Management Services: A Magazine of Planning, Systems, and Controls

Volume 7 | Number 1 Article 6

1-1970

What a Financial Manager Should Know About COBOL and Assembly Language

David K. Banner

If a company man is chosen over a computer expert to run the EDP installation, how much should he know about computers? At least this much about the software used with them, the author maintains —

WHAT A FINANCIAL MANAGER SHOULD KNOW ABOUT COBOL AND ASSEMBLY LANGUAGE

by David K. Banner
Peat, Marwick, Mitchell & Co.

A QUESTION still occupying hours of seminar and discussion time is this: Who is best qualified to run an EDP installation, the experienced company man or the qualified EDP technician? And, assuming this question is resolved in favor of the company executive, the inevitable next question arises:

How much technical knowledge does he need?

Our answer would be that he needs comparatively little provided he has the right EDP technicians working for him and has the proper communication lines open to them. He does not need to know how the machines do what they do, but he does definitely need to know what they can do. And he certainly needs to understand enough of the specialized EDP language so that communications do not constantly bog down between his specialists and himself.

We would suggest, too, that a modern financial manager must be aware of the relative advantages of using COBOL (Common Business Oriented Language) or an assembly language in a given application. Both languages have relative strengths and the enlightened manager must exploit these to maximize his EDP effectiveness. This, then, is the purpose of this article: to give the financial manager a feeling for the characteristics of these computer languages as they might influence his operating environment in hopes his decision making with regard to EDP will be more enlightened as a result.

First of all let's define our terms:

An EDP term of interest is "hardware," i.e., the machine itself with all its peripheral equipment. The term "software" refers to a wide variety of programing aids designed to save time, money, or both by reducing personnel time at the expense of machine time. A program, i.e., the instruction for a specific application, is a part of the software. Programs can be coded in machine language, assembly language, or compiler language, but the choice of language can mean quite a lot in terms of machine efficiency, as we shall see.

To give a frame of reference for this subject: We feel that several additional key definitions are in order. What is an assembly language? An assembly language is made up of symbolic statements; the assembler, i.e., the language processor, converts a single symbolic statement from a source program into a single binary or machine statement (see Exhibit 1 on page 38). Obviously, this language processor offers little leverage since the one-for-one conversion does not substitute "machine time" for "people time." In this light, assembly languages can be said to require more human skills and less machine "skills."

The compiler, on the other hand, is a language processor with great leverage. The compiler expands each macro-code source statement, e.g., move, add, multiply, etc., into several machine statements. (It creates, in other words, a macro-code language, in which each statement or comment incorporates several instructions.) Two familiar examples of compiler languages are COBOL (business-oriented) and FORTRAN (primarily for scientific applications). Another factor distinguishes compiler languages from assembly languages. The compiler source statements emphasize the steps in problem solution rather than being concerned, as are the assembly language statements, with the symbolic expression of machine language (see Exhibit 1).

EXHIBIT I

FOUR LANGUAGES

DESCRIBING ONE PROCESS

ENGLISH

Multiply The Unit Price By The Quantity To Get The Total Price.

COBOL

Multiply Unit-Price In Old Master By Quantity Giving Total Price.

ASSEMBLY LANGUAGE

```
LX,M      X8,OLDMAS      }
LA        A5,UNITPR, X8  }  SLEUTH II
MSI       A5,QNTY        }  UNIVAC 1108
SA        A5,TOTPR       }
```

MACHINE LANGUAGE

```
BINARY                                         OCTAL
0101111110100000000001000100001010001          277200104
01000100000010110000000000000100011100         210160000
0110100000010100000001011010011100110          320120132
0000010000001010000000111000111111001          010120161
```

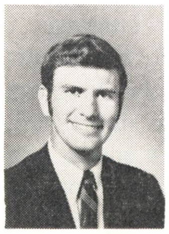
History of COBOL

COBOL was designed to satisfy a need for standardization of coding in business data processing applications. In May, 1959, a Pentagon meeting of users, manufacturers, and other interested parties was called to consider the desirability and feasibility of a common business source language. This meeting, called CODASYL (Conference on Data Systems Languages), developed the specifications for COBOL. A short-range committee was appointed to develop the source language from these specifications. COBOL-60, the first edition of COBOL, was the result. A maintenance committee was set up to initiate and review recommended changes to keep COBOL up to date. COBOL-61, published in 1961, reflected its changes.

COBOL was not an instant success. At first it was plagued with excessive compile times, the generation of inefficient object coding, and excessive core requirements. In the last several years, however, COBOL has undergone a maturing process. The major deficiencies have been overcome: Compile times are good; core requirements have been reduced to workable limits; and user techniques have become more effective. Many manufacturers now offer COBOL compilers with their equipment (see Exhibit 2 on page 39). It is easy to see that COBOL has gained widespread acceptance. However, assembly languages still have many applications. In the next few paragraphs we shall evaluate the advantages and disadvantages of the two types of languages.

These two types of languages will be evaluated in terms of ten criteria (see Exhibit 3 on page 39).

Programer training time — COBOL is designed with the programer in mind; the organization of the language (English-like syntax) is simple. An individual doesn't need to know machine language or assembly language to code COBOL; however, as we shall see later, such knowledge will aid COBOL efficiency. Administrative personnel, e.g., financial managers, can improve their ability to com-



DAVID K. BANNER is a management consultant in the Houston office of Peat, Marwick, Mitchell & Co. He received his BS from the University of Texas at Arlington. He received his MBA from the University of Houston. Mr. Banner has been a guest lecturer on finance at the University of Houston. He has also been employed as an aerospace engineer at the Manned Spacecraft Center, NASA, in Houston.

EXHIBIT 2

EQUIPMENT FOR WHICH COBOL IS OFFERED

MFGR	MODEL (SERIES)
BURROUGHS	5500, 6500, 7500, 500 SYSTEM
CONTROL DATA	3000 SERIES, 6000 SERIES
GE	115,400 SERIES, 600 SERIES
HONEYWELL	110,200 SERIES, 400, 1400, 800, 1800
IBM	1401, 1410, 7070 SERIES SYSTEM/360
NCR	315, CENTURY SERIES
RCA	301, 3301, SPECTRA SERIES
UNIVAC	1050, 1107, 1108, 490 SERIES, 9000 SERIES

municate systems problems or desires to systems people by having a knowledge of COBOL.

Programing (coding) – COBOL coding time constitutes a major saving over assembly language coding. The compiler handles many of the details otherwise attended to by the programer.

Documentation – Both COBOL and assembly language can be documented, but, with COBOL, documentation is inherent in the language. This is a real plus for COBOL. A COBOL compilation, i.e., the conversion of a source language program into a machine language version, results in a printed listing of the COBOL program for documentation purposes. Another plus is that COBOL logic can be readily understood by personnel not familiar with the program.

Standardization – COBOL is called almost machine-independent by some people. This means that, with only minor changes, one can compile and execute a COBOL program on a machine different from the one for which the program was designed. This "machine independence" is somewhat overstated. Each machine has certain strengths and weaknesses (logic, data organization, etc.). A program written for a certain machine is likely to be better than a generalized program. Manufacturers have recognized this fact and now many of them offer several versions of the COBOL compiler. While COBOL may not be machine-independent, it is at least compatible within a manufacturer's product line.

Debugging – COBOL also has the advantage here. COBOL logic can be traced through with ease. During the compilation phase, COBOL generates a list of statements called diagnostics which indicate all errors in the source program excluding logic errors.

Efficiency–As mentioned before, COBOL object programs, i.e., the machine language version, used to be inefficient. This was largely because inexperienced people had coded certain portions of the COBOL programs sloppily. Still, even today, COBOL cannot match assembly language in overall machine efficiency. However, this is not usually significant; the rated speed of the peripheral equipment is as fast as you can go anyway. So, if the peripheral equipment is running at less than top speed, the relative inefficiency of COBOL becomes academic.

EXHIBIT 3

EVALUATION OF COBOL AND ASSEMBLY LANGUAGE

I	PROGRAMER/TRAIN
II	CODING
III	DOCUMENTATION
IV	STANDARDIZATION
V	DEBUGGING
VI	EFFICIENCY
VII	MAINTENANCE
VIII	CORE
IX	COMPILE/ASSEMBLE
X	SOFTWARE

EXHIBIT 4

PRIMARY ADVANTAGES OF COBOL

DOCUMENTATION
MAINTENANCE
COMPATIBILITY
TRAINING
CODING

Program maintenance – COBOL offers an advantage here because changes are easier to make with separate data and procedure divisions. Program maintenance can sometimes develop into a full-time job for one man; hence, with COBOL, a relatively inexperienced man can do the necessary work.

Core requirements–COBOL requires more core for the user coding portion of a program than does a comparable assembly language program. In fact, the advantage for assembly language is sometimes as high as 2:1.

Compilation/assembly – Depending upon the efficiency of the compiler, assembly language is *usually* somewhat faster than COBOL in the compilation (or assembly) phase. However, tests have shown some COBOL compilers to be faster than assembly language in this respect.

Software–COBOL requires more time to execute when input/output bound (or process bound, for that matter). However, this can also be a moot point if the peripherals are not running at full speed.

COBOL, therefore, offers the following real advantages: documentation, maintenance, compatibility (or standardization), training, and coding as shown in Exhibit 4 above.

Compiler languages are the latest technique in the evolution of programing (see Exhibit 5 below). We can conclude, in the final analysis, that the advantages of the compiler language COBOL (in particular) outweigh the disadvantages. We can use the following decision rules in determining whether or not to use COBOL in an application:

If the program is to be run frequently, peripherals are running at rated speeds, and core is nearly used up, the use of assembly language is indicated.

If peripherals are running at less than maximum and core requirements are not restrictive, COBOL should be used (even if the program is to be run frequently). If the program must be revised frequently, COBOL is especially valuable.

One can plainly see the way computer programing is headed. As compilers get more efficient, they will be used more frequently in varied applications. Assembly languages will most probably be restricted in use to manufacturer software. Another apparent trend is toward the use of combined programing languages, i.e., the use of English-type words (where convenient) and the use of mathematical notation (where convenient). Still a third trend seems to be toward a conversational programing. This would be the ultimate compiler language. The computer (instead of the programer) would be asked to remember the language. The computer would display to the user various available alternatives with the results of each. The user would then select and change words until he had a good statement of the problem.

The contemporary financial manager can expect his job to get even more complex in the future. With a basic knowledge of various computer languages, he should be able to use his EDP "arm" to its fullest capability.

EXHIBIT 5

EVOLUTION OF PROGRAMING

$$\begin{tikzcd}
\text{MACHINE} \arrow[r] & \text{SYMBOLIC} \arrow[r] & \text{COMPILER} \\
\text{LANGUAGE} \arrow[r] & \text{SYSTEMS} \arrow[r] & \text{LANGUAGES} \\
\text{CODING} & &
\end{tikzcd}$$

$\left\{ \begin{array}{l} \text{ADDRESSES} \\ \text{OPERATION CODES} \end{array} \right.$