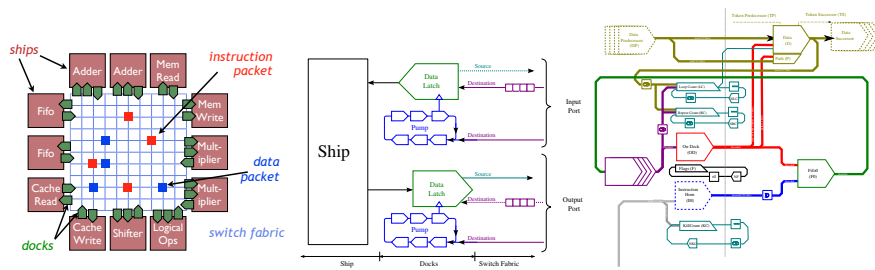


The FleetTwo Dock

January 9, 2008

Abstract



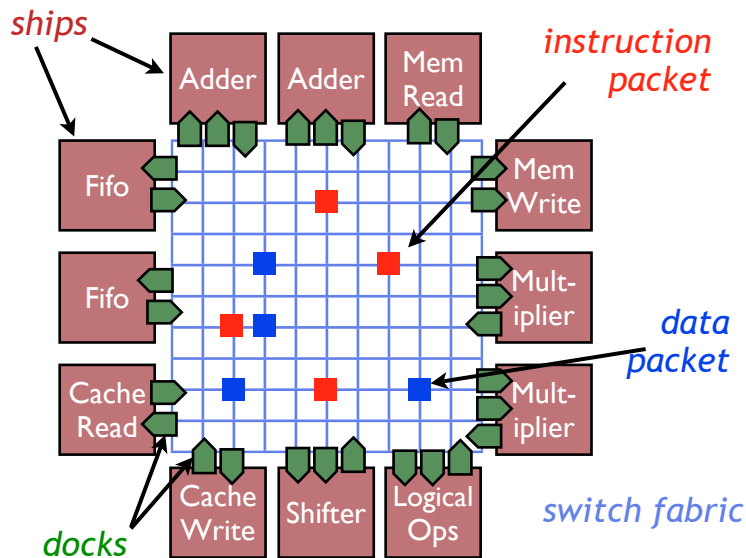
1 Overview of Fleet

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. When the instruction is executed, 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, and may cause a data value to be presented to the ship or transmitted through the switch fabric to some other destination.

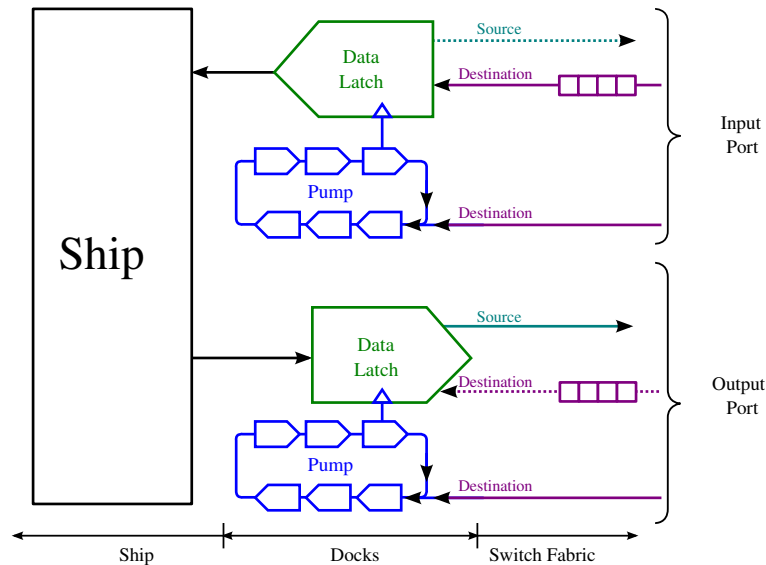
When a dock sends a packet into the switch fabric, it may specify that the payload of the packet is irrelevant. Such packets are known as *tokens*, and consume less energy than data packets. From a programmer's perspective, a token packet is indistinguishable from a data packet with a random payload.



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 which can send and receive only tokens – not data items – are drawn as dashed lines.



The interface between the switch fabric and the ship

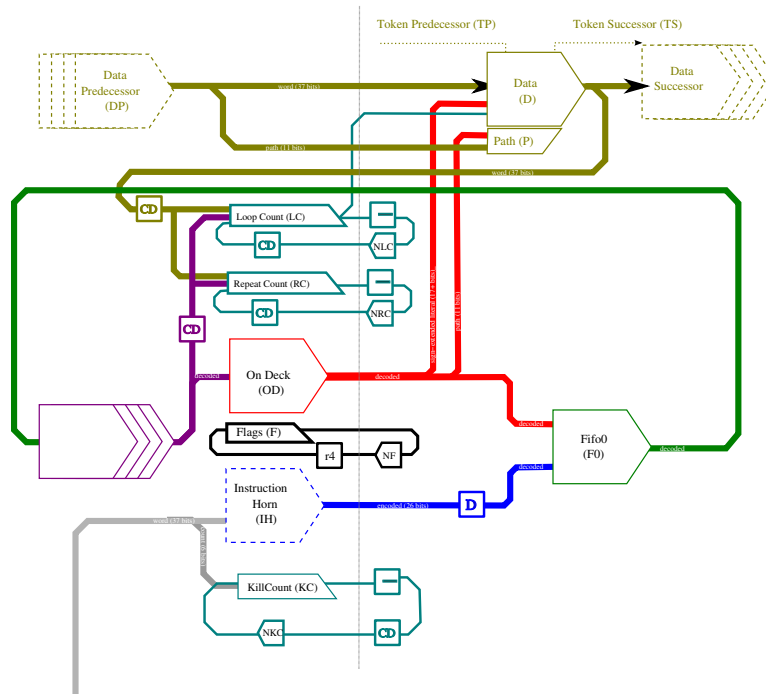
The term *port* refers to an interface to the ship, the *dock* connecting it to the switch fabric, and the corresponding sources and destinations on the switch fabric.

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 there is no buffering fifo guarding this destination, while all other destinations are guarded by a buffering fifo. The size of this fifo is 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 (Requeue) Count, NLC: Next Loop (Requeue) Count
- KC: Kill Count, NKC: Next Kill Count

Each instruction which is executed 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 cycles 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, is generated from the value currently in latch D. The function which generates this flag is ship-specific, but should be assumed to be equal to the most significant bit of the value in the latch unless stated otherwise. Essentially, this function indicates how to interpret a word as a boolean; this interpretation is ship-specific.
- 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:

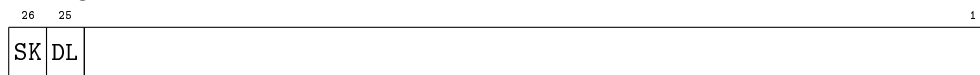
- 00: never
- 10: if A is set
- 01: if B is set
- 11: always

4 Instructions

Each instruction is 26 bits long, which makes it possible for an instruction and an 11-bit path (to the dock which executes the instruction) to fit in a single word of memory.

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 radically different.

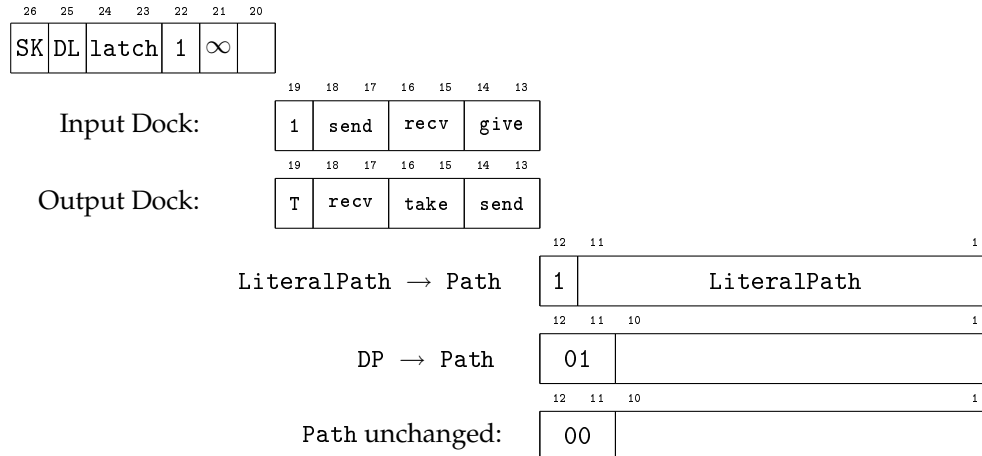
All instructions other than KILL, MASSACRE, CLOG, and UNCLOG have the following format:



The abbreviation SK stands for *Successor Killable*; if this bit is cleared, then a kill will never prevent the following instruction from executing if the current instruction was executed. This functionality is implemented by giving the pump two states: *killable* and *not killable*. Executing an instruction with the SK bit set puts the pump in the *killable* state; executing an instruction with the SK bit cleared takes it out of this state.

The abbreviation DL stands for *Decrement Loop*; if this bit is set, the loop counter is decremented. Once an instruction has finished executing (including repeating, if applicable), the instruction will requeue if the loop count (LC) value was greater than zero *prior to decrementing*.

4.1 Normal Instruction



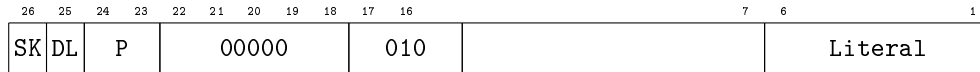
Note that bits 15+16 are effectively “drain data predecessor”, while bits 13+14 are effectively “fill data successor.” This should simplify the hardware.

- latch - pulse data latch (predicate)
- send - fill switch fabric successor (predicate)
- T - tokenhood bit if switch fabric successor is filled
- recv - await and drain switch fabric predecessor (predicate)
- take - await ship predecessor and drain (predicate)
- give - fill ship successor (predicate)
- ∞ - standing instruction

The F0, DS, and TS stages must all be empty in order for an instruction to execute.

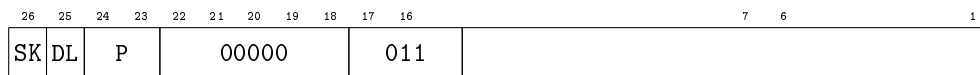
If the ∞ bit is *not* set, the repeat count (RC) is latched with $\max(\text{RC}-1, 0)$, and the instruction retires if $\text{RC}=0$ prior to decrementing. Otherwise, the instruction repeats.

4.4 Load Literal into Repeat Counter



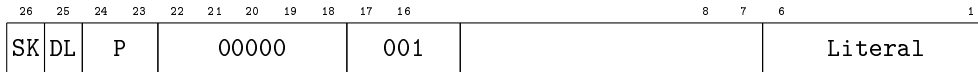
The P field is a predicate; if it does not hold, the instruction is ignored (but may requeue). This instruction sets the repeat counter RC to `Literal[6:1]`.

4.5 Load Data Latch into Repeat Counter



The P field is a predicate; if it does not hold, the instruction is ignored (but may requeue). This instruction sets the repeat counter RC to `D[6:1]` (the low-order bits of the data latch).

4.6 Load Literal into Loop Counter



The P field is a predicate; if it does not hold, the instruction is ignored. This instruction sets the loop counter LC to `Literal[6:1]`.

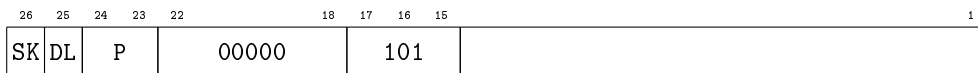
This instruction never requeues.

4.7 Load Data Latch into Loop Counter



The P field is a predicate; if it does not hold, the instruction is ignored (but may requeue). This instruction sets the loop counter LC to `D[6:1]` (the low-order bits of the data latch). **FIXME: does this requeue?**

4.8 Load Loop Count into Data Latch



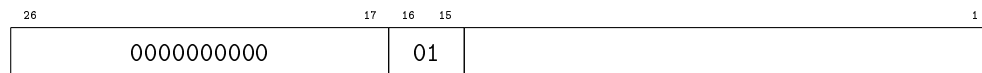
The P field is a predicate; if it does not hold, the instruction is ignored (but may requeue). This instruction loads the value in the loop counter LC into the data latch.

4.9 Kill



When a `kill` instruction reaches IH, it will wait there for the pump to be in the *killable* state and for the OD stage to be full. When this occurs, the instruction at OD is retired and the count field of the `kill` instruction is decremented. If the `kill` instruction's count was 0 before decrementing, the `kill` instruction is retired. The programmer is assured that a `kill` instruction with a count of n will kill $n + 1$ *consecutive* instructions.

4.10 Massacre



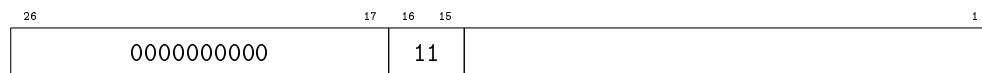
When a `massacre` instruction reaches IH, it will wait there for the pump to be in the *killable* state and for the OD stage to be full. When this occurs, all instructions in the instruction fifo are retired.

4.11 Clog



When a `clog` instruction reaches OD, it remains there and no more instructions will be executed until an `unclog` is performed.

4.12 UnClog



When an `unclog` instruction reaches IH, it will wait there until a `clog` instruction is at OD. When this occurs, both instructions retire.

