LITTLE User Manual

David Shields

LITTLE Project
Department of Computer Science
New York University
Courant Institute
251 Mercer Street
New York, New York 10012

December 8, 1981

The manual describes the NYU LITTLE implementation of LITTLE as defined by

Guide to the LITTLE Language
David Shields
February 4, 1981

INTRODUCTION

This document describes the NYU LITTLE implementation of the LITTLE Language; it is organized so that material common to all implementations is presented first, followed by material applicable to particular implementations.

The LITTLE compiler consists of three phases: lexical scan (LEX), parse and semantic analysis (GEN) and code generation (ASM). All three make use of a library (LIB) of procedures which supports both compile-time and run-time needs.

Implementations currently exist for the following machines:

```
S10 - Digital Equipment DECsystem-10
      TOPS-10, TOPS-20

S32 - Digital Equipment VAX-11/780
      DEC VMS, Bell UNIX

S37 - IBM System/370, System/360
      CMS

S47 - Amdahl 470
      UTS

S66 - Control Data Corporation Series 6000
      NOS, NOS/BE
```

The abbreviated names such as S10 are also used in this document to identify the various implementations.

COMPILER OVERVIEW

The standard LITTLE compiler consists of three phases: lexical scan (LEX), parse and semantic analysis (GEN) and code generation (ASM). All three make use of a library (LIB) of procedures which supports both compile-time and run-time needs. The phases run as separate programs; on several implementations they are combined to appear as a single program, either by building overlay or using a command file. The chief consequence of the separation is that the compilation listing is generated in pieces, though this also is packaged into one listing where possible.

LEX reads the input stream, performs conditional assembly, builds tokens, eliminates comments, and processes macros. LEX produces a 'token file'. LEX is kept as a separate program mainly because macros are global in scope, which forces the use of a global symbol table for the entire program; In contrast, the GEN and ASM phases work at the procedure level.

GEN reads the input stream from the token file, parses it, performs semantic checking and transformations, and writes a set of operand/operation tables (the VOA) for use by ASM. The semantic transformations largely consist of translating the higher level language constructs - such as DO or WHILE loops - into a lower level.

The parse uses a standard top-down advancing scheme with some various refinements; for example, expressions are parsed using operator precedence, and a hashed search is used to determine if the first token of a statement is a keyword, such as IF or DO. The parse is table-driven; the tables are produced by a parser generator called SYN which accepts as input a grammar in a BNF-like form. SYN is also used by the SETL compiler.

GEN performs some local, machine-independent optimizations. The following optimizations are currently provided: evaluation of expressions at compile time; use of formal identities, such as replacing 'X+0' by X; redundant subexpression elimination; exploitation of constant inputs to conditional branches.

ASM reads the tables produced by GEN, expanding the machine-independent entries into target machine code. ASM completes storage allocation, allocates registers, and performs local optimizatons at the machine level. Some code generators produce assembly source, others produce loader tables.

The LEX and GEN phases are machine independent, and have also served as 'off the shelf' starting points for a number of other compilers written using the LITTLE system, including various SETL compilers. LEX contains about 5000 lines of code, GEN about 10000.

Just as the LITTLE language provides a portable model of machine architecture, so the LITTLE library provides a portable interface to various operating systems. As much as possible, the library is written in LITTLE, both to simplify installation of the LITTLE System and, more imporant, to clarify the specifications of the library procedures. Much of the library is concerned with Input/Output.

Since the language has been in use for some time, it has been the practice where possible to introduce new features by adding appropriate procedures to the library.

The lowest level of the interface is typically coded in machine language. These procedures, collectively referred to as ENV, are for the most part concerned with input/output. The LITTLE I/O facilities ultimately call ’SIO’ procedures which are in ENV. The SIO procedures do such tasks as reading a single line, opening a file, and so forth. The SIO procedures can also be called directly from LITTLE. ENV may also contain recodings of procedures which are nominally written in LITTLE, but for which better performance can be obtained by recoding.

The LITTLE system also includes several auxiliary programs, or utilities. UPD is used to maintain source files. REF is used to produce the cross-reference listing using auxiliary files produced by LEX and GEN. SYN is a meta-compiler than transforms a top-down grammar into a tabular form which can be easily interpreted. Both the LITTLE and SETL compilers use SYN to produce these parse tables. These are described in more detail in a separate section devoted to them.

Conditional Names for Machine Environments

Conditional names of the form ’Snn’, where nn is a two digit integer, refer to a machine environment. The following environments are used in existing LITTLE programs:

S10	Digital Equipment DECsystem-10
S32	Digital Equipment VAX-11/780
S37	IBM System/370, System/360
S47	Amdahl 470, UTS
S66	Control Data Corporation Series 6000

The compiler generates the line ’ .+SET Snn’ before the first line of input, where nn reflects the available target machine.

Compilation Listing Control

The LITTLE compiler provides a number of features to control the content and format of the compilation listing. Since the compiler runs as three steps, it is possible to obtain source listings in either the first phase (lexical processing) or in the second phase (parse and semantic analysis). The second phase listing is generally preferable.

Lines with ' .=' in columns 1 through 3 are directives.

The ' .=TITLE' directive sets the listing title. The directive contains a quoted string which is the title text. The first title directive defines the main title which appears on the top of each page; remaining directives set the subtitle and cause a page eject.

The ' .=EJECT' directive begins a new page of listing. An optional integer parameter may be supplied, in which case a new page is begun only if less than the indicated number of lines remain on the current listing page.

The ' .=LIST' and ' .=PUNCH ' directives control listing and macro 'punch' control, respectively.

Each of these directives may contain a list of parameters, separated by commas. An option is disabled by putting the letters NO in front. Only the first three characters of the parameter code are examined.

The parameters for the punch directive are DEFINE to punch macro definitions and EXPAND to punch expanded text.

The parameters for the LIST directive are as follows:

AUTOTITLE	Use first line of procedure to form title.
CODE	List generated code.
DIRECTIVE	List line containing list directive.
INPUT	List source input in parse (GEN) phase.
LINPUT	List source input in scanner (LEX) phase.
QUALIFIERS	List conditional assembly qualifiers.
REF	Collect references if cross reference feature on.
SKIP	List lines 'skipped' by conditional assembly.

By default, only option REF is enabled.

For example, to generate titles automatically and to list lines skipped by conditional assembly, use

```
    .=LIST INP,AUTO,QUAL
```

A stack is kept of the most recent twenty or so LIST directives. The parameter RESUME may be used to restore list control to that established by the previous LIST directive.

The listing options may be initialized by the compiler option LIST.

PRINT FILE CONTROL

The LITTLE system supports a variety of control functions for the standard print file. These features permit control of limits of print file size, and provide means to generate page numbers, titles and subtitles.

Print file size is determined by following execution-time parameters.

PFLP	Lines per page (default 60)
PFLL	Print file line limit (default 0)
PFPL	Print file page limit (default 100)
PFCC	Print file carriage control (default on)

If both PFLL and PFPL have value zero then there is no limit on the size of the print file. If PFLL is zero and PFPL is greater than zero then PFLL is set to PFPL*PFLP. PFLL determines maximum number of lines to be written. PFPL determines the maximum number of lines with the ’new page’ character ’1’ in position one. If PFCC has value zero, the first position in each line will always be blank. Normally, the first position in the print line is used for carriage control characters. Use of PFCC option permits suppression of carriage control characters, to provide form of print file with minimal spacing between lines.

PRINT FILE TITLES

Procedures LTITLR and STITLR provide an easy way to generate subtitles and titles on the standard print file. The easiest way to use this feature is as follows:

```
      CALL LTITLR(’LABEL’);  $ BEGIN PAGING, LABEL UP TO 15 CHARS.
```

Title text can be entered using the procedure STITLR, as in

```
      CALL STITLR(0, STRING);  $ ENTER STRING AS MAIN TITLE.
      CALL STITLR(1, STRING);  $ ENTER STRING AS SUBTITLE.
```

Compilation date symbol .COMPDATE.

An instance of the symbol .COMPDATE. is replaced by a character string of length 30 which gives the date of compilation. The format is that returned by the LSTIME library primitive.

Overview of Standard Library Procedures

This section summarizes the library procedures by their function. The following section describes each procedure. Here we describe the 'packages' into which the library is organized, so that points common to each package need be mentioned only once.

TIM - Time Procedures

The library provides three procedures to obtain various times: LNTIME provides various integers giving the current clock time, LSTIME expresses the current time and date as a character string, and LETIME measures elapsed program execution time. Use LSTIME only to indicate current time, and not to provide input data to a procedure. Use LNTIME to obtain a representation of the time which can be manipulated.

PARM - Retrieving program parameters

LITTLE does not permit a program (PROG) procedure to have arguments, as it not clear how to specify these parameters in a machine-independent form. However, the library includes several procedures which can retrieve values from the execution environment, and so can be used to obtain program parameters. GETIPP obtains integer values, GETSPP obtains character string values. GETAPP returns the fill parameter string.

GETIPP and GETSPP have a similar calling sequence. There are two arguments, the first is a variable to receive the value, the second is a character string giving the parameter code, the default value, and the alternate value if the parameter code alone is given. The form of the second argument is

```
        'Pcode=Defval/Altval'.
```

Altval is optional, but the slash following Defval must be written. The value is obtained as follows:

1. If Pcode not given, take Defval.
2. If Pcode given with no value, take Altval if Altval is given; otherwise take Defval.
3. If Pcode given with value, take the value.

For example, to obtain file title with parameter code FT and linesize with parameter FL, write

```
    +*  SPPLEN = 20 **

    SIZE  FTVAR(WS);
    SIZE  FLVAR(.SDS. SPPLEN);

    CALL GETSPP(FTVAR, 'FT=SYSIN/TTY');
    CALL GETIPP(FLVAR, 'FL=80/');

    FILE F ACCESS=GET, TITLE=FTVAR, LINSIZE=FLVAR;
```

If FT not given, title is SYSIN; if FT alone given, title is TTY; if 'FT=FNAME' given, title is FNAME. The linesize is 80 unless 'FL=nn' given to explicitly set linesize to nn.

The standard length of parameter code strings and the values of character string parameters is a program parameter. It is suggested that the symbol SPPLEN, defined by macro '+* SPPLEN = 20 **', be used to express this standard length.

FIN - Program termination

The standard manner of terminating program execution is to execute, in the program procedure, a RETURN statement or the END statement which terminates the program procedure. The library procedure LTLFIN permits program termination from within other procedures. LTLFIN can be used to indicate normal or abnormal termination. In the abnormal case, LTLFIN calls a procedure USRATP to permit the user to perform application-specific termination processing.

STR - Character string procedures

The library contains procedures to search character strings, effect string replacement and perform case conversion.

The procedures to search character strings are based on those of the SNOBOL4 language. These procedures support "string sets" which associate a set of characters with a mask. Up to sixteen string sets are supported. The pre-defined string sets and their associated masks are as follows:

1	1B'000001'	SS_BLANK	blank
2	1B'000010'	SS_SEPAR	separators (blank, tab, form feed)
4	1B'000100'	SS_DIGIT	digits 0..9
8	1B'001000'	SS_UCLTR	upper case letters A..Z
16	1B'010000'	SS_LCLTR	lower case letters a..z
32	1B'100000'	SS_BREAK	break (underline) character '_'

The names SS_ are by convention used to name string sets; for example

```
          +*  SS_BLANK = 1 **
```

SS_SEPAR includes blank as well as any other characters which by usual practice are considered equivalent to blank for separating symbols. For ASCII environments, the separators include horizontal tab and form feed.

The string search functions are as follows:

ANYC(C, SS)	match any character in string set SS

ANYS(S, SP, SS)	match any character in string set SS
BLDS(S, SS)	build string set from string S
BRKC(S, SP, C)	break to character
BRKS(S, SP, SS)	break to character in string set SS
NAYC(C, SS)	match any character not in character set SS
NAYS(S, SP, SS)	match any character not in character set SS
RBRC(S, SP, C)	right break to character C
RBRS(S, SP, SS)	right break to character in string set SS
RSPC(S, SP, C)	right span character
RSPS(S, SP, SS)	right span to character in string set SS
SPNC(S, SP, C)	span character
SPNS(S, SP, SS)	span characters in string set SS

In the above, S denotes a character string, SP an integer index, C a character code, and SS a string set either pre-defined or established by execution of BLDS procedure.

BLDS constructs a string set. The first argument is a character string containing the characters to be placed in the string set. The second argument is a mask used to identify the string set. The search procedures are functions which return -1 if no characters are matched; otherwise, the value returned is the number of characters matched. ANYC, ANYS, NAYC and NAYS return either zero or one. BRKC, BRKS, RBRC and RBRS must find a break character or they fail, yielding value -1. RSPC, RSPS, SPNC and SPNS must find a character in the set or they fail, yielding value -1. The search functions also fail if position SP does not lie within string S.

Note that ANYC is used to test if character in particular set. For example, ANYC(C,SS_DIGIT) yields one if C is character code of numeric digit, or zero otherwise. BRKC permits searching for character in string; for example BRKC(S,1,C) yields -1 if C does not occur in S, or yields the number of characters before the first instance of S (BRKC does not match the break character). RSPC can be used to count trailing blanks; for example

```
       RSPC(S, (.LEN. S), 1R )
```

yields -1 if the last character in S is not blank, or otherwise yields the number of trailing blanks in S. The union of character sets can be expressed using the inclusive or operator; for example:

```
       ANYC(C, SS_LCLTR ! SS_LCLTR)
```

yields 1 if C is upper or lower case letter.

Several procedures are provided to assist case conversion. Function CTLC(C) has as operand a character code C. If C is an upper case letter and lower case is available, CTLC yields the lower case code for C; otherwise, CTLC yields C. Similarly, CTUC(C) yields upper case code if argument C is in lower case. Procedure STLC(S) converts upper case characters in S to lower case, and STUC similarly converts to upper case.

Procedures RPLD and RPLE permit efficient replacement or translation of a string. The call RPLD(S1,S2) defines the replacement. S1 and S2 must have equal lengths. Subsequent calls to RPLE replace each instance of a character which occurs in S1 by the corresponding character in S2. For example, the sequence

```
    CALL RPLD('AEIOU','11111')
    S = 'AEXYZ';
    CALL RPLE(S);
```
changes S to '11XYZ'.

FPC - Floating Point Conversion

Procedures CEFR$IO and CREF$IO are used for floating point conversion. They are required for LITTLE I/O, but are necessarily machine-dependent in their implementation if accurate representation is to be achieved. They are used by the SETL system for floating point conversion also. Procedure VNUM$IO is used to implement LITTLE I/O and also may be of interest when using these procedures; it checks validity of numeric constant.

LCP - LITTLE Compiler Print procedures

These procedures are used by the LITTLE compiler to construct the standard output (listing) file. Some of them are also used within the library. Most of them are not of interest in that direct LITTLE I/O statements can do what is needed. However, there are additional procedures which permit an extra level of control in producing the standard output file which is not available from the LITTLE I/O features proper; for example, to maintain titles and subtitles, echo output to the terminal, etc. For example, CONTLPR provides a number of extra control functions. LTITLR provides simple way to provide title at top of each page.

SIO - System Input Output

These procedures are used to implement the LITTLE I/O primitives. They can also be called directly from a LITTLE procedure. Files are identified by small nonzero integers. File 1 is reserved for the LITTLE standard input (unit 1) and file 2 is reserved for the LITTLE standard output (unit 2). At least ten files are usually available, several implementations provide up to twenty.

The first argument of an SIO procedure is the file number. The second argument is a return code. It is set to zero to indicate normal completion. It is set nonzero if an error occurs during the course of execution, except that input procedures use the convention of setting the return code to one to indicate end of data, reserving higher values for error returns. The standard convention is for SIO procedures not to return if an error occurs, but to issue an error message and terminate execution. However, it is possible to test that an open failed. The actions to be taken if an error occurs can be selected using the procedure ERETSIO. ECODSIO can be used to obtain the system-level error number of returns permitted.

OPENSIO opens a file, CLOSSIO closes it. There are a variety of procedures to process text and binary files. In general these procedures need not be used directly, as LITTLE IO should be used if possible.

INC - text inclusion procedures

These procedures implement the text inclusion (INCLUDE) feature of LITTLE, and can be used when implementing a similar feature for other applications. OPNINC intializes, POSINC positions to a particular member, GETINC reads a line, CLSINC terminates. UPDINC is used to process UPD sequence numbers (see description of UPD program parameter).

MEM - Memory access and management

These procedures permit direct access to memory contents. MPTR$LI returns an ’address’, MGET$LI returns the contents an addressed location, and MPUT$LI stores a new values in an addressed location. These procedures are required by LIB to implement the STRING file access type. They should be used with extreme caution.

The REF program requires a ’dynamic array’ which is provided by the procedures DADIMS, DAGETR and DAPUTR. These procedures may be of interest in other applications.

MISC - Miscellaneous Procedures

These procedures fall into no particular grouping. The provide general utility functions that may be of interest; in some cases they are artifacts of the implementation.

Standard Library Procedures

This section describes the standard library (LIB) procedures. Arguments are represented by a name, type name, size specification and optional dimension specification for arguments which are arrays. The type specfication is RD for read, WR for write, and RW for read and write (the entry value of the argument is read and possibly a new value is stored back into the argument). The size specification is a parenthesized expression indicating the expected size, using the standard abbreviations of WS for .WS., PS for .PS., CS for .CS. and SDS n for a string of n characters. The dimension specification, if present, indicates the argument is an array, and gives the expected dimension. The symbol '*' may occur in size and array specifications to indicate that the actual values accessed depend on the values of other arguments in a manner explained in the procedure description.

The initial line of a procedure summary is underlined. The procedure name is followed by a dollar sign which is followed by the 'package' name. Packages are summarized in the previous section.

```
RES (WS) = ANYC(      $       STR
---------------------------------
              CH      RD (CS) ,
              SS      RD (SS_SZ) );
```

Succeed if character CH is in string set SS; fail otherwise.

```
RES (WS) = ANYS(      $       STR
---------------------------------
              ST      RD (.SDS. *),
              SP      RD (PS),
              SS      RD (SS_SZ) );
```

Succeed if the SP-th character of string ST is in string set SS; fail otherwise.

```
CALL BLDS(            $       STR
---------------------------------
              ST      RD (.SDS. *),
              SM      RD (SS_SZ) );
```

Construct a string set of the characters in the string ST, where SM is a mask (.NB. SM = 1) used to identify the string set during subsequent string searches.

```
RES (WS) = BRKC(      $       STR
---------------------------------
              ST      RD (.SDS. *),
              SP      RD (PS),
              CH      RD (CS) );
```

Return the length of the longest substring of ST, starting at position

SP, which is followed by an instance of character CH. Fail if the "break" character CH is not found.

```
RES (WS) = BRKS(      $       STR
---------------------------------
              ST      RD (.SDS. *),
              SP      RD (PS),
              SS      RD (SS_SZ) );
```

Return the length of the longest substring of ST, starting at position SP, which is followed by a character in the string set SS. Fail if a "break" character in SS is not found.

```
CALL CEFR$IO(         $       CRF
---------------------------------
              RV      WR (REAL),
              ARA     RD (WS) *,
              ARAPTR  RD (PS),
              EXPVAL  RD (WS) );
```

Convert exponent and fraction to real value. On entry

```
  ARA(1..ARAPTR)  integers in 0..9
                  (There is an implied decimal point after
                  entry at ARA(ARAPTR).)
  ARA(ARAPTR+1)   is zero for positive value, one for negative
  EXPVAL          is (possibly signed) exponent value
On exit
  RV              is to contain real value
  ARA(ARAPTR+2)   is set as follows
                    0  conversion possible and done
                    1  if invalid input, conversion not possible
                    2  overflow, conversion not possible
```

It is permitted to provide more digits than can be accurately represented; truncation is peritted. However, an error does result (overflow) if the implied value does not fall within the range of the available real values.

```
CALL CHARLR(          $       LCP
---------------------------------
              CH      RD (CS) );
```

Write character CH on the standard output file.

```
CALL CLOSSIO(         $       SIO
---------------------------------
              FN      RD (WS),
              RC      WR (WS) );
```

FN must be a connected file. CLOSSIO closes the file by performing any needed actions. The "connection" to the file is terminated, and the file FN must be re-opened, by a call to OPENSIO, before it can again be used for input-output.

```
CALL CLSINC;          $       INC
---------------------------------
```

Close the text inclusion file established by OPNINC.

```
CALL CLSTERM;         $       MISC
----------------------------------
```

Close the terminal file, if one has been established.

```
CALL CONTLPR(         $       LCP
---------------------------------
              ACT     RD(PS),
              ARG     RW(WS) );
```

Provide for control of standard output file. ACT is an action code as noted below. ARG is used to retrieve or set the value of a parameter used in building the file, as follows:

```
       1   get current position in line
       2   set current position in line
       3   skip forward ARG columns, inserting blanks on way.
       4   tab to column ARG (add blanks on forward tab).
       5   new page action:
           if ARG zero, begin new page.
           if ARG not zero, begin new page if less than ARG lines
           remain on current page.
       6   set paging mode (if on, pages formed)
       7   set titling mode (if on, titles cleared)
       8   set page number field in title line
       9   set date field in title line
      10   get lines per page
      11   set lines per page
      12   get page number
      13   set page number
      14   get line number (within page)
      15   set line number (within page)
      16   get number of lines written
      17   set number of lines written
      18   get line limit
      19   set line limit
      20   get page limit
      21   set page limit
      22   get carriage control status
      23   set carriage control status
      24   get carriage control character
      25   set carriage control character
      26   set list output control flag
      27   set terminal output control flag
      28   get terminal header flag
      29   set terminal header flag
      30   get characters per line
```

```
RES (WS) = CREF$IO(   $       CRF
---------------------------------
              RV      RD REAL,
              NSD     WR (WS),
              SE      WR (WS),
              FP      WR (WS) );
```

Represent the real value RV as a signed exponent SE and a fractional part FP, rounded to NSD significant digits. NSD must be non-negative. If entry value of NSD is zero, set NSD to one and proceed. If NSD exceeds precision of available machine, set NSD to that precision and proceed.

The exit value of the integer FP is such that the most significant NSD digits, viewed as a real with an implied decimal point after the first digit, multiplied by the signed exponent returned in SE, are the rounded value of the operand RV.

For example, if RV=0.526E+10, NSD=3, then on exit, NSD=3, FP = 526..., SE = +9.

The function value is zero if conversion was possible, or one if conversion not possible. For S66, the function value is one if the real value is ’indefinite’ and two if the real value is ’infinite’.

```
SUBR CRFNAM(          $       MISC
----------------------------------
                NAM     WR (.SDS. FILENAMELEN),
                PRM     RD (.SDS. FILENAMELEN),
                NUM     RD  (WS) );
```

Return the name of a ’cross-reference’ file from PRM and NUM, by replacing the leftmost instance of a numeric character in PRM by the character corresponding to the digit NUM.

PRM must contain a digit character and PRM must be in the range 0..9; otherwise execution is abnormally terminated.

The result is returned in NAM; PRM is not altered.

This procedure is used to derive the names of the several cross-reference files required for the cross-reference feature. It is used by cross-reference facility, and will not normally be called by user.

```
RES = CTLC(           $       STR
---------------------------------
              CH      RD (CS) );
```

In fixed case environment, return the argument; otherwise, if argument is upper case, return the lower case equivalent.

```
RES = CTUC(           $       STR
---------------------------------
              CH      RD (CS) );
```

In fixed case environment, return the argument; otherwise, if argument is lower case, return the upper case equivalent.

```
CALL DADIMS(          $       MEM
---------------------------------
              NW      RD (WS),
              NG      WR (WS) );
```

Allocate NW words of free storage, set NG to number of words obtained. This procedure is used in preparation for subsequent invocations of DAGETF and DAPUTR.

```
RES (WS) = DAGETF(    $       MEM
---------------------------------
              I       RD (PS) );
```

Return contents of I-th entry of storage provided by DADIMS. I must be greater than zero and not exceed value of first argument to DADIMS.

```
CALL DAPUTR(          $       MEM
---------------------------------
              I       RD (PS),
              V       RD (WS) );
```

Set contents of I-th entry of storage provided by DADIMS to have value V. I must be greater than zero and not exceed value of first argument to DADIMS.

```
CALL DROPSIO(         $       SIO
---------------------------------
              FN      RD (WS),
              RC      WR (WS) );
```

Indicate that the file is to be "dropped" when it is closed. If possible, the file is to be erased and all evidence of its existence removed when it is closed.

This procedure is used for a "scratch file", and is invoked by the reader of the file, after the call to OPENSIO, and before a call to CLOSSIO, to indicate that, since no further use will be made of the file, it should be removed.

```
CALL DUMPAQ(          $       MISC
----------------------------------
              ST      RD (.SDS. *),
              ARA     RD (WS) (*),
              LO      RD (PS),
              HI      RD (PS) );
```

Write a line with string ST, then write, four entries to a line, the contents of ARA(LO) through ARA(HI), in "machine" format.

DUMPAQ provides a crude listing of the contents of an array slice.

```
CALL ECODSIO(         $       SIO
---------------------------------
              FN      RD (WS),
              RC      WR (WS),
              EC      WR (WS) );
```

Set EC to error code for file FN. After call to another SIO operation, ECODSIO may be called. RC is set to the value of RC returned by the prior SIO procedure call. If an error has occurred, EC is set to a system-dependent value describing the error condition.

```
CALL ERETSIO(         $       SIO
---------------------------------
              FN      RD (WS),
              RC      WR (WS),
              ELEV    RD (WS) );
```

Set error level for file FN to ELEV, which is interpreted as follows:

```
  0   No return if error, abnormal termination
  1   Terse return
  2   Verbose return: issue error message and return
```

The standard input file is initially opened with level 1 (terse return) and the standard output file is initially opened with level 2 (verbose return), to permit the user to detect failure attempting to open these files. The error level is then set to zero so subsequent errors will result in abnormal termination unless the level reset by the user program.

```
CALL ENDLR;           $       LCP
---------------------------------
```

End the current line, and write it to the standard output file.

```
CALL ETITLR(          $       LCP
---------------------------------
              LIN     RD (PS),
              STR     RD (.SDS. *),
              POS     RD (PS),
              LEN     RD (PS) );
```

Enter string STR into title line, beginning at position POS. Enter LEN characters, padding with blanks if actual length of STR is less than LEN.

Enter into the main title if LIN is zero, otherwise enter into the subtitle.

```
CALL GETAPP(          $       PARM
----------------------------------
              ST      WR (.SDS. SL);
              SL      RD (PS) );
```

Return up to SL characters of the program parameter string in string ST. Set the string origin and length fields. The returned length of ST will never exceed SL.

```
CALL GETBSIO(         $       SIO
---------------------------------
              FN      RD (WS),
              RC      WR (WS),
              ARA     WR (WS),
              NDX     RD (PS),
              NC      RW (PS) );
```

FN must be the file number of a connected file. On entry, NC gives the number of characters to be transmitted; if NC is zero, then execution proceeds as though NC has as value the LINESIZE of the file.

Read the next line from file FN. Let its length be AC. If AC exceeds NC, set AC to NC. Then set NC to AC, and store the characters of the line in ARA, starting at position NDX. The characters are stored one character per word, right-justified with zero fill. Set NC to the number of characters stored in ARA.

```
CALL GETCSIO(         $       SIO
---------------------------------
              FN      RD (WS),
              RC      WR (WS),
              ARA     WR (WS),
              NDX     RD (PS),
              NC      RD (PS) );
```

FN must be the file number of a connected file. On entry, NC gives the number of characters to be transmitted; if NC is zero, then execution proceeds as though NC has as value the LINESIZE of the file.

Read the next line from file FN. Let its length be AC. If AC exceeds NC, set AC to NC. Store the characters of the line in ARA, starting at position NDX. The characters are stored one character per word, right-justified

```
CALL GETINC(          $       INC
---------------------------------
              ARA     WR (WS) (*),
              LO      RD (PS),
              HI      RD (PS),
              FIN     WR (PS) );
```

Obtain the next line from the standard input file, performing text inclusion. The line obtained is returned in ARA(LO..HI) and is padded with blanks if necessary. FIN is set nonzero when the end of the file is encountered.

```
CALL GETIPP(         $       PARM
----------------------------------
              PVAR    WR (WS),
              PSTR    RD (.SDS. FILENAMELEN) );
```

Return the integer value of a program parameter. PSTR has the form

```
  'KEY=DEFVAL/ALTVAL'
```

where KEY identifies the parameter, and DEFVAL is an integer. ALTVAL is an integer. ALTVAL is optional, execution proceeds as though DEFVAL had been written.

Search the program parameter string for the first instance of KEY. It not found, set PVAR to DEFVAL and return. If found, proceed as follows:

1. If no value given (no '=' after KEY), set PVAR to ALTVAL and return.
2. If value given, convert it as integer and assign to PVAR, and return.

```
CALL GETSPP(         $       PARM
----------------------------------
              PVAR    RD (.SDS. FILENAMELEN),
              PSTR    RD (.SDS. FILENAMELEN) );
```

Return the string value of a program parameter. PSTR has the form

```
  'KEY=DEFVAL/ALTVAL'
```

where KEY identifies the parameter, and DEFVAL is an string. ALTVAL is an string. ALTVAL is optional, execution proceeds as though DEFVAL had been written.

Search the program parameter string for the first instance of KEY. It not found, set PVAR to DEFVAL and return. If found, proceed as follows:

1. If no value given (no '=' after KEY), set PVAR to ALTVAL and return.
2. If value given, assign it to PVAR and return.

```
GETVSIO              $       SIO
---------------------------------
```

GETVSIO is identical to GETWSIO except that the last argument is a WR parameter. Input proceeds without the usual padding and the last parameter is set to the number of characters read.

```
CALL GETWSIO(        $       SIO
---------------------------------
              FN      RD (WS),
              RC      WR (WD),
              ARA     WR (WS) (*),
              NDX     RD (PS),
              NC      RD (WS) );
```

Read the next line from file FN. If end of file, set RC to one; otherwise, set RC to zero. Place at most NC characters from the line in ARA(LO) to ARA(HI). Characters are packed. Truncation occurs if not all NC characters can be stored in ARA. Blanks are added as needed.

```
CALL HEXLPR(         $       LCP
---------------------------------
              ARG     RD (WS),
              COL     RD (PS) );
```

Write the hexadecimal value of ARG in the next COL columns of the standard output file, right adjusted with no leading zeros. Start a new line if less than COL positions remain on the current line.

```
CALL INTLPR(         $       LCP
---------------------------------
              ARG     RD (WS),
              COL     RD (PS) );
```

Write the integer value of ARG in the next COL columns of the standard output file, right adjusted with no leading zeros. Start a new line if less than COL positions remain on the current line. Set the first non-blank position to the minus character '-' if ARG is negative. If the value cannot be fully represented, set the first column of the field to '*'.

```
CALL INTLR(          $       LCP
---------------------------------
              VAL     RD (WS) );
```

CALL INTLR(ARG) is equivalent to CALL INTLR(ARG, 5);.

```
CALL LCTIME(          $       TIM
---------------------------------
              ARA     WR (CS) (*),
              ARAMAX  RD (PS) );
```

Determine the current time by calling LSTIME. Then return the 'unpacked' time in ARA as an array of characters. If ARAMAX exceeds LSTIMELEN, provide extra blanks; if ARAMAX is less than LSTIMELEN, store at most ARAMAX characters. LSTIMELEN is macro for length of LSTIME result, should be 30.

```
CALL LETIME(          $       TIM
---------------------------------
              ETIM    WR (WS) );
```

Set ETIM to the elapsed execution time in milliseconds. The initial value of ETIM is not necessarily zero. No explicit check is made for integer overflow, so that care should be exercised when timing 'long' jobs.

```
CALL LNTIME(          $       TIM
---------------------------------
              ARA     WR (WS) (8) );
```

Return the current time represented as eight integers in ARA:

TA(1)	Year
TA(2)	Month: 1 to 12
TA(3)	Day of month: 1 to 31
TA(4)	Hour of day: 0 to 23
TA(5)	Minute of hour: 0 to 59
TA(6)	Second of minute: 0 to 59
TA(7)	Day of year: 1 to 366
TA(8)	Day of week: 1 to 7 (Sunday is one)

```
CALL LSTIME(          $       TIM
---------------------------------
              TS      (.SDS. LSTIMELEN) ); $ LSTIMELEN=30
```

Set TS to indicate the current time. LSTIMELEN is the length of the result; it is currently 30, and no change in the result length is expected. The format is best shown by example; the next to last second of 23 March 1976 is represented as:

```
      '   TUE  23 MAR 76  23.59.68   '
       123456789A123456789B123456789C
```

The contents of LSTIME result are derived from LNTIME. The format of LSTIME may be adjusted in some cases to conform to the usual conventions for a particular implementation. Programs that need to 'decode' the current time, should use LNTIME and not muck about with LSTIME. The result from LSTIME is typically used to label listings.

```
CALL LTITLR(          $       LCP
---------------------------------
              TS      RD(.SDS. *) );
```

Fully identify the standard output file with a title derived from TS, enable paging, and so forth. Standard components of the LITTLE system identify themselves by using LTITLR. Current practice is to use argument of the form

```
   'NAM(ddddd)'
```

where NAM is three-character component namt, and ddddd is Julian date of last change to component; for example, 'LEX(80023)'.

```
CALL LTLFIN(          $       FIN
---------------------------------
              TLEV    RD (WS),
              TCOD    RD (WS) );
```

Terminate execution of the program. LTLFIN does not return. TLEV is the level of termination, as follows:

0	normal termination
1	abnormal termination detected by LITTLE system
2	abnormal termination due to system problem, error

TCOD is a termination code interpreted as follows:

1. If TLEV = 0, the program terminated, TCOD is set:

0	normal termination
4	'warnings' detected
8	'errors' detected

2. If TLEV = 1, the program terminated abnormally:

3. If TLEV = 2, some 'system' problem forced termination. TCOD is, if possible, a machine-dependent encoding of the code provided by the system to report the problem.

```
CALL LTLINI(          $       MISC
----------------------------------
              WHO     RD (PS) );
```

Start execution of a LITTLE program. Some implementations permit procedures written in LITTLE to be combined with procedures written in other languages. WHO identifies the 'master'. It is normally zero, indicating that LITTLE is to open the standard input and output files, which are the LITTLE units one and two, respectively.

Use of stand-alone LITTLE programs requires no explict user-provided calls to this procedure. The LITTLE PROG procedure implicitly includes a call to LTLINI(0) as the first executable statement to establish control.

Usage with a non-zero argument value is machine-dependent.

```
CALL LTLREGS;         $       MISC
----------------------------------
```

Save the contents of the machine registers, display their current values, restore the registers and continue execution.

LTLREGS is provided to assist in program checkout. It is not essential and will not necessarily be provided for every implementation. The intent is to produce on the standard output file a symbolic display of the current machine register contents, whatever that means, with a minimal corruption. Some corruption may of course result from the act of trying to display them, in a machine- dependent way.

```
CALL LTLSIO(          $       SIO
---------------------------------
              WHO     RD (PS) );
```

Initiate SIO routines. WHO is normally zero to indicate call from LTLINI. Direct user calls to this procedure should not occur, and should occur only if the user is trying to bypass the normal course of events.

```
CALL LTLTERM(         $       MISC
----------------------------------
              PHASE   RD (WS),
              TCOD    RD (WS) );
```

Continue exeucution of a multi-pass program. The intent is to permit successive programs to operate as phases in which each phase reports its phase number and a return code to be used by LTLTERM to determine if execution is to proceed, and if so, which phase is to be invoked next. Where programs cannot be combined, an acceptable implementation, provided in the standard LITTLE library is just:

```
  SUBR LTLTERM(PHASE, TCOD);
  ...
  CALL LTLFIN(TCOD, 0);
```

```
CALL LTLXTR;          $       MISC
----------------------------------
```

Write the current program position on the standard output file, usually as a list of the active procedures and the relative position within them. XTR stands for ’eXecution Traceback Report’.

This procedure is normally called only when serious errors have been encountered, and is invoked to attempt to describe just where the error occurred.

```
R (WS) = MGET$LI(     $       MEM
---------------------------------
              A       RD (PS) );
```

Return contets of memory location with address A.

```
R (PS) = MPTR$LI(     $       MEM
---------------------------------
              ARG     RD (*) );
```

Return address of ARG.

```
CALL MPUT$LI(         $       MEM
---------------------------------
              P       RD (PS),
              V       RD (WS) );
```

Set the contents of memory location P to be V. P is as returned by MPTR$LI. Note that

```
  A = MPTR$LI(V);
  CALL MPUT$LI(A, MGET$LI(A)+1));
```

is one (expensive) way of incrementing value of V.

```
CALL NAMESIO(         $       SIO
---------------------------------
              FN      RD (PS),
              RC      WR (WS),
              S       WR (.SDS. *),
              SL      RD (PS) );
```

Determine the "fully qualified" name of the file currently associated to file FN. Format S as a string and store at most SL characters of the name in S. Set RC as follows:

```
      0  normal return, full name available
      1  string available, but full name could not be stored
      2  file not open, or error attempting to retrieve name
```

```
RES (WS) = NAYC(      $       STR
---------------------------------
              CH      RD (CS),
              SS      RD (SS_SZ) );
```

Succeed if character CH is not in string set SS; fail otherwise. NAYC is inverse is ANYC.

```
RES (WS) = NAYS(      $       STR
---------------------------------
              ST      RD (.SDS. *),
              SP      RD (PS),
              SSK     RD (SS_SZ) );
```

Succeed if the SP-th character of string ST is in string set SS and return the length of the longest substring of characters in SS; otherwise fail.

```
CALL OCTLPR(          $       LCP
---------------------------------
              ARG     RD (WS),
              COL     RD (PS) );
```

Write the value of ARG in octal, right adjusted in COL columns. Blank fill if necessary, with no indication of truncation if the value cannot be represented exactly in COL columns. Start a new line if the result cannot be fully represented on the current line.

```
CALL OCTLR(           $       LCP
---------------------------------
              ARG     RD (WS) );
```

CALL OCTLR(ARG) is just CALL OCTLPR(ARG, (WS+2)/3); .

```
CALL OPENSIO(         $       SIO
---------------------------------
              FN      RD (WS),
              RV      WR (WS),
              AC      RD (PS),
              TITL    RD (.SDS. FILENAMELEN),
              LNS     RD (WS),
              LNSRET  WR (WS),
              DISP    RD (PS),
              SITEA   RD (PS) );
```

Open file FN for processing. AC indicates desired access:

1	GET
2	PRINT
3	PUT
4	READ
6	WRITE

TITL identifies the file. If null an implicit title derived from the file number is to be used. If '0' the file is a null file. Attempts to read a null file (access GET or READ) always return the end-of-file value. Attempts to write (access PRINT PUT or WRITE) succeed, even though no data is actually transmitted.

LNS is meaningful only for access GET, PRINT or PUT, and specifies the number of characters in a line. If zero, a ’default’ value is implied which depends on the access mode and the file number.

LNSRET is meaningful only for access GET, PRIN and PUT, and is returned as follows:

1. If LNS nonzero, it is set to LNS.

2. If LNS zero and access is GET, it is set to the actual maximum length of lines in the file, if this can be determined.

3. If LNS zero and access is PUT or PRINT, it is set to an appropriate system-dependent value.

DISP specifies a ’disposition code’ stating what is to be done when the file is closed. It is currently ignored. Procedure DROPSIO is used to request that file be ’dropped’, and is called after OPENSIO.

SITEA is an additional parameter provided for machine-dependent use. It is not currently used, and can hence be ignored.

```
CALL OPNINC(          $       INC
---------------------------------
              INAME   RD (.SDS. FILENAMELEN),
              MNAME   RD (.SDS. FILENAMELEN),
              ICODE   RD (.SDS. FILENAMELEN),
              UPDARG  RD (PS) );
```

Ignore argument INAME. It was onced used to identify file, but for now it is ignored as the standard input file is implied.

Read the standard input file with text inclusion.

ICODE specifies codes for INCLUDE and MEMBER keywords...

If UPDARG is not zero, the first eight columns of all lines read contain LTLUPD sequence information which is to be ignored.

```
CALL OPNTERM(         $       MISC
----------------------------------
              NAM     RD (.SDS FILENAMELEN) );
```

Open a file NAM which for interactive use is usually connected to the user’s terminal.

```
CALL PROMSIO(         $
-----------------------
              FN      ,
              RC      WR (WS),
              S       RD (.SDS. *) );
```

If file FN is opened for reading from an interactive terminal, set the ’prompt string’ to be S. The TERMP program parameter specifies the initial value of the value of the prompt string for files opened for input.

```
CALL PUTCSIO(         $       SIO
---------------------------------
              FN      RD (PS),
              RC      WR (WS),
              ARA     RD (WS)(*),
              LO      RD (PS),
              NC      RD (WS) );
```

If NC is zero, proceed as though NC=linesize of file FN. Write a line consisting of the unpacked characters of ARA, starting at index LO. Truncate if NC>linesize, pad with blanks if NC>linesize.

```
PUTWSIO               $       SIO
---------------------------------
```

PUTWSIO is similar to PUTCSIO, except that the entries of ARA are packed, WS/CS characters per word.

```
RBRC                  $       STR
---------------------------------
```

RBRC is similar to BRKC, except the string is searched from right to left.

```
RBRS                   $      STR
---------------------------------
```

RBRS is similar to BRKS, except the string is searched from right to left.

```
CALL RDRWSIO(         $       SIO
---------------------------------
              FN      RD (WS),
              RV      WR (WS),
              ARA     RD (WS),
              LO      RD (PS),
              NW      RD (PS) );
```

Read binary data from file FN. Read NW words into array ARA, starting an index LO.

```
CALL READOS(          $       PARM
----------------------------------
              KEY     RD (WS),
              CODE    RD (.SDS. *),
              IFPRES  WR (1),
              IFVAL   WR (1),
              INVAL   WR (WS),
              ISVAL   WR (.SDS. *) );
```

READSOS is called by by GETIPP and GETSPP; it provides some extra functions in obtaining program parameters that be of interest. KEY describes desired function, as follows:

1	seek integer parameter
2	seek octal parameter
3	seek string parameter
4	set INTVAL to number of parameters and return
-I	seek value of I-th parameter

IFPRES is set to one if the parameter is present. If parameter value present IFVAL is set to one. If value present and KEY is one or two, INVAL is set to numeric value. If value present and KEY is three, ISVAL is set to string value.

Parameter specification are separated by commas; some implementations accept commas between brackets as part of directory specification.

```
CALL READSOS(        $       PARM
---------------------------------
             CA      WR (CS)(*),
             NC      RW (PS) );
```

Return the full program parameter string in CA. Each entry contains one character, right-adjusted with zero fill. On entry, NC gives the maximum number of characters to be stored. On exit, NC gives the number of available characters.

```
CALL REMARKL(        $       MISC
---------------------------------
             S       RD (.SDS. *) );
```

Write the string S. For interactive systems, the string should be sent to the user's terminal; otherwise, it can be sent to the standard output file.

REMARKL is a special procedure in that the intent is to make the argument available to the user 'when all else fails'. The exact semantics are hard to specify, but note that REMARKL is used in 'panic mode' when LITTLE sees chaos approaching.

```
CALL REWISIO(        $       SIO
--------------------------------
             FN      RD (WS),
             RC      WR (WS),
             AC      RD (PS) );
```

Rewind file FN. If AC is zero, simply rewind the file. If AC is non-zero it should be one of

1	GET access
4	READ access

implying that writing is to be terminated and the file is to be repositioned at the beginning for input.

```
CALL RPLD(            $       STR
---------------------------------
              S1      RD (SDS. *),
              S2      RD (.SDS. *) );
```

S1 and S2 must have the same length. Initialize for subsequent use of RPLE. Each character of S1 is to be mapped to the corresponding character of S2.

```
CALL RPLE(            $       STR
---------------------------------
              S       RW (.SDS. *) );
```

Translate the string S according to the translate table defined by RPLD.

```
RV (WS) = RSPC(       $       STR
---------------------------------
              ST      RD (.SDS. *),
              SP      RD (PS),
              SS      RD (SS_SZ) );
```

Return the length of the longest substring of ST, starting at position, which consists of characters in the string set SS. RSPC fails if no character in SS is found.

```
RV (WS) = RSPS(       $       STR
---------------------------------
              ST      RD (.SDS. *) ,
              SP      RD (PS),
              SS      RD (SS_SZ) );
```

Return length of longest substring of S, starting at position SP, which consists of characters in SS. A character must be found or else the search fails. The search is from right to left.

```
CALL SIGL$IO(         $       MISC
----------------------------------
              FN      RD (WS),
              IL      RD (PS) );
```

Set the error threshhold level for the file FN to IL. IL is interpreted as follows:

0	any problem treated as error
1	ignore truncation/conversion errors
2	continue even if severe errors

```
RES (WS) = SPNC(      $       STR
---------------------------------
              ST      RD (.SDS. *),
              SP      RD (PS),
              CH      RD (CS) );
```

Return length of longest substring of S, starting at position of SP, which consists of character CH. At least instance must be found or SPNC fails.

```
RES (WS) = SPNS(      $       STR
---------------------------------
              ST      RD (.SDS. *),
              SP      RD (PS),
              SS      RD (SS_SZ) );
```

Return length of longest substring of S, starting at position SP, which consists of characters in the string set SS. At least one character must be found, or SPNS fails.

```
CALL STITLR(          $       LCP
---------------------------------
              L       RD (PS),
              S       RD (.SDS. *) );
```

If L is zero, enter S as the main title for the standard output file; otherwise enter S as the subtitle.

```
CALL STLC(            $       STR
---------------------------------
              S       RW(.SDS. *) );
```

If lower-case available, convert S to lower case; otherwise, S is not changed.

```
CALL STUC(            $       STR
---------------------------------
              S       RW (.SDS. *) );
```

If lower-case available, convert S to upper case; otherwise, S is not changed.

```
CALL SYSFIN(          $       FIN
---------------------------------
              TL      RD (WS),
              TC      RD (WS) );
```

Terminate execution. TL is zero for "normal" completion. In this case, TC is interpreted as follows:

```
      0       normal completion
      4       program completed, but warning messages issued
      8       program completed, but errors were detected
```

If TL is nonzero, the program completed abnormally. TL is interpreted as follows:

```
      1       error detected by LITTLE
      2       error detected by operating system
```

```
CALL SYSINI(          $       MISC
----------------------------------
              C       RD (PS) );
```

Perform any necessary system-dependent initialization. C is normally zero, indicating that LITTLE is to maintain control of the standard input and output files. This procedure is automatically called by a LITTLE program (PROG).

```
CALL TEXTLR(          $       LCP
---------------------------------
              S       RD (.SDS. *) );
```

Write the string S on the standard output file.

```
CALL TINTLR(          $       LCP
---------------------------------
              S (.SDS *),
              I (WS) );
```

Send string S and integer I to standard output, with spacing and label information to make clear intent to take S as label for value I displayed.

```
CALL USRATP;          $       FIN
---------------------------------
```

USRATP stands for ’USer Abnormal Termination Procedure’. When LTLFIN is called in event of abnormal termination, it calls USRATP early on to give the user a chance to provide termination information of his own design. A default USRATP that does nothing is provided, and several implementations, due to loader restrictions, provide only the default implementation.

```
CALL VNUM$IO(         $       FPC
---------------------------------
              ARA     RW (WS) *,
              ARAPTR  RD (PS),
              EXPVAL  WR (WS) );
```

This procedure 'verifies' numeric constants. On entry ARA(1..ARAPTR) holds characters representing numeric constant. On exit:

ARA(1..ARAPTR)	holds integers in range 0..9
ARA(ARAPTR+1)	is zero for positive value, one for negative value
ARA(ARAPTR+2)	is zero if verified, one if failure
ARA(ARAPTR+3)	indicates presence of decimal point. It is zero if no decimal point; otherwise if 1 + number of digits after decimal point
ARA(ARAPTR+4)	is zero if no exponent, one if exponent present
EXPVAL	if ARA(ARAPTR+4) is nonzero, EXPVAL is signed integer holding exponent value

```
CALL WORDLR(          $       LCP
---------------------------------
              W       RD (WS) );
```

Write word W to standard output as characters.

```
CALL WORDSR(          $       LCP
---------------------------------
              ARA     RD (WS) *,
              LO      RD (PS),
              HI      RD (PS) );
```

Same as

```
  DO I = LO to HI;
    CALL WORDLR(ARA(I));
  END DO;
```

```
CALL WRTWSIO(         $       SIO
---------------------------------
              FN      RD (PS),
              RC      WR (WS),
              ARA     RD (WS) (*),
              LO      RD (PS),
              NW      RD (PS) );
```

Write NW words to file FN, starting at ARA(LO).

PROGRAM PARAMETERS

This section describes the program parameters supported by the compiler and run-time library. These parameters are specified as part of the command line used to invoke the LITTLE system. The system program parameters are described in the LITTLE format

```
        NAME=DEFVAL/ALTVAL
```

where NAME is the parameter value, DEFVAL is the default value if the parameter is not otherwise specified, and ALTVAL is the value taken if the parameter name alone is given. Parameter values are either decimal integers or character strings. For example, given

```
        P=0/1
```

then if P not mentioned, value 0 is implied. If P alone specified, then value 1 is implied. If P=n specified, the value n is implied. A number of parameters have the form NAME=0/1. Such values are logical switches in that they select one of two cases, according as value is zero or non-zero. In the latter case, the option is said to be ENABLED or SELECTED.

Each parameter description mentions the phases for which the parameter has meaning. Note that the parameter codes have been chosen so that the same list can be passed to all phases; i.e., the same parameter does not have differing meanings in different phases.

Parameter values are sought left to right so that, for example,

```
        P=1,LIST,P=2
```

yields value 1 for parameter P.
Parameters specifying files tend to be machine-dependent;
hence default filenames are given in the description of the available implementations.

AD=0/1 (GEN checkout) ASM VOA Dump

Controls whether GEN produces internal VOA dump at end of each procedure compiled.

ATS=1/0 (ASM S10,S32) Assembly Time Stamp

Controls whether the generated code (see CODE option) contains a line in each procedure identifying the ASM version and time of compilation.

A0=1/0 (ASM S66) A0 Register Save

Controls whether register A0 is saved and restored by each procedure that uses it. Default is compatible with FTN compiler conventions.

PROGRAM PARAMETERS

B=LGO/ (ASM S66) Binary File

Specifies file to receive generated object code.

B1=1/0 (ASM S66) B1 Constant status

Controls whether register B1 always contains one.

CODE=filename (ASM S10,S32)

Specifies file to receive generated code.

DMP=0/1 (LIB S66) Abnormal Termination Dump

Controls whether storage dump is to be generated if program terminates execution abnormally.

CIS=0/... (GEN) Check Index Size

Controls whether compiler checks that the size of array subscript does not exceed the specified value. CIS=0 suppresses this check. The value obtained if CIS alone is specified is the machine-dependent pointer size (.PS.).

DA=1/0 (GEN) Default Access

Controls whether each procedure compiled is to be given access to each NAMESET defined in the first procedure compiled. If not selected, then each procedure can only reference global variables in NAMESETs explicitly named in an ACCESS statement in the procedure body.

DECK=0/SYSPUNCH (ASM S37,S47) Object DECK file

Specifies file to receive object deck. Parameter NODECK suppresses generation of object deck.

END=PRG/SEC (ASM S10 only) END option

Specifies how ASM is to end the generated code file. If END=0 no last line generated. END=PRG indicates end of program; END=SEG indicates end of segment.

ETIM=1/0 (LIB) Execution Time

Controls whether the amount of execution of time is reported on the terminal. Specify zero to suppress this report. The default for this option varies according as it is considered appropriate to report this time.

EXPIRE=0/366 (GEN) Expiration date

Controls whether call to procedure to check expiration date is to be included. If nonzero value specified, it gives the number of days from compilation for which execution is permited.

FAG=0/1 (ASM S10,S32,S37,S47) Functions Alter Globals

Controls whether generated code must assume that functions may alter global variables. This is required for some parts of the SETL system, which (unfortunately) violate LITTLE specifications.

GS=1/0 (GEN)

Controls construction of a "Global Start" NAMESET for the first procedure compiled which contains all variable not explicitly allocated to a NAMESET. If the GA option is nonzero, subsequent procedures will be given access to this NAMESET.

HELP=/ES (GEN)

MONITOR option which enables MONITOR debugging features. The default is no debugging, and HELP along enables ENTRY and STORES MONITOR traces. Help string may contain any of

```
   E  trace entry
   S  trace stores
   F  trace flow
   C  check indexed array assignments
   0  to suppress HELP
```

I=filename (LIB) Input file

Specifies the standard input file.

ILIB=filename (LIB) Inclusion library file

Specifies the name of the file to be searched to resolve .=INCLUDE text inclusion directives.

IMEM=/ (LEX) Initial MEMBER

Specifies name of MEMBER from standard inclusion library to be included before reading source. In effect, this text is processed as if it occurred just before the first source line.

ISET=/ (LEX) Initial SET

Specifies name of one or more conditional assembly symbols to be initi
SET. If specified, the effect is as if the line

```
  .+SET name
```

occurred just before the first line of the source file for each symbol specified. Multiple symbols are specified by separating them with plus signs; for example

```
  ISET=N1+N2
```

Note also that the symbol Snn is also initially defined, where nn identifies the machine on which the compilation occurs.

IV=0/1 (ASM S32) Integer Overflow

Controls whether the generated code enables integer overflow interrupts. The default (zero) requests that iteger overflow condition be ignored.

L=filename (LIB) Standard Listing (Output) File

Specifies the standard output file. Use L=0 to suppress output file.

LCP=1/0 (LEX, GEN, ASM) List Compilation Parameters

Determines if program parameters are listed. Use zero value to suppress listing of program parameters.

LCR=0/1 (LEX)

Controls whether lexical cross reference map is to be generated.

LCS=1/0 (LEX, GEN, ASM) List Compilation Statistics

Controls whether statistics on performance and resource usage are to be listed on the standard output file.

LEL=25/ (LEX) Lexical Error Limit

Specifies the error limit for LEX. When more than specified number of lexical errors are detected, abnormal termination of the compilation is forced.

LIST=QS/AIQS (LEX,GEN) LISTing options

Determines initial listing parameters. Codes are as follows:

```
    A - autotitle, automatic title mode.
    C - code, list generated code.
    D - define, punch macro definitions.
    E - expand, punch expanded text.
    I - input, list input (GEN phase).
    L - linput, list input (LEX phase).
    Q - qualifier, list conditional assembly
        qualifiers.
    R - reference, enable reference option.
    S - skip, list lines skipped by
        conditional assembly.
    0- ignore LIST directives in input.
```

Note that default is not to list source text so that LIST or LIST=x with L or I in option string x must be specified to include source text in compiler listing file.

LOAD=SYSLIN (ASM S37,S47) LOAD module name file spec

Specifies file to receive generated load module. The NOLOAD parameter can be used to suppress load module generation.

LT=0/1 (LEX checkout) List Tokens

Controls whether LEX lists tokens to be sent to GEN phase.

MDC=0/1 (LEX) List Machine Dependent Constants

Controls whether machine-dependent constants are to be listed at end of LEX listing. The list includes S type constants and R type constants containing more than one character.

MEAL=1/0 (GEN) Monitor Entry Argument List

Controls whether TRACE ENTRY option for Monitor package. is to list values of arguments on entry will not be listed; otherwise, argument values will be listed.

MLEV=1/2 (GEN) Monitor Level

Specifies MONITOR debugging package options, as follows:

```
 0  ignore all MONITOR directives
 1  process only ASSERT directives, ignore others
 2  process all MONITOR directives.
```

NCF=1/0 (GEN) Negative Constant Folding

Controls whether constant folding with negative results is permitted. Usually enabled, constant folding may need to be disabled for cross-compilations.

NOLOAD=0/1 (ASM S37,S47) Suppress LOAD Module output

Select NOLOAD to suppress LOAD module generation.

NOOPT=0/1 (ASM S37,S47) Suppress optimizations

Select NOOPT to suppress all optimizations. Use of this option not recommended as ASM normally developed and tested with the default optimizations enabled.

NSHEAP=/NSHEAP (ASM S10,S32,S37,S47) Dynamic nameset addressing
----
If a name is given, then the nameset of that name is addressed indirectly so that it can be allocated dynamically.

NSPAGE=0/1 (ASM S32) Align namesets on page boundaries

Select NSPAGE to align all namesets on page boundaries.

OPT=... (ASM S10,S32,S37,S47) ASM Optimization level

Default is OPT=BDFL/ for S32, OPT=DFL/ for S10 and S32.
Specify a character code to select an optimization as follows:

```
  B   branch  (S37,S47 only)
  D   'deferring'
  F   'if'
  L   label
```

Specification of values other than the defaults is not recommended.

PC=/1 (ASM S37,S47) Permanent Constant Register

Controls whether ASM is to permanently dedicate a register to hold a specified constant value or values. Values are specified in hexadecimal and are separated by slash (/). By default, no register is so dedicated.

PDIR=0/1 (GEN) Procedure DIRectory

Controls whether GEN is to produce procedure directory at end of listing. If nonzero, GEN is to produce a directory giving procedure names and page numbers at end of listing. Nonzero values should be used in conjuction with I option in LIST to produce full listing by GEN phase. This option selected if LCR also selected.

PEL=50/10000 (GEN) Parse Error Limit

Specifies the error limit for GEN. If more than the specified number are errors are detected, compilation is abnormally terminated.

PFCC=1/0 (LIB) Print File Carriage Control

Controls whether carriage control information is to be included in PRINT files. Specify zero to suppress carriage control.

PFCL=0/80 (LIB) Print File Characters per Line

Specifies number of characters per line in a PRINT file. Use zero to select the LINESIZE of the file. Small values useful when sending output to a terminal.

PFLL=0/0 (LIB) Print File Line Limit

Specifies print file line limit. If specified, execution abends if more than specified number of lines written. Otherwise, limit derived from PFPL paramater.

PFLP=60/ (LIB) Print File Lines per Page

Specifies number of lines per page.

PFN=filename (LEX) Punch File Name

Specifies name of LEX "punch file" generated if LIST E or D options selected.

PFPL=100/... (LIB) Print File Page Limit

Specifies print file page limit. If specified and PFLL=0 then the print file limit is PFPL*PFLP. An explicit value for PFLL overrides the PFPL specification.

PT=0/1 (GEN checkout) Parse Trace

Controls whether trace listing of parse internal actions is generated.

REP=0/PG (GEN) Report opton

Controls whether GEN produces a ’report’ file. Specify zero to suppress this feature. If selected, the report is written on unit 6. Options are as follows:

```
 P    write line for each procedure giving
        proc.name,proc.type,proc.args
      where proc.name is 1 for subr, 2 for fnct, 3 for prod
 C    write line for each call giving
        caller.name,called.name,number.args
 G    write line for each global variable in form
        var.name,size,dimen,nameset.name,address.offset
 N    write linefor each nameset giving
        nameset.name,nameset.length (in words)
```

If G parameter selected, then both G and N entries generated.
Parm string may have any combination of C P G.

RF=... (LEX,GEN) Reference File Skeleton

Specifies name of cross-reference file used by LEX and GEN.

SC=0/1 (ASM S66) Set Core

Controls whether LDSET PRESET directive is to be generated to preset otherwise uninitialized storage to zero.

SFP=0/1 (GEN) Suppress First Procedure

Nonzero to request that GEN not write to the VOA file the text of the first procedure compiled.

SUSP=0/0 (LEX) List Suspicious Names

Controls whether LEX lists ’suspicious’ names. A name is deemed suspicious if it occurs only once in the source.

TERM=/filename (LIB) TERMinal File Name

Specifies name of file to receive messages sent by LITTLE system to the "terminal". Use L=0,TERM to receive errors only at terminal.

TERMH=1/0 (LIB) Terminal Header

Controls whether a "header" line is sent to the terminal if TERM option used. Specify zero to suppress this header. The default is zero for systems where ’quiet’ operation preferred.

TERMP=>/> (LIB) Terminal Prompt

Specifies prompt string to be written to terminal when requesting input from terminal. Specify zero to suppress the prompt.

TERMLEX=0/1 (LEX)

Controls whether compilation ends after LEX phase. Generally only needed when using SYN program.

TM=/ (GEN) Target Machine

Specifies target machine, and need be specified only when generating code for a machine other than machine at hand.

TMP=xxxxx/ (GEN) Target Machine Parameters

Related to TM, this parameter specifies target machine parameters. Needed only for cross-compilation. Defaults are

```
  S10 3618091818
  S32 3230081616
  S37 3224081616
  S47 3224081616
  S66 6017061113
```

The string specifies five decimal values, each of two digits. In order they are word size (.WS.), pointer size (.PS.), character size (.CS.), length of string length field (.SL.) and length of string origin field (.SO.).

TOKENS=filename (LEX, GEN) Token File

Specifies "token" file written by LEX and read by GEN.

TRACE=/ACDORV (ASM S10, S32, S37, S47 checkout) Trace

Controls whether internal trace listing generated.

UNV=T10MAC/ (ASM S10) Universal file

Specifies file to be used for macro definitions. A SEARCH directive is generated for this file.

UPD=0/1 (LEX) UPD Sequence

Controls whether source is assumed to be in UPD format. If selected, then the first 8 characters are assumed to contain UPD sequence numbers.

VOA=filename (GEN, ASM) VOA file

Specifies file used for intermediate representation writtten by GEN and read by ASM.

ZP=0/1 (S66 ASM) Zero Word to End Parameter List

Controls whether a word of zeros is placed at end of each parameter list. The default not to generate this word. The word is used by FTN procedures (and others) to determine number of arguments actually passed. LITTLE never requires this word be present.

Utilities

The LITTLE system includes various utility programs.

The compiler can produce a cross-reference list. Program REF writes the print file for the cross-reference list.

The compiler uses a table-driven top-down advancing parsing scheme implemented by the program SYN. SYN and GEN have the same structure, and SYN is used to produce the parse tables used by GEN. A separate writeup for SYN exists.

There are various utilities, the most important being the simple document processors LTLDOC and LTLPAD. LTLPAD justifies text, LTLDOC formats it. LTLDOC and LTLPAD are used to list the LITTLE guide, and several other documents of the LITTLE system.

The program UPD is used to maintain source file libraries. A separate writeup exists.

For ease of reference, the program parameters for REF, SYN and UPD are summarized in this section.

The following utility programs are of interest to all sites:

```
     ASCINT - format ASCII files for interchange
     LTLDOC - list little document
     LTLPAD - pad (justify) little document
     MERGER - combine files
     P8020L - process 80/20l format text
     REF    - list cross-reference files
     SHRINK - eliminate blank lines and comments
     SYN    - parser generator
     ULST   - structured list of UPD OPL files
     UPD    - source maintenance program
     UPDFND - extract lines with given string in UPD OPL
```

UTILITY ASCINT - Format ASCII files

Purpose: Format ASCII files for interchange

Parameters:

P	/P	input file
N	/N	outupt file
FF	0/1	form feed option (see below)
W	0/1	zero to read ASCINT, one to write ASCINT

This program permits transmission of full ASCII files using only the 64 character subset of ASCII. Each character is transmitted as two characters. The first character indicates if further translation needed, and second character is data character. The first character is one of following:

blank	no further translation needed
<	(less than) subtract 32 to get true code.
>	(greater than) add 32 to get true code.

For example, word LITTLE in lower case transmitted as >L>I>T>T>L>E.

If FF=1 and W=1 then express form feed by writing line with ’ 1’ in first two columns, otherwise write line with ’ ’ in first two columns. If FF=0 translate formfeed as any other ASCII character.

UTILITY LTLDOC - List LITTLE document

Purpose: Format document

Description:

Input consists of lines with control characters in the first two columns, and text in the remaining seventy columns. Control characters are as follows:

D	document: initialize. should be first control line.
E	eject: set eject flag, do not list text.
P	page: set eject flag, list text.
Q	define symbolic page number.
S	subtitle: use text to define subtitle, set eject flag.
T	title: use text to define main title, set eject flag.
U	underline: list text, then underline it.
Y	enable expansion of symbolic page numbers.
Z	disable expansion of symbolic page numbers.
0	skip line before listing text.
1	same as P.
2..9	start a new page if less than the specified number of lines remain on the current page.

The Q, Y and Z directives permit the construction of a simple table of contents at the end of a document. A symbolic page number consists of a string starting with ’<’ and ending with ’>’. The Q directive associates the current page number with the symbolic page number. Later Y and Z directives determine if symbolic page numbers are to be replaced by page numbers.

UTILITY   LTLPAD - pad (justify) text

Purpose: To collect specified lines into paragraphs and to align the margins of these paragraphs.

Parameters:

P	/P	input file
N	/N	output file
PACK	0/1	if selected, no justifying, and output file contains input text packed into minimum number of lines

Description:

LTLPAD uses directives in column two to mark groups of lines to be padded, i.e., aligned on left and right margins. Command characters are in column two:

    N - begin numeric paragraph.
    X - begin text paragraph.

A paragraph ends with next paragraph begin, blank line, or line with directive in column one. The left margin of a text paragraph is the first nonblank in the opening line. The first line of a numbered paragraph must contain an instance of ’. ’, and the left margin is the first nonblank following this instance.

UTILITY MERGER - combine files

Purpose: Combine several files into one file

Parameters:

N	/N	output file
V	0/1	nonzero for verbose output
A	/	string put after each file name
B	/	string put before each file name

Description:

The standard input file is copied to the specified output file. Lines beginning with ’<INCLUDE’ followed by a file name are replaced by the contents of the specified file. If V option selected, each inclusion is reported by writing a message on the standard output file. Parameters B and A give strings put before and after each file name, respectively.

INCLUDE’s can be nested to a depth of six.

UTILITY P8020L - process 80/20L format ext

Purpose: This program provides a means of transmitting text files using upper-case in a form that indicates the correct upper-lower case representation.

Parameters:

P	/P	input file
N	/N	output file
M	0/1	mode, zero to read 80/20L, one to write 80/20L format

Description:

The 80/20L format permits the distribution of mixed case text using only upper case characters. A line of 80 characters is followed by a line with a shift string of 20 hexadecimal digits which associate a ’shift’ bit with each text character. The shift bit is one to indicate a character should (if possible) be translated to corresponding lower case character by the receiver. Each original line is thus transmitted as two lines.

Each hexadecimal digit in the shift string gives the shift bits for four text characters. The most significant bit in the digit gives the shift bit for the leftmost character. The shift string is in the same order as the text string: column 1 contains the shift bits for columns 1-4, column 20 contains the shift bits fol columns 77-80.

UTILITY REF - List Cross-Reference Files

Purpose: List cross-reference files

Parameters:

REF uses the RF parameter to obtain the names of the cross-reference files. It also checks the LCP and LCS parameters.

Description:

REF merges cross-files produced by LEX and GEN in response to a request for a cross-reference listing. See descriptions of LCR and PDIR program parameters. The cross-reference listing is written to the standard output file.

UTILITY SHRINK - eliminate blank lines and comment

Purpose: eliminate blank lines and comments

Parameters:

P	/P	input file
N	/N	output file
OPT	BC/ABC	options

Description:

The input file is copied to the output file with possible reduction in file size according to options:

A	Replace initial three or more blanks by two blanks
B	Discard blank lines
C	Discard comments (first non-blank is dollar sign)

A line size of 72 characters is assumed.

UTILITY SYN - Parser Generator

Purpose: Generate parser tables from grammar description

Parameters:

ASM	0/Snn	assembler option
SYNASM	filename	generated assembler file
SYNOUT	filename	output file with parse tables
SYNBIN	filename	generated parser binary file
EPC	2/3	entries per generated parse table entry
HA	0/1	on to list syn symbol table
IM	/	immediate macro codes
MACP	PARSE/	prefix for macro names
MEMP	SYN/	prefix for member names
SETL	0/1	SETL option
PT	0/1	parse trace flag

Description:

SYN interprets options LCP, LCS, TERM and TOKENS in the same way as the compiler.

SYN is described in the SYN Reference Manual.

UTILITY ULST - Structured Listing of UPD OPL

Purpose: Provide structured listing of UPD library file, with index to procedure header lines.

Parameters:

P	1/0	Procedure header option
S	0/1	SETL option
C	1/0	Select to list comments
T	1/0	Select to list text (non-comments)
H	/	Header word put at top of every page
B	1/0	Select to list blank lines

Description:

ULST reads the standard input file and writes to the standard output file. The input should be prepared by using UPD and specifying IM, D and NS=L options; for example

```
      LTLUPD P=X.OPL,F,IM,D,NS=L,N=X.TMP
      ULST I=X.TMP,L=X.LST
```

ULST produces a structured listing with a procedure directory. Some duplicate instances of an ident name are eliminated, but the name will appear at least every 10 lines. Pages are numbered, and include time and date of program run. Lines which are probably the last of a procedure are followed followed by blank line and line of asterisks. The listing concludes with a list of each line which is the header of a procedure, followed by a sorted list of procedure names and paged numbers.

The procedure processing requires that the keywords SUBR, FNCT, and FUNC begin in column 7. The END statement must also begin in column 7, and must include SUBR or FNCT for LITTLE procedures.

The ’S’ option allows procedures to begin with the keywords MODULE, DEFINE, DEFINEF, and MACRO. These keywords must begin in column 7.

UTILITY UPD - Source Update Program

Purpose: Identify source lines, provide simple batch editor

Parameters:

P	OLD/	Input file
N	NEW/	Output file
UCS	/	file to receive UPDATE form of corrections
M	2/1	Mode
PS	L/R	Old sequence mode
NS	N/R	New sequence mode
F	EC/F	edit, copy options
D	0/1	copy MEMBER lines
IM	/EC6	identify member option
LO	ACDIPU/ADIPU	listing options

Description:

UPD is described in the UPD Reference Manual.

UTILITY   UPDFND - Find lines in UPD library

Purpose: Extract lines from a UPD library containing
              a specified string, with output in the form
              of a UPD correction set.

Parameters:

P	/P	input file
N	/N	output file
COM	1/0	nonzero to skip $ comments
EXACT	0/1	nonzero for exact case in matching
MOD	MOD/	name of generated correction set

Description:

UPDFND reads a UPD OPL file and extracts all lines containing an instance of a specified string. The output file is in the form of a UPD correction set. The program reads a match string from the standard input file. This string is delimited to permit precise specifications of blanks in the match string; for example

```
  /  LITTLE  /
```

indicates instance of LITTLE with two blanks before and after.

ERRORS

Compilation Errors

Compilation errors cause generation of an error message on the standard output file. Since the compiler runs in three phases, three distinct listing files may be produced. Errors in token formation and macros are detected by the first (LEX) phase. Syntactic errors are detected by the second (GEN) phase. Several program parameters are related to error processing. The LEL and PEL parameters can be used to set error limits for LEX and GEN phases respectively. A choice of low values can force compilation to terminate after the detection of a small number of errors.

The compiler contains a number of tables which in some cases may overflow. Overflow is reported by the generation of an error message, usually containing the internal name of the table, and compilation is abnormally terminated. Errors of this sort require that the program be made "smaller", for example, by dividing a large procedure into several smaller procedures. The error message ’EXPECT LESS THAN 512 PARAMETERS OR DATA STATEMENT ENTRIES also indicates a table overflow, due to combination of length of procedure argument lists and DATA statements.

The compiler reads only the first 72 characters of source lines and provides no special indication of longer lines. Thus program text running past column 72 is skipped, and errors may result.

Execution errors

Errors detected during execution are if possible intercepted by the system. Procecure LTLFIN is called with nonzero first argument to indicate abnormal termination. Error messages are intended to be self-explanatory. Procedure LTLXTR is called to obtain an execution ’traceback’ of active procedures; while not of interest to the normal user, this may assist the system manager in resolving or reporting the problem.

The standard LTLFIN error codes are as follows:

1001	Line limit exceeded
1002	Bad goto index
1003	inclusion depth exceeded, inclusion recursion
1004	bad name for cross-reference file
1005	array index out of range
1006	assertion failed
1007	unable to open standard print file
1008	request for undefined/unsupported function
1009	expiration date passed
1101-1199	math library error
1201-1299	multiword error n-1200
1301-1399	little input/output error n-1300
2000+	env error

EFFICIENCY CONSIDERATIONS

This section describes a number of efficiency considerations which may assist in writing more efficient LITTLE programs.

The CODE option of the LIST directive requests listing of the generated code; this feature also enabled by C option of LIST program parameter. This permits examination of the generated code.

The compiler always allocates storage in units of words; there is no automatic packing of distinct variables. For example,

```
      SIZE A(1),B(1);
```

reserves two words of storage. Any required packing is done using fields.

Assignments to ’large’ variables require in general that all bits be set. A common case is setting a string to the null string; hence,

```
      .F. 1, .SL., S = 0;
```

is usually more efficient than

```
      S = ’’;
```

Experience has revealed a number of ’critical sections’ in the standard library (LIB). Where feasible, these sections admit a more efficient implementation by writing a more efficient version, usually in assembly language. In particular the string search primitives (ANYC, SPNS, etc.) typically are an order of magnitude more efficient when hand-coded, so that differences in the performance of programs using these primitives may differ according as the (hoped for) recoding has been done. Also, it is generally advisable to put the SKIP part of a GET statement at the start, and the SKIP part of a PUT statement at then end; for example,

```
      GET IFILE ,SKIP :A,A(10);
      PUT OFILE :A,A(10) ,SKIP;
```

Also, A format is generally faster than R format. STRING files tend to be slow.

In practice, LITTLE achieves much of its expressive power by systematic use of the macroprocessor features. However, these can lead to a code explosion which merits periodic review. The EXPAND option of the LIST directive requests a listing of the program text after macro expansion. This listing is written to the file specified by the PFN program parameter; indeed, PFN was supported originally for checkout, but has been retained primarily to monitor macro usage. The E option of the LIST compiler program parameter selects EXPAND.

IMPLEMENTATION-DEPENDENT INFORMATION

Substantial work has been done to make the NYU LITTLE
plementation portable so that implementations for different
chines will be compatible. Some features, such as file
mes, command line format, and so forth, are necessarily
chine dependent, and are described in this section.

The following sections describe available implementations, indicating features not supported, additional features supported, implementation restrictions, and demonstrations of control statements required to use the LITTLE system.

CDC 6000 IMPLEMENTATION

Configuration requirements

This implementation runs on the Control Data Corporation 6000 Series hardware. It can be configured for NOS or NOS/BE operating systems, 63 or 64 character set. The LEX, GEN and ASM phases are combined into a single overlay LITTLE which requires about 116000B to run.

Operating Instructions

Needed files are kept in directory LITTLE.
The control statements to compile and execute program on file LITTLEI, with listing, are as follows:

```
      ATTACH,LITTLE,LTLLIB/UN=LITTLE.
      LITTLE. (I=LITTLEI,LIST)
      LGO.
```

Specifying parameters

Program parameters are NOT specified in the usual CDC fashion, but are given in a separate list which follows program name. Parameters are enclosed within parentheses and separated by commas. Note that

```
      LITTLE(I=LITTLEIN)
```

is WRONG. The correct specification is:

```
      LITTLE. (I=LITTLEIN)
```

Character set

DISPLAY code. For 64 set sites, the per-cent character can be used where colon required.

Source program format

The compiler examines only the first 72 columns of each line of LITTLE source text, and lists 90 columns to permit use with UPDATE.

Default file names

Default file names are as follows:

I	INPUT/COMPILE
ILIB	INCLIB/
L	OUTPUT/LIST
TERM	/TERM
TOKENS	TOKENS/
PFN	LEXOUT/
RF	REF0/
VOA	VOA/

Sample control statements

Consider the following program DEMO:

```
  prog demo;
  $ obtain integer given by N parameter, write to file given
  $ by F parameter.  If F not specified, write to unit 4.
    size  num(.ws.);  size  filename(.sds. 10);
    call getipp(num,'N=0/1');  call getspp(filename, 'F=/F');
    file 4 access=put, title=filename;
    put 4 :num,i ,skip;
  end prog;
```

The following text shows how to compile the program, execute it, compile with listing, and compile with cross-reference listing. Text to be entered by the user is at the left in upper case, explanatory comments in lower case are to the right:

```
                              $ standard commands needed to use LITTLE
      ATTACH,LITTLE,LTLLIB/UN=LITTLE.
                              $ compile and execute
      LITTLE. (I=DEMO,B=DEMOLGO)
      DEMOLGO. (N=10,F=DEMOUT)
                              $ compile with listing
      LITTLE. (I=DEMO,LIST)
                              $ compile with listing, cross-reference
      LITTLE. (I=DEMO,LIST,LCR)
      GET,LTLREF/UN=LITTLE.
      LTLREF.
```

S66 Procedures

The following additional procedures are available for this implementation. They are described using the conventions of the section on standard procedures.

```
CALL LRECLSIO(         $       S66 SIO extension
------------------------------------------------
              FN      ,
              RC      WR (WS) );
```

RC is set to zero unless the last I/O operation on file FN was a binary read which encountered an end marker (EOR, EOF or EOI) before all requested information transmitted. In this case, RC is set to the the number of words actually transmitted.

```
CALL WEORSIO(         $       S66 SIO extension
----------------------------------------------
              FN      ,
              RC      WR (WS) );
```

An end of record mark (EOF) is written on file FN.

```
CALL WEOFSIO(         $       S66 SIO extension
----------------------------------------------
              FN      ,
              RC      WR (WS) );
```

An end of file mark (EOF) is written on file FN.

Utilities

The following utility programs are principally of interest to sites using S66, the CDC 6000 series implementation:

```
    BLDLTL - build little overlay input
    P8020C - convert 80/20l format to cdc 6/12 bit
    RFLOVL - set field length of overlay
    TIC    - translate individual character
    UPDBRK - break out comments
    UPDEDT - update/edit interface
    UPDLST - list update compiler file
```

S66 UTILITY P8020C - convert 80/20l to CDC 6/12 bit

Purpose: Convert 80/20l format text to CDC 6/12 bit text so those sites which support 6/12 bit can print LITTLE documents (especially LITTLE guide) in upper-lower case.

Parameters:

P	/P	Input file
N	/N	Output file
M	0/1	Mode, 0 to read 8020/L

Description:

This program is a variant of P8020L (deck P8020L on UTLPL) which reads a 80/20L format file and writes CDC 6/12 bit codes. Lower case letters are represented by writing the escape character 3b’76’ before the upper case code.

This version supports only conversion from 80/20L format to CDC 6/12. the characters circumflex, at sign and colon are translated to 12 bit codes; apostrophe is translated to a 6 bit code.

S66 UTILITY TIC - translate individual character code

Purpose: Translate code for single character, assist translation to and from CDC 63/64 character sets.

Parameters:

P	OLD/C	input file (rewound before and after)
N	NEW/CE	output file (rewound before and after)
PC	00/51	old code to translate
NC	51/00	new code desired
M	63/	translation mode if M=63, set PC=00, NC=51, and so translate to 63 set. if M=64, set PC=51, NC=00, and so translate to 64 set. otherwise, take specified values of PC and NC.
U	0/1	if nonzero, set ’UPDEDT’ mode and blank columns 81-90 of each line changed (this for later use of UPDEDT).
RL	90/130	length of input lines

Description:

The principal use of TIC has been to ’combine’ percents and colons in generating files for export, though in practice this function now done largely by MAKUPL. TIC is not just ’replace’, but is additive in that it permits two character codes to be mapped into one code.

S66 UTILITY UPDBRK - break out comments

Purpose: Identify comments and list them to right of program text.

Parameters:

C	1/0	nonzero to list comments
CC	62/	column to begin comments
DC	1/0	process dollar-sign comments
PL1	1/0	process pl/i / * .. */ comments
FC	0/1	process FORTRAN comments
F	3/	file format: 1 - data, text in columns 1-72. 2 - compile, UPDATE compile file, 1-90. 3 - upd, output of UPDLST program

Description:

UPDBRK identifies the comments in the input file, and writes a file in which the comment text is separated and appears on the right.

S66 UTILITY UPDEDT - UPDATE/EDIT interface

Purpose: To permit use of CIMS Reich/Russell editor to interactively edit an UPDATE compile file and express edit as UPDATE correction set.

Parameters:

C	C/COMPILE	input compile file from UPDATE r
CE	CE/	edited compile file
ID	ID/	output file with correction set

Description:

UPDEDT compares an UPDATE compile file and an edited version of the file to produce an UPDATE correction set (IDENT) expressing the results of the edit.

UPDEDT rewinds all files before and after processing.

UPDEDT assumes that editing done using Reich/Russell editor ’E’ in ’UPDATE’ mode (edit command ’UPD’).

S66 UTILITY UPDLST - neat listing of UPDATE compile fi

Purpose: Provide neat listing of UPDATE compile file which is divided into pages and displays procedure boundaries and location.

Parameters:

UPDSCR		scratch file used by UPDLST
B	1/0	on to list blank lines
C	1/0	on to list comments
H	/	title for listing (seven chars)
P	1/0	on to process procedures, and give list of first line of each procedure
S	0/1	on to process setl procedures
T	1/0	on to list text (non-comments)

Description:

UPDLST provides a structured listing of an UPDATE compile file. Sequence numbers are given on the left (this also assists subsequent use of UPDBRK to list comments to the right).

Lines which are probably the last of a procedure are followed by blank line and line of asterisks. The listing concludes with a list of each line which is the header of a procedure, followed by a sorted list of procedure names and paged numbers.

The procedure processing requires that the keywords SUBR, FNCT, and FUNC begin in column 7. The END statement must also begin in column 7, and must include SUBR or FNCT for LITTLE procedures.

The ’S’ option allows procedures to begin with the keyword ’MODULE’, ’DEFINE’, ’DEFINEF’, and ’MACRO’. These keywords must begin in column 7.

Configuration requirements

This implementation runs on the International Business Machines Corporation System/370 hardware. It is configured for the CMS operating system; it should be usable using OS and its extensions (MVS, etc.), though usage for these systems has not been tested. The LEX, GEN and ASM phases are combined into a single program LITTLE.

Operating Instructions

Needed files are kept on a minidisk of user LITTLE. See the
system manager for information about accessing this disk.
The control statements to compile and execute program
on file LITTLEI LITTLE A1, with listing on file LITTLEI LISTING A1,
are as follows:

```
      LITTLE LITTLEI (LIST RUN
```

The single (required) operand of the LITTLE command is a file identifi
(LITTLEI in the previous example) of the form:

```
      fn ft fm
```

The filename is used as the filename for files accessed by the program. The default ft is the name of the program (LITTLE in this example); the filetype is used as the filetype for the standard input file. The default fm is A1. An additional parameter LDISPMOD may be specified to indicate that the standard output file is to be concatenated with an existing file of the same filename and filetype.

For example, consider

```
      UPD ASM (P=OPL N=LTL F
      LITTLE ASM
```

where UPD is assumed to UPD utility program. The UPD command will do full update of ASM OPL, creating ASM LTL, with standard input taken from ASM UPD.

Specifying parameters

Program parameters are entered as CMS options. However, to overcome the CMS limitation of eight characters per argument, the parameter scanner also does the following:

  Blanks not following an equal sign are taken as commas.
  Blanks just after an equal sign are ignored.

As a result, the following are equivalent:

```
      LITTLE LITTLEI (LIST, H=4
      LITTLE LITTLEI (LIST H=4
      LITTLE LITTLEI (LIST H= 4
```

Note that the (added) parameter RUN causes LITTLE LIB to execute the once it has been compiled. To execute an already compiled, program, use the command:

```
      LITTLE PROG (I=0
```

Character Set

EBCDIC with upper and lower case letters. Lower-case letters are generated only by user request and are not generated during normal operation.

Source program format

The compiler examines only the first 72 columns of each line of LITTLE source text, and lists 80 columns to display any sequence information in positions 73..80.

File names

The file names used by LITTLE (and specified as parameters) are DDNAME If an explicit FILEDEF has been given for the DDNAME, it is used. Consistent with the normal conventions for OS compilers running under CMS, the following DDNAMEs are translated in the absence of a FILEDEF for them:

SYSPRINT	to	LISTING
SYSPUNCH	to	PUNCH
SYSTERM	to	TERM
SYSUTn	to	CMSUTn

If an explicit FILEDEF is specified, it will be used. Otherwise, an implicit FILEDEF will be executed. This implicit FILEDEF will be of the form:

```
      FILEDEF ddname DISK fn ddname A1
```

where fn is the filename of the operand of the command. If no FILEDEF is specified for SYSIN, the following is done:

```
      FILEDEF SYSIN DISK fn ft fm
```

where fn ft fm are the components of the operand with the defaults supplied as described above. There are exceptions to the implicit FILEDEF described above. These are ddnames of TERMx, PRINT and PUNCH. In these cases the device represented by the ddname specified will be used, i.e., the following FILEDEF will be executed:

```
      FILEDEF ddname ddname
```

Default file names are as follows:

I	SYSIN/
ILIB	SYSLIB/
L	SYSPRINT/
TERM	SYSTERM/
TOKENS	SYSUT1/
PFN	SYSPUNCH/
RF	SYSREF(REF0)/
VOA	SYSUT2/

Restrictions

Integer arithmetic restricted to single word operands. Integer arithmetic is correct in the range -2**31+1 to 2**31-1. Real arithmetic restricted to single precision.

Sample control statements

Consider the following program DEMO:

```
  prog demo;
  $ obtain integer given by N parameter, write to file given
  $ by F parameter.  If F not specified, write to unit 4.
    size  num(.ws.);  size  filename(.sds. 10);
    call getipp(num,'N=0/1');  call getspp(filename, 'F=/F');
    file 4 access=put, title=filename;
    put 4 :num,i ,skip;
  end prog;
```

The following text shows how to compile the program, execute it, compile with listing, and compile with cross-reference listing. Text to be entered by the user is at the left in upper case, explanatory comments are at the right in lower case.

```
      [See system manager]    $ standard commands needed to use LITTLE
      LITTLE DEMO (RUN        $ compile and execute
      LITTLE DEMO (LIST       $ compile with listing
      LITTLE DEMO (LIST LCR   $ compile with listing, cross-reference
      DEMO                    $ execute previously compiled program
      DEMO (N=10              $ execute with N set to 10
```

DEC DECSYSTEM-10 IMPLEMENTATION

Configuration requirements

This implementation runs on the Digital Equipment Corporation DECsystem-10 hardware using the TOPS-20 operating system. The implementation should also be usable on TOPS-10 and TENEX, although this has not been verified.

Operating Instructions

At Rutgers, using TOPS-20, LITTLE is currently available on s:<setl.final>. The phases of the compiler should be run in turn. For example, to compile and execute X.LTL, proceed as follows: .s.nf def sys: s:<setl.final>,sys: ltllex(i=x.ltl) ltlgen(i=x.ltl) ltlasm(i=x.ltl) ltllib(i=x.ltl)

Specifying parameters

Program parameters are specified in the usual LITTLE fashion, i.e., as list enclosed in parentheses following program name. The I= parameter should always be specified, even if a dummy file must be created; for example, .s.nf ltlupd(i=foo.ltl) .s.f The maximum length of the parameter list is 120 characters; the maximum length of a single parameter is 30 characters. When running the individual phases separately, the parameter list may be entered on the command line which invokes the program; if not entered, the program will prompt for parameters. For example,

```
        $ run stlprs
```

Note that the parameter line is converted to upper case. This is generally not significant. However, arguments to the procedures GETIPP and GETSPP should thus be specified in upper case. For example,

```
        TRVAL := GETIPP('TRACE=0/1');
```

Character set

Full ASCII character set with upper and lower case letters.

Source program format

The compiler examines only the first 72 columns of each line of LITTLE source text. Instances of horizontal tabs and form feeds in the source are processed in the same way as blanks.

Input/Output

Text lines cannot exceed 132
characters. On text output, trailing blanks and tabs are removed.

Default File Names

Default file names are as follows: I *.LTL/*.LTL (however, see section on program parameters below) ILIB SYSLIB/SYSLIB L

```
*.LST/*.LST S10 I=*.LTL/ L=*.LST/ TERM=TTY:/ ILIB=SYSLIB/ TOKENS=*.TOK
PFN=*.PUN RF=*.RF0 VOA=*.VOA CODE=*.MAC TERM TTY:/
```

Note that * indicates that name given by I= parameter is used to derive filename and extent is then chosen based on at most first three characters of parameter values as shown above.

Restrictions

Integer arithmetic is correct in the range -2**35+1 to 2**35-1.

Sample control statements

Consider the following program DEMO:

```
  prog demo;
  $ obtain integer given by N parameter, write to file given
  $ by F parameter.  If F not specified, write to unit 4.
    size  num(.ws.);  size  filename(.sds. 10);
    call getipp(num,'N=0/1');  call getspp(filename, 'F=/F');
    file 4 access=put, title=filename;
    put 4 :num,i ,skip;
  end prog;
```

The following text shows how to compile the program, execute it, compile with listing, and compile with cross-reference listing. Text to be entered by the user is at the left in upper case, explanatory comments are at the right in lower case.

```
                              $ standard commands needed to use LITTLE
                              $ compile and execute
                              $ compile with listing
                              $ compile with listing, cross-reference
```

DEC VAX IMPLEMENTATION

Configuration requirements

This implementation runs on the Digital Equipment Corporation VAX-11 using the VMS V2 operating system.

Operating Instructions

Symbol definitions and command files for using LITTLE are available in file NYU$LITTLE:LTLDEF.COM. The easiest way to access them is to add

```
      $ @NYU$LITTLE:LTLDEF
```

to your LOGIN.COM file.

Individual phases may be run by using the symbolic names LTLLEX, LTLGEN, and LTLASM. However, for most applications the LTL command is more convenient. LTLDEF also defines various symbols such as LTLPAD, LTLDOC, and so forth; this is done to meet the VMS requirement that programs be declared as ’foreign commands’ if they are to receive the parameters on the command line that invokes them.

Command LTL compiles a LITTLE program; use symbol LTLLIB to LINK it. For example, to compile and link T.LTL to produce T.EXE, do

```
      $ LTL T
      $ LINK T+’LTLLIB
```

The form of the command line is:

```
      $ ltl sourcefile [/option...]
```

Sourcefile is the LITTLE source file. The sourcefile by default has extension "LTL", so specification of this extension is unnecessary. The LTL command permits specification of the program parameters in standard VMS format.

/DLTL
  Controls whether the LITTLE source file is deleted after LEX phase, thus reducing disk requirements when LITTLE source file obtained from UPD library and need not be retained. This switch can not be negated.

/KMAR
  Controls whether generated MAR files are to be kept (not deleted). By default they are deleted.

/KOBJ
  Controls whether generated OBJ files are to be kept (not deleted). By default they are deleted and only the result OBJ file obtained by appending them all together is retained.

/KT32
  Controls whether generated T32 file is kept (not deleted). By default it is deleted.

/LIST[=file]
/NOLIST (D)
  The option /LIST is used to obtain a listing of the source file. If no file is specified, then the input file name together with the default extension "LIS" will be used. If an explicit listing file is given, the extension may be omitted, in which case the default, "LIS", will be used. The option /NOLIST signifies that no listing file is to be generated.

/LO=SQ
  Specifies list options. This is one case where command differs from normal parameter passing in that LITTLE uses LIST for LIST options while VMS uses LIST for listing file name.

/OBJ=file
/NOOBJ
  Specifies name of object file. The default is to generate an object file of the same name as the input with extension OBJ.

/PARM=
  Specifies string to be included in parameter list passed to all compiler phases. If the string begins "NO", then these characters are removed, and the characters "=0" added at the end before passing along the argument. For example, /PARM=NOSUSP is translated to SUSP=0 which disables list of suspicious names.

/TERMGEN
  Controls whether compilaton ends after GEN phase. This switch cannot be negated.

/TERMLEX
  Controls whether compilation ends after LEX phase. This switch cannot be negated.

/T32MLB=file
/T32MLB=NYU$LITTLE:T32.MLB (D)
  Specifies MACRO library containing definitions of the macros used to assemble the generated T32 file.

The translation process from the T32 file to produce the OBJ file is done in a subdirectory of the same name as the input file. If no directory exists, one is created for the compilation and deleted at the end of compilation. If the directory already exists it will be filled with a MAR and OBJ file for each procedure in the input.

Given a T32 file, it is possible to obtain the OBJ file by

```
              $ LTLMAC X
```

Linking the program

Given the OBJ file X.OBJ, the EXE file is obtained by

```
      $ LINK X+'LTLLIB
```

Specifying parameters

Program parameters for the LITTLE command are specified in standard VMS fashion. The maximum length of the parameter list is 300 characters; the maximum length of a single parameter is 63 characters. When running the individual phases separately, the parameter list may be entered on the command line which invokes the program; if not entered, the program will prompt for parameters. For example,

```
      $ LTLLEX I=T.LTL
```

Character set

Full ASCII character set with upper and lower case letters.

Source program format

The compiler examines only the first 72 columns of each line of LITTLE source text. Instances of horizontal tabs and form feeds in the source are processed in the same way as blanks.

Input/Output

Text lines cannot exceed 132 characters. On text output, trailing blanks and tabs are removed. The implementation has default PFPL=0/0 so that print file limits are not enforced by default.

Default file names

Default file names are as follows:

I	SYS$INPUT/
ILIB	SYSLIB.DAT/
L	SYS$OUTPUT/
TERM	SYS$ERROR/
TOKENS	TOKENS.TMP/
PFN	LITTLE.PUN/
RF	LITTLE.RF0/
VOA	VOA.TMP/
CODE	LITTLE.COD/

Sample control statements

Consider the following program DEMO:

```
  prog demo;
  $ obtain integer given by N parameter, write to file given
  $ by F parameter.  If F not specified, write to unit 4.
    size  num(.ws.);  size  filename(.sds. 10);
    call getipp(num,'N=0/1');  call getspp(filename, 'F=/F');
    file 4 access=put, title=filename;
    put 4 :num,i ,skip;
  end prog;
```

The following text shows how to compile the program, execute it, compile with listing, and compile with cross-reference listing. Text to be entered by the user is at the left in upper case, explanatory comments are at the right in lower case.

```
                              $ standard commands needed to use LITTLE
      $ @NYU$LITTLE:LTLDEF
                              $ compile and execute
      $ LTL DEMO              $  assume program in DEMO.LTL
      $ LINK DEMO+'LTLLIB
      $ RUN DEMO
                              $  will get prompt PARAMETERS:
      N=10,F=DEMO.OUT
                              $ compile with listing
      $ LTL T/LIST            $   listing to DEMO.LIS
                              $ compile with listing, cross-reference
      $ LTL T/LIST/LCR
      $ LTLREF L=DEMO.REF     $   cross reference to DEMO.REF
```

S32 VAX/VMS Procedures

The following additional procedures are available for this implementation. They are described using the conventions of the section on standard procedures.

```
CALL EXEC$LI(         $       S32 extension
-------------------------------------------
              S1      RD (.SDS. *),
              ...
              Sk      RD (.SDS. *),
              ...
              Sn      RD (.SDS. *) );
```

The argument strings are executed in order in a subprocess, using the VMS library procedure LIB$EXECUTE_CLI. This procedure accepts a varying number of arguments. The generated subprocess does not inherit the current context, especially that established by the LOGIN.COM file, so symbols used must be defined.

```
CALL SPOLSIO(         $       SIO S32 Only
------------------------------------------
              FN      RD (WS),
              RC      WR (WS) );
```

Send the file to the print queue when it is closed. This is S32 (VAX VMS) extension.

```
CALL SUBMSIO(         $       SIO S32 Only
------------------------------------------
              FN      RD (WS),
              RC      WR (WS) );
```

When the file is closed, submit it to the system batch queue. This is S32 (VAX VMS) extension.

HISTORY

In this section we present a brief history of the development of the language, summarize its current status, and indicate future plans.

The language was designed by Jacob Schwartz in 1968. A first version of the compiler, using the parsing scheme described in Cocke and Schwartz, and providing simple optimization of formally redundant expressions at the basic block level, was coded in LITTLE. LITTLE source was mapped into machine language macros, and debuging of the bootstrap compiler began. These efforts, involving one man working part time, culminated in the summer of 1971 in a bootstrap compiler which ’almost’ compiled itself.

In 1970 Jacob Schwartz designed the SETL language. SETL is a very high level language which has finite sets as its fundamental data type. SETL requires an elaborate run-time library to perform the basic set operations, such as set membership, union, etc. Preliminary experiments indicated that an acceptable level of efficiency for this library, called SRTL, was obtainable only if SRTL were coded at quite a low-level. LITTLE was chosen as the implementation language for SRTL, and the implementation efforts for LITTLE were consequently intensified.

These intensified efforts began in the fall of 1971 with the specification of the LITTLE macro processor. The macro processor was coded in FORTRAN, and added to the compiler as a separate job step. During the winter of 1971, a ’LITTLE to FORTRAN’ translator, based on the newly available macro processor, also coded in FORTRAN, was constructed. As a result, the LITTLE compiler was mapped onto a large (15000 lines) FORTRAN program, which was then debugged. An operational compiler for LITTLE became available in September 1972. At this time, debugging work on SRTL began, and the bootstrapping of the compiler started.

The code generator for LITTLE was completely redesigned in February 1973. In November 1973 the entire compiler was successfully bootstrapped, and the FORTRAN bootstrap compiler was discarded. Language changes were much easier to effect once a LITTLE-written version of the compiler was available.

Work on the construction of a cross-compiler from the S66 compiler for the Honeywell Series 16 minicomputer began in 1974. This compiler has been used to construct a graphics package which uses both S66 and S16. The S16 compiler has also been used to write part of the S16 operating system, as well as an operating system for an experimental computer developed at NYU which uses the S16 as a front end. This compiler is no longer supported, but did provide valuable experience as it represented the first attempt to mount LITTLE on a minicomputer.

Work on the IBM 370 code generator began in 1973. This compiler was successfully bootstrapped to the 370 by the summer of 1975. The code generator has since been rewritten.

The modified S37 code generator was adapted in 1978 to produce source code for a "made-up" machine called T10, and this variant was used to bootstrap LITTLE to the DEC-10. This code generator was extended in early 1979 to produce a code generator for the DEC VAX-11/780. The first VAX (S32) implementation was for the DEC operating system VMS.

In the latter half of 1980, Bell Labs did an implementation for the UNIX operating system for the VAX.

NYU has also done substantial work for the DEC PDP-11. This version has not been exported. It was developed as a result of the replacement of the S16 hardware by the PDP-11. The PDP-11 version (S11) is for the DEC RSX-11M operating system.

Table of Contents