C. Greater adaptability to a wide variety of communication environments and procedures, initially and during operation.

These goals were achieved through the design features which are summarized below:

1. The design is made modular through the use of equipment pools. These pools of processing and storage equipment lead to a very high order of reliability with low initial and maintenance costs. In particular, a method has been evolved for using a number of small digital computers to provide message switching functions with a large amount of flexibility and modularity in operation and in installation.

2. A unique method of interfacing lines with processors has been invented which decreases buffering costs, failure interaction, and bit processing equipment requirements. The method is a combination of hardware, logic and software which ideally suits the problem at hand. It makes possible to use general-purpose processors in transferring and processing messages and thus provides great adaptability to changing environments and requirements. The direct access to memory of information lines provided by this technique allows much greater equipment efficiency in handling incoming and outgoing information in a class of multiuser computer systems extending well beyond message processing. The message processing example, however, shows sufficient details to support the claims of better efficiency.

3. An efficient method for the orderly storage and retrieval of messages in a modular drum (or other medium-access-time) storage system has been evolved. This method reduces the cost of core storage by allowing frequent drum accesses and reduces the cost of the drum system by making efficient use of the storage required by the drum.

1.2 Achieving Operational Reliability

To obtain the reliability offered by redundancy without the impoverishing costs of duplexing, we have turned to the use of modular equipment pools. As an example, the processor pool serves incoming and outgoing lines. Each of these lines has a usable service connection to three separate processors. Thus, if a single processor fails completely, its lines can be serviced by other processors that are not completely occupied. Because there are four processors in the pool, we need only 25 percent over-capacity (redundancy) in each processor to assure no loss of system capacity on a single processor failure. Many computer-centered systems require 100 percent redundancy (complete duplexing to achieve the same result. Other such systems do not use the pool concept so that despite similar degrees of modularity, a loss of a single module causes complete loss of service to a group of lines until it is either repaired or replaced. A similar pooled approach has been taken in the message storage area where any of three selection units can give a processor access to the message storage drum modules.

The modularity of this design also improves maintenance times by reducing the time required to isolate faults and by simplifying the training of maintenance personnel. Shorter times to find a difficulty and correct it result in greater system reliability.

An important requirement in pool operation is that failures on one side of an interface do not cause failures on the other. The magnetically coupled interfaces used in the direct-access-to-memory avoid this difficulty. The magnetic coupling is sufficiently loose that the failure of an active component cannot affect other equipments across the interface.

1.3 Achieving Greater Economy

We have achieved economy in this system primarily by the invention of a new type of line access to a processor which makes the processor more than five times more efficient in the acceptance and assembly of bits from a serial line. In all past such switching systems either external equipment was assigned to the task of bit assembly or it was done in the computer at great cost in number of memory cycles per bit. By making possible a single instruction time for the handling of a single bit, we have
been able both to eliminate external equipments and to make efficient usage of our computer in handling the bit transfers and assemblies as well as the more complex but less frequent jobs of switching.

Furthermore, the modularity of the processing pool allows us to choose the switch capacity to suit traffic conditions and numbers of lines in various installations thus minimizing the required equipment. In addition, the pooled approach results in much less redundancy so that we can expect an almost two-to-one cost reduction over other duplexed systems even at their optimum capacity.

1.4 Achieving Greater Operation Adaptability

The advantage of using a programmed processor for flexibility and adaptability to meet new requirements and situations over the older techniques of wired-in operations is now well recognized in the industry. Error correction and detection schemes may be implemented. Very sophisticated priority disciplines can be adopted on a moment’s notice to suit the situation at hand. Changes in codes, formats and routing indications can be handled. With pooled design, we can re-assign lines not only under failure conditions but under conditions of changing traffic patterns because line assignments are made electronically and each line can be assigned to any of three separate processors.

In talking about the adaptability to change, we should also speak of protection against unwarranted change. We observe that this system, being primarily under programmed control, permits protection from tampering by making initial program entry possible only from protected devices, while subsequent modification through the console or other external devices would be solely under the control of some internal program.

2.0 Description of the System

2.1 Description of Switch Operation

The block diagram of Figure 1 shows the equipment pools and their interconnections. The most important pools in the normal on-line operation of the switch are the processor pool, the message drum pool and the processing drum pool. The tape and console pool play a subsidiary job as they are only partially utilized in the routine switch operation. Briefly describing the input processing, the incoming bits for each line are sent to three processors in the processing pool. A supervisory program has previously assigned each line to one of these three processors. As the bits arrive, the processor assembles them into characters and checks the characters for special system coordination and control information. Included in these control information groups are the routing indicators which identify the message destination and precedence characters which indicate the priority of the message. When these arrive, an access is made to the processing drum pool by the processor to translate these groups into outgoing line numbers. When the outgoing line numbers and the precedence of the message are known to the processor and the message has fully arrived, an entry is made into a table (queue list) to alert the outgoing line that a message is awaiting transmission. In addition, the processor enters somewhat different information onto a ledger, which maintains an account of the message status; i.e., those lines on which the message is to be transmitted, those on which it has been transmitted and those which have acknowledged the transmission. Simultaneously, the processor trans-
fers the incoming message onto the ‘in-transit’ message-drum pool and onto the reference-tape pool. This it does in fixed-size bins. The initial entries into a journal are also made in the course of the input message processing. The journal is a chronological listing of the actions taken on a message while it is in the switch.

In output processing when a processor finds that one of its outgoing lines is no longer busy (or at fixed intervals after a nonbusy condition), it refers to the queue lists, which are identified by line and precedence, to obtain the message-drum address of the next message to be transmitted on that line. It then makes arrangements to retrieve that message from the drum and to send out the characters one bit at a time. In the meanwhile recordings of these actions are made upon the journal tape. When the message is completely transmitted, additional entries are made in the ledger to indicate transmission. When all transmissions of a message have been made the in-transit message and the ledger are erased.

2.2 The Processor Pool Functions

The processor pool accepts the bits from a line, assembles them into characters, disassembles them and sends them to a line. Secondly, it examines the incoming characters and performs a variety of routing, queuing and surveillance functions based on these. Thirdly, it stores groups of characters in its core memory for buffering other storage pools (primarily the message drum pool). Finally, during slack time and routinely, the processor evaluates switch operation and traffic for maintenance and adaptation purposes. It is evident that a general-purpose processor can handle all of these functions, and if it is time shared, efficient hardware usage can be achieved along with flexible operation.

In Section 3.1, it is demonstrated that bit and character processing dominate the other processes in computer usage thus making the interface techniques important in improving processor efficiency.

Because this paper is primarily on the interface technique which we have used between the communication lines and the processing pool, our discussion will center on this pool.

2.3 Message Drum Pool Functions

The major message drum pool function is the storage of messages to accommodate line availability/demand variations. This variation shows up as messages stored in “in-transit” storage awaiting lines to become free for transmission. It is clear that this function requires orderly and efficient storage of messages.

Orderly storage of information on the message drum and efficient transfer to and from the processing pool have been achieved by the use of list processing techniques. Because of properly chosen accessing procedures, all bins of information in the processor memory are of the same size, so that the information may be stored on the drum in fixed-size bins and successive bins chained to previous ones. A complete empties list keeps track of all available storage space remaining on the drum, thus permitting very efficient storage filling. Because all messages are stored in fixed-size bins, the problem of cross-office speed conversion is automatically accomplished. A full discussion of the message drum pool techniques is beyond the scope of this paper.

2.4 Processing Drum Pool Function

The processing drum pool stores the lists and registers used in the processing job by the processor pool. Its lists, as a matter of fact, are used in common by all of the processors of the processing pool to provide a record of: where the message will be kept on the message drum pool, on what lines the message is to be transmitted, in what sequence the message is to be transmitted, to what lines the message has been sent, which message to transmit next, where the message is located; and to update the ledger entry of lines to which the message has been sent. Even though normal operation of this pool is independent of the message drum pool, it makes use of the same drums for storage. This is possible and efficient because both storage capacities are determined by the maximum probable queue build up.

2.5 Other Equipment Groups

The other equipment groups within the switch are not as central to the normal operation of the switch, and will not be covered in this paper.
3.0 SYSTEM DESIGN

3.1 The Processor Pool

3.1.1 Processing Jobs

Messages on various lines and trunks may differ in code, bit rate, and message format, but in each case, the message consists of a header, text and ending. The header includes routing and message priority. Some messages are divided into 80 character blocks for transmission and reception purposes.

In a message store and forward system, processing of two types are encountered. The first type centers about the acceptance, storage and transmission of messages and the second type about the control of the switching system.

Tables I, II and III indicate typical message processing functions. Routine functions are classified in these tables as either of a bit, character block or message type.

System control functions keep the switching system performing effectively by supplying the switching center supervisor with data useful in the management of the store and forward center and network, and as an aid in maintaining and testing the switch, its programs, and data base. They are generally not performed regularly or very frequently and are given in Table IV.

3.1.2 Discussion of Store and Forward Switch Functions

An examination of the list of functions given in Tables I–IV shows the store and forward switch functions fall into four classes: data formatting, system operation, signal acceptance and transmission and recording (storage) operations.

In order to obtain an order of the importance of these functions to message switching, it is desirable to classify them in the order of their frequency of occurrence. The acceptance and forwarding of bits are the most frequently occurring functions. They occur at the incoming and outgoing bit rates. The next most frequently occurring functions are those associated with each and every character which enters or leaves the system, for example, control character check. Most functions are not performed on each character. Header validations and entries in queue lists for example are performed on entire messages independent of the number of characters they contain. The character functions occur \( \frac{1}{b} \) times as frequently as the bit function, where \( b \) is the number of bits per character and averages almost 7 bits per char-

<table>
<thead>
<tr>
<th>TABLE I—INPUT PROCESSING—ROUTINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>Accept bits and assemble characters</td>
</tr>
<tr>
<td>Check parity of characters</td>
</tr>
<tr>
<td>Detect system control characters</td>
</tr>
<tr>
<td>Assemble characters in words and bins</td>
</tr>
<tr>
<td>Write messages in “In-Transit Store”</td>
</tr>
<tr>
<td>Initiate preemption for flash messages</td>
</tr>
<tr>
<td>Verify header</td>
</tr>
<tr>
<td>Perform routing</td>
</tr>
<tr>
<td>Enter incoming message data in ledger</td>
</tr>
<tr>
<td>Check block parity</td>
</tr>
<tr>
<td>Write message on reference tape</td>
</tr>
<tr>
<td>Enter data in journal</td>
</tr>
<tr>
<td>Acknowledge accepted messages</td>
</tr>
<tr>
<td>Count blocks</td>
</tr>
<tr>
<td>Enter data in queue lists</td>
</tr>
<tr>
<td>Assign serial number for processing</td>
</tr>
</tbody>
</table>

From the collection of the Computer History Museum (www.computerhistory.org)
TABLE II—OUTPUT PROCESSING—ROUTINE

<table>
<thead>
<tr>
<th>Operation</th>
<th>Bit</th>
<th>Character</th>
<th>Block</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use queue lists to initiate message transmission</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Make journal entries</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Update ledger</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Retrieve messages from “In-Transit Store”</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Convert formats</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convert codes</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Construct block parity</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Check security for each block</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove messages from storage which have been transmitted</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Convert messages to a bit stream</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The next set of operations in the order of descending frequency of occurrence, are those which occur for each block. They occur \(1/c_b\) as often as the bit functions. Here, \(c\) is the number of characters per block and is about 80 characters. The remaining functions occur once per message or so. If there are \(m\) characters per message, the message functions occur \(1/m_b\) as often as the other functions where \(m\) is approximately 2000. Then each character, block or message function occurs respectively 8, 640, 16,000 times as infrequently as a bit function.

The most frequent functions then are the bit functions. When a bit arrives, it must be stored in the proper place in a character, used to update the character parity count, and counted to establish the arrival of a full character. When a full character is received, it is transferred to another location in memory for further processing. A similar per line process occurs in reverse order when information is disassembled for forwarding. These bit functions are performed on each line at a rate determined by the information rate on the line. In designing an equipment to perform this function, various line rates (from 75 bits per second to 4800 bps), and various bits per character (from 5 to 8 depending on their code) must be considered.

The character operations are somewhat more complex. Typically, characters must be examined to determine if they are system control or coordination characters, and if their parities are correct. When transmitting, the character codes may require conversion and the block parity must be determined. The characters also must be counted to determine block length.

Two characteristics of these bit and character functions should be noted. The first is that there exists a variety of functions and the

TABLE III—MESSAGE PROCESSING—NON-ROUTINE

<table>
<thead>
<tr>
<th>Operation</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Control errors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Print or display messages</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initiate service messages</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manually retrieve message</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TABLE IV—SYSTEM CONTROL PROCESSING TASKS

- Maintain in-transit storage status.
- Maintain status of traffic.
- Activate overdue message alarms.
- Activate queue threshold alarms.
- Execute maintenance routines.
- Control program maintenance routines.
- Activate and control start-up.
- Activate and control recovery.
- Check confidence levels on equipments and links.
- Provide line synchronization.
- Allocate hardware.
- Accept supervisor initiated commands.
- Provide statistical analyses of traffic.
second that these same functions are performed on all lines. The latter assumptions imply time sharing of equipment while the former implies a reasonably complex assortment of equipment.

3.1.3 The Processor Interface With Input and Output Lines

Because the most frequent operations in the message switch are the acceptance and the delivery of bits of information, earlier switch designs used special equipment to accept bits from a line and assemble them into characters for use by the processor. That approach involved considerable equipment which was peculiar to a particular line type. Recent designs have made this equipment sufficiently flexible (generally by pluggable programs) to be suitable for a wide variety of such lines. However, in so doing, any efficiency derived from special-purpose equipment was lost. Furthermore, an efficient method of providing alternate capability (redundancy) in case of equipment failure was not provided. Seeking economy, redundancy, flexibility and simplicity in handling bits, we use a general-purpose machine, taking advantage of its high speed to perform service for a number of lines. A direct and inexpensive interface is made to each line. Each line interfaces with a number of processors so that in case of failure, assignments can be made electronically for other processors to take over the lines formerly served by the inoperative processor.

The communication lines interface directly with computer memory cores in our design. A single instruction (one memory cycle in length): (1) accepts a bit from the communication lines; (2) puts it in the proper bit location in a memory word which is employed as a character buffer for that particular line; (3) checks to see if a complete character has yet arrived; and (4) computes the parity bit for the character. The operation is accomplished almost entirely with existing equipment in the main computer memory. Additionally, it provides alternate servers for each line with sufficient decoupling to assure that no failure on one side of the interface can cause a failure on the other side. Both the method of entry and the handling of the bits within the machine will be described in what follows.

3.1.3.1 Bit Handling By A New Instruction

The lines coming into the computer are actually wired into the memory of the computer as described in the next section. Each incoming communication line will be accompanied by a synchronizing line which specifies timing. Each of these wires is wired into a memory core at a location which is permanently reserved for that communication input. A program will cause the line termination memory locations to be scanned at times specified by interrupts from a real-time clock and will then put the received data bits into the proper position in that word. When the word is full, it will be transferred to another location and character processing will begin. A new type of instruction in the processor puts the bit into the proper place in the memory location for the line, checks to see if the location is full and computes the character parity. The entire instruction takes just one memory cycle of the computer. One additional memory cycle must be used to determine the line to be scanned next. This latter instruction is just an unconditional branch instruction whose address portion is determined when scan lists are made up.

The new type of instruction is externally determined which means that its effect is not determined at the time the program is written but rather is determined by subsequent input. This is not quite the same as a branch or skip instruction which merely constitutes a choice of where the next instruction is taken based on post-programming inputs.

With the direct interface it allows inexpensive appropriate control of a processor by a number of external users.

For purpose of accepting bits the instruction nature is determined by an incoming synch signal and by a marker bit which determines the end of character. However, the instruction is programmed in the normal manner as part of a subroutine which performs line scanning. The instruction format is as shown in Figure 2. The instruction code part of the instruction word contains a partial code and two externally set bits. The address field part of the instruction word contains the operand for the instruction. When the scan program causes this instruction to be read-out from the memory, the
operation which is executed will then depend upon the two indeterminate bits. For the time being, let us assume that the second indeterminate bit is a zero. The first indeterminate bit is the synch bit, which will be a one if a new data pulse has come in since the line was last scanned. The full instruction code is then 010110 which (see Figure 2) entails a shift operation upon part of the instruction word. The data bit which was in the last significant bit of the word is shifted left one position and, thereby, entered into the partly assembled character. Simultaneously, the character parity is updated. In addition, the synchronization bit is cleared and the entire word restored in the same memory location. If when we had read the line word out, no synch bit had come in, then the instruction (010100) would be interpreted as a no-op and the word restored without modification.

The use of the marker bit is fairly simple. When we arrive at the end of a character, we would like to know about it so that the entire character can be moved to another location in core memory thence to enter on character processing. To do this, the program sets a "1" into a particular bit of the input line instruction in the normal address field. Since the address field is initially all zeros, the marker bit will be the first one to show up in the second indeterminate bit of the instruction code (due to a succession of shifts). Thus, whenever a "1" appears in this position, it indicates that a complete character has been received, and the instruction becomes a branch instruction which branches to a sub-routine to take care of transferring the character. Two different branches are indicated here because the asynchronous nature of the system may allow a data bit to come in after the character was filled. On the other hand, there may be no new data bit that has come in. In one case, a single bit must be preserved in the memory location and in the other case, it need not be.

While this is one type of externally determined instruction, there are others possible, and in fact, the line equipment used in making the asynchronous to synchronous conversion can be embodied in an additional indeterminate synchronous bit possible in the instruction code. The full power of such an instruction particularly in message switching and command and control has not yet been realized.

3.1.3.2 Description of Interface Electronics

The basic system of entry into the memory is illustrated in Figure 3. Each incoming line is wired into a core which is effectively part of the main memory. These cores are all in the same bit position in the memory. The information coming into the core is written into each core on the basis of a coincidence of input-current and a write-current from the computer; the latter being supplied on every memory write cycle (by the y-drivers in Figure 3).

The information is read out of the core by the standard read cycle of the memory. Thus, once having been written in, a data bit is available for read-out with the rest of the word using the standard memory equipment already in the computer (with some exceptions to be covered later).
However, we recognize that a single bit could be written in and read-out many times since input pulses are much wider than memory cycles. Furthermore, a read-out of a zero data bit is ambiguous since it may indicate no input rather than zero-input. Thus, we must provide a synchronizing channel for each line, which will indicate when a data bit is available for reading. A similar input to another bit of the same word will be used for such synchronization purposes. However, the synchronization pulse must be timed to a single write pulse of the computer. This means there must be an asynchronous to synchronous converter timed from the master timing source (which also times the computers) for each incoming line. While we have devised a method of employing a third channel to obviate the need for the conversion equipment, we will not discuss it here because the economic tradeoffs are not clear.

Actually, the input lines are not wound through the cores which are normally located in the main memory stack. Instead, two additional very small memory planes (Figure 3, Auxiliary Memory Cores) are provided which are the storage locations for those particular bits of memory. These planes are wired in with the x and y and z lines of the main memory. A coincident current memory will provide two half-writes, x and y. Either the x or the y write current will be provided to the external cores as a 1/2 write common to all auxiliary cores. Thus, coincidence with the external 1/2 write signal will write in a "1". A separate sense winding will be provided to prevent interference with the normal words of memory and thus a separate sense amplifier will be provided for each of these two planes. The output will be logically added to the output of the normal sense amplifiers. Thus, all the input lines can be implemented with the addition of the auxiliary planes, two sense amplifiers, a few diodes and two gates.

While each line has been described as threading a single memory, in actuality it would thread cores in three separate processor memories. Thus, any one of the processors coupled to this line could service the line if its programmed scan included the line.

On-line program modification could take care of reassignment if it became necessary. Because the input line is magnetically coupled to the memory, no processor failure can disable it; i.e., it will still deliver its write current to the other processors with which it is associated. Furthermore, if a portion of the line equipment fails, it disables the particular line but in no way prevents the processors from servicing other lines. Thus, this magnetic coupling has the sort of ideal loose coupling described earlier.

3.1.4 Programming System

A master control program schedules operating programs and provides for hardware and line assignments. Consequently, it organizes routine and non-routine activities. The portion of the control program which refers to the routine functions resides in the core memory of each processor and has the facility to call the remaining program or portions of it from the processor drum memory to core storage when required. The operation of the control program is tied to interrupt signals from a real-time clock. These signals occur as often as required for the processor to sample its incoming lines for signals and to supply information to its outgoing lines. The processor need not keep track of elapsed time.

The operation of the control program and the operational programs for the functions proceed as follows: the control program, on the basis of information describing the lines assigned to it, schedules groups of lines to be scanned at a time. When a real-time interrupt occurs the program transfers control to an appropriate program for handling the line scanning functions. If during the line scan a full character is found to have entered the machine, the character will be entered into core memory with others in its message. If the character should be a system control character appropriate action will be noted in a list kept for scheduling by the control program. When all the lines have been scanned, control will be returned to the control program. At this time, the control program decides what its next course of action will be through an examination of its scheduling list. It might examine the control characters to determine their significance. If one of these was a start of message, it would initiate a header verification and then have control.
returned to it for user action. As characters are accumulated in memory or transmitted from memory by the operating routines, they would signal the control program to initiate a program which would bring more information from the drum for transmission or would have information taken from core memory for storage.

Programs for the handling of change in line assignments due to hardware failures will either be manually or automatically initiated. The manual initiation will occur from the supervisor's console. From this position, a computer will be selected and a message sent to this computer to initiate a program that would remove a computer from service and reassign its lines. This program will be retrieved from the processor drum, together with a list describing line assignments and characteristics. It will then determine another group of assignments based on an algorithm previously decided on, which is judged to minimize the overloading. Queue lengths would be available to this program if required. The change in line assignments requires no hardware change.

3.1.5 Processor Rate and Storage Requirements

The total number of memory cycles required for the processing job can be divided by the number of input bits to give a figure of merit which is independent of the capacity of the system. At times one sees such a figure expressed as instructions per throughput bit. However, the number of output bits exceeds the number of input bits in a system where multiple addresses are allowed. According to one estimate for a large system, the average output rate will be 75% higher than the average input rate. Thus, the proposed figure seems more natural and allows one to evaluate the required processor memory speed.

Other system factors do influence the proposed figure of merit. For example, the average number of bits per character, characters per block and blocks per message will determine the number of instructions per bit used in character and message processing. These parameters have been chosen as discussed in Section 3.1.2.

Trial programs of bit and character functions have been worked out to obtain the data on which to assess the processor requirements. In estimating program complexity, a prototype instruction code containing twenty instructions was used. The input bit processing takes two memory cycles per bit of direct input processing and approximately 0.5 memory cycle per bit attributable to the control program. The former load may be reduced by limiting the flexibility of the scan cycle, and in fact may be reduced to 1 memory cycle per bit for a fixed or nearly fixed scan. The output bit processing is similar in memory cycle usage.

Input character processing can be done in 36 memory cycles per character and the output processing in 28 memory cycles per character. To determine the full processor rate required for each incoming bit, the bit and character process rates must be augmented by the memory cycles required for the block and message functions. Our analysis of the drum transfer and other routine block and message functions indicates that 2500 instructions per message received and 50 instructions per block received is a generous allowance for these functions (i.e., 5000 and 100 memory cycles respectively).

The total number of memory cycles per input bit is then conservatively fixed at

$$2.5 + 1.75 \times 2.5 + \frac{36}{7} + \frac{28}{7} \times 1.75 + \frac{5000}{2000 \times 7} + \frac{50}{80 \times 7} = 19.5 \text{ memory cycles}$$

This is equivalent to ten instructions which is what most systems require for a single interrupt to process one input bit.

The core memory associated with each processor is used to hold the currently executed programs (control programs, operating programs, and maintenance programs when required), the data base for the execution of this program (code conversion tables, empties-registers, queue entries by precedence for each line, address of next ledger entry, etc.) and the messages prior to storage on the drum.

It is expected on the basis of preliminary estimates that the programming and its data would consume less than 2000 core memory words.
The message buffer is required to hold about 70 words per line. If a double buffer scheme is used to prevent buffer overlay before transfer to in-transit storage, for a computer handling 67 lines, about 10,000 words for message buffers are required per computer.

The most important facets of the processor, the interface with the communication lines and the memory size and rate have been discussed. The remaining features of the processors are of conventional nature.

BIBLIOGRAPHY


1. Introduction

HITAC 5020 family consists of general purpose computing systems designed to solve a wide variety of problems for both scientific and business data processing.

HITAC 5020 system is a medium-scale junior version of this family, and would have the same performance characteristics as IBM 360/40 - 50. A purely serial logic construction is the remarkable feature of this system.

HITAC 5020E (5020 ENHANCED) system is a large scale senior version of this family, and would have as much performance as IBM 360/62 - 70. But this system is constructed in serio-parallel logic form for economical reasons.

The central processing unit of this family is designed to rapidly and economically perform fixed or floating point arithmetic operations in either single or double precision, and even more to be able to process bit-wise variable length data. It contains 18 MC, 2-phase serial transistor-diode logic circuits and helical transmission lines for accumulators, index registers and other various registers.

Our design goals of 5020 family are as follows:

1. High Performance per Cost.
2. Program Flexibility.
3. Simultaneity and Multiprogram Activity.

The refined and flexible instruction system, in conjunction with a number of multi-purpose registers to be used as accumulators, index registers, and many input-output control registers, gives powerful possibilities to the programming activities.

The memory time sharing, the concurrent operation of various control unit, the automatic program interruption, the memory protection, and the introduction of priority mode are prepared.

This paper reviews the engineering design of HITAC 5020 and 5020E systems with primary concentration on central processing unit.

2. Outline of HITAC 5020 and 5020E System

The 5020 system is organized along four basic lines, Main Memory, Arithmetic and Control Unit, I/O Channels and I/O Devices.

Figures 1 and 2 show those 5020 and 5020E system configurations, respectively, and Figures 3 and 4 are pictures of the 5020 system.

2.1 The Main Core Memory

The core storage of the 5020 has a capacity ranging from 8,192 words to 65,536 words (32 bits each), and is directly accessible from the