

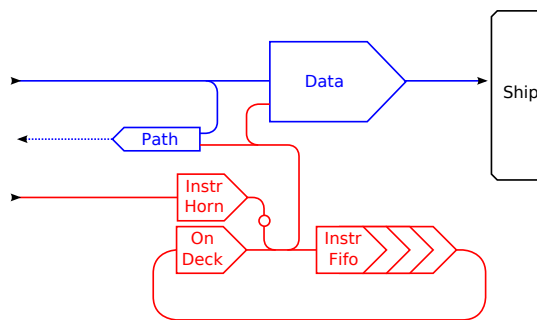
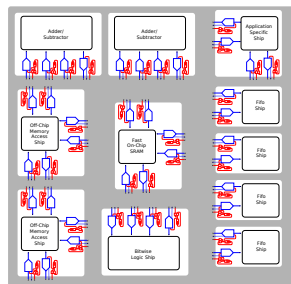
The FleetTwo Dock

March 23, 2008

Abstract

Changes:

- 23-Mar added "if its predicate is true" to repeat count
- added note that red wires do not contact ships
- changed name of flags instruction to setFlags
- removed black dot from diagrams
- changed 0L (Outer Loop participant) to 0S (One Shot) and inverted polarity
- indicated that the death of the tail instruction is what causes the hatch to be unsealed
- indicated that only send instructions which wait for data are torpedoable
- added section "Torpedo Details"
- removed torpedo instruction
- 12-Mar renamed loop+repeat to outer+inner (not in red)
- renamed Z flag to L flag (not in red)
- rewrote "inner and outer loops" section
- updated all diagrams
- 7-Mar Moved address bits to the LSB-side of a 37-bit instruction
- Added *micro-instruction* and *composite instruction* terms
- Removed the DL field, added decrement mode to loop
- Created the Hold field
- Changed how ReLooping works
- Removed clog, unclog, interrupt, and massacre



1 Overview of Fleet

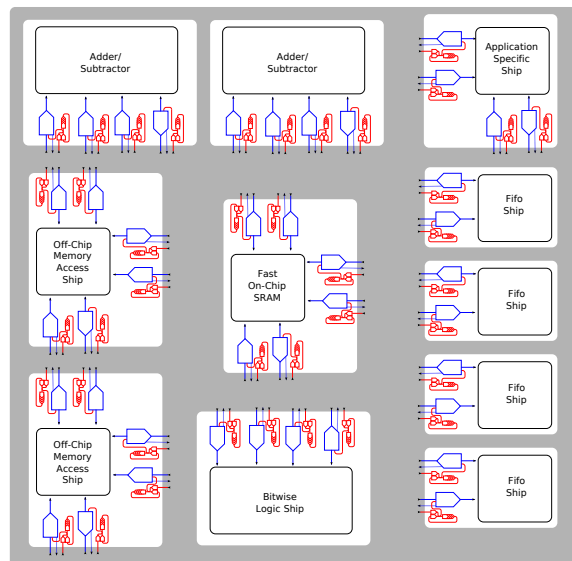
A Fleet processor consists of a *switch fabric* with several functional units called *ships* connected to it. At each connection between a ship and the switch fabric lies a programmable element known as the *dock*.

A *path* specifies a route through the switch fabric from a particular *source* to a particular *destination*. The combination of a path and a single word *payload* is called a *packet*. The switch fabric carries packets from their sources to their destinations. Each dock has two destinations: one for *instructions* and one for *data*. A Fleet is programmed by depositing packets into the switch fabric; these packets' paths lead them to the instruction destinations of the docks.

When a packet arrives at the instruction destination of a dock, it is enqueued for execution. Before the instruction executes, it may cause the dock to wait for a packet to arrive at the dock's data destination or for a value to be presented by the ship. It may present a data value to the ship or transmit it for transmission to some other destination.

When an instruction sends a packet into the switch fabric, it may specify that the payload of the packet is irrelevant. Such packets are known as *tokens*, and consume less energy than data packets. From a programmer's perspective, a token packet is indistinguishable from a data packet with a unknown payload.

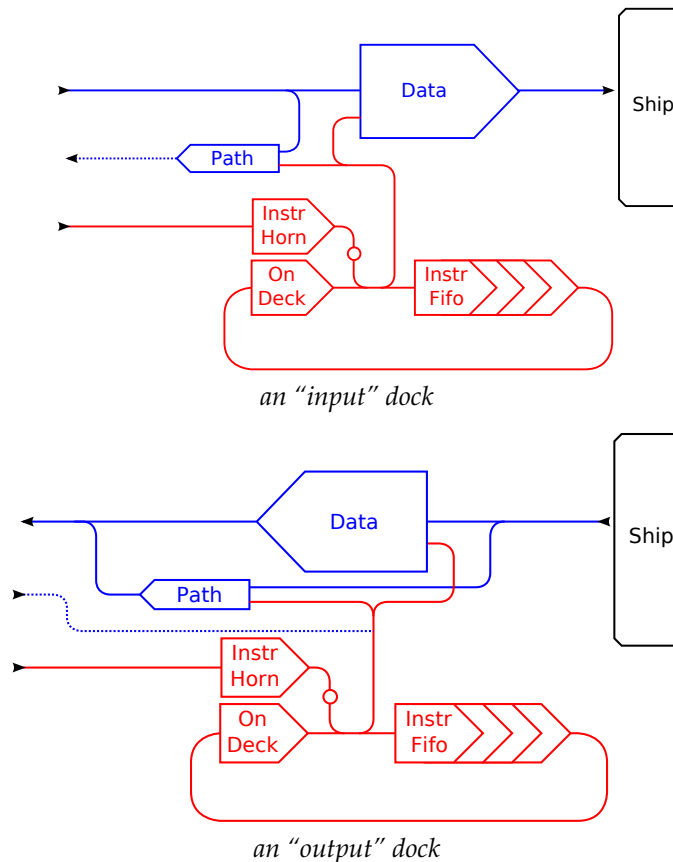
In the diagram below, the red wires carry instructions and the blue wires carry data; the switch fabric (gray area) carries both. Notice that the red (instruction) wires do not contact the ships.



Overview of a Fleet processor; gray shading represents a packet-switched network fabric; blue lines carry data, red lines carry instructions.

2 The FleetTwo Pump

The diagram below represents a *programmer's* conceptual view of the interface between ships and the switch fabric. Actual implementation circuitry may differ substantially. Sources and destinations that can send and receive only tokens – not data items – are drawn as dashed lines.



The term *port* refers to an interface to the ship, the *dock* connecting it to the switch fabric, and the corresponding sources and destinations on the switch fabric.

Each dock consists of a *data latch*, which is as wide as a single machine word and a *pump*, which is a circular fifo of instruction-width latches. The values in the pump control the data latch.

Note that the pump in each dock has a destination of its own; this is the *instruction destination* mentioned in the previous section. Note that unlike all other destinations, there is no buffering fifo guarding this one. The size of these fifos are exposed to the software programmer so he can avoid deadlock.

3 Instructions

In order to cause an instruction to execute, the programmer must first cause that instruction word to arrive in the data latch of some output dock. For example, this might be the “data read” output dock of the memory access ship or the output of a fifo ship. Once an instruction has arrived at this output dock, it is *dispatched* by sending it to the *instruction port* of the dock at which it is to execute.

Each instruction is 26 bits long, which makes it possible for an instruction and an 11-bit path to fit in a single word of memory. This path is the path from the *dispatching* dock to the *executing* dock.



Note: the instruction encodings below are simply “something to shoot at” and a sanity check to make sure we haven’t overrun our bit budget. The final instruction encodings will probably be different.

All instruction words have the following format:



Each instruction word is called a *micro instruction*. Collections of one or more micro instruction are known as *composite instructions*.

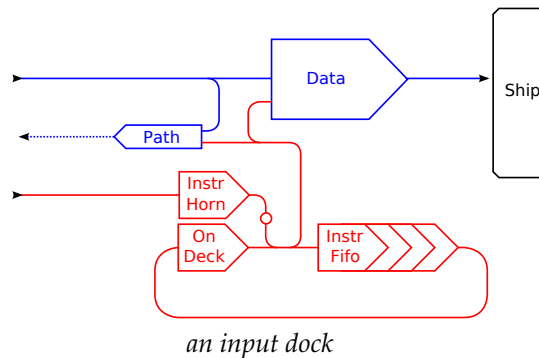
The A bit stands for Armor¹. The OS (“One Shot”) bit indicates whether or not this instruction is part of an outer loop. Both of the preceding bits are explained in the next section.

The abbreviation P stands for *predicate*; this is a two-bit code that indicates if the instruction should be executed or ignored.

¹this is to be pronounced with a Boston accent (“AAHH-mir”)

3.1 Life Cycle of an Instruction

The diagram below shows an input dock for purposes of illustration (behavior at an output dock is identical).



Note the circle on the path between "instr horn" and "instr fifo"; this is known as "the hatch". The hatch has two states: sealed and unsealed. When the machine powers up, the hatch is unsealed; it is sealed by the tail instruction and unsealed as described below.

When an instruction arrives at the instruction horn, it waits there until the hatch is in the unsealed state. The instruction then enters the instruction fifo.

When an instruction emerges from the instruction fifo, it arrives at the "on deck" stage. **At this point, exactly one of the following cases should apply:**

- If the instruction is a `setOuter` instruction with `OS=1`, it executes immediately and then retires.
- otherwise, if the instruction is a `tail` instruction and `OLC=0`, it unseals the hatch and sets the *inner* loop counter `ILC` to zero.
- otherwise, if the outer loop counter `OLC` is zero, the instruction retires immediately and no further action is taken.
- otherwise, if the instruction is a one-shot instruction `OS=1`, the instruction executes zero or more times and then retires.
- otherwise the instruction waits for the hatch to be sealed and for the first stage of the instruction fifo to be empty. When this is the case, the instruction executes zero or more times, filling the first stage of the instruction fifo with a copy of itself during the first execution (or instead of the first execution if zero executions are desired).

The number of times an instruction executes is determined by its predicate (if false, it executes zero times). If the instruction is a `send` instruction and its predicate is true, it will execute a number of times equal to the value in `ILC`, and will leave `ILC=1` after it has finished executing.

In the last two cases, if the instruction is a **torpedoable instruction (see below)**, each execution attempt will be arbitrated against the arrival of a torpedo. If the torpedo wins, the *outer* loop counter is set to zero and this process terminates immediately.

3.1.1 Torpedo Details

There is a small fifo (not shown) before the latch marked “Instruction Horn”; after the tail instruction seals the hatch, any subsequent instructions will queue up in this fifo until the hatch is unsealed. This is typically used as storage for a “loop epilogue” – a sequence of instructions to be executed after a torpedo arrives.

Each dock has a fourth connection to the switch fabric (not shown), called its *torpedo destination*. Anything (even a token) sent to this destination is treated as a torpedo. Note that because this is a distinct destination, instructions or data queued up in the other destination fifos will not prevent a torpedo from occurring.

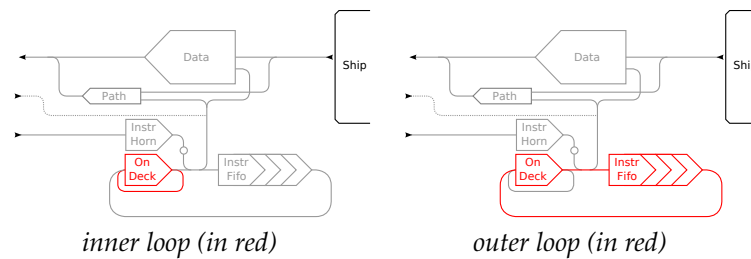
An instruction is *torpedoable* if all of the following are true:

- It is a send instruction
- Its predicate is true
- It waits for data (from the ship or the switch fabric) or tokens
- Its Armor bit is cleared

When a data item or token arrives at the torpedo destination, it lies there in wait until On Deck holds a torpedoable instruction. Once this is the case, the torpedo causes the outer loop counter to be set to zero, and then the torpedo goes away. **Important note:** it is unspecified whether the torpedoed instruction is requeued or not; this may or may not occur, nondeterministically. It is the programmer’s responsibility to ensure that the program behaves the same whether this happens or not.

3.2 Inner and Outer Loops

Using the mechanisms described above, a programmer can perform two types of loops: *inner* loops of only one micro-instruction and *outer* loops of multiple micro-instructions. Inner loops may be nested within an outer loop, but no other nesting of loops is allowed. The paths used by inner loops and outer loops are shown below:



Each type of loop has a counter associated with it: the ILC counter for inner loops and the OLC counter for outer loops. The inner loop counter applies only to send instructions; all other instructions ignore the inner loop counter. When a send instruction reaches the on deck position, **if its predicate is true** it will execute at least once; the number of times it executes after that is determined by the inner loop counter.

The outer loop counter applies to all instructions *except* the instruction `setOuter` with `OS=1`, because such instructions are needed to reset the outer loop counter after it becomes zero. However, `setOuter` with `OS=0` is useful for resetting the loop counter in the middle of the execution of a loop.

3.3 Flags

The pump has four flags: A, B, L, and S. Of these four, only the first two may be modified directly by instructions.

- The A and B flags are general-purpose flags which may be set and cleared by the programmer.
- The L flag, known as the *last* flag, is set whenever the value in the outer counter (OLC) is one, indicating that the dock is in the midst of the last iteration of an outer loop. This flag can be used to perform certain operations (such as sending a completion token) only on the last iteration of an outer loop.
- The S flag, known as the *summary* flag. Its value is determined by the ship, but unless stated otherwise, it should be assumed that whenever the 37th bit of the data (D) latch is loaded, that same bit is also loaded into the S flag. This lets the ship make decisions based on whether or not the top bit of the data latch is set; if two's complement numbers are in use, this will indicate whether or not the latched value is negative.

Many instruction fields are specified as two-bit *predicates*. These fields contain one of four values, indicating if an action should be taken unconditionally or conditionally on one of the A or B flags:

- 00: if A is set
- 10: if B is set

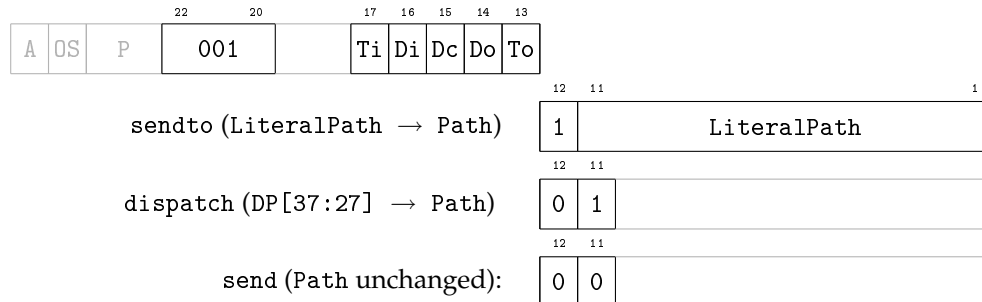
- 01: if L is set (OLC=1)
- 11: always

4 Instructions

Here is a list of the instructions supported by the dock:

```
send (variants: sendto, dispatch)
data (variants: datahi, datao)
setFlags
setInner
setOuter
tail
takeOuterLoopCounter
takeInnerLoopCounter
```

The last two instructions are strictly for debugging purposes.

4.1 send (variants: sendto, dispatch)

- Ti - Token Input: wait for the token predecessor to be full and drain it.
- Di - Data Input: wait for the data predecessor to be full and drain it.
- Dc - Data Capture: pulse the data latch.
- Do - Data Output: fill the data successor.
- To - Token Output: fill the token successor.

The data successor and token successor must both be empty in order for a send instruction to attempt execution.

The inner loop counter can hold a number $0 \dots MAX$ or a special value ∞ . If ILC is nonzero after execution of a send instruction, the instruction will execute again, and ILC will be latched with $(ILC = \infty ? \infty : \max(ILC - 1, 0))$. When the inner loop counter reaches zero, the instruction ceases executing.

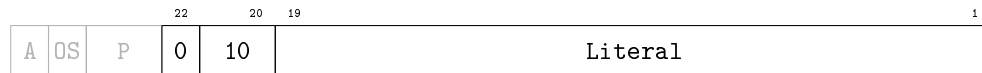
4.2 data, datahi, datalo

These instructions load part or all of the data latch (D).

datahi: Literal[18:1] → D[37:20] (and Literal[18] → S)



datalo: Literal[19:1] → D[19:1]

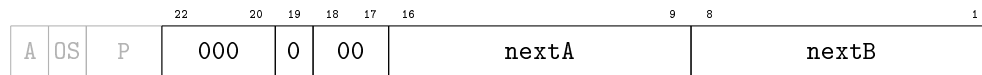


data:

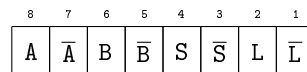


sel	D[37:20]	D[19:1]
00	Literal[18:1]	all 0
01	Literal[18:1]	all 1
10	all 0	Literal[19:1]
11	all 1	Literal[19:1]

4.3 setFlags



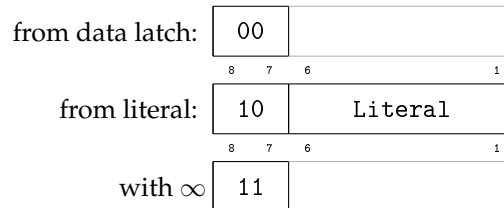
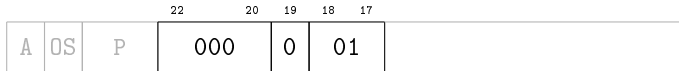
The P field is a predicate; if it does not hold, the instruction is ignored. Otherwise the two flags (A and B) are updated according to the nextA and nextB fields; each specifies the new value as the logical OR of zero or more inputs:



Each bit corresponds to one possible input; all inputs whose bits are set are ORed together, and the resulting value is assigned to the flag. Note that if none of the bits are set, the value assigned is zero. Note also that it is possible to produce a 1 by ORing any flag with its complement.

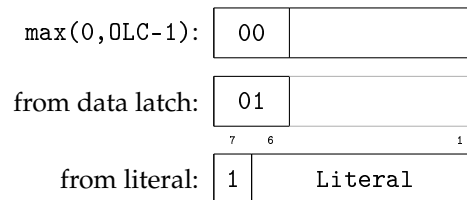
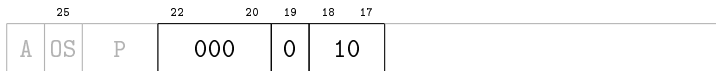
4.4 setInner

This instruction loads the inner loop counter with either a literal number, the special value ∞ , or the contents of the data register.



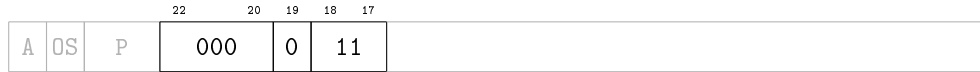
4.5 setOuter

This instruction loads the outer loop counter OLC with either $\max(0, \text{OLC}-1)$, a literal or the contents of the data register.



4.6 tail

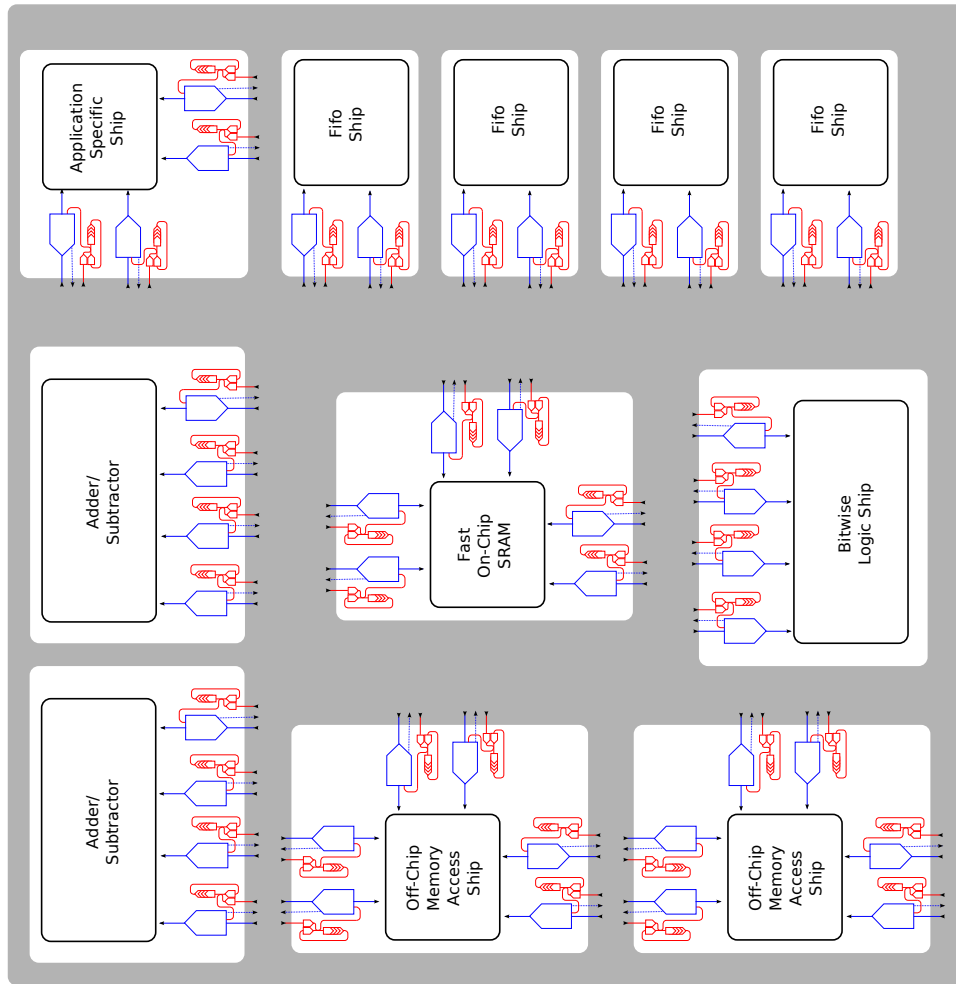
When a `tail` instruction reaches IH, it seals the hatch. The `tail` instruction does not enter the instruction fifo.

4.7 takeOuterLoopCounter

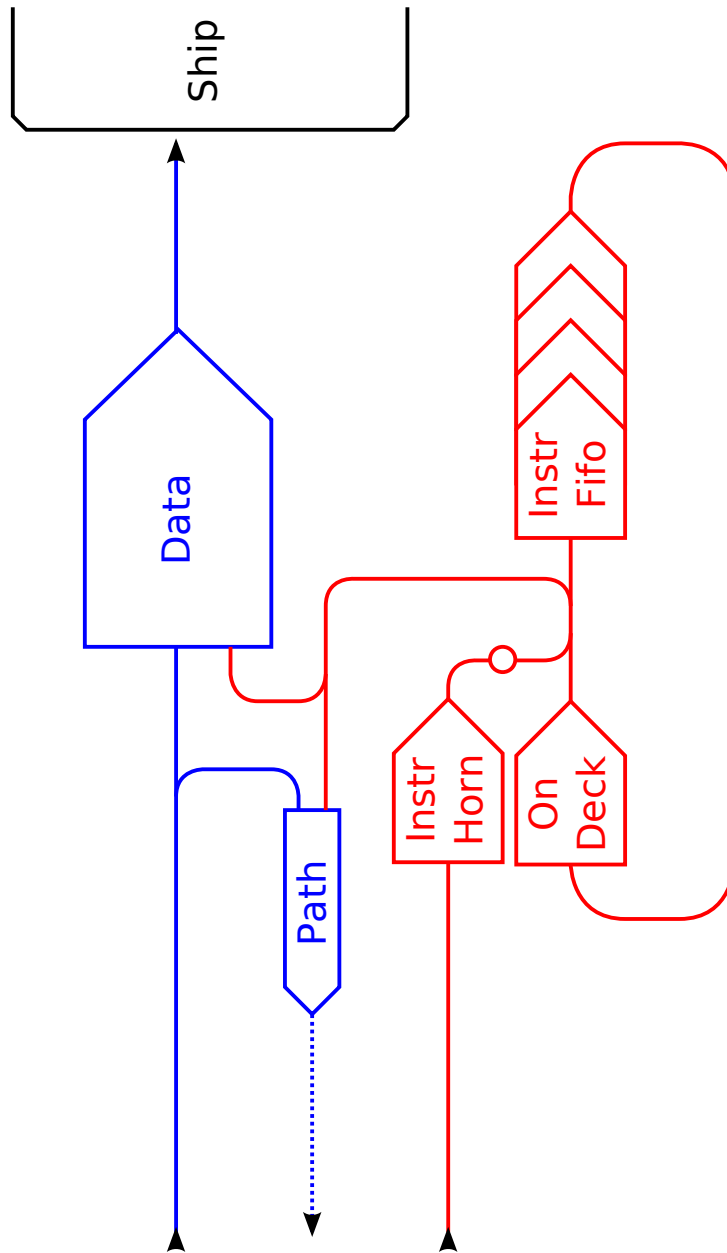
This instruction copies the value in the outer loop counter OLC into the least significant bits of the data latch and leaves all other bits of the data latch unchanged.

4.8 takeInnerLoopCounter

This instruction copies the value in the inner loop counter ILC into the least significant bits of the data latch and leaves all other bits of the data latch unchanged.



Input Dock



Output Dock

