1971

The structure and organization of communication processors

Richard Elmer Zimmerman
Iowa State University

The structure and organization of communication processors

by

Richard Elmer Zimmerman

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of
The Requirements for the Degree of
DOCTOR OF PHILOSOPHY

Major Subject: Electrical Engineering

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa

1971

TABLE OF CONTENTS

INTRODUCTION

Computer communications are defined to be those communications which are necessary for the flow of information between a host computer and its various terminals (1). In recent years, due to the availability of low cost data transmission techniques, wide spread computer communications have become economically viable. As the technology for data transmission has developed computer communication networks have been developed. This has led to communication-based computer systems, i.e., systems that utilize some type of carrier facilities to transmit or receive data.

The full computing power of a communication-based computer system would probably not be realizable without the development of time-shared computer systems. Time-sharing is defined to be the apparently simultaneous access to a computer system by a group of independent users (2). Time-sharing is feasible because of the immense speed differences between the human users and the computing system; each on-line user feels that the central machine is his alone. This feeling occurs as long as the response time, i.e., the time for the host machine to answer, is kept relatively small. The desire to keep response time small has led, in turn, to real-time computer systems. Real-time systems are those in which some particular response time requirement must be met by the system (2).

The juxtaposition of real-time, time-shared computer systems for use on-line using communication networks has led to a specialized processor known as a communication processor. These processors, usually small computers known as mini-computers, are responsible for supervising the communication networks. Smaller versions of these machines are, or can be, used at a remote site as intelligent terminals or as line concentrators.

This need for communication processors has arisen from two basic reasons. One; most large systems have in the past been designed for batch oriented processing. Thus the handling of many individual terminals could greatly overload the system. With the advent of machines oriented toward a multi-user environment this problem has not been totally alleviated. The demands upon a terminal oriented system are severe; more time can be spent receiving data than processing it (3). The heavy load comes about because each terminal generates many interrupts for each line of data transmitted. The communication processor can handle those interrupts and present data to the host machine in a clean format. This greatly reduces the load on the host machine.

In addition, more tasks can be assigned to a communication processor. These tasks might include such things as code conversion, editing and error detection/correction. This, in turn, reduces the host machine's work load.

As the need for communication processors has been increasing a second, parallel, development has been taking place. This is the development of large scale integrated circuit arrays. While some doubt exists as to exactly what constitutes large scale integration (LSI), a general definition is this: Arrays having a gate complexity greater than one hundred gates are referred to as LSI. For arrays with a complexity from ten to one hundred gates the term medium scale integration (MSI) has been adopted. Small scale integration refers to arrays with complexities of less than ten gates.

LSI has had, and will continue to have, a great impact in two areas that are of interest to a system architect. The first area is that of available functions where a function is defined to be some operation, e.g., addition. As it has become apparent that the architect can expect to accomplish more on a single chip of silicon a problem as to what functions were necessary has arisen. This problem has led to a great deal of research with regard to system architectures that could more fully utilize LSI. There have been two primary efforts in this area. One has been to maximize the gate to pin ratio, i.e., minimize the interconnections to the outside world. This approach recognizes that logic, in a large array, can be cheaper than interconnections by a significant amount. The second approach, which does not exclude the first, has been to minimize the number of unique parts in the system. This solution comes about because development costs are closely tied to the number of parts in the system; fewer parts mean, in general, lower costs.

A second area which has been affected by LSI is that of storage. Heretofore, magnetic storage devices have been the dominant storage technology. Magnetic storage devices have inherently complex electronic subsystems associated with the storage system. The need to share this complex electronic subsystem has led to a large storage module size. This has, in turn, yielded great economical benefits. Large scale integrated circuit storage elements, while still requiring some shared logic, have much of the decoding logic and all of the sense amplifiers and word drivers on a single chip of silicon. Thus, while larger semiconductor memories could and would be constructed, very small semiconductor memories can be distributed in a processor. This would not be practical with magnetic stores.

LSI will have a great impact upon the computer systems of the future. Because of this effect it behooves a system architect to fully study LSI techniques. With current semiconductor techniques MSI is much easier to produce than LSI. This research will propose a MSI logic family. This logic family will be a standard minimal number of parts type family rather than a custom set of chips. Unfortunately, because of limited facilities no attempt was made to actually fabricate this logic family.

This choice of MSI logic will be shown to have great ramifications upon the organization of a communication processor. The rest of this research will be a inquiry into efficient organizations and structures for a family of MSI communications processors which are intended solely for use in communication-based computing systems. Finally a small communication processor will be discussed. This small processor will utilize the ideas discussed within this paper. The results of evaluating this processor will also be discussed.

REVIEW OF LITERATURE

Communication Processors

Mini-computers are small, low priced, general purpose computers that can be used as dedicated systems. One of the primary uses of mini-computers has been as dedicated communication processors. These mini-computers have been programmed to reduce host machine overhead. Newport (4) points out that, in a large airlines reservation system or time-shared interactive system, it is very easy to swamp the host machine to the point that little basic computation can be done. It is precisely this large overhead that the communication-oriented mini-computer is intended to reduce.

The need for communication processors was recognized very early in the history of interactive computing systems. Strachey (5), however, first proposed time-sharing to reduce the complexity of terminals. Strachey felt that the large fast main machine could be used to control peripherals and to provide buffer storage for those peripherals. The impetus was, of course, economical. Buffer storage and control hardware were very expensive in 1959.

An early (1962) time-shared system described by Corbato et al. (6), was based on an IBM 7090 computer. As a part of the 7090 a direct data connection port was used to attach a "real-time equipment buffer and control rack" for controlling terminal devices. Thus, three years after Strachey's initial proposal, the buffer has been taken back out of the main machine. The idea of sharing this buffer storage and control was retained however. The real-time equipment was hardwired and any flexibility was contained in the system programs.

In 1965 Corbato and Vyssotsky (7) discussed the Multics System of MIT. The Multics System was based on a GE 645 Computing System. The Multics System utilized a "Generalized Input/Output Controller" for supervising the communication network of Multics. Ossanna et al. (8) described the Generalized Input/Output Controller. The GIOC was an nonprogrammable interface which was much more sophisticated than the earlier MIT buffer and control mentioned above. The flexibility of the GIOC was still the primary responsibility of the system software, however. The GIOC did perform such tasks as word assembly and disassembly, parity checking and generation, sequencing and control, status updating, and priority allocation in hardware.

Cohler and Rubinstein (9) suggested using a general purpose processor in a message switching system in 1964. The reasoning was that the wide varity of requirements in a message switching system dictated the use of a programmable message processor. To achieve high reliability, a multiprocessor scheme was implemented. This is often the case in message switching systems where loss of data is catastrophic failure. Cohler and Rubinstein suggested time-sharing the message switching processors for economical reasons.

Commercial specialized communication processors appeared at a relatively early date. In 1964 Daley, Scott, Drescher and Zito described the IBM 7740 and 7741 (10,11). This system was a programmable front-end processor specialized for the communications network.

To aid in the input/output problem, such as that mentioned by Strachey, computers have utilized a multiplexer type I/O scheme. The third generation machines such as the IBM 360, the Control Data 6600 and the Burroughs B5500 system all utilized some type of intelligent multiplexers for purposes of controlling local peripherals. These intelligent multiplexers were capable of transferring data to or from main memory without central processor interference by utilizing a storage cycle stealing ability. This increased greatly the number and types of peripherals the central computer could control. These third generation machines were still, for the most part, batch processing oriented. The real impact of interactive computing was to take place later.

The present "communication processor" is probably a general purpose mini-computer adapted to handle communication processing by virtue of special hardware and software packages. Examples of this application of mini-computers abound (1,3,4,12,13,14,15). Probably the best recent example that is well documented in the literature is that of Burner et al. (16) of Washington State University. In that application an Interdata Model 3 mini-computer was used as a programmable Data Concentrator for a IBM /360-67. The Interdata 3 was supplied with a special instruction to aid in mulitplexing and concentration. In addition, Burner et al. proposed a dual processor system which would use a Interdata Model 3 and a Interdata Model 4. The slower Model 3 would be used to multiplex 64 slow speed lines (~110 baud) while the Model 4 would handle the 360 interface, the Model 3, and four lines with data rates on the order of 2400 baud. The basic reason for this dual processor system was to distribute the load and to have enough processing power left to do such sophisticated tasks as syntax checking. The Models 3 and 4 would work out of a shared core memory. Thus the communication processor has not only been removed from the host machine but it has been increased by a factor of two.

As the need for programmable communication processors has become apparent, so has a need and desire for more economical communications. It is possible tc reduce communication cost by two methods; multiplexing and concentration. Multiplexing is the assignment of a channel's capacity on some, fixed, predetermined basis. Channel capacity is assigned to a terminal, for example, even if that terminal is not presently in use. Concentration is similar to multiplexing except that channel capacity is assigned on a demand basis; only those terminals which desire service are assigned channel capacity.

The multiplexers available are generally hardwired units that are intended solely for use as multiplexers. The use of these devices has been strictly for the purpose of increasing communication efficiencies. There has been, however, some effort at supplying more intelligent multiplexers which do more than simple multiplexing. The IBM 2905, which was described by Arnold (17), is an example. This idea was first proposed by Filipowsky and Scherer in 1961 (18).

The need for sophisticated terminals has expanded as interactive computing has become feasible. While the most common terminal is still the teletypewriter, the full graphical display terminal is becoming more and more commonplace. Licklider (19) first described the symbiosis, i.e., the union, possible between man and machine. Lewin (20) summarized the technology requirements for graphic terminals. Myer and Sutherland (21) described the general requirements for a display processor, i.e., the part of a display that is responsible for the control sequencing.

Mini-computers have been used as dedicated display processors (20,22). This use of mini-computers is especially desirable when the graphics terminal is located at a site which is remote from the host machine. This has led, in turn, to a sharing of the display processor by several displays (22). Thus it can be seen that time-sharing has an much expanded meaning over Strachey's original idea.

LSI and LSI Architectures

LSI proponents have long recognized two primary obstacles in the path to full utilization of LSI. The first problem is that present day computer structures are typified by highly irregular control structures. These irregular control structures have come about because system designers have, in the past, been very concerned about the sharing of logic functions. Thus computers have been partitioned by function. Even before the advent of LSI, however, it had become apparent that interconnections were contributing a significant portion of the cost of systems. In addition, LSI packaging techniques have limited the number of interconnections to a chip. This has led to a need for minimal interconnections.

Levy et al. (23) point out that conventional machines have achieved gate-to-pin ratios on the order of .85 to 1. Clearly some new approach must be used if arrays having complexities on the order of 1000 gates are to be built. Rice (24) suggests that logic is cheaper than interconnections.

The second major problem is that of a set of minimum number of parts. There must be some small set of LSI chips which will allow the system designer to configure his own system (25,26,27). The absolute minimum would be one chip. The computer-on-a-chip does not seem very practical presently. The computer-on-a-chip would be viable only in those cases where large production volumes are anticipated. The present day "computer-on-a-chip" tends to be relatively inflexible with regard to organization and structure.

Henle and Maley (28) identify four approaches to LSI. These approaches are the custom chip, the master-slice approach, the array chip and the functional unit. The custom chip is designed much as a printed circuit board. The logic diagram is transformed into a custom gate configuration; gate utilization is 100%.

In the master-slice approach, a chip is manufactured with some fixed set of gates. These gates are not interconnected however, i.e., the second level metalization is not specified. When the design is completed the second level metalization is applied; this determines the logic function of a chip. The master-slice approach leads to lower levels of integration than the custom chip (25). In addition, the utilization of gates is almost always less than 100%.

The array chip is defined by Henly and Maley (28) as a read-only memory. In this approach the logic designer specifies a truth table. The ROM approach is probably one of the cheapest approaches and is intrinsically very regular.

A more common definition of the array approach is that of Koczela and Wang (29). In this approach a machine is constructed using an array of many small processors. In this manner the cost of development for a chip can be amortized by using many identical chips in a system. The common chip resembles a mini-computer.

The functional unit is a standard circuit that performs some function such as, for example, arithmetic. The standard MSI functions available currently are examples of this approach.

Architectural approaches to effective utilization of LSI are varied. Beelitz et al. (30) proposed and constructed a functionally partitioned machine known as LIMAC. This machine used functional units with local control; the machine was microprogramed. While the control sector of each functional unit was identical the actual functional unit varied depending upon the tasks to be performed. This approach greatly increased the gate to pin ratio. The ratio improved by a factor of 2:1 to 7:1 over a standard approach depending upon the level of integration. The improvement of 7:1 was predicted for an integration level on the order of 1000 gates.

Rice (31) used an approach which can best be described as a bit-sliced universal register approach. In Rice's technique a "register" that is capable of many basic operations is programmed to perform standard operations. This technique has been attributed to Noyce by Atley (32), Avizienis and Tung (33) and Henle and Maley (28). In Rice's approach the bit width of the universal element is kept small so that a 16-pin dual-in-line package can be used. This results in a bit width of four bits or less. The universal register approach yields a very low number of parts. Noyce (32) states three different parts are enough; the universal register, a random access read-write store, and a read-only store. This coincides with Rice's predictions (31) except that Rice alludes to "a few discrete integrated circuits".

An area which has received much attention over the past two years is that of semiconductor or LSI memories. The LSI memory is said to present a challenge to the predominant main frame storage technology; the core memory. There are several architectual ramifications inherent in the use of LSI memories.

One of the most important differences between magnetic memories and semiconductor memories is that the size of an economical semiconductor module is an order of magnitude less than that of a magnetic memory (31, 34,35). It is now possible for a system designer to distribute small memories throughout the system. This concept can lead to systems which are radical departures from conventional machines.

Raisanen (34) discusses the advantages of distributed memory. This leads directly to logic-in-memory approaches such as that described by Kautz (36). Approaches such as associative processors could also become more feasible economically in the next decade because of semiconductor memories (34).

House and Henzel (37) have discussed the effect of LSI memories upon mini-computer designs. A key point of their argument is that LSI memory speeds are increasing much faster than are logic speeds. This, once again, forces a system designer to reconsider conventional machine designs. As memory speeds increase,more logic will have to be devoted to keeping the memory busy. House and Henzel feel that this will result in mini-computers with the same amount of logic but fewer architectural features than presently are available. This means that a more regular control structure will have to be found.

In 1967 Petritz (38) made some predictions based upon previous experience. His predictions have been proven essentially correct. Silicon chip sizes on the order of 200 square mils have been achieved. Memory cell sizes on the order of 20 square mils are now possible (39) for bipolar devices. Gate areas are on the order of 150 square mils for current mode logic or transistor-transistor logic. This means that complexities on the order of 200 gates per chip can be achieved with present day bipolar technology. Memory chip sizes for random access read-write memories seem to be limited to the order of 256 bits or so at present. These facts will be important in later considerations.

COMMUNICATION PROCESSOR ORGANIZATIONS

Computer organization, i.e., the data flow and the control logic, is affected by several factors. The task to be performed by the processor should influence the organization of a machine heavily. Because logic and fabrication are relatively inexpensive, it is possible to design and build economical special purpose processors to perform certain tasks. The standard "general purpose" arithmetic processor organization may not be a viable answer for a processor which serves a special purpose.

In addition to the tasks to be performed, physical realities play an important role in determing a system's organization. The packaging technique and logic family used can greatly affect the final product. So as to approach the problem of communication processing more fully, a medium scale integrated circuit family will be proposed. This logic family influences the organization of a processor, but does not limit a designer's flexibility. After describing the logic family the considerations which are important in communication processor organization will be discussed. A following section will discuss a MSI communication processor organization which utilizes the ideas set forth in this section.

A MSI Logic Family

The basic considerations and goals in designing a MSI logic family are discussed in this section along with pin count data and silicon area requirements. The detailed descriptions are down to the gate level, in most cases, but do not go further, i.e., the actual transistor configuration is not described. Basic area calculations are included to show that this logic family could be built by current or near future state-of-the-art.

High speed is desirable in a logic family to achieve high system performance while lower power consumption is necessary to minimize heating and power supply problems. Current, well developed technologies that meet the requirements of speed and power are current mode logic and complementary transistor logic. Both technologies, in addition, are capable of wired-or operation. This capability aids in the construction of a bus-oriented system. And/nand notation has been adopted for the standard gate of this family. This would necessitate the use of negative logic for most current mode logic families.

For purposes of area calculations, the following assumptions will be used. Memory cell sizes will be 20 mil^2 for lower speed cells and 30 mil^2 for higher speed cells. Gates will be assumed to have an area on the order of 150 mil^2 irregardless of the number of inputs. This, on the average, should give an approximate area required for the particular circuit. A chip size of 40,000 mil^2 will be adhered to in all cases. This establishes an upper limit on the order of 200 gates per chip for logic devices allowing 25% for chip interconnect pads. This is adequate and within the current state-of-the-art.

Medium Scale Integration (MSI) presents many of the same problems that are encountered in the design of processing systems utilizing large scale integration. These primary problems, which the system architect must fully consider, are minimum interconnections and a minimal number of unique parts. The solutions for these problems are many. All solutions, however, are in accord on one point; that the control logic in a processing system is highly irregular and it is this irregularity that is directly responsible for the high number of interconnections in conventional systems.

System designers have, in the past, attempted to minimize the amount of unique control logic by using centralized control schemes. The new approach to control expanded here could best be described as a distributed control. However, simply distributing the control logic would not of itself make a machine's control logic more regular. Thus a unified control scheme must be proposed that does reduce the amount of control logic.

A key to the effective use of distributed control is how instruction and timing information are to be distributed to the functional units of a machine. One method of distributing this information is the bus-oriented system. In a bus-oriented system all communication between functional units takes place on a single bus structure. This bus, while a single physical bus, can be subdivided into several logical busses each with a particular function to perform. This technique provides minimal interconnections. There are other advantages to using a bus-oriented system. One, the timing can be asynchronous between functional units. This means that, if a different hardware scheme for a functional unit is to be implemented, a simple replacement of the old hardware will suffice. Since instruction decoding and timing are local to a functional unit the rest of the machine is unaware of the change; the new functional unit performs the same tasks at, perhaps, a different rate. A second advantage of bus-oriented systems is that new functions can be added to the system with minimal effort.

An alternate method of reducing the amount of irregular control logic is microprogramming. In microprogramming the control sequence of an instruction execution is programmed. Thus the control sequence can be modified by changing the microprogram. Microinstructions are more basic than typical machine instructions and deal with actual gating and similar signals. This reduces the amount of unique control logic needed in the machine. Usually microprogram storage is a read-only store (ROS) because the microprogram does not need to modify itself.

The key points of the above discussion are:

1. A unified control scheme is necessary to reduce the amount of irregular control logic;
2. Bus-oriented systems reduce the number of interconnections, increase the flexibility of a system, and are capable of modularity;
3. Microprogramming reduces the amount of irregular control logic and increases the flexibility of a system usually at the expense of performance.

The proposed MSI logic family resembles emitter-coupled logic families both in speed and functions. Because of the need for a bus-oriented system devices capable of wired-or operation have been provided.

A prime consideration in the proposed logic family is the control technique to be used. The method of control desired here must:

1. Reduce the amount of irregular logic, i.e., the interconnections;
2. Result in a minimal number of unique parts;
3. Be adaptable;
4. Be capable of asynchronous operation on the functional unit level;
5. Be capable of being integrated on at least a medium scale level;

6. Provide functional unit timing.

These requirements result in some constraints upon the design of a control unit. One important constraint is that, if operation is to be asynchronous between functional units, some mode of signaling task completion is needed. If several functional units take part in an instruction execution, a simple "completed task" signal will not suffice. Secondly, if different instructions use the functional units in different sequences, then some mode of changing sequences must be provided. Note that this discussion assumes that one functional unit does not execute one instruction. This constraint raises the possibility of parallel operation of two or more functional units. This parallel operation mode should not be excluded by the control technique and will, in fact, be desirable.

A hypothetical processor is depicted in Figure 1. A control technique which meets the above listed requirements has been implemented. The basis for this control scheme is a control chip, a system clock, and a functional completion bus. Each functional unit constantly monitors the functional completion bus. When a particular functional completion code is found, that functional unit begins operation. Completion codes can be assigned to vary the sequence of operation. Since the bus is capable of wired-or operation, parallel functional unit operation can be achieved by requiring a code to be furnished in part by one unit and the rest by another. Functional unit timing is derived from the system clock while instruction distribution is via an instruction bus. The number of functional units is not limited in concept; the maximum number of 16 has been chosen in this particular implementation. If necessary, control chips can be cascaded to provide additional functional unit timing signals.

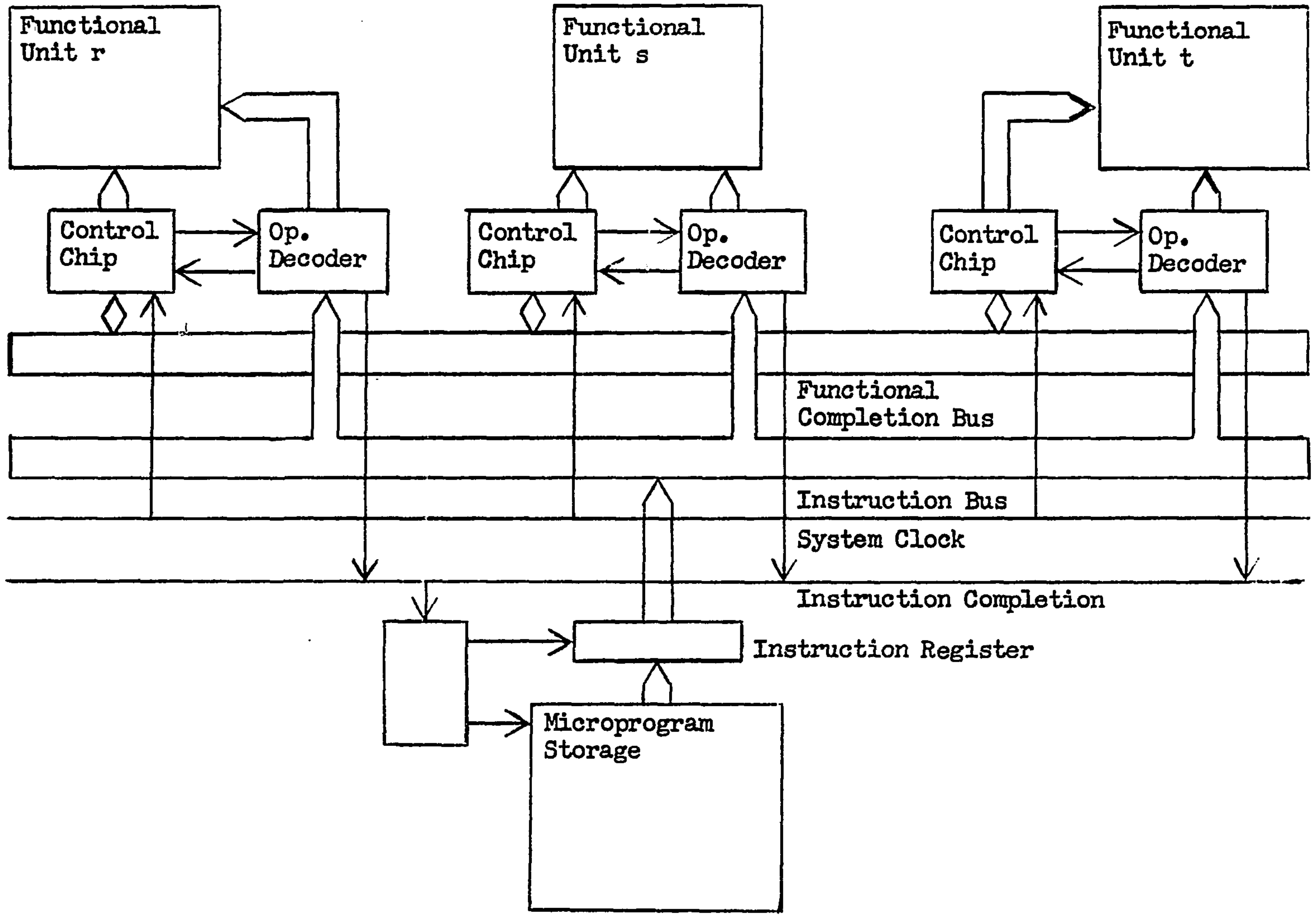


Figure 1. Hypothetical processor control structure

The logic for the control chip is shown in Figure 2. This chip consists of a three-bit up-counter, a one-out-of-eight decoder, and a four-bit latch with comparison circuitry. The up-counter and the one-out-of-eight decoder are used to generate local timing signals. When proper conditions exist,the counter is enabled and begins counting at a rate determined by the system clock. The various states are decoded and generate eight timing signals which last for one, and only one, clock period. When counter state 111 is detected,the clock line to the counter is disabled and the counter is reset. If more than eight local timing signals are desired the chips could easily be cascaded to generate timing signals in increments of eight. The disable line is also brought out so that the timing signal could be disabled by external logic.

The four-bit latch is used to hold a functional completion code. When the contents of the functional completion bus match the contents of the latch,a match signal is generated. This signal can be used to enable the clock and is also available externally. The latch is loaded from the functional completion bus at the start of an instruction.

The 16 completion codes should be adequate for most purposes. Should additional completion codes be required,two control chips and a single and gate would expand the number of unique completion codes to 256. Performing parallel operations reduces the number of available completion codes.

The control chip has a complexity on the order of 70 gates. This is well under the maximum permissable number. The number of pins, i.e., external connections, is 19 including the power supply connections.

Bipolar LSI stores have many characteristics that make them desirable for use in communication processors. Semiconductor stores have a small

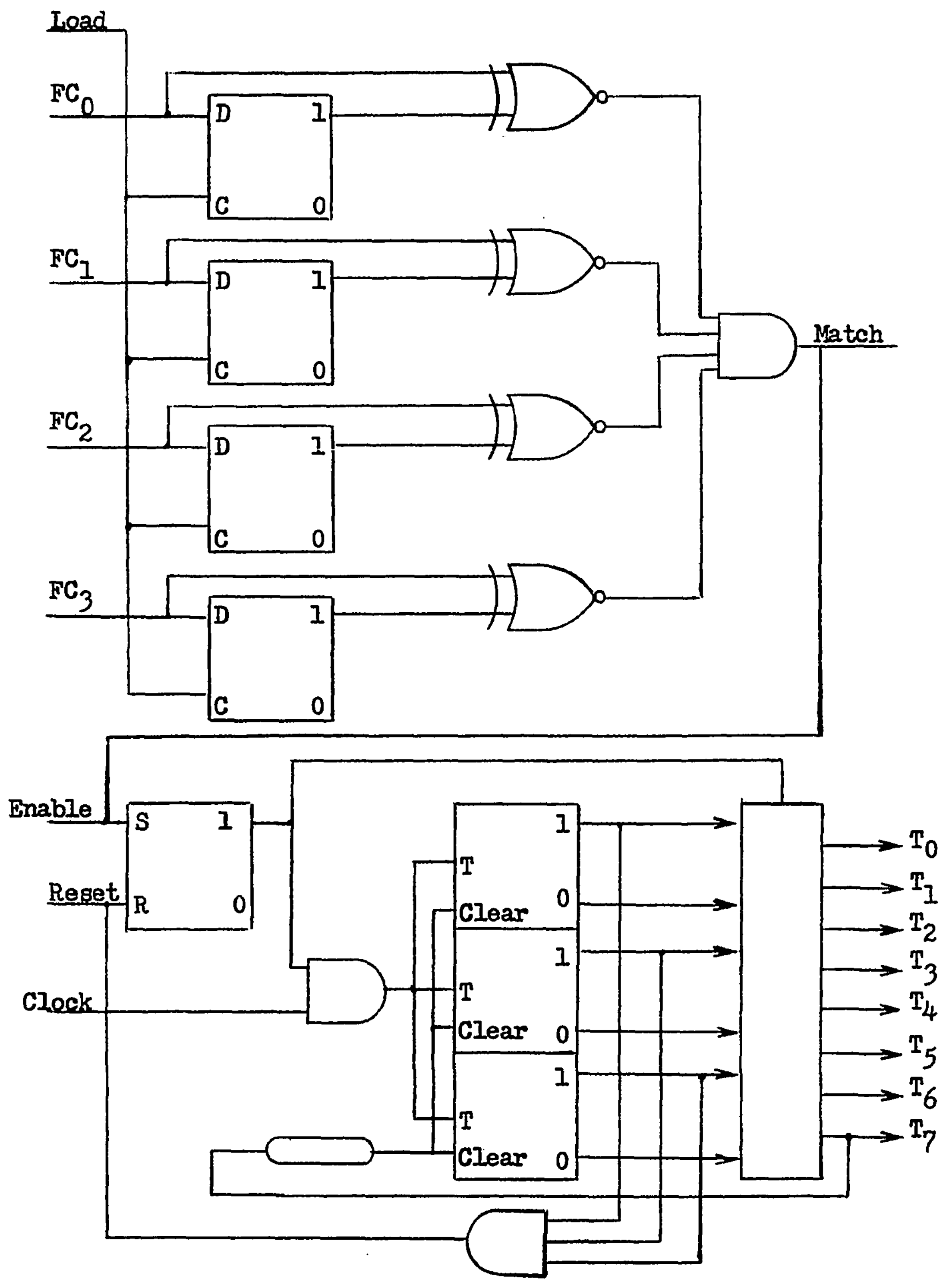


Figure 2. Control chip

economical module size that make it possible to have different types of storage provided. A second desirable feature of bipolar semiconductor stores is the very high performance capability; access times less than 100 ns are commonplace with LSI stores. A third advantage of LSI stores is that the mechanical assembly technology is identical to the assembly technology for logic.

One possible disadvantage of bipolar semiconductor stores is their volatility. This means that, if automatic system start-up is to be provided, some attention must be paid to the initial state of the LSI stores. One solution to this problem will be discussed later. These LSI store characteristics led to machine organizations which are different from conventional organizations.

To take full advantage of the characteristics of semiconductor stores, two basic memory building blocks have been included in the logic family. The first is a small, very high speed device. Access time is assumed to be less than 35 nanoseconds. The chip is organized as an eight character (character equals eight bits) array. The chip is fully decoded, and capable of wired-or operation. The second store chip is a larger, and somewhat slower device; access time is 75 ns with a cycle time of 100 ns to 125 ns. The 512-bit chip is organized as 512-one bit words. Addresses are fully decoded and wired-or operation is possible. Both storage building blocks will fit in a dual-in-line 24-pin package, however, the larger chip is packaged in a 16-pin package to conserve printed circuit 'real estate'.

Figure 3 depicts the small, high-speed memory package. The area requirements are:

2000 mil^2 - storage array

4000 mil^2 - decoding and word drive

2000 mil^2 - sensing and output logic.

This 8000 mils^2 is well within the 40,000 mil^2 allowed. The speed of the small memory is assumed to be in the order of 35 ns. This should be achieved easily using Schottky barrier bipolar technology. The pin connections are shown in Figure 3.

The eight character configuration of the small memory chip does not minimize chip interconnections. The advantages of having a complete character in a single package makes the trade off worthwhile. To minimize chip interconnections, the input-output pins are shared; this increases the complexity of the chip slightly.

The large storage element is depicted in Figure 4. Assuming a cell size of 20 mil^2, the 512-bit array area will be on the order of 10,000 mil^2. The remaining area is sufficient for address decoding and sensing. The next step could be an array of 1024 bits. This size array would take an area of 20,000 mil^2 which allows some 50% of the chip for interconnections, sensing, and address decoding. The 1024-bit chip should be achieved without great difficulty; the 512-bit size should be easily implemented using today's technology.

For arithmetical and logical operations, an arithmetic/logic unit (ALU) has been provided. This unit is similar to an universal register. The ALU is one character or eight bits wide and is capable of performing the following operations on two eight-bit operands: add, subtract, exclusive-or, equivalence, 'and', and 'or'. The ALU is easily controlled and gives

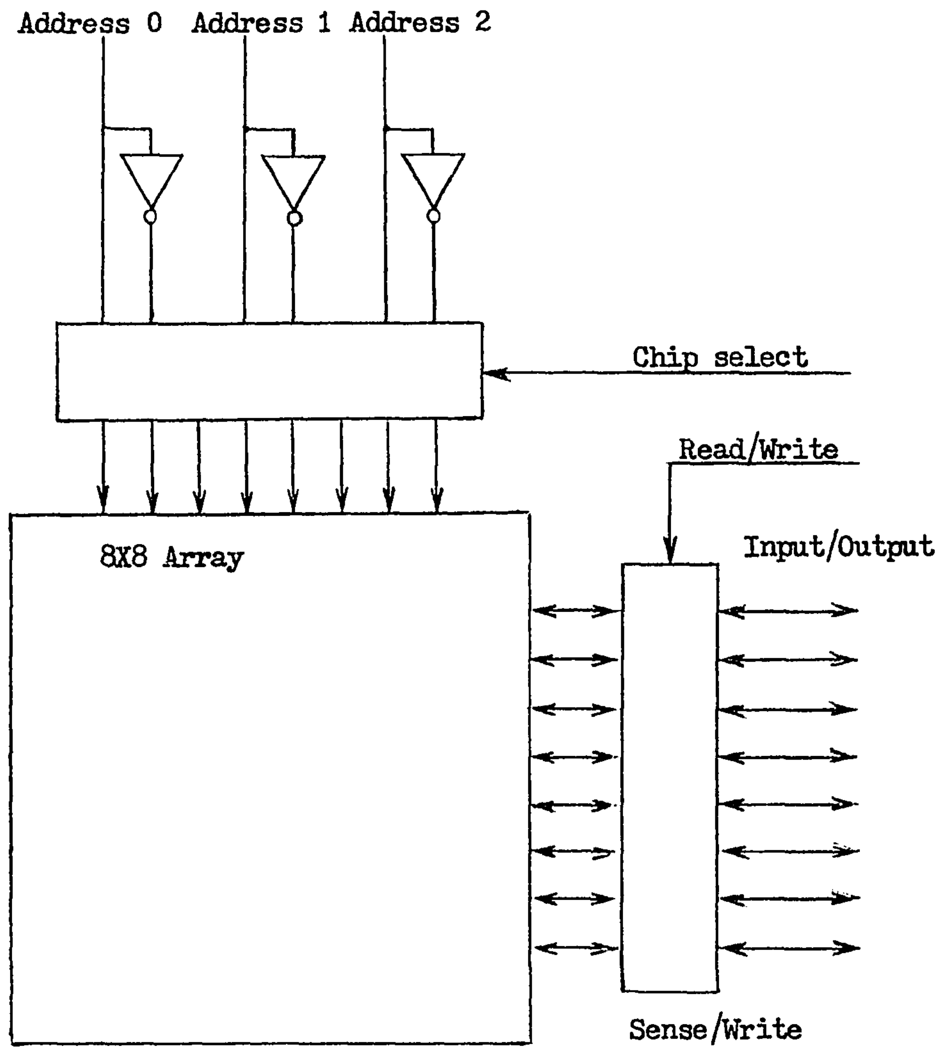


Figure 3. Small memory chip

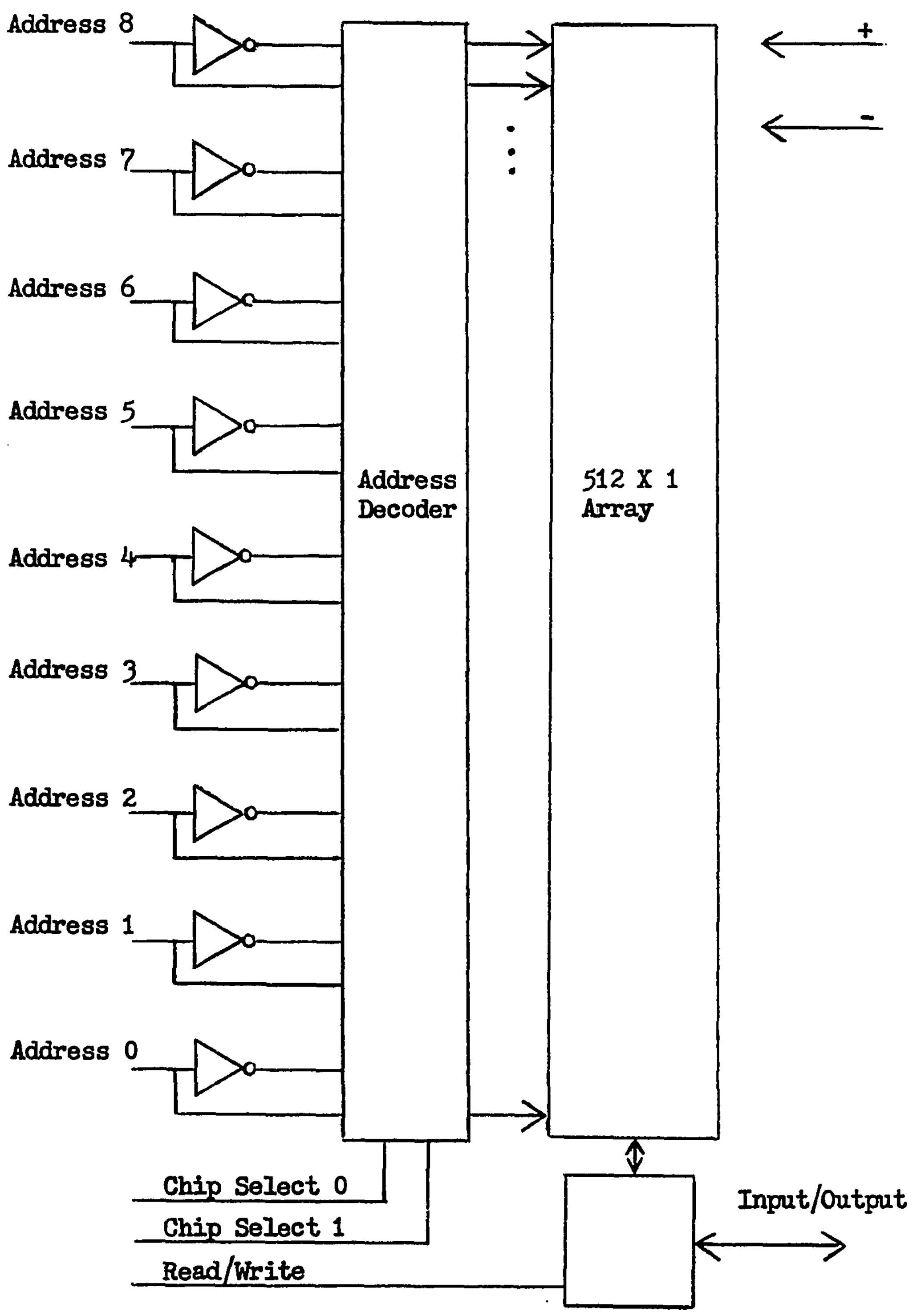


Figure 4. Large memory chip

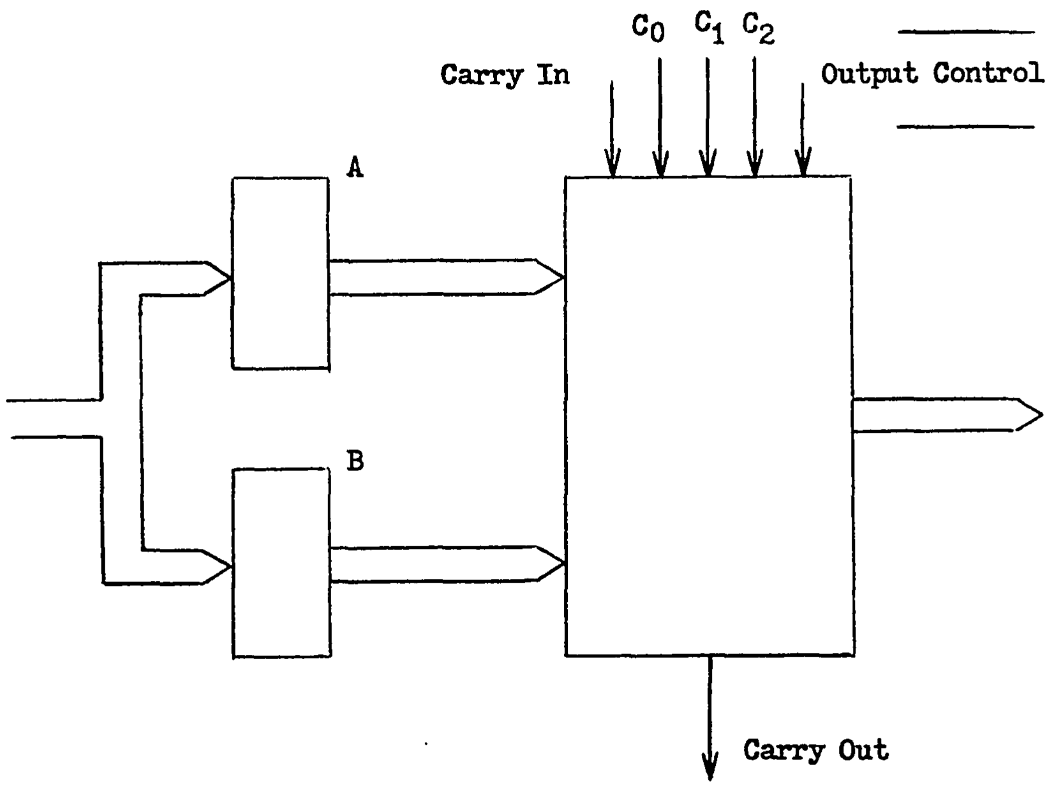
the designer a great deal of flexibility. This approach also results in a minimum number of unique parts while not making any one chip extremely complex.

One approach in 'universal' register designs has been to include shifting capability within the register. This approach has not been used here because of the desire to stay within a 24-pin limit and under the 200 gate limit. A completely general one-character shifter can be constructed using two bus switches and an and/nand package. To include this shifting capability within the ALU would increase the complexity of the chip unnecessarily.

The output of the ALU is under control of the fourth control pin and can be statically or dynamically connected to a bus. The type of connection is determined by the instruction requirements. This mode of operation gives another dimension of flexibility to the arithmetic/logic unit.

The arithmetic/logic unit block diagram is shown in Figure 5. The basic mode of operation is to load the two internal registers with the operands and then perform the desired operation, or operations, upon the contents of the registers. The six basic operations are sufficient to perform many processing tasks while not greatly increasing the complexity of the chip. The total gate count for the ALU is 180 gates which is near the maximum allowable. The pin connections are depicted in Figure 5.

A bus switch has been included in this logic family to facilitate a bus-oriented system. The bus switch is depicted in Figure 6. The capability to connect the complement of a eight-bit character has been included to aid in performing operations which utilize complements. This capability has also been included to limit the number of output pins from a latch



C_2	C_1	C_0	
0	0	0	Add
0	0	1	Subtract
0	1	0	Exclusive Or
0	1	1	Equivalence
1	0	0	And
1	0	1	Or
1	1	0	Load A
1	1	1	Load B

Figure 5. Arithmetic/Logic Unit

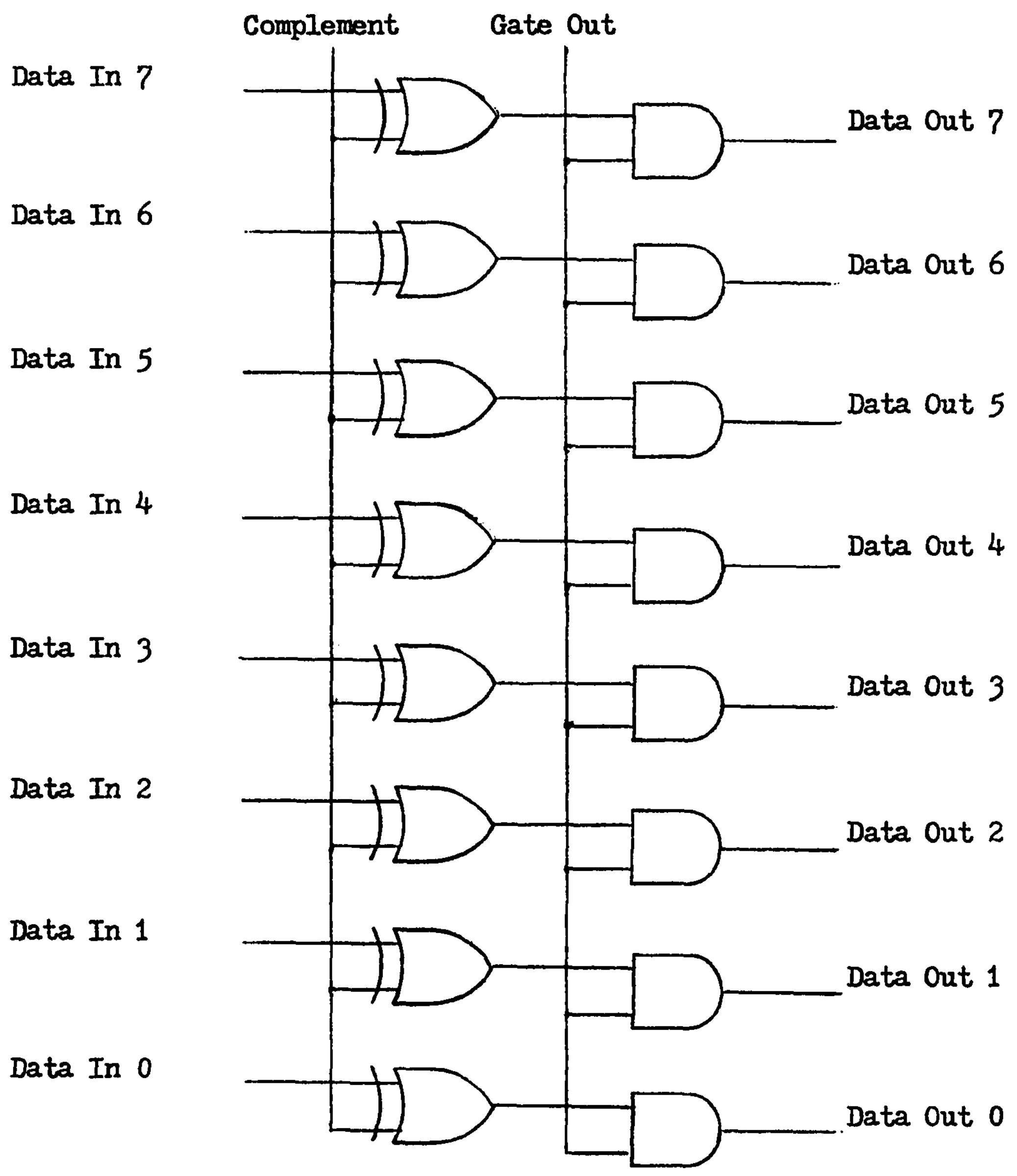


Figure 6. Bus switch

since the complements can be obtained using a bus switch. As pointed out above, the bus switch can also be used to generate logic functions, e.g., using it as for the basis of a shifter. The gate count of the bus switch is 24 gates while 18 pins are used. This is a very low level of complexity.

The one character latch is shown in Figure 7. This latch has been included in the logic family to allow for those functions where one character of storage is desired. A typical application might be an actual memory address register. The incrementing-decrementing capability could be achieved by using an ALU chip. The latch has a complexity of 24 gates and 19 pins.

For generating arbitrary logic functions and for additional storage and sequential logic functions, three additional elements have been included in the family. These elements are an and/nand element, with the number of inputs variable from one to eight, and a JK toggle flip-flop element for those instances where a local flip-flop is desired. These elements are only small scale integration; they are very desirable, however, because of the flexibility that comes from having these units available. Because of the SSI level, however, it would be desirable to adopt a master slice approach to these elements. In this approach some number of standard gates could be fabricated on the chip. The second level metalization could determine the gate configurations. This approach would greatly increase the integration level while not reducing the system designer's options.

The two small scale integrated circuits are shown in Figure 8. As discussed above, the and/nand circuits would be constructed using the master slice approach. The number of gates included upon a single chip

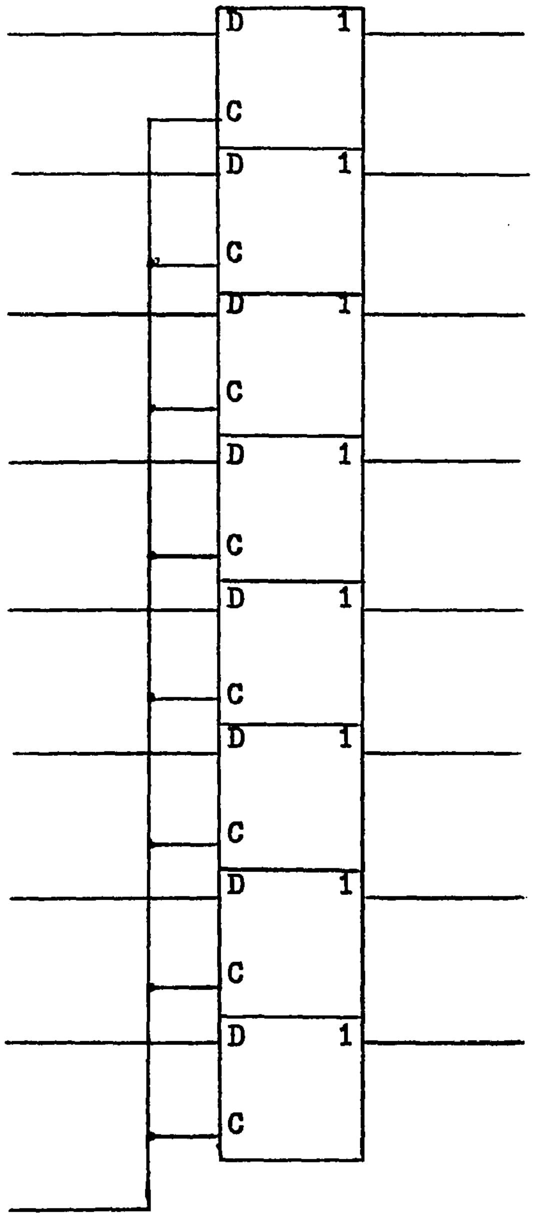
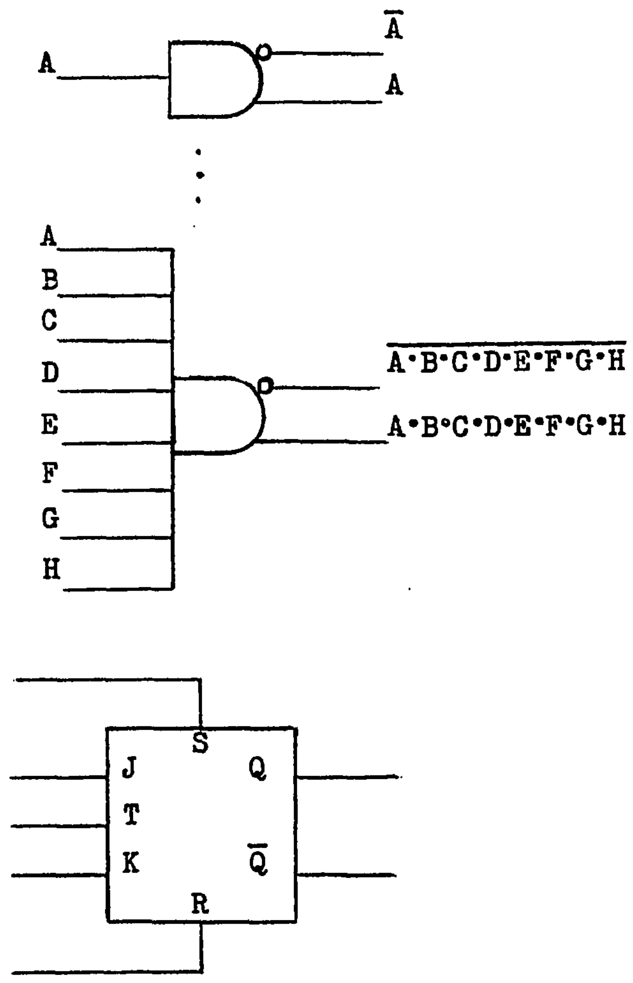


Figure 7. Latch


A
$\overline{A}$
A
A
B
C
D
E
F
G
H
$\overline{A \cdot B \cdot C \cdot D \cdot E \cdot F \cdot G \cdot H}$
$A \cdot B \cdot C \cdot D \cdot E \cdot F \cdot G \cdot H$
S
J
T
K
R
Q
$\overline{Q}$

t		t+
J	K	Q
0	0	Q(t)
0	1	0
1	0	1
1	1	$\overline{Q(t)}$

Figure 8. Small scale integration units

should be on the order of 30. This is enough to generate many logic functions while low enough so as to not result in high inefficiencies in gate usage on the chip.

The drive capability and speed of this logic family has not been specified. Typically, current-mode logic is capable of very high-speed operation in some circuit configurations, e.g., sub nanosecond. To keep power dissipation levels reasonable, the speed of this family should be in the two to three nanosecond range. This should be readily achievable with today's technology.

The fan-out properties of current-mode logic vary from family to family. While actual characteristics could not be determined without performing a detailed circuit design, fan outs from eight to 15 should be reasonable for this family.

To more fully exploit the power of microprogramming, and of this logic family, an electrically alterable read-only memory should be used for microprogram storage. The requirement of writability, coupled with a requirement for nonvolatile operation, effectively eliminates bipolar storage devices from consideration for use as microprogram storage. While plated-wire memories have many of the characteristics called for by this application, the module size necessary for economical operation of a plated wire memory is larger than desired in this case. The so called programmable read-only memories limit the firmware designer to one attempt per device; there is no method to reprogram the PROM. Current MNOS technology also has many of the desired characteristics; the performance is less than that desired, however. A microprogram store which does meet the above requirements is available and has been used in a test processor. This memory device is described later.

There are nine elements in this MSI family. This represents a small number of unique parts. The design of a system which would use this hypothetical family is discussed in a following section.

Organizational Considerations

The communication processor's primary function is input/output (I/O). The basic computational tasks for a communication processor are straight forward and will be discussed later. The I/O technique is of primary importance when considering communication processor organizations.

The I/O techniques used in a communication processor should have the goals of minimum interface hardware and maximum flexibility with respect to the number and type of terminals. Additionally, the communication processor is intended to remove some of the burden of supervising the communication network from the host machine. This has, in the past, been accomplished in part by increasing the complexity of the terminals, i.e., making the terminals more intelligent. This conflicts with the desire of users to have inexpensive terminals. The addition of a communication processor does not remove the conflict between terminals and central machine by itself, however. In effect, the communication processor now is a compromise between the terminal-oriented tasks and the host machine-oriented tasks. Most communication processors have, to date, resolved this conflict by using a general purpose mini-computer as a communication processor and building elaborate terminal device interfaces. These interfaces are inflexible, usually limiting the user to some number of more or less identical terminals, and expensive because, for example, all character assembly/disassembly and checking is done by the interface.

To arrive at a more satisfactory solution to the problem a two level processing capability could be used. The first level would be that level which is primarily concerned with the host machine and the larger computational tasks. The first level corresponds to a 'front-end' processor. The second-level processors would be more terminal-oriented than the first. This division of tasks could resolve the conflict between inexpensive terminals and host machine overhead while having many additional benefits.

A primary benefit of the dual communication processor would be that the second level processor could be used to provide concentration of slow terminals from remote sites. The second level processor could be constructed so as to greatly reduce the amount of interface hardware needed. This would result in a greater flexibility than is possible using a one level system.

If unique machines were to be developed and constructed for the first and second levels this solution could be expensive. The gain in performance might not offset the increased price. To make this two-level communication processor idea an economically viable one the first- and second-level processors must closely resemble one another in organization and structure. This means that the decision to microprogram and to have a bus-oriented system will facilitate this multiprocessor technique. The basic machine structure will be identical; additional functions could be included in the first-level processor only as needed. In fact, the second-level processor would be a stripped down version of the first-level processor.

I/O in communication processing is different from conventional computer I/O in that, in conventional computing, the computer initiates all I/O. That is, the central machine reads or writes data to from or to some device when it desires to do so. In communication processing the terminal devices initiates I/O in some random manner to the communication processor. This necessitates a somewhat different approach to I/O.

There are two basic techniques for communication processor I/O; polling and contention. In polling, each terminal device is tested to find out if that device presently needs service. This testing proceeds under the control of the communication processor. In this manner, some control of over terminal devices is established. In contention, terminal devices interrupt the communication processor when service is needed; terminals contend for processor time. Because it is desirable to mix terminal types and hence, transmission speeds, simple polling is usually less efficient than contention. This is because the polling rate, i.e., how often a terminal is tested, must be at least as fast as the fastest terminal's data rate. Thus a slow terminal would be tested many times over what would actually be necessary. In a contention system, however, the efficiency is high because terminals ask for service only when service is needed. Contention tends to be more expensive in terms of hardware than polling.

The basic considerations when using a polling scheme are that the testing time and the time around the polling loop should be kept as short as possible. In addition, a polling scheme should be as flexible as possible with respect to the number of terminals polled. The time around the loop is a function of the amount of service each device needs. This directly affects the structure of a communication processor and, indirectly, the organization.

The basic computational tasks of a communication processor are assembly/dissassembly of characters, queueing of data and control information, formatting of data, character searching, error detection and correction, code conversion, and real-time housekeeping. Note that maintenance of pointers, table look-up procedures, and character manipulation are the primary tasks. Arithmetic capability beyond that needed to perform those tasks is not necessary.

The division of tasks between the first and second levels has been discussed earlier. Those tasks which are terminal-oriented should be assigned to the second-level processor. A typical second-level processor task would include character assembly/disassembly. This function would, for example, remove the start/stop bits from a asynchronous character before sending the character on to the first-level processor. This would allow greater communication efficiencies. In synchronous transmission special characters would be detected, under program control, and proper action taken. These special characters would include End of Message and other similar characters. Also basic error detection tasks would be assigned to the second-level processors.

The primary tasks of the second-level processors would be to act as a programmable interface to terminal devices and to perform as a data concentrator. The second-level processor would be capable of remote operation and would result in higher communication efficiencies because of its ability to act as a concentrator. In a similar manner, terminal interfaces would be easier to implement resulting in savings for the user. The intelligent terminal is not excluded by this technique, however. The flexibility of the communication system should be such that user needs determine the type of terminal, not communication and host machine needs.

The second-level processor should be capable of modularity on a small incremental basis. That is, it should be relatively inexpensive for a single user to go on-line. If, for example, a module size of 16 interfaces is adopted as the smallest add-on, the price for 17 interfaces would be prohibitive. Thus a small increment of, say, one or two would be preferable. This should not make those instances where a large number of terminals are to be added more expensive, however.

The computational requirements upon the first-level processor are essentially the same as those of the second-level processor. Generally, those tasks which require larger amounts of memory should be performed in the first-level instead of the second. Code conversion is an example of a first-level processing task. The instruction set of the first-level processor will need to be more character-oriented. The bit handling capability that is desirable in the second will not be so desirable in the first level.

Because several second level processors could feed into a single first-level processor, a contention system will probably be necessary. The combined data rate could be very high and, unless the service times were very short, would overburden the first-level processor. In addition, it might be desirable to handle a few terminals which have very high data transfer rates e.g., drum memories. This points out, once again, that the flexibility achieved through the use of microprogramming and bus-orientation is highly desirable.

Both first and second level processors will need to facilitate the use of subroutines. The nature of communication processing is that many similar devices will be serviced with only a few parameters changing from device to device. Thus, if programs are written in terms of subroutines, the necessary storage space is reduced because the same subroutine with different parameters will suffice for many similar devices.

COMMUNICATION PROCESSOR STRUCTURES

The system characteristics of a processor which are visible to the programmer constitute the structure of the machine. Structure differs from organization in that organization is concerned with data flow paths and control techniques. Structure is concerned with how a programmer can use those organizational features to perform efficient processing.

Communication processors should make queue and list maintenance easier and minimize the hardware necessary to interface various terminals. The organization requirements imposed by these considerations has been pointed out. The structure considerations will now be discussed.

Structure Considerations

It is possible to use a general purpose machine as a communication processor. The instruction set of the standard mini-computer tends to be very heavily biased toward arithmetic operations, however. This approach results in device interfaces that are relatively complex. A primary concern of the communication processor architect should be to reduce the amount of logic necessary to interface a terminal device. This means, in general, that the instruction set should be capable of small, low level operations when dealing with a terminal device.

A second basic consideration of the system architect must be the amount of processing accomplished by his instructions. That is, are the instructions to be macroinstructions, accomplishing a large amount of processing, or are the instructions to be more micro, accomplishing a limited amount of processing? A basic factor in making this decision is the speed of the program store. If the program store is slow, then a more macro-instruction might be required to keep the speed of execution at a higher level. If, on the other hand, an extremely fast program store is available, the instruction set can be more basic with less frills. Instructions which are very large, i.e., accomplish much processing, tend to require more hardware. This conflicts with the desire to have an inexpensive and basic machine. Small instructions, on the other hand, require less hardware in the processor while storage needs might be increased. The increased storage is not a certainty.

The requirements of communication processing are complex. At one end of the communication system are the terminals; at the other the host machine. There is an overpowering need for flexibility in the instruction set of the common processor. The more basic instructions are more flexible in that more macroinstructions could be generated using these basic instructions. If very powerful macroinstructions are implemented in hardware, the flexibility of the machine could be limited because the complex macroinstruction could be ineffective at one end of the communication processing network. Hence, the instruction set should be basic, with small instructions.

As an example of this approach, take a shift instruction. A shift instruction might be capable of selecting some register and making n shifts with certain end connections. A more basic instruction set would have a shift instruction which would shift one place only. Thus, multiple shifts would need to be programmed.

The basic instruction approach minimizes the hardware necessary to do the job, reduces the length of the instruction, and is more flexible by virtue of the fact that the programmer, in effect, generates his own macroinstructions. This does not mean that the basic, arithmetic-oriented instruction set of the standard mini-computer is the best instruction set. The particular instruction set must facilitate those communication-oriented tasks described in the preceding sections.

An example of the type of instruction set being advocated is the instruction set of the actual communication processor which is described in Appendix A. No detailed description of an instruction set for the hypothetical processor is given because the instruction sets would be virtually identical. Examples of programs are given in Appendix B. A following section will describe how these instructions are to be used in communication processing as well as a general look at the structure of the particular machine.

A MSI COMMUNICATION PROCESSOR

The basic considerations in designing a communication-oriented processor have been discussed in the preceding sections. The details of an MSI logic family were also presented. To show how the assumed technology affects communication processor organization, the design of a hypothetical MSI communication processor will now be discussed. While this processor is hypothetical, it bears a close resemblance to an actual processor. The details of the organization of this actual processor are discussed in a following section. The conclusions of this thesis report on some of the results obtained from this actual processor. The rest of this section deals entirely with the hypothetical processor.

One primary consideration in the design of microprogrammed machines is that of microinstruction. The two primary approaches to microprogramming are referred to as horizontal and vertical. The first results in long microinstructions lengths. This is because, in horizontal microprogramming, each bit has some dedicated meaning. Thus one bit could, for example, control a gate signal. In vertical microprogramming the microinstruction is coded; a group of bits can carry several meanings, e.g., gate into one of eight registers. Vertical microprogramming results in shorter word lengths while more words of storage might be needed as well as more decoding logic on a per data bit basis.

Because of the desire to keep the instruction word relatively short, the decision was made to use either eight- or 16-bit instructions. The great majority of the instructions are eight bit instructions with some 16-bit instructions for those cases where additional information or parameters are necessary. The following section describes the instructions in more detail. Only the actual processor's instruction set is described.

The block diagram of a MSI communication processor is given in Figure 9. Basic processor communication is on two busses, the M or Memory bus and the FA or Flag Address bus. Both busses are eight-bits wide and bidirectional. The M-bus is the primary machine communication bus while the primary function of a FA-bus is to address the flag and data arrays. This addressing operation is described below.

The scratchpad store is to be used for holding data and parameters. Input/output queues and tables would be stored in the scratchpad as well as any other information which is to be altered. The scratchpad store would be constructed using the large memory chip of 512 bits. The scratchpad would be n words by eight bits where $n \leq 64K$. The store would have an access time on the order of 100 ns and a cycle time of 125 ns.

The scratchpad store is addressed by the address store. The address store would be constructed using two small, high speed chips in parallel to give eight 16-bit addresses. The first four locations of the address memory would be used as scratchpad store address registers. These four address registers are subdivided into two parts, a word register and a page register. These registers are accessible by program control as eight-bit registers.

The reason for this type of scratchpad addressing are twofold. First, in the preceding section it was pointed out that a basic need of communication processing is the ability to process and maintain strings of characters and queues. To facilitate this processing capability, the processor has been given four storage-address registers. Thus, pointer maintenance

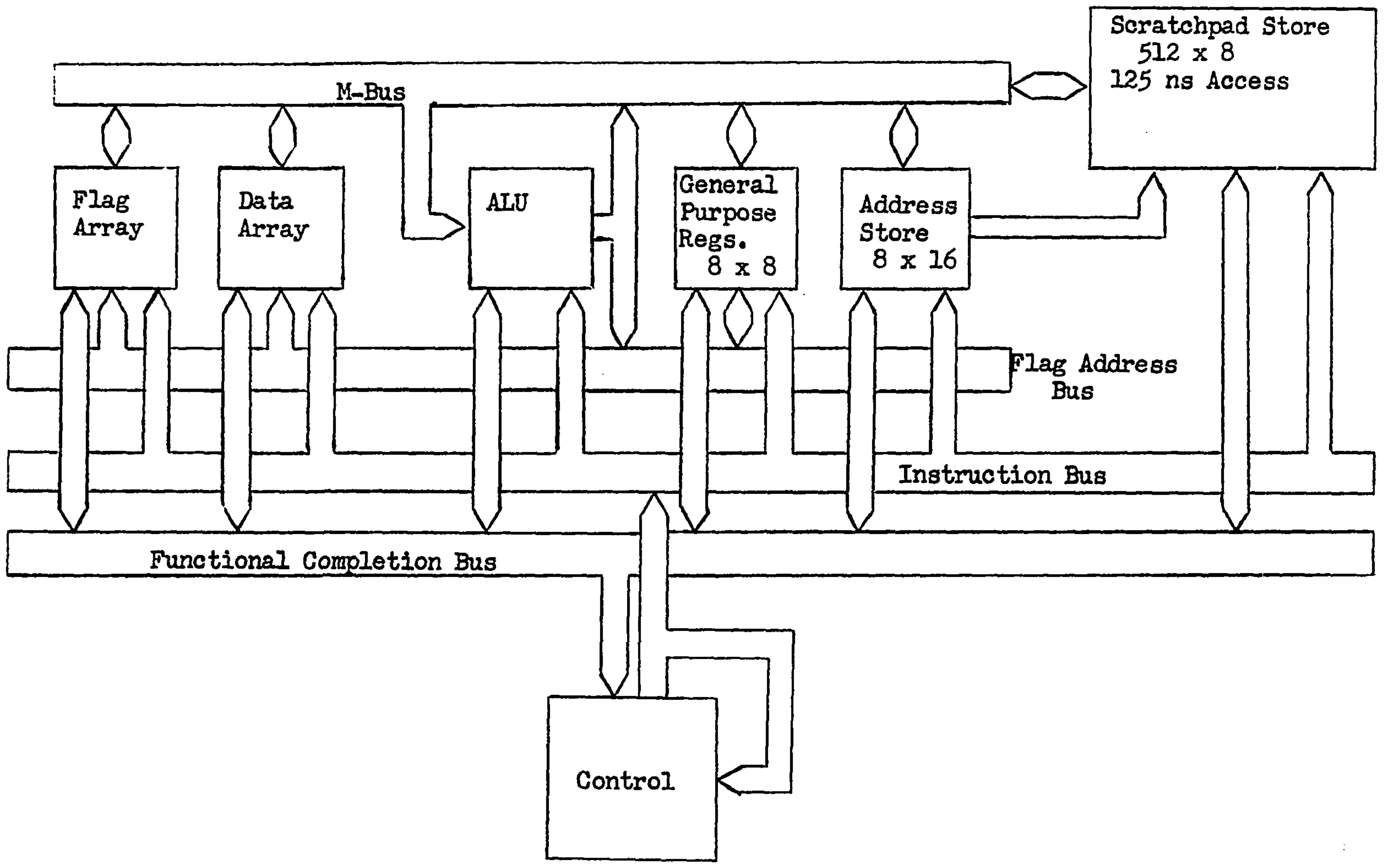


Figure 9. MSI processor

would be easier because pointers, queue addresses and similar addresses could be saved and manipulated. The single address register concept would be very limiting because of the constant movement of data and control information from one queue to another, for example.

The second reason for this mode of storage addressing relates to the desire to have relatively short instructions. The use of general address registers provides an element of indirection; the instruction need only specify the address register to be used and not the address itself.

A benefit which is derived from this approach is that the requirements upon the program memory are relaxed. If an address were to be contained in an instruction, then either a dynamically writable program memory would be required, or a method would have to be devised by which address information could be modified after being read out of a read-only store. The organization described above does not favor either a read-only store or a writable program store.

The high order four locations of address memory are to be used as a push-down stack. The hardware push-down stack would greatly facilitate subroutine linkage; the programmer need not worry about storing a return address. As pointed out in the preceding section,subroutines form an important part of communication processing.

The access time of the address memory is 35 ns. Note that, because the high speed memory device is used, very little speed penalty is paid for this type of organization.

The basic arithmetic/logic capability of the processor is furnished by the arithmetic/logic unit. The ALU would be constructed using the basic ALU chip; this ALU would be shared by the entire machine. For example, the basic instructions, e.g., add, would use the ALU. The scratchpad address registers would also use the ALU for incrementing and decrementing. This sharing of logic would keep the hardware cost at minimal levels. The ALU is capable of very high speed operation, on the order of 20 ns, and would perform well under such conditions.

To provide temporary high-speed storage, a file of general-purpose registers has been provided. These 'registers' would be constructed using a small, high-speed memory chip and would be used for holding operands and data. In addition to the above purpose, general-purpose register seven (GP7) would be used as a special purpose flag address register. This function is explained below.

To minimize hardware, a polling scheme has been adapted for this processor. The polling scheme adapted functions as follows. The flag array and data array are shown in more detail in Figure 10. There is a one-bit flag flip-flop associated with each terminal device. In a corresponding manner each terminal has a data register associated with it. The data register can be from one to eight bits long, depending upon the characteristics of the terminal device.

Typically, polling would proceed testing by a single flag bit at a time. Because of the desire to increase the efficiency of the polling scheme, and to allow more freedom in terminal speeds, a group-polling method has been adapted. A flag register is made up of individual flag flip-flops; the maximum length is eight bits. During a poll an entire flag register can be tested at once. If any bit is true, then further testing is done to find out which device within that flag register needs servicing. When the service routine is completed, polling proceeds with the next flag register.

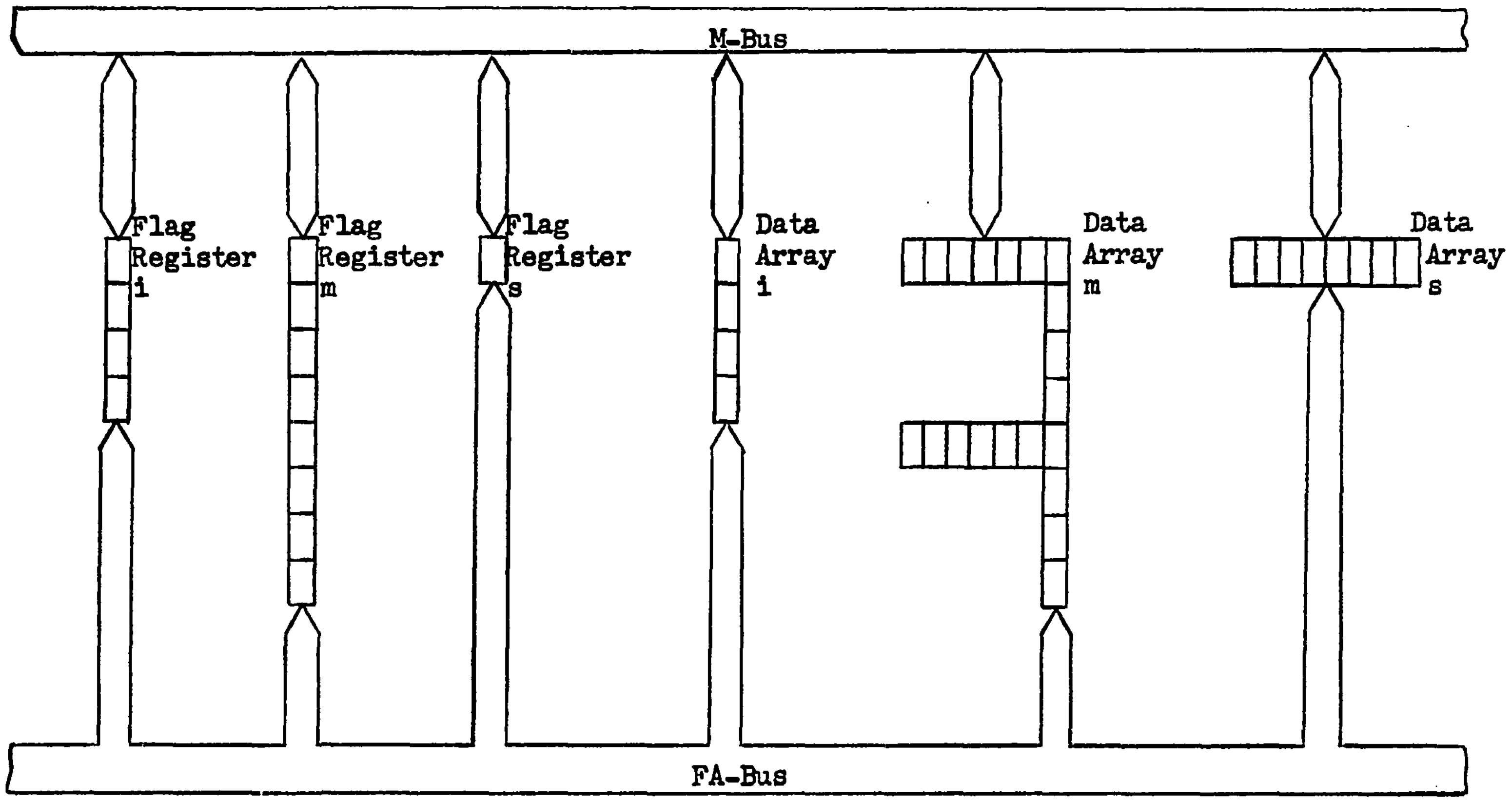


Figure 10. Flag and data section

The flag registers are addressed via the flag address bus as are the data registers. General-purpose register seven is used as a flag address register and also as a general usage register. This gives the system an additional dimension of flexibility.

Flag registers can be from one to eight bits in length. This allows a kind of priority structure within the machine. A high priority device, e.g., a high speed CRT, would be assigned to a one-bit flag register. Slow speed devices would be grouped into a single flag register of eight bits or less, e.g., eight teletypewriters. This could give the CRT a priority over the teletypewriters. The actual processing technique to be used would determine the priority. The organization permits the use of this kind of technique. The number of flag registers has been limited to 16. This would allow up to 128 terminals. Other considerations would probably limit the number of terminals to a somewhat smaller number.

Figure 10 depicts these flag registers (FR) with the associated data registers. Flag register i has four terminal devices represented; all four devices are serial devices. Flag register m has eight devices; two of the devices are seven bit parallel devices while the remaining six are serial devices. The remaining flag register, s, has but one terminal device. The data array in this case consists of a single eight bit register.

This organization treats each input/output port of the communication processor in an identical manner. While some ports might have a priority over other ports in a particular polling scheme,the organization is identical. This eliminates special channels from the communication processor into a host machine and gives the communication processor a greater flexibility. Any port can communicate with any other port. The only limitations are buffer size (speed) and protection considerations.

The control section of the processor is depicted in Figure 11. The basic elements of the control section are the program store, the program store address register (PSAR), and the control logic chips. The control scheme to be implemented is identical to the MSI control technique.

The dual requirements of very high speed and electrically alterable operation pose a complex problem. As was pointed out in a preceding section, many current EAROM technologies are limiting either in alterability or speed characteristics. One technology which fits very nicely into the communication processor niche is the OVONIC memory cell. The OVONIC cell is basically a diode resistor type memory. Hence, the speed capabilities of the cell seem to be limited only by the diode storage time. In addition, the OVONIC device can be written in a straight forward manner. At present OVONIC devices exist which can be written many times; individual devices have an assured lifetime of 100,000 writing operations.

The communication processing task is such that, once a given terminal-machine configuration is running, there are few changes on a day-to-day basis; the program set does not change. In addition, as was pointed out in a preceding section, it is very desirable to have a microprogram store which can not modify itself. Self modification can lead to program errors which would be especially disastrous in a remote, unattended processor. The OVONIC memory technology lends itself quiet nicely to communication processing. The cell is nonvolatile, fast, and writable.

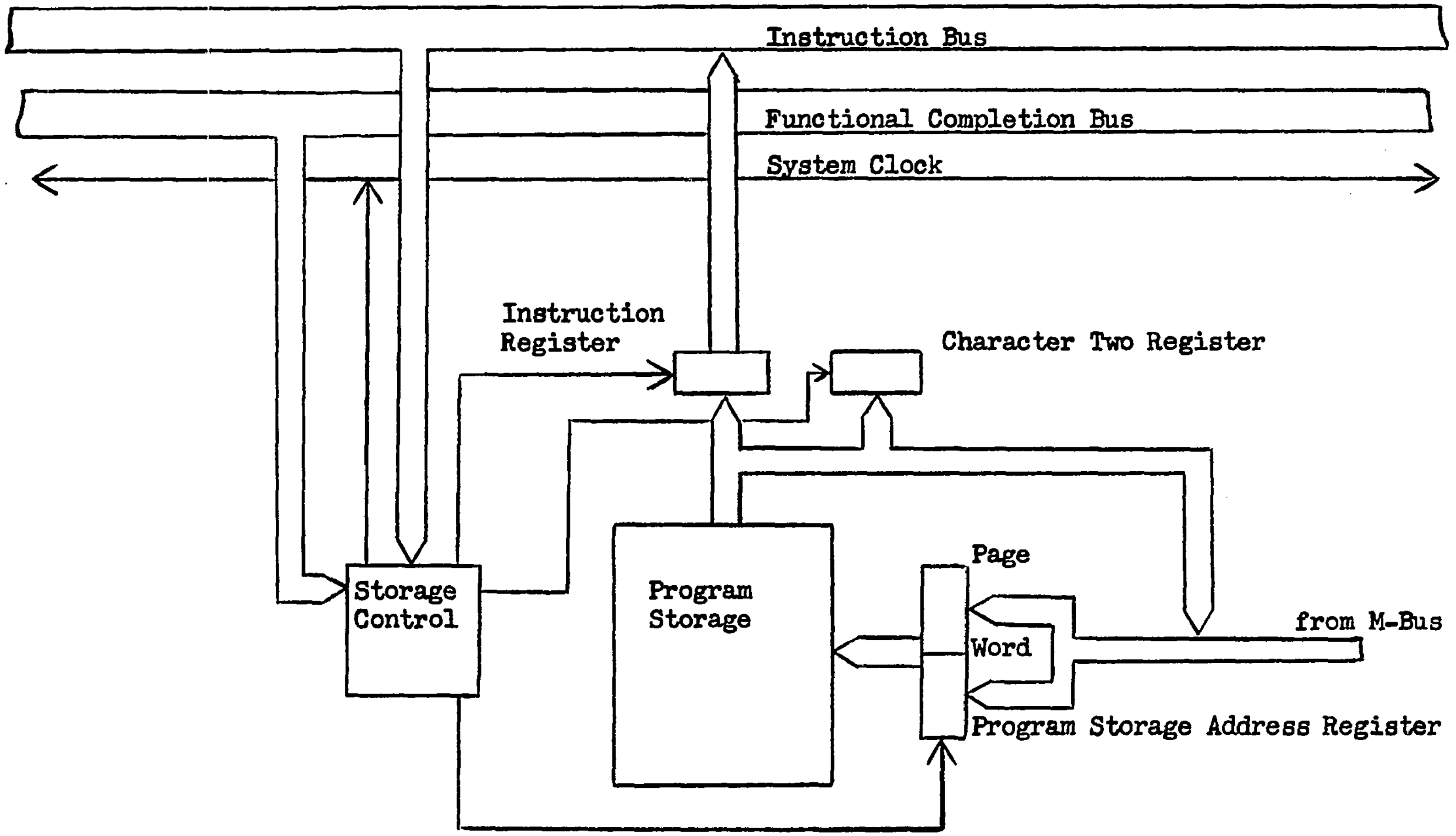


Figure 11. MSI control section

An instruction control sequence would proceed thusly:

Step	Action
0	Access program memory/load instruction register
1	Decode instruction
2	Set functional completion codes
3	Start execution
.	
.	
.	
n	Signal instruction completion.

There are two methods to set functional completion codes. The first would be to carry completion code information in the program memory. This would mean that either instructions for loading functional completion codes would be included in the instruction set or that a single instruction would carry functional completion codes for all functional units. The second alternative would be to set functional completion codes with combinational, hardwired logic. The second alternative gives higher performance levels than the first and would be used. The flexibility of the second approach is less than the first but the higher performance and shorter memory word of the second approach are the primary considerations here.

The functional completion bus is four bits wide. As can be seen from the system block diagram (Figure 9) there are six functional units. Thus 16 functional completion codes are adequate.

The PSAR is subdivided on a page, word basis. Incrementing is automatic and is continous across page boundaries. Note that the PSAR could be constructed using two ALU chips and two eight-bit latches. This demonstrates the generality of the ALU.

The decode/timing unit controls the program memory. This unit would also provide the system clock signal to the system. The system clock rate would be determined by the program store access time as well as functional unit requirements.

There is, in addition to the eight-bit instruction register, a character two register. This register would be used to hold the second character of a two-character instruction. This mode of operation simplifies the design of the local instruction decoders somewhat. The contents of the character two register are distributed on an eight-bit bus which is not shown for simplicity.

There is a data path from the program store to the PSAR. This is to facilitate jumps and similar operations. In addition, a data path from the M-bus has been provided so as to facilitate macroinstruction execution.

The program store can be expanded to 256 pages. Each page contains 256 words. The amount of program storage needed should be considerably less than the maximum amount, however.

In a preceding section, mention was made of the problem of scratchpad volatility. To achieve automatic start up along with fast operation the following solution to the problem of scratchpad volatility will be used.

Certain information would be stored, on a more or less permanent basis, in the program store. This information, upon a cold start, or restart, would be loaded into the scratchpad store. Thus parameters and similar information could be retained even though a power outage. When a system clear is executed, a small program would move information from the program store to the scratchpad store. This would permit the programmer to always use the high-speed scratchpad as his working storage. After execution of the initialization program,the machine could proceed on a polling loop.

The organization that has been presented is somewhat biased toward the requirements of the second-level processors. As pointed out previously, the general requirements of the first level processor are very similar to those of the second level processor. The organization presented here is flexible enough to fit in either slot. The possibility exists that a machine that is limited to strictly polling might be overburdened in the first level position. Thus the use of a bus-oriented organization is very important. Data paths have been provided so that an interrupt structure could be easily added to this machine. For example, the data path from the M-bus to the PSAR could be used to implement hardware interrupt handling. To be sure, additional hardware and programming effort would be necessary. It is significant that this capability could be added without a major redesign.

A TERMINAL PROCESSOR

A small terminal processor which closely resembles the hypothetical processor discussed in the preceding sections has been constructed. This terminal processor is intended for use as a second-level processor. The organization of this machine is very similar to the hypothetical processor as is the structure. Appendix A describes the instruction set of the terminal processor in detail.

A block diagram of the terminal processor is given in Figure 12. Instructions are distributed to functional units by the instruction bus. Both the instruction and its complement are distributed; instructions are decoded locally. The instruction bus is 16 bits wide; eight bits and their complements. Functional unit timing signals are generated from a two-phase system clock. Each printed circuit board has a local clock which can generate up to 16 timing phases. These 16 phases are distributed to the functional units on that printed circuit board. The 16 local timing signals are generated using an eight-bit shift register. An instruction completion signal (TOS) restarts the timing shift registers at time zero (TO). In this manner the instruction execution time can be varied.

The functional completion bus was not implemented in the terminal processor. The idea of using a functional completion bus was partially a result of early work done on the actual processor. Unfortunately, the idea was developed to late for inclusion in the actual machine.

The terminal processor has three data busses; the Memory bus (M-bus), the Flag Address bus (A-bus) and the Flag Data bus (F-bus). These busses are bidirectional, carry true data, and are eight bits wide. The M-bus is the primary system communication bus. The A- and the F-busses are used to

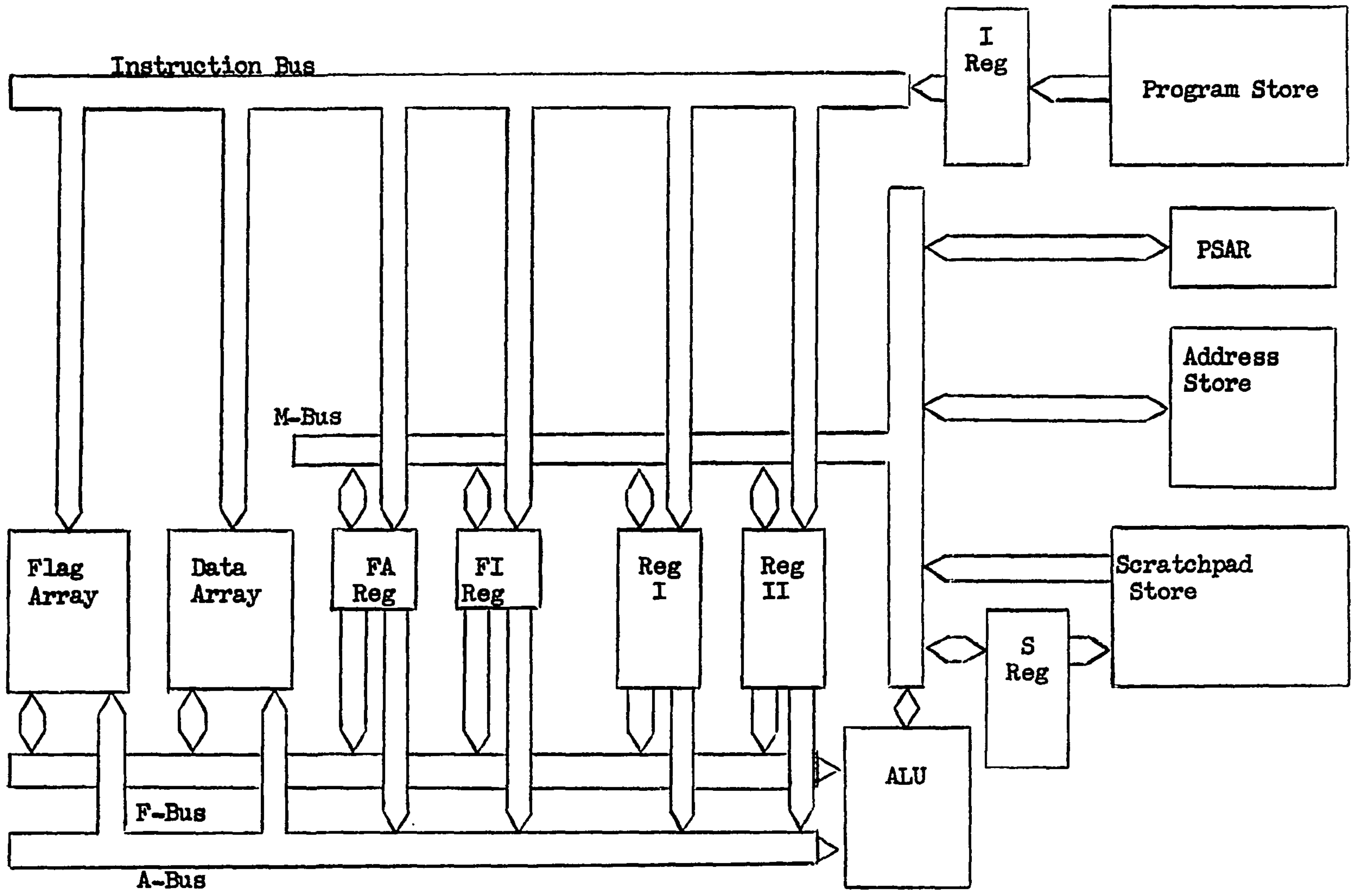


Figure 12. Terminal processor

move data and control information to and from the Flag/Data section of the processor. In addition, the A and F-busses are the operand busses for the ALU.

The three-bus organization allowed the ALU's registers to be used as general working registers in the actual machine. This minimized the logic somewhat. The ALU of the hypothetical machine contains two operand registers, hence a two-bus organization was sufficient.

Program storage for the terminal processor is in the Program Store (PS). The PS has been implemented using an OVONIC read-mostly memory. The basic memory chip is a 256-bit unit which consists of a 16 x 16 OVONIC device and diode array. The OVONIC memory device provides the high-speed and writability required by the terminal processor. In addition, the OVONIC device is nonvolatile which is an important point for this processor.

The Program Storage Address Register (PSAR) is incremented at the start of an instruction. The access time of the PS, therefore, is simply the time needed to gate the outputs of the memory cells into a latch. This approach was taken so as to minimize the access time; some 30 to 40 ns were saved. The PS cycle time is less than 160 ns. No instruction requires a cycle time of less than 160 ns, however.

A block of 256 eight-bit characters has been implemented in the machine. Additional program storage will be added in the near future to support additional firmware.

The scratchpad store is a high speed semiconductor memory which utilizes Schottky bipolar technology. Memory access time is 60 ns from address in to data out. A capacity of 256 eight-bit characters has been implemented. While the amount of scratchpad could be expanded the following approach would seem to be more practical. A large core memory could be added on as input/output device. The processor would continue to use the scratchpad store as the working store. Under program control additional information would be shuttled between the high speed scratchpad and the slower core. This would give the processor a "hierarchy" memory of sorts.

The idea of an address store has been used in the terminal processor. Because the actual chip used to construct the address store is organized as 16 four-bit words, the file registers have been included in the address store. The purpose of the address store is to provide multiple storage address registers. These storage address registers are accessible from the machine under program control. This gives the machine additional flexibility. Not depicted in Figure 12 is a data path from the Address Store to the ALU via the A-bus which permits the use of the ALU for storage address register incrementing or decrementing.

The Address Store is segmented in the following manner. The first four addresses are used as a push-down stack for the program memory. This allows nesting of subroutines up to four deep. The second four addresses are the Scratchpad Storage Address Registers (SAR's). The remaining eight locations constitute the file registers.

The scratchpad input register (S-Reg) is used to hold data during a scratchpad write. The S Register can also be used as a general register. The S Register serves as the implied destination for those two operand instructions which use the ALU.

A more detailed diagram of the flag registers and data arrays is given in Figure 13. The flag/data section is the I/O section of the processor.

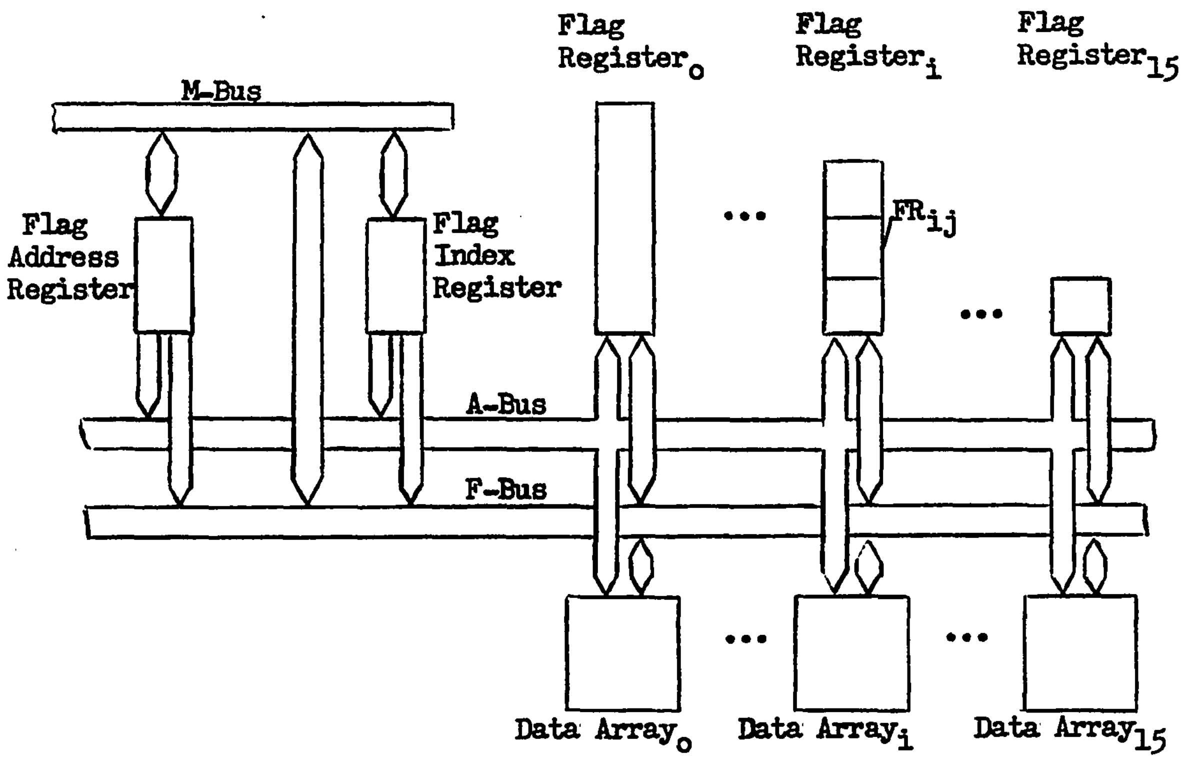


Figure 13. Flag/data unit

Each terminal device has a 'service needed flag' flip-flop (called a flag flip-flop) associated with it. This flag flip-flop is physically located in the processor. Flag flip-flops can be grouped into registers of up to eight bits. These registers are called flag registers. The maximum number of flag registers is 16.

Associated with each terminal device is a data register. The length of a data register is determined by the terminal device. Serial devices would have a data register of one bit. Terminals that are parallel data transmission devices would have longer data registers. If more than eight bits are to be transferred to a terminal device at once, a word assembly would be required. Thus, for example, if a 16-bit device is to be interfaced, two characters would be moved from the processor to the interface.

The flag/data section is addressed via the A-bus. The Flag Address Register (FAR) and the Flag Index Register (FIR) are used to hold flag address information. Both the FAR and the FIR are four-bit registers. The FAR and the FIR correspond to the GPR7 in the hypothetical machine. The FAR selects a flag register while the FIR selects a bit from the previously selected flag register. The three most significant bits of the FIR are used to select the flag bit. This permits the storage of two characters of information at the location in page zero which can be accessed using an RPS instruction which is explained later. For purposes of data movement the FAR and FIR can be treated as a single eight-bit register (the FAIR). The four bits of the FAR occupy the most significant bit positions of the FAIR.

The arithmetic/logic unit (ALU) is depicted in Figure 14. Registers I and II are shown in the ALU but are, in fact, general purpose registers.

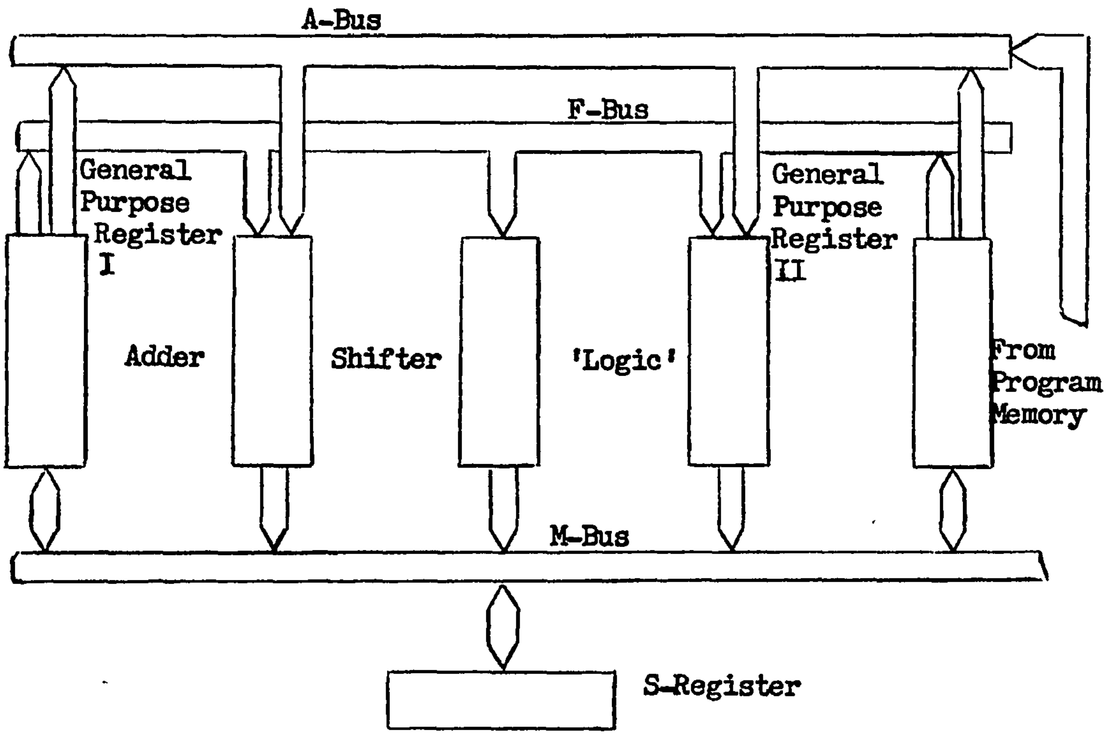


Figure 14. Arithmetic/logic unit

The ALU consists of an adder/subtracter, a shift network, and an 'or' circuit. 'And' capability is achieved by gating two registers onto the same bus at the same time then gating the contents of the bus into a third register. The result is the 'and' of the two registers.

The ALU is shared by the processor. Arithmetic/logic type instructions utilize the ALU during operation as do certain comparison instructions. In addition, the ALU is used to increment or decrement the SAR's.

Subtraction is in two's complement. That is, given two operands and the command to subtract the ALU forms the result in two's complement. The **add over flow flip-flop captures any overflows when adding; it can be tested by instruction (SAV).**

A combinational shifting network has been included in the ALU. This network extends a shifting capability to all working registers in the machine. This approach is less expensive than giving each individual register a shifting capability.

Processing Techniques

In the design of any communication processor certain special instructions can be included if these instructions result in a significant performance increase or a decrease in cost. These special instructions can be justified only if the task warrents their inclusion. Special instructions which are not used do not increase performance. This fact immediately eliminates such instructions as a floating point add from the communication processor.

The organization of the communication processor was oriented toward a polling scheme. Thus, certain instructions have been included to make polling more efficient. A description of how these instructions would be used follows.

The FAIR has the special purpose of addressing the flag and data arrays. The FAIR is divided into the FAR and FIR. The FAR is used to address a particular flag register or data array. The three most significant bits of the FIR are used to address a particular bit in the flag register or register in the data array. Two special instructions have been included to increment the FAR and the FIR. These instructions are Increment Flag Address Register (IFAR) and Increment Flag Index Register (IFIR). In addition, IFAR performs a zero test upon the contents of the FAR after the increment. If the FAR is zero, the contents of the next program store address are placed in the FAR. This provides a method of controlling the length of a polling loop. The length of the loop can be changed by rewriting the location following the IFAR instruction.

To permit using the program store for a nonvolatile storage of permanent parameters, a Read Program Store instruction has been included. Read Program Store uses a programmer selected register as a page zero address. The character read out is placed in the S register. This mode of operation could be used, for example, to perform a table look-up when a set flag is detected. The table would contain a starting address for the service routine.

The maintenance of multiple communication queues is not very sequential in nature even though the queues are verv sequential in nature. Processing usually involves moving data from one queue to another as well as updating queue pointers. Because of this characteristic the standard, automatically incrementing/decrementing scratchpad address register is not practically useful in communication processors. This is especially true in a processor in which separate stores are used for program and for data. The organization presented previously took this characteristic into account. The instruction set includes special instructions which are to be used for the sole purpose of incrementing and decrementing scratchpad address registers. This approach allows for more flexible use of the scratchpad store by virtue of the fact that the programmer is responsible for incrementing or decrementing the scratchpad store address registers. Bits in an instruction word could be used to specify an increment or decrement after each read or write operation. The short instruction word and added flexibility are preferred, however, as is the more basic type of instruction.

A certain number of instructions must be dedicated to I/O. Since carrier communications are serial in nature, instructions have been included to aid in serial I/O. In addition, I/O clock control has been included in the instruction set. This implies that no bit counters and similar devices are required within the individual devices' interfaces.

The majority of the rest of the instructions are skips and branches. That is, after a test which is not satisfied the next n locations are skipped. An example of this type of instruction is Zero Test Flag Character (ZTFC). This approach is implemented to keep the instruction word short. Note that the basic serial devices would be handled using the zero test and set data instructions rather than a move command.

A group of unconditional branches relative to the current location have been included. This, once again, has been done to keep the instructions short. A jump relative to the current location is not always adequate, however. Thus an absolute jump must also be provided.

The remaining instructions are those instructions that are more familiar to the computer user; add, subtraction, shift, etc. The terminal processor uses implied sources for the operands as well as an implied destination. This would probably not be the case in the hypothetical communication processor. The file registers provide an extra dimension of flexibility which would be decreased by the use of implied sources. The use of an implied destination would be retained, however, to keep the instruction word as short as possible.

The terminal processor has been constructed using high powered Transistor-Transistor Logic. Transistor-Transistor Logic does not have many of the exact characteristics of the proposed MSI family but does have many MSI functions. In addition, TTL is very inexpensive. The processor, less memories and I/O section, used about 300 TTL circuits, both MSI and SSI. The system is constructed upon printed circuit boards which are 11" by 14". Each board could contain a maximum of 144 integrated circuits; the actual high is 130. The system interconnections are provided by a 88 pin bus via two 44 pin connectors.

The I/O section, i.e., the Flag and Data arrays, is also constructed on 11" by 14" boards. There are eight data set interfaces and associated logic on a single board with a IC count of about 125. Data rates can be easily varied on an individual interface basis by changing a pair of capacitors. Start-stop bit decoding is, of course, a function of the program and not the hardware.

The entire processor is enclosed in a 17" by 22" by 13" housing complete with power supplies, cooling and front panel control switches. This size of enclosure could have been reduced had not a commercial power supply be used. The project was to be completed in to short a period of time to warrant development of an adequate power supply. The conclusions of this thesis report on the results obtained from this processor as well as comparing its performance with other communication processors.

CONCLUSIONS

This thesis has discussed the application of the new integrated circuit technology to the problems of communication processing. There are several key contributions which are summarized in this section. In addition, the results of the terminal processor project which experimentally verify many of the proposed concepts are outlined.

A MSI circuit family has been proposed which solves the three most important problems encountered in designing systems with MSI components. The first problem solved is that of reducing interconnections. A MSI family which does not take into account system and chip interconnections, i.e., does not reduce interconnections, is not a viable MSI family. The proposed logic family reduces system interconnections by facilitating a bus-oriented system. This is of critical importance in MSI system design.

The second problem solved by this logic family is that of minimal unique parts. The family, which contains nine parts including memories, does not limit a system designer to some standard machine. The designer is free to design an unique system. The complexity of the circuits is close to the current state-of-the-art; the nine MSI elements are practical.

The third problem solved is more subtle than the first two. This problem is that of control technique. The question of how control sequencing is to be done is of major importance when designing a MSI logic family. The solution described herein is an unique and viable one. The flexibility and generality of the control chip is great. The control technique can be expanded to many functional units. The functional units can operate serially or in parallel. Local timing pulses are generated; the number can be greatly expanded. The system designer has total freedom in designing functional units; there are no restrictions.

A MSI communication processor which utilizes the MSI logic family has been described. This communication processor shows how system architects can adapt a standard MSI logic family into an unique system. The MSI communication processor reflects the influence of the universal register approach to LSI architecture as does the MSI family. It is significant that it has been shown that this approach can be adapted into a special-purpose machine. The organization of the communication processor is unique and specialized; the logic family used in the design is unique but not specialized.

There are three organizational features in the MSI processor which make it a useful communication processor. One, the flag/data register concept and the polling technique reduce the interfacing hardware. Two, the indirect addressing method gives the machine a great deal of flexibility in queue maintenance. Three, the use of a bus-oriented system design gives the machine modularity at a low level.

A small version of the MSI processor has been designed and built. Since work on the two processors progressed in parallel not all the features proposed for the MSI processor were included in the actual processor. The most important feature not included was the functional completion bus. Initial testing has shown that the instruction execution rate of the processor is on the order of three million instructions per second. This rate is adequate for the second-level processing tasks. Programs have been written to aid in evaluating the machine. These programs indicate the maximum data rate that could be supported by the processor would be between

50,000 and 100,000 bits per second. This assumes serial devices. This would enable the processor to perform second-level processor functions for up to 128 low-speed terminals which is the maximum addressing capability of the machine.

A major point of this thesis has been that a second-level processor should minimize the device interfaces. The terminal processor data set interfaces, flag flip-flop, data register, and control circuitry amounts to 15 small-scale integrated circuits per line. This is a small amount of logic. Thus, the organization presented does minimize the problem of interfacing.

The data rates above were calculated based on a strictly sequential polling loop. This indicates that group polling is a viable technique. The performance of the processor compares quite favorably with other, similar processors.

The system proposed by Burner et al. (16) was a two-level system in which the second-level processor is a Interdata Model 3. This dual level processing scheme was able to support only 64 low-speed terminals. In addition, Burner's system would support only two different data transmission speeds; the flexibility of the system was very limited.

There are many commercial communication processors. Two, which are representative of this kind of processor, are the Microsystems Model 812 and the Varian 520/DC Communication System. The Microsystems 812 is a microprogrammed machine; the microinstructions are 16 bits. The ROS has a cycle time of 220 ns. The second-level processor is not a separate entity in the 812. The 812 is, in effect, both a first-level processor and a second-level processor. The first-level processor is macroprogrammed by the system programmer, the second-level processor is in the firmware and device interfaces,and can not be programmed without a change in the ROS. The 812 can handle up to 32 low-speed lines. These 32 lines are added in groups of eight and all eight lines must be identical in speed and start-stop codes. Each group of eight lines requires 64 characters of memory.

The terminal processor can handle up to 128 low-speed lines. Each line can be different both in speed and code. The terminal processor, with a Program Store having a bandwidth of only 2/3 that of the 812, is able to achieve throughput on the order of four times greater than that of the 812. This is due to two reasons. First, and most importantly, the organization was intended strictly to enable high data rates and maximum flexibility. The organization does that. Second, the use of a bipolar scratchpad has increased the terminal processor's throughput over what would be obtained using a core memory. This is one important reason that semiconductor stores are very desirable; the price/performance ratio is very good in low capacity memories. The terminal processor requires, in the sample programs, only five characters of scratchpad storage per line. This is slightly less than the 812.

The Varian 520/DC is a two-level processor. The first level is a Varian 520/i, a mini-computer. The second level is a hardwired data communications controller. This controller can only be used with a 520/i; it has no remote capability; stand alone operation is not possible. The controller is a hardwired second-level processor. The flexibility of the controller is achieved by programming the 520/i. The controller is a special I/O device which requires specialized software support. The controller has 10 characters of semiconductor storage per line. Its performance levels are quite high. The Varian 520/DC has a capacity of 64 1200 baud lines. The terminal processor could handle on the order of 40 1200 baud lines at best. This demonstrates the fact that microprogramming usually results in a somewhat reduced performance. It is significant, however, that the specialized terminal processor can handle two thirds the load of the hardwired machine. This speaks well for the polling technique and for the organization.

In conclusion, a viable, state-of-the-art approach to communication processing has been described. This approach has been shown to be practical and realistic. The approach is not based on present day computer architectures but is based on what computer architectures will be in the near future. Computer system architectures are changing rapidly because of the semiconductor industry. The system architect must be aware of, and involved in, the advances of semiconductors.

LITERATURE CITED

1. Hartung, A. F. Computer communications - An Overview. IEEE Convention Record. 1971.

2. Pyke, Thomas N. Time-shared computer systems. In Alt, F. L. and Rubinoff, M., eds. Advances in Computers. Vol. 8. Pp. 1-45. New York, New York, Academic Press. 1967.

3. Becker, H. Communication processing for large data networks. Data Processing Magazine 12, No. 11:51-55. 1970.

4. Newport, C. B. Applications and implications of mini-computers. AFIPS Spring Joint Computer Conf. Proc. 36:691-695. 1970.

5. Strachey, C. Time sharing in large fast computers. In International Conference on Information Processing Proceedings. Pp. 336-341. Paris, UNESCO. 1960.

6. Corbato, F. J., Merwin-Dagget, M. and Daley, R. C. An experimental time-shared system. AFIPS Spring Joint Computer Conf. Proc. 21:335-344. 1962.

7. Corbato, F. J. and Vyssotsky, V. A. Introduction and overview of the multics system. AFIPS Fall Joint Computer Conf. Proc, 27, Part 1:185-196. 1965.

8. Ossanna, J. F., Mikus, L. E. and Danten, S. D. Communications and input/output switching in a multiplex computing system. AFIPS Fall Joint Computer Conf. Proc. 27, Part 1:231-241. 1965.

9. Cohler, E. U. and Rubinstein, H. A bit-access computer in a communication system. AFIPS Fall Joint Computer Conf. Proc. 26:175-185. 1964.

10. Daley, E. A. and Scott, A. E. The IBM 7740 Communication Control System. IEEE Convention Record 12, Part 5:216-224. 1964.

11. Drescher, J. E. and Zito, C. A. The IBM 7741 - A communications-oriented computer. IEEE Convention Record 12, Part 5:207-215. 1964.

12. Byrns, P. D. Considerations in designing a computer communications system. Datamation 15, No. 10:78-83. Oct. 1969.

13. IEEE Computer Society Workshop on Computer Communication. Computer 4, No. 2:31-36. Mar./Apr. 1971.

14. Spencer, H. W., Shepardson, A. D. and McGowan, L. M. Small computer software. Computer Group News 3, No. 4:15-20. Aug. 1970.

15. Barth, J. Using mini-computers in teleprocessing systems. Data Processing Magazine 12, No. 11:51-55. 1970.

16. Burner, H. B., Million, R. P., Rechard, O. W. and Sobolewski, J. S. A programmable data concentrator for a large computing system. IEEE Transactions on Computers C-18:1030-1037. 1969.

17. Arnold, O. E. Automatic polling by remote multiplexers. Tele-communications 3, No. 6:17-19. June 1969.

18. Filipowsky, R. J. and Scherer, E. H. Digital data transmission systems of the future. IRE Transactions on Communication Systems CS-9: 88-96. 1961.

19. Licklider, J. C. R. Man-computer symbiosis. IRE Transactions on Human Factors in Engineering HFE-1:4-11. 1960.

20. Lewin, M. H. An introduction to computer graphic terminals. Proceedings of the IEEE 55:1544-1552. 1967.

21. Myer, T. H. and Sutherland, I. E. On the design of display processors. Communications of the ACM 11:410-414. 1968.

22. Foley, J. D. Evaluation of small computers and display controls for computer graphics. Computer Group News 3, No. 1:9-22. Jan./Feb. 1970.

23. Levy, S. Y., Linhardt, R. J., Miiller, H. S. and Sidnam, R. D. System utilization of large-scale integration. IEEE Transactions on Computers C-16:562-566. 1967.

24. Rice, R. Impact of arrays on digital systems. IEEE J. of Solid State Circuits SC-2:148-155. 1967.

25. Smith, M. G., Notz, W. A. and Schischa, E. The questions of systems implementation with large-scale integration. IEEE Transactions on Computers C-18:690-694. 1969.

26. Smith, M. G. and Notz, W. A. Large scale integration from a user's point of view. AFIPS Fall Joint Computer Conf. Proc. 31:87-94. 1967.

27. Feth, G. C. Systems design and hardware technology. Computer Group News 3, No. 2:24-28. Mar./Apr. 1970.

28. Henle, R. A. and Maley, G. A. How LSI is affecting logic design. IEEE Convention Record 1971:276-277.

29. Koczela, L. J. and Wang, G. Y. The design of a highly parallel computer organization. IEEE Transactions on Computers C-18:520-529. 1969.

30. Beelitz, H. R., Levy, S. Y., Linhardt, R. J. and Miiller, H. S. System architecture for large scale integration. AFIPS Fall Joint Computer Conf. Proc. 31:185-200. 1967.

31. Rice, R. LSI and computer system architecture. Computer Design 9, No. 12:57-63. Dec. 1970.

32. Atley, E., Ed. Can you build a system with off-the-shelf LSI. Electronic Design 18, No. 5:46-51. Mar. 1, 1970.

33. Avizienis, A. and Tung, C. A universal arithmetic building element (ABE) and design methods for arithmetic processors. IEEE Transactions on Computers C-19:733-745. 1970.

34. Raisanen, W. LSI memories change computer design. Electronic Design 16, No. 24:C5-C8. Nov. 21, 1968.

35. Moore, G. E. Semiconductor RAMS - a status report. Computer 4, No. 2:6-10. Mar./Apr. 1971.

36. Kautz, W. H. Cellular logic-in-memory arrays. IEEE Transactions on Computers C-18:719-727. 1969.

37. House, D. L. and Henzel, R. A. Semiconductor memories and minicomputers. Computer 4, No. 2:24-29. Mar./Apr. 1971.

38. Petritz, R. L. Current status of large scale integration technology. AFIPS Fall Joint Computer Conf. Proc. 31:65-86. 1967.

39. Vadasz, L. L., Chua, H. T. and Grove, A. S. Semiconductor random-access memories. IEEE Spectrum 8, No. 5:40-48. May 1971.

40. Zingg, R. J., Pohm, A. V., Haglund, R. A research recognition processor. National Electronics Conf. Proc. 26:104-109. 1970.

ACKNOWLEDGMENTS

The author acknowledges the many hours of assistance and advice supplied by Professor Arthur Pohm. Professor Roy Zingg contributed much of his time to the terminal processor project and gave much helpful advice. The author was influenced by the earlier work of Zingg, Pohm, and Haglund (40) and, in particular, the indirect addressing scheme was derived from their work. Special thanks are due R. Boswell, D. Durbin and D. Holste for constructing the processor. The author also acknowledges the help of his wife, who contributed far more than the typing of this thesis.

The terminal processor project was supported by Energy Conversion Devices, Inc. of Troy, Michigan.

APPENDIX A: INSTRUCTION SET DESCRIPTION

The instructions described herein are those instructions which have been implemented in the actual terminal processor. The bulk of the instructions are eight-bit instructions with a few 16-bit instructions. There are three basic instruction types; Scratchpad Store and Temporary File Register instructions, Skips and Jumps, and instructions which perform some operation on a register or the contents of a register. The instructions will be described in the above listed order. When appropriate, mnemonics are given for various instructions in addition to the hexadecimal and binary codes.

READ SCRATCHPAD, SCRATCHPAD ADDRESS REGISTER, DESTINATION

7	6	5 4	3 0
1	1	SAR	DEST.

Read the contents of the location specified by the selected Scratchpad Address Register (SAR) into the destination given. The destination specifications are given in Table 1. The SAR specifications are:

00 - SAR0

01 - SAR1

10 - SAR2

11 - SAR3

The file registers and the SAR's can not be used as destinations for this command.

Table 1. Destination specifications

Binary Code	Destination/Source
0000	None
0001	Register I
0010	Register II
0011	F Bus, Flag Register
0100	FAIR
0101	FAR
0110	FIR
0111	F Bus, Data Array
1000	SAR0
1001	SAR1
1010	SAR2
1011	SAR3
1100	S Register
1101	None
1110	None
1111	PSAR

WRITE SCRATCHPAD, SAR SOURCE

7	6	5 4	3 0
1	0	SAR	SOURCE

Write the contents of the specified source into the location given by the specified Address Register. The SAR and Source specifications are as in the Read command. The file registers and the SAR's can not be used for data sources.

INCREMENT SAR by n 50 - 57
IS, n

7					2	1 0
0	1	0	1	0	n	SAR

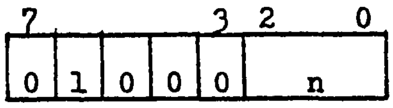
Increment the specified SAR. If n = 0, the increment is 1, if n = 1, the increment is 2.

DECREMENT SAR by n 58 - 5F
DS, n

7					2	1 0
0	1	0	1	1	n	SAR

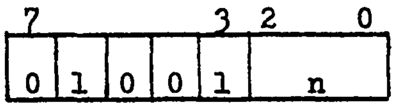
This instruction is identical with IS except the selected SAR is decremented.

READ TEMPORARY FILE REGISTER, n   40 - 47
RTR, n



Place the contents of the specified temporary file register in the S register. $(0 \leq n \leq 7)$

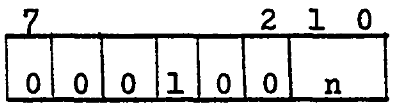
WRITE TEMPORARY FILE REGISTER, n   48 - 4F
WTR, n



Write the contents of the S register into the specified temporary file register. The TR specification is as in RTR.

A large number of the instructions are jumps and conditional skips. Skips and jump relative instructions give flexible but short instructions.

JUMP RELATIVE POSITIVE, n   10 - 13
JRP, n



Increment the program storage address register (PSAR) by one, two, three, or four according to the following specifications:

n	INC
00	1
01	2
10	3
11	4

Note that the amount of the increment does not necessarily correspond to the number of instructions jumped. The PSAR is incremented by one at the beginning of the instruction execution. For example, in Figure 15 the next instruction executed if n = +1 will be B while for n = +4 the next instruction executed will be E.

JUMP RELATIVE NEGATIVE, n 14 - 17
JRN, n

7					2	1	0
0	0	0	1	0	1	n	

Decrement the PSAR by n where n is specified as in JRP. The PSAR is decremented one more time than n to allow for the increment at the start of execution. In Figure 15, for example, the next instruction executed for JRN 1 will be Z while for JRN 4 W will be the next instruction executed.

JUMP ABSOLUTE, LOCATION 1A
JA, LOC

7							0
0	0	0	1	1	0	1	0
LOC							

Replace the contents of the PSAR with the contents of the address following this instruction (LOC). If additional pages of program store are implemented the page address would be stored following LOC.

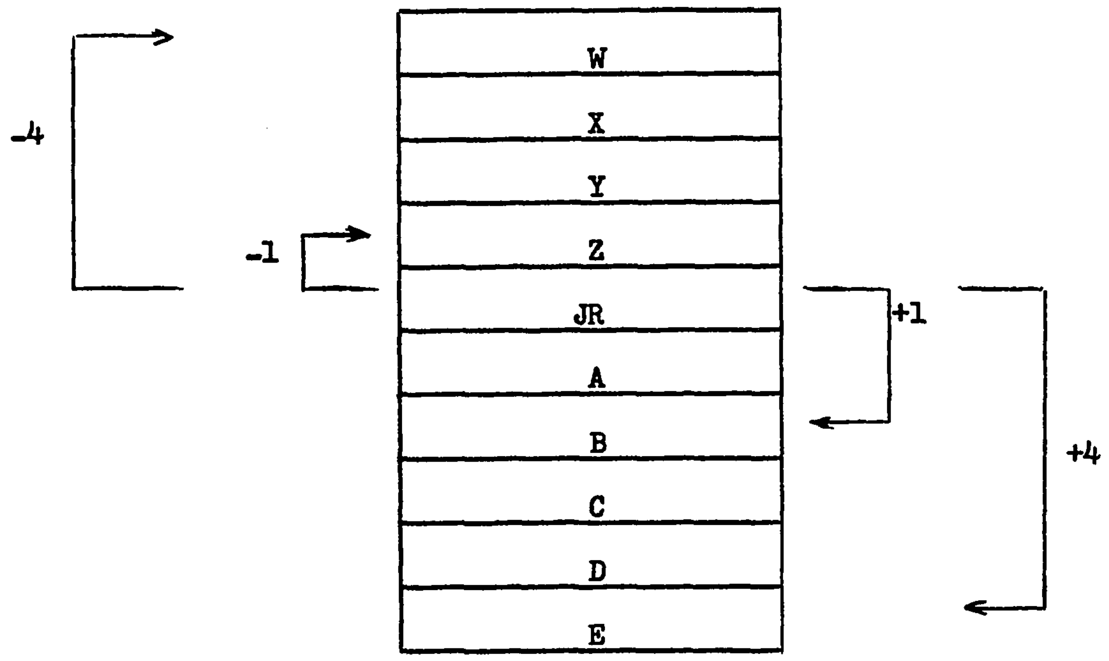


Figure 15. Jump relative examples

SUBROUTINE JUMP, LOCATION 18
SJ, LOC

7							0
0	0	0	1	1	0	0	0
LOC							

Store the current contents of the PSAR +1 in the top of the push-down stack then replace the contents of the PSAR with LOC. Since the PSAR is incremented at the start of instruction execution, the address stored in the push-down stack is PSAR +2.

The push-down stack is four deep; subroutines can be nested up to four deep. If four is exceeded, an error will result. Note also that the instruction RPS utilizes the push-down stack during its execution. Thus, if the maximum number of nested subroutines has been used, and if the last subroutine uses RPS, an unrecoverable error will result. That is, there must be at least one empty push-down location available to use RPS.

RETURN 19

Place the contents of the top of push-down stack in the PSAR and continue program execution with the instruction at that location.

ZERO TEST FLAG CHARACTER, SKIP n 20,21,23
ZTFC, n

7					2	1 0
0	0	1	0	0	0	n

Test the flag register specified by the FAR for zero. If the tested flag register is equal to zero, take the next instruction in sequence. If the tested flag register is not zero, increment the PSAR by the amount specified. The amount specifications is as follows: 00 skip 1, 01 skip 2, 11 skip 4. The skip specification 10 is illegal and will not be executed.

ZERO TEST FLAG BIT, SKIP n 24,25,27
ZTFB, n

7					2	1 0
0	0	1	0	0	1	n

Test the bit of the flag register specified by the FAIR. If the bit is equal to zero, take the next instruction in sequence. If the bit is not zero, increment the PSAR by n where n is specified as in ZTFC.

ZERO TEST DATA CHARACTER, SKIP n 28,29,2B
ZTDC, n

7					2	1 0
0	0	1	0	1	0	n

This instruction is the same as ZTFC except the data register addressed by the FAIR is tested.

ZERO TEST DATA BIT, SKIP n 2C,2D,2F
ZTDB, n

7					2	1 0
0	0	1	0	1	1	n

This instruction is the same as ZTFB except a single bit of the specified data register is tested.

ZERO TEST S REG, SKIP n 30,31,33
ZTSC, n

7					2	1 0
0	0	1	1	0	0	n

Test the contents of the S register for zero. If equal to zero take the next instruction in sequence. If not zero skip as in ZTFC.

ZERO TEST S, BIT ZERO, SKIP n 34,35,37
ZTSB, n

7					2	1 0
0	0	1	1	0	1	n

Test bit zero of the S register. If equal to zero take the next instruction. If not zero skip as in ZTFC.

SKIP IF ADDER OVERFLOW OF
SAV

If the add overflow flip-flop is equal to zero take the next instruction in sequence otherwise increment the PSAR by one.

NO OPERATION 1E
NOP

Increment the PSAR by one.

The rest of the instructions are operational instructions, i.e., they operate on a register or upon the contents of a register. The first group of instructions to be described are set and clear instructions.

SET FLAG 60
SF

Set the flag bit specified by the FAIR equal to one.

CLEAR FLAG 61
CF

Set the flag bit specified by the FAIR equal to zero.

SET DATA 62
SD

Set the data register specified by the FAIR equal to one. If the data register is more than one bit long all bits would be set equal to one.

CLEAR DATA 63
CD

Set the data register specified by the FAIR equal to zero. If the data register is more than one bit long all bits would be set equal to zero.

START I/O CLOCK 64
SIOC

Start the clock in the interface specified by the FAIR. The first negative transition will occur at T/2 seconds after SIOC where T is the clock period.

HIOC 65

Halt the clock in the interface specified by the FAIR. The clock will stop before the next negative transition if HIOC is executed within less than T/2 seconds. If HIOC is executed after T/2 seconds the clock will run until after the next positive transition.

SET S bit ZERO 70
SSO

Set bit zero of the S register equal to one.

CLEAR S bit ZERO 71
CSO

Set bit zero of the S register equal to zero.

ODD PARITY 08
OPAR

If the parity of RI is odd set s_o equal to one. If the parity of RI is even set s_o equal to zero.

EVEN PARITY 09
EPAR

If the parity of RI is even set s_o equal to one, if odd, set s_o equal to zero.

SET CONSTANT; NUMBER OA
SO; N

7							0
0	0	0	0	1	0	1	0
N							

Place the contents of the location following this instruction in RII.

CLEAR; REGISTER OD

7			4	3			0
0	0	0	0	1	1	0	1
				REG.			

Set the contents of the register specified equal to zero. The register specifications are as in READ.

INCREMENT; AMOUNT, REGISTER 00
INC; #, REG

7			4	3			0
0	0	0	0	0	0	0	0
AMOUNT				REG			

Add 'amount' (specified in the second character) to the contents of the specified register. The following registers can not be incremented by this instruction: SAR's, S REG, and FILE REGISTERS.

Otherwise the register specification is as in Table 1.

DECREMENT; AMOUNT, REGISTER 01
DEC; #, REG

7			4	3			0
0	0	0	0	0	0	0	1
AMOUNT				REG			

Subtract 'amount' from the specified register. The register specification is as in INC.

ADD 02

Add the contents of RI to RII and place the result in the S register.

SUBTRACT 03
SUB

Subtract the contents of RII from RI and place the result in the S register. Subtraction is two's complement.

SHIFT; END CONNECTIONS, REGISTER 05
SHIFT; EC, REG

7		5	4	3			0
0	0	0	0	0	1	0	1
SHIFT				REG			

Shift the specified register per the shift specification in Table 2. The register specification is as in READ except the FAR, FIR, FAIR, SAR's, FILE Registers, and PSAR can not be shifted.

Table 2. Shift Specifications

Bit 7	6	5	MEANING
0	0	0	Shift up*, shift in zero
0	0	1	Shift up, shift in one
0	1	0	Shift up, shift circular
0	1	1	Illegal
1	0	0	Shift down, shift in zero
1	0	1	Shift down, shift in one
1	1	0	Shift down, shift circular
1	1	1	Illegal

* 'UP' is toward the most significant bit.

OR 06

Logically 'or' the contents of RI and RII and place the result in the S register.

AND 07

Logically 'and' the contents of RI and RII and place the result in the S register.

COMPARE IMMEDIATE; CHARACTER OB
CPI; CHAR

7							0
0	0	0	0	1	0	1	1
CHAR							

Compare the contents of RI with the character following this instruction. If the contents of RI are equal to CHAR set s_0 equal to zero. If RI does not equal CHAR, set s_0 equal to one.

MOVE; SOURCE, DESTINATION OC
MOVE; SOU, DEST

7			4	3			0
0	0	0	0	1	1	0	0
SOU				DEST			

Move the contents of SOU into DEST. The register specifications are as in READ. The address store can not be used for both source and destination. If it is desired to MOVE from Address Register one to Address Register two, for example, a two step move will be necessary. For example:

MOVE; SARO, RI

MOVE; RI, SAR1.

If the F-bus is specified as a source or destination, the register used is specified by the FAIR according to Table 1.

READ PROGRAM STORE; REGISTER 1B
RPS; REG

7			4	3			0
0	0	0	1	1	0	1	1
				REG			

Use the contents of REG as an address in Program Store. Place the contents of that address in the S register. The current contents of the PSAR are stored in the push-down stack during execution of this instruction. The S register can not be used as base register. (See the SJ instruction description.)

INCREMENT FA REGISTER; MAXIMUM 68
IFAR; MAX

7			4	3			0
0	1	1	0	1	0	0	0
MAX				0	0	0	0

Increment the FAR by one. If the contents of the FAR are zero after the increment replace the contents of the FAR with MAX before proceeding. The FIR is cleared during execution of this instruction.

INCREMENT FI REGISTER 69
IFIR

Increment the FIR by two. The increment is two to allow storage of two characters at the location pointed to by the FAIR. For example:

FAIR	CHAR 1
FAIR +1	CHAR 2

These two characters could be word and page addresses, for example.

APPENDIX B. SAMPLE PROGRAMS

Two sample programs for polling techniques are described. The first is a sequential polling loop in which each flag register (FR) is tested. If any bit within the tested flag register is equal to one, further testing is done to find the first nonzero bit. The poll continues with the next flag register after completing the service routine. Note that one step of the service routine must be the resetting of the flag bit which is being serviced. The flow chart for the first program is given in Figure 16. The symbolic program given in Table 3.

Table 3. Sequential poll

	SYMBOLIC CODE	COMMENTS
POLL:	IFAR ; MAX	Inc. FAR, if zero replace with max
	ZTFC, 1	Test FR (FAR) for zero
	JRN, 3	Try again
	ZTFB, 2	Test flag bit (FAIR) for zero
	IFIR	Inc FIR
	JRN 2	Try again
	SJ; SERVICE ROUTINE	Go to service routine
	JA; POLL	Start poll again

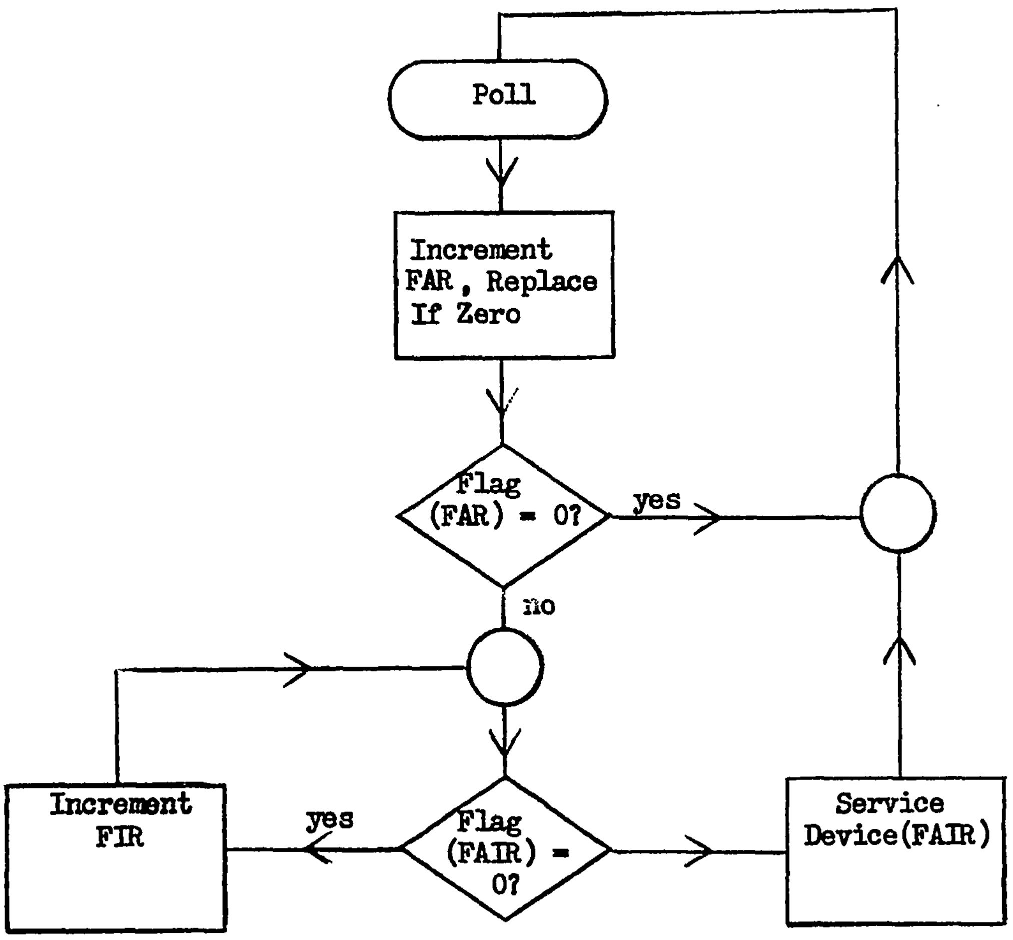
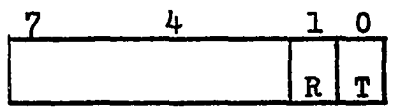


Figure 16. Sequential poll

The sequential polling technique might be to slow for certain devices. A second polling technique would be to poll a device, say device k, every other time. In this manner the time between polls on device k would be limited to the maximum service time for two devices. A program for polling in this manner is given in Table 4 and Figure 17. Locations Temp 1 and Temp 2 are used to store the flag address of k and the flag address that was last polled. The program assumes the FAIR holds the flag address of k at the start while Temp 2 holds the flag address of the last polled device.

If a device requires service a status look-up would be performed. A program for doing this is given in Table 5 and Figure 18. This program assumes that the FAIR holds the number of the device that requires service and the PS page zero location given by the FAIR holds the start of status for that device. (See Figure 19) The first character of status is configured as shown below:



If T = 1 the device is transmitting, if T - 0 and R = 1 the device is receiving. If T = 0, R = 0, and the flag = 1 a clear to send signal has been received from the data set.

Table 4. Polling example

	SYMBOLIC CODE	COMMENTS
Start:	ZTFB, 1	Test device k flag
	JRP 2	Flag (k) = 0
	SJ; SERVICE (k)	Flag (k) = 1
	MOVE; FAIR, S	Save k
	WTR n	
	RTR m	Restore last flag address
	MOVE; S FAIR	
	IFAR; MAX	Inc FAR
	ZTFC, 2	Test flag reg (FAR)
	JA; RESTORE	Flag reg = 0
	ZTFB, 2	Flag reg = 1
	IFIR	Inc flag bit
	JRN 2	Try again
	SJ; SERVICE (FAIR)	Service device (FAIR)
RESTORE:	MOVE; FAIR S	Save poll address
	WTR m	
	RTR n	Restore k
	MOVE; S FAIR	
	JA; START	Poll k

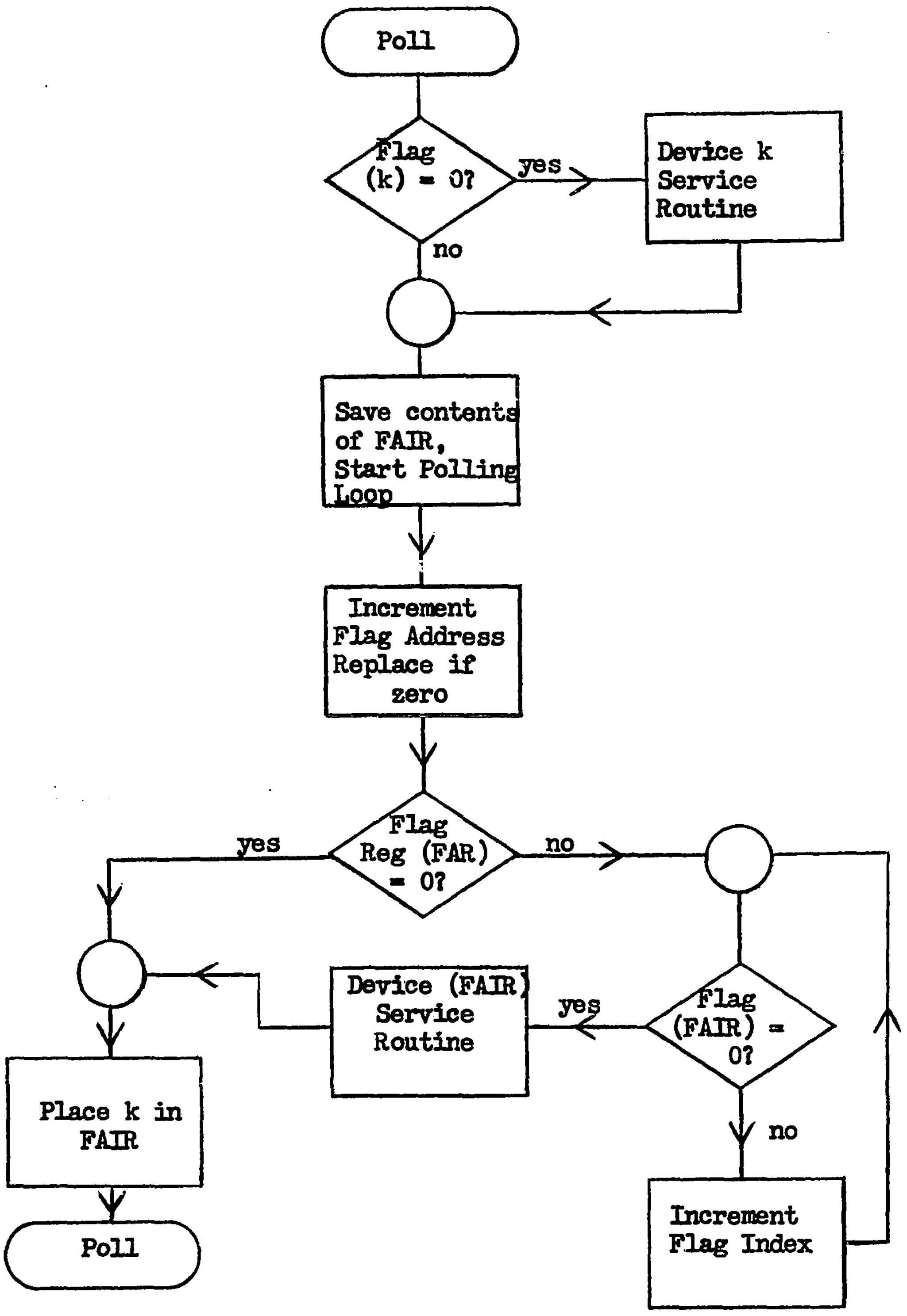


Figure 17. Polling example

Table 5. Status look-up

SYMBOLIC CODE	COMMENTS
RPS; FAIR	Status address to S
MOVE; S, MORO	Status address MARO
READ, MARO, S	Status character to S
ZTSO, 1	Transmit?
JRP 2	Go to receive test
JA; TRANS	Go to transmit
SHIFT; , EA, S	Shift to receive bit
ZTSO, 2	Receive?
JA; CTS	Go to clear to send
JA; REC	Go to receive

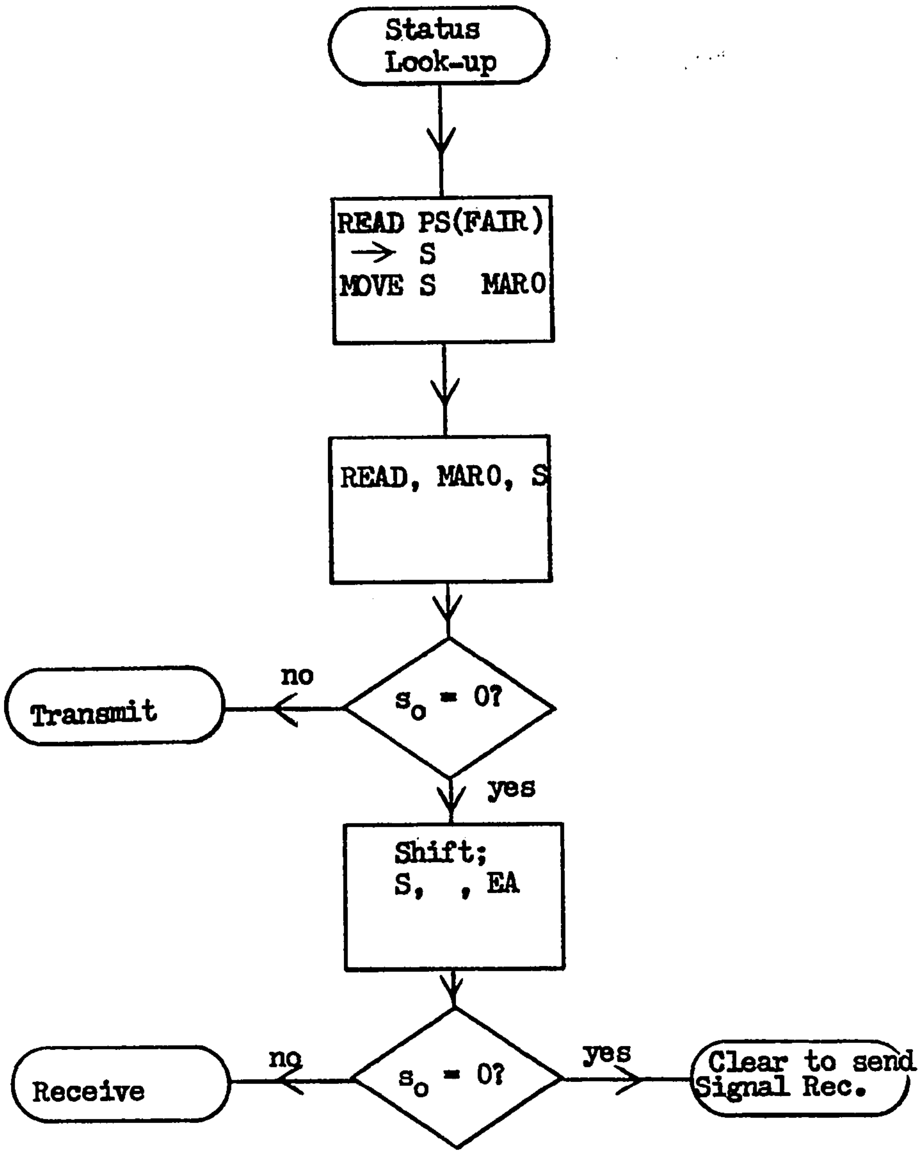


Figure 18. Status look-up

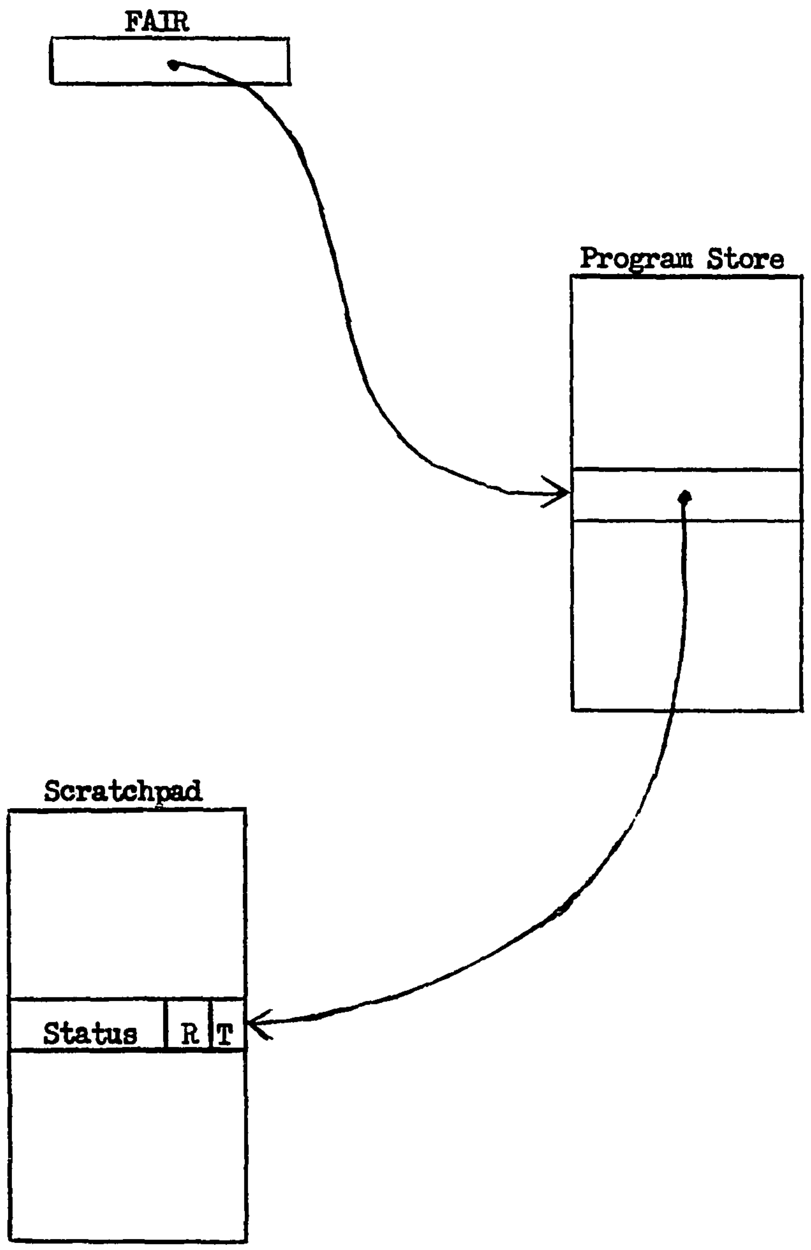


Figure 19. Status look-up technique