CONSIDERATIONS IN COMPUTER DESIGN — LEADING UP TO THE CONTROL DATA 6600

by JAMES E. THORNTON
CONTROL DATA CHIPPEWA LABORATORY

CONTROL DATA CORPORATION
8100 34TH AVENUE SOUTH, MINNEAPOLIS 20, MINNESOTA

Pub. No. BR21 (8-63)
Litho U.S.A.

NO APOLOGY INTENDED

Someone has said that the elephant can grow no larger because of the ratio of its volume to the surface area of its digestive system. On the theory that simplest reasons are best, this certainly ranks high on the list. I don't suppose that this explanation of an elephant's size is entirely accurate. However, it illustrates the idea of an ultimate limit.

Perhaps the steps can be traced in the evolution of the elephant which most affected its final limitation. I can imagine some steps aiding and some reducing the eventual size. Since the evolutionary model states that natural selection controls each step, the short-term corrections predominate. I have no idea what caused the elephant's tusks, for example. The first rudimentary tusks must have satisfied some early need. They evidently helped and were useful; therefore, they were selected. The tusks have no obvious connection with the elephant's maximum size, at least by the above theory. However, they may have been evolved in favor of another set of molars which could improve digestion. Or perhaps the roots of the tusks further limit the intake of food and internally displace the digestive tract, with a net reduction of the eventual size. This little fantasy follows the lines of the natural selection model, by which we attempt to explain what is going on. It may be that this model, invented by man, is most accurate when applied to man's machines, in which a similar situation is developing. I'm not really interested in larger elephants, but rather in faster computers.

The quick fix

The factors influencing the evolution of the computer are economic (what isn't), logistic, comfort, convenience, and any number of other conflicting preferences. Designers have moved through a series of "safe" improvements without seriously tampering with the original idea. The significance of each innovation is largely masked by the mystery and confusion surrounding complex machinery. Actually, a great deal can be accomplished by taking each obstacle and applying a short term correction to circumvent it. Really startling improvements in speed have come from the most innocent and deceptively simple corrections. Ingenuity of computer designers has made the "quick fix" the rule of the industry. The ability to do so much with the simple computer circuits leaves little excuse for attempting almost any new combination. The net effect is computers with superficially simi-

lar outward appearance (speed specification, special features, standard features) but fundamentally different internal methods.

There can be no argument with the desire for faster operation or more effective operation. Our principle of natural selection serves to weed out the weak ideas. A strong feature is easily accepted, copied, and re-copied without much change. Probably a good rule of thumb for measuring success is the number of suggested changes – the fewer the better. On the other hand, following this rule obviously leads to including *every* desirable feature ever mentioned. Lacking economic or electronic reasons for rejecting a new addition, there may be another kind of reason. It has to do with the ultimate limits (something like the elephant) and leads to a wholly different approach to computer design.

From the beginning, there was something clean and straightforward about the digital computer idea. One could visualize enormous potentialities of such machines. The extension of our brainpower was a clear possibility; indeed, very shortly a clear reality. It was easy to think of machines doing every routine computational job, large or small. The very principle of using numbers, with their almost unlimited resources, as the fundamental internal controls further opened the possibilities. However, it was psychologically a little too much to take at once. The temptation to plunge off without careful deliberation was countered with the reaction to do nothing really different. It was a choice of being a fool or a coward. Under the circumstances, the "safe" improvements looked rather good. We were encouraged to go on.

Elementary, my dear

The numerical instructions – the programs – were originally intended to provide desirable deviations in the computation quickly and easily. The computer designer could then concentrate on making the most of the fundamental computer parts without fear of these deviations. The original thought was to . . . "make the machine elementary. The program will provide for the complex needs." The history of the computing machine has recorded some failure in this direction. The first special relief granted to a group who suffered from this elementary phase began a series of evolutionary phases.

It is basic to the computer idea that a problem planner conceive the solution in terms of a sequence of elementary operations. He orders these elementary operations in the amount, sequence, and combination necessary to the solution. A major portion of the utility of the computer is the use of repeatable sets of these instructions, the repetitions or iterations of these sets themselves computable. The successful programmer seeks these iterative loops for a maximum amount of the solution, knowing this to be faster, less wasteful, or otherwise more effective. Now, it is precisely in this area of maximum utility of the computer that the superficial likenesses between machines belie the internal differences to the detriment of the results. A particular repetition on one machine fits its internal methods more exactly (and therefore more effectively) than the same repetition on another machine. The second would prefer a variation in the sequence or combination of the elementary operations making up the iteration. Plainly, the program should correct for this problem, leaving the designer freedom to devise the most effective machine possible. Again, the industry history has recorded some failure in this approach.

The most important area of failure in choosing between what is to be elementary (or wired) and what is to be programmed lies in machine compatibility. If there were only one machine and one set of elementary operations, and if the designer merely reproduced the machine, making it faster each time, there would be very little compatibility problem. The original program deviations and the original optimum iterations would hold. This overlooks, of course, the whole dimension of improvement available at the elementary level. Even if one wanted this simplicity, the psychological atmosphere is against it. Programs must be made compatible by whatever means available. Here the idea of developing common language was most successful. The program was split into two levels, the original with sequences and combinations, and a second one with common language compatibility. Methods devised to transpose between the two levels could be adjusted to optimize for the specific internal needs of each machine. This was the original idea in an evolved form. This should have released the elementary operations for a more effective result. However, a movement to upgrade the machine to fit the new languages bids fair to neutralize this method.

This sort of discussion of elementary versus "special effect" operations runs the risk of being lost in

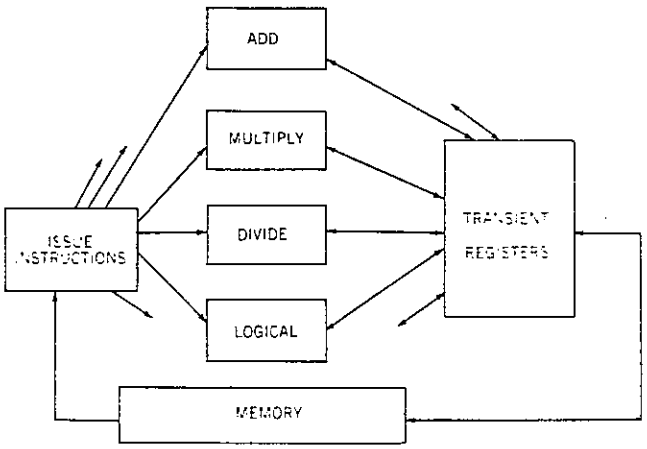
specific argument. Almost any kind of operation can attract support for a time. Only history can select the good from the bad in the absence of specific economic, logistic, or other argument for or against. Therefore, it is not with any specific operation that argument can be successful. What remains to be done now is to clear away *some* of the growth and debris, leaving only those operations which are truly fundamental or for which considerable potential can be demonstrated. The dependence on several levels of language should aid in this effort rather than trigger a series of new corrections to fit. In short, the original idea was so good and so simple that we ought to start again with our experience as a guide. The time is fast approaching when a really serious upper limit will be reached, a direct result of the speed of light limit of electrical signals on wires.

What can be gained by simplicity? I know, of course, there remains the lingering doubt that simplicity is the answer. The computer must be effective, not merely fast. Without attempting to remove that doubt, let me discuss some effects of simplicity. The elementary level (wired-in operation) that I will consider fundamental includes floating point arithmetic as well as the logical and fixed point manipulative operations. Something over one hundred distinct operations can be edited down to about half that number for the machine's elementary set of operations. From this set, it must be possible to construct the most complicated operation. It is obvious that some, if not the majority, of such complicated operations can be made faster by wiring them in. I propose to show how they cause other delays which may result in a net loss.

A significant portion of the time of most elementary operations is absorbed in obtaining and identifying what to do. For two reasons, the simple instruction set is desirable. The fewer instructions require a smaller instruction word, allowing more to be obtained at once from memory (a normally slow operation). The simplicity of *all* instructions allows quick and simple evaluation of status to begin execution. Both reasons add up to faster instruction acquisition. Notice that this applies to *all* instructions. Adding complication to a special operation, therefore, degrades all the others. Particularly in the newest computers with a high degree of parallel operation, this instruction fetch and interpret time becomes a very significant percentage.

Concurrently sequential

Looking further into parallel operation, it is reasonable that more and more of the sequential operations will give way to parallel. As a consequence, more and more circuits are included in the computer. Full utilization of the extra hardware demands that instructions be *issued* quickly and efficiently to the free areas for execution. In order to get several areas in operation concurrently, the time for issuing must be substantially faster than the time for executing. This is precisely the area of fetch and interpret time.



For the repetitive iterations, mentioned earlier, which make up a large part of the computer's utility, this high speed *issuing* of instructions can be augmented by a high speed supply of instructions. Since the very simple instructions can also be made efficient of instruction bits, more can be held at once. Holding complete iterations without need to reference memory offers a significant speed advantage, distinctly improved by simplicity in the instruction set.

It is part of the theory of use of special wired instructions that many normally sequential operations need not operate in sequence. The special instructions remove all but the essential sequential operations from the time sequence. The extras are performed in separate hardware not influencing the total time. This valuable technique can be applied to whole sets of instructions if separate arithmetic and functional units are included in the computer. Assume about ten functional units, such as those in the central processor of the Control Data® 6600. Next, assume that these units contain completely independent controls. Further, assume an over-all control system which can issue instructions to these units, maintaining the necessary

sequence but allowing the "extra" operations to go forward without influencing the total time. It is feasible with such a system to construct a more complex special instruction by programming its parts without sacrificing the special ability of the wired-in special instructions. To the degree that this technique is available in *all* program sequences, not just the special combinations, the entire program is speeded up. Let me continue to point out that the special instructions are first given up in order to obtain this very desirable effect. The extra hardware is merely distributed in a more general way.

I mentioned the difference in internal methods from computer to computer. These methods, of course, are of little interest to the user except as they influence the final effectiveness of each program. In view of impending limits to speed, it may be fruitful to discuss some of the detailed methods. First, a look at the speed of light limit is in order. Among the several ways to interconnect the computer logic circuits, none exceeds about three-quarters of the speed of light. This translates to about 9 inches per nanosecond. A typical present-day computer wastes over ten per cent of its time traversing these interconnecting wires. Assuming factors of circuit improvement, in the future, of two to four times the present rates, one can see the dominating influence of these wires. This is a kind of reverse situation from our friend elephant. In order to reduce the wire length, the total volume must come down at a much higher rate. Some reduction is possible, but the volume-to-area-to-linear dimensions are almost self-defeating. It should be very clear that no really startling speed improvements can be made on these wire transmissions. Furthermore, each such improvement shortens the time when wire speed is a really difficult limit.

Psychological barriers

Computer circuits employ an intricate variety of methods. Such mechanisms as synchronism, sequences of steps, static combinations, storage, etc., depend, at least partly, on the accuracy of the clock. To the waste of time on wires mentioned above can be added the tolerance of the clock, the ratio of longest untimed paths to the shortest, and a host of unnecessary periods of circuits waiting for completion in other circuits. Ingenuity and brute force can occasionally improve on these wastes. Separating these into circuit wastes and logic organization waste, some guides can be

drawn. Circuit waste can be classified in electrical terms and, in turn, in terms of available components and techniques. Circuit waste can be minimized by careful test and good design judgment. However, logic and organization waste is a somewhat different thing. The designer crosses a kind of psychological barrier between the circuits and their logic. The logic carries with it no intrinsic waste. The questions of design begin with economics and markets; they end with the engineer's ingenuity. The pressure to reduce wastes due to the logic is compromised by the availability of outstanding circuit performance. New computers have been begun almost exclusively on the prospect of circuits of greater performance. As a result, the waste due to logical organization has not received equal attention. Consider what is in prospect when the circuit performance well runs dry and the kind of relativistic friction of the wiring can no longer be ignored. What is needed is a plan for removing the logical waste.

Remember

The subject of computer memories jogs my own memory a bit. The usefulness of memory has evolved from a secondary role in the earliest computers to a present primary role. This early role was probably undeserved and unwanted. The fact is, there wasn't much to work with at first. Memory circuits then (and now) were more cantankerous and frustrating than any other. Practical engineers chose those which offered some degree of quick success. Delay lines provided basically serial memory — that is, information was put away, and recovered, one digit at a time. Around this delay line memory grew a serial arithmetic system together with serial control sequences. Logical complexity of these machines was confined to the sequences, and otherwise time-oriented steps, performed on the data flowing to and from memory. Many in those days said that with a rather unlimited fast memory (say several thousand words) a huge improvement could be had. The obvious advantage of parallel memories (all bits of the word at once) broke down any economic obstacles.

The feeling that unlimited high speed memory would give advantages has persisted even with enormous increases in size. The idea of primary and secondary memories, a kind of conscious and sub-conscious, allowed for magnetic drums and tapes. However, it remained for the magnetic ferrite cores to provide a really satisfactory parallel

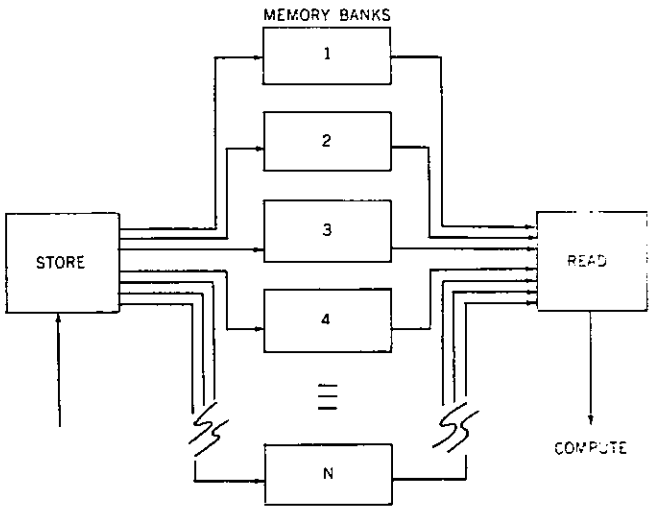
primary memory. The key advantage was parallel operation with no penalty for referencing in odd order (random access). The ferrite memories have become most successful and reliable and provide high-speed memory measured in hundreds of thousands of words. It is nonetheless interesting that the problems to be solved by computers continue to far out-strip this explosive growth.

The matter of primary and secondary memories, of course, offers a variety itself. They take the form of temporary and fixed stores, index stores, in-out buffers, and so on. They range nowadays from transistor registers, small temporary stores of film and ferrite, modular ferrite memories of large size, magnetic drums, magnetic disks, magnetic tapes, magnetic cards, optical stores – an endless array. The continuing success of ferrite memories has led to some intrinsically different methods of use, of which the coincident-current and word-organized memories are the leaders.

With the logic circuit performance keeping just one jump ahead, the memories continued to represent a large part of the time spent in operations. An admittedly brute force improvement in large memories was the separation into several banks of memory, with overlapping of cycles. Truly parallel banks of memory evolved to give an added dimension to the term parallel computer. Through all this, the original concept of primary and secondary holds with its one major problem: the somewhat untidy shuffling back and forth of data between the two. The very necessary data are naturally kept in primary memory, and the little used files in secondary. It is the in-between ground which seems to defy any order.

Some attempts have been made to make sense of this problem. There are schemes of addressing all data, primary and secondary, with somewhat automatic transferring when necessary. Other schemes use direct block transferring at very high speed. It would seem that more parallel trunks for these transfers would help. One trunk could load and another empty large chunks of primary memory not presently in use. In fact, several sets of these might be worthwhile. A fundamental assumption is made, however. If the trunks are to be usable, they must be separate; and there must be a comparable ability to compute on other primary memory at the same time. In fact, in the worst case a meshing of all these operations must be possible

by parallel trunks, time-sharing, random ordering under an over-all control. To obtain this final effect, much more than the memory must be considered.



What is described above could be called another step of parallelism, i.e., parallel-by-function, to be added to the bit parallel, word parallel, and memory bank parallel schemes. It is simply the idea of more things being done at the same time.

Illogical Waste

The computing is done on the data at a point in its trip from memory and back to memory. Most computers contain at least one place outside of memory for holding intermediate or partial results, usually an accumulator. Data to be carried over from one operation to the next can be placed in this accumulator and recovered from it. In fact, it forms a one word high-speed memory attached to the arithmetic and must have a path to the main memory as well.

A good many sequences of operations contain several cases of cumulative results. More than one carry-over register would be attractive for these cases, especially if a net speed improvement were possible. On the premise that transistor register storage is substantially faster than magnetic memory (say ten to one), a number of registers would allow good isolation from memory. These registers would require refilling from memory for incoming data and emptying to memory for final results. Otherwise, the partial results would arise from the computing activity. Thus, it can be seen that memory access is a secondary process as far as time is concerned and is mostly masked by computing time (more on this later).

Typical computer instructions contain memory addresses for the incoming data and the results. By removing memory to a secondary role, most of the computing instructions can refer to the transient registers. A considerable instruction word efficiency is thereby accomplished, since a few bits will entirely identify a small number of registers . . . whereas many bits are needed for the full memory addresses. This may appear a useless efficiency. However, it is an essential part of a new approach along with the concurrence of parallel memories and parallel functions. I plan to describe, from this point on, how this new approach removes a good deal of the logic waste mentioned earlier. Needless to say, this approach is exemplified in the **Control Data** 6600 Computer.

Sequence

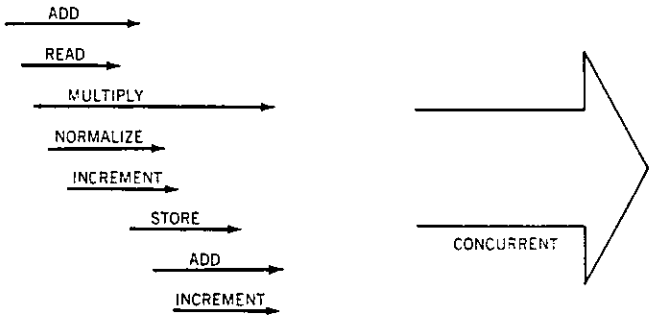
In any computer program, the results are obtained by the execution of *sequential* operations. Among these operations are some whose order of execution is unimportant to the result. In fact, the operations tend also to separate into somewhat independent trains, some housekeeping, some computational, some memory, and so on. These independent trains occur (or can occur) nested, so to speak, in the total sequence. A typical computer makes no attempt to take advantage of this nesting. Each instruction is taken in sequence and performed in sequence. If the computer had several arithmetic units of independent nature, and the ability to discriminate between those steps which must retain the original program order and those which need not, a positive improvement could be had.

It isn't difficult to visualize a number of independent arithmetic units. However, it requires a very detailed examination of each instruction to determine how to discriminate on the sequential order. Back to the above plug-for-simplicity, here is where it really counts. Simply stated, orderly sets of instructions can be checked for sequence order quickly and efficiently. The conditions which make up the basis for the order of events to follow can be logged and up-dated. A quick decision can be made on which kind of order constraints are active, and a proper next step can be taken. The next step can be in the form of a go-ahead or a wait until conditions are more suitable.

This cannot be visualized in the same way as the typical sequential machine. In such a machine, some underlying control mechanism, e.g., a pulse,

is formed at the beginning of a computation and proceeds through paths in the hardware like a mouse in a maze. Sometimes the pulse is duplicated for parallel controls of the data. However, only one of these duplicates provides the sequential continuity to the next step.

In the multiple unit machine, the control system begins similarly. A pulse is formed in the beginning, and sequential steps are taken up to, but not including, the first actual arithmetic or logical operation. From that point on, this original pulse is spread to a most complex network of paths, of which no sensible connection with sequence can be seen. This network serves to maintain an up-to-date reservation list on all units and transient memory registers. New operations can begin execution only if reservation conditions are favorable. Once an operation is issued to its unit, its reservation is made and thereafter monitored until the execution is complete. During the execution, the conflicts of use of trunks, registers, and the order-keeping are more or less an automatic part of this reservation control. As new instructions are brought up and thrown into this caldron, the order of their arrival is the *only* information about the ultimate desired order. Inside the caldron, late arrivals may actually proceed ahead of their turn as long as no impediments exist. (Note: It's like supper out. I've always been delayed getting a table for six.)



Despite all the confusion in describing such a system of multiple units, it makes no sense to have them if they cannot operate concurrently. More than that, concurrency is our only way out of the wire-speed limit. There are drawbacks to a complex system such as this. However, the drawbacks are almost exclusively on the side of design and manufacture, not on the use of the computer. The only reasonable question to ask is: "Do the difficulties of design and manufacture result in cost or competitive disadvantage?" Let me discuss the general subject of design and manufacture.

At odds

One thing has characterized the history of computer design more than any other: flexibility in the small. Building blocks made up of identical repeatable circuits have been constructed into generalized groupings, in themselves very flexible. To keep the number of these groupings small, for logistics and manufacturing reasons, some waste is allowed. By and large, however, the waste is minimal and pays off in over-all flexibility. Design involves mostly the complex interconnection of these grouped circuits obeying the well-established ground rules. Manufacture of the circuits proceeds somewhat independent of design, once the basic groupings are fixed and estimates are made of the number of each. Now then, with standard building blocks, the importance of wiring between them is obvious. In fact, the wiring allows the flexibility, so to speak. We have seen that wire length and speed of signals on wires are fast developing into a limitation. It isn't hard to see that wiring must be minimized, shortened, removed, or otherwise offset. Also, it isn't difficult to see that flexibility may be lost in the process. In truth, the two are really at odds.

The passing of time

The first thought in minimizing wire length is to reduce it. Make everything smaller. Yet work is performed on everything but the wire in this effort. The result is great reduction of circuit volume with no reduction in wiring volume. The wiring volume is now about half the total volume.

If the circuits could be more carefully or cleverly grouped, it might be possible to remove some wires. This very laudable thought hits directly at the idea of flexible circuit blocks. Of course, a multi-level method of assembling modules is possible (the mother-board technique), but this is also basically inflexible. No, it's too bad, but flexibility has got to go. What is the result? The principle effect is in design with some small effect on manufacturing. The design-and-build process is lengthened, since manufacture must wait for complete design. What I have touched on in the last few paragraphs is the very real present-day problem facing the industry. Without exception, the techniques being formulated for the next round of electronic equipment are based on design inflexibility except at very small levels. Integrated circuits offer no improvement unless coupled with more complex groupings to minimize or remove wires.

Depositing techniques demand geometric and topographic design of whole groups of circuits. Multi-layer printed wiring requires photographic design processes. These are *all* at the engineering and designer level, not below. The entire direction of the computing industry is toward *design* inflexibility.

The advantages to be had must, of course, outweigh the loss of quick design and quick change. The idea of parallel functional performance appeared early to be a design problem. But comparing it to the more basic trend of design, the problem of inflexible design technique is already here. Machine speed per dollar is still the principal competitive issue in selling computers. Any and all schemes for improving speed will be tried. The fact that they cause dislocations in the designing and manufacturing is merely a sign of the passing of time.

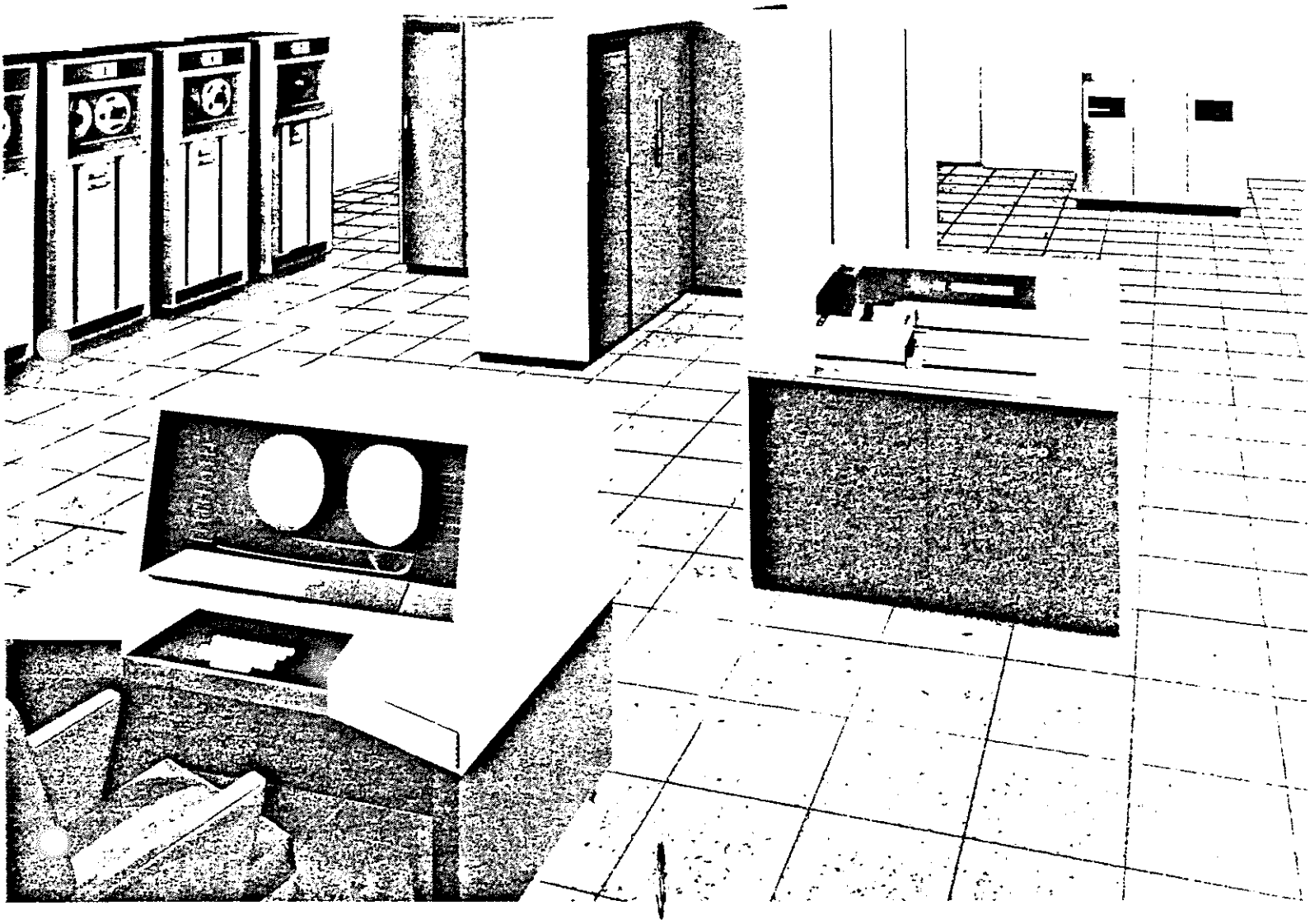
ON TIME

Aside from the weather, time is the subect of more casual discussion than most. In respect to the weather, time may be considered the opposite, in that its passage is highly predictable. In another way, time is very much like the weather. Both have a kind of fundamental rhythm or motion. We are familiar with the inexorable motion of time when we want to slow it; we know its tortuous passage when we want to speed it; we know the gradual acceleration of time as we grow older. In spite of our subjective notion of time, we live by it, watch it, cook by it, and measure by it. It is a distinctive element in almost every modern measurement or analysis, whether physical, chemical, statistical, or whatever.

What is it?

It is difficult to describe time. It is certainly one of the dimensions of the physical universe; but it is that unusual dimension with only one direction: positive. Its measurement has progressed from the hourglass, water clock, pendulum and crystal to the "atomic clock." Man's attempts to give a standard accounting of time have encountered such devious problems as the beginning hour of the day, the duration of a year, and daylight saving time, among hundreds of others.

The principle objectives in accounting accurately for the passage of time are to measure *what is done* in an elapsed period of time and to *synchronize* one



The CONTROL DATA 6600 Computer System

activity with another concurrent activity. The range of measurement is considerable. Biological, mechanical, atomic, atmospheric, and astronomical events operate in grossly different scales of time. It is of interest to examine the relationship of the duration of real events of a scientific nature with the corresponding length of time to solve a mathematical model of the event. Scientific computers were conceived for this work; and for the scientist, the computation time is of critical importance. For him, it constrains the depth and complexity of his model . . . along with the strain on his patience.

This time is real

This is, of course, the familiar "real time" computation. Visualize an atmospheric model of really comprehensive detail. (Now I have succeeded in discussing the two most discussed subjects in the world, time and the weather.) Could such a model be solved as fast as the weather? I know very little about meteorological problems, but I would expect that the thermodynamic and hydrodynamic computation on a world-wide scale is enormous. It would be a very happy circumstance if the mathematical model could be reduced to a workable size for machine solution, and still be effective.

Computing to date has been almost exclusively slower than real time, with notable exceptions in some military cases. These cases demand shortened sights and perhaps qualify only marginally as scientific. This little drawback hasn't restrained the burgeoning computer industry one little bit. The fact is that only occasionally, in the past, has there been a real demand for such speed. Many problems which appeared amenable to solution merely required a single result, once and for all time. Others were sufficiently beyond hand-solution as to welcome the machine's help. These kinds of problems point out or corroborate a new direction, a decision, a solution; or perhaps they fill a gap in the store of knowledge, to become useful later.

Computers made possible the attack on problems which were never attempted before (no one lost his job to the machine). This unusual circumstance is bound up in time relationships. The machine was built to operate without error for a certain period of time (usually as good as the designer could do). The computation, or some major part of it, had to be possible in less than that errorless period. In order for the first computer to be successful, its

speed had to be very high or its health very good! There was some threshold of speed and reliability under which the computer industry conceivably might *not* have been launched. This time relationship was enough to make computing machines practical.

The question of which problems were practical involves another time-speed relationship. Not *all* problems were now practical; only those which could be completed in the life of the machine, the duration of funds, the patience of the user, and so on. This really means that brand-new problems are available *each* time a faster computer is made, not just the first time. It isn't at all difficult to see that the impetus to make machines faster arises from these new problems along with the speedier solution of already-practical problems. We have been discovering a surprising backlog of new problems. The continued—in fact, accelerating—demand for more speed means that more efficiency is needed in the basic machine operations as well as in the use of the machine.

It should be obvious that any treatment of the methods employed in a computing machine must include a substantial discussion of time. It is the single outstanding obstacle met by the designer at every turn. In the following pages, I hope to tread a forward path in the attempt to overcome the time obstacle. Remember that time has a way of fighting back!

We're running out

Modern computer circuits employ high-speed switches for the complex decision networks. These switches require a finite time to change from one state to the other. This time period is an intricate balance of the electrical demands and constraints of the immediate surrounding network. Many careers are devoted to optimum combinations of materials, geometries, packaging, and processing of these switches to give the maximum speed with respect to a set of operating specifications. Many careers are devoted to finding the optimum adjustment of operating specifications to take advantage of the best available switch. Needless to say, the degree of perfection in this optimization is among the highest known in any field. Designers of the newest computers are able to depend on extremely fast and reliable components. It is no longer possible to foresee a factor of five or ten times speed improvement in the components now in use, or

like them. A factor of five or so was a working requirement for beginning a new computer not long ago. This factor came exclusively from the basic circuit. Claims made for many new computers tend to skirt this issue and concentrate on other time considerations (for example, lumping all of the man-and-machine times together). This is certainly understandable and entirely valid. BUT, the issue really can't be skirted, if we wish to move the computing machine up to real time or other comparable uses.

Set your watch

I mentioned earlier the synchronism of concurrent operations. This is, in some quarters, the signal for an immediate argument. It *seems* self-evident that two mechanisms working in unison must be synchronized if they are to work together. Actually this is entirely true; the argument is over a more subtle complication of the mechanism timing. If two mechanisms are to operate concurrently on two suitable portions of a computation providing answers to a third mechanism, the third cannot proceed until both answers are there. This is, in itself, a definition of synchronism. The two concurrent mechanisms may be constructed in a way which insures their simultaneous completion; or they may be constructed with no thought of the completion. In either case, it can be demonstrated that some time waste occurs. The very early computers were designed with a "tight" timing system. That is, every step of the computation (in fact, every simple decision or command) was activated by a central clock. The principal reason was that these early machines used many simple steps in a small amount of simple circuits to make up a major operation. As the economy allowed for more complex circuits, the very tight timing has given way.

Waste of time in a tight timing system is apparent in every step, since the logical decision made must be accompanied by a temporary storage. The storage allows for the circuit tolerances and resynchronizes any concurrent events. However, the circuit tolerances (with regard to time) are not allowed to accumulate beyond the single time period. The circuit tolerances have an upper and lower limit. If these tolerances are allowed to accumulate over a long series of steps, the earliest or latest time for the answer would vary considerably. Eventually, this spread of time makes for time waste, especially with devices which have a

minimum and maximum *rate* of operation. Notwithstanding the tolerance problem, the unclocked methods offer some advantages.

The synchronism problem is, most assuredly, an engineering problem rather than any other. The methods which I have mentioned are entirely valid. That method which produces the most effective result should be chosen. Matters of electrons, voltage, heat, and time have considerable bearing on that choice. The result must be a clock system of dependable tolerances and yet highly effective. In this case, a choice of synchronism in-the-large seems most effective. Computer history can record a long period of comparison by clock frequency. That day is gone. No longer is the basic clock a reliable measure of the performance. The simple reason is that there either isn't any clock at all or that synchronism has moved to a higher level. It is sufficient to say here that the move was fruitful.

Other internal time considerations are also important. The most common one mentioned is the memory access time. This is defined as the time taken to fetch a word from memory. It is normally measured from the instant the address is formulated until the fetched word is available for computation. This is usually about half the total storage time in destructive memories. With one memory, a three-address instruction would require three storage times plus compute time. With two memories having the ability to overlap the second access with the first restore, the above case could be one and a half storage times (three access times) plus compute time. The smaller the ratio of access time to storage time, the better this overlap system looks. Note, however, that the overlap doesn't work for addresses to the same bank of memory.

An extension of the overlapping memories might simply add enough memory banks to reduce the probability of referencing the same bank. To this can be added schemes for overlapping more than the access periods and schemes for reducing addressing bottlenecks. These are certainly important and represent significant speed improvement. However, memory time is typically an integral sequential element in *every* instruction, and as such cannot be reduced to zero. That is, it can't unless memory acquisition is separated from the instruction. To accomplish this, a set of high-speed registers may be included in the computer to serve as

buffer between memory and arithmetic. These registers must refill concurrently with computing and must empty to memory also concurrently with computing.

A concurrent structure such as described above places the memory in a secondary role of refill and empty. Time for this secondary role is a vague complication of memory bank overlaps, conflicts, priorities, and so on. It defies generalizing in the time domain. It, nonetheless, makes for a faster computer and points the way to even more speed. It is most important to note here that this speed increase is entirely aside from circuit or component speeds.

By now, the reader will be aware (and tired of hearing it) that concurrency is the magic way around the time obstacle. The technique need not be limited to concurrent memories. Arithmetic units may be arranged to take advantage of this technique. In fact, it is within reason to consider independent and concurrent processors as an example of the principle. For principle it is, just as serial and bit-parallel computing represent the evolving principles in the past.

Up and down

I can't leave the subject of time without including up and down time. Machines are subject to an imperfection never quite so small as to be neglected. To be sure, methods are available to make this bearable.

Time plays a part in these methods. Aging of components is no longer a primary factor in machine failures. Preventive maintenance allows the machine to be exercised under stress and under critical examination. For such examination to be critical the engineer must have enough time to thoroughly test each element. Here is where the very fast computer really shines. Many more trials may be made in a period of real time than with slower computers. Failure may be stated as a statistical function of the number of trials. *One* failure of a device labels it as faulty but may not be enough to discreetly identify the culprit. Several errors under the critical eye of the maintenance engineer may suffice to identify it. Therefore, the higher rate of trials in real time distinctly improves the maintenance. Axiom – faster computers are more reliable.

LOGIC AND NUMBERS

To a logician, most deductive reasoning can be formulated with symbols and rules very similar to mathematics. In fact, arithmetic could be described as the set of laws governing the logic of numbers. Numerical computation is the logical manipulation of that class of symbols called numbers. Computing machines, of course, are constructed to obey the rules of arithmetic. A common understanding about these machines is that their basic elements are *arithmetic* in nature. Such is not the case. The basic elements are only *logical* and must be especially interconnected for arithmetic.

The machines contain wired-in deductions concerning the beginning arguments. The "wiring-in" is accomplished according to a generalization (about the numerical rules or other logical rules). The deductions are certainties arising from this generalization. By appropriate experiments, the deductions may be tested, with the resulting confirmation or rejection of the generalization. Since the rules governing the wired-in logic of the machine have been fundamentally arithmetic, the confirming experiments are well known. In fact, the generalizations made in the first place are well known and proven.

To be certain

The procedures for using the machine are also based on a deductive method. The factual certainties arising from these procedures are also subject to confirming experiment. The machines, one could say, must first be tested and proven; then the use must be tested and proven. Since the principal use of computing machines has been arithmetic, the problem analysis and the method of solution lend themselves to reasonable test.

Of course, the machine can be turned around and used to perform the tests itself. Assuming the wired-in logic is entirely confirmed, the machine may test the proposed use by solving an experimental problem and comparing with a known answer. The solution is found by an organized program of basic machine steps. We stipulate here that the basic steps are proven. Therefore, the experiment should show that the program represents an accurate and correct generalization of the solution. If the test fails, some aspect of the generalization (or its specific embodiment in the program) is rejected.

There are two points of view about this facet of computing machines. The less there is wired into the machine in the way of logic, the more freedom there is for the programmer. On the other hand, with little wired-in logic, the chance for error (of a logical kind) is greater. I suppose this will be subject for argument forever. The current practical solutions contain a minimum of wired-in logic. The principal reasons for this cover areas such as: inability to provide a universally acceptable higher level of logic, substantially longer development periods for confirming the logic, and simple economics of the extra hardware. None of these need be a permanent deterrent to more internal logic.

What has happened in recent years is an attempt to establish this higher level of logic or reasoning by means of *program* organization. Deductive reasoning demands unambiguous symbols and words as well as the grammatical rules of language. Actually, some of the reasons why higher levels are not built in the machines apply to the programming as well. There is a chaos in the present-day universal languages. The development periods for the programs are fully as long as for the basic machine. Huge expenditures of time and money have been made for the effort. Perhaps a look at the logic already built into a modern computer would help.

You pick yours . . .

The fact that computer circuits are more logical than arithmetic is of considerable interest to the student of artificial intelligence. To the engineer, however, the circuits reduce to a very basic switching logic. In this form, open and short circuits represent the arguments and deductions. Electrical current is made to flow (or not) in resistance by the action of transistor switches. The resulting voltage causes other switches to close (or not). Combinations of switches cause various results. These combinations remain fairly simple since the electrical constraints, along with speed losses, limit the kind and number of interconnections. Being simple, the combinations lend themselves to proof by truth tables. This is a form of symbolic logic itself in which initial conditions are the coordinates of a table and the results fill out the table.

Simple combinations can be wired and connected end to end in sufficient chains to form a complex

logical combination. The number of combinations possible increases rapidly with each added link in the chain. In order to perform simple arithmetic on whole numbers, those logical combinations are selected which obey the rules of arithmetic. It is entirely possible to construct logic for any known number system. However, the binary-octal system is formed by the simplest logical combinations of switches, and this is the most commonly used system inside the computer. Converting between number systems is a logical operation which can be built into the machine. This particular question is determined by the designer with most regard to the average time wasted in a computation converting and re-converting between the internal number system and the external. In some cases, the total time thus spent has been demonstrated to be so high as to warrant use of the external system (usually decimal) internally as well. This is a fairly good example of the selection of wired-in versus programmed logic.

. . . I'll pick mine

It is, of course, necessary to deal with numbers other than whole numbers, for example, fractions. It is necessary to mix, group, and compare numbers in more and more complex ways. Especially in solutions of scientific problems, the range of magnitudes is enormous and not very predictable. For these problems a logarithmic arithmetic is best suited. In modern scientific computers this is called floating point arithmetic.

There are a number of varieties of floating point methods, being different by the superficial detail, rather than fundamentals. Although this kind of arithmetic is a good deal different from simple integer or fractional arithmetic, these can be usually computed in the floating units.

The typical scientific problem is solved by repetitive steps involving intermediate and partial answers. As the problem solution progresses, the error in defining the original numbers is increased by the errors involved in each arithmetic step. This doesn't mean that the deductive logic of the machine's circuits have somehow produced uncertain answers. At the level of the circuit logic, the answers are still certainties. However, in interconnecting by wires and by program steps the floating point operations, an interesting limit occurs. It is possible to manipulate numbers within the machine of a certain maximum size, or number

of digits, that size being limited by the size of register built into the machine. Numbers can therefore be introduced with a limit on the number of significant digits and thus with an error of something less than the least significant digit. This error is real and can contribute measurably to the accuracy or significance of the answer. For example, it is entirely possible for a long series of arithmetic steps to accumulate an error so large as to completely obscure and invalidate the answer.

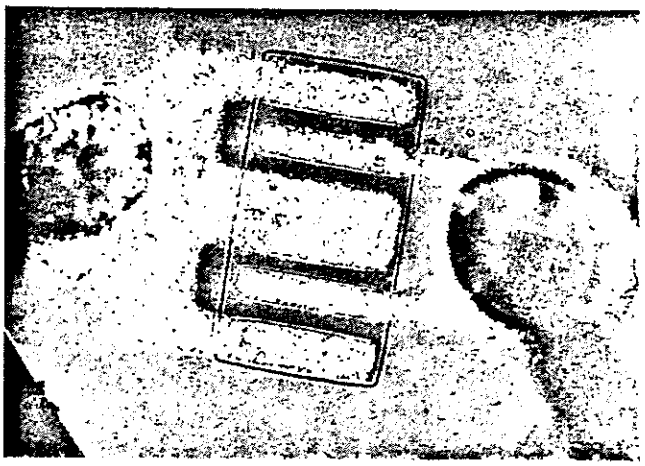
There are methods available for minimizing this sort of floating point loss of significance and accuracy. They range from well accepted methods to quite radical techniques. It must be the designer's duty to provide for as many of these techniques as possible without loss of convenience or speed. This is a good example of a very difficult selection of wired-in versus programmed logic. The logical methods available to the designer are fixed by the nature of the circuits he uses. These are usually of a very simple kind, and he is required to work at the most basic level. It is a tedious hand job, especially as the complexity of the circuit modules increases. There are no barriers more positive to the computer engineer than his use of the last connector pin or transistor location. Fitting the hundreds of thousands of elements together in a sensible array demands a sensible plan.

It can be stated with assurance that the optimum amount of built-in logic will be subject of heated argument. In order to be completely flexible, some convenience and speed must be sacrificed. In order to aim at a higher efficiency in some area, other areas may suffer or be completely ruled out; hence, limiting the general-purpose aspect of the machine. As a matter of fact, though, these thoughts apply to the program standards as well. In order for machine languages to be thorough and efficient, some loss in flexibility will be apparent.

The modern computers have provided a few higher level built-in operations along with the basic ones, most notable being the floating point. The risks in going further are very great. One of the few criteria which makes sense is the computing speed per dollar of cost. The larger the computer, the more freedom there is to include extras. However, it is important to realize that the very large economy-size computer is not feasible unless the user gets large economy.

OF PARTS AND MECHANISMS

In this industry, progress has kept moving with little advance warning of new directions. The lure of profit stimulates innovation, and the spur of competition forces the early arrivals to defend their positions by improved performance. This is a process, described as "creative destruction," where not all are winners. Progress in computing is directly related to time and performance, with economic factors closely following. It is the step up in performance that the engineer seeks by new devices and new logic. The first endorsements don't come from the economist, but the lasting techniques *do* need the stimulant of wide acceptance with resulting savings. The techniques open to the engineer, therefore, tend toward a rather narrow range of devices.



It is especially interesting to me that the most modern universally-accepted device for computer circuits, the transistor, is a triumph of geometry. Most of the recent improvements in the transistor come from ingenious methods for providing a thin layer here, a thick layer there, a large surface here, a minimum surface there. The beauty of geometric design has long thrilled men. The transistor mixes crystalline symmetry with etched surfaces visible only by microscope. More than this, the electrical reactions of the transistor can be sharpened to really surprising speed. For the engineer familiar with "lumped constant" effects, the modern components are a revelation. (Note: Lumped constant effects refer to idealized electrical engineering methods.) Although transistor speeds are still in the order of several hundred times slower than the speed of light through the space taken up by the device, the speed is still a surprise. Transistors make excellent switches when limited to low volt-

ages. The speed with which such switches can be opened or closed has increased by several hundred since they were first introduced. Wide acceptance has added the economic stimulant to the very desirable properties of the transistor.

Machines built with transistors today utilize the most simple known circuits. Several variations are available with relatively equal simplicity. The designer's choice is formed from an amalgam of component capability, size and shape, and the geometry of the packaging. Speed being the prime objective, heat, power, construction methods, and so on are the variables for his use.

Good losses

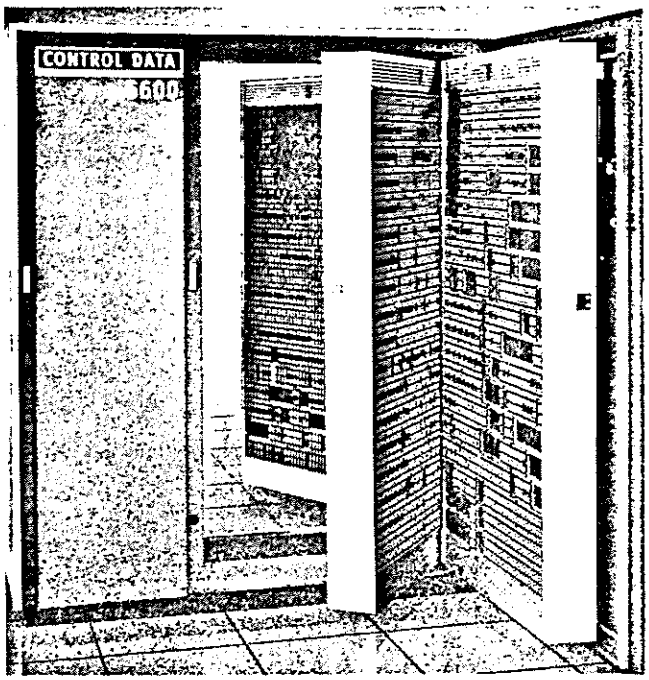
A recent packaging technique with very good efficiency of volume usage is the "cordwood" package used in the **Control Data** 6600. This package gives four surfaces for etching the interconnections. This structure is a step up in module complexity from the small single-board cards of recent years. The density of circuits per unit volume is up by three or four over the cards. A number of gains and losses arise from this kind of packaging. The loss of most importance is in standardization of modules. The package contains so much logic that a flexible minimum set of module types would sacrifice considerable potential efficiency. Another problem is the extra logical complexity of the module. It is of little use to apply mechanized techniques to help with the design of these modules. The job of *designing* with them becomes once more, as in the past, a hand job. Engineering of computers utilizes geometric and topographic methods more than ever.

These are "good" losses, though, since the gains far outweigh them. The improvement in volume density is significant, and well worth the effort in improved speed. The increase in complexity within the module allows for two conditions of circuits, those inside and the interconnections between modules. The effect is to group logic more efficiently in modules so that advantage may be taken of the internal speed and loading rules. Internally, wire lengths of less than three inches are encountered, whereas the average external lengths are perhaps ten times that. This reflects in longer transmission times. The external lines may provide only enough current and voltage to supply (without further transmission time) a fraction of that available internally. These and, of course, the problem of module connector pins and back panel wiring volume all add up to a plus for the more complex "cordwood" package. The loss of flexibility is unfortunate but by no means defeating.

Other more specialized modules are also possible with such techniques, notably the memory package. With suitable connectors and internal framework, a memory may be constructed with complete addressing and storing circuits in one package. This more or less reverses the losses just mentioned, since a memory package may be considered a standard unit to be "plugged in" wherever required. In the **Control Data** 6600 Computer, such memory modules are made up in 4096 word (12-bit) size for use in the peripheral processors. Also, five modules make up one 60-bit memory bank in the central memory.

Form, not dimensions

Packaging techniques which greatly increase the density of circuits are also likely to increase the heat density. The choice of circuit open to the designer may allow a low power dissipation, but generally no large factors are possible, especially for increased speeds. Cooling, or maintaining constant temperature, is very important. Moving air past the dissipating element has been fairly successful in the past. However, one aspect of higher density is the restriction of air flow. Cooling by cold bar conduction, radiating fins and plates, circulating coolant, and the like are among the way out of the dilemma. The 6600 Computer is freon-cooled.



In very large systems, the sheer volume of logical and memory hardware demands several cabinets or bays of chassis. The length of interconnections between the different portions of this array may be a serious speed problem itself. No design is so cooperative as to allow neat groupings adjacent to each other without the long wire. The most effective geometric forms are the cylinder with interconnections at the axis or the cube with interconnections on the surfaces. The sphere, of course, might appear superior to either. However, the fabrication complexity is a significant drawback. My personal preference is the cylinder. The axis can be the location of interconnections as well as the pivot for moving aside the adjacent parts of maintenance. The principal advantage is in uniform interconnection lengths with quite practical fabrication methods.

How to succeed . . .

I suppose the picture of computing is of a topsy-turvy growth obeying laws of a commercial "natural" selection. This could be entirely accurate considering how fast it has grown. Things started out in a scholarly vein, but the rush of commerce hasn't allowed much time to think where we're going. In fact, the essential organization of computers hasn't changed at all. The real differences are in the fringe "special effect" operations and the internal methods, which are hidden except for their effect on performance. Even the peripheral systems are quite similar to each other.

. . . without being trying

In my mind, the greatest potential for improvement is with the internal methods (if this isn't already clear), at the risk of loss of fringe operations. The work to be done is really engineering work, pure and simple. As a matter of fact, that's what the results should also be – pure and simple. It's time to set about developing new wiring schemes and new packaging schemes that really fit together. The best of what has been done should be the guide. Most of the time, the best isn't very spectacular or clever; it's just the best. Physical volumes won't reduce as quickly as we'd like; but they will reduce some. Building blocks won't be very flexible; but they can be made neat and tidy. Transmissions of signals won't exceed the speed of light; but sometimes delays are useful. The direction is clear and we'd best get about it, before our elephant stops growing.