

# AM33: The FleetTwo Dock

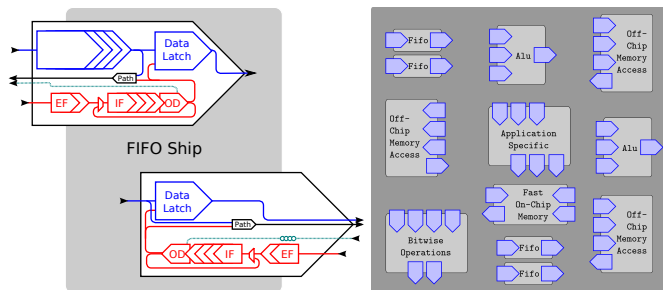
Adam Megacz

June 5, 2008

## Abstract

### Changes:

- 06-Jun Factored in Russell Kao's comments (thanks!)  
Added mechanism for setting C-flag from fabric even on outboxes
- 05-Jun Made OLC test a predicate-controlled condition  
Rewrote "on deck" section  
Added "unset" value for ILC  
Changed DP to DataPredecessor for clarity
- 30-Apr added comment about address-to-path ship  
changed DST field of set instruction from 2 bits to 3  
changed the order of instructions in the encoding map
- 23-Apr added epilogue fifo to diagrams  
indicated that a token sent to the instruction port is treated as a torpedo
- 18-Apr replaced setInner, setOuter, setFlags with unified set instruction  
replaced literal with shift instruction
- 17-Apr Made all instructions except setOuter depend on OLC>0  
Removed ability to manually set the C flag  
Expanded predicate field to three bits  
New literals scheme (via shifting)  
Instruction encoding changes made at Ivan's request (for layout purposes)  
Added summary of instruction encodings on last page



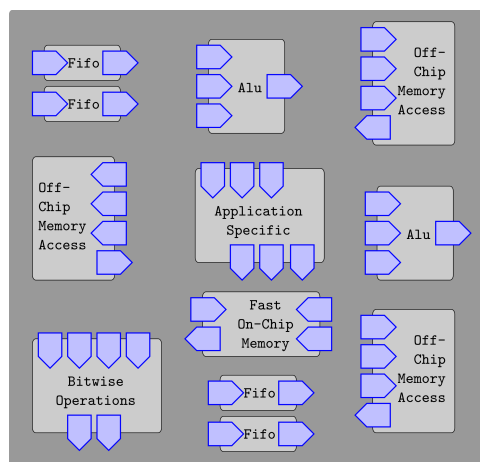
## 1 Overview of Fleet

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 to be delivered 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 to some other destination.

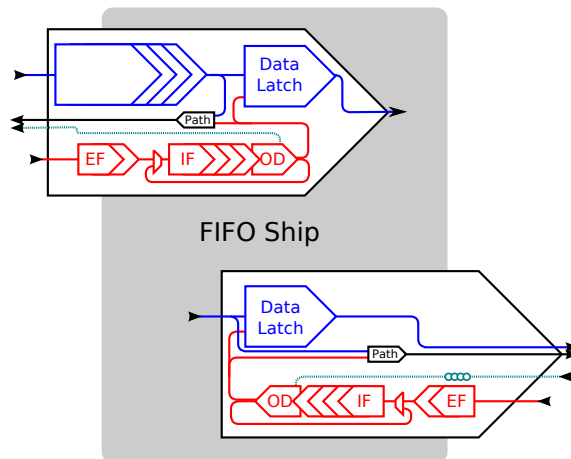
When an instruction sends a packet into the switch fabric, it may specify that the payload of the packet is irrelevant. Such packets are known as *tokens*, and consume less energy than data packets.



Overview of a Fleet processor; gray shading represents a packet-switched network fabric, docks are shown in blue.

## 2 The FleetTwo Dock

The diagram below represents a conceptual view of the interface between ships and the switch fabric; actual implementation circuitry may differ.



An “input” dock and “output” dock connected to a ship. Solid blue lines carry either tokens or data words, red lines carry either instructions or torpedoes, and dashed lines carry only tokens.

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. The dock also includes a *patch latch*, which stores the path along which outgoing packets will be sent.

Note that the pump in each dock has a destination of its own; this is the *instruction destination* mentioned in the previous section.

Every dock’s data destination has *two* addresses on the switch fabric which differ by a single bit; this bit is known as the “signal” bit. A packet will be routed to the same destination regardless of the value of the signal bit; the value of this bit is used to pass the values of flags between docks. Note that instruction destinations do not need to have a signal bit.

### 3 Instructions

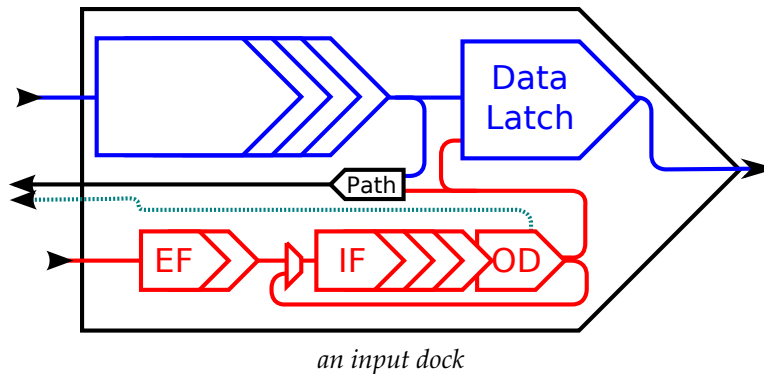
In order to cause an instruction to execute, the programmer must first arrange for that instruction word to arrive in the data latch of some output dock. For example, this might be the “data read” output dock of the memory access ship or the output of a fifo ship. Once an instruction has arrived at this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



#### 3.1 Life Cycle of an Instruction

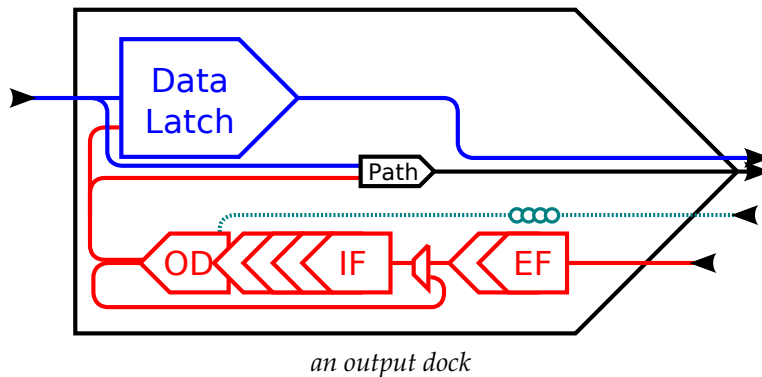
The diagram below shows an input dock for purposes of illustration:



Note the mux on the path between EF (epilogue fifo) and IF (instruction 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 whenever the outer loop counter is set to zero (for any reason<sup>1</sup>).

When an instruction arrives at EF, 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” (OD) stage, where it may execute.

<sup>1</sup>this includes OLC being decremented to zero, a set instruction, or the occurrence of a torpedo



### 3.2 Format of an Instruction

All instruction words have the following format:



- The I bit stands for *Interruptible*, and indicates if an instruction is vulnerable to torpedoes.
- The OS (“One Shot”) bit indicates whether or not this instruction can pass through the pump more than once. If set to 1, then the instruction is a “one-shot” instruction, and does not pass through the instruction fifo more than once.
- The P bits are a *predicate*; this holds a code which indicates if the instruction should be executed or ignored depending on the state of flags in the dock.

### 3.3 Loop Counters

A programmer can perform two types of loops: *inner* loops of only one instruction and *outer* loops of multiple instructions. Inner loops may be nested within an outer loop, but no other nesting of loops is allowed.

The dock has two loop counters, one for each kind of loop:

- OLC is the Outer Loop Counter
- ILC is the Inner Loop Counter

The OLC applies to all instructions and can hold integers  $0 \dots \text{MAX\_OLC}$ .

The ILC applies only to move instructions and can hold integers  $0 \dots \text{MAX\_ILC}$  as well as a special value:  $\infty$ . When  $\text{ILC}=0$  the next move instruction executes zero times (ie is ignored). When  $\text{ILC}=\infty$  the next move instruction executes until interrupted by a torpedo. After every move instruction the ILC is reset to 1 (note that it is reset to 1, *not to 0*).

### 3.4 Flags and Predication

The pump has three flags: A, B, and C.

- The A and B flags are general-purpose flags which may be set and cleared by the programmer.
- The C flag is known as the *control* flag, and may be set by the move instruction based on information from the ship or from an inbound packet. See the move instruction for further details.

The P field specifies a three-bit *predicate*. The predicate determines which conditions must be true in order for the instruction to execute; if it is not executed, it is simply *ignored*. The table below shows what conditions must be true in order for an instruction to execute:

Code	Execute	if
000:	$\text{OLC} \neq 0$	and $A=0$
001:	$\text{OLC} \neq 0$	and $A=1$
010:	$\text{OLC} \neq 0$	and $B=0$
011:	$\text{OLC} \neq 0$	and $B=1$
100:	$\text{OLC} \neq 0$	and $C=0$
101:	$\text{OLC} \neq 0$	and $C=1$
110:	$\text{OLC} \neq 0$	
111:	always	

### 3.5 On Deck

When an instruction arrives on deck, two concurrent processes are started. No subsequent instruction may come on deck until both processes have completed:

1. If  $OLC=0$  or the instruction on deck has  $OS=1$ , do nothing. Otherwise, wait for the hatch to be sealed and enqueue a copy of the instruction on deck.
2. This process has three possible actions:
  - If the instruction's predicate condition is met (see section on predicates); if it is *not* met, do nothing.
  - *Otherwise* if the instruction has  $I=0$  and a torpedo is present: consume the torpedo, set  $OLC=0$ , unseal the hatch, and transmit a token to the address in the *torpedo acknowledgment path latch*
  - *Otherwise* if  $ILC \neq 0$  or the instruction is *not* a move: execute the instruction.

#### 3.5.1 Torpedoes

There is a small fifo marked EF for "Epilogue Fifo" just beyond the instruction destination; 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 or the outer loop counter expires.

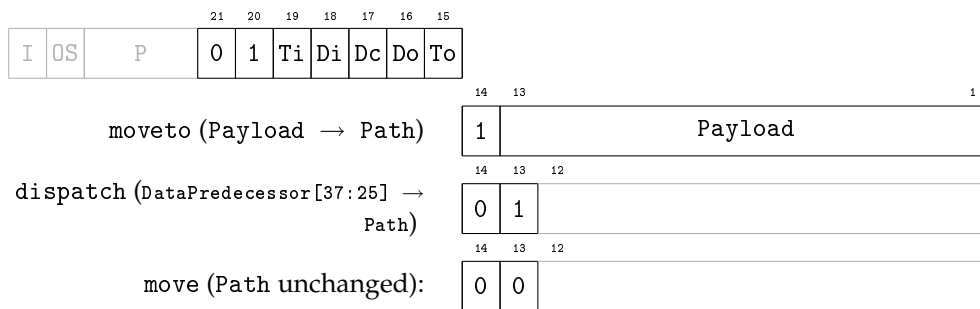
A *token* sent to a ship's instruction destination is treated as a torpedo; the check for a torpedo is performed *before* the head of the Epilogue fifo. Note that because the check for a token is performed "upstream" of the epilogue fifo, torpedos will still be effective even when the Epilogue fifo is nearly full.

## 4 Instructions

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

```
move (variants: moveto, dispatch)
shift
set
tail
```

### 4.1 move (variants: moveto, 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 move instruction to attempt execution.

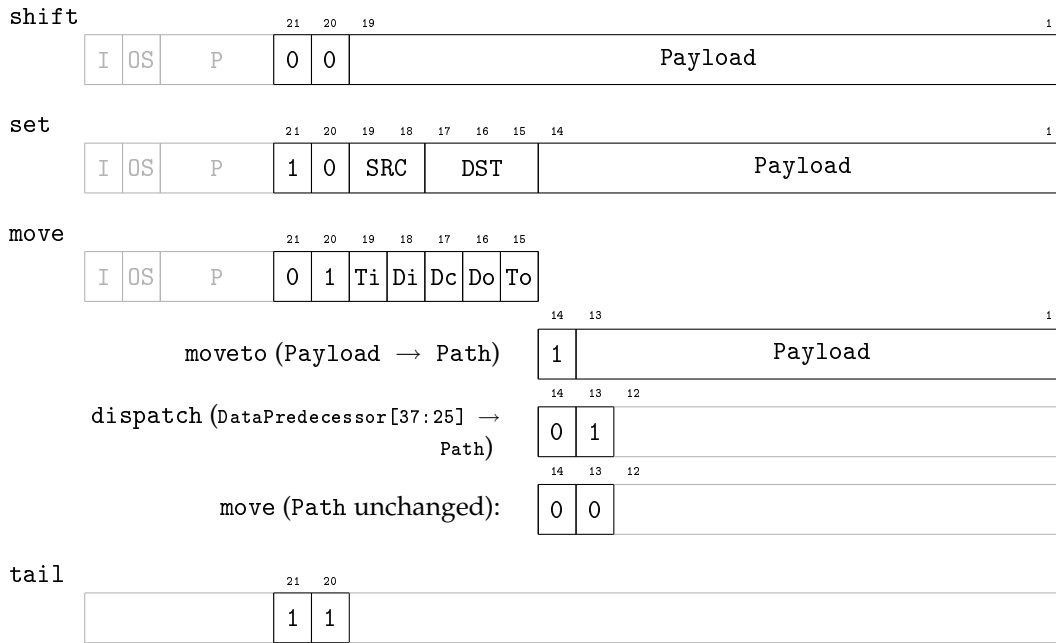
Every time the move instruction executes, the C flag is set:

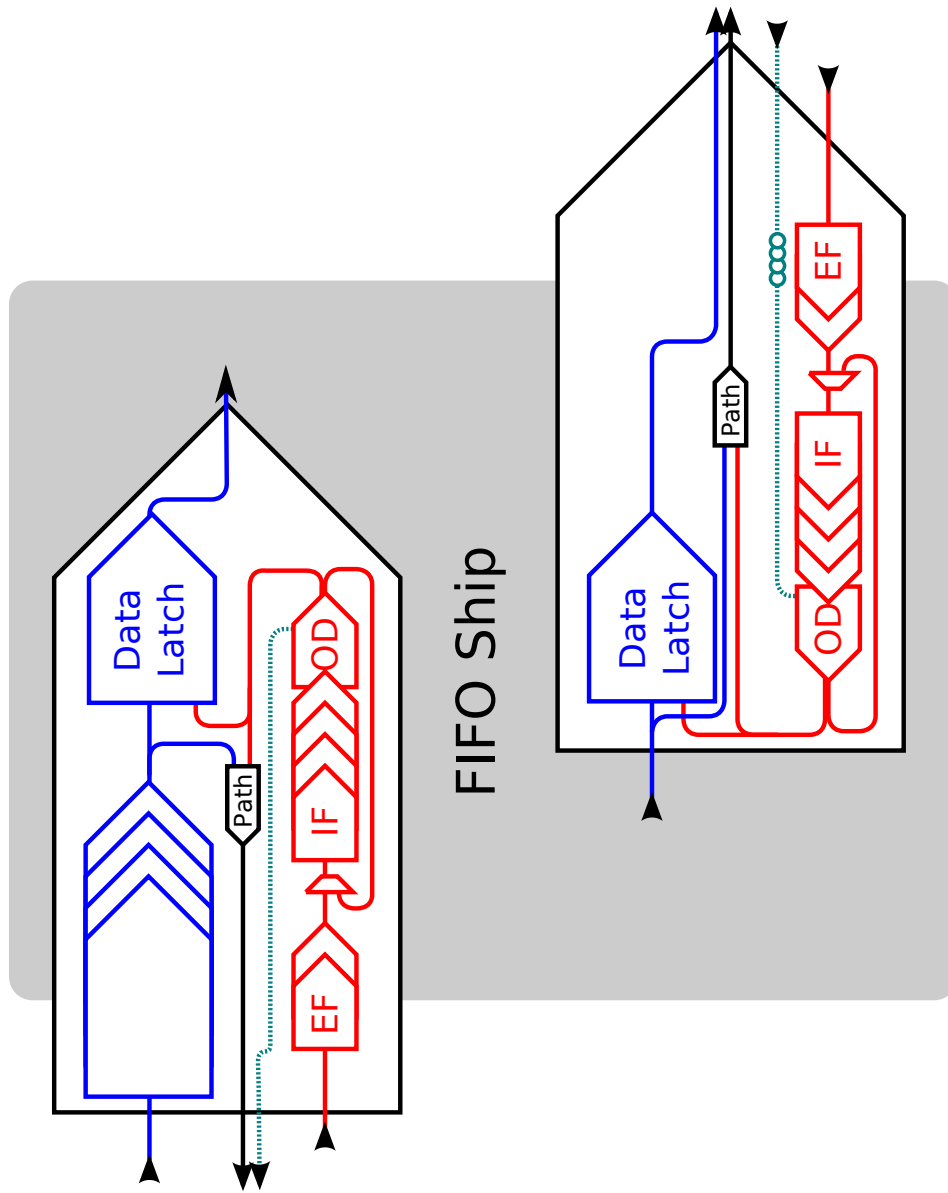
- At an *input* dock the C flag is set to the signal bit of the incoming packet if Di or Ti is set.
- At an *output* dock the C flag is set to a value provided by the ship if the Di bit is set, and to the signal bit of the incoming packet if Di is clear and Ti is set.





## Instruction Encoding Map







Output Dock

