2^n-1} with the probability $|a_{2^n-1}|^2$, **and** U_{2^n-2} with the probability $|a_{2^n-2}|^2$ to $q_0, ..., q_m$. If we continue on this line, we can apply all of $U_0, ..., U_{2^n-1}$ to $q_0, ..., q_m$ with the probabilities $|a_0|^2, ..., |a_{2^n-1}|^2$, as required. Not only this- you can minimise the number of operations performed on $c_0, ..., c_n$ by progressing through the sequence $0, ..., 2^n - 1$ in the sequence corresponding to the **Gray-coded** sequence (reverse order). In this way, only one X-gate need be applied to one of $c_0, ..., c_n$ for each instruction, followed by the application of the controlled-U.

Now, it immediately becomes clear how this corresponds to a circumstance in QFT. Instead of describing the larger circumstance, we may take the example below.

Suppose we have two electrons far apart. The first electron begins in a momentum eigenstate with some small velocity, and we may solve the Dirac equation for it in "initially free space"- $\boldsymbol{i\gamma^\mu \partial_\mu \psi_1 - m\psi_1 = e\gamma_\mu A^\mu \psi_1}$ to get the $\boldsymbol{A^\mu}$ *due* to it. The second one moves into a region where its interaction with A^μ suddenly becomes significant. Now, we may either solve the new Dirac equation for ψ_2, or- if the charge/velocity etc. of the second is small enough to neglect its reaction to its *own* field, then we may simply use the Fourier-expanded version of A^μ, convert the coefficients to operators etc. and then apply those operators to ψ_2.

I feel like there must be some limit wherein we may be allowed to do the following:

1) Take $t = 0$.

2) Solve the free Dirac equation for particle-1 at time t for evolution over a small interval dt.

3) Save the momentum-spectrum (and position spectrum, maybe) of particle -1.

4) Solve the Maxwell equations with particle-1 as source, and substitute *probabilistic* values of the position and J using the above spectrum.

5) Convert the *probabilistic* fields obtained in (4) to operators as generaly prescribed (coefficients in Fourier expansion etc.)

6) Apply these operators to ψ_2 and write down new *probabilistic* bispinor.

7) (Optional) Repeat for further dt, but the approximation may lose validity quickly.

Now, you can see how a simulation of this is implemented by the above circuit- the $c_0, ..., c_n$ contains ψ_1 and $q_0, ..., q_m$ contains ψ_2.

Therefore, this (or some generalisation of this example) should enable us to do at least one of three

things-

1) Simulate certain special problems of QFT

2) Use actual systems modelled by QFT to implement the stored-program concept

3) Find limits of validity of the above method of solution using quantum circuits, and the correct solution (eg. solving the eqn for ψ_2 with A^μ as the external field) or results of a direct experiment.

2. The so-called *physical* problem referred to in the first few paragraphs of the paper refers to the problem of finding Hermitian operators for various physical quantities whose eigenvalues could act as implementations of the *alphabet* of computation. Taking a look at a history of imlpementations of the qubit, I can see a couple of quantities that surely do not have a finite set of eigenvalues under any well-known circumstances. For instance, the implementation whereby $|0\rangle$ was taken to be the ground state and $|1\rangle$ was any state of higher energy seems to correspond to some operator for a quantity, like $\lfloor \frac{2}{1+e^{-(E-E_g)}} \rfloor + 1$ (where E_g is the ground energy) which will have finite eigenvalues {1,2} for sure. But this is, of course, fanciful. Indeed, the construction in "Just an Idea" was, in part, motivated by the problem of finding Hermitian operators for different quantities via canonical quantisation. As you know, this is hindered by the Groenwold-van Hove theorem. However, there just might be a way to exploit that theorem itself to solve the problem at hand. As you can see in "Just an Idea", the periodic square wave of period 2^N (for various integral N) is fundamental to the composition of just about any well defined function in terms of elementary functions. Taking this into consideration, it may become necessary to supplant the trigonometric expressions in our construction with some Chebyshev polynomial approximations of the square wave *directly*.

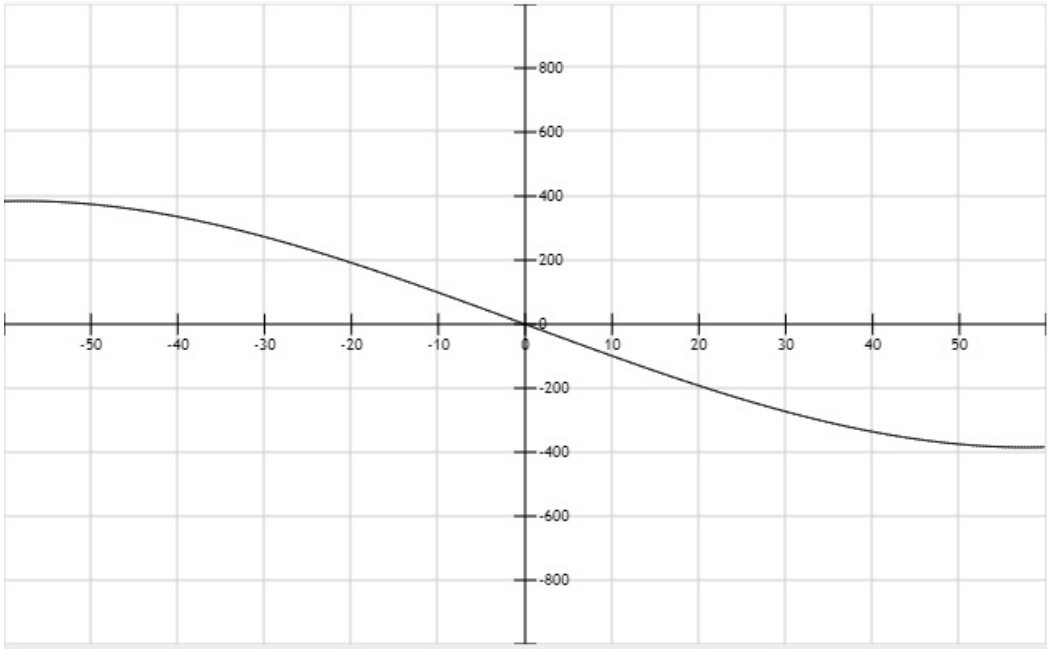
In any case it is somewhat easy to find a cubic polynomial to locally approximate the logistic function. The document entitled "Implementing qubits- program" contains a method to direct the computer to generate operators obtained by blindly following Dirac's original rules for canonical quantisation. Like-

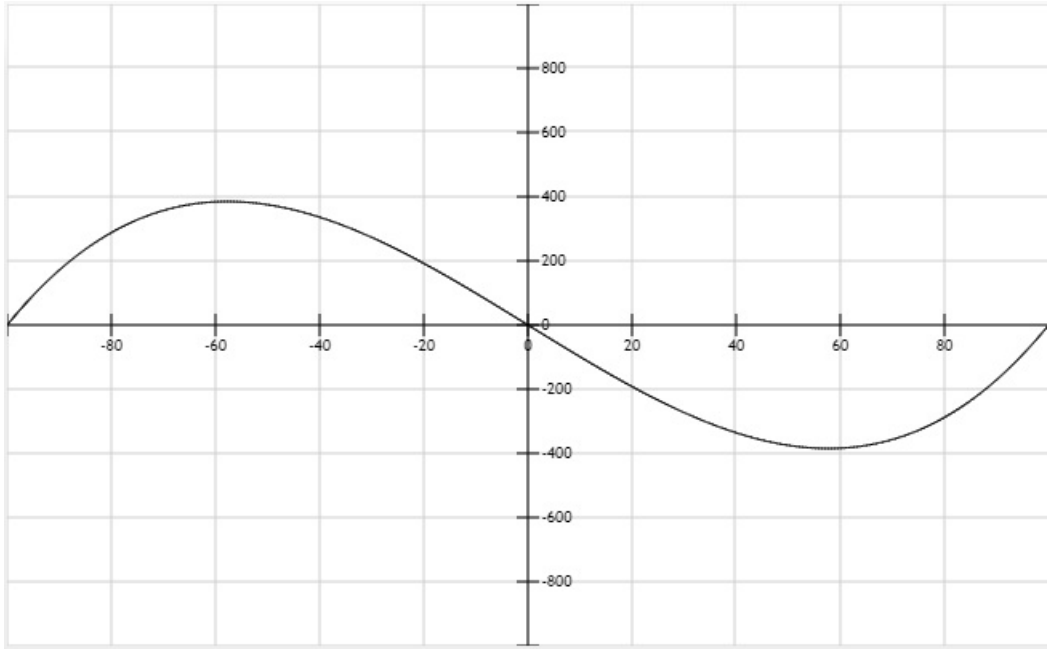
$$\hat{I} + 2\sum_{r=0}^{\infty}\binom{-1}{r}(\hat{I} + \sum_{n=0}^{\infty}\frac{1}{n!}(-\hat{H})^n)^r - \sum_{j=1}^{\infty}2^{-j}(\frac{\hat{I}}{2} - 2\sum_{\nu=1}^{\infty}\frac{\nu mod 2}{\pi\nu}\sum_{i=1}^{\infty}\frac{(-1)^{m+1}(2\pi\nu)^{2m-1}}{(2m-1)!}$$
$$\cdot 2\sum_{r=0}^{\infty}\binom{-1}{r}(\hat{I} + \sum_{n=0}^{\infty}\frac{1}{n!}(-\hat{H}^n)^r)^{2m-1})$$

which was the result of trying to discretise the Hamiltonian to 2 levels. They will surely satisfy 1 or 2 of the 4 associated caveats. However, I suspect this is largely worthless without the "Poisson bracket" caveat. (There may be a small error in the method of the program too. We have Taylor-expanded the trig functions, which must never be done for trig series- they will *surely* diverge at the discontinuities.) Finding the sine or cosine of a linear operator whose eigenvectors you know is exceedingly simple. So we *know* (had I not Taylor expanded the trig functions) that the resultant operators are the faithful representatives of the corresponding physical quantities, within the limits of canonical quantisation's correctness.

For whatever it is worth, we also discover a new way to reaffirm Groenwold's doubts about canonical quantisation by considering the full problem. However this proof too argues only one form of the functions of position & momentum- the "Just an Idea" form- just like Groenwold's own proof only argued polynomials. Let us take the subset of the domain of a square wave centred about a discontinuity and smaller than the period (). Intuitively, we can imagine how a cubic polynomial can approximate this just as we can understand how a cubic polynomial can approximate the logistic function. A poor candidiate would be-

$y = 0.1x(0.1x + 10)(0.1x - 10)$





To improve upon this, we require a sharper transition and a smaller curvature. When we quantify these two conditions, it would amount to an imposition of 4 conditions of 4 exact constraints on its derivatives. Since the derivative of a cubic polynomial is a quadratic polynomial, and a quadratic polynomial has at most 2 distinct roots, we can apply at most 2 such constraints. Therefore, we can never obtain a satisfactory cubic approximation. It must also be noted that (for the reason mentioned in the excerpts from the 2019 mail, vis-a-vis Cantor and the cardinal numbers) most functions for which we would like operators, would correspond to high curvature graphs. Since a quadratic polynomial has an upper or lower bound, that means that the magnitude of curvature of a cubic polynomial is bounded.

Points pertaining to quantum computing

1. There is a great deal to be said about the restriction of the choice of gates in the composite circuits, the expansion of the run of the LB-Turing machine, CNOT without entanglement etc. The

document doesn't even begin to scratch the surface of the origin or usefulness of these notions, and the associated explanations must wait till the full report is modified. I can, if you want, explain some of it the next time we meet.

A few points-

i) The restriction of gates is not absolute. Indeed, in implementing our model, the ALU alone had any actual qubits, while all qubits are virtual when not operated upon- their state (**including their phase in most cases**) is saved in classical memory at the cost of 18 classical bits per qubit. (I could have implemented it at a far lesser cost, but this was left for the future.) In order to prepare qubits in the ALU for the operation (execution) of any instruction, some of the more flexible gates ("advanced gates" in the IBM Q) are required to prepare the qubits correctly. To wit:

Classical memory		**Quantum memory**
(r, θ_0, θ_1)	$\rightarrow$	$u_1(\theta_0) \times \sigma_X \times u_3(2\cos^{-1} r, \theta_1, 0) \times \lvert 0 \rangle$

ii) The simplest method to obtain a function that is $\Omega(T(n))$ is by simply multiplying by n. Unless I'm greatly mistaken, $nT(n)$ is always $\Omega(T(n))$, but the given technique is better for a few reasons. The first reason is formal. The given method always yields a polynomial result, regardless of the form of $T(n)$. So, we *know* it is easily implemented by a computer. The second is empirical- the given method tends to yield a function that grows larger than its argument very quickly- for the given example, just a single term sufficed to get what we wanted!

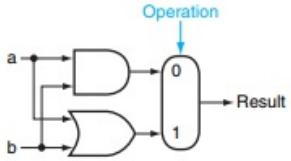
iii) There is highly frequent measurement done in this system for a few reason. One, we cannot maintain many qubits for long periods of time and expect them to be immune to decoherence.

In this light, the policy of maintaining the state of qubits *not* currently being argues by the ALU in classical memory isn't to simply for simulation purposes. Indeed, this is the best way to implement quantum memory in hardware, given the cost of implementing stable qubits. As a matter of fact, as is apparent from this, this further enables the use of the IBM Q (even with its meagre 5-25 qubits) as the underlying hardware on top of which our system runs. It has been observed that even the application of 4-5 "identity" operators ("delay" in the IBM Q) changes the state of the actual qubits- due to decoherence. Hence the policy of "measure ASAP".

There are a few more reasons, but they are all related to other parts of the system.

2. The references to Cook and Reckhow's work in the document can be largely supplanted by simpler references to the basics of practical computer engineering. To wit, the working of a modern *ALU*- **A**rithmetic and **L**ogic **U**nit-, which is at the centre of the computer's operation, is highly instructive.

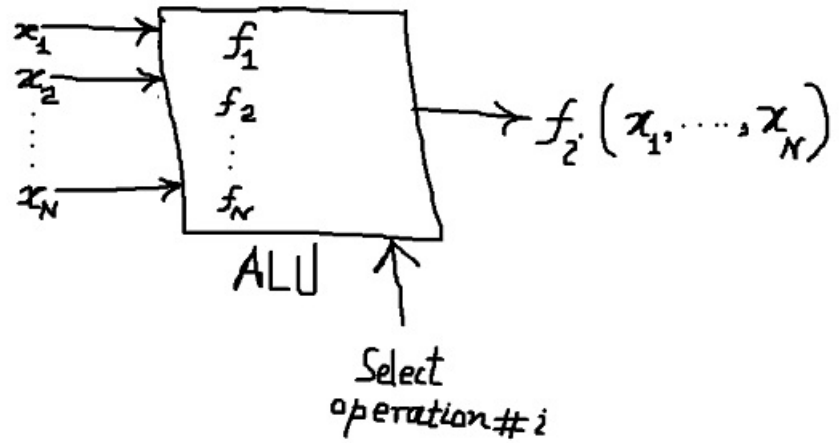
There is a nice illustration of its working in Hennessey and Patterson's "Computer Organisation and Design". In one of the appendices of that book, they begin by showing us the below circuit-



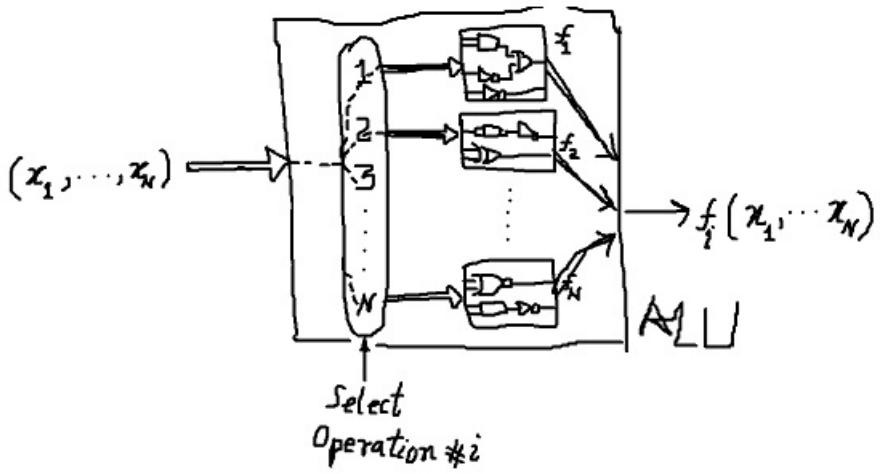
This is intended to represent a simple ALU, which takes in two single-bit inputs, a and b, and yields their logical OR or their logical AND based on the value of the third input- *Operation*. This neatly summarises the working of any ALU.

The generalisation of this is as follows. The ALU takes in inputs of some fixed size, and there is an additional input, i.e. the *Operation Selection*, which decides which operation must be performed on

these inputs to yield an output. You can imagine that the *Operation Selection* input is a number which identifies a function to apply to the other inputs-



The $f_i(x_1, ..., x_N)$ expression is the output of the ALU. The f's are implemented using simple logic circuits, like:

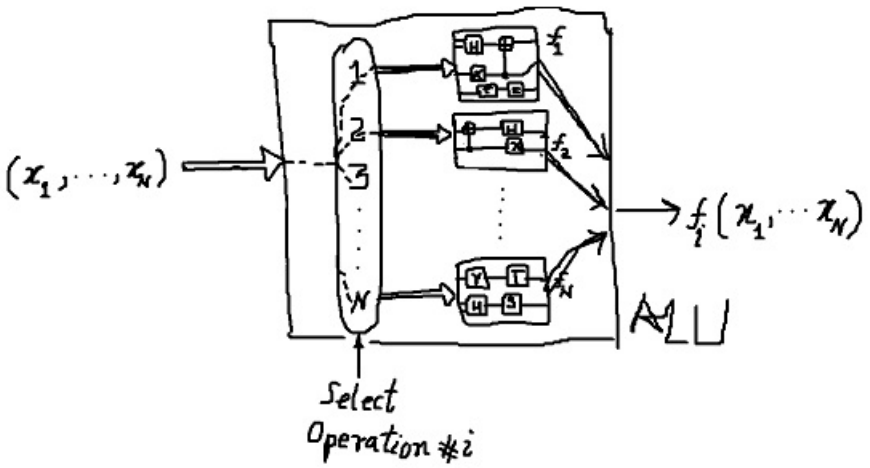


In this way, **every *instruction* in a program identifies a function/logic circuit in the ALU. Furthermore, the arguments/variables/data argued by the instruction correspond to the x_1, x_2, ...etc. inputs to the ALU**. Internally, the instruction is translated into a number, which is fed to the ALU as the *Operation Selection*.

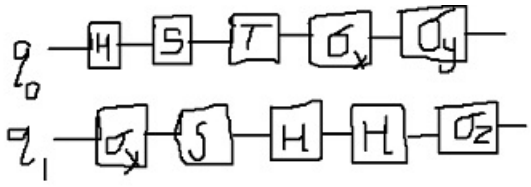
To be clear, this applies to *machine language* programs, and, to a lesser extent, *assembly language* programs.

Now, the fallacy of referring to *OpenQASM* and similar systems as assembly languages should become apparent. Single instructions in these languages involve the application of single logic gates to one or more (qu)bits, whereas a real assembly/machine language would identify an *instruction set* (corresponding to the f's above) in which **each instruction would correspond the application of a (relatively) complex logic circuit to the bit-inputs**.

The USS-QA project aimed to create a *meta simulator* (phrase due to Donald Knuth)- very simpleto use,- that would allow users to design their own *quantum* ALU, with qubit inputs and quantum logic gates:



The ease of use mainly comes from the ability to specify a large circuit in a single line, like:



```
exec( [ [h,s,t,x,y,] , [y,s,h,h,z] ],[1,1],[0,0])
```

But the problem of making such a meta simulator also came with some more theoretical problems. The paper addresses just a couple of these.

3. The quantum circuit model seems to precede Yao, since the modern notation is already found in Feynman's original 1981 papers. Feynman was making arguments about the underlying electronics (logic gates etc.) and never made the mistake of traeating any circuit as a "computer". Any argument about the Turing-completeness of an actual system generally dispenses with the infinite memory requirement. However, the quantum circuits the IBM Q allows us to build, for example, is not Turing complete, even with this relaxation. The solution, therefore (thought the complete proof is not given in any of the docments) is to make reference to the stored program concept. The point of all those references to Cook and Reckhow is that their abstraction is the farhtest from the Turing machine, though closest to any real machine. Beside any machine that implements each of the individual operations of their "RASP" does not even require a PDA to simulate/implement. That is to say, its language is at the lowest level of the Chomsky hierarchy.