

UC Berkeley Computer Science

Subject: Literals for FLEET
Date: September 20, 2005
From: Ivan Sutherland
UCIES #2005-is07

References:

UCIES# 2005-is02: FLEET – A One-Instruction Computer, Ivan Sutherland, 24 August 2005
UCIES# 2005-is03: Defining Some SHIPs, Ivan Sutherland, 24 August 2005
UCIES# 2005-is04: A Dozen Problems, Ivan Sutherland, 6 September 2005
UCIES# 2005-is05: Notation Questions, Ivan Sutherland, 12 September 2005
UCIES# 2005-is06: Some Ideas About Notation, Ivan Sutherland, 13 September 2005
UCIES# 2005-is08: Function or Addressing, Ivan Sutherland, 20 September 2005

PURPOSE

This memo records some ideas that came out of the discussion about literals in yesterday's class. I'm serving mainly as scribe. A companion memo, is08, describes the discussion of addressing vs function.

LITERALS

We have, somehow, to make available to FLEET hardware the particular values that the code in a particular bag needs. For example, the code may need a small integer, the actual value of the code bag descriptor for the next code bag, or a pointer to some item in memory.

One might put all such literals in main memory, seeking to make the literal problem vanish. This plan fails because instead of needing the literal itself, the code would need the address where it can find the literal. There's no way to get around having some literal mechanism.

We considered several proposals. One involved putting all literals in a known place in main memory. For example, if literals were stored in a block starting at memory address 1000, and 1000 was a number known to the hardware, a literal reference could be the offset from location 1000 to the place where the literal is found. The trouble with this proposal is that the literals block might be so large that offsets into it require many bits.

A second proposal chose to garner a few bits of a literal at a time by using the half dozen or so bits otherwise specifying the source address. If only half a dozen bits come with each instruction, many instructions may be squandered to accumulate enough bits to specify the literal.

SMALL INTEGERS

The class discussed and seemed to reach consensus that it's reasonable to use the source address bits to specify small integers. Thus there will be three uses for the source address bits and some bits to distinguish between them. The three choices are: to specify a source address, to specify a small integer, and to specify the offset of a literal from

This document is a product of a collaboration between Sun Microsystems and the University of California at Berkeley.. The ideas contained herein are freely available for any academic purpose.

the base of literals. If the source field contains the number 5, the three uses say: "MOVE the value from source number 5 to destination Y; MOVE the integer 5 to destination Y; and MOVE the 5th value from the literal block to destination Y. The small integer format will require some sign extension so that it can specify both positive and negative integers of small magnitude.

OUR LITERAL PLAN (other than for small integers)

The idea that seemed to appeal most depends on expanding the meaning of the code block descriptor. The expanded code block descriptor will define three positions in memory, namely the beginning and end of the code bag and the base address for any literals referenced in this code bag. Because we choose to reference literals by offset from the base address of the literal block, the code block descriptor need specify neither the length nor the end address of the literal block.

How the code bag descriptor encodes these three pieces of information may vary from implementation to implementation. I prefer to leave encoding details vague for now so that different implementations remain free to choose different encoding. For example, in one implementation of a code bag descriptor might specify a base address for the code bag, the length of the code bag, and an offset from the base to the lowest address of a table of literals. With such a specification the fetch unit can find the address of the end of the code bag with a single addition and the base address of the literals block with a second addition.

Here's another, possibly outrageous encoding that I offer for contrast. The code bag descriptor could indicate the address of the center of the code bag, the half length of the code bag, and the offset from the center pointer to the start of the literal block. This more symmetric description requires a subtraction and an addition to find the beginning and end of the code bag and a second addition to locate the literals block. The important choice we have made is not about details for encoding code bag descriptors, but that code bag descriptors will include three separate pieces of information.

The advantage of a three-part code bag descriptor is that it breaks up the literal block into manageable pieces. Code bags will probably require only a few literals each, and so we can afford to allocate only a few bits to the offset from the literal base to a particular literal. If some code bag needs more literals than the available offset permits, we can split that bag into parts that require fewer literals each. After all, two code bags are as good as one because everything in a code bag is concurrent anyway.

All this leaves one further question. What part of FLEET computes the actual address of literals? If each code bag has a different base for literals, the hardware must associate a possibly unique literal base with each instruction that references a literal. How the hardware does that is, of course, implementation dependent. One way is to compute a real address for each literal before the instructions of the code bag commingle with those of code bags already in the instruction pool.

We might think of a SHIP to handle literals. Given an instruction and the corresponding base of its literals block, such a SHIP would look up the literal in memory and provide its value to the switch fabric for delivery as the instruction directs. The only special equipment required is a way to pass the base of the literals block along with the instruction.