I have created this guide for researchers, developers and students who will use QOA in practial applications, I have designed it to interact with Quantum and/or Optical systems in the same way classical risc syntax based assembly could manipulate electrons in transistors. I hope whoever reads this guide finds it useful.

Sincerely, Rayan

Base Syntax / Operations Overview:
(Not done yet, more operations might be added in the future)

INIT QUBIT n

Initializes qubit n to state $|0\rangle$.

APPLY_GATE G n

Applies a unitary gate G to qubit/mode n.

ENTANGLE n m

Entangles qubits/modes n and m.

MEASURE n

Measures qubit/mode n, collapsing the quantum state.

PHASE_SHIFT n value

Applies a phase shift to qubit/mode n by value radians.

WAIT cycles

Idles the system for a specified number of cycles.

RESET n

Resets qubit/mode n to the ground state.

LOAD n data

Loads classical or quantum data into qubit/mode n.

STORE n dest

Stores measurement result from qubit/mode n into classical memory at dest.

SWAP n m

Swaps the quantum or optical states between n and m.

ENTANGLE_BELL n m

Creates a Bell state between n and m.

PHOTON_EMIT n

Emits a photon into optical mode n.

PHOTON_DETECT n

Detects a photon in optical mode n.

APPLY_HADAMARD n

Applies a Hadamard gate on qubit/mode n.

CONTROLLED_NOT control target

Executes a controlled NOT gate.

ADD n m dest

Performs classical addition of classical data in n and m, stores result in dest.

Quantum/Optical context: Performs reversible classical addition on classical bits encoded within qubits or measurement registers, typically used in quantum arithmetic algorithms or classical control processing in hybrid systems.

SUB n m dest

Classical subtraction of data in m from n, stores in dest.

In hybrid quantum-classical systems, used to manipulate classical data derived from measurement or control registers.

AND n m dest

Logical AND of classical bits in n and m, stores in dest.

Logical operations on classical registers extracted from quantum measurements.

OR n m dest

Logical OR of classical bits in n and m, stores in dest.

XOR n m dest

Logical XOR of classical bits in n and m, stores in dest.

NOT n dest

Logical NOT of classical bit in n, stores in dest.

JUMP label

Unconditional jump to instruction labeled label.

In quantum or optical systems, this controls classical instruction flow in the classical controller managing the quantum device.

JUMP_IF_ZERO n label

Jump to label if classical register or measurement result n is zero.

JUMP_IF_ONE n label

Jump if classical register or measurement result n is one.

CALL label

Calls subroutine at label. Pushes return address onto stack.

RETURN

Returns from subroutine to the address on top of the stack.

PUSH n

Pushes classical data in register n onto the classical call stack.

POP n

Pops data from the stack into classical register n.

LOAD_MEM addr dest

Loads classical data from classical memory addr into register dest.

STORE_MEM src addr

Stores classical data from register src into classical memory addr.

ENTANGLE_MULTI n1 n2 ... nN

Creates a multi-qubit entangled state across qubits or modes n1 through nN.

This instruction facilitates generation of GHZ or cluster states, useful in advanced quantum algorithms and error correction.

APPLY_ROTATION n axis angle

Applies rotation around specified axis (X, Y, or Z) on qubit/mode n by angle radians.

RESET_ALL

Resets all qubits or modes in the system to the ground state.

PHOTON_ROUTE n src dest

Physically routes photon from source optical mode src to destination dest.

Useful in optical circuit switching and path management.

SYNC

Synchronizes classical and quantum/optical subsystems, ensuring all prior operations complete before proceeding.

ENTANGLE_SWAP n1 n2 n3 n4

Performs entanglement swapping operation among four qubits/modes, enabling long-distance entanglement distribution in quantum networks.

ERROR_CORRECT n code

Invokes quantum error correction on qubit/mode n using specified error correction code.

Applicable in quantum systems with error correction hardware or software layers.

PHOTON_COUNT n dest

Counts the number of photons detected in mode n, stores the count in classical register dest.

APPLY_DISPLACEMENT n value

Applies a displacement operation to optical mode n, shifting its phase space representation by value.

Important for continuous-variable optical quantum computing.

APPLY_SQUEEZING n value

Applies a squeezing operation to optical mode n, modifying quantum noise properties in specified quadrature.

MEASURE_PARITY n

Measures the parity (even or odd number of photons) in optical mode n, useful in certain quantum error detection schemes.

LOAD_CLASSICAL src dest

Loads classical data from external source src into classical register dest.

STORE_CLASSICAL src dest

Stores classical data from register src to external classical memory location dest.

APPLY_PHASE_FLIP n

Applies a phase-flip (Z) gate on qubit or mode n.

APPLY_BIT_FLIP n

Applies a bit-flip (X) gate on qubit or mode n.

APPLY_T_GATE n

Applies T gate (π/8 phase) on qubit or mode n.

APPLY_S_GATE n

Applies S gate (π/4 phase) on qubit or mode n.

MEASURE_IN_BASIS n basis

Measures qubit or mode n in a specified basis (X, Y, Z, or custom).

Additional Specialized QOA Instructions

(Not complete, may be more verbose in the future)

DECOHERE_PROTECT n duration

Activates a decoherence protection protocol on qubit/mode n for duration cycles. This might trigger dynamic decoupling pulses or optical error suppression to extend coherence times.

FEEDBACK_CONTROL n measurement_reg

Performs classical feedback control on qubit/mode n based on the measurement result stored in measurement_reg. Enables adaptive algorithms and error mitigation by dynamically adjusting gates.

ENTANGLE_CLUSTER n1 n2 ... nN

Generates a cluster state (multi-qubit highly entangled resource state) over qubits or modes n1 through nN, critical for measurement-based quantum computing models.

APPLY_CPHASE control target angle

Applies a controlled phase gate between control and target qubits/modes with phase angle. Allows flexible conditional phase manipulation beyond fixed CZ gates.

PHOTON_LOSS_SIMULATE n rate duration

Simulates photon loss in optical mode n at specified rate for duration cycles, useful for testing error correction and noise resilience in simulators.

APPLY_KERR_NONLINEARITY n strength duration

Applies a Kerr nonlinearity effect to optical mode n with specified strength for duration. Models photon-photon interaction essential in nonlinear optical quantum gates.

ENTANGLE_SWAP_MEASURE n1 n2 n3 n4 dest

Performs entanglement swapping involving qubits/modes n1 to n4, measures resulting states, and stores measurement outcomes in classical register dest for further processing.

TIME_DELAY n duration

Introduces a controlled time delay of duration cycles to qubit/mode n, simulating photon travel delays or temporal encoding schemes.

APPLY_QND_MEASUREMENT n dest

Performs a Quantum Non-Demolition (QND) measurement on mode/qubit n and stores the result in classical register dest. Enables measurement without collapsing the full quantum state, useful for certain error correction protocols.

ERROR_SYNDROME n code dest

Extracts the error syndrome from qubit/mode n using the specified quantum error correction code and stores the syndrome bits in classical register dest.

PHOTON_BUNCHING_CONTROL n enable

Enables or disables photon bunching control in optical mode n, a quantum optical phenomenon that can be used for entanglement verification and boson sampling experiments.

SINGLE_PHOTON_SOURCE_ON n

Turns on a single-photon source connected to mode n, enabling deterministic photon emission essential for photonic quantum computing.

SINGLE_PHOTON_SOURCE_OFF n

Turns off the single-photon source for mode n.

APPLY_LINEAR_OPTICAL_TRANSFORM matrix_addr src_modes dest_modes count

Applies a linear optical transformation defined by a matrix stored at matrix_addr to count input modes src_modes and outputs results to dest_modes.

Used to program arbitrary unitary transformations on photonic modes for boson sampling and general photonic circuit programming.

PHOTON_DETECT_COINCIDENCE n1 n2 ... nN dest

Measures photon coincidence events across modes n1 through nN, storing results in classical register dest. Crucial for entanglement verification and multi-photon interference experiments.

CONTROLLED_SWAP control target1 target2

Performs a controlled SWAP (Fredkin) gate swapping states of target1 and target2 conditional on the state of control qubit/mode.

APPLY_DISPLACEMENT_FEEDBACK n feedback_reg

Applies a displacement operation on mode n where the displacement magnitude is dynamically computed from classical register feedback_reg.

PHOTON_DETECT_WITH_THRESHOLD n threshold dest

Measures photons in mode n but only reports if detected photons exceed threshold, storing boolean result in dest.

APPLY_MULTI_QUBIT_ROTATION n1 n2 ... nN axis angles

Applies simultaneous rotations around the specified axis with given angles to qubits/modes n1 through nN.

OPTICAL_SWITCH_CONTROL n state

Controls the physical optical switch for mode n to route photons on or off path depending on state.

APPLY_FEEDFORWARD_GATE n control_reg

Applies a gate to qubit/mode n controlled by the value in classical register control_reg, enabling dynamic gate application conditioned on measurement outcomes.

APPLY_NONLINEAR_SIGMA n param

Applies a custom nonlinear optical operation with parameter param to mode n, allowing for experimental or hardware-specific nonlinear effects beyond Kerr.

MEASURE_WITH_DELAY n delay dest

Performs measurement on mode n after a programmable delay, storing the result in classical register dest. Useful for temporal multiplexing or asynchronous protocols.

PHOTON_LOSS_CORRECTION n code

Initiates error correction routines on optical mode n to mitigate photon loss errors using specified quantum error correction code.

PHOTON_EMISSION_PATTERN n pattern duration

Emits photons in mode n following a time-encoded pattern for duration cycles, enabling complex photonic pulse trains or multiplexed sources.

APPLY_SQUEEZING_FEEDBACK n feedback_reg

Applies squeezing operation on mode n with magnitude or phase modulated by classical register feedback_reg.

APPLY_PHOTON_SUBTRACTION n

Performs photon subtraction operation on optical mode n, a probabilistic process used in continuous-variable quantum computing and state engineering.

PHOTON_ADDITION n

Performs photon addition on mode n, complementary to subtraction, useful for non-Gaussian state preparation.

APPLY_MEASUREMENT_BASIS_CHANGE n basis

Dynamically changes measurement basis on mode/qubit n to basis before measurement, supporting flexible adaptive measurements.

CONTROLLED_PHASE_ROTATION control target angle

Performs a phase rotation on target qubit/mode conditioned on the control qubit/mode state by angle.

Up until this point, only functions have been mentioned, lets see what practial applications those functions can do, below is some example code of what QOA is capable of doing.

EXAMPLE 1 START:

Three qubit Quantum Fourier Transform

```
; QOA Program: Quantum Fourier Transform with Optical Readout

; Initialize Quantum Registers
QINI Q0          ; Initialize quantum register Q0
QINI Q1          ; Initialize quantum register Q1
QINI Q2          ; Initialize quantum register Q2

; Load Basis States
QLDB Q0 0        ; Load |0⟩ into Q0
QLDB Q1 1        ; Load |1⟩ into Q1
QLDB Q2 0        ; Load |0⟩ into Q2

; Apply Hadamard Transform to All Qubits
QHAD Q0
QHAD Q1
QHAD Q2

; Apply Controlled Phase Gates
QCPH Q1 Q0 pi/2    ; Apply controlled phase π/2 from Q1 to Q0
QCPH Q2 Q0 pi/4    ; Apply controlled phase π/4 from Q2 to Q0
QCPH Q2 Q1 pi/2    ; Apply controlled phase π/2 from Q2 to Q1

; Swap Qubits to Reverse Order
QSWP Q0 Q2       ; Swap Q0 and Q2

; Measure and Store Result
QMSR Q0 R0       ; Measure Q0 into classical R0
QMSR Q1 R1       ; Measure Q1 into classical R1
QMSR Q2 R2       ; Measure Q2 into classical R2

; Prepare Optical Channels
OINI O0          ; Initialize optical channel O0
OINI O1          ; Initialize optical channel O1

; Modulate Based on Quantum Results
OMOD O0 R0       ; Modulate optical O0 with result in R0
OMOD O1 R1       ; Modulate optical O1 with result in R1

; Interference and Output
OINT O0 O1 O2    ; Interfere O0 and O1, store in O2
ODSP O2          ; Display final optical signal O2

; Hybrid Control Signal Based on Quantum-Classical State
HSIG R0 R1 1     ; Trigger control if R0 AND R1 == 1

; End of Program
HALT
```

What does the above program listed do?

This QOA program performs a three qubit Quantum Fourier Transform (QFT), a fundamental algorithm in quantum computing, and translates the results into optical signals. It initializes three qubits, prepares them in superposition using Hadamard gates, applies controlled phase shifts to encode frequency components, and then reverses their order to complete the QFT. After measuring the quantum state into classical registers, the program modulates optical channels based on these measurements. It then combines the modulated signals through interference and displays the result optically. A hybrid control signal is triggered conditionally based on the quantum measurements, showcasing the integration of quantum logic with optical output.

EXAMPLE 2 START:

QOA Optical Network Switch

```
; Converts incoming optical signals into electrical ones based on routing logic

; Initialize classical registers
LOAD R0 0          ; Clear register R0 (route selector)
LOAD R1 1          ; Route control signal

; Initialize optical input channels
OPT_INIT O0        ; Optical input channel 0
OPT_INIT O1        ; Optical input channel 1
OPT_INIT O2        ; Optical input channel 2

; Initialize optical-to-electrical converters
OPT_ELEC_INIT E0    ; Electrical output from optical input
OPT_ELEC_INIT E1
OPT_ELEC_INIT E2

; Begin input signal capture
OPT_CAPTURE O0
OPT_CAPTURE O1
OPT_CAPTURE O2

; Routing logic - based on R1, select which optical signal to convert
CMP R1 0
JZ ROUTE0
CMP R1 1
JZ ROUTE1
CMP R1 2
JZ ROUTE2
JMP END

ROUTE0
OPT_MODULATE O0 E0     ; Convert O0 to E0 (electrical)
STORE E0 0x1000        ; Output to memory-mapped IO or electrical bus
JMP END
```

```
ROUTE1
OPT_MODULATE O1 E1
STORE E1 0x1000
JMP END

ROUTE2
OPT_MODULATE O2 E2
STORE E2 0x1000
JMP END

END
HALT
```

What does the above program listed do?

This QOA program implements an optical network switch that selectively converts one of several incoming optical signals into an electrical signal based on a classical routing directive. It begins by initializing optical input channels and their corresponding optical-to-electrical converters. After capturing signals from all optical inputs, the program compares the routing value stored in a classical register to determine which optical input should be processed. It then modulates the selected optical signal into an electrical form and outputs it through a memory mapped interface for further classical use. The program is designed to bridge quantum and/or **(mostly)** optical hardware with traditional digital control logic, enabling dynamic reconfiguration of signal pathways in hybrid optical electronic systems, like optical network interfaces.

EXAMPLE 3 START:

```
; Initialize registers
QINIT Q0 Q999           ; Clear and prepare 1000-qubit quantum register

; Apply Hadamard gate to create uniform superposition
QHAD Q0 Q999            ; Distribute amplitude equally

; Load target pattern from quantum memory
LOADPAT P0 QM1000 QM1999 ; Load pattern from quantum memory range into pattern
register P0

; Set iteration count (approximate sqrt of 2^1000)
LOAD R0 25              ; Set iteration counter

; Begin Grover iteration loop
GROVER_LOOP
ORACLE Q0 Q999 P0       ; Phase inversion on pattern match
DIFFUSE Q0 Q999         ; Inversion about the mean
DEC R0                  ; Decrement loop counter
JNZ R0 GROVER_LOOP      ; Loop until finished

; Measure quantum state and move to classical register
```

```
QMEAS Q0 Q999 R1        ; Store measurement in classical register R1

; Initialize optical subsystem
OPT_INIT O0             ; Optical buffer O0 for encoding measured result
ENCODEOPT R1 O0         ; Convert R1 data to modulated optical signal in O0

; Setup modulation hardware
OPT_ELEC_INIT E0        ; Prepare optical-to-electrical converter channel E0
OPT_CAPTURE O0          ; Begin capturing optical signal

; Begin routing and modulation
OPT_MODULATE O0 E0      ; Modulate optical signal O0 into electrical signal E0
STORE E0 0x2000         ; Output electrical result to memory-mapped classical I/O

; End program
HALT
```

What does the above program listed do?

This version loads a pattern for Grover's algorithm from theroetical quantum memory addresses QM1000 to QM1999 into the pattern register P0, simulating a hardware style memory fetch instead of embedding a pattern inline. This is more consistent with scalable systems and architectures where quantum memory modules store pre encoded targets or configurations.

EXAMPLE 4 START:

```
; Initialize registers
QINIT Q0 Q999           ; Clear 1000 qubit quantum register for computation
QINIT QTEMP0 QTEMP999   ; Temporary registers for intermediate quantum states
QINIT QRES0 QRES255     ; Result register to store period/output

; Load target RSA modulus from quantum memory
LOADQM M0 QM2000 QM20255  ; Load 2048-bit modulus from quantum memory into M0

; Pick a random coprime 'a' < N (modulus), hardcoded or loaded
LOADQM A0 QM3000 QM30255  ; Load coprime value into A0

; Compute modular exponentiation of A0^x mod N using quantum circuits
QMOD_EXP A0 M0 Q0 QTEMP0  ; Modular exponentiation circuit initialized

; Perform Quantum Fourier Transform to extract period
QFT Q0 Q999             ; Apply QFT over result register

; Measure Q0 into classical register to extract period guess
QMEAS Q0 Q999 R0        ; Measured guess of the period

; Perform classical post-processing (GCD etc.) not shown here
; Would typically be handled after quantum step
; For completeness:
; CLASSICAL_GCD A0 R0 M0 -> candidate factors
```

```
; Final quantum cleanup (optional)
QRESET Q0 Q999          ; Zero out quantum registers
QRESET QTEMP0 QTEMP999
QRESET QRES0 QRES255

HALT
```

What does the above program listed do?

This QOA code represents a low level quantum program to factor an RSA modulus using Shor's Algorithm. It loads both the modulus and a chosen base value from quantum memory, performs modular exponentiation, applies a Quantum Fourier Transform to find the periodicity, and measures the result to obtain the period, which can be used to derive the prime factors classically.

The entire execution takes place exclusively on a quantum computer, with no classical/optical or external data transfer until after the quantum measurement. The classical GCD and modular checks would happen post-measurement, either through a quantum classical hybrid interface or deferred to another QOA sequence.

This is all for now. QOA is still being modeled and developed solely by me, and any practical applications would not be relevant until optical and quantum systems become more commonplace. As these technologies mature and gain wider adoption, QOA aims to provide a unified low-level assembly language capable of efficiently programming and controlling purely optical, purely quantum, or hybrid quantum-optical computing platforms. Until then, QOA remains my theoretical passion project focused on laying the groundwork for future advancements in these future fields of computing