RPL, A data reduction language

by FRANK C. BEQUAERT*

Computer Research Corporation
Newton, Massachusetts

INTRODUCTION

In the MITRE Interferometer Radar System, the outputs from a number of signal processors are recorded in digital form on magnetic tape utilizing an SDS-930 computer. A large number of these data tapes are recorded during system tests and observation of satellites. Data processing programs must be continually written to reduce these data to a form suitable for the analysis of radar performance.

In the past, the time taken from the specification of a data reduction program until an operational model was available was usually on the order of weeks for the simpler routines and months for more sophisticated programs. In addition, even with the use of FORTRAN, this programming involved a great deal of repetitive effort by the programmers.

With these problems in mind, in December 1965 development began on RPL (Radar Processing Language), a programming language to facilitate the writing of data reduction programs. The objective was a language that would produce useful data reduction programs from a typewriter conversation between a user and the computer. At this writing, a production model of the program is operational with the ability to generate relatively sophisticated data reduction programs.

Design objectives

The major objective in writing RPL was to produce a programming system that would automate as completely as possible all of the routine programming jobs normally associated with the MITRE radar data reduction operation. Specifically, it was desired that the program would be able to generate programs to perform the following functions:

1. Read data from records on a magnetic tape, unpack these data and store them in an organized manner in computer core.
2. Edit specified data values.
3. Generate simple functions of these data.
4. Print values of these functions in readable form.
5. Provide a method of plotting arrays of data.
6. Perform curve fits to arrays of data.
7. Perform correlation between sets of data.

Simplicity of operation was another design goal. It was desired that once the system was set up on the computer, an engineer with little knowledge of programming or machine operation could generate and execute useful data reduction programs through a typewriter conversation with the computer. It was hoped that the user would be able, in the course of an hour on the computer, to generate a program, test the generated program, and make one or more modifications (with subsequent tests) to the original program.

Finally, it was hoped that RPL would be directly useful to other facilities having recorded data to be processed.

Realization of objectives

Except for the correlation of data sets, all of the seven program generation features given above are currently operational in RPL.

The desired simplicity of operation was not fully realized. During the development of RPL it became obvious that the user of the system would require some knowledge of a computer language and of general concepts of computer programming. The specific areas of knowledge necessary to use RPL effectively are as follows:

1. A knowledge of the general concept of data arrays and of a notation for representing such arrays.

*Formerly with The MITRE Corporation, Bedford, Massachusetts.

"The work described in this paper was supported by the U. S. Air Force under Contract No. AF19(628)5165 monitored by the Electronic Systems Division of the Air Force Systems Command."
2 The knowledge of how to write FORTRAN-like arithmetic statements and the notations for such statements.

3 An understanding of the general concept of "flow" of operations in a computer program and the ability to generate simple program flow diagrams.

4 A knowledge of the difference between fixed and floating point numbers and their use.

Some acquaintance with the first three of these areas would seem requisite for programming in any language designed to manipulate data, no matter how "user orientated" it might be. For RPL on the SDS-930, knowledge of fixed and floating point notation is necessary to permit the user to conserve core storage in generated programs as fixed point arrays require half the storage space for the equivalent floating point arrays.

In general, standard FORTRAN notation is used for the representation of arrays and arithmetic operations as the generation of FORTRAN output statements from this input is extremely simple and most of the RPL users have had some FORTRAN programming experience.

**Operation of RPL**

RPL is a pre-compiler written in FORTRAN that generates FORTRAN output statements on magnetic tape. The program allows the use of a data base dictionary. This dictionary is used to specify the position and meaning of data recorded on a digital data tape. This dictionary is punched on cards. When a dictionary card deck is read by RPL, the program produces two outputs:

1. A dictionary listing on the line printer that gives data item names, dimensions, and descriptions (see Figure 1).

2. FORTRAN FUNCTION statements at the beginning of the RPL generated program. These statements stipulate, for the subsequentially generated program, the locations (word number and bit positions) within a data tape record where desired data may be found.

Once the dictionary deck has been generated for a particular class of data tapes, the user of RPL, no longer need concern himself with the position of data words within records on a data tape. The RPL dictionary system is capable of handling a wide variety of digital data tapes in which a particular piece of data on a tape is located by its position in a tape record. RPL will not currently handle tapes in which data is identified only by message labels.

RPL incorporates a number of program generation routines which generated sequences of FORTRAN output statements after typewriter conversation between the user and the computer. Currently the following types of program segments may be generated in this fashion:

1. **Input Variable Request (ASK).** A segment that requests a numerical input variable from the typewriter is generated in the output program.

2. **Data Tape Reading (GET).** A program segment is generated by GET that reads records of recorded data and generates arrays of functions of these data in computer core ready for printout, plotting, or further processing. Specified editing of data may be performed on the generated functions. The editing feature may be used either (1) to eliminate odd, spurious values from the generated data array or (2) to search for a specified value on a data tape.

The printout resulting from the use of the RPL GET function for a typical application is given below. The questions directed to the user and his responses are given at the left of the printout. The output FORTRAN statements generated by GET are preceded by asterisks.

```
GET
ENTER TOTAL NO. OF ROWS OF DATA TO BE STORED IN TABLE
100
ENTER NO. OF FLOATING POINT ARRAYS TO BE MANIPULATED BY GET
1
ENTER FLOATING POINT ARRAY 1 NAME TIME
IS TIME SINGLY DIMENSIONED?
YES
ENTER NO. OF FIXED POINT ARRAYS TO BE MANIPULATED BY GET
1
ENTER FIXED POINT ARRAY 1 NAME NUMBER
IS NUMBER SINGLY DIMENSIONED?
NO
ENTER SECOND DIMENSION SIZE
5
THE ARRAYS AS NOW DIMENSIONED REQUIRE 700 LOCATIONS OF CORE.
IS THIS SIZE ACCEPTABLE?
YES

********** CGET
********** 8000 DIMENSION TIME [ 100]
********** 8001 DIMENSION NUMBER [ 100, 5]
********** 8002 IQXCT = 0
********** 8003 CALL REED 5 [IWORD, 2925, IEOF, [N24,NPASS]
********** 8004 NQXROW = NROW[1DUM]
********** 8006 DO 8005 NROW = 1,NQXROW
********** 8007 IQXCT = IQXCT + 1
ENTER EXPRESSION FOR TIME AS A FUNCTION OF NROW.
ISEC [NROW] + IMIN [NROW]*60
********** 8008 TIME [IQXCT] = ISEC[NROW] + IMIN[NROW]*60
```
ENTER EXPRESSION FOR NUMBER AS A FUNCTION OF \([\text{NROW}, \text{NCOL}]\)

\[
\text{IVAL} \quad [\text{NROW}, \text{NCOL}] = 1.5
\]

\[
\text{IVAL} \quad \text{NROW} = \text{NCOL} \quad \text{IVAL} \quad [\text{NROW}, \text{NCOL}]
\]

AFTER VARIABLES ARE EXTRACTED, THEY WILL BE EXAMINED FOR ANY SPECIFIED EDITING.

IF ANY VARIABLE IN A ROW FAILS THE EDITING, THE ENTIRE ROW WILL BE DISCARDED.

DO YOU WISH TO EDIT DATA?

YES

ENTER ARRAY NAME ON WHICH YOU WISH TO EDIT TIME

TYPE LT IF YOU WANT TO DISCARD DATA LESS THAN A VALUE

TYPE EQ TO DISCARD DATA EQUAL TO A VALUE

TYPE GT TO DISCARD DATA GREATER THAN A VALUE

AND TYPE BET TO DISCARD DATA BETWEEN TWO VALUES

EQ

ENTER VALUE YOU WISH TIME VALUE COMPARED WITH

0

DO YOU WISH TO EDIT DATA?

NO

WOULD YOU LIKE SENSE SWITCH TEST TO GIVE OPTION OF EXITING BEFORE 100 ROWS ARE PROCESSED?

NO

3 Data Printout (PRINT). The PRINT function may be used to generate a program segment to print arrays of data on the line printer. The user specifies any general table heading plus column headings he wishes printed, the names of the variables to be printed and the number of decimal places to be printed.

4 Plotting of Data (PLOT). Program segments to plot data arrays on the line printer may be generated by the RPL PLOT function. Two types of plots are available.

1. A plot on a single page of line printer paper.

2. A multiple page plot in which the independent variable runs along the edge of the printer paper and a single or double plot covers the width of the printer page.

5 Curve Fitting to Data (FIT). The FIT function in RPL generates output program segments that will perform least squares curve fits to arrays of data. The user specifies the names of the independent and dependent variable arrays on which the curve fit is to be performed, the number of points to be fitted from the arrays, the number of coefficients of the curve fit and the name of the coefficient array.

RPL is equipped with a number of features to permit the updating of old RPL generated programs and the modification of new program statements during program generation. The more significant of these features are as follows:

1. The user may copy any portions of a previously generated FORTRAN statement tape onto a new statement tape.

2. The user may backspace a statement tape under generation to the beginning of any specified statement.

3. Any errors in typewriter input that are detected before a line has been read into the computer may be corrected by hitting an error key and retying the line.

4. Complete listings of either new or old FORTRAN statement tapes may be generated at any time.

5. At any time, by typing "EXIT" the user may exit from the middle of any program generation function back to general RPL control.

Features that allow additional flexibility of operation are as follows:

1. FORTRAN statements typed on the console typewriter may be written directly on the output statement tapes.

2. In all cases, punched card inputs may be substituted for typewriter inputs.

3. A complete record of all operations is kept on the line printer. This record gives computer generated typewriter output, users responses, and generated FORTRAN statements.

When the user has completed generation of a FORTRAN program with RPL, control is returned to the SDS MONARCH executive system. If the correct control cards are placed in the card reader, it is possible to compile, load and execute the generated program without user intervention.

RPL design

Initially, it was decided to write RPL as a pre-compiler written in FORTRAN which would generate FORTRAN output. This approach was taken because:

1. The SDS MONARCH executive system for the SDS-930 provided an automatic operating system into which a FORTRAN pre-compiler could be embedded.

2. The general form that FORTRAN data reduction programs should take was well known from previous programming experience.

3. Personnel with extensive FORTRAN programming experience were available.
Many of the programs generated by RPL would be kept as "production" programs for subsequent use. This requirement demanded that a relatively efficient and easily stored object program be produced. It was obvious that this objective could be realized by a pre-compiler.

The design of the RPL language was influenced by the available SOS-930 computer equipment. The MITRE SDS-930 system had neither a card punch nor a paper tape punch. As a result, all computer generated output that was to be subsequently used as computer input had to be written on magnetic tape. Thus, FORTRAN statements generated by RPL are written on magnetic tape and features are provided in RPL that allow for the reading of an old RPL generated FORTRAN statement tape and the generation of a new tape with deletion and insertion of FORTRAN statements. There is, however, nothing in the basic design of RPL to prevent a simple modification of the system to provide FORTRAN output on cards or paper tape.

The data base dictionary is read from a card deck by RPL at program initiation. The card deck consists of one record definition card followed by any number of data definition cards. The record definition card supplies the program with a description of the general makeup (record length, etc.) of the records to be read from the data tape. Each data definition card contains information (e.g., word and bit position) concerning one piece of data within the record.

As each data definition card is read by RPL a corresponding FORTRAN function is generated at the beginning of the output statement tape. A typical such function would be:

\[
\text{NAME(NROW)} = \text{IXTRAK(IWORD)}
\]

\[
(93 + (NROW-1)*193), 2, \phi)
\]

Here IXTRAK(NTABLE, NBITS, NPOS) is a machine language coded FORTRAN FUNCTION that extracts NBITS bits starting at bit position NPOS from core location NTABLE. IWORD is the array into which a tape record is read. NAME is the defined name of the data variable as specified on the data definition card. The numbers (93, 193, 2 and 0 in the above example) that specify the location of the data within the record (IWORD) read from the data tape are obtained from the record and data definition cards and inserted in the FUNCTION statement by RPL. With these FUNCTION statements at the beginning of the generated program, any reference to the specified variables (NAME in the above example) in the body of program will result in the extraction of the appropriate bits from the correct work of a data tape record. The variable names plus a definition (as specified on the data definition cards) are printed as a dictionary listing for reference by the RPL user (See Figure 1).

---

**RPL Dictionary As Follows For UTAPE As Specified In Rot Document 11/29/65.**

**Use NROW To Indicate Row In Table.**

**Use IVAL To Indicate Which Value In An Array Of Identical Values.**

**Variable Name**

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>NREC[NDUMMY]</td>
<td>UTAPE RECORD NUMBER</td>
</tr>
<tr>
<td>NROW[NDUMMY]</td>
<td>NUMBER OF ROWS OF DATA IN UTAPE RECORD</td>
</tr>
<tr>
<td>TTM6 [NROW]</td>
<td>TIME IN MICROSECONDS</td>
</tr>
<tr>
<td>TTM5 [NROW]</td>
<td>TIME IN 100 OF MICROSECONDS</td>
</tr>
<tr>
<td>TTM4 [NROW]</td>
<td>TIME IN 1000 OF MICROSECONDS</td>
</tr>
<tr>
<td>TTM3 [NROW]</td>
<td>TIME IN MILLISECONDS</td>
</tr>
<tr>
<td>TTN2 [NROW]</td>
<td>TIME IN 100 TH OF A SEC.</td>
</tr>
<tr>
<td>TTM1 [NROW]</td>
<td>TIME IN 10TH OF A SEC., IN MICROTIME WORD</td>
</tr>
<tr>
<td>TTSEC [NROW]</td>
<td>TIME IN 10TH OF A SEC., IN MACROTIME WORD</td>
</tr>
<tr>
<td>LAZIMSC [NROW]</td>
<td>AZIMUTH OF BEDFORD ANTENNA (COMMAND) IN .044 DEGREES</td>
</tr>
<tr>
<td>ELEVIMSC [NROW]</td>
<td>ELEVATION BEDFORD ANTENNA (COMMAND) IN .044 DEGREES</td>
</tr>
<tr>
<td>SDPRM [NROW]</td>
<td>SDF RANGE FOR MILLSTONE IN MICROSECONDS</td>
</tr>
<tr>
<td>SDPFM [NROW]</td>
<td>SDF FILTER BANK NUMBER FOR MILLSTONE</td>
</tr>
<tr>
<td>SDPPFNM [NROW]</td>
<td>SDF FILTER NUMBER FOR MILLSTONE</td>
</tr>
<tr>
<td>SDPSIN [NROW,IVAL]</td>
<td>SDF SIN OUTPUTS OF INTEGRATOR MILLSTONE</td>
</tr>
<tr>
<td>SDPCOS [NROW,IVAL]</td>
<td>SDF COS OUTPUTS OF INTEGRATOR MILLSTONE</td>
</tr>
</tbody>
</table>

**Figure 1 - RPL dictionary listing**
Programming of RPL

Programming of RPL and its program generation functions were found, in general, to be considerably easier than was initially expected. Approximately six-man-months of work by experienced FORTRAN programmers were required to write and debug the current RPL system. The FORTRAN program segments generated by RPL use library subroutines whenever possible for such operations as bit extraction, data plotting and curve fitting. Most of these subroutines had been previously developed and used extensively in hand coded FORTRAN data reduction programs. The use of these subroutines greatly simplifies the output program generated by RPL.

CONCLUSIONS

As of this writing RPL has been successfully used to generate a number of complete data reduction programs. In one case, a program consisting of 300 FORTRAN statements was generated and debugged in a single three hour period on the computer. It is estimated that the programming and debugging of this routine would have required at least two weeks of effort if hand coded in FORTRAN by a programmer experienced in the writing of data reduction programs.

In addition, RPL has been used to generate initial models of programs which were later greatly expanded by the addition of hand coded FORTRAN statements. This technique eliminates a great deal of the initial dogwork from the programming process.

A data reduction language of the form of RPL should be of value to a medium scale computer installation involved in the reduction of recorded data. It should be of particular value to an installation where the format of the recorded data and/or the required processing of data are subject to frequent change.

The size of the installation on which RPL will work effectively is probably critical. On a small system (i.e., less than 12K of core) it would become difficult to find a subset of RPL with reasonable enough flexibility. On a small machine the user would also be seriously limited as to the space available for generated programs. RPL is somewhat lavish with computer storage assigned by the generated program although the user is informed during program generation as to the size of any large blocks of storage assigned.

On a large scale computer, the cost of operation of the machine may prohibit sole use of the facility by a single program for hours at a time. In this case, it is possible to consider the use of some type of time-sharing. There should be relatively little difficulty in adapting a system like RPL to operate in a "real-time" environment in which a program under interrupt control (e.g., data recording) has priority over RPL operations, provided the necessary hardware and software (e.g., a real time monitor) are available for the machine in question. The operation of RPL within the framework of a more extensive time sharing system is another question. It is not clear for example, if the pre-compiler approach is the correct one for such an environment.

ACKNOWLEDGMENTS

The author wishes to express his sincere appreciation to Mrs. Patricia Grassler and John Xenakis who participated in the design of the RPL system and wrote a major portion of the program.