EFFECTS OF LARGE ARRAYS ON MACHINE ORGANIZATION AND HARDWARE/SOFTWARE TRADEOFFS

L. C. Hobbs

Hobbs Associates, Inc.
Corona del Mar, California

INTRODUCTION

From the early days of electronic computers until the present, a period of over 20 years, electronic and magnetic hardware for mechanizing logical functions and storage in the central processor portion of a computer system have been extremely expensive. Although these costs have been dropping steadily in terms of the cost per component, increases in the complexity and capacity of central processors have tended to keep pace with decreases in hardware costs. Hence, reductions in hardware costs to date have been reflected primarily in increased performance and capability rather than reduced cost. However, developments presently underway in batch-fabricated technologies will provide such significantly lower hardware costs in the central processor that it will not be possible to maintain a system balance from the standpoint of cost and reliability. If properly used, large-scale integrated-circuit arrays in particular will provide digital logic and control functions at such sharply reduced costs and increased reliability that the central processor will tend to become an almost negligible part of the system from the standpoint of both cost and reliability. The dominant factors in systems cost will be software and electromechanical mass storage and input/output devices.

As a result of these technological advances in batch-fabricated hardware, the three major problems facing designers of future computer systems will be:

1. The necessity of developing machine organization techniques that will permit the efficient utilization of large arrays to achieve their true potential in terms of cost, reliability, and maintainability.
2. An urgent need to minimize the number of electromechanical mass storage and input/output devices required in a system in order to reduce systems cost and increase systems reliability and an accompanying need for developing new and improved types of such peripheral equipments.
3. An equally urgent need for minimizing the cost of providing software, including both operating systems and user programs, even if this requires significant increases in the logical and storage hardware in the central processor.

MACHINE ORGANIZATION IMPLICATIONS

In considering the advantages of large arrays it seems apparent that the larger the array that can be effectively utilized the better the economics and reliability up to the limit that can be achieved technically in terms of the number of components per chip.[1] The use of very large arrays will reduce the initial fabrication costs and improve reliability and maintainability, but they will also present a serious problem. As the module becomes larger it becomes increasingly difficult to use it for more than one function within a single computer. Each packaged unit tends to become unique with only a single one of each type used in a given computer. Dr. R. N. Noyce called attention to this last year and indicated the anticipated progress in array size when he stated:

> However, from a point on the complexity scale now where 50 components is the cheapest level for an integrated circuit, I expect to move to 1000 by 1970. . . . At the same time there will be new problems where it takes only 10 chips to make a computer and almost every circuit made will be different.[2]

This decreased "commonality" increases fabrication costs because of the low production volume of each type of module. It also increases the cost of the spares inventory, but the cost of spares usage will decrease as a result of significantly higher reliability. In fact, the low rate of usage of spares coupled with the difficulty of repairing large arrays should lead to adoption of a "throw-away" maintenance concept where major portions of the computer are replaced but not repaired in event of failure. This will have significant effects on maintenance procedures and costs—particularly in military systems.

The lack of flexibility in large arrays which tends to make each array within a system unique and the possible need for eliminating bad or substandard circuits from the array to achieve a reasonable yield are two of the major problems in utilizing large arrays in computers. At least three different approaches to fabricating large interconnected arrays to overcome these obstacles to their utilization are under consideration. The first is cellular logic in which large arrays of identical circuits are fabricated with a standard interconnection pattern (e.g., connecting each circuit only to its four adjacent neighbors) with the ability to modify the function of the circuit by changing something in the circuit subsequent to fabrication.[3] For example, one approach of this type uses a circuit with four cut-points which can be cut in different combinations to alter the function of the circuit.

In the second approach, a large array of circuits is fabricated and each circuit is individually tested. The test results are put in a computer which is also storing logical equations of the function to be implemented. The computer then generates the proper interconnection pattern to interconnect available good elements (skipping the bad ones) to perform the required logical function.[4] In this approach, a separate mask must be prepared for each array fabricated; hence, this is an expensive operation unless cheap methods can be developed for producing interconnection masks under computer control. On the other hand this approach offers a major advantage in that it is easy to vary the function performed by the array by changing the logical equations supplied to the computer that is controlling the interconnections. If each interconnection mask for each array is generated individually, there is little incentive for rigidly standardized functions.

The third approach is advocated by those who believe that in the future it will be technically feasible to achieve high yields of large integrated circuit arrays in which all circuits are good. This would permit a standardized interconnect pattern to be used for each specific logical function. This has the advantage that only one mask need be made for a particular function. This mask can then be used to interconnect the circuits in many arrays of that type. On the other hand, it is more difficult to change the function to be performed by the interconnected circuit array since this requires making a different mask.

In the future both of the last two fabrication techniques discussed above will probably be used. Programmed control of the interconnection pattern will likely be used for small production volumes and unique or infrequently used functional modules. However, there is strong evidence that the semiconductor industry will produce large arrays with yields sufficiently high to permit the use of standardized interconnection patterns for functional modules that are used in large quantities.

As semiconductor and batch-fabrication technologies advance, the major physical limitation on the size of the functional unit will be the number of external leads that can be provided on a package. Although packages with larger numbers of leads (in the order of 100) are being developed, additional research in machine organization is urgently needed to develop functional organizational concepts that will maximize the interconnections within a replaceable package and minimize the interconnections between packages. The way in which the computer is divided into functional modules can greatly increase or decrease the number of connections needed between such modules.[5]

It will be necessary to use different criteria for design efficiency in batch-fabricated systems. In the past, minimizing the number of logical elements has been a major goal of most logical design efforts. In future systems, logical elements should be used inefficiently in order to minimize the number of interconnections needed between functional modules. For example, frequently in present computers a given gate or flip-flop supplies inputs to a number of logical elements in different parts of the machine; but in future systems the logical gate or flip-flop may be duplicated many times in different parts of the system to minimize the signals transferred from one module to another. Emphasis must be placed on reducing the number of packages and the number of interconnections between packages—even at the expense of increasing the logical complexity of each package significantly. Perhaps an even more difficult problem will be motivating logical designers and systems planners to use standard or predetermined functional modules. It is hoped that Dr. E. A. Sack's optimism was justified when he stated: "It is the author's opinion that the drastic reduction in cost per gate available in multi-gate arrays will overcome the system designer's natural reluctance to employ *prefabricated* digital functions." [6]

In order to achieve the cost and maintenance advantages offered by the use of large batch-fabricated arrays, it is necessary to develop machine organization and system design techniques that permit repetitive use of packages containing very large arrays of circuits. One approach is to change the internal organization and logical design of the large computer so that large functional arrays can be used repetitively even if this means that each array is relatively inefficient in terms of the utilization of circuits within the array.[7]

Another approach is to use very small standard modular computers designed to be used either individually or in multicomputer systems. In this case, the uniqueness of large functional arrays within a given computer is accepted. Such a small standard modular computer can be fabricated with a very limited number of circuit arrays each of which is used only once within that computer. For example, the complete program control unit and all of its internal interconnections may be fabricated in a single package, the complete arithmetic unit in a second package, the complete input/output control and buffering section in a third package, and storage modules in additional packages containing 2000 words each. Economies in fabrication and spares inventory would be achieved as a result of the volume usage of each type of module made possible by the use of a large number of these standardized computers rather than by the use of a large number of identical packages within a single computer. When additional computing speed and capability is required the standardized computer would be used in a multicomputer configuration.

A third approach is to develop parallel processing systems conceptually similar to those that have been discussed extensively in the literature.[8,9] In this approach, large arrays are used effectively by organizing the machine on the basis of a relatively large number of identical processing modules.

INPUT/OUTPUT IMBALANCE

There are three major approaches to the input/output problem:

1. Improvements in the performance of present types of input/output equipment.
2. Development of new types of input/output equipment that are not in widespread use at present.
3. System organization approaches that minimize the need for conventional input/output equipment.

Each of these approaches will play a part in providing better balance in future systems. However, unless much greater effort is placed upon the development of nonmechanical input/output equipment, the best hope for future systems probably lies in developing system techniques that minimize the need for input/output equipment. Although a problem of major importance, these have been discussed previously and will not be considered further here.[10]

HARDWARE/SOFTWARE TRADEOFFS

The memory capacity of early computers was so limited that programming costs were not a significant part of the total cost-of-ownership of a computer system. However, reductions in hardware costs have been accompanied by greatly increased memory capacities which have permitted the storage and operation of larger and more complex programs. The cost of hardware and the cost of engineering design required to efficiently use expensive logical components have exerted a strong pressure in the direction of very general-purpose computers which can be adapted to a large number of different operations so that design and production costs can be amortized over a relatively larger number of units. It has been recognized that a special-purpose computer can perform a particular task more efficiently than a general-purpose computer in terms of the amount of hardware required, but the cost of small volume production and specialized design have favored general-purpose computers.

Under these circumstances, the tasks of specializing the capabilities of a general-purpose computer to a specific job and adapting it to the control of a large number of different types of input/output and peripheral devices have been left to the programmer. However, the increased performance and capability of computers that have accompanied the reductions in basic hardware costs in recent years have placed greater and greater requirements on the programming necessary to adapt more sophisticated general-purpose machines to more complex operations in specific kinds of problems.

While hardware costs have been decreasing, programming costs have been increasing significantly to the point that they now represent at least 50% of the initial cost of a new computer system and perhaps as much as 80% of the total systems operational cost over a 10-year period. This problem is now magnified by new batch-fabrication technologies, such as large-scale integrated circuits and plated-wire or thin film memories, which are expected to reduce the cost of logic circuits and storage elements by one to two orders of magnitude. However, the effect of these hardware cost reductions on the cost-of-ownership (initial procurement cost and systems operational cost over the lifetime of the system) is limited by the overwhelming software costs which will not be affected by these advances in hardware technology unless machine organization and system design concepts are changed.

Fortunately, the significant reductions that are being achieved in the cost of logic and storage offer an opportunity to also reduce the mounting cost of software by trading low-cost hardware for expensive software. Many of the functions relegated to programming in the past because of high hardware costs can be performed in the future by low-cost batch-fabricated hardware with a consequent reduction in programming complexity and costs. Technological changes now make it necessary to reverse the past practice of using additional software to minimize hardware requirements. In the future, additional hardware will be used to reduce programming requirements. This can only be achieved by reevaluating the criteria used for making hardware/software tradeoffs in the past.

At least three different approaches to altering previously accepted hardware/software tradeoffs can be considered:

1. Special-purpose computers and processors.
2. Different types of machine language and machine organization.
3. Additional hardware functions in machines with conventional languages and organizations.

Special-Purpose Computers and Processors

The question of special-purpose versus general-purpose computers has been argued in one way or another since the dawn of the computer era. The major arguments against special-purpose computers have been design costs and lack of flexibility. Special-purpose computers have been frequently favored for applications where a relatively large number of machines have been required to do a certain set of fixed tasks, but in most such cases some limited form of program control (e.g., paper tape, plug board, etc.) has been added to provide some flexibility. With the advent of computer-aided design and computer-controlled preparation of masks for large-scale integrated circuits much of the design cost obstacle is removed. In essence the question then becomes one of trading logical design in a special-purpose machine for programming in a general-purpose machine. In this case, the logical design will probably win out in terms of the number of man-hours required since the logical designer can address himself to the task at hand with few predesign boundary conditions while the programmer does not have a completely free hand because of the characteristics of the general-purpose machine he is adapting to a specific problem.

The problem of flexibility remains, but this may be partially overcome by a compromise in a multicomputer or multiprocessor system. A discussion of the many advantages of multicomputer and multiprocessor systems is outside the scope of this paper, but in many cases it is not necessary that all of the processors or computers in such a system be identical nor that they all be general-purpose. A multicomputer or multiprocessor system is feasible in which some of the computers or processors are general purpose while others are special purpose, designed to perform specific tasks that are relatively common. For example, in a multicomputer scientific computation system one or more of the computers could be DDA's. As another example, in a multiprocessor system one of the processors could be a logical processor, another an arithmetic processor, etc. It seems fairly obvious that such use of special-purpose computers or processors will reduce the programming requirements (as well as probably increasing processing speeds) and will be economically feasible when the low-cost potentials of large-scale integration are realized.

Different Types of Machine Language and Organization

Present machine languages and machine organization concepts have been heavily influenced by the cost and capabilities of specific types of hardware in the past. The storage hierarchy is one example of this. The sequential one-address machine language is another. If word size were not limited by the cost of larger registers and storage, three-address machines would undoubtedly be more prevalent, particularly in data processing type applications.

Machine languages have been designed to permit efficient implementation of the processor itself rather than to facilitate programming. Users on the other hand have developed pseudo-languages that facilitate programming from a human standpoint but that require compiling operations that do not always utilize the true capabilities of the computer. On the surface there seems to be an advantage in using some higher order languages (e.g., FORTRAN or ALGOL) as machine languages, if hardware costs are sufficiently low. It will probably not be feasible to go this far, nor is it necessarily desirable. However, it is feasible and desirable to design machine languages that will facilitate compiling operations and to implement certain parts of problem oriented languages in hardware. The need for better problem oriented languages has been cited frequently.[11] Hence, a joint effort by programmers and engineers to first design better problem-oriented languages and then to implement portions of them (e.g., mathematical operations) with hardware wherever possible should pay handsome dividends.

The Burroughs B5000 with its Polish notation and push-down-list store represented an early step toward development of machine languages and organizations that will facilitate compiling operations.[12] Other work in this direction includes an Air Force-sponsored study at the University of Pennsylvania using associative memory and list processing techniques.[13] With very low-cost large-scale integrated-circuit arrays just over the horizon, it should be economically feasible to implement machine languages that will eliminate many of the steps in present compiling operations.

Additional Hardware Functions in Conventional Machines

It is not necessary to go as far as special-purpose computers or new machine languages and organization to achieve significant economies in software by greater, and perhaps "inefficient," use of low-cost hardware. Significant economies can be achieved within the framework of conventional machine languages and organizations by:

- Hardware implementation of special purpose functions and logical and mathematical operations.
- Implementation of hardware features that minimize "red tape" and "housekeeping" programming requirements.
- Hardware implementation of some of the machine functions presently handled by operating systems software.

Many functions presently handled by programmed subroutines can be implemented easily by special-purpose logic in a straightforward manner. Such functions include:

Binary-to-decimal and decimal-to-binary conversions
Code conversions
Coordinate conversions
Format control
Table look-up operations
Scaling
Mathematical operations (e.g., square root, trigonometric functions, matrix operations, etc.)

In the past such functions have been handled by programmed subroutines using the machine's basic operations (e.g., add, multiply, shift, etc.) because of the cost of the hardware required to mechanize the functions in relation to the frequency of their use and because of the flexibility offered by the ability to modify the routine either as it is stored or by index registers at the time of execution. From the cost standpoint, large-scale integration will make it feasible to mechanize such functions even when they are used relatively infrequently. The necessary flexibility can be retained by hardware mechanizations that permit program control of variable operations in such functions and still significantly reduce the software required. Hardware mechanization of functions of this type will not only reduce the programming effort and the storage space required for the program, but will also offer speed improvements since logical implementation of such functions is invariably faster than the execution of the equivalent sequence of program steps.

A large portion of most programs consist of "red tape" or "housekeeping" instructions that either are not conceptually necessary to the solution of the problem or that can be implicit to the operation performed rather than stated explicitly. These include operations such as:

Register-to-memory or memory-to-register transfers
Certain transfer-of-control operations
Some operations on the contents of index registers
Certain types of timing functions

Additional hardware can greatly minimize the number of operations of this type required in a program. For example, a set of general registers or a small high-speed control memory can be used to represent multiple accumulators, index registers, and control registers. The availability of such multiple registers will sharply reduce the number of register-to-memory and memory-to-register transfers, the number of index modification operations, and the number of transfer-of-control operations required. There are, of course, many other examples of this type. A somewhat complementary concept is the use of large-capacity low-cost storage coupled with higher-speed machine operation to permit the effective utilization of inefficient programs. This increases the size of the program in terms of the number of instructions involved but reduces the man hours required to write a given program by removing the need for polishing and streamlining the program to make it run faster and fit into less storage space.[11]

The operating systems software provided with most computers handles three major functions:

Input/output control and editing
Scheduling and storage allocation
Interrupts and priorities

The operating systems usually represent the most difficult and expensive area of systems programming. It has been estimated that one major computer manufacturer is spending $60 million this year for programming PL1, FORTRAN, COBOL, and the operating systems for a family of new computer systems. The operating systems probably represent at least one half of this cost.

Many of the functions included in operating systems software can be implemented by additional hardware. For example, special-purpose control logic and storage hardware can be provided with each type of input/output equipment to provide a completely standard interface with the computer so that the programmer and the software system need not be concerned with the nature or characteristics of the particular input/output device. Special-purpose hardware and buffer storage can accommodate the differences in characteristics of tape units, disc files, card readers, keyboards, etc. Small associative memories can be used to facilitate the cataloging and indexing of data files and the allocation of storage. Hardware can significantly reduce programming requirements in the servicing of interrupts and handling of priorities. Computer and I/O channel usage accounting can be facilitated by additional hardware.

Most present computer systems use a multilevel storage hierarchy which usually requires program consideration of the particular level of storage being used and to some extent the differences in the characteristics of devices used for different levels. Additional low-cost logic and special storage techniques can be used to cause this multilevel storage hierarchy to appear as a single homogeneous storage to the programmer, thus minimizing the need for programming attention to the capabilities and characteristics of the different types of storage. Many of the storage allocation, page turning, and memory protection schemes used for time-sharing, multiprogramming, multiprocessor, and multicomputer systems can be implemented by low-cost hardware also.

FUTURE PROGRESS

Special-purpose computers or processors may evolve naturally as multicomputer and multiprocessor systems are developed and used on an increasing scale. Hardware features to minimize housekeeping will also tend to evolve as designers find lower and lower cost elements and functions available to them. The hardware implementation of special-purpose functions is straightforward, but a catalog of present subroutines can give a clue to the functions to be considered for implementation.

Present compilers and higher-order languages are a good starting point for considering machine languages and organizations that will simplify programming; but, even with low-cost hardware, further research in programming languages is needed to determine a language closely related to users' problem oriented languages that is still economically and conceptually feasible to implement as machine language.

Much of the conceptual work necessary to implement operating systems functions has already been done. The large and complex software operating systems that have been developed during the past 8 to 10 years represent many man-years of effort in formalizing the necessary procedures and algorithms. Hence, the starting point should be a study of these operating systems to determine areas that meet three criteria—(1) difficult or unsolved problems, (2) significant numbers of instructions, and (3) procedures and algorithms that are more feasible for mechanization by large-scale integrated circuits or other batch-fabricated hardware. Software functions meeting any of these criteria represent a promising area for development. Functions meeting all three will literally represent a gold mine of software cost savings.

Joint hardware, software, and systems design efforts are needed in choosing new hardware/software tradeoffs to properly utilize both the results of past work and the capabilities of new technology. In the past, hardware has been traded for software in order to improve speed and performance with the decisions made primarily on a cost/performance basis. In the future, hardware will be traded for software to reduce the cost of programming with decisions made on the basis of total systems cost rather than only equipment costs.

When transistors were first introduced there was a strong tendency to use them in the same way vacuum tubes had been used previously. Unfortunately in the computer field a similar trend is in process now with integrated circuits being used in the same way that discrete semiconductors have been used in the past. Systems designers must consider large-scale integrated-circuit arrays as a new type of device that necessitates major revisions in systems design concepts, machine organization, and hardware/software tradeoffs.

ACKNOWLEDGMENTS

The author would like to express his appreciation and acknowledge the assistance of several people with whom the subject of this paper was discussed —particularly D. R. Ream, R. Rice, T. B. Steel, Jr., R. G. Tuttle, and D. F. Weinberg. Many of the ideas covered in this paper are an outgrowth of material developed during a study of "Technology Applications for Tactical Data Systems," sponsored by the Naval Analysis Group of the Office of Naval Research under N ONT–4910(00).

REFERENCES

1. E. A. Sack, R. C. Lyman and G. Y. Chang, "Evolution of the Concept of a Computer on a Slice," *Proceedings of the IEEE,* vol. 52, no. 12, pp. 1713–20 (Dec. 1964).

2. R. N. Noyce, San Diego Council of WEMA.

3. R. C. Minnick, "Application of Cellular Logic to the Design of Monolithic Digital Systems," *Microelectronics and Large Systems,* Spartan Books, Washington, D.C., 1965, pp. 225–47.

4. J. S. Kilby, "Device Fabrication," *Proceedings of the 1966 International Solid-State Circuits Conference,* p. 30.

5. R. Rice, "Systematic Procedures for Digital System Realization from Logic Design to Production," *Proceedings of the IEEE,* vol. 52, no. 12, pp. 1691–1702 (Dec. 1964).

6. E. A. Sack, "Complex Digital Integrated Circuits: An Opportunity for the Logic Designer," *Microelectronics and Large Systems,* Spartan Books, Washington, D.C., 1965, pp. 141–54.

7. R. Rice, "Integrateds—The Predictable Effects on Engineering," *Proceedings of the National Symposium on the Impact of Batch Fabrication on Future Computers,* pp. 237–53.

8. J. H. Holland, "Iterative Circuit Computers: Characterization and Resume of Advantages and

Disadvantages," *Microelectronics and Large Systems,* Spartan Books, Washington, D.C., 1965, pp. 171–78.

9. G. A. Crane, "Economics of the DDLM, A Batch-Fabricatable Parallel Processor," *Proceedings of the National Symposium on the Impact of Batch Fabrication on Future Computers,* pp. 144–49.

10. L. C. Hobbs, "The Impact of Hardware in the 1970's," *Datamation,* vol. 12, no. 3, (March 1966) pp. 36–44.

11. T. B. Steel, Jr., "Promising Avenues of Research and Development—Programming Research," panel discussion at 1965 FJCC.

12. "The Descriptor, A Definition of the B5000 Information Processing System," Burroughs Corporation, Detroit, Mich., 1961.

13. H. J. Gray et al, "Interactions of Computer Language and Machine Design," Technical Report No. RADC-TR-64-511 (AD617-616), University of Pennsylvania (May 1965).