

# AM17: The Choice Ship

*Author*  
Adam Megacz

*Contributors*  
Adam Megacz  
Ivan Sutherland

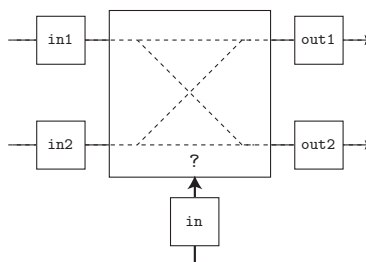
February 22, 2007

**Abstract:**

This memo is not official in any way. It should be considered a “first attempt” at dealing with conditionals, and is likely to be revised significantly before FleetTwo.

This memo introduces the *Choice ship*, a mechanism for implementing conditional behaviors in Fleet. The *Choice ship* takes input on one or both of two data input BenkoBoxes and produces output on one or both of two data output BenkoBoxes. Which input values are transferred to which output ports is determined by a (third) selector input. The selector input has multiple destinations, each of which corresponds to a different test to apply to the selector value.

The diagram below shows the five BenkoBoxes on a *Choice ship*, and the possible paths which data can take from the ship’s in ports to its out ports.



The Choice Ship

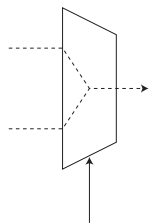
# 1 Motivation

Most conditional behaviors in Fleet can be cast as a choice between two data values.

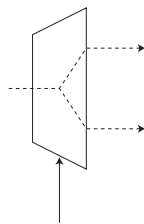
For example, a program construct which might be compiled as a “conditional branch” in a conventional processor like MIPS would likely be compiled into a choice between two code bag descriptors in Fleet. The chosen code bag descriptor would then be sent to the fetch unit.

Alternatively, a programmer might want to construct a circular datapath out of standing moves, for example in a divisibility tester. In such a configuration, one of the elements in the path would be a subtraction element (to subtract the divisor from the remainder) and the others would be responsible for determining the sign of the remainder. If the result is zero or negative, the values should be diverted out of the loop; otherwise they should cycle around again.

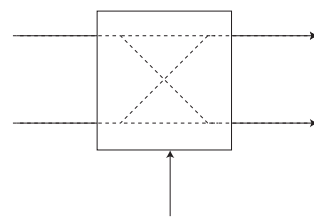
The previous two examples illustrate two kinds of choice. In the first case, one of two inputs is chosen, and the chosen input is sent to a predetermined output. In the second case, a predetermined input is taken, and sent to one of two different outputs. A third case is possible which generalizes these two, taking two inputs and possibly swapping them before outputting both:



Multiplex (mux)



DeMultiplex (deMux)



Swap (swap)

The Choice ship is capable of performing all three of these functions.

## 2 Implementation

The bottom BenkoBox, `in`, has quite a large number of destinations, listed below:

```

in.swapIfZero      in.muxIfZero      in.deMuxIfZero
in.swapIfPositive  in.muxIfPositive  in.deMuxIfPositive
in.swapIfNegative  in.muxIfNegative  in.deMuxIfNegative

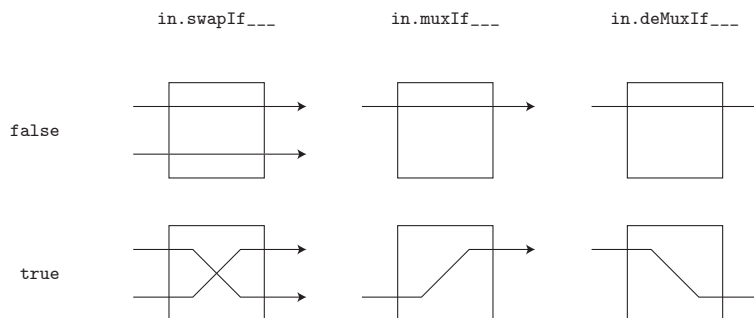
in.swapIfNonZero   in.muxIfNonZero   in.deMuxIfNonZero
in.swapIfNonPositive in.muxIfNonPositive in.deMuxIfNonPositive
in.swapIfNonNegative in.muxIfNonNegative in.deMuxIfNonNegative

```

Depending on the *value* and the *destination* at which it arrives, the Choice ship determines the *condition value*, which is either true or false, according to the table below:

	< 0	= 0	> 0
<code>in.____IfZero</code>	false	true	false
<code>in.____IfNonZero</code>	true	false	true
<code>in.____IfPositive</code>	false	false	true
<code>in.____IfNonPositive</code>	true	true	false
<code>in.____IfNegative</code>	true	false	false
<code>in.____IfNonNegative</code>	false	true	true

Finally, based on the condition value and the destination at which the value arrived, the ship takes *action* on its inputs. The next diagram summarizes the three possible actions (swap, mux, and demux) and their effect on the data when the condition value is either true or false.



Conditions and their Data Paths

As a rule of thumb, when the condition is `false`, data always flows “straight through” (from an input to its corresponding output). This makes the conditions easier to remember.

### 3 Notes

Note that data is only taken from inboxes if it will be sent via an outbox. In particular, the `in.muxIf___` and `in.deMuxIf___` forms will consume data from only *one* data input for each selection value.

Finally, note that the `Choice` ship never waits for data which is not required. For example, if the programmer sends a value to `in1` and a selection value arrives such that the action is `mux` and the condition value is `false`, the `Choice` ship is guaranteed to proceed *even if no value ever arrives at in2*.