Fingers or Fists?

(The Choice of Decimal or Binary Representation)

W. Buchholz, *International Business Machines Corporation, Poughkeepsie, N. Y.*

Abstract. The binary number system offers many advantages over a decimal representation for a high-performance, general-purpose computer. The greater simplicity of a binary arithmetic unit and the greater compactness of binary numbers both contribute directly to arithmetic speed. Less obvious and perhaps more important is the way binary addressing and instruction formats can increase the overall performance. Binary addresses are also essential to certain powerful operations which are not practical with decimal instruction formats.

On the other hand, decimal numbers are essential for communicating between man and the computer. In applications requiring the processing of a large volume of inherently decimal input and output data, the time for decimal-binary conversion needed by a purely binary computer may be significant. A slower decimal adder may take less time than a fast binary adder doing an addition and two conversions.

A careful review of the significance of decimal and binary number systems led to the adoption in the IBM Stretch computer of binary addressing and both binary and decimal data arithmetic, supplemented by efficient conversion instructions.

Motivation

The impetus for this study came from the design effort on the IBM Stretch computer, a computer aimed at the highest range of performance achievable with modern technology and systems organization. In order to arrive at a performance increase of two orders of magnitude, it seemed desirable to question and look carefully at the assumptions underlying the design of earlier computers. High arithmetic speed is, of course, a major goal of such a computer, and the number system is a factor in arithmetic speed. But the performance of a computer can no longer be measured by its arithmetic speed alone. The speed of non-arithmetic operations significantly affects the overall performance of a computer. Equally important is the human factor. Unless the computer is programmed to assist in the preparation of a problem and in the presentation of results, the false starts and waiting time can greatly dilute the effective performance of a high-speed computer.

The present study attempts to consider the implications of such factors in the choice of the number base. Although it was aimed at a specific computer, the scope of the study was not intended to be limited to special applications and the conclusions are believed to have general application to high-performance computers.

Introduction

One of the basic choices the designers of a digital computer must make is whether to represent numbers in decimal or binary form. Civilized man settled on ten as the preferred number base for his own arithmetic a long time ago.[1] The ten digits of the decimal system had their origin when man learned to count on his ten fingers. The word digit is derived from the Latin word for *finger* and remains to testify to the history of decimal numbers. Historically, several other number bases have been employed by various peoples at different times. The smaller number bases are clearly more awkward for human beings to use because more symbols of a smaller set are needed to express a given number. Nevertheless, there is evidence of the use of the base 2, presumably by men who observed that they had two ears, eyes, feet, or fists.

With the decimal symbolism in universal use, it is natural that the earliest automatic digital computers, like the preceding desk calculators and punched card equipment, were decimal. In 1946 John von Neumann and his colleagues at the Institute for Advanced Study, in their report describing the new concept of a stored program computer, proposed to depart from that practice [1]. They chose the base 2 for their system of arithmetic because of the greater economy, simplicity, and speed.

Many designers have followed this lead and built binary computers patterned after the machine then proposed. Others have disagreed and pointed out techniques for obtaining satisfactory speeds with decimal arithmetic without unduly increasing the overall cost of the computer. Since decimal numbers are easier to use, the conclusion has been drawn that decimal computers are easier to use. There have been two schools of thought ever since, each supported by the fact that decimal and binary computers have both been eminently successful.

As the Institute for Advanced Study report has long been out of print, it seems appropriate to quote at some

[1] Although in most languages numbers are expressed by decimal symbols, it is a curious fact that there has been so far no standardization on multiples of ten for all units of money, weight, and measure. We are still content to do much of our everyday arithmetic in what is really a mixed radix system which includes such number bases as 3, 4, 7, 12, 24, 32, 60, 144, 1760, etc.

length the reasons then given for choosing binary arithmetic:

"In spite of the longstanding tradition of building digital machines in the decimal system, we feel strongly in favor of the binary system for our device. Our fundamental unit of memory is naturally adapted to the binary system since we do not attempt to measure gradations of charge at a particular point in the Selectron [the memory device then proposed] but are content to distinguish two states. The flip-flop again is truly a binary device. On magnetic wires or tapes and in acoustic delay line memories one is also content to recognize the presence or absence of a pulse or (if a carrier frequency is used) of a pulse train, or of the sign of a pulse. (We will not discuss here the ternary possibilities of a positive-or-negative-or-no pulse system and their relationship to questions of reliability and checking, nor the very interesting possibilities of carrier frequency modulation.) Hence if one contemplates using a decimal system . . . one is forced into a binary coding of the decimal system—each decimal digit being represented by at least a tetrad of binary digits. Thus an accuracy of ten decimal digits requires at least 40 binary digits. In a true binary representation of numbers, however, about 33 digits suffice to achieve a precision of 10^{10}. The use of the binary system is therefore somewhat more economical of equipment than is the decimal.

The main virtue of the binary system as against the decimal is, however, the greater simplicity and speed with which the elementary operations can be performed. To illustrate, consider multiplication by repeated addition. In binary multiplication the product of a particular digit of the multiplier by the multiplicand is either the multiplicand or null according as the multiplier digit is 1 or 0. In the decimal system, however, this product has ten possible values between null and nine times the multiplicand, inclusive. Of course, a decimal number has only $\log_{10} 2 \approx .3$ times as many digits as a binary number of the same accuracy, but even so multiplication in the decimal system is considerably longer than in the binary system. One can accelerate decimal multiplication by complicating the circuits, but this fact is irrelevent to the point just made since binary multiplication can likewise be accelerated by adding to the equipment. Similar remarks may be made about the other operations.

An additional point that deserves emphasis is this: An important part of the machine is not arithmetical, but logical in nature. Now logics, being a yes-no system, is fundamentally binary. Therefore a binary arrangement of the arithmetical organs contributes very significantly towards producing a more homogenous machine, which can be better integrated and is more efficient.

The one disadvantage of the binary system from the human point of view is the conversion problem. Since, however, it is completely known how to convert numbers from one base to another and since this conversion can be effected solely by the use of the usual arithmetic processes there is no reason why the computer itself cannot carry out this conversion. It might be argued that this is a time consuming operation. This, however, is not the case . . . Indeed a general-purpose computer, used as a scientific research tool, is called upon to do a very great number of multiplications upon a relatively small amount of input data, and hence the time consumed in the decimal to binary conversion is only a trivial per cent of the total computing time. A similar remark is applicable to the output data."

The computer field and, along with it, the technical literature on computers have grown tremendously since this pioneering report appeared. There is considerable material explaining binary, decimal, and other number systems [2]. One looks in vain, however, for a discussion which would bring these early comments up-to-date in the light of experience. In fact, a fairly thorough search failed to reveal any critical comparison of number systems. The nontechnical literature [3] does contain some advice to business management to beware of binary computers.

The present paper not only attempts to fill a gap in the technical literature, it is also intended to widen the scope of the examination so as to reflect experience gained with large computers and their increasing areas of application. Mathematical computations are still important, but the processing of large files of business data has since become a major field. Computers are beginning to be applied to the control of planes in actual flight, to the collection and display of data on demand, and to language translation and systems simulation. Regardless of the application, a great deal of the time of any large computer is spent on preparing programs before they can be run on that computer. Much of this work is non-numeric data processing. The point of view has thus shifted considerably since the days of the von Neumann report, and a re-evaluation seems to be in order.

1. Information Content

Information theory [4, 5] allows us to measure the information content of a number in a specific sense. Assume a set of N possible numbers, each of which is equally likely to occur during a computing process. The information contained in the selection of a number is then

$$H = \log_2 N.$$

Suppose that a set of b binary digits (bits) represents a set of 2^b consecutive integers, extending from 0 to $2^b - 1$, each of these integers being equally probable. Then

$$\begin{aligned} H &= \log_2 2^b \\ &= b \text{ bits.} \end{aligned}$$

(Because in this example the amount of information is equal to the number of bits needed to represent the integer in binary form, the bit is often chosen as the unit of information. The two uses of the term "bit" should not be confused, however. Numbers are defined independently of their representation, and the information content of a number is measured in bits regardless of whether it is in binary, decimal, or any other form.)

Similarly, assume a set of 10^d consecutive integers from 0 to $10^d - 1$ expressed by d decimal digits. Here

$$\begin{aligned} H &= \log_2 10^d \\ &= (\log_2 10)d \\ &\approx 3.322d \text{ bits.} \end{aligned}$$

Thus a decimal digit is approximately equivalent in information content to $\log_2 10 \approx 3.322$ binary digits.

In the actual representation of a number N, both b and d must, of course, be integers. The ranges 10^d and 2^b cannot be compared exactly. For such pairs as $d = 3$ and

$b = 10$, the values $10^3 = 1000$ and $2^{10} = 1024$ come very close to being equal. Here $b/d = 10/3 \approx 3.333$, which agrees well with the above value 3.322. This shows, at least, that the measure of information is a plausible one.

Conversely, to express a decimal number requires approximately 3.322 times as many binary symbols (0 to 1) as decimal symbols (0 to 9). Few truly decimal switching and storage devices have found application in high-speed electronic computers; otherwise a decimal computer might be a great deal more compact than a corresponding binary computer. Generally, only binary (or *on-off*) devices are used, and hence decimal digits must be encoded in binary form even in decimal computers. Since bits cannot be split to make up the 3.322 bits theoretically required, at least 4 bits are needed to represent a decimal digit. Therefore, instead of being more compact, decimal computers require at least $4/3.322 = 1.204$ times as many storage and switching elements in a large portion of their systems. The reciprocal ratio, 3.322/4 or 83 %, might be considered to be the theoretical storage efficiency of a decimal computer. Codes with more than 4 bits for each decimal digit are often used to take advantage of certain self-checking and other properties. The efficiency is then correspondingly lower.

The 83 % efficiency is only a theoretical value even if a 4-bit code is used. A basic assumption was that all of the N possible numbers in the expression $\log_2 N$ are equally likely to occur. Nonuniform distributions are quite frequent, however. A common situation is that a set of b bits (in the binary case) is chosen to represent N integers from 0 to $N-1$ $(N < 2^b)$ and the integers N to $2^b - 1$ are never encountered. The information content, $\log_2 N$, may then be considerably less than b bits. Both binary and decimal computers suffer a loss of efficiency when the number range N is not a power of the number base.

For example, assume $N = 150$, that is, the numbers range from 0 to 149. Then

$$H = \log_2 150 = 7.23 \text{ bits.}$$

Since 8 is the next largest integer, a binary computer requires at least 8 bits to represent these numbers, giving an efficiency of 7.23/8 or 90 %. A decimal computer requires at least 3 decimal digits or 12 bits, with an efficiency of 7.23/12 or 60 %. Relative to the binary number base, the efficiency of the decimal representation is only .60/.90 or 67 %.

The loss in efficiency is greatest for the smaller integers. With binary integers the lowest efficiency of 78 % occurs for $N = 5$. Decimal representation has its lowest efficiency of 25 % at $N = 2$. Decimal representation is never more efficient than binary representation, and only for $N = 9$ and $N = 10$ are they equally efficient.

Figures 1a and 1b show the storage efficiency curves for binary and decimal systems, as well as the efficiency of the decimal representation relative to the binary system.

For the above analysis it was assumed that the least possible number of bits or decimal digits are assigned to represent N. A great many computers are designed around a fixed word length, and even more space will then be wasted unless time is taken to program closer packing of data. It was also assumed that the N integers considered are distributed uniformly throughout the interval; a nonuniform distribution with numbers missing throughout the interval results in a further lowering of storage efficiency, which affects binary and decimal computers alike.

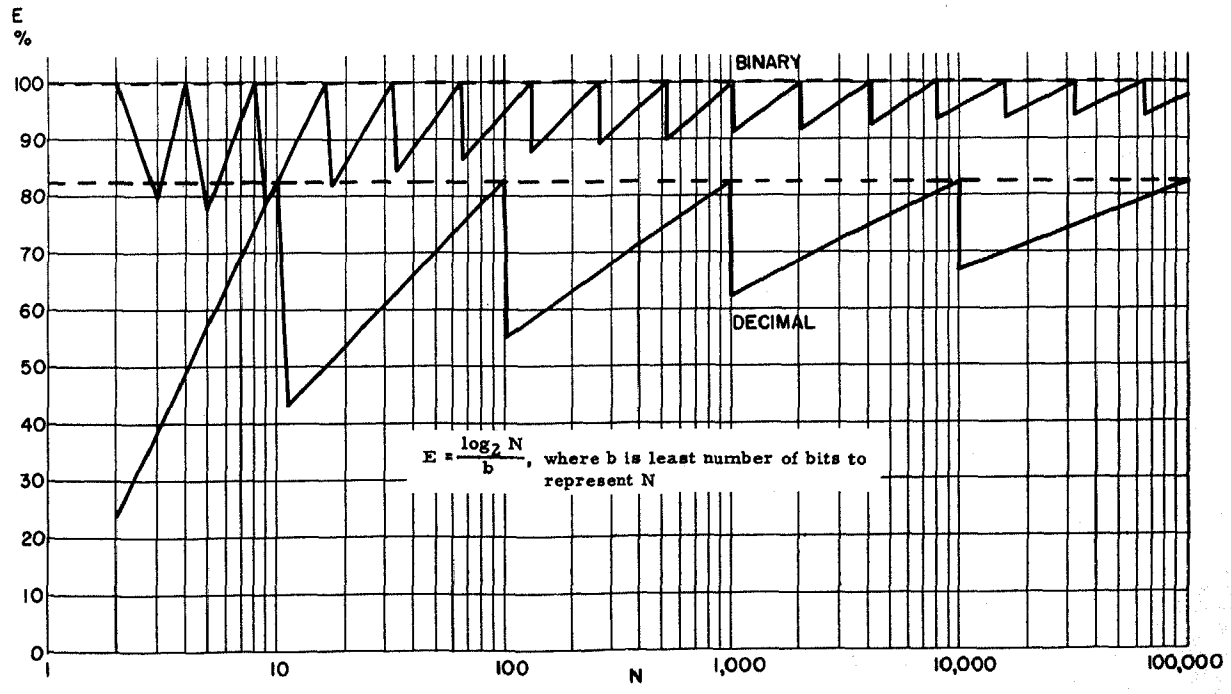

FIG. 1a. Absolute efficiency of decimal and binary number systems

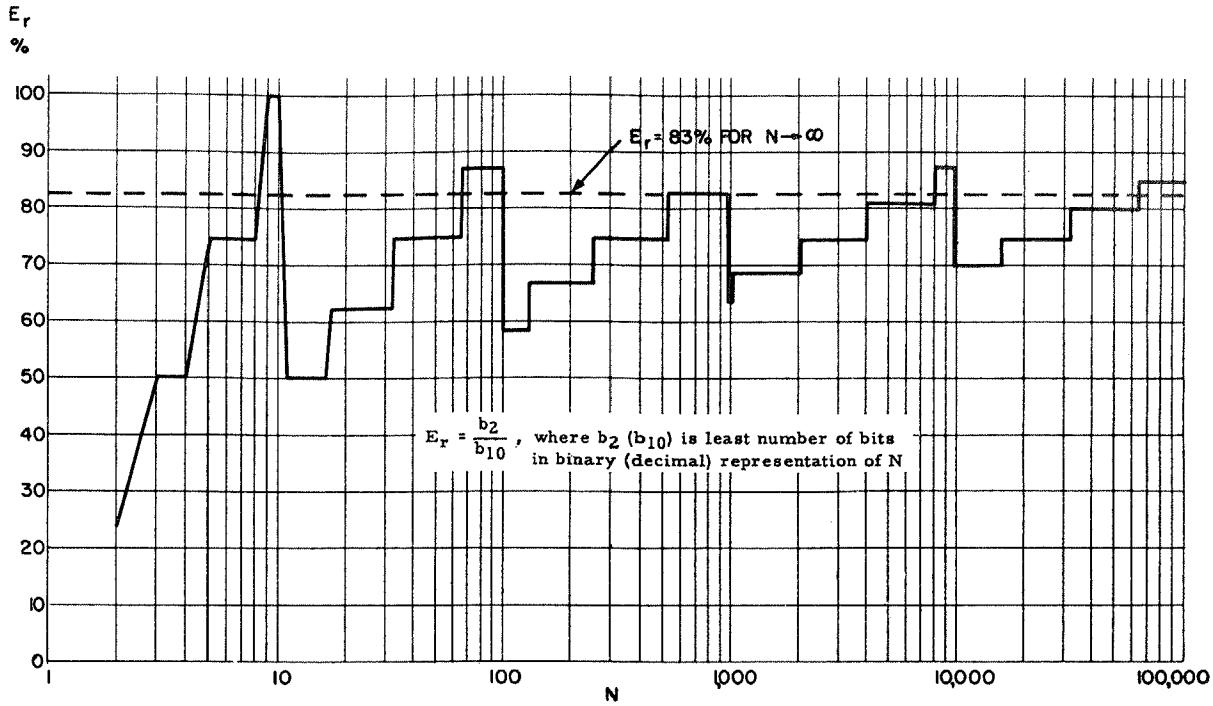

FIG. 1b. Relative efficiency of decimal and binary number systems

While only integers have been considered so far, the same reasoning obviously applies to fractions truncated to a given precision, since they are treated in storage in the same manner as integers. Similarly, the sign of a number may be considered as an integer with $N = 2$. Instructions are always made up of a number of short, independent pieces. For example, an operation code for 45 different operations may be encoded as a set of integers with $N = 45$, for which the binary efficiency is 92% and the decimal efficiency is only 69%.

The lower information handling efficiency of the decimal representation may reflect itself in higher cost, in lower performance, or both. If performance is to be maintained, the cost will go up, but it would be wrong to assume that the extra bits required for decimal representation mean a proportional increase in cost. The ratio of the cost of storage, registers, and associated switching circuits to the total cost of a computer depends greatly on the design. Factors other than hardware cost need to be considered in the overall cost of using a computer on a given job.

When the cost is to be the same, a lower storage efficiency may result in lower performance. For a computer designed to achieve the highest practicable performance with given types of components, the rate of information flow into and out of memory may be a major limiting factor. While it can be very misleading to compare two dissimilar computers on the basis of memory speed only, the comparison is appropriate for two computers using similar components and organization but differing mainly in their number representation.

The memory of a computer may be looked on as an information channel with a capacity of

$$C = nw \text{ bits per second,}$$

where n is the number of bits in the memory word and w is the maximum number of words per second which the memory can handle.

This channel capacity is fully utilized only if the words represent numbers from 0 to $2^n - 1$, each of which is equally probable. If the information content is less than that, the actual performance is limited to Hw, where H is defined as before. More specifically, if a memory word is divided into k fields, of range N_1, N_2, N_3, $\cdots$, N_k, then

$$H = \sum_{i=1}^{k} \log_2 N_i .$$

The maximum performance is lowered by the factor

$$\frac{H}{C} = \frac{\sum \log_2 N_i}{n} .$$

For $k = 1$ this is the same factor as the storage efficiency described above.

Other organizational factors may reduce the performance further, and memory multiplexing can be used to increase overall performance. These matters are independent of the number representation. The fact remains that a decimal organization implies a decided lowering of the maximum performance available. This loss can be overcome only in part by increasing the number of components because of physical and cost limitations.

In summary, to approach the highest theoretical performance inherent in a given complement of components of a given type, it is necessary to make each bit do one bit's worth of work.

2. Arithmetic Speed

A binary arithmetic unit is inherently faster than a decimal unit of similar hardware complexity operating on numbers of equivalent length. While the gain in speed of binary over decimal arithmetic may not be significant in relatively simple computers, it is substantial when the design is aimed at maximum speed with a given set of components. There are several reasons.

A decimal adder position requires more successive levels of switching to produce a correct sum digit than a binary adder position. The delay in successive switching stages places a limit on the attainable speed. Moreover, since more bits are needed for decimal numbers with a given precision, more time is needed to propagate carries in a parallel adder or to process successive digits in a serial adder.

With a base of two, certain measures can be taken to speed up multiplication and division. An example is the skipping of successive zero bits or one bits in the multiplier. When corresponding measures are taken with base ten arithmetic, they are found to give a smaller ratio of improvement. Thus the average number of additions or subtractions needed during multiplication or division is greater, and this is compounded by the extra time needed for each addition or subtraction.

In storage devices limited by their bit transmission rate, such as ordinary magnetic tape, the greater storage space occupied by decimal numbers is directly reflected as a loss in speed. Whenever the transmission rate of such devices becomes the limiting factor in solving large problems, a binary computer is clearly at least 20.4% faster than a corresponding decimal computer.

Scaling of numbers, which is required to keep numbers within the bounds of the registers during computation, results in a greater round-off error when the base is ten. The finest step of adjustment is 3.3 times as coarse when shifting by powers of ten as it is with powers of two. The greater error will in large problems require more frequent use of multiple-precision arithmetic at a substantial loss of speed. The effect is partly offset by the fact that scaling will occur more often in binary arithmetic, and the extra shifting takes more time.

Multiplying or dividing by powers of the number base is accomplished by the fast process of shifting. The coefficients 2 and $\frac{1}{2}$ are found much more frequently in mathematical formulas than other coefficients, including 10 and $\frac{1}{10}$, and a binary computer has the advantage here.

To overcome the lower speed inherent in decimal arithmetic, it is of course possible to construct a more complex arithmetic unit at a greater cost in components. If top speed is desired, however, the designer of a binary arithmetic unit will have taken similar steps. There is a decided limit on this acceleration process. Not only does the bag of tricks run low after a while, but complexity eventually becomes self-defeating. Greater complexity means greater bulk, longer wires to connect the components, and more components to drive the longer wires. The longer wires and additional drivers both mean more delays in transmitting signals which cannot be overcome by adding even more components. When the limit is reached there remains the substantial speed differential between binary and decimal arithmetic, as predicted by theoretical considerations in section 1.

3. Numeric Data

Numeric data entering or leaving a computer system are of two kinds: those which must be interpreted by humans and those which come from or actuate other machines. The first are naturally in decimal form. The second class, which occurs when a computer is part of an automatic control system, could also be decimal since machines, unlike human beings, can readily be designed either way; but binary coding is generally simpler and more efficient.

The previously cited von Neumann report considered only the important applications where the volume of incoming and outgoing data is small compared to the volume of intermediate results produced during a computation. In a fast computer any conversion of input and output data may take a negligible time, whereas the format of intermediate results has a major effect on the overall speed. The von Neumann report did not consider the equally important data processing applications in which but few arithmetic steps are taken on large volumes of input-output data. If these data are expressed in a form different from that used in the arithmetic unit, the conversion time can be a major burden. Any conversion time must be taken into account as reducing the effective speed of the arithmetic unit.

The choice would appear simple if different computers could be applied to different jobs, using decimal arithmetic when the data are predominantly decimal and binary arithmetic elsewhere. Experience has shown, however, that a single large computer is often used on a great variety of jobs which cannot be classified all one way or the other. Moreover, as will be shown in subsequent sections, there are strong reasons for choosing a binary addressing system even where the applications indicate the use of decimal data arithmetic. Some kind of binary arithmetic unit must then be provided anyway, if only to manipulate addresses.

A high-speed binary arithmetic unit is thus clearly desirable for all applications. To handle decimal data, the designer may choose to provide a separate decimal arithmetic unit in the same computer, or he may prefer to take advantage of the speed of his binary arithmetic unit by adding special instructions to facilitate binary-decimal conversion.

The decimal arithmetic and conversion facilities must take into account not only the different number base of

decimal data but also the different format. Binary numbers usually consist of a simple string of numeric bits and a sign bit. Decimal numbers are frequently interspersed with alphabetical data, and extra "zone" bits (sometimes a separate digit) are then provided to distinguish decimal digit codes from the codes for alphabetic and other characters. Distinguishing the numeric and zone portions of coded digits greatly adds to the difficulty of doing conversion by ordinary arithmetic instructions. Hence the decimal arithmetic and conversion instructions should be designed to process decimal data directly in a suitable alphanumeric code.

4. Non-numeric Data

A computer may have to process a large variety of non-numeric information:

Character codes representing alphabetic, numeric, or other symbols for recording data in human-readable form.
Codes used to perform specified functions, such as terminating data transmission.
Yes-no data ("married", "out of stock", etc.).
Data for logical and decision operations.
Instructions (other than numeric addresses).
Machine status information, such as error indications.
Status of switches and lights.

Because the storage and switching elements normally used in computers are binary in nature, all information, numeric or non-numeric, is encoded in a binary form. This binary coding has no direct relation to the number base being used for arithmetic. The number base determines the rules of arithmetic, such as how carries are propagated in addition, but it has no meaning when dealing with non-numeric information. Thus the binary-decimal distinction does not apply directly to the nonarithmetic parts of a computer, such as the input-output system.

A computer may have to process a great variety of information. Even where mathematical computation is the major job, a great deal of computer time is usually spent in preparing programs and reports. It is important, therefore, that the designer avoid constraints on the coding of input and output data, such as are found in many existing decimal computers. Many of these constraints are unnecessary and they place extra burdens of data conversion and editing at greater cost on peripheral equipment.

5. Addresses

Memory addresses are subject to counting and adding and are thus proper numbers which can be expressed with any number base. Base 10 has the same advantage for addresses as for data. Conversion is not required and actual addresses can be continuously displayed on a console in easily readable form.

The compactness of binary numbers is found particularly advantageous in fitting addresses into the usually cramped instruction formats. Tight instruction formats contribute to performance by reducing the number of accesses to memory during the execution of a program, as well as by making more memory space available for data. The low efficiency of decimal coding for addresses has already led designers of nominally decimal computers to introduce a certain amount of binary coding into their instruction formats. Such a compromise leads to programming complications which can be avoided when the coding is purely binary.

Although the compactness of the binary notation is important, the most significant advantage of binary addressing is probably the ease of performing data transformation by address selection (table look-up). This is discussed in the next section.

6. Transformation

A single data processing operation may be considered as transforming one or more pieces of data into a result according to certain rules. The most general way of specifying the rules of transformation is to use a set of tables. The common transformations, such as addition, multiplication, and comparison, are mechanized inside the computer, and some others, such as code conversion, are often built into peripheral equipment; tables (often called matrices) may or may not be employed in the mechanization. All transformations not built into the computer must be programmed. In a computer with a large rapid-access internal memory, the best transformation procedure, and often the only practical one, is table look-up. Each piece of data to be transformed is converted to an address which is used to select an entry in a table stored in memory. (This method of table look-up is to be distinguished from table searching where all entries are scanned sequentially until a matching entry is found.) Figure 2 serves to illustrate the process by a code translation example.

Two methods of encoding the digits 0 to 9, both in current use, are shown in figure 2. One is a 2-out-of-5 code which requires 5 bits for every digit. Two, and only two, one-bits are contained in each digit code with all other 5-bit combinations declared invalid. This property permits checking for single and common multiple errors. The second code is a 4-bit representation using codes 0001 to 1001 for the digits 1 to 9 and 1010 for the digit 0. Codes 0000 and 1011 to 1111 are not used.

To translate from the 5-bit code to the 4-bit code, a table of 32 (2^5) entries is stored in successive memory locations. Each entry contains a 4-bit code. Where the 5-bit combination is a valid code, the corresponding 4-bit code is shown. All invalid 5-bit combinations are indicated in the example by an entry of 1111, which is not a valid 4-bit code.

To look up a given 5-bit code, such as 10001, this code is added to the address of the first entry, the table base address:

	100000	Table base address
	+ *10001*	Incoming 5-bit code
Sum	110001	Address of table entry

The corresponding table entry is seen in figure 2 to be 0111. If the entry had been 1111 the incoming code would be known to contain an error.

The key to this transformation process is the conversion of data to addresses. A system capable of accepting any bit pattern for transformation can adapt itself readily to any external coding, including codes in equipment over which the designer has no control. The desire to accept any bit pattern as an address almost dictates binary addressing. It is true that decimal addressing does not entirely preclude transformation of arbitrary data by indirect methods, but such methods are very wasteful of time or memory space.

Symbol	Code A (5 bits)	Code B (4 bits)
1	00011	0001
2	00101	0010
3	00110	0011
4	01001	0100
5	01010	0101
6	01100	0110
7	10001	0111
8	10010	1000
9	10100	1001
0	11000	1010

(a) Two Codes for Decimal Digits

Address	Entry
...100000	1111
100001	1111
100010	1111
100011	0001
100100	1111
100101	0010
...	...
101110	1111
101111	1111
110000	1111
110001	0111
110010	1000
110011	1111
...	...
...111111	1111

(b) Translation Table, Code A to Code B

	...100000	Table Base Address
+	*10001*	Incoming 5-bit Code
(Sum)	...110001	Address of Table Entry

(c) Example: Translation of Symbol "7"

FIG. 2. Example of code translation by transformation

7. Partitioning of Memory

It has already been mentioned that binary numbers permit scaling in smaller steps which reduces the loss of significance during computation. Binary addresses also have this advantage of greater resolution. Shifting binary addresses to the left or right makes it easy to divide memory into different areas or cells whose size is adjustable by powers of two. With decimal addressing, such partitioning is easily obtained only by powers of ten.

In a core memory, for example, each address refers to a memory word consisting of the number of parallel bits which are accessible in a single memory cycle. Suppose that the number of bits in a memory word is chosen to be a power of 2, such as $2^6 = 64$ bits. Then by extending the word address and inserting 6 bits to the right, and by providing another selection mechanism, it becomes possible to address individual bits in a memory word. When increments are added to these addresses in binary form, carries from the sixth to the seventh bit automatically advance the word address [6].

The flexibility of bit addressing may be illustrated by extending the example of figure 2. Instead of using an entire memory word to hold one 4-bit table entry, the same entries may be stored in a cell only 4 bits long, with 16 cells to each memory word of 64 bits. With respect to the bit address, the incoming code is shifted 2 bits to the left to obtain increments of 4 bits of storage in memory:

	...10000000	Table base address
+	*1000*100	Incoming 5-bit code with two zeros added
Sum	...11000100	Address of table entry

Address of word — Address of bit in word

The example can be readily changed to translate from a 5-bit code to a 12-bit code, such as is used on punched cards. Without showing an actual table, it is evident that the 12-bit code can be conveniently stored in successive 16-bit cells. The proper addresses are then obtained by inserting four zero-bits at the right, instead of two as before:

	...1000000000	Table base address
+	*10001*0000	Incoming 5-bit code with four zeros added
Sum	...1100010000	Address of table entry

Address of word — Address of bit in word

Similarly, the process can be extended to finer divisions. By using the incoming code as the address of a single bit, it is possible to look up a compact table of yes-no bits in memory to indicate, for example, the single fact of whether the code is valid or not.

Now consider these examples in terms of decimal addressing. If single bits were to be addressed, the next higher address digits would address every tenth bit. This is too large a cell size to permit addressing every decimal digit in a data field. To be practical in large-scale numeric computation, the code for a decimal digit cannot occupy a cell of more than 4, 5, or at most 6 bits. When the addressing is chosen to operate on cells of this size, direct addressing of single bits is ruled out. Table entries requiring more than one cell cannot occupy less than ten cells.

The designer of a binary computer may or may not choose to endow it with the powerful facility of addressing single bits. The point remains that flexible partitioning of memory is not available with decimal addressing.

8. Program Interpretation

A major task in any computer installation is the preparation and check-out of programs. Printing portions of a program at the point where an error has been found is a common check-out tool for the programmer. Interpreting such a print-out is greatly simplified if the instructions are printed in the language which the programmer used.

At first glance this seems to be a convincing argument for decimal computers. On closer examination it becomes evident that neither a binary nor a decimal machine is very usable without adequate service programs. When good service programs are available it is hard to see how the number base in the arithmetic unit makes much difference during program check-out.

One reason for service programs is that in practice much programming is done in a symbolic notation regardless of the number base used internally. The programmer's language is then neither binary nor decimal but a set of alphanumeric mnemonic symbols. Conversion to or from the symbolic notation by means of a service program is desirable for the user of either kind of machine, with the possible exception of the programming specialist who writes programs in machine language either by choice or to develop new service programs.

Another and more basic reason is that most computers have more than one format for data and instructions, and a service program is needed to help interpret these formats. In binary computers it is desirable to know whether a data field is an integer or a floating point number with its separate exponent (integer) and mantissa (fraction). The instructions are normally divided in a different manner than either kind of data field. A knowledge of the divisions of each format is required in converting from binary to decimal form.

Many decimal computers do not use purely decimal coding for the instructions, particularly those aimed at efficiently processing large amounts of non-numeric business data. Moreover, alphanumeric character coding usually employs a convention which is different from the coding of instructions. Again a service program is needed to interpret the different data and instruction languages.

Figure 3 illustrates this point with print-outs of actual computer programs. The first example is for an IBM 704, which uses binary arithmetic. The service program lists memory locations and instructions in octal form with the appropriate instruction bits also interpreted as alphabetic operation codes. The service program distinguishes floating-point numbers which are listed in a decimal format with separate exponent, mantissa, and signs.

The second illustration shows a print-out from the IBM 705, a computer with decimal arithmetic and alphanumeric coding for data. Each alphanumeric character has a unique 6-bit code. For reasons of storage efficiency, instructions in the 705 use a different code where some of the bits in a 6-bit character have independent meanings. In the example shown in figure 3b this dual representation

Location	Instruction or Data
00622	FSB 0 30200 0 00637
00623	TZE 0 10000 0 00626
00624	TPL 0 12000 0 00607
00625	STO 0 60100 0 00634
00626	HTR 0 00000 0 00561
00627	−01 + 9.945 2245
00630	+03 + 4.130 0000
00631	−01 + 7.330 4100
00632	+05 + 5.301 7842

(a) Print-Out from IBM 704

Location	A. Straight Print-Out	B. Print-Out Modified for Instructions
01204	8TL-1	8 13301 10
01209	4/QR1	4 11891 10
01214	L1094	L 1094
01219	HW5R4	H 16594 02
01224	7W6N5	7 16655 02
01229	12449	1 2449
. . .	. . .	. . .
11304	IS PA	I 12 7A 14
11309	GE WA	G 35 6A 13
11314	S PRO	S 3 790 10
11319	CESSE	C 35222E 05
11324	D THR	D 3 38R 07
11329	OUGH	O 1478 15

(b) Print-Out from IBM 705

FIG. 3. Examples of program print-outs

is overcome by printing the program and data twice, once for ease of reading data and once for ease of interpreting instructions. A service program was needed to accomplish this.

An objection might be raised that the examples show up problems in existing machine organizations rather than a need for service programs. It is actually possible for "numeric engines" aimed at processing only numeric data to escape the problem of dual representation for instructions and data. When alphanumeric data must also be processed in a reasonable efficient manner, the problem of dual representation must be faced, however.

9. Other Number Bases

Only binary and decimal computers have been considered so far. While other number bases could clearly be selected, they would all require translation to and from decimal formats and they would be less efficient than base two.

For instance, base 100 and base 1000 appear to have an efficiency close to base 2 and a strong relationship to base 10. Actually, the high efficiency holds only for numbers whose upper range is a power of the base. For such large number bases the *average* efficiency is rather low, considering numbers of various ranges. Moreover, the larger the number base the lower is the resolution and flexibility of format design. The fact that the base is a power of ten does not make it decimal. The translation is not much simpler than with base 2, and without conversion the numbers are no easier to interpret.

10. Conclusion

The binary number base has substantial advantages in performance and versatility for addresses, for control data which are naturally in binary form, and for numeric data which are subjected to a great deal of arithmetic processing. Figures of merit are difficult to assign because the performance and cost of a given computer design depends on a great many factors other than the number base. It is clear, however, that the decimal representation has an inherent loss in performance of at least 20 to 40% as compared to the binary number system, and refined design can overcome this loss only partly by increasing the cost. The lower efficiency makes itself felt in a number of ways, so that the combined effect on overall performance may be even greater.

It is equally clear, however, that a computer which is to find application in the processing of large files of information and in extensive man-machine communication, must be adept at handling data in human-readable form. This includes decimal numbers, alphabetic descriptions, and punctuation marks. Since the volume of data may be great, it is important that binary-decimal and other conversions not become a burden which greatly reduces the effective speed of the computer.

Hence it was concluded, in the design of the IBM STRETCH computer, to combine the advantages of binary and decimal number systems. Binary addressing has been adopted for its greater flexibility; each bit in memory has a separate address and the length of a word in memory is a power of two (64 bits). A binary arithmetic unit is provided for manipulating these addresses and for performing floating-point arithmetic at extremely high speed. Efficient binary-decimal conversion instructions minimize the conversion time for input and output data intended for use in extensive mathematical computation. To permit simple arithmetic operations directly on data in a binary-coded decimal form or in an alphanumeric code, a decimal arithmetic unit is also made available.

Such a combination of binary and decimal arithmetic in a single computer provides a high-performance tool for many diverse applications. It may be noted that the conclusion might not be the same for computers with a restricted range of functions or with performance goals limited in the interest of economy; the difference between binary and decimal operation might well be considered too small to justify incorporating both. The conclusion does appear valid for high-performance computers regardless of whether they are aimed primarily at scientific computing, business data processing, or real-time control. To recommend binary addressing for a computer intended for business data processing is admittedly a departure from present practice, but the need for handling and storing large quantities of non-numeric data makes the features of binary addressing particularly attractive. In the past, the real obstacle to binary computers in business applications has been the difficulty of handling inherently decimal data. Binary addressing and decimal data arithmetic, therefore, make a powerful combination.

Acknowledgements

This paper merely attempts to organize and present the thoughts and conclusions of many participants in the planning of the IBM STRETCH computer. The author is particularly indebted to G. A. Blaauw, F. P. Brooks, Jr., and D. W. Sweeney for their constructive criticism of the presentation.

REFERENCES

1. A. H. BURKS, H. H. GOLDSTINE, AND J. VON NEUMANN, Preliminary discussion of the logical design of an electronic computing instrument, Institute for Advanced Study, Princeton, N. J., first edition June 1946, second edition 1947; Section 5.2. Also subsequent reports by H. H. Goldstine and J. von Neumann.
2. See, for example, R. K. RICHARDS, *Arithmetic Operations in Digital Computers*, D. Van Nostrand, 1955; Chapter 1.
3. WM. D. BELL, *A Management Guide to Electronic Computers*, McGraw Hill, 1957; pp. 92–97.
4. C. E. SHANNON AND W. WEAVER, *The Mathematical Theory of Communication*, The University of Illinois Press, 1949.
5. L. BRILLOUIN, *Science and Information Theory*, Academic Press, 1956; pp. 3–4.
6. F. P. BROOKS, JR., G. A. BLAAUW, AND W. BUCHHOLZ, Processing data in bits and pieces, *IRE Transactions on Electronic Computers*, EC-8, No. 2; June 1959.