REMOTE COMPUTING-AN EXPERIMENTAL SYSTEM

Part 1: External Specifications

T. M. Dunn and J. H. Morrissey
Development Laboratory, Data Systems Division
IBM Corporation
New York, N. Y.

INTRODUCTION

Background

Remote computing has been around as long as computers themselves.[1] More recently, interest has revived in providing remote users with convenient, economical access to a large central computer. Considerable attention has been addressed to its economics[2] and practicality.[3] Several batch-oriented systems have been implemented.[4, 5] The techniques of time-sharing[6, 7] a la rge[8, 9, 10] or small[11] system have been described, as have the attendant advantages of man-machine interaction[12, 13] for symbolic mathematics[14] and program testing.[15] Several input-output devices have been considered, including typewriters,[16] displays,[17] and dial-voice equipment.[18]

The management,[19], [20] [21] systems analysis,[22] program testing,[23] and documentation[24] of specialized real-time systems have also been emphasized, but much less attention has been given to the design of general-purpose op.-line systems.

This paper reviews some general system requirements and applications criteria leading to basic design objectives and constraints for remote-computing systems. An experimental system using a number of remote terminals time-sharing a standard computer is then described.

System Requirements

There are several requirements that must be considered when designing a practical remote-computing system.

1. The remote user does not have access to experts for programming assistance and advice. If he uses a problem-oriented language to express his problem, he requires that the request for and display of debugging data be consistent with this programming language.
2. Because his jobs are processed completely without human intervention, the remote user obviously cannot communicate his desires to a machine operator. This leads to several considerations:
   a. The command statements used to regulate the system should have a form and content consistent with the programming languages employed.
   b. The remote user requires a powerful command structure; he should have the ability to state such things as run time, job status, error procedures, and disposition of output.
   c. The conversational remote user requires access to many of the facilities available to the machine operator in the form of console buttons, lights, and switches. He should receive steady reassurance that "all is well" by some

form of periodic "blinking" at his terminal. He also needs the ability to stop his "machine" at any time and without loss of data-so that he can perform such simple functions as changing some printer paper, placing more input cards in a reader, or discontinuing a job.

3. The remote user is very conscious of input/output volumes. He must have the capability to modify decks without complete retransmission, and he should have the option to selectively inspect and list output data, as opposed to massively transmitting entire output files. Also in this spirit, he desires to keep His various decks i random storage--quickly and conveniently available for modffication, processing, or review.
4. Finally, the remote user should be given the impression that he is the only user and that he is in complete control of the situation. More specifically, in a time-sharing environment, he should be totally secure from unwanted, possibly destructive, interaction by ·others. Ideally, the computing and response rate of his terminal should not radically fluctuate according to the demands of the rest of the user population; in other words, his "share" of the central system should perform at a relatively uniform processing rate.

Application Criteria

The following criteria were among those used in deciding whether commercial or scientific applications were more amenable to remote operation:

1. Time devoted to program development vs. production runs;
2. Importance of job turnaround vs. computer throughput;
3. Available programming languages;
4. Conversion problems;
5. Reliability objectives;
6. Input/output volumes;
7. Random-storage requirements.

It was concluded that there was more immediate technological significance and lower hardware-software risk in placing initial emphasis primarily on the scientific applications area.

Design Objectives

The following functional design objectives were then established :

1. Output data should be as user-oriented as the source language ;
2. Diagnostic messages and logical analysis should be definitive enough to allow program debugging to take place at the same level as program construction ;
3. The user should have immediate and sustained access to the computer;
4. The user should have the ability to execute, alter, and change values, variables, and formulas, and to request information selectively;
5. The system should be at least as easy to learn as the FORT RAN [25] language;
6. The print volume should be minimized without loss of quality, on demand of the user;
7. The system should provide the shortest possible solution time, ideally no longer than the time required to construct and run the solution itself.

Design Constraints

Finally, the following restrictions were imposed:

1. Use only an existing standard equipment configuration;
2. Use, and stay consistent with, an existing language.

The first constraint serves to keep attention primarily on fundamental programming problems and discourages the favorite desire of many engineers to solve systems problems by the design of a new feature or the development of new devices.

The second constraint serves to keep attention primarily on the processor design and discourages the favorite desire of most programmers to solve systems problems by the design of new languages or the development of new compilers.

The Approach

Our approach to accomplishing the objectives fuses the old technique of interpretive execution with the relatively new one of time-sharing a CPU. Thus the cost of sustained access to a computer by an individual is spread over a wide base. The internal form suitable for interpretive execution retains all the information contained in the user's original statement of the problem, thereby making symbolic debugging possible. Together, these two techniques make the conversational mode of operation on current equipment a practical reality.

Nevertheless, the service this system performs is not a matter of cleverly getting something for nothing, but a justifiable trade-off. Execution time is greater, but elapsed solution-time is significantly smaller. The cost of the total equipment configuration is comparable to that of typical large computer systems, but the cost per terminal is in the small computer range. In short, this system converts some of the raw power of the computer into condensed solution-time and greater creative power for the user.

OPERATIONAL DESCRIPTION

Equivment Configuration

The hardware (see Figure 1.1) consists of:

1. An IBM 7040[26] with 32K memory;
2. An *IBM* 1301[27] disk-file storage, for permanent retention of user programs;
3. An IBM 7320[28] drum storage, for the continual swapping of user programs;
4. A few magnetic tape units, for logging system transactions and to maintain normal computer capability;
5. An IBM 7740 communications control system,[29] for the real-time acceptance and transmission of messages;
6. A number of IBM 1050[30] terminals with keyboard-printer and, optionally, a card reader and card punch.

The User's Terminal Console

In use, the terminal console (see Figure 1.2) appears to be a self-sufficient FORTRAN machine. The user is completely unaware of any assembly system or the internal organization of the central computer. The language is consistent with FORTRAN, augmented by a set of operating, testing, and debugging statements. The mode of communication is called "conversational," as opposed to "batch," because the basic unit of input is the individual statement rather than an entire program, and every communication by one of the participants is acknowledged _by the other.

The form in the terminal printer (see Figure 1.3) consists of 126 columns (10 characters/ inch). The first 12 (unnumbered) columns are rese; ed for control fields, and the remaining 114 (numbered) columns are identical to a FORTRAN coding form, except for length.

The first five columns of the control-field portion are used to display a line number (101.0

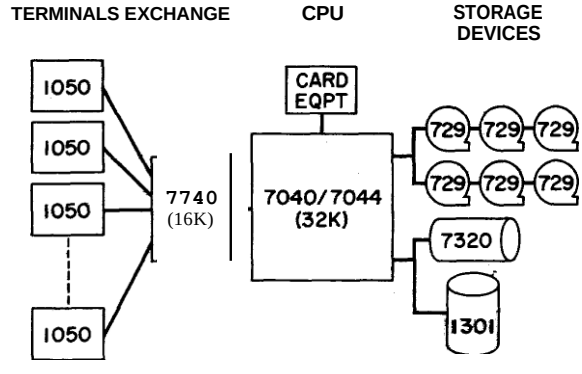

Figure 1.1. Remote-Computing System: Equipment Configuration

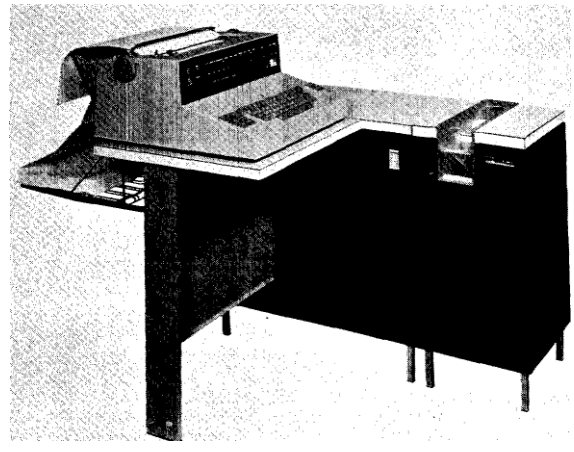

Figure 1.2. IBM 1050 Data Communications System (with card reader) .

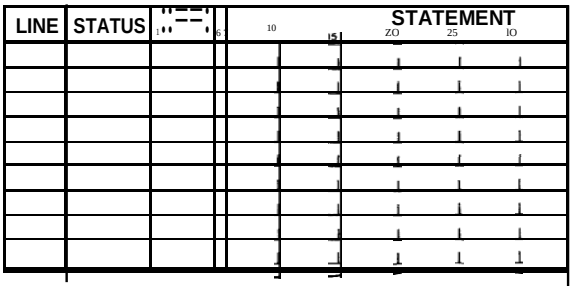

Figure 1.3. Remote-Computing-Terminal Programming Form.

to 999.9), i.e., an identifier automatically generated by the system to uniquely label each program statement. The first of the remaining seven control columns contains a code character -minus, plus, or equal-denoting the source of the line-input in Command or Program mode (see below), or output. The second through sixth columns of this field contain a status word which cues or informs the user : for example, the cue word READY invites the user to enter his next statement; and ERROR readily identifies a diagnostic message. (The seventh column is always blank.)

In the first two (numbered) columns of the FORTRAN-like portion of the form, "C blank" is treated as an ordinary comment, while "CV" causes the statement to be ignored by the system and serves as a hard-copy comment not germane to the program itself. Any character other than "C" in column 1 is considered a monitor-control character and the statement is treated as a normal comment; "CF" is treated as a comment in FORTRAN, but considered a normal statement in this system, and thereby serves to keep a source-program card deck compatible with other FORTRAN compilers.

General Operating Statements

The general operating statements (see Figure 1.4) may be used at any time. COMMAND establishes the Command mode (see below). EXIT signs the terminal off. Each terminal is set to standard real and integer formats (E15.8, I 11) for output of all values not under explicit FORMAT control. EDIT (- - -) permits the user to change either or both.

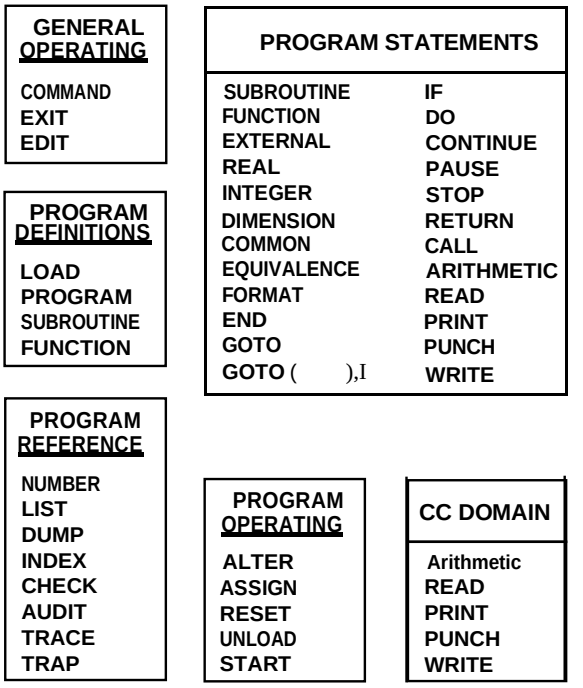

Figure L4. System Language.

Command Mode

When no program is active for a given terminal, that terminal is said to be in the Command mode; and, conversely, the entering of a COMMAND statement will destroy the active image of the user program. Since no program is active, statements cannot be retained, but must be processed immediately. Consequently, the user may employ only the general operating statements, the program-definition statements (see below), or a limited form of the arithmetic-assignment statement. This latter provision allows the terminal to be used as a fast, versatile symbolic desk calculator. In this mode, the user enters a statement of the form, X==any expression consisting of constants and built-in functions, and the system immediately evaluates the expression and prints the result on the user's terminal.

Program Mode

1. *Program-Definition Statements*
   The program-definition statements (see Figure 1.4) initiate the Program .mode. LOAD fetches an existing program from the user's library, while PROGRAM initi-

ates the creation of a new main program, and SUBROUTINE or FUNCTION introduce subprograms.

2. *Program Statements*
In the Program mode, certain statements, called program statements (see Figure 1.4), are translated and retained. Consequently, unlike the arithmetic assignment statement in the Command mode, their execution is initiated by the user. All FORTRAN statements used by this system (see Figure 1.4) are program statements. It is with these that the user may construct a stored-program solution to his problem; all other statements are processed immediately and are not retained. As with FORTRAN, there can be only one main program-but there can be numerous subprograms, with the restriction that no single subprogram exceed 4000 words of storage. In this way, although individual program size is restricted, total program size may be much larger.

3. *Program-Operating Statements*
The program-operating statements (see Figure 1.4) allow the user to execute, alter, select I/0 components, reset certain initial conditions, and unload his programs to library storage, at any time.

START with various operands allows the user to begin execution from the first or any other executable statement, or to execute a segment from one line or statement number to another, or to resume execution after manual intervention.

ALTER allows the deletion and insertion of statements.

SELECT permits specifying the console's I/0 devices other than the keyboard-printer.

RESET with various operands initializes the program for fresh testing runs.

UNLOAD places the user program in library storage, but does not remove it from the active status.

SOURCE LANGUAGE DEBUGGING

Debugging information is requested and displayed in a form consistent with the source programming language.[3, 1, 32]

Diagnostic Structure

Errors committed by the user may be classified in two broad categories : syntactic and semantic.:[33]

1. *Syntactic Errors* [34]
All syntatic errors are considered the responsibility of the system and are further categorized as follows :

Composition. Typographical errors, violations of specified forms, and misuse of variable names (e.g., incorrect punctuation, mixed-mode expressions, undeclared arrays, etc.)

Consistency. Statements which are in themselves correct, but conflict with other statements (e.g., conflicting declaratives, illegal statement ending a DO range, failure to follow each transfer statement with a numbered statement, etc.)

Completeness. Programs which have not been completely specified by the user (e.g., transfers to nonexistent statement numbers, improper DO nesting, illegal transfer into the range of a DO loop, etc.) .

Errors of composition and consistency are detected immediately upon entry of the offending statement. The user may immediately substitute a correct statement.

Errors of completeness are discovered when the user signifies that his program is complete (by entering the END statement).

Some errors (e.g., invalid subscript value, reference to an undefined variable, arithmetic spills, etc.) may, of course be detected only during execution. In this case, after a display of the error condition and its location, execution is interrupted and the terminal reverts to READY status. The user then has the option of either immediately correcting his error or proceeding with the rest of his program.

For all syntactic errors, the diagnostic message is concise-in that the variable in error is named, or the column where the error occurred is specified-and of ten tutorial in suggesting the procedure for obtaining correct results.

2. Semantic Errors

Semantic errors are concerned with the meaning or intent of the programmer and are necessarily his responsibility. However, an extensive set of debugging aids are provided for manipulating and referencing a program in ferreting out errors in logic and analysis.

Value Manipulation

Not too surprisingly, some types of prog.cam statements are also useful for manipulating the values of a user program (see Figure 1.4). Consequently, special characters--called process codes-may be inserted into the first two columns to allow . the use of these statements as commands, thus causing values to be read into or out of selected variables. (This is analogous to the panel-entry/ display functions performed at a conventional computer console.)

For example, "CC" in columns 1 and 2 of the FORTRAN-like form (Figure 1.3) has the following effect on its accompanying statement: the statement is immediately executed with all the effects of normal execution, but no new variable names are created; the statement is then discarded and does not become a part of the program. Thus, values may be inserted into factors or parameters at any time, thereby creating completely new testing situations without having to build their presence into the logic of the program or attempting to anticipate the debugging operations required.

Program Reference Statements

Program reference statements (see Figure 1.4) allow the user to display various vital conditions of his program. These statements are not retained. They are acted upon immediately, then discarded. Through their use, a complete, dynamic record of both control flow and data usage may be obtained.

DUMP (Figure 1.6b, line 143) produces an alphabetically ordered listing of program identifiers with their current values. Array subscripts are stepped automatically, contiguous zero-valued array elements are omitted, and empty elements (i.e., those never receiving a v:ilue) are flagged.

TRACE with various operands (Figure 1.6c, line 144 et seq.) allows the user during later execution to print out every change of value for all variables, for all variables within a specified region, or for specific variables whenever they are changed.

TRAP (Figure 1.6c, line 145 et seq.) is a logical trace of all control transfers, or of all transfers within a specified region.

NUMBER (Figure l.6e) with various operands resequences and lists the program.

LIST generates a listing of the entire program or any specified portion of it.

INDEX (Figure l.6f, line 231) produces a complete cross-reference table, ordered on statement numbers and variable names, showing the line number of every statement in which each statement number or variable name appears and whether it was declared, defined, or referred there; or, INDEX produces a single such line for a specified variable or statement number. Any line of the table which is, or may be, in error is marked with an asterisk. These features are very useful when making program modifications.

```
101 -READY C        C8IIIAND THIS IS A SAIIPI.E PIHIGRAII
                    PRIIGRAII SAIIPI.E
101 -READY          giiNT= ZPU!T <51>, TABLEC500)
104 +READY          X : O
105 +READY          Y : 1.
106 +READY 2 : 1
106 +ERR0R 04200. STATEI'IENT NIIT IN LANGUAGE.
                    TYP8GRAPHICAL ERRIIIRS IIAY BE CIRRECTED IIIIUDIATELY
109 +RE:nY            srSTITUTIN8 A CBRRECT STATEIIENT VIA KEYBIIARD.
              101 R = 11m;2 R, ZPUIT
112 +READY          PRINT 102
              102 ;:11:T(5X1HX5IUHY>
115 +READY        1 PRINT 103, X , y
116 +READY 103 FBRIIAT(2XF5.2,F8.5)
117 +READY CV         ANY STATEIIENT BR SEQUENCE IF STATEIIENTS IIAY BE
           CV       JmFJED BY IIIIIEDIATE EXECUTIIIIN AFTER ENTRY.
110 =II01 •
112 :8102           X       y
114 =ERIHIR TRANSFER PBINT N DBES NIT EXIST
120 +READY          START 1
115 =8103      0.     1000000
              END III ,           ENCIIIUNTERED DURING EXECUTI0N
122 +READY          X : X + DELX
123 +READY          DELY : X• Y*DELX
124 +READY          Y : Y + DELY
125 +READY        2 TABLE< I> : X
126 +READY          TABLE< I+I) : y
127 +READY          IFCX - 1,>1,1,3
128 +READY        3 D0 4 J : 1, I, 2
                    X : TABLE C J>
129 +READY      ARIT r«TABLECJ+I>-TASU:<2»/( TABLE< I+I>-TABLE<2»•50 >
130 +ERR0R  IIIXED          DECBIIP0SITI0N ERRBRCS)
                    STATEMENTS IN ERRBR AT TIIIE 0F ENTRY ARE NBT ACCEPTED.
                    SUBSTITUTI0N I'IAY BE IIADE VITHI!UT RE-ENTERING PRIIGRAII
                     0; m A L   ; 1 >-TABLEC2I >IC TABLE< I+I I-T SLE<2> >•50. >
135 +READY          PRINT 104, X , ZPL9T
136 +READY 104 FIIIRIIATCF5.2,5! At>
              4  ;  Tw :  ZPLIIT( K+I
139 +READY          END
```

Figure 1.6a. Sample Program: Creation and TEsting.

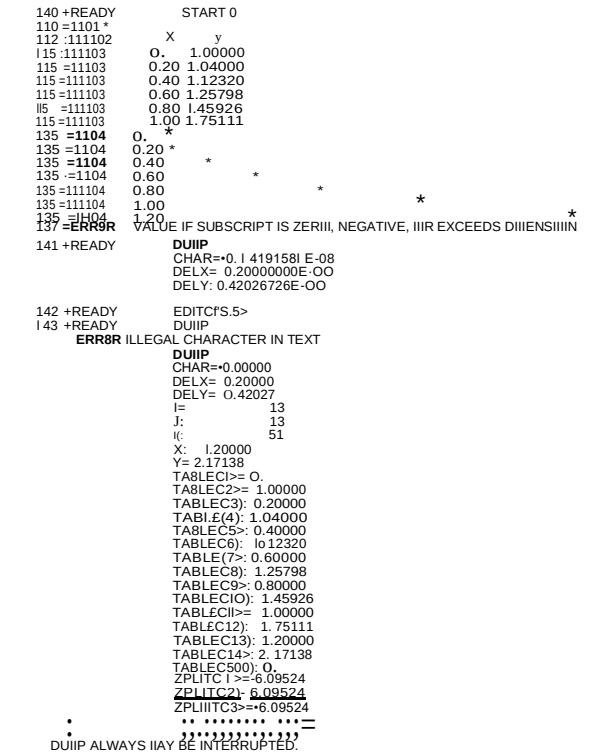

Figure 1.6b. Sample Program: Creation and Te_sting (continued).

CHECK (Figure l.6f, line 232) is an abbreviated INDEX in that only erroneous and suspicious items are displayed (i.e., only those INDEX lines marked with an asterisk).

AUDIT generates cross-reference information based on the execution of the program, showing which sections were never executed and which variables were never set, or set but never used. This concise, post-mortem summary of incomplete control flow and data usage is a powerful aid in ensuring the thoroughness of program debugging.

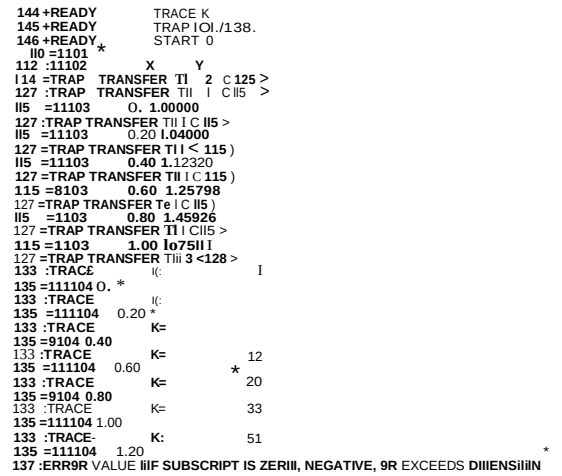

Figure 1.6c. Sample Program: Creation and Testing (continued).

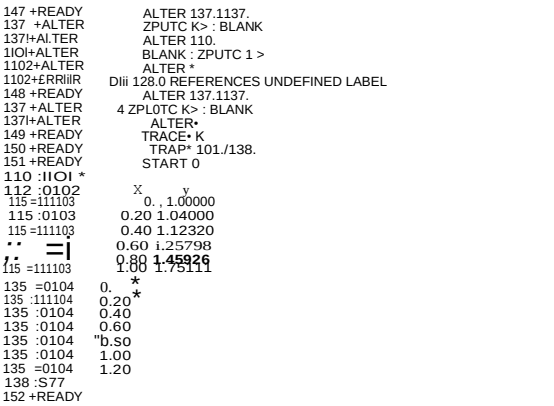

Figure 1.6d. Sample Program: Creation and Testing (continued).

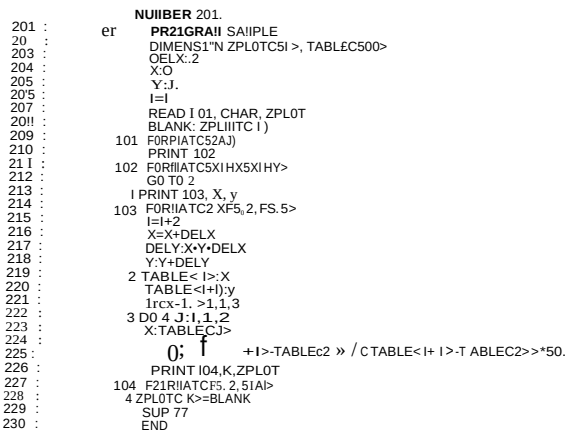

Figure l.6e. Sample Program: Creation and Testing (continued).

```
231 +READY          INDEX
                    I          +213.   -221.
                    2          +219.  -212.
                    3          +222.   -221.
                    4          +228.  -222.
                    5          +207.
                   101         +209.  -201.
                   102         +211.   -210.
                   103         +214.   -213.
                   104         +227.   -226.
                BLANK          +208.   -228.
                CHAR           +207.   -225.
                DELX           +203. -216. -217.
                DELY           +217. -218.
                I              +206. +215. -21,. -219. -220.
                               -222. -224.
                J              +222.   -223.   -224.
                K              +224.    -225.   -228.
               .SAI'IPLE        201 •
                TABLE          202.    +219. +220. -223. -224.
                X              +204. -213. +216. -216. -211.
                               -219. -221. +223. -226.
                               +205. -213. -211. +218. -218.
                               -220.
                ZPL0T          202. +207. -208. +225. -226.
                               +228.
232 +READY          CHECK
                    *    5     +207.
      =
 33 +READY         •SAI'IPLE    201.
```

Figure 1.6f. Sample Program: Creation ap.d Testing (continued).

Built-in Subroutines

Since only program statements are retained, many excellent testing and debugging commands would not be available under program control, but would require the presence of the user at the moment of execution. To overcome this limitation, most of these statements have been designated as "built-in subroutines," a concept completely analogous to FORTRAN built-in functions. These statements, without change in their form, may be made the operand of a subroutine CALL statement. In this way, all the console testing and debugging f eatu res which may be of value are also available under program control.

COMPATIBILITY

Studious regard has been paid to maintaining consistency with other FORTRAN compilers. Programs written in the system language are acceptable without change to conventional FORTRAN IV processors. FORTRAN IV programs are acceptable to the experimental system with the following limitations:

1. The program must be written with statements from the system subset.
2. **A** restriction of all one-pass translators is that the source-deck ordering must have the declarative statements precede the imperative statements. Of course COMMENT and FORMAT statements may appear anywhere.
3. As in most compilers, the sequence of translated code for arithmetic expressions may differ from that produced by other compilers and slight discrepancies due to variations in truncations JIIay occur.
4. Some minor differences in the internal representation of program constants, caused by different conversion routines, may also create slight differences in numerical results.
5. Individual source programs are limited to approximately 400 statements. This limit may often be circumvented by segmenting oversized programs into smaller subprograms.
6. Other Factors :
   a. No arithmetic function statements;
   b. No logical, complex, or double-precision variables;
   c. Number of elements (i.e., constants, variables, arrays, and functions) must be less than 190 ;
   d. Only one continuation card;
   e. No magnetic tape 1/0;
   f. Some minor restrictions on equated variables;
   g. Constants-
      Reals: 8 digits, with magnitude within range 10^{-32} to 10^{32} or with zero magnitude,
      Integers: 10 digits;
   h. Array names must appear in a DIMENSION statement prior to any other appearance;
   i. Maximum 1/0 record size is 133 characters;
   j. Array names used as arguments must be declared in COMMON.

EXAMPLES

A program exhibiting many of the features available in this system is depicted in Figures 1.5 and 1.6. Figure 1.5a shows the final, correct version of the program. Figure 1.5b shows the correct output produced as a result of execution (see START statement, Figure 1.5a, line 128).

```
                    CIIIIIIAND
IOI -READY  C       THIS IS A SAIIPLE PRIIGRAII.
IOI -READY  C -
IOI -READY          PRBGRAII SAIIPLE
102 -IREADY         DIIIENSIIIN ZPLIIT (52>, TABLE (500)
t03  -IREADY        X : 0
104  -IREADY        y = ,.
105  -IREADY        I : I
106  -IREADY        READ IO 1, DELX ,CHAR ,ZPLIIT
!C,7 iI!EADY 101     FIIRIIAT CI' 7.4,53A I)
!08 -IREADY         PRINT 102
109 -IREADY   102    FIIRIIATC51CIHX7XIHY)
I 10 -IREADY        2 TABLE CJ> : X
111  -+READY          TABLE CI +I) : Y
112  -!READY        I PRINT 103, X, Y
I 13 iI!EADY  103    FIIRIIAT<2KF7.4,F8.5)
114  -+READY          IFCX•t.>5,3,3
115  -+READY        51 :J +2
116  -+READY          X:X+DELX
117  iI!EADY         DELY =X •Y •DELX
118  -+READY          y : Y + DELY
Il9 -+READY         GBTO 2
120  -IREAOY        3DO4J=t,!.2
121  -+READY          X : TABLE<J)
122  iI!EADY         K :t.+CCTABLE CJ+D•TABLE C2)>1CTABLE CJ +D•TABLE (2) >•50.>
t23 iIIEADY          ZPL8T CK ) : CHAR
124 -+READY          PRINT 101, X, ZPLIIT
I    -IREADY       4 ZPL8T CK> =ZPLIITCK+I>
126  -IREADY          STfIP 11
127  -IREADY          END
128  -IREADY          START 0
```

Figure 1.5a. Sample Program: Final Form.

```
106  =I 101   00.0625•
108  ::e102        X              y
112  ::9103        O.          1.00000
112  ::e 103       0.0625      t.00391
II2    :8103       0.1250      t.01175
112  ::e103        O.1875      t.02361
112  ::e103        0.2500      t.03960
112  ::e103        0.3125      t.05990
112  ::e103        0.3750      I.08475
112 :8103          0.4375      t. 11441
I 12  :8103        0.5000      I.  14923
112  :8103         o.5625      ,.  18963
112  :8103         0.6250      t.23610
112 :8103          0.6875      t.28922
I 12  :8103        O.7500      t.34965
112  :8103         0.8125      t.41819
112  :8103          0.8750     1.4957-4
112  :8103        o.9375  t.58339     I
112  :8103                        O.
124  :8101           .0000    t.68235
124  ::e101           •
124  :8101       0.0625•
124  :9101       o., a.
124  :8101       o. 1875 •
i2-oi  :iIiOi    0.2500 •
124 :8 IOI       o.3i.25      •
124 :8101        0.3750         •
124  :8 101      0.4375            •
124  :8 IOI      0.5000               •
124 :8101        o.5625
12-4 :8101       0.6250
124  :8101       0.6875
124  :8101       0.7500
124 =e101        0.8125
124 :8 IOI       0.8750
124 =e101        0.9375
126 ::S77         1.0000
129 -IREADY
                 UNL8AD
```

Figure 1.5b. Sample Program: Final Execution.

Figure 1.6 depicts a preliminary attempt to create and test this program. (All references that follow are to Figure 1.6.)

Input to the system may be from the keyboard or card reader at the remote terminals, or through input equipment located at the central computer. At line 106 a mispunched card causes printing of an error message. The user now suspends automatic input, substitutes a correct statement via keyboard, and then resumes automatic input. Of course, the substitution could have been made later by means of an ALTER (see below).

At lines 119 and 120, the user initiates intermediate execution and verifies his FORMAT statements before going further. In this manner, any statement, sequence of statements, DO loop, etc., may be debugged as the program is entered ; or sections may be tested independently of the remainder of the program.

Execution of the entire program, line 140, discloses a number of bugs. Inspection of line 137 discloses the use of K as subscript. K could be printed selectively, but the user decides to dump all variables (see line 141). After DUMP starts, he interrupts it in order to change the format of the display and then dumps again (see line 143). In the event that the dump showing K == 51 is not a sufficient clue to the error, the user establishes a TRACE on Kand a TRAP on the entire program, and starts again (see lines 144-146). This produces, together with his programmed output lines, a dynamic listing of control and data flow, before terminating with the same error message.

At line 147, the statements in error are corrected, but a statement number is inadvertently omitted. On terminating the ALTER status, atline 128 references a nonexistent label. This error i:::; cu1Tected ttud a .:::ub.:::ev Ueut 1 u1111i11g vf the program, line 151, shows that the subscript is now behaving properly.

There are other changes to be made, however. The NUMBER at line 152 yields a clean, renumbered listing of the current state of the program. Line 231 shows a complete INDEX, and line 232, the results of a CHECK statement. All of these will be helpful in reorganizing and documenting the final, correct version of the program.

Figure 1.7 shows the immediate evaluation of arithmetic expressions, consisting solely of constants and FORTRAN functions, in the Command mode.

EXPERIENCE

The system, from its most primitive form to the present, has been running for more than a year. A formal tryout of the system was run in November 1963 with- 10 students attending IBM's Systems Research Institute.

```
                    CfIIIIIAND
101 -READY          Y=2.5065•IO.. c10.+1.>>1EXPrc-10.>
101:                Y: 0.11379485[ 08
JOI -READY          HENRY=2.E-9*50.•a.eer (2.•50./10.>-1.0+10.150.
101 -ERRIIR 04117. PARENTHESES N8T IN BALANCE.
101 -READY          HENRY :2.E •9*50.•a.lIG, cz.•50.110. >-1.+10./50. >
IOI :               HENRY: 0.1502585<E-06
IOI -READY          R8$T I :(•25.-ISQRTFC25.••Z•4.• I.•2.))/C2.•I>
101 -ERRfIR ARITIIIIETIC DECIIIP8SITI8N ERRfJR CS>
101 -ERR8R IIIXED 118DE
101 -READY          R88T I =<•25.-ISQRTF (25.**2•4.U .•z.))/C2.•I• >
101 :               R88T t:-0.80257654:•0I
IOI -READY          K£NRY=2.E·9*50.•Q.8Gr (2.•50./10. l-1.0+10./50. >
101 :               HENRY: 0.!502585<E·06
101 -READY          VAL=1.JCl!ISF(50.HUGFCABSF" CSINF" (50.12.>JCf!SF <50.12.» >
101 :               VAL=•0.9771499EE 00
101 -READY          AREA=2.•10.•5.*5INFCJ.I416/lO.>
101 :               AREA: 0.30901768£ 02
101 -READY          ARC=2•.SQRTI"C4.••2+1.JJJ3$2.••2>
101 :               ARC: 0.92J7575JE 01
101 -READY          ARC:2.•<4.•4.+4.•2.•Z./.J.>••0.5
101:                ARC= 0.92376041[ 01
101 -READY          S:<8SF(40.>••<20.+l.)/20.II.>
101 -£RR9R 04117. PARENTHESES NI!IT IN BALANCE.
101 -READY          G:0.5>11..fGF«l.-lSINFC45.»IC1.-SINFC45.»>
JOI :               8: 0.12'94177£ 01
101 -READY          S ::SINr < 45. >
101 =               S= o.s5o90J5:!E oo
101 -READY          G:0.5>11..f!GF«l.+.70711/Cl.-.7071»
101 :               G: 0.88135999£ 00
101 -READY          E=20.*ATANFC20.l4.l•4.l2.>ll..8GrC4.••2+20.**21
101 :               E: O.15406644: 02
101 -READY          Q:(2.lCJ.1416*10. »••0.5*5INf' < 10. >
101=                o=-o.1J126J5'l[•OO
101 -READY          Q:0.7978/SQRTf'Cl0.>"6INf'Cl0.>
101 :               Q:•0. ll7249ll!t•OO
101 -READY C
101 -READY
```

Figure 1.7. Examples of Command Mode Operation.

USER	BACKGROUND SKILLS		SRI EXPE RIMENT				
			7090 FORTRAN		REMOTE COMPUTING		
	FORTRAN	TYPING	NO. RUIIS	NO. PROGRAMS DEBUGGED	NO. HOURS TRNG	DBUG	NO. PROGRAMS DEBUGGED
A	HIGH	LOW	NONE	NONE	2	2	3
B	HIGH	LOW	5	J	213	11/:S	2
C	HIGH	NONE	NONE	NONE	2	3	3
D	MEDIUM	HIGH	12	5	I	6	4
E	MEDIUM	HIGH	6	'2	3	3	2
F	LOW	MEDIUM	NONE	NONE	2	6	5
G	LOW	NONE	10	3	2	4	2
H	LOW	NONE	4	2	2	I	2
I	LOW	NONE	3	I	11/Z	fl/2	3
TOTALS			40	16	16	28	26

Figure 1.8. Results of SRI Tryout.

The students were divided into two groups, I and II, and given the same set of problems to be solved in FORTRAN. Group I was told to do the odd-numbered problems on the IBM 7090 and the even-numbered ones on the remote-computing terminals. Group II reversed this polarity.

The chart in Figure 1.8 shows the answers given by nine of the participants (the tenth failed to return his questionnaire) to the following questions :

1. "How much FORTRAN experience have you had?" (Answer was evaluated HI, LO, MED, NONE.)
2. "Have you had any typing experience?" (Answer was evaluated HI, LO, MED, NONE.)
3. "How many times did each problem go to the 7090 before you obtained correct results?" (Number of runs were summed.)
4. "How many problems did you debug on the 7090?"

5a. "Approximately how many hours of training did you have on the terminal console?"

5b. "How many debugging hours?"

6. "How many problems did you debug on the terminal console?"

Because this experiment was of limited scope, the experience reported must be taken cautiously. There are many variables which affect the usefulness and economy of this approach, and continuing field trials will yield more precise information.

SUMMARY

The time-shared use of a computer provides a convenient, economical service to numerous remote users. This access is enhanced by use of conversational, source-language debugging techniques. Although the experimental system is oriented to the IBM 1050 terminal, the FORTRAN language, and scientific applications, the techniques described are useful with other terminal devices, programming languages, and application areas. Preliminary opt:rating experience indicates that systems such as the one described have considerable potential in enabling personnel less skilled in the programming art to rapidly obtain solutions to their problems.

ACKNOWLEDGEMENTS

Among the several people making significant contributions to the system, the authors wish to specifically acknowledge the work of Miss Geneva Butts, who implemented a major part of the source-language debugging features, and Mr. Murray Kizner, who implemented the translator routines for many Command features.

REFERENCES

1. E.G. ANDREWS, "Telephone Switching and the Early Bell Lab. Computers," *Bell System Technical Journal*, March 1963.

2. A RAND Symposium, "Economics of Remote Computing," *Datamation,* September 1961, October 1961, November 1961.
3. R. L. PATRICK, "So You Want To Go On Line," *Datamation,* October 1963.
4. D. B. BREEDON and P. A. ZAPHYR, "Pros and Cons of Remote Computing," *Control Engineering,* January 1963.
5. G. L. BALDWIN and N. E. SNOW, "Remote Operation of a Computer by a High Speed Data Link," *Proc. F JGC,* December 1962.
6. C. STRACHEY, "Time Sharing in Large, Fast Computers," *Proceedings of the International Conference on Information Processing-UNESCO,* June 1959.
7. J. McCARTHY, "Time Sharing Computer Systems," *Management and the Computer of the Future,* Chapter 6, John Wiley & Sons, Inc., 1962.
8. F. J. CORBATO, "An Experimental Time Sharing System," *Proc. SJCC,* May 1962.
9. F. J. CORBATO, et al., "The Compatible Time Sharing System-A Programmer's Guide," The M.I.T. Press, May 1963.
10. W. V. CROWLEY, "Why Stretch?" *Proc. ACM National Con/ erence,* September 1962.
11. S. BOILEN, E. FREDKIN, J.C. R. LICKLIDER, and J. McCARTHY, "A Time Sharing Debugging System for a Small Computer," *Proc. SJCC,* May 1963.
12. W. CLARK and J. C. R. LICKLIDER, "On-Line Man-Computer Communication," *P1·oc. SJCC,* May 1962.
13. J. C. **R.** LICKLIDER, "Man-Computer Symbiosis," *IRE Transactions on Human Factors in Electronics,* March 1960.
14. L. C. CLAPP and R. Y. KAIN, "A Computer Aid for Symbolic Mathematics," *Proc. FJCC,* November 1963.
15. H. TEAGER and J. McCARTHY, "Time-Shared Program Testing," *Proc. ACM National Meeting,* September 1959.
16. C. N. MOOERS, "The Reactive Typewriter," *ACM Communications,* January 1963.
17. G. J. CULLEN and R. W. HUFF, "S lution of Non-Linear Integral Equations Using On-Line Computer Control," *Proc. WJCC,* April 1962.
18. T. MARILL, D. EDWARDS, and **W.** FEURZEIG, "DATA-DIAL: Two-Way Communication with Computers from Ordinary Dial Telephones," *ACM Communications,* October 1963.
19. R. HEAD, "The Programming Gap in Real Time Systems," *Datamation,* February 1963.
20. **W. A.** HOSIER, "Pitfalls and Safeguards in Real Time Systems," *Datamation,* April 1962 and May 1962.
21. T. A. HALDIMAN, "Management Techniques for Real Time Computer Programming," *ACM Journal,* July 1962.
22. **W. FRANK, W. GARDNER, and G. STOCK,** Programming On-Line Systems," *Datamation,* May 1963 and June 1963.
23. D. ISRAEL, "Simulation Techniques for the Test and Evalution of Real Time Compiler Programs," *ACM Journal,* 1963.
24. R. HEAD, "Real Time Programming Specifications," *ACM Communications,* July 1963.
25. FORTRAN General Information Manual, TRM H'nl'"n"\ l\Tnl"V\ho:r M'9Q Q0'7A
26. IBM 7040/7044 General Information Manual, IBM Form Number D22-6645.
27. IBM 1301 Disk Storage, IBM Form Number D22-6576.
28. IBM 7320 Drum Storage, IBM Form Number G22-6717.
29. IBM 7740 Communications Control System, IBM Form Number A22-6753.
30. IBM 1050 Data Communications System, IBM Form Number A24-3020.
31. H. FERGUSON and E. BERNER, "Debugging Systems at the Source Language Level," *ACM Communications,* August 1963.
32. M. WILKERSON, "The JOVIAL Checker," *Proc. W JGC,* April 1961.
33. H. SCHWARZ, "An Introduction to ALGOL," *ACM Communications,* February 1963.
34. G. M. WEINBERG and G. L. GRESSETT, "An Experiment in Automatic Verification of Programs," *ACM Communications,* October 1963.