that the computer receives all the data that have collected on a drum field between readings, the normal procedure is to ask for more than expected. A disconnect pulse is automatically generated at the end of a complete drum revolution, and the number of words read is determined from the word counter contents.

Some of the data sources have messages which contain more information than will fit into one drum register. To handle such information, another feature has been added to the drums associated with these sources: the drum has been divided into multiple word slots of adjacent registers. For this application the only meaningful status bit is the one associated with the first register of a slot, and the source identity is contained in the first register of a slot. The operations associated with those drums are almost identical to those of the single register drums.

In addition to the input buffer drums just described, there are output buffer drums which handle outgoing data. They operate in a similar, though inverse, manner to the input drums.

Conclusion

Fig. 4 illustrates a portion of the computer in the test cell. Because of its size and layout, it was not possible to obtain a picture of the whole computer. The unit in the foreground is the operator's maintenance console. It contains switches for manual data or instruction entry and manual control, neon indicator lights for the major flip-flops and registers in the computer, visual and audible indicators for computer generated alarms, marginal checking controls and indicators, and power system and air-conditioning indicators. The cutout contains a view of one of the memory units.

The computer, including the directly connected input-output equipment, contains approximately 12,500 tubes. It has an execution time of 12 microseconds for arithmetic instructions, excluding "multiply" and "divide," which require 15.5 and 53 microseconds, respectively. The prototype model has been in satisfactory operation for more than one year.

Computer Design to Facilitate Linear Programming

R. C. GUNDERSON

AT the risk of being redundant, this paper will begin by stressing the growing importance of linear programming in business, industry, and government, as it is this importance which is its motivation.

Primarily, it is the application of linear programming with which the paper will deal. Optimal planning of procedures has become a necessity, rather than a luxury, to present-day management. For example, one organization is presently saving an estimated $20,000 a day by optimal planning through linear programming procedures. Without too great a stretch of the imagination, the fascinating possibilities of linear programming linked with automation might be pictured. This could yield factories staffed with skilled technicians to feed data from changing markets into computers, which would then choose the optimal combinations of specifications and direct the machinery to produce under these new specifications. There are countless other such possibilities which could make effective use of this powerful tool.

R C. GUNDERSON is with Remington Rand Univac, St. Paul, Minn.

It is not intimated here that the use of linear approximations is a new addition to mathematics or economics. Rather, it is the wider acceptance of their usefulness which is new. This, coupled with the fact that much work has been done in the past decade to develop a general formulation of the computational procedures involved in linear programming, presents an exciting facet of computer application to users and manufacturers.

It is evident, then, that some consideration should be given to the requirements of this problem in the building of our future computers. Essentially, the actual needs are for the most part familiar to the computer industry. Moreover, the logical properties of computers which this problem requires are extremely compatible with those desired by logicians.

Let some of the qualities of this problem which make it especially well adapted to high-speed digital computation be examined for a moment. First, since input and output time still lags behind computation time on all present-day large-scale computers, the relatively small amount of input, simple but voluminous computations and logical operations, and the small amount of output required, lend themselves well to computers. Second, the iterative nature of the matrix manipulations is ideally suited to stored program computation. Finally, the ability to generalize the procedure, enabling the solution of a number of maximization or minimization problems containing dissimilar data, reduces the programming involved to mere data preparation.

In the following, some of the essential physical properties of a computer which make handling linear programming problems more efficient will be discussed. Operating on matrices by rows or columns necessitates much greater rapid access storage than an element by element operation would require. Present-day problems, which undoubtedly will soon be dwarfed, require a minimum of 4,000 words, and would run much more efficiently with 8,000 to 12,000 words of rapid access storage. The so-called "housekeeping" operations required by the limitations of present storage systems increase running time by approximately one fifth.

The Simplex method of solution, probably the most efficient and most used linear programming procedure, requires access to the stored matrix of coefficients at random, as dictated by the computation. Moreover, the generation of an additional vector during each major iteration necessitates a large-capacity secondary storage in the more sophisticated problems. There is, then the additional requirement of a large, random access, secondary storage media, probably 15,000 to 30,000 words in size,

backed up by several hundred thousand words of magnetic tape storage.

The scaling and storing of the matrix elements, the packed floating vector representation of numbers, and the necessity of shifting for multiplication and division of scalars suggests the importance of addressable shifting registers with the possibility of both left and right shifts.

Since the solution process is iterative in nature, and since it is impossible to predict at any point the number of iterations necessary to reach a solution, it is essential that there be some facility for repeating general sequences. This may be accomplished by either or both of two methods, "b-boxing" or repetitive commands. Although b-boxes have some advantages, these seem outweighed by the versatility and flexibility of a command structure which accomplishes the same and possibly more. For example, the entire linear programming procedure is cyclic in nature, and within this major cycle are contained several minor cycles, each of which has subcycles, and in some instances sub-subcycles. So the possibility of an array of b-boxes approaching the number of rapid access storage cells required by the problem is presented. Furthermore, some thought should be given to the considerable cost of additional hardware required by b-boxing.

In addition to the afore-mentioned physical properties, some thought should be given to the command features most desirable for this problem. Let the implications of address structure be considered first. It is quite evident that 2-address logic has distinct advantages over single-address logic in so far as the mathematical operations are concerned. A significant reduction in the number of commands required to perform the arithmetic can be realized by combining several steps in one command. For example, in forming the sum of two vectors, it is necessary to set a component of one of the vectors in the sum register, add the corresponding component of the second vector to it, and store the result. With single-address logic, this would require three operations, but with 2-address logic the procedure may be accomplished by a single command. Similarly, in forming the scalar product of two vectors, it is possible with a 2-address structure to form the product of two corresponding vector components and add the result to the product of the preceding components in a single operation. In fact, by proper modification, the entire scalar product may be performed by a single command. The arithmetic instructions present one area of command structure where 3-address logic might be contemplated.

Evidence indicates that 2-address logic is also desirable for the logical operations required by the procedure. Although a certain amount of the "housekeeping" operations could be done rather efficiently by a single-address scheme, the extreme versatility of the 2-way conditional jumps allows a much more general approach to the problem with less housekeeping. Furthermore, the logical sums and products which prove very useful in the matrix and vector manipulations require two addresses to make their operation feasible. In no instance does it appear that 3-address logic could increase efficiency to any great extent.

In keeping with the logical structure of commands, the actual properties of the command for which linear programming, or any type of problem involving matrix or vector arithmetic, has a particular need will also be considered. As indicated previously, a command which may be easily modified is necessary to perform the sum of two vectors most efficiently. This command should be such that the sum of two corresponding components may be formed and stored in one operation. Similarly the scalar product of two vectors should be performed in such a manner that the procedure may be generalized without extensive housekeeping. It is quite evident that the more instructions that are required to perform these general and often used operations, the more presetting or resetting that becomes necessary. As a result of these first two arithmetic operations, a third requirement is evidenced, namely a method to rescale the resultant vector and scalar to their proper positions; this requires a command which will explore the resultant for the first significant digit of each component, and indicate in which position this digit appears.

Before investigating the properties of the logical commands required for an efficient linear programming procedure, it is necessary to have some understanding of the nature of the linear systems involved, and the methods of representing the numbers occurring in these systems. The matrix of coefficients of the original systems are generally rectangular, with a high incidence of zero coefficients. For greatest efficiency in storage and speed of arithmetic operations, some method of suppressing these zero elements seems to be indicated. For, as the user of any linear program gains experience and confidence, the dimensions of the systems increase to the point where time and storage space are of very real importance. Therefore, the problem of developing some form of number representation which will suppress zero coefficients and still preserve the significance of the nonzero elements of the matrix is presented.

Both of these may be accomplished by a type of floating point representation known as floating vector. This system allows 32 bits of significance plus a sign bit, and a characteristic or scale factor of 15 bits for each column of the matrix. Furthermore, there is no need for additional floating point arithmetic hardware, since each vector is operated on as a unit using the normal arithmetic operations, which are generally faster and more efficient. The use of such a representation is contingent of course upon the existence of a double length accumulator or sum register to prevent overflow in the scalar product of two vectors.

By giving each vector one, or several, keywords, which have digits indicating the position of nonzero components and zeros indicating the position of zero components of the vector, the significant elements only may be stored and all zero elements of the matrix ignored. For example, the vector

$$\mathbf{X} = (3, 0, 1, 0, 0, 5) \tag{1}$$

would have the binary keyword

$$101001 \tag{2}$$

and the vector would be stored thus:

Location	Binary Representation
A	101001
A + 1	000011
A + 2	000001
A + 3	000101

(3)

By a representation of this sort, both increased speed and reduced storage may be accomplished without losing the significance of the numbers involved.

The question now arises as to how this floating vector notation may most efficiently be used. First, the numbers must be packed and the vector keyword formed. To do this, each component of the vector must be tested for significance, a digit inserted in the keyword in the proper position, and the significant components stored. Second, there must be some method of interpreting the keyword in order to unpack the vector for some part of the procedure such as the vector sum. Finally, to reduce the amount of time consumed in forming the countless numbers of scalar products, the corresponding significant components of two vectors must be predetermined to preclude the possibility of multiplication by zero.

Again, caution should be exercised in determining the numbers of commands necessary to perform these operations so that a generalization of the procedure will not entail too extensive housekeeping.

The first of these logical problems may be solved by a number of different approaches. However, the second problem, the interpretation of the keyword for unpacking, requires an instruction which examines the word digit by digit and controls the storing of the significant components in their proper position in the vector, suggesting a type of conditional jump command. The zero suppression multiply may be accomplished by forming the logical product of the keywords of two vectors, forming a new keyword which would control the multiply sequence exactly as the unpacking procedure. To illustrate the advantage of this operation, let it be supposed there is the vector,

$$\mathbf{y} = (2, 1, 0, 4, 0, 0) \tag{4}$$

with the binary keyword,

$$110100 \tag{5}$$

and suppose forming the scalar product of $\mathbf{y}$ and the vector $\mathbf{x}$ given by equation (1) is desired. This would be accomplished by forming the logical product of the keywords (2) and (5)

$$\begin{array}{ll} (2) & 101001 \\ (5) & 110100 \\ \hline & 100000 \end{array} \tag{6}$$

indicating that the only corresponding significant components of the two vectors are the first, and the scalar product would then involve only a single multiplication rather than six multiplications and five additions.

It is estimated that a computer designed with all of the afore-mentioned qualities would reduce the running time of present procedures by as much as a factor of ten. Since most of the requirements of this problem are compatible with many other types of problems, and furthermore, since many of these features may be found on existing machines, such as the Univac Scientific, it is the contention in this paper that all of these requirements should be fully investigated so that they may be incorporated in the future machines of this industry.

Considerations in Making a Data-Gathering System Compatible

B. L. WADDELL

THE paper discusses the design problems facing the data systems engineers who are required to produce a data-collection system that will be able to enter a computer easily. Some empirical formulas are presented with a discussion of how to use these formulas. The many recording tools and their application to data systems which are being prepared for entry into computers are described with a discussion of the place of each recording device. The second section of the paper is a critical analysis of four data recording or gathering systems designed to go directly to a digital computer.

B. L. WADDELL is with G. M. Gianninni and Company, Inc., Pasadena, Calif.

Using a Variable-Word-Length Computer for Scientific Calculation

FRED GRUENBERGER E. H. COUGHRAN

IN discussing the use of a variable-word-length computer, this paper will be restricted entirely to past history; that is, ideas and practices that are actually in operation. This implies, of course, that only the *702* will be talked about and the implied comparison to fixed-word-length machines is to a machine like the *701*.

At Hanford, Wash., the *702* has been used for scientific computing, with what is regarded as considerable success, since its installation in June 1955. The percentage of available machine time devoted to numerical analysis has steadily increased, standing currently at about 20 per cent. This is not to say that mathematics is taking time away from commercial work, but rather reflects the increase in efficiency on commercial problems and a general increase in mathematical problems as time goes by. Scientific computing has covered a wide gamut of problems, including linear programming, numerical integration, differential equations, and many complicated formula evaluations, as well as work on reactor data and weather data which might be called scientific data processing.

This work demands one large machine to serve for both commercial and scientific problems. This is a common situation, and perhaps any machine would suffice; it has been pointed out that through an automatic programming system, any machine can be made to appear like any other. Indeed, most programming is done with reference to the automatic programming system rather than the particular machine. The main contention here is that a variable-word-length

FRED GRUENBERGER is with the General Electric Company, Richland, Wash., and E. H. COUGHRAN is with the International Business Machines Corporation, Richland, Wash.