QUANTUMPULSE™ QP-2A

QP-2A REFERENCE MANUAL

SYSTEM OVERVIEW

The QuantumPulse™ QP-2A is an exciting new state-of-the-art computing platform! The system comprises of three slots that can be filled with AstroWave™ programmable chips of various sizes. Each chip operates independently but perfectly synchronized due to our patented QuantumPulse technology. Though each chip is independent, we all know the real work gets done by collaboration! Note that for quantum observability reasons, chips cannot make decisions based on their own state but can only respond to the state broadcast by other chips.

4. DEBUGGING

The QuantumPulse™ Visualization Interface allows you to step cycle-by-cycle and will highlight which line of code will be executed next in each chip, will show the current value of all registers, radios (both the values being broadcast from a given chip as well as the combined value on the channel that a chip might read), and other internal state of any chip.

4.1. REWIND TIME

Thanks to proprietary hardware, which may or may not involve a fixed particle feedback loop, the Visualization Interface can, at any time, observe any state that led to the current moment in time. This feature can be invoked even after an error has terminated a program, in order to "rewind time" to see what events led to the error. For your own safety, the Visualization Interface does NOT allow you to observe any events from before the start of program execution.

4.2. BREAKPOINTS

Prefixing any instruction with `!` will cause the Visualization Interface to pause execution immediately before that instruction is executed. For example, in the following program, execution will pause whenever a non-zero value is read from `CH1`, immediately before outputting `99`:

```
loop: JEZ CH1 loop
!MOV OUTPUT 99
```

Continuing this example, after pausing on the breakpoint, one could use the Rewind Time feature to look back one cycle to see exactly what non-zero value was broadcast on `CH1` before the conditional jump was executed.

5. TRAINING EXERCISES

The QuantumPulse™ Visualization Interface contains a series of training exercises that will help you master your craft! Every exercise contains a description of the problem you need to solve, and shows you a set of inputs and expected outputs that you need to match. To pass an exercise, your solution needs to solve the provided input/output as well as 2 or more input/output sets which will automatically run after you've solved the first ones. It does not matter what your solution does after writing the final correct output value for a given input/output set.

When comparing your solutions to those of others, we recommend looking at 3 important metrics:

* **Median Cycles** - If a solution takes a different amount of cycles on different input sets, then look at the median across all input sets
* **Lines of Code** (**LOC**) - How many code instructions there are (ignores blank lines, comments, lines with just labels, etc)
* **Cost** - The combined cost for all chips used in creating a solution

Note: Many input sets may have many (potentially irrelevant) limitations not specified in the problem description (e.g. only positive inputs, inputs are less than 100, sorted inputs). These limitations are safe to ignore, but can sometimes be used to improve your solution's metrics. Any such limitations should be fairly clear by browsing the first 3 input/output sets.

Completing all of the included training exercises will grant you access to the exclusive QuantumPulse™ BBS, where even more challenges await. You will also be the first to be shipped new chips to further unlock the potential of your QuantumPulse™ computing device! New chips will come with appendices to supplement this document.

6. CODING CONVENIENCE

Are your `O` and `U` keys in danger of wearing out? At the risk of your code looking booooring, you can pass a numerical constant to `NOP` to specify the number of cycles to wait, or a 3rd operand to `PLS` for the same.

Example: `NOP 3` is equivalent to `NOOOP`

Example: `PLS CH1 7 2` is equivalent to `PUULS CH1 7`

Prefer everything to line up nicely and be 3 letters wide everywhere? Good news, you can abbreviate `OUTPUT` as `OUT` and `INPUT` as `INP` (or, for the especially keystroke-frugal, `IN`).
Hint: Using only 2-letter labels will line up your labels as well!

Example: `MOV OUT INP` is equivalent to `MOV OUTPUT INPUT`

APPENDICES

A. CHIP SUPPLEMENT: OSCILLLILATOR

CHIP NAME: OSCILLLILATOR

MANUFACTURER: KosmosTek®

CONFIGURABLE FEATURES:

* 12 read-only cells
* REPEAT count
* Output channel

SPECIFICATION:

The Oscillilator broadcasts a value from a cell every cycle to its **output channel**, starting with the first cell, and repeating it for **repeat** cycles, then moving on to the next cell until it reaches the final used cell, at which point it starts outputting from the first cell again.

SCORING:

When used in a competitive challenge, each used memory cell counts as **1 LOC**.

B. CHIP SUPPLEMENT: ROM

CHIP NAME: ROM-12

MANUFACTURER: KosmosTek®

CONFIGURABLE FEATURES:

* 12 read-only cells
* Read index channel
* Read value channel

SPECIFICATION:

The ROM reads an index from the **read index** channel, and then on the next cycle broadcasts the value stored at that index to its **read value** channel. If the index is less than or equal to zero, nothing is broadcast. If the index exceeds the number of used cells, the ROM wraps around and broadcasts earlier values. In general, this chip has the same timing characteristics as running the following repeatedly (if an array index operation was generally supported): `MOV CH#OUT DATA[CH#READIDX]`

Timing example:

AW0401 Chip	ROM-12, READ:ch1, OUT:ch2 data: 5 6 7
`MOV CH1 3 ; ask idx 3`	`; broadcasting nothing`
`MOV OUTPUT CH2 ; outputs 0`	`MOV CH2 DATA[CH1] ; reads idx`
`MOV OUTPUT CH2 ; outputs 7`	`; now broadcasting a 7`

Note that on the first cycle, the ROM reads a `0` from the index, so does not broadcast. On the second cycle, the ROM reads a `3` from the index and will start broadcasting the data from that index, and on that same cycle the AW0401 chip still reads nothing being broadcast. On the third cycle, the AW0401 chip finally reads the requested value. This delay between requesting a value from the ROM and being able to read it is necessary to prevent potential quantum paradoxes from damaging your reality.

SCORING:

When used in a competitive challenge, each used memory cell counts as **1 LOC**.

C. DATA MESS COORDINATES

Supplemental data received as a BBS attachment.

ID	Y	X
VB	35	-121
YPG	33	-114
CL	36	-118
ED	35	-118
Replacement		
	39	-77

D. CHIP SUPPLEMENT: RAM

CHIP NAME: RAM-20

MANUFACTURER: KosmosTek®

CONFIGURABLE FEATURES:

* Write index channel
* Write value channel
* Read index channel
* Read output channel

SPECIFICATION:

* 20 ephemeral memory cells

The RAM-20 chip reads an index from the **write index** channel, and if it is greater than 0, it replaces the value in the memory cell at that index with the value read from the **write value** channel. At the same time, the RAM-20 chip reads an index from the **read index** channel and then on the next cycle will broadcast the value stored at that index to its **read value** channel, behaving similarly to the **ROM-12**.

Note that whenever the **write index** channel has a positive index, a new value will be written to a memory cell, so if you wish to stop writing to memory, first stop broadcasting to the **write index** channel before you stop broadcasting to the **write value** channel.

SCORING:

When used in a competitive challenge, the RAM-20 chip counts as **1 LOC**.

E. INTERACTIVE PUZZLES

With the latest hardware supplement, your **QP-2A** device now allows interactive challenges, where the inputs read by your program are not fixed, but instead dependent upon what the program outputs. Essentially, your program can make wrong guesses as to the answer and will get feedback if it is correct or not. Being correct will end the test and move on to the next one. Some interactive puzzles have hundreds of test cases, to eliminate any advantage gained from previous runs.

Note that the **INPUT** and **OUTPUT** shown in the Visualization Interface while you are developing your solution is only an example, the actual values will depend on what your program does.

SCORING:

Any wrong guess by your program adds a cycles penalty to your score.

F. CHIP SUPPLEMENT: PLOTTER

CHIP NAME: PLOTTER-144

MANUFACTURER: Kostronika®

CONFIGURABLE FEATURES:

* COLUMN channel
* ROW channel
* READ value channel
* WRITE value channel

SPECIFICATION:

* 144 LEDs

This exciting new chip from the flagship chip maker in Kostovia, the **Plotter** opens up a whole new realm of possibilities! Whenever it reads a positive value from the **write value** channel, it turns on the LED corresponding to the current **row** and **column**. If a negative value is read, it turns off the corresponding LED. Simultaneously, it also broadcasts a **1** on the **read value** channel if the corresponding LED is currently lit. Note that if the **row** and **column** address anything out of range, no LEDs will be changed, and no value will be broadcast on the **read value** channel.

SCORING:

For challenges that require a particular Plotter output, the challenge is considered successful if the image displayed on the Plotter matches the requirements at any time.

1. CHIP OPTIONS

AstroWave™ programmable chips all share a number of common properties, and only vary in how many of each feature they have. All chips have:

* 1 **accumulator** (ACC) register that can be used for temporary storage as well as incremented and decremented
* 4 or more **lines of code** each of which can contain a label, an operation, and a comment
* 1 or more **radios** which are automatically enabled when you send data to a specific channel

Your **QP-2A** starter kit comes equipped with a supply of AstroWave™ chips with the following capacities:

* **AW0401**: 4 **lines of code** and 1 **radio**
* **AW0903**: 9 **lines of code** and 3 **radios**
* **AW1605**: 16 **lines of code** and 5 **radios**

You box may also contain some damaged chips labeled **B0201** which do **not** have a working **accumulator** and can handle at most **2 lines of code**. Feel free to ignore these, they're probably not useful.

We also look forward to shipping you many new chips which our partners are developing as part of our exclusive subscription service! By purchasing your QuantumPulse™ device you have been automatically subscribed and will be billed a monthly service charge unless you opt-out by sending a request in writing to this address:

2. COMMUNICATION CHANNELS

Chips can communicate with each other by using one of their radios to broadcast on one of limitless communications channels [1]. Any time a chip writes a value to a channel, a radio is allocated and the chip continues to broadcast the value until the chip writes a different value, or a **0** to disable broadcasting.

Notes:

* If a chip is broadcasting to specific channel, it cannot read from that channel in any way until after it has stopped broadcasting (by writing a **0** to that channel)
* If two or more chips are broadcasting on the same channel, any receiver will read their combined (summed) value
* Though channels are theoretically limitless, the QuantumPulse™ Visualization Interface will only show you the values being broadcast on channels CH1 through CH13
* AstroWave chip execution is synchronized at faster than light speed [1]! If one chip reads from a channel on the same cycle that another chip starts broadcasting or changes it's value, the reading chip will receive the value that was being broadcast on the previous cycle. Note: Our Quantum Innovations department assures us of the inaccuracy of our competitors' reductive claims that "reading from a radio channel obtains the value being broadcast on that cycle, and writing to a channel simply causes the written value to be transmitted starting on the following cycle."

[1] This statement is classified by the Truth in Advertising Act of 1972 as advertising-truth.

(this page unintentionally left blank)

3. CODE EXECUTION

Chips execute their operations in sequential order, and execution automatically returns to the beginning after executing the final instruction. All operations take exactly one cycle unless specified otherwise.

3.1. ANATOMY OF A LINE OF CODE

Example: `foo: MOV ACC INPUT ; bar!`

* `foo:` is a label, used as a reference by JUMP instructions
* `MOV` is the instruction (see below)
* `ACC INPUT` are the operands to the instruction (specifying registers or values, varies by instruction)
* `; bar!` is a comment (ignored by the interpreter)

3.2. REGISTERS

3.2.1. ACC

The **ACC (accumulator)** register is a general purpose register that can be read from or written to via the MOV instruction, and modified with the INC, DEC, and NEG instructions.

3.2.2. INPUT

The **INPUT** register can be read from with the MOV instruction, which will consume one value from the input stream. If two or more chips read from **INPUT** at exactly the same time they will all receive the same value.

3.2.3. OUTPUT

The **OUTPUT** register can be written to with the MOV instruction. Multiple chips writing different values to **OUTPUT** at the same time is not allowed.

3.2.4. RADIO CHANNELS CH1...CHX

Channels can be written to or read from with the MOV instruction, as well as used as a source of comparison for the **JUMP** family of instructions

3.2.5. NIL

Writing to the **NIL** register does nothing. Reading from the **NIL** register is the same as using the literal number **0**.

3.2.6. ACCEPTED VALUES

All values in registers will be clamped between negative and positive Biblical Infinity (seventy times seven).

3.3. INSTRUCTIONS

3.3.1. MOV DESTINATION SOURCE

Reads from **source** and writes the read value to **destination**. The source can be any readable register, or a literal number. The destination must be a writable register.

MOV Examples

```
MOV ACC INPUT
MOV OUTPUT ACC
MOV CH7 99
```

3.3.2. DEC AND INC

These decrement or increment the value stored in the **accumulator (ACC)** register.

3.3.3. NEG

This negates the value stored in the **accumulator (ACC)** register (multiplies it by negative 1).

3.3.4. NOP, NOOP, NOOOP, ETC

This does nothing for as many cycles as there are Os.

3.3.5. DLA, DLAA, DLAAA, ETC

This pseudo-instruction delays initializing the chip and running any code for as many cycles as there are As, and is ignored after the initial delay. A DLA statement must be the first thing in a chip.

3.3.6 JMP LABEL

Moves execution to the first instruction after the specified label instead of continuing to the next instruction.

3.3.7. JUMP FAMILY - JLZ/JGZ/JEZ/JNZ - J*Z CH# LABEL

Compares the value read from the specified channel against 0 and jumps to the specified label if the comparison succeeds.

* JLZ - Jump if less than 0
* JLE - Jump if less than or equal to 0
* JGZ - Jump if greater than 0
* JGE - Jump if greater than or equal to 0
* JEZ - Jump if equal to 0
* JNZ - Jump if not equal to 0

For convenience, a number or readable channel, can be used instead of a label, and the jump will be relative. For example `JMP -1` will jump to the preceding line and `JMP CH1` will jump a number of lines based on what is read from **CH1**.

Relative jumps consider only lines of code with instructions, ignoring comments, blank lines, and the `DLA` pseudo-instruction, wrapping around if going past the first or last instruction.

Note that for quantum observability reasons, chips cannot make decisions based on their own state, so can only compare the values broadcast by other chips.

3.3.8. PULSE - PLS DESTINATION SOURCE, PULS, PUULS, ETC

A composite instruction to do a MOV, optional NOP (for as many cycles as there are Us), and then a clearing MOV:

```
MOV destination source
NOP/NOOP/etc
MOV destination 0
```

Example: PLS CH1 INPUT is equivalent to:

```
MOV CH1 INPUT
MOV CH1 0
```

Example: PUULS CH3 7 is equivalent to:

```
MOV CH3 7
NOOP
MOV CH3 0
```

3.3.9. WAIT CH#

This is an alias for JEZ CH# 0, which will continue executing the same instruction until the specified channel reads a non-zero value. This can be useful in synchronization when combined with PLS.

Synchronization example:

Chip 1

```
NOOOOOP       ; do some work
PLS CH1 1     ; signal
MOV ACC INPUT ; in sync
```

Chip 2

```
WAIT CH1      ; wait for signal
MOV ACC INPUT ; in sync
```