Copyright © 1976 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 55, No. 2, February 1976
Printed in U.S.A.

Design of a Microprogram Control for a Processor in an Electronic Switching System

By T. F. STOREY

(Manuscript received May 27, 1975)

The processor in an electronic telephone switching system must be designed to efficiently and reliably process telephone calls. Because of the extended life span of a telephone switching system and the nature of the function it provides, it is advantageous to build a great deal of flexibility into the processor design. This paper provides details of a microprogram control design philosophy for the development of a medium-size processor that provides the required flexibility. The aspects of the microprogram control that make it very suitable for the design of a processor in a fault-tolerant system are also described.*

I. MICROPROGRAMMING BACKGROUND

In the 22 years since M. V. Wilkes proposed the concept of microprogram control,[1] the basic implementation has not varied significantly. Wilkes recognized even then that replacing the complex, irregular structure of the control section with a series of elementary and sequential microinstructions could result in the following advantages:

(*i*) A more regular and systematic approach to the design of the control section of a machine.
(*ii*) The ability to evolve the details of the implementation until late in the design state of the machine.
(*iii*) The ability to change or add to the instruction set after construction of the machine has been completed.
(*iv*) A simplified architecture that more readily lends itself to machine maintenance.

Because of the flexibility of a microcontrol design, many uses and variations of the design can be made to optimize the particular design

* The name of this processor is the 3A CC. It will be used in No. 3 ESS, No. 2B ESS, and other applications where a fault-tolerant system is required.

criteria involved. The design of a microprogram control architecture that utilizes this flexibility to implement additional features in a processor is covered in detail. This processor is used in small- to medium-sized telephone switching systems[2] that must be fault-tolerant. These features provide a processor design that is

(*i*) Self-checking.
(*ii*) Highly maintenance-oriented.
(*iii*) An efficient microstore (i.e., minimizes number of microstore words).
(*iv*) Efficient in real time.
(*v*) Amenable to system interconnection to provide a fault-tolerant system.

II. GENERAL SYSTEM DESCRIPTION

In an electronic switching system (ESS) that performs a telephone switching function, the processor complex[3–7] must have almost 100-percent uptime. (The goal is 2 hours downtime in 40 years.) To provide such reliability, redundancy must be built into the system since hardware failures are inevitable. With redundancy available, the

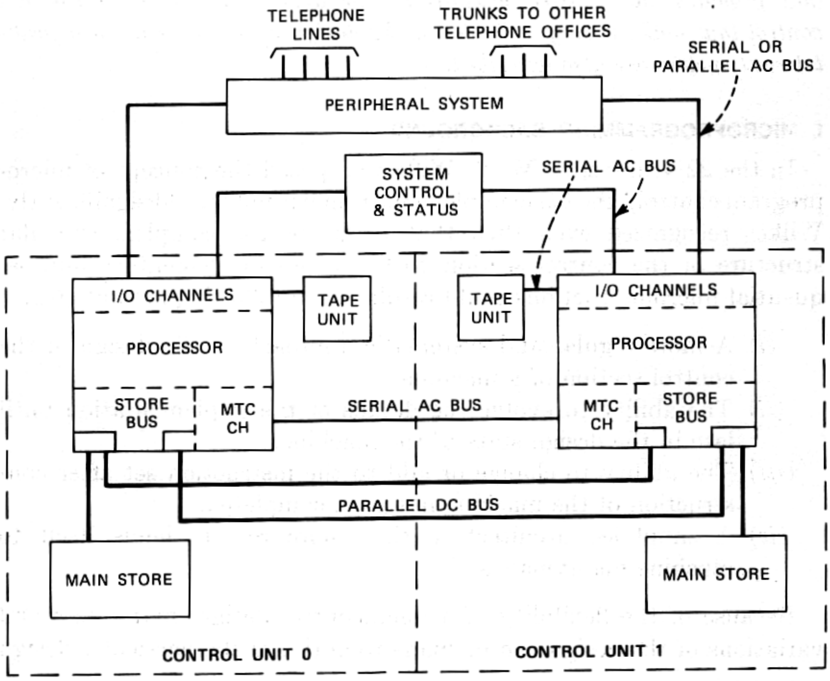


Fig. 1—System block diagram.

system can provide immediate detection of an error, a quick and efficient recovery from error (i.e., it can switch to a functioning unit), and the ability to diagnose and repair the failing unit before a second error can cause complete system failure.

The system environment in which the microprogram-controlled processor operates is shown in Fig. 1. The processor, main store, and tape unit are duplicated for reliability. These control units are treated as a single switchable entity since the quantity of equipment within each switchable block is small enough to meet the reliability requirement.

One system-design goal was to make each functional unit as autonomous and self-checking as possible with a minimum number of external signal leads. This provides sufficient flexibility to make the units expandable and changeable without much difficulty.

A simple dc store bus is used for communication between the stores and processors. The main store uses a semiconductor memory design and is contained within a small area. Even though the processors are not run synchronously, both the on-line and off-line stores are kept up to date by having the on-line processor write into both stores simultaneously. Because of the volatile nature of the semiconductor (dynamic IGFET) writeable memory, bulk storage backup (the tape unit) is required to reload program and translation data after a store failure.

III. GENERAL PROCESSOR DESCRIPTION

Figure 2 shows a detailed block diagram of the processor. It is functionally divided into six parts. There are 16 general-purpose registers and more than 30 special-purpose registers. Five of the special registers are used as the interface to the semiconductor main store. The interface is an asynchronous and relatively simple design. The microcontrol loads an address, data (if a write), and a control register. It then initiates a store cycle by issuing a start signal. Later the microcontrol tests for a store completion.

The microprogram control portion provides the complex control functions required to implement the instruction set and other sequencing functions, e.g., program reloading from the tape unit, trouble initialization, interrupt control, and man-to-machine interface functions.

The data-manipulation orders are designed specifically for implementing call-processing programs. Therefore, the orders include bit manipulation, testing, logical operations, etc., rather than complex arithmetic operations. A binary add is included to allow indexing and other simple arithmetic operations to be easily implemented. The data-manipulation logic includes rotation, all Boolean functions of two variables, first zero detection, and fast binary add.

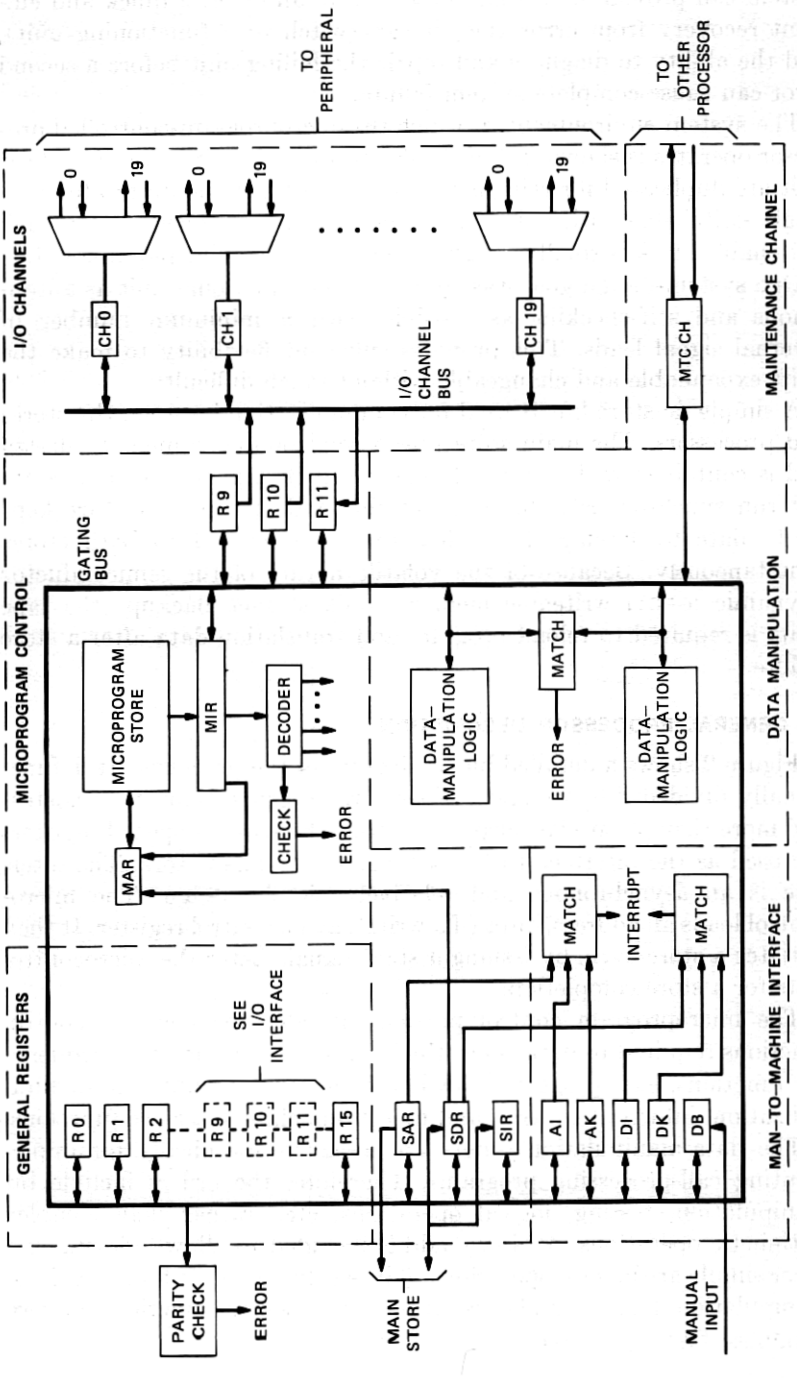


Fig. 2—Processor block diagram.

The remaining functional blocks in Fig. 2 are concerned with the interface of external units. The 20 main I/O channels, each with 20 subchannels, allow the processor to control and access up to 400 peripheral units by means of the serial data link. The serial subchannel transmits a 16-data-bit message using a 6.7-MHz bit rate. The tape unit is accessed by one of these serial channels. In addition, a man-to-machine interface with displays and manual inputs is integrated into the processor and executed under microprogram control. Finally, a maintenance channel can access the standby processor for diagnostic and control purposes. The maintenance channel transmits a switch message when an error is detected in the on-line processor. Control is then transferred to the standby processor. The use of a serial channel reduces the number of leads interconnecting the two processors and causes them to be loosely coupled. In addition to being more economical, this channel facilitates a split mode or stand-alone configuration for factory test or system test.

The basic execution of a macro-level instruction (OP code) by the processor is as follows (Fig. 3):

(*i*) The microcontrol issues a request to the main store and then executes a previously fetched instruction.

(*ii*) This request is performed and the access instruction is placed in a store-instruction register (SIR).

(*iii*) The microcontrol, having completed the previous macro-level instruction, tests for main store completion.

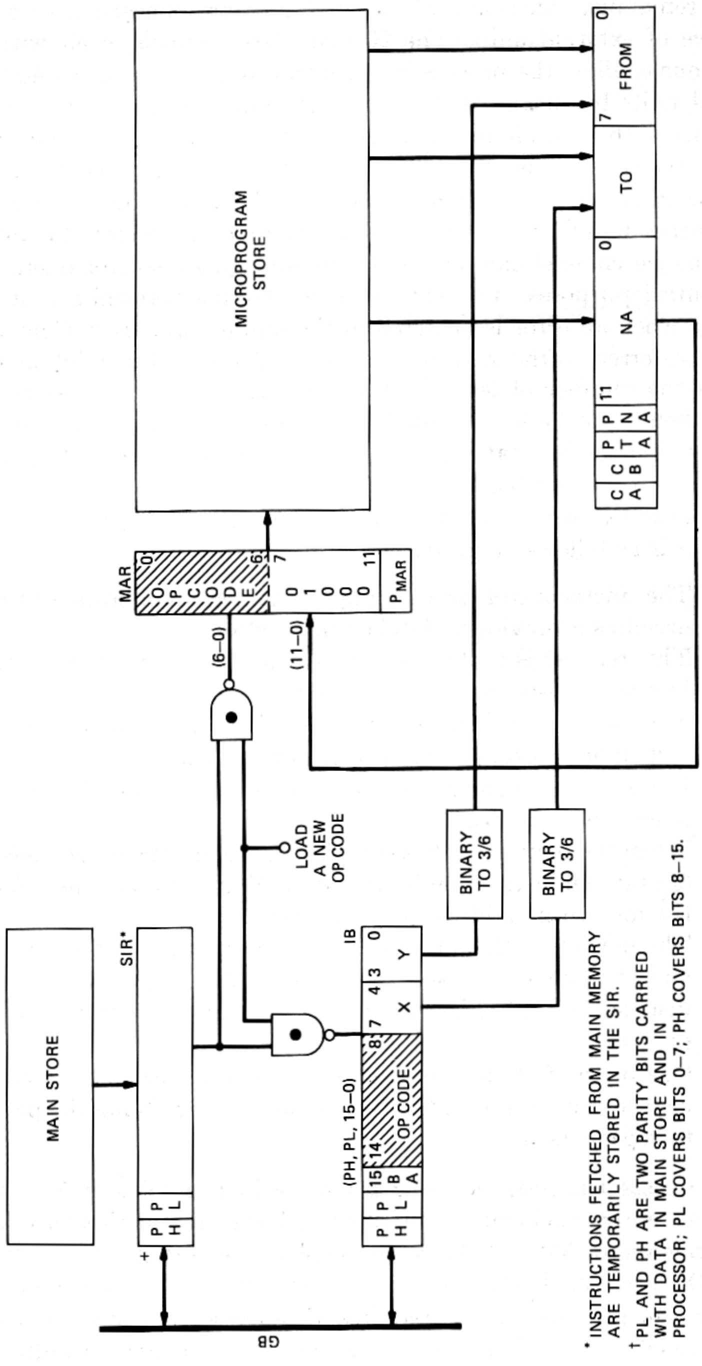
(*iv*) If the main store has not completed the requested cycle, the microcontrol loops.

(*v*) When the main store has completed, the microcontrol loads the SIR into an instruction buffer (IB) and a portion of the SIR into the microstore address register (MAR).

(*vi*) The portion of the SIR loaded into the MAR is the OP code field and it points to a starting address of a sequence of micro-instruction that will perform or interpret the function of that OP code.

(*vii*) One of the functions of each OP code sequence is to fetch the next instruction from the main store, thus enabling the process to repeat itself.

The processor is designed using a new ESS logic gate[8] with 5- to 6-ns delay. An entire packaging technology is built around this gate. The packaging allows 200 to 300 gates to be placed on a single, small circuit pack. On each pack the gates are interconnected in a customized manner. The processor design requires 55 of these logic packs to implement a 16,000-gate design. The microstore is implemented on additional



* INSTRUCTIONS FETCHED FROM MAIN MEMORY ARE TEMPORARILY STORED IN THE SIR.

† PL AND PH ARE TWO PARITY BITS CARRIED WITH DATA IN MAIN STORE AND IN PROCESSOR; PL COVERS BITS 0–7; PH COVERS BITS 8–15.

Fig. 3—Instruction buffer and its operation.

packs of which there are two for each group of 512 words (each word is 32 bits wide).

3.1 Microprogram control in an ESS environment

As in previous electronic switching systems, the goal is to provide a highly reliable switching system. The use of stored program control in ESS machines has provided easy implementation of customized features, system changes, or new services by changing the contents of the stored program. In addition, the microprogram memory provides a second level of flexibility, which creates further advantages.

The use of microprogram control permits designing a processor that is very regular in structure. This is achieved by centralizing the normally complex control section of a processor into one distinct unit or portion. Then, by segmenting each control function into a series of relatively simple microprogrammed steps, it is possible to achieve a uniform control entity. Control is easier to design through a systematic approach, and the rest of the processor is also made less complex. This results from the removal of most of the control and timing leads dispersed throughout each functional part in earlier types of processors.

Overall uniformity allows a self-checking design to be implemented without difficulty. One of the benefits of the self-checking machine in the ESS environment is that when an error is detected, the processor known to be faulty can be switched off-line immediately. That is, in a system where error detection is achieved through the synchronized operation of the *on-line* and *off-line* processors, the error indication from the match function between the two processors is unable to identify the faulty unit. As a result, other means must be provided to identify it.

In each processor, the logic is partitioned in such a way that maximum error detectability and immediate error indications are provided. For example, data registers are bit-sliced onto individual circuit packs so that multiple failures within a circuit pack will not go undetected by the parity check carried on each data register. As a result, the error circuit gives an immediate failure indication, and the problem of resolving which processor is faulty does not arise. In addition, immediate diagnostic results are achieved by sending the output of each error circuit to individual bits in an error register for easy analysis by diagnostic programs. Twenty-two error circuits in the processor are monitored by the error register. Because of the regular structure of the processor, each of these error bits tends to point to a unique portion of the machine (within a few circuit packs) that has failed and caused an error.

The use of a read-only memory (ROM) for the microstore in the microprogram control has facilitated additional simplifications in the control structure of the processor. In the event of a hardware failure or a start-up procedure caused by software problems, it is necessary to evoke a predetermined sequence of control functions. The use of a non-volatile microstore permits the start-up procedure or initialization to be microcoded and initiated easily when required. Hence, even if initialization is caused by a power interruption, these control sequences are available. This initialization may vary from a simple transfer to a starting location in main memory to full initialization that requires reloading main memory from the tape unit. The ability to microcode this sequence of functions not only eliminates complex sequencing logic but also provides the initialization procedures with all the advantages of microprogram control (e.g., self-checking, flexibility, and easy modification).

The ability to easily modify the macro-level instruction set even after the processor has been designed is a very attractive feature, especially in an ESS environment. Due to their function, ESSs must have an extended life. This extended life makes them vulnerable to increasing demands for more capacity and new features. Therefore, it is advantageous to be able to add to or modify the instruction set of the processor if increased throughput or other significant improvements can be obtained.

Another advantage is the potential to adapt to applications where a highly reliable stored program processor is required. This potential provides a step toward standardization of processors, or at least a reduction in the number of processors used in ESS applications. This adaptability, for example, could take the form of emulation of another processor's OP code set.

Most of the man-machine interface functions, which in past designs consisted of irregular and difficult-to-maintain logic, have been incorporated into the microprogram control (i.e., displaying register and/or memory contents). The use of microprogram control in this instance not only provides a relatively maintenance-free console panel but also provides a flexible man-machine interface.

The primary peripheral communication link in this processor is a serial I/O channel. There can be 20 autonomous I/O channels. With microprogramming, the I/O interfaces can be customized to the I/O task or the application. In addition, as other peripheral communication interfaces evolve (e.g., a parallel I/O bus is used), the microcode can easily be adapted to accommodate them.

Although this is a relatively small processor, significant real-time improvements can be attained with microprogramming. That is, register-to-register gating takes only 150 ns in the microcontrol, and transfers take place concurrently with microinstructions. As a result, there is about an 8-to-1 real-time improvement over main memory operations. As a result, the speed and flexibility of microprogram control can be customized to meet diverse requirements in an on-line, real-time, fault-tolerant control system.

3.2 General structure of the microprogram control

The self-checking microprogram control design is built around a high-speed ROM. The microstore has a 32-bit output and a maximum size of 4096 words (32 bits) and grows in increments of 512 words. The maximum access time is 65 ns. As shown in Fig. 4, the output of the microstore has three major fields:

(*i*) A TO field which normally defines a source register for a gating operation to be performed on each microcycle (a 150-ns interval).

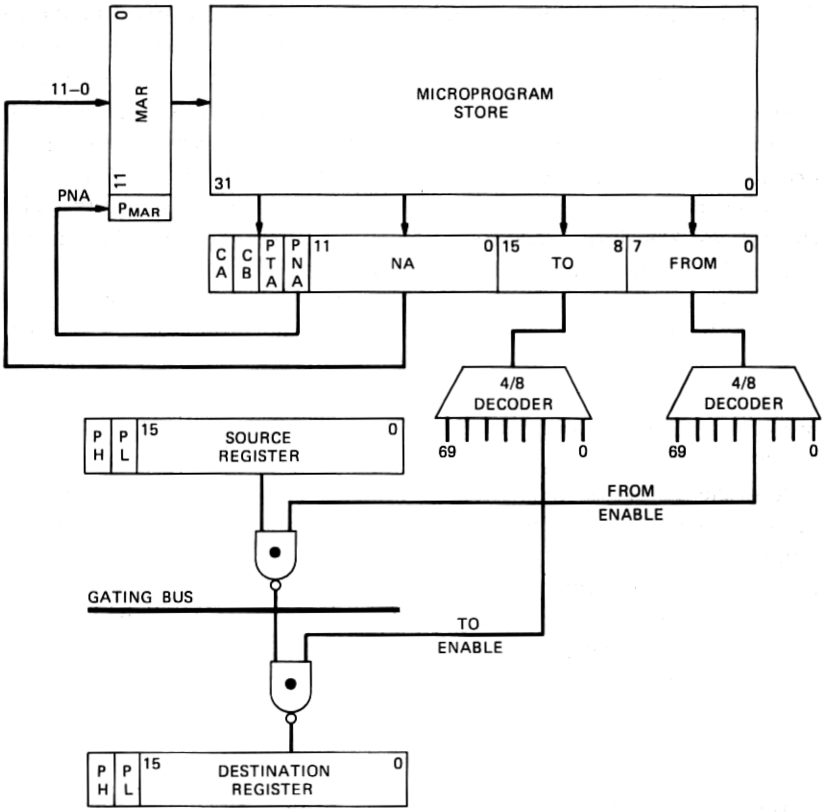


Fig. 4—Normal microinstruction execution.

(*ii*) A FROM field which defines a destination register under the same conditions.

(*iii*) A next address (NA) field that defines the location of the next microinstruction to be performed.

The correct sequencing of the microcontrol is checked by carrying a parity bit with the next-address field (PNA) and matching it with the parity of the accessed location (PTA). The TO and FROM fields are checked by encoding the control information in the microstore using a 4-out-of-8 code in each field. The control information is then decoded into a 1-out-of-70 code to enable a particular function and then re-encoded to a 4-out-of-8 code and checked.[9] The combination of these two checking techniques provides a microcontrol design that is highly self-checking.[10]

In this processor, the basic microinstruction set is centered around a register-to-register gating operation. In its simplest form, this gating function may be to set or clear individual flip-flops. To perform the gating operation, the TO and the FROM encoded control fields are read out of the microstore on each microinstruction. These fields are decoded and are used to enable a source and destination register for the normal gating command. Because an ESS environment emphasizes data processing rather than arithmetic operations, the register-to-register oriented microcommand set is very efficient. It is also useful in implementing the self-checking design of the processor. That is, by bit-slicing the processor's data registers (including the control and data access to each bit on individual circuit packs), the operation of a data transfer can easily be checked. This is done by performing the hardware check of the correct operation of the TO and FROM field decoders on each microcycle. The "checked" control signals are then fed to the bit-sliced data register circuit packs. By maintaining parity over the registers, a single failure of either the control or the data paths will result in a parity check failure.

A single failure in this context implies the failure of a single circuit pack. This could, of course, be caused by multiple failures within the circuit pack. The data registers as well as all of main store are, in fact, 2-bit-sliced to minimize the replication of the control signals to each bit of the respective registers and memory cells. As a result, there are two parity bits associated with each data word, PL, and PH (see Fig. 3). PL covers bits 0 through 7, and PH covers bits 8 through 15. In the actual partitioning, bits are paired on each circuit pack as follows: bits 0 and 8, 1 and 9, etc., and PL and PH. If a parity check error fires, either PL or PH or both could be in error.

When nongating types of microinstructions such as arithmetic or Boolean functions are to be performed, a different strategy is used.

These functions are performed in a separate entity in the processor called the data manipulation logic (DML). The DML is operated by first gating from the bit-sliced data registers to its operand fields which are buffered internally in the DML. A control field is loaded into the DML which defines the function to be performed on the operands previously loaded. The control field comes directly from the microstore and is defined by the microinstruction to be executed. Since the control field and the operands are buffered internally to the DML, the execution of a particular function is independent of the microsequence timing. In this way, functions that take longer than the basic register-to-register gating operation do not penalize the normal microinstruction execution time, nor do they require any special timing sequence. At a predetermined number of microcycles after the DML has been loaded, the microcontrol returns to the DML and gates the resultant data, as determined by the DML internal control states, to a data register.

Operations in the DML are checked by duplicating this section of the processor and matching. This approach has a number of advantages:

(*i*) Duplication is the most complete and most efficient check on the DML functions.

(*ii*) Since there is complete duplication in the DML, the logic partitioning can be optimized without concern for the failure modes for a given partitioning.

(*iii*) Duplication allows one uniform check to be performed on all miscellaneous functions that can be conveniently placed in the DML.

(*iv*) The match circuitry needs to be enabled only when gating out of the DML, and synchronization does not need to be applied to the duplicated copies of the DML other than when loading them with identical copies of operands and control states. The resulting functional execution of the DML as well as the parity generation of the resultant data can be performed independently of microsequence timing. As a result, duplicating and matching are easy to implement.

The processor's interface with main memory and I/O devices is performed in an asynchronous manner similar to that found in the DML operation. That is, interface or buffer registers are loaded with the normal register-to-register gating microinstructions. Then an execution signal is given, and a main memory cycle or an I/O cycle is started. The termination of the cycle is indicated by a completion signal which is then tested directly by the microcontrol. The result of this asynchronism is that the processor design is independent of the memory or I/O device execution times. The obvious advantage is that a variety of memory systems and I/O units can be used with this processor. With rapidly changing technology and the cost savings that may result, this is a major benefit of the processor's architecture.

In this microcontrol design, the microinstruction to be performed and an address are included each time a word is read out of the microstore. The address is then used to transfer to the next microinstruction in a particular sequence. To maintain simplicity in the design, the 12-bit address field can transfer to any word in the microstore address range (i.e., maximum size of microstore is 4096).

To provide a flexible and efficient microcontrol design, a number of alternative methods are provided for sequencing the microcontrol. The options or alternative ways that the next address can be obtained in a microinstruction sequence are as follows:

(*i*) The initial microinstruction of a machine OP code is initiated by having the microcontrol sitting in a microloop waiting for the main memory to fetch an instruction. The loop is excited when the main memory fetch is completed. The result of the fetch is gated into MAR, starting the microinstruction sequence that will perform that OP code.

(*ii*) On a number of microinstructions, it is desirable to obtain data constants from the microstore. To efficiently use the microstore, it is advantageous to store these data in the NA field and to obtain the next address by incrementing the previous contents of the MAR. This type of operation has been used as follows:

(1) To obtain data constants. Data constants, for example, can be used to generate target addresses in main memory.

(2) To obtain additional control constants. The main use of this is the simultaneous loading of the control states into the DML with a normal microinstruction loading of the operand fields. This not only reduces the amount of microcode required, but also increases the speed of DML functions.

(3) To load a return-address register (RAR) that is used in conjunction with the microcontrol to implement microsubroutine returns. This also reduces the amount of microcode needed.

(*iii*) Conditionally branching on a number of status bits allows convenient testing of various machine states such as adder overflows from the DML or completion states from external units. The latter is most useful for providing the asynchronous timing between the processor and its memory, as well as between the processor and peripheral units.

(*iv*) The ability to index into microstore is provided by ORing a 4-bit binary field into the address in MAR. Indexing effectively utilizes the microstore by providing an efficient method of extracting data or control from table structures in the microstore.

(*v*) Interrupts are initiated by jamming a hard-wired address into the MAR. This address points to a microinstruction sequence which interrogates various machine states and then determines what action is to be taken. This feature simplifies implementation and checking of complicated and difficult logic. It also provides a highly flexible and easily modified interrupt structure.

(*vi*) Through the maintenance reset function (MRF), a hard-wired address is jammed into the MAR. The MRF produces a series of microinstructions that perform the bootstrapping operation of the processor during a start-up procedure.

The preceding list of alternative methods for sequencing the microcontrol represents an economic compromise. The increased cost and complexity of more exotic, more powerful microcontrol sequencing would not be offset sufficiently by reduced microstore requirements. Of course, if real time is the prime concern, the increased speed of a more powerful microcontrol would justify the increased cost. Since this machine is intended for the small- to medium-sized ESS offices, cost is the dominant factor. To minimize cost, sufficient microsequencing flexibility has been implemented to reduce costly microstore and at the same time achieve a reasonably good throughput. The standard microcode provided with this processor will be implemented with about 1000 words. For those applications that need additional real time or other features, the ability to expand to 4096 words is provided.

The microcontrol uses a high-speed ROM which has a read access range of 30 to 65 ns. The minimum cycle time of a microinstruction sequence is determined by the maximum access time of the ROM plus the time necessary to calculate the address of the word to be accessed and/or the maximum time to execute each microinstruction. The processor has a gate with a 5- to 6-ns delay. The use of the gate, its associated technology, and the 65-ns ROM results in a microcycle execution time of 150 ns.

The design of the main memory for this processor uses a dynamic IGFET refreshable cell with an access time of about 750 ns as seen at the store itself. Taking into account the control and data delays between the processor and the store, the effective main memory access time will be about 1 μs. It should be again noted that the timing between the processor and the store is completely asynchronous and, if faster or cheaper memories become available, they will be readily adaptable to the architecture of the processor.

The effective execution rate of main memory OP codes is, therefore, determined by three main variables:

(*i*) A 150-ns microinstruction cycle time.
(*ii*) A 1-μs main store access time.
(*iii*) The number and type of microinstructions that are used to implement a main memory OP code.

As far as the architecture of the processor is concerned, the 150-ns microinstruction cycle time is a constant. The memory cycle time is a variable since it depends upon the main memory chosen. The number and type of microinstructions for each OP code is a function of how the microcode sequences are designed. The use of subroutines in the microcode, as well as other techniques to limit the total amount of microstore, minimizes the initial cost of the processor. The resulting mix of the number of microinstructions per OP code tends to make the effective execution rate of OP codes processor-limited, assuming the 1-μs access time for the IGFET refreshable main memory. It can be shown that by recoding the microprogram of the OP codes using more straight in-line coding and more microstore, it is possible to make the effective execution main-memory bound rather than processor bound.

IV. DETAILS OF THE MICROPROGRAM CONTROL DESIGN

4.1 Microinstructions

The processor, excluding the microcontrol, consists of a collection of distinct sets of registers. The set partitioning is done on a functional basis with each set optimized to provide a particular task (see Fig. 2). The various sets and the number of registers included are:

(*i*) General registers (16). These registers provide a set of general-purpose program-addressable registers that are used for high-speed buffer storage for the macro-level (main-memory) programmer.
(*ii*) Special registers (16). These registers are used for a number of special-purpose functions such as generating interrupts, providing error indications, displaying system status, and interfacing with main memory.
(*iii*) DML registers (3). These registers provide the buffer storage required to hold the operands and control states in the DML so that the DML execution timing is asynchronous with respect to the microcontrol timing.

(*iv*) Main memory registers (4). These registers provide addresses to main memory and receive instructions and data coming from main memory.

(*v*) I/O registers (3). One of these registers buffers the control information needed to access the I/O channels. The other registers provide the means of transmitting data to and from the I/O channels.

(*vi*) Console associated registers (5). These registers are used in conjunction with the console panel to load and display various other registers in the machine and to provide match functions on address and data constants associated with main-memory operations.

With the use of the microcontrol, most of the irregular machine structure has been removed. In its place is the collection of register sets just listed. Special functions such as additions, subtractions, Boolean operations, or other operations not easily handled in a single microcycle are performed by attaching combinational logic to the outputs of some of the register sets. The outputs of this logic are then gated under microprogram control to other registers or to status bits in the microcontrol where they can be easily tested by the microcontrol.

In the description to follow, the microinstructions that control the data flow in and out of the registers are partitioned into functional groups. Each group function is described. It should be noted that the set partitioning of the registers previously listed is not related to the functional grouping of the microinstructions that control the registers, although in some cases the partitioning corresponds with them.

4.2 Register-to-register gating

Because of the machine's dependence on register-to-register gating, the microcontrol architecture is centered around this microinstruction. On each microcycle, a TO field and a FROM field are gated out of the microstore and buffered in the microinstruction register (MIR). These fields are decoded and then are used to enable a source and a destination register (see Fig. 4). The control fields are encoded in a 4-out-of-8 code so that faults in the microstore, the MIR, or the decoder will be detected. One of the methods of obtaining the 4-out-of-8 codes that enable the gating TO and FROM registers is to read them directly from the microstore. A disadvantage of this approach is that to provide all the gating combinations between the 16 general registers requires 256 entries in the microstore. Because of the relatively high cost of the microstore, which in turn influences the limited amount of microstore available for use, an alternative method is desirable. The following implementation appears to be the most flexible and requires a minimum amount of circuitry.

It should first be noted that when an OP code is obtained from main memory, it is loaded into the MAR and the IB. (See Fig. 3.) In addition to the 7-bit OP code, two 4-bit binary operand fields may be loaded into the IB, as shown in Fig. 5. These two fields, *X* and *Y*, have binary to 3-out-of-6 translators attached to their outputs. The outputs of *X* and *Y* 3-out-of-6 translators are conditionally gated to the low 6 bits of the TO and FROM field in the MIR. This conditional gating is enabled when all zeros are detected coming from the microstore in the same 6-bit positions. In the upper 2 bits coming from the microstore, a 1-out-of-2 code is normally supplied. This code, which is determined by the OP code, can be used to select either the general register set or the special register set in either the TO or the FROM field. This permits gating of

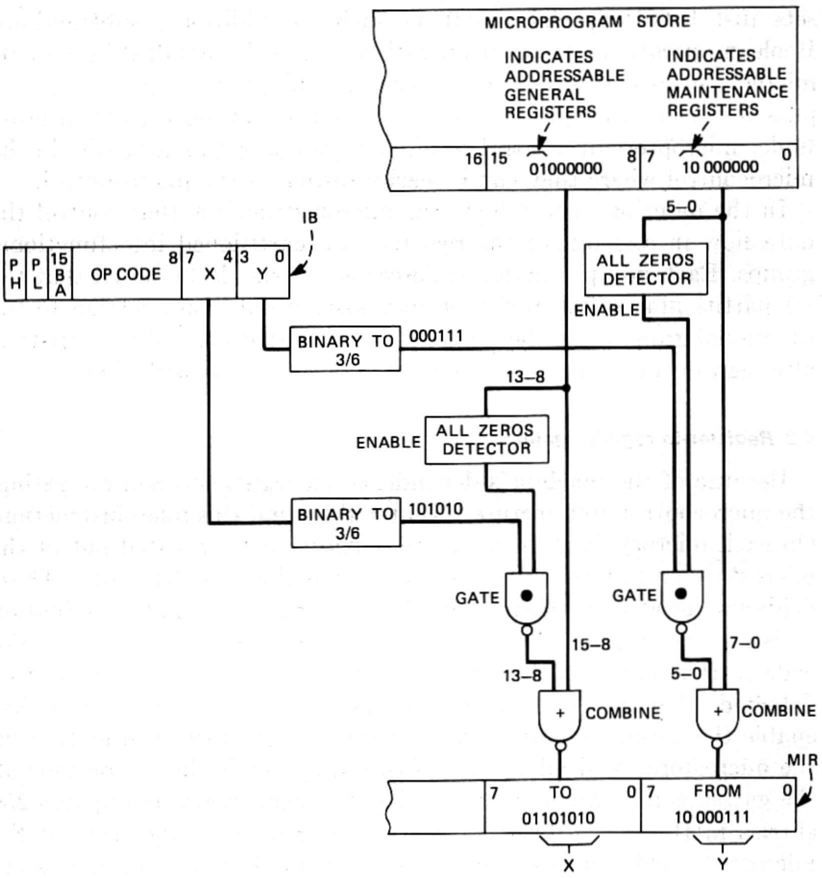


Fig. 5—OPERAND translation (normal).

any general register to any special register in a single microinstruction. One word of microstore provides the enabling signal. As indicated in Fig. 5, the signal coming from the microstore is not inhibited. The result is the logical OR of the microstore output and the 3-out-of-6 translators.

In a number of instances, it is desirable to exchange the roles of the X and Y operand fields. This is useful when the contents of various general registers are to be exchanged or swapped. Again, a number of solutions to this problem are possible. The following solution is preferred because it offers simplicity and speed, resulting from an extension of the method described previously. By storing 1s in the upper 2 bits of the TO or FROM field, and using a 2-input gate to detect this condition, a swap of the 3-out-of-6 codes coming from the X and Y fields can be implemented (Fig. 6). Note that the swap is not performed unless the low 6 bits of the respective TO or FROM coming from the microstore are zeros. It is also necessary in this instance to clear the high-order bit in the MIR field in which a swap is enabled. Because of this, the only combination allowed in the swap operation occurs when the swap involves general registers (01 in the upper 2 bits of the field).

Complete independence exists between the TO and FROM fields relative to the circuitry involved in the operand translation just described. It is therefore possible, for example, to have a 4-out-of-8 code loaded directly from the microstore in the FROM field and to have 11000000 in the TO field from the microstore. The latter results in swapping of the binary to 3-out-of-6 translator from the Y field into the TO field. As a result, most combinations of control signals for the microinstructions that involve register gating can be provided efficiently with a minimum amount of circuitry. Also, because of the use of the m-out-of-n checking techniques on the 4-out-of-8 decoders, all of the circuitry involved in the operand translations described are checked by circuitry that has already been provided.

4.3 DML operation

The DML is that portion of the machine that performs the arithmetic, Boolean, rotates, and other miscellaneous functions. These functions are collected into one entity, duplicated, and matched. The grouping of these functions is advantageous for the microcontrol design as well. A number of the operations that are to be performed in the DML take longer to execute than the basic gating cycle, and, therefore, a single and unified approach can be used for all the DML functions.

A function (FN) register is provided internal to each duplicated copy of the DML (Fig. 7). When the DML function to be performed is not a gating operation, the FN register is loaded with a control constant.

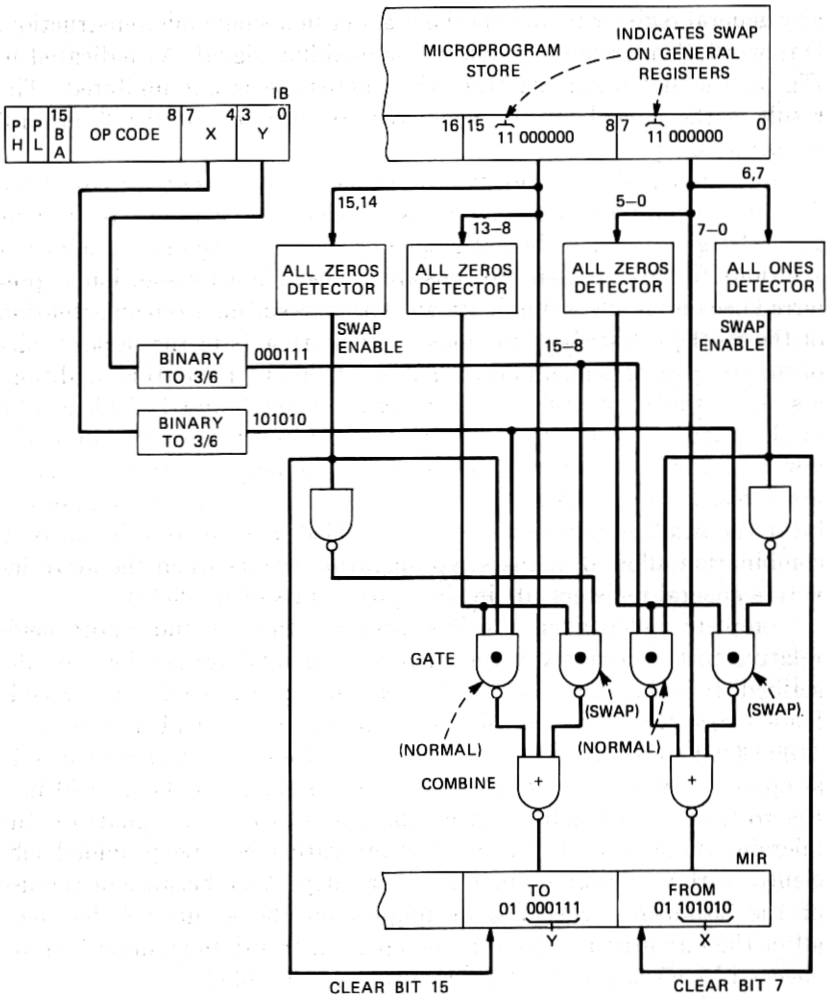


Fig. 6—OPERAND translation (swapping).

This control enables the appropriate control signals to execute the desired function. The function to be performed is executed on the operands which have been loaded into buffer registers AR and BR in the DML. Not only does the FN register set up the combinational logic to perform the logic operation on these registers, but it also determines what status bits are to be gated to the microcontrol status (MCS) register. In this way, the microcontrol can easily test for adder overflows, low-order zero test failures, etc. The use of the FN register and the buffer register for operands makes the DML execution asynchronous

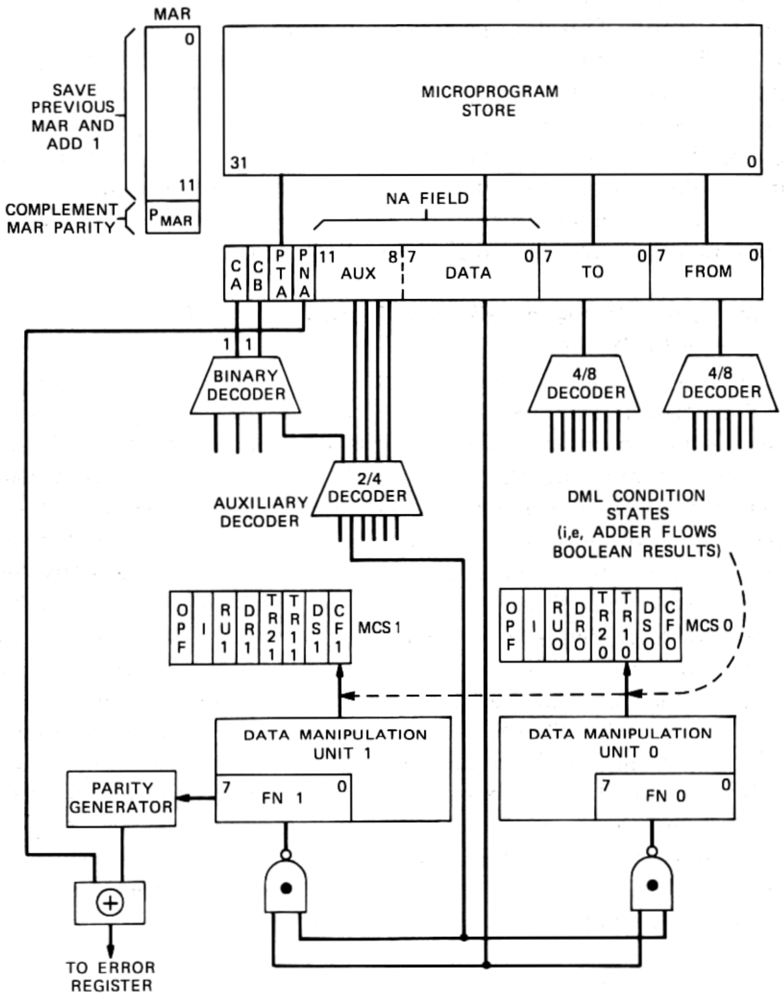


Fig. 7—Microstore-DML interface.

with respect to the microcontrol. That is, the appropriate register in the DML can be loaded, and at a predetermined number of microcyles later, depending upon the function to be performed, the results of DML operation can be gated to some destination register with the normal gating microinstruction.

To increase the efficiency of the DML and to reduce the amount of microcode needed to set up DML functions, the FN register and one of the operand buffers can be loaded in parallel. The operand register is loaded in the usual manner over the data bus with the TO and FROM control signals.

The FN register is loaded over a dedicated path directly from the NA field in the MIR. This data path is 8 bits wide. The remaining 4 bits of the NA field are used to enable a decoder which enables this gating path. To check that the data coming from the microstore are correct, a parity tree is attached to the output of one of the copies of the FN register (Fig. 7) and is checked against a parity bit (PTA) that is stored with this data constant in microstore. Note that the status outputs from each copy of the microcontrol are gated to associated duplicated copies of the microcontrol status register in the microcontrol.

4.4 Setting and clearing miscellaneous flip-flops and enabling dedicated gating paths

Scattered throughout the machine are a number of miscellaneous flip-flops that must be set and cleared under microprogram control. Because of the number of flip-flops, it is desirable to use a more efficient method of controlling them than dedicating a TO or FROM decoder crosspoint for each clear or set function. The 4-out-of-8 codes themselves generate only 70 possible combinations. By assigning, for example, only ten crosspoints from each of the TO and the FROM field decoders and using these two sets of 1-out-of-10 codes to drive a third decoder (designated the miscellaneous decoder), 100 miscellaneous crosspoints can be easily generated. In addition to setting or clearing flip-flops, the miscellaneous decoder outputs are used to enable dedicated gating paths. That is, in places where both a source and a destination do not have to be simultaneously defined for a gating operation, a miscellaneous decoder crosspoint can be used. This is advantageous for I/O interfaces where, for example, an external register can be gated to the processor with a miscellaneous decoder crosspoint. Note that this minimizes any timing restrictions between the processor's basic microcycle and any I/O timing requirements.

4.5 Control bits (CA, CB) and the auxiliary control decoder

In addition to the TO and FROM set, there is another set of control signals. The method of implementing this set of control functions is shown in Fig. 7. As indicated, there are 12 bits in the NA field. Two parity check bits (PNA, PTA) check the address sequencing of the microstore. The remaining two bits of the 32-bit readout of the microstore are control bits. They are encoded into four binary states which correspond to the following:

(*i*) The null state is used for the normal sequencing where no control function is required and the NA field is gated to the MAR.

(*ii*) The main-memory instruction fetch is used to initiate a new main-memory operation.

(*iii*) The data control is used to control the gating of the NA field to the MAR. This is for data operations when the NA field contains data to be gated to some destination register other than the MAR. The data control thus inhibits the normal sequencing and adds a 1 to the previous microstore address contained in the MAR.

(*iv*) The auxiliary control is used to enable an auxiliary decoder attached to the upper four bits of the NA field. This is a 2-out-of-4 decoder and, as such, has six possible control states, four of which are presently used.

(1) The first state enables the gating of the low 8 bits of the NA field directly into the function register in the DML. As a result, the function register can be loaded at the same time that the TO and FROM fields are loading one of the operand registers in the DML. Therefore, both time and microcode are saved on DML operations.

(2) The second state, I/O parity divert, checks the parity on incoming I/O messages. As outlined in Appendix A, incoming serial messages from the peripheral world are autonomously shifted into a serial channel buffer (IOD). Then a miscellaneous decoder instruction gates the IOD to register R11 over a dedicated path. After the data from the periphery are in the machine, they must be checked for correct parity. Note that if these data had bad parity and they were gated from R11 to any other register over the processor's gating bus (GB), a processor parity error would result. This would stop the processor and switch control to the standby processor. To avoid this condition, the output of the bus parity checker is diverted to a nonfatal error which causes an interrupt rather than an error. The control to divert this check is implemented by the auxiliary control decoder. Although its use is intended for unloading R11, it can be used to divert the parity check on any source register using the GB.

(3) The third state, I/O DML match divert, performs a similar function. Again referring to Appendix A, the operation of the I/O channel is such that a microcoded loop-around check is made of the correct loading of the IOD. As such, it is necessary to match R10, which was the source register to load the IOD, and R11, into which the IOD is returned. The following design implements this matching without adding a special hardware matcher. It uses a matcher attached to the outputs of the duplicated DML units.

The match on these two registers is performed by loading R10 into AR1 of DML1 and R11 into AR0 of DML0 and then gating the AR register onto the gating bus. As was the case for the parity check on these two registers, the normal DML match error represents a hardware fault within the machine. As a result, the machine stops, and a switch is performed to the good machine. Again, the state of the auxiliary control decoder diverts the fatal error to an error condition which is handled by an interrupt. In this way, even if a switch of machines is to take place, the software chooses the appropriate point in the processing to initiate the switch. This minimizes the information lost in an initialization procedure.

(4) The fourth state, disable GB parity checker, permits the microprogrammer to turn off parity checks on individual microinstructions. The hardware is such that parity checks are normally suspended for all microinstructions that do not use the GB. In addition, the use of this state in the auxiliary decoder allows turning off the checker when the GB is being used. This is especially valuable during maintenance programming. When it is known that a register has bad parity or when there is a question concerning parity, the register contents can be gated over GB to the DML, for instance, without causing a parity error. Once in the DML, the register parity can be checked, or it can be regenerated.

4.6 Main-memory control

The use of the microprogram control presents a number of possible alternatives in controlling the main store operation. As in the design of the microstore sequencing itself, a compromise between a design that would optimize the real-time capabilities of the memory operations and the economics of such an implementation was made. As such, the main-memory control is, for the most part, sequenced by the microcontrol. This implementation removes the complex sequencing logic, which not only reduces the hardware required but also eliminates circuitry that would be difficult to make self-checking. The control interface between the microprogram control and the main store is performed by a logic entity called the processor's bus controller (PBC). Microprogram control loads a register in the PBC [the main-memory state (MMS) register] with a control constant that defines the basic static mode of operation of the memory bus. For example, if writes are to be issued to both the on-line and off-line store, an appropriate bit in this control register is set. The microcontrol also loads an address register and a data register (if a write). With the static mode of the memory bus defined and the address and data (if necessary) loaded, the microcontrol can then initiate a main-store operation. The type of operation (i.e., instruction fetch, data fetch, or data write) is determined by the microinstruction that is used to initiate the main-store request. The microinstruction will set a request (REQ) flip-flop, set or clear the instruction or data (I.D) flip-flop, set or clear the read or write (R.W) flip-flop, and clear the main-memory cycle complete flip-flop [i.e., data ready (DR) flip-flop]. The I.D flip-flop determines to which register the accessed data are returned. The functions of the R.W flip-flop is obvious. Once the REQ flip-flop is set, the microcontrol can perform other functions. The main-store cycle will be performed concurrently, asynchronously, and autonomously to the processor. When the main store has completed the requested cycle, it will set the DR flip-flop. After the appropriate interval, the DR flip-flop is tested by the microcontrol, and the procedure is repeated.

The only major autonomous function built into the PBC is the ability to time-share the memory bus. This time-sharing capability allows the processor to be used in a multiprocessor configuration or in situations where a direct memory access device (DMA) time-shares the memory resources. This time-sharing capability consists of buffering the main-store request in the REQ flip-flop and testing the memory bus for its occupancy. If the bus is busy, the main-store cycle is delayed. When the bus is idle, the request is placed on the memory bus and the main-store cycle begins. The asynchronous nature of the processor and the main-store interface makes this design very straightforward.

Four registers are used to buffer address and data to and from the memory. These registers, together with a portion of the control that is used to operate them, are shown in Fig. 8. The operation of these registers as it relates to the microcontrol is outlined below.

4.7 Normal instruction fetch

An instruction is fetched from main memory by adding a 1 to the last instruction address located in the PA register. The control to initiate this fetch is performed by the CA, CB control decoder, as previously described. The active CA, CB combination is normally read out of the microstore on the first microcycle of the previously fetched OP code. Thus, if this OP code is, for example, an absolute transfer, the main memory control does not have to initiate the fetch for the next sequential instruction. If the next address is to be accessed, the CA, CB control enables the output of the PA + 1 logic to be gated into the store address register (SAR). It also sets two flip-flops in the memory

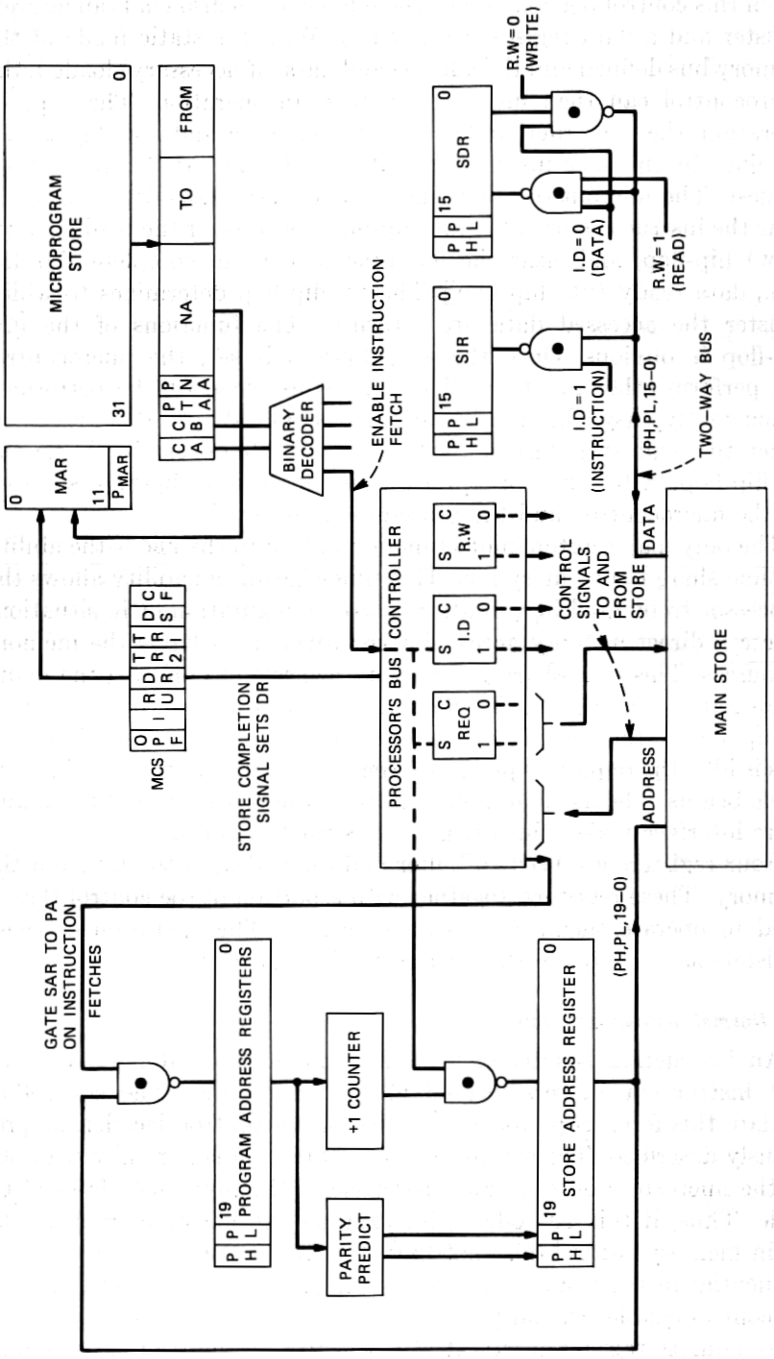


Fig. 8—Microprogram control-main-memory interface.

controller. The REQ flip-flop buffers the request for the store in case the bus is busy. The I.D flip-flop steers the return memory contents into the SIR or SDR. As a result of buffering the request, the microcontrol can continue executing the present OP code. When the memory bus becomes idle, the memory controller issues a request to the store. In addition, the controller gates the contents of SAR back around to the PA. This enables the PA + 1 counter to formulate the next address while the memory fetch is being performed. As a result, the design of the counter can be simplified. The counter is a slow ripple type that takes about 400 ns to increment. The counter is composed only of combination logic, which is attached to the output of the PA, because the PA and the SAR are bit-sliced. The resulting design provides a counter which can be easily checked by a parity predict circuit because of the partitioning which tends to force single-bit errors. It should be noted that most of the logic associated with the counter pertains to a given bit and the failure of even multiple gates within a circuit pack will cause an immediate parity error.

4.8 Data requests

When an OP code data requires a data operation, the microcontrol is used to load the data address in the SAR. If the data address must be calculated, the microcontrol uses the DML logic where additions may be performed. Once the data address is loaded in the SAR, a microinstruction crosspoint is used to initiate the store request. This memory request is again buffered in the request flip-flop, but the I.D flip-flop is put in the data state. Because the I.D is in the data state, the SAR-to-PA gating is inhibited so that the PA is preserved. The only distinction between data reads and data writes is that, for write instruction, the SDR is loaded with the data word before issuing the store request. The loading of the SDR sets the R.W flip-flop. This flip-flop is used by the bus controller and results in writing the contents of the SDR into main memory at the address defined by the SAR.

4.9 Central control panel operation

In keeping with the design goal of self-checking, each processor is assigned its own man-machine interface. This interface is called the central-control (CC) panel. The CC panel provides the ability to set and display registers in the processor, to read or write locations in main store, and to single-cycle macro-level instructions. In addition, the CC panel provides the ability to perform address-matching and data-matching functions on main-store programs. All of these functions are performed under microprogram control with only a few additional registers that provide buffer storage. The registers that are added for these panel functions are bit-sliced and incorporated into the basic self-checking architecture of the processor.

The microcontrol executes panel functions by receiving a panel interrupt when the processor is in the MANUAL mode and off-line. The interrupt begins by gating the contents of a set of three switch input registers into the display buffer (DB) (Fig. 2) with one of three possible microinstructions. The microcontrol then interrogates the switch inputs and translates them into the appropriate panel functions. Thus, the normally complex control functions of the CC panel are incorporated into the self-checking microprogramming structure of the machine, which results in a very flexible and relatively maintenance-free CC panel.

V. SEQUENCING

Several attributes characterize the design goals of the microcontrol sequencing logic.

(*i*) Self-checking. Since the machine is self-checking, the microcontrol sequencing must also be self-checking.

(*ii*) Flexible. To provide all the advantages of microprogram control, the sequencing scheme must be flexible. For example, the ability to conditionally transfer on a number of status bits provides an efficient means to loop or branch in the microcode. The flexibility in the sequencing logic also reduces the amount of store needed to perform a given task. Indexing, the use of the next address field for the auxiliary control, and the subroutine capability are examples of this.

(*iii*) Simplistic. To make the sequencing logic as fast as possible, as well as to make checking easier, it is necessary to keep the design simple and straightforward. This is accomplished by restricting the microstore addresses that can be incremented. When a 1 is added to the MAR, the MAR must contain a 0 in the low-order bit and, as a result, 1 can be jammed into that bit position to avoid the use of the inherently slow and/or complex carry-propagate circuitry. The same approach is taken for indexing by forcing the indexing table to fall on restricted boundaries.

With these characteristics as design criteria, each of the various methods for obtaining the next address (sequencing) will be examined individually.

5.1 Loading an OP code

As previously indicated, a fetched instruction at the completion of a main-store cycle is loaded into SIR. Simultaneously, the DR bit is set asynchronously to the microcontrol. While the main store is fetching the instruction, the microcontrol is operating on the previously fetched instruction. At the termination of the series of microinstructions that constitute this instruction, ALL ZEROs is encountered in the NA field of the microstore. ALL ZEROs is placed in the MIR NA field of the last microinstruction of all OP codes. A special all-zeros detector is used to monitor this condition. The coincidence of the all zeros and the DR bit set results in a new instruction or OP code being loaded into the microcontrol (Fig. 9). If the main store has not completed the instruction fetch (i.e., DR = 0), then the ALL ZEROs in the NA field is gated into the MAR. At the ALL ZEROs location in the microstore, ALL ZEROs is also placed in the NA field. As a result, the microstore will loop on ALL ZEROs waiting for the main store to complete the instruction fetch. Note that null (no operation) microinstructions (NOPs) are placed in the TO and FROM fields of the ALL ZEROs location. These NOPs are valid 4-out-of-8 codes required to keep the 4-out-of-8 checks "happy." If the main store is ready when the last microinstruction of the previous OP code is read out of the microstore, the next OP code will be loaded immediately. As a result, microinstruction NOPs will not be executed between main-memory instructions.

When an OP code is loaded into the MAR from the SIR, it is also gated into the IB and the RAR. The IB is loaded with the complete contents of the SIR. Although the OP code is loaded into the IB, it is not used there. This feature was implemented to preserve bit-slicing. In addition to loading the OP code into the IB, the branch allowed (BA) bit is also loaded into the IB (Fig. 3). This bit must equal a 1 for target addresses on branch instructions. An OP code that calls for a branch performs a microinstruction that sets a BA check bit which in turn enables a check for the BA = 1 on the next instruction fetch.

5.2 Normal sequencing

Each time a word is read out of the microstore on a normal microinstruction, an address is read into the NA field. After being gated to MAR, this address points to the next microinstruction in that sequence. Thus, loading an OP code into MAR initiates a sequence of microinstructions that is programmed to perform that OP code. The last microinstruction of that sequence contains all zeros in its NA field. Consequently, a new OP code is loaded into the MAR, and the process is repeated. The inherent simplicity of the sequencing scheme permits a simple parity check code to verify the proper operation.

Each time a word is read out of the microstore, two parity check bits are included. One check bit (PTA) is matched against the parity of the address that accessed the word (see Fig. 3). The other parity bit (PNA) is associated with the 12-bit NA field and is gated into the MAR

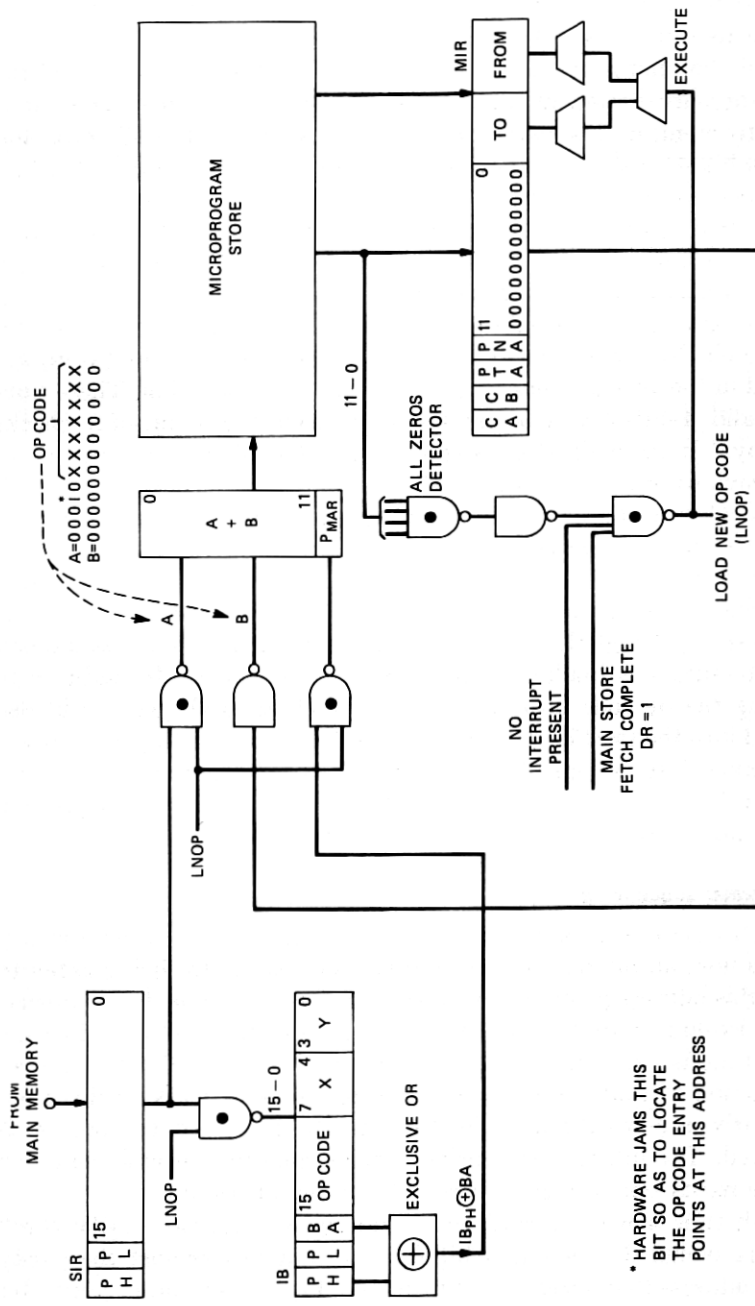


Fig. 9—Loading a new OP code.

parity (PMAR) to check the next word to be accessed. Two control bits, CA and CB, are also included in the parity check of the accessing.

5.3 Data

It is useful to be able to store data constants in the microstore so that they can be easily and quickly generated by the microcontrol. For example, the machine has 16 hardware interrupt levels. Using microprogram control and the data facility, these interrupt levels can be translated from a bit position in the interrupt set (IS) register into address locations in main memory where the appropriate software can implement each one of the individual interrupts. The microcontrol tests for the bit position and then provides access to the word in micromemory that corresponds to this bit. Residing in this word is a data constant which points to a main-memory location. Because the contents of micromemory are changeable, the implementation is a flexible one.

When data are read out of the microstore, they are contained in the NA field. As shown in Fig. 10, the data can be gated on either the high 12 or the low 8 bits of the GB. The FROM decoder determines which bits are used, and the TO decoder selects an arbitrary destination register.

Since the NA field contains data, the next address must be obtained from another source. In this instance, it is generated by saving the last address and adding a 1 to it. As previously outlined, data words are forced to be on even word boundaries. As a result, a 1 is jammed into bit 0 to implement an add. In addition, the parity for this data address can be easily predetermined. It is formed by complementing the present parity bit PMAR in MAR. The CA and CB control bits save the contents of MAR, add a 1 to it, and complement PMAR.

5.4 Auxiliary control

The sequencing of the next address for auxiliary control is identical to the data operation just outlined. In this instance, the NA field contains control information and, in the case of the loading the FN register, data as well. Thus, the next address is obtained by adding a 1 to the address saved in MAR. As with data, the auxiliary control functions must occur on even word boundaries. The CA bit equals 1 for both data and auxiliary control operations. This bit is used as the control to save the MAR and to jam a 1 into it. In checking the operation of the CA, CB control decoder, the data and auxiliary control decoder leads are used to complement the MAR parity bit.

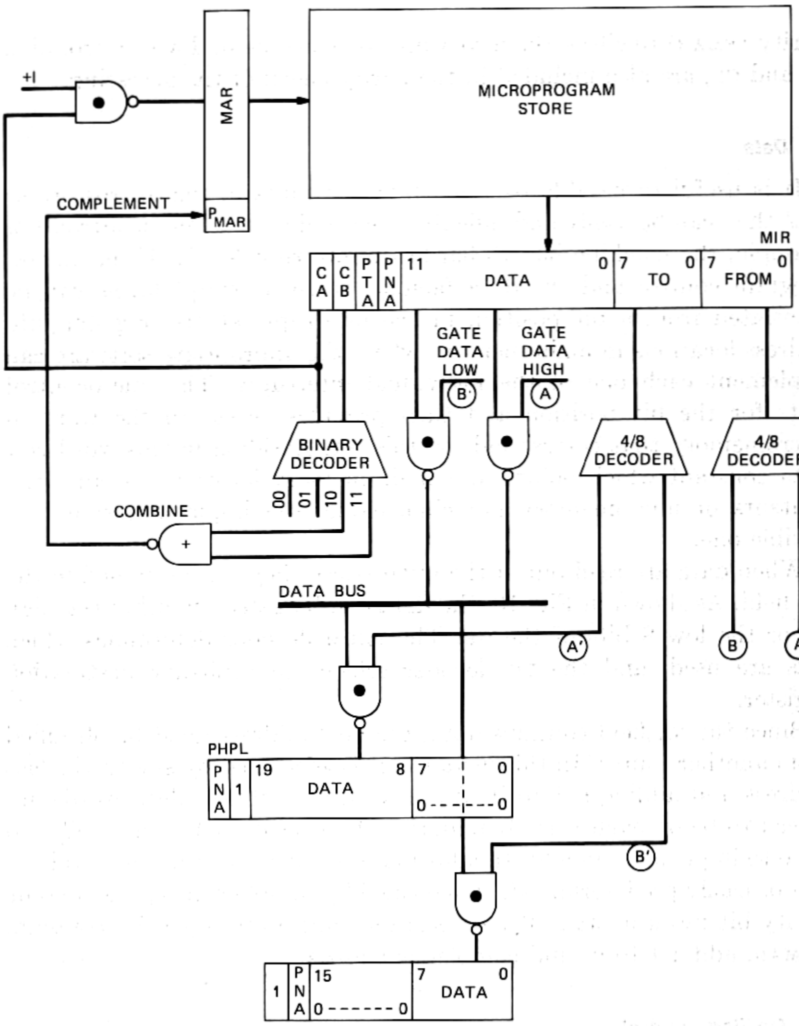


Fig. 10—Data command.

5.5 *Microsubroutine*

To describe the function of a microsubroutine, it is necessary to outline the normal operation of the return address register (RAR). Each time a word is read out of the microstore, the NA field is gated into the RAR. As outlined in Appendix B, this provides an additional check on the address sequencing at a very minimum cost. When a subroutine is to be entered, a data command is used (see Fig. 11). This data command contains the return address in its NA field, which is then gated both into the MIR and the RAR. Contained in this same

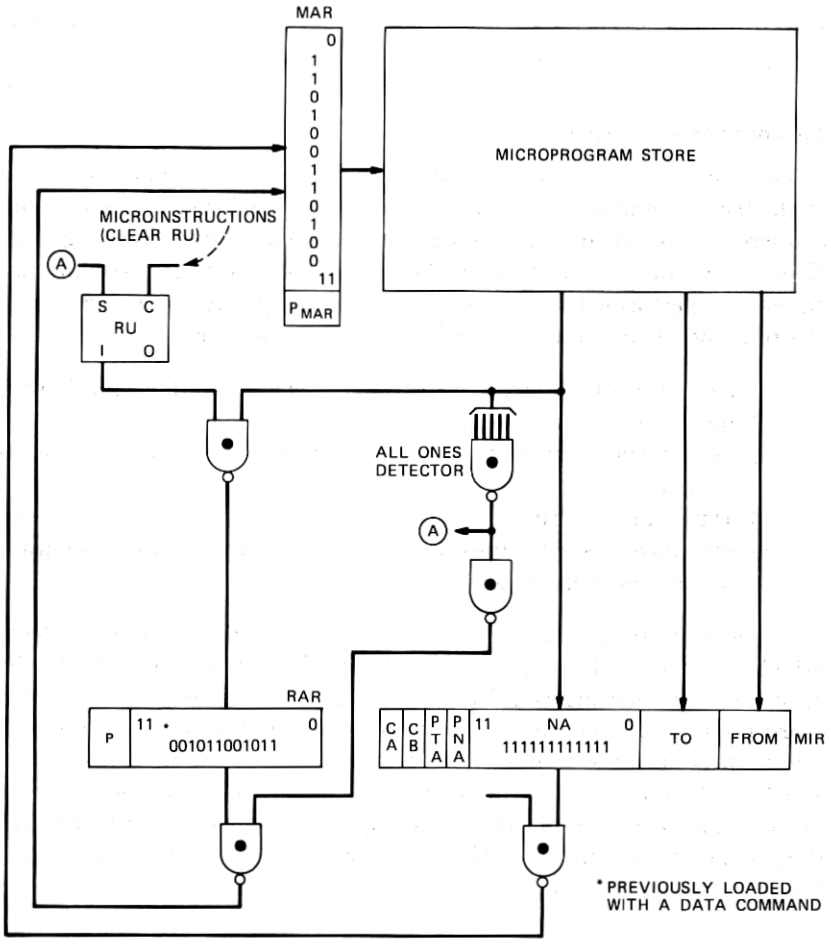


Fig. 11—Subroutine return.

command is a microinstruction which clears the RAR update (RU) flip-flop. This flip-flop saves the contents of the RAR and therefore inhibits gating into the RAR on the subsequent microcycles. As with normal data operations, the first address for the subroutine is obtained by jamming a 1 into the old contents of the MAR. The subroutine continues to sequence the microstore in the normal manner. Except for calling another subroutine, all sequencing operations of the microcontrol can be performed in the subroutine. On the last microinstruction of the subroutine, ALL-ONES are placed in the NA field. An ALL-ONES detector placed on the output of the NA field results in the return address in RAR being gated to the MAR, and the subroutine is excited.

In addition, the ALL-ONES detector sets the RU flip-flop so that it is again in the update mode.

5.6 Conditional branches

The ability to conditionally branch within a microsequence is one of the basic operations of the microcontrol. To facilitate this operation, a microcontrol status (MCS) register was implemented (see Fig. 12). Each of the bits of the MCS can be individually tested: a conditional transfer is performed as a function of their respective states. Some of the bits and their primary function in this register are:

(*i*) DS—Stores the results of DML operations (i.e., adder overflows).
(*ii*) DR—Indicates the completion of main store cycles.
(*iii*) TR1—General-purpose status bit intended for use by microcontrol.
(*iv*) TR2—Same as TR1.
(*v*) CF—Passes status information between macro-level program sequences (i.e., condition flip-flop).

On a conditional branch instruction, a microinstruction selects which MCS bit is to be tested. The state of this bit is then gated into the MAR bit 0. By forcing conditional branch instructions to fall on even word boundaries, the implementation is simple. The NA field from the MIR is gated into the MAR in the normal manner. If the MCS bit is a one, the branch is made to address $X + 1$, and if not, the address X is chosen. The MCS is duplicated. One copy of the MCS feeds the MAR bit 0, and the other copy feeds the MAR parity bit which, if the branch is taken, is complemented.

5.7 Indexing

Indexing is used to permit the microprogram to easily branch into blocks of microstore so that operations like binary to m-out-of-n code conversions can be easily performed. The index operation results in either of the two 4-bit binary fields X or Y in the IB being ORed into the lower four bits of the MAR, as shown in Fig. 13. Again, to provide simplification, indexing tables are forced to start on 16-word boundaries. The implementation is then analogous to the conditional branch. A microinstruction selects either X or Y and jams it into the low four bits of the MAR, which are guaranteed to be zero. The parity for this address is generated by using the parity trees attached to the X and Y fields. If X_P or Y_P is odd, the PNA bit is complemented when it is gated to the MAR register (PMAR).

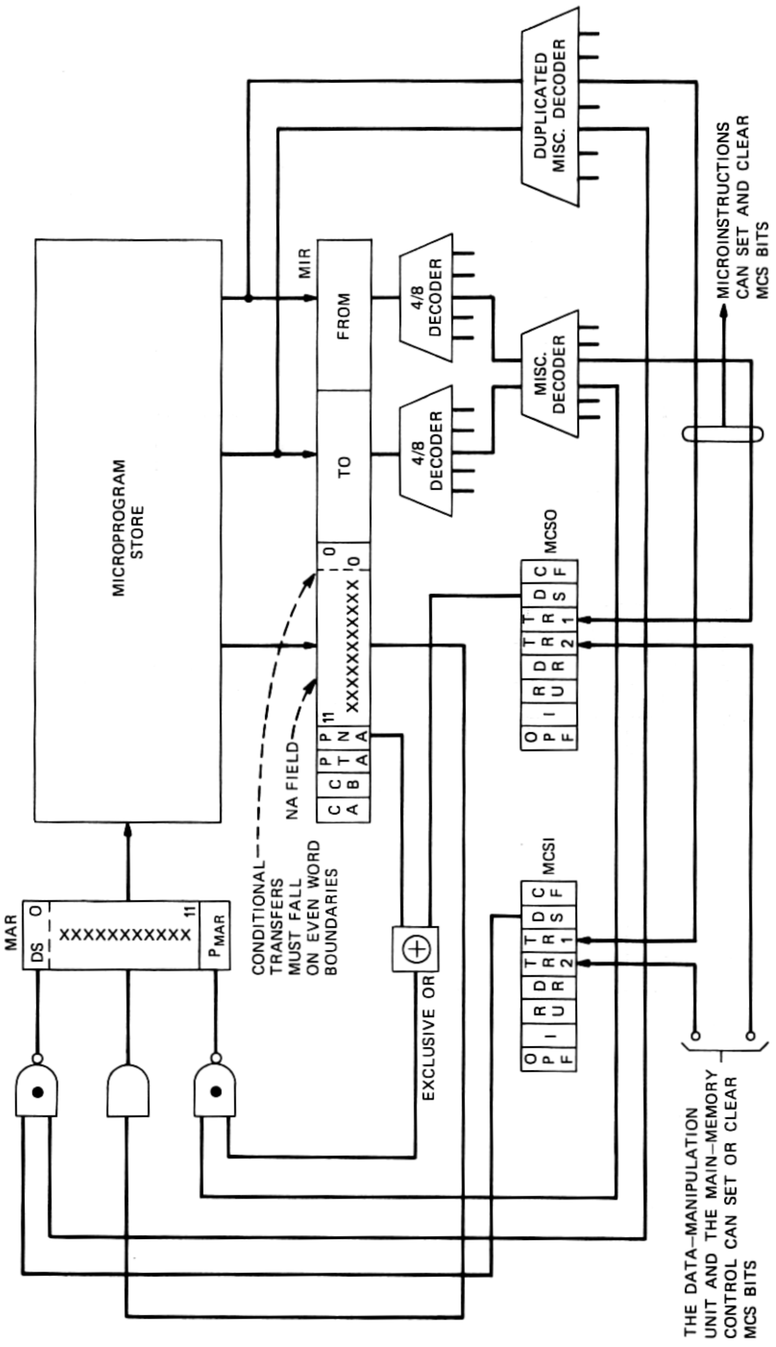

MAR
DS O XXXXXXXXXXX 11 P_MAR
MICROPROGRAM STORE
MIR
C A
C B
P T A
P N A
11 XXXXXXXXXXX 0 0
TO
FROM
NA FIELD
CONDITIONAL TRANSFERS MUST FALL ON EVEN WORD BOUNDARIES
EXCLUSIVE OR
4/8 DECODER
4/8 DECODER
MISC. DECODER
DUPLICATED MISC. DECODER
MCSO
MCSI
O P F
P I
R U
D R
T R 2
T R 1
D S
C F
MICROINSTRUCTIONS CAN SET AND CLEAR MCS BITS
THE DATA–MANIPULATION UNIT AND THE MAIN–MEMORY CONTROL CAN SET OR CLEAR MCS BITS

Fig. 12—Conditional transfers.

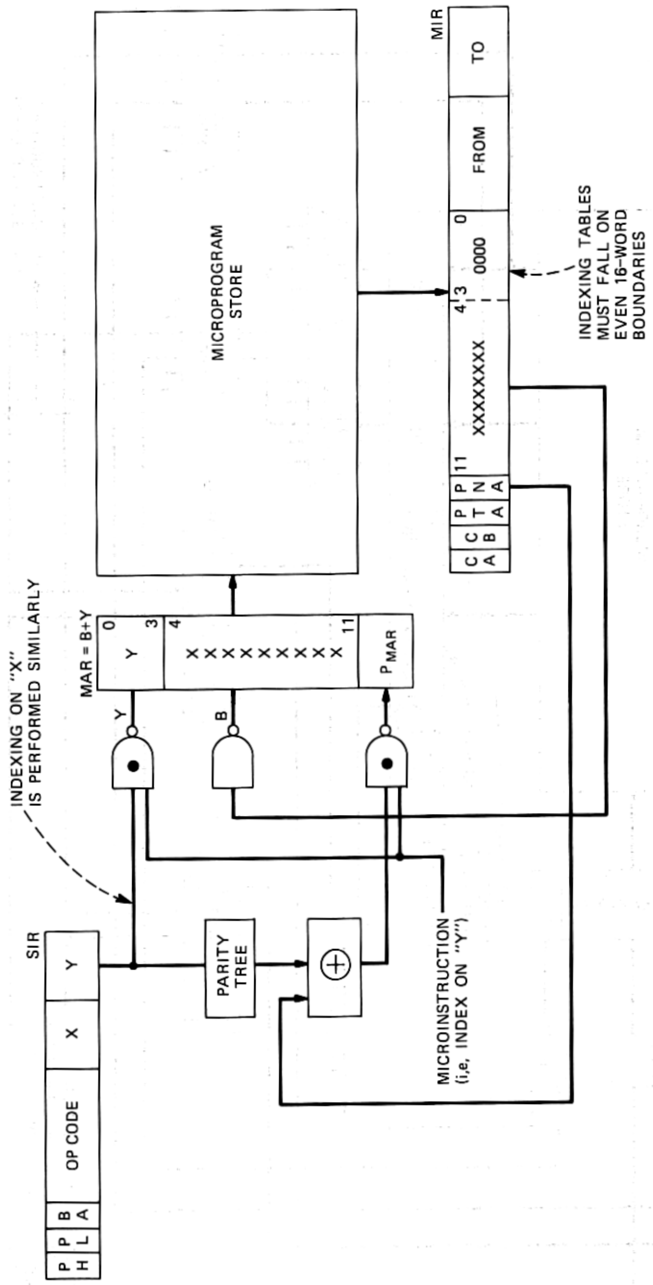

INDEXING ON "X" IS PERFORMED SIMILARLY
MAR = B+Y
0
3
4
11
Y
XXXXXXXX
P MAR
MICROPROGRAM STORE
MIR
TO
FROM
0000
XXXXXXXX
C A
C B
P T A
P N A
INDEXING TABLES MUST FALL ON EVEN 16-WORD BOUNDARIES
SIR
P H
P L
B A
OP CODE
X
PARITY TREE
B
MICROINSTRUCTION (i,e, INDEX ON "Y")

Fig. 13—Microstore indexing.

5.8 Interrupts

Interrupts are buffered in the IS register, which contains 16 bits; 16 different levels of interrupts are possible. An interrupt mask (IM) register is also provided. This register can selectively block any one of the 16 interrupts. When an interrupt enters the IS register and is not masked by the IM register, this condition is monitored by the microprogram control. This monitoring or testing is performed at the end of each microinstruction sequence when ALL ZEROS is read out of the microstore.

Before a new OP code is loaded into the MAR, the state of the interrupt lead is checked. If an interrupt is present, whether a main-memory fetch is completed (DR = 1) or not, an interrupt address constant is hard-wired jammed into the MAR. At this address in the microcontrol, an interrupt microroutine is initiated. It tests which of the 16 interrupt levels is present and transfers the control to the appropriate program in main memory. This transfer consists of translating a bit position in the IS to a data constant in the microstore that points to the main-memory program that handles the interrupt. Before control is passed to software in the main store, the interrupt microcode saves critical registers and states of the processor in a save area in main memory. Thus, when control is returned from the interrupt, the processor can be returned to its original state.

If real time is critical, high usage or frequently called interrupts can be handled entirely with microprogram control. Because of the 8-to-1 speedup of microcycles versus main-memory cycles, this capability represents a very powerful feature. The only change is an increase in the amount of microcode used.

A block interrupts (BIN) flip-flop inhibits the interrupt mechanism. BIN inhibits additional interrupts from being serviced before the interrupt-handling software has recorded the presence of the original interrupt (Fig. 14).

5.9 Maintenance reset functions (MRF)

Several conditions require the processor to be initialized. The source of the initialization may vary from a processor error, where the processor is on-line, to turning on the power in the off-line. All these varied conditions are funneled into a state which results in a hard-wired address being jammed into the MAR. In addition, a few processor state flip-flops must be initialized to ensure that the machine will start up correctly. For example, the clock must be initialized before the microcontrol will sequence. At the MRF address, a microroutine (bootstrap sequence) performs all the complex decisions concerning what caused the initialization and what actions are to be taken. The ability to reduce

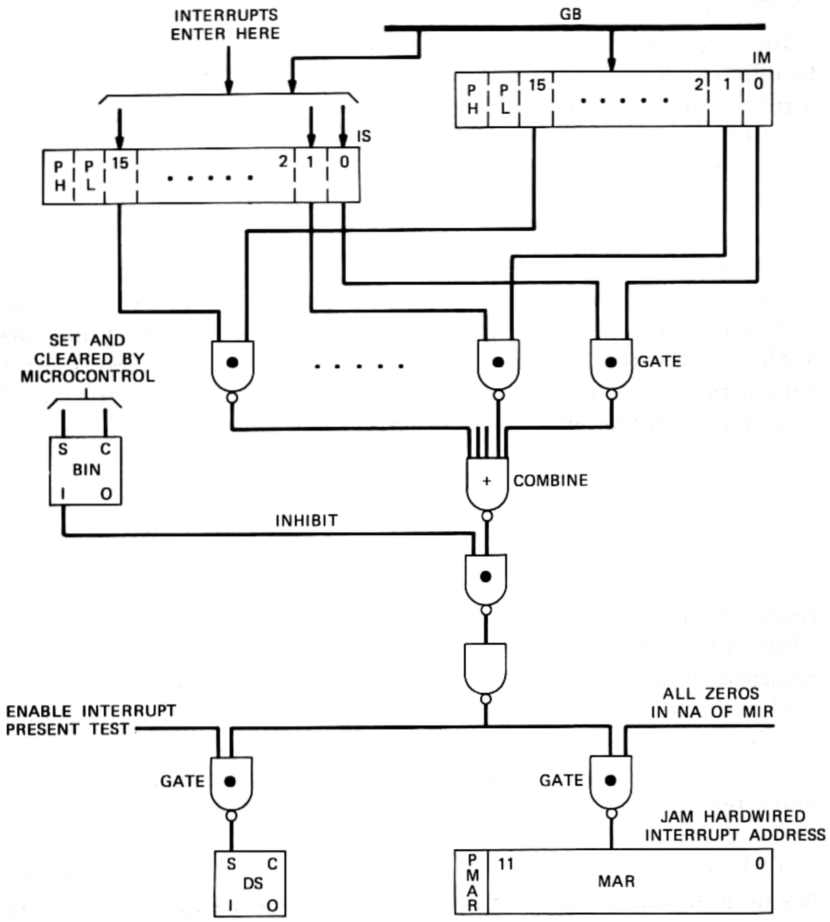


Fig. 14—Interrupt hardware.

the circuitry required to initialize the processor to a flip-flop, a few gates, and a couple of clock phases is a very significant advantage provided by microcontrol. In addition, the use of the bootstrap microsequence provides the ability to implement a very versatile initialization start-up procedure. For example, the bootstrap sequence may be changed to suit the application, such as using a disk instead of a tape unit for backup storage.

VI. MAINTENANCE OF THE MICROCONTROL

The microstore for the standard processor consists of about 1000 words 32 bits each, and it grows in 512-word blocks up to 4096 words. Because of the size and the different applications containing dissimilar

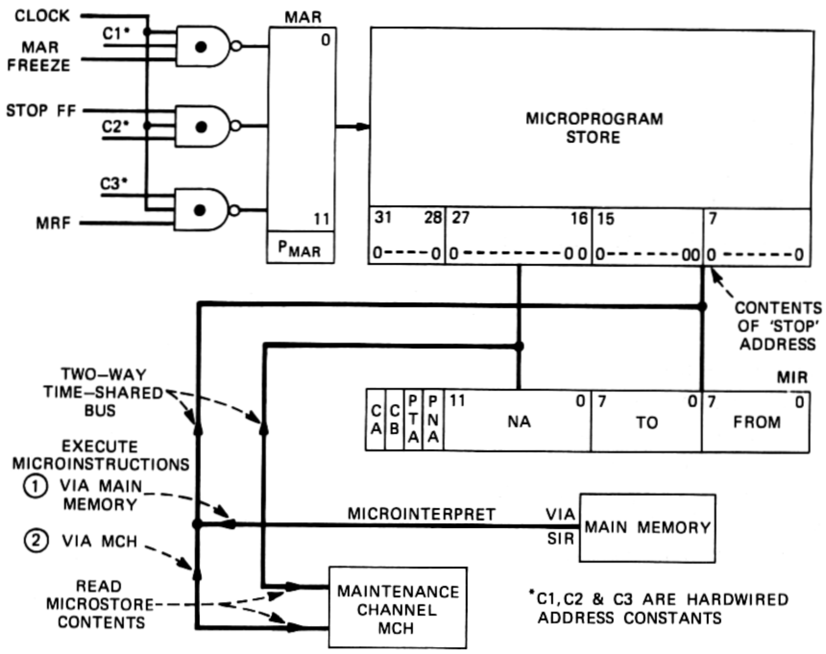


Fig. 15—Maintenance access to microcontrol.

contents, it is necessary to provide maintenance access to verify the operation and contents of the microstore. It is also quite useful to be able to exercise the machine without using the microstore itself. The most useful tool for maintenance access to the processor is via the maintenance channel (Fig. 15). The maintenance channel inhibits the microcontrol and loads microinstruction directly into the MIR. This permits the on-line processor to perform maintenance operations on the off-line processor. The feature also allows microinstructions stored in the on-line main memory to be executed in the off-line processor. This ability to access and control a unit at its most elementary level of control allows a very high degree of diagnostic access. In addition to executing microinstructions, the maintenance channel can load the MAR with an arbitrary address and in turn read the contents of that microstore location. Thus, an image of the microstore contents kept in main store can be matched against the contents of the off-line processor.

6.1 Maintenance-channel access

Before the maintenance channel can gain access to the processor, it is necessary to stop the microsequencing. Also, if a processor error is detected, that processor must be stopped so that it does not attempt to interfere with the healthy processor, which is then switched on-line. To implement this, a STOP flip-flop is used. Setting the STOP flip-flop results in a hard-wired address being jammed and held in the MAR. This prevents the microstore from sequencing until the STOP is cleared. At the STOP address in the microstore, all zeros are contained in the 32 bits. The effect of this is to remove the microstore from the input gating to the MIR. This allows the output of the maintenance channel to be ORed directly onto the output of the microstore. As a result, no additional control signals or gates are needed. The data that are ORed onto these leads, using a NAND gate collector-tie, are clocked into the MIR in the normal manner.

The maintenance channel gains access to the standby processor by setting the STOP flip-flop. With the STOP flip-flop set, the maintenance channel has access to the MIR. As such, it has almost complete control to exercise the processor since all microinstructions emanate from the MIR.

The description and operation of the maintenance channel are covered in other material,[7] but two of its more important functions and one that is implemented for the most part within the microcontrol itself are covered here.

6.2 Single-cycling a microinstruction

With the processor held in the stopped state (STOP FF = 1), all microsequencing ceases, and zeros are read out of the microstore. The stopped state, however, does not inhibit the processor's clock. Consequently, on each microcycle, the data presented to the input of the MIR are clocked into that register. For the stopped state, these data are all zeros, but no errors are registered by the error register because the output of the STOP flip-flop inhibits the 4-out-of-8 checkers on the TO and FROM fields, as well as other checks that are normally performed on each cycle of the microcontrol. Thus, the maintenance channel needs only to gate data onto the inputs of the MIR TO and FROM fields for the duration of the clock phase, which loads the MIR. The result is the execution of a single microinstruction, which uses all of the normal timing and hardware within the machine. To turn on the decoder checkers, the maintenance channel also needs to activate a control lead to the TO and FROM checkers that will override the STOP flip-flop inhibit function of the checker for that single microinstruction.

6.3 Freeze and read microstore

The ability to read the microstore is provided by the freeze state. As has been indicated for single cycling, the input to the MIR represents the key to gaining access to the microcontrol and to the processor. To execute a microinstruction, it is necessary to load the low 16 bits of the MIR. To access a particular word in the microstore, the upper 16 bits of the 32-bit register are loaded. Similar to loading a microinstruction, the same type of operation of the maintenance channel is required for loading the upper 16 bits (NA field). Once in the MIR NA field, the microstore address is gated up to the MAR by the normal clock timing. At this point, the processor uses a control signal from the maintenance channel to set the freeze flip-flop and also to clear the STOP flip-flop. Note that the STOP flip-flop must be cleared because, in its set state, it jams a hard-wired address into the MAR. When set, the freeze flip-flop inhibits further clearing of the MAR after the register contains the address loaded from the maintenance channel via the upper half of the MIR.

One of the added benefits of this implementation is that the inhibiting circuitry on the MAR is already provided. As previously described, the data command functions by inhibiting the clearing of the MAR and adding a 1 to it to obtain its next address. Thus, only a single flip-flop is required to buffer the freeze control signal from the maintenance channel. The output of this freeze flip-flop is ORed into the inhibit circuitry already implemented on the MAR.

Once an address has been loaded and frozen in the MAR, the contents of this address are presented on the outputs of microstore on a dc basis. The maintenance channel can now read the upper and lower halves in succession and send the response back to the controlling source (i.e., the other processor). If another word is to be read out of the microstore, the maintenance channel must first put the processor back into the stopped state so that the output of microstore will return to all-zeros state, allowing the ORing into the input of the MIR again.

6.4 Start microcontrol sequencing

When an arbitrary address has been frozen in the MAR, the maintenance channel can easily implement a start (beginning at this address) of the normal microsequencing by clearing the freeze flip-flop. The ability to start the microcontrol at an arbitrary address gives the maintenance programmer added flexibility, but the primary source of initiating a microcontrol startup is via the MRF hardware. The advantage of using MRF hardware is that it jams a number of key flip-flops in the machine to a predetermined state and then transfers control to the microcode by use of a hard-wired constant into the MAR. The result is an MRF sequence that can start from an unknown state and go first to a known state determined by initializing flip-flops and then to a running condition using minimal hardware. This MRF sequence is invoked by the maintenance channel when a fatal or serious error is detected in the on-line processor. When this error is detected, the on-line processor stops. In turn, maintenance initiates a switch message which results in an MRF or start-up to the off-line processor.

6.5 Microinterpret

Microinterpret allows microinstructions to be stored in main memory of the on-line processor to be executed by the microcontrol of that processor (Fig. 16). One of the advantages of microinterpret is that maintenance instructions (being nonreal-time critical) can be performed without using microinstruction sequences stored in the costly microstore. The maintenance programmer, therefore, has the full use of the microcontrol and yet has the freedom to write microinstruction in a manner best suited to his needs. Not only does the microinterpret reduce microstore requirements, but, as described later, its implementation is such that a very minimal amount of additional hardware is needed to design it into the processor's architecture.

The microinterpret mode essentially allows the enhancement of the instruction set in the on-line processor to perform maintenance-oriented or seldom-needed functions at a minimal cost. The maintenance channel, on the other hand, provides the diagnostic capability to detect and locate troubles in the off-line processor as well as to monitor the general state of affairs in that off-line processor.

The initiation of the microinterpret mode is performed by one of two macro-level OP codes which are executed in the normal manner by the microcontrol. One OP code indicates a single-cycle microinterpret; the other indicates a multiple-cycle microinterpret instruction. The microsequence of each OP code first sends the main memory on for the next instruction located at PA + 1. This fetch request is performed in the usual manner by the CA, CB bit combination set to the fetch state. At this PA + 1 address, a microinstruction is stored (i.e., two 4-out-of-8 codes instead of an OP code). The next step of the microcode is to set or clear the general-purpose microcontrol test bit, TR1. The microinterpret OP code determines whether TR1 is set or cleared. If only a single microinterpret instruction is to be fetched from main memory, the TR1 bit is set.

The third and last function of the microinterpret OP codes results in setting the microinterpret (MINT) flip-flop. Setting the MINT flip-flop enables the next word fetched from main memory to be gated directly from the SIR to the MIR TO and FROM fields. As shown in Fig. 16, this gating is also conditioned on the NA = ALL ZEROS and the DR bit = 1, exactly analogous to the loading of a new OP code. As a result, the microstore loops on the all-zeros address until the main memory has

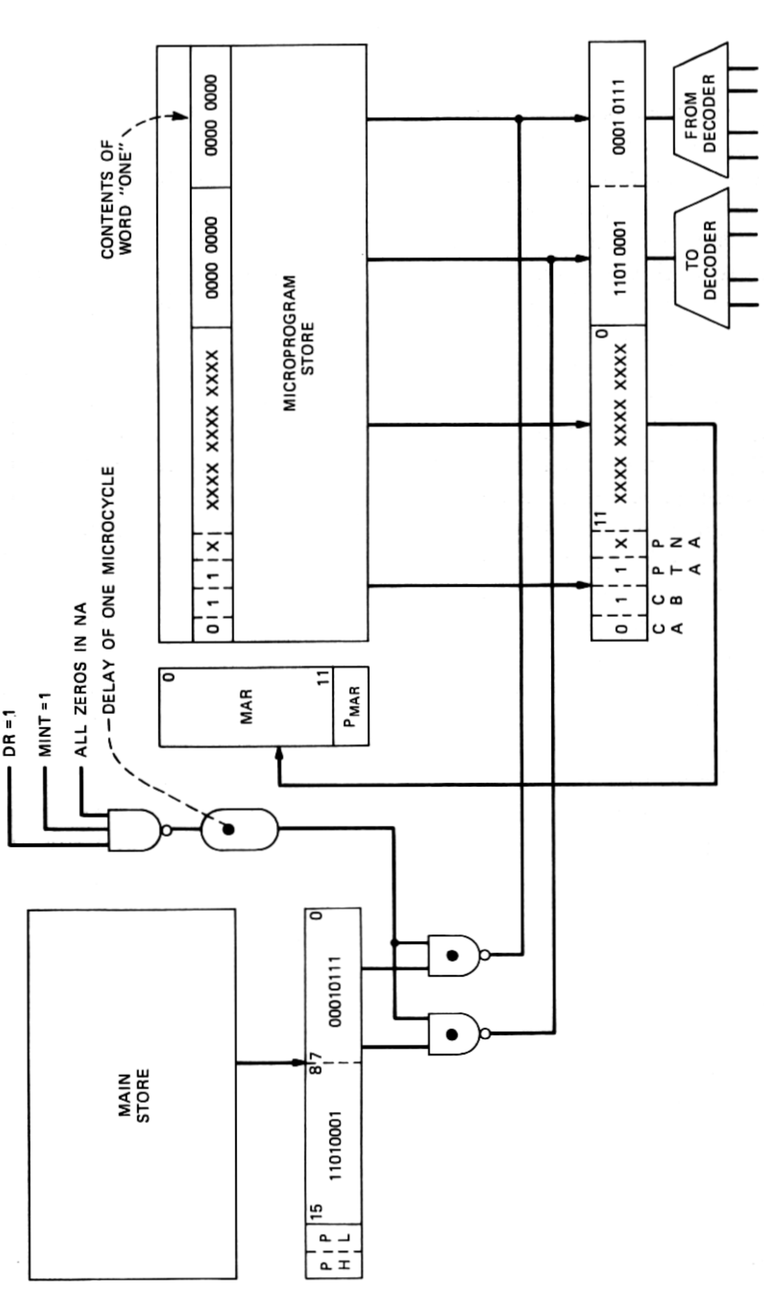


Fig. 16—Microinterpret implementation.

completed the fetch for the microinterpret instruction. When the store has indicated a completion, a 1 is jammed into the MAR, and the gating from the SIR to the MIR is initiated (see next paragraph). The next address for the microcontrol is then at location "1" where the start of the special microsequence handles the microinterpret operation. At word "1," the CA, CB bits are again in the fetch state. They result in the PA + 1 being loaded in the SAR so that the next instruction will be fetched. This instruction can either be a microinterpret instruction or a normal OP code which is determined by the TR1 bit. The microcontrol tests this bit and, if it is set to the single-cycle microinterpret, a microinstruction clears the MINT flip-flop. When this flip-flop is cleared, the microcontrol is returned to its normal state and loops on all zeros. At the completion of the store request in progress, the contents of the SIR are loaded to the MAR to begin the next OP code cycle. If TR1 $\neq$ 1, then a multiple-cycle microinterpret is assumed. Each time the main memory fetches a new microinstruction, the contents of the SIR are gated to the MIR and the cycle is repeated. The termination of the multiple-cycle microinterpret is indicated by clearing the MINT flip-flop. This is accomplished by having the last microinstruction of a microinterpret multiple sequence clear it.

When a microinstruction is performed by microinterpret, the two 4-out-of-8 codes representing that instruction are gated from the SIR to the MIR TO and FROM fields. This gating takes place only once. It is coincident with the reading of the contents of the microstore word located at address "1." Address "1" is unique in that it contains all zeros in its TO and FROM fields. As a result, the contents of SIR can be gated into the MIR TO and FROM fields. In this way, a gating signal need only be applied to the SIR, and no new circuitry or critical timing is required to gate the control fields into the MIR itself, as shown in Fig. 16. Note that a delay of 1 microcycle is required from the time the completion signal is given (i.e., DR = 1) to the time the gating is enabled from the SIR to the MIR. The all-zeros location cannot contain all zeros in its TO and FROM fields since the ALL ZEROs is used as a looping address to await main-memory completions. All zeros in the TO and FROM fields would cause the 4-out-of-8 checker to indicate an error, stopping the processor.

The operation of the microinterpret instructions is also self-checking. It must sequence properly and provide valid 4-out-of-8 codes to the TO and FROM fields or the already-described check circuits will fire.

6.6 Maintainability

One of the most beneficial results of microprogram control design from a maintenance aspect is the absence of complex timing circuitry.

The clock consists of only an oscillator and a simple circuit to generate four clock phases. In addition, these clock phases are used almost entirely within the microcontrol itself. Use within the microcontrol is limited for the most part to gating or strobing data and control information into the MAR, MIR, and RAR registers. The loss of these gating pulses will typically cause the contents of the affected register cells to be stuck in a "1" or "0" state, resulting in immediate fault conditions in the check circuit monitoring these registers and their outputs. This is contrasted to a conventional machine that uses complex timing and clock pulses to avoid race conditions and the like in such areas as the command decoder. In addition, in a conventional machine, the clock is typically distributed throughout the entire processor, making fault analysis difficult. If a fault does occur in a clock phase that is designed to eliminate spikes or race conditions, the problem of fault diagnosis becomes very difficult. A fault of this kind is hard to reproduce consistently and may elude the diagnostic programmer because of its possible transient nature. In addition to the simplified clocking scheme, the inherent regular structure of a microprogram control machine lends itself not only to a self-checking philosophy but also to the diagnosis of the fault.

VII. LOGIC IMPLEMENTATION OF THE MICROCONTROL

The microcontrol is contained on eleven 1A-type logic-circuit packs. A total of 2948 gates are used to implement the microcontrol. Of these, 1158 gates are used in the microinstruction decoders and the check circuits. The remaining 1790 gates are used to design the sequencing logic and its check circuits. Approximately 30 percent of the gates in both the decoders and sequencing logic are associated with check logic. The 2948 gates used in the microcontrol represent approximately 20 percent of the total gates used in the processor.

The standard processor will use approximately one-quarter of the microstore's maximum address space (4K* 32-bit words). These 1K words are used as follows:

Function	Words
Implementation of the OP code set	560
Central control (man-machine interface) panel function	200
Initialization sequencing	75
Initial program load from bulk storage	125
Interrupt handling function	60

* $K \cong 1024$.

VIII. ADVANTAGES OF THE MICROPROGRAM CONTROL DESIGN

The use of microprogram control in this processor provides the following advantages:

(*i*) A uniform processor architecture. This uniformity is very amenable to the self-checking design incorporated into the processor complex.

(*ii*) An easily maintainable processor. The microprogram control design allows external access via a maintenance channel with a minimal amount of circuitry. Access at the most elementary level of control of the processor provides diagnostic access to the entire machine.

(*iii*) A very flexible design. This flexibility is present in many aspects of the processor design. Some of this flexibility is the capability to easily change control features; some of it is the ease in which complex control sequences are incorporated into the micro-control itself. For example, there are

(1) An easily changed macro-level instruction set.
(2) Speed-independent interfaces to main memory and to peripheral units.
(3) An extensive and complete interrupt structure that can be adapted to each application.
(4) A complex and yet versatile initialization procedure.
(5) A powerful and easily maintainable console panel.

(*iv*) Easy integration into an ESS environment to provide a system that can immediately detect faults, recover quickly from them, and then provide the necessary diagnostic access and repair.

IX. ACKNOWLEDGMENTS

The author wishes to acknowledge the contribution of his fellow workers. In developing any successful system it takes the cooperative effort of many people. In particular, W. N. Toy and E. N. Nussbaum have provided the leadership in the initial processor design work and fellow workers R. W. Cook, J. S. Kent, R. E. Libman, R. A. Meister, W. H. Sisson, and many others have contributed to the design. In addition, E. A. Erland, L. A. Weber, and J. G. Whitemyer have guided the project to a successful conclusion.

APPENDIX A

I/O and Its Microcontrol Interface

The interface between the processor and its periphery is primarily performed by serial I/O channels. The interface between the microcontrol and the autonomous circuitry that controls the I/O sequencing is described in the subsequent paragraphs.

The I/O control for the processor consists of expandable, semi-autonomous, functional units called I/O main channels. The architecture of the processor provides the ability to add as many as 20 I/O main channels. Figure 17 shows a single I/O main channel and the major data paths that connect it to the processor. A favorable attribute of the interconnections between the processor and the I/O main channels is the relatively loose coupling between them. Within the processor, three general registers, R9, R10, and R11, provide interface with I/O channels. R9 and R10 send control and data, respectively, to the channels. R11 receives data from the channels. The direct outputs of registers R9 and R10 are presented directly to the I/O channels.

The channel select (CS) field of R9 selects which one of the 20 main channels is to be enabled. The decoding of this 3-out-of-6 channel select results in gate CS becoming enabled. Once a main channel is enabled, microinstruction enables the low-order 12 bits of R9 to be gated into the selected I/O status (IOS) register. Similarly, another microinstruction enables R10 to be gated into the I/O data (IOD) register. The advantage of such a gating scheme is that the timing problem in the interface is greatly simplified. That is, R9 and R10 can be loaded with a normal register-to-register gating instruction and on subsequent microcycles, microinstructions can be used to gate the contents of these registers in the selected I/O main channel. In this way, as soon as R9 and R10 are loaded, the data ripples out to the designated I/O channel. Then, when the microinstruction that gates these registers into the channel becomes active, the data are stable at the input to the designation point. In an analogous way, the output of the IOD of the selected channel is gated into R11 with a miscellaneous decoder crosspoint. Elimination of the timing problem also means that the variability of the fanout, seen when I/O main channels are added, does not pose a problem. The flexibility afforded by this gating scheme and the use of microprogram control also facilitate the design of completely different I/O interfaces, such as a parallel-to-parallel interface rather than the standard parallel-to-serial.

The actual operation of the channel consists of loading the IOS and the IOD and then issuing a microinstruction crosspoint that starts the autonomous sequencing. It should first be noted that each main channel consists of 20 subchannels which provide fanout to the peripheral units and time-share the control and sequencing logic of the main channel. The sequencing consists of shifting the data loaded into the IOD, together with a start code that is prefixed onto the front of the message, and into a serial data link. The serial message is transmitted

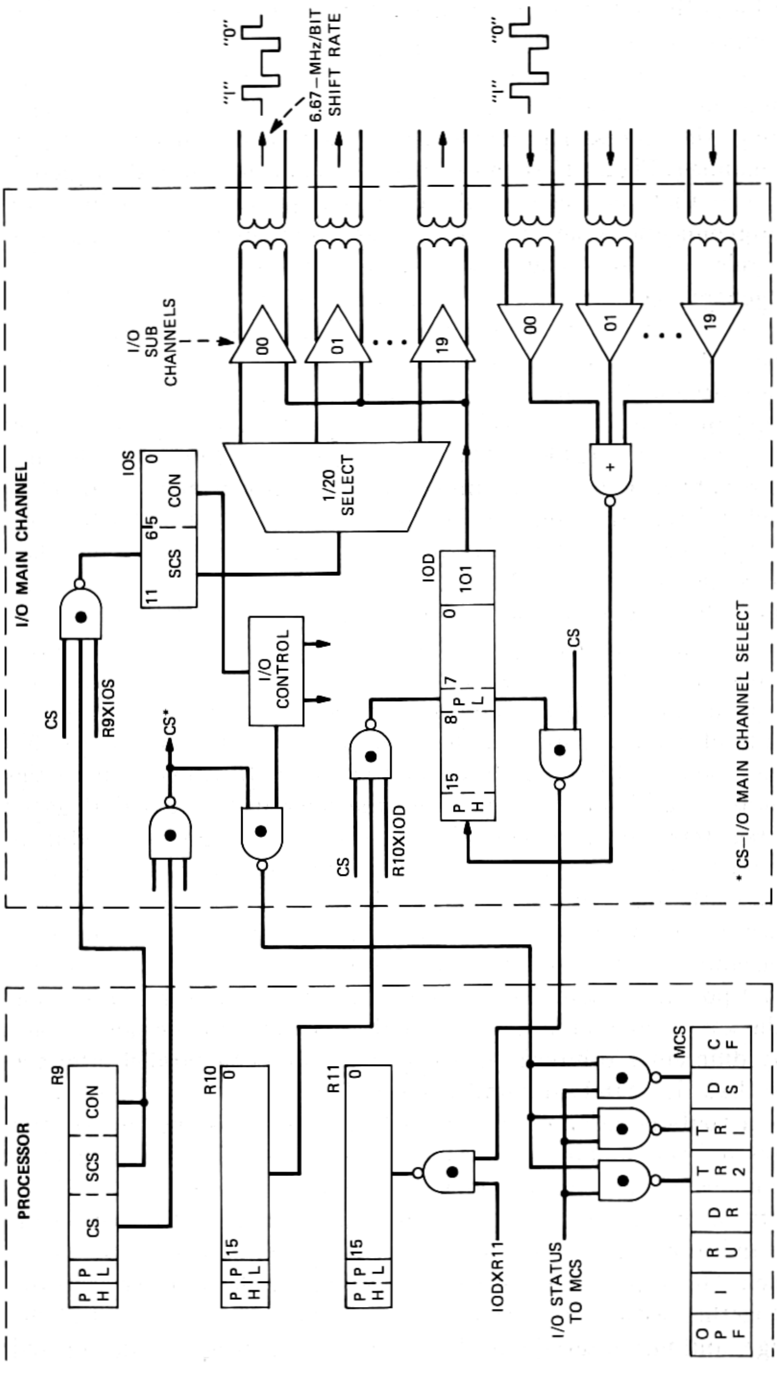


Fig. 17—I/O interface.

on the serial data link using a phase-encoded bit stream (6.67 MHz per bit) so that a separate clock signal is not required at the receiving end. Once the message has been shifted out of the 21-bit IOD, the I/O control continues to send a pulse stream out on the serial link. This pulse stream is used to provide the timing information for the peripheral unit and is required to send a response message back to the sending I/O channel. Therefore, as soon as the outgoing data are shifted out, the I/O control begins to monitor the incoming port. When a leading one is detected on the incoming message, the I/O control stops and sets a flag which the processor can interrogate. The processor can either return to executing other instructions or it can go into a loop, testing the completion flag just described. The microcontrol actually tests the state of the I/O channel by again using a microinstruction to gate the output of three states of the channel directly to the MCS register.

In the processor design, self-checking was achieved by partitioning the logic to obtain certain failure modes so that the check circuits could ensure fault detection. In instances where this partitioning became an unworkable solution, duplication was used, such as for the DML. For the implementation of the I/O channels, it is impractical to partition the logic to force single bit errors; however, it is also not economical to duplicate the channels. Three solutions were used to solve the detection problem. First, *m*-out-of-*n* codes were used in the control fields. Second, a loop-around test is performed on the data paths. Third, the parity check code carried with data within the processor is transmitted to and from the periphery as well. The *m*-out-of-*n* codes are checked using the same method as for the TO and FROM codes in the processor. The data check is performed by gating R10 to IOD and then returning the IOD to R11. Then a match is performed between R10 and R11 using the DML match hardware as previously outlined. This matching technique results in trading speed for hardware. Since the overhead for performing the match does not represent a significant real-time penalty, the choice is well justified. Parity is checked when data are received at the peripheral units and also when data are returned to the processor, as previously described.

APPENDIX B

MAR-RAR Matching

When the microprogram control was designed, an analysis was done to determine the type of faults that were most probable. Check schemes were then designed to detect these faults. A complete report of this work is given in Ref. 10. One of the areas of the microcontrol where it was difficult to use coding techniques to detect multiple faults was in the sequencing logic. That is, it would be disadvantageous to use anything other than a normal binary code to implement microstore addressing. As outlined in the reference, a check of the binary decoding of the address in the microstore address register (MAR), the access of the correct word in the microprogram store, and the proper readout are performed by using a simple parity check scheme and by interleaving binary-encoded words with m-out-of-n encoded words in the microprogram store.

However, to ensure that the proper address is loaded into the MAR, duplication is required for detection. The amount of hardware required to implement the duplication is minimized by time-sharing some circuitry. As a result, only the addition of an 11-bit matcher was required to perform the duplication-and-match function.

The hardware involved and the data flow are indicated in Fig. 18. As described in the section on microcontrol sequencing, when a new OP code is loaded from the SIR, it is gated into IB, MAR, and into RAR. The OP code is loaded into the IB because the operand fields X and Y are normally used directly by means of the translators attached to the outputs of the IB. The OP code is loaded into the MAR to access the first word of the microsequence for that OP code. The OP code is loaded into

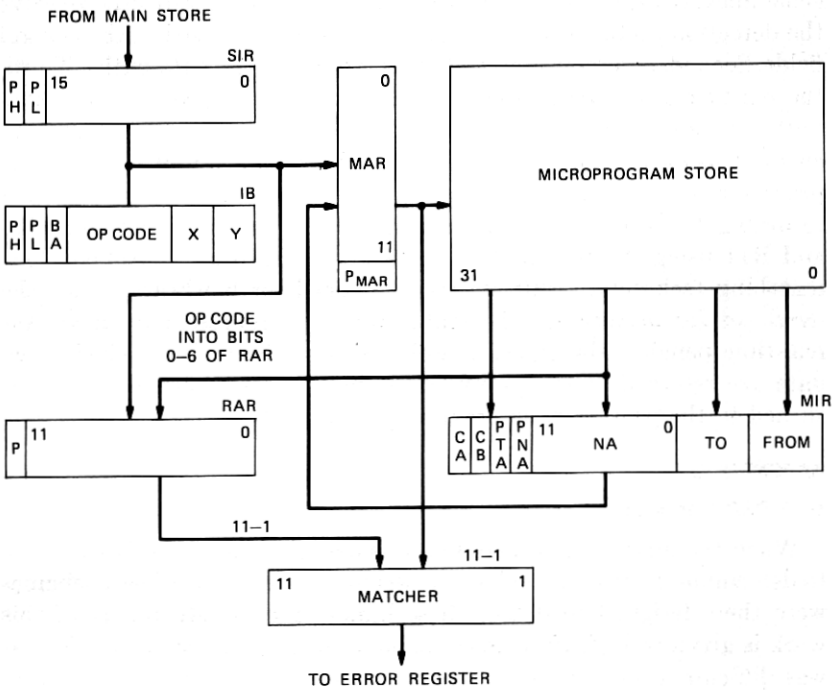


Fig. 18—MAR–RAR matching.

RAR to check that the OP code is correctly loaded into the MAR. This is achieved by the matcher being placed between the outputs of the MAR and RAR. The reason for duplication and matching is that it is not possible to predict the type of multiple fault that might exist when loading the MAR if it is not bit-sliced. The SIR and the IB are bit-sliced. Therefore, any multiple fault within a single bit-sliced circuit pack will be guaranteed to cause a parity failure if the data are indeed in error. For the MAR and MIR, it is not economical or, for that matter, practical to bit-slice them. Thus, the bit-slicing of the SIR and its parity check code together with the duplication and matching of the MAR provide a complete check on the loading of the new OP code into the MAR.

As each word is accessed and read out of the microstore, it is gated into the MIR. The NA field of the MIR is then gated into MAR. To check that this gating is correct, the NA field when gated out of the microstore is also gated into the RAR. As a result, a match can again be performed between the MAR and RAR to check for error-free operation. This same technique of loading the RAR with the same contents as the MAR is performed for indexing and loading the interrupt constant, the MRF constant, and the STOP constant.

Note that the match is performed only over bits 1 through 11 of the MAR and RAR. This simplifies implementation because of the number of operations that can change the state of bit 0. For example, conditional transfers alone have nine different ways of jamming bit 0 to a 1. The result is that the RAR is loaded exactly the same way for conditional transfers as for the normal sequencing case, and again the match is performed. If bit 0 is in error, the parity check on the sequencing will detect it.

The data and auxiliary control sequencing cases are slightly different. As described in Section 5.3, the next address for these commands is obtained by saving the previous contents of the MAR and jamming a 1 into bit 0. The fact that the address presently residing in the MAR was checked when first loaded into the MAR simplifies the design. The CA bit, which, when equal to 1, indicates either a data or auxiliary control is to be performed, is also used to inhibit the MAR-RAR matching for that cycle since the RAR has data in it for these two cases. Since data are loaded into the RAR as well as into the MIR (NA) field for a data command, a subroutine return can be easily implemented as described in Section 5.2. During the subroutine, the MAR-RAR matching is disabled. As a result, if a sequencing error occurred during a subroutine, it may go undetected for a few microcylces. However, as soon as the microcontrol exits the subroutine and returns to the update mode (RU = 1), the matching will again be enabled and a "stuck at fault" will quickly be detected.

GLOSSARY

AR	General-purpose buffer register in the DML logic
BA	Branch allowed
BIN	Block interrupts
BR	General-purpose buffer register in the DML logic
CA, CB	Control bits
CC	Central control
DB	Display buffer
DML	Data-manipulation logic
DR	Data ready
FN	Function register
GB	Gating bus
IB	Instruction buffer
I.D	Instruction or data
IM	Interrupt mask
I/O	Input/output
IOD	I/O data register
IOS	I/O status register
IS	Interrupt set
MAR	Microstore address register
MCH	Maintenance channel
MCS	Microcontrol status
MINT	Microinterpret
MIR	Microinstruction register
MMS	Main-memory state
MRF	Maintenance reset function
NA	Next address
NOP	Null microinstruction
PA	Program address
PBC	Processor's bus controller
PMAR	Parity for the MAR
PNA	Parity bit with next-address field
PTA	Parity of the accessed location
RAR	Return address register
REQ	Request
ROM	Read-only memory
R.W	Read or write
RU	RAR update
SAR	Store address register
SDR	Store data register
SIR	Store-instruction register

REFERENCES

1. M. V. Wilkes and J. B. Stringer, "Microprogramming and the Design of the Control Circuits in an Electronic Digital Computer," Proc. Cambridge Philosophical Soc., April 1953.
2. E. A. Irland and U. K. Stagg, "New Developments in Suburban and Rural ESS (No. 2 and No. 3 ESS)," International Switching Symposium Record, 1974.
3. J. A. Harr, F. F. Taylor, and W. Ulrich, "Organization of No. 1 ESS Central Processor," B.S.T.J., *43*, No. 5 (September 1964), pp. 1845–1922.
4. T. E. Browne, T. M. Quinn, W. N. Toy, and J. E. Yates, "No. 2 ESS Control Unit System," B.S.T.J., *48*, No. 8 (October 1969), pp. 2619–2668.
5. R. W. Downing, J. S. Nowak, and L. S. Tuomenoksa, "No. 1 ESS Maintenance Plan," B.S.T.J., *43*, No. 5 (September 1964), pp. 1961–2019.
6. H. J. Beuscher, G. E. Fessler, D. W. Huffman, P. J. Kennedy, and E. Nussbaum, "Administration and Maintenance Plan," B.S.T.J., *48*, No. 8 (October 1969), pp. 2765–2815.
7. R. W. Cook, W. H. Sisson, T. F. Storey, and W. N. Toy, "Maintenance Design of a Control Processor for Electronic Switching Systems," Proceedings of the Third Texas Conference on Computing Systems, November 1974.
8. R. E. Staehler, "1A Processor—A High Speed Processor for Switching Applications," International Switching Symposium Record, 1972.
9. D. A. Anderson, "Design of Self-Checking Digital Networks Using Code Techniques," CSL Report R527, University of Illinois, Ph.D. Thesis, October 1971.
10. R. W. Cook, W. H. Sisson, T. F. Storey, and W. N. Toy, "Design of a Microprogram Control for Self-Checking," Digest of 1972 Fault-Tolerance Computing, June 1972.