OPEN QUANTUM ASSEMBLY LANGUAGE

NICHOLAS BOLTON

110034690

COMP 2660

27 FEBRUARY 2023

University of Windsor

LINK TO LATEXTEMPLATE:

HTTPS://WWW.IEEE.ORG/CONTENT/DAM/IEEE-ORG/IEEE/WEB/ORG/PUBS/CONFERENCE-LATEX-TEMPLATE_10-17-19.ZIP

I confirm that I will keep the content of this assignment confidential. I confirm that I have not received any unauthorized assistance in preparing for or writing this assignment. I acknowledge that a mark of 0 may be assigned for copied work

Open Quantum Assembly Language

Nicholas Bolton
COMP 2660
University of Windsor
Windsor, Canada
110034690

I. Introduction

Quantum assembly language (QASM) is analogous to assembly language (ASM) in a way that it consists of the fundamental operations that can be executed on a certain type of processor. Just as specific ASM operations can only be executed on their own specific processors, QASM operations can only be executed on quantum processors [3]. Quantum processors are a type of processor that are used for quantum computing, hence the name. Quantum computing is a growing technology that uses quantum mechanics to solve problems that are too complex for classical machines. Quantum computers have the capability for exceeding today's most powerful supercomputers.

Quantum processors are not made up the same way as classical processors. Instead of being made up of bits, they are made up of qubits. A qubit is a value that has a certain probability of being either 0, 1, or both. This is a phenomenon from quantum mechanics known as superposition. Another interesting component of quantum mechanics is entanglement, which is where quantum systems can become entangled with each other. Two systems can become entangled with each other when their global system state cannot be described as a combined state of their subsystems, to say the least. In other words, if the tensor product of the state of the subsystems cannot describe the state of the global system, the subsystems are entangled. The subsystems become literally entangled – whatever operation is applied to one correlates to the other as well. Distance has no factor into this property. This property in quantum systems translate to qubits as well. Measuring the state of one qubit gives information about another entangled qubit [2].

Just as a processor has its own assembly language (ASM), quantum processors have their own assembly languages as well. One specific language available to run on quantum processors is Open Quantum Assembly Language (OpenQASM). OpenQASM was designed for making quantum computing algorithms and applications. It presents physical logic gates and concurrent classical computations. Its purpose is similar to x86 assembly code, where it is structured as human-readable code that is converted to machine operations. Although, unlike x86 assembly code, the operands are not in the form of binary digits, but in pulses.

II. Importance

General programs in quantum computing require the handling of the classical and quantum components of such. However, the implementation of quantum computing can cut down computational times on problems exponentially. A state-of-the-art classical supercomputer that would take 10,000 years to run on an instance of a quantum circuit one million times, a quantum processor can do in about 200 seconds [1]. Because of the usage of qubits, which is further explained in Technological Details, quantum processors can use the superposition principle to perform various calculations at the same time. If one needs to consider many different cases of one problem, a standard processor will have to calculate each case individually. A quantum processor can calculate many cases at the same time.

OpenQASM's main purpose, as stated is to be an interface language that enables experiments with small depth quantum circuits. These circuits allow for the incredible computational speeds and ability to solve complex problems, which lead into optimizing the main computational-heavy topics of the world today (i.e., artificial intelligence, machine learning, security, blockchain, etc.). OpenQASM is designed to act as a low-level language, intended to be an intermediate representation. A large feature of OpenQASM is that it describes quantum computation in a quantum circuit model, analogous to classical circuit models. There are a lot of features OpenQASM includes that are relevant to what researchers and experimentalists need today. For example, researchers in circuit compiling need high-level recorded information in intermediate representations so that they can use the respective optimization and synthesis algorithms properly. Experimentalists prefer writing circuits at high-levels but need to be able to modify timing or pulse-level gate descriptions at different points in the circuit. Hardware engineers who design waveform generators and classical controllers prefer languages that are convenient to compile given hardware constraints and make structured circuits that are optimal for controllers. OpenQASM is designed to attend to all these users. OpenQASM aims at describing a large set of quantum circuits that use more than simple gates and qubits. Such consists of arbitrary classical control flow, gate modifiers, timing, and micro coded pulse implementations. Any quantum computation should be able to be described in principle using OpenQASM [4].

OpenQASM enables the integration of classical and quan-

tum computing. OpenQASM is described as “a happy blend of C and Assembly Language.” [3]. By having a way to describe quantum circuits in a classical programming language, developers can write code that includes both classical and quantum operations. This is important since most quantum algorithms require classical pre-processing and post-processing of data. Since quantum computers are very financially and physically expensive, they are not very easily accessed. Since OpenQASM can run both quantum and classical operations, one can simulate a quantum circuit on a classical computer without going through the trouble of acquiring a quantum computer [3].

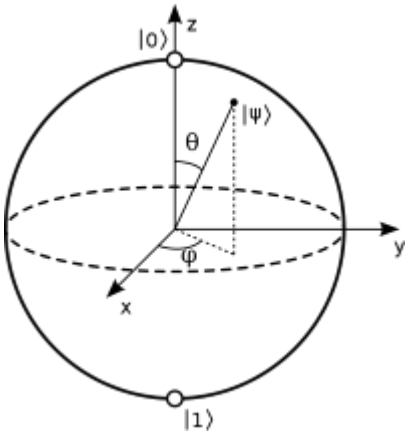

Fig. 1. Block sphere representation of a qubit state [4].

Since OpenQASM can also run multiple variations of a problem at once, users can use this to debug and refine their algorithms. Allowing for the easy modifications of quantum circuits, users can try out different variations of their algorithms and benchmark the performances. Combining this with being able to run on classical processors, one can optimize an algorithm with low-cost before running it on the expensive quantum processor.

III. Technological Details

A. Building Blocks

Unlike classical computers, which use bits, quantum computers operate using qubits. Qubits do not have a binary on/off state, but rather can exist in a state of both simultaneously. Rather than representing operations as binary code, QASM operations are implemented using microwave pulses. There is no binary code in QASM; instead, the equivalent is a sequence of qubits. The behavior of qubits is similar to that of quantum states in quantum physics, which are described using two orthogonal components - spin-up and spin-down. Qubits are represented using Dirac notation with $|0\rangle$ and $|1\rangle$, and can exist entirely in one state or partially in both. The computational basis states of qubits refer to the z-basis states, which are located at opposite points on a Bloch sphere. The Bloch sphere is commonly used to represent the state of a single qubit (as shown in Figure 1) [4].

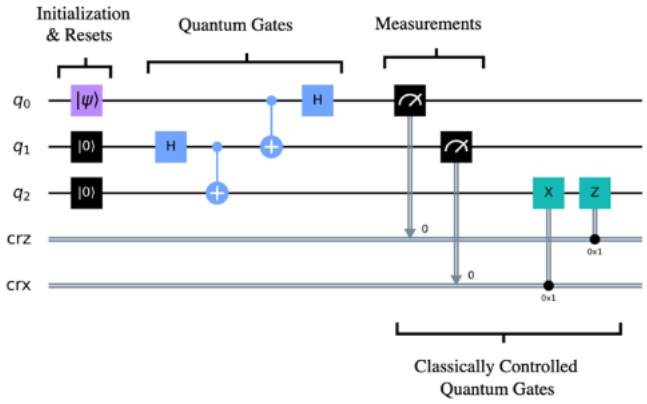

Fig. 2. Quantum teleportation algorithm circuit [5].

There are various physical methods for implementing qubits, such as using the polarization of photons, the discrete energy levels of an ion, or the spin states of an electron [4].

The next fundamental component after a qubit is a quantum circuit, which differs from classical circuits in its use of qubits instead of bits. A quantum circuit is a computational model that performs both quantum and classical operations, represented as a sequence of quantum gates, measurements, and resets. Quantum gates operate on a small number of qubits, similar to classical XOR or NOT gates. An example of a popular quantum circuit is quantum teleportation, which involves initializing and resetting qubits and bits to achieve a desired state, then manipulating the data according to the teleportation algorithm. Finally, the output is stored in bit registers [5].

B. Syntax and Structure

With the fundamental elements of quantum processors being established, the syntax and structure of OpenQASM can be examined. The syntax features elements similar to C and assembly language. The first line of an OpenQASM probram is required to declare the version in the form of `OPENQASM M.m;`, where ”M.m” specifies the version to be used. Currently, OpenQASM is available with version 3.0. Similar to C, OpenQASM ignores whitespace, separates statements using semicolons, allows for commenting with the same double-forward-slash syntax, and is case-sensitive. The only storage types in OpenQASM, similar to ASM, are registers. However, OpenQASM has two types of registers - classical and quantum registers. These are one-dimensional arrays consisting of bits and qubits, respecitlvey. They can be initialized using `creg name[size]` and `qreg name[size]`, where `name` is the identifier and `size` is the size of the array. The identifier must start with a lowercase letter and can include alphanumeric characters and underscores. Once qubit registers are declared, they are initialized to the $|0\rangle$ state [3].

There is a built-in universal gate basis, known as “CNOT + U(2)”. There is also one built-in two-qubit gate known as the controlled-NOT gate. Applying the statement

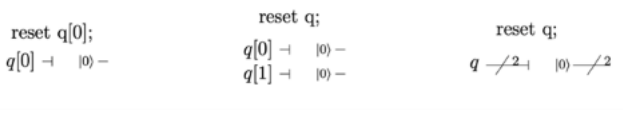

Fig. 3. Reset statement, which puts a qubit or quantum register to the $|0\rangle$ state [3].

`CX a[j], b[j];` means to apply the CNOT gate to `a` and `b`. If the qubit in `a` is one, the operation will flip the qubit stored in `b`. Other built-in gates like this have reserved uppercase keywords. All the built-in single-qubit unitary gates are parameterized as [3]:

$$U(\theta,\phi,\lambda) := \begin{pmatrix} e^{\frac{-i(\phi+\lambda)}{2}}\cos\left(\frac{\theta}{2}\right) & -e^{\frac{-i(\phi-\lambda)}{2}}\sin\left(\frac{\theta}{2}\right) \\ e^{\frac{i(\phi-\lambda)}{2}}\sin\left(\frac{\theta}{2}\right) & e^{\frac{i(\phi+\lambda)}{2}}\cos\left(\frac{\theta}{2}\right) \end{pmatrix} \quad (1)$$

When `a` is a quantum register, writing `U(theta,phi,lambda) a;` applies the gate to each index in `a`. Each parameter can be within the range $[0, 4\pi)$.

Users can also create their own gates as well, similar to how functions are created in C. They are defined in the form [3]:

```
gate name(params) qargs
{
body
}
```

Only built-in gate statements, calls to other created gates, and barrier statements can appear in the body of a gate. The statements can only refer to the symbols given in the parameter list params. The gate can be applied to any combination of qubits and quantum registers of the same size. To measure values, one uses the statement `measure qubit|qred -> bit|creg;`. This measures the qubit(s) in their basis and writes the outcomes onto the bit(s). Measurement corresponds to a projection onto one of the eigenstates of the basis and qubit(s) are available for further quantum computation immediately afterward. To reset a qubit or quantum register to the $|0\rangle$ state, one can use the statement `reset qubit|qreg;`. This corresponds to discarding the qubits before replacing them with $|0\rangle\langle 0|$ as shown in Fig. 3.

There is one quantum operation that is classically controlled: the if statement. It works just as it does in ASM - it checks a classical register if a value is contained in it, and then executes an operation based on if it is true or not. This allows for measurement outcomes to determine what quantum operations will be executed. This and other statements in OpenQASM are summarized in Tab. 1.

C. Compilation and Execution

Once an OpenQASM program is written, one needs to execute it. General quantum programs coordinate the quantum and classical parts of the computation. An important topic to keep in mind is the use of intermediate representations (IRs). An IR of a computation is something between a line of source code and machine instruction. Compilers use multiple IRs while converting and optimizing a program.

The first step in executing an OpenQASM program is compilation. The compilation phase takes place on a classical computer where input variables are not yet known, and the quantum computer is not required. The source code describing the quantum algorithm is compiled into a combined quantum and classical program that is expressed using high-level IRs.

After compilation of the program, the circuit generation phase follows. This stage operates on a classical computer, utilizing known problem parameters, and allowing for some interaction with the quantum computer. The outcome is a set of quantum circuits, which include computations of run-time parameters, branches dependent on measurements, algorithms for processing measurement results, and the creation of new basic blocks dynamically. The resulting output is presented in a quantum circuit IR format.

Following the circuit generation phase, the execution phase occurs on the quantum computer. The quantum circuit IR is transformed into a sequence of instructions that represent physical operations by a high-level controller. The resulting output is a set of processed measurement outcomes provided by the high-level controller.

Once the execution occurs, post-processing takes place on the classical computer, which is a collection of the processed results. What is outputted is the intermediate results for the final result or further computation. A diagram of the phases of execution can be seen in Fig. 4 [3].

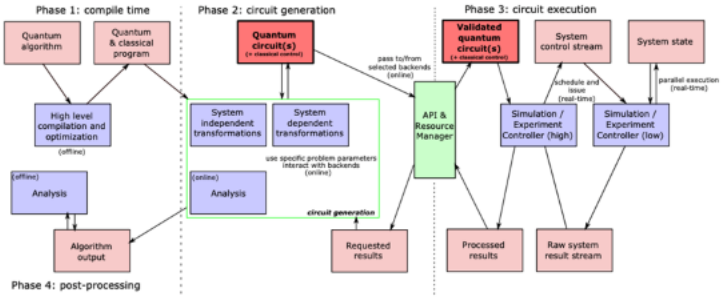

Fig. 4. "Block diagrams of processes (blue) and abstractions (red) to transform and execute a quantum algorithm... The API and Resource Manager (green) represents the gateway to backend processes for circuit execution. Dashed vertical lines separate offline, online, and real-time processes" [3].

IV. Discussion

OpenQASM's simplicity, ease-of-use, and low-level ability proves it to be a useful tool for simulating quantum-related problems. An example is Q²Chemistry, which is a quantum chemistry computation platform built using OpenQASM. With the aspirations of solving quantum chemistry problems, the use of classical computations limits becomes relevant for these high-demanding solutions. Although the scripts and APIs are written in classical languages such as Python and C++, the optimization and reconstruction of the quantum circuit to fit the architecture of specific hardware are done using

Table 1: Open QASM language statements (version 2.0)

Statement	Description	Example
`OPENQASM 2.0;`	Denotes a file in Open QASM format[a]	`OPENQASM 2.0;`
`qreg name[size];`	Declare a named register of qubits	`qreg q[5];`
`creg name[size];`	Declare a named register of bits	`creg c[5];`
`include "filename";`	Open and parse another source file	`include "qelib1.inc";`
`gate name(params) qargs { body }`	Declare a unitary gate	(see text)
`opaque name(params) qargs;`	Declare an opaque gate	(see text)
`// comment text`	Comment a line of text	`// oops!`
`U(theta,phi,lambda) qubit|qreg;`	Apply built-in single qubit gate(s)[b]	`U(pi/2,2*pi/3,0) q[0];`
`CX qubit|qreg,qubit|qreg;`	Apply built-in CNOT gate(s)	`CX q[0],q[1];`
`measure qubit|qreg -> bit|creg;`	Make measurement(s) in Z basis	`measure q -> c;`
`reset qubit|qreg;`	Prepare qubit(s) in $\|0\rangle$	`reset q[0];`
`gatename(params) qargs;`	Apply a user-defined unitary gate	`crz(pi/2) q[1],q[0];`
`if(creg==int) qop;`	Conditionally apply quantum operation	`if(c==5) CX q[0],q[1];`
`barrier qargs;`	Prevent transformations across this source line	`barrier q[0],q[1];`

[a] This must appear as the first non-comment line of the file.
[b] The parameters `theta`, `phi`, and `lambda` are given by *parameter expressions*; see text and Appendix A.

TABLE I
OPENQASM LANGUAGE STATEMENTS [3].

OpenQASM. By using OpenQASM researchers can simulate behavior of molecules and chemical reactions with more cost-efficiency [6].

OpenQASM, like classical ASM, has lots of benefits and drawbacks compared to other quantum programming languages. One of these include it being open source (hence the "Open" prefix), meaning it is accessible to anyone to use and modify. The source code can be easily found on GitHub for download and use [7]. One can also see that, like ASM, OpenQASM is a low-level language. This provides users with the ability to have greater control over quantum hardware, as opposed to other high-level quantum languages like Q#. This feature is great for the same reasons as ASM – it is easiest to optimize (quantum) circuits. This also brings the same drawback that ASM has, where it has a lack of abstraction. This makes it extremely for users who aren't experienced in OpenQASM to write effective code.

Although OpenQASM has helped progress the field of quantum computing, it still has features that are yet to be implemented. IBM (the creator of OpenQASM) lists the language features directly on its website [8]. Some features that have yet to be implemented are float, complex, const, variable constants values, and others. There is partial support in some features such as for loops, where they are supported in all aspects except for discrete sets and negative stepping.

OpenQASM does include advanced features like gate timing and classical control flow. These advances features bridge the gap between hardware description language and end-user interface effectively. The development of OpenQASM and the use of it allows for the acceleration of the research into quantum computing and brings them closer to commercial availability.

V. CONCLUSION

OpenQASM is a very powerful yet simple language that may potentially be a forefront for a standard of a quantum assembly language. The ability to refine quantum circuits dynamically, while also being compatible with other classical languages makes it very attractive to researchers in the quantum computing field. As the field in quantum computing expands, and more relevant problems arise, the use for a language that can optimize problem solving in a cost-effective way will be a great asset.

REFERENCES

[1] Arute, F., Arya, K., Babbush, R. et al (2019). Quantum supremacy using a programmable superconducting processor. Nature, 574(7779), 505–510. https://doi.org/10.1038/s41586-019-1666-5

[2] Bradben. (n.d.). Understanding quantum computing - Azure Quantum. learn.microsoft.com. https://learn.microsoft.com/en-us/azure/quantum/overview-understanding-quantum-computing

[3] Cross, A., Bishop, L., Smolin, J., Gambetta, J. (2017). Open Quantum Assembly Language. https://arxiv.org/pdf/1707.03429.pdf

[4] Cross, A., Javadi-Abhari, et al (2022). OpenQASM 3: A broader and deeper quantum assembly language. ACM Transactions on Quantum Computing. https://doi.org/10.1145/3505636

[5] Defining Quantum Circuits. (n.d.). Community.qiskit.org. https://qiskit.org/textbook/ch-algorithms/defining-quantum-circuits.html

[6] Fan, Y., Liu, J. et al (2022). Q2Chemistry: A quantum computation platform for quantum chemistry. https://doi.org/10.52396/justc-2022-0118

[7] OpenQASM. (2022, August 30). GitHub. https://github.com/openqasm/openqasm

[8] OpenQASM 3 language features. (n.d.). IBM Quantum. Retrieved February 22, 2023, from https://quantum-computing.ibm.com/services/programs/docs/runtime/manage/systems/dynamic-circuits/feature-tables

[9] Siegelwax, B. N. (2022, February 8). Quantum Assembly Language 101. Medium.

[10] Voorhoede, D. (2022). What is a qubit? Quantum Inspire. https://www.quantum-inspire.com/kbase/what-is-a-qubit/