MEMORANDUM

TO: C. Perkins

FROM: D. Knuth

SUBJECT: PRELIMINARY SPECIFICATION OF "SELL"

Attached is a 'complete' report on my plans for the SELL language. I suggest you evaluate the language by trying it out on several example applications. I would also appreciate it if you can check the code on pages 24-31 to see if I have misunderstood GP2 or not.

Any suggestions you have for language improvements are welcome. However, I want to keep the number of reserved words as low as possible; there now are 28 reserved words, and since I intend these to share space with the programmers' identifiers, it will be hard to allow more than about 60 identifiers per program if the number of reserved words increases very much.

The enclosed document only specifies the SELL language, not the compilation technique. Once we can agree on the language, I can get to work on the implementation; I have designed it with implementation plans in mind, but these are still only "in my head." I think it will be possible to write a compiler with two blocks for micro-storage, one block for macro-storage, and one block for dynamic tables. The compiler will be syntax-oriented, and this means its complexity is roughly proportional to the number of syntax rules (see pages 18-21); at about 20 macro-instructions per syntax equation, I think I can fit these into 1024 syllables. I propose writing this part of the compiler myself and testing it out on the B5500 with a simulator for the macro-instructions I specify. Later I can perhaps help with the programming of the micro-operators and the design of some routines to help automate this, but first it seems best to prove out the design of the macro-operations themselves by writing that half of the compiler.

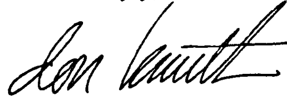
On page 22 I have listed suggested extensions to GP2, including some I didn't think of until after you left. Unfortunately I forgot to discuss "fixups" with you last Saturday, and we will have to work something like that out for the loading routine. (As you can see from the enclosed listing, I have not used that variable-execution-time method of processing fixups which we discussed in June, since that technique will not work reliably with the memory loader.) The LIB and SIB operations mentioned on page 22 would really be better if they could be written LIB 0:3 bwwwwwwww and SIB 0:3 bwwwwwwww instead of LSMR bwwwwwwww followed by LIB 0:3 or SIB 0:3. If that is possible, I would prefer it, although I was afraid you were running out of operation codes.

I think GP2 should set OVERFLOW *or even signal an error somehow* if a number is being printed that has nonzero digits to the left of the field specified. This will help catch programming errors where not enough space has been left in a totals field.

I hope the attached document is sufficiently clear and that it won't take you longer to read it than it took me to write it!

The next move is up to you; we should agree on the language to be implemented before I can go any further.

Cordially,



Donald E. Knuth 10/31/66

P.S. How much of this can I show to people here in Pasadena?

D. Knuth 10/30/66

PRELIMINARY SPECIFICATION OF "SELL"

(SALES ENGINEERS' LITTLE LANGUAGE)

This document has four parts: an example program accompanied by detailed comments about the program; a preliminary draft of a programmer's manual defining the language; a formal syntax which specifies the language a little more precisely; and a hypothetical compiler output for the first example program translated into GP2 language, according to the compiling mechanisms presently envisioned.

The reader is advised to look through the example program first, following at the same time the comments which follow that program. This gives the flavor of the language, and a general idea of its scope. Then read the language definition. The syntax may be used by those who understand such formal definitions; it is included here primarily to serve later as a guide for writing the compiler which will be syntax-oriented. The final section which gives translation into GP2 is also a guide to the compiler design, as well as showing some new operations which ought to be included with GP2 in connection with SELL.

The language has been designed specifically to make it easily learned by a man who knows FORTRAN or ALGOL or a similar language, and at the same time it has been designed so it can be translated into GP2 by a small compiler. Present indications are that the language is not too complicated for a small compiler to handle, but it will be a tight fit; if necessary, some features of the language would have to be dropped, in the following order: Some of the GP2 mask options; the loop statements; subscripted parameters.

Example Program.

The following is an example of a SELL program based on an actual payroll application. The example illustrates most of the features of SELL; unfortunately it assumes a 20" platen instead of a 15" platen, but this should not affect its usefulness as an example of the language.

```
line #        (the program may be broken into lines in any desired manner)
 1.   ◇PAYROLL, MILWAUKEE LUTHERAN HIGH SCHOOL.
 2.   NUMERIC DATA(11) = (REGULAR, OTHER, HOUSING, NFICA, FICA,
 3.       PENSION, CTAX, STAX, INS, MISC, NETPAY).
 4.   NUMERIC TOTALS(13) = (CKNO, TOT(11), W2).
 5.   NUMERIC GTOT(11), TNFICA, TFICA, TCTAX, TSTAX, PROOF, FICAMAX, I.
 6.   ALPHA(7)DATE.
 7.   BEGIN ROUTINE TTD.
 8.   ◇ENTER AND CHECK TOTALS-TO-DATE.
 9.   S1: ENTER(7) TNFICA; A4,A5,B0.
10.       ENTER(7)TFICA; B1.  ENTER(7)TCTAX; B2.  ENTER(7)TSTAX; B3.
11.       ENTER(7)PROOF; B4.
12.   IF PROOF = TNFICA+TFICA-TCTAX-TSTAX, GO TO OKAY.
13.   ALARM. GO TO S1.
14.   OKAY: OPEN 0.
15.   END ROUTINE TTD.
16.   ◇START OF PROGRAM.
17.   KEY ROUTINE A0: VOID,  A1: BACK,  A2: W2FORM,  A3: CCNO,
18.                     A4: SUBTOT, A5: GRANDT.
19.   27 PRINT "PAYROLL".
20.   CLEAR TOTALS.
21.   37 TYPE (7) DATE; A2.
22.   46 ENTER (6) CKNO.  PRINT(ZZZZZZ)CKNO.
23.   71 ENTER (6) FICAMAX; A0.     PRINT (ZZZZ.¢¢) FICAMAX.
24.   ADVANCE 1.
25.   GMAIN: CLEAR GTOT.
26.   MAIN: ## 87 CALL TTD.
27.   CLEAR DATA.
28.   # 87 ENTER(5)REGULAR; A1,B5.   PRINT(ZZZ.¢¢-)REGULAR.
29.   * ADD REGULAR TO GTOT(1).
30.   IF KEY(1), GO TO S2.  IF KEY(4), GO TO S3.
```

```
31.  # 94 ENTER(5) OTHER; A0,A1,B5.  PRINT(ZZZ.¢¢-) OTHER.
32.  * ADD OTHER TO GTOT(2).
33.  IF KEY(1), GO TO S2.  IF KEY(4), GO TO S3.
34.  # 101 ENTER(4) HOUSING; A0,A1,B5.  PRINT(ZZ.¢¢-)HOUSING.
35.  * ADD HOUSING TO GTOT(3).
36.  IF KEY(1), GO TO S2.
37.  S3: # 107 ENTER(5) NFICA; A0,A1,B5.
38.  S2: SET NETPAY = REGULAR + OTHER + HOUSING.
39.  IF KEY(2), SET NFICA = NETPAY.
40.  SET FICA = NETPAY - NFICA.
41.  UNLESS FICAMAX - FICA - TFICA NEGATIVE, GO TO S4.
42.  ADD TFICA - FICAMAX TO FICA.  SET NFICA = NETPAY - FICA.
43.  S4: 107 PRINT(ZZZ.¢¢-) NFICA.
44.  * ADD NFICA TO GTOT(4).
45.  114 PRINT(ZZZ.¢¢-) FICA.
46.  * ADD FICA TO GTOT(5).
47.  BEGIN ROUTINE DEDUCT(P,X,J).
48.  (P) ENTER(5)X; A0,A1,B6.
49.  PRINT (ZZZ¢¢-) X.
50.  * SUBTRACT X FROM NETPAY.  * ADD X TO GTOT(J).
51.  IF KEY(2), GO TO PRNET.
52.  END ROUTINE DEDUCT.
53.  # CALL DEDUCT(120, PENSION, 6).
54.  # CALL DEDUCT(127, CTAX, 7).
55.  # CALL DEDUCT(133, STAX, 8).
56.  # CALL DEDUCT(139, INS, 9).
57.  # CALL DEDUCT(145, MISC, 10).
58.  PRNET: 151 PRINT (ZZZ¢¢-) NETPAY.
59.  IF NETPAY NEGATIVE, ALARM.
60.  * ADD NETPAY TO GTOT(11).
61.  ADD NFICA TO TNFICA. 159 PRINT(ZZZZZ.¢¢)TNFICA.
62.  ADD FICA  TO TFICA.  168 PRINT(ZZZZZ.¢¢)TFICA.
63.  ADD CTAX  TO TCTAX.  176 PRINT(ZZZZZ.¢¢)TCTAX.
64.  ADD STAX  TO TSTAX.  184 PRINT(ZZZZZ.¢¢)TSTAX.
65.  191 PRINT(ZZZZ.DDD) TNFICA+TFICA-TCTAX-TSTAX.
```

```
66.  ◇ NOW OPERATOR TYPES NAME ON CHECK.
67.  0 TYPE(30); A0,A3.
68.  S5: 37 PRINT DATE.  46 PRINT(ZZZZZZ)CKNO.
69.  ADD 1 TO CKNO.
70.  60 PRINT(ZZZ.¢¢)NETPAY.  71 PRINT(ZZZ.¢¢)NETPAY.
71.  80 PRINT DATE.
72.  ADVANCE 1. OPEN 0. GO TO MAIN.
73.  VOID: 31 PRINT "**VOID".
74.  BEGIN LOOP I FROM 0 TO 154 BY 7.
75.  (I+37) PRINT "*******".
76.  END LOOP I.  GO TO ##.
77.  BACK: GO TO #.
78.  CCNO: 46 ENTER(6) CKNO.  GO TO S5.
79.  BEGIN ROUTINE PTOT(A).
80.  ADVANCE 3.  10 PRINT A.
81.  BEGIN LOOP I FROM 1 TO 11 BY 2.
82.  (83 + 7*I) PRINT (ZZZZZZ.¢¢-) GTOT(I).
83.  END LOOP I.  ADVANCE 1.
84.  BEGIN LOOP I FROM 2 TO 10 BY 2.
85.  (83 + 7*I) PRINT (ZZZZZZ.¢¢-) GTOT(I).
86.  END LOOP I.  ADVANCE 3.
87.  END ROUTINE PTOT.
88.  GRANDT: CALL PTOT("DEPARTMENT TOTALS").
89.  BEGIN LOOP I FROM 1 TO 11.  ADD GTOT(I) TO TOT(I).  END LOOP I.
90.  IF W2 = 0, GO TO GMAIN.  GO TO W2FORM.
91.  SUBTOT: BEGIN LOOP I FROM 1 TO 11.
92.  ADD TOT(I) TO GTOT(I).  END LOOP I.
93.  CALL PTOT("SUBTOTALS").
94.  BEGIN LOOP I FROM 1 TO 11.
95.  SUBTRACT TOT(I) FROM GTOT(I).  END LOOP I.
96.  IF W2 = 0, GO TO MAIN.  GO TO W2F.
```

```
97.   W2FORM: CLEAR GTOT. . .  SET W2 = 1.
98.   W2F: ## 87 CALL TTD.
99.   FICA = TFICA * 3625 / 100000.
100.  58 ENTER(7) OTHER.
101.  34 PRINT(#ZZ,ZZZ.¢¢)TCTAX-FICA. *ADD TCTAX-FICA TO GTOT(1).
102.  SET NETPAY = TFICA + TNFICA - OTHER.
103.  46 PRINT(#ZZ,ZZZ.¢¢)NETPAY.  *ADD NETPAY TO GTOT(2).
104.  57 PRINT(#ZZ,ZZZ.¢¢)OTHER.  *ADD OTHER TO GTOT(3).
105.  70 PRINT(#ZZZ.¢¢)FICA.  *ADD FICA TO GTOT(4).
106.  80 PRINT(#Z,ZZZ.¢¢) TFICA.  *ADD TFICA TO GTOT(5).
107.  95 PRINT(#Z,ZZZ.¢¢)TSTAX.  *ADD TSTAX TO GTOT(6).
108.  92 TYPE(1); A1,B7.  ◇OPERATOR TYPES IN STATUS CODE.
109.  OPEN 0.  GO TO W2F.
110.  END.
```

should really type in at begin

Comments on the program, by line number.

1. The ◇ symbol means the following remarks (up to the next period) are comments which do not explicitly affect the behavior of the program.

2-3. These two lines say that DATA(1), DATA(2), DATA(3), ..., DATA(11) are numeric variables, which are equivalent to the numeric variables given the respective names REGULAR, OTHER, HOUSING, ..., NETPAY.

4,5. Line 5 says GTOT(1), GTOT(2), GTOT(3), ..., GTOT(11), TNFICA, TFICA, TCTAX, TSTAX, PROOF, FICAMAX, I are the names of numeric variables. Line 4 says CKNO is equivalent to TOTALS(1), TOT(1) is equivalent to TOTALS(2), TOT(2) is equivalent to TOTALS(3), ..., TOT(11) is equivalent to TOTALS(12), and W2 is equivalent to TOTALS(13).

6. This says DATE is the name of a alphabetic variable whose value may contain up to 7 characters at a time.

7-15. Lines 7 through 15 represent the "TTD" subroutine, which controls the operator's entry of totals from an employee's earnings record. This subroutine is explained further in the discussion of line 26 below.

17,18. These lines say that when key A0 is subsequently enabled and depressed, the program is supposed to go to VOID (i.e., the routine which begins at line 73); when key A1 is enabled and depressed, we should go to BACK (line 77 of this program); etc. When keys not mentioned -- in this case the keys A6, A7, and B0-B7 -- are enabled and depressed, nothing happens.

19. This statement says the heading "PAYROLL" is to be typed starting in column 27 of the current line.

20. Here we set the entire array of TOTALS, which according to line 4 includes CKNO, W2, and the variables TOT(1) through TOT(11), to zero.

21. The print head on the typewriter is moved to column 37 and the operator is to type in up to seven characters. The result of his typing is printed on the form and also entered as the value of the alphabetic variable DATE. If the operator depresses program key A2, go to the routine for W2 forms (see line 17).

22. The print head is moved to column 46. The operator types in a number, up to six digits long, and this number is stored as the value of the numeric variable CKNO. (It represents the number to be typed on the first check.) This value is also printed (starting at column 46, since the ENTER instruction causes no print action), with leading zeroes suppressed.

23. Line 23 is similar to line 22, except the ENTER statement also allows program key A0 to be enabled. If the operator presses A0, we go to VOID (see line 17) and the current line is crossed out and the program starts over again at its beginning. If the operator completes his entry in the normal manner by pressing a motor bar, however, the value of FICAMAX is set (this would be $6600.00 in 1966) and printed out with two decimal places and zero suppression up to the decimal point.

24. The platen feeds one line. (No split platen is being used here.)

25. GMAIN is a label denoting a reference point in the program (see for example line 90 where the statement "GO TO GMAIN" would take the action back from line 90 to this point). "CLEAR GTOT" has a meaning similar to line 20, namely that GTOT(1) through GTOT(11) are cleared to zero.

26. This is reference point MAIN, the main part of the iteration. The two # signs mean it is a rerun point; a statement "GO TO ##" as in line 76 means "return to the last rerun point encountered," undoing the effects of certain critical actions marked by * (for example, resetting the total which may have been updated in line 29). The statement "CALL TTD" activates the TTD subroutine which appears on lines 7-15. This subroutine does the following:

    (a) lets the operator enter "TNFICA", the total non-FICA wages to date shown on the employee's earning record. (Line 9) At this point, program keys A4 and A5 are enabled, to allow the opprator to get grand totals or subtotals instead. Also key B0 is enabled, merely to light the light as a

signal to the operator what we want him to do.

(b) asks the operator to enter the other totals, TFICA, TCTAX, TSTAX, and also the PROOF figure printed.

(c) checks to see if the entries are consistent (line 12)

(d) otherwise rings the bell, and returns to give him another chance to make the entries. (line 13)

(e) When valid entries have been entered, the carriage opens (line 14; OPEN 0 means the carriage will advance 0 lines when it is subsequently closed).

This ends the subroutine. The operator at this point is to put the employee's earnings record in the machine (over the payroll journal which is already in the platen) and it slides down (front feed) into position according to perforations. The operator also puts a check into the carriage, with carbon on the back of the check(to print earnings and deductions on the check as well as the record sheet and the journal).

27. The values of REGULAR, OTHER, ..., NETPAY are set to zero.

28,29. The operator enters regular wages, and this amount is added to the current grandtotal GTOT(1).

30. If motor bar I was used, the program goes to S2; if motor bar IIII was used, we go to S4. Otherwise, continue in sequence.

31-36. Similarly, OTHER earnings and HOUSING allowance may be entered.

37-42. The earnings are split into FICA and non-FICA depending on what the operator enters specifically as non-FICA and any excess of accumulated wages over the $6600 figure. The meaning of line 41 is essentially to skip line 42 unless FICA plus TFICA exceeds FICAMAX.

43-46. The computed values of FICA and NFICA are now printed on the form.

47-52. This is a subroutine with three parameters, P, X, and J. The function of the subroutine is to position the carrier at column P, then ask the operator to enter a value for variable X, and to print the value, and to deduct this value from NETPAY, but to add it to the total GTOT(J). The two actions on line 50 are marked with a * so that they may be undone later (if necessary) by using a "GO TO #" or a "GO TO ##" operation. If the operator has used motor bar 2, it is the last deduction and we are to go to PRNET(line 58).

53-57. These lines each activate the DEDUCT subroutine with a different set of values for the parameters P, X, and J. The # in each line marks it as a secondary rerun point.

58-65 The machine prints the computed net pay (rings the bell if it is negative), then prints new totals-to-date and proof on the employee earnings record. The proof is printed with three decimal places, for obscure reason. (See the mask ZZZZ.DDD in line 65.)

66-67. The carrier now moves all the way left to column 0. The operator may push program key A0 to void the line, or after his typein he may push A3 to reset the check number if by mishap it has gotten out of step with the number printed on the check itself. In a normal case, the operator merely types the man's name on the check at this point, up to 30 characters.

68-71. The machine prints the date and number on the check; updates the number for the next check; and prints the amount NETPAY twice (once for protection). Then in column 80 the date is printed again so it appears on the check stub and the employee record sheet.

72. The machine advances one line and opens the carriage, then goes back for more.

73-76. The voiding routine prints asterisks, seven at a time, in columns starting 37, 44, 51, ..., 154+37= 191; see lines 74-76. Then the statement "GO TO ##" returns to the last major rerun point and resets the actions of the all statements marked with * which have been executed since the most recent rerun point was encountered.

77. The backup routine (which corresponds to the frequently enabled program key A1) simply goes back to the 2nd most recent minor rerun point and resets the actions of *-ed statements.

78. Allows typein of a new check number (see line 68).

79-87 This is a subroutine which prints the values of GTOT(1) through GTOT(11) in staggered form appearing essentially like this:

```
   1      3      5      7      9      11
 xxxxx  xxxxx  xxxxx  xxxxx  xxxxx  xxxxx
     2      4      6      8      10
   xxxxx  xxxxx  xxxxx  xxxxx  xxxxx
```

88-90 Prints out grand totals for department, adds these to accumulated totals, and clears out grand total registers.

91-96 Prints out current value of grand-totals plus accumulated-totals in each category.

97-109 A routine to print out W2 forms, using features borrowed from the payroll check routine (e.g. the TTD subroutine, and the procedures for printing grand totals and subtotals).

110. Designates the end of the SELL program.

Definition of SELL

The following sections give a reasonably complete definition of the allowable constructions of the SELL language and their meaning. Reference to the previous example should make it more easy to understand the general rules given below.

1. Identifiers. An identifier is a sequence of one or more letters and/or digits, starting with a letter. If the first 6 characters of two identifiers are the same, the compiler might regard them as identical. According to rules given below each identifier serves as the name of a variable, a label, a subroutine, a parameter, or a word with special meaning in SELL. No identifier may be used for more than one purpose in a program.

2. Variables. A variable is either NUMERIC (having a signed 15-digit decimal plus 3 flag bits), or ALPHA (having a string of characters as its value). The first appearance of the name of a variable in a program must be in a "declaration"; for example, the sequence of declarations

```
NUMERIC A, B, CAT.
ALPHA (10) X,Y.
```

declares that A, B, and CAT are NUMERIC variables; X and Y are ALPHA variables whose length is at most 10 characters.

Variables may be "subscripted" thus:

```
NUMERIC M(5), N(3).
```

This means there are five NUMERIC variables M(1),M(2),M(3),M(4),M(5) and three others N(1),N(2),N(3). ALPHA variables whose length is eight or higher may not be subscripted.

3. Equivalences. It is possible to give more than one name to the same variable, by using constructions like the following:

```
NUMERIC N(3) = (I,J,K), P = Q = R.
```

This declares four NUMERIC Variables: N(1) which is also called I; N(2) which is also called J; N(3) which is also called K; and P which is also called Q or R. This multiple naming saves memory space inside the machine and allows a program to show more clearly what it is trying to do. As a more complicated use of equivalences, consider the declaration

```
NUMERIC N(4) =(I, J(2) = (P,R), K = L).
```

Here N(1) = I, N(2) = J(1) = P, N(3) = J(2) = R, and N(4) = K = L.

4. Numeric expressions. A numeric expression denotes a numeric value composed of one or more other numeric values by means of arithmetic operations. A primary is the simplest kind of numeric expression: A primary may be a numeric variable, having the form "X" for un~~subscripted~~ variables, or the form "X(numeric expression)" for ~~subscript~~ed variables, where X has appeared in a NUMERIC declaration, or X is a parameter to a subroutine; or a primary may be a numeric constant, 1 to 15 decimal digits. The numeric expression in ~~a subscript~~ must not ~~contain subscripted variables or any multiplication~~ or ~~division operations~~.

For example, "B", "25", and "M(B+25)" are primaries.

A term is the next most complicated type of numeric expression. A term may be simply a primary or it may have one of the forms

primary * primary	(multiplication)
primary / primary	(division)
primary * primary / 10^k	(scaled multiplication)
~~10^k *~~ primary / primary	(scaled division)

The result of division is truncated by throwing away the remainder. For example, 34 * 5 = 170, 34 * 5 / 100 = 1, 34 / 8 = 4, 10 * 34 / 8 = 42.

A numeric expression is either a term or the sum or difference of terms. For example, A + 100 * B / C - D - E * F(I+1) is a (rather complicated) numeric expression.

The above rules define a restricted class of numeric expressions, prohibiting constructions which are allowed in FORTRAN such as the use of parentheses in "(A + B)/C", and prohibiting repeated multiplications and divisions as in "A * B * C". Only sums and differences of terms having the simple forms listed may appear.

5. Alphameric expressions. An alphameric expression is either an alpha-variable (i.e. a variable which has previously appeared in an ALPHA declaration), or a parameter to a subroutine, or an alphameric constant. The latter consists of a quote mark, followed by one or more characters (of which only the first may be a quote mark), followed by another quote mark. Examples of alphameric constants are "HELLO " and ""...GOODBYE".

6. Statements. A SELL program is a sequence of declarations and statements, with optional comments, loops, and/or labels interspersed. A comment is distinguished by a "lozenge" ◇ as its first character, and it has no direct influence on the behavior of the program. A loop is explained below. A label is a label-identifier followed by a colon, and it serves to identify the place it appears in the program. For example, consider the following sequence:

```
READ: 20 ENTER(6) DATA.
          IF DATA NEGATIVE, GO TO READ.
```

The label "READ:" is here followed by a statement which directs the typewriter carrier to move to column position 20 and wait for the operator to enter an integer number of 6 digits or less; the number thus entered is stored as the value of the numeric variable, DATA. The following statement says if this value is negative, we should go back to the first statement and wait for another entry.

7. SET statements. The statement

SET variable = expression

means the value of the variable is to be set equal to the value of the expression. The variable and the expression must both be numeric, or both alphabetic, with the length of the variable greater than or equal to the length of the expression.

Examples:

```
SET FICA = RATE * WAGE.
SET MESSAGE = "HELP?".
SET X(I) = 1.
SET R = 0 - R.
SET N = N + 1.
```

The last statement may also be written "ADD 1 TO N." The statement "ADD numeric expression TO numeric variable" is equivalent to "SET numeric variable = numeric variable + numeric expression". Similarly, "SUBTRACT n.e. FROM n.v." is equivalent to "n.v. = n.v. - n.e."

8. KEY ROUTINE statements. The statement

KEY ROUTINE A0: ROUT1, A1: ROUT2, B4: ROUT3

means that subsequent ENTER or TYPE statements, in which the program keys A0, A1, or B4 have been enabled, will go to labels ROUT1, ROUT2, or ROUT3 respectively, if the keyboard input terminates with that key. If other keys are enabled, they serve only as a light to tell the operator what kind of input is expected; no special action occurs if other enabled keys are depressed. The keys A0, A1, ..., A7, B0, ..., B7 specified in a KEY ROUTINE statement must appear in ascending order.

9. ENTER statements. The statement

ENTER (s, t) numeric variable

means a numeric keyboard entry is expected, with at most s digits to the left of the decimal point and t digits to the right; the value entered is multiplied by 10^t and stored as the value of the numeric variable. Here s and t are numeric constants with $s + t \le 15$; if $t = 0$, the ",t" may be omitted.

An ENTER statement may be followed by a semicolon and a list of program key designators. For example,

ENTER (5,7) COST; A3,B0,B2.

means a keyboard entry of the form xxxxx.yyyyyyy is to be entered (and stored into the variable COST as the integer xxxxxyyyyyyy), with three program keys A3, B0, and B2 enabled. The program keys listed must appear in order.

10. TYPE statements. The statement

TYPE(s)

means an alphabetc keyboard entry is expected, with at most s characters. This entry is typed starting at the current position on the line. (Here s is a numeric constant.)

The statement

TYPE(s) aphpha variable

also sets the value of the alpha variable to the quantity typed in.

Either form of TYPE statement may be followed by a list of program keys to be enabled, as in the ENTER statement.

11. PRINT statements. The statement

PRINT alphabetic expression

prints the value of the alphabetic expression, starting at the current position on the line. The statement

PRINT alphabetic expression LEFT

prints the value backwards going to the left instead of to the right, starting at the current position on the line minus two. (See the examples in section 13 below.)

The statement

PRINT (mask) numeric expression

prints the value of the numeric expression using the mask, as in GP3. The mask here should be shortened to the desired number of characters to be considered; for example,

PRINT (ZZD.DD) X

prints the value of X using the GP3 mask"IIIIIIIIIIZZD.DD". In addition the mask in SELL may be preceded by "$" and/or "#" to denote a floating dollar sign or the suppression of punctuation; and it may be followed by "-" or "cR" to denote a character to be printed only if the value of the expression is negative. Negative values appear in red ribbon. The mask char E should not be used.

12. GO TO statements. The statement

GO TO ROUT1

where ROUT1 is a label identifier means execution of the program (which normally proceeds sequentially from statement to statment in the order written) should continue now from the place in the program labelled"ROUT1:".

13. Carrier positioning. Any statement may be preceded by a numeric constant or by a parenthesized numeric expression, which indicates the desired position of the typewriter print mechanism. For example,

```
10 PRINT "KAOS"
```

prints a K in column position 10, A in position 11, O in 12, S in 13, and leaves the typewriter ready to print next in column 14.

```
10 PRINT "KAOS" LEFT
```

prints a K in column 8, A in 7, O in 6, S in 5, and is ready to print next in column 6. The following sequence of statements may be used to print a value of the form xxx.xx, underline it, and put the total yyyyy.yy in the line below, so that the page looks like this:

```
  xxx.xx
yyyyy.yy
```

```
10 PRINT(DDD.DD) X.
17 PRINT "______" LEFT.  ADVANCE 1.
 8 PRINT(DDDDD.DD) Y.
```

14. Conditional statements. Any statement may be preceded by a condition prefix which makes the effect of the statement optional. For example,

```
IF X = Y, SET X = Z
```

means X is set to Z if it equals Y;

```
UNLESS X = Y, SET X = Z
```

means X is set to Z if it does not equal Y.

A condition prefix in general consists of the word IF or UNLESS, followed by a condition, followed by a comma. A condition may take the following forms:

numeric expression = numeric expression	[true if expressions are equal
numeric expression NEGATIVE	[true if expression is negative]
numeric expression CODE(2)	[true if C flag of expression is on]
numeric expression CODE(3)	[true if M flag of expression is on]
KEY(d)	[d = 1,2,3,or 4; true if d is number of last motor bar used by opprator]
OVERFLOW	[true if arithmetic overflow has occurred
PAGE OVERFLOW	[true if ~~current line number~~ last ADVANCE operation exceeds page size]

The C- and M-flags of a numeric value are set if the keyboard operator includes a "per hundred" or "per thousand" indicator during an ENTRY statement. These flags stay with the numeric value until it is used in any arithmetic operation, and at that time they disappear.

15. Protection of variables. The SELL language includes a general facility for recovering from operator errors. Any statement may be preceded by "#" or by "##", and any ~~numeric~~ SET, ADD, or SUBTRACT statement may be preceded by "*". This means information is automatically saved so that all effects of these SET, ADD, and SUBTRACT statements may be cancelled if later events should make this desirable.

The statement

```
GO TO ##
```

means GO TO the last-executed statement which was tagged "##", and restore the values of all variables that were changed by starred SET, ADD or SUBTRACT statements that occurred since the execution of that statement. (A "##" is implied at the beginning of every program.)

The statement

```
GO TO #
```

means GO TO the second last executed statement which was tagged "#", or to the last "##" statement which was executed, whichever occurred most recently. Variables whose values changed in the intervening time are restored as in GO TO ##.

For example, consider the following sequence:

```
## ENTER(5) X;A2.
# ENTER(6) Y;A1.
* ADD Y TO X.
20 PRINT(ZZZ,ZZD)Y.
* SET Z = 2 * X.
# ENTER(5)' C.
IF KEY(4), GO TO #.
IF C NEGATIVE, GO TO ##.
```

First the keyboard operator types in a 5-digit value, X, then a 6-digit value, Y. After setting X = X+Y, printing Y, setting Z = 2*X, we get to another ENTER statement, so the operator is to enter another 5-digit value, C. If this value is entered with motor bar IIII, we "GO TO #", which means in this case we go back to the "ENTER(6) Y" statement and we restore the previous values of Z and X. Presumably

the operator pressed motor bar IIII because he realized he had mis-entered the value of Y after seeing it printed. At this point he may either enter a new value of Y, or he may press the enabled program key A1 (which might cross out the preceding Y value and advance one line and then "GO TO #" again which would take the program all the way back to the first statement).

In the above example, if motor bar 4 was not used but a negative value ("reverse entry") was entered for C, the statement "GO TO ##" means the values of X and Z are restored and we go back to the original ENTER statement for X. Note that "GO TO ##" is equivalent to repeatedly executing "GO TO #" until reaching the most recently executed ## statement.

This feature of SELL requires memory space to save the previous values of the variables and the locations of # and ## statements. The SELL compiler will indicate to the programmer how much space is available for this backup storage, and it is important that the programmer check this number to make sure enough space is left. (For each occurrence of a * between two ## statements, while the program is running, 1.25 words of memory are used; and for each occurrence of a # between two ## statements while the program is running, .25 words are used.)

16. Subroutines. The declaration

BEGIN ROUTINE NAME

or BEGIN ROUTINE NAME($P_1,\ldots,P_n$)

marks the beginning of a subroutine called NAME, having parameters $P_1,\ldots,P_n$. This subroutine ends with the declaration

END ROUTINE NAME

which follows. An example appears in section 20.

A subroutine is a piece of program which does no action until it is "called." The statement

CALL NAME

or CALL NAME($A_1,\ldots,A_n$)

activates the subroutine, "substituting" the arguments $A_1,\ldots,A_n$ for the parameters $P_1,\ldots,P_n$ during the performance of the subroutine. Arguments may be variables, constants, labels, or an identifier corresponding to a subscripted variable.

The action of a subroutine is terminated either by a GO TO statement that leads out of the subroutine, or by coming to the END ROUTINE statement (in which case we GO TO the statement following the CALL). A GO TO statement may not lead from outside a subroutine to within that subroutine, or from one subroutine out into another subroutine which called it. A subroutine with parameters may not be CALLed recursively (i.e. while it is already CALLed).

Limit of 4 deep.

17. Loops. The construction

```
        BEGIN LOOP V FROM r TO s BY t
        ........ (any sequence of statements, in which all loops
                    and subroutines are closed by END)
        END LOOP V
```

is an abbreviation for the sequence of statements

```
        SET V = r - t.
    L1: ADD t TO V.
        IF s - V NEGATIVE, GO TO L2.
        .........
        GO TO L1.
    L2:
```

Here r, s, t are numeric constants, and V is a numeric variable. If t = 1, the "BY t" may be omitted.

18. Forms control. The following statements are provided for forms control:

ADVANCE LEFT n	feed left platen n lines
ADVANCE RIGHT n	feed right platen n lines
ADVANCE LEFT TO n	advance left platen to n-th line of page
ADVANCE RIGHT TO n	advance right platen to n-th line of page
PAGE LEFT n	size of left page is n lines.
PAGE RIGHT n	size of right page is n lines.
OPEN n	open carriage and after it is later closed advance n lines.

The word LEFT may be omitted (since the platen may be combined and in this case LEFT is implied).

19. Miscellaneous statements.

ALARM	rings the bell
CLEAR numeric-identifier	sets the variable (and if it is subscripted, all of the entries) to zero.

Note: The statement CLEAR X may not be used when X is a subroutine parameter.

CODE($b_{15}b_{14}b_{13}b_{12}b_{11}b_{10}b_9b_8, b_7b_6b_5b_4b_3b_2b_1b_0$)

CODE($b_{15}b_{14}b_{13}b_{12}b_{11}b_{10}b_9b_8$, V)

These two statements perform the actions of a GP2 instruction whose binary form is "$b_{15}\cdots b_0$", in the first case, or "$b_{15}\cdots b_8$00000000 (address of V)" in the second case. The b's are zero or one; V is a variable. If V is a parameter or if V is subscripted, SELL might use the accumulator and might modify this instruction with index register 4 and 1.

The purpose of the CODE instruction is to leave SELL open-ended enough to incorporate any special GP2 features that might exist in non-standard versions, or to do special effects with GP2 which are not available in SELL, without writing a whole program in GP2.

<u>20. A SELL program.</u> A SELL program is a list of statements, declarations, comments, loops, (each followed by a period) or labels (which end with a colon), followed finally by "END." to mark the end.

Here is a short SELL program which makes the machine behave much like a simple adding machine:

```
NUMERIC X,TOTAL.
KEY ROUTINE A0:SUBTOT, A1:TOT.
ZERO: SET TOTAL = 0.
ENT: ENTER(15)X; A0,A1.
ADVANCE 1.
10 PRINT(ZZZZZZZZZZZZZ.DD-)X.
UNLESS KEY(2), ADD X TO TOTAL.
IF KEY(2), 30 PRINT "#".
GO TO ENT.
BEGIN ROUTINE PRTOT(C).
ADVANCE 1.
10 PRINT(ZZZZZZZZZZZZZ.DD-)TOTAL.
30 PRINT C.
END ROUTINE PRTOT.
SUBTOT: CALL PRTOT("◇"). GO TO ENT.
TOT: CALL PRTOT("*"). GO TO ZERO.
END.
```

This test program uses 57% of the macro instructions of the compiler

Syntax for SELL 10/30/66

```
 1. letter → A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V |
             W | X | Y | Z
 2. digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
 3. identifier → letter | identifier letter | identifier digit
 4. numeric-constant → digit | numeric-constant digit
 5. variable-name → new-identifier | new-identifier ( numeric-constant )
        [see note 2]
 6. variable-declaration → NUMERIC variable-name-list |
             ALPHA (numeric-constant) variable-name-list
 7. variable-name-list → variable-name-list-item |
             variable-name-list-item , variable-name-list
 8. variable-name-list-item → variable-name | variable-name = ( variable-name-list ) |
                              variable-name = variable-name-list-item
 9. subroutine-declaration → subroutine-heading . program-piece END ROUTINE comment
10. subroutine-heading → BEGIN ROUTINE new-identifier |
             BEGIN ROUTINE new-identifier ( parameter-list )
11. parameter-list → new-identifier | new-identifier , parameter-list
12. numeric-variable → numeric-identifier | parameter-identifier |
             numeric-identifier ( numeric-expression ) | parameter-identifier ( numeric-expression )   [see note 5]
13. alpha-variable → alpha-identifier | alpha-identifier ( numeric-expression )
14. primary → numeric-variable | numeric-constant
15. term → primary | primary * primary | primary / primary |
             primary * primary / power-of-ten |
             power-of-ten * primary / primary
16. power-of-ten → 1 | power-of-ten 0
17. numeric-expression → term | numeric-expression + term |
             numeric-expression - term
18. alpha-expression → alpha-variable | " char-string " | parameter-identifier
19. simple-statement set-numeric → SET numeric-variable = numeric-expression |
             ADD numeric-expression TO numeric-variable |
             SUBTRACT numeric-expression FROM numeric-variable
20. set-statement → simple-statement set-numeric | * simple-statement set-numeric |
             SET alpha-variable = alpha-expression
21. psk → A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | B0 | B1 | B2 | B3 | B4 | B5 | B6 | B7
22. key-routine-statement → KEY ROUTINE psk : identifier |
             key-routine-statement , psk : identifier        [see note 6]
```

23. go-to-statement → GO TO label-identifier | GO TO # | GO TO # #

24. call-statement → CALL subroutine-identifier |

CALL subroutine-identifier (argument-list)

25. argument-list → argument | argument-list , argument

25a. argument → primary | alpha-expression | numeric-identifier (label-identifier)

26. enter-statement → simple-enter-statement |

simple-enter-statement ; psk-list

27. simple-enter-statement → ENTER (numeric-constant) numeric-variable |

ENTER (numeric-constant , numeric-constant) numeric-variable

28. psk-list → psk | psk-list , psk [see note 6]

29. type-statement → simple-type-statement | simple-type-statement ; psk-list

30. simple-type-statement → TYPE (numeric-constant) |

TYPE (numeric-constant) alpha-variable |

TYPE (numeric-constant) parameter

31. print-statement → PRINT alpha-expression | PRINT alpha-expression LEFT |

PRINT (mask) numeric-expression

32. mask → control-flag mask-sequence sign-character

33. control-flag → $ | # | | $#

34. sign-character → - | CR |

35. mask-sequence → mask-code | mask-sequence mask-code

36. mask-code → I | E | D | .D | D: | D, | Z | Z: | Z, | ¢ | .¢ | X | .X | S | B | 2B

37. carrier-position → numeric-constant | (numeric-expression)

38. condition → numeric-expression = numeric-expression |

numeric-expression NEGATIVE | numeric-expression CODE (digit) |

KEY (digit) | OVERFLOW | PAGE OVERFLOW (condition)

39. condition-prefix → IF condition , | UNLESS condition ,

40. forms-statement → ADVANCE left-right numeric-constant |

ADVANCE left-right TO numeric-constant |

PAGE left-right numeric-constant |

OPEN numeric-constant

41. left-right → LEFT | RIGHT |

42. loop → begin-loop . program-piece END LOOP comment

43. begin-loop → BEGIN LOOP numeric-variable FROM numeric-constant TO numeric-constant |

BEGIN LOOP numeric-variable FROM numeric-constant

TO numeric-constant BY numeric-constant

44. miscellaneous-statement → ALARM | CLEAR numeric-identifier

CODE (8-bits , 8-bits) | CODE (8-bits , numeric-variable) |

CODE (8-bits , alpha-variable) | CODE (8-bits, label)

45. 8-bits → bit bit bit bit bit bit bit bit

46. bit → 0 | 1

47. basic-statement → set-statement | key-routine-statement | go-to-statement | call-statement | enter-statement | type-statement | print-statement | forms-statement | miscellaneous-statement

48. statement → basic-statement | carrier-position basic-statement | condition-prefix statement

49. program-component → statement . | # statement . | # # statement . | loop . | declaration . | ◇ comment . | label-identifier :

50. declaration → variable-declaration | subroutine-declaration

51. program-piece → program-component | program-piece program-component

52. SELL-program → program-piece.END .

Notes:

1. Blank spaces must not appear within the syntactic categories identifier, numeric-constant, power-of-ten, nor within the "reserved words" which are:

ADD ADVANCE ALPHA ALARM BEGIN CALL CLEAR CODE END ENTER FROM GO IF
KEY LEFT LOOP NEGATIVE NUMERIC OPEN OVERFLOW PAGE PRINT RIGHT ROUTINE
SET SUBTRACT TO TYPE UNLESS

When any two of these categories appear next to each other, they must be separated by at least one blank space.

2. The word "identifier" in the syntax is often qualified with the following meanings:

new-identifier: identifier not a reserved word nor has it appeared in the SELL-program before this point

numeric-identifier: identifier which has appeared previously in NUMERIC-variable-declaration

alpha-identifier: identifier which has appeared previously in ALPHA variable-declaration

subroutine-identifier: identifier which has appeared previously in a subroutine-heading

parameter-identifier: identifier which has appeared previously in a parameter-list

label-identifier: an identifier which is not a numeric-, alpha-, subroutine-, or parameter-identifier, nor a reserved word.

3. A "comment" is any sequence of characters not containing a period.
4. A "char-string" is any non-empty sequence of characters, containing no quote marks except possibly as its first character.
5. The numeric-expression in rule 13 must not involve any subscripted variables, multiplication, or division.
6. The psk's in rules 22 and 28 must appear in ascending sequence as defined by the left-to-right order in rule 21.
7. Further restrictions, for example the size of numeric-constants and identifiers, the total number of identifiers allowed, etc., appear in the language definition. (This ayntax does not completely specify the set of character strings which the compiler will accept as error-free SELL programs, although it gives a very good approximation.)

Translation of the example program

The following operation codes have been used in addition to those presently in GP2:

ENTER [same as NKRCM except it also stores the value entered into the accumulator into the location specified by TKMR, before jumping to the routine for a depressed psk.]

LSMR [loads special memory register, which is either a new register or can share space with one of the others like TKMR]

CLEAR [CLEAR A sets locations A, A+1, ..., (SMR) to zero]

LIB 0:3 ["load index binary", loads the specified index register with the contents of the least significant two digits of the word specified in SMR.]

SIB 0:3 ["store index binary", stores the contents of the specified index register into the least significant two digits of the word specified by SMR.]

TRMP, ADMP, SUMP [like TRM, ADM, SUM, except the previous value of the memory word is saved in backup storage before the operation]

MAJOR [marks a major rerun point; backup storage is cleared and current loc. is saved in backup storag]

MINOR [marks a minor rerun point; current location is saved in backup storage]

BMAJ [backup to major rerun point, restore saved values]

BMIN [backup to 2nd last minor rerun point, restore saved values]

PAL [print alpha going to left]

LUBR [load upper limit of backup table]

LLBR [load lower limit of backup table]

In the following code, "fixup" denotes a new loader operation [specified by a code of 6 instead of 2 or 10] which OR's the specified 9-bit address into the least significant 9 bits of the syllable specified. This is used to make the compiler operate in one pass.

The notation "[0]" has been used in the program below to stand for an address which will be fixed up later on.

Memory allocation: The compiler sequentially chooses syllable locations starting from zero, and words for variables in decreasing locations 1.255, 1.254, 1.253, etc. Constants are also located in the area together with the variables. In the example program, the following memory assignments are made for variables and constants:

```
1.255=DATA(11)=NETPAY
1.254=DATA(10)=MISC
1.253=DATA(9)=INC
1.252=DATA(8)=STAX
1.251=DATA(7)=CTAX
1.250=DATA(6)=PENSION
1.249=DATA(5)=FICA
1.248=DATA(4)=NFICA
1.247=DATA(3)=HOUSING
1.246=DATA(2)=OTHER
1.245=DATA(1)=REGULAR
1.244=TOTALS(13)=W2
1.243=TOTALS(12)=TOT(11)
1.242=TOTALS(11)=TOT(10)
1.241=TOTALS(10)=TOT(9)
1.240=TOTALS(9)=TOT(8)
1.239=TOTALS(8)=TOT(7)
1.238=TOTALS(7)=TOT(6)
1.237=TOTALS(6)=TOT(5)
1.236=TOTALS(5)=TOT(4)
1.235=TOTALS(4)=TOT(3)
1.234=TOTALS(3)=TOT(2)
1.233=TOTALS(2)=TOT(1)
1.232=TOTALS(1)=CKNO
1.231=GTOT(11)
1.230=GTOT(10)
1.229=GTOT(9)
1.228=GTOT(8)
1.227=GTOT(7)
1.226=GTOT(6)
1.225=GTOT(5)
1.224=GTOT(4)
1.223=GTOT(3)
1.222=GTOT(2)
1.221=GTOT(1)
1.220=TNFICA
1.219=TFICA
1.218=TCTAX
1.217=TSTAX
1.216=PROOF
1.215=FICAMAX
1.214=I
1.213=DATE
1.212="PAYROLL"
1.211=P
1.210=X
1.209=J
1.208=120
1.207=6
1.206=127
1.205=7
1.204-133
1.203=8
1.202=139
1.201=9
1.200=145
1.199=10
1.198="**VOID"
1.197=154
1.196=37
1.195="*******"
1.194=A
1.193=11
1.192=83
1.191=10
1.190=83
1.189="S"
1.188="NT TOTAL
1.187="DEPARTME
1.186=11
1.185=11
1.184="S"
1.183="SUBTOTAL
1.182=11
1.181=3625
1.180=#IIIIIIIIZZ,ZZZ.¢¢
1.179=IIIIIIIIZZZZ.DDD
1.178=IIIIIIIZZZZZZ.¢¢
1.177=IIIIIIIIIZZZZZZ
```

```
loc     inst
000.0   BRU    [0]
000.1   BRU    [0]     line #7
000.2   PKA    45      line #9
000.3   PKB    0
001.0   LSMR   1.220   TNFICA
001.1   ENTER  7  0
001.2   PKB    1       line #10
001.3   LSMR   1.219   TFICA
002.0   ENTER  7  0
002.1   PKB    2
002.2   LSMR   1.218   TCTAX
002.3   ENTER  7  0
003.0   PKB    3
003.1   LSMR   1.217   TSTAX
003.2   ENTER  7  0
003.3   PKB    4       line #11
004.0   LSMR   1.216   PROOF
004.1   ENTER  7  0
004.2   SUB    1.220   line #12
004.3   SUB    1.219   TFICA
005.0   ADD    1.218   TCTAX
005.1   ADD    1.217   TSTAX
005.2   EXZ
005.3   BRU    [0]     OKAY
006.0   ALARM          line #13
006.1   BRU    000.2   S1
006.2   fixup  005.3   line #14
006.2   OPEN   0
006.3   SRR            line #15
007.0   fixup  000.1
007.0   LPKR   0.008   line #17
007.1   BRU    [0]
008.0   BRU    [0]     VOID
008.1   BRU    [0]     BACK
008.2   BRU    [0]     W2FORM
008.3   BRU    [0]     CENO
009.0   BRU    [0]     SUBTOT
009.1   BRU    [0]     GRANDT
009.2   NOP
009.3   NOP
010.0   NOP
010.1   NOP
010.2   NOP
010.3   NOP
011.0   NOP
011.1   NOP
011.2   NOP
011.3   NOP
012.0   fixup  007.1
012.0   POS    27      line #19
012.1   PA     1.212   "PAYROLL"
```

```
loc     inst

012.2   LSMR  1.244   line #20
012.3   CLEAR 1.232   TOTALS
013.0   POS   37      line #21 —— insert PKA 2
013.1   LTKMR 1.213   DATE
013.2   TKM   7
013.3   POS   46      line #22
014.0   LSMR  1.232   CKNO
014.1   ENTER 6  0
014.2   PNS-  6  0
014.3   POS   71      line #23
015.0   PKA   02
015.1   LSMR  1.215   FICAMAX
015.2   ENTER 6  0
015.3   PNS-  6  1
016.0   AL    1       line #24
016.1   LSMR  1.231   line #25
016.2   CLEAR 1.221   GTOT
016.3   MAJOR         line #26
017.0   POS   87
017.1   SRJ   000.2   TTD
017.2   LSMR  1.255   line #27
017.3   CLEAR 1.245   DATA
018.0   MINOR         line #28
018.1   POS   87
018.2   PKA   1.
018.3   PKB   5
019.0   LSMR  1.245   REGULAR
019.1   ENTER 5  0
019.2   PNS-  5  1
019.3   PC-P  -
020.0   ADMP  1.221   line #29
020.1   EXK   1       line #20
020.2   BRU   [0]     S2
020.3   EXK   4
021.0   BRU   [0]     S3
021.1   MINOR         line #31
021.2   POS   94
021.3   PKA   01
022.0   PKB   5
022.1   LSMR  1.246   OTHER
022.2   ENTER
022.3   PNS-  5  1
023.0   PC-P  -
023.1   ADMP  1.222   line #32
023.2   EXK   1       line #33
023.3   BRU   [0]     S2
024.0   EXK   4
024.1   BRU   [0]     S3
024.2   MINOR         line #34
024.3   POS   101
025.0   PKA   01
025.1   PKB   5
```

```
025.2   LSMR   1.247   HOUSING
025.3   ENTER  4   0
026.0   PNS-   4   1
026.1   PC-P   -
026.2   ADMP   1.223   line #35
026.3   EXK    1       line #36
027.0   BRU    [0]     S2
027.1   fixup  021.0   line #37
027.1   fixup  024.1
027.1   MINOR
027.2   POS    107
027.3   PKA    01
028.0   PKB    5
028.1   LSMR   1.248   NFICA
028.2   fixup  020.2   line #38
028.2   fixup  023.3
028.2   fixup  027.0
028.2   TRA    1.245   REGULAR
028.3   ADD    1.246   OTHER
029.0   ADD    1.247   HOUSING
029.1   TRM    1.255   NETPAY
029.2   SKK    2       line #39
029.3   BRU    [0]
030.0   TRM    1.248   NFICA
030.1   fixup  029.3
030.1   TRA    1.255   line #40
030.2   SUB    1.248   NFICA
030.3   TRM    1.249   FICA
031.0   TRA    1.215   line #41
031.1   SUB    1.249   FICA
031.2   SUB    1.219   TFICA
031.3   SKA    N
032.0   BRU    [0]     S4
032.1   TRA    1.219   line #42
032.2   SUB    1.215   FICAMAX
032.3   ADM    1.249   FICA
033.0   TRA    1.255   NETPAY
033.1   SUB    1.249   FICA
033.2   TRM    1.248   NFICA
033.3   fixup  032.0   line #43
033.3   POS    107
034.0   TRA    1.248   NFICA
034.1   PNS-   5   1
034.2   PC-P   -
034.3   ADMP   1.224   line #44
035.0   POS    114     line #45
035.1   TRA    1.249   FICA
035.2   PNS-   5   1
035.3   PC-P   -
036.0   ADMP   1.225   line #46
036.1   BRU    [0]     line #47
036.2   LSMR   1.211   P line #48
036.3   LIB    0
037.0   MOD    0
037.1   TRA    0
```

```
037.2   TAIR  1
037.3   MOD   1
038.0   POS   0
038.1   PKA   01
038.2   PKB   6
038.3   LSMR  1.210   X
039.0   LIB   1
039.1   MOD   1
039.2   LSMR  0
039.3   ENTER 5 0
040.0   MOD   1       line #49
040.1   TRA   0
040.2   PNS-  5  1
040.3   PC-P  -
041.0   MOD   1       line #50
041.1   TRA   0
041.2   SUMP  1.255   NETPAY
041.3   MOD   1
042.0   TRA   0
042.1   LSMR  1.209   J
042.2   LIB   2
042.3   TRM   1.256   see note below
043.0   MOD   2
043.1   TRA   0
043.2   TAIR  3
043.3   TRA   1.256   temp
044.0   MOD   3
044.1   ADMP  1.220   GTOT(0)
044.2   EXK   2       line #51
044.3   BRU   [0]     PRNET
045.0   SRR           line #52
045.1   fixup 036.1
045.1   MINOR         line #53
045.2   LIR   0.208   =120
045.3   LSMR  1.211   P
046.0   SIB   0
046.1   LIR   0.250   PENSION
046.2   LSMR  1.210   X
046.3   SIB   0
047.0   LIR   1 207   =6
047.1   LSMR  1.209   J
047.2   SIB   1
047.3   SRJ   036.2   DEDUCT
048.0   MINOR         line #54
  :           } lines 54-57 like line 53
059.0   fixup 044.3   line #58
059.1   POS   151
059.2   TRA   1.255   NETPAY
059.3   PNS-  5  1
060.0   PC-P  -
060.1   SKA   N       line #59
060.2   BRU   [0]
060.3   ALARM
061.0   fixup 060.2
```

061.1	ADMP	1.231	line #60
061.2	TRA	1.248	line #61
061.3	ADM	1.220	TNFICA
062.0	POS	159	
062.1	TRA	1.220	TNFICA
062.2	PNS-	7 1	
062.3	PC-P	-	
063.0	TRA	1.249	line #62
063.1	ADM	1.219	TFICA
063.2	POS	168	
063.3	TRA	1.219	TFICA
064.0	PNS-	7 1	
064.1	PC-P	-	
064.2	TRA	1.251	line #63
064.3	ADM	1.218	TCTAX
065.0	POS	176	
065.1	TRA	1.218	TCTAX
065.2	PNS-	7 1	
065.3	PC-P	-	
066.0	TRA	1.252	line #64
066.1	ADM	1.217	TSTAX
066.2	POS	184	
066.3	TRA	1.217	TSTAX
067.0	PNS-	7 1	
067.1	PC-P	-	
067.2	POS	191	line #65
067.3	TRA	1.220	TNFICA
068.0	ADD	1.219	TFICA
068.1	SUB	1.218	TCTAX
068.2	SUB	1.217	TSTAX
068.3	PNS-	7 2	
069.0	POS	0	line #67
069.1	PKA	03	
069.2	TK	30	
069.3	POS	37	line #68
070.0	PA	1.213	DATE
070.1	POS	46	
070.2	TRA	~~CKNO~~ 1.222 CKNO	
070.3	PNS-	6 0	
071.0	LOK	0 1	line #69
071.1	ADM	~~CKNO~~ 1.222 CKNO	
001.2	POS	60	line #70
071.3	TRA	1.255	NETPAY
072.0	PNS-	5 1	
072.1	POS	71	
072.2	PNS-	5 1	
072.3	POS	80	line #71
073.0	PA	1.213	DATE
073.1	AL	1	line #72
073.2	O	0	
073.3	BRU	016.3	MAIN
074.0	fixup	008.0	line #73
074.0	POS	31	
074.1	PA	1.198	"**VOID"

```
074.2   LOK    0  0    line #74
074.3   TRM    1.214   I
075.0   BRU    075.3
075.1   LOK    0  7
075.2   ADM    1.214   I
075.3   TRA    1.197   =154
076.0   SUB    1.214   I
076.1   EXA    N
076.2   BRU    [0]
076.3   TRA    1.214   line #75
077.0   ADD    1.196
077.1   TAIR   0
077.2   MOD    0
077.3   POS    0
078.0   PA     1.195   "*******"
078.1   BRU    075.1   line #76
078.2   fixup  076.1
078.2   BMAJ
078.3   fixup  008.1   line #77
078.3   BMIN
079.0   fixup  008.3   line #78
079.0   POS    46
079.1   LSMR   1.232   CKNO
079.2   ENTER  6  0
079.3   BRU    069.3   S5
080.0   BRU    [0]     line #79
080.1   AL     3       line #80
080.2   POS    10
080.3   LSMR   1.194   A
081.0   LIB    0
081.1   MOD    0
081.2   PA     A
081.3   LOK    0  1    line #81
082.0   TRM    1.214   I
082.1   BRU    083.0
082.2   LOK    0  2
082.3   ADM    1.214   I
083.0   TRA    1.193   =11
083.1   SUB    1.214   I
083.2   EXA    N
083.3   BRU    [0]
084.0   TRA    1.192   line #82
084.1   TRM    1.256   temp
084.2   LOK    0  7
084.3   LSR    0
085.0   MUL    1.214   I
085.1   ADD    1.256   temp
085.2   TAIR   1
085.3   MOD    1
086.0   POS    0
086.1   TRA    1.214   I
086.2   TAIR   2
086.3   MOD    2
087.0   TRA    1.220   GTOT(0)
```

```
087.1 . PNS-  8   1
087.2   PC-P  -
087.3   BRU   082.2  line #83
088.0   fixup 083.3
088.0   AL    1
088.1   LOK   0  2   line #84
...          lines 84-86 like lines 81-83
094.3   SRR          line #87
095.0   fixup 080.0
095.0   fixup 009.1  line #88
095.1   LIR   0 187  "DEPT...TOTALS"
095.2   LSMR  1.194  A
095.3   SIB   0
096.0   SRJ   080.1  PTOT
096.1   LOK   0  1   line #89
096.2   TRM   1.214  I
096.3   BRU   097.2
097.0   LOK   0  1
097.1   ADM   1.214  I
097.2   TRA   1.186  =11
097.3   SUB   1.214  I
098.0   EXA   N
098.1   BRU   [0]
098.2   TRA   1.214  I
098.3   TAIR  0
099.0   MOD   0.
099.1   TRA   1.220  TOT(0)
099.2   MOD   0
099.3   ADM   1.220  GTOT(0)
100.0   BRU   097.0
100.1   fixup 098.1
100.1   TRA   1.244  W2  line #90
100.2   EXZ
100.3   BRU   016.1  GMAIN
101.0   BRU   [0]    W2FORM
101.1   fixup 009.0  line #91
101.1   LOK   0  1
...       lines.91-92 like line 89
105.1   LIR   0 183  line #93
105.2   LSMR  1.194  A
105.3   SIB   0.
106.0   SRJ   080.1  PTOT
106.1   LOK   0. 1   line #94
....      lines 94-95 like line 89
110.1   TRA   1.244  line #96
110.2   EXZ
110.3   BRU   016.3  MAIN
111.0   BRU   [0]    W2F
111.1   fixup 008.2  line #97
111.2   fixup 101.0
111.3   LSMR  1.231
112.0   CLEAR 1.221  GTOT
112.1   fixup 111.0  line #98
112.2   MAJOR
112.3   POS   87
113.0   SRJ   000.2  TTD
```

```
113.1   TRA    1.219   line #99
113.2   LSR    5
113.3   MUL    1.181   =3625
114.0   TRM    1.249   FICA
114.1   POS    58      line #100
114.2   LSMR   1.246   OTHER
114.3   ENTER  7  0
115.0   POS    34      line #101
115.1   TRA    1.218   TCTAX
115.2   SUB    1.249   FICA
115.3   PNS-   7  3
116.0   TRA    1.218   TCTAX
116.1   SUB    1.249   FICA
116.2   ADMP   1.221   GTOT(1)
116.3   TRA    1.219   line #102
117.0   ADD    1.220   TNFICA
117.1   SUB    1.246   OTHER
117.2   TRM    1.255   NETPAY
117.3   POS    46      line #103
118.0   PNS-   7  3
118.1   ADMP   1.222   GTOT(2)
118.2   POS    57      line #104
118.3   TRA    1.246   OTHER
119.0   PNS-   7  3
119.1   ADMP   1.223   GTOT(3)
119.2   POS    70      line #105
119.3   TRA    1.249   FICA
120.0   PNS-   5  3
120.1   ADMP   1.224   GTOT(4)
120.2   POS    80      line #106
120.3   TRA    1.219   TFICA
121.0   PNS-   6  3
121.1   ADMP   1.225   GTOT(5)
121.2   POS    95      line #107
121.3   TRA    1.217   TSTAX
122.0   PNS-   6  3
122.1   ADMP   1.226   GTOT(6)
122.2   POS    92      line #108
122.3   PKA    1
123.0   PKB    7
123.1   TK     1
123.2   0      0       line #109
123.3   BRU    112.2
124.0   fixup  000.0   line #110
124.0   LPNR   1.177   masks
124.1   LUBR   1.176   upper backup limit
124.2   LLBR   0.125   lower backup limit    (approx. 300 words for backup table)
124.3   BRU    000.1   start program
```

Note: By error when I hand-translated this program, I forgot that locations 1.254 and 1.255 were intended to be reserved for temporary storage. So these are listed as the non-existent locations 1.256 and 1.257 in the program above; actually the compiler would shift all block one locations down 2 from those shown.

Donald Knuth 10/31/66

MEMORANDUM

TO: C. Perkins

FROM: D. Knuth

SUBJECT: Progress report on "SELL"

The SELL compiler, at least for the language as it now exists, is complete and rather thoroughly checked out. By complete I mean that the macrocode which does the compilation is entirely written, and I have also prepared a B5000 program which is an assembler/simulator/monitor routine for this macro code. All that remains is to write the micro code which interprets the macros, and since my macro code language appears to be definitely less complex than GP2 I don't believe there will be any problems in its implementation.
So I can essentially turn over this compiler to you. I will have to spend some time with you, of course, explaining in detail what I have done, but then I think you will be best equipped to follow through on the project at your end, with my future rôle as advisor and checker of manuals and programs, not as programmer!
Enclosed are a bunch of things: A few changes I made to the language as I wrote the compiler (some extensions and some restrictions); a few brief comments on your memo of Jan. 24; a definition of the macro language I used to write the compiler; some simple queries about GP3; and a brief discussion of the memory organization of the SELL compiler. There are also some computer listings, giving my ALGOL assembler/simulator/monitor program, the compiler routines, an example of the short "adding machine" SELL program as compiled, and the output of the Lutheran High School example program given in my first memo. The compiler routines are given with rather extensive comments explaining what is being done, and there is a cross-reference listing at the end which will be very useful in making any changes. I have even cross referenced every use of every macro operation code. The simulator also produced dynamic counts of the number of times each instruction is executed, so the "bottlenecks" and most important parts of the compiler are identified for the sake of priorities.
I will be able to explain all these things to you in much greater detail when you come here, but I hope this is enough to get you started.

Note that I am a little tight for space, in fact there are less than 20 syllables to spare. With a lot of work I expect I could find a dozen or so syllables that could be saved, but not many more. If major additions are to be made to the language, however, I suspect there will be room in the blocks now reserved for micro-code storage, and it would not be hard to permit limited jumping ~~from one block~~ of macro-code instructions from one block to another. It would not be easier to go to two passes, in fact it would harder and less desirable from the user's standpoint. Personally I think it will be best to keep SELL a rather simple language, and not take on too many glorious features that have resulted in grandiose systems like IBM now is stuck with on their System 360!

Regards,

Don Knuth 3/6/67

A. Changes to my Oct. 30 memorandum

Change GP2 to GP3 throughout.

In the example program, pages 2-5, change
INC to INS in line 3.
7 to (7) in line 6.
, to ; in line 21.
line 74 should be BEGIN LOOP I FROM 0 BY 7 TO 154.
similarly, "BY 2" in front of "TO" in lines 81, 84.
add "SET W2 = 1." to line 97.

page 9 line 13 change "decimal" to "decimal value"
line 17 change "10" to "(10)"
line 20 change "subscripted" to "indexed"
line 23 change "of" to "or"
line 24 change "subscripted" to indexed"
page 10 line 4 change "unsubscripted" to "nonindexed"
line 5 change "subscripted" to "indexed"
line 8-9, should say
an index must have one of the four forms
V, c, V+c, or V-c,
where V is a numeric variable and c is a numeric constant.

page 10 line 16 should be "primary / primary * 10^k (scaled division)"
line 17, change "a" to "of"
page 12, end of page, add "The mask character E should not be used."
page 15, delete lines 11-18.
page 15, ~~sxs~~ add sentence at end of page, "No more than four subroutines may be CALLed at any one time, i.e. subroutine calls may ~~XX~~ not be nested more than four deep."
page 16, line 2 should be BEGIN LOOP V FROM r BY t TO s.
line 5 should be punctuated by a period.
page 15, line32, insert "labels" after "constants,"
page 16, last two lines replaced by the following
is a parameter or if V is indexed, SELL might need to use the accumulator and ~~XXXXXX~~ might modify this instruction with index registers. Therefore it is important to know what ~~XX~~ the SELL compiler will do when it is given a sequence of CODE statements. The accumulator is used when V has a subscript containing a numeric variable. Index registers are used for subscripts and parameters, essentially one index register per appearance of non-constant subscripts and parameter in V, and index registers are chosen in the order 3, 2, 1, 0.
page 16, third last line, "V is a variable, a label, or an alphabetic expression."

pages 22 and following have been superceded by more recent work (although they helped me to write the compiler).
The syntax equations on pages 18-21 have been updated in accordance with these revisions, but I will not take the time to list the changes here.

Add to page 15, "Parameters may appear ~~XXXXXXX~~ within the subroutine in the context of numeric variables (possibly indexed), as labels, as alphabetic expression, ~~XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX~~ or as the alphabetic variable in a TYPE statement, but not as the alphabetic variable ~~XXXXXXXXXXXXXX~~ on the lefthand side of an alphabetic SET statement."

B. Comments on your memorandum of Jan. 24

1. Fixup load routine. I independently decided I wanted the same thing you suggested, before I had read your memo! I am assuming these syllable groups can follow each other, as the present full word groups do, and that they have similar parity conventions. I am also assuming they end with the same end-syllable and end-message codes as the present end-word and end-message.

2. a. I am assuming only NERCM at the moment (see below).
   b. The present compiler uses LDES and CLEAR
   c. I need these very simple microinstructions to use binary, in order to handle parameters to subroutines. I don't have any use for storing a decimal equivalent. The compiler uses LIB, SIB with the parameter part of the instruction 1,5,9,or 13 to specify index register 0,1,2, or 3.
   d. The main point I made when we discussed this in Pasadena was that if the hardware can print left, there should be at least <u>some</u> way to use that feature in GP3. If necessary, let it be simply to change direction in the output micro, and let the programmer worry about the curious effect it has on positioning. Certainly I don't think a scan backwards should be used. Your idea of handling the positioning (last four lines of first paragraph on page 3 of your memo) is very nice.
   But I have no strong feelings about including this in ~~XX~~ the SELL option, I mainly want it to be possible to have it in GP3. I put it into SELL mostly because it comes so cheaply.
   e. I can't use your "length" idea, since the compiler doesn't know this until the end of the program, but I have another proposal that economizes on micro storage and op codes:

LBR	0:1	0:255	Set lower limit, store present program step as MAJOR ~~XXX~~
LUR	0:1	0:255	Set upper limit
PROT	0:1	0:255	Save data in table
RETPT		0:1	0: Present program step is a minor return point 1: Present program step is a major return point
RETURN		0:255	Restore data past last n minor return points (but not past MAJOR), and up to the (n+1)st, and branch to the (N+1)st, which is left on the table

The compiler uses RETURN 1 and RETURN 255 only.

3. I'm glad you prefer the way I did it, since I now think the concept has worked out well and it is incorporated in the compiler.

4. Let's talk about this when you come out here. I can't think of a nice way to put this into the language without adding either clumsy constructions, or lots of extra work for the compiler, but I suspect the two of us together may be able to find a good way. ~~X~~

5. Fractions can be programmed in SELL, like you programmed them in GP2.

6. a. Does not affect SELL at this time.
   b,c,d,e,f,g, same; ~~my printouts still use LOK as a mnemonic~~ why INK??
   h. The present compiler uses SKA,SKL,SKK,SKT and corresponding EX. The parameter field has U = 1, L = 1,2,4,or 8. Example, SKZ is like SKL 17.

C. SP/I

Here is an informal description of the machine-like language"SP/i" which I used to program the compiler.

Each instruction has an operation code and some of the additional fields N (3 bits), M (8 bits), L (10 bits). The instructions are in locations numbered from 0 to 1023. The SP/I language is organized around a stack; each stack entry is a full ~~XXXXXXXXXXXX~~ 8-character word.

LOD M	Put "0 0 0 0 0 0 0 M" on top of the stack.
ADD M	Add M to the least significant eight bits of the word at the top of the stack. (Carries do not propagate into other character positions. Arithmetic is binary.)
SUB M	Same as ADD (256-M)
TEN	Multiply the least significant eight bits of the word at the top of the stack by ten. (Eight bit ~~binary/ would/be/useful is/is/is~~ binary arithmetic.)
ORR	Replace the two words at the top of the stack by their 64-bit logical "or"
SLC M	Shift the wood at the top of the stack left, circularly, M digits.
SRC M	Shift right, circularly. (SLC 16 = SRC 16 = no-op.)
PCH M	Punch M as the next eight bits on the output tape. (The punching is delayed, this instruction actually only puts M into a buffer area.)
XEC L	Execute the instruction at location L.
SET	Put the values of TSIZ, TLOC (these are special locations in block 1) into the identifier table entry for the last-scanned identifier.
MASK	See if the mask word, which is second on the stack, matches the k-th mask in the mask table. Here k is at the top of the stack. If not, skip the next instruction.
XOR M	Replace PARITY by the exclusive "or" of PARITY and M. (PARITY is a special location in block 1, used to control parity punching for the output.)
SEE N M	~~XXXXXX~~ (a) If the last character or identifier scanned has been "covered", by the previous SEE instruction, then scan the next character or identifier of the source program. (This scanning may involve output of the preceding line, input of a new line, and/or building an identifier with a lookup in the identifier table. The scanning proceeds according to four modes:

MODE = 0 (normal mode): scan to non-blank character, and if it is a letter, build and lookup the whole identifier starting with this letter.
MODE = 1: scan next character.
MODE = 2: scan to next "."
MODE = 3: scan next nonblank character.
(b) Now if the current character or identifier scanned has the internal code M, "cover" it, otherwise skip the next instruction.
(c) Delete N words from the stack, and set MODE = 0.

GE N M — (a) If the contents of the eight least significant bits of the word at the top of the stack is not $\geq$ M, skip the next instruction.
(b) Delete N words from the stack.

EQ N M — Like GE N M, except = instead of $\geq$.

CALL L — Put a special word on the top of the stack, as "subroutine linkage." Jump to a subroutine in location L. The subroutine starting here should eventually return either by using the instruction XT or XF; at this point a return is mode to the instruction just following this CALL command (for XT) or to the next-but-one instruction (for XF).

Note: There are three modifications of each of the operators SEE, GE, EQ, and CALL.
NSEE, NGE, NEQ, NCALL are like SEE, GE, EQ, CALL, except that they reverse the conditions under which the following instruction is to be skipped.
MSEE, MGE, MEQ, MCALL,MNSEE, MNGE, MNEQ, MNCALL are like SEE, GE, EQ, CALL, NSEE, NGE, NEQ, NCALL except that they cause an error printout to occur if the next instruction would be skipped, and the compiler goes into its error recovery procedure.

GET N M — if M = 0 let MM be the N-th word on the stack. (N=1 is the top of the stack.) If M ≠ 0, let MM = word M of block 1. Then load MM onto the top of the stack.

INX N M — If N = M = 0, let MM be the top word on the stack and delete this entry. Otherwise define MM as in GET. Add MM to the M field (or to the least significant bits of the L field) of the following instruction, mod 256.

INC N M — Let MM be defined as in GET. Increase this value, wherever it is stored, by 1.

XCH N — Interchange the top element of the stack with the N-th word.

TGET M — Load the word in location M of block 3 onto the top of the stack.

TST M — Store the word at the top of the stack into location M of block 3 and delete it from the stack.

ST N M — Store the word at the top of the stack into location M of block 1 and delete N words from the stack.

CMP N M (a) Let the word at the top of the stack have abccdd as its six least significant digits. Add M to CC.
(b) If a ≠ 0, its value should be 1,5,9, or 13. In such a case, the instruction MOD0, MOD1, MOD2, MOD3 (respectively) should be compiled as the next instruction of the object program. Also location a of block I should be decreased by one. (This location represents the number of requests for the index register specified.)
(c) If b ≠ 0, it is treated like a.
(d) Compile the command ccdd as the next instruction of the object program.
(e) Delete N words from the stack.

SCMP M Like CMP 0 M except that locations a, b, of block are not altered.

JMP N L (a) Branch to L, i.e. prepare to take succeeding instructions of the program from location L.
(b) Delete N words from the stack.

XT N
XF N The N+1-st wood of the stack should be the subroutine linkage for the current subroutine. Remove this word, and move the other N words down one space to fill up the vacancy. Now return from the subroutine, as spedified in the CALL operation.

HLT End of compilation.

Further details of these operations are spelled out completely in the accompanying ALGOL listing.

D. Some queries.

The compiler found it very expedient to make the following assumptions about GP3:

1. TAIR 13 is equivalent to TAIR 12
2. The result of addition or subtraction can never be minus zero.

I certainly hope assumption number 2 is valid, otherwise the results could be disastrous! On the other hand, assumption number 1 saves me only two locations.

E. Tentative organization of the compiler, as presently simulated.

Block 0: SP/I macro code

Block I: SP/I micro-code, plus about 24 words of storage used for communication between macro and micro code.

Block 2: SP/I micro-code.

Block 3: words 0-15, mask table
words 16-BUFP input and object program buffer
words STAKP-127 stack
words 128-255 identifier table

BUFP advances upward, STAKP advances downwards, an error would be printed if they ever cross each other but I found they never even came within 50 words of each other. The "top" of the stack is in location STAKP, the second is in STAKP+I, etc. The identifier table requires one word per identifier, plus space for 30 reserved words, so we can accommodate 98 identifiers.

The current line of input text which the compiler is scanning resides in words 16, 17, ... as a sequence of characters followed by the character "200" (in binary). Then the object program follows immediately after as a sequence of 8-bit items. When the character "200" is XXXXXXX encountered while scanning, the whole area is punched out onto paper tape. But if an error is detected, this area is cleared and the following input line replaces it.

Simulation indicates that a typical line of input requires about 200 macro instructions to be performed between input and output of the buffer area.