

AM33: The FleetTwo Dock

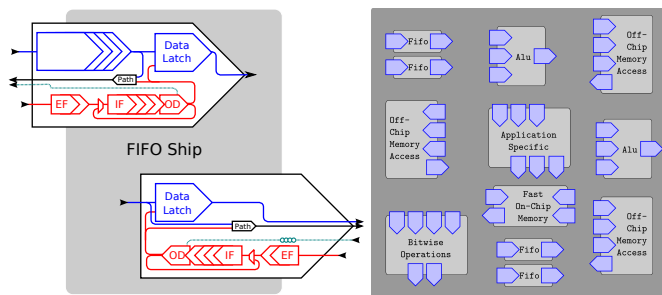
Adam Megacz

September 20, 2008

Abstract

Changes:

- 20-Sep Update hatch description to match IES50
- 28-Aug Note that decision to requeue is based on value of OLC *before* execution
Note that decision to open the hatch is based on value of OS bit
- 10-Jul Added OLC=0 predicate
Eliminated TAPL (made possible by previous change)
Expanded set Immediate field from 13 bits to 14 bits (made possible by previous change)
- 09-Jul Fixed a few typos
Added DataLatch → TAPL (Amir's request)
Eliminate ability to predicate directly on C-flag (Ivan's request)
- 16-Jun When a torpedo strikes, ILC is set to 1
Only move can be torpedoed (removed I-bit from set/shift)
- 11-Jun Changed all uses of "Payload" to "Immediate" (not in red)
Reworked encoding of set instruction
- 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



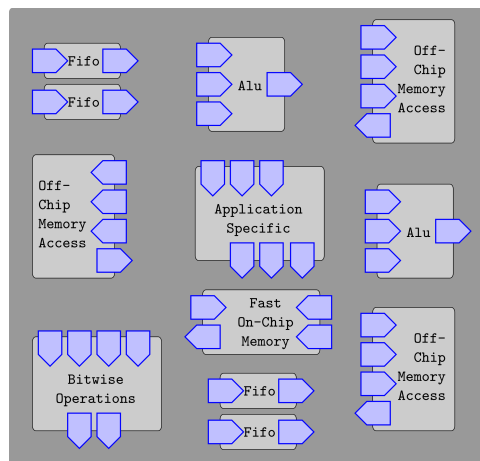
1 Overview of Fleet

A Fleet processor is organized around a *switch fabric*, which is a packet-switched network with reliable in-order delivery. The switch fabric is used to carry data between different functional units, called *ships*. Each ship is connected to the switch fabric by one or more programmable elements known as *docks*.

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 instruction packets into the switch fabric with paths that will lead them to instruction destinations of the docks at which they are to execute.

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. When an instruction executes it may consume this data and may present a data value to the ship or transmit a packet.

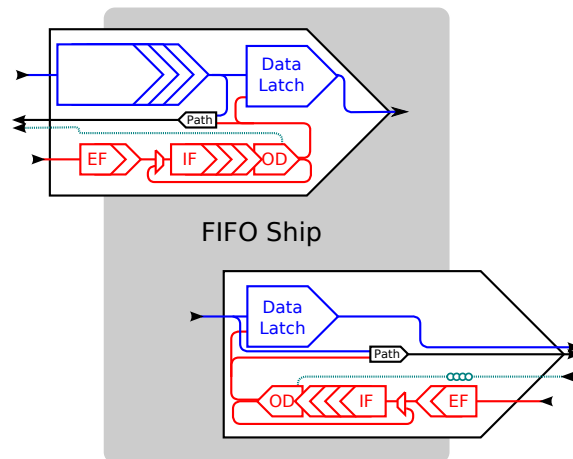
When an instruction sends a packet into the switch fabric, it may specify that the payload of the packet is irrelevant. Such packets are known as *tokens*, and consume less energy than data packets.



Overview of a Fleet processor; dark gray shading represents the switch fabric, ships are shown in light gray, and 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 *path 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.

From any source to any dock’s data destination there are two distinct paths which differ by a single bit. This bit is known as the “signal” bit, and the routing of a packet is not affected by it; the signal bit is used to pass control values between docks. Note that paths terminating at an *instruction destination* need not have a signal bit.

3 Instructions

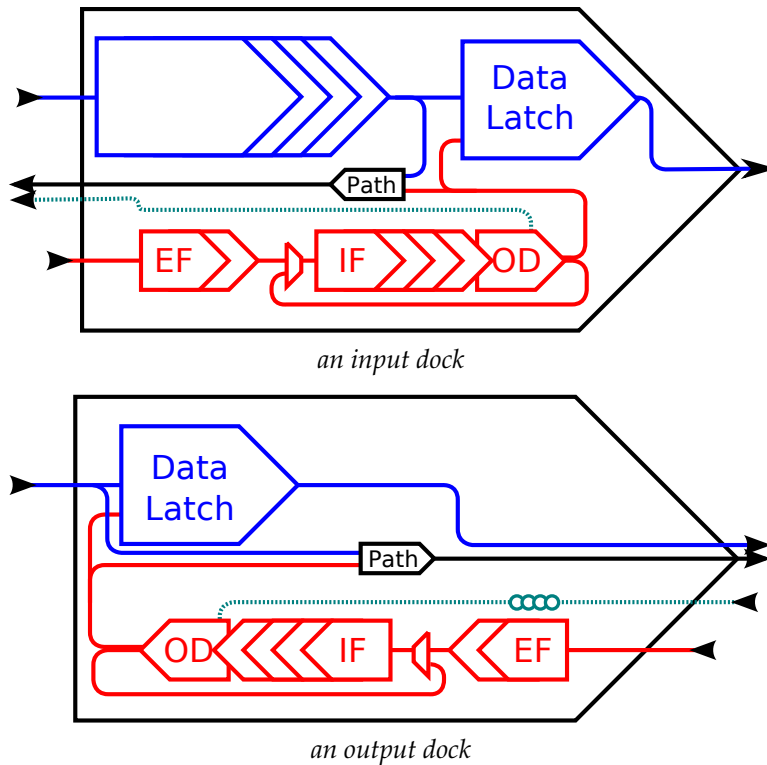
In order to cause an instruction to execute, the programmer must first arrange for that instruction word to arrive in the data latch of some output dock. For example, this might be the “data read” output dock of the memory access ship or the output of a fifo ship. Once an instruction has arrived at this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



3.1.1 Torpedoes

A token sent to an instruction destination is called a *torpedo*. When a torpedo arrives at the tail of EF, it is deposited in a waiting area (not shown) rather than being enqueued into EF.

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. This bit only appears in move instructions.
- 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 . . MAX_OLC.

The ILC applies only to move instructions and can hold integers 0 . . MAX_ILC as well as a special value: ∞ . When ILC=0 the next move instruction executes zero times (ie is ignored). When ILC= ∞ 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:	OLC \neq 0 and A=0
001:	OLC \neq 0 and A=1
010:	OLC \neq 0 and B=0
011:	OLC \neq 0 and B=1
100:	Unused
101:	OLC=0
110:	OLC \neq 0
111:	always

3.5 The Hatch

What follows is a conservative approximation of the actual behavior of the hatch. For complete details on the behavior of the hatch, see [IES50](#).

For the purposes of this section, instructions will be classified into three categories: one-shot instructions ($OS=1$), requeueable instructions ($OS=0$), and tail instructions.

To avoid deadlock, the programmer must ensure that:

- A requeueable instruction is never followed immediately by a one-shot.
- A one-shot instruction is never followed immediately by a tail.
- No contiguous sequence of requeueable instructions is longer than the length of the instruction fifo.
- If a requeueable instruction is preceded by a one-shot instruction or a tail, then it must be the case that $OLC>0$ both before and after the first time that instruction executes.
- If $OLC=0$, only a one-shot instruction may set it to a nonzero value.

The dock guarantees that:

- If a requeueable instruction is preceded by a one-shot instruction or a tail, then the *following* instruction will not execute until a tail has reached the hatch.
- Once a tail instruction reaches the hatch, no further instructions will be enqueued until a requeueable instruction reaches the execution stage and $OLC=0$.

3.6 On Deck

When an instruction arrives on deck, two concurrent processes are started:

1. Requeueing:
 - If the instruction on deck is a requeueable instruction ($OS=1$) and the outer loop counter is nonzero ($OLC>0$), a copy of the instruction is requeued.
2. Execution:
 - If the instruction's predicate condition is not met (see section on predicates), do nothing.
 - *Otherwise* if the instruction is interruptible ($I=1$) and a torpedo is present in the waiting area: consume the torpedo, set the outer loop counter to zero ($OLC=0$) and set the inner loop counter to one ($ILC=1$).

- *Otherwise* if $ILC \neq 0$ or the instruction is *not* a move: execute the instruction.

4.2 set

The set command is used to set or decrement the inner loop counter, outer loop counter, and data latch.

	26	25	24	23	22	21	20	Dest																
	0	S	P		1	0																		
Immediate → OLC								19	18	17	16	15	14	13				6				1		
								1000				100						Immediate						
Data Latch → OLC								19	18	17	16	15	14	13										
								1000				010												
OLC-1 → OLC								19	18	17	16	15	14	13										
								1000				001												
Immediate → ILC								19	18	17	16	15	14	13				7	6				1	
								0100				100						0	Immediate					
∞ → ILC								19	18	17	16	15	14	13				7						
								0100				100						1						
Data Latch → ILC								19	18	17	16	15	14	13										
								0100				010												
0-Extended Immediate → Data Latch								19	18	17	16	15	14											1
								0010				0	Immediate											
1-Extended Immediate → Data Latch								19	18	17	16	15	14											1
								0010				1	Immediate											
Update Flags								19	18	17	16				12				7	6				1
								0001							nextA			nextB						

The FleetTwo implementation is likely to have an unarchitected “literal latch” at the on deck (OD) stage, which is loaded with the possibly-extended literal *at the time that the set instruction comes on deck*. This latch is then copied into the data latch when a set Data Latch instruction executes.

Each of the nextA and nextB fields has the following structure, and indicates which old flag values should be logically ORed together to produce the new flag value:

6	5	4	3	2	1
A	\bar{A}	B	\bar{B}	C	\bar{C}

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, and that set Flags can be used to create a nop (no-op) by setting each flag to itself.

4.3 shift

Each `shift` instruction carries an immediate of 19 bits. When a `shift` instruction is executed, this immediate is copied into the least significant 19 bits of the data latch, and the remaining most significant bits of the data latch are loaded with the value formerly in the least significant bits of the data latch. In this manner, large literals can be built up by “shifting” them into the data latch 19 bits at a time.

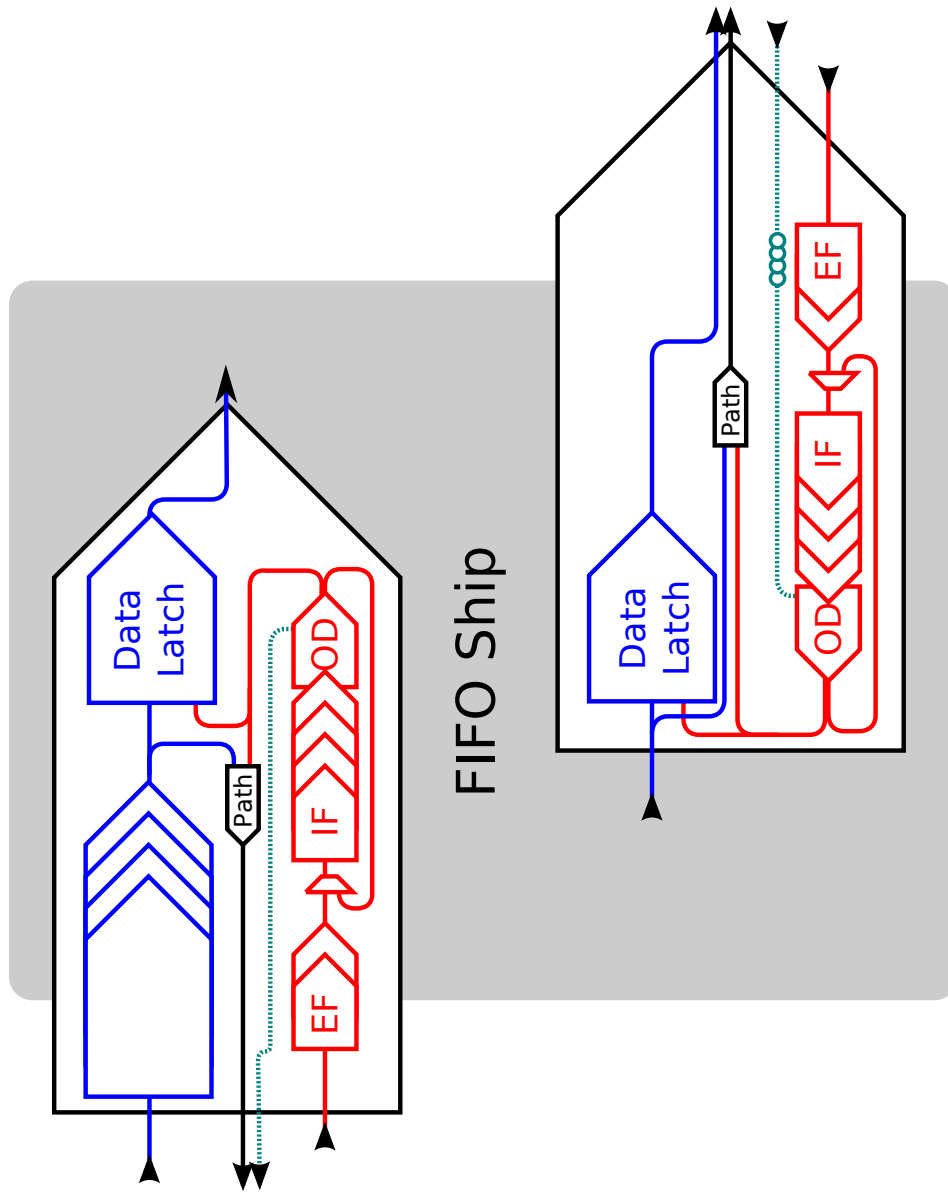


The FleetTwo implementation is likely to have an unarchitected “literal latch” at the on deck (OD) stage, which is loaded with the literal *at the time that the shift instruction comes on deck*. This latch is then copied into the data latch when the instruction executes.

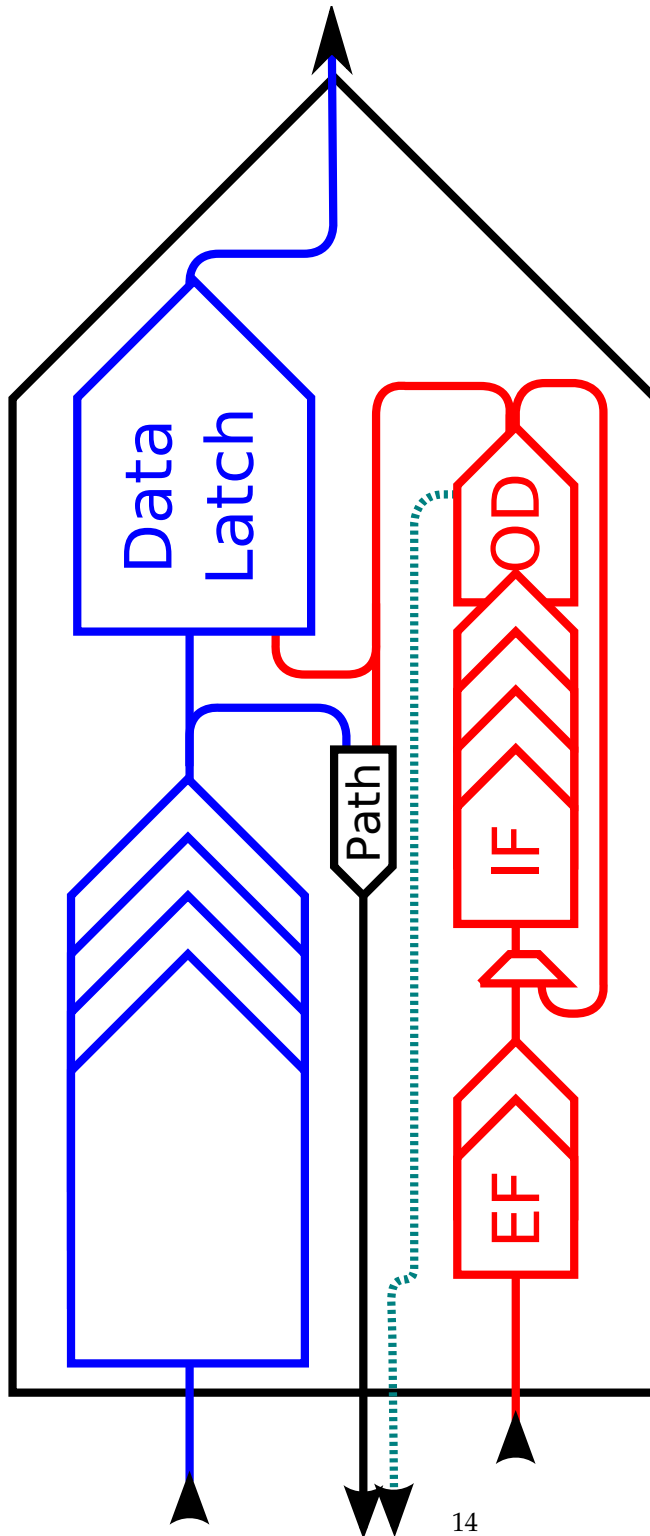
4.4 tail



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



Input Dock



Output Dock

