A Workshop on Microprocessor Architecture and Systems was held on May 6-7, 1976, at Northwestern University under the sponsorship of the Technical Committee on Computer Architecture and the Technical Committee on Minis and Micros of the IEEE Computer Society. No formal theme was established, and no summary positions were presented by the attendees, but the following conclusions appeared to emerge:

1. Multiple microcomputer configurations will become more prominent, but the cost-effectiveness of these approaches will depend on the application.
2. Reliability and fault-tolerance are driving functions in many of these designs.
3. A redistribution of resources can lead to more powerful and efficient systems.
4. There is a need for an integrated, formalized design methodology.
5. Future systems will need more self-test/self-repair capabilities.
6. There is a need for better software development systems.

Session highlights

In a presentation on "MIMD Multimicroprocessors as Main Frame Replacements," Gary Tjaden of Sperry Research Center considered the replacement of a large, medium-capacity (approximately one million instructions per second) multiprogrammed, uniprocessor mainframe with a collection of microprocessors in a general-purpose, batch-oriented environment. The uniprocessor system was assumed to have a multiprogramming level of seven, an average user memory requirement of 128K bytes (for a total main memory size of 896K bytes), and a CPU utilization of 99.5%. Tjaden stated that the utilization figure was somewhat arbitrary and provided a measure to determine appropriate parameters for the multimicroprocessor (MIMD) system. In the multiprocessor system studied by Tjaden the collection of identical microprocessors shared a primary memory and a common I/O system. The model ignored considerations such as cost, contention, and overhead of the memory bus, the I/O bus, the operating system, etc., concentrating instead on high processor utilization (99.5%) within each of the microprocessors in the system destined to replace it. Multiprogramming in the MIMD configuration is accomplished by allowing the individual microprocessors to execute independent jobs. Upon completion of a job or upon encountering a wait condition due to an I/O operation a microprocessor is switched to another job. In order to maintain high utilization it was necessary to add a certain amount of memory for each microprocessor added to the system. A queuing model was developed to estimate this memory size increase. Using estimated costs for the 1979 time-frame, Tjaden observed that for a given level of expenditures (for a 1 MIP uniprocessor), normal user jobs, and non-custom software, it is more cost-effective to invest the financial resources to develop a small number of high-bandwidth microprocessors than a larger number of less effective microprocessors. In the limit this number approaches one, i.e., a high-bandwidth uniprocessor. This conclusion is also valid for timesharing environments and systems with virtual memory. The primary reason for this, observed Tjaden, is that the cost of the memory required to maintain the desired high utilization dominates system cost. He did suggest that multimicroprocessor systems may be more cost-effective than a uniprocessor in specialized environments in which the nature of the computational requirements is known beforehand. Responses by participants to Tjaden’s conclusions included the following: (1) If memory costs do indeed dominate, then perhaps memory efficiency should take precedence over processor efficiency. (2) Microprocessors should be considered as stages in a single-stream, pipelined CPU.

Dan Atkins of the University of Michigan spoke on “Instructional Use of Bit-Slice Architecture LSI.” The reasons for the use of such architec-
tures in an instructional environment, according to Atkins, were (1) they permit non-trivial design case studies with a small set of components; (2) they provide a vehicle for the study of microprogramming and emulation; and (3) they cover the understanding of MOS/LSI microprocessor components, thus providing more design choices and motivate understanding of RT-level design. Atkins discussed and compared the features of a number of bit-slice architectures including the Signetics 3000, Fairchild 9400, Advanced Micro Devices 2900, Monolithic Memories 5700/6700, TI SBS0400, and Motorola 10800.

In response to a question regarding the future of bit-slice architectures in view of the trend toward wider word-length devices, Atkins indicated that these architectures will predominate in special-purpose environments and in those applications requiring high performance. As an example of an environment which can efficiently use bit-slice architectures Atkins mentioned the area of process control in which bit-oriented computations is often required. A continuing problem and a limiting factor in the utilization of these architectures is the development of support software. This support has lagged the support that is available for fixed-length devices.

The next two speakers, Ken King of DEC and Jack Lipovski of the University of Florida, presented different approaches to a common objective: increased performance and computational efficiency through a reduction of unnecessary interaction between the components of a (micro) computer system. Both speakers also concentrated on a common place for accomplishing this improvement, namely the processor/memory interface.

Ken King, who spoke on “Distributed Function Architecture,” advocated the placement of a family of registers within the memory sub-system itself to facilitate address generation for both instructions and operands, and thus relieve the processor of these non-computational functions. These registers could be initialized by specific instructions in an enhanced instruction set. Such an approach would be particularly useful in a situation in which a stream of operands could be shipped to the processor itself. This involvement would be limited to the initialization of starting address and word-count registers in the memory. Such an approach, for example, would relieve the processor of fetching instructions whose only objective would be to increment an index register and generate an address for the next operand. King also proposed solutions to the handling of conditional jumps in such an environment. The implications of such an approach in a microprocessor system is that, as a result of a reduction in the required processor/memory control bandwidth, it is possible to get higher performance out of pin-limited chips by using serial flow between chips to set up the “streaming” environment. The use of a serial flow allows construction of simple networks of processor and memory chips. King observed that approaches of this kind are especially desirable in view of the continuing reduction in the chip cost/interconnection cost ratio. Continuing advances in chip complexity also suggest the feasibility of this approach to a distribution of system intelligence.

In his presentation, “On a Microprocessor Architecture for a Micronetwork,” Jack Lipovski stated that a slight redistribution of microcomputer resources can lead not only to faster microcomputers but also to microcomputer systems in which virtual memory, stack processing, multiprecision and vector arithmetic, and pipelined operations (primarily in terms of overlapped instruction execution and address generation) are obtained. He observed that servicing a page fault in a virtual memory system reduces to some overhead activities plus the transfer of a fixed number of words—a “packet” in Lipovski’s terminology. Lipovski noted that a single memory chip can be configured so that it contains exactly the information which constitutes a page. Two registers added to each of these chips could be used to implement a stack pointer, a bounds register, an address-counter, etc. He presented a scheme which permits chips being loaded with requested pages to be temporarily switched from the processor interface to the I/O interface, thus permitting the processor to access other active pages (chips) without interference from the paging process.

Lipovski observed that his approach to virtual memory implementation in microcomputer networks would (1) allow the sharing of expensive peripherals, (2) facilitate the implementation of large programs, (3) take advantage of packet-sized transfers of information, and (4) lead to lower memory costs.

The presentation closed with the following questions: Does a normal instruction set really consist of two instruction sets, one for handling data and one for handling instructions? And what is the interaction between busses and programs? In supplying his answers to these questions Lipovski further elaborated on his approach to a decentralization of system resources.

Geoff Leach of Sycor, Inc., spoke on “Experiences with the Development of a Microprocessor Language.” SYCLOPS (Sycor Language for Operating Systems) was developed for Sycor’s microprocessor based systems, the first of which is an interactive data entry and verification system, the Sycor 440. Leach outlined the reasons for Sycor’s decision to replace PL/M. From the corporate point of view the principal reason was to develop control over Sycor’s principal software tool. Economically, specialization promised improved efficiency. In a technical sense the development of a new language permitted the implementation of modern advances in language design (i.e., better compiler writing and compilation techniques) and usage (i.e., better programming practices). Sycor established the following language goals for SYCLOPS:

(1) complete control over maintenance and the generation of extensions;
(2) code efficiency should be as high as possible;
(3) the language should permit the development of future systems;
(4) the language should support the use of microprocessors other than the one used in the development of the first SYCLOPS-based system (program portability outside of Sycor was not considered as one of the language goals; and
(5) downward compatibility with PL/M.

Leach also discussed the resources required for compiler development in terms of personnel, analysis tools, and development languages. After relating some of the actual extensions to PL/M that are contained in SYCLOPS, he observed that language development can be a worthwhile undertaking for a small company. Development of SYCLOPS at Sycor required 2-2½ man-years using three people.

John Tartar spoke on “Application of Microcomputers in Control Environments.” Tartar, of the University of Alberta, described a distributed processor system as an implementation of well-defined algorithms with well-defined inputs and outputs migrating outwards from a centralized structure to individual units. Such an approach promotes reliability by assigning few responsibilities (perhaps only one) to each of the individual units. He viewed one of these individual
units as a microprocessor with at most 2K words of private memory.
Tartar cited a number of problem areas in this approach to real-time control. The first of these is the naive user who knows a lot about the problem but has little or no knowledge of microprocessor-based systems. Such a user, for example, may not know how to react to error conditions and may require extensive indoctrination in the use of the controller. A second problem area is that of implementing the algorithm in a cost-effective manner.

In response to a number of questions Tartar made the following observations:

1. Eight bits are satisfactory for most microprocessor-based control systems. Since the nature of the problem is well defined and since there is little need for floating point operations, normal fixed point operations generally are satisfactory. Sixteen-bit microprocessors will have a significant impact on these environments, however.

2. In process control environments many operations are of a logical (boolean) nature. It is expected that a microcomputer will be used to retain historical information, perform complex algorithms, and thus permit smoother projections.

3. Microprocessors are being used to replace minicomputer-based systems. Thus at the present time microprocessors cannot perform some of the more sophisticated operations (e.g., Kalman filtering) performed by minicomputers. Bit-slice architectures can be used for these sophisticated operations, but the trend is toward simpler rather than more complex nodes in distributed processor systems. It may be, however, that larger systems will be characterized by more complex nodes.

4. There is very little (if any) communication between the autonomous outer nodes. Most communication is between the central node and the outer nodes.

5. Most validity checking is in the form of the central node and the outer nodes checking each other periodically.

Tartar concluded with the following comments: (1) Future development systems should be more helpful to the non-computer practitioner. (2) More suitable, environment-oriented languages are needed, but the design of the language should not be left to the control system designer. (3) Reliability and suitability to the environment are the two primary requirements of microprocessor-based control systems. (4) Future chip design should reflect the unique requirements of various process control environments.

“Projections of Microcomputer Usage” was the title of the presentation by Bill Dejka of the Naval Electronics Laboratory Center. Dividing future microcomputer system designs into three areas, Dejka’s presentation concentrated on the first two of these: (1) classical designs consisting of a single computer, (2) dedicated designs consisting of from 1000 to 10,000 computers, and (3) brainlike designs requiring millions of computers.

According to Dejka classical designs will be found in areas such as consumer goods and small business systems. These mass-production-oriented designs will need to pay special attention to support problems and the development of self-diagnosing computers.

Dedicated designs will be characterized by the use of multiple microprocessor and bit-slice architectures and by very complex software problems, which will constitute a significant hidden system development cost. These dedicated designs, tailored to human use, will appear in a number of known high-usage application areas such as linear programming, regression analysis, file searching, and associative processing. These designs will be characterized by self-test/self-repair features and the use of sophisticated computing primitives (e.g., interface chips).

In Dejka’s opinion a very necessary occurrence before these complex designs become reality is the development of a methodology which can compare and evaluate various architectures. Such a methodology should consider more than performance and should include functional integrity, contention for shared resources (and the consequent creation of queues), reliability/maintainability, and life-cycle cost.

Another important consideration is serviceability and a reduction of the “throw-away syndrome” that has characterized recent technological developments in general. Serviceability considerations should include built-in diagnostics, stand-by redundancy, and a reduction of maintenance and the need for skilled technicians. Dejka foresees a need for n-dimensional architectures in which built-in testing is independent of function processing. He also stated that designers should get more information before finalizing a solution to a particular algorithm. Such a process, for example, should consider memory/hard-logic trade-offs.

Dejka summarized by stating that the two most important considerations for future designs are a microcomputer system design methodology and self-diagnosing computers.

The starting point of Bill Lennon’s presentation, “Integrated Design and Working Documentation,” was his observation that structural program design and top-down system design and implementation techniques have proven their worth in the computer systems design. The Northwestern University professor qualified this remark by stating that many of the benefits ascribed to these current design strategies are due in large measure to particular characteristics of individual high-level languages. Lennon then reported on the definition of an integrated design strategy which is relatively independent of both target computer and programming language. This strategy can be used at the present time by non-specialist users of microcomputers rather than at some future time when improved languages for microcomputers become available.

Recognizing the limitations of the human mind, the system designer constrains himself to problem and program descriptions composed of about half a dozen relatively independent units. Details in each unit are then similarly described. The modules thus described are documented within a comment block in a highly stylized fashion. Finally, after all of the stylized descriptions are completed, actual coding in assembly language is begun.

Several advantages accrue from this technique. First, trivial programs may be written to strip off the comment structures from the programs. Because of its convenient access, the “working documentation” keeps an accurate account of the program. Second, by explicitly describing the logic of the program in a pseudo-high-level language, the designer has effectively written a “top down” program without the psychological problems attendant to writing a program. Finally, as the individual modules are coded and combined into subsystems, the designer has the distinct psychological advantage of always putting together working subsystems and getting rapid reinforcement as goals are constantly being met.

Lennon discussed the details of this approach particularly in light of the needs of non-specialists who are currently performing system design in the burgeoning field of microcomputer applications with either existing or home-brewed tools.