September 29, 1988

From: John Backus

Plan for implementing an FL system

General

The following material details a plan for implementing a compiler for the FL programming language, along with a user-friendly system for the RT that makes it possible for users to write, debug, and run FL programs on the RT under AIX.

It should be borne in mind that there are still some research problems that must be solved before a feasible system based on FL can be built. In particular, one of the principal features of the FL language is that often the best and clearest FL program for a problem is easy to write because the user does not have to think about efficiency. In consequence of this property, many programs would be unacceptably slow running without some extraordinary, "housekeeping" optimization (i.e., in some cases this optimization would speed up a program by three orders of magnitude; "housekeeping" optimization eliminates data rearrangements that are basic FL operations).

We have already made good progress in developing a strategy for doing this optimization, but hard problems remain. Therefore it is difficult to be precise about the time required to complete our prototype system. We will do only the essential optimization that must be done to get FL running at an acceptable speed. (No large implementation effort should be expended on FL unless and until it can include this essential optimization level.) On the other hand, for our first prototype, we will avoid expending the effort required to develop additional techniques that might lead to a final additional improvement by a factor of perhaps 2 to 4 beyond the essential level of optimization discussed above.

Current status

The language.

The FL language is now stable and a new language manual is in progress.

Bootstrap compiler.

We have built an FL-to-LISP compiler on VM. The purpose of this compiler is (a) to allow us to run some FL programs, and (b) to allow us to write large portions of the prototype compiler (from FL to C) in FL itself. To use this bootstrap compiler to obtain a working compiler from one written in FL, we would proceed as follows:

1) Run the bootstrap compiler with the prototype compiler (an FL program) as input. (The bootstrap compiler says how to translate FL programs into LISP programs.)

   Result: A LISP version of the prototype compiler.

2) Run the LISP version of the prototype compiler with another copy of the prototype compiler (an FL program) as input. (The prototype compiler says how to translate FL programs into C programs.)

   Result: A version in C of the prototype compiler (after some tuning of parts of this C program, it would become the actual working C version of the prototype compiler; it would then need only to be compiled by the C compiler to run on the RT.)

It is not clear how much of the prototype compiler can be compiled in this way, because the inefficiency of the bootstrap compiler may make certain parts of the prototype uncompilable, even with unlimited time available. We have already completed part of the conversion of the bootstrap compiler from VM-LISP running on VM to Common Lisp running on the RT under AIX. (This conversion is a large task.)

Current status of implementation plans for prototype compiler.

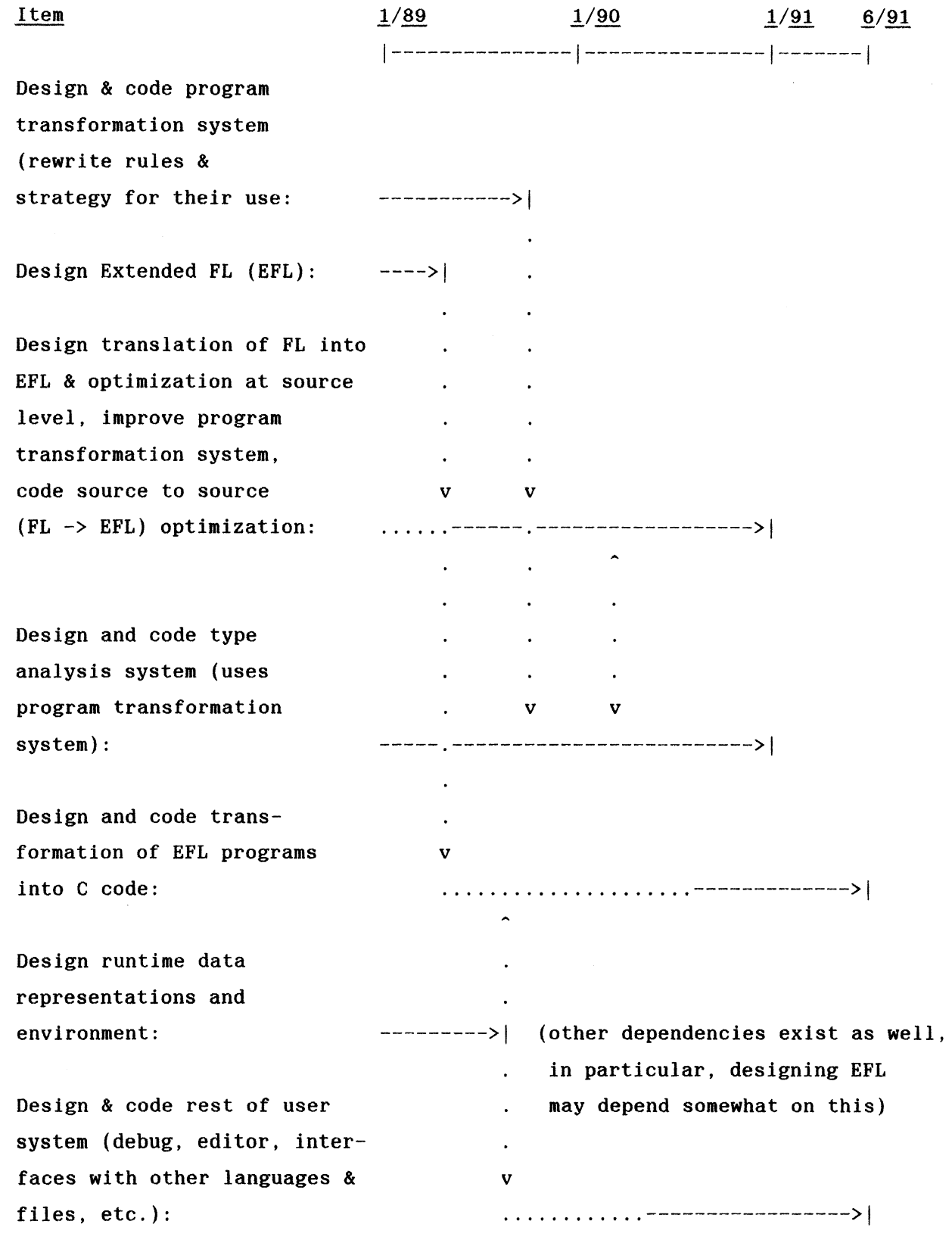
We are in the process of designing the data structures and data descriptors for the data that a compiled program will operate on at runtime. We are also

in the process of designing the elements of Extended FL (EFL) in which the final, optimized version of an FL program will be expressed (these EFL expressions will have obvious efficient C counterparts). We have begun to develop the methods by which FL programs can be converted to efficient EFL programs (i.e., the mathematical laws and thoerems, and the strategy for using these to do the conversion).

Implementation plan for FL system

Development schedule

I list below some of the key things that must be done to complete the prototype system and indicate a rough estimate of the time over which we now expect to work on each item (vertical dots indicate strong dependencies, but there are others as well). It is important to note that there will be a lot of interaction both ways between design and coding.


Item
1/89
1/90
1/91
6/91
Design & code program transformation system (rewrite rules & strategy for their use:
Design Extended FL (EFL):
Design translation of FL into EFL & optimization at source level, improve program transformation system, code source to source (FL -> EFL) optimization:
Design and code type analysis system (uses program transformation system):
Design and code transformation of EFL programs into C code:
Design runtime data representations and environment:
(other dependencies exist as well, in particular, designing EFL may depend somewhat on this)
Design & code rest of user system (debug, editor, interfaces with other languages & files, etc.):
Convert FL-to-Lisp bootstrap compiler to Common Lisp and

improve its speed: ----->| (note that most of the other items depend on this, since much of the coding is done in FL)

|---|---|---|

Item 1/89 1/90 1/91 6/91

Write new language manual: Complete before 1/89.

Comments on Schedule.

I note that there are five time-lines for various efforts that all begin at 1/89; most of these have a heavy design component at the outset. Some of the basic thinking has been done and/or will have been done by 1/89. It is difficult to predict when some design efforts will reach a stage at which programmers will be helpful. However, we hope to have good enough programmers so that we can involve them in the design efforts as well.

Personnel

Current.

5 RSMs: Backus (Mgr), Aiken, Lucas, Williams, Wimmers

1 Programmer: Linden (starts full time 12/1/88, presently part-time)

Thom Linden, an Advisory Programmer, joins us at the end of November (he is already helping us part-time).

We expect to have the help again of two excellent students next summer who worked with us this last summer. One of these (Brian Murphy) is an MIT Co-op student who we expect to be with us for six months. The other (Bill Griswold) may or may not come depending on his Thesis topic.

Future. Programmers: We estimate that before the projected completion date about 6/91 we will need three additional expert programmers at about the following times:

(1) ASAP (to work on Bootstrap compiler and/or program transformation system, the two most critical components)

(hopefully this will be Paul Tucker from the Menlo Park Laboratory, working either on their payroll or ours, see below).

(2) 2Q89 or 3Q89 (to work on FL -> EFL transformation system)

(3) 1/90

We have made a presentation (9/28) to the staff at Menlo Park on FL, and hope to convince them to consider FL as a potential future AI product and to contribute Paul Tucker, another expert programmer, to work with us while on their payroll. We expect to hear from Menlo about Tucker's availability within 2-3 weeks. If Menlo Park does not decide to let Paul work with us on their payroll, we should find a slot for him on ours and try to make him an offer as soon as possible.

Technical writer: We will need a good writer to help prepare the best possible documentation for the system at about the following time:

3Q90

Equipment

Current.

We presently have 6 RT's (counting Thom Linden's, which we will inherit)

We want to have a "standard RT" for each member of the group capable of running the Common Lisp Development Environment. This means an RT with:

Standard RT:

Megapel display
16M memory
3 E70 disk drives
Common Lisp Development Environment (software)
AIX (software)

To achieve this for our current RT's we need:

3 E70 drives

7 8M boards

Future

Our future equipment needs are:

1 Standard RT for each new person
2 Standard RT's for student use
1 RT for a disk server (minimal display, 3 E120 drives, 4M memory)
1 4216 printer & adapter

This adds up to the following requirements at the given times:

1 Standard RT (very soon)
3 Standard RTs (by about 5/89)
1 Standard RT (about 1/90)

1 RT disk server (ASAP)
1 4216 printer (ASAP)