MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Artificial Intelligence Memo. 132 July 1967.
Program Memo.

Additions to LAP

John L. White

In addition to the description on page 13 of A.I. Memo. No. 116A LAP has the following features:

(1) Current Assembly Location Reference.

The atom "*" has as SYM value during assembly an integer which is the current cell address being assembled into. Thus (JRST 0 *) is a well known infinite loop equivalent to A (JRST 0 A).

(2) Assembly Time Arithmetic.

When LAP encounters a non-atomic argument in the position normally occupied by the address part of an instruction, and it is not one of the recognizable forms (QUOTE atom) (E function) or (C constant), then the assembly time values of the list members are summed and this is the quantity assigned as address. Thus (JRST 0 (* 1)) is a do-little instruction roughly equivalent to TRA * + 1 in FAP. Note that in

```
A   (MOVE  1  (H  1))
        :
H   ( 4 0 0)
    (-9.0)
```

H has not been assigned a location when (H 1) is evaluated, but LAP remembers this accurately and when H is finally assembled, the (MOVE 1 (H 1)) instruction is correctly modified.

(3) Constants.

LAP will correctly assemble into a location the constant indicated by the number appearing in any 1 - list. For example, in (2) above, after assembly, H contains 000000000402 and H + 1 has 571337777777. The "C" feature for generating constants is similar to the "literal" feature of FAP. Thus (MOVE 1 (C 104)) is analogous to CLA =104, that is, the address part assembled for (MOVE 1 (C 10.4)) is some new location into which is assembled a constant (10.4) as mentioned above.

(4) Multiple Entry Routines

A pseudo-op has been provided for declaring the current assembly location to be an entry point with another name. For example:

```
(LAP   SIN   SUBR)
(PUSHJ   P   NUMVAL)
(FSUB    1   (C 3.14159)
(MOVN    1    L)
(ENTRY   COS)
   .
   .
   .
(POPJ   P)
```

utilizes the relation $\sin\alpha = \cos(\frac{\pi}{4} - \alpha)$ to save some space in storing SIN and COS. More importantly, it allows two LAP-written routines to be assembled together so that each has access to the others symbols.

ENTRY does not use up any assembly space so that if SIN's entry point were 14732 in the above example then COS's entry would be 14736.

CAVEAT: Features (1) and (4) above were not in the LAP on the LISP SYS tape until about July 12, so that assemblies utilizing these features must be performed with one of the more recent copies of LAP. This is especially important for Vision people since some vision routines will eventually employ them.

(5) Defined machine operations in LAP

(i) Move type instructions
MOVE, MOVEI, MOVEM, EXCH, MOVNI

(ii) Half word instructions
HRRZ, HRRZ@, HLRZ, HLRZ@,
HRRZS@, HLLZS@, HRRM@, HRLM@

(iii) Conditional branch instructions
JRST, JSP, JUMPE, JUMPN
SOJE, SOJN, CAIE, CAIN, CAME, CAMN

(iv) Arithmetic instructions
ADD, SUB

(v) Stack instructions
PUSHJ, POPJ, PUSH, POP

(vi) U U O instructions
CALL, JCALL, CALLF, JCALLF,
CALLF@, JCALLF@

(vii) Miscellaneous
TDZA, DPB, CLEARM, CLEARB

All assembly time symbols are handled the same way so one should be sure not to use instruction names as address labels. Additional definitions may be done by the user with a call to OPS (which is a FEXPR defined at the same time as LAP). Any number of symbols may be given an assembly time value as in the following example:

```
(OPS MOVE 200000000000
     JRST 254000000000
     FADM 142000000000
     FADM@ 142020000000
     DANDY 105
       P    14
     BLT  251000000000)
```

The way LAP puts together a machine command is perhaps of interest to some. It is: (x y z w) becomes $x + 2^{23} \cdot y + \text{Remainder } [Z, 2^{18}] + 2^{18} \cdot w$