DESIGN OF PROGRAMMING LANGUAGE PROCESSORS—II

SYMPOSIUM SUMMARY

CHAIRMAN: B. RANDELL (USA)

THE INTERRELATION BETWEEN PROGRAMMING LANGUAGES AND MACHINE ORGANIZATION

R. S. BARTON

General Electric Company

Several processor organizations have been proposed based on programming languages, notably ALGOL. Euler, a design based on a generalization of ALGOL, is perhaps the best example of such machines, and is of much interest for its built-in storage management and subroutine control functions. Language-based processors, however, are not widely known and understood nor do they seem to have been properly evaluated in comparison with more conventional machines.

Increased interest in non-numeric computer applications and the demands of multi-terminal time-sharing, have provided new motivation for giving attention to the subject of storage management, which now seems more important than language characteristics in determining processor organization. It thus seems appropriate to reconsider built-in functions for an information processor.

Programming languages describe the *structure* of processes and the data upon which they operate. Such structures can be discussed or abstractly transformed without regard to a particular machine's characteristics or the manner in which the process and its data are represented in storage. Tree-like language syntax leads naturally to the use of push-down stores but beyond this programming language developments seem not to have much influenced machine organization. Of course, one may not progress far into a realizable machine design without taking internal machine representation into account.

The *format* in which information is internally represented in the storage of a machine is a most fundamental consideration. A format convention must exist for a program to be interpretable; though it is customary in today's machines to make this quite inflexible with rigid formats for data and fixed instruction layouts. This practice prohibits much versatility in making one machine imitate another. The elusive measurement of "efficiency" is a weighting of storage usage and number of accesses; and, as every programmer has observed, minimizing both storage space and number of accesses are usually conflicting goals. To optimize the performance of an information processor for a particular application, some choice of data and program formats at machine language level should be available to the programmer. A general purpose machine should be able to imitate another machine effectively by allocating its resources in logic and storage in such fashion as to favor the most frequent operations in the machine being imitated. Direct use of the format of that machine is clearly advantageous.

Format is the description of a pattern of fields specifying lengths, sequence, encodings, and interpretation. Each sequence and storage region has a format associated with it and this format description might be distributed throughout a program and data region in storage. The two arguments against hardware provision for variable format: (1) increased equipment complexity, and (2) loss of speed stemming from restrictions on paralleling, are valid only if the performance improvements do not justify the cost.

Few machines have provisions for handling lists directly and in those that do the instruction set is merely supplemented by a few instructions of limited use. Lists, considered apart from list processors, are a simple and useful storage allocation and sequencing device with increasing utility in applications. The microprograms for link manipulation should be incorporated into the hardware since a factor of at least 10 in performance is at stake. One must avoid, however, too rigid a list format.

Paging, which can be looked upon as a form of indirect addressing, uses indirection with the high order bits of the address only; and is another mapping technique which allows storage to be partitioned into equal units of which any subset can be made to have contiguous addresses. Paging is a natural storage management device for time-sharing and also has a role in mapping information across storage levels.

Location *addressing* has been the rule in most storage devices though it is common to use programmed symbolic addressing for dealing with mass storage. While strict hardware symbollically addressed storage has not proved economically attractive for computers, the counterpart in addressing logic (hash-addressing to list-structured equivalence classes) might be advantageously incorporated into hardware so that the approximate mix of location and symbolic addressing techniques would be directly available to the program designer. Instances of the application of symbolic addressing are interpretation during program checkout, in "conventional mode" of communication, storage management functions, and addressing of secondary storage.

The coordinated handling of several *sequences* of intermixed program and data is a natural provision in the organization of a time-shared processor. From this viewpoint sequence counters and index registers become instances of a more general device. Present programming languages do not usually have facility for the expression of multi-sequence control and multi-terminal time sharing usually envisages the running of mixes of independent and individually single-sequence programs. The expressive power of programming languages would be increased by providing the needed sequencing adjuncts; and, the processing options thus enabled would allow for throughput improvements in time-shared processors.

The notion of *substitution* combines the familiar ideas of assignment statement and procedure declaration. A basic machine function distinguishes between a name and a datum in a program. The evaluation of a name combines the ideas of fetch and procedure call (and, incidentally, a generalization of call-by-name).

CONCLUSIONS

The most serious obstacles to a generally acceptable machine language are (1) the rigid formats for program and operands, (2) lack of provision for referencing hierarchical data structures, (3) limited choice of mappings, (4) the inability to express multisequence algorithms conveniently, (5) a too limited concept of substitution.

If the desirability of variable program and data format were accepted and design emphasis directed towards these goals, then some problems of a persistent and annoying nature would be alleviated or even eliminated. The direction indicated, rather than language-based processor design, would seem to offer the best opportunity for major improvement in information processors.

DYNAMIC STORAGE ALLOCATION

R. L. Cook

Elliott Automation Computers Limited

This paper describes SPAN—a dynamic storage allocation scheme that has been implemented on the Elliott 503 Computer. The system is based on code word technique described for example by Iliffe.[1]

Dynamic storage allocation is used here to mean the allocation of core storage for both data and program at run time, i.e., during the actual operation of a program.

The 503 is a medium-sized computer primaril used for scientific computation. As far as this pape is concerned it has four types of storage:

- main core store (8k words)
- auxiliary core store (up to 132k words)
- magnetic disc store
- magnetic tape store

The object of the SPAN system is to present t the user what appears to be a large single leve store–the ability for the user not to have to differ entiate between the two forms of core storage bein of particular importance. A secondary result of th system is to permit a program to be operated un changed on two different storage configurations.

Storage is manipulated in terms of variable block of consecutive words. Each block's current positio is completely defined by means of a single cod word. By creating a block of code words which i turn is defined by a single code word, a tree c blocks of any depth can be created. A block, onc created, is moved between the various storage de vices either automatically by the SPAN executiv system or, via specific macro commands issued b the user. The size of a block is dynamically variabl in size within the prescribed limits of 5 and 4,09 words. Any block may be fixed, permanently c temporarily, so that it is not manipulated by th system.

The system outlined above forms part of th central executive system of the 503 which is use whatever programming is in operation. The syste solves, in an obvious way, the problems of ove laying segments of program, of allocating buff areas for input and output and of allocating storag for arrays and tables. The three principle progra ming languages are an assembly language, ALGO and FORTRAN. In the case of the assembly cod program and data segmentation is in the hands (the programmer but in the case of a high level pr gramming language it is necessary to build into t compiler techniques for determining suitable se mentation points. For the ALGOL language t information contained within the block structu of the program assists in this task. All blocks an procedure bodies are compiled into distinct pr gram blocks. As each block is entered, storag blocks are found for the compiled program and f each of the associated data declarations containe in the block head. If the block had previously bee entered it is possible that the compiled program still in main store and has not been overwritten an the system can then avoid bringing down the pr gram again. On exit from a block, the associate storage areas are marked as being no long required.

We can see that this system avoids the use of t conventional dynamic run time stack by the creati of independent data areas. An interesting side