THE REPORT PROGRAM GENERATOR

by
Harry Leslie

Harry Leslie is a Staff Analyst in the New York office. He has been in the computing field since 1962 and joined CUC in 1963. His experience includes both systems and commercial programming. He was project leader for the design and implementation of RCA's RPG and has also worked on a COBOL compiler, a debugging system, conversion and utility programs.

Harry, who received his B.A. from Columbia College, lives with his wife and their three children in Forest Hills, N. Y.

The shortage of programmers in the United States, or the "software crisis" as it has been dubbed by Dr. Hurd, is producing fallout among practicing professionals today. Writing programs is fascinating and rewarding work but the functions of many programs are identical and rewriting these sections can not only become tiresome but also can get a programmer bogged down on one assignment and unable to move quickly onto a new one.

The Report Program Generator is providing a means, on many programs, for programmers to move more rapidly from one program to another. The purpose of this article is to describe some useful areas of the new generation of RPG processors and to briefly explain the language processor we developed for RCA and its SPECTRA 70. IBM introduced the language on the 1401 series and has also developed a new processor for the System 360. The same RPG language has also been adopted by UNIVAC. Perhaps this means the beginning of an industry standard in RPG language.

Programmers who have never worked with RPGs receive a mild shock when they find the issue neither reads nor writes, and they do not have to move input records to work areas, define special work sections or move fields to output records. Gone then, is a sizeable part of the cause of the programmer's occupational disease, writers cramp. The RPG accomplishes this by taking advantage of the fact that an output record is always made up of previously defined fields, edited fields, or constants. When an output field is specified, it is merely given the same name as the field wanted moved and the RPG moves it.

The RPG has predefined program logic variable enough to fit most applications, which allows it to control the I/O and to determine when to issue a read or write. Most data processing jobs will fit into this program logic (some being squeezed in). As general rule, you could say that the less complicated the file handling problems the more suitable a program is for the RPG. For example, the RPG does not allow work tapes within a program (e.g. tape may not be written on during the first part of a job and read in the second part, this would have to encompass 2 separate RPG programs). Also, the more input files there are to process, the more complicated it is for a programmer to work with the RPG.

Let's pin down the usefulness of the RPG a little bit more. The way the RPG operates, one could not ask "should we use assembly language or RPG" or again "COBOL or RPG". Instead, RPG should be used in addition to some other language, and the question asked should be "when do I use COBOL or assembly language and when RPG".

Six pertinent questions must be applied to the RPG, with the answers telling us not only if it should be used at all, but if so, when:

1. Is it easy to learn?
2. Is it easy to use?
3. Can a program be written quickly in it?
4. Does it produce a fast running program?
5. Does it facilitate conversion?
6. Does it facilitate maintenance?

1. Is it easy to learn?

Most programming languages tend to be about equally difficult to learn. The RPG is no exception to this. It is as difficult to learn as any other programming language.

2. Is it easy to use?

Ease of use depends on the application. The RPG is intended for straightforward applications with-out out complicated logic or complicated use of files. If it is applied where intended, it is certainly easier to use than either COBOL or assembly language. File Description statements are short and effective. One RPG source statement completely defines a file's structure. Input and output field statements also require only one line, giving the from and to address, decimal positions and name and certain other optional features associated with the field (e.g. a programmer could tell the RPG to test if the field is plus, minus, zero or blank). The calculations tend to be fewer than with COBOL or Assembly Language when used in straightforward situations. No file control logic is included in the statements so one source of error and several lines of coding are eliminated.

3. Can a program be written quickly in it?

Once again, if the application is proper, the speed with which something can be programmed is amazing.

For instance, a two tape merge could be written with ten RPG cards in less than half an hour by an experienced RPG programmer. An optional printed report with proper spacing and headings at certain level breaks could be added with perhaps 10 to 20 RPG source cards, also in less than half an hour. The instruction set is comprehensive, featuring a powerful table lookup instruction and including a variety of moves and a compare, add, subtract, multiply and divide with automatic decimal alignment. In addition, it allows reference to linkage to programs external to the RPG.

4. Does it produce a fast running program?

The RPG does not produce an efficient program, but it does usually produce a program that will run as fast as one written in any other language. This is because the execution time will normally fall completely within I/O time, eliminating the necessity of efficiency. Again, we see the need for using the RPG for straightforward problems. A long, complicated problem involving many loops may turn into a slow running program in RPG, for the execution time might exceed I/O time.

5. Does it facilitate conversion?

In the absence of an industry standard for RPGs, conversion will probably mean rewriting. However, all RPGs tend to be very similar and it would certainly be much easier to rewrite than an assembly language program. In fact, it is not a difficult thing to write a program to convert from one RPG to another.

6. Does it facilitate maintenance?

   An RPG program will normally have fewer statements than either assembly language or COBOL. For that reason there will be fewer statements to change if maintenance is necessary. Also, it is quite simple to make additions to an RPG program. However, a complicated problem coded in RPG is as difficult to maintain as, the same problem coded in COBOL or Assembly Language.

The RPG can obviously be a very useful language if used in addition to a more comprehensive language. There are many applications of the RPG and it deserves a place in any data processing shop.

I have drawn a very vague dividing line between the times when the RPG should be used and when it should not. The reason is that no clear dividing line exists. It is possible, though, to apply 3 general questions to a data processing program.

1. Does the input or output structure include multi-file reels, work tapes or special tape control operations?

2. Is the relationship between input files complicated?

3. Does the program include difficult logical processing or many loops?

If the answer to each is no, then the program is probably suitable for the RPG.

The way a processor works can be as important as what it does. For instance, a very slow compiler or one that gives few diagnostics can be pretty useless. The RPG CUC wrote for the SPECTRA 70's POS, TOS and TDOS systems set out to accomplish three major objectives -

1. Fast compilation
2. Efficient object programs
3. A clear comprehensive program listing.

We produced a compiler that will output loadable programs at rates up to 900 cards per minute, with tailored object code, optional double buffered I/O and a printout with calculation object code listed with each source line plus any diagnostics (over 200 possible diagnostics). Listing the object code with the calculation statement is especially appreciated by users who have program errors and need to know in which statement or field an interrupt occurred.

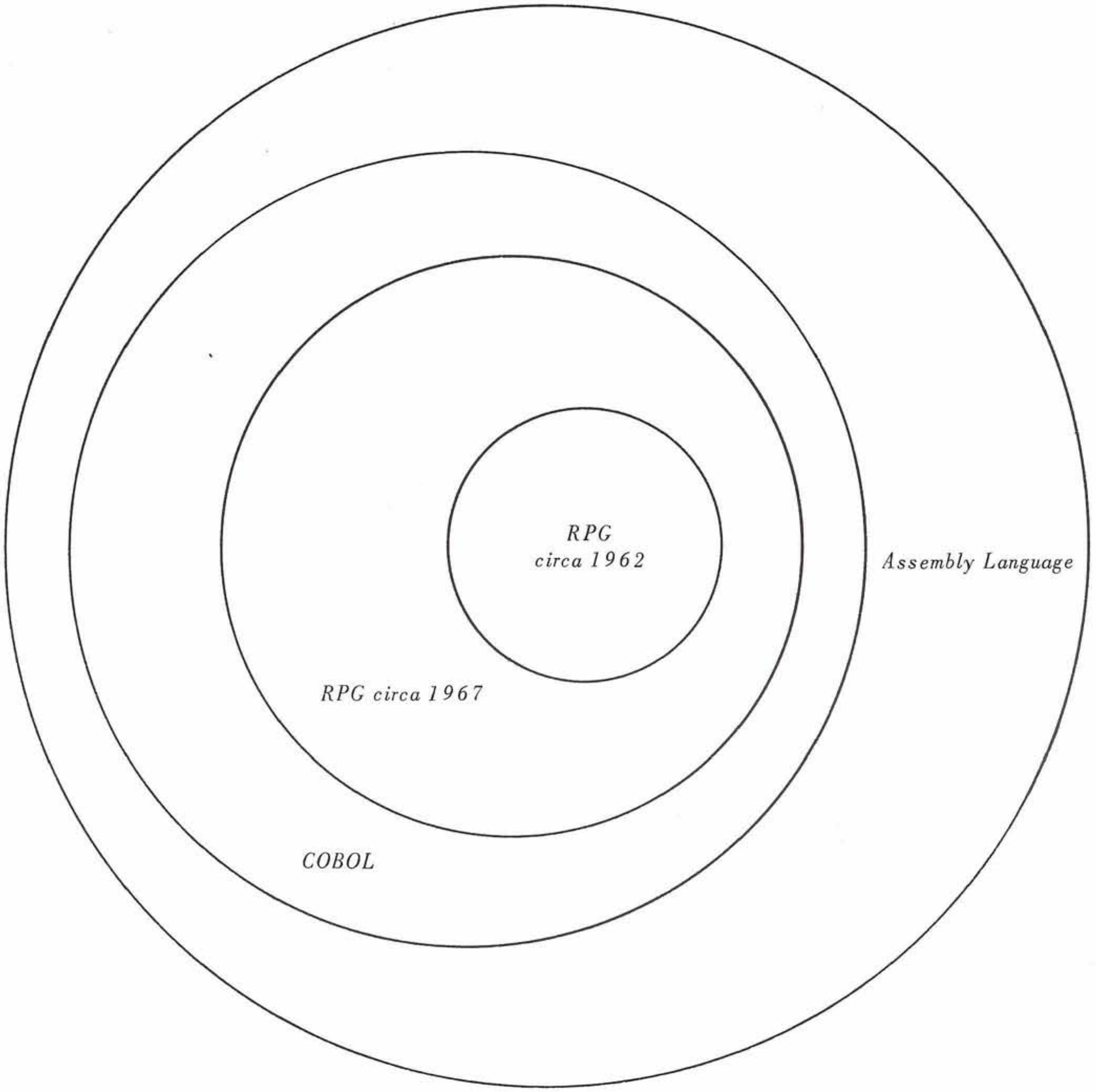
One of the most interesting things to come out of the RPG processor was a new method of handling binary table searches. Previously, a binary table lookup with a table length not a power of two, involved a division each time through the loop. However, the object program the RPG produced is capable of doing a binary search of any size table, using a shift instruction instead of division. This increases the speed of the lookup many times, as the shift instruction on the SPECTRA 70 (or the 360) is usually about 10 times as fast as a divide.

The method simply involves dividing the table into 2 overlapping parts, each a power of 2, in number of entries. A compare to the midpoint off the table to find out which area of the table to search is the only overhead paid for this method. The increment (or decrement for tables in descending order) is kept in a register and each time through, the loop is halved by a shift of one bit. The time saved could be significant - probably half a milisecond per lookup on a SPECTRA 70 model 45.

The RPG processor itself operates in a fairly typical fashion. It breaks source statements down into either data attribute or a codeified form of a statement. After data attributes have been submitted for names, it generates object code in an unfinished format. Then this code is outputed in a form acceptable to the linkage editor. Although this sounds as though it is passing over the input four times, (and in a sense it is) the actual compiling speed is limited mainly by the original input speed and final output speed.

Speed is achieved because only the data that need be passed over again is actually written and read again. Since the needed coded data will occupy only one quarter to one tenth the room of the original input, large savings are made. In fact, only two and a half passes actually take place. This brings us to a point where processing speed is determined mainly by the speed of the input and output units. For instance, a twelve hundred card RPG program using card in, printer out and producing an object program on tape, will take a little more than a minute to read the cards; two and a half minutes to print out; (with object code and diagnostics 2½ times as much is printed as is read in) and less than half a minute for immediate processing. Total time is a little under four minutes.

We accomplished what we set out to do in the RPG for RCA, and, perhaps, most important of all, we did it on time and within the budget.



The tasks accomplishable by RPG have expanded greatly over the past several years to the point where today it deserves special consideration.