II File Stuucture -- Random file Blocks

File Header

Blocks

1. Blocks 2K words

Header CHECKSUM

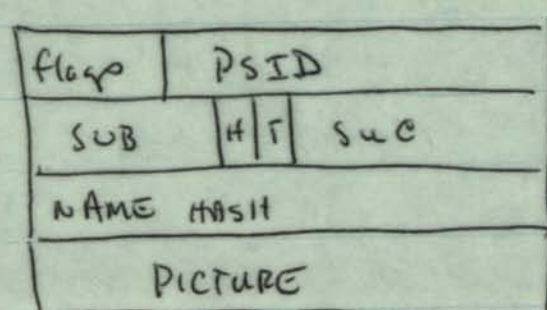
free space pointer

collection & compaction flags

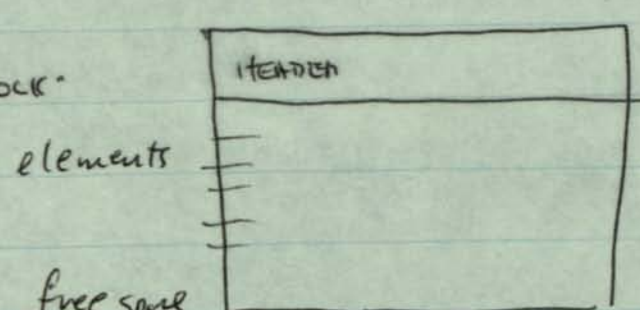
Rest special format

HEADER mapping tables -- must be resident

2. RING ~~B~~ ELEMENT



RING BLOCK:

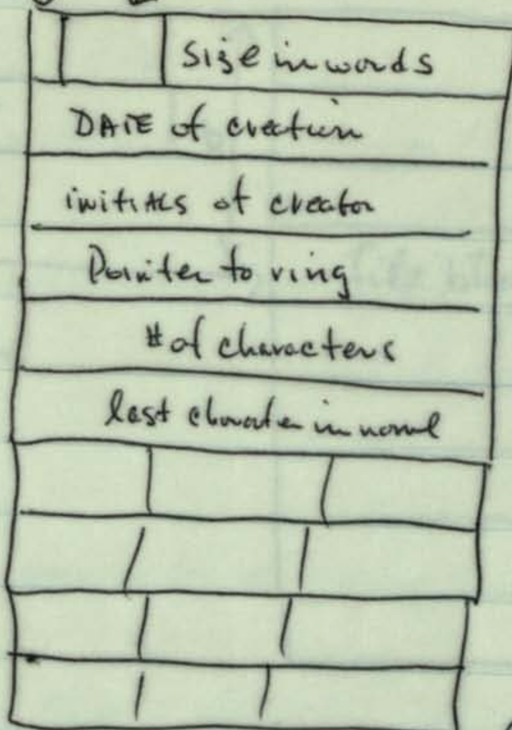


N.B. missing elements!

for header space

3. TEXT ~~BLOCK~~ ELEMENT

JUNK FLAG

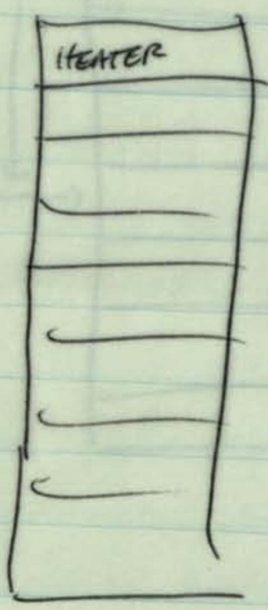


other flags
has picture
has lines
etc.

Starts with -1
TEXT

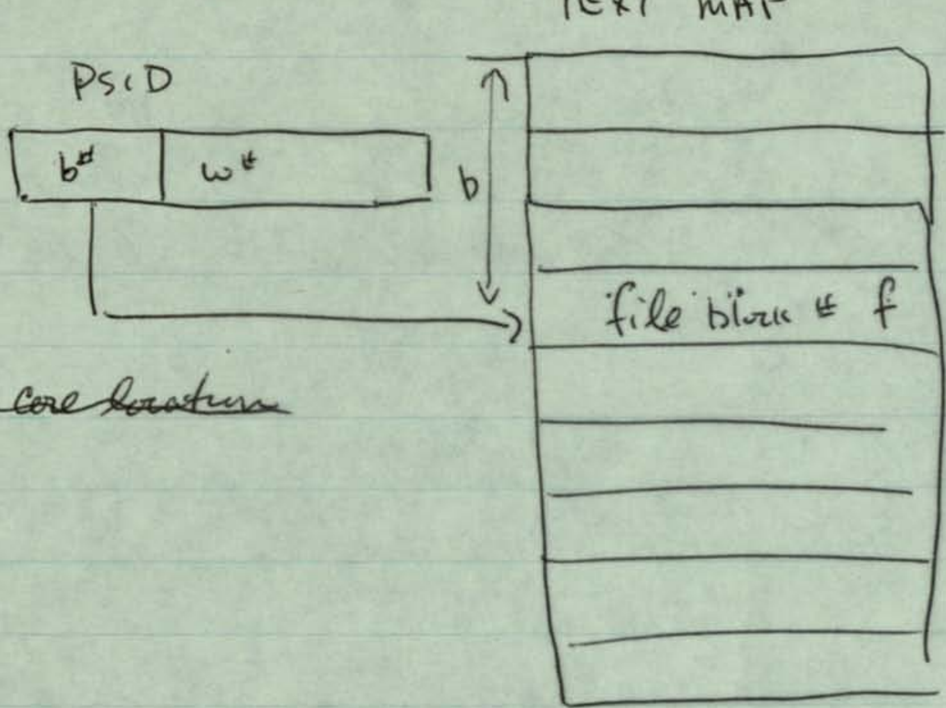
ends with -1
(used in pattern copies)
need both counts & boundary conditions

TEXT BLOCK

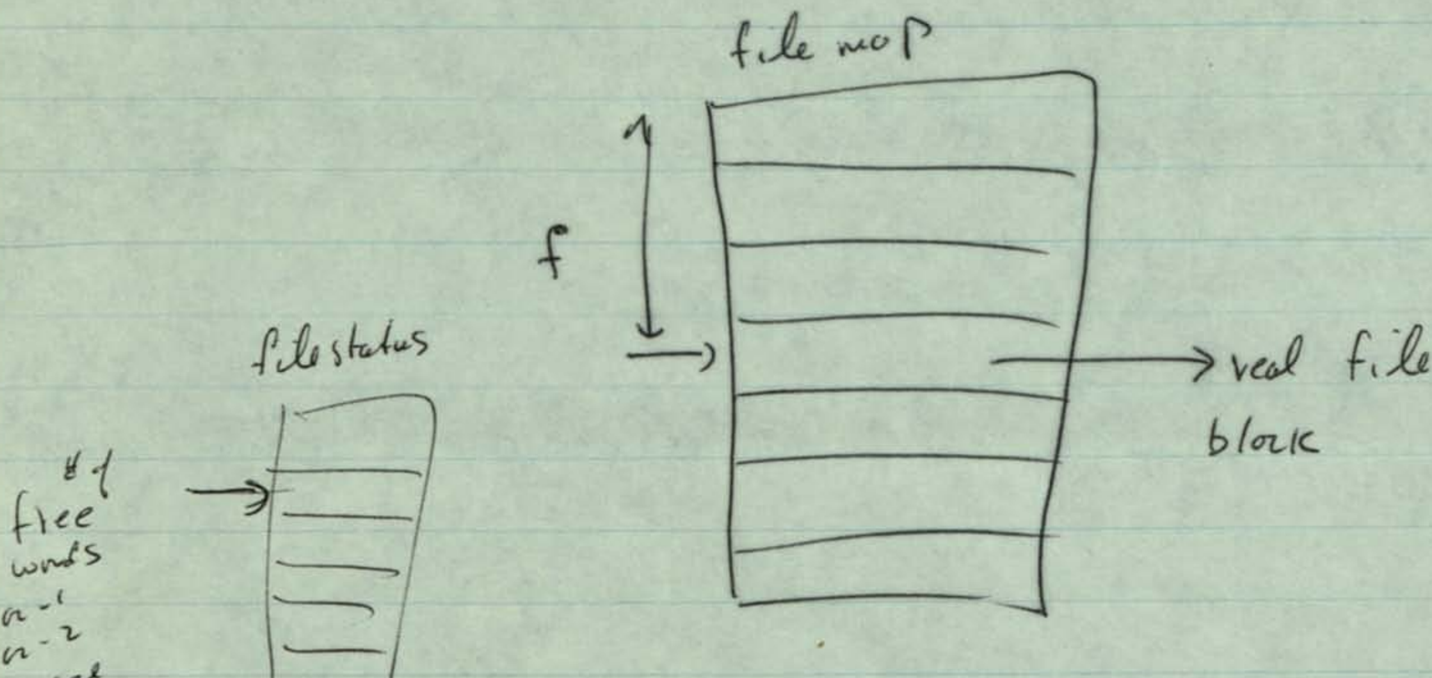


BLOCKS
(Some JUNK some not)

4. MAPPING BLOCKS

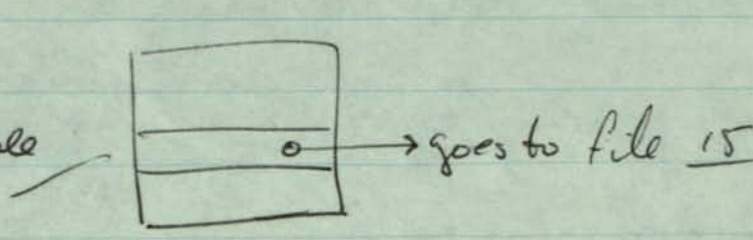


~~on core location~~



5. Real File

program table | in core table
blocks in core



II Good ideas.

1. isolate text

    minimizes movement for edits
    file references
    can search for hash quickly
    (few file references)

2. have complete checking mechanism

    check all structural changes
    forbid certain deletes

    have minimum of pointers in any
    or this is too complicated !!
    (must be able to restore file
    with program)

3. whole thing puts upper bound on
# of file references per edit.
so run time is linear fcn of job

III Access routines

1. A-STRING
   INITIALIZE (word 1 ← -1)
   GET n^{th} CHARACTER
   APPEND CHARACTER (may get error)
   initialize co-routine fetch
   fetch next character
   move whole string

2. RING #
   get & put all stuff
   free ~~the~~ elements

   check any structure changes

3. TEXT
   Same thing
   automatic compactification
   Save & restore search pointers

4. File Block themselves
   (only 5 in NLS)
   freeze thaw mechanism for speed.

IV Character Strings

character set
plus special ~~un~~ preceeding character

V referencing
by any thing
NAMES / numbers
NAMES Automatically recognized during
edits & hash updated in ring

SECTION 4

VI DISPLAY

1. CHAIN GENERATION (conceptual only)

given a starting point generate ~~[illegible]~~ linear list.

a. tree ~~[illegible]~~ structure (generate #'s on fly)

b. link following (or jump in tree)
(uses tree as default)
(uses specific pattern / con chage thing)
(ugly to get #'s for a statement)

2. FILTERS

1. Structural
depth
breath (questionable use)

2. contextual
(according to certain patterns)

3 insert.

VII PATTERNS

REGULAR EXPRESSIONS with FANCY SYNTAX
fast 1. to parse
2. to check for (minimal backup)
i.e. uses non-deterministic
representation with stack
(more compact just as fast)

EXAMPLES of language

look out for
1. different kinds of strings
with / without special characters
& shift

2. can be done with side effects
some restore modes
forward backward

3. handy to have search routines
default correctly for failure
of parse rule.

TABLE OF CONTENTS

insert so it can come out automatically

4e A 2nd class of gen lab

4f "enough ends" to match "

7 production of actual machine code.

12 Future research goals?

This section deals with the extensions made in LOT to the syntax equations of META II. The changes include attempts to speed compiler recognition of syntactical units, inclusion of error diagnostics in the syntax equations, reserved words, inclusion of "control-card" options, and special constucts to handle a few of the more perverse features of PL/I.

4a In the LOT system there is a classification of subexpression types reflected in the LOT syntax equation:

```
subexpression = ( element *c(f,1) / generator /
             control / directive )
  $( element *c(e) / generator / control / directive
```

4a1 The syntax equations for the last three classes each use a unique character to preface the entire class. A generator is always preceded by a "*" (such as *c or *d), a control is preceded by a "+" (+list +punch etc.), and a directive by a "-" (-list -punch etc.). This prefix character makes possible reduced scanning of non-applicable equations, and the suppression of redundant test instructions in the compiler.

4a1a For example, the class of generators includes the *c and *d constructs for output code generation and label definition. The LOT equations are:

```
generator = "*" (
        "c" "(" codefield $( "," codefield ")"
    /
        "d" "("
              ( "*r" "," / .empty )
              ( "*1" / "*2" / "*" )
        ")"   )  ,.
```

4a1b When these equations are driving the interpreter during compilation, the input stream will be checked for a "*" whenever a generator may appear in the syntax. If it is not found, an exit will be made from the entire statement. Only if the "*" is found will an attempt be made to find a "C" or "D" (which are required to follow if the program is syntactically correct).

4b Error diagnostics concerning syntax errors are included in the syntax equations. The diagnostics contain the message to be printed, a string (such as a semicolon) to be matched against the input stream before compiling is to resume, and the name of the next syntax rule to be applied.

4c Reserved words for a source language and operation mnemonics for the target machine's assembly language may be included in the syntax equations of a compiler in LOT. The reserved words are checked by the identifier routine which will then not accept them as identifiers on the input stream. The operation code mnemonics are placed in a table for the use of the code generation routines.

4d The options traditionally associated with control cards (such as whether to list the source code) have been incorporated into the syntax either as "controls" which allow the compiler to set options as it is analysing a program, or as "directives" which allow the program being compiled to exercise control.

4d1 A directive has the form "-" followed by a word indicating the action desired. For example -LIST will cause the program in which this directive appears to be listed on the output device during compilation. Since directives are elements in LOT, directives may appear almost anywhere imbedded within the program.

4d2 Controls have the form "+" followed by a word indicating the action desired. Controls have no immediate effect when they are compiled (as directives do) but cause the generation of instructions which cause the compiler to effect the operation when it is executing. Thus a compiler can cause tracing or listing or a program it is compiling dependent on its own location in the syntax equations.

*L   of   and the generation syntax of *L

4e The special control~~s of +SAVE, +LINK, and~~ -ENDKLUDGE, have been added to LOT specifically to aid the implementation of PL/I declarations and the "enough-ends-to-match" construct.

*L

4e1 ~~+SAVE and +LINK~~ provide a second set of transfer addresses (in addition to *1 and *2) which are used to provide run-time transfer of control through declarations whenever a block is entered. *L

4e2 The +ENDKLUDGE is used whenever an identifier appears directly after an END in PL/I. It has the effect of freezing the END in the input stream until the matching label is popped from the stack.

4f The default options of declarations are contained in the LOT syntax equations rather than in the interpreter, and can easily be modified for special applications.

4f1 For example, the BASE in PL/I is either binary or decimal with the default being decimal. his is reflected in the equation:

```
base = ("binary" / "bin") *c(d,1)  /
       ("decimal" / "dec") *c(d,2)  /
       .empty  *c(d,1)  ;
```

*D(*1
*2
*L

*C(*L)

5 *****(LOT6 ~~sect5~~)***** INTERPRETIVE MACHINE EXTENSIONS

The interpreter has been designed to provide single pass compilation, Program Reference Table organization, a method to choose among sets of run time interpreters, and options for getting symbolic "assembly language" output.

5a The change to single pass compilation eliminates many problems of allocation of intermediate buffers, provides faster overall run time, and reduces the number of times symbols are scanned. not so true

5a1 Operating the LOT system in a time-sharing enviorenment (as is being simulated on the 360/50 implementation) greatly benefits from the reduced input/output of single pass compilation.

5a1a With LOT, the user is characterized by his re-entrant operating code, input and output record area, run-time stack, and about a dozen flags and pointers which are the analogues of hardware registers.

5a1b It is relatively easy to preserve the state of a user in this type of configuration.

5a1c When a user is dismissed in the time sharing model, the flags and pointers which must be saved are pushed into his own stack. The system needs only to keep the location of the top item to be restored and the reason for dismissal (I/O, time-out, etc.).

5a2 Almost all syntax errors detected by the system are on a local level, allowing the possibility of online alarm and correction. When the end of the source program is reached, all that remains is a check for undefined symbols. Execution can begin immediately.

5a3 During compilation compound symbols on the input stream are collected and identified by the compiler at least once. It is redundant to output these identifiers again for an assembler. Many quantities (such as opcodes in the generators) are referred to internally only by table index, and unless the option for producing an assembly PREST deck is set, the actual character mnemonics are never used and not of interest to the compiler.

5b The program reference table (PRT) organization allows flexible handling on block structure, large virtual memory with a short instruction address field, and the ability to "insert" declarations into the run time code without recompilation.

5c The interpreter has the ability to switch opcode interpretation, giving the effect of having separate interpretive machines on call:

5c1 The interpreter program is similiar in organization to a conventional computer. The instruction fetch cycle retrieves an operation code and then branches to the appropriate rotine by means of a FORTRAN computed-goto (identical to an ALGOL-60 switch with numeric labels).

5c2 To get to another interpretive machine (such as when compilation has ended and the compiled code is now to be executed) a special opcode is executed. This branches to a different instruction fetch routine with its own switch. Since the "meaning" of a numeric opcode is defined by the routine's referenced in the switch, this has the effect of changing

(rep 1)

[redacted] sect0 [redacted]

0 Preface

0a date

0b The bulk of the programming work and report preparation was done on a CDC 3100 computer located in the Systems Engineering Laboratory at Stanford Research Institute.

0b1 The peripheral gear includes a card reader, 150 line/minute printer, paper tape reader and punch, and a display system with associated Invac keyboard.

0b2 The software available at the beginning of this project consisted of a basic assembler, a completely non-standard FORTRAN system (containing among other things a mysterious new type of logic in which the operators .AND. and .OR. have the same precedence), imbedded in an operating system that contained the compiler as unblocked relocatable card images on the master tape (loaded anew for each subroutine compilation) and devoured 1/3 of available core during run time to provide dubious interrupt handling.

0b3 The system did provide the comfort of an even chance that a bug was not the programmer's fault, but the fault of the hardware or system software.

~~0c It would be extremely difficult for me to acknowledge all the aid and encouragement that has been given to me during this project's development. Special thanks must be given to Jeff Rulifson for innumerable ideas, coding, criticism, and company on the frequent all night sessions. The IBM 360/50 version of LOT was done with Don Andrews under Professor Miller's guidance. Bert Raphael was the co-author of the original proposal which resulted in SRI's Institute Sponsered Research Grant, and Bill McKeeman offered advice, syntax equations, and reasons why STAPL was better throughout the project.~~

rewrite

0c This paper is intended to report the changes we feel are improvements over past Meta Systems. It is not intended to explain "Meta" philosophy. A reader unfamiliar with previous "Meta" articles is advised to read Shorre 1 and Metcalfe 3 before preceding with further.

1 (SECT1) INTRODUCTION

***** sectl *****

1 Introduction

la The two main roots of this project are the articles "META II - a syntax-oriented compiler writing language" (reference (Schorrel)) and "A parameterized compiler based on mechanical linguistics" (reference (Metcalfel)).

lal The Shorre article was published in the Proceedings of the 1964 ACM National Convention. It described an elegant notation for writing equations for a syntax-directed compiler, and gave an instruction set of an interpretive "machine" capable of processing the equations and operating as a compiler.

lala The interpreter's score of opcodes are easily simulated on a computer, and the programming of this simulation represents the major effort of putting the META II system on a new machine.

lalb The original work done by Val Shorre was on an IBM 1401. Using the article as a guide, versions of the interpreter were programmed for a series of machines. Machines for which debugged versions are known to exist include the CDC 160a, IBM 7094, PDP 1, CDC 3100, and IBM 360/50. B5500, the authors of this paper

lb The relative ease of implementing the simple META II system, and the ease with which it can be used to develop compilers for a wide range of languages led to an attempt to clarify the scope of the system, add conventional compiler features (diagnostics, system controls, etc.) and experiment with extensions to the notation and interpretive machine.

lc A great deal of information has now been gathered on useful extensions to the syntax of META II. The system is in production use as a compiler for teaching machine programs on the PDP 1. The basic LOT system on the CDC 3100 promices to provide a useful experimental facility for testing new features.

ld Further implementations are planned for the Burroughs machines at Stanford and at the University of Washington ~~this summer~~ to provide students with an experimental system in the fall. The CDC 3100 implementation is being used in an attempt to compile full PL/I, and interpretively run a large subset of the produced code.

le The main concerns now are the gathering of statistics on the workings of the interpreters in order to concentrate effort on improving performance, and designing syntactical constructs which will result in ease of description and use. Other efforts include making the interpreter act as a debugging tool with display-console interaction at run time, and extending the interpreter to act as a time sharing executive which can drive multiple consoles.

"

SECTION 2

The META II system

2a The META II system as presented by Shorre (Ref (Shorrel)) provides the basic model for LOT. META II is a compiler writing language consisting of syntax equations resembling Backus Naur form with imbedded instructions to output assembly language code. The 1964 ACM National Conferenece article is reproduced in the Appendix. The following is a summary of some of the more important features of META II.

2a1 Each syntax equation in META is translated into a recursive subroutine which tests an input stream for a particular phrase structure, and deletes it if found.

2a1a Backup is avoided by ordering and factoring the syntax equations.

2a1b The generators imbedded in the syntax equations cause assembly language code to be generated. The code consists either of literal strings from the generators (such as assembly operation codes) or of copies of items deleted from the input stream (such as identifiers).

2a2 The equations have much the same form as the meta language of the ALGOL-60 report.

2a2a The changes from the ALGOL-60 report are mostly for ease of keypunching.

2a2a1 Symbols are represented as strings of characters surounded by quotes (keypunch apostrophes are actually used).

2a2a2 Metalinguistic variables have the form of ALGOL identifiers.

2a2a3 Concatenation is indicated by writing items consecutively.

2a2a4 Alternation is indicated by a slash instead of a vertical bar.

2a2a5 A semicolon is represented by a period following a comma.

2a2b An example of a syntax equation is:

```
logicvalue = .true / .false.,
```

2a2c Optional items are indicated by alternation with the word .empty. For example:

```
subsec = '*' primary / .empty .,
second = primary subsec .,
```

2a2d The above two equations may be factored into:

```
second = primary ( '*' primary / .empty ) ,.
```

2a2e There are three basic symbols which are recognized by META II; .id, .string, .number.

2a2e1 .Id indicates an ALGOL-60 type identifier.

Make notes on + C instead of o C

2a2e2 .String indicates a sequence of characters enclosed within quotes.

2a2e3 .Number indicates a sequence of digits with an allowed imbedded decimal point.

2a2f A sequence can be recognized either by recursion or by the special sequence operator $. An arbitrary number of a's might be defined as:

```
seqa  =  $'a' .,                     or as:
seqa  =  ( 'a' seqa / .empty ) .,
```

2a3 The syntax equations for simple expressions containing parentheses and the operators + and * can be represented in META II by the equations:

```
primary = .id / '(' expression ')' .,
term = primary $( '*' primary ) .,
expression   = term $( '+' term ) .,
```

2a4 The generators to output assembly code are imbedded in the syntax. For the previous example the equations would have the form:

```
primary = .id *c(ld,*) / '(' expression ')' ,.
term = primary $( '*' *c(mlt) )  ,.
expression = term $( '+' .c(add) )  ,.
```

2a5 The style of writing in the these equations is that employed by the LOT system which differs slightly from the original META II article (see appendix for original notation). The intent of the *c form is that the quantities inside are to be coded into the assembly language output. The construct *c(ld,*) means output the operation "ld" followed by the last item deleted from the input stream, which in the equation for primary would be the identifier deleted by the .id instruction.

2a6 META also has the ability to associate internal labels with the assembly code in order to transfer control in compound statements and blocks. In any given syntax equation, the appearance of *1 or *2 is sufficient to generate a label which is local to that equation. The construct *d(*1) defines a generated label and associates the current assembly language location counter with the label. The construct *c(b,*1) would output a branch instruction to the location associated with the current *1 label.

2a6a The use of generated labels may be illustrated by a typical if-statement construct.

```
ifstatement = 'if' exp 'then' *c(bfp,*1) statement
'else' *c(b,*2) *d(*1) statement *d(*2)  ,.
```

2a6b The operation codes bfp and b stand for branch false and pop, and branch, which are primitive operations on the target machine.

2b The syntax equations are to be interpreted as defining a procedure when attempting to compile source code. If the first element of an alternative in a syntax equation is found on the input stream, then all other elements within the alternative must be found or a syntax error has occured. For example if the word 'if' has been found (using the syntax equation of "ifstatement" above) then it must be followed in order by an exp, the word 'then', a statement, 'else', and another statement. If after the word 'then' has been deleted from the input stream a statement fails to materialize, the 'then' cannot be replaced. The compiler operates without any backup.

2c META II outputs assembly code for interpreters which are most easily implemented as stack machines. With the addition of some non-meta kludgery, it is possible to compile efficient actual machine code, as has been done with an IBM 7094 (Ref (Schneiderl)).

2d The syntax equations for META itself compile into interpretive instructions for an extremely simple machine.

2d1 The following list of META machine instructions also form the basis for the LOT interpreter machine.

2d1a (TST) After deleting initial blanks from the input stream, compare it to the string given as an arguement. If the comparison is met, delete the matched portion from the input and set a flag true. If not met, set flag false.

2d1b (ID) After deleting initial blanks in the input stream, test to see if it begins with an identifier. If so, move the identifier to the star register (which will hold the identifier for output until the next .id, .num or .string operation succeeds) and set the flag true. If not, set flag false.

2d1c (SR) After deleting initial blanks on the input stream, test to see if it begins with a string quote. If it does, move the string to the star register, and set the flag true. If not, set the flag false.

2d1d (CLL) Call the subroutine given as argument. Push the top two items of the stack down (the two generated labels), and push the exit address onto the top of the stack. Clear the top two items of the stack to indicate that generated labels may be created.

2d1e (R) Return from a subroutine by popping the stack to expose the generated labels that were pushed down by CLL, and setting the instruction counter to the return address in the stack.

2d1f (set) Set the mflag on.

2d1g (B AAA) Branch to location AAA.

2d1h (BT AAA) Branch (true) to AAA if mflag is on.

2d1i (BF AAA) Branch (false) to AAA if mflag is off.

2d1j (BE) Branch Error. Halt if mflag is off.

2e One of the most significant articles referenced by Schorre is (Metcalfe1) which contains speculations on implementations of the META type of compiler system. The predictions made in this article as to size of programs, rough estimates of run time, and the avenues of exploration which might prove useful in developing the system, have been astonishingly correct.

3

*****(sect3)*****

Goals of the LOT system.

3a LOT is an attempt to provide an experimental system designed to study the utility of the META II approach to compiling, and to examine the question of what extensions need to be made to provide a convenient, reliable total system.

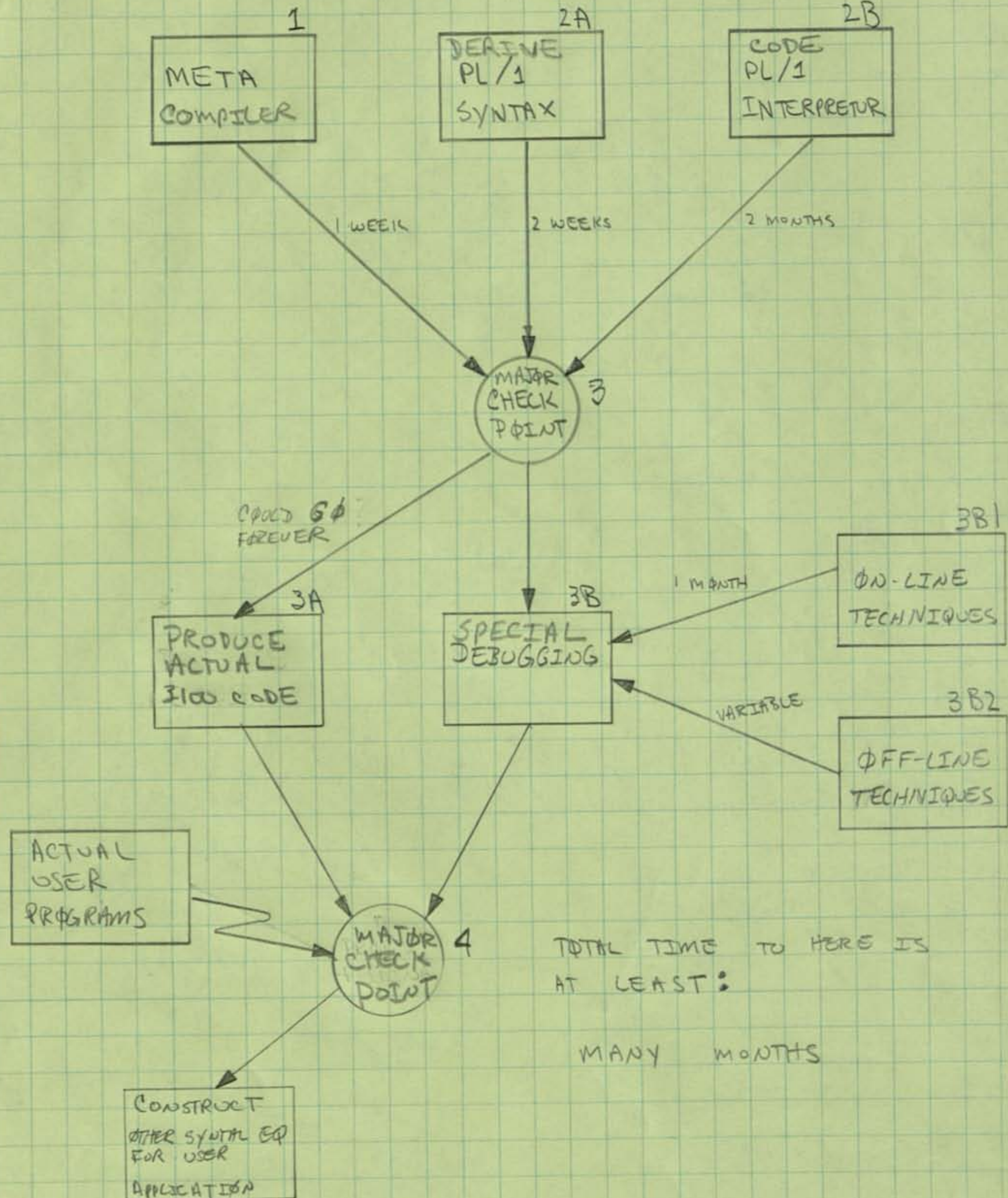
3b The interpretive organization of the LOT system makes possible a great deal of control and interaction with the programmer for studying the effects of display and keyboard debugging techniques in which the programmer need only be concerned with the high-level language in which his source code is written.

3c The LOT system is also an easily used tool for investigating new syntax equations and languages. One of the principal concerns of the oriiginal design was the desire to have a subset of PL/I available for experimentation.

3d An area of concern which has received relatively little publication of general principles is that of translating a polish string into machine code for a complete language, including procedure and function calls, and declarations. One approach to this problem is the definition of a language using LOT which would then be used to program the translation from the interpretive code of LOT to different machine codes.

3d1 This aspect of the LOT system has not yet been explored but will probably be attempted this summer using the CDC 3100 and SDS 940 as target machines.

PERT NETWORK FOR PL/1 PROJECT



0 NOTES ON META AND PL/1

0A JFR 5 APRIL 66

1 THE BASIC PROGRAMMING WORK IS THE META MACHINE AND META COMPILER. THIS IS ALMOST CHECKED OUT NOW. A FACTOR IN ESIGN AND CHECKOUT HAS BEEN ONE OF FLEXIBIBILTY. THAT IS, WE KEEP ADDING AND CHANGING THE META SYSTEM IN ORDER TO AKE IT BIG ENOUGH TO HANDLE A GOOD SIZED SUBSET OF PL/1.

2 DESIGN AND CODING OF PL/1.

2A THE DESIGN OF THE SYNTAX EQUATIONS. AT THIS TIME WE ARE CONSIDERING ADDING EXTRA CHARACTER PUSHING AND STRING FUNCTIONS TO OUR SUBSET OF PL/1.

2B AS THE SYNTAX EQUATIONS ARE WORKED OUT THERE MUST BE AN INTERPRETER TO EXECUTE THE COMPILED PL/1 CODE. THIS WILL BE A MAJOR TASK OF THE PROJECT.

2B1 THE WORKING MEMORY FOR A USER'S EXECUTING PL/1 PROGRAM WILL BE STRUCTURED FROM 8 BIT BYTES.

2B2 A MAJOR FEATURE OF USER MEMORY WILL BE ITS VIRTUAL SIZE, 512K' BYTES, IMPLEMENTED BY AN AUTOMATIC DISK PAGING SCHEME.

2B3 WRITING OF THIS CODE MAY LEAD TO A GENERAL SET OF MACROS IN COPE FOR IMPLEMENTING OTHER PROGRAMS REQUIRING OVERLAY OR PAGING TECHNIQUES FOR DATA OR CODE.

3 WHEN THIS IS DONE WE WILL BE VERY HAPPY. AT THIS POINT FURTHER DEVELOPMENTS CAN GO IN TWO WAYS.

3A PRODUCTION OF MACHINE CODE. ONE MAY BE ABLE TO PRODUCE CODE FOR A CDC 3100 OR SDS 930 QUITE EASILY FROM THE INTERPRETIVE CODE. THE PRODUCTION OF EFFICIENT CODE USING HEURISTICS MAY REQUIRE THE DEVELOPMENT OF MORE THEORY IN THIS FIELD, AND BE TOO TIME COMSUMING.

3B THE PROBLEMS OF DEBUGGING IN A HIGHER LEVEL LANGUAGE.

3B1 ONE APPROACH IS TO IMPLEMENT MANY WAYS OF USING THE DISPLAY FOR METHODS OF ONLINE DEBUGGING. THIS WILL REQUIRE NEW COMPILING CONCEPTS WHICH ALLOW RETENTION OR RECONSTRUCTION OF SYMBOLIC INFORMATION NORMALLY LOST IN THE COMPILATION PROCESS.

3B2 ANOTHER APPROACH IS THE STUDY OF VARIOUS THINGS THAT CAN BE DONE FOR THE USER TO HELP IN OFFLINE DEBUGGING. MANY NEW POSSIBILITIES FOR HELPING THE PROGRAMMER ARE OPENED SIMPLY SINCE WE ARE RUNNING THE PROGRAMS INTERPRETIVELY.

4 ANSWERS TO SOME QUESTIONS.

4A WILL WE INTERFACE WITH ANYBODY ELSE...

WE WILL INTERFACE WITH THE ONLINE SYSTEM IN THAT THERE WILL BE NO INCREMENTAL COMPILTATION. IF A USER, WHILE HE IS DEBUGGING, WANTS TO CHANGE, ADD, OR DELETE ANY CODE HE WOULD GO BACK TO THE ONLINE SYSTEM, CHANGE IT, RECOMPILE, BREAKPOINT, AND RUN. IT IS A GOAL OF THE PROJECT THAT NO ONE ELSE WILL WANT TO INTERFACE WITH THE PL/1 INTERPRETER BUT IF THEY WANT TO THEY WILL BE FREE TO TRY.

4B WHO WOULD USE THE SYSTEM...

ANY ONE WHO NEEDS BIG MEMORY, OR WANTS SIMPLE DISPLAYS, OR WANTS A POWERFUL LANGUAGE THAT COMPILES FAST, OR WOULD LIKE 8 BIT CHARACTERS. THE FAST COMPILATION FEATURE IS VERY IMPORTANT FOR UNDER THE CDC FORTRAN COMPILER AND THE SCOPE SYSTEM, SYSTEM OVERHEAD, AND COMPILATION TIME, AND I/O TIME ARE FAR GREATER THE USER RUN TIME. WE FEEL THAT ONLINE DEBUGGING, VERY FAST COMPILATION, AND THE EASE OF USING A VERY LARGE MEMORY WILL MORE THAT COMPENSATE FOR THE SLOWNESS OF THE INTREPRETUR.

interpreters.

5d There is no assembly pass in LOT, but an option exists for getting a "PREST" deck which resembles assembly language.

5d1 Each LOT interpretive insturction which either puts code into memory or associates a symbol with a location (the *c and *d routines) checks the PREST option before returning for the next instruction fetch. If the PREST option is on, the routine outputs a symbolic representation of the action taken.

5d1a The assembly language format on the PREST deck is sutiable for input to an additive assembler. In LOT the assembler is a trivial set of syntax equations:

```
.meta assembler
assembler = $( "$" ( .id / .number ) *d(*) )  /
        "*" ( .id *c(a,*) / .number *c(*n) )  /
        .string  *c(*q)     )
  "/" +xecute +stop ,.
```

5d1b These equations state that the PREST deck consists of any number of the following constructs: a dollar sign which may be followed by identifiers (labels) and numbers (generated labels); a "*" followed by identifiers (opcodes) and numbers (numeric opcodes); just identifiers and numbers (addresses); literal strings.

5d1c For example the PREST deck translation of the SIMPL statement

```
        X = Y + Z;
would be
        *LDA X*LDA Y*LDA Z*ADD*STO
```

where LDA, LDC, ADD, STO are opcodes and X, Y, Z are addresses.

5d2 These decks aid in directly loading a high level program or compiler without going through the compiling process again. In this respect, they are acting as binary decks.

5d3 The speed of both the compilation process and loading of PREST decks seems to be quite proportional to the number of symbols involved. Loading a PREST deck instead of recompiling does not offer much speed gain unless the compiler itself would have had to be compiled again also.

5d4 The PREST decks are useful mainly in the initial bootstrap phase of new systems, or as intermediate output for use in the generation of actual machine code via a second pass or by macro expansion.

5d5 The following is the PREST deck output of the LOT compiler compiling itself. It is part of the deck that is loaded directly by the assembler for normal use. The full syntax equations and PREST deck defining the compiler will be found in section 12c.

5d5a The LOT equation for a program is shown both in normal input form, and the output that is the result of having the PREST deck option on.

```
program = ".meta" $directive  .id  *c(*)
( ".opcodes" $(  .id *d(*)  )  ",."  /  .empty  )
```

```
( ".reservedwords" $( .id *d(*r,*) ) ",." / .empty )
$statement ".end" ←xecute ←stop ,.
```

```
$program*ts '.meta'*f 1$2*c directive*t 2*ft*e*i*e*s*ts '.opcodes'*f
3$4*i*f 5*d$5$6*t 4*ft*e*ts ',.'*e$3*t 7*ft*f 8$8$7*e*ts
'.reservedwords'*f 9$10*i*f 11*sr*d$11$12*t 10*ft*e*ts ',.'*e$9*t
13*ft*f 14$14$13*e$15*c statement*t 15*ft*e*ts '.end'*e*xc*h$1$16*r
```

5d5b The LOT equation for a statement is shown in both forms.

```
statement = .id *d(*) "=" .e("***no = ", ",. ", statement)
expression .e("***expect exp ", ",. ", statement) *c(r)
",." .e("***no ,. ", ",. ", statement) $directive ,.
```

```
$statement*i*f 17*d*ts '='*be '***no = ' ',. ' statement*c
expression*be '***expect exp ' ',. ' statement*a r*ts ',.'*be '***no
,. ' ',. ' statement$18*c directive*t 18*ft*e$17$19*r
```

'?

```
                                  SET/1499 DECLARE/172 LABEL/915 INITIAL/461 CALLZ1421 END/648 ENTRY/967 DO/998 I
                                 F/1253 BUMP/1378 RETURN/1467 READ/1521 WRITE/1639 DIRECTIVES/96 ARGUMENT.LIST/12
                                 1 PROGRAM.ELEMENT/135 ARB.ARG/2267 GROUP/672 DECLARATION/227 DIMENSION/309 ATTRI
                                 BUTE/367 EXP/1757 TYPE/383 ITEM/541 CONSTANT/2300 ITERATION/577 LABELED.GROUP/68
                                 8 UNLABELED.GROUP/806 MORE.LABELS/770 ASSIGN/920 SIMPLE/1301 ITERATIVE.GROUP/831
                                  LABELED.I.GROUP/847 SUBSCRIPT/2250 ASSIGN2/936 VARIABLE/2237 DORANGE/1134 GOTO/
                                 1449 READ.DATA.LIST/1548 WRITE.DATA.LIST/1666 UNION/1764 INTERSECTION/1794 NEGAT
                                 ION/1824 RELATION/1847 SUM/1955 PRODUCT/1996 FACTOR/2037 BDIFFERENCE/2094 BSUM/2
                                 119 BPRODUCT/2144 COMPLEMENT/2169 UNIT/2192 BUILT.IN.FUNCTION/2302
.SIMPLE -LIST -PUNCH
XID.. PROCEDURE ,.
                                 oMKSoLSA XIDoDo61oB 1$2oLSA IMSBUoMKSoLDC 1oSFRoBUPoLSA DELBLKoCLLoLSA MFLAGoRST
  BUMP IMSBU(1),. CALL DELBLK,. MFLAG=FALSE ,.
  IF IAT(W(1)) = LTR THEN
    DO ,. BUMP IW,.
                                 oSTOoLSA IAToMKSoLSA WoMKSoLDC 1oSFRoSFRoLSA LTRoEQoBFP 3$4oLSA IWoBUPoLSA IWoLS
                                 A IWoSSDoLNAoLNAoSLAoDOIoLSA IAToMKSoLSA WoMKSoLSA IWoSFRoSFRoLSA IPNToLEoPBF 5o
     DO IW = IW WHILE IAT(W(IW)) LE IPNT AND IAT(W(IW)) ,.
                                 LSA IAToMKSoLSA WoMKSoLSA IWoSFRoSFR$5oDOT 6oB 7$6oLSA IWoLSA MAXLENoGToBFP 8oLS
         IF IW GT MAXLEN THEN CALL ERROR ,.
    END ,. END ,.
                                 A ERRORoCLL$8oJMP$7$3oLSA IWoLSA IWoSSDoLDC 1oUMoLNAoFLPoSLAoDOIoLSA IAToMKSoLSA
  DO IW = IW BY -1 WHILE IAT(W(IW)) = IPNT ,. ,. END ,.
                                  WoMKSoLSA IWoSFRoSFRoLSA IPNToEQoDOT 9oB 10$9oJMP$10oLSA CLEARoCLLoLSA PACKoCLL
  CALL CLEAR ,. CALL PACK ,.
  IF IRWFLAG THEN
                                 oLSA IRWFLAGoBFP 11$12oLSA IoLSA IATOMoSTOoLSA ASRoMKSoLSA ISATOMoLDC 1oSUBoSFRo
    DO ,. I=IATOM,. CALL ASR(ISATOM-1),.
                                 CLLoLSA IoPBF 13oLSA LATOMoMKSoLSA IoSFRoLSA IRSWDoAND$13oBFP 14$15oLSA IWoLDC 1
      IF I AND LATOM(I) .o IRSWD THEN DO ,. IW=1,. RETURN,. END ,.
    END ,.
  ELSE MFLAG = TRUE ,. RETURN ,.
                                 oSTOoRET$14oB 16$11oLSA MFLAGoSEToSTO$16oREToB 17$1oMKSoLSA LB1oDo6oPTMoB 18$17o
LB1.. CALL SUB ,.
LB2..LB3.. X=Y ,.
                                 LSA SUBoCLLoB 19$18oMKSoLSA LB2oLSA LB3oDo6oPTMoB 20$19oLSA XoLSA YoSTOoB 21$20o
                                 MKSoLSA LB4oLSA LB5oLSA LB6oDo6oPTMoB 22$21oLSA XoLSA YoLSA ZoLSA AoLSA BoEQoSTN
LB4..LB5..LB6..X,Y,Z=A=B ,.
                                 oSTNoSTOoLSA AoLSA BoPBF 23oLSA C$23oPBT 24oLSA DoPBF 25oLSA E$25oPBT 26oLSA FoP
A=B AND C OR D AND E OR F AND G OR E AND F OR G AND H OR I AND J OR K AND L ,.
                                 BF 27oLSA G$27oPBT 28oLSA EoPBF 29oLSA F$29oPBT 30oLSA GoPBF 31oLSA H$31oPBT 32o
                                 LSA IoPBF 33oLSA J$33oPBT 34oLSA KoPBF 35oLSA L$35$34$32$30$28$26$24oSTOoLSA AoB
IF A THEN LB7..DO ,. CALL SUB1,. RETURN,. END ,. ELSE LB8.. DO ,.
                                 FP 36oB 37$22oMKSoLSA LB7oDo6oPTMoB 38$37$39oLSA SUB1oCLLoREToB 40$36oB 41$38oMK
                                 SoLSA LB8oDo6oPTMoB 42$41$43oB 44$42oMKSoLSA LB9oDo6oPTMoB 45$44oLSA AoPBF 46oLS
                                 A B$46oPBT 47oLSA CoPBF 48oLSA D$48$47oBFP 49oB 50$45oMKSoLSA LB10oDo6oPTMoB 51$
LB9..IF A AND B OR C AND D THEN LB10..RETURN,. LB11.. CALL SUB1 ,.     END ,.
                                 50oRET$49oB 52$51oMKSoLSA LB11oDo6oPTMoB 53$52oLSA SUB1oCLL$40oB 54$53oMKSoMKSoL
                                 SA AoMKSoLDC 0oLDC 10oLSA B
DECLARE (A(0..10),B,C(10,10) INITIAL (0,o,0 (15o1) (0,o,0 (17) o) ) ) INTEGER ,.
```

*****{sect7}*****

7 The special features of LOT peculiar to the CDC 3100 implementation are the use of a non-standard specialized FORTRAN, gathering of timing statistics, and console control of snapshots and traces.

7a The format of the FORTRAN program on the 3100 system as given in section 14a is severe departure from ASA FORTRAN.

7a1 Extensive use was made of the feature allowing successive statements to be separated by a dollar sign if the statements did not require a statement number. This was done to try and minimize the amount of time needed to list the programs on the 150 line/minute line printer.

7a2 The implementation consists of only six routines, each with many entries. This was done since the overhead in compiling any FORTRAN program on the 3100 is 22 seconds per subroutine, regardless of the length of the subroutine.

7b Wide use was made of character declarations. This enables the programmer to declare an array to be a character array, and the interpretation of A(J) is the Jth character of the array. The characters are packed four to the (24 bit) word, and the feature is implemented using a hardware instruction of the 3100 for fetching characters.

7c The gathering of statistics has been of utmost importance in improving the program. During most compilations, five interpretive instructions have accounted for over eighty percent of all operations executed. Statistics are now being gathered on the time spent in each of these instructions to indicate where the basic program might profitably be altered. More importantly, certain ratios of instructions provide a rough measure of the need to restructure the syntax equations.

7c1 The syntax equations for a language may contain elements whose ordering has no effect on the definition of the language. For example, the equation:

```
statement = dost / ifst / gotost / assignst ;
```

defines the syntax of statement, but the relative order of the elements is immaterial. When these equations are being used to drive the LOT interpreter, the attempted parsing of the input stream for a "statement" will be carried out from left to right, searching first for a "dost", then an "ifst" etc. If the programs being compiled contain a high proportion of assignment statements (assignst), then the compiler will run very slowly. This is indicated in the basic instruction counts as the ratio between the number of conditional branches that were satisfied to the number of conditional branches that failed. Even more pertinent statistics may be taken on the number of times a syntactic element (such as dost) was sought but not found.

7d A variety of traces and snapshots are available on the 3100 system under the combined control of the compiler, program being compiled, and console jump switches. These include counts and timings of instructions, core dumps (of simulated interpreter core), and detailed instruction cycle traces in which the internal registers are printed out on each instruction fetch.

***** sect8 *****

8 The IBM 360/50 implementation of LOT contains provisions for time sharing of multiple users, one pass compilation with fixups, and extended syntax equations for compatablility of old and new keypunch card codes.

8a The main goal of the 360 implementation was a simulation of time sharing the basic LOT interpretive machine. Much of the machinery in the basic interpreter was concerned with the goal of having the entire operations of compiling, executing, loading, and input/output be under control on the interpreter while servicing multiple users.

8b Multiple users were assumed to be at consoles connected through a multiplexer to the interpretive machine. The card reader was used to provide the actual input with the first column being a letter indicating the simulated console providing the message. The line printer was used for multiplexer output with the first printing column indicating to which console the message would have been routed.

8c The compilation was one pass direct to core with a PREST deck option, but the internal structure did not include a PRT table. The instructions for defining and referencing symbols used a common symbol table with a linked list fixup scheme for forward references.

8d The language used was Basic Programming System (BPS) FORTRAN exclusive of a few library routines for shifting and masking characters.

8e Compatability with the 3100 system was fairly successful. The different character sets of the 029 and 026 keypunch machines was handled in the syntax equations by constructs such as:

```
semicolon = ",." / ";" ;
```

8f The completely unbuffered I/O of BPS FORTRAN, together with gross implementation of subroutine calls in BPS FORTRAN contributed to make the 360 implementation actually run slower than the 3100 implementation despite a basic machine cycle which is twice as fast as the 3100. The second level of FORTRAN for the model 50 is about to be released, and the expected speed gain should be sufficient to make the 360 version faster.

I DON'T BELIEVE IT

SECTION 9

***** sect9 *****

9 User/System communication via keyboard and display allows for some primitive editting and debugging in the present version. Disc files can be used for input code, and a sophisticated text editor already exists for preparing the files. The major problems arise from partial execution of programs under control of a user, and allowing run time changes in the object code.

9a Editing of the source program is most easily accomplished when the editing process is to be followed by a complete recomplilation.

9a1 PL/I functions do not have to have arguments, so that the assignment statement:

```
s = q ;
```

may be calling on the function q. If q had not been declared to be a function prior to this statement's appearance, it would be declared by default to be a simple real variable. A problem exists in letting a user insert a forgotten declaration ahead of a statement which it would affect, without attempting a recompilation.

9a2 The LOT system is trying two different solutions to this problem. The first is to try and make compilation fast enough that recompilation will not be a significant delay in the editing process. The second solution is to analyze the penalties involved in ignoring declarations and compiling identical code for all syntactically identical elements. It is of course possible to include the typing of variables and functions in the syntax equations (see (Schneider1)), but if this is not done, and ALL declarations are done dynamically, then the problem of whether to fetch q (as a variable), or call q (as a function) can be handled by the interpreter working through the PRT table.

9a3 This has the further advantage of allowing "afterthought" declarations to be inserted at run time without recompilation of code which refers to the affected quantities.

9b A count is kept by the input routine of the number of characters it has handled since the beginning of compilation. This number is compiled into the object code as the address field of a "cursor" instruction. The cursor instruction may be inserted at any desired place in the syntax equations, and is used to control run-time displays.

9b1 When the program is in execution, there are a number of modes of operation. One mode allows the display of the original source code, and stepping of the program on the display. The interpreter, for each step, runs the code until a cursor instruction is encountered (usually inserted in the syntax at the end of compilation of a statement).

9b2 The address of the character in the original source code (which is saved on a disc file during compilation) is computed. If it is in a section of text already on the screen, the cursor is moved to that point. If the referenced character is not on the screen, a check of the disc input buffer is made, and if the character is not in core, a fetch is made from the disc file. In any case, the cursor moves to positions referenced in the original source code under direction of cursor instructions in the the compiled code.

9b3 The extension to this is to allow the positioning of the cursor by the user, and a request to have the indicated statement executed. This will be attempted this summer.

9c A user sitting at the console can also request to have registers opened in the style of a DDT system. The variable names are available to the interpreter at run time since they are kept in string storage and referenced throgh the PRT table. The format of the display is presently fixed, and the display is made according to the declared type of the variable.

9c A variable displayed may be changed, but problems exist as to the indication of which level of recursion the variable exists. The one shown on the screen is the variable as presently visible to the interpreter in the present system.

?

cuts beyond plan for this

***** sect10 *****

10 Experiance with the LOT system is now approaching the point at which a presentation of the entire scheme in an "implementation manual" form could be given with realistic estimates on machine needs, program development time, and scope of application of the resulting system. The gathering of statistics in the present version indicates the critical areas of programming needed to make an efficient implementation, and the possiblility of building a library of compiler equations does not seem too remote.

10a The basic elements of the syntax equations with inclusion of control card options, fast scanning organization, and error diagnostics is very well developed.

10a1 In an "implementation manual" the syntax equations and the first bootstrap compilation would be a basic part of the documentation.

10a2 The equations themselves lend themselves easily to provide compatability across different machines and different character sets.

10b The following typical statistics were gathered on the CDC 3100 implementation while the LOT compiler was compiling the SIMPL syntax equations. The SIMPL equations are given in section 12d.

10b1

OPCODE	INSTRUCTION	USEAGE	PCNT	TIMING	PCNT
f	branch false	3033	25	2067	10
t	branch true	2491	21	1593	7
ts	test string	1503	12	4638	~~14~~ 22
c	call	876	7	1091	5
r	return	876	7	872	4
be	syntax error	378	3	242	0
e	error	351	2	203	0
a	code atom	473	4	359	1
l	literal	0	0	0	0
i	identifier	407	3	4499	21
q	string	322	2	1333	5
n	number	143	1	133	0
ft	set true	302	2	186	0
ff	set false	0	0	0	0
s	code star	122	1	1108	5
sq	code star q	61	0	566	2
sn	code star n	0	0	0	0
sr	set rw flag	21	0	13	0
d	define star	81	0	594	2
d1	define *1	140	1	119	0
d2	define *2	0	0	0	0
g1	generate *1	125	1	118	0
g2	generate *2	0	0	0	0
sp	set snap	1	0	1	0
	totals	11707		20735	

10b1 The counts made of the number of times an insturuction was used should remain constant over different implementations of the interpreter (since they are a function only of the equations and source code being compiled) but the actual timings associated with their execution should vary somewhat dependent on differences in algorithms and actual hardware charateristics.

10b2 The timings of the simplest instruction (FT set flag true) indicate that there is a 33 percent overhead in each instruction during the instruction fetch cycle. Most of this is spent doing statistic gathering, checking array bounds, and checking for trace options. Reorganization of this section will probably yeild a gain of about 20 percent, with more possible by removing the checks for those compilers earning this sort of confidence.

10b3 The three instructions branch false, branch true, and test for string, account for 58 percent of the number of instructions executed. One change contemplated on the basis of these figures is a combination of the test string, and branch false instructions into one instruction since they often follow each other in the object code.

10b4 The three instructions branch false, test for string, and identifier account for 40 percent of the total run time.

10b5 reserved words [illegible] to the time

11 *********sectII************

References:

11a (Metcalf1) Metcalfe, Howard H., "A Parameterized Compiler based on mechanical Linguistics," Annual Review in Automatic Programming, Goodman, R. E. ed., vol 4 (1963) pp125-165.

11b (Schorre1) Schorre, Val., "META II: A syntax-oriented compiler writing language," ACM Proc. 19th National Conf, August 25-27, 1964. New York, D1.3-1 - D1.3-11.

11c (Schneider1) Schneider, F.W., and G.D. Johnson. "Meta-3: A syntax directed compiler - writing compiler to generate efficient code," ACM Proc 19th National Conf., August 25-27, 1964. New York, D1.5-1 - D1.5-8

11d (English1) English W.K., D.C. Engelbart and Bonnie Huddart, "Computer-Aided Display Control," Stanford Research Institute Project 5061, July 1965

12a **** sect12a *****

OBSELETE

LOT syntax equations (without code generators or error recovery)

```
.meta -punch program     /*the rulifson-kirkley meta language*/


/* meta itself has no reserved words */


program = ".meta"  $directive .id
    ( ".opcodes" $(.id )  ",." /  .empty  )
    ( ".reservedwords" $(  .id  )  ",."  /  .empty  )
    $statement  ".end"  ←xecute  ←stop  ,.

statement = .id "=" expression   ",." ,.

expression = subexpression $( "/" subexpression )

subexpression = ( element / generator / control / directive  )
            $( element/ generator / control / directive  ) ,.

element = .string / .id /  "$"  element / "(" expression ")"  /
    "."  ( "id" / "empty" / "string" / "number"  / "reset" )  /
    .number   ,.

generator = "*"  (
      "c" "(" codefield  $( ","  codefield ")"
   /
      "d" "("
          ( "*r"  ","   / .empty )
          ( "*1" / "*2" / "*" )
        ")"   ) ,.

codefield = .id / "*1" / "*2" / "*r" / "*q" / "*n" / "*" / .number  ,.

control = "←"  (
              "anchor"    / "unanchor" /
              "list"      / "unlist"   /
              "recmode"   / "stream"   /
              "xecute"    / "return"   /
              "save"      / "link"     /
              "endkludge" / "b"        /
              "truncate"  / "load"     /
              "punch"     / "unpunch"  /
              "pk"        / "k"        /
              "stop"       )
  ,.

directive =  "-"  (
         "list"      ←list      / "unlist"      ←unlist  /
         "recmode"   +recmode   / "stream"      +stream /
         "punch"      punch     / "unpunch"     +unpunch /
         "anchor"    ←anchor    / "unanchor"    +unanchor /
         "truncate"  ←truncate  / "xecute"      +xecute /
         "return"    ←return    / "load"        ←load /
         "stop"      ←stop      )
   ,.
```

.end

12b ***** sect12b *****

OBSELETE

LOT syntax equations (with code generators but without error recovery)

```
.meta -punch program      /*the rulifson-kirkley meta language*/

.opcodes        /*the order of this table is very important */

f    /*branch to generated label if the meta flag is false     */
t    /*branch to generated label if the meta flag is true      */
ts   /*test for a literal string                               */
c    /*call a meta statement                                   */

r    /*return from a meta statement                            */
be   /*fatal error with recovery for syntax scan only          */
e    /*fatal error without any recovery                        */
a    /*code relative atom number                               */

l    /* code literal                                           */
i    /*test for an identifier                                  */
q    /*test for a string                                       */
n    /*test for a number                                       */

ft   /*set the meta flag true                                  */
ff   /*set the meta flag false                                 */
s    /*code a reference to the atom in the s-register          */
sq   /*code a reference to star and quote in on output         */

sn   /*code the number in the s-register                       */
sr   /*set the reserved word flag on-next atom will be resv    */
d    /*define the atom in the s-register                       */
dl   /*define generated label 1                                */

d2   /*define generated label 2                                */
gl   /*generate label 1                                        */
g2   /*generate label 2                                        */
sp   /*snap according to level number                          */
sl   /*save location for pl/1 declatation link                 */

lk   /*link from save for pl/1 declaration                     */
ek   /*pop save-link stack for pl/1                            */
at   /*set anchor mode on-do not delete blanks automatically*/
af   /*set anchor mode off                                     */

lt   /*set the listing flag true                               */
lf   /*set the listing flag false                              */
pt   /*set punch flag true                                     */
pf   /*set punch flag false                                    */
rt   /*set the record mode flag true                           */

rf   /*set the record mode flag false                          */
xc   /*execute a program                                       */
xr   /*return from a program                                   */
p    /*pack the character from the window into the s-reg       */

k    /*clear the s-register to blanks                          */
db   /*delete blanks                                           */
```

```
tc  /*truncate the output buffer                                    */
b   /*branch                                                        */
ld  /*load a prest deck                                             */
h   /*stop the world i want to get off                              */
  ,.  /* end of op code table */


/* meta itself has no reserved words */


program = ".meta"  $directive  .id  *c(*)
    ( ".opcodes" $(  .id *d(*)  )  ",."  /  .empty  )
    ( ".reservedwords" $(  .id  *d(*r,*)  )  ",."  /  .empty  )
    $statement  ".end"  ←xecute  ←stop  ,.

statement = .id *d(*) "=" expression ",."  ,.

expression = subexpression $( "/" *c(t,*1) subexpression )
    *d(*1) ,.

subexpression = ( element *c(f,*1) /  generator / control /
                   directive     )
    $( element ( ".e" "("  .string
        *c(be,*q)  ","  .string *c(*q)  "," .id  *c(*) ")"
      / .empty *c(e)  )  /
      generator / control / directive    )  *d(*1)  ,.

element = .string *c(ts,*q) / .id *c(c,*) /
    "$" *d(*1) element *c(t,*1,ft) / "(" expression ")" /
    "."
      (    "id" *c(i) / "empty" *c(ft) / "string" *c(q) /
           "number" *c(n) / "reset" *c(ff)  )
    .number *c(*n)  ,.

generator = "*"  (
    "c" "("  codefield   $( ","  codefield) ")"   /
    "d" "("   ( "*r" *c(sr) ","  / .empty )
        ( "*1" *c(d1) / "*2" *c(d2) / "*" *c(d) )
      ")" )

codefield = .id *c(a,*) / "*1" *c(g1) / "*2" *c(g2) /
            "*r" *c(sr) / "*q" *c(sq) / "*n" *c(sn) /
            "*"  *c(s)  / .number *c(1,*n)  ,.

control = "←"  (
            "anchor"     *c(at) / "unanchor" *c(af) /
            "list"       *c(lt) / "unlist"   *c(lf) /
            "recmode"    *c(rt) / "stream"   *c(rf) /
            "xecute"     *c(xc) / "return"   *c(xr) /
            "save"       *c(sl) / "link"     *c(lk) /
            "endkludge"  *c(ek) / "b"        *c(db) /
            "truncate"   *c(tc) / "load"     *c(ld) /
            "punch"      *c(pt) / "unpunch"  *c(pf) /
            "pk"         *c(p)  / "k"        *c(k) /
            "stop"       *c(h)  / "snap" .number *c(sp,*n) )  ,.

directive =  "-"  (
        "list"      ←list     / "unlist"      ←unlist  /
        "recmode"   ←recmode  / "stream"      ←stream /
        "punch"     ←punch    / "unpunch"     ←unpunch /
```

```
"anchor"     ←anchor   / "unanchor"      ←unanchor /
"truncate"   ←truncate / "xecute" ←xecute /
"return"     ←return   / "load"          ←load /
"stop"       ←stop     / "snap" .number ←snap 63  ) ,.

.end
```

12C

LOT syntax eq.

1.0

```
.META -PUNCH PROGRAM      /*THE RULIFSON-KIRKLEY META LANGUAGE*/

.OPCODES        /*THE ORDER OF THIS TABEL IS VERY IMPORTANT */

F    /*BRANCH TO GENERATED LABEL IF THE META FLAG IS FALSE  */
T    /*BRANCH TO GENERATED LABEL IF THE META FLAG IS TRUE   */
TS   /*TEST FOR A LITERAL STRING                            */
C    /*CALL A META STATEMENT                                */

R    /*RETURN FROM A META STATEMENT                         */
BE   /*FATAL ERROR WITH RECOVERY FOR SYNTAX SCAN ONLY       */
E    /*FATAL ERROR WITHOUT ANY RECOVERY                     */
A    /*CODE RELATIVE ATOM NUMBER                            */

L    /* CODE LITERAL                                        */
I    /*TEST FOR AN IDENTIFIER                               */
Q    /*TEST FOR A STRING                                    */
N    /*TEST FOR A NUMBER                                    */

FT   /*SET THE META FLAG TRUE                               */
FF   /*SET THE META FLAG FALSE                              */
S    /*CODE A REFERENCE TO THE ATOM IN THE S-REGISTER       */
SQ   /*CODE A REFERENCE TO STAR AND QUOTE IN ON OUTPUT      */

SN   /*CODE THE NUMBER IN THE S-REGISTER                    */
SR   /*SET THE RESERVED WORD FLAG ON-NEXT ATOM WILL BE RESV */
D    /*DEFINE THE ATOM IN THE S-REGISTER                    */
D1   /*DEFINE GENERATED LABEL 1                             */

D2   /*DEFINE GENERATED LABEL 2                             */
G1   /*GENERATE LABEL 1                                     */
G2   /*GENERATE LABEL 2                                     */
SP   /*SNAP ACCORDING TO LEVEL NUMBER                       */
SL   /*SAVE LOCATION FOR PL/1 DECLATATION L NK              */

LK   /*LINK FROM SAVE FOR PL/1 DECLARATION                  */
EK   /*POP SAVE-LINK STACK FOR PL/1                         */
AT   /*SET ANCHOR MODE ON-DO NOT DELETE BLANKS AUTOMATICALLY*/
AF   /*SET ANCHOR MODE OFF                                  */

LT   /*SET THE LISTING FLAG TRUE                            */
LF   /*SET THE LISTING FLAG FALSE                           */
PT   /*SET PUNCH FLAG TRUE                                  */
PF   /*SET PUNCH FLAG FALSE                                 */
RT   /*SET THE RECORD MODE FLAG TRUE                        */

RF   /*SET THE RECORD MODE FLAG FALSE                       */
XC   /*EXECUTE A PROGRAM                                    */
XR   /*RETURN FROM A PROGRAM                                */
P    /*PACK THE CHARACTER FROM THE WINDOW INTO THE S-REG    */

K    /*CLEAR THE S-REGISTER TO BLANKS                       */
```

```
DB   /*DELETE BLANKS                                          */
TC   /*TRUNCATE THE OUTPUT BUFFER                             */
B    /*BRANCH                                                 */
LD   /*LOAD A PREST DECK                                      */
H    /*STOP THE WORLD I WANT TO GET OFF                       */
  ,.  /* END OF OP CODE TABLE */
```

CN /* CURSOR NUMBER */

DC /* DEFINE CURSOR */

UI /* USER INTERPRETER */

LA /* LISTALL (FULL LIST) */

```
/∘ META ITSELF HAS NO RESERVED WORDS ∘/

PROGRAM = ≠.META≠  $DIRECTIVE                  .ID  ∘C(∘)
    ( ≠.OPCODES≠ $(  .ID ∘D(∘)  )  ≠,.≠  /  .EMPTY  )
    ( ≠.RESERVEDWORDS≠ $(  .ID  ∘D(∘R,∘)  )  ≠,.≠  /  .EMPTY  )
    $STATEMENT  ≠.END≠  +XECUTE  +STOP  ,.

STATEMENT = .ID ∘D(∘) ≠=≠ .E(≠∘∘∘NO = ≠, ≠,. ≠, STATEMENT)
    EXPRESSION .E(≠∘∘∘EXPECT EXP ≠, ≠,. ≠, STATEMENT) ∘C(R)
    ≠,.≠ .E(≠∘∘∘NO ,. ≠, ≠,. ≠, STATEMENT) $DIRECTIVE  ,.

EXPRESSION = SUBEXPRESSION $( ≠/≠ ∘C(T,∘1) SUBEXPRESSION )
    ∘D(∘1) ,.

SUBEXPRESSION = ( ELEMENT ∘C(F,∘1) /   GENERATOR / CONTROL /
                      DIRECTIVE      )
    $( ELEMENT ( ≠.E≠ ≠(≠ .E(≠∘∘∘NO ( ≠, ≠,. ≠, STATEMENT)
                  .STRING   .E(≠∘∘∘NO STRING ≠, ≠,. ≠, STATEMENT)
         ∘C(BE,∘Q)   ≠,≠  .E(≠∘∘∘NO , ≠, ≠,. ≠, STATEMENT)
                  .STRING   .E(≠∘∘∘NO STRING ≠, ≠,. ≠, STATEMENT)
         ∘C(∘Q)        ≠,≠ .E(≠∘∘∘NO , ≠, ≠,. ≠, STATEMENT)
                       .ID .E(≠∘∘∘NO ID ≠, ≠,. ≠, STATEMENT)
         ∘C(∘)         ≠)≠ .E(≠∘∘∘NO ) ≠, ≠,. ≠, STATEMENT)
       / .EMPTY ∘C(E)  )  /
      GENERATOR / CONTROL / DIRECTIVE     )  ∘D(∘1)  ,.

ELEMENT = .STRING ∘C(TS,∘Q) / .ID ∘C(C,∘) /
    ≠$≠ ∘D(∘1) ELEMENT
      .E(≠∘∘∘EXPECT ELEMENT ≠, ≠,. ≠, STATEMENT) ∘C(T,∘1,FT) /
    ≠(≠ EXPRESSION .E(≠∘∘∘NO EXP ≠, ≠,. ≠, STATEMENT)
         ≠)≠  .E(≠∘∘∘UNBAL ( ≠, ≠,. ≠, STATEMENT)  /
    ≠.≠  (
            ≠ID≠ ∘C(I) / ≠EMPTY≠ ∘C(FT) / ≠STRING≠ ∘C(Q) /
            ≠NUMBER≠ ∘C(N) / ≠RESET≠ ∘C(FF)  )
         .E(≠∘∘∘EXPECT RW AFTER . ≠, ≠,. ≠, STATEMENT)  /
    .NUMBER ∘C(∘N)  ,.

GENERATOR = ≠∘≠  (
    ≠C≠ ≠(≠ .E(≠∘∘∘NO ( ≠, ≠,. ≠, STATEMENT)
        CODEFIELD .E(≠∘∘∘NO CODEFILED ≠, ≠,. ≠, STATEMENT)
              $( ≠,≠  CODEFIELD
           .E(≠∘∘∘NO CODEFIELD ≠, ≠,. ≠, STATEMENT))
                    ≠)≠ .E(≠∘∘∘NO ) ≠, ≠,. ≠, STATEMENT)  /
    ≠D≠ ≠(≠ .E(≠∘∘∘NO ( ≠, ≠,. ≠, STATEMENT)
        ( ≠∘R≠ ∘C(SR) ≠,≠ .E(≠∘∘∘NO , ≠, ≠,. ≠, STATEMENT)
                   / .EMPTY )                ≠L ∘C(SL)  ≠CL ∘C(DC)
          ( ≠∘1≠ ∘C(D1) / ≠∘2≠ ∘C(D2) / ≠∘≠ ∘C(D) )
        ≠)≠ .E(≠∘∘∘NO ) ≠, ≠,. ≠, STATEMENT)     )
    .E(≠∘∘∘EXPECT C/D AFTER ∘ ≠, ≠,. ≠, STATEMENT)  ,.

CODEFIELD = .ID ∘C(A,∘) / ≠∘1≠ ∘C(G1) / ≠∘2≠ ∘C(G2) /
```

'*L' *C(LK)   *'CL' + CURSOR

```
         ≠∘R≠ ∘C(SR) / ≠∘Q≠ ∘C(SQ) / ≠∘N≠ ∘C(SN) /
         ≠∘≠  ∘C(S)  / .NUMBER ∘C(L,∘N)  ,.

CONTROL = ≠+≠  (
              ≠ANCHOR≠      ∘C(AT) / ≠UNANCHOR≠ ∘C(AF) /
              ≠LIST≠        ∘C(LT) / ≠UNLIST≠   ∘C(LF) /
              ≠RECMODE≠     ∘C(RT) / ≠STREAM≠   ∘C(RF) /
              ≠XECUTE≠      ∘C(XC) / ≠RETURN≠   ∘C(XR) /
              ~~≠SAVE≠      ∘C(SL) / ≠LINK≠     ∘C(LK) /~~   'CURSOR'  *C(CN)
              ≠ENDKLUDGE≠   ∘C(EK) / ≠B≠        ∘C(DB) /
              ≠TRUNCATE≠    ∘C(TC) / ≠LOAD≠     ∘C(LD) /     'XUI'  *C(UI)
              ≠PUNCH≠       ∘C(PT) / ≠UNPUNCH≠  ∘C(PF) /
              ≠PK≠          ∘C(P)  / ≠K≠        ∘C(K) /
              ≠STOP≠        ∘C(H)  / ≠SNAP≠ .NUMBER ∘C(SP,∘N) )
      .E(≠∘∘∘EXPECT RW AFTER + ≠, ≠,. ≠, STATEMENT)  ,.

DIRECTIVE =  ≠-≠  (
      ≠LIST≠      +LIST     / ≠UNLIST≠        +UNLIST  /
      ≠RECMODE≠   +RECMODE  / ≠STREAM≠        +STREAM /
      ≠PUNCH≠     +PUNCH    / ≠UNPUNCH≠       +UNPUNCH /
      ≠ANCHOR≠    +ANCHOR   / ≠UNANCHOR≠      +UNANCHOR /
      ≠TRUNCATE≠  +TRUNCATE / ≠XECUTE≠ +XECUTE /
      ≠RETURN≠    +RETURN   / ≠LOAD≠          +LOAD /      'XUI'  +XUI
      ≠STOP≠      +STOP     / ≠SNAP≠ .NUMBER +SNAP 63  )
         .E(≠∘∘∘EXPECT RW AFTER - ≠, ≠,. ≠, STATEMENT)   ,.

                                                            'FLIST'  +FLIST
.END
```

'FLIST' *C(LA) /

```
PROGRAM$F$T$TS$C$R$BE$E$A$L$I$Q$N$FT$FF$S$SQ$SN$SR$D$D1$D2$G1$G2$SP$SL$LK$EK$AT
$AF$LT$LF$PT$PF$RT$RF$XC$XR$P$K$DB$TC$B$LD$H$PROGRAM∘TS '.META'∘F 1$2∘C DIRECTIV
E∘T 2∘FT∘E∘I∘E∘S∘TS '.OPCODES'∘F 3$4∘I∘F 5∘D$5$6∘T 4∘FT∘E∘TS ',.'∘E$3∘T 7∘FT∘F 8
$8$7∘E∘TS '.RESERVEDWORDS'∘F 9$10∘I∘F 11∘S ∘D$11$12∘T 10∘FT∘E∘TS ',.'∘E$9∘T 13∘F
T∘F 14$14$13∘E$15∘C STATEMENT∘T 15∘FT∘E∘TS '.END'∘E∘XC∘H$1$16∘R$STATEMENT∘I∘F 17
∘D∘TS '='∘BE '∘∘∘NO = ' ',. ' STATEMENT∘C EXPRESSION∘BE '∘∘∘EXPECT EXP ' ',. ' S
TATEMENT∘A R∘TS ',.'∘BE '∘∘∘NO ,. ' ',. ' STATEMENT$18∘C DIRECTIVE∘T 18∘FT∘E$17$
19∘R$EXPRESSION∘C SUBEXPRESSION∘F 20$21∘TS '/'∘F 22∘A T∘G1∘C SUBEXPRESSION∘E$22$
23∘T 21∘FT∘E∘D1$20$24∘R$SUBEXPRESSION∘C ELEMENT∘F 25∘A F∘G1$25∘T 26∘C GENERATOR∘
F 27$27∘T 26∘C CONTROL∘F 28$28∘T 26∘C DIRECTIVE∘F 29$29$26∘F 30$31∘C ELEMENT∘F 3
2∘TS '.E'∘F 33∘TS '('∘BE '∘∘∘NO ( ' ',. ' STATEMENT∘Q∘BE '∘∘∘NO STRING ' ',. ' S
TATEMENT∘A BE∘SQ∘TS ','∘BE '∘∘∘NO , ' ',. ' STATEMENT∘Q∘BE '∘∘∘NO STRING ' ',. '
 STATEMENT∘SQ∘TS ','∘BE '∘∘∘NO , ' ',. ' STATEMENT∘I∘BE '∘∘∘NO ID ' ',. ' STATEM
ENT∘S∘TS ')'∘BE '∘∘∘NO ) ' ',. ' STATEMENT 33∘T 34∘FT∘F 35∘A E$35$34∘E$32∘T 36∘C
 GENERATOR∘F 37$37∘T 36∘C CONTROL∘F 38$38∘T 36∘C DIRECTIVE∘F 39$39$36∘T 31∘FT∘E∘
D1$30$40∘R$ELEMENT∘Q∘F 41∘A TS∘SQ$41∘T 42∘I∘F 43∘A C∘S$43∘T 42∘TS '$'∘F 44∘D1∘C
ELEMENT∘BE '∘∘∘EXPECT ELEMENT ' ',. ' STATEMENT∘A T∘G1∘A FT$44∘T 42∘TS '('∘F 45∘
C EXPRESSION∘BE '∘∘∘NO EXP ' ',. ' STATEMENT∘TS ')'∘BE '∘∘∘UNBAL ( ' ',. ' STATE
MENT$45∘T 42∘TS '.'∘F 46∘TS 'ID'∘F 47∘A I$47∘T 48∘TS 'EMPTY'∘F 49∘A FT$49∘T 48∘T
S 'STRING'∘F 50∘A Q$50∘T 48∘TS 'NUMBER'∘F 51∘A N$51∘T 48∘TS 'RESET'∘F 52∘A FF$52
$48∘BE '∘∘∘EXPECT RW AFTER . ' ',. ' STATEMENT$46∘T 42∘N∘F 53∘SN$53$42∘R$GENERAT
OR∘TS '∘'∘F 54∘TS 'C'∘F 55∘TS '('∘BE '∘∘∘NO ( ' ',. ' STATEMENT∘C CODEFIELD∘BE '
∘∘∘NO CODEFILED ' ',. ' STATEMENT$56∘TS ','∘F 57∘C CODEFIELD∘BE '∘∘∘NO CODEFIELD
 ' ',. ' STATEMENT$57$58∘T 56∘FT∘E∘TS ')'∘BE '∘∘∘NO ) ' ',. ' STATEMENT$55∘T 59∘
TS 'D'∘F 60∘TS '('∘BE '∘∘∘NO ( ' ',. ' STATEMENT∘TS '∘R'∘F 61∘A SR∘TS ','∘BE '∘∘
∘NO , ' ',. ' STATEMENT$61∘T 62∘FT∘F 63$63$62∘E∘TS '∘1'∘F 64∘A D1$64∘T 65∘TS '∘2
'∘F 66∘A D2$66∘T 65∘TS '∘'∘F 67∘A D$67$65∘E∘TS ')'∘BE '∘∘∘NO ) ' ',. ' STATEMENT
$60$59∘BE '∘∘∘EXPECT C/D AFTER ∘ ' ',. ' STATEMENT$54$68∘R$CODEFIELD∘I∘F 69∘A A∘
S$69∘T 70∘TS '∘1'∘F 71∘A G1$71∘T 70∘TS '∘2'∘F 72∘A G2$72∘T 70∘TS '∘R'∘F 73∘A SR$
73∘T 70∘TS '∘Q'∘F 74∘A SQ$74∘T 70∘TS '∘N'∘F 75∘A SN$75∘T 70∘TS '∘'∘F 76∘A S$76∘T
 70∘N∘F 77∘A L∘SN$77$70∘R$CONTROL∘TS '+'∘F 78∘TS 'ANCHOR'∘F 79∘A AT$79∘T 80∘TS '
UNANCHOR'∘F 81∘A AF$81∘T 80∘TS 'LIST'∘F 82∘A LT$82∘T 80∘TS 'UNLIST'∘F 83∘A LF$83
∘T 80∘TS 'RECMODE'∘F 84∘A RT$84∘T 80∘TS 'STREAM'∘F 85∘A RF$85∘T 80∘TS 'XECUTE'∘F
 86∘A XC$86∘T 80∘TS 'RETURN'∘F 87∘A XR$87∘T 80∘TS 'SAVE'∘F 88∘A SL$88∘T 80∘TS 'L
INK'∘F 89∘A LK$89∘T 80∘TS 'ENDKLUDGE'∘F 90∘A EK$90∘T 80∘TS 'B'∘F 91∘A DB$91∘T 80
∘TS 'TRUNCATE'∘F 92∘A TC$92∘T 80∘TS 'LOAD'∘F 93∘A LD$93∘T 80∘TS 'PUNCH'∘F 94∘A P
T$94∘T 80∘TS 'UNPUNCH'∘F 95∘A PF$95∘T 80∘TS 'PK'∘F 96∘A P$96∘T 80∘TS 'K'∘F 97∘A
K$97∘T 80∘TS 'STOP'∘F 98∘A H$98∘T 80∘TS 'SNAP'∘F 99∘N∘E∘A SP∘SN$99$80∘BE '∘∘∘EXP
ECT RW AFTER + ' ',. ' STATEMENT$78$100∘R$ IRECTIVE∘TS '-'∘F 101∘TS 'LIST'∘F 102
∘LT$102∘T 103∘TS 'UNLIST'∘F 104∘LF$104∘T 103∘TS 'RECMODE'∘F 105∘RT$105∘T 103∘TS
'STREAM'∘F 106∘RF$106∘T 103∘TS 'PUNCH'∘F 107∘PT$107∘T 103∘TS 'UNPUNCH'∘F 108∘PF$
108∘T 103∘TS 'ANCHOR'∘F 109∘AT$109∘T 103∘TS 'UNANCHOR'∘F 110∘AF$110∘T 103∘TS 'TR
UNCATE'∘F 111∘TC$111∘T 103∘TS 'XECUTE'∘F 112∘XC$112∘T 103∘TS 'RETURN'∘F 113∘XR$1
13∘T 103∘TS 'LOAD'∘F 114∘LD$114∘T 103∘TS 'STOP'∘F 115∘H$115∘T 103∘TS 'SNAP'∘F 11
6∘N∘E∘SP∘63$116$103∘BE '∘∘∘EXPECT RW AFTER - ' ',. ' STATEMENT$101$117∘R/
```

OBSOLETE

***** sect12d *****

12d Syntax equations for SIMPL (A minor subset of PL/I). These equations have been used both on the 360 and 3100 versions of LOT to compile the test cases following the syntax. The resemblence to PL/I is fairly superficial, and the equations merely were used to see if some of the styles of if and do instructions could be handled without extensions to LOT.

```
opcodes eob b bfp dot jmp ldn lda stn sto cat or and gt ge ne le lt eq add sub
 mul div pwr ldc mks sbs  up um flp ldl lds red wrt bob  ,.

.reservedwords end do if then else begin to by while cat or and not go le lt
ne gt ge read write            ,.

 program  = $directives block  "*eof*" ←stop  ,.

 directives = "-" ( "punch" +punch / "list" +list )  ,.

 block = beginblock  ,.

 beginblock = "begin" *c(bob) ",." $(blockcontents ←truncate) *c(eob) endst ,.

 blockcontents = gb  ,.

 gb = st / block ,.

 endst = "end" (.id +endkludge / .empty )  ",." ,.

 st = dost / ifst / gotost / iost /
 .id ( ".." *d(*) st / assignst ) ,.

 dost = "do" (
 ",." gb  endst /
 "while" *d(*1) exp *c(bfp,*2) ",."
 gb endst *c(b,*1) *d(*2) /
variable "=" dorage *c(dot,*1) $(",." / "," dorage *c(dot,*1) )
 *c(b,*2) *d(*1) $gb endst *c(jmp) *d(*2)  ) ,.

dorage = exp *c(sto,ldl)
          (  "to" exp ( "by" exp / .empty *c(ldn) )  /
 "by" exp ( "to" exp / .empty *c(ldn) )  *c(flp) /
 .empty *c(ldn,ldn)  )
 ( "while" exp / .empty *c(ldn) ) ,.

 ifst = "if" exp "then" *c(bfp,*1) st
 ( "else" *c(b,*2) *d(*1) st *d(*2) / .empty *d(*1) ) ,.

 gotost = "go" "to" variable *c(jmp) ",."  ,.

 iost = "read" "(" *c(mks) variable  $( "," variable )  ")" *c(red) ",." /
      "write" "("  *c(mks) exp $( "," exp )  ")" *c(wrt) ",." ,.

assignst = *c(lda,*) subscript assign2 *c(sto) ",." ,.

assign2 = "," .id *c(lda,*) subscript assign2 *c(stn) / "=" exp ,.
```

```
exp = bfactor $( "cat" bfactor *c(cat) ) ,.

bfactor = bterm $( "or" bterm *c(or) ) ,.

bterm = bprimary $( "and" bprimary *c(and) ) ,.

bprimary = "not" relation *c(not) / relation ,.

relation = aexp ( "gt" aexp *c(gt) / "ge" aexp *c(ge) /
"ne" aexp *c(ne) / "le" aexp *c(le) /
"lt" aexp *c(lt) / "=" aexp *c(eq) / .empty ) ,.

aexp  = term$( "+"term *c(add) / "-" term *c(sub) ) ,.

term = factor $( "*" factor *c(mul) / "/" factor *c(div) ) ,.

factor = primary $("**" factor *c(pwr) ) /
         "+" factor *c(up) / "-" factor *c(um) ,.

primary = variable / .number *c(ldc,*n) / "(" exp ")" /
      .string  *c(lds,*q)  ,.

variable = .id *c(lda,*) subscript ,.

subscript = ( "(" *c(mks) exp *c(sbs) ")" / .empty ) ,.

.end /* end          s i m p l   c o m p i l e r */

/* simpl pl/i test cases */        -list   -punch
begin,.     /* use a block around whole all of simpl */

      read(a,b,c) ,.
   read( a(i), b(j+25*i-1/k), c) ,.
     write(a,b,c) ,.
     write(a,b,c, "a string" , 25+3*a, ans+2, ix or iy) ,.



   /* pl/1  est for our simpl simpl   */
x=a+b-c*d/e**g*(a+b**-c) ,.

if x lt y then a=b,. else a=c,.

    do ,. x=y ,. end ,.

    do x= 1,. z=q,. p=r ,. end,.

  a,b,c=r,.
 a(i)=v ,.
c(i)=q(i),.
 a(i+b(j)-35),q(12-a**x) = z(i+4) ,.
 do i=1 to 5 ,. z=q,. r=t,. end ,.
    do i=1, 3, 4 ,. z=y ,. t=3,. end,.



if x then if y then go to a ,. else go to b ,. else go to c ,.

if x then go to a ,. else if y then go to b ,. else go to c ,.
```

```
do while x ,. if y then do i = 1 to 10 by 2 ,. y=a(i) ,. end,.
   else go to c ,. end ,.
a=b=c ,.

 bigstring = "bittystring here " cat stringvariable  ,.

   /* the hard way to get to gamma is */
if a then alpha..beta.. go to gamma,. else delta.. go to alpha ,.

do i = 1 by 2 to 10 ,. x(i)=i,. end ,.

do i = 1 to 10 by 2 while a ,. x(i)= b or (not c and d) ,. end ,.

x = a and b or c and d or e and f and not x ,.

 begin,.   x=y,. end,.


     do i=3 to 7,4,j to 5 by 6,.
   al = a(i+j) ,.
 end,.

   do i=3 to 5 while x gt 7 ,.
       j=74,. k=63,.
            do l=j-3 to 6 by 5 ,.
                   m=3,.
            end,.
     end,.

   z,q,r(q)=z(q(r(i)))  ,.
 end,.       /* end of big block */
 /*     end      file     mark      */     *eof*
```

```
.META
-SNAP 2
       PROGRAM  /* PL/1 LIKE LANGUAGE TO ALLOW PRODUCTION OF EFFICIENT
                       MACHINE CODE FOR A CDC 3100 AND AN SDS 940  */

.OPCODES
B    /* BRANCH                       */
BF   /* BRANCH FALSE                 */
BT   /* BRANCH TRUE
BFP  /* BRANCH FALSE AND POP         */
BDN  /* BUMP DOWN ONE                */
BUP  /* BUMP UP ONE                  */
CLL  /* PROCEDURE CALL               */
D    /* DECLARE 1 DIMENSION, 2 INTEGER, 3 CHARACTER, 4 LABEL, 5 PROCEDURE,
              6 LABEL DEFINATION, 7 START INITIAL, 8 INITIALIZE, 9 ENTRY */
DOI  /* DO ITERATION                 */
DOT  /* DO TEST                      */
EOP  /* END OF PROGRAM               */
FLP  /* FLIP                         */
ITR  /* SUB 1 THEN BFP               */
JMP  /* JUMP THRU THE STACK          */
LCM  /* LOGICAL COMPLEMENT           */
LDC  /* LOAD DECIMAL CONSTANT        */
LNA  /* LOAD NULL ARGUMENT           */
LOC  /* LOAD OCTAL CONSTANT          */
LSA  /* LOAD SYMBOLIC ADDRESS        */
LSC  /* LOAD STRING CONSTANT         */
MKS  /* MARK THE STACK               */
OPC  /* OPERAND CALL                 */
POP  /* POP THE STACK                */
PTM  /* POP STACK TO MARK            */
RED  /* READ                         */
RET  /* PROCEDURE RETURN             */
RSM  /* REMOVE STACK MARK            */
RST  /* PUSH FALSE ON STACK          */
SET  /* PUSH TRUE ON STACK           */
SFR  /* SUBSCRIPT OR FUNCTION REFERENCE */
SLA  /* SKIP AND LOAD ADDRESS        */
SSD  /* STORE SEMI-DESTRUCTIVE       */
STN  /* STORE NON-DESTRUCTIVE        */
STO  /* STORE DESTRUCTIVE            */
WRT  /* WRITE                        */

/* ARITHMETIC OPS */ XOR OR AND COM GT GE NE LE LT EQ ADD SUB MUL DIV MOD
                    UP UM MAX MIN
  ,.

.RESERVEDWORDS
PROCEDURE DECLARE DEC INTEGER CHARACTER CHAR LABEL INITIAL INIT CALL
END ENTRY DO WHILE TO BY IF THEN ELSE BUMP UP DOWN GO RETURN SET
   OR AND NOT GT GE NE LE LT MOD MAX MIN TRUE FALSE COMPL READ WRITE
  ,.
```

SOB START OF BLOCK

EOB END OF BLOCK

DFP DECLARE FORMAL PARAMETER

MC MOVE CURSOR

SB

~~Bt~~ Bt LAD load address

BEGIN

```
PROGRAM =
  ≠.SIMPLE≠  $DIRECTIVES      a
 .ID ∘C(MKS,LSA,∘) ≠..≠ $( .ID ∘C(LSA,∘) ≠..≠ )        *C(PTM,MKS)
   ≠PROCEDURE≠ ∘C(D,61,B) +SAVE          ARGUMENT.LIST  ≠,.≠
     ∘D(∘1) $PROGRAM.ELEMENT END ∘C(B,∘2) +LINK ∘C(B,∘1)
       ∘D(∘2) ∘C(EOP) +TRUNCATE  ~~+STOP ,.~~   + XUI  +RETURN ;.

DIRECTIVES = ≠-≠ (  ≠LIST≠ +LIST / ≠PUNCH≠ + PUNCH )  ,.
               '(' .ID *C(DFP,*) $( ',' .ID *C(DFP,*) ) ')'
ARGUMENT.LIST = ~~ARB.ARG~~ / .EMPTY  ,.

PROGRAM.ELEMENT = GROUP $GROUP ∘C(B,∘1) $DECLARE ∘D(∘1) / DECLARE  ,.

DECLARE = ( ≠DECLARE≠ / ≠DEC≠ ) +LINK ∘C(MKS) DECLARATION
   $( ≠,≠ ∘C(PTM,MKS) DECLARATION) ∘C(PTM,B) +SAVE ,.

DECLARATION = .ID ∘C(LSA,∘) (DIMENSION / .EMPTY) $ATTRIBUTE /
   ≠(≠ ∘C(MKS) DECLARATION $(≠,≠ DECLARATION) ≠)≠
      ∘C(D,0) $ATTRIBUTE ∘C(RSM)  ,.
                                                                       $PTM
                          PAIR                  PAIR
DIMENSION = ≠(≠ ∘C(MKS) ~~EXP ≠..≠ EXP~~      $( ≠,≠ ~~EXP ≠..≠ EXP~~) ≠)≠ ∘C(D,1) ,.
                                  PAIR = EXP ('..' EXP /.Empty *C(LNA,FLP))
ATTRIBUTE = TYPE / INITIAL  ,.

TYPE = ≠INTEGER≠ ∘C(D,2) / ( ≠CHARACTER≠ / ≠CHAR≠ ) ∘C(D,3) /
      ≠LABEL≠ ∘C(D,4) / ≠PROCEDURE≠ ∘C(D,5) / .EMPTY ∘C(D,1)  ,.

INITIAL = ( ≠INITIAL≠ / ≠INIT≠ ) ( ≠(≠ ∘C(MKS,D,7) ITEM $( ≠,≠ ITEM) ≠)≠  /
          ≠CALL≠ ( ARB.ARG / .EMPTY)  )  ,.

ITEM =  ( CONSTANT / ≠∘≠ ∘C(LNA) / ITERATION ) ∘C(D,8)  ,.

ITERATION = ≠(≠ EXP ≠)≠ ∘D(∘1)  ( CONSTANT / ≠∘≠ ∘C(LNA) /
                                 ≠(≠ ITEM $( ≠,≠ ITEM) ≠)≠  )  ∘C(ITR,∘1) ,.

END = ≠END≠ ( .ID +ENDKLUDGE / .EMPTY ) ≠,.≠  ,.

GROUP = LABELED.GROUP / UNLABELED.GROUP  ,.

LABELED.GROUP =  LABEL   ( ≠..≠ ∘C(B,∘1) +LINK ∘C(MKS,LSA,∘)
     MORE.LABELS ( ENTRY / .EMPTY ) ∘C(D,6,PTM,B) +SAVE ∘D(∘1)
         ( UNLABELED.GROUP / ASSIGN ) / ASSIGN ) ,.

MORE.LABELS = LABEL (≠..≠ ∘C(LSA,∘)        MORE.LABELS / .EMPTY) / .EMPTY ,.
                                  / BEGIN
UNLABELED.GROUP = DO / IF / SIMPLE  ,.

ITERATIVE.GROUP = LABELED.I.GROUP / UNLABELED.GROUP  ,.

LABELED.I.GROUP =  LABEL   ( ≠..≠ ∘C(B,∘1) +LINK ∘C(MKS,LSA,∘)
     MORE.LABELS ∘C(D,6,PTM,B) +SAVE ∘D(∘1)
         ( UNLABELED.GROUP / ASSIGN ) / ASSIGN ) ,.

LABEL = .ID ,.

BEGIN = 'BEGIN' *C(SOB) ';' *D(*1) $PROGRAM.ELEMENT
    END *C(EOB) *C(B,*2) +LINK *C(B,*1) *D(*2) ;.
```

```
ASSIGN = ∘C(LSA,∘) SUBSCRIPT ASSIGN2 ∘C(STO) ,.

ASSIGN2 = ≠,≠ VARIABLE ASSIGN2 ∘C(STN) / ≠=≠ EXP  ,.

ENTRY = ≠ENTRY≠ ∘C(D,9,PTM,B) +SAVE ∘C(0,0)  ARGUMENT.LIST  ≠,.≠  ,.

DO = ≠DO≠ ( ≠,.≠ ∘D(∘1) $GROUP END /
           ≠WHILE≠ ∘D(∘1) EXP ∘C(BFP,∘2) ≠,.≠ $ITERATIVE.GROUP
                   END ∘C(B,∘1) ∘D(∘2)  /
           VARIABLE ≠=≠ DORANGE ∘C(DOT,∘1) $( ≠,.≠ / ≠,≠ DORANGE ∘C(DOT,∘1) )
                    ∘C(B,∘2) ∘D(∘1) $ITERATIVE.GROUP END
                         ∘C(JMP) ∘D(∘2)  )   ,.

DORANGE = EXP ∘C(SSD)
          ( ≠TO≠ EXP ( ≠BY≠ EXP / .EMPTY ∘C(LNA) ) /
            ≠BY≠ EXP ( ≠TO≠ EXP / .EMPTY ∘C(LNA) ) ∘C(FLP) /
            .EMPTY ∘C(LNA,LNA)  ) ∘C(SLA,DOI)  (LAD, B, *1, DOII) *D(*1)
          ( ≠WHILE≠ EXP / .EMPTY ∘C(LNA) )   ,.

IF = ≠IF≠ EXP ≠THEN≠ ∘C(BFP,∘1) GROUP
       ( ≠ELSE≠ ∘C(B,∘2) ∘D(∘1) GROUP ∘D(∘2) / .EMPTY ∘D(∘1)  )  ,.

SIMPLE = ( BUMP / CALL / GOTO / RETURN / SET / READ / WRITE ) ≠,.≠ / ≠,.≠  ,.

BUMP = ≠BUMP≠ VARIABLE ( ≠UP≠ ∘C(BUP) / ≠DOWN≠ ∘C(BDN) / .EMPTY ∘C(BUP) ) ,.

CALL = ≠CALL≠ VARIABLE  ( ARB.ARG / .EMPTY)  ∘C(CLL)  ,.

                                      GØ
GOTO = ≠GO≠ ≠TO≠ VARIABLE ∘C(JMP) ,.

RETURN = ≠RETURN≠ ( ≠(≠ EXP ≠)≠ / .EMPTY ) ∘C(RET) ,.

SET = ≠SET≠ VARIABLE ≠TO≠ CONSTANT ∘C(STO) ,.

READ = ≠READ≠ ( ≠LIST≠ READ.DATA.LIST / ≠DATA≠ ) ,.

READ.DATA.LIST = VARIABLE ∘C(RED) $( ≠,≠ READ.DATA.LIST) /
     ≠DO≠ VARIABLE ≠=≠ DORANGE ∘C(DOT,∘1) $(≠,.≠/≠,≠ DORANGE ∘C(DOT,∘1) )
          ∘C(B,∘2) ∘D(∘1) READ.DATA.LIST ∘C(JMP) ∘D(∘2) ,.

WRITE = ≠WRITE≠ ( ≠LIST≠ WRITE.DATA.LIST / ≠DATA≠ ) ,.

WRITE.DATA.LIST = EXP ∘C(WRT) $( ≠,≠ WRITE.DATA.LIST ) /
     ≠DO≠ VARIABLE ≠=≠ DORANGE ∘C(DOT,∘1) $(≠,.≠/≠,≠ DORANGE ∘C(DOT,∘1) )
          ∘C(B,∘2) ∘D(∘1) WRITE.DATA.LIST ∘C(JMP) ∘D(∘2)  ,.

EXP  =  UNION  ,.

UNION             = INTERSECTION ( ≠OR≠ ∘C(BT,∘1,POP) UNION ∘D(∘1) / .EMPTY ) ,.

INTERSECTION      = NEGATION ( ≠AND≠ ∘C(BF,∘1,POP) INTERSECTION ∘D(∘1) / .EMPTY ),.

NEGATION          = ≠NOT≠ NEGATION ∘C(LCM) / RELATION  ,.
```

```
RELATION        = SUM ( ≠GT≠ SUM ∘C(GT) / ≠GE≠ SUM ∘C(GE) /
                        ≠NE≠ SUM ∘C(NE) / ≠LE≠ SUM ∘C(LE) /
                        ≠LT≠ SUM ∘C(LT) / ≠=≠  SUM ∘C(EQ) /
                        .EMPTY  )  ,.

SUM             = PRODUCT $( ≠+≠ PRODUCT ∘C(ADD) / ≠-≠ PRODUCT ∘C(SUB) ) ,.

PRODUCT         = FACTOR $( ≠∘≠ FACTOR ∘C(MUL) / ≠/≠ FACTOR ∘C(DIV) ) ,.

FACTOR          = BDIFFERENCE $( ≠MOD≠ FACTOR ∘C(MOD)  )
                  ≠+≠ FACTOR ∘C(UP) / ≠-≠ FACTOR ∘C(UM)  ,.

BDIFFERENCE     = BSUM $( ≠.-≠ BSUM ∘C(XOR) ) ,.

BSUM            = BPRODUCT $( ≠.+≠ BPRODUCT ∘C(OR) ) ,.

BPRODUCT        = COMPLEMENT $( ≠.∘≠ COMPLEMENT ∘C(AND) ) ,.

COMPLEMENT      = ≠COMPL≠ COMPLEMENT ∘C(COM) / UNIT   ,.

UNIT            = VARIABLE / CONSTANT / BUILT.IN.FUNCTION /
                   ≠(≠ EXP ≠)≠ ∘C(OPC)  ,.

VARIABLE        = .ID ∘C(LSA,∘) SUBSCRIPT ,.

SUBSCRIPT       = ARB.ARG ∘C(SFR) / .EMPTY ,.

ARB.ARG         = ≠(≠ ∘C(MKS) EXP $( ≠,≠ EXP )  ≠)≠  ,.

CONSTANT        = .NUMBER ( ≠B≠ ∘C(LOC,∘) / .EMPTY ∘C(LDC,∘) ) /
                  .STRING ∘C(LSC,∘) /
                  ≠TRUE≠ ∘C(SET) / ≠FALSE≠ ∘C(RST) ,.

BUILT.IN.FUNCTION = ≠MAX≠ ARB.ARG ∘C(MAX) /
                    ≠MIN≠ ARB.ARG ∘C(MIN) ,.

.END

.SIMPLE -LIST -PUNCH
XID.. PROCEDURE ,.
  BUMP IMSBU(1),. CALL DELBLK,. MFLAG=FALSE ,.
  IF IAT(W(1)) = LTR THEN
    DO ,. BUMP IW,.
      DO IW = IW WHILE IAT(W(IW)) LE IPNT AND IAT(W(IW)) ,.
          IF IW GT MAXLEN THEN CALL ERROR ,.
   END ,. END ,.
  DO IW = IW BY -1 WHILE IAT(W(IW)) = IPNT ,. ,. END ,.
  CALL CLEAR ,. CALL PACK ,.
  IF IRWFLAG THEN
    DO ,. I=IATOM,. CALL ASR(ISATOM-1),.
      IF I AND LATOM(I) .∘ IRSWD THEN DO ,. IW=1,. RETURN,. END ,.
    END ,.
  ELSE MFLAG = TRUE ,. RETURN ,.
```

```
LB1.. CALL SUB ,.
LB2..LB3.. X=Y ,.
LB4..LB5..LB6..X,Y,Z=A=B ,.
A=B AND C OR D AND E OR F AND G OR E AND F OR G AND H OR I AND J OR K AND L ,.
IF A THEN LB7..DO ,. CALL SUB1,. RETURN,. END ,. ELSE LB8.. DO ,.
LB9..IF A AND B OR C AND D THEN LB10..RETURN,. LB11.. CALL SUB1 ,.     END ,.
DECLARE (A(0..10),B,C(10,10) INITIAL (0,°,0 (15°I) (0,°,0 (17) °) ) ) INTEGER ,.
L1..DO,.I=1,.L2..DO,.I=2,.L3..DO,.I=3,.L4..DO,.I=4,.END L3,. END L1 ,.
END ,.
```

```
***** sect12e *****

.meta  program     /* pl/1  syntax equations */


.opcodes  /* fill in later */ ;
.reservedwords  /* fill in later */ ;

program = $( directive / begin) "*eof*"  +stop  ,.

directive = "-" ("punch" +punch / "list" +list ) ,.

begin = "begin" *c(bob) ",." *c(b) +save *c(0,0) *d(*1)
   $blockcotents endst *c(b,*2) +link *c(b,*1) *d(*2) *c(eob) +truncate  ,.

blockcontents = $gb *c(b,*1) $prex *d(*1)  ,.

prex = ( declare / procedure / entry / format / implicit ) +truncate  ,.




endst = ("end" (.id +endkludge / .empty) / .empty) ",."  ,.




declare = ( "declare" / "dec" ) +link *c(mks) declaration
   $( "," *c(ptm,mks) declaration) *c(ptm,b) +save *c(0,0) endst  ,.

declaration = (.number *c(ni) / .empty)
   .id *c(lda,*) (dimension / .empty) $attribute /
   "(" *c(mks) declaration $("," declaration) ")"
      *c(d,0) $attribute *c(rsm)  ,.

attribute = data / secondary / abnormal / uses / entryn / scope / storage /
   aligned / defined / position / initial / symbol / like / file  ,.

data = arithmetic / string / label / "task" *c(ni) / "event" *c(ni) ,.

arithmetic = base scale mode (precision/.empty) /
   ( "picture" / "pic" ) .string *c(ni)  ,.

base = ("binary" / "bin") *c(d,1) /
   ("decimal" / "dec") *c(ni) / .empty *c(d,1)  ,.

scale = "fixed" *c(d,3) / "float" *c(ni) / .empty *c(d,3)  ,.

mode = "real" *c(d,5) / ("complex" / "cplx") *c(ni) / .empty *c(d,5) ,.

precision = "(" .number ("," .number /.empty) ")" *c(ni)  ,.

string = ("bit" *c(d,7) / ("character"/"char" *c(d,8) )
   "(" ( exp *c(dsl) / "*" *c(ni) ) ")"
   ( ("varying"/"var") *c(d,10) / .empty) /
   ("picture"/"pic") .string *c(ni)  ,.
```

```
label = "label" ( "(" .id $("," .id) ")" / .empty) *c(d,12)  ,.

dimension = "(" bound $("," bound) ")" ,.

bound = "*" / exp ".." exp ,.

secondary = "secondary"  ,.

abnormal = ("abnormal"/"abnl") *c(ni) / "normal"  ,.

uses = ("uses"/"sets") "(" usitem $("," usitem) ")" *c(ni)  ,.

usitem = .number / .id / "*"  ,.

entryn = entrya / generic / builtin  ,.

entrya = "entry" "(" pal $("," pal) ")" *c(ni) ,.

pal = (.number / dimension / attribute) $attribute  ,.

generic = "generic" "(" endec $("," endec) ")" *c(ni)  ,.

endec = .id $attribute  ,.

scope = "internal"/"int" / ("external"/"ext") "(" .id ")" *c(ni)  ,.

storage = "static" *c(ni) / ("automatic"/"auto")/("controlled"/"ctl") *c(ni) ,.

aligned = "aligned" / "packed"  ,.

defined = ("defined" "def") variable *c(ni)  ,.

position = ("position"/"pos") .number *c(ni)  ,.

initial = (("initial"/"init") ( "(" item $("," item) ")" /
   "call" .id subscript  )  *c(ni)  ,.

item = ele   "*" / iteration ,.

ele = (""/" "- empty) constant ((" "/"") constant  .empty )  ,.

iteration = "(" exp ")" (ele   "*"   "(" item $("," item) ")" )  ,.

 ymbol = symbol  "(  .id ")    "nosymbol"  ,.

 ike   "li e"  id  c(n )   .

ile = . es t  .



 b       o        e  beg n ) truncate ,

                     the   do ange afte  the      --  th's must be fixed  */
do = "do" (
   ",." gb endst /
   "while" *d(*1) exp *c(bfp,*2) ",." gb endst *c(b,*1) *d(*@
   variable "=" dorange *c(dot,*1) $/". "/"." dorange *c(dot,*1) )
      *c(b,*2) *d(*1) gb endst *c(jmp) *d(*2)  )  ,.
```

```
dorange = exp *c(sto,ldl)
   (  "to" exp ( "by" exp / .empty *c(ldn) ) /
      "by" exp ( "to" exp / .empty *c(ldn) ) *c(flp) /
      .empty *c(ldn,ldn)  ) /
   (  "while" exp / .empty *c(ldn)  )   ,.

simple = call / goto / if / null / read / return / write /
   .id ( ".." *d(*) simple / assign )  ,.

call = "call" variable *c(cll) endst ,.

goto = "go" "to" variable *c(jmp) endst ,.

if = "if" exp "then" *c(bfp,*1) gb
   ("else" *c(b,*2) *d(*1) gb *d(*2) / .empty *d(*1)  )  ,.

null = endst ,.

read = "read" .id $(dataspec) ("print" /.empty)  ,.

return = "return" ( "(" exp ")" / .empty ) endst ,.

write = "write" .id $(dataspec)  ,.

assign = *c(lda,*) subscript assign2 *c(sto)
   ( "," "by" "name" *c(ni) / .empty) endst  ,.

assign2 = "," variable assign2 *c(stn) / "=" exp  ,.

exp = union $( "cat" union *c(cat) ) ,.

union = intersection $( "or" intersection *c(or) ) ,.

intersection = negation $( "and" negation *c(and) ) ,.

negation = "not" relation *c(not) / relation ,.

relation = sum("gt" sum *c(gt) / "ge" sum *c(ge) /"ne" sum *c(ne) /
   "le" sum *c(le) / "lt" sum *c(lt) / "=" sum *c(eq) / .empty ) ,.

sum = product $( "+" product *c(add) / "-" product *c(sub) ) ,.

product = factor $( "*" factor *c(mul) / "/" factor *c(div) ) ,.

factor = unit "**" factor *c(pwr) /
   "+" factor *c(up) / "-" factor *c(um)  ,.

unit = variable / constant / "(" exp ")" / builtinfcn  ,.

variable = .id *c(lda,*) subscript  ,.

subscript = "(" *c(mks) sub $( "," sub) *c(sbs) / .empty  ,.

sub = "*" *c(ni) / exp  ,.

builtinfcn = agf / sgf / fagf / biffa / cbif / obif ,.

agf = (("abs"/"floor"/"ceil"/"trunc"/"sign"/"real"/"imag"/"conj")
       "(" exp ")" /
```

```
    ("mod"/"complex"/"cplx") "(" exp "," exp ")" /
    ("max"/"min"/"fixed"/"float"/"decimal"/"dec"/"binary"/"bin"/
      "precision"/"prec"/"add"/"multiply"/"divide") "(" exp $("," exp) ")"
    )  *c(ni)   ,.

sgf = ("bit"/"char"/"substr"/"index"/"length"/"high"/"low"/"repeat"/
   "unspec"/"bool") "(" exp $("," exp) ")" *c(ni)  ,.

fagf = (("exp"/"log" ("10"/"2"/.empty)/"tanh"/"tan"/"sind"/"sinh"/
   "sin"/"cosd"/"cosh"/"cos"/"erfc"/"erf") "(" exp ")"  ) /
   ("atand"/"atan") "(" exp ("," exp/.empty) ")"  )  *c(ni)  ,.

biffa = (("sum"/"prod"/"all"/"any") "(" exp ")" /
   ("poly"/"lbound"/"hbound"/"dim") "(" exp "," exp ")" /
   "scan" "(" exp "," exp "," .string ")"  )  *c(ni)  ,.

cbif = ("onpoint"/"onloc"/"onfield"/"onchar"/"oncode")  *c(ni)  ,.

obif = "date" / "time" / ("allocation"/"point"/"count"/"string"/
   "event"/"priority") "(" .id ")" / "round" "(" exp "," .number ")" )
   *c(ni)   ,.

.end
```

13a ***** sect13a *****

This section deals with specific code of the meta compiler and interpreter.

13a1 The followiing is a list of flags and storage cells used in the meta compiler and its subroutines.

13a1a (mflag) The meta machine's only flag. This is conditionally set after executing the meta instructions id, string, number, or tst.

13a1b (ipflag) Indicates a machine language deck (in meta machine language) is to be produced while code is being compiled. Due to the lack of a card punch it is currently listed.

13a1c (irflag) If this flag is true it means that the insymbol subroutine is to return a cr after each record (record mode); if false no cr are returned (stream mode).

13a1d (icmt) If true the insymbol subroutine will delete comments from the input stream; if false no action at all will be taken. Comments are in pl/i format of /* string */. warning- the meta system is buffered ahead maxlen characters so the setting of this flag has a delayed action.

13a1e (idblk) If true blanks will not be automaticly deleted from the input stream. If false blanks will be deleted from the input stream. False is the normal mode for the meta compiler. this flag is controled by the meta system directives anchor and unanchor.

13a1f (lflag) If true insymbol will produce a listing of the input records as they are read. If false no listing is produced. This flag is controled by the meta system directives list and unlist.

13a1g (iat) This is a symbol attribute table. If the ascii code for a character is used as a subscript, say i, then iat(i+1) has the following values: letter=1, number=2, point=4, quote=8.

13a1h (ibcd) This table is used to convert from ascii to bcd.

13a1i (iascii) This table is used to convert from bcd to ascii.

13a1j (ipnt) This is the iat table value for a point.

13a1k (nmr) This is the iat table value for a number.

13a1l (iqte) This is the iat table value for a quote.

13a1m (ltr) This is the iat table value for a letter.

13a1n (i) This common location is used to pass arguments to subroutines.

13a1o (symb) This is a character which is set to the next symbol on the input stream by a call to insymbol, and is appended to the output stream by a call to outsymbol.

13a1p (w) This is the window which slides along the input stream. It is maxlen characters long.

13alq (iw) This is a pointer to the next character of the window to be looked at. That is, if the window were fresh and none of its characters had been used, iw would be one.

13alr (s) This is the star register. It acts as a temporary buffer holding the last characters deleted from the input stream by the meta instructions id, string, and number.

13als (is) This is a pointer to the s-register serving the same purpose as iw does to w.

13alt (ivstr) This cell is set to the numeric value (in binary) of the star register when a *n instruction is executed.

13alu (ic) This is a pointer to the next syllable of code to be interpurted by the meta interpreter.

13alv (isc) This is a pointer to the next syllable of code to be compiled by the compiler currently in execution.

13alw (irswd) This is a bit flag used to mark and check the latom table for reserved words.

13alx (irsf) This is set to irswd if the atom in the s register is to be flaged a reserved word when it is entered in the latom table. It is reset automaticly when the entry is made.

13aly (maxlen) This is the character length of the window and the s-register.

13alz (ss) This is string storage. All atoms are kept here in variable length format.

13alaa (latom) This is an array of length iatmx used for information about each individual atom. The low order 9 bits are the length of the atom. The high order 12 bits are the location in ss of the first character of the atom. bit 10 is the reserved word flag.

13alab (iatom) This is a pointer to one less that the first atom associated with the compiler or program currently in execution.

13alac (isatom) This is a pointer to one less than the first atom associated with the program currently being compiled.

13alad (isgprt) This is a pointer to one less that the first generated label associated with the program currently being compiled. Generated labels are produced by the compilers to transfer control in "if ... then ..." types of constructs.

13alae (iprtg) This is the generated label program reference table, which associates a generated label with an address in the source code.

13alaf (igprt) This is a pointer to one less than the first generated label associated with the program currently being executed.

13alag (isaprt) This is equivalent to isatom.

13alah (iaprt) This is equivalent to iatom.

13alai (iprta) This is the atom program reference table. It contains pointer to atoms in string storage and pointers to the source code locations associated with the atoms.

13alaj (code) This is the memory of the meta and pl/1 machines.

13alak (istk) This is a pointer to the top of the meta push down stack.

13alal (istack) This is the meta push down stack.

13alam (istkx) This is maximum depth of the meta push down stack.

13alan (itl) This is the logical number of the input unit for insymbol.

13alao (it2) This is the logical number of the output unit for outsymbol.

13alap (iorg) This is a pointer to the first syllable of code associated with the program currently in execution.

13alaq (isorg) tHis is a pointer to the first syllable of code associated with the program currently being compiled.

13alar (maxcd) This is the maximum number of syllables of code that can be handled by the system.

13alas (blank) A character whose value is an ascii blank.

13alat (point) A character whose value is an ascii point.

13alau (dollar) A character whose value is an ascii dollar sign.

13alav (quote) A character whose value is an ascii quote mark.

13alaw (slash) A character whose value is an ascii slash.

13alax (star) A character whose ascii value is an ascii astrisik.

13alay (ichr) A four character array equivalent to the cell i.

13alaz (iess) This is the last character used in ss.

13alaaa (iessx) This is the maximum number of characters in ss.

13a2 Subroutines used by the meta interpreter .

13a2a (xid) After deleting blanks on the input stream check for an identifier (a letter followed by an arbitrary number of letters or digits, with embedded points allowed). If one is found, It is checked against the reserved word list. If it is a reserved word the mflag is set false and nothing is deleted from the input stream. If it is not a reserved word, the mflag is set true and the identifier is moved from the input stream into the s-register.

13a2b (xstrng) After deleting blanks on the input stream check for a string (a quote followed by an arbitrary number of symbols followed by a quote). If one is found put it in the s-register and set mflag on. If one is not found set mflag off. Beware - since comments are deleted in the insymbol routine a comment inside a string will be deleted.

13a2c (xnum) After deleting blanks on the input stream check for a number (a digit followed by an arbitrary number of digits). If one is found put it in the s-register and set mflag on. If one is not found set mflag false.

13a2d (xstarn) Convert the s-register from ascii into a binary number and compile the last 6 bits of the number into the source code. This is used to compile numeric opcodes.

13a2e (cdn) Compile the last 6 bits of i (ichar(4)) into the source code.

13a2f (asrch) Check to see if that portion of the latom table assocaiated with the the program being compiled already contains a pointer to an atom identical to the one presently in the s-register. If so return the relative latom table number of that atom's pointer. If not, put a copy of the s-register into string storage, put a pointer to this new atom and its length in the latom table, and return the relative latom table number of the new entry. The result is returned in the cell i.

13a2g (xdg) This associates a generated label with the current source code location counter (isc). I is taken as a generated label. The iprtg entry for the program being compiled corresponding to i is set to the pointer (isc) (the next syllable position for compiled code).

13a2h (xds) This routine associates the label in the s-register with the current source code location counter (isc). First asrch is called. The iprta entry for the program being compiled corresponding to i (the relative number of the atom in the s-register) is set to the pointer (isc) (the next syllable position for compiled code). If the iprta entry was non-zero (meaning the entry was already defined) mflag is set to 0, otherwise m flag is set to 1.

13a2i (slide) The window is adjusted so that used characters are removed and blank positions are filled by calling insymbol.

13a2j (clear) This set the s-register to all blanks and resets its pointer (is) to 1.

13a2k (pack) This takes the characters in window from position 1 to position iw-1, appends them to the s-register and sets is = is+iw-1).

13a2l (delblk) This deletes blanks on the input stream by sliding window so that the first position is non-blank.

13a2m (osreg) This calls outsymbol with all the characters characters of the s-register.

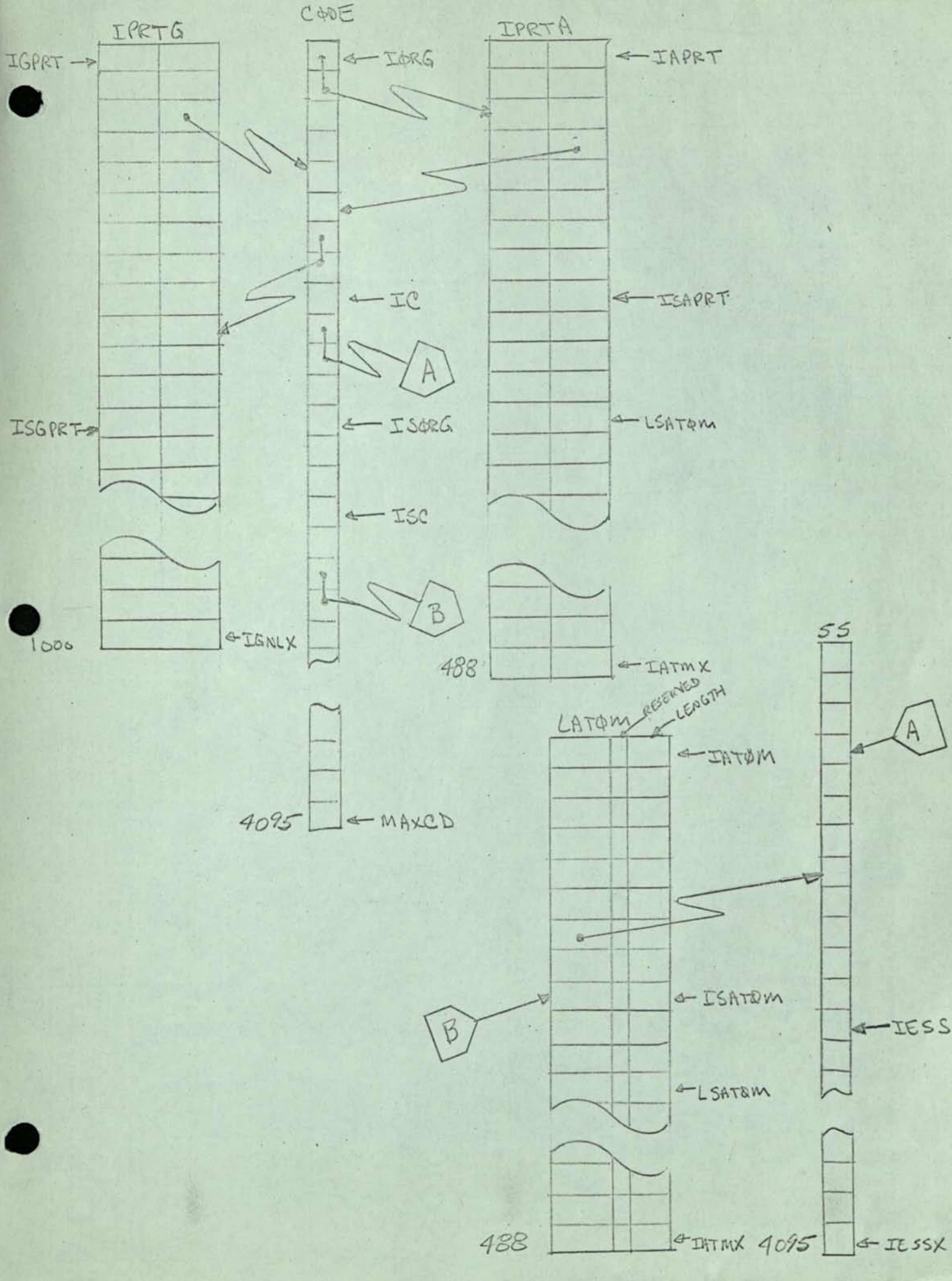
13a2n (addr) This performs an address field fetch by setting i equal to the address defined by the the next two syllables of code in the program being executed. The ic is then bumped by two so that it points to the next syllable to be interpreted.

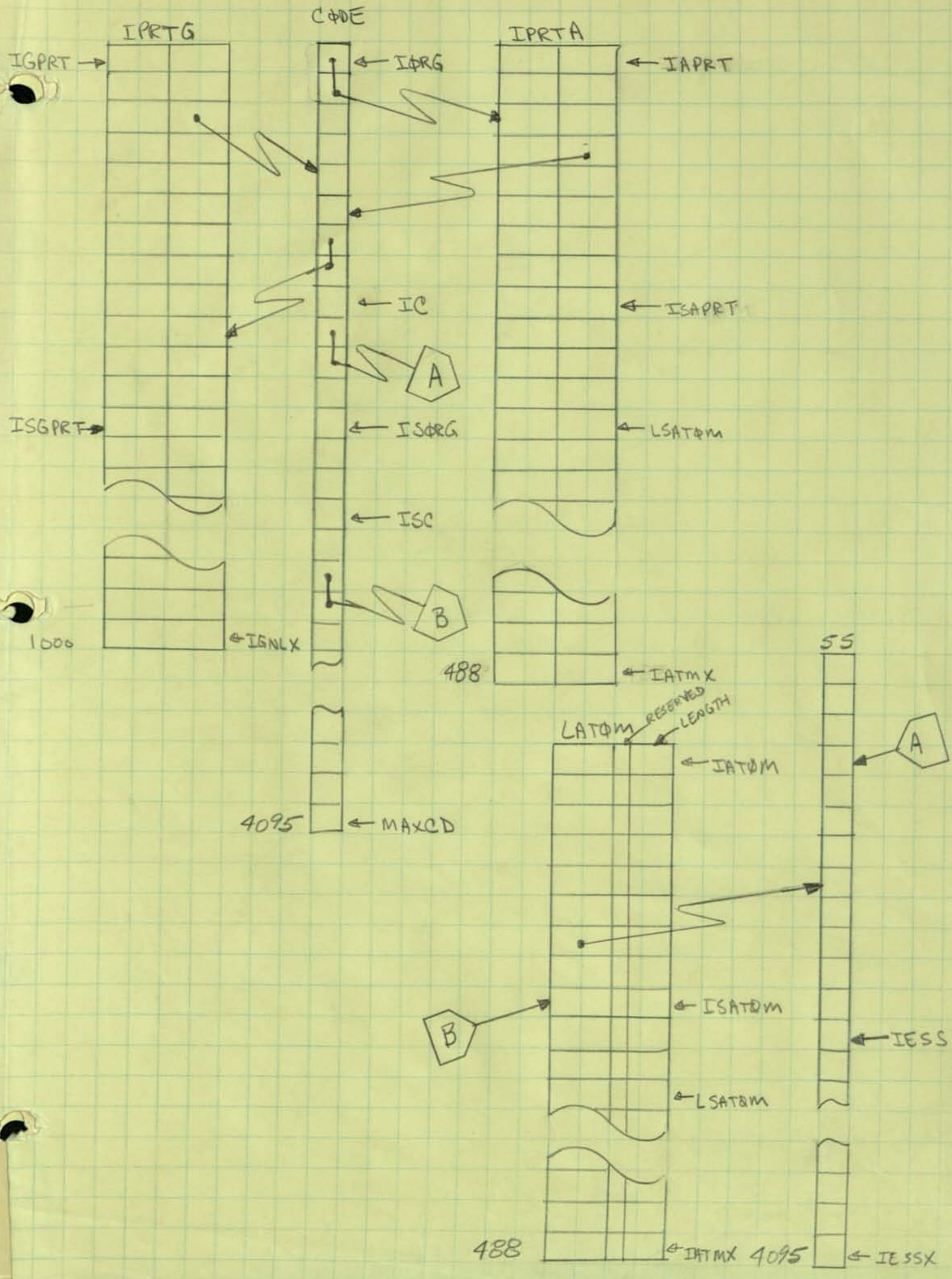
13a2o (gn) This generates a new label in the following way. if i is non-zero it just returns. If i is zero then lgnl (the last generated label) and i are both set to lgnl+1.

13a2p (push) This takes its argument and pushes it down on the general meta stack.

13a2q (pop) This pops the top of the general meta stack into its argument.

)?


IPRTG
CODE
IPRTA
IGPRT
IORG
IAPRT
IC
ISAPRT
A
ISGPRT
ISORG
LSATOM
ISC
B
1000
IGNLX
488
IATMX
SS
LATOM
RESERVED
LENGTH
IATOM
4095
MAXCD
ISATOM
IESS
LSATOM
488
IATMX
4095
IESSX


IPRTG
CØDE
IPRTA
IGPRT
IØRG
IAPRT
IC
ISAPRT
A
ISGPRT
ISØRG
LSATØM
ISC
B
1000
IGNLX
488
IATMX
SS
LATØM
RESERVED
LENGTH
IATØM
4095
MAXCD
A
ISATØM
B
IESS
LSATØM
488
IATMX
4095
IESSX

00BREAK00

This document was produced by SDC and III in performance of contract SD-97
and subcontract 65-107.

TECH MEMO

a working paper

System Development Corporation / 2500 Colorado Avenue / Santa Monica, California 90406
Information International Inc. / 200 Sixth Street / Cambridge, Massachusetts 02142

TM-2710/330/00

AUTHOR
Erwin Book
E. Book, SDC
TECHNICAL
C. Weissman, SDC
Clark Weissman
RELEASE
S L Kameny
S. L. Kameny, SDC
for D. L. Drukey, SDC

DATE
2 November 1965

AUTHOR DELIVERED

The LISP Version of the Meta Compiler

ABSTRACT

This paper describes a meta-compiler program which processes a BNF-like language and produces a LISP II intermediate language program. The program produced is a syntax translator.

The version of the compiler described here exists as a LISP 1.5 program and operates on Q-32 LISP 1.5. It will produce itself as a LISP II intermediate language program.

The work reported herein is based upon the accomplishments of Val Schorre and Lee Schmidt, of the Los Angeles Chapter of ACM, SIGPLAN Working Group I.

1. INTRODUCTION

The LISP II programming language is to be processed into LISP II intermediate language by a syntax-directed compiler. This compiler is to be produced by using a meta compiler. This document describes the Meta Compiler. The technique is based on the work done by Working Group I of the Los Angeles Chapter of ACM.

The Meta Compiler is a model of a machine with an input tape and a push down accumulator; the accumulator is referred to as the star stack and is symbolized by *. The compiler also has a true/false indicator cell called SIGNAL. The Meta Compiler translates a program written in its input language, which resembles BNF* with extensions, into a tree structure. This tree structure is a LISP II intermediate language program. The program so translated is usually referred to as a compiler. The Meta Compiler used here is itself a Q-32 LISP 1.5 program.

A meta-language program is organized into a body of rules. Each rule corresponds to a syntax equation of BNF. A Rosetta paper follows:

BNF	META	Meaning
⟨something⟩	SOMETHING	Meta-linguistic variables
A	'A'	Terminal character or string
|	/	Alternation
{ }	( )	Meta-linguistic parentheses

Writing two entities side by side (such as AB) means that an A is followed by a B. The ending symbol of a rule in LISP-META is the semicolon. BNF has no ending symbol for its syntax equations.

Identifiers are meta-linguistic variables, to wit, other definitions. They may also be the names of subroutines. If an identifier is followed by square brackets, the identifier then is the name of a routine to be executed. Its parameters are enclosed by the brackets and are separated by commas. Strings are groups of characters enclosed in primes. These correspond to terminal characters. If a prime is to be used within a string, two primes are written.

The remainder of this document is organized as follows:

A description of the various routines which are not defined in the syntax equation of the Meta Compiler, appears first. If the meta language refers to these routines by other than their names, the encoding is also shown. The next section is an English-language description of selected equations. A listing of the Meta Compiler, written in its own language, appears last.

---

*Backus Naur Form.

2. SYNTACTIC ROUTINES

These routines are principally concerned with asking questions about the characters on the input tape and what to do with them.

2.1 META LANGUAGE: 'CHARACTERS'

Routine: CMPR string (Compare)

Meaning:

When a string is written in a syntax equation it means: "If the next group of characters on the input tape matches the exhibited string,move the read past those characters and report true. Otherwise report false and do not move the read head".

2.2 META LANGUAGE: + 'CHARACTERS'

Routine: COMPS string (Compare and store)

Meaning:

This expression has the same effect as is presented in Section 2.1, except that if the answer is true the matched characters are put into the accumulator (*).

2.3 META LANGUAGE: - 'CHARACTERS'

Routine: NCOMP string (No compare)

Meaning:

If the next characters on the input tape match the exhibited string, report false. If there is no match, report true. However, do not move the read head in either event.

2.4 META LANGUAGE: ↑ 'CHARACTERS'

Routine: CMPR2 string

Meaning:

If the next characters on the input tape match the exhibited string, make a token (atom) out of the characters, push the token into the accumulator, move the read head past those characters, and report true. Otherwise, report false and do not move the read head.

2.5 META LANGUAGE: ..

Routine: MARK

Meaning:

Syntax equations which have a double period instead of an equal sign are used to collect characters and to make tokens out of them. The routine MARK is executed when these syntax equations are entered. MARK skips blanks on the input tape and stops at the first non-blank character. The routine then sets the skip blanks flag to "off" so that blanks become significant to all routines which look at characters on the input tape; that is, the routines do not bypass leading blanks while this flag is off. MARK then sets a mark in the accumulator so that all characters put into the accumulator on top of this mark will be collected as one token in a first-in first-out manner.

2.6 META LANGUAGE: ; (AT THE END OF .. EQUATION)

Routine: TOKEN

Meaning:

TOKEN collects all characters, starting with the character above the mark and going to the top of the accumulator. These characters are formed into a token. TOKEN then sets the skip blanks flag to "on," so that routines which look at the input tape characters ignore leading blanks.

2.7 META LANGUAGE: ANY

Routine: ANY

Meaning:

Put the next character on the input tape into the accumulator.

2.8 META LANGUAGE: DELETE

Routine: DELETE

Meaning:

Skip the next character on the input tape.

2.9 META LANGUAGE: $

Routine: $

Meaning:

Recognize zero or more of the following syntactic entities.

3. SEMANTIC ROUTINES

These routines are concerned with building up the tree structure which reflects the parse of the syntax.

3.1 META LANGUAGE: ⟨

Routine: FLAG

Meaning:

Set a flag in the accumulator so that a sub-tree will be formed out of the tokens and expressions collected until SEQ (⟩) is executed.

3.2 META LANGUAGE: ⟩

Routine: SEQ

Meaning:

Completes the formation of a sub-tree out of whatever has been collected since FLAG was executed.

3.3 META LANGUAGE: *n

Routine: STARn

Meaning:

STARn produces the nth element of the accumulator and removes it from the accumulator.

3.4 META LANGUAGE: +*n

Routine: STARnP

Meaning:

Copies the nth element of the accumulator onto the top of the accumulator without removing it.

3.5 META LANGUAGE: *[and $[

Routine: PUSH parameter

Meaning:

Creates a list or node out of the parameters of *[or $[and leaves it on top of the accumulator (*).

3.6 META LANGUAGE: , 'CHARACTERS'

Routine: INSERT string

Meaning:

Push the string of characters shown in the syntax equation into the accumulator.

3.7 META LANGUAGE: ↑ IDENTIFIER

Routine: LOAD x

Meaning:

Push the identifier into the stack.

3.8 META LANGUAGE: GN1 or GN2

Routine: GN1 or GN2

Meaning:

The GN1 and GN2 routines are concerned with obtaining labels for transfer points. They manipulate a stack called GEN, which is organized into pairs. The first element of each pair concerns the GN1 routine; the second element of each pair concerns the GN2 routines. If the first (second) element of the top pair is empty, a symbol is generated and put there. The first (second) element of the top pair is always produced as output.

3.9 META LANGUAGE: GEN1 or GEN2

Routine: GEN1 or GEN2

Meaning:

Push the output of GN1 (GN2) into the accumulator.

3.10 META LANGUAGE: MAKEATOM

Routine: MAKEATOM

Meaning:

Replace the string of characters on top of the accumulator by an atom with the same print name.

3.11 META LANGUAGE: MAKENUMBER

Routine: MAKENUMBER

Meaning:

Replace the string of digits on top of the accumulator with its integer value.

4. BACKUP ROUTINES

If more than one syntax equation or alternative start with the same construct, there is a possibility that an ambiguous situation will arise where backup over that first construct must occur in order to go on with the parsing. In order to accomplish the backup, the state of the machine must be saved and restored at critical places. Six routines, a stack called BACK and one called NAME are used to attempt to recover from ambiguous situations.

4.1 META LANGUAGE: RPT1

Routine: RPT1

Meaning:

This routine is invoked at the top of a loop set up by the sequence operation ($). It increments a cell called BACKUP-COUNT.

4.2 META LANGUAGE: RPT2

Routine: RPT2

Meaning:

This routine is invoked at the bottom of a loop set up by the sequence operation ($). It decrements a cell called BACKUP-COUNT. Whenever this cell is greater than zero, nothing is saved and backup does not take place.

4.3 META LANGUAGE: ENTER X

Routine: ENTER

Meaning:

This routine is used upon entering a syntax equation. The name of the syntax equation being entered is saved on a NAME list. A "blip" is pushed into the top of BACK for constructs which are collected by this syntax equation. A blip is an empty list and is used to collect information.

The GEN stack has two blips pushed into it, in case generated labels are needed by this syntax equation. Then the next entity on the INPUT tape is examined. If it is the name of the routine being entered, that name is removed from the input tape and SIGNAL is set to true.

4.4 META LANGUAGE: LEAVE

Routine: LEAVE

Meaning:

Invoked upon leaving a syntax equation. The GEN stack has its top two elements popped off. If the BACKUP-COUNT cell is zero, then SIGNAL is checked. If SIGNAL is true, then the name of the routine being left is put on top of BACK. Otherwise the top element of BACK is popped. In any event, the top element of NAME is popped.

4.5 META LANGUAGE: SAVER

Routine: SAVER

Meaning:

This routine is called at the beginning of an expression and it merely pushes a blip into BACK. The blip gets filled by all constructs which the expression collects.

4.6 META LANGUAGE: RSTOR

Routine: RSTOR

Meaning:

This routine is called at the end of expressions. If SIGNAL is true, it takes the constructs which have been collected by the expression just processed, and groups them with the constructs being collected by the next higher level expression, which may be a syntax equation. Otherwise it puts those constructs on the INPUT tape. In either event BACK is popped.

Now it is possible to examine Figure 1, which shows the Meta Compiler, and get an approximate idea of what it does. A few sample syntax equations are followed through; the remainder of the equations can be used as exercises for the interested reader. The extreme left column shows line numbers and is not part of the syntax equations.

```
4500 ID .. LET $(LET / DGT / '-' ,'.') MAKEATOM;
```

This is the definition of an identifier. The double dot shows we are forming a token and are going to skip leading blanks on the input tape. When the first non-blank character is encountered, the SKIP-BLANKS flag is set to false. We then transfer to a routine called LET which sees whether the next character on the input tape is a letter. If not, we exit ID false. If it is, we look for a sequence, which may be empty of LET letters or DGT digits or minus signs, which represent hyphens. By examining the LET and DGT equations, we see that if

```
.META
SYNTAX = '.META'
       < $(RULES / -'.FINISH' GOBBLE) '.FINISH' > COMPILE ;
RULE = ID < $[ENTER, $[QUOTE, +*1]]
        $[IF,SIGNAL ,$[GO ,GN1[]]]
       ('=' EXPR /
       '..' $[MARK] EXPR $[TOKEN] )
       ';' GEN1 $[LEAVE] >
       $[FUNCTION, *2,NIL,*[BLOCK,NIL,*1]];
EXPR = $[SAVER] SUBEXP
       $('/' $[IF, SIGNAL, $[GO, GN1[] ]]
       SUBEXP) GEN1 $[RSTOR] ;
SUBEXP = $(TESTS
       $[IF, $[NOT, SIGNAL], $[GO, GN1[] ]] /
       ACTION) GEN1;
TESTS =
       ID ('[' PARAMSQ ']' *[ *2, *1] / .EMPTY) /
       STRING1 $[CMPR, *1] /
       '+' STRING1 $[COMPS, *1] /
       '-' STRING1 $[NCOMP, *1] /
       '↑' (ID $[LOAD, $[QUOTE, *1]] /
       STRING1 $[CMPR2, *1] ) /
       '(' EXPR ')' ;
ACTION = '.EMPTY' $[SET, SIGNAL, TRUE] /
       ',' STRING1 $[INSERT, *1] /
       '$[' PARAMSQ ']' $[PUSH, *[LIST, *1]] /
       '*[' PARAMSQ1 ']' $[PUSH, *1] /
       '<' $[FLAG] / '>' $[SEQ] /
       REPEAT;
PARAM = ID('[' PARAMSQ ']' *[ *2, *1] /
       .EMPTY $[LOAD, $[QUOTE, *1]]) /
       '*1' $[STAR1] / '*2' $[STAR2] / '*3' $[STAR3] /
       '+*1' $[CAR, STAR] / '+*2' $[CADR, STAR] /
       '+*3' $[CADDR, STAR] /
       '$[' PARAMSQ ']' *[LIST, *1] /
       '*[' PARAMSQ1 ']' /
       NUM / STRING1 ;
PARAMSQ = < (PARAM $(',' PARAM) / .EMPTY) > ;
PARAMSQ1 = PARAM (',' PARAMSQ1
       $[CONS, *2, *1] / .EMPTY) ;
REPEAT = -'$[' '$' $[RPT1] GEN1 TESTS
       $[IF, SIGNAL, $[GO, GN1[] ]] $[RPT2] ;
GOBBLE = ERROR $( -';' DELETE) ';' ;
STRING1.. '''' $ (-'''' ANY /'''''' ,'''') '''' ;
ID .. LET $(LET / DGT / '-' ,'.') MAKEATOM;
NUM  .. DGT $ DGT MAKENUMBER;
LET = +'A' / +'B' / +'C' / +'D' / +'E' / +'F' / +'G' /
       +'H' / +'I' / +'J' / +'K' / +'L' / +'M' /
       +'N' / +'O' / +'P' / +'Q' / +'R' / +'S' /
       +'T' / +'U' / +'V' / +'W' / +'X' / +'Y' / +'Z' ;
DGT = +'0' / +'1' / +'2' / +'3' / +'4' /
       +'5' / +'6' / +'7' / +'8' / +'9' ;
.FINISH
```

Figure 1. A Listing of the LISP Meta Compiler

an appropriate character is recognized it is pushed into the accumulator, *, because of the + before each string. However, the ID equation indicates that the minus, if recognized, is not put into the accumulator. The comma followed by the string period, '.' shows that a period is inserted instead. After processing such a sequence of characters, a routine called MAKEATOM is called. This subroutine takes all the characters collected as an identifier token and makes an atom of them.

```
200  SYNTAX = '.META'
          < $(RULES / -'.FINISH' GOBBLE) '.FINISH' > COMPILE ;
```

This equation defines a meta-language program as starting with .META. A flag is set up so that the entire program will be collected as one list. Then we encounter zero or more of RULES. If we do not find the characters .FINISH, we call GOBBLE (4300). A cursory glance at GOBBLE shows that it goes to ERROR and then reads the input tape deleting characters until it finds a semicolon, at which time it throws that away also and returns to SYNTAX. If we do find .FINISH, we close the list which contains the parse of the program being defined. Then we go to the LISP compiler.

It is hoped that the availability of a meta compiler in LISP will make it possible to produce the syntax translator for LISP II to intermediate language more easily. A meta compiler should also facilitate the processing of modifications and improvements to the LISP II source language.

The method of production used was to take the already existing Meta Compiler on the Q-32 and make it produce LISP 1.5 output. By the well known bootstrap cannibalism it reproduced itself as a LISP 1.5 program. The syntax equations are modified to produce LISP II intermediate language and the cannibal eats again.

The definition of META and LISPX written in META

META XI page 1

```
.META (META PROGRAM)
SYNTAX = '.META' ( '(' ID ID ')'
        .[DEFINE, .[ .[ .[ *2,
        .[LAMBDA, .[X, Y], .[COMPLETE, .[INITIALIZE, X, Y],
        .[*1] ]] ]]] ] COMPILE / .EMPTY)
        $(RULE\ -'.FINISH' GOBBLE)
        '.FINISH' ;
RULE = ID ( '=' EXPR /
        '..' EXPR .[TOKEN, .[MARK], *1] ) ';'
        .[DEFINE, .[ .[ .[ *2,
        .[LAMBDA, NIL, .[LEAVE, .[ENTER], *1]] ]]] ] COMPILE ;
EXPR = SUBEXP ( EXPR1 <$ EXPR1>
        .[AND, .[OR, *4, *3, *2, -[*1]-], OK] /
        '\' .[BACKUP, *1] <SUBEXP $ EXPR2>
        .[RESTORE, .[SAVER], .[AND, .[OR, *2, -[*1]-], OK]]
        / .EMPTY) ;
EXPR1 = '/' .[NOT, OK] SUBEXP;
EXPR2 = '\' .[BACKUP, *1]  SUBEXP;
SUBEXP = TESTS ( BACKTEST < $ BACKTEST >
        .[AND, *3, *2, -[*1]-] / .EMPTY) ;
BACKTEST = TESTS  .[OR, *1, .[SETQ, OK, F]];
TESTS =
        ID (PARAMSQ .[*2, -[*1]- ] / .EMPTY .[*1] ) /
        STRING1 .[CMPR, *1] /
        '+' STRING1 .[COMPS, *1] /
        '-' STRING1 .[NCOMP, *1] /
        '↑' (ID .[LOAD, .[QUOTE, *1]] /
        STRING1 .[CMPR2, *1] ) /
        '<'  EXPR '>' .[SEQ, .[FLAGS], *1] /
        '(' EXPR ')' / '.EMPTY' ↑TRUE /
        ',' STRING1 .[INSERT, *1] /
        LISTX .[LOAD, *1] /
        '.(' <$OUTPUT> .[PROG, NIL, -[*1]-, .[RETURN, T]] ')' /
        REPEAT ;
OUTPUT = STRING1 .[PRINST, *1] /
        ID ( PARAMSQ .[PRIN0, .[*2, -[*1]-]] /
        .EMPTY .[PRIN0, *1] ) /
        STACK .[PRIN0, *1] / '/' .[TERPRI] ;
REPEAT = '$' TESTS
        .[PROG, NIL, GN1[],
        .[COND, .[*1, .[GO, GN1[] ]]],
        .[RETURN, OK]];
STACK = '*1' .[STAR1] / '*2' .[STAR2] /
        '*3' .[STAR3] / '*4' .[STAR4] /
        '++1' .[CAR, STAR] / '++2' .[CADR, STAR] /
        '++3' .[CADDR, STAR] ;
PARAMSQ = '[' < (EXPNQ $(',' EXPNQ ) /
        .EMPTY ) > ']' ;
GOBBLE = ERRORX $(-';' DELETEX) ';' ;
STRING1 .. '''' $ (-'''' ANY / '''''' , '''')
        '''' .[QUOTE, *1] ;
ID .. LET $(LET / DGT / '-' , '*') MAKEATOM ;
NUM .. DGT $ DGT MAKENUMBER ;
LET = ISIT[LETTER , T];
DGT = ISIT [DIGIT, T];
CHARACTER .. '#' ANY MAKECHR ;
```

The definition of META and LISPX written in META

page 2

```
ELEMENT = NUM / STRING1 / CHARACTER / IDENT / STACK /
        LISTX / '(' EXPNQ ')' ;
IDENT = ID (
        PARAMSQ .[*2, -[*1]- ] /
        (':[' EXPNQ ']'
        (':=' EXPNQ .[DEFLIST,
        .[LIST, .[LIST, *2, *1]], .[QUOTE, *1]] /
        .EMPTY .[GET, *1, .[QUOTE, *1]] )) /
        ':=' EXPNQ .[SETQ, *2, *1] /
        (SETQ[ID-V, ID-V] .[QUOTE, *1] / .EMPTY )) ;
LIST-SEQ = '-[' EXPNQ ']-'
        (',' LIST-SEQ .[APPEND, *2, *1] / .EMPTY)/
        EXPQ (',' LIST-SEQ / .EMPTY .[] )
        .[CONS, *2, *1] ;
LISTX = '.[' (LIST-SEQ / .EMPTY .[] ) ']' ;
EXPQ = WHERE [ FUNCTION[EXPX], T] ;
EXPNQ = WHERE [ FUNCTION[EXPX], F] ;
LISTEXP = ELEMENT $('.1' .[CAR, *1] /
        '.2' .[CADR, *1] / '.3' .[CADDR, *1] /
        '.4' .[CADDDR, *1] / ':2' .[CDR, *1] /
        ':3' .[CDDR, *1] / ':4' .[CDDDR, *1] ) ;
BASIC = LISTEXP ( '*' BASIC .[CONS, *2, *1] /
        .EMPTY);
RELATION = BASIC ('=' BASIC .[EQUAL, *2, *1] /
        '-=' BASIC .[NOT, .[EQUAL, *2, *1]] /
        .EMPTY) ;
NEGATION = '-' RELATION .[NOT, *1] /
        RELATION;
FACTOR = NEGATION ('.A.' FACTOR
        .[AND, *2, *1] / .EMPTY) ;
EXPX = FACTOR ( '.V.' EXPX
        .[OR, *2, *1] / .EMPTY) ;
LOOPST = '.LOOP' 'UNTIL' GEN1
        (EXPNQ \ ERRORX $(-'.BEGIN' DELETEX))
        '.BEGIN' .[COND, .[*1, .[GO, GN2[] ]]]
        $(ST\ -'.END' GOBBLE)
        '.END' .[GO, GN1[] ] GEN2 ;
IFST = '.IF' (EXPNQ \ ERRORX $(-'.BEGIN' DELETEX))
        '.BEGIN' ('.THEN' / .EMPTY)
        .[COND, .[ .[NOT, *1], .[GO, GN1[] ]]]
        $ ST
        ('.ELSE' .[GO, GN2[] ] GEN1
        $(ST \ -'.END' GOBBLE)
        '.END' GEN2 / '.END' GEN1) ;
PRINTST = '.PRINT' $ OUTPUT ';' ;
ST = EXPNQ ';' / LOOPST / IFST / PRINTST ;
IDSEQ = '[' < (FORMAL $( ',' FORMAL) / .EMPTY) ']' > ;
FORMAL = ID / '.LOC' ID ;
PROCEDURE = '.PROCEDURE' ID IDSEQ ';'
        ('.LOCAL' IDSEQ ';' / .EMPTY .[] )
        < $(ST \ -'.RETURN' GOBBLE)
        '.RETURN' ( '[' EXPNQ .[RETURN, *1] ']' / .EMPTY) ';' >
        .[DEFINE, .[ .[ .[ *4,
        .[LAMBDA, *3, .[PROG, *2, -[*1]- ]]] ]]]  COMPILE ;
LISP-DIVISION = '.LISPX' $ PROCEDURE '.FINISH' ;
FLUID-DECLARATION = '.DECLARE' '['
        FLUID1 $(',' FLUID1) ']' ';' ;
FLUID1 = ID .[CSET, .[*1, NIL]] COMPILE;
PROGRAM = $(SYNTAX / LISP-DIVISION /
        FLUID-DECLARATION) '.STOP' ;
.FINISH
.STOP
```

The definition of META and LISPX written in LISP1.5 (Translated from version written in META)

```
(1 DEFINE (((META (LAMBDA (X Y)
    (COMPLETE (INITIALIZE X Y) (PROGRAM)))))))
(1 DEFINE (((SYNTAX (LAMBDA NIL (LEAVE (ENTER)
      (AND (CMPR (QUOTE ('. 'M 'E 'T 'A)))
       (OR (AND (OR (AND (CMPR (QUOTE ('()))
            (OR (ID) (SETQ OK F))
            (OR (ID) (SETQ OK F))
            (OR (CMPR (QUOTE ('))))) (SETQ OK F))
           (OR (LOAD (CONS (QUOTE DEFINE)
              (CONS (CONS (CONS (CONS (STAR2)
                 (CONS (CONS (QUOTE LAMBDA)
                   (CONS (CONS (QUOTE X) (CONS (QUOTE Y) NIL))
                    (CONS (CONS (QUOTE COMPLETE)
                     (CONS (CONS (QUOTE INITIALIZE)
                       (CONS (QUOTE X) (CONS (QUOTE Y) NIL)))
                      (CONS (CONS (STAR1) NIL) NIL))) NIL))) NIL))
                NIL) NIL) NIL))) (SETQ OK F))
           (OR (COMPILE) (SETQ OK F))) (NOT OK) TRUE) OK)
        (SETQ OK F))
       (OR (PROG NIL M00001 (COND ((RESTORE (SAVER)
            (AND (OR (BACKUP (RULE))
              (AND (NCOMP (QUOTE ('. 'F 'I 'N 'I 'S 'H)))
               (OR (GOBBLE) (SETQ OK F)))) OK)) (GO M00001)))
          (RETURN OK)) (SETQ OK F))
       (OR (CMPR (QUOTE ('. 'F 'I 'N 'I 'S 'H))) (SETQ OK F)))))))))
(1 DEFINE (((RULE (LAMBDA NIL (LEAVE (ENTER)
      (AND (ID)
       (OR (AND (OR (AND (CMPR (QUOTE ('=)))
           (OR (EXPR) (SETQ OK F)))
          (NOT OK)
          (AND (CMPR (QUOTE ('. '.)))
           (OR (EXPR) (SETQ OK F))
           (OR (LOAD (CONS (QUOTE TOKEN)
              (CONS (CONS (QUOTE MARK) NIL) (CONS (STAR1) NIL))))
            (SETQ OK F)))) OK) (SETQ OK F))
       (OR (CMPR (QUOTE (' ))) (SETQ OK F))
       (OR (LOAD (CONS (QUOTE DEFINE)
          (CONS (CONS (CONS (CONS (STAR2)
              (CONS (CONS (QUOTE LAMBDA)
                (CONS (QUOTE NIL)
                 (CONS (CONS (QUOTE LEAVE)
                   (CONS (CONS (QUOTE ENTER) NIL)
                    (CONS (STAR1) NIL))) NIL))) NIL)) NIL) NIL) NIL))
        ) (SETQ OK F)) (OR (COMPILE) (SETQ OK F)))))))))
(1 DEFINE (((EXPR (LAMBDA NIL (LEAVE (ENTER)
      (AND (SUBEXP)
       (OR (AND (OR (AND (EXPR1)
           (OR (SEQ (FLAGS)
             (PROG NIL M00002 (COND ((EXPR1) (GO M00002)))
              (RETURN OK))) (SETQ OK F))
           (OR (LOAD (CONS (QUOTE AND)
             (CONS (CONS (QUOTE OR)
               (CONS (STAR4)
                (CONS (STAR3) (CONS (STAR2) (STAR1)))))
              (CONS (QUOTE OK) NIL)))) (SETQ OK F)))
          (NOT OK)
          (AND (CMPR (QUOTE (' )))
           (OR (LOAD (CONS (QUOTE BACKUP) (CONS (STAR1) NIL)))
            (SETQ OK F))
           (OR (SEQ (FLAGS)
             (AND (SUBEXP)
              (OR (PROG NIL M00003 (COND ((EXPR2) (GO M00003)))
```

```
                    (RETURN OK)) (SETQ OK F)))) (SETQ OK F))
                (OR (LOAD (CONS (QUOTE RESTORE)
                    (CONS (CONS (QUOTE SAVER) NIL)
                     (CONS (CONS (QUOTE AND)
                       (CONS (CONS (QUOTE OR) (CONS (STAR2) (STAR1)))
                        (CONS (QUOTE OK) NIL))) NIL))))) (SETQ OK F)))
               (NOT OK) TRUE) OK) (SETQ OK F)))))))))
(1 DEFINE (((EXPR1 (LAMBDA NIL (LEAVE (ENTER)
       (AND (CMPR (QUOTE ('/)))
        (OR (LOAD (CONS (QUOTE NOT) (CONS (QUOTE OK) NIL)))
         (SETQ OK F)) (OR (SUBEXP) (SETQ OK F)))))))))
(1 DEFINE (((EXPR2 (LAMBDA NIL (LEAVE (ENTER)
       (AND (CMPR (QUOTE (' )))
        (OR (LOAD (CONS (QUOTE BACKUP) (CONS (STAR1) NIL)))
         (SETQ OK F)) (OR (SUBEXP) (SETQ OK F)))))))))
(1 DEFINE (((SUBEXP (LAMBDA NIL (LEAVE (ENTER)
       (AND (TESTS)
        (OR (AND (OR (AND (BACKTEST)
            (OR (SEQ (FLAGS)
              (PROG NIL M00004 (COND ((BACKTEST) (GO M00004)))
               (RETURN OK))) (SETQ OK F))
            (OR (LOAD (CONS (QUOTE AND)
               (CONS (STAR3) (CONS (STAR2) (STAR1))))) (SETQ OK F)))
           (NOT OK) TRUE) OK) (SETQ OK F)))))))))
(1 DEFINE (((BACKTEST (LAMBDA NIL (LEAVE (ENTER)
       (AND (TESTS)
        (OR (LOAD (CONS (QUOTE OR)
           (CONS (STAR1)
            (CONS (CONS (QUOTE SETQ)
              (CONS (QUOTE OK) (CONS (QUOTE F) NIL))) NIL))))
         (SETQ OK F)))))))))
(1 DEFINE (((TESTS (LAMBDA NIL (LEAVE (ENTER)
       (AND (OR (AND (ID)
          (OR (AND (OR (AND (PARAMSQ)
              (OR (LOAD (CONS (STAR2) (STAR1))) (SETQ OK F)))
             (NOT OK)
             (AND TRUE (OR (LOAD (CONS (STAR1) NIL))
              (SETQ OK F)))) OK) (SETQ OK F)))
         (NOT OK)
         (AND (STRING1)
          (OR (LOAD (CONS (QUOTE CMPR) (CONS (STAR1) NIL)))
           (SETQ OK F)))
         (NOT OK)
         (AND (CMPR (QUOTE ('+)))
          (OR (STRING1) (SETQ OK F))
          (OR (LOAD (CONS (QUOTE CCMPS) (CONS (STAR1) NIL)))
           (SETQ OK F)))
         (NOT OK)
         (AND (CMPR (QUOTE ('-)))
          (OR (STRING1) (SETQ OK F))
          (OR (LOAD (CONS (QUOTE NCOMP) (CONS (STAR1) NIL)))
           (SETQ OK F)))
         (NOT OK)
         (AND (CMPR (QUOTE (' )))
          (OR (AND (OR (AND (ID)
              (OR (LOAD (CONS (QUOTE LOAD)
                 (CONS (CONS (QUOTE QUOTE)
                   (CONS (STAR1) NIL)) NIL))) (SETQ OK F)))
             (NOT OK)
             (AND (STRING1)
              (OR (LOAD (CONS (QUOTE CMPR2) (CONS (STAR1) NIL)))
               (SETQ OK F)))) OK) (SETQ OK F)))
```

```
          (NOT CK)
          (AND (CMPR (QUOTE (' )))
           (OR (EXPR) (SETQ OK F))
           (OR (CMPR (QUOTE (' ))) (SETQ OK F))
           (OR (LOAD (CONS (QUOTE SEQ)
              (CONS (CONS (QUOTE FLAGS) NIL) (CONS (STAR1) NIL))))
            (SETQ OK F)))
          (NOT CK)
          (AND (CMPR (QUOTE ('()))
            (OR (EXPR) (SETQ OK F))
            (OR (CMPR (QUOTE ('))))) (SETQ OK F)))
          (NOT CK)
          (AND (CMPR (QUOTE ('. 'E 'M 'P 'T 'Y)))
           (OR (LOAD (QUOTE TRUE)) (SETQ OK F)))
          (NOT CK)
          (AND (CMPR (QUOTE (',)))
           (OR (STRING1) (SETQ OK F))
           (OR (LOAD (CONS (QUOTE INSERT) (CONS (STAR1) NIL)))
            (SETQ OK F)))
          (NOT CK)
          (AND (LISTX)
           (OR (LOAD (CONS (QUOTE LOAD) (CONS (STAR1) NIL)))
            (SETQ OK F)))
          (NOT CK)
          (AND (CMPR (QUOTE ('. '()))
            (OR (SEQ (FLAGS)
              (PROG NIL M00005 (COND ((OUTPUT) (GO M00005)))
               (RETURN OK))) (SETQ OK F))
            (OR (LOAD (CONS (QUOTE PROG)
               (CONS (QUOTE NIL)
                (APPEND (STAR1)
                 (CONS (CONS (QUOTE RETURN)
                   (CONS (QUOTE T) NIL)) NIL))))) (SETQ OK F))
            (OR (CMPR (QUOTE ('))))) (SETQ OK F)))
          (NOT CK) (REPEAT)) OK)))))))
 (1 DEFINE (((OUTPUT (LAMBDA NIL (LEAVE (ENTER)
        (AND (OR (AND (STRING1)
           (OR (LOAD (CONS (QUOTE PRINST) (CONS (STAR1) NIL)))
            (SETQ OK F)))
          (NOT CK)
          (AND (ID)
           (OR (AND (OR (AND (PARAMSQ)
               (OR (LOAD (CONS (QUOTE PRINO)
                  (CONS (CONS (STAR2) (STAR1)) NIL))) (SETQ OK F)))
              (NOT CK)
              (AND TRUE (OR (LOAD (CONS (QUOTE PRINO)
                  (CONS (STAR1) NIL))) (SETQ OK F)))) OK)
            (SETQ OK F)))
          (NOT CK)
          (AND (STACK)
           (OR (LOAD (CONS (QUOTE PRINO) (CONS (STAR1) NIL)))
            (SETQ OK F)))
          (NOT CK)
          (AND (CMPR (QUOTE ('/)))
           (OR (LOAD (CONS (QUOTE TERPRI) NIL))
            (SETQ OK F)))) OK)))))))
 (1 DEFINE (((REPEAT (LAMBDA NIL (LEAVE (ENTER)
        (AND (CMPR (QUOTE ('$)))
         (OR (TESTS) (SETQ OK F))
         (OR (LOAD (CONS (QUOTE PROG)
           (CONS (QUOTE NIL)
            (CONS (GN1)
```

```
            (CONS (CONS (QUOTE COND)
              (CONS (CONS (STAR1)
                (CONS (CONS (QUOTE GO)
                  (CONS (GN1) NIL)) NIL)) NIL))
             (CONS (CONS (QUOTE RETURN)
               (CONS (QUOTE OK) NIL)) NIL)))))) (SETQ OK F)))))))))
(1 DEFINE (((STACK (LAMBDA NIL (LEAVE (ENTER)
      (AND (OR (AND (CMPR (QUOTE ('* '1)))
          (OR (LOAD (CONS (QUOTE STAR1) NIL)) (SETQ OK F)))
         (NOT OK)
         (AND (CMPR (QUOTE ('* '2)))
          (OR (LOAD (CONS (QUOTE STAR2) NIL)) (SETQ OK F)))
         (NOT OK)
         (AND (CMPR (QUOTE ('* '3)))
          (OR (LOAD (CONS (QUOTE STAR3) NIL)) (SETQ OK F)))
         (NOT OK)
         (AND (CMPR (QUOTE ('* '4)))
          (OR (LOAD (CONS (QUOTE STAR4) NIL)) (SETQ OK F)))
         (NOT OK)
         (AND (CMPR (QUOTE ('+ '* '1)))
          (OR (LOAD (CONS (QUOTE CAR) (CONS (QUOTE STAR) NIL)))
           (SETQ OK F)))
         (NOT OK)
         (AND (CMPR (QUOTE ('+ '* '2)))
          (OR (LOAD (CONS (QUOTE CADR) (CONS (QUOTE STAR) NIL)))
           (SETQ OK F)))
         (NOT OK)
         (AND (CMPR (QUOTE ('+ '* '3)))
          (OR (LOAD (CONS (QUOTE CADDR) (CONS (QUOTE STAR) NIL)))
           (SETQ OK F)))) OK)))))))
(1 DEFINE (((PARAMSQ (LAMBDA NIL (LEAVE (ENTER)
      (AND (CMPR (QUOTE (' )))
       (OR (SEQ (FLAGS)
         (AND (OR (AND (EXPNQ)
             (OR (PROG NIL M00006 (COND ((AND (CMPR (QUOTE (',)))
                  (OR (EXPNQ) (SETQ OK F))) (GO M00006)))
               (RETURN OK)) (SETQ OK F))) (NOT OK) TRUE) OK))
         (SETQ OK F)) (OR (CMPR (QUOTE (' ))) (SETQ OK F)))))))))
(1 DEFINE (((GOBBLE (LAMBDA NIL (LEAVE (ENTER)
      (AND (ERRORX)
       (OR (PROG NIL M00007 (COND ((AND (NCOMP (QUOTE (' )))
            (OR (DELETEX) (SETQ OK F))) (GO M00007))) (RETURN OK))
         (SETQ OK F)) (OR (CMPR (QUOTE (' ))) (SETQ OK F)))))))))
(1 DEFINE (((STRING1 (LAMBDA NIL (LEAVE (ENTER)
      (TOKEN (MARK)
       (AND (CMPR (QUOTE ('')))
         (OR (PROG NIL M00008 (COND ((AND (OR (AND (NCOMP (QUOTE (''))
               ) (OR (ANY) (SETQ OK F)))
              (NOT OK)
              (AND (CMPR (QUOTE ('' '')))
               (OR (INSERT (QUOTE (''))) (SETQ OK F)))) OK)
            (GO M00008))) (RETURN OK)) (SETQ OK F))
         (OR (CMPR (QUOTE (''))) (SETQ OK F))
         (OR (LOAD (CONS (QUOTE QUOTE) (CONS (STAR1) NIL)))
          (SETQ OK F))))))))))
(1 DEFINE (((ID (LAMBDA NIL (LEAVE (ENTER)
      (TOKEN (MARK)
       (AND (LET)
         (OR (PROG NIL M00009 (COND ((AND (OR (LET)
              (NOT OK)
              (DGT)
              (NOT OK)
```

```
                (AND (CMPR (QUOTE ('-)))
                 (OR (INSERT (QUOTE ('*))) (SETQ OK F)))) OK)
              (GO M00009))) (RETURN OK)) (SETQ OK F))
           (OR (MAKEATOM) (SETQ OK F))))))))))
(1 DEFINE (((NUM (LAMBDA NIL (LEAVE (ENTER)
        (TOKEN (MARK)
         (AND (DGT)
          (OR (PROG NIL M00010 (COND ((DGT) (GO M00010)))
            (RETURN OK)) (SETQ OK F))
          (OR (MAKENUMBER) (SETQ OK F))))))))))
(1 DEFINE (((LET (LAMBDA NIL (LEAVE (ENTER) (ISIT LETTER T)))))))
(1 DEFINE (((DGT (LAMBDA NIL (LEAVE (ENTER) (ISIT DIGIT T)))))))
(1 DEFINE (((CHARACTER (LAMBDA NIL (LEAVE (ENTER)
        (TOKEN (MARK)
         (AND (CMPR (QUOTE (' )))
          (OR (ANY) (SETQ OK F)) (OR (MAKECHR) (SETQ OK F))))))))))
(1 DEFINE (((ELEMENT (LAMBDA NIL (LEAVE (ENTER)
        (AND (OR (NUM)
          (NOT OK)
          (STRING1)
          (NOT OK)
          (CHARACTER)
          (NOT OK)
          (IDENT)
          (NOT OK)
          (STACK)
          (NOT OK)
          (LISTX)
          (NOT OK)
          (AND (CMPR (QUOTE ('()))
            (OR (EXPNQ) (SETQ OK F))
            (OR (CMPR (QUOTE (')))) (SETQ OK F)))) OK)))))))
(1 DEFINE (((IDENT (LAMBDA NIL (LEAVE (ENTER)
        (AND (ID)
         (OR (AND (OR (AND (PARAMSQ)
            (OR (LOAD (CONS (STAR2) (STAR1))) (SETQ OK F)))
            (NOT OK)
            (AND (CMPR (QUOTE ('  ' )))
             (OR (EXPNQ) (SETQ OK F))
             (OR (CMPR (QUOTE (' ))) (SETQ OK F))
             (OR (AND (OR (AND (CMPR (QUOTE ('  '=)))
                 (OR (EXPNQ) (SETQ OK F))
                 (OR (LOAD (CONS (QUOTE DEFLIST)
                    (CONS (CONS (QUOTE LIST)
                      (CONS (CONS (QUOTE LIST)
                        (CONS (STAR2) (CONS (STAR1) NIL))) NIL))
                     (CONS (CONS (QUOTE QUOTE)
                       (CONS (STAR1) NIL)) NIL)))) (SETQ OK F)))
                (NOT OK)
                (AND TRUE (OR (LOAD (CONS (QUOTE GET)
                    (CONS (STAR1)
                     (CONS (CONS (QUOTE QUOTE)
                       (CONS (STAR1) NIL)) NIL)))) (SETQ OK F)))) OK)
              (SETQ OK F)))
            (NOT OK)
            (AND (CMPR (QUOTE ('  '=)))
             (OR (EXPNQ) (SETQ OK F))
             (OR (LOAD (CONS (QUOTE SETQ)
               (CONS (STAR2) (CONS (STAR1) NIL)))) (SETQ OK F)))
            (NOT OK)
            (AND (OR (AND (SETQ ID*V ID*V)
              (OR (LOAD (CONS (QUOTE QUOTE) (CONS (STAR1) NIL)))
```

```
            (SETQ OK F))) (NOT OK) TRUE) OK)) OK)
        (SETQ OK F)))))))))
(1 DEFINE (((LIST*SEQ (LAMBDA NIL (LEAVE (ENTER)
      (AND (OR (AND (CMPR (QUOTE ('- ' )))
          (OR (EXPNQ) (SETQ OK F))
          (OR (CMPR (QUOTE ('  '-))) (SETQ OK F))
          (OR (AND (OR (AND (CMPR (QUOTE (',)))
              (OR (LIST*SEQ) (SETQ OK F))
              (OR (LOAD (CONS (QUOTE APPEND)
                 (CONS (STAR2) (CONS (STAR1) NIL)))) (SETQ OK F)))
            (NOT OK) TRUE) OK) (SETQ OK F)))
        (NOT OK)
        (AND (EXPQ)
         (OR (AND (OR (AND (CMPR (QUOTE (',)))
              (OR (LIST*SEQ) (SETQ OK F)))
            (NOT OK) (AND TRUE (OR (LOAD NIL) (SETQ OK F)))) OK)
          (SETQ OK F))
         (OR (LOAD (CONS (QUOTE CONS)
            (CONS (STAR2) (CONS (STAR1) NIL))))
          (SETQ OK F)))) OK)))))))
(1 DEFINE (((LISTX (LAMBDA NIL (LEAVE (ENTER)
      (AND (CMPR (QUOTE ('. ' )))
       (OR (AND (OR (LIST*SEQ)
          (NOT OK) (AND TRUE (OR (LOAD NIL) (SETQ OK F)))) OK)
        (SETQ OK F)) (OR (CMPR (QUOTE (' ))) (SETQ OK F)))))))))
(1 DEFINE (((EXPQ (LAMBDA NIL (LEAVE (ENTER)
      (WHERE (FUNCTION EXPX) T)))))))
(1 DEFINE (((EXPNQ (LAMBDA NIL (LEAVE (ENTER)
      (WHERE (FUNCTION EXPX) F)))))))
(1 DEFINE (((LISTEXP (LAMBDA NIL (LEAVE (ENTER)
      (AND (ELEMENT)
       (OR (PROG NIL M00011 (COND ((AND (OR (AND (CMPR (QUOTE ('. '1)
               ))
              (OR (LOAD (CONS (QUOTE CAR) (CONS (STAR1) NIL)))
               (SETQ OK F)))
            (NOT OK)
            (AND (CMPR (QUOTE ('. '2)))
             (OR (LOAD (CONS (QUOTE CADR) (CONS (STAR1) NIL)))
              (SETQ OK F)))
            (NOT OK)
            (AND (CMPR (QUOTE ('. '3)))
             (OR (LOAD (CONS (QUOTE CADDR) (CONS (STAR1) NIL)))
              (SETQ OK F)))
            (NOT OK)
            (AND (CMPR (QUOTE ('. '4)))
             (OR (LOAD (CONS (QUOTE CADDDR) (CONS (STAR1) NIL)))
              (SETQ OK F)))
            (NOT OK)
            (AND (CMPR (QUOTE ('  '2)))
             (OR (LOAD (CONS (QUOTE CDR) (CONS (STAR1) NIL)))
              (SETQ OK F)))
            (NOT OK)
            (AND (CMPR (QUOTE ('  '3)))
             (OR (LOAD (CONS (QUOTE CDDR) (CONS (STAR1) NIL)))
              (SETQ OK F)))
            (NOT OK)
            (AND (CMPR (QUOTE ('  '4)))
             (OR (LOAD (CONS (QUOTE CDDDR) (CONS (STAR1) NIL)))
              (SETQ OK F)))) OK) (GO M00011))) (RETURN OK))
        (SETQ OK F))))))))))
(1 DEFINE (((BASIC (LAMBDA NIL (LEAVE (ENTER)
      (AND (LISTEXP)
```

```
            (OR (AND (CR (AND (CMPR (QUOTE ('*)))
                (OR (BASIC) (SETQ OK F))
                (OR (LCAD (CONS (QUOTE CONS)
                   (CONS (STAR2) (CONS (STAR1) NIL)))) (SETQ OK F)))
              (NOT OK) TRUE) OK) (SETQ OK F)))))))))
(1 DEFINE (((RELATION (LAMBDA NIL (LEAVE (ENTER)
         (AND (BASIC)
          (OR (AND (CR (AND (CMPR (QUOTE ('=)))
                (OR (BASIC) (SETQ OK F))
                (OR (LCAD (CONS (QUOTE EQUAL)
                   (CONS (STAR2) (CONS (STAR1) NIL)))) (SETQ OK F)))
              (NOT OK)
              (AND (CMPR (QUOTE ('- '=)))
                (OR (BASIC) (SETQ OK F))
                (OR (LCAD (CONS (QUOTE NOT)
                   (CONS (CONS (QUOTE EQUAL)
                     (CONS (STAR2) (CONS (STAR1) NIL))) NIL)))
               (SETQ OK F))) (NOT OK) TRUE) OK) (SETQ OK F)))))))))
(1 DEFINE (((NEGATION (LAMBDA NIL (LEAVE (ENTER)
         (AND (OR (AND (CMPR (QUOTE ('-)))
            (OR (RELATION) (SETQ OK F))
            (OR (LOAC (CONS (QUOTE NOT) (CONS (STAR1) NIL)))
             (SETQ OK F))) (NOT OK) (RELATION)) OK)))))))
(1 DEFINE (((FACTOR (LAMBDA NIL (LEAVE (ENTER)
         (AND (NEGATION)
          (OR (AND (CR (AND (CMPR (QUOTE ('. 'A '.)))
                (OR (FACTOR) (SETQ OK F))
                (OR (LCAD (CONS (QUOTE AND)
                   (CONS (STAR2) (CONS (STAR1) NIL)))) (SETQ OK F)))
              (NOT OK) TRUE) OK) (SETQ OK F)))))))))
(1 DEFINE (((EXPX (LAMBDA NIL (LEAVE (ENTER)
         (AND (FACTOR)
          (OR (AND (CR (AND (CMPR (QUOTE ('. 'V '.)))
                (OR (EXPX) (SETQ OK F))
                (OR (LCAD (CONS (QUOTE OR)
                   (CONS (STAR2) (CONS (STAR1) NIL)))) (SETQ OK F)))
              (NOT OK) TRUE) OK) (SETQ OK F)))))))))
(1 DEFINE (((LOOPST (LAMBDA NIL (LEAVE (ENTER)
         (AND (CMPR (QUOTE ('. 'L 'O 'O 'P)))
          (OR (CMPR (QUOTE ('U 'N 'T 'I 'L))) (SETQ OK F))
          (OR (GEN1) (SETQ OK F))
          (OR (RESTORE (SAVER)
            (AND (OR (BACKUP (EXPNQ))
              (AND (ERRORX)
               (OR (PROG NIL M00012 (COND ((AND (NCOMP (QUOTE ('. 'B 'E
                    'G 'I 'N))) (OR (DELETEX) (SETQ OK F)))
                  (GO M00012))) (RETURN OK)) (SETQ OK F)))) OK))
           (SETQ OK F))
          (OR (CMPR (QUOTE ('. 'B 'E 'G 'I 'N))) (SETQ OK F))
          (OR (LCAD (CONS (QUOTE COND)
             (CONS (CONS (STAR1)
               (CONS (CONS (QUOTE GO) (CONS (GN2) NIL)) NIL)) NIL)))
           (SETQ OK F))
          (OR (PROG NIL M00013 (COND ((RESTORE (SAVER)
               (AND (OR (BACKUP (ST))
                 (AND (NCOMP (QUOTE ('. 'E 'N 'D)))
                  (OR (GOBBLE) (SETQ OK F)))) OK)) (GO M00013)))
            (RETURN OK)) (SETQ OK F))
          (OR (CMPR (QUOTE ('. 'E 'N 'D))) (SETQ OK F))
          (OR (LCAD (CONS (QUOTE GO) (CONS (GN1) NIL))) (SETQ OK F))
          (OR (GEN2) (SETQ OK F)))))))))
(1 DEFINE (((IFST (LAMBDA NIL (LEAVE (ENTER)
```

```
(AND (CMPR (QUOTE ('. 'I 'F)))
 (OR (RESTORE (SAVER)
  (AND (OR (BACKUP (EXPNQ))
   (AND (ERRORX)
    (CR (PROG NIL M00014 (COND ((AND (NCOMP (QUOTE ('. 'B 'E
         'G 'I 'N))) (OR (DELETEX) (SETQ OK F)))
       (GO M00014))) (RETURN OK)) (SETQ OK F)))) OK))
 (SETQ OK F))
 (OR (CMPR (QUOTE ('. 'B 'E 'G 'I 'N))) (SETQ OK F))
 (OR (AND (CR (CMPR (QUOTE ('. 'T 'H 'E 'N)))
   (NOT OK) TRUE) OK) (SETQ OK F))
 (OR (LCAD (CONS (QUOTE COND)
   (CONS (CONS (CONS (QUOTE NOT) (CONS (START) NIL))
    (CONS (CONS (QUOTE GO) (CONS (GN1) NIL)) NIL)) NIL)))
  (SETQ OK F))
 (OR (PROG NIL M00015 (COND ((ST) (GO M00015))) (RETURN OK))
  (SETQ OK F))
 (OR (AND (CR (AND (CMPR (QUOTE ('. 'E 'L 'S 'E)))
    (OR (LCAD (CONS (QUOTE GO) (CONS (GN2) NIL)))
     (SETQ OK F))
    (OR (GEN1) (SETQ OK F))
    (OR (PROG NIL M00016 (COND ((RESTORE (SAVER)
         (AND (OR (BACKUP (ST))
          (AND (NCOMP (QUOTE ('. 'E 'N 'D)))
           (OR (GOBBLE) (SETQ OK F)))) OK)) (GO M00016)))
      (RETURN OK)) (SETQ OK F))
    (OR (CMPR (QUOTE ('. 'E 'N 'D))) (SETQ OK F))
    (OR (GEN2) (SETQ OK F)))
   (NOT OK)
   (AND (CMPR (QUOTE ('. 'E 'N 'D)))
    (OR (GEN1) (SETQ OK F)))) OK) (SETQ OK F))))))))
(1 DEFINE (((PRINTST (LAMBDA NIL (LEAVE (ENTER)
      (AND (CMPR (QUOTE ('. 'P 'R 'I 'N 'T)))
       (OR (PROG NIL M00017 (COND ((OUTPUT) (GO M00017)))
         (RETURN OK)) (SETQ OK F))
       (OR (CMPR (QUOTE (' ))) (SETQ OK F)))))))))
(1 DEFINE (((ST (LAMBDA NIL (LEAVE (ENTER)
      (AND (OR (AND (EXPNQ) (OR (CMPR (QUOTE (' ))) (SETQ OK F)))
        (NOT OK)
        (LOOPST) (NOT OK) (IFST) (NOT OK) (PRINTST)) OK)))))))
(1 DEFINE (((IDSEQ (LAMBDA NIL (LEAVE (ENTER)
      (AND (CMPR (QUOTE (' )))
       (OR (SEQ (FLAGS)
         (AND (AND (OR (AND (FORMAL)
             (OR (PROG NIL M00018 (COND ((AND (CMPR (QUOTE (',)))
                 (OR (FORMAL) (SETQ OK F))) (GO M00018)))
                (RETURN OK)) (SETQ OK F))) (NOT OK) TRUE) OK)
          (OR (CMPR (QUOTE (' ))) (SETQ OK F)))) (SETQ OK F)))))))))
(1 DEFINE (((FORMAL (LAMBDA NIL (LEAVE (ENTER)
      (AND (OR (ID)
        (NOT OK)
        (AND (CMPR (QUOTE ('. 'L 'O 'C)))
         (OR (ID) (SETQ OK F)))) OK)))))))
(1 DEFINE (((PROCEDURE (LAMBDA NIL (LEAVE (ENTER)
      (AND (CMPR (QUOTE ('. 'P 'R 'O 'C 'E 'D 'U 'R 'E)))
       (OR (ID) (SETQ OK F))
       (OR (IDSEQ) (SETQ OK F))
       (OR (CMPR (QUOTE (' ))) (SETQ OK F))
       (OR (AND (CR (AND (CMPR (QUOTE ('. 'L 'O 'C 'A 'L)))
           (OR (IDSEQ) (SETQ OK F))
           (OR (CMPR (QUOTE (' ))) (SETQ OK F)))
          (NOT OK) (AND TRUE (OR (LCAD NIL) (SETQ OK F)))) OK)
```

```
        (SETQ OK F))
       (OR (SEQ (FLAGS)
         (AND (PRCG NIL M00019 (COND ((RESTORE (SAVER)
              (ANC (OR (BACKUP (ST))
                (AND (NCOMP (QUOTE ('. 'R 'E 'T 'U 'R 'N)))
                 (OR (GOBBLE) (SETQ OK F)))) OK)) (GO M00019)))
          (RETURN OK))
         (OR (CMPR (QUOTE ('. 'R 'E 'T 'U 'R 'N))) (SETQ OK F))
         (OR (ANC (OR (AND (CMPR (QUOTE (' )))
              (OR (EXPNQ) (SETQ OK F))
              (OR (LOAD (CONS (QUOTE RETURN) (CONS (STAR1) NIL)))
               (SETQ OK F)) (OR (CMPR (QUOTE (' ))) (SETQ OK F)))
             (NOT OK) TRUE) OK) (SETQ OK F))
         (OR (CMPR (QUOTE (' ))) (SETQ OK F)))) (SETQ OK F))
       (OR (LOAD (CONS (QUOTE DEFINE)
         (CONS (CONS (CONS (CONS (STAR4)
              (CONS (CONS (QUOTE LAMBDA)
                (CONS (STAR3)
                 (CONS (CONS (QUOTE PROG)
                   (CONS (STAR2)
                    (STAR1))) NIL))) NIL)) NIL) NIL) NIL)))
        (SETQ OK F)) (OR (COMPILE) (SETQ OK F))))))))
(1 DEFINE (((LISP*DIVISION (LAMBDA NIL (LEAVE (ENTER)
      (AND (CMPR (QUOTE ('. 'L 'I 'S 'P 'X)))
       (OR (PROG NIL M00020 (COND ((PROCEDURE) (GO M00020)))
         (RETURN OK)) (SETQ OK F))
       (OR (CMPR (QUOTE ('. 'F 'I 'N 'I 'S 'H))) (SETQ OK F)))))))))
(1 DEFINE (((FLUID*DECLARATION (LAMBDA NIL (LEAVE (ENTER)
      (AND (CMPR (QUOTE ('. 'D 'E 'C 'L 'A 'R 'E)))
       (OR (CMPR (QUOTE (' ))) (SETQ OK F))
       (OR (FLUID1) (SETQ OK F))
       (OR (PROG NIL M00021 (COND ((AND (CMPR (QUOTE (',)))
            (OR (FLUID1) (SETQ OK F))) (GO M00021))) (RETURN OK))
        (SETQ OK F))
       (OR (CMPR (QUOTE (' ))) (SETQ OK F))
       (OR (CMPR (QUOTE (' ))) (SETQ OK F)))))))))
(1 DEFINE (((FLUID1 (LAMBDA NIL (LEAVE (ENTER)
      (AND (ID)
       (OR (LOAD (CONS (QUOTE CSET)
          (CONS (CONS (STAR1) (CONS (QUOTE NIL) NIL)) NIL)))
        (SETQ OK F)) (OR (COMPILE) (SETQ OK F)))))))))
(1 DEFINE (((PROGRAM (LAMBDA NIL (LEAVE (ENTER)
      (ANC (PROG NIL M00022 (COND ((AND (OR (SYNTAX)
            (NOT OK)
            (LISP*DIVISION) (NOT OK) (FLUID*DECLARATION)) OK)
          (GO M00C22))) (RETURN OK))
       (OR (CMPR (QUOTE ('. 'S 'T 'O 'P))) (SETQ OK F)))))))))

        ECF CARD  ALL PROGRAM COMPATIBLE
  T             ENDINPUT
```

Library of META written in LISPX

MLIBR1

```
.DECLARE [STAR, GEN, SKIP-BLANKS, LINE,
      INPUT, COLUMN, BACK, CHR, FLAGX, LETTER, DIGIT,
      ODGT, ID-V, COUNT, MAXIMUM, OK, TRUE] ;
.LISPX
.PROCEDURE INITIALIZE [A, B];
.IF A = QUOTE[TTY]
      .BEGIN
      .THEN RDS[NIL];
      .ELSE OPEN[A, QUOTE[DISC], QUOTE[PERM] ];
      RDS[A];
      .END
.IF B = QUOTE[TTY]
      .BEGIN
      .THEN WRS[NIL];
      .ELSE  .IF  B = QUOTE[CORE]
            .BEGIN
            .THEN OPEN[ QUOTE[SDCSDC], QUOTE[DISC] ];
            WRS[QUOTE[SDCSDC] ];
            .ELSE  OPEN[ B, QUOTE[DISC] ];  WRS[B];
            .END
      .END
STAR := NIL;  GEN := NIL;  SKIP-BLANKS := T;
COLUMN := 0; LINE := READLN[]; INPUT := LINE; CHR := INPUT.1;
BACK := NIL;  COUNT := 1;  MAXIMUM := 1;  OK := T;
FLAGX := NIL; LETTER := 2; DIGIT := 12; ODGT := 8;
GENCH1[]; TRUE := T;
.RETURN;
.PROCEDURE COMPLETE[Z, A];  .LOCAL[X];
.IF -A
      .BEGIN
      .THEN  ERRORX[] ;
      .END
X := RDS[NIL];
.IF X -= QUOTE[TTY]
      .BEGIN
      .THEN  SHUT[X] ;
      .END
X := WRS[NIL];
.IF X = QUOTE[SDCSDC] .A. A
      .BEGIN
      .THEN  POSITION[X, QUOTE[WEOF]] ;
      POSITION[X, QUOTE[REWIND] ];
      .LOOP UNTIL  LOADEXP[X] = QUOTE[EOF]
            .BEGIN
            .END
      SHUT[X, QUOTE[DELETE]];
      .ELSE .IF X -= QUOTE[TTY]
            .BEGIN
            .THEN  POSITION[X, QUOTE[WEOF]];
            SHUT[X];
            .END
      .END
.RETURN[A];
```

MLIBR/

```
.PROCEDURE READLN[]; .LOCAL[X];
X := READCH[];
.IF NULL[X]
        .BEGIN
        .THEN RETURN[NIL];
        .ELSE RETURN[X*READLN[]];
        .END
.RETURN;
.PROCEDURE PRINTLN[U];  .LOCAL[X];
X := PRINTLN1[U];
.LOOP UNTIL NULL[X]
        .BEGIN
        PRINTCH[X.1];  X := X:2;
        .END
TERPRI[];
.RETURN;
.PROCEDURE NXTCHR[];
.IF NULL[INPUT:2]
        .BEGIN
        .THEN COLUMN := COUNT;  LINE := READLN[];
        RPLACD[INPUT, LINE];
        .END
INPUT := INPUT:2;  CHR := INPUT.1;
COUNT := ADD1[COUNT];  MAXIMUM := MAX[COUNT, MAXIMUM];
.RETURN;
.PROCEDURE LOAD[X];
STAR := X * STAR ;
.RETURN[T];
.PROCEDURE COMPARE[S, N];  .LOCAL[U, V, W, SIGNAL];
.IF SKIP-BLANKS
        .BEGIN
        .THEN
        .LOOP UNTIL CHR -= #
              .BEGIN
              NXTCHR[];
              .END
        .END
U := S;  V := INPUT;  W := COUNT;
.LOOP UNTIL (NULL[U] .V. U.1 -= CHR)
        .BEGIN
        U := U:2;  NXTCHR[];
        .END
SIGNAL := NULL[U];
.IF N = 3
        .BEGIN
        .THEN  SIGNAL := -SIGNAL ;  INPUT := V;
        CHR := INPUT.1;  COUNT := W;
        .ELSE
        .IF SIGNAL
              .BEGIN
              .THEN
              .IF N = 2
                    .BEGIN
                    .THEN STAR :=
                    APPEND[STAR.1, S] * STAR:2 ;
                    .END
              .ELSE  INPUT := V;  CHR := INPUT.1;  COUNT := W;
              .END
        .END
.RETURN[SIGNAL];
```

MLIB21

```
.PROCEDURE CMPR2[X];
.IF CMPR[X]
      .BEGIN
      .THEN  STAR := COMPRESS[X] * STAR ;  RETURN[T];
      .END
.RETURN[F];
.PROCEDURE CMPR[S];
.RETURN[COMPARE[S, 1]];
.PROCEDURE COMPS[S];
.RETURN[COMPARE[S, 2]];
.PROCEDURE NCOMP[S];
.RETURN[COMPARE[S, 3]];
.PROCEDURE ERRORX[];  .LOCAL[X];
X := WRS[NIL];
PRINTLN[LINE];  BLANKS[ SUB1[ DIFFER [MAXIMUM, COLUMN]]] ;
PRINTCH[#↑];  TERPRI[];
WRS[X];  OK := T;
.RETURN[T];
.PROCEDURE MARK[];
.LOOP UNTIL CHR -= #
      .BEGIN
      NXTCHR[];
      .END
SKIP-BLANKS := F;
STAR := NIL * STAR ;
.RETURN;
.PROCEDURE TOKEN[W, X];
SKIP-BLANKS := T ;
.IF -X
      .BEGIN
      .THEN STAR := STAR:2 ;
      .END
.RETURN[X];
.PROCEDURE INSERT[S];
STAR := APPEND[STAR.1, S] * STAR:2 ;
.RETURN[T];
.PROCEDURE STAR1[];  .LOCAL[X];
X := STAR.1;  STAR := STAR:2;
.RETURN[X];
.PROCEDURE STAR2[];  .LOCAL[X];
X:= STAR.2;  STAR:= STAR.1 * STAR:3;
.RETURN[X];
.PROCEDURE STAR3[];  .LOCAL[X];
X := STAR.3;  STAR:= STAR.1 * STAR.2 * STAR:4;
.RETURN[X];
.PROCEDURE STAR4[]; .LOCAL[X];
X := STAR.4; STAR := STAR.1 * STAR.2 * STAR.3 * STAR:4:2;
.RETURN[X];
.PROCEDURE FLAGS[];
FLAGX := STAR * FLAGX;  STAR := NIL;
.RETURN;
.PROCEDURE SEQ[W, X];
.IF X
      .BEGIN
      .THEN  STAR := REVERSE[STAR] * FLAGX.1 ;
      .ELSE STAR := FLAGX.1;
      .END
FLAGX := FLAGX:2 ;
.RETURN[X];
```

MLIBB/

```
.PROCEDURE GN1[];
.IF NULL[GEN.1]
      .BEGIN
      .THEN GEN := GENSYM[] * GEN:2 ;
      .END
.RETURN[GEN.1];
.PROCEDURE GN2[];
.IF NULL[GEN.2]
      .BEGIN
      .THEN GEN := GEN.1 * GENSYM[] * GEN:3 ;
      .END
.RETURN[GEN.2];
.PROCEDURE GEN1[];
STAR := GN1[] * STAR;
.RETURN[T];
.PROCEDURE GEN2[];
STAR := GN2[] * STAR;
.RETURN[T];
.PROCEDURE ANY[];
STAR := APPEND[STAR.1, .[(CHR)]] * STAR:2;
NXTCHR[];
.RETURN[T];
.PROCEDURE DELETEX[];
NXTCHR[];
.RETURN[T];
.PROCEDURE MAKEATOM[];
STAR := COMPRESS[STAR.1] * STAR:2 ;
.RETURN[T];
.PROCEDURE MAKENUMBER[];   .LOCAL[S, N];
S := STAR.1;  N := 0;
.LOOP UNTIL NULL[S]
      .BEGIN
      N := PLUS[TIMES[N, 10], CHR2OCT[S.1]];  S := S:2;
      .END
STAR := N * STAR:2;
.RETURN[T];
.PROCEDURE COMPILE[];
PRINT[1 * STAR.1]; STAR := STAR:2;
.RETURN[T];
.PROCEDURE ISIT[X, Y];   .LOCAL[SIGNAL];
SIGNAL := NOT[ZEROP[LOGAND[X, CONVERT[CHR]]]];
.IF SIGNAL
      .BEGIN
      .THEN .IF Y
            .BEGIN
            .THEN   STAR := APPEND[STAR.1, .[(CHR)]] * STAR:2 ;
            .END
      NXTCHR[];
      .END
.RETURN[SIGNAL];
.PROCEDURE MAKECHR[];
STAR := STAR.1.1 * STAR:2;
.RETURN[T];
.PROCEDURE  WHERE[X, Y];  .LOCAL[A, B];
A := ID-V;  ID-V := Y;  B := X[];  ID-V := A;
.RETURN[B];
```

ML1BR1

```
.PROCEDURE CHECK[A, B];  .LOCAL[X];
OPEN[A, QUOTE[DISC], QUOTE[PERM] ] ;
OPEN[B, QUOTE[DISC], QUOTE[PERM] ] ;
.LOOP UNTIL (X := PROG2[RDS[A],READ[]] ) = QUOTE[EOF]
      .BEGIN
      .IF X -= PROG2 [ RDS[B], READ[] ]
            .BEGIN
            .THEN  PRINT[QUOTE[BAD]];
            .END
      .IF X.2 = QUOTE[DEFINE]
            .BEGIN
            .THEN  PRINT [ X.3.1.1.1 ] ;
            .END
      .END
RDS[NIL];  SHUT[A];  SHUT[B];
.RETURN;
.PROCEDURE PRINTLN1[U];  .LOCAL[X];
.IF U = NIL
      .BEGIN
      .THEN  RETURN[NIL];
      .END
X := PRINTLN1[U:2] ;
.IF X = NIL .A. U.1 = #
      .BEGIN
      .THEN  RETURN[NIL];
      .END
.RETURN[U.1 * X] ;
.PROCEDURE SAVER[];
BACK := .[(INPUT),(STAR),(FLAGX),(COUNT)] * BACK;
.RETURN;
.PROCEDURE RESTORE[W, X];
BACK := BACK:2 ;
.RETURN[X];
.PROCEDURE BACKUP[X];  .LOCAL[A];
A := X .A. OK;
.IF -OK
      .BEGIN
      .THEN  INPUT := BACK.1.1 ;  STAR := BACK.1.2 ;
      FLAGX := BACK.1.3;  COUNT := BACK.1.4;
      CHR := INPUT.1;  OK := T;
      .END
.RETURN[A];
.PROCEDURE ENTER[];
GEN := NIL*NIL*GEN;
.RETURN;
.PROCEDURE LEAVE[X, Y];
GEN := GEN:3;
.RETURN[Y];
.PROCEDURE PRINST[X];  .LOCAL[Z];
Z := X;
.LOOP UNTIL Z = NIL
      .BEGIN
      PRINTCH[Z.1];  Z := Z:2;
      .END
.RETURN;
.FINISH
.STOP

READY
```

```
(SESAME
LAP(
   ((CHR2OCT SUBR1)(BAX($ 2)1)(O(E CHR2OCT)1)
    (LDA $A 0 370Q)(SUB 1Q4 0 100370Q)(LDM $A)
    (BSX *MKNO 2 5)(BUC 0 4))
())
LAP(
   ((CONVERT SUBR 1)(BAX($ 2)1)(O(E CONVERT)1)
    (SUB 1Q4 0 100370Q)(LDA KONVERT 17Q)(BSX *MKNO 2 5)
    (BUC 0 4)
  KONVERT(11Q)(11Q)(11Q)(11Q)(11Q)(11Q)(11Q)(11Q)
         ( 5Q)( 5Q)(41Q)( 1Q)( 1Q)(41Q)(41Q)(41Q)
         ( 1Q)( 3Q)( 3Q)( 3Q)( 3Q)( 3Q)( 3Q)( 3Q)
         ( 3Q)( 3Q)(41Q)( 1Q)( 1Q)(41Q)(41Q)(41Q)
         ( 1Q)( 3Q)( 3Q)( 3Q)( 3Q)( 3Q)( 3Q)( 3Q)
         ( 3Q)( 3Q)(41Q)( 1Q)( 1Q)(41Q)(41Q)(41Q)
         (21Q)( 1Q)( 3Q)( 3Q)( 3Q)( 3Q)( 3Q)( 3Q)
         ( 3Q)( 3Q)(41Q)( 1Q)( 1Q)(41Q)(41Q)(41Q)
)())
DEFINE(( (GENCH1(LAMBDA() (PROG()
    (*PLANT (LOGOR 201Q13 777744Q) 41242Q)
    (*PLANT 0 41251Q)))) ))
DEFINE(( (DUMP(LAMBDA(L)(MAPCAR L (FUNCTION EVAL1)))))
DEFINE((
    (MAGIC(LAMBDA(X)(PROG(W Y)
    (OPEN X (QUOTE DISC)(QUOTE PERM))
    (SETQ W (RDS X))
    A1 (COND((NOT(EQ(LOADEXP X)(QUOTE EOF)))(GO A1)))
    A2 (RDS W)(SHUT X)
    (RETURN Y ) ))) )) )

          EOF CARD  ALL PROGRAM COMPATIBLE
```

Library
Routines written directly in
Lisp 1.5

```
(1 CSET (STAR NIL))
(1 CSET (GEN NIL))
(1 CSET (SKIP*BLANKS NIL))
(1 CSET (LINE NIL))
(1 CSET (INPUT NIL))
(1 CSET (COLUMN NIL))
(1 CSET (BACK NIL))
(1 CSET (CHR NIL))
(1 CSET (FLAGX NIL))
(1 CSET (LETTER NIL))
(1 CSET (DIGIT NIL))
(1 CSET (ODGT NIL))
(1 CSET (ID*V NIL))
(1 CSET (COUNT NIL))
(1 CSET (MAXIMUM NIL))
(1 CSET (OK NIL))
(1 CSET (TRUE NIL))
(1 DEFINE (((INITIALIZE (LAMBDA (A B)
      (PROG NIL (COND ((NOT (EQUAL A (QUOTE TTY))) (GO M00001)))
       (RDS NIL)
       (GO M00002)
       M00001 (OPEN A (QUOTE DISC) (QUOTE PERM))
       (RDS A)
       M00002 (COND ((NOT (EQUAL B (QUOTE TTY))) (GO M00003)))
       (WRS NIL)
       (GO M00004)
       M00003 (COND ((NOT (EQUAL B (QUOTE CORE))) (GO M00005)))
       (OPEN (QUOTE SDCSDC) (QUOTE DISC))
       (WRS (QUOTE SDCSDC))
       (GO M00006)
       M00005 (OPEN B (QUOTE DISC))
       (WRS B)
       M00006 M00004 (SETQ STAR NIL)
       (SETQ GEN NIL)
       (SETQ SKIP*BLANKS T)
       (SETQ COLUMN 0)
       (SETQ LINE (READLN))
       (SETQ INPUT LINE)
       (SETQ CHR (CAR INPUT))
       (SETQ BACK NIL)
       (SETQ COUNT 1)
       (SETQ MAXIMUM 1)
       (SETQ OK T)
       (SETQ FLAGX NIL)
       (SETQ LETTER 2)
       (SETQ DIGIT 12) (SETQ ODGT 8) (GENCH1) (SETQ TRUE T)))))))
(1 DEFINE (((COMPLETE (LAMBDA (Z A)
      (PROG (X)
       (COND ((NOT (NOT A)) (GO M00007)))
       (ERRORX)
       M00007 (SETQ X (RDS NIL))
       (COND ((NOT (NOT (EQUAL X (QUOTE TTY)))) (GO M00008)))
       (SHUT X)
       M00008 (SETQ X (WRS NIL))
       (COND ((NOT (AND (EQUAL X (QUOTE SDCSDC)) A)) (GO M00009)))
       (POSITION X (QUOTE WEOF))
       (POSITION X (QUOTE REWIND))
       M00010 (COND ((EQUAL (LOADEXP X) (QUOTE EOF)) (GO M00011)))
       (GO M00010)
       M00011 (SHUT X (QUOTE DELETE))
       (GO M00012)
       M00009 (COND ((NOT (NOT (EQUAL X (QUOTE TTY)))) (GO M00013)))
```

Library of META in
LISP 1.5
(Translated from version
written in LISPX)

```
        (POSITION X (QUOTE WEOF))
        (SHUT X) M00013 M00012 (RETURN A)))))))
(1 DEFINE (((READLN (LAMBDA NIL (PROG (X)
        (SETQ X (READCH))
        (COND ((NOT (NULL X)) (GO M00014)))
        (RETURN NIL)
        (GO M00015) M00014 (RETURN (CONS X (READLN))) M00015))))))
(1 DEFINE (((PRINTLN (LAMBDA (U)
       (PROG (X)
        (SETQ X (PRINTLN1 U))
        M00016 (COND ((NULL X) (GO M00017)))
        (PRINTCH (CAR X))
        (SETQ X (CDR X)) (GO M00016) M00017 (TERPRI)))))))
(1 DEFINE (((NXTCHR (LAMBDA NIL (PROG NIL (COND ((NOT (NULL (CDR
            INPUT))) (GO M00018)))
        (SETQ COLUMN COUNT)
        (SETQ LINE (READLN))
        (RPLACD INPUT LINE)
        M00018 (SETQ INPUT (CDR INPUT))
        (SETQ CHR (CAR INPUT))
        (SETQ COUNT (ADD1 COUNT))
        (SETQ MAXIMUM (MAX COUNT MAXIMUM))))))))
(1 DEFINE (((LOAD (LAMBDA (X)
       (PROG NIL (SETQ STAR (CONS X STAR)) (RETURN T)))))))
(1 DEFINE (((COMPARE (LAMBDA (S N)
       (PROG (U V W SIGNAL)
        (COND ((NOT SKIP*BLANKS) (GO M00019)))
        M00020 (COND ((NOT (EQUAL CHR ' )) (GO M00021)))
        (NXTCHR)
        (GO M00020)
        M00021 M00019 (SETQ U S)
        (SETQ V INPUT)
        (SETQ W COUNT)
        M00022 (COND ((OR (NULL U) (NOT (EQUAL (CAR U) CHR)))
          (GO M00023)))
        (SETQ U (CDR U))
        (NXTCHR)
        (GO M00022)
        M00023 (SETQ SIGNAL (NULL U))
        (COND ((NOT (EQUAL N 3)) (GO M00024)))
        (SETQ SIGNAL (NOT SIGNAL))
        (SETQ INPUT V)
        (SETQ CHR (CAR INPUT))
        (SETQ COUNT W)
        (GO M00025)
        M00024 (COND ((NOT SIGNAL) (GO M00026)))
        (COND ((NOT (EQUAL N 2)) (GO M00027)))
        (SETQ STAR (CONS (APPEND (CAR STAR) S) (CDR STAR)))
        M00027 (GO M00028)
        M00026 (SETQ INPUT V)
        (SETQ CHR (CAR INPUT))
        (SETQ COUNT W) M00028 M00025 (RETURN SIGNAL)))))))
(1 DEFINE (((CMPR2 (LAMBDA (X)
       (PROG NIL (COND ((NOT (CMPR X)) (GO M00029)))
        (SETQ STAR (CONS (COMPRESS X) STAR))
        (RETURN T) M00029 (RETURN F)))))))
(1 DEFINE (((CMPR (LAMBDA (S) (PROG NIL (RETURN (COMPARE S 1))))))))
(1 DEFINE (((CCMPS (LAMBDA (S) (PROG NIL (RETURN (COMPARE S 2))))))))
(1 DEFINE (((NCOMP (LAMBDA (S) (PROG NIL (RETURN (COMPARE S 3))))))))
(1 DEFINE (((ERRORX (LAMBDA NIL (PROG (X)
        (SETQ X (WRS NIL))
        (PRINTLN LINE)
```

```
      (BLANKS (SUB1 (DIFFER MAXIMUM COLUMN)))
      (PRINICH ' ) (TERPRI) (WRS X) (SETQ OK T) (RETURN T))))))
(1 DEFINE (((MARK (LAMBDA NIL (PROG NIL M00030 (COND ((NOT (EQUAL CHR
          ' )) (GO M00031)))
      (NXTCHR)
      (GO M00030)
      M00031 (SETQ SKIP*BLANKS F) (SETQ STAR (CONS NIL STAR))))))))
(1 DEFINE (((TOKEN (LAMBDA (W X)
     (PROG NIL (SETQ SKIP*BLANKS T)
      (COND ((NOT (NOT X)) (GO M00032)))
      (SETQ STAR (CDR STAR)) M00032 (RETURN X))))))))
(1 DEFINE (((INSERT (LAMBDA (S)
     (PROG NIL (SETQ STAR (CONS (APPEND (CAR STAR) S) (CDR STAR)))
      (RETURN T)))))))
(1 DEFINE (((STAR1 (LAMBDA NIL (PROG (X)
      (SETQ X (CAR STAR)) (SETQ STAR (CDR STAR)) (RETURN X)))))))
(1 DEFINE (((STAR2 (LAMBDA NIL (PROG (X)
      (SETQ X (CADR STAR))
      (SETQ STAR (CONS (CAR STAR) (CDDR STAR))) (RETURN X)))))))
(1 DEFINE (((STAR3 (LAMBDA NIL (PROG (X)
      (SETQ X (CADDR STAR))
      (SETQ STAR (CONS (CAR STAR) (CONS (CADR STAR) (CDDDR STAR))))
      (RETURN X)))))))
(1 DEFINE (((STAR4 (LAMBDA NIL (PROG (X)
      (SETQ X (CADDDR STAR))
      (SETQ STAR (CONS (CAR STAR)
        (CONS (CADR STAR) (CONS (CADDR STAR) (CDR (CDDDR STAR))))))
      (RETURN X)))))))
(1 DEFINE (((FLAGS (LAMBDA NIL (PROG NIL (SETQ FLAGX (CONS STAR FLAGX)
      ) (SETQ STAR NIL)))))))
(1 DEFINE (((SEQ (LAMBDA (W X)
     (PROG NIL (COND ((NOT X) (GO M00033)))
      (SETQ STAR (CONS (REVERSE STAR) (CAR FLAGX)))
      (GO M00034)
      M00033 (SETQ STAR (CAR FLAGX))
      M00034 (SETQ FLAGX (CDR FLAGX)) (RETURN X)))))))
(1 DEFINE (((GN1 (LAMBDA NIL (PROG NIL (COND ((NOT (NULL (CAR GEN)))
        (GO M00035)))
      (SETQ GEN (CONS (GENSYM) (CDR GEN)))
      M00035 (RETURN (CAR GEN))))))))
(1 DEFINE (((GN2 (LAMBDA NIL (PROG NIL (COND ((NOT (NULL (CADR GEN)))
        (GO M00036)))
      (SETQ GEN (CONS (CAR GEN) (CONS (GENSYM) (CDDR GEN))))
      M00036 (RETURN (CADR GEN))))))))
(1 DEFINE (((GEN1 (LAMBDA NIL (PROG NIL (SETQ STAR (CONS (GN1) STAR))
      (RETURN T)))))))
(1 DEFINE (((GEN2 (LAMBDA NIL (PROG NIL (SETQ STAR (CONS (GN2) STAR))
      (RETURN T)))))))
(1 DEFINE (((ANY (LAMBDA NIL (PROG NIL (SETQ STAR (CONS (APPEND (CAR
         STAR) (CONS CHR NIL)) (CDR STAR))) (NXTCHR) (RETURN T)))))))
(1 DEFINE (((DELETEX (LAMBDA NIL (PROG NIL (NXTCHR) (RETURN T)))))))
(1 DEFINE (((MAKEATOM (LAMBDA NIL (PROG NIL (SETQ STAR (CONS
       (COMPRESS (CAR STAR)) (CDR STAR))) (RETURN T)))))))
(1 DEFINE (((MAKENUMBER (LAMBDA NIL (PROG (S N)
      (SETQ S (CAR STAR))
      (SETQ N 0)
      M00037 (COND ((NULL S) (GO M00038)))
      (SETQ N (PLUS (TIMES N 10) (CHR2OCT (CAR S))))
      (SETQ S (CDR S))
      (GO M00037)
      M00038 (SETQ STAR (CONS N (CDR STAR))) (RETURN T)))))))
(1 DEFINE (((COMPILE (LAMBDA NIL (PROG NIL (PRINT (CONS 1 (CAR STAR)))
```

```
        (SETQ STAR (CDR STAR)) (RETURN T)))))))
  (1 DEFINE (((ISIT (LAMBDA (X Y)
        (PROG (SIGNAL)
         (SETQ SIGNAL (NOT (ZEROP (LOGAND X (CONVERT CHR)))))
         (COND ((NOT SIGNAL) (GO M00039)))
         (COND ((NOT Y) (GO M00040)))
         (SETQ STAR (CONS (APPEND (CAR STAR) (CONS CHR NIL))
           (CDR STAR))) M00040 (NXTCHR) M00039 (RETURN SIGNAL)))))))
 (1 DEFINE (((MAKECHR (LAMBDA NIL (PROG NIL (SETQ STAR (CONS (CAR (CAR
           STAR)) (CDR STAR))) (RETURN T)))))))
 (1 DEFINE (((WHERE (LAMBDA (X Y)
        (PROG (A B)
         (SETQ A ID*V)
         (SETQ ID*V Y) (SETQ B (X)) (SETQ ID*V A) (RETURN B)))))))
 (1 DEFINE (((CHECK (LAMBDA (A B)
        (PROG (X)
         (OPEN A (QUOTE DISC) (QUOTE PERM))
         (OPEN B (QUOTE DISC) (QUOTE PERM))
         M00041 (COND ((EQUAL (SETQ X (PROG2 (RDS A) (READ)))
           (QUOTE EOF)) (GO M00042)))
         (COND ((NOT (NOT (EQUAL X (PROG2 (RDS B) (READ)))))
          (GO M00043)))
         (PRINT (QUOTE BAD))
         M00043 (COND ((NOT (EQUAL (CADR X) (QUOTE DEFINE)))
          (GO M00044)))
         (PRINT (CAR (CAR (CAR (CADDR X)))))
         M00044 (GO M00041) M00042 (RDS NIL) (SHUT A) (SHUT B)))))))
 (1 DEFINE (((PRINTLN1 (LAMBDA (U)
       (PROG (X)
        (COND ((NOT (EQUAL U NIL)) (GO M00045)))
        (RETURN NIL)
        M00045 (SETQ X (PRINTLN1 (CDR U)))
        (COND ((NOT (AND (EQUAL X NIL) (EQUAL (CAR U) '  )))
         (GO M00046)))
        (RETURN NIL) M00046 (RETURN (CONS (CAR U) X))))))))
 (1 DEFINE (((SAVER (LAMBDA NIL (PROG NIL (SETQ BACK (CONS (CONS INPUT
         (CONS STAR (CONS FLAGX (CONS COUNT NIL)))) BACK))))))))
 (1 DEFINE (((RESTORE (LAMBDA (W X)
      (PROG NIL (SETQ BACK (CDR BACK)) (RETURN X)))))))
 (1 DEFINE (((BACKUP (LAMBDA (X)
      (PROG (A)
       (SETQ A (AND X OK))
       (COND ((NOT (NOT OK)) (GO M00047)))
       (SETQ INPUT (CAR (CAR BACK)))
       (SETQ STAR (CADR (CAR BACK)))
       (SETQ FLAGX (CADDR (CAR BACK)))
       (SETQ CCUNT (CADDDR (CAR BACK)))
       (SETQ CHR (CAR INPUT)) (SETQ OK T) M00047 (RETURN A)))))))
 (1 DEFINE (((ENTER (LAMBDA NIL (PROG NIL (SETQ GEN (CONS NIL (CONS
        NIL GEN)))))))))
 (1 DEFINE (((LEAVE (LAMBDA (X Y)
     (PROG NIL (SETQ GEN (CDDR GEN)) (RETURN Y)))))))
 (1 DEFINE (((PRINST (LAMBDA (X)
     (PROG (Z)
      (SETQ Z X)
      M00048 (COND ((EQUAL Z NIL) (GO M00049)))
      (PRINTCH (CAR Z)) (SETQ Z (CDR Z)) (GO M00048) M00049))))))

        EOF CARD  ALL PROGRAM COMPATIBLE
```

This document was produced by SDC in performance of contract F 19(628)-5166 with the Electronic Systems Division, Air Force Systems Command, in performance of ARPA Order 773 for the Advanced Research Projects Agency Information Processing Techniques Office.

TECH MEMO

SDC

a working paper

System Development Corporation/2500 Colorado Ave./Santa Monica, California 90406

TM- 3086/001/00

AUTHOR E. Book *E. Book*
D. V. Schorre *Schorre*

TECHNICAL C. Weissman *C. Weissman*

RELEASE C. Weissman *C. Weissman*

for J. I. Schwartz

DATE 8/12/66 PAGE 1 OF 29 PAGES

(page 2 blank)

A Higher-Level Machine-Oriented Language as an Alternative to Assembly Language

ABSTRACT

This paper explains our concept of a higher-level machine-oriented language and illustrates it in detail with a description of MOL-32, which is such a language for the Q-32. A compiler for this language has been implemented and is being used in our research to write library routines for the META compiler; the MOL-32 compiler will not be released for general use.

1. INTRODUCTION

This document is not intended as a user's manual but rather as an explanation of a part of the work done in extending META compiler techniques. In a higher-level machine-oriented language, the operations and types of data are the same as those of the machine, but the format of the language is similar to that for a procedural compiler language, such as ALGOL or JOVIAL. Henceforth, machine-oriented language is referred to as MOL. Arithmetic calculations are written in the form of assignment statements. The flow of control is handled by Boolean expressions together with <u>if statements</u>, <u>for statements</u>, and <u>loop statements</u>. Direct code is allowed to give the user complete control over the machine.

The reason for using assembly language, as opposed to a machine-independent language, is that (1) the efficiency of the resultant program is of prime importance and (2) the program cannot be expressed naturally in a machine-independent language, to wit, recursive subroutines in JOVIAL or fixed-point arithmetic in FORTRAN.

Most of the programming which is now being done in assembly language could be done in a higher-level MOL. At present we are using MOL-32 to write library routines rather than entire programs. Since the purpose of these routines is to store and retrieve information in a manner that is efficient for the Q-32, they could not have been implemented in a machine-independent language. This means parts of the syntax of MOL-n would be changed if it were implemented for computer m.

2. DESCRIPTION OF MOL-32

A program written in MOL-32 consists of a declaration followed by a sequence of procedures and ended by the word .STOP. Blanks are ignored, except within strings. Let us get the flavor of the language by examining a sample procedure. The purpose of the procedure shown in Figure 1 is to read a line from teletype and to unpack it into an area specified as a parameter. A flow chart is given to assist in explanation. In actual practice, flow charts are unnecessary, because the flow of control is graphically expressed by conventions of indentation.

In the procedure shown in Figure 1, there are several reserved words. These words are listed below:

.LOCAL	.EXIT	.FOR	.THEN	.IF
.FROM	.END	.RETURN	.ELSE	

All reserved words of the language begin with a period so that the user does not have to worry if he is using a reserved word for one of his identifiers. This is especially important because we are continually adding new reserved words to the language.

2.1 SUBROUTINE LINKAGE

Parameters are passed to subroutines by means of a calling sequence. This means that parameters are supplied in consecutive words after the instruction that branches to the subroutine. Usually these parameters are addresses which are set up at compile time. The subroutine being called uses these addresses to obtain a value or as a location into which to store a value. Literal integers are also passed by putting them directly in the calling sequence. Another type of parameter which is often passed to a subroutine is a string. This consists of one word containing the number of characters in the string, followed by the characters of the string packed eight per word. Actually, any type of data can be put in a calling sequence so long as the routine being called has instructions to pick it up correctly.

Now look at Figure 1. The name of this procedure is TTYIN. It has one parameter, which is the address of the first word of a 72-word block into which the typed line is to be read, one character per word in the rightmost byte. A future version of MOL-32 will allow the names of formal parameters to be written within the parentheses which follow the name of the procedure being defined. The current compiler requires the user to write instructions to pick up these parameters. In Figure 1A these instructions appear in line 34.

```
TTYIN():   .LOCAL BUF(10), AREA, T,
     ITLTY :=('META  16TNSTAT00MOVE  66TELTYP  INPUT 4 COREIX1',
     #BUF);
     AREA := [.EXIT];   .EXIT := .EXIT+1;
     SPEAK('');   <LDA (ITLTY)R;   BUC 202;>
     $I := AREA;   $J := #BUF;
     AR1 := AREA + 72;   EXPLODEX();  T := 0;
     .FOR $I .TO 71:
          .IF T = 0
               .THEN .IF [AREA+$I] = 63
                    .THEN T := 1;   [AREA+$I] := ' '; .END
               .ELSE [AREA+$I] := ' '; .END .END
     .RETURN
```

Figure 1A. A procedure in MOL-32 which reads a line from the teletype and unpacks it into an area specified as a parameter.

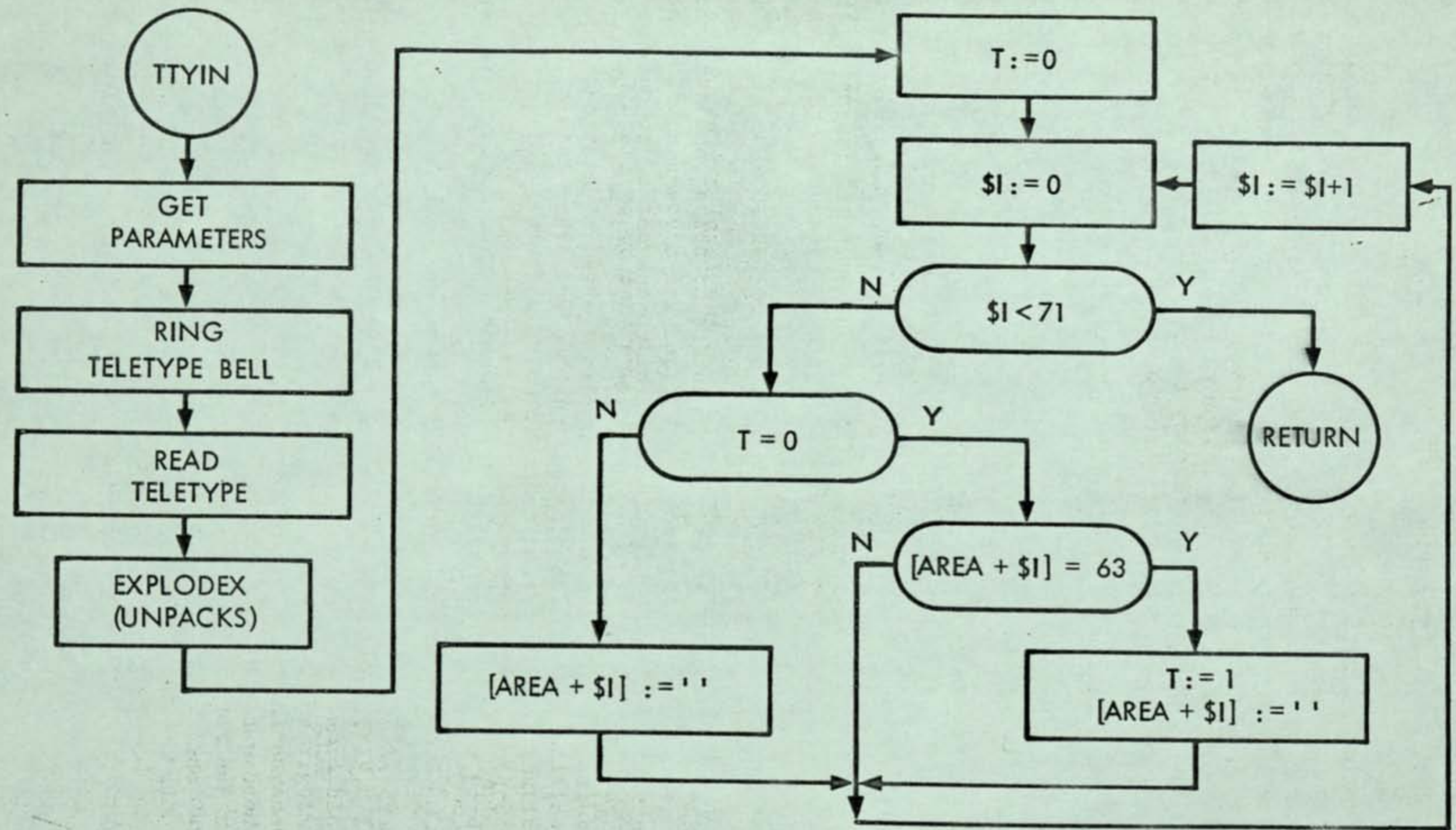


Figure 1B. A flow chart of the procedure to assist in explanation of the language. In actual practice, flow charts are unnecessary because the flow of control is graphically expressed by conventions of indentation.

2.2 THE ASSIGNMENT STATEMENT

The colon-equal (:=) is used in the assignment statement, just as in ALGOL. JOVIAL uses a single equal sign (=) for assignment, but we are using the equal sign for the relational operator which JOVIAL calls EQ. In line 34 of Figure 1A the word AREA is a local variable into which the parameter is stored. The square brackets around .EXIT indicate indirect addressing. The second assignment statement in line 34 adds one to the exit address so that control returns to the first word beyond the parameter.

In the current version of MOL-32 parentheses are not allowed within the expression on the right side of the assignment statement. All operations are performed from left to right without regard for precedence of operators. This simplification was made so that the compiler would not have to allocate temporary storage. A future version of MOL-32 will allow parentheses and follow the usual conventions for precedence of operators.

2.3 RELATIVE ADDRESSING AND INDIRECT ADDRESSING

An identifier followed by an expression enclosed in square brackets ([]) refers to a word whose address is obtained by adding the value of the expression to the address assigned to the identifier. For example, if AREA has been assigned the address 4050, then

```
AREA [25] := X;
```

means to store the contents of X into location 4075. Any legal arithmetic expression can occur between the square brackets.

An expression enclosed in square brackets but not preceded by an identifier indicates indirect addressing. For example,

```
X := [Y] ;
```

means to store the contents of the contents of location Y into location X. Any legal arithmetic expression can occur between the square brackets. Two levels of indirect addressing are shown in the example below:

```
X := [[Y]] ;
```

2.4 THE LOCAL DECLARATION

Now consider the local declaration which begins on line 31 of Figure 1A. The entries of the declaration are separated with commas (,); the declaration ends with a semicolon (;) on line 33. The first entry reserves a block of ten words, where BUF is the address of the first word. The second entry reserves one word to be called AREA, and the third reserves a word to be called T. Remember that we previously saw the identifier AREA in an assignment statement. The fourth and last entry says that ITLTY is the address of the first word of a block of preassigned data. In a declaration, the colon-equal indicates preassigned data, whereas in the body of a procedure it indicates an assignment statement. The open parenthesis after the colon-equal indicates that more than one piece of preassigned data is to be given. The first piece of data is a long string of characters enclosed by single quotes. This is stored eight characters per word, and the last word is filled in with blanks on the right. The second piece of data is an address constant, #BUF, which is a word that contains the address of BUF in its address part. A programmer familiar with the Q-32 may recognize the preassigned data as a move call for the teletype.

Line 35 contains a call to the subroutine SPEAK. One argument is given to the subroutine, and this argument is a string consisting only of the single character <u>bell</u>. You cannot see this character in the listing, but you can hear it as the listing is being typed out. Usually one can see the characters inside a string, so it is especially unfortunate that this situation occurred in a procedure chosen for an example. The purpose of the procedure SPEAK is to print the given character string on the teletype or, as in this case, to ring the teletype bell.

2.5 DIRECT CODE

Direct code can be written between angle brackets (< and >). A programmer familiar with the Q-32 may recognize the code on line 35 of Figure 1A as a call to the system to read from the teletype.

2.6 INDEX REGISTERS

Index registers 1-6 are referred to directly as $I through $N. On lines 36 and 37 of Figure 1A, index registers 1 and 2, as well as the global variable AR1, are initialized before entry to the subroutine EXPLODEX. This routine unpacks the characters just read from the teletype and stores them in the block specified as parameter for TTYIN. Then the end of message character is removed, and the rest of the block filled with blanks.

2.7 THE FOR STATEMENT

The rest of the procedure can be understood if the <u>for statement</u> and the <u>if statement</u> are explained. The <u>for statement</u> is used to specify an indexed loop. All statements within the loop are indented. In Figure 1, the scope of the <u>for statement</u> continues to the end of the procedure, so that every statement is indented up to the reserved word .RETURN. Standard conventions for indenting should be followed by all programmers using MOL-32 so that the flow of control within a program can be recognized at a glance. The compiler ingores indentation, and considers the <u>for statement</u> to be terminated by the reserved word .END. The last .END on line 42 terminates the <u>for statement</u>. The other occurrence of .END on line 42 terminates an <u>if statement</u>, which is inside the <u>for statement</u>.

The general form of the for statement is given below:

$$\text{.FOR}\left\{\begin{matrix}\$I\\ \$J\\ \vdots\\ \vdots\\ \$N\end{matrix}\right\}\left\{\begin{matrix}\text{.TO}\\ \\ \text{.FROM}\end{matrix}\right\}\quad \langle\text{expression}\rangle\quad :$$

<sequence of statements> .END

The index always ranges from 0 through the value of the expression; thus, the number of times the program goes through the loop is one greater than the value of the expression. When the value of the expression is 0, the loop is executed only once; when the value is negative, the instructions within the loop are not executed at all. Thus, index registers either start at 0 and are incremented by 1 up to the value of the expression, or start at the value of the expression and go down to 0. This is determined by the use of the words .TO and .FROM, respectively.

2.8 THE IF STATEMENT

The if statement has two basic forms, both of which are illustrated in Figure 1. The first form allows either of two sequences of statements to be executed, depending upon the value of a Boolean expression. Both sequences of statements are indented. The first sequence of statements begins with the reserved word .THEN; the second sequence begins with the reserved word .ELSE. The words .THEN and .ELSE are written at the same level of indentation, to indicate a parallel in the flow of control. The compiler ignores indentation, and considers the if statement to be terminated by the reserved word .END. The next to last .END on line 42 of Figure 1A terminates the if statement which begins on line 39. The .END on line 41 terminates another if statement which is inside this if statement.

The second form of the if statement allows for the optional execution of a sequence of statements. It is identical to the other form except that the else clause is omitted. The innermost if statement of the example, which begins on line 40, is of this form. The .END on line 41 terminates this statement. Notice that line 41 is indented five spaces from the beginning of line 40, and not five spaces from the occurrence of .IF on line 40.

The forms of the *if statement* may be summarized in meta-language as follows:

```
1.  .IF <Boolean expression>
        .THEN <sequence of statements>
        .ELSE <sequence of statements>  .END

2.  .IF <Boolean expression>
        .THEN <sequence of statements>  .END
```

2.9 BOOLEAN EXPRESSIONS

Now consider what is allowed in a Boolean expression. The relational operators are as follows:

Relation Operator	Meaning
==	equal, full word
=/=	unequal, full word
=	equal, numerical
/=	unequal, numerical
<	less than, numerical
>	greater than, numerical
<=	less than or equal, numerical
>=	greater than or equal, numerical

The reason for distinguishing between full word comparisons and numerical comparisions is that on the Q-32, +0 is different from -0.

There are two Boolean operators, .A. for *and* and .V. for *or*. Unlike arithmetic expressions, Boolean expressions may contain parentheses for grouping. The operator .A. takes precedence over .V.

Identifiers are not allowed to take on Boolean values; in other words, the operators .A. and .V. always connect relational expressions, never identifiers. Neither do we allow the Boolean operator *not*, but the effect of this operator can be obtained by using the appropriate relational operators.

2.10 THE LOOP STATEMENT

The procedure EXPLODEX, which is shown in Figure 2, contains an example of a *loop statement*, which was not illustrated in Figure 1. The general form of this statement is:

```
.LOOP WHILE <Boolean expression>  :
     <sequence of statements>  .END
```

```
EXPLODEX():
      .LOOP WHILE $I < AR1:
            [$I] := [$J].0;
            [$I+1] := [$J].1;
            [$I+2] := [$J].2;
            [$I+3] := [$J].3;
            [$I+4] := [$J].4;
            [$I+5] := [$J].5;
            [$I+6] := [$J].6;
            [$I+7] := [$J].7;
            $I := $I+8;  $J := $J+1; .END
      .RETURN
```

Figure 2. A procedure in MOL-32 to unpack characters. Before entry to this procedure, the arguments are set up in index registers 1 and 2 ($I and $J) and in the global variable AR1.

The meaning of the "loop statement" is that the sequence of statements is to be executed as long as the Boolean expression is true. If the Boolean expression is false to begin with, the sequence of statements is never executed.

2.11 REFERENCING PART OF A WORD

In procedure EXPLODE, you can see the way of referring to parts of words. For example, [$J].0 on line 178 means the left-most byte of the word whose address is in index register 2. The computer word is divided into eight character-bytes, referred to as .0, .1, .2, ..., and .7, and into four other parts, referred to as prefix, decrement, tag, and address, referred to as .P, .D, .T, and .A, respectively. These may be used on either side of the colon-equal (:=) as illustrated in the following example:

```
A [B.D] .P:=C.5;
```

which stores the fifth character of the word whose address is C into the prefix of the word whose address is obtained by adding the address of A to the decrement of the word whose address is B.

2.12 DECLARATION BEGINNING A PROGRAM

Most of the features of the language that are used within procedures have been illustrated. The declaration that begins the source program has the same form as the local declaration within individual procedures, except that it begins with .DECLARE instead of .LOCAL.

2.13 THE GO STATEMENT

Statement labels and go statements are included in MOL-32, although they are seldom used. The statement label consists of an identifier followed by a colon (:); the go statement consists of the reserved word .GO followed by an expression (such as a statement label) which evaluates to the address of some instruction. A future version of MOL-32 will include a case statement which will handle situations presently requiring a computed go statement. In some cases a procedure can be made more efficient by using direct go statements, but this practice is not recommended because the flow of control will not be indicated by the indentation conventions.

2.14 THE ASSEMBLER

Figure 3 shows the SCAMP-like assembly language that is produced for the procedures shown in Figures 1 and 2. Most compilers generate statement labels which the assembler puts in a symbol table along with the identifiers in the source program. Instead of generating statement labels, the META SCAMP compiler turns out the labels *A and *B as well as the pseudo-instructions PSHA, POPA, PSHB, and POPB, which manage two stacks at assembly time. The pseudo-instructions PSHA and POPA serve as brackets so that *A is assigned the same address within their range. Similarly, the assignment of *B is done within the range of PSHB and POPB. We have drawn lines on the listing in Figure 3 in order to clarify this bracketing convention.

```
TTYIN
        SBR
        STP,567,7            EXIT
        PSHA
        BUC                  *A
BUF
        BLK                  10
AREA
        BLK     1
T
        BLK     1
ITLTY
        ('META   16TNSTAT00MOVE   66TELTYP   INPUT 4 COREIX1')R
        (BUF)R
*A
        POPA
        LDA,567,7            EXIT,I
        STA                  AREA
        LDA,567,7            EXIT
        ADD,567,7,S          (1)R
        STA,567,7            EXIT
        BUC                  SPEAK
        (1)
        ('')
        LDA (ITLTY)R
        BUC 202
        LDA                  AREA
        STA,567,7,S          $X1
        LDA,567,7,S          (BUF)R
        STA,567,7,S          $X2
        LDA                  AREA
        ADD,567,7,S          (72)R
        STA                  AR1
        BUC                  EXPLODEX
        LDA,567,7,S          (0)R
        STA                  T
```

Figure 3. The procedures of Figures 1 and 2 are shown here in assembly language, Book's version of SCAMP.

```
        LDX,1             (0)R
        PSHA
  *A
        PSHA
        LDA               T
        SUB,567,7,S       (0)R
        BNZ               *A
        PSHB
  *B
        POPB
        PSHA
        LDA               AREA
        ADD,567,7,S       $X1
        LDA               0,A
        SUB,567,7,S       (63)R
        BNZ               *A
        PSHB
  *B
        POPB
        LDA,567,7,S       (1)R
        STA               T
        LDA               AREA
        ADD,567,7,S       $X1
        LDX,7    $A
        LDA,567,7,S       (' ')R
        STA               0,7
  *A
        POPA
        PSHB
        BUC               *B
  *A
        POPA
        LDA               AREA
        ADD,567,7,S       $X1
        LDX,7    $A
        LDA,567,7,S       (' ')R
        STA               0,7
  *B
        POPB
        BXE,71            $+3,1
        ATX,1             (1)R
        BUC               *A
        POPA
  EXIT  BUC
        RET
```

Figure 3. (Continued)

```
EXPLODEX
          SBR
          STP,567,7          EXIT
          PSHA
          PSHB
*B
          LDA,567,7,S        $X1
          SUB                AR1
          BOZ                *A
          BOP                *A
          PSHB
*B
          POPB
          LDA,7,0            ,2
          STA                ,1
          LDA,7,1            ,2
          STA                +1,1
          LDA,7,2            ,2
          STA                +2,1
          LDA,7,3            ,2
          STA                +3,1
          LDA,7,4            ,2
          STA                +4,1
          LDA,7,5            ,2
          STA                +5,1
          LDA,7,6            ,2
          STA                +6,1
          LDA,7,7            ,2
          STA                +7,1
          ATX,1              (+8)R
          ATX,2              (+1)R
          BUC                *B
*A
          POPA
          POPB
EXIT      BUC
          RET
          FIN
```

Figure 3. (Continued)

3. CONCLUSION

We have eliminated a major bottle-neck in our research by writing library routines of the META compiler in MOL-32 instead of in SCAMP assembly language. Not only are routines easier to write and check out, but easier to modify after they get cold.

In comparing the library written in MOL-32 to the library written in SCAMP, we noticed that the MOL-32 library took about 15% more space and executed about 15% slower than the SCAMP library. This seems reasonable considering the advantages gained.

The MOL-32 compiler was written in a version of META called SCAMP META. It took one month to program and only one and one-half days to check out.

The appendix contains (1) a specification of MOL-32 in SCAMP META and (2) the library of the META compiler, which serves as an example of a program written in the MOL-32.

APPENDIX A

Specification of MOL-32 in SCAMP META

```
00100-.SYNTAX  PROGRAM
00200-NUM .. DGT $ DGT ;
00210-NUM1 .. ,'+' NUM ;
00300-ID .. LET $(LET / DGT ) ;
00400-STRING = QUOTE .(,'(' .L ')' /,'(' *1 ')' /) ;
00500-DELETE = .Q(ISIT, CHAR, 0);
00600-ANY = .Q(ISIT, CHAR, 1) ;
00700-LET = .Q(ISIT, LETTER, 1) ;
00800-DGT = .Q(ISIT, DIGIT, 1) ;
00900-QUOTE .. +'''' $(-'''' ANY / +'''''') +'''' ;
01000-SCAMP1 .. ID $' ' (S1A / -'[') /
01100-      '[' $' ' ID $' '  ']' ,',I' / S1A ;
01200-INDX .. '$I' ,'$X1' / '$J' ,'$X2' / '$K' ,'$X3' /
01300-      '$L' ,'$X4' / '$M' ,'$X5' / '$N' ,'$X6' ;
01800-SCAMP2 = '.EXIT' ,'EXIT' ,',567,7' /
01900-      '[' '.EXIT' ']' ,'EXIT,I' ,',567,7' /
02000-      SCAMP1 PART / (CONST1/INDX) ,',567,7,S' ;
02100-S-NUM .. (+'+' / +'-') $' ' NUM ;
02200-CONSTANT .. S-NUM / NUM / QUOTE /
02300-      '#' ID $' ' (S-NUM / .EMPTY) ;
02400-CONST1 .. ,'(' CONSTANT ,')R' ;
02500-INDEX = '$I' ,',1' / '$J' ,',2' /
02600-      '$K' ,',3' / '$L' ,',4' /
02700-      '$M' ,',5' / '$N' ,',6' ;
02800-CONST2 .. S-NUM / '+#' ID $' '(S-NUM / .EMPTY) ;
02900-S1A = '[' $' ' (
03000-      '$I' $' ' (S-NUM / .EMPTY) ,',1' /
03100-      '$J' $' ' (S-NUM / .EMPTY) ,',2' /
03200-      '$K' $' ' (S-NUM / .EMPTY) ,',3' /
03300-      '$L' $' ' (S-NUM / .EMPTY) ,',4' /
03400-      '$M' $' ' (S-NUM / .EMPTY) ,',5' /
03500-      '$N' $' ' (S-NUM / .EMPTY) ,',6' /
03600-      S-NUM / NUM1) $' ' ']' ;
03700-PART = '.0' ,',7,0' / '.1' ,',7,1' /
03800-      '.2' ,',7,2' / '.3' ,',7,3' /
03900-      '.4' ,',7,4' / '.5' ,',7,5' /
04000-      '.6' ,',7,6' / '.7' ,',7,7' /
04100-      '.P' ,',7,0' / '.D' ,',567,3' /
04200-      -'.THEN' '.T' ,',7,4' / -'.A.' '.A' ,',567,7' /
04300-      .EMPTY ,' ' ;
```

APPENDIX A (Cont'd)

```
EXP = (SCAMP2 (-'[') .(,'LDA' *1, *1/) /
      (ID '['/'[' ,'0') EXP ']' PART .(,'LDA' *1, *1 ',A'/) ) $(
      '+' SCAMP2 .(,'ADD' *1, *1/) /
      '-' SCAMP2 .(,'SUB' *1, *1/) /
      '*' SCAMP2 .(,'MUL' *1, *1/ ,'STB', '$A'/) /
      '/' SCAMP2 .(,'LDB', '$A' / ,'SFC', '(47)R' /
           ,'DVD' *1, *1/) /
      '\' SCAMP2 .(,'LDB', '$A' / ,'SFC', '(47)R' /
           ,'DVD' *1, *1 / ,'STB' ,'$A' /) /
      '↑' CONST1 .(,'CYA', *1/) ) ;
ASSIGNST = SCAMP2 ':=' EXP ';' .(,'STA' *1, *1/) /
      (ID '[' EXP ']' / '[' EXP ']' ,'0') PART
      ':=' .(,'LDX,7  $A' /)
      EXP ';' .(,'STA' *1, *1 ',7' /) ;
INDEXST = '$I' ':=' '$I' CONST1 ';' .(,'ATX,1', *1  /)/
      '$J' ':=' '$J' CONST1 ';' .(,'ATX,2', *1  /)/
      '$K' ':=' '$K' CONST1 ';' .(,'ATX,3', *1  /)/
      '$L' ':=' '$L' CONST1 ';' .(,'ATX,4', *1  /) /
      '$M' ':=' '$M' CONST1 ';' .(,'ATX,5', *1  /)/
      '$N' ':=' '$N' CONST1 ';' .(,'ATX,6', *1  /);
RELATION = EXP (
      '==' CONSTANT .(,'BXE,' *1, '$+2,A' / ,'BUC', *A /) /
      '==' SCAMP2 .(,'CML'*1, *1/ ,'BUC', *A/)/
      '=/=' CONSTANT .(,'BXE,' *1, *A ',A' /) /
      '=/=' SCAMP2 .(,'CML' *1, *1/ ,'BUC  $+2' / ,'BUC', *A/) /
      '=' SCAMP2 .(,'SUB' *1, *1/ ,'BNZ', *A/) /
      '/=' SCAMP2 .(,'SUB' *1, *1 / ,'BOZ', *A/) /
      '<' SCAMP2 .(, 'SUB' *1, *1/ ,'BOZ', *A / ,'BOP', *A/) /
      '>' SCAMP2 .(,'SUB' *1, *1 / ,'BOZ', *A / ,'BNP' ,*A/ ) /
      '<=' SCAMP2 .(,'SUB' *1, *1 / ,'BOP', *A/) /
      '>=' SCAMP2 .(,'SUB' *1, *1 / ,'BOZ $+2'/ ,'BNP', *A/) );
BASIC = RELATION / '(' BOOLEAN ')' ;
FACTOR = BASIC $('.A.' BASIC) ;
BOOLEAN = FACTOR .(+B)
      $('.V.' .(,'BUC', *B/ *A/ -A+A) FACTOR)
      .(*B / -B) ;
```

APPENDIX A (Cont'd)

```
FORST = '.FOR' INDEX '.FROM' CONST1 ':'
        .(,'LDX' +*2, *1/ +A*A/) $ ST '.END'
        .(,'BPX,1' ,*A *1/-A)/
        '.FOR' INDEX '.FROM' EXP ':'
        .(+B, 'BXL,-0' ,*B ',A' / ,'SUB   (0)R'/ ,'LDX' +*1, '$A'/
        +A*A /) $ ST '.END' .(,'BPX,1', *A *1/ -A*B/ -B)/
        '.FOR' INDEX '.TO' CONSTANT  ':'
        .(,'LDX' +*2, '(0)R' / +A*A/) $ ST '.END'
        .(, 'BXE,' *1, '$+3' +*1 / , 'ATX' *1, '(1)R' /
        ,'BUC', *A/ -A)/
        '.FOR'INDEX '.TO' EXP ':'
        .(+B, 'BXL,-0', *B ',A' / ,'SUB   (0)R' /
        ,'STA,567,3', *B '-3' / ,'LDX' +*1, '(-0)R' / +A *A/)
        $ ST '.END' .(,'BXE,0', *B +*1 / ,'ATX' *1, '(1)R' /
        ,'BUC' , *A / -A *B / -B) ;
LOOPST = '.LOOP' 'WHILE' .(+A +B *B/) BOOLEAN ':'
        $ ST '.END' .(,'BUC', *B/ *A/ -A -B);
IFST = '.IF' .(+A) BOOLEAN '.THEN' $ ST
        ('.ELSE' .(+B,'BUC',*B / *A / -A)
        $ST '.END' .(*B / -B) /
        '.END' .(*A / -A) ) ;
ERRORST = '.ERROR' 'UNLESS' .(+A) BOOLEAN ':'
        .(+B, 'BUC', *B / *A, 'BUC SPEAK'/)
        STRING ';' .(-A *B / -B) ;
CALLST = ID '(' .(,'BUC', *1 /)
        (ARG $(',' ARG) / .EMPTY) ')' ';' ;
ARG = STRING / CONSTANT .(,'(' *1 ')'/) ;
PUSHST = '.PUSH' INDEX ',' NUM ';'
        .(,'ATX' +*2, '(-' *1 ')R' / ,'BMX,0   PUSHER' *1/) ;
POPST = '.POP' INDEX ',' NUM ';'
        .(,'ATX' +*2, '(' *1 ')R'/) .F(POP) ;
GOST = '.GO' SCAMP2 .F(POP) .(,'BUC', *1/) ';' /
        '.GO' EXP .(,'BUC   0,A'/) ';' ;
LABEL = ID (-':=') ':' .(*1/) ;
MACHINEST .. $(-';' ANY ) ';' ;
MACHINE-CODE = '<' $(LABEL / -'>'
        MACHINEST .(,*1/)) '>' ;
ST = FORST / LOOPST / IFST / PUSHST / POPST /
        GOST / MACHINE-CODE / ERRORST /
        LABEL / INDEXST / CALLST / ASSIGNST ;
```

APPENDIX A (Cont'd)

```
12000-DATA = ID .(*1/) ('(' CONSTANT ')' .(,'BLK', *1/) /
12100-      ':=' (DATA1 /'(' DATA1 $(';' DATA1)')') /
12200-      .EMPTY .(,'BLK   1'/)) /
12300-      '#' ID '=' .(*1/) CONSTANT .(,'EQU ' *1/) ;
12400-DATA1 = CONST1 .(,*1/);
12500-DECLARATION = '.DECLARE' DATA $(',' DATA) ';' ;
12600-PROCEDURE = ID '():' .(*1/ , 'SBR' / ,'STP,567,7', 'EXIT'/)
12700-      ('.LOCAL' .(+A,'BUC',*A/) DATA $(',' DATA) ';'
12710-      .(*A /-A) / .EMPTY)
12800-      $(ST / -'.RETURN' ERROR
12900-      $(-'.RETURN' DELETE))
13000-      '.RETURN' ('[' EXP ']' / .EMPTY)
13010-      .('EXIT' ,'BUC' / ,'RET' /) ;
13100-PROGRAM = ('.PRIMITIVE' ID .(,'BUC', *1/,'BUC 195'/) /
13200-      .EMPTY) $ (PROCEDURE/DECLARATION) '.STOP' .(,'FIN'/) ;
13300-.END
```

APPENDIX B

Library of the META Compiler Written in MOL-32

```
.DECLARE  #STARL = 1000,  STAR(#STARL),
        #STACKL = 400,  STACK(#STACKL),
        #INL1 = 73, #INL2 = 146, INBUF(#INL2),
        #INPLACE = #INBUF + 73, INCOUNT, INX, MAXIMUM,
        #OUTL1 = 73, #OUTL2 = 146, OUTBUF(#OUTL2),
        #OUTPLACE = #OUTBUF + 73, OUTCOUNT, OUTX,
        SIGNAL, TOKENDEPTH, TK(#INL1+1),
        XPLBUF(#INL1), ERBUF(#INL1),
        AR1, AR2, AR3, VL1, VL2, VL3, T1, T2, T3,
        #ATOM = 1, P1INP(512), P1OUT(512),
        BLANKS := '          ',
        IUNIT, OUNIT, TTY := 'TTY      ',
        FIELD, COUNT1, COUNT2,
        SOURCE:=('META  18TNSTAT00MOVE  66SOURCE  INPUT 4 NUMWDS1',
                512,'DISCIX1',0),
        PRGRM:=('META  18TNSTAT00MOVE  66PRGRM   OUTPUT4 NUMWDS1'
                ,512,'DISCIX1',0),
        PERM  :=('META  17TNSTAT00INNAME66          INSERT66PRGRM  ',
                'NUMWDS1', 0),
        OLINE, ILINE, LETTER := 2, DIGIT := 12, CHAR := 1;
SPEAK():  .LOCAL T, BUF(10), LFCR := 03277000000000000,
        MESG :=('META  16TNSTAT00MOVE  66TELTYP  OUTPUT4 COREIX1',
        #BUF);
        T := [.EXIT]-1/8+1;
        .FOR $I .FROM T-1:
             BUF[$I] := [.EXIT+1+$I]; .END
        BUF[T] := LFCR;
        <LDA (MESG)R; BUC 202;>
        .EXIT := .EXIT + T + 1;
        .RETURN
TTYIN():  .LOCAL BUF(10), AREA, T,
        ITLTY :=('META  16TNSTAT00MOVE  66TELTYP  INPUT 4 COREIX1',
        #BUF);
        AREA := [.EXIT];  .EXIT := .EXIT+1;
        SPEAK('');  <LDA (ITLTY)R;  BUC 202;>
        $I := AREA;  $J := #BUF;
        AR1 := AREA + 72;  EXPLODEX(); T := 0;
        .FOR $I .TO 71:
             .IF T = 0
                  .THEN .IF [AREA+$I] = 63
                        .THEN T := 1;  [AREA+$I] := ' '; .END
                  .ELSE [AREA+$I] := ' '; .END .END
        .RETURN
```

APPENDIX B (Cont'd)

```
TTYOUT():  .LOCAL BUF(10), AREA, I,
      OTLTY :=('META  16TNSTAT00MOVE  66TELTYP  OUTPUT4 COREIX1',
      #BUF);
      AREA := [.EXIT];  .EXIT := .EXIT+1;
      $I := 71;
      .LOOP WHILE [AREA + $I] = ' ' .A. $I >= 0:
           $I := $I-1; .END
      I := $I;
      .IF I < 70
           .THEN[I+AREA+1] := 032;
           [I+AREA+2] := 077; .END
      $I := AREA;  $J := #BUF; AR1 := AREA+72;  COMPRESSX();
      .IF I < 70
           .THEN[I+AREA+1] := ' ';
           [I+AREA+2] := ' '; .END
      <LDA (OTLTY)R; BUC 202;>
      .RETURN
READ():  .LOCAL AREA, FILE;
      FILE := [.EXIT];  AREA := [.EXIT+1];  .EXIT := .EXIT+2;
      .IF P1INP[ILINE-1].7 /= 26
           .THEN .ERROR UNLESS P1INP[ILINE-1].7 /= 63:
           'EOF READ';  ILINE := 0;
           <LDA FILE;  BUC 202;>
           .ERROR UNLESS SOURCE[1].7 =3: 'BAD READ';
           SOURCE[8] := SOURCE[8]+1;
           .LOOP WHILE P1INP[9].7 = 61:
                <LDA FILE;  BUC 202;>
                .ERROR UNLESS SOURCE[1].7 = 3:'BAD READ';
                SOURCE[8] := SOURCE[8]+1; .END .END
      $I := AREA;  $J := ILINE+#P1INP;
      AR1 := AREA+#INL1-1;  EXPLODEX();
      ILINE := ILINE+10;
      .RETURN
```

APPENDIX B (Cont'd)

```
WRITE():  .LOCAL AREA, FILE, T,
          MORE :=('META  1STNSTAT00MODIFY66PRGRM   NUMWDS1',4096);
          FILE := [.EXIT];  AREA := [.EXIT+1];  .EXIT := .EXIT+2;
          .IF OLINE = 510
               .THEN .IF PRGRM[8]+1\8 = 0
                    .THEN MORE[3] := [FILE+3];
                    MORE[5] := MORE[5]+4096;
                    <LDA (MORE)R;  BUC 202;>
                    .ERROR UNLESS MORE[1].7 = 3:
                    'NO MORE DISC SPACE'; .END
               T1 := COUNT2 * 100;
               P1OUT[510] := COUNT1*100↑24+T1;
               P1OUT[511].3 := COUNT2-COUNT1+1;
               <LDA FILE; BUC 202;> COUNT1 := COUNT2 + 1;
               .ERROR UNLESS PRGRM[1].7 = 3: 'BAD WRITE';
               OLINE := 0;  PRGRM[8] := PRGRM[8]+1; .END
          $I := AREA;  $J := OLINE+#P1OUT;
          AR1 := AREA+#OUTL1-1;  COMPRESSX();
          COUNT2 := COUNT2 + 1;
          P1OUT[OLINE+9].4 := COUNT2\10;  T := COUNT2/10;
          P1OUT[OLINE+9].3 :=  T\10;  T := T/10;
          P1OUT[OLINE+9].2 := T\10;  T := T/10;
          P1OUT[OLINE+9].1 := T\10;
          OLINE := OLINE + 10;
          .RETURN
IONAMES():  .LOCAL V(6);
          .FOR $I .FROM 5:
               V[$I] := ' '; .END
          .ERROR UNLESS STAR[$M].A <= 6:'LONG NAME';
          .FOR $I .FROM STAR[$M].A-1:
               V[$I] := STAR[$M+$I+1]; .END
          POP();  VL1 := BLANKS;
          VL1.0 := V[0]; VL1.1 := V[1]; VL1.2 := V[2];
          VL1.3 := V[3];  VL1.4 := V[4];  VL1.5 := V[5];
          .RETURN
```

APPENDIX B (Cont'd)

```
INITIALIZE(): .LOCAL
     ODISCF:=('META  19TNSTAT00FILE  66PRGRM   UNIT  0=',
              'FORM  5CINLOC 1',#P1OUT,'NUMWDS1',4096),
     IDISCF:=('META  17TNSTAT00REFILE66          RENAME66',
              'SOURCE  INLOC 1',#P1INP);
     .FOR $I .FROM 71:
          ERBUF[$I] := ' '; .END
     TOKENDEPTH := 0;  TK := 0;
     $M := #STARL;  $N := #STACKL-1;  <LDI,070000 (6)R;>
     INCOUNT := 0;  MAXIMUM := 0;
     OUTCOUNT := 0;  OUTX :=1 ; SIGNAL := .EXIT;
     $I := 0;
     .FOR $I .FROM #OUTL2-1:
          OUTBUF[$I] := ' '; .END
     FIELD := 0;
     INBUF[#INL1-1] := 077;  INBUF[#INL2-1] := 077;
     IUNIT := TTY;  INBUF[#INL2-2] := ' ';
     INX := #INL2-2;  SPEAK('INPUT OUTPUT');
     ID();  ID();
     IONAMES();  OUNIT := VL1;
     .IF OUNIT =/= TTY
          .THEN PERM[3] := OUNIT;  OLINE := 0;
          <LDA (ODISCF)R;  BUC 202;>
          PRGRM[8] := 0;  COUNT1 := 1;  COUNT2 := 0;
          P1OUT[511] := 51;
          .LOOP WHILE $I < 510:
               P1OUT[$I+9] := 032;
               $I := $I + 10;  .END .END
          P1OUT[509] := 076;
     IONAMES();  IUNIT := VL1;
     .IF IUNIT =/= TTY
          .THEN IDISCF[3] := IUNIT;
          <LDA (IDISCF)R; BUC 202;>
          SOURCE[8] := 0;
          ILINE := 10;  P1INP[9] := 076; .END
     .RETURN
```

APPENDIX B (Cont'd)

```
COMPLETE():
        .IF OUTX > 0
             .THEN CARD();  .END
        .IF OUTX > #OUTL1+1
             .THEN CARD();  .END
        .IF OUNIT =/= TTY
             .THEN  P1OUT[OLINE-1].7 := 63;
             T1 := COUNT2*100;
             P1OUT[510] := COUNT1*100↑24+T1;
             P1OUT[511].3 := COUNT2-COUNT1+1;
             <LDA (PRGRM)R;  BUC 202;>
             .ERROR UNLESS PRGRM[1].7 = 3: 'BAD WRITE';
             PERM[7] := PRGRM[8] + 1 * 512;
             <LDA (PERM)R;  BUC 202;>
             .ERROR UNLESS PRGRM[1].7 = 3:
             'REDUNDANT OUTPUT NAME';  .END
        .ERROR UNLESS SIGNAL /= 0: 'ILLEGAL PROGRAM';
        .RETURN
NXTCHR():
        INX := INX+1;
        .IF INX >= #INL2
             .THEN .FOR $I .FROM #INL1-2:
                  INBUF[$I] := INBUF[$I+#INL1];  .END
             INX := #INL1;  INCOUNT := INCOUNT+ #INL1;
             .IF IUNIT =/= TTY
                  .THEN READ(#SOURCE, #INPLACE);
                  .ELSE TTYIN(#INPLACE);  .END .END
        .RETURN
EXPLODEX():
        .LOOP WHILE $I < AR1:
             [$I] := [$J].0;
             [$I+1] := [$J].1;
             [$I+2] := [$J].2;
             [$I+3] := [$J].3;
             [$I+4] := [$J].4;
             [$I+5] := [$J].5;
             [$I+6] := [$J].6;
             [$I+7] := [$J].7;
             $I := $I+8;  $J := $J+1;  .END
        .RETURN
```

APPENDIX B (Cont'd)

```
COMPRESSX():
        .LOOP WHILE $I < AR1:
              [$J].0 := [$I];
              [$J].1 := [$I+1];
              [$J].2 := [$I+2];
              [$J].3 := [$I+3];
              [$J].4 := [$I+4];
              [$J].5 := [$I+5];
              [$J].6 := [$I+6];
              [$J].7 := [$I+7];
              $I := $I+8;  $J := $J+1;  .END
        .RETURN
ERROR():  .LOCAL A;
        TTYOUT(#INPLACE);
        .IF INCOUNT + INX > MAXIMUM
            .THEN MAXIMUM := INCOUNT + INX;  .END
        A := MAXIMUM - INCOUNT - #INL1;
        ERBUF[A] := '↑';
        TTYOUT(#ERBUF);  ERBUF[A] := ' ';
        .RETURN
PUSHER():
        .ERROR UNLESS 1 = 0 : 'OUT OF PUSHDOWN LIST';
        .RETURN
ISIT():  .LOCAL X, CONVERT :=(
         9,  9,  9,  9,  9,  9,  9,  9,
         5,  5, 33,  1,  1, 33, 33, 33,
         1,  3,  3,  3,  3,  3,  3,  3,
         3,  3, 33,  1,  1, 33, 33, 33,
         1,  3,  3,  3,  3,  3,  3,  3,
         3,  3, 33,  1,  1, 33, 33, 33,
        17,  1,  3,  3,  3,  3,  3,  3,
         3,  3, 33,  1,  1, 33, 33, 33 );
        AR1 := [[.EXIT]];  AR2 := [.EXIT+1];  .EXIT := .EXIT +2;
        X := CONVERT[INBUF[INX]];  <ANA AR1; STF SIGNAL;>
        .IF SIGNAL /= 0
            .THEN .IF AR2 /= 0
                   .THEN TK := TK+1;
                   .ERROR UNLESS TK < #INL1: 'LONG TOKEN';
                  TK[TK] := INBUF[INX];
                  .IF TOKENDEPTH = 0
                       .THEN MAKETOKEN();  .END .END
            NXTCHR();  .END
        .RETURN
```

APPENDIX B (Cont'd)

```
EXPLODE():   .LOCAL T;
        VL1 := [AR1];  VL2 := VL1 - 1 / 8 + 2;
        $J := 0;
       .FOR $I .TO VL2 - 2 :
              T := [AR1 + $I + 1];
              XPLBUF[$J] := T.0;
              XPLBUF[$J+1] := T.1;
              XPLBUF[$J+2] := T.2;
              XPLBUF[$J+3] := T.3;
              XPLBUF[$J+4] := T.4;
              XPLBUF[$J+5] := T.5;
              XPLBUF[$J+6] := T.6;
              XPLBUF[$J+7] := T.7;
              $J := $J + 8;  .END
        .RETURN
SKIPBLANKS():
        .LOOP WHILE INBUF[INX] = ' ' .V. INBUF[INX] = 63:
              NXTCHR();  .END
        .RETURN
INSERT():
        AR1 := .EXIT;
        EXPLODE();  .EXIT := .EXIT + VL2;  INSERTX();
        .RETURN
INSERTX():
        .ERROR UNLESS TK+VL1 < #INL1: 'LONG TOKEN';
        .FOR $I .FROM VL1-1:
              TK[$I + TK + 1] := XPLBUF[$I];  .END
        TK := TK + VL1;
        .IF TOKENDEPTH = 0
              .THEN MAKETOKEN();  .END
        .RETURN
MAKETOKEN():
        $M := $M - 1 - TK;  .ERROR UNLESS $M > 0:'FULL STACK' ;
        STAR[$M].D := TK + 1;
        STAR[$M].A := TK;
        STAR[$M].P := #ATOM;
        .FOR $I .FROM TK - 1:
              STAR[$M + $I + 1] := TK[$I + 1];  .END
        TK := 0;
        .RETURN
COMP():
        AR1 := .EXIT;  AR2 := 1;  COMPARE();  .EXIT := VL2;
        .RETURN
COMPS():
        AR1 := .EXIT;  AR2 := 2;  COMPARE();  .EXIT := VL2;
        .RETURN
NCOMP():
        AR1 := .EXIT;  AR2 := 3;  COMPARE();  .EXIT := VL2;
        .RETURN
```

APPENDIX B (Cont'd)

```
COMPARE():  .LOCAL TYPE, L;
      TYPE := AR2;  L := [AR1];
      EXPLODE();  VL2 := VL2 + AR1;
      .IF TOKENDEPTH = 0
            .THEN SKIPBLANKS();  .END
      SIGNAL := .EXIT;
      .FOR $I .FROM L-1:
            .IF INBUF[INX + $I] /= XPLBUF[$I]
                  .THEN SIGNAL := 0;  .END .END
      .IF TYPE = 3
            .THEN .IF SIGNAL /= 0
                  .THEN SIGNAL := 0;
                  .ELSE SIGNAL := .EXIT;  .END
            .ELSE .IF TYPE = 2 .A. SIGNAL /= 0
                  .THEN INSERTX();  .END
            .IF SIGNAL /= 0
                  .THEN INX := INX + L;  .END .END
      .RETURN
MARK():
      .IF TOKENDEPTH = 0
            .THEN SKIPBLANKS();  .END
      TOKENDEPTH := TOKENDEPTH + 1;
      .RETURN
TOKEN():
      TOKENDEPTH := TOKENDEPTH - 1 ;
      .IF TOKENDEPTH = 0 .A. SIGNAL /= 0
            .THEN MAKETOKEN();  .END
      .RETURN
SAVE():
      .PUSH $N, 3;
      STACK[$N+1].D := INCOUNT + INX;
      STACK[$N+1].A := OUTCOUNT + OUTX;
      STACK[$N+2].A := $M;
      STACK[$N+2].D := TK;
      STACK[$N+3] := TOKENDEPTH;
      .RETURN
BACKUP():
      .IF INCOUNT + INX > MAXIMUM
            .THEN MAXIMUM := INCOUNT + INX;  .END
      INX := STACK[$N+1].D - INCOUNT;
      OUTX := STACK[$N+1].A - OUTCOUNT;
      $M := STACK[$N+2].A;
      TK := STACK[$N+2].D; TOKENDEPTH := STACK[$N+3];
      .ERROR UNLESS 0 < INX .A. 0 < OUTX:  'EXCESSIVE BACKUP';
      .RETURN
RSTOR():
      .POP $N,3;
      .RETURN
```

APPENDIX B (Cont'd)

```
CARD():
        FIELD := 0;
        .IF #OUTL1 < OUTX
            .THEN .IF OUNIT =/= TTY
                .THEN WRITE(#PRGRM, #OUTBUF);
                .ELSE TTYOUT(#OUTBUF); .END
            OUTCOUNT := OUTCOUNT + #OUTL1;
            .FOR $I .FROM #OUTL1 - 1:
                OUTBUF[$I] := OUTBUF[$I + #OUTL1];
                OUTBUF[$I + #OUTL1] := ' ' ; .END .END
        OUTX := #OUTL1 + 1;
        .RETURN
TAB():  .LOCAL  TABCOL := (0, 7, 23, 47), #N = 4;
        FIELD := FIELD + 1;
        .ERROR UNLESS FIELD < #N : 'TAB ERROR' ;
        T1 := OUTX \ #OUTL1 ;
        OUTX := TABCOL[FIELD] + OUTX - T1 ;
        .RETURN
OUTSTG():
        AR1 := .EXIT;  EXPLODE();  .EXIT := .EXIT + VL2;
        T1 := OUTX + VL1 - 1 / #OUTL1;
        .IF OUTX - 1 / #OUTL1 /= T1
            .THEN CARD();  .END
        .FOR $I .FROM VL1 - 1:
            OUTBUF[$I + OUTX] := XPLBUF[$I]; .END
        OUTX := OUTX + VL1;
        .RETURN
OUTTOKEN():  .LOCAL L;
        L := [AR1].A;
        .ERROR UNLESS [AR1].P = #ATOM: 'NOT A TOKEN';
        T1 := OUTX + L - 1 / #OUTL1;
        .IF OUTX - 1 / #OUTL1 /= T1
            .THEN CARD();  .END
        .FOR $I .FROM L - 1:
            OUTBUF[$I + OUTX] := [AR1 + 1 + $I]; .END
        OUTX := OUTX + L;
        .RETURN
```

APPENDIX B (Cont'd)

```
LENGTH():   .LOCAL L;
        L := STAR[$M].A-2;
        .FOR $I .FROM L-1:
                .IF STAR[$M+2+$I] = ''''
                        .THEN L := L-1;  $I := $I-1; .END .END
        .IF L > 9
                .THEN T1 := OUTX+1 / #OUTL1;
                .IF OUTX-1 / #OUTL1 /= T1
                        .THEN CARD();  .END
                OUTBUF[OUTX] := L / 10;
                OUTBUF[OUTX+1] := L \ 10;
                OUTX := OUTX + 2;
                .ELSE T1 := OUTX / #OUTL1;
                .IF OUTX-1 / #OUTL1 /= T1
                        .THEN CARD();  .END
                OUTBUF[OUTX] := L \ 10;
                OUTX := OUTX + 1; .END
        .RETURN
STAR1P():
        AR1 := #STAR + $M;  OUTTOKEN();
        .RETURN
STAR2P():
        AR1 := #STAR + $M + STAR[$M].D;  OUTTOKEN();
        .RETURN
STAR1():
        STAR1P();  POP();
        .RETURN
POP():
        $M := $M + STAR[$M].D;
        .RETURN
.STOP
```

Distribution List

Name	Room
S. Aranda	2214
J. Barnett	2025
P. Bartram	2336
F. Blair (IBM)	2306
M. Bleier	2324
R. Bleier	3673
E. Book	2332
D. Boreta	1218
R. Bosak	2013
S. Bowman	2322
H. Bratman	2340
R. Brewer	2320
E. Clark	2338
V. Cohen	2326
W. Cozier	2224
P. Cramer	1141B
R. Dinsmore	2220
G. Dobbs	2111B
L. Durham	2424
J. Farell	9731
D. Firth	2310
E. Foote	2415
D. Haggerty	9726
L. Hawkinson (III)	9717
J. Hopkins	2423
C. Irvine	1139
E. Jacobs	2344
S. Kameny	2009B
C. Kellogg	9514
H. Manelowitz	9915
B. Saunders (III)	9717
M. Schaefer	2424
V. Schorre (30)	2330
J. Schwartz	2123
C. Shaw	2428
H. Silberman	9518
R. Simmons	9439
T. Steel	9024
E. Stefferud	9734
P. Styar	9717
A. Vorhaus	2213
C. Weissman (10)	2214
K. Yarnold	9222

External Distribution List

Barry B. Smith
1608 Manzanita Lane
Manhatten Beach, California 90267

George Kreglow
1074 N/7312 S. Jefferson St.
Anaheim, California 92805

Edward Manderfield
North American Aviation
Space & Information Systems Division
D196/322 Bldg. 4
Downey, California

Lee O. Schmidt
Beckman Instruments Systems Division
2400 Harbor Blvd.
Fullerton, California 92634

Andy Chapman
1850 Colby Avenue
Los Angeles, California 90025

Fred Schneider
UCLA Computing Facility
3532 Engineering 3
405 Hilgard Avenue
Los Angeles, California 90024

Dr. P. Abrahams
Information International, Inc.
Room 400, 119 W. 23rd St.
New York, New York 10011

M. Levin
Information International, Inc.
545 Technology Square
Cambridge, Massachusetts 02139

Jorge Mezei 12-018
T. J. Watson Research Center
P. O. Box 218
Yorktown Heights, New York 10598

KIRKLEY

TABLE 1. Syntax of Revised ALGOL 60

<table>
<tr><th>Ref.</th><th>Name</th><th>Definition</th></tr>
<tr><td>[illegible]</td><td>Boolean (logical) value</td><td>true | false</td></tr>
<tr><td>d</td><td>digit</td><td>0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 9</td></tr>
<tr><td>l</td><td>letter</td><td>a | b | c | ··· | z | A | B | C | ··· | Z</td></tr>
<tr><td>r</td><td>relational operator</td><td>$<$ | $\leqq$ | $=$ | $\geqq$ | $>$ | $\neq$</td></tr>
<tr><td>t</td><td>type</td><td>Boolean | integer | real</td></tr>
<tr><td>u</td><td>universe</td><td>All Algol Symbols</td></tr>
<tr><td>ϕ</td><td>empty</td><td></td></tr>
<tr><td>1</td><td>identifier</td><td>l | * l ∪ d</td></tr>
<tr><td>2</td><td>unsigned integer</td><td>d | * d</td></tr>
<tr><td>3</td><td>integer</td><td>2 | {+−} 2</td></tr>
<tr><td>4</td><td>decimal fraction</td><td>.2</td></tr>
<tr><td>5</td><td>decimal number</td><td>2 | 4 | 2 4</td></tr>
<tr><td>6</td><td>exponent part</td><td>$_{10}$ 3</td></tr>
<tr><td>7</td><td>unsigned number</td><td>5 | 6 | 5 6</td></tr>
<tr><td>8</td><td>number</td><td>7 | {+−} 7</td></tr>
<tr><td>9</td><td>proper string</td><td>ϕ | uΔ{‘ ’} | *uΔ{‘ ’}</td></tr>
<tr><td>10</td><td>open string</td><td>9 | ‘*’ | **</td></tr>
<tr><td>11</td><td>string</td><td>‘10’</td></tr>
<tr><td>12</td><td>relation</td><td>24 r 24</td></tr>
<tr><td>13</td><td>Boolean primary</td><td>b | 39 | 36 | 12 | (19)</td></tr>
<tr><td>14</td><td>Boolean secondary</td><td>13 | ¬13</td></tr>
<tr><td>15</td><td>Boolean factor</td><td>14 | *∧14</td></tr>
<tr><td>16</td><td>Boolean term</td><td>15 | *∨15</td></tr>
<tr><td>17</td><td>implication</td><td>16 | *⊃16</td></tr>
<tr><td>18</td><td>simple Boolean</td><td>17 | *≡17</td></tr>
<tr><td>19</td><td>Boolean expression</td><td>18 | 20 18 else*</td></tr>
<tr><td>20</td><td>if clause</td><td>if 19 then</td></tr>
<tr><td>21</td><td>primary</td><td><s>2 | 39 | 36 | (25)</s> 7 | 39 | 1 35 | (25)</td></tr>
<tr><td>22</td><td>factor</td><td>21 | * ↑ 21</td></tr>
<tr><td>[illegible]</td><td>term</td><td>22 | *{×/÷}22</td></tr>
<tr><td>24</td><td>simple arithmetic expression</td><td>23 | {+−}23 | *{+−}23</td></tr>
<tr><td>25</td><td>arithmetic expression</td><td>24 | 20 24 else*</td></tr>
<tr><td>26</td><td>label</td><td>1 | 2</td></tr>
<tr><td>27</td><td>switch designator</td><td><s>1 | [25]</s> 1 [25]</td></tr>
<tr><td>28</td><td>simple designational expression</td><td>26 | 27 | (29)</td></tr>
<tr><td>29</td><td>designational expression</td><td>28 | 20 28 else*</td></tr>
<tr><td>30</td><td>expression</td><td>19 | 25 | 29</td></tr>
<tr><td>31</td><td>actual parameter</td><td>1 | 11 | 30</td></tr>
<tr><td>32</td><td>letter string</td><td>l | *l</td></tr>
<tr><td>33</td><td>parameter delimiter</td><td>, |)32:(</td></tr>
<tr><td>34</td><td>actual parameter list</td><td>31 | *33 31</td></tr>
<tr><td>35</td><td>actual parameter part</td><td>ϕ | (34)</td></tr>
<tr><td>36</td><td>function designator</td><td>1 35</td></tr>
<tr><td>37</td><td>subscript list</td><td>25 | *, 25</td></tr>
<tr><td>38</td><td>subscripted variable</td><td>1 [37]</td></tr>
<tr><td>39</td><td>variable</td><td>1 | 38</td></tr>
<tr><td>40</td><td>left part</td><td>39 :=</td></tr>
<tr><td>41</td><td>left part list</td><td>40 | *40</td></tr>
<tr><td>42</td><td>assignment statement</td><td>41 19 | 41 25</td></tr>
<tr><td>43</td><td>go to statement</td><td>go to 29</td></tr>
<tr><td>44</td><td>dummy statement</td><td>ϕ</td></tr>
<tr><td>45</td><td>for list element</td><td>25 | 25 step 25 until 25 | 25 while 19</td></tr>
<tr><td>[illegible]</td><td>for list</td><td>45 | *, 45</td></tr>
<tr><td>[illegible]</td><td>for clause</td><td>for 39 := 46 do</td></tr>
<tr><td>48</td><td>for statement</td><td>47 54 | 26:*</td></tr>
</table>

TABLE 1. Continued

<table>
<tr><th>Ref.</th><th>Name</th><th>Definition</th></tr>
<tr><td>49</td><td>unlabeled basic statement</td><td>42 | 43 | 44 | 36</td></tr>
<tr><td>50</td><td>basic statement</td><td>49 | 26:*</td></tr>
<tr><td>51</td><td>unconditional statement</td><td>50 | 59 | 60</td></tr>
<tr><td>52</td><td>if statement</td><td>20 51</td></tr>
<tr><td>53</td><td>conditional statement</td><td>52 | 52 else 54 | 20 48 | 26:*</td></tr>
<tr><td>54</td><td>statement</td><td>51 | 53 | 48</td></tr>
<tr><td>55</td><td>compound tail</td><td><s>54 end | 51;*</s> 54 end | 54 ; *</td></tr>
<tr><td>56</td><td>block head</td><td>begin 79 |*; 79</td></tr>
<tr><td>57</td><td>unlabelled compound</td><td>begin 55</td></tr>
<tr><td>58</td><td>unlabelled block</td><td>56; 55</td></tr>
<tr><td>59</td><td>compound statement</td><td>57 | 26:*</td></tr>
<tr><td>60</td><td>block</td><td>58 | 26:*</td></tr>
<tr><td>61</td><td>identifier list</td><td>1 | 1,*</td></tr>
<tr><td>62</td><td>local or own type</td><td>t | own t</td></tr>
<tr><td>63</td><td>type declarations</td><td>62 61</td></tr>
<tr><td>64</td><td>bound pair</td><td>25: 25</td></tr>
<tr><td>65</td><td>bound pair list</td><td>64 | *, 64</td></tr>
<tr><td>66</td><td>array segment</td><td>1 [65] | 1,*</td></tr>
<tr><td>67</td><td>array list</td><td>66 |*, 66</td></tr>
<tr><td>68</td><td>array declaration</td><td>array 67 | 62 array 67</td></tr>
<tr><td>69</td><td>switch list</td><td>29 |*, 29</td></tr>
<tr><td>70</td><td>switch declaration</td><td>switch 1 := 69</td></tr>
<tr><td>71</td><td>formal parameter list</td><td>1 | * 33 1</td></tr>
<tr><td>72</td><td>formal parameter part</td><td>ϕ | (71)</td></tr>
<tr><td>73</td><td>value part</td><td>value 61; | ϕ</td></tr>
<tr><td>74</td><td>specifier</td><td>string | t | array | t array | label | switch | procedure | t procedure</td></tr>
<tr><td>75</td><td>specification part</td><td>ϕ | 74 61; |* 74 61 ;</td></tr>
<tr><td>76</td><td>procedure heading</td><td>1 72; 73 75</td></tr>
<tr><td>77</td><td>procedure body</td><td>54 | code</td></tr>
<tr><td>78</td><td>procedure declaration</td><td>procedure 76 77 | t procedure 76 77</td></tr>
<tr><td>79</td><td>declaration</td><td>63 | 68 | 70 | 78</td></tr>
<tr><td>80</td><td>program</td><td>59 | 60</td></tr>
</table>

The following four sets of synonyms occur in Algol:

identifier (1), variable identifier, simple variable, array identifier, procedure identifier, switch identifier, formal parameter
arithmetic expression (25), subscript expression, lower bound, upper bound
function designator (36), procedure statement
identifier list (61), type list

I am indebted to Mr. A. D. Falkoff and Dr. E. H. Sussenguth for helpful suggestions in the preparation of this note.

Received June 1964; revised July 1964

REFERENCES

1. Naur, P. (Ed.) Report on the algorithmic language ALGOL 60. *Comm. ACM 3*, 5 (May 1960), 299.
2. Taylor, W., Turner, L., and Waychoff, R. A syntactical chart of ALGOL 60. *Comm.* ACM *4*, 9 (Sept. 1961), 393.
3. Naur, P. (Ed.) Revised report on the algorithmic language ALGOL 60. *Comm. ACM 5*, 1 (Jan. 1963), 1.

Abstracts of Papers 1963 ACM National Conference, Denver August

CPM, such as the presentation of tasks, events and activities using a sequential network combined with time estimates.

2. Delineation of the algorithms that are the computational basis for the techniques and a discussion of the activity time estimates that govern the validity of the results.

3. A critical examination of the usefulness of these techniques, which in essence consist of the capacity to distill large amounts of information to obtain forecasts that will be accurate enough to enable effective control to be exercised.

13A PANEL ON COMPUTER SCIENCE CURRICULUM

Moderator: W. F. ATCHISON, *Rich Electronic Computer Center, Georgia Institute of Technology, Atlanta, Ga.*
Panelists: BRUCE W. ARDEN, *University of Michigan, Ann Arbor, Mich.;* ALAN J. PERLIS, *Carnegie Institute of Technology, Pittsburgh, Pa.;* GEORGE E. FORSYTHE, *Stanford University, Stanford, Calif.;* DAVID E. MULLER, *University of Illinois, Urbana, Ill.;* SAUL GORN, *University of Pennsylvania, Philadelphia, Pa.;* AND ROBERT R. KORFHAGE, *Purdue University, Lafayette, Ind.*

The Special Interest Session on Computer Science Curriculum will discuss the general course areas that may be included in a computer science curriculum. There will be explicit discussion of the following six important course areas: (1) Introduction to Digital Computing; (2) Programming Courses; (3) Numerical Analysis; (4) Logical Design; (5) Mechanical Languages; (6) Logic and Algorithms.

These discussions will be led by specialists in the area and are intended to be explicit enough to be of assistance to colleges moving in the direction of a Computer Science Curriculum. There will also be discussion of where such a curriculum should fall within the university complex.

13B COMPILERS FOR SMALL COMPUTERS

no published reference request from author

13B.1: Implementation of a Symbol Manipulator for Heuristic Translation. LEE O. SCHMIDT, *Beckman Instruments Corp., Fullerton, Calif.*

In cooperation with a working group of the ACM Los Angeles Chapter, which was formed to study "syntax-directed" compiling, this translating technique was developed. This paper presents a complete Meta Language for the description of heuristic translation processes, and a Symbol Manipulator on which translation is accomplished. Information is given relating to experience in implementing this Symbol Manipulator on a PDP-1 computer. The translation of POL's is discussed.

13B.2: A Syntax Directed SMALGOL for the 1401. V. A. SCHORRE, *University of California, Los Angeles, Calif.*

The syntax-directed approach toward compiling provides flexibility in implementing various languages, but the advantage emphasized here is saving space at compilation time. Despite the space saving advantages there is little loss in compiling speed, and it becomes possible to implement an elaborate source language, such as ALGOL, on a very small computer. Experience with this approach indicates that ALGOL compilers will prove useful on machines too small to make assemblers feasible.

A compiler is described by syntax equations to which output commands have been added. This description is translated into an interpretive program, which analyzes syntax by an algorithm called "recursive descent."

no published reference request from author

13B.3: A SMALGOL Compiler for the ALWAC III-E at Oregon State University. PHILIP H. HARTMAN, *Harvard University, Cambridge, Mass.*

The OSU SMALGOL compiler allows variables of type **Boolean** and permits the operators ⊃ and ≡, which are excluded from official SMALGOL, but at the moment it does not accept procedure declarations. The compiler is unusual in that it performs the entire translation process in one pass, using only one pushdown list and a table to define identifiers. The push-down list unravels the block structure of the language as well as the algebraic syntax within the statements. The compiler was written in about eight man-weeks and is now in use in several computer courses taught at the University.

13B.4: A Parameterized Compiler Based on Mechanical Linguistics. HOWARD H. METCALFE, *Planning Research Corp., Los Angeles, Calif.*

In 1962, the ACM Los Angeles Chapter established a special interest group on programming languages, and a workshop on syntax-directed compilers was formed within the special interest group. Working as individuals with close coordination, the workship produced four experimental compilers based on syntactical notation. This report introduces the concepts and techniques applied in one of these compilers, as developed by the author.

A translation algorithm is presented, capable of being conveniently parametrized for various source language-target language pairs. Concepts are drawn from modern linguistic theory, and practical considerations in implementation and application are discussed.

Paper Same title Published, Annual Review in Automatic Programming 1963, Vol. 4, p. 125-

10/15/65 requested from Stanford Univ. Library

13B.5: 1620 ALGOL, A Hardware Representation of ALGOL 60 for the IBM 1620. WILLIAM BLOSE, STANLEY POPE AND CHARLES WRIGHT, JR., *Southern Illinois University, Carbondale, Ill.*

1620 ALGOL is the full ALGOL 60 less **own** arrays, integer labels, and a few other restrictions imposed on the system by the character set and 40K memory positions of the IBM 1620. The language is implemented using no more than 3000 instructions for the processor and a relatively small portion of memory for control of the object program. A discussion is given concerning the restrictions imposed on ALGOL 60 with emphasis not only on the restrictions, themselves, but on additions permitted by the machine configuration.

14A PANEL ON THE USE OF COMPUTERS FOR MEDICAL DIAGNOSIS

Chairman: ROBERT S. LEDLEY, *National Biomedical Research Corp., Silver Springs, Md.*
Panelists: THEODOR D. STERLING, *University of Cincinnati, Cincinnati, Ohio;* CLIFTON F. MOUNTAIN, *University of Texas, Houston, Texas;* CEASAR CACERES, *Department of Health, Education, and Welfare, Washington, D. C.;* G. STANLEY WOODSON, *Lovelace Medical Clinic, Albuquerque, New Mexico;* AND JOSEPH BALINTFY, *Tulane University, New Orleans, La.*

There has been recent interest in the possibility of using computers to aid various aspects of medical diagnosis. The problem involved not only include the data collection processes but also, and perhaps even more importantly, include the mathematical and decision-theory model that will be used for the computer program itself. Various topics associated with both the modeling and data collection aspects of the use of computers in medical diagnosis will be discussed by the panelists.

14B SOFTWARE, I/O BUFFERING

14B.1: Data Flow and Storage Allocation for the PDQ-5 Program on the Philco-2000. C. J. PFEIFER, *Westinghouse Electric Corp., Pittsburgh, Pa.*

PDQ-5 is a FORTRAN program which solves the two-dimensional few-group time-independent neutron-diffusion equations. The discrete approximation of these equations results in a matrix