The views, conclusions, or recommendations expressed in this document do not necessarily reflect the official views or policies of agencies of the United States Government.

This document was produced by SDC in performance of contract SD-97

2/19

TECH MEMO *a working paper*

System Development Corporation / 2500 Colorado Ave. / Santa Monica, California

TM-2260/001/00

AUTHOR M. Levin
M. Levin

TECHNICAL S L Kameny
S. L. Kameny

RELEASE SM Cooney
S. M. Cooney

for
D. L. Drukey

DATE 2/19/65 PAGE 1 OF 5 PAGES

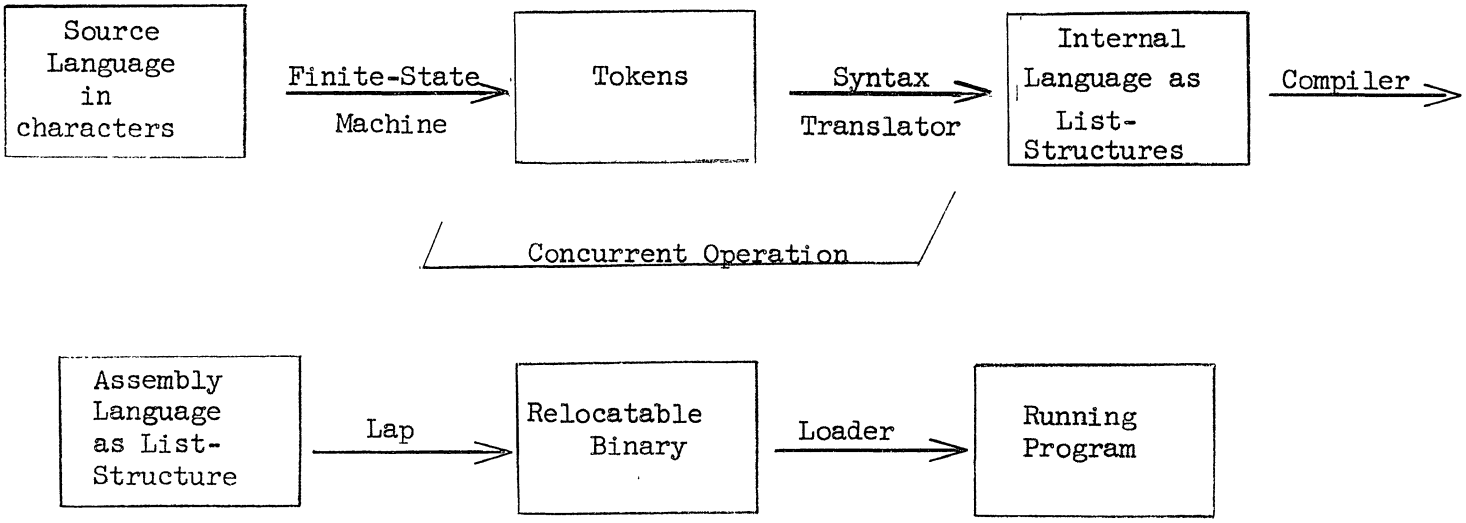
LISP II PROJECT

MEMO NO. 4:LAP

ABSTRACT: This is the fourth in a series of working memos documenting LISP II development, The position of the LAP assembly language compiler in the LISP II system is specified by flow chart--including alternative approaches to converting a source language into a running program. The LAP terminology is defined.

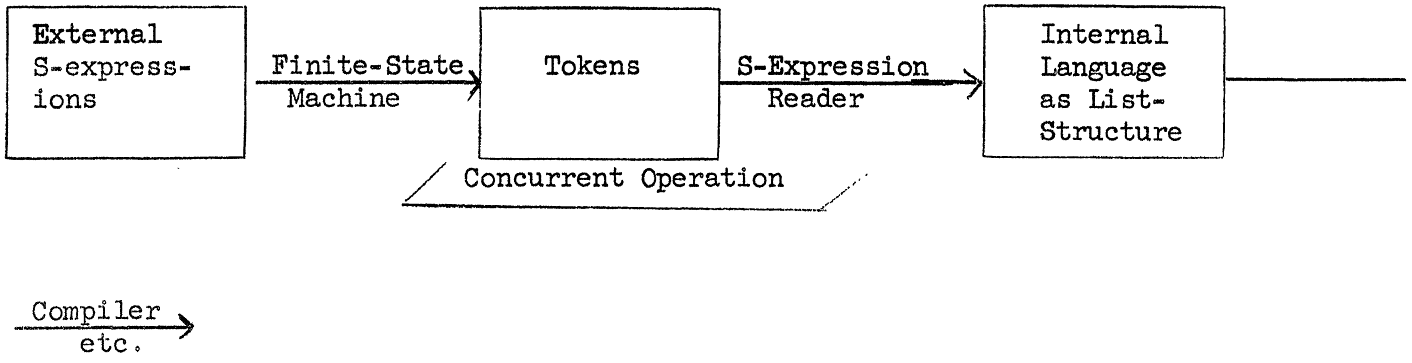
I. OVERVIEW OF LISP II

There are five distinct processes that are involved in converting source language into a running program.

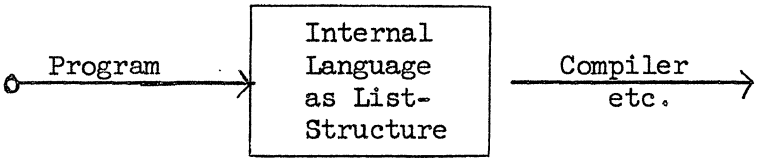


There are various alternative routes not shown in this diagram.

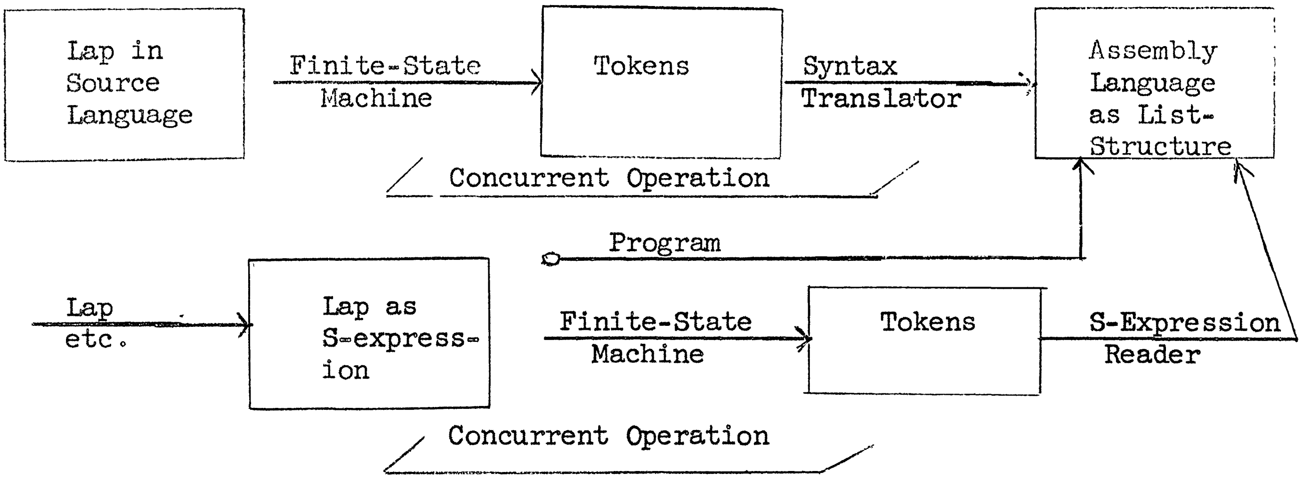
1. LISP II internal language can be input as S-expressions, bypassing the syntax translator.



2. Internal language may be generated by a program that produces programs. There is no input in this case.



3. Assembly language may be introduced as source language, or S-expressions; or generated internally.



II ASSEMBLY LANGUAGE

The argument of LAP is assembly language as an S-expression. The value of LAP is relocatable binary which is an array. Unlike most assemblers, the input to LAP is not a linear list but a nested list with indefinite depth. In internal language its syntax is as follows:

$\langle$assembly language$\rangle$ ::= $(\langle$part $\rangle^n)$

$\langle$part$\rangle$ ::= $\langle$instruction$\rangle$ | $\langle$macro$\rangle$ | $\langle$pseudo-instruction$\rangle$ | $\langle$label$\rangle$

A label is an identifier. It gives a symbolic name to the following part.

A macro is a list beginning with its key word. It is expanded to a list of parts which is concatenated into place. Thus a macro which looks like a single instruction can be expanded into several instructions inserted at the same level.

$\langle$instruction$\rangle$ ::= $(\langle$op-code$\rangle$ $\langle$field$\rangle^n)$

The op-code is an identifier naming an instruction, e.g., BSX. The number of fields is machine-dependent. They are evaluated, shifted left if appropriate, and logically or'ed into place.

The types of field are:

1. A number
2. $ meaning current location
3. An identifier which could have several meanings as a symbol
4. (QUOTE α) where α is an S-expression. This produces a quote cell.
5. A list of fields which is evaluated as their sum.

Each pseudo-instruction is a special case.

1. (ORG ⟨field⟩)

   An assembly has ORG only if it is absolute and is going directly into core. Each ORG starts the subsequent program assembling at the location specified by the field.

2. (FUNCTION ⟨name⟩ ⟨formal parameter list⟩ ⟨part⟩^{n})

   This pseudo-instruction generates code for a closed subroutine. The declarative information following the word FUNCTION is of the same format as if this were a declaration of a function in internal language.

   a. A brick is planned with space for all parameters. They can then be referenced symbolically within the parts that follow.

   b. Instructions are generated for establishing a brick using MOVE@.

   c. The parts are assembled.

   d. The exit for the subroutine is assembled.

3. (PROG ⟨program variable list⟩ ⟨part⟩^{n})

   This is similar to FUNCTION, but creates open code corresponding to a block rather than a procedure.

   In the case of (FUNCTION ⟨name⟩ ⟨formal parameter list⟩ (PROG ⟨program variable list⟩ ⟨code⟩^{n})) where the PROG is the only part within the FUNCTION, only one brick is created serving both purposes.

4. (⟨number⟩)

   This assembles into a number.

   The definition of fields for function names, own variables, and global variables is left unspecified in this memo.

III. LAP IN SOURCE LANGUAGE

LAP ⟨part⟩ ; ⟨part⟩ ; ... END

1. Labels are followed by colons.

2. Parts are separated by semi-colons.

3. Fields are separated by commas or spaces.

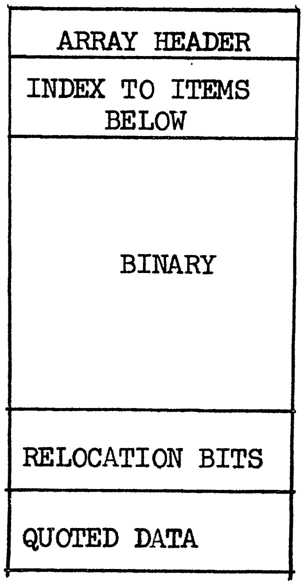
4. ⟨procedure heading⟩ $\{;\ \langle\text{part}\rangle\}^n$ END

e.g., REAL FUNCTION FN(U, V); REAL U; ⟨part⟩ ; ⟨part⟩ END FN

5. ⟨block heading⟩ $\{;\ \langle\text{part}\rangle\}^n$ END

e.g., A: BEGIN (4) REAL U, V; GLOBAL V, W; ⟨part⟩ END (4) A

IV. RELOCATABLE BINARY


ARRAY HEADER
INDEX TO ITEMS BELOW
BINARY
RELOCATION BITS
QUOTED DATA