LITTLE Newsletter # 31

January 9,1974

J. Schwartz,
A. Stein

Representation of BALM in LITTLE

1. Introduction.

The BALM semantic environment provides recursive procedures and allows procedures to be ordinary, assignable semantic objects, which LITTLE does not. This raises the question of how these features are to be provided when BALM is translated into LITTLE. Unless the whole of a BALM source code is represented by a single LITTLE subroutine (which is hardly compatible with the incremental character of BALM) this requires some semantic extension of LITTLE. This note will suggest what seem to be fairly minimal sufficient extensions.

These extensions may at a later date be added as new LITTLE statements, and cause the compilation of in-line code. For present use however we will suggest an off-line technique which exploits the known structure of the LITTLE linkages and makes use of a very few quite simple assembly language routines.

2. Representation of BALM Procedures.

A BALM procedure will be represented by a code block generated by the LITTLE compiler and the system loader. The starting location at which a newly compiled group of code blocks is to begin will be chosen by the garbage collector which will pass this starting address to the loader. Presently these blocks are non-relocatable, and hence not garbage-collectible. For this reason, they will probably be placed for the time being immediately below the BALM stack. Small assembler charges (available as an assembler option for generating this type of code) can make these blocks relocatable; when this is done, they can be represented by standard garbage-collector many-word blocks consisting entirely of 'header'

area with no 'pointer' area (see On Programming, Installment 1, p. 50).

When a procedure F is set up, its root word (see the diagram in Installment 1, p. 52) will be established by executing a macro (perhaps later to become a LITTLE statement)

(1) RECENTRY(V,F).

This macro will place the entry address of F, with the additional garbage-collector boiler-plate shown in the cited diagram, into V, which must be a variable having the same size as heap and stack entries. (Note that F is simply a unique dummy name generated for loader purposes; BALM procedures are in principle values and as such have no inherent names). For the time being, this macro can expand as

(2)
```
CALL ENTROUT;
CALL F(V);
```

where ENTROUT is an assembly language routine which, knowing what code the statement CALL F(V) will generate, uses this information to get the entry address of F, forms the required root word, stores it in V, and then on return jumps over the call to F(V).

BALM source code

(3) V = PROC(*args*) *body*;

is then compiled into

RECENTRY(V,F),

with F being a unique name generated for the PROC standing on the right of (3). The code actually representing the PROC is of course collected elsewhere. All the 'main program' code collected from a single BALM 'compute block' will be collected into a procedure represented in the same way, but called by the BALM master compiler-controller responsible for reading and compiling source code. When the 'main program' terminates it will return control to the master compile-controller.

3. Recursive Calls, recursive returns.

The invocation of a BALM procedure with root word V is accomplished by executing a macro

(4) CALLREC(V);

At a later date, some version of this might also be made into a LITTLE statement. For the present, it can expand simply as

(5)
```
INPUT1 = V;
CALL KALLER;
```

where INPUT1 is one of the global SRTL variables likely to be 'registerised' and KALLER is an assembly language routine. This routine is 'minimally recursive': it will stack the address to which it would return on the SRTL stack (without checking for overflow: this check can be combined with the overflow check associated with other subroutine-entry stacking operations).

Returns from recursive routines can then be made by executing a macro

(6) RETREC;

A version of this might at a later date become an additional LITTLE statement. For the moment, it can expand simply as

```
CALL RECTRTRN;
```

where RECRETRN gets an address from the top of the SRTL stack (without needing to check for underflow) and jumps to this address.