

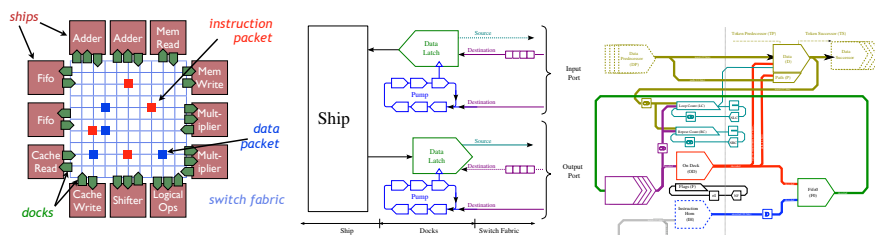
The FleetTwo Dock

March 7, 2008

Abstract

Changes:

- 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*



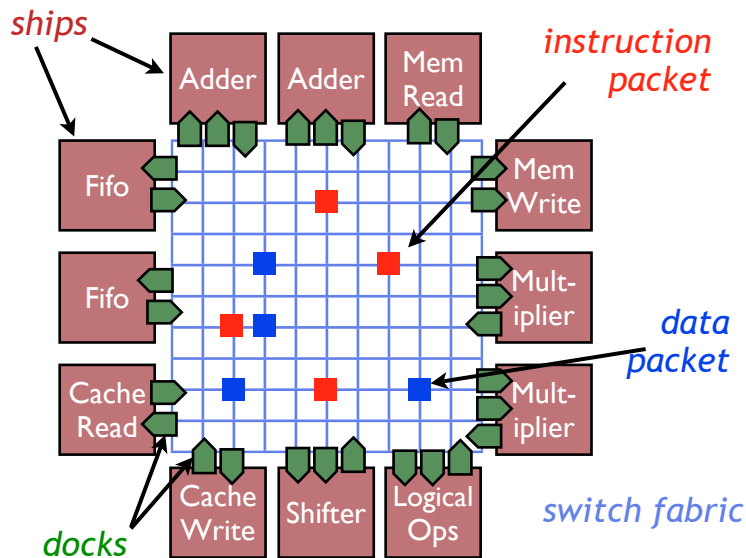
1 Overview of Fleet

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.

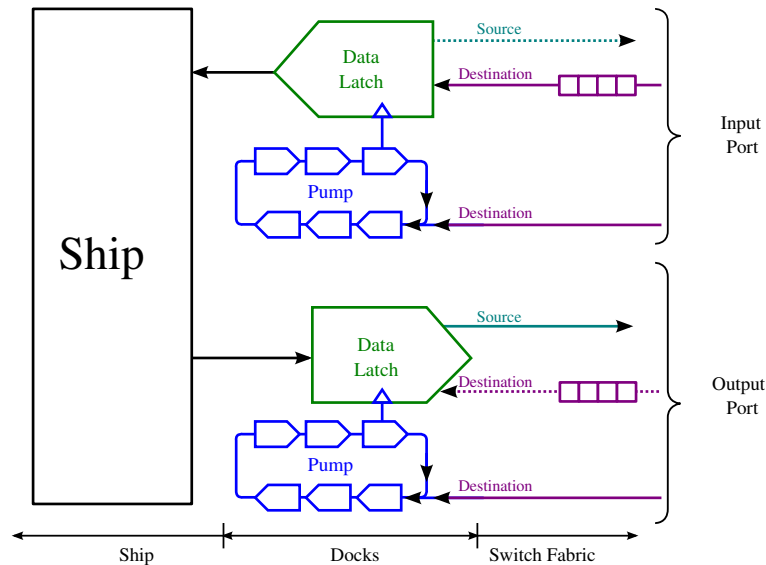
When an instruction sends a packet into the switch fabric, it may specify that the payload of the packet is irrelevant. Such packets are known as *tokens*, and consume less energy than data packets. From a programmer's perspective, a token packet is indistinguishable from a data packet with a unknown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



Overview of a Fleet processor

2 The Ship-Switch Fabric Interface

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 interface between the switch fabric and the ship

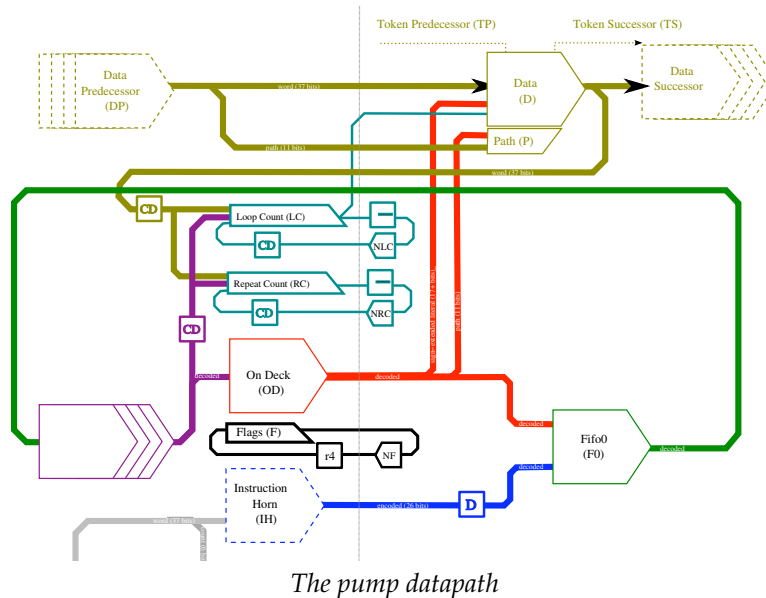
The term *port* refers to an interface to the ship, the *dock* connecting it to the switch fabric, and the corresponding sources and destinations on the switch fabric.

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 instruction fifo 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 she can avoid deadlock.

3 The FleetTwo Pump

The diagram below shows the datapath for the FleetTwo pump circuitry. The square box marked D on the output from the IH latch is the instruction decoder, which decodes word-width instructions into a set of control signals suitable for operating the pump. The boxes marked CD are carry detectors. These detect zero values in the count and also generate the partial differences used in the decrement operation.



The pump datapath

The latches of primary interest here are:

- IH: Instruction Horn (leaf node; may be shared)
- F0: Fifo Stage 0 (first fifo stage)
- OD: On Deck
- F: Flags, NF: Next Flags
- P: Path (the path to use for outbound data/tokens)
- D: Data
- DP: Data Predecessor (ship for output ports, switch fabric for input ports)
- DS: Data Successor (switch fabric for output ports, ship for input ports)
- RC: Repeat Count, NRC: Next Repeat Count
- LC: Loop Count, NLC: Next Loop Count

Each instruction that executes causes the latches of the pump to fire in two phases, denoted as the “left phase” and the “right phase”. In the diagram, the left phase latches are those to the left of the vertical line down the center, and the right phase latches are to the right. Therefore each instruction execution requires two GasP pipeline stages to complete.

3.1 Flags

The pump has four flags: A, B, S, Z. 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 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.
- The Z flag, known as the *zero* flag, is set whenever the value in the loop counter (LC) is zero. This flag can be used to perform certain operations (such as sending a completion token) only on the last iteration of a loop.

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 Z is set (LC=0)
- 11: always

4 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 *Hold* field indicates how micro instructions are gathered together into composite instructions:

- 00: *solo* – this word is not part of a composite instruction
- 01: *soloT* – like *solo*, but torpedo-able
- 10: *body* – this word is part of a composite instruction, but not the last
- 11: *tail* – this is the last micro instruction in a composite instruction

Solo instructions never reloop (described later); they are “one-shot” instructions. Multiple *solo* instructions may be in the instruction fifo simultaneously. A *solo* instruction is immune to torpedos (described later); a *soloT* instruction is not¹.

Composite instructions reloop until the loop counter is zero. When a composite instruction is in the instruction fifo, no other instructions may enter the fifo. A *body* instruction is immune to torpedos; a *tail* instruction is not.

The abbreviation *P* stands for *predicate*; this is a two-bit code that indicates if the instruction should be executed or ignored. If an instruction is ignored, it might still reloop.

¹the *soloT* instruction is meant to be used for “standing repeating” instructions

4.1 RePeating and ReLooping

	RePeating?	ReLooping?
send	Y	Y
literal	N	Y
flags	N	Y
repeat	N	Y
loop	N	Y ^a
takeLoopCounter	N	Y
takeRepeatCounter	N	Y
torpedo	n/a	n/a

^anote, however, that the decision to reloop or not is based on the value in the loop counter *before* execution of the `loop` instruction

Table 1: classification of instructions

RePeating

An instruction will repeat if it is classified as a repeating instruction and the repeat counter is nonzero. Non-repeating instructions have no effect on the repeat counter (except for `repeat`, of course).

ReLooping

Solo instructions (both `solo` and `soloT`) completely ignore the loop counter; it has no effect on them.

If a body or tail instruction reaches the on deck stage and the loop counter (LC) is zero, the instruction dies immediately without executing or relooping.

If a body or tail instruction reaches the on deck stage and the loop counter (LC) is nonzero, a (duplicate) copy of that instruction is immediately enqueued at the head of the instruction fifo; the original instruction then waits at OD until either its execution conditions are met or it is torpedoed.

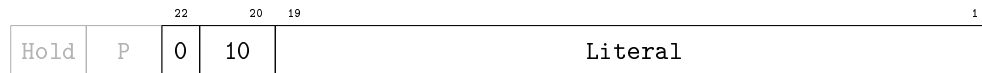
4.3 data, datahi, datalo

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

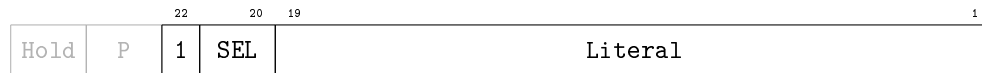
datahi: Literal[18:1] → D[37:20] (and Literal[18] → S)



datalo: Literal[19:1] → D[19:1]

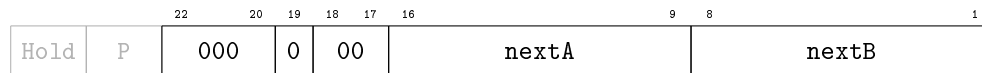


data:

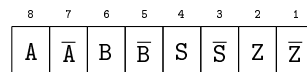


sel	D[37:20]	D[19:1]
00	Literal[18:1]	all 0
01	Literal[18:1]	all 1
10	all 0	Literal[19:1]
11	all 1	Literal[19:1]

4.4 flags



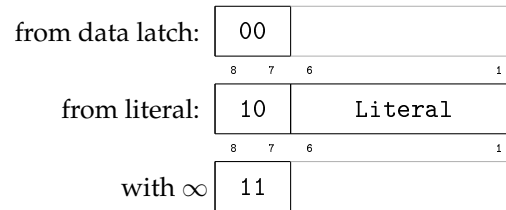
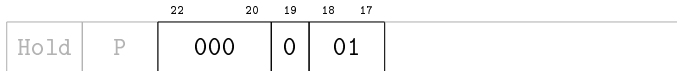
The P field is a predicate; if it does not hold, the instruction is ignored. Otherwise the two flags (A and B) are updated according to the nextA and nextB fields; each specifies the new value as the logical OR of zero or more inputs:



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.

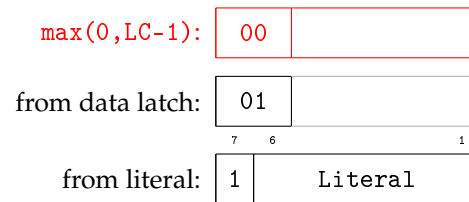
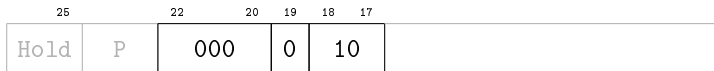
4.5 repeat

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



4.6 loop

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

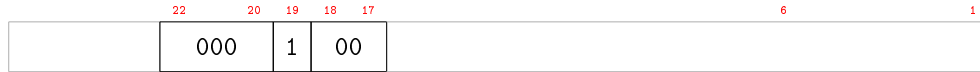


4.7 takeLoopCounter

The P field is a predicate; if it does not hold, the instruction is ignored (but may reloop). This instruction copies the value in the loop counter LC into the least significant bits of the data latch and leaves all other bits of the data latch unchanged.

4.8 takeRepeatCounter

The P field is a predicate; if it does not hold, the instruction is ignored (but may reloop). This instruction copies the value in the repeat counter RC into the least significant bits of the data latch and leaves all other bits of the data latch unchanged.

4.9 torpedo

When a torpedo instruction reaches IH, it will wait there until an instruction is on deck (at OD) and that instruction's Hold field is tail or soloT. The torpedo will then annihilate the on-deck instruction *and set the loop counter to zero.*

