AM42: The F2 Dock

Adam Megacz

August 29, 2009

Abstract

To Do:

```
- tokenhood as address bit
- signal/path boundary/etc
- Rename EPI and OD to something more meaningful
- Get rid of OD?
- get rid of shadow latch
- single counter
- figure out C-flag / signal bit situation
- Suggestion that there should be a "T" flag
- Get rid of "shadow latch" for literals?
- unify flags and signal bit by saying that the dock can
  see the upper X bits of a word?
  - should have a way to set just the upper X bits of the word
  - flags are actually part of the data latch!
  - the signal bit(s) belong to the Destination (or is it the Path?)
- flushing situation
- How do you get a runtime count value to an input dock?
- Simplify the whole c-flag/signal-bit situation
- tokenhood should be LITERALLY an address bit!

- kinds of data:
    - ship-to-ship data (words)
    - dock-to-dock data (signal bits)
    - ship-to-dock data (c-flag)
    - dock-to-ship data (flushing/bonus bit)
- sources of these:
    - instruction stream
    - ship word (at output dock)
    - ship extras
    - packet word (at input dock)
    - packet signal bit
```

29-Aug Initial Version

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 four destinations: one each for *instructions*, *torpedoes*, *tokens*, and *words*. 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.

Packets sent to token and torpedo destinations carry no payload. Such packets consume less energy than instruction packets or word packets.

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.

X

Each dock consists of a *data latch*, which is as wide as a single machine word and a circular *instruction fifo* of instruction-width latches. The values in the instruction fifo 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 instruction fifo in each dock has a destination of its own; this is the *instruction destination* mentioned in the previous section. A token sent to an instruction destination is called a *torpedo*; it does not enter the instruction fifo, but rather is held in a waiting area where it may interrupt certain instructions (see the section on the `move` instruction for further details).

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.

Source-sequence guarantee. Shared across instruction/torpedo (?) and token/word destinations.

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 destination* of the dock at which it is to execute.

Each instruction is 25 bits long, which makes it possible for an instruction and an 12-bit path to fit in a single word of memory. This path is the path from the *dispatching* dock to the *executing* dock.

Note that the 12 bit `dispatch path` field is not the same width as the 13 bit `Immediate` path field in the `move` instruction, which in turn may not be the same width as the actual path latches in the switch fabric.

The algorithm for expanding a path to a wider width is specific to the switch fabric implementation, and is not specified by this document.[1] In particular, because the `dispatch path` field is always used to specify a path which terminates at an instruction destination (never a data destination), and because instruction destinations ignore the signal bit, certain optimizations may be possible.

3.1 Loop Counter

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

The dock has one loop counter, called `LC`. It is the same width as a word carried through the switch fabric (37 bits).

3.2 Flags

The dock has four flags: `A`, `B`, `C`, and `Z`.

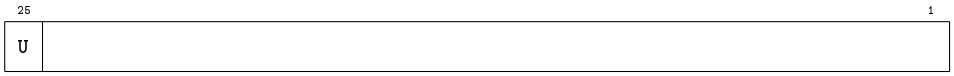
- The `A` and `B` flags are general-purpose flags which may be set and cleared by the programmer.

[1] for the Marina experiment, the correct algorithm is to sign-extend the path; the most significant bit of the given path is used to fill the vacant bit of the latch

- 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` flag is used for predication; see the next section for details. When a torpedo strikes or the counter is decremented from any value to zero, the `P` flag is cleared. The `P` flag may also be set and cleared by the `set` instruction.
- The `Z` flag is known as the *zero* flag. The `Z` flag is *set* whenever the `LC` is zero. In an actual implementation the `Z` flag might require an actual latch; it might simply be derived from the "zeroness" of the `LC`.

3.3 Predication

All instructions except for `head` and `tail` have a bit marked `U`, for *unconditional*. An instruction with the `U` bit set always executes. An instruction with the `U` bit cleared will execute *only if the* `P` *flag is set*.



3.4 The Requeue Stage

The requeue stage has two inputs, which will be referred to as the *enqueueing* input and the *recirculating* input. It has a single output which feeds into the instruction fifo.

The requeue stage has two states: UPDATING and CIRCULATING.

3.4.1 The UPDATING State

On initialization, the dock is in the UPDATING state. In this state the requeue stage is performing three tasks:

- it is draining the previous loop's instructions (if any) from the fifo
- it is executing any "one shot" instructions which come between the previous loop's `tail` and the next loop's `head`
- it is loading the instructions of the next loop into the fifo.

In the UPDATING state, the requeue stage will accept any instruction other than a `tail` which arrives at its *enqueueing* input, and pass this instruction to its output. Any instruction other than a `head` which arrives at the *recirculating* input will be discarded.

Note that when a `tail` instruction arrives at the *enqueueing* input, it "gets stuck" there. Likewise, when a `head` instruction arrives at the *recirculating* input, it also "gets stuck". When the requeue stage finds *both* a `tail` instruction stuck at the *enqueueing* input and a `head` instruction stuck at the *recirculating* input, the requeue stage discards both the `head` and `tail` and transitions to the CIRCULATING state.

3.4.2 The CIRCULATING State

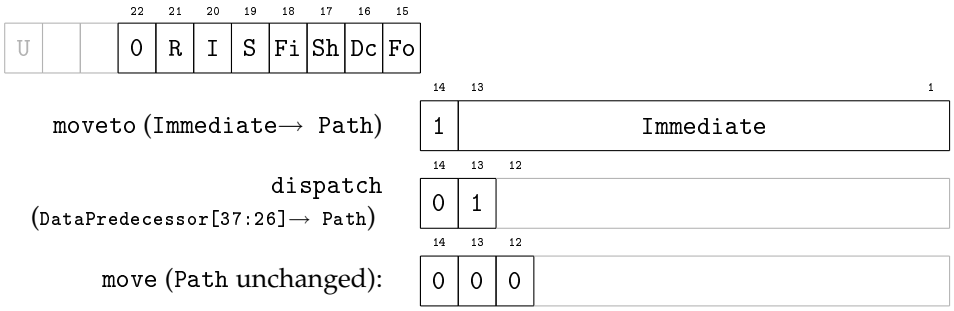
In the CIRCULATING state, the dock repeatedly executes the set of instructions that are in the instruction fifo.

In the CIRCULATING state, the requeue stage will not accept items from its *enqueueing* input. Any item presented at the *recirculating* input will be passed through to the requeue stage's output.

When an `abort` instruction is executed, the requeue stage transitions back to the UPDATING state. Note that `abort` instructions include a `U` bit – an `abort` instruction with that bit set will not cause this transition when the `P` flag is cleared.

4 Instructions

4.1 move



- `Fi` - Fabric input: wait for fabric predecessor to be full and drain it.
- `Fo` - Fabric output: wait for fabric successor to be empty and fill it.
- `Dc` - Data Capture: pulse the data latch.
- `Sh` - Ship: at an input/output dock, wait for the ship successor/predecessor to be empty/full and fill/drain it.

The fabric successor must be empty in order for a `move` instruction to attempt execution.

If the `S` bit is set, the `move` instruction will subtract one from the `LC` counter each time it executes. An instruction with only this bit set (and no other) takes the place of the dedicated "decrement OLC" instruction in previous designs.

If the `R` bit is set, the `move` instruction will execute repeatedly until its predicate no longer holds (or a torpedo strikes). An "infinite" or "standing" move can be achieved by setting the `R` bit and clearing the `S` bit.

The `I` bit stands for `Immune`, and indicates if the instruction is immune to torpedoes. If a `move` instruction which is not immune is waiting to execute and a torpedo is lying in wait, the torpedo *strikes*. When a torpedo strikes, the `move` instruction and the torpedo are both consumed and the `LC` is set to zero.

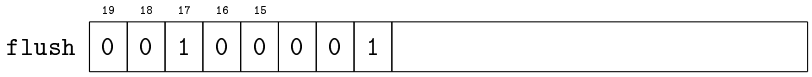
The C Flag

Every time the `move` instruction executes, the `C` flag may be set:

- At an *input* dock the `C` flag is set to the signal bit of the incoming packet.
- At an *output* dock the `C` flag is set to a value provided by the ship if the `Dc` bit is set. If the `Dc` bit is not set, the `C` flag is set to the signal bit of the incoming packet.

Flushing

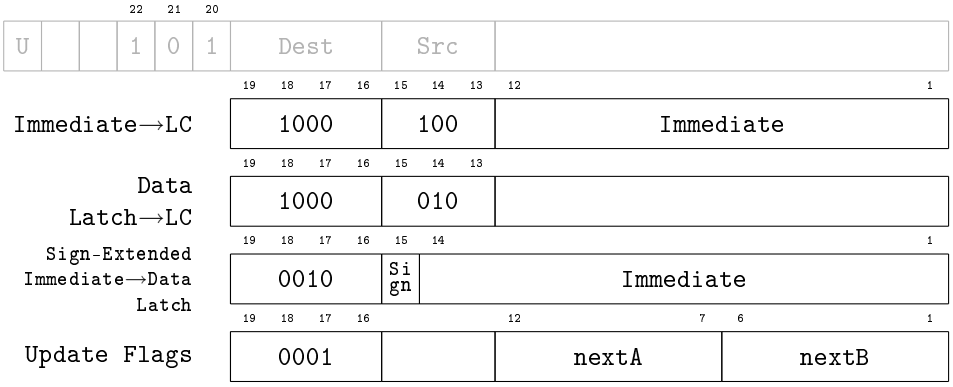
The `flush` instruction is a variant of `move` which is valid only at input docks. It has the same effect as `deliver`, except that it sets a special "flushing" indicator along with the data being delivered.



When a ship fires, it must examine the "flushing" indicators on the input docks whose fullness was part of the firing condition. If all of the input docks' flushing indicators are set, the ship must drain all of their data successors and take no action. If some, but not all, of the indicators are set, the ship must drain *only the data successors of the docks whose indicators were* **not** *set*, and take no action. If none of the flushing indicators was set, the ship fires normally.

4.2 set

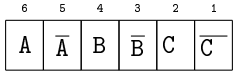
The set command is used to set the data latch, the flags, or the loop counter.



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.

The Sign-Extended Immediate instruction copies the Immediate field into the least significant bits of the data latch. All other bits of the data latch are filled with a copy of the bit marked "Sign."

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:



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 abort

An abort instruction causes a loop to exit; see the section on the Requeue Stage for further details.

4.5 head

A head instruction marks the start of a loop; see the section on the Requeue Stage for further details.

4.6 tail

A tail instruction marks the end of a loop; see the section on the Requeue Stage for further details.

Instruction Encoding Map