

# UC Berkeley Computer Science

**Subject:** A Dozen Problems  
**Date:** September 6, 2005  
**From:** Ivan Sutherland  
**UCIES** #2005-is04

## 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  
J.Ebergen, A. Chakraborty, and I. Sutherland, New Division Algorithms by Digit Recurrence, Asilomar Conference on Signals, Systems and Computer, 30 October 2004.

## PURPOSE

This memo offers a dozen problems for students of CS 294-4 to solve.

## INTRODUCTION

The FLEET architecture as described in is02 above contemplates a great deal of concurrency. Into this concurrency we must insert some sequential behavior. The problems posed here seem to me to involve this chore and cover a spectrum of difficulty. I offer these problems to seek student and auditor solutions. Yes, auditors should do the homework.

Students and auditors may work alone or in groups not to exceed three people. Each student or group should select one problem, seek a solution to it, and be prepared to describe his or her solution in class on 12 September. I'd like a paper document from each student (or group) with illustrations of the solution, a program listing, and not more than a printed page of English narrative. For program listings you will need to define symbolic names for sources and destinations. You may wish to make a labeled diagram of your SHIPs.

I am not yet pleased with my notations for code, so I leave you to define your own code notation. You will have to invent some way to name code bags, to record the contents of each bag, and to express literals. I want to see your ideas about notation!

You may use any small collection of SHIPs you wish. Simpler ships are generally better. You can use multiple copies of SHIPs like adders, FIFOs and memory control units if you wish. You can start with the SHIPs defined in is03 above, but feel free to define other SHIPs if you need them.

What is the central problem? What SHIPs are required? What are their interfaces? How is memory used? What assumptions do you make about performance? What was hard about fitting this problem into the FLEET architecture?

Next to each problem is Igor's judgment of its difficulty in parenthesis e.g. (**hard**) and Ivan's judgement of its difficulty in brackets, e.g. [ **easy** ].

---

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.

**Problem 1 – Sorting ( messy ) [ easy, requires detail ]**

Figure 11 of the second picture suggests a pipeline solution to the sorting task. However, it depends on a particular sorting SHIP, and it ignores the code bag descriptors.

Your task, should you decide to accept it, is to do another solution substituting the code bag descriptors for the pipeline connections. You will need to write the code, however, where the figure only suggests what the code might be.

**Problem 2) – Subroutine call ( requires thinking ) [ easy, answer in English? ]**

The subroutine makes it possible to encapsulate a complex behavior into a single package. Every computer I know of has a subroutine mechanism, and every programming language has a procedure call. Modern machines tend to stack parameters for the subroutine and hold return addresses also on a stack. Before stacks were invented, computers passed a return address into the subroutine which was, in turn, responsible for safe storage of the return address, often in a memory cell that was part of the subroutine. Recursion was not possible in such machines.

How should a FLEET machine handle subroutines? Indeed, is the idea of subroutine “return” so sequential as to be meaningless in the FLEET context? How might one pass parameters?

**Problem 3) – Defining a STACK SHIP ( easy ) [ hard – sequential behavior ]**

We can build an asynchronous hardware stack that will PUSH and POP very quickly, and yet which manages to limit its high bandwidth behavior to near the top of the stack. By using several registers at each level in the hardware and invoking hysteresis, such a stack can confine local PUSH and POP behavior to its upper levels. Its lower levels need push and pop only in response to long periods of relatively intense push action or long periods of relatively intense pop action in the upper levels. It's a sort of multi-level cache for stacked data.

If we assume such a hardware device, what interface might we give it as a SHIP? The main thing here is to understand what is essentially sequential about stack behavior. PUSH, PUSH, POP produces a different result from PUSH, POP, PUSH, even though both leave the depth of the stack the same.

**Problem 4) – Transpose a matrix ( messy ) [ easy ]**

Suppose we have a matrix of integers with 100 rows of 200 entries each. You get to define how that matrix is arranged in memory. Now write code that will transpose your matrix into one of the same format with 200 rows of 100 entries each.

It's probably simple to deal with a single row as a pipeline process. However, how do you plan to reload the stride SHIPs between rows? You may define whatever simple SHIPs you need.

**Problem 5) – Copy a block of memory ( options ) [ easy to hard ]**

Given a block of memory with a million entries, copy it into another block. You may assume that the blocks do not overlap.

Nothing in this problem says that the block must be copied sequentially. Can you use concurrent programming to advantage? What assumptions do you make about caching in the memory system?

Now consider how to get better performance if the memory read and write processes are subject to random delays. Can you use a high speed FIFO to smooth out the flow of data?

### **Problem 6) – Dot product ( hard ) [ easy ]**

Two vectors of 3 million integer elements each rest in memory. Compute their dot product.

Nothing in this problem says that the partial products need be computed sequentially. Can you use half a dozen multiplier and adder SHIPs to advantage to make your task run quicker? If you access multiple parts of each vector concurrently, what assumptions must you make about the memory system to choose your sub-vectors?

Sadly, the vectors contain long strings of large positive values and long strings of large negative values, even though the dot products are known to be near zero. What sort of accumulator SHIP might maintain numerical accuracy?

### **Problem 7) – Division ( hard ) [ too hard for me ]**

I have been wrestling with a division algorithm. The third reference above presents a particularly clear view of division. You can find it in the (pre-2005) publications section of: [research.sun.com/async/](http://research.sun.com/async/)

You may wish to confine your attention to the simplest of the division algorithms, namely the one we learned in grade school.

I can see how multiple adders might concurrently compute the values needed at each step. We can use comparison of the remainder and the divisor to pick the useful precomputed value and discard the others.

What I do not yet understand is how to ensure that sums to be discarded don't get used by premature entry of the next code bag into the instruction pool. In other words, how do we know that everything is ready for the next cycle?

Of course, one could just specify a SHIP that can divide. Instead, I'd like to use this example to understand how to get the needed sequentially into otherwise concurrent code. Help me out here with a simple solution, please.

### **Problem 8) – Find a person's name in a long string of text ( special SHIP? ) [ easy with simple ships ]**

Maybe this problem has two parts: First, find candidate sub-strings by comparing a single character with the first character of the sub-string. Second, compare the text following the candidate location on a letter-by-letter basis against the person's name to see if the candidate is, indeed, a match. Maybe these two processes could proceed concurrently.

OK, so do it. The devil here is in the details.

**Problem 9) – Vector sum ( illuminating ) [ easy ]**

Two vectors of 2000 elements each rest in memory. Add them together to produce their vector sum.

Nothing in this problem says that the component sums need be computed sequentially. Can you use half a dozen adder SHIPs to make your task run quicker? If you access multiple parts of each vector concurrently, what assumptions must you make about the memory system to choose your sub-vectors?

**Problem 10) – A keyboard SHIP ( so how does a keyboard work ) [ easy ]**

I want to connect a keyboard to my FLEET computer. Define a SHIP suitable for this service.

Recall that the keys on a keyboard go both up and down. For some keys only the down stroke matters, for others both matter. For example, both up and down matter on the shift key, because for it, the duration of depression matters. In fact, the duration of depression of any key may matter if it serves as a “repeating key.” So what is your SHIP going to do for each keystroke? How does it distinguish one key type from another, or does it?

**Problem 11) – Downsizing a picture ( easy ) [ easy ]**

I have a gray scale photographic image in memory that consists of 1 million pixels, 1000 rows by 1000 columns. I want to downsize it by a factor of two in each axis to make a picture of 500 rows by 500 columns. Averaging groups of four pixels will serve.

Show a set of SHIPs and some code suitable for this task. As with every matrix problem, the inner iteration (for rows, say) is fairly easy, but how about the outer iteration? Can you use one or more FIFOs to store the row parameters so as to make the row setup concurrent with the inner loop?

How would you deal with a more complex calculation of weighted sums over a larger area, e.g. on a 4x4 array of pixels?

**Problem 12) – Reader’s choice ( way too hard ) [ easy for the right problem ]**

Put your own favorite problem here. You will need a problem statement as well as a solution.