INTRODUCTION
The best approach to a large computing capability is to build your own multi-processor system, utilizing the most effective elements available from the leading hardware manufacturers. To understand this approach, it would be useful to describe the type of system which is to be built. It is evident that many of the system characteristics will be prescribed by the application, others will be purely a matter of personal choice. The essential characteristics are few in number and must not be confused with the non-essential characteristics. The remainder of this paper presents a set of do-it-yourself instructions for designing your own multi-processor system which stresses the essential characteristics. The intent being that the reader can follow through these instructions and design his own system for his own particular application. In this way, each reader can have a specific system in mind which he can evaluate critically. Although there would be individual differences in the systems under consideration by different individuals, the essential characteristics would be the same, and discussion would then tend to center about these essential features. The paper concludes with a summary in which the design approach is reviewed in an attempt to emphasize the utility of the approach.

The first step in designing the multi-processor system is to select a computer which is to be the basic module of computational capability. Select one of the many word-oriented full-word computers with a word length of at least 24 bits. Select a computer with multiple memory modules; one with some capability for a shared use of memory. Make the selection based on cost/effectiveness. Select the computer because it gives the most computer power for the money. Don't worry about channel capacity, total memory capacity, etc. These characteristics will be considered later. The criteria for evaluation of competing computers may be simple or complex. If you have no other criterion, you can follow the crowd. Some of the best performing computers in the past have also been the best sellers.

The next step is to select a small character-oriented (byte) computer with good channel characteristics to serve as the I/O, or peripheral, controller and processor. Select the computer so that the large computer's word length is a multiple of the small computer's word length. This is relatively easy for a character-oriented small machine. We will decide how many of each type of module to provide later.

Now, since we have selected a small general-purpose computer as the I/O controller, we will shop for peripherals that have a minimum of control logic in each unit. Simply look for good clean electromechanical design and select each peripheral from the most appropriate vendor. It is very unlikely that any one vendor can supply more than one or two types of modules for the system.

Estimate the total system computational capacity, based upon the particular requirements of your application; then select the number of large processors based upon this computational load. Do not include extra capacity for executive functions. The number of memory modules to be provided is based upon the performance you wish to provide to the different classes of users. In general, two memory modules per processor will provide overlapped swap and computer capabilities for a large class of applications. A relatively simple computation will provide estimates of the system performance with different numbers of extra (swap) memory modules. The computation will reflect your own mix of problems, the grade of service you hope to give each class of user, and the expected loading for each class of user. These performance estimates can be as simple as statistical estimates derived from drawing samples from a set of hats with the correct mix of alternatives in each hat, or as sophisticated as detailed simulations. Results derived from published queueing tables
should provide sufficient accuracy for determination of the size of the system memory. Next, determine the number of peripherals required for your application, and from this estimate how many small peripheral processors will be required to support your I/O functions. In addition, these small processors must also perform some executive functions, which should require no more than about 3/4 of the capacity of one of the small computers. This estimate depends upon your own particular application as well as upon the amount of I/O supported. If there are several I/O processors, the executive can be interleaved with I/O idle time so that little extra capacity is required. If your I/O requires most of one small computer’s capacity, the executive function might necessitate a separate processor.

The system components must now be interconnected by designing a high-speed communications network so that the memory modules are all accessible to all computers. This network may take the form of a multiple-path, high-speed switch between the processors and the memory modules. It should include special logic to facilitate the executive function. In particular, it should provide hardware for translation of symbolic memory addresses into actual cross-point locations on the switch. It should include hardware to facilitate queuing of processor assignments and should provide for interrupt routing to the appropriate processor.

In addition, a low-speed network should be provided for connecting the peripheral units to the I/O processors. This network might take the form of a multiple-path relay switch. It requires only very simple control logic since it will be controlled by the I/O processors, which execute only I/O and executive programs. Peripheral reconfiguration will seldom occur. I/O functions would normally be grouped, and assigned in groups to each of the I/O processors. In addition to the assigned I/O functions, any of the I/O processors could be preempted to perform an executive function, such as swapping a program segment in core with one on the backup storage (probably drum or disk). This low-speed network should be required only if graceful degradation is desired.

The final step is to write the executive program and I/O programs for the small I/O processor.

The full-word processors are to be used only for execution of object programs. In order to do this, we must consider the type of object programs which the users of the system will provide. Let us, therefore, define certain aspects of the object programs which are required by this system. The entire computational task, from initial input to final output, in support of a given user of the system is defined as a job. When each task is initiated, it is assigned a sequential job number by the system. If a user attempts to initialize a task which is already in execution in the system, the job number will permit the system to distinguish each use as a separate task. The job number is utilized to assure privacy between users, in that different jobs are not permitted to interact. Communication between jobs is permitted through use of system tables specifically designed for common use by two or more jobs. The system I/O routines also permit shared use of a peripheral storage medium, such as magnetic tape or disk packs, for communication between jobs. Each job is to be broken down into threads of computation. The maximum length of each thread is set by the core limitations of the memory module and the maximum execution time imposed by the executive system, so that, if the time limitation is exceeded, the job may be temporarily swapped out of core.

Threads of code are linked by a macro instruction CUT to X which initiates subsequent threads within the job. The system does not utilize re-entrant code. That is, only one processor will execute a thread of code at a time. System tables and subroutines are replicated in the environments of different threads of code. When a large computer and a memory module are associated in computation, there will be very few memory cycles available for other uses without slowing down the computations. For this reason, system tables can be designated as guaranteed, in which cases the tables will be restored to the system data base when the task is completed, thereby, updating automatically, the contents of the table in the system data base. System macros provided permit the program to designate parallel threads which could be executed concurrently. The programmer may use recursive coding techniques so that parallel sections of code may be nested. The executive system which includes both hardware and software, does not recognize a macro instruction for automatically synchronizing the tying back of parallel paths. This bookkeeping must be provided in the object code itself.

There is no limitation to the user of the system if the synchronizing code is generated by the compiler rather than by requiring the executive program to effect the synchronization. By limiting the executive functions to essentials, the basic system approach may be more easily understood. For this discussion, the non-essential software features, such as the compiler and diagnostic aids are not considered part of the system. Without carefully defining these non-essential software features, there would be a great deal of un-
certainty as to the overall cost of the software system. There are three system macros provided for programmer identification of parallel threads of computation. They are:

Cut — Enter a memory assignment in the queue waiting for a full-word processor to become available.

Wait — Release full-word processor for a new assignment, but retain this thread of computation and its environment (subroutines, tables, etc.) in core, if at all possible.

Release — Release full-word processor for a new assignment and release the program thread so that guaranteed data items can be updated in the system database, and a new program thread can be loaded by the I/O processors.

In addition to these system macros, there are three system queues: one for the full-word processors and the other two for the small I/O processors. The queues are:

- Full-Word Processor Queue — Computational threads which are loaded and ready for execution.
- Executive Queue — Calls for executive functions, primarily swapping of program threads between drum (or disk) and core.
- I/O Queue — Calls for I/O. This includes routine loading of program threads upon release of a memory module.

Both I/O and Executive calls are handled by the small I/O processors. However, the Executive calls take precedence over the I/O calls. These queues would be implemented in the high-speed communications network so that, under normal operating conditions, all processors would simply accept an assignment from the queue and proceed as far as possible with that assignment before accepting the next assignment from the queue. Priority logic complicates this simple mode of operation only by use of the interrupt feature to preempt certain computational assignments. The interrupt releases the processor and automatically inserts a cut into the appropriate queue to reinstate the interrupted task at a later time.

With this design philosophy, the executive and I/O functions can be developed for the small I/O processors. The system design is now complete.

Without specific applications in mind, cost estimates would be meaningless. The system has been described in such a way as to identify essential functions while simplifying interface considerations so that each element of the system may be costed independently. Having completed the design for your particular application, you should now be able to estimate the total system cost. For a large class of applications, this approach provides the best way of attaining large computational capability.

The system outlined has several interesting characteristics.

1. It provides a high-performance system since we have delineated specific capabilities, such as arithmetic, memory, and I/O, and optimized each independently from a cost/effectiveness point of view.

2. System overhead time is minimal since the central computers perform only job computations. The executive time is confined to a partial use of the character-oriented I/O processors.

3. System overhead hardware costs consisting of the high- and low-speed interconnection networks are clearly delineated.

I/O and Executive software development costs contribute to system overhead. If more manufacturers followed the practice of providing separate cost for hardware and software, a portion of the software costs would be offset by the hidden cost of whatever software operating system the manufacturer provided with his hardware. However, by applying currently emerging management techniques to software development, the system software can be produced at a lower cost.

Let us summarize by attempting to review the rationale behind this particular design philosophy. The use of peripherals with minimal control logic selected from many vendors will provide the least expensive user interface if the control functions stripped from the peripherals can be supplied at the same cost by the data system. The use of character-oriented small computers instead of special I/O controllers provides an improved level of support to the peripherals and their users at a cost lower than that of special-purpose controllers. The user's arithmetic support is provided by selecting the most economical word-oriented computer for that task. The central question then is whether the high-speed switch and its associated queues simplify the executive functions sufficiently to warrant the extra hardware costs.