

AM14: Syntax

Adam Megacz

February 19, 2007

Abstract:

This memo proposes a new syntax for Fleet assembly code which handles the new BenkoBox bits of AM13. The new syntax “factors out” the common source in a group of source-sequenced instructions in order to simplify the syntax. This memo also points out how certain single instructions are equivalent to sequences of instructions, and how this symmetry is reflected in the new syntax.

1 Bags of Fibers

When FLEET was first introduced, it had no “*source sequence guarantee*.” At the time, codebags truly were unordered bags of instructions.

With the introduction of the source sequence guarantee, codebags are no longer completely unordered. Within a bag, the relative ordering of instructions sharing a source matters. If one were to give a *canonical form* for these codebags, it would be one in which all instructions with a single source appeared contiguously. We will use the term *fiber* for each of these contiguous sequences, since this is the most fundamental unit of sequencing in fleet (even more fundamental than threads).

With this new terminology in hand, we can now say that codebags are actually *unordered bags of fibers*, where a fiber is an *ordered set of instructions with the same source*. This conceptual simplification leads to a syntactic simplification: we can list the source only once for a given fiber:

```
inbox5:
  instruction dest1;
  instruction dest2;
  instruction dest3;

outbox99:
  instruction dest4;
```

2 Terminology

To avoid confusion, we will use different terms for the process of *emitting a data item* depending on whether an inbox or an outbox is doing the emitting:

- An *outbox* will send a datum via *the switch fabric*.
- An *inbox* will deliver a datum to *its ship*.

3 Instruction Equivalences

Every fleet instruction has three components: its *trigger component*, its *action component*, and its *acknowledgement component*:

1. An instruction's *trigger component* consists of optionally waiting for a token on the trigger input.
2. An instruction's *action component* consists of optionally ignoring, discarding, or taking a datum and then optionally delivering or sending it.
3. An instruction's *acknowledgement component* consists of optionally sending an acknowledgement token to some recipient.

That is, the instruction

- “wait for a trigger, then deliver a datum, then send an acknowledgement to Bob”

is equivalent to the three-instruction fiber

1. “wait for a trigger, then nop”
2. “deliver a datum”
3. “nop, then send an acknowledgement to Bob”

Indeed, if standing and counting moves are ignored and all three instructions share the same source, the programmer can completely forget about the fact that an instruction has these three components, and can instead simply pretend that there are two extra instructions: “wait for a trigger” and “send an acknowledgement.”

This simplifies matters considerably! It adds two new instruction types, but now each instruction is quite simple – just an opcode, source, and destination.

3.1 Reinstating Standing and Counting Moves

Since the previous simplification makes things so much nicer, we will accept a small amount of ugliness when reinstating standing/counting moves.

We will introduce a generic repeat n modifier ($n \in \{1 \dots \infty\}$), which can *syntactically* be applied to *any sequence of instructions that share a common source*. A *semantic* check applied after parsing will ensure that any repeated instruction sequence consists of at most three instructions, where only the first can be a trigger, only the last can be an acknowledgement, and at most one of the instructions may mention a destination.

3.2 Syntax

The following example illustrates all parts of the revised syntax:

```

{
  source.out:
    sendto dest.in;
    [*] wait, take, sendto dest.in;

  50: sendto dest.in2;
  10: sendto dest.in2;

  dest.in0:
    take;
    [3] ack source.out;
    [*] deliver, ack source.out;

  dest.in1:
    [*] take, deliver, ack source.out;
}

```

A codebag consists of zero or more fibers. A fiber may be either a *literal fiber* or *port fiber* and consists of the literal or port name followed by a colon followed by one or more instructions belonging to that fiber.

Each instruction consists of an optional repetition marker followed by one or more commands separated by commas.

Each command is one of:

- wait waits for a token on a trigger port
- nop does nothing
- discard waits for and discards a datum
- take waits for and latches a datum
- deliver transfers the latched datum to the ship (**inboxes only**)
- sendto $\langle \text{dest} \rangle$ transmits the latched datum to **dest** (**outboxes only**)
- ack $\langle \text{dest} \rangle$ sends an acknowledgement token to **dest**

When counting or standing moves are not in play, any arbitrary sequence of commands can always be rewritten into an equivalent stream of instructions using the five instruction bits of AM13 and the source-sequence (fiber) guarantee.

When counting and standing moves come into play, the assembler must enforce certain conditions on the combination of commands to which repetition is applied:

- only the first command may be a `wait`
- at most one instance of either `take` or `discard` may appear
- at most one instance of either `deliver` or `sendto` may appear
- only the last command may be an `ack`
- a `sendto` and `ack` may not appear in the same instruction

A repetition marker consists of square brackets containing either an integer (such as `[12]`) for a counting move or an asterisk (such as `[*]`) for a standing move.

4 Syntax Diagrams

Superseded; please see AM15.